Westfälische
Wilhelms-Universität
Münster

› Optimierung einer 3D-Umgebungsmodellierung mittels Octree am Beispiel von Laserdaten

Michael Scholz

Münster 2012

wissen.leben
WWU Münster

Institut für
Geoinformatik

Optimierung einer 3D-Umgebungsmodellierung mittels Octree am Beispiel von Laserdaten

Westfälische
Wilhelms-Universität
Münster

Diplomarbeit

am Institut für Geoinformatik

des Fachbereichs Geowissenschaften

der Westfälischen Wilhelms-Universität Münster

vorgelegt von

Michael Scholz

im August 2012

Erster Gutachter: Prof. Dr. Edzer Pebesma

Zweiter Gutachter: Stefan Krause

Kurzfassung/Abstract

Für die Kollisionsvermeidung bei autonomer Navigation eines UAV in urbanem Gelände ist ein hinreichend genaues Umgebungsmodell von entscheidender Bedeutung. Die Leistung entfernungsmessender Sensoren nimmt stetig zu, so muss das Umgebungsmodell die Sensordaten weiterhin performant verarbeiten können, um ein rechtzeitiges Reagieren zu ermöglichen. Diese Arbeit optimiert ein bestehendes Verfahren zur 3D-Umgebungsmodellierung, welches Array-basierte Belegtheitsgitter zur Sensordatenfusion mit einer Polygonrepräsentation von Hindernissen kombiniert. Zur Optimierung wird das probabilistische Belegtheitsgitter durch einen Octree implementiert, der effektiv die Dimension eines räumlichen Problems reduzieren soll. Simulationsergebnisse basierend auf realen Messdaten zeigen einen Leistungszuwachs und gesenkten Speicherbedarf bei der Modellierung.

For obstacle avoidance during autonomous navigation of UAVs in urban area the used world model plays an important role. With increasing performance of distance sensors, also high-performance data processing has to be guaranteed in order to react on obstacles in time. This thesis optimises an existing technique for 3D world modelling, which combines array-based occupancy grids for sensor data fusion with polygonal obstacle representation. For optimisation, this probabilistic grid is implemented with an octree which is known to effectively reduce dimensionality of spatial problems. Results of real data simulations show increasing performance and reduced memory consumption during the modelling process.

Danksagung

In erster Linie möchte ich meinem Betreuer Stefan Krause danken, der mich stets inspirierte und bei all meinen Problemen hilfreich zur Seite stand. Ebenfalls danke ich Franz Andert für seine Unterstützung bei der Einarbeitung zu Beginn meines Aufenthalts beim DLR. Über den gesamten Zeitraum meiner Diplomarbeit war ich motiviert und hatte immerzu das Gefühl etwas zu bewegen.

Vielen Dank an Herrn Pebesma für die Freiheit bei der Themenwahl und seine Geduld. Weiterhin danke ich Hendrik und Max für den interdisziplinären Gedankenaustausch und kann Verhoeven, Kono und Sven wärmstens als Korrekturleser empfehlen.

Ganz besonders danke ich meinen Eltern dafür, dass es uns alle gibt und sie mir dieses tolle Studium ermöglicht haben.

Keep on smiling.

Inhalt

Abbildungen

Abbildungen

Tabellen

Abkürzungen

ARTIS Autonomous Rotorcraft Testbed for Intelligent Systems (Forschungsprogramm des DLR)

BSD Berkeley Software Distribution (Unix-Version unter Open-Source-Lizenz)

DIP Digital Image Processing (digitale Bildverarbeitung)

DLR Deutsches Zentrum für Luft- und Raumfahrt e.V.

GNSS Global Navigation Satellite System (globales Navigationssatellitensystem)

GPS Global Positioning System (GNSS zur Positionsbestimmung und Zeitmessung)

IMU Inertial Measurement Unit (Inertialmesssystem)

LAA Large Address Aware (Compileroption für erweiterten Adressraum)

LUT Lookup-Tabelle (speichert statische Informationen)

NED North, East, Down (Koordinatensystemkonvention)

UAV Unmanned Aerial Vehicle (unbemanntes Luftfahrzeug)

Symbole

Gitter, Zonen

M	Belegtheitsgitter
m, m_i	Zelle im Belegtheitsgitter, Zelle an i-ter Stelle
`A`, `B1`, `5`	Knotenschlüssel im Baum
$\mathtt{Z}_{25}$	lokale Zone der Dimension 64 m × 64 m × 25 m
$\mathtt{Z}_{64}$	lokale Zone der Dimension 64 m × 64 m × 64 m

Indizes

f	Index des trägerfesten Koordinatensystems
g	Index des Gitterkoordinatensystems
s	Index des Sensorkoordinatensystems
w	Index des Weltkoordinatensystems

Landau-Symbole

$O(\dots)$	asymptotische obere Schranke für Komplexität
$\Theta(\dots)$	asymptotische scharfe Schranke für Komplexität (gleichbleibend)

Operatoren

$\lfloor \dots \rfloor$	Abrunden
$\dots^{-1}$	Invertieren einer quadratischen Matrix
$\dots^{\top}$	Transponieren eines Vektors, einer Matrix

Skalare

c	metrische Raster-/Gitterauflösung, Zellgröße
d	Tiefe im Baum
h	Baumhöhe

Symbole

l	Seitenlänge einer Region/Zone
N	maximale Kinderzahl eines Knotens im N-Baum
n	Anzahl von Elementen oder Knoten (auch als Problemgröße)
o	Belegtheitszustand einer Zelle
L	Logit-Wert (Belegtheitswert einer Zelle)
P	Wahrscheinlichkeit (der Belegung einer Zelle)
φ, ϑ	Azimut- und Polarwinkel polarer Sensorkoordinaten
Ψ, Θ, Φ	Gier-, Nick-, Roll-Winkel zur Beschreibung von Rotationen
r, r_{ab}	gemessene Distanz, bzgl. Koordinatensystem ab (siehe Indizes)
t	Zeitpunkt eines Ereignisses
x_a, y_a, z_a	Koordinaten im Koordinatensystem a (siehe Indizes)

Vektoren, Rotationen

$\mathbf{p}_a$, $\mathbf{s}_a$	Punkte in Koordinatensystem a (siehe Indizes)
$\mathbf{t}_{ba}$	Translationsvektor von Koordinatensystem a nach b (Ursprung von b in a, siehe Indizes)
$\mathbf{R}_{ba}$	Rotation von Koordinatensystem a nach b (siehe Indizes)

1 Einleitung

Diese Diplomarbeit wurde im Rahmen eines Forschungsprogramms am Deutschen Zentrum für Luft- und Raumfahrt e. V. (DLR) am Standort Braunschweig angefertigt. Die Abteilung für unbemannte Luftfahrzeuge des Instituts für Flugsystemtechnik des DLR entwickelt Technologien für zukünftige unbemannte Luftfahrzeuge und hat großes Interesse an der Umweltwahrnehmung für Navigationszwecke in unbekanntem Gelände. Schwerpunkte sind unter anderem die Hinderniserkennung und Kollisionsvermeidung.

Im Rahmen dieser Arbeit wird ein bestehendes Weltmodell zur dreidimensionalen Umgebungskartierung erweitert und optimiert. Das Modell kommt bei der kollisionsfreien Navigation eines unbemannten Kleinhubschraubers zum Einsatz und wurde primär zur Kartierung von Stereokameradaten entwickelt. Am Beispiel von Daten eines Laserscanners wird diese Einschränkung aufgehoben und ein sensorunabhängiges Modell vorgestellt. In Zukunft wird beabsichtigt, beide Sensortypen im Verbund zu benutzen, um ihre Vorteile zu kombinieren.

Der Schwerpunkt liegt auf einer Leistungssteigerung des 3D-Umgebungsmodells durch Verwendung einer alternativen Datenstruktur zur Verarbeitung von Sensordaten. Im zugrundeliegenden Modell werden einfache Arrays verwendet, welche im dreidimensionalen Fall große Speichernachteile besitzen und darauf arbeitende Algorithmen in ihrer Leistung beschränken. Octrees zur hierarchischen Raumgliederung versprechen Vorteile in beiden Punkten. So soll die Komplexität von Algorithmen auf Octrees proportional zur Oberfläche des Volumens sein, statt proportional zum Volumen selbst, wie es bei Verwendung von Arrays der Fall ist (Samet 2006, Kapitel 2.2.2.2). Das Array-basierte Weltmodell ist für die Anwendung von Stereokameradaten bereits echtzeitfähig, allerdings stellen aktuelle und zukünftige Sensoren durch größere Messentfernungen, Öffnungswinkel und höhere Datenmengen neue Herausforderungen dar, die es mit dem optimierten Octree-Modell zu meistern gilt.

Im Zuge der Umstellung auf Octrees wird eine freie Programmbibliothek benutzt und um erforderliche Funktionen erweitert, damit Octrees auch für räumlich unbeschränkte Szenarien genutzt werden können. Alle nötigen Erweiterungen werden im Verlauf dieser Arbeit vorgestellt und wurden bereits teilweise dem Projekt beigesteuert. Die Erweiterung des Array-Modells auf Verarbeitung von Laserdaten offenbart einige Probleme, die zwar von der zugrundeliegenden Datenstruktur unabhängig sind, aber ebenfalls mit Hilfe des neuen Octree-Modells umgangen werden können.

1.1 Anforderungen an das Umgebungsmodell

Ein Wechsel der Datenstruktur zur Sensordatenintegration kann neben den Vorteilen auch Einschränkungen mit sich bringen. Da ein bestehendes Konzept modifiziert wird, muss in erster Linie sichergestellt werden, dass dessen eigene Anforderungen beibehalten werden. Dabei wird die Anwendbarkeit des Konzepts auf eine statische Umgebung, also ohne dynamische Objekte, beschränkt. Im Kontext dieser Arbeit soll das 3D-Umgebungsmodell

- unabhängig vom verwendeten Sensor sein;
- mit jeder Sensormessung aktualisiert werden, damit rechtzeitig auf erkannte Hindernisse reagiert werden kann;
- räumlich erweiterbar sein, ohne sich auf ein festes Gebiet zu beschränken;
- weiterhin echtzeitfähig bleiben, wobei das zugrundeliegende Array-basierte Modell bereits echtzeitfähig ist und durch die Optimierung eine Leistungssteigerung erwartet wird.

Dieses Umgebungsmodell kommt bei der autonomen Navigation von Kleinhubschraubern des ARTIS-Programms zum Einsatz.

1.2 ARTIS-Programm

Autonomous Rotorcraft Testbed for Intelligent Systems (ARTIS) ist ein 2003 gestartetes Programm zur Bereitstellung einer Testumgebung für Forschungszwecke an unbemannten

Abbildung 1.1: midiARTIS mit frontal montiertem Laserscanner.

Luftfahrzeugen (UAVs). Es verbindet Modellierung und Simulation (Hardware-/Software-in-the-Loop) mit Systemidentifikation und umfasst mehrere Trägersysteme unterschiedlicher Größen[1]. Durch Kombination umgebungswahrnehmender Sensoren mit angewandter Bildverarbeitung wird zum Beispiel die on-board-Hinderniserkennung zur kollisionsfreien Navigation in unbekanntem Gelände realisiert.

Der für diese Arbeit verwendete Laserscanner kommt am Kleinhubschrauber midiARTIS zum Einsatz. Abbildung 1.1 zeigt den Sensorträger mit an der Front montiertem Laserscanner. Die technischen Merkmale des Hubschraubers sind in Tabelle 1.1 aufgeführt. Von den 6 kg der maximalen Zuladung werden etwa 4 kg von der Avionik[2] einschließlich des Treibstoffs für den Verbrennungsmotor und der Stromversorgung eingenommen. Die übrigen 2 kg können für Experimentalsysteme wie Laserscanner und Stereokamera

Tabelle 1.1: Technische Merkmale des midiARTIS.

Rotordurchmesser	1,9 m
Motorleistung	2 PS
Max. Flugzeit	20 min
Leergewicht	6 kg
Max. Zuladung	6 kg

[1] Eine Beschreibung weiterer Trägersysteme ist unter `http://www.dlr.de/ft/artis` zu finden.
[2] Umschließt die Elektronik und Instrumente zur Flugsteuerung, Kommunikation und Datenverarbeitung.

mit zugehörigem Bildverarbeitungsrechner verwendet werden. Je nach Abfluggewicht und Energievorrat ist eine Flugzeit von bis zu 20 Minuten möglich (Thieleke u. a. 2004).

Der eingesetzte Bildverarbeitungsrechner für die Prozessierung der Daten des Laserscanners ist ein Intel Core 2 Duo mit 1,5 GHz. Er ist mit dem Bordrechner vernetzt und kommuniziert mit ihm zur Weiterverarbeitung der Daten, beispielsweise zur Pfadplanung. Generell regelt der Bordrechner die Ausführung von Flugbefehlen in Abhängigkeit verschiedener Sensordaten. Für die Kommunikation mit der Bodenstation steht eine WLAN- und bei größerer Entfernung eine Funkmodem-Verbindung zur Verfügung. Es ist hervorzuheben, dass jegliche für die autonome Navigation erforderliche Datenauswertung an Bord des Hubschraubers geschieht.

1.3 DIP-Framework

Die Implementierung der vorgestellten Verfahren erfolgt als Erweiterung des beim DLR entwickelten DIP-Frameworks (Digital Image Processing). Das DIP-Framework stellt verschiedene Algorithmen zur Bildverarbeitung bereit und dient ebenfalls zur Verarbeitung der als Bildzeilen kodierten Messungen des Laserscanners. Es kommt als „on-board-DIP“ direkt auf den Kleinhubschraubern zum Einsatz, sowie als „on-ground-DIP“ in einer entsprechenden Simulationsumgebung. Es können aufgezeichnete Daten aus vorherigen Flugeinsätzen zum Testen von Bildverarbeitungsalgorithmen in der Simulationsumgebung verwendet werden. So werden neue Verfahren anhand von Realdaten „on-ground“ entwickelt und getestet, bevor sie „on-board“ zum Einsatz kommen.

Bildverarbeitungsfilter im DIP-Framework verarbeiten eingehende Datenströme, wie Rohdaten verschiedener Sensoren, um daraus weitere Informationen abzuleiten. Ein Filter implementiert zum Beispiel entsprechende Algorithmen zur Erstellung eines 3D-Umgebungsmodells aus Messdaten eines Laserscanners. Es können mehrere solcher Filter für verschiedene Anwendungszwecke kombiniert werden.

1.4 Themenverwandte Arbeiten

In dieser Arbeit wird das 3D-Umgebungsmodell von Andert (2011) angepasst und optimiert. Es benutzt verschiedene Kartentypen zur Abbildung der Umgebung aus Sensormess-

daten und anschließenden Extraktion von Hindernissen, die zur Kollisionsvermeidung benötigt werden. Dabei werden Sensordaten wie oft üblich in ein *metrisches Gitter* übertragen, welches den Raum mit fester Zellgröße diskretisiert. Die Hindernisinformationen werden losgelöst davon gespeichert. Das metrische Gitter wird mittels *Array*-Datenstrukturen realisiert.

Auch Scherer u. a. (2008) benutzen Arrays zur Realisierung des metrischen Gitters und erreichen damit bemerkenswerte Ergebnisse. Der leistungsfähige Laserscanner detektiert auch dünne Stromleitungen aus 100–150 Metern Entfernung. Sie kombinieren eine globale Pfadplanung mit reaktivem Ausweichen und arbeiten direkt auf dem 3D-Gitter, ohne Hindernisse gesondert zu extrahieren. Bei einer hohen Gitterauflösung von einem Meter reicht das Verfahren bereits zur autonomen Echtzeitnavigation aus.

Hrabar (2012) gibt einen anschaulichen Überblick über Laser- und Stereokamera-Systeme und vergleicht diese hinsichtlich Eignung der Kollisionsvermeidung im Flug. Er erreicht mit dem Laserscanner wesentlich bessere Ergebnisse als nur mit der Stereokamera, kombiniert beide Sensoren allerdings auch in *einem* Gitter mit 0,5 Metern Auflösung. Die relativ feine Auflösung des Gitters stellt hohe Anforderungen an das System, da die Datenmenge durch parallelen Einsatz zweier Sensoren erhöht wird. Die Leistung ist ausreichend und das System liefert ebenfalls gute Ergebnisse.

Das Projekt OctoMap (Wurm, Hornung u. a. 2010) realisiert das erwähnte Gitter durch eine hierarchische Baumstruktur: durch einen *Octree*. Solche Strukturen können Speichervorteile bringen und sind flexibler, da ihre Größe nicht im Vorfeld festgelegt werden muss. Der Octree aus OctoMap findet im Rahmen dieser Diplomarbeit zur Optimierung des Array-Modells Verwendung.

Fournier u. a. (2007) nutzen den Octree zur inkrementellen Kartierung aus Laser- und Stereokameradaten. Wie beim midiARTIS ist GPS-Empfang vorhanden und folglich wird eine hinreichend genaue Navigationslösung vorausgesetzt. Es wird ebenfalls von unbekanntem Gelände ausgegangen und das Umgebungsmodell schrittweise aus Sensordaten für ein *Bodenfahrzeug* erstellt. Fournier u. a. benötigen in Bodennähe kein vollständiges 3D-Modell und führen ihre Kollisionsprüfung gegen ein abgeleitetes 2,5D-Modell[3] aus. Trotzdem bleibt das 3D-Gitter persistent für weitere Datenintegration.

[3] Im Gegensatz zu 3D beschreibt 2,5D eine Grundfläche mit maximal *einer* Höheninformation pro Messpunkt, wodurch z. B. Überhänge oder Brücken nicht modelliert werden können.

Bry u. a. (2012) zeigen überzeugend wie ein autonomes Kleinflugzeug in geschlossenen Räumlichkeiten Hindernissen ausweicht und benutzen als Grundlage ebenfalls einen Octree. Da in geschlossenen Räumen gewöhnlich kein GPS-Signal vorhanden ist, liegt der Fokus auf einer Selbstlokalisierung aus den gemessenen Daten, und nicht auf der Kartierung. Dabei wird die Karte des Raums mit den Hindernissen *im Vorfeld* erzeugt und als Octree bereitgestellt. Zur Laufzeit wird dann die Lage der gemessenen Punkte in dem bereitgestellten Umgebungsmodell bestimmt und auf dieser Grundlage, kombiniert mit einem Inertialsensor (IMU), die aktuelle Flugzeugposition im Raum ermittelt.

Diese relativ aktuellen Projekte benutzen zur Abbildung von Sensordaten metrische Gitter, deren Verwendung zu Kartierungszwecken in der Robotik weit verbreitet und auf Moravec (1988) und Elfes (1989) zurückzuführen ist. Zwar werden vereinzelt Hindernisinformationen gesondert extrahiert (Fournier u. a. 2007), die Verbindung eines Octree-Gitters mit Extraktion von 3D-Hindernissen stellt allerdings einen neuen Ansatz dar.

1.5 Struktur dieser Arbeit

Nach diesem Einführungskapitel widmet sich Kapitel 2 den Grundlagen zum verwendeten Laserscanner und dessen Koordinatenherleitung. Alle relevanten Koordinatensysteme mit den nötigen Transformationen werden eingeführt und Datenstrukturen zur Abbildung der angesprochenen Gitter vorgestellt. Grundlagen zur Wahrscheinlichkeitsrechnung erleichtern das spätere Verständnis von Operationen im Gitter.

Kapitel 3 „Umgebungsmodellierung“ behandelt detailliert die Übertragung von Sensormessungen ins räumliche Gitter, mit dem eine inkrementelle Kartierung ermöglicht wird. Anschließend wird das für diese Arbeit zentrale Weltmodell von Andert (2011) eingeführt und mit der Octree-Struktur optimiert.

Kapitel 4 „Analyse beider Weltmodelle“ vergleicht beide Weltmodelle hinsichtlich Laufzeit und Speicher anhand von Simulationsdaten und diskutiert die Ergebnisse kurz. Abschließend werden in Kapitel 5 „Schlussfolgerungen und Ausblick“ Schlussfolgerungen aus den Resultaten gezogen und ein Ausblick gegeben.

2 Grundlagen

2.1 Sensoren zur Umgebungskartierung

Zum Erstellen eines Modells der Umgebung müssen ihre Merkmale erkannt und entsprechend festgehalten werden. Dafür wird auf ein System bestehend aus einem Sensor oder aus einem Verbund von Sensoren zurückgegriffen. Für die Merkmalserkennung und speziell für kartografische Zwecke eignen sich Sensoren zur Entfernungsmessung (Borenstein u. a. 1996, Kapitel 4). Diese können nach Albertz (2009, Kapitel 2) in passive und aktive Systeme klassifiziert werden:

- Passive Sensoren messen zur Datenaufnahme bereits vorhandene Energie, wie eine Digitalkamera die Reflexion elektromagnetischer Strahlung. Auf diese Weise können Entfernungsinformationen aus Stereobildern einer Stereokamera abgeleitet werden (siehe Kapitel 2.1.1).

- Aktive Sensoren erzeugen die Energie für das zu messende Phänomen selbst. Diese arbeiten meist nach dem Prinzip der Laufzeitmessung von Schallwellen (Ultraschallsensoren) oder elektromagnetischen Wellen unterschiedlicher Wellenlängen (Radar oder Lasermessgeräte, Kapitel 2.1.2).

Je nach Anwendung hängt die Wahl des Sensors unter anderem von dem Detailgrad seiner Messung ab. Als wichtigste Eigenschaften sind die *Genauigkeit*, *Auflösung* und *Frequenz* der Datenakquise zu nennen. Genauigkeit beschreibt die Güte einer Entfernungsmessung bezogen auf die tatsächliche Entfernung. Unter Auflösung versteht man, wie detailliert solch eine Messung durchführbar ist, wie klein also der minimale Unterschied zwischen Messgrößen sein kann. Für den Echtzeiteinsatz ist stets eine hohe Frequenz anzustreben, um rechtzeitig auf Änderungen in der Umgebung reagieren zu können.

Einfach aufgebaute Ultraschallsensoren können zur Detektion von Hindernissen und zur groben Entfernungsmessung bereits genügen. Sie haben ihren Einsatz zum Beispiel im Automobilbereich bei Abstandsmessungen zwischen Fahrzeugen oder bei Systemen zur elektronischen Einparkhilfe. Ein unbemanntes Luftfahrzeug kann während eines autonomen Landemanövers zusätzlich zur aktuellen GNSS-Höhe den Abstand zum Boden mit Hilfe eines Ultraschall- oder Radarsensors bestimmen. Ultraschallsensoren sind klein und günstig, schränken jedoch das Einsatzspektrum für kartographische Zwecke durch zugrunde liegende physikalische Grenzen ein. Beispielsweise breiten sich Schallwellen im Gegensatz zu elektromagnetischen Wellen im Medium Luft langsamer aus. Ultraschallsensoren haben meist eine niedrigere Auflösung und unterliegen Streueffekten stärker (Leonard u. a. 1992).

Sollen detaillierte Informationen aus der Umgebung gezogen werden, bieten passive und aktive Sensoren zur Messung *elektromagnetischer Strahlung* vorzuziehende Eigenschaften. Aufgrund sinkender Kosten, kompakterer Bauweisen und zugleich leistungsfähigerer Bildsensoren konnten sich vor allem kamerabasierte Systeme in den letzten Jahren in der Robotik behaupten.

2.1.1 Stereokamera

Kameras als passive bildgebende Systeme werden seit vielen Jahren zur Ableitung von Entfernungsinformationen in verschiedenen Disziplinen der Wissenschaft genutzt. Die Fernerkundung setzt auf fotografische Systeme, um mit Hilfe des stereoskopischen Effekts aus Luft- und Satellitenbildern Höheninformationen des abgebildeten Geländes abzuleiten. Dabei werden sich überschneidende Aufnahmen einer Bilderreihe paarweise verknüpft. Jede Aufnahme nimmt beim Überflug das Gelände aus einem leicht anderen Winkel auf. Diese Bildpaare werden zur stereoskopischen Betrachtung herangezogen und ermöglichen, einen räumlichen Eindruck des Geländes zu erlangen.

Für den midiARTIS wird zur Navigation im Gelände ein vollwertiges 3D-Modell der Umgebung benötigt, um auch Überhänge, Brücken oder Vegetation für die Kollisionsvermeidung modellieren zu können. Man greift zu einem Stereokamerasystem bestehend aus zwei kalibrierten Kameras, mit denen die Aufnahme eines Bildpaares *zur selben Zeit* ermöglicht wird. Das System liefert in hoher Frequenz (z. B. 30 Hz) Disparitätsbilder, mit deren

Hilfe Entfernungsangaben interpoliert werden. Die Erzeugung von Tiefenbildern mittels Stereokamera wird von Andert (2011, Kapitel 2.5) detalliert beschrieben.

Theoretisch liegt ein großer Nachteil dieser *indirekten* Entfernungsmessung in der quadratisch abnehmenden Tiefenauflösung. In der Praxis ist eine noch stärkere Abnahme der Auflösung mit wachsender Entfernung zu beobachten (Hrabar 2012). Ein Laserscanner unterliegt dem Effekt der entfernungsabhängigen Tiefenauflösung wesentlich weniger. Eine Stereokamera ist weiterhin auf hohe Kontraste angewiesen, also auf eine ausreichende Ausleuchtung der Umgebung, um Übereinstimmungen innerhalb der Bilder zur Tiefenwertberechnung ermitteln zu können.

2.1.2 Lasermessung

Entfernungsmessung mittels Laserstrahlen, oft auch als *Lidar* (Light detection and ranging) bezeichnet, ist ein aktives Verfahren zur *direkten* Distanzmessung. Aus der Laufzeit des Lichts können ohne Interpolation direkt Rückschlüsse auf die Distanz gezogen werden, da Laufzeit und Distanz sich proportional zueinander verhalten. Es handelt sich in erster Linie nicht um ein bildgebendes System, denn erst durch Nachbearbeitung der resultierenden Daten ist eine bildhafte Darstellungen der Punktwolke möglich. Allerdings zeichnen *abbildende Systeme* zusätzlich die Intensität der reflektierten Wellenlänge auf, wodurch eine direkte grafische Darstellung mit Grauwerten ermöglicht wird (Albertz 2009, Kapitel 2).

Häufig sind Lidar-Systeme als Laserscanner umgesetzt. Es ist wichtig zu unterscheiden, dass im Gegensatz zu einer Kamera, deren gesamte Messpunkte in einem Sichtvolumen *gleichzeitig* aufgezeichnet werden, der Laserscanner Punkte in seiner Scan-Ebene *sequentiell* generiert. Daher die Bezeichnung *Scanner*, da die Gesamtmessung aus vielen Messungen einzelner Laserimpulse resultiert. Es gibt verschiedene technische Umsetzungen, um den Laserstrahl eines Scanners in die gewünschte Richtung zu lenken. Die meisten Systeme, wie auch der für diese Arbeit verwendete Scanner UTM-30LX des Herstellers Hokuyo, basieren auf einem optisch-mechanischen Prinzip, bei dem der Lichtstrahl mit Hilfe eines Spiegels abgelenkt wird. Der Spiegel wird von einem Motor in festen Winkelabständen um eine fixe Achse rotiert und definiert so die Winkelauflösung des Scanners. Dabei dient ein Encoder als „Taktgeber“ und übernimmt die korrekte Ansteuerung des Motors (siehe Abbildung 2.1, S. 10). Die mechanische Funktionsweise kann zu Problemen führen, wenn

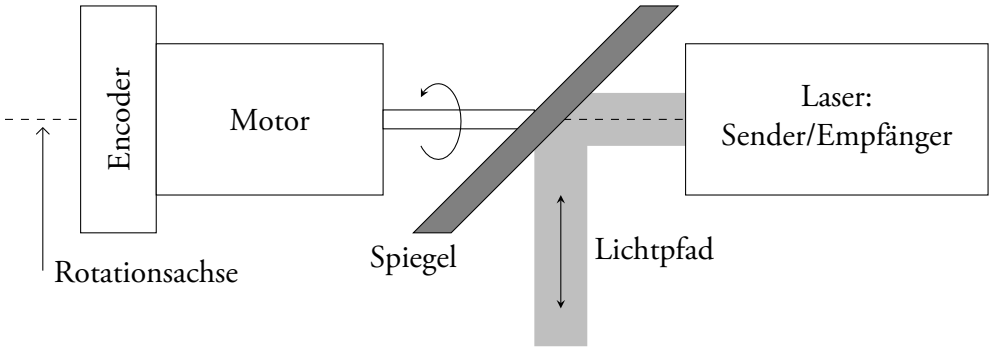


Abbildung 2.1: Schematischer Aufbau eines Laserscanners (nach Miller u. a. 1998). Ein Spiegel lenkt den Lichtstrahl, Auswertung im selben System.

ein Scanner starken Vibrationen eines Trägersystems ausgesetzt wird. Wie erwähnt sind aktive Systeme nicht auf eine ausreichende Ausleuchtung der Umgebung angewiesen. Dafür können Schwierigkeiten bei glatten Oberflächen auftreten, bei denen aufgrund geringer Streuung kein Licht zurück zum Sensor reflektiert wird. In diesem Fall ergibt die Kombination mit einem bildgebenden System wie einer Stereokamera Sinn, da diese ergänzende Informationen liefern kann. Tabelle 2.1 gibt einen kompakten Vergleich der Stereokamera- und Laser-Systeme, wie sie zur Umgebungswahrnehmung sowohl bei Boden- als auch bei Luftfahrzeugen eingesetzt werden.

Tabelle 2.1: Stereokamera und Laserscanner im Vergleich.

	Stereokamera	Laserscanner
Arbeitsweise	passiv	aktiv
Entfernungsmessung	interpoliert	direkt
Datenaufnahme	3D (Sichtvolumen)	2D (Ebene)
Leistungsaufnahme	geringer	höher

Hokuyo UTM-30LX

Im Rahmen dieser Arbeit wird mit Daten eines Hokuyo UTM-30LX Laserscanners gearbeitet (Abbildung 2.2), allerdings wird die grundlegende Integration dieses Laserscanners ins Bildverarbeitungssystem nicht behandelt. Diese Vorarbeit wurde bereits an anderer Stelle geleistet und eine korrekte Funktionsweise wird vorausgesetzt. Tabelle 2.2 auf Seite 12

Abbildung 2.2: Laserscanner UTM-30LX (Hokuyo Automatic Co. 2009).

fasst die wichtigsten technischen Eigenschaften[1] des Scanners bei Idealbedingungen zusammen. Der UTM-30LX zeigt kompakte Abmessungen und eine für ein aktives System akzeptable Leistungsaufnahme. Daher eignet sich dieses Modell gut für den Einsatz an Hubschraubern wie dem midiARTIS oder sogar kleineren Trägern. Zusätzlich zur Entfernungsmessung mit 40 Hz kann das Gerät die Intensität der gemessenen Reflexion aufzeichnen, was die Messfrequenz jedoch auf 20 Hz reduziert (Hrabar 2012). Die Intensitätswerte werden im Rahmen dieser Arbeit nicht weiter verwendet, allerdings geht Kapitel 5 auf eine mögliche Anwendung ein.

2.1.3 Weitere Sensoren

Weiterentwicklungen bringen neue Sensortypen auf den Markt, mit denen unter anderem versucht wird, die Vorteile beider oben beschriebenen Systeme zu kombinieren. Time-of-flight-Kameras oder spezielle 3D-Flash-Lidar funktionieren nach dem Kameraprinzip, indem die Reflexion eines gestreuten Laserimpulses in einem *Sichtvolumen* aufgenommen wird. Dabei wird für jeden Pixel des Sensors die Laufzeit des ausgesendeten und reflektierten Laserimpulses berechnet, was eine *direkte* Distanzmessung ermöglicht. Weiterhin ist laut Advanced Scientific Concepts Inc. (2012) die Aufnahme von Entfernungs- und Intensitätsdaten gleichzeitig möglich. Aus den Sensordaten können mit einer Frequenz von bis

[1] Für die vollständige Systemspezifikation siehe Hokuyo Automatic Co. 2009.

Tabelle 2.2: Technische Merkmale des Laserscanners UTM-30LX.

Wellenlänge	905 nm
Reichweite	0,1 m bis 30 m
Frequenz	40 Hz
Öffnungswinkel	270°
Winkelauflösung	0,25°
Tiefenauflösung	1 mm
Standardfehler	< 10 mm bei 0,1 m bis 10 m < 30 mm bei 10 m bis 30 m
anliegende Spannung	12 V±10 %
anliegende Stromstärke	0,7 A bis 1 A
Leistungsaufnahme	< 8 W
Gewicht ohne Kabel	210 g
Abmessungen	60 mm × 60 mm × 85 mm

zu 100 Hz 3D-Punktwolken zur Weiterverwendung generiert werden. Da diese Systeme, genau wie Kameras, ohne komplexe mechanische Bauteile auskommen, sind sie im Vergleich zu zeilenbasierten Laserscannern äußeren Einflüssen gegenüber robuster. Allerdings ist die Verfügbarkeit solcher Systeme bisher gering und durch die momentan noch geringe Pixeldichte gegenüber aktuellen Digitalkameras ergeben sich Nachteile.

2.2 Geometrische Grundlagen

Das Ziel der einzelnen Entfernungsmessungen ist die Erstellung einer global gültigen Hinderniskarte der Umgebung des Sensorträgers. Dafür ist es wichtig, den genauen Messvorgang sowie die Orientierung des Sensors bezüglich seiner Umgebung zu kennen. Bei den im Folgenden vorgestellten Koordinatensystemen handelt es sich um *rechtshändige* kartesische 3D-Koordinatensysteme die als Basis der Umgebungsmodellierung dienen. Abbildung 2.3 verdeutlicht den Zusammenhang der Koordinatensysteme untereinander.

Es wird unterschieden zwischen innerer und äußerer Orientierung (Albertz 2009, Kapitel 2), wobei *innere Orientierung* die Eigenschaften des Messvorgangs und die Verortung der Messpunkte im *Sensorkoordinatensystem* (Index *s*) bezeichnet. Wie in der Computergrafik üblich, wird die „Blickrichtung“ des Sensors entlang der *z*-Achse des Sensorkoordi-

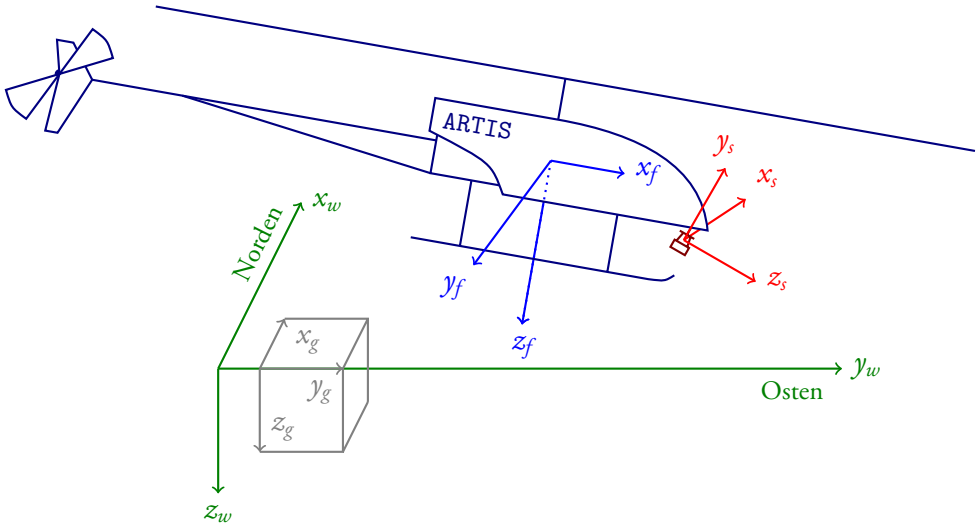


Abbildung 2.3: Verwendete Koordinatensysteme (nach Andert 2011).

natensystems ausgerichtet. Die detaillierte Herleitung der kartesischen Sensorkoordinaten erfolgt im folgenden Abschnitt 2.2.1. Die Ausrichtung des Sensors und Montageposition am Träger definieren die Basis des Sensorkoordinatensystems.

Die *äußere Orientierung* beschreibt die Lage des Sensors im Bezug zu seiner Umgebung. Über die bekannte Anbauposition am Träger ist eine Überführung der Messpunkte ins *trägerfeste Koordinatensystem* (Index *f*) möglich. Nach Luft- und Raumfahrtkonvention wird der Ursprung des Trägerkoordinatensystems im Massenzentrum des Luftfahrzeugs positioniert. Dabei zeigt die positive *x*-Achse in Richtung der „Nase“ und die positive *y*-Achse in Richtung der rechten Tragfläche, also „nach rechts“ beim Hubschrauber (vergleiche Buchholz (2002)). Die *z*-Achse steht nach rechter-Hand-Regel senkrecht dazu und zeigt „nach unten“. Die Ausrichtung und Position des Hubschraubers im Raum definieren die Basis für das Trägerkoordinatensystem.

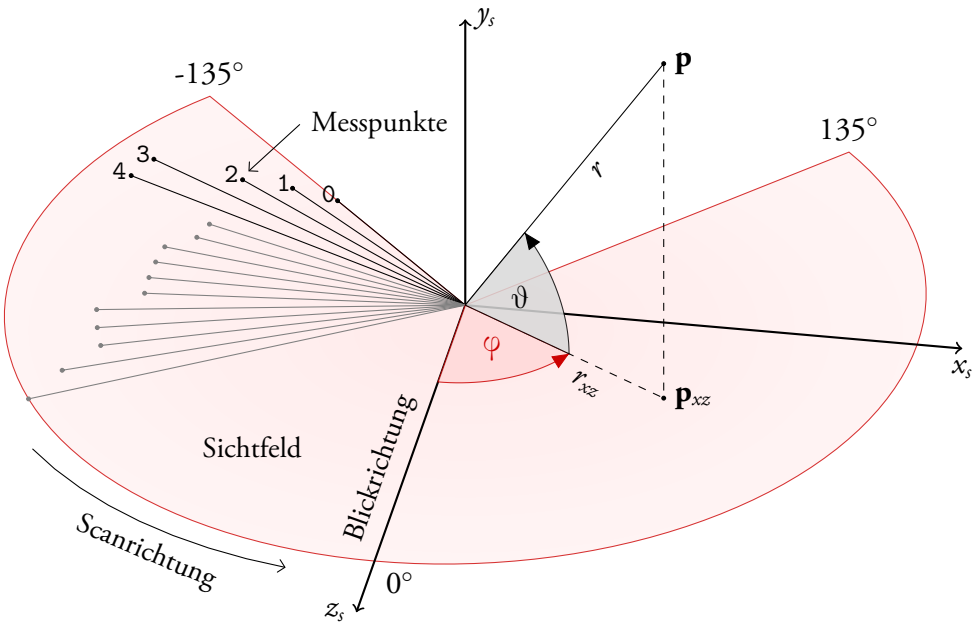
Ist die Orientierung des Trägers im Raum bekannt, können Messpunkte vom Träger- ins Weltkoordinatensystem übertragen werden. Als geodätisches *Weltkoordinatensystem* (Index *w*) wird ein NED-System (north, east, down) benutzt, bei dem die *x*-Achse nach Norden, die *y*-Achse nach Osten und die *z*-Achse in Richtung der Schwerkraft zeigt. Mit den für eine Verortung notwendigen Transformationen befasst sich Abschnitt 2.2.2.

Zur Erstellung einer Karte wird später der Raum diskretisiert und in ein Raster mit

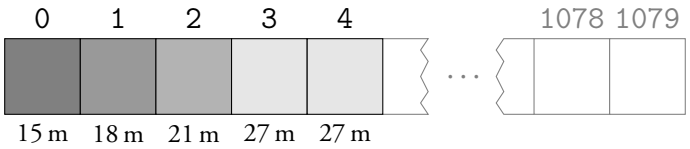
fester Zellgröße eingeteilt. Dafür wird ein viertes, das *Gitterkoordinatensystem* (Index *g*), eingeführt, welches sich zwar am Weltkoordinatensystem orientiert, aber nicht in dessen Ursprung liegen muss.

2.2.1 Herleitung der Sensorkoordinaten

Für einen starr montierten, zeilenbasierten Laserscanner wie den UTM-30LX genügt ein zweidimensionales Koordinatensystem zur Berechnung der Sensorkoordinaten, da die einzelnen eindimensionalen Messungen alle in einer Ebene liegen. Es wird allerdings ein drei-



(a) 1080 Messimpulse werden über einen Öffnungswinkel von $\varphi = 270°$ verteilt und liegen für $\vartheta = 0°$ alle in einer Ebene.



(b) Gemessene Distanzen werden als Grauwerte codiert in einer Bildzeile ausgegeben (Messpunkte 0–4).

Abbildung 2.4: Datenaufzeichnung des verwendeten Laserscanners.

dimensionales Koordinatensystem verwendet, um Messungen anderer Sensoren im gleichen System abbilden zu können, oder nach Möglichkeit den Scanner auf einer zusätzlichen Achse schwingend zu montieren. Abbildung 2.4(a) zeigt schematisch, wie der eingesetzte Laserscanner beim Messvorgang vorgeht und wie sich die Messpunkte über räumliche Polarkoordinaten darstellen lassen. Es werden 1080 Messungen in Winkelabständen von 0,25° in einer Ebene durchgeführt. In der aktuellen Implementierung werden die in Millimetern gemessenen Distanzen in 16 bit-Grauwerte codiert und als Bildzeile zur Weiterverarbeitung ausgegeben. Abbildung 2.4(b) veranschaulicht dies am Beispiel der Messpunkte 0–4.

Verallgemeinert lässt sich jeder Messpunkt $\mathbf{p}$ als Tripel aus einer gemessene Distanz r und den zugehörigen Winkeln φ und ϑ mit den räumlichen Polarkoordinaten

$$\mathbf{p} : \begin{pmatrix} r \\ \varphi \\ \vartheta \end{pmatrix}_{polar} \quad \text{mit } r \in \mathbb{R}_0^+,\ \varphi \in \mathbb{R}_{-180°}^{180°},\ \vartheta \in \mathbb{R}_{-90°}^{90°} \tag{2.1}$$

beschreiben. Wird die Basis des Polarkoordinatensystems wie in Abbildung 2.4(a) gewählt, ist φ der Winkel zwischen positiver z_s-Achse und der Projektion $\mathbf{p}_{xz}$ von $\mathbf{p}$ in die x_sz_s-Ebene, sowie ϑ der Winkel zwischen x_sz_s-Ebene und $\mathbf{p}$. Am Beispiel von geografischen Koordinaten entspricht φ der geografischen Länge und ϑ der geografischen Breite.

Zur weiteren Transformation werden kartesische Sensorkoordinaten aus den Polarkoordinaten wie folgt berechnet:

$$\mathbf{p}_s : \begin{pmatrix} x_s \\ y_s \\ z_s \end{pmatrix} = \begin{pmatrix} r \cdot \cos\vartheta \cdot \sin\varphi \\ r \cdot \sin\vartheta \\ r \cdot \cos\vartheta \cdot \cos\varphi \end{pmatrix} \in \mathbb{R}^3. \tag{2.2}$$

Der verwendete Laserscanner misst mit einem Öffnungswinkel von 270° (vgl. Tabelle 2.2, S. 12) nur in der Ebene. Daher gilt stets $\vartheta = 0°$ für alle Messungen und es folgt

$$\begin{pmatrix} x_s \\ y_s \\ z_s \end{pmatrix} = \begin{pmatrix} r \cdot \sin\varphi \\ 0 \\ r \cdot \cos\varphi \end{pmatrix} \in \mathbb{R}^3. \tag{2.3}$$

Messpunkt $\mathbf{p}$ fällt somit in den Kreissektor beschränkt durch $\varphi \in \mathbb{R}_{-135°}^{135°}$ mit $r = r_{xz}$ und es gilt $\mathbf{p} = \mathbf{p}_{xz}$. Dieser Kreissektor liegt in der $x_s z_s$-Ebene des kartesischen Koordinatensystems und definiert so das Sichtfeld des Laserscanners mit Blickrichtung entlang der positiven z_s-Achse.

2.2.2 Koordinatentransformation

Die Transformation eines Punktes $\mathbf{p}$ aus einem Koordinatensystem a in ein Koordinatensystem b erfolgt wie von Andert (2011, Kapitel 2.2) beschrieben. Zuerst wird das System a nach b durch den Vektor $\mathbf{t}_{ba}$ verschoben, der den Ursprung von b in a darstellt. Die anschließende *passive* Rotation $\mathbf{R}_{ba}$ rotiert die Achsen von System a so, dass sie auf denen von System b liegen und somit gilt zusammenfassend

$$\mathbf{p}_b = \mathbf{R}_{ba} \cdot \left(\mathbf{p}_a - \mathbf{t}_{ba}\right) \tag{2.4}$$

für die Vorwärtstransformation. Die Rücktransformation geschieht durch

$$\mathbf{p}_a = \mathbf{R}_{ba}^{-1} \cdot \mathbf{p}_b + \mathbf{t}_{ba}, \tag{2.5}$$

wobei die Matrix $\mathbf{R}$ orthonormal und somit $\mathbf{R}^{-1} = \mathbf{R}^{\top}$ ist.

Rotiert wird um die Eulerwinkel Ψ, Θ und Φ (Gier-, Nick-, Roll-Winkel in Anlehnung an die Deutsche Luft- und Raumfahrtnorm, DIN 9300). Gierwinkel Ψ rotiert um die z-Achse, Nickwinkel Θ um die y-Achse und Rollwinkel Φ um die x-Achse. Die Verknüpfung der einzelnen Drehmatrizen

$$\mathbf{R}_z(\Psi) = \begin{pmatrix} \cos\Psi & \sin\Psi & 0 \\ -\sin\Psi & \cos\Psi & 0 \\ 0 & 0 & 1 \end{pmatrix}, \quad \mathbf{R}_y(\Theta) = \begin{pmatrix} \cos\Theta & 0 & -\sin\Theta \\ 0 & 1 & 0 \\ \sin\Theta & 0 & \cos\Theta \end{pmatrix} \text{ und}$$
$$\mathbf{R}_x(\Phi) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\Phi & \sin\Phi \\ 0 & -\sin\Phi & \cos\Phi \end{pmatrix} \tag{2.6}$$

ergibt die finale Rotationsmatrix $\mathbf{R}_{ba}$ wie folgt:

$$\mathbf{R}_{ba} = \mathbf{R}_x(\Phi)\mathbf{R}_y(\Theta)\mathbf{R}_z(\Psi). \tag{2.7}$$

Da Matrixmultiplikationen nicht kommutativ sind, ist die korrekte Multiplikationsreihenfolge zu beachten, damit die Drehungen in vorgesehener Reihenfolge (z-Achse $\rightarrow$ y-Achse $\rightarrow$ x-Achse) ablaufen. Anders als bei üblichen Rotationsmatrizen, die um *feste* Achsen des Ausgangssystems drehen, werden die Achsen hier bei jedem Schritt mitgedreht.

2.3 Datenstrukturen zur Raumteilung

Im Verlauf dieser Arbeit wird dreidimensionaler Raum zum Verarbeiten von Sensormessungen diskretisiert und in ein gleichmäßiges Raster unterteilt. Einen Einfluss auf die Leistungsfähigkeit eines Systems hat in diesem Zusammenhang die Wahl der Datenstruktur, mit der diese Rasterung implementiert wird. Die vorgestellten Datenstrukturen weisen Unterschiede im Laufzeitverhalten für Elementzugriff und Elementsuche, sowie im Speicherplatzverhalten auf. Zum Ausdrücken der asymptotischen Laufzeitkomplexität, welche beschreibt, wie sich ein Algorithmus bei Änderung einer Problemgröße n verhält, wird im Folgenden die gängige Landau-Notation benutzt (Mehlhorn u. a. 2008). Formal gilt für die Funktionen $f, g : \mathbb{N}_0 \rightarrow \mathbb{R}^+$ wenn $\exists k \in \mathbb{R}^+$ und $\exists n_0, n \in \mathbb{N}_0$, dass

- mit $g \in O(f)$ Funktion g höchstens so schnell wächst wie f wenn

$$\forall n \geq n_0 : g(n) \leq k \cdot f(n), \tag{2.8}$$

- mit $g \in \Theta(f)$ Funktion g sich gleich verhält mit f wenn

$$\forall n \geq n_0 : \frac{1}{k} \cdot f(n) \leq g(n) \leq k \cdot f(n). \tag{2.9}$$

$O(f)$ dient somit als obere Schranke und $\Theta(f)$ wird benutzt, wenn das Verhalten exakt beschrieben werden kann. Die Größe n steht, am Beispiel einer Datenstruktur, für die Anzahl der verwalteten Elemente.

2.3.1 Arrays

Ein *Array* (Feld) bezeichnet in der Informatik eine statische, primitive Containerstruktur, in der Elemente aneinandergereiht gespeichert werden. Während der Laufzeit bleibt, unabhängig von Einfüge- und Löschoperationen, die Größe eines initialisierten Arrays gleich. Der Zugriff auf Elemente geschieht direkt über Indizes auf Zellen im Speicher. Dabei ist es egal, ob es sich um ein eindimensionales, zweidimensionales oder beliebigdimensionales Array handelt: Der Zugriff auf ein konkretes Element ist immer konstant, also in $\Theta(1)$, da die Größe eines Arrays im Vorfeld bekannt ist und jedes Element über den eindeutigen Index erreichbar ist. Die Suche im Array nach einem beliebigen Element, dessen Index nicht bekannt ist, nimmt $O(n)$ Zeit in Anspruch. Im Zuge der 3D-Hindernismodellierung muss ein Raster komplett durchlaufen werden, um an alle Informationen zu gelangen. Bei Arrays geschieht dies sequenziell und hat bei einer Anzahl von n Elementen entsprechend die Komplexität $\Theta(n)$. Ähnlich sieht es bei dem Speicherplatzbedarf für statische Arrays aus: Durch die feste Größe wird beim Erstellen des Arrays direkt der gesamt benötigte Speicher reserviert, also n Speicherbereiche der Größe des zu speichernden Datentyps, *auch* wenn noch keine Daten enthalten sind. Die Bildzeile in Abbildung 2.4(b) auf Seite 14 ist ein Beispiel eines eindimensionalen Arrays.

2.3.2 Baumstrukturen

Eine Alternative stellen hierarchische Baumstrukturen zur Speicherung oder Verwaltung von Daten dar. Sie werden oft dort eingesetzt, wo dynamische Daten mit einer bestimmten Ordnung zu modellieren sind. Baumstrukturen können im Gegensatz zu Arrays zur Laufzeit ihre Größe und Struktur ändern. Dabei wird zu Anfang nicht direkt ein großer zusammenhängender Speicherbereich alloziert, sondern werden nach und nach kleinere Fragmente zur Speicherung der nötigen Daten reserviert. Der Zugriff auf einzelne Elemente erfolgt über Referenzen auf Speicherbereiche.

Allgemeine Definition und N-Bäume

In diesem Kontext wird ein Baum definiert als ein azyklischer, zusammenhängender, ungerichteter Graph mit einem als *Wurzel* ausgezeichneten *Knoten* (Mehlhorn u. a. 2008, Kapitel 2). In solch einem Baum verlaufen die Kanten zwischen Elternknoten und Kinder-

knoten, wobei sich Kinder immer auf einer ihren Eltern untergeordneten Ebene befinden. Knoten mit dem selben Elternknoten heißen Geschwister. Knoten ohne Kinder sind *Blätter* und alle auf dem Pfad von den Blättern zur Wurzel liegenden Knoten werden als *innere Knoten* bezeichnet. Abbildung 2.5 veranschaulicht dies am Beispiel eines Binärbaums der Höhe 3, wobei Blätter als rechteckige Knoten gekennzeichnet sind. Es besteht eine Ord-

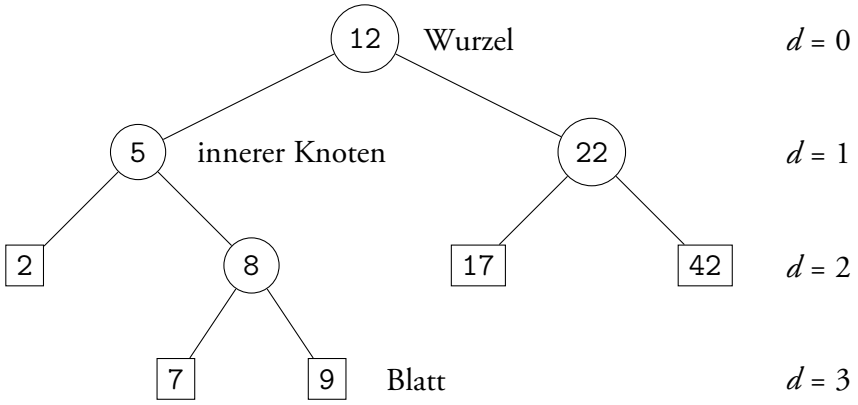


Abbildung 2.5: Geordneter Binärbaum der Höhe 3.

nungsstruktur, in der Knoten mit niedrigen Werten linke Kinder und höherwertige Knoten rechte Kinder ihres Elternknotens sind. Die Länge des Pfades von der Wurzel zu einem Knoten bezeichnet die Tiefe d eines Knotens. Die Höhe h des Baums ist die Länge des Pfades von der Wurzel zum tiefsten Knoten.

Der Binärbaum (2-Baum) ist die spezielle Form eines N-Baums, bei dem jeder Knoten nicht mehr als zwei Kinder hat. Die maximale Anzahl an Knoten im N-Baum der Höhe h ist

$$\frac{N^{h+1} - 1}{N - 1} \tag{2.10}$$

und es gibt maximal N^h Blätter (Preiss 1998, Kapitel 9.2). Der Zugriff auf Knoten geht immer von der Wurzel aus und ist nicht wie beim Array mit konstanter Laufzeit möglich, sondern resultiert immer in der Suche nach einem konkreten Knoten, auf dessen Information anschließend zugegriffen wird oder der bei Nichtvorhandensein erstellt wird. Dabei ist die Komplexität der Suche abhängig von der Struktur des Baums und nimmt im schlimmsten Fall $O(n)$ in Anspruch, wenn der Baum in Abbildung 2.5 zum Beispiel nur aus den Knoten 12, 22 und 42 bestehen würde. Dabei steht n hierbei für die Gesamtzahl der Knoten des Baums. Für den durchschnittlichen Fall kann allerdings eine Komplexität

von $O(\log n)$ für eine Suche (Knotenzugriff) im geordneten Baum angenommen werden (Mehlhorn u. a. 2008, Kapitel 7). Ein *Traversieren* des Baums bezeichnet einen kompletten Durchlauf aller Knoten, ausgehend vom Wurzelknoten. Es kann auf verschiedene Weisen implementiert werden und nimmt gewöhnlich $\Theta(n)$ Zeit in Anspruch. Tabelle 2.3 fasst die Komplexität gängiger Operationen auf Arrays und geordneten N-Bäumen zusammen. Die konkrete Anwendung eines N-Baums als 8-Baum (Octree) für dreidimensionale Gitter wird im Kapitel zur Umgebungsmodellierung 3.4.1 näher erläutert.

Tabelle 2.3: Komplexität gängiger Operationen mit n als Anzahl der Elemente beim Array und als Anzahl der Knoten beim N-Baum.

	Array	N-Baum
Zugriff	$\Theta(1)$	$O(\log n)$
Suche	$O(n)$	$O(\log n)$
Traversierung	$\Theta(n)$	$\Theta(n)$
Speicher	$\Theta(n)$	$O\left(\frac{N^{h+1}-1}{N-1}\right)$

Weitere Baumstrukturen

Es gibt weitere hierarchische Baumstrukturen, die sich auch zur Verwaltung mehrdimensionaler Daten eignen. k-d-Bäume sind stets geordnete Binärbäume, die vorwiegend zur Speicherung von Punktdaten in k Dimensionen Verwendung finden (Bentley 1975). Sie können Arrays gegenüber ebenfalls Vorteile bieten, eigenen sich aber weniger zur Darstellung eines homogen gerasterten Raums, vor allem in höheren Dimensionen. Da nur zwei Kinder pro Knoten möglich sind, wird der Baum schnell sehr hoch. Jeder Knoten teilt den Raum in eine Koordinatenrichtung. Wenn in 2D zum Beispiel die Wurzel den Raum in x-Richtung teilt, teilen ihn Knoten bei Tiefe 1 in y-Richtung und Knoten auf Tiefe 2 wieder in x-Richtung, usw.

R-Bäume und B-Bäume finden in Datenbanken Verwendung, wo sie zum Indizieren von (räumlichen) Daten genutzt werden. Ihr Fokus liegt auf möglichst schnellen Suchanfragen, weswegen sie ausgewogen sind. Das heißt, dass alle Blätter möglichst auf einer Tiefe liegen. Dieses Kriterium sichert eine Komplexität von $O(\log n)$ für Suchanfragen und verhindert degenerierte Bäume mit Suchen nahe von $O(n)$ (wie im Baum aus Abbildung 2.5,

nur aus den Knoten 12, 22, 42 bestehend). Für eine gleichmäßige Raumteilung kann diese Ausgewogenheit allerdings nicht immer gewährleistet werden.

2.4 Probabilistische Grundlagen

Dieser Abschnitt behandelt die nötigen Grundbegriffe der Wahrscheinlichkeitsrechnung und deren Notation. *Zustände* der Umgebung oder Sensormessungen können unterschiedliche Werte annehmen und werden daher als Zufallsvariablen modelliert (Thrun u. a. 2005, Kapitel 2.2). Sei X eine Zufallsvariable und das Ereignis x ein konkreter Wert, den diese Zufallsvariable annehmen kann, dann bezeichnet

$$P(X = x) = P(x) \in [0, 1]_{\mathbb{R}} \tag{2.11}$$

die Wahrscheinlichkeit, dass X den Wert x annimmt, also das Ereignis x eintritt. Bei einer diskreten Zufallsvariable wie einem Münzwurf mit $X = \{\text{Kopf}, \text{Zahl}\}$ kann eine *Anfangswahrscheinlichkeit* von $P(\text{Kopf}) = P(\text{Zahl}) = 0,5$ formuliert werden. Dabei wird ohne weitere Annahmen beiden Ereignissen die gleiche Wahrscheinlichkeit zugeordnet. Diskrete Wahrscheinlichkeiten addieren sich stets zu 1:

$$P(\text{Kopf}) + P(\text{Zahl}) = 1. \tag{2.12}$$

Die Gegenwahrscheinlichkeit $P(\neg\text{Kopf})$ ist

$$P(\neg\text{Kopf}) = 1 - P(\text{Kopf}). \tag{2.13}$$

Die *multivariate Verteilung* zweier Zufallsvariablen X und Y

$$P(X = x \text{ und } Y = y) = P(x, y) \tag{2.14}$$

ist die Wahrscheinlichkeit, dass beide Ereignisse x und y eintreten. Sind die Zufallsvariablen X und Y unabhängig, dann gilt

$$P(x, y) = P(x) \cdot P(y). \tag{2.15}$$

Für den häufigen Fall der Abhängigkeit zweier Zufallsvariablen beschreibt die *bedingte* Wahrscheinlichkeit

$$P(X = x | Y = y) = P(x|y) \tag{2.16}$$

die Wahrscheinlichkeit des Ereignisses x, für den Fall dass y bereits bekannt ist. Für $P(y) > 0$ ist die bedingte Wahrscheinlichkeit definiert als

$$P(x|y) = \frac{P(x, y)}{P(y)}. \tag{2.17}$$

2.5 Zusammenfassung

Die nun folgende 3D-Modellierung der Hubschrauberumgebung wird am Beispiel von Lasermessdaten vorgenommen (vgl. Kapitel 2.1.2 und 2.2.1). Ein Messdurchgang des Scanners besteht aus 1080 aufeinander folgenden Einzelmessungen. Die Umgebung wird durch ein metrisches Raster diskretisiert, welches anfangs als Array implementiert ist und im Zuge der Optimierung auf eine Baumstruktur umgestellt wird (Kapitel 2.3). Probabilistische Funktionen (Kapitel 2.4) ermöglichen dabei ein inkrementelles Verarbeiten der Einzelmessungen, um mit jedem neuen Messdurchgang den aktuellen Zustand der Umgebung darstellen zu können.

3 Umgebungsmodellierung

Bei der autonomen Navigation besteht die Herausforderung darin, Hindernisse rechtzeitig zu erkennen und auf diese reagieren zu können. Dazu wird ein geeignetes System zur Modellierung der Umgebung benötigt, um Hindernisse persistent über den Flugzeitraum vorzuhalten. Der Kleinhubschrauber midiARTIS muss laufend mit seiner Umgebung interagieren und auf aktuelle Sensormessungen zugreifen, um entsprechende Entscheidungen, beispielsweise zur Pfadplanung, treffen zu können. Dieses Kapitel gibt einen Überblick über die Umgebungsmodellierung auf Basis sogenannter Belegtheitsgitter, welche zur Integration von Sensordaten verwendet werden.

In Abschnitt 3.2 wird die Integration der Messdaten am Beispiel eines Scanners zuerst für den allgemeinen Fall, unabhängig von einer konkreten Datenstruktur des Gitters, durchgeführt. Auf dieser Grundlage wird in 3.3 das zugrundeliegende Weltmodell von Andert (2011) eingeführt, welches diese Belegtheitsgitter mittels Array-Datenstrukturen implementiert und damit die Modellierung abstrahierter Hindernisse ermöglicht.

Zur Leistungsoptimierung und größeren Flexibilität bei der Sensorwahl erfolgt in Abschnitt 3.4 eine Optimierung dieses Weltmodells, um die Speicher- und Leistungsvorteile hierarchischer Baumstrukturen nutzen zu können. Dabei wird das Belegtheitsgitter durch einen Octree realisiert und zusätzlich die Extraktion von Hindernissen aus dem Gitter speichereffizienter gestaltet. Auf beide Abschnitte zur Weltmodellierung folgt je eine kurze Zusammenfassung der jeweiligen Prozessierungsschritte. So können Unterschiede beider Modelle zur Analyse übersichtlich gegenübergestellt werden.

3.1 Rasterkarten als stochastisches Belegtheitsgitter

Werden für die Umgebungswahrnehmung Sensoren zur Entfernungsmessung eingesetzt, liegt es nahe, eine metrische Darstellung des Raums zu wählen. Die Kartierung von Hin-

dernissen erfolgt vorerst in einer Rasterkarte, da sich diese gut zur Objektmodellierung aus Sensormessdaten eignet. Dabei wird der betrachtete dreidimensionale Raum um den Sensorträger in ein regelmäßiges Raster kubischer Zellen gleicher Größe eingeteilt. Dieses Raster ist parallel zu den Achsen des Weltkoordinatensystems ausgerichtet, so dass Hindernisse der Umgebung diskretisiert durch eine Vielzahl solcher Zellen dargestellt werden, wobei eine Zelle das kleinstmögliche Hindernis darstellt und ihre Größe somit die räumliche Auflösung der Rasterkarte definiert. Die Diskretisierung erlaubt eine einfachere Handhabung von Koordinaten und ermöglicht die Anwendung zellbasierter Algorithmen. Im Folgenden werden die würfelförmigen Rasterzellen auch als *Voxel*[1] bezeichnet. Die Begriffe *Raster* und *Gitter* sind im Kontext dieser Arbeit allgemein gehalten und stehen nicht im Bezug zu einer konkreten Datenstruktur. Es gibt weiterhin topologische und semantische Kartenformen, die hier allerdings nicht näher betrachtet werden, da sie zur direkten Abbildung metrischer Sensordaten weniger geeignet sind. Meist sind diese Darstellungsformen als anschließend abgeleitete Karten zu finden.

Zur Kollisionsvermeidung bei der Navigation reicht es theoretisch aus, Freiraum als „freie“ Zellen abzubilden und für jeden Sensormesspunkt eines Hindernisses die ihn umschließende Zelle als „belegt“ zu kennzeichnen. Solch eine binäre Rasterkarte wird auch als *Belegtheitsgitter* bezeichnet. Die direkte binäre Abbildung von Sensordaten ist allerdings in der Praxis problematisch, da von unsicheren Sensormessungen oder Fehlern in der Positionierung des Trägers ausgegangen werden muss, welche eine fehlerhafte Karte zur Folge haben können. Zum Vermeiden etwaiger Fehler bezieht das Verfahren die *Sicherheit* einer Belegung mit ein: Wird eine Zelle mehrmals „getroffen“, so stellt sie mit höherer Sicherheit ein Hindernis dar, als eine Zelle die lediglich einmal getroffen wurde. Auf diese Weise können auch fälschlich als belegt gekennzeichnete Zellen durch Reduzierung ihrer Sicherheit wieder „freigezeichnet“ werden.

Es gibt unterschiedliche Ansätze zum Einbeziehen solcher Unsicherheiten in der Interpretation von Sensormessungen. Andert (2011) gibt einen Überblick der verbreitetsten Methoden. Für das von ihm beschriebene Weltmodell und diese Diplomarbeit werden Belegtheitswahrscheinlichkeiten nach Moravec und Elfes (1985) und Elfes (1989) verwendet, um Unsicherheiten auf Zellebene einfließen zu lassen. Mit diesem Ansatz wird der Wertebereich von Zellen im Raster entsprechend erweitert, um ihren Belegtheitszustand als

[1] Zusammensetzung aus *volumetric* und *pixel*, wobei pixel für *picture element* (Bildelement) steht.

Belegtheitswahrscheinlichkeit (occupancy probability) speichern zu können. Es handelt sich um ein stochastisches Verfahren, mit dem unbekannte Bereiche der Umgebung einbezogen werden. Ausgangspunkt für die Datenintegration ist entsprechend kein leerer Raum, in den Belegungen eingetragen werden, sondern ein *unbekannter* Raum, in den Belegungen und Freiraum eingetragen werden. Auch Eigenschaften unterschiedlicher Sensoren, wie beispielsweise entfernungsabhängige Signifikanz einer Messung, können durch individuelle Sensormodelle berücksichtigt werden (siehe Abschnitt 3.2). Dadurch wird ebenfalls eine Datenfusion unterschiedlicher Entfernungssensoren in *einem* Gitter ermöglicht. Der folgende Abschnitt befasst sich mit der Notation von Belegtheitswahrscheinlichkeiten im Belegtheitsgitter, damit diese bei der Integration von Messdaten berücksichtigt werden können.

3.1.1 Notation von Belegtheitsgittern

Ein Belegtheitsgitter M fester räumlicher Ausdehnung im Gitterkoordinatensystem g hat eine Gitterauflösung c, welche die metrische Kantenlänge einer Zelle dieses Gitters festlegt. Einer Sensormessung $\mathbf{p}_w$ in Weltkoordinaten wird der Mittelpunkt ihrer umschließenden Gitterzelle $m \in M$ durch die ganzzahlige Gitterkoordinate $\mathbf{p}_g$ als

$$\mathbf{p}_g : \begin{pmatrix} x_g \\ y_g \\ z_g \end{pmatrix} = \left\lfloor \frac{(\mathbf{p}_w - \mathbf{t}_{gw})}{c} + 0,5 \right\rfloor \in \mathbb{Z}^3 \tag{3.1}$$

zugeordnet, wobei Vektor $\mathbf{t}_{gw}$ den Ursprung des Gitterkoordinatensystems in Weltkoordinaten darstellt. Umgekehrt lässt sich der Mittelpunkt der Gitterzelle m in Weltkoordinaten durch

$$\mathbf{p}_w = c \cdot \mathbf{p}_g + \mathbf{t}_{gw} \tag{3.2}$$

berechnen.

Sei $m_i \in M$ die i-te Gitterzelle und ihr möglicher Belegtheitszustand $o \in \{\text{frei}, \text{belegt}\}$, dann bezeichnet

$$P(m_i = \text{belegt}) = P(m_i) \tag{3.3}$$

die Wahrscheinlichkeit der Belegung dieser Zelle und P dient als „Maß für die Sicherheit

der Vermutung eines Objektes an der entsprechenden Koordinate“ (Andert 2011, Kapitel 3.2). Auf dieser Grundlage können nun die Messdaten des Laserscanners ins Gitter übertragen werden.

3.2 Integration von Laserscans

Für die Integration von Laserscans ins Umgebungsmodell wird von einer hinreichend genauen Navigationslösung des Hubschraubers ausgegangen. Dabei berechnet der Bordcomputer zur Laufzeit Navigationsdaten bestehend aus Position und Ausrichtung des Hubschraubers durch Kombination einer GNSS-Lösung (hier am Beispiel von GPS) mit Messungen des Inertialsensors (IMU) und des Magnetometers in einem Kalman-Filter. Die detaillierte Herleitung der Navigationslösung wird in der Arbeit von Lorenz (2010) beschrieben.

Jeder Laserscan wird mit einem Zeitstempel versehen, wodurch zu jeder seiner 1080 Einzelmessungen ein Messzeitpunkt interpoliert werden kann. Über interne Zeitsynchronisation kann so jeder Einzelmessung über ihren Messzeitpunkt eine globale Hubschrauberposition und -ausrichtung aus den Navigationsdaten zugeordnet werden. Damit lässt sich unter Verwendung von Gleichung 2.4 eine Einzelmessung $\mathbf{p}$ aus dem Sensorkoordinatensystem s indirekt über das Trägersystem f mit

$$\mathbf{p}_w = \mathbf{R}_{wf} \cdot (\underbrace{\mathbf{R}_{fs} \cdot \left(\mathbf{p}_s - \mathbf{t}_{fs}\right)}_{\mathbf{p}_f} - \mathbf{t}_{wf}) \tag{3.4}$$

in das Weltkoordinatensystem w überführen.

Um Sensormessungen im Belegtheitsgitter abzubilden, müssen diese zunächst korrekt interpretiert werden. Messvorgänge verschiedener Sensoren weisen unterschiedliche Eigenschaften auf, die meist charakteristisch für einen Sensortyp und abhängig von dessen Funktionsweise und Systemparametern sind. Diese Eigenschaften werden modelliert, um eine möglichst genaue Interpretation der Messdaten und damit verbunden eine korrekte Abbildung in ein Belegtheitsgitter zu erreichen. Zum Beispiel nimmt die Tiefenauflösung einer Stereokamera mit der Entfernung quadratisch ab (Andert 2011; Hrabar 2012), was bei der direkten Messung des verwendeten Laserscanner nicht der Fall ist. Jedem Sensor oder

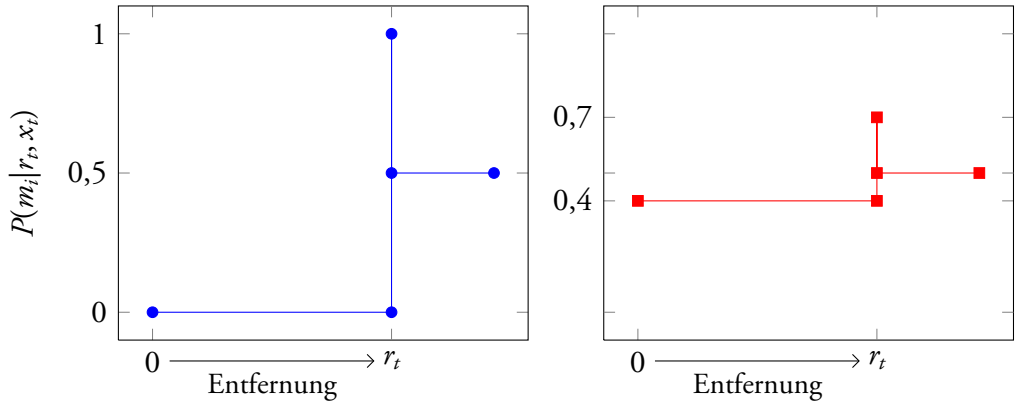


Abbildung 3.1: Ideales —●— und verwendetes Sensormodell —■—.

Sensortyp wird demnach ein *Sensormodell* zugeordnet, welches dessen Messeigenschaften beschreibt.

Für die Kartierung mittels Laserscanner wird hier ein *inverses* Modell nach Elfes (1989) benutzt. Es beschreibt den Belegtheitszustand einer Gitterzelle in Abhängigkeit von einer Sensormessung und der Sensorposition. Inverse Modelle werden oft für diejenigen Fälle benutzt, bei denen der Zustandsraum einfacher, hier binär, als der Messvorgang ist: Es ist einfacher von einer Messung auf die Belegtheit einer Zelle zu schließen, als umgekehrt alle möglichen Messungen zu beschreiben, für die eine Zelle belegt wäre, wie es im *direkten* Messmodell der Fall ist (Thrun u. a. 2005, Kapitel 4.2). Demnach wird die bedingte Wahrscheinlichkeit (Abschnitt 2.4) der Belegtheit einer Zelle abhängig von einer Entfernungsmessung r und Trägerposition x ausgedrückt als $P(m_i|r, x)$. Abbildung 3.1 zeigt ein ideales Modell für $P(m_i|r_t, x_t)$ für eine Messung zum Zeitpunkt t welches einen idealisierten Sensor beschreibt, bei dem an der gemessenen Entfernung r_t zu 100 % ein Hindernis erwartet werden kann. Entsprechend liegt die Wahrscheinlichkeit auf Hindernisse im Sichtbereich davor bei 0 % und der Raum dahinter ist für alle $r > r_t$ mit $P = 0,5$ unbekannt.

Für den verwendeten Laserscanner wird das ideale Modell mit realistischeren Werten angepasst, da aufgrund von Ungenauigkeiten durch äußere Einflüsse im Flug und die Koordinatentransformationen ein ideales Modell kaum realisierbar ist. Abbildung 3.1 zeigt

neben dem idealen auch das hier verwendete Sensormodell:

$$P(m_i|r_t, x_t) = \begin{cases} 0,4 & \text{frei} & \text{für } 0 \leq r < r_t \\ 0,7 & \text{belegt} & \text{für } r = r_t \\ 0,5 & \text{unbekannt} & \text{für } r > r_t \end{cases} \tag{3.5}$$

Zur größeren Sicherheit, Hindernisse nicht fälschlich „zu übersehen“, wird das Freizeichnen mit $P = 0,4$ gewählt. Die Messungenauigkeit des Scanners von 30 mm (vgl. Abschnitt 2.1.2) spielt bei der vergleichsweise groben Diskretisierung des Belegtheitsgitter, mit Auflösungen im Meter- bis Submeterbereich, fast keine Rolle und fließt hier nicht weiter ins Sensormodell hinein. Hrabar (2012) benutzt den gleichen Laserscanner für eine ähnliche Anwendung, allerdings mit abweichendem Sensormodell, bei dem freie Voxel entlang des Strahls zusätzlich klassifiziert werden.

3.2.1 Inkrementelles Update

Die im vorherigen Abschnitt geschilderte Integration von Messdaten ist für eine inkrementelle Kartierung noch nicht ausreichend, da Zellwerte nach dem Sensormodell immer *überschrieben* werden. Es muss eine *Akkumulation* dieser Werte stattfinden, um unterschiedliche Sicherheiten der Belegung einer Zelle ausdrücken zu können. Um bei der Akkumulation besonders hohe und niedrige Belegtheitswahrscheinlichkeiten präziser ausdrücken zu können und Probleme aufgrund von Rundungsungenauigkeiten nahe der Werte 0 und 1 zu vermeiden, wird statt der Wahrscheinlichkeit P ihr Logit-Wert L als

$$L(m_i) = \ln\left(\frac{P}{1 - P}\right) \in \mathbb{R} \tag{3.6}$$

verwendet (Thrun u. a. 2005, Kapitel 4.2). Dadurch wird der mögliche Wertebereich von $[0, 1] \in \mathbb{R}$ auf $\mathbb{R}$ erweitert.

Der Laserscanner ist am Träger montiert, die globale Trägerposition bekannt. Gemessenen Distanzen können nun nach Gleichung 3.4 in dreidimensionale Weltkoordinaten transformiert werden. Über die verlustbehaftete Umwandlung 3.1 werden sie in Koordinaten des Belegtheitsgitters transformiert und auf Grundlage des Sensormodells 3.5 ins Gitter eingetragen. Das stochastische Belegtheitsgitter wird zu Beginn als unbekannter Raum mit

der Anfangswahrscheinlichkeit von $P = 0,5$ (entsprechend $L = 0$) initialisiert. Für jede Messung ergibt sich ein Strahl zwischen Sensorposition und errechnetem Messpunkt. Ziel der Kartierung ist zunächst die bedingten Wahrscheinlichkeiten für alle Zellen m_i des Gitters M als

$$P(M|r_{1:t}, x_{1:t}) = \prod_i P(m_i|r_{1:t}, x_{1:t}) = P(M)_{1:t} \tag{3.7}$$

zu berechnen. Demnach spiegelt das Gitter zum Zeitpunkt t den aktuellen Zustand der Umgebung wider (Thrun u. a. 2005, Kapitel 9.2). $r_{1:t}$ bezeichnet alle Messungen bis zu diesem Zeitpunkt, $x_{1:t}$ beschreibt den Pfad des Trägers aus allen Teilpositionen.

Dieses inkrementelle Aktualisieren des Gitters mit jeder Messung nimmt sich das Bayestheorem zur Hilfe und wird im Anhang A für den Fall einer statischen Umgebung hergeleitet. Demnach wird die Berechnung dieser akkumulierten Wahrscheinlichkeit ermöglicht durch

$$\frac{P(M)_{1:t}}{1 - P(M)_{1:t}} = \frac{P(M)_{1:t-1}}{1 - P(M)_{1:t-1}} \cdot \frac{P(M)_t}{1 - P(M)_t} \cdot \frac{1 - P(M)_0}{P(M)_0} \tag{3.8}$$

mit $P(M)_0$ als Anfangswahrscheinlichkeit. Die Substitution mit der Logit-Funktion aus Gleichung 3.6 führt zu der einfacheren Darstellung

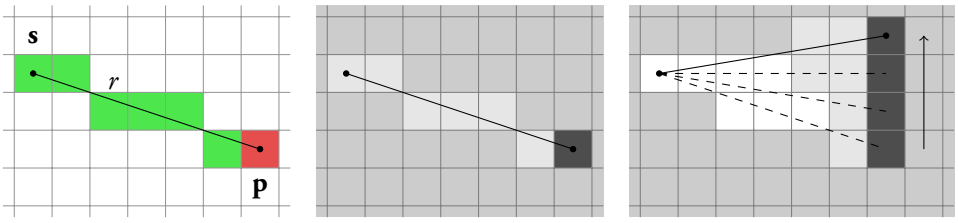
$$L(M)_{1:t} = L(M)_{1:t-1} + L(M)_t - L(M)_0. \tag{3.9}$$

Für $L(M)_0 = 0$ als Logit der Anfangswahrscheinlichkeit resultiert ein inkrementelles Update einzelner Zellen in der *Addition* des aktuellen Logit-Wertes zum Zeitpunkt t mit dem vorherigen akkumulierten Wert. Dabei werden die Zellen jedoch nur *individuell* betrachtet, das heißt es fließen keinerlei Nachbarschaftsbeziehungen ein. Das kann zu Problemen führen, worauf in Abschnitt 3.2.3 näher eingegangen wird. Zunächst aber werden mit Hilfe des inkrementellen Updates Sensormessungen ins 3D-Belegtheitsgitter rasterisiert.

3.2.2 Rasterisierung

Das Einfügen von Sensormessungen ins 3D-Belegtheitsgitter geschieht mit gängiger Linienrasterisierung, wie dem Bresenham-Algorithmus (Angel 2006, Kapitel 7.9) oder Ray Tracing (Amanatides u. a. 1987; Angel 2006, Kapitel 12.2). Zwischen Sensorposition und Messpunkt entsteht ein Strahl, dessen geschnittene Voxel im Belegtheitsgitter bestimmt

werden (Abbildung 3.2(a)). Der probabilistische Belegtheitswert dieser Voxel wird nach dem Sensormodell 3.5 und Formel 3.9 angepasst und ist in Abbildung 3.2(b) in Grauwerten dargestellt. Die Zelle um den Treffpunkt wird mit dem entsprechenden Wert für



(a) (b) (c)

Abbildung 3.2: 2D-Rasterisierung im Belegtheitsgitter zwischen Sensor **s** und Messpunkt **p** (a), Wahrscheinlichkeiten als Grauwerte (b), nach Einfügen mehrerer Strahlen (c).

„belegt“ aktualisiert. Alle Zellen zwischen Sensor und Treffpunkt werden freigezeichnet und der Raum dahinter bleibt unbekannt. Abbildung 3.2(c) zeigt die Logit-Werte des Gitters nach sequentiellem Einfügen von vier Strahlen von unten nach oben. Eine besonders geringe Belegtheitswahrscheinlichkeit mehrmals aktualisierter Zellen in Folge der inkrementellen Aktualisierung, wird durch die weißen Zellen widergespiegelt.

3.2.3 Phänomene bei der Rasterisierung

Während der Implementierung des beschriebenen Verfahrens, welches eine Aufsummierung neuer Belegtheitswahrscheinlichkeiten erlaubt, haben sich Probleme aufgrund der gekapselten Betrachtung einzelner Zellen ergeben. Dabei spielt neben der Werte für „frei“ und „belegt“ vor allem der Diskretisierungsgrad des Belegtheitsgitters eine entscheidende Rolle. Abbildungen 3.3(a) und (b) zeigen was beim sukzessiven Einfügen mehrerer Strahlen passiert: Bewegt sich der Sensor gleichbleibend nach rechts und findet die Messung in einem relativ flachen Winkel statt, wird die nach dem ersten Strahl $\overline{\mathbf{s}_0\mathbf{p}_0}$ als belegt markierte Zelle für $\mathbf{p}_0$ beim Rasterisieren des darauffolgenden Strahls $\overline{\mathbf{s}_1\mathbf{p}_1}$ wieder freigezeichnet. Gleiches trifft für die Zelle an $\mathbf{p}_1$ zu, die ebenfalls durch den nächsten Strahl $\overline{\mathbf{s}_2\mathbf{p}_2}$ „ihre Belegtheit verliert“. Wird hingegen der Diskretisierungsgrad vermindert, das heißt eine höhere Auflösung für das Gitter gewählt, kann die unerwünschte Freizeichnung für diesen

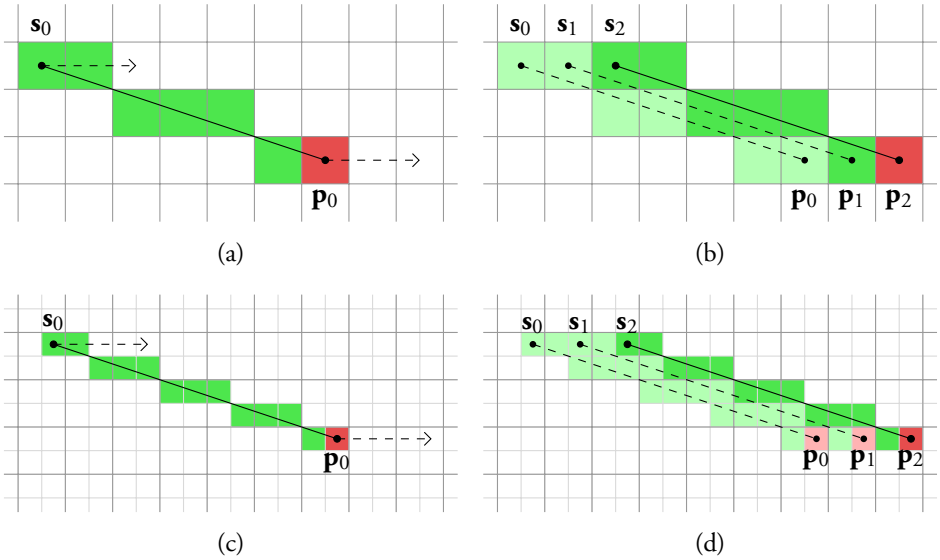
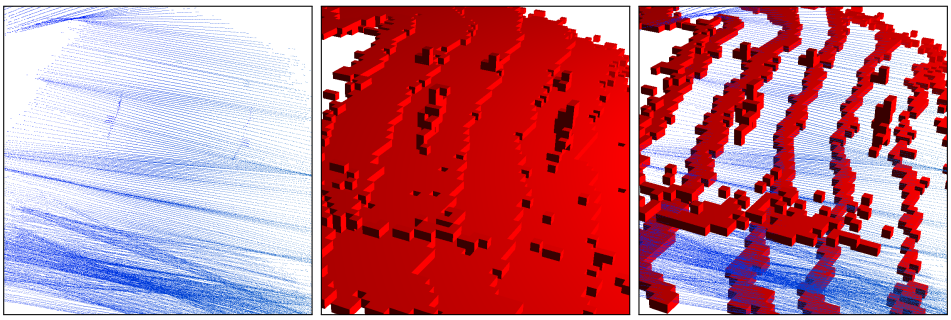


Abbildung 3.3: Unerwünschtes Freizeichnen bei der 2D-Rasterisierung: Belegte Zelle an $\mathbf{p}_0$ (a) wird durch angrenzende Strahlen wieder freigezeichnet (b). Schwächere Diskretisierung umgeht das Problem (c)–(d)

Fall vorerst umgangen werden, wie Abbildungen 3.3(c) und (d) mit halbierter Zellgröße verdeutlichen: Es werden die selben Strahlen im gleichen Abstand eingefügt. Trotzdem sind die Zellen für $\mathbf{p}_0$, $\mathbf{p}_1$ und $\mathbf{p}_2$ korrekt belegt. Hier kann das Problem für flacher eintreffende Strahlen allerdings erneut auftreten. Dieses vom Diskretisierungsgrad abhängige Phänomen wird in der Computergrafik auch als *Aliasing* bezeichnet. Ein Umgehen durch die Wahl einer hohen Auflösung ist möglich, bedeutet aber einen stark wachsenden Speicherbedarf des Gitters und somit Abstriche bei der Leistung.

Das Freizeichnungsproblem tritt nicht auf wenn lediglich die Messpunkte eingefügt werden, also nicht der gesamte Strahl rasterisiert wird. So kann zwar auch eine Karte der Hindernisse erstellt werden, aber es findet keinerlei Betrachtung des Freiraums mehr statt. Abbildung 3.4 auf Seite 32 zeigt ein Realszenario mit 0,5 m Voxelgröße als Beispiel. Der Hubschrauber fliegt im Bild von unten links nach oben rechts. Die aufgezeichnete Punktwolke aus 3.4(a) wird in (b) ohne Freiraumberechnungen eingefügt und repräsentiert den Boden korrekt. Abbildung 3.4(c) zeigt die auftretenden Freizeichnungsprobleme bei Rasterisierung der gesamten Strahlen sehr gut: Ein Großteil der belegten Zellen in Flugrichtung wird unerwünscht wieder freigezeichnet. Es resultiert generell eine Lückenhafte Darstel-



(a) (b) (c)

Abbildung 3.4: 3D-Szenario: Gescannte Punktwolke (a) im Gitter (b). Inkrementelle Kartierung mit Freiraumbetrachtung zeichnet frei (c).

lung zusammenhängender Flächen in Bewegungsrichtung. Scherer u. a. (2008) beschreiben dieses Phänomen ebenfalls und auch von Wurm, Hornung u. a. (2010) wird es bei der Kartierung mittels geschwenktem Laserscanner beobachtet. Als Lösungsvorschlag zur Vorbeugung dürfen nicht einzelne Scans eingetragen werden, sondern erst das Endresultat aller Messpunkte darf ins Gitter übertragen werden. Bei der Kartierung zwecks on-the-fly-Kollisionsvermeidung ist dies nicht möglich, da auf die Daten eines jeden Scans entsprechend reagiert werden muss.

Im Rahmen dieser Diplomarbeit wird das Aliasing nicht weiter behandelt, allerdings bietet das optimierte Octree-Weltmodell in Kapitel 3.4 eine Möglichkeit zur Umgehung des Problems, welche zum Ende in Kapitel 5.2 näher betrachtet wird. Das folgende Kapitel 3.3 widmet sich dem Modellierungsprozess, der das beschriebene Belegtheitsgitter mit einer Array-Datenstruktur realisiert.

3.3 Array-basiertes Weltmodell als Ausgangspunkt

In seiner Arbeit stellt Andert (2011, Kapitel 3) unter anderem ein Konzept zur Umgebungskartierung vor. Als Grundlage dient das beschriebene Belegtheitsgitter, welches er für die Kartierung mittels Stereokameras verwendet und welches er mit Array-Datenstrukturen implementiert. Mit den Methoden aus Abschnitt 3.2 wurde es um die Verarbeitung von Laserscannerdaten erweitert. Andert kombiniert weiterhin verschiedene Kartentypen, um aus den erfassten Sensormessdaten generalisierte Objektformen von Hindernissen extrahie-

ren zu können. Solche vereinfachten Objektformen reichen zur Umgebungsbeschreibung aus. Sie zeichnen sich durch geringen Speicherbedarf aus und eignen sich daher für die Übertragung über das on-board-Netzwerk für die anknüpfende Pfadplanung (vgl. Kapitel 1.2). Wie einleitend erwähnt, erfolgt die Steuerung des Helikopters über einen mit dem Bildverarbeitungsrechner vernetzten Bordrechner, der für die Pfadplanung mit zugrundeliegender Hinderniskarte zuständig ist. Dieses anknüpfende Verfahren wird hier nicht näher erläutert und ist in der Arbeit von Andert und Adolf (2009) beschrieben.

3.3.1 Verwendete Kartentypen

Für die spätere Pfadplanung wird ein kompakterer Kartentyp benötigt, da Rasterkarten speicherintensiv werden können und ein Zugriff auf Informationen oft eine Iteration des gesamten Rasters bedeutet, was sich negativ auf die Laufzeit auswirkt. Somit sind Rasterkarten zur Weitergabe an anschließende Verarbeitungsschritte für diesen Anwendungsfall weniger gut geeignet. Außerdem ist eine detaillierte Hindernisrepräsentation *auf Zellbasis* für die Nachverarbeitung nicht notwendig, wodurch in diesem Fall generalisierte Objektformen genügen. Abbildung 3.5 zeigt die im Weiteren verwenden Kartentypen. Zusammenhängende Voxel im Belegtheitsgitter werden gruppiert und stellen Hindernismerkmale dar. Diese werden anschließend als kompakte Prismen ausgegeben. Die Prismenrepräsen-

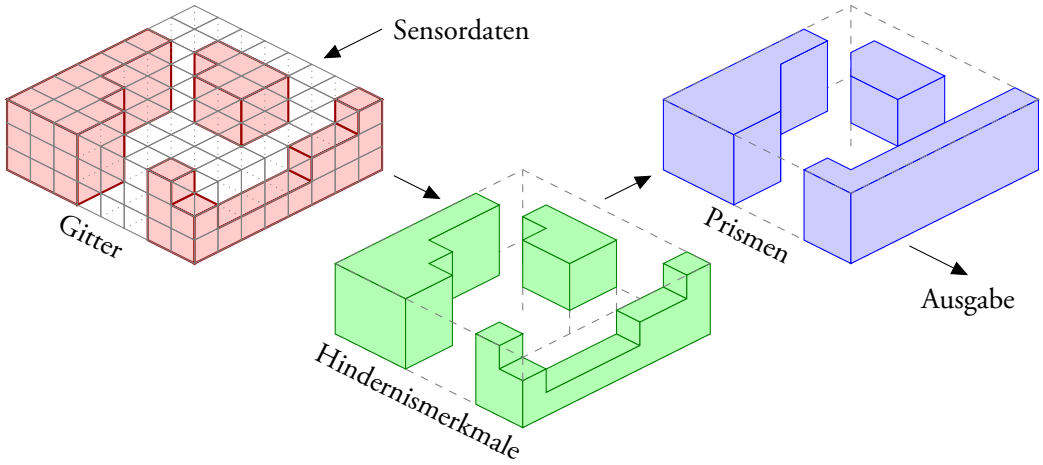


Abbildung 3.5: Verwendete Kartentypen (nach Andert 2011): Integration von Sensormessungen im Belegtheitsgitter, Gruppierung zusammenhängender Voxel zu Hindernismerkmalen und Ausgabe als kompakte Prismen.

tation erfolgt nicht auf Zellbasis, sondern abstrahiert ein Objekt durch seine Grundfläche mit einer entsprechenden Prismenhöhe.

3.3.2 Zonenaufteilung im Array-Modell

Eine Anforderung des Weltmodells ist, nicht auf ein festes Gebiet beschränkt zu sein. Da das Belegtheitsgitter in Anderts Modell mit einfachen Arrays umgesetzt ist, flogt, dass Szenarien mit großer räumlicher Ausdehnung unzureichend durch nur *ein* dreidimensionales Gitter abgebildet werden können. Auch bei verhältnismäßig grober Auflösung wäre die maximale Speicherkapazität eines Systems mit stets zunehmender Array-Größe schnell erreicht. Weiterhin sind Arrays statisch im Bezug auf ihre Größe, weshalb diese im Vornherein festgelegt werden muss.

Andert und Goormann (2007) führen ein Zonenkonzept ein, welches die Verwendung von Belegtheitsgittern in Szenarien mit theoretisch unbegrenzter räumlicher Ausdehnung ermöglicht. Es basiert auf der Voraussetzung, dass ein Belegtheitsgitter lediglich in unmittelbarer Sensorreichweite zur Integration laufender Sensormessungen benötigt wird und das Vorhalten von Hindernissen im weiteren Umfeld in kompakterer Darstellung genügt. Solch ein hinreichendes Belegtheitsgitter muss mindestens das Sichtfelds des Sensors in alle Richtungen abdecken und sollte ein für den jeweiligen Anwendungsfall angemessenes Verhältnis von Größe und Effizienz aufweisen. Im Folgenden wird ein Gitter, welches diese Anforderungen erfüllt, auch als *lokales Gitter* bezeichnet, da es durch seine beschränkten Abmessungen nur lokal, das heißt für die aktuelle Sensorposition im Raum, gültig ist. Der Begriff *lokale Zone* bezeichnet die Ausmaße dieses lokalen Gitters.

Wird von einer im Gitter zentrierten Sensorposition ausgegangen, ist für den verwendeten Laserscanner mit 30 Metern Maximalreichweite theoretisch ein lokales Gitter von $60\,\mathrm{m} \times 60\,\mathrm{m} \times 60\,\mathrm{m}$ Ausdehnung notwendig. Die Sensorposition wird allerdings auf die diskrete Koordinate einer Rasterzelle gerundet, weshalb sie variabel innerhalb einer Rasterzelle sein kann. Also wird zur Sicherheit die maximale Ausdehnung des Gitters um eine Zelle in jede Dimension erweitert. In der Praxis muss daher, bei einer Gitterauflösung von einem Meter, eine Gittergröße von $61 \times 61 \times 61$ Zellen gewählt werden.

Ändert sich die Position des Trägers, also auch die des Sensors, müsste das lokale Gitter aufgrund seiner fixen Größe ständig verschoben werden. Das aufwändige Verschieben ein-

zelner Rasterzellen wird nun verhindert, indem der gesamte dreidimensionale Raum, also die unbeschränkte Umgebung, in Zonen mit jeweiliger Größe der lokalen Zone aufgeteilt wird. Die Begrenzungen der lokalen Zone schneiden demnach maximal $2 \times 2 \times 2 = 8$ solcher Zonen. Abbildung 3.6 verdeutlicht dies an einem 2D-Beispiel zu unterschiedlichen Zeitpunkten bei sich ändernder Sensorposition. Im 3D-Fall kommen insgesamt *neun* Gitter zum Einsatz: In dem lokalen Gitter wird zunächst die laufende Sensormessung integriert. Für *jede* der acht geschnittenen Zonen wird ein weiteres Belegtheitsgitter entsprechender Größe angelegt, in das die Daten des lokalen Gitters nun anteilig eingefügt werden. Für jede folgende Messung wird das lokale Gitter wieder zurückgesetzt, die *Akkumulation* der Belegtheitswerte geschieht beim Übertragen in die acht Schnittzonen.

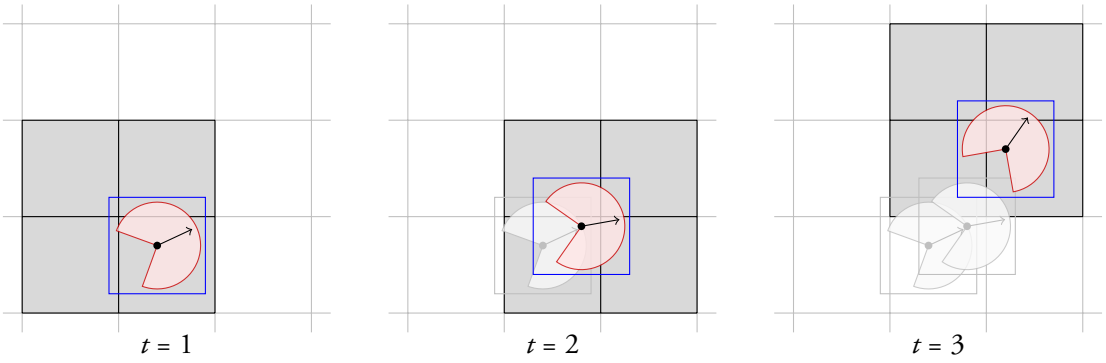


Abbildung 3.6: 2D-Beispiel: Belegtheitsgitter zu unterschiedlichen Zeitpunkten *t*. Das lokale Gitter (blau) ist dem Sensorsichtfeld (rot) angepasst, Belegtheitsgitter (grau) zusätzlich in Zonen mit Überdeckung des lokalen Gitters.

Für das Beispiel eines *lokalen Gitters* (lokale Zone) mit $61 \times 61 \times 61$ Zellen besteht das einhüllende achtteilige Gesamtgitter aus $122 \times 122 \times 122$ Zellen. Verschiebt sich die Sensorposition dahingehend, dass die lokale Zone außerhalb der aktuellen Schnittzonen liegt, werden weitere Belegtheitsgitter in den neuen Schnittzonen angelegt und Gitter der nicht mehr benötigten Schnittzonen gelöscht. Für $t = 2$ und $t = 3$ in Abbildung 3.6 sind lokale Zonen aus vergangenen Zeitpunkten hellgrau angedeutet.

Nach jedem neuen Datensatz werden Hindernisse aus den Belegtheitsgittern extrahiert und in vereinfachter Darstellung vorgehalten. Auf diese Weise gehen Informationen über Hindernisse aus den gelöschten „überflüssigen“ Zonen nicht verloren. Dieser Extraktionsvorgang wird im folgenden Abschnitt näher erläutert.

3.3.3 Extraktion von Merkmalen

Das *probabilistische* Belegtheitsgitter ermöglicht die inkrementelle Integration der Sensormessungen, so dass es zu unterschiedlichen Zeitpunkten verschiedene Zustände der Umgebungsabbildung aus Sicht des Sensors darstellt. Zur Weiterverarbeitung reicht eine Binärdarstellung in *belegten* und *freien* Zellen aus, wobei belegte Zellen Hindernismerkmale beschreiben. Mit einfacher Schwellwertbildung wird nun für jede Zelle aus dem Logit-Wert ihrer Belegtheitswahrscheinlichkeit ihr Belegtheitszustand o zum aktuellen Zeitpunkt t ermittelt als

$$o_t = \begin{cases} \text{belegt}, & \text{für } L > L_{min} \\ \text{frei}, & \text{für alle anderen Fälle.} \end{cases} \tag{3.10}$$

Der Schwellwert L_{min} kann beliebig gewählt werden. Höhere Schwellwerte schließen Unsicherheiten durch fehlerhafte Messungen aus, niedrigere Werte geben eine höhere Sicherheit Hindernisse als solche zu erkennen. In dieser Arbeit wird $L_{min} = 0$ ($P_{min} = 0,5$) gewählt, wodurch jeder Wert über der Anfangswahrscheinlichkeit als Hindernis klassifiziert wird. Die Wahrscheinlichkeit P kann nach Formel 3.6 jeder Zeit aus ihrem Logit-Wert über

$$P_t = 1 - \frac{1}{1 + e^{L_t}} \tag{3.11}$$

wieder hergestellt werden.

3.3.4 Segmentierung von Zellen im Array

Über die Schwellwertbildung werden aus Belegtheitswahrscheinlichkeiten einzelner Zellen binäre Hindernisinformationen gewonnen. Diese Darstellung reicht bereits aus, um „kollisionsfrei ans Ziel zu gelangen“. Jedoch ist der Berechnungsaufwand für die Kollisionsvermeidung unnötig hoch, da ein Objekt in dieser Form durch mehrere Zellen repräsentiert wird und die Prüfung auf ein mögliches Hindernis eine Prüfung gegen jede einzelne dieser Zellen bedeutet. Ebenfalls ist dadurch der Speicherbedarf ungünstig hoch für eine Weitergabe an anknüpfende Algorithmen.

Zur weiteren Hindernismodellierung entwickelten Andert und Goormann (2008) ein Verfahren zur geometrischen Beschreibung von Hindernissen durch polygonisierbare Objekthüllen, welche die Berechnung von Kollisionen vereinfachen. Um zellbasierte Objekte

durch polygonale Hüllen darstellen zu können, müssen belegte Zellen zunächst in zusammenhängende Bereiche gruppiert werden, damit Objekte klar voneinander getrennt sind. Ansonsten würde eine Hüllendarstellung alle belegten Zellen der Umgebung umfassen, was zu Informationsverlust über die Freiräume zwischen einzelnen Objekten führte. Es wird ein 3D-Floodfill-Verfahren nach Förterer (1995) eingesetzt, wie es in der Computergrafik zum identifizieren zusammenhängender Bildelemente genutzt wird (Andert 2011). Dabei werden die Arrays aller Zonen mit stochastischem Belegtheitsgitter zeilenweise durchlaufen und es wird für jede, dem Schwellwert L_{min} nach belegte Zelle, nach Nachbarn über eine 6er-Nachbarschaft[2] gesucht. Auf diese Weise identifizierte Objekte werden im Folgenden nach Goormann (2004) als *Blobs*[3] bezeichnet. Ihnen wird vorerst keine semantische Information zugeschrieben, wodurch sie ein beliebiges Hindernis darstellen. Entscheidend ist, dass jeder Blob nicht mehr im gesamten Belegtheitsgitter vorliegt, sondern individuell in einem *separaten* binären 3D-Array gespeichert wird. Abbildung 3.7 veranschaulicht, wie Blobs aus einem Gitter mit Belegtheitswahrscheinlichkeiten abgeleitet werden. Für $L_{min} = 0$ werden nach Gleichung 3.11 Zellen mit einer Belegtheitswahrscheinlichkeit von mehr als 50 % als Hindernismerkmal in Betracht gezogen.

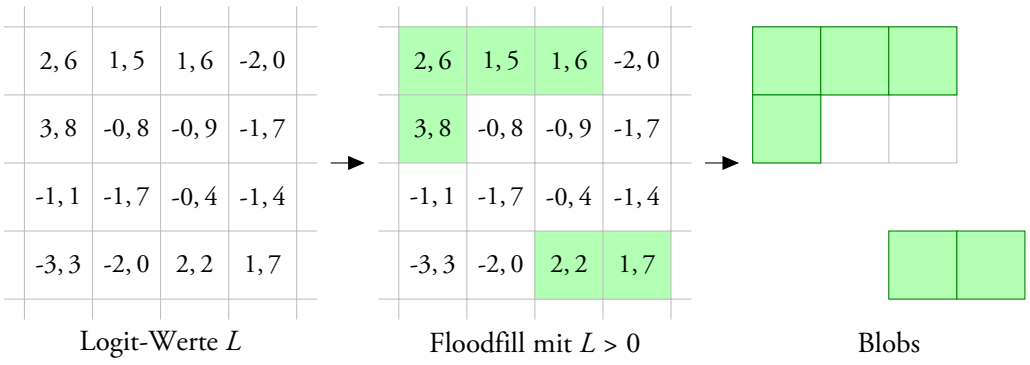


Abbildung 3.7: 2D-Darstellung: Extraktion von Blobs (grün) für einen Schwellwert von $L_{min} = 0$. Blobs liegen anschließen als *individuelle* Raster vor.

[2] 6er-Nachbarschaft bezeichnet direkte *Flächennachbarn*. Ein Voxel kann 20 weitere über seine Kanten und Ecken definierte Nachbarn besitzen.
[3] Binary large object

Reintegrieren und Zusammenfassen von Blobs

Wird im Zuge eines Zonenwechsels ein neues Belegtheitsgitter in einer Zone aufgebaut, aus der bereits Blobs extrahiert wurden, so werden diese Blobs zuerst ins neue Gitter reintegriert, bevor neue Sensormessungen aufgenommen werden. Die Blobs liegen bereits als Arrays mit bekannter globaler Position vor und können daher mit geringem Aufwand in ein Array-basiertes Belegtheitsgitter mit gleicher Auflösung eingetragen werden. Entsprechend wird im Belegtheitsgitter die Wahrscheinlichkeit einer Blob-Zelle auf „belegt“ gesetzt.

Einhergehend mit dem Zonenkonzept aus Abschnitt 3.3.2 werden im Array-Modell einzelne Blobs für die anschließende Prismengenerierung zusammengefasst, wenn sich ihre Seitenflächen an den Grenzen benachbarter Belegtheitsgitter berühren. Dieses Verfahren wird hier nicht näher erläutert, da es später im optimierten Octree-Modell nicht benutzt wird (Details dazu bei Andert 2011, Kapitel 3.3.4).

3.3.5 Weltmodellierung mit Prismen

Um die Datenmenge bei der Weitergabe an die Pfadplanung möglichst klein zu halten, werden die extrahierten Blobs aus ihrer Voxeldarstellung zu Prismen vereinfacht. Es wird in urbanen Räumen von Hindernissen mit geraden Wänden ausgegangen, wobei auch schräge Wände mit beschränktem Detailgrad dargestellt werden können, indem eine vertikale Partitionierung der Objekte vorgenommen wird. Eine solche Abstraktion ist für die Kollisionsvermeidung in der Luft ausreichend. Im Folgenden wird das Prinzip kurz beschrieben. Die detaillierte Generierung von Prismen kann der Arbeit von Andert (2011, Kapitel 3.3.4) entnommen werden.

Ein Prisma in diesem Kontext beschreibt ein *gerades* Prisma als geometrischen Körper mit einer beliebigen Grundfläche. Diese wird entlang einer senkrechten Geraden verschoben, spannt somit einen Körper auf, und bildet die Deckelfläche des Prismas. Der Abstand dieser beiden Flächen ist die Prismenhöhe und beide Flächen liegen in der $x_w y_w$-Ebene des Weltkoordinatensystems. Alle zwischen den Flächen aufgespannten Seitenkanten sind parallel und haben die selbe Länge.

Ein Blob mit einer Vertikalausdehnung von genau *einer* Zelle kann besonders gut durch solch ein Prisma dargestellt werden, da er mit seiner Grundfläche bereits die Form eines Prismas der Höhe einer Zelle beschreibt. Blobs beliebiger Höhe, also aus mehreren Schei-

ben bestehend, können in mehrere solcher Teilprismen der Höhe 1 zerlegt werden. Für jedes Teilprisma eines Blobs wird dessen Grundfläche mit Hilfe eines Konturfindungsalgorithmus im Blob-Array berechnet. Ein Blob mit einer Höhe von n Voxeln wird so vorerst durch n Prismen der Höhe 1 dargestellt. Abbildung 3.8 veranschaulicht die Modellierung von Blobs als Prismen. Es liegt nahe, angrenzende Prismen gleicher Grundfläche zu

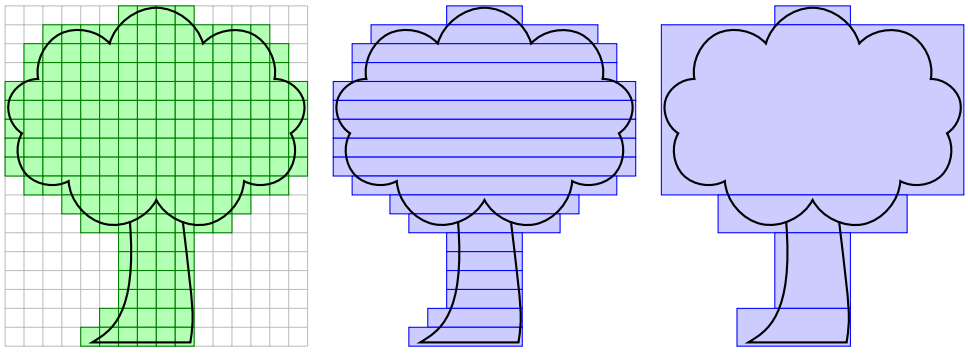


Abbildung 3.8: 2D-Seitenansicht eines Blobs (a), Auflösen der Gitterstruktur zu Teilprismen (b), Zusammenfassen ähnlicher Prismen (c) (vgl. Andert 2011).

einem Prisma zusammenzufassen. Darüber hinaus beschreibt Andert ebenfalls eine Gruppierung *ähnlicher* Prismen, wobei ihre „Ähnlichkeit“ durch entsprechende Schwellwerte bei der Prismengenerierung festgelegt wird. Auf diese Weise können nicht nur Hindernisse mit gänzlich geraden Wänden, wie beispielsweise einfache Gebäude, als Prismen dargestellt werden, sondern auch komplexere Objekte mit schrägen Außenflächen.

Die generierten Prismen sind hinsichtlich des Speicherplatzbedarfs „kompakt“, da sie sich mit den Punkten ihrer Grundflächen und der Höhe bereits vollständig beschreiben lassen. Für die anschließende Kollisionsvermeidung werden zu jedem 3D-Prisma die Polygone seiner Teilflächen berechnet und an den Bordrechner über das Netzwerk verschickt.

3.3.6 Zusammenfassung

Zusammenfassend werden die einzelnen Prozessierungsschritte, von der Datenaufnahme im Belegtheitsgitter bis zur Ausgabe der Hindernisse als Prismen, in Abbildung 3.9 auf Seite 41 als Ablaufplan dargestellt. Rote Elemente kennzeichnen dabei die Ein- und Ausgabe von Daten. Graue Elemente beschreiben Operationen auf Ebene des Belegtheitsgitters,

grüne Elemente beziehen sich auf Blobs und Prismen-bezogene Prozesse sind blau dargestellt.

Für jede Sensorposition wird geprüft, ob das gesamte Belegtheitsgitter der acht Schnittzonen weiterhin die lokale Zone umhüllt. Falls dies zutrifft, können direkt Messdaten ins Gitter der lokalen Zone rasterisiert und die Werte anschließend ins Gesamtgitter zur Aktualisierung übertragen werden. Liegt die Lokale Zone jedoch außerhalb des Gesamtgitters, werden Teilgitter in neuen Zonen erzeugt und gegebenenfalls Blobs reintegriert, falls diese in den neu akquirierten Zonen liegen. Sind die Messdaten eingefügt, können Blobs aus dem Gesamtgitter extrahiert werden. Diese liegen zunächst in kleineren individuellen Arrays vor und werden vor der Umwandlung in Prismen an den Zonengrenzen zusammengefasst.

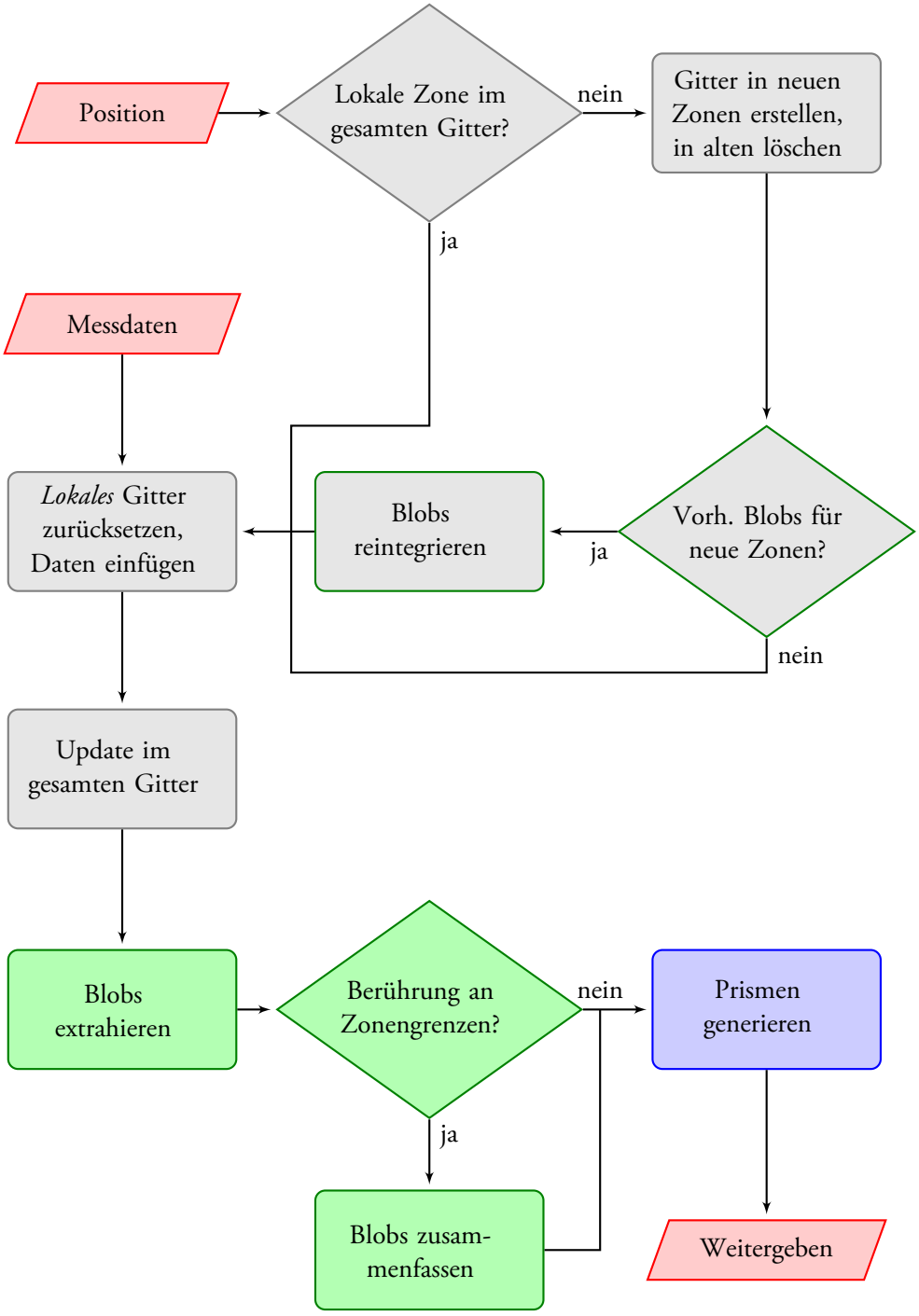


Abbildung 3.9: Prozessierungsschritte im Array-Modell.

3.4 Optimierung mittels Octree-Datenstruktur

In diesem Abschnitt wird das beschriebene Array-basierte Weltmodell von Andert (2011) um die Verwendung einer Octree-Datenstruktur erweitert. Das Grundkonzept wird beibehalten, indem Messdaten ins Belegtheitsgitter eingetragen und Hindernisse auf ähnliche Weise daraus extrahiert werden. Jedoch wird hier das Belegtheitsgitter durch einen Octree und nicht durch einzelne Arrays realisiert. Im Zuge dessen wird die Blob-Extraktion neu gestaltet, da sich Operationen auf einem Octree von Array-Operationen unterscheiden. Analysen im Vorfeld zeigten einen auffällig hohen Speicherbedarf von Blobs auf Array-Basis. Es wird ein optimiertes Speicherkonzept vorgestellt, welches die Grundeigenschaften der Blobs beibehält, so dass sie anschließend auf gleiche Weise zu Prismen vereinfacht werden können. Die eigentliche Prismengenerierung bedarf nur einer kleinen Anpassung und bleibt weitestgehend äquivalent zur im Abschnitt 3.3.5 beschriebenen Vorgehensweise.

3.4.1 Octree als stochastisches 3D-Belegtheitsgitter

Bäume ermöglichen eine hierarchische Untergliederung von Daten. Mit einem Binärbaum (2-Baum) wie in Abbildung 2.5 auf Seite 19 mit maximal zwei Kindern pro Knoten lassen sich eindimensionale Daten darstellen. In der Computergrafik, Robotik und Kartografie kommen oft Quadtrees (4-Bäume) für zweidimensionale und *Octrees* (8-Bäume) für dreidimensionale Daten zum Einsatz (Angel 2006, Kapitel 10.10.3; Wurm, Hornung u. a. 2010; Longley u. a. 2005, Kapitel 10.7.2). In einem Quadtree gibt es vier, in einem Octree entsprechend acht Kinder pro inneren Knoten. Die Kernidee des Baumkonzepts ist eine rekursive Dekomposition des Raums in *gleiche* Teile, wie vereinfacht am Beispiel eines Octrees in Abbildung 3.10 zu sehen ist: Die Wurzel repräsentiert einen würfelförmigen Raum mit festgelegter Ausdehnung. Jedes Kind stellt ein kubisches Teilvolumen mit halbierter Kantenlänge seines Elternknotens dar, also einen Oktanten. Diese rekursive Zerteilung kann prinzipiell beliebig oft erfolgen und wird durch die maximale Höhe eines Baums begrenzt. Dadurch ist die Rasterung eines Raums in verschiedenen Auflösungsstufen möglich. Die Blätter auf tiefster Ebene im Baum stellen dann die höchste/feinste räumliche Auflösung dar, welche mit jeder Ebene nach oben niedriger/gröber wird. Dieses Konzept ist für zweidimensionale Daten in Quadtrees identisch, wobei jeder Knoten nicht ein Volumen sondern ein Flächenstück repräsentiert.

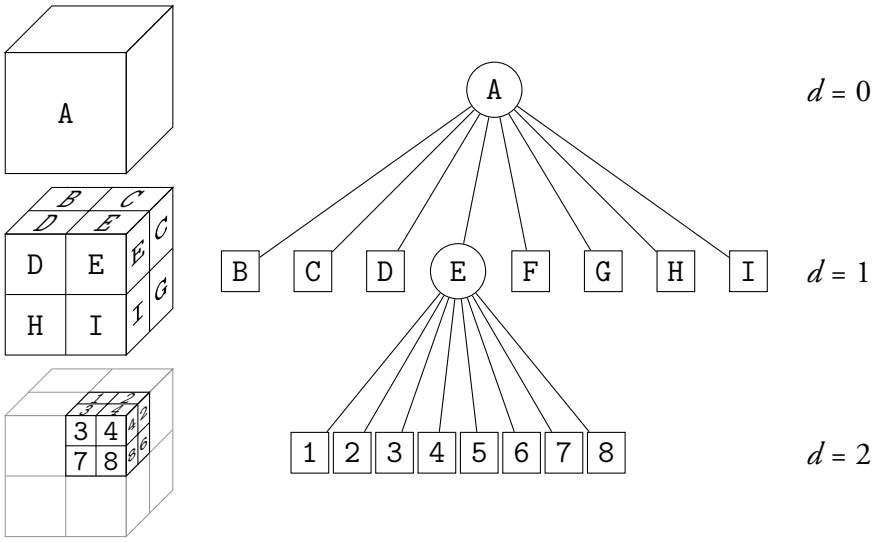


Abbildung 3.10: Teilweise rekursive Dekomposition eines Raums A durch einen Octree der Höhe 2.

Die Ordnungsstruktur solcher Bäume ist durch die hierarchische Raumgliederung gegeben. Jeder Knoten entspricht in physikalischer Hinsicht einem eindeutigen Raumsegment und hat im Baum ebenfalls eine eindeutige Identifikation: Beispielsweise einen Schlüssel (key, ID) bestehend aus allen Schlüsseln seiner Elternknoten.

Bei den beschriebenen Baumstrukturen mit gleichmäßiger Raumteilung sind die drei Parameter *räumliche Ausdehnung*, *Gitterauflösung* und *Baumhöhe* nicht unabhängig voneinander skalierbar. Die maximale räumliche Ausdehnung, in diesem Fall die Seitenlänge l der vom Wurzelknoten dargestellten Region, ergibt sich aus der Zellgröße c auf unterster Ebene und der Baumhöhe h nach

$$l = c \cdot 2^h. \tag{3.12}$$

Ein Octree mit maximaler Auflösung von 0,5 m und der Höhe 7 hat somit eine maximale räumliche Ausdehnung von 64 m × 64 m × 64 m.

Die Baumstruktur ermöglicht eine bedarfsabhängige Verwendung von Knoten. Im zweidimensionalen Fall veranschaulicht dies Abbildung 3.11 auf Seite 44 für eine Beispielregion mit innenliegendem Objekt. Wo ein Array 4 × 4 = 16 Zellen benötigt, um die Region inklusive des Objekts in der gewählten Rasterung darzustellen (b), sind für die Darstellung mittels Quadtree nur vier Knoten nötig (c)–(d). Dabei können Blätter mit *identischer* In-

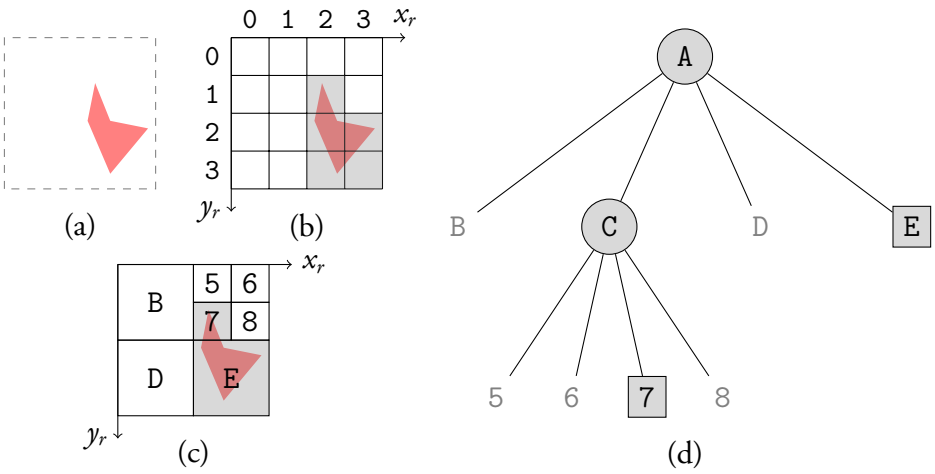


Abbildung 3.11: Region mit Beispielobjekt (a), diskrete Arraydarstellung der Region (b), hierarchische Teilung (c) auf Grundlage des Quadtrees (d).

formation als Speichervorteil zu einem Blatt auf höherer Ebene zusammengefasst werden, wie es bei Blatt `E` der Fall ist. Falls erwünscht, kann Blatt `E` jeder Zeit wieder zu vier Blättern auf niedrigerer Ebene erweitert werden, sofern es die definierte Maximalhöhe des Baums erlaubt. Leere Bereiche einer Region (`B`, `D`, `5`, `6`, `8`) entsprechen uninitialisierten Knoten im Baum, die durch Nullzeiger realisiert werden. Für einen Nullzeiger wird nur der Speicherplatz des Zeigers selbst benötigt und somit kein weiterer Speicher für Knoten reserviert.

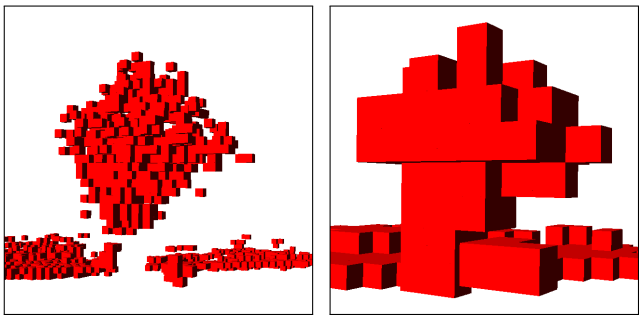
Ein theoretischer Nachteil bei Quad- und Octrees besteht in der Abbildung *heterogener* Information: Für die Region im obigen Beispiel können Knoten nicht wie Knoten `E` zusammengefasst werden, wenn sie unterschiedliche Information beinhalten. Ein Beispiel dafür können verschiedene Belegtheitswahrscheinlichkeiten im Gegensatz zu binärer Belegtheit sein. Weiterhin besteht der Baum bei feingliedriger Darstellung der *gesamten* Region aus 21 Knoten (nach Gleichung 2.10), wobei das Array weiterhin mit 16 Zellen auskommt. Für die Verwendung als stochastisches 3D-Belegtheitsgitter überwiegen die Vorteile von Octrees, da im Realfall von hauptsächlich unbekanntem Raum ausgegangen werden kann, für den im Baum keinerlei Knoten angelegt werden. Lediglich für vereinzelt gemessene Hindernisse und den gemessenen Freiraum werden Knoten erstellt. So enthält der Baum nur einen Bruchteil der maximal möglichen Knoten, an dessen Stelle ein Array ebenfalls Speicherzellen für den unbekannten Raum belegen würde. Hunter (1978) formuliert dazu

das generelle *Quadtree Complexity Theorem*, wonach im Regelfall die Anzahl der Knoten eines Quadtrees proportional zum Umfang seiner Region ist und nicht proportional zum Flächeninhalt, wie die Anzahl der Zellen bei Verwendung eines Arrays. Dieser zweidimensionale Fall trifft auch auf Octrees in 3D oder allgemein auf *n*-dimensionale Strukturen zu (Samet 2006, Kapitel 2.2.2.2). Somit ist mittels solcher hierarchischer Datenstrukturen effektiv eine Reduzierung der Dimension eines Problems möglich (Umfang im Gegensatz zum Flächeninhalt).

Die freie OctoMap-Bibliothek

Im Zuge der Octree-Umstellung des Array-basierten Weltmodells wurde auf die Octree-Implementierung *OctoMap* der Freiburger Universität um Prof. Dr. Wolfram Burgard zurückgegriffen (Wurm, Hornung u. a. 2010). OctoMap wurde zur probabilistischen 3D-Modellierung entwickelt, mit dem Fokus auf eine besonders speicherfreundliche Umsetzung. Die Bibliothek ist momentan unter der freien BSD-Lizenz veröffentlicht und stellt einen Octree mit den im Vorfeld beschriebenen Eigenschaften bereit. Viele Funktionen zum Einfügen von Messdaten und zur Traversierung des Baums sind bereits vorhanden und wurden im Kontext dieser Diplomarbeit teilweise erweitert.

Wie eingangs in Abschnitt 3.4 erwähnt, diskretisiert der Octree, abhängig von seiner Höhe, den Raum in abgestuften Auflösungen. Abbildung 3.12 zeigt ein gescanntes Hindernis, in diesem Fall einen Busch, mit maximaler Auflösung von 0,125 m auf unterster Octree-Ebene. Wird der Octree zwei Ebenen darüber „abgeschnitten“, liegt das Objekt



(a) (b)

Abbildung 3.12: Skalierung im Octree: Ein Busch auf unterster Octree-Ebene mit Zellgröße 0,125 m (a) und zwei Ebenen darüber mit 0,5 m (b).

entsprechend in 0,5 m-Zellgröße vor. Der mit dem Octree dargestellte Freiraum ist in Abbildung 3.13 zu sehen. Belegte Zellen sind rot, der Freiraum ist grün. Der Scanner ist am Hubschrauber im Winkel von 15° nach unten montiert. An den schräg verlaufenden Strahlen der freien Zellen ist erkennbar, dass der Hubschrauber von rechts nach links über das Haus geflogen ist.

Die Bibliothek ist sehr flexibel und bietet die Möglichkeit, eigene Knotentypen für den Octree zu definieren. So können nicht nur Knoten mit ihrer Belegtheitswahrscheinlichkeit gespeichert werden, sondern beispielsweise Zeitstempel sowie weitere semantische oder statistische Merkmale hinzugefügt werden. Ähnliche Octree-Implementierungen sind in der ebenfalls freien Point Cloud Library (Rusu u. a. 2011) zu finden, mit teilweise spezifischeren Anwendungen als nur der Verwendung für probabilistische Belegtheitsgitter.

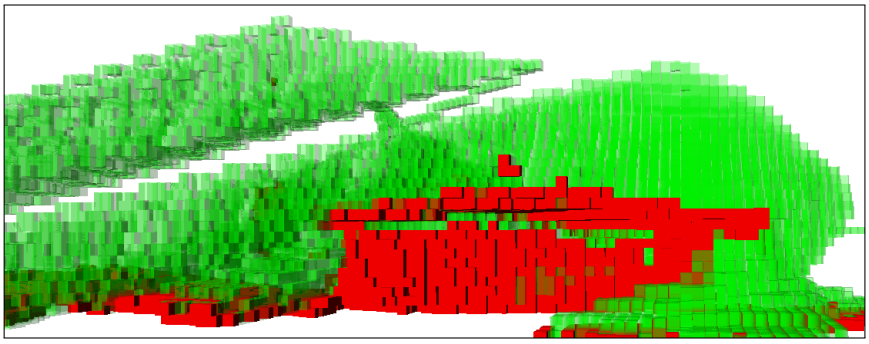


Abbildung 3.13: Gescanntes Haus mit grünen Zellen als Freiraum.

3.4.2 Zonenaufteilung im Octree-Modell

Mit der Octree-Umstellung des Weltmodells wurde ein abweichendes Zonenkonzept entwickelt. Auf einer Tiefe von 1 entsprechen die Oktanten des Octrees bereits den acht Zonen, die von der lokalen Zone um den Sensor geschnitten werden (Knoten `B`–`I` in Abb. 3.10, S. 43). Somit werden nicht acht Octrees für acht separate Belegtheitsgitter erstellt, sondern nur ein entsprechend großes Gitter als Octree. Das Einfügen neuer Scandaten wird direkt in diesem Octree vollzogen und nimmt keinen Umweg mehr über ein lokales Gitter.

Die lokale Zone um den Sensor existiert lediglich, damit der Octree beim Überschreiten seiner Grenzen korrekt an der nächsten Position wieder aufgebaut werden kann. Dabei

wird beim Verlassen des aktuellen Octrees ein komplett neuer Baum erstellt, der sich teilweise mit dem alten überlappt. Jeder Octree bekommt eine Kennung abhängig von seiner Position im Raum und ist dementsprechend nur für sein Gebiert gültig. Wird ein Octree an einer Position aufgebaut, an der zuvor schon ein Octree existierte, trägt er wieder die gleiche Kennung. In Abbildung 3.14 ist ein ungültiger Octree nach dem Zonenwechsel gepunktet dargestellt. In der aktuellen Implementierung müssen in den überlappenden Zo-

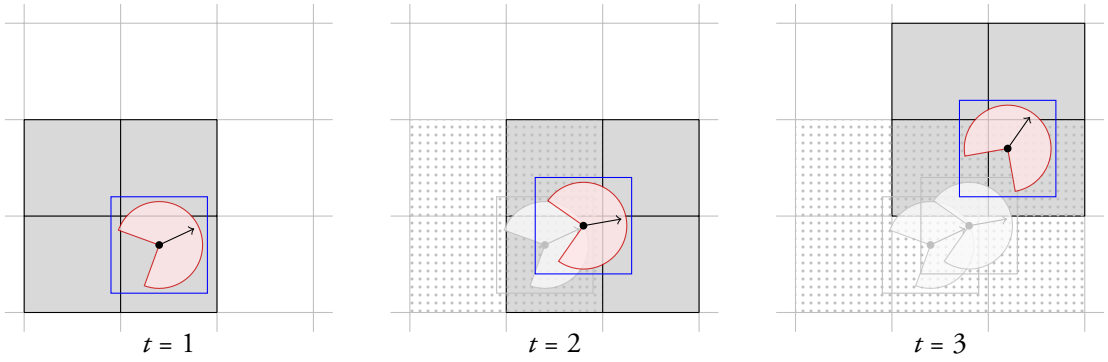


Abbildung 3.14: Draufsicht: Octree (grau) als Belegtheitsgitter zu unterschiedlichen Zeitpunkten *t*. Die lokale Zone (blau) ist dem Sensorsichtfeld (rot) angepasst. Gelöschter Octree nach Verschiebung ist gepunktet.

nen die Daten entsprechend aus dem ungültig gewordenen in den neuen Octree übertragen werden. Der dafür nötige Kopiervorgang zieht bei steigender Knotenzahl Laufzeiteinbußen mit sich. In Kapitel 4.4 wird dieser Aspekt näher untersucht und in Kapitel 5.1.2 eine alternative Umsetzung vorgeschlagen.

Ein Problem bei diesem Zonenkonzept sind die zu großen Zonen, also ein unnötig großes Belegtheitsgitter, da die OctoMap-Bibliothek die Höhe des Octrees auf genau 16 festlegt. Besteht beispielsweise die Anforderung Daten in einer Auflösung von 0,5 Metern aufzuzeichnen, besitzt das gesamte Gitter eine Seitenlänge von $0,5 \cdot 2^{16} = 32\,768$ m, ein Oktant und somit die lokale Zone also eine Seitenlänge von 16 384 m (vgl. Formel 3.12). Für die Sensordatenintegration hat dies zunächst keine größeren Nachteile. Allerdings wird bei der Hindernisextraktion der Octree komplett traversiert, wobei die große Baumhöhe ein Überprüfen von unnötig vielen inneren Knoten mit sich zieht. Dieser Aspekt wirkt sich auf die Leistung aus und wird in Kapitel 4.4 beim Vergleich unterschiedlicher Octree-Größen untersucht. OctoMap wurde in diesem Zuge erweitert, um Octrees mit einer Höhe von

weniger als 16 erzeugen zu können. Die interne Schlüsselberechnung der Knoten arbeitet auf einem 16 bit-Datentyp, weswegen höhere Octrees momentan nicht möglich sind.

3.4.3 Segmentierung von Zellen im Octree

Im Octree-Ansatz werden Zellen für die Extraktion der Hindernisse ebenfalls durch einen Floodfill segmentiert (vgl. Abschnitt 3.3.4). Die Nachbarfindung im Baum gestaltet sich komplexer als in einem Array, da physikalisch benachbarte Knoten nicht zwangsweise im Baum benachbart sind, wie Abbildung 3.15 an Knoten 4 und 7 verdeutlicht. Ferner hat ein

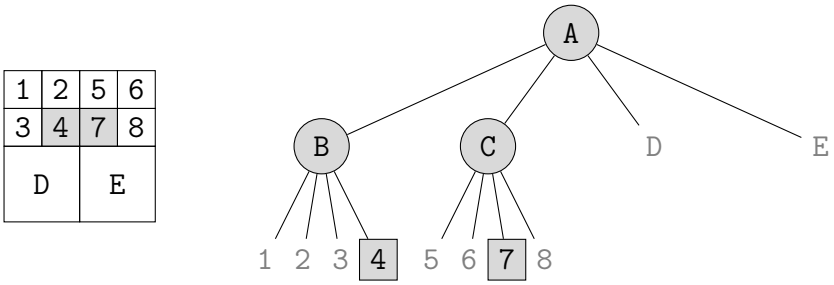


Abbildung 3.15: Zellnachbarn im Quadtree.

Knotenzugriff im Baum logarithmische Komplexität (siehe Abschnitt 2.3.2). Daher wird der Floodfill nicht direkt im Octree angewandt, sondern ausweichend eine Hash-Tabelle verwendet, die einen konstanten Zugriff auf Elemente ermöglicht. Dazu wird der Octree traversiert und die eindeutigen Schlüssel der *belegten* Blätter werden in die Hash-Tabelle eingetragen. Der in Abschnitt 3.4 beschriebene „dimensionsreduzierende Vorteil“ von Octrees kommt hier zur Geltung: Ein 3D-Array komplett zu durchlaufen schließt das Prüfen von „unbekanntem Raum“ mit ein. Der Octree hingegen wird über seine Knoten traversiert und enthält für unbekannten Raum keine Knoten. Zwar werden auch alle inneren Knoten überprüft, obwohl für die Hindernisextraktion nur Interesse an belegten Blättern besteht, jedoch ist die Gesamtknotenzahl in der Regel kleiner als alle Zellen eines Arrays mit selber Ausdehnung und Auflösung.

Ein 3D-Floodfill-Algorithmus von Feng u. a. (1998) wurde so erweitert, dass die in der Hash-Tabelle vorliegenden Knoten über ihren eindeutigen Schlüssel gruppiert werden können. Der Schlüssel eines physikalischen Nachbarknotens in den Richtungen Nord, Süd, Ost, West, oben oder unten (6er-Nachbarschaft), wird dafür über eine im Vorfeld angeleg-

te Lookup-Tabelle (LUT) ermittelt. Die LUT beinhaltet das nötige Inkrement/Dekrement des Schlüsselwertes, um den Schlüssel eines Nachbarn in gewünschter Richtung zu berechnen. In konstanter Zeit kann nun innerhalb der Hash-Tabelle auf einen etwaigen Nachbarn geprüft und dieser zu einem Blob gruppiert werden.

Mit der Skalierbarkeit des Octrees können Blobs in verschiedenen Detailgraden extrahiert werden, indem der Baum auf gewünschter Tiefe abgeschnitten wird. Ein extrahierter Blob liegt nicht wie beim Array-Modell im eigenen Array vor, sondern in einer Hash-Tabelle, welche die Schlüssel der zu ihm gruppierten Knoten und deren globale Koordinaten beinhaltet. Diese kompakte Tabelle bringt einen weiteren großen Speichervorteil, weil Speicher nur für die tatsächlich zum Blob gruppierten Knoten verwendet wird (siehe Abbildung 3.16).

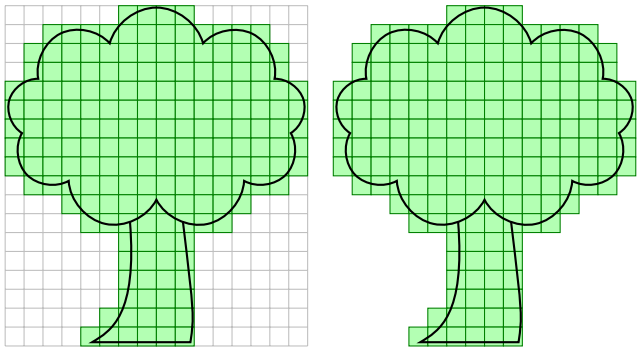


Abbildung 3.16: Blob im Array- (a) und im Octree-Modell (b).

Die anknüpfende Prismenerstellung arbeitet zum Berechnen der Grundfläche eines Prismas auf Zellebene, wie in Abschnitt 3.3.5 beschrieben. Ein Blob als Hash-Tabelle kennt seine Auflösung und seine räumliche Begrenzung in allen drei Dimensionen. So kann dieser Blob on-the-fly bei Bedarf in eine Array-Darstellung überführt werden und die Prismenerstellung bedarf sonst keiner weiteren Anpassung.

3.4.4 Reintegration bestehender Hindernisse

Ähnlich zum Array-Modell (Abschnitt 3.3.4) können bereits extrahierte Blobs wieder ins Octree-basierte Belegtheitsgitter eingefügt werden, falls der Octree an einer Position aufge-

baut wird, an der bereits im Vorfeld Blobs extrahiert wurden. Zuvor kartierte Hindernisse aus bereits nicht mehr vorhandenen Octrees können also weiterhin bei der Integration neuer Sensordaten im Gitter berücksichtigt werden.

Da Zonenüberschneidungen bei den Octrees vorkommen, kann ein Blob für verschiedene Octrees (maximal acht in 3D) gültig sein und wird im jeweiligen Baum durch andere Knoten dargestellt (Abbildung 3.17). Jeder Blob kennt daher den Octree aus dem er extrahiert wurde über dessen Baum-Kennung (siehe Abschnitt 3.4.2). Die Schlüssel der im Blob gespeicherten Knoten können umgerechnet werden, um den selben Blob jeder Zeit in unterschiedlichen Bäumen darstellen zu können. Ein aus Baum `A1` extrahierter Blob wird somit auch für Baum `A2` gültig. Bei der Reintegration werden alle Knotenschlüssel aus dem Blob gelesen und an deren Stelle im Baum neue belegte Blätter erstellt. Informationen über zuvor kartierten Freiraum können in diesem Fall nicht wiederhergestellt werden.

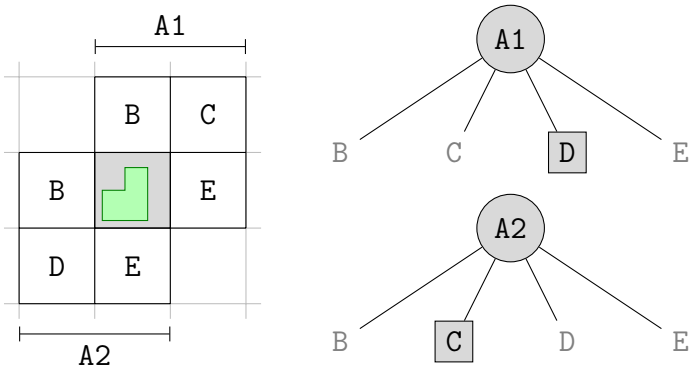


Abbildung 3.17: Ein Blob (grün) gültig in Quadtree `A1` und `A2`.

3.4.5 Zusammenfassung

Zusammenfassend werden die einzelnen Prozessierungsschritte des optimierten Modells in Abbildung 3.18 auf Seite 52 als Ablaufplan dargestellt. Die Datenaufnahme geschieht bei dieser Variante direkt im gesamten Octree-basierten Belegtheitsgitter, ohne den Umweg über ein lokales Gitter. Das Grundprinzip wird beibehalten, und die Prismenerstellung arbeitet weiterhin wie im Array-Modell beschrieben (Abschnitt 3.3.5). Rote Elemente kennzeichnen die Ein- und Ausgabe von Daten. Graue Elemente beschreiben Operationen auf Ebene des Octrees, grüne Elemente stehen in Bezug zu Blobs und blaue im Bezug zu

Prismen.

Theoretisch besteht ein Leistungsvorteil beim Octree, da die Komplexität der Operationen nicht von der Zellenzahl sondern von der Knotenzahl abhängt. Dieser Vorteil wird allerdings geringer, je mehr Zellen im Gitter mit Information gefüllt werden. Weiterhin fällt ein Kopieren von Zellen beim Zonenübergang an, welches beim Array-Modell ausbleibt.

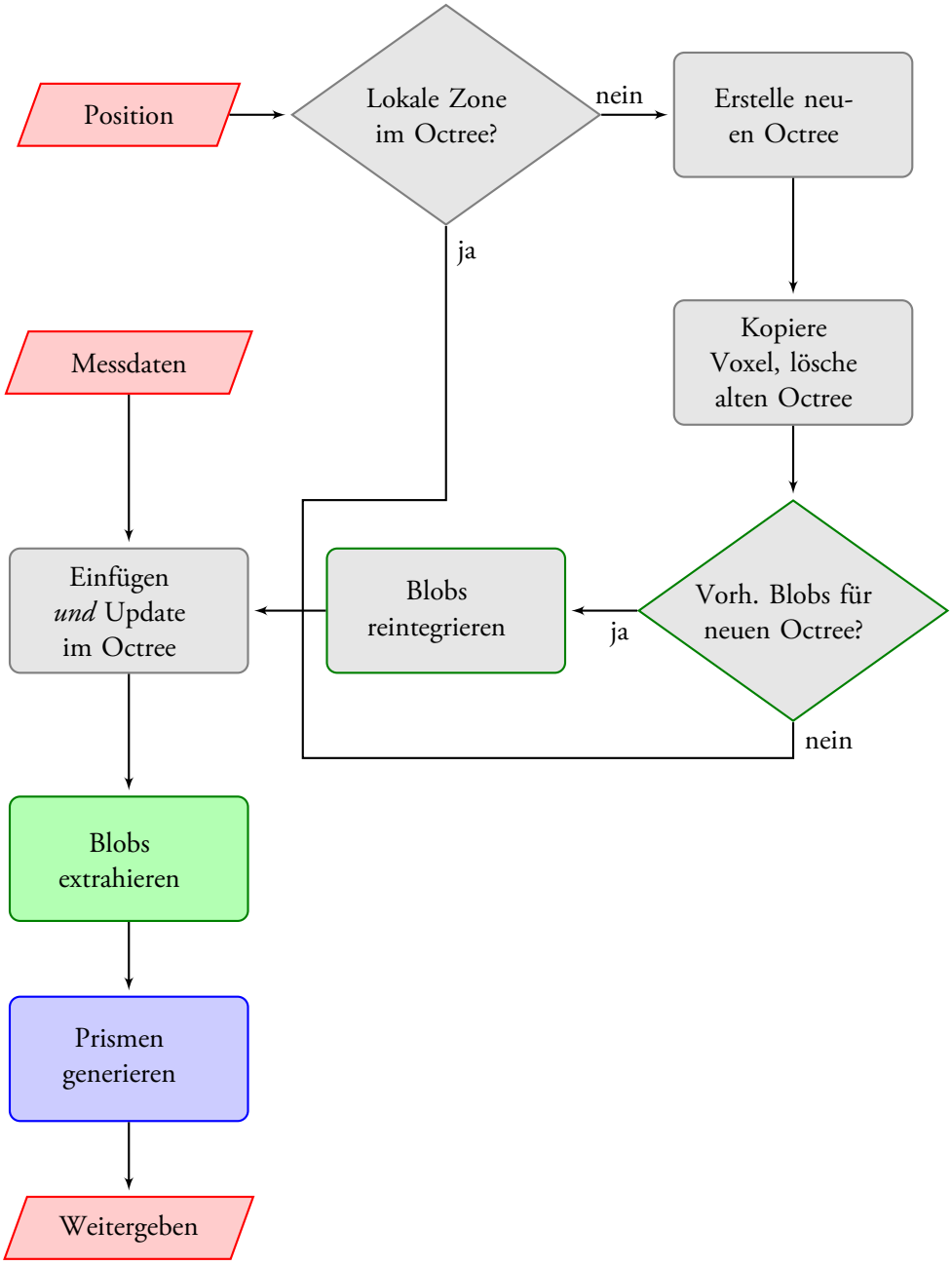


Abbildung 3.18: Prozessierungsschritte im Octree-Modell.

4 Analyse beider Weltmodelle

In diesem Kapitel wird ein Vergleich hinsichtlich der Performance zwischen dem erweiterten Array-Modell und dem neu vorgestellten Octree-Modell geführt. Dabei wird die Leistung anhand von Bildraten und durchschnittlicher Laufzeiten verglichen, sowie der theoretische Speicherbedarf beider Modelle gegenüber gestellt. Bei ähnlichen Voraussetzungen wird eine verbesserte Leistung des Octree-Modells in allen Punkten erwartet. Abschnitt 4.1 zur Testumgebung beschreibt das verwendete Testsystem und die zur Auswertung gewählten Szenarien. Die Gesamtleistungen beider Weltmodellansätze werden mit vergleichbaren Parametern anhand dieser Szenarien untersucht und die Ergebnisse direkt diskutiert. Anhand *eines* Szenarios erfolgt zusätzlich eine Aufschlüsselung der Leistung einzelner Prozessierungsschritte in beiden Modellen. Im Hinblick auf die Verwendung anderer Sensoren mit beispielsweise größeren Messentfernungen, die größere lokale Zonen mit sich ziehen, wird abschließend das Verhalten des Octree-basierten Modells mit verschiedenen Baumgrößen untersucht.

4.1 Testumgebung

Um den Testablauf zu beschleunigen, wurde ein leistungsstärkeres System als der Bildverarbeitungsrechner des ARTIS gewählt (vgl. Kapitel 1.2). Tabelle 4.1 auf Seite 54 zeigt die wichtigsten Eckdaten des Testsystems. Die Testanwendung ist für 32 bit entwickelt und unterstützt prinzipiell Multithreading, was beispielsweise von extern eingebundenen Bibliotheken genutzt wird. Für die Bildverarbeitungsfilter zur Prozessierung der Daten ist momentan keine Unterstützung der Nebenläufigkeit implementiert.

Tabelle 4.1: Das Testsystem.

Prozessor	Intel Xeon E5645
Anzahl Kerne	6
Anzahl Threads	12
Taktrate	2,4 GHz
L3-Cache pro Kern	2 MiB
Arbeitsspeicher	6 GiB
Betriebssystem	Windows 7, 64 bit
Testanwendung	32 bit

4.1.1 Testszenarien

Die Auswertung wird am Beispiel von Daten aus zwei verschiedenen Flugeinsätzen und einem künstlichen Szenario durchgeführt.

I. Szenario „Tor“ basiert auf dem Datensatz von einem Tordurchflug. Ein Tor von 6 m × 6 m mit seitlich angebrachten Fahnen steht aufrecht auf einem Feld. Der Hubschrauber fliegt hindurch und zeichnet dabei mit einem 18° nach unten geneigten Laserscanner den Boden sowie einige Merkmale des Tors auf. Nach dem ersten Durchflug dreht der Hubschrauber eine Schleife und beendet die Kartierung.

II. Szenario „Gebäude“ basiert auf einem Datensatz bei dem ein halboffenes Gebäude in zwei Überflügen kartiert wird. Der Scanner ist im 15°-Winkel nach unten ausgerichtet und nimmt den Bodenbereich sowie weitere Objekte um das Gebäude herum mit auf. Durch die Fenster erfolgt ebenfalls eine partielle Kartierung vom Inneren des Gebäudes. Nach dem ersten Überflug dreht der Hubschrauber eine Schleife und kehrt für eine anschließende Kartierung von der anderen Seite zurück.

III. Szenario „Ausdehnung“ testet das Verhalten beider Weltmodellansätze bei großer räumlicher Ausdehnung. Momentane Flugeinsätze sind lediglich auf wenige Minuten beschränkt was die räumliche Ausdehnung automatisch begrenzt. Dieses künstliche Szenario simuliert eine hohe Ausdehnung entlang der x_w-Achse. Alle 2,5 Meter wird ein synthetischer, gerader Laserscan in die $x_w y_w$-Ebene des Weltmodells eingetragen. Die Messpunkte des Scans liegen dabei auf einer 10 m langen Geraden. Diese resul-

tierende Zeile wird 10 m vom simulierten Träger entfernt senkrecht nach unten ins Modell eingetragen.

Abbildung 4.1 zeigt Ausschnitte der drei Szenarien im Octree, bei einer Auflösung von 0,125 m. Informationen über die Räumliche Ausdehnung und den Datenumfang dieser

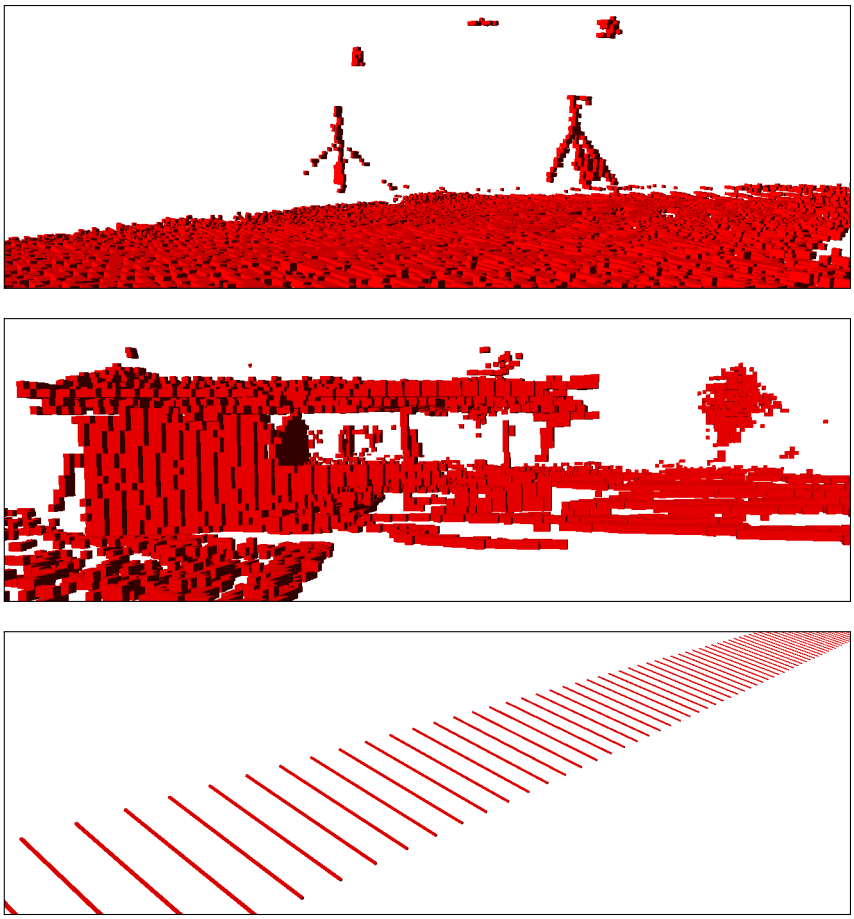


Abbildung 4.1: Testszenarien bei Zellgröße 0,125 m von oben nach unten: I Tor, II Gebäude und III Ausdehnung.

Szenarien sind aus Tabelle 4.2 auf Seite 56 zu entnehmen.

4.1.2 Freiraumbetrachtung

Die folgenden Untersuchungen werden mit einer partiellen Betrachtung des Freiraums durchgeführt. In Abbildung 4.2 auf Seite 56 sind die Unterschiede zwischen vollständiger,

Tabelle 4.2: Räumliche Ausdehnung und Datenumfang der drei Szenarien.

		Gesamtzahl der	
Szenario	Ausdehnung [m]	Laserscans	Messpunkte
I	$56 \times 82 \times 12$	2736	487 078
II	$48 \times 44 \times 13$	2623	282 848
III	$2500 \times 10 \times 1$	1000	213 000

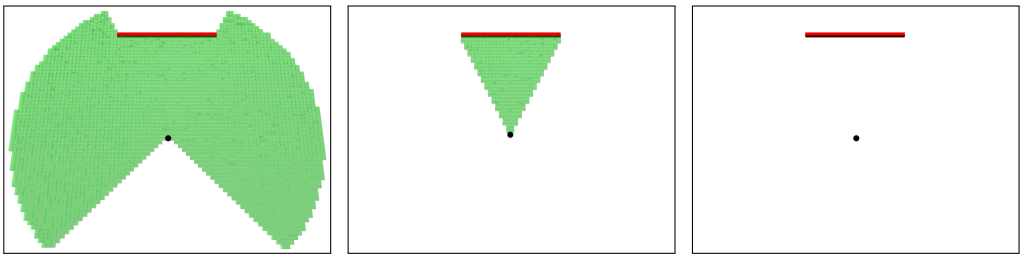


Abbildung 4.2: Vollständige, partielle und keine Freiraumbetrachtung.

partieller und keiner Freiraumbetrachtung dargestellt. Die vollständige Freiraumbetrachtung rasterisiert leere und belegte Zellen für jeden ausgesandten Strahl des Sensors. Partielle Freiraumbetrachtung rasterisiert nur freie Voxel der Strahlen, die auch einen Messpunkt liefern. Beim reinen Einfügen der Messpunkte findet keinerlei Betrachtung von freien Zellen mehr statt.

Die teilweise Freiraumbetrachtung reduziert die Datenmenge im Gitter und bietet trotzdem ein gewisses Maß an Fehlerkorrektur. Bei den beiden Realszenarien I und II kann dies zu „leeren“ Scans führen, für die keine Daten ins Gitter übertragen werden. Ein Beispiel dafür ist eine Flughöhe oder eine Ausrichtung des Trägers, bei der keine Messpunkte aufgezeichnet werden, da die gemessenen Distanzen außerhalb der Reichweite des Sensors liegen. Dieser Fall tritt bei Szenario I und II auf, während der Hubschrauber seine Schleife in größerer Höhe fliegt.

Weiterhin liefert der Scanner zwar Entfernungsmessungen auch jenseits der Einsatzreichweite, allerdings werden diese auf 30 m beschnitten, um innerhalb der Spezifikationen des Herstellers zu bleiben. Bei größeren Entfernungen und daraus resultierend abweichenden Messeigenschaften müsste anderenfalls ein komplexeres Sensormodell zum Einsatz kom-

men (näheres zum Sensormodell in Kapitel 3.2). Wie auch von Hrabar (2012) erwähnt, wird in Außenszenarien die Reichweite von 30 m allerdings kaum erreicht.

4.2 Vergleich hinsichtlich Laufzeit

Die Leistung des Array-Modells hängt stark von der Größe der verwendeten Arrays und der Zellauflösung ab und passt sich in der aktuellen Implementierung nicht dynamisch den Sensorparametern an. Im Normalfall wird die Array-Ausdehnung für den verwendeten Sensor passend gewählt, um einerseits alle Daten des Sensors integrieren zu können und andererseits kein unnötig großes Array prozessieren zu müssen.

Für den eingesetzten Laserscanner mit Reichweite von 30 m wird bei einer Rasterauflösung von einem Meter im Array-basierten Ansatz mindestens ein lokales Gitter der Dimension 61 m × 61 m × 61 m benötigt (Kapitel 3.3.2). Die Größe der lokalen Zone beim Octree-Ansatz wird zur Basis 2 ausgedrückt, womit 2^6 m = 64 m die nächstmögliche Approximation der Größe ist. Um eine Vergleichbarkeit beider Ansätze zu ermöglichen, wird hier die Größe der lokalen Zone beim Array- und Octree-Ansatz folglich mit 64 m gleich gewählt. Damit lassen sich ebenfalls Daten der von Andert (2011) verwendeten Stereokamera mit 25 m bis 30 m Reichweite erfassen. Im Folgenden wird diese Konfiguration mit Z_{64} bezeichnet.

Der Array-Ansatz erweist sich in den Tests als speicherintensiv, weshalb zusätzlich eine Untersuchung mit reduzierter *z*-Ausdehnung der lokalen Zone als Konfiguration Z_{25} stattfindet. Dadurch wird die Datenmenge durch kleinere Gitter gesenkt und die Datenprozessierung beschleunigt, was eine bessere Vergleichbarkeit mit dem Octree-Modell erlaubt. Bei allen drei Testszenarien misst der Hubschrauber aus einer Höhe von weniger als 20 Metern, so ist nicht zwingend eine *z*-Ausdehnung des Arrays von 64 m nötig. Der Träger wird innerhalb der lokalen Zone bei Z_{25} nicht mittig auf der *z*-Achse ausgerichtet, da mit einer Sensorausrichtung „nach unten“ nicht beabsichtigt wird, Hindernisse „über dem Träger“ zu erfassen. Mit diesen Voraussetzungen kann eine lokale Zonengröße von 25 m in *z*-Richtung gewählt und der Träger bei 20 m „im oberen Viertel“ positioniert werden. Im Anwendungsfall spontaner Einsätze in unbekanntem Gelände darf die Wahl des Gitters allerdings *nicht* vom Szenario abhängig sein, da dies nur durch Postprocessing ermöglicht wird.

Tabelle 4.3 stellt die Leistungen beider Ansätze anhand von durchschnittlichen Bildwiederholfrequenzen für die drei beschriebenen Szenarien bei unterschiedlichen Zellgrößen der Belegtheitsgitter dar. Geschätzt kann das Testsystem als drei bis vier Mal schneller als der Bildverarbeitungsrechner des ARTIS angesehen werden. Die extrem hohen Bildraten bei den Realszenarios I und II sind auf die *partielle* Freiraumbetrachtung zurückzuführen, wo viele Scans ohne Messpunkte ignoriert werden (siehe Abbildung 4.2).

Tabelle 4.3: Durchschnittlicher Bildfrequenzen im Array- und Octree-Ansatz anhand der Testszenarien mit variabler Zellgröße.

			Bildfrequenz [Hz] bei Zellgröße				
Szen.	Weltmodell		2 m	1 m	0,5 m	0,25 m	0,125 m
I	Array	Z_{64}	480	66	4	–	–
	Array	Z_{25}	720	131	10	–	–
	Octree	Z_{64}	1190	558	215	58	13
II	Array	Z_{64}	535	67	4	–	–
	Array	Z_{25}	971	144	11	–	–
	Octree	Z_{64}	1639	771	341	120	30
III	Array	Z_{64}	294	56	–	–	–
	Array	Z_{25}	400	92	19	–	–
	Octree	Z_{64}	323	132	86	44	19

Die Größe der lokalen Zone ist für Konfiguration Z_{64} mit 64 m × 64 m × 64 m festgelegt, was eine Gesamtgröße des Belegtheitsgitters von 128 m × 128 m × 128 m für den Array- sowie Octree-Ansatz ergibt. Ergänzend wird wie beschrieben der Array-Ansatz in Konfiguration Z_{25} mit einem gesamten Belegtheitsgitters von 128 m × 128 m × 50 m untersucht. Bei festgelegter räumlicher Ausdehnung des Gitters ändert sich mit zunehmender Auflösung entsprechend die Gesamtzahl der Zellen innerhalb des Gitters, was in zunehmender Laufzeit resultiert. Im Bezug auf die Gittergrößen beider Konfigurationen hat das Gitter in Konfiguration Z_{25} mit $(128^2 \cdot 50)/128^3 \approx 0.39$ etwa 39 % der Zellenanzahl vom Gitter in Konfiguration Z_{64}.

Tabelle 4.3 zeigt eine deutliche Erhöhung der Bildrate für reduzierte z-Ausdehnung der Belegtheitsgitter im Array-Ansatz. Testdurchläufe mit feinerer räumlicher Auflösung als 0,5 m waren auf Grundlage der Arrays für kein Szenario möglich, da die Speicherverwal-

tung des Betriebssystems die Testanwendung ab einem Speicherbedarf größer als 2 GiB zum Absturz brachte. Näheres zu den Gründen in Abschnitt 4.3.2.

Bei Szenario III ist das Array-basierte Weltmodell für Konfiguration Z_{25} etwas schneller als der Octree. Ein Grund für die ansonsten relativ niedrigen Bildraten des Array-Ansatzes, im Vergleich zu den Auswertungen in Anderts Arbeit, ist sein noch kleiner gewähltes Belegtheitsgitter mit einer *z*-Ausdehnung von lediglich 16 Metern (vgl. Andert 2011, S. 103f.). Weiterhin werden bei seinen Auswertungen im Vorfeld einige Messpunkte gefiltert und daraus eine Bodenebene ermittelt (vgl. Andert 2011, Kapitel 3.3.5). Diese Punkte fließen entsprechend nicht ins Belegtheitsgitter und somit auch nicht in die nachfolgenden Berechnungen zur 3D-Hindernisgenerierung ein.

4.2.1 Aufschlüsselung der Laufzeit

Es folgt die Aufschlüsselung der Laufzeit auf die einzelnen Komponenten des Weltmodells für einen Szenariodurchlauf des Gebäude-Szenarios II. Es liefert die komplexesten Hindernisse und im Vergleich mit den anderen Szenarien nimmt hier die Extraktion der Blobs die meiste Zeit in Anspruch. Die Aufschlüsselung wird für eine Zellgröße von 0,5 Metern mit den genannten Konfigurationen Z_{64} und Z_{25} durchgeführt. Eine Auflösung von 0,5 Metern bietet einen guten Kompromiss zwischen Performance der Modellierung und Detailgrad der Hindernisse.

Aus den Ergebnissen in Tabelle 4.4 auf Seite 60 wird ersichtlich, dass der Hauptteil der Laufzeit (beim Array-Ansatz mit über 90 %) der Blob-Extraktion zuzuschreiben ist. Operationen auf dem Array-basierten Belegtheitsgitter selbst geraten stark in den Hintergrund und nehmen mit 6,1 s für Konfiguration Z_{25} nur knapp dreifache Zeit in Anspruch wie für das Octree-basierte Belegtheitsgitter mit 2,3 s. Der Octree bietet also eine Leistungsverbesserung auf Ebene des Belegtheitsgitters, den größten Teil macht allerdings die modifizierte Blob-Extraktion aus.

Der hohe Laufzeitanteil der Blob-Extraktion ist damit erklärbar, dass der von Andert (2011) verwendete Floodfill-Algorithmus auf den gesamten Arrays der acht Teilgitter arbeitet. Die Effizienz dieses Algorithmus wurde allerdings nicht näher untersucht, da für das Octree-basierte Weltmodell ein abweichender Algorithmus von Feng u. a. (1998) so angepasst wurde, dass er ohne feste Array-Grenzen auf einer Hash-Tabelle aus Knoten-

Tabelle 4.4: Szenario II, Zellgröße 0,5 m: Laufzeit [s] im Array- und Octree-Modell für Gitter, Blobs und Prismen mit ihrem jeweiligen Anteil [%].

	Array Z_{64}		Array Z_{25}		Octree Z_{64}	
	s	%	s	%	s	%
Gitter[1]	15,9	2,7	6,1	2,5	2,3	30,3
Blobs[2]	573,1	96,6	237,0	95,9	3,3	43,3
Prismen	4,1	0,7	4,1	1,6	2,0	26,4
Gesamt	593,1	100,0	247,2	100,0	7,6	100,0

[1] Belegtheitsgitter entsprechend auf Array-/Octreebasis.
[2] Extraktion entsprechend auf Array-/Octreebasis.

Schlüsseln des Octrees arbeitet (siehe Kapitel 3.4.3). Offensichtlich besteht hier ein deutlicher Leistungszuwachs, da beide Ansätze vergleichbare Ergebnisse der extrahierten Hindernissen liefern. Die Laufzeit der Blob-Extraktion ist beim Array-Modell von der Größe des verwendeten Belegtheitsgitters abhängig, wie Unterschiede zwischen Konfiguration Z_{64} und Z_{25} mit 573,1 bzw. 237,0 Sekunden zeigen. Dabei ist die Menge der belegten Zellen in beiden Fällen gleich.

Die Prismengenerierung wurde im Zuge der Octree-Umstellung nur geringfügig verändert und verläuft in beiden Weltmodellen gleich, und zwar auf 3D-Arrays. Diese werden beim Octree-Ansatz temporär on-the-fly erstellt. Andert (2011) fasst Blobs an Zonengrenzen zusammen (Kapitel 3.3.4), woraus im Gegensatz zum Octree größere, zusammenhängende Blobs resultieren. Daher nimmt die Konturfindung während der Polygonisierung im Array-Modell mehr Zeit in Anspruch, als für kleinere, an Zonengrenzen geteilte Blobs beim Octree-Modell. Für Konfiguration Z_{64} und Z_{25} ist der Aufwand in Anderts Modell für die Prismenbildung mit 4,1 Sekunden identisch.

4.3 Vergleich hinsichtlich Speicherbedarf

Die Analyse der Leistung im Bezug auf Speicherplatzbedarf untersucht den *theoretische* Speicherbedarf. Eine tatsächliche Speicherbelegung der einzelnen Weltmodellkomponenten zur Laufzeit ist kompliziert zu ermitteln, da unter anderem Compilereinstellungen und die Speicherverwaltung des Betriebssystems einen großen Einfluss darauf haben. Tabel-

le 4.5 zeigt den theoretischen Speicherbedarf der beiden Weltmodellansätze bei den selben Szenariendurchläufen wie in Abschnitt 4.2. Da während der Laufzeit Datenstrukturen nach Bedarf auf-, abgebaut und umgeschichtet werden, ist jeweils nur der Maximalbedarf eines Durchgangs aufgeführt.

Tabelle 4.5: Maximaler theoretischer Speicherbedarf beim Array- und Octree-Ansatz anhand der Testszenarien mit variabler Zellgröße.

			Speicherbedarf [KiB] bei Zellgröße				
Szen.	Weltmodell		2 m	1 m	0,5 m	0,25 m	0,125 m
I	Array	Z_{64}	1110	21 678	557 167	–	–
	Array	Z_{25}	418	7555	214 571	–	–
	Octree	Z_{64}	36	128	574	3088	17 161
II	Array	Z_{64}	1031	13 759	285 462	–	–
	Array	Z_{25}	386	4939	109 989	–	–
	Octree	Z_{64}	20	67	249	1128	5900
III	Array	Z_{64}	10 523	295 127	–	–	–
	Array	Z_{25}	3861	101 204	906 727	–	–
	Octree	Z_{64}	305	649	977	1863	4603

Beim Vergleich der Array-Werte für Z_{64} und Z_{25}, ist eine proportionale Abhängigkeit des Gesamtspeicherbedarfs von der Zellanzahl des Belegtheitsgitters zu erkennen: Im Fall Z_{25} orientiert sich der eingenommene *Gesamtspeicher* des Modells sehr stark an den 39 %, die allein das *Belegtheitsgitter* für Z_{25} im Vergleich zu Z_{64} einnimmt. Dies bedeutet, dass anteiliger Speicherbedarf für Blobs und Prismen entweder ebenfalls proportional zur initialen Gittergröße, oder im Bezug aufs Gitters zu vernachlässigen ist. Die laufzeitvariable Speicherbelegung dieser einzelnen Komponenten wird im anschließenden Abschnitt 4.3.1 aufgeschlüsselt.

Wie erwartet nimmt das Array-Modell bedingt durch die Datenstruktur mehr Speicher in Anspruch als im Octree-Fall. Mit dem Octree sind selbst feine Auflösungen im Bereich der Genauigkeit des Laserscanners möglich, wenn auch nicht immer sinnvoll: Zur Kollisionsvermeidung ist ein hoher Detailgrad nicht notwendig. Im Fall des midiARTIS reicht eine Auflösung von 0,5 m bis 1 m, für schneller fliegende Vehikel kann die Umgebung noch stärker abstrahiert werden, um schneller auf Hindernisse reagieren zu können.

4.3.1 Aufschlüsselung des Speicherbedarfs

Die Ermittlung des theoretischen Speicherbedarfs einzelner Weltmodellkomponenten ist ebenfalls dreigeteilt wie die Aufschlüsselung der Laufzeit. Das Belegtheitsgitter im Array-Modell besteht aus insgesamt neun 3D-Arrays: acht Schnittzonen und die lokale Zone. Die individuellen Belegtheitswerte sind auf die Größe eines Char („character", ein Zeichen) skaliert, der auf dem Testsystem eine Speichermenge von einem Byte einnimmt. Für den „worst case" lässt sich somit die theoretische Anzahl $n_{\mathrm{B}}^{\mathrm{arr}}$ der Bytes ($_{\mathrm{B}}$) des gesamten Array-basierten ($^{\mathrm{arr}}$) Belegtheitsgitters approximieren durch

$$n_{\mathrm{B}}^{\mathrm{arr}} = n_x \cdot n_y \cdot n_z \cdot \underbrace{\mathrm{sizeof(Char)}}_{1} \cdot \underbrace{n_{\mathrm{Zonen}}}_{9} \tag{4.1}$$

mit n_x, n_y, n_z als Anzahl der Zellen in entsprechender Dimension einer Zone.

Der Octree speichert Belegtheitswerte der Zellen intern als Float („floating point number", Gleitkommazahl), was 4 Byte pro Belegtheitswert in der Testumgebung entspricht. Zusätzlich zu ihrem Belegtheitswert speichern alle *inneren* Knoten des Baums acht Zeiger auf etwaige Kinderknoten. Ein Zeiger hat hier ebenfalls die Größe von 4 Byte. Theoretischer Speicherbedarf des Gitters im Octree-Modell ($^{\mathrm{oct}}$) ergibt sich also aus

$$n_{\mathrm{B}}^{\mathrm{oct}} = n_{\mathrm{Knoten}} \cdot \underbrace{\mathrm{sizeof(Float)}}_{4} + n_{\mathrm{InnereKnoten}} \cdot \underbrace{\mathrm{sizeof(Zeiger)}}_{4} \cdot 8 \tag{4.2}$$

Tabelle 4.6 zeigt, wie der maximal eingenommene Speicher auf die Gitter, Blobs und Prismen verteilt ist. Die Blobs im Array-Modell sind individuelle Arrays, die gerade bei zusammenhängenden Blobs mit großer räumlicher Ausdehnung schnell sehr groß werden können. Im vorgestellten Octree-Modell wird ein Blob lediglich als Tabelle von tatsächlich zu ihm gruppierten Baumknoten gespeichert, wobei ein Baumknoten als Schlüsselwert mit Mittelpunktkoordinaten der korrespondierenden Gitterzelle abgebildet wird.

Generalisierte Prismen werden bei beiden Modellen identisch erstellt und bestehen aus den Punkten der Polygone ihrer Seitenflächen, was sie sehr kompakt hinsichtlich des Speicherplatzbedarfs macht. Wie bereits erwähnt nehmen Blobs im Array-Modell unter anderem mehr Speicher als im Octree-Modell ein, da sie an Zonengrenzen zusammengefasst werden. Die resultierenden Prismen aus beiden Modellen sind ähnlich, wenn auch

Tabelle 4.6: Szenario II, Zellgröße 0,5 m: Maximaler Speicherbedarf [KiB] im Array- und Octree-Modell für Gitter, Blobs und Prismen mit ihrem jeweiligen Anteil [%].

	Array Z_{64}		Array Z_{25}		Octree Z_{64}	
	KiB	%	KiB	%	KiB	%
Gitter[1]	18 433	6,5	7201	6,5	156	62,8
Blobs[2]	266 856	93,4	102 615	93,3	37	15,1
Prismen	174	0,1	174	0,2	55	22,1
Gesamt	285 462	100,0	109 989	100,0	248	100,0

[1] Belegtheitsgitter entsprechend auf Array-/Octree-Basis.
[2] Extraktion dem Array-/Octree-Modell entsprechend.

nicht gleich. Ein visueller Vergleich zeigt, dass die aus dem Octree generierten Prismen „sauberer“ sind als die vom Array abgeleiteten: Es treten weniger Überschneidungen auf und es sind weniger Kanten vorhanden. So ist der höhere Speicherbedarf für Prismen im Array-Modell erklärbar, da zu komplexeren Prismen mehr Information gespeichert wird. Generell repräsentieren die Octree-Prismen die Form der Blobs besser als die Prismen aus dem Array-Modell es tun. Warum diese Unterschiede auftreten ist allerdings unklar, da die Blobs aus beiden Modellen in ihrer Form quasi identisch sind.

Abbildung 4.3 auf Seite 64 visualisiert die Entwicklung der theoretischen Speicherbelegung beider Weltmodellansätze bei fortschreitender Integration von Laserscans in Szenario II, wobei hier die reduzierte *z*-Ausdehnung des Belegtheitsgitters (Z_{25}) für den Array-Ansatz gewählt wird, um die Gesamtdatenmenge kleiner zu halten. Wie zu sehen ist, nimmt das Gitter im Array-Modell stets konstanten Speicher in Anspruch. Etwa zwischen den Scannummern 650 und 2120 findet kein Einfügen neuer Messdaten statt, da der Hubschrauber wie erwähnt eine Schleife in größerer Höhe dreht und Scans ohne Messpunkte ignoriert werden (vgl. Abschnitt 4.1.2). Im Octree-Modell ist für diesen Zeitraum zu erkennen wie ein Zonenwechsel stattfindet, bei dem ein neuer Baum in einem Bereich ohne vorherige und aktuell erkannte Hindernisse aufgebaut wird. Entsprechend gering ist der Speicherbedarf für den leeren Baum. Kommt der Träger zurück ins Hindernisgebiet, werden im Vorfeld extrahierte Blobs als belegte Zellen in den Baum reintegriert, was in einem Sprung ab Scannummer 1660 und 1950 zu erkennen ist. Die Blobs werden ebenfalls im obigen

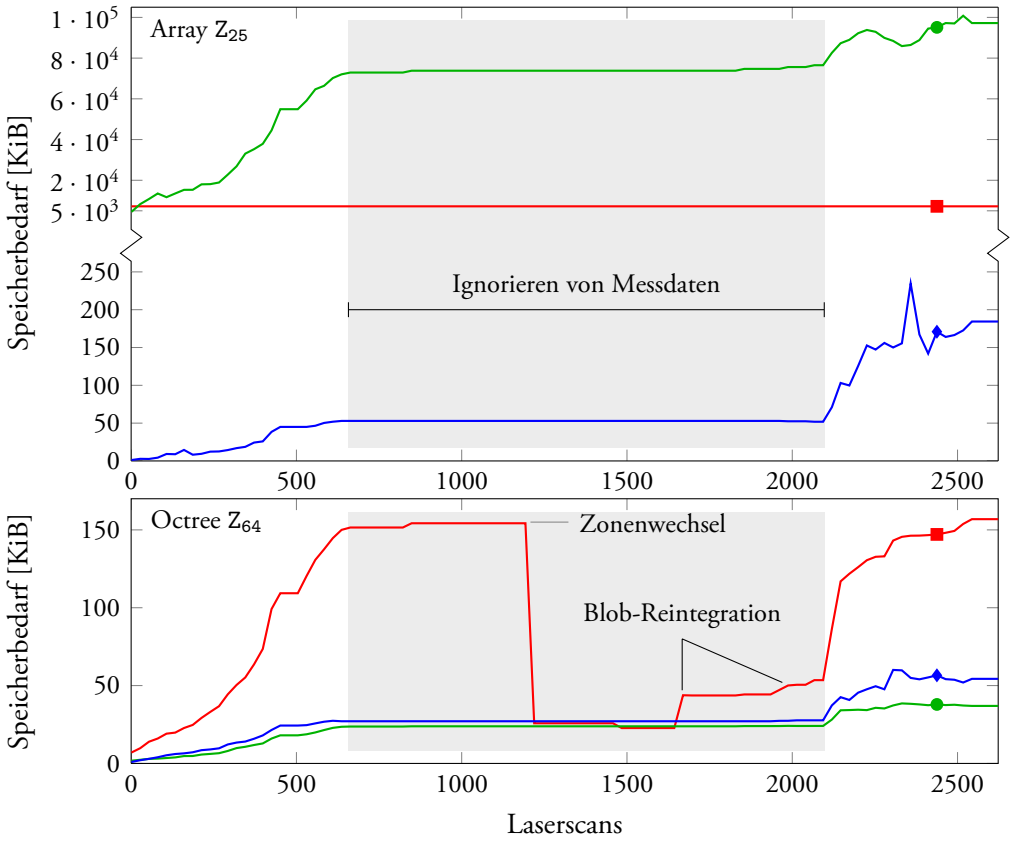


Abbildung 4.3: Szenario II: Variation der theoretischen Datenmenge für ■ Gitter, ● Blobs und ◆ Prismen bei inkrementeller Erzeugung des Array- und Octree-Weltmodells.

Array-Modell in das Belegtheitsgitter reintegriert, allerdings ist dies am Speicherbedarf des Gitters selbst nicht zu erkennen, da dieser immer konstant bleibt. Ab Scannummer 2120 werden neue Messdaten eingetragen, was an einem weiteren Zuwachs der Datenmenge für Blobs und Prismen in beiden Modellen zu erkennen ist.

Wie Tabelle 4.6 und der Octree-Plot in Abbildung 4.3 zeigen, ist die neu entwickelte Blob-Darstellung speichereffizienter als die sehr kompakte Prismendarstellung aus Anderts Modell. Es bleibt lediglich der Vorteil für die anschließende Pfadplanung, da diese ihre Berechnungen zur Kollisionsvermeidung gegen einfache Polygone durchführen kann und nicht gegen einzelne Voxel eines Blobs prüfen muss.

4.3.2 Exkurs: Speicherverwaltung des Betriebssystems

Wie die Auswertungen zeigen, führt eine speicherintensive Konfiguration des Weltmodells gerade beim Array-Modell zum Absturz der Testanwendung. Dies passiert, sobald mehr als 2 GiB an Arbeitsspeicher benötigt werden und ist in der Speicherverwaltung des Betriebssystems bei 32-bit-Anwendungen begründet. So stehen auf einem 32-bit-System theoretisch maximal 2^{32} B = 4 GiB adressierbarer Speicher als *physikalischer Adressraum* zur Verfügung. Unter 32-bit-Windows werden davon standardmäßig 2 GiB als *virtueller Systemadressraum* für Hardwarekomponenten und das Betriebssystem reserviert. Somit bleiben für Prozesse ebenfalls 2 GiB als *virtueller Benutzeradressraum* adressierbar. Wird diese Grenze überschritten, führt dies zu einem unkontrollierten Verhalten und in den meisten Fällen zum Absturz des Prozesses. Aktuelle 32-bit-Linux-Kernels[1] reservieren hingegen nur 1 GiB als virtuellen Systemadressraum (Shah 2004).

Generell besteht diese Einschränkung ebenfalls auf einem 64-bit-Windows, sofern es sich bei dem ausgeführten Prozess um einen 32-bit-Prozess handelt, obwohl theoretisch bis zu 2^{64} B = 16 EiB (Exbibyte) physikalischen Adressraums adressierbar sind. Microsoft listet aktuelle Windows-Systeme mit ihren Speicherbeschränkungen auf und beschreibt Möglichkeiten, den virtuellen Adressraum für 32-bit-Prozesse zu erweitern (Microsoft Corporation 2012). Dazu kann eine Anwendung mit der `/LARGEADDRESSAWARE`-Option (LAA) kompiliert werden, um Zugriff auf den nicht nutzbaren virtuellen Systemadressraum zu erhalten. Unter 32-bit-Windows sind damit maximal 3 GiB, unter 64-bit-Windows maximal 4 GiB von 32-bit-Prozessen adressierbar. Mehr ist allerdings auch dann nicht möglich. Diese LAA-Option kann ebenfalls im Nachhinein aktiviert werden. Die hier durchgeführten Tests lassen jedoch alle Möglichkeiten zur Erweiterung des Speicheradressraums außen vor und gehen von einer Standardkonfiguration des Systems und der Anwendung aus.

4.4 Vergleich unterschiedlicher Octree-Größen

Sensoren werden mit der Zeit weiterentwickelt und ermöglichen bereits heute größere Reichweiten, wenn auch bei größerer Bauform. Komplexere Lidar-Systeme wie das HLD-32E und HDL-64E von Velodyne (Velodyne LiDAR Inc. 2012) mit 70 m beziehungswei-

[1] Kernel bezeichnet den zentralen Betriebssystemkern, zuständig für Datenorganisation mit direktem Hardwarezugriff.

se 120 m Reichweite, oder Scanner der LUX-Familie (ibeo Automotive Systems GmbH 2012) mit bis zu 200 m Reichweite, kommen momentan hauptsächlich im Automobilbereich zum Einsatz, könnten aber auch an größeren UAVs mit höherer Traglast verwendet werden. Diese Systeme erhöhen nicht nur die Reichweite sondern stellen weitere Herausforderungen durch eine wesentlich höhere Datenmenge, da sie teilweise mit mehreren Laserstrahlen gleichzeitig über große Öffnungswinkel oder mit höheren Abtastraten arbeiten.

Im Bezug auf die Sensorunabhängigkeit des Octree-Modells stellt sich somit die Frage, wie es sich bei zunehmender Zonengröße verhält. Dass ein System zur Umgebungsmodellierung Leistungsverluste bei größeren Mengen an Messdaten mit sich zieht, ist verständlich. Allerdings verzeichnet zum Beispiel das Array-Modell trotz gleichbleibender Mengen an Messdaten bei unterschiedlichen Zonengrößen (Konfiguration Z_{64} und Z_{25}) schon starke Leistungsunterschiede.

Für den Array-Ansatz wird einerseits mehr Speicher für das Belegtheitsgitter reserviert, dafür ist aber das Zonenkonzept (siehe Kapitel 3.3.2) momentan flexibler als für den Octree und kommt bei den beschriebenen Zonenübergängen ohne Kopieroperationen aus. Der Octree hingegen ist speichereffizienter.

Anhand des Szenarios III mit 2500 m räumlicher Ausdehnung in eine Dimension kann untersucht werden, wie sich das Octree-Modell bei gleichbleibender Mengen an Messdaten aber mit unterschiedlichen Zonengrößen verhält. Größere Zonen bedeuten einerseits weniger Kopiervorgänge, da Zonengrenzen seltener überschritten werden, andererseits wächst die Baumhöhe bei fester Zellgröße, was sich beim Knotenzugriff und Traversieren des Baums negativ auswirkt (siehe Kapitel 2.3.2). Für die folgende Untersuchung wurde für den Octree eine gleichbleibende Zellgröße von 0,5 m gewählt. Die Baumhöhe verhält sich entsprechend nach der Formel

$$h = \log_2\left(\frac{l}{c}\right), \tag{4.3}$$

die aus Gleichung 3.12 folgt.

Tabelle 4.7 zeigt die gemessenen durchschnittlichen Bildfrequenzen und den maximalen Speicherbedarf. Der beobachtete Einbruch in der Bildfrequenz kann zum Einen mit der wachsenden Baumhöhe und somit steigenden inneren Knotenzahl des Octrees zu erklären sein. Zum Anderen zeigt Abbildung 4.4 auf Seite 68 bei der Aufschlüsselung der einzelnen Komponenten, dass der höchste Laufzeitzuwachs mit steigender Zonengröße bei der Blob-

Tabelle 4.7: Szenario III: Durchschnittliche Bildfrequenzen und maximaler theoretischer Speicherbedarf im Octree-Modell mit unterschiedlichen Zonengrößen bei Zellgröße 0,5 m.

	Seitenlänge der lokalen Zone					
	64 m	128 m	256 m	512 m	1024 m	2048 m
Bildfrequenz [Hz]	86	79	68	53	39	30
Speicherbedarf [KiB]	877	1213	1634	2489	4193	6570
Baumtiefe	8	9	10	11	12	13

Extraktion zu verzeichnen ist. Das ist nachvollziehbar, da die Blob-Extraktion bei mehr Hindernissen in den größeren Zonen entsprechend länger dauert.

Operationen auf Ebene des Belegtheitsgitters, wie Einfügen und Kopieren von Voxeln, fallen eher gering ins Gewicht. An dem Plot für das Gitter ist ein Einfluss von Kopiervorgängen durch Zonenübergänge nicht erkennbar, was an der geringen Datenmenge der Testdaten liegen wird. Ein kurzer Test mit dichter beieinander liegenden „künstlichen" Zeilen, und daraus resultierender größerer Datenmenge im Baum, offenbarte allerdings keine anderen Ergebnisse. Es ist jedoch zu erwarten, dass Kopieroperationen auf einem größtenteils gefüllten Baum merklich mehr Zeit in Anspruch nehmen. Dies kann bei *vollständiger* Freiraumbetrachtung der Fall sein, wenn hauptsächlich Knoten für freie Zellen vorhanden sind. Die Prismengenerierung arbeitet losgelöst und ist lediglich abhängig von der Blob-Anzahl. Nach jedem eingefügten Scan werden die Prismen auf Grundlage aller Blobs neu berechnet.

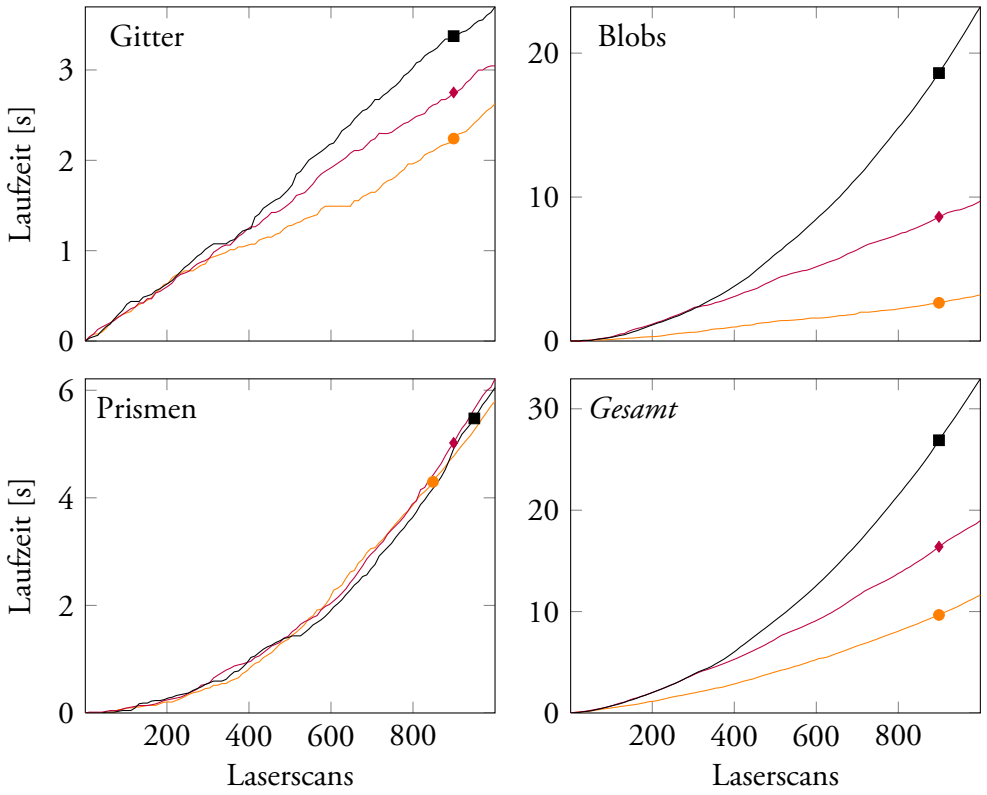


Abbildung 4.4: Szenario III: Laufzeitverhalten für Gitter, Blobs und Prismen bei inkrementeller Erzeugung des Octree-Modells. *Lokale* Zonengröße von —●— 64 m, —♦— 512 m und —■— 2048 m.

5 Schlussfolgerungen und Ausblick

5.1 Leistung des Octree-Modells

Insgesamt hat sich der vorgestellte Octree-Ansatz als leistungsstärker dargestellt. Operationen auf Ebene des Belegtheitsgitters werden beschleunigt, allerdings ist der Großteil an Performancegewinn der in diesem Kontext abweichend umgesetzten Blob-Extraktion (Kapitel 4.2.1) zuzuschreiben. Das Belegtheitsgitter in Octree-Form erlaubt größere Zonen, ohne dabei drastische Leistungseinbußen zu verzeichnen. Dies wiederum erleichtert den Einsatz von Sensoren mit höheren Messentfernungen. Im Array-Modell ist hingegen die gesamte Prozessierung abhängig von der initialen Gittergröße und verzeichnet hohe Laufzeiteinbußen mit wachsendem Gitter.

5.1.1 Mögliche Optimierung der Hindernisextraktion

Die Ergebnisse zeigen, dass die Blob-Extraktion weiterhin einen Großteil der Berechnungen im kombinierten Verfahren aus Sensordatenabbildung im Gitter und anschließender Umwandlung zu Prismen darstellt (Kapitel 4.2.1). Gerade bei einer vollständigen Freiraumbetrachtung (Kapitel 4.1.2) können sich so aufgrund vieler freier Blätter im Baum die Leistungsvorteile des Octrees in Nachteile umkehren: Der Octree wird in der aktuellen Implementierung *komplett* traversiert, um an alle belegten Blätter zu gelangen (Kapitel 3.4.3), was bei einem Großteil freier Blätter in einem Overhead an inneren Knoten resultieren kann. Wird in Erwägung gezogen, zukünftig Sensoren mit größerer Reichweite oder 3D-Sensoren wie Flash-Lidar (Kapitel 2.1.3) zu verwenden, ist vor allem mit einer Zunahme an freien Blättern zu rechnen.

Die Blob-Verarbeitung kann deutlich verbessert werden, indem die vollständige Octree-Traversierung vermieden wird. Dazu stellt die verwendete OctoMap-Bibliothek (Wurm,

Hornung u. a. 2010) eine Möglichkeit bereit, Änderungen im Baum zu verfolgen (change detection). Diese Funktion zieht zwar Leistungseinbußen bei der Integration von Scans nach sich, hat aber zur Folge, dass der Octree nicht jedes mal vollständig traversiert werden muss: Es genügt, die festgehaltenen Knoten-Änderungen zu verarbeiten und nur davon betroffene Hindernisse zu aktualisieren. Gegebenenfalls muss diese Änderungsverfolgung für kleinere Baumhöhen als 16 angepasst werden, was den Rahmen dieser Diplomarbeit übersteigen würde. Die Point Cloud Library (Rusu u. a. 2011) bietet ebenfalls eine ähnliche Funktionalität.

Als Alternative kann der Floodfill zur Segmentierung der Voxel *direkt im* Octree durchgeführt werden. Dabei liegt der Zugriff auf gesuchte Nachbarknoten zur Prüfung auf ihre Belegtheit allerdings in $O(\log n)$, im Gegensatz zu $\Theta(1)$ in der verwendeten Hash-Tabelle (Kapitel 3.4.3). Hier muss untersucht werden, wie sich solch ein direkter Floodfill im durchschnittlich gefüllten Baum im Vergleich zum schnellen Floodfill auf der Hash-Tabelle verhält, dem allerdings ein komplettes Traversieren des Baums vorausgeht.

Weiterhin werden Prismen nach jedem Sensordatenupdate aus *allen* Blobs generiert, auch wenn einige Blobs sich nicht verändert haben. Da die Blobs nur *zonenweise* aus dem Belegtheitsgitter extrahiert werden, könnte die Neuberechnung der Prismen ebenfalls nur auf diese Zonen beschränken werden, da Blobs außerhalb der Zonen konstant bleiben. Die gesamte Prozessierung ist nicht nebenläufig implementiert, kann aber gerade mit dem Zonenkonzept davon profitieren: Bei acht Zonen des Belegtheitsgitters können die anfallenden acht sequentiellen Extraktionsschritte parallelisiert werden, ohne von Problemen durch gegenseitigen Ausschluss betroffen zu sein.

5.1.2 Mögliche Optimierung auf Ebene des Belegtheitsgitters

Andert (2011) benutzt beim Verarbeiten seiner Stereodaten bereits Bilderpyramiden beim Übertragen von Disparitätsbildern ins Belegtheitsgitter. Dadurch kann zum Beispiel unnötiges mehrfaches Rasterisieren gleicher Voxel für nah beieinander liegende Strahlen vermieden werden. Alle Sensordaten werden momentan encodiert als Bilder verarbeitet und somit können Bilderpyramiden auch für Laserdaten verwendet werden.

Im aktuellen Zonenkonzept des Octrees (Kapitel 3.4.2) fallen beim Zonenübergang Kopieroperationen zwischen dem alten und neu erstellten Octree an. Beim Array-Modell (An-

dert 2011) ist dies nicht der Fall, da die acht Zonen aus einzelnen Arrays bestehen. Der Octree arbeitet intern mit Zeigern auf Knoten. Jedes Kind der Wurzel stellt mit seinen eigenen Kindern wieder einen eigenen, kleineren Octree dar, der auch als *Ast* des gesamten Baums bezeichnet werden kann. Statt Knoten vom alten in den neuen Octree zu kopieren, erscheint es plausibel einen kompletten Ast „umzuhängen“, indem man einen Knoten des neuen Baums auf den entsprechenden Ast des alten Baums zeigen lässt. Abbildung 5.1 veranschaulicht dies für zwei Quadtrees: Die Äste `C` und `E` des ungültig gewordenen Baums `A1` werden weiterverwendet und an Baum `A2` gehängt. Im Bezug auf den verwendeten Octree muss für diesen Vorgang die interne Speicherverwaltung des Baums angepasst werden, um beim darauffolgenden Löschvorgang des alten Octrees dessen umgehängte Äste nicht mit zu entfernen.

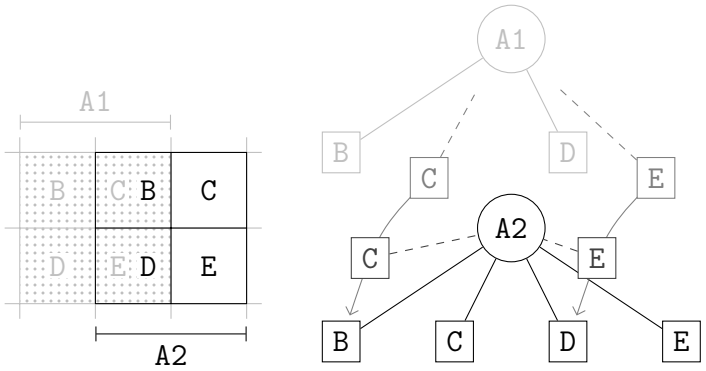


Abbildung 5.1: Umhängen alter Äste `C` und `E` aus Baum `A1` zum neuen `A2`.

5.2 Umgehen der Freizeichnungsproblematik

In Kapitel 3.2.3 wurden diskretisierungsbedingte Probleme bei der Rasterisierung im Belegtheitsgitter genannt, welche mit dem verwendeten Verfahren der Rasterisierung nicht lösbar sind. Wie die Auswertungen zeigen, bietet das Octree-Modell mit seiner höheren Leistung eine Möglichkeit, dieses Problem zu umgehen. Ein Beispiel kann das Einfügen von Messdaten bei einer Zellgröße von 0,5 Metern sein, bei dem das unerwünschte Aliasing auftritt. Die Sensordaten können nun bei einer höheren Auflösung von beispielsweise 0,125 Metern integriert werden. Auf dieser niedrigeren Diskretisierungsstufe ist die Freizeichnungsproblematik wesentlich schwächer ausgeprägt. Im Anschluss wird der Baum

zwei Ebenen darüber abgeschnitten, was wieder der Ausgangsauflösung von 0,5 Metern entspricht. Die Hindernisextraktion auf dieser Tiefe liefert nun Blobs in gewünschter Auflösung, die weniger stark vom genannten Problem betroffen sind. Die Betrachtung des Freiraums bleibt bei diesem Verfahren weiterhin korrekt.

Eine Besonderheit in diesem Fall stellt die Reintegration solcher Blobs dar (vgl. Kapitel 3.4.4). Hat der Octree zum Beispiel eine Höhe von 10 und somit auf dieser Tiefe die Auflösung von 0,125 Metern, wird ein Blob mit einer Auflösung von 0,5 Metern bei Tiefe 8 extrahiert (zwei Ebenen darüber). Er muss wieder auf der Tiefe reintegriert werden, aus der er extrahiert wurde. Die OctoMap-Bibliothek ermöglicht standardmäßig ein Einfügen von Knoten nur auf unterster Ebene, was hier zu einem Konflikt führt. Zur korrekten Reintegration solcher Hindernisse wurde OctoMap ebenfalls erweitert und ermöglicht nun, Knoten in beliebigen Tiefen im Baum einzufügen beziehungsweise zu aktualisieren.

Ein genereller Ansatz zum Mindern der Freizeichnungsproblematik ist in der Computergrafik angesiedelt und wird als *Antialiasing* bezeichnet (Angel 2006, Kapitel 7.12). Es dient genau dem Umgehen der geschilderten Aliasing-Probleme und kann entsprechend bei der Rasterisierung der Linen angewandt werden. Durch unterschiedliche Verfahren und mit verschiedenen Stärkestufen bietet es Flexibilität im Hinblick auf Rechenleistung und Qualität des Ergebnisses. Parallel wird am DLR an einer Lösung gearbeitet mit der Grundidee, zur Berechnung einzelner Zellen wie beim Antialiasing die Werte ihrer Nachbarn mit einfließen zu lassen, was mit der hier durchgeführten Rasterisierung bisher nicht möglich ist.

5.3 Klassifikation

Einen weiteren Untersuchungsaspekt stellt die Klassifikation von Messdaten dar, so dass zum Beispiel Punkte einer Bodenebene nicht zwangsweise zu Hindernissen segmentiert werden. Einfache Bodenebenen für einen Höhenschwellwert zu definieren ist ein einfacher Ansatz, der allerdings nur bei relativ ebenem Gelände sinnvoll ist (Andert 2011). Sollen Messpunkte automatisch klassifiziert werden, so müssen zuerst genügend vorhanden sein, um Muster erkennen zu können. Die Point Cloud Library (Rusu u. a. 2011) stellt Methoden zur Klassifizierung von Punktwolken bereit. Bei inkrementeller Kartierung ohne Referenzpunkte ist dies zu Beginn einer Mission allerdings schwierig. Eine Kombination mit

Intensitätswerten des Laserscanners kann das Verfahren vereinfachen (Wurm, Kummerle u. a. 2009). Allerdings müssen so für jedes Szenario zuerst Referenzwerte ermitteln werden oder die Interpretation der Intensitäten erfolgt durch Postprocessing. Zusätzlich können Grau- oder Farbwerte einer einfachen Kamera unterstützend mit Laserdaten zwecks Klassifikation verbunden werden.

5.4 Alternative Modellierungsansätze

Es kommt die Frage auf, ob die bisherige Kollisionsvermeidung mit der vorausgehenden Hindernisextraktion nicht durch ein Verfahren abgelöst werden kann, welches direkt auf dem Belegtheitsgitter arbeitet. Eingangs in Kapitel 1.4 wurden ähnliche Projekte mit guten Ergebnissen vorgestellt, die keine gesonderte Objektextraktion vornehmen. OctoMap (Wurm, Hornung u. a. 2010) bietet zum Beispiel Möglichkeiten, direkt im Octree Kollisionschecks gegen einzelne Strahlen durchzuführen.

6 Zusammenfassung

Ziel dieser Arbeit ist die Optimierung eines bestehenden Array-basierten Ansatzes zur 3D-Umgebungsmodellierung eines Kleinhubschraubers zwecks Kollisionsvermeidung. Durch den Einsatz einer Octree-Datenstruktur wird die Leistung des gesamten Systems erhöht und die Verwendung von leistungsfähigeren Sensoren ermöglicht. Bereits jetzt ist die simultane Verwendung einer Stereokamera mit einem Laserscanner zu Kartierzwecken möglich.

Anhand des Laserscanners wird das Prinzip der Datenaufnahme beschrieben. Es werden die nötigen Transformationen eingeführt, um gemessene Daten in einem global orientierten Belegtheitsgitter aufzunehmen. Das Gitter speichert Wahrscheinlichkeiten der Belegung seiner Zellen und ermöglicht eine inkrementelle Datenverarbeitung. Es kann durch unterschiedliche Datenstrukturen implementiert werden, wobei hierarchische Baumstrukturen wie Octrees Vorteile gegenüber Array-Strukturen aufweisen.

Mit der Umstellung des Umgebungsmodells auf den Octree der freien OctoMap-Bibliothek (Wurm, Hornung u. a. 2010) wird ein alternatives Zonenkonzept eingeführt, welches an die Struktur eines Octrees angelehnt ist. Durch anfallende Kopiervorgänge beim Zonenwechsel ist es allerdings noch nicht optimal umgesetzt. Ebenfalls besteht Optimierungspotenzial bei der Hindernisextraktion auf Octree-Ebene, die zwar auf einer performanten Hash-Tabelle arbeitet, aber eine komplette Traversierung des Baums voraussetzt. Es werden Ideen vorgestellt, wie diese Schritte zukünftig weiter verbessert werden können. Die im Zuge der Umstellung gemachten Änderungen an OctoMap umschließen zusammengefasst

- das Suchen von Knoten auf einer Tiefe kleiner der Maximaltiefe,
- das Einfügen und Aktualisieren von Knoten auf einer Tiefe kleiner der Maximaltiefe,
- die Verwendung von Octrees mit einer Tiefe kleiner als 16

6 Zusammenfassung

und wurden teilweise wieder dem Projekt zur Verfügung gestellt. Ab OctoMap-Version 1.4.2 sind die ersten beiden Punkte öffentlich verfügbar.

Eine Weiterentwicklung der freien OctoMap-Bibliothek kann jedoch zu Problemen in abhängigen Funktionen führen. Das in Kapitel 3.4.2 vorgestellte Zonenkonzept stellt einen speziellen Anwendungsfall des Octrees aus OctoMap dar und ist daher nicht zurück ins OctoMap-Projekt geflossen. Viele Methoden des Zonenkonzepts hängen von der aktuellen OctoMap-Implementierung in Version 1.4.2 ab. Daher kann bei signifikanten Änderungen innerhalb der Bibliothek eine fehlerfreie Funktionalität dieser Methoden nicht gewährleistet werden. Bei Verwendung einer stets aktuellen Version der eingebundenen Bibliothek ist mit jedem Update daher auch der Modellierungsprozess auf Korrektheit zu überprüfen.

A Inkrementelles Update im Belegtheitsgitter

Die Herleitung basiert auf den Grundlagen von Thrun u. a. (2005, Kapitel 4.2) und ist ebenfalls bei Stachniss (2006, Kapitel 2.2.1) für den statischen Fall zu finden. Das inkrementelle Update im stochastischen Belegtheitsgitter nimmt sich das Bayestheorem zur Hilfe, welches für den einfachen Fall die bedingte Wahrscheinlichkeit $P(x|y)$ (Kapitel 2.4) in Bezug zu ihrer „inversen" $P(y|x)$ setzt. Für $P(y) > 0$ gilt

$$P(x|y) = \frac{P(y|x) \cdot P(x)}{P(y)} \tag{A.1}$$

und für den Fall einer zusätzlichen Zufallsvariable Z gilt gleichermaßen

$$P(x|y, z) = \frac{P(y|x, z) \cdot P(x|z)}{P(y|z)}, \tag{A.2}$$

solange $P(y|z) > 0$ ist (Thrun u. a. 2005, Kapitel 2.2).

Ziel der Kartierung mittels stochastischer Belegtheitsgitter ist, die bedingten Wahrscheinlichkeiten für alle Zellen m_i des Gitter M von Beginn der Kartierung bis zum Zeitpunkt t, zu berechnen:

$$\prod_i P(m_i|r_{1:t}, x_{1:t}) = P(M|r_{1:t}, x_{1:t}). \tag{A.3}$$

Sind $r_{1:t-1}$ und $x_{1:t}$ bekannt, kann das Bayestheorem auf $P(M|r_{1:t}, x_{1:t})$ angewandt werden und es folgt

$$P(M|r_{1:t}, x_{1:t}) = \frac{P(r_t|M, r_{1:t-1}, x_{1:t}) \cdot P(M|r_{1:t-1}, x_{1:t})}{P(r_t|r_{1:t-1}, x_{1:t})}. \tag{A.4}$$

A Inkrementelles Update im Belegtheitsgitter

Mit der Annahme, dass r_t unabhängig von $r_{1:t-1}$ und $x_{1:t-1}$ ist und M bekannt ist, folgt

$$P(M|r_{1:t}, x_{1:t}) = \frac{P(r_t|M, x_t) \cdot P(M|r_{1:t-1}, x_{1:t})}{P(r_t|r_{1:t-1}, x_{1:t})}. \tag{A.5}$$

Wird nun das Bayestheorem auf $P(r_t|M, x_t)$ angewandt, folgt

$$P(r_t|M, x_t) = \frac{P(M|r_t, x_t) \cdot P(r_t|x_t)}{P(M|x_t)}. \tag{A.6}$$

Jetzt werden Gleichungen A.5 und A.6 kombiniert. Ferner wird angenommen, dass M nicht von x_t allein beeinflusst werden kann, falls keine Messung r_t vorliegt. Die Kombination ergibt dann

$$P(M|r_{1:t}, x_{1:t}) = \frac{P(M|r_t, x_t) \cdot P(r_t|x_t) \cdot P(M|r_{1:t-1}, x_{1:t-1})}{P(M) \cdot P(r_t|r_{1:t-1}, x_{1:t})}. \tag{A.7}$$

Da $m_i \in M$ eine binäre Zufallsvariable ist, kann auf gleiche Weise die Gegenwahrscheinlichkeit definiert werden als

$$P(\neg M|r_{1:t}, x_{1:t}) = \frac{P(\neg M|r_t, x_t) \cdot P(r_t|x_t) \cdot P(\neg M|r_{1:t-1}, x_{1:t-1})}{P(\neg M) \cdot P(r_t|r_{1:t-1}, x_{1:t})}. \tag{A.8}$$

Wird nun Gleichung A.7 durch A.8 geteilt, ergibt sich

$$\frac{P(M|r_{1:t}, x_{1:t})}{P(\neg M|r_{1:t}, x_{1:t})} = \frac{P(M|r_t, x_t) \cdot P(\neg M) \cdot P(M|r_{1:t-1}, x_{1:t-1})}{P(\neg M|r_t, x_t) \cdot P(M) \cdot P(\neg M|r_{1:t-1}, x_{1:t-1})}. \tag{A.9}$$

Hält man sich vor Augen, dass $P(\neg M) = 1 - P(M)$ (Abschnitt 2.4), folgt

$$\frac{P(M|r_{1:t}, x_{1:t})}{1 - P(M|r_{1:t}, x_{1:t})} = \frac{P(M|r_{1:t-1}, x_{1:t-1})}{1 - P(M|r_{1:t-1}, x_{1:t-1})} \cdot \frac{P(M|r_t, x_t)}{1 - P(M|r_t, x_t)} \cdot \frac{1 - P(M)}{P(M)}, \tag{A.10}$$

was vereinfacht dargestellt mit $P(M)$ als Anfangswahrscheinlichkeit $P(M)_0$

$$\frac{P(M)_{1:t}}{1 - P(M)_{1:t}} = \frac{P(M)_{1:t-1}}{1 - P(M)_{1:t-1}} \cdot \frac{P(M)_t}{1 - P(M)_t} \cdot \frac{1 - P(M)_0}{P(M)_0} \tag{A.11}$$

ergibt. Nach Logit-Schreibweise (Gleichung 3.6) folgt

$$L(M)_{1:t} = L(M)_{1:t-1} + L(M)_t - L(M)_0. \tag{A.12}$$

Erklärung

Hiermit versichere ich, die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt, sowie die Zitate deutlich kenntlich gemacht zu haben.

Münster, 20. November 2012

(Michael Scholz)

Literatur

Advanced Scientific Concepts Inc. (2012): *3D flash Lidar - technology overview*, URL: `http://advancedscientificconcepts.com/technology/technology.html` (aufgerufen am 27. 08. 2012).

Albertz, J. (2009): *Einführung in die Fernerkundung: Grundlagen der Interpretation von Luft- und Satellitenbildern*, 4. Aufl., Wissenschaftliche Buchgesellschaft.

Amanatides, J. und Woo, A. (1987): „A fast voxel traversal algorithm for ray tracing", in *Eurographics*, S. 3–10.

Andert, F. (2011): „Bildbasierte Umgebungserkennung für autonomes Fliegen", Dissertation, Technische Universität Braunschweig.

Andert, F. und Adolf, F. (Aug. 2009): „Online world modeling and path planning for an unmanned helicopter", in *Autonomous Robots* 27.3, S. 147–164.

Andert, F. und Goormann, L. (2007): „Combined grid and feature-based occupancy map building in large outdoor environments", in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, S. 2065–2070.

Andert, F. und Goormann, L. (2008): „A fast and small 3-D obstacle model for autonomous applications", in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, S. 2883–2889.

Angel, E. (2006): *Interactive computer graphics*, 4. Aufl., Pearson Education.

Bentley, J. L. (Sep. 1975): „Multidimensional binary search trees used for associative searching", in *Communications of the ACM* 18.9, S. 509–517.

Borenstein, J., Everett, H. R. und Feng, L. (1996): *Where am I? Sensors and methods for mobile robot positioning*, Techn. Ber., University of Michigan.

Bry, A., Bachrach, A. und Roy, N. (Mai 2012): „State estimation for aggressive flight in GPS-denied environments using onboard sensing", in *IEEE International Conference on Robotics and Automation*, S. 1–8.

Literatur

Buchholz, J. J. (2002): *Regelungstechnik und Flugregler*, Grin.

DIN 9300: *Begriffe, Größen und Formelzeichen der Flugmechanik*, Teil 1: Bewegung des Luftfahrzeugs gegenüber der Luft, S. 13–29.

Elfes, A. (1989): „Occupancy grids: A probabilistic framework for mobile robot perception and navigation", Dissertation, Carnegie Mellon University.

Feng, L. und Soon, S. H. (1998): „An effective 3D seed fill algorithm", in *Computers & Graphics* 22.5, S. 641–644.

Förterer, H. (1995): *Iterativer 3D flood fill*, URL: `http://www.foerterer.com/gfx/volume/3dflood.htm` (aufgerufen am 27. 08. 2012).

Fournier, J., Ricard, B. und Laurendeau, D. (Mai 2007): „Mapping and exploration of complex environments using persistent 3D model", in *Canadian Conference on Computer and Robot Vision*, S. 403–410.

Goormann, L. (2004): „Objektorientierte Bildverarbeitungsalgorithmen zum relativen Hovern eines autonomen Helikopters", Diplomarbeit, Fachhochschule Braunschweig-Wolfenbüttel.

Hokuyo Automatic Co. (2009): *Scanning laser range finder UTM-30LX/LN specification*, URL: `http://www.hokuyo-aut.jp/02sensor/07scanner/utm%5C_30lx.html` (aufgerufen am 27. 08. 2012).

Hrabar, S. (2012): „An evaluation of stereo and laser-based range sensing for rotorcraft unmanned aerial vehicle obstacle avoidance", in *Journal of Field Robotics* 29.2, S. 215–239.

Hunter, G. M. (1978): „Efficient computation and data structures for graphics", Dissertation, Princeton University.

ibeo Automotive Systems GmbH (2012): *Ibeo LUX*, URL: `http://www.ibeo-as.com` (aufgerufen am 27. 08. 2012).

Leonard, J. J. und Durrant-Whyte, H. F. (1992): *Directed sonar sensing for mobile robot navigation*, Springer.

Longley, P. A., Goodchild, M. F., Maguire, D. J. und Rhind, D. W. (2005): *Geographic information systems and science*, 2. Aufl., John Wiley & Sons.

Lorenz, S. (2010): „Adaptive nichtlineare Regelung zur automatisierten Flugbereichserweiterung für den Technologiedemonstrator ARTIS", Dissertation, Technische Universität Braunschweig.

Mehlhorn, K. und Sanders, P. (2008): *Algorithms and data structures - The basic toolbox*, Springer.

Microsoft Corporation (2012): *Memory limits for Windows releases*, URL: `http://msdn.microsoft.com/en-us/library/aa366778(VS.85).aspx` (aufgerufen am 27. 08. 2012).

Miller, R. und Amidi, O. (1998): *3-D site mapping with the CMU autonomous helicopter*, Techn. Ber., Carnegie Mellon University.

Moravec, H. P. (1988): „Sensor fusion in certainty grids for mobile robots", in *AI Magazine* 9.2, S. 61–74.

Moravec, H. P. und Elfes, A. (1985): „High resolution maps from wide angle sonar", in *International Conference on Robotics and Automation*, S. 116–121.

Preiss, B. R. (1998): *Data structures and algorithms with object-oriented design patterns in Java*, John Wiley & Sons.

Rusu, R. B. und Cousins, S. (2011): „3D is here: Point Cloud Library (PCL)", in *IEEE International Conference on Robotics and Automation*.

Samet, H. (2006): *Foundations of multidimensional and metric data structures*, Morgan Kaufmann.

Scherer, S., Singh, S., Chamberlain, L. und Elgersma, M. (2008): „Flying fast and low among obstacles: Methodology and experiments", in *International Journal of Robotics Research* 27.5, S. 549–574.

Shah, A. (2004): *Feature: High memory in the Linux kernel*, URL: `http://kerneltrap.org/node/2450` (aufgerufen am 24. 06. 2012).

Stachniss, C. (2006): „Exploration and mapping with mobile robots", Dissertation, Universität Freiburg.

Thieleke, F., Dittrich, J. S. und Bernatz, A. (2004): *ARTIS - ein VTOL UAV Demonstrator*, Techn. Ber., Deutsches Zentrum für Luft- und Raumfahrt e.V.

Thrun, S., Burgard, W. und Fox, D. (2005): *Probabilistic robotics*, MIT Press.

Velodyne LiDAR Inc. (2012): *HDL-64E*, URL: `http://velodynelidar.com/lidar/hdlproducts/hdl64e.aspx` (aufgerufen am 27. 08. 2012).

Wurm, K. M., Hornung, A., Bennewitz, M., Stachniss, C. und Burgard, W. (2010): „OctoMap: A probabilistic, flexible, and compact 3D map representation for robotic sys-

Literatur

tems“, in *ICRA 2010 workshop on best practice in 3D perception and modeling for mobile manipulation*.

Wurm, K. M., Kummerle, R., Stachniss, C. und Burgard, W. (Okt. 2009): „Improving robot navigation in structured outdoor environments by identifying vegetation from laser data“, in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, S. 1217–1222.