technische universität
dortmund

fakultät für informatik

Projektgruppen-Endbericht

inStaNt-drone
Grafisch interaktive Steuerung und autonome Navigation von Drohnen

Projektgruppe 591

10. Oktober 2016

Teilnehmer:

Henning Brockmann, Jan Goclik, Christian Günter, Sergei Harder, Tim Heming, Sebastian Humann, Michael Ihne, Jan Mühlig, Tibor Ostwald, Erik Thordsen, Sascha Threbank

Betreuer:

Dr. Frank Weichert

Dipl.-Inform. Denis Fisseler, Dipl.-Inform. Frederik Lütkes

Lehrstuhl Informatik VII
Graphische Systeme
TU Dortmund

Inhaltsverzeichnis

Mathematische Notation **1**

1. Einleitung **3**

1.1. Motivation und Hintergrund 3
1.2. Ziel 4
1.3. Aufbau der Arbeit 4

2. Exploration ubiquitärer Umgebungen **7**

2.1. Stand der Technik 7
2.1.1. Autonome Exploration mittels Micro Aerial Vehicles (MAV)s . 7
2.1.2. Rekonstruktion anhand von Kamerabildern 9
2.2. Technische Möglichkeiten 10
2.3. Aufgabenaufteilung 11

3. Drohnen **15**

3.1. Einleitung 15
3.1.1. Parrot Bebop Drone 15
3.1.2. Parrot AR.Drone2 16
3.2. Anwendungsbereiche 16
3.3. Sensoren 18
3.4. Aktoren 19
3.5. Steuerung der Rotoren 21
3.6. Regelungstechnik 22
3.6.1. Regelung allgemein 22
3.6.2. Cyber-Physical Systems: hardware-in-the-loop 24
3.6.3. Regelung der Lage und Bewegung 25
3.6.4. Regler-Kaskadierung 25
3.6.5. Navigation 26
3.6.6. PID-Regler 28

4. Selbstlokalisation und Erkennung von Hindernissen **31**
4.1. Einleitung . . . 31
4.2. Odometrie . . . 32
4.2.1. Kalman-Filter . . . 33
4.2.2. Fourier-Filter . . . 37
4.2.3. Optischer Fluss . . . 41
4.2.4. Sensorfusion . . . 42
4.2.5. Datenspeicherung . . . 45
4.3. Detektion der Umgebung . . . 46
4.3.1. SLAM-Verfahren . . . 46
4.3.2. LSD-SLAM . . . 48
4.3.3. Einsatz und Erweiterung von LSD-SLAM . . . 57
4.3.4. Effiziente Datenhaltung . . . 68
4.4. Diskussion . . . 73

5. Pfadplanung **77**
5.1. Einleitung . . . 77
5.2. Berechnung eines Erkundungsplans . . . 79
5.2.1. FloodFill . . . 80
5.2.2. Frontiers . . . 83
5.2.3. Kombination aus FloodFill und Bresenham . . . 85
5.3. Umgebungsrepräsentation . . . 86
5.3.1. Polygonbasierte Verfahren . . . 86
5.3.2. Voxelbasierte Verfahren . . . 87
5.4. Methoden zur Erzeugung eines Pfades . . . 89
5.4.1. Einfache Pfadsuche . . . 90
5.4.2. A*-Algorithmus . . . 90
5.4.3. Jump Points Search . . . 92
5.5. Simplifizierung des berechneten Pfades . . . 94
5.6. Simulation . . . 100
5.6.1. Aufgaben . . . 100
5.6.2. Schnittstelle zur Anwendung . . . 100
5.6.3. Benutzerschnittstelle . . . 101
5.7. Zusammenfassung . . . 102

6. Rekonstruktion der Umgebung **105**
6.1. Mapping und Tracking . . . 105
6.1.1. Vorverarbeitung der Tiefendaten . . . 107
6.1.2. Szenenrepräsentation . . . 110
6.1.3. Tiefenvorhersage . . . 112
6.1.4. Posenschätzung . . . 113
6.1.5. Fusion mit der Szene . . . 114
6.1.6. Transformation in Polygonale Repräsentation . . . 115
6.1.7. Erweiterung zum Weltmodell . . . 116
6.2. Punktwolkenbasierte Rekonstruktion . . . 120
6.2.1. Punktwolkenrepäsentation und Verarbeitung . . . 120
6.2.2. Registrierung . . . 124
6.2.3. Iterative Closest Point Algorithmus . . . 127
6.2.4. Poisson Oberflächenrekonstruktion . . . 134
6.3. Zusammenfassung . . . 136

7. Adaptive Steuerungs- und Akquisitionskonzepte **141**
7.1. Einleitung . . . 141
7.2. Steuerung über die offizielle Schnittstelle . . . 141
7.2.1. Befehlssatz zur Steuerung . . . 142
7.2.2. Odometrie . . . 144
7.3. Ansteuerung der Motoren . . . 144
7.3.1. Parrot AR.Drone2 . . . 145
7.3.2. Parrot Bebop Drone . . . 145
7.4. Sensordaten-Erfassung . . . 145
7.5. Schnittstellen zu den Bildsensoren . . . 146
7.5.1. Video for Linux Two . . . 146
7.5.2. OpenNI2 . . . 147
7.6. Weiterverarbeitung der Bilddaten . . . 148
7.6.1. Videoformate . . . 148
7.6.2. Video-Codecs . . . 148
7.6.3. Einbetten der Odometrie . . . 150
7.6.4. Kompression von Tiefenbildern . . . 151
7.7. Kommunikation mit der Drohne . . . 152
7.7.1. RTSP/RTP . . . 152
7.7.2. TCP-Socket für Tiefenvideo . . . 153
7.7.3. JSON . . . 153

7.7.4. Websockets . . . 156
7.8. Software-Schnittstelle zur Steuerung . . . 157

8. Benutzerschnittstellen **161**
8.1. ARGOS-Control . . . 161
8.1.1. Manuelle Steuerung . . . 162
8.1.2. Live-Visualisierung . . . 162
8.2. Rekonstruktion . . . 163

9. Evaluierung **169**
9.1. Evaluierungsszenarien . . . 169
9.1.1. ARGOS-Agent . . . 169
9.1.2. Raumerkundung . . . 172
9.1.3. Tracking . . . 173
9.1.4. Umgebungsrekonstruktion . . . 178
9.2. Auswertung . . . 180
9.2.1. ARGOS-Agent . . . 180
9.2.2. Raumerkundung . . . 185
9.2.3. Tracking . . . 186
9.2.4. Umgebungsrekonstruktion . . . 189
9.3. Diskussion . . . 192

10. Zusammenfassung und Ausblick **197**
10.1. Zusammenfassung . . . 197
10.2. Ausblick . . . 198

A. Anhang **201**
A.1. Pflichtenheft . . . 201

Abbildungsverzeichnis **230**

Algorithmenverzeichnis **231**

Quellcodeverzeichnis **233**

Literaturverzeichnis **235**

Mathematische Notation

Notation	Bedeutung
$\mathbb{N}$	Menge der natürlichen Zahlen $1, 2, 3, \ldots$
$\mathbb{R}$	Menge der reellen Zahlen
$\mathbb{R}^d$	d-dimensionaler Raum
$\mathcal{M} = \{m_1, \ldots, m_N\}$	Menge $\mathcal{M}$ von N Elementen m_i
$\mathbb{G} = (\mathcal{V}, \mathcal{E})$	Graph $\mathbb{G}$ mit Knotenmenge $\mathcal{V}$ und Kantenmenge $\mathcal{E}$
$\mathrm{T} = (a, b)$	Tupel T mit Elementen a und b
$\mathbf{p}$	Vektor
$\mathbf{p}_i$	i-tes Element eines Vektors
$\mathbf{v}_i^{(j)}$	i-tes Element des j-ten Vektors
$\mathbf{A}$	Matrix
$\|\|\mathbf{x}\|\|_2$	Euklidische Norm
$exp(x)$	Exponentialfunktion e^x
$\langle \mathbf{x}, \mathbf{y} \rangle$	Skalarprodukt zweier Vektoren $\mathbf{x}$, $\mathbf{y}$
$\mathbf{x} \times \mathbf{y}$	Kreuzprodukt zweier Vektoren $\mathbf{x}$, $\mathbf{y}$

1. Einleitung

Die Entwicklung von Drohnen, speziell Quadrocoptern, wurde in den letzten Jahren stark forciert. Die Vielfalt der Einsatzmöglichkeiten und die einfache Handhabung durch ein stabiles Flugverhalten sind Gründe für die starke Verbreitung. Dabei gilt es grundsätzlich zwei Steuerungsmethoden zu unterscheiden. Zum einen die direkt gesteuerten Drohnen, welche mit einer Fernbedienung kontrolliert und geflogen werden. Diese Drohnen verfügen meist über ein Assistenzsystem, dass auf der Drohne läuft und dafür sorgt, dass sie beispielsweise eine bestimmte Postion und Höhe hält, falls keine Eingaben getätigt werden. Der Anwender steuert in diesem Fall nicht direkt die Motoren, sondern teilt dem System eine gewünschte Richtung und Beschleunigung mit. Diese werden daraufhin von der Drohne ausgeführt. Solche Drohnen werden beispielsweise eingesetzt, wenn präzise Flugmanöver benötigt werden, um Luftaufnahmen eines bestimmten Gebietes zu erhalten. Die zweite Steuerungsmethode, der autonome Flug, wird meist im offenen Gelände verwendet. Dazu sind die Versuche von Amazon zum Transport von Paketen ein aktuelles Anwendungsbeispiel[1]. Diese Projektgruppe verfolgt ebenfalls den Ansatz der autonomen Steuerung. Im Vergleich zu anderen Verfahren wird jedoch nicht im Gelände mit GPS navigiert, sonder innerhalb von eines geschlossenen Raumes.

1.1. Motivation und Hintergrund

Mit dem schnell voranschreitenden Virtual Reality Technologien entsteht auch eine größere Nachfrage nach 3D-Modellen realer Umgebungen. Problematisch ist bislang jedoch die Erzeugung solcher Modelle, weil dafür der zu erzeugende Raum mit einer 3D-Kamera abgelaufen werden muss. Ein System, welches autonom den Raum erkundet und in ein 3D-Modell überführt, wäre jedoch wünschenswert. Aus diesem Grund befasst sich diese Projektgruppe mit der Möglichkeit die Raumerkundung mit Hilfe einer Drohne vorzunehmen, die autonom den Raum erkundet. Auf diese Weise

[1] `http://www.handelsblatt.com/unternehmen/handel-konsumgueter/prime-air-mit-mini-drohnen-achtung-da-kommt-ein-amazon-paket-geflogen/9155732.html` Abruf: 22.06.16 14:00

könnte Architekten und Raumgestaltern ein 3D-Modell eines Raumes bereitgestellt werden, das sie für ihre Arbeit verwenden.

1.2. Ziel

Das Ziel dieser Projektgruppe ist es eine Drohne so zu programmieren, dass sie autonom, folglich ohne Eingaben des Anwenders, startet und einen geschlossenen Raum erkundet. Das bedeutet, dass die Drohne selbständig ihre Position im Raum bestimmen soll. Zur Erkundung gehört ebenfalls die Kenntnis über bereits angeflogene Punkte im Raum. Der Weg, den die Drohne durch den Raum nimmt, soll zudem dynamisch an die Gegebenheiten angepasst werden. Das heißt, dass Hindernisse erkannt und umflogen werden müssen. Ein Eingreifen, wie beispielsweise Notlanden und Lenken, durch den Anwender sollte jedoch aus Sicherheitstechnischen Gründen zu jedem Zeitpunkt möglich sein. Nachdem der Raum vollständig erkundet wurde, soll mithilfe der gesammelten Daten einer 3D-Kamera ein möglichst realitätsgetreues 3D-Modell des Raums erzeugt werden. Während des Fluges soll der Status der Drohne und der Fortschritts der Erkundung visualisiert werden.

1.3. Aufbau der Arbeit

Diese Arbeit befasst sich mit den Arbeitsschritten und Ergebnissen der Projektgruppe. Dem Leser wird mit den Kapiteln 1 und 2 eine Einführung in das Thema der Projektgruppe gegeben. Hierzu werden in Kapitel 2 Methoden zur Exploration vorgestellt, die den aktuellen Stand der Technik vermitteln. Weitergehend erklärt der Abschnitt 2.3, wie sich die Projektgruppe zur Lösung der Problemstellung organisiert hat. Das Grundlagenkapitel 3 erläutert allgemein die verwendete Hardware der Drohne und ihre möglichen Anwendungsbereiche. Dabei werden die Sensoren, Aktoren und die Regelungstechnik erklärt und analysiert. Die Selbstlokalisation und Erkennung von Hindernissen, Kapitel 4, erläutert den ersten Arbeitsschritt der Projektgruppe zur Kalkulation der Postion der Drohne aus den Kamera- und Sensor-Daten. Diese berechneten Informationen dienen im Kapitel 5 zur Berechnung eines Pfades, der von der Drohne geflogen werden soll. Eine Übersicht zum Aufbau der einzelnen Module und deren Kommunikation befindet sich im Abschnitt 2.3. Das Kapitel 6 erläutert, wie aus den rohen 3D-Kameraaufnahmen ein fertiges 3D-Modell generiert wird. Das Kapitel 7 befasst sich mit der hardwarenahen Programmierung der Drohne. Besonderes Augenmerk verlangen die Schritte zur Erfassung der Kame-

radaten in 2D sowie 3D und die anschließende Codierung und Weiterleitung über eine Netzwerkverbindung. Hierzu werden Kodierungsformate vorgestellt die zum Datenaustausch dienen. Das fertige Produkt aus Anwendersicht wird in Kapitel 8 präsentiert. Die Ergebnisse der Projektgruppe werden im Kapitel 9 evaluiert und diskutiert. Eine abschließende Zusammenfassung bietet das Kapitel 10. Hinzukommt ein Ausblick zur möglichen Erweiterung oder Verwendung des erarbeiteten Systems.

2. Exploration ubiquitärer Umgebungen

In diesem Kapitel wird zuerst ein Überblick über verschiedene aktuelle und ähnliche Projekte gegeben. Dazu werden im Unterkapitel „Stand der Technik“ (siehe Abschnitt 2.1) mehrere Projekt aus den Bereichen der autonomen Raumerkundung mit sog. MAV und der 3D-Rekonstruktion anhand von Tiefendaten gezeigt. Im zweiten Unterkapitel „Technische Möglichkeiten“ (siehe Abschnitt 2.2) werden neue Technologien in den Bereichen der MAV und der 3D-Rekonstruktion vorgestellt. Das letzte Unterkapitel „Aufgabenaufteilung“ (siehe Abschnitt 2.3) erläutert die Ziele der Projektgruppe.

2.1. Stand der Technik

In diesem Unterkapitel werden verschiedene Ansätze im Bereich der Exploration ubiquitärer Umgebungen mittels sog. MAV vorgestellt. Unter diese fällt auch die in Abschnitt 3.1 vorgestellte Drohne. Bei Erstellung dieses Berichts ist keine Veröffentlichung bekannt, welche die Fähigkeiten des ARGOS-Systems komplett abdeckt. Das ARGOS-System lässt sich in zwei Bereiche unterteilen, welche in verschiedenen Projekten bereits umgesetzt wurden. In Abschnitt 2.1.1 werden mehrere aktuelle Projekte vorgestellt, welche einen Raum mit einem MAV autonom erkunden. Daraufhin werden in Abschnitt 2.1.2 Ansätze im Bereich der Rekonstruktion anhand von Kamerabildern gezeigt.

2.1.1. Autonome Exploration mittels MAVs

In diesem Abschnitt werden mehrere Systeme vorgestellt, welche einen Raum, der im Folgenden als Szene bezeichnet wird, mit einem MAV autonom erkunden. Diese Systeme bieten teilweise auch eine Rekonstruktion der Szene als 2D-Karte, jedoch nicht als 3D-Karte, an. Zudem werden die Berechnungen der Selbstlokalisation und

der Bahnplanung nicht auf der Drohne ausgeführt, sondern auf einem externen Gerät, wie bspw. einem Laptop.

Camera-Based Navigation of a Low-Cost Quadrocopter

An der Technischen Universität München wurde 2012 ein Programm entwickelt, welches einen Quadcopter autonom einen unbekannten Raum erkunden lässt [23]. Die Berechnungen für den Flug finden auf einem externen Gerät, wie bspw. einem Laptop, statt. Besondere Merkmale sind, dass ein Flug im unbekannten Raum, eine Genauigkeit von bis zu minimal 4,9cm (Innenflug) und 18cm (Außenflug) aufweist und das Wiederfinden der eigenen Position nach dem Verlust des Trackings (siehe Kapitel 4.1) möglich ist. Zusätzlich ist das System robust gestaltet, so kann die Drohne trotz einer Kommunikationsverzögerung von bis zu 400ms weiterfliegen. Das Programm basiert auf drei Hauptkomponenten. Dazu zählt ein monokulares Simultaneous Localization and Mapping (SLAM)-Verfahren (siehe Kapitel 4.3.1), welches die aktuelle Position der Drohne berechnen und einordnen soll. Die errechneten Daten werden zur Selbstlokalisation und zur Prognose der Pose benutzt. Eine weitere Methode zur Selbstlokalisation, falls die Daten des SLAM-Verfahrens nicht verwendbar sind, ist das Schätzen der Position anhand der Odometriedaten (siehe Kapitel 4.2). Zu den anderen Hauptkomponenten zählt ein erweiterter Kalman-Filter (siehe Kapitel 4.2.1) und ein PID-Kontroller (siehe Kapitel 3.6.6). Diese werden in der Steuerung der Drohne eingesetzt, damit diese gleichmäßig und stillstehend in der Luft schweben und Flugbefehle mit einer hohen Präzision befolgen kann.

Vision-Based Autonomous Mapping and Exploration Using a Quadrotor MAV

An der ETH Zürich wurde 2012 ein ähnliches Projekt entwickelt [30]. Im Gegensatz zum Projekt der TU München (siehe Kapitel 2.1.1) wurde hier eine Stereokamera (siehe Kapitel 3.3) benutzt, welche eine texturierte 3D-Szene mit den vorher aufgezeichneten Originalbildern herstellt. Große Unterschiede der Verfahren liegen in der Schätzung der Position, in der autonomen Erkundung sowie in der Erstellung der Umgebungskarte. Auch hier wird ein SLAM-Verfahren eingesetzt, welches auf einem externen Gerät ausgeführt wird. Das Erfassen der aktuellen Position erfolgt durch zwei verschiedene Verfahren. Zum einen wird die Pose partiell durch den Optischen Fluss (siehe Kapitel 4.2.3) geschätzt. Dazu wird ein eindimensionaler Kalman-Filter (siehe Kapitel 4.2.1) verwendet. Zum anderen wird die gesamte Pose anhand von visueller Odometrie geschätzt. Dafür wird ein SLAM-Verfahren verwendet.

Monocular Autonomous Exploration in Unknown Environment with Low-Cost Quadrotor

Das vom „Laboratorio di Intelligenza Artificiale e Robotica del Politecnico di Milano“ entwickelte System [27] hat Ähnlichkeiten zu den zuvor genannten Systemen der TU München (siehe Kapitel 2.1.1) und ETH Zürich (siehe Kapitel 2.1.1). Hier wird ebenfalls ein SLAM-Verfahren und ein erweiterter Kalman-Filter verwendet. Ein Unterschied besteht in der Erkundung des Raumes. Hierfür wird eine zweidimensionale Karte der Szene erstellt und zum Navigieren und Planen der Route verwendet. Diese Karte gibt einen Grundriss des erkundeten Raumes an, doch die Repräsentation des Raumes ist ungenau und dient nur einer groben Übersicht. Alle Berechnungen finden auf einem externen Gerät statt.

2.1.2. Rekonstruktion anhand von Kamerabildern

Im Folgenden werden zwei Technologien vorgestellt, welche anhand von Tiefendaten eine 3D-Karte rekonstruieren. Beide Systeme benutzen eine kommerziell erhältliche 3D-Kamera, wie bspw. eine Kinect von Microsoft [14].

KinectFusion: Real-Time Dense Surface Mapping and Tracking

Das am „Imperial College London“ und von Microsoft Research entwickelte System ermöglicht die Rekonstruktion von 3D-Karten sowie die Selbstlokalisation [58]. Hier wird nur auf die Rekonstruktion des Raumes eingegangen, da Selbstlokalisation bereits in Kapitel 2.1.1 besprochen wurde. Die Rekonstruktion ist für Räume von bis zu $7m^3$ geeignet. Für diese wird ein iterative closest point (ICP) Algorithmus verwendet, welcher die gesamten Tiefendaten benutzt, um die Szene zu rekonstruieren. Interessant ist, dass die Rekonstruktion in Echtzeit passiert. Um die Verarbeitung der Daten in Echtzeit zu garantieren, werden die Berechnungen auf einer Grafikkarte ausgeführt. Dies ist möglich, da die Berechnungen oft parallelisiert werden können [58]. Die resultierende 3D-Karte kann um Farbinformationen erweitert werden und die interne Repräsentation der Daten wird mittels einer Signed Distance Function (SDF) vorgenommen (siehe Kapitel 6.1.2). Nähere Informationen hierzu finden sich in Kapitel 6.1.

Real-Time Camera Tracking and 3D Reconstruction Using Signed Distance Functions

Das an der „Lund University“ in Schweden und an der TU München entwickelte System unterscheidet sich vom System des Imperial College London (siehe Kapitel 2.1.2) in einem Bereich. Es wird behauptet, dass das System schneller und robuster als der Ansatz von KinectFusion mittels ICP ist [11]. Die interne Repräsentation der Szene wird durch SDF realisiert. Die Kolorierung wird durch die drei Farben Rot, Grün und Blau sowie einer vierten Variable, welche die Gewichtung der Farbe angibt, definiert. Alle Berechnungen können parallelisiert werden, sodass auch hier eine Berechnung der Szene auf einer Grafikkarte in Echtzeit möglich ist.

2.2. Technische Möglichkeiten

Im Bereich der Umgebungserkundung werden aktuell „light detection and ranging“ (LiDAR)-Systeme verwendet. Diese Systeme messen mithilfe eines rotierenden Lasers die radiale Entfernung der Umgebung. So gemessene Punkte lassen sich in eine Umgebungskarte eintragen, womit sich eine Punktwolke ergibt. Kaul u. a. haben eine Flugdrohne mit einem Unterbau versehen, der die Verwendung eines LiDAR-System ermöglicht. Dieser Unterbau ist eine spezielle Anfertigung, die es dem Sensor Bewegungen und Drehungen ermöglicht. Durch die Bewegung kann der Sensor in mehr Richtungen scannen und so eine höhere Genauigkeit erreichen [45].
Neben einer Lasermessung kann eine Messung ebenso über Kameras stattfinden. Die Autoren Engel u. a. haben bereits ein Tracking mithilfe eines SLAM-Algorithmus über eine von Werk verbaute Frontkamera einer handelsüblichen Drohne umgesetzt. Da im Handel jedoch viele Flugdrohnen frei erhältlich sind, wäre eine Auflistung aller Drohnen für diese Arbeit zu umfangreich. Stattdessen wird eine beispielhafte Auflistung weniger Drohnen ausreichen müssen.

–	Hexacopter FLAT-HEAD S800	Drohne DJI Inspire 1	Parrot Bebop
Kamera	Kamera nachrüstbar	FC350	Kamera mit Fisheye-Linse
Videoauflösung	—	ca 3000px×4000px @ 24-30fps	1920px×1080px @30fps
Preis beim Hersteller	1489€	3199€	399€
Traglast	exkl. Akku 2kg	keine Angabe	keine Angabe

Tabelle 2.1.: Kurzüberblick über die Drohnen (siehe Abbildung 2.1)

(a) **(b)** **(c)**

Abbildung 2.1.: (a) Hexacopter FLAT-HEAD S800 (Bild der Thesis von Chávez et al. [12] entnommen) (b) Drohne DJI Inspire 1 (Bild der Webseite der Firma DJI entnommen [1].) (c) Parrot Bebop (Die Abbildung stammt aus der *Parrot AR.Drone2.0 Bedienungsanleitung* von Parrot SA [62].)

In der Tabelle 2.1 wird deutlich, dass unterschiedliche Drohnen für unterschiedliche Anwendungsbereiche existieren. So ist bspw. der Hexacopter (vergleiche Abbildung 2.1(a)), bei dem verschiedene Module angebaut werden können, für den professionellen Bereich ausgelegt. Zu den Drohnen DJI Inspire 1 und Parrot Bebop sind keine Traglasten angegeben. Das ist ein eindeutiger Hinweis darauf, dass die Hersteller keine zusätzlichen Ladungen eingeplant haben.
Geringe Zusatzladungen ($\lessapprox$100g) dürften jedoch bei allen Drohnen möglich sein, sofern eine entsprechende Halterung installiert werden kann. Ultraschallsensoren wiegen etwa 5g und sind für Kollisionserkennungen auf kurze Distanz nützlich. Eine andere Möglichkeit besteht in der Nutzung einer Tiefenkamera wie bspw. dem „Structure Sensor“. Die Kamera hat ein Gewicht von 95g[1] und kann somit ebenfalls montiert werden.
Dieses Kapitel gibt einen kurzen Überblick zur Aufgabenverteilung, um das gewählte Problem zu bearbeiten.

2.3. Aufgabenaufteilung

Für die Entwicklung von ArgosControl wurde die Projektgruppe zu Beginn in vier Gebiete mit jeweils drei Personen eingeteilt: die Pfadplanung, die Rekonstruktion, das Tracking und die Drohnen-Steuerung. Die Abbildung 2.2 und die folgende Erläuterung soll hierzu die initiale Idee hinter der Aufteilung begründen und das Ziel der Projekt-Gruppe (Kapitel 1.2) ergänzen.

Drohnen-Steuerung Die Drohne muss von der Software des externen Rechners gesteuert werden und es müssen zusätzlich angebundene Sensoren (Kapitel 3.3)

[1] `http://structure.io/support/what-are-the-structure-sensors-technical-specifications` Abruf: 23.05.2016 10:00

verwendbar sein. Weiterhin sind Daten von der Drohne, wie etwa Bilder und Sensor-Daten zu übermitteln (Kapitel 7.7), damit diese extern verarbeitet werden können. Die Personen, die dieser Gruppe zugeordnet sind, beschäftigen sich daher vorwiegend mit der Drohnen-Hardware.

Pfadplanung Zum Zweck der Erkundung muss ein Pfad geplant werden (Kapitel 5.4), sodass während des Flugs genügend Material für die Rekonstruktions-Phase gesammelt und Hindernissen ausgewichen wird. Gesteuert wird die Drohne daher vom Teilmodul der Pfadplanungs-Gruppe. Die Informationen über die Position und Hindernisse soll die Tracking-Gruppe bereitstellen.

Tracking Die von der Drohne gelieferten Daten sollten dazu verwendet werden, eine Position und Ausrichtung der Drohne zu bestimmen. Gleichzeitig ist es nötig Hindernisse zu erkennen, die bei der Pfadplanung berücksichtigt werden müssen (Kapitel 4).

Rekonstruktion In der Rekonstruktions-Phase (Kapitel 6) sollen die gesammelten Bild- und Tiefendaten verwendet werden um eine Karte der Umgebung aufzubauen und darzustellen.

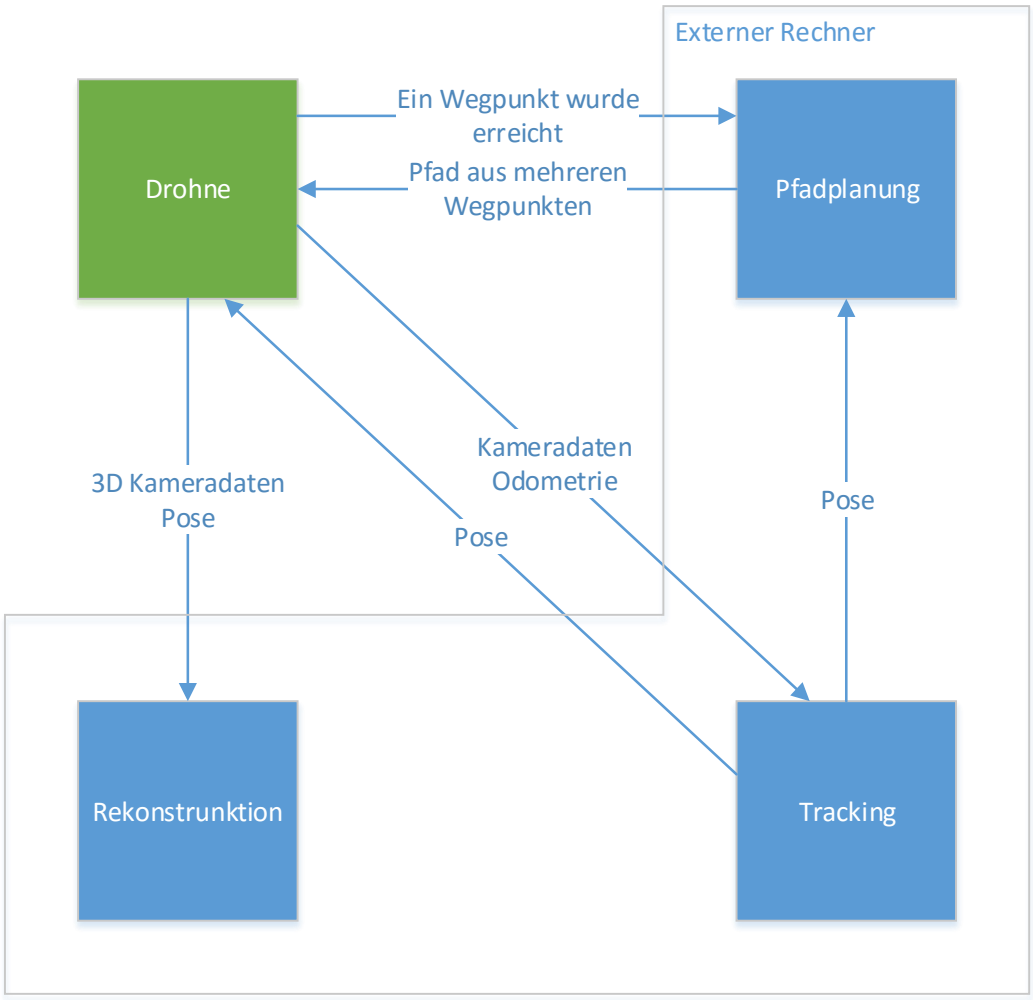


Abbildung 2.2.: Übersicht über die vier Aufgabenfelder des Projekts und die Kommunikation zwischen ihnen (dargestellt mit Pfeilen). Die Pfadplanung zusammen mit der Selbstlokalisierung findet auf einem externen Rechner statt und steuert die Drohne, die wiederum Software bereitstellt um ihre Daten an den externen Rechner zu übermitteln. Die Rekonstruktion verwendet die gesammelten Daten um eine Umgebungskarte zu konstruieren und darzustellen.

3. Drohnen

Dieses Kapitel beschreibt Drohnen, ihre Nutzungsmöglichkeiten im Allgemeinen und gibt einen Überblick über Sensoren, Aktoren und die notwendige Regelungstechnik.

3.1. Einleitung

Der Begriff Drohne beschreibt einen (halb-)autonomen Roboter. Die abstrakte Definition einer Drohne sieht im Wesentlichen drei Bestandteile vor:

Sensorik Aufnahme der Umwelt durch Sensoren, wie bspw. Kameras, Mikrofone oder Entfernungsmesser. Damit können aber auch Funkverbindungen für eine Befehlsschnittstelle sein.

Aktorik Die Manipulation der physikalischen Welt durch Aktoren wie Motoren, Lautsprecher, Scheinwerfer und ähnlichem. Auch die aktive Kommunikation mit beispielsweise einem Anwender wird von der Aktorik repräsentiert, wobei jedoch mindestens ein Aktor mit der physikalischen Welt interagieren muss.

Regeltechnik Durch die Regeltechnik werden Sensorik und Aktorik miteinander verbunden, indem Signale der Sensorik in Befehle für die Aktorik umgesetzt werden. Diese Umsetzung soll autonomes Verhalten simulieren.

3.1.1. Parrot Bebop Drone

Die Parrot Bebop Drone ist ein nur $33 \times 38 \times 3,6$cm großer, 420g schwerer Quadrocopter in X-Anordnung, d. h. eine Flugdrohne mit vier diagonal angeordneten Rotoren. Sie verfügt über alle für den autonomen Flug erforderlichen Sensoren, wie einem Drei-Achsen-Beschleunigungssensor, einem Drei-Achsen-Gyroskop und einem Magnetometer. Daneben stehen der Drohne unter anderem auch eine horizontale Frontkamera mit einer Auflösung von 14Megapixel für Bilder bzw. 1080p für Videos, eine vertikale Bodenkamera im QVGA-Format, ein auf Ultraschall basierender Höhensensor und ein GPS-Empfänger zur Verfügung. Die Drohne wird von einem

vollwertigen, eingebauten Rechner gesteuert. Eine Parrot P7 ARM Cortex A9 Dual-Core CPU mit 900MHz Taktfrequenz, 1GB RAM und 8GB Flash-Speicher führt eine vom Hersteller vorinstallierte Firmware auf Linux-Basis aus. Dieser Rechner ist als WLAN-Access-Point erreichbar, damit der Benutzer die Drohne steuern kann, und verfügt außerdem über einen USB2.0-Anschluss.

3.1.2. Parrot AR.Drone2

Wie die Bebop Drone ist auch die AR.Drone2 ein Quadrocopter in X-Anordnung des Herstellers Parrot. Diese Flugdrohne ist der Vorläufer der Bebop Drone und wurde in diesem Projekt vor allem deshalb eingesetzt, da ihre Hardware-Komponenten leichter zugänglich sind, so dass die Entwicklung trotz einiger technischen Einschränkungen einfacher war. Der AR.Drone2 verfügt über einen ähnlichen Umfang von Sensoren wie die Bebop Drone. Sie ist mit Verkleidung $52,5 \times 51,5$cm groß und wiegt 420g. Die Auflösung der horizontalen Frontkamera beträgt aber nur 720p, während die vertikale Bodenkamera wie in der Bebop Drone das QVGA-Format hat. Als Rechner ist eine mit 1GHz getaktete ARM Cortex A8 Single-Core CPU vorhanden, der 1GB DDR2-Arbeitsspeicher zur Seite stehen und die eine Firmware auf Basis von Linux 2.6.32 ausführt. Der Flash-Speicher beträgt nur 64MB, aber wie die Bebop Drone betreibt auch die AR.Drone2 einen WLAN-Access-Point und hat einen USB2.0-Anschluss.

3.2. Anwendungsbereiche

Drohnen verfügen über einige Eigenschaften, die sie in Anwendungsszenarien gegenüber herkömmlichen Methoden überlegen machen. Der Betrieb von programmierten Drohnen ist deutlich günstiger als die Nutzung gängiger Alternativen. Für die Vermessung von Industrieanlagen bestehen zwei Alternativen zu einer Drohne darin, ein Team von Ingenieuren mit Vermessungswerkzeugen das Gelände von Hand zu vermessen, oder die Vermessungen analog zur Drohne mit einem Helikopter durchzuführen.
Durch aktuelle Quadro- und Oktokopter ist ein präzises Anfliegen von Punkten im Raum möglich. Ergebnisse lassen sich so schnell und einfach reproduzieren. Während der Vermessung können zurückgelegte Strecken abgespeichert werden um bei weiteren Messungen die Ergebnisse mit neuen Messwerten zu verbessern. Mehrere Flüge können somit zu einem Ergebnis zusammengefasst werden.

Abbildung 3.1.: Parrot Bebop Drone
Die Abbildung stammt aus der *Parrot Bebop Drone Bedienungsanleitung* von Parrot SA [63].

Abbildung 3.2.: Parrot AR.Drone2
Die Abbildung stammt aus der *Parrot AR.Drone2.0 Bedienungsanleitung* von Parrot SA [62].

Ein weiterer Vorteil des Einsatzes von Drohnen ist das reduzierte menschliche Risiko, sollte der Einsatz in einem Gefahrengebiet stattfinden. Bspw. kann mithilfe einer Drohne in einem einsturzgefährdetem Haus nach einem Evakuierungsweg gesucht werden.

Auch der stark kontrollierte Einfluss auf die Umgebung kann von entscheidender Bedeutung sein. Während ein Helikopter einen starken Luftdruck nach unten aussendet, ist der von Quadro- und Oktokoptern ausgehende Luftdruck vernachlässigbar. Da Drohnen durch Sensoren auch nachträglich aufrüstbar sind, können diese in verschiedensten Bereichen eingesetzt werden.

3.3. Sensoren

Die Parrot Bebop besitzt verschiedene Sensoren, um Daten der Umgebung zu erhalten. Zu den Sensoren gehören ein Beschleunigungssensor, ein Gyroskop, ein Magnetometer, einen Höhensensor, einen Tiefensensor sowie eine horizontale und eine vertikale Kamera[1] . Im Folgenden werden die Sensoren einzeln beschrieben.

Beschleunigungssensor und Gyroskop Zuerst werden die Sensoren beschrieben, welche zur Inertial Measurement Unit (IMU) gehören. Dazu zählt der Beschleunigungssensor und das Gyroskop. Der Beschleunigungssensor misst die Beschleunigung in eine Richtung anhand der Messung der Trägheitskraft einer verbauten Masse im Chip. Ein Gyroskop hingegen misst die Rotation, indem ein drehender Kreisel im Vergleich zur rotierenden Drohne gemessen wird. Wenn die Drohne rotiert, dann bleibt der Kreisel annähernd unbeeinflusst und der Unterschied der beiden Richtungen ist die Messgrundlage des Gyroskops. Beide sind in der Bebop als ein einzelner Chip verbaut. Dieser Chip liest Werte auf drei Achsen aus und kann zusätzlich externe Sensoren, wie das Magnetometer, über einen eingebauten Bus auslesen.

Magnetometer und Höhensensor Das verwendete Magnetometer basiert auf dem Hall-Sensor-Prinzip [66] und misst die magnetische Flussdichte. Dieser Kompass misst das Erdmagnetfeld und erhält dadurch die eigene Richtung im Vergleich zur magnetischen Nordrichtung. Der Hall-Sensor hat den Vorteil, dass auch beim Stillstand die magnetische Richtung gemessen werden kann. Genutzt werden diese Informationen beim rotieren der Drohne. In der Drohne wird ein Höhensensor verwendet, welcher anhand von Ultraschall den Abstand zum Boden misst. Die Bebop kann mit dem Ultraschall eine Höhe von bis zu 8m messen. Relevant sind die Höheninformationen bspw. wenn die Drohne landet. So kann eine abrupte und zu schnelle Landung vermieden werden.

Structure Sensor Der Structure Sensor ist ein Infrarot-Tiefensensor der Firma Occipital, Inc. [60]. Durch das geringe Gewicht von 95g und den Maßen von 119,2mm in der Breite, 28mm in der Tiefe und 29mm in der Höhe, kann der Sensor wie in Abbildung 3.3 auf die Drohne montiert werden, ohne starke Beeinträchtigungen während des Fliegens zu verursachen. In einem Bereich von 0,4m bis 3,5m werden Tiefendaten laut Hersteller erfasst. Mit einer Auflösung von 640x480 Pixeln werden 30 Frames pro Sekunde aufgenommen. Laut der

[1] `http://www.parrot.com/de/produkte/bebop-drone/` Abruf: 22.06.16 14:00

Produktbeschreibung [59] wird eine Präzision von 0,5mm auf 40cm (0,15%) Entfernung und 30mm auf 3m (1%) Entfernung erreicht. Der Sensor benutzt die Technik der Streifenprojektion [29]. Dazu wird ein Infrarotprojektor und eine Infrarotkamera verwendet. Zuerst strahlt der Projektor ein Muster von Streifen mit dunklen und hellen Infrarotlicht ab. Daraufhin nimmt die Kamera dieses Bild auf. Den Streifen wird bspw. ein Gray-Code zugeteilt, sodass eine spätere Zuordnung möglich ist. Im Anschluss werden Fourier-Transformationen angewendet, um eine eindeutige Rekonstruktion des Raums zu erhalten.

Horizontale und vertikale Kamera Die Bebop besitzt zwei weitere Kameras, welche mit Blickrichtung nach vorne (horizontal) und nach unten (vertikal) montiert sind. Die vertikale Kamera wird verwendet, um einen optischen Fluss (siehe Kapitel 4.2.3) aus den Bilddaten abzuleiten. Dadurch wird ein horizontaler Bewegungssensor realisiert. Des Weiteren wird ein dreidimensionaler Bewegungssensor aus den Daten aller Kameras abgeleitet.

Lage- und Posesensor Weitere indirekte Sensoren sind die Lage- und Posesensoren. Der Lagensensor berechnet anhand der IMU-Messdaten und ggf. durch das Magnetometer die aktuelle Lage im Raum. Wohingegen der Posesensor die Odometriedaten verwendet, welche in Kapitel 4.2 näher beschrieben werden.

3.4. Aktoren

Als Aktoren verfügt ein Quadrocopter über vier Rotoren, durch die alle Bewegungen realisiert werden. Hierbei werden sowohl der Schub als auch die Drehmomente der Rotoren eingesetzt, um den Quadrocopter steuern und fliegen zu können. Weitere Aktoren sind nicht erforderlich und nicht vorhanden.
Um stabil in der Luft stehen zu können, muss der Quadrocopter durch seine Rotoren einen Schub erzeugen, der zu einer waagerechten Lage führt. Dazu muss der Schub der Rotoren senkrecht und damit der Schwerkraft entgegen wirken. Dabei wird hier der Einfachheit halber völlige Windstille angenommen.
Zusätzlich darf kein Drehmoment von den Rotoren auf den Quadrocopter einwirken, da er sich andernfalls um seine Hochachse drehen würde. Dazu drehen zwei nebeneinander liegende Rotoren immer in die entgegengesetzte Richtung (d. h. zwei diagonal gegenüberliegende Rotoren drehen immer in dieselbe Richtung). Dies wird durch Abbildung 3.4 veranschaulicht, die einen Quadrocopter schematisch in Draufsicht

Abbildung 3.3.: Tiefensensor „Structure Sensor“ auf der Drohne montiert

darstellt: die rot gefärbten Rotoren A und C drehen sich im Uhrzeigersinn, während die blauen Rotoren B und D sich gegen den Uhrzeigersinn drehen, sodass sich die Drehmomente der Rotoren aufheben. Alle weiteren Flugbewegungen leiten sich durch relative Änderung der Schübe ab, wobei bei einer X-Anordnung der Rotoren folgendes gilt:

Um zu steigen, wird der Schub aller Rotoren erhöht. Das Sinken erfolgt analog durch Verringern des Schubs aller Rotoren.

Um nach links oder rechts um die eigene Achse zu drehen (zu „gieren“), wird eine Differenz im Drehmoment zwischen AC und BD erzeugt, d. h. die Drehzahl von AC erhöht und von BD verringert (Linksdrehung) oder die Drehzahl von BD erhöht und von AC verringert (Rechtsdrehung).

Um vorwärts oder rückwärts zu fliegen, wird die Drohne nach vorn oder hinten gekippt („nicken“), so dass der Schub nicht nur nach unten, sondern auch in die gewünschte Flugrichtung wirkt. Dazu wird das Schub-Verhältnis zwischen AB und DC angepasst.

Seitwärts zu fliegen geschieht analog zum Vorwärts/Rückwärts-Fliegen durch seitliches Kippen („rollen“).

Dabei erhält der Quadrocopter durch seine Sensoren (Gyroskope, Beschleunigungssensoren) ununterbrochen Rückmeldung über seine tatsächlichen Bewegungen. Dadurch werden alle Abweichungen von den gewünschten Bewegungen festgestellt und

[2] „Quadrokopter-X-H-Konfiguration“ von UlrichHeither - Eigenes Werk. Lizenziert unter CC BY-SA 3.0 über Wikimedia Commons)

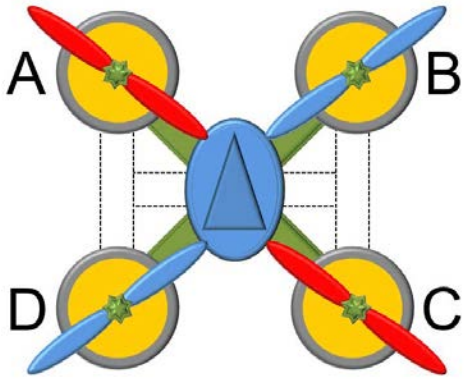


Abbildung 3.4.: Schematische Draufsicht eines Quadrokopters mit farblich gekennzeichneter Drehrichtung der vier Rotoren A bis D[2].

der Quadrocopter kann sein Flugverhalten anpassen, d. h. stabil fliegen. Der grobe Zusammenhang dieser Komponenten ist in Abbildung 3.5 veranschaulicht. Das „eingebettete System“ ist hierbei die vom Hersteller vorgegebene Firmware und die „Aktoren“ sind die Rotoren.

3.5. Steuerung der Rotoren

Da die Drohne durch die Rotoren gesteuert wird, entstehen die gewollten Flugbewegungen aus dem Schub der Rotoren. Daher ist für die Stabilisierung und Steuerung eine Einstellung des Schubs erforderlich. Offensichtlich steht der Schub im direkten Verhältnis zur Drehzahl der Rotoren und dem Blattwinkel, so dass ein Betrieb der Motoren mit einstellbarer Drehzahl möglich ist. Die Rotoren der hier verwendeten Quadrocopter sind starr. Daher muss der Blattwinkel nicht weiter berücksichtigt werden.

Um die zu gewünschte Drehzahl zu erreichen, soll den Motoren eine bestimmte Leistung bereitgestellt werden. Die tatsächlich resultierende Drehzahl ist jedoch noch von verschiedenen weiteren Faktoren abhängig, wie bspw. dem Ladestand des Akkus, der die zu einem bestimmten Zeitpunkt zur Verfügung stehenden Maximalleistung bestimmt, und den Toleranzen des Bauteils selbst.

Um trotz dieser Faktoren zu der gewünschten Drehzahl zu gelangen, muss der Motor überwacht werden. Dazu wird durch ein geeignetes Programm (oder auch eine

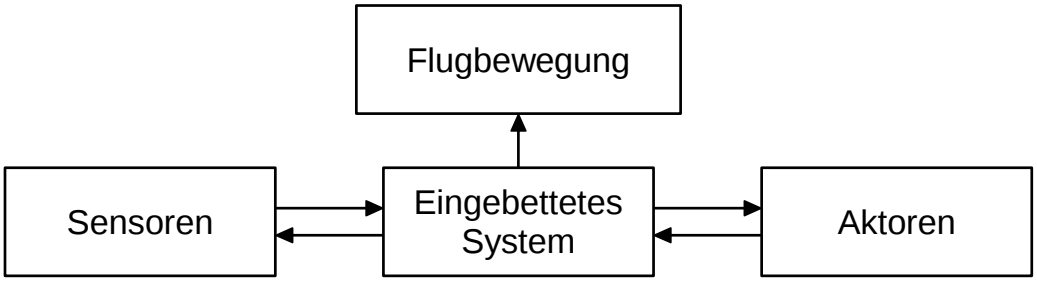


Abbildung 3.5.: Grobe Unterteilung der Komponenten einer Drohnensteuerung

elektrische Schaltung) mit Hilfe eines Drehzahlsensors festgestellt, ob der Sollwert tatsächlich eingehalten wird. Ist dies nicht der Fall, kann die tatsächliche Drehzahl durch Erhöhen oder Verringern der Leistung an die gewünschte Drehzahl angeglichen werden. Diese Überwachung wird technisch als Rückkopplung bezeichnet und stellt den entscheidenden Unterschied zwischen einer rückkopplungsfreien Steuerung und einer Regelung dar.

3.6. Regelungstechnik

Der folgende Abschnitt beschreibt einige Grundlagen aus der technischen Informatik und der Regelungstechnik, welche die im letzten Abschnitt beschriebenen Zusammenhänge abstrahieren und formalisieren sollen. Diese Grundlagen dienen als Basis für die in Kapitel 7 weiter ausgeführte Implementierung einer Drohnen-Firmware.

3.6.1. Regelung allgemein

Verallgemeinert man das in Abschnitt 3.5 entwickelte Konzept einer Regelung, erhält man die im Blockschaltbild in Abbildung 3.7 gezeigte grundlegende Struktur eines Regelkreises, wie sie in der Regelungstechnik definiert ist (siehe [76]).
Eine „Regelstrecke“ ist der Prozess, der geregelt werden soll. Dieser kann bspw. ein Motor der Drohne sein, der mit einer vorgegebenen Drehzahl drehen soll. Die dazugehörigen Sensoren der Drohne werden durch das „Messglied“ repräsentiert (am Beispiel der Motoren wären es Drehzahlsensoren). Durch den unteren Verbinder im Blockschaltbild wird die Rückkopplung realisiert, die einen Regler von einer Steuerung unterscheidet: Die Summationsstelle ganz links im Blockschaltbild, die $e(t) = w(t) - x(t)$ berechnet, ermittelt durch Subtraktion die Regelabweichung

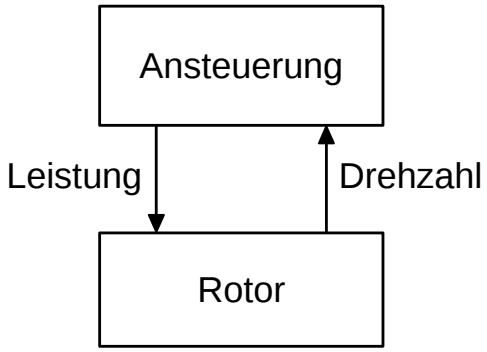


Abbildung 3.6.: Drehzahlregelung der Rotoren durch eine Ansteuerung, welche die Leistung an die gemessene tatsächliche Drehzahl anpasst

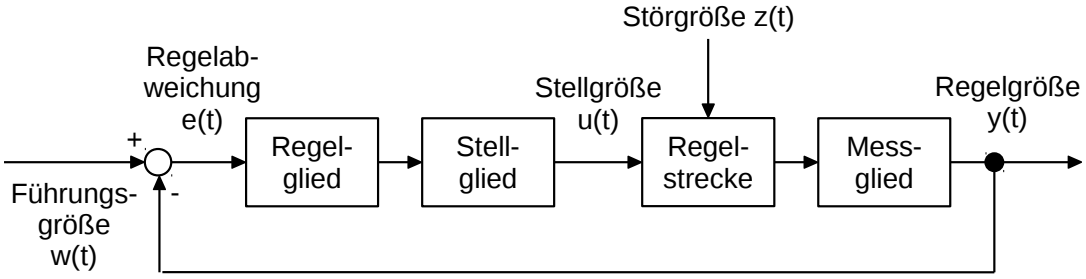


Abbildung 3.7.: Blockschaltbild eines Regelkreises, verkürzt nach [76]. Die Summationsstelle ganz links, die $e(t) = w(t) - x(t)$ berechnet, ermittelt durch Subtraktion die Regelabweichung $e(t)$, welche die Differenz zwischen Führungsgröße $w(t)$ (Sollwert) und Regelgröße $x(t)$ (Istwert) ist. Dies ermöglicht es dem „Regelglied“, der Störgröße $z(t)$ dauerhaft entgegenzuwirken.

$e(t)$, welche die Differenz zwischen Führungsgröße $w(t)$ (Sollwert) und Regelgröße $x(t)$ (Istwert) ist. Dies ermöglicht einem Regler, der Regelabweichung in geeigneter Form entgegenzuwirken und sie damit so klein wie möglich zu halten, d. h. dauerhaft einen Istwert beizubehalten, der trotz der Störgröße $z(t)$ dem Sollwert möglichst nahe kommt.

Das „Regelglied“ ist der eigentliche Regler. Mögliche Regelglieder, die hier betrachtet werden, sind der Proportionalregler (kurz P-Regler) oder der PID-Regler aus Abschnitt 3.6.6. Der P-Regler verstärkt als einfachster Linearregler lediglich die Regelabweichung um einen festgelegten Faktor, so dass die Stellgröße der Regelabweichung immer entgegenwirkt. Bereits dadurch wird der gewünschte Effekt einer Regelung erreicht.

3.6.2. Cyber-Physical Systems: hardware-in-the-loop

Alle oben beschriebenen Regelglieder können in Software realisiert werden, sofern der dazu eingesetzte Rechner bestimmte Voraussetzungen erfüllt. Parrot hat die Regelglieder in beiden Drohnen, der AR.Drone2 und der Bebop Drone, auf diese Weise implementiert.

Einen Rechner, der dezidiert für die Steuerung eines Gerätes eingesetzt wird und der vom Benutzer nicht unabhängig vom Gerät oder zu anderen Zwecken eingesetzt werden kann (oder einsetzbar sein soll), ist ein eingebettetes System. Abbildung 3.8 zeigt schematisch, wie ein eingebettetes System typischerweise in seine Umgebung integriert ist. Der Begriff „System“ ist hierbei umgangssprachlich gemeint. Von besonderer Bedeutung bei eingebetteten Systemen sind die Echtzeitanforderungen, die, auf ein Regelglied bezogen, garantieren, dass die zeit- und wertdiskrete Verarbeitung im Rechner in vorgeschriebenen Zeiträumen abgeschlossen wird. Wird ein Regelglied in Software mithilfe eines Rechners realisiert, stellt dies ein eingebettetes System dar.

Abbildung 3.8 zeigt, wie die Rotorregelung in einer Flugdrohne umgesetzt werden kann. Als eingebettetes System („Informationsverarbeitung“) kommt ein Mikrocontroller zum Einsatz, um eine hohe Zuverlässigkeit zu gewährleisten. Dies wäre durch eine Integration der Regelung in ein unter Linux laufendes System-on-a-Chip (SoC) zwar kostengünstiger, würde aber deutlich höhere Anforderungen an das Betriebssystem stellen. Das „Stellglied“ aus Blockschaltbild 3.7 wird hier durch den „D/A-Wandler“ und die „Aktuatoren“ repräsentiert. Dies impliziert eine analoge Ausgabe, aber auch eine digitale Ausgabe ist beispielsweise durch eine Kombination aus Pulsweitenmodulation (PWM) und Glättung durch die Trägheit der Motoren möglich. Die sich daraus ergebende Wirkung auf die Umgebung, d. h. die tatsächliche Drehzahl der Rotoren, wird durch die „Sensoren“ und den „A/D-Wandler“ erfasst, die zusammen die „Sensoren“ aus Blockschaltbild 3.7 bilden. Eine mögliche technische Realisierung wäre ein Hall-Sensor, der die Umdrehungen durch Impulse an die „Informationsverarbeitung“ meldet.

In der AR.Drone2 und der Bebop Drone ist die Rotorregelung in vergleichbarer Weise vom Hersteller realisiert worden. In der Bebop Drone kommt als Mikrocontroller ein Cypress Motor Controller mit eigener Firmware (Parrot BLDC) zum Einsatz. Die Motoren verwenden in dieser Drohne jedoch eine komplexere, mehrphasige Ansteuerung. Alle anderen, höheren Regelungsaufgaben wurden vom Hersteller in vergleichbarer Weise in Software gelöst, wobei diese jedoch als Prozesse im echtzeitfähigen Linux auf dem SoC der Drohne ablaufen.

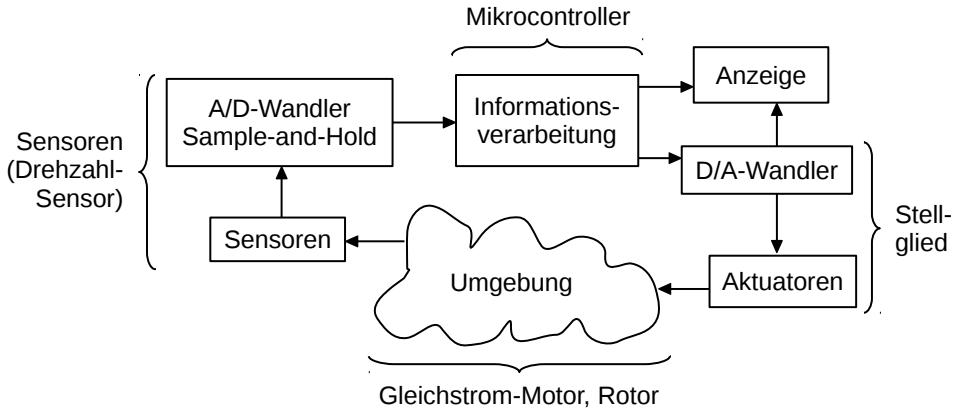


Abbildung 3.8.: Cyber-Physical Systems: hardware-in-the-loop, nach [52]. Ein Beispiel für eine Drehzahlregelung durch ein eingebettetes System.

3.6.3. Regelung der Lage und Bewegung

Eine weitere, auf den ersten Blick weniger technische Anwendungsmöglichkeiten für einen Regler ist die Lageregelung der Drohne, aus der sich die Bewegung im Raum ergibt. Auch hier sind die gewünschten Richtungs- und Winkelbeschleunigungen Führungsgrößen, deren Einhaltung durch einen Regler implementiert werden kann bzw. bei denen eine Implementierung zur Einhaltung als Regler aufgefasst werden kann.

Abbildung 3.9 zeigt das Blockschaltbild eines solchen Regelkreises, bei dem ein Proportionalregler (kurz P-Regler) eingesetzt wird. Als Führungsgröße erhält der Regler Lagevorgaben, die er in Abhängigkeit von der derzeitigen Lage der Drohne in Änderungen der Rotorschübe übersetzt. Diese Änderungen haben Bewegungen zur Folge, die aber teilweise auch durch externe Störgrößen wie Wind hervorgerufen werden. Die Sensoren der Drohne erfassen die tatsächliche Lage der Drohne und melden diese als Regelgröße zurück. Durch diese Rückkopplung passt sich der Regler jederzeit sowohl an das konkrete Verhalten der Drohne als auch an das gerade vorliegende Ausmaß der Störgrößen an.

3.6.4. Regler-Kaskadierung

Am Beispiel der Bewegung des Quadrocopters durch einen Regelkreis (Abschnitt 3.6.3) und der darin enthaltenen Drehzahlregelung der Rotoren durch einen zweiten, untergeordneten Regelkreis (Abschnitt 3.5), ist erkennbar, dass Regler eine Hierarchie bilden können, d. h. ein Regler kann einen anderen Regler steuern. Da jeder

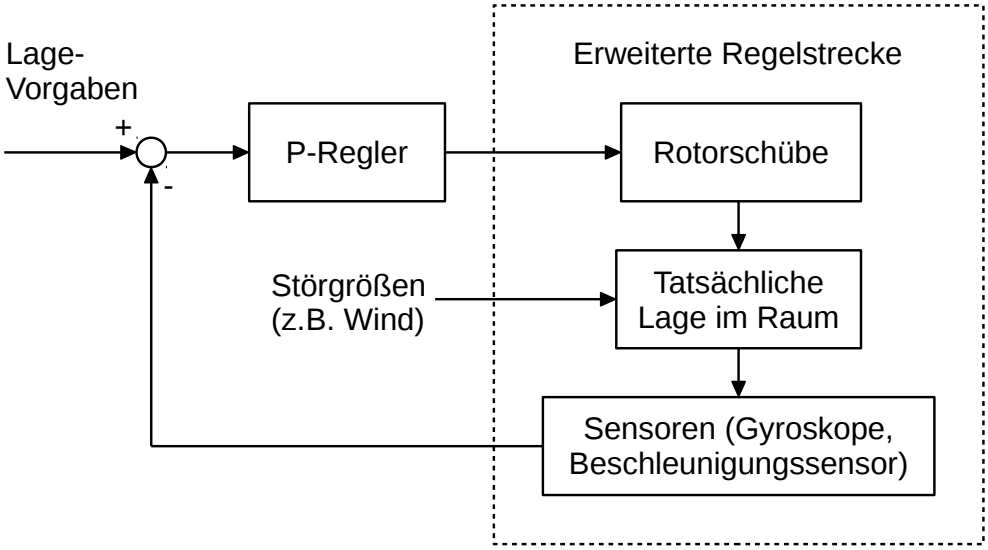


Abbildung 3.9.: Das Blockschaltbild eines Regelkreises für die Bewegung (Lage) einer Drohne mit einem P-Regler. Hierbei wurden die Regler für die Lage und für die Rotordrehzahlen zu einem Regler zusammengelegt.

Regelkreis ein abstraktes System mit Eingang und Ausgang bildet und die Regelstrecke als Teil des Regelkreises wiederum ein System ist, kann sie damit einen eigenständigen Regelkreis enthalten.
Enthält eine Regelstrecke einen weiteren Regelkreis, kann durch ein Zusammenführen der ursprünglich getrennt betrachteten Systeme zu einem einzelnen, größeren Regler das Verständnis verbessert werden. Systemgrenzen sind frei definierbar und sollten immer so gezogen werden, dass ein für die Betrachtung des Problems sinnvoller Abstraktionsgrad entsteht.
Diese Zusammenführung der Konzepte der Drehzahlregelung und der Lageregelung ist als kaskadierter Regelkreis in Blockschaltbild 3.10 abgebildet.

3.6.5. Navigation

Eine weitere Anwendungsmöglichkeit der Regelung ist die Navigation: ein Kurs wird als Führungsgröße vorgegeben, um einen bestimmten Zielort zu erreichen.
Der in den vorangegangenen Beispielen verwendete Proportionalregler wirkt dabei der jeweils vorliegenden Kursabweichung entgegen. Dieser Regler minimiert zwar den Fehler, „lebt“ jedoch auch von ihm. Dadurch bleibt immer eine geringfügige Abweichung bestehen. Betrachtet man den in Abbildung 3.11b gezeigten Verlauf eines hypothetischen Kurses in Form der Richtung φ (die gestrichelte Linie) als Regel-

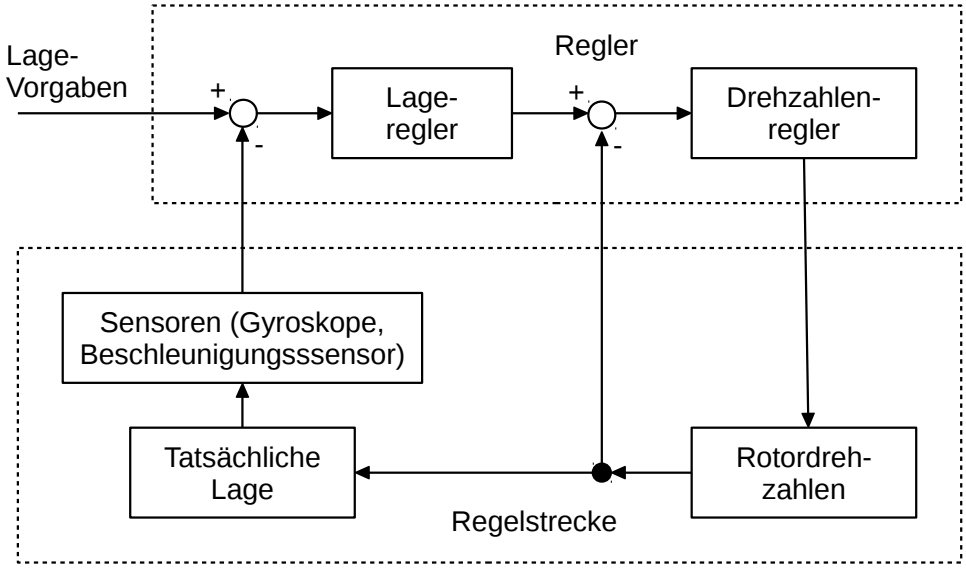


Abbildung 3.10.: Blockschaltbild eines kaskadiertern Regelkreises, der sowohl die Lage als auch die Drehzahlen der Rotoren regelt.

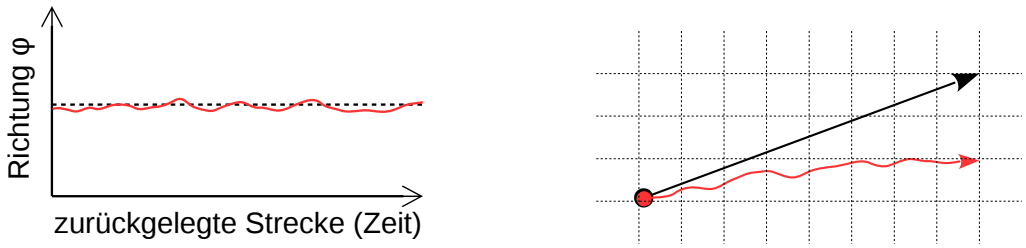
größe über die Zeit, ist der P-Regler (rote Kurve) zwar der Führungsgröße gefolgt, tendierte jedoch dazu, die Führungsgröße eher zu unterschreiten als zu überschreiten.

An der resultierenden Flugbahn in Abbildung 3.11b ist deutlich erkennbar, dass sich diese verbliebenen Regelabweichungen aufsummiert haben und der Regler trotz minimierter Kursabweichung das Ziel nicht erreicht hat. Die Navigation erfordert mehr als nur die Minimierung der Regelabweichung.

Eine Lösung hierfür besteht darin, neben der aktuell vorliegenden Regelabweichung auch die sich aufsummierten Abweichungen aus der Vergangenheit bei der Berechnung der Stellgröße zu berücksichtigen, so dass der Regler vorsätzlich Abweichungen von der Führungsgröße erzeugt, um die über die Zeit entstehende Gesamtabweichung gering zu halten. Da die Gesamtabweichung zu jeder Zeit minimiert wird, kann die dafür benötigte Regelabweichung sehr gering bleiben.

Eine ganz andere, für die Navigation ebenfalls relevante Erweiterung besteht darin, auch die Entwicklung der Regelabweichung einfließen zu lassen. Je schneller die Regelgröße aufgrund einer Störgröße von der Führungsgröße abweicht, desto stärker sollte der Regler der Abweichung entgegenwirken, um Änderungen möglichst schnell auszugleichen.

Die technische Umsetzung dieser Konzepte ist der PID-Regler.



(a) geregelter Flug (Kurs) **(b)** resultierende Flugbahn

Abbildung 3.11.: Hypothetisches Beispiel für Navigation mit einem P-Regler

3.6.6. PID-Regler

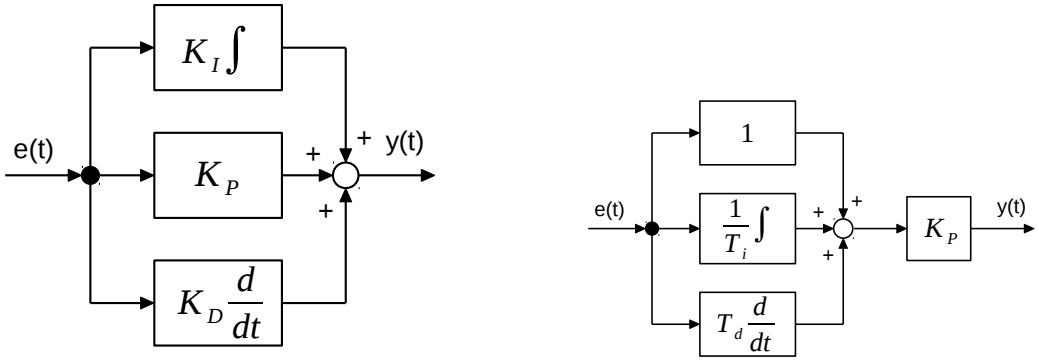
Beim PID-Regler (proportional, integral, derivate) wird die Stellgröße als Summe der proportional verstärkten Regelabweichung, dem Integral aller Regelabweichungen der Vergangenheit und der Ableitung der derzeitigen Regelabweichung gebildet [76]. Der PID-Regler ist ein vielseitig einsetzbarer Standardregler. Teilformen wie der PI- oder der PD-Regler sind dabei auch möglich. Der zuerst vorgestellte P-Regler kann als Sonderfall eines PID-Reglers aufgefasst werden.

Der im Blockschaltbild in Abbildung 3.12a gezeigte Zusammenhang zwischen Eingangssignal $e(t)$ und Ausgangssignal $y(t)$ kann durch die folgende lineare Differenzialgleichung beschrieben werden:

$$y(t) = K_P \cdot e(t) + K_I \int_0^t e(\tau)\,\mathrm{d}\tau + K_D \cdot \dot{e}(t) \tag{3.1}$$

$$\Leftrightarrow \frac{1}{K_D}\dot{y}(t) = \ddot{e}(t) + \frac{K_P}{K_D}\dot{e}(t) + \frac{K_I}{K_D}e(t)\,. \tag{3.2}$$

Hierbei ist $\dot{e}(t)$ die erste Ableitung von $e(t)$ und $\ddot{e}(t)$ die zweite. K_P gibt die Verstärkung vor, während K_I den Einfluss der integrierten dauerhaften Regelabweichung und K_D den Einfluss der Änderungsrate der Regelabweichung bestimmen. Dementsprechend kann ein PID-Regler durch $K_I = 0$, $K_D = 0$ bzw. $K_I = K_D = 0$ zu einem PD-, PI- oder P-Regler gemacht werden. Üblich ist jedoch die Darstellung als technisches Blockschaltbild (vergleiche Abbildung 3.12b), bei der $T_i = \frac{K_P}{K_I}$ und $T_d = \frac{K_D}{K_P}$ gelten. Diese Umformung erleichtert die manuelle Einstellung des Reglers.



(a) mathematisch einfache Form **(b)** technisch übliche Form

Abbildung 3.12.: Blockschaltbilder des PID-Reglers in zwei verschiedenen Darstellungen. Blockschaltbild (a) zeigt den mathematischen Zusammenhang zwischen Eingangs- und Ausgangssignal, während (b) die technisch übliche Form zeigt, bei der die Reglerparameter anders ausgedrückt werden.

Hier gilt die Gleichung

$$y(t) = K_P \left(e(t) + \frac{1}{T_i} \int_0^t e(\tau)\, \mathrm{d}\tau + T_d \cdot \dot{e}(t) \right). \tag{3.3}$$

K_P gibt dabei weiterhin die Verstärkung des Reglers vor. T_i ist die Nachstellzeit und T_d die Vorhaltezeit, mit denen das zeitliche Verhalten des Reglers an die Regelstrecke angepasst werden kann. Zwar können diese Reglerparameter mithilfe eines mathematischen oder simulatorischen Modells der Regelstrecke ermittelt werden, jedoch ist eine einfache experimentelle Einstellung häufig ausreichend.

4. Selbstlokalisation und Erkennung von Hindernissen

Dieses Kapitel beschäftigt sich mit Verfahren zur Berechnung der Position und Orientierung als Tupel aus Koordinaten in einem globalen Koordinatensystem und Winkel, relativ zu den Achsen des Koordinatensystems. Dabei entstehen außerdem Informationen über Hindernisse, welche als Volumina repräsentiert gespeichert werden. Im Abschnitt 4.1 wird dafür zunächst erläutert, wie die Algorithmen und berechneten Daten grob in einen Gesamtzusammenhang gebracht werden können. Aufbauend auf diesem Überblick über die Kombination der Verfahren, wird im Abschnitt 4.2 der Teil der Berechnungen, welcher auf der Drohne ausgeführt wird, thematisiert und anschließend in Abschnitt 4.3 der Teil der Berechnungen, welcher auf einem externen Server ausgeführt wird, erläutert.

4.1. Einleitung

Die Selbstlokalisation bildet eines der elementaren Module des Projektes. Sie dient dazu dem Programm und insbesondere der Pfadplanung während des Fluges zu assistieren, indem eine Schätzung der momentanen Pose der Drohne berechnet wird. Eine Pose setzt sich dabei aus einem dreidimensionalen Vektor, der die Position der Drohne in einem globalen Koordinatensystem angibt, und wahlweise einer Rotationsmatrix oder einem Quaternion zusammen, mit dem die Drehung der Drohne relativ zum globalen Koordinatensystem beschrieben wird. Als Nebenprodukt der Selbstlokalisation, entstehen Informationen über den Aufenthaltsort anderer Objekte im Raum, die als Hindernisse bei der Pfadplanung berücksichtigt werden müssen. Die Implementierung dieses Moduls ist dabei grundlegend in zwei Blöcke unterteilt: Das Large-Scale Direct Monocular SLAM (LSD-SLAM) (siehe Abschnitt 4.3), welches auf einem Rechner ausgeführt wird und aus dem Videostream der Frontkamera der Drohne sowohl die Umgebung als (Tiefen-)Bilder katalogisiert, als auch die Position der Drohne bestimmt und die Odometrie, die auf der Drohne ausgeführt wird und

aus den eingebauten Sensoren und der Bodenkamera eine Schätzung der Position berechnet, die sowohl als Grundlage für die Berechnungen im LSD-SLAM verwendet werden kann, als auch als Ersatzlösung fungiert, sollte das LSD-SLAM ausfallen. Da die Odometrie allerdings mit zunehmender Ungenauigkeit rechnet, während das LSD-SLAM sogar dazu in der Lage ist Stellen im Raum wiederzuerkennen und auf diese Art eine Korrektur vorhergegangener Fehler vorzunehmen, werden die Ergebnisse des LSD-SLAM zurück in die Odometrie gespeist, um den zunehmenden Fehler in der Odometrie zu eliminieren. Neben der unterschiedlichen Qualität der erzeugten Posenschätzung, ist ein weiterer wichtiger Unterschied, dass die Odometrie für jeden Datenpunkt aus den Sensoren der Drohne eine Pose berechnet, während das LSD-SLAM nur für jedes verarbeitete Bild des Videostreams eine Pose ausgibt, wodurch die Posenschätzung der Odometrie stets aktueller ist.

4.2. Odometrie

Die Odometrie ist der Teil der Selbstlokalisation, der unmittelbar auf der Drohne ausgeführt wird. Die Notwendigkeit für die Ausführung auf der Drohne entspringt dabei mehreren Faktoren. Zum einen werden die Daten, aus denen die Odometrie berechnet wird, von den Sensoren der Drohne mit 400Hz und der Bodenkamera der Drohne mit 30Hz generiert. Eine Übertragung der Daten würde zu einer zusätzlichen Belastung der Netzwerkkommunikation führen. Zum anderen werden ausschließlich von der Flugsteuerung der Drohne selbst Echtzeitinformationen über die Position der Drohne benötigt. Für alle anderen Komponenten genügt die langsamere Frequenz des LSD-SLAMs, inklusive dessen Zeitversatz durch Auswertung von Bildern.
Die Odometrie selbst ist in mehrere Komponenten unterteilt:

Verarbeitung der direkten Sensordaten (Drehmoment und Beschleunigung aus Gyroskopen und Akzelerometern)

Verarbeitung der indirekten Sensordaten (Geschwindigkeit aus dem optischem Fluss aus den Bodenkamerabildern)

Speicherung der berechneten Posen an Schlüsselpunkten, die rückwirkend korrigiert werden können

Da die Sensordaten rauschbehaftet sind, findet vor deren Verarbeitung eine Filterung mittels Kalman-Filter und einem aus der Fourier-Transformation hergeleiteten Filter statt, der im Folgenden als Fourier-Filter bezeichnet werden wird.

In diesem Abschnitt werden zunächst die Komponenten zur Filterung und Generierung von Daten erläutert und anschließend auf deren Komposition und Speicherung eingegangen.

4.2.1. Kalman-Filter

In diesem Abschnitt wird auf Grundlage der Arbeit von Welch und Bishop [77] beschrieben, wie der Kalman-Filter zur Selbstlokalisation eingesetzt werden kann. Zunächst wird der diskrete Kalman-Filter betrachtet und anschließend zum erweiterten Kalman-Filter übergegangen.

Der diskrete Kalman-Filter

Der Lokalisation mit dem Kalman-Filter liegt die Idee zugrunde, zunächst die nächste Pose der Drohne zu schätzen bzw. vorauszuberechnen und diesen im nächsten Zeitschritt mit aufgenommenen Messwerten zu korrigieren. Die Funktionen des Kalman-Filters sind daher in die zwei Phasen Vorhersage (Zeitupdate) und Korrektur (Messungsupdate) unterteilt.
Ausgehend von der Pose $\mathbf{x}^{(k)}$ zum Zeitpunkt k lässt sich die nächste Pose $\mathbf{x}^{(k+1)}$ aus den in der Pose enthaltenen Daten über die aktuelle Position und den ankommenden Steuerimpulsen $\mathbf{u}^{(k)}$ berechnen. Unter Berücksichtigung des unbekannten Fehlers $\mathbf{w}^{(k)}$, der bei der Bewegung der Drohne auftreten kann, ist

$$\mathbf{x}^{(k+1)} = \mathbf{A}_k\mathbf{x}^{(k)} + \mathbf{B}_k\mathbf{u}^{(k)} + \mathbf{w}^{(k)}. \tag{4.1}$$

Dabei sind $\mathbf{A}_k$ und $\mathbf{B}_k$ Überführungsmatrizen, die benutzt werden, um aus der Pose $\mathbf{x}^{(k)}$ bzw. den Steuerimpulsen $\mathbf{u}^{(k)}$ die neue Pose zu berechnen [77].
Ebenfalls wichtig für die Lokalisation sind die Messwerte $\mathbf{z}^{(k)}$ der Drohne. Darunter zu verstehen, sind durch Sensoren, wie beispielsweise einem Ultraschallsensor, aufgezeichnete Daten. Diese lassen sich aus der Pose der Drohne, folglich der Position und Ausrichtung, als

$$\mathbf{z}^{(k)} = \mathbf{H}_k\mathbf{x}^{(k)} + \mathbf{v}^{(k)} \tag{4.2}$$

schätzen. Dabei beschreibt die Matrix $\mathbf{H}_k$ den Zusammenhang zwischen Pose und Messung und $\mathbf{v}^{(k)}$ ist der bei den Messungen auftretende Fehler [77].
Das Problem an dieser Stelle sind die beiden unbekannten Fehlerwerte $\mathbf{w}^{(k)}$ und $\mathbf{v}^{(k)}$,

die als unabhängig und mit Normalverteilungen

$$p(w) \sim N(0, Q), \tag{4.3}$$

$$p(v) \sim N(0, R) \tag{4.4}$$

angenommen werden. Um diese zu umgehen, wird zunächst eine A-Priori-Pose

$$\tilde{\mathbf{x}}^{(k+1)} = \mathbf{A}_k \mathbf{x}^{(k)} + \mathbf{B}_k \mathbf{u}^{(k)} \tag{4.5}$$

eingeführt, der den Fehlerwert $\mathbf{w}^{(k)}$ ignoriert. Dieser wird in der Vorhersagephase des Filters berechnet und anschließend in der Korrekturphase mit dem aufgenommenen Messwert $\mathbf{z}^{(k)}$ zur A-Posteriori-Pose

$$\hat{\mathbf{x}}^{(k)} = \tilde{\mathbf{x}}^{(k)} + \mathbf{K}_k(\mathbf{z}^{(k)} - \mathbf{H}_k \tilde{\mathbf{x}}^{(k)}) \tag{4.6}$$

korrigiert. Dies ist in Abbildung 4.1 dargestellt. Der Wert $(\mathbf{z}^{(k)} - \mathbf{H}_k \tilde{\mathbf{x}}^{(k)})$ ist die Abweichung zwischen tatsächlicher und geschätzter Messung und $\mathbf{K}_k$ ist der Kalman-Faktor, der im Weiteren näher erklärt wird [77].
Der Kalman-Faktor

$$\mathbf{K}_k = \frac{\mathbf{P}_k^- \mathbf{H}_k^T}{\mathbf{H}_k \mathbf{P}_k^- \mathbf{H}_k^T + \mathbf{R}_k} \tag{4.7}$$

ist ein Gewichtungsfaktor, der in Abhängigkeit von den Fehlerkovarianzen $\mathbf{P}_k^-$ der A-Priori-Pose und $\mathbf{R}_k$ der Messung festlegt, wie stark die Messungsabweichung $(\mathbf{z}^{(k)} - \mathbf{H}_k \tilde{\mathbf{x}}^{(k)})$ bei der Korrektur der Pose ins Gewicht fällt. Für den Fall, dass der Fehler der geschätzten A-Priori-Pose gegen Null geht, gilt

$$\lim_{\mathbf{P}_k^- \to 0} \mathbf{K}_k = 0 \tag{4.8}$$

und die Messungsabweichung geht dementsprechend schwächer in die korrigierte A-Posteriori-Pose ein. Geht hingegen der Fehler der Messung gegen Null, so gilt

$$\lim_{\mathbf{R}_k^- \to 0} \mathbf{K}_k = \mathbf{H}_k^{-1} \tag{4.9}$$

und folglich wird die Messungsabweichung stärker gewichtet. Dadurch wird bei der Berechnung der A-Posteriori-Pose erreicht, dass je nach dem ob den Messwerten oder der A-Priori-Pose mehr vertraut wird, diese auch stärker berücksichtigt werden bzw. wird [77].
In der Abbildung 4.1 ist eine vollständige Übersicht über die Funktionsweise des

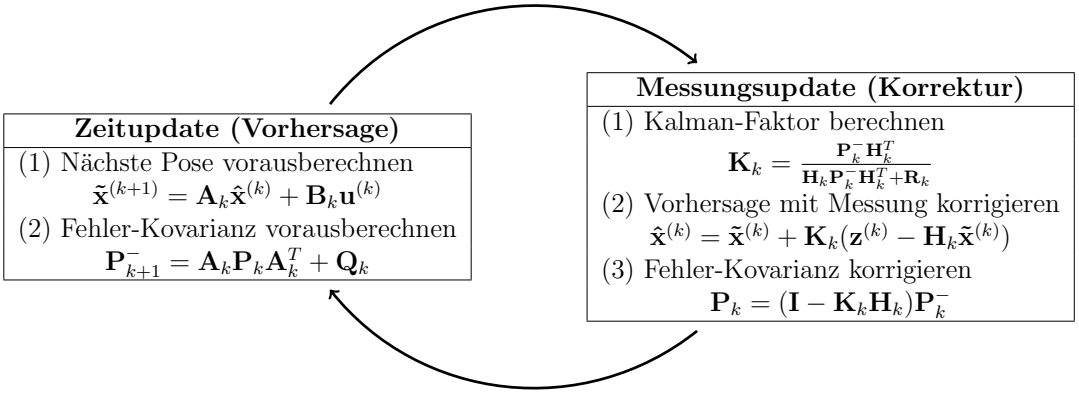


Abbildung 4.1.: Übersicht über die Funktionsweise des Kalman-Filters (vergleiche [77]). Im Vorhersage-Schritt wird die nächste Pose $\tilde{\mathbf{x}}^{(k+1)}$ mit Hilfe der vorherigen Pose $\hat{\mathbf{x}}^{(k)}$ und des Steuerimpulses $\mathbf{u}^{(k)}$ vorausberechnet und anschließend im Korrektur-Schritt zu $\hat{\mathbf{x}}^{(k)}$ korrigiert, sobald Messdaten $\mathbf{z}^{(k)}$ vorliegen.

Kalman-Filters zu sehen. Bei der Vorhersage wird zunächst die A-Priori-Pose und anschließend die zugehörige Fehlerkovarianz berechnet. Der Parameter $\mathbf{Q}_k$ ergibt sich aus der Normalverteilung des Fehlers (siehe Gleichung 4.3). Bei der Korrektur im nächsten Zeitschritt wird zuerst der Kalman-Faktor $\mathbf{K}_k$ berechnet, der anschließend dazu benutzt wird, die A-Priori-Pose $\tilde{\mathbf{x}}^{(k+1)}$ zur A-Posteriori-Pose $\hat{\mathbf{x}}^{(k)}$ zu korrigieren. Danach muss nur noch die Fehlerkovarianz korrigiert werden. Auf diese Weise funktioniert ein Position Tracking mit einem Kalman-Filter.

Der erweiterte Kalman-Filter

Der diskrete Kalman-Filter bietet eine Lösung für die Selbstlokalisation, solange die Beziehung zwischen Messung und Pose linear ist. Zur Lösung des Problems einer nicht linearen Beziehung wurde der erweiterte Kalman-Filter [77] entwickelt, welcher in diesem Abschnitt vorgestellt wird.
Die Idee für den erweiterten Kalman-Filter ist es, die bisher verwendeten linearen Übergangsmatrizen durch nicht lineare Funktionen zu ersetzten. Aus der Gleichung 4.1 für die Berechnung der nächsten Pose entsteht dadurch die Gleichung

$$\mathbf{x}^{(k+1)} = f(\mathbf{x}^{(k)}, \mathbf{u}^{(k)}, \mathbf{w}^{(k)}). \tag{4.10}$$

Dabei wurden die Matrizen $\mathbf{A}_k$ und $\mathbf{B}_k$ durch die Funktion f ersetzt. Auf dieselbe Weise wird die Gleichung 4.2 für die Messwerte

$$\mathbf{z}^{(k)} = h(\mathbf{x}^{(k)}, \mathbf{v}^{(k)}) \tag{4.11}$$

modifiziert. Dabei wurde die Überführungsmatrix $\mathbf{H}_k$ durch die nicht lineare Funktion h ersetzt. Da, wie auch zuvor, die Fehlerwerte $\mathbf{w}^{(k)}$ und $\mathbf{v}^{(k)}$ unbekannt sind, werden für die Vorhersage die fehlerfreie Pose

$$\tilde{\mathbf{x}}^{(k+1)} = f(\mathbf{x}^{(k)}, \mathbf{u}^{(k)}, 0) \tag{4.12}$$

berechnet und die fehlerfreien Messwerte $h(\mathbf{x}^{(k)}, 0)$ genutzt. Bei der Korrektur der geschätzten Pose wird, wie schon in Gleichung 4.6, der Kalman-Faktor zur Gewichtung der Messungsabweichung benutzt. Die korrigierte Pose ergibt sich daher zu

$$\hat{\mathbf{x}}^{(k)} = \tilde{\mathbf{x}}^{(k)} + \mathbf{K}_k(\mathbf{z}^{(k)} - h(\tilde{\mathbf{x}}^{(k)}, 0)). \tag{4.13}$$

In Abbildung 4.2 ist eine Übersicht über die Funktionsweise des erweiterten Kalman-Filters zu sehen. Die Schritte sind dabei genau die selben wie beim diskreten Kalman-Filter aus Abschnitt 4.2.1. Der wesentliche Unterschied besteht darin, dass beim Berechnen der Zustände nun statt linearer Matrizen, nichtlineare Funktionen verwendet werden. Dadurch ändert sich jedoch auch die Berechnung der Fehlerkovarianzen wie in der Arbeit von Welch und Bishop [77] beschrieben.

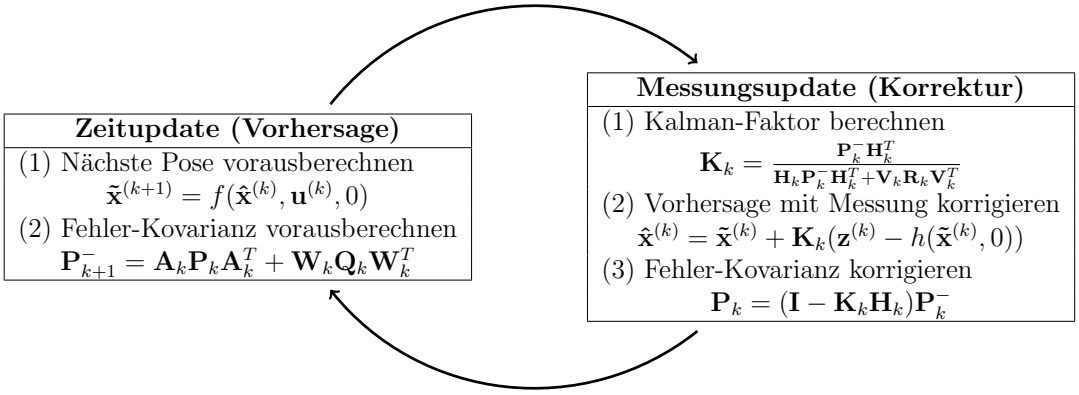


Abbildung 4.2.: Übersicht über die Funktionsweise des erweiterten Kalman-Filters (vgl. [77]). Im Vorhersageschritt wird die nächste Pose $\tilde{\mathbf{x}}^{(k+1)}$ mit Hilfe der vorherigen Pose $\hat{\mathbf{x}}^{(k)}$ und des Steuerimpulses $\mathbf{u}^{(k)}$ vorausberechnet und anschließend im Korrekturschritt zu $\hat{\mathbf{x}}^{(k)}$ korrigiert, sobald Messdaten $\mathbf{z}^{(k)}$ vorliegen.

Implementierung

Im Rahmen der Odometrie werden ausschließlich eindimensionale Kalmanfilter verwendet. Diese erhalten einen Wert als Eingabe und geben den gefilterten Wert als Ausgabe aus. In diesem Fall vereinfacht sich die Berechnung des Kalman-Filters auf Algorithmus 4.1.

1: $P \leftarrow P + Q$
2: $K \leftarrow \frac{P}{P+R}$
3: $x \leftarrow x + K \cdot (z - x)$
4: $P \leftarrow (1 - K) \cdot P$
5: **return** x

Algorithmus 4.1: Eindimensionaler Kalman-Filter

Um die Fehlerkovarianzen Q und R nicht im Vorhinein fest definieren zu müssen, schlagen Galanis und Anadranistakis vor, diese Parameter während der Ausführung aus den letzten Ein- und Ausgaben zu bestimmen [31]. Sei dafür z_t die aktuelle Eingabe und x_{t-1} der letzte gefilterte Wert, dann ergeben sich die Fehlerkovarianzen mit

$$Q = \frac{1}{N-1} \cdot \sum_{i=0}^{N-1} \left((x_{t-i} - x_{t-i-1}) - \frac{1}{N} \sum_{j=0}^{N-1} x_{t-j} - x_{t-j-1} \right)^2, \tag{4.14}$$

$$R = \frac{1}{N-1} \cdot \sum_{i=0}^{N-1} \left((z_{t-i} - x_{t-i-1}) - \frac{1}{N} \sum_{j=0}^{N-1} z_{t-j} - x_{t-j-1} \right)^2. \tag{4.15}$$

Dabei gibt N die Anzahl an Werten an, die zur Berechnung berücksichtigt werden sollen. Da der Rechenaufwand für große Werte von N höher ist, und die Qualität der Schätzung nicht signifikant mit N steigt, wurden in den Experimenten Werte von $N \leq 20$ verwendet.

4.2.2. Fourier-Filter

Der Fourier-Filter dient zur effizienten Tiefpass-Filterung der Sensordaten mittels einer aus der Fourier-Transformation abgeleiteten Berechnung, wobei die Laufzeit nicht von der Größe des untersuchten Bereichs, sondern lediglich von der Anzahl verwendeter Frequenzen im Tiefpass abhängig ist. Dadurch ist das Verfahren deutlich schneller als eine vollständige Fourier-Transformation.

Fourier-Transformation

Die Fourier-Transformation entspricht der Zerlegung eines Signals in die darin enthaltenen Frequenzen, indem das Signal in mehrere Kosinus- und Sinuswellen zerlegt wird. Dabei werden üblicherweise die Kosinus- und Sinusterme mittels der Gleichung $\cos(x) - i \cdot \sin(x) = e^{ix}$ zu einem Exponentialfunktionsterm zusammengefasst, wobei hier und im Folgenden e die eulersche Zahl und i die imaginäre Zahl ist. Es ergibt sich im eindimensionalen Fall die Amplitude einer Frequenz u in einem unendlichen, kontinuierlichen Signal $f(t) : \mathbb{R} \to \mathbb{R}$ über die Funktion

$$F(u) = \int f(t) \cdot e^{-2\pi i u t} dt, \tag{4.16}$$

bzw. bei einem diskreten, endlichen Signal $f(t) : \{0, \ldots, N-1\} \to \mathbb{R}$ über die Funktion

$$F(u) = \sum_{t=0}^{N-1} f(t) \cdot e^{-2\pi i t \frac{u}{N}}. \tag{4.17}$$

Dabei ist der Definitionsbereich von $F(u)$ ebenfalls $\{0, \ldots, N-1\}$. Die Zerlegung des Signals in Frequenzen ist ersichtlich mit $e^{-xi} = \cos(x) + i \sin(x)$. Da in diesem Anwendungsfall nur die diskrete Version der Fouriertransformation relevant ist, wird die kontinuierliche nicht weiter berücksichtigt. Die inverse diskrete Fourier-Transformation lautet

$$f(t) = \frac{1}{N} \sum_{u=0}^{N-1} F(u) \cdot e^{2\pi i t \frac{u}{N}}. \tag{4.18}$$

Die Darstellung eines Signals, durch die darin enthaltenen Frequenzen, ist besonders für eine Implementierung von Tief- und Hochpassfiltern geeignet, da eine unmittelbare Reduzierung oder Verstärkung der Frequenzen im Frequenzbereich erfolgen kann.

Herleitung des Filters

Der Filter selbst basiert auf der Idee alle unerwünschten Frequenzen aus dem Signal zu eliminieren. Dies erfolgt indem ausschließlich die Werte zurück transformiert werden, die zu den ausgewählten Frequenzen gehören. Das ist äquivalent dazu in der Fourier-Transformierten alle anderen Frequenzen auf 0 zu setzen. Ist $F(u)$ die Fourier-Transformierte eines endlichen, diskreten Signals $f(t) : \{0, \ldots, N-1\} \to \mathbb{R}$. Dann gilt: $F(u) = \overline{F(N-u)}$, welches einer Frequenz mit Periodenlänge $L = N - 2u$ entspricht. Folglich befinden sich die langen Frequenzen in den äußeren Werten des Definitionsbereichs und die hohen Frequenzen im Inneren. Die exakten Frequenzen

für eine bestimmte Periodenlänge lassen sich aus dieser Gleichung herleiten. Werden ausschließlich die Frequenzen $U \subseteq \{0, \ldots, \lfloor \frac{N-1}{2} \rfloor\}$ zurück transformiert, so ergibt sich die inverse Fourier-Transformation als

$$f'(t) = \frac{1}{N} \sum_{u \in U} \left(F(u) \cdot e^{2\pi i t \frac{u}{N}} + F(N-u) \cdot e^{2\pi i t \frac{N-u}{N}} \right) \tag{4.19}$$

$$= \frac{1}{N} \sum_{u \in U} \left(F(u) \cdot e^{2\pi i t \frac{u}{N}} + \overline{F(u)} \cdot e^{-2\pi i t \frac{u}{N}} \right) \tag{4.20}$$

Durch Einsetzen der Gleichung 4.20 in die Definition der Fourier-Transformation, ergibt sich

$$\begin{aligned}
f'(t) = & \frac{1}{N} \sum_{u \in U} \left(\left(\sum_{k=0}^{N-1} f(k) \cdot e^{-2\pi i k \frac{u}{N}} \right) \cdot e^{2\pi i t \frac{u}{N}} \right. \\
& \left. + \overline{\left(\sum_{k=0}^{N-1} f(k) \cdot e^{-2\pi i k \frac{u}{N}} \right)} \cdot e^{-2\pi i t \frac{u}{N}} \right) \\
= & \frac{1}{N} \sum_{u \in U} \left(\left(\sum_{k=0}^{N-1} f(k) \cdot e^{-2\pi i k \frac{u}{N}} \right) \cdot e^{2\pi i t \frac{u}{N}} + \left(\sum_{k=0}^{N-1} f(k) \cdot e^{2\pi i k \frac{u}{N}} \right) \cdot e^{-2\pi i t \frac{u}{N}} \right) \\
= & \frac{1}{N} \sum_{u \in U} \left(\sum_{k=0}^{N-1} f(k) \cdot e^{2\pi i (t-k) \frac{u}{N}} + \sum_{k=0}^{N-1} f(k) \cdot e^{-2\pi i (t-k) \frac{u}{N}} \right) \\
= & \frac{1}{N} \sum_{u \in U} \left(\sum_{k=0}^{N-1} f(k) \cdot 2 \cos(2\pi (t-k) \frac{u}{N}) \right) \\
= & \frac{1}{N} \sum_{u \in U} \left(\sum_{k=0}^{N-1} f(k) \cdot \left(\cos(2\pi k \frac{u}{N}) \cos(2\pi t \frac{u}{N}) + \sin(2\pi k \frac{u}{N}) \sin(2\pi t \frac{u}{N}) \right) \right) \\
= & \frac{1}{N} \sum_{u \in U} \cos(2\pi t \frac{u}{N}) \left(\sum_{k=0}^{N-1} f(k) \cdot \cos(2\pi k \frac{u}{N}) \right) \\
& + \sin(2\pi t \frac{u}{N}) \left(\sum_{k=0}^{N-1} f(k) \cdot \sin(2\pi k \frac{u}{N}) \right).
\end{aligned} \tag{4.21}$$

Die Formel 4.21 lässt sich mit

$$c_u := \sum_{k=0}^{N-1} f(k) \cdot \cos(2\pi k \frac{u}{N}) \tag{4.22}$$

und

$$s_u := \sum_{k=0}^{N-1} f(k) \cdot \sin(2\pi k \frac{u}{N}) \tag{4.23}$$

vereinfachen zu

$$f'(t) = \frac{1}{N} \sum_{u \in U} \cos(2\pi t \frac{u}{N}) c_u + \sin(2\pi t \frac{u}{N}) s_u. \tag{4.24}$$

Die Idee des Filters ist, die Koeffizienten c_u und s_u für jeden neuen Wert effizient zu aktualisieren. Dies erfolgt durch eine systematische Annahme von Periodizität im Signal, indem für ein Array der Größe N für jeden neuen Wert der älteste Wert überschrieben wird, d. h. der i-te Wert wird an die ($i \mod N$)-te Stelle geschrieben. Dadurch muss in jedem Schritt nur ein Term der Koeffizienten s_u und c_u geändert werden. Sei $\tilde{x}$ der nächste zu filternde Wert, x der von diesem Wert überschriebene Wert im Array, t der Index von x im Array und s_u und c_u die zuletzt berechneten Koeffizienten des Filters, dann ergeben sich für die neue Eingabe x die neuen Koeffizienten $\tilde{s_u}$ und $\tilde{c_u}$ als

$$\tilde{s_u} = s_u + (\tilde{x} - x) \sin(2\pi t \frac{u}{N}), \quad \tilde{c_u} = c_u + (\tilde{x} - x) \cos(2\pi t \frac{u}{N}) \tag{4.25}$$

Um also einen Wert zu filtern, müssen nur die s_u und c_u aktualisiert und die Formel 4.24 berechnet werden. $\sin(2\pi t \frac{u}{N})$ und $\cos(2\pi t \frac{u}{N})$ sind darin die einzigen, zu berechnenden Sinus- und Kosinusterme. Da t und u ganzzahlig sind, lassen sich diese Terme in einem Array der Größe N vorberechnen und speichern. Das Ergebnis ist dann der gefilterte Wert.

Initialisierung

Als initiale Wertebelegung wird ein Startwert für alle zu filternden Werte in das Array geschrieben und die Sinus- und Kosinusterme, ebenso wie die s_u und c_u, werden vorberechnet. Durch iteratives Ersetzen von Werten im Array konvergiert das Verhalten des Filters nach N Schritten gegen die erwarteten Werte, weshalb eine Initialisierungsphase empfehlenswert, wenn auch nicht zwingend notwendig, ist, sofern das Verhalten des Signals vor Initialisierung als konstant angenommen werden kann.

4.2.3. Optischer Fluss

Neben den Filtern, die auf den direkten Sensordaten operieren, wird aus dem Videostream der Bodenkamera ein optischer Fluss berechnet. Dieser entspricht der Verschiebung von Pixeln in zwei aufeinanderfolgenden Bildern. Mittels dieser Verschiebung kann darüber hinaus die Bewegung der Kamera ermittelt und damit die Berechnung aus Sensordaten stabilisiert werden. Um den optischen Fluss und die Egomotion, d. h. die Bewegung der Kamera, aus den Bildern der Bodenkamera zu gewinnen, werden drei Algorithmen mit einander kombiniert: Die Eckenerkennung nach Rosten und Drummond [68], der Lucas-Kanade-Algorithmus zur Berechnung eines optischen Flusses und einem Algorithmus aus dem Quellcode von Paparazzi[1], einem verbreiteten Open Source Projekt zur Steuerung von Drohnen, welcher den berechneten optischen Fluss in eine parallel zum Boden gerichtete Geschwindigkeit umwandelt. All diese Algorithmen wurden aus dem Quellcode von Paparazzi übernommen, syntaktisch für die Ausführung im Projekt angepasst und an ein paar Stellen für eine effizientere Berechnung umgeschrieben, ohne jedoch das Resultat der Berechnung zu verändern. Das Vorgehen der einzelnen Algorithmen wird hier kurz erläutert.

Lucas-Kanade-Algorithmus

Das Prinzip des Lucas-Kanade-Algorithmus ist es, für gegebene Punkte p_i in einem Bild, eine Verschiebung t_i in Pixeln zu berechnen, die der Verschiebung der p_i und deren Umgebung möglichst gut entspricht. Die Intuition der zu Grunde liegenden Berechnung ist, dass sich starke Veränderungen eines Pixelwertes in aufeinander folgenden Bildern nur dann erzielen lassen, wenn sich der Bildinhalt in Richtung des lokalen Gradienten bewegt hat. Dies basiert auf der Annahme, dass gleiche Bildinhalte in auf einander folgenden Bildern sehr ähnliche Farbwerte aufweisen. Diese Intuition lässt sich dann in der Gleichung

$$\nabla I(n_{i,k})^\top t_i = I(n_{i,k}) - I'(n_{i,k}) \tag{4.26}$$

umsetzen. Dabei sind $I(p)$ bzw. $I'(p)$ die Farbwerte des Pixels p im ersten bzw. zweiten Bild und $n_{i,k}$ Pixel aus der Umgebung von p_i. Für gewöhnlich wird an statt der Differenz der Farbwerte auf der rechten Seite der Gleichung 4.26 die negative Ableitung des Farbwerts des Pixels nach der Zeit verwendet. Bei nur zwei betrachteten Bildern wird diese Ableitung jedoch mittels der Differenz approximiert. Wäre

[1] `https://wiki.paparazziuav.org/wiki/Main_Page` Abruf: 10.09.16 13:42

Gleichung 4.26 für alle Punkte der Nachbarschaft von p_i erfüllt, so wäre der berechnete optische Fluss t_i optimal. Dies ist praktisch, jedoch selten möglich, wodurch der Fehler, folglich die Differenz der linken und rechten Seite der Gleichung, über das Prinzip der kleinsten Quadrate minimiert und der bestmögliche optische Fluss ausgegeben wird.

Eckenerkennung

Die Motivation der Eckenerkennung rührt daher, dass es sich für die Berechnung eines optischen Flusses als sinnvoll erwiesen hat, Ecken als designierte Punkte zu verwenden, da die Umgebung von Ecken verschieden gerichtete Gradienten aufweist. Der Algorithmus von Rosten und Drummond erzeugt dafür um alle Pixel einen „Kreis“ aus 16 Pixeln und prüft, ob die Farbwerte einer beliebigen Kombination von neun Pixeln um mehr als einen vordefinierten Schwellwert vom Farbwert des zentralen Pixels abweichen. Darüber hinaus wird über Parameter vermieden, dass ausgewählte Ecken zu nahe bei einander liegen, indem die Umgebung gefundener Ecken nicht weiter betrachtet wird.

Egomotion

Der Egomotion Algorithmus aus Paparazzi erhält, zusätzlich zu den optischen Flüssen an den auserwählten Ecken, die Inklination, den Roll- und Neigewinkel, der Drohne zum Zeitpunkt der Bildaufnahme. Andere Parameter, wie die Brennweite, sind im Programmcode fest eingebaut. Aus diesen Parametern eliminiert der Algorithmus zunächst die Anteile des optischen Flusses, die durch die Inklination erzeugt werden, da zum Beispiel ein Neigen der Kamera nach vorne eine Verschiebung der Bildinhalte zur Folge hat. Die optischen Flüsse werden dann entsprechend ihrer euklidischen Norm sortiert und anschließend der Mittelwert des Median und seiner Nachbarn von Pixeln in Meter umgerechnet. Diese Distanz ergibt dann dividiert durch die Zeit zwischen den Bildern die Geschwindigkeit der Kamera.

4.2.4. Sensorfusion

Um die Daten aus den verschiedenen Sensoren zu vereinen wird ein mehrstufiges System angewandt. Zunächst werden die Akzelerometer- und Gyrometerdaten miteinander korrigiert. Dies ist notwendig, da die Inertial Measurement Unit (IMU), i.e. die physischen Sensoren, im Rumpf der Drohne verbaut ist, welcher federgelagert ist, was zur Folge hat, dass die Bewegung der Sensoren von der Bewegung des Rumpfes

relativ zum Schwerpunkt der Drohne überlagert wird. Das Resultat dieser Korrekturen wird mittels eines Kalman-Filters geglättet, um eine stabilere Berechnung zu gewährleisten. Da die verbauten Akzelerometer einen schwankenden Bias, d. h. einen variierenden Nullpunkt besitzen, wird ein Fourier-Filter als Tiefpassfilter eingesetzt, um die tiefsten Frequenzen im Signal heraus zu filtern und vom gemessenen Wert abzuziehen. Diese Korrektur des Bias ist zwar lediglich eine Approximation, jedoch ist eine exakte Berechnung eines variierenden Bias zur Laufzeit nicht möglich. Die resultierenden Werte werden anschließend verwendet, um den Zustand der Drohne zu aktualisieren. Dieser Zustand ermittelt sich durch Integration und beinhaltet die Geschwindigkeit der Drohne in Weltkoordinaten in $\frac{m}{s}$, die Position der Drohne in Weltkoordinaten in m und die Rotation der Drohne relativ zum Weltkoordinatensystem als Quaternion

Die Daten aus dem optischen Fluss werden über den Algorithmus zur Egomotion in eine Geschwindigkeit umgesetzt. Diese befindet sich zunächst im Koordinatensystem der Drohne und wird anschließend mit der letzten Posenabschätzung in das globale Koordinatensystem transformiert. Da der Fehler der Geschwindigkeit durch variierenden Bias und Integration allein aus Akzelerometerdaten linear und der Fehler in der Position der Drohne folglich quadratisch ist, wird die aus der Egomotion ermittelte Geschwindigkeit bei jeder Berechnung in den Zustand der Odometrie geschrieben, so dass der Fehler in der Position nur noch linear ist. Auf die Daten der Akzelerometer, zu den Zeitpunkten zwischen den Bildern der Bodenkamera, wird jedoch zu Gunsten von Anfragen nicht verzichtet.

Kreuzkombination von Akzelerometer- und Gyrometerdaten

Um die Bewegung der IMU relativ zum Drohnenschwerpunkt zu berechnen, wird eine Neigungsabschätzung der Drohne mittels *atan*2-Funktion, entsprechend den Herleitungen von Pedley, vorgenommen. Die *atan*2-Funktion ist definiert als

$$atan2(y,x) = \begin{cases} \arctan(\frac{y}{x}) & x > 0 \\ \arctan(\frac{y}{x}) + \pi & x < 0, y \geq 0 \\ \arctan(\frac{y}{x}) - \pi & x < 0, y < 0 \\ \frac{\pi}{2} & x = 0, y > 0 \\ -\frac{\pi}{2} & x = 0, y < 0 \\ \text{undefined} & x = y = 0 \end{cases} . \tag{4.27}$$

Dabei wird die Neigung als Winkel für Rollen (*roll*) ϕ_r und Neigen (*pitch*) ϕ_p aus den Akzelerometerdaten für Beschleunigung in Richtung Norden[2] a_n, Osten a_e und unten a_d berechnet mittels

$$\phi_r = atan2(a_e, a_d) \cdot \frac{180}{\pi}, \tag{4.28}$$

$$\phi_p = atan2(a_n, \sqrt{a_e^2 + a_d^2}) \cdot \frac{180}{\pi}. \tag{4.29}$$

Die Differenz der auf diese Art ermittelten Winkel, zu den von den Gyrometerwerten errechneten, ergibt dann die tatsächlichen Gyrometerwerte, da der Fehler, der durch die Bewegung des Drohnenrumpfes relativ zum Schwerpunkt entsteht, eliminiert wird. Umgekehrt können auf diese Art auch die Akzelerometerwerte von dieser Bewegung bereinigt werden. Dies geschieht indem die Gleichungen umgestellt werden zu

$$atan2(a_e, a_d) = \phi_r \cdot \frac{\pi}{180}, \tag{4.30}$$

$$atan2(a_n, \sqrt{a_e^2 + a_d^2}) = \phi_p \cdot \frac{\pi}{180}. \tag{4.31}$$

Die Inverse der $atan2$-Funktion ist nur für ein festes $\frac{y}{x}$ eindeutig definiert. Auf Grund des Rauschens auf den Akzelerometerdaten, würde die Verwendung des Quotienten aus den gemessenen Beschleunigungen zu einer numerisch instabilen Berechnung führen, wodurch das Ergebnis dieser Berechnung so skaliert wird, dass die Summe der Ergebnisse mit der Summe der Messwerte übereinstimmt. Auf diese Art wird der Fehler, der durch Rauschen entsteht, reduziert. Mit diesem Ansatz ergeben sich dann die neuen Beschleunigungen mit

$$\tilde{a_n} = \frac{\sqrt{a_e^2 + a_d^2} \cdot atan2^{-1}(\phi_p) \cdot (a_n + \sqrt{a_e^2 + a_d^2})}{\sqrt{a_e^2 + a_d^2} \cdot atan2^{-1}(\phi_p) + \frac{a_n}{atan2^{-1}(\phi_p)}}, \tag{4.32}$$

$$\tilde{a_e} = \frac{a_d \cdot atan2^{-1}(\phi_r) \cdot (a_e + a_d)}{a_d \cdot atan2^{-1}(\phi_r) + \frac{a_e}{atan2^{-1}(\phi_r)}}, \tag{4.33}$$

$$\tilde{a_d} = \frac{\frac{a_e}{atan2^{-1}(\phi_r)} \cdot (a_e + a_d)}{a_d \cdot atan2^{-1}(\phi_r) + \frac{a_e}{atan2^{-1}(\phi_r)}}. \tag{4.34}$$

[2]Die Beschreibung der Achsen als Norden und Osten entstammt einer Konvention aus der Luftfahrt. Die Nordachse zeigt aus Perspektive des Flugobjektes nach vorne und die Ostachse aus Perspektive des Flugobjektes nach rechts.

4.2.5. Datenspeicherung

Nachdem die Daten von den Sensoren verarbeitet sind, müssen diese gespeichert werden. Zu diesem Zweck wird eine einfach verkettete Liste von Posen, d. h. Drehung und Translation, der Drohne relativ zum Weltkoordinatensystem angelegt. Anstatt allerdings für jeden Datenpunkt einen Eintrag in dieser Liste anzulegen, was bei einer Datenrate von 400 Hz bereits nach kurzer Zeit viel Speicher belegt und auf Grund der Adressierung einzelner Einträge mit zunehmender Länge der Liste mehr Zeit beansprucht, werden stattdessen nur Einträge an Schlüsselmomenten, den hier so genannten *Keypoints* angelegt. Im Rahmen des Projektes entsprechen diese Keypoints den Momenten, in denen ein Bild an das LSD-SLAM gesandt wird. Dies ist notwendig, da die korrigierten Posen der Drohne, die vom LSD-SLAM zurückgesandt werden, an diese Zeitpunkte gekoppelt sind und somit eine Korrektur des in der Odometrie entstandenen Fehlers nur an diesen Stellen sinnvoll ist. Damit dies möglich ist, werden die Keypoints mit Zeitstempeln versehen, sodass bei einem Aufruf einer Update-Funktion der zugehörige Keypoint korrigiert werden kann. Dieser Unterabschnitt widmet sich daher den Keypoints, wie diese entstehen und wie Updates auf der Liste von Keypoints durchgeführt werden können.

Erstellen von Keypoints

Zu Beginn der Ausführung und bei jeder Anforderung eines Keypoints mit übergebenem Zeitstempel, wird der Zustand der Odometrie als Keypoint zur Liste hinzugefügt. Die Werte der Translation und Rotation der Odometrie werden auf Null zurück gesetzt. Der gespeicherte Geschwindigkeitsvektor bleibt jedoch erhalten, da dieser nicht vom LSD-SLAM korrigiert werden kann. Auf diesem neuen Zustand werden bis zum nächsten Keypoint die Bewegungen aufintegriert.

Propagation von Odometriedaten

Sobald die Odometrie ein Update, bestehend aus einem Zeitstempel und einer Pose, erhält, wird der Keypoint gesucht, in dem der gleiche oder der nächst kleinere Zeitstempel steht. In diesem wird die alte durch die neue Pose ersetzt und alle älteren Keypoints werden gelöscht. Wird nun die aktuelle Position der Drohne angefragt, so ergibt sich die Position durch rekursives Zusammenfügen der einzelnen Posen, indem jeweils die Translation des aktuellen Keypoints mit der Rotation des vorigen gedreht und anschließend die aktuelle Rotation mit der vorigen kombiniert wird. Auf diese Weise lassen sich, durch Updates an einzelnen Keypoints, alle nach diesem erstellten

Keypoints korrigieren. Damit jedoch nicht bei jeder Anfrage der Pose alle gespeicherten Keypoints aufsummiert werden müssen, wird in den einzelnen Keypoints die Gesamttranslation und -rotation seit dem letzten Update gespeichert. So fällt nur bei jedem Update einmalig die Rechenlast an. In der Berechnung der aktuellen Pose wird der Zustand der Odometrie als virtueller Keypoint verrechnet.

4.3. Detektion der Umgebung

In diesem Abschnitt wird darauf eingegangen, wie die Umgebung mit Hilfe der Kamera detektiert werden kann. Dazu wird zunächst als allgemeiner Lösungsansatz das SLAM-Verfahren eingeführt. Anschließend wird das in diesem Projekt verwendete LSD-SLAM vorgestellt und ausgeführt wie das Verfahren in das Projekt eingegliedert wurde.

4.3.1. SLAM-Verfahren

Das beim SLAM-Problem betrachtete Szenario umfasst ein unbemanntes Vehikel, welches an einer unbekannten Position in einer unbekannten Umgebung ausgesetzt wird. Dieses Vehikel soll autonom mit Hilfe seiner Sensoren eine Karte der Umgebung erstellen und sich selbst auf dieser Karte lokalisieren können [19].
Bei einem allgemeinen Ansatz zur Lösung des Problems werden zum Erstellen der Umgebungskarte vom Vehikel aus sog. Landmarks in der Umgebung detektiert. Dies sind markante Punkte, die auf den Kamerabildern leicht zu erkennen sind und dadurch genutzt werden können, um die Position zu bestimmen, von der aus diese Bilder gemacht wurden. Zum Zeitpunkt t befindet sich das Vehikel in der Pose $\mathbf{x}^{(t)}$, die sowohl die räumliche Position als auch die Ausrichtung des Vehikels beschreibt. Des Weiteren werden die Bewegungsbefehle des Vehikels als Vektor $\mathbf{u}^{(t)}$ angenom-

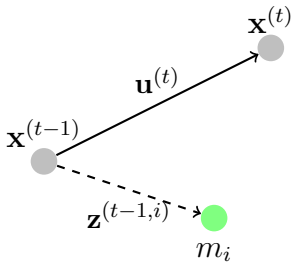


Abbildung 4.3.: Das Vehikel wird im Zeitschritt t durch den Bewegungsbefehl $\mathbf{u}^{(t)}$ von Position $\mathbf{x}^{(t-1)}$ zu Position $\mathbf{x}^{(t)}$ bewegt. Zum Zeitpunkt $t-1$ wurde Landmark m_i durch Beobachtung $\mathbf{z}^{(t-1,i)}$ gesehen (vergleiche [19]).

men. In Abbildung 4.3 ist die Bewegung des Vehikels von Pose $\mathbf{x}^{(t-1)}$ zur Pose $\mathbf{x}^{(t)}$ dargestellt. Die Beobachtung eines Landmarks m_i zum Zeitpunkt t wird als $\mathbf{z}^{(t,i)}$ beschrieben. Alle Beobachtungen zum Zeitpunkt t werden als $\mathbf{z}^{(t)}$ dargestellt. Im Weiteren werden die Mengen aller zum Zeitpunkt t bekannten Posen, Bewegungsanweisungen und Beobachtungen als $\mathcal{X}_t, \mathcal{U}_t$ und $\mathcal{Z}_t$ und die Menge aller Landmarks als $\mathcal{M}$ bezeichnet [19].
Um das SLAM-Problem zu lösen, soll zu jedem Zeitpunkt t eine Annahme über die Posen des Vehikels und der Landmarks existieren. Diese ist als Wahrscheinlichkeitsverteilung

$$P(\mathbf{x}^{(t)}, \mathcal{M} | \mathcal{Z}_t, \mathcal{U}_t, \mathbf{x}^{(0)}) \tag{4.35}$$

gegeben. Der Vektor $\mathbf{x}^{(0)}$ ist dabei die Startpose. Aus dieser Wahrscheinlichkeitsverteilung geht hervor wie die Wahrscheinlichkeit für die Posen des Vehikels und der Landmarks in Abhängigkeit von den bisherigen Beobachtungen, Bewegungen und der Startpose sind. Wie bereits aus Abschnitt 4.2.1 bekannt, wird diese Wahrscheinlichkeitsverteilung in einem rekursiven Ablauf zuerst für den nächsten Zeitpunkt geschätzt und anschließend, durch zu diesem Zeitpunkt aufgenommene Messdaten, korrigiert.
Um eine Vorhersage über die zukünftige Pose $\mathbf{x}^{(t)}$ des Vehikels zu machen, reicht das Wissen über die vorherige Pose $\mathbf{x}^{(t-1)}$ und den Bewegungsbefehl $\mathbf{u}^{(t)}$ aus. Die Wahrscheinlichkeit für eine Pose $\mathbf{x}^{(t)}$ ist daher gegeben als

$$P(\mathbf{x}^{(t)} | \mathbf{x}^{(t-1)}, \mathbf{u}^{(t)}) \ . \tag{4.36}$$

Des Weiteren kann eine Aussage über die vom Vehikel beobachteten Landmarks getroffen werden, wenn die Posen des Vehikels und der Landmarks bekannt sind. Die Wahrscheinlichkeit für eine Beobachtung $\mathbf{z}^{(t)}$ ist daher gegeben als

$$P(\mathbf{z}^{(t)} | \mathbf{x}^{(t)}, \mathcal{M}) \ . \tag{4.37}$$

Mit Hilfe der Wahrscheinlichkeitsverteilungen 4.36 und 4.37 ist jetzt die Definition eines Vorhersage- und eines Korrekturschrittes für das rekursive Verfahren möglich. Bei der Vorhersage wird zunächst

$$\begin{aligned} &P(\mathbf{x}^{(t)}, \mathcal{M} | \mathcal{Z}_{t-1}, \mathcal{U}_k, \mathbf{x}^{(0)}) = \\ &\int P(\mathbf{x}^{(t)} | \mathbf{x}^{(t-1)}, \mathbf{u}^{(t)}) \times P(\mathbf{x}^{(t-1)}, \mathcal{M} | \mathcal{Z}_{t-1}, \mathcal{U}_{t-1}, \mathbf{x}^{(0)}) \mathrm{d}\mathbf{x}^{(t-1)} \end{aligned} \tag{4.38}$$

berechnet und anschließend im Korrekturschritt benutzt, um

$$P(\mathbf{x}^{(t)}, \mathcal{M}|\mathcal{Z}_t, \mathcal{U}_t, \mathbf{x}^{(0)}) = \frac{P(\mathbf{z}^{(t)}|\mathbf{x}^{(t)}, \mathcal{M})P(\mathbf{x}^{(t)}, \mathcal{M}|\mathcal{Z}_{t-1}, \mathcal{U}_t, \mathbf{x}^{(0)})}{P(\mathbf{z}^{(t)}|\mathcal{Z}_{t-1}, \mathcal{U}_t)} \tag{4.39}$$

zu berechnen [19].
Um diesen allgemeinen Ansatz zur Lösung des SLAM-Problems zu einem funktionierenden SLAM-Verfahren auszubauen, wird vor allem eine gute Umsetzung der rekursiven Berechnung benötigt. Zwei bekannte Verfahren in diesem Bereich sind Extended Kalman Filter (EKF)-SLAM [18] und FastSLAM [54]. Bei EKF-SLAM wird zur Berechnung ein erweiterter Kalman-Filter wie der aus Abschnitt 4.2.1 benutzt. Bei FastSLAM wird ein Rao-Blackwellized Partikel-Filter eingesetzt.
Gerade in den letzten Jahren sind allerdings auch viele neue monokulare SLAM-Verfahren wie Oriented Fast and Rotated Brief (ORB)-SLAM [56] und das im nächsten Abschnitt beschriebene LSD-SLAM [21] veröffentlicht worden, die von den bisherigen Landmark-basierten Verfahren abweichen und auf Keyframes arbeiten.

4.3.2. LSD-SLAM

LSD-SLAM ist ein Verfahren mit dem dreidimensionale Karten von größeren Umgebungen zur Laufzeit der Anwendung konstruiert werden können. Das Verfahren kommt ohne der Erkennung von Landmarks in Bildern aus und benötigt lediglich eine monokulare Kamera, die einen Bildstrom liefert. Dies ermöglicht den Einsatz auf kleineren Geräten, wie etwa auf einem Mobiltelefon oder auf Drohnen, sodass die Verwendung von Tiefensensoren, welche häufig eine geringe Reichweite haben, zu diesem Zweck nicht nötig ist.
Wie im Artikel [21] des Verfahrens, wird in den folgenden Kapiteln zunächst das allgemeine Verfahren mit den drei Hauptkomponenten, der Selbstlokalisierung, Tiefenwert-Schätzung, sowie der Optimierung im Posen-Graph beschrieben. Anschließend wird auf diese Komponenten näher eingegangen.

Allgemeines Verfahren

Verarbeitet werden kalibrierte schwarz-weiß Bilder, die zur Einfachheit an dieser Stelle Frames genannt werden. Einen wesentlichen Bestandteil der konstruierten Karte stellen dabei die Keyframes $\mathcal{K}^{(i)} = (\mathcal{I}^{(i)}, \mathcal{D}^{(i)}, \mathcal{V}^{(i)})$ dar, welche aus einem Bild $\mathcal{I}^{(i)} : \Omega^{(i)} \to \mathbb{R}$, den zu den Pixeln des Bildes zugehörigen inversen Tiefenwerten $\mathcal{D}^{(i)} : \Omega^{(D^{(i)})} \to \mathbb{R}$ und dessen Varianzen $\mathcal{V}^{(i)} : \Omega^{D^{(i)}} \to \mathbb{R}$ bestehen. Sie sind die

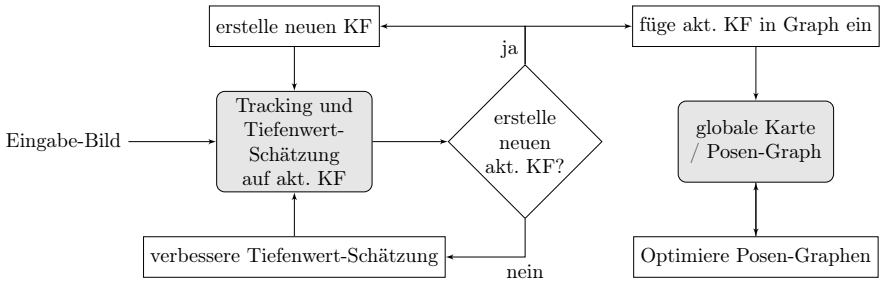


Abbildung 4.4.: Überblick des Verfahrens von LSD-SLAM, welches sich in drei Gebiete unterteilen lässt: Tracking, Tiefenwert-Schätzung und in die Optimierung des Posen-Graphen.

Knoten eines Graphen (Kapitel 4.3.2), welcher die vollständige Karte repräsentiert. Hierzu wird zu jeder Zeit genau ein Keyframe als Referenz zu den aktuell zu verarbeitenden Frames für die Selbstlokalisation verwendet. Das Verfahren kann in drei Hauptpunkte unterteilt werden: der Selbstlokalisation (bzw. Schätzung der Pose für das aktuelle Bild), die Tiefenwertschätzung und die Optimierung der globalen Karte. Der Definitionsbereich $\mathcal{D}^{(i)}$ deutet darauf hin, dass nicht jedem Pixel ein Tiefenwert zugeordnet wird.

Zu jeder Zeit dient genau ein Keyframe als Referenz in der Selbstlokalisation. Zur Bestimmung der Pose des aktuellen Frames wird ein photometrischer Fehler (Gl. 4.44) bezüglich dieses Keyframes und des zu lokalisierenden Frames minimiert. Unter Umständen, bspw. wenn der Abstand zum aktiven Keyframe groß ist, wird der zu bearbeitende Frame selbst zum neuen aktiven Keyframe.

Wenn ein neuer Keyframe gewählt wird, werden dessen Tiefenwerte mit Hilfe der Werte seines Vorgängers initialisiert. Anschließend werden alle nicht zu Keyframes gewählten Frames zur Verbesserung dieser Tiefenwerte genutzt. Dabei wird zunächst eine Menge an Pixeln aus dem Frame ausgewählt und anschließend für jeden Pixel der Menge eine Tiefenwertschätzung berechnet. Diese Schätzungen werden danach in die Tiefenkarte des Keyframes eingefügt. Wird ein neuer Keyframe erstellt, endet die Verbesserung der Daten für seinen Vorgänger.

Nachdem ein neuer Keyframe gesetzt wurde, wird sein Vorgänger in einen globalen Graphen eingebunden. Dieser Graph speichert alle bisher gefundenen Keyframes als Knoten und deren relative Position zueinander als Kanten. Insbesondere wird, sobald ein neuer Keyframe eingefügt wird, überprüft, ob dieser eine Schleife in der Kamerabewegung schließt. Ist dies der Fall, kann dadurch eine

Abweichung in der Genauigkeit der Posenschätzungen festgestellt und korrigiert werden. Somit dient diese parallel verlaufende Optimierung vor allem der Vorbeugung von sich aufsummierenden Fehlern der Posenschätzungen in längeren Durchläufen.

Selbst-Lokalisierung

Sei $\mathcal{K}^{(i)}$ der aktuelle Keyframe und sei $\mathcal{I}^{(j)}$ das Bild, dessen Pose geschätzt werden soll. Gesucht wird die Kamera-Bewegung $\xi^{(\mathbf{ji})} \in \mathfrak{se}(3)$ vom Bild $\mathcal{I}^{(i)}$ nach $\mathcal{I}^{(j)}$, wobei $\mathfrak{se}(3) \subseteq \mathbb{R}^7$ eine Lie-Algebra (vergleiche [7]) und ihre Elemente sind Vektoren sind. Die Pose, bestehend aus der Kamera-Position und Ausrichtung des Keyframes i im Weltkoordinatensystem, sei im folgenden $\xi^{(i)}$. Jeder Keyframe erhält hierfür einen Verweis auf den vorherigen aktiven Keyframe, sodass die Pose bestimmt werden kann, indem der Graph durchlaufen wird und die Kamera-Bewegungen konkateniert werden.

Alternativ kann eine Pose als Matrix des euklidischen Raums, der Lie-Gruppe $SE(3)$ dargestellt werden:

$$\mathbf{M} = \begin{bmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0_{3\times 1}} & 1 \end{bmatrix} \in \mathbb{R}^{4\times 4}, \tag{4.40}$$

worin $\mathbf{R} \in \mathbb{R}^{3\times 3}$ die Rotation und $\mathbf{t} \in \mathbb{R}^3$ die Translation bezeichnen. Die Darstellung über $\mathfrak{se}(3)$ ist kompakter. Mit Spezialfällen der Exponentialfunktion $exp : \mathfrak{se}(3) \to SE(3)$, sowie der Logarithmusfunktion $log : SE(3) \to \mathfrak{se}(3)$ kann aus der zwischen den Repräsentationen gewechselt werden (vergleiche [7]).

Um einen Bildpunkt $\mathbf{p}$ des Bildes $\mathcal{I}^{(i)}$ eines Keyframes auf ein weiteres Bild $\mathcal{I}^{(j)}$ abzubilden, ist die Warping-Funktion w nötig (siehe Gl. 4.41). Sie hängt vom Tiefenwert $\mathcal{D}^{(i)}(\mathbf{p})$, sowie von der zu bestimmenden Kamerabewegung $\xi^{(\mathbf{ji})}$ ab.

$$\mathcal{I}^{(i)}(\mathbf{p}) = \mathcal{I}^{(j)}(w(\mathbf{p}, \mathcal{D}^{(i)}(\mathbf{p}), \xi^{(\mathbf{ji})})) \tag{4.41}$$

Diese Warping-Funktion projiziert $\mathbf{p}$ in das Koordinatensystem des Bildes $\mathcal{I}^{(j)}$ (Gl. 4.42)

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} := exp(\xi^{(\mathbf{ji})}) \begin{bmatrix} p_x/\mathcal{D}^{(i)}(\mathbf{p}) \\ p_y/\mathcal{D}^{(i)}(\mathbf{p}) \\ 1/\mathcal{D}^{(i)}(\mathbf{p}) \\ 1 \end{bmatrix} \tag{4.42}$$

und anschließend auf dessen Bildebene (Gl. 4.43)

$$w(\mathbf{p}, \mathcal{D}^{(i)}(\mathbf{p}), \xi^{(\mathbf{ji})}) := \begin{bmatrix} x'/z' \\ y'/z' \\ 1/z' \end{bmatrix}. \tag{4.43}$$

Die Schätzung $\hat{\xi}^{(\mathbf{ji})}$ der Kamera-Bewegung liefert die Gauß-Newton-Minimierung, ein Verfahren zur Lösung nicht linearer Gleichungssysteme (vergleiche [64]), des Varianz-normalisierten photometrischen Fehlers

$$E(\mathbf{p}, \xi^{(\mathbf{ji})}) = \sum_{\mathbf{p} \in \Omega^{(\mathcal{D}^{(i)})}} \left\| \frac{r_p^2(\mathbf{p}, \xi^{(\mathbf{ji})})}{\sigma^2_{r_p}(\mathbf{p}, \xi^{(\mathbf{ji})})} \right\|_\delta. \tag{4.44}$$

Dabei ist $\Omega^{(\mathcal{D}^{(i)})}$ die Menge der Pixel, über welche die Tiefenkarte $\mathcal{D}^{(i)}$ definiert ist. Die Normalisierung der Residuen $r_p(\mathbf{p}, \xi^{(\mathbf{ji})}) = \mathcal{I}^{(i)}(\mathbf{p}) - \mathcal{I}^{(j)}(w(\mathbf{p}, \mathcal{D}^{(i)}(\mathbf{p}), \xi^{(\mathbf{ji})}))$ (den Abweichungen vom gewünschten Ergebnis) mit ihrer Varianz $\sigma^2_{r_p}(\mathbf{p}, \xi^{(\mathbf{ji})})$ stellt sicher, dass Residuen mit einer hohen Unsicherheit schwächer in der Summe gewichtet werden. Die Varianz $\sigma^2_{r_p}$ kann mit dem Verfahren der Fehlerfortpflanzung (vergleiche [3]) geschätzt werden, damit die Unsicherheiten der Variablen, hier das Rauschen der Bild-Intensität und der Tiefenwert eines Pixels, in r_p übernommen werden. Für eine nicht-lineare Funktion f mit der Annahme von unabhängigen Variablen $x^{(1)}, x^{(2)}, \cdots, x^{(n)}$ und zugehörigen Varianzen $\sigma^2(x^{(i)})$ kann die Unsicherheit berechnet werden mit

$$\begin{aligned} &\sigma^2(x^{(1)}, x^{(2)}, \cdots, x^{(n)}) = \\ &(\frac{\partial f}{\partial x^{(1)}})^2 \sigma^2(x^{(1)}) + (\frac{\partial f}{\partial x^{(2)}})^2 \sigma^2(x^{(2)}) + \cdots + (\frac{\partial f}{\partial x^{(n)}})^2 \sigma^2(x^{(n)}) \end{aligned} \tag{4.45}$$

bzw. für die aktuellen Residuen somit durch

$$\sigma^2_{r_p}(\mathbf{p}, \xi^{(\mathbf{ji})}) = (\frac{\partial r}{\partial \mathcal{I}})^2 \sigma_I^2 + (\frac{\partial r}{\partial \mathcal{D}^{(i)}})^2 V^{(i)}(\mathbf{p}). \tag{4.46}$$

Hier wird angenommen, dass das Rauschen der Bild-Intensitäten normal-verteilt mit Varianz σ_I^2 ist.

$$\sigma^2_{r_p}(\mathbf{p}, \xi^{(\mathbf{ji})}) = 2\sigma_I^2 + (\frac{\partial r}{\partial \mathcal{D}^{(i)}})^2 V^{(i)}(\mathbf{p}) \tag{4.47}$$

Weiterhin wird in der Minimierungs-Funktion 4.44 mit $||.||_\delta$ die Huber-Norm verwendet, welche robust gegenüber Ausreißern ist [34]. In der Gauss-Newton-Minimierung dieser Funktion wird initial die Pose des zuletzt getrackten Frames verwendet.

Die Möglichkeit besteht, dass das Tracking auf dem Referenz-Keyframe fehlschlägt. Dies tritt ein, wenn ein großer Teil der mit der Warping-Funktion (Gl. 4.42) auf die Bild-Ebene transformierten Punkte außerhalb der gültigen Bild-Koordinaten liegt. Dies wird bspw. durch eine schnelle Kamera-Bewegungen verursacht. In diesem Fall wird eine Relokalisierung ausgelöst, die zum aktuellen Frame einen geeigneten ähnlichen Keyframe-Kandidaten sucht. Die Suche erfolgt über alle Keyframes im Graphen, jedoch in einer komplett zufällig gewählten Reihenfolge. Wird ein Kandidat mit einer guten Punktüberdeckung gefunden, so werden zusätzlich die Nachbar-Keyframes bzw. die im Graphen adjazenten Keyframes überprüft. Für diese wird jedoch ein schwächerer Schwellwert verwendet. Übersteigt die Anzahl an guter Nachbarn die Anzahl der schlechten, ist das Tracking auf dem Kandidaten erfolgreich.

Tiefenwert-Schätzung

Wenn sich die Position der Kamera zu weit vom aktuellen Keyframe entfernt, wird ein neuer Keyframe bestimmt und die Berechnung der Tiefenwerte für den alten Keyframe beendet. Um aussagen zu können, wann die Kamera zu weit entfernt ist, wird eine gewichtete Kombination aus relativer Distanz und Winkel zwischen den Positionen von Keyframe und Kamera betrachtet, welche durch die Funktion

$$dist(\xi^{(ji)}) = (\xi^{(ji)})^T \mathbf{W} \xi^{(ji)} \tag{4.48}$$

gegeben ist. Der Faktor $\mathbf{W}$ ist dabei eine Matrix mit der Gewichtung [21].
Jeder Keyframe verwaltet eine Tiefenkarte, die den Pixeln des Frames inverse Tiefenwerte zuweist. Nachdem ein neuer Keyframe gewählt wurde, wird dessen Tiefen-Karte mit Hilfe einer Projektion der im vorherigen Keyframe gefundenen inversen Tiefenwerten initialisiert. Danach wird in einem Regulierungsschritt jeder inverse Tiefenwert auf einen Wert gesetzt, der dem durchschnittlichen Wert seiner Nachbarn entspricht. Diese werden dabei durch ihre Varianz gewichtet. Nachbarn, deren Wert sehr stark vom Wert des betrachteten Punkts abweichen, mehr als die doppelte Varianz, werden nicht berücksichtigt. Dadurch bleiben die scharfen Kanten zwischen Flächen mit großem Höhenunterschied erhalten [24].
Um Ausreißer zu erkennen, verwaltet jede Tiefenwertschätzung eines Pixels einen Validitätswert, der verringert wird, wenn der Tiefenwert durch einen Stereo-Vergleich mit einem weiteren Frame bestätigt wurde und erhöht wird, wenn ein abweichender Wert festgestellt wurde. Überschreitet dieser Validitätswert für einen bereits gesetzten Tiefenwert bei einem Pixel eine festgesetzte Grenze, wird der Tiefenwert

verworfen. Unterschreitet der Validitätswert einer Tiefenwertschätzung einen gewissen Wert, wird er als korrekt angenommen und als neuer Tiefenwert für den Pixel gesetzt [24].
Jeder nicht als Keyframe gewählte Frame wird benutzt, um die Tiefenwerte des aktiven Keyframes zu verbessern. Dazu wird zunächst eine Teilmenge an Pixeln aus dem Frame gewählt, die eine ausreichende Genauigkeit für eine Verschiebungssuche haben. Für jeden Pixel dieser Teilmenge wird ein Referenzframe gewählt. Im optimalen Fall maximiert dieser Referenzframe die Genauigkeit der Suche, ohne den Suchbereich unnötig zu vergrößern.
Bei einem Stereovergleich zweier Bilder muss immer ein Kompromiss zwischen Präzision der Schätzung und Größe der Ungenauigkeit gemacht werden. Da die Kamera sich fortlaufend bewegt, wird der Abstand zwischen den Positionen eines Pixels innerhalb der verglichenen Frames größer je weiter diese zeitlich auseinander liegen. Dies ist in der Bilderfolge in Abbildung 4.5 dargestellt. Zur Auswahl des Referenz-Frames wird eine Heuristik benutzt, die den ältesten Frame wählt, auf dem der Pixel noch zu sehen ist und der innerhalb einer bestimmten Grenze für den Suchbereich liegt. Auf diesem wird eine eindimensionale Verschiebungssuche durchgeführt. Ist die Suche erfolglos wird das Alter des Frames erhöht und ein neuer Referenzframe gesucht. Bei der Verschiebungssuche wird entlang der Epipolarlinie des Referenzframes nach der Intensität des Pixels gesucht. Die Epipolarlinie eines Kamerabildes ist eine Gerade innerhalb des Bildes, die, in Abhängigkeit von einem weiteren Kamerabild der Szene und einem Bildpunkt in diesem, aus allen möglichen Positionen des korrespondierenden Bildpunktes besteht. Der Suchbereich kann dabei durch frühere inverse Tiefenwertschätzungen eingeschränkt werden [24].
Um für einen Pixel zu berechnen, ob er eine ausreichende Genauigkeit für einen Stereovergleich hat, werden drei wesentliche Faktoren betrachtet. Der erste Faktor ist der geometrische Verschiebungsfehler bzw. dessen Varianz $\sigma_{\lambda(\xi,\pi)}$, die von der relativen Kameraausrichtung ξ und der Kalibrierung π abhängt. Der geometrische Verschiebungsfehler beeinflusst die Berechnung der Epipolarlinie. Die Varianz des Fehlers lässt sich, wie in der Arbeit von Engel u. a. [24] beschrieben, als

$$\sigma^2_{\lambda(\xi,\pi)} = \frac{\sigma_l^2}{\langle \mathbf{g}, \mathbf{l} \rangle^2} \tag{4.49}$$

darstellen. Dabei ist $\mathbf{l}$ der normalisierte Richtungsvektor der Epipolarlinie, σ_l die Varianz des Positionierungsfehlers ϵ_l und $\mathbf{g}$ der normalisierte Bildgradient.
Der zweite Faktor ist der photometrische Verschiebungsfehler bzw. dessen Varianz

(a) Position des Pixels im Referenzbild

(b) Verschiebung zwischen ehemaliger und neuer Position des Pixels im nächsten aufgenommenen Bild

(c) Verschiebung zwischen ehemaliger und neuer Position des Pixels in einem kurze Zeit später aufgenommenen Bild

(d) Verschiebung zwischen ehemaliger und neuer Position des Pixels in einem einige Zeit später aufgenommenen Bild

Abbildung 4.5.: Verschiebung der Position eines Pixel innerhalb des aufgenommenen Bildes bei konstanter Kamerabewegung (Entnommen aus der Arbeit von Engel u. a. [24])

$\sigma_{\lambda(I)}$, die von der Intensität I der verwendeten Bilder abhängt. Durch den photometrischen Verschiebungsfehler wird die Suche nach einer Pixelübereinstimmung entlang der Epipolarlinie beeinflusst. Die Varianz des Fehlers lässt sich, wie in der Arbeit von Engel u. a. [24] beschrieben, als

$$\sigma^2_{\lambda(I)} = \frac{\sigma_i^2}{g_p^2} \tag{4.50}$$

darstellen. Dabei ist σ_i die Varianz des Fehlers der Bildintensität und g_p der Gradient der Bildintensität auf der Epipolarlinie.
Der dritte Faktor ist das Verhältnis zwischen Pixel und inverser Tiefe, welches in der Arbeit von Engel u. a. [24] als

$$\alpha = \frac{\delta_d}{\delta_\lambda} \tag{4.51}$$

beschrieben wird. Dabei ist δ_d die Länge des untersuchten Intervalls der inversen Tiefe und δ_λ die Länge des untersuchten Segments der Epipolarlinie.

Globale Graph-Optimierung

Bei monokularen SLAM-Verfahren ist ein großes Problem, dass reale Distanzen nicht gemessen, sondern nur im Maßstab zueinander angegeben werden können. Bei längerer Ausführung sind Abweichungen in diesem Maßstab unausweichlich. LSD-SLAM verwaltet einen Posen-Graphen $G = (V, E)$, in dem die Keyframes $K^{(i)}$ mit ihren geschätzten Posen als Knoten V und ihre relative Kamerabewegungen zueinander $\xi^{(ji)}$ als Kanten E gespeichert werden. Angenommen, die Drohne startet an einem Anfangspunkt und lässt das zu diesem Zeitpunkt aufgenommene Kamerabild $K^{(i)}$ von LSD-SLAM verarbeiten. Nach einem Kreisflug ist sie wieder an ihrer Ausgangsposition angelangt und ein zum Anfangspunkt ähnliches Bild $K^{(j)}$ wird anschließend verarbeitet. Erhofft wird, dass sich die Distanz-Abweichungen zwischen Anfangs- und Endpunkt nicht sehr weit unterscheiden und LSD-SLAM zu ihnen die korrekte Pose bestimmt hat. Während des Drohnen-Fluges ist ein größerer kommulierter Fehler in der Posen-Schätzung nicht auszuschließen. Dieser Fehler kommt deshalb zustande, da das Tracking eines Frames immer auf dem Referenz-Keyframe geschieht (und diese jeweils auf ihren Vorgängern). Der Posen-Graph, dessen Knoten Keyframes sind und welche durch ihre geschätzten Posen repräsentiert werden, soll dazu dienen, erneute Messungen, wie es im vorherigen Beispiel zwischen Anfangs- und Endpunkt geschieht, auszunutzen und Posen-Schätzungen nachträglich zu korrigieren. Eine erneute Messung zwischen zwei ähnlichen Keyframes geschieht dabei durch das Bestimmen ihrer Kamerabewegung, wie es bisher während des Trackings durchgeführt wurde. Seien an dieser Stelle $\xi_W^{(1)}, \xi_W^{(2)}, \cdots, \xi_W^{(n)} \in V$ die geschätzten Posen in Weltkoordinaten von n Keyframes, $\xi^{(ji)}$ die erneuten gemessenen relativen Kamerabewegungen von Keyframe $K^{(i)}$ nach $K^{(j)}$ und $\Sigma^{(ji)}$ eine zugehörige Matrix, welche die Unsicherheit ausdrückt, dann minimiert der Graph mit den Kanten E den Fehler

$$e(\xi_W^{(1)}, \xi_W^{(2)}, \cdots, \xi_W^{(n)}) := \sum_{(\xi^{(ji)}, \Sigma^{(ji)}) \in E} (\xi^{(ji)} \circ (\xi_W^{(i)})^{-1} \circ (\xi_W^{(j)}))^T \Sigma_{ji}^{-1} (\xi_{ji} \circ \xi_{W_i}^{-1} \circ \xi_{W_j}) \quad (4.52)$$

Der Term $\xi^{(ji)} \circ (\xi_W^{(i)})^{-1} \circ \xi_W^{(j)}$ drückt den Fehler zwischen der zuvor geschätzten Kamerabewegung $(\xi_W^{(i)})^{-1} \circ \xi_W^{(j)}$ und der neuen Messung $\xi^{(ji)}$ aus. Um das Problem der Maßstabsabweichung zu lösen, wird ausgenutzt, dass die Genauigkeit der Lokalisation von der Tiefe der Szene abhängt. Dazu werden die Tiefenkarten der Keyframes

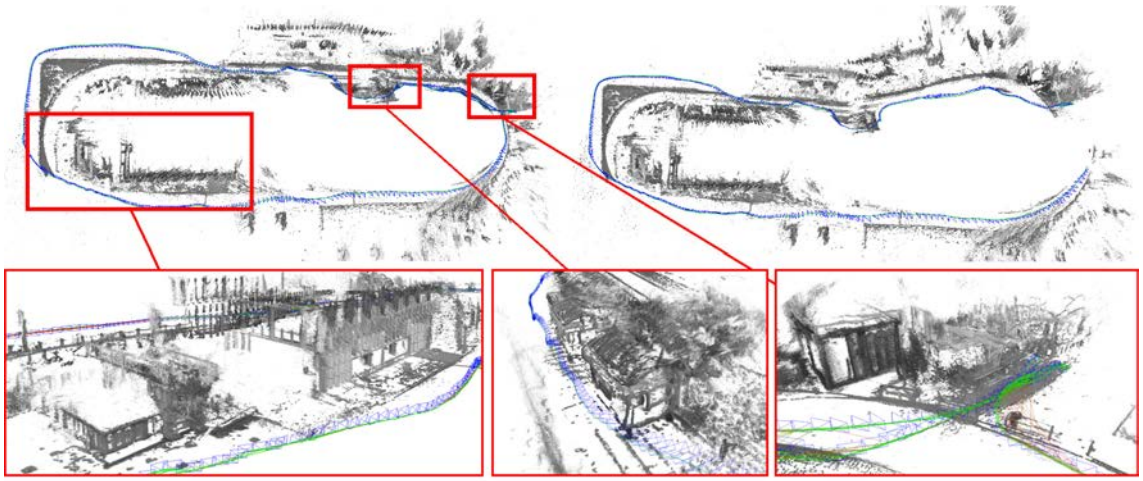


Abbildung 4.6.: Globale Karte eines längeren Durchlaufs vor der Schleifenerkennung (rechts) und nach der Schleifenerkennung (links)(Entnommen aus der Arbeit von Engel u. a. [21])

vor dem Einfügen in den Graphen so skaliert, dass der durchschnittliche Tiefenwert eins ist. Dadurch können die Kanten $\xi^{(ji)}$ als Elemente der Ähnlickeitstransformation $\mathfrak{sim}(3)$ dargestellt werden. Nach dem Einfügen eines neuen Keyframes, wird überprüft, ob der Keyframe eine Schleife in der Kamerabewegung schließt und daher, ob in dem neuen Keyframe $K^{(i)}$ eine bereits gesehene Umgebung zu sehen ist. Dazu wird zunächst eine Menge von Keyframes $K^{(j^{(1)})}, \cdots, K^{(j^{(n)})}$ zusammengestellt, die Kandidaten dafür sind, zusammen mit $K^{(i)}$ die Schleife zu schließen. Diese Kandidaten setzen sich aus den zehn Keyframes, die $K^{(i)}$ am nächsten sind, und einer von OpenFABMAP [32] berechneten Menge von Keyframes zusammen. Für jeden Kandidaten $K^{(j^{(k)})}$ werden unabhängig voneinander die Transformationen $\xi^{(j^{(k)}i)}$ und $\xi^{(ij^{(k)})}$ berechnet und überprüft, ob diese sich statistisch ähnlich sind. Dazu wird der Wert

$$\begin{aligned} &e(\xi^{(j^{(k)}i)}, \xi^{(ij^{(k)})}) = \\ &(\xi^{(j^{(k)}i)} \circ (\xi^{(ij^{(k)})}))^T (\Sigma^{(j^{(k)}i)} + Adj^{(j^{(k)}i)} \Sigma^{(ij^{(k)})} (Adj^{(j^{(k)}i)})^T)^{-1} (\xi^{(j^{(k)}i)} \circ \xi^{(ij^{(k)})}) \end{aligned} \tag{4.53}$$

berechnet und überprüft, ob er niedriger als eine gewählte Schranke ist. Diese wird vor Ausführung des Verfahrens vom Anwender festgelegt. Ist der berechnete Wert kleiner als die Schranke, werden $\xi^{(j^{(k)}i)}$ und $\xi^{(ij^{(k)})}$ zum Graphen hinzugefügt und somit die Schleife geschlossen. Wie in Abbildung 4.6 dargestellt ist, kann, durch das Erkennen von Schleifen und den damit verbundenen Informationen über die Skalierung der Umgebung und die aktuelle Position der Kamera, die gesamte globale Karte optimiert werden [21].

Nachdem ein passender Keyframe $K^{(j^{(k)})}$ gefunden wurde, kann der Maßstabsabweichung entgegengewirkt werden. Um dies zu tun, wird in der Arbeit von Engel u. a. [21] eine Methode vorgestellt, mit der zwei unterschiedlich skalierte Keyframes aneinander angepasst werden können. Der dabei zu minimierende Fehler

$$E_p(\xi^{(ji)}) = \sum_{\mathbf{p}\in\Omega(D^{(i)})} \left\| \frac{r_p^2(\mathbf{p},\xi^{(ji)})}{\sigma^2_{r_p}(\mathbf{p},\xi^{(ji)})} + \frac{r_d^2(\mathbf{p},\xi^{(ji)})}{\sigma^2_{r_d}(\mathbf{p},\xi^{(ji)})} \right\|_\delta \tag{4.54}$$

ist eine Erweiterung des Fehlers aus Gleichung 4.44. Zusätzlich zum photometrischen Residuum r_p wird dabei das Tiefenresiduum

$$r_d(\mathbf{p},\xi^{(ji)}) = \mathbf{p}'_3 - D^{(j)}(\mathbf{p}'_{1,2}) \tag{4.55}$$

und dessen Varianz

$$\sigma_{r_d}(\mathbf{p},\xi^{(ji)}) = V^{(j)}(\mathbf{p}'_{1,2})(\frac{\partial r_d(\mathbf{p},\xi^{(ji)})}{\partial D^{(j)}(\mathbf{p}'_{1,2})})^2 + V^{(i)}(\mathbf{p})(\frac{\partial r_d(\mathbf{p},\xi^{(ji)})}{\partial D^{(i)}(\mathbf{p})})^2 \tag{4.56}$$

benutzt, um Abweichungen der inversen Tiefenwerte zwischen Keyframes zu bestrafen [21]. Dabei ist $\mathbf{p}' = \omega_s(\mathbf{p}, \mathcal{D}^{(i)}((\mathbf{p})), \xi^{(ji)})$ der transformierte Bildpunkt. Die Minimierung selbst verläuft genau wie bereits bei der Gleichung 4.44. Zusätzlich wird der Graph im Hintergrund fortlaufend durch das Framework g^2o [49] optimiert.

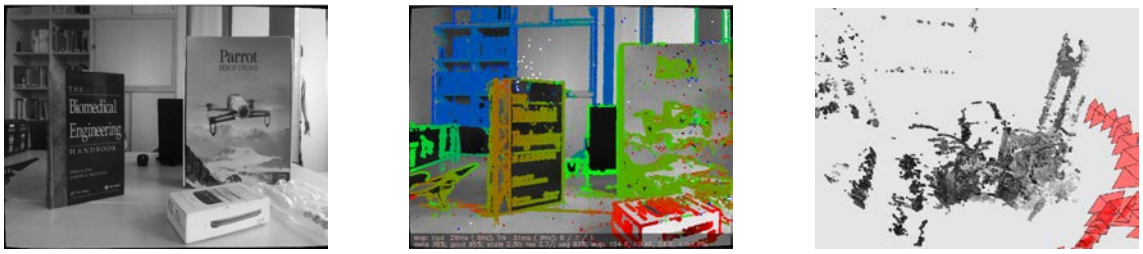
Initialisierungs-Phase

Die Tiefenwerte eines neuen Keyframes werden mit den Tiefenwerten überlappender Pixel des vorherigen Keyframes initialisiert. Zu beachten ist, dass für den ersten Keyframe keine initialen Schätzungen existieren. Die Autoren verwenden aus diesem Grund zunächst zufällige Tiefenwerte mit hoch gewählten Varianzen und erhoffen sich, dass die Frames, welche zur Verbesserung der Schätzungen im ersten Keyframe herangezogen werden, ausreichen, um Werte mit kleinen Varianzen zu erhalten. Alle darauf folgenden Keyframes sind von dieser Initialisierungs-Phase abhängig und folglich auch die Qualität der resultierenden Punktmenge.

4.3.3. Einsatz und Erweiterung von LSD-SLAM

Zunächst soll eine kurze Anmerkung zur Basis-Implementierung vorgestellt werden. Für LSD-SLAM existiert eine Implementierung[3], welche von den Autoren des

[3] `https://github.com/tum-vision/lsd_slam` Abruf: 28.06.16 12:00



(a) **(b)** **(c)**

Abbildung 4.7.: Zu jedem Keyframe (a) werden mit Tiefen (farblich kodiert in Abb. (b)) geschätzt, die abschließend zu einer Punktmenge (c) führen.

Original-Papers bereitgestellt wird. Sie ist aufgrund des Einsatzes einer GPL-Lizenz, ist die Verwendung und Modifikation für die Zwecke des Projekts möglich. Das Software-Paket beinhaltet eine Möglichkeit zur direkten Visualisierung des Trackings und der konstruierten Punktmenge während der Ausführung. Die Kommunikation zwischen LSD-SLAM und der zuletzt genannten Visualisierung ist mit Hilfe von ROS[4], einem modularen Framework mit einer Reihe an Algorithmen für den Einsatz in Roboter-bezogenen Projekten, realisiert und einige Programmteile sind von dessen Verwendung abhängig. Da sich die Projekt-Gruppe zu Beginn auf den Einsatz von Windows als das Host-System für ARGOS-Control (der auf einem externen Rechner auszuführenden Programm-Komponente) einigte, muss die LSD-SLAM-Implementierung, welche zunächst hauptsächlich auf Linux-Systemen ausführbar ist, entsprechend der Anforderungen angepasst werden. Eines der Probleme stellte dabei das ROS-Framework dar, da dessen Pakete für Ubuntu-Systeme angedacht sind. Windows-Varianten des ROS-Frameworks sind zwar verfügbar, die Einbindung gestaltete sich jedoch aufgrund inkompatibler Versionen als schwierig. Da der Einsatz von ROS sich hauptsächlich auf die Eingabe der Bild-Daten und der Kommunikation mit der Visualisierung beschränkt, wird in ARGOS-Control auf den Einsatz von ROS verzichtet. Etwaige Linux-bezogene Funktionen lassen sich problemlos ersetzen. Um einen Einblick in die Ausgabe-Daten zu erhalten, wäre es hingegen zusätzlich notwendig gewesen, die mitgelieferte Visualisierungs-Komponente ebenfalls anzupassen. Stattdessen wurde eine eigene Anwendung für die Visualisierung konstruiert, welche neben der resultierenden Punktmenge auch den Octree zu unterschiedlichen Baum-Tiefen darstellen kann, da der Octree letztendlich die Datenstruktur ist, mit der Hindernisse repräsentiert werden. Näheres zur Verwendung des Octrees in diesem Zusammenhang wird in einem der folgenden Kapitel 4.3.4 erläutert. Es sind einige

[4] `http://www.ros.org/` Abruf: 28.06.16 12:00

Anpassungen an der Basis-Implementierung vorzunehmen, sodass die Kommunikation mit den restlichen ARGOS-Control-Komponenten möglich ist.

Gewinnung der Punktmenge aus den Tiefenwerten

Sei $\mathbf{K}^{(cam)}$ (4.57) die Projektionsmatrix zu den intrinsischen Parametern der Kamera, welche im Lochkamera-Model betrachtet wird. $\mathbf{K}^{(cam)}$ projiziert einen 3D-Punkt des Kamera-Koordinatensystems $\mathbf{p}^{(cam)} := (\mathbf{p}_x^{(cam)}, \mathbf{p}_y^{(cam)}, \mathbf{p}_z^{(cam)})^T$ auf dessen Bildebene (4.58). Die Parameter (f_x, f_y) bezeichnen hier die Brennweite in Pixelgrößen und (c_x, c_y) sind die Koordinaten des Sensors.

$$\mathbf{K}^{(cam)} := \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \tag{4.57}$$

$$\mathbf{p}^{(cam)'} := \mathbf{K}^{(cam)} \cdot \mathbf{p}^{(cam)} = \begin{bmatrix} f_x \cdot \mathbf{p}_x^{(cam)} + c_x \cdot \mathbf{p}_z^{(cam)} \\ f_y \cdot \mathbf{p}_y^{(cam)} + c_y \cdot \mathbf{p}_z^{(cam)} \\ \mathbf{p}_z^{(cam)} \end{bmatrix} \tag{4.58}$$

Umgekehrt wird die bekannte Pixelkoordinate $\mathbf{q}^{(cam)}$ und die zugehörige Tiefe $z = 1/\mathcal{D}^{(i)}(\mathbf{q})$ mit der Umformung der Gleichung (4.58) zur (4.59) in den Punkt $\mathbf{p}^{(cam)}$ im Kamera-Koordinatensystem transformiert.

$$\mathbf{p}^{(cam)} = \begin{bmatrix} (\mathbf{p}_x^{(cam)} - c_x) \cdot \frac{z}{f_x} \\ (\mathbf{p}_y^{(cam)} - c_y) \cdot \frac{z}{f_y} \\ z \end{bmatrix}. \tag{4.59}$$

Wie in Kapitel 4.3.2 beschrieben ist, besitzt die Pose neben der Position und Translation einen Skalierungs-Wert s , der beim Einfügen eines Keyframes in den Posen-Graphen zunächst so gesetzt ist, sodass die auf ihren Mittelwert skalierten inversen Tiefendaten auf ihre usprünglichen Werte gebracht werden können, und anschließend wird der Skalierungs-Wert im Posen-Graph im Rahmen weiterer Optimierungen modifiziert. Die Skalierung s eines Keyframes wird in der Transformationsmatrix $\mathbf{M} = exp(\xi_{\mathbf{w}}^{(\mathbf{i})})$, die einen Punkt $\mathbf{p}^{(cam)}$ in homogenen Koordinaten vom Kamera-Koordinatensystem in den Welt-Koordinatensystem transformiert, berücksichtigt

(siehe Gleichung (4.60)).

$$\mathbf{p}^{(world)'} := \begin{bmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0_{1\times 3}} & s \end{bmatrix} \cdot \begin{bmatrix} \mathbf{p}^{(cam)} \\ 1 \end{bmatrix} = \begin{bmatrix} x' \\ y' \\ z' \\ s \end{bmatrix}, \mathbf{p}^{(world)} := \begin{bmatrix} x/s \\ y/s \\ z/s \end{bmatrix} = \begin{bmatrix} x \\ y \\ z \end{bmatrix}. \tag{4.60}$$

Heuristische Parameteroptimierung

In diesem Unterkapitel wird beschrieben, wie vorgegangen wurde, um eine geeignete Parametrisierung von LSD-SLAM zu finden, die mit der Kamera der AR.Drone2 vereinbar ist und gute Ergebnisse liefert. Die Wahl erfolgt heuristisch nach Beurteilung ihrer individuellen Auswirkungen und stellt keineswegs die beste Vorgehensweise bei der Suche dar.

Für den Einsatz des Verfahrens ist zunächst die Bestimmung der intrinsischen Kamera-Parametern nötig, für die OpenCV[5] Methoden bereit stellt. Die intrinsischen Parameter der Drohnen-Kamera sind nicht nur nötig, um die geschätzten Tiefenwerte in entsprechende Punkte zu überführen, sondern sie finden auch in der Tiefenwert-Schätzung Verwendung, weswegen die Qualität der Resultate von ihnen abhängt.

Nach den Hinweisen der LSD-SLAM-Autoren, sind für gute Ergebnisse eine Kamera mit einer hohen Bildrate von mindestens 30 Bildern pro Sekunde (FPS) empfohlen, die ein Sichtfeld von circa 130° oder höher aufweist [22]. Die Drohnen-Kamera hält die FPS-Anforderung ein, es muss jedoch auch die drahtlose Datenübermittlung und die dabei entstehende Bildqualität beachtet werden (Kapitel 3.3), weswegen höhere Auflösung nicht in Betracht kommen. Nach Aussagen der Autoren, sollen Auflösungen von 640×480 ausreichen und eigene Tests haben dies in Anbetracht höherer Auflösungen bestätigt. Die AR.Drone2.0 liefert zudem ein Sichtfeld von 92° in der Diagonale. Es gibt eine Bevorzugung von Kameras mit Global-Shutter gegenüber Rolling-Shutter, allerdings kann dies von keinen der verfügbaren Drohnen-Kameras eingehalten werden. Bei Rolling-Shutter-Kameras werden Bildpunkte nicht zur selben Zeit, sondern schrittweise vertikal oder horizontal zeitversetzt aufgenommen. Dies kann bei schnellen Bewegungen zum Rolling-Shutter-Effekt führen, sodass Bilder verzerrt erscheinen. Um das Gewicht der Drohne nicht unnötig zu erhöhen und damit die Flugzeit zu verringern (oder weitere Probleme für den Flug zu verursachen), wurde daher beschlossen, die bereits existierende Drohnen-Kamera zu ver-

[5] `http://opencv.org/` Abruf: 28.06.16 11:00

wenden.
Die vorliegende Implementierung von LSD-SLAM lässt sich über eine Reihe von Parametern konfigurieren. Dieses Kapitel beschränkt sich auf eine Auswahl an Parametern, dessen Auswirkungen auch in [22] beschrieben werden. Aufgrund fehlender Dokumentation konnte allerdings nicht zu jedem Parameter die genauer Bedeutung erschlossen werden. Ein Teil erfolgt durch die Interpretation der vorliegenden Implementierung.

Minimaler Gradient eines Pixels Je kleiner dieser gewählt wird, desto mehr wird von feineren Strukturen, Pixeln mit niedrigen Gardienten, im Bild bei der Tiefenwert-Schätzung Gebrauch gemacht.

Rauschen der Bild-Intensität Die Konstante wird beispielsweise in der Berechnung der Gewichtungen zur Selbstlokalisierung als σ_I^2 (siehe Kapitel 4.47) berücksichtigt.

Gewichtung für die Distanzen Durch eine Gewichtung der Distanzen von Keyframes wird die Menge an erzeugten Keyframes anhand der Distanzen ihrer Positionen beeinflusst. Ist die Distanz der neu geschätzten Pose zu der des Referenz-Keyframes größer als der von diesem Parameter abhängigen Schwellwert, wird ein neuer Keyframe erzeugt. Dies bedeutet, dass je größer der Wert gesetzt wird, desto mehr Keyframes werden erstellt.

Gewichtung für die Punktüberdeckung Die Menge an erzeugten Keyframes wird durch eine Gewichtung in der Punktüberdeckung zwischen dem aktuellen Frame und dem aktiven Keyframe beeinflusst. Auch hier führt ein großer Wert zu mehr Keyframes.

maximale Anzahl an Kanten bzw. Kandidaten , die beim Einfügen eines Keyframes in den Graphen bestimmt werden.

Weitere Parameter, die oben nicht aufgeführt sind, ermöglichen es Funktionalitäten, wie etwa die Verwendung des Graphen, abzuschalten, oder abweichende Funktionalitäten hinzuzufügen.
Je nach Wahl der Wertebereiche, kann die Suche nach geeigneten Parametern durch Testen aller Kombinationen lange andauern. Deswegen sollen an dieser Stelle Vorüberlegung zu den Parametern besprochen werden, um im Vorfeld bereits ungünstige Werte auszuschließen und so den Suchraum eingrenzen.

Die Pixel haben einen Graustufen-Wert zwischen 0 und 255. Der minimale Gradient eines Pixels führt bei einem hohen Wert dazu, dass Bilder viel "Struktur" (Bereiche mit Pixeln, dessen Nachbarn hohe Helligkeitsunterschiede aufweisen) enthalten müssen, um bei der Tiefenwert-Schätzung beachtet zu werden. Sollte ein hohes Bildrauschen enthalten sein, darf der Wert nicht zu niedrig sein. Ein Wert von 0 wirkt unrealistisch und Werte von über 10 haben häufig dazu geführt, dass nur wenige Pixel als relevant betrachtet wurden, sodass letztendlich das Tracking (Kapitel 4.3.2) häufig in den ersten Tests verloren ging. Für den Wertebereich wurden demnach Werte zwischen 1 und 10 in Einer-Schritten untersucht.

Die Verwendung der Konstante des Bildrauschens fand in der Implementierung an mehreren Stellen statt, ihr genauer Einfluss konnte jedoch nicht erschlossen werden, weswegen spontan quadratische Werte $4, 9, 16, 25$ getestet wurden. Bei größeren Werten schlug das Tracking selbst bei kleinen und langsamen Bewegungen fehl.

Die Gewichtungen für die Distanzen und Punktüberdeckungen von Keyframes beeinflusst, wie häufig Frames zu Keyframes ernannt werden. Die Distanz-Gewichtung wird als Schwellwert für den quadratischen Abstand zwischen Posen zweier Frames verwendet. Die Wahl des Wertebereiches diesen Parameters erweist sich als schwierig, da Distanzen in LSD-SLAM keinen metrischen Distanzen entsprechen müssen. In Abschnitt 4.3.3 werden Lösungen angesprochen, mit denen metrische Distanzen im Graph angestrebt werden, wofür jedoch die Odometrie-Daten der Drohne nötig sind.

Bei der Wahl der maximalen Anzahl an Kanten, die einem Keyframe beim Einfügen zugeordnet werden, wurde vom Referenz-Wert von 10 Kanten, wie er in [21] verwendet wird, nur abgewichen, wenn während der Tests Probleme mit dem Schließen von Schleifen oder dem Wiederaufnehmen der Selbstlokalisation auftraten.

Die Beurteilung der Parameter-Konfiguration erfolgt über die subjektive Wahrnehmung der resultierenden Punktmenge bei Eingabe des selben Bild-Materials. Bei der Aufnahme des Test-Videos wird auf die Initialisierungs-Phase (Kapitel 4.3.2) acht gegeben, indem zu Beginn langsame Translationen ohne Rotationen durchgeführt werden. Beachtet werden sollte jedoch, dass hierdurch ein Überanpassung der Parameter an das eine Testvideo entstehen kann, weswegen gute Ergebnisse mit weiterem Bild-Material überprüft werden muss. Subjektiv gute Ergebnisse, sind Punktmengen, die ein geringes Rauschen aufweisen und dessen Tiefenwert-Schätzungen plausibel erscheinen. Die Pfadplanung sollte sich nicht mit vielen Hindernissen aufhalten, die nicht existieren.

(a) (b)

Abbildung 4.8.: Darstellung der resultierende Punktmenge im Bild (b) für das Test-Video im Bild (a).

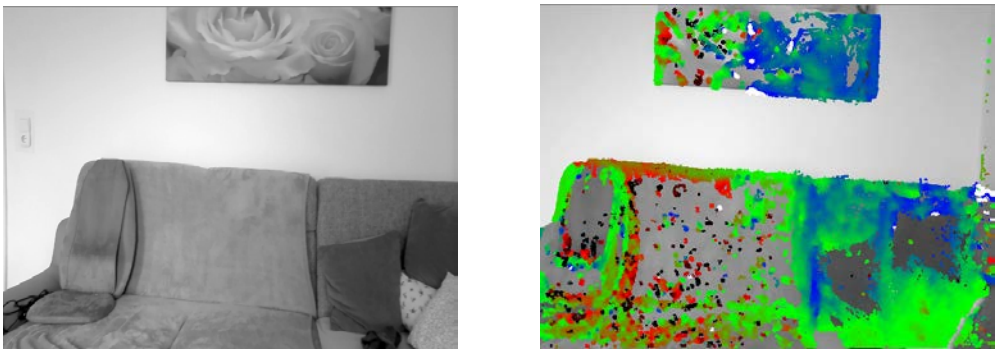
Initialisierungs-Phase

Wie in Kapitel 4.3.2 erwähnt wird, kann eine schlechte Initialisierung zu einer schlechten Gesamtqualität der Resultate führen. Ein "schlechter" Start muss aus diesem Grund möglichst vermieden oder, falls eine Meidung nicht möglich war, erkannt und behandelt werden. Sie treten etwa bei ungünstig gewählten Parametern, Umgebungseinflüssen und drastischen Kamera-Bewegungen ein. Zur Beobachtung der Tiefenwert-Schätzungen steht eine farblich dargestellte Ausgabe der inversen Tiefenwerte zur Verfügung. Ein Beispiel für schlecht geschätzte Tiefenwerte in einem Keyframe ist in der Abbildung 4.9b veranschaulicht. Tiefen sind an den Färbungen zu erkennen, wobei rot etwa nahe Punkte und blau eher ferne Punkte andeutet. In der angegebenen Abbildungen nähern die geschätzten Tiefen eindeutig nicht die tatsächlichen Tiefen an und sind verrauscht.

Zur Abhilfe empfehlen die Autoren, Engel u. a., dass in den ersten Frames Objekte mit klar erkennbaren unterschiedlichen Tiefen vorzufinden sind. Zudem sollte sich die Kamera-Bewegung auf langsame Translationen beschränken [22]. Die Drohnen-Kamera sollte nicht gegen eine leere Wand gerichtet sein und langsam abheben, sowie ruckartige Bewegungen vermeiden, um die Konstruktion zu Beginn vieler Keyframes hinauszuzögern, damit Tiefenwerte der ersten Keyframes durch möglichst viele Frames verbessert werden können.

Anstatt konstante Gewichtungen für die Häufigkeit von Keyframes zu verwenden (Kap. 4.3.3), kann beim Start zunächst unterstützend ein kleiner Wert gewählt werden, wenn zumindest in dieser Phase langsame Bewegungen und die Meidung von Kamera-Rotationen eingehalten wird. Der Parameter kann anschließend für die folgenden Keyframes angehoben werden.

Eine weitere Möglichkeit besteht darin, die Tiefensensor-Kamera, dessen Daten oh-



(a) **(b)**

Abbildung 4.9.: Beispielhafte Darstellung von schlechten Tiefenwert-Schätzung (Abb. (b)) in einem Keyframe zum Bild (a). Hier stellen im Abbild (b) rote Färbungen geringe Tiefe bzw. blaue Färbungen hohe Tiefen dar.

nehin zwecks Rekonstruktion von der Drohne an den externen Rechner übermittelt werden, zur Hilfe zu nehmen. Die Übergabe der Bilder an LSD-SLAM wird solange ignoriert, bis das erste Tiefenbild dieser Tiefensensor-Kamera verfügbar ist. Dessen Tiefenwerte werden auf den Bildbereich des ersten Keyframes $\mathcal{K}^{(0)}$ projiziert. Sei die Fotokamera der Drohne an Position $(0,0,0)^T$ angebracht und sei $\mathbf{H}^{(depth)}$ die Transformationsmatrix zu den extrinsischen Parameters der Tiefensensor-Kamera, die einen Punkt aus dem Koordinatensystem der Tiefensensor-Kamera in das Koordinatensystem der Fotokamera transformiert. Seien zudem $\mathbf{K}^{(cam)}$ und $\mathbf{K}^{(depth)}$ die jeweiligen Transformationsmatritzen der intrinischen Parameter. Um nun einen Pixel $\mathbf{q}^{(depth)}$ mit bekannter Tiefe auf die Bildebene der Fotokamera zu projizieren, welche den Bildstrom für LSD-SLAM liefert, wird der entsprechende 3D-Punkt $\mathbf{p}^{(depth)}$ zu $\mathbf{q}^{(depth)}$ mit $\mathbf{K}^{(depth)}$ wie in Gleichung (4.59) berechnet. Anschließend wird $\mathbf{p}^{(depth)}$ durch $\mathbf{p}^{(cam)} := \mathbf{H}^{(depth)} \cdot \mathbf{p}^{(depth)}$ in das Koordinatensystem der Fotokamera überführt und mit Gleichung (4.58) über $\mathbf{K}^{(cam)}$ auf dessen Bildebene in den Pixel $\mathbf{q}^{(cam)}$ abgebildet (vergleiche Abbildung 4.10). Liegt $\mathbf{q}^{(cam)}$ außerhalb der gültigen Pixelkoordinaten für die Bildebene, so wird $\mathbf{q}^{(cam)}$ verworfen. Es werden offensichtlich nicht jedem Pixel des ersten Keyframes $\mathcal{K}^{(0)}$ ein gültiger Tiefenwert zugeordnet. Jedem Pixel, welchem auf diese Weise kein Tiefenwert vergeben wird, erhält eine negative Varianz, wie sie bereits verwendet wird, um nicht existierende Tiefenwerte zu vermerken. Allen anderen Pixeln wird eine initiale Varianz zugeordnet, die vom Benutzer gesetzt festgelegt wird.

Zur Erkennung eines schlechten Starts wird eine heuristische Methode verwendet, bei denen die Varianzwerte $\mathcal{V}^{(i)}$ (Kapitel 4.3.2) als Anhaltspunkt dienen. Untersucht werden dazu die ersten n Keyframes und von ihnen jeweils der Mittelwert ihrer

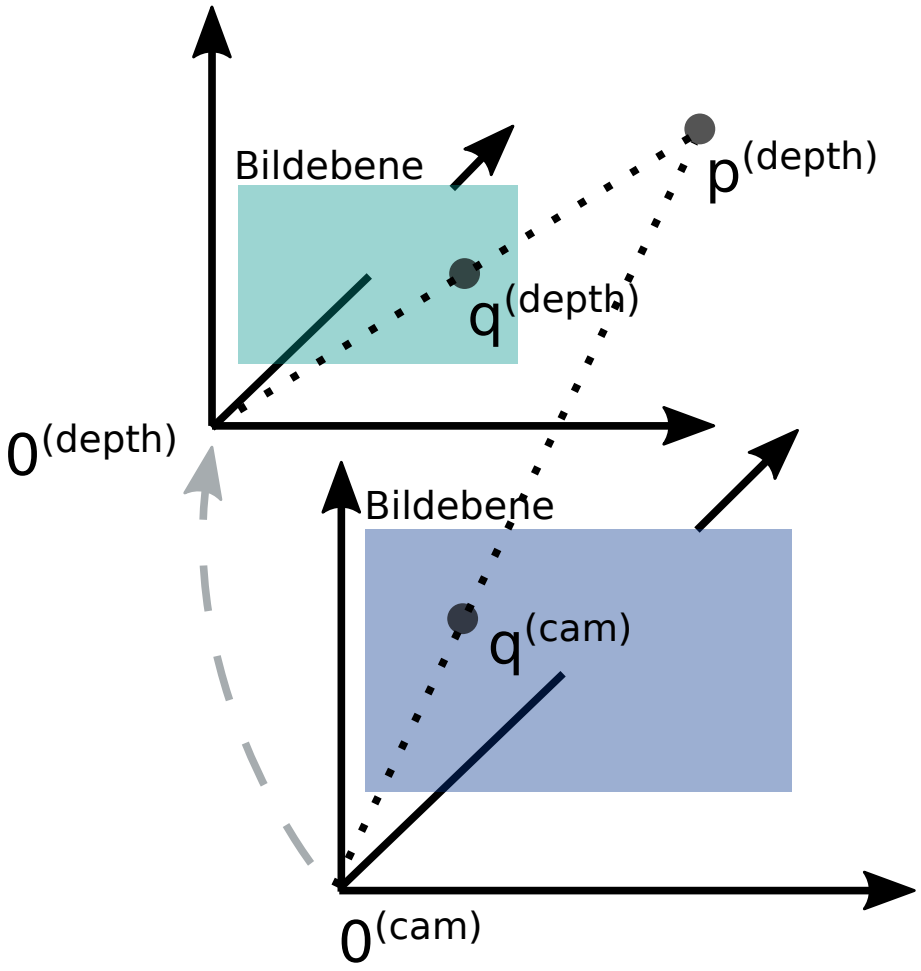


Abbildung 4.10.: Skizze zur Projektion eines Pixels von der Bildebene der Tiefensensor-Kamera (Koordinatensystem mit Ursprung $0^{(depth)}$) auf die Bildebene der Drohnen-Kamera (Koordinatensystem mit Ursprung $0^{(cam)}$)

Varianzwerte. Liegen diese über einem zuvor gewählten Schwellenwert, wird das System zurückgesetzt. Die Übergabe der Daten an die Bahnplanung findet erst nach der Initialisierungs-Phase statt, um eventuelle Inkonsistenzen mit den Daten im Octree (Kapitel 4.3.4) zu vermeiden.

Behandlung von Ausreißern

Die Pixel in einem Keyframe, für die keine Tiefenwerte berechnet wurden, sind in der Datenstruktur an negativen Varianz-Werten erkennbar. Jedem anderen Pixel wurde eine Tiefenwert-Schätzung und eine zugehörige Varianz (siehe Kap. 4.3.2) zugewiesen. Eine ungefilterte Verwendung dieser Werte würde zu einer verrauschten Punktmenge führen, was wiederum ein Problem in der Bahnplanung bedeuten würde, da sich die Drohne unnötig mit nicht vorhandenen Hindernissen aufhalten

muss.
Die Betrachter-Komponente der Implementierung von LSD-SLAM macht hierzu von den Varianz-Werten Gebrauch [22] und verwirft Punkte, die mit hohen Varianzen assoziiert werden. Die Abbildungen 4.11a - 4.11c zeigen die Auswirkungen auf Resultate bei Verwendung von unterschiedlichen Schwellenwerten. Zum anderen wird eine Methode verwendet, die ähnliche benachbarte Pixel berücksichtigt, welche die Schätzung unterstützen bzw. sich von ihren Nachbarn nicht stark unterscheiden. Dabei ist ein benachbarter Pixel $\mathbf{p}^{(j)}$ in der Nachbarschaft des Punktes $\mathbf{p}^{(i)}$ anhand der Bildpunkte eines Keyframes (bspw. in einem 3×3-Fenster) dem Punkt $\mathbf{p}^{(i)}$ ähnlich, wenn der Abstand $||\mathbf{p}^{(i)} - \mathbf{p}^{(j)}||_2$ klein ist, bspw. kleiner als der Wert $2\mathcal{V}(\mathbf{p}^{(i)})$. Gezählt wird die Anzahl Pixel in einer gewählten Nachbarschaft, die dies erfüllen und $\mathbf{p}^{(i)}$ wird als Ausreißer klassifiziert wenn wenige ähnliche Punkte vorzufinden sind. $\mathbf{p}^{(i)}$ wird folglich ignoriert. Der Einsatz dieser Methode findet auch in diesem Projekt statt, die nach subjektiver Einschätzung gute Ergebnisse liefert (vergleiche Abbildung 4.7).
Zur Entfernung der verbleibenden Punkte, die sich durch ihren Abstand zu anderen Punkten bemerkbar machen und in der Verwendung des Octrees (siehe Kapitel 4.3.4) störend sind, werden weitere Methoden verwendet. Getestet wurde unter anderem die Radius-Ausreißer-Methode. Weiterhin wurde eine Mittelwert-Filterung im dreidimensionalen Raum in Betracht gezogen, bei der Punkte in Volumen mit einer geringen Punktanzahl verworfen werden und dadurch eine reduzierte resultierende Punktmenge entsteht. Beide Methoden behandeln die Punkte einzelner Keyframes im Weltkoordinatensystem, wodurch zu viele korrekte Punkte verworfen werden, die bei Betrachtung der kompletten Punktmenge aller Keyframes verblieben wären. Eine Lösung besteht darin, zunächst mehrere Keyframes zu puffern und die Methoden verzögert anzuwenden. Dies würde allerdings die Hindernis-Erkennung zusätzlich verlangsamen, was beim Drohnen-Flug im schlimmsten Fall zu einer Kollision führt. Da die Punkte nicht direkt, sondern mit Hilfe des Octrees übergeben werden, entstand eine andere Idee, bei der eine ähnliche Methode direkt auf diesem angewendet wird. Sie ist in Kapitel 4.3.4 näher beschrieben. Zu bedenken ist, dass hiermit nachträglich Punkte verloren gehen.

Einbindung der Drohnen-Odometrie

Ein Problem von LSD-SLAM ist, dass die tatsächliche Skalierung der Welt nicht bekannt ist. Die Distanzen zweier Keyframe-Posen entsprechen folglich keinen metrischen Werten, was ein großes Problem ist, wenn LSD-SLAM eingesetzt wird um

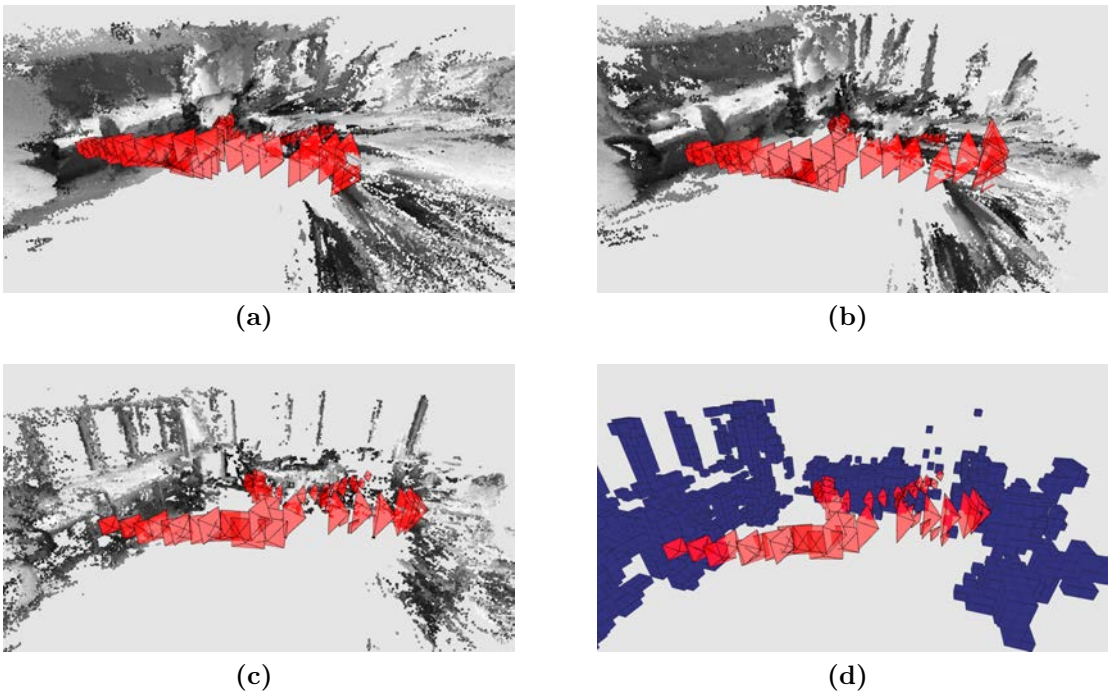


Abbildung 4.11.: Die Bilder (a), (b) und (c) zeigen die Auswirkungen der Erhöhung des Varianz-Schwellwerts auf die Punktmenge zum Test-Video aus Abbildung 4.8b. Das Bild (d) zeigt die Voxel des Octrees nach der Entfernung zusätzlicher Ausreißer aus der Punktmenge des Bildes (c).

die Drohne zu lokalisieren.

Die Autoren Engel u. a. schlagen hierzu Lösungen vor (vergleiche [22]), mit denen zusätzliche bekannte Information eingebunden werden können, sodass im Posen-Graph (Kapitel 4.3.2) die korrekten Skalierungen propagiert wird.

Im g^2o-Hypergraph wird nebenläufig eine Minimierung der Fehlerfunktion (Gl. 4.52) über alle Restriktionen $(\xi^{(ji)}, \Sigma^{(ji)})$, den Kanten des Graphen, durchgeführt. $Sim(3)$-Posen stellen in diesem Zusammenhang die Knoten dar und die Funktionen e_k berechnen einen Fehler zwischen zwei Posen. Für nähere Informationen zur Verwendung des Graphen sei auf [33] verwiesen. Ein Vorschlag ist es, bekannte Distanzen zweier skalierter Keyframe-Posen als zusätzliche Kanten einzubinden, sodass die tatsächlichen metrischen Distanzen in der Minimierung berücksichtigt werden. Auf der Drohne werden hierzu eigene Schätzungen der Odometrie bestimmt. Daraus ergibt sich, dass zwischen dem ersten Keyframes mit Pose im Ursprung und dem neuen Keyframe eine Odometrie-Kante hinzugefügt wird. Die Anzahl an Kanten, welche anhand der Ähnlichkeits-Tranformation bestimmt werden (Gl. 4.53), wird zusätzlich von den im Artikel [21] vorgeschlagenen 10 reduziert, sodass die neuen Kanten höher gewichtet werden.

Die Verwendung der RGB-D-Kamera wäre eine weitere Möglichkeit, mit der zudem das Problem in der Initialisierungs-Phase (Kapitel 4.3.3) reduziert werden kann. Ist die korrekte Skalierung eines Keyframes bekannt, so kann eine unäre Kante auf den entsprechenden Knoten gesetzt werden, die mit Hilfe einer geeigneten Fehlerfunktion die Änderung der Skalierung bestraft. Zusätzlich können beim ersten Keyframe, anstelle einer zufälligen Initialisierung der inversen Tiefen-Werte, die Daten eines mit der RGB-D-Kamera erhaltenen Tiefenbildes genutzt werden, um dem zuvor beschriebene Problem in der Initialisierungs-Phase (Kapitel 4.3.2) vorzubeugen. Hierzu muss das Bild der Front-Kamera, sowie das Tiefenbild der RGB-D-Kamera zeitlich nah konstruiert worden sein und die Tiefen-Daten über bekannte extrinsische Parameter auf den Keyframe abgebildet werden.

4.3.4. Effiziente Datenhaltung

Um die im Abschnitt 4.3.3 generierte Punktmenge in einer Datenstruktur zu sammeln, damit diese an die Pfadplanung gesendet und dort ein Pfad zwischen den erkannten Hindernispunkten erzeugt werden kann, wird ein Octree als gemeinsame Datenstruktur verwendet. In diesem Abschnitt wird zunächst erklärt, was ein Octree im Allgemeinen ist und anschließend die für diesen Anwendungsfall implementierte Version mit den zugehörigen Verbesserungen, die aus dem Anwendungsfall entspringen, erläutert.

Octree

Ein Octree ist ein Baum, in dem jeder Knoten bis zu acht Kinder haben kann. Diese Datenstruktur eignet sich zum Zerlegen eines dreidimensionalen Raumes, da jedem Knoten im Octree ein dreidimensionaler Vektor zugeordnet wird, der dessen Position bestimmt. Durch diesen Vektor werden drei Ebenen $x-y, x-z, y-z$ gelegt, wodurch ein Knoten im Octree den Raum in acht Teile spaltet. Jeder dieser Teile ist eindeutig dadurch definiert, ob er sich linksseitig oder rechtsseitig der einzelnen Ebenen befindet. Diese acht Teilräume können anschließend wiederum durch einen Kindknoten des gerade betrachteten Knotens unterteilt werden. Um dreidimensionale Punkte in einem Octree zu speichern, kann die Punktmenge auf jeder Ebene des Octrees entsprechend der Zerlegung des Raums, die durch die Knoten des Octrees induziert wird, aufgeteilt werden und rekursiv in die Octrees mit den Kindknoten als Wurzel eingefügt werden. Die Wahl der Koordinaten der Knoten des Octrees und die Speicherung der Punkte kann dabei nach verschiedenen Verfahren vorgenommen werden.

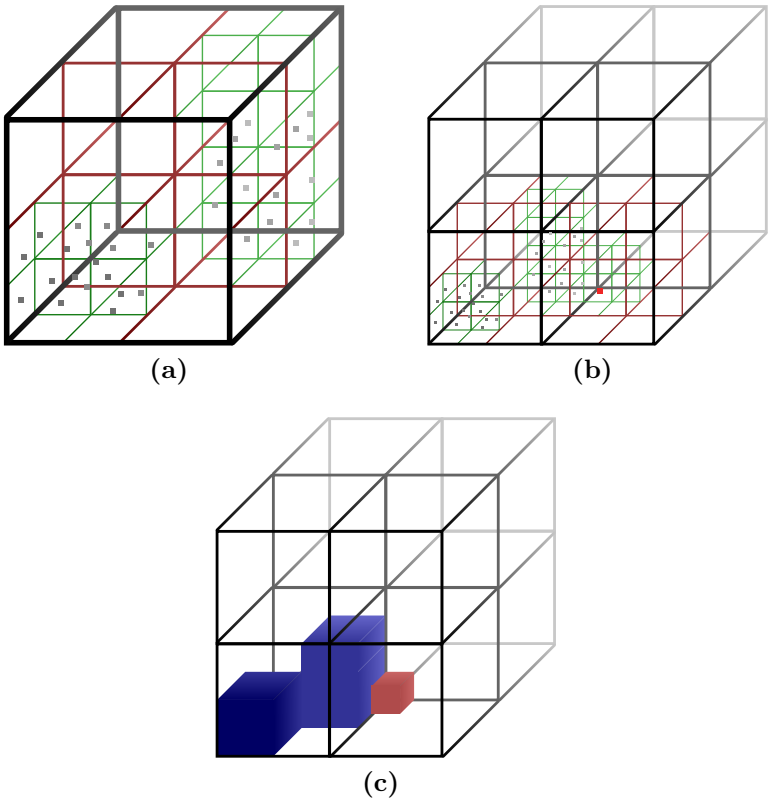


Abbildung 4.12.: In (a) ist der implementierte Octree visualisiert, der durch das Einfügen der grau dargestellten Punkte erzeugt wird. Die Erweiterung des repräsentierten Raums durch Hinzufügen außerhalb liegender Punkte ist in (b) dargestellt. Die durch den Octree in (b) induzierten Volumina bzw. Hindernisse sind in (c) abgebildet.

Eine Möglichkeit ist es, die Punkte selbst als Koordinaten der Knoten des Octrees zu verwenden. Hierfür kann zum Beispiel in jeder Iteration ein möglichst zentraler Punkt, der die restliche Punktmenge möglichst gleichmäßig zerlegt, als Knoten des Octrees verwendet werden. Das Einfügen der restlichen Punktmenge erfolgt dann rekursiv, bis jeder Punkt einem Knoten im Octree zugeordnet ist. Dieses Verfahren setzt voraus, dass alle Punkte im Vorhinein bekannt sind, da sonst die Auswahl von Punkten als Knoten beliebig schlecht werden kann und der Baum im schlimmsten zu einer Liste degeneriert. Da in dem hier beschriebenen Anwendungsfall die Daten nicht a priori bekannt sind, sondern im Verlaufe des Fluges generiert werden, eignet sich dieser Ansatz also nicht. Eine Alternative dazu bietet die Anordnung der Knoten des Octrees entlang einem regulären Gitter, wobei jeder Kindknoten den Teilraum, in dem er liegt, entlang der Achsen halbiert. Diese rekursive Zerlegung des Raums wird durchgeführt, bis in jedem Blatt nur noch ein Punkt liegt. Für den Ansatz mit einem regulärem Gitter ist es allerdings notwendig, dass der Wurzel des Octree ein

initiales Volumen, d. h. ein achsenparalleler Quader, zugeordnet ist, in dem sich alle Punkte befinden. Für den Fall, dass ein Raum erkundet werden soll, dessen Ausmaße nicht a priori gegeben sind, ist diese Wahl nicht immer möglich oder sinnvoll. Außerdem ist die Größe der Datenstruktur linear in der Anzahl der Punkte, die für einen Flug sehr groß werden kann. Die Implementierung des verwendeten Octrees vermeidet diese Probleme, indem ein Wachstum des Octrees außerhalb des initialen Volumens ermöglicht und die Punktmenge durch Diskretisierung komprimiert wird.

Implementierung des Octrees

Wie im Abschnitt 4.3.4 erwähnt, basiert der verwendete Octree auf dem Ansatz regulärer Gitter. Dabei beschreibt jede Zelle im Gitter einen Würfel. Um sowohl den Speicherbedarf des Octrees, als auch die Laufzeit für das Einfügen und Auslesen zu reduzieren, werden die Volumina nur bis zu einer minimalen Kantenlänge min_length zerlegt. Das Einfügen eines Punktes in einen Teilbaum, der nicht weiter zerlegt werden darf, kann durch das Verwerfen dieses Punktes realisiert werden. Auf diese Art wird die repräsentierte Punktmenge auf Volumina mit einer Kantenlänge von mindestens min_length diskretisiert. Für den Anwendungsfall der Speicherung von Informationen über Hindernisse in einem Raum ist dieses Vorgehen angemessen, denn im Rahmen des Fluges, soll die Drohne einen Mindestabstand zu Hindernissen einhalten. Durch Diskretisierung auf Volumina mit mindestens der Kantenlänge min_length entsteht auf diesem Mindestabstand im schlimmsten Fall ein Fehler von $\sqrt{3} \cdot min\_length$, also der Diagonalen des Volumens. Dieser Fehler kann die Repräsentation der Hindernisse jedoch nur vergrößern, wodurch aus dem Fehler nicht resultieren kann, dass die Drohne mit einem Hindernis kollidiert. Bei geeigneter Wahl von min_length kann der Fehler beliebig klein werden. Die Speichergröße des Octrees hängt dabei direkt von der Tiefe des Octrees und somit von min_length ab, da dieser Octree mit einer Tiefe h maximal einen Würfel mit Kantenlänge $min\_length \cdot 2^h$ erfassen kann. Im Rahmen der Experimente wurde $min\_length = 10\,\text{cm}$ verwendet. Durch diese Wahl beträgt der Fehler maximal 17,3 cm und ein Octree mit Tiefe 10 kann bereits einen Würfel mit Kantenlänge über 100 m erfassen. Die Tiefe dieses Octrees kann im praktischen Fall als konstant angenommen werden, wodurch auch sämtliche Zugriffszeiten im Octree konstant sind.
Die zweite Anpassung des Octrees, an den hier präsentierten Anwendungsfall, ist das dynamische Wachstum, folglich das Vergrößern des repräsentierten Volumens. Wird beim Einfügen eines Punktes festgestellt, dass dieser nicht im Volumen der Wurzel liegt, so wird für die Wurzel ein Elternknoten generiert, der als neue Wurzel des

Octrees fungiert. Um das Zentrum des Volumens der neuen Wurzel zu bestimmen, wird die Ecke des Volumens der alten Wurzel gewählt, in deren Richtung der einzufügende Punkt liegt. Durch dieses dynamische Wachstumsverhalten, kann der Octree aus Gründen der Einfachheit als einzelner Knoten mit Kantenlänge *min_length* initialisiert werden. Dadurch wird auch der initiale Speicherbedarf minimiert. Eine bessere Wahl in Richtung des Zentrums der endgültigen Punktmenge ist ohne a priori Informationen über die Punktmenge nicht möglich. Im Rahmen der Octreeimplementierung wurden allerdings keine solchen a priori Informationen angenommen. Die dritte Anpassung gilt dem Umstand, dass die vom LSD-SLAM generierten Punkte vereinzelte Ausreißer enthalten können. Da diese bei einer Diskretisierung auf Volumina und anschließender Addition eines Mindestabstandes ganze Bereiche des sonst freien Raums einnähmen, welches zu deutlich schlechteren Ergebnissen in der Pfadplanung führen würde, besitzen Knoten im Octree eine Schranke, ab der sie als existent gewertet werden. Der Knoten wird erst in der Ausgabe berücksichtigt, sobald die Anzahl der eingefügten Punkte den geforderten Grenzwert überschreitet. Zuvor wird an der zugehörigen Position kein Hindernis angenommen.

Optimierungen

Um die Laufzeit und den Speicherbedarf des Octrees weiter zu reduzieren, werden einige Optimierungen vorgenommen. Die erste Optimierung ist, dass die inneren Knoten des Octrees als *voll* markiert werden können. Dies erfolgt, wenn nach dem Einfügen eines Punktes alle Kinder eines inneren Knotens Blätter sind und ein Hindernis enthalten oder wenn alle Kinder als voll markiert wurden. Der Teilbaum von diesem Knoten abwärts kann in diesem Fall aus dem Octree entfernt werden, da er keine zusätzlichen Informationen beisteuert.
Die zweite Optimierung ist das Verwenden von Zeitstempeln. Jedem eingefügten Punkt wird ein Zeitstempel mitgegeben. Jeder Knoten des Octrees erhält diesen Zeitstempel, wenn er oder einer seiner Nachfahren durch Hinzufügen des Punktes verändert wurde. Dies kann beim Auslesen des Octrees in der Pfadplanung genutzt werden, da nur die neu vom LSD-SLAM hinzugefügten Punkte relevant sind.
Als dritte Optimierung wurde der Programmcode so implementiert, dass er für maximal einen Schreiber und beliebig viele Leser *threadsafe* ist, also keine *Race Conditions* durch gleichzeitige Verwendung des Octrees im LSD-SLAM und in der Pfadplanung auftreten können.

Funktionen

In diesem Unterabschnitt werden die von außen sichtbaren Funktionen kurz erläutert.

Punkt einfügen:
Mit dieser Funktion kann ein Punkt in den Octree eingefügt werden. Als Parameter müssen ein Punkt und ein Zeitstempel übergeben werden. Der Punkt wird eingefügt und das durch Einfügen des Punktes entstandene Volumen (Blatt oder voller innerer Knoten) erhält den Zeitstempel. Das Einfügen von mehreren Knoten in einer Funktion ist nicht praktikabel, da wie in Abschnitt 4.3.4 die Zugriffszeiten im Octree als konstant angenommen werden können und jegliche Vorverarbeitung innerhalb einer Punktmenge mehr Zeit benötigt. Diese Funktion vergrößert auch automatisch den Octree, falls ein Punkt außerhalb dessen Volumens hinzugefügt wird.

Volumina abfragen:
Mit dieser Funktion können alle neu entstandenen Volumina nach einem Zeitpunkt abgefragt werden. Die Ausgabe erfolgt als Liste von Vierertupeln, in denen die ersten drei Elemente dem Zentrum und das Vierte der halben Kantenlänge entsprechen. Auf diese Art können die Eckpunkte effizient durch Addition und Subtraktion der vierten Komponente erzeugt werden. Dies ist korrekt, da alle Würfel achsenparallel sind. Als Parameter muss dieser Funktion ein Zeitstempel übergeben werden, ab der Volumina ausgegeben werden sollen. Alle älteren Volumina werden nicht in der Ausgabe aufgelistet. Die Laufzeit dieser Funktion ist linear, da die hierarchische Struktur des Octrees genutzt werden kann, um Zweige mit einem älteren Zeitstempel von der Suche auszuschließen.

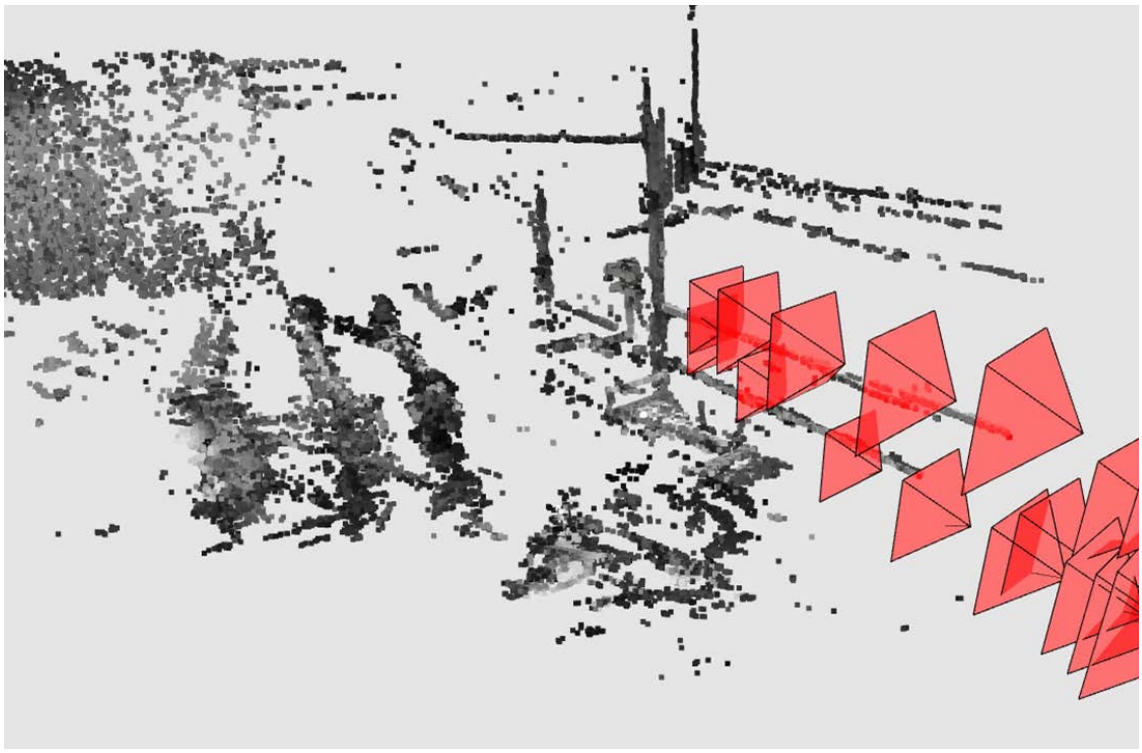
Visualisierung des Octrees

Zur Betrachtung des Octrees dient eine separate Anwendung, an der die vollständigen Daten am Ende der Octree-Konstruktion übergeben werden. Zwar ist eine gleichzeitige Sammlung und Darstellung der Punktmengen zur Laufzeit mit dieser Anwendung nicht möglich, kann jedoch auf gegebene Zeitpunkte eingeschränkt werden. Abgebildet werden die Punkte selbst, die nicht-leeren Knoten des Octree, sowie die Kamera-Posen mit zusätzlicher Skalierung anhand des entsprechenden Keyframes, die in Kapitel 4.3.2 beschrieben werden. Die Navigation durch die Szene erfolgt per Tastatur- und Maussteuerung. Zusätzlich lässt sich die zu betrachtende Octree-Tiefe einstellen. Eine Beispiel-Ausgabe zu der Aufnahme eines Hörsaals ist in der Abbildung 4.13 dargestellt.

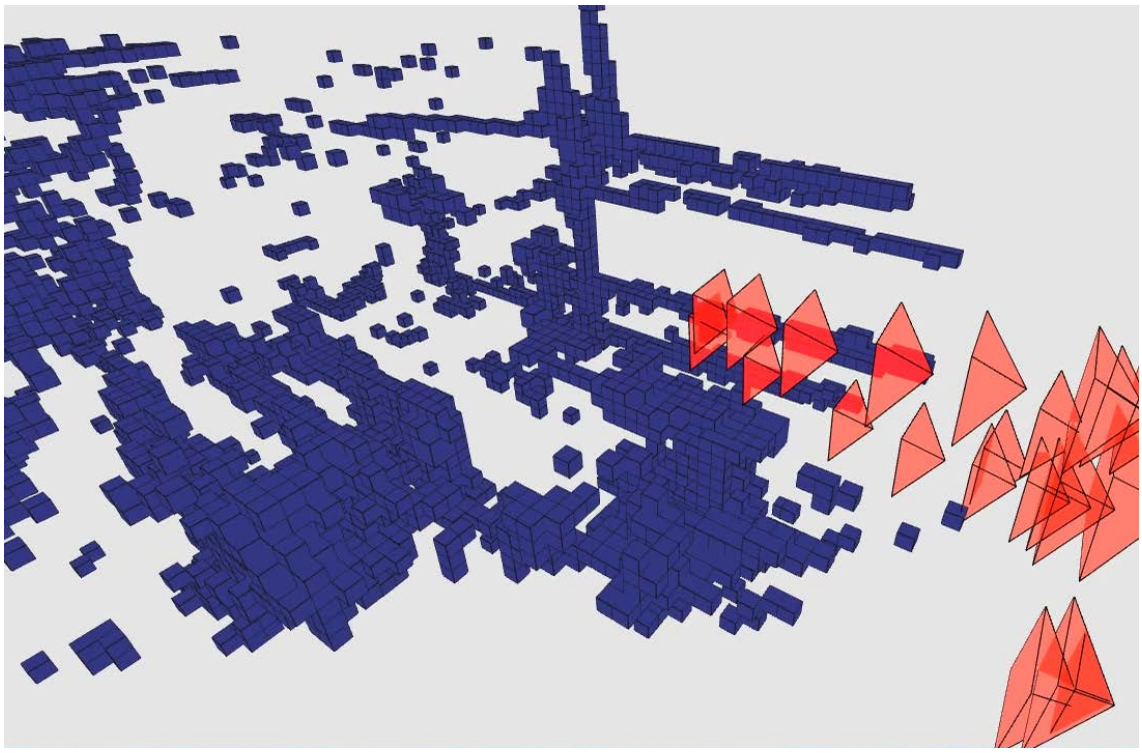
4.4. Diskussion

Gesucht wurde ein robustes Verfahren für die Selbstlokalisation. Weiterhin war gefordert, dass Hindernisse rechtzeitig erkannt werden können, sodass keine Kollisionen mit physischen Objekten entstehen. LSD-SLAM (Kapitel 4.3.2) scheint zunächst aufgrund der im Artikel vorgestellten Ergebnisse ein geeignetes Verfahren für die Selbstlokalisation und gleichzeitige Erkennung von Hindernissen zu sein und kann damit beide Probleme lösen. Beim Tracking (Kapitel 4.3.2) werden anstelle von Merkmalen Keyframes mit geschätzten Tiefenwerten herangezogen, die letztendlich in Punkte umgerechnet und als Hindernisse genutzt werden können. Benötigt wird lediglich eine Kamera mit einer mäßigen Auflösung (in Tests entstanden die besten Ergebnisse mit einer Auflösung von 320×240) und einer Bildrate von etwa 30 FPS. Dies sind Eigenschaften, welche die AR.Drone2-Kamera erfüllt. Diese Voraussetzungen erleichtern zudem das Testen des Verfahrens, insbesondere auch ohne der Verwendung der Drohne. Solange keine weiteren Sensoren verwendet werden, können hierfür zunächst externe Kameras herangezogen werden.
Ein Problem von LSD-SLAM ist, dass die resultierende Skalierung der Posen und Punktmenge durch den Einsatz einer einzigen Kamera nicht der tatsächlichen Skalierung entspricht. Der zusätzliche Einsatz der von der Drohne geschätzten Pose verspricht hierfür Besserung, muss jedoch mit realen Testbedingungen getestet werden. Dazu wird eine funktionsfähige Drohne benötigt, welche die geschätzten Posen bereits liefern kann.
Spiegelnde Oberflächen sind problematisch, da LSD-SLAM ein bildbasiertes Verfahren ist (und fälschlicherweise in Spiegelbildern Tiefen schätzt). Zumindest für Fenster ist dank des vorhandenen Grundrisses eine gewisse Absicherung gegeben, sodass die Drohne sich nicht aus dem Gebäude bewegt. Weiterhin sind erkennbare Strukturen nötig, wodurch aufgrund glatter Oberflächen Löcher an Stellen angenommen werden, an denen sich keine befinden. Der Octree (Kapitel 4.3.4) vereinfacht die resultierenden Daten und durch die Reduktion der Ausreißer werden Punkte zusätzlich entfernt, wobei keine Garantie besteht, dass diese nicht vorhandenen Objekten entsprechen. Unklar ist demnach, ob sich dieses Vorgehen bewährt. Ungelöst ist zudem das Problem, was geschehen soll, wenn LSD-SLAM das Tracking verliert und folglich keine Posen-Schätzungen mehr liefert (Kapitel 4.3.2). Mit der aktuellen Situation muss die Drohne ihren Flug abbrechen und eine Notlandung durchführen. Da Wände wenig Struktur aufweisen können, ist dieser Fall durchaus möglich. Eine denkbare Lösung bestünde darin, das komplette LSD-SLAM zu beenden und neu zu starten. Dazu müssten jedoch die komplette Initialisierungs-Phase wiederholt

und Überlegungen zur Vereinigung der alten und neuen Daten durchgeführt werden. Andererseits könnte die Drohne von der Relokalisierung von LSD-SLAM Gebrauch machen, indem sie beispielsweise eine kurze Strecke zurück fliegt und versucht bekannte Keyframes wiederzufinden. Hierbei könnten die mit der Drohnen-Odometrie geschätzten Posen in der Bahnplanung für eine kurze Zeit als Ersatz verwendet werden.



(a)



(b)

Abbildung 4.13.: Ein Beispiel für eine mit dem Verfahren konstruierte Punktemenge zu den Aufnahmen, die in einem Hörsaal durchgeführt wurden. Abbildung (a) stellt die Punktmenge und Abbildung (b) die nicht leeren Knoten des zugehörigen Octrees einer festen Tiefe dar. Keyframe-Posen sind durch rote Pyramiden angedeutet.

5. Pfadplanung

Dieses Kapitel beschreibt Pfadplanung im Allgemeinen, die Nutzbarmachung von Umweltbedingungen und die Überprüfung der Pfadplanung. Ein Überblick über existierende Methoden und Anwendungsbereiche der Pfadplanung wird in Abschnitt 5.1 beschrieben. Abschnitt 5.2 beschreibt wie Pfade für eine vollständige Erkundung erzeugt werden. Die Speicherung und Adressierungsmöglichkeiten von Aspekten der Umgebung, in der die Erkundung stattfindet, wird in Abschnitt 5.3 erläutert. Die Pfadfindung zwischen zwei Punkten wird in Abschnitt 5.4 näher beschrieben, bevor Abschnitt 5.5 zeigt, wie der ermittelte Pfad verbessert werden kann.
Eine Überprüfung der Pfadfindung findet mithilfe einer Simulation statt, die in Abschnitt 5.6 näher beleuchtet wird. In Abschnitt 5.7 wird eine Zusammenfassung über dieses Kapitel gegeben.

5.1. Einleitung

Pfadplanung ist die Vorausplanung eines Weges zwischen zwei Punkten $\mathbf{p}^{(1)}$ und $\mathbf{p}^{(2)}$. Dieser Pfad ist dazu gedacht, dass sich ein Akteur von Punkt $\mathbf{p}^{(1)}$ zu Punkt $\mathbf{p}^{(2)}$ fortbewegen kann. Ein Pfad π besteht aus mindestens zwei Punkten $\mathbf{p}$.

$$\pi = \left\{\mathbf{p}^{(1)}, \mathbf{p}^{(2)}, \ldots, \mathbf{p}^{(n)} | n \in N\right\} \tag{5.1}$$

Dabei ist in der Regel eine kollisionsfreie Bewegung gewünscht. Anwendungen für Pfadplanung sind in vielen verschiedenen Bereichen zu finden. Beispielsweise werden in der Automobilbranche immer mehr Programme eingesetzt, mit denen Autos auf Autobahnen autonom fahren. Damit das Auto während des Fahrens keine Kollisionen verursacht [67], ist bei allen autonom gesteuerten, zielgerichteten Akteuren eine Pfadplanung notwendig. So existieren auch in der Medizin Anwendungsbereiche für Pfadplanung. Im Beispiel von Kiraly u. a. werden mithilfe von Computer-Tomographie-Bildern (CT-Bilder) Pfade für eine Bronchoskopie berechnet. So kann mithilfe von Scans des Brustkorbs berechnet werden, wie eine Kamera über die Luftröhre in die Lunge eingeführt werden kann [47]. Auch hier sind Kollisionen zu

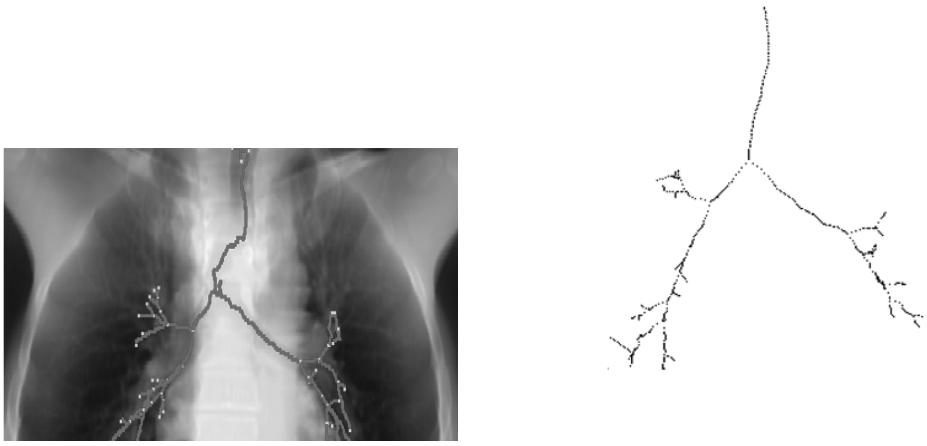
vermeiden, weswegen unter Berücksichtigung der Maße der Kamera ansprechbare Punkte in der Luftröhre mit Knoten versehen werden. Ein Vorteil bei der Berechnung des Pfades für die Bronchoskopie gegenüber der Pfadplanung für autonome Fahrzeuge ist, dass die Kamera keine Wegpunkte mehrfach besuchen muss. Bei autonomen Fahrzeugen hingegen kann dies durchaus der Fall sein, damit alle Winkel der Umwelt einmal gesichtet werden. Aufgrund der fehlenden Notwendigkeit von Mehrfachbesuchen ist im Falle der Bronchoskopie eine Baumstruktur als Repräsentation ausreichend. Die zuvor generierten Knoten werden reduziert und in einer Baumstruktur untereinander verknüpft.

So entsteht ein Baum mit einem Wurzelknoten, der einen Punkt in der oberen Luftröhre repräsentiert und inneren Knoten, welche die Verzweigungen innerhalb der Lunge bis zu den Bronchen darstellen. Die Bronchen sind im Baum in Form von Blattknoten dargestellt. Da zu jedem Knoten ein eindeutiger Pfad besteht, kann dieser generiert werden, indem von dem entsprechenden Knoten bis zum Wurzelknoten der Baum rückwärts durchlaufen wird. Als Projektion auf die CT-Bilder dienen diese Pfade dem behandelnden Bronchoskopisten als Wegweiser.

Ist jedoch die Umwelt nicht bereits im Vorfeld bekannt, so können nicht alle möglichen Pfade vorausberechnet werden und die Pfadplanung muss spontan neue Pfade berechnen. Dies ist der Fall, wenn ein Akteur sich in einer unbekannten Umgebung frei bewegen soll, bspw. ein autonom fahrendes Automobil auf einer Autobahn. Hier wird eine grobe Richtung vorgegeben, z. B. eine Adresse. Dann findet eine Pfadplanung auf Basis von Straßenkarten vom Ursprung bis zum Ziel, bei der die zu nutzenden Straßen ausfindig gemacht werden. Zur Orientierung des Fahrzeugs in der Umwelt können verschiedene Sensoren zum Einsatz kommen, wie beispielsweise GPS-Sensoren, Ultraschallsensoren und Kameras.

Während der Fahrt übernimmt dann eine stetige Pfadplanung, von der die jeweilige Situation immer neu ausgewertet wird. Abbildung 5.2 zeigt die Auswertung einer Möglichkeit zum Überholen als Riskmap. Das autonome Fahrzeug erkennt das vorausfahrende Fahrzeug und stuft es als Risiko ein, wohingegen die freie Fahrspur daneben ohne erhöhtes Risiko befahrbar ist. So kann ein Pfad über Punkte gefunden werden, die kein erhöhtes Risiko widerspiegeln.

Auch Flugdrohnen werden immer häufiger als sich autonom bewegende Vehikel benutzt. Im Gegensatz zu autonom fahrenden Autos können Flugdrohnen Hindernissen auch durch das Verändern der Flughöhe ausweichen. Außerdem existieren für die Routenplanung im Regelfall keine Kartographierungen, mit denen wie bei einem Straßenplan vorausgeplant werden kann. So unterliegt die Umwelt außerhalb



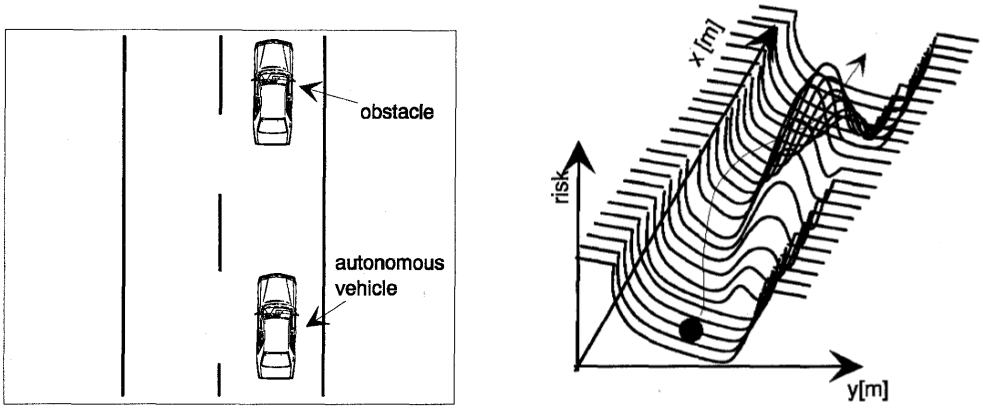
(a) **(b)**

Abbildung 5.1.: Pfadfindung als Unterstützung bei einer Bronchoskopie. Die Pfade werden mithilfe von Computertomographie ermittelt und dienen dem Bronchoskopisten als Wegweiser. (a) Ein Ausgabebild einer Computertomographie der Lunge mit Markierungen für Punkte, an denen ausreichend Platz für die bei der Bronchoskopie genutzten Kamera ist. (b) Die in Bild (a) beschriebene Punktemenge extrahiert und als Baum aufbereitet. Die Bilder sind dem Paper von Kiraly u. a.[47] entnommen.

der Reichweite der Sensoren und Kameras Schätzungen. Alle Hindernisse und Umwelteinflüsse, die in der Wahrnehmungsreichweite auftauchen, müssen so schnell wie möglich interpretiert und bewältigt werden.

5.2. Berechnung eines Erkundungsplans

Eine autonome Erkundung sollte vollständig sein. Das bedeutet in Bezug auf dieses Projekt, dass die Drohne eine Umgebung so abfliegt, dass diese über die Kameras in einer Qualität und Vollständigkeit aufgenommen wird, die eine gute Rekonstruktion ermöglichen. Maßgeblich zur Förderung einer kurzen Flugdauer ist die Häufigkeit mit der Orte angeflogen werden. Mithilfe der Abbildung bereits besuchter Orte in den Speicher ist ein Überblick über bekannte Bereiche der Umgebung möglich. Je nach Menge und Art der gespeicherten Daten lassen sich mehr oder minder ausführliche Aussagen über den einmaligen oder die wiederholten Besuche eines Punktes treffen. Eine simple Überlegung ist die Speicherung eines binären Wertes, der angibt, ob ein Punkt bereits besucht wurde. Dies ist eine schnelle und einfache Methode einen Überblick über den bisherigen Verlauf der Erkundung zu erhalten. In diesem Projekt liegt ein Grundriss vor Beginn der Erkundung vor. Dieser Grundriss kann durch ein



(a) (b)

Abbildung 5.2.: Ein vorausfahrendes Fahrzeug wird von einem autonomen Fahrzeug erkannt und als Risiko bewertet. Die Darstellung der Auswertung geschieht mithilfe einer dreidimensionalen Grafik. (a) Typische Szene auf einer zweispurigen Straße. Ein autonom fahrendes Fahrzeug hinter einem zweiten (nicht notwendigerweise autonomem) Fahrzeug. (Inhalt und Bild in Anlehnung an Reichardt und Shick [67]) (b) Die Riskmap zu der beschriebenen Situation. Das vorausfahrende Auto wird als hohes Risiko bewertet und ist durch einen hohen Ausschlag im Graphen zu erkennen. Die Leitplanken an beiden Seiten sind ebenfalls durch hohe Risikowerte dargestellt (Inhalt und Bild aus dem Artikel von Reichardt und Shick [67]).

Raster von Pixeln dargestellt werden, wobei jeder Pixel für eine zuvor festgelegte Fläche steht, die entweder freien Raum, eine Wand oder ein Hindernis darstellt. Das Ziel der Erkundung ist ein Besuch aller Punkte der Umwelt. Daher liegt der Gedanke einer Einfärbung des Grundrisses nahe, bei der das Notieren eines Besuches mit dem Wechseln der Farbe des jeweiligen Pixel im Grundriss gleichgesetzt wird. Für das Einfärben von Flächen existieren Algorithmen, die auch in der Pfadplanung bereits Anwendung finden. Ein Beispiel dafür ist der FloodFill-Algorithmus, dessen ursprünglicher Nutzungsbereich die Bildverarbeitung ist.

5.2.1. FloodFill

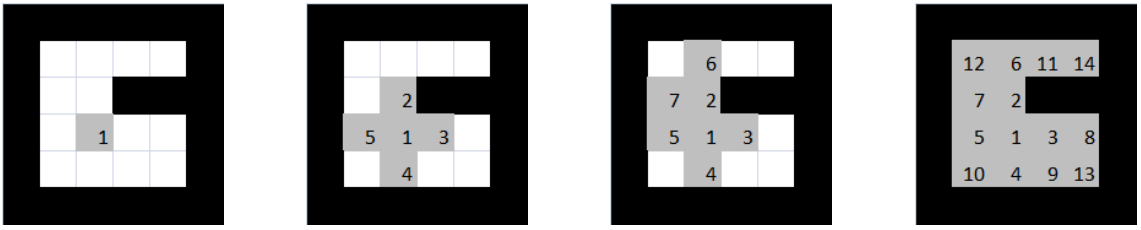
Im Algorithmus werden ausgehend vom Startknoten benachbarte, gleichfarbige Knoten mit einer festgelegten neuen Farbe rekursiv eingefärbt [17]. Dabei prüft der Algorithmus die Nachbarknoten stets in der gleichen Reihenfolge und lässt sich mit dem Prinzip der Tiefen- oder Breitensuche steuern. Der Algorithmus 5.1 beschreibt den Ablauf als Pseudocode.

floodFill (Punkt **p**, Richtung **r**)

1: $NONE \leftarrow 0$
2: $FOLLOW\_UP \leftarrow 1$
3: $PUSHED \leftarrow 2$
4: $retval \leftarrow NONE$
5: **if** **p** bereits besucht **then**
6: **return** NONE
7: **end if**
8: markiere **p** als besucht
9: **if** alle Nachbarn $\mathbf{p}^{(1)}$ bis $\mathbf{p}^{(4)}$ von **p** sind bereits besucht **or** als Hindernis markiert **then**
10: Füge **p** dem Erkundungsplan hinzu
11: **return** PUSHED
12: **end if**
13: **for all** Nachbarn $\mathbf{p}^{(n)}$ von **p** mit $n \in \{1, 2, 3, 4\}$ **do**
14: **if** $\mathbf{p}^{(n)}$ ist noch nicht besucht **and** $\mathbf{p}^{(n)}$ ist kein Hindernis **then**
15: $retval \leftarrow floodFill(\mathbf{p}^{(n)}, \mathbf{p}^{(n)} - \mathbf{p})$
16: **if** $\mathbf{p}^{(n)} - \mathbf{p} \neq r$ **and** $retval > 0$ **then**
17: Füge **p** dem Erkundungsplan hinzu
18: $retval \leftarrow max(retval, PUSHED)$
19: **else if** $retval > 0$ **then**
20: $retval \leftarrow max(retval, FOLLOW\_UP)$
21: **end if**
22: **end if**
23: **end for**
24: **if** der Erkundungsplan soll in beide Richtungen abgeflogen werden **and** $retval = PUSHED$ **then**
25: füge **p** dem Erkundungsplan hinzu
26: **end if**
27: **return** retval

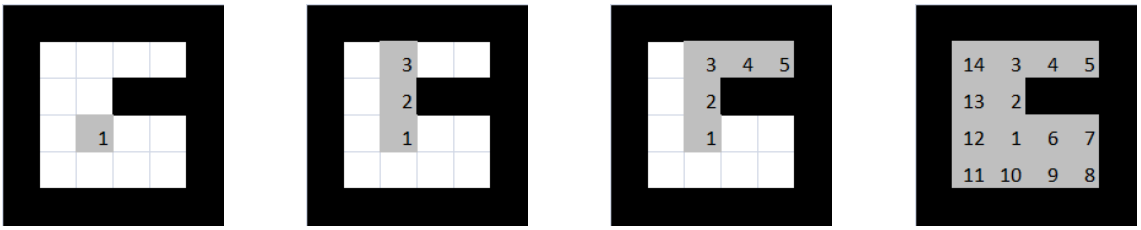
Algorithmus 5.1: Einfacher Algorithmus „floodFill (Punkt **p**, Richtung **r**)“, mit Vektoren als Eingabeparameter, zur Erstellung eines Erkundungsplans auf Basis von FloodFill

Wird das Prinzip der Breitensuche angewendet, so werden nahegelegene Knoten vor weiter entfernten Knoten angesprochen. Mit dem Prinzip der Tiefensuche wird eine Richtung abgearbeitet, bis auf ein Hindernis gestoßen wird, bevor der Algorithmus mit einer weiteren Richtung fortfährt.
Der sich mit der Breitensuche ergebene Ablaufplan, der in Abbildung 5.3d dargestellt wird, ist kreisförmig um den Startpunkt herum angeordnet. Für eine Erkundung, bei der über die Umwelt im Vorfeld keine Informationen vorhanden sind, ist dies eine gute Taktik, da die Umwelt schrittweise aufgebaut wird. So kann die während der Erkundung generierte Karte stetig wachsen.



(a) (b) (c) (d)

Abbildung 5.3.: Erstellen eines Ablaufplans mithilfe von Floodfill nach dem Prinzip der Breitensuche. Nachbarknoten werden zuerst abgearbeitet. Auf diese Weise wird das Umfeld des Startknotens vor weiter entfernten Knoten besucht. (a) Ausgangssituation. (b) Zustand nach dem Markieren der Nachbarknoten des ersten Startknotens. (c) Zustand nach dem Markieren der verbliebenen Nachbarknoten des zweiten Knotens. (d) Der sich ergebene Ablaufplan.

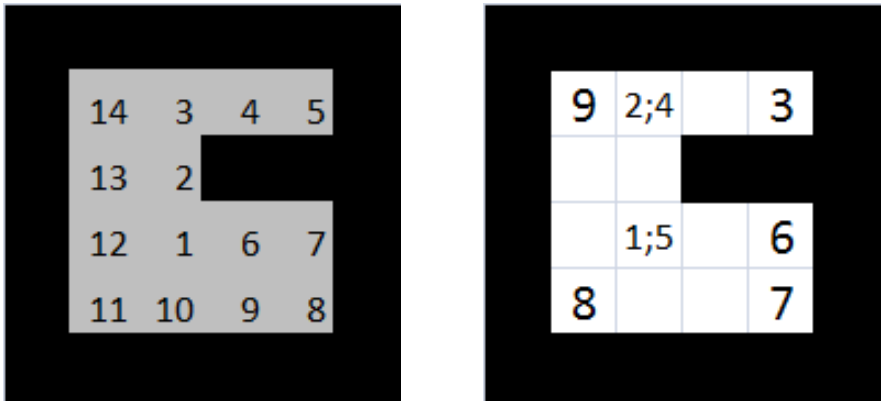


(a) (b) (c) (d)

Abbildung 5.4.: Erstellen eines Ablaufplans mithilfe von Floodfill nach dem Prinzip der Tiefensuche. Auf diese Weise wird ein Pfad mit langen geraden Strecken generiert. In Abbildung (c) wird bei den Knoten 3, 4 und 5 versucht den Knoten jeweils ein Feld weiter oben zu markieren. Da diese jedoch mit Hindernissen belegt sind, wird die zweite Richtung (hier rechts) genutzt. (a) Ausgangssituation. (b) Zustand nach dem Markieren der Knoten bis zum ersten Richtungswechsel. (c) Zustand nach dem Markieren der Knoten bis zum zweiten Richtungswechsel. (d) Der sich ergebene Ablaufplan.

Ist aber der Grundriss vor der Erkundung bekannt, kann dies genutzt werden, um möglichst lange Strecken ohne Richtungswechsel abzufliegen. Dies ist von Vorteil, um möglichst stabile Kamerabilder mit wenig Unschärfe zu erhalten. Abbildung 5.4d zeigt den durch die Tiefensuche generierten Ablaufplan mit langen, geraden Strecken und wenigen Richtungswechseln.

Im beschriebenen Beispiel wurde zur einfacheren Darstellung ausschließlich die zweidimensionale Variante erklärt. Für eine Flugdrohne muss die Funktionsweise um die Richtungen oben und unten erweitert werden.



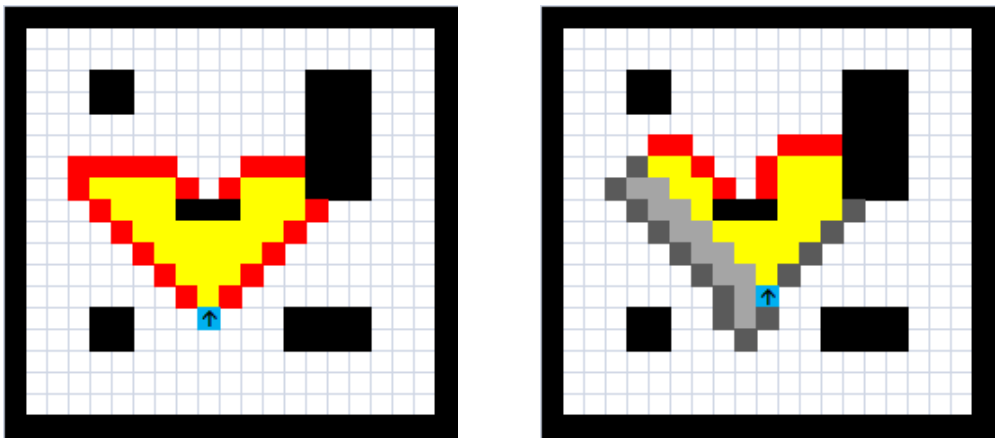
(a) (b)

Abbildung 5.5.: Mit einer Reduktion von Wegpunkten lässt sich die Menge der zu Speichernden Daten reduzieren. (a) Der zuvor in Abbildung 5.4d ausgegebene Plan. Jeder Schritt entspricht einem eigenen Wegpunkt. (a) Die Anzahl an Wegpunkten kann reduziert werden, indem lediglich Knoten, bei denen eine Änderung der Bewegung stattfinden, als Wegpunkte gewählt werden. Knoten, von denen in eine Richtung expandiert wird, sind im Ablaufplan mehrfach vorhanden, da die Drohne diese erneut besucht.

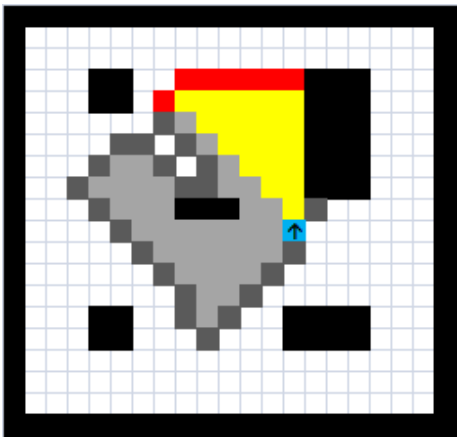
5.2.2. Frontiers

Neben der statischen Berechnung eines Erkundungsplan besteht die Möglichkeit, die Erkundung flexibel mithilfe des bereits Gesehenen zu gestalten. Dafür muss zusätzlich zur Position das Blickfeld der Drohnenkamera bekannt sein. Um dieses zu bestimmen, müssen Informationen wie Blickrichtung, maximal erkannte Entfernung und Blickwinkel bekannt sein. Mit diesen Informationen können alle Knoten, die innerhalb des Blickfelds liegen, markiert werden. So ist während der Erkundung stets bekannt, welche Bereiche bereits gesehen wurden. Diese müssen jedoch nicht notwendigerweise auch besucht worden sein. Unbekannte Knoten, die bisher nicht im Blickfeld der Kamera waren, sind noch zu sichten. Unbekannte Knoten, die einen bereits bekannten Knoten als Nachbarn haben, werden als Grenzen (engl. frontiers) definiert. Diese sind die Knoten, welche als nächstes gesichtet werden müssen, um einen kontinuierlichen Aufbau der gesichteten Umwelt zu gewährleisten [17]

Abbildung 5.6 zeigt den Aufbau eines Blickfelds vor der Drohne. Bereits in Bild (a) ist zu erkennen, dass das Blickfeld um das Hindernis vor der Drohne gelegt ist und die Felder dahinter als Frontiers markiert werden. So ist im Verlauf der Erkundung gewährleistet, dass diese Knoten entweder von einem anderen Standpunkt aus gesehen oder selbst besucht werden. Um die Punkte zu berechnen, die



(a) (b)



(c)

Abbildung 5.6.: Erkundung mithilfe von Markierungen bereits gesehener Punkte. Die Ausrichtung der Drohne ist mit dem Pfeil gekennzeichnet. Das aktuelle Blickfeld der Drohne ist gelb und ältere bereits gesichtete Felder hellgrau eingefärbt. Die roten Felder bezeichnen neue und die dunkelgrauen bereits bestehende Frontiers. (a) Ausgangssituation (b) Zustand nach dem Markieren der Knoten nach Erreichen des nächsten Frontier-Knotens. (c) Zustand nach dem Markieren der Knoten nach mehreren Schritten.

in das aktuelle Sichtfeld fallen, kann der Bresenham-Algorithmus benutzt werden. Der Bresenham-Algorithmus berechnet aus zwei Punkten eine Linie und passt diese auf die entsprechende Rasterung an. Abbildung 5.7 zeigt wie eine Linie zwischen den Punkten $\mathbf{p}^{(a)} = (0,0)^T$ und $\mathbf{p}^{(b)} = (8,2)^T$ auf dem zugrundeliegenden Raster markiert wird.

Die unterschiedlichen Ergebnisse kommen zustande, indem bei der oberen Variante ab $0,5$ auf- und bei der unteren Variante bis $0,5$ abgerundet wird. Wird nun beispielsweise das Feld, welches oberhalb der gerasterten Linie liegt, betrachtet, er-

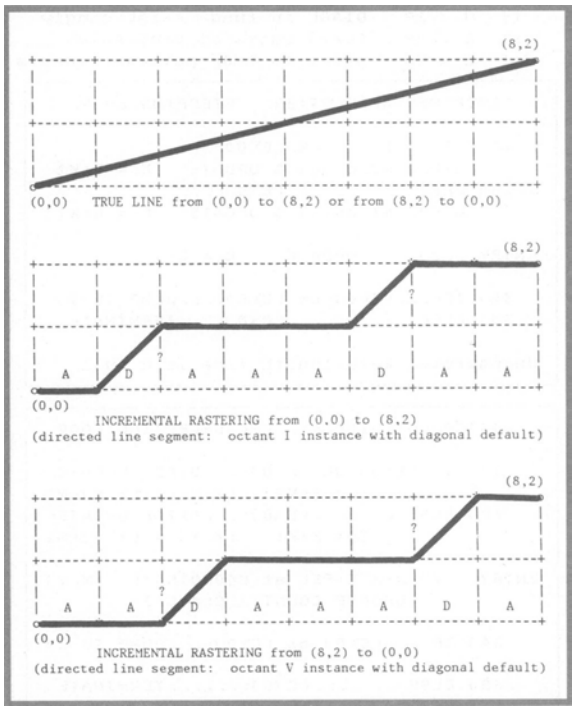


Abbildung 5.7.: Rasterung der Linie von Punkt $\mathbf{p}^{(a)} = (0,0)^T$ bis Punkt $\mathbf{p}^{(b)} = (8,2)^T$[10]. Das Bild ist dem Paper von Bresenham entnommen.

gibt sich eine zusammenhängende Linie von Feldern zwischen den beiden Punkten. Stellen diese Punkte nun den Standpunkt der Kamera und das erwartete Ende der Sichtlinie dar, sind alle dazwischenliegenden Punkte als bereits gesehen zu markieren. Hierbei ist jedoch darauf zu achten, dass mit einem Hindernis die Sichtlinie in der Regel beendet ist. Dieses Verfahren wird für alle Punkte am Ende des Kegels, der den Blickwinkel darstellt, bei jeder Positions- und Blickrichtungsänderung wiederholt. So wächst die Menge der bereits gesehenen Knoten stetig und ergibt einen unmittelbaren Überblick über die bisherige Umwelt. Auch hier ist für die Korrektheit eine Erweiterung in die Höhe notwendig.

5.2.3. Kombination aus FloodFill und Bresenham

Mit in den Abschnitten FloodFill und Frontiers beschriebenen Verfahren lassen sich auf verschiedene Arten Erkundungspläne errechnen. Werden beide Verfahren kombiniert, so kann der strukturierte Flug der Floodfills mit dem Überblick über bereits gesehene Knoten der Frontier-Methode verbunden werden. Zu Beginn wird auf Basis des Grundrisses ein Ablaufplan erzeugt, den die Drohne abfliegt. Während die Drohne diesen Plan abfliegt, wird stetig das Blickfeld überprüft und die darin erfassten Knoten markiert. Nach Beendigung des Abfliegens wird geprüft, ob Knoten existie-

ren, die als Frontier markiert sind. Sollte dies der Fall sein, werden diese als neue Zielpunkte gewählt. Das soll gewährleisten, dass jeder Knoten mindestens einmal gesichtet, aber dennoch möglichst strukturierte Flüge generiert werden.

5.3. Umgebungsrepräsentation

Im vorliegenden Abschnitt werden verschiedene Verfahren vorgestellt, mit deren Hilfe die räumliche Umgebung der Drohne repräsentiert werden kann. Diese Modelle sollen sowohl gegebene Daten, als auch zur Laufzeit erkannte Hindernisse erfassen. Auf dieser Basis sollen im Nachgang sowohl die abstraktere Planung der Erkundung, als auch die konkrete Pfadfindung durchgeführt werden.
In Unterabschnitt 5.3.1 werden zunächst Verfahren vorgestellt, die Polygone für die Repräsentation der Umgebung einsetzen. In Unterabschnitt 5.3.2 wird auf gitter- bzw. voxelbasierte Verfahren eingegangen.

5.3.1. Polygonbasierte Verfahren

Ein möglicher Ansatz der Umgebungsrepräsentation liegt in der Verwendung von Polygonen bzw. Polygonnetzen, welche diejenigen Strukturen modellieren, die sich innerhalb des zu erkundenden Bereichs befinden, wie bspw. Böden, Decken, Wände, oder auch sonstige Hindernisse. Dies wird im Rahmen dieses Projekts bei der Pfadplanung verwendet, um den Grundriss zu repräsentieren, der optional vor Beginn der autonomen Erkundung übergeben werden kann.
Dabei werden einige Annahmen getroffen, welche die Komplexität des Problems eingrenzen und die Modellierung der Räumlichkeiten vereinfachen, jedoch auch gewisse Einschränkungen mit sich bringen. So wird angenommen, dass Wände immer senkrecht zum Fußboden stehen und dessen Normale sich wiederum nur durch einen skalaren Faktor vom Richtungsvektor der Gravitation unterscheidet. Somit werden Konstrukte wie Rampen oder Treppen vom Grundriss ausgeschlossen. Diese können allerdings als Hindernisse betrachtet und somit während der Erkundung trotzdem berücksichtigt werden. Ferner wird davon ausgegangen, dass sich die Drohne nur innerhalb einer Etage bewegt.
Eine einzelne Wand bzw. die Fläche einer Wand wird durch ein Rechteck im Raum modelliert, das immer senkrecht zum Boden steht. Dieses ist gegeben durch ein Tupel $\mathrm{R} := (\mathbf{r}^{(1)}, \mathbf{r}^{(2)})$. Dabei sind $\mathbf{r}^{(1)}, \mathbf{r}^{(2)} \in \mathbb{R}^3$ zwei Eckpunkte des Rechtecks, die sich diagonal gegenüberliegen. Der Grundriss ist damit eine Menge $\mathcal{B} := \{\mathrm{R}_1, \ldots, \mathrm{R}_n\} \subseteq \{(\mathbf{r}^{(1)}, \mathbf{r}^{(2)}) | \mathbf{r}^{(1)}, \mathbf{r}^{(2)} \in \mathbb{R}^3\}$ von Tupeln mit $n \in \mathbb{N}$. In Abbildung 5.8a findet sich eine

exemplarische Darstellung eines solchen Grundrisses. Aus den so gegebenen Daten wird die minimale $h_{min} \in \mathbb{R}$ bzw. maximale Flughöhe $h_{max} \in \mathbb{R}$ abgeschätzt. Als minimale Flughöhe wird das Maximum der Unterkantenhöhen aller Wände verwendet und als maximale Flughöhe analog das Minimum der Oberkantenhöhen (siehe auch Abbildung 5.8b):

$$h_{min} := \max\{\min\{\mathbf{r}_k^{(1)}, \mathbf{r}_k^{(2)}\} | (\mathbf{r}^{(1)}, \mathbf{r}^{(2)}) \in \mathcal{B}\}, \tag{5.2}$$

$$h_{max} := \min\{\max\{\mathbf{r}_k^{(1)}, \mathbf{r}_k^{(2)}\} | (\mathbf{r}^{(1)}, \mathbf{r}^{(2)}) \in \mathcal{B}\}. \tag{5.3}$$

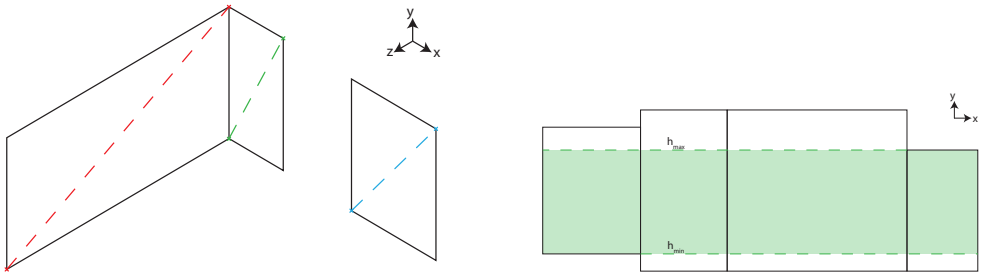
Dabei sei $k \in \{1, 2, 3\}$ der Index derjenigen Vektorkomponente, welche die Flughöhe der Drohne darstellt.
In einem Vorverarbeitungsschritt werden die mit dem Grundriss gelieferten Informationen in die im folgenden Unterabschnitt beschriebene Datenstruktur übertragen. Dadurch können sie auf die gleiche Art genutzt werden, wie Informationen über Hindernisse, die während der Erkundung gewonnen werden.

5.3.2. Voxelbasierte Verfahren

Der Begriff des Voxels bezeichnet ein reguläres Volumen im Raum und ist abgeleitet von „volumetric pixel“. Die grundlegende Idee ist es, den betrachteten Raum mit Hilfe eines regelmäßigen, dreidimensionalen Gitters in einzelne Voxel zu unterteilen. Dies wird bspw. im medizinischen Bereich bei CT- oder MRI-Scans eingesetzt [2]. Ein entsprechender Datensatz könnte zum Beispiel aus $256 \times 256 \times 256$ Voxeln bestehen, denen jeweils ein oder mehrere Werte zugeordnet sind. Im einfachsten Fall wird jedem Voxel ein binärer Wert zugeordnet, der repräsentiert ob der Raum an der betreffenden Stelle leer oder gefüllt ist. Im Folgenden wird ein Voxel V als Tupel betrachtet, für das gilt:

$$\mathrm{V} := (\mathbf{p}, v) \text{ mit } \mathbf{p} \in \mathbb{R}^3, v \in \{0, 1\}. \tag{5.4}$$

Dabei wird $v = 1$ gesetzt, falls das Voxel als gefüllt bzw. belegt gilt; andernfalls wird $v = 0$ gesetzt. Ferner wird der eine Kantenlänge von 1 für jedes Voxel angenommen. Die Speicherung und Verwaltung der Voxel kann auf unterschiedliche Arten mittels verschiedener Datenstrukturen erfolgen. Eine naheliegende Möglichkeit ist die Speicherung in einem dreidimensionalen Array, was den Vorteil konstanter Zugriffszeiten mit sich bringt. Allerdings würde dieses Vorgehen bei entsprechend großen Voxelzahlen schnell dazu führen, dass im Vergleich zum enthaltenen Informationsgehalt



(a) **(b)**

Abbildung 5.8.: Die Teilabbildung (a) zeigt den grundlegenden Aufbau eines Grundrisses bestehend aus Rechtecken, die durch zwei Punkte gegeben sind. Teilabbildung (b) illustriert den abgeschätzten Flugraum in einem Grundriss. Es handelt sich bei h_{min} um die minimal bzw. bei h_{max} um die maximal zulässige Flughöhe.

große Mengen an Speicher belegt werden. Um dieses Problem zu umgehen, wird im Kontext des vorliegenden Projekts ein Octree (siehe Abschnitt 4.3.4) für die Datenhaltung eingesetzt. Der Zugriff erfolgt dabei anhand von Indizes, wie es auch bei der Verwendung eines dreidimensionalen Arrays der Fall wäre. Damit werden für Erkundungsplanung und Pfadfindung unwichtige Details ignoriert.

Bei dem Einsatz des Octrees innerhalb der Pfadplanung wird ausgenutzt, dass ein großer Teil des zu erkundenden Raums oftmals leer ist. Die entsprechenden Voxel müssen nicht explizit abgespeichert werden. Ebenso werden Voxel mit identischen Werten zusammengefasst. Allerdings wird pro Voxel mehr hinterlegt, als nur ein Indikator für die Füllung des korrespondierenden Raums. Voxel können zudem auch von der Erkundungsplanung annotiert werden. So kann beispielsweise festgehalten werden, ob das einem Voxel zugehörige Volumen bereits erkundet bzw. mit der Kamera erfasst worden ist, oder nicht.

Wie bereits erwähnt, werden die in den Voxeln gespeicherten Daten der Einfachheit halber hier allerdings als nur ein einzelner Wert $v \in \{0, 1\}$ betrachtet. Ferner besitzt jeder Knoten genau acht Zeiger auf seine Kindknoten, die allerdings nicht zwingend immer auf ein vorhandenes Kind zeigen müssen, da auch Nullzeiger gestattet sind. Außerdem wird in jedem Knoten gespeichert, ob er voll (2), leer (0) oder teilweise gefüllt (1) ist. Ein Blattknoten kann allerdings nur leer oder voll sein. Handelt es sich um einen inneren Knoten, so gilt er als teilweise gefüllt, wenn mindestens ein Kindknoten voll oder teilweise voll ist, jedoch nicht alle. Sind alle Kindknoten voll, so ist es auch der jeweilige Elternknoten selbst. Nullzeiger werden als leere Kindknoten angesehen. Es existiert genau ein ausgezeichneter Wert, für den Knoten als leer gelten.

Damit wird der Octree als ein Graph bzw. Baum

$$\mathbb{O} := (\mathcal{V}, \mathcal{E}) \tag{5.5}$$

mit der Knotenmenge $\mathcal{V}$ und der Kantenmenge $\mathcal{E}$ betrachtet. Es gilt:

$$\mathcal{V} \subseteq \{(v, o) \mid v \in \{0, 1\}, o \in \{0, 1, 2\}\}, \tag{5.6}$$

$$\mathcal{E} \subseteq \{(\mathrm{A}, \mathrm{B}) \mid \mathrm{A}, \mathrm{B} \in \mathcal{V}\}. \tag{5.7}$$

Dabei wird auf die explizite Speicherung der Positionen der einzelnen Knoten verzichtet. Diese wird mithilfe der Baumtiefe und eines Ortsvektors bestimmt, der für den gesamten Baum gültig ist.
Besitzen alle Kindknoten desselben Elternknotens den gleichen Wert v, so nimmt dieser ebenfalls diesen Wert an und es wird mit $o := 2$ vermerkt, dass der Knoten voll ist. Bei Lesezugriffen auf Blätter unterhalb eines solchen Elternknotens, kann beim Erreichen desselben bereits ein Resultat geliefert werden, ohne den Octree bis zum gewünschten Blatt hinabsteigen zu müssen.
Wie in Unterabschnitt 5.3.1 angedeutet, werden die als Grundriss in Form einer Menge von Rechtecken im Raum übergebenen Daten vor der Erkundung in Voxel überführt. Dazu wird jedes Rechteck schichtenweise mit Hilfe des Algorithmus von Bresenham [10] (siehe Abschnitt 5.2.2) gerastert. Die so gewonnenen Voxel werden jeweils als ein Hindernis in den Octree eingefügt.

5.4. Methoden zur Erzeugung eines Pfades

Nachdem in Abschnitt 5.3 beschrieben wurde, wie die Umgebung der Drohne repräsentiert werden kann, gibt der folgende Abschnitt einen Überblick über Methoden, mit welchen in der Umgebung ein Pfad geplant werden kann.
Zuerst wird ein sehr naives Verfahren vorgestellt, bei welchem der Agent sein Ziel durch einfaches ausprobieren erreicht. Im Anschluss werden zwei Methoden präsentiert, welche Pfade auf Basis von Graphen berechnen und Kollisionen mit Hindernissen bereits bei der Planung vorbeugen.
Ein gewichteter, ungerichteter Graph G ist ein geordnetes Tupel (K, E) mit der Knotenmenge K, welche mit festem Abstand auf einem Raster in der Umgebung verteilt werden und einer Menge an Kanten

$$E = \{(k_1, k_2, g) \mid k_1, k_2 \in K \wedge g \in \mathbb{K}\}, \tag{5.8}$$

wobei das Gewicht g durch die Funktion w beschrieben wird mit

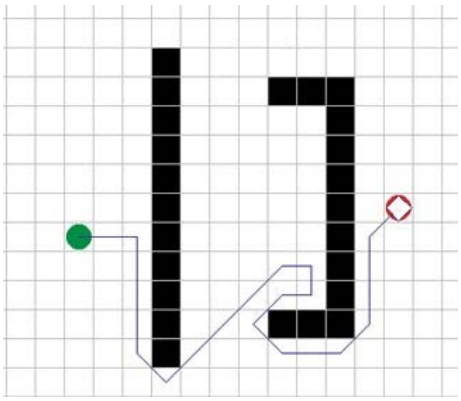
$$w(n_1, n_2) = \begin{cases} 1 & \text{falls } k_1 \text{ und } k_2 \text{ direkte Nachbarn sind und kein Hindernis zwischen ihnen steht} \\ 0 & \text{sonst} \end{cases} . \quad (5.9)$$

5.4.1. Einfache Pfadsuche

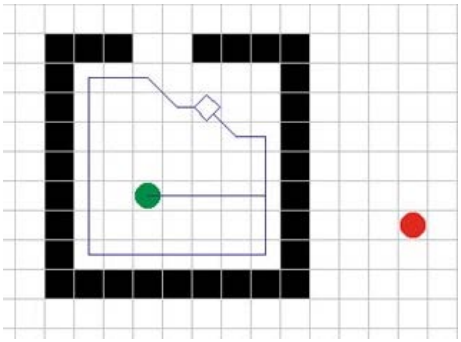
Typisch für die Suche nach einem Pfad ist die Vermeidung von Kollisionen mit Hindernissen in der Umgebung. Die Einheit bewegt sich in die Richtung des Ziels, bis dieses erreicht ist. Wird der Weg durch ein Hindernis versperrt, muss eine andere Richtung gewählt werden, wofür es verschiedene Strategien gibt. Diese werden im Folgenden vorgestellt.
Die erste Strategie ist die Wahl einer zufälligen Richtung um dem Hindernis auszuweichen. Kleine Hindernisse können möglicherweise umgangen werden, indem sich der Agent ein wenig und zufällig in eine andere Richtung bewegt. Sind die Hindernisse jedoch in großer Zahl vorhanden oder konkav, kann es vorkommen, dass der Agent festsitzt und eine lange Zeit benötigt, um einen Weg zu finden, der das Hindernis umläuft.
Ist das Hindernis groß, kann der Verlauf des Hindernisses so lange verfolgt werden, bis dieses umlaufen ist. Stout vergleicht in seinem Artikel [73] das Verfahren mit dem Umgehen einer Wand, z. B. wenn es dunkel ist: Die Hand wird auf die Wand gelegt und dieser so lange gefolgt, bis die Wand zu Ende ist. Das Problem an dieser Technik ist die Suche nach dem Zeitpunkt, ab welchem das Hindernis nicht mehr verfolgt werden soll. Eine mögliche Lösung ist folgende Heuristik: Bewege dich wieder in Richtung des Ziels, wenn diese frei zugänglich ist. Abbildung 5.9 zeigt, wie der Agent das Hindernis umlaufen konnte, indem er dem Verlauf gefolgt ist (Teilabbildung 5.9a). In Teilabbildung 5.9b hingegen konnte das Hindernis nicht umlaufen werden, weil an der entsprechenden freien Stelle eine Bewegung in Richtung des Ziels nicht möglich war. Hier muss eine andere Heuristik gefunden werden.

5.4.2. A*-Algorithmus

Komplexere Pfadsuche ist möglich, indem die Umgebung durch einen Graph abgebildet und die Suche nach einem Weg in dem Graphen durchgeführt wird. Dafür muss zuerst ein Graph aus der Umgebung erstellt werden, wobei das Gewicht g der Kanten abhängig vom gewünschten Optimum der Pfadsuche gewählt werden muss. Denkbare Gewichtungen sind die Distanz zwischen den verbundenen Knoten,



(a) Einfache Pfadfindung mit Kollisionsvermeidung von Hindernissen



(b) Das konkav geformte Hindernis kann nicht überwunden werden

Abbildung 5.9.: Die Abbildung zeigt verschiedene Szenarien. In Teilabbildung (a) kann das Hindernis erfolgreich umlaufen werden In Teilabbildung (b) wird kein Weg aus dem konkav geformten Hindernis gefunden. Die Teilabbildungen sind dem Artikel von Stout entnommen [73].

um den kürzesten Weg zu finden, oder die Dauer, die benötigt wird, um den einen Knoten vom anderen zu erreichen, um die Ausführungszeit zu optimieren.
Der A*-Algorithmus[36] ermittelt den kürzesten Weg zwischen zwei Knoten in einem Graphen. Unter der Annahme, dass die Funktion $f(k)$ die Kosten für den Knoten k berechnet, agiert der Algorithmus wie folgt:

1. Markiere den Knoten s als *offen* und berechne für ihn die Kosten $f(s)$.
2. Wiederhole, solange noch offene Knoten vorhanden sind:
   a) Wähle den offenen Knoten k, für welchen die kleinsten Kosten $f(k)$ berechnet wurden.
   b) Wenn k der Zielknoten ist, markiere k als *abgearbeitet* und terminiere.
   c) Wenn k als abgearbeitet markiert wurde, gehe zu Schritt 2.
   d) Berechne für jeden Nachbarknoten k' von k die Kosten $f(k')$ und markiere ihn als offen, wenn der Knoten k' nicht bereits als abgearbeitet markiert wurde.
   e) Markiere den Knoten k' auch als offen, wenn er bereits vorher abgearbeitet wurde, die neu berechneten Kosten $f(k')$ aber kleiner sind als vorher.
   f) Markiere k als abgearbeitet.

Die Evaluierungsfunktion $f(k)$ berechnet die Kosten für den Pfad zwischen dem Startknoten und dem Knoten k nicht nur aus dem Gewicht der Kanten, sondern

ebenfalls aus einer Heuristik, welche für jeden Knoten abschätzt, wie gut er zum Ziel führt. Sie wird definiert durch

$$f(k) = g(k) + h(k) \tag{5.10}$$

wobei $g(k)$ die Kosten des Pfades vom Startknoten zu k und $h(k)$ die Heuristik für k angibt. Diese kann beispielsweise der euklidische Abstand zwischen den Knoten k und dem Zielknoten sein. Dadurch würde der Algorithmus im zweiten Schritt Knoten präferieren, welche näher am Ziel liegen und sich somit in Richtung des Zielknoten ausbreiten. Solange die heuristische Funktion $h(k)$ nie einen Wert schätzt, der größer als die tatsächliche Distanz ist, wird immer der garantiert kürzeste Pfad ermittelt [73].

5.4.3. Jump Points Search

Das Verfahren der Jump Points Search [35], welches im Folgenden vorgestellt wird, beschränkt sich auf die Pfadsuche in einer Grid-Map und geht deshalb von einem Graph aus, in welchem jeder Knoten mit acht oder weniger Knoten verbunden ist. Die Kosten, um sich zu einem Nachbar zu bewegen, der horizontal oder vertikal an dem aktuellen Knoten liegt, liegen bei 1. Einen diagonalen Nachbarn zu erreichen kostet $\sqrt{2}$. Bewegungen zu einem Knoten, der ein Hindernis auf der Karte darstellt (Hindernis-Knoten), sind nicht erlaubt.
Die Notation $\overrightarrow{d}$ bezeichnet die Bewegung in eine der möglichen Richtungen links, rechts, oben, unten, rechts-oben, rechts-unten, links-oben und links-unten. Ist der Knoten y über k Schritte von dem Knoten x in Richtung $\overrightarrow{d}$ erreichbar, wird dies durch $y = x + k \cdot \overrightarrow{d}$ ausgedrückt. Der Elternknoten von x ist der Knoten, welcher zu x geführt hat und wird durch $\mathrm{p}(x)$ angegeben.
Ein azyklischer Pfad $\pi = \{k_0, k_1, \ldots, k_m\}$ beschreibt die Bewegung von Knoten k_0 bis k_m. Die Kosten eines Pfades π werden als $\mathrm{c}(\pi)$ definiert. Die Funktion $\mathrm{d}(k_i, k_j)$ gibt die Distanz zwischen die Knoten k_i und k_j an.
Abbildung 5.10 zeigt, wie die Suche nach einem optimalen Pfad beschleunigt werden kann, indem nur bestimmte Knoten expandiert werden. Diese werden *Jump Points* genannt.
In Teilabbildung 5.10a soll der optimale Pfad von dem aktuellen Knoten $\mathrm{p}(x)$ zum Zielknoten y gefunden werden. Die Knoten über und unter dem aktuellen Knoten sind Hindernisse und deshalb schwarz markiert. Um von $\mathrm{p}(x)$ nach y zu gelangen, bewegt sich der Algorithmus zuerst nach rechts und der Knoten x wird expandiert.

Anschließend wird Evaluiert, ob die Nachbarn von x möglicherweise eine bessere Lösung für den Pfad darstellen. In der Abbildung sind diese grau markiert. Die Evaluierung wird später genauer betrachtet. Im aktuell betrachteten Fall stellt der Knoten x und die entsprechende Richtung die beste Lösung dar. Der Knoten x wird, wie beim in Abschnitt 5.4.2 gezeigten $A^\star$-Algorithmus, als offen gekennzeichnet und somit später abgearbeitet.
Wird in der entsprechenden Richtung ein Knoten gefunden, welcher einen Jump Point darstellt (wie in Teilabbildung 5.10a der Knoten y), wird ihm der Wert $\mathrm{g}(y) = \mathrm{g}(x) + \mathrm{d}(x, y)$ zugewiesen. Für den Fall, dass kein solcher Knoten in der Richtung gefunden wird, weil beispielsweise der Sprung ein Hindernis trifft, ist die Richtung keine Option und findet keine weitere Beachtung.
Damit nicht alle Nachbarknoten eines aktuellen Knotens expandiert werden, muss die Menge dieser Knoten reduziert werden. Durch Regeln, welche im vorliegendem Abschnitt erläutert werden, können Knoten ausgeschlossen werden, die nicht zur optimalen Lösung führen. Dafür werden die Kosten c von zwei möglichen Pfaden verglichen, wobei beide Pfade mit dem Knoten $\mathrm{p}(x)$ beginnen und mit einem Knoten k enden. Zudem dürfen beide Pfade nur Nachbarknoten von x nutzen und nur einer der Pfade darf über x führen. Die Notation $\pi \setminus u$ beschreibt einen Pfad π, welcher den Knoten u nicht enthält.

Straight Moves Alle Knoten k, welche Nachbarn von x sind, werden ausgeschlossen, wenn für sie

$$\mathrm{c}(\{\mathrm{p}(x), \ldots, k\} \setminus x) \leq \mathrm{c}(\{\mathrm{p}(x), x, k\} \tag{5.11}$$

gilt. Teilabbildung 5.11a zeigt ein Beispiel für diese Regel. Knoten 4 ist der Elternknoten von x, alle Knoten bis auf $k = 5$ werden nicht expandiert.

Diagonal Moves Für diagonale Bewegungen gilt: Alle Knoten n, welche Nachbarn von x sind, werden ausgeschlossen, wenn die Bedingung

$$\mathrm{c}(\{\mathrm{p}(x), \ldots, k\} \setminus x) < \mathrm{c}(\{\mathrm{p}(x), x, k\} \tag{5.12}$$

wahr ist. Teilabbildung 5.11c zeigt diesen Fall beispielhaft für $\mathrm{p}(x) = 6$. Alle Nachbarnknoten von x, bis auf die Knoten $2, 3$ und 5, werden ausgeschlossen.

Wenn der Knoten x keine Hindernisse in seiner direkten Nachbarschaft hat, werden alle Knoten, welche nach den beiden gezeigten Regeln nicht ausgeschlossen sind,

weiter verfolgt und als *natürliche Nachbarknoten* bezeichnet. Andernfalls können nicht alle nicht-natürlichen Nachbarn ausgeschlossen werden und die Evaluierung jedes solchen Nachbarknotens muss erzwungen werden, wenn Folgendes für einen Knoten x gilt:

1. k ist kein natürlicher Nachbarknoten von x
2. $\mathrm{c}(\{\mathrm{p}(x), x, k\} < \mathrm{c}(\{\mathrm{p}(x), \ldots, k\} \setminus x)$

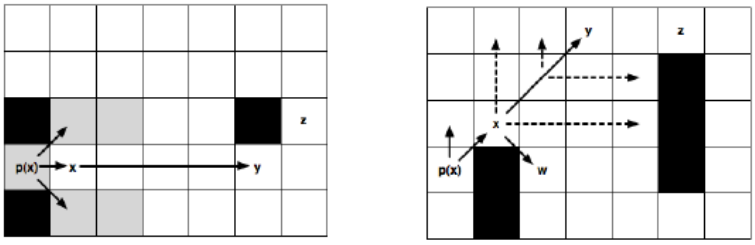
Abbildung 5.11 zeigt ein Beispiel für einen erzwungenen Knoten $k = 3$ für direkte Sprünge (Teilabbildung (b)) und $k = 1$ für diagonale Sprünge (Teilabbildung (d)). Ein Knoten y, der in der gewählten Richtung $\overrightarrow{d}$ von x liegt ist ein Jump Point von x, wenn er k minimiert, so dass $y = x + k \cdot \overrightarrow{d}$ und eine der folgenden Voraussetzungen erfüllt ist:

1. y ist der Zielknoten.
2. y hat einen Nachbarknoten, dessen Evaluierung erzwungen wird.
3. $\overrightarrow{d}$ ist eine diagonale Bewegung und es existiert ein Knoten $z = y + k_i \cdot \overrightarrow{d_i}$, mit $k_i \in \mathbb{N}$ Schritten in Richtung $\overrightarrow{d_i}$, so dass z ein Jump Point von y ist.

Ein Beispiel für die dritte Voraussetzung zeigt Teilabbildung 5.10b, in welcher erst von x zu y gesprungen wird, wobei z ein Jump Point von y und somit y ein Jump Point von x ist.

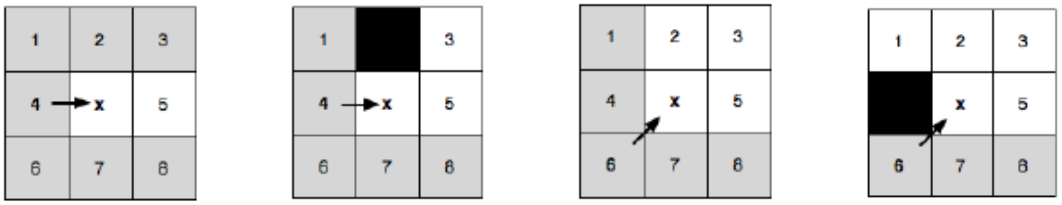
5.5. Simplifizierung des berechneten Pfades

Der berechnete Pfad kann unter Umständen unerwünschte Eigenschaften aufweisen. Ein konkretes Beispiel stellen zu häufige Richtungswechsel dar. Ebenfalls kann die Wahl der Heuristik im Fall des A*-Algorithmus oder damit verwandter Verfahren zu Pfaden führen, die aus der Sicht eines menschlichen Betrachters offensichtlich abgekürzt werden können.
Abbildung 5.12 zeigt exemplarisch einen Pfad in der Ebene, der aufgrund der Flugrichtung eine „gezackte“ Form aufweist. Das ist insbesondere deshalb nicht wünschenswert, da sich die vielen Richtungswechsel negativ auf die Qualität der Selbstlokalisation auswirken können. Dies ist darin begründet, dass die Richtungswechsel zu häufigen Drehungen der Drohne und damit der Kamera führen. Als Nebeneffekt



(a) Direkter Sprung **(b)** Diagonaler Sprung

Abbildung 5.10.: Die Abbildung zeigt den direkten (Teilabbildung (a)) sowie diagonalen (Teilabbildung (b)) Sprung von einem Knoten in Richtung des Ziels. Sie ist dem Paper [35] von Daniel Harabor und Alban Grastien entnommen.



(a) **(b)** **(c)** **(d)**

Abbildung 5.11.: Die Abbildung zeigt das Ausschließen von Nachbarknoten des Knoten x für direkte (Teilabbildung (a)) und diagonale (Teilabbildung (c)) Sprünge. Zudem stellt sie ein Beispiel für das Erzwingen von Nachbarknoten dar, die keine natürlichen Nachbarn sind, jedoch durch ein Hindernis "verdeckt"werden (Teilabbildung (b) und (d)). Sie ist dem Paper[35] von Daniel Harabor und Alban Grastien entnommen.

der Pfadvereinfachung fallen die nachbearbeiteten Pfade in der Regel kürzer, höchstens jedoch so lang wie der ursprüngliche Pfad, aus, da nur diejenigen Wegpunkte passiert werden, deren Anflug auf Grund von Hindernissen unvermeidlich ist.
Ein simples Verfahren zur Pfadvereinfachung – ähnlich dem, wie es von Botea et al. vorgeschlagen wird [9] – besteht darin, nach erfolgreicher Pfadsuche in einem Nachbearbeitungsschritt alle Wegpunkte nacheinander zu durchlaufen und für jeden Wegpunkt zu testen, bis zu welchem Nachfolger der Teilpfad durch eine direkte Verbindung ersetzt werden kann. Vereinfacht werden jedoch keine Teilpfade, die eine Höhenänderung der Drohne beinhalten, da diese weiterhin nur senkrecht auf- oder absteigen soll. Das dient der Vermeidung von Kollisionen mit Hindernissen, die ggf. noch nicht im Sichtfeld der Drohne waren und dementsprechend noch nicht erkannt werden konnten.
Algorithmus 5.2 zeigt das Verfahren in Form von Pseudocode. Dabei seien $\mathbf{p}^{(i)} \in \mathbb{R}^3$ mit $i = 1, \ldots, n$ die Wegpunkte des zuvor errechneten Pfads und $n \in \mathbb{N}$ die Anzahl

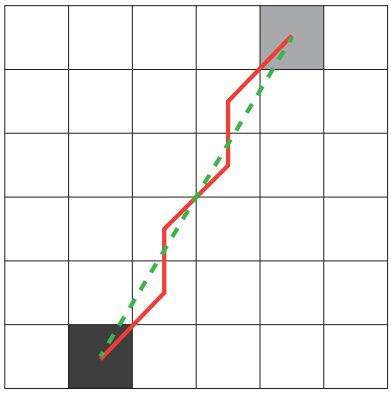


Abbildung 5.12.: Die Abbildung zeigt den vom Pfadfindungsalgorithmus erzeugten, „gezackten“ Pfad (rot, durchgängig) und einen alternativen, geraden Pfad (grün, gestrichelt).

der Wegpunkte. Ferner sei $k \in \{1, 2, 3\}$ der Index derjenigen Vektorkomponente, welche die Flughöhe der Drohne darstellt. Die Ausgabe des Algorithmus ist eine neue Liste von Wegpunkten.

1: Füge $\mathbf{p}^{(1)}$ der Ausgabeliste hinzu
2: **for** $i \leftarrow 1$ **to** $n - 2$ **do**
3: **for** $j \leftarrow i + 2$ **to** n **do**
4: **if** Hindernis zwischen $\mathbf{p}^{(i)}$ und $\mathbf{p}^{(j)}$ **or** $\mathbf{p}_k^{(j)} \neq \mathbf{p}_k^{(j)}$ **then**
5: Füge $\mathbf{p}^{(j-1)}$ der Ausgabeliste hinzu
6: $i \leftarrow j - 1$
7: **break**
8: **else**
9: **if** $j = n$ **then**
10: Füge $\mathbf{p}^{(n)}$ der Ausgabeliste hinzu
11: **return** Ausgabeliste
12: **end if**
13: **end if**
14: **end for**
15: **end for**
16: Füge $\mathbf{p}^{(n)}$ der Ausgabeliste hinzu
17: **return** Ausgabeliste

Algorithmus 5.2: Einfacher Algorithmus zur Pfadsimplifizierung

Der Test, ob zwischen zwei gegeben Wegpunkten Hindernisse liegen, wird dabei unter der Annahme durchgeführt, dass die Zellgröße des Gitters, welches die Hindernisinformationen repräsentiert, nicht kleiner ist als der Radius des kugelförmigen Hüllkörpers der Drohne. Abbildung 5.13 zeigt schematisch, wie dieser Schnitttest durchgeführt wird.

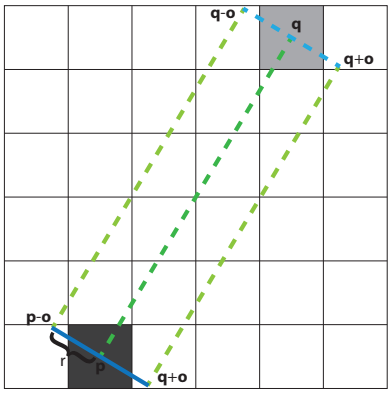


Abbildung 5.13.: Die Abbildung skizziert, welche Strecken bei der Pfadvereinfachung auf Schnitte mit Hindernissen getestet werden.

Es seien $\mathbf{p}, \mathbf{q} \in \mathbb{R}^3$ die beiden Punkte, für die geprüft werden soll, ob sich zwischen ihnen Hindernisse befinden. Es seien außerdem $r \in \mathbb{R}$ der Radius der Drohne und $\mathbf{u} \in \mathbb{R}^3$ mit $||\mathbf{u}||_2 = 1$ derjenige Vektor, welcher die Richtung „oben“, folglich die entgegengesetzte Richtung der Gravitation angibt.
Schnitttests werden nun ausgeführt für die drei Strecken

$$\overline{\mathbf{pq}},\ \overline{(\mathbf{p}+\mathbf{o})(\mathbf{q}+\mathbf{o})},\ \overline{(\mathbf{p}-\mathbf{o})(\mathbf{q}-\mathbf{o})} \tag{5.13}$$

mit

$$\mathbf{o} := r \cdot \frac{(\mathbf{q}-\mathbf{p}) \times \mathbf{u}}{||(\mathbf{q}-\mathbf{p}) \times \mathbf{u}||_2}. \tag{5.14}$$

Wird bei einem der ersten beiden Tests bereits festgestellt, dass die betreffende Strecke Hindernisse schneidet, werden die übrigen, bzw. der übrige Test, nicht mehr ausgeführt, da die potenzielle Abkürzung bereits nicht mehr in Frage kommt.
Die Umgebung wird aus Sicht des Pfandfindungsalgorithmus durch ein regelmäßiges Gitter repräsentiert. Intern wird sie allerdings, wie in Abschnitt 5.3.2 erläutert, mit Hilfe eines Octrees verwaltet, was für die konkrete Umsetzung des Schnitttests zwischen Strecke und Hindernissen ausgenutzt wird.
Da jeder Knoten des Octrees durch einen achsenparallelen Würfel repräsentiert wird, werden hierarchisch auf jeder Ebene Schnitttests zwischen Strecke und Würfel durchgeführt. Wird ein Knoten, der als nicht leer gilt, von der Strecke geschnitten, so werden seine Kindknoten solange weiter getestet, bis feststeht, ob mindestens ein Schnittpunkt existiert. Dies ist der Fall, wenn ein Knoten geschnitten wird, der als voll gilt. Leere Knoten werden übersprungen. Ebenso werden Knoten, die von der Strecke nicht geschnitten werden, nicht weiter überprüft. In Abbildung 5.14 wird dies anhand eines Quadtrees in der Ebene skizziert.

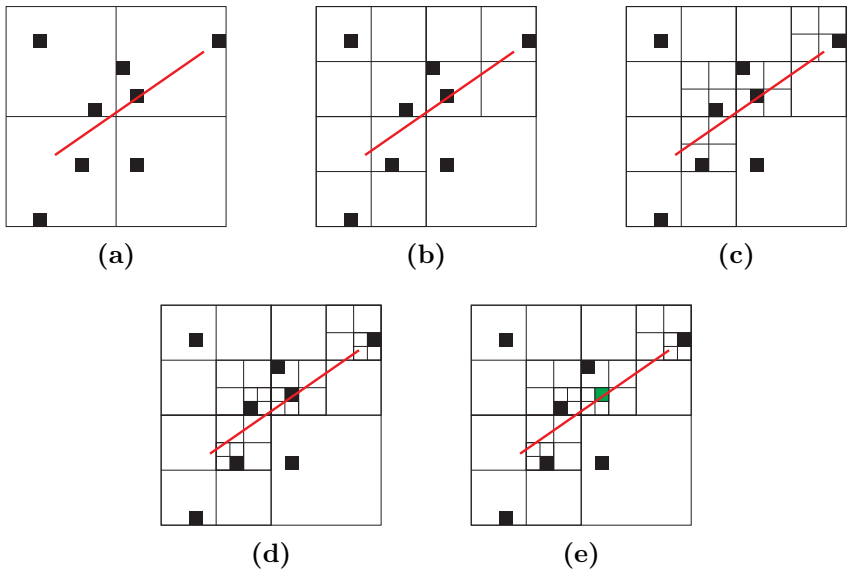


Abbildung 5.14.: Die Abbildung zeigt verschiedene Schritte des Schnitttests einer gegeben Strecke mit den (zu diesem Zeitpunkt) bekannten Hindernissen der Umgebung in einem Quadtree. Betrachtet werden jeweils nur diejenigen Knoten, die von der Strecke geschnitten werden und nicht leer sind, folglich entweder selbst als voll markiert sind, oder mindestens einen als voll markierten Kindknoten besitzen.

Für den Schnitttest einzelner Knoten mit der Strecke besteht die Notwendigkeit, zu prüfen, ob jene Strecke einen gegeben Quader bzw. im vorliegenden Fall einen Würfel schneidet. Dieser Test wird mit der von Akenine-Möller vorgestellten Methode „Line Segment/Box Overlap Test“ durchgeführt, die auf dem *Separating Axis Theorem* beruht [2].

Es seien die Strecke zwischen den beiden Punkten $\mathbf{p}, \mathbf{q} \in \mathbb{R}^3$ und der achsenparallele Quader mit dem Mittelpunkt $\mathbf{m} \in \mathbb{R}^3$ und den Seitenlängen $2\mathbf{h}_1, 2\mathbf{h}_2, 2\mathbf{h}_3 \in \mathbb{R}$ gegeben. Zunächst werden der Quader und die Strecke derart verschoben, dass der Mittelpunkt des Quaders auf dem Ursprung liegt. Anschließend werden der Mittelpunkt der Strecke $\mathbf{c} \in \mathbb{R}^3$ und der Vektor $\mathbf{w} \in \mathbb{R}^3$ bestimmt, sodass $\mathbf{c} \pm \mathbf{w}$ den verschobenen Endpunkten entspricht:

$$\mathbf{c} := \frac{1}{2}(\mathbf{p} + \mathbf{q}) - \mathbf{m} \tag{5.15}$$

$$\mathbf{w} := \frac{1}{2}(\mathbf{p} - \mathbf{q}) \tag{5.16}$$

Insgesamt werden sechs Tests, jeweils bezüglich einer bestimmten Achse, durchgeführt. Die ersten drei Achsen sind $(1,0,0)^T, (0,1,0)^T, (0,0,1)^T$ und die zweiten drei

ergeben sich aus $\mathbf{w} \times (1,0,0)^T, \mathbf{w} \times (0,1,0)^T, \mathbf{w} \times (0,0,1)^T$.

Wenn die Ungleichung

$$|\mathbf{c}_1| > |\mathbf{w}_1| + \mathbf{h}_1. \tag{5.17}$$

erfüllt ist, schneidet die Strecke den Quader nicht. Analog verhält es sich für den zweiten und dritten Test. Für den vierten Test wird zunächst der Quader auf die Achse $\mathbf{w} \times (1,0,0)^T$ projiziert. Die halbe Länge der Projektion beträgt dann $\mathbf{h}_2|\mathbf{w}_3| + \mathbf{h}_3|\mathbf{w}_2|$ und die Projektion von $\mathbf{c}$ ist $\mathbf{c}_2\mathbf{w}_3 - \mathbf{c}_3\mathbf{w}_2$. Damit ergibt sich die folgende Ungleichung:

$$|\mathbf{c}_2\mathbf{w}_3 - \mathbf{c}_3\mathbf{w}_2| > \mathbf{h}_2|\mathbf{w}_3| + \mathbf{h}_3|\mathbf{w}_2| \tag{5.18}$$

Ist diese Ungleichung erfüllt, schneidet die Strecke den Quader nicht. Auch hier sind die folgenden beiden Tests analog. Ist keine der insgesamt sechs Ungleichungen erfüllt, schneidet die Strecke den Quader. In welchen Punkten der Schnitt stattfindet, ergibt sich aus dem Verfahren zwar nicht, ist in diesem konkreten Fall jedoch nicht von Belang, da lediglich die Information benötigt wird, ob ein Schnitt vorhanden ist. Algorithmus 5.3 zeigt den Ablauf des Tests in Form von Pseudocode.

```
1: v ← (w_1, w_2, w_3)^T
2: if |c_1| > v_1 + h_1 then
3:    return false
4: end if
5: if |c_2| > v_2 + h_2 then
6:    return false
7: end if
8: if |c_3| > v_3 + h_3 then
9:    return false
10: end if
11: if |c_2w_3 − c_3w_2| > h_2v_3 + h_3v_2 then
12:    return false
13: end if
14: if |c_1w_3 − c_3w_1| > h_1v_3 + h_3v_1 then
15:    return false
16: end if
17: if |c_1w_2 − c_2w_1| > h_1v_2 + h_2v_1 then
18:    return false
19: end if
20: return true
```

Algorithmus 5.3: Schnitttest zwischen einem achsenparallelen Quader und einer Strecke

5.6. Simulation

Simulationen werden in vielen Wissenschaftlichen Bereichen eingesetzt, um beispielsweise die Formation und Entwicklung von Sternen in der Galaxie oder den Ausbruch von Kriegen zu simulieren [37]. Das Ziel ist, von einem Versuch an einem Modell auf das Verhalten in der realen Umgebung zu schließen. Der für dieses Projekt entwickelte Simulator stellt als Modell eine Drohne in einem Zimmer mit Gegenständen wie Tischen und Stühlen bereit, um das Flugverhalten der Drohne zu testen.

5.6.1. Aufgaben

Die Pfadplanung ist als eigenständige Komponente entworfen und daher unabhängig von der Selbstlokalisation und Hinderniserkennung (siehe Kapitel 4) sowie der Steuerung der Drohne (siehe Kapitel 7). Um Fehler möglichst genau aufzudecken, sind Tests der einzelnen Komponenten hilfreich. Durch die Simulation wird die Gefahr für Menschen und Gegenstände gesenkt, welche bei inkorrektem Flugverhalten von der Drohne ausgehen würde.
Damit die Pfadplanung als eigene Komponente getestet werden kann, werden die Abhängigen Komponenten der Selbstlokalisation, Hinderniserkennung und die Drohne als solche simuliert. Der Simulator nimmt anstelle der Drohne Befehle (siehe Abschnitt 5.6.2) zur Steuerung entgegen und liefert auf Anfrage die aktuelle Position und gefundene Hindernisse.
Weiter kann die Simulation durch grafische Ausgaben, welche in der Anwendung angestoßen werden können, weitere Informationen visualisieren. Beispielsweise kann die Einheit zur Berechnung eines Erkundungspfades darstellen, welche Bereiche der Umgebung bereits abgeflogen wurden.
Durch die Simulation kann ein Fehlverhalten innerhalb der Pfadplanung, welche die Drohne in der Realität z. B. gegen Hindernisse fliegen lassen würde, aufgedeckt werden, ohne dass die Drohne Schaden nimmt.

5.6.2. Schnittstelle zur Anwendung

Die Anbindung des Simulators erfolgt über eine Client-Server-Architektur, wobei der Simulator die Rolle des Servers übernimmt. Der Befehlssatz, welcher von der Simulation interpretiert werden kann, gleicht dem der Drohne. Zusätzlich kann die Simulation auf Befehl Objekte im dreidimensionalen Raum erstellen, wodurch eine visuelle Testmöglichkeit geschaffen wird. Diese kann bspw. genutzt werden, um Hindernisse, welche der Anwendung bekannt sind, testweise anzuzeigen.

Tabelle 5.1 zeigt alle Befehle, welche von der Anwendung (Client) an die Simulation (Server) gesendet und von dieser empfangen werden können. Einige Befehle enthalten Parameter (z. B. eine Koordinate im Raum), welche durch Doppelpunkte getrennt an den Befehl selbst konkateniert werden.

5.6.3. Benutzerschnittstelle

Das Ergebnis der Simulation muss vom Benutzer ausgewertet werden. Abbildung 5.15 zeigt die Benutzerschnittstelle des Simulators, welcher eine Drohne sowie deren Umgebung modelliert. Dem Benutzer werden in der oberen linken Ecke des Fensters Informationen über die Verbindungen zur Anwendung angezeigt. Diesen ist in dieser beispielhaften Abbildung zu entnehmen, dass die Steuerung (*Agent*) und die Simulation der Selbstlokalisation und Hinderniserkennung (*Tracking*) verbunden sind. Die Software zur Ausgabe von Test-Objekten innerhalb der Simulation (*Debug*) ist nicht verbunden.
In Abbildung 5.16 ist eine Simulation dargestellt, in welcher sich die modellierte Drohne falsch verhalten hat und mit einer Wand kollidiert ist, welche sich nach der Kollision rot eingefärbt hat. Zudem wird die Anzahl der Kollisionen im aktuellen Durchlauf rechts oben im Fenster angezeigt.
Die Ausgabe von Hilfsobjekten im Simulator zeigt Abbildung 5.17. Bei der Berech-

Sender	Befehl	Beschreibung
Client	move:x:y:z	Die simulierte Drohne bewegt sich zu der Koordinate (x, y, z).
Client	turn:x:y:z	Die simulierte Drohne dreht sich, so dass die Kamera in Richtung der Koordinate (x, y, z) ausgerichtet ist.
Client	position	Die Anwendung möchte die Position der Drohne erfahren.
Client	stop	Die Simulation wird gestoppt.
Client	draw:x:y:z:c	Zeichnet ein Test-Objekt mit der Farbe c an Koordinate (x, y, z).
Client	undraw:x:y:z	Entfernt ein gezeichnetes Test-Objekt an der Koordinate (x, y, z).
Server	obstacle:x:y:z	Die Simulation hat ein Hindernis an der Koordinate (x, y, z) gefunden.
Server	position:x:y:z	Die Simulation übermittelt die aktuelle Position (x, y, z) an die Anwendung.

Tabelle 5.1.: Befehle die an die Simulation (Server) gesendet und von der Anwendung (Client) empfangen werden können.

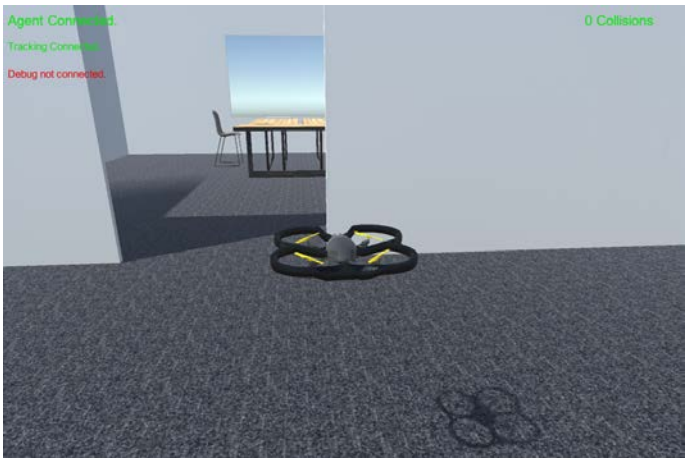


Abbildung 5.15.: Die Abbildung zeigt die Ansicht des Simualtors, der eine Drohne in einem Zimmer modelliert.

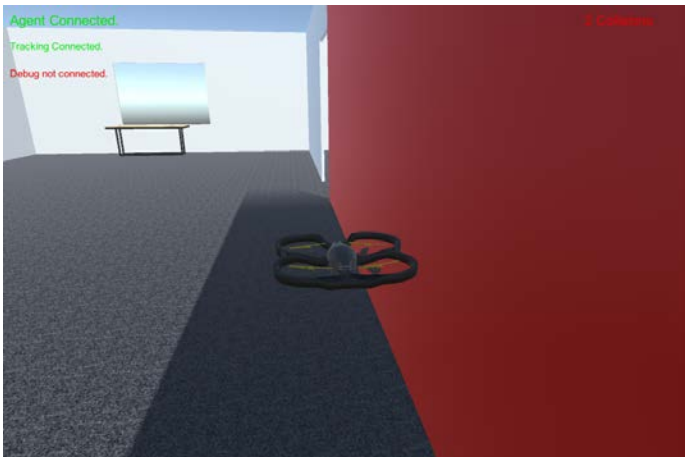


Abbildung 5.16.: Die Abbildung zeigt den Simulator, nachdem die modellierte Drohne mit einer Wand kollidiert ist.

nung eines Erkundungsplanes werden Koordinaten im Raum berechnet, welche von der Drohne abgeflogen werden, um die Umgebung möglichst vollständig zu erkunden (vgl. Kapitel 5.2). Diese werden im Simulator, falls die Ausgabe-Komponente verbunden ist, violett dargestellt. Zusätzlich wird der zuletzt berechnete Pfad angezeigt.

5.7. Zusammenfassung

Die Pfadplanung stellt in diesem Projekt einen Bereich mit großem Einfluss dar. Bevor die Drohne abheben kann, wie in Abschnitt 5.2 beschrieben, wird ein Plan vorausberechnet. Durch diesen Plan ist ein grober Ablauf gegeben, mit dem möglichst viele Umweltdetails über die Kamera der Drohne aufgenommen werden sollen.

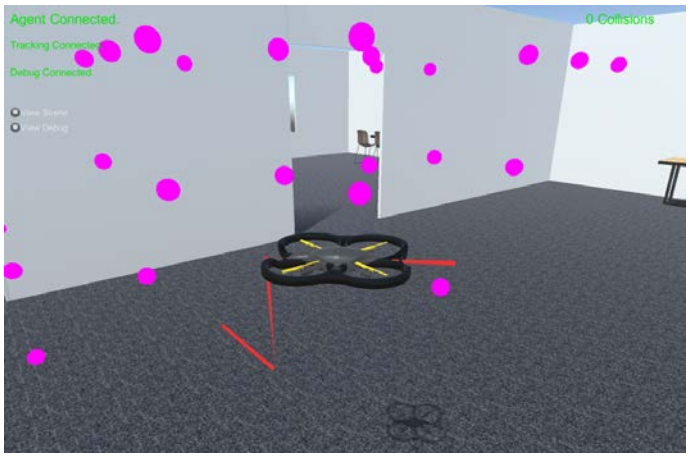


Abbildung 5.17.: Hilfsobjekte, wie violette Kugeln, helfen bei der Entwicklung der Pfadplanung und können im Simulator angezeigt werden.

Die Vollständigkeit gesichteter Volumen soll durch die Frontier-Erkundungsmethode gesichert werden (siehe Abschnitt 5.2). Während der Erkundung werden einzelne Punkte als zwischenzeitliche Ziele angegeben. Diese Ziele werden mithilfe der direkten Pfadplanung, die in Abschnitt 5.4 beschrieben wird, angesteuert. So wird der zuvor berechnete Erkundungsplan schrittweise abgeflogen. Um eine schnelle Berechnung der jeweiligen Pfade zu gewährleisten, wird im Projekt auf den A*-Algorithmus zurückgegriffen.
Die durch den Grundriss eingelesenen Umgebungsdaten, sowie die im Flug erkannten Hindernisse, werden in einem Octree zur Einsicht und Pfadplanung abgelegt. Diese Speichermethode wird in Abschnitt 5.3 näher erläutert. Mithilfe des Octrees ist ein schneller und sicherer Zugriff auf bereits erkannte Hindernisse möglich. So kann die Pfadplanung angepasst werden, um der Drohne einen kollisionsfreien Flug zu ermöglichen.
Da jedoch Tests der Pfadplanung während der frühen Phasen des Projekts nicht mit der Drohne möglich sind, musste eine Alternativmöglichkeit entworfen werden. Die in Abschnitt 5.6 beschriebene Simulationsumgebung stellt den Flug einer Drohne dar. Mithilfe dieses Simulators ist während früher Entwicklungsphasen die graphische Überprüfung von Algorithmen möglich.

6. Rekonstruktion der Umgebung

In diesem Kapitel wird die dreidimensionale Rekonstruktion der Umgebung auf Grundlage der aufgezeichneten Tiefendaten erläutert. Dabei liegt der Fokus, im Gegensatz zu den in Abschnitt 4.3 vorgestellten Verfahren, auf der Generierung einer möglichst dichten dreidimensionalen Umgebungskarte, welche die Szene detailgetreu visuell wiedergibt. Dazu wird in Abschnitt 6.1 das verwendete KinectFusion-Verfahren und zugehörige Erweiterungen ausführlich erklärt. In Abschnitt 6.2 wird die Rekonstruktion auf Basis von Punktwolken erläutert. Dafür werden ICP(iterative closest points)-Algorithmen vorgestellt, welche ebenfalls in Abschnitt 6.1 Verwendung finden. Des Weiteren wird in Abschnitt 6.2.4 ein Verfahren zur Gewinnung eines Oberflächenmodells aus Punktwolken dargestellt.

6.1. Mapping und Tracking

In diesem Abschnitt wird die Rekonstruktion mithilfe des KinectFusion-Verfahren erläutert. KinectFusion ermöglicht echtzeitfähiges hochauflösendes Mapping und Tracking einer statischen Umgebung mithilfe einer Tiefenkamera und einer modernen handelsüblichen Grafikkarte. Dieses Kapitel basiert dabei größtenteils auf den Veröffentlichungen von Newcombe et al. [58] und Izadi et al. [43], in denen das Verfahren von den Entwicklern vorgestellt wurde.
Die bereitgestellten Tiefendaten werden dabei in ein globales implizites Oberflächenmodel der bereits beobachteten Szene integriert. Die aktuelle Position der Kamera wird dabei verfolgt, indem das jeweils aktuelle Tiefenbild mithilfe eines ICP-Algorithmus (siehe Abschnitt 6.2.3) in das globale Modell eingepasst wird. Um die Performanz des Verfahrens wesentlich zu erhöhen, werden zentrale Berechnungen parallelisiert und auf der GPU durchgeführt.

Komponenten

KinectFusion besteht im Wesentlichen aus folgenden vier Komponenten, die, wie in Abbildung 6.1 grob verdeutlicht, zusammen arbeiten:

Surface measurement In diesem Vorverarbeitungsschritt werden die Tiefendaten gefiltert und aus diesen Bildpyramiden mit verschiedenen Auflösungen berechnet. Aus den einzelnen Bildpunkten werden im Anschluss entsprechende 3D-Vertices und Normalenvektoren im Kamerakoordinatensystem gewonnen. Diese Komponente wird im Abschnitt 6.1.1 erörtert.

Surface prediction Im Gegensatz zur Posenschätzung mithilfe eines „frame-to-frame"-Ansatzes, wird das aktuelle Tiefenbild in das globale Modell eingepasst. Dazu wird ein Raycasting im impliziten Oberflächenmodell durchgeführt, um so die Oberfläche und damit auch die Tiefe im Bezug zur aktuellen Kameraposition vorherzusagen. Dazu wird in Abschnitt 6.1.2 die Szenenrepräsentation erklärt. Im Anschluss wird in Abschnitt 6.1.3 auf die Tiefenvorhersage durch Raycasting eingegangen.

Sensor pose estimation Für die Schätzung der aktuellen Kamerapose wird ein multiskalarer ICP-Algorithmus (vergleiche [5]) verwendet. Dieser richtet die aktuell gemessenen Tiefendaten an den im vorherigen Schritt aus dem Modell berechneten Oberflächen aus. Dafür werden die gesamten Tiefendaten des Sensors verwendet und insbesondere keine Merkmale extrahiert. Diese Posenschätzung wird in Abschnitt 6.1.4 näher erläutert.

Surface reconstruction update In diesem Teilschritt werden die neuen Tiefendaten anhand der geschätzten Kamerapose in das globale Koordinatensystem überführt und dann in das bisher berechnete Umgebungsmodell integriert. Dazu wird anstatt von Polygonnetzen oder Punktwolken eine volumetrische Repräsentation der Oberflächen als Distanzfeld, genauer eine diskretisierte TSDF (truncated signed distance function), verwendet. Diese Form der Repräsentation vereinfacht die Fusion mehrerer Teilmodelle, welche in Abschnitt 6.1.5 beschrieben ist, wesentlich.

Die Größe der Szene wird beim KinectFusion-Verfahren durch die Größe des Grafikspeicher limitiert. Dabei wird beispielsweise für einen 4.5 m×4.5 m×4.5 m großen Würfel in einer angemessenen Auflösung von 768×768×768 1728 MB Grafikspeicher benötigt. Um KinectFusion für größere Räume verwenden zu können, muss das grundlegende Verfahren erweitert werden. In Abschnitt 6.1.7 werden dazu die verwendeten Erweiterungen näher beleuchtet. In Abbildung 6.15 wird der Ablauf des erweiterten KinectFusion-Verfahrens in Form eines Aktivitätsdiagramms verdeutlicht.

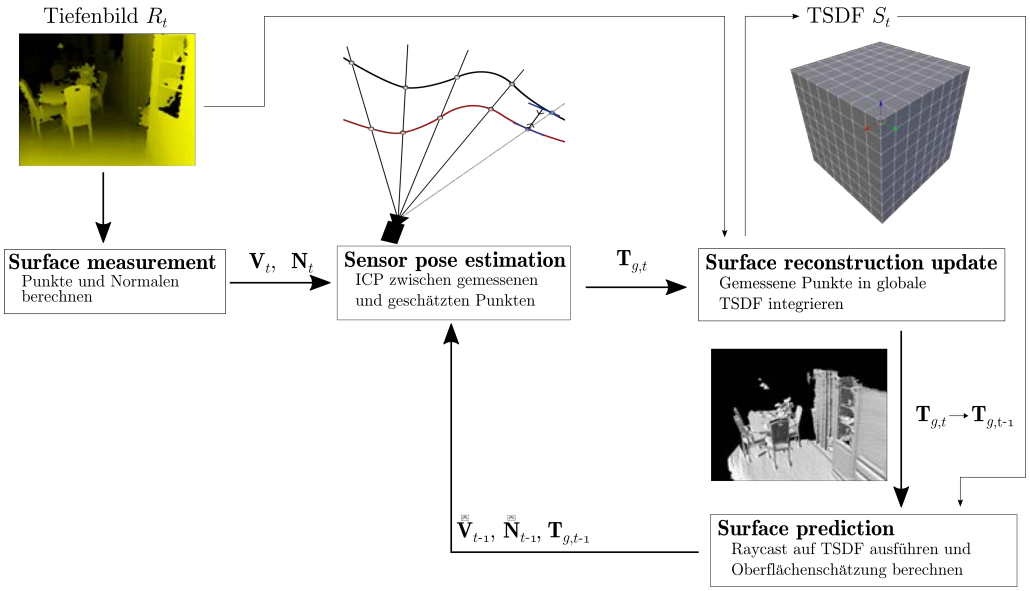


Abbildung 6.1.: Komponenten und Verarbeitungspipeline von KinectFusion

6.1.1. Vorverarbeitung der Tiefendaten

Das unbearbeitete Tiefenbild, welches die Kamera zum Zeitpunkt t liefert, wird mit $R_t(\mathbf{u}) \in \mathbb{R}$ bezeichnet. Dabei ist $\mathbf{u} = (u, v)^T \in \mathcal{U} \subset \mathbb{N}^2$ ein Pixel des Bildbereiches $\mathcal{U}$. Jeder Pixel $\mathbf{u}$ entspricht dabei der Messung eines Punktes $\mathbf{p}_t = R_t(\mathbf{u})\mathbf{K}^{-1}\dot{\mathbf{u}}$, welcher auf der Oberfläche eines Objektes im Raum liegt. Dabei ist $\mathbf{K}$ die intrinsische Kalibrierungsmatrix der Kamera, welche zuvor berechnet wurde. Umgekehrt projiziert $\pi(\mathbf{p}) = \mathbf{u}$ einen beliebigen Punkt $\mathbf{p} = (x, y, z)^T \in \mathbb{R}^3$ auf die Bildebene. Die Punktnotation wird in diesem Abschnitt für homogene Vektoren $\dot{\mathbf{u}} = (\mathbf{u}^T|1)^T$ verwendet. Die konkret verwendete Tiefenkamera (siehe Abschnitt 3.3) liefert Tiefenbilder in der Auflösung 640×480 mit einer Bildrate von 30 Hz und gibt die Tiefe in Metern an. Um Messrauschen zu unterdrücken, dabei jedoch Kanten zu erhalten, wird ein bilateraler Filter (vergleiche [75]) verwendet. Dieser arbeitet nach dem Prinzip der sogenannten lokalen Operatoren. Bei diesen wird an jeder Position des Tiefenbildes $R_t(\mathbf{u})$ ein neuer Bildpunkt berechnet. Dazu wird eine Maske um den aktuellen Pixel $\mathbf{u}$ gelegt, die einen definierten Bereich im Umfeld $\mathcal{W}$ abdeckt. Durch die Maske wird eine Gewichtung der einzelnen Pixel in diesem Umfeld definiert. Die von der Maske überdeckten Pixel werden mit ihrer jeweiligen Gewichtung multipliziert und aufsummiert. So wird ein neuer Intensitätswert an dieser Stelle berechnet, der auch Filterantwort genannt wird. Der lokale Operator wird ferner über die Maske, auch Kernel genannt, definiert. Der bilaterale Filter verwendet dabei einen speziellen Kernel, welcher beispielsweise in Abbildung 6.2 dargestellt ist. Dabei werden Pixel in

der Fensterumgebung nicht nur aufgrund der räumlichen Entfernung zum Filterkern (Domain-Gewichte) gewichtet, sondern auch über die Entfernung ihrer Intensität zur Intensität des Filterkerns (Range-Gewichte). Das so gefilterte Tiefenbild ist definiert als

$$D_t(\mathbf{u}) = \frac{1}{N_\mathbf{u}} \sum_{\mathbf{q} \in \mathcal{W}} W_s(\mathbf{u}, \mathbf{q}) W_r(\mathbf{u}, \mathbf{q}) R_t(\mathbf{q}) \tag{6.1}$$

mit den Domain-Gewichten

$$W_s(\mathbf{u}, \mathbf{q}) = \mathcal{N}_{\sigma_s}(||\mathbf{u} - \mathbf{q}||_2) \tag{6.2}$$

und den Range-Gewichten

$$W_r(\mathbf{u}, \mathbf{q}) = \mathcal{N}_{\sigma_r}(||R_t(\mathbf{u}) - R_t(\mathbf{q})||_2). \tag{6.3}$$

Dabei ist $\mathcal{N}_\sigma(x) = \exp(\frac{-x^2}{\sigma^2})$ die Dichtefunktion der Normalverteilung und $N_\mathbf{u} = \sum_{\mathbf{q} \in \mathcal{U}} W_s(\mathbf{u}, \mathbf{q}) W_r(\mathbf{u}, \mathbf{q})$ dient der Normalisierung des Gesamtgewichtes auf Eins. Durch die Verwendung der Range-Gewichte bleiben trotz starker Rauschunterdrückung Kanten sehr gut erhalten.

Die Intensitäten der gefilterten Tiefenbilder werden anschließend von der Bildebene in das dreidimensionale Kamerakoordinatensystem projiziert. Die resultierende Menge von Vertices

$$\mathbf{V}_t = \{\mathbf{v}_t(\mathbf{u}) | \mathbf{u} \in \mathcal{U}\} \tag{6.4}$$

mit

$$\mathbf{v}_t(\mathbf{u}) = D_t(\mathbf{u}) \mathbf{K}^{-1} \dot{\mathbf{u}} \tag{6.5}$$

wird dabei für jeden Pixel parallel auf der GPU berechnet. Die zugehörigen Normalenvektoren

$$\mathbf{N}_t = \{\mathbf{n}_t(\mathbf{u}) | \mathbf{u} \in \mathcal{U}\} \tag{6.6}$$

werden mithilfe des Kreuzproduktes benachbarter Punkte ebenfalls parallel berechnet und mit $l[\mathbf{x}] = \mathbf{x}/||\mathbf{x}||_2$ auf Einheitslänge normalisiert:

$$\mathbf{n}_t(\mathbf{u}) = \mathbf{n}_t(u, v) = l[(\mathbf{v}_t(u+1, v) - \mathbf{v}_t(u, v)) \times (\mathbf{v}_t(u, v+1) - \mathbf{v}_t(u, v))]. \tag{6.7}$$

Außerdem wird eine Bildmaske $m_t(\mathbf{u})$ bestimmt, welche den Wert 0 für eine fehlende und den Wert 1 für eine vorhandene Tiefenmessung am Pixel $\mathbf{u}$ annimmt. Diese Maske wird sowohl bei der Posenschätzung als auch bei der Fusion mit der Szene verwendet, um Pixel mit fehlender Messung auszuschließen.

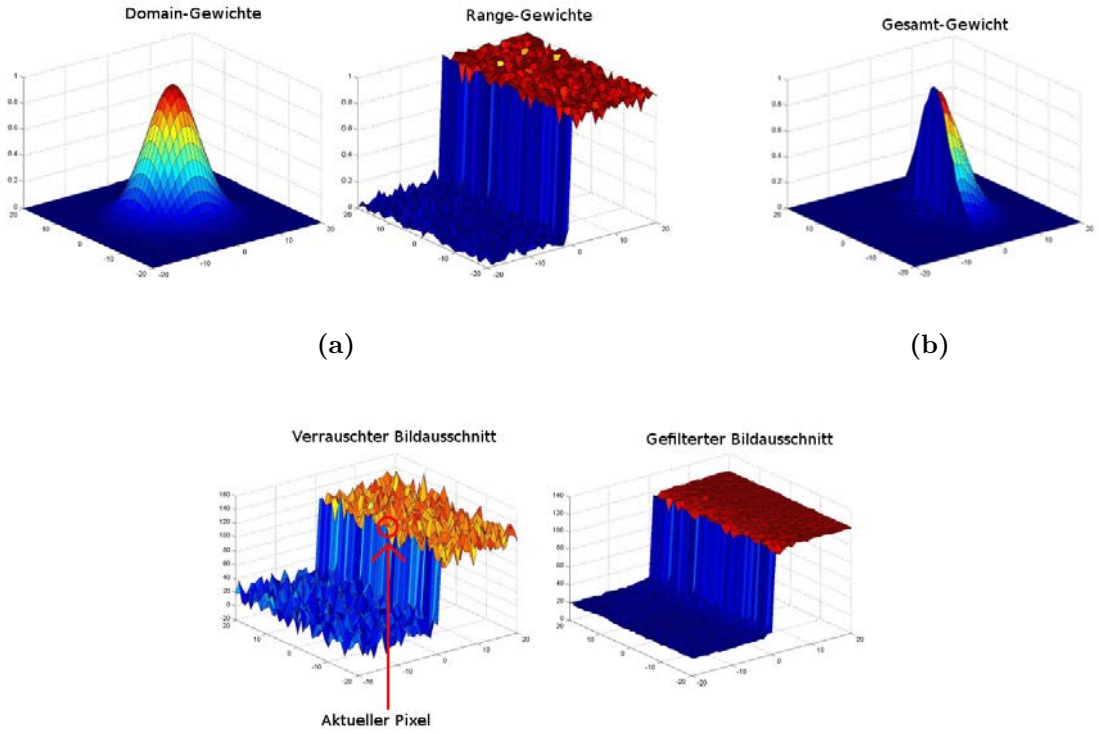


Abbildung 6.2.: Funktionsweise des bilateralen Filters. **(a)** Die Domain-Gewichte sind nur vom Abstand zum zentralen Pixel auf der Bildebene (Domain) anhängig und sind hier als Gauß-Kernel ausgeprägt. Die Range-Gewichte entsprechen den Intensitätsdifferenzen zum zentralen Pixel. **(b)** Das Gesamtgewicht entsteht durch Multiplikation der Domain- mit den Range-Gewichten. **(c)** Diese Grafik zeigt einen beispielhaften Bildausschnitt vor und nach Anwendung des bilateralen Filters. Der aktuell neu zu berechnende Pixel befindet sich dabei im Zentrum des Filterkerns. (Entnommen aus einer Präsentation von Jens Krommweh [48])

Da für die Schätzung der Sensorpose im späteren Verlauf ein „coarse-to-fine"-Algorithmus verwendet wird, werden an dieser Stelle Bildpyramiden als multiskalare Repräsentation der Tiefenbilder berechnet. Dazu wird eine 3-stufige Bildpyramide $D^{l \in [1 \ldots 3]}$ aufgebaut, deren Aufbau in Abbildung 6.3 verdeutlicht wird. Die unterste Ebene D^1 entspricht genau unserem gefilterten Tiefenbild D_t. Die weiteren Ebenen D^{l+1} entstehen aus der Ebene D^l, indem diese in 2×2-Blöcke unterteilt und jeweils ein neuer Bildpunkt aus dem Durchschnitt eines Blockes gebildet wird. So hat jede nächsthöhere Ebene die halbe Auflösung der Vorangegangenen. Pixel, deren Tiefenwert mehr als $3\sigma_r$ von den Tiefenwerten der restlichen Pixel eines Blockes abweichen, werden nicht berücksichtigt, um Glättung entlang von Kanten zu verhindern. Aus der Bildpyramide D^l werden mithilfe der Gleichungen 6.4 und 6.7 Bildpyramiden

mit den Vertices $\mathbf{V}^{l\in[1\dots3]}$ und Normalenvektoren $\mathbf{N}^{l\in[1\dots3]}$ gebildet.
Die Pose der Kamera zum Zeitpunkt t wird durch die homogene Transformationsmatrix

$$\mathbf{T}_{g,t} = \begin{bmatrix} \mathbf{R}_{g,t} & \mathbf{t}_{g,t} \\ \mathbf{0}^T & 1 \end{bmatrix} \tag{6.8}$$

repräsentiert. Dabei ist $\mathbf{R}_{g,t}$ eine Rotationsmatrix sowie $\mathbf{t}_{g,t}$ ein Translationsvektor. Diese Transformation bildet einen beliebigen Punkt $\mathbf{p}_t$ des Kamerakoordinatensystems zum Zeitpunkt t auf einen Punkt $\mathbf{p}_g = \mathbf{T}_{g,t}\mathbf{p}_t$ des globalen Koordinatensystems g ab. Die Vertices $\mathbf{v}_t$ und Normalenvektoren $\mathbf{n}_t$ können so ebenfalls in das globale System g überführt werden:

$$\mathbf{v}_t^g = \mathbf{T}_{g,t}\dot{\mathbf{v}}_t, \tag{6.9}$$

$$\mathbf{n}_t^g = \mathbf{R}_{g,t}\mathbf{n}_t. \tag{6.10}$$

6.1.2. Szenenrepräsentation

Um die zu rekonstruierende Umgebung bzw. Szene zu repräsentierten, wird eine volumetrische TSDF (truncated signed distance function) verwendet. Eine SDF (signed distance function) gibt für jeden Punkt im Raum die Entfernung zur nächst gelegenen Oberfläche an. Dabei sind die Werte in Richtung des freien Raumes positiv und zunehmend und in Richtung der nicht sichtbaren Seite der Oberfläche negativ und nehmen mit weiterem Abstand zur Oberfläche ab (siehe Abbildung 6.4). Dabei ist es möglich, mehrere SDFs zu vereinigen, um so eine Fusion der Oberflächen zu erreichen. Die Wahl eines Distanzfeldes wie der TSDF als Repräsentationsform bietet für die Rekonstruktion auf Basis von Tiefenbildern einige vorteilhafte Eigenschaften gegenüber Punktwolken oder Polygonnetzen (siehe [15]). Dazu zählen bspw. das Verhalten im Bezug auf Messrauschen, die Effizienz, die Möglichkeit verschiedene Modelle zu fusionieren oder die Abwesenheit von Restriktionen auf topologischer Ebene.
Sei $\mathbf{p} \in \mathbb{R}^3$ ein Punkt des globalen Koordinatensystems. Das globale Modell bzw. die TSDF, welche die Vereinigung aller Tiefendaten bis zum Zeitpunkt t enthält, wird mit $S_t(\mathbf{p})$ bezeichnet. Dabei wird eine diskretisierte Form der TSDF mit fester Auflösung im Speicher der Grafikkarte gespeichert. Der Speicherbedarf ist damit nicht effizient. Beispielsweise wird für eine Auflösung von 512^3 bei einem Speicherbedarf von 32 Bit pro Voxel 512 MB Speicher belegt. Dafür wirkt sich diese simple Reprä-

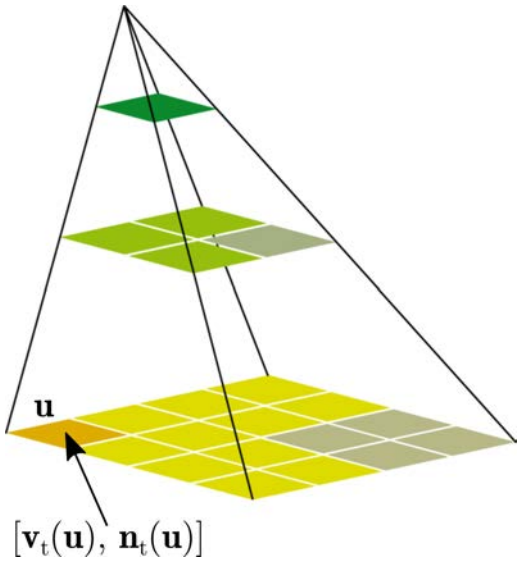


Abbildung 6.3.: Skizzenhafter Aufbau der multiskalaren Bildpyramiden (Grundlage der Grafik entnommen aus der Diplomarbeit von Fabian Naumann [57])

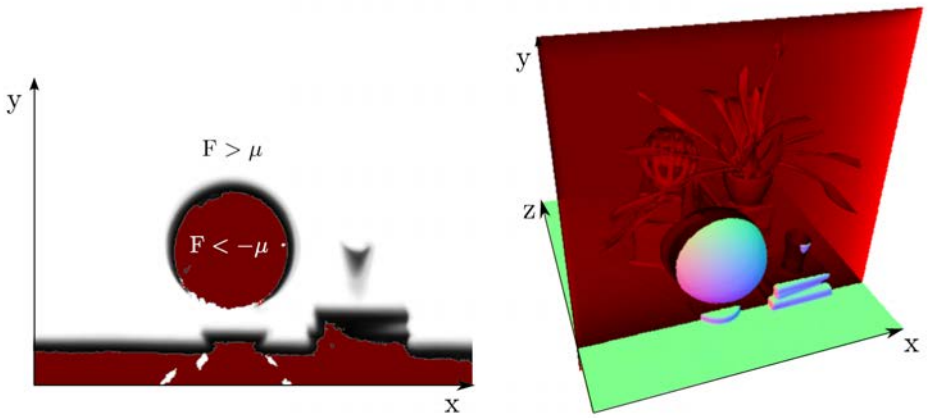


Abbildung 6.4.: Beispielhafte Schnittebene durch eine TSDF: Weiß entspricht begrenzten TSDF-Werten mit $F > \mu$, Rot repräsentiert Voxel ohne gültige Messung (Entnommen aus dem Paper von Newcombe et al. [58])

sentation der TSDF im Speicher positiv auf die Berechnungszeit aus. Dabei werden für jeden Voxel neben dem eigentlichen Distanzwert $F_t(\mathbf{p})$ auch ein Gewicht $W_t(\mathbf{p})$ gespeichert:

$$S_t(\mathbf{p}) \mapsto [F_t(\mathbf{p}), W_t(\mathbf{p})]. \tag{6.11}$$

Eine Tiefenmessung, wie die gegebene Eingabe $R_t(\mathbf{u})$, impliziert zwei wichtige Einschränkungen über die zu rekonstruierte Oberfläche. Dazu wird angenommen, dass die tatsächliche Tiefe in einem Bereich $\pm\mu$ der gemessenen Tiefe liegt. Dann kann davon ausgegangen werden, dass jeder Punkt auf einem von der Kamera ausgehenden

Strahl, dessen Tiefe unterhalb dieses Bereiches liegt, definitiv im leeren Raum liegt. Außerdem können für Punkte, die hinter diesem Bereich liegen, keine Oberflächeninformationen gewonnen werden. Folglich sind zur Repräsentation der Oberflächen nur Distanzwerte in diesem $\pm\mu$-Bereich nötig. Dazu werden die Distanzwerte einer TSDF auf den Bereich $[-\mu, \mu]$ begrenzt. So müssen in einer TSDF Unterschiede zwischen freiem Raum, unsicherer Messung und unbekannten Bereichen nicht explizit modelliert werden.

6.1.3. Tiefenvorhersage

Mithilfe der jeweilig aktuellen Rekonstruktion der Umgebung in Form einer TSDF kann ein virtuelles Tiefenbild bzw. die korrespondierenden Vertices $\hat{\mathbf{v}}_t(\mathbf{u})$ und Normalen $\hat{\mathbf{n}}_t(\mathbf{u})$ aus Sicht der aktuell geschätzten Kamerapose $\mathbf{T}_{g,t}$ berechnet werden. Diese dienen der folgenden Posenschätzung als Referenzeingabe. Des Weiteren kann die TSDF, welche Oberflächeninformationen nicht explizit, sondern lediglich implizit, enthält, so aus beliebiger Kameraperspektive gerendert werden. Dazu wird parallel für jeden Pixel $\mathbf{u}$ des zu generierenden Tiefenbildes ein Raycasting durchgeführt. Dabei wird jeder Strahl $\mathbf{T}_{g,t}\mathbf{K}^{-}1\dot{\mathbf{u}}$ verfolgt, bis ein Nulldurchgang von positiv nach negativ (entspricht in der TSDF einer Oberfläche) gefunden wird. Wird die Rückseite einer Oberfläche (negativ $\rightarrow$ positiv) oder kein Übergang gefunden, kann keine gültige Messung berechnet werden. Um für einen Schnittpunkt mit der Oberfläche $\mathbf{p}$ auch die Normalen zu ermitteln, wird angenommen, dass für Punkte nahe der Oberfläche der Gradient orthogonal zu dieser verläuft. Dazu wird die numerische Ableitung der TSDF $\nabla F = \left[\frac{\partial F}{\partial x}, \frac{\partial F}{\partial y}, \frac{\partial F}{\partial z}\right]^T$ verwendet:

$$\mathbf{R}_{g,t}\hat{\mathbf{n}}_t = \hat{\mathbf{n}}_t^g(\mathbf{u}) = l[\nabla F(\mathbf{p})]. \tag{6.12}$$

Um das Verfahren zu beschleunigen, wird eine möglichst große Schrittweite $\Delta s < \mu$ gewählt. Damit wird garantiert, dass bei jedem Schritt mindestens ein nicht begrenzter Wert erfasst wird, falls eine Oberfläche geschnitten wird. Um daraus den Schnittpunkt mit der Oberfläche zu ermitteln, wird folgende Näherung s^* verwendet. Für zwei Punkte auf dem Strahl mit Abstand s und $s + \Delta s$ vom Ursprung, welche auf unterschiedlichen Seiten des Nullüberganges liegen, sind dabei F_s^+ und $F_{s+\Delta s}^+$ die trilinear interpolierten Werte der SDF:

$$s^* = s - \frac{\Delta s F_s^+}{F_{s+\Delta s}^+ - F_s^+}. \tag{6.13}$$

Aus dem so ermittelten virtuellen Tiefenbild wird im Anschluss, wie bereits in Abschnitt 6.1.1 erläutert, eine Bildpyramide mit den geschätzten Punkten $\hat{\mathbf{V}}_t$ und Normalen $\hat{\mathbf{N}}_t$ aufgebaut, welche der folgenden Posenschätzung als Referenz dient.

6.1.4. Posenschätzung

Um die Tiefendaten der Kamera in die TSDF zu integrieren, muss die Kamera im Raum lokalisiert werden. Dazu wird für jeden Frame bzw. Zeitschritt die aktuelle Kamerapose $\mathbf{T}_{g,t}$ (siehe Gleichung 6.8) geschätzt. Dabei werden alle Tiefendaten verwendet, um eine ICP-basierte Posenschätzung durchzuführen. Da davon ausgegangen werden kann, dass sich aufgrund der hohen Abtastrate von 30 Hz nur kleine Bewegungen zwischen aufeinanderfolgenden Frames stattfinden, lässt sich der „projective data association"-Algorithmus (siehe [6]) für die Auswahl korrespondierender Punkte und die Punkt-Ebenen-Metrik (siehe [13]) für die Optimierung der Pose verwenden. Dieser Ansatz ist, wie bereits im Artikel von Rusinkiewicz et al. [70] erfolgreich demonstriert wurde, sogar echtzeitfähig. Außerdem werden die Berechnungen parallelisiert auf der GPU durchgeführt, sodass eine vorherige Filterung der Daten unter Berücksichtigung der Laufzeit nicht benötigt wird. Dabei wird die aktuelle Tiefenmessung $(\mathbf{V}_t, \mathbf{N}_t)$ und die in Abschnitt 6.1.3 definierte Vorhersage aus dem Umgebungsmodel $(\hat{\mathbf{V}}_{t-1}, \hat{\mathbf{N}}_{t-1})$ als Eingaben für den ICP-Algorithmus verwendet.

Der „projective data association"-Algorithmus findet korrespondierende Punkte zwischen diesen beiden Datensätzen. Jeder Punkt $\mathbf{v}_t$ wird anhand der vorherigen Kamerapose $\mathbf{T}_{g,t-1}$ und der neuen geschätzten Pose $\tilde{\mathbf{T}}^z_{g,t}$ auf einen Pixel

$$\hat{\mathbf{u}} = \pi(\mathbf{K}\tilde{\mathbf{T}}_{t-1,t}\dot{\mathbf{v}}_t(\mathbf{u})) \tag{6.14}$$

der neuen Bildebene projiziert. Dabei dient

$$\tilde{\mathbf{T}}^z_{t-1,t} = (\mathbf{T}_{g,t-1})^{-1}\tilde{\mathbf{T}}^z_{g,t} \tag{6.15}$$

der Transformation vom Vorherigen in das aktuelle Kamerakoordinatensystem. Aus den so berechneten Vertexpaaren $\{\mathbf{v}_t(\mathbf{u}), \hat{\mathbf{v}}_{t-1}(\hat{\mathbf{u}})\}$ werden Paare entfernt, für die gilt, dass ihr Tiefenwert im aktuellen Tiefenbild undefiniert oder ihr Abstand zueinander im Raum bzw. der Winkelabstand der zugehörigen Normalen größer als ein zuvor definierter Schwellenwert ist.

Das Abstandsmaß bzw. die point-plane Energiefunktion einer Posenschätzung $\mathbf{T}_{g,t}$

ist

$$E(\mathbf{T}_{g,t}) = \sum_{\mathbf{u}\in\mathcal{U}\wedge m_t(\mathbf{u})=1} \left|\left|\left(\mathbf{T}_{g,t}\dot{\mathbf{v}}_t(\mathbf{u}) - \hat{\mathbf{v}}^g_{t-1}(\hat{\mathbf{u}})\right)^T \hat{\mathbf{n}}^g_{t-1}(\hat{\mathbf{u}})\right|\right|_2. \tag{6.16}$$

Die Minimierung einer linearisierten Form dieser Energiefunktion liefert eine iterative Lösung $\tilde{\mathbf{T}}^z_{g,t}$. Dabei wird $\tilde{\mathbf{T}}^0_{g,t}$ mit der vorherigen Pose $\mathbf{T}_{g,t-1}$ initialisiert. Um die Robustheit der Posenschätzung zu verbessern, kann $\tilde{\mathbf{T}}^0_{g,t}$ ebenfalls mit einer Schätzung der Drohnenodometrie, bspw. aus LSD-SLAM (siehe Abschnitt 4.3.2), initialisiert werden. Der verwendete ICP-Algorithmus wird zusätzlich in einen „coarse-to-fine"-Algorithmus eingebettet. Dazu werden die in Abschnitt 6.1.1 vorberechneten Vertex- und Normalen-Pyramiden verwendet. Für die Posenschätzung werden, angefangen mit der gröbsten Darstellung, nacheinander die verschiedenen Ebenen verwendet. Dabei werden maximal 4, 5 bzw. 10 Iterationen pro Ebene berechnet. Die Schätzung nach der letzten Iteration $\tilde{\mathbf{T}}^{z_{max}}_{g,t}$ wird als neue Kamerapose $\mathbf{T}_{g,t}$ verwendet und die aktuellen Tiefendaten, wie im folgenden Abschnitt erläutert, mit dem globalen Modell fusioniert.

6.1.5. Fusion mit der Szene

Um die, mithilfe der geschätzten Pose, in das globale Koordinatensystem transformierten Punkte in die globale TSDF aufzunehmen, muss für diese zunächst eine SDF berechnet werden. Um die parallele Verarbeitung auf der GPU wesentlich zu beschleunigen, wird dazu eine sogenannte „projective signed distance function" eingesetzt. Für ein Tiefenbild R_t mit bekannter Kamerapose $\mathbf{T}_{g,t}$ ist die entsprechende projektive TSDF $S_{R_t}(\mathbf{p}) = [F_{R_t}(\mathbf{p}), W_{R_t}(\mathbf{p})]$ am Punkt $\mathbf{p} \in \mathbb{R}^3$ des globalen Koordinatensystems g wie folgt definiert:

$$F_{R_t}(\mathbf{p}) = \Psi\Big(\lambda^{-1}||(\mathbf{t}_{g,t}) - \mathbf{p}||_2 - R_t(\mathbf{x})\Big) \tag{6.17}$$

mit

$$\lambda = ||\mathbf{K}^{-1}\dot{\mathbf{x}}||_2, \tag{6.18}$$

$$\mathbf{x} = \Big\lfloor \pi(\mathbf{K}\mathbf{T}^{-1}_{g,t}\mathbf{p})\Big\rfloor, \tag{6.19}$$

$$\Psi(\eta) = \begin{cases} \min(1, \frac{\eta}{\mu})\mathrm{sgn}(\eta) & \text{falls } \eta \geq -\mu \\ undefined & \text{sonst} \end{cases}. \tag{6.20}$$

Dabei dient λ der Umrechnung der Distanz entlang des Strahles in die Distanz entlang der optischen Achse. Im Weiterem ist $\lfloor \mathbf{u} \rfloor$ der nächste Nachbar von $\mathbf{u}$ in der Bildebene. Die Wahl des nächsten Nachbarn verhindert im Gegensatz zur Interpolation das Verschmieren von Kanten. Die Funktion Ψ begrenzt die maximale Distanz der TSDF. Die Gewichte der TSDF $W_{R_t}(\mathbf{p})$ hängen dabei von der Distanz und dem Winkel θ zwischen dem optischen Strahl und der Oberflächennormalen ab:

$$W_{R_t}(\mathbf{p}) = \frac{\cos(\theta)}{R_t(\mathbf{x})}. \tag{6.21}$$

Die globale TSDF ist dann gerade gleich dem gewichteten Mittel der einzelnen, pro Frame berechneten TSDFs. Dadurch wird gleichzeitig das Messrauschen der einzelnen Frames unterdrückt. Die neben den eigentlichen Distanzwerten der TSDF gespeicherten Gewichte ermöglichen die Verwendung eines gleitenden Mittelwertes:

$$F_t(\mathbf{p}) = \frac{W_{t-1}(\mathbf{p})F_{t-1}(\mathbf{p}) + W_{R_t}(\mathbf{p})F_{R_t}(\mathbf{p})}{W_{t-1}(\mathbf{p}) + W_{R_t}(\mathbf{p})} \tag{6.22}$$

$$W_t(\mathbf{p}) = W_{t-1}(\mathbf{p}) + W_{R_t}(\mathbf{p}). \tag{6.23}$$

Dabei werden nur Voxel aktualisiert, welche nicht verdeckt sind oder im leeren Raum liegen. Bei diesen ist folglich der Distanzwert nicht *undefined* (vergleiche Gl. 6.20). Für die TSDF werden die unveränderten Tiefenwerte R_t statt den bilateral gefilterten D_t benutzt, um feine Strukturen nicht zu stark zu reduzieren. Bei der Berechnung wird außerdem aus Effizienzgründen ein GPU-Thread pro (x,y)-Position im Voxelgitter angelegt und dieses in z-Richtung durchlaufen, anstatt bspw. einen Thread pro Voxel anzulegen.

6.1.6. Transformation in Polygonale Repräsentation

Um die Betrachtung und Verarbeitung des Umgebungsmodells außerhalb des KinectFusion-Frameworks zu ermöglichen, wird die TSDF nach Fusion aller zur Verfügung stehenden Tiefendaten in eine polygonale Repräsentation überführt. Dazu wird aufgrund des implizit vorhandenen Oberflächenmodells ein Marching-Cubes-Algorithmus [51] verwendet.

6.1.7. Erweiterung zum Weltmodell

Die Auflösung bzw. die damit verbundene Größe der TSDF wird aufgrund der direkten Repräsentation als Voxelgitter durch die Größe des Grafikspeichers stark begrenzt. Die Auflösung pro Meter bestimmt wesentlich die Qualität der Rekonstruktion und die Robustheit des Verfahrens. Die auch von den Entwicklern von KinectFusion typischerweise verwendete Größe der TSDF beträgt 3m × 3m × 3m bei einer Auflösung von 512 × 512 × 512. Eine Übersicht verschiedener Auflösungen bzw. Größen und des jeweils benötigte Speicher ist in Tabelle 6.1 gegeben.
Aufgrund dieser Limitierung muss das KinectFusion-Verfahren erweitert werden, um auch große Räume abdecken zu können. Dabei werden im Allgemeinen mehrere TSDF-Volumen angelegt, welche zusammen das wesentlich größere Weltmodell bilden. Wie eine solche Erweiterung mit den restlichen Komponenten von KinectFusion zusammenhängt wird in Abbildung 6.15 (siehe pinker Bereich) verdeutlicht.

KinectFusion Large Scale

Im Rahmen der PCL (Point Cloud Library) wurde beispielsweise die Erweiterung KinectFusion Large Scale (nachfolgend KinfuLS) entworfen und implementiert. Bei KinfuLS werden die verschiedenen quaderförmigen TSDF-Volumen äquidistant entlang der Koordinatenachsen angeordnet und können sich auch überlappen. Die Orientierung der Koordinatenachsen der TSDF-Volumen entsprechen dabei der Orientierung des Weltmodells. Zu jedem Zeitpunkt ist dabei nur ein TSDF-Volumen aktiv und daher direkt in den Grafikspeicher geladen und für KinectFusion verwendbar. Die nicht aktiven TSDF-Volumen werden als Punktwolken (siehe Abschnitt 6.2) abgelegt. Diese sind dabei wesentlich kleiner, da sie nur Punkte enthalten, welche nah an Oberflächen liegen. Insbesondere werden undefinierte Bereiche und freier

Tabelle 6.1.: Speicherbedarf der diskreten TSDF im Vergleich zur Auflösung und zur realen Größe

Auflösung	Größe in Meter	Speicherbedarf in MB
256	1.5	64
384	2.25	216
512	3	512
640	3.75	1000
768	4.5	1728
1024	6	4096

Raum nicht explizit in den Punktwolken gespeichert. Für jeden Voxel der TSDF wird genau ein Punkt in der Punktwolke erzeugt. Um das aktuell aktive TSDF-Volumen zu bestimmen, wird ein Punkt entlang der optischen Achse der Kamera mit definierten Abstand zum Augenpunkt berechnet. Das TSDF-Volumen, dessen Mittelpunkt am nächsten zu diesem Punkt liegt, wird aktiv. Wird ein inaktives TSDF-Volumen aktiviert, wird das vorherig aktive in eine Punktwolke überführt. Anschließend wird, falls vorhanden, die zum jetzt aktiven TSDF-Volumen gehörige Punktwolke in die direkte Repräsentation auf der Grafikkarte transformiert. Dieser Schritt wird als Shift bezeichnet. Der größte Nachteil dieser Art der Anordnung von TSDF-Volumen manifestiert sich beim Übergang von einem TSDF-Quader zum Nächsten. Aufgrund der Tatsache das jeweils nur eine TSDF aktiv ist, besteht die Möglichkeit, dass sich nur ein Teil der aufgenommenen Tiefendaten mit dem aktuellen TSDF-Quader decken. Dies führt dazu, dass das KinectFusion-Verfahren an diesen Stellen wesentlich schlechter funktioniert und eventuell keine hinreichende Posenschätzung vorgenommen werden kann. Im nächsten Unterabschnitt wird daher ein weiteres Verfahren zur Erweiterung von KinectFusion vorgestellt, welches diese Problematik umgeht.

Moving Volume KinectFusion

Im der Veröffentlichung von Roth und Vona [69] wird ein Verfahren vorgestellt, das es erlaubt, die TSDF-Volumen so anzuordnen, dass immer einer optimale Überdeckung der TSDF mit den Tiefendaten gewährleistet ist. Dieses Verfahren wird in diesem Unterabschnitt näher erläutert. Dabei werden die einzelnen TSDF-Volumen nicht wie bei KinfuLS anhand einer festgelegten Anordnung verteilt, sondern nach und nach entlang der Kameratrajektorie ausgerichtet. Dabei wird nicht zu jedem Zeitpunkt ein Shift durchgeführt, sondern nur wenn sich die Kamera mehr als eine bestimmten Länge bewegt oder um mehr als einen bestimmten Winkel gedreht hat. Diese Anordnung wird beispielhaft in Abbildung 6.5 dargestellt.
Zu einem Zeitpunkt t lässt sich die aktuelle Kamerapose $\mathbf{T}_{g,t}$ des aktiven TSDF-Volumen wie folgt auf eine Pose $\mathbf{T}_{w,t}$ im Weltmodell abbilden:

$$\mathbf{T}_{w,t} = \mathbf{P}_0 \dots \mathbf{P}_{vf(t)} \mathbf{T}_{g,t}. \tag{6.24}$$

Dabei ist $\mathbf{P}_0 = \mathbf{I}_{3\times 3}$ und $\mathbf{P}_{i>0}$ gleich der Transformation des i-ten TSDF-Volumen zum $i+1$-ten. Die Funktion $vf(t)$ gibt den Index des zum Zeitpunkt t aktiven TSDF-Volumens an. Um festzustellen, ob ein Shift durchgeführt werden muss, wird der

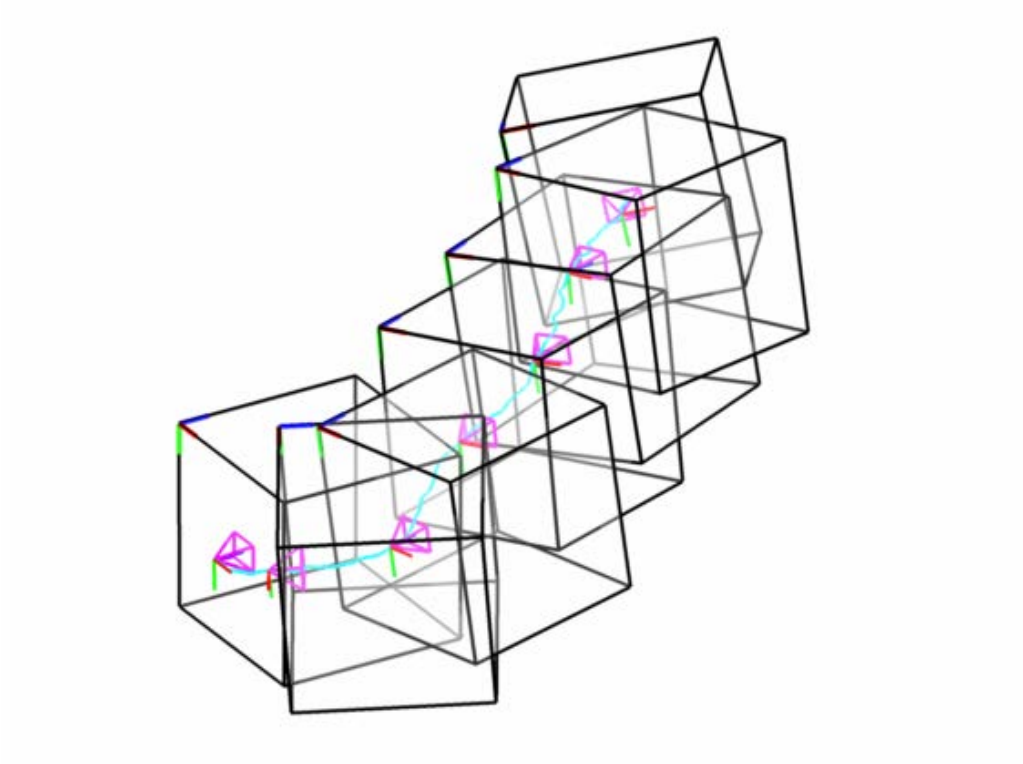


Abbildung 6.5.: Beispielhafte Anordnung der TSDF-Volumen bei Moving Volume KinectFusion (Entnommen aus dem Paper von Roth und Vona [69])

Winkelversatz a_d und die Verschiebung l_d der aktuellen Pose $\mathbf{T}_{g,t}$ zur erwünschten Pose $\mathbf{T}_s$ im Bezug zum TSDF-Volumen berechnet:

$$\mathbf{D} = \begin{bmatrix} \mathbf{R}_d & \mathbf{t}_d \\ \mathbf{0}^T & 1 \end{bmatrix} = \mathbf{T}_s^{-1}\mathbf{T}_{g,t},\ l_d = ||\mathbf{t}_d||_2,\ a_d = ||\text{rodrigues}^{-1}(\mathbf{R}_d)||_2 \tag{6.25}$$

Als erwünschte Pose wird die standardmäßige initiale Kamerapose

$$\mathbf{T}_s = \begin{bmatrix} \mathbf{I}_{3\times3} & \mathbf{t}_s \\ \mathbf{0}^T & 1 \end{bmatrix}, \mathbf{t}_s = \begin{bmatrix} W_m/2 \\ H_m/2 \\ -D_m/10 \end{bmatrix} \tag{6.26}$$

gewählt. Dabei sind W_m, H_m und D_m die Breite, Höhe und Tiefe der TSDF-Volumen. Ist $l_d > l_{max}$ oder $a_d > a_{max}$ wird ein Shift durchgeführt. Dabei werden von den Autoren $l_{max} = 0.3$ m und $a_{max} = 0.05$ rad als typische Schwellwerte definiert. Dies sorgt dafür, dass sich die Kamera immer leicht hinter dem TSDF-Volumen befindet und sich die Tiefendaten so sehr gut von dem TSDF-Volumen abgedeckt werden.

Bei einem Shift wird die vorherige in die neue TSDF transformiert. Dazu werden

zwei TSDFs im Grafikspeicher angelegt, um einen „swap buffer“ zu realisieren und die Transformation schneller durchführen zu können. Dadurch verdoppelt sich zwar der Speicherbedarf, jedoch reicht der Speicherplatz auf modernen Grafikkarten je nach Größe der TSDF trotzdem aus (siehe Tabelle 6.1). Nach der Transformation in die neue TSDF wird die relative Transformation zum letzten TSDF-Volumen als $\mathbf{P}_{n+1} = \mathbf{T}_{g,t}\mathbf{T}_{g+1,t}^{-1}$ definiert. Dabei entspricht $\mathbf{T}_{g+1,k}$ der Kamerapose im Bezug zum neuen TSDF-Volumen. Konzeptionell ist $\mathbf{T}_{g+1,t} = \mathbf{T}_s$. In der Realität wird jedoch ein kleiner Versatz erlaubt, was nachfolgend näher erläutert wird. Zur Berechnung der neuen TSDF werden die Werte der vorherigen interpoliert. Um das Verfahren zu beschleunigen, werden zwei Arten von Transformationen verwendet. Ist $l_d > l_{max}$ und $a_d <= a_{max}$, wird die Rotation vernachlässigt und eine schnelle und exakte Verschiebung der TSDF entlang der Koordinatenachsen durchgeführt. Andernfalls wird ein Resampling mit trilinearer Interpolation verwendet.

Bei der einfachen Verschiebung wird die TSDF um ganze Voxel verschoben. Dazu werden lediglich die Voxel im Speicher umkopiert und müssen nicht approximiert werden. Da wir nur um ganze Voxel verschieben wird die neue Kamerapose und die Transformation zur neuen TSDF wie folgt definiert:

$$\mathbf{T}_{g+1,t} = \begin{bmatrix} \mathbf{R}_{g,t} & \mathbf{t}_{g,t} - \text{round}(\mathbf{t}_s) \\ \mathbf{0}^T & 1 \end{bmatrix}, \mathbf{P}_{n+1} = \begin{bmatrix} \mathbf{I}_{3\times3} & \text{round}(\mathbf{t}_s) \\ \mathbf{0}^T & 1 \end{bmatrix}. \tag{6.27}$$

Gerundet wird dabei im Bezug auf die Größe der Voxel.

Ist $a_d > a_{max}$ wird ein Resampling mit trilinearer Interpolation der Distanz- und Gewichtungswerte durchgeführt. In diesem Fall ist die neue Kamerapose und die Transformation des Volumen

$$\mathbf{T}_{g+1,t} = \mathbf{T}_s,\ \mathbf{P}_{n+1} = \mathbf{T}_{g,t}\mathbf{T}_s^{-1}. \tag{6.28}$$

Zur Approximation der TSDF könnte ebenfalls ein „nearest neighbor resampling“ verwendet werden, welches offensichtlich schneller ist als die trilineare Interpolation. Experimente der Autoren haben jedoch gezeigt, dass dies schnell zu Fehlern beim Tracking und zu offensichtlichen geometrischen Artefakten führt. Zur Beschleunigung der Interpolation wird allerdings vor der teureren trilinearen Interpolation der nächste Nachbar gesucht und anhand von diesem überprüft, ob sich überhaupt eine Oberfläche in der Nähe befindet und somit eine Interpolation nötig ist.

6.2. Punktwolkenbasierte Rekonstruktion

Die erzeugten Aufnahmen einer Tiefenkamera werden gewöhnlich als Punktwolken gespeichert. Dabei ist eine Punktwolke $\mathcal{P} \subseteq \mathbb{R}^k$ eine Menge von Punkten im k-dimensionalen Raum. Für unsere Anwendung ist $k = 3$ anzunehmen, wobei spätere Erläuterungen der Einfachheit halber mit $k = 2$ erfolgen. Die Punktwolken müssen effizient gespeichert, entrauscht und reduziert werden, damit eine erfolgreiche Rekonstruktion erfolgen kann. Besonders die Poissonrekonstruktion (Abschnitt 6.2.4) liefert bei stark verrauschten Daten unbrauchbare Ergebnisse. In diesem Abschnitt werden Verfahren zur Repräsentation, Reduktion, Rekonstruktion und zum Entrauschen vorgestellt.

6.2.1. Punktwolkenrepäsentation und Verarbeitung

Aufnahmen einer Tiefenkamera sind in der Regel verrauscht und umfassen eine Vielzahl an Punkten. In der Projektgruppe gab es Szenen, die Punktwolken mit mehreren Millionen Punkten lieferten. Dies ist ein Problem für verarbeitende Algorithmen, da sie über diese Menge an Punkten iterieren müssen, um beispielsweise passende Korrespondenzen (siehe Kapitel 6.2.3) oder die Umgebung (oft als k-Nachbarschaft bezeichnet) zu finden. Die Anzahl k der betrachteten benachbarten Punkte wird vor der Berechnung festgelegt oder mit Hilfe eines Radius r um den relevanten Punkt bestimmt. Hierbei ist die k-Nachbarschaft die Menge an k Punkten mit minimalen Abstand zu Referenzpunkt, oder die Menge an Punkten, welche in der Umgebung des Referenzpunktes mit Radius r liegen.
Diese Berechnungen kosten viel Zeit, wodurch die Möglichkeit für eine effiziente Speicherung und Suche maßgebend für die Laufzeit dieser Algorithmen ist. Eine zusätzliche Verbesserung kann durch eine gezielte Reduktion der Punktwolke erreicht werden, indem eine Teilmenge an Punkten gewählt, oder neue Repräsentanten für die Punktwolke erzeugt werden.

k-d-Baum

Eine effiziente Datenhaltung von Punktwolken ist mit einem k-d-Baum möglich. Dies ist eine Verallgemeinerung von binären Bäumen und für den Iterative Closest Point Algorithmus (vergleiche Kapitel 6.2.3), sowie für die Suche nach benachbarten Punkten von zentraler Bedeutung. Hierbei werden für die Initialisierung, einer Organisation der Daten anhand ihrer Koordinaten, erhöhte Kosten in Kauf genommen, was die Suche in der Datenstruktur beschleunigt. Wie bei der binären Suche

lässt sich ein entsprechender Punkt in $O(\log \mathcal{P})$ finden. $\mathcal{P}$ ist dabei die Menge aller gespeicherten Punkte in dem k-d-Baum.
Ein binärer k-d-Baum $T = (\mathcal{P}, \mathcal{E}) \subset \mathbb{R}^2$ besitzt für den zwei-dimensionalen Fall folgende Eigenschaften:

Für jeden Knoten $k \in \mathcal{P}$ gerader Höhe gilt, dass die x-Koordinate jedes Kindes k' im rechten Teilbaum größer als die x-Koordinate von k ist und im linken Teilbaum von k kleiner oder gleich.

Für jeden Knoten $k \in \mathcal{P}$ ungerader Höhe gilt, dass die y-Koordinate jedes Kindes k' im rechten Teilbaum größer als die y-Koordinate von k ist und im linken Teilbaum von k kleiner oder gleich.

Die Punktmenge $\mathcal{P} \subset \mathbb{R}^2$ wird somit an einem gegebenen Punkt $\mathbf{p} \in \mathcal{P}$ in zwei Teilmengen gespalten. Dies wird abwechselnd für die x- und y-Koordinate durchgeführt, bis die zu teilende Menge nur noch aus genau einem Punkt besteht. Dieser Punkt bildet ein Blatt des Baumes und enthält keine Kinder. Um einen ausgeglichen k-d-Baum zu erhalten und dabei eine Laufzeit von $O(\log \mathcal{P})$ garantieren zu können, sollte der Median als Wurzel gewählt werden, wobei die Menge der Punkte dafür nach den jeweiligen Koordinaten sortiert sein muss. Ein Beispiel für einen ausgeglichenen zweidimensionalen k-d-Baum ist in Abbildung 6.6 gegeben. Die Erweiterung für den drei-dimensionalen Fall besteht darin, dass zusätzlich eine Teilung nach der z-Koordinate vorgenommen wird.
Der k-d-Baum wird in vielen Algorithmen der Projektgruppe eingesetzt um die Laufzeit für die Suche deutlich zu verbessern. So wird der k-d-Baum z. B. für die Suche der Korrespondenzen im ICP-Algorithmus, die Ermittlung der k-Nachbarschaft innerhalb der Normalen- und Merkmalsbestimmung, sowie der Poisson Oberflächenrekonstruktion verwendet.

Entrauschen

Durch Messfehler der Tiefenkamera kann sogenanntes Rauschen entstehen. Das Rauschen ist dabei eine Menge von Punkten innerhalb der Punktwolke, welche nicht zur eigentlichen Szene gehören. Diese Punkte vergrößern die Punktwolke und können z. B. zu falschen Korrespondenzen in der Registrierung führen. Sei $\mathcal{P} \subseteq \mathbb{R}^2$ die ursprüngliche und rauschfreie Punktwolke und $\mathbf{p}^{(u)} \in \mathcal{P}$ ein Punkt dieser Punktwolke. Der korrespondierende verrauschte Punkt $\mathbf{p}^{(r)}$ weicht von $\mathbf{p}^{(u)}$ ab, sodass $\|\mathbf{p}^{(u)} - \mathbf{p}^{(r)}\|_2 \neq 0$ gilt. Ziel der Rauschentfernung ist es, diese Störung zu finden

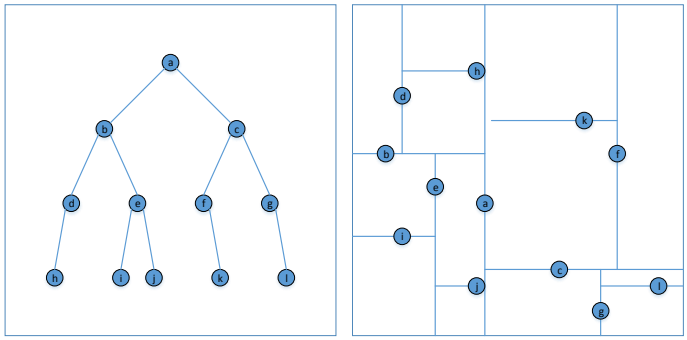


Abbildung 6.6.: Schematische Punktmengenaufteilung eines zwei-dimensionalen k-d-Baumes

und eliminieren, sodass nachfolgende Bearbeitungsverfahren ordentliche Ergebnisse liefern können.
Eine Bestimmung, ob ein Punkt verrauscht ist, erfolgt mit der Berechnung der k-Nächsten-Nachbarn, dabei steht k für die Anzahl der gesuchten Punkte mit minimalen euklidischen Abstand. Hierbei wird der mittlere euklidische Abstand und die Standardabweichung für die k nächsten Nachbarn ermittelt. Sei $n_k(\mathbf{p})$ die Menge der nächsten k Nachbarn des Punktes $\mathbf{p}$, dann wird der mittlere Abstand durch

$$M(\mathbf{p}) = \frac{1}{k} \sum_{\mathbf{p}' \in n_k(\mathbf{p})} (\mathbf{p}' - \mathbf{p}) \tag{6.29}$$

berechnet. Ebenso wird die Standardabweichung $\sigma_{\mathbf{p}}$ berechnet und damit ein Intervall $G = M(\mathbf{p}) \pm \sigma_{\mathbf{p}} \cdot f$ definiert, wobei der Faktor f das Gewicht der Standardabweichung regeln kann. Die Wahl des Faktors f ist entscheidend, je geringer f gewählt wird desto stärker wird gefiltert. Alle Punkte, deren mittlere Abweichungen nicht in dem gegebenen Intervall G liegen, werden entfernt. Dies führt zu einer entrauschten und gleichzeitig reduzierten Punktwolke. Dieses Verfahren kann direkt für den dreidimensionalen Fall übertragen werden.
Innerhalb der Entwicklungsphase stellte sich heraus, dass die Ergebnisse des Kinfu-Verfahrens (siehe Kapitel 6.1) sehr gut sind und kein zusätzliches Entrauschen nötig ist. Nur für die Poisson Oberflächenrekonstruktion erwies sich eine Filterung von Ausreißern als sinnvoll. Hierfür wird der Faktor für das Intervall G mit $f = 1.0$ festgelegt.

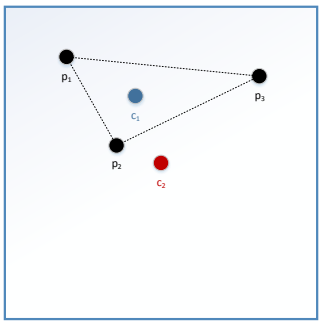


Abbildung 6.7.: Inhalt eines Voxels: 3 Punkte $\mathbf{p}_1$ - $\mathbf{p}_3$ sowie die berechneten Repräsentanten, Zentroid c_1 (blau) und der Mittelpunkt c_2 (rot)

Reduktion

Eine zusätzliche Reduktion der Punktwolken ist mit Hilfe des Voxelgrid-Filters möglich. Mit diesem wird der Raum in Voxel mit festgelegter Größe k unterteilt. Ein Voxel entspricht einem Würfel und enthält eine Teilmenge der ursprünglichen Punktwolke. Gesucht wird nun ein Repräsentant für alle Punkte innerhalb des Würfels. Im einfachsten Fall wird der Mittelpunkt gewählt, wodurch nur überprüft werden muss, ob der Würfel überhaupt Punkte enthält. Die Berechnung eines Zentroids, dem Mittelpunkt aller im Voxel liegenden Punkte, ist genauer, jedoch zeitaufwändiger. Ein zwei-dimensionales Beispiel mit drei Punkten ist in Abbildung 6.7 zu sehen.
Die Zentroiden ersetzten alle Punkte in dem betrachteten Voxel der Punktwolke. Pro Voxel existiert somit höchstens ein Punkt. Die Wahl des Parameters k für die Größe der Voxel ist ausschlaggebend für die Größe der resultierenden Punktwolke. Wird k zu groß gewählt können wichtige Details verloren gehen. Ein vollkommener Verzicht des Voxelgrid-Filters ist jedoch aufgrund von Laufzeit- und Speicherbeschränkungen nicht möglich.
Für die Projektgruppe ist eine Größe zwischen 5mm und 1cm angebracht und wird besonders bei der Grobregistrierung (siehe Kapitel 6.2.2) eingesetzt um die Laufzeit bei der Berechnung einer initialen Transformation deutlich zu reduzieren. Eine Verwendung bei der Poisson Oberflächenrekonstruktion wurde im Rahmen der Projektgruppe getestet, um die Rekonstruktion großer Räume zu ermöglichen. Dabei wurde das rekonstruierte Modell jedoch sehr detailarm, wodurch auf eine vollständige Rekonstruktion mit Hilfe des Voxelgrid-Filters verzichtet wurde.

6.2.2. Registrierung

Eine Registrierung der Teilergebnisse des Kinfu-Verfahrens ist nötig, falls das Tracking während der Aufnahme abbricht und somit eine Menge von Teilmodellen entsteht. Im schlimmsten Fall gibt es keine Informationen über deren Positionen innerhalb des Raumes. Diese Teilmodelle müssen erneut registriert werden, damit ein vollständiges Modell des erkundeten Raumes entstehen kann. Hierfür können die Vertices des Modells als eine Punktwolke interpretiert und eine punktwolkenbasierte Registrierung durchgeführt werden. Mit der ermittelten Transformation der Registrierung lassen sich die Teilmodelle zusammenfügen. Abbildung 6.8 verdeutlicht den Ablauf der Registrierung. Die Registrierung orientiert sich an der Arbeit von Hsieh [39] und verwendet die Point Cloud Library (PCL[1]). Alle für die Registrierung relevanten und hier aufgelisteten Verfahren lassen sich in der PCL finden, wobei die PCL eine open-source Programmbibliothek mit einer Vielzahl von Algorithmen zur Bearbeitung von Punktwolken und Polygonnetzen ist.
Zuerst können die optionalen Schritte Filter und Reduktion ausgeführt werden. Im Filterungsschritt können potentielle Ausreißer aus den Punktwolken entfernt und in der Reduktion eine grundsätzliche Verringerung der Daten erzielt werden. Beide Schritte verbessern erheblich die Laufzeit sowie den Speicherbedarf. Gerade für die nachfolgenden Schritte, der Merkmalsbestimmung und Grobregistrierung, können auf Details in den Punktwolken verzichtet werden, um eine grobe initiale Transformation zwischen den zu registrierenden Punktwolken zu finden. Erst im letzten Schritt, der Feinregistrierung mit dem Iterative Closest Point Algorithmus (ICP), sind die Details entscheidend für die Qualität des Ergebnisses.
Die Verarbeitungsschritte Filter und Reduktion wurden in dem vorherigen Abschnitt (vergleiche Abschnitt 6.2.1) behandelt. Im folgendem Abschnitt erfolgt die Erläuterung der Merkmalsbestimmung und abschließenden Grobregistrierung. Eine ausführliche Erklärung des ICP-Algorithmus findet im Kapitel 6.2.3 statt.

Merkmalsbestimmung

Um interessante Punkte, auch Schlüsselpunkte genannt, bestimmen zu können, müssen zuerst die Merkmale eines Punktes beschrieben werden. Zu den Merkmalen gehören die Lage des Punktes $\mathbf{p}$, seine Umgebung und seine Richtung als Normalenvektor $\mathbf{n}$. Die Umgebung, mit einem vorgegebenen Radius r, wird über die k-Nachbarschaft definiert. Die k-Nachbarschaft ist die Menge an k Punkten mit minimalen Abstand

[1] `http://pointclouds.org/` Abruf: 19.06.16 17:10

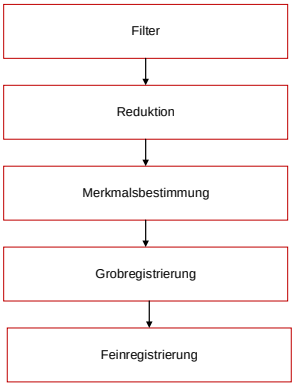


Abbildung 6.8.: Verarbeitungsschritte für die Registrierung von Punktwolken: Filterung und Reduktion, Berechnung von räumlichen Merkmalen sowie Grob-/Feinregistrierung

zu Referenzpunkt $\mathbf{p}$, oder alle Punkte, welche in der Umgebung des Referenzpunktes $\mathbf{p}$ mit Radius r liegen. Alle Normalenvektoren werden mit Hilfe der Hauptkomponentenanalyse [55] ermittelt. Dafür muss eine orthogonale Basis gefunden werden, welche die k-Nachbarschaft des Punktes am Besten repräsentiert. Abschließend muss das Vorzeichen für alle berechneten Normalen überprüft und ggf. angepasst werden, da das Verfahren keine einheitliche Richtung liefert.
Mit diesen Informationen ist nun eine Beschreibung der lokalen Geometrie möglich. Für alle Punktepaare $\mathbf{p}^{(i)}$ und $\mathbf{p}^{(j)}$ (i $\neq$ j) in der k-Nachbarschaft des Punktes $\mathbf{p}$ wird folgendes *Simplified Point Feature Histogram (SPFH)* bestimmt:

$$\begin{aligned} \alpha &= v \cdot \mathbf{n}^{(j)} \\ \phi &= (u \cdot (\mathbf{p}^{(j)} - \mathbf{p}^{(i)}))/\|\mathbf{p}^{(j)} - \mathbf{p}^{(i)}\| \\ \theta &= arctan(w \cdot \mathbf{n}^{(j)}, u \cdot \mathbf{n}^{(j)}), \end{aligned} \tag{6.30}$$

mit $u = \mathbf{n}^{(i)}, v = (\mathbf{p}^{(j)} - \mathbf{p}^{(i)}) \times u, w = u \times v$ und $\mathbf{n}^{(i)}$ bzw. $\mathbf{n}^{(j)}$ zu den Punkten $\mathbf{p}^{(i)}$ und $\mathbf{p}^{(j)}$ gehörigen Normalen. Abschließend wird das *Fast Point Feature Histogram (FPFH) [71]* für den Punkt $\mathbf{p}$ berechnet:

$$FPFH(\mathbf{p}) = SPFH(\mathbf{p}) + \frac{1}{k} \sum_{i=1}^{k} \frac{1}{\omega_k} SPFH(\mathbf{p}^{(k)}), \tag{6.31}$$

wobei ω_k dem Gewicht zwischen Punkt $\mathbf{p}$ und seinem Nachbarn $\mathbf{p}^{(k)}$ entspricht.

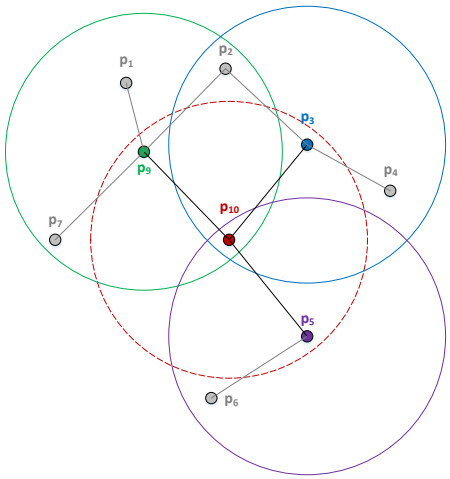


Abbildung 6.9.: Die Umgebung des roten Punktes p_{10} mit seinen k-Nachbarn p_3 (blau),p_5 (violett) und p_9 (grün) und dessen Einflussbereich. Die restlichen Punkte (grau) werden für die FPFH-Berechnung nicht betrachtet

Abbildung 6.9 zeigt ein Beispiel für die k-Nachbarschaft der FPFH-Berechnung im zwei-dimensionalen Fall. Besonders geeignet sind Punkte, die eine markante Umgebung (z. B. an Ecken eines Objektes) besitzen, sodass eine eindeutige Zuordnung erfolgen kann. Diese Punkte bilden die Schlüsselpunkte für die nachgeschaltete Registrierung.
Die Merkmalsbestimmung mit den FPFHs ist in der Projektgruppe ein essenzieller Schritt um die Umgebung eines Punktes zu beschreiben und somit eine initiale Transformation durch die Grobregistrierung erhalten zu können. Merkmalsbestimmung und Grobregistrierung sind stark miteinander verknüpft.

Grobregistrierung

Die Grobregistrierung liefert eine initiale Transformationsschätzung, wodurch der abschließende *Iterative Closest Point Algorithmus (ICP)* eine bessere Chance besitzt, gegen ein globales Minimum zu konvergieren und die Basis für eine erfolgreiche Oberflächenrekonstruktion zu liefern. Auch die Registrierung von Teilmodellen aus dem Kinfu-Verfahren benötigt eine gute Grobregistrierung, um erfolgreich zu sein. Der entscheidende Schritt ist dabei alle Punktkorrespondenzen in den zu registrierenden Punktwolken zu finden.
Anfangs wird eine Teilmenge an Punkten $\mathbf{p}^{(i)}$ aus der Startpunktpunktwolke $\mathcal{S}$ und gleichzeitig eine Teilmenge an Punkten $\mathbf{p}^{(j)}$ aus der Zielpunktpunktwolke $\mathcal{Z}$ gewählt,

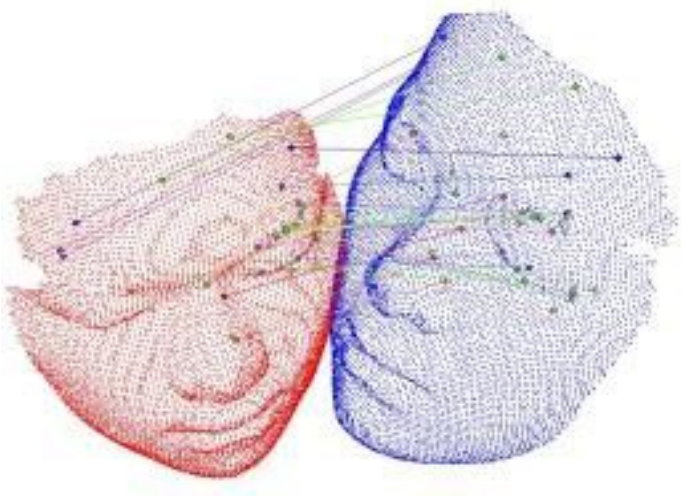


Abbildung 6.10.: Punktkorrespondenzen zwischen zwei Punktwolken
Die Abbildung stammt aus dem Paper *An efficient development of 3D surface registration by Point Cloud Library (PCL)* von Hsieh [39].

wobei der Abstand zwischen den Punkten eine gewisse Distanz d aufweisen sollte. Dies lässt sich leicht mit einer Reduktion durch den Voxelgrid-Filter (vergleiche Abschnitt 6.2.1) realisieren, da pro Voxel höchstens ein Punkt existiert. Alle Punkte $\mathbf{p}^{(i)}$ der Teilmenge aus $\mathcal{S}$ werden mit allen Punkten $\mathbf{p}^{(j)}$ aus der Teilmenge der Zielpunktwolke $\mathcal{Z}$ verglichen. Besitzen beide ähnliche Merkmale, wird die Korrespondenz gespeichert. Bei mehreren Korrespondenzen wird ein zufälliger Punkt bestimmt. Ein Beispiel für eine Menge von Punktkorrespondenzen zwischen zwei Punktewolken ist in Abbildung 6.10 zu sehen.
Anschließend findet die Berechnung der Transformation anhand einer Fehlerfunktion statt. Diese Schritte werden solange wiederholt bis eine maximale Anzahl an Iterationen erreicht wurde. Der Algorithmus basiert auf dem RANSAC-Verfahren (siehe das Paper von Fischler et al. [26]). Abschließend kann die Feinregistrierung, z. B. durch den ICP-Algorithmus, durchgeführt werden.

6.2.3. Iterative Closest Point Algorithmus

Um zwei Punktwolken, ohne vorheriges Wissen über deren Beziehung zueinander, miteinander verknüpfen zu können, wird der Iterative Closest Point (ICP [5]) Algorithmus verwendet. Dieser ist auf Grund seiner Geschwindigkeit besonders etabliert und es sind zahlreiche Modifikationen und Erweiterungen des Algorithmus (eine Zusammenfassung lässt sich in dem Paper von Pomerleau [65] finden) vorgeschlagen worden, wodurch besonders die Änderung, einen k-d-Baum zur Suche benachbarter

Punkte zu nutzen (vergleiche die Arbeit von Zhang [78]), in vielen Implementierungen zum Tragen kommt. Im Folgenden wird nur der ursprüngliche Algorithmus mit den Veränderungen durch Zhang betrachtet.
Eine typische Verarbeitungspipeline ist in Abbildung 6.11 dargestellt. Um die Konvergenz des Algorithmus zu verbessern, sollten die zu registrierenden Punktwolken zuerst initialisiert werden. Dies ist notwendig falls der Abstand der beiden Punktwolken zueinander groß ist, da der Algorithmus fehlerhafte Punktkorrespondenzen ermitteln und somit gegen ein lokales Minimum konvergieren kann. Anschließend können die Punktwolken reduziert (siehe Abschnitt 6.2.1) werden, was nachfolgende Berechnungen deutlich beschleunigen kann. Der entscheidende und aufwändigste Teil des Algorithmus ist die Suche nach den nächsten Punkten der jeweils anderen Punktwolke. Hierbei müssen Punkte, welche nur in einer der beiden Punktwolken existieren, herausgefiltert werden. Dies ist mit Hilfe des Parameters für die maximale Distanz möglich, womit Punkte mit einer größeren Distanz nicht bei der Suche berücksichtigt werden. Abschließend erfolgt die Berechnung der Transformation, um die Punktwolken ineinander zu überführen. Ist der Abstand kleiner als ein vorgegebener Grenzwert oder eine maximale Anzahl an Iterationen erreicht, terminiert der Algorithmus und liefert die entsprechende Transformation zurück. Diese beiden Parameter sind, wie auch die maximale Distanz, zu Beginn des Verfahrens zu definieren und beeinflussen maßgeblich die Resultate der Registrierung.
Der ICP-Algorithmus kommt auch auch im Kinfu-Verfahren für die Posenschätzung (siehe Abschnitt 6.1.4) zum Einsatz, wird dort jedoch modifiziert um Berechnungen parallel auf einer GPU durchführen zu können.

Problembeschreibung

Neben einer Startpunktwolke $\mathcal{S}$ mit Punkten $\mathbf{p}^{(i)}$ $(i = 1, ..., m) \in S$ existiert eine Zielpunktwolke $\mathcal{Z}$ mit Punkten $\mathbf{p}^{(j)}$ $(j = 1, ..., n) \in \mathcal{Z}$. Gesucht ist eine Transformation $\mathbf{T}$, die auf $\mathcal{S}$ angewendet wird, sodass der Abstand zwischen den beiden Punktwolken $\mathcal{S}$ und $\mathcal{Z}$ minimal ist. Die Transformation $\mathbf{T}$ besteht aus einem Translationsvector $\mathbf{t}$ sowie einer Rotationsmatrix $\mathbf{R}$. Um die optimale Transformation zu finden muss

$$T(\mathbf{R}, \mathbf{t}) = \frac{1}{\sum_{i=1}^{m} k_i} \sum_{i=1}^{m} k_i d^2(\mathbf{R}\mathbf{p}^{(i)} + \mathbf{t}, \mathcal{Z}) + \frac{1}{\sum_{j=1}^{n} k_j} \sum_{j=1}^{n} k_j d^2(\mathbf{R}\mathbf{p}^{(j)} + \mathbf{t}, \mathcal{S}) \quad (6.32)$$

minimiert werden, wobei $d(\mathbf{p}^{(i)}, \mathcal{Z})$ (siehe Gl. 6.32) die Distanz zwischen dem Punkt

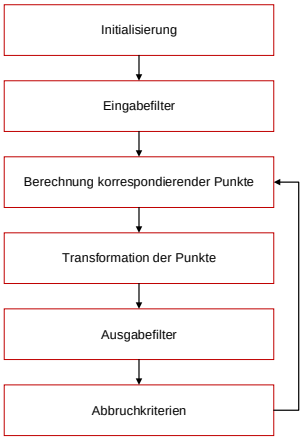


Abbildung 6.11.: ICP Verarbeitungsschritte: initiale Transformationsschätzung, optionaler Eingabefilter und der Interation zur finalen Transformation

$\mathbf{p}^{(i)}$ und der Zielpunktwolke $\mathcal{Z}$ und $d(\mathbf{p}^{(j)}, \mathcal{S})$ die Distanz zwischen $\mathbf{p}^{(j)}$ und der Startpunktwolke $\mathcal{S}$ beschreibt. Die Koeffizienten k_i bzw. k_j nehmen den Wert Eins an, wenn ein passender Punkt in der jeweils anderen Punktwolke existiert. Ist dies nicht der Fall, sind sie Null. In rauschfreien identischen Punktwolken existiert eine optimale Transformation $T(\mathbf{R}, \mathbf{t})$, sodass, nach Durchführung der Transformation, der Abstand jedes Punktes zu seiner Korrespondenz genau Null ist. In der Realität entsprechen die Punkte einer Punktwolke den Punkten auf der Oberfläche eines Objektes. Jedoch werden nicht genau die gleichen Punkte aus verschiedenen Perspektiven, auf Grund der Pixelanzahl einer Kamera, aufgenommen, wodurch eine optimale Transformation trotzdem einen minimalen Abstand besitzt. Die vorherige Gl. 6.32 ist symmetrisch bzgl. der Punktwolken, wodurch auf den zweiten Term für die Berechnung verzichtet und die Gl. 6.32 zu

$$T(\mathbf{R}, \mathbf{t}) = \frac{1}{\sum_{i=1}^{m} k_i} \sum_{i=1}^{m} k_i d^2(\mathbf{R}\mathbf{p}^{(i)} + \mathbf{t}, \mathcal{Z}) \tag{6.33}$$

vereinfacht werden kann. Dabei ist ein Problem, dass die korrespondierenden Punkte erst durch den Algorithmus ermittelt werden, wodurch eine Konvergenz gegen ein lokales Minimum möglich ist (siehe Abbildung 6.12). Dies lässt sich durch eine gute Initialisierung des Algorithmus beheben.

Initialisierung

Eine gute initiale Transformation ist wichtig, damit der ICP-Algorithmus gegen ein globales Minimum konvergiert. Diese Transformation lässt sich schätzen z. B. mit

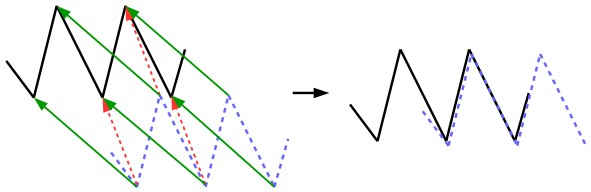


Abbildung 6.12.: Konvergenz gegen ein lokales Minimum: korrekte Transformation (grün) und fehlerhafte Transformation (rote), wodurch ein falsches Gesamtbild (rechts) erzeugt wird

Hilfe einer Grobregistrierung, indem eine Menge von Punkten, entweder zufällig oder durch die Umgebungseigenschaften, bestimmt und mit diesen Teilmengen eine erste Transformation berechnet wird (siehe Abschnitt 6.2.2). Auch besteht die Möglichkeit eine Transformation durch andere Quellen z. B. mit Hilfe der Odometrie der Drohne zu gewinnen. Hierbei könnte die Bewegung der Drohne zwischen zwei Kameraaufnahmen genutzt werden um eine Transformationsschätzung zu erhalten, ohne dabei erheblichen zusätzlichen Aufwand betreiben zu müssen. Der ICP-Algorithmus ist nur erfolgreich, wenn die Transformation zwischen zwei Punktwolken gering ist, z.B. durch eine hohe Frequenz der Aufnahmen oder eine genaue Approximation der Transformation zu Beginn vorliegt.
Im Rahmen der Projektgruppe wird bei dem Kinfu-Verfahren (siehe Abschnitt 6.1.4) auf eine Initialisierung verzichtet, da eine sehr hohe Bildrate durch die Kamera erwartet wird. Für die Registrierung von Teilmodellen, auf Grund von Trackingproblemen während der Aufnahme, wird eine Grobregistrierung, basierend auf einer Merkmalserkennung (siehe Abschnitt 6.2.2), verwendet.

Eingabefilter

Punktwolken können abhängig von der Auflösung der Tiefenkamera sehr groß sein, wodurch die Rechenzeit nicht mehr akzeptabel ist. Zwei unterschiedliche Ansätze, um dieses Problem zu beheben, sind in der Literatur zu finden. Zum einen kann nur eine Teilmenge der Punktewolke für die Registrierung verwendet werden, indem nur jeder n-te Punkt in die Berechnung einfließt oder eine neue reduzierte Punktwolke z. B. mit Hilfe eines Voxelgrid-Filters (siehe Abschnitt 6.2.1) erzeugt wird. Zum anderen gibt es die Möglichkeit besondere Schlüsselstellen in den Punktwolken zu definieren und die anschließenden Berechnungen nur mit diesen durchzuführen. Diese Schlüsselstellen werden über eine Merkmalsbestimmung (siehe Abschnitt 6.2.2) definiert. Dieser Schritt ist optional und sollte besonders bei zeitkritischen Anwen-

dungen in Betracht gezogen werden.
Für die Projektgruppe entfällt der Schritt des Eingabefilters vollkommen, da bei dem Kinfu-Verfahren zum einen ein geringer Abstand zwischen den zu registrierenden Punktwolken existiert und zum anderen die Berechnungen parallel auf einer GPU durchgeführt werden und dadurch eine erhebliche Beschleunigung erreicht wird. Bei der Registrierung von Teilmodellen des Kinfu-Verfahrens ist die Qualität des Ergebnisses wichtiger als eine Verbesserung der Geschwindigkeit, wodurch eine Filterung nicht in Betracht gezogen wurde.

Berechung korrespondierender Punkte

Sei $d(\mathbf{p}^{(i)}, \mathcal{Z})$ die Distanz zwischen dem Punkt $\mathbf{p}^{(i)} \in \mathcal{S}$ und der Punktwolke $\mathcal{Z}$, dann gilt

$$d(\mathbf{p}^{(i)}, \mathcal{Z}) = \min_{\mathbf{p}^{(j)} \in \mathcal{Z}} d(\mathbf{p}^{(i)}, \mathbf{p}^{(j)}), \tag{6.34}$$

wobei $d(\mathbf{p}^{(i)}, \mathbf{p}^{(j)})$ dem euklidischem Abstand $\|\mathbf{p}^{(i)} - \mathbf{p}^{(j)}\|$ entspricht. Die Laufzeit für die Suche des nächstgelegenen Punktes in $\mathcal{Z}$ liegt im schlimmsten Fall bei $O(\mathcal{Z})$ bzw. $O(log\mathcal{Z})$ mit Hilfe eines k-d-Baumes. Wird die Berechnung für alle Punkte aus $\mathcal{S}$ durchgeführt, ergibt sich eine Laufzeit von $O(\mathcal{S} \cdot log\mathcal{Z})$. Ohne die Änderungen durch Zhang [78] einen k-d-Baum zu nutzen würde sich eine Laufzeit von $O(\mathcal{S} \cdot \mathcal{Z})$ ergeben. Gerade bei sehr großen Punktwolken ist dies nicht akzeptabel.

Transformation der Punkte

Mit diesen Punktkorrespondenzen kann die 3×3 Rotationsmatrix $\mathbf{R}$ sowie der Translationsvektor $\mathbf{t}$ berechnet werden (Details lassen sich in der Arbeit von Horn [38] finden). Die vollständige Registrierung ist durch $T(\mathbf{R}, \mathbf{t})$ gegeben.
Sei die Anzahl der Punkte in der Start- und Zielpunktwolke gleich ($\mathcal{S} = \mathcal{Z}$), sodass die Indizes der Punktkorrespondenzen übereinstimmen ($\mathbf{s}^{(i)}$ wird $\mathbf{z}^{(i)}$ zugeordnet). Dann ist die Funktion des mittleren quadratischen Abstandes

$$T(\mathbf{R}, \mathbf{t}) = \frac{1}{\mathcal{S}} \sum_{i=1}^{N} \|\mathbf{z}^{(i)} - \mathbf{R}\mathbf{s}^{(i)} - \mathbf{t}\|^2 \tag{6.35}$$

zu minimieren. Dies wird auch als Punkt-zu-Punkt-Metrik bezeichnet, da nur der euklidische Abstand zwischen zwei Punkten berücksichtigt wird. Alternativ kann auch die Punkt-zu-Ebene-Metrik genutzt werden. Hierbei wird mit dem Normalenvektor $\mathbf{n}^{(i)}$ die Tangente am korrespondierenden Punkt gebildet und der orthogonale

Abstand zu dieser berechnet. In diesem Fall muss

$$T(\mathbf{R}, \mathbf{t}) = \frac{1}{\mathcal{S}} \sum_{i=1}^{N} \|(\mathbf{z}^{(i)} - \mathbf{R}\mathbf{s}^{(i)} - \mathbf{t}) \cdot \mathbf{n}^{(i)}\|^2 \tag{6.36}$$

minimiert werden. Die Kreuz-Kovarianz-Matrix $\sum_{sz}$ der beiden Mengen $\mathcal{S}$ und $\mathcal{Z}$ lässt sich als

$$\sum\nolimits_{sz} = \frac{1}{\mathcal{S}} \sum_{i=1}^{S} [(\mathbf{s}^{(i)} - \mu_{\mathcal{Z}})(\mathbf{z}^{(i)} - \mu_{\mathcal{Z}})^t] = \frac{1}{\mathcal{S}} \sum_{i=1}^{S} [\mathbf{s}^{(i)} \mathbf{z}^{(i)^t}] - \mu_{\mathcal{S}} \mu_{\mathcal{Z}}^t \tag{6.37}$$

mit den Schwerpunkten $\mu_{\mathcal{S}}$ bzw. $\mu_{\mathcal{Z}}$ der Punktwolken $\mathcal{S}$ und $\mathcal{Z}$ darstellen. Mit Hilfe der zyklischen Komponenten der Matrix $A_{ij} = (\sum_{sz} - \sum_{sz}^T)_{ij}$ wird der Spaltenvektor $\mathbf{\Delta} = [A_{23} A_{31} A_{12}]^T$ und danach die symmetrische 4×4 Matrix

$$\mathbf{Q}(\sum\nolimits_{\mathbf{sz}}) = \begin{bmatrix} tr(\sum_{sz}) & \mathbf{\Delta}^T \\ \mathbf{\Delta} & \sum_{sz} + \sum_{sz}^T - tr(\sum_{sz}) \mathbf{I_3} \end{bmatrix} \tag{6.38}$$

bestimmt. Dabei ist $\mathbf{I_3}$ die 3×3 Einheitsmatrix. Die optimale Rotation $\mathbf{R} = (q_0, q_1, q_2, q_3)^t$ entspricht dem maximalen Eigenwert der Matrix $\mathbf{Q}$ und die daraus folgende Translation $\mathbf{t}$ ergibt sich aus:

$$\mathbf{t} = \mu_{\mathcal{Z}} - \mathbf{R}\mu_{\mathcal{S}}. \tag{6.39}$$

Die Startpunktwolke $\mathcal{S}$ wird abschließend mit $T(\mathbf{R}, \mathbf{t})$ aktualisiert. Dies wird solange wiederholt, bis die Differenz beider Punktwolken einen vorgegebenen Grenzwert unterschreitet. Zusätzlich kann der Prozess terminieren, wenn die maximale Anzahl an Iterationen erreicht wurde. Diese Werte sind zu Beginn festzulegen.
Im Kontext der Projektgruppe wird in der Posenschätzung des Kinfu-Verfahrens (siehe Abschnitt 6.1.4) die Punkt-zu-Ebene-Metrik verwendet, wobei dort Modifikationen vorgenommen werden, sodass parallele Berechnungen auf einer GPU möglich sind. Bei der Registrierung von Teilmodellen wird auf den Generalized Iterative Closest Point Algorithmus (GICP siehe Abschnitt 6.2.3) zurückgegriffen. In diesem wird eine spezielle Ebene-zu-Ebene-Metrik ausgenutzt um Oberflächeninformationen stärker in die Berechnung einfließen zu lassen.

Ausgabefilter

Falsche Punktkorrespondenzen können die Registrierung negativ beeinflussen, da z. B. gegen ein lokales Minimum konvergiert wird. Dies tritt auf, wenn gewisse Punk-

te sich in genau einer der beiden Punktwolken befinden und somit keine passende Punktkorrespondenz gefunden werden kann. Solche Punkte sollten herausgefiltert werden. Die einfachste Möglichkeit dafür ist es die maximale Distanz, die zwei Punkte zueinander aufweisen dürfen, festzulegen und alle Punkte, die diesen Grenzwert überschreiten, vollkommen aus der Berechnung auszuschließen. Der Grenzwert der maximalen Distanz kann fix oder alternativ als die durchschnittliche Distanz aller möglichen Punktkorrespondenzen gewählt werden.
Im Laufe der Projektgruppe hat sich gezeigt, dass eine fixe maximale Distanz von 20 Zentimetern angebracht ist, da durch die Hohe Bildrate der Tiefenkamera nur kleine Abstände zwischen korrespondierenden Punkten existieren. Der Registrierung von Teilmodellen ist eine Grobregistrierung (siehe Abschnitt 6.2.2) vorgeschaltet und garantiert, bei erfolgreich Durchführung, eine geringe Distanz zwischen den Teilmodellen.

Generalized Iterative Closest Point Algorithmus

Eine der entscheidenden Änderungen an dem ursprünglichen ICP-Algorithmus nehmen die Autoren (vergleiche die Arbeit von Segal u. a. [72]) vor. Sie modifizieren den Minimierungsschritt mit einem probabilistischen Modell, dabei werden die Oberflächeninformationen beider zu registrierender Punktwolken berücksichtigt (auch als Ebene-zu-Ebene-Metrik bezeichnet). Der Rest des ursprünglichen ICP-Verfahrens bleibt unverändert, damit die Geschwindigkeit der Suche nach korrespondierenden Punkten durch einen k-d-Baum genutzt werden kann.
Jedem Punkt $\mathbf{s}^{(i)} \in \mathcal{S}$ bzw. $\mathbf{z}^{(i)} \in \mathcal{Z}$ in den zu registrierenden Punktwolken wird eine Kovarianzmatrix $C^{(i)}$ zugeordnet. Dies basiert auf der Annahme, dass jeder Messpunkt zu einem realen Punkt der Oberfläche gehört und durch eine multivariate Normalverteilung repräsentiert werden kann. Die Fehlerfunktion

$$T = \underset{T}{\operatorname{argmin}} \sum_{i} d^{(i)^t} (C_{\mathcal{Z}}^{(i)} + T C_{\mathcal{S}}^{(i)} T^t)^{-1} d^{(i)^t} \tag{6.40}$$

ist das Ergebnis der Modifikation, wobei $d^{(i)} = \mathbf{s}^{(i)} - T\mathbf{z}^{(i)}$ und $C_{\mathcal{S}}^{(i)}$ bzw. $C_{\mathcal{Z}}^{(i)}$ die Kovarianzen der Punkte $\mathbf{s}^{(i)} \in \mathcal{S}$ bzw. $\mathbf{z}^{(i)} \in \mathcal{Z}$ sind. Alle Punkte ohne eine passende Korrespondenz sind zuvor herausgefiltert worden. Die Herleitung des ursprünglichen ICP-Algorithmus ist möglich indem die Kovarianzmatrizen mit den Werten $C_{\mathcal{S}}^{(i)} = 0$ und $C_{\mathcal{Z}}^{(i)} = \mathbf{I}$ belegt werden. Eine detaillierte Erklärung der Modifikation findet sich in der Arbeit von Segal u. a. [72].
Während der Projektgruppe bestätigte sich die Aussagen der Autoren, dass der

GICP-Algorithmus eine bessere Transformation liefert und dabei gleichzeitig robuster gegenüber der Parametrisierung ist. Eine wesentliche Verschlechterung der Laufzeit resultierte dabei nicht. Aus diesem Grund wir der GICP-Algorithmus für die Registrierung der Teilmodelle des Kinfu-Verfahrens eingesetzt. Ein Vergleich zwischen den Ergebnissen des ICP- und GICP-Algorithmus findet in der Evaluierung 9.2 statt.

6.2.4. Poisson Oberflächenrekonstruktion

Ziel der Projektgruppe ist die Erstellung eines möglichst genauen 3D-Modells der erkundeten Umgebung. Dazu muss mit den zuvor gewonnenen und registrierten Punktwolken eine Oberflächenrekonstruktion durchgeführt werden. Eine Vielzahl an verschiedenen Verfahren z.B.: Delaunay Triangulierung (vergleiche den Artikel von Boissonnat und Geiger [8]), Marching Cubes-Verfahren (siehe die Arbeit von Lorensen und Cline [50]) sowie die Poisson Oberflächenrekonstruktion wurden in der Literatur vorgestellt. Die Poisson Oberflächenrekonstruktion ist besonders geeignet, da sehr glatte Oberflächen entstehen und gleichzeitig das Rauschen, ein großes Problem der Aufnahmen der Tiefenkamera, in den Daten kompensiert wird.
Zuerst wird dabei eine Indikatorfunktion χ mit Hilfe der gerichteten Punkte $\mathcal{V}$ der Punktwolke $\mathcal{P}$ bestimmt. Anschließend kann eine passende Isofläche $\partial\mathcal{P}$ extrahiert werden. Die Abbildung 6.13 verdeutlicht den Ablauf.

Erzeugung des Vektorfeldes

Der Algorithmus verwendet einen adaptiven Octree $\mathcal{O}$ (siehe Kapitel 4.3.4). Dieser wird solange verfeinert, bis sich jeder Punkt in einem Knoten der Tiefe D befindet. Jedem Knoten k wird eine Funktion

$$F_k(\mathbf{p}) = F(\frac{\mathbf{p} - k.z}{k.b})\frac{1}{k.b^3} \tag{6.41}$$

zugeordnet. Der Term $k.z$ ist das Zentrum des Knotens k und $k.b$ die Breite. Anschließend erfolgt eine trilineare Interpolation, um die Daten auf die acht benachbarten Knoten zu verteilen und somit die Genauigkeit des Vektorfeldes zu verbessern. Nun lässt sich das Vektorfeld

$$V(\mathbf{p}) = \sum_{\mathbf{p}^{(i)} \in \mathcal{S}} \frac{1}{W_{\hat{D}}(\mathbf{p}^{(i)})} \sum_{k \in u_{t(\mathbf{p}^{(i)})}(\mathbf{p}^{(i)})} \alpha_{k,s} F_k(\mathbf{p}) \tag{6.42}$$

erzeugen. In dieser Definition des Vektorfeldes ist $t(\mathbf{p}^{(i)})$ die gewünschte Tiefe des

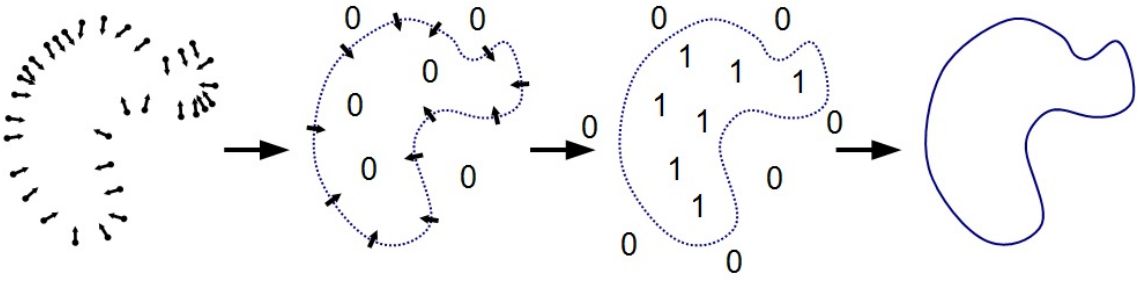


Abbildung 6.13.: Poisson Ablauf: aus dem Vektorfeld (links) wird ein Indikatorgradient $\nabla\chi$ und die Indikatorfunktion χ (mitte) sowie die Oberfläche (rechts) berechnet
Das Bild ist dem Paper *Poisson surface reconstruction* von Kazhdan et al. entnommen[46].

Punktes $\mathbf{p}^{(i)}$, $u_{t(\mathbf{p}^{(i)})}(\mathbf{p}^{(i)})$ die acht benachbarten Knoten mit gleicher Tiefe und $\alpha_{k,s}$ das Gewicht der trilinearen Interpolation. $W_{\hat{D}}(\mathbf{p}^{(i)})$ ist durch

$$W_{\hat{D}}(\mathbf{p}^{(i)}) = \sum_{\mathbf{p}^{(i)} \in \mathcal{S}} \sum_{k \in u_{t(\mathbf{p}^{(i)})}(\mathbf{p}^{(i)})} \alpha_{k,s} F_k(\mathbf{p}) \tag{6.43}$$

definiert. Dies ist genauer in der Arbeit von Kazhdan (et al.) [46] erklärt.

Lösung des Poissonproblems

Nachdem V vollständig definiert wurde, soll die Indikatorfunktion χ gelöst werden, sodass der Gradient von χ dem Vektorfeld V entspricht ($\Delta\chi = \nabla V$). Dies ist auch als Poisson-Problem bekannt. Die Berechnung ist sehr aufwändig, da die Funktionen F_k normalerweise keine orthogonale Basis bilden. Deshalb wird vereinfacht angenommen, dass

$$\sum_{k \in \mathcal{O}} \|\langle \Delta\chi - \nabla V, F_k \rangle\|^2 = \sum_{k \in \mathcal{O}} \|\langle \Delta\chi, F_k \rangle - \langle \nabla V, F_k \rangle\|^2 \tag{6.44}$$

mit $v^{(k)} = \langle \nabla V, F_k \rangle$ gilt. Sei $\chi = \sum_k \mathbf{x}^{(k)} F_k$, dann wird der Vektor $\mathbf{x} \in \mathbb{R}^{|O|}$ gesucht. Sei L eine $|\mathcal{O}| \times |\mathcal{O}|$ Matrix ist, wobei die Einträge (k, k') für alle $k, k' \in \mathcal{O}$ folgende Form

$$L_{k,k'} = \langle \frac{\partial^2 F_k}{\partial x^2}, F_{k'} \rangle + \langle \frac{\partial^2 F_k}{\partial y^2}, F_{k'} \rangle + \langle \frac{\partial^2 F_k}{\partial z^2}, F_{k'} \rangle \tag{6.45}$$

besitzen. Somit kann χ durch das Minimierungsproblem

$$\min_{\mathbf{x}\in\mathbb{R}^{|\mathcal{O}|}} \|Lx - v\|^2 \tag{6.46}$$

gelöst werden. Eine Möglichkeit besteht in der Verwendung des iterativen Gauss-Seidel-Verfahrens [20] für lineare Gleichungssysteme.

Extraktion der Isofläche

Um die angestrebte Oberflächenrekonstruktion

$$\partial\mathcal{P} = \{\mathbf{p} \in \mathbb{R}^3 | \chi(\mathbf{p}) = y\} \quad mit \quad y = \frac{\sum \frac{1}{W_{\hat{D}(\mathbf{p}^{(i)})}} \chi(\mathbf{p}^{(i)})}{\sum \frac{1}{W_{\hat{D}(\mathbf{p}^{(i)})}}} \tag{6.47}$$

zu erhalten, muss zuerst ein Isowert, der die Grenzen der Oberfläche definiert, bestimmt werden. Es wird hierfür der gewichtete Mittelwert y der Punkte (siehe Gl. 6.47) gewählt. Dies erlaubt eine gute Approximation der Oberfläche.
Die anschließende Extraktion der Isofläche findet mit einer Adaption des Marching Cubes Algorithmus [50] statt. Dabei werden oberflächennahe Knoten so aufgeteilt, dass pro Knoten nur genau ein Punkt existiert, welcher die Oberfläche berührt. Ein typisches Ergebnis einer Poisson Rekonstruktion ist in Abbildung 6.14 zu sehen. Hier wurde der Armadillo, eines der bekannten Stanford 3D-Modelle, verwendet. Die Abbildung zeigt die ursprüngliche Punktwolke (links), die Darstellung der Indikatorfunktion (mitte) und zuletzt das resultierende 3D-Modell (rechts).
Ein Problem der Poisson Oberflächenrekonstruktion ist, besonders bei großen Modellen, der Speicher- und Laufzeitbedarf. Eine vollständige Rekonstruktion eines großen Raumes auf einem herkömmlichen PC, das Ziel der Projektgruppe, ist nur durch sehr starke Reduktion und Filterung der Punktemengen (siehe Kapitel 6.2.1) möglich. Dadurch verringern sich jedoch die Details des Modells erheblich und machen das Ergebnis unbrauchbar. Aus diesem Grund wurde im Rahmen der Projektgruppe auf eine vollständige Rekonstruktion durch die Poisson Oberflächenrekonstruktion verzichtet und auf die Modelle des Kinfu-Verfahrens (siehe Kapitel 6.1) zurückgegriffen.

6.3. Zusammenfassung

Die Rekonstruktion umfasst den Teil der Projektgruppe, der ein dreidimensionales Modell der erkundeten Umgebung liefern soll. Gefordert ist dabei, dass das Verfah-

Abbildung 6.14.: Ergebnis einer Poisson Oberflächenrekonstruktion: Startpunktwolke (links), Indikatorfunktion χ (mitte) und 3d-Modell (rechts)
Die Abbildung ist aus dem Paper *Poisson surface reconstruction* von Kazhdan et al. [46].

ren ein möglichst detailreiches Modell berechnet, in dem sich auch feinere Strukturen wiedererkennen lassen. Gleichzeitig soll das Verfahren auch für große Umgebungen geeignet sein. Das etablierte KinectFusion-Verfahren (vergleiche Abschnitt 6.1) liefert sehr gute Ergebnisse und lässt sich prinzipiell auch auf große Räume anwenden, erfüllt daher alle Anforderungen der Projektgruppe.
Bei diesem Verfahren wird als Umgebungsrepräsentation eine truncated signed distance function (TSDF) verwendet. Dies ist ein Distanzfeld in dem für jeden Raumpunkt der Abstand zur nächstgelegenen Oberfläche des Modells gespeichert wird. Dabei ist die parallele Berechnung und Speicherung auf einer Grafikkarte möglich. Die Posenschätzung erfolgt mit einer modifizierten Variante des Iterative-Closest-Point-Algorithmus (ICP vergleiche Abschnitt 6.2.3). Zudem wird die Posenschätzung anhand des rekonstruierten Modells durchgeführt, wodurch der Drift, welcher bei einer paarweisen Bildregistrierung auftreten würde, vermieden wird. Außerdem wird durch die zeitliche Akkumulation im Modell das Rauschen der Sensordaten kompensiert. Ein Problem ist jedoch der limitierte Grafikkartenspeicher, wodurch die maximale Raumgröße vorgegeben ist. Bei einer Auflösung der TSDF von 1024^3 lässt sich ein Raum mit den Ausmaßen von $6m^3$ speichern und besitzt einen Speicherbedarf von 4 GB. Dies stößt an die Grenzen aktueller Grafikkarten und erfordert ggf. eine Aufteilung der Umgebung in mehrere TSDFs. Kinfu Large Scale löst dieses Problem indem äquidistant neue TSDFs angelegt werden und nur die aktuelle TSDF zur Berechnung verwendet wird. An den Grenzübergängen der einzelnen TSDFs ist dies jedoch problematisch, da oft zu wenig Punkte für eine gute Rekonstruktion, insbesondere für eine hinreichend gute Posenschätzung, zur Verfügung stehen. Moving Volume KinectFusion (siehe Abschnitt 6.1.7) hat dieses Problem nicht, da die Anordnung der TSDFs entlang der Kameratrajektorie erfolgt. Durch diese Modifikation

ist die Entstehung eines Driftes zwischen den TSDFs möglich, da keine globale Registrierung stattfindet. Im Rahmen der Projektgruppe ist diese Problematik nicht weiter behandelt worden, jedoch ist es möglich eine Loop-Closure, wie bereits im LSD-SLAM (siehe Abschnitt 4.3.1), einzubinden.
Die resultierende polygonale Repräsentation der TSDFs wird durch einen Marching-Cubes-Algorithmus generiert.
Eine Verbesserung der Robustheit der Posenschätzung ist mithilfe der Odometrie der Drohne möglich. Ist trotzdem keine erfolgreiche Posenschätzung möglich, entsteht eine Menge von Teilmodellen, wobei im schlimmsten Fall keine brauchbaren Informationen über deren Position zueinander existiert. Um eine möglichst genaues Modell der aufgenommenen Umgebung zu erhalten müssen diese Modelle erneut registriert werden. Hierfür findet eine Merkmalsbestimmung (vergleiche Abschnitt 6.2.2) in den zu registrierenden Teilmodellen statt. Mit Hilfe der berechneten Merkmalen wird eine Grobregistrierung (siehe Abschnitt 6.2.2) auf Basis eines RANSAC-Verfahrens durchgeführt, sodass eine Feinregistrierung durch den GICP-Algorithmus erfolgen kann. Die Grobregistrierung ist erforderlich, da die Möglichkeit besteht, dass der GICP-Algorithmus gegen ein lokales Minimum konvergiert und somit kein korrektes Ergebnis liefert. Der GICP-Algorithmus wurde dem ICP-Algorithmus auf Grund seiner Parameterrobustheit und Qualität der Registrierung vorgezogen. Beide Registrierungsschritte nutzen einen k-d-Baum, um die Suche nach korrespondieren Punkten zu beschleunigen.
Der ursprüngliche Ansatz der Projektgruppe für die anschließende Erstellung eines zusammenhängenden dreidimensionalen Modells aus den Teilmodellen war die Poisson-Oberflächenrekonstruktion vorgesehen, jedoch ist diese aufgrund von Speicherbeschränkungen bei großen Räumen nicht möglich. Eine Möglichkeit ein zusammenhängendes Modell zu erhalten ist die Reduzierung der Punkte z. B. durch den Voxelgrid-Filter (vergleiche Abschnitt 6.2.1), wodurch ein sehr detailarmes Modell entsteht. Dies genügt jedoch nicht der Anforderung eine möglichst genaue Repräsentation der erkundeten Umgebung zu erhalten, weshalb auf eine vollständige Rekonstruktion verzichtet wurde. Grundsätzlich wird dieses Problem durch die steigende Speicherkapazität neuer Hardware behoben. Außerdem könnten die beim Aufbau der Rekonstruktion aufgebauten TSDF-Volumen gespeichert und iterativ ein konsistentes Polygonnetz aus der Abfolge der Volumen gewonnen werden [41].

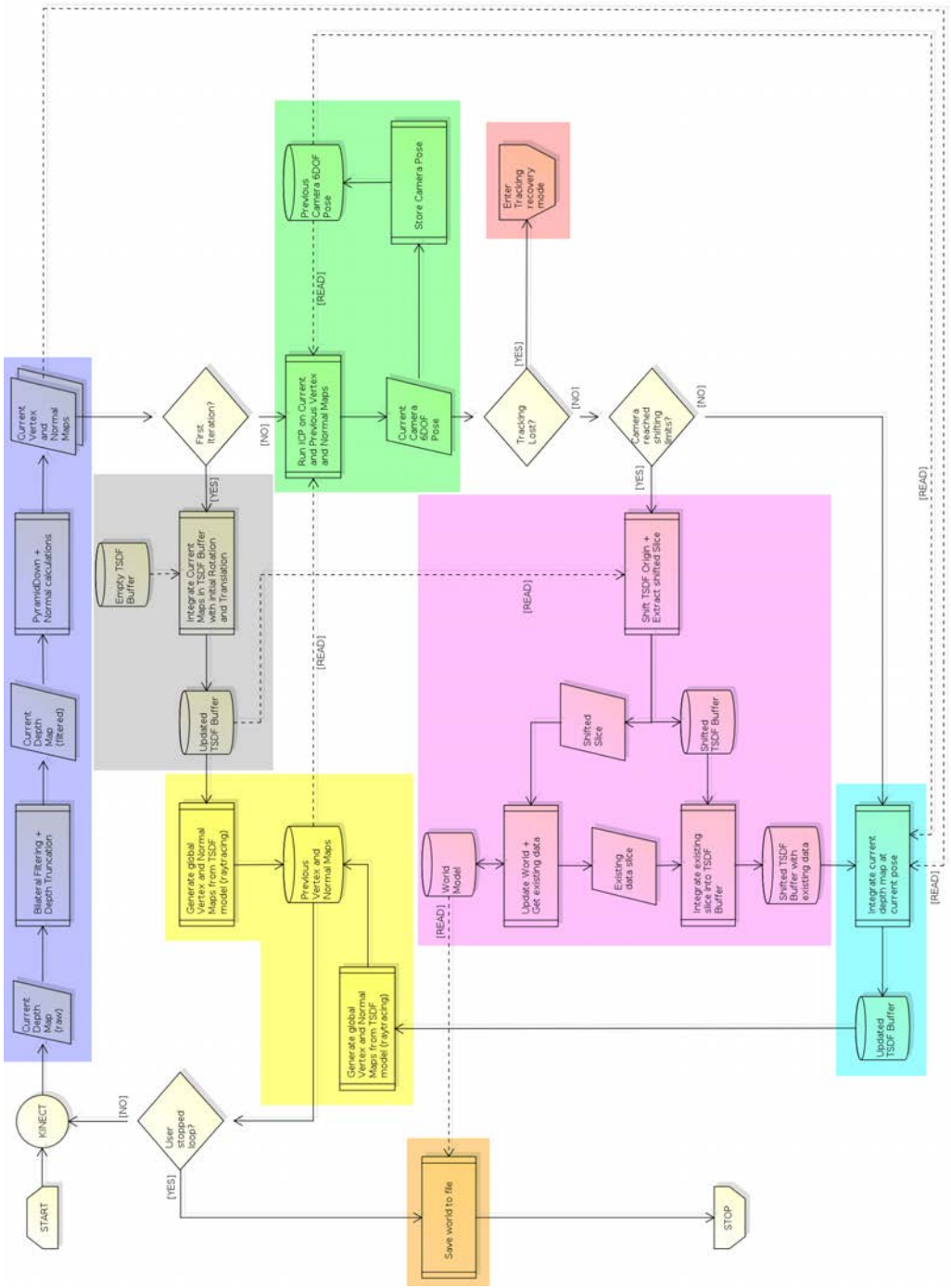


Abbildung 6.15.: Das erweiterte KinectFusion-Verfahren in Form eines Aktivitätsdiagramm. Die farbigen Markierungen stellen die einzelnen Komponenten dar: Vorverarbeitung (dunkelblau, siehe Abschnitt 6.1.1), Posenschätzung (grün, siehe Abschnitt 6.1.4), Fusion mit der Szene (türkis, siehe Abschnitt 6.1.5), Tiefenvorhersage (gelb, siehe Abschnitt 6.1.3), Initialisierung der diskreten TSDF (grau), Notfallbehandlung bei Verlust des Tracking bzw. zu ungenauer Posenschätzung (rot), Erweiterung von Kinfu um mehrere TSDF-Volumen (pink, siehe Abschnitt 6.1.7), Transformation in polygonale Repräsentation (orange, siehe Abschnitt 6.1.6) (Grundlage der Grafik entnommen aus einem Blogbeitrag von Heredia und Favier [28])

7. Adaptive Steuerungs- und Akquisitionskonzepte

7.1. Einleitung

Zur Realisierung der Erkundung wurde als Agent eine Flugdrohne gewählt, da sich diese den besten Überblick über große, dreidimensionale Räume verschaffen kann, indem sie eine fast beliebige Position im Raum einnimmt. Andere Drohnen, die bspw. auf dem Boden fahren, sind zwar leichter zu steuern, können jedoch an Hindernissen und hohen Decken scheitern und nicht von oben auf größere Objekte, wie Tische oder Schränke, blicken.
Konkret fiel die Wahl dabei auf die beiden Quadrocopter AR.Drone2 und Bebop Drone der Firma Parrot, da diese die notwendigen Schnittstellen (Frontkamera, WLAN, USB) bereitstellen, klein genug sind, die notwendige Tragkraft für zusätzliche Sensoren bieten und bereits zur Verfügung standen. Da die Software dieser Quadrocopter nicht für den hier vorliegenden Anwendungsfall entwickelt wurde, sondern vor allem als Fernbedienung dienen soll, war eine Neuimplementierung der grundlegenden Firmware-Komponenten erforderlich. Die zwei wichtigsten Kritikpunkte an der mitgelieferten Software bestehen darin, dass Sensordaten (vergleiche Abschnitt 3.3) grundsätzlich nicht zeitlich zugeordnet werden können, was eine autonome Navigation deutlich erschwert, und dass der Betrieb des Tiefensensors im Flug nicht möglich ist.
Im Folgenden werden die wichtigsten Komponenten der entwickelten Firmware erläutert.

7.2. Steuerung über die offizielle Schnittstelle

Die Firma Parrot stellt mit dem ARDroneSDK3 (ARSDK)[61] eine Softwareschnittstelle bereit, welche es ermöglicht, die von diesem Unternehmen entwickelten Drohnen zu steuern. Dieses wurde innerhalb des Projektes eingesetzt, um die Drohne zu

Testzwecken mittels der Pfadplanung und des Trackings zu steuern, ohne weitere, mögliche Fehlerquellen, wie eine eigene Firmware, einzuschließen.
Abbildung 7.1 zeigt die Struktur der Testanwendung, welche von der Struktur der Endanwendung abweicht (vergleiche Abbildung 2.2). Die Komponenten Pfadplanung und Tracking senden ihre Befehle zu einer neu eingeführten Steuerungskomponente, welche diese mit dem von der Schnittstelle bereitgestellten Befehlssatz ausführt. Dieser wird in Abschnitt 7.2.1 beschrieben. Ebenso empfängt die Steuerungskomponente den Video-Stream sowie Statusänderungen der Drohne. Die empfangenen Bilder werden dekodiert und dem Tracking zur Verfügung gestellt. Aus den Statusänderungen, welche sowohl den Flugstatus, als auch die aktuelle Geschwindigkeit der Drohne umfassen, lassen sich die Position sowie die Orientierung ermitteln. Abschnitt 7.2.2 beschreibt das Vorgehen der Odometrie.

7.2.1. Befehlssatz zur Steuerung

Das ARSDK stellt eine Menge von Befehlen bereit, von denen an dieser Stelle nur diejenigen beschrieben werden, welche im Projekt genutzt werden.

Abheben Sobald sich der Client mittels der Schnittstelle mit der Drohne verbunden hat, kann diese per Befehl abheben. Nachdem sie den Befehl erhalten hat, bewegt sie sich standardmäßig in eine Höhe von einem Meter und versucht diese Position zu halten. Der verbundene Client erhält eine Statusänderung der Drohne, welche ihm mitteilt, wenn sie die entsprechende Höhe erreicht hat.

Bewegung Die Drohne bewegt sich, indem sie die Neigung um einen der zwei Winkel Roll oder Pitch vergrößert. Die Neigung der entsprechenden Achsen kann über das ARSDK durch Angabe eines Wertes im Bereich -100 bis 100 bestimmt werden. Der Wert gibt dabei die zu erreichende Geschwindigkeit in Prozent der Maximalgeschwindigkeit an, welche ebenso über die Schnittstelle bestimmt werden kann. Nachdem die Neigung von einer oder mehreren Achsen an die Drohne gesendet wurde, behält sie diese so lange bei, bis sie einen anderen Befehl erhält.

Algorithmus 7.1 zeigt beispielhaft, wie die Drohne um 2,5 m nach vorne bewegt werden kann, wenn die Maximalgeschwindigkeit 0,5 m/s beträgt. Dafür neigt sich die Drohne so um die Pitch-Achse, dass sie sich mit 50 % der Maximalgeschwindigkeit bewegt. Diese Bewegung muss sie für 10 Sekunden halten, bevor sie gestoppt wird.

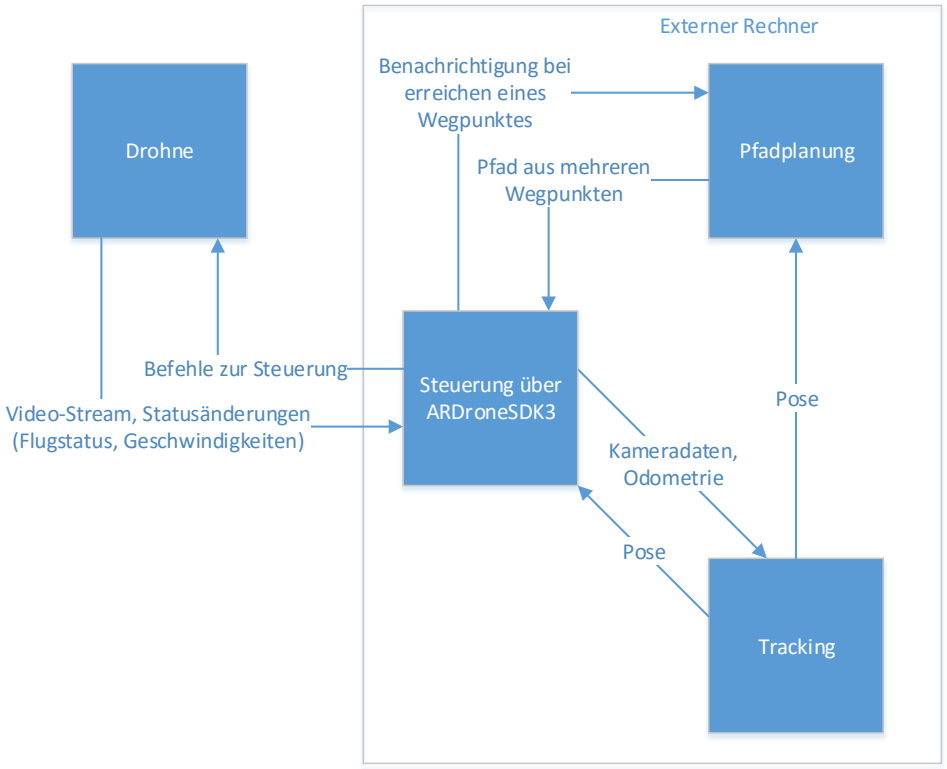


Abbildung 7.1.: Anwendungsstruktur mit Drohnen-Steuerung über die Schnittstelle von Parrot.

1: PITCH(50)
2: WAIT(10)
3: PITCH(0)

Algorithmus 7.1: Bewegung der Drohne um 2,5 m, bei einer Maximalgeschwindigkeit von 0,5 m/s.

Orientierung Das Verfahren zur Änderung der Orientierung gleicht demjenigen, zur Bewegung der Drohne: Der Drohne wird eine Dreh-Geschwindigkeit mitgeteilt, welche sie so lange annimmt, bis diese geändert wird. Um die Drehung zu stoppen, muss der Wert entsprechend auf Null gesetzt werden.

Landen Nachdem sie den Befehl erhalten hat, landet die Drohne selbstständig und sendet eine Statusänderung, welche den verbundenen Client über die Landung informiert. Der Landevorgang wird von der Drohne so ausgeführt, dass sie sanft auf dem Boden landet, um die interne Hardware nicht zu beschädigen.

7.2.2. Odometrie

Das ARSDK unterstützt die Schätzung von Position aus Ausrichtung, indem der verbundene Client über Änderungen von Geschwindigkeiten in Metern pro Sekunde sowie Rotationen der Drohne benachrichtigt wird. Änderungen der Geschwindigkeit werden pro Achse im globalen North East Down (NED)-Koordinatensystem übertragen, wobei positive Werte auf der N-Achse für eine Vorwärtsbewegung, auf der E-Achse für eine Bewegung nach Rechts und auf der D-Achse für Bewegung nach unten stehen. Unter der Annahme, dass die Drohne von der Position $(0, 0, 0)$ startet, kann bei jeder Geschwindigkeitsänderung die Position der Drohne geschätzt werden. Dafür wird die Geschwindigkeit v in Metern pro Sekunde sowie der Zeitpunkt t in Millisekunden bei einer Änderung gespeichert. Bei der nächsten Korrektur der Geschwindigkeit zu Zeitpunkt t' kann die Bewegung in Zentimetern durch $h = \frac{t'-t}{10} \cdot v$ bestimmt werden. Die Bewegung h kann damit für jede Achse bei einer Geschwindigkeitsänderung berechnet werden und ergibt einen Bewegungsvektor (h_x, h_y, h_z), welcher, in Addition mit der aktuellen Position, die geschätzte Position liefert. In der Praxis hat sich gezeigt, dass die Drohne sehr oft Änderungen der Geschwindigkeit feststellt und an den Client übermittelt, so dass die Schätzung entsprechend oft durchgeführt wird.

Die Rotation der Drohne wird mit Hilfe der vom ARSDK übertragenen Lage-Änderungen geschätzt. Die Schnittstelle informiert den Client über jede Änderung der Fluglage der drei Achsen, wobei für die Rotationsbestimmung nur die Werte des Yaw-Winkels interessant sind.

7.3. Ansteuerung der Motoren

Sowohl in der Parrot AR.Drone2 als auch in der Parrot Bebop Drone ist die Regelung der Motoren unabhängig vom SoC durch einen eigenen Microcontroller (MCU) realisiert. In der AR.Drone2 übernimmt diese Aufgabe das Motorboard, in der Bebop Drone das Parrot BLDC, welches ebenfalls eine eigenständige MCU auf Basis eines Cypress Motor Controllers darstellt. Die Firmware dieser beiden MCU ist durch den Hersteller vorgegeben und kann unverändert genutzt werden, d. h. in beiden Fällen beschränkt sich die Ansteuerung der Motoren auf das Senden von Befehlen an den zuständigen MCU.

7.3.1. Parrot AR.Drone2

In der AR.Drone2 wird zur Kommunikation mit dem Motorboard asynchrone serielle Kommunikation eingesetzt, über die das SoC gemäß eines von Parrot festgelegten Protokolls Pakete an den MCU sendet. Der Aufbau eines Pakets ist in Tabelle 7.1 dargestellt. Hierbei wird durch `cmd = 1` festgelegt, dass Sollwerte für die Rotordrehzahlen vorgegeben werden sollen. In den Feldern `front-left`, `front-right`, `rear-left` und `rear-right` wird ein Wert für die gewünschte Drehzahl des jeweiligen Rotors als 9-Bit Ganzzahl im Intervall $[0; 511]$ angegeben, die das Verhältnis zur maximal möglichen Drehzahl angibt.

7.3.2. Parrot Bebop Drone

In der Bebop Drone wird zur Kommunikation mit dem Motorboard Inter-Integrated Circuit (I^2C) eingesetzt. Auch hier sendet das SoC gemäß eines von Parrot festgelegten Protokolls Pakete an den MCU, wobei das Protokoll deutlich umfangreicher ist.

7.4. Sensordaten-Erfassung

Der Zugriff auf die Sensoren erfolgt unterschiedlich. Während in der Parrot AR.Drone2 alle Sensoren mit einem eigenen MCU, dem „Navboard“, verbunden sind, bedient das SoC der Parrot Bebop Drone die Sensoren direkt über I^2C und andere Schnittstellen.

Parrot AR.Drone2

In der AR.Drone2 gelangt das SoC durch Auslesen einer asynchronen seriellen Schnittstelle an alle relevanten Sensordaten, die vom Navboard in regelmäßigen Ab-

0	1	2	3	4	5	6	7
front-left[4:8]					cmd = 1		
front-right[5:8]					front-left[0:3]		
rear-right[6:8]					front-right[0:4]		
rear-left[7:8]					rear-right[0:5]		
0	rear-left[0:6]						

Tabelle 7.1.: Format eines Datenpakets an das Motorboard der AR.Drone2

ständen (200Hz) übertragen werden. Der Aufbau eines Sensordaten-Pakets ist in Tabelle 7.2 dargestellt. Jedes Paket beginnt mit einer Längenangabe („length“), die 56 betragen muss und daher auch zur Erkennung eines möglichen Paketanfangs dienen kann. Für den Beschleunigungssensor („accelerometer“), das Gyroskop („ gyroscope“) und das Magnetometer („magnetometer“) werden jeweils drei skalierte ganzzahlige Werte übertragen. Für den Höhensensor („ultrasound“) werden eine ganze Reihe von Messwerten und Parametern geliefert, von denen „ultrasound“ den berechneten Abstand zum Boden unterhalb der Drohne enthält. Das Paket schließt mit einer Prüfsumme („checksum“) ab, die durch Addition der einzelnen Bytes des Pakets berechnet wird.

Parrot Bebop Drone

Bei der Bebop Drone erfordert die Erfassung der Sensordaten mehr Aufwand, da die Sensoren direkt mit dem SoC verbunden sind. Dabei wird vor allem I^2C eingesetzt, über die der jeweilige Treiber mit dem Sensor (IMU oder Magnetometer) kommuniziert. Die für die Bebop Drone benötigten Treiber fanden sich aber bereits in anderen Drohnen-Projekten.
Einen Sonderfall stellt der Ultraschall-Sensor dar. Dieser wird direkt durch einen digitalen Impuls gesteuert und über einen Analog-Digital-Wandler ausgewertet. Dabei kommt eine Kombination aus Serial Peripheral Interface (SPI) und der Linux Industrial I/O (IIO)-Schnittstelle zum Einsatz.

7.5. Schnittstellen zu den Bildsensoren

Dieser Abschnitt beschreibt zwei Schnittstellen zum Auslesen von Bilddaten der Drohne.

7.5.1. Video for Linux Two

Die V4L2-API dient unter Linux als Schnittstelle zu meist bildorientierten Multimediakomponenten des Systems, die von einem Treiber-Modul im Betriebssystem-Kernel kontrolliert werden. Dabei grenzt sie sich von anderen Schnittstellen zur Bildausgabe (Grafiktreibern) dadurch ab, dass sie auf Streaming fokussiert ist. Nach eigenen Angaben ist mit der V4L2-API u.a. der Zugriff auf Video- und Radio-Streaming-Geräte (dies beinhaltet auch Kameras), analoge und digitale TV-Empfänger, Radio-Empfänger, Codecs und Fernbedienungen möglich [16]. Sie ist daher für dieses Pro-

jekt von großer Bedeutung, da sie den Zugriff auf die Kameras der Drohnen ermöglicht.
Zusätzlich beinhaltet die Schnittstelle in neueren Linux-Kerneln die „media structure API“, mit der Untergeräte einer Multimediakomponente konfiguriert und miteinander verbunden werden können. Auf diese Weise wird bspw. in der Parrot Bebop Drone eine Umsetzung des Pixelformats der Frontkamera in Hardware realisiert, damit die aufgenommenen Bilder weiterverarbeitet werden können.

7.5.2. OpenNI2

Open Natural Interaction ist ein Standard Framework für 3D Sensoren. OpenNI2 wird von Occipital[1] weiterentwickelt und aktualisiert. OpenNI ist notwendig für die Ansteuerung des Structure Sensors, der an die Drohne angebunden werden soll und ebenfalls von dem Unternehmen Occipital entwickelt wurde. Zur Erreichung des Zieles die 3D Daten mit Hilfe der Drohne aus dem Sensor auszulesen und anschließend zu übertragen, wird ein eigener OpenNI Treiber erzeugt, der von OpenNI verwendet werden kann. Auf diese Weise kann der Treiber an beliebiger Stelle und mit allen OpenNI Version ab Version 2 verwendet werden. Zur Implementierung des Treibers müssen Unterklassen von `DriverBase`, `DeviceBase` sowie `StreamBase` angelegt werden. `DriverBase` dient der allgemeinen Anbindung an die Hardware. Zu überschreibende Funktionen sind hierbei `deviceOpen`, `deviceClose` und `tryDevice`. `tryDevice` versucht für eine gegebene URL ein Objekt der Klasse `DeviceBase` zu erstellen und gibt dieses zurück. Für die Anwendung in diesem Projekt wird auf die Methode der Erstellung eines Devices zurückgegriffen, da, im Gegensatz zu anderen herkömmlichen Treibern, kein Hardware Interrupt bei der Anbindung eines neuen Sensors erzeugt wird. Es wird auf die angeben URL gelauscht und geprüft, ob es sich um einen aktive RTSP Server handelt. Anschließend wird das Device erzeugt und zurück gegeben. Mit `deviceOpen` öffnet der OpenNI Kontext das Gerät und kann es nach Verwendung mit `deviceClose` schließen. In der `DeviceBase` Klasse wird hinterlegt, welche möglichen Werte (Farbbilder, Tiefenwerte) der Sensor ausgeben kann und welche Auflösung und Framerate diese haben. Mit `createStream` wird für einen eingegeben Sensortypen ein `StreamBase` Objekt erstellt und zurückgegeben. Von diesem `StreamBase` können einzelne Frames gelesen werden. Hier ist es notwendig mit Hilfe der Live555 Bibliothek für ein Auslesen des RTSP Streams zu sorgen und die Frames in das Struct `OniFrame` zu überführen. Um die zusätzlichen Informationen über die Odometrie zu jedem Frame herausgeben zu können, wird

[1] `http://structure.io/` Abruf: 19.06.16 17:00

das Struct um sieben Werte für die Postion und Ausrichtung erweitert.

7.6. Weiterverarbeitung der Bilddaten

Die Kameras der Drohnen liefern ihre Bilder als Bitmaps an das SoC. Um das so gewonnene Bildmaterial zum Tracking und zur Rekonstruktion weitergeben zu können, muss die Datenmenge durch Kompression in Form einer Video-Kodierung soweit verringert werden, dass eine Echtzeit-Übertragung über die WLAN-Verbindung möglich ist.

7.6.1. Videoformate

Für die Video-Kodierung stehen mehrere weit verbreitete Standards zur Verfügung:

Motion-JPEG ist eine zwar nicht standardisierte, aber übliche Vorgehensweise, um ein Video über ein Netzwerk übertragen zu können. Eine indirekte Festlegung besteht durch den in [53] erläuterten JPEG-Standard und die in [4] vorgenommene Einschränkung des JPEG-Standards und Definition eines Protokolls. Motion-JPEG-Kodierung weist die schlechteste Bildqualität auf, ist dafür aber sehr ressourcenschonend und kann auch durch schwache CPUs in Echtzeit ausgeführt werden.

MPEG-4 part 2 [42] ist die ursprüngliche Videokodierung des MPEG-4-Standards. Diese Kodierung ist Motion-JPEG weit überlegen und wird von beiden Drohnen unterstützt, wurde aber bereits durch Advanced Video Coding (AVC) ersetzt. Da AVC aber aus lizenzrechtlichen Gründen nicht immer zur Verfügung steht, verwendet die Firmware-Implementierung dieses Projekts MPEG-4 part 2.

AVC (MPEG-4 part 10) ist eine Erweiterung zu MPEG-4, die platz-effizienter kodiert, d. h. die bei gleicher Bitrate die Bildqualität erhöht. Parrot verwendet diese Kodierung standardmäßig für beide Drohnen.

7.6.2. Video-Codecs

Um die im vorangegangen Abschnitt erwähnten Standards einzusetzen, ist ein dazugehöriger Video-Codec erforderlich, der die einzelnen von einer der Drohnenkameras aufgenommenen Bilder im entsprechenden Format kodiert. Video-Codecs existieren sowohl als Software-Implementierung als auch als Hardware-Komponente.

Um Motion-JPEG zu erzeugen, genügt bereits eine Bibliothek, die allgemeine JPEG-Bilder erzeugen kann. Hier fiel die Wahl auf libjpeg-turbo[2], da diese Variante der Standard-Bibliothek libjpeg der Independent JPEG Group (IJG) die NEON-Befehlssatzerweiterung des ARM-Prozessors beider Drohnen unterstützt und so deutlich effizienter arbeitet.
Für MPEG-4 part 2 wurde der XviD[3] Software-MPEG-4-Codec eingesetzt, da dieser zu den schnellsten (am wenigsten die CPU belastenden) MPEG-4-Codecs gehört. Mit diesem Codec kann der Video-Teil der Firmware auf einem gewöhnlichen PC das Bild einer Webcam in Echtzeit kodieren und übertragen. Die CPUs beider Drohnen sind jedoch zu schwach, um MPEG-4 part 2 durch XviD in Software kodieren zu können, d. h. die Drohnen würden nur einen kleinen Teil der Bilder, welche die Kameras pro Sekunde liefern, kodieren können. Hier wurde vom Hersteller stattdessen eine der im folgenden beschriebenen Hardware-Lösungen eingesetzt.

Parrot AR.Drone2

In der Parrot AR.Drone2 wird die Video-Kodierung von einem Digital Signal Processor (DSP) übernommen, der parallel zur CPU arbeitet. Dies ist streng genommen ein Software-Codec. Allerdings wird der DSP hier ausschliesslich für die Videokodierung eingesetzt und steht dem Betriebssystem nur indirekt zur Verfügung, d. h. er kann nicht eingesetzt werden, um allgemein Programme des Betriebssystems direkt auszuführen.
Der Hersteller des SoC, Texas Instruments, bietet fertige Codecs für verschiedene Video-Kodierungen an, darunter MPEG-4 part 2 und AVC. Diese können über einen im Linux-Kernel der AR.Drone2 enthaltenen Treiber geladen und so vom DSP ausgeführt werden.

Parrot Bebop Drone

Die Parrot Bebop Drone verfügt über eine Video Processing Unit (VPU), die in das SoC integriert ist und ausschließlich für das Kodieren und Dekodieren von Videos zuständig ist. Die konkrete VPU der Bebop Drone ist ein Hantro 6280/7280/8270, der diese Aufgabe vollkommen unabhängig von der CPU erledigt. Der für diese Komponente benötigte Treiber besteht aus einem Stub im Kernel, der einem im Userspace laufenden Programm den direkten Zugriff auf die Hardware ermöglicht, und dem eigentlichen Treiber, der im Programm selbst läuft.

[2] `http://libjpeg-turbo.virtualgl.org/` Abruf: 19.06.16 17:00
[3] `https://labs.xvid.com/` Abruf: 19.06.16 17:00

Da der Hersteller Hantro den Treiber für die VPU nicht zugänglich macht, stellt sich die Verwendung der VPU als äußerst schwierig dar und ist bisher noch nicht gelungen.

7.6.3. Einbetten der Odometrie

Um die in der Einleitung erwähnten Zeitstempel und die von der Drohne zum Zeitpunkt der Bildaufnahme eingenommene Pose als Zusatzinformation für das Tracking und die Rekonstruktion zur Verfügung zu stellen, werden diese so gut wie möglich in die Videostreams eingebettet. Dies hat gegenüber einer separaten Übertragung die zwei entscheidenden Vorteile, dass die Daten dem jeweiligen Bild sicher zugeordnet werden können und auch beim Abspielen eines gespeicherten Videos wieder zur Verfügung stehen.
Motion-JPEG und MPEG-4 part 2 bieten hierfür Möglichkeiten, die im nachfolgenden beschrieben werden. Unabhängig von der verwendeten Video-Kodierung haben die Zusatzinformationen immer folgenden Aufbau:

Zeitstempel Der Zeitpunkt, an dem das Bild aufgenommen wurde. Die Einheit ist Mikrosekunden.

Translation Der Ort der Aufnahme im NED-System, bestehend aus der x, y und z-Koordinate. Die Einheit ist Millimeter.

Rotation Die Rotation als Quarternion, bestehend aus re, i, j und k.

Dabei wird eine einzelne Textzeile erzeugt, die die genannten Werte durch Kommata getrennt enthält (dies entspricht einer Zeile einer CSV-Datei).

Motion-JPEG

Bei Motion-JPEG ist das ursprüngliche JPEG-Bild üblicherweise reduziert auf einen einzigen Entropie-„Scan“, der das gesamte Bild enthält. Der Kopf der Datei wird weggelassen und durch einige Bytes ersetzt, die nur die nötigsten Parameter beschreiben, wodurch bei der Kodierung vorgegebene Standardparameter verwendet werden müssen. Das Ende der Datei, bestehend aus dem End-of-Image-Marker, ist optional. Wird dieser Marker weggelassen, ist die Länge des Scans implizit durch seinen Inhalt bestimmt und dem Scan dürfen ausdrücklich eine beliebige Anzahl willkürlicher Bytes folgen, um die Kompatibilität mit bestimmten Hardware-Kodierern zu wahren. Die Firmware macht sich dies zunutze, indem sie an den Scan einen Kommentar-

Marker anhängt (`COM`, Byte-Sequenz `FFh FEh`). In diesem Kommentar befindet sich die zu dem Bild gehörende Pose und der Zeitstempel.

MPEG-4 part 2

Bei MPEG-4-Videos wird ein sog. „Elementary Stream“ übertragen, der aus einer nahtlosen Aneinanderreihung einzeln kodierter Bilder (Frames) besteht. Bilder werden dabei als sog. I-, P- oder B-Frame dargestellt: I-Frames sind vollständige Bilder, die unabhängig dekodiert werden. P- und B-Frames beinhalten nur die Differenz zu einem oder zwei anderen Bildern, wodurch sie besonders effizient kodiert, jedoch auch nur mithilfe der anderen Bilder dekodiert werden können. Syntaktisch bestehen alle Frames aus einem oder mehreren Objekten, die mit einem sog. „Startcode“ (Byte-Sequenz `0h 0h 1h` bei MPEG-4 part 2) anfangen und sich bis zum nächsten Startcode erstrecken.
MPEG-4 part 2 erlaubt das Hinzufügen beliebiger Daten an bestimmten, dafür vorgesehenen Stellen. Diese Daten werden in sog. „UserData“-Objekte verpackt, die einen nahezu beliebigen Inhalt haben dürfen und auch in ihrer Länge nicht eingeschränkt sind. Die einzige Einschränkung besteht darin, dass die Byte-Sequenz eines Startcodes nicht enthalten sein darf, da diese als echter Startcode fehlinterpretiert würde. Da die von der Firmware eingebetteten Daten aber ASCII-Text sind, kann ein Startcode nicht versehentlich erzeugt werden.
Für UserData vorgesehene Stellen befinden sich nur vor I-Frames, daher enthält das UserData-Objekt der Firmware immer alle Posen der Bilder, die auf den letzten I-Frame folgen, und die Pose des unmittelbar nachfolgenden I-Frames. Die Firmware hinterlegt ihr UserData-Objekt dabei im „Group of Video Object Planes“-Objekt und trennt die einzelnen Posen durch Zeilenumbrüche (dies entspricht mehreren Zeilen einer CSV-Datei). Die Posen werden dabei in der Reihenfolge abgelegt, in der die Bilder beim Abspielen dargestellt werden (im MPEG-Standard auch als „composition order“ bezeichnet).

7.6.4. Kompression von Tiefenbildern

Da die Tiefenbilder für die Rekonstruktion aufgezeichnet werden müssen, ist eine Kompression erforderlich, um die Bilder über das Netzwerk der Drohne übertragen zu können (eine lokale Speicherung ist nicht immer möglich, da die Parrot AR.Drone2 nur über 64MB Flashspeicher verfügt). Dazu wurde inspiriert durch den bestehenden Algorithmus in OpenNI2 und andere, gängige Bildformate ein eigener

Algorithmus entworfen, der schnell genug auf der Drohne ausgeführt werden kann, die Tiefenbilder ausreichend komprimiert und dabei verlustfrei arbeitet.

Minimierung des Kontrastumfangs Die eingesetzte 3D-Kamera verwendet nur etwa 1000 Werte im 16-Bit breiten Wertebereich der Pixel. Daher werden die Werte der Pixel durch ihren Rang in der Liste aller bisher vorgekommenen Pixel ersetzt. Durch diese verlustfreie Reduktion des Kontrastumfangs wird die Differenz zwischen benachbarten Pixeln deutlich verringert.

Differenzberechnung benachbarter Pixel Da benachbarte Pixel in einem Bild oft fast den gleichen Wert haben, werden alle Pixel durch die Differenz zum vorangegangene Pixel ersetzt, d. h. $P'_{i,j} = P_{i,j} - P_{i,j-1} \forall j > 1$ und $P'_{i,1} = P_{i,1} - P_{i-1,n}$. Lediglich der allereste Pixel $P_{1,1}$ bleibt als absoluter Startwert erhalten.

Entropiekodierung Abschließend wird das nur noch aus Pixeldifferenzen bestehende Bild mittels Huffman-Kodierung[40] von Redundanz befreit und dadurch auf einen Bruchteil seiner Größe reduziert. Dabei wird die Häufigkeitsverteilung der Symbole (Bytes) des vorangegangenen Bildes benutzt, damit der Algorithmus nur ein einziges Mal über ein Tiefenbild iteriert.

7.7. Kommunikation mit der Drohne

Zur Kommunikation mit der Drohne kommen mehrere standardisierte Protokolle zum Einsatz. Im Wesentlichen werden damit die zwei Anwendungsfälle „Senden von Befehlen“ (Abschnitt 7.7.3) und „Empfangen von Sensordaten“ (Abschnitt 7.7.1) umgesetzt.

7.7.1. RTSP/RTP

Um die von den Kameras der Drohne aufgenommenen Bilder an das Tracking und die Rekonstruktion zu übertragen, wird das für die Übertragung von Videostreams konzipierte Real-Time Transport Protocol (RTP) in Kombination mit dem Real-Time Streaming Protocol (RTSP) eingesetzt. Dabei dient RTSP zum Aushandeln der Stream-Eigenschaften und zur Steuerung, während mittels RTP der eigentliche Videostream übertragen wird. Die Firmware setzt hierfür die live555-Bibliothek[4] ein, die beide Protokolle für gängige Video-Kodierungen implementiert, darunter Motion-JPEG, MPEG-4 part 2 und AVC.

[4] `http://www.live555.com/liveMedia/` Abruf: 19.06.16 16:00

7.7.2. TCP-Socket für Tiefenvideo

Für die Übertragung des von der 3D-Kamera erzeugten Tiefenvideos wird ein Transmission Control Protocol (TCP)-Socket eingesetzt, um eine zuverlässige Übertragung der Bilder zu gewährleisten und damit sicherzustellen, dass alle Bilder für die Rekonstruktion aufgezeichnet werden können. Dabei wird ein sehr schlichtes proprietäres Protokoll eingesetzt, bei dem die Bilder nur durch einen Header voneinander abgegrenzt versendet werden. Ein mit der Drohne verbundener Client empfängt die Bilder ohne weitere Nachfrage allein dadurch, dass er eine TCP-Verbindung zur Drohne aufbaut. Gegenüber RTP hat dieses Protokoll zwar den Nachteil, dass es nicht Multicast-fähig ist, aber RTP kann die Anforderung an die Zuverlässigkeit nicht erfüllen, da es für Echtzeitübertragungen konzipiert wurde und daher der Schwerpunkt auf der zeitnahen Übertragung liegt, bei der veraltete Daten verworfen werden sollen. Der Aufbau des Protokolls ist in Abbildung 7.2 dargestellt. Die dabei eingesetzte Kompression ist in Abschnitt 7.6.4 beschrieben.

7.7.3. JSON

JavaScript Object Notation (JSON) ist ein leichtgewichtiges und textbasiertes Übertragungsformat [74]. Im Vergleich zu XML erzeugt JSON einen geringen Overhead. Zudem wird die Menschenlesbarkeit erhalten. JSON-Zeichenfolgen bestehen im Wesentlichen aus Arrays, Objekten und Daten in Form von Schlüssel-Wert-Paaren. Die primitiven Werte sind dabei Zahlen, Zeichenketten, Wahrheitswerte oder der `null` Wert. Strukturierend wirken Arrays, die als Auflistung dienen sowie Objekte, die eine gruppierende Funktion haben. In der Kommunikation mit der Drohne dient JSON als Übertragungsformat von Steuerbefehlen (zur Drohne) sowie Statusaktualisierungen (von der Drohne). Für Steuerbefehle wurden bestimmte Schemata festgelegt, die in diesem Abschnitt ausgeführt werden. Zunächst sollte für die Übertragung zur Drohne eine manuelle Steuerung sowie eine Notlandung zu jedem Zeitpunkt möglich sein. Hinzu kommen Befehle zum Anhalten und Fortsetzen des Erkundungsfluges. Die anzufliegenden Punkte, die von der Pfadplanung (siehe Kapitel 5) berechnet wurden, müssen auch berücksichtigt werden. Außerdem müssen die Ergebnisse der Selbstlokalisation (siehe Kapitel 4) in die Flugsteuerung der Drohne übernommen werden. Das allgemeine Schema, das für alle Übertragungen übereinstimmt, sieht ein JSON-Objekt mit einem Schlüssel `command` vor (vergleiche Listing 7.1).
Der Wert von `command` legt fest wie der Inhalt der Nachricht interpretiert werden soll. Mögliche Werte sollen hierbei `manual`, `pathplanning`, `tracking`, `wait` und

```
DEPTH <Größe in Bytes><nl>
<komprimiertes Bild><nl>
```

Abbildung 7.2.: Protokoll zur Abgrenzung einzelner Tiefenbilder in einem TCP-Socket

```
1 {
2         "command" : "manual"
3 }
```

Listing 7.1: Beispiel für eine JSON Zeichenkette für einen manuellen Steuerbefehl ohne Inhalt

`continue` sein, deren Funktionsweisen im Folgenden erläutert werden.

Manuelle Steuerungsbefehle - manual Unter einem weiteren Schlüssel `setdesired` wird ein Objekt mit vier Zahlenwerten zu den Schlüsseln `roll`, `pitch`, `yaw` und `altitude` erwartet. Diese Werte überschreiben die automatische Steuerung. Der Schlüssel `landing`, sofern er den Wert `true` enthält, fordert die Drohne zur sofortigen Landung auf.

Erkundung - pathplanning Im Schlüssel `waypoints` wird eine geordnete Folge von Wegpunkten abgelegt. Ein Wegpunkt ist dabei eine Position im Raum und wird mit seinen Koordinaten x, y und z in ein JSON Objekt überführt, welches die entsprechenden Schlüssel `x`, `y` und `z` enthält.

Selbstlokalisation - tracking Die Selbstlokalisation aus Kapitel 4 bestimmt die Position der Drohne im Raum. Damit der nächste Wegpunkt sicher angesteuert werden kann, muss die Drohne über ihre aktuelle Position informiert werden. Dazu wird der `command`-Schlüssel auf `tracking` gesetzt. Listing 7.2 zeigt eine Beispiel JSON-Zeichenfolge, die von ARGOS-Control an die Drohne gesendet wurde. `position` und `orientation` beschreiben die aktuelle Position sowie Ausrichtung der Drohne. Der Zeitstempel `timestamp` wird als Unix Timestamp angegeben.

Unterbrechungsanforderung - wait Unterbrechungsanforderungen sind notwendig, falls die Selbstlokalisation oder Pfadplanung mehr Zeit zur Berechnung benötigt. Die Drohne wird in einen Wartezustand versetzt, in dem sie versucht die aktuelle Position zu halten und auf einen Fortsetzungsbefehl wartet. Da mehrere Akteure der Drohne Befehle senden, wird jeder Warte- und Fortsetzungsbefehl mit einer Priorität versehen, die im Schlüssel `wait` als ein Ganzzahlwert gespeichert wird.

```
{
        "command" : "tracking",
        "timestamp" : 1455468521,
        "position" :
        {
                "x" : 10,
                "y" : 40,
                "z" : 50
        },
        "orientation" :
        {
                "x" : 10,
                "y" : 40,
                "z" : 50,
                "angle" : 10
        }
}
```

Listing 7.2: Beispiel für eine JSON Zeichenkette für eine Positionsübertragung von oder zur Drohne

Fortsetzungsbefehl - continue Die Priorität von Fortestzungsbefehlen wird im Schlüssel `continue` gespeichert und muss mindestens derjenigen, des vorangegangenen Wartebefehls, entsprechen. Ist dies der Fall, setzt die Drohne ihre Arbeit fort und fliegt den nächsten Wegpunkt, sofern vorhanden, an.

ARGOS-Control benötigt von der Drohne Rückmeldung darüber, ob ein Wegpunkt erreicht wurde, wo sich die Drohne aktuell befindet sowie über den allgemeinen Status der Drohne. Das Schema zur Kommunikation von Drohne zu ARGOS-Control folgt dem im Abbildung 7.1 beschriebenen.

Wegpunkt wurde erreicht - pathplanning Dieser Aufruf gibt eine Rückmeldung an ARGOS-Control, wenn ein Wegpunkt erreicht wurde. Dabei wird unter dem Schlüssel `arrivedatwaypoint` die Koordinaten des Wegpunktes übergeben. Zusätzlich wird noch ein Zeitstempel gesetzt, um die Chronologie nachzuvollziehen.

Positionsschätzung - tracking Über die Odometrie (vergleiche Abschnitt 4.2) verfügt die Drohne über eine Schätzung hinsichtlich ihrer Position im Raum. Die Postion und Orientierung wird in der gleichen Form wie die der Selbstlokalisation übertragen (vergleiche Abbildung 7.2).

Statusaktualisierung - status Um ARGOS-Control über den Zustand der Drohne zu informieren, werden regelmäßig Statusupdates veröffentlicht. Dieser Status umfasst den aktuellen Modus der Drohne (Manueller Flug, autonome Erkundung oder Wartezustand) und speziell bei der Parrot Bebop Drone die Ladung der Batterie.

Position und Ausrichtung Bei der manuellen Steuerung wird eine Rückmeldung über die aktuelle und über die gewünschte Position gegeben. Hierbei werden die Werte jeweils im Format `roll`, `pitch`, `yaw` und `altitude` angegeben. Die aktuelle Position wird in `actual` und die gewünschte im Schlüssel `desired` gespeichert.

Die Serialisierung und Deserialisierung der JSON-Zeichenfolgen erfolgt clientseitig mit der C++ Bibliothek „jsoncpp“[5].

7.7.4. Websockets

Websockets ermöglichen bidirektionale Kommunikation über eine TCP-Verbindung [44]. Dazu wird vom Client (ARGOS-Control) eine HTTP-Get-Anfrage an den Server (Drohne) gesendet, die ein Upgrade der Verbindung auf das Websocket-Protokoll [25] anfragt. Falls der Server diese Anfrage bestätigt, wird eine Bidirektionale Verbindung zwischen den beiden Komponenten aufgebaut. Dieses Protokoll wurde dazu verwendet die JSON-Befehlsfolgen zu übertragen. Die Steuerungsbibliothek verwendet dazu die Implementierung von `websocketpp`[6]. Das Websocket-Protokoll besteht aus zwei Teilen: Dem „handshake“ und der eigentlichen Datenübertragung. Listing 7.3 und 7.4 zeigen die HTTP Header, die zwischen dem Client und dem Server ausgetauscht werden, um ein Upgrade auf das Websocket-Protokoll auszuführen.
Der Client erstellt eine Base64 enkodierte Zeichenkette, die in dekodierter Form eine Länge von 16 Bytes hat. Diese Zeichenfolge wird im Header als `Sec-WebSocket-Key` an den Server übergeben.
Zur Verifikation, dass der Server die Websocket Anfrage akzeptiert, wird der `Sec-WebSocket-Key` vom Client mit der festen Zeichenfolge „`258EAFA5-E914-47DA-95CA-C5AB0DC85B11`“ konkateniert. Dies würde im Beispiel aus Listing 7.3, den Wert „`dGhlIHNhbXBsZSBub25jZQ==258EAFA5-E914-47DA-95CAC5AB0DC85B11`“ ergeben. Zur Rückmeldung muss daraus der SHA-1 Hashwert gebildet werden und

[5] `https://github.com/open-source-parsers/jsoncpp` Abruf: 19.06.16 16:00
[6] `https://github.com/zaphoyd/websocketpp` Abruf: 19.06.16 16:00

```
GET /chat HTTP/1.1
Host: server.example.com
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Key: dGhlIHNhbXBsZSBub25jZQ==
Origin: http://example.com
Sec-WebSocket-Protocol: chat, superchat
Sec-WebSocket-Version: 13
```

Listing 7.3: Handshake (Get-Anfrage) zum Upgrade auf das Websocket-Protokoll vom Client zum Server (Drohne) [25]. Der Client sendet einen `Sec-WebSocket-Key` als Identifikation.

```
HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept: s3pPLMBiTxaQ9kYGzzhZRbK+xOo=
Sec-WebSocket-Protocol: chat
```

Listing 7.4: Handshakeantwort zum Upgrade auf das Websocket-Protokoll vom Server zum Client akzeptiert nur den Client mit der zugehörigen `Sec-WebSocket-Key`, die jetzt im Header `Sec-WebSocket-Accept` codiert ist. [25]

Base64 enkodiert in den Header `Sec-WebSocket-Accept` übergeben werden. Dieser Handshake garantiert dem Client, dass er sich mit einem Websocket-Server verbunden hat. Der Server akzeptiert nur Verbindungen die Websocketverbindungen sind und denen ein Handshake vorausgeht.

7.8. Software-Schnittstelle zur Steuerung

Die Schnittstelle zur Drohne wurde von ARGOS-Control ausgelagert, um eine unabhängige Entwicklung zu ermöglichen. Diese Bibliothek beinhaltet alle Befehle, die zur Steuerung der Drohne nötig sind sowie die Möglichkeit die Sensordaten der Drohne zu empfangen. Sie verwendet Websockets sowie RTSP (vergleiche Abschnitt 7.7.1) zur Übertragung von Steuerbefehlen und dem Empfangen von Kameradaten. Callbacks dienen der Abfrage von Sensordaten. Vier verschiedene Callbacks, die von der Schnittstelle aufgerufen werden, sobald neue Daten verfügbar sind, können festgelegt werden. Der `ArrivedAtWaypointCallback` dient der Benachrichtigung zu dem Zeitpunkt des Erreichens eines Wegpunktes durch die Drohne. Übergeben wird dabei die Koordinaten des Wegpunkte sowie ein Zeitstempel. Ein weiterer Callback ist `TrackingPositionCallback`, der die Positionsbestimmung der Odometrie der Droh-

ne wiedergibt. Diese besteht aus einem dreidimensionalem Vektor für die Position sowie einem vierdimensionalem Vektor für die Orientierung. Diese Informationen werden von der Schnittstelle ausgelesen und als Parameter des Callbacks weitergereicht. Für die manuelle Steuerung eignet sich der `ManualControlCallback`, der regelmäßig über die aktuelle und gewünschte Ausrichtung und Position der Drohne informiert. Hierbei wird im Gegensatz zum `TrackingPositionCallback` die Orientierung als Roll-Nick-Gier-Winkel angegeben. Des Weiteren enthält der Callback als zusätzliche Information die Höhe, in der sich die Drohne befindet. Der Roll-Nick-Gier-Winkel, „roll-pitch-yaw angle„ , ist eine Möglichkeit die Orientierung von Objekten, in der Regel Flugobjekte, im dreidimensionalen Raum darzustellen. Dazu wird im Mittelpunkt der Drohne ein dreidimensionales Koordinatensystem aufgespannt, dessen z-Achse gerade nach oben gerichtet ist. Die x-Achse zeigt in die normale Flugrichtung und die y-Achse liegt normal zu der z- und x-Achse.

Gieren (engl. yaw): Beschreibt die Drehung um die z-Achse und somit eine Richtungsänderung.

Nicken (engl. pitch): Ist die Drehung um die y-Achse und bewirkt bei der Drohne einen Vorwärts- bzw. Rückwärts-Flug.

Rollen (engl. roll): Die Drehung um die x-Achse und ermöglicht es die Drohne seitlich fliegen zu lassen.

Initial muss die Schnittstelle mit der Drohne verbunden werden. Dazu dient der Befehl `Connect(string ip)`, der eine Verbindung zur Drohne mit der IP Addresse `ip` aufbaut. Anschließend können Steuerbefehle direkt an die Drohne gesendet werden. `ManualControl(Position a)` erlaubt die direkte Beeinflussung von Ausrichtung und Flughöhe der Drohne. Jede automatische Steuerung wird von den als Parameter `Position` übergebenen Werten überschrieben. Das C++ Struct `Position` setzt sich aus den vier `double` Werten `yaw`, `pitch`, `roll` und `altitude` zusammen. Die automatische Steuerung kann mit Hilfe der Methode `Wait(int prio)` unterbrochen werden, wobei `prio` die Priorität angibt mit der gewartet werden soll. Der korrespondierende `Continue(int prio)`-Befehl setzt die automatische Erkundung der Drohne fort (vergleiche Abschnitt 7.7.3). In einer Ausnahmesituation, wenn z. B. die Selbstlokalisation auf Grund von schlechten Lichtverhältnissen versagt und die Drohne droht in ein Hindernis zu fliegen, kann der Aufruf `Land()` dazu dienen die Drohne schnell zum Boden zu bringen, um Schäden zu verhindern. Neben dieser Anwendung dient `Land()` dem Landen der Drohne, wenn die Erkundung abgeschlos-

sen wurde oder die Batterie ausgewechselt werden muss. Zur automatischen Steuerung wird der Drohne mit der Methode `FollowPath(vector<Eigen::Vector3d> path, long time)` eine geordnete Liste von dreidimensionalen Vektoren übergeben, die in der gegebenen Reihenfolge von der Drohne angeflogen werden soll. Die von der Selbstlokalisation (siehe Kapitel 4) erfasste Position der Drohne kann mit dem Aufruf von `CalculatedPosition(Eigen::Vector3d position, const Eigen::Vector4d& orientation,long time)` an die Drohne weitergegeben werden. Die Parameter umfassen eine Position im Raum als dreidimensionalen Vektor sowie die Ausrichtung der Drohne als vierdimensionalen Vektor. Der zusätzliche Parameter `time` dient der Drohne zur Verifikation der Aktualität der Daten.
Die Bibliothek wurde als eine Dynamic Link Library zusammengefügt und verwendet für das Auslesen des RTSP-Streams (siehe Abschnitt 7.7.1) die Bildbearbeitungsbibliothek OpenCV[7]. Außerdem wird die Bibliothek Eigen[8] für die Übergabe der Parameter an die Schnittstelle verwendet.

[7] `http://opencv.org/` Abruf: 19.06.16 17:00
[8] `http://eigen.tuxfamily.org/` Abruf: 19.06.16 17:00

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15	
length	
sequence	
acc_roll (unsigned)	accelerometer
acc_pitch (unsigned)	
acc_yaw (unsigned)	
gyro_roll (signed)	gyroscope
gyro_pitch (signed)	
gyro_yaw (signed)	
temperature_acc (unsigned)	
temperature_gyro (unsigned)	
ultrasound (unsigned)	ultrasound
us_debut_echo (unsigned)	
us_fin_echo (unsigned)	
us_association_echo (unsigned)	
us_distance_echo (unsigned)	
us_curve_time (unsigned)	
us_curve_value (unsigned)	
us_curve_ref (unsigned)	
nb_echo (unsigned)	
sum_echo[0:15] (unsigned)	
sum_echo[16:31] (unsigned)	
gradient (signed)	
pressure (signed)	
temperature_pressure (signed)	
m_x (signed)	magnetometer
m_y (signed)	
m_z (signed)	
checksum	

Tabelle 7.2.: Format eines Datenpakets vom Navboard der AR.Drone2

8. Benutzerschnittstellen

In diesem Kapitel werden die zwei Benutzerschnittstellen vorgestellt und beschrieben. Im ersten Unterkapitel wird die Webbrowser-basierte Schnittstelle von ARGOS-Control gezeigt. Dort wird die manuelle Steuerung und die Live-Visualisierung während eines Erkundungsfluges erläutert. Im zweiten Unterkapitel wird die Schnittstelle der Rekonstruktion beschrieben. Dort werden die einzelnen Menüeinträge erläutert und die Funktionen des Tiefendaten-Players aufgelistet.

8.1. ARGOS-Control

Die Anwendung *ARGOS-Control* stellt die Steuerungssoftware für die Drohne dar. Sie setzt einen gültigen Grundriss der zu erkundenden Umgebung voraus und steuert die Drohne selbstständig, sodass der Tiefensensor während des Erkundungsflugs die gesamte Umgebung aufzeichnen kann.

Während des Fluges hat der Benutzer die Möglichkeit, diesen über eine Visualisierung zu verfolgen. Diese stellt die Umgebung anhand des eingegebenen Grundrisses dar und zeigt die aktuelle Position der Drohne sowie die von der Anwendungen analysierten Hindernisse visuell an.

Abbildung 8.1 zeigt die initiale Ansicht der Benutzerschnittstelle von ARGOS-Control. Der linke Bereich des Anwendungsfensters ist in die Reiter *Connection and Log* und *Data and Control* aufgeteilt. Ersterer ist für die Verbindungen zur Drohne und zu ARGOS-Control zuständig. Zudem werden dort die Nachrichten, welche über beide Verbindungskanäle ausgetauscht werden, angezeigt.

Der zweite Reiter, welcher in Abbildung 8.2 dargestellt wird, zeigt die aktuellen Daten der Drohne an. Diese umfassen die Werte der Neigungssensorik sowie die aktuelle Position und Rotation der Drohne in Form von Koordinaten. Weiter werden die Verbindungsstatus zur Drohne sowie zum ARGOS-Control angezeigt.

Die folgenden Abschnitte behandeln die Möglichkeit der manuellen Steuerung der Drohne über die Benutzerschnittstelle sowie die Live-Visualisierung.

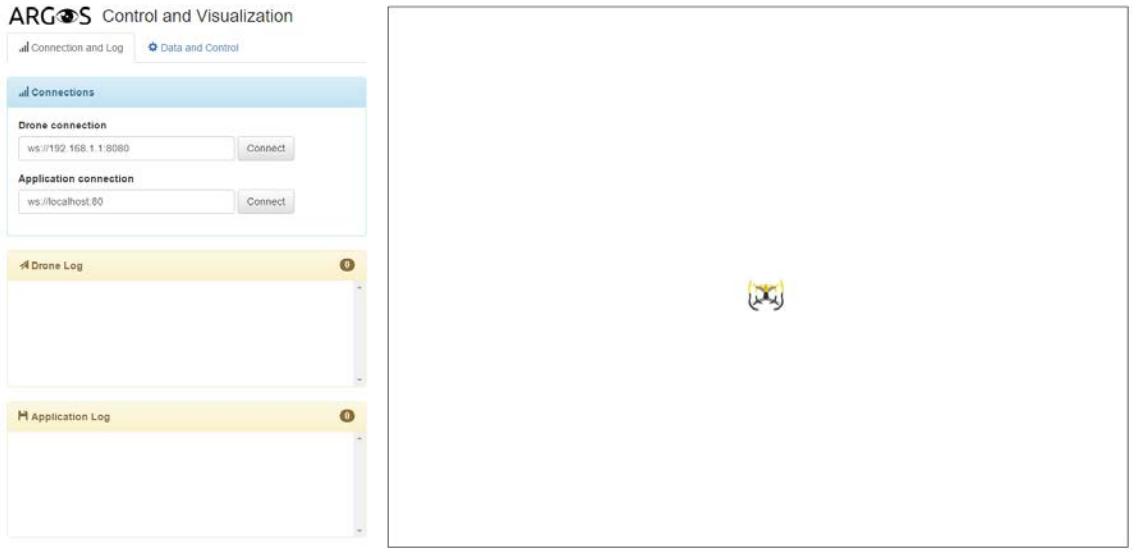


Abbildung 8.1.: Die Benutzerschnittstelle zu ARGOS-Control ermöglicht die Verbindung zur Drohne sowie zur Anwendung (links) und visualisiert den aktuellen Drohen-Status (rechts).

8.1.1. Manuelle Steuerung

Die Schnittstelle gibt dem Benutzer die Möglichkeit zur Laufzeit das Verhalten der Drohne zu beeinflussen. Dies dient dazu, die Drohne im Notfall wie beispielsweise einem Softwarefehler, manuell zu landen oder den Erkundungsflug zu pausieren. Weiter kann die Drohne vom Anwender gesteuert werden. Durch Benutzung der Pfeiltasten auf der Tastatur sowie der Tasten W, A, S und D wird die Neigung und die Flughöhe beeinflusst. Alternativ können Koordinaten eingegeben werden, zu welchen sich die Drohne drehen oder welche sie anfliegen soll.
Abbildung 8.2 zeigt im unteren Bereich mit dem Titel *Control* die Benutzerschnittstelle zur manuellen Steuerung. Neben den Bezeichnungen für die Neigungswinkel sind die Tasten abgebildet, über welche sie gesteuert werden können. Wird eine Taste auf der Tastatur gedrückt, färbt sich die Tastendarstellung orange. Zusätzlich wird die Änderung rechts daneben angezeigt.
Unter der Tastennavigation stehen Eingabefelder für Koordinaten zum Drehen und Anfliegen zur Verfügung. Zudem existieren Bedienungselemente zum Pausieren des Erkundungsfluges sowie zur Erzwingung einer Landung.

8.1.2. Live-Visualisierung

Die von der Drohne empfangenen Positions- und Rotationsdaten werden im rechten Bereich der Benutzerschnittstelle visuell aufbereitet. Abbildung 8.1 zeigt den

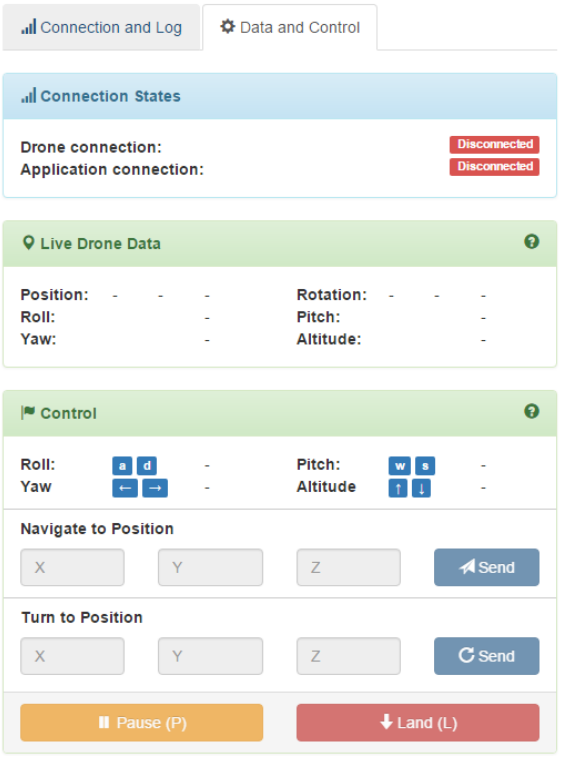


Abbildung 8.2.: Anzeige der Drohnendaten und Möglichkeiten der manuellen Steuerung im Reiter *Data and Control*.

initialen Bildschirm der Visualisierung an, welche durch einen schwarzen Rahmen abgegrenzt wird. Bei einem Erkundungsflug wird zunächst der eingegebene Grundriss dargestellt und so skaliert, dass er auf der längeren Seite die volle Breite der Visualisierung ausnutzt.
Während die Drohne ihre Umgebung erkundet, bewegt sich das Drohnen-Symbol, welches zu Beginn zentriert positioniert ist, innerhalb des Grundrisses. Von der Drohne entdeckte Hindernisse werden als grüne Kästchen dargestellt, wobei Hindernisse auf der gleichen Höhe der Drohne in dunklerem Grünton erscheinen. Dadurch sollen dem Benutzer für die Drohne gefährliche Hindernisse und damit Kollisionspotenziale kenntlich gemacht werden, so dass dieser im Zweifelsfall eingreifen kann.

8.2. Rekonstruktion

Die grafische Oberfläche, welche als Benutzerschnittstelle für die Rekonstruktion dient, wird in diesem Kapitel vorgestellt. Der Hauptnutzen liegt im Erstellen einer rekonstruierten 3D-Karte mithilfe von Tiefendaten. Weitere Funktionen sind u. a. das Darstellen und Schneiden von Tiefendaten, das Öffnen und Aufzeichnen von

Daten eines lokal angeschlossenen Tiefensensors sowie verschiedene Einstellungen für den Rekonstruktionsprozess. In den einzelnen Unterkapiteln wird die Benutzung und die Funktionsweise näher beschrieben.
In Abbildung 8.3 ist das Hauptfenster der grafischen Oberfläche zu sehen. Im Folgenden werden die einzelnen Menüleisteneinträge erklärt.

Menüleiste - File Im ersten Eintrag der Menüleiste sind Funktionen zum Öffnen und Schneiden von Tiefendaten. Zudem kann ein lokal angeschlossener Tiefensensor gestartet und beendet werden. Die aufgenommenen Daten werden lokal abgespeichert.

Menüleiste - KinFu Die Funktionen zum Rekonstruieren und Visualisieren der Szene befinden sich unter diesem Menüpunkt. Des Weiteren können Einstellungen und Parameter für KinectFusion gesetzt werden.

Menüleiste - Options und Windows Weitere Einstellungsmöglichkeiten, wie bspw. das Ein- bzw. Ausschalten der Komprimierung beim Schneiden, werden bereitgestellt. Verschiedene Fenster, wie bspw. die Vorschaufenster von KinFu (siehe Abbildung 8.5), können versteckt bzw. gezeigt werden.

Log-Ausgabe und Fortschrittsanzeige In der Log-Ausgabe werden verschiedene Informations-, Debug- und Error-Ausgaben dargestellt. Die jeweiligen Einträge werden farblich hervorgehoben. Sollte ein Prozess begonnen werden, der voraussichtlich länger als eine Sekunde dauert, wird der Fortschritt des Prozesses in der Fortschrittsanzeige dargestellt. Der Fortschritt der Rekonstruktion wird bspw. dort angezeigt.

Depth Image Sequence

Dateien mit Tiefendaten werden durch die Depth Image Sequence visualisiert. Fortan wird das Wort Player synonym für die Depth Image Sequence benutzt, da diese einem Mediaplayer nahe kommt. Der Player verfügt über Funktionen, um die Tiefendaten zu visualisieren und abzuspielen. Diese Funktionen werden durch die Tastatur gesteuert. Dazu zählt das Pausieren, Stoppen, Vor- und Zurückspulen des Videos.
Wie in Abbildung 8.4 zu sehen ist, befinden sich zusätzliche Informationen im dargestellten Fenster. In der oberen linken Ecke wird der aktuelle Frame und der maximale Frame angezeigt. Unten befindet sich eine Trackbar, auf der sich zum einen ein weißer Strich befindet, welcher die relative Position des aktuellen Frames darstellt, und

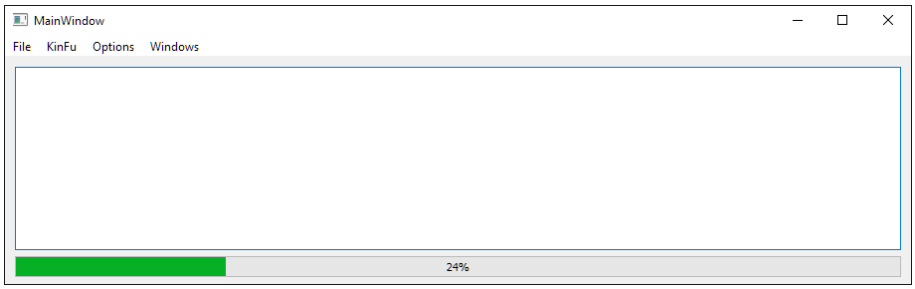


Abbildung 8.3.: Hauptfenster der Benutzerschnitstelle der Rekonstruktion mit einer Menüleiste für diverse Einstellungs- und Ausführoptionen, einem Bereich für Log-Ausgaben und einer Fortschrittsanzeige bei zeitintensiven Berechnungen

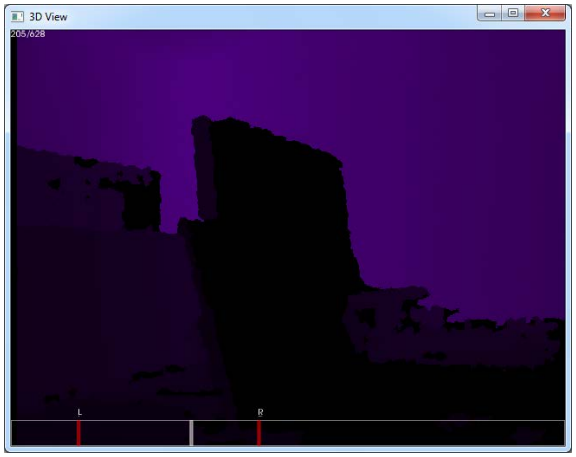


Abbildung 8.4.: Fenster der DepthImageSequence mit angezeigten Tiefendaten. Zu sehen ist außerdem die Trackbar mit der aktuellen Position des Frames (weißer Strich) und die Start- bzw. Endposition des zu schneidenden Videos (rote Striche).

zum anderen zwei rote Striche, welche zum Schneiden der Tiefendaten benutzt werden. Der Bereich zwischen den roten Strichen wird extrahiert, sodass die Tiefendaten vor und nach den roten Strichen abgeschnitten werden. Ein Linksklick innerhalb der Trackbar setzt den aktuellen Frame neu. Durch den ersten Rechtsklick innerhalb der Trackbar wird die linke Position zum Schneiden gesetzt und durch einen weiteren Rechtsklick die rechte Position. Das Abspeichern der geschnittenen Daten erfolgt in eine Datei mit der Endung „dat“, welche auch vom Player dargestellt werden kann.

KinectFusion

Nach dem Start von KinFu wird ein Fenster geöffnet (siehe Abbildung 8.5), welches die generierten Tiefendaten und die Differenz zum Eingabebild zeigt. Die Ausgangsposition ist die aktuelle Pose. Es ist möglich diese durch die Pfeiltasten zu

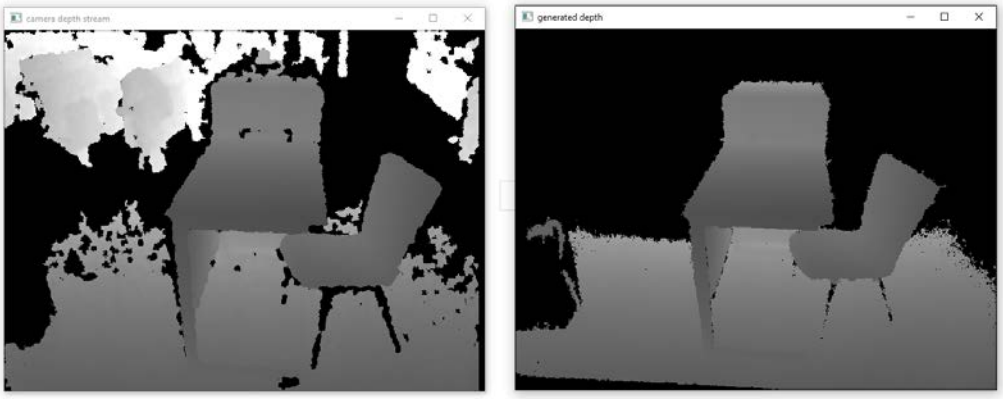


Abbildung 8.5.: KinFu Hauptfenster mit dem Tiefenbild (links) und TSDF Tiefenbild (rechts). Erkannte Entfernungen zu Objekten werden als Grauton dargestellt.

verschieben und zu rotieren.

Nach der Rekonstruktion wird ein weiteres Fenster geöffnet (siehe Abbildung 8.6), welches die rekonstruierte 3D-Szene darstellt. Zu sehen sind die beleuchteten TSDFs, wobei auch hier die aktuelle Pose gezeigt ist. Auch diese lässt sich durch Drücken der Pfeiltasten frei verschieben.

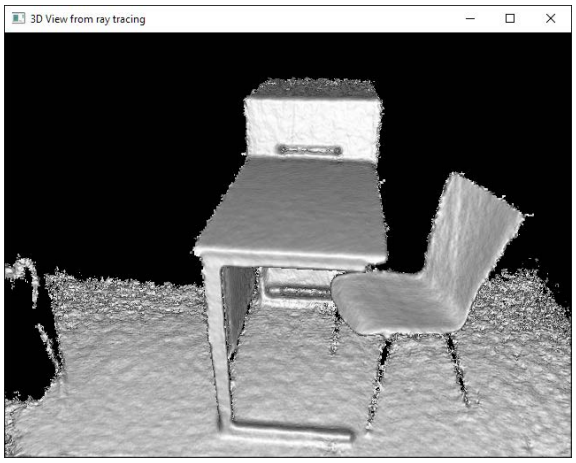


Abbildung 8.6.: Rekonstruierte 3D-Szene oder auch „ray-casting“-Szene. Zu sehen ist die berechnete 3D-Szene durch KinFu, wobei die weißen Flächen Objekte repräsentieren und den schwarzen Flächen keine Tiefendaten zugeordnet wurden.

9. Evaluierung

In diesem Kapitel wird anhand verschiedener Evaluierungsszenarien das entwickelte Projekt getestet und ausgewertet. Die folgenden Unterkapitel beschreiben die Evaluierungsszenarien und den Evaluierungsvorgang. Unterkapitel 9.1 beschreibt die ausgewählten Evaluierungsszenarien und die Testumgebung. Das Unterkapitel ist in mehrere Abschnitte unterteilt. Dazu werden die Szenarien gruppiert und einem Teilbereich zugeordnet. Bereits im Pflichtenheft wurden diverse Testszenarien definiert. Um hier einen konkreten und aussagekräftigen Überblick über das Projekt zu erhalten, wurden wichtige Szenarien aus dem Pflichtenheft (siehe Anhang A.1) übernommen.
Im Unterkapitel 9.2 werden die verschiedenen Ergebnisse der zuvor getesteten Testszenarien ausgewertet und evaluiert. Die Auswertung wird anhand mehrerer Kriterien bewertet und zugeordnet.
Desweiteren wird im Unterkapitel 9.3 eine Diskussion über die Evaluierung gehalten. So wird u. a. diskutiert, ob die einzelnen Module insgesamt ausreichende Daten liefern, damit andere Module diese verwenden können.

9.1. Evaluierungsszenarien

In diesem Kapitel werden die Szenarien der verschiedenen Module beschrieben und erläutert. Die Reihenfolge der Module entspricht der des Endberichts. Es wird zuerst der ARGOS-Agent beschrieben, danach die Raumerkundung und das Tracking. Zum Schluss werden Szenarien der Umgebungsrekonstruktion definiert.

9.1.1. ARGOS-Agent

Im Teilgebiet ARGOS-Agent wird die Funktionalität der Datenübertragung von der Drohne zum Client (dem Computer) evaluiert. Diese wird grob in drei Kategorien unterteilt: die Übertragung der Steuerbefehle über Websockets, der Kamerabilder über einen RTP-Server auf der Drohne, und der 3D-Kamerabilder direkt über TCP.

1. **Die Odometriedaten der Drohne können über die Webschnittstelle eingesehen werden**
   a) *Aufbau*: Die Drohne wird gestartet und die Firmware geladen. Der Rechner wird mit der Drohne verbunden.
   b) *Ablauf*: Auf dem Rechner kann mit einem Webbrowser (z. B. Mozilla Firefox) auf die Drohne zugegriffen werden. Daten zur Odometrie sind sofort einsehbar. Befehle (z. B. starten, landen) können an die Drohne gesendet werden.
2. **Die Odometriedaten der Drohne können über die Software-Bibliothek abgerufen werden**
   a) *Aufbau*: Die Drohne wird gestartet und die Firmware geladen. Der Rechner wird mit der Drohne verbunden.
   b) *Ablauf*: Auf dem Rechner können unter Verwendung der Software-Bibliothek (vergleiche Abschnitt 7.7) Daten eingesehen werden sowie Befehle an die Drohne verschickt werden.
3. **Das Kamerabild der Drohne kann bei bestehender W-LAN Verbindung als RTP-Stream mit einem Videoplayer (VLC-Player) abgespielt werden**
   a) *Aufbau*: Die Drohne wird gestartet. Der Computer auf dem die Clientsoftware läuft wird mit der Drohne verbunden.
   b) *Ablauf*: Auf dem Computer wird mit einer Videosoftware (z. B. VLC-Player[1]) der RTP-Stream geöffnet.
4. **Das Kamerabild der Drohne kann bei bestehender W-LAN Verbindung als RTP-Stream mit der Software-Bibliothek aufgerufen werden**
   a) *Aufbau*: Die Drohne wird gestartet. Der Computer auf dem die Clientsoftware läuft wird mit der Drohne verbunden.
   b) *Ablauf*: Auf dem Computer wird unter Verwendung der Software-Bibliothek der RTP-Stream geöffnet.
5. **Ein mit der 3D-Kamera vorher aufgezeichnetes Video mit einer Auflösung von 640×480 Pixeln bei 30fps wird von der Drohne übertragen**

[1] `http://www.videolan.org/vlc/` Abruf: 29.06.16 11:00

a) *Aufbau*: Ein USB-Stick wird mit einem Testvideo mit einer Auflösung von 640×480 Pixeln und 30 Bildern pro Sekunde bespielt und an die Drohne angeschlossen. Die Drohne wird gestartet. Der Computer auf dem die Clientsoftware läuft wird mit der Drohne verbunden.

b) *Ablauf*: Auf dem Computer kann mit dem OpenNI2 SimpleViewer der Stream geöffnet und betrachtet werden.

6. **Die Bilder der 3D-Kamera auf der Drohne werden mit einer Auflösung von 320×240 Pixeln bei 30fps übertragen**

   a) *Aufbau*: Die 3D-Kamera wird mit der Drohne verbunden. Die Einstellungen für die 3D-Kamera werden auf die Auflösung 320×240 Pixeln gestellt. Die Drohne wird gestartet. Der Computer auf dem die Clientsoftware läuft wird mit der Drohne verbunden.

   b) *Ablauf*: Auf dem Computer kann mit dem OpenNI2 SimpleViewer der Stream geöffnet und betrachtet werden.

7. **Die Bilder der 3D-Kamera auf der Drohne werden mit einer Auflösung von 640×480 Pixeln bei 30fps übertragen**

   a) *Aufbau*: Die 3D-kamera wird mit der Drohne verbunden. Die Einstellungen für die 3D-Kamera werden auf die Auflösung 640×480 Pixeln gestellt. Die Drohne wird gestartet. Der Computer auf dem die Clientsoftware läuft wird mit der Drohne verbunden.

   b) *Ablauf*: Auf dem Computer kann mit dem OpenNI2 SimpleViewer der Stream geöffnet und betrachtet werden.

8. **Die Tiefenbild-Kompression ist funktionsfähig, echtzeitfähig und erzeugt ausreichend kleine Komprimate**

   Der in den vorangegangenen Szenarien für die 3D-Kamera eingesetzte, selbst entwickelte Algorithmus zur Kompression der Tiefenbilder wird betrachtet. Die Zielsetzung dieser Evaluierung ist die Feststellung von drei Kriterien. Das erste Kriterium ist die Funktionsfähigkeit, d. h. dass die Ausgabe der Eingabe entspricht (dies ist kein Korrektheitsbeweis, würde aber grobe Fehler aufzeigen). Das zweite Kriterium ist die Echtzeitfähigkeit, d. h. dass ein Limit für die verbrauchte Rechenzeit eingehalten wird. Das dritte Kriterium ist die Tauglichkeit für die Übertragung von Tiefenvideos über ein Netzwerk, d. h. eine ausreichend hohe Kompressionsrate. Dabei wird der eigene Algorithmus einer

Reihe anderer Algorithmen gegenübergestellt, auch um seine Notwendigkeit aufzuzeigen.

a) *Aufbau*: Ein etwa 27 Sekunden langes Video (803 Einzelbilder) wurde erstellt, das aus mehreren unterschiedlichen Szenen in der von der Rekonstruktion benötigten Auflösung von 640×480 Pixeln bei einer Bildwiederholrate von 30 Tiefenbildern pro Sekunde besteht.

b) *Ablauf*: Der eigene Algorithmus, ein vom Hersteller der Kamera eingesetzter Algorithmus und verschiedene andere Standard-Kompressionsalgorithmen wurden auf dieses Video angewendet. Dabei wurde sowohl die benötigte Rechenzeit als auch die Kompressionsrate für jedes einzelne Bild gemessen.

9.1.2. Raumerkundung

Im Bereich der Raumerkundung wird die Genauigkeit des Fluges und der Pfadplanung sowie die Laufzeit evaluiert. Dazu wird gemessen, wie oft die Drohne eine Drehung im Bezug zum optimalen Pfad vornimmt oder es wird gemessen, wie weit der Abstand zu den gefundenen Hindernissen ist. Des Weiteren wird die Distanz zu verschiedenen Erkundungspunkte (siehe Kapitel 5.2) gemessen.
Die Messung der Laufzeit erfolgt in Abhängigkeit von Variablen, wie der Raumgröße oder der Anzahl an Hindernissen.

1. **Abstandsmessung zwischen der Drohne und ihrem Ziel, um den Abstand zu umflogenen Hindernissen zu messen**

   a) *Aufbau*: Im Simulator wird eine Szene mit einer Breite von 10m und einer Länge von 7,9m erstellt. Die Drohne startet 50cm von der südlichen Wand entfernt (siehe Abbildung 9.1). Das Ziel liegt 6,5m von der Drohne entfernt und auf dem Weg dorthin befindet sich ein Hindernis, dass eine Breite von 1m hat. Der Versuch wird 4 mal wiederholt und das Hindernis jedes Mal einen Meter breiter.

   b) *Ablauf*: Nach dem Start der Drohne wird versucht das Ziel zu erreichen. Gemessen wird die Zeit vom Start bis zur Landung der Drohne, wobei das Ziel erreicht sein muss. Der Versuch wird 4 mal mit einem größeren Hindernis wiederholt.

2. **Laufzeitmessung der Pfadfindung durch Änderung der Raumgröße und Anzahl der Hindernisse**

   a) *Aufbau*: Dieses Szenario wird in drei grundlegenden Varianten ausgeführt (siehe Abbildung 9.2), die sich durch die Anordnung bzw. das Vorhandensein von Hindernissen unterscheiden. In jeder Variante liegt der Startpunkt von oben betrachtet in der linken oberen und der Zielpunkt in der rechten unteren Ecke des Raums. Der Abstand zwischen den Hindernissen und deren Größe wird konstant gehalten.

   b) *Ablauf*: Für jede Variante werden 20 Durchläufe mit jeweils steigender Raumgröße und entsprechend wachsender Hindernisanzahl ausgeführt. Die Raumseitenlänge beträgt initial 59 Voxel und im letzten Durchlauf 1199 Voxel. Es wird jeweils ein Pfad vom Start- zum Endpunkt berechnet und die dafür notwendige Zeit gemessen.

3. **Distanzmessung von verschiedenen Points of Interest**

   a) *Aufbau*: Die Drohne startet immer im Zentrum des Raumes, wobei die verschiedenen Räume, Maße und Hindernisse den Abbildung 9.3 bis 9.9 entnommen werden können. Zudem wird der Versuch mit dem FloodFill-Explorer (siehe Kapitel 5.2.1) durchgeführt, wobei der Grundriss und die Hindernisse bekannt sind.

   b) *Ablauf*: Ziel ist es, dass die Drohne den kompletten Raum erkundet und dabei so wenig Drehungen wie möglich macht. Während des Fluges werden die Anzahl der Drehungen und die zurückgelegte Strecke gemessen.

9.1.3. Tracking

Das Tracking wird anhand von drei Szenarien evaluiert. Zum einen wird ein Testflug vorgenommen und LSD-SLAM mit dem entstandenen Video gestartet. Der Testflug besteht aus einer vorher definierten Strecke mit verschiedenen Objekten, welche Hindernisse darstellen. Evaluiert wird die Genauigkeit von LSD-SLAM anhand von Vergleichen zwischen Objekten im Video und den gefundenen Objekten durch LSD-SLAM. Zum anderen wird die Robustheit von LSD-SLAM bei verschiedenen Lichtverhältnissen evaluiert. Dazu zählen drei Unterszenarien, welche jeweils eine andere Beleuchtung verwenden. Dabei wird Tageslicht, abgedunkeltes Tageslicht durch einen Vorhang und Licht durch Neonröhren benutzt.

[2]Satellitenaufnahme: `https://maps.google.com/` Abruf: 20.06.2016 20:00 Uhr

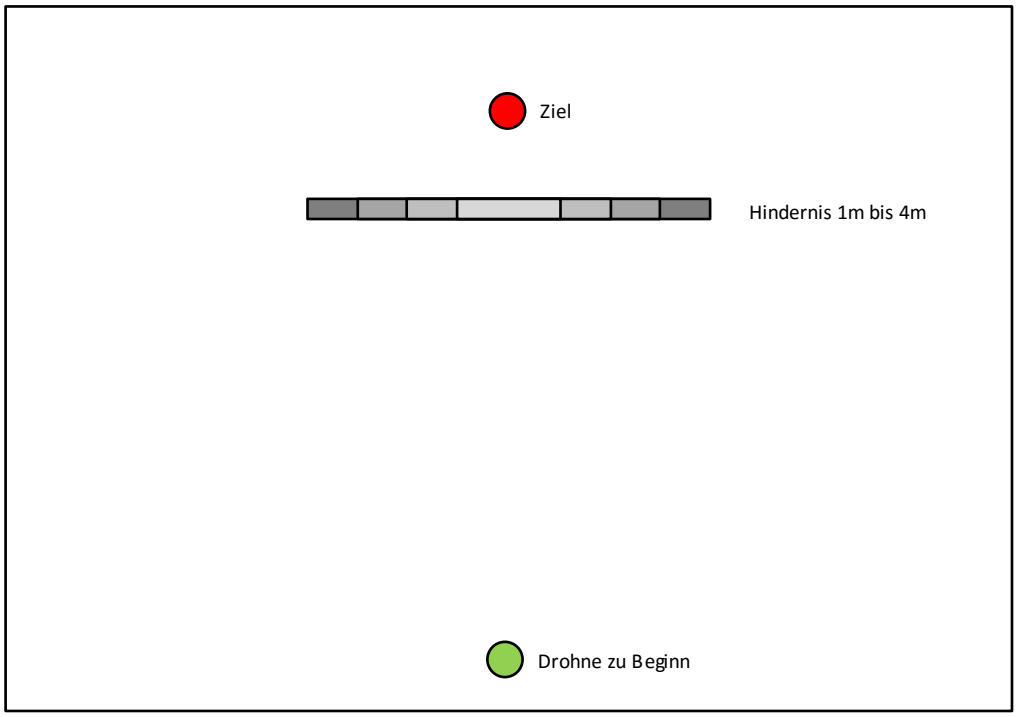


Abbildung 9.1.: Versuchsaufbau des Szenarios 5 der Pfadplanung. Die Drohne startet an der südlichen Wand mit 50cm Entfernung und versucht das Ziel zu erreichen. Zudem werden die Hindernisse in jedem Versuch um einen Meter breiter.

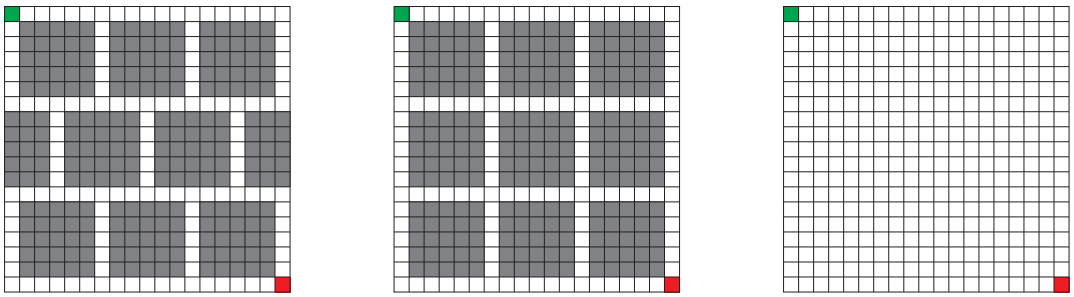
Im folgenden werden die verschiedenen Szenarien definiert:

1. **Hinderniserkennung bei verschiedenen Lichtverhältnissen**

   a) *Aufbau*: Gleichbleibende Szene bei verschiedenen Lichtverhältnissen. Es wird mit drei verschiedenen Lichtverhältnissen getestet: (a) Sonnenlicht durchs Fenster, (b) Abgedunkeltes Sonnenlicht durchs Fenster, (c) Raumbeleuchtung durch Neonröhre.

   b) *Ablauf*: Die Kamera wird bei jeweiligem Lichtverhältnis per Hand getragen. Es wird eine vorher definierte und gleichbleibende Strecke abgeflogen. Das aufgenommene Video wird durch LSD-SLAM verarbeitet und ausgewertet.

2. **Genauigkeit der Hinderniserkennung**



(a) Mit Versatz **(b)** Ohne Versatz **(c)** Ohne Hindernisse

Abbildung 9.2.: Versuchsaufbau des Szenarios 5 der Pfadplanung. Es werden drei Grundvarianten an Raumkonfigurationen untersucht. Bei der ersten Variante (siehe (a)) sind die Hindernisse derartig versetzt angeordnet, dass sich in z-Richtung kein langer Gang über die gesamte Breite des Raums ergibt. In der zweiten Variante (siehe (b)) sind die Hindernisse gitterförmig angeordnet. In der dritten Variante (siehe (c)) sind die untersuchten Räume leer. Die Startpunkte der zu berechnenden Pfade sind jeweils grün und die Zielpunkte rot markiert.

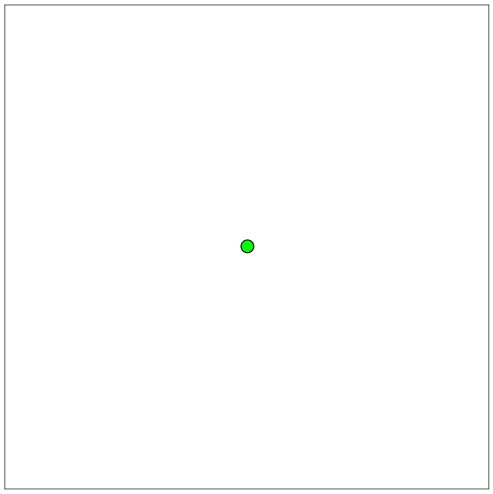


Abbildung 9.3.: Visualisierung eines quadratischen Raumes im Pfadplanungs-Explorer ohne Hindernisse. Der Raum ist 8 Meter lang und breit.

a) *Aufbau*: Mit Sonnenlicht bestrahlter Raum, welcher mehrere Hindernisse enthält.

b) *Ablauf*: Die Kamera wird per Hand getragen und an allen Hindernissen vorbeigeführt. Währenddessen wird die Szene nicht verändert und das aufgenommene Video ist ca. 90 Sekunden lang. Danach wird das aufgenommene Video durch LSD-SLAM verarbeitet und ausgewertet.

3. **Tracking-Genauigkeit von LSD-Slam**

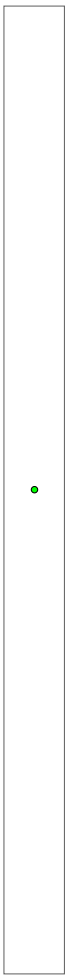


Abbildung 9.4.: Visualisierung eines langen Raumes im Pfadplanungs-Explorer ohne Hindernisse. Der Raum ist 2 Meter breit und 32 Meter lang.

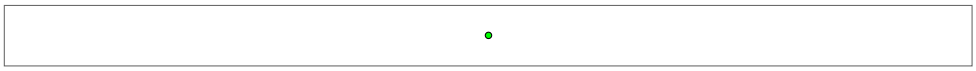


Abbildung 9.5.: Visualisierung eines breiten Raumes im Pfadplanungs-Explorer ohne Hindernisse. Der Raum ist 2 Meter lang und 32 Meter breit.

a) *Aufbau*: In einem mit Sonnenlicht beleuchtet Raum, wird eine gerade Strecke abgemessen und alle 10 cm markiert. Der Startpunkt befindet sich am Anfang der abgemessenen Strecke und der Endpunkt nach 130 cm.

b) *Ablauf*: Die Kamera wird in verschiedenen Testläufen zwischen 10 cm und 130 cm entlang einer geraden Linie bewegt.

4. **Tracking durch LSD-SLAM mit sich ähnelnden Bildern**

a) *Aufbau*: Eine Szene mit mehreren Räumen, die klar von einander zu unterscheiden sind und eine Szene, in der sich Ansichten auf einer Strecke

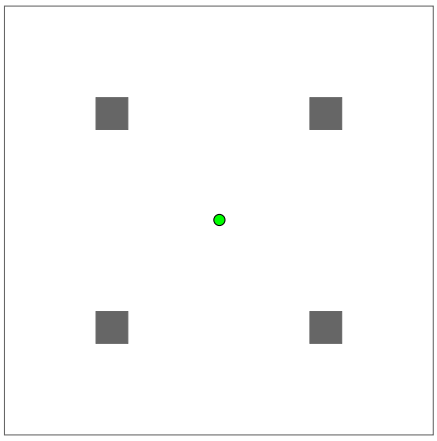


Abbildung 9.6.: Visualisierung eines Raumes im Pfadplanungs-Explorer mit vier Hindernissen, welche Säulen darstellen. Der Raum ist 8,1 Meter lang und breit.

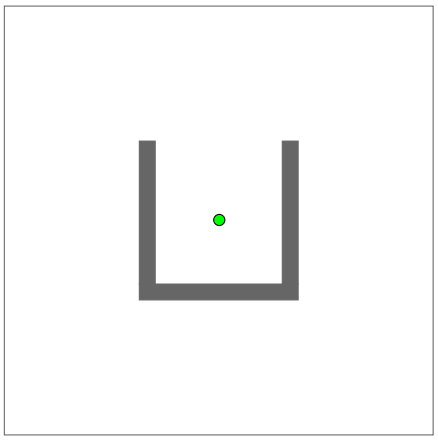


Abbildung 9.7.: Visualisierung eines Raumes im Pfadplanungs-Explorer mit drei Hindernissen, welche wie ein U angeordnet sind. Der Raum ist 8,16 Meter lang und breit.

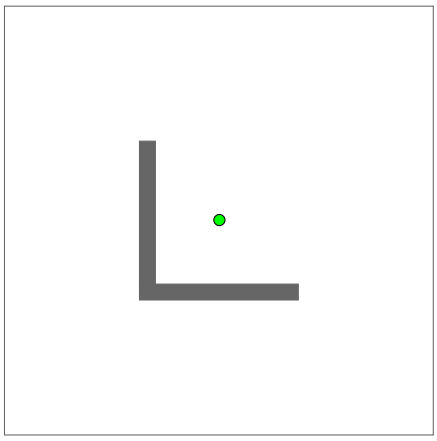


Abbildung 9.8.: Visualisierung eines Raumes im Pfadplanungs-Explorer mit zwei Hindernissen, welche wie ein L angeordnet sind. Der Raum ist 8,1 Meter lang und breit.

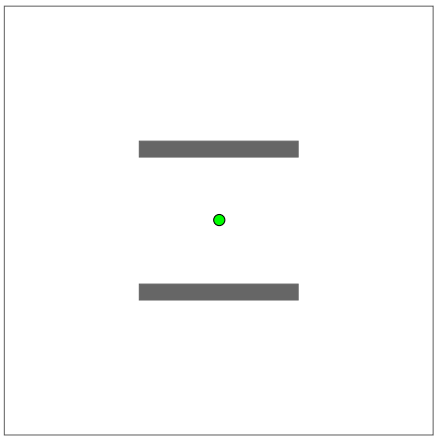
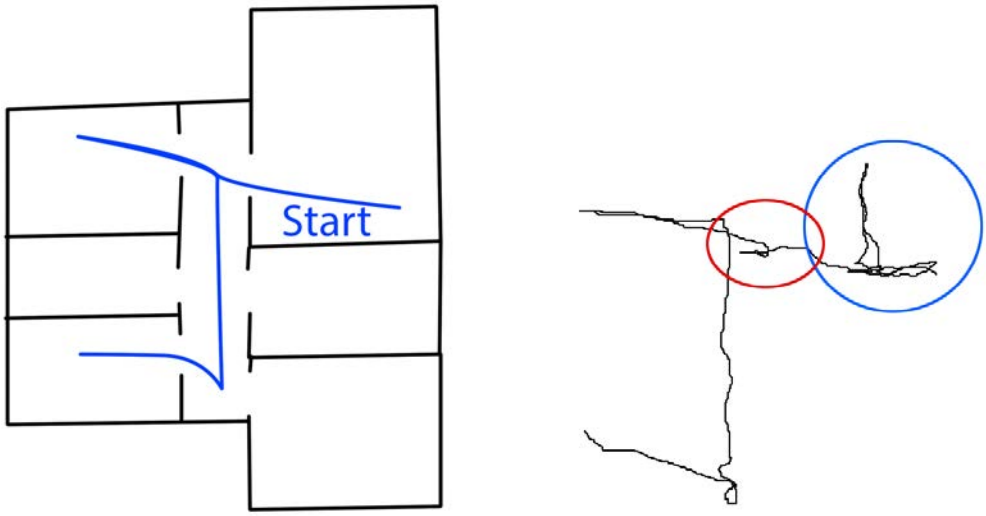


Abbildung 9.9.: Visualisierung eines Raumes im Pfadplanungs-Explorer mit zwei Hindernissen, welche parallel angeordnet sind. Der Raum ist 8,12 Meter lang und breit.

ähneln (siehe Abbildung 9.10a). Die erste Szene ist die einer Wohnung, während für die zweite Szene ein gerader Gehweg ausgesucht wurde (siehe Abbildung 9.11).

b) *Ablauf*: Aufzeichnung von Strecken innerhalb der beiden Szenen, sodass die tatsächlichen Strecken mit den geschätzten Positionen verglichen werden können. Jede Aufnahme beginnt mit einer Initialisierungs-Phase, in der die Kamera zunächst langsam ohne bzw. möglichst geringer Rotation bewegt wird. Es werden zudem ruckartige Bewegungen vermeidet.



(a) Grundriss **(b)** geschätzte Trajektorie

Abbildung 9.10.: Erste Szene für die beispielhafte Auswertung der mit LSD-SLAM erhaltenen Trajektorie. (a) zeigt den Grundriss der Wohnung (schwarz), sowie den Pfad (blau), der abgelaufen wurde. Die Abbildung (b) zeigt die zugehörige geschätzte Trajektorie, wobei der Startpunkt blau und offensichtliche Probleme rot umkreist sind.

9.1.4. Umgebungsrekonstruktion

Bei der Rekonstruktion wird getestet, wie groß die Abweichung zwischen dem berechneten, rekonstruierten 3D-Modell und dem echten Modell ist. Um ein Modell zu erhalten, dass simpel ist und immer die selben Maße hat, wurden mehrere Würfel mit Lego gebaut (siehe Abbildung 9.12). Das Lego-Modell wird als 3D-Modell am PC erstellt und der mittlere Fehler zum rekonstruierten Modell berechnet, um die Genauigkeit von KinectFusion (siehe Kapitel 6.1) zu evaluieren. Des Weiteren wird die Robustheit von KinectFusion durch die Reduzierung der Frames pro Sekunde und der Auflösung der Kamera evaluiert. Die Ergebnisse werden untereinander verglichen und ausgewertet.
Die Registrierung wird mit dem oben genannten Modellen getestet. Es wird mit den verschiedenen Algorithmen ICP und GICP registriert (siehe Kapitel 6.2.3 und ff.). Dafür wird die Paarregistrierung von Punktwolken und der mittlere Fehler Ursprungswolke verglichen.

Im Folgenden werden verschiedene Szenarien definiert:

1. **Registrierung eines erstellten 3D-Modells mit einem rekonstruierten**

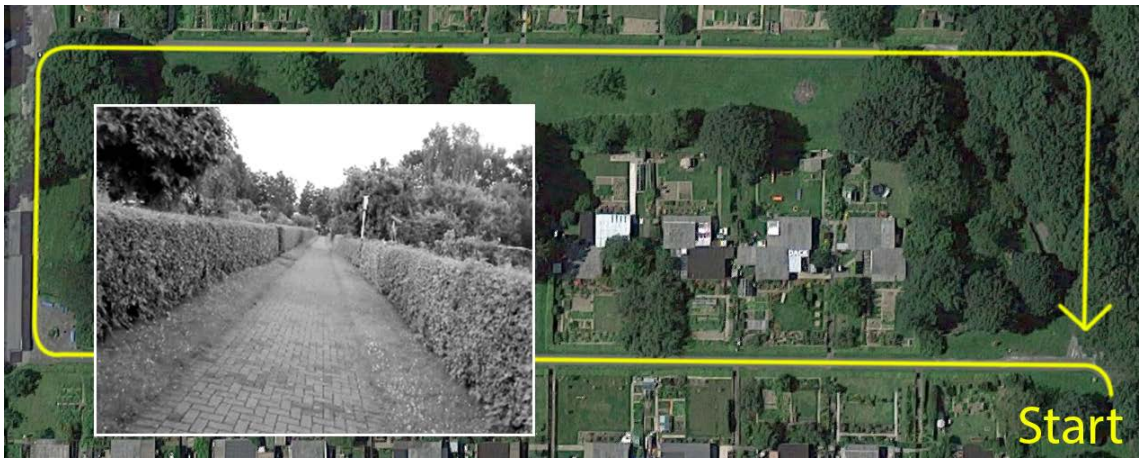


Abbildung 9.11.: Zweite Szene für die beispielhafte Auswertung der mit LSD-SLAM erhaltenen Trajektorie. Dargestellt ist die abgelaufene Strecke (gelb) in einer Gartenanlage.[2]

Modell

a) Detailliertes 3D-Modell mit mehreren Objekten
Aufbau: Es wurden zwei Würfel und ein dreiseitiger, allgemeiner Zylinder mit Lego gebaut und als CAD modelliert (siehe Abbildung 9.12).
Ablauf: Das Lego-Modell wurde mit dem Tiefensensor gefilmt und mit KinectFusion rekonstruiert. Danach wurde das Ergebnis mit den GroundTruth-Daten des CAD-Modells verglichen. Berechnet wurde der mittlere Fehler zwischen zwei korrespondierenden Punkten.

b) Rekonstruktion von Ebenen
Aufbau: Es wurden mehrere Abstände (50 cm, 1 m, 2 m und 3 m) zu einer weißen Wand gemessen und der Tiefensensor auf der gemessenen Entfernung positioniert. Zudem wurde die Wand als CAD modelliert.
Ablauf: Die Wand wurde aus den verschiedenen Entfernungen abgefilmt, wobei die Kamera dabei mit ca. 10° Neigung nach links, rechts, oben und unten bewegt. Die Daten wurden mit KinectFusion rekonstruiert und der mittlere Fehler zu den GroundTruth-Daten ermittelt. Des Weiteren wurde als Referenzwert die Werte der Abweichung aus der Spezifikation des Tiefensensors genommen [3].

2. **Genauigkeit der Registrierung**
Aufbau: In dem Zimmer befanden sich zufällig platzierte und gängige Objekte,

[3] `http://io.structure.assets.s3.amazonaws.com/structure_sensor_precision.pdf` Abruf: 22.06.16 13:00

unter anderem ein Schrank, eine Lampe und eine Heizung.
Ablauf: Das Zimmer (siehe Abbildung 9.13) wurde mit dem Tiefensensor abgefilmt und die Daten mit KinectFusion rekonstruiert. Jedoch wurde das Resultat in drei Bereiche aufgeteilt, wobei das Zimmer in links, mittig und rechts eingeteilt wurde und jeder Bereich einem KinFu-Volumen entspricht. Die Bereiche überlappen sich und wurden links bzw. rechts mit dem mittleren Bereich registriert.

9.2. Auswertung

Dieses Kapitel wertet die in Kapitel 9.1 beschriebenen Szenarien aus und evaluiert die Ergebnisse. Hier wird auch zuerst der ARGOS-Agent evaluiert, danach die Raumerkundung, das Tracking und zum Schluss die Umgebungsrekonstruktion. Einen Überblick und ein Fazit zu den Ergebnissen befindet sich in Kapitel 9.3.

9.2.1. ARGOS-Agent

Der ARGOS-Agent wird anhand der acht Szenarien, die in Abschnitt 9.1.1 aufgeführt sind, ausgewertet. Alle Versuche wurden bei einer stabilen WLAN-Verbindung zwischen Drohne und Computer ausgeführt. Das erste Szenario zeigt, ob die Daten der Sensoren richtig ausgelesen und übertragen werden. Zudem ist bei erfolgreicher Ausführen davon auszugehen dass die zugrundeliegenden technologischen Standards (Websockets, Json) eingehalten wurden. Dieses Szenario gelang ohne Probleme. Im zweiten Szenario wird die entwickelte Software-Schnittstelle getestet. Hierbei funktionierte ebenfalls der Empfang von Daten der Drohne sowie das Senden von Befehlen problemlos. Somit hält auch die Software-Schnittstelle die technologischen Standards ein und kann bei bestehender Verbindung mit der Drohne kommunizieren.
In den Szenarien 3 und 4 wird die Übertragung des Kamerabilds getestet. Dazu dient in Szenario 3 zunächst ein frei verfügbarer Videoplayer zum Abspielen des Videostreams. Da das Bild von dem Videoplayer in voller Auflösung und Bildrate angezeigt werden konnte und davon auszugehen ist, dass der Videoplayer die Technologie RTP korrekt implementiert hat, gelingt dieses Szenario. Die Drohne kann ein Videostream dem RTP-Standard entsprechend kodieren und über die WLAN-Verbindung freigeben. Im dazugehörigen vierten Szenario wird geprüft, ob auch die Software-Schnittstelle in der Lage ist, diesen Videostream zu dekodieren und auszu-

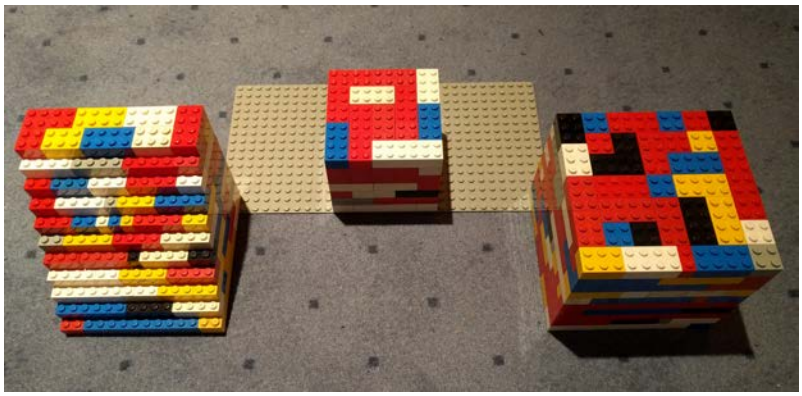


Abbildung 9.12.: Das Lego-Modell, welches zwei Würfel und ein Prisma modelliert und mit einem Tiefensensor abgefilmt wurde.

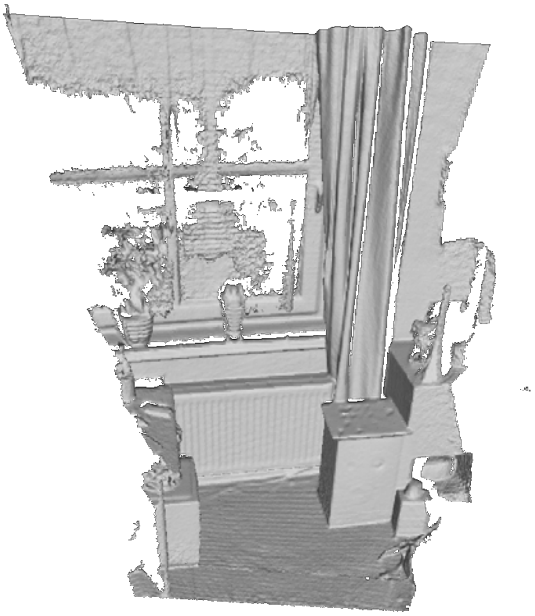


Abbildung 9.13.: Mit dem Tiefensensor abgefilmte Szene; zu sehen ist der Raum mit verschiedenen Objekten.

geben. Dieser Fall wurde erfolgreich mit einem Live-Video getestet.
Die Szenarien 5, 6 und 7 befassen sich mit der Frage zur Übertragung der 3D-Kameradaten. Im Szenario 5 wurde zunächst ein zuvor aufgenommenes Video auf einem USB-Stick der Drohne bereitgestellt. Die Drohne liest mithilfe der OpenNI2 Programmbibliothek das Video ein und komprimiert die Frames mithilfe der Tiefenbild-Kompression (vergleiche Abschnitt 7.6.4). Anschließend werden die komprimierten Frames über eine TCP-Verbindung an den Computer übertragen. Ein

erweiterter OpenNI2 Treiber sorgt dafür, dass der Videostream mit dem mitgeliefertem SimpleViewer betrachtet werden kann. Dieser Test wurde erfolgreich mit einem Video der Auflösung 640×480 Pixel und einer Bildrate von 30 Bildern pro Sekunden durchgeführt. Im nächsten Szenario 6 ändert sich im Versuchsaufbau lediglich die Quelle und die zu übertragende Auflösung. Die 3D-Kamera wird mit OpenNI2 direkt ausgelesen. Die halbe Kameraauflösung (320×240 Pixel) lässt sich ohne Probleme übertragen. Ein umschalten auf die volle Auflösung in Szenario 7 sorgt jedoch für einen Absturz der Kamera. Hier werden keine Bilddaten mehr an die Drohne weitergeleitet.
Szenario 8 bezieht sich auf die in den Szenarien 5 bis 7 für die Übertragung von Tiefenvideos eingesetzte Kompression. Um diese Kompression zu evaluieren, wurde mit der 3D-Kamera ein Testvideo erstellt , das mit verschiedenen Algorithmen komprimiert wurde. Das Video besteht aus sechs durch Kameraschwenks am Stück aufgenommenen Sequenzen, die sehr unterschiedliche Szenen zeigen und damit viele Aufnahmesituationen der Kamera im realen Einsatz abdecken. Die erste Sequenz zeigt eine Wand mit mehreren Fenstern aus einer Entfernung von 2 bis 3 Metern. Die zweite Sequenz zeigt dann aus geringer Entfernung einen Tisch, auf dem sich viele verschiedene Gegenständen befinden, und eine darüber hängende Lampe. Bei der dritten Sequenz bewegt sich die Kamera so nah zum Tisch, dass kein Tiefenbild mehr erzeugt wird. Dadurch entstehen Bilder, deren meiste Pixel den Wert 0 haben. Als vierte Sequenz folgt ein langer Schwenk über die Decke, der zu sehr gleichförmigen Bildern führt. In der fünften Sequenz schaut die Kamera in einen langen Flur, so dass auch einige weit entfernte Objekte aufgezeichnet werden. Die Bilder dieser Sequenz zeigen nahe und weit entfernte Objekte gleichzeitig. Abschließend zeigt die Kamera wieder den Tisch und dann die Decke des Raums, wobei auf der Decke ein starkes Rauschen entsteht, das in den vorangegangenen Sequenzen nicht zu sehen ist. Auf die Bilder dieses Videos wurden der eigene Kompressionsalgorithmus und die folgenden weiteren Algorithmen angewendet:

OpenNI2[4] 2.2 bietet als Teil des SDKs die Möglichkeit, die Tiefenbilder einer Kamera in dem proprietären ONI-Format komprimiert als Video zu speichern, das anschließend wie ein Eingabegerät behandelt werden kann. Dies ist das vom Hersteller der 3D-Kamera eingesetzte Verfahren. Dabei setzt das SDK einen einfachen, schnellen Kompressionsalgorithmus ein, der die Pixel des Bildes als Differenz zum jeweils vorangegangenen Pixel speichert, wobei in Abhängigkeit des Betrags der Differenz ein 4-, 8- oder 16-bit breites Wort für den Pixel ge-

[4] `http://structure.io/openni` Abruf: 22.06.16

speichert wird. Für undefinierte Pixel im Tiefenbild, denen von der Kamera der Wert 0 zugewiesen wird, setzt der Algorithmus zusätzlich eine Lauflängenkodierung ein. Sofern die Tiefe für die allermeisten Pixel des Bildes bekannt ist, liegt die Kompressionsrate daher bei mindestens 25%.

gzip[5] 1.8, bzip2[6] 1.0.6 und xz[7] 5.2.2 sind freie Kompressionsprogramme, die verschiedene Algorithmen einsetzen. gzip setzt den deflate-Algorithmus ein, bzip2 die Burrows-Wheeler-Transformation und xz den Lempel-Ziv-Markow-Algorithmus. Dateien werden dabei immer einzeln komprimiert, sofern sie nicht vorher mit einem anderen Werkzeug in einem Containerformat zusammengelegt wurden. Daher wurden mit diesen Programmen aus den in einzelnen Dateien abgelegten Bildern des Videos komprimierte Kopien erzeugt. Bei gzip wurde dies zweimal durchgeführt: zuerst mit der Standardeinstellung „`-6`“ und dann mit der schnellsten, aber schlechtesten Einstellung „`-1`“.

Info-ZIP[8] 3.0 erzeugt Zip-Dateien, die unter Microsoft Windows gebräuchlich sind. Das Zip-Format ist ein Container-Format, in dem eine beliebige Anzahl anderer Dateien in komprimierter Form abgelegt werden. In Abhängigkeit von den Eingabedaten und der konkreten Implementierung kommen dabei verschiedene Kompressionsalgorithmen zum Einsatz. Info-ZIP setzte für alle Bilder des Videos wie gzip den deflate-Algorithmus ein. Dementsprechend liefern Info-ZIP und gzip (mit der Standardeinstellung) ein nahezu identisches Ergebnis.

Alle Algorithmen wurden auf der Parrot AR.Drone2 ausgeführt, da diese die schwächste der zur Verfügung stehenden Hardware-Plattformen ist. Diese Drohne verfügt über einen mit 1GHz getakteten ARM Cortex-A8 Single-Core-Prozessor, 1 GB Arbeitsspeicher und läuft unter Linux 2.6.32. Die auf dieser Platform ermittelten Rechenzeiten sind daher Worst-Cases. Als Netzwerk bietet die Drohne ein IEEE 802.11n WLAN, wobei praktische Messungen auch unter sehr günstigen Bedingungen eine Netto-Bandbreite von 25MBit/s ergaben.
Die Funktionsfähigkeit des eigenen Algorithmus wurde durch einen byteweisen Vergleich der wieder entkomprimierten Bilder mit den Originalbildern sichergestellt. Dabei wurden alle Bilder verlustfrei wiederhergestellt, d. h. der Algorithmus erfüllt dieses Kriterium. Für die Echtzeitfähigkeit wurde festgelegt, dass der Algorithmus maximal 50% der Rechenzeit verbrauchen darf, um die restlichen Komponenten der

[7] `http://www.gnu.org/software/gzip/` Abruf: 30.06.16 14:00
[7] `http://www.bzip.org/` Abruf: 22.06.16 15:00
[7] `http://tukaani.org/xz/` Abruf: 22.06.16 12:00
[8] `http://www.info-zip.org/` Abruf: 22.06.16 12:00

Firmware selbst auf der AR.Drone2 nicht zu behindern. Bezogen auf ein 27 Sekunden langes Video bedeutet dies, dass maximal 13,5 Sekunden Rechenzeit verbraucht werden dürfen. Bei der Bestimmung der Netzwerkbandbreite wurden für die Übertragung der restlichen Videos und der Steuerung 8MBit reserviert, so dass auf der AR.Drone2 maximal 17MBit/s für das Tiefenvideo zur Verfügung stehen. Damit darf kein komprimiertes Tiefenbild größer als 70833 Bytes sein.

Die Ergebnisse der Videokompressionsläufe sind in Tabelle 9.1 aufgelistet. Hier zeigt sich, dass der eigene Algorithmus optimal ist, da er ausreichend schnell ausreichend gut komprimiert, d. h. auf dem Testvideo wurde die Echtzeitfähigkeit in Bezug auf Rechenzeit und Netzwerkbandbreite eingehalten bzw. deutlich unterschritten. xz erreicht das beste Kompressionsverhältnis, benötigt dafür aber mit Abstand die meiste Rechenzeit und könnte niemals in Echtzeit auf der Drohne ausgeführt werden. Obwohl bzip2 nur ein Drittel der Laufzeit von xz benötigt, ist das Kompressionsverhältnis kaum schlechter, aber auch dieser Algorithmus ist nicht echtzeitfähig auf der Drohne. gzip und Info-ZIP komprimieren deutlich schlechter als der eigene Algorithmus und benötigen dafür trotzdem noch die achtfache Rechenzeit. gzip ist selbst in der schnellesten Einstellung „`-1`“ nicht echtzeitfähig auf der AR.Drone2 und komprimiert dann ohnehin nicht mehr ausreichend, d. h. gzip und Info-ZIP sind ebenfalls ungeeignet. Der Algorithmus des OpenNI2-SDKs komprimiert mit $\geq 25\%$ nicht annähernd gut genug um eine Netzwerkübertragung zu ermöglichen, so dass er nicht weiter berücksichtigt wird.

Außerdem ist in Abbildung 9.14 für alle Bilder die durch den jeweiligen Algorithmus erzeugte komprimierte Größe in Bytes im zeitlichen Verlauf des Videos aufgetragen. Am einigermaßen parallelen Verlauf der Kurven zeigt sich bereits, dass der eigene Algorithmus („custom“) die Redundanz in den einzelnen Bildern, d. h. ihren Informationsgehalt, genauso gut ausnutzt wie die Standardverfahren. Außerdem entsprechen die Größenverhältnisse der verschiedenen Komprimate eines Bildes zueinander weitgehend den in Tabelle 9.1 dokumentierten Mittelwerten. Der in das OpenNI2-SDK integrierte Algorithmus ist nicht dargestellt, da er aufgrund seines schlechten Kompressionsverhältnisses ohnehin nicht in Frage kommt. Besonders auffällig am Kurvenverlauf ist das Tal bei $t \in [11..12]$. Hierbei handelt es sich um die dritte Sequenz des Videos, in der die Tiefenbilder nichts enthalten. Solange die Tiefenbilder Daten enthalten, schwankt die komprimierte Größe aber immer um den Mittelwert des jeweiligen Algorithmus – lediglich gzip erzeugt mehrere deutliche Ausreisser, die den Worst Case für diesen Algorithmus sehr schlecht ausfallen lassen.

Algorithmus	Kompression (in Bytes)			Rechenzeit	
	Worst	Gesamt	Rate	Zeit	Belastung
eigener Algorithmus	64240	37903468	7,68%	11,55s	43,15%
OpenNI2	–	131360968	26,63%	–	–
gzip -1	127383	66455871	13,47%	35,42s	132,33%
gzip -6	92761	40843071	8,28%	101,84s	380,47%
bzip2	67067	30006749	6,08%	699,22s	2612,28%
xz	62660	28102628	5,7%	1834,44s	6853,45%
Info-ZIP	92733	40803321	8,27%	103,06s	385,03%

Tabelle 9.1.: Kompressionsraten und verbrauchte Rechenzeit verschiedener auf Tiefenbilder angewendeter Algorithmen.

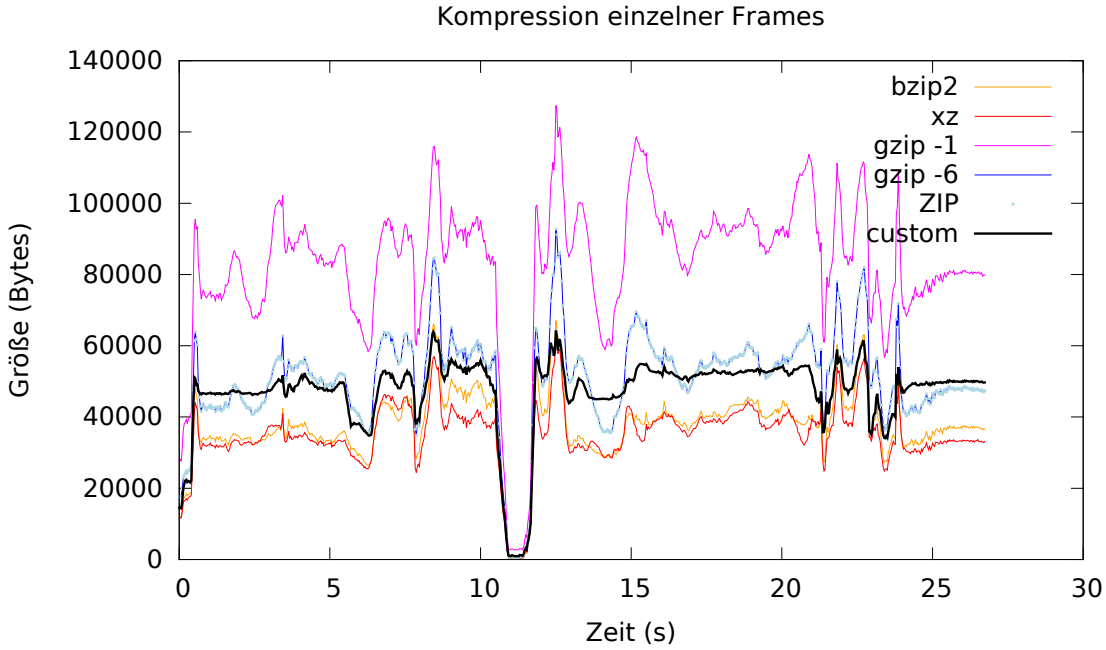


Abbildung 9.14.: Zeitlicher Verlauf der Kompressionsraten verschiedener Algorithmen über das Testvideo

9.2.2. Raumerkundung

Wie in Kapitel 9.1.2 beschrieben, wird die Raumerkundung durch drei Szenarien evaluiert.

Das erste Szenario evaluiert die Anzahl der Drehungen und die Flugdauer bei Änderung eines Hindernisses. Sobald die Drohne das Hindernis erreicht, versucht sie es zu umfliegen. Zuerst wird nach links geflogen, bis ein Schwellwert erreicht wurde. Danach dreht die Drohne nach rechts um und probiert das Hindernis dort zu umfliegen. Sobald auch hier der Schwellwert erreicht ist, wird dieser Wert erhöht

und die Drohne beginnt wieder von vorne. Dadurch benötigt die Drohne immer mehr Zeit um ein größer werdendes Hindernis zu umfliegen. Wie in Abbildung 9.15 zu sehen ist, verringert sich der Abstand bis zum Hindernis bei jedem Versuch konstant. Wie zu erwarten, benötigt die Drohne zum umfliegen des Hindernisses mehr Zeit, wenn das Hindernis größer wird.

Mit dem zweiten Szenario soll die Laufzeit der Pfadfindung evaluiert werden. Unter Berücksichtigung, dass die Zell- bzw. Voxelgröße nicht den Radius der Drohne unterschreiten darf und somit nicht unter grob 30 cm fällt, würden zu erkundende Räume oder Etagen mit Grundfächen jenseits von 400 Voxel × 400 Voxel eher als unwahrscheinlich einzustufen zu sein. Wie in Abbildung 9.16 zu sehen ist, liegt die Laufzeit bei dieser Raumdimensionierung im Bereich von 10 ms. Zu beachten ist, dass dabei eine Raumhöhe von 10 Voxel bzw. 3 m angenommen wurde. Eine derartige Laufzeit ist für den vorliegenden Anwendungsfall ausreichend. Durch die Wahl eines gröberen Rasters, also größerer Voxel, könnte die Zahl dieser und damit die Laufzeit jedoch optional zu Lasten der Genauigkeit reduziert werden.
Das letzte Szenario evaluiert die Anzahl der Drehungen und die zurückgelegte Strecke beim Erkunden eines Raumes. Dabei werden insgesamt sieben verschiedene Räume erkundet. Wie in Tabelle 9.2 zu sehen, benötigt der Algorithmus zum erkunden des Raumes weniger Zeit, wenn der Raum lang ist und mehr Zeit, wenn er breit ist. Da der Algorithmus rekursiv versucht eine Strecke zu finden und die Reihenfolge Nord, Ost, Süd und West einhält, werden weniger Wegpunkte angeflogen, wenn der Raum lang ist. Es werden also zuerst Nord/Süd-Strecken abgeflogen und wie in Abbildung 9.4 zu sehen ist, ist die abgeflogene Strecke sehr lang, bis die nördliche (oben) Wand erreicht wird. Hingegen wird bei einem breiten Raum die Anzahl der Wegpunkte deutlich größer, da der Algorithmus immer kurze Strecken fliegt, bis die nördliche bzw. südliche Wand erreicht wird. Um ein besseres Ergebnis bei breiten Räumen zu erzielen, müsste der Algorithmus abgeändert werden, was ihn aber ungleich komplexer machen würde. Die restlichen Räume erzielen akzeptable Ergebnisse, da die zurückgelegte Strecke pro Wegpunkt im Durchschnitt zwischen 160 m und 200 m liegt. Nur der breite Raum erzielt ein schlechtes Ergebnis mit 67 m zurückgelegter Strecke pro Wegpunkt.

9.2.3. Tracking

Wie in Kapitel 9.1.3 beschrieben, wird das Tracking anhand von vier Szenarien evaluiert. Im ersten Szenario wurde die Genauigkeit von LSD-Slam ausgewertet.

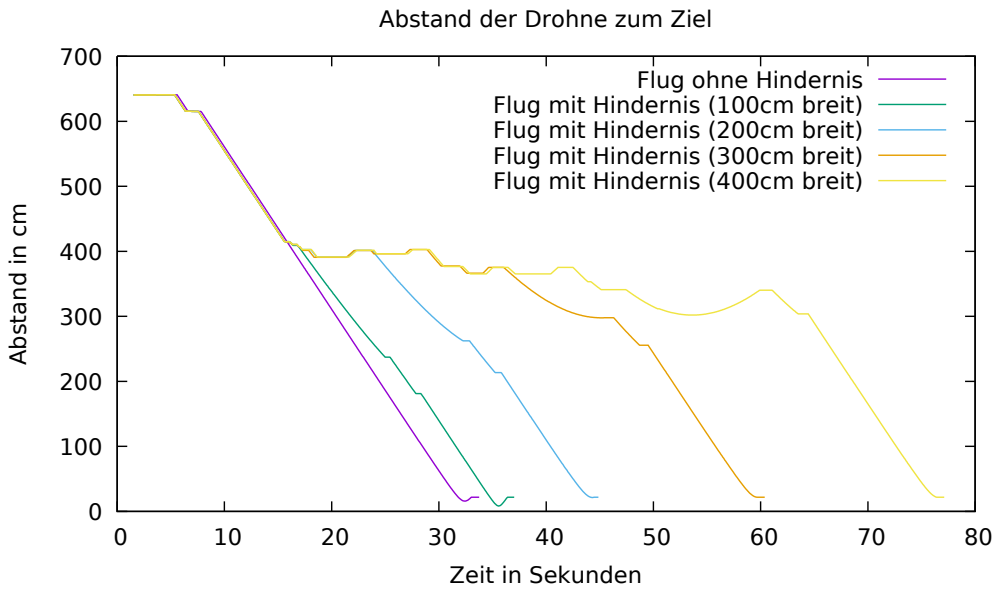


Abbildung 9.15.: Abstand der Drohne zum Ziel und Flugdauer, wobei die Größe des Hindernisses mit jedem Versuch um einen Meter verbreitert wird

Form	**Fläche**	**Strecke**	**Wegpunkte**	**Strecke/Wegpunkte**
Quadratischer Raum	640000 m^2	17420 m	52	335
Langer Raum	640000 m^2	29610 m	30	987
Breiter Raum	640000 m^2	13952 m	208	67,08
Vier Säulen	641700 m^2	15500 m	92	168,49
U-Form	640656 m^2	14696 m	84	174,95
L-Form	639000 m^2	15402 m	80	192,53
Parallele Hindernisse	641344 m^2	15722 m	116	135,53

Tabelle 9.2.: Auswertung des siebten Szenarios der Pfadplanung. Pro erkundeten Raum wurde die zurückgelegte Strecke und die Anzahl der Wegpunkte gemessen. Impliziert dreht sich die Drohne bei jedem Wegpunkt. Die letzte Spalte zeigt das Verhältnis zwischen Strecke und Wegpunkten.

Wie in der Tabelle 9.3 zu erkennen, wurde eine geringe durchschnittliche Abweichung gemessen. Wobei die Werte eine Abweichung zwischen dem Verhältnis zweier Distanzen in der realen Szene, zu dem Verhältnis der Distanzen in der von LSD-SLAM generierten Punktwolke sind.

Das zweite Szenario evaluiert die Robustheit von LSD-SLAM bei verschiedenen Lichtverhältnissen. Der Tabelle 9.4 und der Abbildung 9.18 ist zu entnehmen, dass das beste Ergebnis bei Tageslicht erzielt, welches quantitativ und qualitativ die besten Werte erreicht. Die Raumbeleuchtung erzielt das nächstbeste Ergebnis. Dieser Versuchsausgang war zu erwarten, da die Helligkeit des Lichts ausschlaggebend ist

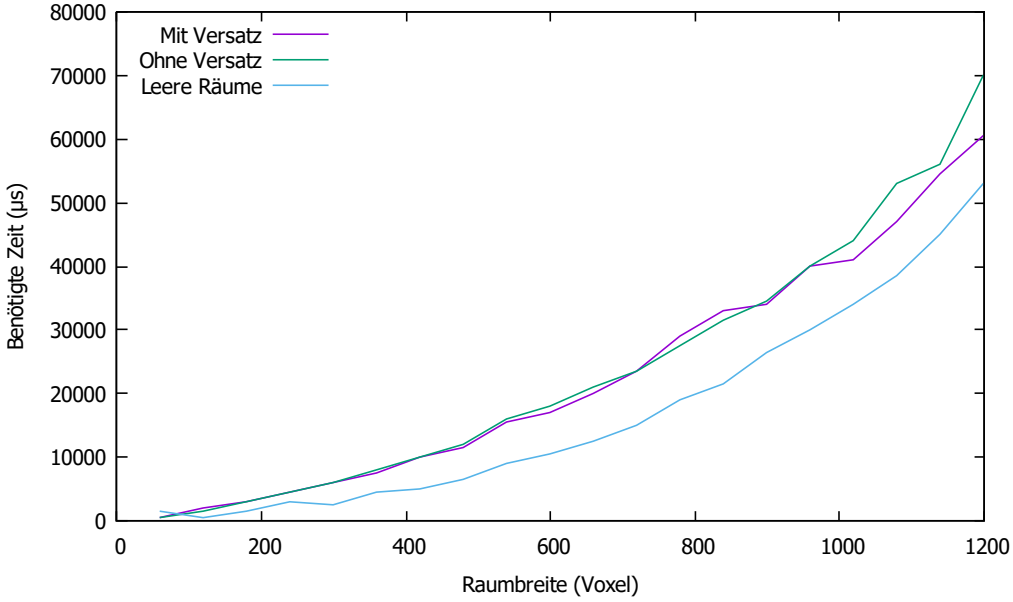


Abbildung 9.16.: Laufzeiten der Pfadfindung bei steigenden Raumgrößen

Durchschnittliche gemessene Abweichung	12%
Maximale gemessene Abweichung	26%
Minimale gemessene Abweichung	4%

Tabelle 9.3.: Abweichungen der Genauigkeit von LSD-SLAM

und das Szenario dies bestätigt hat.

Im dritten Szenario wurden die Abweichungen auf verschiedenen Entfernungen gemessen und die Daten in Abbildung 9.17 dargestellt. Die erzielten Werte sind gut bis optimal, wobei mit Ausreißern in der Abweichung, wie bei einer zurückgelegten Strecke von 100cm, zu rechnen ist. Die Extrema sind gering genug um keine fehlerhaften Ergebnisse zu erzielen. Die Genauigkeit ist ausreichend, da durch die Pfadplanung ein Sicherheitsabstand mit eingeplant wird.

Zum vierten Szenario wurden beispielhaft Trajektorien von zwei Szenen untersucht. Abbildung 9.10b zeigt die resultierende Trajektorie für die Wohnungs-Szene, die mit LSD-SLAM geschätzt wurde. Zu Beginn wurden langsame und kurze Bewegungen durchgeführt, die sich in der Trajektorie (blau umkreist) jedoch nicht korrekt wiederfinden lassen. Hingegen entspricht die Trajektorie im weiteren Verlauf der Strecke ungefähr dem tatsächlich abgelaufenen Pfad. Ein offensichtliches Problem lies sich im Zusammenhang mit einer spiegelnden Oberfläche erkennen (in der Trajektorie an der roten Umkreisung zu erkennen), bei der die geschätzte Pose kurzzeitig fehlerhaft

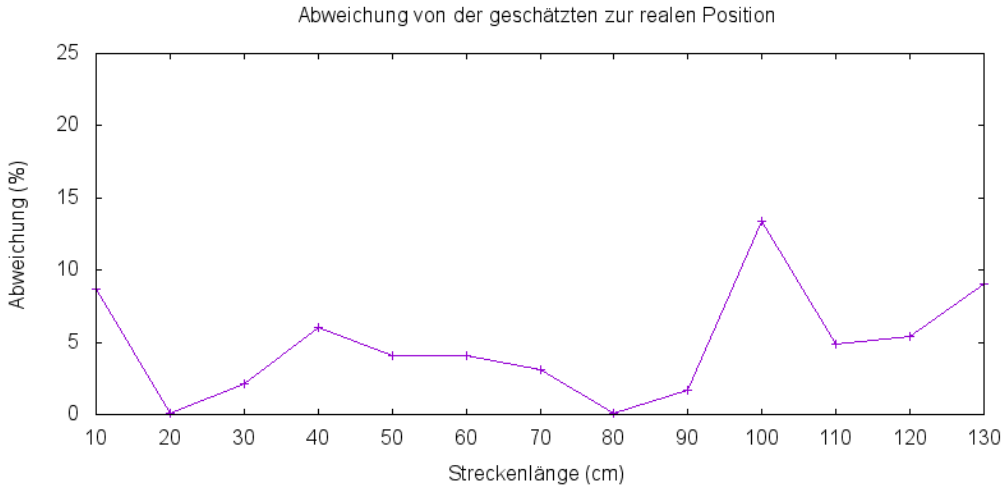


Abbildung 9.17.: Abweichung der geschätzten Position zur realen Position

Tiefenwerte pro Keyframe	
Abgedunkelt	21495
Raumbeleuchtung	27660
Tageslicht	31672

(a) Tiefenwerte pro Keyframe

Anteil der genutzten Tiefenwerte	
Abgedunkelt	69%
Raumbeleuchtung	71%
Tageslicht	85%

(b) Genutzte Tiefenwerte

Tabelle 9.4.: Auswertung der Robustheit von LSD-SLAM bei verschiedenen Lichtverhältnissen

war, was zu weiteren Problemen im Drohnen-Flug hätte führen können. Weitaus problematischer sind die Resultate zur zweiten Szene (Abbildung 9.20). Die abgelaufene Strecke besteht zu einem Großteil aus ähnlichen Ansichten, wodurch Selbstlokalisierung von Schwierigkeiten hatte (auch bei Deaktivierung der Relokalisation). Während der Start (Abbildung 9.20a, blau umkreist) eine einigermaßen akzeptable Schätzung geliefert hat, war der weitere Verlauf der Strecke fehlerbehaftet, was sich an der Trajektorie (rot umkreist) bemerkbar macht. Entsprechend ist die resultierende Punktwolke (Abbildung 9.20b) unbrauchbar für die Hinderniserkennung. Große Probleme können sich daher auch ergeben, wenn sich die Drohne beispielsweise in langen leeren Fluren fortbewegt. Angemerkt sei jedoch, dass die Ausführungen ohne den zusätzlichen Tiefen- und Odometrie-Daten durchgeführt wurden.

9.2.4. Umgebungsrekonstruktion

Wie in Kapitel 9.1.4 beschrieben, wird die Umgebungsrekonstruktion anhand verschiedener Szenarien evaluiert. Um die Genauigkeit zu evaluieren, wurden

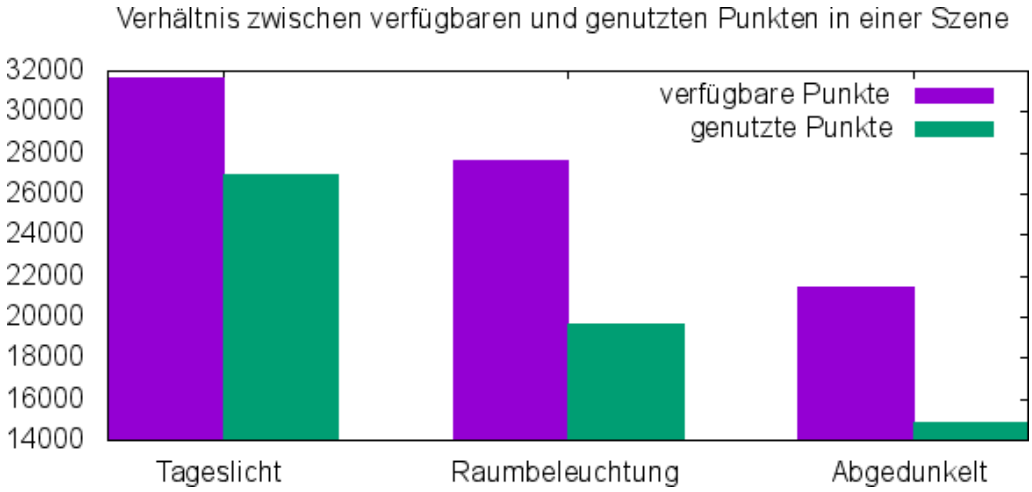


Abbildung 9.18.: Verhältnis zwischen verfügbaren und benutzten Punkten in der aktuellen Szene

(a) Tageslicht **(b)** Raumbeleuchtung **(c)** Abgedunkelter Raum

Abbildung 9.19.: Kamerabilder und LSD-SLAM-Visualisierung der Daten

verschiedene Szenen aufgenommen, registriert und der mittlere Fehler zu einem Computer-Aided Design (CAD), welches als GroundTruth-Modell benutzt wird, berechnet.

Das erste Szenario wurde in zwei Unterszenarien aufgeteilt. Im ersten Unterszenario wurde ein detailliertes 3D-Modell mit mehreren Objekten gewählt, welches in Abbildung 9.12 zu sehen ist. Die Aufnahmen des Tiefensensors waren nicht genau genug. Ein Problem war die Größe der Würfel. Da diese klein waren, war eine präzise Rekonstruktion als Würfel nicht möglich. So wurden die Würfel teilweise rundlich rekonstruiert, wie in Abbildung 9.21 zu sehen ist. Dennoch konnte der Euklidische

(a) geschätzte Trajektorie **(b)** resultierende Punktwolke

Abbildung 9.20.: Resultierende Trajektorie und zugehörige Punktwolke für die zweite Szene, in der ein Gehweg abgelaufen wurde.

Abstand zum korrespondierenden 3D-Modell berechnet werden.

Wie der Tabelle 9.5 zu entnehmen ist, wurden hohe Abweichungen berechnet. Das folgt aus dem ungenauen Modell, welches von KinFu rekonstruiert wurde. Ein besseres Ergebnis konnte das zweite Unterszenario erzielen, welches in Kapitel 4.5d beschrieben wurde. Als Referenzwert wurde die Spezifikation des Tiefensensors gewählt [9]. Die dort angegebenen Abweichungen sind ab 2 Meter Abstand zum Objekt schlechter als die gemessenen Abweichungen, welche in Tabelle 9.6 dargestellt werden.

Das vergleichsweise gute Ergebnis lässt sich auf die strukturierte Szene zurückführen, denn die abgefilmte Wand wurde mit KinFu sehr gut rekonstruiert, da eine Wand nahezu perfekt als Ebene rekonstruiert wird. Bei den Aufnahmen wurden auch andere Objekte gefilmt, wie bspw. ein Tisch. Diese wurden vor der Registrierung ausgeschnitten, um eine einheitliche Wand zu erhalten, da das 3D-Modell nur eine Wand darstellt und zusätzliche Objekte das Ergebnis verfälschen würden.

Im zweiten Szenario wurde die in Kapitel 9.1.4 beschriebene Szene rekonstruiert und die Registrierung zwischen verschiedenen KinFu-Volumen evaluiert. Dabei wurde die Szene in mehrere Bereiche aufgeteilt und registriert. Hier wurden drei Bereiche ausgewählt, welche als links, mittig und rechts beschrieben werden können. Es wurde der mittlere Bereich der Szene mit dem linken und mit dem rechten Bereich registriert (siehe Abbildung 9.23). Das Ergebnis im rechten Bereich ist vergleichsweise schlechter ausgefallen, was sich auf die Objekte in der Szene zurückführen lässt (siehe

[9] `http://io.structure.assets.s3.amazonaws.com/structure_sensor_precision.pdf` Abruf: 22.06.16 12:00

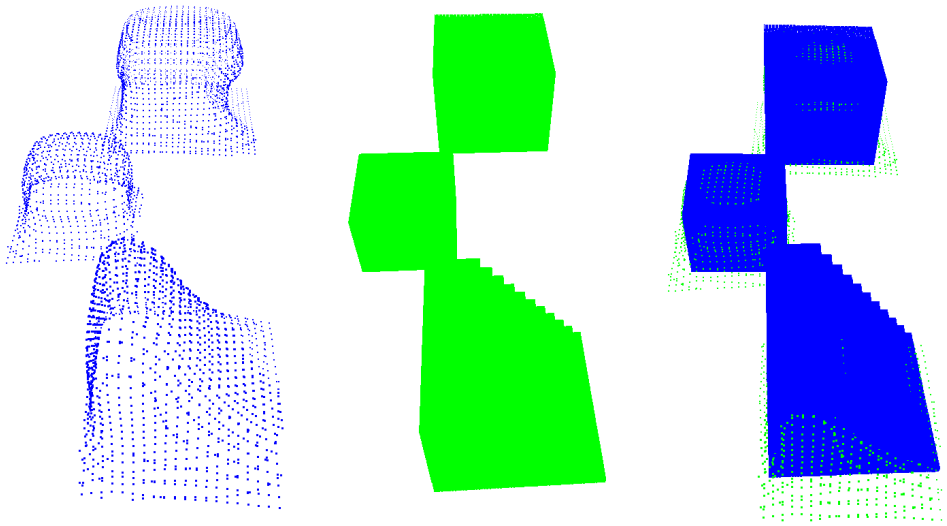


Abbildung 9.21.: Vergleich zwischen dem registrierten Modell (links) und dem Ground-Truth-Modell (mitte). Das rechte Bild zeigt beide Modelle übereinandergelegt. Zu sehen ist das Bild aus einer links-seitigen Perspektive.

Mittlerer Abstand in Meter	Standardabweichung
0.0059	0.0057%

Tabelle 9.5.: Abstand der korrespondierenden Punkte zwischen dem 3D-Modell und dem rekonstruierten Modell von Szenario 1a.

Abbildung 9.13). Obwohl GICP ein schlechteres Ergebnis erzielt, ist das subjektive Wahrnehmen der Szene oft besser als ICP. Gestützt wird diese Aussage durch Abbildung 9.23. Dort ist zu erkennen, dass GICP ein subjektiv besseres Ergebnis erzielt, da die einzelnen Ebenen in der Szene besser übereinander liegen und weniger Abweichung existiert.

9.3. Diskussion

In diesem Kapitel wird die Evaluierung zusammengefasst und die Ergebnisse bewertet. Da die einzelnen Ergebnisse der verschiedenen Testszenarien bereits in Kapitel 9.2 ausgewertet und analysiert wurden, wird in diesem Kapitel übergreifende Ergebnisse bewertet. Zuerst werden die vier Teilbereiche, im folgenden Module genannt, bewertet. Es wird bewertet ob die Ergebnisse aller Szenarien eines Moduls ausreichend sind, damit andere Module diese nutzen können. Zudem wird diskutiert wie die Ergebnisse verbessert werden können und welche Vor- oder Nachteile diese er-

Abstand zur Wand	Standardabweichung
50cm	0,122%
100cm	0,341%
200cm	1,315%
300cm	2,45%

Tabelle 9.6.: Abstand der korrespondierenden Punkte zwischen dem 3D-Modell und dem rekonstruierten Modell von Szenario 1b.

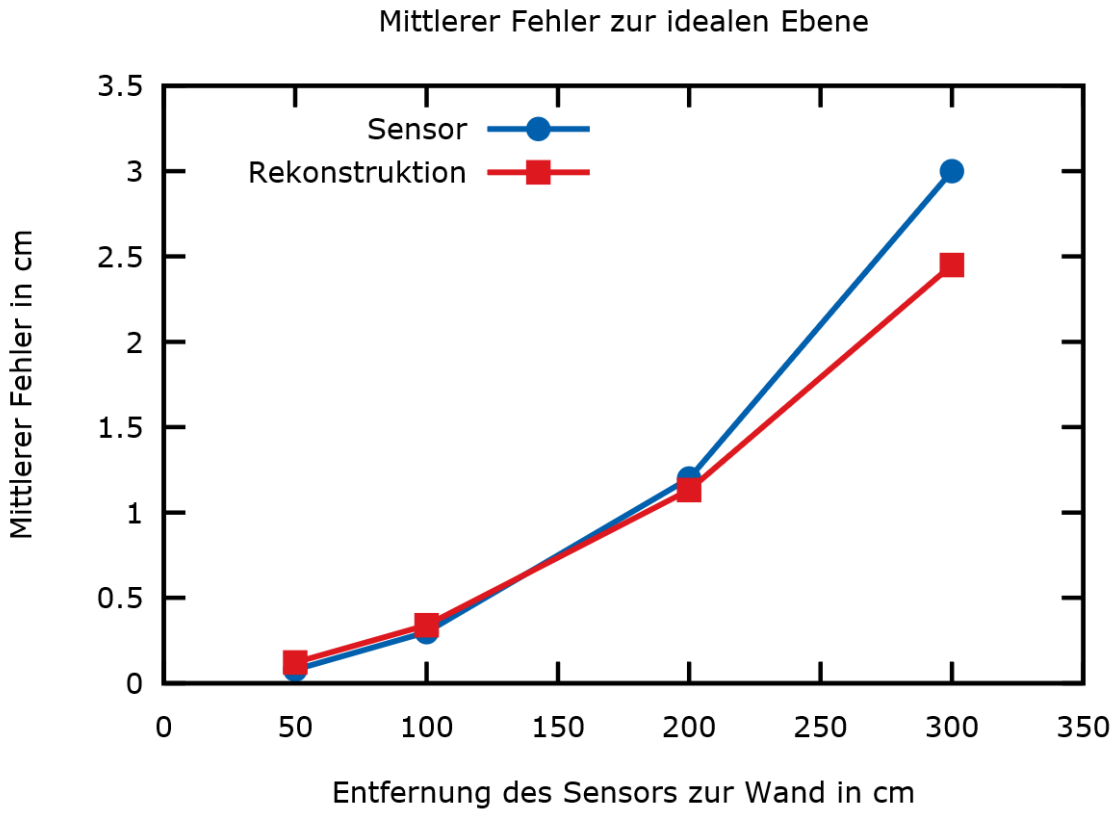


Abbildung 9.22.: Vergleich der berechneten Standardabweichung des KinFu-Verfahrens und den spezifizierten Daten des Tiefensensors.

zeugen können. Danach wird ein Fazit zur Evaluierung gemacht.

ARGOS-Agent Die Auswertung des ARGOS-Agents hat ergeben, dass alle Daten bei einer bestehenden WLAN-Verbindung zuverlässig übertragen werden. Die Odometrie-Sensordaten werden von der Drohne technologiekonform versandet und auch von der Software-Schnittstelle korrekt ausgelesen. Befehle gelangen von der Software-Schnittstelle zuverlässig zur Drohne. Der Videostream der Drohne kann von dem Client betrachtet werden. Einzig das Übertragen der vollen Auflösung der 3D-Kamera bereitet Probleme, was jedoch auf ein Hardware- oder Treiber-Problem der Kamera zurückzuführen ist. Außerdem erfüllt der

	ICP	GICP
Mittlerer Abstand (links)	2.071cm	1.45cm
Minimaler Abstand (links)	0.0008cm	0.0006cm
Maximaler Abstand (links)	18.824cm	20.376cm
Mittlerer Abstand (rechts)	2.459cm	2.0495cm
Minimaler Abstand (rechts)	0.0135cm	0.0104cm
Maximaler Abstand (rechts)	20.123cm	21.31cm
σ_X (links)	2,112%	2,187%
σ_X (rechts)	2,382%	2,758%

Tabelle 9.7.: Genauigkeit der Registrierung anhand von verschiedenen KinFu-Volumen.

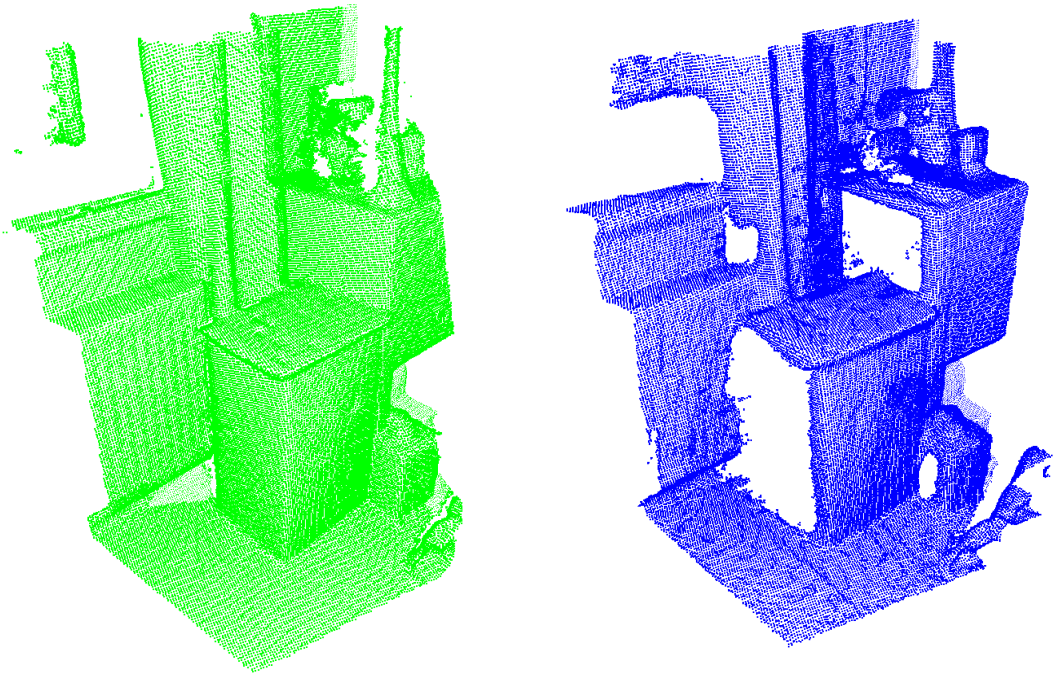
Registrierungsverfahren	Standardabweichung (links)	Standardabweichung (rechts)
ICP	2,112%	2,381%
GICP	2,187%	2,759%

Tabelle 9.8.: Genauigkeit der Registrierung anhand von verschiedenen KinFu-Volumen.

entwickelte Kompressionsalgorithmus die an ihn gestellten drei Kriterien der Funktionsfähigkeit, Echtzeitfähigkeit und Einhaltung der Netzwerkbandbreite. Er ermöglicht es, die Bilder der 3D-Kamera in der benötigten Qualität in Echtzeit zu übertragen. Dies wäre weder mit Standardalgorithmen noch mit dem vom Hersteller eingesetzten Algorithmus möglich.

Raumerkundung Die erzielten Ergebnisse der Raumerkundung sind ausreichend, um in akzeptabler Zeit einen Raum zu erkunden. Wie bereits diskutiert, braucht der Algorithmus zum Erkunden eines Raumes ungleich mehr Zeit, wenn der Raum länglich ist und dadurch kurze Nord/Süd-Strecken aufweist. Eine Verbesserung des Algorithmus wäre nötig, doch dadurch wird der simple Algorithmus sehr komplex und braucht möglicherweise wesentlich mehr Rechenzeit. Weiterhin umfliegt die Drohne ein großes Hindernis durch einfaches Probieren, bis definierte Schwellwerte erreicht sind. Um die Flugzeit beim umfliegen von Hindernissen zu verkürzen, könnten Kamerabilder analysiert werden und damit die Schwellwerte angepasst werden. Das resultierende Problem ist die benötigte Rechenzeit, welche schon durch die anderen Module stark limitiert ist.

Tracking Das Tracking erzielt Ergebnisse, welche unter idealen Umständen gut ausfallen. In den Punktmengen lässt sich die erkundete Umgebung gut wiedererkennen, jedoch fällt sie um so schlechter aus, wenn spiegelnde Oberflächen oder leere Wände berücksichtigt werden sollen. Verbesserungen könnten mit der

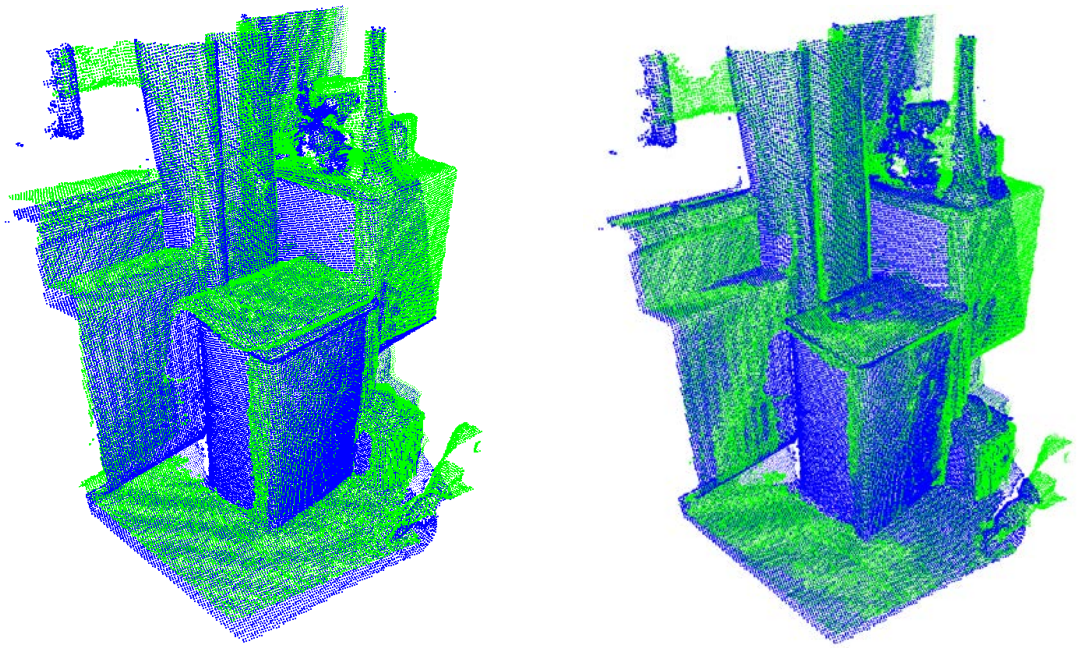


(a) Linkes Volumen **(b)** Mittleres Volumen

Abbildung 9.23.: Visualisierung der KinFu-Volumen des Raumes aus dem zweiten Szenario. Zu erkennen sind fehlende Flächen im mittleren Volumen, welche durch die Registrierung gefüllt werden sollen.

Verwendung einer anderen Kamera erzielt werden, welche ein höheres Sichtfeld, sowie eine höhere Bildrate liefern kann (siehe Kapitel 4.4), was jedoch eine zusätzlichen Traglast für die Drohne bedeuten würde. Weitere Daten, wie die Tiefensensor-Daten und die Drohnen-Odometrie, wurden eingebunden, um Probleme beim Tracking und der Initialisierungs-Phase vorzubeugen. Weiterhin soll mit ihnen korrekte metrische Distanzen ermittelt werden. Die Evaluierung in Verwendung dieser Daten fand allerdings unter gegebenen Umständen mit der Drohnen-Hardware nie statt.

Umgebungsrekonstruktion Die Auswertung der Szenarien der Umgebungsrekonstruktion erzielte unterschiedliche Ergebnisse. Im Ganzen werden Objekte in einer echten Szene, wie einem Zimmer, gut dargestellt, sodass diese wiedererkennbar sind. Hingegen werden schlechte Ergebnisse bei kleinen Objekten, welche in kleinem Abstand zueinander gefilmt werden, erzielt. Diese Ergebnisse können auf Hardware-Ebene durch einen besseren Tiefensensor verbessert werden. Das Problem ist, dass andere Tiefensensoren, welche auch bessere Qualität bieten, größer und schwerer sind als der benutzte Tiefensensor und damit



(a) Registrierung mit ICP **(b)** Registrierung mit GICP

Abbildung 9.24.: Die in Abbildung 9.23 visualisierten Volumen wurden mit ICP und GICP registriert. Zu erkennen sind subjektiv bessere Ergebnis durch GICP, da die Ebenen besser übereinander liegen und weniger Abstand dazwischen existiert.

nicht auf die Drohne montiert werden können. Ein Beispiel ist der Tiefensensor Kinect von Microsoft [10], welcher mit einem Gewicht von 1,4 kg zu schwer für die Drohne ist. Da die Rohdaten des Tiefensensors nicht ausreichend gut genug sind, kann auf der Software-Ebene keine effiziente Verbesserung vorgenommen werden.

Alle Module liefern verwertbare Ergebnisse, damit diese von anderen Modulen genutzt werden können. Die Szenarien konnten alle erfolgreich ausgeführt werden und lieferten gute Ergebnisse.

[10] `https://developer.microsoft.com/en-us/windows/kinect/hardware` Abruf: 28.06.16 13:00

10. Zusammenfassung und Ausblick

Im folgenden Kapitel werden die Ergebnisse der Arbeit der Projektgruppe zusammengefasst und anschließend ein kurzer Ausblick über Möglichkeiten der Verbesserung gegeben.

10.1. Zusammenfassung

Drohnen sind ein aktuelles Thema, für private Konsumenten bezahlbar geworden und ihre Anwendungen finden sich in unterschiedlichen Gebieten wieder. Die Projektgruppe beschloss, eine Software zu erstellen, mit der eine Flugdrohne autonom in geschlossenen Räumen gesteuert werden kann, sodass sie dabei genügend Datenmaterial für eine dreidimensionale Rekonstruktion der gesichteten Umgebung sammelt. Diese Repräsentation kann dann etwa in der Architektur oder in Rettungsaktionen genutzt werden. Um diese Aufgabe zu lösen, wurde die Gruppe in vier kleinere Teilgruppen eingeteilt, die jeweils einen Teilaspekt der Lösung behandeln.
Die Softwarekomponente der Drohnen-Hardware übernimmt ihre Steuerung, indem sie Zielpunkte von einem externen Rechner entgegen nimmt und diese anfliegt. Schwierigkeiten fanden sich in der Sensordatenauswertung aufgrund der unbekannten zeitlichen Zuordnung, sowie in der Ansteuerung der Drohnen-Komponenten. Zu den übermittelnden Daten gehören Odometrie und Tiefendaten, die als Stream über RTSP/RTP gesendet und komprimiert werden.
Die Navigation der Drohne musste so konstruiert sein, sodass eine Erkundung des Gebietes statt findet. Um eine Datensammlung von möglichst viel Fläche zu gewährleisten, geschieht dies mit Verfahren, welche die abgeflogenen oder gesehenen Gebiete vermerkt und iterativ bekannte verbleibende Orte als nächste Ziele wählt. Initial beläuft die Konstruktion des Ablaufplans auf dem bekannten Grundriss. Die Pfadplanung ist dynamisch, sodass während des Fluges erkannte Hindernisse berücksichtigt werden. Eine zusätzliche Simplifizierung des Pfades vermeidet unnötige Rotationen der Drohne.
Die Aufgabe der Selbstlokalisierung übernimmt LSD-SLAM, die zugleich die Erken-

nung von Hindernissen liefert. Da das Verfahren auf Basis eines Bildstroms arbeitet, sind reflektierende Oberflächen und leere Wände ein Problem. Weiterhin sind tatsächliche metrische Distanzen nicht bekannt, weshalb zusätzliche Odometrie-Daten, die auf der Drohne anhand von Sensordaten berechnet werden, in den Posen-Graph eingebunden sind. In der Initialisierungs-Phase wird als Hilfestellung direkt von Tiefendaten des angebundenen Tiefensensors Gebrauch gemacht.
Die Rekonstruktion verläuft in einem separaten Verarbeitungs-Schritt und nimmt die gesammelten Tiefendaten zur Grundlage. Als Rahmen für das Tracking und Mapping auf den Tiefendaten dient KinectFusion. Hierbei werden Tiefendaten einzelner Frames über ICP am bereits berechneten Modell ausgerichtet. Die zu rekonstruierende Szene wird dabei als Distanzfeld repräsentiert. Ein Problem stellt die durch den Grafikspeicher limitierte Größe der Szene dar. Daher wurde das Verfahren erweitert, um auch die Rekonstruktion großer Räume zu ermöglichen. Um bei einem möglichen Fehlschlag des Trackings trotzdem ein verwendbares Modell zu erzeugen, wurde die punktwolkenbasierte Registrierung und Oberflächenrekonstrukion aus Teilmodellen diskutiert.
Dem Benutzer ist eine manuelle Steuerung gegeben. Weiterhin kann die Position und Ausrichtung der Drohne während des Drohnen-Fluges in einer Weboberfläche beobachtet werden.

10.2. Ausblick

In der Arbeit traten Probleme mit der Positionserkennung der Drohne auf. Dies war bedingt durch die Odometrie und ließe sich durch eine Ergänzung, bzw. Ausbesserung in diesem Bereich verbessern. Alternativen zur Odometrie sind u. a. Optical Flow, Markerbasierte Odometrie oder eine verbesserte Sensorik. Der Vorteil des Optical Flow liegt darin, dass die notwendige Hardware, eine zum Boden gerichtete Kamera. Die in diesem Porjekt verwendete Drohne verfügt bereists über eine solche Kamera. So könnte die Erkennung verbessert werden ohne zusätzliche Hardware oder Hilfsmittel verwenden zu müssen. Eine Odometrie basierend auf externen, im Raum verteilten Markern ist im Vergleich zur aktuellen Odometrie genauer, kann jedoch nicht dynamisch verwendet werden, da eine Vorbereitung vor der Erkundung notwendig ist.
Damit die Drohne auch Erkundungen in unbekannten Umgebungen durchführen kann, muss die Software so angepasst werden, dass sie auf einen Grundriss verzichten kann. Dafür muss die Hinderniserkennung um eine Erkennung von Wänden, welche

zumeist eine große Flächen ohne erkennbare Struktur darstellen, erweitert werden. Ein Ultraschallsensor könnte beispielsweise diesbezüglich die benötigten Daten zur Erfassung von Wänden sammeln. Ebenso muss die Erkundungsplanung insofern angepasst werden, dass sie nicht mehr von dem zur Verfügung gestellten Grundriss bedingt ist, sondern eigenständig einen Umgebungsplan erstellt, mithilfe dessen die abzufliegenden Routen ermittelt werden. Zusätzlich muss ein Abbruchkriterium festgelegt werden, welches die Größe der Umgebung begrenzt.
Das verwendete Rekonstruktionsverfahren arbeitet lediglich auf den bereitgestellten Tiefendaten. Grundsätzlich erlaubt das Verfahren aber auch die Integration von Farbe. Dazu ist es nötig, dass im Bezug auf die Tiefenkamera registrierte Farbinformation zur Verfügung stehen. Die Farbinformationen der Frontkamera können zu diesem Zweck registriert und verwendet werden. Um die Genauigkeit der Rekonstruktion gerade für sehr große Räume zu verbessern, könnte außerdem Moving Volume KinectFusion, ähnlich zu LSD-SLAM, um ein Verfahren für das „loop closing“erweitert werden. Außerdem würde sich die zuvor erläuterte Verbesserung der Drohnenodometrie ebenfalls positiv auf die Robustheit der Rekonstruktion auswirken.

A. Anhang

A.1. Pflichtenheft

inStaNt-drone - Grafisch interaktive Steuerung und autonome Navigation von Drohnen

Pflichtenheft

Bernd das Brot in Kollaboration mit PG591

Sommersemester 2015, Wintersemester 2015/2016
Entwurf, 12. Januar 2016

Inhaltsverzeichnis

1 Zielbestimmungen **1**
- 1.1 Musskriterien . . . 3
- 1.2 Wunschkriterien . . . 5
- 1.3 Abgrenzungskriterien . . . 6

2 Produkteinsatz **8**
- 2.1 Anwendungsbereiche . . . 8
- 2.2 Zielgruppen . . . 8
- 2.3 Betriebsbedingungen . . . 8

3 Produktumgebung **10**
- 3.1 Software . . . 10
- 3.2 Hardware . . . 10

4 Produktfunktionen **11**
- 4.1 Drohnenbewegung . . . 11
- 4.2 Serveranwendung . . . 12

5 Produktdaten **13**

6 Produktleistungen **14**

7 Benutzeroberfläche **15**
- 7.1 Dialogstruktur . . . 15
- 7.2 Layout . . . 15

8 Qualitätsbestimmungen **17**

9 Globale Testszenarien und Testfälle **18**

10 Entwicklungsumgebung **22**
- 10.1 Software . . . 22
- 10.2 Hardware . . . 22
- 10.3 Orgware . . . 23

1 Zielbestimmungen

Dieses Pflichtenheft beschreibt die Anforderungen an das zu entwickelnde Endprodukt namens „inStaNt-drone - Grafisch **in**teraktive **St**euerung und **a**utonome **N**aviga**t**ion von **Dro**h**ne**n“ der Projektgruppe 591 an der Technischen Universität Dortmund.

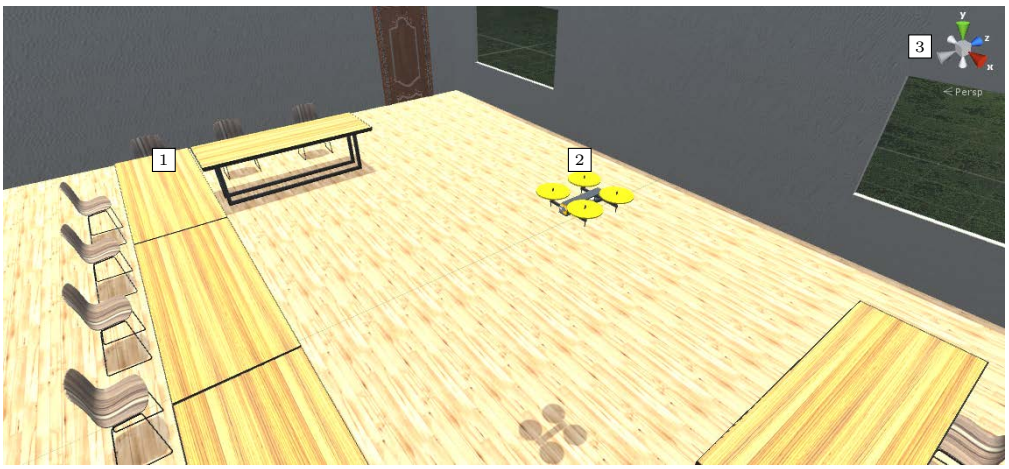


Abbildung 1.1 – Eine schematische Darstellung einer Drohne in einem Seminarraum.

1. Hindernisse im Raum
2. Drohne
3. Achsen im Raum

Abbildung 1.1 zeigt einen Screenshot eines Simulators, der das Testen verschiedener Algorithmen zur Erkundung von Räumen erlaubt. Der Screenshot soll die Zielsetzung der Projektgruppe verdeutlichen. Ziel der Projektgruppe ist es, eine Desktopanwendung zu entwickeln, mit der es möglich ist eine Parrot Bepop anzusprechen, Daten zu empfangen und Daten sowie Steuerbefehle zu senden. Die Drohne soll die Umgebung, welche in Abschnitt 2.1 näher beschrieben wird, selbstständig abfliegen können. Aus den empfangenen Daten soll es den Benutzern möglich sein, die abgeflogene Umgebung als 3D-Modell zu rekonstruieren. Um dies zu ermöglichen, wird auf der Drohne eine angepasste Firmware aufgespielt, welche ebenfalls von der Projektgruppe entwickelt wird. Die Drohne wird zusammen mit einer Auflistung der verwendeten Sensorik in Abschnitt 3.2 näher beschrieben.

Abbildung 1.2 zeigt schematisch die Verbindung zwischen der verwendeten Parrot Bebop mit dem zusätzlich montierten Structure-Sensor und der PC-Anwendung auf einem Desktop-PC über eine WLAN-Verbindung gemäß der Angaben der Produktumgebung.

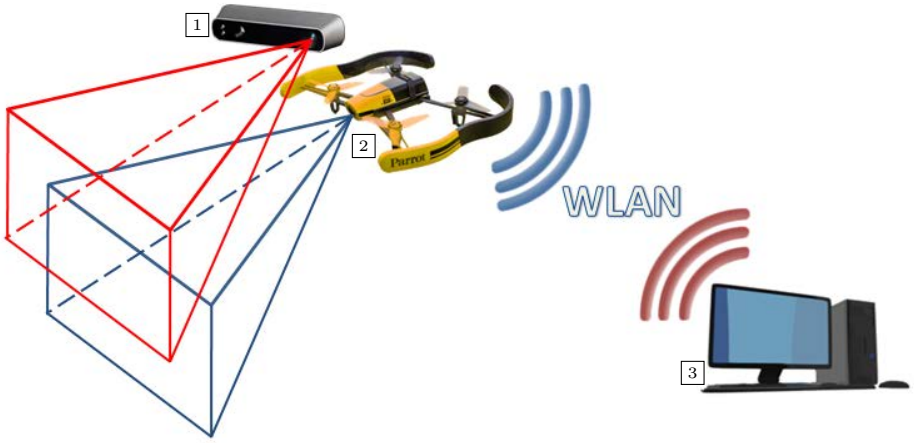


Abbildung 1.2 – Kontakt der Drohne und der Anwendung auf einem Desktop-PC über WLAN.

1. zusätzlicher Sensor an der Drohne: Structure Sensor
2. Drohne: Parrot Bebop
3. Desktop-PC mit den installierten PC-Anwendungen

Dabei dienen die Kamera der Drohne, sowie der Structure-Sensor als optische Aufnahmemöglichkeit. Neben einer optischen Erfassung wird die Umgebung über mehrere in der Drohne integrierten Sensoren, die in Abschnitt 3.2 aufgeführt werden, erfasst.

Auf der Drohne soll eine angepasste Firmware genutzt werden, die eine Kommunikation mit der Drohne über ein Netzwerk ermöglicht um beispielsweise Steuerbefehle zu senden oder aufgenommene Umgebungsdaten zu empfangen. Für die Steuerung der Drohne und der Erkundung sowie für die Rekonstruktion der erfassten Umgebung soll eine Anwendungen programmiert und auf dem Desktop-PC installiert werden. Die zu programmierende Anwendung ist in zwei Aspekte unterteilt. Der erste Aspekt bezieht sich auf den Kontakt zur Drohne und soll dabei die Erkundung der Umgebung steuern und der benutzenden Person eine Statusanzeige mit einer Karte und aktuellen Sensordaten bieten. Der zweite Augenmerk der Anwendung liegt auf der Rekonstruktion der durch die Erkundung gewonnenen Daten und soll aus während aufgenommenen Tiefendaten des Structure-Sensors die Umgebung zu einem 3D-Modell rekonstruieren und darstellen.

In diesem Pflichtenheft werden 3-Tupel aus reellen Zahlen, die Abstandsangaben zu einem zuvor festgelegten Fixpunkt in der jeweiligen Koordinatenebene in Zentimetern angeben, als Koordinaten bezeichnet. Die Achsen im Raum, die für die Angabe von Koordinaten notwendig sind, werden in Abbildung 1.1 bei Punkt 3 dargestellt.

In Abschnitt 1.1 werden die Kriterien, die durch die Projektgruppe auf jeden Fall umgesetzt werden sollen, erläutert. Die Wunschkriterien, die nicht notwendigerweise umzusetzen sind, werden in Passus 1.2 aufgelistet und beschrieben. Aspekte, die innerhalb der

Projektgruppe ausdrücklich nicht umgesetzt werden, werden in Abschnitt 1.3 beschrieben.

1.1 Musskriterien

/K010/ Der Drohne soll ein selbstständiger Flug ermöglicht werden, sodass diese, ohne menschliche Interaktion, vorgegebene Positionen im Raum anfliegt und im Falle keiner weiteren Befehle auf der aktuellen Position schwebend verweilt.

/K 020 / Den entwickelnden Personen soll eine API bereitgestellt werden, mit der die Drohne während des Flugs manuell gesteuert werden und wichtige Informationen ausgelesen werden können. Die folgenden Punkte (/K021/ bis /K029/) listen Funktionen der API auf.

/K 021 / Über die API soll die Drohne zu einer vorgegebenen Koordinate gesteuert werden können.

/K022/ Über die API soll die Drohne um $\alpha, mit -180° \leq \alpha \leq 180°$ gedreht werden können.

/K023/ Über die API soll die Drohne manuell gesteuert werden können. Dabei existieren die Bewegungsrichtungen: vorwärts, rückwärts oder seitwärts.

/K 024/ Über die API soll die Bewegungsgeschwindigkeit der Drohne steuerbar sein.

/K 025 / Über die API sollen die dekodierten Bilder der Frontkamera bereitgestellt werden.

/K026/ Über die API sollen alle Daten des Structure-Sensors bereitgestellt werden.

/K027/ Über die API sollen alle weiteren verfügbar gemachten Sensordaten bereitgestellt werden.

/K028/ Über die API soll die geschätzte Position und Ausrichtung der Drohne bereitgestellt werden.

/K029/ Über die API soll eine Möglichkeit zur Übermittlung von Rückmeldungen der Datenverarbeitung an die Drohne bereitgestellt werden. Dadurch soll insbesondere die Navigation der Drohne präziser werden und die Drohne näher an angezielte Koordinaten

fliegt.

/K030/ Es soll eine Inertialnavigation (Kurzzeit-Navigation) durch die Drohne stattfinden. Dadurch soll es der Drohne möglich sein, sich durch die Auswertung der durch die verschiedenen Sensoren ermittelten Daten selbst zu lokalisieren.

/K040/ Die Drohne ist über eine WLAN-Verbindung im Flug ansprechbar.

/K041/ Bricht der Kontakt der WLAN-Verbindung ab, soll die Drohne selbstständig eine Notlandung durchführen.

/K050/ Es soll möglich sein, geschlossene Räume, die den Betriebsbedingungen entsprechen, durch die Drohne erkunden zu lassen. Dabei wird zuvor der Grundriss in die Software geladen und der Ursprungspunkt innerhalb diesem festgelegt. Die Drohne soll die Räume vollständig zu erkunden, sodass diese als 3D-Modell rekonstruiert werden kann.

/K060/ Über die gesamte Flugdauer sollen Steuerkorrekturen durch die Inertialnavigation ermittelt werden, die eine verlässliche Navigation gewährleisten.

/K070/ In einer zweidimensionalen Kartendarstellung in der PC-Anwendung sollen erfasste Räume zur Flugzeit grob repräsentiert und den benutzenden Personen zugänglich gemacht werden.

/K071/ In dieser Karte soll eine textuelle Ausgabe der Position, der Ausrichtung und der Höhe der Drohne in Form von Koordinaten, Dreh- und Neigungswinkeln bzw. Zentimetern zur Flugzeit stattfinden.

/K080/ Zusätzlich zur Karte wird den nutzenden Personen der Videostream der Frontkamera der Drohne angezeigt.

/K090/ Auf Basis der Daten der Tiefenkamera (Structure-Sensor) soll die aufgenommene Umgebung als dreidimensionales Modell rekonstruiert werden.

/K100/ Eine Darstellung der rekonstruierten dreidimensionalen Szene soll auf einem Desktop-PC bereitgestellt werden.

/K110/ Die von der Drohne erfassten Sensordaten und Kamerabilder werden für die benutzenden Personen bereitgestellt. Alle Daten und Bilder werden mit einem Zeitstempel sowie der Position, Ausrichtung und Geschwindigkeit der Drohne annotiert.

/K111/ Die Bilder der Frontkamera sollen in Form eines Videostreams gemäß dem

MPEG-4-Standard (ISO/IEC-14496-2)[1] verfügbar gemacht werden.

/K112/ Die Daten des Structure-Sensors werden den entwickelnden Personen in einer Form bereitgestellt, die eine Weiterverarbeitung unter Verwendung der OpenNI2-Schnittstelle[2] ermöglicht.

/K113/ Die Daten der Inertialsensoren, des Höhensensors und des Magnetometers werden zur Entwicklung in numerischer Form verfügbar gemacht.

/K120/ Die durch die verwendete Tiefenkamera ermittelten Daten sollen wahlweise über die WLAN-Verbindung zur Laufzeit abrufbar sein oder auf dem internen Speicher abgelegt und zu einem späteren Zeitpunkt abrufbar sein.

1.2 Wunschkriterien

/K130/ Eine auf der Drohne abgelegte, über das WLAN-Netzwerk erreichbare Webseite ermöglicht den benutzenden Personen die Steuerung der Drohne. Zusätzlich zur Steuerung wird eine Übersicht über eingehende Sensordaten der Drohne bereitgestellt.

/K131/ Die Drohne soll über diese Webseite mithilfe der Tastatur, einen Controller oder in der Anzeige des Videostreams eingeblendete Eingabeelemente gesteuert werden können.

/K140/ Durch die Auswertung der Bodenkamera zur Flugzeit soll die Stabilität der Drohne im Flug erhöht werden.

/K150/ Mithilfe der Tiefenkamera soll eine primitive Kollisionserkennung auf der Drohne stattfinden.

/K160/ Neben der Vermeidung von Kollisionen mit statischen Objekten soll versucht werden auch Kollisionen mit beweglichen Objekten zu vermeiden, wie zum Beispiel Menschen, die sich in dem Raum bewegen.

/K170/ Die Drohne soll ohne vorherige Kenntnis der Umgebung diese erkunden können. Es muss kein Grundriss zur Verfügung gestellt werden, da die Drohne diesen Raum von alleine erkundet und dabei den Grundriss selbst erzeugt.

/K180/ Bei der Erkundung werden bis zu fünf zusammenhängende, erreichbare Räume

[1] ISO/IEC 14496-2:1999 Information technology – Coding of audio-visual objects – Part 2: Visual
[2] `http://com.occipital.openni.s3.amazonaws.com/OpenNI_Programmers_Guide.pdf` (2015/07/22)

erfasst.

/K190/ Die nach der Erkundung rekonstruierte Szene wird mithilfe der Erfassung und Darstellung (engl. *matching*) um die aus den gelieferten Videodaten der RGB-Kamera gewonnenen Farbinformationen erweitert.

/K200/ In der Darstellung des rekonstruierten 3D-Modells sollen über ein Kontextmenü Informationen und gegebenenfalls Einstellungsmöglichkeiten abrufbar sein.

/K 210 / Die Frontkamera liefert einen „Full-HD"-Videostream mit 30 Bildern pro Sekunde bei 1920×1080 Bildpunkten.

/K220/ Alle Sensordaten sind in Echtzeit über eine WLAN-Verbindung abrufbar.

/K230/ Der Videostream der Frontkamera ist gemäß Standard H.264/MPEG-4 AVC komprimiert.

/K240/ Der Zugriff auf die Tiefenkamera erfolgt durch eine Nachbildung der OpenNI2-API, sodass ein für diese API geschriebenes Programm die Tiefenkamera während des Flugs nutzen kann.

/K 250 / Die zur Laufzeit dargestellte Karte mit Statusinformationen wird auf eine dreidimensionale Repräsentation erweitert.

/K260/ Bewegliche Objekte, wie z.B. Fahrzeuge in einer Lagerhalle, werden erkannt und als solche in der Visualisierung markiert.

1.3 Abgrenzungskriterien

/K270/ Die Funktionen der Drohne sollen nicht im Freien nutzbar sein.

/K280/ Ohne WLAN-Verbindung ist die Nutzung der Software und Drohne mit allen Funktionen nicht möglich.

/K290/ Es wird kein stabiler Flug nach einer Beeinflussung der Drohne durch Hindernisse gewährleistet.

/K300/ Die verlässliche Navigation ist zwingend auf Rückmeldungen durch die sofortige Auswertung eines Teils der Sensordaten angewiesen.

/K310/ Es findet keine Erkundung der Umgebung über mehrere Gebäudeetagen hinweg statt.

/K320/ Treppenhäuser, Fahrstühle, etc. werden während der Erkundung nicht genutzt.

/K330/ Die Übermittlung von Sensordaten ist bei unzureichend guter WLAN-Verbindung nicht zwingend möglich (weniger als 30% der maximalen Signalstärke).

2 Produkteinsatz

Die komplette Hardware in Verbindung mit der entwickelten Software wird im Folgenden Produkt genannt.

2.1 Anwendungsbereiche

Das Produkt soll ein dreidimensionales Abbild der räumlichen Umgebung erzeugen, indem eine Drohne diesen selbstständig abfliegt und erkundet. Ein oder mehrere zusammenhängende Räume eines Gebäudes bilden dabei die Umgebung, wobei zu beachten ist, dass diese Räume alle in einer Etage befindlich sind und ausreichend große Verbindungen, z. B. offenstehende Türen, untereinander besitzen müssen.

2.2 Zielgruppen

Personengruppen, die Innenräume eines Gebäudes als dreidimensionales Modell abbilden wollen. Denkbare Anwender und Anwendungsbereiche sind u.a.

- Architekturbüros bei der Abbildung neu errichteter Gebäude
- Innenausstattex, die einzurichtende Räume als dreidimensionales Modell vorliegen haben möchten
- Vermietex, die in potenziellen Mieter einen Einblick in die Immobilie geben möchten
- das technische Hilfswerk im Einsatz in Katastrophengebieten
- Sprengmeistex, die zu sprengende Gebäude als dreidimensionales Modell nutzbar haben möchten
- Vermessungstechnikex von Bunkeranlagen.

Die Produktandwendex müssen im Stande sein, handelsübliche Computersoftware zu bedienen und benötigen keine zusätzliche fachliche Vorkenntnis.

2.3 Betriebsbedingungen

Grundlage zum Betrieb des Produkts sind Räumlichkeiten, die ein Mindestvolumen von $1m^3$ und ein Maximalvolumen von $4000m^3$ besitzen. Dabei muss es der Drohne möglich

sein, sich in dieser Umgebung fortzubewegen. Dafür müssen Wege existieren, die eine Breite und Höhe mindestens derer der Drohne besitzen. Existieren mehrere Räume, die untersucht werden sollen, so müssen diese für die Drohne erreichbar sein. Außerdem sollten es nicht mehr als fünf zusammenhängende Räume sein.

Darüber hinaus müssen die Räumlichkeiten mit einer Intensität zwischen 150 und 1000 Lux beleuchtet werden. Dabei dürfen die Kamerasysteme nicht durch die Art der Beleuchtung gestört werden, beispielsweise durch Interferenzen mit Neonröhren.

Außerdem muss eine WLAN-Verbindung (gemäß IEEE 802.11n[1]) zwischen der auf einem Desktop-PC ausgeführten Auswertungssoftware und der Drohne hergestellt werden. Die technischen Anforderungen sind in Abschnitt 3.2 näher beschrieben.

[1] IEEE Standard for Information technology–Telecommunications and information exchange between systems Local and metropolitan area networks–Specific requirements Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications

3 Produktumgebung

3.1 Software

- Windows 7 oder höher
- Mozilla Firefox v.„33“ oder höher

3.2 Hardware

- Parrot Bepop Drone mit angepasster Firmware und folgender verbauter Sensorik:
  1) 3 Achsen - Magnetometer
  2) 3 Achsen - Gyroskop
  3) 3 Achsen - Beschleunigungsmesser
  4) Kamera zur vertikalen Stabilisierung
  5) Frontkamera
  6) Ultraschall-Sensor
  7) Barometer
- Structure Sensor (Anbindung über Drohne)
- PC oder Notebook mit
  - Dual-Core Prozessor mit min. 2,4 GHz (z.B. Intel i5 4440)
  - min. 8GB RAM Arbeitsspeicher
  - NVidia-Grafikkarte mit Fermi oder Kepler Architektur (z.B. Nvidia GTX 760)
  - WLAN-Sender/Empfänger (gemäß IEEE 802.11n)

4 Produktfunktionen

Diese Kapitel beschreibt die Funktionen des Produkts, die die Nutzenden während des Betriebs aufrufen kann. Die Funktionen wurden in die Gruppen Drohnenbewegung und Serveranwendung eingeteilt.

4.1 Drohnenbewegung

In diese Funktionsgruppe sind Funktionen untergeordnet, mit denen Einfluss auf die Drohne und deren Flug genommen werden kann.

4.1.1 Steuerung

/F1010/ *Verbindung aufbauen*: Eine Verbindung zwischen der Anwendung auf dem Server und der Drohne per WLAN aufbauen

/F1020/ *direkte Steuerung*: Die Steuerung der Bewegung der Drohne.

/F1021/ *grundlegende Bewegung 1*: Vorwärts-, Rückwärtsflug

/F1022/ *grundlegende Bewegung 2*: Seitwärtsflug nach links und rechts

/F1023/ *grundlegende Bewegung 3*: Drehung per Winkelangabe

/F1024/ *grundlegende Bewegung 4*: Bewegung zu gegebener Koordinate

4.1.2 Flug

/F1030/ *Notabschaltung*: Unmittelbares Abschalten der Drohne, auch im Flug.

/F1040/ *sicheres Landen*: Die Drohne selbstständig landen und Motoren ausstellen lassen.

/F1050W/ *zurück zum Start*: Die Drohne selbstständig zum Startpunkt zurückfliegen lassen, wo sie dann landet und sich abstellt.

4.1.3 Erkundung

/F1060/ *autonome Erkundung starten*: Die Erkundung der Umgebung durch die Drohne auf Basis vorhandener Daten starten.

/F1070W/ *autonome Erkundung beenden*: Die Erkundung der Umgebung durch die Drohne beenden. Dabei landet die Drohne und alle bisher berechneten Daten werden für eine künftige Weiterbearbeitung gespeichert.

/F1080/ *Grundrisseingabe*: Den Grundriss in der Software des Servers laden.

4.2 Serveranwendung

In diese Funktionsgruppe sind Funktionen untergeordnet, mit denen kein Einfluss auf die Drohne und deren Flug genommen werden kann.

4.2.1 Visualisierung

/F2010/ *Anzeige*: Mit dieser Funktion kann sich der Nutzer die erkannte Umgebung anzeigen lassen.

/F2011/ *3D-Modell-Darstellung*: Die erkannte Umgebung kann als 3D-Modell angezeigt werden.

/F2012W/ *Grundriss-Darstellung*: Die erkannte Umgebung kann als Grundriss angezeigt werden.

4.2.2 IO

/F2020/ *Umgebungseingabe*: Eine vorher ermittelte Umgebung in die Software für die Visualisierung laden.

/F2030/ *Umgebungssicherung*: Die berechnete Umgebung in eine Datei speichern.

5 Produktdaten

/D010/ *Grundriss*: Der Grundriss ist eine Darstellung der zu untersuchenden Räumlichkeiten und wird durch einen oder mehrere Polygonzüge repräsentiert. Diese Polygonzüge werden in einer CSV-Datei mit fünf Spalten angegeben. Die Spalten *wall*, *vertex*, x, y, z geben einen fortlaufenden Index für die Polygonzüge, einen fortlaufenden Index für die Vertices in den Polygonzügen, die x-Koordinate des Vertex, die Deckenhöhe des entsprechenden Raumes am Punkt des Vertex, die z-Koordinate des Vertex an.

/D020/ *Rekonstruktion*: Die Aufbereitung der gesammelten Daten wird als Punktwolke gespeichert.

/D030/ *Video*: Die Videoaufnahmen des Flugs werden im MPEG-4-Format gespeichert.

/D040/ *3D-Modell*: Die gesammelten Daten werden als dreidimensionales Modell dargestellt.

/D050/ *Konfiguration für den Betrieb*: Alle notwendigen Parameter für den Betrieb sind in einer Konfigurationsdatei definiert.

/D051/ *Kommunikationsparameter*: In der Konfigurationsdatei sind die Netzwerkadressen der kommunizierenden Netzwerkteilnehmer eingetragen.

6 Produktleistungen

/L010/ *Erkundung*: Abfliegen der in Abschnitt 2.3 beschriebenen Räumlichkeiten mit maximal einer kompletten Akkuladung

/L020/ *Videoaufnahme-Bildrate*: Die Kamera nimmt 30 Bilder pro Sekunde auf.

/L030/ *Videoaufnahme-Auflösung*: Die Kamera nimmt Bilder mit einer Auflösung von 640×480 Pixeln auf.

7 Benutzeroberfläche

Die Benutzeroberfläche ermöglicht die schnelle und benutzerfreundliche Bedienung, um eine 3D-Karte aus einem Videostream der Drohne oder aus einer Menge von PCD-Dateien zu generieren. Es können verschiedene Optionen ein- bzw.ausgeschaltet werden, wie beispielsweise ein Voxel-Grid-Filter oder Passthrough-Filter. Über die Oberfläche können auch konstruierte 3D-Karten gespeichert und bereits vorhandene Karten geladen werden. Wenn eine 3D-Karte angezeigt wird, so öffnet sich immer ein neues Fenster, in welchem der Benutzer durch die Karte navigieren kann. Dazu werden die Pfeiltasten sowie die Maus verwendet. Zusätzlich hat die Benutzeroberfläche ein mehrzeiliges Textfeld für Logausgaben und einen Fortschrittsbalken, welcher den aktuellen Fortschritt beim Berechnen der 3D-Karte anzeigt.

7.1 Dialogstruktur

1. File
   a) Open File - öffnet eine 3D-Karte oder einen Ordner mit PCD-Dateien
   b) Open Stream - lädt einen Stream
   c) Close Stream - schließt den aktuellen Stream und startet die Berechnung der 3D-Karte
   d) Save - Speichert die berechnete 3D-Karte
   e) Exit - Beendet das Programm
2. Options - zur Wahl verschiedener Filter und Registrierungsverfahren
3. Features
   a) Triangulization - die berechnete 3D-Karte wird trianguliert
   b) Coloring - sofern das Wunschkriterium /K190/ erfüllt wurde, wird die triangulierte Fläche entsprechend dem Originalbild eingefärbt

7.2 Layout

In der Menüleiste können die in Abschnitt 7.1 genannten Optionen eingestellt werden. Zudem können hier 3D-Karten gespeichert und geladen werden und das Programm kann beendet werden.

Dort befinden sich auch drei Einträge um einen Stream zu öffnen bzw.zu schließen und um eine Datei zu laden, welche einen Kamerastream der Drohne öffnen oder einen

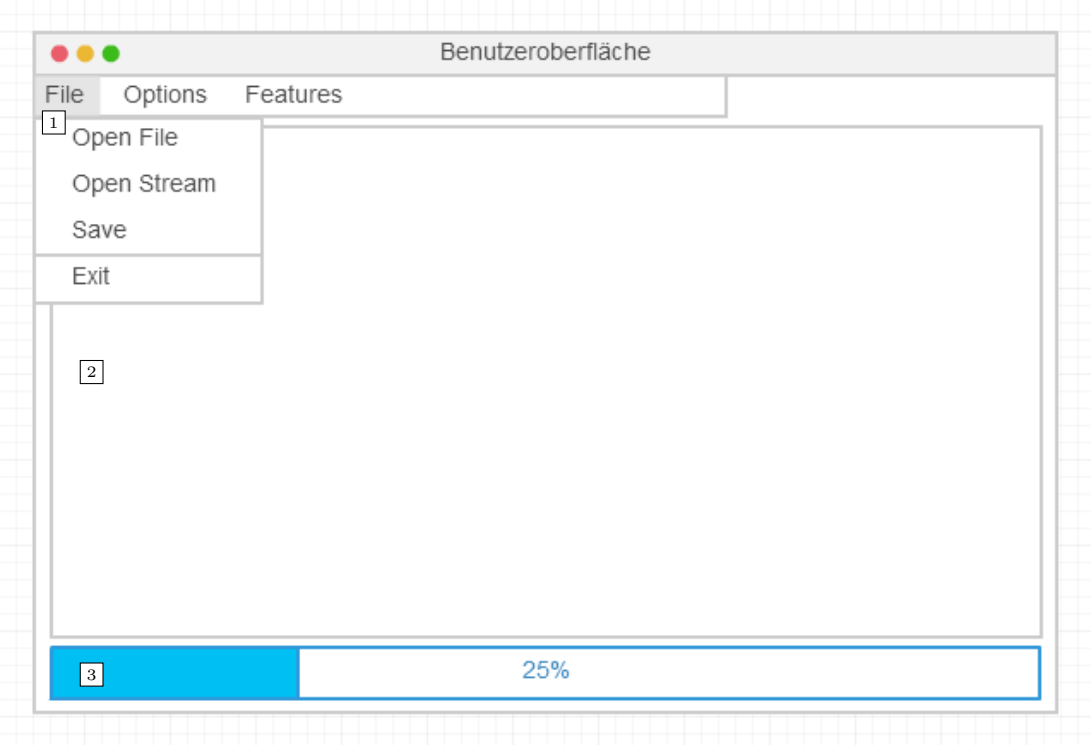


Abbildung 7.1 – Entwurf der Benutzeroberfläche

1. Menüleiste mit verschiedenen Funktionen und Einstellungsmöglichkeiten
2. Textfeld für Log- und Fehlerausgaben
3. Fortschrittsleiste bei Berechnungen

Ordner mit PCD-Dateien lesen kann. Nach dem Öffnen des Stream bzw.der Dateien wird eine 3D-Karte erstellt, welche im Anschluss in einem neuen Fenster geöffnet wird.

Im Textfeld werden alle Logausgaben des Programms dargestellt. Der Fortschrittsbalken befindet sich unter dem Textfeld und gibt den Fortschritt der Kartenberechnung in Prozent an.

8 Qualitätsbestimmungen

Folgende Tabelle stellt dar, auf welche Qualitätsaspekte während der Entwicklung Wert gelegt wurde.

	sehr wichtig	wichtig	eher unwichtig	unwichtig
Robustheit	X			
Zuverlässigkeit		X		
Korrektheit		X		
Benutzungsfreundlichkeit			X	
Effizienz		X		
Portierbarkeit				X

9 Globale Testszenarien und Testfälle

/T010/ *Netzwerkanbindung*: Ein Rechner mit WLAN-Modul verbindet sich mit dem Netzwerk der eingeschalteten Drohne (/F1010/). : *Der Test ist erfolgreich, wenn* eine WLAN-Verbindung zur Drohne aufgebaut werden kann.

/T020/ *Abruf von Sensordaten der Drohne*: Über die bereitgestellte API können Sensordaten direkt von der Drohne ausgelesen werden. : *Der Test ist erfolgreich, wenn* Sensordaten in der Anwendung verwendet oder ausgegeben werden können.

/T030/ *Direkte Steuerung*: Der Benutzer kann Steuerungsbefehle an die Drohne senden, die daraufhin ausgeführt werden (/F1020/-/F1023/). : *Der Test ist erfolgreich, wenn* die Drohne sich entsprechend der gesendeten Befehle bewegt.

/T031/ *Navigatorische Steuerung*: Nach Übergabe eines Vektors bewegt sich die Drohne selbständig zu der relativen Position (/F1024/). : *Der Test ist erfolgreich, wenn* die Drohne bis auf einen halben Meter genau an dem entsprechendem Punkt ankommt und dort verweilt.

/T040/ *Videostream*: Das Bild der Frontkamera soll mit einem verbundenen Computer empfangen werden. : *Der Test ist erfolgreich, wenn* auf dem Computer das Bild angezeigt werden kann.

/T050/ *Test von Beleuchtungsbedingungen*: Die Erkundung wird bei verschiedenen Beleuchtungen getestet (/F1060/). : *Der Test ist erfolgreich, wenn* die Drohne eine Karte, bzw. ein Modell erstellt, das den untersuchten Raum zu mindestens 70% korrekt abbildet.

/T051/ *Unterbelichtung*: Die Erkundung wird bei einer Beleuchtungsintensität von unter 150 Lux getestet (/F1060/). : *Der Test ist erfolgreich, wenn* die Drohne eine Karte, bzw. ein Modell erstellt, das den untersuchten Raum zu mindestens 70% korrekt abbildet.

/T052/ *Überbelichtung*: Die Erkundung wird bei einer Beleuchtungsintensität von über 1000 Lux getestet (/F1060/). : *Der Test ist erfolgreich, wenn* die Drohne eine Karte, bzw. ein Modell erstellt, das den untersuchten Raum zu mindestens 70% korrekt abbildet.

/T053/ *Tageslicht*: Die Erkundung wird in einem Raum ohne aktive künstliche Beleuchtung - ausschließlich Tageslicht, das durch Fenster und Türen eindringt, wobei die

Intensität zwischen 150 und 1000 Lux liegt - getestet (/F1060/). : *Der Test ist erfolgreich, wenn* die Drohne eine Karte, bzw. ein Modell erstellt, das den untersuchten Raum zu mindestens 70% korrekt abbildet.

/T054/ *Neonröhren*: Die Erkundung wird in einem Raum mit einer Beleuchtung ausschließlich durch Neonröhren, wobei die Intensität zwischen 150 und 1000 Lux liegt, getestet (/F1060/). : *Der Test ist erfolgreich, wenn* die Drohne eine Karte, bzw. ein Modell erstellt, das den untersuchten Raum zu mindestens 70% korrekt abbildet.

/T055/ *Neonröhren und Tageslicht*: Die Erkundung wird in einem Raum mit einer Beleuchtung durch Neonröhren sowie Tageslicht, das durch Fenster und Türen eindringt, wobei die Intensität zwischen 150 und 1000 Lux liegt, getestet (/F1060/). : *Der Test ist erfolgreich, wenn* die Drohne eine Karte, bzw. ein Modell erstellt, das den untersuchten Raum zu mindestens 70% korrekt abbildet.

/T 060 / *Test von Formbedingungen*: Test der Erkundung in Räumen verschiedener Formen. : *Der Test ist erfolgreich, wenn* die Drohne eine Karte, bzw. ein Modell erstellt, das den untersuchten Raum zu mindestens 70% korrekt abbildet.

/T061/ *Grundriss laden*: In der Software werden die Grundrisse verschiedener Räume geladen und abgeflogen (/F1080/). : *Der Test ist erfolgreich, wenn* die Drohne eine Karte, bzw. ein Modell erstellt, das den untersuchten Raum zu mindestens 70% korrekt abbildet.

/T 062 / *große Räume*: Ein großer Raum (über $100m^2$, z. B. eine Lagerhalle) wird abgeflogen. : *Der Test ist erfolgreich, wenn* die Drohne eine Karte, bzw. ein Modell erstellt, das den untersuchten Raum zu mindestens 70% korrekt abbildet.

/T063/ *wenig Struktur*: Ein Raum, welcher wenig Struktur aufweist (z. B. ein unmöblierter Raum) wird erkundet. : *Der Test ist erfolgreich, wenn* die Drohne eine Karte, bzw. ein Modell erstellt, das den untersuchten Raum zu mindestens 70% korrekt abbildet.

/T 064 / *viel Struktur*: Ein Raum, welcher viel Struktur aufweist (z. B. möblierter Büroraum mit vielen Bildern an den Wänden) wird erkundet. : *Der Test ist erfolgreich, wenn* die Drohne eine Karte, bzw. ein Modell erstellt, das den untersuchten Raum zu mindestens 70% korrekt abbildet.

/T 070 / *Test der autonomen Steuerung*: Die autonome Steuerung muss die Drohne in verschiedenen Szenarien kollisionsfrei zu einer angegebenen Koordinate steuern (/F1024/). : *Der Test ist erfolgreich, wenn* die Drohne die angegebene Position bis auf einen halben Meter genau, ohne eine Kollision zu verursachen, erreicht.

/T071/ *enge Gassen*: Die autonome Steuerung fliegt die Drohne durch einen Raum, der

beispielsweise mit Stellwänden zugestellt ist, sodass kaum freie große Flächen vorhanden sind, zu einer angegebenen Koordinate (/F1024/). : *Der Test ist erfolgreich, wenn* die Drohne die angegebene Position bis auf einen halben Meter genau, ohne eine Kollision zu verursachen, erreicht.

/T072/ *Störgrößen*: Luftbewegungen wirken (z. B. durch einen Ventilator) während des Flug auf die Drohne ein, und es wird geprüft, ob die Drohne die Position halten kann bzw. zu ihr zurückkehrt. : *Der Test ist erfolgreich, wenn* die Drohne nach fünf Minuten Beeinflussung den Standpunkt bis auf einen halben Meter genau gehalten hat.

/T073/ *WLAN-Störung*: Die WLAN-Verbindung wird abrupt unterbrochen, z.B. über einen Schalter am Notebook. : *Der Test ist erfolgreich, wenn* die Drohne landet ohne Schaden zu nehmen oder zu verursachen.

/T074/ *Notlandung*: Die Drohne wird durch den Benutzer notgelandet. Die Drohne sinkt bei Befehl zu landen auf eine minimale Flughöhe und schaltet daraufhin alle Motoren ab (/F1040/). : *Der Test ist erfolgreich, wenn* die Drohne landet ohne Schaden zu nehmen oder zu verursachen.

/T075/ *Notabschaltung*: Der Benutzer schaltet die Drohne über die Funktion der Notabschaltung ab (/F1030/). : *Der Test ist erfolgreich, wenn* die Drohne alle Motoren unmittelbar abschaltet.

/T080/ *Test der Navigationsgenauigkeit*: Die Drohne soll nach bestimmter, einstellbarer Flugdauer wieder genau dort landen, wo sie gestartet ist : *Der Test ist erfolgreich, wenn* die Drohne nach der genannten Dauer bis auf einen halben Meter genau wieder am Ursprung landet.

/T081/ *nur Inertialnavigation*: Es wird der Testfall /T080/ durchgeführt, wobei nur die Inertialnavigation genutzt wird. : *Der Test ist erfolgreich, wenn* die Drohne nach der genannten Dauer bis auf einen halben Meter genau wieder am Ursprung landet.

/T082/ *mit Rückmeldung durch Bildverarbeitung*: Es wird der Testfall /T080/ durchgeführt, wobei die Inertialnavigation sowie die Lokalisierung durch die Bildverarbeitung genutzt wird. : *Der Test ist erfolgreich, wenn* die Drohne nach der genannten Dauer bis auf einen halben Meter genau wieder am Ursprung landet.

/T090/ *Test der Erkundung*: Prüfung ob die Drohne den Raum komplett abfliegt, sodass die Daten für eine Rekonstruktion verwertet werden können (/F1060/). : *Der Test ist erfolgreich, wenn* die Drohne eine Karte bzw. ein Modell erstellt, das den untersuchten Raum zu mindestens 70% korrekt abbildet.

/T091/ *viele Sichtschutze*: Der Raum ist mit vielen Hindernissen ausgestattet, sodass viele Sichtblenden entstehen (/F1060/). : *Der Test ist erfolgreich, wenn* die Drohne eine

Karte, bzw. ein Modell erstellt, das den untersuchten Raum zu mindestens 70% korrekt abbildet.

/T100/ *Test des Speicherns von Rekonstruktionen*: In der Software werden erzeugte Rekonstruktionen abgespeichert (/F2030/). : *Der Test ist erfolgreich, wenn* Daten erzeugt werden, die den Spezifikationen entsprechen.

/T110/ *Test des Ladens von Rekonstruktionen*: In der Software werden zuvor gespeicherte Rekonstruktionen wieder eingelesen (/F2020/). : *Der Test ist erfolgreich, wenn* das zuvor gespeicherte Rekonstruktion wieder korrekt geladen wird.

10 Entwicklungsumgebung

10.1 Software

- Plattformen
  - Qt 5.4.2
  - PCL 1.7.2 mit KinFu und OpenNI 2
  - CUDA 7.0
  - Asio 1.10.6
  - Eigen 3.2.1
  - OpenMP
  - EasyLogging++ 9.8.0
- Programme
  - Microsoft Visual Studio 2013 oder höher
  - Unity 5.1 oder höher
  - QtCreator 3.3.0
  - CMake 3.2.2
  - Ubuntu 14.04 LTS für Firmware-Entwicklung
    - arm-linux-gnueabi-g++ 4.7.3
    - GNU Binutils (arm-linux-gnueabi) 2.24
  - gängiger Browser

10.2 Hardware

- Parrot Bepop Drone mit angepasster Firmware
- Parrot AR Drone 2.0
- PC oder Notebook mit
  - Dual-Core Prozessor mit min. 2,4 GHz
  - min. 8GB RAM Arbeitsspeicher
  - NVidia-Grafikkarte mit Fermi oder Kepler Architektur
  - WLAN-Sender/Empfänger (gemäß IEEE 802.11n)

10.3 Orgware

- Microsoft Project 2013
- Microsoft Visio 2013
- Trac 1.0.2
- SVN

Abbildungsverzeichnis

2.1. Bilder drei beispielhafter Drohnen. 11
2.2. Übersicht über die vier Aufgabenfelder des Projekts 13

3.1. Parrot Bebop Drone 17
3.2. Parrot AR.Drone2 17
3.3. Tiefensensor „Structure Sensor“ auf der Drohne montiert 20
3.4. Schematische Draufsicht eines Quadrokopters 21
3.5. Grobe Unterteilung der Komponenten einer Drohnensteuerung 22
3.6. Rotordrehzahlregelung 23
3.7. Blockschaltbild eines Regelkreises 23
3.8. Beispiel für eine Drehzahlregelung durch ein eingebettetes System . . 25
3.9. Blockschaltbild eines Regelkreises für die Drohnenlage 26
3.10. Blockschaltbild Regelkreises für Lage und Rotordrehzahlen 27
3.11. Hypothetisches Beispiel für Navigation mit einem P-Regler 28
3.12. Blockschaltbilder des PID-Reglers in zwei verschiedenen Darstellungen 29

4.1. Übersicht über die Funktionsweise des Kalman-Filters 35
4.2. Übersicht über die Funktionsweise des erweiterten Kalman-Filters . . 36
4.3. Bewegung und Beobachtung eines Vehikels 46
4.4. Bausteine des LSD-SLAM-Verfahrens 49
4.5. Verschiebung der Position eines Pixel innerhalb des aufgenommenen Bildes 54
4.6. Globale Karte vor und nach der Schleifenerkennung 56
4.7. Ergebnisse einer ungenügenden Initialisierung des ersten Keyframes in LSD-SLAM 58
4.8. Visualisierung eines Ergebnisses zur heuristischen Parametersuche . . 63
4.9. Farbliche Darstellung von ungünstigen Tiefenwert-Schätzungen durch LSD-SLAM 64
4.10. Skizze zur Überführung von Pixeln mit bekannten Tiefen auf einen Keyframe 65

4.11. Auswirkungen der Erhöhung des Varianz-Schwellwerts zur Ausreißer-Entfernung . . . 67
4.12. Visualisierung des verwendeten Octrees . . . 69
4.13. Beispielhafte Ausgabe der Octree-Visualisierung . . . 75

5.1. Beispiel einer Pfadplanung für eine Bronchoskopie . . . 79
5.2. Riskmap eines vorausfahrenden Autos auf einer Straße mit Leitplanken. 80
5.3. Erstellen eines Ablaufplans mithilfe von Floodfill nach dem Prinzip der Breitensuche. . . . 82
5.4. Erstellen eines Ablaufplans mithilfe von Floodfill nach dem Prinzip der Tiefensuche. . . . 82
5.5. Berechneter Pfad nach Reduktion der Anzahl der Wegpunkte. . . . 83
5.6. Erkundung mithilfe von Markierungen bereits gesehener Punkte. . . . 84
5.7. Rasterung der Linie von Punkt zu Punkt . . . 85
5.8. Grundlegender Aufbau eines Grundrisses . . . 88
5.9. Gegenüberstellung möglicher Hindernis-Szenarien. . . . 91
5.10. Sprung in Richtung des Zieles . . . 95
5.11. Eliminierung von Nachbarknoten bei Jump-Point-Search . . . 95
5.12. Vergleich zwischen einem Pfad und seiner vereinfachten Variante . . 96
5.13. Auf Schnitte mit Hindernissen getestete Strecken . . . 97
5.14. Schritte eines Schnitttests zwischen Strecke und Quadtree . . . 98
5.15. Simulatioransicht einer Drohne in einem Zimmer . . . 102
5.16. Darstellung einer simulierten Kollision zwischen Drohne und Wand . 102
5.17. Darstellung von Hilfsobjekten im Simulator . . . 103

6.1. KinectFusion: Komponenten und Pipeline . . . 107
6.2. Bilateraler Filter . . . 109
6.3. Bildpyramide . . . 111
6.4. Visualisierung TSDF . . . 111
6.5. Moving Volume . . . 118
6.6. Schematische Punktmengenaufteilung eines zwei-dimensionalen k-d-Baumes . . . 122
6.7. Inhalt eines Voxels . . . 123
6.8. Registrierung von Punktwolken . . . 125
6.9. Umgebung eines Punktes . . . 126
6.10. Punktkorrespondenzen zwischen zwei Punktwolken . . . 127
6.11. ICP Verarbeitungsschritte . . . 129

6.12. Konvergenz von ICP gegen lokale Minima 130
6.13. Poisson Ablauf 135
6.14. Ergebnis einer Poisson Rekonstruktion 137
6.15. KinectFusionLS: Aktivitätsdiagramm 139

7.1. Übersicht aller Komponenten bei Steuerung mit der Schnittstelle von Parrot. 143
7.2. Protokoll zur Abgrenzung einzelner Tiefenbilder in einem TCP-Socket 154

8.1. Gesamtbild der Benutzerschnittstelle zu ARGOS-Control 162
8.2. Manuelle Steuerung und Ansicht der Drohnendaten 163
8.3. Hauptfenster der Benutzerschnitstelle der Rekonstruktion 165
8.4. Fenster der DepthImageSequence mit angezeigten Tiefendaten 165
8.5. KinFu Hauptfenster mit dem Tiefenbildern 166
8.6. Rekonstruierte 3D-Szene 167

9.1. Versuchsaufbau des Evaluierungsszenarios 4 der Pfadplanung 174
9.2. Versuchsaufbau des Evaluierungsszenarios 5 der Pfadplanung 175
9.3. Visualisierung eines quadratischen Raumes im Pfadplanungs-Explorer ohne Hindernisse. 175
9.4. Visualisierung eines langen Raumes im Pfadplanungs-Explorer ohne Hindernisse. 176
9.5. Visualisierung eines langen Raumes im Pfadplanungs-Explorer ohne Hindernisse. 176
9.6. Visualisierung eines Raumes im Pfadplanungs-Explorer mit vier Hindernissen, welche Säulen darstellen. 177
9.7. Visualisierung eines Raumes im Pfadplanungs-Explorer mit drei Hindernissen, welche wie ein U angeordnet sind. 177
9.8. Visualisierung eines Raumes im Pfadplanungs-Explorer mit zwei Hindernissen, welche wie ein L angeordnet sind. 177
9.9. Visualisierung eines Raumes im Pfadplanungs-Explorer mit zwei Hindernissen, welche parallel angeordnet sind. 177
9.10. Beispiel-Szene 1 für die Trajektorien-Auswertung 178
9.11. Beispiel-Szene 2 für die Trajektorien-Auswertung 179
9.13. Mit dem Tiefensensor abgefilmte Szene 181
9.14. Zeitlicher Verlauf der Kompressionsraten verschiedener Algorithmen über das Testvideo 185
9.15. Abstand der Drohne zum Ziel und Flugdauer 187

9.16. Laufzeiten der Pfadfindung bei steigenden Raumgrößen 188
9.17. Abweichung der geschätzten Position zur realen Position 189
9.18. Verhältnis zwischen verfügbaren und benutzten Punkten in der aktuellen Szene 190
9.19. Kamerabilder und LSD-Slam-Visualisierung der Daten 190
9.20. Resultatierende Trajektorie für die Gehweg-Szene. 191
9.21. Vergleich zwischen registriertem Modell und Ground-Truth-Modell . . 192
9.22. Vergleich der berechneten Standardabweichung des KinFu-Verfahrens und den spezifizierten Daten des Tiefensensors. 193
9.23. Visualisierung der KinFu-Volumen des Raumes aus dem zweiten Szenario. 195
9.24. Mit ICP und GICP registrierte Volumen in der Gegenüberstellung . . 196

Algorithmenverzeichnis

4.1. Eindimensionaler Kalman-Filter 37

5.1. Einfacher Algorithmus „floodFill (Punkt **p**, Richtung **r**)“ 81
5.2. Einfacher Algorithmus zur Pfadsimplifizierung 96
5.3. Schnitttest zwischen einem achsenparallelen Quader und einer Strecke 99

7.1. Bewegung der Drohne um 2,5 m, bei einer Maximalgeschwindigkeit von 0,5 m/s. 143

Quellcodeverzeichnis

7.1. Beispiel für manuellen Steuerbefehl 154
7.2. Beispiel für Positionsübertragung 155
7.3. Handshake zum Upgrade auf das Websocket-Protokol 157
7.4. Handshakeantwort zum Upgrade auf das Websocket-Protokoll 157

Literaturverzeichnis

[1] : *DJI Inspire 1.* http://www.dji.com/de/inspire-1

[2] Akenine-Möller, T.; Haines, E.; Hoffman, N.: *Real-Time Rendering 3rd Edition.* A. K. Peters, Ltd., 2008. – 1045 S

[3] Barlow, R. J.: *Statistics: A Guide to the Use of Statistical Methods in the Physical Sciences.* 1989. – S. 60 S. – ISBN 978-0-471-92295-7

[4] Berc, L.; Fenner, W.; Frederick, R.; McCanne, S.; Stewart, P.: *RTP Payload Format for JPEG-compressed Video.* RFC 2435. 1998. – URL http://www.ietf.org/rfc/rfc2435.txt. – Zugriffsdatum: 21.02.2016

[5] Besl, P. J.; McKay, N. D.: Method for registration of 3-D shapes International Society for Optics and Photonics (Veranst.), 1992, S. 586–606

[6] Blais, G.; Levine, M. D.: Registering multiview range data to create 3D computer objects. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 17 (1995), Nr. 8, S. 820–824

[7] Blanco, J.-L.: A tutorial on SE(3) transformation parameterizations and on-manifold optimization / University of Almería. 2014. – Forschungsbericht

[8] Boissonnat, J.-D.; Geiger, B.: Three-dimensional reconstruction of complex shapes based on the Delaunay triangulation International Society for Optics and Photonics (Veranst.), 1993, S. 964–975

[9] Botea, A.; Müller, M.; Schaeffer, J.: Near Optimal Hierarchical Path-Finding. In: *Journal of Game Development* (2004)

[10] Bresenham, J. E.: Ambiguities in incremental line rastering. In: *Computer Graphics and Applications, IEEE* 7 (1987), Nr. 5, S. 31–43

[11] Bylow, E.; Sturm, J.; Kerl, C.; Kahl, F.; Cremers, D.: Real-time camera tracking and 3d reconstruction using signed distance functions Robotics: Science and Systems (Veranst.), 2013

[12] Chávez, O.; Enrique, J.; Sebastián, D. A. P.: *Diseño, construcción y control de un hexacóptero de monitoreo*, Quito, 2015., Dissertation, 2015

[13] Chen, Y.; Medioni, G.: Object modeling by registration of multiple range images, 1991, S. 2724–2729 vol.3

[14] Corporation, M.: *Kinect Fusion*. 2016. – URL `https://msdn.microsoft.com/en-us/library/dn188670.aspx`. – Zugriffsdatum: 21.02.2016

[15] Curless, B.; Levoy, M.: A Volumetric Method for Building Complex Models from Range Images, ACM, 1996, S. 303–312

[16] Developers, L.: *Linux Media Infrastructure API*. 2015. – URL `http://linuxtv.org/downloads/v4l-dvb-apis/`. – Zugriffsdatum: 2015-02-15

[17] Digor, E.; Birk, A.; Nüchter, A.: Exploration strategies for a robot with a continously rotating 3d scanner. In: *Simulation, Modeling, and Programming for Autonomous Robots*. Springer, 2010, S. 374–386

[18] Dissanayake, M.; Newman, P.; Clark, S.; Durrant-Whyte, H. F.; Csorba, M.: A solution to the simultaneous localization and map building (SLAM) problem. In: *Robotics and Automation, IEEE Transactions on* 17 (2001), Nr. 3, S. 229–241

[19] Durrant-Whyte, H.; Bailey, T.: Simultaneous localization and mapping: part I. In: *Robotics & Automation Magazine, IEEE* 13 (2006), Nr. 2, S. 99–110

[20] Elfving, T.: Block-iterative methods for consistent and inconsistent linear equations. In: *Numerische Mathematik* 35 (1980), Nr. 1, S. 1–12

[21] Engel, J.; Schöps, T.; Cremers, D.: LSD-SLAM: Large-Scale Direct Monocular SLAM, September 2014

[22] Engel, J.; Schöps, T.; Cremers, D.: *GitHub-Repository von LSD-SLAM*. 2016. – URL `https://github.com/tum-vision/lsd_slam`. – Zugriffsdatum: 21.02.2016

[23] Engel, J.; Sturm, J.; Cremers, D.: Camera-based navigation of a low-cost quadrocopter IEEE (Veranst.), 2012, S. 2815–2821

[24] Engel, J.; Sturm, J.; Cremers, D.: Semi-dense visual odometry for a monocular camera, 2013, S. 1449–1456

[25] Fette, I.: The WebSocket Protocol. In: *Internet Engineering Task Force* (2011)

[26] Fischler, M. A.; Bolles, R. C.: Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. In: *Communications of the ACM* 24 (1981), Nr. 6, S. 381–395

[27] Fossati, S.: Monocular autonomous exploration in unknown environment with low cost quadrotor. (2013)

[28] Francisco Heredia, R. F.: *Kinfu Large Scale*. Juni 2012. – URL `http://www.pointclouds.org/blog/srcs/`. – Zugriffsdatum: 21.02.2016

[29] Frankowski, G.; Chen, M.; Huth, T.: Real-time 3D shape measurement with digital stripe projection by Texas Instruments Micro Mirror Devices DMD International Society for Optics and Photonics (Veranst.), 2000, S. 90–105

[30] Fraundorfer, F.; Heng, L.; Honegger, D.; Lee, G. H.; Meier, L.; Tanskanen, P.; Pollefeys, M.: Vision-based autonomous mapping and exploration using a quadrotor MAV IEEE (Veranst.), 2012, S. 4557–4564

[31] Galanis, G.; Anadranistakis, M.: A one-dimensional Kalman filter for the correction of near surface temperature forecasts. In: *Meteorological Applications* 9 (2002), Nr. 04, S. 437–441

[32] Glover, A.; Maddern, W.; Warren, M.; Reid, S.; Milford, M.; Wyeth, G.: Openfabmap: An open source toolbox for appearance-based loop closure detection IEEE (Veranst.), 2012, S. 4730–4735

[33] Grisetti, G.; Kummerle, R.; Strasdat, H.; Konolige, K.: *g2o: A general Framework for (Hyper) Graph Optimization*, 2011

[34] Guitton, A.; Symes, W. W.: Robust inversion of seismic data using the Huber norm. In: *GEOPHYSICS* 68 (2003), Juli-August, Nr. 4, S. 1310–1319

[35] Harabor, D. D.; Grastien, A.: Online Graph Pruning for Pathfinding On Grid Maps., 2011

[36] Hart, P. E.; Nilsson, N. J.; Raphael, B.: A formal basis for the heuristic determination of minimum cost paths. In: *Systems Science and Cybernetics, IEEE Transactions on* 4 (1968), Nr. 2, S. 100–107

[37] Hartmann, S.: The world as a process. Springer, 1996, S. 77–100

[38] Horn, B. K.: Closed-form solution of absolute orientation using unit quaternions. In: *JOSA A* 4 (1987), Nr. 4, S. 629–642

[39] Hsieh, C.-T.: An efficient development of 3D surface registration by Point Cloud Library (PCL) IEEE (Veranst.), 2012, S. 729–734

[40] Huffman, D. A.: A Method for the Construction of Minimum-Redundancy Codes / Proceedings of the I.R.E. 1952. – Forschungsbericht

[41] Igelbrink, T.; Wiemann, T.; Hertzberg, J.: Generating topologically consistent triangle meshes from large scale Kinect Fusion. In: *Mobile Robots (ECMR), 2015 European Conference on* IEEE (Veranst.), 2015, S. 1–6

[42] ISO: Information technology – Coding of audio-visual objects – Part 2: Visual / International Organization for Standardization. oct 2005 (ISO/IEC 14496-2). – Forschungsbericht

[43] Izadi, S.; Kim, D.; Hilliges, O.; Molyneaux, D.; Newcombe, R.; Kohli, P.; Shotton, J.; Hodges, S.; Freeman, D.; Davison, A. u. a.: KinectFusion: real-time 3D reconstruction and interaction using a moving depth camera ACM (Veranst.), 2011, S. 559–568

[44] Jannach, D.: Webtechnologien 1. In: *Vorlesung TU Dortmund* (2013)

[45] Kaul, L.; Zlot, R.; Bosse, M.: Continuous-Time Three-Dimensional Mapping for Micro Aerial Vehicles with a Passively Actuated Rotating Laser Scanner. In: *Journal of Field Robotics* 33 (2016), Nr. 1, S. 103–132

[46] Kazhdan, M.; Bolitho, M.; Hoppe, H.: Poisson surface reconstruction, 2006

[47] Kiraly, A. P.; Helferty, J. P.; Hoffman, E. A.; McLennan, G.; Higgins, W. E.: Three-dimensional path planning for virtual bronchoscopy. In: *Medical Imaging, IEEE Transactions on* 23 (2004), Nr. 11, S. 1365–1379

[48] Krommweh, J.: Bilaterale Filter zur Entstörung von Bildern. (2007)

[49] Kümmerle, R.; Grisetti, G.; Strasdat, H.; Konolige, K.; Burgard, W.: g 2 o: A general framework for graph optimization IEEE (Veranst.), 2011, S. 3607–3613

[50] Lorensen, W. E.; Cline, H. E.: Marching cubes: A high resolution 3D surface construction algorithm ACM (Veranst.), 1987, S. 163–169

[51] Lorensen, W. E.; Cline, H. E.: Seminal Graphics. ACM, 1998, S. 347–353

[52] Marwedel, P.: *Eingebettete Systeme*. Springer, 2007

[53] Mitchell, J. L.; Pennebaker, W. B.: *JPEG still image data compression standard*. Van Nostrand Reinhold, 1993

[54] Montemerlo, M.; Thrun, S.; Koller, D.; Wegbreit, B. u. a.: FastSLAM: A factored solution to the simultaneous localization and mapping problem, 2002, S. 593–598

[55] Moore, B. C.: Principal component analysis in linear systems: Controllability, observability, and model reduction. In: *Automatic Control, IEEE Transactions on* 26 (1981), Nr. 1, S. 17–32

[56] Mur-Artal, R.; Montiel, J.; Tardos, J. D.: ORB-SLAM: a versatile and accurate monocular SLAM system. In: *Robotics, IEEE Transactions on* 31 (2015), Nr. 5, S. 1147–1163

[57] Naumann, F.: *Implementierung Tile Caching fähiger Map Clients*, Hochschule für Technik und Wirtschaft Dresden, Diplomarbeit, 2010

[58] Newcombe, R. A.; Izadi, S.; Hilliges, O.; Molyneaux, D.; Kim, D.; Davison, A. J.; Kohi, P.; Shotton, J.; Hodges, S.; Fitzgibbon, A.: KinectFusion: Real-time dense surface mapping and tracking IEEE (Veranst.), 2011, S. 127–136

[59] Occipital, I.: *Structure Sensor Depth Precision*. – URL `http://io.structure.assets.s3.amazonaws.com/structure_sensor_precision.pdf`. – Zugriffsdatum: 21.02.2016

[60] Occipital, I.: *Technical Specifications of the Structure Sensor*. 2015. – URL `http://structure.io/support/what-are-the-structure-sensors-technical-specifications`. – Zugriffsdatum: 21.02.2016

[61] Parrot: *ARDroneSDK3 API Reference*. 2016. – URL `http://developer.parrot.com/docs/bebop/#commands-and-events`. – Zugriffsdatum: 21.02.2016

[62] Parrot SA (Veranst.): *Parrot AR.Drone2.0 Bedienungsanleitung.* 2013

[63] Parrot SA (Veranst.): *Parrot Bebop Drone Bedienungsanleitung.* 2015

[64] Pfeifer, R.: *Effektive Messauswertung mit der Gauß'schen Fehlerquadratmethode: Einführung und Beispiele aus der Sportwissenschaft zur Lösung von komplexen Problemen mit Hilfe von Tabellenkalkulationen.* Sport und Buch Strauß, 2001

[65] Pomerleau, F.; Wild, M.: Recent Development of the Iterative Closest Point (ICP) Algorithm. (2010)

[66] Ramsden, E.: Hall-effect sensor: theory and aplications(2, illustrated ed.) Elsevier / ISBN 0-7506-7934-4. 2006. – Forschungsbericht

[67] Reichardt, D.; Shick, J.: Collision avoidance in dynamic environments applied to autonomous vehicle guidance on the motorway. In: *Intelligent Vehicles' 94 Symposium, Proceedings of the* IEEE (Veranst.), 1994, S. 74–78

[68] Rosten, E.; Drummond, T.: Machine learning for high-speed corner detection. Springer, 2006, S. 430–443

[69] Roth, H.; Vona, M.: Moving Volume KinectFusion., 2012, S. 1–11

[70] Rusinkiewicz, S.; Hall-Holt, O.; Levoy, M.: Real-time 3D Model Acquisition. In: *ACM Trans. Graph.* 21 (2002), Nr. 3, S. 438–446

[71] Rusu, R. B.; Blodow, N.; Beetz, M.: Fast point feature histograms (FPFH) for 3D registration IEEE (Veranst.), 2009, S. 3212–3217

[72] Segal, A.; Haehnel, D.; Thrun, S.: Generalized-ICP., 2009

[73] Stout, B.: Smart moves: Intelligent pathfinding. In: *Game developer magazine* 10 (1996), S. 28–35

[74] T. Bray, E.: The JavaScript Object Notation (JSON) Data Interchange Format. In: *Internet Engineering Task Force* (2014)

[75] Tomasi, C.; Manduchi, R.: Bilateral filtering for gray and color images, 1998, S. 839–846

[76] Unbehauen, H.: *Regelungstechnik I.* Vieweg + Teubner Verlag, 2008

[77] Welch, G.; Bishop, G.: *An Introduction to the Kalman Filter*. University of North Carolina at Chapel Hill. 1998

[78] Zhang, Z.: Iterative point matching for registration of free-form curves and surfaces. In: *International journal of computer vision* 13 (1994), Nr. 2, S. 119–152