Westfälische Hochschule
Gelsenkirchen Bocholt Recklinghausen

Fraunhofer IAIS

Masterthesis

3D Navigation for UAVs in GPS denied environments

3D Navigation eines UAVs in Umgebungen ohne GPS Empfang

Vorgelegt von: Stefan Wilkes
Hartmannsweilerstr. 16a
46119 Oberhausen
200823277

Datum: 12. Januar 2015

Gutachter: Prof. Dr. Hartmut Surmann
Dipl. Inf. Rainer Worst

Datum und Unterschrift des Betreuers

Unterlassungserklärung

Hiermit versichere ich, die Arbeit selbstständig angefertigt und keine anderen als die angegebenen und bei Zitaten kenntlich gemachten Quellen und Hilfsmittel benutzt zu haben.

Datum und Unterschrift des Verfassers:

Diese Masterthesis ist ein Prüfungsdokument. Eine Verwendung zu einem anderen Zweck ist mit dem Einverständnis von Verfasser und Prüfern erlaubt.

Abstract

Die Masterthesis „3D Navigation für UAVs in Umgebungen ohne GPS Empfang“ beschäftigt sich mit der autonomen Navigation eines *unmanned aerial vehicle*, zu Deutsch unbemannten Luftfahrzeugs im dreidimensionalen Raum. Es werden verschiedene Sensorgruppen sowie die notwendige Algorithmik zur Wahrnehmung der Umgebungsinformationen vorgestellt und hinsichtlich der Anwendungsmöglichkeiten in städtischen Such- und Rettungsmissionen evaluiert. Ferner wird ein erster Ansatz zur Umsetzung einer solchen Navigation vorgestellt, welcher durch das Robot Operating System sowohl in der Simulation als auch auf einem beliebigem UAV einsetzbar ist.

Schlüsselwörter: #ROS, #3D Pfadplanung, #UAV, #Navigation, #Stereo Vision, #Struktur durch Bewegung

The masterthesis “3D Navigation for UAVs in GPS denied environments” is concerned with the autonomous navigation of an unmanned aerial vehicle in three dimensional space. It analyses and explains several sensor groups and their corresponding algorithms to perceive the environemt in urban search and rescue missions. Afterwards the results are going to be evaluated. Furthermore a first approach for the implementation of such a navigation is presented, which can be used both in simulation and on an arbitrary UAV, due to the Robot Operating System.

Keywords: #ROS, #3D Path planning, #UAV, #Navigation, #Stereo Vision, #Structure from motion

Danksagung

An dieser Stelle möchte ich meinen Dank aussprechen. Ganz besonderer Dank geht an meinen Betreuer Herrn Prof. Dr. Hartmut Surmann, der mir im Laufe meines Studium und auch während der Bearbeitung dieser Thesis stets mit gutem Rat zur Seite stand. Die Begeisterung welche er der Forschung von Robotik und Bildverarbeitung zukommen lässt und gleichermaßen auch an die Studenten weitergibt, sorgt stets für Motivation und formte die Arbeit letzten Endes zu dem was sie ist. Ebenfalls großer Dank geht an meinen Zweitprüfer Dipl. Ing. Rainer Worst. Die Zusammenarbeit mit dem Fraunhofer Institut für Intelligente Analyse- und Informationssysteme hat mich um viele Erfahrungen bereichert und auch der internationale Zusammenschluss mit dem gesamten TRADR Team war eine Bereicherung für mich. Die Momente, welche ich bei der gemeinsamen Übung in Pisa erleben durfte, werden mir stets in Erinnerung bleiben. Somit geht mein Dank auch an das gesamte Team des Projektes. Ebenfalls bedanken möchte ich mich bei meinen Freunden und Kommilitonen Christopher Eulering und Dennis Lünsch, die sich bereit erklärt haben, diese Arbeit Korrektur zu lesen und sich jederzeit für anregende Diskussionen zur Verfügung gestellt haben.

Abschließender Dank gilt meinen Freunden, meiner Familie sowie meinen Eltern. Ihr habt mich stets motiviert, mein Studium zu meistern und mich jeder Zeit in jeglicher Hinsicht unterstützt. Ohne euch wäre all dies nicht möglich gewesen.

Vielen Dank!

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

Dieses Kapitel soll einen ersten Einblick in die Thematik dieser Arbeit und den damit verbundenen Problemstellungen liefern. Nach einer Definition der Kernproblematik wird der Verlauf der vorliegenden Arbeit vorgestellt.

1.1 Motivation

Der Einsatz von Robotern in Umgebungen, welche für Menschen auf Grund von gesundheitlichen Gefahren oder strukturellen Gegebenheiten nicht passierbar sind, ist bereits gängiger Alltag. Flugroboter liefern einen Überblick über das gesamte Szenario während Bodenroboter in Gebäude eindringen oder das Terrain genauer erkunden. Angebrachte Sensoren erfassen die Umgebung des Roboters und liefern visuelle oder metrische Informationen, die dem Einsatzteam helfen die Situation einzuschätzen und das weitere Vorgehen zu planen. Im Idealfall werden bei der maschinellen Erkundung bereits erste Opfer gefunden und deren Position ermittelt. Spezielle Sensorik weist ebenfalls auf Gefahrenquellen wie stickstoffhaltigen Rauch, Chemikalien oder gar Radioaktivität hin.

So wurden auch auch bei der Nuklearkatastrophe von Fukushima, welche als Folge des schweren Erdbebens im Jahre 2011 hervorgeht, Roboter zur ersten Erkundung des Gebietes eingesetzt. Selbst 3 Jahre nach dem Unfall spielen Roboter dort weiterhin eine bedeutende Rolle. Neben einfachen Aufräum- oder Säuberungsarbeiten von nuklearem Abfall, suchen die Roboter in den Tiefen des Kraftwerkes nach undichten Stellen und versuchen diese zu schließen. Der Einsatz eines Menschen in der Nähe dieses Ortes hätte zur Folge, dass der Fünf-Jahres-Grenzwert radioaktiver Strahlung binnen einer Stunde überschritten werden würde [26].

Abbildung 1: Roboter der Firma iRobot erkunden das Atomkraftwerk in Fukushima und liefern Videomaterial des Geschehens (Foto: Tepco)

Allgemein wird dieser Einsatzbereich als *Urban Search and Rescue*, kurz USAR bezeichnet. Zu Deutsch das Suchen und Retten in Städten oder Szenarien städtischer Bauweise. Die Kombination von Mensch und Technik soll eine schnellstmögliche Bergung der Opfer ermöglichen und benötigt somit eine spezielle Ausbildung zur Nutzung und Interaktion mit dem technischen Gerät. Die Steuerung der Roboter erfolgt in der Regel teleoperiert, sodass der Erfolg der Mission stets an die Fähigkeiten und Möglichkeiten des Operators gebunden ist. Geht beispielsweise die direkte Verbindung zum Roboter verloren, ist dieser nicht mehr kontrollierbar und wird lediglich ein weiteres Objekt, welches geborgen werden muss.

Im Idealfall arbeiten Mensch und Maschine autonom und erreichen durch Koordination und Kooperation das Ziel der Mission. Diesem Sachverhalt widmet sich auch das TRADR Projekt[55]. TRADR, kurz für *Long-Term Human-Robot Teaming for Robot Assisted Disaster Response* ist ein durch die europäische Union finanziertes Projekt, welches durch Zusammenarbeit von europäischen Forschungsunternehmen und den Endnutzern der Feuerwehr durchgeführt wird. Der Schwerpunkt des Forschungsprojekts liegt in der Entwicklung von Methodiken, welche es Teams von Menschen und Robotern ermöglichen, ein Katastrophenszenario über mehrere Missionen hinweg zu bewältigen. Dabei spielen Autonomie, Teamarbeit, Kommunikation, Planung und persistente Datenmodelle eine große Rolle.

Abbildung 2: Ruine eines ehemaligen Militärhospiz in Pisa. Testgelände für das TRADR Projekt. Viele der Trümmerteile sind auch für flexible Bodenroboter nur schwer oder gar nicht passierbar.

Das Team der Roboter besteht, wie bereits erwähnt, aus Flug- und Bodenrobotern. Stößt der Bodenroboter innerhalb eines Gebäudes auf ein Hindernis, welches er nicht überwinden kann, ergibt sich ein denkbares Kooperationsszenario. Der Flugroboter, nachfolgend auch UAV genannt, erhält die Position des Bodenroboters und setzt die Mission an der Stelle fort. Der Einsatz der Roboter kann somit fortgesetzt werden, wodurch

das Sicherheitsrisiko, Menschen in das Gebäude zu entsenden, genauer kalkuliert werden kann. Die Steuerung beziehungsweise die autonome Navigation eines UAVs innerhalb von Gebäuden bringt jedoch einige Problematiken mit sich, welche in der vorliegenden Arbeit analysiert und behandelt werden.

Die Durchführung der Arbeit erfolgte in Kooperation mit dem Fraunhofer Institut für Intelligente Analyse- und Informationssysteme, welches ebenfalls Teil des TRADR Konsortiums ist.

1.2 Problemstellung

Betrachtet man beispielsweise das Szenario eines eingestürzten Gebäudes, so lässt sich der Einsatzbereich in ein Innen- sowie in ein Außenareal einteilen. Der wesentliche Unterschied besteht darin, dass im Innenbereich einerseits kein GPS zur Verfügung steht und zudem geometrische Bewegungseinschränkung entstehen können.

Zunächst sollen UAV Bauarten ihrer Größe entsprechend kategorisiert und den verschiedenen Anwendungsbereichen zugeordnet werden.

1. Hubschrauber
2. Maxi UAV
3. Medium UAV
4. Micro UAV

Hubschrauber oder Maxi UAVs, wie sie häufig im militärischen Bereich eingesetzt werden, eignen sich auf Grund ihrer langen Flugzeiten hervorragend für großflächige Observationen aus der Luft. Diese Baugruppen sind dem Außenbereich zuzuordnen. Der Unterschied zwischen einem Micro UAV und einem Medium UAV liegt nicht sofort auf der Hand und wird in der Regel über die Tragkraft und damit auch die in der Ausstattung zur Verfügung stehende Sensorik unterschieden. Medium UAVs mit höherer Tragkraft und der daraus resultierenden höheren Spannbreite können zwar mehr Sensorik tragen, jedoch scheint ein Flug im Innenraum, geschweige denn in Korridoren mit vielleicht nur einem Meter breite, schier unmöglich.

Zu diesem Zweck existieren auch kompakte UAVs. Auf Grund der Traglast sind diese jedoch nur bedingt mit einem Laserscanner ausgestattet. Ein Alleinstellungsmerkmal, über das nahezu jede Art von Drohne verfügt ist das visuelle System, mit dem Kameradaten live übertragen werden.

1.2.1 Kernproblematik

Die Kernproblematik dieser Arbeit beschäftigt sich mit der autonomen Navigation eines UAVs im Innenbereich, beziehungsweise in Bereichen in denen kein GPS Signal zur Verfügung steht. Um den Verlauf der Arbeit zu präsentieren ist es zunächst nötig, Teilprobleme zu identifizieren und zur Zeit nicht lösbare Aufgaben auszuschließen.

1.2.2 Teilproblematiken

Zunächst gilt es eine passende Sensorik zu bestimmen, welche es dem UAV ermöglicht sich ohne GPS Signal zu lokalisieren. Die eigentliche Funktionsweise der Lokalisierung ist abhängig von der Sensorik und soll vorerst nicht weiter beachtet werden. Anschließend gilt es, die verschiedenen Sensorgruppen an Hand plausibler Daten zu evaluieren um somit final eine Spezifikation liefern zu können.

Es wird sich zeigen, dass zur Navigation des UAVs stets dreidimensionale Sensordaten benötigt werden, wodurch im Anschluss der Evaluierung ein erster Ansatz zur autonomen Navigation präsentiert werden kann. Wurde ein Weg zur Navigation bestimmt, gilt es diesen unter Vermeidung von Kollisionen zu verfolgen.

1.3 Struktur der Arbeit

Gemäß der bestimmten Teilproblematiken ist die vorliegende Arbeit in 7 Kapitel unterteilt. Im folgenden Kapitel wird zunächst der aktuelle Stand aus Technik und Forschung präsentiert. Daraufhin wird das allgemeine Design des zu realisierenden Systems analysiert sowie die verwendeten Hard- und Softwarekomponenten vorgestellt. Die darauffolgenden Kapitel beschäftigen sich mit den Teilproblemen selbst und stellen die Ergebnisse vor. Dies sind die Kapitel:

- Verfahren zur Wahrnehmung der Umgebung
- Evaluierung der Verfahren
- Navigation

Ein abschließendes Fazit sowie ein Ausblick über zukünftige Ausarbeitungen bilden den Abschluss dieser Arbeit. Der Anhang enthält Organisatorisches zur Durchführung der Arbeit sowie Hinweise, welche für zukünftige Projekte verwendet werden können.

2 Stand der Technik

In diesem Kapitel wird der zu Grunde liegende Stand aus Technik und Forschung präsentiert. Neben einem Überblick verschiedener UAV Typen, werden Forschungsansätze zur dreidimensionalen Umgebungswahrnehmung präsentiert. Diese Verfahren sind für die autonome Navigation von essentieller Bedeutung, weshalb deren Anwendbarkeit im weiteren Verlauf der Arbeit genauer analysiert wird.

2.1 Unmaned Aerial Vehicles

Unmaned Aerial Vehicle, kurz UAV, sind Luftfahrzeuge verschiedener Größenordnung, welche autonom oder vom Boden kontrolliert fliegen. Der Anwendungsbereich streckt sich dabei vom kommerziellen Heimbedarf über observatorische Flugeinsätze bis hin zu militärischen Einsatzzwecken. Einen Überblick für den Markt der Privatanbieter zu liefern ist derweil unmöglich geworden. Ob im Modellbau- oder im Spielwarenbereich locken viele Hersteller mit unterschiedlichen Angeboten. Dabei werden relativ einfach aufgebaute Micro UAVs mit einem Gewicht von ca. 300 Gramm zu sehr kleinen Preisen angeboten. Aber auch professionelle Umsetzungen sind auf dem Markt zu finden. So bieten Firmen wie Parrot[46] oder DJI[47] robuste Quadcopter mit erweiterter Sensorik an. Diese UAVs werden auch häufig in der Lehre und der Forschung verwendet, da diese leicht zu ersetzen und auf Grund der einfachen Bauweise auch in Innenräumen anwendbar sind.

Abbildung 3: Kommerzieller Nano Quadrotor[40]. Ausgestattet mit grundlegender Stabilitätsensorik. Gewicht ca. 300g. Kosten: Ca. 50 Euro.

Anders als bei den soeben erwähnten Modellen, beschäftigen sich Firmen wie Ascending Technologies oder die Fraunhofer Gesellschaft mit der Entwicklung von individuellen

Abbildung 4: Kommerzieller semi-professioneller Quadrotor DJI Phantom II[47]. Ausgestattet mit verschiedenen Sensoren zur Positions- und Inertialsmessung. Ebenfalls mit Kamerasystem und GPS ausgestattet. Kosten: Ab 599 Euro.

Modellen, welche stets auf den Einsatzzweck abgestimmt sind und somit die besten Ergebnisse liefern sollen. So wurde seitens der Fraunhofer im Nifti Projekt das Mosquito UAV entwickelt. Ein Quadrotor, ausgestattet mit einem Laserscanner und verschiedener inertial Sensorik. Mit Hilfe dieses UAVs ist es möglich gezielt GPS Positionen anzufliegen und Kameraaufnahmen sowie zweieinhalb dimensionale Abtastungen der Umgebung zu generieren. Dabei werden aufeinander folgende zweidimensionale Laserscans der gemessenen Höhe nach akkumuliert und visualisiert[41].

Ascending Technologies ist für die Entwicklung des UAVs im TRADR Projekt verantwortlich und stellt auch den in dieser Arbeit verwendeten Prototypen zur Verfügung, welcher im nächsten Kapitel näher vorgestellt wird.

2.2 Dreidimensionale Umgebungswahrnehmung

Forschungen zur dreidimensionalen Umgebungswahrnehmung beschäftigen sich mit der Erfassung von strukturellen Merkmalen der Umgebung. Dabei ist je nach der verwendeten Sensorgruppe komplexe Algorithmik notwendig. Dieser Abschnitt soll aktuelle Forschungsansätze vorstellen und diese an Hand ihrer grundlegenden Erkennungsweise klassifizieren.

Stereo visuelle Verfahren

Stereo visuelle Verfahren beschäftigen sich mit der Erfassung der Umgebung an Hand zweier Kameras. Das Verfahren findet seinen Ursprung in der Biologie und basiert auf dem menschlichen Sehprozess. In der Bildverarbeitung gehört die Stereoskopie zu den

Grundlagen und Arbeiten von Lauterbach[32] und Klingbeil u.a.[28] beschäftigen sich mit der Integration des Wahrnehmungsverfahrens auf dem UAV. Lauterbach kommt zu der Erkenntnis, dass die alleinige Verwendung einer Stereo Kamera zwar akzeptable Distanzmessungen liefert, jedoch für einen zuverlässigen Einsatz im Flug ungeeignet ist. Auch Klingbeil u.a. nutzen die Information der Stereo Kamera lediglich zur Schätzung der Eigenbewegung und lassen diese als Teilinformation in eine Fusion mit weiterer Sensorik eingehen. Vorteilhaft ist jedoch, dass die Rekonstruktion der Tiefeninformation, auf Grund des geometrischen Aufbau der Kameras, metrisch korrekt ist. Was hingegen bei der mono visuellen Rekonstruktion nicht der Fall ist.

Mono visuelle Verfahren

Die mono visuelle Rekonstruktion beschäftigt sich mit der Erfassung von Umgebungsinformation an Hand kontinuierlicher Aufnahmen einer einzelnen Kamera. Diese Art der Rekonstruktion wird international auch als *Structure from Motion* bezeichnet. Hier gilt es die Trajektorie zwischen den einzelnen Aufnahmen zu schätzen. Dazu existieren verschiedene Ansätze, welche sich in die Klassen der punktbasierten Verfahren oder der featurebasierten Verfahren unterteilen lassen.

Punktbasierte Verfahren betrachten das gesamte Bild zur Rekonstruktion der Bewegung. Als Beispiel ist hier der optische Fluss zu nennen, wie ihn auch Campbell u.a.[11] benutzen. Nach der Bestimmung des Flusses teilen Campbell u.a. das Bild in eine Boden-, eine Himmels- und eine Horizontsektion ein. Anschließend kann die Odometrie rekonstruiert werden, wobei die Translation über die Bodensektion und die Rotation über die Himmelssektion bestimmt wird. Ein neueartiges punktbasiertes Verfahren stammt von Forster u.a.[19]. Anstelle von rechenintensiven Feautureberechnungen wird hier direkt auf den Intensitäten der Pixel gearbeitet. Über eine probabilistische Methode werden Ausreißer detektiert und das dreidimensionale Modell geschätzt.

Featurebasierte Verfahren hingegen arbeiten auf dem Vergleich von wiedererkannten Merkmalen in den Bildern und schätzen somit die Bewegung zwischen diesen. Das Wiederkennen setzt jedoch das Finden von robusten Featurepunkten voraus, was häufig mit erhöhtem Rechenaufwand verbunden ist. Folkesson u.a.[18], Krajnik u.a.[30] sowie Dijkshoorn[12] stellen in ihren Arbeiten featurebasierte mono visuelle Verfahren auf Basis der SIFT und SURF Deskriptoren vor. Doch auch diese Verfahren arbeiten in der Regel mit einem erweitertem Kalman Filter um verschiedene Sensorinformationen zu kombinieren und somit erst eine genaue Positionsschätzung zu ermöglichen. Auch die Kombination mit hochwertigen Distanzinformationen wie die eines Laserscanners ist hier möglich[56][28].

Ein neues Verfahren welches fernab von punkt- oder featurebasierten Verfahren arbeitet stellten Engel u.a.[16] im Jahre 2013 vor. Sie präsentierten ein Verfahren zur semidichten Wahrnehmung der Umgebung durch eine einfache Kamera, welches in Echtzeit

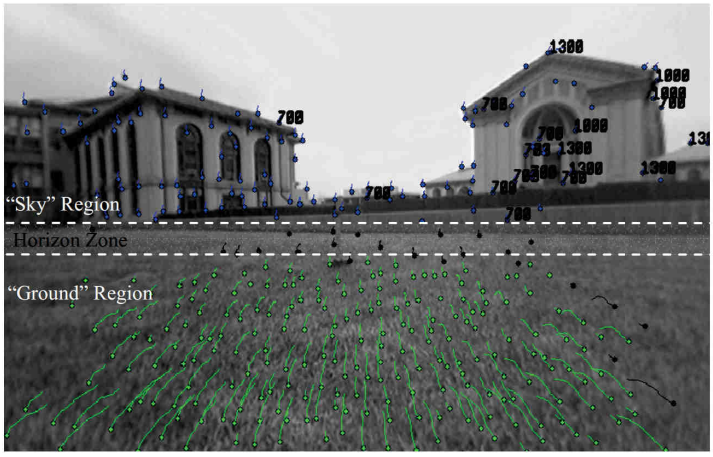

Abbildung 5: Berechnung des optischen Flusses zur Rekonstruktion der Odometrie nach Campbell[11].Vektoren über dem Horizont dienen der Rotationsschätzung, Vektoren unter dem Horizont der Translationsschätzung.

arbeiten soll. Dabei wird die Tiefe über die Gradienten der einzelnen Bildinformationen bestimmt. Diese Gradienten werden registriert und in neu eintreffenden Bilder verglichen um so das dreidimensionale Modell zu bestimmen. Das Verfahren hat somit eine Dichte wie punktorientierte Verfahren jedoch mit der Performanz, wie die eines reinen Featurevergleiches.

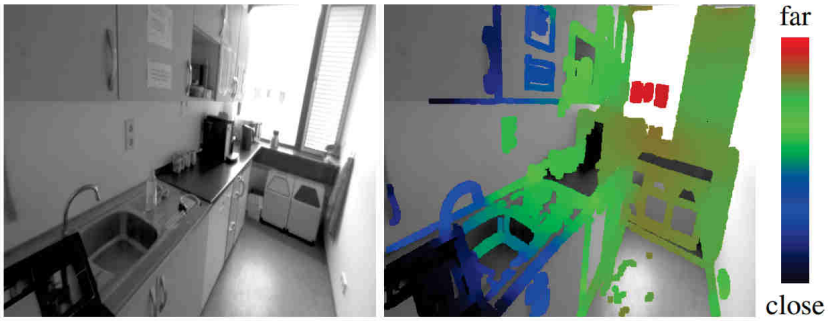

Abbildung 6: Semi-Dense Visual Odometry for a Monocular Camera von Engel u.a.[16]. Über die Gradienten in aufeinander folgenden Bildaufnahmen wird die Tiefe rekonstruiert und eine Karte der Umgebung gebildet.

Ein weiteres Verfahren zur monovisuellen Wahrnehmung liefert das Parallel Tracking and Mapping Framework, kurz PTAM [27]. Korrespondierende Punkte werden über einfache Eckdetektoren bestimmt und in einer Karte registriert. Das Tracken der einzelnen Punkte sowie das Bauen der Umgebungskarte läuft hier parallel. Dadurch ist die Möglichkeit gegeben, die Pose jederzeit zu korrigieren sofern ein zuvor bestimmter Keyframe wiedererkannt wird. Ansonsten arbeitet das Verfahren ähnlich zu dem optischen Fluss. Engel

u.a.[15] portierten das Framework 2012 auf die ARDrone der Firma Parrot wodurch eine gute Bestimmung der Lokalität- und eine relative Navigation ermöglicht wurde. Auch Neyman[43] verwendete den PTAM Algorithmus um Wände in der direkten UAV Umgebung zu detektieren.

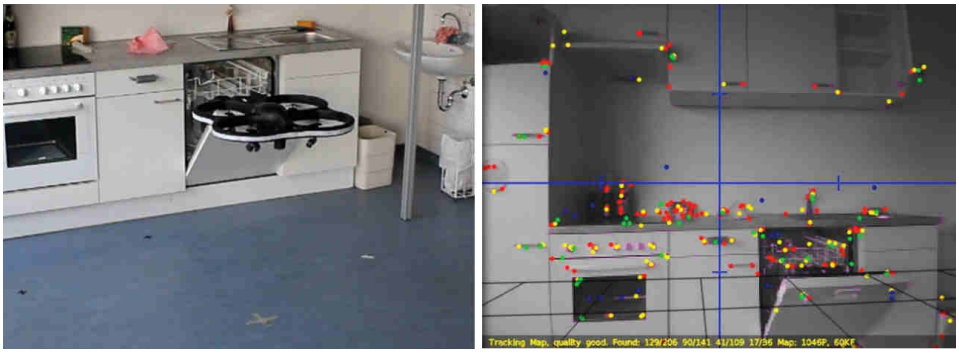

Abbildung 7: PTAM Algorithmus zur Stabilisierung der Parrot ARDrone nach Engel u.a.[15]. Ermöglicht ebenfalls einfache relative Navigation.

Neben den soeben vorgestellten Rekonstruktionsverfahren existieren auch abstrakte Ansätze zur Navigation mit einer einzelnen Kamera. So stellen Bills u.a.[8] in ihrer Arbeit ein Verfahren vor, welches zunächst die Umgebung als Korridor, als Treppenhaus oder als Raum bzw. unbekannte Region klassifiziert. Anschließend werden je nach Kategorie einfache Algorithmen zur direkten Navigation verwendet. In einem Korridor wird beispielsweise der klassische Fluchtpunktansatz verwendet um diesen zu passieren. Im Treppenhaus hingegen wird das Verfahren dahingehend erweitert, sodass stets mittig der Stufen ein kontinuerlicher Auf- bzw. Abstieg statt findet. In einem Raum oder einer unbekannten Kategorie beginnt das UAV an Hand der Ultraschallsensoren eine Exploration um zurück in eine bekannte Klasse der Umgebung zu finden.

Distanzsensorik

Neben der visuellen Umgebungswahrnehmung existieren Sensoren, welche die Umgebungsinformationen in Form von der Distanz zum Hindernis instantan messen können. In der klassischen Robotik werden so mit Hilfe eines zweidimensionalen Laserscanners Karten berechnet, in denen sich der Roboter gleichzeitig lokalisieren kann. Durch die kontinuierliche Rotation des Scanners können diese Laserscans ebenfalls dreidimensionale Modelle aufnehmen, welche nach einem Verfahren von Nüchter u.a.[42] zu einem Gesamtmodell der Umgebung verrechnet werden können. Droeschel u.a.[13] arbeiten zur Zeit ebenfalls an einem solchen Messverfahren auf dem UAV. Hier werden unter Verwendung eines kleinen Laserscanners, ebenfalls an einer Rotationseinheit befestigt, mit einer Rate von zwei Hertz Umgebungsmodelle während des Fluges aufgenommen und unter Verwendung weiterer Sensorik in einer Karte registriert.

Ein relativ neuartiges Sensorsystem ist die Kinect der Firma Microsoft oder die xTion der Firma Asus. Beide Kamerasysteme projizieren ein Infrarotmuster und messen gleichzeitig die Reflektion der Matrix. Dadurch ist die Kamera in der Lage instantan dreidimensionale Abbildungen der Umgebung zu erfassen und gleichzeitig die passenden RGB Information zuzuordnen. Unter Verwendung dieser Information stellten Endres u.a.[14] ein Verfahren vor, welches die einzelnen Messungen auf Basis der RGB Informationen miteinander registriert und so komplexe Umgebungsmodelle erstellt.

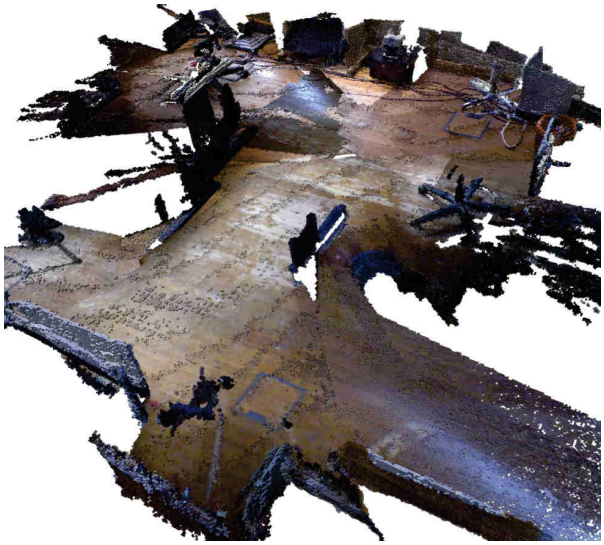

Abbildung 8: Ergebnis des RGB-D SLAM Verfahren nach Endres u.a.[14]. Darstellung als Voxel-Grid (1cm).

Einen weiteren Ansatz stellen Michels u.a. vor[39]. Durch die Verwendung eines Laserscanners und zugehörigen mono Kameraaufnahmen wird ein maschineller Lernalgorithmus ausgeführt, welche die Tiefe von bestimmten Mustern der Kamera einem realen Messwert zuordnet. Nach dem Lernprozess ist das System in der Lage die Tiefe lediglich an Hand neuer Kameraufnahmen der erlernten Szene zu schätzen. Diese Art der Umgebungsinterpretation ist jedoch für den Einsatzbereich dieser Arbeit nicht von Bedeutung, da ein willkürliches USAR Szenario nicht generalisiert und zuvor erlernt werden kann.

2.3 Robot Operating System

Seitens der allgemeinen Software bildet das Robot Operating System ganz klar den Stand der Technik. Neben einer Vielzahl von Treibern und Basisalgorithmen werden ebenfalls umfangreiche Frameworks zur Loaklisierung sowie zur Navigation verschiedener Robotersysteme zur Verfügung gestellt. ROS ist universell einsetzbar und wird im weiteren Verlauf der Arbeit genauer vorgestellt.

3 Systemdesign

Dieses Kapitel soll Aufschluss über die verwendeten Hard- und Softwarekomponenten geben. Im Anschluss werden der Gesamtaufbau des Systems, deren Grundfunktionalitäten sowie die Schnittstellen zwischen der Hardware und den entwickelten Software-Komponenten beschrieben.

3.1 Software

Für die Kommunikation mit dem UAV wird das Roboterbetriebssystem ROS[48] verwendet. ROS, kurz für Robot Operating System, ist kein eigenes Betriebssystem in diesem Sinne sondern kann eher als komplexes Framework zur Ansteuerung und Interaktion von Komponenten wie Sensoren, Robotern oder Logik-/Verarbeitungseinheiten gesehen werden. Zudem stellt ROS eine Vielzahl von quell-offenen Algorithmen für Basis-Aufgaben wie Navigation, Kartierung oder Lokalisierung bereits zur Verfügung.

Die Hauptaufgabe von ROS besteht darin, eine Hardwareabstraktion zu ermöglichen. Es existieren eine Vielzahl von vordefinierten Botschaften, welche in den eigentlichen Komponenten, nachfolgend Nodes genannt, genutzt werden sollen. Es entsteht ein konsequenter Datenaustausch über standardisierte Schnittstellen. Der Integrationsaufwand von komplexen Bibliotheken entfällt somit völlig. Es muss lediglich sichergestellt werden, dass eigens entwickelte Nodes Botschaften senden, welche von externen Nodes auch interpretiert werden können.

Das Dateisystem von ROS ist hierarchisch aufgebaut. Eine Komponente wird über ein sogenanntes *meta-package* angeboten. Dieses beinhaltet weitere *packages* welche Teilaufgaben definieren. Die unterste Ebene ist der bereits erwähnte *Node*, das eigentliche ausführbare Programm. Zusammengefasst bedeutet das: Ein *meta-package* repräsentiert einen Oberbegriff von Algorithmen, wie beispielsweise die Navigation. Das *package* ist eine Teilaufgabe, was in diesem Fall die Lokalisierung, die Pfadplanung oder die Regelung bedeuten kann. Der Node ist die konkrete Implementierung dieser Aufgabe.

Entwickelt wurde das System 2007 am Stanford Artificial Intelligence Laboratory im Rahmen des Stanford AI Robot Projekts. Mittlerweile existieren weit über 4000 offiziell angebotene Pakete[50]. Bis April 2012 wurde das Projekt von WillowGarage betreut und weiterentwickelt. Heute wird ROS von der OpenSource Robotics Foundation unterstützt und durch eine stetig wachsende Community weiterentwickelt.

Neben ROS wurde für Aufgaben aus der Bildverarbeitung die OpenCV[44] Bibliothek verwendet. Die Open Computer Vision Library, wurde ursprünglich von der Firma Intel entwickelt und anschließend durch Willow Garage gepflegt und weiterentwickelt. Nach deren Auflösung hat die Firma Itseez diese Aufgabe übernommen. Neben einfachen

Matrix-Operationen bietet die OpenCV auch einen plattformunabhängigen Zugriff auf Datei- und Kamerahardwareebene. Ferner stellt die Bibliothek eine umfangreiche Sammlung von Algorithmen aus den Bereichen Bildoperationen, Videoanalyse, Kamerakalibirierung und 3D-Rekonstruktion, 2D Feature Erkennung, Objekterkennung, maschinelles Lernen, Visualisierung und weiteren Thematiken zur Verfügung. Viele dieser Algorithmen werden nach außen hin über einen einfachen Funktionsaufruf angeboten, deren Verwendung und Funktionalität an passenden Stellen im Laufe dieser Arbeit präsentiert wird.

Die erste Hauptaufgabe in der Durchführung dieser Arbeit bestand darin, einen Treiber-Knoten zu entwickeln, welche die abstrakten Schnittstellen seitens ROS in passende Hardwarekommunikationsstrukturen übersetzt und somit einen Daten- beziehungsweise Befehlsaustausch mit dem UAV realisiert. Die Funktionsweise des Treibers soll in diesem Kapitel näher erläutert werden. Zunächst muss dazu jedoch auf die eigentliche Hardware eingegangen werden.

3.2 Hardware

Dieser Abschnitt beschreibt die Hardware der verwendeten UAVs. Im TRADR Projekt wird die Drohne von der Firma Ascending Technologies entwickelt und zur Verfügung gestellt. Der endgültige Aufbau des kommerziellen Produkts stand jedoch zum Zeitpunkt der Durchführung dieser Thesis noch nicht fest, so dass vorerst ein Prototyp zur Entwicklung benutzt wurde. Bei diesem Prototyp handelt es sich um das Modell Pelican[5]. Auf Grund des Asctec Auto Pilot und einem zugehörigen Interface zur Kommunikation mit der Platine ist es jedoch möglich die Software auch auf der finalen Version des UAVs zu nutzen.

3.2.1 Ascending Technologies: Pelican

Bei der Pelican Drohne handelt es sich um einen Quadrotor, welches mit einer Größe von 651 x 651 x 188 cm bereits im Medium-UAV Segment anzusiedeln ist. Angetrieben wird das UAV von vier bürstenlosen Elektromotoren mit einer Spitzenleistung von je 160 Watt. Mit einer maximalen Nutzlast von 650 Gramm ergeben sich Flugzeiten von bis zu 16 Minuten.

Die Pelican Drohne verfügt über einen Luftdrucksensor, einem 3-Achsen Gyroskop, einem 3-Achsen Beschleunigungssensor, einem GPS Sensor und einem Kompass. Ferner wurde ein Hukoyu Laserscanner zur schnellen zweidimensionalen Abtastung der Umgebung installiert. Weiterhin verfügt die Pelican über ein 3-Schichten Computersystem, welches zur Ansteuerung und Abgreifen der Sensordaten genutzt werden kann. Die unterste Schicht, der Low-Level Prozessor, dient zur direkten Ansteuerung der Sensoren und

Abbildung 9: Entwicklungsprototyp: Asctec Pelican

Aktoren. Die Software des LL Prozessors ist nicht einsehbar, da hier Fusionen der Sensordaten stattfinden, welche das stabile Flugverhalten der Drohne sicherstellen. Die zweite Schicht bildet der High-Level Prozessor. Dieser dient zum Setzen von Wegpunkten, zur Weitergabe von direkten Steuerdaten oder zum Einsehen der fusionierten Messwerte. Die Kommunikation zwischen LL Prozessor und HL Prozessor läuft über einen internen SPI Bus. Diese beiden Prozessoren bilden den Asctec Autopilot.

Interessant für die weitere Entwicklung ist die dritte Schicht des Systems. Das Asctec Mastermind. Das Mastermind bildet einen vollwertigen PC, welcher über einen Intel Core2Duo mit 2 x 1.86 GHz sowie 4 GB DDR3 RAM verfügt. Über eine serielle Schnittstelle wird direkt mit dem HL Prozessor kommuniziert. Die Kommunikation läuft hierbei über das seitens Ascending zur Verfügung gestellte Asctec ACI Protokoll. Als Betriebssystem kommt ein Ubuntu 12.04 mit installiertem ROS Hydro zum Einsatz.

Das Board stellt neben einer WLAN Schnittstelle fünf USB 2.0 Ports zur Verfügung an die weitere Sensorik angeschlossen werden kann. Über eine dieser Schnittstellen wird beispielsweise mit dem Hukoyu Laserscanner kommuniziert.

Der entwickelte ROS Treiber hat die Aufgabe, grundlegende Sensorinformationen und Steuerkommandos zwischen UAV und ROS auszutauschen. Dabei handelt es sich um folgende Daten:

- IMU (Beschleunigung, Winkelgeschwindigkeit, fusionierte Winkelposition)
- Flugzeit
- Batteriestatus

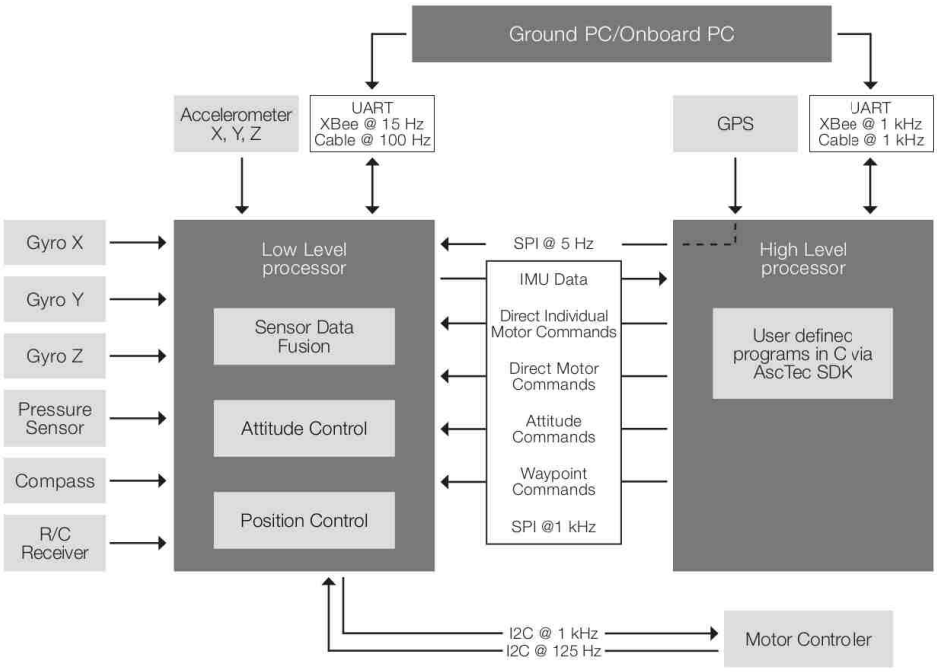

Abbildung 10: Schematische Übersicht des Asctec Pelican AutoPilot[4]

- GPS Status (Wegpunkte, Status, Aktuelle Position)
- Höhe
- Allgemeiner Status
- Manuelle Steuerung (Neigungswinkel, Schub und Auftrieb)

Diese Informationen wurden teils auf allgemeine ROS-Botschaften und teils auf eigens definierte Botschaften übersetzt. Die Daten werden dabei direkt über das Mastermind aus dem HL Prozessor ausgelesen werden. Für weitere Informationen über den Datenfluss sei nochmals auf die Dokumentation des Asctec ACI Protokolls[3] verwiesen.

Der Treiber kann somit als Übersetzer angesehen werden und bietet, abgesehen von der Wegpunkt - Verwaltung für einen autonomen Outdoor Flug über GPS, keinerlei Intelligenz.

Das Treiberpaket nutzt die standardisierten ROS - Botschaften sensor_msgs und geometry_msgs. Diese werden für die IMU, die manuelle Steuerung und für den GPS Status benutzt. Eine weitere Abhängigkeit ist das Paket pelican_msgs, welches eigens definierte Botschaften für die Flugzeit, die Höhe, den Batteriestatus und das Setzen von Wegpunkten anbietet. Das Paket roscpp ermöglicht es, die ROS Schnittstellen in C++ zu nutzen. Zu guter Letzt das Paket pelican_launch, welches eine Sammlung von Skripten

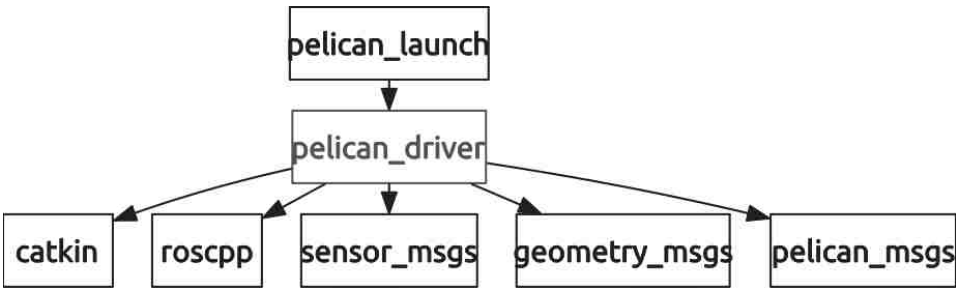

Abbildung 11: Struktur des pelican_driver Pakets. Die Pakete *catkin* und *roscpp* sind Systemabhängigkeiten, welche zur Kompilierung erforderlich werden

zur Verfügung stellt, um die Treiberkomponenten komfortabel zu starten.

Weiterhin wurde ein Paket entwickelt, welches die Sensorbotschaften der Pelican empfängt und diese dem ROS Transformation Framework zur Verfügung stellt. Das Transformation Framework soll dazu dienen, die Transformationen zwischen den einzelnen Roboterkoordinatensystemen zu registrieren und anderen Komponenten zur Verfügung zu stellen. Voraussetzung dazu ist, dass jede Komponente in ROS, welche über ein eigenes Koordinatensystem verfügt, ihre Transformation zum Eltern-Koordinatensystem im Framework registriert. Steht jede Teiltransformation zur Verfügung ist das Framework in der Lage jeder ROS Komponente die Koordinaten eines Systems im angefragten Bezugssystem mitzuteilen.

Somit ist jeder ROS Knoten in der Lage, die relative Ausrichtung des UAVs und dessen Sensorik abzufragen. Dies ist eine Grundvoraussetzung für die Kartierung, insbesondere für die korrekte Registrierung der Sensorinformationen.

3.2.2 Parrot ARDrone

Für die Evaluierung von Algorithmen, welche auf rein visueller Sensorik arbeiten, stand ebenfalls eine AR Drone der Firma Parrot[46] zur Verfügung. Die Drohne hat eine Abmessung von 50 cm x 50 cm x 15 cm und ist somit im Micro / Medium Bereich einzuordnen. Auf Grund des Eigengewichts von ca. 400 Gramm und der daraus resultierenden geringen Traglast, verfügt die AR Drone über deutlich weniger Sensorik.

Angetrieben wird das UAV von vier bürstenlosen DC-Motoren mit einer Leistung von je 15 Watt. Neben einer IMU stehen ein Ultraschallsensor und eine WLAN Verbindung zur Interaktion mit dem Controller Board zur Verfügung. Ferner verfügt die Drohne über zwei Kameras: Eine hochauflösende Kamera für das Ego-Feld sowie eine QVGA Kamera für die Bodensicht. In der 2.0 Version der Drohne wurde ebenfalls ein Barometer sowie ein optionaler GPS Sensor verbaut.

Das Montieren eines hochwertigen, leistungsstarken PCs oder schwerer Sensorik ist nicht möglich, was voraussetzt, dass die Drohne stets mit einer Bodenstation verbunden ist, welche die Sensordaten verarbeitet und Steuerbefehle an den internen Controller weiterleitet. Auf dieser Bodenstation, ein Notebook, PC oder ähnliches, arbeitet ROS. Ein Treiber, welcher sich über WLAN mit der AR Drone verbindet und die Kommunikation mit ROS realisiert wird bereits über die Community offiziell zur Verfügung gestellt.

Abbildung 12: ARDrone der Firma Parrot

3.2.3 Stereo RGB-D Testsystem

Zur Aufnahme der RGB-D und Stereo Datenreihe wurde ein einfaches System bestehend aus zwei Logitech C270[36] Kameras und einer Asus Xtion[6] konzipiert. Die Xtion ist vergleichbar mit der klassischen Kinect Kamera, benötigt jedoch keine zusätzliche Stromversorgung. Die C270 Kameras haben eine Auflösung von 1280x720 Pixeln und wurden auf einem Aluminiumprofil fixiert, sodass die extrinsischen Parameter der Kameras nur einmalig bestimmt werden müssen. Die Kalibrierung einer Kamera wird in Kapitel 4.1 näher erläutert.

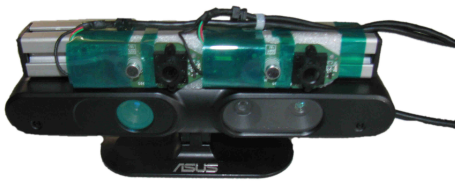

Abbildung 13: Testsystem zur Aufnahme der Stereo und RGB-D Datensätze

3.2.4 3D Laserscanner

Wie bereits im Stand der Technik vorgestellt[13], besteht die Möglichkeit, ein UAV mit einem rotierendem Laserscanner auszurüsten um dreidimensionale Messungen durch-

zuführen. Um diese Sensortechnik zu evaluieren, wurde zur Aufnahme von Testdaten ein externer Scanner, bestehend aus einer Rotationsplattform und einem darauf befestigtem LMS100 Laserscanner der Firma Sick verwendet. Das Verfahren zur Fusion von zweidimensionalen Teilaufnahmen in eine dreidimensionale Messung wird im nächsten Kapitel vorgestellt.

Abbildung 14: 3D Laserscanner bestehend aus Rotationsplattform und klassischem Laserscanner

3.3 Inertial Measurement Unit (IMU)

Wie sich bereits im vorherigen Abschnitts dieses Kapitels herausgestellt hat, ist das Herzstück eines jeden UAVs die Inertial Measurement Unit, kurz IMU. Diese Sensorgruppe kann aus verschiedenen Sensoren bestehen und dient zur Schätzung der Orientierung sowie zur Stabilisierung des Flugverhaltens. Ohne eine IMU wäre es kaum möglich einen Quadkopter stabil in der Luft zu halten, geschweige denn zu kontrollieren.

Der Aufbau einer IMU wird über den Freiheitsgrad ausgedrückt. Eine optimale IMU hat 9 DOF (Degrees of freedom). Diese besteht aus einem 3 Achsen Gyroskop, einem 3 Achsen Magnetometer und einem 3 Achsen Beschleunigungssensor. Jeder dieser Elementarsensoren liefert einen Vektor, welcher ein bestimmtes Messergebnis enthält. Durch die Fusion dieser Vektoren lässt sich die Orientierung der IMU und damit die Orientierung des UAVs schätzen. Das Ergebnis dieser Schätzung wird direkt zur Regelung an die Motorsteuerung weitergeleitet und ist somit Alleinstellungsmerkmal über das Flugverhalten.

Auf Grund dessen ist die Implementierung einer solchen Sensor-Fusion in der Regel nicht freigegeben. Jeder Hersteller optimiert stets die Algorithmik der IMU um das Produkt stabiler zu machen. So wird beim Asctec Autopilot zwar dem Anwender das Ergebnis zur Verfügung gestellt, die Berechnung selbst erfolgt jedoch im nicht zugänglichen LL Prozessor. Nachfolgend sollen dennoch zwei allgemeine Ansätze zur Fusion der Messwerte

einer IMU präsentiert werden um auf die Problematiken der Odometrie eines UAVs hinzuweisen.

Das Magnetometer liefert im Idealfall einen Vektor, welcher auf den Nordpol zeigt. Der Beschleunigungssensor repräsentiert einen Vektor, der zum Erdmittelpunkt führt. Befindet sich das UAV im Ruhezustand hat der Vektor stets einen Betrag von $9,81\frac{m}{s^2}$, da das UAV die Gravitation der Erde kompensiert. Das Gyroskop liefert, analog zum Beschleunigungssensor, die Geschwindigkeiten der Kreisel und damit die Winkelgeschwindigkeiten der Achsen.

Nachfolgend sollen drei allgemeine Ansätze zur Sensorfusion beschrieben werden.

3.3.1 Bestimmung der Orientierung durch Beschleunigungssensor und Magnetometer

Die Messwerte des Magnetometers und des Beschleunigungssensor liefern zwei Vektoren, welche stets senkrecht zueinander stehen. Den Vektor zum Erdmittelfeld (nachfolgend Z) und den Vektor zum Nordpol (nachfolgend Y). Durch das Kreuzprodukt dieser beiden Vektoren erhält man ein vollständiges Koordinatensystem und damit auch ein stetig konstantes Welt-Referenzsystem.

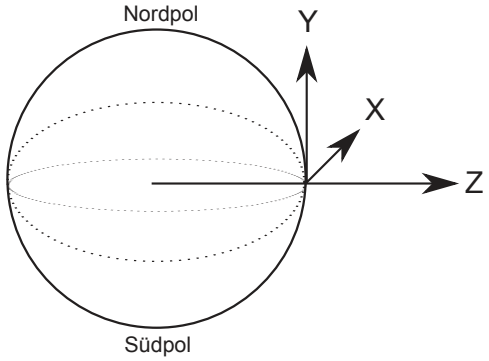

Abbildung 15: Darstellung des konstanten Koordinatensystems auf Grund von Magnetometer und Beschleunigungssensor

Aus diesen orthogonalen Vektoren kann direkt die Rotationsmatrix gebildet werden. Die Abszisse entsteht, wie bereits erwähnt, durch das Kreuzprodukt der Sensorvektoren:

$$\vec{X} = \vec{Mag} \times \vec{Acc} \tag{3.1}$$

Die Applikate ist trivial und zeigt stets in den Himmel um die Erdanziehung zu kompensieren:

$$\vec{Z} = \vec{Acc} \tag{3.2}$$

Damit die Matrix definitiv eine orthogonale Basis bildet, wird der letzte Vektor, die Ordinate, rekursiv gebildet. Es handelt sich jedoch um den Ergebnisvektor des Magnetometers.

$$\vec{Y} = \vec{Acc} \times \vec{X} \approx \vec{Mag} \tag{3.3}$$

Und somit gilt für die Drehmatrix R:

$$R = \begin{bmatrix} \vec{X}^T \\ \vec{Y}^T \\ \vec{Z}^T \end{bmatrix} = \begin{bmatrix} x_1 & x_2 & x_3 \\ y_1 & y_2 & y_3 \\ z_1 & z_2 & z_3 \end{bmatrix} \tag{3.4}$$

Aus dieser Matrix können die Roll-, Nick- und Gierwinkel direkt bestimmt werden:

$$\begin{aligned} pitch &= atan2(-z_1, \sqrt{x_1^2 + y_1^2}) \\ yaw &= atan2(\frac{y_1}{cos(pitch)}, \frac{x_1}{cos(pitch)}) \\ roll &= atan2(\frac{z_2}{cos(pitch)}, \frac{z_3}{cos(pitch)}) \end{aligned} \tag{3.5}$$

Das Verfahren ist relativ genau. Jedoch sind besonderes im Innenbereich viele elektromagnetische Störfelder, die das Magnetometer beeinflussen können. Diese Störungen machen sich durch spontane Odometrieänderungen bemerkbar. Vorteilhaft ist hier die Orientierung zu einem festen Bezugssystem. Hat sich das Magnetometer beruhigt, so wird sofort die korrekte Pose eingenommen.

Jedoch ist das Verfahren anfällig für hohe Beschleunigungen oder den freien Fall, da hier der Beschleunigungsvektor gegen Null geht und kein orthogonaler Raum mehr gebildet werden kann.

3.3.2 Bestimmung der Orientierung durch Gyroskop

Die Bestimmung der Orientierung beim Gyroskop ist trivial und bedarf keiner weiteren Sensoren. Das Gyroskop liefert Winkelgeschwindigkeiten. Iteriert man diese nach der Zeit, erhält man konkrete Winkelpositionen. Im diskreten Anwendungsfall geht diese Iteration in eine faktorisierte Summe über.

$$\begin{pmatrix} roll \\ pitch \\ yaw \end{pmatrix} (t) = \begin{pmatrix} roll \\ pitch \\ yaw \end{pmatrix} (t-1) + \begin{pmatrix} \omega_{roll} \\ \omega_{pitch} \\ \omega_{yaw} \end{pmatrix} * t \tag{3.6}$$

Zu beachten ist jedoch, dass ein Gyroskop rauscht. Diese minimalen Messschwankungen werden nun aufsummiert und bilden schon nach kurzer Zeit einen gravierenden Messfehler. Dieses Phänomen wird als der Gyro-Drift[62] bezeichnet.

3.3.3 Optimierung der Orientierung durch Fusion der beiden Verfahren

Betrachtet man die soeben vorgestellten Verfahren, so zeigen sich Vor- und Nachteile in beiden Ansätzen. Die Orientierung über das Gyroskop liefert für einen kurzen Zeitbereich die korrekten Änderungen. Über einen längeren Zeitraum ist das Ergebnis auf Grund des Drifts jedoch unbrauchbar.

Die Berechnung der Orientierung über den Beschleunigungssensor und das Magnetometer liefert eine relative Position zum Weltkooridiantensystems, welche auf kurze Zeit auf Grund von elektromagnetischen Feldern jedoch stark abweichen kann.

Es liegt somit nahe diese beiden Verfahren zu fusionieren und somit eine optimierte Odometrie zu erhalten. Dazu wird ein Komplementärfilter verwendet, welcher die negativen Eigenschaften der partiellen Odometriebestimmung eliminiert. Mit Hilfe eines Hochpassfilters erhält man aus dem Gyroskop nur Änderungen über einen sehr kurzen Zeitraum, in welchem der Drift keinen Einfluss haben sollte. Analog dazu werden aus dem anderen Verfahren nur langfristige Änderungen mittels eines Tiefpassfilters extrahiert.

Damit die Fusion stattfinden kann, ist es erforderlich, dass beide Odometrieverfahren auf dem gleichen Koordinatensystem basieren. Dazu wird das Iterationsverfahren des Gyroskop zunächst mit einer bereits berechneten Orientierung des Beschleunigungs- beziehungsweise Magnetsensor Verfahren initialisiert.

Es zeigt sich, dass erst durch eine Fusion der Sensormesswerte eine genaue Schätzung der absoluten Orientierung realisierbar ist. Anders sieht es bei der sogenannten Koppelnavigation aus. Hier wird versucht an Hand der internen Sensorik die zurückgelegte Strecke zu schätzen. Die IMU bietet dafür den Beschleunigungssensor. Durch eine doppelte Integration der Zeit erhält man den Weg auf den einzelnen Achsen. Dieser Sensor weist jedoch ebenfalls ein gewisses Rauschen auf, dessen Fehler in einer diskreten Doppelsumme stark ansteigt. Dieses Ergebnis legt auch Herr Rudall in seiner Versuchsreihe vor[53].

3.4 Struktur des Gesamtsystems

Im Folgenden soll eine abstrakte Struktur des Gesamtsystems vorgestellt werden. Das System besteht aus redundanten Komponenten, da im Verlauf der Arbeit Algorithmen

evaluiert und bewertet werden. So besteht das Gesamtsystem anfänglich aus verschiedenen Sensoren zur Umgebungswahrnehmung. Das finale System hingegeben wird sich auf eine dieser Sensorgruppen beschränken. Auch werden die abstrakten Paketgruppen vorgestellt, deren Funktionsweise und Implementierung in den nachfolgenden Kapiteln beschrieben wird.

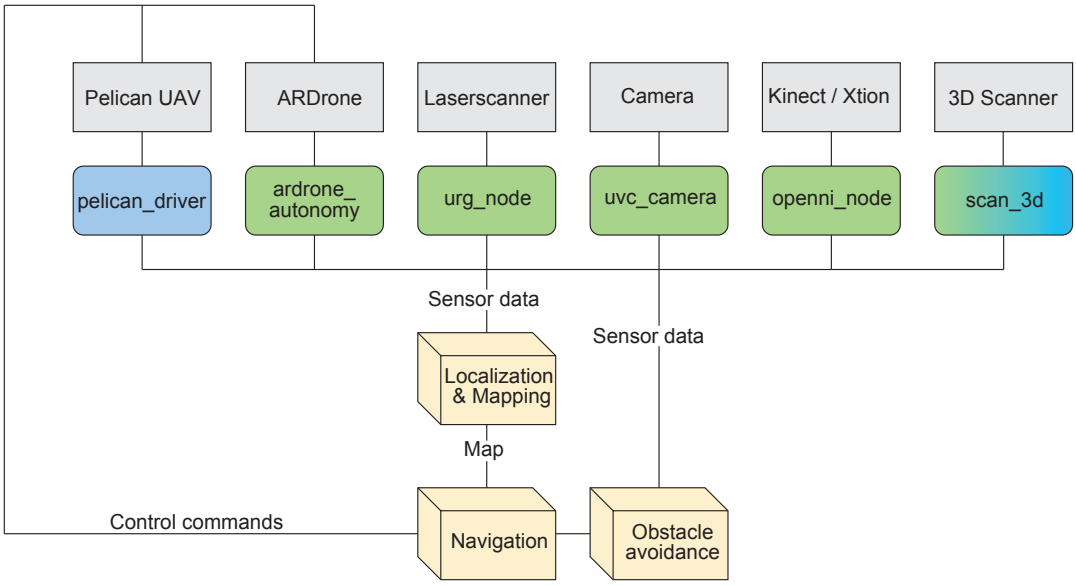

Abbildung 16: Abstrakte Struktur des gesamten Systems.

In der obersten Schicht der Struktur finden sich die Hardwarekomponenten, gefolgt von ihren ROS Treibern. Blaue Treiberknoten wurden selbst implementiert, grüne Knoten werden von der Community angeboten. In der dritten Schicht befinden sich die abstrakten Komponenten, deren Eingänge die Sensordaten sind. Zusammenfassend lässt sich sagen, dass zu jeder evaluierenden Sensorgruppe eine abstrakte Komponente konkretisiert werden muss. Es stellen sich folgende Fragen: Wie ist es möglich, aus einer Reihe von Laserscans, aus einer Reihe von Mono- oder Stereobildern oder aus einer Reihe von RGB-D Aufnahmen eine Karte zu erzeugen und dynamisch die Umgebung zu erkennen. Zuletzt beschäftigt sich diese Arbeit mit der Frage in wie weit das UAV in der Lage ist, autonom in dieser erkannten Umgebungen zu navigieren.

Es zeigt sich als sinnvoll, zunächst Algorithmen zu entwickeln und zu evaluieren, welche sich auf rein visuelle Sensorik beziehen, da diese Sensoren gerade im Mikro-/Medium UAV Sektor häufig vertreten sind. Es wird sich zeigen, dass viele Arbeitspakete auch für die Nutzung eines Laserscanners benutzt werden können.

4 Verfahren zur Wahrnehmung der Umgebung

Im folgenden Kapitel werden verschiedene Verfahren zur Erfassung der Umgebung eines UAVs vorgestellt. Wie bereits in Kapitel 3.4 erwähnt, arbeitet jede dieser Prozeduren auf den Daten einer bestimmten Sensorgruppe und kann somit nicht auf jedem UAV Typ genutzt werden. Der erste Teil dieses Abschnittes befasst sich mit der Umgebungswahrnehmung an Hand einer einzelnen Kamera. In der Fachsprache wird dies durch das Schlüsselwort *Mono Vision* ausgedrückt. Dies ist aus materieller Sicht die einfachste Methode um Strukturen der Umgebung wahrzunehmen und steht seitens der Sensorik auf nahezu jedem UAV bereits zur Verfügung. Der zweite Abschnitt befasst sich mit der *Stereo Vision*. Hier wird in Analogie zu dem menschlichen Sehvermögen versucht, die Umgebungsstrukturen aus verschiedenen Perspektiven zu triangulieren. Im letzten Abschnitt werden Sensorgruppen vorgestellt, welche eine Distanzmessung instantan liefern. Hier spielt es eine große Rolle, wie diese Informationen interpretiert und weiterverarbeitet werden um möglichst viel über das Umfeld des UAVs in Erfahrung zu bringen. Im Anschluss wird der 3D Scanner beschrieben, welcher auf einem zweidimensionalen Tiefensensor basiert und durch einfache geometrische Algorithmik komplexe dreidimensionale Umgebungsmodelle erzeugt.

4.1 Mono visuelle Odometrie und Strukturen durch Bewegung

Das Ablichten einer dreidimensionalen Umgebung erzeugt ein zweidimensionales Abbild der Szene, mit einem gravierenden Nachteil: Der Verlust der Tiefeninformation. Die Rekonstruktion dieser Informationen kann auf verschiedene Arten realisiert werden. Dieser Abschnitt befasst sich mit der Wiederherstellung der Szene auf Basis multipler Aufnahmen einer einzelnen Kamera.

Um die visuelle Odometrie oder gar die Rekonstruktion dreidimensionaler Elemente aus einer zweidimensionalen Umgebung zu beschreiben, sind zunächst grundlegende Informationen zur Kamerageometrie von Nöten. Ebenfalls ist es notwendig, die Kamera zu kalibrieren, so dass die aufgenommen Bilder frei von allen intrinsischen Verzerrungen sind. Diese Verzerrungen entstehen durch die Wölbung der Kameralinse und dem Abstand zum eigentlichen CCD - Sensor, auch Brennweite genannt.

4.1.1 Grundlagen

Zunächst soll ein grundlegendes Kameramodell, das Modell der Lochkamera, vorgestellt werden. Auf diesem Modell basieren alle internen Berechnungen die zur Projektion eines Bildpunktes und auch für die Bestimmung der Parameter von Nöten sind.

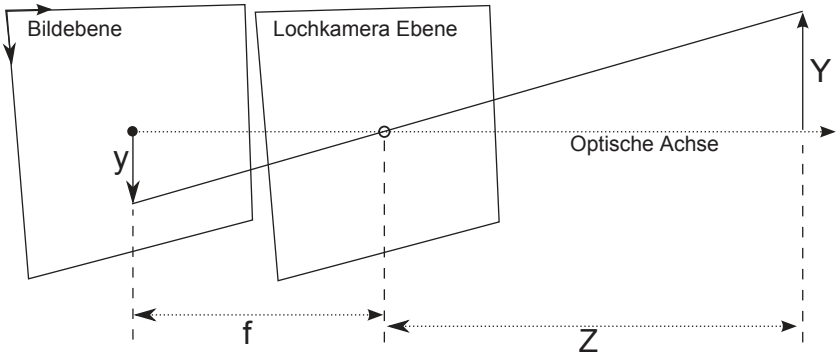

Abbildung 17: Lochkameramodell zur Bestimmung der Entfernung bei bekannter Objektgröße

Mit dem bekannten Strahlensatz aus der Geometrie lässt sich sofort folgende Beziehung herstellen.

$$\frac{y}{f} = \frac{Y}{Z} \tag{4.1}$$

Mit dieser Gleichung ist man der Lage, einen dreidimensionalen Punkt auf dem zweidimensionalen Bild zu lokalisieren. Diese Gleichung stellt ein Verhältnis zwischen der Größe eines Objekts im Bild und der Größe des Objekts in der Welt auf. Ist die Brennweite bekannt, kann die Entfernung des Objektes bestimmt werden oder anders herum, bei bekannter Entfernung die Brennweite. Voraussetzung ist jedoch, dass die Größe des Objektes bekannt ist.

Analog dazu kann man den Satz auch für die Breite eines Objektes anwenden.

$$\frac{x}{f} = \frac{X}{Z} \tag{4.2}$$

Der Vektor (x, y) repräsentiert nun einen zweidimensionalen Punkt im Bild, wohingegen der Vektor (X, Y, Z) den korrespondierenden Punkt im dreidimensionalen Raum beschreibt. Zu beachten ist jedoch, dass eine Kamera für jede Achse verschiedene Brennweiten haben kann. Es ergeben sich somit folgende Gleichungen, welche direkt nach den Bildkoordinaten umgestellt werden.

$$x = f_x \frac{X}{Z} \tag{4.3}$$

$$y = f_y \frac{Y}{Z} \tag{4.4}$$

Betrachtet man nun den Mittelpunkt des Bildes, welcher die Ursprungskoordinaten im Bild darstellt, so ist ein Offset zu addieren. Dadurch verschiebt sich der Ursprung in die Ecke des Bildes, so wie man es aus der klassischen Bildverarbeitung kennt. Auf Grund der Bauweise der Kamera kann es sein, dass der Ursprung des Bildes nicht immer genau mittig liegt.

$$x = f_x \frac{X}{Z} + c_x \tag{4.5}$$

$$y = f_y \frac{Y}{Z} + c_y \tag{4.6}$$

Um mit diesen Gleichungen schnell rechnen und auch anschließende Operationen mit komplexeren Transformationen durchführen zu können, wird eine Matrix definiert, welche die oben genannten Gleichungen repräsentiert.

$$\begin{aligned} p_{2D} &= M P_{3D} \\ \begin{pmatrix} x \\ y \\ w \end{pmatrix} &= \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{pmatrix} X \\ Y \\ Z \end{pmatrix} \end{aligned} \tag{4.7}$$

Multipliziert man nun einen Punkt mit der Matrix M erhält man den passenden Bildpunkt. Nach der Normierung, der Division durch w, erhält man die genauen x und y Koordinaten. Die Matrix ist für eine Kamera konstant, da sie ihr Modell abbildet und enthält die ersten vier intrinsischen Parameter. Zu Beachten ist jedoch, dass die inverse Operation nicht möglich ist. Es ist nicht möglich einen zweidimensionalen Punkt im Bild ohne weitere Informationen auf die 3D Umgebung zu projizieren, da die Projektion jeden beliebigen Punkt der Z-Achse treffen kann.

4.1.2 Bestimmung der intrinsischen Parameter

Mit Hilfe der grundlegenden Kamerageometrie kann eine Kamera kalibriert werden. Wie es bereits im Kameramodell dargestellt wird, bestehen die ersten vier intrinsischen Parameter aus der sogenannten Kamera Matrix. Es existieren fünf weitere Parameter, welche Auskunft über die Verzerrung der Linse geben. Ziel der Kalibrierung ist es nun, diese Parameter zu bestimmen.

Die OpenCV bietet dazu eine Funktion an, welche die Kalibrierung an Hand von bestimmten Punkten in den Bildern übernimmt.

```
1 double calibrateCamera(objectPoints, imagePoints, imageSize,
      cameraMatrix, distCoeffs, ...)
```

Quellcode 1: Kalibrierung via OpenCV

Die Funktion erwartet korrespondierende Punkte im Bild sowie in der dreidimensionalen Umgebung und bestimmt anschließend die Verzerrungskoeffizienten sowie die Kamera Matrix. In diesem Abschnitt soll die Funktionsweise des Kalibrierungsalgorithmus näher beschrieben werden. Zunächst braucht man jedoch korrespondierende Punkte in den Aufnahmen. Dazu wird ein gewöhnliches Schachmuster verwendet. Die OpenCV sucht nun, über eine interne Funktion, nach diesem besagten Schachmuster.

```
1 bool findChessboardCorners(const Mat image, ...)
```

Quellcode 2: Detektieren eines Schachbrettmusters

Die Suche geschieht über einfache Eckdetektoren und dem Auswerten der gefundenen Eckdaten mit den Daten des Schachbretts. Die genauere Funktionsweise eines Eckdetektors wird vorerst nicht näher erläutert. Hat der Algorithmus das Schachbrett gefunden, kennt er jede Position der Ecken im zweidimensionalen Bild. Das dreidimensionale Modell eines Schachbrettes lässt sich über die Anzahl der Ecken im Schachbrett erstellen. Als Beispiel für ein Schachbrett mit 4 Ecken:

$$P1 = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}, P2 = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}, P3 = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}, P4 = \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix} \tag{4.8}$$

Die dreidimensionalen Punkte des Schachbretts liegen jedoch zurzeit im lokalen Schachbrettkoordinatensystem und nicht im Kamerakoordinatensystem. Es muss somit noch eine zusätzliche Transformation erfolgen um die Koordinatensysteme in Deckung zu bringen, damit anschließend das Kamera Modell angewandt werden kann. Diese Transformation besteht aus einer Rotationsmatrix R sowie einer Translation t.

Da nun der detektierte Punkt im dreidimensionalen Kameraraum sowie im zweidimensionalen Bild bekannt ist, kann die Kalibrierung erfolgen. Dazu wird zunächst angenommen, dass die Kameralinse keine Verzerrungen hervorruft.

Mit Hilfe des Kameramodells sowie der Transformation zwischen dem Koordinatensystem lässt sich folgende Beziehung zwischen einem detektierten Punkt im Bild und dem korrespondierenden Punkt im dreidimensionalen System herstellen:

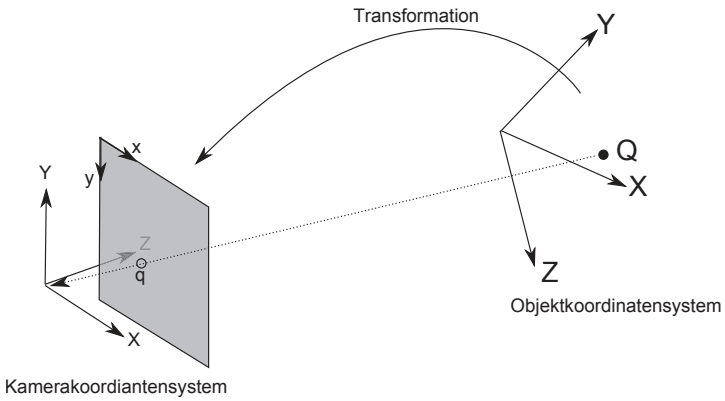

Abbildung 18: Transformation zwischen Kamera- und Schachbrettkoordinatensystem

$$q = sHQ \tag{4.9}$$

Dabei bezeichnet s eine eventuelle Skalierung. Der Parameter ist Bestandteil der Homografie H, wurde allerdings zur Erläuterung herausgezogen. Welche Bedeutung diese Skalierung hat, wird im späteren Verlauf des Kapitels deutlich werden. Die Homografiematrix besteht aus zwei Teilen: Einer physischen Transformation, welche die Position des Schachbretts im Raum wiedergibt sowie einer Projektion, welche den detektierten Punkt an Hand der Kameramatrix M auf das Bild abbildet.

$$q = sMWQ \tag{4.10}$$

$$W = \begin{bmatrix} R & t \end{bmatrix} \tag{4.11}$$

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = sM \begin{bmatrix} \overrightarrow{r_x} & \overrightarrow{r_y} & \overrightarrow{r_z} & \overrightarrow{t} \end{bmatrix} \begin{bmatrix} X \\ Y \\ 0 \\ 1 \end{bmatrix} \tag{4.12}$$

Die Z Koordinate wurde zu 0 gesetzt, da es sich bei dem Schachbrett um ein planares Objekt handelt.

Dieses Gleichungssystem kann nun mit Hilfe algebraischer Methoden gelöst werden. Doch wie viele Bilder und welche Schachbrettgröße werden dafür benötigt? Aus den Grundlagen der Mathematik ist bekannt, dass zur Lösung von m Unbekannten mindestens n Gleichungen notwendig sind. Betrachtet man zunächst die Anzahl der unbekannten Variablen, findet sich in jedem Kamerabild eine Transformation W, welche Auskunft über die Lage des Schachbretts gibt. Weiterhin gilt für jedes Bild die konstante Kameramatrix

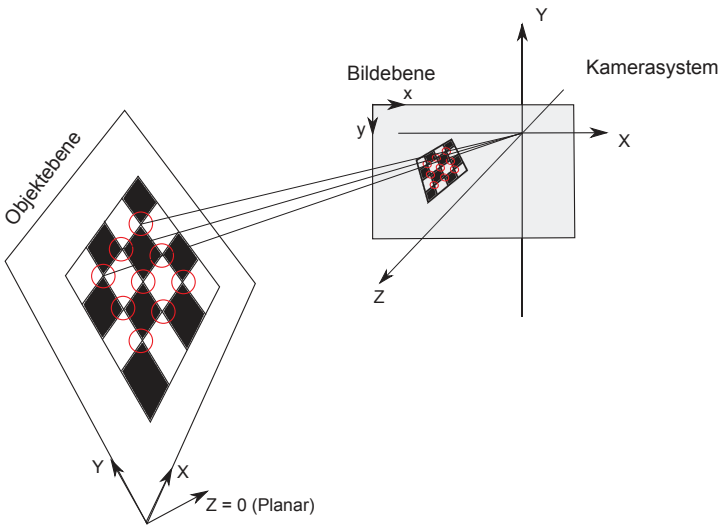

Abbildung 19: Vollständige Projektion eines Punktes aus dem Schachbrett in das zweidimensionale Kamerabild

M. Die Matrix M beinhaltet die vier intrinsischen Parameter. Die Matrix W verfügt über drei Rotationswinkel sowie den drei Parametern der Translation. Daraus ergeben sich bei k Bildern $6 * k + 4$ Unbekannte. Für jedes Kamerabild können die detektierten Punkte mit Hilfe der oben genannten Gleichung bestimmt werden. Das sind bei einem $m \times n$ Schachbrett und einer Bildreihe von k Bildern $2 * m * n * k$ Gleichungen, da wir für einen Bildpunkt x sowie y Koordinaten haben. Für die Anzahl von Bilder und die Größe des Schachbretts gilt folglich:

$$2mnk \geq 6k + 4 \tag{4.13}$$

Je mehr Gleichungen man erhält, desto besser können die Ergebnisse geglättet und verifiziert werden. Zur Lösung dieser Gleichungssysteme nutzt die OpenCV den Algorithmus von Zhang[63].

Nachdem die Kameramatrix und damit die ersten vier intrinsischen Parameter bestimmt wurden, werden nun die restlichen Parameter bestimmt. Diese Parameter geben Auskunft über die Verzerrung, welche durch die verwendete Linse in der Kamera hervorgerufen werden. Diese Verzerrungen können neben den radialen Verzerrungen, welche durch die Form der Linse auftreten, zusätzlich auch aus tangentialen Verzerrungen bestehen. Tangentiale Verzerrungen entstehen, wenn physikalische Elemente in der Linse nicht 100% ausgerichtet sind. Um diese Verzerrungen zu berechnen verwendet die OpenCV das Verzerrungsmodell von Brown[10].

$$x_{undistorted} = x(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) + [2p_1 y + p_2(r^2 + 2x^2)] \tag{4.14}$$

$$y_{undistorted} = y(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) + [p_1(r^2 + 2y^2) + 2p_2 x] \tag{4.15}$$

Mit Hilfe des Kameramodells und der Koordinatentransformationen kann das Schachbrett nun vollständig in optimale Bildkoordinaten umgerechnet werden. Diese Bildkoordinaten sind dann frei von Verzerrungen und werden mit den original detektierten Bilddaten verrechnet um so die Verzerrungskoeffizienten zu bestimmen.

Auch hier erhält man $2 * m * n * k$ Gleichungen, diesmal jedoch nur fünf konstante Unbekannte, so dass keine zusätzlichen Anforderungen an die Anzahl der Bilder entstehen. Die OpenCV löst diese Gleichungssysteme nun nach einem Algorithmus von Brown[10] auf.

Nachdem die Verzerrungskoeffizienten bestimmt wurden, werden die Eingangsbilder entzerrt und die ersten intrinsischen Parameter abermals bestimmt, um diese nachträglich zu optimieren. Anschließend werden die berechneten Parameter in YAML Konfigurationsdateien abgelegt. Diese Parameter sind für die Kamera konstant und können wiederverwendet werden, ohne die Kamera neu kalibrieren zu müssen.

4.1.3 Die Epipolargeometrie

Ist die interne Kamerageometrie bekannt, kann das Ablichten der Szene ebenfalls mathematisch beschrieben werden. Dazu wird in der Bildverarbeitung die Epipolargeometrie verwendet. Die Epipolargeometrie dient zur Unterstützung der Lokalisierung von Punkten in einer sequentiell abgelichteten Szene. Kurz gesagt: Ist ein Punkt im ersten Bild gegeben, schränkt sich der Suchbereich des korrespondierenden Punktes im zweiten Bild auf die sogenannte Epipolarlinie ein. Voraussetzung dafür ist jedoch, dass die Epipolargeometrie bekannt ist.

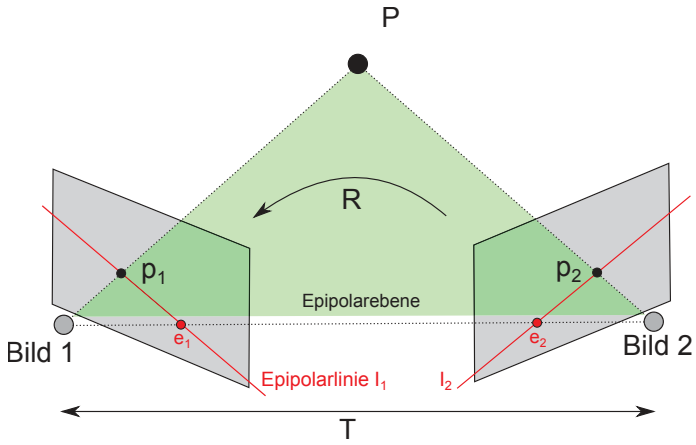

Abbildung 20: Darstellung der Epipolargeometrie bei multiplen Kameraaufnahmen

Der Aufbau der Epipolargeometrie folgt einer festen Terminologie. Die aufgespannte Ebene zwischen den Kamerapositionen und einem abgelichteten Objekt wird als Epipolarebene bezeichnet. Der Schnittpunkt zwischen diesen Ebenen und den projizierten Bildebenen nennt man den Epipol e. Die Epipolarlinie I verbindet nun den Epipol mit den projizierten Objektpunkten in den jeweiligen Bildern. Folglich kann der Epipol auch als Schnittpunkt aller Epipolarlinien eines Bildes bezeichnet werden.

Die gesamte Epipolargeometrie wird durch die sogenannte Fundamentalmatrix beschrieben. Mit Hilfe dieser Matrix wird zu einem beliebigen Bildpunkt im ersten Bild die zugehörige Epipolarlinie im zweiten Bild und umgekehrt bestimmt.

$$I_2 = Fp_1 \tag{4.16}$$

$$I_1 = F^T p_2 \tag{4.17}$$

Hartley[24] zeigt, dass die Fundamentalmarix eng mit der bereits erwähnten Homografiematrix H verbunden ist, welche die Eigenschaft besitzt einen Punkt p_1 direkt in einen korrespondierenden Punkt p_2 zu überführen.

$$p_2 = Hp_1 \tag{4.18}$$

Es ist bekannt, dass die Epipolarlinie durch den Bildpunkt selbst und den Epipol in der Bildebene gebildet wird. Verglichen mit Gleichung 4.16 wird der Zusammenhang zwischen F und H sofort ersichtlich.

$$\begin{aligned} I_2 = e_2 \times p_2 = e_2 \times Hp_1 = Fp_1 \\ \Rightarrow F = e_2 \times H \end{aligned} \tag{4.19}$$

Eine wichtige Grundeigenschaft der Fundamentalmatrix ist gegeben, wenn ein korrespondierendes Punktepaar (p_1, p_2) bekannt ist, denn wenn die Punkte korrespondieren liegt p_2 auf der Epipolarlinie $I_2 = Fp_1$. Anders ausgedrückt bedeutet das: Der Abstand von p_2 zur Epipolarlinie I_2 ist 0, die Vektoren sind koplanar. Diese Eigenschaft wird als Epipolargleichung bezeichnet.

$$0 = p_2^T I^2 = p_2^T F p_1 \tag{4.20}$$

Sind eine Reihe von korrespondierenden Punkten in den Bildern bekannt, kann die Fundamentalmatrix, auf Grund der Epipolargleichung 4.20, geschätzt werden. Dazu wird diese zunächst ausmultipliziert und in die vektorielle Schreibweise überführt.

$$\begin{aligned} x_2x_1f_{11} + x_2y_1f_{12} + x_2f_{13} + y_2x_1f_{21} + y_2y_1f_{22} + y_2f_{23} + x_1f_{31} + y_1f_{32} + f_{33} = 0 \\ \Rightarrow \begin{pmatrix} x_2x_1 & x_2y_1 & x_2 & y_2x_1 & y_2y_1 & y_2 & x_1 & y_1 & 1 \end{pmatrix} \cdot \mathbf{f} = 0 \end{aligned} \tag{4.21}$$

mit der vektoriellen Darstellung der Fundamentalmatrix:

$$f = \begin{pmatrix} f_{11} \\ f_{12} \\ f_{13} \\ f_{21} \\ f_{22} \\ f_{23} \\ f_{31} \\ f_{32} \\ f_{33} \end{pmatrix}$$

Im Allgemeinen sind n Punktkorrespondenzen bekannt wodurch der Vektor in eine Matrix A übergeht.

$$\begin{bmatrix} x_2^1x_1^1 & x_2^1y_1^1 & x_2^1 & y_2^1x_1^1 & y_2^1y_1^1 & y_2^1 & x_1^1 & y_1^1 & 1 \\ x_2^2x_1^2 & x_2^2y_1^2 & x_2^2 & y_2^2x_1^2 & y_2^2y_1^2 & y_2^2 & x_1^2 & y_1^2 & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ x_2^nx_1^n & x_2^ny_1^n & x_2^n & y_2^nx_1^n & y_2^ny_1^n & y_2^n & x_1^n & y_1^n & 1 \end{bmatrix} = A \tag{4.22}$$

Man erhält folgendes lineares Gleichungssystem welches, ab einer gewissen Menge von Punktkorrespondenzen, gelöst werden kann.

$$A \cdot f = 0 \tag{4.23}$$

Die Fundamentalmatrix besteht aus neun Elementen, wodurch auch neun korrespondierende Punktepaare benötigt werden um das Gleichungssystem vollständig lösen zu können. Wie bereits erwähnt, werden jedoch n Punktepaare detektiert, so dass zur optimierten Lösung auf den RANSAC Algorithmus[17][35] zurückgegriffen wird. RANSAC, kurz für Random Sample Consenus (deut. Übereinstimmung zufälliger Stichproben), ist ein iterativer Algorithmus, mit dessen Hilfe ein mathematisches Modell durch einen gegebenen Datensatz approximiert wird. Ferner vergleicht der Algorithmus den gegeben Datensatz mit der aktuell approximierten Modellfunktion und ist so in der Lage, Ausreißer zu detektieren, welche das Modell verfälschen würden.

Quellcode 3 zeigt den Ablauf des RANSAC Algorithmus und lässt sich in fünf Schritten beschreiben.

```
1  while iterations < K:
2      maybe_inliers = pick_random(data) #1
3      maybe_model = estimate_model(maybe_inliers) #2
4
5      for each point in data:
6          if calculate_error(point, maybe_model) < max_error:
7              consensus_set.add(point) #3
8
9      if consensus_set.size() > d: #4 Good model, but best?
10         this_model = estimate_model(consensus_set)
11         this_error = calculate_error_all(data, this_model)
12         if this_error < best_error
13             best_model = this_model
14             best_consensus_set = consensus_set
15             best_error = this_error
16     iterations++ #5
```

Quellcode 3: RANSAC Algorithmus zur Bestimmung einer Modellfunktion

1. Wähle eine zufällige Menge an Punkten aus dem Datensatz, die zur Bestimmung des Modells erforderlich sind. In diesem Fall betrifft das neun korrespondierende Punktepaare um das Gleichungssystem 4.23 für die Fundamentalmatrix vollständig zu lösen.

2. Berechne das Modell und bestimme anschließend für den gesamten Datensatz einen Abstand zur Modellfunktion. Das bedeutet, dass nach dem Lösen des Gleichungssystems jedes Korrespondenzpaar nach der Gleichung 4.20 auf Gültigkeit geprüft wird. Weicht das Ergebnis von 0 um mehr als einen maximalen Fehler ab, ist das Modell für dieses Punktepaar schlecht bestimmt und der Abstand zur Modellfunktion nimmt zu.

3. Bestimme eine Teilmenge der Datenwerte, deren Abstand zur Modellfunktion unter dem definierten Maximum liegt. Diese Teilmenge wird als *Consensus Set* bezeichnet und mit dem aktuellen Modell abgespeichert.

4. Hat das *Consensus Set* genügend Punkte, auch *Inlier* genannt, um das Modell zu approximieren, wird die Modellfunktion abermals berechnet und der gesamte resultierende Fehler mit den bisherigen Sets verglichen. Ist der Fehler geringer, so weist dieses Set für ihr bestimmtes Modell die meisten Punktepaare in der Nähe der Modellfunktion auf und approximiert F am besten.

5. Wiederhole Schritte 1 - 4 bis die maximale Anzahl an Iterationen erreicht ist.

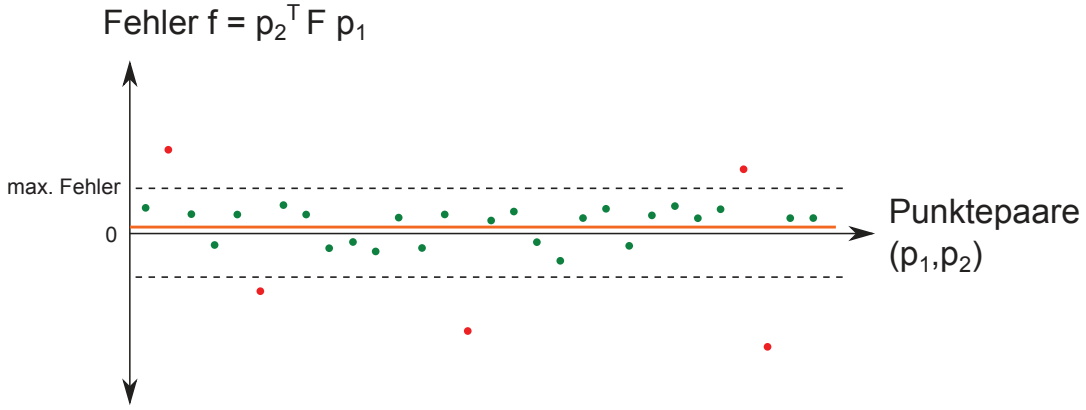

Abbildung 21: Abbildung der Modellfunktion nach einer RANSAC Iteration

Abbildung 21 zeigt beispielhaft das Ergebnis einer RANSAC Iteration mit 30 Punktkorrespondenzen. Mit Hilfe von neun Zufälligen dieser 30 Punktepaare wurde eine Fundamentalmatrix berechnet. Durch Anwenden der Epipolargleichung auf jedes dieser Punktepaare sowie dem Vergleich mit einem maximal definierten Fehler können diese evaluiert und Ausreißer detektiert werden. In diesem Beispiel wurden fünf von 30 Punkten als Fehler bestimmt, 25 Punktepaare passen zu der gewählten Approximation. Der Mittelwert der *Inlier* definiert die hier eingezeichnete Modellfunktion in orange.

Zum Finden der Fundamentalmatrix, durch Anwendung des RANSAC Algorithmus, bietet die OpenCV eine einfache Funktion an.

```
1 Mat findFundamentalMat(InputArray points1, InputArray
    points2, int method=FM_RANSAC, double param1=, double
    param2)
```

Quellcode 4: OpenCV Funktion zur Bestimmung der Fundamentalmatrix

Neben dem RANSAC Algorithmus können hier auch weitere Implementierungen wie der 7- oder 8-Punkt Algorithmus[58] gewählt werden, welche in dieser Arbeit jedoch nicht verwendet wurden und somit auch nicht näher vorgestellt werden.

4.1.4 Detektieren korrespondierender Punkte

Der vorherige Abschnitt zeigte die Bestimmung der Fundamentalmatrix auf Grund gegebener Punktkorrespondenzen. Diese Korrespondenzen sind jedoch in zwei aufeinanderfolgenden Bildaufnahmen einer beliebigen Umgebung, zunächst nicht gegeben. Es werden nun Verfahren vorgestellt, welche es ermöglichen, eine Reihe von passenden Punkten in

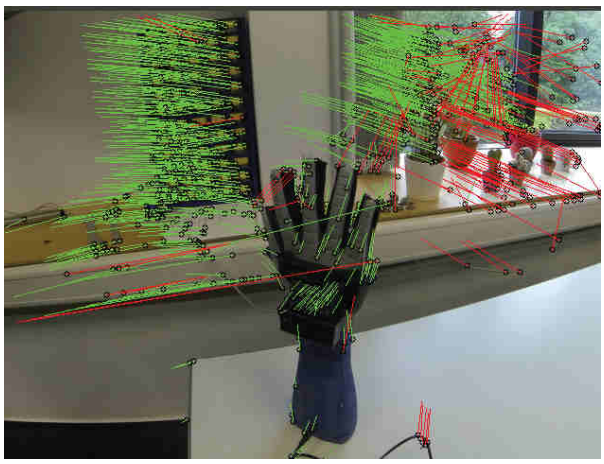

Abbildung 22: Darstellung des besten *Consesus Sets*. Das Bild zeigt Korrespondenzpunkte sowie deren Herkunftsrichtung aus einer vorherigen Aufnahme. Grüne Pfade ließen sich vollständig rekonstruieren, rote Pfade passen nicht zu der finalen Fundamentalmatrix und wurden als Ausreißer markiert.

Teilaufnahmen zu detektieren. Die OpenCV bietet dazu verschiedene Algorithmen an, so dass im Vorfeld bereits eine Entscheidung darüber getroffen werden musste, welche Detektoren am Besten geeignet sind.

Der de facto Standard in der Bildverarbeitung sind die SIFT[37] und SURF[7] Detektoren. Der SIFT-Algorithmus, kurz für *Scale Invariant Feature Transform* besteht aus zwei Komponenten: Einem Keypoint Detektor und einem Deskriptor. Das Eingangsbild wird zunächst in den gaußschen Skalenraum überführt. Das heißt, dass das Bild in verschiedene Stufen herunter skaliert und aus jeder Stufe eine Menge von gaußgeglätteten Bildern erzeugt wird. Dabei haben die Bilder verschiedene Glättungen. Im Anschluss werden von jeweils zwei Bildern einer Stufe gaußsche Differenzbilder erzeugt. Mit Hilfe dieser Differenzbilder ist es nun möglich lokale Extrema und damit interessante Keypoints zu detektieren. Auf Grund der Bildung dieser Pyramidenstruktur ist das Verfahren sehr rechenintensiv.

Aus den detektierten Minima und Maxima werden nun über eine Suche in deren Nachbarschaft die Gradienten bestimmt. Dazu wird eine 16×16 Umgebung des Pixels betrachtet. Diese Umgebung wird in weitere 4×4 Subregionen aufgeteilt. Anschließend werden die Gradientenvektoren jeder Subregion in einem Histogramm aufgetragen, welches in 45 Grad Schritten dimensioniert ist. Die Hauptausrichtung der Gradienten gibt die Orientierung eines Pixel an, wodurch das Verfahren robust gegen Rotationen ist. Der Deskriptor besteht final aus einem Vektor mit 128 Elementen und beschreibt seinen Bezugspunkt. Mit Hilfe dieser Deskriptoren können Keypoints verglichen und eine eventuelle Korrespondenz festgestellt werden. Jedoch ist auch der Vergleich mit erhöhtem Rechenaufwand verbunden, da eine Reihe von Vektoren mit der zugehörigen euklidischen

Distanz verglichen werden muss.

Der SURF-Algorithmus, kurz für *Speeded Up Robust Features* basiert auf SIFT, ersetzt jedoch die in SIFT verwendeten Gauß-Filter durch einfache Mittelwertfilter, welche die Verwendung von Integralbildern ermöglichen und somit die Berechnungsdauer minimieren.

SIFT und SURF liefern reproduzierbare Deskriptoren und sind dabei rotations- und skalierungsinvariant. Es ist jedoch bekannt, dass beide Verfahren auf einfachen Prozessorstrukturen nicht echtzeitfähig sind. Es ist ebenfalls bekannt, dass durch gaußsche Verzerrungen Rauschen reduziert und markante Objekte hervorgehoben werden, jedoch natürliche Objektkanten verwischen, was wiederum zu Ungenauigkeiten der Lokalität führt. Für die Verwendung in dieser Arbeit scheiden SIFT und SURF ohnehin aus, da deren Implementierung zwar quelloffen, die Verwendung allerdings, auf Grund von Patentbeschränkungen, nur für private, nicht kommerzielle Zwecke bestimmt ist.

Pablo Alcantarilla stellt in seiner Arbeit[2] eine interessante Alternative vor. Die AKAZE Features. AKAZE, kurz für Accelerated KAZE, ist eine optimierte Version der zuvor entwickelten KAZE Features[1]. Das Wort KAZE stammt aus dem Japanischen und bedeutet Wind. In der Natur bezeichnet der Wind einen groß skalierten Luftfluss, welcher durch nicht lineare Prozesse definiert wird. Im Vergleich zum SIFT-Algorithmus, welcher auf einer linearen gaußschen Weichzeichnung basiert und somit auf allen Skalen eine gleichmäßige Verzerrung stattfindet, arbeitet KAZE auf einer lokalen nichtlinearen Diffusion. Lokales Rauschen wird geglättet jedoch bleiben natürliche Objektkanten erhalten, was sich positiv auf deren Lokalität auswirkt.

Für die nichtlineare Diffusion gilt die folgende Gleichung.

$$\frac{\delta L}{\delta t} = div(c(x, y, t) \bigtriangledown L) \tag{4.24}$$

Die Operatoren *div* und $\bigtriangledown$ stehen für die Divergenz und die Gradienten. L steht für die Bildintensität. Für den diskreten Anwendungsfall stellt Alcantarilla folge Approximation namens *Fast Explicit Diffusion*, kurz FED, vor.

$$\frac{L^{i+1} - L^i}{\tau} = A(L^i)L^i \tag{4.25}$$

L repräsentiert nun für die Faltungsebenen des Eingangsbildes. Die Matrix $A(L^i)$ beinhaltet die nichtlinearen Konduktivitäten eines Bildes. τ ist ein konstanter Zeitschritt. Für die Bildung der Matrix $A(L^i)$ sei auf die Arbeit von Alcantarilla[2] verwiesen. Wichtig für das Allgemeinverständnis ist, dass die Lösung L^{i+1} direkt aus der vorherigen Faltung L^i und der Konduktivitätsmatrix $A(L^i)$ gebildet wird. Um nun den gesamten Skalenraum zu berechnen, werden wie auch schon beim SIFT Algorithmus Oktaven O und Sub-Level S erzeugt.

$$\sigma_i(o,s) = 2^{o+s/S}, o \in [0 \dots O-1], s \in [0 \dots S-1], i \in [0 \dots M] \tag{4.26}$$

Dabei wird ein Zeitschritt des Filters auf den jeweiligen Index eines Oktaven-Sub-Level Paares abgebildet, wobei M für die gesamte Anzahl gefilterte Bilder steht.

$$t_i = \frac{1}{2}\sigma_i^2, i \in [0 \dots M] \tag{4.27}$$

Mit steigender Zeit werden somit weitere Skalierungen mit steigenden Filterstufen erzeugt. Abbildung 23 zeigt den Unterschied zwischen einer linearen Diffusion (SIFT) und der nichtlinearen Diffusion (AKAZE). Es zeigt sich deutlich, dass bei der linearen Diffusion das Bild gleichmäßig verwischt wird, wohingegen bei der nichtlinearen Diffusion die Bildtexturen zwar weicher werden, die Objektkanten jedoch deutlich erhalten bleiben.

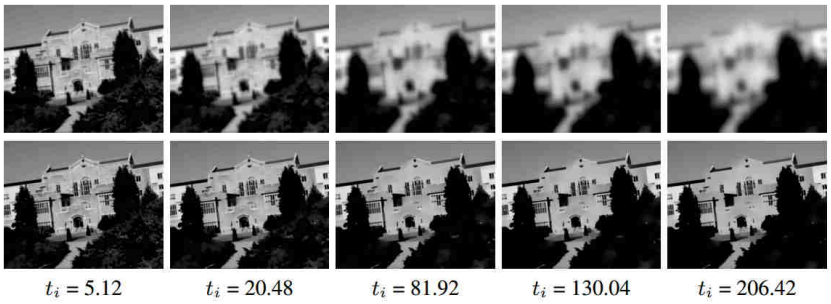

Abbildung 23: Vergleich zwischen linearer Diffusion (oben) und nichtlinearer Diffusion (unten) aus Alcantarilla[1]

Das Finden der eigentlichen Features basiert auch hier über das Bestimmen der lokalen Extrema im Skalenraum. Ein jedoch entscheidender Vorteil findet sich bei der Deskription der Features. Wohingegen der KAZE Algorithmus eine angepasste Implementierung der SURF Deskriptoren nutzt, werden beim AKAZE Verfahren binäre Deskriptoren gebildet. Binärvektoren benötigen wenig Speicher und lassen sich effizient vergleichen. Alcantarilla stellt dazu den *Modified Local Difference Binary*, kurz M-LDB, Algorithmus vor. Im Wesentlichen werden die Intensitäts- und Gradientenunterschiede eines Featurepunktes im nichtlinearen Skalenraum bestimmt und binär kodiert.

Alcantarilla stellt ebenfalls Experimentierergebnisse vor, welche die Erkennungsrate und die Performanz der AKAZE Features im Vergleich zu SIFT, SURF und weiteren gängigen Featuredetektoren zeigen.

Der Vergleich zeigt, dass die AKAZE Features bei zunehmender Diffusion die höchste Wiedererkennungsrate besitzen. Ferner ist die Berechnungsdauer signifikant kleiner als bei der SURF Implementierung. Dennoch ist das Bilden des Skalenraums mit einer gewissen Rechenzeit verbunden, sodass die Echtzeitfähigkeit je nach Rechenleistung und

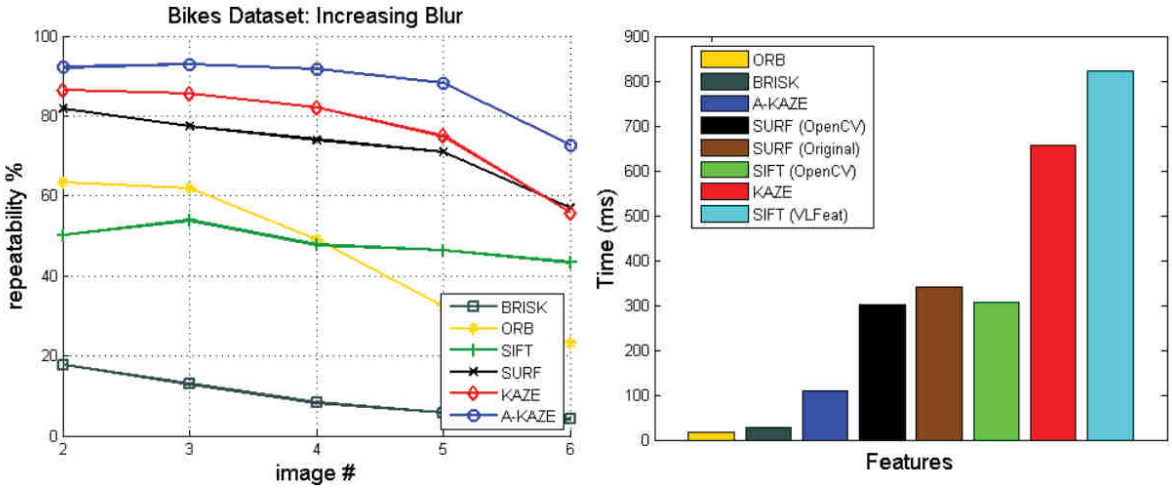

Abbildung 24: Vergleich der Feature Detektoren hinsichtlich Performanz und Erkennungsrate aus Alcantarilla[2]

Kameraauflösung zunächst analysiert werden muss. Lediglich der ORB[52]- und der BRISK[34] Algorithmus weisen über minimale Ausführungszeiten auf, zu Lasten der Erkennungsrate.

ORB und BRISK sollen in dieser Arbeit nicht näher vorgestellt werden, beide Algorithmen arbeiten jedoch auf Basis eines sehr schnellen Feature Detektors: Die *FastFeatures* nach Rosten u.a. [51]. Das Verfahren, auch als Eckdetektor bekannt, arbeitet lediglich auf den Intensitäten I einzelner Pixel und hat somit stets einen linearen Aufwand. Die Features werden über markante Kantenübergange gefunden. Dabei wird für jedes Pixel ein Intensitätsvergleich mit einer Nachbarschaft aus 16 Punkten durchgeführt.

$$S_{p\to x} = \left\{ \begin{array}{cc} I_{p\to x} < I_p - t & ,\text{dunkler} \\ I_{p\to x} > I_p + t & ,\text{heller} \\ I_p - t \leq I_{p\to x} \geq I_p + t & ,\text{gleich} \end{array} \right\}, x \in \{1 \dots 16\} \tag{4.28}$$

Sind eine gewisse Anzahl n aus $S_{p\to x}$ insgesamt heller oder dunkler als die Intensität des Ankerpunktes selbst, so handelt es sich um einen Eckpunkt, oder in diesem Kontext um ein Feature. Rosten u.a. zeigen zudem, dass die Anzahl gleicher Intensitäten von mindestens 12 , die besten Ergebnisse liefert. Der Schwellwert t für die Intensitäten muss so gewählt werden, das Belichtungsunterschiede der Umgebung, die Kantenerkennung nicht beeinflussen. Für kontinuierliche Kamerabilder, ist der Unterschied in den Belichtungen multipler Aufnahmen einer Szene jedoch minimal.

Der Vorteil ist durch das lineare Durchlaufverhalten offenkundig, ein wesentlicher Nachteil dieses Verfahren liegt jedoch in der Beschreibung der Features. Man erhält lediglich

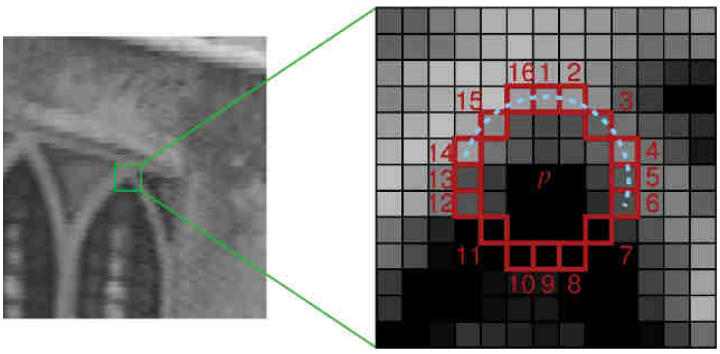

Abbildung 25: Ankerpunkt und zugehörige Intensitätsnachbarschaft zur Veranschaulichung des 12-Punkt Eckdetektors [51]

die Information, welche Pixel in einem Bild dominant sind. Ein Deskriptor zu diesem Pixel, welcher einen direkten Vergleich ermöglicht, wird nicht bestimmt.

Um dennoch eine Korrespondenz zwischen den detektierten Features zu bestimmen, wird auf den optischen Fluss zurückgegriffen. Unter der Annahme, dass die Aufnahmen der Szene unmittelbar aufeinander erfolgen, kann der optische Fluss auf die detektierten Punkte im ersten Bild angewandt werden. Das Ergebnis ist eine Schätzung der Position in der darauf folgenden Aufnahme, in der abermals detektierte Features zur Verfügung stehen. Durch den Vergleich zwischen der geschätzten Position der Features aus dem ersten Bild und den neuen detektierten Features, kann eine Korrespondenz bestimmt werden.

Zunächst soll der optische Fluss etwas näher erläutert werden. Angenommen ein Pixel $I(x, y, t)$ in der ersten Aufnahme bewegt sich um die Distanz (dx, dy) in der Zeit dt so gilt für die Position im zweiten Bild:

$$I(x, y, t) = I(x + dx, y + dy, t + dt) \tag{4.29}$$

Es wird angenommen, dass der eigentliche Farbwert sowie die Intensität des Pixels auf Grund des kleinen Zeitintervalls erhalten bleiben. Durch eine approximierte Taylorreihe und der Division durch die Zeit erhält man die Gleichung des optischen Flusses:

$$f_x u + f_y v + f_t = 0 \tag{4.30}$$

Mit:

$$
\begin{aligned}
f_x = \frac{\delta f}{\delta x}; fy = \frac{\delta f}{\delta y} \\
u = \frac{\delta x}{\delta t}; v = \frac{\delta y}{\delta t}
\end{aligned}
\tag{4.31}
$$

Nach der Zeit t erhält man die neuen Koordinaten u und v des Pixels (x, y). Die Variablen f_x und f_y sind die Gradienten des Ursprungspixels und f_t definiert den Gradienten entlang der Zeit. Dennoch sind u und v unbekannt, sodass die Gleichung nicht eindeutig gelöst werden kann. Abhilfe schafft die Lukas-Kanade Methode[59]. Unter einer weiteren Annahme, dass benachbarte Pixel die gleiche Bewegung haben, gilt für den optischen Fluss eines Pixels in einer 3×3 Umgebung folgende Gleichung.

$$
\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} \Sigma_i {f_{x_i}}^2 & \Sigma_i f_{x_i} f_{y_i} \\ \Sigma_i f_{x_i} f_{y_i} & \Sigma_i {f_{y_i}}^2 \end{bmatrix}^{-1} \begin{bmatrix} -\Sigma_i f_{x_i} f_{t_i} \\ -\Sigma_i f_{y_i} f_{t_i} \end{bmatrix}
\tag{4.32}
$$

Man erhält für jeden der Nachbarn (f_x, f_y, f_t), so dass zu der Bestimmung von u und v neun Gleichungen definiert sind. Diese Überbestimmung wird mit Hilfe der Methode der kleinsten Quadrate gelöst. Die OpenCV bietet zur Berechnung des optischen Flusses eine Funktion an:

```
1 void calcOpticalFlowPyrLK(InputArray prevImg, InputArray
    nextImg, InputArray prevPts, InputOutputArray nextPts,
    OutputArray status, ...);
```

Quellcode 5: Berechnung des optischen Flusses durch die OpenCV

Die Bestimmung der Korrespondenz erfolgt über den Vergleich der geflossenen Features aus dem ersten Bild und den detektierten Features im zweiten Bild. Dazu werden für jeden Pixel aus dem optischen Fluss die detektierten Nachbarn, innerhalb eines definierten Radius, im zweiten Bild bestimmt. Die Größe des Radius erfolgt variabel, sollte jedoch stets gering gewählt werden, da eine Punktekorrespondenz im Idealfall keinen Abstand hat. Nun kann über eine Fallunterscheidung bestimmt werden, ob zwischen dem geflossenen Feature aus dem ersten Bild sowie dem Feature im darauffolgenden Bild eine Korrespondenz besteht.

- Fall 1: Es wurde nur ein Nachbar gefunden. Die Korrespondenz zwischen den Punkten ist direkt gegeben.
- Fall 2: Es wurden mehrere Nachbarn gefunden. Es wird geprüft, ob die Nachbarn nicht zu dicht beieinander liegen. Ist dies der Fall, so wird der nächstliegendste

Nachbar als Korrespondent gewählt. Ist dies nicht der Fall, wird der Punkt verworfen, da die Korrespondenz nicht eindeutig bestimmt werden kann.

- Fall 3: Es wurden keine Nachbarn gefunden. Zu dem geflossenen Feature kann kein neues Feature zugeordnet werden.

Es wurden nun zwei Verfahren vorgestellt, welche zur Bestimmung der Punktekorrespondenz verwendet werden können. Über die Korrespondenzen kann die Fundamentalmatrix geschätzt und die visuelle Odometrie berechnet werden.

4.1.5 Berechnung der visuellen Odometrie

Die Berechnung der Odometrie einer Kamera ist eine Grundvoraussetzung für die Triangulation der dreidimensionalen Umgebung. Wie bereits in Abbildung 20 angedeutet besteht diese Odometrie aus einer Rotation und einer Translation zwischen den Aufnahmen einer Kamera. Um diese Komponenten nun zu bestimmen wird eine weitere Eigenschaft der Fundamentalmatrix ausgenutzt. Wird die Matrix um die intrinsischen Parameter der Kamera erweitert, entsteht die essentielle Matrix E. Konkretisiert bedeutet das, dass durch die Anwendung der Kameramatrix die Bildpunkte normiert werden. Auf Grund dieser Normierung liegt das Zentrum der Kamera nun in der Bildmitte, weshalb das Abbilden der Epipolargeometrie durch die Fundamentalmatrix in eine Rotation und eine Translation übergeht. Die essentielle Matrix beschreibt somit die Transformation vom ersten Kamerasystem in das Zweite, was einer klassischen Bewegung entspricht.

$$E = EK^T FK = R[t]_x \tag{4.33}$$

$R[t]_x$ ist eine vereinfachte Schreibweise für die Matrixdarstellung des Kreuzproduktes von R und t. Ferner lässt sich E mittels Singulärwertzerlegung [60] in drei Komponenten zerlegen.

$$E = U\Sigma V^T, \Sigma = \begin{bmatrix} s & 0 & 0 \\ 0 & s & 0 \\ 0 & 0 & 0 \end{bmatrix} \tag{4.34}$$

U und V sind orthogonale Matrizen, welche auch komplexe Zahlen beinhalten. Σ ist eine Diagonalmatrix, welche die Singulärwerte von E enthält. Alle Matrizen haben die Dimension 3×3. Die Singulärwerte einer essentiellen Matrix weisen eine besondere Eigenschaft auf. Zwei dieser Werte sind stets gleich, der dritte Singulärwert ist 0. Die Zerlegung kann mit Hilfe der OpenCV direkt durchgeführt werden.

```
1 void SVD::compute(InputArray src, OutputArray w, OutputArray
      u, OutputArray vt, int flags=0 )
```

Quellcode 6: Singulärwertzerlegung mit der OpenCV

Hartley [24] zeigt in seinem Buch wie mit einer Hilfsmatrix W aus den Komponenten der essentiellen Matrix die Rotation und Translation extrahiert werden können.

$$W = \begin{bmatrix} 0 & -1 & 0 \\ 1 & s & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{4.35}$$

$$R = UWV^T \tag{4.36}$$

$$t = (u_{02}, u_{12}, u_{22})^T \tag{4.37}$$

Mit Hilfe der bestimmen Rotation und der Translation kann nun die Projektionsmatrix P gebildet werden, welche einen dreidimensionalen Punkt X in die normierten Bildkoordinaten x des Kameraprojektionszentrums überführt.

$$P = (R|t) \tag{4.38}$$

$$x = PX \tag{4.39}$$

Die Odometrie der Kamera kann nun über eine Koppelnavigation der Rotationsmatrix und dem Translationsvektor geschätzt werden. Jedoch ist eine Lokalisierung exklusive an Hand der Eigenbewegung nicht zu empfehlen, da durch geringes Bildrauschen sowie den daraus resultierenden Bewegungen, in der Summierung gravierende Fehler entstehen.

4.1.6 Triangulation der dreidimensionalen Umgebung

Auf Grund der soeben bestimmen Projektionsmatrix, siehe Gleichung 4.39, kann die dreidimensionale Umgebung an Hand einfacher Triangulationen bestimmt werden.

Betrachtet man einen dreidimensionalen Punkt X von zwei Positionen, so entstehen auch zwei projizierte Bildpunkte x_1 und x_2, mit den zugehörigen Projektionsmatrizen P_1 und

P_2. Die Koordinaten des dreidimensionalen Punkts sollen dabei im Koordinatensystem der ersten Kameraposition angegeben werden, so dass für die Projektion P_1 folgende Matrix gilt.

$$P_1 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \tag{4.40}$$

Es findet keine Transformation des Punktes X in das Projektionszentrum der ersten Kamera statt. Für die zweite Kameraposition gilt die Gleichung aus dem vorherigen Abschnitt, da hier die Projektionszentren aufeinander abgebildet werden müssen.

$$P_2 = \begin{bmatrix} r_{00} & r_{01} & r_{02} & t_0 \\ r_{10} & r_{11} & r_{12} & t_1 \\ r_{20} & r_{21} & r_{22} & t_2 \end{bmatrix} \tag{4.41}$$

Nach Anwendung der Projektionsgleichung 4.39 auf den ersten Korrespondenzpunkt erhält man folgende Gleichung. Zur Erinnerung sei nochmals erwähnt, dass es sich bei den Bildpunkten x_1 und x_2 um homogene Bildpunkte handelt. Die Anwendung der Kameramatrix hat bereits stattgefunden, so dass das Projektionszentrum im jeweiligen Kamerakoordinatensystem liegt.

$$x_1 = P_1 X \tag{4.42}$$

$$\begin{aligned} w \begin{pmatrix} u_1 \\ v_1 \\ 1 \end{pmatrix} &= \begin{bmatrix} P_1[0] \\ P_1[1] \\ P_1[2] \end{bmatrix} \begin{pmatrix} X_x \\ X_y \\ X_z \\ X_w \end{pmatrix} \\ \Leftrightarrow w u_1 &= P_1[0] X \\ w v_1 &= P_1[1] X \\ w &= P_1[2] X \\ \Leftrightarrow u_1 P_1[2] X &= P_1[0] X \\ v_1 P_1[2] X &= P_1[1] X \\ \Leftrightarrow X(u_1 P_1[2] - P_1[0]) &= 0 \\ X(v_1 P_1[2] - P_1[1]) &= 0 \end{aligned} \tag{4.43}$$

$P_1[\dots]$ bezeichnet jeweils die indexierte Reihe der Projektionsmatrix. Analog gilt für den zweiten Korrespondenzpunkt folgende Gleichung.

$$\begin{aligned} X(u_2 P_2[2] - P_2[0]) = 0 \\ X(v_2 P_2[2] - P_2[1]) = 0 \end{aligned} \tag{4.44}$$

Mit Hilfe dieser vier Gleichungen kann für jedes korrespondierende Punktepaar eine Lösungsmatrix A erstellt werden, welche den zugehörigen dreidimensionalen Punkt X, mit seinen vier unbekannten Komponenten, eindeutig bestimmt.

$$\overbrace{\begin{bmatrix} u_1 P_1[2] - P_1[0]) \\ v_1 P_1[2] - P_1[1]) \\ u_2 P_2[2] - P_2[0]) \\ v_2 P_2[2] - P_2[1]) \end{bmatrix}}^{\text{A}} X = 0 \tag{4.45}$$

Auch zur maschinellen Lösung von Gleichungssystemen liefert die OpenCV eine Funktion. Durch das Aufrufen dieser Funktion mit der Matrix A und dem Lösungsvektor B, in diesem Fall ein Nullvektor, wird nach X gelöst und der dreidimensionale Punkt bestimmt.

```
1 bool solve(InputArray A, InputArray B, OutputArray X, int
      flags)
```

Quellcode 7: Funktion zur Lösung eines Gleichungssystems

Dieser Vorgang wird für jedes korrespondierende Punktepaar wiederholt.

4.1.7 Das Skalierungsproblem

Betrachten wir die Gleichung 4.7 der Projektion eines dreidimensionalen Punktes auf die zweidimensionale Ebene, so ist diese auch in folgender Form gültig.

$$sp_{2D} = sMP_{3D} \tag{4.46}$$

Der Faktor s bezeichnet eine Skalierung, wie sie auch bei der Bestimmung der intrinsischen Parameter bereits als Teil der Homografie erwähnt wurde. Das Problem liegt darin, dass die Skalierung als Solche nie bestimmt wurde, sondern lediglich die daraus abgeleiteten Rotationen und Translationen. Die Tiefeninformation eines zweidimensionalen Bildpunkts kann somit nur in skalierter Form rekonstruiert werden. Es lässt sich

die gesamte geometrische Proportion herstellen, jedoch bleibt die Einheit des Systems unbekannt.

4.1.8 Implementierung des Frameworks

Nachdem nun der mathematische Hintergrund der mono visuellen Umgebungswahrnehmung analysiert wurde, soll im Nachfolgenden das implementierte Framework vorgestellt werden.

Zu Beginn wurden abstrakte Objekte implementiert, welche die zuvor erwähnten Grundfunktionalitäten, wie das Finden korrespondierender Punkte oder die Rekonstruktion der dreidimensionalen Umgebung an Hand der Korrespondenzpunkte, anbieten. Das Herzstück des Frameworks ist der *MotionProcessor*. Dieser implementiert das mathematische Modell der mono visuellen Odometrie und bietet nach außen Funktionen an, um eine Reihe von Eingangsbildern zu verarbeiten und die berechneten Posen abzufragen. Auch das Abrufen der triangulierten Punktewolke oder des Tiefenbildes werden als Funktionen angeboten.

Abbildung 26 zeigt den Grundaufbau des Frameworks, mit seinen öffentlichen Schnittstellen. Der *MotionProcessor* nutzt zur Verarbeitung der Eingangsdaten den *Reconstructor3D* und den *FeatureMatcher*. Der *FeatureMatcher* hat die Aufgabe in zwei aufeinander folgenden Bildern markante Punkte zu detektieren und eine Korrespondenz zwischen diesen zu bestimmen. Mit Hilfe dieser Korrespondenzen ist der *MotionProcessor* in der Lage, nach dem oben beschriebenen Verfahren, die Odometrie zu berechnen und die Kameraposen zu den aufgenommenen Bildern zu bestimmen. Anschließend wird der *Reconstructor3D* dazu verwendet die Tiefeninformation zu den detektieren Punkten zu bestimmen. Die daraus resultierende Punktewolke wird durch Weitergabe an den *CloudOptimizer* optimiert, in dem nur Punkte berücksichtigt werden welche über eine große Nachbarschaft verfügen.

Die Klasse *LinearTriangulator* implementiert einen *Reconstructor3D* nach dem vorgestellten Verfahren der einfachen Triangulation. Für den *FeatureMatcher* stehen Implementierungen bereit, welche eine Korrespondenz über die AKAZE Features oder diese durch Anwendung des optischen Flusses bestimmt.

Der Vorteil dieses Frameworks besteht darin, dass Grundkomponenten wie die Berechnung der Features oder die Rekonstruktion einfach ausgetauscht und auch in Zukunft weiter optimiert werden können. Konstante mathematische Abläufe, wie die Berechnung der Fundamentalmatrix, lassen sich nur hinsichtlich der Parameter optimieren, weshalb deren Implementierung im *MotionProcessor* selbst verankert ist. Mit Hilfe der OpenCV[49] Bridge, einem Konverter zwischen ROS Botschaften und der OpenCV Bildstruktur, ist es möglich, dass Framework direkt in ROS zu verwenden. Die Ergebnisse der Tiefenrekonstruktion, unter Anwendung der verschiedenen Featuredetektoren, wer-

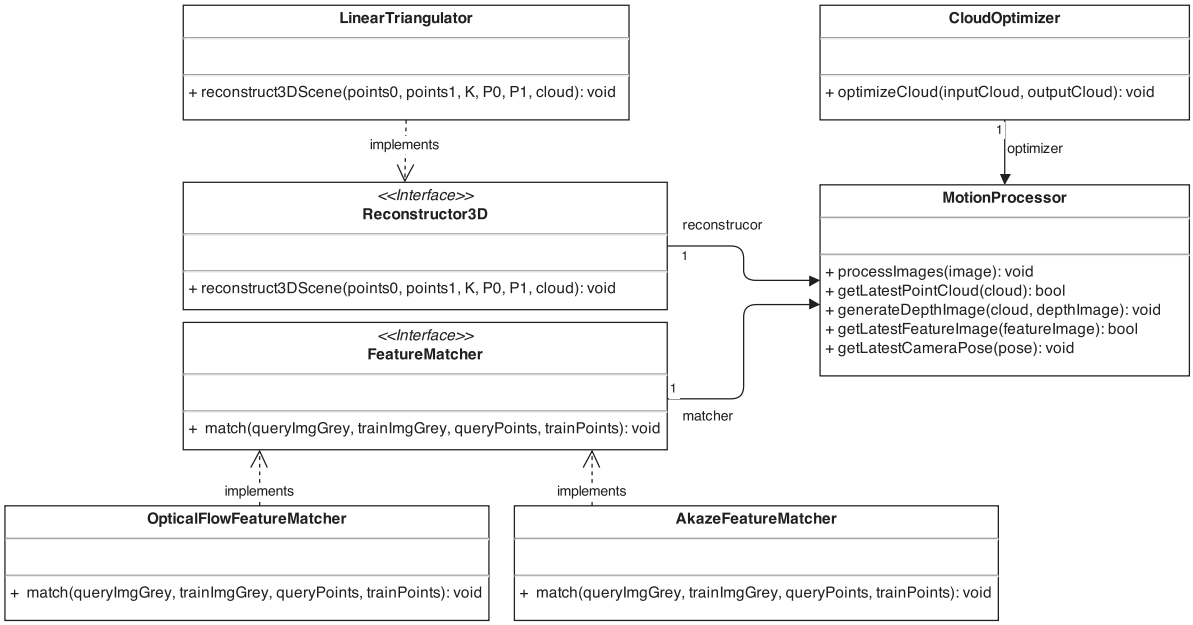

Abbildung 26: UML Diagramm des Structure from Motion Frameworks (Public)

den im Kapitel 5 vorgestellt.

4.2 Parallel Tracking and Mapping (PTAM)

Ein weiteres Framework zur mono visuellen Umgebungswahrnehmung wurde 2007 von Klein und Murray[27] vorgestellt. *Parallel Tracking and Mapping*, kurz PTAM, vereint die Berechnung der eigenen Position sowie die Erstellung einer Karte der Umgebung. Dieser Ansatz wird allgemein als *Simultaneous Localization and Mapping*, kurz SLAM, bezeichnet.

Auch hier dient als Basis der *FastFeature* Detektor. Zu Beginn wird aus zwei Aufnahmen entlang der Basislinie, mit Hilfe der Epipolargeometrie, eine initiale Karte erstellt. Von nun an laufen das Tracking und das Mapping parallel. Das Tracking basiert auf einem zwei-Phasen Filter und lässt sich in 5 Schritten beschreiben.

1. Bei Eintreffen eines neuen Kamerabildes wird die Position an Hand des aktuellen Bewegungsmodells geschätzt.

2. Mit Hilfe der geschätzten Position wird die aktuelle Karte in das Kamerabild projiziert (siehe Kapitel 4.1.1).

3. Phase 1: Eine kleine Menge (ca. 50) Korrespondenzen werden gesucht, in dem die Karte mit den aktuell gefundenen Features verglichen wird.

4. Phase 2: An Hand der Korrespondenzmenge wird die geschätzte Pose optimiert und Schritt 2 wiederholt. Diesmal jedoch mit einer größeren Menge von Punktkorrespondenzen (ca. 1000).

5. Bestimme finale Pose aus allen gefundenen Korrespondenzen.

Die Pose wird jeweils über folgende Gleichung aus einer Menge S von Korrespondenzen optimiert.

$$\mu' = argmin \sum_{j \in S} Obj(\frac{|e_j|}{\sigma_j}, \sigma_T) \tag{4.47}$$

Die Funktion $Obj(\ldots)$ ist eine Gewichtsfunktion, ausschlaggebend ist der Wert e_j, welcher den Projektionsfehler der aktuellen Korrespondenz repräsentiert. Die Menge von korrespondierenden Punkten sowie der Betrag des Projektionsfehlers geben die Qualität des Trackings an.

Ist die Qualität des Trackings gut und hat sich das Kameramodell um eine gewisse Distanz bewegt, versucht der Mapper neue Features in die Karte einzutragen. Dazu müssen zunächst an Hand einer epipolaren Suche aus dem aktuellen Bild und einem bereits registrierten Bild die Tiefeninformationen rekonstruiert werden. Anschließend werden die dreidimensionalen Punkte mit Hilfe des Levenberg-Marquart Bundle Adjustment[57] eingetragen. Der Algorithmus verändert die Karte iterativ um die neuen Informationen fehlerfrei zu integrieren.

Auf Grund der Parallelität von Tracking und Mapping sowie der Eigenschaft, dass nur der Tracker mit einer hohen Prozessrate laufen muss, ist PTAM auch auf kleinen Rechensystemen, je nach Kameraauflösung, echtzeitfähig. Dennoch gilt auch hier, wie bei allen mono visuellen Systemen, das Problem der unbekannten Skalierung.

4.3 Stereo Vision

Der Begriff Stereo Vision vereint die Extraktion von dreidimensionalen Informationen aus digitalen Bildern, welche von einer gewöhnlichen Kamera stammen. Dabei werden zwei Aufnahmen der gleichen Szene aus verschiedenen Blickwinkeln verglichen und so die Tiefeninformationen rekonstruiert. Im Vergleich zur Mono Vision, siehe Kapitel 4.1, ist hier keine ständige Bewegung der Kamera erforderlich, da die Bilder instantan aus zwei Blickwinkeln aufgenommen werden. Voraussetzung ist jedoch, dass die Positionen der beiden Kameras, sowie der Abstand dieser zueinander bekannt sind. Die Idee hinter der dreidimensionalen Bildverarbeitung an Hand von Stereo Bildern basiert auf dem biologischen Prozess der Stereoskopie[61].

In diesem Abschnitt soll die Stereo-Verarbeitung von Bildern anhand der Algorithmen, welche durch die OpenCV zur Verfügung gestellt werden, vorgestellt werden. Zunächst muss jedoch zusätzlich zum intrinsischen Aufbau der einzelnen Kameras, siehe Kapitel 4.1.2, der extrinsische Aufbau des Stereo-Systems beschrieben werden.

4.3.1 Bestimmung der extrinsischen Parameter

Allgemein versteht man unter den extrinsischen Parametern eine Rotation sowie eine Translation zwischen zwei Koordinatensystemen. Bereits bei der Bestimmung der intrinsischen Parameter sind lokale extrinsische Parameter erforderlich gewesen und zwar jene, welche die Ausrichtung des Schachbretts im Kamerakoordinatensystem beschrieben haben.

Bei einem Stereo System versteht man unter extrinsischen Parametern jedoch die relative Ausrichtung der zweiten Kamera zur ersten Kamera. Wir suchen also eine Translation sowie eine Rotation, welche das Koordinatensystem der rechten Kamera in das der linken Kamera überführt.

Vorab sei bereits erwähnt, dass die Gültigkeit der extrinsischen Parameter erlischt, sobald eine der Kamera relativ zur anderen bewegt wird. Dies war auch bei lokalen extrinsischen Parametern der Fall, da diese für jedes Bild, auf Grund der veränderten Lage des Schachbretts, neu bestimmt werden mussten.

Die OpenCV bietet auch zur Kalibrierung einer Stereo Kamera eine Funktion an:

```
1 stereoCalibrate(objectPoints, imagePoints1, imagePoints2,
      cameraMatrix1, distCoeffs1, cameraMatrix2, distCoeffs2, R
      , T,...)
```

Quellcode 8: Stereo-Kalibrierung via OpenCV

Bereits der Aufbau der Funktion zeigt, dass zur Kalibrierung wiederholt ein Datensatz von korrespondierenden Punkten benötigt wird. Der Unterschied liegt hier nur darin, dass zu einem dreidimensionalen Modellpunkt ein zweidimensionaler Punkt im linken sowie im rechten Bild benötigt wird. Dazu wird die Eckpunktsuche des Schachbretts auf beiden Teilbildern ausgeführt.

Bei der Stereo Kalibrierung wird zunächst intern die intrinsische Kalibrierungsfunktion für beide Kameras aufgerufen. Anschließend sind die Bilder frei von Verzerrungen und die extrinsische Kalibrierung kann beginnen.

Betrachtet man nochmal Abbildung 20, so trifft diese Geometrie auch auf den Aufbau

eines Stereosystems zu. Der wesentliche Unterschied besteht darin, dass die Rotation und Translation zwischen den beiden Kameras nun konstant sind, wohingegen beim Monosystem diese für jede Bewegung der Kamera bestimmt werden musste. Ein Punkt in der dreidimensionalen Umgebung wird auf beiden Kameras abgelichtet. Es gelten somit folgende Beziehungen:

$$p_1 = R_1 P + T_1 \tag{4.48}$$

$$p_2 = R_2 P + T_2 \tag{4.49}$$

Da die beiden Kameras bereits intrinsisch kalibriert wurden und somit die Matrizen R_1 sowie R_2 und die Vektoren T_1 sowie T_2 für jedes Kalibrierungsbild bekannt sind lassen sich die Gleichungen sofort lösen. Es wird jedoch nach der Transformation gesucht, welche den Punkt der rechten Kamera in das System der linken Kamera überführt. Bezieht man diese Transformation nun auf den rechten Punkt erhält man folgende Gleichung, wobei R und T die gesuchten extrinsischen Parameter sind:

$$p_1 = R^T(p_2 - T) \tag{4.50}$$

Diese Gleichung wird nun nach R sowie auch nach T umgestellt. Das Umstellen ist problemlos möglich, da es sich bei Rotationsmatrizen um Orthogonalmatrizen handelt und die transponierte Matrix der inversen Matrix entspricht. Betrachtet man zunächst den Rotationsanteil und anschließend den Translationsanteil lässt sich aus den Gleichungssystemen folgende Beziehung herstellen.

$$R = R_2 (R_1)^T \tag{4.51}$$

$$T = T_2 - R T_1 \tag{4.52}$$

Die globalen extrinsischen Kameraparameter lassen sich somit einfach aus den lokalen extrinsischen Parametern, welche als Teilergebnisse aus der intrinsischen Kalibrierung hervorgehen, berechnen. Auch hier gilt: Je mehr Kalibrierungsbilder verwendet wurden, desto mehr Gleichungssysteme existieren und desto genauer werden die finalen Parameter bestimmt. Die extrinsischen Parameter werden ebenfalls in einer YAML Datei abgespeichert.

4.3.2 Rektifizierung der Eingangsdaten

Bevor mit der eigentlichen Rekonstruktion der Tiefeninformation in den Eingabedaten begonnen werden kann, müssen die Daten vorverarbeitet werden. Dieser Schritt wird als Rektifizierung bezeichnet. Dies ist von großer Bedeutung, da in einem kontinuierlichen Strom von Eingangsdaten keine bekannten Punkte mehr in den Bildern existieren, an denen die Tiefe an Hand eines Größenvergleichs bestimmt werden kann. Ferner sind alle Pixel in den Datensätzen willkürlich angeordnet und besitzen keinerlei Beziehungen zu einander. In einem rektifizierten Datensatz jedoch verlaufen die Epipolarlinien nahezu parallel. Das heißt, dass die aufgenommene Szene der Umgebung auf beiden Teilbildern auf einer Höhe liegt. Somit liegt die Information nahe, dass korrespondierende Bildpunkte in der gleichen Zeile zu finden sind. Warum dies Voraussetzung für das Matching ist, wird im späteren Verlauf der Arbeit noch deutlich werden.

Die Rektifizierung besteht im wesentlichen aus dem Anwenden der zuvor bestimmten intrinsischen und extrinsischen Parameter auf die Eingangsdaten. Auch hierzu bietet die OpenCV drei Funktionen an.

```
1 stereoRectify(cameraMatrix1, distCoeffs1, cameraMatrix2,
      distCoeffs2, imageSize, R, T, R1, R2, P1, P2, Q, ...)
2 initUndistortRectifyMap(cameraMatrix, distCoeffs, R,
      newCameraMatrix, ..., map1, map2)
3 remap(src, dst, map1, map2, ...)
```

Quellcode 9: Rektifizierung via OpenCV

Die Funktion *stereoRectify(...)* hat die Aufgabe die finalen Rotationen und Projektionen für die beiden Teilbilder der Stereo Kamera zu bestimmen, so dass diese direkt rektifiziert werden können. Durch die extrinsische Kalibrierung und der daraus folgenden Transformation, können die Teilbilder schon recht nahe aufeinander projiziert werden. Die Transformation zwischen den beiden Kamerakoordinatensystemen wird dazu in eine zweidimensionale Bildrotation sowie der zugehörigen Projektion umgerechnet. Dennoch wird mit Hilfe des Algorithmus von Bougets[9] versucht den daraus resultierenden Projektionsfehler zu minimieren. Dies wird dadurch erreicht in dem von Hand generierte Epipolarlinien, beispielsweise durch die Bildmitte der Teilbilder, rechnerisch horizontal überlagert werden. Auf Grund der Berechnung entsteht eine neue Kameramatrix für jede Kamera, $P1$ und $P2$, welche nun einen dreidimensionalen Punkt in das rektifizierte Bild projiziert. Des Weiteren werden die zwei Rotationen zur Rektifizierung der jeweiligen Teilbilder zueinander, $R1$ und $R2$, zurück geliefert. Zusätzlich generiert die Funktion eine Matrix Q, durch die es möglich wird aus einem Bildpunkt und der zugehörigen Tiefeninformation, welche durch das Blockmatching bestimmt wird, den korrespondierenden Punkt im dreidimensionalen Raum zu bestimmen. Diese Bestimmung ist dann

ohne weitere bekannte Objektinformationen möglich. Der genaue Aufbau der Matrix Q, sowie deren Verwendung, wird im Kapitel 4.3.4 beschrieben.

Die Funktion *initUndistortRectifyMap(...)* fügt im Anschluss alle errechneten Rotationen und Projektionen zu einer sogenannten Projektionskarte zusammen. Benötigte Eingangsdaten sind die intrinsischen Parameter der jeweiligen Kamera (Kameramatrix und Verzerrungskoeffizienten) sowie die soeben errechnet Rotationen und Projektionsmatrizen. Ausgegeben werden anschließend die Projektionsskarten, jeweils eine für X- und Y-Achse des Eingangsbildes, welche mit der Funktion *remap(...)* endgültig auf die Eingabebilder angewandt werden kann.

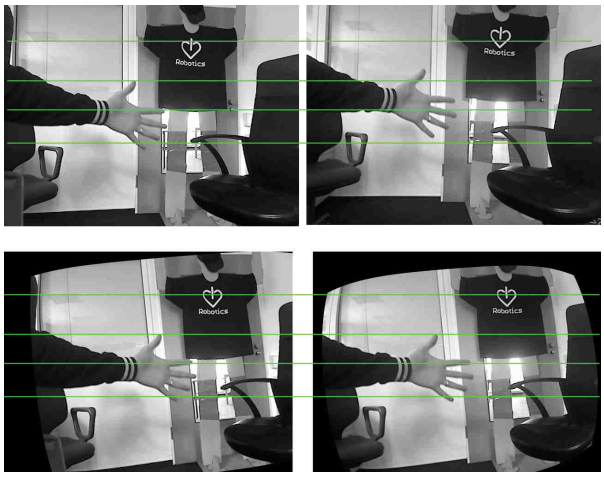

Abbildung 27: Linkes und rechtes Kamerabild vor (oben) und nach (unten) der Anwendung der berechnetet Projektionskarten mit eingezeichneten Epipolarlinien zur Veranschaulichung

4.3.3 Rekonstruktion der Tiefeninformation

In diesem Abschnitt wird die Rekonstruktion der Tiefeninformation und damit das Erzeugen der Disparity Map (zu deutsch Versatz) aus einem Stereo Datensatz beschrieben. Die Disparity Map ähnelt einem gewöhnlichen zweidimensionalen Bild, jedoch sind an den X- und Y- Koordinaten des Bildes keine Farbinformationen sondern der errechnete Versatz gespeichert, was auf Grund der Epipolargeometrie die Tiefe repräsentiert. Dabei bezieht sich das Tiefenbild immer auf das linke Kamerabild.

Auch zur Berechnung der Disparity Map bietet die OpenCV eine einfache Funktion deren Funktionsweise und Verwendung später näher erläutert werden soll. Zunächst wird der Zusammenhang zwischen der Entfernung eines Objektes und der zu errechnenden Tiefe, nachfolgend d (eng. depth) genannt, hergestellt.

Beide Kamerakoordinatensysteme besitzen die gleiche Tiefenachse Z, da sie nebeneinan-

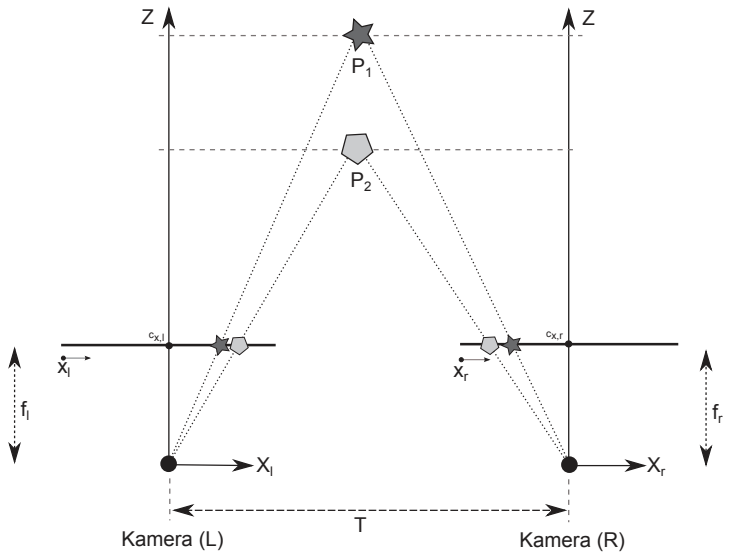

Abbildung 28: Stereo Kamera Modell zur Herleitung der Tiefeninformation d

der stehen. Der Vektor T bezeichnet die Translation zwischen den Koordinatensystemen, welcher aus den extrinsischen Parametern hervorgeht. Die Parameter f_l, f_r, $c_{x,l}$ und $c_{x,r}$ stammen aus den intrinsischen Parametern der beiden Kameras. Die beiden Punkte P_1 sowie P_2 lassen sich bekanntlich nun auf die Bildpunkte $x_{l,1}$, $x_{r,1}$, $x_{l,2}$ und $x_{r,2}$ projizieren.

Bei vollständig entzerrten, ausgerichteten Stereo Bildern gilt nun folgende Gleichung zur Bestimmung der Tiefeninformation d:

$$d = x_l - x_r \tag{4.53}$$

Berechnen wir die Tiefeninformation nun beispielhaft für die zwei Objekte oben im Modell erhalten wir folgende Ergebnisse. Die Teilbilder haben eine Breite von 300px.

$$d_1 = x_{l,1} - x_{r,1} = 210 - 70 = 140$$
$$d_2 = x_{l,2} - x_{r,2} = 240 - 40 = 200$$

Je weiter das Objekt von der Kamera entfernt ist, desto kleiner wird der Tiefenwert. Um diese Tiefenwerte nun für die gesamte Szene zu berechnen stellt die OpenCV folgende Klasse zur Verfügung:

Die Klasse *StereoBM* beinhaltet die Implementierung des Stereo Block Matching Verfahrens nach Konolige [29]. Bei der Initialisierung werden die notwendigen Parameter wie die Anzahl der maximal zu suchenden Tiefen oder die Suchfenstergröße übergeben.

```
1 StereoBM(int preset, int ndisparities=0, int SADWindowSize
    =21)
2 StereoBM::operator()(InputArray left, InputArray right,
    OutputArray disparity, ...)
```

Quellcode 10: Block-Matching Implementierung der OpenCV

Der eigentliche Algorithmus wird über den Operator aufgerufen. Wobei die Eingangsdaten aus den rektifizierten Teilbildern der Stereokamera bestehen und das Ergebnis die Disparity Map der aufgenommenen Szene repräsentiert.

Der Algorithmus arbeitet nach dem eben vorgestellten Modell zur Bestimmung der Tiefe. Das Problem ist dabei nur, dass die jetzigen Daten der Kamera keine bekannten Objekte mehr beinhalten. Es muss lediglich an Hand der Farbinformationen entschieden werden ob zwei Punkte zueinander passen. Dazu wird ein Fehlerfenster, das sogenannte SAD (engl. Sum of absolute difference) Fenster, verwendet. Die Größe des Fensters wird über den Konstruktor des Algorithmus definiert und muss dabei immer eine ungerade Zahl haben (5x5, 9x9, 21x21), so dass ein Ankerpunkt in der Mitte des Fensters existiert. Dieses Fenster repräsentiert nun das Objekt welches in beiden Teilfenstern registriert werden soll. Dazu iteriert der Algorithmus über alle möglichen Position im linken Bild in denen das gesamte SAD Fenster rein gelegt werden kann. Für jeden Iterationsschritt wird das SAD Fenster an die gleiche Stelle im rechten Bild gelegt. Nun beginnt die eigentliche Arbeit des Algorithmus: Für beide Bilder werden die Farbwerte der Pixel im SAD Fenster voneinander abgezogen und ein absoluter Fehler bestimmt. Im Anschluss wird das Fenster, seiner Breite nach, nach links verschoben, was einer Tiefenänderung von 1 entspricht und es wird abermals der absolute Fehler bestimmt. Diese Schritte werden solange wiederholt, bis die maximal zu suchende Tiefe, ebenfalls über den Konstruktor des Algorithmus definiert, erreicht wurde. Dabei werden alle Fehlerwerte in einer Vergleichsfunktion festgehalten. Es ist davon auszugehen, dass irgendwann ein minimaler Fehler gefunden wird, da das Fenster entlang der Epipolarlinie verschoben wird, was nun auch die Notwendigkeit der Rektifizierung offenlegt. Durch das Bestimmen dieses Minimums in der Fehlerfunktion können korrespondierende Bildpunkte identifiziert werden. Der zugehörige Versatz zwischen linken und rechtem Bild repräsentiert die Tiefe und wird in der Disparity Map, an der Stelle in der das SAD Fenster im linken Bild angesetzt wurde, registriert.

Da die Suche lediglich auf einem linearen Vergleich der Farbinformationen der einzelnen Pixel basiert ist der Algorithmus sehr schnell, jedoch stark abhängig von der Textur. Somit können die Disparity Maps in schwach strukturierten Umgebungen, wie z.B. einem langen hellen Flur, unvollständig oder ungenau sein. Gleichzeitig kann es passieren, dass durch Unterschiedliche Belichtungen in den Teilbildern der zugehörige Punkt fehlinterpretiert wird. Um den Algorithmus gegen Beleuchtungsdifferenzen robuster zu machen

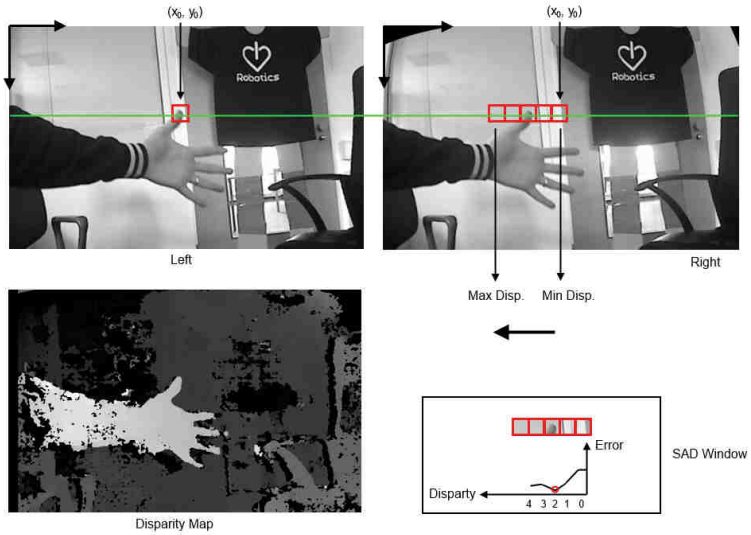

Abbildung 29: SAD Suchverfahren zur Bestimmung der Verschiebung unbekannter Objekte in zwei ausgerichteten Teilbildern

wendet die OpenCV zuvor eine Normalisierung auf die Teilbilder an. Dadurch werden die Teilbilder, je nach Beleuchtungsdifferenz, aufgehellt oder abgedunkelt und Texturen besser hervorgehoben.

Im finalen Tiefenbild erkennt man nun die Tiefenstruktur der aufgenommenen Szene. Objekte im Vordergrund, wie beispielsweise die Hand erhalten helle Schattierungen wohingegen Objekte im Hintergrund, wie z.B. der Stuhl, dunkler schattiert werden. Ein weiteres Merkmal im Tiefenbild ist der Rand an der linken Seite des Bildes. Die Größe des Rands basiert auf der Größe des SAD Suchfensters sowie auch der Verschiebung zwischen den Kameras. Da stetig Bezug auf das linke Bild genommen wird, kann es vorkommen, dass der Bereich, in dem das Fenster im rechten Bild verschoben werden soll, gar nicht existiert. Aus diesem Grund werden die Pixel während der Berechnung zu Null gesetzt.

4.3.4 Rekonstruktion der dreidimensionalen Umgebung

Durch die Stereo-Aufnahme und der anschließenden Verarbeitung erhält man zu jedem Bildpunkt im linken Bild eine zugehörige Tiefeninformation, so dass die Rekonstruktion der dreidimensionalen Umgebung möglich ist. Während der Kalibrierung der Stereo-Kamera wurde eine Matrix Q generiert, mit deren Hilfe die Projektion in eine einfache Rechnung übergeht.

$$Q = \begin{pmatrix} 1 & 0 & 0 & -c_x \\ 0 & 1 & 0 & -c_y \\ 0 & 0 & 0 & f \\ 0 & 0 & \frac{-1}{T_x} & \frac{(c_x - c'_x)}{T_x} \end{pmatrix} \tag{4.54}$$

Die Matrix Q vereint somit das Kameramodel der linken Kamera und die Verschiebung zur rechten Kamera. Durch die Anwendung dieser Matrix auf einen beliebigen Pixel im linken Bild (x, y) sowie der zugehörigen Tiefe d erhält man sofort die Raumkoordinaten (X, Y, Z) im linken Kamerakoordinatensystem.

$$Q \begin{pmatrix} x \\ y \\ d \\ 1 \end{pmatrix} = \begin{pmatrix} X \\ Y \\ Z \\ W \end{pmatrix} \tag{4.55}$$

Es ist somit Möglich aus jedem Bild eine Punktewolke der Umgebung zu generieren. Dieser Punktewolke können auch noch die Farbinformationen hinzugefügt werden, da jeder Punkt aus einem Pixel des linken Bildes bestimmt wurde.

4.4 Tiefensenorik

Analog zu den oben vorgestellten Verfahren sollen nun zwei Sensoren beschrieben werden, welche die Tiefeninformation der Umgebung direkt messen und keine weitere Algorithmik benötigen.

4.4.1 RGB-D Sensor

Aus der Rubrik der RGB-D Sensoren wird der PrimeSense Sensor der Asus Xtion vorgestellt. Der Sensor besteht aus einem IR-Sendearray und einem IR-Empfangsarray. Das Sendearray kann als einfache Infrarotdiode angesehen werden, jedoch projiziert diese keinen einfachen Strahl, sondern ein zweidimensionales Punktmuster mit einer Auflösung von 640×512 Pixeln. Treffen Punkte dieses Musters auf ein Hindernis, werden diese reflektiert und nach einer gewissen Zeit vom IR-Empfänger detektiert. Über die gespeicherte Struktur des Projektionsmusters sowie eine Korrelation mit einem 9×9 Fenster mit dem empfangenen Projektionsmuster wird, analog zu einer Stereo-Kamera, die Tiefeninformation berechnet. Die Berechnung erfolgt direkt auf der Sensorik und wird mit einer Auflösung von 11 Bit gespeichert, sodass 2048 verschiedene Tiefen gemessen werden können.

Die Xtion arbeitet in einer Umgebung von 0,8 Meter bis 3,5 Meter. Die Funktionsweise im Outdoor-Bereich kann, auf Grund störender Infrarotstrahlung des Sonnenlichts, beeinträchtigt werden.

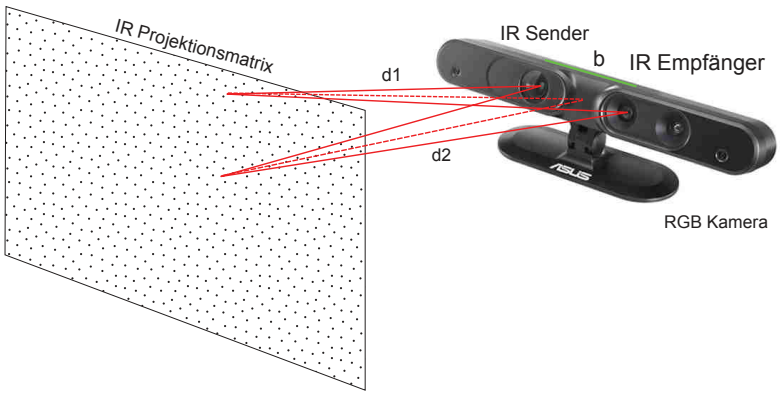

Abbildung 30: Funktionsweise des PrimeSense Sensors

4.4.2 Laserscanner

Der klassische Laserscanner, in dieser Arbeit der Hukoyo UTM-30LX, arbeitet nach dem *Time of flight* Prinzip. Innerhalb des Laserscanners befinden sich, auf einer Rotationsplattform, ein Laseremmiter sowie ein Laserdetektor. Für jede Winkelstellung wird ein Laserpuls emittiert, welcher nach einer gewissen Zeit t durch Reflektion am Hindernis am Detektor detektiert wird. Die vergangene Zeit ist proportional zur Entfernung zum Reflektionspunkt, wobei das Licht stets den doppelten Weg der eigentlichen Entfernung zurücklegt.

$$d = \frac{ct}{2} \tag{4.56}$$

Auf Grund der kürzeren Wellenlänge eines Laserstrahls ergibt sich ein Messbereich von bis zu 30 Meter. Die Rotationsplattform ermöglicht dabei einen Messwinkel von 270 Grad. In Bezug auf die Pulslänge und der Rotationsdauer lassen sich Messungen mit bis zu 40 Hz durchführen. Jedoch liefert der Sensor nur eine zweidimensionale Ebene der gesamten Umgebung, so dass zur vollständigen Umgebungswahrnehmung weitere Aufbauten und Algorithmen notwendig sind. Zur Hinderniserkennung kann jedoch auch die Betrachtung einer Ebene auf gleicher Höhe des UAVs von Vorteil sein.

4.5 3D Scanner

Der dreidimensionale Scanner kann als Erweiterung des zweidimensionalen Scanners angesehen werden. Neben der internen Rotationsplattform wurde der Scanner orthogonal auf eine weitere Plattform montiert um gemessenen Distanzinformationen ebenfalls eine dritte Dimension zuweisen zu können. Durch die schrittweise Rotation der beiden Plattformen ist somit eine vollständige Messung der gesamten Umgebung möglich. Der Laserscanner wurde so montiert, dass der optische Mittelpunkt der Sensordaten genau auf der Z-Achse der Rotationseinheit liegt. Betrachtet man diese nun als Weltkoordinatensystem, so geht die gesamte Transformation eines Scans, neben dem statischen Aufbau der Plattform, in eine einfache Rotation über.

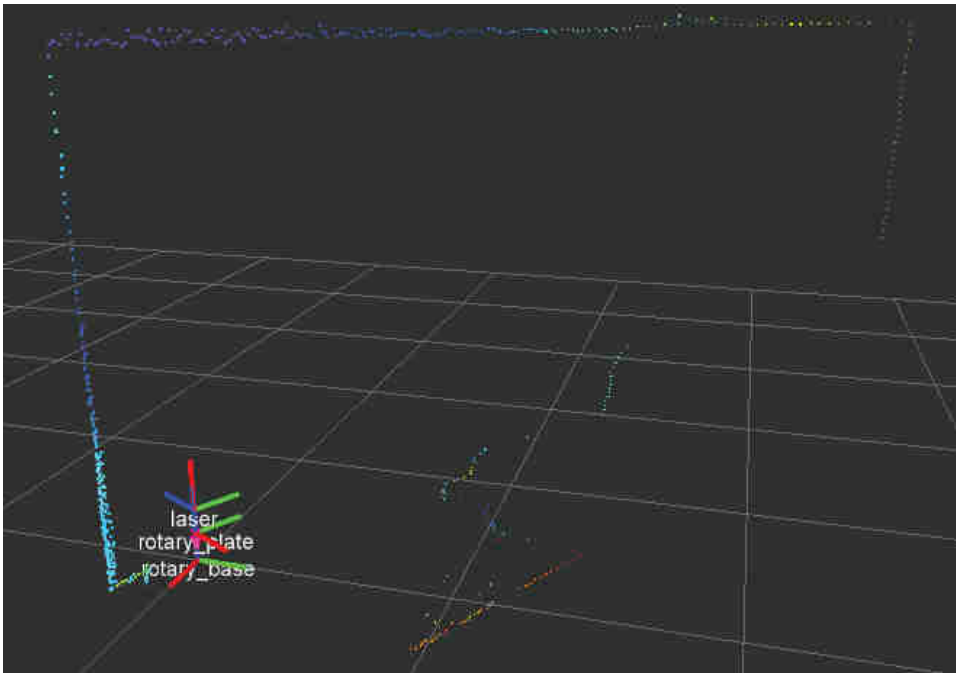

Abbildung 31: Transformation eines einzelnen Laserscans

Die Abbildung 31 zeigt den Aufbau der gesamten Transformationskette bezüglich eines Laserscans. Das Koordinatensystem *rotary_base* kann als Weltkoordinatensystem angesehen werden. Das System *rotatary_plate* definiert die Plattform, welche auf der Dreheinheit montiert ist und beinhaltet neben der statischen Höhe des Aufbaus die aktuelle Winkelstellung des Motors. Das System *laser* beinhaltet lediglich die statische Transformation über die Geometrie des Scanners. Mit jeder Winkelstellung der Motors wird somit ein kompletter dreidimensionaler Schritt des Raumes aufgenommen. Interessant ist nun die Fusion dieser Schnittmengen zu einem komplexen Umgebungsmodell.

Der ROS-Knoten zur Ansteuerung des Motors lässt den Motor nach dem Start sofort mit der eingestellten Drehzahl rotieren. Ferner liefert dieser Treiberknoten die aktuelle Winkelstellung als *TF* Botschaft. Im Weiteren wurde der ROS Umgebung lediglich die

statische *TF Information* über den geometrischen Gesamtaufbau des Systems mitgeteilt. Die gesamte Funktionalität der Fusion befindet sich in der Implementierung des sogenannten *laser_assemblers*. Dieser Knoten empfängt die Daten des Laserscanners sowie die aktuelle Transformation des Scans zur Rotationsplattform. An Hand dieser Transformationsmatrix können die Scanpunkte in dreidimensionale Raumpunkte umgewandelt und in einer Punktewolke gespeichert werden. Nach einer gewissen Zeit, welcher aus der eingestellten Drehgeschwindigkeit resultiert, wird diese Punktewolke vom Assembler an die ROS Umgebung gesendet und beinhaltet das fusionierte dreidimensionale Umgebungsmodell.

Auf Grund der einfachen Speicherung der transformierten Daten, besteht das Modell aus vielen redundanten Punkten. Um diese Redundanz zu lösen und das Modell hinsichtlich der Größe zu optimieren wurde ein einfacher Voxel-Grid Filter implementiert. Dieser Filter fasst alle Nachbarn eines Punktes in einer $n \times n \times n$ Umgebung, auch Blattgröße genannt, zu einem einzigen Punkt zusammen. Wählt man die Blattgröße nun so, dass die theoretische Auflösung des Scanners beibehalten bleibt, wird die Punktewolke verlustlos komprimiert.

5 Evaluierung der Verfahren

Das Kapitel beschreibt die Aufnahme der Testdaten sowie eine Bewertung der gewonnenen Umgebungsinformationen. Auf Basis dieser Evaluierung soll im Anschluss eine Sensorgruppe gewählt werden, welche die Anforderungen an eine autonome Navigation erfüllt und somit am ehesten geeignet ist.

5.1 Definition geeigneter Testdatensätze

Zunächst gilt es, notwendige Testszenarien zu definieren. Die Szenarien sollen sich dabei stets an den späteren Anwendungseinsätzen orientieren und somit eine ähnliche Umgebung, wie die eines USAR Einsatzes, wiedergeben.

Abbildung 32: Beispielhafte Umgebung eines Indoor USAR Szenarios, aufgenommen auf der TJex 2014.

Abbildung 32 zeigt das Innere einer militärischen Hospiz-Ruine in Pisa. Eine solche Umgebung ist ein klassischer Fall für ein „Such- und Rettungsszenario“. Abgesehen von dem Graffiti, welches im Laufe der Zeit durch Vandalismus entstanden ist, zeigt die Umgebung nur wenig Strukturinformationen. Leider konnten während des Aufenthaltes in Pisa nicht alle Testdaten gesammelt werden, so dass für weitere Aufnahmen ebenfalls Aufnahmen im Kellergewölbe eines Wohnhauses durchgeführt wurden.

Neben den allgemeinen Anforderungen an die Umgebungen gilt es, spezifische Merkmale für die Testszenarien zu definieren, welche anschließend in allen Sensoraufnahmen detektiert und bewerten werden können. Zu erwähnen sei hierbei, dass es sich bei den Bewertungen um rein subjektive Ergebnisse handelt. Auf Grund der Struktur im Keller, sowie den verschiedenen Sensoren werden keine metrisch-vergleichbaren Resultate

präsentiert. In jedem Abschnitt dieses Kapitels wird ein solches Szenario vorgestellt sowie die verschiedenen Ergebnisse dokumentiert. Am Ende des Kapitels soll die Auswertung der Szenarien tabellarisch dargestellt werden. Ein abschließendes Experiment wird Aufschluss über die Genauigkeit geben und somit vergleichbare Messergebnisse liefern.

S1: Bewegung durch das Szenario

Das erste Szenario soll eine einfache Bewegung durch das Gelände wiedergeben. Dieses Szenario entspricht in der Regel dem Durchschnittseinsatz und soll als Resultat erste Informationen zur allgemeinen Raumstruktur liefern. Das Szenario wurde wie folgt nachgestellt. In einem langen strukturlosen Korridor galt es rechts eine Treppe zu erkennen. Der Korridor wurde auf der Hälfte des Weges durch einen „eingestürzten“ Querbalken blockiert. Neben einer offenen und zwei verschlossenen Türen, gab es einen kleinen Bruch in einer Wand zu detektieren. Die Aufnahme der Daten wurde mit allen Sensoren bei gleicher, niedriger Geschwindigkeit durchgeführt.

Die Anforderung an die niedrige Geschwindigkeit ergibt sich aus zwei Bedingungen. Zum einen ist bekannt, dass visuelle Kameraaufnahmen bei schnellen Bewegungen oder Wacklern Unschärfe entwickeln oder das Bild stark fragmentieren. Zum anderen wird das autonome UAV auf Grund der Sicherheit im Innenraum über Geschwindigkeitsbegrenzungen verfügen. Dennoch zeigten sich schon beim ersten Testszenario seitens der mono-visuellen Umgebungswahrnehmung gravierende Probleme. Bei den featurebasierten Verfahren wurden, auf Grund der der strukturlosen Umgebung, nur wenig oder unzureichende Features detektiert. Dies hat zur Folge, dass die Fundamentalmatrix und damit die Eigenbewegung falsch geschätzt wird. Unabhängig davon sind zur Umgebungsinterpretation zu wenig Elemente vorhanden, sodass bereits der Querbalken, welcher im Szenario aufgestellt wurde, nur über eine Hand von Punkte erkannt wurde. Auch das PTAM Framework verlor nach kurzer Zeit das Tracking und registrierte die wenig erkannten Featurepunkte nicht mehr in der Karte.

Die stereo-visuelle Abtastung zeigte hier mehr Erfolge. Grundlegende Raumkonturen und Hindernisse, wie der Querbalken, wurden vollständig erkannt. An langen strukturlosen Wänden kam es jedoch zu Fragmenten undefinierter Tiefe. Diese können jedoch in der Nachberarbeitung durch Filter interpoliert werden.

Mit Hilfe des zweidimensionalen Scanners konnte mittels SLAM Verfahren eine genaue Karte des Szenarios erstellt. Diese gab jedoch nur die Ebene des Scanners wieder, sodass auch nur auf dieser Ebene sicher navigiert werden könnte. Anders hier die dreidimensionale Sensorik. Der 3D Scanner sowie auch die xTion haben das Szenario vollständig abgetastet. Der Bruch in der Wand, die Durchgänge, der Querbalken und auch die Treppe wurden erfolgreich erfasst. Die Aufnahmen selbst unterschieden sich in zwei Dingen.

Der 3D Scanner liefert zum einen hoch-aufgelöste Aufnahmen von der Umgebung, welche

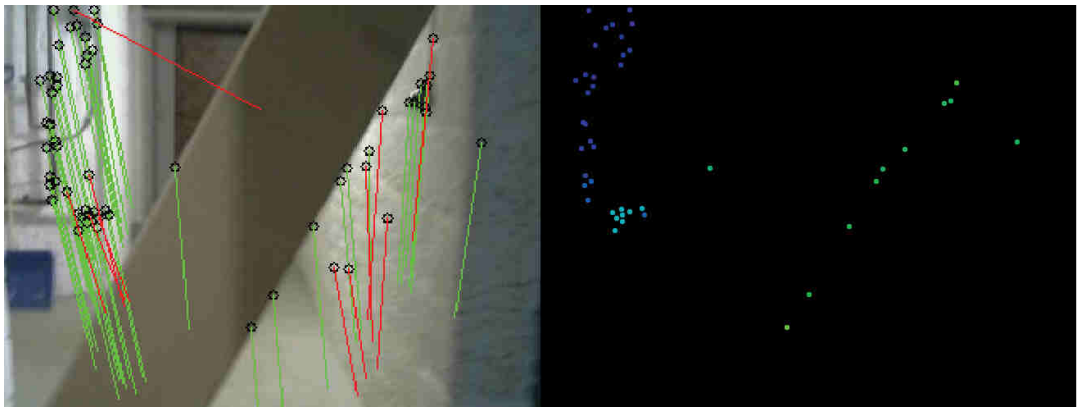

Abbildung 33: Ergebnis des mono-visuellen Featureerkennung (links) bei Szenario 1 am Beispiel des Querbalkens. Es wurden nur teilweise korrekte Korrespondenzen gefunden, sodass die Punktewolke (rechts) sehr dünn besetzt ist. Der Balken ist mit „relativen“ Distanzinformationen eingezeichnet, dessen Größe lässt sich jedoch nicht rekonstruieren.

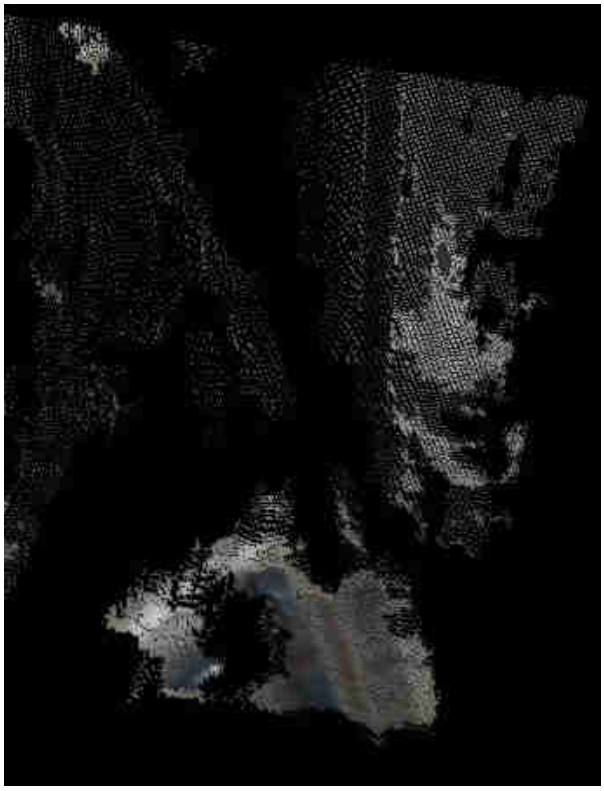

Abbildung 34: Ergebnis des stereo-visuellen Wahrnehumg von Szenario 1. Balken, Türen und Bodenhindernis sind erkennbar. Die schwarzen Löcher resultieren aus Schatten und den gefilterten fehlerhaften Fragmenten

auch in den hinteren Bereichen keine sichtbaren Diskretisierungsschwankungen aufweisen. Die xTion hingegen wird mit zunehmender Distanz ungenauer, sodass eine Schubladenoptik entsteht. Vorteilhaft hier ist die zugehörige, registrierte RGB Information, welche dem humanoiden Betrachter eine bessere Sichtweise des Szenarios ermöglicht.

S2: Bewegung durch das Szenario bei sehr schlechten Lichtverhältnissen

Analog zum ersten Szenario soll die Grundanforderung der Umgebungserkennung ebenfalls bei sehr schlechten Lichtverhältnissen evaluiert werden. In Katastrophenszenarien stehen häufig schwache, bis hin zu nicht ausreichenden, Lichtquellen zur Verfügung. Gerade die visuelle Sensorik ist jedoch stark von der Umgebungsbeleuchtung abhängig. Dieses Szenario ist von essentieller Bedeutung, da in Rettungseinsätzen die Lichtverhältnisse nicht vorhersehbar und in der Regel nicht dem Idealfall entsprechend sind. Zwar kann durch künstliche Lichtquellen Abhilfe geschaffen werden, dies setzt jedoch voraus, dass das UAV weitere Hardware tragen muss oder der Einsatzort selbst vorab manipuliert werden muss. Hinzu kommt, das künstliche Lichtquellen durch Schatten die visuell aufgenommene Umgebung stark verfälschen können.

Die Ergebnisse dieses Szenarios sind vorhersehbar. Die Featureerkennung bricht mit nachlassender Ausleuchtung völlig ein, auch die stereo-visuelle Wahrnehmung wird zunehmend fehlerhaft und fragmenthaltig. Bei sehr schwacher bis fehlender Lichtquelle ist das aufgenommene Bild vollkommen Schwarz und somit keine Umgebungsinformationen mehr vorhanden.

Die laser- beziehungsweise infrarotbasierten Verfahren sind für schwankende Lichtquellen unanfällig. Auch bei völliger Dunkelheit werden Distanzmessungen vorgenommen. Lediglich die xTion verliert die RGB Registrierung, zeigt jedoch weiterhin das korrekte Raummodell.

S3: Erkennung eines statischen Hindernisses in Bewegungsrichtung

Das dritte Szenario prüft explizit die Bewegung in Richtung eines statischen Hindernisses. Dieses Szenario deckt sich mit dem ersten Szenario, da der Querbalken als statisches Hindernis betrachtet werden kann. Somit sind auch die Ergebnisse deckungsgleich. Die mono-visuelle Sensorik erfasst das Hindernis bei gefundenen Features nur partial und lässt somit keine Rückschlüsse auf dessen Größe zu. Die Stereokamera ist in der Lage das Hindernis sowie die Entfernung zu diesem zu erfassen. Auch die instantane, dreidimensionale Messsensorik erkennt das Hindernis problemlos. Mit welcher Entfernung ein statisches Hindernis erkannt werden kann, wird im späteren Verlauf des Kapitels durch die metrische Versuchsreihe deutlich.

Abbildung 35: Ergebnis der Abtastung von Szenario 1 durch die xTion. RGB Information sind entsprechend der gemessenen Distanzen dargestellt. Auch hier entstehen die schwarzen Flecken durch Verdeckung von Hindernissen.

Abbildung 36: Ergebnis des 3D Scans von Szenario 1. Die Auflösung ist deutlich höher als bei der xTion auch der Sichtbereich ist durch den 360 Grad Scan deutlich höher. Lediglich die RGB Informationen fehlen.

S4: Erkennung eines dynamischen Hindernisses in Bewegungsrichtung

Anders verhält es sich beim vierten Szenario, der Erkennung eines dynamischen Hindernisses. Ein dynamisches Hindernis bewegt sich unabhängig vom Sensor, sprich die Richtung sowie die Geschwindigkeit variieren von der des UAVs. Die Ergebnisse dieses Szenarios lassen sich allgemein festhalten. Mono-visuelle Systeme sind stark anfällig für dynamische Bewegungen. Arbeitet das System nicht auf Basis von Keyframes oder werden gar Features des dynamischen Objektes zu einem Keyframe hinzugefügt kann die Eigenbewegung völlig fehlinterpretiert werden. Auch bei der Stereokamera können, je nach Verarbeitungszeit der einzelnen Bilder, Verschiebungen eingefangen und Fehlinformationen generiert werden. Die Umgebungswahrnehmung wird somit Abhängig von der Prozesszeit des Sensorsystems sowie der resultierenden Geschwindigkeit des dynamischen Objektes.

Diese Aussage trifft ebenfalls auf den dreidimensionalen Laserscanner zu. Bewegt sich ein Objekt während der Aufnahme eines Scans, sprich während einer halben Drehung des Scanners, wird das Hindernis gleich mehrfach an verschiedenen Positionen abgetastet. Die verdeckte Umgebungsinformation geht verloren. Beim zweidimensionalen Scanner, trifft dieses Phänomen ebenfalls zu, jedoch ist hier die Prozesszeit einer Abtastung weitaus geringer, wodurch die Dynamik in der Regel messbar bleibt.

Bei der xTion hingegen liegt das Messergebnis quasi instantan vor, sodass ein dynamisches Objekt zum Zeitpunkt seiner Abtastung in ein statisches Hindernis übergeht. Es gelten somit die Ergebnisse aus dem vorherigen Szenario.

S5: Erkennung eines statischen Objekts am Boden

Die Erkennung eines statischen Objektes am Boden ist für die eigentliche Navigation des UAVs zu vernachlässigen, dennoch ist das Detektieren solcher Objekte für die kooperative Arbeit mit einem Bodenroboter oder gar die Erkennung eines menschlichen Opfers von großer Bedeutung. Aus diesem Grund soll dieses Szenario eine solche Erkennung abbilden.

Die Bewertung dieses Szenarios orientiert sich ebenfalls an den Ergebnissen des dritten Szenarios. Zusätzlich spielt hier jedoch der Öffnungswinkel, beziehungsweise der Erfassungsbereich des Sensorsystems eine große Rolle. Beispielhaft verfügt die xTion über einen Öffnungswinkel von 45 Grad[6], sodass bei einer Flughöhe von 1,5 Metern Objekte am Boden erst ab einer Entfernung von 3,6 Metern erkennbar werden, wie Abbildung 37 unter Verwendung einfacher trigonometrischer Funktionen zeigt.

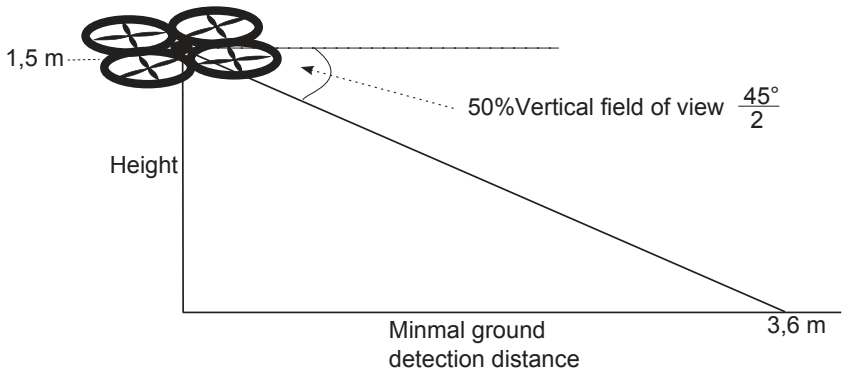

Abbildung 37: Darstellung der minimalen Erkennungsweite von Bodenobjekten in Abhängigkeit von der Flughöhe

S6: Bewegung in Richtung eines lichtdurchlässigen Hindernisses

Es ist bekannt, dass alle getesteten Sensorsysteme auf Basis von sichtbarem oder nicht sichtbarem Licht arbeiten. Aus diesen Grund soll Szenario 6 die Option testen, lichtdurchlässige Objekte in Form von Fenstern ganz oder teilweise zu detektieren.

Das Ergebnis bestätigt natürlich den Verdacht, dass die Sensoren konsequent durch lichtdurchlässige Objekte hindurch messen. Als Ergebnis erhält man Distanzen zu den dahinter liegenden Objekten. Abhilfe schafft hier ein einfacher Ultraschall Sensor, welcher das gemessene Ergebnis in Sensorrichtung verifiziert, beziehungsweise stets einen elementaren Kollisionsschutz liefert.

Abbildung 38: Abtastung eines lichtdurchlässigen Hindernisses am Beispiel der xTion. Die Infrarotstrahlen dringen durch und erfassen die Wand hinter dem Fenster.

5.2 Distanz- und Genauigkeitsmessungen

In diesem Abschnitt sollen, wie bereits erwähnt, metrische Messungen präsentiert werden. Mit Hilfe dieser Daten kann die Genauigkeit der Sensoren sowie die minimal messbare Distanz bestimmt werden. Diese Messungen erfolgen nicht für das mono-visuelle System, da einerseits das Ergebnis unskaliert wäre und andererseits eine Bewegung der Kamera erfolgen muss, um einen Featurevergleich oder ähnliches durchführen zu können. Diese Bewegung macht eine genaue Distanzmessung unmöglich.

Abbildung 39: Strukturierte Wand, aufgestellt im 90 Grad Winkel, zur Messung der Sensorgenauigkeit.

Mit Hilfe einer stark strukturierten Wand, welche in verschiedenen Entfernungen zu den Sensoren positioniert wird, soll die Messgenauigkeit der Sensorik bewertet werden. Die Struktur wurde durch einfaches Zeitungspapier hergestellt. Die einzelnen Sensoren wurden frontal zur Messwand auf einer Wasserwaage ausgerichtet, so dass die reale Distanz genauestens bestimmt werden konnte. Auf Grund der Bauweise der verschiedenen Sensoren ergeben sich Unterschiede von bis zu einem Zentimeter bei der realen Entfernung zur Strukturwand, weswegen die gesamte Messreiche in Zentimetern angegeben wird.

Sensor	Minimal messbare Distanz (cm)
Laser	1
xTion	49
Stereokamera	138

Tabelle 1: Minimal messbare Distanz der Sensoren

Tabelle 1 zeigt zunächst die minimal messbaren Distanzen der Sensoren. Der Laserscanner misst ab einem Zentimeter, was auf das Gehäuse des Scanners selbst zurückzuführen ist. Die xTion liefert ab einer Entfernung von 49 Zentimetern brauchbare Messergebnisse, was somit 31 cm unter den Angaben des Herstellers liegt[6]. Die minimal messbare Distanz der Stereokamera ist stark abhängig von der Qualtiät der Kalibrierung, der

Auflösung der Kamera selbst, sowie den gewählten Disparitätsparametern. In diesem Fall wurde die Kamera einfach Kalibriert und so parametrisiert, dass mittlere Distanzen über höhere Auflösung verfügen, zu Lasten der minimalen Distanz.

Die maximalen Distanzen des Lasers und der Kinect sind ebenfalls den Datenblättern zu entnehmen. Für die Stereokamera ist auch dies wieder von den Parametern abhängig. In dieser Arbeit wurde der Bereich bis zu 6 Metern getestet, was zur autonomen Navigation im Innenraum als ausreichend eingestuft wurde.

Distanz	Laserscanner		xTion		Stereokamera	
	μ	σ	μ	σ	μ	σ
100	99	0	100	1	-	-
200	199	0	202	0	219	3
300	299	0	303	1	311	6
400	399	0	406	1	399	3
500	499	0	510	5	478	13
600	599	0	614	8	571	9

Tabelle 2: Mittelwert und Standardabweichung der Sensoren bei fünf wiederholten Messvorgängen auf verschiedenen Distanzen.

Tabelle 2 zeigt die Ergebnisse der Distanzmessungen. Es ist zu sehen, dass der Laserscanner im Messbereich stets ohne Abweichungen im Zentimeterbereich misst. Bei der xTion macht sich die in Abhängigkeit zur Entfernung zunehmende Varianz bemerkbar, dennoch liefert der Sensor bis zu 6 Meter Ergebnisse, wobei der maximale Messbereich laut Hersteller auf 3,8 Meter begrenzt ist[6]. Bei der Stereokamera zeigt sich die Diskretisierung der Disparitäten. Im mittleren Bereich sind die Abweichungen geringer, bei zunehmender Distanz steigen jedoch die Abstände zwischen den Disparitäten und damit die Varianz des Ergebnisses.

Die Varianzschwankungen des RGBD Sensors stellte auch Herr Oppermann in seiner Messreihe zur Analyse der Genauigkeit unter Zuhilfenahme der Kinect fest[45]. In seinem Experiment zeigten sich Messchwankungen von bis zu 1,7 cm in einem Messbereich von 3 Metern.

5.3 Ergebnis der Evaluierung

Bereits während der Aufnahme der Testdaten und somit auch den ersten Tests der verschiedenen Wahrnehmungsverfahren zeigte sich, dass mono-visuelle Sensoren zur Lokalisierung oder gar zur autonomen Navigation eines UAVs völlig ungeeignet sind. Zwar lässt sich die Eigenbewegung des Sensors unter guten Bedingungen wie Sensorqualität, Kalibrierung, Lichtverhältnis oder die Wahl der Parameter schätzen, dennoch geht das Tracken der Bewegung schnell verloren. Wird ein featurefreies Verfahren wie der op-

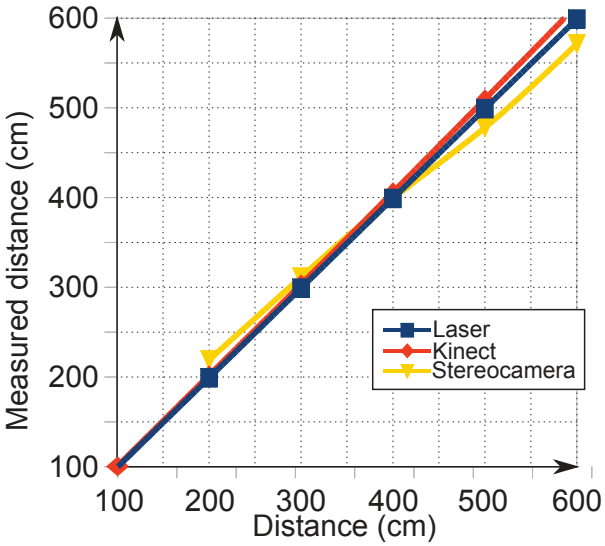

Abbildung 40: Mittlere gemessende Distanz der Sensoren

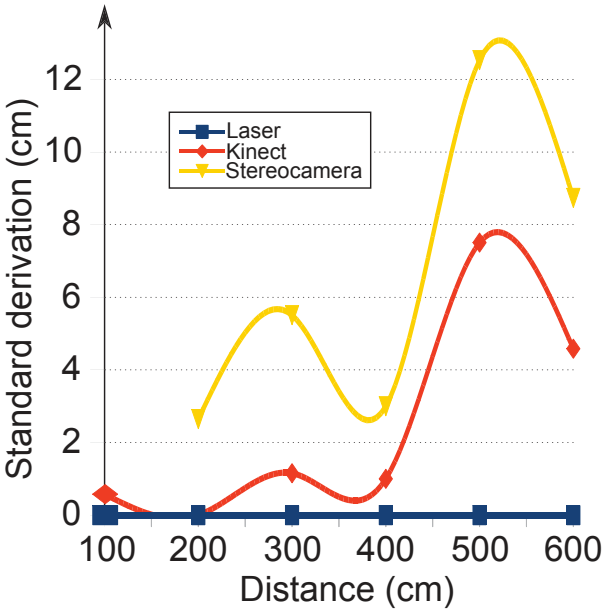

Abbildung 41: Gemessene Standardabweichung der Sensoren

tische Fluss benutzt, ist die Schätzung zwar permanent gegeben, aber auf Grund der Integration kleiner Fehlschätzungen nach kurzer Zeit unbrauchbar. Einfache Feldversuche ermöglichten relative Flugbewegungen von 2-3 Metern mit Hilfe des PTAM Systems. Rotationen oder weitere Bewegungen führten jedoch zum Verlust des Initialsystems. In strukturlosen Umgebungen, in denen auch starke Rotationen nötig sind um generell voran zu kommen, beispielsweise um in den nächsten Raum zu fliegen, ist diese Art von Sensoren ungeeignet.

Auch das *svo*[19] Verfahren bestätigt diese Aussage. Die Entwickler des Verfahren selbst geben an, dass zur Schätzung der Eigenbewegen auf den Boden gerichtete Kameras mit einer hohen Flughöhe am besten funktionieren, da so stets kleine Änderungen bezüglich des Intitalsystems stattfinden. Ebenfalls sollen pure Rotationen vermieden werden.

Die nachfolgende Tabelle 3 soll die Sensoren bezüglich der Ergebnisse der einzelnen Szenarien sowie weiteren wichtigen Kriterien bewerten. Wichtige Kriterien, welche bei einem UAV nicht außer Acht gelassen werden sollten, sind die Größe sowie das Gewicht, da das Gewicht erhebliche Auswirkungen auf die Flugzeit hat. Ebenfalls nicht zu vernachlässigen sind die Kosten eines Sensorsystems. Die Szenarien S5 sowie S6 werden auf Grund redundanter Ergebnisse nicht aufgeführt.

Sensor	S1	S2	S3	S4	Gewicht (g)	Größe (cm)	Preis (Eur.)
Mono (PTAM)	-	-	-	-	89	20x7x1	20
Mono (SfM)	-	-	-	-	89	20x7x1	20
Stereo	+	-	+	+	202	40x7x1	40
Laserscanner	+	+	+	+	370	60x60x87	4900
xTion	++	++	++	++	170	18x3x5	100
3D Scanner	++	++	++	-	>370	>60x60x87	>4900

Tabelle 3: Aufstellung der Sensorbewertungen. (-) Szenario nicht erfüllt. (+) Szenario erfüllt. (++) Optimales Ergebnis

Die mono-visuelle Sensorik ist zwar leicht und sehr günstig in der Anschaffung, dennoch erfüllt sie keines der vorgestellten Szenarien. Der Laserscanner liefert gute Ergebnisse, jedoch nur im zweidimensionalen. Daher kann auch hier kein optimales Ergebnis erreicht werden. Die xTion sowie der dreidimensionale Scanner sind, abgesehen von der Erfassung dynamischer Hindernisse, gleichgestellt. Zwar ist die Qualität der Daten eines 3D Scanners bei weitem hochauflösender, dies führt jedoch bei reinen Lokalisierungs- / Navigationsanwendung zu unnötiger Informationsverarbeitung. Zu dem kommt das geringe Gewicht sowie der niedrige Preis der xTion, welche dem dreidimensionalen Scanner, dessen Preis und Gewicht von der Bauweise der Rotationsplattform abhängigen, weit unterlegen sind.

Auf Grund dieser Auswertung wird für die autonome Navigation eines UAVs ein verlässlicher, dreidimensionaler Sensor wie die xTion vorgeschlagen. Durch das geringe Gewichts ist das System leicht zu integrieren und ersetzt ebenfalls eine RGB Kamera.

Hinzu kommt, dass zunächst keine weitere Algorithmik zur Aufnahme einer Momentmessung notwendig ist. Sind hochwertige Karten nötig, kann der Einsatz eines rotierenden Laserscanners, wie Droeschel u.a.[13] ihn verwenden, vorgezogen werden.

Alles in allem liefert die dreidimensionale Sensormessung in jedem Fall eine Punktewolke, welche ein momentanes Abbild der erfassten Umgebung enthält. Aus diesem Grund soll im nächsten Kapitel ein erstes Verfahren zur Navigation in diesen präsentiert werden.

6 Navigation

Die Navigation beschäftigt sich grundlegend mit den Fragestellungen „Wo bin ich?“ und „Wie komme ich an mein Ziel?“. Diese Fragestellungen definieren die Komponenten zur autonomen Navigation eines UAVs.

- Lokalisierung
- Pfadplanung

Die erste Komponente ist zuständig für die Bestimmung der aktuellen Position des UAVs und somit Grundvoraussetzung für die eigentliche Pfadplanung. Die Lokalisierung basiert auf einer Fusion der internen IMU mit den gemessenen Umgebungsinformationen und ist somit stark abhängig von der Sensorik des UAVs, welche zum Zeitpunkt der Durchführung dieser Arbeit noch nicht final feststand.

Aus dem vorherigen Kapitel geht jedoch hervor, dass jede evaluierte Sensorgruppe als Ergebnis eine Punktewolke liefert, welche ein Abbild der erfassten Umgebung repräsentiert. Auf Grund dieser Information konnte ein erstes allgemeines Verfahren zur dreidimensionalen Pfadplanung entwickelt und in einer Simulation getestet werden. Die Funktionsweise dieses Navigationsverfahren sowie zukünftige Anwendungsmöglichkeiten sollen im nachfolgenden beschrieben werden. Gegen Ende des Kapitels soll ebenfalls die Simulationsumgebung vorgestellt werden.

6.1 Dreidimensionale Pfadplanung

Die Pfadplanung beschäftigt sich mit der Suche einer Konfigurationsfolge, welche das UAV sicher durch den dreidimensionalen Raum und stets in Richtung des definierten Ziels bewegt. Um diese Aufgabe zu bewältigen lassen sich folgende Teilprobleme definieren.

- Diskretisierung des Konfigurationsraums, nachfolgend *C-Space* genannt
- Bestimmung des Suchalgorithmus
- Neuberechnung eines Suchpfades
- Verfolgung eines gefundenen Pfades, umgangssprachlich *Path tracking*

Dieses Kapitel beschäftigt sich mit der Definition und der Implementierung dieser Teilproblematiken um anschließend sicher navigieren zu können. Im Nachfolgenden wird angenommen, dass die Position des UAVs als Transformation sowie die gemessenen Umgebungsinformationen als Punktewolke zur Verfügung stehen. Es wird von der eigentlichen

Hardwareplattform abstrahiert, wodurch diese in einer einfachen Simulationsumgebung beschrieben werden kann.

Der Stand der Technik zeigt, dass seitens ROS bereits ein umfangreiches Framework zur Navigation zur Verfügung gestellt wird. Diese Algorithmen arbeiten jedoch alle im zweidimensionalen Raum und sind somit zur Pfadplanung im Dreidimensionalen nur bedingt geeignet. Es existieren diverse Ansätze, welche die zweidimensionale Navigation ebenfalls für das UAV verwenden[21], das UAV dadurch jedoch in seinen Freiheitsgraden einschränken und nur in einer definierten Ebene sicher navigieren beziehungsweise kartieren können. Ein weiteres Framework zur Pfadplanung ist das Paket *move_it*[25], welches Pfadplanungsalgorithmik für klassische Gelenke im dreidimensionalen Raum bietet und ebenfalls Teil der ROS Umgebung ist. Anwendung findet diese Art der Pfadplanung bei der Bestimmung von Gelenkwinkeln um einen Roboterarm beispielsweise in eine vorgegebene Position zu verfahren. Ein UAV kann jedoch ebenfalls als einfaches Gelenk im Raum beschrieben werden, welches alle möglichen Positionen, physikalische Einschränkungen vorerst außer Acht gelassen, einnehmen kann. Ferner bietet MoveIt! ein Sensorinterface, welches es erlaubt eingehende Punktewolken direkt in die Pfadplanung einzubinden. Auf Grund dieser Optionen sollte MoveIt! als Navigationsframework näher betrachtet werden. Bevor dieses nun vorgestellt und die eigentliche Pfadplanung erläutert wird, soll zunächst die Datenrepräsentation der Punktewolke sowie die notwendige Vorverarbeitung beschrieben werden.

6.1.1 C-Space Diskretisierung

Hoch aufgelöste Punktewolken sind extrem speicheraufwendig und erzeugen somit auch ineffiziente Zugriffszeiten. Die Diskretisierung des sogenannten *C-Spaces* ist ein essentieller Schritt um den Zugriff zu optimieren, damit Suchalgorithmen effizienter arbeiten können. Im Vergleich zum klassischen Arbeitsraum, dem realen Umfeld des UAVs, repräsentiert der Konfigurationsraum alle möglichen Konfigurationen oder auch Zustände, welche das UAV einnehmen kann. Dadurch benötigt der *C-Space* in der Regel mehr als drei Dimensionen zur Beschreibung einer dreidimensionalen Pose. Eine Konfiguration kann die eigentliche Position, die Orientierung sowie eventuelle Geschwindigkeiten beinhalten und beansprucht damit bereits 9 Freiheitsgrade. Der *C-Space* kann als Erweiterung des klassischen Arbeitsraumes angesehen werden.

Für die Navigation des UAVs wird ein Konfigurationsraum mit 6 Freiheitsgraden verwendet. Drei für die Translation und drei für die Rotation. Es wird sich zeigen, dass zur Kollisionsprüfung lediglich der Translationszustand geprüft werden muss, da das UAV in der Pfadplanung als Punkt betrachtet wird.

Es ist klar, dass klassische Ansätze wie die Rasterzerlegung zur zweidimensionalen Pfadplanung, bei der die Umgebung diskretisiert und die Belegtheit einer Zelle in einer Matrix abgespeichert wird, bei höheren Freiheitsgraden an ihre Grenzen stoßen. Aus diesem

Grund werden bei höheren Dimensionen Baumstrukturen zur Planung verwendet. Denkt man an klassische Computeralgorithmen, wo Probleme zur effizienten Lösung mittels Binärbäumen dargestellt werden, so bedient man sich im dreidimensionalen Raum den Octrees (von octo, lat. „acht“).

Auch hier wird nach der klassischen Strategie „Teile und Herrsche“ der dreidimensionale Raum in jeder Dimension geteilt. Die entstandenen Oktanten beinhalten nun Teildaten der Vorgängers und nehmen einen bestimmen Zustand ein. Die Hierarchie dieser Oktanten lässt sich als Baum darstellen, wobei ein Knoten genau acht oder keine Kinder hat.

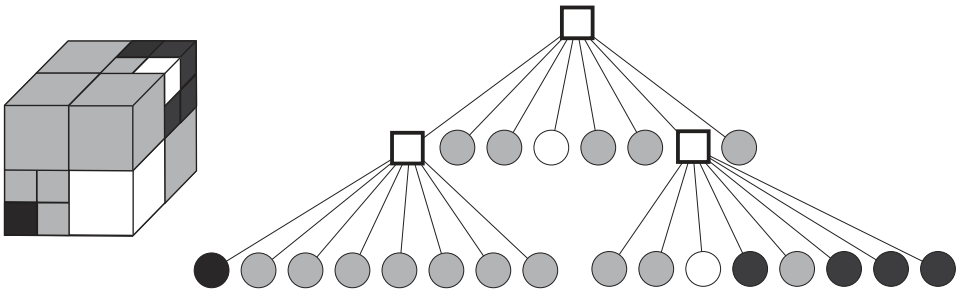

Abbildung 42: Beispielhafte Darstellung zur Bildung eines Octrees. Links die geometrische Darstellung, rechts die Darstellung als Baum. Ein Blatt trägt den Zustand belegt (dunkelgrau), frei (weiß) oder ist nicht definiert (grau).

Abbildung 42 zeigt beispielhaft die Zerlegung einer dreidimensionalen Darstellung in seine Oktanten. Eine Datenmenge muss dabei weiter untergliedert werden, wenn ihre Oktanten unterschiedliche Zustände einnehmen. So wird zunächst der gesamte Raum geteilt. Als Ergebnis zeigt sich, dass sechs der entstanden Oktanten in einer weiteren Unterteilung allesamt den gleichen Zustand erhalten würden, womit eine weitere Gliederung nicht notwending ist. Es wird Speicherplatz gespart. Zwei der Oktanten müssen weiter geteilt werden. Dabei ist ebenfalls zu erwähnen, dass lediglich die Blätter des Baumes, also Knoten, welche keine Kinder tragen, einen Zustand zugewiesen bekommen.

Die Darstellung dieses Modell als Voxel-Grid, welches im vorherigen Kapitel vorgestellt wurde, würde die Verwaltung von 512 Datenpunkte in Anspruch nehmen, da hier auch bei gleichen Zuständen alle Oktette gebildet werden. Die Speicherung als Octree hingegen erfordert lediglich die Verwaltung von 25 Knoten, so dass bereits in diesem einfachem Beispiel eine Kompression von 96% stattgefunden hat.

Die Darstellung einer Punktewolke als Voxel-Grid, beziehungsweise als optimierter Octree bietet einen weiteren essentiellen Vorteil. Durch die geeignete Wahl einer Blattgröße, welches im Falle eines Octrees der kleinsten Größe eines Oktetts entspricht, kann gleichzeitig ein Schutzbereich modelliert werden. Als Beispiel beträgt der Radius eines UAVs

25cm. Wählt man nun die Blattgröße ebenfalls mit 25cm, wird das dreidimensionale Modell stark diskretisiert. Jedoch nimmt in dieser Darstellung auch das kleinste Hindernis ein gesamtes Oktett von 25cm ein, so dass die Pfadplanung keine weiteren Schutzbereiche beachten muss. Das UAV kann somit in der Pfadplanung wie ein einfacher Punkt im Raum betrachtet werden. Dieses Operation wird auch als die Minkowski Summe bezeichnet.

Im Vergleich zum diskretisierten Arbeitsraum kann der *C-Space* direkt definiert werden. Die einzelnen Zustände können, nach Anwendung der Minkowski Summe, mit Hilfe des Octrees auf Belegtheit beziehungsweise Gültigkeit geprüft werden. Der *C-Space* kann somit als Menge aller einnehmbaren Zustände im Arbeitsraum betrachtet werden. Zur Visualisierung der Umgebung werden später lediglich die belegten Oktanten des geometrischen Modells eines Octrees dargestellt.

6.1.2 Globale Pfadplanung

Nach der Diskretisierung des Umgebungsmodells des UAVs, kann nun die Planung des globalen Pfades vorgestellt werden. Was das Schlüsselwort *Global* bedeutet, wird im späteren Verlauf des Kapitels deutlich.

Zur Planung des Pfades wird, auf Grund der hohen Dimensionen, ein zufallbasiertes Verfahren verwendet. Dies hat den Vorteil, dass nicht iterativ in einem Baum nach einer Reihen von gültigen Zustandskonfigurationen gesucht wird, sondern diese stichprobenartig bestimmt werden. Zwar werden bei einem Zufallsverfahren in der Regel nicht optimale Pfade bestimmt, jedoch kann auch bei hoch dimensionalen Konfigurationsräumen ein gültiger Pfad effizient bestimmt werden.

In der Robotik werden zur Bewegungsplanung nach dem Zufallsprinzip häufig *Rapidly-Exploring Random Trees*[33], kurz RRT, verwendet. Ein RRT ist eine Datenstruktur, welche auf Basis eines einfachen Algorithmus inkrementell gebildet wird.

```
1 def build_rrt(q_init):
2   tree.init(q_init)
3   for k = 1 to K:
4     q_rand = random_config(C)
5     extend(tree, q_rand)
6   return tree
```

Quellcode 11: RRT Konstruktions Algorithmus

Quellcode 11 zeigt den grundlegenden Algorithmus zur Bildung eines RRTs. Für K Iterationen werden beliebige Konfigurationen aus dem Konfigurationsraum C gewählt und

dem RRT hinzugefügt. Die Wurzel des Baumes bildet die Startkonfiguration q_{init}. Um nun das Finden einer Konfigurationsfolge und damit das Finden eines gültigen Pfades nachzuvollziehen, soll die Funktion *extend* genauer betrachtet werden.

```
1  def extend(tree, q):
2    q_near = nearest_neighbour(q, tree)
3    if new_config(q, q_near, q_new):
4      tree.add_node(q_new)
5      tree.add_edge(q_near, q_new)
6      if q_new = q:
7        return REACHED
8      else:
9        return ADVANCED
10   return TRAPPED
```

Quellcode 12: Funktion zum Hinzufügen einer einzelnen Konfiguration

Quellcode 12 zeigt das Verbinden der Knoten eines RRTs und damit die eigentliche Funktionalität der Pfadplanung. Wurde eine zufällige Konfiguration q_{rand} bestimmt, wird zunächst die am nächsten liegende Konfiguration q_{near} im RRT bestimmt. Die Bewertung des Abstandes ist abhängig von der Wahl des Konfigurationsraumes. Anschließend wird der Baum in Richtung der Konfiguration q_{near} und einer Metrik ϵ um die Konfiguration q_{new} erweitert, vorausgesetzt die Konfigurationen zwischen q_{near} und q_{new} sind gültig. Gültig bedeutet, dass diese nicht belegt sind.

Die Erweiterung des RRTs endet in einem von drei definierten Zuständen. Bildet die gewählte Konfiguration die übergebene Konfiguration oder allgemein die gesuchte Zielposition ab, so wurde ein Pfad gefunden und kann direkt an die Regelung übergeben werden. Handelt es sich um eine gültige Erweiterung, wird der neue Baum für die nächste Iteration verwendet. War die Erweiterung ungültig, kann der Pfad von dieser Konfiguration nicht mehr erweitert werden. Der Zustand *Reached* spielt in einer späteren Erweiterung des RRT Algorithmus eine wichtige Rolle. Hier kann er als Prüfung auf die Zielkonfiguration angesehen werden.

Abbildung 43 zeigt einen Iterationsschritt auf Basis eines zweidimensionalen Konfigurationsraumes. Der am nächsten liegende Nachbar wird hier über die euklidische Distanz bestimmt. Es zeigt sich sofort, dass neue Konfigurationen, auch wenn sie zufällig gewählt wurden, immer aus einer gültigen Konfiguration heraus bestimmt und um eine definierte Distanz erweitert wurden. Je größer man die Distanz ϵ zwischen den Konfigurationen wählt, desto mehr nicht angefasste Konfigurationen müssen auf Gültigkeit geprüft werden.

Im Falle des UAVs handelt es sich jedoch um einen 6-dimensionalen Konfigurationsraum,

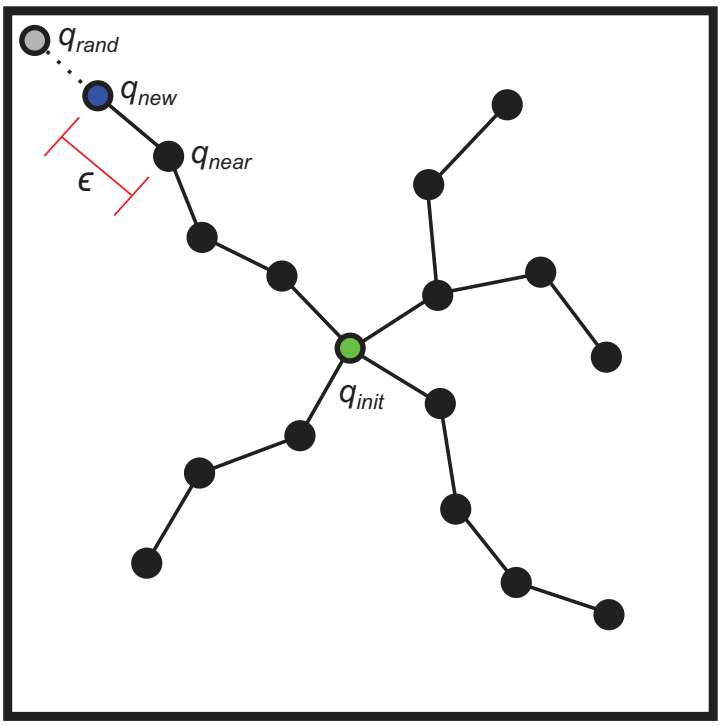

Abbildung 43: Schematische Darstellung einer RRT Iteration im zweidimensionalen Konfigurationsraum.

so dass die Metrik in einen dreidimensionalen Abstand sowie der Rotationsänderung übergeht. Wählt man nun das Erweiterungsintervall ϵ passend der Blattgröße des Octrees, welcher die diskretisierte Umgebung des UAVs abbildet, so kann die Gültigkeit der neu gewählten Konfiguration direkt an Hand der Belegtheit eines einzelnen Oktanten geprüft werden. Wie bereits im vorherigen Abschnitt erwähnt, kann die Rotation dabei außer Acht gelassen werden.

RRTs bieten einen weiteren Vorteil in der Pfadplanung. Das zufällige Finden neuer Konfigurationen kann an gewisse Anforderungen angepasst werden. In der allgemeinen Robotik werden so nicht holonomische Antriebe wie das klassische Ackermann Fahrwerk modelliert und können direkt zur Regelung des Antriebes verwendet werden. Auch für das UAV können so physikalische Bewegungseinschränkungen, welche im schlimmsten Fall zu einem Absturz führen würden, modelliert werden. Die Verwendung einer solch probabilistischen Datenstruktur wird auch als Monte-Carlo Simulation bezeichnet. Die Generierung und Bewertung zufälliger Zustände bilden die Basis einer analytischen Lösung.

Kaffner und LaValle[31] stellten 2000 ein erweitertes Verfahren zur Bestimmung einen Pfades mit Hilfe von RRTs vor. Der Algorithmus *RRTConnect* basiert auf der parallelen Bildung von zwei Bäumen wobei der erste Baum als Wurzel die Startkonfiguration und

der zweite Baum die Zielkonfiguration erhält. Die Suche besteht nun nicht mehr aus dem Finden einer Konfiguration zwischen Start- und Zielkonfiguration sondern lediglich über das Bestimmen von Schnittstellen dieser zwei RRTs.

```
1  def rrt_connect(q_init):
2    tree_a.init(q_start)
3    tree_b.init(q_goal)
4    for k = 1 to K:
5      q_rand = random_config(C)
6      if not extend(tree_a, q_rand) == Trapped:
7        if connect(tree_b, q_new) == Reached:
8          return Path(tree_a, tree_b)
9      swap(tree_a, tree_b)
10   return Failure
```

Quellcode 13: RRTConnect Algorithmus

Quellcode 13 zeigt den vollständigen Ablauf des RRTConnect Algorithmus. Wie bereits erwähnt, starten beide Bäume von der Start- beziehungsweise Zielkonfiguration. Anschließend wird eine zufällige Konfiguration gewählt und Baum A erweitert. War dies erfolgreich, wird Baum B ebenfalls gegen diesen Punkt erweitert und zwar solange bis eine Verbindung hergestellt oder auf ein Hindernis gestoßen wurde, wie die *connect* Funktion im Quellcode 14 zeigt. Konnten die Bäume verbunden werden, wurde ein Pfad gefunden. Andernfalls werden die bisherigen RRTs getauscht und der Algorithmus wiederholt von der anderen Seite aus angewandt.

```
1  def connect(tree, q):
2    while S == Advanced
3      S = extend(tree, q)
4    return S
```

Quellcode 14: Connect Funktion zum Verbinden eines RRTs mit einer Konfiguration eines zweiten RRTs

Der RRTConnect Algorithmus versucht somit ständig die Bäume zueinander hin zu erweitern, was zu deutlich kürzeren Durchlaufzeiten als in der alleinigen Bildung eines RRTs zur vollständigen Pfadfindung führt. Aus diesem Grund sowie der freien Definition der Konfigurationsräume und den zugehörigen Metriken wurde der RRTConnect Algorithmus als Pfadplanungsverfahren gewählt.

6.1.3 Path tracking

Nach dem ein Pfad gefunden wurde, gilt es diesen zu verfolgen und passende Steuerkommandos für das UAV zu genieren. Dieser Vorgang wird international als *path tracking* bezeichnet. Häufig geht das Verfolgen des Pfades auch als Teil der lokalen Navigation hervor, in dieser Arbeit ist dies jedoch auf Grund des abstrakten Systemdesigns nicht möglich. Im weiteren Verlauf werden jedoch Ansätze zur lokalen Navigation vorgestellt.

Bekannte Verfahren zur Pfadverfolgung sind der klassische PID Controller und die Potentialfelder. Bei den Potentialfeldern handelt es sich um eine Menge von Vektoren, welche stets von einem Hindernis weg- oder zur Zielposition hinzeigen. Die Vektoren werden über den Abstand sowie dem Winkel zum Ziel, beziehungsweise zum Hindernis gebildet und anschließend an Hand von Parametern wie beispielsweise dem Radius eines Hindernisses gewichtet. Durch die Überlagerung beider Potentialfelder ensteht eine Menge von Vektoren, welche für jede Position den optimalen Steuervektor aufzeigt, wie in Abbildung 44 zu sehen ist.

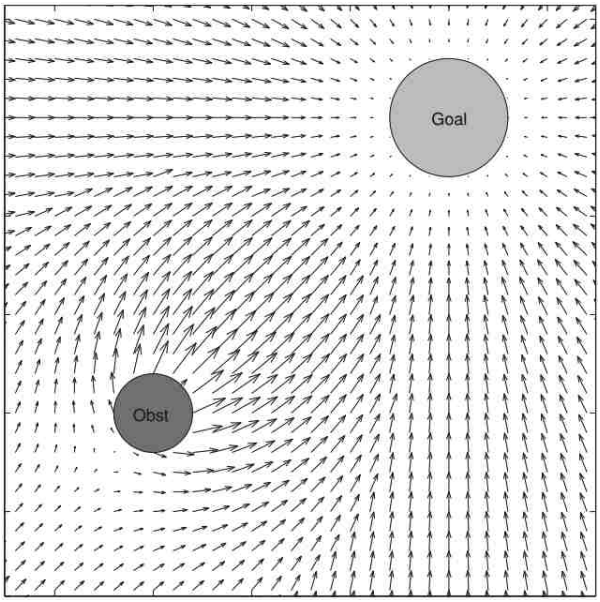

Abbildung 44: Darstellung eines überlagerten Potentialfeldes im zweidimensionalen Konfigurationsraum[23]

Der RRTConnect Algorithmus hat jedoch bereits einen Pfad unter Berücksichtigung von Hindernissen bestimmt, so dass die Berechnung des Potentialfeldes, auch in Anbetracht der hohen Dimensionen, unnötige Rechenzeit erfordern würde. Aus diesem Grund wurde für die Generierung der Steuerkommandos ein PID Regler verwendet.

Der PID Regler wird in vielen Anwendungsbereichen verwendet. Durch einen kontinuierlichen Soll-Ist Vergleich werden, je nach Differenz, Steuerbefehle generiert. Gleichung

6.1 zeigt die Differenzengleichung für einen zeit diskreten PID Regler, welche für jede Dimension des Konfigurationsraum angewandt wird. Der Term e bezeichnet dabei den Fehler zwischen der Sollposition der Konfiguration auf dem Pfad und der tatsächlichen Konfiguration des UAVs. Über die Parameter P, I und D kann die Stellgröße y dimensioniert werden, wobei je ein Parameter für den Proportional-, den Integral- und Differentialanteil ausschlaggebend ist.

$$y_n = e_n * P + \sum_{i=0}^{n} e_i * I + \frac{e_n - e_{n-1}}{T} * D \tag{6.1}$$

Der Proportionalanteil regelt die sofortige Auswirkung eines Fehlers auf den Regelkreis. Dessen alleinige Verwendung hat jedoch den Nachteil, dass eine sich eine bleibende Regelabweichung einstellt. Abhilfe schafft hier der Integralanteil. Dieser berücksichtigt den Regelfehler, welcher sich nach der Zeit aufsummiert und ist somit in der Lage, einen Regelfehler von Null einzustellen. Es ist jedoch bekannt, dass der Integralanteil in der Regelung sehr langsam ist. Hier kommt der Differentialanteil zum Einsatz. Unter Berücksichtigung des aktuellen und des letzten Regelfehlers bringt der Differentialanteil schnelle Dynamiken in den Regelkreis. Voraussetzung ist allerdings, das die Regelstrecke selbst stabil ist. Die parametrisierte Kombination dieser Anteile führt zu einer schnellen und stabilen Regelung des Systems. Die Wahl der Parameter soll hier nicht näher vorgestellt werden, da diese für die Simulation empirisch bestimmt wurden und keine wissenschaftlichen Fakten repräsentieren.

Für die Steuerung des UAVs wurden drei Regler implementiert, einen für jede translatorische Bewegung im Raum. Die Rotationskonfiguration werden vorerst nicht geregelt, da einerseits Roll- und Nickbewegungen für eine autonome Navigation unnötige Risiken aufbringen und zum anderen die Gierposition erst an der Zielposition eingestellt werden kann. Nach der Berechnung des aktuellen Translationsfehlers zwischen UAV und Pfad wird die Stellgröße über ein passendes ROS-Topic an das UAV gesendet.

Die Anwendung dieses Reglers findet periodisch mit einer Durchlaufzeit T statt. Wurde eine aktuelle Position des Pfades erreicht, sprich geht der Regelfehler gegen Null und beharrt auch auf dieser Position, kann als Sollposition die nächste Position des Pfades eingestellt werden.

6.1.4 Lokale Navigation

Im bisherigen Verlauf dieses Kapitel sind häufig Begriffe wie *globale Navigation*, *lokale Navigation* oder *dynamische Navigation* gefallen. In diesem Abschnitt des Kapitels sollend diese Begriffe differenziert und Verfahren zur lokalen Navigation vorgestellt werden. Die Implementierung des Verfahrens ist nicht Teil dieser Arbeit, da diese zu stark von der finalen Sensorik abhängig ist.

Die Planung des Pfades basiert zur Zeit auf der Planung eines kollisionsfreien Weges in der gesamten, erfassten Umgebung. Man spricht deshalb von der globalen Pfadplanung. In der Regel ist die Umgebung jedoch dynamisch, das heißt es können jederzeit Änderungen auftreten oder Hindernisse mit einer Eigenbewegung die Szenerie verändern. Da die dynamische Erfassung der gesamten erfassten Umgebung nicht möglich ist, unterscheidet in man in der dynamischen Navigation zwischen einer globalen und einer lokalen Navigation.

Die globale Navigation entspricht vollständig dem eben vorgestellten Verfahren, jedoch wird der gefundene Pfad nicht direkt an den Controller übertragen, welcher mit dem Abfliegen dieses Pfades beginnen würde, sondern eine lokale Navigation in der aktuellen Pfadumgebung durchgeführt.

Fox, Burgard und Thrun stellten bereits 1997 ein Verfahren zur Realisierung einer dynamischen lokalen Navigation vor: Den Dynamic Window Approach, kurz DWA engl. für dynamische Fenstermethode [20]. Diese Strategie zur Kollisionsvermeidung simuliert mögliche Bewegungen sowie eventuelle Dynamiken über ein kleines Zeitintervall, sodass ein Fenster von Trajektorienen entsteht. Eine anschließende Bewertung dieser Trajektorieren gibt Aufschluss darüber, welche der Steuerkommandos für die simulierte Zeit zur Beihaltung des Pfades sowie zur Hindernisvermeidung am ehesten geeignet sind. Das ausgewählte Steuerkommando wird an den Roboter übertragen, wodurch ein periodischer Regelkreis entsteht. Der soeben vorgestellte Controller für das Tracking des globalen Pfads entfällt.

Mögliche Bewertungskriterien sind der Abstand zu einem Hindernis, der Abstand zum Navigationsziel, der Abstand zum globalen Pfad oder gar erreichte Geschwindigkeiten. Wichtig zu erwähnen ist, dass zur Bewertung der Trajektorien nur das aktuelle Umfeld des UAVs genutzt wird, die lokale Umgebungskarte. Bei Sensoren wie der xTion oder einem schnellem dreidimensionalen Laserscanner, stehen diese Umgebungsinformationen jederzeit instantan zur Verfügung.

Anwendung findet der DWA auch im ROS Navigation Stack, welcher für die zweidimensionale Navigation ausgelegt ist. Das Fenster besteht dort aus einer Reihe von Geschwindigkeits- und Drehwinkelombinationen. Für das UAV werden für jede Achse im Raum Geschwindigkeiten generiert, da dieses holonome Bewegungen ausführen kann. Das weitere Verfahren bleibt jedoch identisch. Die Wahl der Simulationszeit bestimmt die Größe des Fensters der zu generierenden Kommandos sowie die Regelzeit, in der das System auf externe Einflüsse reagieren kann. Abbildung 45 zeigt beispielhaft die Bewertung eines Steuerkommandos für ein UAV im dreidimensionalen Raum. Die Bildung möglicher Steuervektoren ist hier zur Vereinfachung stark diskretisiert dargestellt. Dem Abstand zum globalen Pfad entsprechend würde der erste Steuervektor am ehesten gewählt werden. Dieser wird jedoch nun durch ein Hindernis blockiert, welches bei der Planung des Pfades nicht berücksichtigt wurde. Der Algorithmus wählt daraufhin Steuervektor 15 als die zunächst auszuführende Aktion, da diese zwar etwas weiter vom Pfad

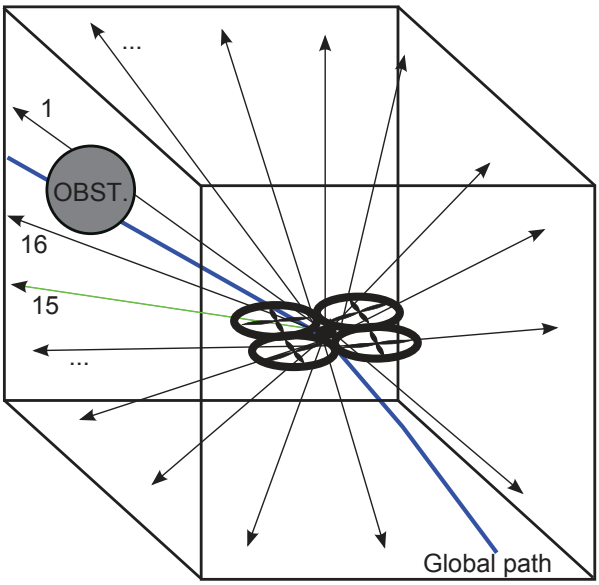

Abbildung 45: Stark diskretisierte Darstellung des DWA Verfahrens im dreidimensionalen Raum

entfernt liegt, jedoch das Hindernis ausreichend umfliegt. Die Namensgebung *dynamic window* beruht auf dem periodischen Verhalten des Verfahrens. Durch die wiederholte Ausführung der Überprüfung und dem Finden eines lokalen Pfades, verschiebt sich das Fenster, beziehungsweise der Würfel im Dreidimensionalen, stets mit dem Roboter selbst.

Erfahrungsgemäß erzielte das Verfahren im Zweidimensionalen stets gute Ergebnisse. Nach Angaben der Entwickler soll bereits 1997 eine robuste dynamische Navigation für einen Roboter mit Geschwindigkeiten von bis zu einem Meter pro Sekunde möglich gewesen sein. Wird für den finalen Aufbau des UAVs dreidimensionale Sensorik verbaut, welche die lokale Umgebung nahezu instantan liefert, so sollte das DWA Verfahren eingesetzt werden.

6.1.5 MoveIt! Framework

Nachdem nun die Datenverwaltung, die Pfadplanung und das Tracken eines gefundenen Pfades vorgestellt wurden, sollen in diesem Abschnitt die Schnittstellen des MoveIt! Frameworks[25] präsentiert werden.

MoveIt!, ursprünglich von Willow Garage ins Leben gerufen, ist eine Sammlung von Algorithmen und Schnittstellen für mobile Manipulation, Pfadplanung, 3D Erkennung, Kinematiken, Steuerung und Navigation. Das Framework wurde erstmals im Jahre 2013

vorgestellt, hat sich seit dem stetig fortentwickelt und unterstützt nun eine Vielzahl von Robotern, Sensoren und Aktoren. Zudem ist MoveIt! ein Teil von ROS, was die Integration in das Gesamtsystem, welches ebenfalls auf ROS basiert, erheblich vereinfacht.

Abbildung 46 zeigt den grundlegenden Aufbau des Frameworks. Von der Hardware sowie der Sensorik wird völlig abstrahiert. Die Abbildung zeigt ebenfalls über welche standardisierten ROS Botschaften die Kommunikation zwischen dem Navigations Framework und dem UAV stattfindet.

Wie bereits zu Beginn des Kapitels angedeutet, setzt auch MoveIt! eine dreidimensionale Abtastung der Umgebung sowie die aktuelle Position in dieser voraus. Über den globalen Parameterserver werden Informationen über die Geometrie des UAVs sowie verschiedene Parameter zur Steuerung und Pfadplanung abgerufen. Über diese Parameter können Bewegungseinschränkungen sowie der Konfigurationsraum definiert werden.

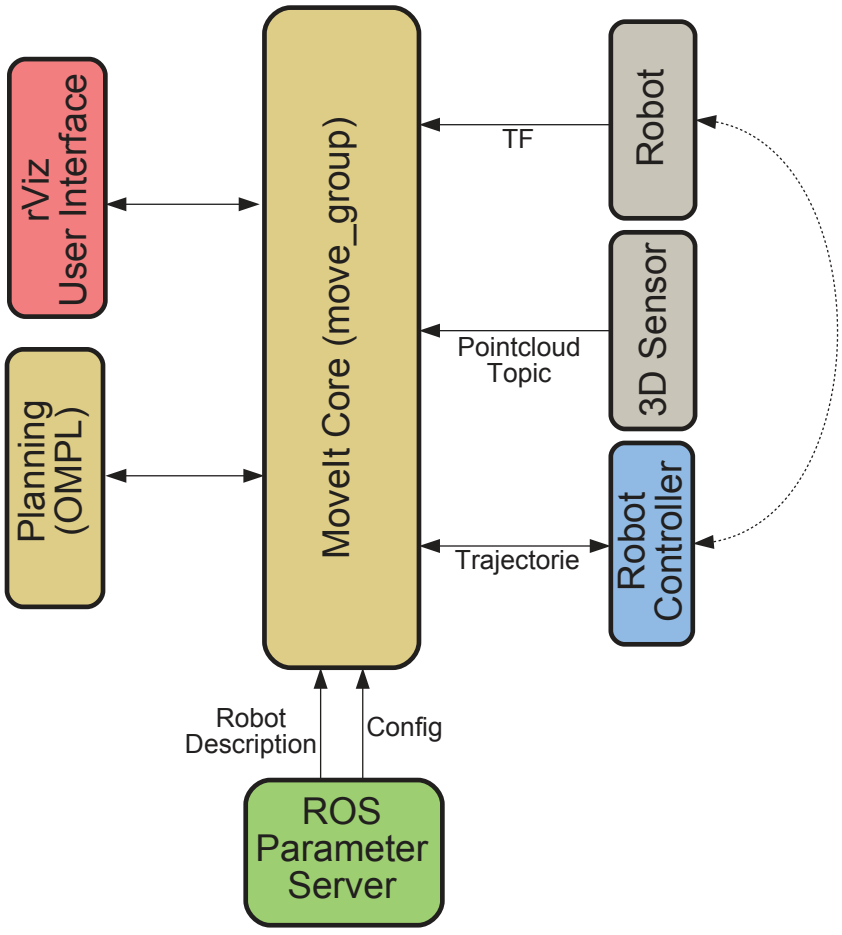

Abbildung 46: MoveIt! Framework

Über die Visualisierungskomponente rViz, ebenfalls Teil von ROS, wird auch die Verarbeitung der Navigationsdaten dargestellt. Weiter sieht man die Schnittstelle zur Open

Motion Planning Library, kurz OMPL [54], welche die Implementierungen der Planungsalgorithmik liefert. Über die OMPL wird auch eine Implementierung des RRTConnect Algorithmus zur Verfügung gestellt. Zuletzt zeigt das Framework ein Interface für die Steuerungskomponente. Diese kann als Übersetzer zwischen geplanter Trajektorie und den eigentlichen Steuerbefehlen an das UAV angesehen werden. Die Implementierung dieses Interfaces wurde im vorherigen Abschnitt vorgestellt und ist somit für das Tracking beziehungsweise für die lokale Navigation verantwortlich.

Der generelle Arbeitsablauf der MoveIt! Navigation sieht somit wie folgt aus. Ankommende Sensordaten werden als Octree gespeichert und zusammen mit der zugehörigen Position registriert. Wird nun eine Zielposition übergeben, beginnt der Planer mit der Planung des Pfades unter Berücksichtigung der aktuellen Umgebungsinformationen. Wurde ein Pfad gefunden, wird dieser an den Controller übergeben, welcher daraufhin mit der Regelung des UAVs zur Einhaltung dieses Pfades beginnt.

Das MoveIt! Framework kann ebenfalls so konfiguriert werden, das bei Umgebungsänderungen die Regelung unterbrochen und der Pfad neu geplant wird. In wie fern dies einer dynamischen Navigation, sprich der vollständigen, kontinuierlichen Berücksichtigung der gesamten Arbeitsumgebung, entspricht, konnte Hilfe der Simulationsumgebung nicht getestet werden.

6.2 Simulationsumgebung

Zur Implementierung und Evaluierung der Navigationsalgorithmen wurde eine einfache Simulationsumgebung mit Hilfe der Software Gazebo[22] aufgesetzt. Gazebo, 2002 an der Universität in Süd-Kalifornien ins Leben gerufen, ist ein weit verbreiteter OpenSource Simulator, welcher durch einfache XML Beschreibungen konfiguriert wird. Die Schnittstellen der Simulation sind kompatibel zu ROS, sodass zwischen der Anwendung von Software auf dem UAV oder in der Simulation keine hardwarespezifischen Änderungen von Nöten sind.

Die für diese Arbeit verwendete Simulationsumgebung weist einen sehr einfachen Charakter auf. Simuliert wurde lediglich ein Quadrotor in einer leeren Welt unter normalen physischen Einwirkungen. Somit konnten grundlegende Flugbewegungen ausgeführt, die Bewegungsinformationen des UAVs bestimmt und das Abfliegen eines gefundenen Pfades evaluiert werden. Die Sensorinformationen selbst wurden nicht simuliert sondern dem Navigationsframework direkt aus zuvor aufgenommen Daten eingespielt, was den Vorteil bietet auf realen Messdaten zu arbeiten.

Das Modell des Quadrotors wurde im Jahr 2012 von Meyer u.a.[38] modelliert und speziell für die Zusammenarbeit von ROS und Gazebo entwickelt. Mit Hilfe dieses Modells wurden unter anderem auch die Anwendung des Hector-SLAM Verfahrens auf Flugrobotern getestet. Abbildung 47 zeigt das simulierte UAV in einer auf Octrees basierten

Darstellung der Umgebung. Nach der Planung des Pfades nimmt der Simulator die Steuerbefehle entgegen und beginnt mit dem Flug zur angegebenen Zielposition.

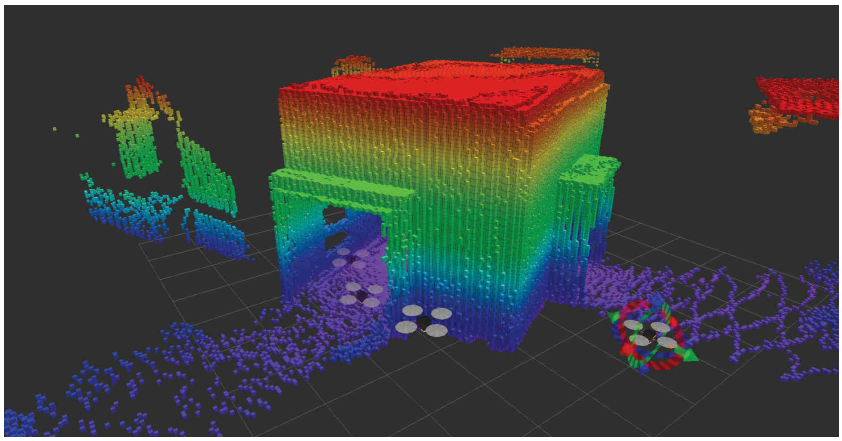

Abbildung 47: Simulation des UAVs zur Evaluierung der Pfadplanung

Da zum Zeitpunkt der Simulation kein Verfahren zur Kartierung zur Verfügung stand und die Anwendung der Pfadplanung ebenfalls in großflächigen Szenarien getestet werden sollte, wurde ein Verfahren zur Registrierung von Teilaufnahmen entwickelt, welches nachfolgend vorgestellt werden soll.

6.2.1 ICP Algorithmus

Einen einfachen Mechanismus zur Registrierung von dreidimensionalen Umgebungsmodellen, wie sie die Sensorik liefern wird, ist der *Iterative closest point* Algorithmus, kurz ICP.

Der Algorithmus ermittelt Transformationen, welche iterativ versuchen den Abstand zwischen den Punkten zweier Datensätze zu minimieren. Dazu wird zu einem bestimmten Punkt der am nächsten liegende Punkt aus dem anderen Datensatz bestimmt und die Distanz berechnet. Diese Distanzen werden für alle Punktepaare aufsummiert. Durch die Anpassung der Transformationsparameter wird anschließend versucht diese Summe zu minimieren. Dieser Vorgang wird solange wiederholt bis ein Minimum oder die maximale Anzahl an Iterationen erreicht ist. Über die Angabe eines maximalen Fehlers, können fehlerhaft ermittelte Punktkorrespondenzen ausgeschlossen werden.

Ein Effekt der stetigen Registrierung eingehender Daten ist die Bestimmung aller Transformationen zwischen diesen, welche ebenfalls Aufschluss über die aktuelle Position bei der Aufnahme der Daten liefert. Dieser Effekt wird auch als Simultanous Localization and Mapping (kurz. SLAM) bezeichnet. Jedoch gilt auch hier, dass die alleinige Aufsummierung einer fehlerbehafteten Information schnell zu einer falschen Lokalisierung

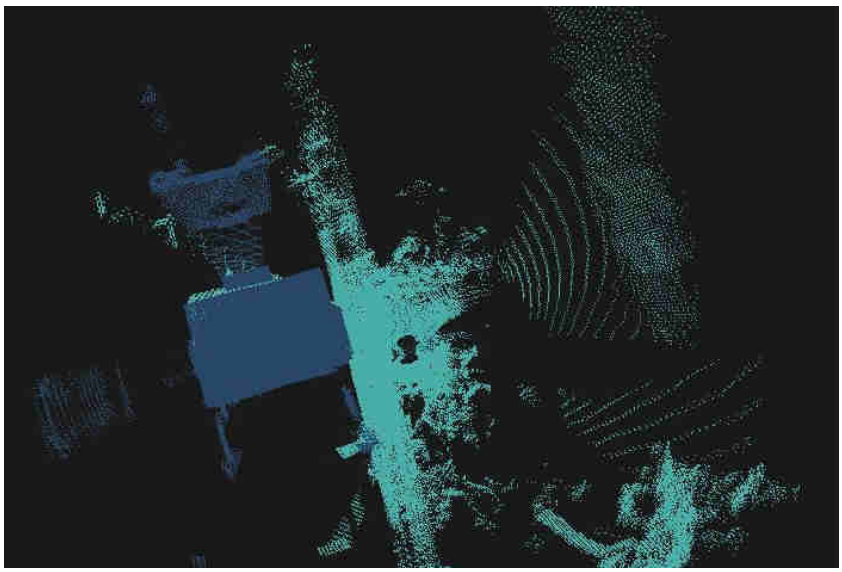

Abbildung 48: Registrierung zweier dreidimensionaler Datensätze aus dem Pisa Datensatz mit Hilfe des ICP Algorithmus.

führt.

Abbildung 48 zeigt die Registrierung zweier realer Punktewolken. Das Ergebnis der Registrierung kann als Octree dargestellt und zur Navigation verwendet werden. Abbildung 49 hingegen zeigt das resultierende Gesamtmodell der Registrierung des kompletten Pisa Datensatzes, welcher mit dem 3D Scanner aufgenommen wurde.

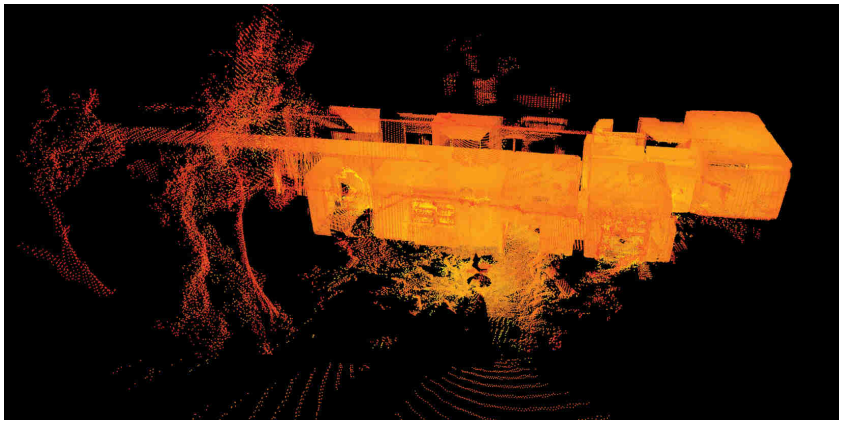

Abbildung 49: Registrierung des vollständigen Pisa Datensatzes.

7 Abschlussbetrachtung

Dieses Kapitel präsentiert das Resümee dieser Arbeit. Es wird gezeigt was erarbeitet wurde und welche Problematiken dabei entstanden sind. Ferner sollen Aufgaben und Ideen definiert werden, welche für zukünftige Ausarbeitungen interessant sind.

7.1 Zusammenfassung

In der zugrunde liegenden Arbeit wurde ein erster Ansatz zur autonomen Navigation eines UAVs entwickelt und vorgestellt. Es wurden verschiedene Sensorgruppen sowie zugehörige Algorithmen präsentiert, erläutert und hinsichtlich der Anwendbarkeit analysiert. Dabei stellte sich heraus, dass visuelle Sensorik zur Abtastung der Umgebung keineswegs einen sicheren, autonomen Flug ermöglicht. Zwar ist es möglich durch Kombination mit weiterer Inertialsensorik eine Schätzung der Eigenbewegung durchzuführen, jedoch können Abtastungen der Umgebung nicht als „verlässliche“ Information interpretiert werden. Es kann jederzeit vorkommen, dass bei mono-visuellen Verfahren Features nicht detektiert oder allgemein Korrespondenzen nicht hergestellt werden. Speziell in USAR Missionen erschweren eventuelle Sicht- oder Beleuchtungseinschränkungen die Interpretation der Kameradaten. Aus diesen Gründen wurde zur Umgebungswahrnehmung ein Sensor vorgeschlagen, welcher direkte Distanzmessungen durchführt. Dies kann ein infrarot-basierter, dreidimensionaler Sensor wie die xTion oder die vorgestellte Erweiterung eines klassischen Laserscanners sein. Die Wahl des finalen Sensorsystems richtet sich nach den Anforderungen an das Gesamtgewicht sowie die benötigte Auflösung der Umgebungsmessung.

Durch die Nutzung allgemein gültiger Schnittellen des Roboter Betriebssystems ROS konnte, unabhängig von der Wahl des endgültigen Sensorsystems, ein erster Ansatz zur Navigation in der dreidimensionalen Umgebung entwickelt werden. Dabei wurde auf Basis des MoveIt! Frameworks ein Pfadplanungsverfahren vorgestellt, welches den Pfad durch Nutzung zufällig bestimmter Zustände ermittelt und auf Gültigkeit überprüft. Die Überprüfung der Gültigkeit findet dabei auf einer diskretisierten Darstellung der abgetasteten Umgebung statt, der Octree Darstellung. Diesbezüglich wurden ebenfalls verschiedene Verfahren zum Abfliegen des gefundenen Pfades analysiert sowie ein einfacher abstrakter Regler implementiert.

Auf Basis des hardwareunabhängigen Navigationsverfahren konnte eine Simulationsumgebung eingerichtet werden, welche die Pfadplanung und -regelung mit Hilfe eingehender Punktewolken, aufgenommen in realen Einsatzszenarien, testet.

Neben der Entwicklung eines allgemeinen Konzeptes zur dreidimensionalen Navigation eines UAVs in Umgebungen ohne GPS Empfang, wurde der vorliegende Prototyp der Firma Ascending Technologies für die Kommunikation mit ROS vorbereitet. Ne-

ben einem Treiber, welcher grundlegende Steuerbefehle und Sensorinformationen über allgemeine Botschaften austauscht, wurden Algorithmen entwickelt, welche die Eigenbewegung des UAVs an Hand der Inertialsensorik schätzen. So ist es möglich das UAV über ROS Komponenten zu steuern und Stautsinformationen abzufragen. Entwickelte Plugins ermöglichen eine Steuerung des UAVs über einen Kartenausschnitt der aktuellen Umgebung. Die Pelican kommuniziert über das sogenannte ACI Interface, sodass die finale Umsetzung des Ascending UAVs im TRADR Projekt ebenfalls über diesen Treiber mit ROS kommunizieren kann.

7.2 Ausblick

Im folgenden sollen Arbeitspakete für weitere Ausarbeitungen, mit absteigender Wichtigkeit, vorgeschlagen werden.

Wahl der finalen Sensorik

Zunächst gilt es die finale Sensorik für das UAV, welches autonom im Innenbereich navigieren soll, festzulegen. Je nach weiterer Verwendung der Daten sollte hier, wie bereits vorgeschlagen, ein RGB-D Sensor oder ein hochwertiges laser-basiertes Verfahren gewählt werden. Daraufhin sollte der Sensor in das Gesamtsystem integriert und die Kommunikation mit dem ROS System überprüft werden. Abschließend ist ein geeignetes Lokalisierungs- oder gar Kartierungsverfahren zu ermitteln. Unter Zuhilfenahme der Odometrie kann mittels des vorgestellten ICP Verfahren die Position korrigiert und eine einfache Karte registriert werden. Es bietet sich jedoch an, passende Verfahren aus dem Stand der Technik[14][42] zu evaluieren und gegebenenfalls zu integrieren.

Evaluierung der dynamischen Navigation

Nach der Wahl eines Sensors und dessen Integration in das Gesamtsystem, sollte zunächst in einer abgesicherten, kontrollierten Umgebung die dynamische Navigation evaluiert werden. Wie bereits in Kapitel 6 erwähnt, bietet das MoveIt! Framework die Option, einen Pfad bei Änderungen der Arbeitsumgebung neu zu berechnen. Ob die Dynamik den Anforderungen entspricht, muss in Feldversuchen bestimmt werden. Reicht die Neuberechnung des Pfades nicht aus, kann das lokale Kartensystem verwendet werden, welches unter dem Namen *Dynamic Window (Cube) Approach* präsentiert wurde. Dieses muss auf den gewählten Sensor abgestimmt werden. Ferner entfällt die Notwendigkeit an die Regler-Komponente.

Optimierung des Reglers

Stellt sich die die dynamische Navigation des MoveIt! Frameworks als ausreichend heraus, so ist der implementierte Regler zu optimieren. In dieser Arbeit wurde lediglich ein simuliertes UAV angesteuert. Die Parameter wurden empirisch bestimmt und werden mit großer Wahrscheinlichkeit nicht den Anforderungen an einen ruhigen, realen Flug gerecht. Auch dies sollte in einer kontrollierten, sicheren Testumgebung durchgeführt werden.

8 Anhang

Dieses Kapitel bildet den Anhang dieser Arbeit. Es werden Informationen zur Planung, zur Organisation und Statistiken zur Durchführung der Arbeit gezeigt.

8.1 Planung der Arbeit

Nach ersten Überlegungen sollte vorerst eine Schätzung der Position an Hand der internen Positionssensorik (IMU, Ultraschall) realisiert werden. Anschließend können weitere Entwicklungen zur Navigation folgen. Vorab wurden folgende Meilensteine geplant, welche Auskunft über den Arbeitsverlauf und auch erste Implementierungsideen bieten.

Meilenstein 1: Aufnahme von Testdaten

Ziel dieses Meilensteins ist es, ein Testsystem zu entwickeln, mit dessen Hilfe genügend Testdaten zur Evaluation und Implementierung der Algorithmen gesammelt werden können. Dieses Testsystem soll möglichst viele verschiedene Sensoren enthalten beispielsweise zwei Kameras, Laserscanner, Ultraschall und ein Smartphone. Durch ein Smartphone ist eine ganze Reihe an Positionssensorik (GPS, IMU, Magneotemeter, ...) gegeben. Anschließend können optische Mono- und Stereoverfahren, inklusive aktueller Bewegungsinformationen evaluiert werden. Die Testdaten werden als Bag File zur Verfügung gestellt und simulieren somit eine funktionsfähige Drohne mit gängiger Sensorik. Die zu implementierenden / evaluierenden Algorithmen müssen nicht alle gesammelten Sensoren verwenden. Nicht verwendete Sensorinformationen dienen als Referenz (Ground truth), wie beispielsweise die GPS Daten. Die Testdaten sollen möglichst viele reale Situationen abbilden beispielsweise Flug durch einen Korridor, Raum, aber auch Kollisionskurse mit Türen, Wänden, Fenstern, Menschen und typischen Hindernissen. Jeder Datensatz wird mindestens zweimal aufgenommen, wobei genau definierte Unterschiede in den Datensätzen vorkommen (Persistenz).

Update: Das Testsystem wird lediglich zur Aufnahmen der Datensätze, welche Stereo / RGB-D Aufnahmen enthalten sollen, genutzt. Für die üblichen Testreihen wird auf die Drohnen der Firma Parrot (1.0 und 2.0) zurückgegriffen, da hier ebenfalls gängige Sensorik, mit Ausnahme eines direkten Tiefenbilds, zur Verfügung gestellt wird. Der Vorteil liegt darin, dass die Testdaten unter realistischen Flugbedingungen aufgenommen werden. Ein weitere Vorteil ist die Ergänzung der Testdaten um reale Steuerbefehle, welche an die Drohne gesendet werden. Diese Steuerbefehle werden häufig zur einer initialen Schätzung des Bewegungsmodells verwendet.

Meilenstein 2: Grundlegende Steuerung

Der vorhandene ROS Knoten für die Nifti-Drohne ist in der Lage, GPS Wegpunkte zu verarbeiten und die Sensorinformationen zur Verfügung zu stellen. Diese Funktionalität soll auch auf den Ascending Drohnen gegeben sein. Zu dem ist ein manuelles Setzen der Geschwindigkeiten erforderlich damit die UAVs über ROS gesteuert werden können. Dabei sollen nur standardisierte ROS Botschaften (Twist, IMU) verwendet werden, so dass die entwickelte Software möglichst hardwareunabhängig ist. Auch die Sensorinformationen sollen über Standardbotschaften gesendet werden. Somit kann entwickelte Software einerseits in der Simulation und anschließend auf dem finalen System getestet werden.

Meilenstein 3: Odometrie

Voraussetzung: M1 bzw. M2
Die Sensorwerte der Drohne sollen fusioniert und gefiltert werden. Anschließend soll daraus eine Transformation berechnet und der ROS Umgebung zur Verfügung gestellt werden. Über diese Transformation kann das Bewegungsmodell der Drohne geschätzt werden. Voraussetzung für Kartierung, Lokalisierung und Navigation.

Meilenstein 4: SLAM

Voraussetzung: M3
Ziel von Meilenstein 3 ist die Erstellung einer groben Karte der Umgebung. Diese Karte dient nicht zur detaillierten Abbildung der Umgebung, sondern lediglich zur Navigation und Lokalisierung (Trennung von menschlicher und maschinellen Karte). Dazu kann beispielsweise GraphSlam verwendet werden. Das Verfahren wurde schon in 2D implementiert und liefert eine effiziente Berechnung der Umgebung sowie eine akkurate Schätzung der Position. Das Verfahren basiert auf einer Matrix, deren Dimension der Größe Größe der Karte entspricht, zusätzlich einer Zeile / Spalte für die Positionsvorhersage. Jeder Sensor Wert oder jede Bewegungsinformation ist eine Addition in dem entsprechenden Matrixelement. Um die Berechnungen performant zu halten, soll die Karte als Octo-Map gespeichert werden. Die Punkte in der Cloud haben dann bspw. eine Größe von 5cm. Das ist zur Lokalisierung und Navigation völlig ausreichend und reduziert eine 5 Meter Umgebung aus eine 100x100x100 Cloud.

Wenn auf das PTAM Verfahren gesetzt wird, ist die Karte bereits gegeben. Wird Stereo Vision genutzt muss ein Cloud Matching zur Bildung der Karte genutzt werden. Laut DOW WP1 kann bereits eine Karte der Umgebung zur Verfügung stehen.

Meilenstein 5: Lokalisierung

Voraussetzung: M3
Dieser Meilenstein dient zur Lokalisierung in einer bekannten Umgebung. Die vorhandene Karte muss gefiltert werden um auch in einer OctTree Darstellung vorzuliegen. Anschließend soll ein 6D Partikelfilter realisiert werden, der an Hand kontinuierlicher Messung die Position des Roboters in der Umgebung schätzt.

Meilenstein 6: Pfadplanung (Global)

Voraussetzung: M4 oder M5, wenn Karte vorhanden
Ziel dieses Meilensteins ist es, einen Pfad in der Umgebung zu planen. Dazu soll ein Suchalgorithmus auf Basis einer Baumstruktur verwendet werden, da auf Grund der hohen Dimensionen, welche durch die Bewegung im dreidimensionalen Raum entstehen, hohe Rechen- und Suchzeiten entstehen können. Das Finden eines optimalen Pfades ist vorerst nicht von hoher Priorität. Der globale Pfad dient zur Orientierung, beachtet jedoch keine dynamischen Hindernisse.

Meilenstein 7: Navigation

Voraussetzung: M6
Dieser Meilenstein beschreibt die Fähigkeit zum Abfliegen eines gegeben Pfads. Dabei müssen auch dynamische Hindernisse berücksichtigt werden, da hier die Steuerbefehle an die Drohne gesendet werden sollen. Es lassen sich folgende Teilziele definieren:

- Reglerentwurf zur Weitergabe der Bewegungsinformationen an die Motoren.
- Anfliegen einer relativen Position. Bspw. 10 Meter vorwärts
- Kollisionsvermeidung: Ein Ansatz, im Vergleich zum Navigation-Stack, wäre die Erstellung einer lokalen Karte aus aktuellen Sensorinformationen. Hier werden dynamische Änderungen berücksichtigt. Anschließend kann ein Dynamic Cube Verfahren entwickelt werden. Das Verfahren basiert auf dem bekannten Dynamic Window Verfahren, berücksichtigt jedoch auch die Bewegungen im 3D Raum. Alternativ kann ein primitiver Ansatz realisiert werden. Die Drohne wird frontal mit einer LED (sichtbar oder IR) ausgestattet. Diese LED kann auch batteriebetrieben als Modul angeklebt werden. Nähert sich die Drohne einem Hindernis, wird sich auf dem Kamerabild ein Lichtkreis bilden, welcher in Relation zur Distanz immer kleiner und heller werden wird. Diese Lichtreflexionen lassen sich im Bild detektieren und durch einen einfachen Regler kann der Lichtkegel umflogen werden.

Die Planung der Meilensteine bietet die Option, dass SLAM Verfahren optional zu entwickeln, sofern eine 3D Karte der Umgebung vorliegt. In wie weit die Implementierung von SLAM notwendig ist, hängt vom gewählten Verfahren ab (siehe Kapitel 2) ab. Es ist jedoch davon auszugehen, dass M4 oder M5 zu realisieren ist.

Meilenstein 8: Tracking

Voraussetzung: M7.1, M7.2 Auf Grund von Informationen der Feuerwehr, wäre es im Szenario hilfreich, wenn das UAV der Hundestaffel folgen kann. Ziel ist es, eine optische Verfolgung auf Grund charakteristischer Bildmerkmale (Farbe der Weste eines Hunds) zu realisieren.

8.2 Aufwandsanalyse

Dieser Abschnitt gibt Aufschluss, was zum Ende der Arbeit erreicht wurde und wie viele Stunden dafür aufgebracht wurden. Tabelle 4 zeigt die prozentuale Umsetzung der zuvor geplanten Meilensteine zum Ende der Bearbeitungszeit. Lokalisierung und Kollisionsvermeidung wurden ansatzweise umgesetzt, da abstrakte Verfahren vorgestellt wurden, welche speziell für die finale Sensorgruppe gewählt und implementiert werden müssen.

Meilenstein	Beschreibung	Fortschritt	Notwendigkeit
M1	Aufnahme von Testdaten	100 %	Pflicht
M2	Grundlegende Steuerung	100 %	Pflicht
M3	Odometrie	100 %	Pflicht
M4	SLAM	20 %	Optional (verfahrensabhängig)
M5	Lokalisierung	30 %	Optional (verfahrensabhängig)
M6	Pfadplanung (global)	100 %	Pflicht
M7.1	Regelung	100 %	Pflicht
M7.2	Relative Bewegung	100 %	Pflicht
M7.3	Kollisionsvermeidung	30 %	Pflicht
M8	Tracking	0 %	Optional

Tabelle 4: Abschlussstand der geplanten Meilensteine

8.2.1 Quellcode

Tabelle 5 zeigt die Anzahl geschriebener Quellcodezeilen für eigens implementierte ROS Pakete. Betrachtet man nun im Vergleich die Anzahl wiederverwendeter ROS Pakete,

wie Tabelle 6 zeigt, wird deutlich wie „wenig“ Quellcode geschrieben werden muss und warum Konfigurationen zur Verknüpfung von Paketen benutzt werden.

Sprache	Dateien	Leerzeilen	Kommentarzeilen	Codezeilen
C/C++	9	2344	1891	9131
C/C++ Header	44	963	3198	456
Python	6	53	24	154
CMake	35	265	644	627
XML	28	139	104	506

Tabelle 5: Entwickelter Quellcode. XML Dateien sind notwendig für Paketkonfigurationen und -verknüpfungen.

Herkunkft	Anzahl Pakete
Eigene Implementierung	17
ROS Community	8
ROS Betriebssystem	>> 63

Tabelle 6: Anzahl verwendeter ROS Pakete. Die Anzahl der von ROS verwendeten Pakete lässt sich auf Grund rekursiver Abhängigkeiten nur schwer ermitteln. Hier beispielhaft angegeben durch das *sfm* Paket, welches 63 Pakete des Systems nutzt.

8.2.2 Aufgebrachte Stunden

Abbildung 50 zeigt den Verlauf der geleisteten Stunden. Das Prüfungsamt sieht für die Durchführung der Masterarbeit 30 Leistungspunkte voraus. Dies entspricht einer gesamten Arbeitszeit von 900 Stunden. Auf Grund von Nebenbeschäftigungen wurden für die Masterarbeit eine wöchentliche Arbeitszeit von ca. 20 Stunden eingeplant. Dies entspricht bei einer Durchführungsdauer von 50 Wochen 1000 geplanten Stunden. Ebenfalls sichtbar sind die Phasen, zu welcher Zeit welche Thematik behandelt wurden sowie die Begründungen von Ausreißern im Stundenverlauf.

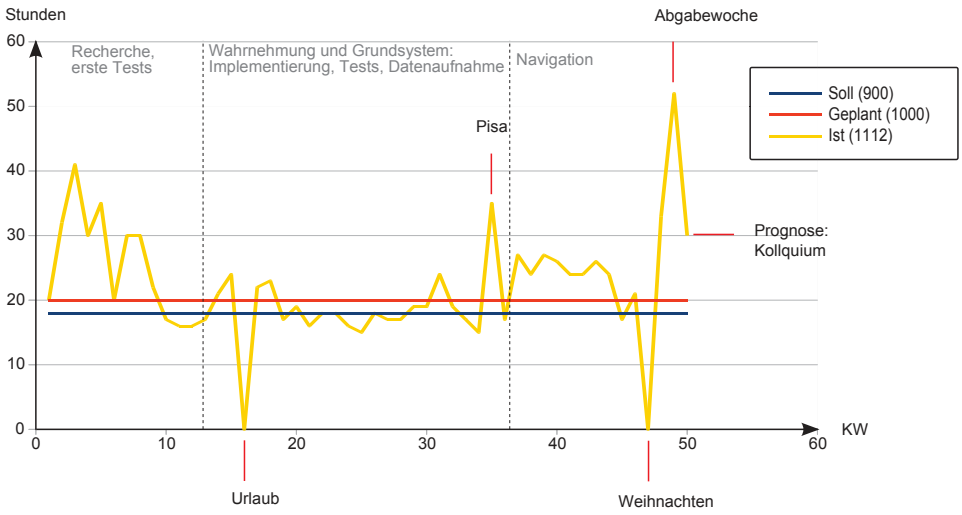

Abbildung 50: Aufstellung der geleisteten Stunden

8.3 Vergleich: Sequentielle zu parallele Implementierung

In diesem Abschnitt soll der Vergleich zwischen der normalen Implementierung und einer parallelen Implementierung vorgestellt werden.

Zum Vergleich wurden in jeder Implementierungsvariante ein bereits entzerrter Datensatz mit einer Auflösung von 1344x372 Pixeln verwendet. Die Berechnungsdauer ist stark abhängig von der Anzahl der gefunden Features und der Auflösung der Eingabebilder. Zum Vergleich wurde der OpticalFlow Feature Matcher benutzt, welcher auf den Testdaten 4091 Features detektiert.

Verfahren	Berechnungsdauer sequentell	Berechnungsdauer parallel
Optical Flow Matching	635 ms	635 ms
Freak Feature Matching	3 s	1.83 s
Lin. Triangulation	16 ms	7ms

Tabelle 7: Vergleich: Sequentielle zu paralleler Implementierung des *Structure from Motion* Frameworks

Die Funktionen der OpenCV calcOpticalFlow(...) und radiusSearch(...) sind nicht parallelisierbar, da es nur ein einzelner Funktionsaufruf ist. Diese beiden Funktionen benötigen jedoch, mit dem oben genannten Datensatz, eine Rechenzeit von 0,21s + 0,40s. Somit ist eine Optimierung des optischen Flusses nicht möglich. Das Freak Feature Matching lässt sich auf Grund der vorherigen Featuresuche und Deskriptorberechnung, welche unabhängig in beiden Teilbildern durchgeführt werden muss, sehr gut parallelisieren. Auch die Triangulation lässt sich durch parallelisierte Schleifendurchläufe auf festen Indexbereichen optimieren.

Literatur

[1] Alcantarilla, P. F. ; Bartoli, A. ; Davison, A. J.: KAZE Features. In: *Eur. Conf. on Computer Vision (ECCV)*, 2012

[2] Alcantarilla, P. F. ; Nuevo, J. ; Bartoli, A.: Fast Explicit Diffusion for Accelerated Features in Nonlinear Scale Spaces. In: *British Machine Vision Conf. (BMVC)*, 2013

[3] *AscTec ACI Dokumentation*. `http://wiki.asctec.de/display/AR/List+of+all+predefined+variables%2C+commands+and+parameters`. – [Online; Stand 15. Juli 2014]

[4] *Produktseite: AscTec AutoPilot*. `http://wiki.asctec.de/display/AR/AscTec+AutoPilot`. – [Online; Stand 09. Juli 2014]

[5] *Produktseite: Asctec Pelican*. `http://www.asctec.de/uav-applications/research/products/asctec-pelican`. – [Online; Stand 09. Juli 2014]

[6] *Produktseite: Asus Xtion*. `https://www.asus.com/de/Multimedia/Xtion/`. – [Online; Stand 15. Juli 2014]

[7] Bay, Herbert ; Ess, Andreas ; Tuytelaars, Tinne ; Van Gool, Luc: Speeded-Up Robust Features (SURF). In: *Comput. Vis. Image Underst.* 110 (2008), Juni, Nr. 3, 346–359. `http://dx.doi.org/10.1016/j.cviu.2007.09.014`. – DOI 10.1016/j.cviu.2007.09.014. – ISSN 1077–3142

[8] Bills, C. ; Chen, J. ; Saxena, A: Autonomous MAV flight in indoor environments using single image perspective cues. In: *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, 2011. – ISSN 1050–4729, S. 5776–5783

[9] Bouguet, Jean-Yves: *Camera Calibration Toolbox for Matlab*. `http://www.vision.caltech.edu/bouguetj/calib_doc/htmls/example5.html`. Version: 2008. – [Online; Stand 16. Juli 2014]

[10] Brown, D. C.: Decentering Distortion of Lenses. In: *Photometric Engineering* 32 (1966), Nr. 3, S. 444–462

[11] Campbell, J. ; Sukthankar, R. ; Nourbakhsh, I ; Pahwa, A: A Robust Visual Odometry and Precipice Detection System Using Consumer-grade Monocular Vision. In: *Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on*, 2005, S. 3421–3427

[12] Dijkshoorn, Nick: *Simultaneous localization and mapping with the AR. Drone*, Master's thesis, Universiteit van Amsterdam, Diss., 2012

[13] Droeschel, David ; Stückler, Jörg ; Behnke, Sven: Local Multiresolution Representation for 6D Motion Estimation and Mapping with a Continuously Rotating 3D Laser Scanner. In: *Proc. of IEEE Int. Conf. on Robotics and Automation (ICRA)*, 2014

[14] Endres, F. ; Hess, J. ; Engelhard, N. ; Sturm, J. ; Cremers, D. ; Burgard, W.: An Evaluation of the RGB-D SLAM System. In: *Proc. of the IEEE Int. Conf. on Robotics and Automation (ICRA)*. St. Paul, MA, USA, May 2012

[15] Engel, J. ; Sturm, J. ; Cremers, D.: Camera-based navigation of a low-cost quadrocopter. In: *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, 2012. – ISSN 2153–0858, S. 2815–2821

[16] Engel, J. ; Sturm, J. ; Cremers, D.: Semi-dense Visual Odometry for a Monocular Camera. In: *Computer Vision (ICCV), 2013 IEEE International Conference on*, 2013. – ISSN 1550–5499, S. 1449–1456

[17] Fischler, Martin A. ; Bolles, Robert C.: Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography. In: *Commun. ACM* 24 (1981), Juni, Nr. 6, 381–395. `http://dx.doi.org/10.1145/358669.358692`. – DOI 10.1145/358669.358692. – ISSN 0001–0782

[18] Folkesson, John ; Christensen, Henrik: Sift based graphical slam on a packbot. In: *Field and Service Robotics* Springer, 2008, S. 317–328

[19] Forster, Christian ; Pizzoli, Matia ; Scaramuzza, Davide: SVO: Fast Semi-Direct Monocular Visual Odometry. In: *Proc. IEEE Intl. Conf. on Robotics and Automation* (2014)

[20] Fox, D. ; Burgard, W. ; Thrun, S.: The dynamic window approach to collision avoidance. In: *Robotics Automation Magazine, IEEE* 4 (1997), Mar, Nr. 1, S. 23–33. `http://dx.doi.org/10.1109/100.580977`. – DOI 10.1109/100.580977. – ISSN 1070–9932

[21] Gäbel, M ; Krüger, T ; Bestmann, U: Design of a Quadrotor System for an Autonomous Indoor Exploration. In: *IMAV 2014: International Micro Air Vehicle Conference and Competition 2014, Delft, The Netherlands, August 12-15, 2014* Delft University of Technology, 2014

[22] *Produktseite: Gazebo*. `http://gazebosim.org//`. – [Online; Stand 16. Dezember 2014]

[23] Goodrich, Michael A.: Potential fields tutorial. In: *Class Notes* (2002)

[24] Hartley, R. I. ; Zisserman, A.: *Multiple View Geometry in Computer Vision*. Second. Cambridge University Press, ISBN: 0521540518, 2004

[25] Ioan A. Sucan, Sachin C.: *MoveIt!* http://moveit.ros.org. Version: 2013. – [Online; Stand 12. Dezember 2014]

[26] Iwata, Mari: *Bei Aufräumarbeiten in Fukushima geht nichts ohne Roboter*. http://www.wsj.de/nachrichten/SB10001424052702304554004579425201034056212. Version: 2014. – [Online; Stand 17. Dezember 2014]

[27] Klein, Georg ; Murray, David: Parallel Tracking and Mapping for Small AR Workspaces. In: *Proc. Sixth IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR'07)*, 2007

[28] Klingbeil, Lasse ; Nieuwenhuisen, Matthias ; Schneider, Johannes ; Eling, Christian ; Droeschel, David ; Holz, Dirk ; Läbe, Thomas ; Förstner, Wolfgang ; Behnke, Sven ; Kuhlmann, Heiner: Towards Autonomous Navigation of an UAV-based Mobile Mapping System. In: *4th International Conference on Machine Control & Guidance*, 2014, 136–147

[29] Konolige, Kurt: Small vision system: Hardware and implementation. In: *Proc. of the Intl. Symp. of Robotics Research (ISRR*, 1997, S. 111–116

[30] Krajník, Tomáš ; Faigl, Jan ; Vonásek, Vojtěch ; Košnar, Karel ; Kulich, Miroslav ; Přeučil, Libor: Simple yet stable bearing-only navigation. In: *Journal of Field Robotics* 27 (2010), Nr. 5, S. 511–533

[31] Kuffner, J.J. ; LaValle, S.M.: RRT-connect: An efficient approach to single-query path planning. In: *Robotics and Automation, 2000. Proceedings. ICRA '00. IEEE International Conference on* Bd. 2, 2000. – ISSN 1050–4729, S. 995–1001 vol.2

[32] Lauterbach, Helge A. ; Gageik, Dipl-Ing N.: Stereo-Optische Abstandsmessung für einen autonomen Quadrocopter. (2013)

[33] Lavalle, Steven M.: Rapidly-Exploring Random Trees: A New Tool for Path Planning. 1998. – Forschungsbericht

[34] Leutenegger, S. ; Chli, M. ; Siegwart, R.Y.: BRISK: Binary Robust invariant scalable keypoints. In: *Computer Vision (ICCV), 2011 IEEE International Conference on*, 2011. – ISSN 1550–5499, S. 2548–2555

[35] Lünsch, Dennis: *3D Objekterkennung und Lokalisierung in der Intralogistik*, Westfälische Hochschule Gelsenkirchen, Bachelorthesis, 2012

[36] *Produktseite: Logitech C270.* http://www.logitech.com/de-de/product/hd-webcam-c270. – [Online; Stand 15. Juli 2014]

[37] Lowe, David G.: Distinctive Image Features from Scale-Invariant Keypoints. In: *Int. J. Comput. Vision* 60 (2004), November, Nr. 2, 91–110. http://dx.doi.org/10.1023/B:VISI.0000029664.99615.94. – DOI 10.1023/B:VISI.0000029664.99615.94. – ISSN 0920–5691

[38] Meyer, Johannes ; Sendobry, Alexander ; Kohlbrecher, Stefan ; Klingauf, Uwe ; Stryk, Oskar von: Comprehensive Simulation of Quadrotor UAVs using ROS and Gazebo. In: *3rd Int. Conf. on Simulation, Modeling and Programming for Autonomous Robots (SIMPAR)*, 2012, S. to appear

[39] Michels, Jeff ; Saxena, Ashutosh ; Ng, Andrew Y.: High speed obstacle avoidance using monocular vision and reinforcement learning. In: *In ICML*, 2005, S. 593–600

[40] *Produktseite: Nano RC Quadcopter.* `http://www.amazon.de/Worlds-Smallest-Wltoys-Quadcopter-Transmitter/dp/B00M32ZFFS/ref=sr_1_13?s=computers&ie=UTF8&qid=1420200886&sr=1-13&keywords=mini+quadcopter`. – [Online; Stand 02. Januar 2015]

[41] *NIFTi UAV - Mosquito - performs 2D mapping and 3D perception using Hector SLAM.* `https://www.youtube.com/watch?v=-KzifavR6bA`. – [Online; Stand 02. Januar 2015]

[42] Nüchter, Andreas ; Lingemann, Kai ; Hertzberg, Joachim ; Surmann, Hartmut: 6D SLAM&Mdash;3D Mapping Outdoor Environments: Research Articles. In: *J. Field Robot.* 24 (2007), August, Nr. 8-9, 699–722. `http://dx.doi.org/10.1002/rob.v24:8/9`. – DOI 10.1002/rob.v24:8/9. – ISSN 1556–4959

[43] Nyman, Jens: *Designing an Autonomous Exploration Architecture for an Indoor Quadcopter*, Universität Gent, Diplomarbeit, 2012

[44] *Produktseite: OpenCV.* `http://opencv.org/`. – [Online; Stand 25. Juli 2014]

[45] Oppermann, Nils: *Gestenbasierte Mensch-Roboter-Schnittstelle*, Westfälische Hochschule Gelsenkirchen, Masterthesis, 2011

[46] *Produktseite: ARDrone 2.0.* `http://ardrone2.parrot.com/`. – [Online; Stand 15. Juli 2014]

[47] *Produktseite: DJI Phantom II.* `http://www.dji.com/product/phantom-2`. – [Online; Stand 02. Januar 2015]

[48] *Produktseite: ROS - Robot Operating System.* `http://www.ros.org`. – [Online; Stand 10. Juli 2014]

[49] *ROS OpenCV Bridge.* `http://wiki.ros.org/cv_bridge`. – [Online; Stand 31. Juli 2014]

[50] *ROS Paket Übersicht.* `http://www.ros.org/browse/list.php`. – [Online; Stand 10. Juli 2014]

[51] Rosten, Edward ; Drummond, Tom: Machine Learning for High-speed Corner Detection. In: *Proceedings of the 9th European Conference on Computer Vision - Volume Part I.* Berlin, Heidelberg : Springer-Verlag, 2006 (ECCV'06). – ISBN 3–540–33832–2, 978–3–540–33832–1, 430–443

[52] Rublee, E. ; Rabaud, V. ; Konolige, K. ; Bradski, G.: ORB: An efficient alternative to SIFT or SURF. In: *Computer Vision (ICCV), 2011 IEEE International Conference on*, 2011. – ISSN 1550–5499, S. 2564–2571

[53] Rudall, Yan: *Kooperative Robotik - Kooperation zwischen einem unbemannten Flugobjekt und einem Rettungsroboter*, Westfälische Hochschule Gelsenkirchen, Bachelorthesis, 2012

[54] Şucan, Ioan A. ; Moll, Mark ; Kavraki, Lydia E.: The Open Motion Planning Library. In: *IEEE Robotics & Automation Magazine* 19 (2012), December, Nr. 4, S. 72–82. http://dx.doi.org/10.1109/MRA.2012.2205651. – DOI 10.1109/MRA.2012.2205651. – http://ompl.kavrakilab.org

[55] *Projektseite: TRADR*. http://www.tradr-project.eu/. – [Online; Stand 20. Dezember 2014]

[56] WANG, Fei ; CUI, Jin-Qiang ; CHEN, Ben-Mei ; LEE, Tong H.: A Comprehensive {UAV} Indoor Navigation System Based on Vision Optical Flow and Laser FastSLAM. In: *Acta Automatica Sinica* 39 (2013), Nr. 11, 1889 - 1899. http://dx.doi.org/http://dx.doi.org/10.1016/S1874-1029(13)60080-4. – DOI http://dx.doi.org/10.1016/S1874–1029(13)60080–4. – ISSN 1874–1029

[57] Wikipedia: *Bundle adjustment — Wikipedia, The Free Encyclopedia*. http://en.wikipedia.org/w/index.php?title=Bundle_adjustment&oldid=614853841. Version: 2014. – [Online; Stand 05. August 2014]

[58] Wikipedia: *Epipolargeometrie — Wikipedia, Die freie Enzyklopädie*. http://de.wikipedia.org/w/index.php?title=Epipolargeometrie&oldid=130740678. Version: 2014. – [Online; Stand 25. Juli 2014]

[59] Wikipedia: *Lucas–Kanade method — Wikipedia, The Free Encyclopedia*. http://en.wikipedia.org/w/index.php?title=Lucas%E2%80%93Kanade_method&oldid=607297873. Version: 2014. – [Online; Stand 04. August 2014]

[60] Wikipedia: *Singulärwertzerlegung — Wikipedia, Die freie Enzyklopädie*. http://de.wikipedia.org/w/index.php?title=Singul%C3%A4rwertzerlegung&oldid=127726476. Version: 2014. – [Online; Stand 23. Juli 2014]

[61] Wikipedia: *Stereoskopie — Wikipedia, Die freie Enzyklopädie*. http://de.wikipedia.org/w/index.php?title=Stereoskopie&oldid=132017719. Version: 2014. – [Online; Stand 17. Juli 2014]

[62] Wikipedia: *Kreiseldrift — Wikipedia, Die freie Enzyklopädie*. http://de.wikipedia.org/w/index.php?title=Kreiseldrift&oldid=137482680. Version: 2015. – [Online; Stand 7. Januar 2015]

[63] Zhang, Zhengyou: A flexible new technique for camera calibration. In: *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 22 (2000), Nov, Nr. 11, S. 1330–1334. http://dx.doi.org/10.1109/34.888718. – DOI 10.1109/34.888718. – ISSN 0162–8828