Julius-Maximilians-Universität Würzburg

Fakultät für Mathematik und Informatik

Informationstechnik für Luft- und Raumfahrt

Lehrstuhl für Informatik 8 Prof. Dr. Sergio Montenegro

Bachelorarbeit

Entwicklung einer ad-hoc W-Lan Gui-basierten Steuerung von multiplen Quadrokoptern

Vorgelegt von

Joshua Smolka

Matr.-Nr.: 1710819

Prüfer: Prof. Dr. Sergio Montenegro

Betreuender wissenschaftlicher Mitarbeiter: Dipl.-Ing. Nils Gageik

Würzburg, 16. 09. 2013

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, 16. 19. 2013

_____

Joshua Smolka

Entwicklung einer ad-hoc W-LAN GUI-basierten Steuerung von multiplen Quadrokoptern

Joshua Smolka

Die Fortschritte im Bereich Sensorik und Mikrotechnik ermöglichen heutzutage den kostengünstigen Bau kleiner unbemannter Luftfahrzeuge (UAV, unmanned airial vehicle, Drohne) wie Quadrokopter. Die Forschung und Entwicklung dieser Systeme wurde in den letzten Jahren aufgrund der vielfältigen Anwendungsmöglichkeiten stark vorangetrieben. Wenngleich im Bereich UAV viel geforscht wurde, ist das Thema Autonomes Flugobjekt längst noch nicht vollständig behandelt. Insbesondere der Indoor-Betrieb ist aufgrund fehlender absoluter Positionsstützung durch GPS problematisch. Der Aufbau eines eigenen autonomen Systems wird daher am Lehrstuhl Aerospace Information Technology der Uni Würzburg erforscht und erprobt. Im Rahmen dieses Forschungsvorhabens ist ein System inklusive GUI zur Steuerung von Quadrokoptern über ein W-LAN ad-hoc Netzwerk zu entwickeln.

Hauptaugenmerk der Arbeit ist die Realisierung der ad-hoc W-LAN Kommunikation. Daneben ist die GUI zur Anzeige von Zustandsinformationen und Steuerung des Quadrokopters über W-LAN zu erweitern. Es soll eine W-LAN ad-hoc Kommunikation mehrere Quadrokopter möglich gemacht werden. Die Kommunikation soll vollduplexfähig sein und es soll eine gleichzeitige Kommunikation aller Teilnehmer untereinander möglich sein. Später soll ein

Schwarm von Quadrokopters Kommandos und Zustandsinformationen auf diesem Weg austauschen können, um Aufgaben zu verteilen und sich bei der Ausführung zu synchronisieren.

Der Programmcode der GUI soll so strukturiert sein, dass er eine einfache Erweiterung der Software zu mehr Funktionalität ermöglicht. Des Weiteren sind Funktionen zur Steuerung des Quadrokopters hinzuzufügen, wie das Importieren und Exportieren von Wegpunkt- und Kommandolisten. Die erstellte GUI-Software und die Quadrokopter-Firmware sind zu evaluieren. Zur Aufgabe gehört eine ausführliche Dokumentation der Software.

Aufgabenstellung (Stichpunktartig):

- GUI Erweiterung: Wegpunktliste (Import/Export, Alignement), Update alter Funktionen
- Konzept Kommunikation
- Implementierung W-LAN ad-hoc Netzwerk für mehrere Quadrokopter: Vollduplex, gleichzeitig mehrere Verbindungen, Server/Client einstellbar, volle Flexibilität
- Integration in bestehendes System
- Evaluierung
- Dokumentation

Zusammenfassung

Am Lehrstuhl VII (Aerospace Information Technology) wird ein System aus mehreren Quadrokoptern aufgebaut. In dieser Arbeit wird eine Grundlage für die Kommunikation per Wireless Local Area Network (WLAN) gelegt.

Dabei wird ein Client/Server-System entworfen, das über das Tranmission Control Protocol (TCP) eine verbindungsbasierte Datenübertragung ermöglicht. Jeder Teilnehmer des Netzwerks besitzt einen Client, mit dem er vollduplexfähige Verbindungen zu anderen Teilnehmern aufbauen kann, sowie einen Server, der eingehende Verbindungen annimmt. An einen Server können sich dabei mehrere Clients verbinden.

Nach der Implementierung des Entwurfs wird der Programmcode in die bereits bestehende Software zur Quadrokoptersteuerung integriert. Das beinhaltet auch die Erweiterung der Benutzeroberfläche, zu der ein neues Modul hinzugefügt wird, das auf eine bedienungsfreundliche Kommunikation mit vier Teilnehmern ausgelegt.

Dann findet eine Evaluation der Software, getestet mit vier Teilnehmern, statt. Abschließend wird ein Ausblick auf weitere Entwicklungs- und Nutzungsmöglichkeiten gegeben.

Inhaltsverzeichnis

1 Einleitung

Es existiert bereits ein funktionierendes System von Quadrokoptern, die ihre Lage selbstständig halten und zu gewünschten Orten fliegen können. Die bisher anfallende Kommunikation, wie etwa die Übertragung von Borddaten, wurde mit der Bodenstation über eine Bluetoothverbindung realisiert.

Diese Lösung weist jedoch einige Probleme bzw. Einschränkungen auf. Die Datenübertragungsrate sowie die Reichweite der Verbindung genügen den Ansprüchen nicht. Außerdem handelt es sich bei der Bluetoothverbindung um eine Peer-to-Peer-Verbindung, also einer Verbindung zwischen nur zwei Teilnehmern. Es soll nun ein Netzwerk eingerichtet werden, in dem mehrere Quadrokopter oder Bodenstationen teilnehmen können.

Diese Arbeit hat zum Ziel, die Vorteile des Einsatzes eines Wireless Local Area Networks (WLAN) zu evaluieren und anschließend ein Server/Client-System zu implementieren, welches Datenübertragungen via WLAN ermöglicht. Dieses Netzwerk soll im Adhoc-Modus betrieben werden, also ohne feste Infrastruktur im Bedarfsfall flexibel aufgebaut oder beendet werden.
Beim Entwurf des WLANs steht nicht nur die Verbindung eines Quadrokopters zur Bodenstation im Fokus, sondern auch Verbindungen der Quadrokopter untereinander. Dies bietet den Vorteil, dass die Quadrokopter miteinander kommunizieren können, was potentiell von Kollisionsvermeidung bis hin zu Schwarmverhalten in vielen Anwendungsgebieten zum Einsatz kommen könnte.

Im Rahmen dieser Arbeit wird zunächst ein theoretisches Modell zur

Kommunikation entwickelt. Anschließend wird das Konzept implementiert und in die bestehende Steuerungssoftware der Quadrokopter integriert. Dabei wird auf die Entwicklungsumgebung Qt Creator zurückgegriffen, die bereits zur Entwicklung der Steuerungssoftware genutzt wurde.

2 Stand der Wissenschaft

2.1 OSI-Modell

Bei dem „Open System Interconnection Model“ (OSI-Modell) handelt es sich um ein Architekturmodell, das eine Netzwerkarchitektur in sieben verschiedene Schichten zerlegt und dabei jeder einzelnen feste Aufgaben zuteilt. Weiterhin werden die Schnittstellen zu darauf aufbauenden Schichten klar definiert, sodass sich auf der selben Schicht befindliche Netzwerkprotokolle problemlos untereinander austauschen lassen und beliebig kombinierbar sind.

Entwickelt wurde das OSI-Modell von der „International Standardization Organisation“ (ISO).

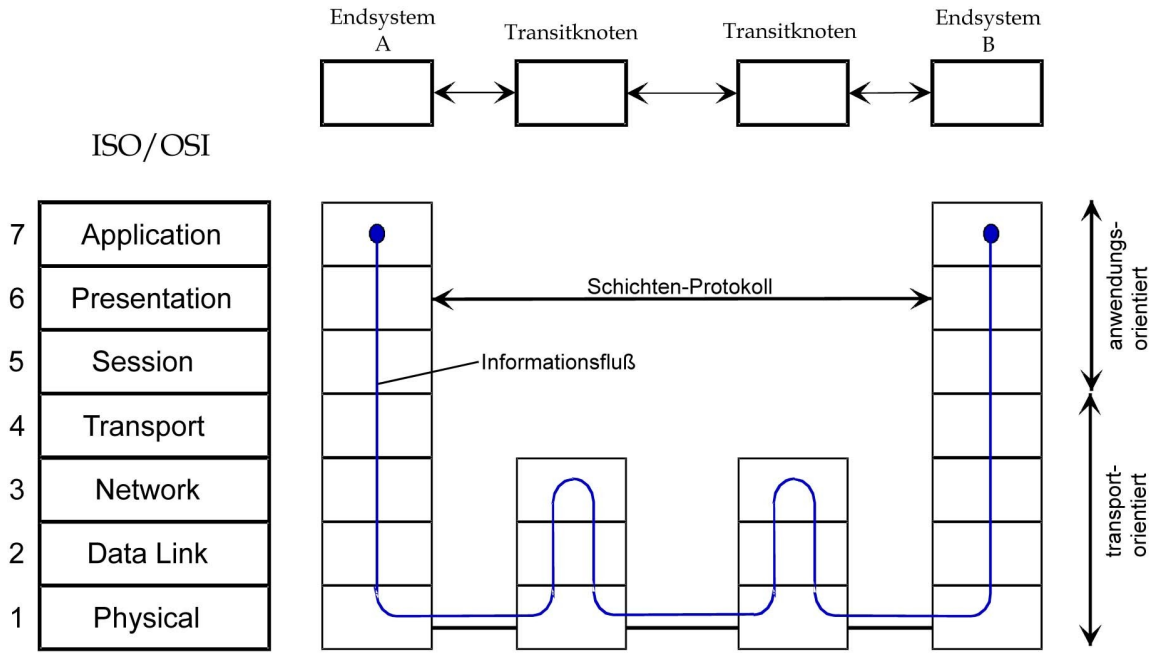

Abbildung 1: OSI-Modell[1]

Im Folgenden werden die einzelnen Schichten des OSI-Modells kurz beschrieben.

Schicht 1 – Physical Layer (Bitübertragungsschicht)

Diese Schicht ist dafür zuständig, physikalische Verbindungen zu aktivieren und deaktivieren. Digitale Daten werden hier zu elektromagnetischen Wellen oder Lichtwellen umgewandelt und verschickt, ankommende Signale wieder digitalisiert.

Schicht 2 – Data Link Layer (Sicherungsschicht)

Der Data Link Layer sorgt für eine möglichst fehlerfreie Übertragung der Daten. Er paketiert zudem den Datenstrom oder fügt ankommende Daten wieder zu einem Datenstrom zusammen. Kanalcodierung, Fehlererkennung sowie Fehlerkorrektur sind weitere Aufgaben des Data Link Layer. Hardwarekomponenten, die auf dieser Schicht arbeiten sind Bridge oder Switch.

Schicht 3 – Network Layer (Vermittlungsschicht)

Bei leitungsorientierten Diensten werden durch den Network Layer Verbindungen geschaltet, bei paketorientierten Diensten werden Datenpakete weitergereicht. Zusätzlich ist der Network Layer für das Routing zuständig, was Aufgaben wie Aufbau und Aktualisierung von Routing-Tabellen beinhaltet.

Das bekannteste Protokoll dieser Schicht ist das Internet Protocol (IP).

Schicht 4 – Transport Layer (Transportschicht)

Hierbei handelt es sich um die unterste Schicht, die eine vollständige Kommunikation zwischen zwei Teilnehmern bereitstellt. Alle folgenden, anwendungsorientierten Schichten brauchen durch einheitlichen Zugriff auf diese Schicht die spezifischen Eigenschaften des Kommunikationsnetzes nicht zu berücksichtigen.

Aufgaben des Transport Layer sind Adressierung, Anpassung von Paktelängen,

Multiplexierung, Überlastungssteuerung, Überprüfung von Dienstqualität.

Beispiele für Protokolle der Transportschicht sind Transmission Control Protocol (TCP) und User Datagram Protocol (UDP).

Schicht 5 – Session Layer (Sitzungsschicht)

Hier werden Sprachmittel zur Eröffnung, Durchführung und Schließung einer Session bereitgestellt. Über diese Schicht werden Sitzungsparameter zwischen den Teilnehmern vereinbart, zudem wird der Dialog hier gesteuert und synchronisiert. Die Teilnahmeberechtigung der Nutzer an der Session wird hier ebenfalls kontrolliert. Auch Checkpoints werden gespeichert, um nach Ausfall einer Sitzung bei Wiederaufnahme nicht erneut von vorne starten zu müssen.

Schicht 6 – Presentation Layer (Darstellungsschicht)

Diese Schicht stellt eine Art Übersetzer zwischen den Teilnehmern dar. Die von der Anwendungsschicht des Systems erhaltenen Daten werden in systemunabhängige Darstellungen umgewandelt. Dadurch können bespielsweise die ASCII-codierten Daten eines Teilnehmers nach der Übertragung bei einem anderen Teilnehmer UNICODE-codiert dargestellt werden.

Andere Aufgaben des Presentation Layer sind Verschlüsselung von Nachrichten sowie Datenkompression.

Schicht 7 – Application Layer (Anwendungsschicht)

Bei der Anwendungsschicht handelt es sich um die Verbindung zwischen der eigentlichen Anwendung sowie den unteren, bereits beschriebenen Schichten.

Hier lassen sich Übermittlungsparameter festlegen, die Verschlüsselung von Nachrichten steuern oder Verfügbarkeit und Identität von Kommunikationspartnern prüfen.

Beispiele für Application Layer sind WWW, E-Mail oder Domain Name System (DNS).[2][3][4]

2.2 Wireless Local Area Network

Ein Local Area Network (LAN) bezeichnet ein Rechennetz mit einer Ausdehnung bis zu 500 Metern. Es wird oft als Heimnetz oder in Unternehmen eingesetzt. Dabei existieren verschiedene Techniken zur Datenübertragung. Am häufigsten wird eine Datenübertragung per Kabel eingesetzt, meist mit der Ethernet-Technologie. Eine weitere Möglichkeit ist die Datenübertragung per Funk, das sogenannte Wireless Local Area Network (WLAN).

Wireless Local Area Network (WLAN) bezeichnet also ein lokales Funknetz, meist mit dem Standard der IEEE 802.11-Familie. Es stellt eine Anpassung an die ersten beiden Schichten des OSI-Modells, dem Physical Layer und dem Data Link Layer, dar.

Im Vergleich zu anderen drahtlosen Übertragungsmöglichkeiten wie etwa Bluetooth bieten WLANs gute Reichweiten bei gleichzeitig hoher Datenübertragungsrate.

2.2.1 Betriebarten

Für Wireless Local Area Networks gibts es zwei verschiedene mögliche Betriebsarten, den Infrastruktur-Modus und den Adhoc-Modus. Diese lassen sich wiederum in untergeordneten Modi klassifizieren.

Infrastruktur-Modus

Beim Infrastruktur-Modus handelt es sich um ein WLAN mit einem zentralen Wireless Access Point oder einem drahtlosen Router. Dieser ist für die

Kommunikation zwischen den Teilnehmern zuständig.

Zum Verbindungsaufbau sendet der Access Point sogenannte „Leuchtfeuer“, die verschiedene Parameter beinhalten. Es werden der Netzwerkname (SSID), eine Liste der unterstützten Übertragungsraten sowie die Art der Verschlüsselung übertragen. Außerdem kann durch diese „Leuchtfeuer“ ständig die Verbindung zum Netzwerk überprüft werden, auch wenn sonst keine Datenübertragung stattfindet. Neue Teilnehmer melden sich mit ihrer MAC-Adresse am Wireless Access Point an und bekommen von diesem eine IP-Adresse zugeteilt.

Ein **Wireless Access Point** (AP) bietet für Endgeräte eine kabellose Schnittstelle zur Kommunikation. Die Teilnehmer stellen mit einem Wireless Adapter eine drahtlose Verbindung zum AP her, der seinerseits per Kabel mit einem weiteren Kommunikationsnetz verbunden sein kann.

Der Wireless Access Point ist in der Schicht 2 (Data Link Layer) des OSI-Modells angesiedelt. Er hat die Aufgabe, Datenkollisionen zu vermeiden und gleichzeitig eventuelle Unterschiede der Übertragungsmedien zu überbrücken.

Für den Wireless Access Point gibt es darüber hinaus verschiedene Betriebsmodi.

Basis Service Set

Beim „*Basic Service Set*“ gibt es einen einzelnen Access Point. Über diesen können sich beliebig viele Teilnehmer zum Netzwerk hinzufügen und untereinander Daten austauschen.

Ethernet Bridge

Im Modus „*Ethernet Bridge*“ ist zusätzlich zur Funkschnittstelle ein Netzwerkinterface als Schnittstelle zum Ethernet vorhanden. Es ist also ein

Bridging zwischen WLAN und dem kabelgebundenen Ethernet möglich, wobei der Wireless Access Point die Daten vermittelt.

Extended Service Set

„*Extended Service Set*“ nennt sich das Verfahren, bei dem für dasselbe Funknetzwerk mehrere Wireless Access Points eingerichtet werden. Diese sind zusätzlich miteinander über Ethernet verbunden. Dies hat zur Folge, dass die Reichweite des WLANs bei verschiedenen Standorten der Access Points deutlich erhöht wird, da die Teilnehmer sich, je nach Standort, mit verschiedenen Access Points verbinden können. Bei Standortwechsel der Teilnehmer werden diese automatisch von einem AP zum Nächsten übergeben (Roaming).

Wireless Distribution System

Im Modus „*Wireless Distibution System*“ werden erneut mehrere Access Points miteinander verbunden. Hierbei wird jedoch kein Ethernet verwendet, sondern WLAN. Dabei gibt es zwei verschiedene Methoden, der Point-to-Point-Modus und der Point-to-Multipoint-Modus. Zu beachten ist hier, dass Wireless Access Points desselben Herstellers verwendet werden sollten.

Adhoc-Modus

Ein Adhoc-Netzwerk zeichnet sich dadurch aus, dass es spontan und ohne großen Aufwand aufgebaut werden kann. Anders als im Infrastruktur-Modus gibt es keine feste Infrastruktur, wie etwa einen Wireless Access Point. Alle Teilnehmer des Netzwerks sind gleichberechtigte Endbenutzer.

Man unterscheidet zwei Arten von Adhoc-Netzen, dem „Independent Basic Service Set“ (IBSS) und einem „Mesh Network“.

IBSS

Beim IBSS kommunizieren die Teilnehmer direkt untereinander, das heißt, die

Datenübertragungen laufen über keine Zwischenstation. Dafür müssen sich jedoch stets alle Teilnehmer in gegenseitiger Funkreichweite befinden, was die Reichweite des Netzes einschränkt.

Mesh Network

Beim Mesh Network stellt jeder Teilnehmer gleichzeitig Endbenutzer und Netzknoten dar. Dies bringt den Vorteil, dass Datenübertragungen zwischen zwei Teilnehmern, die sich nicht in gegenseitiger Funkreichweite befinden, über Dritte realisiert werden kann. [5][6][7][8]

2.2.2 Frequenzstandards

In Tabelle 1 sind die verschiedenen IEEE-Standards im Vergleich aufgelistet.

Protokoll	**Frequenz**	**Brutto-durchsatz**	**Netto-durchsatz**	**Reichweite (im Haus)**	**Reichweite (im Freien)**
802.11b	2,4 GHz	11 Mbit/s	4-5 Mbit/s	38 m	140 m
802.11g	2,4 GHz	54 Mbit/s	19 Mbits/s	38 m	140 m
802.11n	2,4 / 5 Ghz	600 Mbit/s	240 Mbit/s	70 m	250 m

Tabelle 1: Vergleich verschiedener IEEE-Standards[9]

2.3 Transportschicht

2.3.1 Transmission Control Protocol

Das Transmission Control Protocol (TCP) ist ein verbindungsbasiertes Protokoll, das auf dem Transport Layer angesiedelt ist. Es bietet eine verlässliche Datenübertragung zwischen zwei Prozessen, wobei einer der beiden als Server, der andere als Client fungiert. Die Verbindung ist dabei vollduplexfähig, es können also beide Teilnehmer gleichzeitig Daten senden. Zusätzlich bietet das

TCP automatische Fehlererkennung und -behebung. Es gibt außerdem spezielle Algorithmen zur Vermeidung von Überlastungen und Kontrolle des Datenflusses.

Bevor eine Datenübertragung stattfinden kann, wird zunächst eine Verbindung zwischen einem Socket des Senders (Client) und des Empfängers (Server) aufgebaut. Sockets sind vom Betriebssystem bereitgestellte, plattformunabhängige Schnittstellen zwischen der Netzwerkprotokollimplementierung des Betriebssystems und der Anwendungssoftware. Sie sind durch eine IP-Adresse und eine Portnummer eindeutig identifizierbar. Möchte ein Programm mit einem anderen Teilnehmer Daten transferieren, so muss es dafür ein Socket vom Betriebssystem anfordern, welches alle Sockets verwaltet.

Die Verbindung wird beim TCP durch den sogenannten „Three-way Handshake“ durchgeführt. Dabei sendet der Client zunächst eine „Synchronisations-Paket“ an den Server, der an der Zieladresse wartet. Erreicht dieses den Server, so schickt dieser ein „Acknoledgement-Paket“ zurück. Nun wechselt der Client in den Zustand „Connection Established“ und schickt seinerseits ein „Acknoledgement-Paket“. Sobald dieses den Server erreicht, wechselt auch er in den Zustand „Connection Established“.

Steht die Verbindung, so werden die zu übertragenden Datenpakete gesendet. Dabei wird mit sogenannten Acknoledgement-Paketen der Erhalt der Daten bestätigt beziehungsweise durch Ausbleiben der Bestätigung ein erneutes Senden angefordert. Somit wird sichergestellt, dass jedes Datenpaket am Ziel ankommt.

Das Beenden der Verbindung beginnt mit dem Versenden des „FIN-Pakets“ vom

Client an den Server. Dieser bestätigt den Erhalt mit dem „Acknoledgement-Paket“ und wechselt in den Zustand „CLOSE_WAIT“. Anschließend sendet er ein „FIN-Paket“ zurück und wartet auf ein letztes „Acknoledgement-Paket“ vom Client. Ist dieses gesendet, so schließt er die Verbindung endgültig, während der Client typischerweise noch 30 Sekunden wartet, bis er seinen Socket schließt.[10][11]

2.3.2 User Datagram Protocol

Das User Datagram Protocol (UDP) ist im Gegensatz zum TCP ein verbindungsloses Netzwerkprotokoll, das heißt, anstatt eine Verbindung zwischen Sender und Empfänger aufzubauen wird das Datenpaket direkt an den Zielport des Empfängers gesendet. Zusätzliche Funktionalitäten wie das Erkennen und Beheben von Datenverlusten sind nicht implementiert.

Da beim UDP keine Verbindung zwischen den beiden Teilnehmern aufgebaut wird, ist, im Gegensatz zum TCP, keine wechselseitige Kommunikation möglich. Das zu übertragende Datenpaket wird übermittelt, ohne dass der Empfänger ohne weiteres antworten kann. Diese beidseitige Kommunikation ist beim TCP gegeben, beide Teilnehmer können gleichzeitig Nachrichten übertragen.[12]

2.4 Internet Protocol (IP)

Das am weitesten verbreitete Protokoll der dritten Schicht, dem Network Layer, ist das Internet Protocol (IP).

Durch Vergabe einer IP-Adresse können Teilnehmer innerhalb eines Netzwerks eindeutig adressiert werden. Falls Teilnehmer aus zwei verschiedenen Netzwerken miteinander kommunizieren möchten, kann das Problem auftreten,

dass beide in ihrem Netzwerk dieselbe IP-Adresse zugewiesen bekommen haben. Um dieses Problem zu lösen, wird eine Subnetzmaske hinzugefügt, mit der sich das Netzwerk identifizieren lässt.

Zum Datentransfer werden Datagramme benutzt, typischerweise mit der Länge von 1,5 kB. Die Datagramme sind durch einen Header näher spezifiziert. Dieser enthält standardmäßig 5 Wörter (20B), kann jedoch durch verschiedene optionale Angaben auf bis zu 15 Wörter erweitert werden. [13][14]

2.5 Bluetooth

Bluetooth ist ein Standard zur Datenübertragung gemäß IEEE 802.15.1 per Wireless Personal Area Network (WPAN). Es sind sowohl verbindungslose als auch verbindungsbehaftete Übertragungen möglich.

Im Vergleich zu WLANs fällt die Übertragungsrate mit 2,1 Mbit/s eher gering aus. Auch können maximal sieben aktive Verbindungen zu anderen Teilnehmern gleichzeitig gehalten werden. Allerdings lässt sich diese Zahl auf Kosten der Übertragungsrate erhöhen, etwa durch Zeitmultiplexen.

Die Reichweite von Bluetoothverbindungen liegt, je nach Klasse, bei 10 bis 100 Metern. [15]

3 Konzept

Es soll nun ein Kommunikationskonzept entwickelt werden, welches die gestellten Anforderungen erfüllt. Über ein Wireless Local Area Network im Adhoc-Modus sollen mehrere Teilnehmer mit einem Client/Server-System verbunden werden können. Für die Verbindungen selbst ist gefordert, dass sie vollduplexfähig sind. Außerdem werden die Clients und Server dahingehend implementiert, dass problemlos mehrere Verbindungen gleichzeitig offen gehalten werden können. Dies beinhaltet auch die Möglichkeit, gleichzeitig an mehrere Teilnehmern Nachrichten zu verschicken.

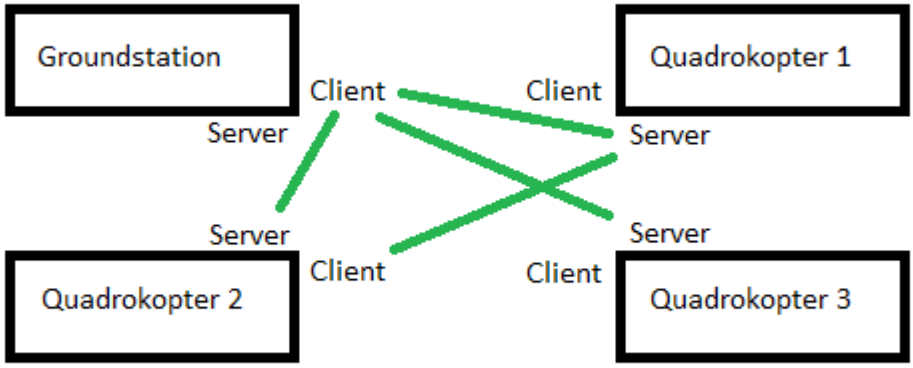

Abbildung 2: Beispielhafte Verbindung der Teilnehmer

In Abbildung 2 ist eine beispielhafte Situation des WLANs mit vier Teilnehmern dargestellt. Jeder Teilnehmer besitzt einen Server sowie einen Client. Die Verbindungen zwischen den Teilnehmern sind durch grüne Linien kenntlich gemacht. Da es sich um ein Adhoc WLAN handelt, gibt es keine feste Infrastruktur wie beispielsweise einen Wireless Access Point, die Verbindungen sind direkt zwischen den beiden Teilnehmern aufgebaut. In diesem Beispiel hält

die Bodenstation (Groundstation) eine Verbindung zu jedem der drei Quadrokopter, außerdem besteht eine Verbindung zwischen Quadrokopter 2 und Quadrokopter 1.

Im Nachfolgenden wird das Konzept zur Netzwerkarchitektur erläutert.

Transportschicht

Als Protokoll der vierten Schicht, dem Transport Layer, kommen das User Datagram Protocol (UDP) und das Transmission Control Protocol (TCP) infrage. UDP und TCP sind zwei sehr gegensätzliche Protokolle. Während UDP sehr minimalistisch gestaltet ist, beispielsweise keine Fehlererkennung und -behebung besitzt, bietet TCP viele nützliche Funktionalitäten.

Da UDP außerdem paketbasiert ist und somit keine Verbindung zwischen Sender und Empfänger hergestellt wird, ist das Protokoll nicht vollduplexfähig, TCP als verbindungsorientiertes Protokoll hingegen schon.

Da für den Flugbetrieb eine verlässliche Kommunikation erforderlich ist, sind Eigenschaften wie Fehlererkennung und -behebung wichtig, was für das TCP spricht.

Aufgrund der vielen Vorteile, die TCP gegenüber UDP besitzt, wird in dieser Arbeit auf das Transmission Control Protocol zurückgegriffen.

Da das Programm selbst auf die TCP-Verbindung zugreift, werden keine Protokolle für Schichten fünf bis sieben verwendet.

Infrastruktur-Modus oder Adhoc-Modus

In der Aufgabenstellung ist zwar gefordert, dass das WLAN im Adhoc-Modus betrieben wird. Das Programm ist allerdings sowohl im Infrastruktur-Modus als auch im Adhoc-Modus funktionsfähig. Für den Infrastruktur-Modus wird ein

Wireless Access Point benötigt, über den das WLAN erstellt werden kann. Das WLAN im Adhoc-Modus kann über die Software „Realtek 11n USB WLAN Utility“, geliefert mit den Funkadaptern, manuell gestartet werden. Eine Anleitung hierfür findet sich im Anhang (Kapitel 8.1). Dazu bietet sich die Bodenstation an, es kann aber auch von einem Quadrokopter gestartet werden. Bei dem WLAN im Adhoc-Modus handelt es sich um ein „Independent Basic Service Set“ (IBSS). Somit müssen alle Teilnehmer in direktem Funkkontakt stehen, damit sie miteinander kommunizieren können. Ein Routing über andere Teilnehmer ist nicht möglich.

Sobald ein Teilnehmer mit dem WLAN verbunden ist und das Programm ausführt, wird der sogenannte MainServer gestartet, der auf einem festgelegten Port auf eingehende Verbindungen wartet. Möchte ein anderer Teilnehmer über seinen Client eine Verbindung aufbauen, so geschieht dies immer auf dem MainServer. Dann wird dem Anfragenden eine einzigartige Portnummer zugeteilt und auf diesem Port ein Server aufgemacht, der nur für die Kommunikation zwischen den beiden Teilnehmern zuständig ist. Anschließend schließt der Client die Verbindung und verbindet sich auf dem neu erstellten Server erneut. Über diese Verbindung läuft anschließend die Kommunikation, bis die Verbindung wieder getrennt wird. Auf dem MainServer können sich weiterhin andere Teilnehmer verbinden.

Durch dieses Verfahren wird sichergestellt, dass jede Verbindung zwischen zwei Teilnehmern auf verschiedenen Server/Client-Paaren stattfindet. Zudem ist die Anzahl der Verbindungen je nach Bedarf dynamisch erweiterbar bzw. nicht mehr benötigte Server und Clients können gelöscht werden. Es hat also jeder Teilnehmer soviele Clients wie ausgehende Verbindungen, außerdem soviele

Server wie eingehende Verbindungen (sowie den MainServer).

4 Implementierung

4.1 Überblick

Im nächsten Schritt wird das Konzept realisiert. Dazu wird zum bereits bestehenden Programmcode ein weiteres Modul hinzugefügt. Dieses beinhaltet die Klassen *CommServer* und *Client*. Es werden außerdem einige Funktionen in der Klasse *MainWindow* implementiert sowie bestehende Funktionen erweitert.

Die Klasse *CommServer* stellt den Server eines Teilnehmers dar. Hier werden ankommende Verbindungsanfragen beantwortet und bestehende Verbindungen verwaltet. Zudem werden ankommende Nachrichten ausgelesen.

Mithilfe der Klasse *Client* lassen sich Verbindungen zu anderen Netzwerkteilnehmern herstellen. Die ausgehenden Verbindungen werden verwaltet und es ist möglich, Nachrichten zu verschicken.

In der Klasse *MainWindow* werden je ein Objekt der Klasse *Client* und *CommServer* angelegt. Hier wird zudem die Benutzeroberfläche programmiert.

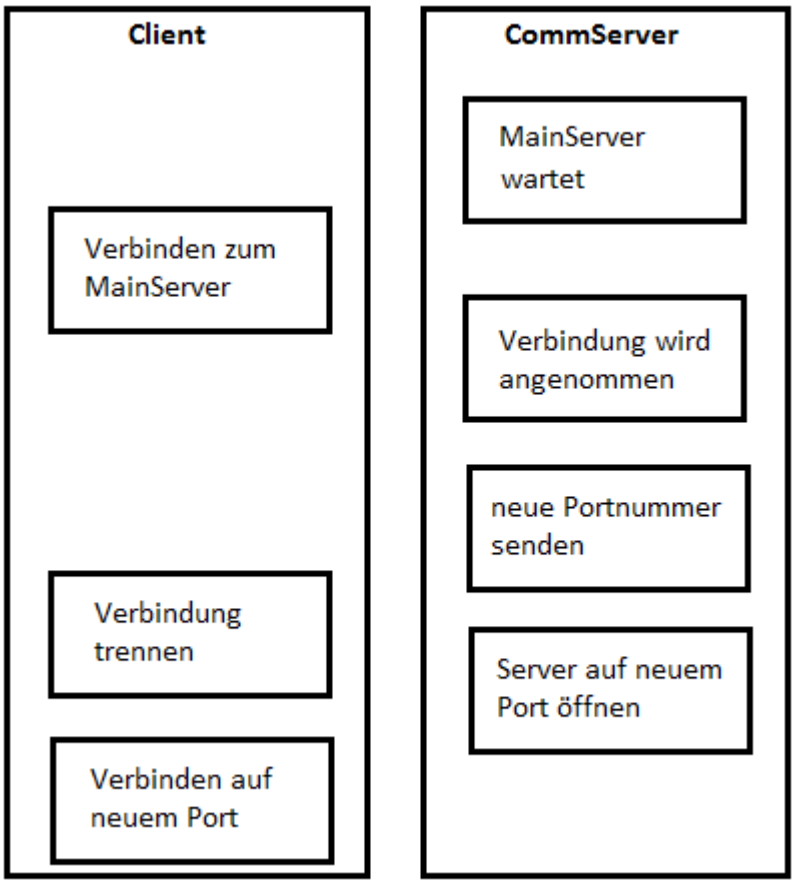

Abbildung 3: Verbindungsaufbau

Zum besseren Verständnis der Interaktion zwischen den Klassen *Client* und *CommServer* während eines Verbindungsaufbaus ist diese in Abbildung 3 schematisch dargestellt. Die genaue Vorgehensweise wird in den nächsten Kapiteln erläutert.

Im Folgenden wird ein Überblick über die einzelnen Klassen und deren wichtigsten Funktionen und Methoden gegeben.

4.2 Client

Mit der Klasse Client lässt sich eine Verbindung zu einem CommServer aufbauen. Sobald diese Verbindung erstellt ist, kann über die Funktion *sendMsg(QString msg, QStringList targets)* eine Nachricht versendet werden. Mit

receiveMsg() werden ankommende Nachrichten weiterverarbeitet. Andere implementierte Funktionen schließen die Verbindung oder eine Verbindung auf einen anderen Port wechseln.

4.2.1 *start(QString ip, QString port)*

Diese Methode bekommt als Argumente eine IP-Adresse sowie eine Portnummer übergeben. Zunächst wird geprüft, ob die IP-Adresse nicht leer ist, dann, ob bereits eine Verbindung zum Teilnehmer besteht. Falls einer der beiden Fälle eintritt, so wird eine Fehlermeldung ausgegeben und die Methode beendet.

Andernfalls wird ein QTcpSocket erstellt und versucht, mit diesem über die vorimplementierte Methode *connectToHost()* eine Verbindung zur gewünschten Adresse und dem ausgewählten Port zu etablieren. Ist dieser Versuch erfolgreich, so wird der QTcpSocket in einer Liste abgespeichert und sein Signal *readyRead()* mit dem Slot *receiveMsg()* verbunden. Sonst wird eine Fehlermeldung ausgegeben und der QTcpSocket gelöscht.

4.2.2 *receiveMsg()*

Diese Methode liest ankommende Nachrichten aus. Sie wird auf das Signal *readyRead()* hin aktiv, welches ausgesandt wird sobald Daten zum Auslesen verfügbar sind. Die Liste mit allen aktiven QTcpServern wird nach dem mit auszulesenen Daten durchsucht. Dann werden die Daten mithilfe der *read()*-Funktion in einem char-Array zwischengespeichert. Nach der Konvertierung zu einem QString wird das Signal *msgRec()* ausgesandt, das nach außen hin den Erhalt einer neuen Nachricht signalisiert.

Anschließend wird die Nachricht auf mögliche enthaltene Befehle überprüft. Bisher implementiert sind die Befehle *!RECONNECT* und *!DISCONNECT*. Vorgesehen, aber noch nicht ausgearbeitet sind die Befehle *!SUCHEN*, *!LANDEN* und *!POSITION_HALTEN*. Falls der Befehl *!RECONNECT* gesendet wurde wird die Methode *reconnect()* aufgerufen, im Falle von *!DISCONNECT* die Methode *disc()*. Der Befehl *!RECONNECT* wird außerdem mit einer angehängten Portnummer gesendet, auf die sich der Client verbinden soll (beispielsweise *!RECONNECT 7778*). Wurde einer der anderen Befehle erkannt, so wird das Signal *command(QString)* ausgesendet, wobei als Argument der Befehl eingeüfgt wird.

4.2.3 *sendMsg(QString msg, QStringList targets)*

Der Client kann nicht nur Nachrichten erhalten, sondern auch selbst verschicken. Da der Client mehrere Verbindungen gleichzeitig halten kann, ist *sendMsg()* auf Versenden an mehrere Adressaten ausgelegt. Als Argumente werden der Methode ein QString *msg* mit der zu versendenden Nachricht sowie eine QStringList *targets* mit allen Adressaten übergeben. Dann wird für jeden Empfänger nach dem korrespondierenden QTcpSocket gesucht und die Nachricht mit *write()* gesendet. Falls kein Socket existiert wird eine Fehlermeldung ausgegeben.

4.2.4 *reconnect(QString port, QString address)*

reconnect() bekommt eine Portnummer sowie eine IP-Adresse übergeben. Falls eine ungültige Portnummer übergeben wurde, wird eine Fehlermeldung ausgegeben und die Methode beendet. Andernfalls wird der korrespondierende QTcpSocket aus der Liste herausgesucht und mit diesem zunächst die bereits

bestehende Verbindung mittels *disconnectFromHost()* getrennt. Anschließend wird über *connectToHost()* eine Verbindung auf dem neuen Port geöffnet. Zusätzlich wird das Signal *newConnection()* ausgesendet.

4.2.5 *disc(QString address)*

Mithilfe dieser Methode lässt sich eine offene Verbindung schließen. Dazu wird ein QString übergeben, der eine IP-Adresse beinhaltet. Dann wird der entsprechende QTcpSocket aus der Liste gesucht. Existiert kein solcher, so wird die Methode mit einer Fehlermeldung abgebrochen. Sonst wird mittels der *write()*-Methode die Nachricht „!SHUTTING_DOWN“ an den Server geschickt. Nach einer kurzen Wartezeit, die dem Server das Lesen der Nachricht ermöglicht, schließt der Client die Verbindung und löscht den QTcpSocket. Als letztes wird eine Meldung über das erfolgreiche Schließen der Verbindung ausgegeben.

4.2.6 Weitere Funktionen

Außer den oben genannten gibt es die Funktionen *getMsg()* und *hasConnection(QString address)*. *GetMsg()* gibt die letzte erhaltene Nachricht als QString zurück. *hasConnection(QString address)* wird eine durch einen QString repräsentierte IP-Adresse übergeben und eine boolean-Variable zurückgegeben, der angibt, ob eine Verbindung zu dieser Adresse bereits besteht.

4.3 CommServer

Die Klasse CommServer stellt Schnittstellen für eingehende

Kommunikationsversuche eines Clients bereit. Dazu wird der *MainServer* eingerichtet, der auf einem vordefinierten Port auf eingehende Verbindungsanfragen wartet. Für jede Verbindung wird bei eingehender Anfrage ein eigener Server gestartet, auf dem die Verbindung dauerhaft eingerichtet wird.

Außerdem besitzt diese Klasse Funktionen zum Empfangen und Senden von Nachrichten. Dabei ist zu beachten, dass die Funktion *answer(QString msg, QString address)* nur für automatisierte Nachrichten zu benutzen ist. Manuelle Befehle werden über *Client* versendet.

4.3.1 Konstruktor

Im Konstruktor wird ein sogenannter *MainServer*, ein Objekt der Klasse QTcpServer, angelegt. Dieser wartet unter allen verfügbaren IP-Adressen auf einem festgelegten Port (standardmäßig 7777) und steht zum Etablieren von Verbindungen zur Verfügung. Außerdem wird hier der Port für die erste dauerhafte Verbindung festgelegt (standardmäßig 7778).

4.3.2 *acceptConnection()*

Diese Methode wird aufgerufen, sobald sich ein Client zum MainServer verbunden hat. Der Socket wird über die bereitgestellte Methode *nextPendingConnection()* der Klasse QTcpServer aufgerufen und in einer temporären Variable abgespeichert. Danach wird auf einem anderen Port ein neuer QTcpServer erstellt, auf dem sich der Client wiederverbinden soll. Sein Signal *newConnection()* wird mit der Methode *acceptReconnect()* verbunden und er wird in einer Liste gespeichert.

Sobald der QTcpServer eingerichtet ist, wird über die noch bestehende Verbindung des *MainServer* zum Client der Befehl „!RECONNECT“ mit angefügter Portnummer geschrieben. Daraufhin beendet der Client die Verbindung zum *MainServer* und verbindet sich neu auf der angegebenen Portnummer. Dies ist die Verbindung, die für den Rest der Session beibehalten wird.

4.3.3 *acceptReconnect()*

Sobald sich der Client mit dem neu angelegten Server verbindet wird die Methode *acceptReconnect()* aufgerufen. Nun steht die endgültige Verbindung zwischen Client und *CommServer*.

Um Nachrichten entgegennehmen zu können, wird das Signal *readyRead()* des QTcpSockets mit dem Slot *startRead()* verbunden. Anschließend wird eine Meldung zum erfolgreichen Verbinden ausgegeben.

4.3.4 *startRead()*

Mit *startRead()* können eingehende Nachrichten entgegengenommen werden. Diese Methode funktioniert ähnlich wie die bereits vorgestellte Methode *readMsg()* der Klasse Client: Zunächst wird der Socket mit auszulesenen Daten aus der Liste gesucht, dann mittels *read()* die Bytes in ein char-Array ausgelesen. Falls die Nachricht „!SHUTTING_DOWN“ ist, bedeutet dies, dass der Client im Begriff ist die Verbindung zu schließen. In diesem Fall wird der betreffende Socket aus der Liste mit den aktiven Verbindungen entfernt und die Methode beendet.

Auch die Befehle „Suchen“, „Landen“ und „Position halten“ werden erkannt und

bei Erhalt ein entsprechendes Signal ausgesendet.

4.3.5 *answer(QString msg, QString address)*

Ein CommServer kann nicht nur Empfänger, sonder auch Sender von Nachrichten sein. Dazu wird dieser Methode die zu übertragende Nachricht und die IP-Adresse des Adressaten übergeben. Nachdem der entsprechende Socket gefunden ist, wird mittels *write()* die Nachricht übertragen.

4.4 MainWindow

In Abbildung 4 ist die Benutzeroberfläche zu sehen. Der rot eingerahmte Bereich ist für die Kommunikation per WLAN hinzugefügt worden.

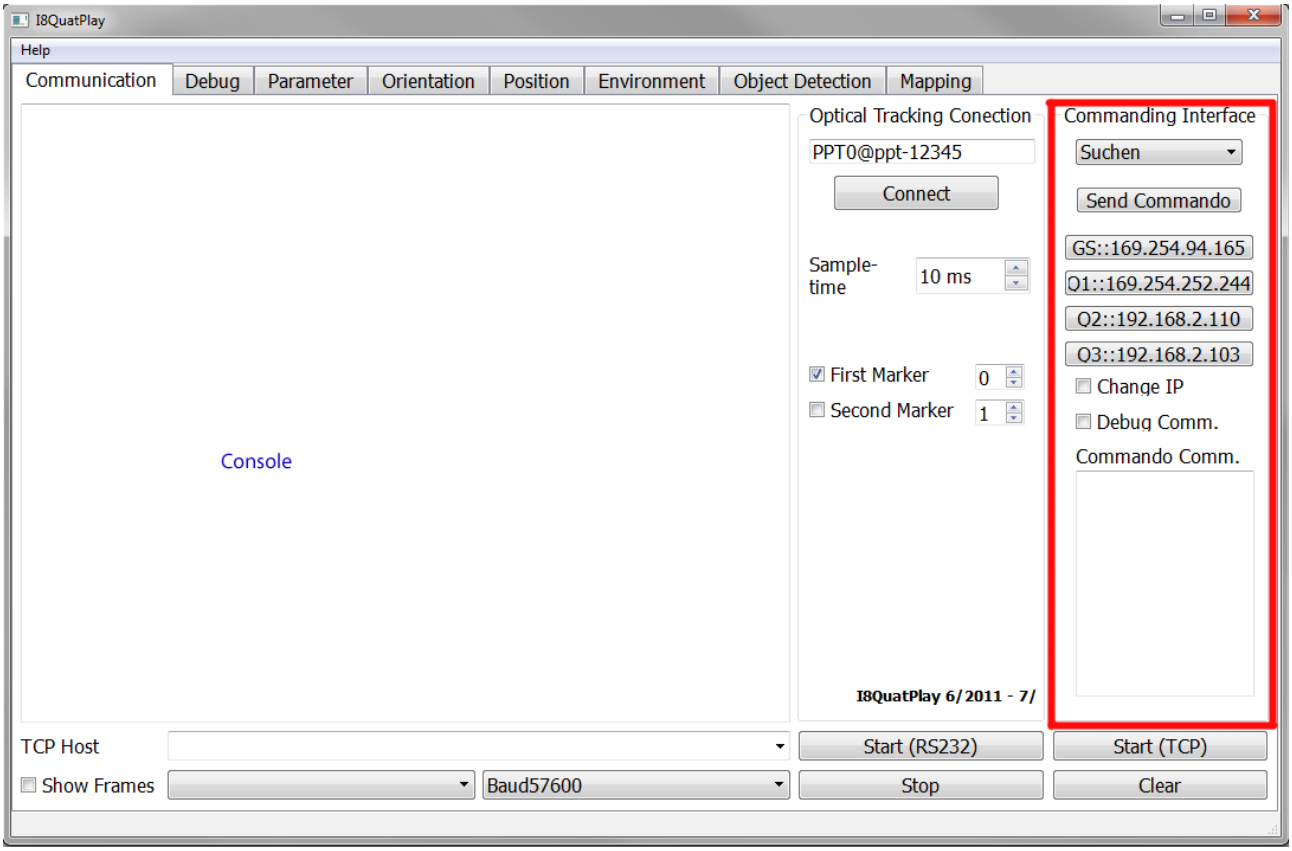

Abbildung 4: Benutzeroberfläche

Sämtliche Funktionalitäten der Benutzeroberfläche sind im der Klasse

MainWindow implementiert. Auf die für diese Arbeit relevanten und neu hinzugefügten Funktionen wird in diesem Kapitel näher eingegangen. Im Nächsten wird die Benutzeroberfläche an sich vorgestellt.

4.4.1 Initialisierungen

Im Konstruktor des MainWindows werden verschiedene initialisierende Schritte durchgeführt, damit die TCP-Kommunikation möglich ist.

Mittels aufruf der Methode *initIPs()* werden die IP-Adressen standardmäßig konfiguriert. Dafür existieren vier verschiedene defines, INIT_IP_GS, INIT_IP_Q1, INIT_IP_Q2 und INIT_IP_Q3. Dort sind die standardmäßigen IP-Adressen der vier Teilnehmer gespeichert. In *initIPs()* werden diese Werte nun in die Variablen überschrieben. Anschließend wird jede IP-Adresse mit der eigenen IP-Adresse verglichen. Sollte es eine Übereinstimmung geben, so wird der zugehörige Button disabled, also nicht anwählbar gemacht.

4.4.2 *on_pushButton_sendCommando_clicked()*

Dieser Slot wird ausgeführt, wenn der Button mit der Beschriftung „Send Commando“ geklickt wurde. Zunächst werden alle Teilnehmer, zu denen eine Verbindung besteht, einer Liste angefügt. Dann wird der aktuell ausgewählte Eintrag der ComboBox „comboBox_commando“ ausgelesen und in den entsprechenden Befehl umgewandelt. Ist der Eintrag beispielsweise „Suchen“, lautet der Befehl „!SUCHEN“. Dieser wird dann zusammen mit der Liste der Adressaten an den Client im Funktionsaufruf *sendMsg(QString msg, QStringList targets)* übergeben. Zuletzt wird eine Meldung über das Verschicken in der Benutzeroberfläche ausgegeben.

4.4.3 *on_checkBox_changeIP_clicked()*

Sobald die CheckBox „Change IP“ angewählt wurde, werden alle Teilnehmer-Buttons enabled. Würde dies nicht durchgeführt werden, entstünde das Problem, dass Buttons mit der eigenen IP nicht geändert werden können, da diese nicht angeklickt werden können.

Wird die Checkbox erneut angeklickt, so werden die Buttons überprüft, ob ihre neue IP-Adresse mit der des eigenen Computer übereinstimmt. Ist dies der Fall, dann wird der Button disabled.

4.4.4 *on_pushButton_connectGroundstation_clicked()*

Stellvertretend für die vier verschiedenen Teilnehmer-Buttons wird hier die aufgerufene Methode beim Anklicken des „pushButton_connectGroundstation“ vorgestellt. Die anderen Methoden funktionieren analog.

Zunächst wird überprüft, ob die Checkbox „Change IP“ angewählt ist. Ist dies der Fall, so wird ein QInputDialog erstellt. Dieser besitzt ein Textfeld, in dem die aktuelle IP-Adresse geändert werden kann. Im Falle des Bestätigens der Änderung wird die neue IP-Adresse abgespeichert. Mittels *updateIPs()* wird außerdem die Beschriftung der Buttons aktualisiert.

Falls die Checkbox nicht angewählt wurde, so wird versucht, eine neue Verbindung zur zugehörigen IP-Adresse aufzubauen. Existiert eine solche bereits, wird die Verbindung geschlossen.

4.4.5 Weitere Funktionen

Bei Erhalt eines Befehls wird von der Klasse *CommServer* ein Signal ausgesandt. Dieses wird vom MainWindow abgefangen und die Funktion

on_command_received(QString command) aufgerufen. Dort werden die bei Erhalt des Befehls durchzuführenden Schritte implementiert.

4.5 Beschreibung der Benutzeroberfläche

Zur bereits bestehenden Benutzeroberfläche wurde eine Einheit namens Commanding Interface hinzugefügt. Sie ist in Abbildung 5 zu sehen.

Es sind fünf QPushButtons, zwei QCheckBoxen sowie eine QComboBox und ein QTextArea im Commanding Interface enthalten. Diese sind in Abbildung 5 am rechten Rand nummeriert, worauf sich im Folgenden bezogen wird.

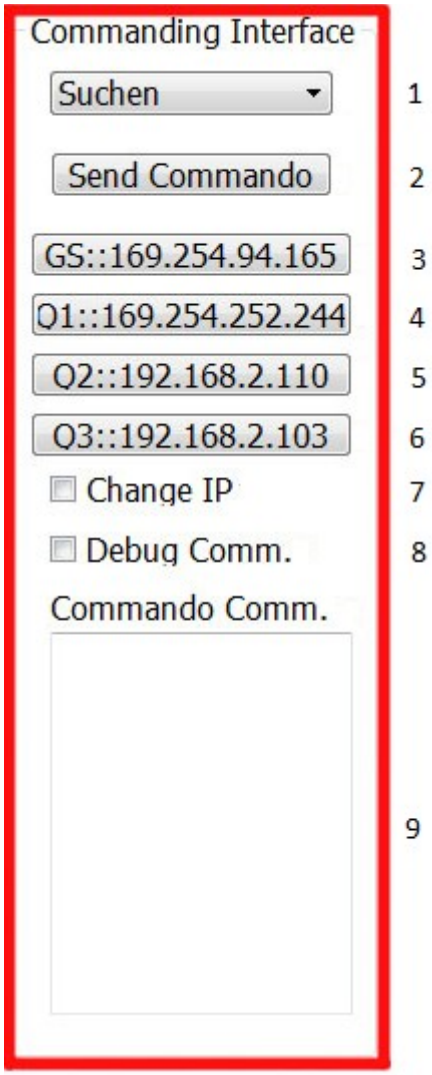

Abbildung 5: Commanding Interface

Vier der Buttons (Nummer 3 bis 6) repräsentieren je einen Teilnehmer des Netzwerks. Sie sind durch einen Namen sowie eine IP-Adresse, die als Beschriftung des Buttons eingetragen sind, spezifiziert. GS steht hierbei für Groundstation, Q1 für Quadrokopter 1, usw. Sollte einer der Buttons den eigenen Computer repräsentieren, so wird der entsprechende Button disabled. Er ist nun grau unterlegt und nicht mehr anzuwählen. Damit wird sichergestellt, dass keine Verbindung zu sich selbst aufgebaut wird.

Mit einem Klick auf einen dieser Buttons wird versucht, eine Verbindung zu diesem Teilnehmer aufzubauen. Ein erneuter Klick auf diesen Button schließt die Verbindung wieder. Die Buttons der Teilnehmer, zu denen von diesem

Computer aus eine Verbindung aufgebaut wurde, sind grün gefärbt. Wenn die Verbindung beendet wird, wird der Button wieder entfärbt. Falls eine Verbindung besteht, die jedoch vom anderen Teilnehmer initiiert wurde, so wird diese nicht optisch hervorgehoben.

Unter den Buttons befindet sich eine CheckBox mit der Beschriftung „Change IP“ (Nummer 7). Ist diese angewählt, so lassen sich die IP-Adressen der Buttons mittels eines Klicks auf selbigen ändern. Daraufhin wird ein QInputDialog aufgerufen, in der sich ein Eingabefeld mit der bisherigen IP befindet. Ist diese geändert worden, so lässt sich die Eingabe mit dem OK-Button bestätigen. Mit Klick auf diesen Button wird das Fenster geschlossen und die IP-Adresse des Teilnehmers aktualisiert. Falls doch keine Änderung gewünscht wird, steht der Cancel-Button zur Verfügung.

Um anschließend wieder per Buttonklick eine Verbindung aufzubauen zu können, muss der Haken der CheckBox „Change IP“ entfernt werden.

Oberhalb der gerade beschriebenen Buttons befindet sich eine Combobox mit den Auswahlmöglichkeiten „Suchen“, „Landen“ und „Position halten“ (Nummer 1). Diese repräsentieren verschiedene Befehle, die an andere Teilnehmer gesendet werden können. Darunter liegt ein weiterer Button mit der Aufschrift „Send Commando“ (Nummer 2). Drückt man diesen, so wird der aktuell in der ComboBox ausgewählte Befehl an alle Teilnehmer, deren Buttons grün eingefärbten sind, gesendet.

Abschließend gibt es eine Checkbox mit der Beschriftung „Debug Comm“ (Nummer 8) sowie ein Textfeld (Nummer 9). In dem Textfeld werden standardmäßig die Meldungen des Clients sowie des Servers ausgegeben. Falls jedoch ausführliche Meldungen über Verbindungen gewünscht sind, lassen sich

mit dem Aktivieren der Checkbox detaillierte Meldungen in der Console (siehe Abb. 4, blau beschriftet) ausgegeben. Zudem wird in diesem Modus kenntlich gemacht, ob es sich um eine Meldung des Clients oder des Servers handelt.

4.6 verwendete Hardware

Das Konzept wurde in Qt realisiert. Qt ist eine plattformübergreifende C++-Bibliothek, die vorrangig zur Programmierung graphischer Oberfächen entwickelt wurde. Als Entwicklungsumgebung wurde der Qt Creator benutzt.
Die Hardware ist der WLAN USB 2.0 Nano Adapter WL0084A des Herstellers LogiLink. Dieser beherrscht die gängigen WLAN-Standards IEEE 802.11b, IEEE 802.11g und IEEE 802.11n. Zur Verschlüsselung stehen die Standards WEP, WPA sowie WPA2 zur Verfügung. Die Datenübertragungsrate liegt bei bis zu 150 Mbps. Unterstützte Betriebssysteme sind Windows XP/Vista/7, Linux 2.6.x und MAC OS [16]. Der Adapter wird über einen USB 2.0 Anschluss mit dem Quadrokopter verbunden. Mit der mitgelieferten Software REALTEK 11n USB Wireless LAN Utility lässt sich zudem auf einfache Weise ein Ad-hoc-Netzwerk aufbauen (siehe Anhang, Kapitel 8.1).

5 Evaluierung

Zu Evaluationszwecken wurde ein Test der Software mit vier Teilnehmern durchgeführt. Nachfolgend wird zunächst der Versuchsaufbau beschrieben und anschließend die Ergebnisse dargestellt.

Als Teilnehmer werden vier Computer verwendet. Da drei von ihnen keine Hardwarekomponenten für WLAN-Kommunikation besitzen, werden externe WLAN-Adapter über USB angeschlossen.

Mittels der Software „Realtek 11n USB WLAN Utility“ wird ein Adhoc WLAN im IBSS-Modus erstellt. Eine Anleitung dazu findet sich im Anhang (Kapitel 8.1).

Steht das WLAN, so werden die Computer manuell mit dem Netzwerk verbunden. Nun wird die Software gestartet. Falls eine Firewall vewendet wird, so sollte darauf geachtet werden, dem Programm Kommunikationsrechte durch die Firewall zu erlauben. Andernfalls wird das Programm blockiert und ist nicht funktionsfähig.

Mit den vier Computern werden nun verschiedene Funktionalitäten getestet. Die Checkbox „Debug Comm“ wird aktiviert, damit auf der Console ausführliche Meldungen ausgegeben werden. Der Verbindungsaufbau funktioniert wie im Konzept beschrieben. Nach Klicken auf einen der vier Teilnehmerbuttons wird in der Console die Meldung „C: Command RECONNECT received from xxx. Reconnecting at Port yyy.“ ausgegeben. Ist die Verbindung dann endgültig etabliert, erscheint eine entsprechende Meldung sowohl in der Console als auch in der Console des Teilnehmers, zu dem die Verbindung aufgebaut wurde.

Auch das Versenden von Kommandos funktioniert problemlos. Mit Klick auf den

Button „Send Commando“ wird der ausgewählte Befehl an jeden Computer, zu dem der Client eine Verbindung hält, gesendet. Dies wird auf den jeweiligen Computern mit einer Ausgabe in der Console bestätigt.

Ein Problem tritt jedoch beim Senden von Kommandos auf: Wenn ein Befehl zwei mal zu schnell hintereinander gesendet wird, kann die erste Nachricht nicht vom Server ausgelesen werden, bevor die zweite Nachricht ankommt. Somit kommen beim Server anstatt zwei Befehlen nur eine Nachricht an, beispielsweise „!SUCHEN!SUCHEN“ anstatt zwei mal dem Befehl „!SUCHEN“. Außerdem kann diese Nachricht nicht mehr als Befehl erkannt werden.

Verbindungen zu mehreren anderen Computern beeinträchtigen die Funktionalität des Programmes dagegen nicht. Auch doppelte Verbindungen zwischen zwei Computern, die entstehen sobald sich bei mit ihrem Client auf den Server des Anderen verbinden, führen zu keinerlei Problemen.

Durch Aktivieren der Checkbox „Debug Comm“ kann zudem zwischen normalen Meldungen und ausführlichen Meldungen gewechselt werden.

6 Diskussion und Ausblick

Mit Abschluss dieser Arbeit ist eine Gruppe aus mehreren Computern beziehungsweise Quadrokoptern in der Lage, miteinander zu kommunizieren. Dabei sind die Teilnehmer in der Lage, die Befehle „Suchen“, „Position halten“, „Landen“, „Disconnect“ und „Reconnect“ zu erkennen, wobei nur für die letzten beiden Befehle die gewünschten Funktionalitäten implementiert wurden.

Damit ist die bisher verwendete Lösung für Datenübertragungen, eine Bluetoothverbindung zwischen Quadrokopter und Bodenstation, deutlich verbessert worden. Das WLAN bietet eine verlässliche Kommunikationsalternative, die darüber hinaus eine höhere Reichweite besitzt. Zudem ist die Datenübertragungsrate erhöht worden, außerdem sind nun Möglichkeiten zur Kommunikation zwischen den Quadrokoptern gegeben.

Allerdings wurden nicht alle angedachten Aufgaben umgesetzt. Eine praktische Funktionalität wäre beispielsweise ein automatisches Verbinden der Teilnehmer zum bereits bestehenden WLAN beim Starten der Software. Weiterhin sollten neue Funktionen zur Benutzeroberfläche hinzugefügt werden, etwa das Speichern und Laden von Wegpunktlisten. Dies wurde aufgrund von Zeitmangels nicht mehr implementiert.

Somit sind weitere Arbeiten erforderlich, damit die WLAN-Kommunikation sinnvoll zum Einsatz kommen kann. Neben den bereits erwähnten Aufgaben sind auch andere Anwendungsgebiete denkbar.

Zunächst kann die Übertragung der Telemetriedaten von Bluetooth auf WLAN umgestellt werden.

Weiterhin bieten sich zusätzliche Positionierungsbefehle an. So könnten beispielsweise Wegpunktlisten verschickt werden, die der Quadrokopter anschließend selbstständig abfliegt. Das bietet gegenüber einzelnen Positionsbefehlen den Vorteil, dass der Quadrokopter auch dann noch neue Punkte anfliegt, wenn die Verbindung unterbrochen oder beendet wurde.

Die Vernetzung der Quadrokopter untereinander bietet noch einmal viele neue Möglichkeiten. So wäre ein Austausch der Positionsdaten denkbar, auf dessen Basis ein Mindestabstand zwischen zwei benachbarten Quadrokoptern eingehalten wird. Zudem könnten die Quadrokopter selbstständig Formationsflüge absolvieren.

Bei Aufgaben wie dem Mapping von unbekannten Gebäuden ist die Kommunikation zwischen mehreren Quadrokoptern besonders interessant: Hier können die Quadrokopter untereinander Aufgaben aufteilen. Befinden sich beispielsweise zwei Quadrokopter an einer Abzweigung, so können beide Wege zwischen den Quadrokoptern aufgeteilt und unbekannte Gebäude so deutlich effizienter erforscht werden.

Somit ist diese Arbeit mit einem positiven Ergebnis abgeschlossen worden. Auch wenn nicht alle Ziele erfüllt wurden, so ist mit dieser Arbeit ein Grundstein gelegt worden, auf den nachfolgende Projekte mit unterschiedlichen Inhalten aufbauen können.

7 Literaturverzeichnis

[1]: Prof. P. Tran-Gia: Vorlesungsskript „Rechnernetze und Kommunikationssysteme“, Kapitel 5.2, Seite 4, Würzburg, 2011.

[2]: http://de.wikipedia.org/wiki/OSI-Modell, 07.09.2013.

[3]: Matthias Kleine: http://www.selflinux.de/selflinux/html/osi.html, 07.09.2013.

[4]: Prof. P. Tran-Gia: Vorlesungsskript „Rechnernetze und Kommunikationssysteme“, Kapitel 5.2, Würzburg, 2011.

[5]: http://de.wikipedia.org/wiki/Wireless_Local_Area_Network, 07.09.2013

[6]: http://de.wikipedia.org/wiki/Wireless_Access_Point, 07.09.2013

[7]: http://de.wikipedia.org/wiki/Ad-hoc-Netz, 07.09.2013

[8]: Prof. P. Tran-Gia: Vorlesungsskript „Rechnernetze und Kommunikationssysteme“, Kapitel 2.3, Würzburg, 2011.

[9]: http://de.wikipedia.org/wiki/IEEE_802.11n#Vergleich_der_IEEE-Standards, 07.09.2013

[10]: http://de.wikipedia.org/wiki/Transmission_Control_Protocol, 07.09.2013

[11]: Prof. P. Tran-Gia: Vorlesungsskript „Rechnernetze und Kommunikationssysteme“, Kapitel 6.3, Würzburg, 2011.

[12]: http://de.wikipedia.org/wiki/User_Datagram_Protocol, 07.09.2013

[13]: http://de.wikipedia.org/wiki/Internet_Protocol, 07.09.2013

[14]: Prof. P. Tran-Gia: Vorlesungsskript „Rechnernetze und Kommunikationssysteme“, Kapitel 6.2, Würzburg, 2011.

[15]: http://de.wikipedia.org/wiki/Bluetooth, 07.09.2013

[16]: LogiLink: Datenblatt für Wireless 15N0Mbps USB Adapter

8 Anhang

8.1 Einrichten eines WLANs im Adhoc-Modus

Zum Erstellen eines Wireless Local Area Networks im Adhoc-Modus wird auf die Software „Realtek 11n USB WLAN Utility“ zurückgegriffen. Damit kann auf einfache Weise ein Adhoc WLAN im IBSS-Modus eingerichtet werden.
Falls bereits ein Profil für das WLAN besteht, so ist dieses im Reiter „Profile“ aufgelistet. Mit einem Doppelklick wird es aktiviert und das WLAN aufgebaut. Sofern noch kein Profil vorhanden ist, so muss ein solches erstellt werden. Dazu wird im Reiter „Profile“ auf den Button „Hinzufügen“ geklickt. Anschließend erscheint ein Fenster, in dem man verschiedene Eigenschaften auswählen kann. Durch Anwählen der obersten Checkbox wird das WLAN im Adhoc-Modus ausgeführt, in den Editfeldern darunter wird der Name des Profils sowie die SSID des WLANs eingetragen. In weiteren Auswahlfeldern können der Kanal, Sicherheitstyp sowie Verschlüsselungstyp des WLAN eingestellt werden. Abschließend muss ein Sicherheitsschlüssel eingegeben werden. Dieser muss von Teilnehmern, die dem Netzwerk beitreten möchten, eingegeben werden.

8.2 Hinzufügen eines neuen Befehls

Die Liste der Befehle ist beliebig erweiterbar. Dazu muss der entsprechende Eintrag in der ComboBox der Benutzeroberfläche hinzugefügt werden. In der Methode *on_pushButton_SendCommando_clicked()* des MainWindows wird dann der Befehl definiert, der an den Quadrokopter gesendet wird. Standardmäßig

beginnt er mit einem Ausrufezeichen und wird in Großbuchstaben geschrieben (Bsp: „!SUCHEN“). Nun muss in der Methode *startRead()* ein entsprechendes Signal geworfen werden, sobald der Befehl ankommt. Dieses wird anschließend im MainWindow mit einem Slot verbunden, der den eigentlich auszuführenden Programmcode enthält.