Diplomarbeit

Neurocopter

Eine fliegende Experimentierplattform zur Erforschung der Hirnaktivität von Honigbienen

Tobias Ludwig

2. Mai 2016

Freie Universität Berlin
Fachbereich Mathematik und Informatik
Institut für Informatik
Arbeitsgruppe Intelligente Systeme und Robotik
Biorobotics Lab

Betreuer
Prof. Dr. Tim Landgraf

Gutachter
Prof. Dr. Raúl Rojas
Prof. Dr. Tim Landgraf

Eidesstattliche Erklärung

Ich versichere hiermit an Eides Statt, dass diese Arbeit von niemand anderem als meiner Person verfasst worden ist. Alle verwendeten Hilfsmittel wie Berichte, Bücher, Internetseiten oder ähnliches sind im Literaturverzeichnis angegeben. Zitate aus fremden Arbeiten sind als solche kenntlich gemacht. Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungskommission vorgelegt und auch nicht veröffentlicht.

Berlin, den 2. Mai 2016

(Tobias Ludwig)

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Zusammenfassung

Die Honigbiene besitzt bemerkenswerte Gedächtnisleistungen und Navigationsfähigkeiten, was sie zum idealen Modellorganismus der Hirnforschung macht. Aufgrund ihrer geringen Größe ist es jedoch bislang nicht möglich, sie im freien Flug zu untersuchen. Um sowohl die Hirnaktivität elektrophysiologisch zu analysieren als auch das Verhalten künstlicher neuronaler Netze zu untersuchen, wurde in dieser Arbeit mit dem *Neurocopter* eine fliegende Experimentierplattform geschaffen.

Das entwickelte System besteht aus einem Quadcopter, der über unterschiedliche Sensoren, eine Kamera und einen leistungsstarken Bordcomputer verfügt. Ein auf dem Bordrechner laufendes Framework ermöglicht den einfachen Zugriff auf die Sensordaten und Kamerabilder, so dass hiermit Experimente dokumentiert und kontrolliert werden können. Über eine Funkverbindung können die Daten zu einer Bodenstation übertragen und dort visualisiert werden. Zur Planung der Versuche lassen sich anhand von GPS-Koordinaten Routen festlegen, die der Copter autonom abfliegt.

1. Einleitung

Die Europäische Honigbiene (*Apis mellifera*) eignet sich hervorragend als Modellsystem der Hirnforschung. Als Insekt ist sie einfach in Haltung und Handhabung. Trotz ihres sehr kleinen Gehirns mit einem Volumen von nur 1 mm^3 mit etwa 960 000 Neuronen [1] weist sie Fähigkeiten auf, die sonst nur bei höher entwickelten Lebewesen wie Wirbeltieren zu erwarten sind [2]. So besitzt sie beispielsweise ein ausgeprägtes Sozialverhalten [3], woraus sich besondere Lern- und Gedächtnisleistungen ergeben [1]. Beim Ausschwärmen zur Nahrungssuche entfernt sich eine Biene über 10 km von ihrem Heimatstock [4]. Nach dem Auffinden einer neuen Tracht (Futterquelle) fliegt sie dann auf direktem Weg zurück [5] und teilt den anderen Arbeiterinnen Richtung und Entfernung der Tracht durch einen Schwänzeltanz im Inneren des Stocks mit [4]. Hierbei läuft die Tänzerin auf der Wabe den Hinterleib rhythmisch hin und her bewegend ein Stück geradeaus, kehrt in einem Halbkreis zum Ausgangspunkt zurück, schwänzelt dann erneut die gerade Strecke, führt nun den Halbkreis in die entgegengesetzte Richtung aus und fährt so immer wieder im Wechsel fort [4]. Die Himmelsrichtung zur Tracht wird dabei durch den Winkel der Schwänzelstrecke relativ zur Schwerkraft und die Entfernung durch das Tanztempo angegeben [4]. Für dieses Verhalten sind gute Navigationsfähigkeiten und ein Gedächtnis über die Futterquellen nötig [6]. Um die Luftlinie zur Tracht im Tanz angeben zu können, summiert die Biene die bei der Suche geflogenen und durch optischen Fluss gemessenen [7] Teilstrecken per Pfadintegration [4] auf und erhält so den direkten Vektor zum Ziel.

Außerdem sind Bienen in der Lage, den direkten Weg zwischen zwei bekannten Orten durch unbekanntes Terrain zu fliegen, was sich nicht mehr allein durch das Modell der Pfadintegration erklären lässt [8]. Eine mögliche Interpretation dieses Phänomens ist eine kognitive Karte [8, 9], in der Orte und Landmarken räumlich so zueinander in Relation gesetzt werden, dass sie zur Selbstlokalisierung genutzt werden können [10]. Ein solches Kartengedächtnis wird dem nur bei höher entwickelten Lebewesen vorhandenen Hippocampus zugeschrieben [10, 11] und ist erst etwa bei Ratten [10, 11] und Mäusen [12] anzutreffen, weswegen diese These noch umstritten ist [13, 14]. Daher sind diese für Insekten einzigartigen Fähigkeiten der Honigbienen umso beeindruckender und geben der Forschung noch viele Fragen auf.

1.1. Motivation: Die Honigbiene in der Neurobiologie

In der Neurobiologie lassen sich die Einsatzmöglichkeiten der Honigbiene grundlegend in zwei Bereiche teilen: Zum einen werden in der Elektrophysiologie Hirnaktivitäten analysiert. Im Gegensatz dazu verfolgt die Neuroinformatik einen synthetischen Ansatz, der auf künstlichen neuronalen Netzen basiert.

Die Elektrophysiologie stößt schnell an physikalische Grenzen, wenn die Hirnströme im Flug untersucht werden sollen. Dies kann erforderlich sein, da ein alleiniges Beobachten des Verhaltens und daraus gezogene Rückschlüsse auf die tatsächlichen Vorgänge im Gehirn einen erheblichen Interpretationsspielraum lassen. Eine Biene kann etwa 50 mg und ohne größere Einschränkungen sogar nur 20 mg über längere Zeit tragen [15]. Folglich scheidet ein Befestigen der nötigen Sensorik und Aufzeichnungselektronik am Insekt selbst aus. Als mögliche Lösung kann der freie Flug im Labor in einer virtuellen Arena aus LCD-Monitoren simuliert werden [16]. Ein solcher Aufbau ermöglicht zwar eine hervorragende Reproduzierbarkeit des Experiments, jedoch weicht er erheblich von der Realität ab und erzeugt nur einen Bruchteil der tatsächlich auftretenden Reize. So hat etwa ein zusätzliches Anströmen des Insekts mit Luft zur Vortäuschung von Flugwind Einfluss auf sein Verhalten [17]. Ein möglicher Schritt in Richtung der idealen Lösung wäre ein *fliegendes Labor*, in dem die Biene allen Umwelteinflüssen ausgesetzt ist.

Genauso muss bei einer Computersimulation der Welt zur Analyse neuronaler Netze berücksichtigt werden, dass sich diese immer von der Realität unterscheidet. Eine biorobotische fliegende Plattform, die eine Biene sensorisch und motorisch imitiert, kann strengere Kriterien als eine reine Simulation erfüllen und somit ergänzend zur Überprüfung von Hypothesen eingesetzt werden.

1.2. Zielsetzung: Eine fliegende Experimentierplattform

Wie in Abschnitt 1.1 erläutert, gibt es in der Neurobiologie mehrere Einsatzmöglichkeiten für ein *fliegendes Labor* zur Erforschung von Honigbienen. Dieser Abschnitt befasst sich mit den Anforderungen an eine solche Plattform, die hier für Hardware und Software aufgestellt werden.

1.2.1. Hardware

Um Bienen oder das Verhalten neuronaler Netze während des Flugs zu untersuchen, sollten die Manövrierfähigkeit und das Flugverhalten der Insekten möglichst naturgetreu wiedergeben werden können. Neben schnellen Richtungs- und Geschwindigkeitswechseln können Bienen auch punktgenau landen und auf einer Stelle schwe-

ben [4, 18]. Diese Anforderungen können von Flugzeugen nicht erfüllt werden, wodurch als Fluggerät nur noch Helikopter und Multicopter zur Wahl stehen. Erstere sind technisch sehr komplex: Um etwa den Hubschrauber entlang der Quer- oder Längsachse zu neigen, muss der Einstellwinkel der einzelnen Rotorblätter zyklisch während eines Umlaufs des Hauptrotors variiert werden. Dies wird mechanisch durch ein über eine Taumelscheibe angesteuertes Gestänge gelöst [19]. Dem gegenüber zeichnet sich ein Multicopter durch den kompletten Verzicht auf mechanische Ansteuerungen aus. Ein solches Fluggerät besitzt mehrere auf einer Ebene angeordnete Propeller mit nach unten wirkendem Schub. Gesteuert wird es allein durch Drehzahländerungen der Motoren [20]. Dieser einfache und kostengünstige Aufbau reduziert den Wartungsaufwand und erhöht die Zuverlässigkeit [20, 21]. Somit soll das *fliegende Labor* auf Basis eines Multicopters konstruiert werden.

Um Experimente verfolgen und dokumentieren zu können oder Reize für ein neuronales Netz zu erzeugen, werden etliche Sensoren benötigt. Neben einem GPS-Modul zur Positionsbestimmung sowie Lage- und Beschleunigungssensoren, soll der Copter auch über eine Kamera verfügen. Prinzipbedingt neigt sich ein Multicopter beim Fliegen in die entsprechende Richtung. Daher kann es nötig sein, dass eine Kamera oder fixierte Biene gedreht werden muss, um immer im gleichen Winkel zum Boden gehalten zu werden. Dies kann von einem Gimbal, einer elektronischen kardanischen Aufhängung, bewerkstelligt werden. Zur Auswertung der Kamerabilder und anderer versuchsspezifischer Daten bedarf es eines leistungsfähigen Bordrechners. Dieser muss über ein Funkmodul verfügen, damit während eines Experiments etwa zusätzliche Steuersignale gesendet oder Statusinformationen übertragen werden können.

1.2.2. Software

Wie in Abschnitt 1.2.1 auf Seite 4 beschrieben, sollte der Multicopter über einen leistungsstarken Bordcomputer verfügen, der zur Auswertung und Steuerung der Experimente dient. Diese müssen reproduzierbar sein, damit sich Ergebnisse verifizieren lassen. Das lässt sich erreichen, indem der Copter die Möglichkeit bietet, vorgegebene Routen autonom anhand von GPS-Koordinaten abzufliegen und an festgelegten Punkten oder bei anderen Ereignissen immer die gleiche Aktion auszuführen. Dabei muss es trotzdem immer möglich sein, im Zweifelsfall händisch einzugreifen und die Steuerung sofort zu übernehmen.

Zur Bereitstellung dieser Funktionalität wird ein Framework (Rahmenstruktur) benötigt, das einen einfachen Zugriff auf die Sensordaten des Copters ermöglicht. Zudem muss die Kommunikation über die Funkverbindung integriert sein, so dass auch auf diese Daten zugegriffen werden kann.

Als Gegenstück wird eine Bodenstation-Software benötigt, mit der die empfangenen Daten aufbereitet und visualisiert werden können, um die laufenden Versuche beobachten zu können. Zur Konfiguration der abzufliegenden Routen wird ebenfalls

entsprechende Software benötigt, mit der es beispielsweise möglich ist, die Punkte anhand einer virtuellen Karte vorzugeben.

Um die Auswahl weiterer für Versuche benötigter Programme nicht vorab einzuschränken, müssen Bodenstation und Konfigurationssoftware des Copters weitgehend plattformunabhängig sein, das heißt unter Microsoft Windows, Mac OS X und Linux nutzbar sein. Als weitere wichtige Anforderung kommt die Quelloffenheit sämtlicher verwendeter Software, insbesondere der Firmware des Flugcontrollers, hinzu. Kommerzielle zivile Multicopter werden häufig zum Kunstflug oder der Anfertigung von Luftaufnahmen verwendet. Der geplante Einsatzzweck als *fliegendes Labor* weicht davon teils erheblich ab. Deswegen ist damit zu rechnen, dass langfristig Änderungen an Kernkomponenten der Flugsteuerung nötig werden, um bestimmte Versuche zu ermöglichen. So etwas ist ohne erheblichen Aufwand nur möglich, wenn der Quelltext vollumfänglich zur Verfügung steht.

2. Stand der Technik

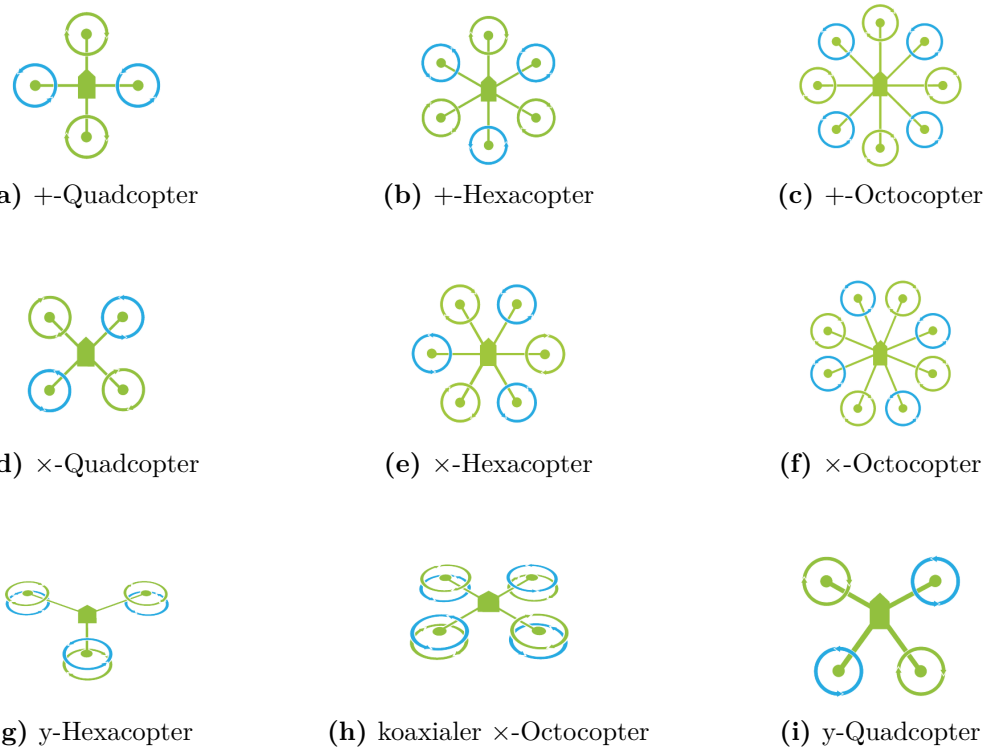
Die fliegende Experimentierplattform *Neurocopter* soll als Multicopter konstruiert werden. Dieses Kapitel verschafft einen Überblick der zum Zeitpunkt des Entwurfs verfügbaren Komponenten und erläutert die allgemeine Funktionsweise und den Aufbau eines Multicopters.

Wie bereits in Abschnitt 1.2.1 auf Seite 4 erwähnt, wird ein Multicopter ohne zusätzliche bewegliche Teile allein durch Drehzahländerungen seiner Motoren gesteuert. Um die erzeugten Drehmomente dieser auszugleichen und eine ständige Rotation des Copters um die Hochachse zu verhindern, wird eine gerade Anzahl Motoren verwendet, von denen sich eine Hälfte mit und die andere gegen den Uhrzeigersinn dreht. Typische Konfigurationen verwenden vier, sechs oder acht Motoren, die an Auslegern rund um eine zentrale Plattform auf unterschiedlichste Weise angeordnet sind, wie in Abbildung 2.1 auf der nächsten Seite veranschaulicht wird. Es existieren auch Lösungen, die bis zu zwölf Motoren zulassen [22]. Diese Vielfalt dient der Erfüllung unterschiedlicher Anforderungen, wie beispielsweise eines möglichst großen Sichtfelds für eine Kamera oder zur Erzeugung von Redundanz für den Fall eines Motorversagens.

Der Flugcontroller nimmt vom Piloten per Fernsteuerung gesendete Befehle entgegen. Dabei wird der Ist-Zustand der Lage des Copters im Raum mit Hilfe verschiedenster Sensoren bestimmt und mit dem übermittelten Soll-Zustand verglichen. Daraus werden dann entsprechende Drehzahlen für die einzelnen Propeller berechnet. Aus einem Lithium-Polymer-Akku gespeiste Motorregler erzeugen daraus dann die Ströme zur Ansteuerung der Motoren, die die Propeller in der Regel direkt antreiben. Der allgemeine Aufbau dieser elektronischen Komponenten wird in Abbildung 2.2 auf Seite 9 schematisch dargestellt.

2.1. Manuelle Steuerung

Zur manuellen Steuerung können herkömmliche Funkfernsteuerungen aus dem Modellbau verwendet werden. In der Regel verfügen diese über zwei Steuerknüppel, die jeweils horizontal und vertikal bewegt werden können und so insgesamt vier Funktionen kontrollieren. Zusätzlich gibt es meist noch mehrere Schalter und Drehregler für weitere Funktionen. Die Übertragung an den Empfänger findet auf dem 2,4 GHz-Band statt und verwendet für jede Funktion einen Kanal. Abbildung 2.3 auf Seite 10 zeigt exemplarisch eine Fernbedienung und den dazugehörigen Empfänger.



(a) +-Quadcopter **(b)** +-Hexacopter **(c)** +-Octocopter

(d) ×-Quadcopter **(e)** ×-Hexacopter **(f)** ×-Octocopter

(g) y-Hexacopter **(h)** koaxialer ×-Octocopter **(i)** y-Quadcopter

Abbildung 2.1.: Multicopter-Konfigurationen [23]

Allen Konfigurationen gemein ist die gerade Anzahl von Rotoren, von denen sich eine Hälfte mit dem Uhrzeigersinn (grün) und die andere dagegen (blau) dreht. Die Spitze an der Plattform deutet die Vorderseite des Copters an. Abbildungen 2.1a bis 2.1c zeigen Copter in +-Konfiguration, die sich durch Arme entlang der Längsachse auszeichnen. Die Bezeichnung rührt vom Quadcopter, der mit seinen vier Armen an ein + erinnert. Analog dazu zeigen die Abbildungen 2.1d bis 2.1f ×-Konfigurationen. Die Abbildungen 2.1g und 2.1h zeigen Copter, bei denen sich an jedem Arm zwei koaxial angeordnete Propeller befinden. Von all diesen symmetrischen Aufbauten unterscheidet sich der in Abbildung 2.1i dargestellte Copter, bei dem die Plattform nach vorn versetzt ist. Dies ermöglicht ein ungestörteres Sichtfeld für eine nach vorn gerichtete Kamera.

2.2. Flugcontroller

Das Flugverhalten von Multicoptern ist inhärent instabil [20,25]. Daher wird zur Lageregelung ein Flugcontroller benötigt. Dieser soll außerdem das autonome Abfliegen von Routen übernehmen.

Gängige Flugcontroller verfügen über eine Vielzahl von Sensoren, die zur Lageregelung und Steuerung des Copters verwendet werden.

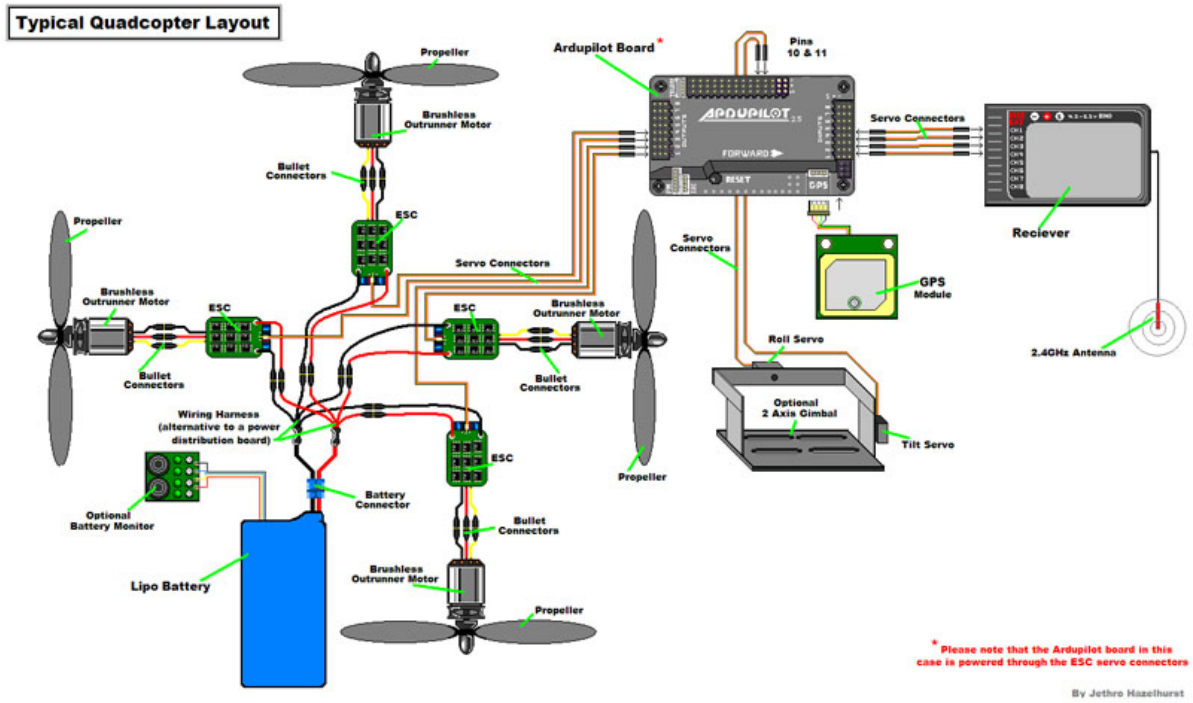


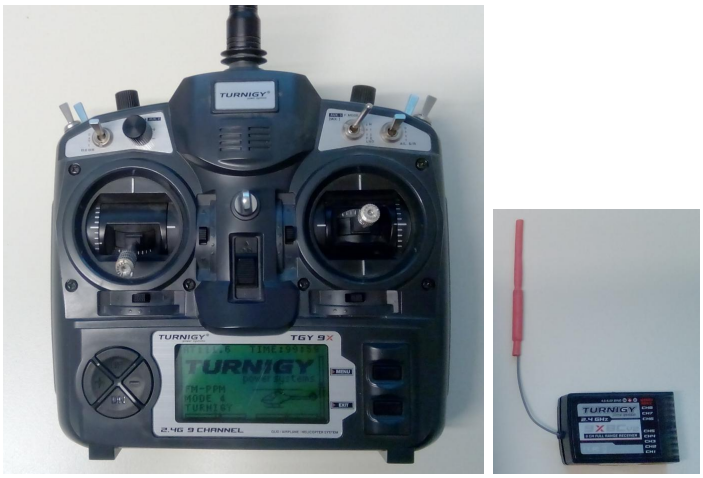
Abbildung 2.2.: Schematischer Aufbau eines Multicopters [24]

Herzstück des Quadcopters ist der Flugcontroller (*Ardupilot Board*). Steuersignale der Fernsteuerung werden vom Empfänger (*Receiver*) an den Flugcontroller geleitet und dort unter Zuhilfenahme mehrerer interner (nicht gezeigt: Gyroskop, Accelerometer, Magnetometer, Barometer) und externer (GPS-Modul) Sensoren in Motordrehzahlen umgerechnet. Daraus erzeugen die Motorregler (*ESC*) die Ströme zum Antrieb der bürstenlosen Gleichstrommotoren (*Brushless Outrunner Motor*), die direkt mit den Propellern verbunden sind. Die gesamte Stromversorgung wird von einem Lithium-Polymer-Akku (*LiPo Battery*) geleistet. Zusätzlich kann der Flugcontroller mit einem Gimbal (*Optional 2 Axis Gimbal*) eine Kamera unabhängig von der Lage des Copters ausrichten.

2.2.1. Sensoren

Gyroskop

Die in Multicoptern verwendeten Gyroskope messen die Winkelgeschwindigkeiten um die drei Achsen des kartesischen Koordinatensystems. Diese Werte werden vom Flugcontroller verwendet, um die Lage des Copters zu stabilisieren [26]. Dabei wird einem plötzlich auftretenden Kippen durch entsprechende Drehzahländerungen der Motoren entgegengewirkt [27]. Die absolute Lage des Copters im Raum kann mit einem Gyroskop nicht bestimmt werden [20]. Die durch Aufintegration der Messwerte erhaltene Schätzung weicht wegen Messungenauigkeiten, der diskretisierten Zeitschritte und nicht zuletzt der ungenauen Repräsentation numerischer Werte im Mikroprozessor mit der Zeit immer mehr vom tatsächlichen Wert ab [28].



(a) Fernbedienung **(b)** Empfänger

Abbildung 2.3.: Funkfernbedienung und Empfänger
Turnigy 9X Fernbedienung (Abb. 2.3a): Mit den beiden Steuerknüppeln lässt sich das Flugmodell kontrollieren. Bei einer der vielen möglichen Konfigurationen steuert der rechte Knüppel das Modell in der Ebene, indem er es seitwärts *rollt* oder vor und zurück *nickt*. Der linke Hebel kontrolliert über den Schub die Auf- und Abbewegungen des Modells und dreht es um die Hochachse, wenn er seitwärts bewegt wird. Einer der Kippschalter lässt sich so konfigurieren, dass er zwischen verschiedenen Modi eines Flugcontrollers wechseln kann.
Turnigy 9X8C-V2 Empfänger (Abb. 2.3b): Der Empfänger besitzt für jede der acht steuerbaren Funktionen einen Anschluss, der entweder einen Servo oder Motorregler steuert, oder an einen Flugcontroller angeschlossen werden kann.

Accelerometer

Zur Beschleunigungsmessung entlang der drei Achsen des Koordinatensystems werden Accelerometer verwendet. Mit diesen Messwerten lassen sich Nick- und Rollwinkel bestimmen, um den Copter waagerecht zu halten [26] und den Drift des Gyroskops zu kompensieren [27]. Bei der Positionsbestimmung durch Aufintegration driftet das Ergebnis – genauso wie bei der Lagebestimmung mit einem Gyroskop – mit der Zeit immer weiter ab.

Magnetometer

Die Lage des Copters im Raum lässt sich durch ein Magnetometer, einen 3-Achsen-Kompass, bestimmen. Die Ausrichtung muss bekannt sein, wenn der Copter GPS-Koordinaten anfliegen soll [26]. Neben der Stärke des Erdmagnetfelds misst das Magnetometer allerdings auch die Felder, die etwa durch die Ströme zur Steuerung der Motoren erzeugt werden. Diese Störungen müssen gegebenenfalls kompensiert werden. Abbildung 2.4 auf der nächsten Seite zeigt ein Magnetometer, dass getrennt vom Flugcontroller ausgeführt ist und so entfernt von Störquellen angebracht werden kann.

Barometer

Ein Anhaltspunkt über die Höhe des Copters lässt sich mit einem Barometer aus dem Luftdruck über die barometrische Höhenformel bestimmen [26]. Dieser Sensor kann leicht durch Windböen oder witterungsbedingte Luftdruckschwankungen gestört werden, weswegen die Messwerte zumindest gefiltert oder mit denen anderer Sensoren kombiniert werden sollten.

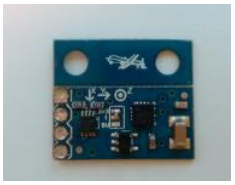


Abbildung 2.4.: Magnetometer
Dieses externe 3-Achsen-Magnetometer wird mit dem Flugcontroller verbunden und kann entfernt von Störquellen angebracht werden.

Sonar

Ein Sonar sendet Ultraschallsignale aus, die zurückgeworfen werden, wenn sie auf ein Hindernis treffen. Aus der Laufzeit lässt sich die Entfernung bestimmen. Multicopter können dieses System zur Kollisionsvermeidung und Höhenbestimmung nutzen [21]. Abbildung 2.5 zeigt ein typisches in Multicoptern verwendetes Sonar.

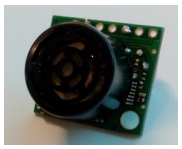


Abbildung 2.5.: Sonar
Ein Sonar bestimmt durch Aussenden von Ultraschallwellen die Entfernung zu Objekten und kann so zur Kollisionsvermeidung und dem Halten der Flughöhe verwendet werden.

GPS

Mit einem GPS-Sensor wird über ein Netz aus Navigationssatelliten die Position auf der Erde bestimmt, die sich aus Breitengrad, Längengrad und Höhe zusammensetzt. Außerdem können noch die Uhrzeit mikrosekundengenau, Geschwindigkeit und Bewegungsrichtung ermittelt werden [29]. Abbildung 2.6 zeigt eine Platine bestückt mit GPS-Modul und Antenne.

(a) Oberseite **(b)** Unterseite

Abbildung 2.6.: GPS-Modul
Platine mit Keramikantenne (Abb. 2.6a) und u-blox GPS-Modul (Abb. 2.6b)

Optischer-Fluss-Sensor

Bei der Bestimmung des optischen Flusses werden Verschiebungsvektoren von wiedererkannten Regionen in aufeinander folgenden Bildern berechnet. Bei einer nach unten gerichteten Linse kann aus diesen Informationen zusammen mit der genauen Entfernung und Orientierung des Sensors zum Boden die Geschwindigkeit des Copters bestimmt werden. Diese Technik findet beispielsweise in optischen Computermäusen Anwendung. Deren Sensoren lassen sich mit einer anderen Linse in

Multicoptern nutzen. Abbildung 2.7 zeigt einen solchen mausbasierten und einen aufwändigeren Sensor.

(a) mausbasierter Sensor ADNS3080 [30] **(b)** PX4Flow [31]

Abbildung 2.7.: Sensoren zur Bestimmung des optischen Flusses
Zur Geschwindigkeitsbestimmung berechnet ein Optischer-Fluss-Sensor Verschiebungsvektoren von wiedererkannten Regionen in aufeinander folgenden Bildern. Einfache Sensoren, wie der in Abbildung 2.7a, basieren auf der Technik von optischen Computermäusen. Abbildung 2.7b zeigt ein komplexeres Modell, das ein Gyroskop und Sonar zur Lage- und Abstandsbestimmung besitzt [31].

2.2.2. Gegenüberstellung gängiger Flugcontroller

Die Vielzahl erhältlicher Flugcontroller unterscheidet sich je nach angedachtem Einsatzzweck. So verfügen ausschließlich fürs manuelle Fliegen konzipierte Modelle nur über die nötigsten Sensoren zur Stabilisierung des Copters und bieten keine Erweiterungsmöglichkeiten. Demgegenüber besitzen aufwendigere autonom fliegende Modelle deutlich mehr Sensoren und Erweiterungsmöglichkeiten. Auch die Konfiguration des Controllers reicht vom integrierten Display mit einigen Tastern bis hin zur umfangreichen Software, die verschiedene Einstellungen speichern und laden kann. Hier werden nun einige Flugcontroller anhand der in Abschnitt 1.2 auf Seite 4 gestellten Anforderungen gegenübergestellt. Hardwareseitig werden hauptsächlich die verfügbaren Sensoren verglichen. Um deren Werte während des Flugs vom geforderten Bordcomputer auswerten lassen zu können, muss der Flugcontroller über eine entsprechende Schnittstelle verfügen. Diese kann direkt als serielle Schnittstelle (UART, *universal asynchronous receiver/transmitter*) oder über einen integrierten USB-Konverter ausgeführt sein. Außerdem wird ein Augenmerk auf die verwendbaren Motorsteuerungen gelegt. Einige Flugcontroller setzen spezielle hauseigene Regler voraus und schließen so die Verwendung besser verfügbarer, deutlich kostengünstigerer Modellbau-ESCs (siehe Abschnitt 2.3.3 auf Seite 17) aus. Zudem wird untersucht, ob es Anschlussmöglichkeiten für ein Gimbal gibt, um etwa eine Kamera lageunabhängig vom Copter bewegen zu können. Softwareseitig wird zum einen die Firmware des Controllers betrachtet. Hierbei werden Funktionen wie das Abfliegen von Routen, die Ansteuerung eines Gimbals und die Anordnungsmöglichkeiten der Motoren, wie sie bereits in Abbildung 2.1 auf Seite 8 gezeigt wurden, untersucht. Zum anderen wird die Verfügbarkeit quelloffener plattformunabhängiger Software

zur allgemeinen Konfiguration des Copters und zum Programmieren von Routen verglichen. Tabelle 2.1 zeigt die Ergebnisse.

Flugcontroller		Eigenschaften																	
		Software								Hardware									
		Firmware					Boden					Sensoren							
Firmware	Hardware	quelloffen	Programmiersprache	Routen abfliegen	Motoranordnung[a]	Gimbal steuern	plattformunabhängig	quelloffen	Route programmieren	Datenschnittstelle	Modellbau-ESCs	Gyroskop	Accelerometer	Magnetometer	Barometer	GPS	Sonar	optischer Fluss
ArduCopter [32]	APM 2.5[b] [33]	✓	C++	✓	✓	✓ [34]	✓ [35]	✓	✓	✓	✓	✓	✓	✓	✓	●	●	●[c] [30]
	Pixhawk [36]									✓	✓	✓	✓	✓	✓	●	●	●
	PX4FMU [37,38]									✓	✓	✓	✓	✓	✓	●	●	●
PX4 [39,40]	Pixhawk [36]	✓	C, C++	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	●	●	●
	PX4FMU [37,38]									✓	✓	✓	✓	✓	✓	●	●	●
MultiWii	MWC Crius SE [41,42]	✓ [43]	C++	✓ [44, 45]	✓ [46]	✓ [47]	✓ [48]	✓ [48]	-[d]	✓ [50]	✓	✓	✓	✓	✓	●	●	●[e]
Mikrokopter	FlightCtrl V2.5 [52–54]	✓ [55]	C++	✓	✓ [56]	✓ [57]	- [58]	- [55]	✓ [59]	✓ [60]	-[f]	✓	✓	✓	✓	●	-	-
Steveis KK2.1 [62]	Hobbyking KK2.1.5 [63,64]	✓	ASM	-	✓	✓	-[g]			-	✓	✓	✓	-	-	-	-	-

✓ ja / vorhanden ● optional

\- nein / nicht vorhanden ● optional, eingeschränkt unterstützt

[a] einfache und koaxiale Anordnung von 4, 6 oder 8 Motoren in + oder × Konfiguration (vgl. Abbildung 2.1 auf Seite 8)

[b] wird nur von Firmwareversionen kleiner v3.3 unterstützt

[c] unterstützt nur den mausbasierten Sensor ADNS3080 und nicht den PX4FLOW [32]

[d] Konfiguration von Wegpunkten nur per Drittanbietersoftware möglich [49]

[e] Unterstützung nur in modifizierter Firmware von Drittanbietern [49,51]

[f] es können nur per I^2C angesteuerte Motorsteuerungen verwendet werden [61]

[g] keine Konfigurationssoftware vorhanden

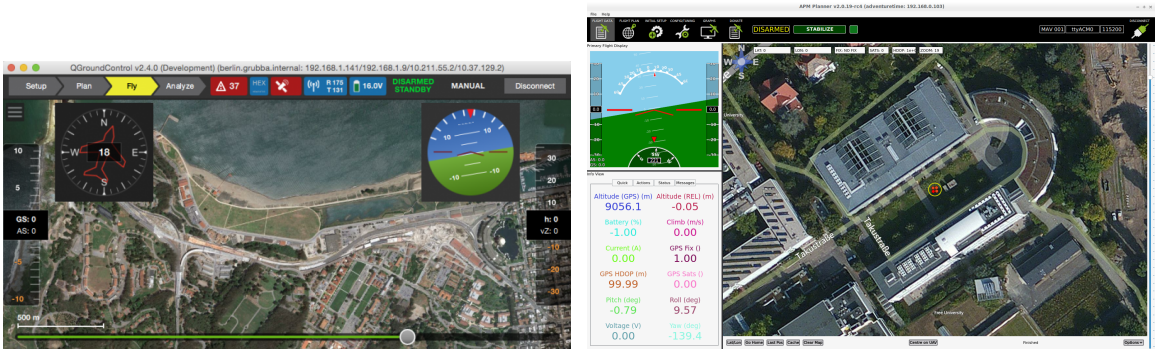
Tabelle 2.1.: Flugcontroller

Gegenüberstellung verschiedener Flugcontroller. Verglichen werden Hardwareeigenschaften wie Sensoren, Fähigkeiten der Firmware und am Boden laufender Software zur allgemeinen Konfiguration des Copters und zur Programmierung abzufliegender Wegpunkte.

2.2.3. Bodenstation

Eine Bodenstationssoftware zur allgemeinen Konfiguration des Copters, Planung von Routen und Anzeige von Flugdaten ist prinzipbedingt stark an den verwendeten Flugcontroller gekoppelt, da diese unterschiedliche Protokolle zur Datenübertragung verwenden. Deswegen wird diesem Thema nur ein kurzer Unterabschnitt gewidmet, der exemplarisch den Funktionsumfang einer Bodenstation eines der im Abschnitt 2.2.2 auf Seite 12 gezeigten Flugcontrollers umreißt.

Sowohl die PX4-Firmware [65] als auch ArduCopter [66] setzen auf das offene, erweiterbare Protokoll MAVLink [67], was prinzipiell die Nutzung verschiedener Bodenstationsprogramme ermöglicht. Da sich die Flugcontroller aber in ihren Konfigurationsmöglichkeiten unterscheiden, bringen beide eigene Software mit, die speziell für sie angepasst ist. Das PX4 verwendet hierfür QGroundControl (Abbildung 2.8a) und ArduCopter setzt das Programm APM Planner (Abbildung 2.8b) ein, welches im Folgenden beschrieben wird.



(a) QGroundControl [68] **(b)** APM Planner

Abbildung 2.8.: Bodenstationssoftware
Die Programme QGroundControl (Abb. 2.8b) und APM Planner (Abb. 2.8a) kommunizieren über das MAVLink-Protokoll mit dem Multicopter und können zur Konfiguration und Planung von Flugrouten verwendet werden. Bei beiden wird hier eine Flugdatenansicht gezeigt, die neben der Lage des Copters durch Kompass und künstlichen Horizont auch die aktuelle Position in einer Landkarte anzeigt.

Bei der grundlegenden Konfiguration wird die Art des Multicopters durch Anzahl und Positionierung der Motoren festgelegt (vgl. Abbildung 2.1 auf Seite 8) sowie die angeschlossenen Sensoren ausgewählt und kalibriert. Daneben lässt sich auch die Fernsteuerung einrichten. Hierbei werden die maximalen Ausschläge und Nullstellungen der Steuerhebel bestimmt und optional weiteren Kanälen, die etwa über Kippschalter oder Taster gesteuert sein können, Funktionen wie der Wechsel zwischen verschiedenen Flugmodi zugewiesen. Sofern der Flugcontroller über eine Funkverbindung oder auch per Kabel mit dem Bodenstationsrechner verbunden ist, können Fluglage und Position des Copters angezeigt werden (Abbildung 2.8b). In einer

anderen Ansicht kann der Verlauf verschiedener per MAVLink übertragener Sensordaten visualisiert werden (Abbildung 2.9a). Außerdem können in dieser Darstellung im Stillstand die Logdaten des internen Speichers des Flugcontrollers übertragen werden, so dass sie sich als Logdatei speichern oder genauso wie die MAVLink Daten anzeigen lassen. Nicht zuletzt stellt APM Planner eine Landkarte zur einfachen Planung von Flugrouten aus Wegpunkten bereit (Abbildung 2.9b). Für jeden dieser Punkte lassen sich verschiedene Optionen wie die Änderung der Flughöhe, Geschwindigkeit oder Ausrichtung des Copters festlegen. Auch können Punkte zum automatischen Starten, Landen oder Rückkehren zum Startpunkt bestimmt werden.

(a) Sensordaten **(b)** Routenplanung

Abbildung 2.9.: Bodenstation APM Planner
Weitere Ansichten der Bodenstationssoftware APM Planner (Abbildung 2.8b auf Seite 14): Sensordaten können visualisiert (Abb. Abbildung 2.9a) und abzufliegende Routen auf einer Landkarte geplant werden (Abb. Abbildung 2.9b).

2.3. Antrieb

Der Modellbau bietet besonders im Bereich der Flugmodelle viele teils sehr kostengünstige Komponenten, die in Multicoptern verwendet werden können. Deswegen konzentriert sich dieser Abschnitt besonders auf solche Teile.

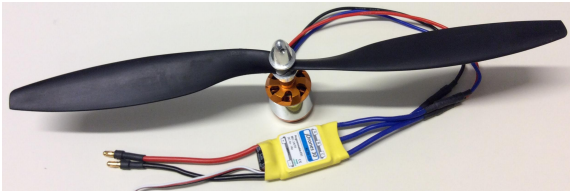


Abbildung 2.10.: Motor mit Propeller
Der Propeller ist direkt mit der Welle des bürstenlosen Außenläufers verbunden, der den Strom von einem Motorregler (ESC) bezieht.

Die Propeller sind in der Regel direkt auf der Motorwelle befestigt und werden ohne zusätzliche Getriebe in Rotation versetzt, wie Abbildung 2.10 zeigt. Dieser Verzicht auf zusätzliche bewegliche Teile verringert die Komplexität des Systems und reduziert das Gewicht.

2.3.1. Stromversorgung

Zur Stromversorgung bieten sich wegen ihrer relativ hohen Energiedichte Lithium-Polymer-Akkus an. Bei gängigen Exemplaren aus dem Modellbau sind mehrere Zellen mit einer Nennspannung von jeweils 3,7 V durch Reihen- oder Parallelschaltungen verbunden. Abbildung 2.11 zeigt einen Akku mit drei in Reihe geschalteten Zellen.

Abbildung 2.11.: Lithium-Polymer-Akku
Bei diesem Akku mit einer Ladung von 5000 mAh sind drei Zellen mit einer Nennspannung von jeweils 3,7 V in Reihe geschaltet, so dass sich eine Gesamtspannung von 11,1 V ergibt.

2.3.2. Motoren

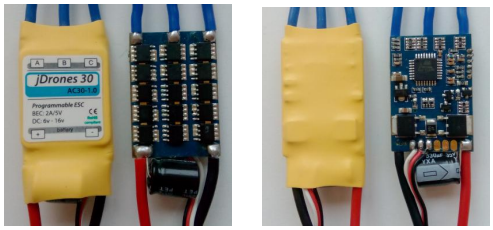
Als Motoren werden bürstenlose Außenläufer verwendet. Diese sind effizienter als das Pendant mit Schleifkontakten zur Stromübertragung in die sich drehenden Spulen des Motors. Beim bürstenlosen Motor (Abbildung 2.12a) werden die Spulen des Stators (Abbildung 2.12b) direkt mit Drehstrom versorgt, der ein sich drehendes magnetisches Feld erzeugt und so den mit Permanentmagneten bestückten Rotor (Abbildung 2.12c) in Bewegung versetzt.

(a) Motor **(b)** Stator **(c)** Rotor

Abbildung 2.12.: Bürstenloser Motor
Bei diesem bürstenlosen Außenläufer (Abb. 2.12a) dreht sich der glockenförmige außen liegende mit Permanentmagneten besetzte Rotor (Abb. 2.12c) um den im Inneren fest stehenden Stator (Abb. 2.12b), der durch Spulen ein magnetisches Feld erzeugt.

2.3.3. Motorsteuerung (ESC, *electronic speed control*)

Um aus dem Gleichstrom der Akkus den Drehstrom für die Motoren zu erzeugen, werden elektronische Steuergeräte benötigt (Abbildung 2.13). Diese besitzen einen Mikroprozessor, der den eingehen Gleichstrom durch Transistoren ein- und ausschaltet und so den benötigten Dreiphasenwechselstrom moduliert. Die Firmware der für Modellflugzeuge entwickelten ESCs verfügt oft über zusätzliche Logik, die die Geschwindigkeitsvorgabe weichzeichnet [20]. Dies soll sonst entstehende starke Spitzen des Stromverbrauchs glätten, die die Schaltelektronik des Reglers beschädigen könnten [20]. Auch kann der damit einhergehende Spannungsabfall einen Ausfall der Bordelektronik verursachen [20]. Das Weichzeichnen führt aber auch dazu, dass ausreichend dimensionierte, von dieser Problematik nicht betroffene, ESCs unnötig träge reagieren und der Copter weniger präzise gesteuert werden kann. Um dem entgegen zu wirken, kann eine alternative Firmware verwendet werden, die die Drehzahlvorgaben direkt umsetzt [69].



(a) Oberseite **(b)** Unterseite

Abbildung 2.13.: Motorregler
Gezeigt wird ein Modellbaumotorregler, der Ströme von bis zu 30 A steuern kann. Dabei schaltet ein Mikroprozessor (Abb. 2.13b) über Transistoren (Abb. 2.13a) den eingehenden Gleichstrom (rotes und schwarzes Kabel) pulsweise an und erzeugt so einen dreiphasigen Wechselstrom (blaue Kabel) zur Ansteuerung des bürstenlosen Motors. Über die weiße Leitung des dreiadrigen Kabels wird die gewünschte Drehzahl vorgegeben.

2.3.4. Propeller

Da sich die Motoren zum Drehmomentausgleich in unterschiedliche Richtungen drehen, werden dementsprechend links- und rechtsdrehende Propeller benötigt. Ver-

(a) rechts **(c)** rechts

(b) links **(d)** links

Abbildung 2.14.: Propeller
Gezeigt werden jeweils ein links- und ein rechtsdrehender Propeller zwei verschiedener Varianten einer $12 \times 4{,}5''$-Luftschraube, die sich in der Form der Blätter unterscheiden. Die beiden Ziffern geben dabei Propellerdurchmesser und Steigung in Zoll an (Durchmesser: 305 mm, Steigung: 114 mm).

fügbare Modelle besitzen zwei bis vier Rotorblätter und sind aus Kunststoff, Holz, glas- oder kohlefaserverstärktem Kunststoff oder aus Kohlefasergewebe laminiert. Abgesehen vom Durchmesser unterscheiden sie sich in Form und Steigung der Rotorblätter. Abbildung 2.14 auf Seite 17 stellt zur Veranschaulichung zwei Modelle gleichen Durchmessers mit unterschiedlich geformten Blättern gegenüber.

2.4. Rahmen

Genau wie beim Antrieb (Abschnitt 2.3 auf Seite 15) werden aus Kostengründen nur Multicopterrahmen aus dem Modellbau- und Hobbybereich betrachtet. Der generelle Aufbau dieser besteht aus einer zentralen Plattform, um die herum die Arme zur Befestigung der Motoren angeordnet sind, wie Abbildung 2.15 veranschaulicht.

(a) Turnigy Talon [70] **(b)** PYRAMID X580 [71] **(c)** RotorBits HexCopter [72]

Abbildung 2.15.: Multicopterrahmen
Übersicht einiger Multicopterrahmen aus dem Modellbau. Von der zentralen Plattform, die Akku und Elektronik tragen soll, gehen Ausleger zur Befestigung der Motoren ab.

(a) Turnigy Talon [70] **(b)** HMF U580 [73] **(c)** RotorBits HexCopter [72]

Abbildung 2.16.: Motorbefestigungen
Am Ende der Arme der Multicopterrahmen (siehe Abbildung 2.15) sitzen die Motoren. Deren Halterungen sind auf unterschiedliche Weise an den runden oder quadratischen Auslegern befestigt.

Für das zentrale Stück wird auf glasfaserverstärkte und kohlenstofffaserverstärkte Platten und Kunststoffspritzgussteile (GFK bzw. CFK) zurückgegriffen. Die Ausle-

ger sind in der Regel Aluminium-, GFK- oder CFK-Rohre mit rundem oder quadratischem Querschnitt. Miteinander verbunden werden die Teile durch direkte Verschraubung oder Kunststoff- und Aluminiumklemmen. Zur Befestigung der Motoren an den Armen werden meist Klemmen verwendet (Abbildung 2.16 auf Seite 18), die am Ende aufgesteckt oder an beliebiger Stelle geklemmt werden können. Das Aufstecken bringt den Nachteil mit sich, dass die Motorpositionen und somit das Verhalten des Copters nicht variiert werden können und fest vorgegeben sind. In der Gegenüberstellung gängiger Rahmen in Tabelle 2.2 fällt besonders das teils hohe Gewicht auf, obwohl auf leichte Materialien wie Kohlefaser zurückgegriffen wurde. Neben diesem Aspekt sind die meisten Rahmen so konstruiert, dass der Schwerpunkt des Copters weit unterhalb der Rotorebene liegt. Bedingt ist dies durch die Positionierung schwerer Komponenten wie Akku und Gimbal an der Unterseite der Plattform, die in einer Ebene mit den Armen und somit unter der Rotorebene liegt. Entgegen der Intuition ist ein tiefer Schwerpunkt dem Flugverhalten nicht zuträglich. Da die Lageregelung von einem Flugcontroller gesteuert wird, kann ein instabiles oszillierendes System in Kauf genommen werden [74]. Ein knapp unterhalb der Rotorebene platzierter Schwerpunkt ergibt einen Copter, der unanfällig für Störungen ist und schnell auf Steuerbefehle reagiert [74].

Rahmen	Propeller	Spannweite	Gewicht	Propellerdurchmesser	Materialien
Turnigy Talon [70]	4	498 mm	240 g	229 mm[a]	CFK
Turnigy Talon V2 [75]	4	550 mm	280 g	229 mm[a]	CFK
PYRAMID X650F [76]	4	550 mm	598 g	280 mm	GFK, Aluminium
HMF U580 [73]	4	580 mm	558 g	381 mm	CFK, GFK
PYRAMID X580 [71]	4	585 mm	418 g	280 mm	GFK, Aluminium
PYRAMID T650-X4-16 [77]	4	650 mm	580 g	356 mm	GFK
Skylark M4-680 [78]	4	680 mm	420 g	381 mm	CFK
RotorBits HexCopter [72]	6	720 mm	360 g	254 mm	CFK, GFK

[a]Wert stammt mangels Herstellerangabe von mit dem Rahmen kompatibel gelisteten Motoren

Tabelle 2.2.: Multicopterrahmen

Gegenüberstellung von Multicopterrahmen aus dem Modellbau anhand ihrer Eckdaten.

2.5. Bordcomputer

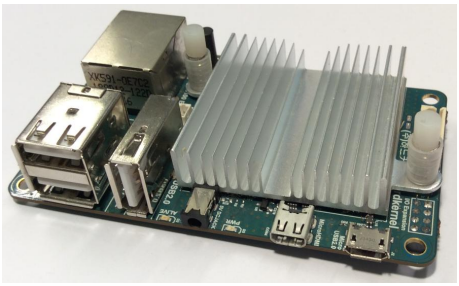


Abbildung 2.17.: Einplatinencomputer
Einplatinencomputer ODROID-U3: 2 GiB Arbeitsspeicher, 1,7 GHz 4-Kern ARM Cortex-A9 CPU, 83 mm × 48 mm, 48 g

Neben dem Flugcontroller, der die grundlegende Steuerung des Copters übernimmt, wird ein zusätzlicher Rechner benötigt, der neben komplexen Aufgaben wie Bildverarbeitung auch die Datenübertragung zur Bodenstation übernehmen soll. Für die Eignung als Bordcomputer sind neben Rechenleistung und Schnittstellen vor allem Größe und Gewicht entscheidend. Abbildung 2.17 zeigt ein kreditkartengroßes Modell, das einer Auswahl gängiger Einplatinencomputer in Tabelle 2.3 gegenübergestellt wird. Abmessungen und Gewicht der Computer liegen nah beieinander, einzig Prozessor und Arbeitsspeicher weisen sichtliche Unterschiede auf. Inwiefern diese relevant sind, wird sich erst in der Praxis zeigen und hängt stark von den Anforderungen der Software des jeweiligen Experiments ab.

		Raspberry Pi 2 Model B [79, 80]	ODROID-U3 [81, 82]	IGEPv2 [83, 84]	Banana Pi M3 [85]
CPU	Architektur	Cortex-A7	Cortex-A9	Cortex-A8	Cortex-A7
	Kerne	4	4	1	8
	Takt	900 MHz	1,7 GHz	1 GHz	2 GHz
RAM		1 GiB	2 GiB	512 MiB	2 GiB
Gewicht		45 g	48 g	47 g[a]	45 g
Abmessungen		85,6 mm × 56 mm	83 mm × 48 mm	95 mm × 65 mm	92 mm × 60 mm
USB Ports		4	3	2	3
GPIO-Pins		40	36[b]	28[c]	40

[a] nachgewogen
[b] nur über separates IO Shield
[c] expansion connector J990

Tabelle 2.3.: Auswahl gängiger Einplatinencomputer

Die Einplatinencomputer werden anhand von Eigenschaften, die für den Bordcomputer eines Multicopters besonders relevant sind, gegenübergestellt. Neben Rechenleistung und Anschlüssen für weitere Komponenten (USB-Ports und GPIO-Pins) spielen vor allem Größe und Gewicht eine entscheidende Rolle.

2.6. Zusammenfassung

In diesem Kapitel wurden gängige Flugcontroller, zugehörige Software, Einplatinencomputer sowie Multicopterrahmen und -antrieb vorgestellt. Es hat sich gezeigt, dass kein Komplettsystem die Anforderungen des Projekts *Neurocopter* vollumfänglich erfüllt.

Verfügbare Flugcontroller und Konfigurationssoftware können je nach Modell unverändert verwendet werden. Für die Anbindung des Controllers an den Bordcomputer und die dortige Auswertung, Kombination mit Kamerabildern und Weiterleitung der Flugdaten muss jedoch entsprechende Software entwickelt werden. Für den Antrieb des Copters kann vollständig auf kostengünstige Modellbaukomponenten zurückgegriffen werden. Einzig die betrachteten Rahmen weichen zu sehr von den Anforderungen ab. Sie sind entweder unnötig schwer, haben eine sehr geringe Armlänge, einen ungünstigen Schwerpunkt oder sind durch fest vorgegebene Motorpositionen und andere Befestigungen zu unflexibel.

Daraus ergibt sich, dass zur Durchführung des Projekts ein Multicopterrahmen konstruiert wird, für dessen Antrieb Standardkomponenten verwendet werden können. Außerdem wird eine Software entwickelt, die die Kombination von Flugcontroller, Bordcomputer, Kamera und Bodenstation bewerkstelligt und so ein Framework für zukünftige Experimente bereitstellt.

3. Implementierung

Dieses Kapitel beschreibt sowohl die Konstruktion des Multicopters, als auch die Software des Bordcomputers die zusammen als *fliegendes Labor* zur Erforschung der Hirnaktivität von Honigbienen verwendet werden sollen (siehe Abschnitt 1.2 auf Seite 4).

3.1. Auswahl der Bordelektronik

Die Bordelektronik des Copters wird von einem Spannungswandler (Abbildung 3.1) gespeist, der direkt an den Akku, der auch die Motoren mit Strom versorgt, angeschlossen ist.

Abbildung 3.1.: Spannungswandler [86]
Spannungswandler zur Stromversorgung der Bordelektronik. Aus den 11,1 V des Akkus werden 5 V erzeugt.

3.1.1. Bordcomputer

Aus den in Abschnitt 2.5 auf Seite 20 vorgestellten Einplatinencomputern wurde das ODROID-U3 ausgewählt. Dieses hat zwar nicht die größte Rechenleistung, verfügt aber neben dem Banana Pi M3 mit 2 GiB über den größten Arbeitsspeicher, hat die kleinste Platine und einen geringeren Preis als das leistungsstärkere Banana Pi M3. Für die Funkverbindung zur Bodenstation wird ein herkömmlicher WLAN-USB-Stick verwendet, der direkt in den Rechner gesteckt und mit Klebeband an der Buchse gegen Herausfallen gesichert wird.

3.1.2. Flugcontroller

Von den in Abschnitt 2.2.2 auf Seite 12 vorgestellten Flugcontrollern scheinen das APM 2.5, PX4FMU und Pixhawk mit der ArduCopter- oder PX4-Firmware am besten geeignet. Dies sind die einzigen Systeme, für die eine quelloffene plattformunabhängige Bodenstationssoftware existiert, mit der unter anderem Flugrouten konfiguriert werden können.

Da Teile der PX4-Codebasis in C verfasst sind, wurde die Entscheidung zu Gunsten von ArduCopter getroffen. Dies ist in der Annahme begründet, dass der komplett in C++ verfasste Quelltext wegen der erweiterten Möglichkeiten der Sprache gegenüber C einfacher zu verstehen ist, so dass gegebenenfalls nötige Anpassungen leichter durchgeführt werden können. Zudem teilt sich die ArduCopter-Firmware mit den beiden anderen auf die gleiche Hardware setzenden Projekten *ArduRover* [87] und *ArduPlane* [88] eine gemeinsame Codebasis. Daher wäre es prinzipiell möglich, das für den *Neurocopter* entwickelte Framework auch mit fahrenden ArduRover-Modellen zu verwenden, um beispielsweise den Verlauf geplanter Experimente vorab in einer einfacheren Umgebung zu überprüfen.

Von den drei von der ArduCopter-Firmware unterstützen Flugcontrollern wurde das APM 2.5 wegen des geringen Preises gewählt, da es sich abgesehen von einem deutlich schwächeren Prozessor nicht erheblich von den anderen unterscheidet.

Sensoren

Das APM 2.5 bringt auf der Platine Gyroskop, Accelerometer, Barometer und Magnetometer mit. Letzteres wird durch ein externes ersetzt, das zur besseren Abschirmung gegenüber Störfeldern weiter entfernt von der restlichen Elektronik angebracht werden kann. Außerdem werden noch ein GPS-Modul und ein Sonar angeschlossen. Eine Beschreibung der verschiedenen Sensortypen befindet sich in Abschnitt 2.2.1 auf Seite 9.

Flugmodi

Die ArduCopter-Firmware stellt verschiedene Flugmodi bereit [89], zwischen denen über einen zusätzlichen Kanal der Fernsteuerung gewechselt werden kann. Dazu wird der entsprechende Ausgang des Empfängers mit dem APM 2.5 verbunden (vgl. Abbildung 2.2 auf Seite 9), das Signaländerungen registriert und sofort den Modus ändert. Das Flugverhalten geht dabei von einem einfachen manuellen Modus, bei dem Schub, Roll- und Nickwinkel des Copters direkt über die Steuerknüppel vorgegeben werden [90], über das zusätzliche Halten der Höhe [91] oder auch der Position im Raum [92, 93] bis hin zum automatischen Abfliegen von vorkonfigurierten Routen aus Wegpunkten [94]. Letzteres lässt sich jederzeit unterbrechen, indem in einen anderen Modus gewechselt wird [94]. Zudem kann der Copter über weitere Modi angewiesen werden, sofort zu landen [95] oder zuvor noch zum Startpunkt zurückzukehren [96].

Gimbal

In der Gegenüberstellung der Flugcontroller (siehe Tabelle 2.1 auf Seite 13) wurde bereits festgestellt, dass das APM 2.5 in der Lage ist, ein Gimbal anzusteuern und

so eine Kamera unabhängig von der Lage des Copters auszurichten. Dazu werden die Servos der entsprechenden Drehachsen entweder mit gesonderten speziell dafür vorgesehenen oder auch nicht verwendeten Ausgängen der Motorsteuerungen verbunden [34]. Mit der Konfigurationssoftware lassen sich dann Winkelbereiche und Nullstellungen festlegen. Auch können Kanäle der Fernsteuerung ausgewählt werden, um manuell einen anderen Winkel zur Horizontalen vorzugeben.

Verbindung zum Bordrechner

Zur Verbindung mit dem Bordcomputer wird ein kurzes USB-Kabel verwendet, das auf Flugcontroller-Seite einen Micro-USB-Stecker und an der anderen einen herkömmlichen USB-A-Stecker besitzt. Dieser lässt sich einfach aus dem ODROID-U3 ausstecken und kann dann mithilfe eines USB-Verlängerungskabels mit einem anderen Computer, wie dem Bodenstationsrechner, verbunden werden, so dass ein direkter Zugriff auf das APM 2.5 möglich ist.

3.2. Konstruktion des Copters

Dieser Abschnitt beschreibt die Konstruktion der einzelnen Komponenten, die zusammen den in Abbildung 3.2 gezeigten Quadcopter ergeben.

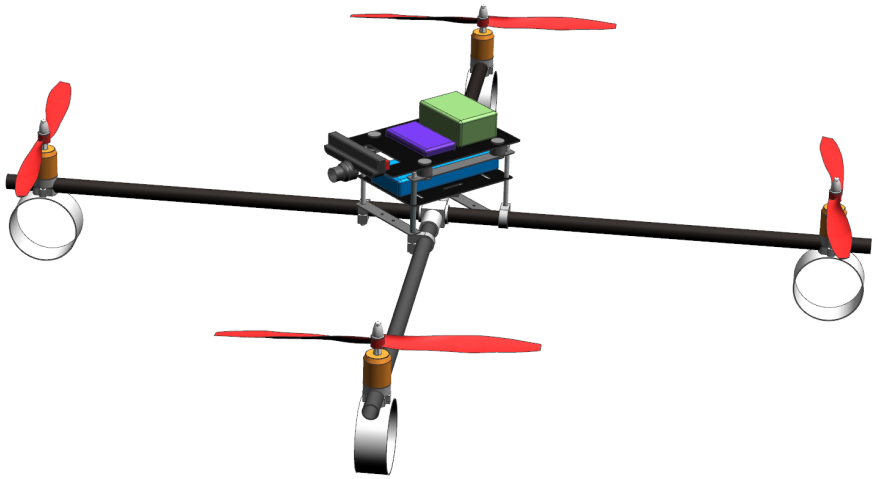


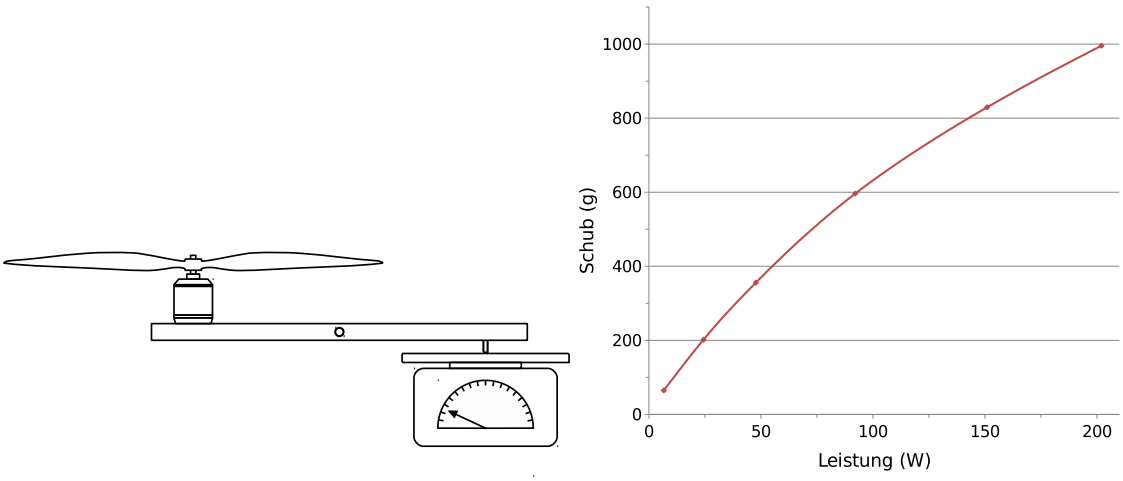
Abbildung 3.2.: Neurocopter-3D-Modell
Gezeigt wird ein Modell des Copters. Oberhalb der Arme, die durch ein Aluminiumteil in der Mitte starr verbunden sind, befindet sich die Bordelektronik auf mehreren Ebenen. Zuoberst befinden sich Flugcontroller, Empfänger, Kamera und Bordrechner auf einer gedämpften Platte. Darunter liegt der Akku. Zwischen diesem und den Armen werden die ESCs und ein Spannungswandler zur Stromversorgung der Bordelektronik platziert.

Wie sich in Abschnitt 2.4 auf Seite 18 herausgestellt hat, muss ein Rahmen konstruiert werden, der die Anforderungen des Projekts erfüllt. Dazu muss zuerst entschieden werden, wie viele Arme und Motoren der Multicopter besitzen soll, da dieser Entschluss das Design grundlegend beeinflusst. Ein viermotoriger Quadcopter bietet sich aus vielerlei Gründen an: Zum einen wird die Fertigung der Einzelteile durch die rechtwinklige Anordnung der Arme vereinfacht. Zum anderen nimmt der Wirkungsgrad eines Propellers mit dessen Durchmesser zu [97], so dass ein möglichst effizienter Multicopter den nötigen Schub durch größere, statt zusätzliche Propeller erzeugt. Selbstverständlich hat die Formgebung der Propellerblätter auch einen erheblichen Einfluss auf den Wirkungsgrad [98], doch das ist für diese Entscheidung unerheblich.

Der verwendete Flugcontroller bietet die Möglichkeit zur Ansteuerung mehrerer Servos, die ein Gimbal drehen und so die Neigung des Copters ausgleichen können (vgl. Abschnitt 3.1.2 auf Seite 24). Die in Abschnitt 1.2.1 auf Seite 4 angedeuteten Einsatzmöglichkeiten für solch einen Lageausgleich sind jedoch sehr versuchsspezifisch. Daher kann die Konstruktion erst vorgenommen werden, wenn konkretere Einsatzszenarien feststehen, und sie ist somit nicht Teil dieser Arbeit.

3.2.1. Schubmessung

Um einen Anhaltspunkt für die Dimensionierung des Copters zu erhalten, wurde der Schub eines 305 mm-Propellers an einem 200 W-Motor bestimmt. Solch ein Antrieb



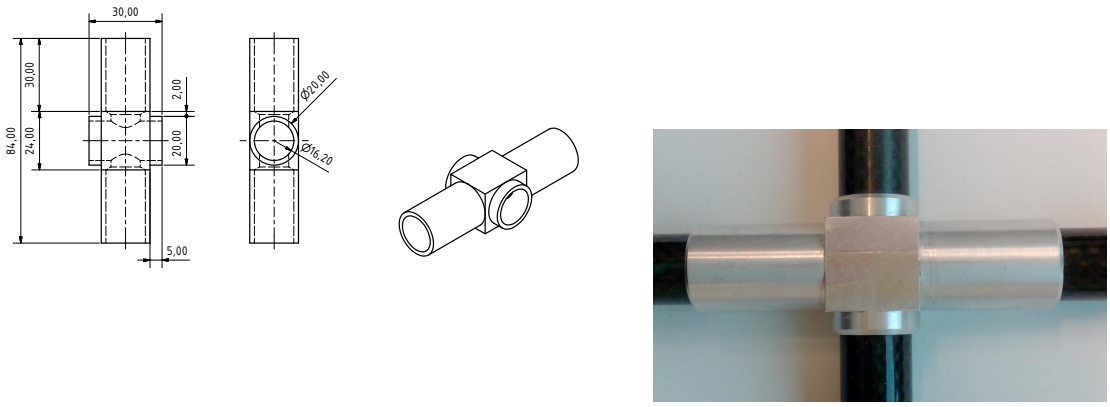
(a) Prüfstand **(b)** Schubkurve

Abbildung 3.3.: Schubmessung
Zur Schubmessung wurde ein 200 W-Motor mit einem 305 mm-Propeller an einem Hebel befestigt, über den er auf eine Waage drückt (Abb. 3.3a). Auf diese Weise wurden bei verschiedenen Drehzahlen der statische Schub und und die Leistungsaufnahmen gemessen (Abb. 3.3b): Der Wirkungsgrad nimmt mit zunehmendem Schub ab.

wird etwa in Quadcoptern verwendet, die ein Gimbal mit einem Action-Camcorder tragen können und somit eine ähnliche Nutzlast wie der geplante *Neurocopter* haben. Abbildung 3.3 auf Seite 26 zeigt den prinzipiellen Aufbau der Messapparatur und die damit ermittelte Schubkurve. Demnach beträgt der maximale Schub eines Propellers knapp 1000 g. Um den Copter auch bei Windböen jederzeit kontrollieren zu können, sollte der gemeinsame Schub aller Propeller mindestens das doppelte des Gesamtgewichts betragen, was demnach unter 2000 g liegen muss. Da das Gewicht solcher Copterrahmen bei bis zu 600 g liegt (siehe Abschnitt 2.4 auf Seite 18), scheint der geprüfte Antrieb ausreichend für das *fliegende Labor*, den *Neurocopter* zu sein.

3.2.2. Arme

Als Material für die Arme bieten sich CFK-Rohre an. Diese sind leicht und äußerst verwindungssteif. Das Gewicht der verwendeten 16 mm dicken Rohre mit einer Wandstärke von 1 mm beträgt nur 79,5 $\frac{g}{m}$. Die gezeigten erhältlichen Rahmen (Abschnitt 2.4 auf Seite 18) verschrauben oder klemmen die Arme an einer zentralen Plattform. Dieser Aufbau bringt den Nachteil mit sich, dass das Mittelstück, das ansonsten nur die Bordelektronik trägt, stabiler ausgelegt werden muss, um die durch die langen Ausleger auftretenden Kräfte aufnehmen zu können, die an dieser Stelle durch deren Hebelwirkung am größten sind. Um dies zu umgehen, wurde ein Aluminiumdrehteil konstruiert, in dem die Arme verklebt werden (Abbildung 3.4).



(a) Konstruktionszeichnung **(b)** mit Armen verklebt

Abbildung 3.4.: Mittelstück
Ein aus Aluminium gedrehtes und gefrästes 28 g schweres Mittelstück verbindet die eingeklebten Copterarme aus CFK-Rohr.

Die Spannweite wurde mit 900 mm sehr großzügig bemessen und lässt zwischen den 305 mm-Propellern einen Abstand von einen Rotordurchmesser zu. Zum einen wird

der Copter mit zunehmender Armlänge träger, wodurch ein stabileres Flugverhalten erzielt wird. Außerdem stören sich zu nah beieinander liegende Rotoren und verringern den Wirkungsgrad des Gesamtsystems [99].

3.2.3. Plattform

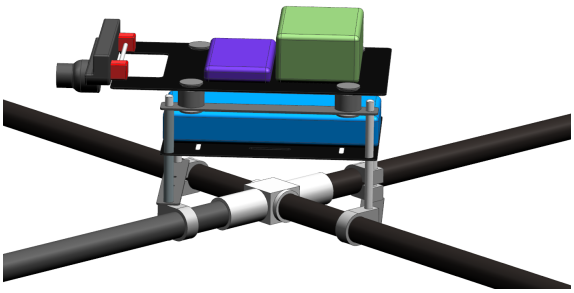


Abbildung 3.5.: Plattform-3D-Modell
Schematischer Aufbau der Elektronikplattform, die über vier Klemmen mit den Armen des Copters verbunden ist.

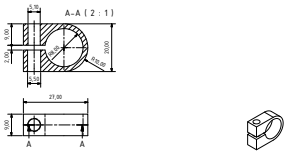
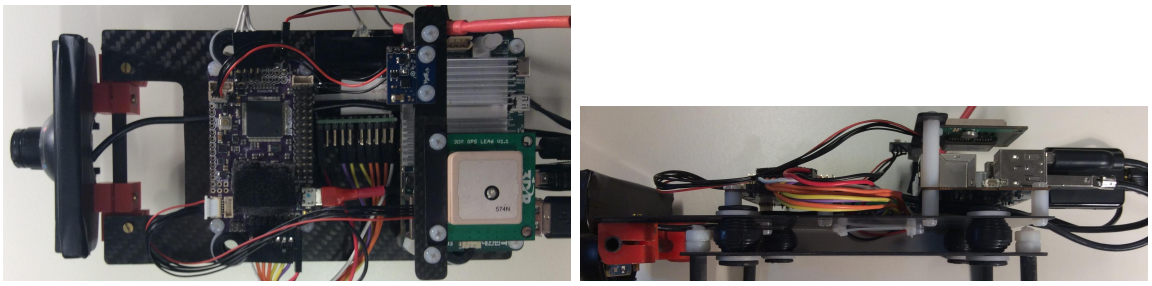


Abbildung 3.6.: Plattformklemme
Die Elektronikplattform wird mit vier dieser gefrästen Klemmen an den Armen des Copters befestigt.

Da die Arme des Quadcopters durch ein Aluminiumteil verbunden sind, muss die Plattform lediglich die Bordelektronik tragen und kann somit aus nur 1 mm dicken CFK-Platten konstruiert werden. Wie bereits in Abschnitt 2.4 auf Seite 18 erläutert, sollte der Schwerpunkt des Copters knapp unterhalb der Rotorebene liegen. Dazu wird die gesamte Elektronik oberhalb der Arme auf mehreren Ebenen platziert. Diese



(a) Draufsicht **(b)** Seitenansicht

Abbildung 3.7.: Elektronikplattform
Auf der gedämpften Elektronikplattform befinden sich Flugcontroller und Sensoren, der Empfänger der Fernsteuerung und der Bordcomputer. Auf der linken Seite ist die beweglich angebrachte Kamera zu sehen. Die Komponenten sind so angeordnet, dass sich der Schwerpunkt in der Mitte der Plattform befindet.

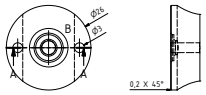
sind zusammen mit dünnen CFK-Rohren als Abstandshalter auf Polyamidgewindestangen aufgefädelt und werden so mit den Klemmen (Abbildung 3.6 auf Seite 28) zur Befestigung an den Armen verspannt. Dieser Aufbau wird in Abbildung 3.5 auf Seite 28 schematisch gezeigt: Direkt über den Armen werden die ESCs und ein Spannungswandler zur Stromversorgung der Bordelektronik angebracht. Darüber liegt der Akku auf einer CFK-Platte, an der er mit einem Riemen befestigt wird. Zuoberst befindet sich die restliche Elektronik samt Kamera auf einer gedämpften Platte (Abbildung 3.7 auf Seite 28). Dadurch werden die vibrationsanfälligen Sensoren von den Motoren entkoppelt, das Magnetometer möglichst weit von den restlichen Komponenten entfernt und das GPS mit ungestörter Sicht zum Himmel ausgerichtet. Zur Dämpfung werden Gummipuffer verwendet, die ursprünglich zur Aufhängung von Action-Camcordern konzipiert sind. Abbildung 3.7 auf Seite 28 zeigt die fertig bestückte gedämpfte Elektronikplatte, an deren Vorderseite eine Kamera drehbar angebracht ist. Dafür wurde das Kunststoffgehäuse einer *PlayStation Eye*-Kamera zur Gewichtsersparnis zurechtgeschnitten und mit gefrästen Polyamidhalterungen und einem CFK-Rohr als Achse gelenkig befestigt. Die CFK-Platte ist in diesem Bereich durch eine auflaminierte zweite Schicht verstärkt.

3.2.4. Motorklemmen

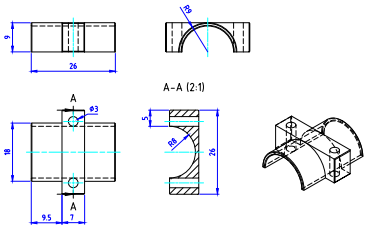
Zur Befestigung der Motoren an den Armen wurde ein Klemmsystem konzipiert. Die in Abbildung 3.8 auf der nächsten Seite gezeigten zweiteiligen Aluminiumklemmen sind als Drehteile ausgeführt und wurden mit einer Fräse nachbearbeitet. Neben den Motoren halten die Klemmen auch die Landefüße des Copters, die aus Ringen eines PVC-Rohres gefertigt wurden. Abbildung 3.9 auf der nächsten Seite zeigt einen montierten Motor.

3.2.5. Antrieb

Bei Motoren, ESCs und Propellern wurde wie in Abschnitt 2.3 auf Seite 15 beschrieben auf Modellbaukomponenten zurückgegriffen. Als Motoren kommen bürstenlose Außenläufer mit einer maximalen Leistungsaufnahme von 243 W und Drehzahl von $880\,\frac{\mathrm{U/min}}{\mathrm{V}}$ zum Einsatz, die die Propeller mit einem Durchmesser von 305 mm antreiben. Zur Stromversorgung wird ein dreizelliger Lithium-Polymer-Akku mit einer Nennspannung von 11,1 V und einer Ladung von 5000 mAh verwendet. Einzig die Motorregler, die einen maximalen Strom von 30 A schalten können, wurden durch Aufspielen einer anderen Firmware modifiziert, um die in Abschnitt 2.3.3 auf Seite 17 erläuterten negativen Eigenschaften von Modellbau-ESCs auszugleichen. Dazu wurden die zur Programmierung des Mikroprozessors nötigen, auf der Oberseite der Platine befindlichen 6 Pins über ein Flachbandkabel nach außen geführt. An dessen Ende wurde eine kleine Buchsenleiste mit einem Rastermaß von 1,27 mm gelötet, so dass ein Programmiergerät jederzeit erneut für eine mögliche Aktualisierung angeschlossen werden kann.



(a) Oberseite



(b) Unterseite

Abbildung 3.8.: Motorhalterung
Zur Befestigung der Motoren an den runden CFK-Armen werden zweiteilige 10 g schwere Klemmen aus Aluminium verwendet.

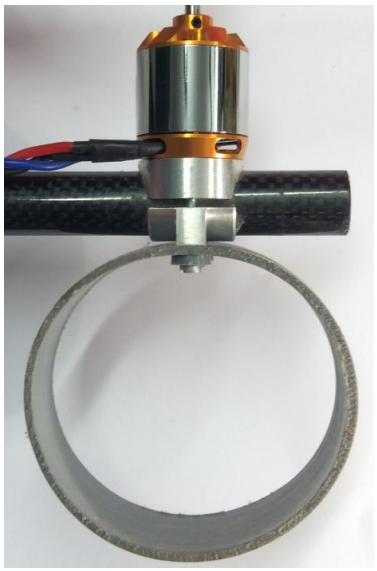


Abbildung 3.9.: Montierter Motor
Der Motor, die Klemme am Arm und der Landefuß aus PVC-Rohr werden durch zwei Schrauben zusammengehalten.

3.2.6. Manuelle Steuerung

Zur manuellen Steuerung wird die in Abschnitt 2.1 auf Seite 7 vorgestellte Funkfernbedienung *Turnigy 9X* verwendet. Dieses sehr kostengünstige Exemplar kann neben den vier Kanälen zur Steuerung noch vier weitere an den Empfänger *Turnigy 9X8C-V2* übertragen, womit mehr als ausreichend viele Zusatzfunktionen kontrolliert werden können.

3.2.7. Technische Daten

Abschließend werden hier noch die Eckdaten des fertig konstruierten Quadcopters aufgelistet (Tabelle 3.1 auf der nächsten Seite). Zur Schwerpunktbestimmung wur-

den sämtliche Komponenten einzeln gewogen und vermessen, so dass daraus der vertikale Schwerpunkt berechnet werden konnte.

Rahmengewicht[a]	343 g
Gewicht des Antriebs [b]	546 g
Gewicht der Bordelektronik [c]	218 g
Gesamtgewicht ohne Akku	1107 g
Gesamtgewicht mit Akku	1479 g
Schwerpunkt unter Rotorebene	26 mm
maximale Spannweite	900 mm
Schub[d]	3984 g

[a]incl. gedämpfter Elektronikplatte, Akkubefestigung, Motorhalterungen und Schrauben
[b]Motoren, Motorregler, Verkabelung, Propeller, etc.
[c]Flugcontroller, Sensoren, Bordcomputer, Kamera, etc.
[d]Summe des statischen Schubs aller vier Motoren

Tabelle 3.1.: Technische Daten

Eckdaten des *Neurocopters*. Der theoretische Maximalschub beträgt mehr als das doppelte des Gesamtgewichts, so dass der Copter über genügend Leistung verfügt, um jederzeit manövrierfähig zu sein und eine zusätzliche Nutzlast von 500 g zu tragen. Durch den knapp unter der Rotorebene liegenden Schwerpunkt kann er schnell auf Steuerbefehle reagieren.

3.3. Software

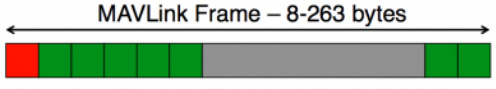
Nachdem in den vorherigen Abschnitten die Auswahl der Hardwarekomponenten und die Konstruktion des Copters beschrieben wurden, widmet sich dieses Kapitel nun der Software des *Neurocopters*. Dabei werden verwendete Programmbibliotheken und Frameworks (Rahmenstrukturen) charakterisiert und deren Integration und darauf aufbauende Entwicklungen veranschaulicht.

Um Programme zur Dokumentation oder Steuerung von Experimenten mit dem *fliegenden Labor* entwickeln zu können, wird ein Framework bereitgestellt, das die Kommunikation mit dem Flugcontroller und einen einfachen Zugriff auf dessen Sensordaten und die Bordkamera ermöglicht. Außerdem regelt das in C++ entwickelte Framework die Datenübertragung an die Bodenstation. C++ wurde als Sprache gewählt, um die begrenzten Ressourcen des Bordrechners möglichst effizient nutzen zu können. Als Betriebssystem kommt die vom Hersteller des ODROID-U3 (vgl. Abschnitt 3.1.1 auf Seite 23) bereitgestellte Linux-Distribution Ubuntu zum Einsatz.

3.3.1. MAVLink

ArduCopter setzt auf das quelloffene erweiterbare Protokoll MAVLink zur Datenübertragung. Auf diese Weise kann der Flugcontroller mit der Bodenstation und anderen Komponenten kommunizieren. Dieser Unterabschnitt beschreibt das Protokoll, verfügbare C- und neu entwickelte C++-Software.

3.3.1.1. Protokollbeschreibung



0	STX	Präambel
1	LEN	Nutzdatenlänge n
2	SEQ	Sequenznummer
3	SYS	Absender-System-ID
4	COMP	Absender-Komponenten-ID
5	MSG	Nachrichten-ID
6 $5+n$	PAYLOAD	Nutzdaten (0 bis 255 B)
$6+n$	CKA	Prüfsumme
$7+n$	CKB	

Abbildung 3.10.: Paketstruktur [67]
Ein MAVLink-Paket ist 8 B bis 263 B groß und enthält maximal 255 B Nutzdaten.

Ein MAVLink-Paket (Abbildung 3.10) kann bis zu 255 B Nutzdaten enthalten und verwendet nur zusätzliche 8 B zur Erkennung von Übertragungsfehlern und -verlusten, Kennzeichnung des Pakettyps und Identifikation des Absenders [67]. Die verschiedenen Pakettypen sind in XML spezifiziert, woraus automatisiert C-, C#- oder Python-Programmcode zum Senden und Empfangen der Daten generiert wird [100]. Die Pakete können dabei Schriftzeichen, Ganzzahlen und Fließkommazahlen und Arrays dieser Datentypen enthalten [67, 101]. Das Senden und Empfangen einer Nachricht wird am Bei-

spiel eines *Heartbeat*-Pakets (Auflistung 3.1), das die Betriebsbereitschaft des Systems signalisiert, erläutert.

Auflistung 3.1: *Heartbeat*-Paket

```
typedef struct __mavlink_heartbeat_t {
  uint32_t custom_mode;
  uint8_t  type;
  uint8_t  autopilot;
  uint8_t  base_mode;
  uint8_t  system_status;
  uint8_t  mavlink_version;
} mavlink_heartbeat_t;
```

Auflistung 3.2: MAVLink-Paket

```
typedef struct __mavlink_message {
  uint16_t checksum;
  uint8_t  magic;
  uint8_t  len;
  uint8_t  seq;
  uint8_t  sysid;
  uint8_t  compid;
  uint8_t  msgid;
  uint64_t payload64[
    (MAVLINK_MAX_PAYLOAD_LEN
     + MAVLINK_NUM_CHECKSUM_BYTES + 7)/8];
} mavlink_message_t;
```

Senden Um ein `mavlink_heartbeat_t`-Paket zu senden, muss zuerst ein entsprechendes `mavlink_message_t`-Paket (Auflistung 3.2) mit den gepackten Daten erzeugt werden. Dazu wird die Funktion `uint16_t mavlink_msg_heartbeat_encode_chan(...)`[1] verwendet, die als Eingabeparameter das *Heartbeat*-Paket, die Absenderadresse und die zum Kodieren verwendete Kanalnummer erhält. Beim Kodieren werden nun die einzelnen Komponenten des Pakets in Little-Endian-Byte-Reihenfolge in das Array `payload64` des als Ausgabeparameter übergebenen `mavlink_message_t`-Pakets geschrieben. Anhand der Kanalnummer wird dabei auf eine Struktur zugegriffen, die kanalspezifische Informationen wie die nächste Sequenznummer enthält. Alternativ kann auch die Funktion `uint16_t mavlink_msg_heartbeat_pack_chan(...)` verwendet werden, der anstelle des *Heartbeat*-Pakets sämtliche darin enthaltenen Felder als zusätzliche Parameter übergeben werden.
Anschließend werden die Nutzdaten aus dem `mavlink_message_t`-Paket mit der Funktion `uint16_t mavlink_msg_to_send_buffer(...)` extrahiert und serialisiert in einen übergebenen Puffer geschrieben. Aus diesem können sie dann über eine beliebige Verbindung übertragen werden.

[1] Die Parameter wurden zur besseren Lesbarkeit ausgelassen und durch „...“ angedeutet.

Empfangen Zum Parsen von empfangenen Daten werden diese byteweise mit der Funktion `uint8_t mavlink_parse_char(...)` ausgewertet, bis sie über einen Rückgabewert von 1 signalisiert, dass sich nun eine vollständige Nachricht im übergebenen Ausgabeparameter `mavlink_message_t` befindet. Dabei wird, genauso wie beim Senden, über einen weiteren Parameter auf den aktuellen Parsezustand des verwendeten Kanals zugegriffen.
Nun kann über das Feld `type` der `mavlink_message_t` der Nachrichtentyp abgefragt werden, um dann über entsprechende Funktionen auf den Inhalt der Nachricht zuzugreifen. Handelt es sich um ein `mavlink_heartbeat_t`-Paket, so kann beispielsweise mit der Funktion `uint8_t mavlink_msg_heartbeat_get_system_status(...)` dessen Feld `system_status` ausgelesen werden. Alternativ können auch alle Felder mit der Funktion `void mavlink_msg_heartbeat_decode(...)` in ein `mavlink_message_t`-Paket extrahiert werden.

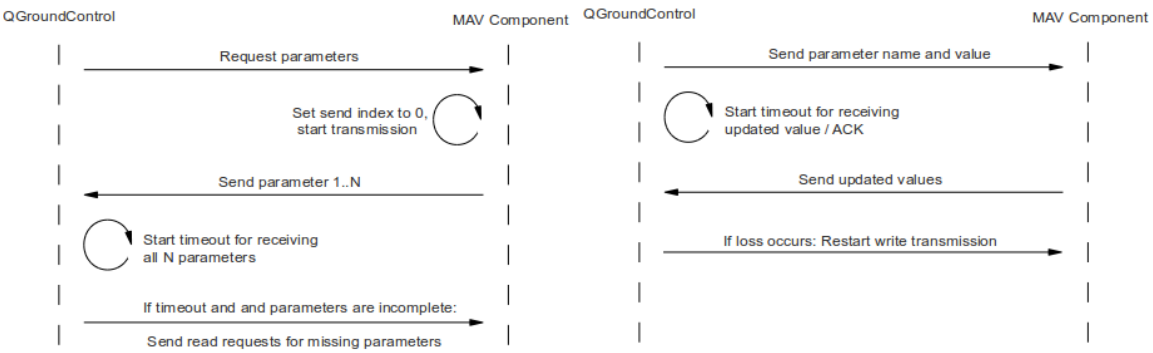
Waypoint-Protokoll Das Waypoint-Protokoll dient zur Konfiguration von abzufliegenden Routen aus Wegpunkten, an denen bestimmte Aktionen ausgeführt werden sollen. Um die Zustände von Copter und Bodenstation konsistent zu halten, ist das Protokoll transaktionsbasiert gestaltet, so dass im Fehlerfall der vorherige Zustand der Wegpunktliste des Copter unverändert bleibt [102]. Ein Wegpunkt-Paket (`mavlink_mission_item_t`) enthält neben der Empfängeradresse, Sequenznummer und Koordinaten die an diesem Punkt auszuführende Aktion sowie vier weitere Parameter mit aktionsspezifischer Bedeutung. Mögliche Aktionen sind beispielsweise das Starten oder Landen, Positionshalten, Ändern der Flughöhe, Ausrichten des Copters in eine bestimmte Himmelsrichtung, Ansteuern eines Servos, Abwerfen einer Nutzlast, Ausrichten eines Gimbals oder Auslösen einer Kamera.

Zur Manipulation der Liste gibt es mehrere Pakettypen, mit denen die Liste gelesen, gelöscht oder erweitert werden kann. Auch kann der aktive Punkt über seinen Index in der Liste bestimmt werden (`mavlink_mission_current_t`). Änderungen werden dabei vom Flugcontroller mit Empfangsbestätigung (`mavlink_mission_ack_t`) quittiert. Wenn der nächste Punkt der Liste erreicht ist, wird dies der Bodenstation durch ein `mavlink_mission_item_reached_t`-Paket mitgeteilt.

Parameterprotokoll Mit dem Parameterprotokoll können verschiedenste Werte des Flugcontrollers [103] wie etwa Parameter von PID-Reglern, maximale Geschwindigkeiten, die Wertebereiche der Kanäle der Funkfernsteuerung oder die Belichtungszeit einer angeschlossenen Kamera konfiguriert werden.

Mit einem `mavlink_param_request_list_t`-Paket wird der Flugcontroller angewiesen, alle Parameter zu übertragen. Dazu werden `mavlink_param_value_t`-Pakete verwendet, in denen die Gesamtzahl der Parameter und Name, Index, Typ und Wert des jeweiligen Parameters enthalten sind. Nachdem die Liste übertragen wurde, kann ein Parameter durch ein `mavlink_param_set_t`-Paket überschrieben werden, das ihn durch seinen Index oder Namen referenziert. Einzelne Parameter können

auch durch `mavlink_param_request_read_t`-Pakete erneut gelesen werden, die ebenfalls Index oder Namen zur Referenzierung verwenden. Veranschaulicht werden diese Vorgänge noch einmal mit Sequenzdiagrammen in Abbildung 3.11.



(a) Lesen aller Parameter **(b)** Schreiben eines Parameters

Abbildung 3.11.: MAVLink-Parameterprotokoll [104]
Sequenzdiagramme zur Veranschaulichung des Lesens und Schreibens von Parametern eines Flugcontrollers (*MAV Component*) über das MAVLink-Parameterprotokoll von einer Bodenstation aus (*QGroundControl*).

Data Streams Ein Datenstrom (*Data Stream*) setzt sich aus verschiedenen Pakettypen zusammen, die vom Flugcontroller mit einer festgelegten Datenrate gesendet werden. Zur Konfiguration wird ein `mavlink_request_data_stream_t`-Paket gesendet, das neben der Empfängeradresse die Datenstrom-ID, gewünschte Frequenz und ein Flag zum Ein- und Ausschalten des Streams enthält.

3.3.1.2. Codegenerierung

Wie in Abschnitt 3.3.1.1 auf Seite 32 beschrieben, wird automatisiert C-Programmcode zum Kodieren, Dekodieren und Zugriff auf die einzelnen Felder für die in einer XML-Datei spezifizierten Nachrichtentypen erzeugt. Da die Programmiersprache C keine Möglichkeiten zur Überladung von Funktionen bietet, enthalten etwa die Namen der Kodierungsfunktionen den entsprechenden Pakettypen zur Differenzierung. Daraus ergibt sich der Nachteil, dass es nicht möglich ist, template-basierten Code zu schreiben, der den Parametern entsprechend die richtige Funktion auswählt. Deswegen wurde der in Python verfasste C-Codegenerator so erweitert, dass zusätzliche C++-Funktionen generiert werden.

Kodierung Die Funktion `mavlink_message_t mavlink_msg_encode(...)` wird für alle Pakettypen zum Kodieren der Nachrichten überladen. Intern ruft sie hierzu die entsprechende C-Funktion auf, die am Beispiel des *Heartbeat*-Pakets folgende wäre: `uint16_t mavlink_msg_heartbeat_encode_chan(...)`.

Dekodierung Da zum Dekodieren einer `mavlink_message_t`-Nachricht die Funktion nicht einfach überladen und so anhand ihrer Parameter ausgewählt werden kann, wird ein anderer Ansatz verfolgt. Dazu wird das Funkions-Template

```
template<typename MAVLINK_MSG>
MAVLINK_MSG mavlink_msg_decode(const mavlink_message_t&);
```

für sämtliche Nachrichtentypen spezialisiert. Für das exemplarische *Heartbeat*-Paket wird somit

```
template<>
mavlink_heartbeat_t mavlink_msg_decode<>(
    const mavlink_message_t&);
```

generiert.

Hilfsfunktionen Die MAVLink-Pakettypen werden anhand eindeutiger IDs unterschieden, die die Struktur `mavlink_message_t` im Feld `msgid` speichert. Der C-Codegenerator erzeugt für jeden Typen ein Makro, das die ID enthält. Durch Spezialisierung des Funktions-Templates

```
template<typename MAVLINK_MSG>
constexpr uint8_t mavlink_message_id();

template<>
constexpr uint8_t mavlink_message_id<mavlink_heartbeat_t>() {
  return MAVLINK_MSG_ID_HEARTBEAT;
}
```

kann der Wert nun auch in Template-Code verwendet werden.

Ausgabe Zur Visualisierung der MAVLink-Pakete wurde außerdem der Ausgabeoperator für sämtliche Typen überladen:

```
std::ostream& operator<<(std::ostream&,
                         const mavlink_heartbeat_t&);
```

Dieser gibt die Namen der einzelnen Felder und deren Inhalt aus. Einige Felder enthalten dabei Werte aus Aufzählungstypen. Da diese Zuordnung jedoch nicht in der XML-Paketspezifikation festgehalten ist, können die Namen der numerischen Werte nicht vollständig automatisiert ausgegeben werden. Zur Abhilfe wurde diese Zuordnung manuell im C++-Codegenerator festgehalten. Gleiches gilt für ein einziges Bitfeld, für dessen Ausgabe jedoch direkt manuell C++-Code geschrieben wurde.

Zudem wird eine allgemeine Variante für `mavlink_message_t`-Pakete generiert, die die Headerinformationen ausgibt, das Paket der ID entsprechend dekodiert und dann dessen Ausgabeoperator verwendet.

3.3.2. Berlin United Framework

Das Berlin United Framework [105] ist ein modulares in C++ verfasstes Softwarepaket, das zur Implementierung autonomer Agenten konzipiert wurde. Durch die starke Modularisierung werden Wiederverwendbarkeit und Testbarkeit von Code begünstigt [106], wodurch es sich hervorragend für das Projekt *Neurocopter* eignet. Entwickelt wurde es in Zusammenarbeit der FUmanoids, dem Team humanoider Fußballroboter der Freien Universität Berlin, und dem NaoTeam der Humboldt-Universität zu Berlin [107] ausgehend von deren NaoTH-Framework.

3.3.2.1. Architektur

Die zugrunde liegende Struktur des Frameworks ist das Blackboard, in dem Daten zusammen abgelegt werden. Im Folgenden werden die weiteren Komponenten erläutert.

Repräsentationen Repräsentationen sind Objekte, die Daten speichern und keine weiteren komplexen Operationen bereitstellen [106].

Blackboard Ein Blackboard fasst mehrere Repräsentationen zusammen und bildet so eine Datenbasis, die zur Lösung komplexer Probleme herangezogen werden kann [106].

Module Die Module arbeiten auf den Repräsentationen eines Blackboards und beinhalten die Programmlogik. Dabei kann jeweils genau ein Modul eine Repräsentation für andere Module bereitstellen und erhält dafür schreibenden Zugriff, so dass es die Daten erzeugen kann. Diese können dann von beliebig vielen anderen Modulen ausschließlich lesend verwendet werden [106].

Modulmanager Ein Modulmanager führt seine Module sequentiell aus [106]. Dabei muss die Ausführungsreihenfolge so bestimmt werden, dass das eine Repräsentation bereitstellende Modul ausgeführt wurde, bevor diese von anderen verwendet wird [105]. Diese Abhängigkeiten der Module und Repräsentationen lassen sich durch Kanten eines gerichteten Graphen formulieren, der azyklisch ist, wenn eine gültige Reihenfolge existiert. Für verschiedene Funktionen gibt es unterschiedliche Arten von Modulmanagern: So kann die Ausführung der Module etwa durch ein Ereignis wie das Eintreffen eines neuen Kamerabildes oder zeitgesteuert mit einer festgelegten Frequenz veranlasst werden.

Services Die Ausführung der Modulmanager sowie weitere Funktionen, die sich nicht allein mit Modulen und Repräsentationen lösen lassen, werden als Dienste realisiert. Diese laufen als eigenständige Threads (Ausführungsstränge) und sind global verfügbar. Zu ihren Aufgaben zählt etwa das Warten auf ein neues Kamerabild und anschließendes Senden eines Signals, um die Ausführung eines Modulmanagers zu veranlassen, oder die Verwaltung der Netzwerkkommunikation [105].

Dieser Aufbau birgt mehrere Vorteile. Durch die Modularisierung und Kapselung der Daten in Repräsentationen wird die gemeinsame Arbeit an einem Projekt vereinfacht. Es bestehen keine direkten Abhängigkeiten zwischen den Modulen und einmal berechnete Daten stehen allen zur Verfügung. Durch die sequentielle Ausführung der Module muss bei der Entwicklung außerdem keinerlei Rücksicht auf Synchronisationsmechanismen für den Zugriff auf die Daten genommen werden.

Dieses Konzept eignet sich hervorragend für den *Neurocopter*, da sich auf diese Weise in Modulen realisierte Versuche zum einen unabhängig voneinander realisieren lassen, aber trotzdem der Zugriff auf eine gemeinsame Datenbasis möglich ist.

3.3.2.2. Konfiguration

Das Framework stellt eine Schnittstelle bereit, mit der verschiedenste Konfigurationsparameter hierarchisch in Sektionen unterteilt in einer Baumstruktur verwaltet werden können. Solch ein Parameter ist entweder eine Fließkommazahl, eine Ganzzahl, eine Zeichenkette oder ein auf diesen Typen basierender neuer Typ, wie viele aus der Boost.Units-Bibliothek. Er verfügt über eine optionale Beschreibung und einen Standardwert. Zugegriffen werden kann auf den Parameter über seinen eindeutigen Namen, der das entsprechende Blatt in der Baumstruktur identifiziert. Zur Unterteilung der Sektionen werden Punkte im Namen verwendet.

Hinzugefügt werden können Konfigurationsparameter über die Klasse `ConfigRegistry`. Soll beispielsweise die Parametergruppe `address` mit den Parametern `ip` und `port` hinzugefügt werden, so wird dafür folgender Programmcode verwendet:

```
auto cfg_ip   = ConfigRegistry::registerOption<std::string>(
        "address.ip",  "Standardwert", "Beschreibung");
auto cfg_port = ConfigRegistry::registerOption<uint16_t>(
        "address.port",         12345, "Beschreibung");
```

Ausgelesen werden kann der Parameter `ip` dann mit der Zeile:

```
std::string ip = cfg_ip->get();
```

Alternativ kann auch über den Namen auf den Parameter zugegriffen werden:

```
std::string ip =
        services.getConfig().get<std::string>("address.ip");
```

Für alle registrierten Parameter werden außerdem automatisch Kommandozeilenoptionen zum Setzen beim Starten des Programms erzeugt.

3.3.2.3. Testumgebung zur Fehlersuche

Zum Testen der Module stellt das Framework eine Reihe von Funktionen zur Verfügung. So können Codesegmente zur Laufzeit deaktiviert, Variablen angezeigt und

modifiziert, Statusinformationen gesendet und Kamerabilder, in die mit speziellen Funktionen gezeichnet wurde, übertragen werden [106]. Die Klasse `Debugging` stellt ähnlich wie die `ConfigRegistry` (vgl. Abschnitt 3.3.2.2 auf Seite 38) Funktionen bereit, über die bestimmte Optionen zur Analyse und Fehlerfindung registriert werden können. Solch eine Option kann zur Laufzeit an- und ausgeschaltet werden und überträgt beispielsweise Zeitmessungen, Text oder Linien, die ins Kamerabild gezeichnet werden können.

Der Zugriff von außerhalb auf diese Funktionen ist entweder direkt über *telnet*[2] oder spezielle grafische Software wie FUremote möglich.

FUremote FUremote [108] ist ein grafisches plattformunabhängiges erweiterungsfähiges Programm, das auf dem *Eclipse Rich Client Platform (RCP)* Framework basiert [109]. Mit ihm kann anstelle von *telnet* auf Funktionen des Berlin United Frameworks zugegriffen werden. Es bietet interaktive Ansichten zum Editieren von Variablen, zeigt gesendete Statusinformationen und Kamerabilder an, kann Module ein- und ausschalten und stellt viele weitere, teils Fußballroboter-spezifische, Ansichten und Optionen bereit.

3.3.3. ar2clipse

Bei genauerer Betrachtung stellte sich heraus, dass die MAVLink-Dokumentation sehr knapp gehalten ist. Daher wurde es an mancher Stelle nötig, den ArduCopter-Quelltext zu lesen, um etwa den Inhalt bestimmter Pakete zu verstehen. So enthält beispielsweise das `mavlink_raw_imu_t`-Paket die Accelerometer-, Gyroskop- und Magnetometerwerte, deren Einheiten jedoch nicht angegeben sind. Auch ist an keiner Stelle festhalten, aus welchen Pakettypen sich die MAVLink-Datenströme (*data streams*, vgl. Abschnitt 3.3.1.1 auf Seite 35) zusammensetzen.

Der ArduCopter-Code wurde ursprünglich für das APM1 mit der Arduino IDE (Integrierte Entwicklungsumgebung) entwickelt [110], die nur die nötigsten Funktionen bereitstellt und weder eine syntaktische noch semantische Analyse des Quelltexts beherrscht [111]. Dann wurde zu Gunsten der Unterstützung weiterer Hardwareplattformen, insbesondere des PX4 [112], in der Software eine Schicht zur Abstraktion der Hardware hinzugefügt [113]. Diese verwendet nicht mehr die Arduino-Laufzeitbibliothek, wodurch die Kompatibilität mit der Arduino IDE aufgegeben wurde [110]. Stattdessen wird nun auf das Build-Management-Tool *make* [114] oder eine modifizierte Variante der Arduino IDE [110] zurückgegriffen.

Trotzdem wird weiterhin am Arduino-Erstellungsprozess (*build process*) festgehalten, der einen eigenen C++-Dialekt verwendet, der einen Zwischenschritt mit automatisierter Codeerzeugung benötigt, bevor der Quelltext von einem regulären C++-Compiler verarbeitet werden kann [115]. Dabei werden die Arduino-Sketch-Dateien

[2] *telnet* ist ein Programm, das das TELNET-Protokoll zur interaktiven zeichenorientierten Kommunikation über TCP-Verbindungen nutzt.

[116] zu einer einzigen großen Datei zusammengefügt, `#include`-Direktiven eingefügt und Funktionsprototypen erzeugt. Außerdem werden dem Linker die Verzeichnisse der verwendeten Arduino-Bibliotheken einzeln als Suchpfade übergeben [117]. Dies ist nötig, da eine Arduino-Bibliothek aus einem Verzeichnis besteht, in dem sich eine gleichnamige Header-Datei und Quelltextdateien befinden [111]. Zum Verwenden der Bibliothek wird aber nur die Header-Datei ohne das vorangestellte gleichnamige Verzeichnis in der `#include`-Direktive angegeben [111].

Durch diesen sehr eigenen Prozess ist es nicht möglich, den Quelltext in einer vollwertigen Entwicklungsumgebungen wie *Eclipse* zu untersuchen, da deren semantische Codeanalyse am Arduino-Dialekt scheitert.

Zur Lösung dieser Probleme wurde das Python-Programm *ar2clipse* entwickelt, das die Einstellungen eines Eclipse-Projekts anpasst, Dateien durch symbolische Verknüpfungen einbindet und ein Minimum an Quelltext generiert.

3.3.3.1. Funktionsweise

Das Kommandozeilenprogramm setzt ein mit dem *AVR Eclipse Plug-in* erzeugtes *Eclipse CDT*-Projekt mit einigen manuell vorgenommenen Grundeinstellungen voraus und modifiziert dieses dann so, dass der ArduCopter-Code bearbeitet und kompiliert werden kann. Eingestellt werden müssen zum alleinigen Bearbeiten lediglich der AVR-Prozessortyp des APM und einige wenige Makros mit vom Quelltext verwendeten Konstanten. Soll der Code auch kompiliert werden, so müssen zusätzlich noch die Compiler- und Linkerparameter angegeben werden. Nach dieser Konfiguration führt das Skript bei jedem Aufruf die im Folgenden beschriebenen Schritte aus, um das Projekt anzupassen:

- Um die große C++-Datei zu erhalten, die aus den zusammengefügten Arduino-Sketch-Dateien und Funktionsprototypen besteht, wird der *make*-Prozess des ArduCopter-Projekts angestoßen.
- Alle Bibliotheksunterverzeichnisse des ArduCopter-Projekts werden dem Eclipse-Projekt durch symbolische Verknüpfungen auf Dateisystemebene hinzugefügt.
- Genauso werden alle Sketch-Dateien verknüpft.
- In der großen C++-Datei wird der Inhalt sämtlicher Sketch-Dateien durch `#include`-Direktiven der entsprechenden Verknüpfungen auf diese Dateien ersetzt. Dies ist der einzige generierte Code, der dem Eclipse-Projekt hinzugefügt wird.
- Nun werden die Projekteinstellungen bearbeitet, indem alle verwendeten Bibliotheksverzeichnisse zu den Suchpfaden hinzugefügt werden.
- Außerdem werden die nicht verwendeten Bibliotheken und eventuelle Code-Beispiele der Bibliotheken in den Projekteinstellungen exkludiert.

Bearbeitung der Eclipse-Projekteinstellungen Die Einstellungen eines *Eclipse CDT*-Projekts werden in der XML-Datei `.cproject` im Projektverzeichnis festgehalten. Zum Bearbeiten wird das Python-Modul `xml.etree.ElementTree` verwendet, mit dem auf die Elemente der Baumstruktur des Dokuments zugegriffen werden kann. Dabei wurden für die wichtigsten Knoten, auf denen mehrere Operationen nötig sind, Python-Klassen zur Abstraktion verwendet, die die zahlreichen Funktionen kapseln.

Ein Projekt kann über mehrere Konfigurationen (`<cconfiguration>`) wie etwa `Debug` oder `Release` verfügen, die die Einstellungen für unterschiedliche Erstellungsprozesse verwalten. Eine Konfigurationen verfügt wiederum über *Toolchains* (`<toolChain>`), die jeweils die einzelnen Werkzeuge (`<tool>`) wie Compiler, Linker und Assembler für einen bestimmten Erstellungsprozess bestimmen.

Um die Suchpfade wie in Abschnitt 3.3.3.1 auf Seite 40 beschriebenen anzupassen, wird bei jeder Konfigurationen des manuell erstellen Projekts in der `AVR-GCC Toolchain` in den Werkzeugen `AVR Assembler`, `AVR Compiler` und `AVR C++ Compiler` der Knoten `<option name="includePath">` entsprechend modifiziert oder auch erstellt, sollte er nicht bereits existieren. Die zu exkludierenden Verzeichnisse werden hingegen direkt in den Konfigurationen als Kind des `<sourceEntries>` Knotens angegeben: `<entry excluding="..."/>`.

3.3.4. Bordsoftware

Nachdem in den vorangegangenen Unterabschnitten 3.3.1, 3.3.2 und 3.3.3 verwendete, erweiterte und eigens entwickelte Werkzeuge vorgestellt wurden, befasst sich dieser Unterabschnitt nun mit der Implementierung der Bordsoftware des *Neurocopters*, die versucht, die in Abschnitt 1.2.2 auf Seite 5 gestellten Anforderungen umzusetzen. Dabei geht es in erster Linie um die Bereitstellung eines Frameworks, das den Zugriff auf die Sensordaten des Flugcontrollers ermöglicht und die Kommunikation mit der Bodenstation regelt, um so eine Umgebung zur Durchführung und Beobachtung von Experimenten rund um die Erforschung von Honigbienen zu schaffen.

Als Basis der Bordsoftware wird das *Berlin United Framework* (vgl. Abschnitt 3.3.2 auf Seite 37) mit zwei Modulmanagern verwendet. Einer davon dient der Verarbeitung von Kamerabildern und stößt die Ausführung seiner Module beim Eintreffen eines neuen Bildes an. Der andere Modulmanager ist hingegen zeitlich gesteuert und dient der Verarbeitung sämtlicher MAVLink-Pakete. Diese werden kontinuierlich von einem im Hintergrund laufenden Service empfangen und bis zur Ausführung der Module in einem Puffer zwischengespeichert.

3.3.4.1. Datenaustausch zwischen den Modulmanagern

Da die beiden Modulmanager nebenläufig ausgeführt werden, ist ein direkter Zugriff auf die Repräsentationen des Blackboards des jeweils anderen Managers ohne Synchronisationsmechanismen nicht möglich. Dies kann jedoch nötig sein, wenn beispielsweise zur Bildverarbeitung die Lage des Copters herangezogen werden soll. Um den Zugriff dennoch zu ermöglichen, wird jedem Manager hierfür ein weiteres Modul hinzugefügt und mit dem *Berlin United Framework* ein neues *Event* registriert, das zu deren Kommunikation verwendet wird. Dabei löst das Modul im MAVLink-Modulmanager das *Event* aus, das einen Zeiger auf die benötigten Daten enthält und synchron verarbeitet wird. Um darauf reagieren zu können, implementiert das Modul im anderen Manager eine zusätzliche Schnittstelle. In deren Rückruffunktion (*callback function*) werden die Daten dann kopiert und vorerst im Modul zwischengespeichert, um anschließend bei seiner nächsten Ausführung in die von ihm bereitgestellte Repräsentation kopiert zu werden, auf die die anderen Module regulär zugreifen können.

3.3.4.2. Manipulation verschiedener Daten von MAVLink-Geräten

Viele Operationen müssen für alle angeschlossenen, per MAVLink kommunizierenden Geräte gleichermaßen ausgeführt werden. Um Codeduplikate zu minimieren und das Entfernen oder Hinzufügen eines neuen Geräts so einfach wie möglich zu gestalten, wurde die template-basierte Struktur `device_data` konzipiert, die mehrere Instanzen einer Klasse enthält und beliebige Operationen darauf ausführen kann:

```
template<typename T>
struct device_data {
  T gnd;
  struct copter_t {
    T apm;
    T px4flow;
    template<typename F, typename... U>
    void exec(const F &f, U&... u);
  } copter;
  template<typename F, typename... U>
  void exec(const F &f, U&... u);
}
```

Derzeit sind Verbindungen mit einer Bodenstation (`gnd`) und auf dem Copter selbst mit dem Flugcontroller (`apm`) und einem Sensor zur Bestimmung des optischen Flusses (`px4flow`) vorgesehen. Um eine Operation für jedes dieser drei Geräte auszuführen, wird die Methode `exec` verwendet, die als Parameter einen Funktor und beliebig viele `device_data`-Objekte erhält. Der Funktor wird dann für jedes der drei MAVLink-Geräte mit dem Gerät selbst und den entsprechenden Feldern aus den weiteren Parametern aufgerufen. Für die Bodenstation wird dementsprechend

`f(this->gnd, u.gnd...);` ausgeführt. Sollte eine Operation nur für die Geräte auf dem Copter ausgeführt werden, kann die Methode `exec` des geschachtelten Objekts `copter_t` verwendet werden.

3.3.4.3. MAVLink-Service

Der MAVLink-Service läuft im Hintergrund und empfängt kontinuierlich die Nachrichten aller angeschlossenen MAVLink-Geräten.

Abstraktion des Übertragungskanals Da MAVLink-Pakete über beliebige Kanäle wie serielle Schnittstellen oder TCP-Verbindungen übertragen werden können, wurde die abstrakte Klasse `abstract_read_write` mit Schnittstellen zum Lesen und Schreiben verwendet, von der die konkreten Implementierungen erben, um die verschiedenen Übertragungsmöglichkeiten zu abstrahieren. Diese Struktur wird in Abbildung 3.12 in vereinfachter Weise dargestellt.

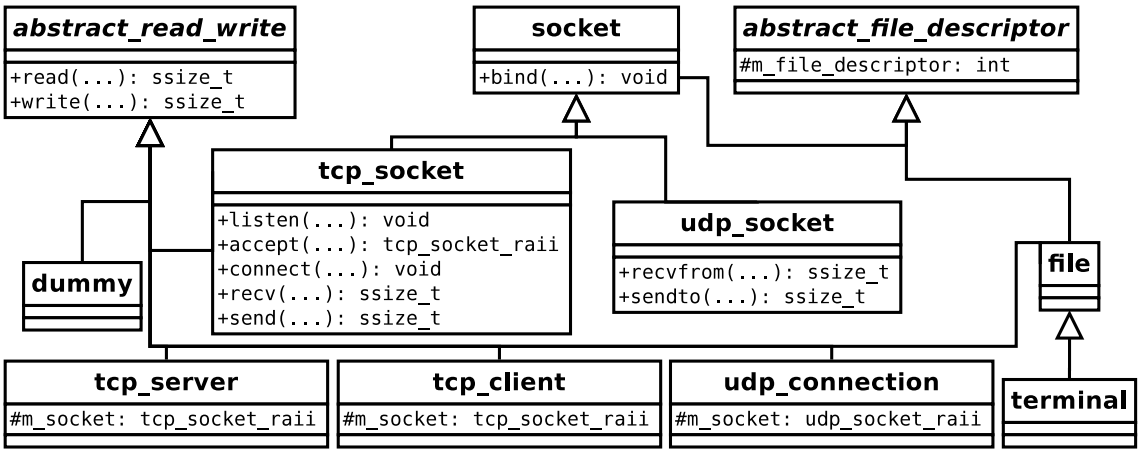


Abbildung 3.12.: Klassendiagramm der abstrahierten Übertragungsmöglichkeiten
Zur Abstraktion der Übertragung von MAVLink-Paketen wird von der abstrakten Klasse abstract_read_write geerbt, so dass TCP-Verbindungen (tcp_server und tcp_client), UDP-Übertragungen (udp_connection) und serielle Schnittstellen (terminal) gleichermaßen genutzt werden können. Die Klasse dummy kann als Platzhalter verwendet werden, wenn keine Verbindung verwendet werden soll.

Senden und Empfangen von MAVLink-Paketen In Abschnitt 3.3.1.1 auf Seite 32 wurde bereits erläutert, dass zum Senden und Empfangen von MAVLink-Paketen den entsprechenden Funktionen eine Kanalnummer übergeben werden muss, mit der auf Variablen wie die aktuelle Sequenznummer oder den Parsezustand des Kanals zugegriffen wird. Diese Funktionalität wurde in der Klasse `mavlink_send_reveive` abstrahiert, die ein `abstract_read_write`-Objekt kapselt und so mit einer festgelegten Kanalnummer Funktionen zum Senden und Empfangen bereitstellt. Zum Senden

wird eine template-basierte Methode verwendet, die auf die automatisch generierten Sendefunktionen zurückgreift (vgl. Abschnitt 3.3.1.2 auf Seite 35). Beim Parsen einer eingehenden Nachricht müssen die Daten solange byteweise einer MAVLink-Funktion übergeben werden, bis ein vollständiges Paket erkannt wurde. Dies ist durch eine Schleife realisiert worden, die vom `abstract_read_write`-Objekt liest.

Da die Methode zum Empfangen folglich solange blockiert, bis eine komplette Nachricht erhalten wurde, wird dieser Vorgang nebenläufig ausgeführt. Dies geschieht in der Klasse `mavlink_channel`, die intern das zuvor beschriebene `mavlink_send_reveive`-Objekt verwendet. Die erhaltenen Nachrichten werden zusammen mit ihrem Empfangszeitpunkt solange in einem `std::vector` zwischenspeichert, bis sie über die Methode `std::vector<timestamped_mavlink_message_t> get_received_messages()` abgeholt werden. Über eine im *Berlin United Framework* registrierte Debug-Option können die erhaltenen Nachrichtentypen optional angezeigt werden. Außerdem stellt die Klasse eine Methode zur Verfügung, die überprüft, ob sich unter den aktuell erhaltenen Nachrichten ein *Heartbeat*-Paket befindet. Dies kann verwendet werden, um zu warten, bis ein Gerät betriebsbereit ist. Zu sendende Nachrichten werden direkt an das `mavlink_send_reveive`-Objekt weitergereicht.

Der eigentliche Service Durch die Kapselung der gesamten benötigten Funktionalität (vgl. Abschnitt 3.3.4.3 auf Seite 43) besteht die eigentliche Aufgabe des Dienstes `mavlink_communication` nur noch in der Erzeugung von `mavlink_channel`-Objekten für die verbundenen Geräte. Zur Verwaltung der verschiedenen Verbindungen wird ein `device_data<std::unique_ptr<mavlink_channel>>`-Objekt verwendet (vgl. Abschnitt 3.3.4.2 auf Seite 42).

Um die Verbindungsarten der einzelnen Geräte frei konfigurieren zu können, wurde für jeden Verbindungstyp eine Klasse konzipiert, die im *Berlin United Framework* entsprechende Konfigurationsparameter (vgl. Abschnitt 3.3.2.2 auf Seite 38) in einer Sektion einträgt. Diese Klassen werden wiederum in der Klasse `connection` zusammengefasst, die zusätzlich einen Konfigurationsparameter für den verwendeten Verbindungstyp enthält. Anhand dessen kann dann die entsprechende Klasse instanziiert und über die gemeinsame Schnittstelle `abstract_read_write` zurückgegeben werden.

Da diese Verbindungsoptionen für jedes der MAVLink-Geräte benötigt werden, wird auf die template-basierte Struktur `device_data<connection>` zur Zusammenfassung zurückgegriffen.

3.3.4.4. Bereitstellung empfangener MAVLink-Pakete

Für den Zugriff auf die Sensordaten und andere Statusinformationen des Flugcontrollers ist die MAVLink-Kommunikation die zentrale Komponente. Daher müssen die mithilfe des MAVLink-Services (vgl. Abschnitt 3.3.4.3 auf Seite 43) empfangenen Nachrichten allen Modulen in geeigneter Form zur Verfügung gestellt werden.

Repräsentation Die Repräsentation verwaltet alle MAVLink-Nachrichten, die seit der letzten Ausführung des Modul-Managers empfangen wurden. Dabei wird wieder auf die template-basierte Struktur `device_data` zurückgegriffen, die für jedes MAVLink-Gerät ein eigenes `mavlink_message_container`-Objekt enthält, das nun näher erläutert wird. Zum einen enthält es die vom MAVLink-Service erzeugte Liste (`std::vector<timestamped_mavlink_message_t>`), in der sich die Nachrichten zeitlich sortiert zusammen mit ihren Empfangszeitstempeln befinden. Außerdem wird ein assoziatives Datenfeld (`std::map`) angelegt, in dem die Nachrichten nach ihren IDs abgelegt werden. Die gespeicherten Werte setzen sich dabei aus je zwei Listen (`std::vector`) zusammen. Eine davon referenziert lediglich die kodierten Nachrichten in der Liste aller Nachrichten, während die andere die dekodierten Nachrichten, wie etwa das `mavlink_heartbeat_t`-Paket, enthält. Hierbei wird die *Lazy Evaluation*-Auswertungsstrategie verfolgt: Die Dekodierung wird bis zum ersten Zugriff auf die Liste verzögert, die bis dahin leer ist. Dieser Vorgang findet in einer template-basierten Methode statt, die den korrekten Rückgabetyp hat, so dass das Typsystem von C++ beim Zugriff nicht untergraben werden muss. Dies wird nur innerhalb der Methode mit einem einzigen `static_cast` getan, um die Listen verschiedener Typen zusammen in dem assoziativen Datenfeld speichern zu können.

Durch die *Lazy Evaluation*-Auswertungsstrategie ist es außerdem ohne zusätzlichen Aufwand möglich festzustellen, für welche MAVLink-Pakettypen auf die dekodierten Nachrichten zugegriffen wurde. Dafür müssen nur die Längen der beiden Listen des Pakettyps im Datenfeld verglichen werden. Dadurch ist es beispielsweise möglich auf einfache Weise festzustellen, ob ein angeschlossenes MAVLink-Gerät etwa nach einem Softwareupdate neue Nachrichtentypen versendet.

Modul Das Modul zur Bereitstellung der empfangenen Nachrichten kann nun wegen der Verwendung der Struktur `device_data` sowohl im MAVLink-Service als auch in der Repräsentation auf deren Methode `exec` zugreifen, um die Nachrichten aller MAVLink-Geräte zu aktualisieren:

```
this->getreceived_mavlink_messages().exec(
  [](mavlink_message_container &lhs,
     std::unique_ptr<mavlink_channel> &rhs)
  { lhs.update(*rhs); },
  services.getMavlink_communication().get_channels()
);
```

Dabei wird die Methode `update` jedes `mavlink_message_container`-Objekts mit dem entsprechenden `mavlink_channel`-Objekt aufgerufen.

3.3.4.5. Weitere MAVLink-Module und Repräsentationen

Betriebsbereitschaft (Heartbeat) Das beispielhaft schon oft herangezogene *Heartbeat*-Paket signalisiert die Betriebsbereitschaft eines MAVLink-Geräts. Die Boden-

stationssoftware APM Planner (vgl. Abschnitt 2.2.3 auf Seite 14) fügt beispielsweise ein Gerät automatisch zur Ansicht hinzu, sobald ein *Heartbeat*-Paket von ihm empfangen wurde. Sollte es dann wieder für eine bestimmte Zeitspanne ausbleiben, wird dies signalisiert.

Zum Senden dieser Pakete wird ein Modul verwendet, das auf die Repräsentation der empfangenen Nachrichten des Flugcontrollers zugreift und jeweils das aktuellste *Heartbeat*-Paket an die Bodenstation weiterleitet.

Data Streams Ein MAVLink-Data-Stream (vgl. Abschnitt 3.3.1.1 auf Seite 35) setzt sich mehreren Pakettypen zusammen, die kontinuierlich gesendet werden. Für einige Streams sind die enthalten Pakete zwar spezifiziert, bei anderen werden sie aber frei von der Implementierung des Flugcontrollers bestimmt [118]. An dieser Stelle wurden die Typen mangels Dokumentation per Codeanalyse mit *ar2clipse* bestimmt, wobei sich herausstellte, dass auch die eigentlich vorgegebenen Typen von der Spezifikation abweichen. Dabei fiel außerdem auf, dass der Wert zur Bestimmung der Senderate im `mavlink_request_data_stream_t`-Paket vom ArduCopter-Code anders als erwartet interpretiert wird. Die Dokumentation des `req_message_rate` genannten Parameters besagt, dass er das Intervall zwischen zwei Übertragungen angibt: „The requested interval between two messages of this type“ [119]. Stattdessen wird der Wert vom ArduCopter-Code aber als Frequenz in Hertz interpretiert.

Um den Zugriff auf die Pakete typsicher zu gestalten, wurde für die Implementierung der Repräsentation zur Zusammenfassung der Typen eines Streams das Klassen-Template `std::tuple` gewählt, das eine feste Anzahl verschiedener Typen zusammenlegt. Für den Strom einer einzelnen Nachricht wird die template-basierte Klasse `template<typename TYPE> class message_stream` verwendet. Diese kann aus einem `mavlink_message_container`-Objekt, der Repräsentation der erhaltenen MAVLink-Pakete (vgl. Abschnitt 3.3.4.4 auf Seite 44), die Liste der entsprechenden Pakete auswählen und daraus die durchschnittliche Nachrichtenrate bestimmen. Außerdem können diese Pakete über ein `mavlink_channel`-Objekt, das das Senden und Empfangen von Nachrichten kapselt (vgl. Abschnitt 3.3.4.3 auf Seite 43), weitergeleitet werden.

Auflistung 3.3: Datenstrom aus mehreren MAVLink-Pakettypen

```
typedef data_stream<
  MAV_DATA_STREAM_RAW_SENSORS,
  mavlink_raw_imu_t,
  mavlink_scaled_pressure_t,
  mavlink_sensor_offsets_t
> apm_mavlink_data_stream_raw_sensors;
```

Die einzelnen Nachrichtenströme werden nun mit dem Klassen-Template `template<enum MAV_DATA_STREAM ID, typename... TYPES> class data_stream` unter Verwendung des bereits erwähnten Klassen-Templates `std::tuple` zu einem Da-

tenstrom zusammengefasst. Auflistung 3.3 zeigt dies exemplarisch für einen Datenstrom. Hier werden außerdem die gewünschte Datenrate und der Aktivierungszustand des Streams gespeichert. Über Objektmethoden kann die Frequenz geändert und abgefragt werden. Dabei stehen Minimum, Maximum und Durchschnitt der einzelnen Nachrichtenströme zur Verfügung. Um die Methoden der einzelnen Nachrichtenströme zum Weiterleiten und Ermitteln der Rate sinnvoll nutzen zu können, werden hierfür Funktionen bereitgestellt, die auf jedem Element des intern verwendeten `std::tuple`-Objekts arbeiten. Für den Zugriff auf die einzelnen Tupelelemente anhand ihres Typs wird eine zur Kompilierzeit auswertbare Funktion verwendet, die den entsprechenden Index bestimmt, über den dann auf den Tupel zugegriffen werden kann:

```
template<typename T>
constexpr std::size_t get_index_of() {
  return 0;
}
template<typename T, typename HEAD, typename... TAIL>
constexpr std::size_t get_index_of() {
  return std::is_same<T, HEAD>::value
    ? 0
    : 1 + get_index_of<T, TAIL...>();
}
```

Dabei werden die Template-Parameter `HEAD, TAIL...` rekursiv bis zum ersten Auftreffen des Typen `T` durchsucht, dessen Index dann zurückgegeben wird.

Sämtliche Data Streams werden mit einem weiteren variadischen Klassen-Template zusammenfasst. Dadurch kann eine Operation gemeinsam auf allen Datenströmen ausgeführt werden, die diese wiederum an all ihren Nachrichtenströmen vornehmen. Auf diese Weise können die Datenraten aktualisiert, Stream-Anforderungen bearbeitet und entsprechende Pakete weitergeleitet werden.

Durch diese aufwändige Repräsentation beschränken sich die Aufgaben des Data-Stream-Moduls lediglich auf den Aufruf von Methoden, denen andere Repräsentationen übergeben werden, so dass etwa auf die erhaltenen MAVLink-Nachrichten zugegriffen werden kann. Auf diese Weise werden Datenströme aktiviert, deren Rate geändert und Anfragen der Bodenstation beantwortet.

Parameterprotokoll Um Konfigurationsparameter des mit dem Bordcomputer verbundenen Flugcontrollers von der Bodenstation aus auslesen zu können, wurde das MAVLink-Parameterprotokoll (vgl. Abschnitt 3.3.1.1 auf Seite 34) implementiert. Alternativ wäre es auch möglich gewesen, sämtliche Pakete des Protokolls direkt an den Flugcontroller weiterzuleiten. Dieser Ansatz wurde jedoch nicht gewählt, da es so beispielsweise möglich wäre, Einstellungen im Flugcontroller vorzunehmen, die den Anforderungen von Modulen der Bordsoftware widersprechen. So ist es etwa

möglich, Datenströme abzuschalten, da deren Datenrate direkt über einen Konfigurationsparameter zugänglich ist. Außerdem ist die Implementierung ohnehin nötig, sobald die Bordsoftware Parameter lesen oder sogar schreiben soll.

Die vorgenommene Implementierung fordert die Parameter aller MAVLink-Geräte an und speichert sie mit der Struktur `device_data` in entsprechenden `param_map`-Repräsentationen. Anfragen der Bodenstation werden dann mit diesen gepufferten Werten beantwortet, so dass die MAVLink-Geräte entlastet werden. Die Klasse `param_map` legt die Parameter in einem assoziativen Datenfeld (`std::map`) ab, so dass sie über ihren Namen zugänglich sind. Zusätzlich wird eine Liste (`std::vector`) mit Zeigern auf diese Werte verwaltet, so dass auch ein Zugriff über den Index möglich ist.

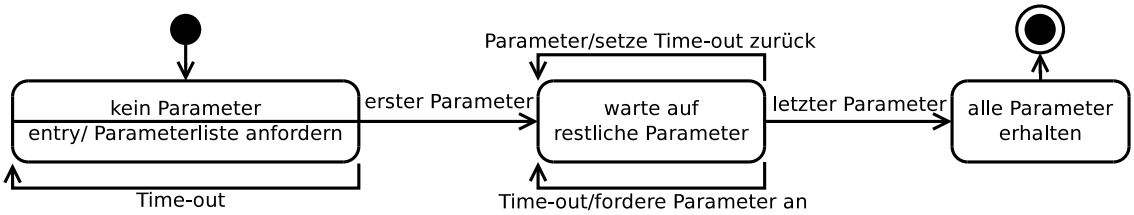


Abbildung 3.13.: Zustandsdiagramm zur Anforderung der Parameter

Dieses Zustandsdiagramm zeigt den allgemeinen Ablauf zur Anforderung der Parameterliste eines MAVLink-Geräts. Solange noch nicht alle Parameter erhalten wurden, werden alle oder einzelne fehlende angefordert.

Das vom Manager regelmäßig ausgeführte Parameterprotokollmodul fordert über entsprechende Methoden der Repräsentation zuerst alle Parameter an. Über Timeouts geregelt werden solange entweder erneut alle oder einzelne fehlende Parameter angefordert, bis die gesamte Liste erhalten wurde. Dies wird im zugehörigen Zustandsdiagramm in Abbildung 3.13 veranschaulicht. Außerdem behandelt das Modul die Parameteranfragen der Bodenstation, sobald sämtliche Parameter des Flugcontrollers verfügbar sind. Dabei wird sowohl auf einzelne namentlich oder per Index angefragte Parameter als auch auf die Anforderung der gesamten Liste reagiert. Letztere wird nicht auf einmal, sondern parameterweise mit jeder Ausführung des Moduls übertragen, um dessen Ausführungszeit konstant zu halten.

Um sicherzustellen, dass das Modul die Parameter gemäß der Implementierung des ArduCopter-Codes anfragt, wurde dieser unter Zuhilfenahme von *ar2clipse* analysiert. Dabei zeigte sich, dass das Feld `param_value` des zur Übertragung genutzten `mavlink_param_value_t`-Pakets nicht der Dokumentation entsprechend genutzt wird. Eigentlich sollten die vier Byte des `float`-Werts dem Feld `param_type` entsprechend interpretiert werden, da sie auch zur Übertragung von Nicht-Fließkommazahlen genutzt werden. Stattdessen wird der zu übertragende Wert in eine Fließkommazahl konvertiert, wodurch er sich ändern und an Genauigkeit verlieren kann. Dies spielt im Allgemeinen keine große Rolle, da die meisten übertragenen ganzzahligen Werte eher klein sind und somit verlustfrei in eine Fließkommazahl und wieder zurück

konvertiert werden können. Dennoch muss diese vom Protokoll abweichende Art der Übertragung der Gegenseite bekannt sein, da sonst falsche Werte gesendet werden würden. Aus diesem Grund stellte sich ArduCopter auch als inkompatibel zur Bodenstation QGroundControl (vgl. Abschnitt 2.2.3 auf Seite 14) heraus, da diese das Protokoll gemäß der Spezifikation implementiert.

3.3.5. Bodenstation

Als Bodenstationssoftware werden die MAVLink-Bodenstation APM Planner (siehe Abschnitt 2.2.3 auf Seite 14) und FUremote des *Berlin United Frameworks* (siehe Abschnitt 3.3.2.3 auf Seite 39) zusammen eingesetzt. APM Planner eignet sich gut zur Anzeige der weitergeleiteten Parameter und Datenströme des Flugcontrollers und zur Konfiguration des Copters, da es über die reine MAVLink-Kommunikation hinaus speziell auf den ArduCopter-Code abgestimmt ist und eigene Menüs für die einzelnen Konfigurationsoptionen bietet. Außerdem wurde die von der Spezifikation abweichende Implementierung einiger MAVLink-Protokollbestandteile des ArduCopter-Codes in APM Planner gleichermaßen umgesetzt, so dass keine Kompatibilitätsprobleme wie mit anderen Bodenstationen bestehen (vgl. Abschnitt 3.3.4.5 auf Seite 47).

Modulspezifische Einstellungen und Datenübertragungen werden hingegen gesondert mit FUremote vorgenommen, da eine Integration der beiden Programme sehr aufwändig erscheint und keine erheblichen Vorteile birgt.

3.3.6. Konfiguration von Flugrouten und Verhalten

Um eine bessere Reproduzierbarkeit der Experimente zu gewährleisten, wurde gefordert, dass der *Neurocopter* die Möglichkeit bietet, vorkonfigurierte Routen abzufliegen und dabei an festgelegten Punkten bestimmte Aktionen ausführt (vgl. Abschnitt 1.2.2 auf Seite 5). Zur Planung der Routen wird die Bodenstationssoftware APM Planner verwendet (vgl. Abschnitt 3.3.5), die über das MAVLink-Waypoint-Protokoll (vgl. Abschnitt 3.3.1.1 auf Seite 34) mit dem Flugcontroller kommuniziert und so eine Liste von Wegpunkten überträgt. Einfache Aktionen wie Flugmanöver und die Ausrichtung des Gimbals können dabei zusammen mit den Punkten in der Bodenstation konfiguriert werden. Darüber hinausgehendes Verhalten kann realisiert werden, indem das experimentspezifische Modul (vgl. Abschnitt 3.3.2.1 auf Seite 37) das Erreichen eines Wegpunkts durch den Empfang der entsprechenden `mavlink_mission_item_reached_t`-Nachricht vom Flugcontroller erkennt (vgl. Abschnitt 3.3.1.1 auf Seite 34) und dann die gewünschte Funktionalität ausführt.

3.4. Zusammenfassung

In diesem Kapitel wurden die Konstruktion des *Neurocopters*, die Auswahl von Flugcontroller und Bordcomputer und die Entwicklung der Bordsoftware beschrieben.

Hardware Der als Quadcopter konzipierte Rahmen wurde vollständig aus CFK-Rohren und -Platten und Aluminium-Dreh- und -Frästeilen konstruiert. Dadurch ist ein sehr robuster und dennoch leichter Rahmen entstanden, der die Bordelektronik zur Steuerung des Copters und Durchführung von Experimenten auf einer gedämpften Plattform trägt. Als Flugcontroller wurde das APM 2.5 ausgewählt, das mit dem Bordcomputer ODROID-U3 verbunden ist, der über ein zusätzliches Funkmodul zur Kommunikation mit einer Bodenstation verfügt. Für den Antrieb wurden kostengünstige Propeller, Motoren, Motorregler und Akkus aus dem Modellbau verwendet.

Software Die entwickelte Bordsoftware stellt ein Framework zur Verfügung, mit dem Softwaremodule ausgeführt und auf die Kamera und Sensordaten des Flugcontrollers zugegriffen werden können. Zur Verwaltung und Ausführung der Module wird das *Berlin United Framework* verwendet. Der zentrale Bestandteil der entwickelten Software ist die Kommunikation über das MAVLink-Protokoll, das von Flugcontroller und Bodenstation zur Datenübertragung verwendet wird. Vom Flugcontroller werden dabei Sensordaten und Konfigurationsparameter angefordert, so dass sie den Modulen zur Verfügung gestellt oder an die Bodenstation weitergesendet werden können. Zur Umsetzung wurden dabei mit dem *Berlin United Framework* Module zur Anforderung und Aufbereitung der Daten und Repräsentationen für deren Bereitstellung entwickelt. Zur teilweise nötigen Analyse des Flugcontrollerquelltexts wurde das eigens entwickelte Werkzeug *ar2clipse* verwendet, durch das es möglich wurde, die semantische Quelltextanalyse der IDE *Eclipse* zu verwenden.

Die Bodenstation zur Visualisierung der Flugdaten und des Experimentierverlaufs setzt sich aus zwei getrennten Komponenten zusammen. Zum einen wird die Bodenstationssoftware APM Planner des Flugcontrollers und zum anderen das Konfigurationswerkzeug FUremote des *Berlin United Frameworks* verwendet.

4. Evaluierung

Nachdem im vorherigen Kapitel die Konstruktion des *Neurocopters* unter Verwendung von Modellbaukomponenten und die Implementierung der Bordsoftware auf Basis des Berlin United Frameworks beschrieben wurde, befasst sich dieses nun mit der Evaluierung des Systems.

4.1. Copter

Für den *Neurocopter* wurde ein Rahmen konstruiert, da die erhältlichen Modelle unnötig schwer erscheinen, einen ungünstigen Schwerpunkt besitzen und im Bereich der Elektronikplattform nicht unverändert für dieses Projekt verwendet werden können, da angepasste Befestigungsmöglichkeiten für den Flugcontroller und zusätzlichen Bordrechner geschaffen werden müssten. Zudem verfügt keiner der Rahmen über eine Vibrationsdämpfung der Bordelektronik.

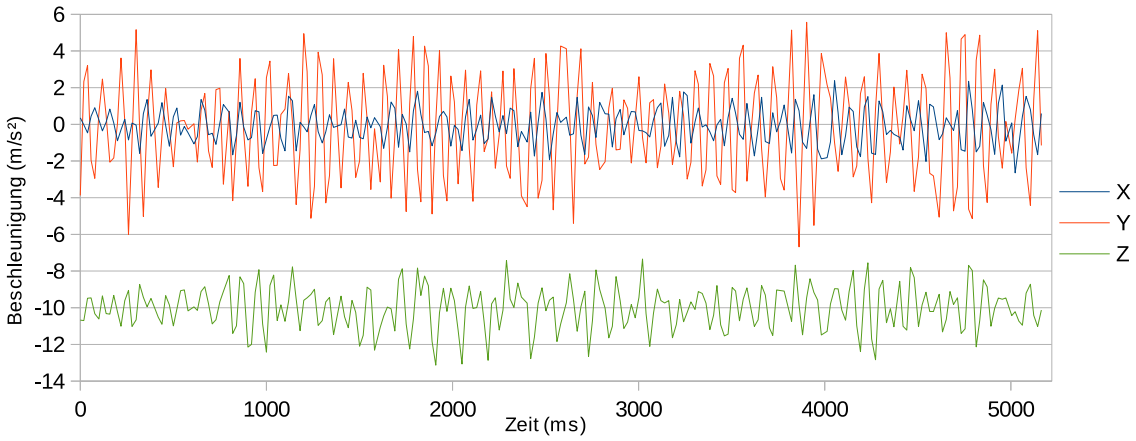
4.1.1. Rahmen

Der entwickelte Copter wird von vier 304 mm großen Propellern angetrieben. Verfügbare Rahmen, die ähnliche Rotorgrößen zulassen, weisen ein Gewicht von 418 g bis 598 g auf (vgl. Tabelle 2.2 auf Seite 19). Damit ist die 343 g schwere Konstruktion um 17,9 % bis 42,6 % leichter. Unter Berücksichtigung des Gewichts der 28 g schweren gedämpften Elektronikplatte des *Neurocopters*, die in ähnlicher Weise für die anderen Modelle ebenso hätte konstruiert werden müssen, vergrößert sich die Gewichtsersparnis auf 23,1 % bis 45,2 %. Durch ein zusätzliches Kürzen der Arme des *Neurocopters* auf die gleiche Länge der verglichenen Rahmen würde sich der Vorteil auf 27,3 % bis 49,7 % erhöhen.

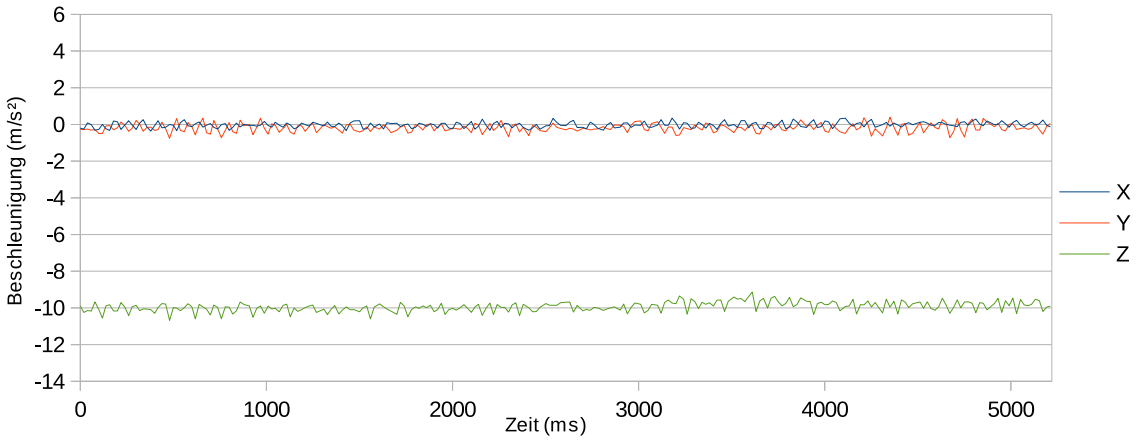
4.1.2. Plattformdämpfung

Die gesamte Bordelektronik des Copters ist auf einer gedämpften CFK-Platte angebracht, um die empfindlichen Sensoren des Flugcontrollers von den Vibrationen der Motoren zu entkoppeln, die diese auf den Rahmen übertragen (vgl. Abschnitt 3.2.3 auf Seite 28). Um die Wirksamkeit dieser Konstruktion zu überprüfen, wurden Sensordaten im Schwebeflug mit und ohne Dämpfung mittels der Logging-Funktionalität

des APM mit annähernd 50 Hz aufgezeichnet. Dabei wurde zur Erhebung der ungedämpften Daten die Elektronikplattform direkt mit dem Rahmen verschraubt. Die Ergebnisse dieser Messungen werden in Abbildung 4.1 gegenübergestellt. Der deutlich stärkere Ausschlag in y-Richtung bei ungedämpfter Plattform ist auf deren asymmetrischen Aufbau zurückzuführen. Dies wurde in einem zweiten Versuch bestätigt, bei dem der Flugcontroller um 90° rotiert nun größeren Vibrationen in x-Richtung ausgesetzt war. In der stärker gestörten y-Richtung führt die Plattformdämpfung zu einer Verringerung der Standardabweichung der Messwerte um den Faktor 12,4, in x- und z-Richtung hingegen um 6,2 beziehungsweise 4,4.



(a) Elektronikplattform direkt mit dem Rahmen verbunden



(b) gedämpfte Elektronikplattform

Abbildung 4.1.: Beschleunigungssensordaten im Schwebeflug
Vergleich der Beschleunigungssensordaten im Schwebeflug mit ungedämpfter und gedämpfter Elektronikplattform.

4.2. Flugcontroller

Zur Steuerung des Copters kommt der Flugcontroller APM 2.5 zum Einsatz, dessen Funktionsweise in diesem Abschnitt untersucht wird.

4.2.1. Position halten

Die Firmware des Flugcontrollers bietet einen Modus zum Halten der aktuellen Postion. Dabei werden GPS- und andere Sensordaten zur Abschätzung der aktuellen Position herangezogen, um ein Abdriften zu erkennen und dem entgegenzuwirken. Dieser Unterabschnitt beschreibt die Auswertung des Systems.

4.2.1.1. GPS-Genauigkeit

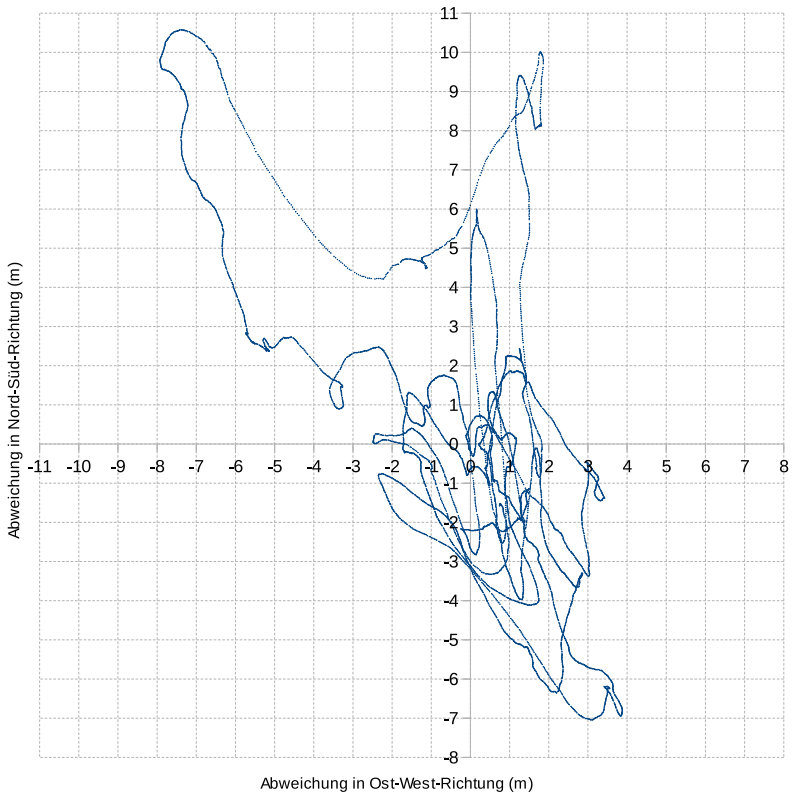
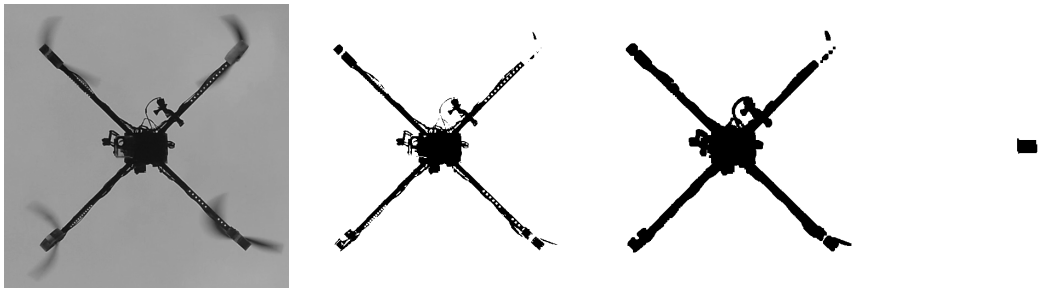


Abbildung 4.2.: Positionsdaten eines unbewegten GPS-Empfängers
Zeitlich korrelierte Messwerte des mit dem APM verbundenen u-blox GPS-Moduls. Angegeben ist die horizontale Abweichung der einzelnen Messwerte zu deren Mittelwert. Die Werte bewegen sich dabei in Ost-West-Richtung in einem Bereich von 11,8 m und in Nord-Süd-Richtung von 17,6 m. Dabei beträgt der mittlere Abstand zum Erwartungswert 3,6 m und der maximale 13 m.

Da die Positionsbestimmung nicht allein durch Aufintegration der Accelerometerwerte (vgl. Abschnitt 2.2.1 auf Seite 10) durchgeführt werden kann, müssen verlässlichere absolute Positionsdaten in die Schätzung einbezogen werden. Die Genauigkeit des dazu verwendeten u-blox LEA-6 GPS-Moduls [29] wird hier anhand von 9700 Messwerten eines unbewegten Empfängers untersucht, die über einen Zeitraum von 35 Minuten mit 4,5 Hz aufgezeichnet wurden. Dafür wurde ein Modul für das Berlin United Framework (vgl. Abschnitt 3.3.2.1 auf Seite 37) entwickelt, das den entsprechenden MAVLink-Data-Stream (vgl. Abschnitt 3.3.1.1 auf Seite 35) vom APM anfordert und in einer Logdatei des Bordrechners speichert. Dabei beträgt die Auflösung der übermittelten Breitengrade immer etwa 1,1 cm, während die der Längengrade ortsabhängig auch feiner sein kann. Die so bestimmten Werte werden in Abbildung 4.2 auf Seite 53 visualisiert. Dazu ist die horizontale Abweichung der einzelnen Messwerte vom Erwartungswert aufgetragen. Auffällig ist dabei die starke zeitliche Korrelation der einzelnen Punkte, die in der systembedingten Art der verschiedenen Fehlerquellen des GPS begründet ist [120]. Während der Abstand zweier aufeinanderfolgender Messwerte maximal 14,5 cm und im Mittel 1,9 cm beträgt, ist die Abweichung zum Erwartungswert mit maximal 13 m und im Mittel 3,6 m um mehrere Größenordnungen höher. Daraus ergibt sich, dass die GPS-Daten mangels ausreichender Genauigkeit nicht allein zur absoluten Positionsbestimmung herangezogen werden können und mit anderen Werten in geeigneter Weise kombiniert werden müssen.

4.2.1.2. Visuelle Positionsbestimmung



(a) Graustufenbild **(b)** Binärbild **(c)** Dilatation **(d)** Erosion

Abbildung 4.3.: Optische Positionsbestimmung des Copters

Zur Positionsbestimmung wurde der Copter mit einer nach oben gerichteten Kamera gefilmt (Abb. 4.3a). In dem mit einem konstanten Schwellwert erzeugten Binärbild (Abb. 4.3b) werden zunächst kleinere Löcher durch Dilatation geschlossen (Abb. 4.3c). Anschließend können die schlanken Arme des Copters durch Erosion entfernt werden, so dass nur noch der Kern der Plattform übrig bleibt (Abb. 4.3d). Dessen Schwerpunkt wird als Approximation für die Position des Copters verwendet.

Um die Abweichung des fliegenden Copters von der zu haltenden Position zu bestimmen, wurde er im Schwebeflug in einer Höhe von 2,6 m von einer auf dem Boden

befindlichen nach oben gerichteten Kamera mit 30 Bildern pro Sekunde gefilmt. Aus den Videodaten ließ sich dann die Position bildweise mittels einfacher Bildverarbeitungstechniken unter Verwendung der C++-Programmbibliothek *OpenCV* extrahieren. Dieser Vorgang wird in Abbildung 4.3 auf Seite 54 anhand eines Einzelbildes veranschaulicht. Die Auflösung der so bestimmten Positionsdaten ergibt sich aus der Flughöhe des Copters und der Bildgröße der Kamera und beträgt etwa 1,7 mm pro Pixel. Der so ermittelte Positionsverlauf ist in Abbildung 4.4 visualisiert und zeigt, dass sich der Copter in einem etwa 800 mm × 740 mm großen Bereich bewegt. Zwischen zwei aufeinanderfolgenden Einzelbildern driftet der Copter dabei im Mittel um 3 mm und maximal um 9 mm.

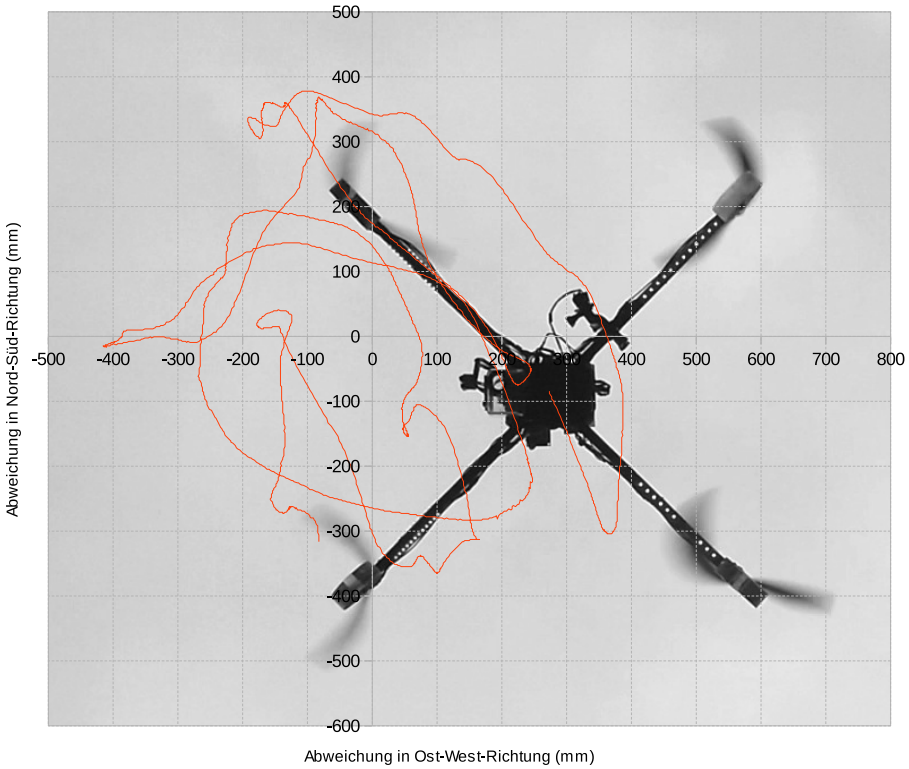


Abbildung 4.4.: Positionsdaten des schwebenden Copters
Gezeigt ist der horizontale Positionsverlauf des für 85 s auf der Stelle schwebenden Copters. Bei einer mittleren Abweichung von 270 mm vom Erwartungswert und einer maximalen von 479 mm bewegt er sich um 800 mm in Ost-West-Richtung und um 740 mm in Nord-Süd-Richtung.

4.2.1.3. Fazit

Unter Berücksichtigung der unterschiedlichen Aufnahmeraten der GPS-Positionsdaten und der visuellen Positionsbestimmung des fliegenden Copters zeigt sich, dass der Flugcontroller trotz zusätzlicher Sensoren wie Accelerometer und Gyroskop die Position nicht genauer halten kann, als es unter alleiniger Verwendung des GPS zu

erwarten wäre. Dessen Werte unterscheiden sich pro Sekunde im Mittel um 8,8 cm, die des fliegenden Copters um 8,7 cm.

4.2.2. Senderate der MAVLink-Datenströme

Die ArduCopter-Firmware des Flugcontrollers kann auf Anfrage verschiedene Datenströme aus MAVLink-Paketen (vgl. Abschnitt 3.3.1.1 auf Seite 35) mit Sensordaten und anderen Statusinformationen des Flugcontrollers über dessen serielle Schnittstelle aussenden. Zur Analyse der Senderaten wurde ein Modul für das Berlin United Framework (vgl. Abschnitt 3.3.2.1 auf Seite 37) geschrieben, das die Empfangszeitpunkte der Pakete aufzeichnet. Das fürs Senden verantwortliche Codesegment der ArduCopter-Firmware wird mit 50 Hz ausgeführt und darf eine festgelegte maximale Verarbeitungsdauer nicht überschreiten. Die zur Verfügung stehende Rechenzeit wird dabei nicht gleichmäßig auf alle angeforderten Pakettypen verteilt. Stattdessen arbeitet der Code die Nachrichten bei jeder Ausführung solange in einer festgelegten Reihenfolge ab, bis der Zeitschlitz aufgebraucht ist. Wegen der schwachen, mit 16 MHz getakteten, 8-bit AVR ATmega2560 CPU [121] des APM 2.5, die es nicht schafft, alle ausstehenden Pakete im aktuellen Zeitschlitz zu senden, und der Bevorzugung bestimmter Nachrichten kann es dazu kommen, dass andere überhaupt nicht übertragen werden. Auch wird das zur Verfügung stehende Zeitfenster durch weitere Faktoren beeinflusst: Während das Aktivieren der Logging-Funktion des APM die Pakete nur um wenige Millisekunden verzögert und so die Senderate um 0,5 Hz verlangsamt, haben die Flugmodi des Copters einen größeren Einfluss. Werden alle verfügbaren Streams mit 10 Hz angefordert, so kann der sich nicht im Flugmodus befindliche gesicherte Controller alle Pakete mit 8 Hz senden. Ein Wechsel in den Modus zur manuellen Steuerung verringert die mittlere Rate auf 6,2 Hz, wobei die niedrig priorisierten Nachrichten auf bis zu 4 Hz abfallen.

4.3. Werkzeug: ar2clipse

Zur Codeanalyse der ArduCopter-Firmware wurde das Programm *ar2clipse* (vgl. Abschnitt 3.3.3 auf Seite 39) entwickelt, um die semantische Analyse der Entwicklungsumgebung *Eclipse* nutzen zu können. Insbesondere bei der Implementierung der MAVLink-Datenströme (vgl. Abschnitt 3.3.4.5 auf Seite 46) und des Parameterprotokolls (vgl. Abschnitt 3.3.4.5 auf Seite 47) zeigte sich, wie nötig dieses Hilfsmittel für eine korrekte Umsetzung ist.

Durch die fast ausschließliche Verwendung von Verknüpfungen zur Einbindung der Quelltextdateien ins Eclipse-Projekt können die Originaldateien auch direkt aus der Entwicklungsumgebung heraus bearbeitet werden. Da zur Organisation des Quelltexts der Firmware die Versionsverwaltungssoftware *Git* verwendet wird, können etwaige Korrekturen mit dieser direkt dem ArduCopter-Projekt zugänglich gemacht werden.

Der Python-Code zur Bearbeitung der Eclipse-Projektdatei ist nicht speziell für diesen Anwendungsfall entwickelt, sondern sehr allgemein gehalten, so dass sich weitere Konfigurationsmöglichkeiten von Projektdateien leicht ergänzen lassen. So wäre es beispielsweise möglich, die im derzeitigen Arbeitsablauf noch einmalig manuell vorgenommenen Einstellungen ebenso zu automatisieren.

4.4. Bordsoftware

Die Bordsoftware des *Neurocopters* wurde auf Basis des Berlin United Frameworks (vgl. Abschnitt 3.3.2 auf Seite 37) entwickelt. Das so geschaffene Programmiergerüst soll einfache Zugriffsmöglichkeiten auf die Sensordaten des Flugcontrollers und die Bordkamera des Copters bereitstellen sowie die Funkkommunikation zu einer Bodenstation ermöglichen. In diesem Abschnitt wird das so entstandene Framework des *Neurocopters* evaluiert.

4.4.1. MAVLink

Der Großteil der entwickelten Software dient der MAVLink-Kommunikation des Bordcomputers mit dem Flugcontroller und der Bodenstation sowie der Aufbereitung der empfangenen Pakete für die weitere Verwendung in Modulen der Bordsoftware. Dieser Unterabschnitt befasst sich mit der Auswertung dieser Komponenten.

4.4.1.1. Codegenerierung

Auf dem C-Codegenerator der MAVLink-Bibliothek [122] aufbauend wird zur einfacheren Verwendung innerhalb der Bordsoftware automatisiert C++-Code zur Verarbeitung der MAVLink-Pakete erzeugt (vgl. Abschnitt 3.3.1.2 auf Seite 35). Die Erweiterung des bestehenden Generators wurde dabei minimalinvasiv gehandhabt, um mögliche zukünftige Aktualisierungen des Generators so einfach wie möglich zu gestalten: Es werden lediglich zwei aufeinanderfolgende Zeilen in einer Funktion hinzugefügt, die den C++-Generator aufrufen.

Generierte Funktionen Die generierten überladenen und template-basierten Funktionen zum Kodieren und Dekodieren von MAVLink-Nachrichten ermöglichen es, generischen Code zur Verarbeitung der Pakete zu schreiben. Demonstriert wird dies in der template-basierten Repräsentation `mavlink_message_container` zur Bereitstellung der empfangenen Pakete in dekodierter Form (vgl. Abschnitt 3.3.4.4 auf Seite 44). Beide Funktionsklassen verwenden im Gegensatz zu den durch sie gekapselten C-Funktionen keine Ausgabeparameter, sondern nutzen den Rückgabewert der Funktion. Auf diese Weise wird vermieden, dass für die Ausgabe uninitialisierte Objekte erzeugt werden müssen, die erst durch die Übergabe an die Funktion in

einen definierten Zustand versetzt werden. Der übersetzte Maschinencode ist dabei trotz der Größe der zurückgegebenen Objekte nicht weniger effizient. Dies ist in der für C++-Compiler als obligatorisch zu betrachtenden NRV-Optimierung (*Named Return Value optimization*) [123] bedingt.

Aktualisierung Die Codegenerierung muss nach einer Aktualisierung der verwendeten MAVLink-Bibliothek einmalig ausgeführt werden. Anpassungen im bestehenden, die generierten Funktionen verwendenden Quelltext sind danach im Regelfall nicht nötig. Ausnahmen bilden lediglich Änderungen in der Semantik bereits existierender MAVLink-Pakete, die nun neu interpretiert werden müssen. Ein Beispiel hierfür ist das Hinzufügen eines neuen Pakets zu einem MAVLink-Datenstrom (vgl. Abschnitt 3.3.1.1 auf Seite 35), das folglich bei dessen Verarbeitung berücksichtigt werden muss. Dies wird in den Abschnitten 4.4.1.2 und 4.4.1.3 noch im Detail erläutert.

4.4.1.2. MAVLink-Kommunikation

Wie in Abschnitt 3.3.4.3 auf Seite 43 beschrieben, werden empfangene MAVLink-Nachrichten optional über eine Debug-Option des Berlin United Frameworks ausgegeben. Auf diese Weise konnte die Kommunikation mit anderen MAVLink-Geräten wie dem Flugcontroller und der Bodenstation beobachtet und überprüft werden.

Änderungen im Kommunikationsverhalten des Flugcontrollers können schnell erkannt werden, da die Repräsentation zur Bereitstellung empfangener MAVLink-Pakete die Nachrichtentypen auflisten kann, die nicht dekodiert worden sind (vgl. Abschnitt 3.3.4.4 auf Seite 44). Dies gilt insbesondere für neu hinzugekommene Pakete, die dementsprechend noch nicht verarbeitet und somit angezeigt werden. Auf diese Weise wurde festgestellt, dass dem MAVLink-Datenstrom `MAV_DATA_STREAM_EXTRA3` der Pakettyp `mavlink_rangefinder_t` nachträglich in einer neueren Firmwareversion hinzugefügt worden ist.

4.4.1.3. MAVLink-Datenströme

Zur Anforderung und Bereitstellung der MAVLink-Datenströme des Flugcontrollers wurden eine template-basierte Repräsentation und ein entsprechendes Modul entwickelt (vgl. Abschnitt 3.3.4.5 auf Seite 46). Das Hauptaugenmerk wurde dabei auf eine möglichst einfache, nicht-redundante Konfiguration der Datenströme gelegt.

Erweiterbarkeit Die Konfiguration der Pakete eines Datenstroms findet an einer einzigen Stelle im Quelltext statt. So muss etwa ein neu hinzugekommenes Paket nur dort eingetragen werden und steht ohne weitere Anpassungen über die Repräsentation der Datenströme zur Verfügung. In Abschnitt 4.4.1.2 wurde bereits

angedeutet, dass ein MAVLink-Datenstrom bei einem Softwareupdate des Flugcontrollers verändert wurde. Auflistung 4.1 zeigt die farblich hervorgehobene nötige Anpassung in der Datenstromdefinition zur Auswertung des neu hinzugefügten `mavlink_rangefinder_t`-Pakets.

Auflistung 4.1: Datenstrom-Anpassung für ein neues MAVLink-Paket

```
typedef data_stream<
  MAV_DATA_STREAM_EXTRA3,
  mavlink_ahrs_t,
  mavlink_hwstatus_t,
  mavlink_system_time_t,
  mavlink_rangefinder_t
> apm_mavlink_data_stream_extra3;
```

Funktionstest Zur Überprüfung der Funktionsweise des Moduls zur Anforderung der Datenströme wurden gezielt Pakete verworfen. Auf diese Weise konnte sichergestellt werden, dass Datenstromanforderungen an den Flugcontroller nach Ablauf eines Timeouts erneut gesendet werden.

4.4.1.4. MAVLink-Parameterprotokoll

Für den lesenden Zugriff auf die Konfigurationsparameter des Flugcontrollers wurde das MAVLink-Parameterprotokoll (vgl. Abschnitt 3.3.4.5 auf Seite 47) implementiert. Um die Funktionsweise gemäß des Zustandsdiagramms in Abbildung 3.13 auf Seite 48 zu verifizieren, wurde der Quelltext zu Testzwecken durch Präprozessordirektiven modifiziert. So werden zum einen die Zustandsübergänge durch Debug-Ausgaben angezeigt, um Zustandswechsel nachvollziehen zu können. Um dabei auch das Auslösen des Timeouts zur Neuanforderung einzelner Pakete testen zu können, wurden einzelne Pakete vor der Verarbeitung durch das Modul mehrfach unterdrückt. So wurde auch eine den Flugcontroller nicht erreichende Wiederanforderung simuliert. Auf die Weise wurde die Korrektheit der Implementierung bestätigt.

Genauso wurden Parameteranfragen von der Bodenstation zurückgehalten oder nicht beantwortet, um diesen Teil der Implementierung erfolgreich zu überprüfen.

4.4.2. Integration in das Berlin United Framework

Das Berlin United Framework (vgl. Abschnitt 3.3.2 auf Seite 37) wurde als Basis der Neurocopter-Bordsoftware ausgewählt. Grund hierfür ist sein modularer Aufbau, der Abhängigkeiten reduziert und viele nützliche Debug-Optionen beinhaltet, was ein einfaches verteiltes Arbeiten begünstigt.

Sämtliche bereitgestellten Funktionen der Bordsoftware wurden mit den Mitteln des Frameworks entwickelt, so dass keinerlei Anpassungen nötig waren: Ein *Service* realisiert den Empfang aller eingehenden MAVLink-Nachrichten. Diese Daten werden dann von *Modulen* verarbeitet und in *Repräsentationen* abgelegt. Die Bilder der Bordkamera werden ebenfalls über eine Repräsentation verfügbar gemacht. Darauf aufbauend können experimentspezifische Module entwickelt werden.

4.4.3. Wartbarkeit

Zur Versionsverwaltung des Neurocopter-Frameworks wird die Software *Git* verwendet. Dabei sind sowohl das Berlin United Framework (vgl. Abschnitt 3.3.2 auf Seite 37) als auch die MAVLink-Bibliothek als Untermodule eingebunden und können so auf einfache Weise durch die Ausführung entsprechender Skripte aktuell gehalten werden. Anschließend muss lediglich wie bereits in Abschnitt 4.4.1.1 auf Seite 58 beschrieben der MAVLink-C++-Codegenerator ausgeführt werden, um die Funktionen zur Verarbeitung der MAVLink-Pakete zu erzeugen.

Zusammenfassend ergibt sich also eine sehr einfach zu bedienende Integration der externen Bibliotheken, so dass diese aktuell gehalten werden können, um etwa Fehlerkorrekturen und Erweiterungen zeitnah verwenden zu können.

4.4.3.1. Hinzufügen weiterer MAVLink-Geräte

Die derzeitige Bordsoftware sieht neben der Bodenstation zwei per MAVLink angeschlossene Geräte vor: Den Flugcontroller und den PX4Flow-Sensor zur Bestimmung des optischen Flusses (vgl. Abbildung 2.7b auf Seite 12). Sollen weitere Geräte hinzugefügt werden, so reicht es, in der template-basierten Struktur `device_data` (vgl. Abschnitt 3.3.4.2 auf Seite 42) einen Eintrag für sie anzulegen, um sämtliche implementierte Funktionen wie etwa den Empfang von MAVLink-Nachrichten und deren Organisation in einer Repräsentation oder den Zugriff auf die mit dem Parameterprotokoll empfangenen Parameter zu nutzen.

5. Diskussion

Mit dem *Neurocopter* soll eine fliegende Experimentierplattform zur Erforschung der Hirnaktivität von Honigbienen geschaffen werden. Der Quadcopter verfügt über einen leistungsstarken Bordcomputer, der mit dem Flugcontroller und einer am Copter befindlichen Kamera verbunden ist. Über ein bereitgestelltes Framework kann von der Bordsoftware auf die Sensordaten des Flugcontrollers und die Kamerabilder zugegriffen werden. Außerdem stellt es einen Übertragungskanal zu einer Bodenstationssoftware bereit, mit der die Empfangenen Daten während des Fluges visualisiert und Experimente gesteuert werden können. Das entwickelte System wird im Folgenden diskutiert.

5.1. Flugcontroller

Als Flugcontroller kommt das APM 2.5 mit der ArduCopter Firmware zum Einsatz (vgl. Abschnitt 3.1.2 auf Seite 23). Es wurde aus Kostengründen trotz seiner schwachen 8-bit CPU mit 16 MHz wegen der sonst vergleichbaren Eigenschaften dem leistungsstärkeren PX4FMU vorgezogen. So kann es etwa vorprogrammierte Routen aus Wegpunkten abfliegen, ein Gimbal ansteuern und seine Sensordaten über eine serielle Schnittstelle zur Verfügung stellen.

5.1.1. Position halten

Bei der Überprüfung des Flugmodus zum Halten der aktuellen Position stellte sich heraus, dass der Flugcontroller trotz zusätzlicher Sensoren keine Verbesserung der Genauigkeit gegenüber der reinen GPS-Daten erzielen kann (vgl. Abschnitt 4.2.1.3 auf Seite 55). Selbst in einem kleinen Zeitfenster von 85 s weicht der Copter im Mittel um 27 cm und maximal um fast 50 cm von der zu haltenden Position ab. Dies genügt nicht der Präzision einer Biene, die das nur wenige Zentimeter große Flugloch des Bienenstocks zielsicher anfliegen kann. Inwiefern sich die Ungenauigkeit auf Experimente auswirkt, bei denen etwa Routen mit bekannten Landmarken abgeflogen werden sollen, lässt sich derzeitig jedoch nicht beurteilen.

Bei der anschließenden Codeanalyse des ArduCopter-Quelltexts mittels *ar2clipse* (vgl. Abschnitt 3.3.3 auf Seite 39) zeigte sich, dass Techniken zur Verbesserung der Positionsschätzung wie der erweiterte Kalman-Filter (*EKF*) zwar im Code vorhanden sind, jedoch leistungsschwachen Flugcontrollern wie dem verwendeten APM 2.5

nicht zur Verfügung stehen und erst mit Prozessoren ab einer Taktrate von 150 MHz genutzt werden können [124,125]. Eine mögliche Verbesserung durch den Einsatz des vom APM 2.5 unterstützten mausbasierten Sensors ADNS3080 zur Bestimmung des optischen Flusses (vgl. Abbildung 2.7a auf Seite 12) wurde mangels eines Sensors nicht untersucht und bleibt weiterführenden Arbeiten überlassen.

5.1.2. Senderate der MAVLink-Datenströme

Die Senderate der MAVLink-Datenströme kann je nach angeforderten Strömen, Flugmodus und Pakettyp auf bis zu 4 Hz abfallen, was der leistungsschwachen CPU des Flugcontrollers zuzuschreiben ist (vgl. Abschnitt 4.2.2 auf Seite 56). Der GPS-Sensor liefert seine Positionsdaten mit 5 Hz. Insofern wird das Auslesen dieses Sensors durch die geringe Senderate nicht beeinträchtigt. Accelerometer und Gyroskop können allerdings um Größenordnungen häufiger abgetastet werden. Daher musste etwa bei der Evaluierung der Vibrationsdämpfung (vgl. Abschnitt 4.1.2 auf Seite 51) zur Aufzeichnung der Beschleunigungssensordaten auf die interne Logging-Funktionalität des APM 2.5 zurückgegriffen werden, um die Daten mit annähernd 50 Hz zu erheben. Inwiefern sich diese Einschränkung auf bienenspezifische Experimente auswirkt, lässt sich derzeitig mangels genauerer Anforderungen nicht beurteilen.

5.2. Copter

Mit dem Rahmen des Neurocopter ist eine Konstruktion gelungen, die erheblich leichter als andere erhältliche Modelle ist (vgl. Abschnitt 4.1.1 auf Seite 51). Mehrere Abstürze bei Testflügen haben gezeigt, dass diese Gewichtsersparnis nicht zu Lasten der Stabilität geht. Beispielsweise hat ein durch einen Pilotenfehler verursachter ungebremster Fall aus etwa 6 m Höhe auf Betonboden alle Rotoren des Copters zerstört und eine Motorwelle verbogen, wohingegen der Rahmen keinerlei Schaden genommen hat. Im Gegensatz dazu verbiegen die bei einigen Modellbaurahmen verwendeten dünnwandigen Aluminiumausleger bereits beim Aufsetzen des Arms auf den Boden bei einer unsauberen Landung.

Nicht bedacht wurden bei der Konstruktion allerdings unsachgemäße Montagearbeiten am Copter durch dessen Anwender. In einer ersten Version waren beispielsweise die Unterteile der Motorklemmen (vgl. Abbildung 3.9 auf Seite 30) zur Gewichtsersparnis aus Polyamid gefertigt. Diese sind durch ein zu festes Anziehen der Schrauben gebrochen und wurden durch die nun verwendeten Nachbauten aus Aluminium ersetzt.

Ebenso muss bedacht werden, dass aufgrund des kompletten Eigenbaus durch Abstürze oder fehlerhafte Anwendung beschädigte oder zerstörte Komponenten neu konstruiert werden müssen und nicht nachgekauft werden können. Dieser Punkt

muss aber nicht zum Nachteil ausgelegt werden: Zum einen kann, die entsprechenden Werkzeuge und Fachkunde vorausgesetzt, mit dem Neubau unmittelbar begonnen werden, so dass sich die Reparaturzeit gegenüber einer Bestellung der Komponenten sogar verkürzen kann. Zum anderen hat sich die Konstruktion als sehr Robust erwiesen, so dass derartige Reparaturen als unwahrscheinlich einzustufen sind.

5.2.1. Plattform

Aufgrund der durchgehenden Arme des Rahmens konnte die Plattform sehr leicht konstruiert werden, da sie nur die Bordelektronik tragen muss. Dieser Aufbau erlaubt es beispielsweise, auf einfache Weise die Plattform durch eine andere nicht im Zentrum des Rahmens liegende zu ersetzen. So kann der Schwerpunkt des Copters verlagert werden, um etwa das Gewicht eines einseitigen Auslegers auszugleichen, der für ein Experiment benötigt wird.

Die Elektronikplatte ist zur Vibrationsdämpfung über Gummipuffer mit der restlichen Plattform verbunden. Dieses System erlaubt es, die gesamte Elektronik auf eine etwaige andere Plattform zu übertragen.

Die Dimensionierung der konstruierten Plattform wurde anhand der Größe des Akkus und der restlichen verwendeten Bordelektronik vorgenommen. Daraus ergibt sich eine rechteckige, längliche, nicht-quadratische Grundfläche. Wenn der *Neurocopter* in +-Konfiguration statt der aktuellen ×-Konfiguration (vgl. Abbildung 2.1 auf Seite 8) verwendet werden soll, ist die Plattform nicht mehr symmetrisch zu der neuen um 45° rotierten Quer- und Längsachse des Copters. Dies gilt insbesondere auch für die Anordnung der Gummidämpfer der Elektronikplattform, die die Restvibrationen nun nicht mehr gleichmäßig entlang der Achsen übertragen. Ob und in welchem Ausmaß dies messbar ist, gilt es noch zu überprüfen. Diese Frage müsste jedoch bei einer quadratischen Plattform erst gar nicht gestellt werden, weswegen die minimale Dimensionierung der Plattform nicht optimal ist.

Zudem lässt die kleine Elektronikplatte, die so bestückt ist, dass ihr Schwerpunkt mittig der Dämpfer ist, keinen Spielraum zur Montage einer zusätzlichen Kamera. Dies ist nicht Teil der Projektanforderungen (vgl. Abschnitt 1.2.1 auf Seite 4); jedoch ist denkbar, dass eine zweite Kamera zur Erzeugung stereoskopischer Aufnahmen in Experimenten mit neuronalen Netzen benötigt wird.

5.2.2. Antrieb

Der Antrieb des *Neurocopters* setzt auf vier kompakte, leichte, bürstenlose Motoren, die die 305 mm großen Propeller antreiben. Wie der Schubmessungsversuch zur Dimensionierung des Copters gezeigt hat (vgl. Abschnitt 3.2.1 auf Seite 26), nimmt der Wirkungsgrad des Systems mit zunehmendem Schub ab. Außerdem ist bekannt, dass die Effizienz von Propellern mit deren Durchmesser zunimmt (vgl.

Abschnitt 3.2 auf Seite 25). Von daher ist der derzeitige Antrieb in Bezug auf eine möglichst lange Flugzeit nicht optimal. Andererseits weisen kleine Propeller ein geringeres Trägheitsmoment auf, wodurch Drehzahländerungen schneller wirksam werden und der Copter agiler wird. Der *Neurocopter* weist jedoch kein zu träges Flugverhalten auf, weswegen das Antriebskonzept in Hinblick auf eine Maximierung der Flugzeit überdacht werden sollte.

5.3. Bordsoftware

Die Bordsoftware des *Neurocopters* basiert auf dem Berlin United Framework und ermöglicht den Zugriff auf die Sensordaten des Flugcontrollers APM 2.5 sowie auf die Bilder der Bordkamera PlayStation Eye.

Die entwickelten Programmteile dienen hauptsächlich der MAVLink-Kommunikation mit dem Flugcontroller, der Bodenstation und anderen MAVLink-Geräten. Der Funktionsumfang entspricht somit einem Teil der Funktionalität der Bodenstationssoftware APM Planner des Flugcontrollers. Das Framework wurde jedoch auch nicht mit dem Anspruch entwickelt, die Möglichkeiten einer Bodenstation komplett nachzubilden. Die bereitgestellten und nicht vorhandenen Funktionen werden im Folgenden diskutiert.

Parameterprotokoll Die Implementierung des Parameterprotokolls erlaubt nur den lesenden Zugriff auf die Konfigurationsparameter des Flugcontrollers. Auf diese Weise kann die Bodenstation so auf die Parameter zugreifen, als wäre sie direkt mit dem Flugcontroller verbunden. Der schreibende Zugriff wurde jedoch nicht implementiert, da dies weit über die gestellte Anforderung des Lesens der Sensordaten hinausgeht (vgl. Abschnitt 1.2.2 auf Seite 5). Ferner sind Änderungen der Parameter nur während der grundlegenden Konfiguration des Flugcontrollers nötig, so dass die hierzu nötige direkte Verbindung keine nennenswerte Beeinträchtigung im allgemeinen Arbeitsablauf darstellt.

MAVLink-Datenströme Über das Bereitstellen von Datenströmen ermöglicht der Flugcontroller den Zugriff auf seine Sensordaten und andere Statusinformationen. Mit der Bordsoftware können die Datenströme angefordert werden. Der Bodenstation wird ebenfalls der Zugriff ermöglicht, wobei ihr jedoch untersagt ist, geringere Datenraten zu setzen, als sie von der Bordsoftware festgelegt sind. Diese Einschränkung musste vorgenommen werden, um zu verhindern, dass für Experimente nötige Datenströme durch die Bodenstation negativ verändert werden.

Waypoint-Protokoll Das MAVLink-Waypoint-Protokoll wird zur Konfiguration von Flugrouten verwendet (vgl. Abschnitt 3.3.1.1 auf Seite 34). Es wurde jedoch

nicht implementiert, so dass der Bodenstation-Rechner zur Übertragung direkt mit dem Flugcontroller verbunden werden muss (vgl. Abschnitt 3.1.2 auf Seite 25). Diese Einschränkung wurde zu Gunsten der Implementierung der Verwaltung der Datenströme und des Parameterprotokolls in Kauf genommen, da diese Funktionen für die grundlegende Anbindung der Bodenstation unabdingbar sind.

Weitere MAVLink-Pakete Die Bordsoftware leitet keine MAVLink-Pakete direkt zwischen den MAVLink-Geräten weiter. Dafür ist immer ein Modul nötig, das aufs entsprechende Paket reagiert und es verarbeitet oder explizit weiterleitet. Diese Entscheidung erzwingt, dass die Kommunikation zwischen den Geräten genau bekannt sein muss, so dass die benötigten Abläufe implementiert werden können. Dieses Verfahren ist aufwändig, stellt aber sicher, dass etwaige Protokolländerungen schneller auffallen.

5.4. Bodenstation

Die Bodenstationssoftware des *Neurocopters* setzt sich aus zwei getrennten Programmen zusammen (vgl. Abschnitt 3.3.5 auf Seite 49). Zur Konfiguration des Flugcontrollers und zur Anzeige dessen von Sensordaten wird die zugehörige Software APM Planner verwendet. Weitere experimentspezifische Daten können über FUremote des Berlin United Frameworks bearbeitet und angezeigt werden. Diese Teilung bringt keine erheblichen Nachteile mit sich. Beide Programme erkennen von der Bordsoftware beim Start automatisch gesendete Daten und zeigen den Copter an. Während der Arbeit mit der Software muss zwischen Programmen statt Programmfenstern oder Ansichten gewechselt werden. Der Mehraufwand in der Anwendung liegt folglich beim Start zweier Programme und deren getrennter Pflege.

6. Ausblick

Im Rahmen dieser Arbeit wurde mit dem *Neurocopter* eine Basis für Experimente zur Erforschung von Honigbienen während des Fluges geschaffen. Inwieweit die konzipierte Plattform unverändert verwendet werden kann, wird sich erst im Verlauf zukünftiger Versuche zeigen.

Nun liegt es an weiterführenden Arbeiten, auf den hier gewonnenen Erkenntnissen und dem konstruierten Copter aufbauend die Experimentierplattform zu perfektionieren. Dabei besteht in einigen Punkten Optimierungspotential, was im Folgenden beschrieben wird.

6.1. Flugcontroller

Der verwendete Flugcontroller APM 2.5 hat sich als unzureichend erwiesen (vgl. Abschnitt 5.1 auf Seite 61), was im wesentlichen auf dessen leistungsschwachen Mikroprozessor zurückzuführen ist. Folglich sollte zukünftig auf einen schnelleren Controller wie den PX4FMU (vgl. Abschnitt 2.2.2 auf Seite 12) gesetzt werden. Zu erwarten sind zum einen eine deutlich höhere Senderate der MAVLink-Datenströme und zum anderen ein besseres Halten der Flugposition. Letzteres wird durch mehrere Aspekte beeinflusst: Der schnellere Prozessor kann aufwändigere Verfahren zur Positionsabschätzung wie den erweiterten Kalman-Filter einsetzen. Außerdem kann der PX4FLOW-Sensor zur Bestimmung des optischen Flusses verwendet werden, um die Schätzung weiter zu verbessern. Nicht zuletzt wird dieser Controller auch von der neusten ArduCopter-Firmware unterstützt, die weitere Verbesserungen mit sich bringen kann und stetig weiterentwickelt wird.

Diese aufgeführten Schritte sollten iterativ ausgeführt werden, um die ausschlaggebenden Faktoren genau bestimmen und die einzelnen Phasen vergleichen zu können.

6.2. Copter

Um die Flugdauer zu maximieren, können größere, langsamer drehende 356 mm- oder 381 mm-Propeller statt der derzeitig verwendeten 305 mm-Rotoren eingesetzt werden. Für diesen Umbau werden allerdings auch entsprechende Motoren niedrigerer Drehzahl benötigt. Hier bietet es sich an, leicht überdimensionierte, schwerere

Modelle in Betracht zu ziehen, die im Schwebeflug einen besseren Wirkungsgrad als ausreichend ausgelegte Modelle besitzen und so ihr Mehrgewicht durch eine geringere Leistungsaufnahme trotz des nötigen zusätzlichen Schubs kompensieren.

6.2.1. Plattform

Die rechteckige gedämpfte Elektronikplatte des Copters ist sehr knapp dimensioniert und bietet keine Möglichkeiten zur Unterbringung weiterer Komponenten (vgl. Abschnitt 5.1 auf Seite 61). Dies gilt insbesondere für eine zusätzliche zweite Kamera. Sollte sich im Verlauf der Experimente ein Bedarf hierfür herausstellen, so bietet sich eine Neugestaltung der gesamten Plattform an. Zur Befestigung am Rahmen kann weiterhin auf die vorhandenen Klemmen und Abstandhalter zurückgegriffen werden, so dass lediglich neu dimensionierte CFK-Platten gefertigt werden müssen. Dabei kann die Konstruktion auch derartig gestaltet werden, dass die Elektronikplatte mehrere Aufnahmen für die Gummidämpfer besitzt, um eine um 45° rotierte Montage für die Verwendung in der +-Konfiguration des Copters zu ermöglichen.

6.3. Bordsoftware

Neben neuen experimentspezifischen Anforderungen an die Bordsoftware existieren auch allgemeine Erweiterungen, von denen alle Versuche profitieren können. Dabei geht es in erster Linie um nicht implementierte Teile des MAVLink-Protokolls. Zum einen kann die Implementierung des Parameterprotokolls so erweitert werden, dass auch ein schreibender Zugriff auf die Werte möglich ist. Dabei ist abzuwägen, ob Änderungen im Flug generell oder partiell untersagt werden sollten. Zum anderen erscheint eine Implementierung des MAVLink-Waypoint-Protokolls nützlich, da so die Routenplanung auch über die drahtlose Verbindung von der Bodenstation über den Bordrechner an den Flugcontroller erfolgen kann.

Eine weitere mögliche Erweiterung liegt in der Steuerung des Copters durch die Bordsoftware. Dies kann entweder durch die Vorgabe von Bewegungen entlang der Achsen durch das `mavlink_manual_control_t`-Paket ähnlich einer Fernsteuerung oder abstrakter durch Angabe von GPS-Koordinaten geschehen. Letzteres Verfahren greift auf das Waypoint-Protokoll zurück. Dabei wird ein `mavlink_mission_item_t`-Paket verwendet, das das Kommando `MAV_CMD_OVERRIDE_GOTO` enthält. Dieses gibt über den Wert `MAV_GOTO_HOLD_AT_SPECIFIED_POSITION` die Zielposition an, die der Copter anfliegt, um dort zu verharren, bis weitere Befehle folgen.

A. Anhang

Die beiliegende DVD enthält den im Rahmen dieser Arbeit entstandenen Quelltext sowie die zur Auswertung erhobenen Daten. Die Struktur des Datenträgers wird durch die Abbildungen A.1 bis A.3 auf den Seiten 69–71 beschrieben.

`/`	
`ar2clipse/`	Programm, das die Bearbeitung des ArduCopter-Quelltexts mit der IDE Eclipse ermöglicht (vgl. Abschnitt 3.3.3 auf Seite 39)
`daten/`	Im Rahmen dieser Arbeit erhobene Daten (siehe Abbildung A.2 auf der nächsten Seite)
`Neurocopter.pdf`	Digitale Fassung: „Neurocopter - Eine fliegende Experimentierplattform zur Erforschung der Hirnaktivität von Honigbienen“
`neurocopteronboard/`	Bordsoftware (siehe Abbildung A.3 auf Seite 71)

Abbildung A.1.: Wurzelverzeichnis

Struktur des Wurzelverzeichnisses der beigelegten DVD. Enthalten sind der im Rahmen dieser Arbeit entstandene Quelltext und erhobene Daten, sowie eine digitale Fassung dieser Arbeit.

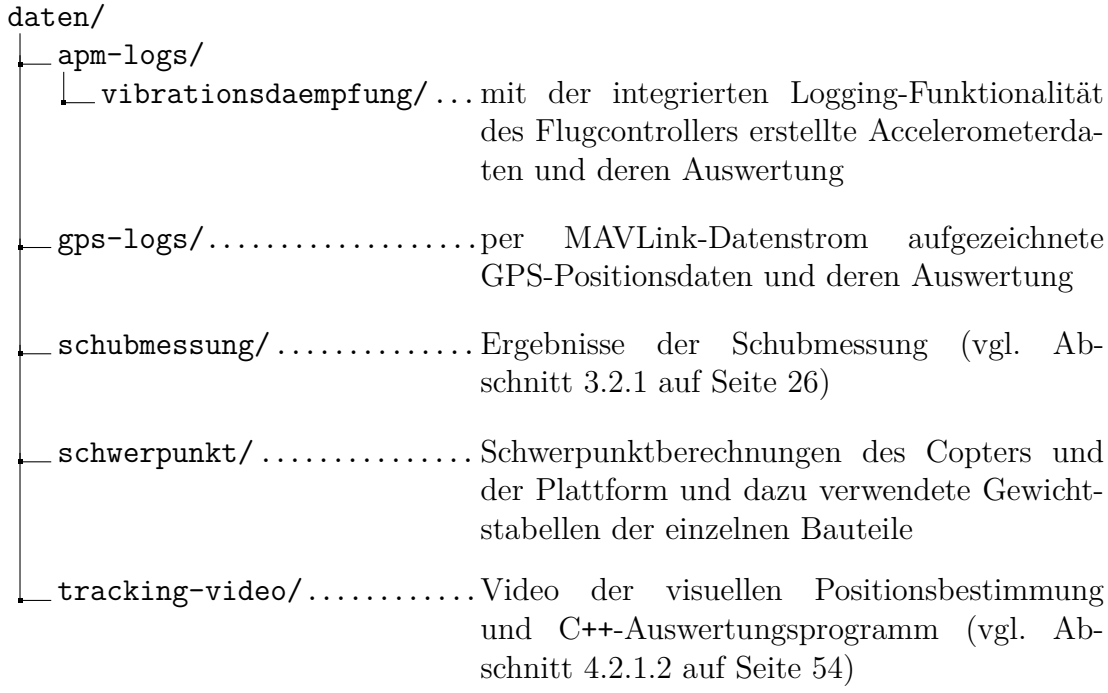


Abbildung A.2.: Datenverzeichnis

Struktur des Datenverzeichnisses, das Messwerte und deren Auswertungen beinhaltet.

- `neurocopteronboard/` — Bordsoftware
  - `berlinunited/` — Berlin United Framework (vgl. Abschnitt 3.3.2 auf Seite 37)
  - `src/`
    - `berlinunited_extensions/` — Hilfsfunktionen zur einfacheren Verwendung einiger Bestandteile des Berlin United Frameworks
    - `communication` — Klassen zur Abstraktion des Übertragungskanals (vgl. Abschnitt 3.3.4.3 auf Seite 43)
    - `libs` — generierter MAVLink-Code, MAVLink-Hilfsfunktionen, allgemeine Hilfsfunktionen, Klassen zur nebenläufigen Programmierung
    - `modules` — Module zur Verabeitung der MAVLink-Pakete
    - `representations` — von den Modulen verwendete Repräsentationen
    - `services` — Dienst zum Empfangen der MAVLink-Nachrichten (vgl. Abschnitt 3.3.4.3 auf Seite 43)
    - `tools/`
      - `mavlink_generator/` — MAVLink-Codegenerator (vgl. Abschnitt 3.3.1.2 auf Seite 35)
      - ...
    - ...
  - ...

Abbildung A.3.: Neurocopter-Verzeichnis

Allgemeine Verzeichnisstruktur des Quelltexts der Bordsoftware. Auslassungen sind durch „...“ markiert.

Literaturverzeichnis

[1] Menzel, Randolf ; Giurfa, Martin: Cognitive architecture of a mini-brain: the honeybee. In: *Trends in Cognitive Sciences* 5 (2001), Nr. 2, 62–71. DOI: 10.1016/S1364-6613(00)01601-6. – ISSN 1364–6613

[2] Chittka, Lars ; Niven, Jeremy: Are Bigger Brains Better? In: *Current Biology* 19 (2009), Nr. 21, R995–R1008. DOI: 10.1016/j.cub.2009.08.023. – ISSN 0960–9822

[3] Seeley, T.D.: *The Wisdom of the Hive: the social physiology of honey bee colonies*. Harvard University Press, 1995. – ISBN 9780674043404

[4] Frisch, Karl von: *Tanzsprache und Orientierung der Bienen*. Springer Berlin Heidelberg, 1965

[5] Riley, J. R. ; Greggers, U. ; Smith, A. D. ; Stach, S. ; Reynolds, D. R. ; Stollhoff, N. ; Brandt, R. ; Schaupp, F. ; Menzel, R.: The automatic pilot of honeybees. In: *Proceedings of the Royal Society of London B: Biological Sciences* 270 (2003), Nr. 1532, 2421–2424. DOI: 10.1098/rspb.2003.2542. – ISSN 0962–8452

[6] Menzel, R. ; Geiger, K. ; Chittka, L. ; Joerges, J. ; Kunze, J. ; Müller, U.: The knowledge base of bee navigation. In: *Journal of Experimental Biology* 199 (1996), Nr. 1, 141–146. `http://jeb.biologists.org/content/199/1/141`. – ISSN 0022–0949

[7] Esch, Harald E. ; Zhang, Shaowu ; Srinivasan, Mandyan V. ; Tautz, Juergen: Honeybee dances communicate distances measured by optic flow. In: *Nature* 411 (2001), May, Nr. 6837, 581–583. DOI: 10.1038/35079072. – ISSN 0028–0836

[8] Menzel, Randolf ; Kirbach, Andreas ; Haass, Wolf-Dieter ; Fischer, Bernd ; Fuchs, Jacqueline ; Koblofsky, Miriam ; Lehmann, Konstantin ; Reiter, Lutz ; Meyer, Hanno ; Nguyen, Hai ; Jones, Sarah ; Norton, Philipp ; Greggers, Uwe: A Common Frame of Reference for Learned and Communicated Vectors in Honeybee Navigation. In: *Current Biology* 21 (2011), Nr. 8, 645–650. DOI: 10.1016/j.cub.2011.02.039. – ISSN 0960–9822

[9] Cheeseman, James F. ; Millar, Craig D. ; Greggers, Uwe ; Lehmann, Konstantin ; Pawley, Matthew D. M. ; Gallistel, Charles R. ; Warman, Guy R. ; Menzel, Randolf: Way-finding in displaced clock-shifted bees proves bees use a cognitive map. In: *Proc. Natl. Acad. Sci. USA* 111 (2014), Nr. 24, 8949–8954. DOI: 10.1073/pnas.1408039111. – ISSN 1091–6490

[10] O'Keefe, J. ; Nadel, L.: *The hippocampus as a cognitive map.* Clarendon Press, 1978

[11] McNaughton, Bruce L. ; Battaglia, Francesco P. ; Jensen, Ole ; Moser, Edvard I. ; Moser, May-Britt: Path integration and the neural basis of the 'cognitive map'. In: *Nat Rev Neurosci* 7 (2006), Aug, Nr. 8, 663–678. DOI: 10.1038/nrn1932. – ISSN 1471–003X

[12] Fyhn, Marianne ; Hafting, Torkel ; Witter, Menno P. ; Moser, Edvard I. ; Moser, May-Britt: Grid cells in mice. In: *Hippocampus* 18 (2008), Nr. 12, 1230–1238. DOI: 10.1002/hipo.20472. – ISSN 1098–1063

[13] Cheung, Allen ; Collett, Matthew ; Collett, Thomas S. ; Dewar, Alex ; Dyer, Fred ; Graham, Paul ; Mangan, Michael ; Narendra, Ajay ; Philippides, Andrew ; Stürzl, Wolfgang ; Webb, Barbara ; Wystrach, Antoine ; Zeil, Jochen: Still no convincing evidence for cognitive map use by honeybees. In: *Proc. Natl. Acad. Sci. USA* 111 (2014), Nr. 24, E4396–E4397. DOI: 10.1073/pnas.1413581111. – ISSN 1091–6490

[14] Cheeseman, James F. ; Millar, Craig D. ; Greggers, Uwe ; Lehmann, Konstantin ; Pawley, Matthew D. M. ; Gallistel, Charles R. ; Warman, Guy R. ; Menzel, Randolf: Reply to Cheung et al.: The cognitive map hypothesis remains the best interpretation of the data in honeybee navigation. In: *Proc. Natl. Acad. Sci. USA* 111 (2014), Nr. 24, E4398. DOI: 10.1073/pnas.1415738111. – ISSN 1091–6490

[15] Wolf, Thomas J. ; Schmid-Hempel, Paul: Extra Loads and Foraging Life Span in Honeybee Workers. In: *Journal of Animal Ecology* 58 (1989), Nr. 3, 943–954. DOI: 10.2307/5134. – ISSN 00218790, 13652656

[16] Luu, Tien ; Cheung, Allen ; Ball, David ; Srinivasan, Mandyam V.: Honeybee flight: a novel 'streamlining' response. In: *Journal of Experimental Biology* 214 (2011), Nr. 13, S. 2215–2225. DOI: 10.1242/jeb.050310. – ISSN 0022–0949

[17] Taylor, Gavin J. ; Luu, Tien ; Ball, David ; Srinivasan, Mandyam V.: Vision and air flow combine to streamline flying honeybees. In: *Scientific Reports* 3 (2013), Sep, 2614. DOI: 10.1038/srep02614. – ISSN 2045–2322

[18] Evangelista, C. ; Kraft, P. ; Dacke, M. ; Reinhard, J. ; Srinivasan, M. V.: The moment before touchdown: landing manoeuvres of the honeybee Apis mellifera. In: *Journal of Experimental Biology* 213 (2009), Nr. 2, S. 262–270. DOI: 10.1242/jeb.037465. – ISSN 0022–0949

[19] Bramwell, A.R.S. ; Done, George ; Balmford, David: Basic mechanics of rotor systems and helicopter flight. In: *Bramwell's Helicopter Dynamics.* Second Edition. Oxford : Butterworth-Heinemann, 2000. DOI: 10.1016/B978-075065075-5/50004-X. – ISBN 978–0–7506–5075–5, Kapitel 1, 1–32

[20] Pounds, P. ; Mahony, R. ; Corke, P.: Modelling and control of a large quadrotor robot. In: *Control Engineering Practice* 18 (2010), Nr. 7, 691–699.

DOI: 10.1016/j.conengprac.2010.02.008. – ISSN 0967–0661. – Special Issue on Aerial Robotics

[21] Hoffmann, Gabriel M. ; Huang, Haomiao ; Waslander, Steven L. ; Tomlin, Claire J.: Quadrotor Helicopter Flight Dynamics and Control: Theory and Experiment. In: *Proceedings of the AIAA Guidance, Navigation and Control Conference and Exhibit*. Hilton Head, South Carolina : American Institute of Aeronautics and Astronautics, August 2007

[22] *Mixer Tabellen - MikrokopterWiki*. http://wiki.mikrokopter.de/mkm, Abruf: 2015-12-03

[23] Carpenter, Michael ; Bonney, Bill ; Dade, Stephen u. a.: *APM Planner v2.9.19-rc4*. http://planner2.ardupilot.com/. http://planner2.ardupilot.com/home/credits-and-contributors/, Abruf: 2015-12-09

[24] Hazelhurst, Jethro: *APM2.x Wiring QuickStart | ArduCopter*. http://copter.ardupilot.com/wiki/connecting-the-apm2/, Abruf: 2015-12-03

[25] Pounds, Paul ; Mahony, Robert ; Gresham, Joel ; Corke, Peter ; Roberts, Jonathan M.: Towards dynamically-favourable quad-rotor aerial robots. In: Barnes, Nick (Hrsg.) ; Austin, David (Hrsg.): *Australasian Conference on Robotics and Automation 2004 ACRA2004*. Australian National University Canberra : Australian Robotics & Automation Association, December 2004

[26] Bäumker, M. ; Przybilla, H.-J. ; Zurhorst, A.: Enhancements in UAV Flight Control and Sensor Orientation. In: *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences* XL-1/W2 (2013), 33–38. DOI: 10.5194/isprsarchives-XL-1-W2-33-2013

[27] Sidea, Adriana-Gabriela ; Brogaard, Rune Y. ; Andersen, Nils A. ; Ravn, Ole: General model and control of an n rotor helicopter. In: *Journal of Physics: Conference Series* 570 (2014), Nr. 5. DOI: 10.1088/1742-6596/570/5/052004

[28] *Kapitel* Development of a Low-Cost Experimental Quadcopter Testbed using an Arduino controller for Video Surveillance. In: Turkoglu, Kamran ; Ji, Ankyda: American Institute of Aeronautics and Astronautics, 2015 (AIAA SciTech)

[29] u-blox: *LEA-6 - u-blox 6 GPS Modules*. Version: 2014. https://www.u-blox.com/sites/default/files/products/documents/LEA-6_DataSheet_%28GPS.G6-HW-09004%29.pdf, Abruf: 2015-12-09

[30] *Mouse-based Optical Flow Sensor (ADNS3080) | ArduPilot*. http://copter.ardupilot.com/wiki/common-mouse-based-optical-flow-sensor-adns3080/, Abruf: 2015-12-02

[31] *PX4FLOW Smart Camera - PX4 Autopilot Project*. https://pixhawk.org/modules/px4flow, Abruf: 2015-12-02

[32] *PX4FLOW Optical Flow Camera Board | ArduPilot.* http://copter.ardupilot.com/wiki/common-px4flow-overview/, Abruf: 2015-12-11

[33] *APM 2.5 and 2.6 Overview | ArduPilot.* http://copter.ardupilot.com/wiki/common-apm25-and-26-overview/, Abruf: 2015-12-02

[34] *Camera Gimbal with Servos | ArduPilot.* http://copter.ardupilot.com/wiki/common-camera-gimbal/, Abruf: 2015-12-02

[35] *Choosing a Ground Station | ArduCopter.* http://copter.ardupilot.com/wiki/common-choosing-a-ground-station/, Abruf: 2015-12-02

[36] *Pixhawk Autopilot - PX4 Autopilot Project.* https://pixhawk.org/modules/pixhawk, Abruf: 2015-12-02

[37] *PX4FMU Autopilot / Flight Management Unit - PX4 Autopilot Project.* https://pixhawk.org/modules/px4fmu, Abruf: 2015-12-02

[38] *PX4IO Airplane/Rover Servo and I/O Module - PX4 Autopilot Project.* https://pixhawk.org/modules/px4io, Abruf: 2015-12-02

[39] *Home - PX4 Autopilot Project.* https://pixhawk.org/start, Abruf: 2015-12-02

[40] *PX4 Firmware - PX4 Autopilot Project.* https://pixhawk.org/firmware/start, Abruf: 2015-12-02

[41] *Controller boards - MultiWii.* http://www.multiwii.com/wiki/index.php?title=Controller_boards, Abruf: 2015-12-07

[42] *Hardware - MultiWii.* http://www.multiwii.com/wiki/index.php?title=Hardware, Abruf: 2015-12-07

[43] *multiwii/multiwii-firmware - GitHub.* https://github.com/multiwii/multiwii-firmware, Abruf: 2015-12-07

[44] *Flightmodes - MultiWii.* http://www.multiwii.com/wiki/index.php?title=Flightmodes, Abruf: 2015-12-07

[45] *GPS - MultiWii.* http://www.multiwii.com/wiki/index.php?title=GPS, Abruf: 2015-12-08

[46] *Multicopter Types - MultiWii.* http://www.multiwii.com/wiki/index.php?title=Multicopter_Types, Abruf: 2015-12-07

[47] *Extra features - MultiWii.* http://www.multiwii.com/wiki/index.php?title=Extra_features, Abruf: 2015-12-07

[48] *Software - MultiWii.* http://www.multiwii.com/software, Abruf: 2015-12-08

[49] *Mods - MultiWii.* http://www.multiwii.com/wiki/index.php?title=Mods, Abruf: 2015-12-07

[50] *Multiwii Serial Protocol - MultiWii.* http://www.multiwii.com/wiki/index.php?title=Multiwii_Serial_Protocol, Abruf: 2015-12-07

[51] *MultiWii Forum - Position Hold with Optical Flow sensor - done.* http://www.multiwii.com/forum/viewtopic.php?f=7&t=1413, Abruf: 2015-12-07

[52] *FlightCtrl V2.5 - MikrokopterWiki.* http://wiki.mikrokopter.de/FlightCtrl_ME_2_5, Abruf: 2015-12-02

[53] *FlightCtrlAnleitung - MikrokopterWiki.* http://wiki.mikrokopter.de/FlightCtrlAnleitung, Abruf: 2015-12-02

[54] *NaviCtrl_2.1 - MikrokopterWiki.* http://wiki.mikrokopter.de/NaviCtrl_2.1, Abruf: 2015-12-02

[55] *MikroKopterRepository - MikrokopterWiki.* http://wiki.mikrokopter.de/en/MikroKopterRepository, Abruf: 2015-12-02

[56] *MK-Parameter/Mixer-SETUP - MikrokopterWiki.* http://wiki.mikrokopter.de/MK-Parameter/Mixer-SETUP, Abruf: 2015-12-02

[57] *MK-Parameter/Camera - MikrokopterWiki.* http://wiki.mikrokopter.de/MK-Parameter/Camera, Abruf: 2015-12-02

[58] *MikroKopterTool - MikrokopterWiki.* http://wiki.mikrokopter.de/MikroKopterTool?action=show&redirect=KopterTool, Abruf: 2015-12-02

[59] *MikroKopterTool-OSD - MikrokopterWiki.* http://wiki.mikrokopter.de/MikroKopterTool-OSD, Abruf: 2015-12-02

[60] *PiMote - MikrokopterWiki.* http://wiki.mikrokopter.de/PiMote, Abruf: 2015-12-02

[61] *BL-Ctrl_Anleitung - MikrokopterWiki.* http://wiki.mikrokopter.de/BL-Ctrl_Anleitung, Abruf: 2015-12-02

[62] *KK2.1 V1.19S1 Updated Firmware & Manual.* http://www.rcgroups.com/forums/showthread.php?t=2298292, Abruf: 2015-12-12

[63] *KK 2.1 Multi-Rotor Control Board User Guide.* http://www.hobbyking.com/hobbyking/store/uploads/1026124741X1235859X24.pdf, Abruf: 2015-12-12

[64] *KK2.1 & KK2.1.5 instruction manual.* http://www.hobbyking.com/hobbyking/store/uploads/161447249X30206X38.pdf, Abruf: 2015-12-12

[65] *Mavlink (developer page) - PX4 Autopilot Project.* https://pixhawk.org/dev/mavlink, Abruf: 2015-12-15

[66] *MAVLink Commands | ArduPilot Developer.* http://dev.ardupilot.com/wiki/mavlink-commands/, Abruf: 2015-12-15

[67] *MAVLink Micro Air Vehicle Communication Protocol.* http://qgroundcontrol.org/mavlink/start, Abruf: 2015-12-15

[68] *Home - QGroundControl GCS*. http://qgroundcontrol.org/, Abruf: 2015-12-15

[69] Kirby, Simon: *tgy – Open Source Firmware for ATmega-based Brushless ESCs*. https://github.com/sim-/tgy, Abruf: 2015-12-11

[70] *Turnigy Talon Carbon Fiber Quadcopter Frame*. http://www.hobbyking.com/hobbyking/store/uh_viewitem.asp?idproduct=22397, Abruf: 2015-12-14

[71] *PYRAMID X580 Glass Fiber Quadcopter Frame*. http://www.pyramidmodels.com/shop/product.php/1318/pyramid_x580_glass_fiber_quadcopter_frame_w_camera_mount_585mm, Abruf: 2015-12-14

[72] *RotorBits HexCopter Kit*. http://www.hobbyking.com/hobbyking/store/uh_viewitem.asp?idproduct=51415, Abruf: 2015-12-14

[73] *HMF U580 Carbon Fiber Umbrella Folding Quadcopter Kit*. http://www.hobbyking.com/hobbyking/store/uh_viewitem.asp?idproduct=77173, Abruf: 2015-12-14

[74] Pounds, Paul ; Mahony, Robert ; Corke, Peter: Modelling and control of a quad-rotor robot. In: *Australasian Conference on Robotics and Automation 2006*. Auckland, New Zealand : Australian Robotics and Automation Association Inc., 2006

[75] *Turnigy Talon Quadcopter (V2.0) Carbon Fiber Frame 550mm*. http://www.hobbyking.com/hobbyking/store/uh_viewitem.asp?idproduct=36427, Abruf: 2015-12-14

[76] *PYRAMID X650F Glass Fiber Quadcopter Frame*. http://www.pyramidmodels.com/shop/product.php/1449/pyramid_x650f_glass_fiber_quadcopter_frame_550mm_folding_frame_kit, Abruf: 2015-12-14

[77] *PYRAMID T650-X4-16 FibreGlass Quadcopter*. http://www.pyramidmodels.com/shop/product.php/941/pyramid_t650_x4_16_fibreglass_quadcopter_650mm_dia, Abruf: 2015-12-14

[78] *Skylark M4-680 Quadcopter Frame 680mm*. http://www.skylarkfpv.com/store/index.php?route=product/product&product_id=112, Abruf: 2015-12-14

[79] *Raspberry Pi 2 Model B*. https://www.raspberrypi.org/products/raspberry-pi-2-model-b/, Abruf: 2015-12-01

[80] *Raspberry Pi FAQs - Frequently Asked Questions*. https://www.raspberrypi.org/help/faqs/#generalDimensions, Abruf: 2015-12-01

[81] *ODROID-U3 - Hardkernel*. http://www.hardkernel.com/main/products/prdt_info.php?g_code=g138745696275, Abruf: 2015-12-01

[82] *ODROID IO-Shield - Hardkernel*. http://www.hardkernel.com/main/products/prdt_info.php?g_code=G138760240354, Abruf: 2015-12-01

[83] *ISEE - IGEPv2 DM3730.* https://www.isee.biz/products/igep-processor-boards/igepv2-dm3730, Abruf: 2015-12-01

[84] ISEE (Hrsg.): *IGEPv2 Hardware Reference Manual.* ISEE, https://www.isee.biz/support/downloads/item/igepv2-hardware-reference-manual-2, Abruf: 2015-12-01

[85] *Banana Pi BPI - M3 Octa-core Computer.* http://www.banana-pi.org/m3.html, Abruf: 2015-12-01

[86] *Turnigy 5-7.5A (8∼42v) HV UBEC for Lipoly.* http://www.hobbyking.com/hobbyking/store/__6320__TURNIGY_5_7_5A_8_42v_HV_UBEC_for_Lipoly.html, Abruf: 2015-12-21

[87] *Rover | ArduPilot.* http://rover.ardupilot.com/, Abruf: 2016-02-01

[88] *Plane | ArduPilot.* http://plane.ardupilot.com/, Abruf: 2016-02-01

[89] *Flight Modes | ArduPilot.* http://copter.ardupilot.com/wiki/flight-modes/, Abruf: 2016-01-30

[90] *Stabilize Mode | ArduPilot.* http://copter.ardupilot.com/wiki/stabilize-mode/, Abruf: 2016-01-30

[91] *Altitude Hold Mode | ArduPilot.* http://copter.ardupilot.com/wiki/altholdmode/, Abruf: 2016-01-30

[92] *Loiter Mode | ArduPilot.* http://copter.ardupilot.com/wiki/loiter-mode/, Abruf: 2016-01-30

[93] *PosHold Mode | ArduPilot.* http://copter.ardupilot.com/wiki/poshold-mode/, Abruf: 2016-01-30

[94] *Auto Mode | ArduPilot.* http://copter.ardupilot.com/wiki/auto-mode/, Abruf: 2016-01-30

[95] *Land Mode | ArduPilot.* http://copter.ardupilot.com/wiki/land-mode/, Abruf: 2016-01-30

[96] *RTL Mode | ArduPilot.* http://copter.ardupilot.com/wiki/rtl-mode/, Abruf: 2016-01-30

[97] Schulte, Klaus L.: *Der Propeller - das unverstandene Wesen.* Version: 2007. http://klspublishing.de/ejourns/e-Journ%20Al-05%20Der%20Propeller%20das%20unverstandene%20Wesen.pdf, Abruf: 2015-12-21. (Al-05)

[98] Brandt, John B. ; Selig, Michael S.: Propeller Performance Data at Low Reynolds Numbers. In: *49th AIAA Aerospace Sciences Meeting including the New Horizons Forum and Aerospace Exposition.* (2011). DOI: 10.2514/6.2011-1255

[99] Aleksandrov, D. ; Penkov, I.: optimal gap distance between rotors of mini quadrotor helicopter. In: *8th International DAAAM Baltic Conference* (2012). http://innomet.ttu.ee/daaam_publications/2012/Aleksandrov.pdf

[100] *Create a new MAVLink Message*. http://qgroundcontrol.org/mavlink/create_new_mavlink_message, Abruf: 2015-12-23

[101] *MAVLink Data Types and Conventions*. https://github.com/mavlink/mavlink/blob/master/pymavlink/generator/C/include_v1.0/mavlink_types.h, Abruf: 2015-12-23

[102] *MAVLink Waypoint Protocol*. http://qgroundcontrol.org/mavlink/waypoint_protocol, Abruf: 2016-01-02

[103] *Building ArduPilot for APM2.x with Make | ArduPilot*. http://copter.ardupilot.com/wiki/arducopter-parameters/, Abruf: 2016-01-02

[104] *MAVLink Parameter Protocol*. http://qgroundcontrol.org/mavlink/parameter_protocol, Abruf: 2015-12-23

[105] FUmanoids: *Berlin United Framework 2014.0*. http://www.fumanoids.de/code/framework/. Version: 2014, Abruf: 2015-12-28

[106] Mellmann, Heinrich ; Xu, Yuan ; Krause, Thomas ; Holzhauer, Florian: NaoTH Software Architecture for an Autonomous Agent. In: *Proceedings of SIMPAR 2010 Workshops Intl. Conf. on SIMULATION, MODELING and PROGRAMMING for AUTONOMOUS ROBOTS* (2010), 316–327. http://www.naoteamhumboldt.de/wp-content/papercite-data/pdf/scpr-mellmannxuetal-10.pdf. ISBN 978–3–00–032863–3

[107] Mellmann, Heinrich ; Scheunemann, Marcus ; Burkhard, Hans-Dieter ; Hafner, Verena: Berlin United - NaoTH 2014. (2014). http://fei.edu.br/rcs/2014/TeamDescriptionPapers/StandardPlatform/bu-naoth-tdp14-final.pdf, Abruf: 2015-12-26

[108] FUmanoids: *Berlin United Code Release, 2014.0*. http://www.fumanoids.de/publications/coderelease. Version: January 2014, Abruf: 2015-12-28

[109] Hohberg, Simon: *Interactive Key Frame Motion Editor for Humanoid Robots*, Freie Universität Berlin, Bachelorarbeit, February 2012. https://maserati.mi.fu-berlin.de/fumanoids/wp-content/papercite-data/pdf/hohberg2012.pdf, Abruf: 2015-12-28

[110] *ArduPilot Development Site | ArduPilot Developer*. http://dev.ardupilot.com/, Abruf: 2015-12-30

[111] *Arduino Library Tutorial*. https://www.arduino.cc/en/Hacking/LibraryTutorial, Abruf: 2016-01-01

[112] *Lots of changes to APM development - DIY Drones*. http://diydrones.com/profiles/blogs/lots-of-changes-to-apm-development, Abruf: 2015-12-30

[113] *History of ArduPilot | ArduPilot Developer*. http://dev.ardupilot.com/wiki/history-of-ardupilot/, Abruf: 2015-12-30

[114] *Building ArduPilot for APM2.x with Make | ArduPilot Developer*. http://dev.ardupilot.com/wiki/building_with_make/, Abruf: 2015-12-31

[115] *Arduino FAQ*. https://www.arduino.cc/en/Main/FAQ, Abruf: 2015-12-30

[116] *Arduino Software (IDE)*. https://www.arduino.cc/en/Guide/Environment, Abruf: 2015-12-31

[117] *Arduino Build Process*. https://www.arduino.cc/en/Hacking/BuildProcess, Abruf: 2015-12-30

[118] *ardupilot/common.h · diydrones/ardupilot · GitHub*. https://github.com/ArduPilot/ardupilot/blob/ArduCopter-3.2.1/libraries/GCS_MAVLink/include/mavlink/v1.0/common/common.h, Abruf: 2016-01-05

[119] *ardupilot/mavlink_msg_request_data_stream.h · diydrones/ardupilot · GitHub*. https://github.com/ArduPilot/ardupilot/blob/ArduCopter-3.2.1/libraries/GCS_MAVLink/include/mavlink/v1.0/common/mavlink_msg_request_data_stream.h, Abruf: 2016-01-04

[120] Kaplan, E. ; Hegarty, C.: *Understanding GPS: Principles and Applications*. Artech House, 2005. – ISBN 1–58053–894–0

[121] *Atmel ATmega2560*. http://www.atmel.com/devices/atmega2560.aspx, Abruf: 2016-03-04

[122] *MAVLink micro air vehicle marshalling / communication library · mavlink/-mavlink · GitHub*. https://github.com/mavlink/mavlink, Abruf: 2016-03-07

[123] Lippman, Stanley B.: *Inside the C++ Object Model*. Addison Wesley Publishing Company, 1996. – ISBN 0–201–83454–5

[124] *ardupilot/libraries/AP_AHRS/AP_AHRS_NavEKF.h · diydrones/ardupilot · GitHub*. https://github.com/ArduPilot/ardupilot/blob/ArduCopter-3.2.1/libraries/AP_AHRS/AP_AHRS_NavEKF.h, Abruf: 2016-04-04

[125] *ardupilot/libraries/AP_HAL/AP_HAL_Boards.h · diydrones/ardupilot · GitHub*. https://github.com/ArduPilot/ardupilot/blob/ArduCopter-3.2.1/libraries/AP_HAL/AP_HAL_Boards.h, Abruf: 2016-04-04

Nomenklatur

APM	ArduPilotMega
CFK	Kohlenstofffaserverstärkter Kunststoff
EKF	Erweiterter Kalman-Filter (Extended Kalman Filter)
ESC	Motorregler (Electronic Speed Control)
GFK	Glasfaserverstärkter Kunststoff
GPIO	Allzweckeingabe/-ausgabe (General Purpose Input/Output)
GPS	Globales Positionsbestimmungssystem (Global Positioning System)
IDE	Integrierte Entwicklungsumgebung (Integrated Development Environment)
PVC	Polyvinylchlorid
UART	Serielle Schnittstelle (Universal Asynchronous Receiver/Transmitter)