Entwicklung eines Indoor-Assistenzsystems für Multicopter

mit Hilfe von Monocularer Tiefenbild Rekonstruktion

Studienarbeit

Studiengang Angewandte Informatik

Duale Hochschule Baden-Württemberg Karlsruhe

von

Christoph Meise, Max Lenk

Datum der Abgabe:	15.05.2017
Bearbeitungszeitraum:	2 Semester
Matrikelnummern und Kurse:	4050853, 3460046, TINF14B2, TINF14B1
Betreuer:	Markus Strand

Ehrenwörtliche Erklärung

"Ich erkläre ehrenwörtlich:

1. dass ich meine Projektarbeit mit dem Thema

   Entwicklung eines Indoor-Assistenzsystems für Multicopter

   ohne fremde Hilfe angefertigt und selbstständig verfasst habe;
2. dass ich die Übernahme wörtlicher Zitate aus der Literatur sowie die Verwendung der Gedanken anderer Autoren an den entsprechenden Stellen innerhalb der Arbeit gekennzeichnet habe;
3. dass ich meine Projektarbeit bei keiner anderen Prüfung vorgelegt habe.

Ich bin mir bewusst, dass eine falsche Erklärung rechtliche Folgen haben wird."

Walldorf, der 16.09.2016

Christoph Meise, Max Lenk

Restriction notice

This report contains confidential information of

SAP SE

Dietmar-Hopp-Allee 16

69190 Walldorf, Germany

It may be used for examination purposes as a performance record of the department of Applied Computer Science at the Cooperative State University Karlsruhe. The content has to be treated confidentially.

Duplication and publication of this report - as a whole or in extracts - is not allowed.

Sperrvermerk

Der Inhalt dieser Arbeit darf weder als Ganzes noch in Auszügen Personen außerhalb des Prüfungsprozesses und des Evaluationsverfahrens zugänglich gemacht werden, sofern keine anders lautende Genehmigung der Ausbildungsstätte vorliegt.

Abstrakt

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einleitung

Autonomes Fahren, Machine Learning und Industrie 4.0. Hinter diesen aktuellen Themen steckt das Ziel, Abläufe und Zusammenhänge kontinuierlich zu automatisieren und für den Menschen zu vereinfachen. Das Thema der Automation kann vor allem in der Interaktion zwischen Mensch und Maschine hilfreich sein. Auf Grund der geringen Kosten und der hohen Anwendungsvielfalt bieten sich vor allem Drohnen für die Forschung zu Automation in der Robotik an.
Mit Modellen ab 30 und bis zu mehreren tausend Euro gibt es Ausführungen für nahezu jeden Anwendungsfall. Meist mit mehreren Sensoren und Kameras ausgestattet, stellen sie nicht nur Spielzeug dar, sondern sind essentiell für reale Anwendungsgebiete.
So werden heute schon Drohnen genutzt, um Katastrophengebiete und Kriegsregionen aus sicheren Standorten aufzuklären, oder um die Feuerwehr bei der Branderkundung und Menschensuche zu unterstützen.
Natürlich können sie auch genutzt werden, um alltägliche Probleme zu lösen, wie die schnelle und direkte Lieferung von Paketen.
Da diese große Zahl an Drohnen nicht mehr manuell gesteuert werden kann, müssen sich diese größtenteils autonom bewegen. Dabei treten eine Vielzahl von komplexen Problemen auf, wie das zurechtfinden in einem unbekannten Raum und die Objekterkennung.
Außerdem ist bei Fluggeräten das Problem, dass die verwendete Ausrüstung ten-

denziell leicht und klein sein muss, damit die Flugeigenschaften nicht eingeschränkt werden bzw. die Drohne nicht zu groß wird.
Im Umfang dieser Arbeit soll eine Vorstufe zum autonomen Fliegen untersucht und implementiert werden: ein Assistenzsystem für den manuellen Flug. Dies ist Vergleichbar mit den Assistenzsystemen in PKWs, bei denen ein Tempomat, Licht- und Regensensoren oder Spurhalteassistenten den Fahrer unterstützen, jedoch das Fahren nicht abnehmen.
Eine solche semi-autonome Assistenzfunktion könnte beispielsweise dabei helfen, die Drohne durch enge Räume und durch Hindernisse zu fliegen, also Flugmanöver, bei denen ein Mensch eine hohe Fehleranfälligkeit aufweist.

1.1 Motivation

Das Ziel besteht darin, dass die Drohne aktiv die Umgebung auswertet und dabei Objekte wie Türen, oder Wände erkennt. Der Unterschied zu bereits bestehenden Projekten in diesem Themengebiet besteht darin, dass nur eine einzelne monokulare Kamera verwendet werden soll, anstatt externe Tiefenbildkameras zu montieren.
Dadurch trifft man auf eine Vielzahl von komplexen Problemen, welche im weiteren Verlauf dieser Arbeit dargestellt werden. Auf der anderen Seite könnte man dadurch teure Hardware sparen und somit auch auf andere Projekte anwenden.
Anhand der Tiefenbilder soll die Drohne anschließend in der Lage sein, die Entscheidungen des Nutzers zu unterstützen und abzuändern, um somit beispielsweise Kollisionen zu vermeiden, oder gezielt durch Hindernisse zu fliegen.
Das Problem soll nur durch die integrierte Hardware gelöst werden, also einer einfachen 720p Kamera die nach vorn gerichtet an der Drohne befestigt ist. Weiterhin ist das Ziel der Arbeit eine vollständige Modularisierung der Softwarearchitektur.

Dies ist essentiell, da das Assistenzsystem sowohl bei einer realen Drohne, als auch in einer Simulation funktionieren soll.

1.2 Aufbau

Diese Studienarbeit besteht aus 3 Kapiteln. Im ersten Abschnitt der Arbeit sollen die Grundlagen erklärt werden. Zuerst wird dabei die genutzte Drohne und deren Spezifikationen dargestellt.
Anschließend wird das Software Framework Roboter Operating System *(ROS)* eingeführt, welches ein Hauptbestandteil der Projektarchitektur ausmacht.
Im weiteren Verlauf wird dann die Simulationsumgebung beschrieben, in der eine simulierte Drohne geflogen werden kann. Im letzten Teil dieses Kapitels wird die Tiefenbildkamera Kinect beschrieben, welche zur Umsetzung der Gestensteuerung genutzt wird.
Der zweite Abschnitt dieser Arbeit umfasst die Software Architektur. Zuerst werden hierbei die technischen und Anforderungen an das Projekt aufgestellt. Anschließend ist das Kapitel in drei logische abgegrenzte Unterkapitel unterteilt: Bildverarbeitung, Implementierung und das Assistenzsystem.
Im Teil Bildverarbeitung wird beschrieben, wie aus den einfachen Bildern zuerst die Position der Kamera im Raum approximiert wird, um anschließend aus aufeinanderfolgenden Aufnahmen Tiefeninformationen zu gewinnen.
Im Folgenden wird im Abschnitt Implementierung die softwaretechnische Umsetzung und die Architektur beschrieben. Abschließend dient das Unterkapitel Assistenzsystem dazu, eine allgemeine Hinführung zu bieten und mögliche Implementierungstechniken vorzuschlagen.
Der Letzte Teil der Arbeit ist die Evaluation. Diese besteht aus dem erzielten Ergebnis, sowie aus dem Ausblick für weitere Betrachtungen dieser Problematik. Dabei

werden außerdem aufgetretene Probleme und Hindernisse detailliert beschrieben.

2 Grundlagen

2.1 AR.Drone 2.0

Bei der AR.Drone 2.0 handelt es sich um einen ferngesteuerten Quadrocopter des französischem Herstellers Parrot SA. [4] Die Drohne ist standardmäßig steuerbar mit einer mobilen Applikation für Android und iOS Geräte. Dafür baut sie ein WLAN Netzwerk auf, mit dem sich die Geräte verbinden können. Zur Steuerung stellt die AR.Drone ein Interface zur Verfügung, mit dem sie ferngesteuert werden kann.
Im Umfang der Studienarbeit wird, wie in der folgenden Abbildung zu sehen, die aktuellste Version der AR.Drone 2.0 verwendet.

Abbildung 2.1: Parrot AR.DRONE 2.0 Elite Edition [1]

Diese zeichnet sich unter Anderem durch eine Frontkamera mit einer Auflösung von 1280×720 Pixeln und einer Bildrate von 30 fps aus. Weiterhin ist Sie mit einer zum Boden gerichteten QVGA Kamera ausgerüstet, welche 60 Bilder pro Sekunde aufnimmt.
Die Drohne orientiert sich beim Fliegen mit Hilfe einer Vielzahl von Sensoren. Dazu gehören ein dreiachsiges Gyroskop und ein Magnetometer. Weiterhin nutzt sie Beschleunigungs-, Ultraschall- und Luftdrucksensoren.
Der Grund für die Wahl der Drohne ist vor allem der vergleichsweise niedrige Preis von ca. 200€ und der starken Verbreitung in der Forschung. Dadurch gibt es bereits eine Vielzahl von Projekten, die dazu führen, dass die Drohne und das dazugehörige Interface zu einem großen Umfang fehlerfrei funktionieren.
Weiterhin gibt es schon ROS Nodes (siehe 2.2) und konfigurierte Modelle in Simulationsumgebungen, welche die Arbeit an dem Projekt beschleunigen.

2.2 ROS

2.2.1 Allgemeines

Das Robotic Operating System, kurz ROS, ist eine Sammlung von Softwareframeworks für die Entwicklung von Software für persönliche Roboter. Es stellt entsprechende Bibliotheken und Werkzeuge zur Verfügung, um Entwicklern die Programmierung zu vereinfachen. Dabei bietet ROS einem Betriebssystem ähnliche Funktionalitäten auf Basis eines homogenen Computercluster. Dazu gehören Hardwareabstraktion, low-level Steuerung, Nachrichtevermittlung zwischen verschiedenen Prozessen und Paketmanagement. Trotz der Notwendigkeit hoher Reaktivität und geringer Latenz bei der Steuerung von Robotern handelt es sich es de facto nicht um ein richtiges Betriebssystem , obwohl es durch den Namen(“Operating System“) suggeriert wird. Dennoch ist es möglich Echtzeitcode (“realtime code“) in ROS zu

integieren [5]. ROS ist eins der am meisten genutzten Frameworks und hat eine stark wachsende Gemeinschaft, was es in Kombination mit dessen Feature zu einer enorm wichtigen Technologie macht.

2.2.2 Design Prinzipien

Das Robotic Operating verfolgt fünf grundlegende Design Prinzipien:

- Peer-to-Peer
- werkzeugbasiert
- Mehrsprachigkeit
- Unabhänigkeit
- Open Source

Peer-to-Peer: Im Normalfall besteht ein Roboter aus mehreren Komponenten und oft aus verschiedenen Recheneinheiten. Oft gibt es einen zentralen leistungsstarken Rechner der unabhängig vom Roboter existiert, welcher für die Koordination und rechenintensive Aufgaben, wie zum Beispiel Bildverarbeitung verantwortlich ist. Da dieser oft wegen Mobilitätsgründen nicht per Kabel mit dem Roboter verbunden ist, geschieht die Kommunikation mittels WLAN oder vergleichbarer drahtloser Kommunikation. Dies kann unter Umständen sich schnell zu einem Flaschenhals entwickeln, da große Datenmengen transportiert werden müssen, wenn die zentrale Einheit für den Datentransport zuständig ist. Daher baut ROS auf ein Peer-to-Peer Konzept auf bei dem zentrale Einheit sich lediglich darum kümmert die kommunizierenden Nodes vor Kommunikationsbeginn miteinander verbindet. Dafür ist lediglich ein Auskunftsmechanismus notwendig, der Prozessen oder ähnlichem ermöglicht korrespondierende Kommunikationspartner zur Laufzeit zu finden und

eine Verbindung herzustellen. Somit können mögliche Flaschenhälse entschärft werden.

Werkzeugbasiert: Dabei weiterer grundlegendes Prinzip um die Benutzbarkeit zur vereinfachen und gleichzeitig eine höher Modularität zu gewähren. Anstelle ein großes komplexes Werkzeug zum Arbeiten mit ROS, existieren mehrere kleine Werkzeuge die dem Single-Responsibility-Prinzip(SRP) folgen, also nur eine konkrete Aufgabe haben. Dieses Prinzip lässt sich mit eine Mikroservice-Architektur vergleichen, wobei es sich allerdings um Werkzeuge handelt.

Mehrsprachigkeit: Damit möglichst viele Systeme an ROS angebunden werden können und auch die Vorteile einzelner Programmiersprachen ausgereizt werden können, existiert das Prinzip der Mehrsprachigkeit. Somit wird es dem Nutzer ermöglicht die Sprache mit der programmieren will, frei zu wählen und erhöht den Nutzkomfort. Momentan werden C++, Python und Lisp durch sogenannte ROS Client Libraries. Weiterhin existieren experimentelle Client Libaries für Sprachen GO, Haskell, Java und viele weitere.

Unabhängigkeit: Die Kopplung der Software mit der Hardware bei Robotikszenarien ist immer relativ hoch, da die Treiber meistens plattformspezifisch sind. Um allerdings die Wiederverwendbarkeit diverser Algorithmen zu fördern, setzt man darauf die Algorithmen möglichst unabhängig vom jeweiligen Roboter zu implementieren, sodass sie im Optimalfall bei jedem System Anwendung finden können. Entwickler stellen die entsprechenden als Bibliotheken zur Verfügung und die starke Kopplung wird etwas entschärft.

Open Source: Da ROS unter der BSD-Lizenz[1] steht, kann es sowohl für nicht-kommerzielle Projekte als auch kommerzielle Projekt verwendet werden. Der Source Code ist öffentlich zugänglich. Entwickler können ihre eignen Module beliebig lizenzieren.

[1] siehe: http://www.linfo.org/bsdlicense.html

2.2.3 ROS Nodes

ROS baut auf einem einfachen Konzept auf, dem Publish-Subscribe Pattern, bei welchem ein Publisher ein Nachricht mit einem festem Thema verschicken kann und ein beliebiger Subscriber, der sich für das Thema interessiert, ist in der Lage diese zu empfangen, zu verarbeiten und unter Umständen erneut zu versenden. Somit besteht eine ROS Anwendung aus der Regel aus vielen kleinen Teilen, den sogenannte Nodes. Jede Node hat ihre eigenen Aufgabe und registriert sich auf ein bestimmtes Thema(Topic), verarbeitet die empfangen Daten und publiziert sie für andere Nodes. Somit kann eine Node sowohl Publisher, als auch Subscriber sein. Topics werden in ROS durch einen festen Nachrichtentyp definiert. Dessen exakter Aufbau muss vor Verwendung deklariert werden um einen einheitlichen Nachrichtenaustausch zu gewähren. Wie in der Abbildung unterhalb zu sehen, wird der Nachrichtentransfer durch eine zentrale Einheit geregelt, sodass Daten wirklich nur die Nodes erreichen, die sich auch dafür interessieren. Dieser zentrale Bestandteil ist in ROS der Roscore, bei welchem sich alle Nodes zur Erstellung registrieren. Sozusagen eine Masternode, welche sich um die Verwaltung der anderen Nodes kümmert. Im Unterschied zum Pattern kümmert sich der Roscore nur um die anfängliche Vermittlung der Nodes untereinander.
ROS Nodes können in verschiedenen Sprachen implementiert werden, da die Kommunikation über die festen Nachrichtentypen geschieht, welche über den Roscore ausgetauscht werden. Dadurch ist es möglich Nodes in C++, Python und ebenfalls Lisp zu programmieren. Wodurch das ganze Konzept enorm flexibel gestaltet wird. Die fest definierten Nachrichtentypen verhindern, dass es an den Schnittstellen zu Problemen kommt und Nachrichten von allen Nodes einheitlich empfangen und versendet werden.
Durch diese Modularität ist einfacher Komponenten und Funktionalitäten sowohl zu verwalten, als auch zu warten. Ebenfalls sind durch einheitliche Schnittstellen

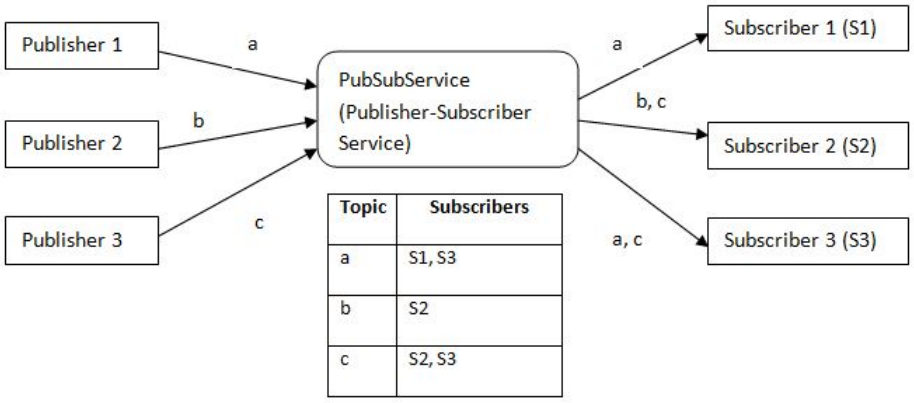


Abbildung 2.2: Publish-Subscribe Pattern

der Austausch von Nodes einfach und ermöglicht ein flexibles Ökosystem.

2.2.4 Software Qualität

Um die Qualität der Anwendung zu sichern wird Catkin Build anstelle des veralteten rosbuild verwendet. Bei Catkin sind Pakete der atomare Bestandteil der Anwendung.

2.3 Simulation mit Gazebo

Da es besonders beim Flug von Quadrocoptern schnell zu Schäden kommen kann und das sowohl in langen Ausfallzeiten, als auch zu erhöhten Materialkosten führen. Speziell beim Test von autonomen Verhalten ist der Test der Features in realer Umgebung somit mit hohem Risiko verbunden. Um dies zu vermeiden ist die Simulation von Quadrocoptern und verschiedener Umgebungen unabdingbar. Ebenfalls erleichtert eine simulierte Version der Drohne die Entwicklung, da sie nicht immer physikalisch vorhanden sein muss. Dabei ist es allerdings notwendig, insbesondere bei Bilddaten, dass die reale Situation möglichst realitätsnah abgebildet wird, so-

dass das Verhalten im realen Umfeld entsprechend ähnlich ist. Insbesondere bei der Verarbeitung von Kameradaten ist es allerdings abzuwägen ob Ergebnisse in der Simulation mit Resultaten in realen Umgebungen vergleichbar sind, da die simulierten Bilddaten in der Regel eher steril sind.
Durch die Integration von ROS kommt die Simulationsumgebung Gazebo als Bestandteil mit. Anfänglich handelte es sich um ein ROS Paket, mittlerweile ist es allerdings ein eigenständiges Ubuntupaket und benötigt de facto kein ROS.
Da die realistische Simulation einer Drohne sehr umfangreich ist wird das ROS Paket tum simulator"verwendet. Es enthält eine Implementation der AR Drone 2.0 für den Gazebo Simulator. Es wurde von Hongrong Huang und Jürgen Sturm aus der Computer Vision Group von der Technischen Universität München entwickelt. Die Drohne wird komplett abstrahiert, wodurch die Simulation sich ohne Veränderung mit der realen Drohne austauschen lässt. Es ist auch möglich sowohl den Quadro-

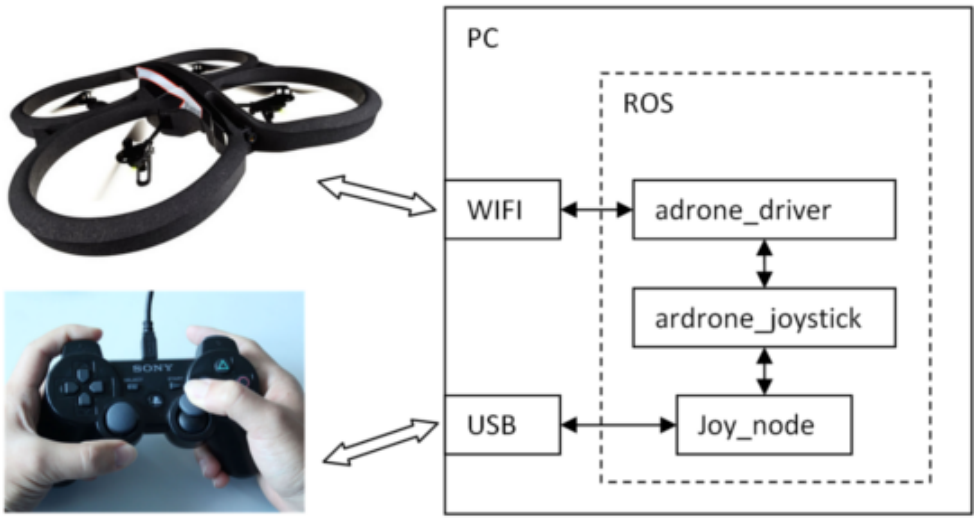


Abbildung 2.3: Programmstruktur mit dem realen Quadrocopter [**?**]

copter real zu steuern, als auch gleichzeitig in der Simulation. Das ermöglicht einen direkten Vergleich zwischen simulierten und realen Ergebnisse, vorausgesetzt, dass

die simulierte Umgebung der echten annähernd entspricht. Die Steuerung der Droh-

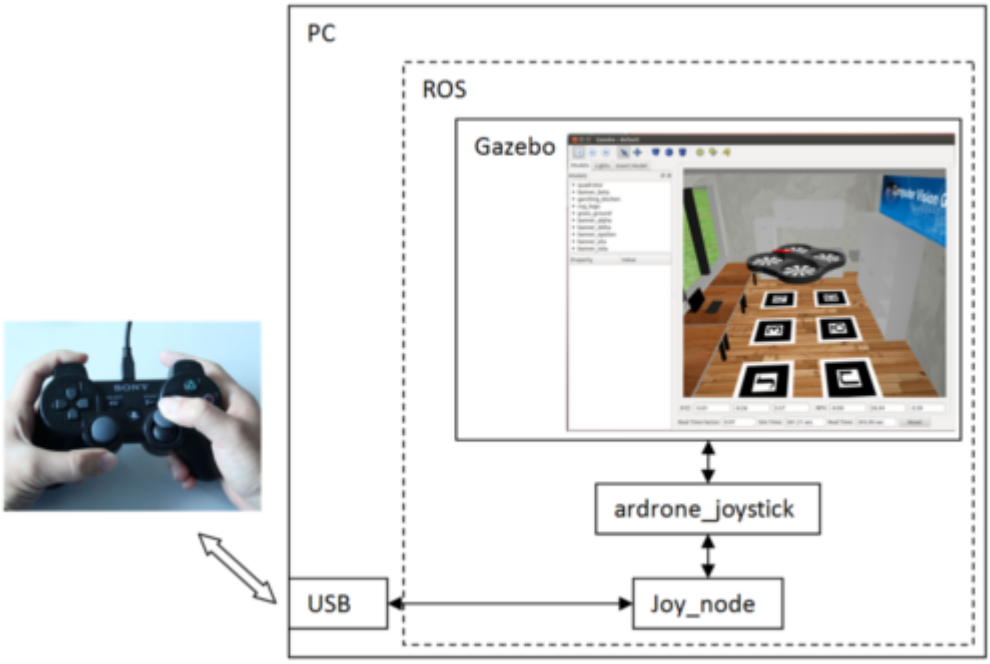


Abbildung 2.4: Programmstruktur unter Verwendung von Gazebo [**?**]

ne, wird bei dem Package nativ über einen Playstation 3 Controller realisiert. Dessen Eingaben werden von der ROS-Node "Joy Node"verarbeitet und anschließend an die Node ärdrone joystick"weitergeleitet. Dort werden die Daten entsprechend übersetzt um entweder von Gazebo und/oder der realen Drohne verwendet werden zu können. Wenn sie an den realen Quadrocopter gesendet werden sollen, müssen sie noch entsprechend vom ärdrone driver"übersetzt werden, da lediglich Bitfolgen per WLAN versendet werden.

2.4 Fuzzylogik

2.5 Kinect

Um Gesten des Benutzers zur erkennen und entsprechend auszuwerten wird der visuelle Sensor Microsoft Kinect verwendet. Normalerweise wird er zur Steuerung der Videospiel Konsole Xbox360/Xbox One verwendet. Das System ist in der Lage Tiefenbilder zu erstellen, besitzt sowohl eine 1080p Farbkamera, als auch einen Infrarotsensor und mithilfe mehrere Mikrofone Sprache und Bewegung im Raum zu erkennen. Anhand der verschiedenen Kameraeingaben ist es möglich den Körper von bis zu 2 Nutzern zu erkennen und auch entsprechend zu verfolgen. Für Windows stellt Microsoft ein entsprechendes Software Development Kit (SDK) zur Verfügung um die Benutzung der Funktionalität zu vereinfachen.

3 Software Architektur

3.1 Anforderungen

Im Folgenden sollen die Anforderungen an die Studienarbeit festgelegt werden.
Ein Hauptbestandteil der Arbeit besteht darin, ein Vorgängerprojekt mit einer AR.Drone 2.0 in das ROS Framework zu portieren. Dies beinhaltet eine Modularisierung der Projektbestandteile in ROS Nodes. Weiterhin läuft ROS nur unter UNIX basierten Betriebssystemen läuft und das bestehende Projekt Libraries verwendet, welche nur unter Windows verfügbar sind.
Ziel soll sein, den Quadrocopter mit Hilfe einer Kinect und Gestenerkennung steuern zu können. Diese soll mit Hilfe einer Kinect Stereo Kamera von Microsoft umgesetzt werden. So soll der Nutzer beispielsweise mit einer Bewegung von beiden ausgestreckten Armen nach oben die Drohne starten, steigen, senken und landen lassen können.
Eine weitere Vorgabe besteht darin, dass man neben der realen Drohne jegliche Funktionalität auch in einer Simulation laufen soll. Somit kann man für Präsentationen, in denen der Flug einer Drohne nicht möglich ist, den Quadrocopter in einer frei gestaltbaren simulierten Umgebung fliegen lassen.
Ein weiterer Bestandteil dieser Arbeit besteht darin, Ansätze und Limitationen der Implementierung eines Assistenzsystems zu testen und zu bewerten. Um dieses Vorhaben umzusetzen sollen externe Projekte und Abhandlungen zur Extraktion von Tiefeninformationen genutzt werden. Diese werden aus mehreren aufeinanderfol-

genden Bildern einer monokularen Kamera [1] bestimmt. Hierbei soll es wiederum möglich sein, dass der Videostream sowohl von der realen, als auch von der simulierten Drohne gesendet werden kann. Es soll dabei ermittelt werden, ob die Nutzung der externen Arbeiten für den Anwendungszweck praktikabel und sinnvoll ist.
Das Ziel der Arbeit ist anschließend herauszufinden, wie man mit den Tiefeninformationen die manuelle Steuerung der Drohne durch eine Person mit Hilfe von Assistenzfunktionen unterstützen kann. Ein Anwendungsbeispiel ist hierbei das sichere Fliegen durch ein Hindernis wie eine offene Tür, oder das verhindern von Kollisionen mit Objekten in der Umgebung.

[1] Monokular ist die Bezeichnung für Kameras mit einer einzelnen Linse

3.2 Bildverarbeitung

In diesem Abschnitt wird beschrieben, wie die Position der Kamera im Raum bestimmt werden kann, um anschließend aus aufeinanderfolgenden Aufnahmen Tiefeninformationen zu gewinnen. Die beschriebenen Implementierungen beziehen sich dabei auf die Arbeit von Christian Forster, Matia Pizzoli und Davide Scaramuzza, welche ihre Abhandlungen zum Thema zusammen mit dem Programmcode frei zugänglich gemacht haben.

3.2.1 Semi-Direct Monocular Visual Odometry - SVO

Eine Grundanforderung an das Projekt ist die Nutzung einer nicht modifizierten AR.Drone. Dadurch entsteht die Problematik, dass keine Tiefenbildkamera genutzt werden kann, um in Echtzeit Tiefenbilder zu erhalten. Die Drohne ist lediglich mit einer monokularen Frontkamera ausgestattet.
Um Tiefeninformationen aus den Bildern einer solchen Kamera zu gewinnen, wird eine Szene aus verschiedenen Perspektiven aufgenommen. Anschließend gibt es unterschiedliche Ansätze um aus den aufeinanderfolgenden Bildern Kamerapositionen und Umgebungsstrukturen zu ermitteln.
Feature basierte Ansätze sind der aktuelle Standard zur Berechnung der Kamerapostion. Diese versuchen die wichtigsten Merkmale eines Bildes, die Features, zu extrahieren. Mit Hilfe von Feature Deskriptor Algorithmen werden Vektoren mit Informationen zu invarianten Bildbereichen berechnet. Diese Vektoren verhalten sich wie ein einzigartiger Fingerabdruck, der die Merkmale repräsentiert.
Aufeinanderfolgende Bilder werden dann mit Hilfe dieser Deskriptoren abgeglichen und sowohl Kamerabewegungen, als auch Strukturen werden rekonstruiert. Zur Optimierung sind abschließend die ermittelten Kamerapositionen anzugleichen. Dies

geschieht mit Hilfe von Algorithmen zur Minimierung des Reprojektionsfehlers. [2]
Ein weiterer Ansatz ist die direkte Methode. Hierbei werden die Features nicht über Deskriptor Algorithmen bestimmt, sondern das Problem wird über die Intensitäten der Pixel gelöst. Bei einem Graustufenbild entspricht diese Intensität der Helligkeit von Bildbereichen.
Somit kann bei der Rekonstruktion im Gegensatz zum Feature basierten Ansatz auch die Richtung der Gradienten von Intensitäten genutzt werden. Dadurch funktioniert diese Methode auch bei Bildern mit sehr wenig Textur, Bewegungsunschärfe und fehlerhaftem Kamerafokus.
Das für diese Arbeit relevante Vorgehen kombiniert die Vorteile der beschriebenen Methoden. Die semi-direkte Odometrie verwendet einen Algorithmus der ebenfalls auf Zusammenhängen von Features basiert. Diese werden jedoch implizit aus einer direkten Bewegungsabschätzung bezogen, anstatt explizit durch Algorithmen mit Feature Deskriptoren berechnet zu werden.
Dadurch müssen Features nur extrahiert werden, wenn diese noch nicht auf einem der vorherigen Bildern vorhanden waren.
Insgesamt ist dieser Ansatz sehr schnell, da wenig Berechnungen pro Bild stattfinden und auf Grund der Verwendung von Intensitätsgradienten äußerst genau und robust.
Diese Eigenschaften sind für die Anforderungen der Studienarbeit essentiell, da die Drohne sich sehr schnell bewegen kann und somit in möglichst kurzer Zeit neue Bilder auswerten muss. Das beschrieben Verfahren minimiert damit die Auswirkungen der typischen Probleme von Drohnen. Diese sind einerseits die niedrige Texturierung der Umgebung, welche hauptsächlich in Innenräumen auftritt und andererseits kameraspezifische Probleme wie Bewegungsunschärfe und der Verlust des Kamerafokus.

[2] Reprojektionsfehler sind geometrische Fehler die im Zusammenhang zwischen abgebildeten und berechneten Bildpunkten entstehen.

Im Folgenden zeigt die Abbildung wie die Nutzung von SVO in der Praxis aussieht.

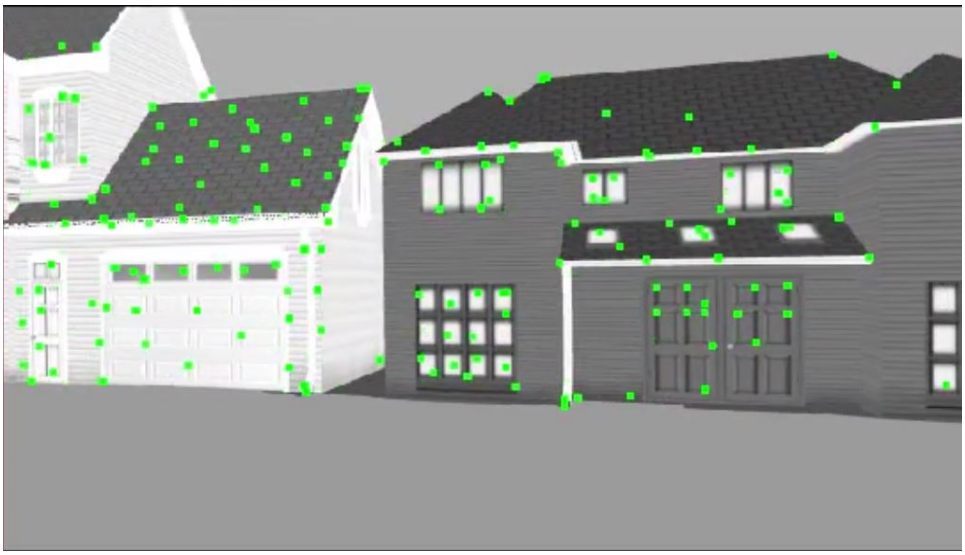


Abbildung 3.1: Semi-Direct Monocular Visual Odometry im Simulator

Hierbei stammt das Kamerabild von einer Drohne im Simulator Gazebo in einer frei verfügbaren Testwelt mit einigen Gebäuden. Die grünen Punkte sind die Features, die SVO anhand der beschriebenen Strategie ermittelt hat.
Die Anzahl der aktuell gefundenen Features kann dabei stark variieren. Diese Schwankung entsteht hauptsächlich durch die unterschiedliche Texturierung und Anzahl der Kanten von Objekte in der Umgebung.
Weiterhin kann auch die Kamera selbst ein Grund für eine geringe Anzahl gefundener Features sein. Gründe und Lösungen dafür werden im Folgenden beschrieben.

3.2.2 Kamerakalibrierung

Ein Grundproblem der Bildverarbeitung ist die Verzerrung des Bildes. Da die Bestimmung der Kameraposition möglichst genau sein soll, muss die Kamera vorher kalibriert werden. Dies bedeutet, dass Ungenauigkeiten der Linse erkannt und soft-

wareseitig ausgeglichen werden. Dafür werden die intrinsischen Parameter der Kamera bestimmt

Kalibrierungsmodelle

Hierbei unterstützt SVO drei Kamera Modelle: ATAN, Ocam und Lochkamera. [6]
Das ATAN Modell basiert auf dem *Field of View* (FOV) Verzerrungsmodell SStraight lines have to be straight [...]"von Devernay und Faugeras.
Der Vorteil dieser Kalibrierungsmethode ist die äußerst schnelle Berechnung der Projektion des Bildes. Das Modell vernachlässigt jedoch tangentiale Verzeichnung, welche auftritt, wenn optische und mechanische Bestandteile des Objektives, sowie der CCD-Sensor [3] nicht perfekt zueinander ausgerichtet sind. Weiterhin sollte die Kamera mit einem globalem Shutter ausgestattet sein, um die Extraktion von Bildmerkmalen bei Bewegungen zu gewährleisten. Kameras mit Global-Shutter-CMOS [4] Sensoren und CCD-Sensoren nehmen Bild nicht zeilen- und spaltenweise, sondern vollständig auf und sind daher für das Verfahren geeignet.
Die Drohne besitzt eine veraltete und günstige Kamera mit einem CMOS Sensor, wodurch sowohl tangentiale Verzerrung, als auch der Rolling-Shutter-Effekt auftreten können.
Daher ist das ATAN Modell zwar eine der besten Kalibrierungsmethoden für teure Hochleistungskameras, jedoch ist es für die Betrachtungen dieser Arbeit nicht optimal.
Der zweite Ansatz zur Kalibrierung ist das Ocam Modell von Davide Scaramuzza. Diese Methode sollte für Kameras mit sehr weitem Sichtfeld, oder omnidirektiona-

[3] CCD steht für *charge-coupled device*, was übersetzt ladungsgekoppeltes Bauteil bedeutet. Dieses lichtempfindliche elektronische Bauteil wird zur Bildaufnahme verwendet.
[4] CMOS steht für *Complementary metal-oxide-semiconductor* und ist ein spezieller Halbleiter der zur Bildaufnahme verwendet wird.

len Kameras genutzt werden. Damit ist es für die Drohne nicht geeignet.
Die dritte unterstützte Kalibrierungsmethode ist das Modell der *Lochkamera*, bzw. auf Englisch *Pinhole Model*. Der Name und das zugrunde liegende Prinzip basieren, wie der Name es sagt, auf der Lochkamera. Die Abbildung zeigt die grundsätzliche Funktionsweise.

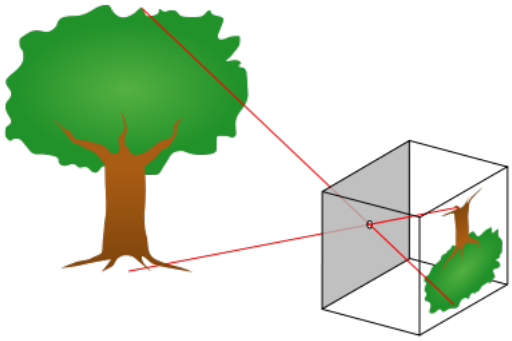


Abbildung 3.2: Prinzip der Lochkamera [2]

Die Darstellung zeigt, dass bei der Lochkamera Licht durch eine kleine Öffnung in einen kleinen Hohlkörper fällt. Dadurch entsteht auf der Rückseite ein auf dem Kopf stehendes Bild.
Bei dem Pinhole Model handelt es sich um den aktuellen Standard in OpenCV [5] und ROS. Hierbei wird die Verzerrung mit Hilfe von fünf intrinsischen Parametern beschrieben, welche im Rahmen der Kalibrierung bestimmt werden müssen.

$$\mathrm{Distortion}_{\mathrm{coefficients}} = (k_1 \quad k_2 \quad p_1 \quad p_2 \quad k_3)$$

OpenCV betrachtet dabei radiale und tangentiale Faktoren. Die Formel für radiale Verzeichnung ist die Folgende:

Hierbei wird aus einem alten Bildpunkt (x, y) des Eingabebildes die korrigierte

[5] "OpenCV is the leading open source library for computer vision, image processing and machine learning, and now features GPU acceleration for real-time operation."

$$x_{corrected} = x(1 + k_1 r^2 + k_2 r^4 + k_3 r^6)$$
$$y_{corrected} = y(1 + k_1 r^2 + k_2 r^4 + k_3 r^6)$$

Position $x_{corrected} y_{corrected}$ bestimmt.
Die Berechnung der tangentialen Verzerrung erfolgt durch die Formel:

$$x_{corrected} = x + [2p_1 xy + p_2(r^2 + 2x^2)]$$
$$y_{corrected} = y + [p_1(r^2 + 2y^2) + 2p_2 xy]$$

Abschließend werden die Einheiten angepasst:

$$\begin{bmatrix} x \\ y \\ w \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}$$

Dabei entspricht f_x und f_y der Brennweite der Linse und c_x, sowie c_y beschreiben die optische Bildmitte in Pixelkoordinaten.
Dieser Ansatz ist der einfachste und funktioniert grundsätzlich mit jeder Kamera.

Umsetzung der Kalibrierung

Um die intrinsischen Parameter der Kamera zu bestimmen, wird ein bekanntes Bild oder Muster aufgenommen. Dazu wird ein Vergleich zwischen den theoretischen und tatsächlichen Abmessungen angestellt. Hierzu wird meist ein einfaches Schachbrettmuster genutzt, welches im möglichst vielen verschiedenen Perspektiven aufgenommen wird. Bei der realen Drohne wird dazu das Muster ausgedruckt, bei der Simulation muss hingegen ein muss ein solches Objekt in die Welt eingefügt werden. Da die Kamera in der Simulation ohnehin keine intrinsischen Fehler aufweisen sollte, kann auf eine Kalibrierung verzichtet werden.

Die Kalibrierung wurde mit dem frei verfügbaren ROS Node *camera_calibration* umgesetzt. Das Ergebnis ist abhängig von der Anzahl der Perspektiven und der

Qualität der Aufnahmen. Aufgegeben wird dann die List der Parameter, die für das Lochkamera Modell notwendig sind.

3.2.3 Regularized Monocular Depth Estimation - REMODE

Im vorherigen Abschnitt wurde beschrieben, wie anhand von aufeinanderfolgenden Bildern die Position der Kamera im Raum bestimmt werden kann. Dieses Problem wird seit mehr als 20 Jahren untersucht und wird als Structure From Motion *SFM* in der Bildverarbeitung und Simultaneous Localization and Mapping *SLAM* in der Robotik bezeichnet.

Um den Nutzer aktiv bei der Steuerung der Drohne zu unterstützen werden jedoch Tiefeninformationen benötigt. Dazu müssen Tiefenbilder und Tiefenkarten (footnote) aus den Bildern der monokularen Kamera bestimmt werden.

Für diesen Schritt gibt es unterschiedliche Ansätze. Der State of the Art ist die Berechnung mit Hilfe des Bayes-Schätzers. Dabei handelt es sich um eine Schätzfunktion in der mathematischen Statistik, welche eventuell vorhandenes Vorwissen bei

der Schätzung eines Parameters berücksichtigt. Dabei wird in der bayesschen Statistik das initiale Vorwissen mit Hilfe der A-priori-Verteilung modelliert, die bedingte Wahrscheinlichkeit des Parameters unter Betrachtung dieses Vorwissens mit der A-posteriori-Verteilung.

Im Umfang dieser Arbeit wird die Forschung und Implementierung des Projekts Regularized Monocular Depth Estimation *REMODE* genutzt. In dieser Ausarbeitung von Matia Pizzoli, Christian Forster und Davide Scaramuzza wird die bayessche Schätzung mit neusten Entwicklungen in der Konvexoptimierung verbunden. Hierbei stützen sie ihre Forschungen auf die Ergebnisse von G. Vogiatzis und C. Hernandez und ihrer Abhandlung mit dem Titel "Video-based, real-time multi-view stereo" von 2011.

REMODE erfüllt den Zweck, mit Hilfe der gewonnenen Informationen ein dreidimensionales Modell des Raumes zu erstellen. Die Abbildung zeigt ein Beispiel dazu, welches von den Entwicklern verbreitet wurde.

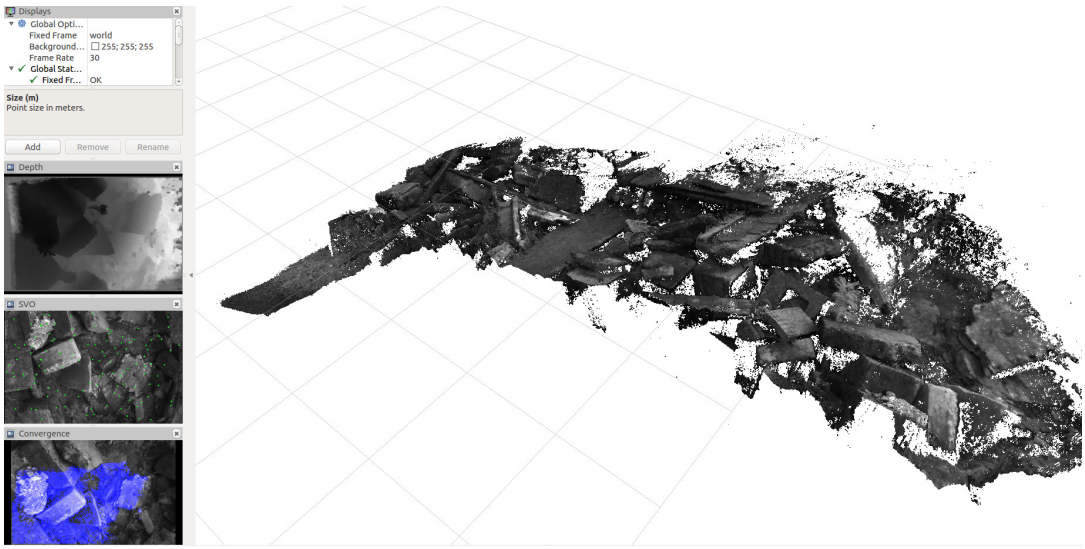
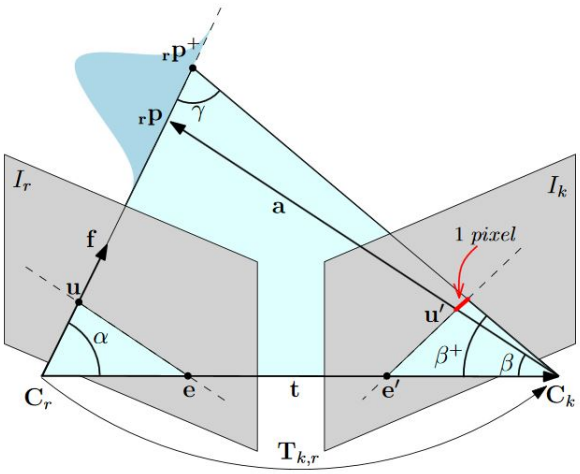


Abbildung 3.3: REMODE 3D-Modell, Flug über Trümmer [3]

Die Darstellung zeigt ein 3D Modell eines Trümmerhaufens, welches mit Hilfe der aufeinanderfolgenden Einzelbildern generiert wurde. Die Visualisierung erfolgte mit dem Standard 3D Visualisierungstool für ROS, genannt *RVIZ*.
REMODE ermöglicht eine Berechnung der Tiefeninformationen in Echtzeit und auf Pixelbasis. Weiterhin ist die aktuelle Genauigkeit und Fehlerrate im Vergleich zur Realität zu jeder Zeit sichtbar.
Im Folgenden wird der grundsätzliche Ansatz der Implementierung skizziert. Die genauen Details übersteigen dabei auf Grund der hohen Komplexität den Umfang dieser Arbeit.



In der Übersicht sieht man die beiden Kamerapositionen I_r und I_k mit den zugehörigen Kamerazentren C_r und C_k. Die Positionen im Raum dieser Kameras wurden im vorherigen Schritt mit Hilfe von SVO ermittelt. T_k, r zeigt dabei die starre Transformation der Kamerabilder.
Der Punkt r_P ist die aktuelle Schätzung der Position eines Punktes auf den epipolaren Flächen. Die Varianz der Abweichung von einem Pixel auf der epipolaren Linie durch estrich und ustrich wird berechnet mit der Gleichung tkquadr. Mit Hilfe der oben angesprochenen mathematischen und statistischen Auswertungen kann nun

die Tiefe eines Punktes r_p approximiert werden.
Das Zusammenspiel von SVO und REMODE ist für handelsüblichen Laptops mit CPU und GPU ausgelegt. Dabei läuft SVO komplett auf dem Prozessor und REMODE verwendet das Framework NVIDIA CUDA, (footnote) was auf den Grafikchip des Rechners zugreift.
Die Implementation setzt auf eine durchschnittliche Bewegung der Kamera von 0.0038 Meter pro Sekunde und einer mittleren Tiefe von einem Meter bei einer Berechnungszeit von 3.3 ms pro Bild.

3.2.4 Performanceprobleme

Trotz der Nutzung neuster Methoden und Techniken zur Berechnung der Kamerapositionen und Tiefenbilder, gibt es Probleme hinsichtlich der Performance. Dabei ist das Hauptproblem die Differenz im verfolgten Ziel zwischen dieser Arbeit und der Implementierung von SVO und REMODE.
Der Hauptanwendungszweck ist ein langsamer, stetiger Flug einer Drohne über ein Gebiet. Dabei zeigt die aufnehmende Kamera nach unten und hat sowohl eine sehr hohe Qualität, als auch ein großes Sichtfeld von mehr als 110 Grad. Die Bewegungen der Drohne sind nur nach seitlich, nach vorne und nach hinten, nicht jedoch um die eigene Achse. Somit wird sicher gestellt, dass immer genügend Referenzfeatures vorhanden sind, damit zu jedem Zeitpunkt die Position der Kamera bekannt ist.
Im Gegensatz dazu sind die Anforderungen der Arbeit stark abweichend. So ist sowohl die Qualität, als auch die Verarbeitung der Kamera minderwertig. Auch das Blickfeld ist mit 90 Grad deutlich zu klein, wodurch weniger Features auf einem Bild Platz finden.
Dadurch, dass die Kamera nach vorne und nicht nach unten gerichtet ist, treten weitere Komplikationen auf. Gleiche Bewegungen verursachen somit größere Änderun-

gen am Bild, wodurch mehr Berechnungen notwendig werden und somit die Fehleranfälligkeit steigt. Vor allem Drehbewegungen um die eigene Achse sorgen für erhebliche Änderungen und führen meist zum Abbruch von SVO.
Auch die Kalibrierung der Kamera der realen Drohne war auf Grund der schlechten Qualität schwierig. So konnte der Richtwert des Reprojektionsfehlers von rund 0,1 Pixel nicht erreicht werden, sondern lag eher bei 0,3 Pixel.
Bei Tests in Innenräumen mit der AR.Drone konnte SVO nicht mehr als 50 Features finden. Um jedoch die Position der Kamera zu bestimmten, sind mehr als 100 Features notwendig. Diese Beobachtungen aus den Testläufen decken sich mit den Hinweisen zur Performance in der Projektdokumentation[7].

3.3 Implementierung

3.3.1 Grundlegende Herausforderungen

Es bestand eine nicht modularisierte, schlecht dokumentierte C# Dokumentation für eine Windowsumgebung. Diese galt es teilweise für ROS zu übernehmen. Für Windows existiert ein Kinect Software Development Kit, welches die Programmierung erleichtert.

3.3.2 Kinect

3.3.3 ROS Nodes

3.3.4 Architektur

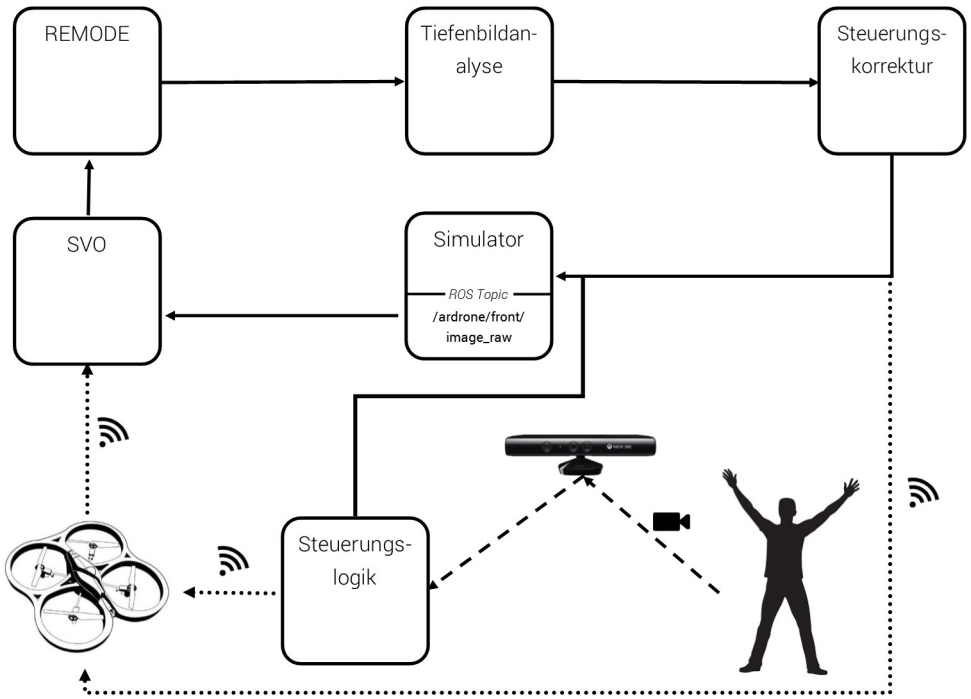


Abbildung 3.4: Übersicht zur Softwarearchitektur

3.4 Assistenzsystem

Im Kapitel 3.2 wurde der Prozess beschrieben, wie aus den monokularen Aufnahmen der Frontkamera der Drohen Tiefenbilder ermittelt werden können. Aufbauend darauf soll nun erarbeitet werden, wie man aus diesen Informationen ein grundlegendes Assistenzsystem implementieren könnte.

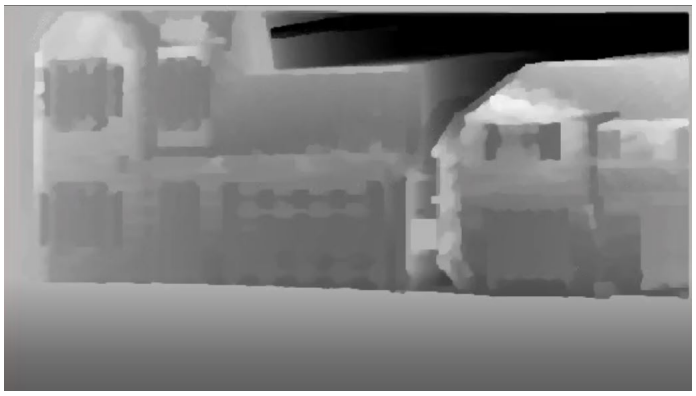
3.4.1 Abgrenzung

Die Tiefenbilder die REMODE erstellt sind in Form von sogenannten Punktwolken, bzw. Point Clouds über ein ROS Topic verschickt. Diese Punktwolken sind eine Men-

ge von Punkten in einem Vektorraum, welche jeweils durch ihre Raumkoordinaten in einem dreidimensionalen kartesischen Koordinatensystem beschrieben sind. Somit ist jedes Element im Datensatz durch die Attribute X, Y und Z gekennzeichnet.

Die Darstellung zeigt beispielhaft die Visualisierung einer Punktwolke. Dabei entscheidet die Helligkeit der Punkte über die relative Tiefe in Bezug zur Kamera. Der Vektorraum $V \in \mathbb{R}^3$ mit den entsprechenden Datenpunkten wird über ROS Topics in einem Intervall von wenigen Sekunden übertragen.
Wie zuvor beschrieben, ist ein einfacher Anwendungsfall das Fliegen durch Hindernisse wie offene Türen. Dabei stößt man in der Bildverarbeitung auf eine Reihe von Herausforderungen. Wie in 3.2.4 beschrieben, sind SVO und REMODE nicht optimal für den Anwendungsfall dieser Arbeit. Dabei treten Probleme schon beidem ersten Schritt zum Assistenzsystem auf, die Analyse des Tiefenbildes. Dabei ist sowohl die niedrige Qualität der Tiefenbilder problematisch, als auch die hohe Zeit die zwischen der Bewegung der Drohne und der Berechnung der Tiefenpunktwolke

vergehen kann.
Die Folgende Darstellung zeigt ein Beispiel für ein von REMODE berechnetes Bild.



Dabei wird ersichtlich, das der grundlegende Kontext für einen menschlichen Betrachter verständlich ist - in diesem Fall sind dies zwei Häuser mit einer schmalen Lücke als Zwischenraum. Dabei sind genaue Texturen stark reduziert und teilweise verloren gegangen. Weiterhin sind Objektränder zum Teil von einem leichten hellen Schimmer umgeben, welcher in der Bildanalyse zu Problemen führen kann. Um an diesem Punkt ein Objekt wie eine Tür, bzw. den Zwischenraum zu erkennen, gibt es verschiedene Ansätze, um die Punktwolke zu analysieren. Eine Möglichkeit ist das suchen nach definierten Abmessungen, Abständen und Formen. Es ist auch möglich, die Daten mit einer Referenzpunktwolke von einem ähnlichen Objekt abzugleichen. Für beide Ansätze sind komplexe mathematische Berechnungen und Bildverarbeitungskenntnisse notwendig. Für dieses Problem gibt es das OpenSource Projekt Point Cloud Library *PCL*. Dabei handelt es sich um ein Open Source Projekt zur 2D und 3D Bild-, sowie Punktwolkenverarbeitung. Dabei stellt die Programmbibliothek zahlreiche Algorithmen bereit, die z.B. bei der Filterung, Segmentierung und Visualisierung helfen. Dabei verfügt das Robot Operating System außerdem eine Schnittstelle zur PCL, wodurch sie sich bestens in die Projektumgebung dieser Arbeit integrieren lässt.

3.4.2 Implementierungsvorschlag

4 Evaluation

4.1 Ergebnis

Insgesamt kam es im Verlauf der Arbeit zu einer Vielzahl von unerwarteten Schwierigkeiten und Verzögerungen. Die Anfangsphase war hauptsächlich von technischen Hardwareproblemen geprägt. Dabei war zuerst die Drohne defekt und anschließend ist die Grafikkarte des Laptops ausgestiegen.
Auch softwareseitig kam es zu Verzögerungen. Die externen Projekte SVO und REMODE setzen eine sehr spezielle Umgebungskonfiguration voraus. So musste durch einen iterativen Prozess die richtige Kombination aus Betriebssystemversion, ROS Distribution und Grafikkartentreiber herausgefunden werden. Hinzu kommen eine Vielzahl von nötigen Dependencies, wie beispielsweise NVIDIA CUDA.
Auch die Kalibrierung der Kamera und die Konfiguration der Parameter war eine Herausforderung. Dies war das Resultat der Abweichenden Projektbedingungen, wie die nach vorne gerichtete Kamera, oder die simulierte Drohne.
Trotzdem konnten am Ende erfolgreich Tiefenbilder aus der Simulation gewonnen werden. Die Anforderung, die Gestensteuerung der Drohne von C# unter Windows auf C++ unter Ubuntu zu migireren konnte ebenso erfolgreich umgesetzt werden.
Als Folge der Verzögerungen war der Zeitraum für die Implementation eines Assistenzsystems sehr klein. Damit war es nur noch möglich das Thema theoretisch auszuarbeiten.

4.2 Ausblick

4.2.1 Andere Simulatoren

Die Simulationsumgebung Gazebo ist nicht die einzige verfügbare zur Simulation von Quadrocoptern. Jedoch bringt sie durch die einfache Anbindung von ROS einiges an Vorteilen mit sich. Damit gehen allerdings auch Nachteile einher. So sind die Kameraeingaben nicht sehr realistisch und Szenarien die in der Simulation funktionieren müssen in der Realität nicht funktionieren. Ebenso ist das Flugverhalten in manchen Situation nicht realitätsgetreu und kann zu verfälschten Ergebnisse führen. Um diesem Vorzubeugen ist es ratsam die Resultate mit anderen Simulationen vergleichen. Eine aktuelle Simulationsumgebung zur Simulation von Quadrocoptern ist der AirSim von Microsoft.[8] Ursprünglich entwickelt um Trainingsdaten zum maschinellem Lernen sammeln, kann er ebenfalls auch für herkömmliche Simulation verwendet werden. Aktuell ist allerdings nur für Windows Betriebssysteme verfügbar. [9] Er biete eine fotorealistische Umgebung und ein akkurates Flugverhalten, dadurch ist besonders für Demos und Showcases besser geeignet.
Es existieren weiterhin andere Simulatoren, allerdings sind die meisten spielerisch veranlagt und bieten keine programmatische Schnittstelle, weshalb sie für den Zweck der Studienarbeit nicht sinnvoll verwendet werden könne

4.2.2 Assistenzsystem

Literaturverzeichnis

[1] "Parrot AR.Drone." https://www.parrot.com/fr/sites/default/files/styles/product_teaser_hightlight/public/parrot_ar_drone_gps_edition.png?itok=OshlzcXW, 2017. visited: 01.22.2017.

[2] "Lochkamera Prinzip." https://en.wikipedia.org/wiki/Pinhole_camera_model#/media/File:Pinhole-camera.svg, 2017. visited: 03.22.2017.

[3] "Lochkamera Prinzip." https://github.com/uzh-rpg/rpg_open_remode/wiki/Run-Using-SVO, Dec. 2015. visited: 02.15.2017.

[4] "Parrot AR.Drone." https://de.wikipedia.org/wiki/Parrot_AR.Drone, 2016. visited: 01.22.2017.

[5] "Realtime code in ROS." http://www.willowgarage.com/blog/2009/06/10/orocos-rtt-and-ros-integrated, 2016. visited: 01.22.2017.

[6] Robotics and Perception Group, "Camera Calibration." https://github.com/uzh-rpg/rpg_svo/wiki/Camera-Calibration, 2014. visited: 02.19.2017.

[7] "Perfomance Dokumentation SVO." https://github.com/uzh-rpg/rpg_svo/wiki/Obtaining-Best-Performance, June 2014. visited: 02.18.2017.

[8] Shital Shah and Debadeepta Dey and Chris Lovett and Ashish Kapoor, "Microsoft AirSim." https://github.com/Microsoft/AirSim, 2017. visited: 03.16.2017.

[9] S. Shah, D. Dey, C. Lovett, and A. Kapoor, "Aerial Informatics and Robotics platform," Tech. Rep. MSR-TR-2017-9, Microsoft Research, 2017.