Julius-Maximilians-Universität Würzburg
Fakultät für Mathematik und Informatik

Informationstechnik für Luft- und Raumfahrt
Lehrstuhl für Informatik 8 Prof. Dr. Sergio Montenegro

Bachelorarbeit

Kooperative Steuerung multipler Quadrokopter mittels optischem Tracking

Vorgelegt von

Arthur Scharf

Matr.-Nr.: 1837155

Prüfer:	Prof. Dr. Sergio Montenegro
Betreuende wissenschaftliche Mitarbeiter:	Dipl.-Ing. Nils Gageik
	Qasim Ali

Würzburg, 11. Mai 2015

Erklärung

Ich versichere, dass ich die vorliegende Arbeit einschließlich aller beigefügter Materialien selbstständig und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe.
Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten oder unveröffentlichten Werken entnommen sind, sind in jedem Einzelfall unter Angabe der Quelle deutlich als solche kenntlich gemacht. Die Arbeit ist in gleicher oder ähnlicher Form noch nicht als Prüfungsarbeit eingereicht worden.
Mir ist bekannt, dass Zuwiderhandlungen gegen diese Erklärung und bewusste Täuschungen die Benotung der Arbeit mit der Note 5.0 zur Folge haben kann.

Würzburg, 11. Mai 2015

_____

Arthur Scharf

Aufgabenstellung

Die Fortschritte im Bereich Sensorik, Aktuatorik und Mikrotechnik ermöglichen heutzutage den kostengünstigen Bau kleiner, unbemannter Luftfahrzeuge (UAV, unmanned aerial vehicle, Drohne) wie Quadrokopter. Zur Positionsbestimmung im Labor wird oftmals optisches Tracking (OTS) eingesetzt, ein externes Kamerasystem, mit dem Marker und somit die Position von Quadrokoptern Millimeter genau bestimmt werden können.

Der Lehrstuhl Informationstechnik für Luft- und Raumfahrt der Universität Würzburg forscht ebenfalls an Quadrokoptern und hat dazu das System AQopterI8 mit dem Forschungsschwerpunkt Autonomie entwickelt. Im Rahmen des von der IHK ausgezeichneten Projekts Rettungshelfer mit Propellern wird ein Quadrokopter mit Manipulator entwickelt, der im Feuerwehreinsatz die Einsatzkräfte unterstützen kann. Ein Ziel dabei ist es, dass multiple Quadrokopter gemeinsam eine Aufgabe erfüllen, wozu einzelne Quadrokopter nicht in der Lage sind.

Das aktuelle System kann mit Hilfe multipler Sensoren autonom operieren. OTS wird dazu bisher nicht verwendet. Zu Testzwecken und zum Einstellen der Steuerung und Regelung des komplexen Systems samt Manipulator, ist es von Vorteil, bei der Entwicklung und Forschung auf ein hochgenaues System wie das OTS zurückgreifen zu können.

Hauptaugenmerk dieser Arbeit ist die simultane Steuerung multipler Quadrokopter (zumindest zweier Quadrokopter) mittels OTS zur kooperativen Lösung eines Problems. Dazu gehört das Einstellen bzw. Modifizieren der bisherigen Regelung zur Benutzung von OTS, sowie die Kommunikation der Quadrokopter untereinander. Dabei kann und soll auf bisherige Arbeiten aufgebaut werden. Am Ende soll das Gesamtsystem ein Problem, wie zum Beispiel das Absuchen eines Suchgebietes nach einem Zielobjekt, kooperativ mit Hilfe multipler Quadrokopter lösen können. Zur Arbeit gehört eine ausführliche Dokumentation.

Aufgabenstellung (stichpunktartig):

- Einarbeitung & Konzeptionierung (Wahl Problem und Lösungsmethode)
- Positionierung mittels OTS (Integration in bestehendes System)
- Implementierung: Kommunikationslösung zwischen den Quadrokoptern
- Implementierung: Algorithmus zur Kooperativen Lösung des gewählten Problems (Durchführung darf (teilweise) simuliert sein)

Zusammenfassung

Das Thema dieser Arbeit ist die Entwicklung und Durchführung bzw. Simulation eines Konzepts zur kooperativen Suche mit Hilfe zweier Quadrokopter. Dazu wird das vorhandene optische Tracking in das das bestehende System so integriert, dass eine Positionsregelung mittels optischem Tracking möglich ist. Zusätzlich wird, da für die Kooperation der Quadrokopter eine Kommunikationslösung notwendig ist, eine Kommunikationsarchitektur bestehend aus *Groundstation*, *Master* und *Slave* implementiert. Für die kooperative Suche werden dabei zwei Ansätze gewählt: das kooperative Absuchen von Wegpunkten, sowie das kooperative Absuchen einer Fläche. Durch die Kooperation müssen diese beiden Suchen auf die beiden Quadrokopter verteilt werden, was für das Absuchen von Wegpunkten mittels eines eigenen Algorithmus, und für das Absuchen der Fläche durch die Nutzung und Modifikation bereits bestehender Suchalgorithmen realisiert wird. Die Suchalgorithmen sowie die Positionsregelung mittels optischem Tracking werden evaluiert.

Inhaltsverzeichnis

Abbildungsverzeichnis

1. Einleitung

Betrachtet man die Entwicklung autonomer Vehikel in den letzten Jahren, lässt sich ein immer schneller werdender Anstieg in der alltäglichen Verwendung solcher autonomen Roboter beobachten, sei es nun ein Lager-Roboter bei einem Versandunternehmen, ein vollkommen selbst fahrendes Auto inmitten von San Francisco oder auch eine Drohne, die ein Paket bis vor die Haustüre liefert.

Zum einen wird dies durch Vorstöße großer Firmen beschleunigt, die die Technik dieser autonomen Roboter erschwinglich machen, zum anderen ist die Forschung und Entwicklung solch autonomer Roboter ein stetig wachsendes Feld.

Auch der Lehrstuhl für Informationstechnik für Luft- und Raumfahrt der Universität Würzburg forscht bereits seit einigen Jahren im Bereich der Autonomie, speziell im Zusammenhang mit Quadrokoptern, sogenannte *Unmanned Aerial Vehicles* - oder kurz *UAVs*. Der Vorteil solcher Quadrokopter gegenüber herkömmlichen UAVs in Flugzeugform, wie man sie vom Militär kennt, besteht vor allem in ihrer extrem hohen Mobilität und Agilität, da ein Quadrokopter Ziele punktgenau anfliegen, und dort seine Position auch halten kann. Dadurch können Quadrokopter in Bereichen eingesetzt werden, die für Menschen im Allgemeinen unzugänglich oder gefährlich sind, um beispielsweise, wie bei einem in der Aufgabenstellung beschriebenen Feuerwehreinsatz, den Einsatzkräften einen schnellen Überblick über die Situation zu verschaffen, oder um bei einem sogenannten *Search-and-Rescue*-Einsatz bei der Suche zu helfen. Bei solchen Einsätzen spielt jedoch meistens die Zeit eine große Rolle, weil man im Allgemeinen einen möglichst schnellen Überblick haben will, bzw. ein großes Gebiet möglichst schnell absuchen will. Da ein Quadrokopter alleine normalerweise jedoch nur sehr begrenzte Ressourcen was Akkulaufzeit und Rechenleistung angeht, zur Verfügung hat, muss, um beispielsweise eine große effizient Fläche abzusuchen, eine größere Anzahl solcher Quadrokopter eingesetzt werden, was einige Herausforderungen an die Aufteilung der Flächesowie an das kollisionsfreie Absuchen des Suchgebiets stellt. Dieses Problem der kooperativen Suche soll in der vorliegenden Arbeit angegangen werden,

wobei für die Positionserfassung ein externes optisches Tracking System, kurz OTS, verwendet wird, um möglichst genaue Positionsdaten für die spätere Suche zur Verfügung zu haben.

Die Arbeit gibt dabei zunächst eine Einführung in aktuelle Techniken und Methoden, die zur Lösung eines solchen Problems notwendig sind. Danach wird eine konzeptuelle Lösung des Problems beschrieben, wobei zwischen einem kooperativem, kollisionsfreiem Abfliegen von Wegpunkten und dem kooperativen, kollisionsfreien Absuchen einer Fläche unterschieden wird. Das Kapitel zur Implementierung schildert dann die entwickelte Lösung des Problems und die dabei verwendeten Techniken und Methoden. Abschließend wird dieser Lösungsansatz evaluiert und diskutiert.

2. Stand der Technik

Dieses Kapitel gibt einen Überblick über aktuell verwendete Technologien und Verfahren in den Bereichen optisches Tracking, Kommunikationsnetzwerke und Architektur von Multi-UAV Systemen geben. Abschliessend werden einige algorithmische Ansätze zur Flugbahnplanung, sowie zur Aufteilung eines Suchgebiets vorgestellt.

2.1. Einführung in das optische Tracking

Das optische Tracking ist eine von vielen Trackingtechnologien, beispielsweise GPS-Tracking oder Inertialtracking, mit dessen Hilfe die Positions- und Lagedaten eines Objektes im dreidimensionalen Raum erkannt und mit entsprechender Software verfolgt werden können. Nach Tönnis [2010] unterscheidet man dabei zwischen zwei unterschiedlichen Funktionsprinzipien:

- *Inside-Out*-Tracking
- *Outside-In* Tracking

Das ***Inside-Out***-Tracking beschreibt dabei ein System, bei dem das zu verfolgende Objekt selber Sensoren bzw. Empfänger mitführt und mithilfe externer Ressourcen die eigenen Positions- und Lagedaten ermitteln kann [Menache, 2011]. GPS-Tracking- oder optische Trackingsysteme mit bewegter Kamera sind ein Beispiel für dieses Funktionsprinzip; beim GPS-Tracking erhält der GPS-Empfänger, der auf dem Objekt platziert ist, GPS-Daten und kann anhand dieser seine Position, und unter gewissen Voraussetzungen auch seine Lage errechnen. Bei optischen Trackingsystemen, bei denen eine oder mehrere Kameras auf dem sich bewegenden Objekt platziert sind, werden im Raum verteilte Marker verwendet, mit derer die eigene Lage und (relative) Position ermittelt wird [Tönnis, 2010].

Das ***Outside-In***-Tracking dreht das eben genannte Verhältnis von Sender und Empfänger um: Hier wird die Positions- und Lagebestimmung mittels externer Sensoren bestimmt, wobei das zu verfolgende Objekt eine, je nach System unterschiedliche Art von Marker trägt. Ein Beispiel

hierfür ist wiederum das optische Trackingsystem, bei dem diesmal jedoch die Kameras im Raum verteilt sind, und die Marker auf dem Objekt platziert sind [Menache, 2011; Tönnis, 2010].

Da für die vorliegende Arbeit ein Tracking-System essentiell ist, wird im Folgenden das zuletzt erwähnte *Outside-In*-System näher erläutert.

Ein optisches Trackingsystem arbeitet, wie der Name bereits aussagt, im optischen Bereich. Damit werden für die externen Sensoren meist herkömmliche CCD-Kameras verwendet, die alle an einem zentralen Rechner angeschlossen werden, der die Daten der Kameras interpretiert [Menache, 2011]. Die Kameras arbeiten dabei meistens im infraroten Bereich des Lichtspektrums (780nm bis 1400nm), was gegenüber Systemen, die mit sichtbarem Licht arbeiten, den Vorteil hat, dass der Sichtbereich der Kameras zusätzlich ausgeleuchtet werden kann, ohne dass der Nutzer gestört oder geblendet wird [Tönnis, 2010].

Die CCD-Kameras können dabei eine Auflösung von 128x128 Pixel bis hin zu 16 Megapixel haben. Die Auflösung der Kamera ist aber nicht die einzige Anforderung an das optische Trackingsystem, vielmehr spielt die Framerate, also die Anzahl der Bilder die die Kamera pro Sekunde machen kann, eine größere Rolle, vor allem wenn man sehr schnelle Bewegungen verfolgen möchte. Die Kameras müssen für den Einsatzzweck in einem Trackingsystem zudem synchronisiert werden, d.h. die Kameras müssen alle zur gleichen Zeit ein Bild machen können, um eine möglichst fehlerlose Kalkulation der Positionsdaten zu ermöglichen. Dazu werden die Kameras meist über ein externes, zusätzliches Kabel miteinander verbunden [Menache, 2011].

Eine weiterer wichtiger Aspekt ist die Positionierung der Kameras im Raum, da diese so aufgestellt werden sollten, dass sich immer mindestens die Bildbereiche zweier Kameras überschneiden. Um das Tracking jedoch nutzen zu können, muss das System zu Anfang kalibriert werden, das heißt, dass das System die genauen Position der Kameras im Raum auf irgendeine Art und Weise erfassen muss. Dies wird durch das sogenannte „Wanding“ erreicht, bei dem ein Objekt, meist ein Stab, ausgestattet mit einigen Markern im Sichtbereich der Kameras umhergeschwenkt wird. Dadurch kann die Tracking Software durch Kombination der Kamerabilder die exakte 3D-Positionen der Kameras berechnen, sofern die Abmessungen des „Wanding“-Stabes bekannt sind. Die Marker haben dabei, je nach System, unterschiedliche Eigenschaften wie spezielle Muster oder bestimmte Oberflächenmaterialien [Tönnis, 2010]. Meistens werden jedoch Marker verwendet, die eine sehr gut reflektierende Oberfläche haben, damit sie von den Kameras möglichst gut erkannt werden. Dazu werden in den meisten Fällen auch externe Lichtquellen verwendet, die oft

durch außen an den Kameras angebrachte LEDs realisiert sind, und die Umgebung zusätzlich mit infrarotem, oder im Falle des sichtbaren Lichts, rotem Licht ausleuchten [Menache, 2011].

Hat man nun ein so kalibriertes System, kann dieses Objekte, die mit Markern ausgestattet sind, erkennen und je nach Rechenleistung des Zentralrechners fast in Echtzeit verfolgen. Zu beachten ist dabei jedoch, dass jeder Marker von mindestens zwei Kameras gesehen werden muss, um eine Positionierung des Markers im 3D-Raum zu ermöglichen.

Wie sich in den letzten Absätzen herauskristallisiert hat, ist einer der Haupt-Nachteile des optischen Trackings der, dass eine stark kontrollierte Umgebung vorhanden sein muss und das Tracking nur innerhalb der Grenzen dieser Umgebung, d.h. der Bildausschnitte der Kameras, funktioniert. Zusätzlich müssen die Marker immer von mindestens zwei Kameras gesehen werden, was nicht in jedem Fall möglich ist. Auch ist die Rechenleistung, die für die Extraktion der Positionsdaten aus den gelieferten Bilddaten der Kameras benötigt wird, relativ hoch, und steigt auch mit jeder zusätzlichen Kamera, was einen Trade-Off zwischen Anzahl der Kameras und Sichtbarkeit der Marker nötig macht [Menache, 2011].

Nichtsdestotrotz bietet die Bewegungsverfolgung via optischem Tracking Vorteile, die von anderen Systemen so nicht realisierbar sind. Dazu gehört zum einen die sehr hohe Genauigkeit der Positionsdaten, beim Produkt *OptiTrack Motive* der Firma NaturalPoint liegt die Genauigkeit beispielsweise im Millimeter-Bereich [OptiTrack]. Ein weiterer Vorteil ist, dass sehr viele Marker getrackt werden können, und die Marker auch sehr leicht angebracht werden können. Dazu kommt, dass die Bildfrequenz bei optischem Tracking je nach verwendeter Kamera bei bis zu 120fps liegt, was zu einer hohen Anzahl von Positionsberechnungen pro Sekunde führt und das Tracken von sich schnell bewegenden Objekten ermöglicht [Menache, 2011].

Somit stellt das optische Tracking unter gewissen Voraussetzungen ein sehr genaues Positions- und Lageerfassungssystem dar.

2.2. Kommunikationsnetzwerke

In der vorliegenden Arbeit zur Kooperativen Steuerung mehrerer Quadrokopter stellt die Kommunikation zwischen den einzelnen Quadrokoptern ein zentrales Element dar, da ohne eine Kommunikationsstruktur ein kooperatives Verhalten nur schwer möglich ist.

Dieses Kapitel stellt daher die Grundlagen und das Referenzmodell der Netzwerkkommunikation, sowie einige wichtige, oft genutzte Kommunikationsprotokolle vor. Abschließend wird auf einige Übertragungsmedien und ein paar Technologien zur Vernetzung eines Netzwerks eingegangen.

2.2.1. ISO/OSI Schichtarchitektur

Das ISO/OSI-Schichtenmodell (**O**pen **S**ystem **I**nterconnection - zu deutsch „Offenes System für Kommunikationsverbindungen“) ist ein von der International Standardization Organisation, kurz ISO, im Jahr 1979 eingeführtes Referenzmodell für die Entwicklung von offenen Kommunikationsstandards bzw. -protokollen. [Alani, 2014]. Es schreibt dabei nicht vor, wie ein Netzwerk zu funktionieren hat, sondern unterteilt das Netzwerk in 7 sogenannte „Schichten“, die jeweils eine eigene Aufgabe oder Funktion erfüllen sollen. Es ist jedoch nicht verpflichtend, diese Schichten genau so einzuhalten, es können einzelne Schichten beispielsweise auch zusammengefasst oder weggelassen werden.Die Idee hinter einem solchen Schichtmodell ist, dass das Verständnis des Netzwerks und somit auch die Implementierung eines Netzwerkprotokolls vereinfacht wird, da die einzelnen Schichten ineinander konsistent und ihre jeweilige Funktion klar von den anderen Schichten abgesetzt ist. Dies erleichtert auch die Suche nach eventuellen Fehlern oder Störungen, da das Problem schneller eingegrenzt werden kann [Alani, 2014]. Dieser Flexibilität hat es das OSI Modell zu verdanken, dass es über 30 Jahre später immer noch angewandt wird. Nach Gessler und Krause [2009] Alani [2014] sind die 7 Schichten des OSI-Modells wie folgt definiert:

- Schicht 1 - **Physical Layer** (Bitübertragungsschicht)
  In dieser Schicht werden die Übertragung der einzelnen Bits über ein Übertragungsmedium spezifiziert und die physikalische Verbindung aktiviert bzw. deaktiviert. Das Medium an sich wird hier jedoch nicht definiert.

- Schicht 2 - **Data Link Layer** (Sicherungsschicht)
  Diese Schicht ist für die fehlerfreie Übertragung der Daten und für die Zugriffssteuerung auf den Übertragungskanal zuständig. Hier wird auch der Datenstrom paketiert bzw. depaketiert.

- Schicht 3 - **Network Layer** (Vermittlungsschicht)
  Die Vermittlungsschicht ist hauptsächlich für das Routing, also die Kontrolle des Datenflus-

ses zuständig. Es wird dabei zwischen verbindungsorierten und verbindungslosen Diensten unterschieden.

- Schicht 4 - **Transport Layer** (Transportschicht)
  In der Transportschicht wird eine logische Ende-zu-Ende-Verbindung hergestellt. Hier wird auch auf Fehlererkennung und -korrektur geachtet .

- Schicht 5 - **Session Layer** (Kommunikationssteuerungsschicht)
  Die Hauptaufgabe der Kommunikationssteuerungsschicht ist die Bereitstellung, Initiierung und Terminierung sogenannter „Sessions“, in denen die beteiligten Kommunikationsprozesse Daten senden empfangen können. Diese Schicht verwaltet dabei die „Sessions“ und kontrolliert so, welche Prozesse wann kommunizieren können.

- Schicht 6 - **Presentation Layer** (Darstellungsschicht)
  Hier werden die empfangenen bzw. zu sendenden Daten verschlüsselt, komprimiert und/oder übersetzt, so können die Daten beispielsweise in verschiedene Formate oder Codecs umgewandelt werden.

- Schicht 7 - **Application Layer** (Anwendungsschicht)
  Diese Schicht definiert die Verbindung zur Nutzeranwendung, wobei verschiedene anwendungsspezifische Anforderungen festgelegt werden, beispielsweise die Verschlüsselung der Nachrichten.

Die meisten Kommunikationsprotokolle orientieren sich an diesem Referenzmodell, da durch die Definition der Schichten eine Kombination verschiedener Protokolle möglich ist, die somit auch aufeinander aufbauen können. Abb. 2.1 zeigt eine solche Schichtung, bzw. Kapselung verschiedener Protokolle am Beispiel des IP-Protokolls. Einige der wichtigen Protokolle werden nun im folgenden Kapitel genauer erläutert.

2.2.2. Kommunikationsprotokolle

Baun [2012] beschreibt ein Protokoll als ein Regelwerk, auf dem der Austausch von Informationen zwischen zwei oder mehr Teilnehmern in einem Netzwerk basiert. Dieses Regelwerk definiert dabei die Syntax, d.h. den Aufbau einer Nachricht, die Semantik, d.h. die Bedeutung einzelner

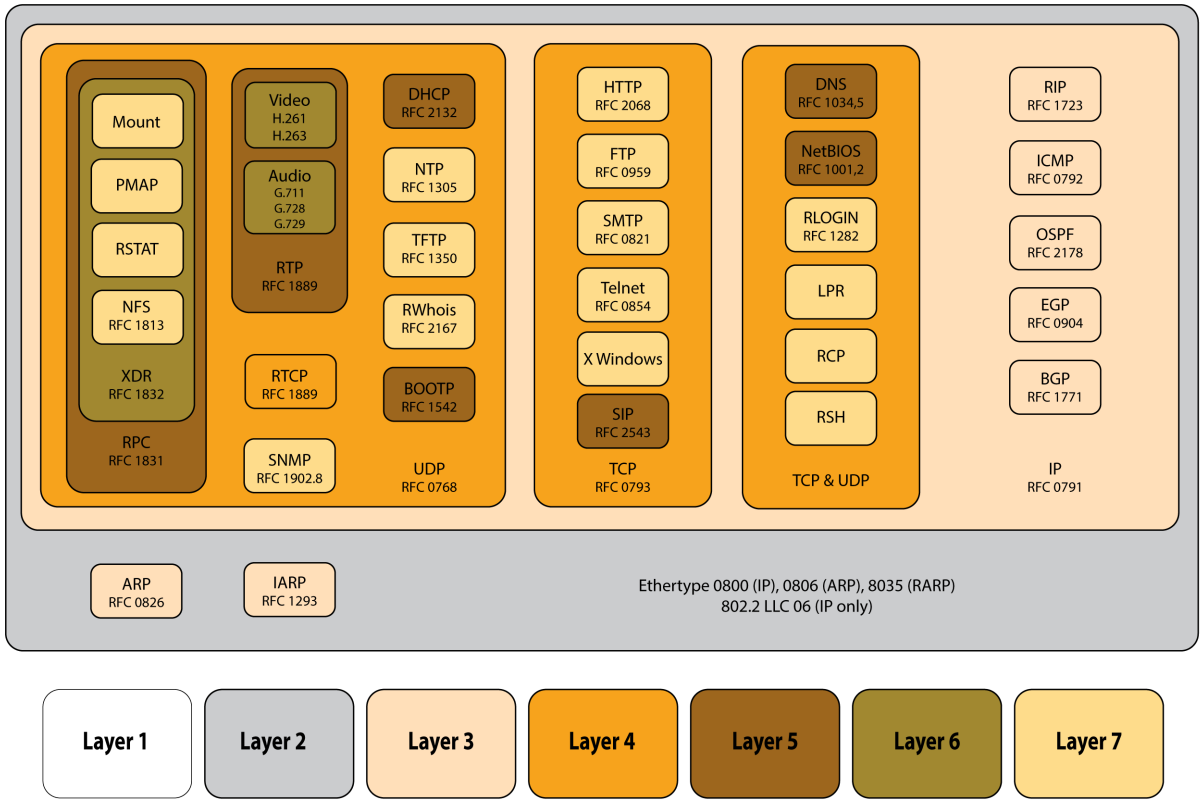

Abbildung 2.1.: Kapselung verschiedener Protokolle in der *Internet Protocol Suite* (Abb. angelehnt an eine Darstellung der Firma WildPackets)

Abschnitte der Nachricht, und die Synchronisation, die zeitliche Abfolge und das Aufeinander-Abstimmen der Nachrichten.

Betrachtet man nun ein Computernetzwerk, lässt sich schnell feststellen, dass es eine große Anzahl Kommunikations- bzw. Netzwerkprotokolle gibt, da sie im Allgemeinen unterschiedliche Zwecke erfüllen. So definiert beispielsweise das Protokoll *1000-BASE-T* die Übertragung von Ethernet-Frames über ein Kupferkabel, und das Protokoll *NTP* die Synchronisation von Uhren in Computern über ein Kommunikationsnetzwerk.

Es gibt jedoch auch Protokolle, die essentiell für die meisten Arten von Netzwerkkommunikation sind, da sehr viele Anwendungen darauf aufbauen. Nach Scherff [2010] und Alani [2014] handelt es sich dabei um TCP, UDP und IP, drei Protokolle, die die Basis der sogenannten *Internet Protocol Suite*, ein Überbegriff für Protokolle, die auf IP aufbauen, bilden und auch als die drei "Basisprotokolle"bezeichnet werden.

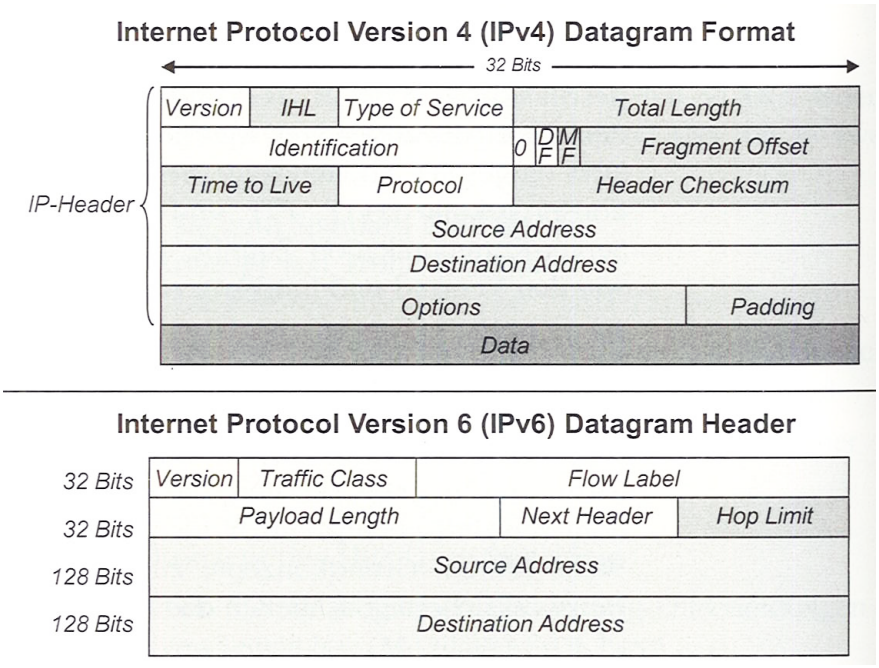

Abbildung 2.2.: Aufbau eines IP-Headers (IPv4 und IPv6) [Scherff [2010], Abb. 9.1]

- **IP - Internet Protocol**

  Das IP-Protokoll ist in der 3. OSI-Schicht, der Vermittlungsschicht angesiedelt. Die Hauptaufgabe des Protokolls ist dabei, Pakete von einem Netzwerkteilnehmer, im Folgenden Host genannt, zu einem anderen Host zu vermitteln. Dies geschieht durch Hinzufügen eines sogenannten *Headers* zu den eigentlichen Daten, der Adressierungs- und Kontrollinformation enthält. Abb. 2.2 zeigt den allgemeinen Aufbau eines solchen IP-Headers, wobei *Source Address* und *Destination Address* die jeweilige eindeutige logische Adressierung, die IP-Adresse des Senders bzw. Empfängers, bezeichnen [Alani, 2014]. Sollte ein Datenpaket die maximale Länge, die ein Netzwerk zulässt, überschreiten, kann ein Datenpaket mithilfe der Identification und Fragment - Flags , auf mehrere IP-Pakete aufgeteilt, fragmentiert und später wieder zusammengesetzt werden. Allerdings ist hier zu beachten, dass IP ein verbindungsloses Protokoll ist, das heißt, es wird nicht garantiert, dass die Pakete auch wirklich ihr Ziel erreichen. Es wird jedes IP-Paket unabhängig von anderen Paketen an die Ziel-Adresse vermittelt, was dazu führen kann, dass Pakete gar nicht, in falscher Reihenfolge oder sogar doppelt ankommen, da vor allem große Netzwerke (beispielsweise dem Internet) nicht statisch sind und die Übertragung der Datenpakete durch Faktoren wie Auslastung und/oder Verbindungsabbrüche beeinflusst wird.

- **TCP - Transmission Control Protocol**

  Um den genannten Problemen des IP-Protokolls zu begegnen, sind verbindungsorientierte Transportprotokolle notwendig, die sicherstellen, dass alle übertragenen Pakete angekommen sind, und die dazu fähig sind, die Pakete in der richtigen Reihenfolge wieder zusammenzusetzen. Das *Transmission Control Protocol* ist ein solches Protokoll, das im Gegensatz zu IP verbindungsorientiert arbeitet. Verbindungsorientiert bedeutet dabei, dass vor der eigentlichen Datenübertragung ein sogenannter *Threeway-Handshake* durchgeführt. Dieser bezeichnet ein mehrstufiges Authentifizierungsverfahren zwischen zwei Hosts, um sicherzustellen dass die nachfolgenden Pakete ihr Ziel erreichen. Dabei quittiert der Empfänger immer die vom Sender empfangenen Pakete, sodass ein eventuell unterwegs verlorengegangenes Paket erneut gesendet werden kann, da der Sender durch das Ausbleiben der Quittierung den Paketverlust bemerkt. Auf den genauen Ablauf einer TCP-Kommunikation wird im Folgenden jedoch nicht weiter eingegangen.

- **UDP - User Datagram Protocol**

  Es gibt jedoch auch Fälle bei denen die gesicherte Übertragung nicht erforderlich ist, bzw. die ständige Quittierung und Aufrechterhaltung der gesicherten Verbindung zu langsam für eine bestimmte Anwendung ist, beispielsweise bei einem Video-Stream. Hier kommt das UDP Protokoll ins Spiel.

  UDP ist wie das IP-Protokoll verbindungslos und garantiert nicht eine erfolgreiche Datenübertragung. Im Grunde ist UDP eine einfachere Version des TCP-Protokolls, da bei UDP die Sequenzierung, Quittierung und die Flags wegfallen (siehe Abbildung), aber die übertragenen Daten dennoch mit einer Prüfsumme auf Korrektheit geprüft werden [Scherff, 2010]. Eine solche verbindungslose Datenübertragung wird, wie bereits erwähnt, oft bei z.B. Video-Streams genutzt, da hier ein fehlendes Daten-Paket keine großen Auswirkungen hat und eine Neu-Anforderung des Pakets wertlos wäre, da das Video mit den restlichen, erfolgreich übertragenen Paketen bereits weiterläuft. Das neu angeforderte Paket würde in diesem Falle zu spät ankommen und hätte keine Verwendung mehr.

2.2.3. Kommunikationsmedien und Vernetzungstechnologien

Damit die im letzten Kapitel beschriebenen Transport- und Vermittlungsprotokolle jedoch überhaupt zum Einsatz kommen, braucht man Standards und Spezifikationen auf der Sicherungs- und Bitübertragungsschicht, den OSI-Schichten 1 und 2, die das Übertragungsmedium und verwendete Protokolle genau beschreiben. Übertragungsmedien lassen sich in zwei große Kategorien einteilen, die kabelgebundenen und die kabellosen Medien. Zu den kabelgebundenen Übertragungsmedien gehören dabei die elektrischen Leiter, sowie die Lichtwellenleiter, umgangssprachlich auch als Glasfaserkabel bekannt. Basierend auf diesen drei grundlegenden Übertragungsmedien spezifiziert *IEEE-802*, ein von der *Institute of Electrical and Electronics Engineers Standards Association* im Jahr 1980 ins Leben gerufene Projekt um Netzwerkstandards auf OSI-Schichten 1 und 2 zu definieren, verschiedene technische und elektrische Anforderungen, um durch eine solche Standardisierung Interoperabilität zwischen Geräten verschiedener Hersteller zu gewährleisten [Alani, 2014]. Der wohl bekannteste, von diesem Projekt entwickelte Standard, ist IEEE 802.3, auch oft als *Carrier Sense Multiple-Access with Collision Detection*, kurz CSMA/CD, oder Ethernet bezeichnet. Die ursprüngliche Idee hinter dieser Technologie war, dass mehrere Teilnehmer in einem gemeinsamen Leitungsnetz, einem sogenannten *Local Area Network*, kurz LAN, über hochfrequente Signale Nachrichten untereinander austauschen können. Die Basis dafür bildet der CSMA/CD Algorithmus, mit dessen Hilfe eine störungsfreie Nutzung eines Basisbands, d.h. einer Leitung, die nur einen Kanal bietet, möglich ist. Mittlerweile haben sich jedoch sehr viele Varianten des ursprünglichen Ethernets entwickelt, so zum Beispiel 10GBASE-SR, eine 10GBit/s-Verbindung über ein Glasfaserkabel mit Multimode-Fasern [Baun, 2012].

Die verschiedenen Ethernet-Varianten und deren technischen Details der Übertragungsverfahren sind jedoch nur von geringer Relevanz für diese Arbeit und werden im Folgenden nicht näher behandelt.

Kabellose Kommunikationsstandards hingegen spielen eine größere Rolle. Wie bei kabelgebundener Kommunikation müssen auch hier technische und elektrische Aspekte spezifiziert, bzw. standardisiert werden, um Kommunikation zwischen verschiedensten Geräten zu ermöglichen. Und auch hier ist das bereits vorgestellte IEEE-802 Projekt von großer Bedeutung, da sich, analog zu Ethernet, verschiedene Arbeitsgruppen wie beispielsweise 802.11 mit der Standardisierung von kabelloser Kommunikation beschäftigen. Eine Auswahl solcher kabelloser Übertragungs-

technologien, die für die Kooperation multipler Quadrokopter relevant sind, wird jedoch erst in Kapitel 2.3.3 gegeben.

2.3. Multiple UAV's

Dieses Kapitel gibt zunächst einen Einblick in die verschiedenen Klassifizierungen von Multi-UAV-Architekturen und Konstellationen. Darauf aufbauend der aktuelle Stand der Technik zur verwendeten Hardware und Sensorik, auch unter dem Gesichtspunkt der Kollisionsvermeidung, erläutert werden. Abschließend wird auf aktuelle Techniken zur Kommunikation zwischen den einzelnen UAVs einer Multi-UAV-Konstellation, im Folgenden als *Agent* bezeichnet, eingegangen.

2.3.1. Klassifizierung von Multi-UAV Architekturen

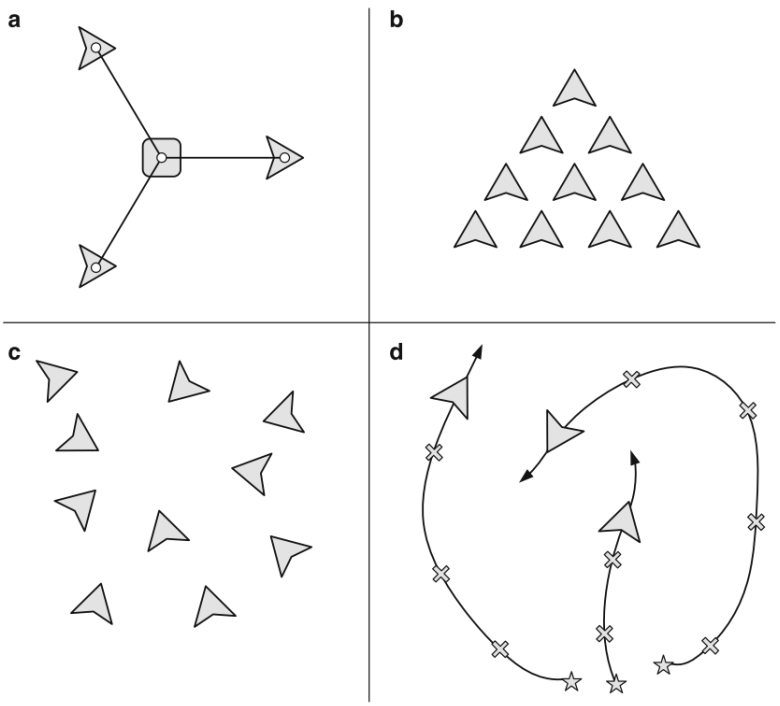

Abbildung 2.3.: Graphische Darstellung der möglichen Klassifizierung nach Kopplungsgrad: **a)** Physische Kopplung - UAVs transportieren ein Objekt, **b)** Formationsflug **c)** Schwarm von UAVs, **d)** Intentionale Kooperation - UAVs erfüllen eine Zielvorgabe (Abflug der Kreuze) - (Maza et al. [2014], Fig 38.1)

Systeme bestehend aus mehreren UAVs lassen sich, je nach Sichtweise, unterschiedlich klassifizieren. Eine mögliche Art ist, die Kopplung, bzw. den Grad der Kopplung zwischen den einzelnen UAVs eines Systems, zu betrachten. Dadurch lassen sich nach Maza et al. [2014] vier verschiedene Schemata definieren, die in Abb. 2.3 illustriert sind:

1. **Physische Kopplung**

   Hier sind die einzelnen Agenten des Systems physisch miteinander verbunden, z.B. im Zuge einer Zusammenarbeit bei der schwere Lasten gemeinsam von mehreren UAV gehoben oder transportiert werden. Eines der größten Probleme bei dieser Art von Kopplung ist die Koordination und Kontrolle der Bewegungen der UAVs, da zusätzlich zu der eigenen hochgradig nicht-linearen Flugdynamik eines UAVs, Krafteinwirkungen durch die verbunden UAVs entstehen, die bei der Betrachtung der Flugdynamik eine Rolle spielen.

2. **Formationen**

   Hierbei sind die Flugkörper nicht physisch miteinander verbunden, aber aufgrund der Bedingung ihre Formation zu halten, sind die erlaubten relativen Eigenbewegungen der Agenten zueinander stark eingeschränkt. Dadurch lässt sich die Formation als ein einzelner starrer Körper betrachten (vgl. *Virtual Structure*), was in Abb. 2.3b) sehr gut deutlich wird.

3. **Schwärme**

   Schwärme bestehen meist aus vielen homogenen Agenten, wobei der Kopplungsgrad ähnlich dem bei Formationen ist, da sich die Bewegungen eines einzelnen Agenten auf das gesamte Kollektiv auswirken. Aufgrund der hohen Anzahl an Agenten ist hier eine dezentralisierte Kontrollarchitektur gemeinhin unumgänglich. Der Unterschied zur Formation besteht jedoch darin, dass die Agenten keinen „starren“ Körper bilden, die Form des Schwarms sich also dementsprechend beliebig ändern kann, siehe Abb. 2.3c.

4. **Intentionale Kooperation**

   Bei dieser Art von Kopplung steht die Erfüllung einer globalen Mission im Vordergrund. Die einzelnen Bewegungen eines Agenten wirken sich, im Gegensatz zu Schwärmen oder Formationen, nicht unmittelbar auf die Bewegungen der anderen Agenten aus, da jeder Agent eine individuelle Aufgabe und eine eigene Trajektorie besitzt. Bei dieser Art von Kopplung besteht das Problem darin, dass meist komplexe Aufgaben- und Trajektorieplanungen sowie Möglichkeiten zur Konfliktlösung und -vermeidung notwendig sind.

Eine weitere Möglichkeit Multi-UAV-Systeme zu klassifizieren, ist die Betrachtung der Konstellations- bzw. Kontrollarchitektur des Systems. Nach Ali et al. [2014] lassen sich die verschiedenen Konstellationen in drei Hauptkategorien einteilen: „Leader-Follower“, „Virtual Structure“ und der sogenannte „Behavioural Approach“.

Die **„Leader-Follower“**-Struktur ist dabei, bei einer überschaubaren Menge von Agenten, nach Kumar [2015] die am Meisten verwendete Konstellation in der Forschung, da hier eine zentralisierte Kontrolllösung möglich ist, und diese in den allermeisten Fällen einfacher zu implementieren ist als eine dezentralisierte Lösung [Ali et al., 2014]. Das „Leader-Follower“-Prinzip ist leicht verständlich und auch in der Natur, beispielsweise bei Zugvögeln zu beobachten. Dabei ist einer oder mehrere Agenten der *Leader*, der sowohl Richtung als auch Geschwindigkeit der gesamten Konstellation vorgibt. Die *Follower* folgen dabei jeder Bewegung des *Leaders*. Aus technischer Sicht vereinfacht dies die Systemmodellierung und Kontrolle einer solchen Konstellation, weil die Zustände des *Leaders*, in der Literatur auch als Koordinationsvariable bezeichnet, alle Bewegungen der *Follower* definieren [Beard et al., 2006].

In der Konstellation **„Virtual Structure“** wird, ähnlich wie bei der Leader-Follower-Struktur, ein *Leader* genutzt, der hier jedoch rein virtuell ist. Dabei werden alle Agenten als eine virtuelle Einheit, ähnlich einer Formation betrachtet, was ebenfalls die Systemmodellierung vereinfacht, da der virtuelle Leader auch hier die Zustände der Agenten definiert. Der Vorteil gegenüber des *Leader-Follower*-Ansatzes ist hierbei, dass der virtuelle Leader nicht verloren gehen, oder abstürzen kann.[Ali et al., 2014; Li und Liu, 2008].

Die dritte Art von Konstellationsarchitektur, der **„Behavioral Approach“** basiert auf der kooperativen Spieltheorie, wobei die Kontrollfunktion eines jeden Agenten eine gewichtete Funktion der gewünschten Verhaltensweisen darstellt [Ali et al., 2014] . Diese gewünschten Verhaltensweisen, nämlich Kollisionsvermeidung, Hindernisvermeidung, Zielsuche und Halten der Formation, bilden dabei die Vektoren der Kontrollfunktion, wobei die Gewichtung der einzelnen Vektoren abhängig von der Anwendung ist [Anderson und Robbins, 1998].

2.3.2. Hardware und Sensorik

Der Aufbau und die Aufrechterhaltung einer der im vorangegangenen Kapitel beschriebenen Konstellationen stellt jedoch auch einige Anforderungen an die Hardware-Ausstattung, speziell an die mitgeführte Sensorik der einzelnen UAVs in der Konstellation. So müssen die einzelnen UAVs fähig sein, ihre Umgebung wahrzunehmen und ihre eigene Position und Lage zu bestimmen, um anhand dieser Informationen z.B. Kollisionsvermeidung zu betreiben. Im Folgenden werden einige der typischerweise verwendeten Sensoren, die für die Kollisionsvermeidung und Hinderniserkennung, sowie zur Positions- und Lagebestimmung notwendig sind, vorgestellt.

2.3.2.1. Kollisionsvermeidung und Hinderniserkennung

Für UAVs die in einer Konstellation, wie sie in Kapitel 2.3.1 beschrieben ist, zusammenarbeiten müssen, ist eine zuverlässige Erkennung von Hindernissen oder anderen UAVs unentbehrlich um Zusammenstöße und in letzter Konsequenz Abstürze zu vermeiden.

Um dies zu vermeiden, können Sensoren eingesetzt werden, die die 360°-Umgebung eines UAV in Echzeit beobachten können. Dazu gehören beispielsweise Ultraschallsensoren oder Laserscanner, wobei Ultraschallsensoren nur bis zu einer Entfernung von 2,5m anwendbar sind [Gageik et al., 2012].

Aber auch andere elektro-optische Sensoren wie Infrarotsensoren oder herkömmliche Kameras in Verbindung mit sogenannter *machine vision* bieten eine gute Grundlage, um Hindernisse zu erkennen [Ali et al., 2014].

Eine etwas modernere Technologie sind PMD-Kameras, die ähnlich einem Laserscanner ein räumliches Abbild der Umgebung erstellen können, allerdings den Vorteil haben den Raum nicht abtasten zu müssen, sondern mit nur einem „Bild“ der Umgebung sofort 3D Informationen zur Verfügung stellen können. Solch komplexe Technologien wie PMD oder *Machine Vision* brauchen jedoch noch relativ viel Rechenleistung, was die Flugzeit, speziell bei UAVs die nur mithilfe von Akkus betrieben werden, stark verkürzen kann. Daher sind Infrarot- oder Ultraschallsensoren aufgrund ihrer kleineren Energie- und Leistungsaufnahme aktuell eine gute Wahl [Ali et al., 2014]. Durch sogenannte *Sensor Fusion*, also Nutzung mehrerer Sensoren und Verschmelzung bzw. Verrechnung ihrer gelieferten Informationen lässt sich dabei die Zuverlässigkeit der Informationen über die Umgebung noch steigern.

Eine weitere Technologie, die zur Erkennung von Hindernissen und zur Vermeidung von Kollisionen bei UAVs genutzt werden können, ist RADAR bzw. LIDAR. Beide basieren dabei auf dem Prinzip, dass Radio- bzw. Lichtimpulse ausgesandt werden. Wird dieser ausgesandte Impuls von einem Hindernis zurückgeworfen, lässt sich anhand des zurückgeworfenen Anteils die Entfernung errechnen. Doch auch hier ist die Energie- bzw. Leistungsaufnahme noch sehr hoch, weshalb die beiden Systeme eher bei größeren Drohnen oder Flugzeugen zum Einsatz kommen [Mejias et al., 2014].

2.3.2.2. Lage- und Positionsbestimmung

Der am meisten verbreitete Sensor für die Lagebestimmung, ist die sogenannte *IMU*, eine inertiale Messeinheit, die Beschleunigungssensor und Gyroskop vereint. Mittels des Beschleunigungssensors, der die Beschleunigungen in allen drei Achsen misst, sowie dem Gyroskop, das die Drehraten um die drei Achsen misst, lässt sich durch Integrieren die Lage errechnen. Allerdings ist aufgrund der starken Drift des Beschleunigungsmesser und Gyroskops eine IMU nur sehr bedingt zur Positionsbestimmung geeignet.

Eine weitere sehr populäre Technik, GPS, wird oft zur Positionsbestimmung eingesetzt. Allerdings ist die Nutzung von GPS meist auf Außenanwendungen beschränkt, da die Signale der GPS-Satelliten in Gebäuden nur schlecht empfangen werden können. Dadurch wird die Positionsbestimmung mittels GPS innerhalb eines Gebäudes sehr ungenau.

Für Anwendungen im Innenbereich lassen sich auch die im vorangegangenen Kapitel beschriebenen Ultraschall- und Infrarotsensoren nutzen, da die beschränkte Reichweite dieser Sensoren im Innenbereich meist ausreichend für eine relative Positionsbestimmung ist. Durch die bereits erwähnte Technik der Fusionierung der Sensordaten und die Filterung beispielsweise mit einem Zustandsschätzer wie dem Kalmann-Filter lässt sich hier auch eine relativ hohe Zuverlässigkeit der von den genannten Sensoren gelieferten Information erzielen [Ali et al., 2014].

Eine weitere Methode, mit deren Hilfe die relative Positionsänderung bestimmt werden kann, ist das sogenannte Verfahren des optischen Flusses. Dabei werden mit einer Kamera zwei kurz aufeinanderfolgende Bilder aufgenommen. Anhand der Veränderung in den Bilddaten kann die Positionsänderung errechnet werden. Diese Methode benötigt jedoch eine relativ hohe Rechenleistung [Gageik et al., 2014].

2.3.3. Kommunikation zwischen UAVs

Die Kommunikation ist, abgesehen von der Hardware und Flugregelung, der wichtigste Punkt bei der Kooperation von UAVs. Nachfolgend sollen ein paar der meistgenutzten Kommunikationstechnologien vorgestellt und erläutert werden. Dabei wird jedoch die Kommunikation der Nutzlast des UAV, beispielsweise eine Videoübertragung an eine Groundstation, außer Acht gelassen.

2.3.3.1. Bluetooth

Bluetooth, spezifiziert durch IEEE 802.15.1, ist ein sehr populäres Funksystem zur Datenübertragung mit maximal 2,1 MBit/s bis zu einer Distanz von, je nach Spezifikation, 10 bis 100m, das im 2,4GHz-Band arbeitet [Smolka, 2013]. Die Datenrate von Bluetooth ist relativ gering, trotzdem ist Bluetooth sehr beliebt, da es es im Vergleich zu anderen Technologien relativ wenig Strom verbraucht und die Datenrate bei den meisten Anwendungen, z.B. senden/empfangen von Telemetriedaten komplett ausreicht. Mit dem aktuell neuesten Bluetooth-Standard 4.2 wird dabei der Stromverbrauch noch einmal gesenkt [Bluetooth-Spec], was Bluetooth zu einer attraktiven Option für Multi-UAV Konstellationen macht, bei denen die einzelnen Agenten relativ nah beieinander sind.

2.3.3.2. Wireless LAN

Eine Alternative zu Bluetooth, die nicht dessen Schwächen hat, ist ebenfalls eine der bekanntesten Technologien: das sogenannte *Wireless Local Area Network* - kurz WLAN, ein - wie der Name bereits andeutet - lokales Funknetz das bis zu einer Entfernung von 250m arbeitet [Gessler und Krause, 2009]. Die verschiedenen Versionen und Iterationen von WLAN sind durch IEEE 802.11 spezifiziert, und unterscheiden sich dabei durch die maximale Datenrate bzw. Protokollspezfikation und Modulationsverfahren. WLAN arbeitet dabei, je nach Version im 2,4 bzw. 5GHz-Bereich und hat, bei WLAN 802.11n, dem meistgenutzten Standard, eine maximale Datenrate von 600MBit/s. Ein weiterer Vorteil im Vergleich zu Bluetooth, ist die um einiges bessere kryptographische Verschlüsselung des Funknetztes, z.B. durch WPA oder WPA2, was WLAN bei sensitiviten Daten zu einer besseren Lösung als Bluetooth macht [Baun, 2012]. Allerdings gehen die genannten Vorteile zu Lasten des Stromverbrauchs, da WLAN aufgrund der höheren Reichweite und Datenrate mit einer höheren Sendeleistung sendet [Gessler und Krause, 2009]. Daher muss je nach Anwendung entschieden werden, ob man auf WLAN oder Bluetooth zurückgreift, da beides Vor- und Nachteile bietet.

2.3.3.3. Mobilfunknetze

Eine auf UAVs relativ wenig genutzte Technologie sind die Mobilfunknetzte. Mobilfunknetzte bieten eine extrem große, fast weltweite Netzabdeckung und sind daher fast überall nutzbar, al-

lerdings auf Kosten einer höheren Latenzzeit, was sie für Echtzeit-Anwendungen nur bedingt nutzbar macht. Für weit auseinander liegende Agenten bzw. UAVs wäre solch ein Einsatz jedoch denkbar. Alexander Mazur von der University of Western Australia hat im Oktober 2014 die Machbarkeit eines autonomen UAV-Flugs im Rahmen eines Master-Projekts demonstriert, bei dem ein Hexacopter, ausgerüstet mit einem Raspberry Pi und einer Internetverbindung via 3G/4G über eine Weboberfläche gesteuert werden konnte [Mazur, 2014].

2.4. Kooperative Missionplanung

Nachdem im letzten Kapitel auf die technischen Anforderungen von UAVs, die in einer Konstellation fliegen, eingegangen worden ist, gibt dieses Kapitel einen Überblick über aktuell verwendete Methoden und Algorithmen in der Flugbahnplanung und der Kooperation multipler UAVs. Zunächst jedoch einige Worte zur kooperativen Missionsplanung.

Kooperation zwischen UAVs wird vor allem dann nötig, wenn die zu erfüllende Mission, beispielsweise das Absuchen eines Areals, mit einem einzelnen UAV nicht mehr erfüllbar oder nicht effektiv ist, da UAVs im Allgemeinen verschiedene Einschränkungen wie eine begrenzte Flugdauer und Kommunikationsreichweite haben. Um eine solche Mission dennoch zu erfüllen, bzw. effizienter zu erfüllen, können mehrere UAVs eingesetzt werden, die die Mission und die einzelnen Aufgaben unter sich aufteilen [Sathyaraj et al., 2008]. Allerdings ist eine solche Aufteilung der Mission meist komplex, da die einzelnen Agenten untereinander kommunizieren und ihre einzelnen Bewegungen koordiniert werden müssen, speziell unter dem Gesichtspunkt der Kollisionsvermeidung und dem effizienten Einsatz der einzelnen Agenten. Die Komplexität ist dabei stark mit der Anzahl der verwendeten UAVs verbunden, was schnell deutlich wird, wenn man zwei UAVs, die sich auf Kollisionskurs befinden, betrachtet. Die beiden UAVs müssen einander ausweichen um eine Kollision zu vermeiden, doch damit kommen sie unter Umständen auf einen Kurs, der eine Kollision mit einem anderen UAV verursachen würde, und so weiter [Tsourdos et al., 2011]. Offensichtlich ist also Kommunikation und Koordination zwischen den einzelnen Agenten unabdingbar.

Tsourdos et al. [2011] versucht, die eben gezeigte Komplexität der Missionsplanung durch Aufteilen der Planung in kleinere Pakete zu reduzieren und stellt für die Missionsplanung ein abstrahiertes 3-Schichten-Modell vor (siehe Abb. 2.4), das aber je nach Einsatzzweck der UAVs

variieren kann. Auf der ersten Schicht residiert dabei der *Missionsplaner*, der je nach verwendeter

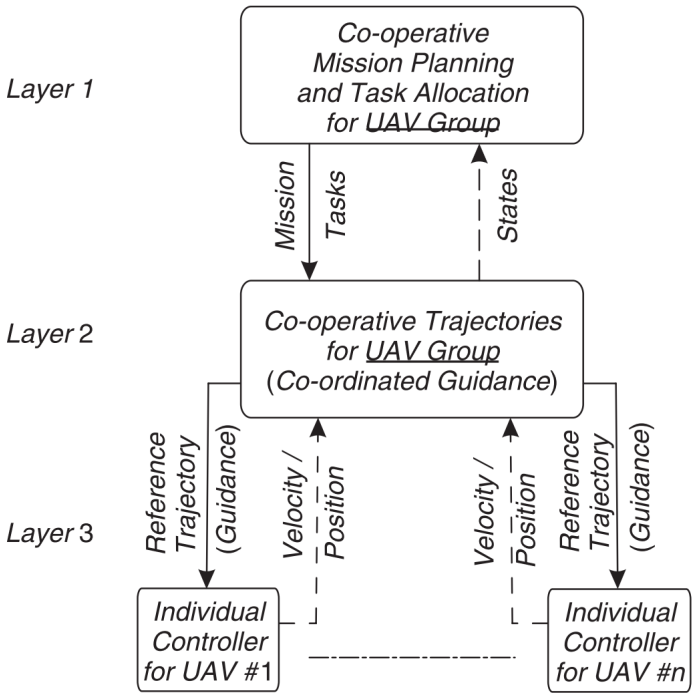

Abbildung 2.4.: Hierarchie einer Missionsplanung - (Tsourdos et al. [2011], Fig 1.4)

Konstellationsarchitektur zentral oder dezentral realisiert sein kann. Hier wird die globale Mission der Agenten in kleinere Portionen aufgebrochen und entsprechend den Agenten zugewiesen. Zusätzlich wird darauf geachtet, dass die globale Mission erfüllt wird.

In der zweiten, oder mittleren Schicht wird die Trajektorie oder die Flugbahn für alle Agenten errechnet, um die einzelnen zugewiesenen Teilaufgaben zu erfüllen, wobei hier auch auf Hindernis- bzw. Kollisionsvermeidung geachtet wird. Die dritte und letzte Schicht ist schließlich die Kontrolle bzw. Führung der individuellen Agenten auf ihrer berechneten Flugbahn.

Für diese Arbeit ist speziell die Schicht 1 und 2 von Interesse, weshalb im folgenden auf die Flugbahnplanung eingegangen wird.

2.4.1. Flugbahnplanung

Die Planung der Flugbahn lässt sich dabei für Quadrokopter in zwei, für sogenannte *fixed-wing*-UAVs in drei, Segmente teilen, da bei *fixed-wing* UAVs die physikalischen Randbedingungen eine größere Rolle spielen. Beispielsweise müssen solche UAVs, im Gegensatz zu Quadrokoptern, eine gewisse Mindestgeschwindigkeit halten, um nicht abzustürzen. Dadurch muss bei solchen

UAVs, zusätzlich zur Routenplanung, auch eine von den physikalischen Eigenschaften geleitete Trajektorie, in den meisten Fällen sind das gekrümmte Flugbahnen, geplant werden, was mithilfe von Algorithmen wie *Dubins Path*, *Pythagorean-hodographs* (PH) und *2D Clothoids* [Tsourdos et al., 2011] gelöst werden kann, auf welche in dieser Arbeit jedoch nicht näher eingegangen wird.

Nach Tsourdos et al. [2011] wird für UAVs wie Quadrokopter zunächst eine simplifizierte, mathematische Beschreibung der Umgebung erstellt, die alle möglichen Routen enthält, die Start- und Zielpunkt miteinander verbinden, wobei hier auch statische Hindernisse beachtet werden. Danach wird, mithilfe entsprechender Algorithmen, die in Kap. 2.4.1.2 näher erörtert werden, und abhängig von der Missions- und Zielvorgabe, eine geeignete Flugbahn gewählt.

Eine solche Flugbahnplanung ist naturgemäß sehr stark von Faktoren wie den physikalischen Eigenschaften der verwendeten UAVs, der Missionsvorgabe, der Umgebung, beispielsweise ob Hindernisse bekannt oder unbekannt sind, etc. abhängig. Einige Methoden zum Finden von solchen Flugbahnen sollen nun näher beleuchtet werden.

2.4.1.1. Methoden für die Routensuche

Drei der vorherrschenden Methoden zum Finden der möglichen Flugbahnen sind nach Tsourdos et al. [2011] die sog. *Road Map*, die *Cell Decomposition* und die *Potential Field* Methode, wobei diese drei mittlerweile in noch mehr Unterarten und Abwandlungen gegliedert sind [Valavanis, 2010].

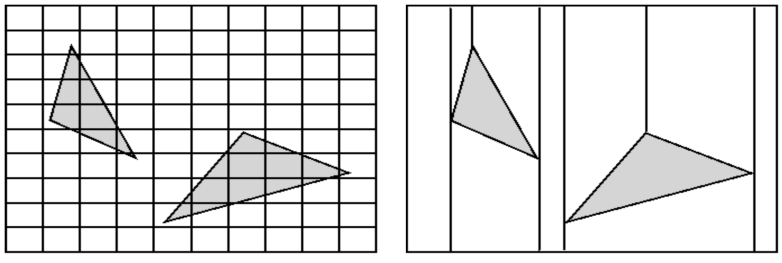

Abbildung 2.5.: a) genäherte Cell Decomposition - eine Aufteilung des gesamten Gebiets in gleich große Teile (Hindernisse sind dunkelgrau eingefärbt) **b)** Exakte Cell Decomposition - Hindernisse sind exakt eingegrenzt - (Valente et al. [2011], Fig 14)

- **Road Map Methode**

  Bekannten Methoden, wie *Voronoi Diagramme*, der *Visibility Graph* und teils auch die *Probabilistic Random Roadmap (PRR)* basieren alle auf einem ähnlichem Prinzip, der sogenannten Road Map Methode, wobei die abzusuchende Fläche hierbei bekannt sein muss, es

sich also um eine sog. *informierte* Suche handelt. Dabei wird versucht, die Umgebung, bzw. den zur Verfügung stehenden Raum, unter Beachtung eventueller Hindernisse als Graph darzustellen. Die einzelnen Kanten des Graphen repräsentieren in diesem Fall potentielle Pfade. Dadurch kann das Problem der optimalen Flugbahn unter Verwendung von Suchalgorithmen (näher erläutert in Kap. 2.4.1.2) gelöst, und ein entsprechender Pfad vom Start- zum Zielpunkt gefunden werden [Valavanis, 2010].

- **Cell Decomposition Methode**

  Analog zur Road-Map Methode gibt es auch hier verschiedene Ableger und Modifikationen der Methode, wobei die Grundlage immer das Aufteilen der Umgebung in disjunkte, also sich nicht überschneidende *Zellen*, bildet, auch hier jedoch ist die Voraussetzung, dass das Suchgebiet von vornherein bekannt ist. Adjazente Zellen, die frei sind, also solche, die sich nicht mit Hindernissen überschneiden, werden dann durch Kanten miteinander verbunden, was einen Graphen ergibt [Lingelbach, 2004]. Dieser kann wiederum mithilfe von Suchalgorithmen (siehe Kap. 2.4.1.2) nach dem optimalen Flugpfad abgesucht werden. Zwei Varianten der *Cell Decomposition* Methode sind die *exakte* und die *genäherte Cell Decomposition* (siehe Abb. 2.5), wobei die genäherte Methode speichereffizienter, aber auch ungenauer ist [Tsourdos et al., 2011].

- **Potential Field Methode**

  Die Potentialfeldmethode ist im Gegensatz zu den anderen zwei vorgestellten Methoden nicht diskret, sondern kontinuierlich [Tsourdos et al., 2011]. Hier wird die Umgebung als ein künstliches Potentialfeld dargestellt, wobei der Zielpunkt ein globales Minimum repräsentiert. Der Zielpunkt ist damit ein anziehendes Feld, und Hindernisse, die in diesem künstlichen Feld ein höheres Potential haben, sind abstoßende Felder. Damit wird versucht, den Agenten zum Zielpunkt zu lotsen [Lingelbach, 2004]. Die ursprüngliche Methode birgt jedoch einige Schwierigkeiten, da der Agent beispielsweise in lokalen Minima gefangen werden kann. Mithilfe von Verbesserungen und Erweiterungen der Methode, beispielsweise dem *randomized path planner* (RPP), können diese Schwierigkeiten jedoch überwunden werden [Tsourdos et al., 2011; Lingelbach, 2004].

2.4.1.2. Suchalgorithmen für die Routenwahl

Nachdem nun mögliche Routen vom Start- zum Zielpunkt mit einer der im vorigen Kapitel beschriebenen Methode erfasst wurden, muss ein optimaler Pfad gefunden werden, der den Ziel- mit dem Startpunkt verbindet. In dieser Arbeit wird davon ausgegangen, dass der kürzeste Pfad der optimale Pfad ist. Je nach Missions- und Aufgabenstellung kann dies jedoch variieren. Die möglichen Routen werden dabei als Graph $G = \{V, E\}$, mit V als Knoten und E als Kanten, bezeichnet.

Um den optimalen Pfad zu finden, gibt es mehrere algorithmische Ansätze, von denen im Folgenden einige erläutert werden.

- **Bellman Ford Algorithmus**

  Der Bellman-Ford-Algorithmus, oft auch als *Distanzvektoralgorithmus* referenziert, der beispielsweise auch für das Erstellen von Routing-Tabellen in Netzwerken genutzt wird, ist ein iterativer Prozess zur Suche nach dem kürzesten Weg ausgehend von einem Startknoten. Die Kanten *E* des Graphen sind dabei gewichtet, und der Algorithmus führt seine Suche nach dem Weg des geringsten Gewichts solange fort, bis er die Distanz zu jedem Knoten berechnet hat, wobei die Gewichtung der Kanten auch negativ sein kann, d.h. kostengünstig ist. [Sathyaraj et al., 2008]

- **Dijkstra's Algorithmus**

  Der Dijkstra-Algorithmus gehört zu den sog. *greedy* (engl. greedy - gierig) Algorithmen, das heißt der Algorithmus wählt immer die Möglichkeit bei der er seinen Gewinn zum Zeitpunkt der Wahl maximieren, bzw. in diesem Fall die Kosten minimieren kann. Dabei arbeitet Dijkstra, wie der Bellman-Ford Algorithmus, ebenfalls mit gewichteten Kanten in einem Graphen. Im Unterschied zu Bellman-Ford wählt Dijkstra jedoch nur dann den optimalen Weg, solange die Gewichtung nicht negativ ist [Weicker, 2013; Sathyaraj et al., 2008].

- **Floyd-Warshall Algorithmus**

  Floyd-Warshall, der ebenfalls ein Suchalgorithmus aus der Graphentheorie ist, berechnet die kürzesten Pfade zwischen allen Paaren der Knoten *V* eines Graphen. Der Algorithmus basiert dabei auf dem Prinzip der dynamischen Programmierung, d.h. der Algorithmus betrachtet Teilprobleme des Gesamtproblems, und speichert die Lösungen der Teilprobleme,

anhand derer er andere Teillösungen kombinieren kann [Weicker, 2013]. Dadurch verbessert sich die Gesamtlaufzeit, es wird jedoch mehr Speicher benötigt [Sathyaraj et al., 2008].

- **A* Algorithmus**

  A* (gesprochen „A-Stern“) ist ein mit dem Dijkstra verwandter Algorithmus, nutzt jedoch für die Wahl des nächsten Knotens, von dem die Berechnung weiterläuft, eine Heuristik, also eine Schätzfunktion, um die Laufzeit zu verringern. Der A*-Algorithmus ist dabei optimal, dass heißt es wird immer eine optimale Lösung gefunden, sofern sie existiert. Nach Sathyaraj et al. [2008] besitzt der A*-Algorithmus die kürzeste Laufzeit von den hier vorgestellten Algorithmen. Es gibt jedoch auch viele Weiterentwicklungen von A*, beispielsweise den D* (*dynamischer* A*)-Algorithmus, der in der Lage ist, Änderungen in der Umgebung zu berücksichtigen, und diese Änderungen effektiv in die weitere Pfadplanung mit einfließen lässt, indem er nur einen Teil des Pfades neu berechnet [Stentz, 1997].

2.4.2. Kooperative Suche

Ein Einsatzgebiet, bei dem eine wie in den einleitenden Worten des Kapitels beschriebene Kooperation mehrerer UAVs sehr oft angewandt wird, ist das Absuchen oder die Überwachung eines Gebiets, wie etwa bei Waldbränden, um Einsatzkräfte mithilfe genauer Lageinformation zu unterstützen [Casbeer et al., 2005]. Aber auch bei sog. *Search-and-Rescue*-Missionen, bei denen eine raum-zeitlich effiziente Suche erforderlich ist, bietet sich die Nutzung mehrerer kooperierender UAVs an, da das abzusuchende Gebiet meist sehr groß ist und die Sensorik der UAVs oft eingeschränkt ist, bei optischen Kameras beispielsweise durch die Auflösung und den Öffnungswinkel. Um nun eine parallelisierte Suche mit mehreren UAVs durchführen zu können, muss die Flugbahn der einzelnen Agenten entsprechend geplant und angepasst werden, weshalb die gesamte abzusuchende Fläche unter den Agenten aufgeteilt wird. Bewährt hat sich für so eine Aufteilung eine Methode wie die *Cell Decomposition* (vgl. 2.4.1) [Valente et al., 2011].

Bei der bereits erwähnten *genäherten* Cell Decomposition wird dabei die gesamte abzusuchende Fläche in vieleckige *Zellen*, sogenannte Polygone, unterteilt. Traditionell sind diese Polygone Vierecke (vgl. Abb. 2.5), wobei aktuelle Forschungsergebnisse bei einer hexagonalen Form der Polygone eine bessere Koordination der UAVs in Aussicht stellen [Gao und Zhao, 2014]. Diese Unterteilung des gesamten Suchraums in gleichartige Zellen hat den Vorteil, dass ein anfangs

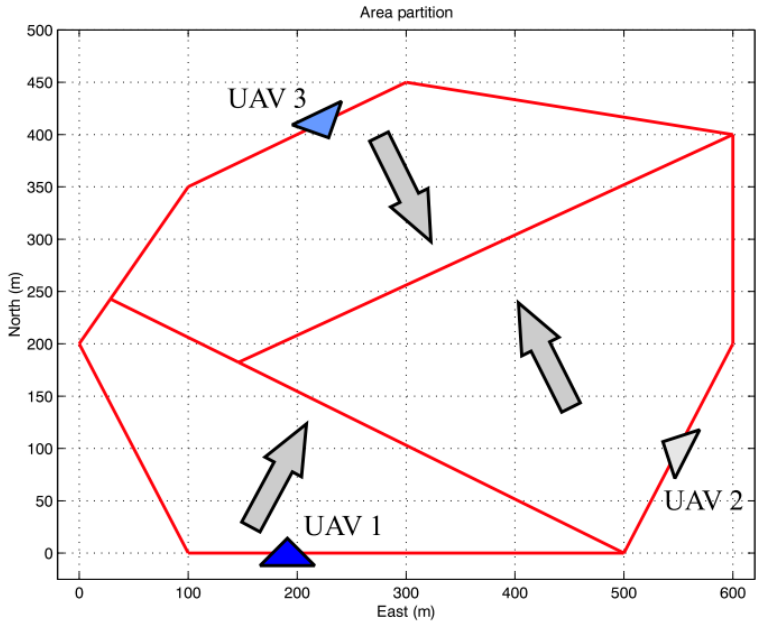

Abbildung 2.6.: Aufteilung eines Gebiets in drei Teilgebiete für die einzelnen UAVs. Die grau gefüllten Pfeile geben dabei die Richtung an, in die die Suche erfolgen soll. - (Maza und Ollero [2007], Fig 5)

komplexes Gebiet in handhabbare Teilstücke unterteilt wird, die dann als Graph oder Spannbaum repräsentiert werden können und aus diesen dann mithilfe von den in Kapitel 2.4.1.2 vorgestellten Algorithmen abzufliegende optimale Pfade erstellt werden [Jones, 2009].

Jedoch kann diese Methode bei Flächen, die keine trivialen Polygone darstellen (siehe Abb. 2.7b), nur bedingt angewandt werden, falls man ein komplettes Absuchen benötigt, ohne die Begrenzungslinien zu überschreiten. Dafür eignet sich die sogenannte *Area Decomposition*, die ähnlich der beschriebenen *exakten* Cell Decomposition das gesamte Gebiet auf die UAVs verteilt. Der Unterschied besteht jedoch darin, dass bei der Area Decomposition keine Zellen erstellt werden, die noch auf die verschiedenen Agenten mithilfe von Routenplanung verteilt werden müssen, sondern das gesamte Gebiet flächenmäßig auf die Agenten verteilt wird (siehe Abb. 2.6). Diese Flächen werden dann von den einzelnen Agenten nach Möglichkeit optimal, das heißt komplett und mit einem minimalen Zeit-/Wegaufwand abgesucht [Maza und Ollero, 2007]. Der Vorteil bei dieser Methode besteht darin, dass das Gesamtproblem von miteinander kooperierenden UAVs nach dem Teile-und-Herrsche-Prinzip als jeweils ein kleineres Teilproblem mit einem einzelnen UAV betrachtet werden kann. Damit lassen sich auch heterogene Teams von Agenten, beispielsweise unterschiedlich große UAVs mit verschiedenen Sensoren, sehr einfach einsetzen, da die Aufteilung der gesamten Fläche an die jeweiligen Fähigkeiten der UAVs, beispielsweise höhere Geschwindigkeit und damit ein schnelleres Absuchen, angepasst werden kann [Valente et al.,

2011]. Allerdings hat diese Methode den Nachteil, dass das Suchgebiet eine konvexe Form ha-

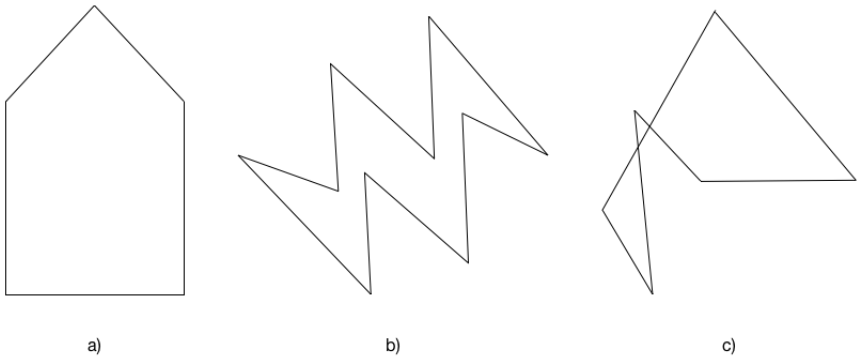

Abbildung 2.7.: Verschiedene Arten eines Polygons **a)** konvexes Polygon, **b)** kokaves Polygon **c)** komplexes Polygon

ben muss, das heißt keine Einbuchtungen hat, und dass bei Formen eine effektive Aufteilung der Suchfläche meist nicht mehr trivial ist [Maza und Ollero, 2007]. Dieses Problem der Aufteilung eines nicht-konvexen Polygons in Subpolygone ist ein Teilbereich der Informatik und wird als *Algorithmische Geometrie* bezeichnet, an welcher bereits seit Jahrzenten geforscht wird [Sack und Urrutia, 2000]. Deshalb existieren viele Techniken zur sogenannten *Polygon Decomposition*, der Aufspaltung einer vieleckigen Fläche in einfache Formen, zum Beispiel der Keil-Snoeyink-, Hertel-Mehlhorn- oder Seidel's Algorithmus [Keil und Snoeyink, 2002; Stancheva, 2003], die alle eine möglichst optimale Aufteilung der gegebenen Suchfläche realisieren. Diese Algorithmen werden jedoch im Rahmen dieser Arbeit nicht näher beleuchtet.

Um nun eine solche zugewiesene Fläche abzusuchen, existieren verschiedene Methoden, die jedoch stark an die Randbedingungen der Missionsanforderungen gekoppelt sind, beispielsweise ob das abzusuchende Gebiet und eventuelle Hindernisse bekannt oder unbekannt sind. Eine sehr weit verbreitete Methode zum Absuchen ist dabei nach Moon und Shim [2009] die sogenannte *Boustrophedon*-Methode, bei der der Agent so lange Vor- und Zurück Bewegungen durchführt, bis die gesamte Fläche abgesucht ist, wobei bei jedem *turn* - also bei jeder Richtungsänderung des Agenten - das nächste Flugsegment etwas nach rechts oder links versetzt wird, sodass sich das abgesuchte Gebiet nicht überlappt [Choset und Pignon, 1997].

Nach Maza und Ollero [2007] ist bei einer Suche nach der *Boustrophedon*-Methode, die Anzahl der *turns*, die der Agent zu absolvieren hat, der größte Faktor für die Gesamtzeit, die ein Agent braucht um ein Gebiet abzusuchen. Die Gesamtzeit der Suche ergibt sich dabei aus der Zeit für die einzelnen Vor- bzw. Zurückbewegungen und der Zeit, die ein Agent braucht um die Richtung zu

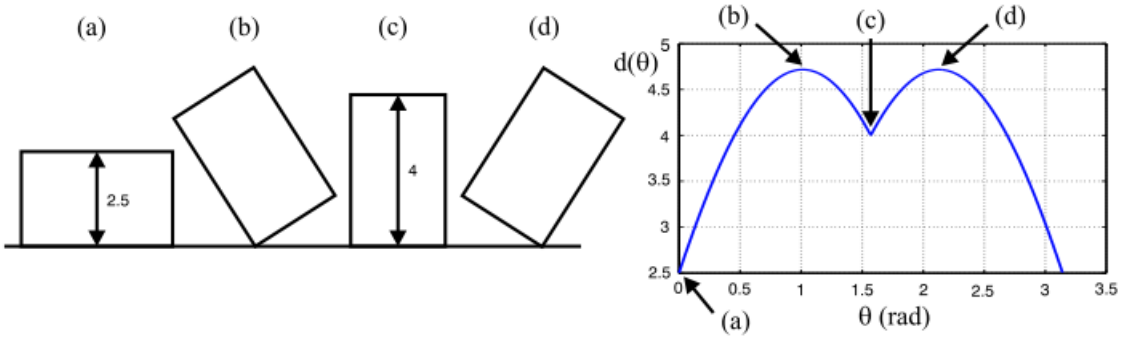

Abbildung 2.8.: Darstellung der Durchmesserfunktion für ein einfaches Polygon zur Bestimmung der Suchrichtung, bei der möglichst wenige Richtungswechsel nötig sind - (Maza und Ollero [2007], Fig 4)

ändern. Nach Huang [2001] ist dabei eine Vor- und Zurückbewegung senkrecht zur Suchrichtung des Agenten am effektivsten.

Um die gesamte Suchzeit möglichst klein zu halten, muss also Anzahl der *turns* minimiert werden, was dadurch erreicht wird, dass man die Richtung in die die Suche erfolgen soll an die kleinstmögliche, größte Ausdehnung des zugewiesenen Suchfeldes anpasst. Diese kleinstmögliche, größte Ausdehnung lässt sich mithilfe der Durchmesserfunktion berechnen. Anschaulich erklärt wird dabei das Polygon auf einer ebenen Fläche um 360° gedreht und die Höhe, das heißt die Ausdehnung des Polygons senkrecht zur ebenen Fläche während des Drehens betrachtet, wobei sich für die Höhe ein lokales Minimum zwischen zwei Maxima, also einer größtmöglichen Ausdehnung des Polygons, ausbildet. Abb. 2.8 verdeutlicht dieses Vorgehen. Dieses lokale Mi-

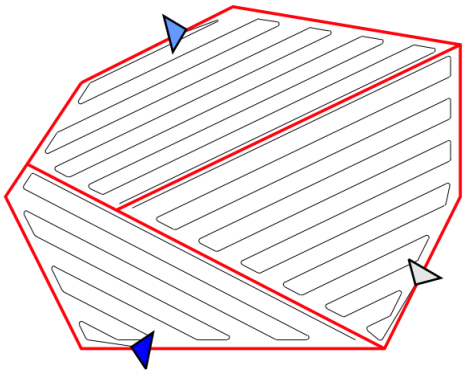

Abbildung 2.9.: Absuchen einer Fläche nach der Boustrophedon-Methode, wobei die Richtung der Suche mittels der Durchmesserfunktion bestimmt wurde. - (Maza und Ollero [2007], Fig. 6)

nimum entspricht der kleinstmöglichen, größten Ausdehnung des Polygons und bestimmt damit die Suchrichtung des Agenten, wobei die Anzahl der *turns* minimiert ist. Wendet man dies auf Abb. 2.6 an, ergibt sich für die Suche der einzelnen UAVs Abb. 2.9.

Dieser Ansatz des Absuchens durch die *Boustrophedon*-Methode lässt sich nach Huang [2001] auch für Gebiete mit Hindernissen anwenden, wobei hier die Fläche so aufgeteilt werden muss, dass sie aus konvexen Teilflächen besteht, was in den meisten Fällen relativ kompliziert ist. Im Allgemeinen kann dieser Ansatz des Absuchens jedoch bei fast allen Anwendungen, die eine komplette Erfassung eines Gebiets erfordern, angewandt werden.

3. Konzept

Das letzte Kapitel hat einen Überblick über aktuelle Techniken und Lösungen vorgestellt, die zur Kooperation mehrerer UAVs verwendet werden können. In diesem Kapitel geht es nun darum, die in der Einleitung erläuterte Problemstellung der kooperativen Suche mithilfe der genannten Techniken und Methoden zu erfassen und konzeptuell zu lösen, wobei das Problem zunächst auf zwei kooperierende Quadrokopter reduziert wird, die im ersten Teil der Lösung vorgegebene Wegpunkte und im zweiten Teil der Lösung eine vorgegebene, rechteckige Fläche absuchen.

3.1. Überblick

Abbildung 3.1.: Aufteilung des Problems in drei Teilbereiche

Um das spezifizierte Problem anzugehen, wird dieses zunächst in drei kleinere Teilprobleme unterteilt, in die Positionserkennung via optischem Tracking System, die Kommunikationslösung und schlussendlich die kooperative Suche, respektive die Aufteilung der Suche auf die einzelnen Quadrokopter.

Für die Positionserkennung wird dabei im Rahmen der Arbeit das bereits vorhandene optische Trackingsystem genutzt und in das bestehende System zur Positionserkennung integriert. Die Kommunikation und Kooperation der Quadrokopter, die für die spätere Suche erforderlich ist, basiert dabei auf einem Kommunikationsframework, das ebenfalls auf das bereits vorhandene System aufsetzt und dieses um notwendige Eigenschaften und Befehle erweitert. (siehe Kap. 3.3). Zuletzt verteilen Algorithmen die abzusuchenden Wegpunkte oder das abzusuchende Gebiet so

auf die Quadrokopter, dass die gesamte Fläche bzw. alle Wegpunkte effektiv abgesucht werden, und dabei keine Kollisionen entstehen.

3.2. Positionserkennung via Optischem Tracking System

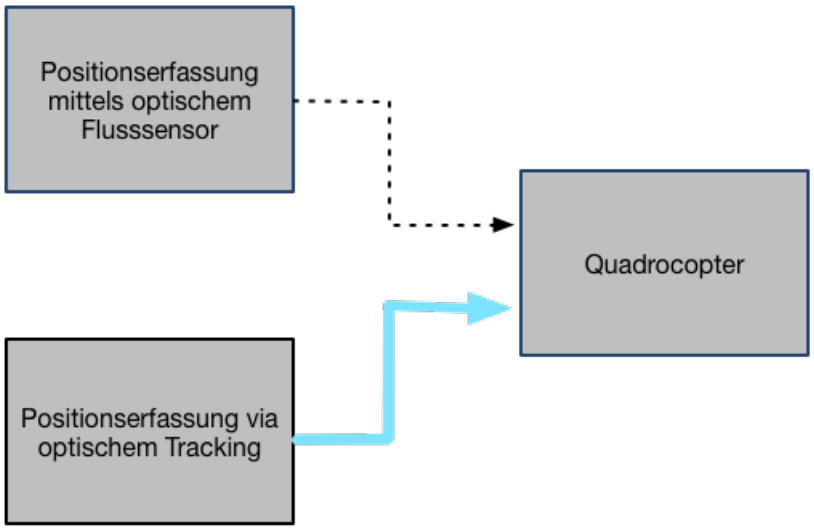

Abbildung 3.2.: Nutzung der optischen Trackingdaten anstelle des optischen Flusssensors - grau hinterlegtes ist dabei bereits vorhanden, blau symbolisiert zu Entwickelndes

Das aktuell genutzte System besitzt bereits eine Möglichkeit zur relativen Positionserfassung des Quadrokopters, basierend auf einem optischen Flusssensor, wobei das optische Tracking als Referenz verwendet wurde, um die Genauigkeit der Steuerung mittels des optischen Flusses zu verifizieren und zu überprüfen. Nun sollen die Positionsdaten, die das optische Tracking-System liefert, anstelle der Daten des optischen Flusssensors genutzt werden, es müssen also die Daten des OTS entsprechend empfangen und an den Quadrokopter weitergesendet werden, wie Abb. 3.2 verdeutlicht.

3.3. Kommunikationslösung

Wie bereits in vorangegangenen Kapiteln erwähnt, ist Kommunikation zwischen den einzelnen Quadrokoptern entscheidend für die spätere Kooperation. Deshalb muss zunächst eine geeignete Konstellationsarchitektur und dann geeignete Protokolle und Übertragungsmedien gewählt

werden, die zum einen eine sichere Übertragung gewährleisten, zum anderen aber auch eine entsprechende Reichweite haben. Das kooperative Absuchen entspricht dabei der in Kapitel 2.3.1

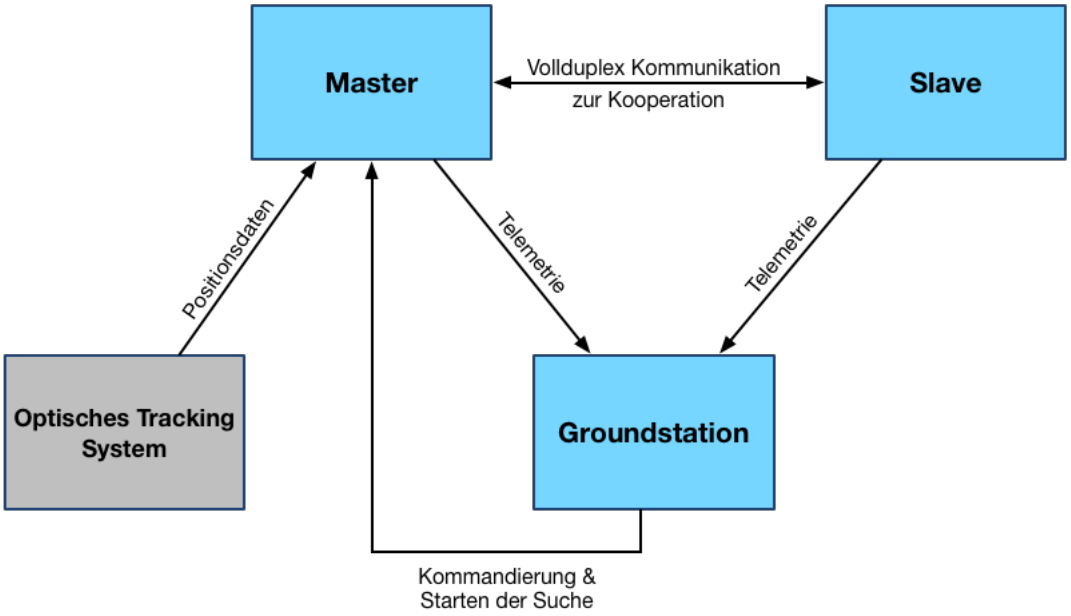

Abbildung 3.3.: Kontroll- und Kommunikationsarchitektur des entwickelten Systems. Die Pfeile symbolisieren dabei die Kommunikation, die ausschließlich über WLAN stattfindet.

beschriebenen *Intentionalen Kopplung* der Quadrokopter, da die Quadrokopter ein gemeinsames Ziel verfolgen, jedoch nicht miteinander physisch in Verbindung stehen, oder eine Formation aufrechterhalten müssen. Da das Problem zunächst auch auf zwei Quadrokopter reduziert ist, wird eine zentralisierte Kontrollarchitektur der dezentralisierten vorgezogen, da dies die Aufgabenverteilung auf die einzelnen Quadrokopter und die Sicherstellung, dass das Ziel erreicht wird, vereinfacht. Dazu wird eine modifizierte *Leader-Follower*-Architektur verwendet. Im Gegensatz zur Definition in Kapitel 2.3.1 folgt der *Follower* jedoch nicht jeder Bewegung des *Leaders*, sondern wird vom *Leader* kommandiert. Um diesbezüglich Verwechslungen zu vermeiden, wird im Folgenden der Quadrokopter, der Befehle aussendet als *MASTER* bezeichnet, und der Quadrokopter, der die Befehle des *MASTERs* empfängt und ausführt, als *SLAVE*.

Um die Befehle zu versenden, bzw. zu empfangen, ist eine Kommunikationslösung notwendig. Hierbei wird auf herkömmliche kommerzielle Lösungen zurückgegriffen, die bereits erprobt

sind, um Kosten und Aufwand niedrig zu halten. Zur Verfügung stehen dabei die in Kap. 2.3.3 beschriebenen Möglichkeiten Bluetooth und WLAN.

Der Vorteil, den Bluetooth bietet, ist die aus Sicht der Implementierung sehr einfache Kommunikation. Allerdings sind die in Kap. 2.3.3.1 genannten Spezifikationen (maximal 7 Geräte - Aufteilung der Bandbreite), vor allem für eventuelle spätere Einsatzzwecke, bei denen mehr Quadrokopter eingesetzt werden sollen, nicht gut geeignet. Auch spielt in einem solchen Szenario die Reichweite der Funkübertragung, die bei Bluetooth viel stärker begrenzt ist als bei WLAN (siehe Kap. 2.3.3.2), eine Rolle, da die Kommunikation, wie bereits erwähnt, essentiell für eine erfolgreiche Kooperation ist. Um also auch für größer ausgelegte Missionen tauglich zu sein, sollte die verwendete Kommunikationsstruktur die gerade genannten Nachteile vermeiden. Eine Kommunikationslösung mit WLAN überwindet die Limitierungen von Bluetooth, jedoch auf Kosten des benötigten Stroms. Sie ist aber für den gewollten Einsatzzweck besser geeignet, da zum einen eine höhere Reichweite möglich ist und zum anderen mehr Geräte miteinander vernetzt werden können.

Darauf aufbauend werden eigene Befehle und Kommunikationsframes definiert, mit deren Hilfe die Quadrokopter untereinander kommunizieren und Telemetrie an eine Groundstation senden können. Abb. 3.3 veranschaulicht dabei das Konzept der Kommunikationsstruktur, wobei die Groundstation nur als Kommandointerface zum Senden der abzusuchenden Punkte, bzw. der Fläche und Telemetrieanzeige dient.

3.4. Kooperative Suche

In den folgenden zwei Unterkapiteln werden die zwei in der Einleitung angesprochenen Probleme, das kooperative Absuchen von Wegpunkten und das Absuchen einer Fläche, näher betrachtet.

3.4.1. Wegpunktsuche

Nach der in 2.4.1.2 gegebenen Defintion eines optimalen Pfads - optimal bedeutet möglich kurz - muss also, um das Problem der kooperativen Wegpunktsuche zu lösen, für jeden Quadrokopter ein optimaler Pfad gefunden werden, wobei dieser sich nur bedingt überschneiden darf, da Kollisionen vermieden werden müssen.

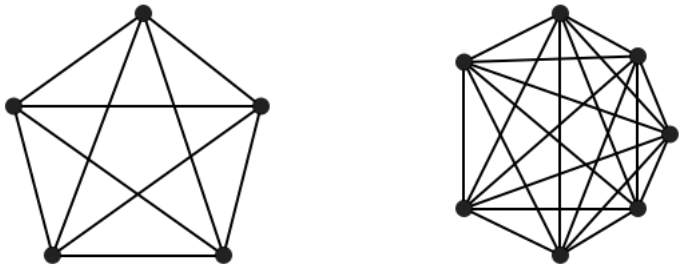

Abbildung 3.4.: Vollständiger Graph, **a)** 5 Knoten, 10 Kanten **b)** 7 Knoten, 21 Kanten

Ein Ansatz ist, diese Wegpunkte in einem vollständigen kantengewichteten Graphen G als Knotenmenge V darzustellen. Das Gewicht einer Kante $w_{1,2}$ zwischen zwei Knoten V_1 und V_2 entspricht dabei der Länge des Pfades zwischen Wegpunkt 1 und Wegpunkt 2. Um nun für jeden Quadrokopter den optimalen Pfad zu finden, müssten alle möglichen Pfade beider Quadrokopter berechnet, miteinander verglichen, auf Kollisionsvermeidung geachtet und sichergestellt werden, dass kein Wegpunkt doppelt angeflogen wird. Auch muss darauf geachtet werden, dass unter Umständen ein Pfad der beispielsweise nur 2 Wegpunkte enthält genauso lang sein kann wie ein Pfad der die restlichen Wegpunkte enthält. Dadurch müssen alle möglichen Permutationen der Wegpunkte beachtet werden, da die Pfadlänge durch die Wahl der Anordnung der Wegpunkte variiert werden kann.

Bei zwei Quadrokoptern müssen also zwei Subgraphen $P_1(K_1, L_1), P_2(K_2, L_2)$ von $G(V, E)$ mit $L_1 = \{l_{1,1} \dots l_{1,n}\}$ und $L_1 = \{l_{2,1} \dots l_{2,m}\}$ gefunden werden, sodass

$$K_1 \cup K_2 = V \quad \wedge \quad K_1 \cap K_2 = \emptyset \tag{3.1}$$

wobei K_1 und K_2 so gewählt werden sollen, dass

$$w_{ges} = w_1 + w_2 = \sum_{i=0}^{n-1} w_{i,i+1} \quad + \quad \sum_{j=0}^{m-1} w_{j,j+1} \tag{3.2}$$

minimal wird, mit

$$w_1 = w_2 + \varepsilon \quad \text{mit} \quad |\varepsilon| \geq 0 \tag{3.3}$$

der Gesamtpfad also möglichst kurz, und beide Pfade der Quadrokopter möglichst gleich lang sind. Betrachtet man Abb. 3.4 stellt man jedoch schnell fest, dass die Berechnung aller mögliche

Pfade sehr lange dauern kann, weil die Anzahl der verfügbaren Kanten viel stärker als die Anzahl verwendeter Knoten ansteigt. Dabei beträgt die Anzahl der möglichen Permutationen von n Wegpunkten für einen Quadrokopter

$$P_n = \sum_{k=0}^{n} \frac{n!}{(n-k)!}$$

Dieses Problem der Suche nach dem kürzesten Pfad ist dabei ähnlich dem in der Literatur als *Travelling-Salesman* bezeichnetem Problem, dem Problem des Handlungsreisenden, bei dem dieser verschiedene Städte genau einmal besuchen muss, und dafür die optimale, das heißt die kürzeste Gesamtroute nehmen soll.

In Kapitel 2.4.1.2 wurden für das Finden eines kürzesten Pfades bereits einige Suchalgorithmen vorgestellt. In unserem Fall ist das Problem jedoch noch etwas komplizierter als das *Travelling-Salesman* Problem, da man zwei Quadrokopter hat und für *beide* den jeweils kürzesten Pfad finden muss, wobei kein Wegpunkt mehr als einmal angeflogen werden darf.

Die Berechnung eines optimalen Pfades durch eine reine *Brute-Force-Methode*, also durch das einfache Berechnen und Vergleichen aller möglichen Pfade, scheidet damit für die gewünschte Anwendung aus, da die Aufteilung der Wegpunkte erst auf dem Quadrokopter vorgenommen werden soll, dessen Leistung jedoch begrenzt ist. Bereits beim einfachen *Travelling Salesman*-Problem beträgt die Laufzeit einer solchen Berechnung aller möglicher Pfadlängen $\mathcal{O}(n!)$. Muss man nun zwei solche kürzeste Pfade finden, ist die Laufzeit noch um einiges höher. Dadurch wäre das gewollte möglichst schnelle Absuchen von Wegpunkten nicht gewährleistet, da der Quadrokopter bei vielen Wegpunkten eine sehr lange Zeit für die Pfadberechnung brauchen würde.Deshalb wird zur Problemlösung ein eigener Algorithmus konzipiert, der auf einem Dijkstra-ähnlichen *Greedy*-Algorithmus beruht und mittels eines Sortierverfahrens kürzeste Pfade ermittelt.

Zunächst werden dabei die Anzahl der Wegpunkte s auf die Anzahl der verfügbaren Quadrokopter q aufgeteilt, so dass jeder Quadrokopter etwa gleich viele Wegpunkte zugewiesen bekommt.

Es müssen also q einfach-gerichtete Graphen $P_1(K_1, L_1) \dots P_q(K_q, L_q)$ gefunden werden, wobei für die Anzahl n der Knoten in der Knotenmenge K gilt

$$\begin{gathered} n_1 = \left\lceil \frac{s}{q} \right\rceil + \epsilon_1, \quad \dots \quad , n_q = \left\lfloor \frac{s}{q} \right\rfloor + \epsilon_q \quad \text{mit} \quad 0 \leq \epsilon_1 \dots \epsilon_q \leq 1 \\ \text{sodass} \quad \sum_{i=1}^{q} n_i = s \quad \text{und} \quad \epsilon_q < \epsilon_1 \quad \text{falls} \quad \epsilon_1 > 0 \end{gathered} \tag{3.4}$$

Nun werden die möglichen Permutationen $a_{k_1,1} \dots a_{k_q,q}$ der Knotenmengen $K_1 \dots K_q$ berechnet und gespeichert, wobei

$$k_1 =_{n_1} P_s, \quad \dots \quad , k_q =_{n_q} P_s \qquad \text{mit} \quad {}_nP_s = \frac{n!}{(n-s)!} \tag{3.5}$$

die jeweilige Anzahl der entsprechenden Permutationen der Knoten in der Knotenmenge K darstellt. Dadurch ergibt sich folgende Darstellung:

$$A_{k,q} = \begin{pmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,q} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,q} \\ \vdots & \vdots & \ddots & \vdots \\ a_{k,1} & a_{k,2} & \cdots & a_{k,q} \end{pmatrix} \quad \text{mit} \quad k =_{n_1} P_s \quad \text{und} \quad (a_{i,j})_{i=1,\dots,k}^{j=1,\dots,q} \tag{3.6}$$

Falls nun $k > k_j$ wird das Element $a_{k,j}$ mit 0 aufgfüllt. werden mit 0 aufgefüllt.

Die Knotenmenge jeder möglichen Permutation $a_{i,j}$ wird nun so sortiert, dass die Summe der Kantengewichte w_{ges} (Definition siehe 3.2) der Permutation möglichst klein ist. Dies wird mit einem *Dijkstra*-ähnlichem Verfahren bestimmt. Dabei wird, ausgehend vom Startknoten, in unserem Fall dem Startpunkt des Quadrokopters, der beste nächste Wegpunkt gewählt, wobei der beste nächste Wegpunkt derjenige ist, zu dem die Entfernung am geringsten ist.

Eine Permutation $a_{i,j}$ mit Knotenmenge $M = \{m_1 \dots m_{n_j}\}$ wird also so sortiert, dass

$$w_{1,2} \quad \leq \quad w_{2,3} \quad \leq \dots \leq \quad w_{n_j-1\ ,\ n_j} \tag{3.7}$$

Hat man so nun das möglichst kleinste Gesamtgewicht der Kanten einer Permutation $a_{i,j}$ bestimmt, werden die Spalten der Matrix $A_{k,q}$ nach diesem Gesamtgewicht aufsteigend sortiert, wobei Matrixelemente die $a_{i,j} = 0$ entsprechen, nicht sortiert werden, sondern an ihrem Platz

verbleiben. Das heißt es werden in der j-ten Spalte nur die ersten k_j-Elemente aufsteigend sortiert. Dabei entsteht eine neue Matrix

$$B_{k,q} = \begin{pmatrix} b_{1,1} & b_{1,2} & \cdots & b_{1,q} \\ b_{2,1} & b_{2,2} & \cdots & b_{2,q} \\ \vdots & \vdots & \ddots & \vdots \\ b_{k,1} & b_{k,2} & \cdots & b_{k,q} \end{pmatrix} \tag{3.8}$$

Mithilfe dieser Matrix lässt sich relativ einfach eine flugtaugliche Pfadkombination finden, indem man iterativ die einzelnen Spalten j der Zeilen i, beginnend bei $b_{1,1}$ solange durchgeht, bis eine Pfadkombination $\{b_{i,1}, b_{i,2}, \ldots, b_{i,q}\}$ gefunden ist, die der Definition in 3.1 genügt, so dass alle Wegpunkte genau einmal in der Pfadkombination vorkommen. Dabei ist diese Pfadkombination auch möglichst kurz, da die Elemente der Matrix $B_{k,q}$ nach der Pfadlänge sortiert sind. Allerdings wird dabei nicht unbedingt der optimalste Pfad gefunden, da nicht alle Pfade miteinander verglichen werden.

3.4.2. Flächensuche

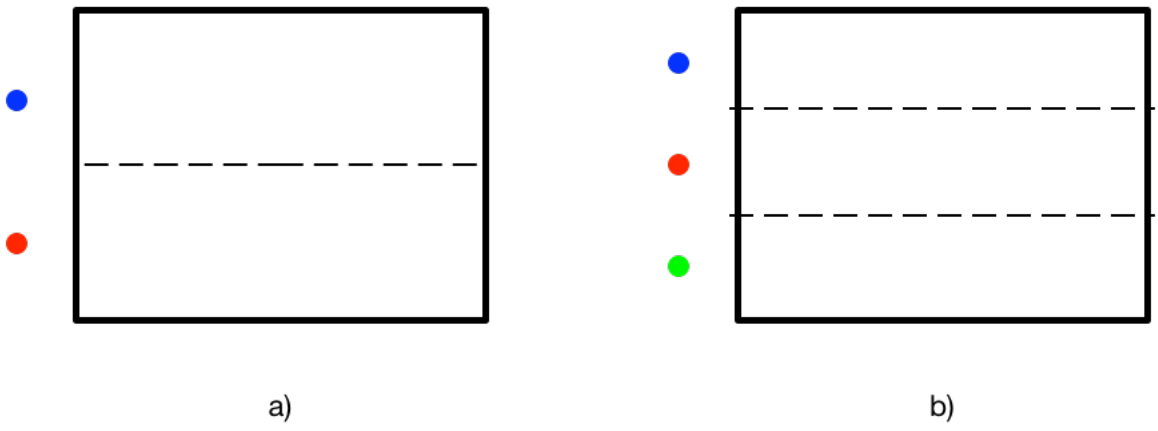

Abbildung 3.5.: Aufteilung der Fläche eines Rechtecks in **a)** zwei gleiche Flächen und **b)** drei gleiche Flächen; die farbigen Punkte stellen dabei die Quadrokopter dar

Nachdem nun ein Konzept für das Finden eines kürzesten Pfades zum Absuchen von Wegpunkten vorgestellt wurde, geht es in diesem Kapitel um das kooperative Absuchen einer Fläche.

Um das Suchgebiet, wie in der Problemstellung gefordert, effizient mit mehreren Quadrokoptern abzusuchen, müssen mehrere Aspekte betrachtet werden. Zunächst muss entschieden werden, wie das Suchgebiet aufgeteilt wird. In Kap. 2.4.2 wurden dazu zwei bewährte Methoden, die *Cell-* sowie die *Area Decomposition*, vorgestellt, wobei bei ersterer die gesamten Fläche in gleichmäßige Polygone unterteilt wird und bei zweiterer das Gesamtgebiet flächenmäßig aufgeteilt wird. Beide Techniken sind für das gegebene Problem zunächst denkbar, jedoch sind die in Kap. 2.4.2 genannten algorithmischen Ansätze, wie beispielsweise die Polygonzerlegung nach *Hertel-Mehlhorn* oder *Keil-Snoeyink*, relativ komplex, was die Frage aufwirft, ob ein solcher Aufwand für die Zerlegung überhaupt notwendig ist, da nach Problemstellung zunächst nur ein rechteckiges Suchgebiet betrachtet wird.

Bei einem Rechteck lässt sich die Komplexität der Zerlegung nämlich auf einen eindimensionalen Fall reduzieren, indem man die Seite des Rechtecks, bei der die zwei Quadrokopter starten, betrachtet. Teilt man die Länge dieser Linie durch zwei, und zieht von diesem Punkt aus eine Linie orthogonal durch das Rechteck, dann ist die Fläche des Rechtecks durch zwei geteilt, siehe auch Abb. 3.5a. Eine solche Aufteilung der Fläche lässt sich auch mit mehr als zwei Quadrokoptern realisieren, da nur die Länge der Linie durch die Anzahl der verfügbaren Quadrokopter geteilt werden muss, um die gesamte Fläche aufzuteilen, siehe Abb.3.5b.

Der zweite Aspekt, der bei der Flächensuche wichtig ist, ist das Absuchen der Fläche selber. Eine entsprechende Technik, die *Boustrophedon*-Methode, bei der die Quadrokopter durch Vor- und Zurück-Bewegungen orthogonal zur Suchrichtung und durch das Versetzen des Quadrokopters nach rechts bzw. links bei jeder Richtungsänderung das gesamte Suchgebiet absuchen, wurde ebenfalls bereits in Kapitel 2.4.2 vorgestellt. Dabei wird die Suchrichtung mittels einer Durchmesserfunktion (vgl. 2.8) bestimmt, um die Anzahl der Richtungsänderungen des Quadrokopters zu minimieren, um so zeiteffektiv zu arbeiten. Im Falle der vorliegenden rechteckigen Fläche

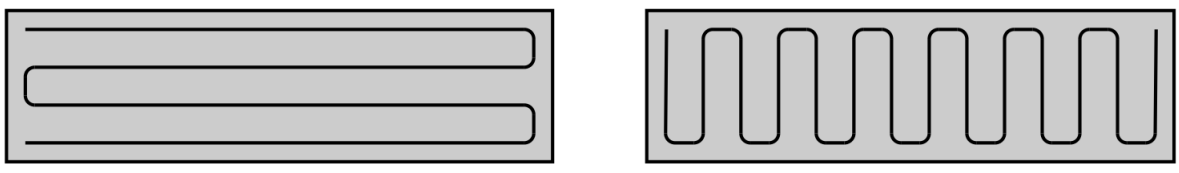

Abbildung 3.6.: Boustrophedon Methode bei Suchrichtung orthogonal zur längeren bzw. zur kürzeren Seite - (Maza und Ollero [2007], Fig.3)

lässt sich allerdings auch hier eine sehr viel einfachere Methode verwenden: Man finde die län-

gere Seite des Rechtecks. Legt man die Suchrichtung orthogonal zu dieser längeren Seite fest, wird dabei die Anzahl der Richtungsänderungen minimiert wird, was nach Kapitel 2.4.2 effizient ist, da Richtungsänderungen durch das Abbremsen und wieder beschleunigen des Quadrokopters Zeit kosten. Abb. 2.9 veranschaulicht dies sehr gut.

Eine Alternative zu dieser *Boustrophedon*-Methode wäre die Nutzung einer sogenannten Breiten- bzw. Tiefensuche, beides Suchansätze, die aus der Graphentheorie stammen, mit einigen Modifikationen allerdings auch zum Erkunden einer Fläche genutzt werden können [Barth, 2013]. Bei der Breitensuche wird dabei zunächst die Seite der Suchfläche betrachtet, es wird also Ebene für Ebene abgesucht, während die Tiefensuche den Suchbereich als erstes “in die Tiefe„ das heißt die lange Seite zuerst absucht. Wenn die Tiefensuche nun an eine Wand kommt, oder an einen Suchbereich der bereits besucht wurde, wird nach rechts abgebogen. Dies wird solange wiederholt, bis die gesamnte Fläche abgesucht ist.

Diese beiden Methoden sind bereits für einen einzelnen Quadrokopter implementiert, das heißt, es lassen sich bereits Wegpunkte so für ein vorgegebenes Rechteck generieren, dass eine Fläche abgedeckt wird. Mit ein paar Erweiterungen lassen sich diese beiden implementierten Suchmethoden auch sehr gut für das vorliegende Problem der Suche mit mehreren Quadrokoptern nutzen.

4. Implementierung

Dieses Kapitel beschreibt die Implementierung und Integration des in Kapitel 3 vorgestellten Konzepts auf dem bereits bestehenden Quadrokopter-System, das im Rahmen des AQopterI8-Projekts entwickelt wurde.

Es wird dabei zum einen ein kurzer Überblick über das Quadrokopter-System gegeben, zum anderen wird die Implementierung der Nutzung des optischen Tracking Systems, der Kommunikationslösung, sowie der Algorithmen zur Aufteilung der Wegpunkte, bzw. der Fläche gegeben. Schlussendlich wird die GUI, die graphische Benutzeroberfläche für die Steuerung der Quadrokopter näher beschrieben.

4.1. Überblick

Das aktuelle Quadrokopter-System besteht, abgesehen von der Hardware, die für den Flug notwendig ist, aus zwei separaten Software-Teilen. Zunächst gibt es die Quadrokopter-Flugsoftware, die auf einem Mikrocontroller ausgeführt wird und die dabei für die Regelung des Quadrokopters zuständig ist. Der zweite Software-Teil, der auf einem separaten, kleinen eingebetteten Windows-Rechner läuft, ist dabei hauptsächlich für komplexere Aufgaben zuständig, die unter Anderem eine höhere Rechenleistung benötigen, als sie der Mikrocontroller bereitstellen kann, so wie beispielsweise beim 3D-Mapping. Dieser Teil basiert auf Qt/C++ und ist aktuell über eine GUI bedienbar, wobei eine *Remotedesktopverbindung* zum Quadrokopter aufgebaut werden muss.

Für die Implementierung des Konzepts werden möglichst keine Modifikationen an der Flugsoftware auf dem Mikrocontroller und der Hardware vorgenommen, da der Fokus auf dem Software-Teil liegt, der auf dem Windows-Rechner läuft.

Im Rahmen dieser Arbeit wird dabei eine neue Klasse `OTS_Control` erstellt, die alle nötigen Funktionen für die Implementierung des Konzeptes enthält.

4.2. Optisches Tracking System

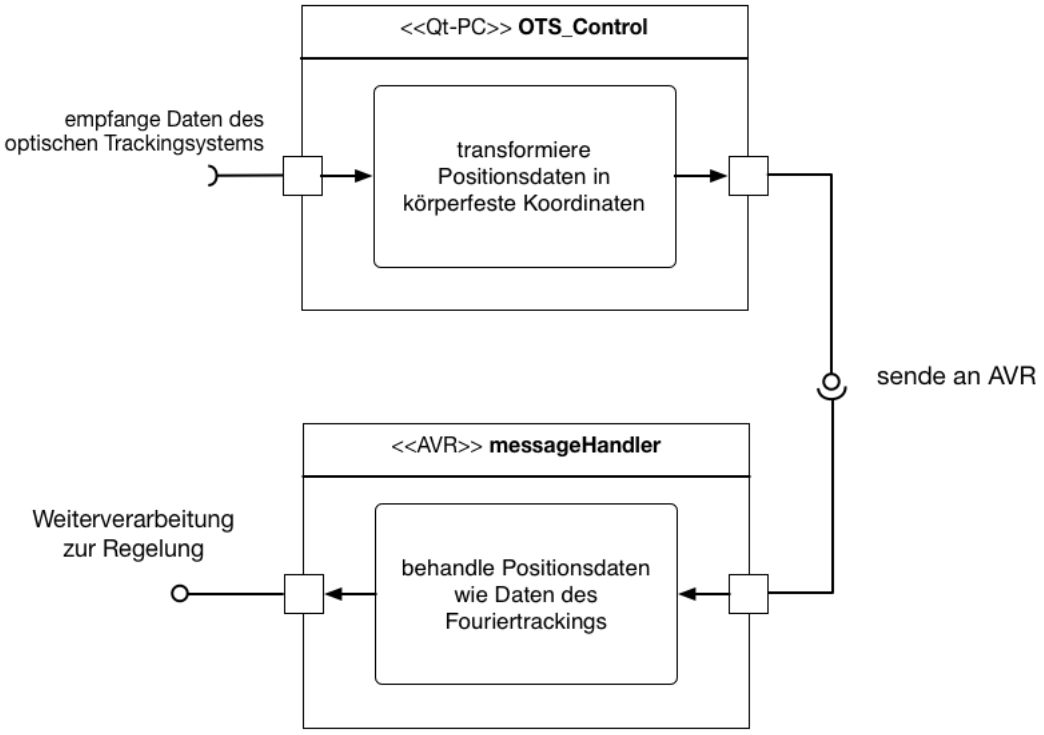

Abbildung 4.1.: Datenfluss der optischen Trackingdaten. Die Einspeisung der transformierten Positionsdaten im messageHandler erfolgt per `OF_FOURIER_UPDATE_POSITION`-Kommando. Der Qt-PC ist dabei der Windows-Rechner des Quadrokopters, und der AVR steht für den Mikrocontroller, der für die Regelung zuständig ist.

Das optische Tracking-System, das für die Positions- und Lageerkennung der Quadrokopter verwendet wird, beruht auf den *Flex 3*-Kameras der Firma OptiTrack. Die Kameras, die im Infrarot-Spektrum arbeiten, besitzen dabei eine Auflösung von 640x480 Pixeln und nehmen Bilder mit einer maximalen Framerate von 100fps auf. Mittels der zugehörigen Software *Motive* lässt sich, nachdem die Software mithilfe eines *Wanding*-Stabes wie in Kap.2.1 beschrieben, kalibriert wurde, ein Quadrokopter tracken. Dazu werden am Quadrokopter vier Marker angebracht, die das Infrarotlicht, das die an den Kameras angebrachten Infrarot-LEDs aussenden, reflektieren. Diese Marker werden dabei nicht symmetrisch am Quadrokopter angebracht, da sonst zweideutige Lage- bzw. Positionsdaten entstehen können. Es empfiehlt sich auch die Nutzung von mindestens vier Markern, da unter Umständen einer der Marker verdeckt und damit von der Software nicht

erkannt werden kann, was bei Nutzung von weniger als vier Markern zu unkorrekten oder keinen Positions- und Lagedaten führt.

Sobald die Marker in den Sichtbereich der Kameras kommen und von der Tracking-Software erkannt werden, können die Marker zu einem sogenannten *Rigid Body*, einem festen Körper zusammengefasst werden, für den die 6DoF-Daten, also X,Y,Z - Koordinaten, sowie die Lagedaten in Form eines Quaternions, ausgegeben werden.

Diese Daten können nun über ein VRPN, ein *Virtual Reality Private Network*, mittels einer vom Hersteller zur Verfügung gestellten Bibliothek (DLL) abgefragt werden. Eine solche Abfrage der Daten ist, wie in Kapitel 3.1 dargelegt, bereits möglich und wird in der Klasse `opticaltracking` mit der erwähnten Bibliothek realisiert. Um nun die empfangenen Daten weiter nutzen zu können, wird die genannte Klasse um ein *Signal* `sendOTSData` erweitert, und mit einem *Slot* `trackingData` in der `OTS_Control`-Klasse verbunden.

Die so vom optischen Tracking System erhaltenen Daten liegen allerdings in raumfesten Koordinaten vor, die Schnittstelle auf dem AVR, die für die Positionsregelung genutzt werden soll, erwartet jedoch körperfeste Koordinaten. Die Schnittstelle wird dabei eigentlich von der `NavigationControl`-Klasse verwendet, um die mittels des optischen Flusssensors gewonnen Positionsdaten zur Positionsregelung zu nutzen. Nachdem nun die vom OTS empfangenen Positionsdaten in körperfeste Koordinaten transformiert sind, werden diese mit dem Kommando `OF_FOURIER_UPDATE_POSITION` an den AVR gesendet, wo sie wie Daten des optischen Flusssensors behandelt werden. Das Senden der Daten ist dabei in der Funktion `sendPositionAVR` realisiert. Abbildung 4.1 veranschaulicht diesen Ansatz. Zu beachten ist dabei, dass nur die X- und Y- Komponenten der Position genutzt werden, die Höhenregelung zunächst also außer Acht gelassen wird.

4.3. Kommunikation zwischen Quadrokoptern

Die Kommunikation zwischen den Quadrokopter und der Groundstation basiert, wie in Kapitel 3 erläutert, auf WLAN, wobei auch hier auf ein bereits implementiertes Modul zurückgegriffen werden kann, das allerdings etwas erweitert werden muss.

Dabei handelt es sich um die beiden Klassen `Client` sowie `CommServer`, mit deren Hilfe sich ein *Client* zu einem *Server* unter Verwendung des TCP-Protokolls (näher erläutert in Kap.

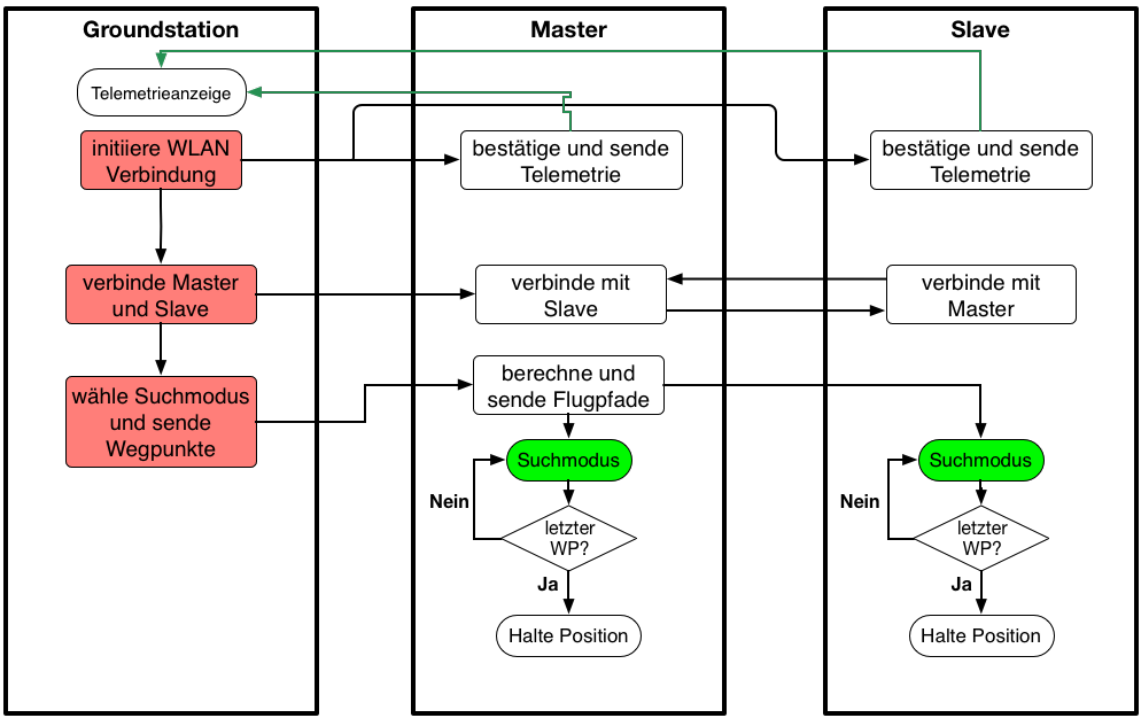

Abbildung 4.2.: Kommunikationsfluss zwischen Groundstation, Master und Slave. Rot markiert dabei Aktivitäten die vom Nutzer gestartet werden müssen

2.3.3.2) verbinden kann. Zu einem Server können sich dabei beliebig viele Clients verbinden, und vordefinierte Nachrichten senden.

Da diese vordefinierten Nachrichten für die aktuelle Problemstellung nicht zielführend sind, müssen eigene Nachrichten definiert werden. Dabei wird sich für die Trennzeichen der Nachrichten an der Syntax orientiert, die auch für die Kommunikation zwischen Windows-PC und AVR mittels sogenannter *Frames* genutzt wird, um eine größtmögliche Kompatibilität, im Falle einer zukünftigen Verwendung zu gewährleisten.

Deshalb wird ein eigener *MessageType*, `OTS_COMMANDED`, definiert, der verschiedene Befehle tragen kann. Tabelle A.1 gibt einen Überblick über die für die Problemstellung wichtigen Befehle und deren Parameter, die an `OTS_COMMANDED` angehängt werden können. Eine vollständige Tabelle, die auch verschiedene Debug-Kommandos und weitere Flugmodi enthält, die für die Problemstellung zunächst nicht relevant sind, findet sich im Anhang **??**. Für das Empfangen der Nachrichten werden die Klassen `Client` und `CommServer` noch um einen Parser erweitert, der die in Tab. A.1 definierten Befehle dechiffrieren kann. Mehrere Befehle können dabei auch direkt hintereinander gesendet werden.

Kommando	Argument	WP - X	WP -Y	Beschreibung
OTS_CONNECT_OTS	1=an; 0=aus	-	-	schaltet Empfang der OTS-Daten ein
OTS_CONNECT_GS	IP ; Modus	-	-	verbindet sich zur Groundstation mit IP und setzt Master- bzw. Slave Modus
OTS_CONNECT_SLAVE	IP-Adresse	-	-	Master verbindet sich zum Slave mit IP
OTS_START_COMMANDING	1=an; 0=aus	-	-	Startet das Absuchen
OTS_SEARCHMODE	A oder P	-	-	A = Flächensuche, P = Wegpunktsuche
OTS_WAYPOINT	-	X	Y	Koordinaten eines Wegpunkts

Tabelle 4.1.: wichtige Befehle zur Kommandierung der Quadrokopter. (WP = Wegpunkt)

Die Software ist dabei so gestaltet, dass die gleiche Version der Software sowohl auf dem Master-Quadrokopter, dem Slave-Quadrokopter und der Groundstation läuft, wobei die Groundstation per Mausklick aktiviert werden muss. (vergleiche dazu Abb. 4.3 in Kap. 4.5). Hat man nun die Groundstation aktiviert, können zunächst der Master und dann der Slave durch Angabe ihrer jeweiligen IP-Adresse mit der Groundstation verbunden werden. Dies geschieht durch den Aufbau einer TCP-Verbindung zur angegebenen IP. Ist die Verbindung erfolgreich aufgebaut wird das Kommando `OTS_CONNECT_GS` gesendet, wodurch dem jeweiligen Quadrokopter mitgeteilt wird, in welchen Modus, Master oder Slave, er schalten soll. Zusätzlich wird, beim Empfang des Kommandos, eine Verbindung von beiden Quadrokoptern zur Groundstation aufgebaut, um Telemetriedaten zu übertragen. Die Telemetriedaten bestehen dabei hauptsächlich aus Quittierungen der empfangenen Nachrichten, es können auf Wunsch aber auch Debug-Daten übertragen werden, die zum Beispiel den aktuellen Modus, die aktuelle Position, die nächste Soll-Position etc. enthalten. Danach müssen die beiden Quadrokopter miteinander und mit dem optischen Tracking verbunden werden, was mithilfe der Kommandos `OTS_CONNECT_SLAVE` und `OTS_CONNECT_OTS` realisiert wird. Der Master stellt dabei eine Verbindung zum Slave mit der übergebenen IP her. Die Verbindung zum optischen Tracking System wird dabei, wie in Kapitel 4.2 beschrieben, über ein VRPN verwirklicht.

Um nun eine Wegpunkt- oder Flächensuche zu starten, sendet die Groundstation mit dem Kommando `OTS_SEARCHMODE` den entsprechenden Modus, und danach die Wegpunkte, be-

ziehungsweise die Fläche, die abgesucht werden soll, an den Master. Das Zusammenstellen der Wegpunkte und der Fläche wird jedoch erst in Kapitel 4.5 genauer erklärt.

Der Master empfängt nun das Kommando und die Wegpunkte/Fläche und berechnet den Flugpfad in Form einer geordneten Wegpunktliste für beide Quadrokopter. Ist die Berechnung abgeschlossen, wird dem Slave die entsprechende abzufliegende Wegpunktliste gesendet. Danach starten der Master und der Slave das selbstständige Abfliegen der berechneten Wegpunktliste. Abb. 4.2 veranschaulicht diesen Kommunikationsfluss zwischen Groundstation, Master und Slave.

Das selbstständige Abfliegen der Wegpunktliste wird dabei mit Hilfe der Funktion `OTS_Commanding` realisiert. Hier wird ein Wegpunkt aus der abzufliegenden Wegpunktliste in raumfesten Koordinaten an die Flugsoftware auf dem Mikrocontroller gesendet, die sich um den Flug zum gegebenen Wegpunkt kümmert, und periodisch überprüft, ob der Quadrokopter den übertragenen Wegpunkt bereits erreicht hat. Ist der Quadrokopter nach einer gewissen Zeit nicht am Wegpunkt angekommen, wird der Wegpunkt erneut gesendet, solange bis der Quadrokopter den Wegpunkt erreicht. Ist dies der Fall, wird der nächste Wegpunkt gesendet. Diese Vorgehensweise wird dabei solange wiederholt, bis der Quadrokopter den letzten Wegpunkt erreicht. An diesem letzten Wegpunkt hält der Quadrokopter seine Position und wartet auf neue Befehle.

4.4. Suchalgorithmen

Im Folgenden werden nun die Implementierungen der Wegpunkt- sowie der Flächensuche beschrieben.

4.4.1. Implementierung Wegpunktsuche

Um das Testen des algorithmischen Konzepts 3.4.1 und dessen Evaluierung zu vereinfachen, wurde dieser Teil der Arbeit in einer eigenen Klasse `waypointalgorithm` und zunächst unabhängig von der Quadrokopter-Software implementiert. Der Fokus der Implementierung liegt dabei beim Finden der optimalen Pfade für zunächst zwei Quadrokopter.

Die Funktion `calculateShortestPath` in der Klasse nimmt dabei zwei Argumente, eine Liste mit Wegpunkten und eine Liste mit Startpunkten entgegen. Aus diesen übergebenen Punkten wird zunächst eine sogenannte *Lookup Table* erstellt, die die Pfadlängen zwischen je-

weils zwei Punkten enthält, sodass man diese später nicht immer wieder berechnen muss. Nun werden die Anzahl der Wegpunkte, wie in Def. 3.4 beschrieben, möglichst gleichmäßig auf die Anzahl der Startpunkte verteilt und für jeden Wegpunkt die Kombinationen mit der zugeteilten Anzahl an Wegpunkten aus der Gesamtanzahl der Wegpunkten berechnet. Werden zum Beispiel 19 Wegpunkte und 2 Startpunkte übergeben, werden dem ersten Startpunkt 10 und dem zweiten Startpunkt 9 Wegpunkte zugewiesen. Dementsprechend werden für den ersten Startpunkt alle Kombinationen mit 10 aus 19, und für den zweiten Startpunkt 9 aus 19 Wegpunkten berechnet, was mit der rekursiven Funktion `getCombinations` erledigt wird.

Für jede dieser Wegpunktkombinationen wird nun durch simples Berechnen und Vergleichen eine, wie in Kapitel 3.4.1 beschriebene, kürzeste Anordnung bestimmt, wobei die Länge der Wegpunktkombination immer mit abgespeichert wird. Nun wird mit der Funktion `sortData` das Array, das die Wegpunktkombinationen und ihre Länge enthält, nach der Länge der Wegpunktkombinationen sortiert. `sortData` ist dabei eine Implementierung des QuickSort-Algorithmus. Jetzt geht der Wegpunkt-Algorithmus dieses sortierte Array von oben nach unten, das heißt von der kürzesten bis zur längsten Wegpunktkombination für jeden Startpunkt durch, und überprüft, ob die Wegpunktkombinationen alle abzufliegenden Wegpunkte enthalten und ob keine Wegpunkte mehrfach vorkommen.

Ist dies nicht der Fall, wählt der Algorithmus für denjenigen Startpunkt die nächste Wegpunktkombination, bei dem die Änderung der Pfadlänge, sollte die nächste Wegpunktkombination genommen werden, am geringsten ist. Dann wird erneut überprüft, ob alle anzufliegenden Wegpunkte enthalten sind, und so weiter. Damit erhält der Algorithmus garantiert eine Wegpunktkombination für alle Startpunkte. Wie in 3.4.1 bereits erwähnt, ist diese Wegpunktkombination unter Umständen nicht optimal, was in Kapitel 5.1.1 näher betrachtet wird.

4.4.2. Implementierung Flächensuche

Die Implementierung der Flächensuche stützt sich, wie in Kapitel 3.4.2 beschrieben, auf ein bereits existierendes Modul zur Flächensuche, der Klasse `waypoint_generator`. Hier können für verschiedene Suchmethoden, wie die in 3.4.2 erwähnten *Tiefen-* und *Breitensuche*, Wegpunkte in Abhängigkeit von den Abmessungen der gegebenen Fläche, sowie vom Sichtbereich des Quadrokopters generiert werden. Als Randbedingung für die Implementierung soll dabei gelten, dass die beiden Quadrokopter immer nebeneinander starten, das heißt, dass die beiden Quadrokopter

einen ähnlichen Y-Koordinatenwert haben (die Y Achse zeigt dabei nach oben, die X-Achse nach rechts).

Eine Einschränkung der `waypoint_generator`-Klasse ist jedoch, dass Wegpunkte für nur einen Quadrokopter generiert werden können, der zusätzlich seinen Startpunkt immer im Koordinatenursprung hat. Für die kooperative Suche müssen allerdings unterschiedliche Startpunkte gewählt werden können und mehrere Pfade generiert werden.

Da im vorliegenden Fall das Problem der Flächensuche auf ein Rechteck reduziert ist, die auf die zwei Quadrokopter aufgeteilte Fläche also wieder einem Rechteck entspricht, wobei die beiden Flächen exakt gleich sind, kann das existierende Modul dennoch verwendet werden. Der Wegpunktgenerator wird dazu einfach mit den Abmessungen der halbierten Fläche aufgerufen, und die dadurch generierten Wegpunkt im Nachhinein dupliziert. Mit der oben genannten Randbedingung der Implementierung, nach der die Startpunkte der Quadrokopter auf gleichem Y-Wert liegen, kann auf die duplizierten Wegpunkte einfach ein *Offset* in X-Richtung addiert werden, um die Wegpunkte des Pfades für den zweiten Quadrokopter zu erhalten (siehe dazu auch Abb. 4.4).

Dem Problem des festen Startpunkts bei der Wegpunktgenerierung kann ebenfalls durch die Nutzung eines *Offsets* begegnet werden. Dabei entspricht der *Offset*, der auf die generierten Wegpunkte addiert wird, genau den Koordinaten des Startpunkts des ersten Quadrokopters.

4.5. Graphische Benutzeroberfläche

Um die in den vorangegangenen Kapiteln erläuterten Funktionen nutzen zu können, wurde die vorhandene graphische Oberfläche um zwei neue Reiter im *Control*-Bereich erweitert: einem Reiter *OTS Control* und einem Reiter *OTS Simulation*.

OTS Control ist dabei in 4 Bereiche, in Abb. 4.3 rot eingezeichnet, gegliedert: dem Bereich für das optische Tracking, für die Kontrolle der Quadrokopter, einem Bereich für Eingabe/Ausgabe und zuletzt eine Karte. Im Folgenden werden die einzelnen Bereiche genauer erklärt.

1. **Kontrollbereich für das optisches Tracking**

   Dieser Bereich ist auf jedem Quadrokopter verfügbar, das heißt, die Einstellungen in diesem Bereich können auf jedem Quadrokopter unabhängig voneinander eingestellt werden. Ändert man jedoch diese Einstellungen auf der Groundstation, werden diese an die verbundenen Quadrokopter gesendet.

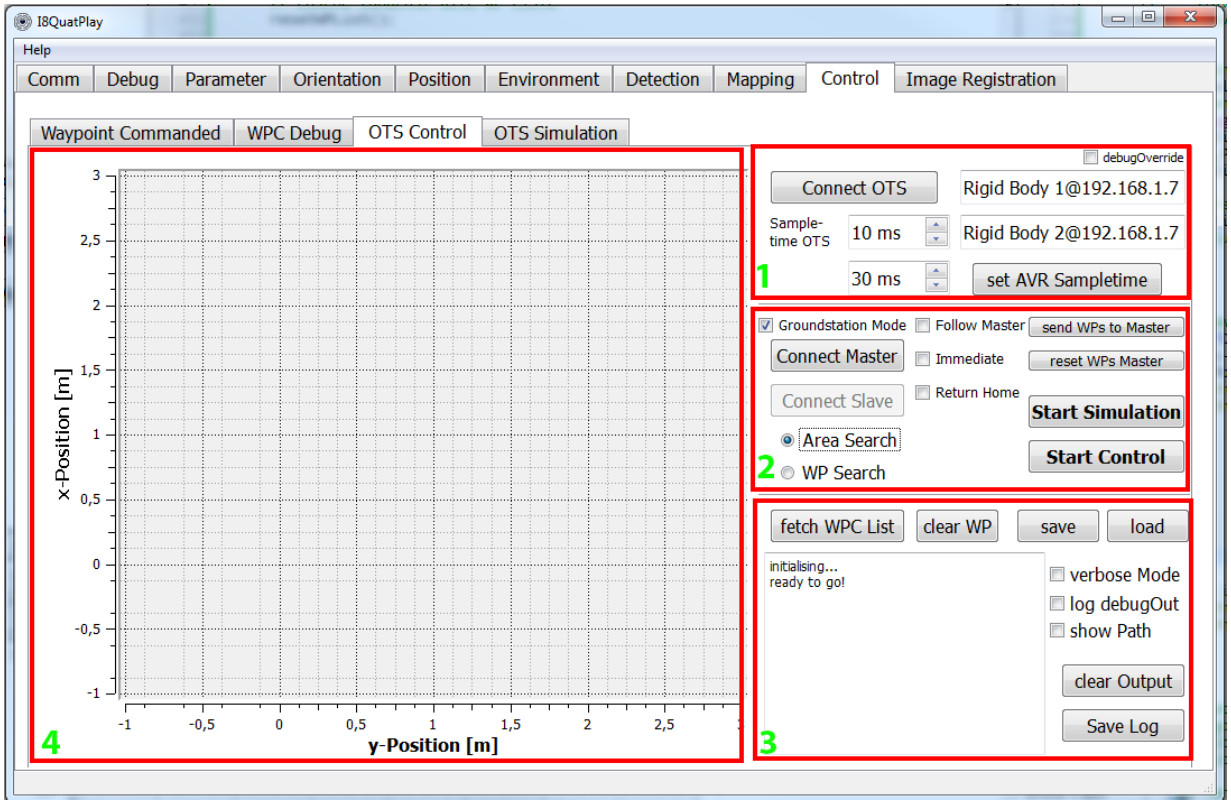

Abbildung 4.3.: Überblick der implementierten GUI für die Kontrolle zweier Quadrokopter

- Der Button **Connect OTS** verbindet den Master bzw. den Slave mit dem optischen Trackingsystem. Die Groundstation
- Unterhalb des Buttons lässt sich die **Sampletime** für das **optische Tracking** einstellen. Mit diesem Zeitintervall wird vom optischen Tracking ein Update der Positionsdaten angefordert.
- Wiederum unterhalb, kann die Sampletime für das Senden der Positionsdaten an den AVR eingestellt werden. Durch Klicken auf **set AVR Sampletime** wird diese auf allen verbundenen Quadrokoptern gesetzt.
- Die beiden **Textfelder** auf der rechten Seite enthalten den Namen des *Rigid Bodys*, der in der Software des optischen Trackings erstellt wurde (sieh dazu Kap. 4.2), sowie die IP Adresse, auf der die Software des optischen Trackings läuft. Zu beachten ist, dass das obere Textfeld dem Master entspricht, das untere dem Slave.
- Mithilfe der checkBox **debugOverride** lässt sich der Bereich 2 aktivieren, auch wenn die Software nicht auf der Groundstation, sondern auf einem Quadrokopter läuft. Da-

durch kann die Software auch zur Kontrolle eines Quadrokopters verwendet werden, wenn keine Groundstation genutzt wird.

2. **Kontrolle der Quadrokopter**

   Dieser Bereich ist standardmäßig deaktiviert, kann auf der Groundstation aber durch anklicken der checkBox **Groundstation Mode** aktiviert werden.

   - Die beiden Buttons **Connect Master** und **Connect Slave** verbinden die Groundstation mit dem jeweiligen Quadrokopter. Solange kein Master verbunden ist, kann auch kein Slave verbunden werden. Wenn jedoch ein Slave mit der Groundstation verbunden wird, wird dieser automatisch auch mit dem Master verbunden (vgl. dazu den Kommunikationsfluss in Kap. 4.3)
   - **Area Search** bzw. **WP Search** setzt den Suchmodus auf die Flächen- bzw. Wegpunktsuche.
   - Aktiviert man den **Follow Master**-Modus, folgt der Slave dem Master im Abstand von einem Wegpunkt.
   - Der **Immediate**-Modus kann nur aktiviert werden, wenn nur der Master verbunden ist. Dabei wird ein Wegpunkt sofort nachdem er in Bereich 4 angeklickt wurde, an den Quadrokopter gesendet.
   - Ist die **Return Home**-checkBox und die *Area Search* aktiviert, kehren die Quadrokopter nach erfolgreichem Absuchen der Fläche an ihren Ausgangspunkt zurück.
   - **send WPs to Master** bzw. **reset WPs Master** sendet die in Bereich 4 angeklickten Wegpunkte an den Master bzw. setzt die Wegpunkte zurück.
   - **Start Simulation** aktiviert den Reiter *OTS Simulation* und startet die Simulation einer Wegpunktsuche (siehe Abb. 4.4).
   - **Start Control** startet die Berechnung der abzusuchenden Fläche bzw. Wegpunkte und startet die Suche.

3. **Bereich für Ein- und Ausgabe**

   - Mittels **fetch WPC List** lassen sich die im Reiter *Waypoint Commanded* erstellten Wegpunkte übernehmen.

- **clear WP** löscht die in Bereich 4 erstellten Wegpunkte
- Mit den Buttons **save** und **load** lässt sich eine Wegpunktliste speichern und laden.
- Ist der **verbose Mode** aktiv, werden zusätzlich zur normalen Ausgabe erweiterte Informationen im nebenstehenden Textfeld angezeigt.
- **log debugOut** speichert die Ausgabe, die im Textfeld angezeigt wird, sodass sie mit dem Button **Save Log** in eine Datei gespeichert werden kann.
- **show Path** blendet den von den Quadrokoptern zurückgelegten Weg in die Karte ein, sofern das optische Tracking aktiv ist.
- der Button **clear Output** löscht die Ausgabe im nebenstehenden Textfeld

4. **die Karte**

   In der Karte können beliebige Wegpunkte gesetzt werden, die dann je nach Suchmodus als Wegpunkte, oder als Begrenzung der Suchfläche angesehen werden.

Der zweite Reiter, *OTS Simulation*, simuliert den Abflug einer Flächensuche mit wahlweise einem oder zwei Quadrokoptern. Der Reiter ist dabei verknüpft mit *OTS Control*, da die in *OTS Control* eingegebenen Wegpunkte für die Simulation genommen werden. Abb. 4.4 illustriert diesen Reiter. Analog zu *OTS Control* besitzt auch dieser Reiter auf der linken Seite eine Karte, hier wird anstatt der Wegpunkte der abzufliegende Pfad eingezeichnet, der aus der in *OTS Control* ausgewählten Fläche besteht.

Auf der rechten Seite lässt sich im oberen Bereich die Anzahl der Quadrokopter für die Flächensuche einstellen, sowie deren Startpuntke festlegen. Der mittlere Bereich, der in Abb. 4.4 Nullen anzeigt, stellt, wenn die Simulation gestartet wird, verschiedene Parameter dar, so zum Beispiel die IST- und SOLL-Position, die Anzahl der abgeflogenen Wegpunkte, sowie die gesamte Pfadlänge und die abgesuchte Fläche.

Der untere und letzte Bereich bietet verschiedene Einstellmöglichkeiten zur Simulation an. Mit **Step size** wird dabei die Schrittweite festgelegt, mit welcher, bei Klick auf **Previous Step** oder **Next Step**, ein Schritt in der Simulation vor oder zurück gemacht wird. Das Einstellen der Flughöhe war ursprünglich geplant, ist aber in dieser Software-Version nicht aktiv, da auch in *OTS Control* nur die X-Y-Ebene betrachtet wird. Mit dem **Safety Radius** lässt sich der Radius um den Quadrokopter einstellen, der nicht von anderen Quadrokoptern gekreuzt werden darf.

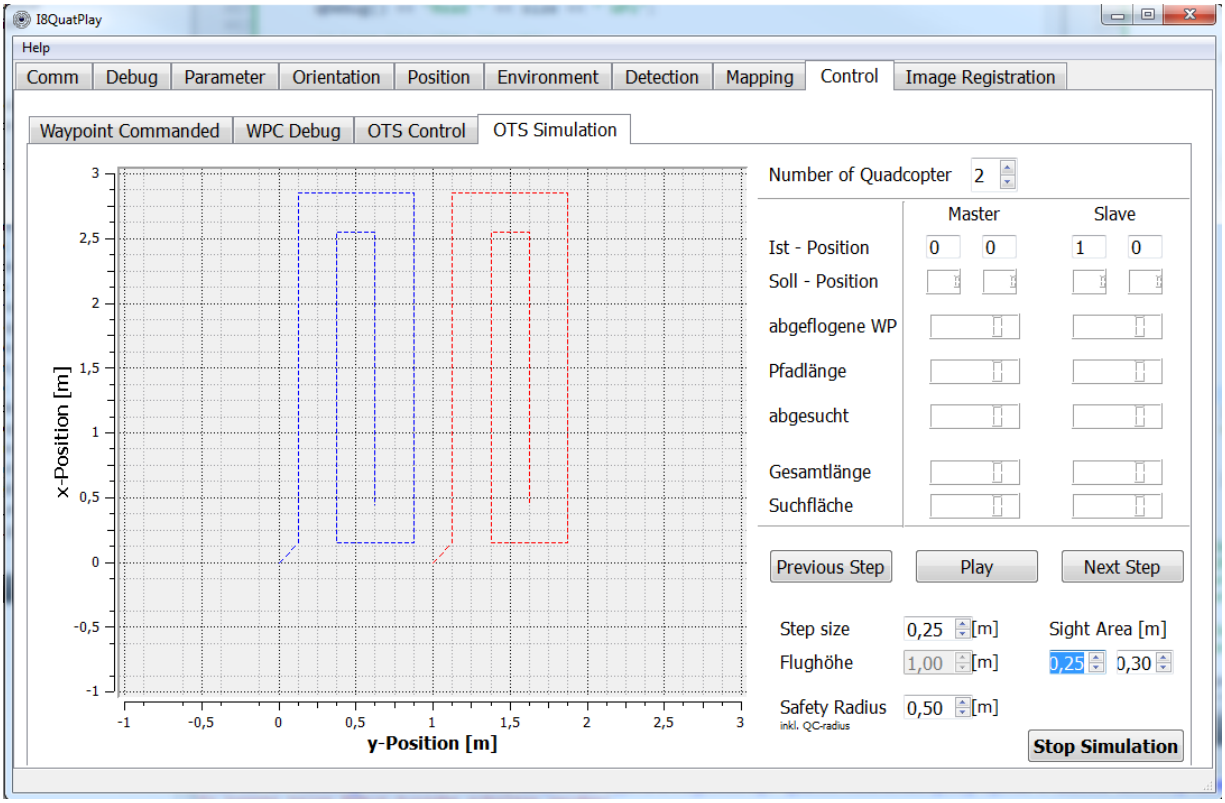

Abbildung 4.4.: Simulation des Abfluges einer Flächensuche

Die **Sight Area** beschreibt das Sichtfeld des Quadrokopters auf dem Boden, was natürlich den abzufliegenden Pfad beeinflusst, da die gesamte gewählte Fläche abgesucht werden muss.

Schlussendlich gibt es noch den **Play/Pause**-Button, der den Flug der Quadrokopter links in der Karte mit der eingestellten Schrittweite abspielt. **Stop Simulation** beendet letztlich die Simulation und wechselt wieder in den *OTS Control*-Reiter.

5. Evaluierung

In diesem Kapitel wird die Implementierung der Wegpunkt- und Flächensuche, sowie die Nutzung des optischen Trackingsystems für die Positionsregelung hinsichtlich des Problems der kooperativen Suche evaluiert.

5.1. Evaluierung der Suchansätze

5.1.1. Tiefen- und Breitensuche

In Kapitel 4.4.2 wurden zwei Suchmethoden, die Breiten- und Tiefensuche, implementiert, bzw. erweitert, sodass diese für eine kooperative Suche angewandt werden können.

Mit Hilfe des implementierten GUI-Reiters *OTS Simulation* wurden die beiden Methoden für zwei Quadrokopter im Hinblick auf Effizienz untersucht, wobei in diesem Fall eine Methode als *effizienter* angesehen wird, wenn sie kürzere Pfade und weniger *turns*, also Richtungsänderungen, für die kooperierenden Quadrokopter generiert. Um die Suchmethoden vergleichen zu können, werden beide auf einer rechteckigen Fläche mit Kantenlängen 3m x 2m angewandt, wobei den Quadrokoptern ein Sichtbereich von 0,5m x 0,5m und die Startpunkte $(0|0)$ und $(0|1)$ zugewiesen werden. Bereits auf den ersten Blick lässt sich erkennen, dass die Tiefensuche weniger

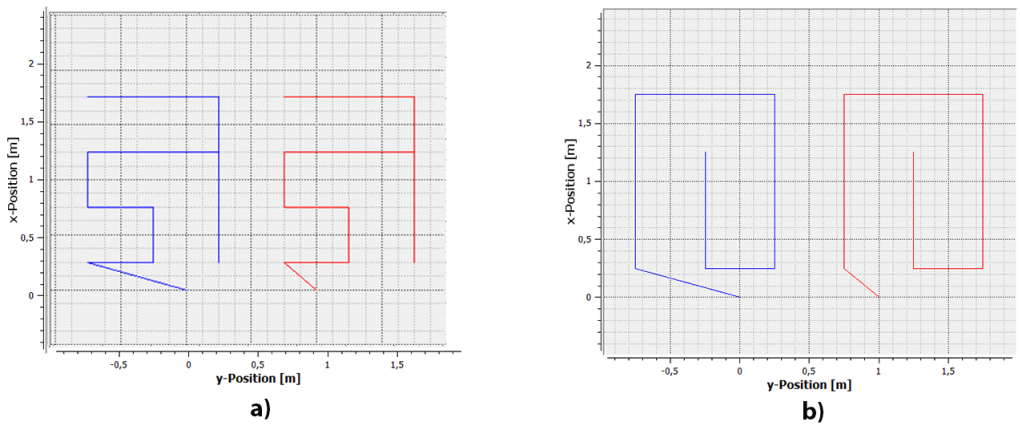

Abbildung 5.1.: Vergleich der **a)** Breitensuche und der **b)** Tiefensuche für die kooperative Suche zweier Quadrokopter

Richtungsänderungen generiert. Auch die Pfadlänge ist bei der Tiefensuche mit einer Gesamtlänge von 12,58m kürzer als die Breitensuche mit 15,14m. Es muss jedoch beachtet werden, dass die Pfadlänge mit der Wahl der Startpunkte variiert werden kann.

Vergleicht man die Tiefensuche mit der in Kapitel 3.4.2 genannten *Boustrophedon*-Methode (siehe Abb.), stellt man fest, dass diese zwar einen anderen Pfad generieren würde, der aber gleich lang wäre und gleich viele Richtungsänderungen hätte. Die Tiefensuche scheint, für diesen vereinfachten Fall der Flächensuche, dementsprechend gut zu funktionieren und sollte beim Absuchen einer Fläche der Breitensuche bevorzugt werden, falls ein ähnliches Suchgebiet vorliegt.

5.1.2. Algorithmus zur Aufteilung von Wegpunkten

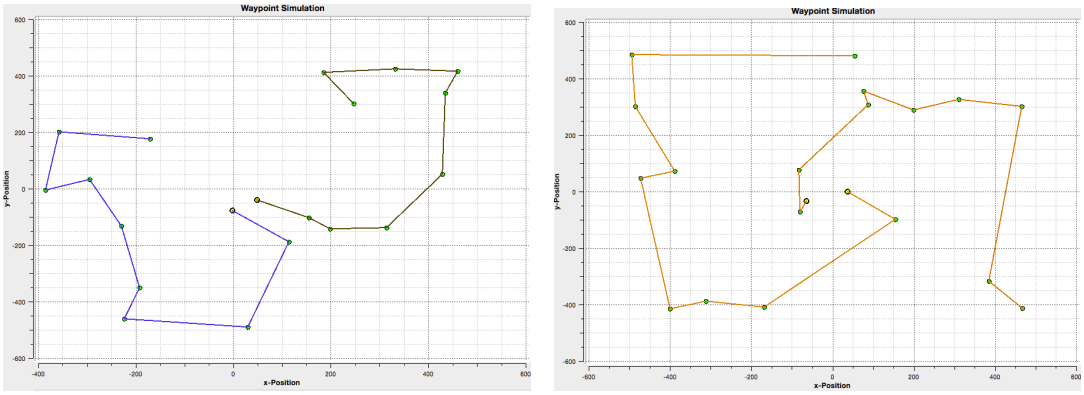

Abbildung 5.2.: Möglichst kurze Pfade aus je 18 Wegpunkten (grün) und je 2 Startpunkten (gelb). Die Punkte wurden dabei zufällig generiert.

Das Ziel des in 3.4.1 entwickelten Algorithmus war die Aufteilung von einer gegebenen Anzahl an Wegpunkten auf zwei Quadrokopter, sodass beide Pfade möglichst optimal sind. Das heißt, die beiden Pfade müssen in etwa gleich lang, dabei aber so kurz wie möglich sein, und die Berechnung soll in akzeptabler Zeit ablaufen. In Abb. 5.3 wird die Laufzeit des Algorithmus gegenüber der Anzahl der Wegpunkte aufgetragen und mit einer reinen Bruteforce-Methode verglichen. Die Laufzeit entspricht dabei der Durchschnittslaufzeit, die mithilfe von 10 Ausführungen pro Wegpunktanzahl ermittelt wurde. Die Weg- und Startpunkte werden dabei zufällig generiert.

Getestet wurde der Algorithmus sowie die Brute-Force-Methode auf einem Intel i5 Dual-Core mit 2,4GHz. Der Algorithmus scheint zunächst eine gute Alternative zur Brute-Force Methode zu sein, da bis zu 18 Wegpunkte in einer akzeptablen Zeit prozessiert werden können. Auch scheinen

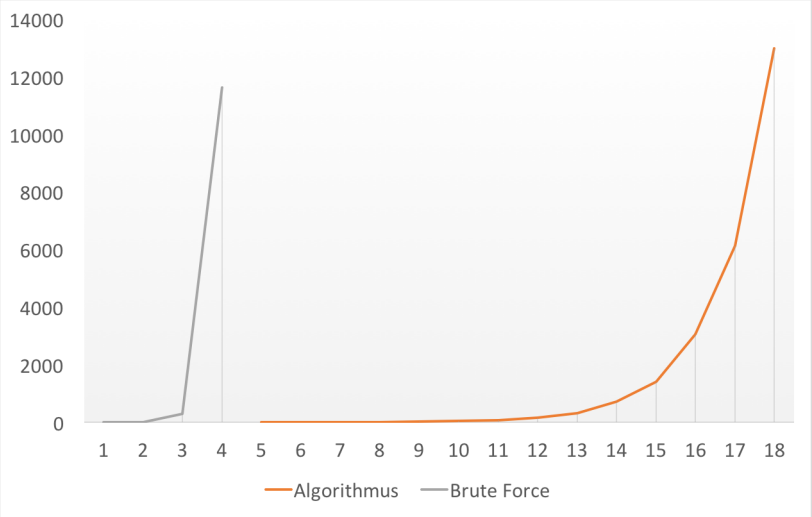

Abbildung 5.3.: Dauer der Pfadberechnung in Millisekunden für bestimmte Anzahl der Wegpunkte.

die erstellten Pfade für die zufällig generierten Wegpunkte relativ gut zu sein, siehe Abbildung 5.2.

Die größte Schwachstelle des Algorithmus ist allerdings die feste Zuteilung der Anzahl an Wegpunkten. Dadurch spart man sich zwar einige Berechnungen und Vergleiche, bei gewissen

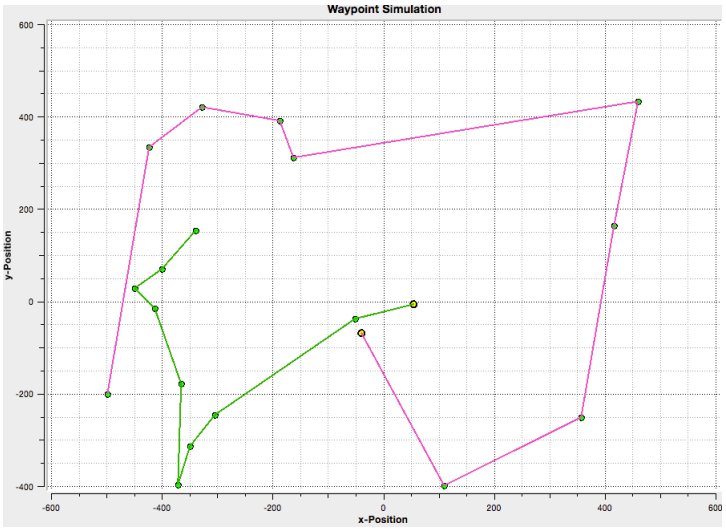

Abbildung 5.4.: Der Wegpunktalgorithmus berechnet unterschiedlich lange Pfade, da der grüne Pfad bereits die maximale Anzahl an Wegpunkten, die diesem Startpunkt zugeteilt wurden, erreicht hat. Gelbe Punkte sind Startpunkte, grüne Punkte sind Wegpunkte

Wegpunktkonstellationen entstehen so aber unterschiedlich lange Pfade, betrachte dazu Abb.5.4. Hier ist der pinke Pfad mehr als doppelt so lang als der grüne Pfad.

Eine Möglichkeit, den Algorithmus zu verbessern, wäre die Einführung eines Schwellenwerts für die Differenz der Pfadlängen, ab welchem eine Pfadkombination verworfen wird. Dadurch

würde eine Pfadkombination wie in Abb. 5.4 nicht auftreten. Alternativ könnte eine Variaton der Anzahl der zugewiesenen Wegpunkte erlaubt werden, was jedoch stark auf Kosten der Laufzeit gehen würde.

5.2. Nutzung des Optisches Tracking Systems

Die Evaluierung der Nutzung des optischen Tracking Systems ist in zwei Teile gegliedert. Zunächst wird die implementierte Software, die die Trackingdaten empfängt und an den Mikrocontroller des Quadrokopters sendet, auf Funktionalität überprüft. Im Zuge dieses Funktionalitätstests wird auch die Kommunikation, wie sie in Kapitel 4.3 beschrieben ist, getestet.

Ist dieser „Trockentest“ erfolgreich, wird überprüft, wie genau der Quadrokopter mithilfe der optischen Trackingdaten eine Position halten kann.

5.2.1. Empfang der Positionsdaten

Um die Funktionalität der implementierten Software für den Empfang der optischen Trackingdaten zu testen, wird zunächst ein WLAN-Netzwerk erstellt, zu dem sich sowohl der Rechner auf dem die optische Tracking Software *Motive* läuft, als auch ein Quadrokopter und zwei weitere Rechner verbinden. Einer der Rechner fungiert dabei als Groundstation, der andere Rechner wird als Ersatz für einen *Slave* verwendet (vgl. Kap. 4.3). Der Quadrokopter wird bei diesem Test als Master genutzt. Nachdem nun das optische Trackingsystem entsprechend kalibriert ist, wird der Groundstation-Rechner mit dem Master und dem Slave verbunden, und das optische Tracking von der Groundstation aus gestartet. Nun werden die Marker, die in der Software des optischen Tracking Systems als *Rigid Body* deklariert wurden, im Sichtbereich der Tracking-Kameras umhergeschwenkt, bzw. umhergetragen, siehe dazu Abb. 5.5.

Während dieses Tests ist aufgefallen, dass die Positionserkennung bereits durch schwache, indirekte Sonneneinstrahlung stark beeinflusst wird, sodass keine korrekten Positionsdaten mehr gesendet werden können, da die Marker nicht mehr erkannt werden. Ein weiterer Punkt ist, dass die Positionsdaten leicht, aber merklich der realen Bewegung der Marker hinterherhängen, das System also nicht in Echtzeit arbeitet.

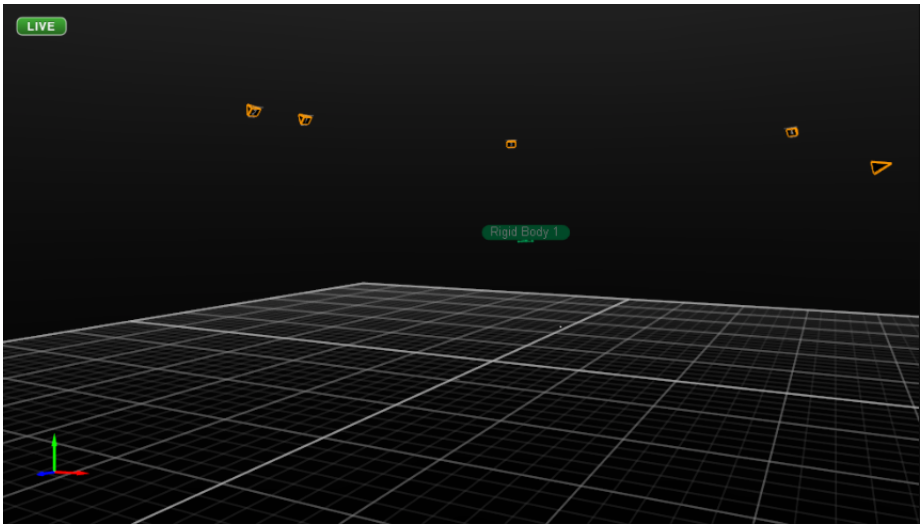

Abbildung 5.5.: Testen der optischen Tracking Verbindung mittels mehrerer Marker, die als Rigid Body deklariert sind

5.2.2. Statisches Regelverhalten

Nun soll festgestellt werden, wie gut der Quadrokopter mit den Positionsdaten des optischen Trackings seine Position halten kann. Dazu wird der Quadrokopter auf den Koordinatenursprung des optischen Trackings gestellt, und manuell gestartet. Hat der Quadrokopter eine bestimmte Höhe erreicht, wird die Positionsregelung für ein paar Minuten eingeschaltet, um zu überprüfen, dass diese funktioniert.

Bei den ersten Versuchen zum statischen Regelverhalten ist schnell aufgefallen, dass das ursprüngliche Konzept, die Schnittstelle für die Positionsdaten, und somit auch die Regelung des optischen Flusssensors zu nutzen, nicht geeignet ist, um den Quadrokopter mittels der Positionsdaten des optischen Trackings zu regeln. Dies liegt zum einen an den für den Regler verwendeten PID-Werten, da der Regler unter anderem die Qualität des optischen Flusses mitbeachtet, die bei Nutzung des optischen Tracking Systems natürlich nicht gegeben ist. Zum anderen scheint die Samplerate des optischen Trackings nicht stabil zu sein, da die Zeit zwischen den ankommenden Nachrichten mit Positionsupdates variiert, was Einfluss auf den Positionsregler hat.

Abbildung 5.6 veranschaulicht einen der ersten Versuche zum statischen Regelverhalten, wobei die P und D Werte in Richtung der X-Achse gegen die Zeit aufgetragen sind. Der Quadrokopter scheint sich nicht stabilisieren zu können, und der D-Anteil wird immer wieder bei +4 bzw. -4 gesättigt. Nach einer Modifikation der Steuersoftware auf dem Mikrocontroller, wobei ein neues Kommando `OTS_UPDATE_POSITION`, sowie eine Überprüfung der ankommenden Positions-

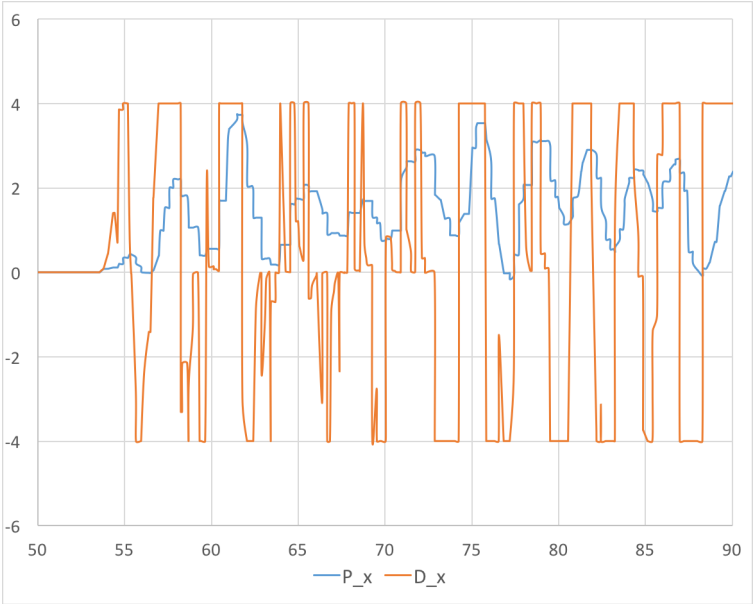

Abbildung 5.6.: Ein erster Versuch zum statischen Regelverhalten

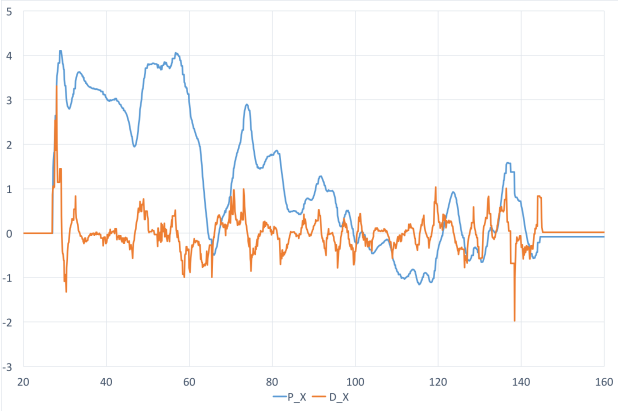

Abbildung 5.7.: Statisches Regelverhalten mit P=20.0 ; I=0.0005 ; D=1200.0. Es sind P und D-Werte in Richtung der X-Achse gegen die Zeit (in sec) aufgetragen.

daten und ein EWMA-Filter (*Exponentially Weighted Moving Average Filter*) für die stärkere Gewichtung aktueller Positionsdaten hinzugefügt wurde, konnten geeignetere PID-Werte gefunden werden. Abbildung 5.7 zeigt einen ca. zweiminütigen Testlauf bei dem sich die P- und D-Werte in Richtung X-Achse immer näher an die Nulllinie bewegen, und der Quadrokopter seine aktuelle Position relativ gut hält.

Allerdings gibt es auch immer wieder Ausreißer, wie zum Beispiel bei $t = 65$. Auch scheint sich der Quadrokopter nur sehr langsam an die gewünschte Position im Koordinatenursprung zu bewegen. Die Entfernung zum Ursprung variiert während des Versuch zwischen ca. 40 und 85cm.

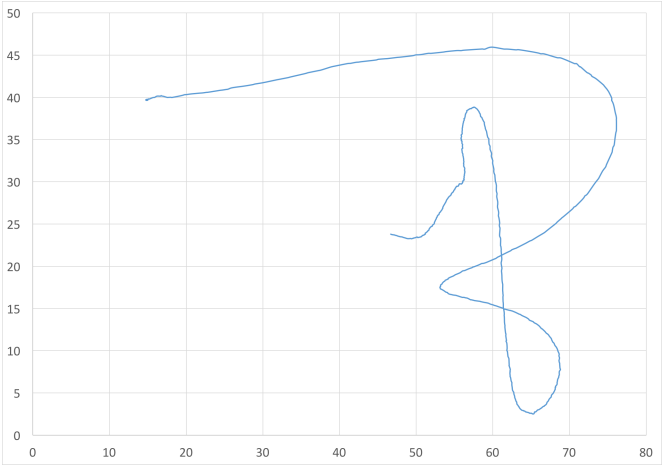

Abbildung 5.8.: Positionsdaten während des Versuchs zum statischen Regelverhalten in Abb. 5.7. Die X-Achse verläuft nach rechts, die Y-Achse nach oben. Werte sind in Zentimetern angegeben

Zum einen entsteht die relativ große Entfernung zum Koordinatenursprung dadurch, dass der Quadrokopter bereits zu Anfang des Versuchs nicht im Ursprung ist, zum anderen scheinen die PID-Werte noch nicht gut genug zu sein, da der Quadrokopter sich nur sehr langsam und mit Ausreißern - sowohl in X- als auch in Y-Richtung - zum Koordinatenursprung bewegt (siehe dazu Abb. 5.8).

6. Diskussion und Ausblick

Eine Hauptaufgabe dieser Arbeit war zunächst die Integration des bereits bestehenden optischen Tracking Systems in das bestehende Quadrokopter-Projekt. Da

Während der Evaluierung hat sich jedoch herauskristallisiert, dass das Konzept des „Einspeisens“ der vom optischen Tracking gelieferten Positionsdaten in die Schnittstelle des optischen Flusssensors zur Positionsregelung nicht funktioniert. Dies liegt hauptsächlich daran, dass der Regler für den optischen Flusssensor optimiert ist. Während der Evaluierung konnte zwar mithilfe einiger Workarounds ein akzeptables Ergebnis für das statische Regelverhalten erzielt werden, jedoch ist die Abweichung der Ist- von der Soll-Position noch sehr groß.

Da das optische Tracking an sich sehr genaue Positionsdaten liefert, besteht hier sehr viel Optimierungspotential hinsichtlich der Positionsregelung. Die wahrscheinlich effektivste Lösung wäre, eine separate Regelung für das optische Tracking System, bzw. für dessen Positionsdaten, in der Flugsoftware des Mikrocontrollers zu implementieren. Dadurch könnte man diese Regelung für das optische Tracking optimieren und gleichzeitig Kompatibilität zur Implementierung der Regelung des optischen Flusssensors wahren.

Eine weitere Aufgabe dieser Arbeit war das kooperative Lösen eines Problems mithilfe zweier Quadrokopter. Dafür wurde als Problemstellung ein kooperatives Absuchen von Wegpunkten und einer Fläche gewählt, wobei dieses Absuchen nur simuliert worden ist. Dafür wurde zum einen ein eigener, relativ schneller Algorithmus für die Aufteilung der Wegpunkte entworfen, der jedoch nicht immer gute Ergebnisse liefert. Eine Optimierungsmöglichkeit dafür wurde bereits in der Evaluierung in Kap. 5.1.2 erläutert. Zum anderen wurde, aufbauend auf dem vorhandenen implementierten Wegpunktgenerator undder Tiefensuche eine Möglichkeit entworfen, ein Absuchen einer Fläche auf mehrere Quadrokopter aufzuteilen. Dazu wurde eine Kommunikationsstruktur aus Groundstation, Master und Slave entwickelt mit welcher die Suche gesteuert werden kann, ohne auf die aktuell verwendete *Remotedesktopverbindung* zuzugreifen.

Da die genannte Flächensuche auf einem einfachen Rechteck beruht, wäre es denkbar, dass die kooperative Suche dahingehend erweitert wird, dass ein Raum, der auch Hindernisse enthalten kann, abgesucht wird.

7. Literaturverzeichnis

[Alani 2014] ALANI, Mohammed M.: *Guide to OSI and TCP/IP Models (SpringerBriefs in Computer Science)*. Springer, 2014. – URL http://dx.doi.org/10.1007/978-3-319-05152-9. – ISBN 3319051512

[Ali et al. 2014] ALI, Qasim ; GAGEIK, Nils ; MONTENEGRO, Sergio: A Review on Distributed Control of Cooperating Mini UAVS. In: *International Journal of Artificial Intelligence & Applications (IJAIA)* 5 (2014). – URL http://airccse.org/journal/ijaia/papers/5414ijaia01.pdf

[Anderson und Robbins 1998] ANDERSON, Mark ; ROBBINS, Andrew: Formation flight as a cooperative game. In: *Guidance, Navigation, and Control Conference and Exhibit*, American Institute of Aeronautics and Astronautics (AIAA), aug 1998. – URL http://dx.doi.org/10.2514/6.1998-4124

[Barth 2013] BARTH, Paul: *Bachelorarbeit - Bachelorarbeit Implementierung und Evaluierung verschiedener Algorithmen zur autonomen Suche eines Quadrokopters*. 2013

[Baun 2012] BAUN, Christian: *Computernetze kompakt*. Berlin : Springer Vieweg, 2012. – ISBN 978-3642289873

[Beard et al. 2006] BEARD, R.W. ; MCLAIN, T.W. ; NELSON, D.B. ; KINGSTON, D. ; JOHANSON, D.: Decentralized Cooperative Aerial Surveillance Using Fixed-Wing Miniature UAVs. In: *Proc. IEEE* 94 (2006), jul, Nr. 7, S. 1306–1324. – URL http://dx.doi.org/10.1109/JPROC.2006.876930

[Bluetooth-Spec] BLUETOOTH-SPEC: *aktuelle offizielle Bluetooth-Spezifikation Core Version 4.2*. – URL https://www.bluetooth.org/en-us/specification/adopted-specifications. – Zugriffsdatum: 16. April 2015

[Casbeer et al. 2005] CASBEER, D.W. ; LI, Sai-Ming ; BEARD, R.W. ; MEHRA, R.K. ; MCLAIN, T.W.: Forest fire monitoring with multiple small UAVs. In: *Proceedings of the 2005,*

American Control Conference, 2005., Institute of Electrical & Electronics Engineers (IEEE), 2005. – URL http://dx.doi.org/10.1109/ACC.2005.1470520

[Choset und Pignon 1997] CHOSET, Howie ; PIGNON, Philippe: Coverage Path Planning: The Boustrophedon Decomposition. In: *International Conference on Field and Service Robotics*, URL https://www.ri.cmu.edu/pub_files/pub1/choset_howie_1997_5/choset_howie_1997_5.pdf, 1997

[Gageik et al. 2012] GAGEIK, Nils ; MÜLLER, Thilo ; MONTENEGRO, Sergio: *Obstacle Detection and Collision Avoidance using ultrasonic distance sensors for an autonomous Quadrocopter*. 2012. – URL http://www8.informatik.uni-wuerzburg.de/fileadmin/10030800/user_upload/quadcopter/Paper/Gageik_Mueller_Montenegro_2012_OBSTACLE_DETECTION_AND_COLLISION_AVOIDANCE_USING_ULTRASONIC_DISTANCE_SENSORS_FOR_AN_AUTONOMOUS_QUADROCOPTER.pdf

[Gageik et al. 2014] GAGEIK, Nils ; REINTHAL, Eric ; BENZ, Paul ; MONTENEGRO, Sergio: Complementary Vision Based Data Fusion For Robust Positioning And Directed Flight Of An Autonomous Quadrocopter. In: *International Journal of Artificial Intelligence & Applications* 5 (2014), sep, Nr. 5, S. 01–19. – URL http://dx.doi.org/10.5121/ijaia.2014.5501

[Gao und Zhao 2014] GAO, Chun Y. ; ZHAO, Zhen Y.: A New Multiple UAVs Cooperative Search Model Building and Route Planning Method. In: *AMM* 701-702 (2014), dec, S. 160–166. – URL http://dx.doi.org/10.4028/www.scientific.net/AMM.701-702.160

[Gessler und Krause 2009] GESSLER, Ralf ; KRAUSE, Thomas: *Wireless-Netzwerke für den Nahbereich*. Springer Science & Business Media, 2009. – URL http://dx.doi.org/10.1007/978-3-8348-9601-8

[Huang 2001] HUANG, W.H.: Optimal line-sweep-based decompositions for coverage algorithms. In: *Proceedings 2001 ICRA. IEEE International Conference on Robotics and Automation (Cat. No.01CH37164)*, Institute of Electrical & Electronics Engineers (IEEE), 2001. – URL http://dx.doi.org/10.1109/ROBOT.2001.932525

[Jones 2009] JONES, Phillip J.: *Cooperative Area Surveillance Strategies using multiple Unmanned Systems*, Georgia Institute of Technology, Dissertation, 2009. – URL https://smartech.gatech.edu/bitstream/handle/1853/28134/jones_phillip_j_200905_phd.pdf

[Keil und Snoeyink 2002] KEIL, Mark ; SNOEYINK, Jack: On the Time bound for Convex Decomposition of Simple Polygons. In: *International Journal of Computational Geometry & Applications* 12 (2002), jun, Nr. 03, S. 181–192. – URL http://dx.doi.org/10.1142/S0218195902000803

[Kumar 2015] KUMAR, Vijay: *Aerial Robot Swarms*. Max-Planck-Lecture. 2015. – URL https://www.is.mpg.de/15769518/Aerial_Robot_Swarms. – Zugriffsdatum: 9. März 2015

[Li und Liu 2008] LI, Norman H. ; LIU, Hugh H.: Formation UAV flight control using virtual structure and motion synchronization. In: *Proc of the 2008 American Control Conf. Washington*, URL http://www.nt.ntnu.no/users/skoge/prost/proceedings/acc08/data/papers/0579.pdf, 2008, S. 1782–1787

[Lingelbach 2004] LINGELBACH, F.: Path planning using probabilistic cell decomposition. In: *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA '04. 2004*, Institute of Electrical & Electronics Engineers (IEEE), 2004. – URL http://dx.doi.org/10.1109/ROBOT.2004.1307193

[Maza und Ollero 2007] MAZA, Ivan ; OLLERO, Anibal: Multiple UAV cooperative searching operation using polygon area decomposition and efficient coverage algorithms. In: *Distributed Autonomous Robotic Systems 6*. Springer Science Business Media, 2007, S. 221–230. – URL http://dx.doi.org/10.1007/978-4-431-35873-2_22

[Maza et al. 2014] MAZA, Ivan ; OLLERO, Anibal ; CASADO, Enrique ; SCARLATTI, David: *Classification of Multi-UAV Architectures*. S. 953–975. In: VALAVANIS, Kimon P. (Hrsg.) ; VACHTSEVANOS, George J. (Hrsg.): *Handbook of Unmanned Aerial Vehicles*, Springer Netherlands, 2014. – URL http://dx.doi.org/10.1007/978-90-481-9707-1_119. – ISBN 978-90-481-9706-4

[Mazur 2014] MAZUR, Alexander: *Autonomous operation and control of a Multirotor Unmanned Aerial Vehicle through 4G LTE using onboard GPS and image processing*. 2014. – URL http://robotics.ee.uwa.edu.au/theses/2014-Hexacopter-Interface-Mazur.pdf. – Zugriffsdatum: 20. April 2015

[Mejias et al. 2014] MEJIAS, Luis ; LAI, John ; BRUGGEMANN, Troy: Sensors for Missions. In: VALAVANIS, Kimon P. (Hrsg.) ; VACHTSEVANOS, George J. (Hrsg.): *Handbook of Unmanned Aerial Vehicles*. Springer Netherlands, 2014, S. 385–399. – URL http://dx.doi.org/10.1007/978-90-481-9707-1_6. – ISBN 978-90-481-9706-4

[Menache 2011] MENACHE, Alberto: *Understanding motion capture for computer animation, 2nd Edition*. Burlington, MA : Morgan Kaufmann, 2011. – ISBN 978-0-12-381496-8

[Moon und Shim 2009] MOON, Sang-Woo ; SHIM, David Hyun-Chul: Study on path planning algorithms for unmanned agricultural helicopters in complex environment. In: *International Journal of Aeronautical and Space Sciences* 10 (2009), Nr. 2, S. 1–11

[OptiTrack] OPTITRACK: *Website zum Produkt OptiTrack der Firma NaturalPoints*. – URL http://www.optitrack.com/support/faq/mocap.html#tracking_accuracy. – Zugriffsdatum: 25. April 2015

[Sack und Urrutia 2000] SACK, Jörg-Rüdiger ; URRUTIA, Jorge: *Handbook of computational geometry 1st Edition*. Amsterdam New York : Elsevier, 2000. – ISBN 9780444825377

[Sathyaraj et al. 2008] SATHYARAJ, B. M. ; JAIN, L. C. ; FINN, A. ; DRAKE, S.: Multiple UAVs path planning algorithms: a comparative study. In: *Fuzzy Optimization and Decision Making* 7 (2008), jun, Nr. 3, S. 257–267. – URL http://dx.doi.org/10.1007/s10700-008-9035-0

[Scherff 2010] SCHERFF, Jürgen: *Grundkurs Computernetzwerke : eine kompakte Einführung in Netzwerk- und Internet-Technologien*. Wiesbaden : Vieweg + Teubner, 2010. – ISBN 9783834803665

[Smolka 2013] SMOLKA, Joshua: *Bachelorarbeit - Entwicklung einer ad-hoc W-Lan Gui-basierten Steuerung von multiplen Quadrokoptern*. 2013

[Stancheva 2003] STANCHEVA, Galina: *Interaktive Visualisierung von Algorithmen für die Polygonzerlegung*, Ludwig-Maximilians-Universität München, Dissertation, 2003. – URL http://www.pms.ifi.lmu.de/publikationen/diplomarbeiten/Galina.Stancheva/DA.ps.gz

[Stentz 1997] STENTZ, Anthony: Optimal and Efficient Path Planning for Partially Known Environments. In: *Intelligent Unmanned Ground Vehicles*. Springer US, 1997, S. 203–220. – URL http://dx.doi.org/10.1007/978-1-4615-6325-9_11

[Tönnis 2010] TÖNNIS, Marcus: *Augmented Reality - Einblicke in die Erweiterte Realität*. Berlin, Heidelberg : Springer-Verlag Berlin Heidelberg, 2010. – URL http://dx.doi.org/10.1007/978-3-642-14179-9_1. – ISBN 978-3-642-14179-9

[Tsourdos et al. 2011] TSOURDOS, Antonios ; WHITE, Brian ; SHAMMUGAVEL, Madhavan: *Cooperative path planning of unmanned aerial vehicles*. Chichester, West Sussex, U.K. Hoboken, N.J : Wiley, 2011. – ISBN 9780470741290

[Valavanis 2010] VALAVANIS, K: *Selected papers from the 2nd International Symposium on UAVs, Reno, U.S.A. June 8-10, 2009*. Dordrecht : Springer Science, 2010. – ISBN 978-90-481-8763-8

[Valente et al. 2011] VALENTE, Joao ; BARRIENTOS, Antonio ; DEL CERRO, Jaime: Coverage Path Planning to Survey Large Outdoor Areas with Aerial Robots: A Comprehensive Analysis. In: *Introduction to Modern Robotics* 11 (2011). – URL https://www.iconceptpress.com/download/paper/101108074033.pdf

[Weicker 2013] WEICKER, Karsten: *Algorithmen und Datenstrukturen*. Wiesbaden : Springer Vieweg, 2013. – ISBN 978-3-8348-2074-7

[WildPackets] WILDPACKETS: *Illustration of a TCP/IP Encapsulation chart*. – URL https://web.archive.org/web/20120529200700/http://www.wildpackets.com/elements/misc/WP_encapsulation_chart.pdf. – Zugriffsdatum: 5. April 2015

A. Anhang

Tabelle mit Kommandos, die gesendet bzw. empfangen werden können

Kommando	Argument	WP - X	WP -Y	Beschreibung
OTS_CONNECT_OTS	1=an; 0=aus	-	-	schaltet Empfang der OTS-Daten ein
OTS_CONNECT_GS	IP ; Modus	-	-	verbindet sich zur Groundstation mit IP und setzt Master- bzw. Slave Modus
OTS_CONNECT_SLAVE	IP-Adresse	-	-	Master verbindet sich zum Slave mit IP
OTS_START_COMMANDING	1=an; 0=aus	-	-	Startet das Absuchen
OTS_FOLLOW_ME	1=an; 0=aus	-	-	Slave folgt dem Master
OTS_IMMEDIATE	1=an; 0=aus	-	-	nur ein Quadrokopter; fliegt Wegpunkte sofort an
OTS_SEARCHMODE	A oder P	-	-	A = Flächensuche, P = Wegpunktsuche
OTS_WAYPOINT	-	X	Y	Koordinaten eines Wegpunkts
OTS_WP_REACHED	true	X	Y	bestätigt Erreichen des Wegpunkts
OTS_AVR_SAMPLETIME	int	-	-	Setzt die Rate, mit der die Positionsdaten an AVR gesendet werden
OTS_COMM_MESSAGES	string	-	-	Enthält dem debug output der Software

Tabelle A.1.: Befehle zur Kommandierung der Quadrokopter. (WP = Wegpunkt)