Höhere Technische Bundeslehr- und Versuchsanstalt Rankweil

Höhere Lehranstalt für Elektronik
Schuljahr 2016/17

Diplomarbeit

Self Controlled Aeronautic Vessel

Projektteam:
Benjamin Beinder
Gabriel Häusle
Felix Halbwedl

Betreuer:
Prof. Dipl.-Ing. Christoph Stüttler

Rankweil, am 01.04.2017

1 Eidesstattliche Erklärung

Wir versichern an Eides statt, dass wir alle entsprechend gekennzeichneten Teile der vorliegenden Diplomarbeit selbstständig verfasst, keine anderen als die angegebenen Quellen und Hilfsmittel benutzt, sowie alle wörtlich oder sinngemäß übernommenen Stellen in der Arbeit gekennzeichnet haben. Ferner gestatten wir der Höheren Technischen Lehr- und Versuchsanstalt Rankweil (HTLBuVA Rankweil), die vorliegende Diplomarbeit unter Beachtung insbesondere datenschutzrechtlicher und wettbewerbsrechtlicher Vorschriften für Lehre und Forschung zu benutzen.

Declaration of Oath

We hereby declare by oath that all appropriately labelled parts of our thesis were independently written by ourselves, no other than the indicated sources and tools have been used and that all parts of the thesis which have been borrowed, either literally or in an analogous manner, have been properly cited. Furthermore, we permit the Higher Technical College and Laboratory of Rankweil (Höhere Technische Bundeslehr- und Versuchsanstalt Rankweil) to use the thesis for teaching and research, with due regard to matters of privacy and antitrust regulations.

Rankweil, am _____

_____ _____
Benjamin Beinder Gabriel Häusle

_____
Felix Halbwedl

2 Abstract

Deutsch

Das Ziel ist es, ein autonomes Prallluftschiff (Blimp) zu bauen. Dieses Luftschiff soll Koordinaten über das GSM-Netz empfangen und über GPS mithilfe einer Regelung das so spezfizierte Ziel auf möglichst kurzem Weg anfliegen. Die Regelung arbeitet zum Einen mit den GPS Daten, zum anderen aber auch mit den Daten der Sensorik (Kompass, Beschleunigungssensor, Gyrosensor). Der Antrieb soll über drei Brushless-Motoren realisiert werden, wovon sich zwei an einer Achse, welche über einen Servo-Motor gedreht werden kann, befinden. Dadurch kann der Blimp sowohl Schub nach oben als auch Schub nach vorne generieren. Der dritte Motor soll seitliche Windböen ausgleichen bzw. für die Rotation des Blimps behilflich sein. Die Hülle soll möglichst leicht sein und mit Helium gefüllt werden.

English

The goal is to construct a non-rigid airship (blimp). This airship is controlled through GPS coordinates which are transmitted over the GSM network and used to calculate the defined path. Additionally the controller uses data from a compass, accelerometer and gyro sensor to stabilize itself while in midair. Three brushless DC motors are used for propulsion and maneuvering. One of them is positioned at the rear for a better rotational ability. The other two BLDCs are fitted on a through a servo motor rotatable axis beneath the center of the airship, enabling it to create upward as well as forward thrust. Depending on the adjustment of these motors the airship can be moved in any direction. Uplift will be created by filling the hull with helium.

3 Kurzfassung

Deutsch

Das Ziel ist es, ein autonom fliegendes Prallluftschiff (Blimp) zu konstruieren, welches bei ruhigem Wind im Außenbereich fliegen kann. Dazu muss besonders auf das Gewicht der verwendeten Komponenten geachtet werden. Der durch das Helium in der Hülle generierte Auftrieb bringt das Luftschiff in eine schwebende Lage. Die Positionsänderung erfolgt dann über die Antriebsrotoren.

Die Hülle wird aus einer speziellen leichten Folie hergestellt, die auf einer Seite silber-beschichtet ist um möglichst wenig Wärme durch die Sonneneinstrahlung aufzunehmen. Auf der anderen Seite ist die Folie verschweissbar. Dies ermöglicht den Bau der Hülle, durch Verschweissen in einer bestimmten Form, von zwei Folienstreifen. Aufgrund der aerodynamischen Form ist eine Volumen Berrechnung nur annähernd durchführbar. Der 3m lange Blimp soll über ein Füllvolumen von ca. 600 ml verfügen.

Der GD32F100C8, ein nachgebauter 32Bit Microcontroller der Firma ST-Microelektronik, wird als zentrales Steuerelement der Elektronik verwendet. Mit seiner maximaler Betriebsfrequenz von 108MHz und 7 Timern sowie 2xI2C, 2xSPI und 3xUSART Schnittstellen ist er für die Regelung und Kommunikation mit den diversen Sensoren gut geeignet. Die Firmware wird mit der Programmiersprache C in der Programmierumgebung TrueStudio und Eclipse programmiert.

Versorgt werden die Elektronik und Motoren über einen 7.4V 1500mAh LiPo Akku, der mit seinen 100g einen Großteil des Gewichts ausmacht. Die Motoren werden direkt mit der Akkuspannung betrieben, während die Elektronik über 3.3V und 4V versorgt wird.

Die Steuerung des Luftschiffes erfolgt über SMS, über das GSM-Netz, wobei Koordinaten vorgegeben werden, welche anschließend automatisiert abgeflogen werden. Eine grafische Oberfläche soll die Eingabe der Koordinaten des Pfades und das Senden dieser erleichtern. Die Automation des Blimps wird über einen zentralen Regler auf dem Microcontroller erreicht, welcher die Daten des MPU6050, ein Gyro-, Beschleunigungs- und Temperatursensor, des PAC6, ein sehr kleines und genaues GPS Modul, und des HMC5883L, ein Kompass IC, verwendet um das Luftschiff auf der Flugbahn zu halten.

Drei brushless DC Motoren mit Propeller werden vom Regler direkt gesteuert, um die für die Bewegung benötigte Schubkraft zu erzeugen. Zwei dieser Motoren sind auf einer über einen Servomotor um 180° drehbaren Achse unter dem Luftschiff angebracht. Durch das Umkehren der Drehrichtung, einstellbare Schubkraft auf beiden Motoren und eine Versetzung der Achse mit dem Schwerpunkt, kann eine Rotation um die 3-Achsen, sowie eine Beschleunigung in 2 Richtungen erzeugt werden. Der dritte Motor wird am Heck des Blimps angebracht um eine höhere Manövrierfähigkeit zu erreichen und Seitenwind besser auszugleichen.

Um die Funktionalität der Software zu testen verfügt das Prallluftschiff zusätzlich über ein HC-05 Bluetoothmodul. Dies dient ledlglich für Tests mit kleinen Reichweiten. Ein C Programm sorgt dabei für einen möglichst komortablen Testmodus.

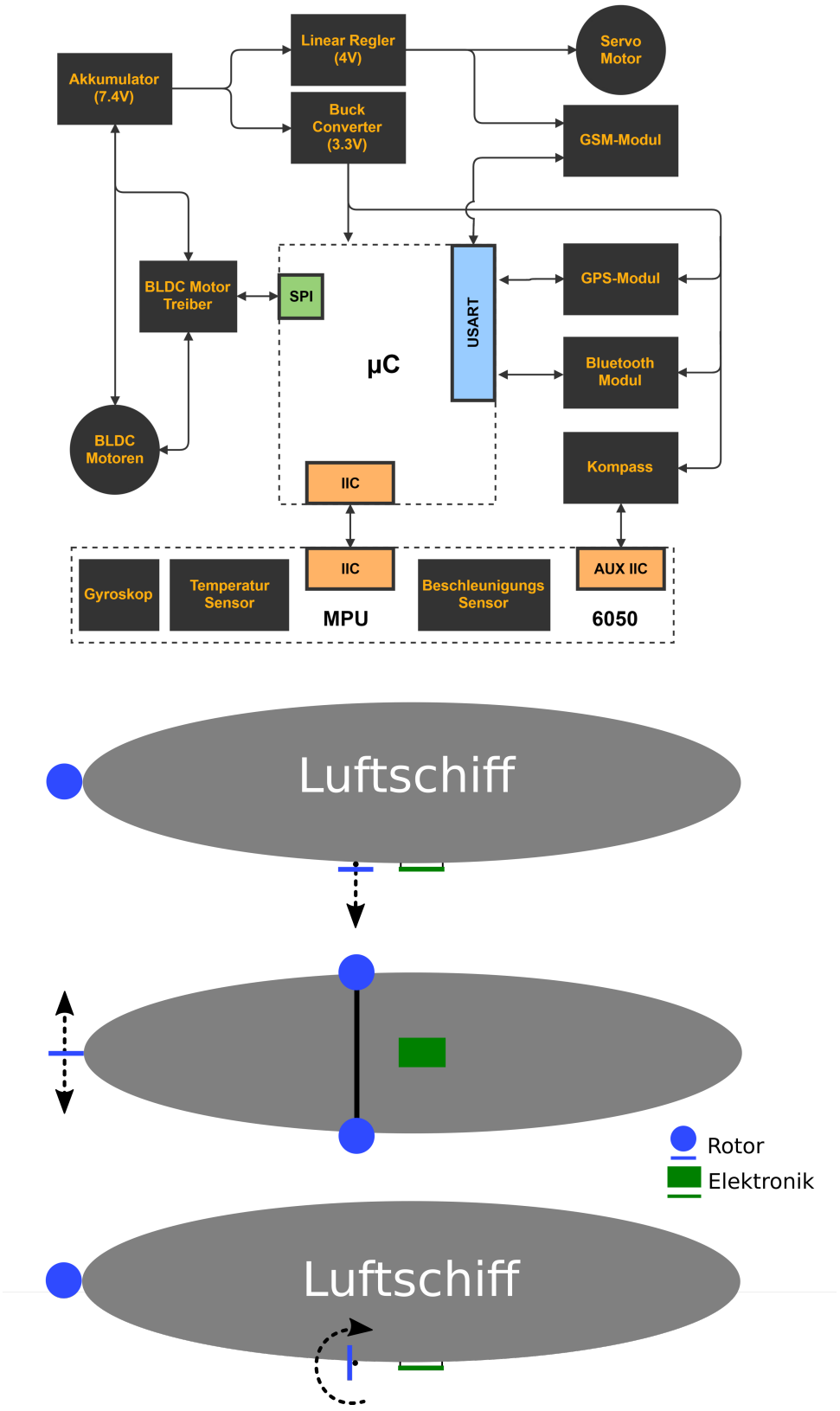


Abbildung 1: Funktionsprinzip

English

The goal of the project is to construct a self controlled aeronautic vessel, which is able to fly outdoors in calm wind. A major factor which must be minded for the functionality of the non-rigid airship is the weight of the used components. Uplift will be created through a helium filled hull, bringing the vessel in a floating state. By the use of three propulsion rotors a change of positions is achieved.

A special lightweight foil is used for the hull construction. One side is silver coated to counteract temperature differences created by the sun while the other side can be welded. This makes it possible to build a hull by welding two stripes of foil into a particular shape. Volume calculation of this aerodynamic shape proofs rather difficult which makes it only possible to estimate it. The airship is 3m in length and has an estimated volume of 600ml.

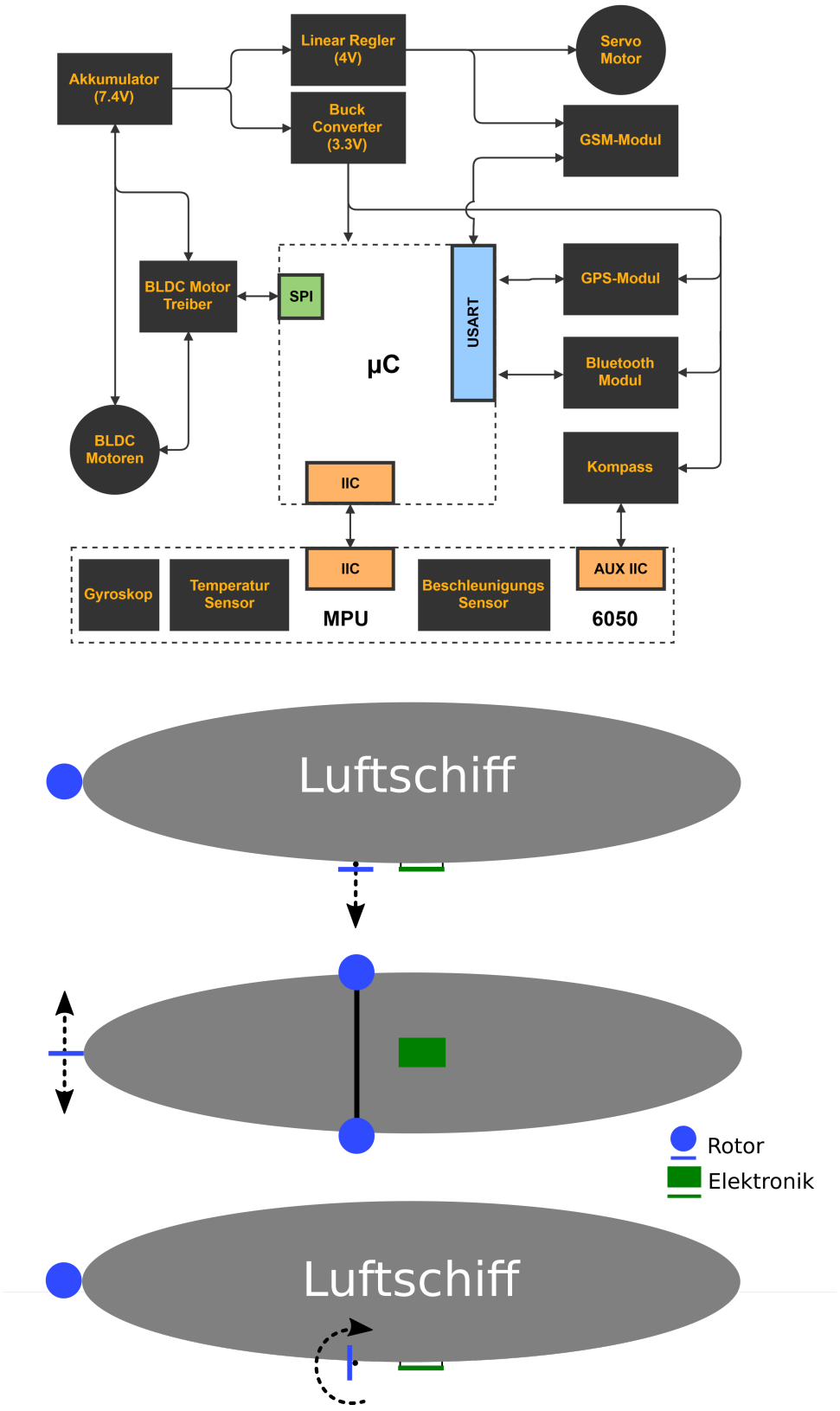
As central control unit the GDF1003C8, a recreated version of the 32bit Microcontroller by ST-Microelectronics, is used. Its core frequency of 108MHz and 7 Timers as well as 2xI2C, 2xSPI and 3xUSART interfaces make it a preferable choice as controller and communicator with the variety of sensors used in the project. Attolic TrueStudio based on Eclipse is used to programm the firmware in the programming language C.

For the power source a 7.4V 1500mAh LiPo Akku is used. Its weight of 100g is a big part of the airships load. While the motors are directly powered over the akkus voltage, the supply for the electronics are converted to be 3.3V and 4.V.

The airship can be navigated by sending a text via the GSM network containing coordinates which are followed automatically by the vessel. To make the input of those coordinates easier a graphical interface for the phone is developed. A central controller on the microcontroller is responsible for the automation proccess which uses the data gathered from the MPU6050, a gyro-, temperature-, and acceleration sensor, the PAC6, a very small and precise GPS module and the HMC5883L, a compass IC. Combining this data is then used to keep the airship on its track by adjusting the motors thrust.

Three brushless DC motors with propellor are controlled through a driver IC by the microcontroller to create the neccessary propulsion. Two of these motors are fixed on a axle underneath the vessel, which can be rotated for 180° with a servo motor. By reversing the direction of rotation, changing the thrust of the motors and dislocating the axle from the centre of gravity a 3-axis rotation as well as a acceleration in 2 directions can be created. The third motor is placed at the rear of the vessel to improve maneuvering and comensate drifts from crosswind.

To test the functionality of the airship a HC-05 module is added to the electronics. This allows for short range communication and debugging of the micrcontroller software. For this purpose a C programm is developed, allowing easy and comfortable testing.



Principle of Function

Inhaltsverzeichnis

4 Pflichtenheft

4.1 Motivation

Im fünften Jahrgang sind die Diplomarbeiten zu absolvieren. Im Zuge dessen wurde entschieden, ein Pralluftschiff kleiner Größe zu realisieren, welches befähigt sein soll, im Innenraum oder bei guter Witterung im Freien eigenständig einfache Manöver durchzuführen bzw. vorgegebene Strecken abzufliegen. Hauptaugenmerk dabei ist die Automation des Luftschiffes.

4.2 Beteiligte Personen

4.2.1 Gruppenmitglieder

Benjamin Beinder

Projektleiter & Hardware

Gabriel Häusle

Software & Mechanik

Felix Halbwedl

Regelung

4.2.2 Betreuung

Prof. Dipl.-Ing. Christoph Stüttler

Projektbetreuer

MSFC Rheintal

Beratung zur Mechanik

Firma Air Liquide

Sponsoring Ballongas

4.2.3 Zielsetzung

Das "eigenständige einfache Manöver" bedeutet in diesem Zusammenhang das Anfliegen von Koordinaten, welche dem Luftschiff per SMS übermittelt werden. Abgesehen davon soll das Luftschiff auf einer, wenn dies denn möglich ist, konstanten Höhe Richtung Ziel fliegen, ohne in Schieflage oder gar ins Trudeln zu geraten. Diese Anforderungen machen ein ausgefeiltes Regelungssystem notwendig, welches digital über einen leistungsstarken Mikrocontroller realisiert wird.

4.2.4 Nichtziele

Nicht erwünschenswert wäre es, sollte das Luftschiff befähigt sein, Hindernissen in dessen Flugbahn auszuweichen. Deren Erkennung zu realisieren wäre viel zu aufwändig, vor allem im Hinblick der begrenzten Tragkraft der Hülle. Dementsprechend ist dafür zu sorgen, dass die Flugbahn zum Ziel frei von Hindernissen ist. Eingesetzt wird das Luftschiff nur unter günstigen Bedingungen, sodass eine Schutzverkleidung der Gondel dem Luftschiff nur nicht notwendiges Gewicht hinzufügt.

4.2.5 Optionale Ziele

Grundsätzlich lassen sich die durchführbaren Flugmanöver beliebig erweitern, so könnten zum Beispiel auch Kreise, Achter und andere Figuren geflogen werden. Reicht die Tragkraft der Hülle aus, kann eine Kamera an der Gondel montiert werden. Sollte das Projekt der Zielsetzung entsprechend fertiggestellt sein, so sind die oben genannten Möglichkeiten nur zwei von vielen, das Luftschiff auszubauen.

4.3 SMART-Analyse

4.3.1 Spezifisch

Es soll ein Prallluftschiff mit einer Länge von etwa drei Metern konstruiert werden. Die Hülle werden wir selbst konstruieren, ebenso wie die Gondel, welche die Sensorik, Aktorik sowie die Steuerung und Versorgung beinhaltet. Manövriert wird das Luftschiff über zwei Rotoren, an einer Achse drehbar angebracht links und rechts von der Gondel.

4.3.2 Messbar

Der Erfolg des Projekts hängt davon ab, inwieweit die gegebenen Anforderungen, insbesondere was die Manövrierbarkeitdes Luftschiffs betrifft, erfüllt werden können. Die Aspekte der Manövrierbarkeit bauen aufeinander auf, so muss das Luftschiff, um überhaupt in gerader Linie zu fliegen, zunächst im Stillstand eine konstante Höhe halten können.

4.3.3 Attraktiv

Natürlich war die Vorstellung von einem kleinen Luftschiff, welches auf Kommmando zum genannten Ort fliegt, eine unserer Hauptgründe, sich für dieses Projekt zu entscheiden, doch abgesehen davon ist dieses Projekt die ideale Gelegenheit, unsere Kenntnisse vor allem im Bereich der Regelungstechnik zu vertiefen.

4.3.4 Realistisch

Ohne jeden Zweifel besteht das Luftschiff aus einigen kritischen Komponenten, welche den Ausgang dieses Projekts in recht ungünstiger Weise beeinflussen könnten. Besonders hervorzuheben sei an dieser Stelle die Hülle des Luftschiffs, welche zum Beispiel bei einem Zusammenstoß schwer beschädigt werden könnte. Ebenfalls sehr erwähnenswert wären die die gegebenen Beschränkungen, hervorgerufen etwa durch die Auftriebskraft, die Höchstleistung der Motoren und Weitere, die es dem Regelungssystem unmöglich machen könnten, dessen Aufgaben zu erfüllen. Doch auch wenn obige Aspekte nicht zu unterschätzen sind, so sind es vor allem die in dieser Lehranstalt erworbenen Kenntnisse und Fähigkeiten, sowie der bei allen Teammitgliedern vorhandene Wille, welche eine positive Betrachtungsweise des Projektverlaufs vernünftig und realistisch machen.

4.3.5 Terminiert

Das Projekt wurd am 13.09 gestartet und ist bis zum 22.04.2016. fertigzustellen. Grundlegende Funktionen, etwa die Flugfähigkeit des Luftschiffes, sollten bis vor Weihnachten erfüllt sein.

4.4 Beschreibung des Produkts

Funktionsbeschreibung

Manövriert wird das Luftschiff über zwei Rotoren, schwenkbar um die Querachse, deren Neigung abhängig von der zu erreichenden Fluggeschwindigkeit eingestellt wird. Diese bestimmen nicht nur die Fluggeschwindigkeit, sondern auch die Flughöhe des Luftschiffes. Damit das Luftschiff im Falle einer Fehlfunktion nicht unkontrolliert aufsteigt, übertrifft die Gewichtskraft die Auftriebskraft geringfügig, sodass für das Halten der Flughöhe nur eine minimale Leistung aufgewendet werden muss.

Ein typisches Flugmanöver soll in etwa wie folgt ablaufen: Nach Erhalt der Zielkoordinaten über das Mobilfunk-Modul begibt sich das Luftschiff zunächst auf der Stelle schwebend in die Höhe des Zielortes. Anschließend dreht sich das Luftschiff derart, dass dessen Bug in die Richtung des Zieles weist. Die dafür benötigten Daten werden vom Microcontroller aus den Messdaten des GPS-Moduls ermittelt. Die Ausrichtung wird von einem Kompass-Modul überprüft. Nach dem Vorgang der Ausrichtung fliegt das Luftschiff auf gerader Linie Richtung Ziel, wenn nötig, werden Bahnkorrekturen vorgenommen.

Zur Gewährleistung der Stabilisierung des Luftschiffes dient ein Gyrosensor, welcher in der Schwerelinie des Luftschiffes liegt.

4.4.1 Blockschaltbild

Das Blockschaltbild in Abbildung 2 deutet die Struktur der elektronischen Systeme des Luftschiffes an:

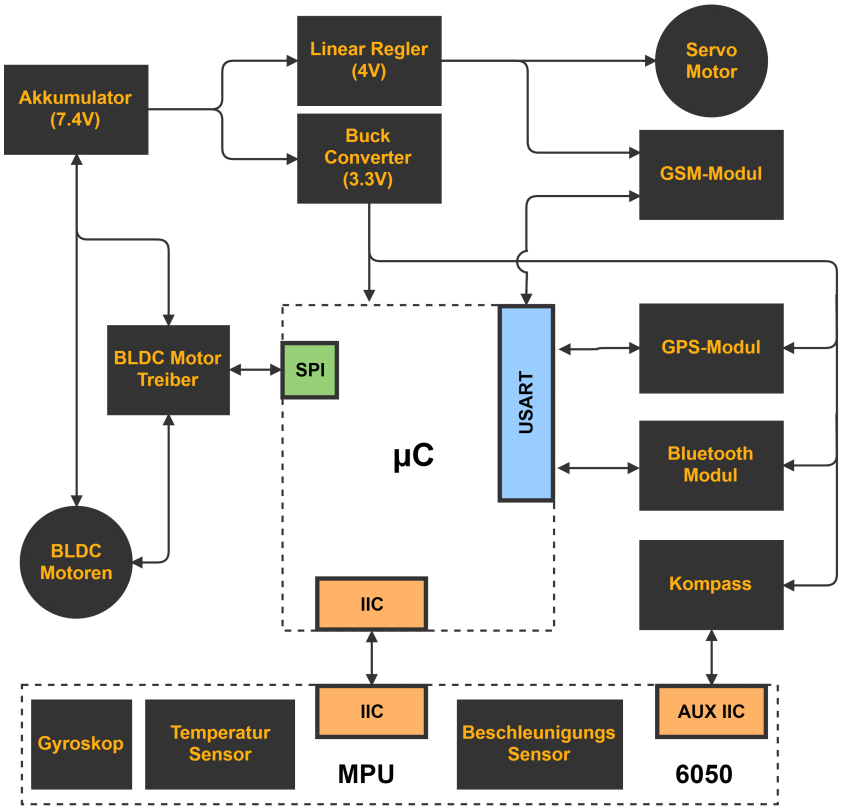


Abbildung 2: SCAV Blockschaltbild

4.4.2 Komponenten

Die verbauten elektronischen Komponenten und ihre Funktion werden in Tabelle 1 kurz zusammengefasst.

	Bezeichnung	**Interface**	**Funktion**
1x	2000 mAh 7.4V LiPo	-	Energieversorgung des Luftschiffes
1x	Buck-Konverter	-	Versorgung für die Elektronik
1x	Linearregler	-	Versorgung für das GSM-Modul
1x	µC: STM32F1003CB	2xI2C, 3xUSART, 2xSPI	Zentrale Steuereinheit
3x	A4963 Motortreiber	SPI	Treiber für die Antriebs-Motoren
1x	PA6C GPS-MODUL	USART	Positionsermittlung
1x	SIM800L GSM-MODUL	USART	Kommunikation zwischen Blimp & Benutzer
1x	HC06 Bluetooth-MODUL	USART	Debugschnittstelle
1x	MPU6050 Gyro-, Beschleunigungs- & Temperatursensor	I2C	Messung der Bewegung & Ausrichtung
1x	HMC5883L Kompass IC	I2C	Messung der Ausrichtung

Tabelle 1: Komponentenbeschreibung

5 Hardware

5.1 Allgemein

Die Hardware des Projektes besteht aus diversen Sensoren, dem Mikrocontroller als zentrale Steuereinheit, HF-Modulen zur Kommunikation über kurze bzw. längere Distanzen und Treibern für die Antriebsmotoren. Das Blockschaltbild in Abbildung 3 zeigt dabei die Funktion der Hardware.

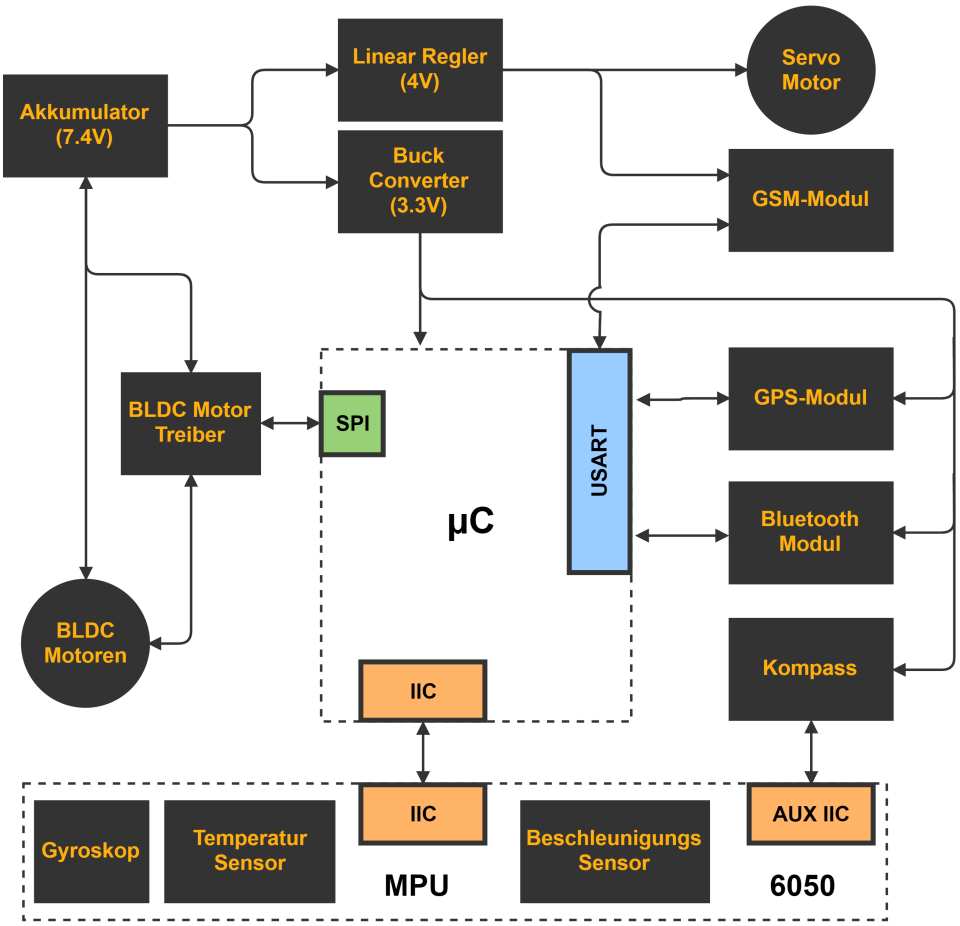


Abbildung 3: SCAV Blockschaltbild

Als Versorgung der Elektronik dient ein LiPo Akku mit 1500mAh und zwei Zellen, was einer Spannung von 7.4V entspricht. Diese Spannung wird direkt verwendet, um die BLDC Motoren und ihre zugehörigen Treiber zu versorgen.

Die BLDC Antriebsmotoren sind mit ihren drei Phasen an jeweils drei Halbbrücken angeschlossen, welche vom Motortreiber gesteuert werden. Die Elektronik wird mit 3.3V und 4V versorgt, wobei die zuerst genannte Spannung durch einen Buck Converter und die letztere über einen Linearregler erzeugt wird. Die 4V werden für die Versorgung des GSM-Moduls sowie des Servo Motors benötigt. Alles andere wird mit 3.3V betrieben. Das GPS-Modul, GSM-Modul, der Kompass, der Gyrosensor sowie der Beschleunigungssensor sind über ihre entsprechenden Interfaces mit dem µC verbunden und liefern Daten für die Regelung bzw. Steuerung. Zusätzlich verfügt der µC über die Möglichkeit, die Motortreiber über SPI einzustellen. Der Mikrocontroller wird so programmiert, dass er als Treiber für den Servo Motor agiert.

5.2 Akkumulator

5.2.1 Charakteristik

Der Akkumulator dient der Versorgung der Elektronik und der Motoren des Luftschiffes. Mit 100g Gewicht ist er eine der schwersten Komponenten und macht einen Großteil des zu tragenden Gewichtes aus. Bei der Auswahl des Akkumulators musste ein Kompromiss zwischen Betriebsdauer und Gewicht getroffen werden. Eine genaue Abschätzung dieser Betriebsdauer ist nicht möglich, da sie abhängig von vielen Faktoren wie Windstärke und Komplexität des abzufliegenden Pfades ist. Eine ungefähre Schätzung beläuft sich auf drei Stunden.

Verwendet wird ein Akkumulator von *ZOP Power* mit 1500mAh. Er verfügt über zwei Zellen, was einer Spannung von 7.4V entspricht und einem maximalen Entladestrom von 20C (= 30A siehe Gleichung 5.1).

Abbildung 4: Akkumulator

5.2.2 Spannungsmessung

Da LiPo Akkumulatorzellen niemals komplett entladen werden dürfen, war eine Spannungsmessung des Akkustandes über einen ADC des Mikrocontrollers vorgesehen, welche aufgrund von Zeitersparnis jedoch nicht eingebaut wurde.

5.2.3 Laden eines LiPo

Lithium-Polymer-Akkumulatoren können nicht nur durch Entladen, sondern auch Überladen zerstört werden. Auch ist es ungünstig, wenn die einzelnen Zellen des Akkus eine unterschiedlieche Spannung vorweisen. Daher werden solche Akkumulatoren generell nur mit sogenannten Balance-Ladegeräten aufgeladen, welche den etwas komplizierteren Ladevorgang und das Ausbalanzieren der Spannung in den Zellen übernehmen.

Beim Ladevorgang wird der Akku zuerst mit einem konstanten Strom von 1C geladen und nach Erreichen der maximalen Spannung mit dieser Spannung konstant weitergeladen, während der Ladestrom absinkt.

$$1C = \frac{I}{Q} = \frac{1500mA}{1500mAh} \qquad 0.5C = \frac{I}{Q} = \frac{750mA}{1500mAh} \qquad 2C = \frac{I}{Q} = \frac{3000mA}{1500mAh} \tag{5.1}$$

5.3 Buck Converter

5.3.1 Charakteristik

Um die für einen Großteil der Elektronik benötigte Spannung von 3.3V zu erreichen, wird ein einstellbarer Buck Converter verwendet. Dabei handelt es sich um den Step-Down Regulator MCP16331 von Microchip mit einem Eingangsspannungsbereich von 4.4V - 50V und einem minimalen Ausgangsstrom von 500mA, abhängig von der Einstellung des IC.

5.3.2 Beschaltung

Um die 3.3V zu erreichen, wird der Buck Converter wie in Abbildung 5 beschalten.

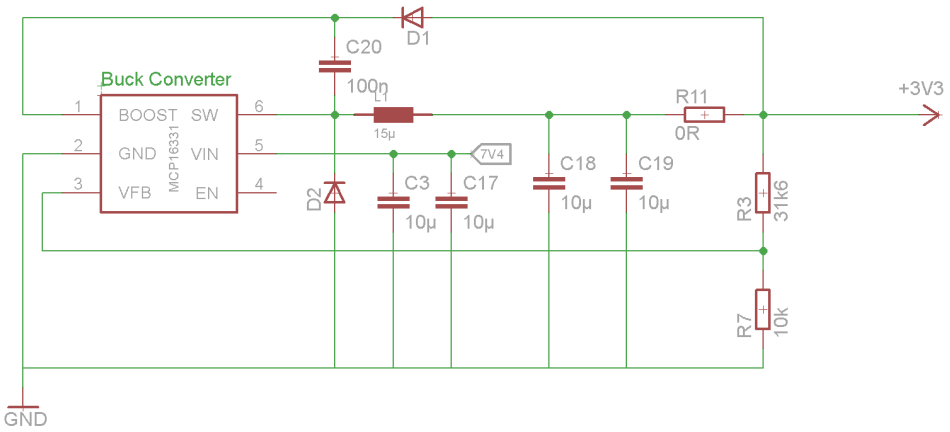


Abbildung 5: Beschaltung Buck Converter

5.3.3 Ausgangsstrom

Der minimale Ausgangsstrom des Spannungswandlers ist für die angeschlossene Elektronik, bei gleichzeitigem Betrieb von allen zu versorgenden Komponenten, ausreichend. Eine Schätzung ergibt einen maximal benötigten Strom von 300mA, im Falle eines kompletten Resets und Neustarts der Module.

5.3.4 Bauteile & Berechnungen

In Abbildung 5 sind die Stützkondensatoren C3, C17, C18 und C19, für welche 10µF Folienkondensatoren der Bauform 0805 verwendet werden, zu erkennen. R11 stellt lediglich eine 0Ω Brücke für die Einfachheit des Layouts dar. Zusätzlich wird ein 100nF Boost Kondensator am Boost Pin eingesetzt. Die Ausgangsspannung des Converters wird über die Spannung am Feedback Pin VFB bestimmt. Dazu wird R7 des Spannungsteilers auf 10kΩ festgelegt und R3 wie in Gleichung 5.3 berechnet.

$$R_7 = 10k\Omega \qquad V_{FB} = 0.8V \tag{5.2}$$

$$V_{OUT} = 3.3V\Omega \qquad R_3 = R_7 \cdot \left(\frac{V_{OUT}}{V_{FB}} - 1\right) = 31.6k\Omega \tag{5.3}$$

5.3.5 Induktivität

Als Induktiviät wird eine Spule in SMD Bauform mit dem vom Hersteller empfohlenen Standardwert bei einer Spannung von 3.3V verwendet. Diese Standardwerte werden in Tabelle 2 gezeigt.

V_{OUT}	K	$L_{STANDARD}$
2.0V	0.20	10 µH
3.3V	0.22	15 µH
5.0V	0.23	22 µH
12V	0.21	56 µH
15V	0.22	68 µH
24V	0.24	100 µH

Tabelle 2: MCP16331: Empfohlene Induktivitäten[1]

5.3.6 Dioden

Bei den verwendeten Dioden handelt es sich um MBR1H100SFT3G Shottky Dioden von ON Smiconductor, welche für einen maximalen Strom von 1A und einer DC Sperrspannung von 100V vorgesehen sind. Um Komplexität zu vermeiden, wurde auch für die Diode am Boost Pin eine dieser Schottky Dioden verbaut, was sich jedoch nicht negativ auf das Verhalten der Schaltung einwirkt, sondern lediglich die Boost Versogungsspannung für den Gate Treiber erhöht.

5.3.7 Funktion

Abbildung 6 zeigt die Funktion des Abwärtswandlers. Der Schalter in Form eines FETs wird im selben Verhältnis wie $\frac{V_{OUT}}{V_{IN}}$ geschlossen bzw. geöffnet. Ist der Schalter geschlossen, steigt der Strom I_L durch die Induktivität über die Zeit an. Sobald er sich öffnet, sinkt dieser Strom wieder. Dadurch ergibt sich ein mittlerer Ausgangsstrom I_{OUT} und eine mittlere Spannung V_{OUT}.

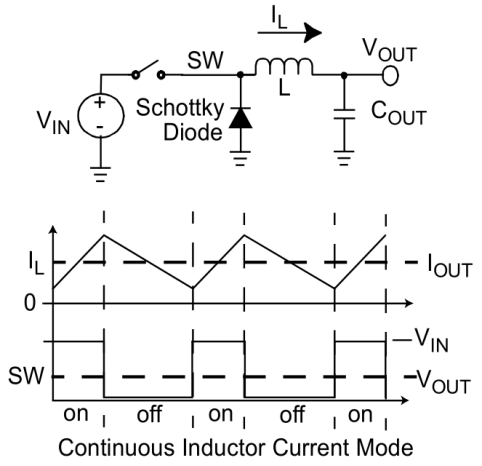


Abbildung 6: Funktion des Buck Converters[1]

5.4 Linearregler

5.4.1 Charakteristik

Da das verwendete GSM-Modul (siehe 5.9) eine Versorgungsspannung im Bereich von 3.4V - 4.4V und eine stabile Stromversorgung bis ca. 1A benötigt, wird ein zusätzlicher Linearregler verbaut, der diese Anforderungen erfüllt. Der LP3878-ADJ von Texas Instruments hat einen Eingangsspannungsbereich von 2.5V - 16 und Ausgangsspannungs- von 1V - 5.5V. Der maximale Ausgangsstrom ist mit 800mA bemessen, was aber nach Tests mit dem Modul als ausreichend befunden wurde.

5.4.2 Beschaltung

Abbildung 7 zeigt die Beschaltung des Linearreglers für eine Ausgangsspannung von 4V.

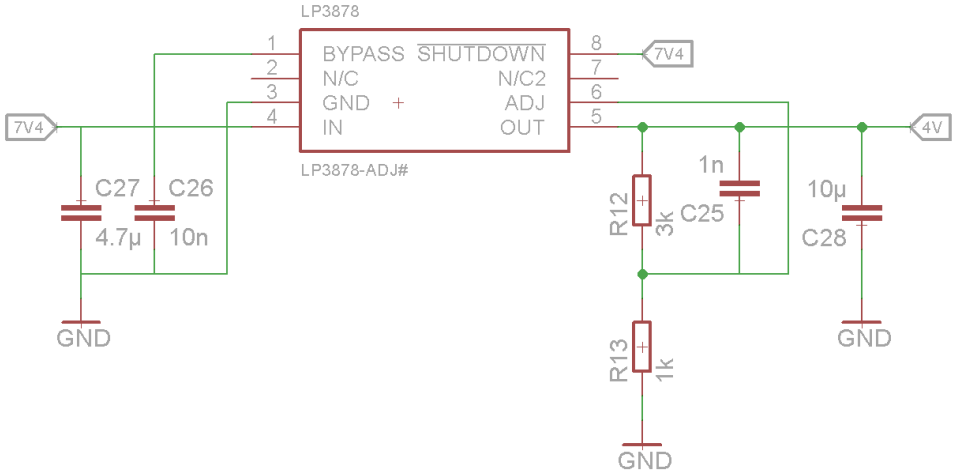


Abbildung 7: Beschaltung Linearregler

5.4.3 Bauteile & Berechnungen

Neben den Stützkondensatoren C27 und C28 wird der Bypass Kondensator C26 mit 10nF verwendet, um einem Rauschen am Ausgang entgegenzuwirken, sowie C25 mit 1nF, um den Phasenrand und somit die Stabilität des internen Regelkreises zu verbessern. Die Ausgangsspannung wird, wie in Gleichung 5.5 beschrieben, über den Spannungsteiler bestehend aus R12 und R13 berechnet, wobei R13 mit einem fixen Wert von 1kΩ bemessen ist.

$$R_{13} = 1k\Omega \qquad V_{ADJ} = 1V \tag{5.4}$$

$$V_{OUT} = 4V\Omega \qquad R_{13} = R_{12} \cdot \left(\frac{V_{OUT}}{V_{ADJ}} - 1\right) = 3k\Omega \tag{5.5}$$

5.5 Bluetooth-Modul

5.5.1 Charakteristik

Um mit dem Luftschiff auf kurze Distanzen zu kommunizieren und in den Anläufen die Firmware zu debuggen, wird ein Bluetooth-Modul in die Schaltung eingebaut. Das verwendete Modul HC-06 hat einen Versorgungsspannungsbereich von 3.1V - 4.2V und einen Stromverbrauch von 30mA - 40mA während dem Kopplungsvorgang mit einem Bluetooth kompatiblem Gerät. Der Stromverbrauch während des Kommunikationsvorgangs wird vom Hersteller auf 8mA geschätzt. Das eingebaute USART Interface wird verwendet, um den Datenaustausch mit dem µC herzustellen. Wie in Abbildung 8 zu erkennen ist hat das Modul eine integrierte Antenne für die 2.4GHz Bluetooth Übertragung.

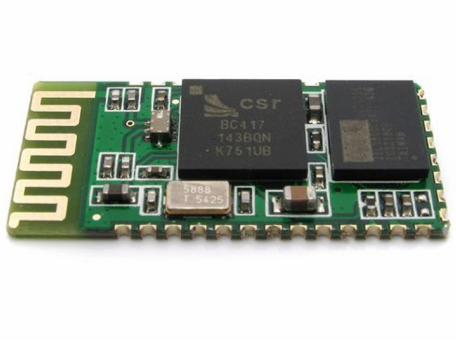


Abbildung 8: HC-06 Bluetooth-Modul

5.5.2 Beschaltung

Abbildung 9 zeigt, wie das Bluetooth-Modul in die Schaltung implementiert wird. Da die Pinbelegung des HC-06 und die des Vorgängers HC-05 identisch sind, wird eine Eagle-Library des HC-05 verwendet.

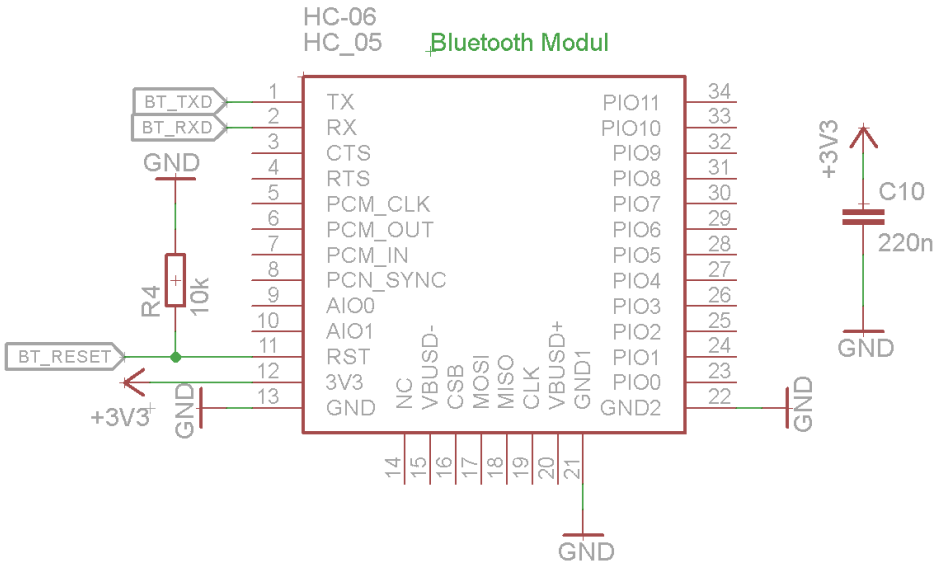


Abbildung 9: Beschaltung HC-06

Da es sich beim HC-06 um ein Modul handelt, müssen extern kaum Bauteile angebracht werden. So ist lediglich C10 als Stützkondensator zwischen V_{CC} und GND, sowie ein 10kΩ Widerstand am Reset Pin auf GND geschaltet. Die RX und TX Pins für die USART Schnittstelle und der Reset Pin sind mit dem µC verbunden.

5.5.3 Konsideration im Layout

Aufgrund der hohen Frequenz von Bluetooth wird im Layout die Kupferfläche unter der Antenne des HC-06 Moduls entfernt, um Störungen zu vermeiden. (Abbildung 10)

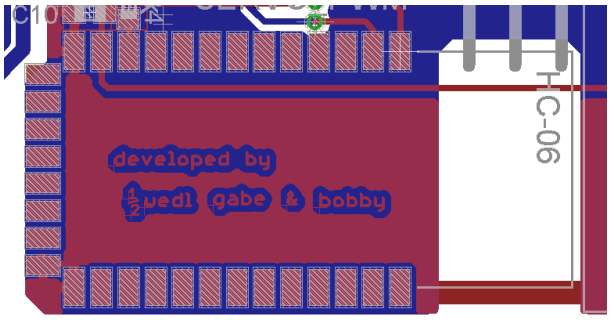


Abbildung 10: HC-06 im Layout

5.5.4 Leistung und Reichweite

Bluetooth Geräte sind wie Tabelle 3 zeigt in Leistungsklassen eingeteilt, wodurch sich auch die Reichweite ableiten lässt. Das HC-06 befindet sich in der Klasse 2.

Klasse	**Max. Leistung**	**Max. Leistung**	**Reichweite allgemein**	**Reichweite im Freien**
Klasse 1	100 mW	20 dBm	ca. 100 m	ca. 100 m
Klasse 2	2.5 mW	4 dBm	ca. 10 m	ca. 50 m
Klasse 3	1 mW	0 dBm	ca. 1 m	ca. 10 m

Tabelle 3: Bluetooth Leistungsklassen[2]

5.5.5 Frequenzbereich & Übertragungsrate

Bei Bluetooth handelt es sich um einen Industriestandard für die Datenübertragung zwischen Geräten. Dabei sind jedoch nur kurze Strecken möglich. Der Frequenzbereich, auf welchem Bluetooth überträgt, liegt zwischen 2,402GHz und 2,480GHz. Es wäre theoretisch möglich, eine Übertragungsrate von 706,25kBit/s zu erreichen. Das HC-06 jedoch ermöglicht eine maximale Baud-Rate von 1382400, was im Falle dieses Moduls einer Bitrate von 172,8kBit/s entspricht. Für die Anwendung des Debuggens ist eine Übertragungsrate von 57600 Baud jedoch völlig ausreichend und wird daher verwendet.

5.6 Gyro-, Temperatur- & Beschleunigungssensor

5.6.1 Charakteristik

Um die Geschwindikeit und die Rotation des Luftschiffes zu ermitteln, werden ein Gyro- sowie ein Beschleunigungssensor in Form eines MPU6050 IC von InvenSense eingebaut. Dieser Chip verfügt zusätzlich über einen Temperatursensor, der aber für die Anwendung nicht notwendig ist. Der MPU6050 kann in einem Versorgungsspannunsbereich von 2.375V - 3.46V verwendet werden und wird somit über den Buck Converter versorgt. Falls alle Sensoren im Betrieb sind, hat der IC einen typischen Stromverbrauch von 3.8mA.

Der MPU6050 verfügt über ein IIC-Interface, über welches er vom µC angesprochen wird, sowie ein Auxiliary IIC-Interface, welches die Möglichkeit bietet, externe Sensoren an den IC anzuschliesen und periodisch Daten einzulesen. Diese Funktion wird in Kombination mit einem Kompass IC verwendet. (siehe 5.7)

5.6.1.1 Empfindlichkeit der Sensoren

Die Empfindlichkeit der Sensoren ist in Tabelle 4 abgebildet. Die Bereiche für den Gyro- bzw. Beschleunigungssensor sind abhängig von den verwendeten Einstellungen des MPU.

Sensor	**Empfindlichkeit**	**Einheit**
Temperatur	-40 - +85	°C
Gyro	±16	g
Beschleunigung	±2000	°/s

Tabelle 4: Empfindlichkeiten MPU

5.6.2 Beschaltung

Für das SCAV wird der MPU6050 wie in Abbildung 11 beschalten.

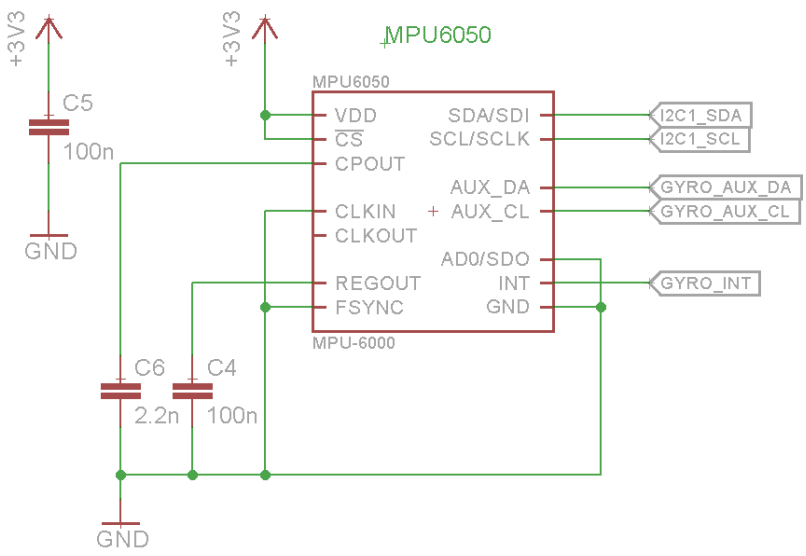


Abbildung 11: Beschaltung MPU6050

Neben dem Stützkondensator C5 benötigt der MPU6050 einen Kondensator mit 2.2nF für die integrierte Ladungspumpe, welche eine höhere Spannung für die internen MEMS[1] Oszillatoren erzeugt. C4 wird für die Ausgangsspannung des LDO[2] im Chip benötigt.

Das IIC-Interface mit SDA und SCL sowie der INT Pin ist mit dem µC verbunden. Über den INT Pin kann ein periodischer Interrupt nach Erfassen der Sensordaten ausgeben werden, wobei das Zeitintervall, in welchem diese Daten gemessen werden, eingestellt werden kann. Über das Auxiliary Interface wird ein Kompass angeschlossen und dessen gemessene Daten ebenfalls periodisch vom MPU6050 ausgelesen.

5.6.3 Orientierung

Abbildung 12 zeigt die Ausrichtung der Achsen bzw. Rotation um die Achsen. Um Fehler zu vermeiden, muss darauf geachtet werden, dass der MPU6050 in der richtigen Ausrichtung auf der Leiterplatte platziert wird und diese anschließend ohne Verschiebungen und normal zum Erdboden an der Unterseite der Hülle befestigt ist. Abweichungen durch Ungenauigkeit können im Notfall jedoch in der Software herausgerechnet werden.

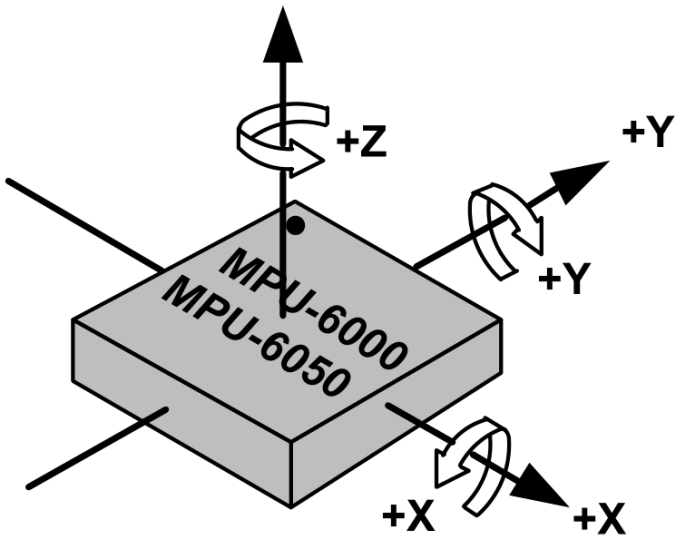


Abbildung 12: Orientierung der Achsen

[1] Microelectromechanical system oscillator
[2] Low-dropout regulator

5.7 Kompass

5.7.1 Charakteristik

Zur Bestimmung der genauen Ausrichtung des Prallluftschiffes wird der Kompass IC HMC5883L verwendet. Mit einer Versorgungsspannung im Bereich von 2.16V - 3.6V wird auch dieser Baustein vom Buck Converter versorgt. Besonders hevorzuheben ist der niedrige Stromverbrauch von 100µA. Der HMC5883L misst magentische Felder im Bereich von ±8 gauss und hat eine maximale Messfrequenz von 75 Hz. Der Sensor verfügt über ein IIC-Interface, welches mit dem MPU6050 verbunden ist.

5.7.2 Beschaltung

Der Kompass IC HMC5883L wird folgendermaßen in die Schaltung implementiert. (Abbildung 13)

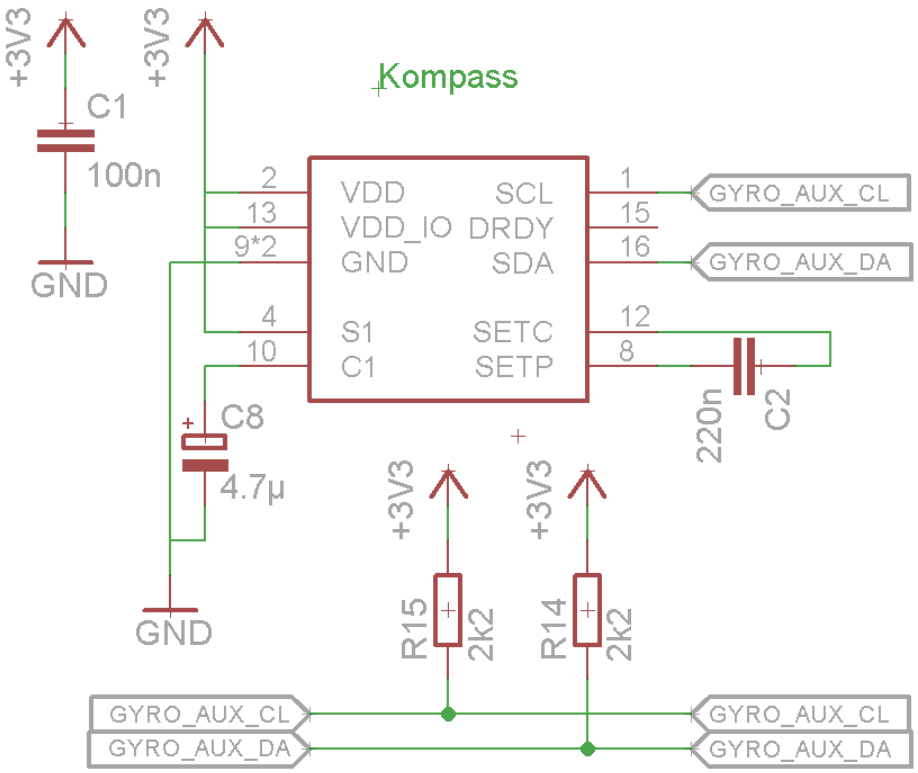


Abbildung 13: Beschaltung HMC5883L

Unter den externen Bauteilen befinden sich ein Stützkondensator (C1), ein Speicher-Elektrolytkondensator (C8) und ein Set/Reset Kondensator (C2) mit 220nF zwischen dem SETC und SETP Pin. SDA und SCL sind mit dem MPU6050 verbunden und über einen Pullup mit 2.2kΩ Widerständen, welcher für die IIC Kommunikation benötigt wird, auf 3.3V gezogen.

5.8 GPS-Modul

5.8.1 Charakteristik

Die Position des Prallluftschiffes wird mithilfe eines GPS-Moduls ermittelt. Dazu wird das FGPMMOPA6C-Modul der Firma GlobalTop Technology Inc. mit den Abmessungen von 16x16x6.2 mm verwendet. Das Modul läuft mit einem MT3339 Chip von MediaTek, welches eine sehr hohe Empfindlichkeit (-165dBm) aufweist. Das Modul läuft in einem Spannungsbereich von 3V - 4.3V und wird über den Buck Converter mit 3.3V versorgt. Der Stromverbrauch während der Erfassung der Satelliten wird auf 25mA und während des Verfolgens auf 20mA geschätzt.

Ansprechbar ist das GPS-Modul über eine USART-Schnittstelle, welche mit dem µC verbunden ist. Je nach Einstellung können die aktuellen Daten periodisch und mit einer maximalen Frequenz von 10Hz übertragen werden. Zusätzlich verfügt das Modul über die Möglichkeit, eine Backup Versorgung anzuschließen, um den Erfassungsprozess nicht bei jedem Neustart zu wiederholen. Auf dies wird aufgrund der dauernden Versorgung über Akku und Buck Converter jedoch verzichtet.

5.8.2 Beschaltung

In der Schaltung wird das GPS-Modul wie in Abbildung 14 integriert.

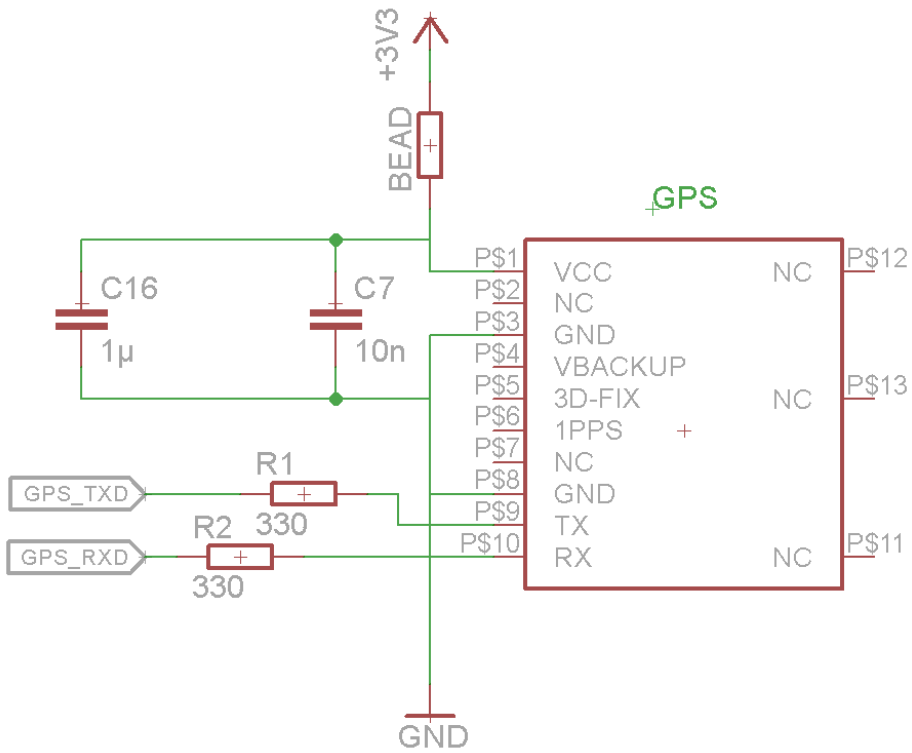


Abbildung 14: Beschaltung GPS-Modul

Das Modul wird mit zwei Stützkondensatoren, C16 mit 1µF und C7 mit 10nF sowie einer Ferritspule, welche sich wie ein Hochpass auswirkt und somit zur Rauschunterdrückung beiträgt, verbaut. Ansonsten wird lediglich die USART-Schnittstelle mit dem µC verbunden, wobei zwei

Widerstände R1 und R2 in die Leitung eingebaut werden, um bei möglicher falscher Verbindung für die Sicherheit des Moduls durch Begrenzung des Stroms zu sorgen.

5.8.3 Funktion

Die Ermittlung einer Position auf der Erde über das GPS funktioniert über die Kommunikation mit Satelliten in der Erdumlaufbahn. Dabei senden die Satelliten ihre Position und Ausrichtung im Orbit an die GPS Geräte, welche anschließend mithilfe von Trilateration bzw. Multilateration eine Position in der Ebene bestimmen. Bei der Trilateration werden durch die Information von drei Abständen zu den Satelliten die Position in der 2D Ebene bestimmt (siehe Abbildung 15). Um eine 3D Position zu bestimmen, sind vier oder mehr Satelliten notwendig. Man spricht von Multilateration. Die Zahl der verbundenen Satelliten erhöht also die Genauigkeit der Positionsbestimmung.

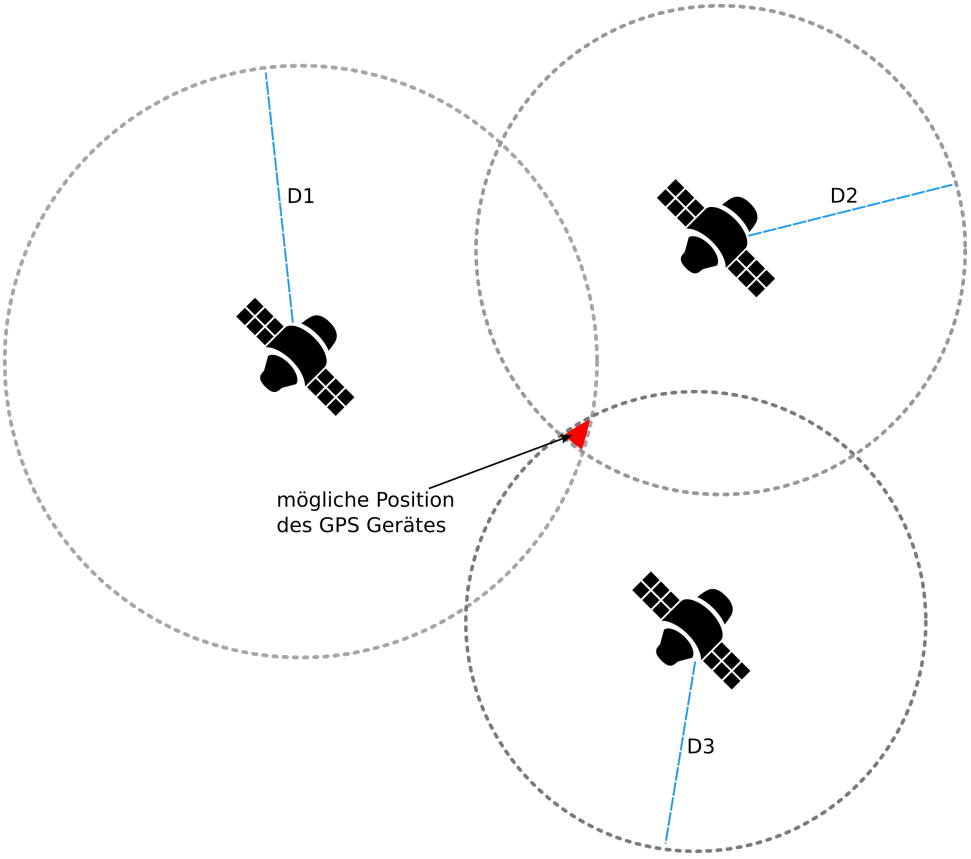


Abbildung 15: 2D Positionsbestimmung mit Trilateration

5.8.4 DGPS Modus

Das FGPMMOPA6C-Modul verfügt zusätzlich über die Möglichkeit, den DGPS Modus zu aktivieren, welcher die Genauigkeit der Positionsbestimmung drastisch erhöht. Dabei werden zusätzliche Korrekturdaten versendet, über welche durch Laufzeitdifferenzen der Satelliten zum Empfänger eine genauere Positionserkennung durchgeführt wird. Je nach Abstand zu den fixen Referenzstationen, in welchen die Differenzen zwischen theoretischer und tatsächlicher Signallaufzeiten gespeichert sind, kann die Genauigkeit auf 0.3 - 2.5m verbessert werden.

5.9 GSM-Modul

5.9.1 Charakteristik

Die Kommunikation mit dem Luftschiff über längere Distanzen erfolgt über das GSM-Netz in Vorarlberg, da dieses gut ausgebaut ist und eine unkomplizierte Methode zur Datenübertragung bietet. Das Ziel des Luftschiffes bzw. der abzufliegende Pfad wird mithilfe von SMS übermittelt. Dazu wird das GSM-Modul SIM800L der Firma SIMCom auf einem Breakout Board verwendet. Die Versorgungsspannung dieses Moduls liegt im Bereich von 3.4V - 4.4V und es kann Stromspitzen bis zu 1A sind möglich. Aufgrund dessen wird ein Linearregler (siehe 5.4) eingebaut. Befindet sich das Modul im Sleep-Modus, benötigt es lediglich 1mA und im Idle-Modus, welcher dauerhaft verwendet wird, ca. 19mA.

Für das Empfangen bzw. Versenden von SMS genügt eine serielle Verbindung des integrierten USART-Interface des SIM800L mit dem µC, sowie eine SIM Karte eines in Vorarlberg vertretenen Netzbetreibers.

5.9.2 Beschaltung

Abbildung 5 zeigt die aufgrund der Verwendung eines Breakout Boards recht einfache Beschaltung. RX & TX des USART-Interface sind mit dem µC und VCC mit dem 4V Ausgang des Linearreglers verbunden.

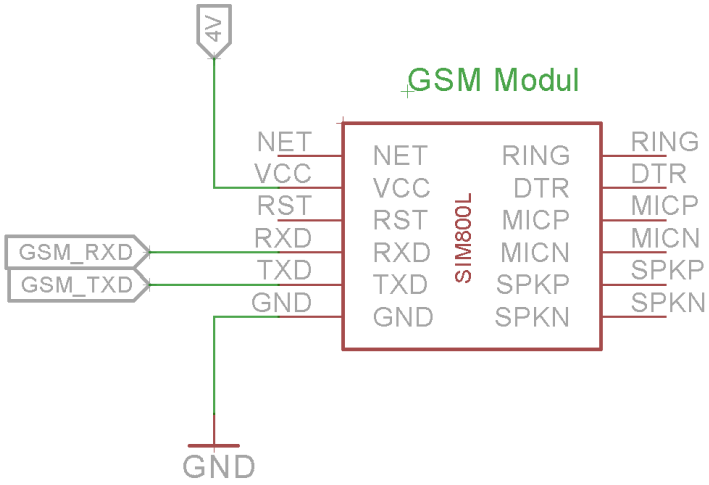


Abbildung 16: Beschaltung SIM800L Breakout Board

5.9.3 Funktion

SIM800L ist ein vierfach Band GSM-Modul und arbeitet somit in den Frequenzbändern GSM850MHz, EGSM900MHz, DCS1800MHz und PCS1900MHz, wobei das Senden bzw. Empfangen mithilfe von Frequenzmultiplexing geregelt wird und die Daten digital über Zeitmultiplexing übertragen werden. Die Daten werden nicht kontinuierlich, sondern in gepulsten Paketen mit 217Hz übertragen. Die HF-Signale werden 8-PSK[3] moduliert.

[3]Phase Shift Keying

5.10 BLDC Motortreiber

5.10.1 Charakteristik

Bei den drei Motoren, welche zur Steuerung des Luftschiffes verwendet werden, handelt es sich um BLDC Motoren (siehe 8.3.4). Die Spannung an den drei Phasen der Motoren wird jeweils über einen A4963 Sensorless BLDC Motortreiber IC von Allegro MicroSystems gesteuert. Dieser Motortreiber ermöglicht eine Regelung des Motors in mehreren Betriebsmodi über die Ansteuerung von FET Halbbrücken (siehe 5.11). Dabei hat es einen Versorgungsspannungsbereich von 4.2V - 50V und wird so wie auch die Motoren, direkt über die Spannung des Akkumulators versorgt. Im Ruhezustand hat der A4963 einen Stromverbrauch von 12mA, wohingegen dieser im Betrieb abhängig von den Einstellungen stark schwanken kann.

Die Motortreiber ICs werden mit dem µC über die integrierte SPI Schnittstelle angesteuert, wobei die Auswahl des anzusprechenden Treibers über Chip Select erfolgt. Wie Abbildung 17 zeigt verfügt der SMD Chip über ein thermal Pad, welches zur Wärmeabfuhr benötigt wird.

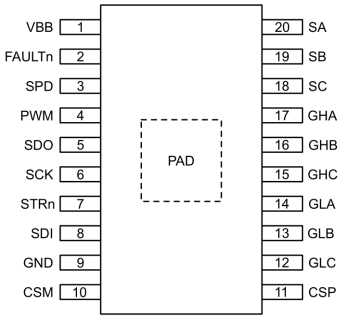


Abbildung 17: Pinout des A4963 IC[5]

5.10.2 Beschaltung

In Abbildung 18 wird die Beschaltung eines der drei Motortreiber exemplarisch dargestellt.

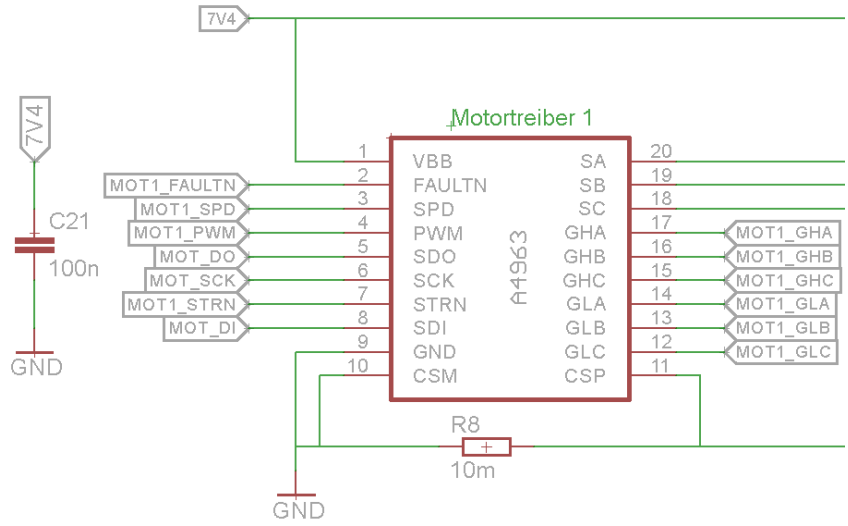


Abbildung 18: Beschaltung A4963 Motortreiber

Der IC benötigt abgesehen von den FETs nur wenige externe Bauteile. Neben dem Stützkondensator C21 wird lediglich der Sense Widerstand R8 benötigt. Dieser Widerstand spielt für die Messung und Begrenzung des Stroms durch die Motoren eine wichtige Rolle. Alle drei Source Anschlüsse der FETs und somit der gesamte Strom durch den Motor, fließt über R8, weswegen ein 10mΩ Widerstand verwendet wird. Gleichung 5.6 zeigt, wie sich die Strombegrenzung aus dem Sense Widerstand R8 und dem über die serielle Schnittstelle eingestellten Register VIL[3:0] berechnen lässt.

$$\begin{aligned} V_{ILIM} &= (n+1) * 12.5mV \qquad n...VIL[3:0] \\ I_{LIM} &= \frac{V_{ILIM}}{R_{SENSE}} \qquad R_{SENSE} = 20m\Omega \end{aligned} \tag{5.6}$$

5.10.3 Konsideration im Layout

Betrachtet man Abbildung 17, kann festgestellt werden, dass sich die I/O zur Regelung, Fehlererkennung und Kommunikation des A4963 auf der linken Seite des ICs und die Pins zur Steuerung der FETs auf der rechten Seite befinden. Im Layout wird wie in Abbildung 19 die Masse- & V_{BB}-Verbindung des Chips und die Masse- & V_{BB} Verbindung des Motors, sprich die Verbindung mit dem CSM Pin (siehe Abbildung 18), getrennt ausgeführt, um Spannungseinbrüche zu vermeiden.

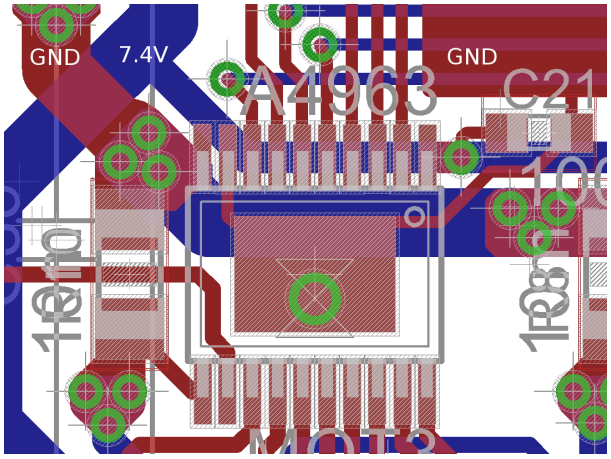


Abbildung 19: A4963 im Layout

Da kurzzeitig Ströme bis zu 10A fließen können, werden die Durchkontaktierungen wie in Abbildung 19 mehrfach platziert, um den Widerstand, welcher durch das Durchkontaktieren entsteht, zu minimieren. Zusätzlich wird bei den Thermal Pads eine Durchkontaktierung angebracht, um das Einlöten des A4963 von Hand zu ermöglichen.

5.10.4 Betriebsmodi

Der A4963 IC verfügt über vier Betriebsmodi, welche über die serielle Schnittstelle eingestellt werden können. Dabei unterscheidet man in:

- Indirect Speed (Duty Cycle)
- Direct Speed (Duty Cycle)
- Closed-loop Current
- Closed-loop Speed

Für die Steuerung der BLDCs wird der Modus *Closed-loop Speed* verwendet. Bei diesem Modus wird die Regelung der Geschwindigkeit des Motors auf den Sollwert während des Startvorgangs und dem Betrieb vom IC übernommen. Mithilfe eines LF PWM Signals im Bereich von 5Hz - 1kHz wird dem Motortreiber dieser Sollwert über Einstellen des Duty Cycles übermittelt. Die aktuelle Geschwindigkeit des angeschlossenen Motors wird über den SPD Ausgang des A4963 als Spannungssignal, dessen Periodendauer einem elektrischen Zyklus entspricht, ausgegeben (siehe 5.10.5)

5.10.5 Funktion

Um einen BLDC Motor zu betreiben, müssen drei in der Phase um 120° Rechtecksignale an die Spulen des Motors angelegt werden. Diese werden über den Motortreiber, welcher wiederrum die FET Halbbrücken steuert, erzeugt und sind in Abbildung 20 dargestellt.

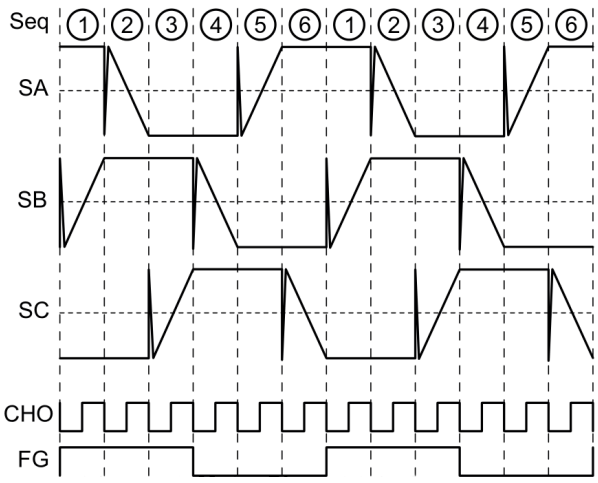


Abbildung 20: Phasen am BLDC Motor[5]

Die Frequenz f_s von FG entspricht hierbei einem elektrischen Zyklus und ermöglicht, wie in Gleichung 5.7 gezeigt, die Berechnung der tatsächlichen Geschwindigkeit ω in rpm des Motors, wenn die Anzahl der im Motor vorhandenen Polpaare N_{PP} bekannt ist.

$$\omega = \frac{60 * f_s}{N_{PP}} \tag{5.7}$$

5.11 Halbbrücken

5.11.1 Charakteristik

Der BLDC Motortreiber A4963 (siehe 5.10) liefert die Steuersignale für drei FET-Halbbrücken, an welchen der BLDC Motor angeschlossen werden kann. Für die Elektronik des Luftschiffes werden die MOSFET Halbbrücken IC DMC3016LSD von DIODES Incorporated verwendet. Diese beinhalten einen N- sowie P-Kanal MOSFET in einem SO-8 Gehäuse wie in Abbildung 21 dargestellt. Die Spezifikationenen der MOSFETs sind in Tabelle 5 angegeben.

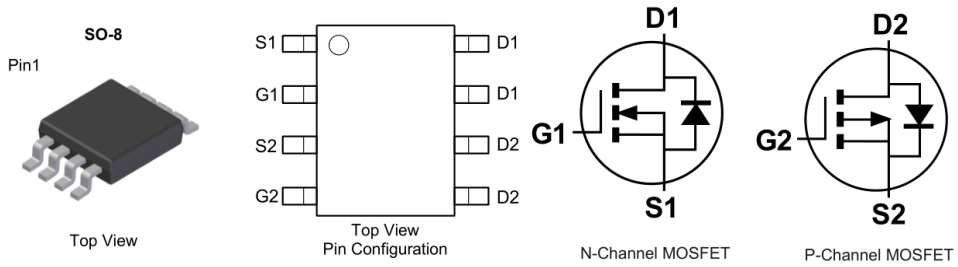


Abbildung 21: Pinout des A4963 IC[8]

Device	$V_{(BR)DSS}$	$R_{DS(ON)}Max$	I_D
Q1	$30V$	$16m\Omega$ @ V_{GS} = $10V$ $20m\Omega$ @ V_{GS} = 4.5	$8.2A$ $7.3A$
Q2	$-30V$	$28m\Omega$ @ V_{GS} = $-10V$ $38m\Omega$ @ V_{GS} = $-4.5V$	$-6.2A$ $-5.2A$

Tabelle 5: Spezifikationen DMC3016LSD

5.11.2 Beschaltung

Abbildung 22 zeigt die Beschaltung der drei MOSFET Halbbrücken über den Motortreiber A4963 (siehe 5.10.2) und die Anschlüsse SA, SB und SC des BLDC Motors.

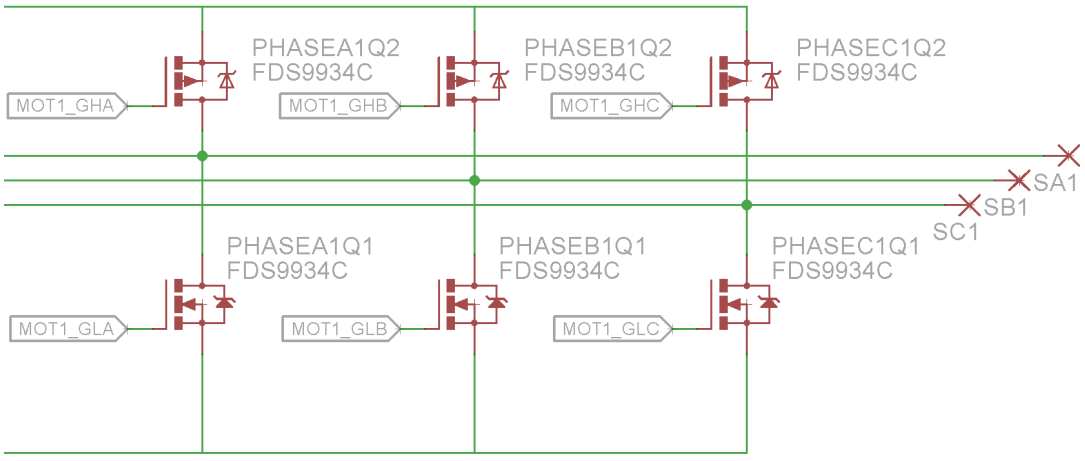


Abbildung 22: Beschaltung Halbbrücke

5.12 Mikrocontroller

5.12.1 Charakteristik

Die zentrale Steuereinheit der Elektronik des Luftschiffes ist der Mikrocontroller. Dafür wird der GD32F103CB, eine 32 bit MCU von Giga Device, welche über 128KB Flash, 20KB SRAM, 7 Timer und 9 Kommunikationsschnittstellen verfügt, verwendet. Es handelt sich dabei um einen nahezu identischen Nachbau des STM32F103CB, wobei der GD32 im Gegensatz zum STM32 mit 64MHz eine maximale Oszillatorfrequenz von 108MHz erreicht.

Versorgt wird er Mikrocontroller in seinem Spannungsbereich von 2V - 3.6V über den Buck Converter mit 3.3V.

5.12.2 ST-Link

Der Mikrocontroller wird mit dem ST-Link Programmer (siehe Abbildung 24) beschrieben. Dieser übernimmt die Funktion eines Debuggers. Der ST-Link verfügt über 2x3.3V und 2x5V sowie 2xGND Ausgänge einem SWIM[4], einem SWD-Interface[5] und einem Reset Ausgang. Der GD32 wird dabei über das SWD Interface programmiert und gedebuggt.

Es zeigt sich, dass der GD32 ein exakter Nachbau des STM32 ist, da dieser auch über den ST-Link programmierbar ist.

Abbildung 23: USB ST-Link Programmer

5.12.3 Evaluation Board

Um die Funktion der Komponenten sowie ersten Code zu testen, wird ein Evaluation Board des GD32F103C8, welcher dem schlussendlich verwendeten GD32F103CB sehr ähnlich ist, verwendet. In Abbildung 24 ist links die Schnittstelle für das SWD Interface zu sehen, über welches der ST-Link angeschlossen werden kann. Zusätzlich ist es auch möglich, das Evaluation Board über USB zu versorgen, zu programmieren und zu debuggen.

Neben den für die Funktion des µC essentiellen Pins, wie die Anschlüsse für die externen Oszillatoren oder die beiden Boot Pins (Boot0, Boot1), sind alle Pins herausgeführt und können verwendet werden. Aufgrund der Aufteilung der Arbeiten in der DA wird anfänglich mit zwei dieser Testboards gearbeitet.

[4]Serial Wire Interface Module
[5]Serial Wire Debug Interface

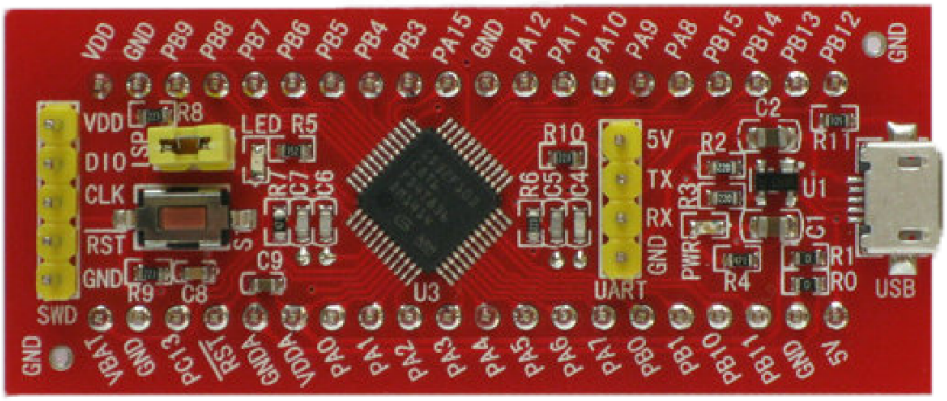


Abbildung 24: GD32 Evaluation Board

5.12.4 Beschaltung

In Abbildung 25 ist die Beschaltung des GD32 dargestellt. Die dabei verwendete Eagle Library ist für den STM32F103C8. Aufgrund der gleichen Pinbelegung und Gehäuseform spielt dies jedoch keine Rolle. Um den Überblick in der Schaltung zu bewahren, werden die meisten Signale mit Labels versehen.

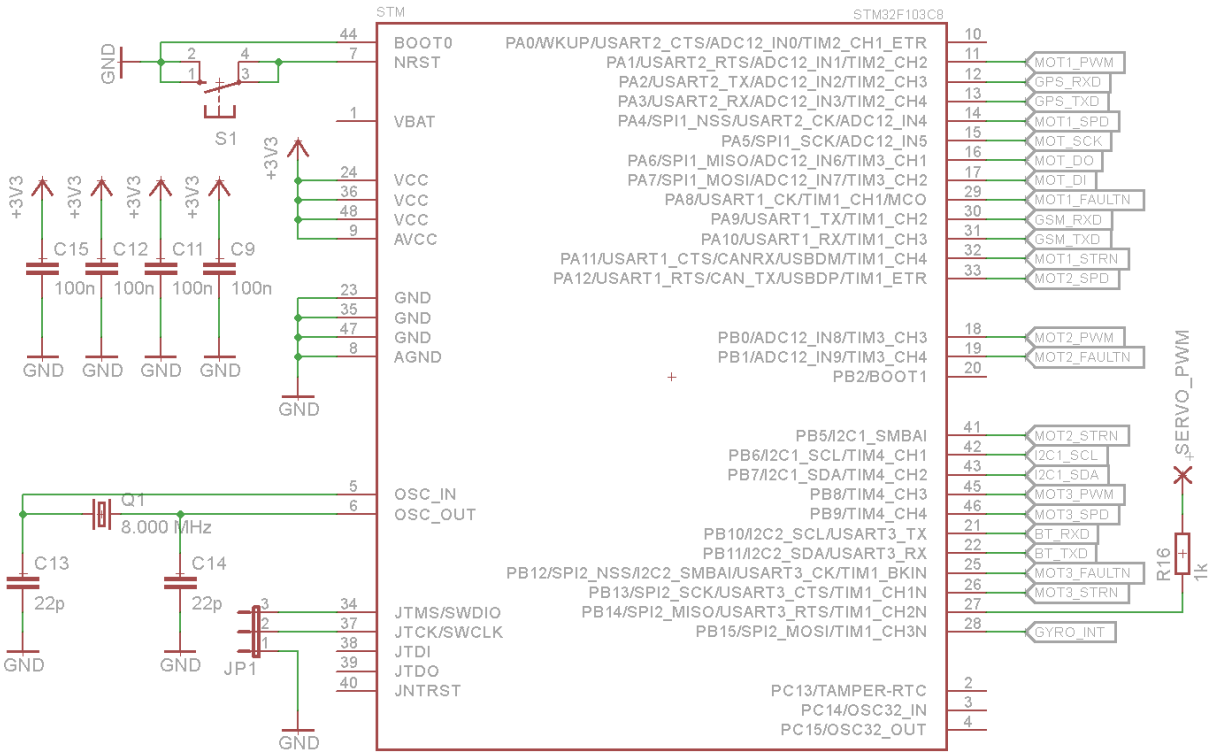


Abbildung 25: Beschaltung A4963 Motortreiber

C9, C11, C12 und C15 zu je 100nF sind Stützkondensatoren zwischen den vier GND und VCC (bzw. GND und AVCC) Anschlüssen des Mikrocontrollers. Diese werden möglichst nah am µC platziert. Bei S1 handelt es sich um einen Taster, welcher durch Drücken den NRST Pin auf LOW zieht und somit einen Reset im µC auslöst. Der Boot0 Pin wird auf Ground verbunden und somit wie in Tabelle 6 dargestellt, vom Flash gebootet.

BOOT1	BOOT0	Boot Mode	Aliasing
x	0	Main Flash memory	Main Flash memory is selected as boot space
0	1	Main System memory	System memory is selected as boot space
1	1	Embedded SRAM	Embedded SRAM is selected as boot space

Tabelle 6: Boot modes[9]

An OSC_IN und OSC_OUT ist ein 8.000MHz Quarz Oszillator mit den beiden 22pF Kondensatoren C13 & C14 angeschlossen. Dieser dient als externe und sehr genaue Clock Quelle, was jedoch nicht unbedingt benötigt wird, bei hohen Frequenzen und Benutztung von USART jedoch empfohlen ist. JP1 ist mit der JTAG Schnittstelle des µC verbunden und erlaubt das Programmieren und Debuggen über den ST-Link.

In Tabelle 7 ist die restliche für die Elektronik bedeutende Pinbelegung des GD32F103CB, mit Bezug auf die verbundenen Komponenten dargestellt.

Komponente	PIN	Funktion
Servo Motor	PB14	PWM Output für die Servo Steuerung
MPU6050	PB6	Clock Leitung für den IIC Datentransfer
	PB7	Daten Leitung für den IIC Datentransfer
	PB15	Data Ready Interrupt Eingang des MPU6050
HC-06	PB10	Sendeleitung für den USART Datentransfer zum BT-Modul
	PB11	Empfangsleitung für den USART Datentransfer vom BT-Modul
	PC13	Reset des BT-Moduls
SIM800L	PA9	Sendeleitung für den USART Datentransfer zum GSM-Modul
	PA10	Empfangsleitung für den USART Datentransfer zum GSM-Modul
A4963	PA6	Clock Leitung für den SPI Datentransfer zu allen Motortreibern
	PA7	Empfangsleitung für den SPI Datentransfer zu allen Motortreibern
	PA7	Sendeleitung für den SPI Datentransfer von allen Motortreibern
	PA1	PWM Steuerleitung für die Motortreiber
	PB0	
	PB8	
	PA5	Übertragungsleitung der Speed-Frequenz der Motortreiber
	PA12	
	PB9	
	PA8	Fehlerübertragung von den Motortreibern
	PB1	
	PB12	
	PA1	Chip Select für den SPI Datentransfer zu den Motortreibern
	PB5	
	PB13	

Tabelle 7: Pinbelegung am µC

Der Kompass IC, welcher in Tabelle 7 nicht aufgelistet ist, wird nicht direkt mit dem µC verbunden, sondern über eine elektronische Durchschaltung des IIC Buses am MPU6050 auf dessen Auxiliary Interface angesteuert. (siehe 5.7)

5.13 Layout

5.13.1 Top

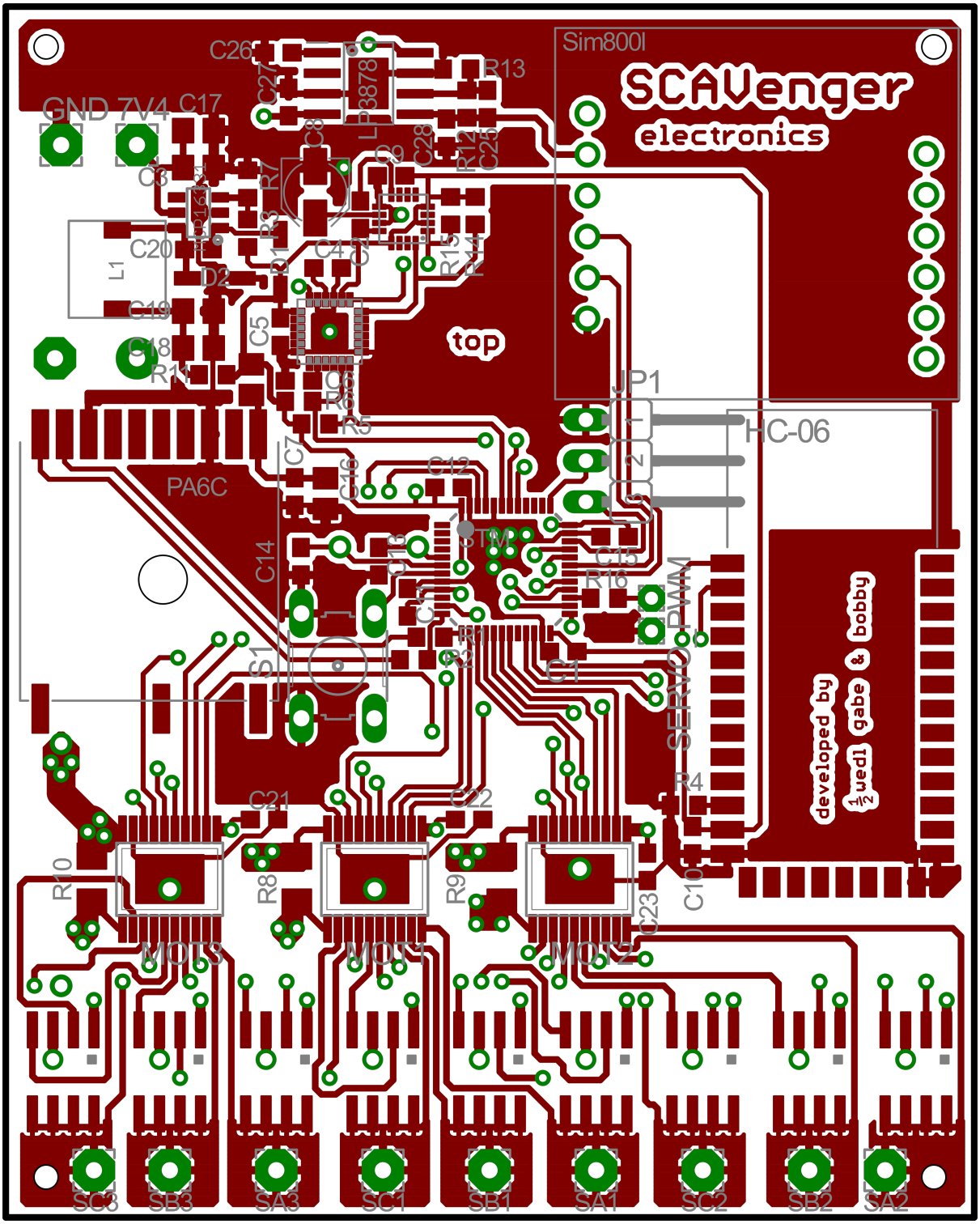


Abbildung 26: Layout: top layer

5.13.2 Bottom

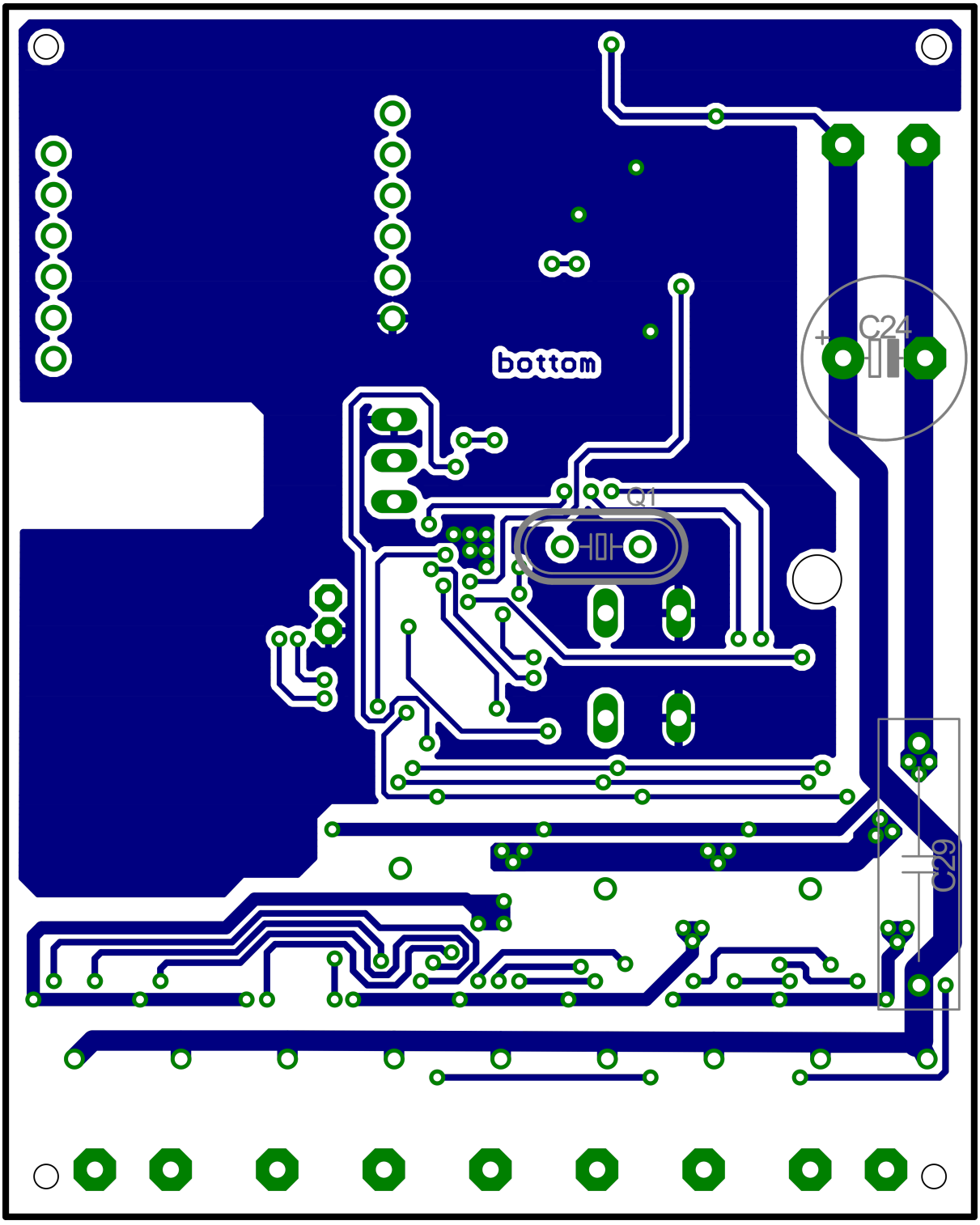


Abbildung 27: Layout: bot layer

5.13.3 3D-View

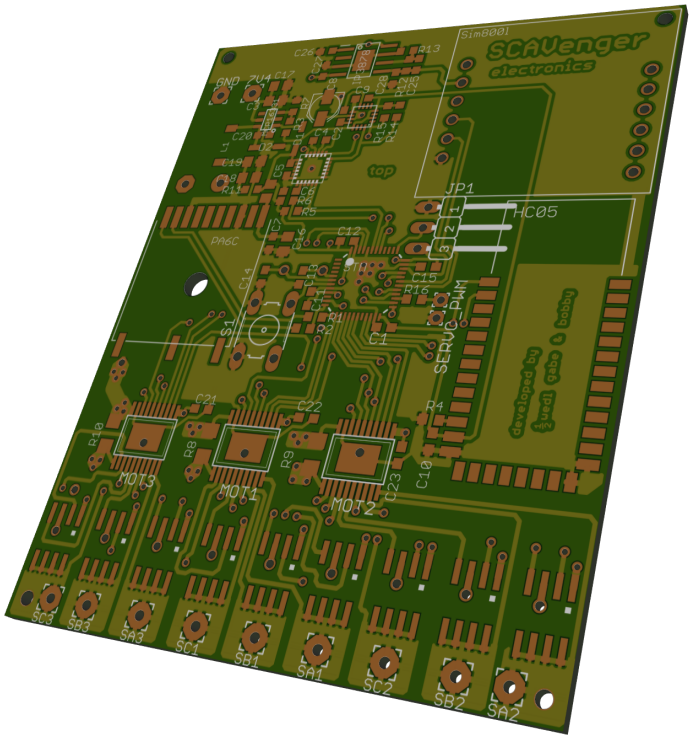


Abbildung 28: Layout 3D: top layer

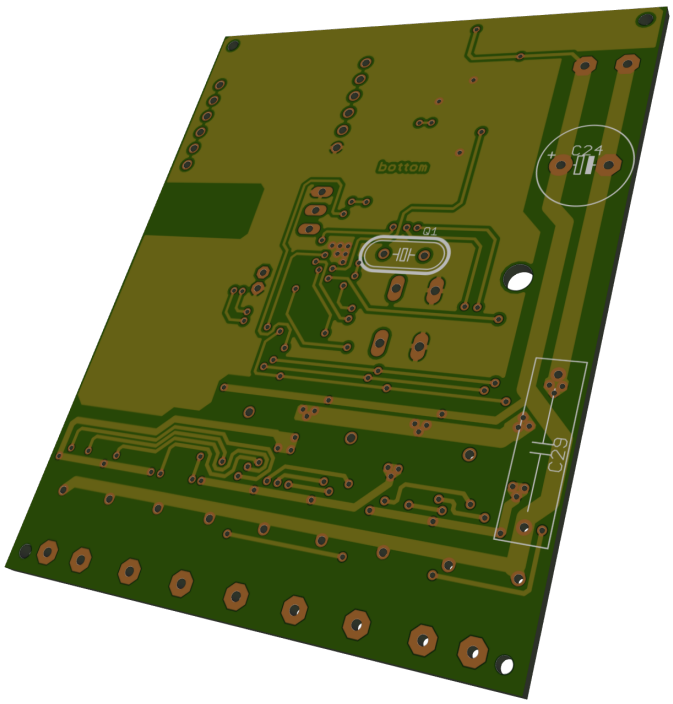


Abbildung 29: Layout 3D: bot layer

6 Firmware

6.1 Allgemein

Die Firmware ist das auf dem Mikrocontroller (siehe 5.12) laufende Programm und umfasst mehrere Bereiche. Generell wird die Firmware für die Steuerung, Regelung und Kommunikation mit der restlichen Elektronik benötigt. Programmiert ist sie in der Sprache C, wobei das für ARM[6] speziallisierte Tool Atollic TrueSTUDIO (siehe 11) verwendet wird.

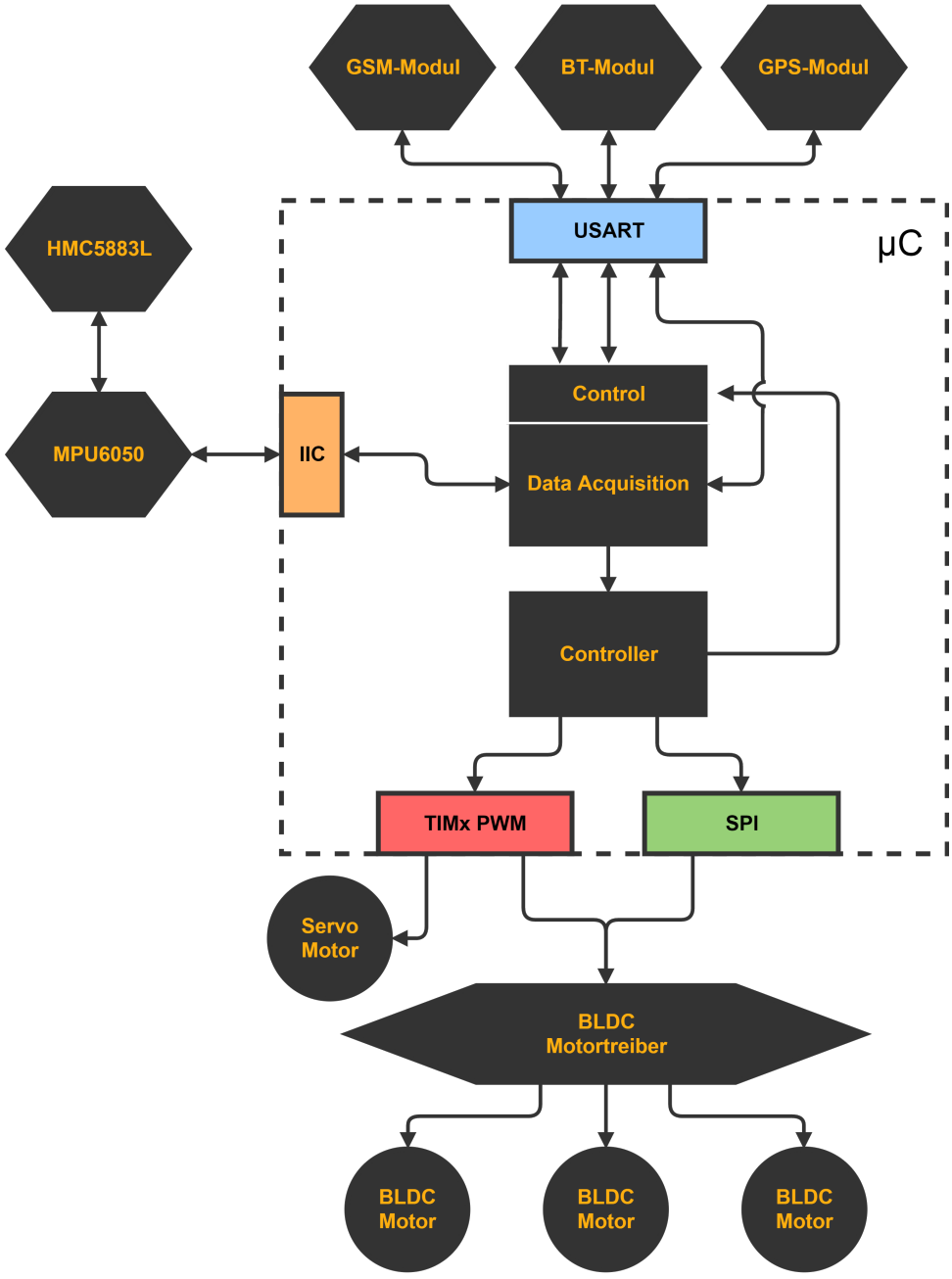


Abbildung 30: Firmware Prinzip

[6] Advanced reduced instruction set computing machine

In Abbildung 30 ist das funktionelle Prinzip der Firmware vereinfacht in einem Diagramm dargestellt. Dabei sind die einzelnen Komponenten und Module, mit den entsprechenden Interfaces über welche sie angeschlossen sind, zu sehen. Zusätzlich ist der prinzipielle Ablauf des internen Programms zu erkennen. Dieses wird in folgende drei Schritte unterteilt und in einer Schleife durchgehend abgelaufen:

1. Program Control
    - Versenden bzw. Empfangen von Debug-Daten
    - Verarbeitung der empfangenen GSM-Kommandos
    - Starten / Stoppen des Luftschiffes (Schritt 2. & 3. auslassen)
2. Data Acquisition
    - Auslesen der Sensor- und GPS-Daten
    - Umrechung der Rohdaten
3. Controller
    - Regelung des Luftschiffes
    - Verarbeitung der in Schritt 2. gespeicherten Daten
    - Ausgabe der Steuerungsdaten

Der TIMx Block stellt die Erzeugung der PWM Signale, welche für die Steuerung der vier Motoren benötigt werden, über verschiedene Timer des µC dar. Die Firmware ist modular aufgebaut und wird somit in verschiedene Komponenten getrennt. Im Hauptprogramm werden diese Komponenten geladen und der oben beschriebene Ablauf durch Aufruf einiger weniger bestimmter Routinen in einer Dauerschleife vollzogen.

6.2 Clock Einstellung

6.2.1 Allgemeines

Unmittelbar nach Start des Programms müssen die Clock und PLL Einstellungen des µC vorgenommen werden, um eine höhere Taktfrequenz als die standardmäßigen 8MHz zu erreichen. Dazu wird das clock.h sowie clock.c Header- und Source File eingebunden sowie die entsprechende Initialisierungsfunktion aufgerufen.

6.2.2 C-Code

Codeabschnitt 1: Clock Configuration Source-File

```
1  #include "stm32f10x.h"
2  #include "clock.h"
3
4  void Init_Clock()
5  {
6    ErrorStatus HSE_Error_Status;
7    RCC_DeInit();
8
9    RCC_HSEConfig(RCC_HSE_ON);
10   HSE_Error_Status = RCC_WaitForHSEStartUp();
11
12   if (HSE_Error_Status == SUCCESS)
13   {
14     RCC_PLLConfig(RCC_PLLSource_HSE_Div2, RCC_PLLMul_16);
15   }
16   else
17   {
18     RCC_PLLConfig(RCC_PLLSource_HSI_Div2, RCC_PLLMul_16);
19   }
20
21   FLASH_SetLatency(FLASH_Latency_2);
22   RCC_PLLCmd(ENABLE);
23   while(RCC_GetFlagStatus(RCC_FLAG_PLLRDY) == RESET) {}
24   RCC_SYSCLKConfig(RCC_SYSCLKSource_PLLCLK);
25
26   /* Set HCLK, PCLK1, and PCLK2 to SCLK (these are default */
27   RCC_HCLKConfig(RCC_SYSCLK_Div1);
28   RCC_PCLK1Config(RCC_HCLK_Div2);
29   RCC_PCLK2Config(RCC_HCLK_Div1);
30
31   //RCC_ADCCLKConfig(RCC_PCLK2_Div4);
32 }
```

Codeabschnitt 1 zeigt wie, die Clock Konfiguration vorgenommen wird. Dabei wird als Erstes versucht, die externe Clock Quelle zu verwenden. Je nachdem, ob dieser Versuch gelingt oder fehlschlägt, werden die PLL Teiler und Multiplikatoren unterschiedlich gesetzt. Unabhängig von der Clock Quelle wird die Flash Latency gesetzt und die PLL eingeschaltet. Sobald die PLL bereit ist, wird sie als System Clock Quelle verwendet und entsprechende Vorteiler für bestimmte Peripherien gesetzt.

6.3 Servo PWM

6.3.1 Allgemeines

Für die Steuerung des Servomotors wird ein PWM Signal von 20ms benötigt, wobei die Stellung des Motors in einem Bereich von 180°über den Duty Cycle der PWM eingestellt wird. Wie in Abbildung 31 dargestellt können die 0°- 180°über einen Duty Cycle von ca. 5% - 10% bestimmt werden.

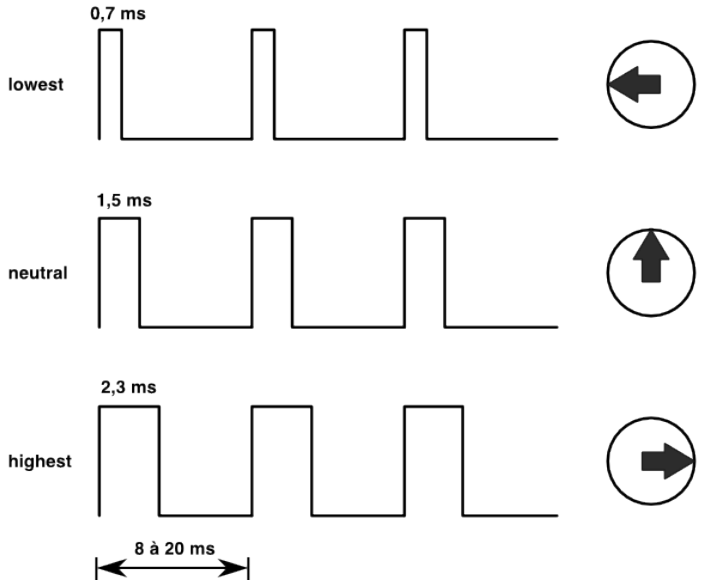


Abbildung 31: Servo: PWM Duty Cycle

6.3.2 C-Code

6.3.2.1 Servo PWM Header-File

Codeabschnitt 2: Servo PWM Header-File

```
1 #ifndef SERVOPWM_H_
2 #define SERVOPWM_H_
3
4 void Init_Servo();
5 void Servo_SetRotation(uint8_t angle);
6
7 #endif /* SERVOPWM_H_ */
```

Für das Modul, welches die PWM zur Servo Motor Steuerung übernimmt, werden lediglich zwei Funktionen benötigt. Wie Codeabschnitt 2 veranschaulicht, handelt es sich dabei um eine Funktion zur Initialisierung des Moduls und eine Weitere, mit welcher die Stellung des Motors gesetzt werden kann.

6.3.2.2 Servo PWM Source-File

Codeabschnitt 3: Servo PWM Source-File

```
1  #include "stm32f10x_tim.h"
2  #include "stm32f10x_gpio.h"
3  #include "servopwm.h"
4
5  uint16_t ServoPWM_Period = 0;
6  uint16_t ServoPWM_Pulse = 0;
7
8  // lower and upper time limits of the PWM signal for the servo angle in  s  (0 ↙
       ↳ - 180 degree)
9  uint16_t ServoPWM_UpperAngle = 2200;
10 uint16_t ServoPWM_LowerAngle = 600;
11 uint16_t ServoPWM_RotationRange = 1000;
12
13 void Init_Servo()
14 {
15   ServoPWM_RotationRange = ServoPWM_UpperAngle - ServoPWM_LowerAngle;
16   RCC_APB2PeriphClockCmd(RCC_APB2Periph_TIM1 | RCC_APB2Periph_GPIOB | ↙
         ↳ RCC_APB2Periph_AFIO, ENABLE);
17
18   GPIO_InitTypeDef gpioObj;
19   TIM_TimeBaseInitTypeDef TIM_TimeBase_InitStructure;
20   TIM_OCInitTypeDef TIM_OC_InitStructure;
21
22   gpioObj.GPIO_Mode = GPIO_Mode_AF_PP;
23   gpioObj.GPIO_Speed = GPIO_Speed_50MHz;
24   gpioObj.GPIO_Pin = GPIO_Pin_14;
25   GPIO_Init(GPIOB, &gpioObj);
26
27   // f = 50 Hz
28   RCC_ClocksTypeDef clk;
29   RCC_GetClocksFreq(&clk);
30   ServoPWM_Period = clk.HCLK_Frequency / 10000;
31   ServoPWM_Pulse = ServoPWM_Period / 10;
32
33
34   TIM_TimeBase_InitStructure.TIM_ClockDivision = TIM_CKD_DIV1;
35   TIM_TimeBase_InitStructure.TIM_CounterMode = TIM_CounterMode_Up;
36   TIM_TimeBase_InitStructure.TIM_Period = ServoPWM_Period - 1;
37   TIM_TimeBase_InitStructure.TIM_Prescaler = 199;
38   TIM_TimeBaseInit(TIM1, &TIM_TimeBase_InitStructure);
39
40   TIM_OC_InitStructure.TIM_OCMode = TIM_OCMode_PWM1;
41   TIM_OC_InitStructure.TIM_OCIdleState = TIM_OCIdleState_Reset;
42   TIM_OC_InitStructure.TIM_OCNIdleState = TIM_OCNIdleState_Reset;
43   TIM_OC_InitStructure.TIM_OCPolarity = TIM_OCPolarity_Low;
44   TIM_OC_InitStructure.TIM_OCNPolarity = TIM_OCNPolarity_High;
45   TIM_OC_InitStructure.TIM_OutputState = TIM_OutputState_Disable;
46   TIM_OC_InitStructure.TIM_OutputNState = TIM_OutputNState_Enable;
47   TIM_OC_InitStructure.TIM_Pulse = ServoPWM_Pulse;
48   TIM_OC2Init(TIM1, &TIM_OC_InitStructure);
49
50   TIM_CtrlPWMOutputs(TIM1, ENABLE);
51
52   TIM_Cmd(TIM1, ENABLE);
53 }
54
55 /*
56   Sets the rotation of the servo
```

```
57      uint8_t angle
58        --> angle in degrees from 0 - 180
59    */
60  void Servo_SetRotation(uint8_t angle)
61  {
62    uint32_t anglesec = (uint32_t)ServoPWM_LowerAngle + ((((uint32_t) angle * ↙
          ↳ 100) / 18) * (uint32_t)ServoPWM_RotationRange) / 1000;
63    ServoPWM_Pulse = (uint16_t)(((uint32_t)ServoPWM_Period / 20 * anglesec) / ↙
          ↳ 1000);
64    TIM1->CCR2 = ServoPWM_Pulse;
65  }
```

Initialisierung Servo PWM
Methode: *Init_Servo()*

Die Funktion *Init_Servo* initialisiert die benötigte PWM über den Timer 1 des µC. Dabei wird die Periodendauer und die Pulsdauer der PWM aus der eingestellten Taktfrequenz automatisch berechnet. Über die Variablen *ServoPWM_LowerAngle* & *ServoPWM_UpperAngle* kann das einstellbare untere und obere Limit des Dutycycles in µs festgelegt werden. Je nach Motor können diese Limits von den in Abbildung 31 gezeigten Werten abweichen und so ermöglicht das Modul eine komfortable Möglichkeit diese Limits auszutesten. (siehe Codeabschnitt 3)

Setzen der Motorrotation
Methode: *Servo_SetRotation(uint8_t angle)*

Die Methode *Servo_SetRotation* wird zum Setzen der Motor Rotation verwendet. Wie Gleichung 6.3 zeigt, wird für die benötigten Berechnungen ein Fixpoint System angewandt, um nicht mit floats rechnen zu müssen und dadurch Rechenleistung zu sparen. (siehe Codeabschnitt 3)

$$t_{diff} = t_{high} - t_{low} \tag{6.1}$$

$$t_{pulse} = t_{low} + \frac{\left(\left(\frac{angle \cdot 100}{18}\right) \cdot t_{diff}\right)}{1000} \tag{6.2}$$

$$t_{tim1pulse} = \frac{\left(\frac{T_{PWM}}{20} \cdot t_{pulse}\right)}{1000} \tag{6.3}$$

Hierbei sind die Klammern, auch wenn sie rein mathematisch nicht notwendig wären, aufgrund der Datentypen, welche keine Kommazahlen erlauben, wichtig.

6.4 MPU & HMC Kommunikation

6.4.1 Allgemeines

Dieses Modul wird für die Kommunikation und Datenübertragung zwischen dem MPU6050, dem HMC5883L und dem µC benötigt. Dabei wird zuerst die Initalisierung der beiden ICs über IIC vorgenommen und anschließend eine Automation der Datenabfrage eingestellt. Der MPU6050 wird so konfiguriert, dass er alle 2ms die Daten seiner Sensoren (inklusive des externen Sensors HMC5883L) ausliest und anschließend einen Spannungspuls generiert, welcher beim µC einen Interrupt auslöst und dieser wiederum über einen DMA die Daten vom MPU ausliest.

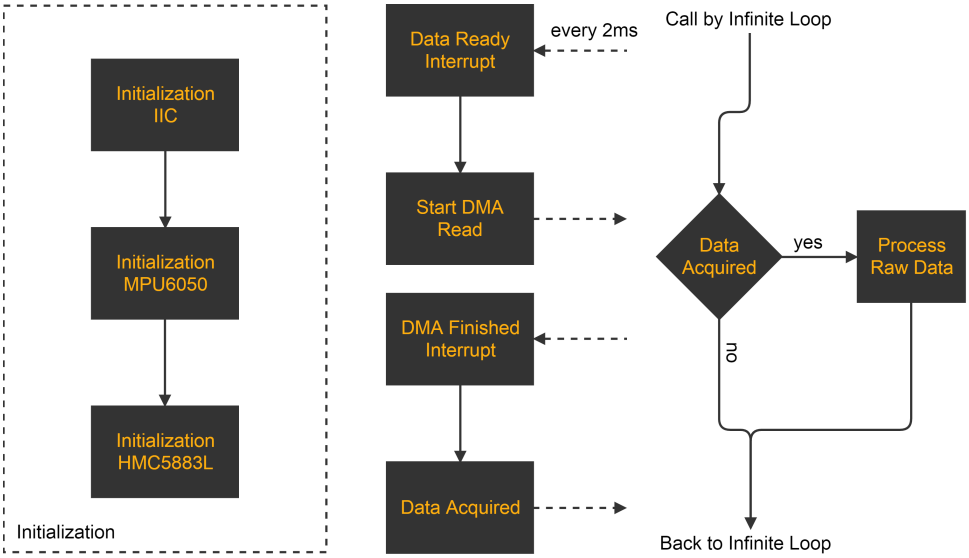


Abbildung 32: MPU & HMC Datenaustausch

Abbildung 32 zeigt den prinzipiellen Ablauf des Moduls, mit den Interrupts und Verarbeitung der Rohdaten.

6.4.2 C-Code

Der Code des Moduls ist in zwei Header-Files und ein Source-File getrennt, wobei das eine Header-File die Definitionen für den MPU6050 und das andere die für den HMC5883L beinhaltet.

6.4.2.1 MPU6050 Header-File

Codeabschnitt 4: MPU6050 Header-File

```
1  #ifndef MPU6050_H_
2  #define MPU6050_H_
3
4  ...
5
6  #define MPU6050_GYRO_DIVIDER 16384
7  #define MPU6050_ACCEL_DIVIDER 208713
8  #define MPU6050_TEMP_DIVIDER 340000
9  #define MPU6050_TEMP_OFFSET 3653
10
```

```
11 typedef struct
12 {
13   int16_t xaccel, yaccel, zaccel;
14   int16_t temp;
15   int16_t xgyro, ygyro, zgyro;
16   int16_t xmag, ymag, zmag;
17 } MPU6050_RawData;
18
19 typedef struct
20 {
21   int32_t xaccel, yaccel, zaccel;
22 } MPU6050_AccelData;
23
24 typedef struct
25 {
26   int32_t xgyro, ygyro, zgyro;
27 } MPU6050_GyroData;
28
29 // Initialization
30 void Init_MPU6050();
31 void Init_MPU6050_I2C();
32 void Init_MPU6050_EXTI();
33
34 // Data
35 MPU6050_AccelData MPU6050_GetAccelData();
36 uint32_t MPU6050_GetTempData();
37 MPU6050_GyroData MPU6050_GetGyroData();
38 void MPU6050_CopyDataToBuffer();
39
40 // IO
41 void I2C1_Write(uint8_t slaveAddr, uint8_t regAddr, uint8_t data);
42 void I2C1_ByteWrite(u8 slaveAddr, u8* pBuffer, u8 writeAddr);
43 void MPU6050_DMA_Read(u8 slaveAddr, u8 readAddr, u16 NumByteToRead);
44
45 #endif /* __MPU6050_H */
```

Im Codeabschnitt 4 werden die unwichtigen Definitionen ausgelassen und nur die relvanten Informationen gezeigt. Zu sehen sind Teiler für die Umrechnung der Rohdaten sowie Structdefinitionen zur übersichtlichen Speicherung der Daten. Anschließend bietet das Header-File einen Überblick über die verwendeten Methoden, welche im Anhang zum Source-File genauer beschrieben werden.

6.4.2.2 HMC5883L Header-File

Codeabschnitt 5: HMC5883L Header-File

```
1  #ifndef HMC5883L_H_
2  #define HMC5883L_H_
3
4  ...
5
6  typedef struct
7  {
8    int32_t xmag, ymag, zmag;
9  } HMC5883L_MagData;
10
11 void Init_HMC5883L();
12
13 #endif /* HMC5883L_H_ */
```

Im Codeabschnitt 4 werden die unwichtigen Definitionen ausgelassen und nur die relevanten Informationen gezeigt. Aufgrund der Automation sind im Header-File des Kompass nur wenige Definitionen zu sehen. Neben dem Strukt der empfangenen Daten gibt es lediglich eine Intialisierungs-Funktion.

6.4.2.3 MPU6050 Source-File

Das Source-File ist aufgrund seiner Komplexität, nach zusammenhängenden Bereichen, in mehrere bedeutende Codeabschnitte geteilt.

Codeabschnitt 6: MPU6050 Source-File Deklarationen

```
1  #include "stm32f10x.h"
2  #include "mpu6050.h"
3  #include "hmc5883l.h"
4  #include "timebase.h"
5
6  I2C_TypeDef * MPU6050_I2C;
7  DMA_Channel_TypeDef * MPU6050_DMA_Channel;
8  MPU6050_RawData I2C_Rx_Buffer;
9
10 MPU6050_AccelData MPU6050_AccelDataBuff;
11 MPU6050_GyroData MPU6050_GyroDataBuff;
12 int32_t MPU6050_TempDataBuff;
13
14 volatile u8 MPU6050_DataAquiredFlag = 0;
```

Die benötigten Files sowie die Variablen und Buffer werden zu Beginn deklariert und sind in Codeabschnitt 6 gezeigt.

Codeabschnitt 7: MPU6050 Source-File Daten Umwandlung

```
1  MPU6050_AccelData MPU6050_GetAccelData()
2  {
3    return MPU6050_AccelDataBuff;
4  }
5  uint32_t MPU6050_GetTempData()
6  {
7    return MPU6050_TempDataBuff;
8  }
9  MPU6050_GyroData MPU6050_GetGyroData()
10 {
11   return MPU6050_GyroDataBuff;
12 }
13
14 void MPU6050_CalculateAccelData()
15 {
16   (&MPU6050_AccelDataBuff)->xaccel = ((int32_t)(&I2C_Rx_Buffer)->xaccel * ↙
       ↳ 100000) / MPU6050_ACCEL_DIVIDER;
17   (&MPU6050_AccelDataBuff)->yaccel = ((int32_t)(&I2C_Rx_Buffer)->yaccel * ↙
       ↳ 100000) / MPU6050_ACCEL_DIVIDER;
18   (&MPU6050_AccelDataBuff)->zaccel =((int32_t)(&I2C_Rx_Buffer)->zaccel * ↙
       ↳ 100000) / MPU6050_ACCEL_DIVIDER;
19 }
20 void MPU6050_CalculateTempData()
21 {
22   MPU6050_TempDataBuff = ((int32_t)(&I2C_Rx_Buffer)->temp * 100000) / ↙
       ↳ MPU6050_TEMP_DIVIDER + MPU6050_TEMP_OFFSET;
23 }
24 void MPU6050_CalculateGyroData()
25 {
```

```
26    (&MPU6050_GyroDataBuff)->xgyro = ((int32_t)(&I2C_Rx_Buffer)->xgyro * 100000) ↙
          ↳ / MPU6050_GYRO_DIVIDER;
27    (&MPU6050_GyroDataBuff)->ygyro = ((int32_t)(&I2C_Rx_Buffer)->ygyro * 100000) ↙
          ↳ / MPU6050_GYRO_DIVIDER;
28    (&MPU6050_GyroDataBuff)->zgyro =((int32_t)(&I2C_Rx_Buffer)->zgyro * 100000) ↙
          ↳ / MPU6050_GYRO_DIVIDER;
29  }
30
31  int16_t Swap2Bytes(int16_t bytes)
32  {
33    int16_t tmp = bytes;
34    bytes = (u8)(bytes >> 8);
35    bytes |= (u16)(tmp << 8);
36    return bytes;
37  }
38
39  void MPU6050_SwapBytes(MPU6050_RawData * data)
40  {
41    data->temp = Swap2Bytes(data->temp);
42    data->xaccel = Swap2Bytes(data->xaccel);
43    data->yaccel = Swap2Bytes(data->yaccel);
44    data->zaccel = Swap2Bytes(data->zaccel);
45    data->xgyro = Swap2Bytes(data->xgyro);
46    data->ygyro = Swap2Bytes(data->ygyro);
47    data->zgyro = Swap2Bytes(data->zgyro);
48  }
49
50  void MPU6050_CopyDataToBuffer()
51  {
52    if (MPU6050_DataAquiredFlag)
53    {
54      MPU6050_SwapBytes(&I2C_Rx_Buffer);
55      MPU6050_CalculateAccelData();
56      MPU6050_CalculateTempData();
57      MPU6050_CalculateGyroData();
58      MPU6050_DataAquiredFlag = 0;
59    }
60  }
```

Die Rohdaten müssen nach Empfang von den Sensoren in für den Regler verwertbare Daten umgewandelt werden (siehe Codeabschnitt 7). Dafür sind folgende Methoden notwendig:

Auslesen der Beschleunigungssensordaten
Methode: *MPU6050_AccelData MPU6050_GetAccelData()*

Diese Methode ermöglicht den Zugriff auf die aktuellen berechneten Daten des Beschleunigungssensors von einem anderen Modul aus.

Auslesen der Temperatursensordaten
Methode: *uint32_t MPU6050_GetTempData()*

Diese Methode ermöglicht den Zugriff auf die aktuellen berechneten Daten des Temperatursensors von einem anderen Modul aus.

Auslesen der Gyrosensordaten
Methode: *MPU6050_GyroData MPU6050_GetGyroData()*

Diese Methode ermöglicht den Zugriff auf die aktuellen berechneten Daten des Gyrosensors von einem anderen Modul aus.

Berechnung der Beschleunigungssensordaten
Methode: *MPU6050_CalculateAccelData()*

Über diese Methode werden die Beschleunigungssensordaten aus den Rohdaten, durch Multiplikation mit einem Faktor, berechnet.

Berechnung der Temperatursensordaten
Methode: *MPU6050_CalculateTempData()*

Über diese Methode werden die Daten des Temperatursensors aus den Rohdaten, durch Multiplikation mit einem Faktor und Verschiebung durch einen Offset berechnet.

Berechnung der Gyrosensordaten
Methode: *MPU6050_GyroData MPU6050_GetGyroData()*

Über diese Methode werden die Gyrosensordaten aus den Rohdaten, durch Multiplikation mit einem Faktor berechnet.

Byte Tausch Helper
Methode: *int16_t Swap2Bytes(int16_t bytes)*

Dieser Helper übernimmt das einfache Tauschen der zwei Bytes eines Word.

Byte Tausch der Daten
Methode: *MPU6050_SwapBytes(MPU6050_RawData * data)*

Mithilfe dieser Funktion werden die in der falschen Reihenfolge empfangenen Bytes der MPU-Daten getauscht. Die Reihenfolge ergibt sich aus den Registern des MPU6050 und kann somit nicht beeinflusst werden.

Berechnung der aktuellen Sensordaten
Methode: *MPU6050_CopyDataToBuffer()*

Diese Methode wird von der Hauptschleife des Programms aufgerufen und prüft, ob Daten zur Verarbeitung vorhanden sind. Falls dem so ist, werden die Daten präpariert, umgerechnet und in die entsprechenden Buffer geschrieben, aus welchen sie über Get-Methoden extern abgefragt werden können.

Codeabschnitt 8: MPU6050 Source-File MPU Initialisierung

```
1  void Init_MPU6050_I2C()
2  {
3
4    GPIO_InitTypeDef GPIO_InitStructure;
5    NVIC_InitTypeDef NVIC_InitStructure;
6    I2C_InitTypeDef I2C_InitStructure;
7    DMA_InitTypeDef  DMA_InitStructure;
8
9    RCC_APB1PeriphClockCmd(RCC_APB1Periph_I2C1, ENABLE);
10   RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOB | RCC_APB2Periph_AFIO, ENABLE);
11   RCC_AHBPeriphClockCmd(RCC_AHBPeriph_DMA1, ENABLE);
12
13   /* I2C RX & TX */
14   GPIO_InitStructure.GPIO_Pin = GPIO_Pin_6 | GPIO_Pin_7;
15   GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_OD;
16   GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
17   GPIO_Init(GPIOB, &GPIO_InitStructure);
18
19   /* I2C */
20   MPU6050_I2C = I2C1;
21
```

```
22    I2C_DeInit(I2C1);
23
24    I2C_InitStructure.I2C_Ack = I2C_Ack_Enable;
25    I2C_InitStructure.I2C_AcknowledgedAddress = I2C_AcknowledgedAddress_7bit;
26    I2C_InitStructure.I2C_ClockSpeed = 400000;
27    I2C_InitStructure.I2C_DutyCycle = I2C_DutyCycle_2;
28    I2C_InitStructure.I2C_Mode = I2C_Mode_I2C;
29    I2C_InitStructure.I2C_OwnAddress1 = 0;
30    I2C_Init(I2C1, &I2C_InitStructure);
31    I2C_Cmd(I2C1, ENABLE);
32    delay_ms(100);
33    /* DMA */
34
35    MPU6050_DMA_Channel = DMA1_Channel7;
36
37    DMA_DeInit(MPU6050_DMA_Channel); //reset DMA1 channe1 to default values;
38
39    DMA_InitStructure.DMA_PeripheralBaseAddr = (uint32_t)&I2C1->DR; ↙
          ↳ //=0x40005410 : address of data reading register of I2C1
40    DMA_InitStructure.DMA_MemoryBaseAddr = (uint32_t)&I2C_Rx_Buffer; //variable ↙
          ↳ to store data
41    DMA_InitStructure.DMA_M2M = DMA_M2M_Disable; //channel will be used for ↙
          ↳ peripheral to memory transfer
42    DMA_InitStructure.DMA_Mode = DMA_Mode_Normal;//DMA_Mode_Normal; //setting ↙
          ↳ normal mode (non circular)
43    DMA_InitStructure.DMA_Priority = DMA_Priority_Medium; //medium priority
44    DMA_InitStructure.DMA_DIR = DMA_DIR_PeripheralSRC;  //Location assigned to ↙
          ↳ peripheral register will be source
45    DMA_InitStructure.DMA_BufferSize = 20;  //number of data to be transfered
46    DMA_InitStructure.DMA_PeripheralInc = DMA_PeripheralInc_Disable; //automatic ↙
          ↳ memory increment disable for peripheral
47    DMA_InitStructure.DMA_MemoryInc = DMA_MemoryInc_Enable; //automatic memory ↙
          ↳ increment enable for memory
48    DMA_InitStructure.DMA_PeripheralDataSize = DMA_PeripheralDataSize_Byte; ↙
          ↳ //source peripheral data size = 8bit
49    DMA_InitStructure.DMA_MemoryDataSize = DMA_MemoryDataSize_Byte; ↙
          ↳ //destination memory data size = 8bit
50    DMA_Init(MPU6050_DMA_Channel, &DMA_InitStructure);
51    DMA_ITConfig(MPU6050_DMA_Channel, DMA_IT_TC, ENABLE);
52
53    NVIC_InitStructure.NVIC_IRQChannel = DMA1_Channel7_IRQn; //I2C1 connect to ↙
          ↳ channel 7 of DMA1
54    NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0x05;
55    NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0x05;
56    NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
57    NVIC_Init(&NVIC_InitStructure);
58
59  }
60
61  void Init_MPU6050()
62  {
63    //reset the whole module first
64    I2C1_Write(MPU6050_DEFAULT_ADDRESS, MPU6050_RA_PWR_MGMT_1, 1<<7);
65
66    delay_ms(50); //wait for 50ms for the gyro to stable
67
68    //PLL with Z axis gyroscope reference
69    I2C1_Write(MPU6050_DEFAULT_ADDRESS, MPU6050_RA_PWR_MGMT_1, ↙
          ↳ MPU6050_CLOCK_PLL_ZGYRO);
70
71    //DLPF_CFG = 1: Fs=1khz; bandwidth=42hz
72    I2C1_Write(MPU6050_DEFAULT_ADDRESS, MPU6050_RA_CONFIG, 0x01);
```

```
73
74    //500Hz sample rate ~ 2ms
75    I2C1_Write(MPU6050_DEFAULT_ADDRESS, MPU6050_RA_SMPLRT_DIV, 0x01);
76
77    //Gyro full scale setting
78    I2C1_Write(MPU6050_DEFAULT_ADDRESS, MPU6050_RA_GYRO_CONFIG, ↙
          ↳ MPU6050_GYRO_FS_2000);
79
80    //Accel full scale setting
81    I2C1_Write(MPU6050_DEFAULT_ADDRESS, MPU6050_RA_ACCEL_CONFIG, ↙
          ↳ MPU6050_ACCEL_FS_16);
82
83    //interrupt status bits are cleared on any read operation
84    I2C1_Write(MPU6050_DEFAULT_ADDRESS, MPU6050_RA_INT_PIN_CFG, 1<<4);
85
86    //interupt occurs when data is ready
87    I2C1_Write(MPU6050_DEFAULT_ADDRESS, MPU6050_RA_INT_ENABLE, 1<<0);
88
89    //reset gyro and accel sensor
90    I2C1_Write(MPU6050_DEFAULT_ADDRESS, MPU6050_RA_SIGNAL_PATH_RESET, 0x07);
91  }
92
93  void Init_MPU6050_EXTI()
94  {
95      /* Enable EXTI */
96      GPIO_InitTypeDef GPIO_InitStructure;
97      NVIC_InitTypeDef NVIC_InitStructure;
98      EXTI_InitTypeDef EXTI_InitStructure;
99
100     GPIO_InitStructure.GPIO_Pin = GPIO_Pin_15;
101     GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN_FLOATING;
102     GPIO_Init(GPIOB, &GPIO_InitStructure);
103     GPIO_SetBits(GPIOB, GPIO_Pin_15); // pullup
104
105     GPIO_EXTILineConfig(GPIO_PortSourceGPIOB, GPIO_PinSource15);
106
107     EXTI_InitStructure.EXTI_Line = EXTI_Line15;
108     EXTI_InitStructure.EXTI_Mode = EXTI_Mode_Interrupt;
109     EXTI_InitStructure.EXTI_Trigger = EXTI_Trigger_Rising;
110     EXTI_InitStructure.EXTI_LineCmd = ENABLE;
111     EXTI_Init(&EXTI_InitStructure);
112
113     NVIC_PriorityGroupConfig(NVIC_PriorityGroup_4);
114
115     NVIC_InitStructure.NVIC_IRQChannel = EXTI15_10_IRQn;
116     NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
117     NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0x0F;
118     NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0x0F;
119     NVIC_Init(&NVIC_InitStructure);
120 }
121
122 void EXTI15_10_IRQHandler(void)
123 {
124   if (EXTI_GetITStatus(EXTI_Line15))      //MPU6050_INT
125   {
126     EXTI_ClearITPendingBit(EXTI_Line15);
127
128     DMA_I2C1_Read(MPU6050_DEFAULT_ADDRESS, MPU6050_RA_ACCEL_XOUT_H, 20);
129   }
130 }
```

Die Initialisierung des MPU6050 umfasst das richtige Konfigurieren des IIC Interfaces, des DMA, welcher zum Auslesen der Daten verwendet wird, des externen Interrupts und das tatsächliche Beschreiben der Register des MPU über IIC. Dabei wird erst das IIC Interface und der DMA initialisert und anschließend die Konfigurationsregister des MPU und HMC beschrieben. Zuletzt wird der externe Interrupt aktiviert und dadurch das automatische Auslesen der Daten gestartet. (siehe Codeabschnitt 8)

Initialisierung IIC Interface und DMA
Methode: *Init_MPU6050_I2C()*

Über diese Methode wird das IIC Interface und der DMA initialisiert, wobei zuerst die nötige Peripherie sowie die I/O Pins aktiviert werden. Das IIC Interface läuft mit einer Taktfrequenz von 400kHz. Der DMA wird für das I2C1 Interface des µC und Output zu Buffer konfiguriert und bietet somit eine sehr schnelle Möglichkeit, automatisch Daten direkt vom IIC Interface in den Speicher zu schreiben. Für den DMA wird ein Transmission Complete Interrupt eingerichtet, welcher nach Einlesen der Daten aufgerufen wird.

Beschreiben der MPU6050 Konfigurationsregister
Methode: *Init_MPU6050()*

Nach Initialisierung der IIC Schnittstelle wird der MPU6050 entsprechend konfiguriert. Dazu wird der IC zuerst einem Reset unterzogen. Anschließend wird die Clock Quelle auf die z-Achsen des Gyroskops gesetzt. Das interne Tiefpassfilter für Gyrosensor und Beschleunigungssensor wird eingestellt (siehe Gleichung 6.4). Die Abtastrate der Sensordaten wird auf 2ms und die Skalierung von Gyro- und Beschleunigungssensor auf Full Scale eingestellt. Aus diesen Skalierungen ergeben sich auch die Teiler für die Umrechnung der Rohdaten. Zum Schluss wird noch der Data Ready Interrupt, welcher an den µC weitergeleitet wird, aktiviert und die Sensoren zurückgesetzt.

$$f_S = 1kHz \qquad B_{gyro} = 188Hz \qquad B_{accel} = 184Hz \tag{6.4}$$

Beschreiben der MPU6050 Konfigurationsregister
Methode: *Init_MPU6050_EXTI()*

Der externe Interrupt, welcher alle 2ms durch ein Signal vom MPU6050 ausgelöst wird, wird über diese Methode initialisiert. Dabei wird der benötigte I/O Pin mit Pullup und entsprechender externer Interrupt Line aktiviert.

Beschreiben der MPU6050 Konfigurationsregister
Methode: *EXTI15_10_IRQHandler(void)*

Dies ist der externe Interrupt Handler, der alle 2ms aufgerufen wird. In ihm wird der DMA Lese Vorgang der Daten vom MPU6050 gestartet.

Codeabschnitt 9: MPU6050 Source-File HMC Initialisierung

```
1  void Init_HMC5883L()
2  {
3    // enable multi master mode; wait for external data before creating the ↙
       ↳ interrupt; i2c clkspeed = 400kHz
4    I2C1_Write(MPU6050_DEFAULT_ADDRESS, MPU6050_RA_I2C_MST_CTRL, 0b11011101);
5
6    // disable i2c master mode
7    I2C1_Write(MPU6050_DEFAULT_ADDRESS, MPU6050_RA_USER_CTRL, 0);
8
9    // set mpu6050 i2c interface to pass through
10   I2C1_Write(MPU6050_DEFAULT_ADDRESS, MPU6050_RA_INT_PIN_CFG, (1 << 4) | (1 << ↙
       ↳ 1));
11
```

```
12    // set output Rate to 75 Hz
13    I2C1_Write(HMC5883L_ADDRESS, HMC5883L_RA_CONFIG_A, 0b00011000);
14
15    // set gain to +-1.3 Gauss
16    I2C1_Write(HMC5883L_ADDRESS, HMC5883L_RA_CONFIG_B, 0b10100000);
17
18    // set mode to continuous
19    I2C1_Write(HMC5883L_ADDRESS, HMC5883L_RA_MODE, 0b00000000);
20
21    // transmit slave address of external sensor
22    I2C1_Write(MPU6050_DEFAULT_ADDRESS, MPU6050_RA_I2C_SLV0_ADDR, (1 << 7) | ↙
          ↳ HMC5883L_DEFAULT_ADDRESS);
23
24    // transmit first register address of external sensor
25    I2C1_Write(MPU6050_DEFAULT_ADDRESS, MPU6050_RA_I2C_SLV0_REG, ↙
          ↳ HMC5883L_RA_DATAX_H);
26
27    // transmission with length = 6
28    I2C1_Write(MPU6050_DEFAULT_ADDRESS, MPU6050_RA_I2C_SLV0_CTRL, 0b11000110);
29
30    // set disable pass through
31    I2C1_Write(MPU6050_DEFAULT_ADDRESS, MPU6050_RA_INT_PIN_CFG, (1 << 4));
32
33    // enable i2c master mode
34    I2C1_Write(MPU6050_DEFAULT_ADDRESS, MPU6050_RA_USER_CTRL, (1 << 5));
35  }
```

Nach der Initialisierung der IIC Schnittstelle und Konfigurieren des MPU6050 wird der Kompass IC konfiguriert. (siehe Codeabschnitt 9)

Beschreiben der MPU6050 & HMC5883L Konfigurationsregister

Methode: *Init_HMC5883L()*

Mithilfe dieser Methode wird der Kompass IC konfiguriert. Erst wird der Multi Master Modus des MPU6050 aktiviert, da er später gleichzeitig als Slave und Master agieren muss. Um den Kompass IC aber anfänglich zu konfigurieren, wird der Pass Through Modus des MPU6050 eingeschalten. So kann der µC direkt mit dem HMC5883L kommunizieren. Der Kompass IC wird dann auf eine Samplerate von 75Hz und eine Sensitivität von +-1.3 Gauss eingestellt sowie in den kontinuierlichen Modus versetzt, in welchem er periodisch die Sensordaten ausliest und speichert. Anschließend wird der MPU6050 so konfiguriert, dass er alle 2ms an der richtigen Stelle im Speicher des HMC5883L diese Messdaten ausliest und wiederrum speichert. Abschließend wird der Pass Through Modus ausgeschaltet und der automatische Betrieb ist bereit zur Verwendung.

Codeabschnitt 10: MPU6050 Source-File DMA

```
1  void DMA1_Channel7_IRQHandler(void)
2  {
3    if (DMA_GetFlagStatus(DMA1_FLAG_TC7))
4    {
5      /* Clear transmission complete flag */
6      DMA_ClearFlag(DMA1_FLAG_TC7);
7
8      I2C_DMACmd(MPU6050_I2C, DISABLE);
9      /* Send I2C1 STOP Condition */
10     I2C_GenerateSTOP(MPU6050_I2C, ENABLE);
11     /* Disable DMA channel*/
12     DMA_Cmd(MPU6050_DMA_Channel, DISABLE);
13
14     MPU6050_DataAquiredFlag = 1;
15   }
```

```
16 }
17
18 void DMA_I2C1_Read(u8 slaveAddr, u8 readAddr, u8 bytes)
19 {
20   /* Disable DMA channel*/
21   DMA_Cmd(MPU6050_DMA_Channel, DISABLE);
22   /* Set current data number again to 14 for MPu6050, only possible after ↙
         ↳ disabling the DMA channel */
23   DMA_SetCurrDataCounter(MPU6050_DMA_Channel, bytes);
24
25   /* While the bus is busy */
26   while(I2C_GetFlagStatus(MPU6050_I2C, I2C_FLAG_BUSY));
27
28   /* Enable DMA NACK automatic generation */
29   I2C_DMALastTransferCmd(MPU6050_I2C, ENABLE);
30
31   /* Send START condition */
32   I2C_GenerateSTART(MPU6050_I2C, ENABLE);
33
34   /* Test on EV5 and clear it */
35   while(!I2C_CheckEvent(MPU6050_I2C, I2C_EVENT_MASTER_MODE_SELECT));
36
37   /* Send MPU6050 address for write */
38   I2C_Send7bitAddress(MPU6050_I2C, slaveAddr, I2C_Direction_Transmitter);
39
40   /* Test on EV6 and clear it */
41   while(!I2C_CheckEvent(MPU6050_I2C, ↙
         ↳ I2C_EVENT_MASTER_TRANSMITTER_MODE_SELECTED));
42
43   /* Clear EV6 by setting again the PE bit */
44   I2C_Cmd(MPU6050_I2C, ENABLE);
45
46   /* Send the MPU6050's internal address to write to */
47   I2C_SendData(MPU6050_I2C, readAddr);
48
49   /* Test on EV8 and clear it */
50   while(!I2C_CheckEvent(MPU6050_I2C, I2C_EVENT_MASTER_BYTE_TRANSMITTED));
51
52   /* Send STRAT condition a second time */
53   I2C_GenerateSTART(MPU6050_I2C, ENABLE);
54
55   /* Test on EV5 and clear it */
56   while(!I2C_CheckEvent(MPU6050_I2C, I2C_EVENT_MASTER_MODE_SELECT));
57
58   /* Send MPU6050 address for read */
59   I2C_Send7bitAddress(MPU6050_I2C, slaveAddr, I2C_Direction_Receiver);
60
61   /* Test on EV6 and clear it */
62   while(!I2C_CheckEvent(MPU6050_I2C, I2C_EVENT_MASTER_RECEIVER_MODE_SELECTED));
63
64   /* Start DMA to receive data from I2C */
65   DMA_Cmd(MPU6050_DMA_Channel, ENABLE);
66   I2C_DMACmd(MPU6050_I2C, ENABLE);
67 }
```

Für den DMA der IIC Schnittstelle werden eine Methode zum Starten des Lese-Vorgangs und ein Interrupt Handler benötigt. (siehe Codeabschnitt 10)

DMA Auslese-Vorgang
Methode: *DMA_I2C1_Read(u8 slaveAddr, u8 readAddr, u8 bytes)*

Das automatische Auslesen der Sensordaten und Speichern über DMA wird mit dieser Methode, welche alle 2ms aufgerufen wird, bewerkstelligt. Dabei wird dem DMA die Anzahl der zu lesenden Bytes mitgeteilt und die automatische NACK Generation aktiviert. Anschließend wird der erste IIC Vorgang manuell initiiert und der DMA aktiviert. Das Lesen aller weiterer Bytes läuft automatisiert ab.

DMA Transmission Complete Handler
Methode: *DMA1_Channel7_IRQHandler(void)*

Nach Abschluss des Auslese-Vorgangs, wirft der DMA einen Interrupt, welcher in diesem Handler abgehandelt wird. Um keine langen Rechenvorgänge im Interrupt zu vollziehen, wird lediglich der Auslesevorgang durch ein IIC Stop beendet, der DMA deaktiviert und das *MPU6050_DataAquiredFlag* gesetzt.

Codeabschnitt 11: MPU6050 Source-Fil IIC

```
1  void I2C1_Write(uint8_t slaveAddr, uint8_t regAddr, uint8_t data)
2  {
3      uint8_t tmp;
4      tmp = data;
5      I2C1_ByteWrite(slaveAddr,&tmp,regAddr);
6  }
7
8  void I2C1_ByteWrite(u8 slaveAddr, u8* pBuffer, u8 writeAddr)
9  {
10   /* Send START condition */
11   I2C_GenerateSTART(MPU6050_I2C, ENABLE);
12   /* Test on EV5 and clear it */
13   while(!I2C_CheckEvent(MPU6050_I2C, I2C_EVENT_MASTER_MODE_SELECT));
14   /* Send MPU6050 address for write */
15   I2C_Send7bitAddress(MPU6050_I2C, slaveAddr, I2C_Direction_Transmitter);
16   /* Test on EV6 and clear it */
17   while(!I2C_CheckEvent(MPU6050_I2C, ↙
         ↳ I2C_EVENT_MASTER_TRANSMITTER_MODE_SELECTED));
18   /* Send the MPU6050's internal address to write to */
19   I2C_SendData(MPU6050_I2C, writeAddr);
20   /* Test on EV8 and clear it */
21   while(!I2C_CheckEvent(MPU6050_I2C, I2C_EVENT_MASTER_BYTE_TRANSMITTING));
22   /* Send the byte to be written */
23   if (pBuffer!=0) I2C_SendData(MPU6050_I2C, *pBuffer);
24   /* Test on EV8_2 and clear it */
25   while(!I2C_CheckEvent(MPU6050_I2C, I2C_EVENT_MASTER_BYTE_TRANSMITTED));
26   /* Send STOP condition */
27   I2C_GenerateSTOP(MPU6050_I2C, ENABLE);
28 }
```

Für die Kommunikation über IIC stellt ST für den STM32 eine Standardbibliothek mit einigen hilfreichen Routinen zur Verfügung, welche für eine Sendefunktion verwendet werden. (siehe Codeabschnitt 11)

IIC Sende-Vorgang
Methode: *void I2C1_ByteWrite(u8 slaveAddr, u8 * pBuffer, u8 writeAddr)*

Mit dieser Methode wird ein Byte in eine bestimmte Speicheradresse eines Slaves geschrieben. Dazu wird ein IIC Start generiert und die Adresse des Slaves übermittelt. Anschließend wird die Adresse des zu beschreibenden Registers und die Daten gesendet. Zum Schluss wird ein IIC Stop generiert um die Übertragung zu beenden.

6.5 BLDC Motor Steuerung

6.5.1 Allgemeines

Dieses Modul ermöglicht die Kommunikation mit den drei A4963 BLDC Motortreibern und liefert die benötigten drei PWM Steuersignale über die Timer des GD32. Anfänglich wird die SPI Schnittstelle und die Timer für die Steuersignale initialisiert und anschließend die Konfiguration der Motortreiber über SPI vorgenommen. Für das SPI Interface wird zusätzlich ein Data Received Interrupt eingerichtet, in welchem von den Treibern empfangene Daten in einen Ringbuffer geschrieben werden. Anhand dieser Daten werden die Treiber über einen Aufruf aus der Dauerschleife der Firmware auf Fehler überprüft und bei Bedarf neu gestartet. Die Geschwindigkeit des Motors sowie dessen Drehrichtung können über zwei Routinen von externen Modulen, wie z.B. dem, Regler gesetzt werden.

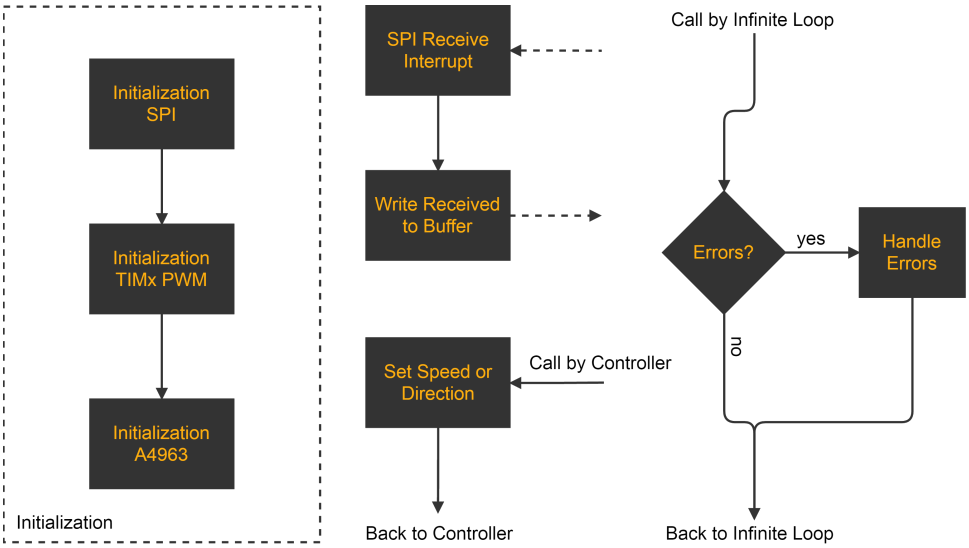


Abbildung 33: BLDC Motor Steuerung

6.5.2 C-Code

Der Code des Moduls besteht aus zwei Header- und einem Source-File, wobei die Header File nach Register- und Bitdefinitionen sowie nach Methoden und Variablen getrennt sind.

6.5.2.1 A4963 Header-File

Codeabschnitt 12: A4963 Header-File

```
1  #ifndef A4963_H_
2  #define A4963_H_
3
4  #define MOTOR1_GPIO     GPIOA
5  #define MOTOR1_CS_PIN     GPIO_Pin_11
6  #define MOTOR1_TIM      TIM2
7  #define MOTOR2_GPIO     GPIOB
8  #define MOTOR2_CS_PIN     GPIO_Pin_5
9  #define MOTOR2_TIM      TIM3
10 #define MOTOR3_GPIO     GPIOB
11 #define MOTOR3_CS_PIN   GPIO_Pin_13
```

```
12 #define MOTOR3_TIM      TIM4
13 #define SPI_BUFMASK     0b01111111 // 127
14
15 typedef enum {
16   MOTOR1,
17   MOTOR2,
18   MOTOR3
19 } Motor;
20 typedef enum {
21   FORWARD = 0,
22   REVERSE = 1,
23 } Direction;
24
25 void Init_A4963();
26 void Init_A4963_SPI1();
27
28 void Init_Motor1();
29 void Init_Motor2();
30 void Init_Motor3();
31
32 void A4963_SetMotorDirection(Motor mot, Direction dir);
33 void A4963_SetSpeed(Motor mot, uint8_t speed);
34
35 uint16_t * GetMotorTIM_Pulse(Motor mot, uint16_t ** pulse);
36 GPIO_TypeDef * GetMotorCS(Motor mot, uint16_t * CS_Pin);
37
38 void SPI1_SendData(GPIO_TypeDef*  GPIO_CS_Pin, uint16_t CS_Pin, uint16_t word);
39 void SPI1_SendConfiguration(Motor mot);
40
41 #endif /* A4963_H_ */
```

Im Header File für Methoden und Variablen (siehe Codeabschnitt 12) sind die Chip Select Pins, die dazugehörigen GPIO sowie die Timer zu den Motortreibern definiert. Neben Buffermaske und Enumerationen, welche das Programmieren erleichtern, sind die Initialisierungsroutinen, die auf die SPI bezogenen Routinen und Routinen zur Motorsteuerung deklariert.

6.5.2.2 A4963 Source-File

Das Source-File ist aufgrund seiner Komplexität, nach zusammenhängenden Bereichen, in mehrere bedeutende Codeabschnitte geteilt.

Codeabschnitt 13: A4963 Source-File: Deklarationen

```
1  #include <stddef.h>
2  #include <string.h>
3
4  #include "stm32f10x.h"
5  #include "a4963_def.h"
6  #include "a4963.h"
7  #include "timebase.h"
8
9  volatile uint16_t spiDataBuffer[128];
10 volatile uint8_t spiRxCounter = 0;
11 uint16_t Control_PWM_Period = 10000;
12 uint16_t Control_PWM_Pulse_MOT1 = 5000;
13 uint16_t Control_PWM_Pulse_MOT2 = 5000;
14 uint16_t Control_PWM_Pulse_MOT3 = 5000;
```

Neben den Includes der benötigten externen Files wird der Buffer und die Variablen für den PWM Pulse der drei Timer deklariert. (siehe Codeabschnitt 13)

Codeabschnitt 14: A4963 Source-File: SPI

```
1  void Init_A4963_SPI1()
2  {
3    GPIO_InitTypeDef GPIO_InitStructure;
4    NVIC_InitTypeDef NVIC_InitStructure;
5    SPI_InitTypeDef SPI_InitStructure;
6
7    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA | RCC_APB2Periph_GPIOB | ↙
         ↳ RCC_APB2Periph_AFIO | RCC_APB2Periph_SPI1, ENABLE);
8
9    // SPI Pins [MISO(PA6), MOSI(PA7), SCK(PA5)]
10   GPIO_InitStructure.GPIO_Pin = GPIO_Pin_5 | GPIO_Pin_7;
11   GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP;
12   GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
13   GPIO_Init(GPIOA, &GPIO_InitStructure);
14
15   GPIO_InitStructure.GPIO_Pin = GPIO_Pin_6;
16   GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN_FLOATING;
17   GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
18   GPIO_Init(GPIOA, &GPIO_InitStructure);
19
20   // Chip Select Pins (PA11, PB5, PB13)
21   GPIO_InitStructure.GPIO_Pin = GPIO_Pin_11;
22   GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
23   GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
24   GPIO_Init(GPIOA, &GPIO_InitStructure);
25
26   GPIO_InitStructure.GPIO_Pin = GPIO_Pin_5 | GPIO_Pin_13;
27   GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
28   GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
29   GPIO_Init(GPIOB, &GPIO_InitStructure);
30
31   GPIO_WriteBit(GPIOA, GPIO_Pin_11, SET);
32   GPIO_WriteBit(GPIOB, GPIO_Pin_5, SET);
33   GPIO_WriteBit(GPIOB, GPIO_Pin_13, SET);
34
35   // Read Interrupt
36   NVIC_InitStructure.NVIC_IRQChannel = SPI1_IRQn;
37   NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;
38   NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;
39   NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
40   NVIC_Init(&NVIC_InitStructure);
41
42
43   // SPI Initialization
44   SPI_InitStructure.SPI_BaudRatePrescaler = SPI_BaudRatePrescaler_32;
45   SPI_InitStructure.SPI_CPHA = SPI_CPHA_2Edge;
46   SPI_InitStructure.SPI_CPOL = SPI_CPOL_High;
47   SPI_InitStructure.SPI_CRCPolynomial = 0;
48   SPI_InitStructure.SPI_DataSize = SPI_DataSize_16b;
49   SPI_InitStructure.SPI_Direction = SPI_Direction_2Lines_FullDuplex;
50   SPI_InitStructure.SPI_FirstBit = SPI_FirstBit_MSB;
51   SPI_InitStructure.SPI_Mode = SPI_Mode_Master;
52   SPI_InitStructure.SPI_NSS = SPI_NSS_Soft;
53   SPI_Init(SPI1, &SPI_InitStructure);
54
55   SPI_I2S_ITConfig(SPI1, SPI_I2S_IT_RXNE, ENABLE);
56
57   SPI_Cmd(SPI1, ENABLE);
58 }
59
```

```
60 void SPI1_IRQHandler(void)
61 {
62   if(SPI_I2S_GetFlagStatus(SPI1, SPI_I2S_FLAG_RXNE) == SET)
63   {
64     spiDataBuffer[spiRxCounter] = SPI_I2S_ReceiveData(SPI1);
65     spiRxCounter++;
66     spiRxCounter &= SPI_BUFMASK;
67   }
68 }
69
70 void SPI1_SendData(GPIO_TypeDef*  GPIO_CS_Pin, uint16_t CS_Pin, uint16_t word)
71 {
72   GPIO_WriteBit(GPIO_CS_Pin, CS_Pin, RESET);
73   delay_us(1);
74   while(SPI_I2S_GetFlagStatus(SPI1, SPI_I2S_FLAG_TXE) != SET);
75   SPI_I2S_SendData(SPI1, word);
76   delay_us(1);
77   GPIO_WriteBit(GPIO_CS_Pin, CS_Pin, SET);
78   delay_us(1);
79 }
```

Die Kommunikation mit den A4963 Motortreibern erfolgt über die SPI-Schnittstelle des µC, welche mit den zusammenhängenden Pins (MISO[7], MOSI[8] und Chip Select) initialisiert wird. MISO und MOSI sind dabei einmalig für alle Treiber zu initialisieren. Die Chip Select Leitung ist jedoch für alle drei Motortreiber vorhanden und je nach elektrischem Pegel wird über diese Leitungen entschieden, welcher Chip über SPI angesprochen wird. Zusätzlich wird der Data Received Interrupt der SPI Schnittstelle aktiviert, um empfangene Daten in einem Ringbuffer zu speichern. (siehe Codeabschnitt 14)

Initialisierung der SPI-Schnittstelle
Methode: *Init_A4963_SPI1()*

Die Initialisierungsroutine für die SPI-Schnittstelle, in welcher zusätzlich die Peripherie, GPIO und Chip Select Pins aktiviert werden. Das SPI-Interface wird mit CPHA = 1 und CPOL = 1 (siehe Abbildung 34), einer Datengröße von 16 Bit und dem MSB als erstes Bit initialisiert.

SPI Data Received Handler
Methode: *SPI1_IRQHandler(void)*

In diesem Handler werden die über die SPI-Schnittstelle empfangenen Daten in einen Ringbuffer mit einer Größe von 128 Words gespeichert. Die Buffer Maske wird verwendet, um den Ringbuffer auf eine Ressourcen sparende Art zu implementieren.

SPI Sende Routine
Methode: *SPI1_SendData(GPIO_TypeDef * GPIO_CS_Pin, uint16_t CS_Pin, uint16_t word)*

Mithilfe dieser Routine können zwei Bytes an Daten über das SPI-Interface versendet werden. Dabei wird der Funktion zusätzlich die GPIO des Chip Select Pins und der Pin selbst mitgegeben, um ein das Auswählen des gewünschten Motortreibers zu vollziehen.

[7] Master In Slave Out
[8] Master Out Slave In

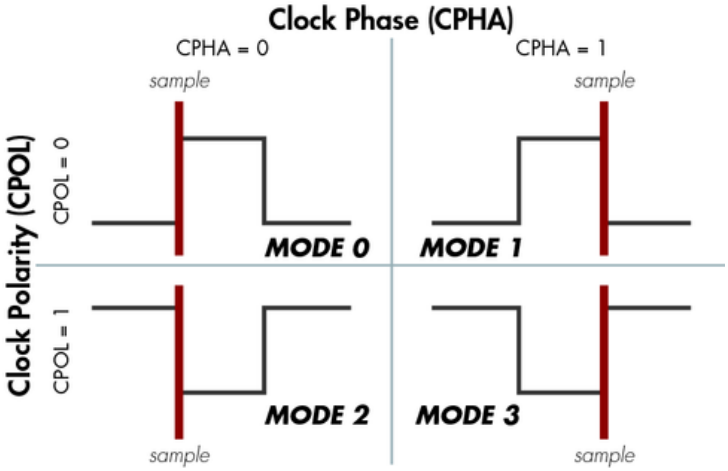


Abbildung 34: SPI Modes[12]

Codeabschnitt 15: A4963 Source-File: Initialisierung

```
1  void Init_A4963()
2  {
3    // Period for all Timers
4    RCC_ClocksTypeDef clk;
5    RCC_GetClocksFreq(&clk);
6    Control_PWM_Period = clk.HCLK_Frequency / 10000;
7    Control_PWM_Pulse_MOT1 = Control_PWM_Period / 3;
8    Control_PWM_Pulse_MOT2 = Control_PWM_Period / 2;
9    Control_PWM_Pulse_MOT3 = Control_PWM_Period / 2;
10
11   // Initialize Motor Control PWMs
12   Init_Motor1();
13   Init_Motor2();
14   Init_Motor3();
15
16   // Configure A4963 for all Motors
17   SPI1_SendConfiguration(MOTOR1);
18   SPI1_SendConfiguration(MOTOR2);
19   SPI1_SendConfiguration(MOTOR3);
20 }
21
22 void Init_Motor1()  // TIM2 PWM
23 {
24     RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM2, ENABLE);
25     RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA | RCC_APB2Periph_AFIO, ENABLE);
26
27     GPIO_InitTypeDef gpioObj;
28     TIM_TimeBaseInitTypeDef TIM_TimeBase_InitStructure;
29     TIM_OCInitTypeDef TIM_OC_InitStructure;
30
31     // Pullups for FAULTN (PA8) & SPD (PA4) Pin
32     gpioObj.GPIO_Mode = GPIO_Mode_IPU;
33     gpioObj.GPIO_Speed = GPIO_Speed_50MHz;
34     gpioObj.GPIO_Pin = GPIO_Pin_4 | GPIO_Pin_8;
35     GPIO_Init(GPIOA, &gpioObj);
36     GPIO_WriteBit(GPIOA, GPIO_Pin_4, SET);
37     GPIO_WriteBit(GPIOA, GPIO_Pin_8, SET);
38
39     // AF for PWM (PA1) Pin
40     gpioObj.GPIO_Mode = GPIO_Mode_AF_PP;
```

```
41      gpioObj.GPIO_Speed = GPIO_Speed_50MHz;
42      gpioObj.GPIO_Pin = GPIO_Pin_1;
43      GPIO_Init(GPIOA, &gpioObj);
44
45      TIM_TimeBase_InitStructure.TIM_ClockDivision = TIM_CKD_DIV1;
46      TIM_TimeBase_InitStructure.TIM_CounterMode = TIM_CounterMode_Up;
47      TIM_TimeBase_InitStructure.TIM_Period = Control_PWM_Period;
48      TIM_TimeBase_InitStructure.TIM_Prescaler = 10;
49      TIM_TimeBaseInit(TIM2, &TIM_TimeBase_InitStructure);
50
51      TIM_OC_InitStructure.TIM_OCMode = TIM_OCMode_PWM2;
52      TIM_OC_InitStructure.TIM_OCIdleState = TIM_OCIdleState_Reset;
53      TIM_OC_InitStructure.TIM_OCNIdleState = TIM_OCNIdleState_Set;
54      TIM_OC_InitStructure.TIM_OCPolarity = TIM_OCPolarity_Low;
55      TIM_OC_InitStructure.TIM_OCNPolarity = TIM_OCNPolarity_High;
56      TIM_OC_InitStructure.TIM_OutputState = TIM_OutputState_Enable;
57      TIM_OC_InitStructure.TIM_OutputNState = TIM_OutputNState_Disable;
58      TIM_OC_InitStructure.TIM_Pulse = Control_PWM_Pulse_MOT1;
59      TIM_OC2Init(TIM2, &TIM_OC_InitStructure);
60
61      TIM_Cmd(TIM2, ENABLE);
62  }
```

Nach Initialisierung des SPI-Interfaces wird der Wert der Timer Periode berechnet, diese drei Timer im PWM Modus aktiviert und die Konfigurationsdaten der Motortreiber über SPI übertragen. (siehe Codeabschnitt 15)

Initialisierung des Moduls
Methode: *Init_A4963()*

Diese Routine übernimmt die Initialisierung der Motortreiber, indem zuerst die Timer Periode berechnet wird und die Timer Initialsierungs- sowie die SPI Konfigurations Sende-Routinen aufgerufen werden.

Initialisierung Timer und I/O Pins
Methode: *Init_MotorX()*

Exemplarisch für die restlichen Routinen findet sich im Codeabschnitt 14 die Funktion *Init_Motor1()*, mit welcher die Timer PWM für das Steuersignal der Motortreiber und die I/O Pins initialisiert werden. Dazu wird erst die entsprechende Peripherie und die GPIOs aktiviert und anschließend der Timer mit einer Frequenz von 1kHz konfiguriert. Zuletzt aktiviert die Funktion die PWM und den vom Timer abhängigen Output Pin.

Codeabschnitt 16: A4963 Source-File: A4963 Motor Konfiguration

```
1   void A4963_SetSpeed(Motor mot, uint8_t speed) // 10% - 100%
2   {
3     uint16_t * pulse;
4     uint16_t * TIMx_CCRx = GetMotorTIM_Pulse(mot, &pulse);
5     *pulse = (uint16_t)(((uint32_t)Control_PWM_Period * (uint32_t)speed) / 100);
6     *TIMx_CCRx = *pulse;
7   }
8
9   void A4963_SetMotorDirection(Motor mot, Direction dir)
10  {
11    uint16_t CS_Pin;
12    GPIO_TypeDef * GPIO_CS_Pin = GetMotorCS(mot, &CS_Pin);
13    SPI1_SendData(GPIO_CS_Pin, CS_Pin, REG_RUN | REG_WRITE | RUN_ENABLE | ↙
          ↳ RESTART_ENABLE | CONTROL_CLOSED_SPEED | (dir << DIR)); // run
14  }
15
```

```
16 uint16_t * GetMotorTIM_Pulse(Motor mot, uint16_t ** pulse)
17 {
18   switch (mot)
19   {
20     case MOTOR1:
21       *pulse = &Control_PWM_Pulse_MOT1;
22       return &(MOTOR1_TIM->CCR2);
23     case MOTOR2:
24       *pulse = &Control_PWM_Pulse_MOT2;
25       return &(MOTOR2_TIM->CCR3);
26     case MOTOR3:
27       *pulse = &Control_PWM_Pulse_MOT3;
28       return &(MOTOR3_TIM->CCR3);
29       break;
30   }
31   return 0;
32 }
33
34 GPIO_TypeDef * GetMotorCS(Motor mot, uint16_t * CS_Pin)
35 {
36   switch (mot)
37   {
38     case MOTOR1:
39       *CS_Pin = MOTOR1_CS_PIN;
40       return MOTOR1_GPIO;
41     case MOTOR2:
42       *CS_Pin = MOTOR2_CS_PIN;
43       return MOTOR2_GPIO;
44     case MOTOR3:
45       *CS_Pin = MOTOR3_CS_PIN;
46       return MOTOR3_GPIO;
47       break;
48   }
49   return 0;
50 }
51
52 void SPI1_SendConfiguration(Motor mot)
53 {
54   uint16_t CS_Pin;
55   GPIO_TypeDef * GPIO_CS_Pin = GetMotorCS(mot, &CS_Pin);
56   SPI1_SendData(GPIO_CS_Pin, CS_Pin, REG_CONFIG0 | REG_WRITE | ↙
       ↳ RECIRCULATION_AUTO | BLANK_TIME(8) | DEAD_TIME(20)); // config 0
57   SPI1_SendData(GPIO_CS_Pin, CS_Pin, REG_CONFIG1 | REG_WRITE | ↙
       ↳ PERCENT_FAST_DECAY_12_5 | CURRENT_LIMIT(9) | VDS_DEBOUNCE_MODE | ↙
       ↳ VDS_THRESHOLD(31)); // config 1
58   SPI1_SendData(GPIO_CS_Pin, CS_Pin, REG_CONFIG2 | REG_WRITE | ↙
       ↳ POSITION_P_GAIN(7) | OVERSPEED_150| INDIRECT_PWM_PERIOD(19)); // ↙
       ↳ config 2
59   SPI1_SendData(GPIO_CS_Pin, CS_Pin, REG_CONFIG3 | REG_WRITE | ↙
       ↳ POSITION_I_GAIN(7) | HOLD_DUTY_CYCLE(4) | HOLD_TIME(2)); // config 3
60   SPI1_SendData(GPIO_CS_Pin, CS_Pin, REG_CONFIG4 | REG_WRITE | SPEED_P_GAIN(7) ↙
       ↳ | STARTUP_DUTY_CYCLE(7) | START_SPEED(4)); // config 4
61   SPI1_SendData(GPIO_CS_Pin, CS_Pin, REG_CONFIG5 | REG_WRITE | SPEED_I_GAIN(7) ↙
       ↳ | SPEED_FG | MAXIMUM_SPEED(7) | PHASE_ADVANCE(4)); // config 5
62   SPI1_SendData(GPIO_CS_Pin, CS_Pin, REG_RUN | REG_WRITE | RUN_ENABLE | ↙
       ↳ RESTART_ENABLE | CONTROL_CLOSED_SPEED); // run
63 }
```

Nach Konfiguration der SPI, Timer und I/O werden die Konfigurationsregister der Motortreiber beschrieben werden. Daraufhin befinden sich die Treiber im richtigen Modus und werden die Halbbrücken (siehe 5.11) so angesteuert, dass sich die Geschwindigkeit der Motoren entsprechend

dem PWM Steuersignal anpasst. Durch Verändern dieser PWM Signale kann nun über eine Routine die Geschwindigkeit eingestellt werden. Um die Drehrichtung eines Motors zu ändern, kann über eine weitere Routine die benötigte Konfiguration an den Treiber geschickt werden. (siehe Codeabschnitt 16)

Setzen der Motorgeschwindigkeit
Methode: *A4963_SetSpeed(Motor mot, uint8_t speed)*

Durch Angabe des Motors und der Geschwindigkeit von 10 - 100 (in Prozent), kann der Duty Cycle der PWM des Steuersignals verändert werden und somit auch die Geschwindigkeit des BLDC Motors selbst.

Ändern der Drehrichtung
Methode: *A4963_SetMotorDirection(Motor mot, Direction dir*

Bei entsprechenden Parametern kann die Drehrichtung eines Motors angepasst werden. Dabei versendet die Funktion die dazu benötigten Daten mithilfe des SPI-Interface an den über den Parameter *mot* angegebenen Motortreiber.

Timer zu Motor Helper
Methode: *uint16_t * GetMotorTIM_Pulse(Motor mot, uint16_t ** pulse)*

Diese Helper Routine liefert durch Angabe des Motors den Pointer auf den dazugehörigen Timer, sowie den Pointer auf die Variable, welche den dafür eingestellten Pulswert (Duty Cycle) beinhaltet.

GPIO und CS Pin zu Motor Helper
Methode: *GPIO_TypeDef * GetMotorCS(Motor mot, uint16_t * CS_Pin)* Mithilfe dieser Helper Routine wird der Chip Select Pin und die damit verbundene GPIO ermittelt. Dabei ist der Pointer auf die GPIO Typen Definiton der Rückgabewert der Methode. Der Wert des Chip-Select Pin wird in eine Variable über Call by Reference gespeichert.

Treiberkonfiguration
Methode: *SPI1_SendConfiguration(Motor mot)*

Diese Funktion übernimmmt die Konfiguration der Motortreiber über die SPI-Schnittstelle. Erst wird ermittelt, welchem Motor die Konfiguration gesendet werden soll, woraufhin die Register beschrieben werden. Neben den wichtigsten Einstellungen wie dem Betriebsmodus (siehe 5.10.4), der Drehrichtung, Fehlerverhalten und Aktivierung gibt es zahlreiche Einstellungen für die Ansteuerung des BLDC Motors. Der A4963 verüfgt über einen PI-Regler für die Position und Geschwindigkeit des Motors, wessen Parameter eingestellt werden. Zusätzlich wird die Start-, die maximale Geschwindigkeit, der Phasenvorsrpung und einige andere Einstellungen, abhängig vom Motor, der betrieben wird, vorgenommen.

6.6 GPS-Modul Kommunikation

6.6.1 Allgemeines

Die Ermittlung der aktuellen Position des Luftschiffes wird über dieses Modul vollzogen. Dabei wird die benötigte USART Schnittstelle initalisiert und das Modul anschließend entsprechend eingestellt. Beim Empfangen der Daten des GPS-Moduls werden diese in einem Ringbuffer gespeichert, um später durch einen Aufruf aus der Hauptschleife verarbeitet zu werden. Dabei wird die benötigte Position herausgearbeitet und aufgrund des DOP Wertes, welcher ebenfalls in den Daten enthalten ist, verworfen oder gespeichert. Die aktuelle Position ergibt sich durch die Mittelung aller gültiger Positionswerte und kann über eine Routine von externen Modulen abgefragt werden. (siehe Abbildung 35)

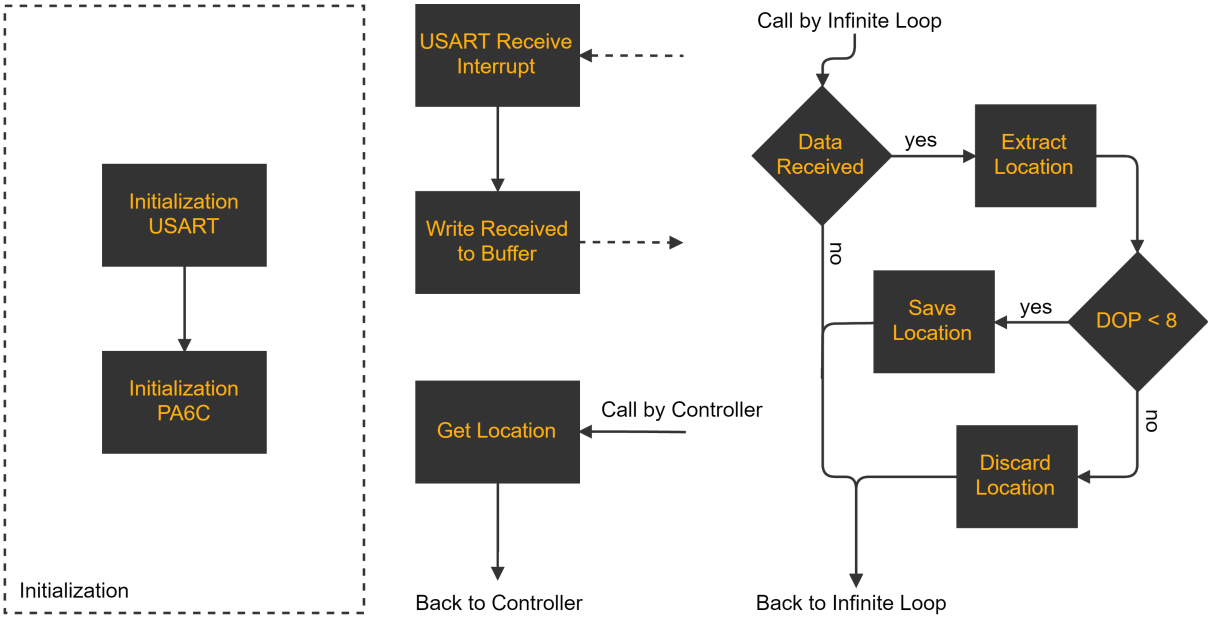


Abbildung 35: GPS-Modul Ablauf

6.6.2 C-Code

6.6.2.1 GPS Kommunikation Header-File

Codeabschnitt 17: GPS Header-File

```
1  /* MTK Commands */
2  #define PMTK_TEST              "$PMTK000"
3  #define PMTK_CMD_AIC_MODE          "$PMTK286"
4  #define PMTK_CMD_EASY_ENABLE         "$PMTK869"
5  #define PMTK_CMD_PERIODIC_MODE         "$PMTK225"
6  #define PMTK_SET_NMEA_BAUDRATE         "$PMTK251"
7  #define PMTK_API_SET_SBAS_ENABLED      "$PMTK313"
8  #define PMTK_API_SET_SBAS_MODE         "$PMTK319"
9  #define PMTK_API_SET_DGPS_MODE         "$PMTK301"
10 #define PMTK_API_SET_SUPPORT_QZSS_NMEA    "$PMTK351"
11 #define PMTK_API_SET_STOP_QZSS         "$PMTK352"
12 #define PMTK_API_SET_NMEA_OUTPUT       "$PMTK314"
13 #define PMTK_SET_NMEA_UPDATERATE       "$PMTK220"
14
```

```
15 #define GPS_BUFMASK 0b01111111 // 127
16
17 void Init_GPS_USART();
18 void Init_GPS();
19
20 void PMTK_SendCmd(char * cmd, char * data);
```

Codeabschnitt 17 zeigt einen wichtigen Abschnitt des Header-Files des Moduls. Darin werden die bedeutendsten MTK NMEA 0183 Kommando Header sowie die Initialisierungsfunktionen und Sendefunktion zum GSM-Modul definiert.

6.6.2.2 GPS Kommunikation Source-File

Das Source-File ist aufgrund seiner Komplexität, nach zusammenhängenden Bereichen, in mehrere bedeutende Codeabschnitte geteilt, wobei das Auslesen der Position sowie dessen Auswertung und Abfrage auf der beigelegten CD zu finden sind.

Codeabschnitt 18: GPS Source-File Deklarationen

```
1  #include <stddef.h>
2  #include <string.h>
3  #include "stm32f10x.h"
4  #include <stm32f10x_usart.h>
5  #include "gps.h"
6
7  volatile char buff[1024];
8  volatile uint16_t writepointer = 0;
9  volatile uint16_t readpointer = 0;
10
11 char hex_to_char[] = {'0', '1', '2', '3', '4', '5', '6', '7', '8', '9', 'A' , ↙
       ↳ 'B', 'C', 'D', 'E', 'F'};
```

Neben den Includes der benötigten externen Files wird der Buffer und ein Lookup Array von Hexadezimal-Buchstaben für das Speichern der Checksumme bei der Datenübertragung zum GPS-Modul deklariert. (siehe Codeabschnitt 13)

Codeabschnitt 19: GPS Source-File USART Initialisierung

```
1  void Init_GPS_USART()
2  {
3    RCC_APB1PeriphClockCmd(RCC_APB1Periph_USART2, ENABLE);
4    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA | RCC_APB2Periph_AFIO, ENABLE);
5
6    USART_InitTypeDef USART_InitStructure;
7
8    USART_InitStructure.USART_BaudRate = 9600;
9    USART_InitStructure.USART_WordLength = USART_WordLength_8b;
10   USART_InitStructure.USART_StopBits = USART_StopBits_1;
11   USART_InitStructure.USART_Parity = USART_Parity_No;
12   USART_InitStructure.USART_HardwareFlowControl =
13   USART_HardwareFlowControl_None;
14   USART_InitStructure.USART_Mode = USART_Mode_Rx | USART_Mode_Tx;
15
16   /* USART configuration */
17   USART_Init(USART2, &USART_InitStructure);
18
19   /* Enable the USART2 */
20   USART_Cmd(USART2, ENABLE);
21
22   USART_ITConfig(USART2, USART_IT_RXNE, ENABLE);
23   //USART_ITConfig(USART1, USART_IT_TXE, ENABLE);
```

```
24
25    GPIO_InitTypeDef gpioConfig;
26
27    //PA9 = USART1.TX => Alternative Function Output
28    gpioConfig.GPIO_Mode = GPIO_Mode_AF_PP;
29    gpioConfig.GPIO_Pin = GPIO_Pin_2;
30    gpioConfig.GPIO_Speed = GPIO_Speed_2MHz;
31    GPIO_Init(GPIOA, &gpioConfig);
32
33    //PA10 = USART1.RX => Input
34    gpioConfig.GPIO_Mode = GPIO_Mode_IN_FLOATING;
35    gpioConfig.GPIO_Pin = GPIO_Pin_3;
36    GPIO_Init(GPIOA, &gpioConfig);
37
38    NVIC_InitTypeDef NVIC_InitStructure;
39
40    /* Configure the NVIC Preemption Priority Bits */
41    NVIC_PriorityGroupConfig(NVIC_PriorityGroup_0);
42
43    /* Enable the USART1 Interrupt */
44    NVIC_InitStructure.NVIC_IRQChannel = USART2_IRQn;
45    NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;
46    NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;
47    NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
48    NVIC_Init(&NVIC_InitStructure);
49  }
50
51  void USART2_SendByte(char byte) {
52
53    USART_SendData(USART2, byte);
54    while (!USART_GetFlagStatus(USART2, USART_FLAG_TXE));
55  }
56
57  void USART2_IRQHandler(void) {
58
59    if (USART_GetITStatus(USART2, USART_IT_RXNE) != RESET) // Received ↙
        ↳ characters modify string
60    {
61      buff[writepointer++] = (char)USART_ReceiveData(USART2);
62      writepointer &= GPS_BUFMASK;
63    }
64  }
```

Für die Funktionalität der Kommuniktion muss die USART2 Schnittstelle des µC initialisiert werden. Dazu wird die benötigte Peripherie und GPIO sowie ein Data Received Interrupt aktiviert und eingestellt. (siehe Codeabschnitt 19)

Initialisierung der USART2
Methode: *Init_ GPS_ USART()*

Die USART Schnittstelle wird anfänglich über diese Methode mit einer Baudrate von 9600 und einer Übertragungsgröße von 8bit, im normalen RX, TX-Modus, ohne zusätzliche Kontrollflusssteuerung, intitialisiert.

Senden eines Bytes
Methode: *USART2_ SendByte(char byte)*

Das Senden eines Bytes über die USART2 Schnittstelle erfolgt mit dieser Methode, wobei die Standardbibliothek von ST für die entsrpechende Registerkonfiguration verwendet wird. Das Senden von Konfigurationsdaten an das GPS-Modul spielt nur beim Start der Elektronik eine

Rolle und ist daher auch nicht sehr zeitkritisch, was es erlaubt, die Zeit, welche dafür benötigt wird, einfach abzuwarten.

Data Received Handler
Methode: *USART2_IRQHandler(void)*

In diesem Handler werden die eingestellten Daten vom GPS-Modul empfangen und in einen Ringbuffer geschrieben.

Codeabschnitt 20: GPS Source-File Modul Konfiguration

```
1  void PMTK_SendCmd(char * cmd, char * data)
2  {
3    uint8_t checksum = 0;
4    char checksum_h, checksum_l;
5    uint8_t i;
6    USART2_SendByte(cmd[0]);
7    for (i = 1; i < 8; i++)
8    {
9      checksum ^= cmd[i];
10     USART2_SendByte(cmd[i]);
11   }
12   i = 0;
13   while (data[i] != '*')
14   {
15     checksum ^= data[i];
16     USART2_SendByte(data[i]);
17     i++;
18   }
19   USART2_SendByte('*');
20   checksum_h = hex_to_char[(checksum >> 4) & 0x0F];
21   checksum_l = hex_to_char[checksum & 0x0F];
22   USART2_SendByte(checksum_h);
23   USART2_SendByte(checksum_l);
24   USART2_SendByte(13); // <cr>
25   USART2_SendByte(10); // <lf>
26 }
27
28 void Init_GPS()
29 {
30   PMTK_SendCmd(PMTK_TEST, "*");
31
32   PMTK_SendCmd(PMTK_API_SET_NMEA_OUTPUT, ↙
       ↳ ",0,1,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0*");
33
34   PMTK_SendCmd(PMTK_API_SET_SBAS_ENABLED, ",1*");
35   PMTK_SendCmd(PMTK_API_SET_SBAS_MODE, ",1*");
36   PMTK_SendCmd(PMTK_API_SET_DGPS_MODE, ",2*");
37   PMTK_SendCmd(PMTK_CMD_PERIODIC_MODE, ",0*");
38   PMTK_SendCmd(PMTK_CMD_AIC_MODE, ",1*");
39   PMTK_SendCmd(PMTK_CMD_EASY_ENABLE, ",1,1*");
40 }
```

Das Protokoll, welches verwendet wird, um mit dem GPS-Modul zu kommunizieren, ist MTK NMEA 0183, welches neben bestimmten Kommando Schlüsselwörtern eine Checksumme der gesendeten Daten benötigt, die vor dem Senden berechnet und angehängt werden muss.

Versenden eines NMEA Kommando
Methode: *PMTK_SendCmd(char * cmd, char * data)*

Diese Methode übernimmt das Übertragen der Daten durch Angabe des Schlüsselwortes *cmd* und den Parametern *data*. Dabei wird die Checksume des Kommandos durch ein simplex XOR über alle chars der Kommandozeile abzüglich des anfänglichen $ gebildet und in hexadezimal aber wiederrum als char angehängt.

Initialisierung des GPS-Moduls
Methode: *PMTK_SendCmd(char * cmd, char * data)*

Schlussendlich wird das GPS-Modul über diese Funktion in die gewünschte Konfiguration versetzt. Dazu werden nur die benötigten Informationspakete zur Übertragung aktiviert. Dabei handelt es sich um das GNSS DOPS and Active Satellites Datenpaket, welches Informationen zu den Satelliten sowie des DOP Werts liefert und die Recommended Minimum Navigation Information, welche die Koordinaten sowie Kurs und Geschwindigkeit beinhaltet. Anschließend wird der SBAS[9] Modus zur zusätzlichen Genauikeit aktiviert, sowie dessen Testmodus nicht verwendet. Der DGPS Modus (siehe 5.8.4) wird auf WAAS[10] eingestellt, das Modul in den normalen Betriebsmodus versetzt. Zuletzt wird der AIC-Modus[11] und der EASY-Modus aktiviert. Die Konfiguration ist somit abgeschlossen und das GPS-Modul sendet periodisch die eingestellten Daten an den µC.

[9] Satellite-Based Augmentation System
[10] Wide Area Augmentation System
[11] Active Interference Cancellation Mode

6.7 GSM-Modul Kommunikation

6.7.1 Allgemeines

Das GSM-Modul hat einen Chip der Marke SIMCOM, dieser wird aufgrund des minimalen Gewichts, aber auch wegen der trotzdem vorhandenen vollständigen Funktionalität verwendet. Diese Funktionalität wird durch das verwendete Breakout-Board zwar eingeschränkt, stellt jedoch für die aktuelle Anwendung keine Einschränkung dar, weil sowieso nur SMS verschickt werden müssen und diese Funktion auch von der Breakoutboard-Version des Chips unterstützt wird. Das GSM-Modul fungiert als Verbindung zwischen Luftschiff und Benutzer. So kann der Benutzer dem Blimp Kommandos per SMS schicken sowie Statusmeldungen erhalten. Das GSM Modul wertet dann gemeinsam mit dem µC die gesendeten Daten aus. Die Kommunikation zwischen GSM-Modul und µC funktioniert über UART und wird mittels AT-Befehlen bewerkstelligt. Wenn eine SMS-Nachricht vom GSM-Modul empfangen wird, meldet sich dieses über die UART-Schnittstelle beim µC. Dieser liest die Nachricht so schnell wie möglich aus. Daraufhin schreibet der µC die empfangenen Daten in den GSM-Ringbuffer. Die Daten im GSM-Ringbuffer werden dann vom µC in der Dauerschleife ausgewertet, analysiert und natürlich wird entsprechend darauf reagiert.

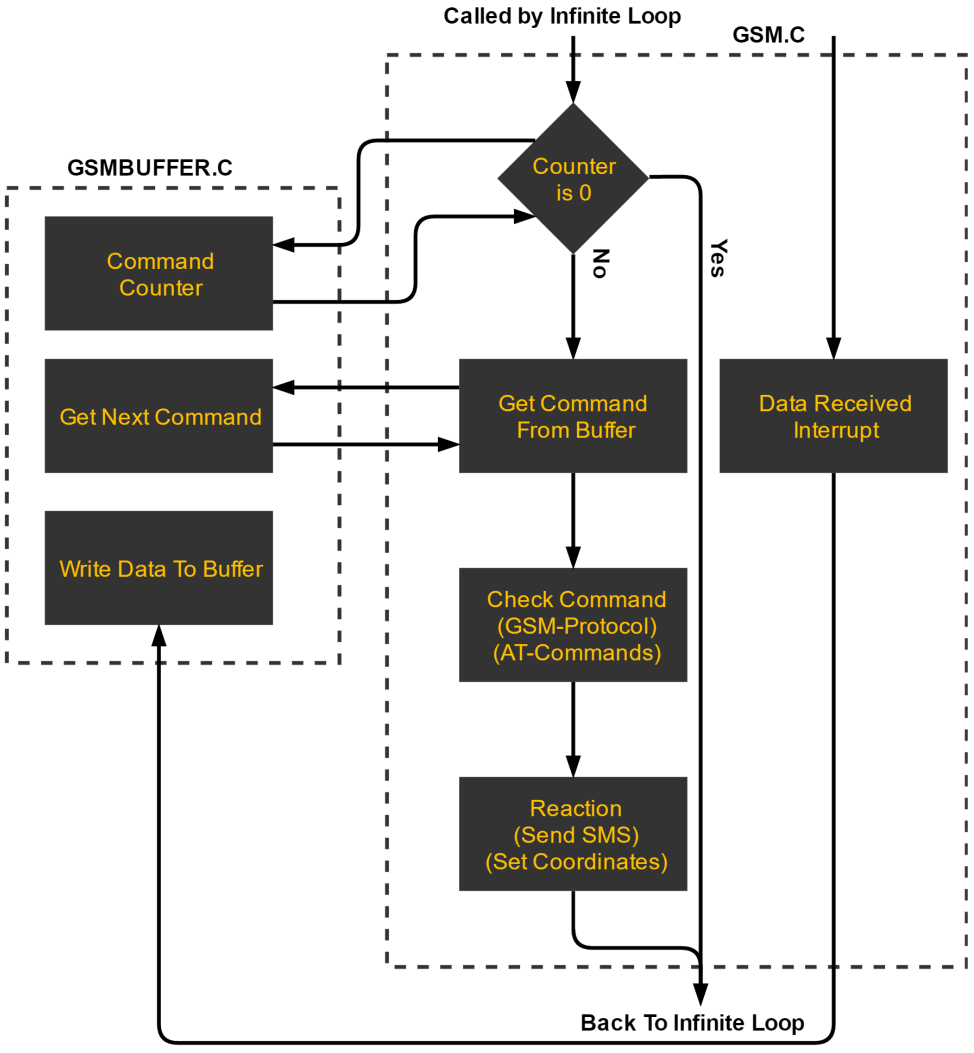


Abbildung 36: GSM-Firmware Programmablauf

In dem Flussdiagramm Abbildung 36 wird der Programmablauf und die Synergie zwischen GSM.C (siehe Codeabschnitt 22 und Codeabschnitt 23)und GSMBUFFER.C (siehe Codeabschnitt 24) kurz und knapp dargestellt.

6.7.1.1 Protokoll

Das Handy, das über GSM kommuniziert, fungiert als Schnittstelle zwischen Benutzer und Luftschiff. Deshalb wird auch ein kleines Protokoll (siehe Tabelle 8 und Tabelle 9) mit Befehlen benötigt.

Im Allgemeinen identifiziert ein "?" eine Abfrage. Eine einfache Nachricht ohne "?" ist ein Kommando.

Kommandos	**Erklärung**
destination "point"	Setzt die Zielkoordinaten auf die in "point" mitgegebenen Daten.
begin	Das Luftschiff fliegt zu den Zielkoordinaten.
stop	Das Luftschiff pausiert den Flugvorgang und wartet auf weitere Kommandos.

Tabelle 8: GSM-Protokoll Teil 1

Abfragen	**Erklärung**
help?	Gibt eine Liste mit allen möglichen Kommandos und Abfragen zurück.
status?	Gibt eine allgemeine Statusmeldung zurück.
voltage?	Gibt den aktuellen Akkustand in % zurück.
position?	Gibt die aktuellen Koordinaten des Luftschiffes zurück.
destination?	Gibt die aktuellen Zielkoordinaten zurück.

Tabelle 9: GSM-Protokoll Teil 2

6.7.2 AT-Befehle

AT-Befehle des GSM-Moduls empfangen

Das GSM-Modul sendet am Ende jedes AT-Befehls ein "\n"[12] [3], um den Befehl abzuschließen. Somit kann mit Hilfe des genannten Zeichens zwischen der Befehlskette separiert werden (siehe Codeabschnitt 24).

6.7.2.1 Text-Mode vs. PDU-Mode

Bei einem GSM-Modul muss entschieden werden, ob entweder der PDU-Mode oder der Text-Mode verwendet wird, wobei der PDU-Mode der modernere Modus von beiden ist, da er Verschlüsselungsmechanismen bietet. Der Text-Mode hingegen sendet alle Daten unverschlüsselt. Verschlüsselung ist aber bei einem Prallluftschiff als Diplomarbeit eher irrelevant. Deshalb wird der einfachere und unkompliziertere Text-Mode gewählt.

[12] Line Feed

6.7.2.2 Nachrichten senden (at+cmgs)

Wenn mit dem GSM-Modul eine Nachricht per SMS gesendet werden soll, muss zuallererst der µC den Befehl "at+cmgs="Telefonnummer"" an das GSM-Modul senden. Daraufhin wird das GSM-Modul den µC auffordern, die gewünschte Nachricht zu senden. Das Ende der Nachricht markiert ein 0x1A. Sobald dieser Wert vom GSM-Modul empfangen wird, sendet dieses die Nachricht an die gewünschte Telefonnummer. Wenn der Sendevorgang erfolgreich war, wird eine kurze Bestätigung an den µC zurückgesendet.

6.7.2.3 Standard Befehlsreaktion

Wenn dem GSM-Modul ein Befehl gesendet wird, sendet dieses zu allererst den empfangen Befehl als Bestätigung zurück. Erst nach dieser Bestätigung wird auf den Befehl reagiert. Meist ist dies ein schlichtes "OK". Es bedeutet, dass der Befehl erfolgreich vom GSM-Modul empfangen wurde und auch dementsprechend erfolgreich verarbeitet worden ist. Da das GSM-Modul von Zeit zu Zeit auch SMS-Nachrichten empfangen wird, ist diese Meldung, dass eine SMS empfangen wurde, welche an den µC weitergeleitet wird, von der zuvor erklärten Regel natürlich ausgenommen.

6.7.2.4 Groß- und Kleinschreibung

Ganz interessant zu beobachten ist, dass wenn man dem GSM-Modul die Befehle großgeschrieben sendet, die Antworten auch großgeschrieben zurückgesendet werden. Wenn aber die Befehle klein geschrieben werden, kommen sie auch klein zurück.

Befehl	**Erklärung**
at+cpin=xxxx	Schickt dem GSM-Modul den bei xxxx eingegebenen PIN-Code.
at+cmgf=1	Setzt die Betriebsart des GSM-Modul auf Text-Mode. Ansonsten würde der PDU-Mode verwendet werden (siehe 6.7.2.1).
at+cmgs="Telefonnummer"	Sendet eine Nachricht an die eingegebene Telefonnummer (siehe 6.7.2.2).
at+cmgl="REC UNREAD"	Fordert alle empfangenen, aber ungelesenen Nachrichten des GSM-Moduls an.
at+cmgd=1,4	Löscht alle auf dem GSM-Modul gespeicherten SMS-Nachrichten.
at+cmee=1	Aktiviert den Modus im GSM-Modul, welcher bei einem Fehler genau normierte Fehlerberichte sendet.

Tabelle 10: GSM AT-Befehle

6.7.3 C-Code

Der Programmteil für die GSM-Kommunikation ist in 2 Teile aufgeteilt. Zum einen das gsm.c Source-File (siehe Codeabschnitt 22 und Codeabschnitt 23) mit dem zugehörigen gsm.h Header-File (siehe Codeabschnitt 21) und zum anderen das GSM-Ringbuffer.c (siehe Codeabschnitt 24) Source-File mit dem (siehe Codeabschnitt 25) GSM-Ringbuffer.h Header-File.

6.7.3.1 GSM Header File

Codeabschnitt 21: GSM-Header

```
1
2  #ifndef GSM_H_
3  #define GSM_H_
4
5  void Init_GSM_USART1(void);
6  void Clear_SendFlag();
7  char Get_SendFlag();
8  void Init_GSM(void);
9  void USART1_SendByte(char byte);
10 void GSM_CheckCMDs(void);
11
12 #define AT "at\r"
13
14 #define PINSET  "at+cpin=3673\r"
15 #define TXTModeSet "at+cmgf=1\r"
16 #define SEND "at+cmgs=\"+4369910280625\"\r" //"at+cmgs=\"+436801449720\"\r"
17 #define READUNREAD "at+cmgl=\"REC UNREAD\"\r"
18 #define READALL "at+cmgl=\"ALL\"\r"
19 #define DeleteAll "at+cmgd=1,4\r"
20 #define DETAILEDERROR "at+cmee=1\r"
21
22 #define msg "SCAVenger\r"
23 #define msgReady "SCAVenger at your service.\r"
24 #define msgStatus "Well, its a little bit cold outside.\r"
25 #define msgHelp "CMDs:\ndestination ↙
       ↳ \"point\"\nbegin\nstop\n\nRequests:\nhelp?\nstatus?\nposition?\ndestination?\r"
26 #define MSGEND 0x1A
27
28 #define InitializeDelay 10000
29 #define StandardDelay 50
30 #define LongDelay 1000
31 #endif
```

Im GSM.h Header-File (siehe Codeabschnitt 21) werden Funktionen des Source-Files deklariert. Zusätzlich werden die benötigten AT-Befehle definiert. Am Ende jedes AT-Befehls muss ein "\r"[13] [3] als Abschluss des Befehls stehen. Die definierten Delays am Ende des Files werden benötigt, wenn dem GSM-Modul viel Ressourcen abverlangt werden und es nicht direkt zurück gesendet werden kann. Das ist der Fall, wenn das GSM-Modul sich in das Netz registriert, dem µC Nachrichten schicken muss und es wäre der Fall, wenn eine SMS gesendet wird, allerdings ist an dieser Stelle ein asynchroner Sendevorgang implementiert.

6.7.3.2 Verzögerungen

Abfragen	Wert	Erklärung
InitializeDelay	10000 ms	Wird bei der Registrierung in das GSM-Netz benötigt.
LongDelay	1000 ms	Wird bei der Abfrage von SMS-Nachrichten vom GSM-Modul benötigt.
StandardDelay	50 ms	Wird nach jedem Befehl benötigt, welcher dem GSM-Modul geschickt wird.

Tabelle 11: GSM-Delays

[13] Carriage Return

6.7.3.3 GSM Source-File

GSM Source-File Initialisierungen

Codeabschnitt 22: GSM Source-File Initialisierungen

```
1  #include <stddef.h>
2  #include <string.h>
3  #include "stm32f10x.h"
4  #include "stm32f10x_usart.h"
5  #include "gsm.h"
6  #include "gsmBuffer.h"
7  #include "clock.h"
8  #include "timebase.h"
9  #include "bluetooth.h"
10 #include "tools.h"
11
12
13 char cmd[512];
14 char temp = 0;
15 volatile char SendFlag = 0;
16
17 void Init_GSM_USART1(void) {
18
19   RCC_APB2PeriphClockCmd(
20   RCC_APB2Periph_USART1 | RCC_APB2Periph_GPIOA | RCC_APB2Periph_AFIO, ENABLE);
21
22   USART_InitTypeDef USART_InitStructure;
23
24   USART_InitStructure.USART_BaudRate = 115200;
25   USART_InitStructure.USART_WordLength = USART_WordLength_8b;
26   USART_InitStructure.USART_StopBits = USART_StopBits_1;
27   USART_InitStructure.USART_Parity = USART_Parity_No;
28   USART_InitStructure.USART_HardwareFlowControl =
29   USART_HardwareFlowControl_None;
30   USART_InitStructure.USART_Mode = USART_Mode_Rx | USART_Mode_Tx;
31
32   /* USART configuration */
33   USART_Init(USART1, &USART_InitStructure);
34
35   /* Enable the USART1 */
36   USART_Cmd(USART1, ENABLE);
37
38   USART_ITConfig(USART1, USART_IT_RXNE, ENABLE);
39   //USART_ITConfig(USART1, USART_IT_TXE, ENABLE);
40
41   GPIO_InitTypeDef gpioConfig;
42
43   //PA9 = USART1.TX => Alternative Function Output
44   gpioConfig.GPIO_Mode = GPIO_Mode_AF_PP;
45   gpioConfig.GPIO_Pin = GPIO_Pin_9;
46   gpioConfig.GPIO_Speed = GPIO_Speed_2MHz;
47   GPIO_Init(GPIOA, &gpioConfig);
48
49   //PA10 = USART1.RX => Input
50   gpioConfig.GPIO_Mode = GPIO_Mode_IN_FLOATING;
51   gpioConfig.GPIO_Pin = GPIO_Pin_10;
52   GPIO_Init(GPIOA, &gpioConfig);
53
54   NVIC_InitTypeDef NVIC_InitStructure;
55
56   /* Configure the NVIC Preemption Priority Bits */
```

```
57    NVIC_PriorityGroupConfig(NVIC_PriorityGroup_0);
58
59    /* Enable the USART1 Interrupt */
60    NVIC_InitStructure.NVIC_IRQChannel = USART1_IRQn;
61    NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;
62    NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;
63    NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
64    NVIC_Init(&NVIC_InitStructure);
65  }
66
67  void Init_GSM(void) {
68    memsetpersonal(&cmd[0], temp, 512);
69    delay_ms(StandardDelay);
70    USART1_SendString(DETAILEDERROR);
71    delay_ms(StandardDelay);
72    USART1_SendString(PINSET);
73    delay_ms(InitializeDelay);
74    USART1_SendString(TXTModeSet);
75    delay_ms(StandardDelay);
76    SendSMS_Test();
77  }
```

Im GSM Source-File Initialisierungen (siehe Codeabschnitt 22) werden die Initialisierungen des GSM.c Source-Files dargestellt. Dazu gehören die Routine *Init_GSM_USART1* und die *Init_GSM Routine*. Die USART1 wird mit folgenden Eigenschaften definiert:

- Die Baudrate beträgt 115200.
- Einer Wortlänge von 8 bit.
- Einem Stopbit.
- Keiner Datenflusssteuerung.
- Der Mode ist "USART empfangen und senden".

Die Pins welche die USART1 verwendet, sind:

- TX –> Pin PA9
- RX –> Pin PA10

Zusätzlich wird natürlich auch ein Empfangs-Interrupt benötigt, welcher mit der *USART1_IRQn* Routine verlinkt ist.
Init_GSM initialisiert das GSM-Modul und das Kommandarray, welches für die Auswertung der Befehle gebraucht wird. Zusätzlich wird das Flag *SendFlag* initialisiert, welches für den asynchronen Sendevorgang von SMS benötigt wird (siehe 6.7.3.1 und 6.7.3.5).
Bevor man mit einem GSM-Modul arbeiten kann, um SMS zu versenden, telefonieren oder ähnliche Tätigkeiten durchführen kann, muss sich das Modul in ein Netz registrieren/einwählen. Also wird zuerst für allgemeine Zwecke der Fehlerberichtmodus aktiviert. Dieser dient zugleich direkt als Kommunikationstest. Dann erfolgt das StandardDelay (siehe Tabelle 11). Nun erfolgt der PINSET Befehl, welcher dem GSM-Modul den PIN-Code sendet. Wenn dieser PIN-Code als korrekt erkannt wird, sendet das GSM-Modul "+CPIN:Ready". Danach wählt sich das GSM-Modul in ein Netz ein. Dies benötigt relativ viel Zeit (ca. 5-10 Sekunden), daher muss hier das InitializeDelay (siehe Tabelle 11) eingesetzt werden. Hat sich das GSM-Modul erfolgreich eingewählt, so sendet es als Bestätigung zuerst "SMS Ready" und danach "Call Ready". Als

Abschluss wird eine kurze SMS gesendet, welche dem Anwender signalisiert, dass alle Initialisierungsroutinen erfolgreich abgeschlossen worden sind.

6.7.3.4 GSM Source-File Routinen

Codeabschnitt 23: GSM Source-File Routinen

```
1  void USART1_SendString(char *string) {
2    uint8_t tx_index = 0;
3    while (tx_index < (strlen(string))) {
4      USART_SendData(USART1, string[tx_index++]);
5
6      while (!USART_GetFlagStatus(USART1, USART_FLAG_TXE))
7        ;
8    }
9  }
10 void USART1_SendByte(char byte) {
11
12   USART_SendData(USART1, byte);
13   while (!USART_GetFlagStatus(USART1, USART_FLAG_TXE))
14     ;
15 }
16 void USART1_IRQHandler(void) {
17
18   if (USART_GetITStatus(USART1, USART_IT_RXNE) != RESET) // Received ↙
       ↳ characters modify string
19       {
20     writeBuffer(USART_ReceiveData(USART1));
21   }
22 }
23 void SendSMS_Test() {
24   delay_ms(StandardDelay);
25   USART1_SendString(SEND);
26   delay_ms(StandardDelay);
27   USART1_SendString(msgReady);
28   delay_ms(StandardDelay);
29   USART1_SendByte(MSGEND);
30   SendFlag = 1;
31 }
32 void SendSMS_Status() {
33   delay_ms(StandardDelay);
34   USART1_SendString(SEND);
35   delay_ms(StandardDelay);
36   USART1_SendString(msgStatus);
37   delay_ms(StandardDelay);
38   USART1_SendByte(MSGEND);
39   SendFlag = 1;
40 }
41 void SendSMS_Help() {
42   delay_ms(StandardDelay);
43   USART1_SendString(SEND);
44   delay_ms(StandardDelay);
45   USART1_SendString(msgHelp);
46   delay_ms(StandardDelay);
47   USART1_SendByte(MSGEND);
48   SendFlag = 1;
49 }
50 void SendSMS_Custom(char* array) {
51   delay_ms(StandardDelay);
52   USART1_SendString(SEND);
53   delay_ms(StandardDelay);
```

```
54    USART1_SendString(array);
55    delay_ms(StandardDelay);
56    USART1_SendByte(MSGEND);
57    SendFlag = 1;
58  }
59  void Read_All() {
60    delay_ms(StandardDelay);
61    USART1_SendString(READALL);
62    delay_ms(LongDelay);
63    USART1_SendString(DeleteAll);
64    delay_ms(LongDelay);
65  }
66  void GSM_CheckCMDs() {
67    if (getCMDCount() > 0 && SendFlag == 0) {
68      if (getNextCMD(&cmd[0]) == 1) {
69
70        if (cmd[0] == '+' && cmd[3] == 'T' && cmd[4] == 'I') {
71          Read_All();
72          lowerCMDCount();
73          return;
74        }
75        switch (cmd[6]) {
76
77        case 'f': //TxTModeSet
78
79          if (getNextCMD(&cmd[0]) == 1) {
80            lowerCMDCount();
81            lowerCMDCount();
82          }
83          break;
84        case 'n': //PINSET
85          if (getNextCMD(&cmd[0]) == 1) {
86            lowerCMDCount();
87            lowerCMDCount();
88
89          }
90          break;
91        case 's': //Send
92          if (getNextCMD(&cmd[0]) == 1) {
93            lowerCMDCount();
94            lowerCMDCount();
95
96          }
97          break;
98        case 'e': //Detailed Error
99
100         if (getNextCMD(&cmd[0]) == 1) {
101           lowerCMDCount();
102           lowerCMDCount();
103
104         }
105         break;
106       case 'd': //Deleted
107
108         if (getNextCMD(&cmd[0]) == 1) {
109           lowerCMDCount();
110           lowerCMDCount();
111         }
112         break;
113       case 'l': //Read
114
115         if (getNextCMD(&cmd[0]) == 1) //Dont Need Phone Number
```

```
116             if (getNextCMD(&cmd[0]) == 1) {
117               lowerCMDCount();
118               lowerCMDCount();
119               lowerCMDCount();
120
121               if (cmd[0] == 'd' && cmd[1] == 'e' && cmd[11] != '?') // destination
122               {
123                 char destination[] = "Destination set!";
124                 SendSMS_Custom(destination);
125               }
126               if (cmd[0] == 'b' && cmd[1] == 'e' && cmd[5] != '?') // begin
127               {
128                 char begin[] = "SCAVenger to ground control: We are off!";
129                 SendSMS_Custom(begin);
130               }
131               if (cmd[0] == 's' && cmd[1] == 't' && cmd[6] != '?') // stop
132               {
133                 char stop[] = "SCAVenger has successfully stopped!";
134                 SendSMS_Custom(stop);
135               }
136               if (cmd[0] == 'h' && cmd[1] == 'e' && cmd[4] == '?') // help?
137               {
138                 SendSMS_Help();
139               }
140               if (cmd[0] == 'p' && cmd[1] == 'o' && cmd[8] == '?') // position?
141               {
142                 char position[] = "Postion: ";
143                 SendSMS_Custom(position);
144               }
145               if (cmd[0] == 'd' && cmd[1] == 'e' && cmd[11] == '?') // ↙
                      ↳ destination?
146               {
147                 char destination[] = "Destination: ";
148                 SendSMS_Custom(destination);
149               }
150               if (cmd[0] == 's' && cmd[1] == 't' && cmd[6] == '?') // status?
151               {
152                 SendSMS_Status();
153               }
154
155
156             }
157           break;
158
159         default:
160           lowerCMDCount();
161           break;
162         }
163       }
164
165     }
166   }
167   char Get_SendFlag()
168   {
169     return SendFlag;
170   }
171   void Clear_SendFlag()
172   {
173     SendFlag = 0;
174   }
```

Im GSM Source File Routinen (siehe Codeabschnitt 23) werden alle Routinen, welche für den

Betrieb des GSM-Moduls benötigt werden, definiert. Diese lauten wie folgt:

1. USART1_SendString(char * string)
2. USART1_SendByte(char byte)
3. USART1_IRQHandler()
4. SendSMS_Test()
5. SendSMS_Help()
6. SendSMS_Custom(char * array)
7. Read_All()
8. GSM_CheckCMDs()
9. char Get_SendFlag()
10. Clear_SendFlag()

USART1-Senderoutinen
Methoden: *USART1_SendString(char * string)* und *USART1_SendByte(char byte)*

Die zwei oben genannten Routinen haben beide den Zweck, Daten an das GSM-Modul zu senden. Sie funktionieren beide nach demselben Prinzip. Zunächst wird ein Byte seriell über die USART-Schnittstelle gesendet. Darauffolgend wird so lange zugewartet, bis das Flag USART_FLAG_TXE gesetzt wird. Dieses Flag gibt an, dass der Sendevorgang abgeschlossen worden ist. Der Unterschied der Methoden *USART1_SendString* und *USART1_SendByte* besteht lediglich darin, dass bei der einen nur ein Byte gesendet wird und bei der anderen ein ganzes Byte-Array.

USART1-Interrupt
Methode: *USART1_IRQHandler()*

Wenn Daten vom GSM-Modul über die USART1 empfangen werden, dann wird der Interrupt *USART_IRQHandler* aufgerufen. Dieser gibt die empfangenen Daten über die Methode *writeBuffer* (siehe 6.7.3.5) and den GSM-Ringbuffer weiter.

SMS-Senderoutinen
Methoden: *SendSMS_Test()*, *SendSMS_Help()* und *SendSMS_Custom(char * array)*

Die SMS-Senderoutinen haben den Zweck, die zu sendenden Daten für die *USART1_SendString* vorzubereiten. Zunächst senden alle den AT-Befehl für "Nachricht senden" (siehe Tabelle 10) "at+cmgs". Darauf folgt der Nachrichteninhalt, der bei jeder Methode unterschiedlich, und bei der *SendSMS_Custom* sogar variabel, ist. Am Ende des Inhalts wird ein Byte mit dem Wert 0x1A angehängt, welches das Ende der SMS markiert. Für dieses letzte Byte wird die Routine *USART1_SendByte* benötigt. Zum Schluss der Routine wird das *SendFlag* für den asynchronen Sendevorgang gesetzt (siehe 6.7.3.5).

SMS lesen
Methoden: *Read_All()*

Wenn in der Routine 6.7.3.4 ermittelt wird, dass das GSM-Modul eine SMS empfangen hat, wird diese prompt mit der Methode *Read_All* abgefragt. Diese Methode sendet im Endeffekt

zum einen dem GSM-Modul den AT-Befehl, um Nachrichten abzufragen: "at+cmgl="REC UNREAD"" (siehe Tabelle 10) und zum anderen den AT-Befehl, um Nachrichten auf dem GSM-Modul zu löschen "at+cmgd=1,4" (siehe Tabelle 10).

Senden-Flag
Methoden: *char Get_SendFlag()* und *Clear_SendFlag()*

Diese beide Routinen haben den einfachen Zweck, das *SendFlag* (siehe Codeabschnitt 22) zu setzen bzw. zurückzusetzen (siehe 6.7.3.5 und 6.7.3.4).

Kommandoüberprüfung
Methode: *GSM_CheckCMDs()*

Die Hauptroutine der GSM-Firmware ist die Methode *GSM_CheckCMDs*. Sie dient dazu, die Informationen des Ringbuffers zu verarbeiten. Dies wird aber nur getan, wenn die Funktion *getCMDCount* (siehe 6.7.3.5) angibt, dass Befehle im Buffer enthalten sind. Wenn also Informationen im Buffer stehen, wird die nächste Information bzw. Befehl über die Routine *getNextCMD* (siehe 6.7.3.5) ausgelesen und in das Array "cmd" (siehe Codeabschnitt 22) geladen. Nun wird das Kommando auf seinen Inhalt geprüft. Da das GSM-Modul zunächst immer den empfangenen Befehl als Bestätigung zurücksendet (siehe 6.7.2.3), muss das Array auf alle versendbaren Befehle hin überprüft werden. Nach dieser Überprüfung wird der nächste Befehl aus dem GSM-Buffer (siehe 6.7.3.5) ermittelt. Dieses ermittelte Kommando enthält die eigentliche Nachricht und wird dann noch einmal auf folgende Übereinstimmungen überprüft (siehe Tabelle 8 und Tabelle 9). Je nachdem, wie viele Kommandos ausgelesen werden müssen, um zu ermitteln, welche Informationen empfangen wurden, wird dementsprechend der Kommandozähler (siehe 6.7.3.5) dekrementiert.

6.7.3.5 GSM-Buffer Source File

Codeabschnitt 24: GSM-Buffer Source-File

```
1  #include <stdint.h>
2  #include "stm32f10x.h"
3  #include "gsm.h"
4  #include "gsmBuffer.h"
5
6  volatile char Buffer[512];
7  volatile uint16_t readPos = 0;
8  volatile uint16_t writePos = 0;
9  volatile char CMDCount = 0;
10
11 void writeBuffer(uint16_t received) {
12   Buffer[writePos] = received;
13   if (Buffer[writePos] == '\n') {
14     if (Get_SendFlag())
15     {
16       Clear_SendFlag();
17     }
18     CMDCount++;
19   }
20   writePos++;
21   writePos &= GSM_BUFFMASK;
22 }
23
24 char getCMDCount() {
25   return CMDCount;
```

```
26  }
27
28  void lowerCMDCount() {
29    CMDCount--;
30  }
31
32  char getNextCMD(char* cmd)
33  {
34    uint16_t old = readPos;
35    while(Buffer[readPos]!='\n')
36    {
37      *cmd = Buffer[readPos];
38      Buffer[readPos] = 0;
39      readPos++;
40      readPos &= GSM_BUFFMASK;
41      cmd++;
42      if(readPos == old)
43        return 0;
44    }
45
46    if(Buffer[readPos]=='\n')
47    {
48    Buffer[readPos]=0;
49    readPos++;
50    return 1;
51    }
52    return 0;
53  }
```

Im GSM-Buffer Source File wird der Ringbuffer textitBuffer verwaltet. Somit werden hier die Routinen, die den Buffer verwalten, definiert. Diese Methoden lauten wie folgt:

1. writeBuffer(uint16_t received)
2. char getNextCMD(char *)
3. getCMDCount()
4. lowerCMDCount()

Buffer beschreiben
Methode: *writeBuffer(uint16_t received)*(siehe Codeabschnitt 24)

Diese Methode wird immer vom Empfangsinterrupt (siehe Codeabschnitt 23) aufgerufen. Sie beschreibt den Buffer kontinuierlich mit Werten, erhöht dann den Zeiger *writePos* und maskiert diesen mit der Maske *GSM_BUFFMASK* (siehe 6.7.3.6). Wenn der empfangene Wert ein "\n"[14] [3] ist, so wird der Kommandozähler *CMDCount* (siehe 6.7.3.5) erhöht. Das bedeutet, dass ein vollständiger Befehl angekommen ist (siehe 6.7.2). Die Abfrage *Get_SendFlag* bezieht sich auf das asynchrone senden von SMS (siehe 6.7.3.1). Wenn also eine SMS gesendet wird, so ist das GSM-Modul beschäftigt und unansprechbar, bis es die Bestätigung für einen erfolgreichen Sendevorgang an den µC gesendet hat (siehe 6.7.2.2). Während dieses Sendevorgangs wird das Flag *SendFlag* (siehe 6.7.3.3) gesetzt. Dann wird so lange das GSM-Modul nicht mehr angesprochen bzw. der vielleicht noch unvollständige Buffer nicht mehr ausgelesen, bis die Bestätigung vom GSM-Modul in der Routine 6.7.3.5 empfangen worden ist. Dann wird mittels der Routine *Clear_SendFlag* (siehe 6.7.3.3) das Flag zurückgesetzt. Die Alternative zu diesem asynchronen

[14] Line Feed

Sendevorgang wäre eine 5 Sekunden lange Verzögerung, was bedeuten würde, dass die Regelung des Luftschiffes für diese Zeit keine Berechnungen vornehmen könnte, was fatal wäre.

Buffer auslesen
Methode: *char getNextCMD(char *)* (siehe Codeabschnitt 24)

Der GSM-Buffer wird nur dann ausgelesen, wenn der Kommandozähler (siehe 6.7.3.5) ungleich 0 ist. Dann wird die Funktion *getNextCMD* aufgerufen, welche das nächste Kommando aus dem Ringbuffer ausliest. Die Stelle, welche als nächste ausgelesen wird, gibt der Wert *readPos* an. Dieser Wert wird jedes Mal inkrementiert, wenn ein Wert aus dem Buffer ausgelesen wurde. Die Befehle, welche sich im GSM-Ringbuffer befinden, enden immer mit einem "\n"[15] [3] (siehe 6.7.2). Daher liest die Funktion den Ringbuffer so lange aus, bis zum nächsten "Line Feed". Wenn der Buffer kein "Line Feed" findet, dann wird nach einem kompletten Durchlauf des Buffers abgebrochen und es wird der Wert "0" zurückgegeben, um zu signalisieren, dass der Vorgang gescheitert ist. Wenn hingegen der Vorgang erfolgreich abgeschlossen wurde, so wird der Wert "1" zurückgegeben und das Kommando kann weiterverabeitet werden. Dieses ermittelte Kommando wird während des Bufferdurchlaufs mittels "Call by Reference" in das Array *cmd* (siehe 6.7.3.3) geladen.

Kommandozähler
Methoden: *getCMDCount()* und *lowerCMDCount()* (siehe Codeabschnitt 24)

Der Kommandozähler hat die Funktion, die Auslastung für den GSM-Teil der Firmware, für den µC, zu minimieren. Der Wert des Kommandozählers wird über die Funktion *getCMDCount* am Anfang der Routine *GSM_CheckCMDs* überprüft, diese wird nur durchgeführt, wenn der Wert des Kommandozählers größer "0" ist. Der Kommandozähler wird erhöht, wenn ein neuer vollständiger Befehl über die Routine *writeBuffer* (siehe 6.7.3.5) in den GSM-Buffer geschrieben wird. Dekrementiert wird der Kommandozähler mit der Methode *lowerCMDCount*, aber nur dann, wenn in der Routine *GSM_CheckCMDs* eine Übereinstimmung gefunden wurde (siehe 6.7.3.4).

6.7.3.6 GSM-Buffer Header File

Codeabschnitt 25: GSM-Buffer Header-File

```
1  #ifndef GSMBUFFER_H_
2  #define GSMBUFFER_H_
3
4  #define GSM_BUFFMASK 0b111111111
5
6  void writeBuffer(uint16_t received);
7  char getCMDCount(void);
8  void lowerCMDCount();
9  char getNextCMD(char* cmd);
10
11 #endif /* GSMBUFFER_H_ */
```

Im GSM-Buffer Header-File werden die Methoden des GSM-Buffer Source-Files (siehe Codeabschnitt 24) deklariert. Zusätzlich wird die *GSM_BUFFMASK* definiert, welche für den GSM-Ringbuffer in der Methode *writeBuffer* benötigt wird, um den Lese- und Schreibezeiger zu maskieren.

[15] Line Feed

6.8 Bluetooth Kommunikation

6.8.1 Allgemeines

Da ein Debug-Mechanismus über die GSM-Kommunikation (siehe 6.7) aufgrund der großen Anzahl an zu versendenden SMS zum einen relativ teuer, zum anderen aber auch viel zu langsam wäre, um logische und vertretbare Schlussfolgerungen zu ziehen, wird zusammen mit dem *Bluetooth_DebugWindow* (siehe 7.2) und der Bluetooth-Firmware eine Debugmöglichkeit geschaffen. Dazu wird ein Bluetooth-Modul mit dem µC verbunden. Dieses bietet somit über die serielle Bluetooth-Schnittstelle eine Verbindung zu einem Laptop bzw. PC, auf dem das *Bluetooth_DebugWindow* (siehe 7.2) in Betrieb ist. Dies eröffnet somit die Debugmöglichkeit. Diese Möglichkeit ist vor allem bei der Einstellung der BLDC-Motoren und bei der Testung der Regelung von Vorteil.

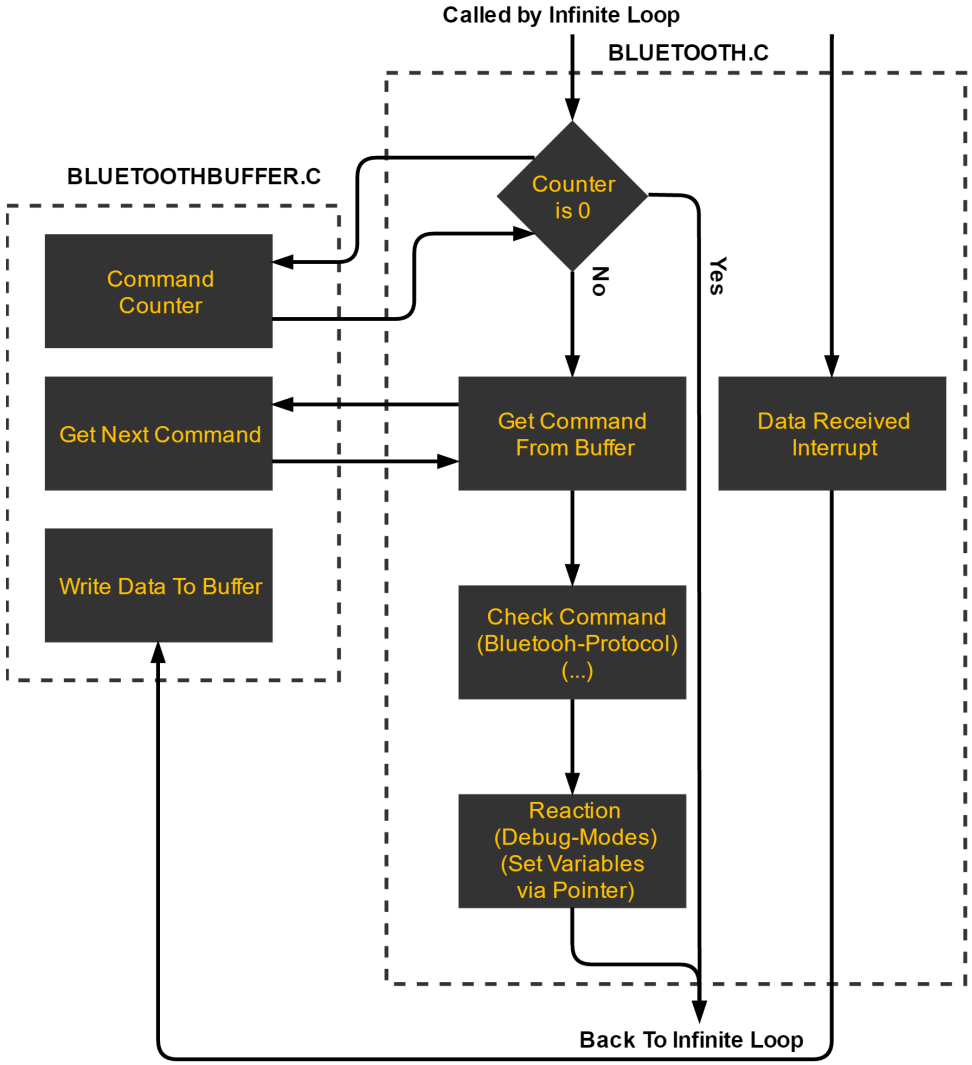


Abbildung 37: Bluetooth-Firmware Programmablauf

Wie in Abbildung 37 zu sehen ist, wurde das Konzept der Bluetooth-Kommunikation der GSM-Kommunikation angelehnt. Dabei wurde wie bei der GSM-Firmware (siehe 6.7) ein Bluetooth-ringbuffer erstellt und eine Routine, welche den Buffer kontinuierlich auf Befehle überprüft. Um

mehrere Debugmöglichkeiten zu haben, wurde ein kleines Protokoll erstellt.
Wenn das Bluetooth-Modul serielle Daten empfängt, werden diese an den µC über UART weitergeleitet. Daraufhin wird der *USART3_IRQHandler* (siehe Codeabschnitt 26) Interrupt ausgelöst. In diesem Interrupt werden die Daten in den Bluetooth-Buffer geschrieben. Am Ende jedes Befehls wird vom *Bluetooth_DebugWindow* (siehe 7.2) ein "\n"[16] [3] gesendet. Somit können die Befehle separiert und gezählt werden, was den Vorteil hat, dass nur dann Befehle verarbeitet werden, wenn der Bluetooth-Kommandozähler vollständige Befehle gezählt hat. Dadurch wird nicht sinnlos Ressourcen verschwendet.

6.8.1.1 Debugmöglichkeiten

Die verfügbaren Bluetooth-Debugmöglichkeiten können jeweils über das *Bluetooth_DebugWindow* (siehe 7.2) aktiviert und deaktivert werden.

1. Debuggen einer Kategorie
    - Es können Werte bzw. Variablen vordefiniert und im Bluetooth Source-File in der Routine *Write_DebugInfo* hinzugefügt werden. Diese Variablen werden dann regelmäßig nach einer bestimmten Zeitspanne an das *Bluetooth_DebugWindow* (siehe 7.2) gesendet. Die Zeitspanne kann im *Bluetooth_DebugWindow* eingestellt werden. Folgende Kategorien stehen zum Debuggen zur Verfügung:
      GSM, GPS, MPU, SERVO, BLDC und CONTROLLER
2. Debuggen mittels Zeiger
    - Debugvariablen können mit der Routine *Add_DebugVariable* hinzugefügt werden. Dabei wird der Zeiger, der auf die gewünschte Variable zeigt, mitgegeben. Alle hinzugefügten Variablen werden im selben Zeitabstand wie "Debuggen einer Kategorie" regelmäßig versendet. Da Zeiger verwendet werden, können die Variablen auch vom *Bluetooth_DebugWindow* (siehe 7.2) aus, wieder gesetzt werden. Daher ist es möglich, Variablen bzw. Koeffizienten in Echtzeit zu verändern. So kann zum Beispiel die Regelung sehr gut während des Betriebs analysiert werden.

6.8.1.2 Protokoll

In der Tabelle 14 werden die Eingabemöglichkeiten des *Bluetooth_DebugWindow* (siehe 7.2) erläutert. Das sind zugleich auch die Eingänge der Bluetooth-Firmware, welche von der Firmware dementsprechend überprüft werden müssen.

6.8.2 C-Code

6.8.2.1 Bluetooth-Buffer Source-File

Codeabschnitt 26: Bluetooth-Buffer Source-File

```
1 #include <stdint.h>
2 #include "stm32f10x.h"
3 #include "bluetoothBuffer.h"
4 #include "clock.h"
```

[16] Line Feed

```
5  #include "timebase.h"
6
7  volatile char blue_Buffer[512];
8  uint16_t blue_readPos = 0;
9  uint16_t blue_writePos = 0;
10 char blue_CMDCount = 0;
11
12
13 void blue_writeBuffer(uint16_t received)
14 {
15   blue_Buffer[blue_writePos] = received;
16   if(blue_Buffer[blue_writePos]=='\n')
17     blue_CMDCount++;
18   blue_writePos++;
19   blue_writePos &= Blue_UART_BUFFMASK;
20 }
21
22 void blue_getNextCmd(char* cmd)
23 {
24     while(blue_Buffer[blue_readPos]!='\n')
25     {
26       *cmd = blue_Buffer[blue_readPos];
27       blue_Buffer[blue_readPos] = 0;
28       blue_readPos++;
29       blue_readPos &= Blue_UART_BUFFMASK;
30       cmd++;
31     }
32     blue_Buffer[blue_readPos] = 0;
33     blue_readPos++;
34 }
35
36 char get_blue_CMDCount(void)
37 {
38   return blue_CMDCount;
39 }
40 void lower_blue_CMDCount(char* array)
41 {
42   blue_CMDCount--;
43   memsetpersonal(array, '\0', 20);
44 }
```

Bluetooth-Buffer Source-File Routinen folgende Routinen definiert:

1. blue_writeBuffer(uint16_t received)
2. blue_getNextCmd(char * cmd)
3. char get_blue_CMDCount()
4. lower_blue_CMDCount(char * array)

Bluetooth-Buffer schreiben
Methode: *blue_writeBuffer(uint16_t received)* (siehe Codeabschnitt 26)

Diese Methode wird in der Interruptroutine *USART3_IRQHandler* (siehe Codeabschnitt 26) aufgerufen und schreibt die empfangenen Daten in den Bluetooth-Buffer. Wenn der geschriebene Wert einem "Line Feed" entspricht, so wird der Bluetooth-Kommandozähler inkrementiert. Der Zeiger *blue_writePos* gibt an, an welcher Stelle im Buffer als nächstens geschrieben wird. Da es sich beim Buffer um einen Ringbuffer handelt, wird jener bei jedem Schreibvorgang am Schluss mit der Variable *Blue_UART_BUFFMASK* (siehe Codeabschnitt 27) maskiert.

Bluetooth-Buffer lesen
Methode: *blue_ getNextCmd(char * cmd)* (siehe Codeabschnitt 26)

Das Auslesen des Ringbuffers geschieht über die Funktion *blue_ getNextCmd*. In dieser wird bis zu einem "Line Feed" gelesen bei jedem einzelnen Schritt der Zeiger *blue_ readPos* inkrementiert und zusätzlich mit dem Wert *Blue_ UART_ BUFFMASKh* (siehe Codeabschnitt 27) maskiert. Die gelesenen Werte in jedem Schritt werden über "Call by Reference" in das *blue_ cmd* Array geladen (siehe Codeabschnitt 28).

Bluetooth-Kommandozähler
Methoden: *char get_ blue_ CMDCount()* und *lower_ blue_ CMDCount(char * array)* (siehe Codeabschnitt 26)

Die Methode *get_ blue_ CMDCount* gibt schlicht den Wert des Bluetooth-Kommandozählers zurück. Dieser wird in der *blue_ CheckCmds* Routine benötigt, um zu entscheiden, ob Befehle überprüft werden müssen oder nicht. Jedes Mal, wenn erfolgreich, muss der Bluetooth-Kommandozähler mit der Methode *lower_ blue_ CMDCount* entsprechend dekrementiert werden.

6.8.2.2 Bluetooth-Buffer Header-File

Codeabschnitt 27: Bluetooth-Buffer Header-File

```
1  #ifndef bluetoothBuffer_H_
2  #define bluetoothBuffer_H_
3
4  #define Blue_UART_BUFFMASK 0b111111111
5
6  void blue_writeBuffer(uint16_t received);
7  void blue_getNextCmd(char* cmd);
8  char get_blue_CMDCount(void);
9  void lower_blue_CMDCount(char* array);
10
11 #endif
```

Im Bluetooth-Buffer Header-File werden die Routinen des Bluetooth-Buffer Source-Files deklariert. Zusätzlich wird dort die Buffermaske definiert, welche im Bluetooth-Buffer Source-File benötigt wird (siehe 6.8.2.1).

6.8.2.3 Bluetooth Source-File Initialisierungen

Codeabschnitt 28: Bluetooth Source-File Initialisierungen

```
1  #include <stddef.h>
2  #include <string.h>
3  #include <stdio.h>
4
5  #include "stm32f10x_usart.h"
6  #include "clock.h"
7  #include "timebase.h"
8  #include "bluetoothBuffer.h"
9  #include "bluetooth.h"
10 #include "tools.h"
11
```

```
12 char blue_cmd[20];
13 volatile char Debug_gsmFlag = 0;
14 volatile char Debug_gpsFlag = 0;
15 volatile char Debug_mpuFlag = 0;
16 volatile char Debug_servoFlag = 0;
17 volatile char Debug_bldcFlag = 0;
18 volatile char Debug_controllerFlag = 0;
19 volatile char Debug_Data = 0;
20
21 DebugInfo DebugInfos[64];
22 uint8_t debugInfoCounter = 0;
23 char array[1000];
24 uint16_t wx = 0;
25
26 void Init_Bluetooth() {
27
28   //Init USART
29   RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOB | RCC_APB2Periph_AFIO, ENABLE);
30   RCC_APB1PeriphClockCmd(RCC_APB1Periph_USART3, ENABLE);
31
32   USART_InitTypeDef USART_InitStructure;
33
34   USART_InitStructure.USART_BaudRate = 57600;
35   USART_InitStructure.USART_WordLength = USART_WordLength_8b;
36   USART_InitStructure.USART_StopBits = USART_StopBits_1;
37   USART_InitStructure.USART_Parity = USART_Parity_No;
38   USART_InitStructure.USART_HardwareFlowControl = ↙
       ↳ USART_HardwareFlowControl_None;
39   USART_InitStructure.USART_Mode = USART_Mode_Rx | USART_Mode_Tx;
40
41   USART_Init(USART3, &USART_InitStructure);
42
43   // Enable the USART3
44   USART_Cmd(USART3, ENABLE);
45   USART_ITConfig(USART3, USART_IT_RXNE, ENABLE);
46
47   GPIO_InitTypeDef gpioConfig;
48
49   //PB10 = USART3.TX = Alternative Function Output
50   gpioConfig.GPIO_Mode = GPIO_Mode_AF_PP;
51   gpioConfig.GPIO_Pin = GPIO_Pin_10;
52   gpioConfig.GPIO_Speed = GPIO_Speed_2MHz;
53   GPIO_Init(GPIOB, &gpioConfig);
54
55   //PB11 = USART3.RX = Input
56   gpioConfig.GPIO_Mode = GPIO_Mode_IN_FLOATING;
57   gpioConfig.GPIO_Pin = GPIO_Pin_11;
58   GPIO_Init(GPIOB, &gpioConfig);
59
60   NVIC_InitTypeDef NVIC_InitStructure;
61
62   // Configure the NVIC Preemption Priority Bits
63   NVIC_PriorityGroupConfig(NVIC_PriorityGroup_0);
64
65   // Enable the USART3 Interrupt
66   NVIC_InitStructure.NVIC_IRQChannel = USART3_IRQn;
67   NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;
68   NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;
69   NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
70   NVIC_Init(&NVIC_InitStructure);
71
72   RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOC, ENABLE);
```

```
73    GPIO_InitTypeDef GPIO_InitStructure;
74    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_13;
75    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
76    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_10MHz;
77    GPIO_Init(GPIOC, &GPIO_InitStructure);
78    GPIO_ResetBits(GPIOC, GPIO_Pin_13);
79    delay_ms(200);
80    GPIO_SetBits(GPIOC, GPIO_Pin_13);
81  //  Bluetooth-Module Configuration
82  //  delay_ms(200);
83  //  Bluetooth_SendData("AT");
84  //  delay_ms(1000);
85  //  Bluetooth_SendData("AT+NAMESCAVenger");
86  //  delay_ms(1000);
87  //  Bluetooth_SendData("AT+BAUD7");
88  //  delay_ms(1000);
89  }
```

Im Bluetooth Source-File Initialisierungen (siehe Codeabschnitt 28) wird Folgendes definiert:

1. Die USART3 wird mit beschrieben Eigenschaften definiert:
   - Die Baudrate beträgt 57600.
   - Einer Wortlänge von 8 bit.
   - Einem Stopbit.
   - Keiner Datenflusssteuerung.
   - Der Mode ist "USART empfangen und senden".
   - Die verwendeten Pins der USART3 sind:
     - TX –> Pin PB10
     - RX –> Pin PB11
2. char blue_cmd
   - In dieses Array werden die Werte mittels "Call by Reference" geladen, welche bei der Funktion *blue_ getNextCMD* ermittelt wurden (siehe 6.8.2.1).
3. zahlreiche Debug-Flags
   - Alle definierten Debug-Flags werden gesetzt, wenn eine bestimmte Kategorie gedebugt wird. Das setzen und zurücksetzen dieser Flags erfolgt in der Routine *blue_ CheckCmds* (siehe 6.8.2.4).
4. DebugInfo DebugInfos
   - Dieses Array besteht aus einer Struktur, welche im Bluetooth Header-File definiert wurde (siehe Codeabschnitt 31). Ein solches Element beinhaltet Informationen über den Titel der Debug-Variable, den Zeiger auf das zu debuggende Element. (siehe Codeabschnitt 31).
5. uint8_t debugInfoCounter
   - Diese Variable zählt schlicht, wieviele Einträge in *DebugInfos* in Verwendung sind. Wenn eine Variable in *Add_ DebugVariable* (siehe 6.8.2.4) hinzugefügt wird, so wird der Wert erhöht. Dies hat den Vorteil das der µC nicht sinnlos ausgelastet wird.
6. char array und uint16_t wx

- Diese beiden Variablen haben mehr oder weniger nur eine temporäre Rolle. (siehe 6.8.2.4)

Der auskommentierte Codeteil am Ende des Codeabschnitt 28 wird benötigt, wenn Änderungen am Namen oder an der Baudrate des Bluetooth-Moduls vorgenommen werden sollen. Da dies im Grunde nur einmal ausgeführt werden muss, ist dieser Codeabschnitt nicht standardmäßig im Programm enthalten.

6.8.2.4 Bluetooth Source-File Routinen

Codeabschnitt 29: Bluetooth Source-File Routinen

```
1  void USART3_SendString(char *string) {
2    uint8_t tx_index = 0;
3    while (tx_index < (strlen(string))) {
4      USART_SendData(USART3, string[tx_index++]);
5
6      while (!USART_GetFlagStatus(USART3, USART_FLAG_TXE))
7        ;
8    }
9  }
10 void USART3_IRQHandler(void) {
11   if (USART_GetITStatus(USART3, USART_IT_RXNE) != RESET) // Received ↙
       ↳ characters modify string
12       {
13     blue_writeBuffer(USART_ReceiveData(USART3));
14
15   }
16 }
17
18 void Add_DebugVariable(char* title, void * pointer) {
19   strcpy((&DebugInfos[debugInfoCounter])->string, title);
20   (&DebugInfos[debugInfoCounter])->pointer = pointer;
21   debugInfoCounter++;
22 }
23
24 void Write_DebugInfo() {
25   if (Get_DebugTick() == 1) {
26     if (Debug_gsmFlag == 1) {
27       USART3_SendString("Insert GSM Debug!*a;");
28     }
29     if (Debug_gpsFlag == 1) {
30       USART3_SendString("Insert GPS Debug!*b;");
31     }
32     if (Debug_mpuFlag == 1) {
33       USART3_SendString("Insert MPU Debug!*c;");
34     }
35     if (Debug_servoFlag == 1) {
36       USART3_SendString("Insert Servo Debug!*d;");
37     }
38     if (Debug_bldcFlag == 1) {
39       USART3_SendString("Insert BLDC Debug!*e;");
40     }
41     if (Debug_controllerFlag == 1) {
42       USART3_SendString("Insert Controller Debug!*f;");
43     }
44
45     if (Debug_Data == 1) {
46       if (Debug_Data)
```

```
47          USART3_SendString("?;");
48        if (debugInfoCounter != 0) {
49          uint8_t i;
50          for (i = 0; i <= debugInfoCounter; i++) {
51            sprintf(&array[wx], (&DebugInfos[i])->string,
52                *((uint16_t *) (&DebugInfos[i])->pointer) + '\0');
53
54            while (array[wx] != '\0')
55              wx++;
56
57              array[wx] = ';';
58
59          }
60        }
61        while (array[wx] != '\0')
62          wx++;
63
64        USART3_SendString(&array[0]);
65        memsetpersonal(array, 0, 1000);
66        wx = 0;
67      }
68      Reset_DebugFlag();
69    }
70  }
71
72  void blue_CheckCmds() {
73    if (get_blue_CMDCount() != 0) {
74      blue_getNextCmd(&blue_cmd[0]);
75      if (blue_cmd[0] == 't') {
76        int i = 0;
77        i = atoi(&blue_cmd[1]);
78        SetDebugCounterMax(i);
79        USART3_SendString("Debug Time updated.*g;");
80        lower_blue_CMDCount(blue_cmd);
81        return;
82      }
83      if (blue_cmd[0] == 's') {
84        char array[2] = { blue_cmd[1], blue_cmd[2] };
85        uint8_t index = atoi(array);
86        int16_t value = atoi(&blue_cmd[4]);
87        *((int16_t *) ((&DebugInfos[index])->pointer)) = value;
88        USART3_SendString("Value changed*g;");
89        lower_blue_CMDCount(blue_cmd);
90        return;
91      }
92      if (blue_cmd[0] != '!') {
93        switch (blue_cmd[2]) {
94        case 'm': //GSM
95          Debug_gsmFlag = 1;
96          lower_blue_CMDCount(blue_cmd);
97          USART3_SendString("GSM-Debug=1*a;");
98          break;
99        case 's': //GPS
100         Debug_gpsFlag = 1;
101         lower_blue_CMDCount(blue_cmd);
102         USART3_SendString("GPS-Debug=1*b;");
103         break;
104       case 'u': //MPU
105         Debug_mpuFlag = 1;
106         lower_blue_CMDCount(blue_cmd);
107         USART3_SendString("MPU-Debug=1*c;");
108         break;
```

```
109         case 'r': //SERVO
110           Debug_servoFlag = 1;
111           lower_blue_CMDCount(blue_cmd);
112           USART3_SendString("Servo-Debug=1*d;");
113           break;
114         case 'd': //BLDC
115           Debug_bldcFlag = 1;
116           lower_blue_CMDCount(blue_cmd);
117           USART3_SendString("BLDC-Debug=1*e;");
118           break;
119         case 'n': //CONTROLLER
120           Debug_controllerFlag = 1;
121           lower_blue_CMDCount(blue_cmd);
122           USART3_SendString("Controller-Debug=1*f;");
123           break;
124         case 'b': //debugData
125           Debug_Data = 1;
126           lower_blue_CMDCount(blue_cmd);
127           USART3_SendString("Debug-Data=1*g;");
128           break;
129         }
130       }
131       if (blue_cmd[0] == '!') {
132         switch (blue_cmd[3]) {
133         case 'm': //GSM
134           Debug_gsmFlag = 0;
135           lower_blue_CMDCount(blue_cmd);
136           USART3_SendString("GSM-Debug=0*a;");
137           break;
138         case 's': //GPS
139           Debug_gpsFlag = 0;
140           lower_blue_CMDCount(blue_cmd);
141           USART3_SendString("GPS-Debug=0*b;");
142           break;
143         case 'u': //MPU
144           Debug_mpuFlag = 0;
145           lower_blue_CMDCount(blue_cmd);
146           USART3_SendString("MPU-Debug=0*c;");
147           break;
148         case 'r': //SERVO
149           Debug_servoFlag = 0;
150           lower_blue_CMDCount(blue_cmd);
151           USART3_SendString("Servo-Debug=0*d;");
152           break;
153         case 'd': //BLDC
154           Debug_bldcFlag = 0;
155           lower_blue_CMDCount(blue_cmd);
156           USART3_SendString("BLDC-Debug=0*e;");
157           break;
158         case 'n': //CONTROLLER
159           Debug_controllerFlag = 0;
160           lower_blue_CMDCount(blue_cmd);
161           USART3_SendString("Controller-Debug=0*f;");
162           break;
163         case 'b': //debugData
164           Debug_Data = 0;
165           lower_blue_CMDCount(blue_cmd);
166           USART3_SendString("Debug-Data=0*g;");
167           break;
168         }
169       }
170     }
```

```
171
172 }
```

Bluetooth Source-File Routinen folgende Routinen definiert:

1. USART3_SendString(char * string)
2. USART3_IRQHandler()
3. Add_DebugVariable(char * title, void * pointer)
4. Write_DebugInfo()
5. blue_CheckCmds()

USART3-Senderoutinen
Methode: *(char * string)*

USART3_IRQHandler()

Diese Methode sendet das mitgereichte Array aus Bytes über die USART3 an das Bluetooth-Modul. Das Modul versendet diese Daten dann direkt weiter an den PC. Nach jedem gesendeten Byte wird gewartet, bis das *USART_FlAG_TXE* gesetzt wurde, da es ansonsten zu einer unvollständigen Übertragung kommen könnte und infolgedessen auch zu Interferenzen.

USART3-Interrupt
Methode: *USART3_IRQHandler()*

Wenn über Bluetooth Daten empfangen werden, wird das Interrupt *USART3_IRQHandler* aufgerufen, welcher die Daten an den Bluetooth-Buffer über die Methode *blue_writeBuffer* (siehe 6.8.2.1) weiterleitet.

Zeiger zum Debuggen hinzufügen
Methode: Add_DebugVariable(char * title, void * pointer)

Wie in 6.8.1.1 schon beschrieben, bietet die Bluetooth-Firmware die Möglichkeit, "mittels Zeiger zu debuggen". Alle diese Zeiger werden mit der Methode *Add_DebugVariable* dem Array *DebugInfos* an der Stelle von *debugInfoCounter*, welcher auf den nächsten freien Platz im Array zeigt, hinzugefügt. In dieser Methode werden die Variablen der Struktur mithilfe der mitgegebenen Parameter gesetzt. Zum Schluss wird der *debugInfoCounter* inkrementiert, da wieder ein Platz belegt wurde. Wichtig ist, dass der Parameter *title* mit einem "$" gefolgt vom Index beginnt. Das Dollarzeichen markiert für das *Bluetooth_DebugWindow* (siehe 7.2), dass eine neue Debuginformation beginnt. Der Index danach ist vonnöten damit auf beiden Seiten klar ist, welche Debuginformation bei einer Veränderung betroffen ist.

Codeabschnitt 30: Add_DebugVariable Beispiel

```
1 Add_DebugVariable("$0Angle:%d", &angle);
2 Add_DebugVariable("$1Koeffizient:%d", &koeff);
3 Add_DebugVariable("$3PAnteil:%d", &p);
4 Add_DebugVariable("$2IAnteil:%d", &i);
```

Debuginformationen senden
Methode: *Write_DebugInfo()*

Diese Methode wird kontinuierlich in der Endlosschleife des Hauptprogrammes aufgerufen. Sie hat die Funktion, die Informationen aller Debugmöglichkeiten (siehe 6.8.1.1) an die PC-Anwendung

Bluetooth_DebugWindow (siehe 7.2) zu senden. Dies geschieht immer dann, wenn die Funktion *Get_DebugTick* den Wert 1 zurückgibt. Wenn das der Fall ist, werden zuerst die Flags der Kategorien überprüft und dementsprechend die Zeichenketten gesendet. Danach wird überprüft, ob der "Zeigerdebugmodus" seitens des PC-Anwenders aktiviert worden ist. Ist dies der Fall, dann wird sofort ein "?;" über Bluetooth gesendet (siehe 6.8.1.1). Da das *Bluetooth_DebugWindow* (siehe 7.2) mit einem Semikolon separiert, fällt dieses Zeichen weg. Nun bleibt noch das Fragezeichen über, welches der PC-Andwenung mitteilt, dass nach diesem Kommando die Debuginformationen gesendet werden. Mittels einer Schleife wird nun das Array *DebugInfos* durchgegangen. Für jedes einzelne Element werden mit der Funktion "sprintf" die relevanten Informationen in eine Zeichenkette umgewandelt. Diese Zeichenkette wird direkt in dem "array" gespeichert. Danach wird der Zeiger *wx* solange erhöht, wie die vorhergehende Zeichenkette lang war. Im nächsten Punkt wird schlicht ein Semikolon am Ende der Zeichenkette angefügt. Zum Schluss werden die Daten mit der Methode *USART3_SendString* gesendet und das Array, der Zeiger und das Flag *Debug_Data_Now* zurückgesetzt.

Bluetooth Kommandoüberprüfung
Methode: *blue_CheckCmds()*

Wie auch die Methode *Write_DebugInfo* wird auch die Methode *blue_CheckCmds* in der Endlosschleife des Hauptprogramms aufgerufen. Sie dient dazu, die empfangenen Daten, die vom PC (*Bluetooth_DebugWindow* siehe 7.2) gesendet werden, zu interpretieren und die dementsprechenden Aktionen einzuleiten. Zunächst wird der Kommandozähler (siehe 6.8.2.1) überprüft. Wenn Kommandos im Ringbuffer stehen, wird das nächste Kommando mit der Methode blue_getNextCmd (siehe 6.8.2.1) ausgelesen. Dies wird nun auf alle möglichen Übereinstimmungen geprüft. Dabei wird das Kommando hautsächlich mit den Einträgen der (siehe Tabelle 14) verglichen. Da die Zeitspanne des Debuggens über die PC-Anwendung veränderbar ist, muss dieser Befehl, welcher immer mit einem "t" beginnt, zusätzlich überprüft werden. Nach dem empfangenen "t" werden noch Ziffern empfangen, die die Debug-Zeitspanne in "ms" angeben. Wie in 6.8.1.1 erklärt, gibt es die Möglicht des Debuggens mittels Zeiger, in dem das Setzen der Variablen mittels der Zeiger möglich ist. Deshalb wird in der *blue_CheckCmds* das "s" (set) überprüft. In dieser Überprüfung wird zuallererst der Index der Variable, die gedebugt wird (siehe 6.8.2.4), in ein verarbeitbares Zahlenformat mit der Methode "atoi" umgewandelt. Im nächsten Schritt wird zuerst der Zeiger der gedebugten Variable aus dem *DebugInfos* herausgelesen. Dauraffolgend wird mittels "Typecast" der Zeiger in einen uint16_t * umgewandelt. Nun wird der Inhalt dieses Zeigers mit dem neuen Wert beschrieben.

6.8.2.5 Bluetooth Header-File

Codeabschnitt 31: Bluetooth Header-File

```
1  #ifndef bluetooth_H_
2  #define bluetooth_H_
3
4  void Init_Bluetooth(void);
5  void USART3_IRQHandler(void);
6  void blue_CheckCmds();
7  void Write_DebugInfo();
8  void Add_DebugVariable(char* title, void * pointer);
9
10 typedef struct
11 {
12   char string[30];
13   void * pointer;
14 } DebugInfo;
```

```
15
16 #endif
```

Im Bluetooth Header-File werden die Methoden des Bluetooth Source-Files (siehe Codeabschnitt 28 und Codeabschnitt 29) deklariert. Zusätzlich werden die Enumeration *Category* und die Struktur *DebugInfo*, definiert. Diese Struktur enthalt zum einen den Titel der Debugvariable und zum anderen einen void * Zeiger auf die Variable. Dies kann gemacht werden, da alle Zeiger, egal welches Zahlenformat, dieselbe Länge haben. Dieser kann im Nachhinein, mittels "Typecast" in einen Zeiger, realen Zeigerformates, umgewandelt werden (siehe 6.8.2.4).

6.9 Regelung

6.9.1 Allgemeines

Im Unterabschnitt SSimulation"des Abschnitts SSoftwareäb Seite 107 wird kurz der Entwurf des benötigten Regelungssystems erläutert, der sich nach dem Konfigurationsvorschlag 3 richtet. Dessen Umsetzung soll hier nun erläutert werden.

Der Regler hat folgende Schritte durchzuführen:

1. *Vorverarbeitung der Eingangsdaten*: Die Führgrößen und Regelgrößen werden zu den im oben erwähnten Unterabschnitt spezifizierten Stellgrößen zusammengeführt, umgewandelt.
2. *Ermittlung der Stellgrößen*: Mittels vier P-Reglern werden die vier Regelabweichungen *Abstand*, *Höhendifferenz*, *Kursabweichung* sowie Nickwinkel in ihre entsprechenden Stellgrößen umgesetzt
3. *Nachverarbeitung der Ausgangsdaten* Die erhaltenen Stellgrößen werden derart umkonvertiert, dass sie auf die Aktoren beaufschlagt werden können.

6.9.1.1 Verarbeitung der Eingangsdaten

Zur Ermittlung der Regelabweichungen werden nachfolgende Methoden implementiert:

Lineare Interpolaton
Methode: *int16_t lin_int(int16_t* table_x, int16_t* table_y, int16_t arg)*
Bestimmt zwei nebeneinander liegende Einträge der Tabelle *table_x*, zwischen denen *arg* liegt. Ermittelt anschließend die die beiden Einträge der Tabelle *table_y* mit derselben Indizierung. Errechnet aus diesen vier Tabelleneinträgen eine lineare Funktion. Gibt den interpolierten Wert mittels Einsetzen von *arg* in diese Funktion aus. Die Interpolation entfällt, wenn *arg* mit einem Eintrag von *table_x* zusammenfällt. Dann wird der korrespondierende Eintrag von *table_y* ausgegeben. Liegt *arg* außerhalb des Wertebereichs von *table_x*, wird 0 zurückgeliefert.

Berechnung der Quadratwurzel
Methode: *uint8 sqrt(int16_t arg)*
Approximiert die Quadratwurzel von *arg* mittels einer linear interpolierten Wertetabelle mit einer akzeptablen Genauigkeit.

Berechnung des Abstands
Methode: *int16_t dr(int16_t current_pos, int16_t next_pos)*
Ermittelt den Abstandsvektor aus *current_pos* sowie *next_pos*. Quadriert die Komponenten dieses Vektors und speichert sie in *int16_t* -Variablen ab. Addiert die beiden Variablen und gibt deren Quadratwurzel aus.

Berechnung von arctan2
Methode: *int16_t tan2(int16_t x, int16_t y)*
Bestimmt den Quadranten, in den der Vektor zeigt. Ist x gleich null sowie y ungleich null, liegt der Vektor in der y-Achse. Ist x ungleich null sowie y gleich null, liegt der Vektor in der x-Achse. In obigen Fällen kann der Winkel sofort bestimmt werden. Ist x gleich null sowie y gleich null, kann kein Winkel festgelegt werden. In allen anderen Fällen wird der Winkel mittels *tan* berechnet und abhängig vom Quadranten wird 90°, 180° bzw. 270° hinzuaddiert. Die Implementierung von *tan* erfolgt dabei innerhalb dieser Methode mittels einer linear interpolierten Wertetabelle.

Aufintegration der Gyrodaten
Methode: *int16_t* i_gyro(int16_t* rot_last, int16_t* omega)*

Errechnet die momentane Auslenkung des Luftschiffs, indem eine durch *omega* spezifizierte infinitesimale Drehung von *rot_last* durchgeführt wird.

Aufintegration der Gyrodaten
Methode: *int16_t* to_eulerian(int16_t* rot_cur, int16_t* rot_last)*

Konvertiert die erhaltene Rotationsmatrix in eulersche Winkel um.[?]

Behebung der Integrationsfehler
Methode: *int16_t* comp_filter(int16_t phi_gyr, int16_t phi_acc)*

Unterdrückt den fehlerhaften durch die Integrationsdrift verursachten niederfrequenten Anteil der aufintegrierten Gyrodaten mittels eines digitalen Tiefpasses, realisiert durch ein IIR-Filter erster Ordnung. Addiert diese gefilterten eulerschen Winkel zu den Daten des Beschleunigungssensors, dessen hochfrequenter fehlerhafter Anteil zuvor mittels eines zum digitalen Tiefpass komplementären digitalen Hochpasses unterdrückt wird.

Berechnung der Kursabweichung
Methode: *int16_t* delta_phi(int16_t phi, int16_t phi_blimp)*
Berechnet die Differenz der Winkel *phi* sowwie *phi_blimp*. Auftretende Winkelsprünge werden behoben.

Gewichtungsfunktion
Methode: *int16_t weight_f(int16_t arg, int16_t mue, int16_t sigma)*

Realisiert eine Gewichtungsfunktion, wie auf Seite **??** in Gleichung 7.13 dargelegt wird. Mit dem Parameter *mue* wird die Lage des Wendepunktes, mit *sigma* die Breite der Gewichtungsfunktion festgelegt. Ein negatives Vorzeichen von *mue* bedeutet eine Spiegelung der Gewichtungsfunktion um den Wendepunkt.

Toleranzen der Regelabweichung: Abstand
Methode: *int16_t r_tol(int16_t delta_r)*

Unterschreitet der Abstand einen spezifizierten Wert, wird er auf 0 gesetzt.

Toleranzen der Regelabweichung: Höhe
Methode: *int16_t h_tol(int16_t delta_h)*

Unterschreitet die Höhendifferenz einen spezifizierten Wert, wird sie auf 0 gesetzt.

Toleranzen der Regelabweichung: Kursabweichung
Methode: *int16_t phi_tol(int16_t delta_phi, int16_t delta_r)*

Unterschreitet der Abstand zum Ziel einen bestimmten Minimalwert, wird zwischen einer festgelegten Konstante und dem Abstand zum Ziel *delta_r* eine Ganzzahldivision durchgeführt. Führt das Ergebnis einer Gewichtungsfunktion zu. Gibt das Ergebnis der Multiplikation der Gewichtungsfunktion mit der Kursabweichung *delta_phi* zurück.

6.9.1.2 Ermittlung der Stellgrößen

Die Stellgrößen werden mittels P-Regler aus den vorverarbeiteten Regelabweichungen ermittelt, es werden keine andere Regler verwendet. Die Hinzunahme eines I-Anteils erbringt nur wenig Nutzen, da es aufgrund der gegebenen Ungenauigkeiten nicht möglich ist, die Regelabweichung unter einen bestimmten Schwellenwert zu bringen. Die Hinzunahme eines D-Anteils könnte zwar eine Stabilitätsverbesserung erbringen, jedoch den Blimp aufgrund von nicht weggefilterten Rauscheinflüssen zum Absturz bringen.

P-Regler
Methode: *int16_t p_controller(int16_t input)*
Multipliziert die Stellgröße mit einem konstanten Wert. Hat einen konstanten Phasengang von 0°.

6.9.1.3 Nachverarbeitung der Stellgrößen

Einige Regelabweichungen des Systems, wie etwa die Höhendifferenz, nehmen die meiste Betriebszeit über Werte von großem Betrag ein. Dies resultiert in großen Stellgrößen, welche ohne Nachverarbeitung nicht die erwünschte Wirkung hätten.

Stellgrößenbeschränkung
Methode: *int16_t saturation(int16_t F_x, int16_t F_y)*
Berechnet den Betrag des Kraftvektors, dessen Komponenten F_x, F_y darstellen. Übersteigt dieser einen spezifizierten Maximalwert, wird der Vektor den Erläuterungen auf Seite 117 folgend skaliert.

7 Software

7.1 Simulation

7.1.1 Allgemeines

Die Komplexität des Projekts macht es notwendig, das Regelungssystem schon vor dessen eigentlichen Einsatz zu verifizieren. Die Verifikation erfolgt dabei zunächst über eine mittels Simulink realisierte Simulation. Ihr Inhalt sowie ihr Zustandekommen soll in diesem Abschnitt erläutert werden.

7.1.2 Arten der Realisierung

Im Zuge dieser Arbeit wird die Simulation des physikalischen Modell auf dreierlei Arten durchgeführt, welche hier in chronologischer Reihenfolge aufgelistet sind.

- *Beschreibung mittels Matlab-Code*: Die im Abschnitt "Mechanischer Aufbau" dargelegten Gleichungen können in einen ausführbaren Matlab-Code übersetzt werden. Da der Matlab-Syntax mathematisch orientiert ist, können auch komplexe statische Zusammenhänge ohne größeren Aufwand formuliert werden. Problematischer ist die Formulierung von dynamischen Zusammenhängen, wofür ein System von Differentialgleichungen aufgestellt werden muss, welches spätestens beim Versuch, einen Regelkreis zu beschreiben, undurchschaulich wird.
- *Beschreibung mittels Simulink-Blöcken*: Mittels Simulink-Blöcken können Regelkreise optimal formuliert werden, weil diese weitestgehend dessen Zeichnung auf Papier entsprechen. Allerdings stellen sie eine abstrakte Formulierung von algebraischen Gleichungen dar, weshalb sich der Realisierungsaufwand sowie die Fehleranfälligkeit für größere Modelle stark erhöht.
- *Beschreibung mittels Simulink-Blöcken und Matlab-Code*: Simulink stellt spezielle Blöcke, *Matlab-Function* genannt, bereit, mit denen selbst geschriebene Matlab-Funktionen in Simulink integriert werden können. Damit ist es möglich, die Vorteile der obengenannten Varianten zu vereinen. Jenen Vorteilen steht eine etwas längere Einarbeitungszeit gegenüber.

7.1.2.1 Standardregelkreis

In diesem Unterabschnitt sollen die Herausforderungen, die sich beim Entwurf des Regelungssystems ergeben, exemplarisch anhand eines Standardregelkreises gezeigt werden.

Der Standardregelkreis besteht aus einer Regelstrecke und einem Regler. Regler und Regelstrecke sind in Kette geschaltet, der Ausgang der Regelstrecke, *Regelstrecke* genannt, wird zum Eingang, der *Führgröße*, gegengekoppelt. Die Differenz zwischen Führ- und Regelgröße ergibt dann die *Regelabweichung*, welche vom Regler in die *Stellgröße* umgesetzt wird. Dabei sollte die Umsetzung derart erfolgen, dass die Regelabweichung minimal wird. Dabei kann das Verhalten der Strecke zusätzlich durch externe Größen, die *Störgrößen*, beeinflusst werden. In Abbildung 38 wird ein solcher Standardregelkreis dargestellt.

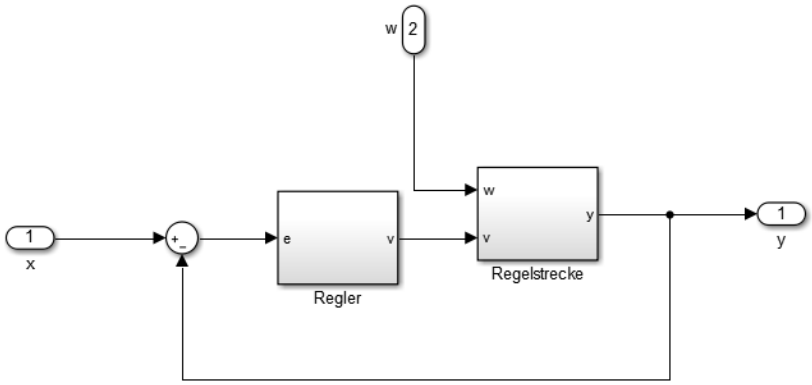


Abbildung 38: Standardregelkreis

Kürzel	**Bezeichnung**
x	Führgröße
e	Regelabweichung
v	Stellgröße
w	Störgröße
y	Regelgröße

Tabelle 12: Größen des Standardregelkreises

Allgemein muss ein Regelkreis folgende Bedingungen erfüllen. Sei

$$H_L(s) = H_R(s) \cdot H_M(s) \tag{7.1}$$

mit

$$H_R(s) = \frac{y(s)}{v(s)} \tag{7.2}$$

$$H_M(s) = \frac{y(s)}{v(s)} \tag{7.3}$$

die Schleifenverstärkung des Regelkreises.

Dann gilt

$$\forall \mid L(s) \mid \geq 1 \quad : \quad \mid \phi\,(L(s)) \mid < 180^\circ \Leftrightarrow \text{Regelkreis stabil} \tag{7.4}$$

Weil zudem gilt

$$H_T(s) = \frac{H_L(s)}{1 + H_L(s)} \tag{7.5}$$

ist man bestrebt, $|H_L(s)|$ für ein möglichst schnelles Verhalten des Regelkreises unter Einhaltung von Gleichung 7.4 zu maximieren.

Des weiteren sollen die Einflüsse der Störgrößen auf den Regelkreis minimiert werden. Diese wird durch Gleichung 7.6 dargelegt.

$$H_{LS}(s) = H_S(s) \cdot (1 + H_L(s)) \tag{7.6}$$

mit

$$H_S(s) = \frac{v}{w} \tag{7.7}$$

Ausgehend von den obigen allgemeineren Ausführungen werden die Größen der für die Realisierung in Betracht gezogenen Regelkreise genauer spezifiziert. Da es sich dabei um ein Mehrgrößensystem handelt, existieren zur jeder Kategorie mehrere Größen.

Es existieren folgende Führgrößen, mit den korrespondierenden Regelgrößen:

- *gewünschte Kursabweichung des Luftschiffs*: gibt die zu erreichende Abweichung des Luftschiffs vom Kurs an. Sie ist standardmäßig gleich 0, muss aber für kleine Abstände vergrößert werden.
- *Position des Luftschiffs*: gibt die Positon des Luftschiffs in der XY-Ebene in kartesischen Koordinaten an.
- *Höhe des Luftschiffs*: Gibt die Meereshöhe des Lufschiffs an

Die Stellgrößen ergeben sich nicht direkt aus der Differenz der Regel- und Führgrößen. Es erweist sich als praktikabeler, diese in folgende Regelabweichungen umzurechnen.

- *Abstand zum Zielpunkt*: Aus der momentanen Position und der zu erreichenden Position wird mittels des Satz des Pythagoras der Abstand berechnet
- *Kursabweichung*: Mittels der Lage des Zielpunkts und dem tatsächlichen Gierwinkel kann die momentane Kursabweichung angegeben werden.
- *Höhendifferenz*: Ergibt sich aus der Differenz der zu erreichenden und der tatsächlichen Höhe.

Die Werte der zur Umrechnung benötigten Funktionen müssen, um den Rechenaufwand für den Mikrocontroller zu reduzieren, innerhalb von Arrays abgespeichert werden.

Die zu den Regelabweichungen korrespondierenden Stellgrößen sind *virtuell*, dass heißt, sie können nicht direkt auf die Regelstrecke beanschlagt werden. Sie werden, weil sie spezifisch für die einzelnen Konfigurationsvorschläge sind, später erläutert.

7.1.2.2 2. und 3. Konfigurationsvorschlag

Bevor näher auf den gesamten Regelkreis eingegangen werden kann, ist es sinnvoll, die Simulink-Implementierungen der in Betracht gezogenen Konfigurationsvorschläge zu erläutern. Weil sich der 2. und der 3. Konfigurationsvorschlag in vielen Punkten ähneln, wird die Realisierung ihrer Simulation in diesem Unterabschnitt behandelt.

Die Modellierung erfolgt dabei wie im Abschnitt *Mechanischer Aufbau* beschrieben (siehe 8.3.5.4). In Abbildung 39 wird das Blockschaltbild für beide Modelle gezeigt. Unterschiede zwischen ihnen zeigen sich erst beim Öffnen der Matlab-Funktion.

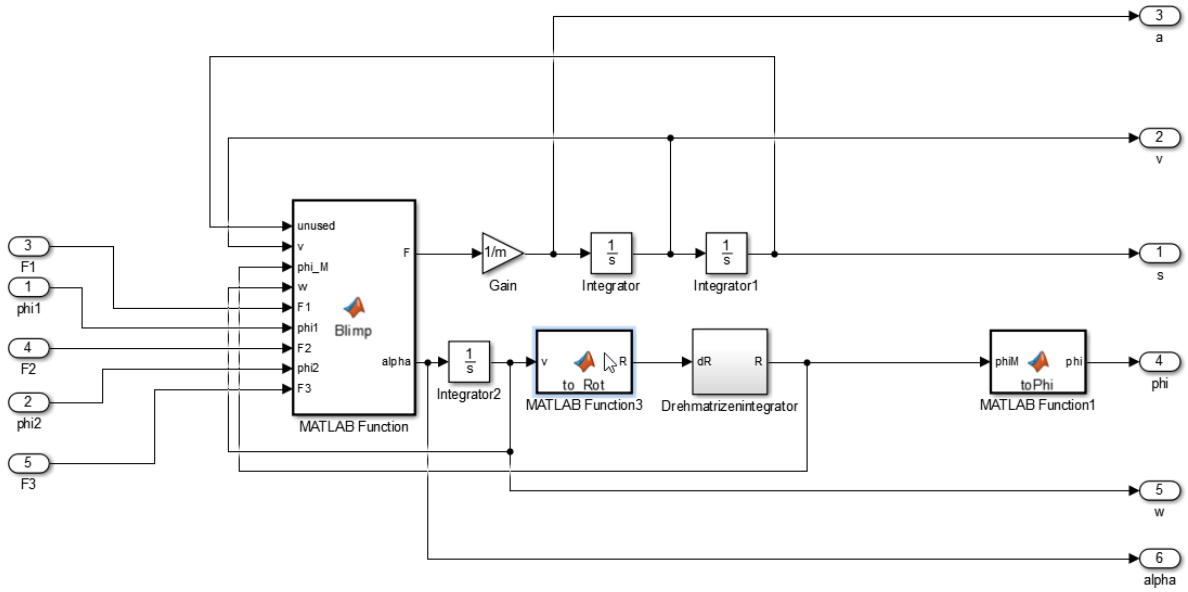


Abbildung 39: Simulink-Blockschaltbild für beide Regelstrecken

Das Modell zeigt sich von folgenden Größen abhängig:

- *F1*: Kraft des linken Rotors
- *F2*: Kraft des rechten Rotors
- *F3*: Kraft des hinteren Rotors
- *phi1*: Auslenkung des linken Rotors
- *phi2*: Auslenkung des rechten Rotors

Weil der linke und der rechte Rotor über eine Kohlefaserstange fest miteinander verbunden sind, gilt immer $phi1 = phi2$.

Folgende Zustandsgrößen werden ausgegeben:

- a: translatorische Beschleunigung des Luftschiffs
- v: translatorische Geschwindigkeit des Luftschiffs
- s: Ort des Luftschiffs
- *alpha*: rotatorische Beschleunigung des Luftschiffs
- w: rotatorische Geschwindigkeit des Luftschiffs
- *phi*: Lage des Luftschiffs

Die Eingangssignale werden von folgenden Blöcken verarbeitet:

- *Matlab Function*: Dieser Block beschreibt die Gesamtkraft, das gesamte Drehmoment in Abhängigkeit von der Lage, der Geschwindigkeiten sowie der Rotorenkräfte. In diesem Block unterscheiden sich die Realisierungen der Konfigurationsvorschläge 2 und 3.

- *Gain*: Mit diesem Block wird die gesamte auf das Luftschiff wirkende Kraft in eine Beschleunigung umgesetzt.
- *Integrator*: Setzt die translatorische Beschleunigung in die translatorische Geschwindigkeit um.
- *Integrator1*: Setzt die translatorische Geschwindigkeit in den zurückgelegten Weg um.
- *Integration*: Setzt die rotatorische Beschleunigung in die rotatorische Geschwindigkeit um.
- *Matlab Function3*: Konvertiert die rotatorische Geschwindigkeit von ihrer Vektorschreibweise in eine infinitesimale Rotationsmatrix.
- *Drehmatrixintegrator*: Setzt die Rotationsgeschwindigkeit in die Ausrichtung des Luftschiffs um.

Es sei angemerkt, dass es sich bei sämtlichen Signalen mit Ausnahme der Eingangsgrößen um vektorielle Größen handelt, was aus Abbildung 39 nicht sofort ersichtlich ist.

7.1.2.3 Test des zweiten Konfigurationsvorschlags

Bevor das Modell in den Regelkreis implementiert werden kann, muss zuerst die dessen Funktion getestet werden. Für den 2. Konfigurationsvorschlag werden einige Tests durchgeführt. Für den 3. Konfigurationsvorschlag haben sich diese Tests aufgrund der minimalen Unterschiede erübrigt.

Das Simulink-Modell des zweiten Konfigurationsvorschlags wird mittels *Model Referencing* in ein anderes Modell, das Testmodell, eingebunden. Die Eingänge werden mit Konstanten beaufschlagt, die Ausgänge mittels Scopes gemessen oder in den *Matlab-Workspace* exportiert. (siehe Abbildung 40)

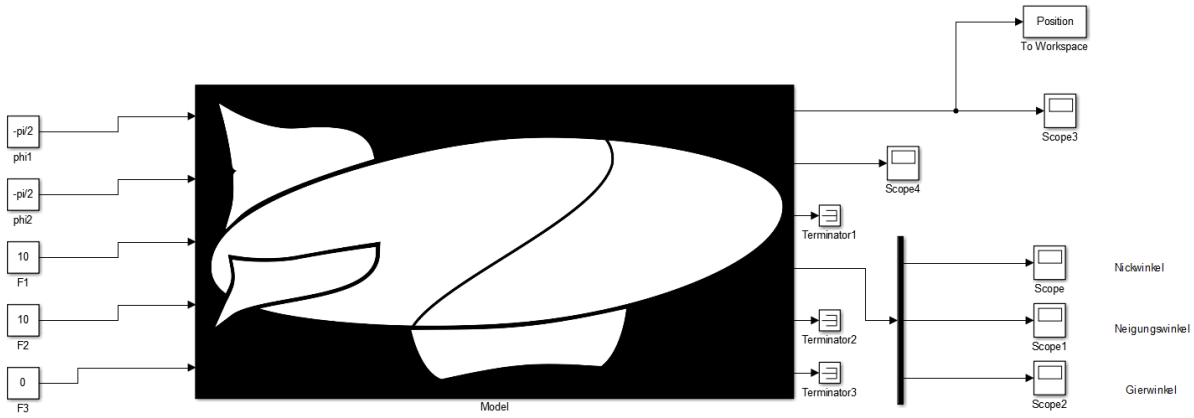


Abbildung 40: Aufbau des Testmodels

Test 1: Loopings fliegen

Für diesen Test werden F1 sowie F2 auf 10N gestellt. Es ergibt sich folgende Abbildung.

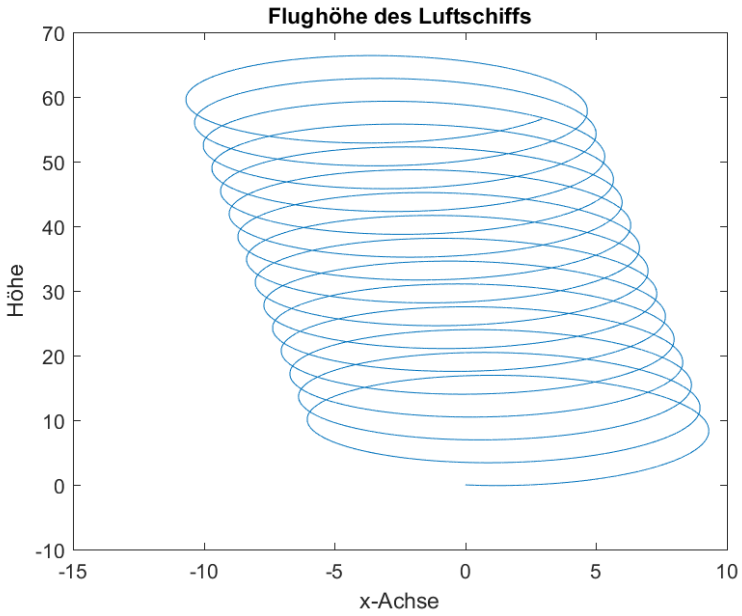


Abbildung 41: Loopings

Test 2: Kreise fliegen

Für diesen Test werden F1 sowie F2 auf 5N und F3 auf 1N gestellt. Die sich ergebende Flugbahn wird in Abbildung 42 dargestellt.

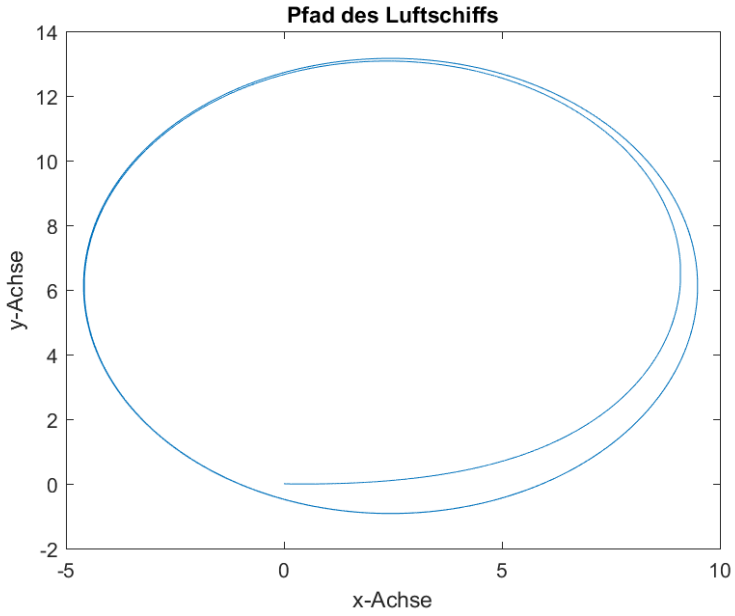


Abbildung 42: Kreise

7.1.2.4 Regelung

In diesem Abschnitt sollen die Regelungen der Konfigurationsvorschläge 2 und 3 genauer in Betracht gezogen werden. Es wird der zu erfüllende Zweck erläutert sowie die dafür notwendigen

Algorithmen ausgearbeitet.

Vorab sei darauf hingewiesen, dass die spezifizierten Zielpunkte nur mit einer relativ hohen Ungenauigkeit von etwa 10 Metern angeflogen werden kann, was durch die Ungenauigkeit des GPS sowie der einwirkenden Störgrößen bedingt ist. Demzufolge macht die Implementierung einer allzu komplizierten Regelung nur wenig Sinn, zumal die Rechenleistung des *STM32* ein begrenzender Faktor dieser Regelung ist. Um den Entwurf des Regelungssystems zu vereinfachen, wird dieses in zwei Unterkomponenten aufgeteilt:

- *Translatorische Regelung*: Diese Regelung ist für den Vorwärts- sowie Steig- und Sinkflug des Luftschiffs verantwortlich. Sie ist für beide Konfigurationsvorschläge diesselbe.
- *Rotatorische Regelung*: Diese Regelung ist für die korrekte Ausrichtung des Luftschiffes verantwortlich. Sie soll die Abweichung vom gewünschten Kurs minimieren und das Luftschiff in horizontaler Lage halten. Jene Regelung ist für die beiden Konfiguratonsvorschläge unterschiedlich ausgelegt.

Die oben angeführten Regelungen sollen nun im Einzelnen erläutert werden.

Translatorische Regelung

Folgende Größen sind für die translatorische Regelung relevant:

- *Regelabweichungen*: Abstand vom Zielpunkt (Δr); Höhendifferenz (Δh);
- *Stellgrößen*: horizontale Rotorenkraft(F_x); vertikale Rotorenkraft (F_y)

Jene Größen lauten als Vektoren formuliert

$$\vec{s} = \begin{pmatrix} \Delta R \\ \Delta h \end{pmatrix} \tag{7.8}$$

$$\vec{F} = \begin{pmatrix} F_x \\ F_y \end{pmatrix} \tag{7.9}$$

Als Regelansatz werden die Stellgrößen so gewählt, dass der Kraftvektor Richtung Ziel zeigt, wie in Gleichung 7.10 dargelegt wird.

$$\vec{F} = p_1 \cdot \vec{s} \tag{7.10}$$

Dieser Regelansatz kann um einen P-Wert erweitert werden. (Gleichung 7.11)

$$\vec{F} = M \cdot \vec{s} \tag{7.11}$$

mit

$$M = \begin{pmatrix} P_1 & 0 \\ 0 & P_2 \end{pmatrix} \tag{7.12}$$

P_1 und P_2 bezeichnen die Werte der verwendeten P-Regler. Hierbei sollte P_2 größer als P_1 gewählt werden, um die Gewichtskraft des Luftschiffs sowie den höheren Luftwiderstand in Querrichtung zu berücksichtigen. Unterschreitet ΔR bzw. Δh einen bestimmten Mindestwert,

werden sie, mittels einer Gewichtungsfunktion kontinuierlich zurückgesetzt. Die geschieht deshalb kontinuierlich, um durch Unstetigkeiten verursachte Instabilitäten des Regelungssystems zu umgehen. Die verwendete Gewichtungsfunktion muss zum Einen einfach berechenbar sein, zum Anderen eine stetige 1. Ableitung besitzen. Als Ansatz wird eine stückweise definierte Funktion angenommen, wie in Gleichung 7.13 dargelegt wird.

$$g(x) = \begin{cases} 0 & \text{wenn } x < -1 \\ f_1(x) & \text{wenn } -1 \geq x \geq 0 \\ f_2(x) & \text{wenn } 0 < x < 1 \\ 1 & \text{wenn } x > 1 \end{cases} \tag{7.13}$$

mit

$$f_1(x) = a_1 \cdot (x+1)^2 \tag{7.14}$$

$$f_2(x) = a_2 \cdot (x-1)^2 + 1 \tag{7.15}$$

Für $f_1(x)$ und $f_2(x)$ müssen folgende Gleichungen gelten:

$$\frac{d}{dx} f_1(x)\Big|_{x=0} = \frac{d}{dx} f_2(x)\Big|_{x=0} \tag{7.16}$$

$$2 \cdot a_1 \cdot (x+1)\Big|_{x=0} = 2 \cdot a_2 \cdot (x-1)\Big|_{x=0} \tag{7.17}$$

$$a_1 = -a_2 \tag{7.18}$$

Des Weiteren muss gelten:

$$f_1(0) = a_1 \cdot (0+1)^2 = \frac{1}{2} \tag{7.19}$$

$$a_1 = \frac{1}{2} \tag{7.20}$$

$$f_2(0) = a_2 \cdot (0-1)^2 + 1 = \frac{1}{2} \tag{7.21}$$

$$a_2 = -\frac{1}{2} \tag{7.22}$$

Einsetzen in $f_1(x)$ sowie $f_2(x)$ ergibt:

$$f_1(x) = \frac{1}{2} \cdot (x+1)^2 \tag{7.23}$$

$$f_2(x) = -\frac{1}{2} \cdot (x-1)^2 \tag{7.24}$$

Damit ist die Gewichtungsfunktion definiert.

Rotatorische Regelung

Für die rotatorische Regelung sind folgende Größen relevant:

- *Regelabweichung*: Abweichung vom absoluten Kurs in Grad ($\Delta\phi_z$) und von der horizontalen Lage ($\Delta\phi_y$)

- *Stellgrößen*:
  - M_z: Giermoment
    - für Konfigurationsvorschlag 2: F_{R3}
    - für Konfigurationsvorschlag 3: F_{R1}, F_{R2}, ϕ_R
  - M_y: Nickmoment
    - für Konfigurationsvorschlag 2: F_{R1}, F_{R2}, ϕ_R
    - für Konfiguratonsvorschlag 3: F_{R3}

Von den oben genannt Stellgrößen kann je Konfigurationsvorschlag nur eine direkt, über einen einzelnen Rotor, beeinflusst werden. Beim 2. Konfigurationsvorschlag wäre dies das Giermoment, da der Rotor, wie im mechanische Aufbau auf Seite 145 beschrieben wird, nach unten zeigt. Dadurch erhält das Luftschiff eine höhere Wendigkeit, allerdings ist es nur bedingt fähig, eine waagrechte Ausrichtung, insbesondere was den Vorwärtsflug betrifft, zu halten.

Zur Realisierung besser geeignet ist der 3. Konfiguratonsvorschlag. Weil es, wie auf Seite 145 beschrieben wird, möglich ist, dass Nickmoment direkt zu kontrollieren, kann eine waagrechte Ausrichtung einfacher gehalten werden. Dementsprechend vereinfacht sich das zu realisierende Regelungssystems, allerdings müssen Abstriche bei der Wendigkeit gemacht werden.

Jene Größen lauten als Vektoren formuliert, wobei für folgende Gleichungen selbiges wie für (8.73) gilt.

$$\vec{M} = \begin{pmatrix} M_y \\ M_z \end{pmatrix} \tag{7.25}$$

$$\vec{F} = \begin{pmatrix} -F_{R1} \cdot \sin(\phi_R) \\ F_{R1} \cdot \cos(\phi_R) \\ -F_{R2} \sin(\phi_R) \\ F_{R2} \cdot \cos(\phi_R) \\ F_{R3} \end{pmatrix} \tag{7.26}$$

für Konfigurationsvorschlag 2 sowie

$$\vec{F} = \begin{pmatrix} -F_{R1} \cdot \sin(\phi_R) \\ F_{R1} \cdot \cos(\phi_R) \\ -F_{R2} \sin(\phi_R) \\ F_{R2} \cdot \cos(\phi_R) \\ F_{R3} \cdot \cos(\phi_{R3}) \\ F_{R3} \cdot \sin(\phi_{R3}) \end{pmatrix} \tag{7.27}$$

für Konfigurationsvorschlag 3

Dann ergibt sich $\vec{M}$ aus $\vec{F}$ wie folgt:

$$\vec{M} = A \cdot \vec{F} \tag{7.28}$$

mit

$$A = \begin{pmatrix} -r_{1z} & r_{1x} & -r_{2z} & r_{2x} & 0 \\ r_{1y} & 0 & r_{2y} & 0 & -r_{3x} \end{pmatrix} \tag{7.29}$$

für Konfigurationsvorschlag 2 sowie

$$A = \begin{pmatrix} -r_{1z} & r_{1x} & -r_{2z} & r_{2x} & -r_{3z} & r_{3x} \\ r_{1y} & 0 & r_{2y} & 0 & r_{3y} & 0 \end{pmatrix} \tag{7.30}$$

für Konfigurationsvorschlag 3

Für Gleichung 7.30 ergibt sich mit $r_{3y} = 0$:

$$A = \begin{pmatrix} -r_{1z} & r_{1x} & -r_{2z} & r_{2x} & -r_{3z} & r_{3x} \\ r_{1y} & 0 & r_{2y} & 0 & 0 & 0 \end{pmatrix} \tag{7.31}$$

Die zweite Zeile von (7.31) ist dünner besetzt als die zweite Zeile von (7.29), was auf eine geringere Manövrierbarkeit des Luftschiffs beim dritten Konfiguratonsvorschlag hinweist.

Der vierte Eintrag der ersten Zeile von (7.29) ist unbesetzt, wodurch das Nickmoment durch die ersten beiden Rotoren erzeugt werden muss.

Für den zweiten Konfigurationsvorschlag wird als Regelansatz für den Gierwinkel das Giermoment proportional zur Kursabweichung gesetzt.

$$M_z = -k_3 \cdot \Delta\phi_z \Leftrightarrow F_{R3} = P_3 \cdot \Delta\phi_z \tag{7.32}$$

$\Leftrightarrow$: Die Äquivalenz ergibt sich daraus, dass A für die Konfigurationsvorschläge 2 und 3 eine Konstante ist.

Das negative Vorzeichen kommt durch die Ausrichtung des hinteren Rotors zustande.

Der Nickwinkel des Luftschiffs ist aufgrund der Luftreibung proportional zu dessen Geschwindigkeit (siehe Seite 145). Dementsprechend kann die Abweichung von der waagrechten Lage nur über die Reduktion der Geschwindigkeit verkleinert werden. In der Simulaton ist es möglich, auf die Regelung des Nickwinkels zu verzichten. Das Luftschiff erreicht den spezifizierten Zielpunkt.

Für das reale Luftschiff ist jedoch die Ausrichtung der Rotoren aufgrund des Servomotors auf $-90° \geq \phi_3 \geq 90°$ beschränkt. Bei Berücksichtigung dieser Einschränkung kann der spezifizierte Zielpunkt nicht mehr angeflogen werden.

Für den dritten Konfigurationsvorschlag gilt folgender Ansatz für den Gierwinkel:

$$\Delta F = P_3 \cdot \Delta\phi_z \tag{7.33}$$

mit

$$\Delta F = F_1 - F_2 \tag{7.34}$$

$$F_{ges} = F_1 + F_2 \tag{7.35}$$

mit

$$F_{ges} = \sqrt{F_x^2 + F_y^2} \tag{7.36}$$

Der Ansatz für den Nickwinkel des dritten Konfigurationsvorschlags wird in Gleichung 7.37 dargelegt.

$$M_y = k_4 \cdot \Delta\phi_y \Leftrightarrow F_{R3} = P_4 \cdot \Delta\phi_y \tag{7.37}$$

Damit ist gewährleistet, dass die rotatorische Regelung die translatorische nicht allzu sehr beeinflusst.

Konvertierung der Stellgrößen

Die Stellgrößen der translatorischen Regelung können nicht direkt auf das Luftschiff beaufschlagt werden. Des weiteren können durch Stellgrößenbeschränkungen unerwünschte Abweichungen auftreten. Deshalb müssen sie zunächst normiert werden, wie in Gleichung 7.38 dargelegt wird.

$$\vec{F}^* = \begin{cases} \vec{F} & \text{wenn } |F| \leq F_{max} \\ \frac{\vec{F}}{|F|} & \text{wenn } |F| > F_{max} \end{cases} \tag{7.38}$$

ΔF der rotatorischen Regelung für den dritten Konfigurationsvorschlag muss ebenfalls begrenzt werden.

Die Ermittlung von ϕ_R erfolgt mittels arctan. Dessen Funktionswerte werden mit den zugehörigen Argumenten, um Rechenzeit zu sparen, innerhalb des µC als Array abgespeichert.

$$\phi_R = \arctan\left(\frac{F_x^*}{F_y^*}\right) \tag{7.39}$$

7.2 Debugger

7.2.1 Allgemeines

Um den vielen digitalen Elementen in der Diplomarbeit eine Testmöglichkeit zu bieten, wird eine Debugmöglichkeit via Bluetooth (siehe 5.5 und 6.8) geschaffen. Daher wird eine Benutzeroberfläche benötigt, welche es dem Anwender erlaubt, die digitalen Komponenten während des Betriebes zu überwachen und diese, wenn nötig, auch zu verändern. Diese PC-Anwendung ist eine WPF-Anwendung, die mit Visual Studio 2015 (siehe 11) entwickelt worden ist. Um dem Anwender die Debugmöglichkeit zu bieten, wird ein Protokoll (siehe Tabelle 14) benötigt, welches im *Bluetooth_DebugWindow* angewendet werden kann.

7.2.1.1 Protokoll

Command	**Description**
clear	clears the entire history of both command windows
clear received	clears the received command window history
clear sent	clears the sent command window history
debugData	Prints added variables in *Add_DebugVariable* (see 6.8) frequently
Category seperated	**The following commands print out data, which can be defined in *Write_DebugInfo* (see 6.8).**
gsm	activates the gsm debug mode
gps	activates the gps debug mode
mpu	activates the mpu debug mode
servo	activates the servo debug mode
bldc	activates the bldc debug mode
controller	activates the controller debug mode

Tabelle 13: Bluetooth Protokoll

7.2.2 GUI

Die GUI des *Bluetooth_DebugWindow* erfüllt folgende Funktionen und wird mit den beschriebenen Steuerelementen realisiert:

Thema	**Steuerelement**
Öffnen bzw. Schließen der Bluetoothverbindung	ToggleSwitch
Aktiveren bzw. Deaktiveren des Debugmodus	ToggleSwitch
Eingeben von Kommandos	TextBox
Sendeverlauf	TextBox
Empfangsverlauf	TextBox
Datentabelle	DataGrid
Portauswahl	ComboBox
Aktualisierungszeit der Datentabelle	TextBox
Speichern des Datentabellenverlaufs	Button

Tabelle 14: Steuerelemente

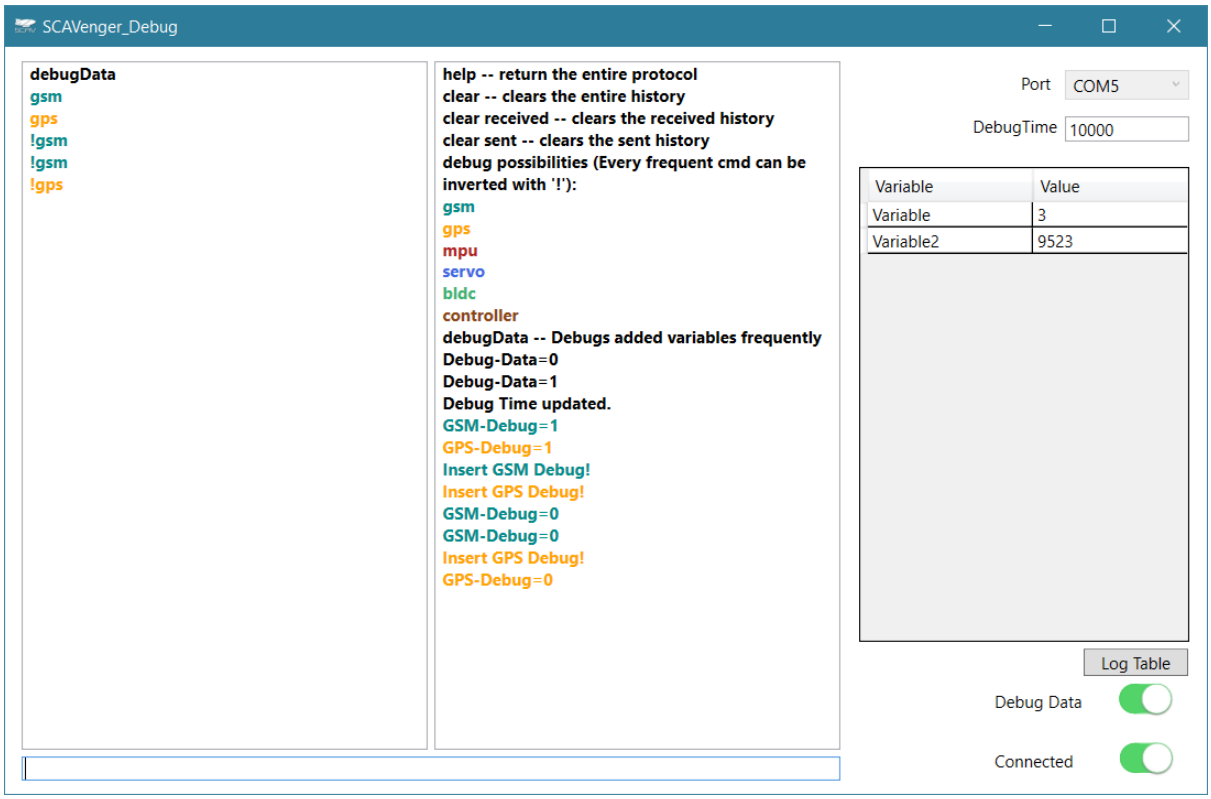


Abbildung 43: Bluetooth_DebugWindow

7.2.2.1 Thema: Öffnen bzw. Schließen der Bluetoothverbindung
Steuerelement: ToggleSwitch

Der ToggleSwitch ist ein von Marc Angers entwickeltes Steuerelement für WPF. Dieses Steuerelement wird verwendet, um die Bluetoothverbindung zu öffnen bzw. zu schließen.[13] Die Wartezeit während des Öffnens dauert ca. 3-10 Sekunden, um diese zu überbrücken, wurde eine Animation eingebaut.

7.2.2.2 Thema: Aktiveren bzw. Deaktiveren des Debugmodus
Steuerelement: ToggleSwitch

Das Steuerelement von Marc Angers[13] wird noch einmal eingebaut, um den Debugmodus zu aktivieren bzw. zu deaktiveren. Dies soll die Bedienung der Anwendung einfacher und schneller gestaltlten.

7.2.2.3 Thema: Eingeben von Kommandos (Kommandozeile)
Steuerelement: TextBox

Um die Kommandos, welche an das Bluetooth-Modul des µC gesendet werden, eingeben zu können, wird eine TextBox verwendet. Die TextBox ist ein Standardsteuerelement, welches im .NET Framework[17] enthalten ist.

[17] Microsoft .NET Framework

7.2.2.4 Thema: Sendeverlauf
Steuerelement: TextBox

Im Sendeverlauf sind alle Kommandos zu sehen, die in der Kommandozeile eingegeben und versendet wurden. Als Steuerelement wird eine TextBox im "Multiline-Mode" verwendet.

7.2.2.5 Thema: Empfangsverlauf
Steuerelement: TextBox

Wenn über Bluetooth Nachrichten empfangen werden, werden sie im Empfangsverlauf dargestellt. Hierbei handelt es sich, wie beim Sendeverlauf (siehe 7.2.2.4), um eine TextBox im "Multiline-Mode".

7.2.2.6 Thema: Datentabelle
Steuerelement: DataGrid

Das Steuerelement, das für die Datentabelle verwendet wird, ist das DataGrid. Mithilfe von diesem .NET[18] Steuerelement[14] können Einträge in einer Tabelle dargestellt und editiert werden. In dieser Datentabelle werden die Daten, welche mit dem Zeiger-Debugmodus (siehe 6.8.1.1) gedebugt werden, angezeigt, verändert und aktualisiert je nach Einstellung der Aktualisierungszeit.

7.2.2.7 Thema: Portauswahl
Steuerelement: ComboBox

Um den Port, mit dem sich der PC verbinden soll, darzustellen, wird die ComboBox des .NET Frameworks verwendet. In dieser werden alle verfügbaren Ports aufgelistet. Von diesen Ports kann der gewünschte Port ausgewählt werden.

7.2.2.8 Thema: Aktualisierungszeit der Datentabelle
Steuerelement: TextBox

Um die Aktualisierungszeit des Zeigerdebugmodus und infolge auch die Aktualisierungszeit der Datentabelle einstellen zu können, wird eine TextBox hinzugefügt, in der die gewünschte Zeit eingetragen wird.

7.2.2.9 Thema: Speichern des Datentabellenverlaufs
Steuerelement: Button

Die Daten, welche durch den Zeigerdebugmodus empfangen werden, werden im Hintergrund aufgezeichnet. Um diese aufgezeichneten Daten zu speichern, kann der Button Log Table gedrückt werden, welcher den Vorgang zur Datenspeicherung in eine CSV-Datei einleitet.

7.2.3 Klassenaufbau

Der Klassenaufbau des Bluetooth_DebugWindows ist relativ simpel und enthält folgende Klassen:

[18] Microsoft .NET Framework

- CmdHistory
- TextSelection
- DebugData
- CsvCollumn
- CsvWriter
- MainWindow

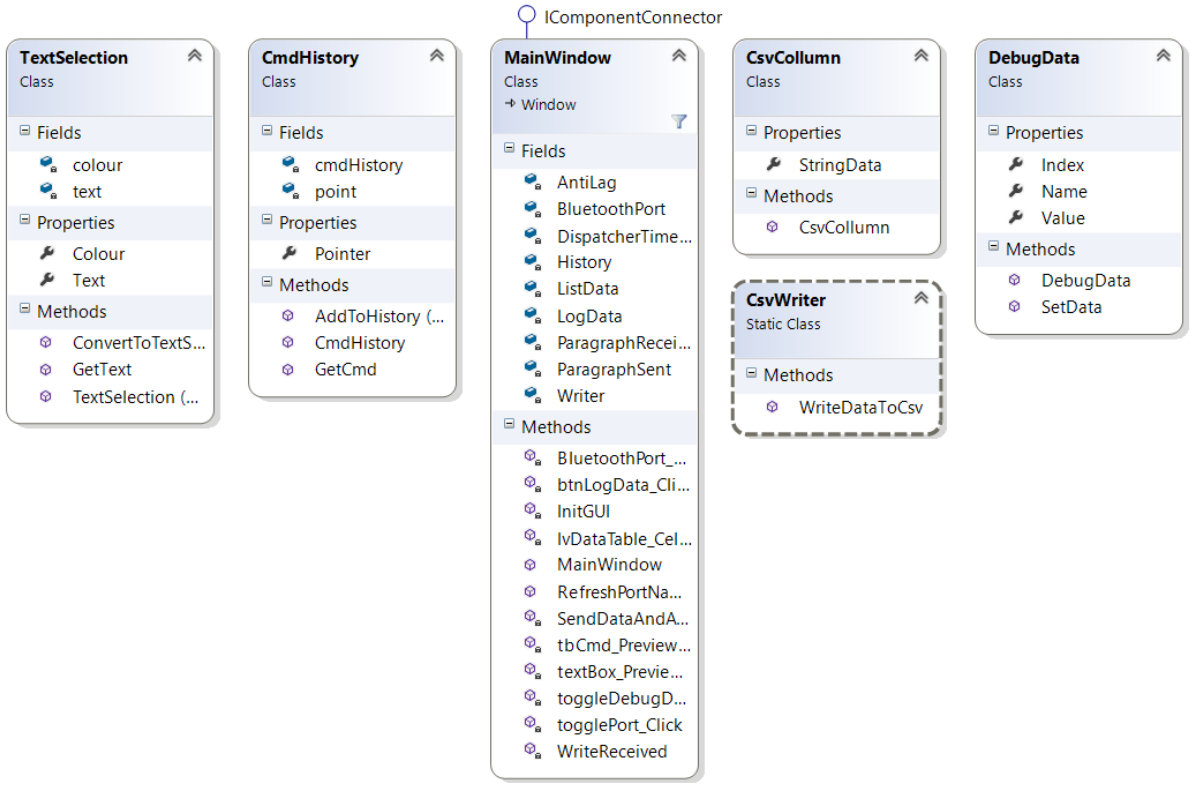


Abbildung 44: Klassendiagramm des Bluetooth_DebugWindows

Kommandoverlauf
Klasse: CmdHistory

Um die Benutzerfreundlichkeit der Anwendung zu erhöhen, wird ein Kommandoverlauf realisiert. Dieser ermöglicht es, mit den Pfeiltasten zwischen zuvor eingegeben Befehlen zu wechseln. Das reduziert den Schreibaufwand des Anwenders und macht die Verwendung schneller und praktikabler.

Textabschnitt
Klasse: TextSelection

Diese Klasse hat den Zweck jeden Befehlsabschnitt, welcher eingegeben wird, einer Farbe zuzuordnen. Somit hat jede Kategorie (siehe 6.8.1.1), welche gedebugt wird, eine eigene Farbe. So lassen sich die Daten, die empfangen werden, leichter unterscheiden und beobachten.

Debugdaten
Klasse: DebugData

Um die Daten, welche beim Debuggen von Zeigern (siehe 6.8.1.1) empfangen werden, zu verwalten, wird die Klasse DebugData benötigt. Diese fasst alle wesentlichen Informationen der Daten für die Weiterverarbeitung zusammen.

CSV Spalten
Klasse: CsvCollumn

Da die Daten, die beim Debuggen von Zeiger (siehe 6.8.1.1) entstehen, eventuell auch für eine Analyse aufgezeichnet werden müssen, wird die CsvCollumn Klasse realisiert. Diese enthält für jeweils eine Variable, alle Messwerte, die über die Zeit empfangen werden.

CSV Daten speichern
Klasse: CsvWriter

Diese statische Klasse enthält die Methode, welche mit den CsvCollumn die schlussendliche CSV-Datei erzeugt.

MainWindow
Klasse: MainWindow

Im MainWindow werden alle benötigten Initialsierungen vorgenommen, alle Events abonniert und sonstige wichtige Funktionen und Methoden definiert.

7.2.3.1 Kommandoverlauf

Codeabschnitt 32: CmdHistory.cs

```
1  class CmdHistory
2  {
3
4      TextSelection[] cmdHistory;
5      private int point;
6      public int Pointer
7      {
8          get { return point; }
9          set
10         {
11             if (value == -1)
12                 point = 0;
13             else if (value == cmdHistory.Length + 1)
14                 point = cmdHistory.Length;
15             else
16                 point = value;
17
18         }
19     }
20     public CmdHistory(int amount)
21     {
22         cmdHistory = new TextSelection[amount];
23         for (int i = 0; i < amount - 1; i++)
24             cmdHistory[i] = new TextSelection();
25         cmdHistory[0].Text = "";
26     }
27     public void AddToHistory(string text)
28     {
```

```
29          TextSelection data = TextSelection.ConvertToTextSelection(text);
30          for (int i = cmdHistory.Length - 1; i > 1; i--)
31          {
32              cmdHistory[i] = cmdHistory[i - 1];
33          }
34          cmdHistory[1] = data;
35      }
36      public void AddToHistory(TextSelection data)
37      {
38          for (int i = cmdHistory.Length - 1; i > 1; i--)
39          {
40              cmdHistory[i] = cmdHistory[i - 1];
41          }
42          cmdHistory[1] = data;
43      }
44
45      public TextSelection GetCmd()
46      {
47          return cmdHistory[Pointer];
48      }
49
50
51
52  }
```

Die Klasse CmdHistory (siehe Abbildung 45) hat folgende Member:

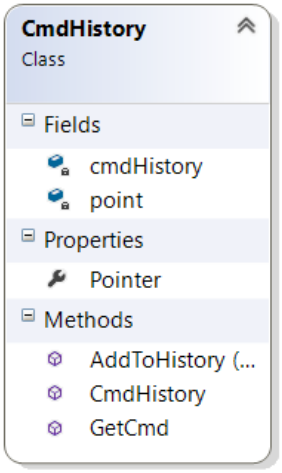


Abbildung 45: Klassendiagramm: CmdHistory

int Pointer
Klasse: CmdHistory

Diese Eigenschaft gibt die aktuelle Position im Kommandoverlauf an. Dies ist der Index des aktuellen Textes bzw. Kommandos, welcher in der Kommandozeile der Anwendung geschrieben steht. Beim Drücken der Pfeiltasten hoch und runter wird diese Zahl inkrementiert bzw. dekrementiert.

AddToHistory(string text)
Klasse: CmdHistory

Diese Methode fügt den in der Kommandozeile eingegeben Text, welcher als Argument mitgegeben wird, dem Kommandoverlauf hinzu. Dabei werden die schon im Kommandoverlauf enthal-

tenen Kommandos, um je eine Stelle nach hinten geschoben und das neue an der ersten Stelle angefügt. Die nullte Stelle wird immer freigehalten, da dort neue Kommandos eingegeben werden können.

TextSelection GetCmd()
Klasse: CmdHistory

Diese Funktion gibt den aktuellen Text, der im Kommandoverlauf mit der Eigenschaft *Pointer* ausgewählt ist, zurück.

7.2.3.2 Text Abschnitt

Codeabschnitt 33: TextSelection.cs

```
1  public class TextSelection
2  {
3      private string text;
4      private Color colour;
5
6      public string Text { get { return text; } set { text = value; } }
7      public Color Colour { get { return colour; } set { colour = value; } }
8      public TextSelection()
9      {
10         text = "";
11         colour = Colors.Black;
12     }
13     public TextSelection(string tx, Color col)
14     {
15         text = tx;
16         colour = col;
17     }
18
19     public static string GetText(TextSelection temp)
20     {
21         if(temp.Colour == Colors.DarkCyan) //GSM, GPS, MPU, SERVO, BLDC, ↙
               ↳ CONTROLLER, GLOBAL
22             return temp.Text + "*a";
23         if (temp.Colour == Colors.Orange)
24             return temp.Text + "*b";
25         if (temp.Colour == Colors.Firebrick)
26             return temp.Text + "*c";
27         if (temp.Colour == Colors.RoyalBlue)
28             return temp.Text + "*d";
29         if (temp.Colour == Colors.MediumSeaGreen)
30             return temp.Text + "*e";
31         if (temp.Colour == Colors.SaddleBrown)
32             return temp.Text + "*f";
33         if (temp.Colour == Colors.Black)
34             return temp.Text + "*g";
35         return "";
36     }
37     public static TextSelection ConvertToTextSelection(string text)
38     {
39         if (!text.Contains("*"))
40             text += "*g";
41         string [] array = text.Split('*');
42         Color tp = Colors.Black;
43         switch(array[1]) //GSM, GPS, MPU, SERVO, BLDC, CONTROLLER, GLOBAL
```

```
44          {
45              case "a":
46                  tp = Colors.DarkCyan;
47                  break;
48              case "b":
49                  tp = Colors.Orange;
50                  break;
51              case "c":
52                  tp = Colors.Firebrick;
53                  break;
54              case "d":
55                  tp = Colors.RoyalBlue;
56                  break;
57              case "e":
58                  tp = Colors.MediumSeaGreen;
59                  break;
60              case "f":
61                  tp = Colors.SaddleBrown;
62                  break;
63              case "g":
64                  tp = Colors.Black;
65                  break;
66          }
67          return new TextSelection(array[0], tp);
68      }
69
70  }
```

Die Klasse TextSelection (siehe Abbildung 46) hat folgende Member:

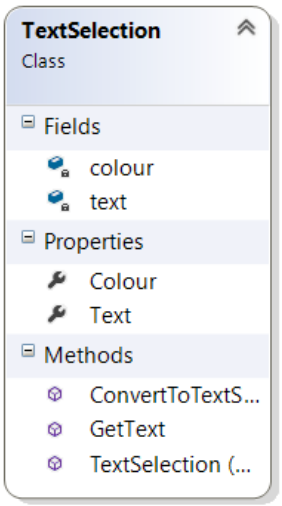


Abbildung 46: Klassendiagramm: TextSelection

string Text
Klasse: TextSelection

Die Eigenschaft Text enthält den Text des Textabschnittes, dieser kann Kommandos oder auch empfangene Daten enthalten.

Color Colour
Klasse: TextSelection

Um die Benutzerfreundlichkeit zu erhöhen werden die verschiedenen Kategorien, die gedebugt werden, im Empfangsverlauf aber auch im Sendeverlauf farbig auseinandergehalten. Die Information der Farbe ist in dieser Eigenschaft gespeichert. Diese Eigenschaft ist ein Objekt, welches im Namensraum System.Windows.Media enthalten ist.

string GetText(TextSelection temp)
Klasse: TextSelection

Die statische Funktion *GetText* konvertiert ein TextSelection Objekt in eine Zeichenkette. Die Eigenschaft der Farbe wird der Zeichenfolge am Ende angefügt. Dabei wird zuerst ein "*" ergänzt, dies markiert den Schluss des eigentlichen Textes. Darauf folgt ein Buchstabe, welcher jeweils ein Synonym für eine Farbe darstellt. Beispiel 1: TextSelection temp hat folgende Eigenschaften: *Colour* = Colors.Firebrick und *Text* = "Beispiel". Die Funktion *GetText* konvertiert diese zwei Eigenschaften nun zu einer Zeichenkette die lautet "Beispiel*c".

Buchstabe	Farbe	Kategorie
a	Colors.DarkCyan	GSM
b	Colors.Orange	GPS
c	Colors.Firebrick	MPU
d	Colors.RoyalBlue	SERVO
e	Colors.MediumSeaGreen	BLDC
f	Colors.SaddleBrown	CONTROLLER
g	Colors.Black	GLOBAL

Tabelle 15: Farbsynonyme

TextSelection ConvertToTextSelection(string text)
Klasse: TextSelection

Diese statische Funktion konvertiert einen Text in eine TextSelection. Bei fehlender Farbinformation wird standardmäßig ein "*g" angefügt, was für die Farbe schwarz steht (siehe Tabelle 15). Wenn hingegen die Zeichenkette schon eine Farbinformation enthält, dann wird diese mit den Farbsynonymen (siehe Tabelle 15) verglichen. Am Ende werden die Informationen über Farbe und Text in eine TextSelection umgewandelt und daraufhin zurückgegeben.

7.2.3.3 Debug Daten

Codeabschnitt 34: DebugData.cs

```
1  public class DebugData
2  {
3      public string Name { get; }
4
5      public int Value { get; set; }
6
7      public int Index { get; }
8
9      public DebugData(string name, int val, int ind)
10     {
11         Name = name;
12         Value = val;
```

```
13          Index = ind;
14      }
15      public void SetData(int value, SerialPort port)
16      {
17          Value = value;
18          if (Index > 9)
19              port.WriteLine("s" + Index + "*" + Value);
20          else
21              port.WriteLine("s0" + Index + "*" + Value);
22      }
23
24  }
```

Die Klasse DebugData (siehe Abbildung 47) hat folgende Member:

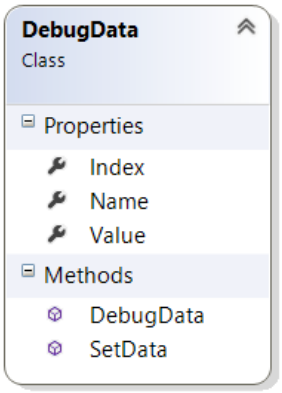


Abbildung 47: Klassendiagramm: DebugData

string Name
Klasse: DebugData

Diese Zeichenfolge beinhaltet den Titel der Variable, welche gedebugt wird (siehe 6.8.2.4). Dieser kann allerdings nur gelesen und nicht geschrieben werden.

int Value
Klasse: DebugData

Dieser Integer beinhaltet den Wert der gedebugten Variable (siehe 6.8.2.4), welcher sowohl geschrieben als auch gelesen werden kann.

int Index
Klasse: DebugData

Der Integer *Index* trägt die Information des Debugindexes, welcher vom µC gesendet wird (siehe 6.8.2.4). Dieser wird benötigt, damit die Variable, welche verändert wird, klar erkennbar ist. Dieser Index kann, ebenso wie der Titel der Debugvariable, nur gelesen, aber nicht geschrieben werden.

SetData(int value, TextSelection port)
Klasse: DebugData

Diese Methode überschreibt zuerst den Wert der Debugvariable durch den Parameter *value*, welcher mitgegebenen wird. Das "s", das zuallererst versendet wird, markiert für den µC, dass

eine Variable verändert wurde. Die Zeichenkette, welche sich nun aus "s", *Index* und *Value* zusammensetzt, wird an das Bluetooth-Modul versendet. Um diese Informationen zu versenden, wird das Objekt SerialPort verwendet, das sich im Namesraum System.IO.Ports befindet.

7.2.3.4 CSV Spalten

Codeabschnitt 35: CsvWriter.cs

```
1  public class CsvCollumn
2  {
3      public List<string> StringData { get; set; }
4      public CsvCollumn()
5      {
6          StringData = new List<string>();
7      }
8  }
```

Die Klasse CsvCollumn (siehe Abbildung 48) hat folgende Member:

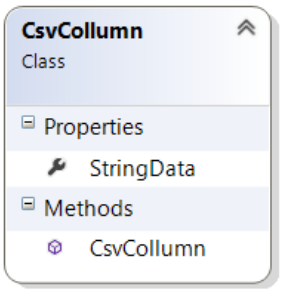


Abbildung 48: Klassendiagramm: CsvCollumn

Die Klasse CsvCollumn ist lediglich eine anschauliche Form für die Liste "List<string> StringData", welche in dieser als Eigenschaft realisiert, enthalten ist.

7.2.3.5 CSV Daten schreiben

Codeabschnitt 36: CsvWriter.cs

```
1  public static class CsvWriter
2  {
3      public static void WriteDataToCsv(List<CsvCollumn> collumns, StreamWriter ↙
           ↳ writer)
4      {
5          string text = "";
6          string col = "";
7          for (int row = 0; row < collumns[0].StringData.Count; row++)
8          {
9              for (int i = 0; i < collumns.Count; i++)
10             {
11                 col = "";
12                 if(i != 0)
13                     col += ";";
14                 text += col + collumns[i].StringData[row];
15             }
16             text += '\n';
17         }
18         writer.Write(text);
19     }
```

```
20  }
```

Die statische Klasse CsvWriter (siehe Abbildung 49) hat folgende Member:

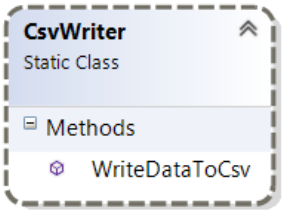


Abbildung 49: Klassendiagramm: CsvWriter

WriteDataToCsv(List<CsvCollumn> collumns, StreamWriter writer)
Klasse: CsvWriter

Diese statische Methode schreibt die Debugvariablen, welche in einer "List<CsvCollumn>" mitgegeben werden, mittels StreamWriter in eine Csv-Datei. Der StreamWriter ist ein Mittel, um Informationen in beliebigen Dateiformaten abzuspeichern. Dieser ist im Namesraum System.IO enthalten.

7.2.3.6 MainWindow

Codeabschnitt 37: MainWindow.xaml.cs

```
1  public partial class MainWindow : Window
2  {
3      #region Fields
4      SerialPort BluetoothPort;
5      CmdHistory History;
6      Paragraph ParagraphSent, ParagraphReceived;
7      System.Windows.Threading.DispatcherTimer DispatcherTimerAnimation;
8      Thread AntiLag;
9      List<DebugData> ListData; // anzahl der debggten Elemente
10     List<CsvCollumn> LogData; // jedes element bekommt eine eigene Liste
11     StreamWriter Writer;
12     bool CloseWindow = false;
13     #endregion
14
15     #region Initializing
16     public MainWindow()
17     {
18         InitializeComponent();
19         LogData = new List<CsvCollumn>();
20         ListData = new List<DebugData>();
21         BluetoothPort = new SerialPort("COM5", 57600, Parity.None, 8, ↙
               ↳ StopBits.One);
22         BluetoothPort.DataReceived += BluetoothPort_DataReceived;
23         BluetoothPort.Disposed += BluetoothPort_Disposed;
24         DispatcherTimerAnimation = new ↙
               ↳ System.Windows.Threading.DispatcherTimer();
25         DispatcherTimerAnimation.Tick += DispatcherTimer_Tick;
26         DispatcherTimerAnimation.Interval = new TimeSpan(0, 0, 0, 0, 300);
27         InitGUI();
28
29     }
30     void InitGUI()
31     {
```

```
32          RefreshPortNames();
33          History = new CmdHistory(20);
34          ParagraphSent = new Paragraph();
35          tbSent.Document = new FlowDocument(ParagraphSent);
36          ParagraphReceived = new Paragraph();
37          tbReceived.Document = new FlowDocument(ParagraphReceived);
38          Enable_Disable_Controls(false);
39      }
40      #endregion
41
42      #region Events
43      private void toggleDebugData_PreviewMouseDown(object sender, ↙
            ↳ MouseButtonEventArgs e)
44      {
45          if (toggleDebugData.IsChecked == false)
46          {
47              tbCmd.Text = "debugData";
48          }
49          if (toggleDebugData.IsChecked == true)
50          {
51              tbCmd.Text = "!debugData";
52          }
53
54          tbCmd_PreviewKeyDown(new object(), new ↙
                ↳ KeyEventArgs(Keyboard.PrimaryDevice, ↙
                ↳ Keyboard.PrimaryDevice.ActiveSource, 0, Key.Enter));
55      }
56      private void BluetoothPort_Disposed(object sender, EventArgs e)
57      {
58          if (CloseWindow)
59              Application.Current.Shutdown();
60      }
61      private void BluetoothPort_DataReceived(object sender, ↙
            ↳ SerialDataReceivedEventArgs e)
62      {
63          SerialPort temp = (SerialPort)sender;
64          string receivedData = BluetoothPort.ReadTo(";");
65          Dispatcher.Invoke(new Action(() =>
66          {
67              WriteReceived(receivedData);
68          }));
69      }
70      private void DispatcherTimer_Tick(object sender, EventArgs e)
71      {
72          togglePort.IsChecked = !togglePort.IsChecked;
73      }
74
75      private void cbPort_PreviewMouseDown(object sender, MouseButtonEventArgs e)
76      {
77          RefreshPortNames();
78      }
79      private void Window_Closing(object sender, ↙
            ↳ System.ComponentModel.CancelEventArgs e)
80      {
81          if (BluetoothPort.IsOpen)
82          {
83              CloseWindow = true;
84              e.Cancel = true;
85              BluetoothPort.Close();
86              BluetoothPort.Dispose();
87          }
88      }
```

```
89      private void textBox_PreviewKeyDown(object sender, KeyEventArgs e)
90      {
91          if (e.Key == Key.Enter)
92                  SendData("t" + textBox.Text);
93
94      }
95      private void cbPort_SelectionChanged(object sender, ↙
            ↳ SelectionChangedEventArgs e)
96      {
97          if (cbPort.SelectedItem != null && !BluetoothPort.IsOpen)
98              BluetoothPort.PortName = cbPort.SelectedItem.ToString();
99      }
100     private void lvDataTable_CellEditEnding(object sender, ↙
            ↳ DataGridCellEditEndingEventArgs e)
101     {
102         TextBox t = e.EditingElement as TextBox;
103         ListData[ListData[e.Row.GetIndex()].Index].Value = ↙
                ↳ Convert.ToInt32(t.Text);
104         ListData[ListData[e.Row.GetIndex()].Index].SetData(Convert.ToInt32(ListData[ListData[e
                ↳ BluetoothPort);
105     }
106     private void btnLogData_Click(object sender, RoutedEventArgs e)
107     {
108         SaveFileDialog filedialog = new SaveFileDialog();
109         filedialog.Filter = "CVS-File (*.csv)|*.csv|All Files|*.*";
110         if ((bool)filedialog.ShowDialog())
111         {
112             using (Writer = new StreamWriter(filedialog.FileName))
113             {
114                 CsvWriter.WriteDataToCsv(LogData, Writer);
115             }
116             HelpWrite("DebugVariables successfully saved to log.csv*g;");
117             LogData.Clear();
118         }
119     }
120     private void tbCmd_PreviewKeyDown(object sender, KeyEventArgs e)
121     {
122         switch (e.Key)
123         {
124             case Key.Enter:
125                 if (tbCmd.Text == "clear sent")
126                 {
127                     tbCmd.Clear();
128                     ParagraphSent.Inlines.Clear();
129                     return;
130                 }
131                 else if (tbCmd.Text == "clear received")
132                 {
133                     tbCmd.Clear();
134                     ParagraphReceived.Inlines.Clear();
135                     return;
136                 }
137                 else if (tbCmd.Text == "clear")
138                 {
139                     tbCmd.Clear();
140                     ParagraphSent.Inlines.Clear();
141                     ParagraphReceived.Inlines.Clear();
142                     return;
143                 }
144                 if (tbCmd.Text == "help")
145                 {
146                     HelpWrite("\nhelp -- return the entire protocol*g\nclear ↙
```

```
                        ↳ -- clears the entire history*g\nclear received -- ↙
                        ↳ clears the received history*g\nclear sent -- clears ↙
                        ↳ the sent history*g\n\ndebug possibilities (Every ↙
                        ↳ frequent cmd can be inverted with ↙
                        ↳ '!'):*g\ngsm*a\ngps*b\nmpu*c\nservo*d\nbldc*e\ncontroller*f\ndebugDa
                        ↳ -- Debugs added variables frequently*g;\n");
147                     tbCmd.Clear();
148                     return;
149                 }
150                 else if (tbCmd.Text.Contains("gsm")) //GSM, GPS, MPU, SERVO, ↙
                        ↳ BLDC, CONTROLLER, GLOBAL
151                     tbCmd.Text += "*a";
152                 else if (tbCmd.Text.Contains("gps"))
153                     tbCmd.Text += "*b";
154                 else if (tbCmd.Text.Contains("mpu"))
155                     tbCmd.Text += "*c";
156                 else if (tbCmd.Text.Contains("servo"))
157                     tbCmd.Text += "*d";
158                 else if (tbCmd.Text.Contains("bldc"))
159                     tbCmd.Text += "*e";
160                 else if (tbCmd.Text.Contains("controller"))
161                     tbCmd.Text += "*f";
162                 else if (tbCmd.Text.Contains("debugData"))
163                     tbCmd.Text += "*g";
164                 else
165                 {
166                     tbCmd.Clear();
167                     HelpWrite("Unkown Command Error 404. For help write ↙
                            ↳ \"help\"");
168                     return;
169                 }
170                 SendDataAndApplyToTextBox(TextSelection.ConvertToTextSelection(tbCmd.Text));
171                 tbCmd.Clear();
172                 History.Pointer = 0;
173                 break;
174             case Key.Up:
175                 History.Pointer++;
176                 tbCmd.Text = TextSelection.GetText(History.GetCmd());
177                 tbCmd.CaretIndex = tbCmd.Text.Length;
178                 break;
179             case Key.Down:
180                 History.Pointer--;
181                 tbCmd.Text = TextSelection.GetText(History.GetCmd());
182                 tbCmd.CaretIndex = tbCmd.Text.Length;
183                 break;
184         }
185     }
186     private void togglePort_Click(object sender, RoutedEventArgs e)
187     {
188         if (!DispatcherTimerAnimation.IsEnabled && togglePort.IsChecked == true)
189         {
190             DispatcherTimerAnimation.Start();
191             Enable_Disable_Controls(false);
192             lbPortStatus.Content = "Connecting...";
193             #region AntiLag
194             AntiLag = new Thread(() =>
195             {
196                 try
197                 {
198                     BluetoothPort.Open();
199                 }
200                 catch
```

```
201                 {
202                     DispatcherFunction(false, false);
203                     Dispatcher.Invoke(new Action(() =>
204                     {
205                         WriteReceived("Connecting failed... pls try again.*g;");
206                         lbPortStatus.Content = "Not Connected";
207                         DispatcherTimerAnimation.Stop();
208                     }));
209                     return;
210                 }
211                 DispatcherTimerAnimation.Stop();
212 
213                 DispatcherFunction(true, true);
214                 Dispatcher.Invoke(new Action(() =>
215                 {
216                     WriteReceived("Successfully Connected to " + ↙
                        ↳ BluetoothPort.PortName + " with " + ↙
                        ↳ BluetoothPort.BaudRate.ToString() + " Baud*g;");
217                     lbPortStatus.Content = "Connected";
218                 }));
219                 AntiLag.Abort();
220             });
221             #endregion
222             AntiLag.Start();
223         }
224 
225         if (!DispatcherTimerAnimation.IsEnabled && togglePort.IsChecked == ↙
                ↳ false)
226         {
227             BluetoothPort.Close();
228             lbPortStatus.Content = "Not Connected";
229             DispatcherFunction(false, false);
230         }
231     }
232 
233     #endregion
234 
235     #region Methods
236     public void RefreshPortNames()
237     {
238         cbPort.Items.Clear();
239         string[] ports = SerialPort.GetPortNames();
240         for (int i = ports.Length - 1; i >= 0; i--)
241             cbPort.Items.Insert(0, ports[i]);
242     }
243     void SendData(string data)
244     {
245         BluetoothPort.WriteLine(data);
246     }
247     void Enable_Disable_Controls(bool state)
248     {
249         tbCmd.IsEnabled = state;
250         tbReceived.IsEnabled = state;
251         tbSent.IsEnabled = state;
252         cbPort.IsEnabled = !state;
253         textBox.IsEnabled = state;
254         toggleDebugData.IsEnabled = state;
255         btnLogData.IsEnabled = state;
256     }
257     void HelpWrite(string text)
258     {
259         string[] array = text.Split('\n');
```

```
260             for (int i = 0; i < array.Length; i++)
261                 WriteReceived(array[i]);
262         }
263         void DispatcherFunction(bool toggle, bool EnableDisable)
264         {
265             DispatcherTimerAnimation.Stop();
266             Dispatcher.Invoke(new Action(() =>
267             {
268                 SetTogglePortState(toggle);
269                 Enable_Disable_Controls(EnableDisable);
270             }));
271         }
272         void SetTogglePortState(bool state)
273         {
274             togglePort.IsChecked = state;
275         }
276         void SendDataAndApplyToTextBox(TextSelection data)
277         {
278             SendData(TextSelection.GetText(data));
279             History.AddToHistory(data);
280             ParagraphSent.Inlines.Add(new Bold(new Run(data.Text + "\n"))
281             {
282                 Foreground = new SolidColorBrush(data.Colour)
283             });
284         }
285         void WriteReceived(string data)
286         {
287             if (data.Length != 0)
288             {
289                 if (data[0] == '?')
290                 {
291                     ListData.Clear();
292                     lvDataTable.ItemsSource = ListData;
293                     return;
294                 }
295                 else if (data[0] == '$')
296                 {
297                     string[] variables = data.Split('$');
298                     for (int i = 1; i < variables.Length; i++)
299                     {
300                         int dataIndex = 0;
301                         string name = "";
302                         if (int.TryParse(variables[i].Split(':')[0][1].ToString(), ↙
                                ↳ out dataIndex))
303                         {
304                             dataIndex = ↙
                                    ↳ (int)char.GetNumericValue(variables[i].Split(':')[0][0]) ↙
                                    ↳ * 10;
305                             dataIndex += ↙
                                    ↳ (int)char.GetNumericValue(variables[i].Split(':')[0][1]);
306                         }
307                         else
308                             dataIndex = ↙
                                    ↳ (int)char.GetNumericValue(variables[i].Split(':')[0][0]);
309                         if (dataIndex >= 10)
310                             name = variables[i].Split(':')[0].Remove(0, 2);
311                         else
312                             name = variables[i].Split(':')[0].Remove(0, 1);
313 
314                         int value = ↙
                                ↳ Convert.ToInt32((variables[i].Split(':')[1].Split('*')[0]));
315 
```

```
316                         if (ListData.Count == 0 || ListData.Count - 1 < dataIndex)
317                             ListData.Add(new DebugData(name, value, dataIndex));
318                         else
319                             ListData[dataIndex].Value = value;
320                         while (LogData.Count < ListData.Count)
321                             LogData.Add(new CsvCollumn());
322 
323                         if (LogData[dataIndex].StringData.Count == 0)
324                         {
325                             LogData[dataIndex].StringData.Add(name);
326                             LogData[dataIndex].StringData.Add(value.ToString());
327                         }
328                         else
329                             LogData[dataIndex].StringData.Add(value.ToString());
330                     }
331                     lvDataTable.ItemsSource = ListData;
332                     try
333                     {
334                         Dispatcher.Invoke(new Action(() =>
335                         {
336                             lvDataTable.Items.Refresh();
337                         }));
338                     }
339                     catch
340                     {
341                         return;
342                     }
343                     return;
344                 }
345             }
346 
347                 TextSelection temp = TextSelection.ConvertToTextSelection(data);
348             if (temp.Text.Length > 0)
349             {
350                 ParagraphReceived.Inlines.Add(new Bold(new Run(temp.Text + "\n"))
351                 {
352                     Foreground = new SolidColorBrush(temp.Colour)
353                 });
354                 tbReceived.ScrollToEnd();
355             }
356 
357         }
358 
359 
360 
361         #endregion
362 
363     }
```

Die Klasse MainWindow (siehe Abbildung 50) hat folgende Member:

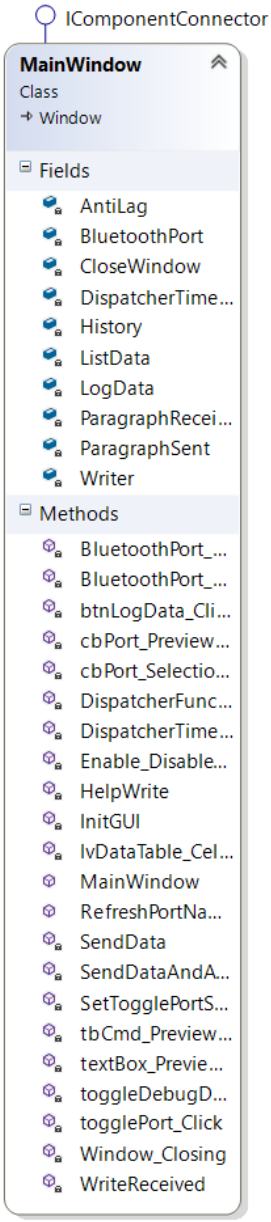


Abbildung 50: Klassendiagramm: MainWindow

Objekte und Felder

Klasse: MainWindow

Das Objekt *BluetoothPort* ist ein serieller Ein- Ausgang, welcher verwendet wird, um Daten zu senden und zu empfangen. Die SerialPort Klasse befindet sich im Namesraum System.IO.Ports.

Die Instanz *History* der Klasse CmdHistory wird benötigt, um den Kommandoverlauf (siehe 7.2.3.1) zu Verfügung zu stellen.

Die beiden Instanzen der Klasse Paragraph (*ParagraphSent* und *ParagraphReceived*), werden benötigt, um den Empfangsverlauf bzw. Sendeverlauf beschreiben zu können.

Das Objekt *DispatcherTimerAnimation*, welche eine Instanz der Klasse System.Windows.Threading.DispatcherT

ist, wird für die Animation des Steuerelementes ToggleSwitch benötigt.

Dieselbe Funktion hat auch das Objekt *AntiLag*, welches eine Instanz von System.Threading.Thread ist.

Die Liste List<DebugData> *ListData* wird benötigt, damit die Daten, welche über Bluetooth empfangen werden, in einem DataGrid (siehe 7.2.2.6) dargestellt werden können.

Das Feld *LogData*, welches eine Liste aus CsvCollumn-Objekten ist, enthält die Variablen und deren Werte, welche über die Zeit über Bluetooth empfangen werden.

Die StreamWriter Klasse und deren Instanz *writer* enthält alle notwendigen Methoden, um Daten in Dateien schreiben und diese abspeichern zu können.

Das Feld *CloseWindow* ist ein bool und wird beim Schließen der Anwendung verwendet.

Initialisierungen
Klasse: MainWindow

Im Konstruktor dieser Klassen werden erstens die beiden Listen *LogData* und *ListData* erzeugt. Zweitens wird der serielle Port mit einer Baudrate von 57600, keinem Paritätsbit, einer Wortlänge von 8 Bit und keinem Stoppbit initialisiert. Von diesem Port werden die beiden Events *BluetoothPort_ DataReceived* und *BluetoothPort_ Disposed* abonniert. Als drittes wird der DispatcherTimerAnimation mit einer Periodendauer von "300 ms" erzeugt und das Event *DispatcherTimer_ Tick* abonniert.

RefreshPortNames()
Klasse: MainWindow

Diese Funktion aktualisiert die verfügbaren seriellen Porteinträge in der Portauswahl.

SendData(string data)
Klasse: MainWindow

Die Methode *SendData* sendet die Daten, die über den Parameter *data* mitgegeben werden, über den seriellen Port. Dabei wird die Methode *WriteLine* des Objektes *BluetoothPort* verwendet. Diese fügt der Zeichenkette am Ende noch ein "Line Feed" an.

Events
Klasse: MainWindow

Enable_Disable_Controls(bool state)
Klasse: MainWindow

Diese Methode deaktiviert bzw. aktiviert alle Steuerelemente die erst verwendet werden dürfen, wenn eine geöffnete Verbindung über den seriellen Port besteht.

HelpWrite(string text)
Klasse: MainWindow

Die *HelpWrite* Methode eröffnet die Möglichkeit Daten, die als Parameter mitgegeben werden, weiterzugeben, sodass es den Anschein hat, dass die Daten über den seriellen Port empfangen wurden.

DispatcherFunction(bool toggle, bool EnableDisable)
Klasse: MainWindow

Da der *DispatcherTimerAnimation* in einem anderen Thread läuft, müssen Zugriffe auf die Steuerelemente, welche vom Timer ausgehen, über Dispatcher.Invoke durchgeführt werden. Die Methode *DispatcherFunction* erfüllt genau diese Funktion, indem Steuerelemente über die Methode *HelpWrite* aktiviert bzw. deaktiviert werden.

SendDataAndApplyToTextBox(TextSelection data)
Klasse: MainWindow

Diese Methode versendet eine TextSelection (siehe 7.2.3.2), welche über die Funktion .*GetText* zu einer Zeichenkette konvertiert wird, mit der Methode *SendData*. Danach wird dieselbe Zeichenkette der Eigenschaft Text des Steuerelementes Sendeverlauf mit entsprechender Farbe (siehe Tabelle 15) angehängt.

WriteReceived(string data)
Klasse: MainWindow

Der eigentliche Zweck der Methode *WriteReceived* ist es die empfangenen Daten in den Steuerelementen entsprechend darzustellen. Allerdings teilen sich die Daten auf Datentabelle und Empfangsverlauf auf. Die Daten, welche mit "Debuggen mittels Zeiger" (siehe 6.8.1.1) gedebugt werden, sind in der Datentabelle dargestellt und alle anderen im Empfangsverlauf.

Beim "Debuggen mittels Zeiger" sendet der µC die Daten, mithilfe des Bluetooth-Moduls, in regelmäßigen Zeitabständen. Diese Datenaktualisierung beginnt jeweils mit einem "?" (siehe 6.8.2.4). Wenn dies der Fall ist, wird zuerst der Inhalt von *ListData* gelöscht. Da jede Variable von der anderen durch ein "$" getrennt ist, wird nun mit diesem Zeichen separiert und die gesamte Zeichenkette mit einer Schleife durchgegangen. Nun werden Index, Titel und Wert der Variable herausgefiltert.

Der Index ist vom Wert durch ein ":" getrennt und ist von der entstehenden Folge das erste und/oder das zweite Zeichen (siehe Codeabschnitt 30). Da unbekannt ist, ob der Index zweistellig ist, wird versucht, ob das zweite Zeichen mit der Methode int.*TryParse* in eine Zahl konvertierbar ist. Danach wird entsprechend darauf reagiert.

Der Titel ist der erste Teil der Zeichenkette, welche mit dem Zeichen ":" separiert wurde, ausgenommen der ermittelten Indexzeichen. Diese müssen demzufolge entfernt werden.

Der Wert ist der zweite Teil der Zeichenkette. Diese muss allerdings noch einmal mit dem Zeichen "*" getrennt werden, weil sich am Ende der Kette die Kategorieinformation befindet. Von dem Array, welches sich aus dem Separieren ergibt, ist nun der erste Teil der Wert. Diese Zeichen müssen nun nur noch in eine Zahl umgewandelt werden.

Index, Titel und Wert werden nun als ein DebugData Objekt abgespeichert, welches sich wiederum in der Liste *ListData* befindet. Da alle Werte mitaufgezeichnet werden, wird der Liste *LogData* der neue Wert hinzugefügt. Die Datentabelle ist mit der Liste *ListData* verknüpft und übernimmt deren Werte. Deshalb muss die Datentabelle noch aktualisiert werden. Dies muss über Dispatcher.Invoke durchgeführt werden, da die *WriteReceived* Methode im *BluetoothPort_Received* Event keinen Zugriff auf den Thread hat, in dem sich die Datentabelle befindet. Wenn die Daten kein spezielles Indentifizierungszeichen beinhalten, werden die Daten mit entsprechender Farbe (siehe Tabelle 15) in den Empfangsverlauf geschrieben.

toggleDebugData_PreviewMouseDown

Klasse: MainWindow

Dieses Event tritt auf, wenn mit der Maus auf den ToggleSwitch, welcher den Debugmodus verändert, geklickt wird. Je nach dem ob dieser nun aktiviert oder deaktiviert wird, folgt daraus das Kommando, welches den Debugmodus aktiviert "debugData" oder deaktiviert "!debugData" (siehe Tabelle 14). Dieses Kommando wird in die Kommandozeile geschrieben und mit einem virtuellen *tbCmd_PreviewKeyDown* Event an jenes Event weitergegeben.

BluetoothPort_Disposed
Klasse: MainWindow

Das Event *BluetoothPort_Disposed* tritt auf wenn das Objekt verworfen wird. Wenn dies geschieht, kann die Applikation zum Schließen freigegeben werden.

BluetoothPort_DataReceived
Klasse: MainWindow

Das Event *BluetoothPort_DataReceived* tritt auf, wenn Daten über den seriellen Port empfangen werden. Die Daten, die empfangen wurden, sind im object *sender* enthalten. Dieses wird daraufhin in ein SerialPort Objekt umgewandelt. Nun werden die empfangenen Daten immer bis zum nächsten Semikolon ausgelesen, weil jeder Befehl bzw. jede Nachricht mit diesem Zeichen abgeschlossen wird. Nun wird der ausgelesene Teil an die Methode *WriteReceived* weitergegeben, welche die Daten auswertet.

DispatcherTimer_Tick
Klasse: MainWindow

Wenn der Timer gestartet wird, vergeht die Periodendauer und dieses Event tritt auf. In diesem wird das Steuerelement ToggleSwitch, welches die Bluetoothverbindung öffnet bzw. schließt permanent verändert. Dieses Steuerelement, wechselt infolge davon ständig den Zustand, was das Verbindungsprozedere für den Anwender klar sichtbar macht.

cbPort_PreviewMouseDown
Klasse: MainWindow

Wenn das Steuerelement ComboBox mit der Maus betätigt wird, werden die verfügbaren Ports mit der Methode RefreshPortNames (siehe 7.2.3.6) aktualisiert.

Window_Closing
Klasse: MainWindow

Dieses Event tritt auf, wenn der Benutzer das Fenster schließen will. Doch bevor die Anwendung geschlossen werden kann, muss sicher gestellt werden, dass der serielle Port geschlossen ist. Wenn der Port schon geschlossen wurde, wird die Applikation normal geschlossen. Wenn hingegen der Port noch offen ist, wird der Port geschlossen und verworfen und der Schließvorgang abgebrochen. Das Verwerfen des Ports ruft das *BluetoothPort_Disposed* Event auf und die Anwendung wird geschlossen.

textBox_PreviewKeyDown
Klasse: MainWindow

Wenn im Steuerelement der Aktualisierungszeit eine Taste auf der Tastatur betätigt wird, dann tritt dieses Event auf. Wenn die betätigte Taste *Enter* war, dann wird der eingegebenen Zeit ein "t" vorne angefügt. Diese Zeichenkette wird mit der Methode *SendData* (siehe 7.2.3.6) versen-

det. Dies hat zur Folge, dass die Aktualisierungszeit auf dem µC entsprechend verändert wird und somit Debugdaten mit einem anderen Periodendauer versendet werden.

cbPort_SelectionChanged
Klasse: MainWindow

Bei einer Veränderung der Portauswahl tritt dieses Event auf. Diese Veränderung wird ebenso an dem *BluetoothPort* Objekt durchgeführt.

lvDataTable_CellEditEnding
Klasse: MainWindow

Dieses Event tritt auf wenn Einträge der Datentabelle verändert wurden. Zuallererst wird der alte Wert in der Liste *ListData* aktualisiert. Anschließend wird die veränderte Variable inklusive, mit der Methode *SetData* der Klasse DebugData, an den µC versendet.

btnLogData_Click
Klasse: MainWindow

Wenn der Button zum Speichern der Debugdaten betätigt wird, dann tritt dieses Event auf. In diesem Event wird eine Instanz der Klasse SaveFileDialog erstellt. Die erwähnte Klasse befindet sich im Namesraum Microsoft.Win32. Dieses Objekt ermöglicht es, ein komfortables Abspeicherprozedere zu bieten. Wenn der Dialog erfolgreich beendet wurde, dann folgt das Auslesen des Pfades und Dateinames. Anschließend werden die Daten mit einer Instanz der Klasse System.IO.StreamWriter über die statische Methode CsvWriter.*WriteDataToCsv* in der gewünschten Datei abgespeichert. Als Rückmeldung für den Benutzer wird noch eine kurze Bestätigungsnachricht mit der Methode *HelpWriter* in den Empfangsverlauf geschrieben und die Inhalte der Liste LogData gelöscht.

tbCmd_PreviewKeyDown
Klasse: MainWindow

Wenn im Steuerelement Kommandozeile einê Taste betätigt wird, folgt darauf dieses Event. Wenn diese Taste *Enter* ist, dann wird folglich die eingegebene Zeichenkette auf Übereinstimmungen überprüft. Wenn der Inhalt der Kommandozeile davon "clear sent", "clear received" oder "clear" ist, wird folgendes durchgeführt. Im Falle von "clear sent" wird der Inhalt des Sendeverlaufs gelöscht. Wenn es mit "clear received" übereinstimmt, dann wird der Inhalt des Empfangsverlaufs gelöscht. Und im Falle von "clear" wird der Inhalt von beiden Verläufen gelöscht. Wenn der Text in der Kommandozeile mit "help" übereinstimmt, dann wird die vordefinierte Nachricht in den Empfangsverlauf geschrieben. Im Falle der anderen Befehle des Bluetooth-Protokolls (siehe Tabelle 14), welche noch nicht behandelt wurden, wird jeweils die entsprechende eingegebene Nachricht an den µC gesendet.

Wenn anstatt "Enter", die Tasten "Pfeil hoch" bzw. "Pfeil runter" betätigt werden, dann wird der *History.Pointer* inkrementiert bzw. dekrementiert, der entsprechende Befehl mit der Methode *History.GetText* ausgelesen und mit der statischen Methode TextSelection.*GetText* in eine Zeichenkette umgewandelt.

togglePort_Click
Klasse: MainWindow

Dieses Event tritt auf, wenn der ToggleSwitch, der die Bluetoothverbindung öffnet, betätigt wird.

In diesem Event wird ein neuer Thread definiert. Dieser Thread versucht zuerst den Port zu öffnen. Wenn dies nicht gelingt, wird eine Fehlermeldung in den Empfangsverlauf geschrieben und der Timer gestoppt. Wenn das Öffnen erfolgreich verläuft, werden die Steuerelemente mit der Methode *DispatcherFunction* aktiviert und eine Bestätigungsnachricht in den Empfangsverlauf geschrieben. Am Ende bricht der Thread sich selber ab.

Wenn also das Event auftritt, wird der Status des ToggleSwitch geprüft. Wenn dieser auf true ist und der Timer noch nicht gestartet worden ist, wird der Timer gestartet. Das bedeutet, dass die Animation begonnen hat. Da das Öffnen des Ports nicht im selben Thread stattfinden kann, wird der zuvor definierte Thread, welcher den Bluetooth-Port öffnet, gestartet.

Wenn der Status des ToggleSwitch false und der Timer nicht am Laufen ist, dann wird der Port geschlossen und die Steuerelemente deaktiviert.

8 Mechanischer Aufbau

8.1 Erste Überlegungen

In diesem Abschnitt sollen jene Überlegungen, welche dem Zweck dienen, die an die Mechanik gestellten Anforderungen mit den uns zur Verfügung stehenden Mitteln in Einklang zu bringen, in allgemeiner Weise dargelegt werden.

Die Mechanik des Luftschiffs kann im Wesentlichen in zwei Hautpkomponenten aufgeteilt werden, der Hülle und der Gondel. Die Komponente *Hülle* umfasst zum einen den Gaskörper an sich, sowie sämtliche Leitwerke. Ihre Funktionen liegen unter anderem in der Lieferung des notwendigen Auftriebs, sowie in der Stabilisierung des Luftschiffs. Die Komponente *Gondel* umfasst die Leiterplatte, den Akku und sämtliche Aktoren. Sie ist für die Manövrierung des Luftschiffes verantwortlich.

Die einzelnen Unterkomponenten werden zum einen selbst erstellt, entworfen, zum anderen in fertiger Form gekauft. Welche der beiden Optionen gewählt wird, hängt von der Komplexität der Komponente und deren Preis ab. So werden die Motoren als Ganzes gekauft, ebenso der Akkumulator. Ihr Realisierungsaufwand würde deren Kaufpreis bei Weitem übersteigen. Die Hülle stellt, wie bereits im Pflichtenheft erwähnt, eine kritische Komponente dar, weil die Komplexität ihrer Fertigung geringen Vorkenntnissen davon gegenüberstehen. Detailliertere Informationen finden sich ab Abschnitt 8.3.1.

8.2 Planung

Dieser Abschnitt umfasst spezifischere Überlegungen zu den einzelnen Unterkomponenten.

8.2.1 Gaskörper

Da der Bau einer Hülle mit den für dieses Projekt notwendigen Anforderungen recht komplex ist, wird zunächst überlegt, eine solche über das Internet zu erwerben. Dies ist beim Versuch geblieben, weil sich sämtliche Angebote entweder als zu teuer, zu unseriös oder den Anforderungen nicht gerecht erwiesen. Aus diesem Grund wird beschlossen, die Hülle selbst zu fertigen. Zur Herstellung wird eine Spezialfolie[19] verwendet, welche mittels Bügeleisen "verschweißt" werden kann.

Die Fertigung der Hülle soll wie folgt ablaufen: Von der erwähnten Spezialfolie werden zwei gleich lange Streifen abgeschnitten. Diese werden auf einem geeigneten Untergrund deckungsgleich übereinander gelegt und dann fixiert. Über diese beiden Streifen wird wiederum eine Schablone gelegt, welche die Kontur des Gaskörpers festlegt. Dieser Kontur wird mit dem vorgeheizten Bügeleisen vorsichtig nachgefahren. Dabei sollte das Bügeleisen möglichst selten abgesetzt werden. Nach Beendigung des Schweißvorgangs kann der Gaskörper ausgeschnitten werden. Beim Ausschneiden sollte ein Sicherheitsabstand von etwa 0.5 cm zur Schweißnaht gewahrt werden.

Im Prinzip ließen sich auch mehr als zwei Folienstreifen zur einer Hülle zusammenschweißen. Diese hätte dann zwar eine rundere Form, jedoch ist die Fertigung eines solchen Gaskörpers ungemein schwieriger, weshalb im Zuge dieses Projekts alle Hüllen aus zwei Folienstreifen gefertigt

[19] Windreiter [19]

werden.

8.2.2 Finnen

Die Finnen haben den Zweck, den Flug des Luftschiffs zu stabilisieren. Sie sollen aus einem, dennoch stabilen Material bestehen. Dieses wurde dem Projektteam vom Modellflugsportverein Rheintal empfohlen. Die Finnen werden an der gefüllten Hülle angeklebt.

8.2.3 Achse

An der Achse, welche sich unter dem Luftschiffs befindet, werden jeweils an beiden äusseren Seiten die BLDC-Motoren montiert. In der Mitte der Achse befindet sich ein Zahnrad, welches durch eine 1:1 Übersetzung mit dem Servo-Motor gekoppelt ist. Der Servomotor hat den Nutzen, die Achse von -90° bis +90° zu drehen und somit die Schubkraft der Motoren in verschiedene Richtungen zu lenken.

8.2.4 Rotoren

Als Rotoren werden in diesem Kontext [20] der Verbund zwischen Propeller und Rotor bezeichnet. Beide Unterkomponenten werden erworben, wobei auf die Spezifikationen zu achten ist. Die Anforderungen an beide Unterkomponenten werden im Folgenden beschrieben.

8.2.4.1 Motoren

Die Motoren sollen leicht und leistungsstark sein. Ihre Ansteuerung darf keine schweren Kühlkörper benötigen, des Weiteren sollen sie einen hohen Wirkungsgrad besitzen.

8.2.4.2 Propeller

Mittels der Propeller wird die von den Motoren erzeugte Kraft derart auf die umliegende Luft übertragen, dass sich gemäß Netwons 3. Axiom sich ein Schub in die gewünschte Richtung einstellt.[4]

8.2.5 mögliche Konfigurationen der Aktoren

Die verwendeten Komponenten müssen auf bestimmte Art und Weise zusammengefügt werden. Dazu sind verschiedene Konfiguratonsvorschläge ausgearbeitet worden, welche im Folgenden erläutert werden.

8.2.5.1 1. Konfigurationsvorschlag

Um den Mittelpunkt des Gaskörpers wird ein stabiler, leichter Ring angebracht. An diesem sollen in einem Winkel von 120° zueinander drei Rotoren angebracht. Zwei Rotoren sind schwenkbar und befinden sich auf der Unterseite des Luftschiffs, die Gondel flankierend. Sie sollen Steig-Sink und Drehmanöver ermöglichen. Der dritte Rotor befindet sich dementsprechend auf der

[20] In Abweichung von der offiziellen Definition

Oberseite des Gaskörpers. Dieser soll den Kippeffekt, welcher kurz im 2. Konfigurationsvorschlag beschrieben wird, ausgleichen.

Mittels dieses Konfiguration wäre das Luftschiff in der Lage, sowohl auf der Stelle zu drehen als auch den Nickwinkel zu kontollieren. Das Luftschiff besäße die größte Manövrierbarkeit. Dem gegenüber steht allerdings ein höherer Realisierungsaufwand, vor allem, was die Herstellung des erwähnten Rings betrifft. Deshalb wird nach einem anderen Konfigurationsvorschlag gesucht.

8.2.5.2 2. Konfigurationsvorschlag

Zwei Rotoren werden an einer drehbaren Achse befestigt seitlich an der Gondel angebracht. Diese Rotoren sollen dem Luftschiff Steig- und Sinkmanöver sowie den Vorwärtsflug ermöglichen. Ein Rotor soll am hinteren Ende des Luftschiffs, zur Seite wegzeigend, befestigt werden. Jener Rotor soll dem Luftschiff eine Drehung um dessen Gierachse im Stand und somit einen Kurvenflug ermöglichen.

Da die ersten beiden Rotoren dieses Konfigurationsvorschlag nicht auf Höhe des Mittelpunkts des Luftschiffs liegen, kann das Drehmoment um die Nickachse, welche durch die Reibungskräfte, auf den Gaskörper wirkend, verursacht werden, nicht durch dieses Rotorenpaar ausgeglichen werden. Dies resultiert in eine Zunahme des Nickwinkels, welche so lange andauert, bis die Wirkungslinie zwischen dem Schwerpunkt des Luftschiffs und dem Auftriebspunkt, der Mittelpunkt des Gaskörpers, groß genug wird, sodass das dadurch erzeugte Drehmoment alle anderen Drehmomente um die Nickachse aufhebt. Salopp gesprochen, das Luftschiff stellt sich während des Flugs auf. Dieser Effekt ließe sich möglicherweise durch eine komplexe Regelung zumindest teilweise aufheben. Im Nachfolgendem wird jedoch ein Konfigurationsvorschlag dargelegt, mit dem sich dieses Problem auf einfachere Weise beseitigen lässt.

8.2.5.3 3. Konfigurationsvorschlag

Der 3. Konfigurationsvorschlag deckt sich größtenteils mit dem 2. Konfigurationsvorschlag. Der wesentliche Unterschied besteht in der Lage und Ausrichtung des hinteren Rotors. Dieser ist am hinteren Drittel auf der Oberseite des Gaskörpers derart angebracht, dass die Drehachse des Propellers parallel zur Konturlinie des Gaskörpers liegt. Dadurch kann der Rotor ein Drehmoment erzeugen, dessen axialer Vektor mit der Nickachse des Luftschiffs zusammenfällt. Da es aufgrund jener Gegebenheit möglich ist, den Nickwinkel des Luftschiffs einzustellen, wird dieser Konfigurationsvorschlag schlussendlich realisiert.

Mittels dieser Konfiguraton ist es nicht möglich, das Luftschiff auf der Stelle zu drehen. Daraus folgt, dass der Kurvenradius auf einen Minimalwert beschränkt wird. Jener soll im nachfolgenden Abschnitt "Entwurf" auf Seite **??** überschlagsweise berechnet werden.

8.2.5.4 4. Konfigurationsvorschlag

Der 4. Konfiguratonsvorschlag hat ebenfalls viele Gemeinsamkeiten mit 2. Konfigurationsvorschlag. Anstelle der drei Rotoren besitzt das Luftschiff in dieser Konfiguration nur 2 Rotoren, ebenfalls schwenkbar an der Seite des Luftschiffs montiert. Mittels eines an der Unterseite der Gondel montierten schwenkbaren Akkus wird der Schwerpunkt des Luftschiffs und somit dessen Nickwinkel in Ruhelage variiert. Dadurch ist es möglich, den Nickwinkel und somit den Kippeffekt zu kontrollieren.

Weil sämtliche Aktoren direkt mit der Gondel vebunden sind, wäre der Aufbau der notwendigen Testkonstruktion mit dem geringsten Aufwand verbunden. Dem gegenüber stehen eine minimale

Flughöhe, beschränkt durch die Länge des Hebelarms, sowie eine begrenzte Fluggeschwindigkeit, weil die durch die Reibungskräfte erzeugten Drehmomente nur bedingt ausgeglichen werden können.

8.3 Realisierung

8.3.1 Gaskörper

Die Folienstreifen müssen, damit sie zu einem Gaskörper zusammengeschweißt werden können, gewisse Anforderungen an ihre Form erfüllen. Dies bedeutet, dass die Form nicht willkürlich gewählt werden kann und daher gezielt entworfen werden muss. Aus Zeitgründen wird zum Entwurf des Folienstreifens die Excel-Datei *Iris-Gertler4166.xlsx* herangezogen. Diese Excel-Datei besteht aus den Blättern *Parameters*, *Shape*, *Pattern* sowie *Tabelle2*.[6]

In die gelben Felder des Blatts *Parameters*, dargestellt in Abbildung 51 werden die Parameter des Gaskörpers eingegeben. Diese lauten, in absteigender Reihenfolge, wie folgt:

1. *Anzahl der Punkte:* Gibt die Anzahl der Punkte an, die zum Plotten der Streifenform verwendet werden.
2. *Anzahl der Streifen:* Gibt die Anzahl der Streifen an, aus denen der Gaskörper zusammengeschweißt wird.
3. *Längen-Durchmesser-Verhältnis:* Gibt das Verhältnis zwischen Länge und Durchmesser des Luftschiffs an.
4. *Volumen:* Gibt das Volumen des Gaskörpers an.

Coefficients			
a_6	19,621670		
a_5	-56,480030		
a_4	59,357210		
a_3	-26,960996		
a_2	3,462153		
a_1	1,000000		
	Input		
n	50	[-]	Number of Points
nG	2	[-]	Number of Gores
L2D	5,0	[-]	Length to Diameter ratio
Cp	0,70	[-]	Prismatic coefficient
Cb	0,55	[-]	Block Coefficient
Vb	1,100	[m^3]	Block Volume
V	0,605	[m^3]	Volume
L	3,018	[m]	Length
D	0,604	[m]	Diameter

Abbildung 51: Parametereingabe zum Entwurf des Gaskörpers

Im nächsten Tabellenblatt *Shape*, dargestellt in Abbildung 52, wird die schlussendliche Form des Luftschiffs dargestellt.

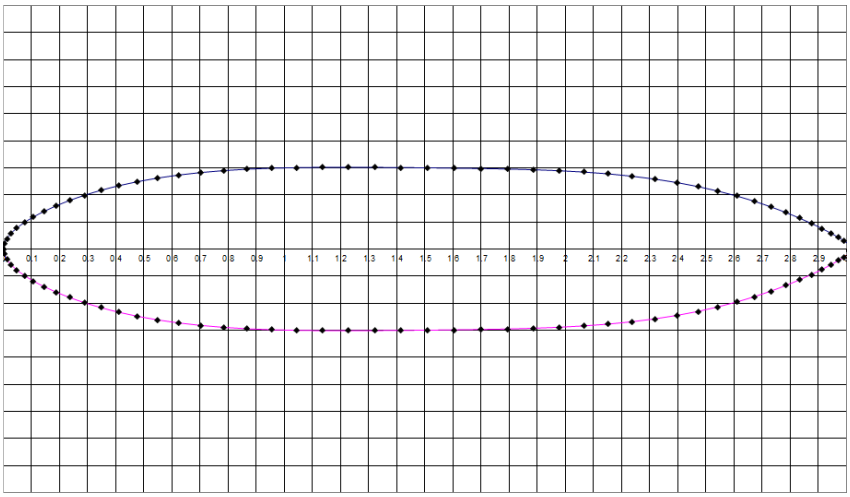


Abbildung 52: Kontur des Gaskörpers

Die Form eines Folienstreifens wird in Abbildung 53 dargestellt. Werden die beiden Konturen miteinander verglichen, fällt auf, dass der Folienstreifen breiter ist als die Kontur des Gaskörpers. Dies ist durch die Krümmung des Gaskörpers begründet, welche in Querrichtung stärker ausfällt als in Längsrichtung.

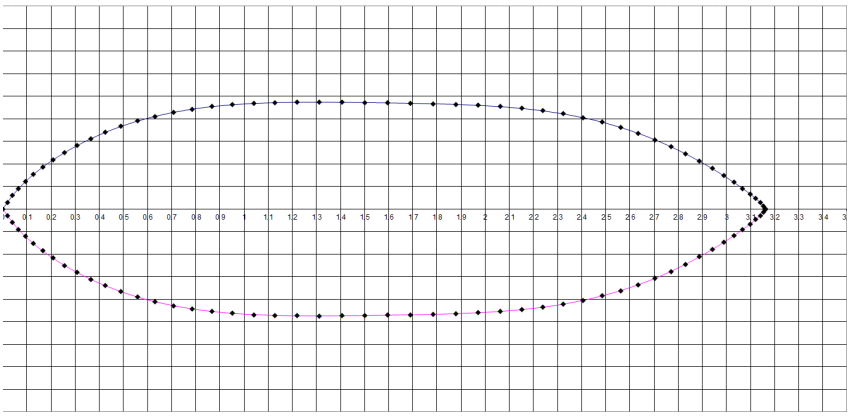


Abbildung 53: Kontur eines Folienstreifens

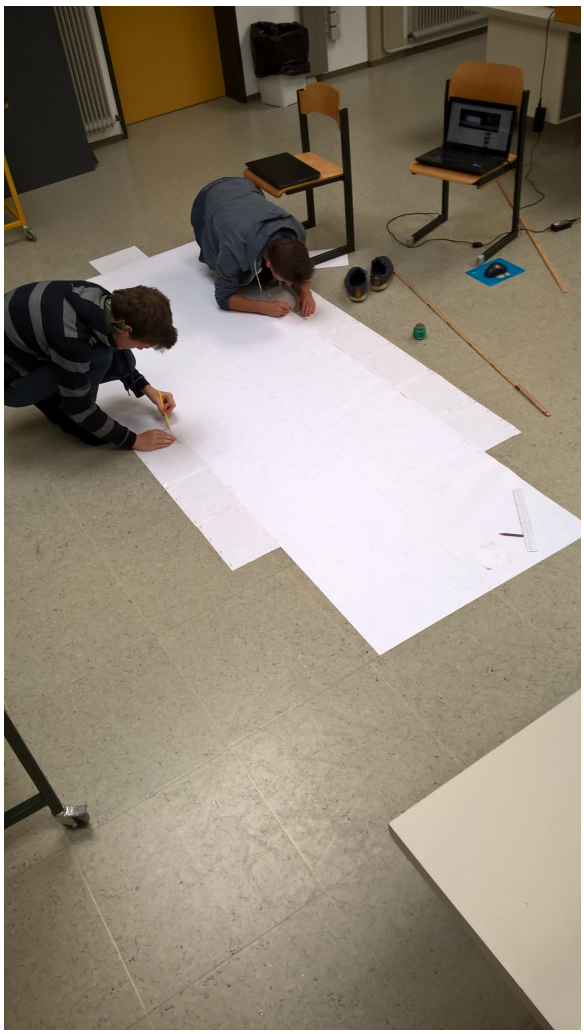


Abbildung 54: Zeichnen der Form

Abbildung 55: Bügeln der Form

Abbildung 56: Fertigung des Gaskörpers

8.3.2 Finnen

Die Finnen werden aus Depron[21] ausgeschnitten. Damit sie an den Gaskörper angebracht werden können, muss eine Kante gerade sein. Die Form der anderen Seiten wird frei gewählt, soll jedoch ästhetisch ansprechend sein.

[21] Ein sehr leichtes aber dennoch recht stabiles Material (ähnlich Styropor). Ist im Modellflugbereich stark verbreitet.

8.3.3 Achse

Die Zahnräderskizze in 2-D, welche für die Kopplung von Achse zu Servomotor gebruacht wird, ist mittels dem Gear Generator[22] erstellt worden. Die Skizze wurde mittels Blender[23] in ein 3D-Modell umgewandelt.

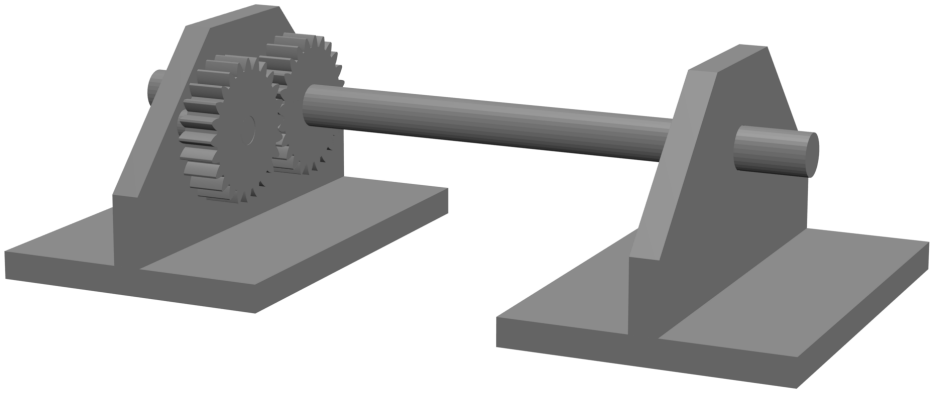


Abbildung 57: 3-D Modell der Achse mit Getriebe

Dieses Modell wird freundlicherweise von Lucas Kneissl mittels 3D-Drucker ausgedruckt. Dies ermöglicht beliebige Teile in sehr schneller Zeit und aktzeptabler Genauigkeit zu bekommen.

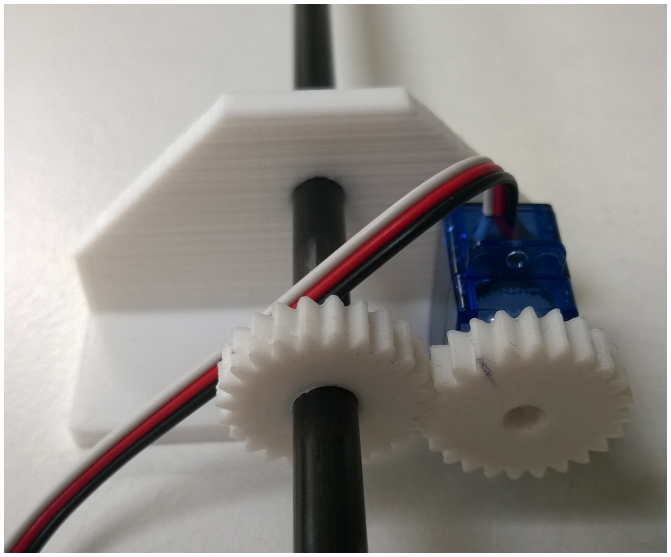


Abbildung 58: Ausgedrucktes 3-D Modell der Achse mit Getriebe

Die Achse selber besteht aus Kohlefaser, das sich als sehr stabil aber gleichzeitig unwahrscheinlich leicht erweist. Um diese noch leichter zu gestalten, wird eine hohle Kohlefaserstange gewählt. Die Teile, welche die Achse in Position halten, werden mit der flachen Seite an das Luftschiff angeklebt. Somit kann eine relativ gute Kraftubertragung von Motoren über die Achse bis zum Luftschiff gewährleistet werden.

[22] Gear Generator – Created by Abel Vincze/IPARIGRAFIKA LTD. [16]
[23] Blender – Open Source 3D Modelling [17]

8.3.4 Rotoren

8.3.4.1 Motoren

Zum Antrieb der Propeller werden BLDC-Motoren verwendet. BLDC-Motoren sind kleine Synchronmaschinen, welche ein ähnliches Regelverhalten wie die Gleichstrommaschine aufweisen. In Abbildung 59 werden die Spezifikationen eines solchen Motors dargestellt.

No.Of cells	2 X LI-Poly
Stator dimensions	22x5mm
Shaft diameter	3mm
Weight	29g/1.02oz.
Recomended model weight	200-300g
Recomended prop without gearbox	APC 7X4 APC 6X4

Model	Cell Count	RPM/V	Prop (APC)	RPM	MAX current (<60S)	Thrust
CF2805	2S	2840	6x4	13000	12A	413g 0.91lb.
			7x4	9900	14.4A	381g 0.84lb.

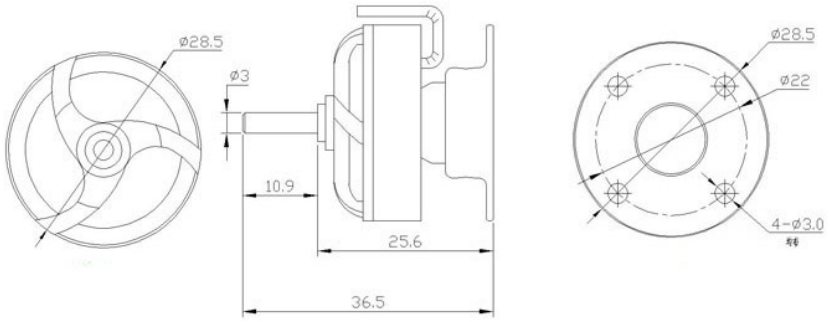


Abbildung 59: BLDC Motor Spezifikationen[18]

8.3.5 mögliche Konfigurationen der Aktoren

In diesem Unterabschnitt sollen die Gleichungen, welche die entsprechenden Konfigurationsvorschläge beschreiben, dargelegt werden. Als Grundlage zur Modellbildung wird das 2. Newtonsche Gesetz herangezogen.[4]

$$\vec{F} = m \cdot \vec{a} \tag{8.1}$$

$$\vec{M} = I \cdot \vec{\alpha} \tag{8.2}$$

Gleichung 8.1 gilt für die Translation, Gleichung 8.2 die Rotation. m sei die Masse, I der Trägheitstensor des gesamten Luftschiffs. Die Masse des Luftschiffs ist bekannt, sie kann mit

$$m = 1.2kg \tag{8.3}$$

beziffert werden. Der Trägheitstensor kann mit den zur Verfügung stehenden Kenntnissen nicht ermittelt werden. Für die Simulation wird er mit

$$I = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{8.4}$$

angenommen.

Die Kräfte, Drehmomente, welche auf das Luftschiff wirken, können in eine dynamische ($*_{dyn}$) sowie statische Komponente ($*_{stat}$) zerlegt werden, wie in **??** für die Gesamtkraft F_{ges} und in

Gleichung 8.6 für das gesamte Drehmoment dargelegt wird.

$$\vec{F}_{ges} = \vec{F}_{stat} + \vec{F}_{dyn} \tag{8.5}$$

$$\vec{M}_{ges} = \vec{M}_{stat} + \vec{F}_{stat} \tag{8.6}$$

Als statische Einflussgrößen werden bezeichnet

- *Schwerkraft:* wirkt sich auf die Gesamtkraft aus. hat keinen Einfluss auf das Drehmoment, weil das Luftschiff frei im Raum schwebt. (F_g)
- *Auftriebskraft:* wirkt sich sowohl auf die Gesamtkraft als auch auf das gesamte Drehmoment aus, weil der Auftriebspunkt nicht im Schwerpunkt liegt. (F_a, M_a)
- *Rotorenkräfte:* wirkt sich auf die Gesamtkraft und das gesamte Drehmoment aus. (F_{mot}, M_{mot})
- *Ausrichtung des Luftschiffs:* wirkt sich sowohl auf die Gesamtkraft als auch auf das gesamte Drehmoment aus. zum Einen, weil die Lage des Auftriebspunkt relativ zum Schwerpunkt in Abhängigkeit mit der Ausrichtung steht, zum Anderen, weil die Richtung der Rotorenkräfte direkt davon abhängt. (F_a, F_{mot}, M_a, M_{mot})
- *Ausrichtung der Rotoren:* wirkt sich sowohl auf die Gesamtkraft als auch auf das gesamte Drehmoment aus. (F_a, F_{mot}, M_a, M_{mot})

Als dynamische Einflussgrößen werden bezeichnet

- *Geschwindigkeit:* wirkt sich sowohl auf die Gesamtkraft als auch auf das gesamte Drehoment aus, weil die Linie zwischen dem Angriffspunkt der Luftreibung sowie dem Schwerpunkt eine vom Luftreibungsvektor verschiedene Richtung hat.
- *Rotationsgeschwindigkeit:* wirkt sich nur auf das gesamte Drehmoment aus.

Für die Gesamtkraft gilt demzufolge, sich auf Gleichung 8.7 beziehend:

$$\vec{F}_{ges} = \vec{F}_{rei}(\vec{\phi}, \vec{v}) + \vec{F}_{mot}(\vec{\phi}, \vec{F}_1, \cdots, \vec{F}_n) + \vec{F}_g + \vec{F}_a \tag{8.7}$$

Für das gesamte Drehmoment gilt, sich auf **??** beziehend:

$$\vec{M}_{ges} = \vec{M}_{rei}(p\vec{h}i, \vec{v}) + \vec{M}_{mot}(\vec{\phi}, \vec{F}_1, \cdots, \vec{F}_n) + \vec{F}_a \tag{8.8}$$

Sind $\vec{F}_{ges}$ sowie $\vec{M}_{ges}$ bekannt, kann mit Hilfe von Gleichung 8.1 sowie Gleichung 8.2 die Beschleunigung Gleichung 8.9 sowie die Winkelbeschleunigung Gleichung 8.10 ermittelt werden.

$$\vec{a}_{ges} = \frac{\vec{F}_{ges}}{m} \tag{8.9}$$

$$\vec{\alpha}_{ges} = J^{-1} \cdot \vec{M}_{ges} \tag{8.10}$$

Die Geschwindigkeit sowie die Winkelgeschwindigkeit des Luftschiffs kann dementsprechend durch einfache Integration der Beschleunigung (Gleichung 8.9) sowie der Winkelbeschleunigung (Gleichung 8.10) bestimmt werden.

$$\vec{v}_{ges} = \int \vec{a}_{ges} dt \tag{8.11}$$

$$\vec{\omega}_{ges} = \int \vec{\alpha}_{ges} dt \tag{8.12}$$

Der Ort des Luftschiffs ergibt sich aus der Integration der Geschwindigkeit (Gleichung 8.11).

$$\vec{s}_{ges} = \int \vec{v}_{ges} dt \tag{8.13}$$

Die Ausrichtung des Luftschiffs kann jedoch nicht ohne Weiteres durch Integration aus der Winkelgeschwindigkeit bestimmt werden. Dies hat folgenden Grund: Sei $\vec{v}$ der Geschwindigkeitsvektor zum Zeitpunkt t. Dann lässt sich eine infinitesimale Drehung $d\vec{phi}$ schreiben als

$$d\vec{\omega} = \vec{\omega} \cdot dt \tag{8.14}$$

Diese infinitesimale Drehung lässt sich auch als Matrix darstellen:

$$d\vec{\phi} \Leftrightarrow d\Phi \tag{8.15}$$

Die Gesamtdrehung ist dann gleich dem Wert des Produkts aller infinitesimalen Drehungen:

$$\Phi = \prod_{i=0}^{\infty} d\Phi_i \tag{8.16}$$

Die infinitesimale Rotationsmatrix Phi_i kann in ein infinitesimalen und nichtinfinitesimalen Teil zerlegt werden.

$$\Phi = \prod_{i=0}^{\infty} (E + d\Phi_i^*), \tag{8.17}$$

wobei E die $[3x3]$-Einheitsmatrix bezeichnet. Aus Gleichung 8.17 ist eine gewisse Ähnlichkeit zur Definition der Exponentialfunktion zur Basis e ersichtlich.

$$e^x = \lim_{n\infty} \left(1 + \frac{x}{n}\right)^n \tag{8.18}$$

Mit

$$d\Phi_i^* = \frac{\Phi^*}{n} \tag{8.19}$$

Kann nun das Matrixexponential definiert werden:

$$e^{\Phi} = \left(1 + \frac{\Phi^*}{n}\right)^n \tag{8.20}$$

Dieses Matrixexponential gehorcht ebenfalls den Rechenregeln für Potenzen.[10]

$$e^{\Phi_1} \cdot e^{\Phi_2} = e^{\Phi_1 + \Phi_2} \tag{8.21}$$

Die Multiplikation aller infinitesimalen Rotationsmatrizen, dargestellt in Gleichung 8.16, vereinfacht sich zu folgendem Ausdruck

$$\phi_{ges} = \prod_{i=0}^{\infty} e^{d\Phi_i^*} = e^{\Sigma_{i=0}^{\infty} d\Phi_i^*} = e^{\int_0^T \Phi^* dt} \tag{8.22}$$

Um aus dem Rotationsgeschwindigkeitsvektor $\vec{v}$ den Rotationsvektor bestimmen zu können, kann dieser zunächst in eine infinitesimale Rotationsmatrix Φ konvertiert werden. Jene ist dann laut Gleichung 8.17 in Φ^* sowie E zu zerlegen. Daraus kann dann laut **??** die Rotationsmatrix bestimmt werden, welche schließlich in den Rotationsvektor konvertiert wird.

In der Simulation wird der Rotationsvektor auf eine andere Art aus dem Rotationsgeschwindigkeitsvektor ermittelt.

8.3.5.1 1. Konfigurationsvorschlag

In diesem Abschnitt werden die Gleichungen aufgestellt, welche die Zusammenhänge zwischen den Rotorkräften und -stellungen sowie der Gesamtkraft und des gesamten Drehmoments des ersten Konfigurationsvorschlags beschrieben.

Zur Formulierung der Gleichungen werden Quaternionen zur Hilfe genommen. Mittels Quaternionen können Rotationen um eine beliebige Achse elegant formuliert werden. Sie sind eine Erweiterung der komplexen Zahlen und sind, wie in Gleichung Gleichung 8.23 erläutert, aufgebaut:[11]

$$x = \underbrace{x_0}_{Skalarer} \quad \underbrace{+ i \cdot x_1 + j \cdot x_2 + k \cdot x_3}_{Vektorieller \quad Teil} \quad Teil \tag{8.23}$$

Auf den skalaren und vektoriellen Teil wird noch später eingegangen.

Die Elemente i, j, k bilden den imaginären Teil des Quaternions und weisenfolgende Eigenschaften auf:

$$i^2 = j^2 = k^2 = i \cdot j \cdot k = -1 \tag{8.24}$$
$$i \cdot j = k \tag{8.25}$$
$$j \cdot k = i \tag{8.26}$$
$$k \cdot j = i \tag{8.27}$$

Quaternionen sind kommutativ bezüglich der Addition und nichtkommutativ bezüglich der Multiplikation, wie in Gleichung 8.28 dargestellt ist.

$$a \cdot b \neq b \cdot a \tag{8.28}$$

mit

$$a = a_0 + i \cdot a_1 + j \cdot a_2 + k \cdot a_3 \tag{8.29}$$
$$b = b_0 + i \cdot b_1 + j \cdot b_2 + k \cdot b_3 \tag{8.30}$$

Die Konjugation eines Quaternions ist ähnlich wie die Konjugation einer komplexen Zahl definiert:

$$x^* = x_0 - i \cdot x_1 - j \cdot x_2 - k \cdot x_3 \tag{8.31}$$

Mittels diesen Definitionen kann nun eine Rotation um eine beliebige Achse definiert werden. Sei $\vec{r}$ ein Einheitsvektor, welcher die Rotationsachse bezeichnet.

$$\vec{r} = \begin{pmatrix} r_x \\ r_y \\ r_z \end{pmatrix} \tag{8.32}$$
$$|\vec{r}| = 1 \tag{8.33}$$

Dann ist die Drehung eines Punktes $\vec{p}$ um diese Achse wie folgt definiert:

$$\vec{p}' = \vec{r} \cdot \vec{p} \cdot \vec{r}^* \tag{8.34}$$

mit

$$\vec{r} = cos\left(\frac{\phi}{2}\right) + i \cdot sin\left(\frac{\phi}{2}\right) + j \cdot sin\left(\frac{\phi}{2}\right) + k \cdot sin\left(\frac{\phi}{2}\right) \tag{8.35}$$

$$\vec{p} = i \cdot p_x + j \cdot p_y + k \cdot p_z \tag{8.36}$$

Der skalare Teil eines Punktes, dargestellt als eine Quaternion, ist somit gleich Null, wie in Gleichung 8.36 dargestellt wird.

Im Gegensatz dazu ist der skalare Teil der Rotation, dargestellt in Gleichung 8.35, gleich $\cos(\phi/2)$.

Nun können die Gleichungen für die Rotorenkräfte aufgestellt werden. Der Ring, an welchen die Rotoren sowie die Gondel angebracht sind (siehe **??**), liegen in der xy-Ebene. F_1 bezeichnet die Kraft des oberen Rotors, F_2 die Kraft des linken Rotors, F_3 bezeichnet die Kraft des rechten Rotors.

Die Ortsvektoren aller drei Rotoren lauten wie folgt:

$$\vec{r}_1 = r \cdot \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} \tag{8.37}$$

$$\vec{r}_2 = r \cdot \begin{pmatrix} \frac{\sqrt{3}}{2} \\ -\frac{1}{2} \\ 0 \end{pmatrix} \tag{8.38}$$

$$\vec{r}_3 = r \cdot \begin{pmatrix} -\frac{\sqrt{3}}{2} \\ -\frac{1}{2} \\ 0 \end{pmatrix} \tag{8.39}$$

Die Rotationsachsen der Rotoren fällt mit deren Ortsvektoren zusammen. In den folgenden Gleichungen werden die Rotationsachsen in Quaternionenschreibweise angegben:

$$\vec{r}_1 = j \tag{8.40}$$

$$\vec{r}_2 = i \cdot \frac{\sqrt{3}}{2} - j \cdot \frac{1}{2} \tag{8.41}$$

$$\vec{r}_3 = -i \cdot \frac{\sqrt{3}}{2} - j \cdot \frac{1}{2} \tag{8.42}$$

Damit können nun die Rotorenkräfte in Quaternionenschreibweise formuliert werden.

$$\vec{F}_1 = \left(cos\left(\frac{\phi_1}{2}\right) + j \cdot sin\left(\frac{\phi_1}{2}\right)\right) \cdot k \cdot F_1 \cdot \left(cos\left(\frac{\phi_1}{2}\right) - j \cdot sin\left(\frac{phi_1}{2}\right)\right) \tag{8.43}$$

$$\begin{aligned} \vec{F}_2 = &\left(cos\left(\frac{\phi_2}{2}\right) + i \cdot \frac{\sqrt{3}}{2} \cdot sin\left(\frac{\phi_2}{2}\right) - j \cdot \frac{1}{2} \cdot sin\left(\frac{phi_2}{2}\right)\right) \cdot k \\ &\cdot F_2 \cdot \left(cos\left(\frac{\phi_2}{2}\right) - i \cdot \frac{\sqrt{3}}{2} \cdot sin\left(\frac{\phi_2}{2}\right) + j \cdot \frac{1}{2} \cdot sin\left(\frac{\phi_2}{2}\right)\right) \end{aligned} \tag{8.44}$$

$$\begin{aligned} \vec{F}_3 = &\left(cos\left(\frac{\phi_3}{2}\right) - i \cdot \frac{\sqrt{3}}{2} \cdot sin\left(\frac{\phi_3}{2}\right) - j \cdot \frac{1}{2} \cdot sin\left(\frac{\phi_3}{2}\right)\right) \cdot k \\ &\cdot \vec{F}_3 \cdot \left(cos\left(\frac{1}{2}\right) + i \cdot \frac{\sqrt{3}}{2} \cdot sin\left(\frac{\phi_3}{2}\right) + j \cdot \frac{1}{2} \cdot sin\left(\frac{\phi_3}{2}\right)\right) \end{aligned} \tag{8.45}$$

Die Ausmultiplikation von Gleichung 8.43, Gleichung 8.44, Gleichung 8.45 ergibt

$$\vec{F}_1 = F_1 \cdot (i \cdot \sin(\phi_1) + k \cdot \cos(\phi_1)) \tag{8.46}$$

$$\vec{F}_2 = F_2 \cdot \left(-i \cdot j\frac{1}{2} \cdot \sin(\phi_2) - j \cdot \frac{\sqrt{3}}{2} \cdot \sin(\phi_2) + k \cdot \cos(\phi_2) \right) \tag{8.47}$$

$$\vec{F}_3 = F_3 \cdot \left(-\frac{1}{2} \cdot \sin(\phi_3) + \frac{\sqrt{3}}{2} \cdot \sin(\phi_3) + k \cdot \cos(\phi_3) \right) \tag{8.48}$$

Für weitere Berechnungen werden die Rotorenkräfte von der Quaternionen- in die Vektorschreibweise umgeschrieben.

$$\vec{F}_1 = F_1 \cdot \begin{pmatrix} \sin(\phi_1) \\ 0 \\ \cos(\phi_1) \end{pmatrix} \tag{8.49}$$

$$\vec{F}_2 = F_2 \cdot \begin{pmatrix} -\frac{1}{2} \cdot \sin(\phi_2) \\ -\frac{\sqrt{3}}{2} \cdot \sin(\phi_2) \\ \cos(\phi_2) \end{pmatrix} \tag{8.50}$$

$$\vec{F}_3 = F_3 \cdot \begin{pmatrix} -\frac{1}{2} \cdot \sin(\phi_3) \\ \frac{\sqrt{3}}{2} \cdot \sin(\phi_3) \\ \cos(\phi_3) \end{pmatrix} \tag{8.51}$$

In Gleichung 8.52 wird die Summe aller Rotorenkräfte dargelegt:

$$\vec{F}_{ges} = \vec{F}_1 + \vec{F}_2 + \vec{F}_3 = \begin{pmatrix} \sin(\phi_1) - \frac{1}{2} \cdot (\sin(\phi_2) + \sin(\phi_3)) \\ \frac{\sqrt{3}}{2} \cdot (-\sin(\phi_2) + \sin(\phi_3)) \\ \cos(\phi_1) + \cos(\phi_2) + \cos(\phi_3) \end{pmatrix} \tag{8.52}$$

Zur Ermittlung des gesamten von den Rotoren ausgehenden Drehmoments muss zunächst mittels des Kreuzprodukts die Drehmomente der einzelnen Rotoren bestimmt werden.

Die Summe aller dieser Drehmomente ergibt, wie in Gleichung 8.53 dargelegt wird:

$$\begin{aligned} \vec{M}_{ges} &= \vec{M}_1 + \vec{M}_2 + \vec{M}_3 \\ &= \begin{pmatrix} 0 \\ -r \cdot \cos(\phi_1) \\ 0 \end{pmatrix} + \begin{pmatrix} -\frac{1}{2} \cdot r \cdot \cos(\phi_2) \\ -\frac{\sqrt{3}}{2} \cdot r \cdot \cos(\phi_2) \\ -r \cdot \sin(\phi_2) \end{pmatrix} + \begin{pmatrix} -\frac{1}{2} \cdot r \cdot \cos(\phi_3) \\ -\frac{\sqrt{3}}{2} \cdot r \cdot \cos(\phi_3) \\ -r \cdot \sin(\phi_3) \end{pmatrix} \\ &= \begin{pmatrix} -\frac{1}{2} \cdot r \cdot (\cos(\phi_2) + \cos(\phi_3)) \\ -r \cdot \left(\cos(\phi_1) + \frac{\sqrt{3}}{2} \cdot (\cos(\phi_2) + \cos(\phi_3)) \right) \\ -r_2 \cdot \sin(\phi_2) - r_3 \cdot \sin(\phi_3) \end{pmatrix} \end{aligned} \tag{8.53}$$

In Gleichung 8.52 wird die Summe aller Rotorenkräfte, in Gleichung 8.53 die Summe aller von den Rotoren ausgehenden Drehmomente dargelegt. Weil dieser Konfiguratonsvorschlag, wie in der Skizzierung beschrieben, nicht realisiert wird, soll auf dessen weitere Ausarbeitung verzichtet werden.

8.3.5.2 Allgemeines zu den Konfigurationsvorschlägen 2 bis 4

Anmerkung: Für die Konfigurationsvorschläge 2 bis 4 wird für die Kräfte eine andere Indizierung vorgenommen.

Wie im Abschnitt "Skizzierung" beschrieben wird, haben die Konfigurationsvorschläge 2 bis 4 einige Gemeinsamkeiten. Diese sollen nachfolgend erläutert werden. In den darauf folgenden Unterabschnitten soll dann weiters auf die Konfigurationsvorschläge eingegangen werden.

Sei $(\vec{S}, \vec{A}, R, \vec{\rho})$ ein Quadrupel, welches den Blimp als System von Punkten beschreibt. Als Koordinatensystem wird ein kartesisches systembezogenes gewählt. Dabei ist

- $\vec{S}$: Schwerpunkt des Luftschiffs
- $\vec{A}$: Auftriebspunkt des Luftschiffs
- $R = (R_1, \cdots, R_n)$: n-Tupel, umfasst die Position aller n Rotoren des Luftschiffs
- $\vec{\rho}$: Angriffspunkt der Luftreibungskraft

Der Angriffspunkt der Luftreibungskraft $\vec{\rho}$ sowie der Auftriebspunkt des Luftschiffs $\vec{A}$ ist für die behandelten Konfigurationsvorschläge derselbe, ebenso die Lage des ersten Rotors R_1 sowie des zweiten Rotors R_2.

$$\vec{\rho} = \begin{pmatrix} \frac{l_H}{2} \\ 0 \\ \frac{h_H}{2} + h_G \end{pmatrix} \tag{8.54}$$

$$\vec{A} = \begin{pmatrix} 0 \\ 0 \\ \frac{h_H}{2} + h_G \end{pmatrix} \tag{8.55}$$

$$\vec{R}_1 = \begin{pmatrix} 0 \\ \frac{b_H}{2} \\ \frac{h_G}{2} \end{pmatrix} \tag{8.56}$$

$$\vec{R}_2 = \begin{pmatrix} 0 \\ -\frac{b_H}{2} \\ \frac{h_G}{2} \end{pmatrix} \tag{8.57}$$

Es gelten folgende Bezeichnungen:

- l_H: Länge der Hülle
- b_H: Breite der Hülle
- h_H: Höhe der Hülle
- h_G: Höhe der Gondel

Beide Rotoren üben jeweils eine Kraft auf das Luftschiff aus, welche durch folgende Gleichungen dargelegt wird. Gleichung 8.58 beschreibt die Kraft des linken, Gleichung 8.59 die Kraft des

rechten Rotors, auf das Koordinatensystems des Luftschiff beziehend.

$$\vec{F}_{R1} = \begin{pmatrix} -F_{R1} \cdot \sin(\phi_1) \\ 0 \\ F_{R1} \cdot \cos(\phi_1) \end{pmatrix} \tag{8.58}$$

$$\vec{F}_{R2} = \begin{pmatrix} -F_{R2} \cdot \sin(\phi_2) \\ 0 \\ F_{R2} \cdot \cos(\phi_2) \end{pmatrix} \tag{8.59}$$

Dabei gelten folgende Bezeichnungen:

- F_{R1}: Kraft des linken Rotors
- F_{R2}: Kraft des rechten Rotors
- ϕ_1: Ausrichtung der beiden Rotoren. bei $\phi_1 = 0$ zeigen beide Rotoren nach unten.

Die auftretenden Reibungskräfte sind durch Gleichung 8.60 für die Translation sowie Gleichung 8.61 für die Rotation gegeben.

$$\vec{F}_{rei} = -\begin{pmatrix} k_{Rei} \\ 0 \\ 0 \end{pmatrix} \cdot v_x \tag{8.60}$$

$$\vec{F}_{rot} = -\begin{pmatrix} 0 \\ 0 \\ k_{Reiz} \end{pmatrix} \cdot \omega_z \tag{8.61}$$

Die Rotorenkräfte sowie die Reibungskraft werden in Bezug auf das relative Koordinatensystem des Blimps angegeben. Die Gewichtskraft (Gleichung 8.62) sowie die Auftriebskraft (Gleichung 8.63) werden in Bezug auf das absolute Koordinatensystem angegeben.

$$\vec{F}_G = \begin{pmatrix} 0 \\ 0 \\ -F_G \end{pmatrix} \tag{8.62}$$

$$\vec{F}_a = \begin{pmatrix} 0 \\ 0 \\ F_A \end{pmatrix} \tag{8.63}$$

mit

$$F_G = m \cdot g \tag{8.64}$$

$$F_A = V \cdot k_A \tag{8.65}$$

Dabei ist

- m: Gesamtmasse des Luftschiffs
- $g = 9.81 \frac{m}{s^2}$: Erdbeschleunigung
- V: Volumen des Gaskörpers
- $k_A \simeq 12 \frac{N}{m^3}$: Auftriebskonstante [15]

Die Gesamtmasse sowie das Volumen des Gaskörpers werden in den folgenden Unterabschnitten spezifiziert.

8.3.5.3 2. Konfigurationsvorschlag

Wie im vorhergehenden Unterabschnitt "Skizzierung" auf Seite **??** beschrieben wird, existiert noch ein dritter Rotor, welcher zur Seite wegzeigt. Dessen Position wird durch Gleichung 8.66, der Kraftvektor wird durch Gleichung 8.67 dargelegt.

$$\vec{R}_3 = \begin{pmatrix} -\frac{l_H}{2} \\ 0 \\ h_G + \frac{h_H}{2} \end{pmatrix} \tag{8.66}$$

$$\vec{F}_{R3} = \begin{pmatrix} 0 \\ -F_{R3} \\ 0 \end{pmatrix} \tag{8.67}$$

Der Schwerpunkt des Luftschiffs liegt in

$$\vec{S} = \frac{m_G \cdot \vec{S_G} + m_H \cdot \vec{S}_H + m_{R3} \cdot \vec{S}_{R3}}{m_G + m_H + m_{R3}} \tag{8.68}$$

mit

$$\vec{S}_G = \begin{pmatrix} 0 \\ 0 \\ h_H + \frac{h_G}{2} \end{pmatrix} \tag{8.69}$$

$$\vec{S}_H = \begin{pmatrix} 0 \\ 0 \\ h\frac{h_H}{2} \end{pmatrix} \tag{8.70}$$

$$\vec{S}_{R3} = \begin{pmatrix} -\frac{l_H}{2} \\ 0 \\ h_G + \frac{h_H}{2} \end{pmatrix} \tag{8.71}$$

$$\tag{8.72}$$

Dabei ist:

- m_G: Masse der Gondel exlusive der Rotoren
- m_H: Masse der Hülle. inklusive deren Inhalt
- m_{R3}: Masse des dritten Rotors
- S_G: Schwerpunkt der Gondel
- S_H: Schwerpunkt der Hülle
- S_{R3}: Schwerpunkt des dritten Rotors

Die sich aus den Kräften resultierenden Drehmomente sollen nur für den 3. Konfigurationsvorschlag behandelt werden.

8.3.5.4 3. Konfigurationsvorschlag

Der dritte Konfigurationsvorschlag wird, wie im vorhergehenden Abschnitt "Skizzierung" beschrieben, in die Realität umgesetzt. Aus diesem Grund werden alle Gleichungen zum dritten Konfigurationsvorschlag angeschrieben.

Im Vergleich zum zweiten Konfigurationsvorschlag ist der dritte Rotor am hinteren oberen Ende des Luftschiffs montiert. Seine Position ist durch Gleichung 8.73 gegeben.

$$\vec{R}_3 = \begin{pmatrix} -\frac{l_H}{2} + \Delta l_{R3} \\ 0 \\ h_G + \frac{h_H}{2} + \Delta h_{R3} \end{pmatrix} \tag{8.73}$$

Dabei ist

- Δl_{R3}: horizontaler Versatz des Rotor 3 vom hintersten Punkt des Gaskörpers
- Δh_R: vertikaler Versatz des Rotor 3 vom hintersten Punkt des Gaskörpers

Dementsprechend lässt sich der Schwerpunkt des Luftschiffs mit Gleichung 8.74 berechnen.

$$\vec{S} = \frac{m_G \cdot \vec{S}_G + m_R \cdot \left(\vec{S}_{R1} + S_{R2} + \vec{S}_{R3}\right)}{m_G + m_H + 3 \cdot m_R} = \begin{pmatrix} S_x \\ S_y \\ S_z \end{pmatrix} \tag{8.74}$$

mit

$$\vec{S}_{R1} = \vec{R}1 \tag{8.75}$$
$$\vec{S}_{R2} = \vec{R}2 \tag{8.76}$$
$$\vec{S}_{R3} = \vec{R}3 \tag{8.77}$$

Die Kraft des Rotors 3 wird durch Gleichung 8.78 dargelegt.

$$\vec{F}_{R3} = \begin{pmatrix} F_{R3} \cdot \cos(\phi_{R3}) \\ 0 \\ F_{R3} \cdot \sin(\phi_{R3}) \end{pmatrix} \tag{8.78}$$

Dabei ist

- F_{R3}: Kraft des dritten Rotors
- ϕ_{R3}: Winkel des dritten Rotors. Konstante, weil der Rotor fixiert ist

Die Summe aller Rotorenkräfte (Gleichung 8.58, Gleichung 8.59, Gleichung 8.78) beträgt

$$\begin{aligned} \vec{F}_{ges} &= F_{R1} + F_{R2} + F_{R3} \\ &= \begin{pmatrix} -F_{R1} \cdot \sin(\phi_1) \\ 0 \\ F_{R1} \cdot \cos(\phi_1) \end{pmatrix} + \begin{pmatrix} -F_{R2} \cdot \sin(\phi_1) \\ 0 \\ F_{R2} \cdot \cos(\phi_1) \end{pmatrix} + \begin{pmatrix} F_3 \cdot \cos(\phi_{R3}) \\ 0 \\ F_3 \cdot \sin(\phi_{R3}) \end{pmatrix} \\ &= \begin{pmatrix} -F_{R1} \cdot \sin(\phi_1) - F_{R2} \cdot \sin(\phi_1 + F_3 \cdot \cos(\phi_{R3}) \\ 0 \\ F_{R1} \cdot \cos(\phi_1) + F_{R2} \cdot \cos(\phi_1) + F_3 \cdot \sin(\phi_{R3}) \end{pmatrix} \end{aligned} \tag{8.79}$$

Aus den Kräften F_{R1} (Gleichung 8.58), F_{R2} (Gleichung 8.59) sowie F_{R3} (Gleichung 8.78)können die daraus folgenden Drehmomente berechnet werden.

$$\begin{aligned}
\vec{M}_{R1} &= \vec{F}_{R1} \times \left(\vec{S}_1 - \vec{S}\right) \\
&= \begin{pmatrix} -F_{R1} \cdot \sin(\phi_1) \\ 0 \\ F_{R1} \cdot \cos(\phi_1) \end{pmatrix} \times \begin{pmatrix} -S_x \\ \frac{b_H}{2} - S_y \\ h_G + \frac{h_H}{2} - S_z \end{pmatrix} \\
&= \begin{pmatrix} -F_{R1} \cdot \cos(\phi_1) \cdot \left(\frac{b_H}{2} - S_y\right) \\ F_{R1} \cdot \cos(\phi_1) \cdot S_x + F_{R1} \cdot \cos(\phi_1) \cdot \left(h_G + \frac{h_H}{2} - S_z\right) \\ -F_{R1} \cdot \cos(\phi_2) \cdot \left(\frac{b_H}{2} - S_x\right) \end{pmatrix}
\end{aligned} \tag{8.80}$$

$$\begin{aligned}
\vec{M}_{R2} &= \vec{F}_{R2} \times \vec{R}_2 \\
&= \begin{pmatrix} -F_{R2} \cdot \sin(\phi_1) \\ 0 \\ F_{R2} \cdot \cos(\phi_1) \end{pmatrix} \times \begin{pmatrix} -S_x \\ -\frac{b_H}{2} - S_y \\ h_G + \frac{h_H}{2} - S_z \end{pmatrix} \\
&= \begin{pmatrix} F_{R2} \cdot \cos(\phi_1) \cdot \left(\frac{b_H}{2} + S_y\right) \\ -F_{R2} \cdot \cos(\phi_1) \cdot S_x + F_{R2} \cdot \cos(\phi_1) \cdot \left(h_G + \frac{h_H}{2} - S_z\right) \\ F_{R2} \cdot \cos(\phi_2) \cdot \frac{b_H}{2} \end{pmatrix}
\end{aligned} \tag{8.81}$$

$$\begin{aligned}
\vec{M}_{R3} &= \vec{F}_{R3} \times \vec{R}_3 \\
&= \begin{pmatrix} F_{R3} \cdot \cos(\phi_{R3}) \\ 0 \\ F_{R3} \cdot \sin(\phi_{R3}) \end{pmatrix} \times \begin{pmatrix} -\frac{l_H}{2} + \Delta l_{R3} - S_x \\ -S_y \\ h_G + \frac{h_H}{2} + \Delta h_{R3} - S_z \end{pmatrix} \\
&= \begin{pmatrix} F_3 \cdot \sin(\phi_3) \\ -F_3 \cdot \sin(\phi_{R3}) \cdot \left(\frac{l_H}{2} - \Delta l_{R3} S_x\right) - F_3 \cdot \cos(\phi_{R3}) \cdot \left(h_G + \frac{h_H}{2} + \Delta h_{R3} - S_z\right) \\ 0 \end{pmatrix}
\end{aligned} \tag{8.82}$$

Die Summe dieser Drehmomente ergibt

$$\begin{aligned}
\vec{M}_{ges} &= \vec{M}_{R1} + \vec{M}_{R2} + \vec{M}_{R3} \\
&= \begin{pmatrix} -F_{R1} \cdot \cos(\phi_1) \cdot \left(\frac{b_H}{2} - S_y\right) \\ F_{R1} \cdot \cos(\phi_1) \cdot S_x + F_{R1} \cdot \cos(\phi_1) \cdot \left(h_G + \frac{h_H}{2} - S_z\right) \\ -F_{R1} \cdot \cos(\phi_2) \cdot \left(\frac{b_H}{2} - S_x\right) \end{pmatrix} \\
&\quad + \begin{pmatrix} F_{R2} \cdot \cos(\phi_1) \cdot \left(\frac{b_H}{2} + S_y\right) \\ -F_{R2} \cdot \cos(\phi_1) \cdot S_x + F_{R2} \cdot \cos(\phi_1) \cdot \left(h_G + \frac{h_H}{2} - S_z\right) \\ F_{R2} \cdot \cos(\phi_2) \cdot \frac{b_H}{2} \end{pmatrix} \\
&\quad + \begin{pmatrix} F_3 \cdot \sin(\phi_3) \\ -F_3 \cdot \sin(\phi_{R3}) \cdot \left(\frac{l_H}{2} - \Delta l_{R3} S_x\right) - F_3 \cdot \cos(\phi_{R3}) \cdot \left(h_G + \frac{h_H}{2} + \Delta h_{R3} - S_z\right) \\ 0 \end{pmatrix}
\end{aligned} \tag{8.83}$$

Der dritte Summand von Gleichung 8.83 ist zu umfangreich, um dargestellt zu werden. Deshalb wird bei der Konstruktion der Simulation versucht, mit möglichst einfachen Ausdrücken zu arbeiten.

Mit den Gleichungen

- (8.62): Gewichtskraft
- (8.63): Auftriebskraft

- (8.60): Reibungskraft bezüglich der Translation
- (8.61): Reibunskraft bezüglich der Rotation
- (8.79): Gesamtkraft
- (8.83): gesamtes Drehmoment

kann mittels Einsetzen in die Gleichungen

- (8.7): Berechnung der Gesamtkraft
- (8.8): Berechnung des gesamten Drehoments
- (8.10): Berechnung der Winkelbeschleunigung
- (8.12): Berechnung der Winkelgeschwindigkeit
- (8.13): Berechnung des zurückgelegten Wegs
- (8.22): Berechnung der Lage

das physikalische Modell des Blimps gebildet werden. Dessen Umsetzung soll im Abschnitt "Software" erläutert werden.

8.3.5.5 4. Konfigurationsvorschlag

Wie im vorhergehenden Abschnitt "Skizzierung" erwähnt wird, soll im Zuge des 4. Konfigurationsvorschlags der Schwerpunkt des Luftschiffes mittels eines beweglichen Akkus variiert werden können.

Die Lage des Schwerpunkts ergibt sich somit aus:

$$\vec{S} = \frac{\vec{S}_{R1} \cdot m_{R1} + \vec{S}_{R2} \cdot m_{R2} + \vec{S}_H \cdot m_H + \vec{S}_P \cdot m_P + \vec{S}_A(\phi_A) \cdot m_A}{m_{R1} + m_{R2} + m_H + m_P + m_A} \tag{8.84}$$

mit

$$m_G = m_P + m_A \tag{8.85}$$

$$\vec{S}_a = \begin{pmatrix} \sin(\phi_A) \\ 0 \\ -\cos(\phi_A) \end{pmatrix} \cdot l \tag{8.86}$$

hierbei ist

- m_P: Masse der Platine
- m_A: Masse des Akkus
- S_P: Schwerpunkt der Platine
- S_A: Schwerpunkt des Akkus
- ϕ_A: Auslenkung des Akkus relativ zum Luftschiff: bei $\phi_A = 0$ zeigt der Akku nach unten.

Wenn $\phi_A > 0$, verschiebt sich der Schwerpunkt des Akkus nach vorne. Somit verschiebt sich der Auftriebspunkt relativ zum Schwerpunkt nach hinten, was zu einem Drehmoment führt, welches das Drehmoment, verursacht durch die Luftreibungskräfte (Gleichung 8.60), zum Teil kompensieren kann.

Die Realisierung des 4. Konfigurationsvorschlags wird im Abschnitt "Software" nochmals kurz beschrieben.

9 Verwaltung der Diplomarbeit

9.1 Versionkontrolle

Damit alle Teammitglieder zu jedem Zeitpunkt immer den gleichen Stand der Diplomarbeit haben, wurde die Verionskontrolle Mercurial mit dem Client SourceTree verwendet (siehe 11). Als Onlineschnittstelle wurde der Online-Service Bitbucket[24] verwendet. Dieser unterstützt die verwendete Versionkontrolle Mercurial. Bei richtiger Verwendung der Versionskontrolle fügt diese verschiedene Versionen gleicher Dokumente richtig zusammen. Wenn allerdings die zwei unterschiedlichen Versionen an der selben Stelle bearbeitet worden ist, dann löst dies in der Versionskontrolle ein Konflikt aus. Dieser muss von einem Benutzer selbst gelöst werden.

Abbildung 60: Service Atlassian Bitbucket

Um die Diplomarbeit übersichtlich zu halten wurde zuallerst auf Bitbucket ein Team erstellt. In dieses Team wurden alle Teammitglieder aufgenommen. Eine durchdachte Ordnerstruktur bei einer Diplomarbeit ist sehr von nöten. Diese teilt die Diplomarbeit in verschiedene Themen auf. Für jedes dieser Themen wurde auf Bitbucket ein "Repository" erstellt. Diese "Repositories" lauten:

- Code
- Documentation
- Electronics
- Mechanics
- Other

[24] Atlassian Bitbucket, https://bitbucket.org

9.2 Stundenaufzeichnung

Um die investierte Arbeit jedes einzelnen Teammitglieds im Überblick zu behalten, wird eine Stundenaufzeichnung in Microsoft Excel erstellt (siehe 11). In diese werden nach jeder vollbrachten Arbeit die Stunden eingetragen.

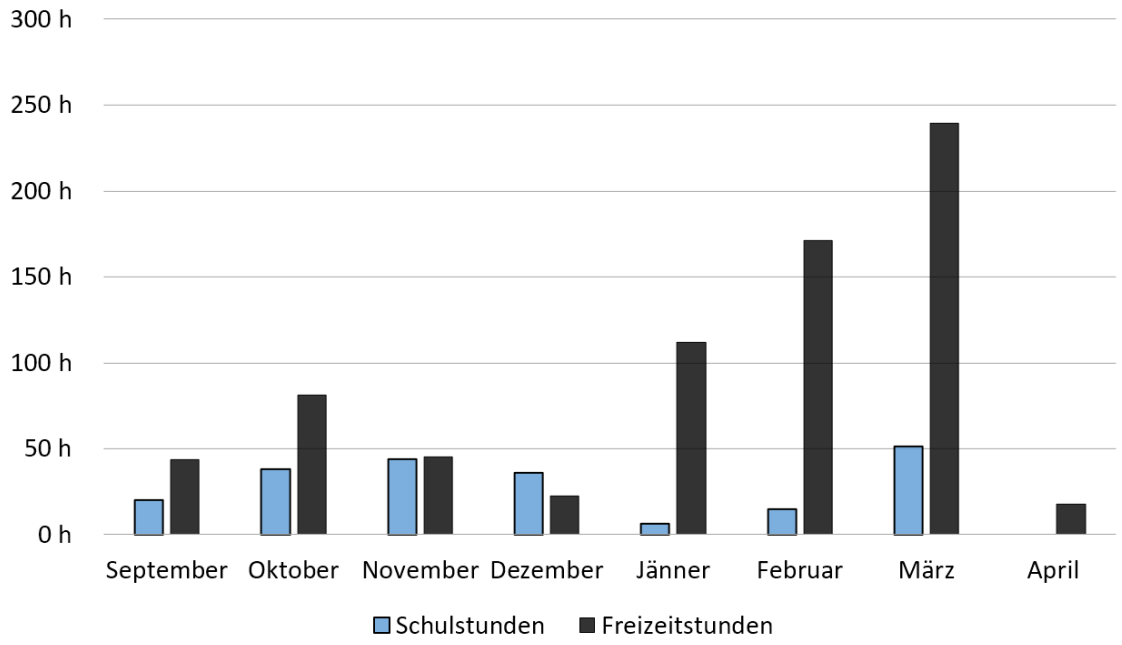


Abbildung 61: Freizeit- und Schulstundenverteilung über Monate

Das Diagramm in der Abbildung 61 stellt Freizeit- und Schulstunden über die Monate verteilt dar. Erwähnenswert sind die Freizeitstunden, die gegen Ende der Diplomarbeit signifikant ansteigen.

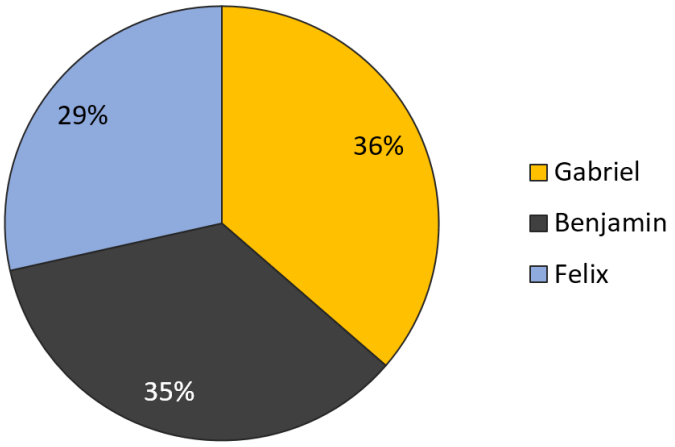


Abbildung 62: Stundenverteilung innerhalb des Teams

Im Diagramm in der Abbildung 62 ist die Verteilung der Gesamtstunden zwischen den einzelnen Teammitgliedern zu erkennen.

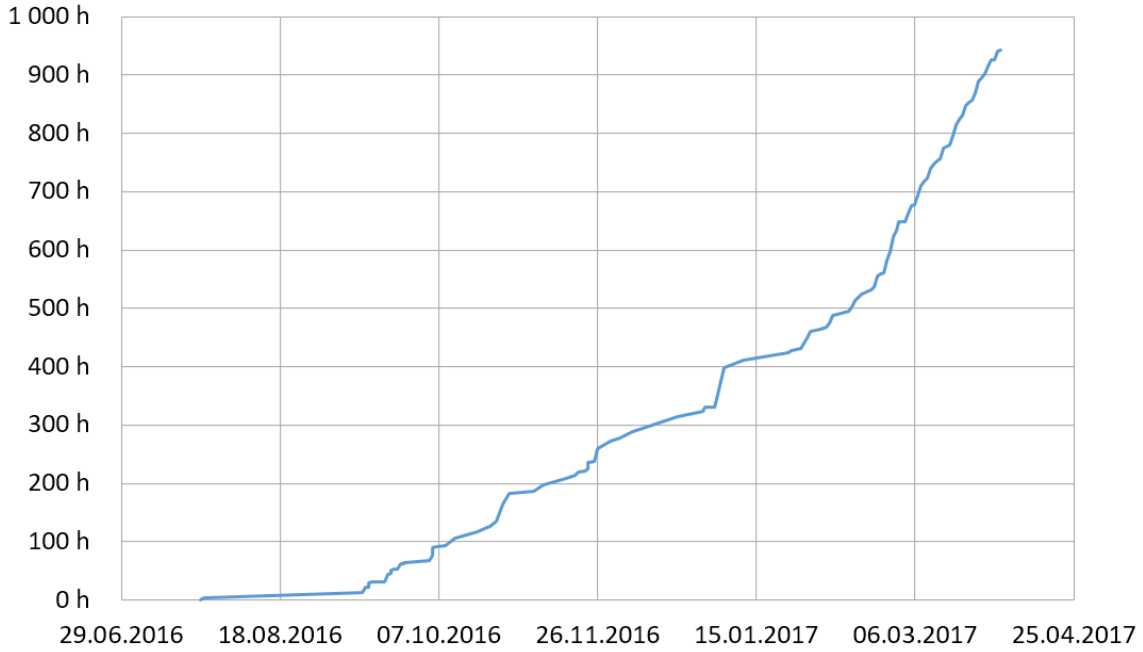


Abbildung 63: Gesamtstundenverlauf

Der Verlauf der Gesamtstunden ist in der Abbildung 63 dargestellt. Die Kurve steigt am Schluss stark an, dies ist in der 61 ebenfalls erkennbar.

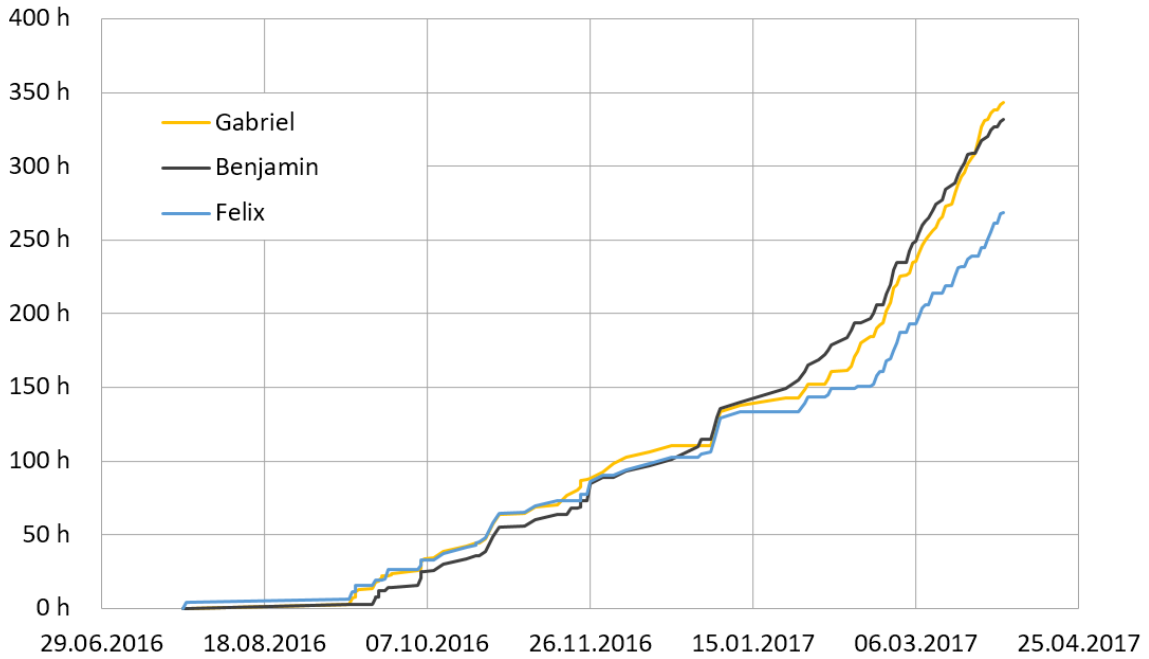


Abbildung 64: Stundenverlauf der einzelnen Mitglieder

Die Stundenverläufe der einzelnen Teammitgliedern wird in der Abbildung 64 ersichtlich.

10 Quellenverzeichnis

[1] MCP16331 Datenblatt
http://ww1.microchip.com/downloads/en/DeviceDoc/20005308B.pdf, abgerufen 2017

[2] Deutsche Wikipedia: Bluetooth
http://de.wikipedia.org/wiki/Bluetooth, abgerufen 2017

[3] Deutsche Wikipedia: Steuerzeichen
http://de.wikipedia.org/wiki/Steuerzeichen, abgerufen 2017

[4] Deutsche Wikipedia: Newtonsche Gesetze
http://de.wikipedia.org/wiki/Newtonsche_Gesetze, abgerufen 2017

[5] A4963 Datenblatt
http://www.allegromicro.com/~/media/Files/Datasheets/A4963-Datasheet.ashx, abgerufen 2017

[6] Projekt Silentrunner: Envelope Tutorial
http://silent-runner.net/index.php?title=Envelope_tutorial, abgerufen 2017

[7] Deutsche Wikipedia: Microsoft Powerpoint
http://de.wikipedia.org/wiki/Microsoft_PowerPoint, abgerufen 2017

[8] DMC3016LSD Datenblatt
http://www.diodes.com/assets/Datasheets/DMC3016LSD.pdf, abgerufen 2017

[9] STM32F4xxxx Application Note
ö www.st.com/resource/en/application_note/dm00115714.pdf, abgerufen 2017

[10] Exponentialmatrix: Rechengesetze
https://de.wikipedia.org/wiki/Matrixexponential, abgerufen 2017

[11] Quaternionen: Definition
https://de.wikipedia.org/wiki/Quaternion, abgerufen 2017

[12] Total Phase: Support User Manual
https://www.totalphase.com/support/articles/213706178, abgerufen 2017

[13] ToggleSwitch
http://marcangers.com/animated-switch-togglebutton-style-in-wpf/, abgerufen 2017

[14] .NET Framework
https://de.wikipedia.org/wiki/.NET_Framework, abgerufen 2017

[15] Deutsche Wikipedia: Luft
https://de.wikipedia.org/wiki/Luft, abgerufen 2017

[16] Gear Generator
https://de.wikipedia.org/wiki/Luft, abgerufen 2017

[17] Blender
https://www.blender.org/, abgerufen 2017

[18] EMAX 2805 Spezifikationen
http://willysrc.com/image/data/EMAX_cf/cf2805_2.jpg, abgerufen 2017

[19] Windreiter
http://windreiter.de/wordpress/, abgerufen 2017

11 Verwendete Tools

LTspice http://www.linear.com/designtools/software

LTspice ist eine graphische Oberfläche für das SPICE[25] Simulationstool, entwickelt von Linear Technology. Es ermöglicht die Simulation von elektrischen System im Zeit- und Frequenzbereich. Die Simulation erfolgt anhand von hierarchischen Modellen welche selbst erstellt werden oder im Internet gefunden können. Zusätzlich werden viele für gängige Modelle mitgeliefert.

LaTeX http://latex-project.org

LaTeX ist eine Makrosammlung für das Textsatzsystem TeX. Es vereinfacht die Erstellung von strukturierten Dokumenten und erzwingt die Einhaltung typographischer Standards in der Dokumentation.

Mercurial http://mercurial.selenic.com

Mercurial ist ein verteiltes, plattformunabhängiges Versionskontrollsystem. Es wird zum Austausch und Zusammenfügen der Beiträge der Mitarbeiter bei Softwareprojekten eingesetzt. Bei ILPS wurde es zum Ablgeich der LaTeX - Dokumentation und der Firmware verwendet. Bei richtiger Anwendung erfolgt die Zusammenführung von Dateien welche von verschiedenen Mitarbeitern bearbeitet wurden vollautomatisch. Um den Abgleich durchzuführen, wurde der kostenloses Repository beim Service Bitbucket[26] registriert.

Inkscape http://www.inkscape.org

Inkscape ist ein Zeichenprogramm für Vektorgrafiken in vielen Formaten, allen voran das SVG[27] - Format. Für die Dokumentation von ILPS wurde Inkscape zum Zeichnen einiger Illustationen verwendet.

Atollic TrueStudio ARM http://atollic.com

TrueStudio ist eine vereinfachte und abgeänderte Version des bekannten Programms Eclipse. TrueStudio ist abgeändert für die Programmierung von STM32 und beinhaltet bereits viele benötigte Funktionen für die Programmierung am STM32.

SourceTree http://www.sourcetreeapp.com

SourceTree ist eine grafische Benutzeroberfläche zur Bedienung der Versionskontrollsysteme Git und Mercurial. Das Programm von Atlassian, Hersteller von Entwickler- und Projektmanagement-Werkzeugen, zeigt die wichtigsten Befehle in einer einfachen Oberfläche an, sodass für diese Funktionen eine Bedienung von der Kommandozeile aus nicht notwendig ist.

Google Drive https://drive.google.com

Google Drive ist eine von Google angebotene Webanwendung, die ein Netzwerk-Dateisystem für die Synchronisation von Dateien zwischen verschiedenen Rechnern und Google-Benutzern bereitstellt und damit gleichzeitig eine Online-Datenspeicherung ermöglicht. Zusätzlich stellt es Funktionalitäten für Textverarbeitung, Tabellenkalkulation, Erstellung von Bildschirmpräsentationen, Formularen und Zeichnungen zur Verfügung. Drive ermöglicht Anwendern, diese Dokumente gemeinsam mit anderen Nutzern zu bearbeiten, wobei Änderungen in Echtzeit bei allen Beteiligten angezeigt werden.

Blender https://www.blender.org/

Blender ist ein freie 3D-Grafiksoftware, welche es erlaubt Bild-Bearbeitung, 3D-Modellierung, -Animierung, -Texturierung auf einfache aber auch professionelle Art vorzunehmen. Dabei bietet die Software die zusätzliche Möglichkeit von physikalischen Simulationen für grafische Effekte. Für die Darstellung eines Prinzips in 3D ist Blender eine unkompliziertes Tool welches dies ermöglicht.

Microsoft Excel https://products.office.com/de-at/excel

Microsoft Excel ist eine sehr etablierte Tabellenkalkulation, welche zur Stundenaufzeichnung verwendet wurde. Diese Stundenaufzeichnung wurde in dem Clouddienst Onedrive abgespeichert und konnte so von allen Teammitgliedern bearbeitet werden. Die aktuellste Version zurzeit ist Microsoft Excel 2016.

Microsoft Powerpoint https://products.office.com/de-at/powerpoint

Microsoft Powerpoint ist die meist verbreitetste Präsentationsoftware am Markt [7]. Es wurde damit die Zwischenpräsentation und die Abschlusspräsentation der Diplomarbeit erstellt. Die aktuellste Version von Powerpoint ist Microsoft Powerpoint 2016

Microsoft Visual Studio https://www.visualstudio.com/de/

Microsoft Visual Studio ist eine sehr komfortable und monderne Entwicklungsumgebung für zahlreiche Hochsprachen. Im Falle von SVAC wurde Sie benutzt um die WPF-Anwendung *Bluetooth_DebugWindow* (siehe 7.2) zu gestalten. Dabei wurde die Objektorientierte Programmiersprache C# verwendet. Die WPF-Anwendung wurde mit Microsoft Visual Studio 2015 erstellt, wobei die aktuellste Version Microsoft Visual Studio 2017 ist (Stand März 2017).

Microsoft Onedrive https://onedrive.live.com/

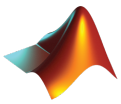
Die Funktion des Clouddienstes Microsoft Onedrive ist mit der Funktion des Clouddienst Google Drive zu vergleichen. Onedrive wurde verwendet, weil die Stundenaufzeichnung mit Excel durchgeführt wurde und Onedrive mit Excel gut harmoniert.

ST-Link Utility http://www.st.com/en/embedded-software/stsw-link004.html

Die ST-Link Utility von STMicroelectronics bietet eine Vielzahl an Möglichkeiten zur Einstellung und Programmierung eines STM Mikrocontrollers. Dabei können die Register des µC ausgelesen und Konfigurations Bits wie z.B. der Lese-Schutz gesetzt werden. Die Utility ermöglicht zusätzlich das automatische Programmieren des verbundenen Mikrocontrollers

Mathworks Matlab https://de.mathworks.com/products/matlab.html

Matlab bietet die Möglichkeit, komplexe mathematische Aufgaben, insbesondere der linearen Algabra, numerisch zu lösen. Zahlreiche Erweiterungen, Toolboxes genannt, erweitern den Funktionsumfang beträchtlich. Hierbei sei exemplarisch die *Control System Toolbox* erwähnt, welche den Entwurf, Verfizierung von Regelungssystemen signifikant vereinfacht. Matlab bildet die Grundlage für Simulink, ein Simulationstool mit graphischer Eingabe, geeignet zur Modellbildung.



Draw IO https://www.draw.io/

Draw IO ist ein Add-On des Clouddienstes Google-Drive, mit dem es einfach und praktikabel ist, Fluss-, Prozess-, UML-, oder ähnliche Diagramme zu gestalten. Die erstellten Werke können in verschiedenen Dateiformaten exportiert werden (z.B PNG, SVG oder PDF).

12 Abkürzungsverzeichnis

Abkürzung	**Bezeichnung**
2D	Zweidimensional
3D	Dreidimensional
µC	Mikrocontroller
ADC	Analog to Digital Converter
Akku	Akkumulator
Bit	Binary Digit
BLDC	Brushless DC
BT	Bluetooth
CSV	Comma Separated Values
DC	Direct Current
DGPS	Differential GPS
DMA	Direct Memory Access
DOP	Dillution of Precision
FET	Feldeffekttransistor
GPIO	General-purpose input/output
GPS	Global Positioning System
GSM	Global System for Mobile Communications
GUI	Graphical User Interface
HF	Hochfrequenz
IIC / I^2C	Inter-Integrated Circuit
IC	Integrated Circuit
IO	Input / Output
JTAG	Joint Test Action Group
KB	Kilo Bytes
LF	Low Frequency
LiPo	Lithium-Polymer-Akkumulator
MATLAB	Matrix Laboratory
MCU	Microcontroller Unit
MOSFET	Metall-Oxid-Halbleiter-FET
PDF	Portable Document Format
PDU	Packet Data Unit
PIN	Persönliche Identifikationsnummer
PLL	Phase-Locked Loop
PNG	Portable Network Graphics
PWM	Pulsweitenmodulation
SCAV	Self Controlled Aeronautic Vessel
SMD	Surface Mount Device
SMS	Short Message Service
SPI	Serial Peripheral Interface
ST	STMicroelectronics
SVG	Scalable Vector Graphics
USART	Universal Asynchronous Receiver/Transmitter

Tabelle 16: Abkürzungsverzeichnis

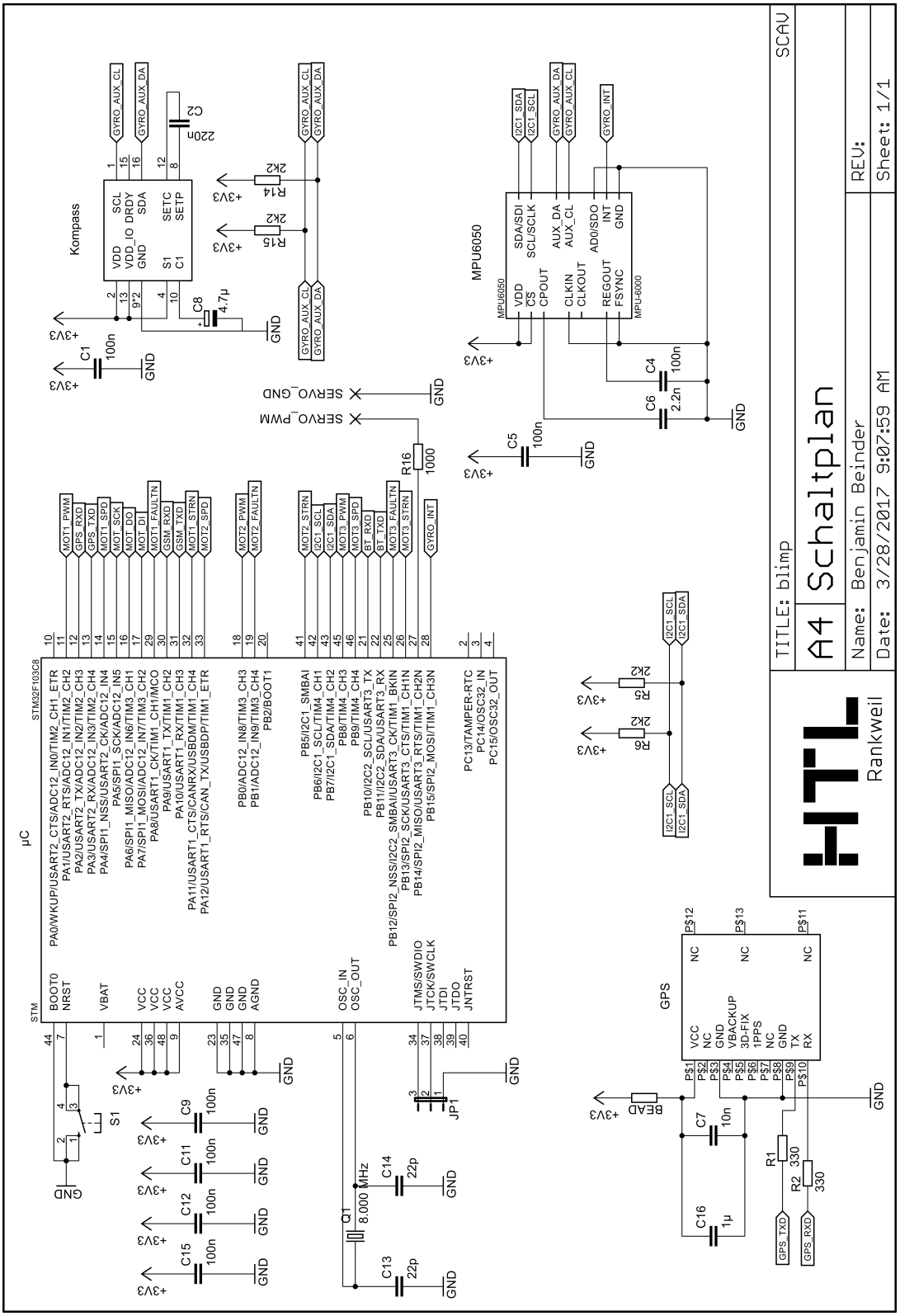
13 Abbildungsverzeichnis

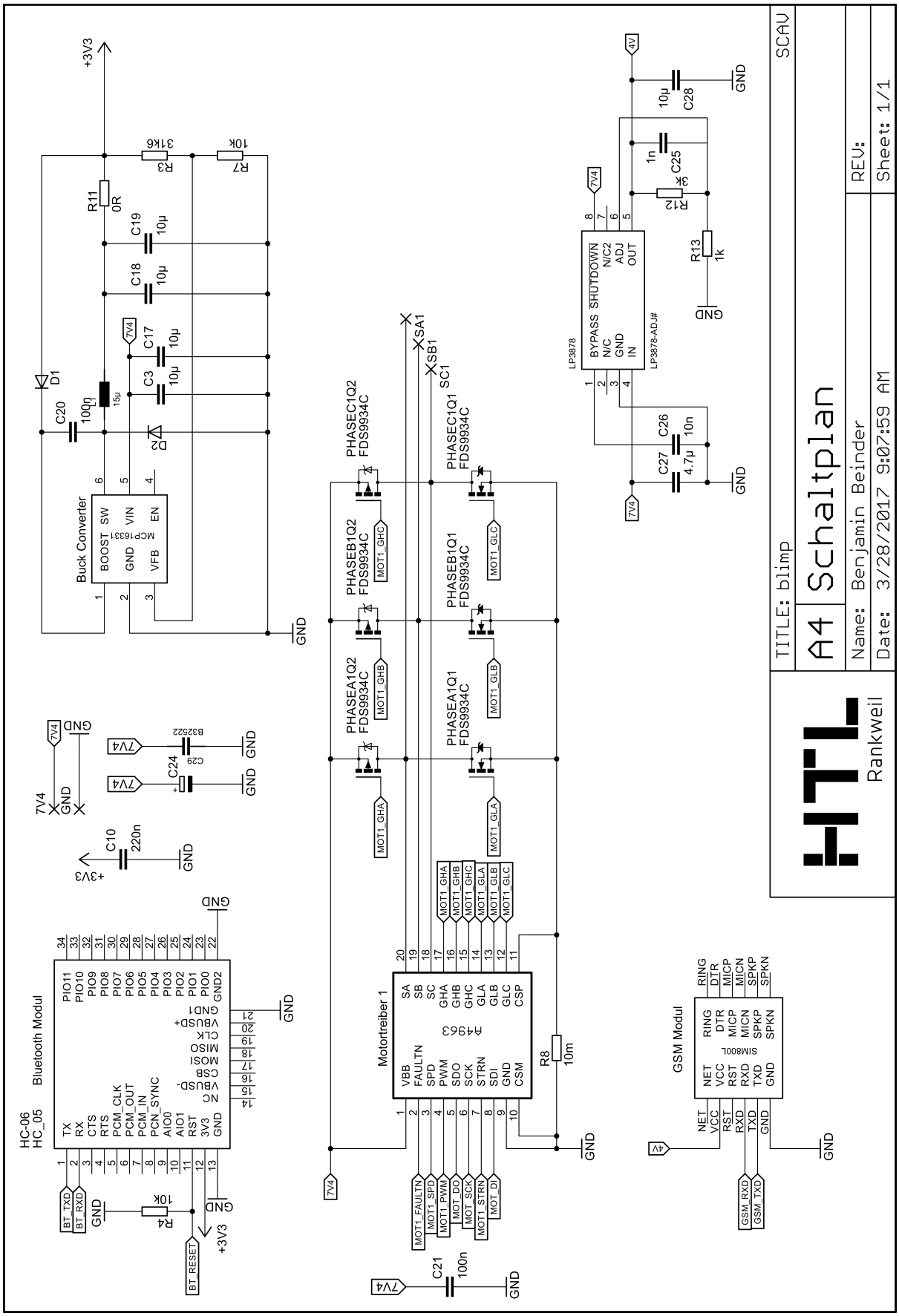
14 Tabellenverzeichnis

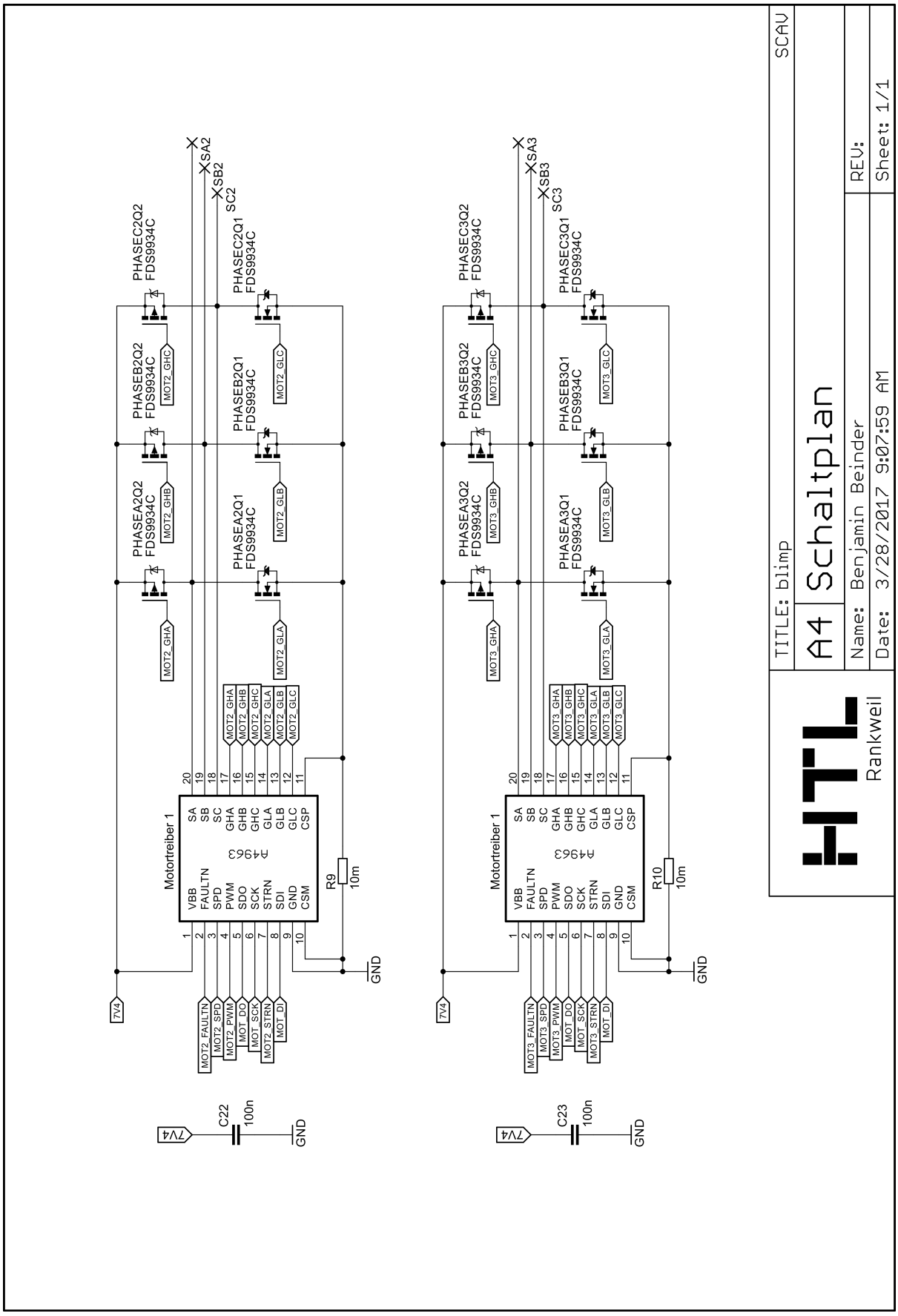
15 Codeabschnittverzeichnis

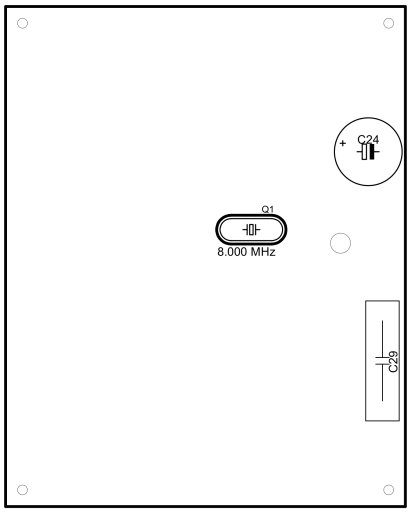
16 Anhang

16.1 Mainboard

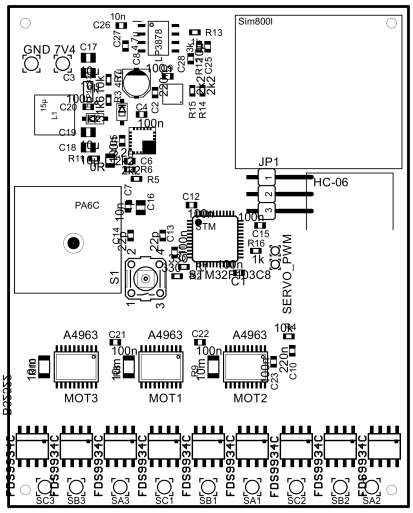



SCAV
TITLE: blimp
A4 Schaltplan
Name: Benjamin Beinder
Date: 3/28/2017 9:07:59 AM
REV:
Sheet: 1/1
HTL Rankweil
+3V3
GND
7V4
4V
Buck Converter
MCP1633
BOOST SW VIN EN
GND VFB
D1
D2
L1 15µ
C20 100n
C3 10µ
C17 10µ
C18 10µ
C19 10µ
R11 0R
R3 31k6
R7 10k
C10 220n
C24 B32522
C29
Bluetooth Modul
HC-06
HC_05
TX RX CTS RTS PCM_CLK PCM_OUT PCM_IN PCN_SYNC AIO0 AIO1 RST 3V3 GND
PIO11 PIO10 PIO9 PIO8 PIO7 PIO6 PIO5 PIO4 PIO3 PIO2 PIO1 PIO0 GND2
GND1 VBUSB+ CLK MISO MOSI CSB VBUSB- NC
BT_TXD BT_RXD BT_RESET
R4 10k
Motortreiber 1
A4963
VBB FAULTN SPD PWM SDO SCK STRN SDI GND CSM
SA SB SC GHA GHB GHC GLA GLB GLC CSP
R8 10m
C21 100n
MOT1_FAULTN MOT1_SPD MOT1_PWM MOT_DO MOT_SCK MOT1_STRN MOT_DI
MOT1_GHA MOT1_GHB MOT1_GHC MOT1_GLA MOT1_GLB MOT1_GLC
PHASEA1Q1 FDS9934C
PHASEB1Q1 FDS9934C
PHASEC1Q1 FDS9934C
PHASEA1Q2 FDS9934C
PHASEB1Q2 FDS9934C
PHASEC1Q2 FDS9934C
SA1 SB1 SC1
GSM Modul
SIM800L
NET VCC RST RXD TXD GND
RING DTR MICP MICN SPKP SPKN
GSM_RXD GSM_TXD
LP3878
LP3878-ADJ#
BYPASS SHUTDOWN N/C GND IN
N/C2 ADJ OUT
R12 3k3
R13 1k
C25 1n
C26 10n
C27 4.7µ
C28 10µ

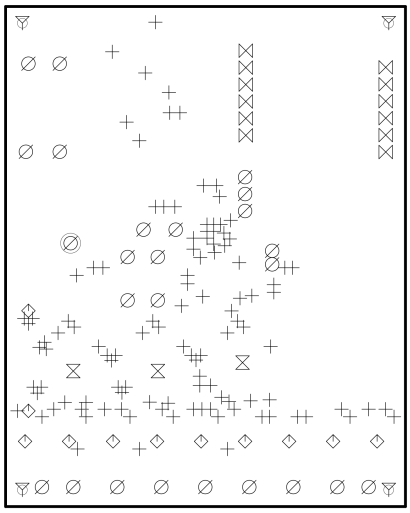

SCAV
TITLE: blimp
A4 Schaltplan
Name: Benjamin Beinder
Date: 3/28/2017 9:07:59 AM
REV:
Sheet: 1/1
HTL Rankweil
Motortreiber 1
A4963
VBB FAULTN SPD PWM SDO SCK STRN SDI GND CSM
SA SB SC GHA GHB GHC GLA GLB GLC CSP
R9 10m
R10 10m
PHASEA2Q2 FDS9934C
PHASEB2Q2 FDS9934C
PHASEC2Q2 FDS9934C
PHASEA2Q1 FDS9934C
PHASEB2Q1 FDS9934C
PHASEC2Q1 FDS9934C
PHASEA3Q2 FDS9934C
PHASEB3Q2 FDS9934C
PHASEC3Q2 FDS9934C
PHASEA3Q1 FDS9934C
PHASEB3Q1 FDS9934C
PHASEC3Q1 FDS9934C
MOT2_GHA
MOT2_GHB
MOT2_GHC
MOT2_GLA
MOT2_GLB
MOT2_GLC
MOT3_GHA
MOT3_GHB
MOT3_GHC
MOT3_GLA
MOT3_GLB
MOT3_GLC
SA2
SB2
SC2
SA3
SB3
SC3
7V4
GND
C22 100n
C23 100n
MOT2_FAULTN
MOT2_SPD
MOT2_PWM
MOT_DO
MOT_SCK
MOT2_STRN
MOT_DI
MOT3_FAULTN
MOT3_SPD
MOT3_PWM
MOT3_STRN



HTL Rankweil	TITLE: blimp	SCAV
	A4 Bestückungsplan (Bottom)	
	Name: Benjamin Beinder	REV:
	Date: 3/29/2017 3:19:09 PM	Sheet: 1/1



HTL Rankweil	TITLE: blimp		SCAV
	A4	Bestückungsplan (Top)	
	Name: Benjamin Beinder		REV:
	Date: 3/29/2017 3:17:53 PM		Sheet: 1/1



Material: FR4 1,5mm 35u Cu
Toleranz: Aussenmass -0,5mm, sonst +/_0,2mm

Bohrplan

SYM	SIZE	CNT
+	0.50 mm	112
◇	0.80 mm	11
∅	0.81 mm	4
⧖	0.90 mm	3
∅	0.91 mm	3
⋈	1.00 mm	12
∅	1.02 mm	17
▽	1.50 mm	4
∅	3.00 mm	1
	TOTAL	167

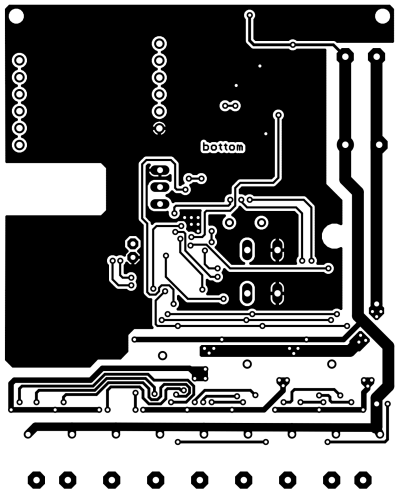
HTL Rankweil

TITLE: blimp SCAV

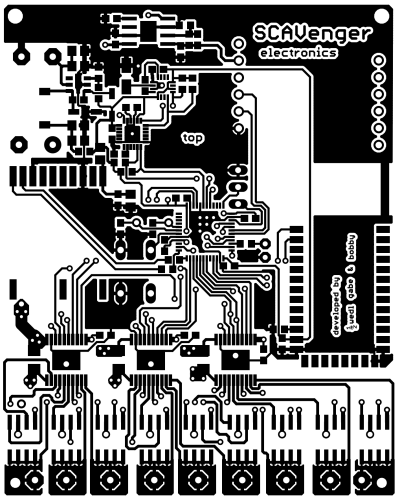
A4 Bohrplan (Top)

Name: Benjamin Beinder REV:

Date: 3/29/2017 3:24:37 PM Sheet: 1/1



HTL Rankweil	TITLE: blimp		SCAV
	A4	Layout (Bottom)	
	Name: Benjamin Beinder		REV:
	Date: 3/29/2017 3:22:01 PM		Sheet: 1/1



HTL Rankweil	TITLE: blimp		SCAV
	A4	Layout (Top)	
	Name: Benjamin Beinder		REV:
	Date: 3/29/2017 3:21:50 PM		Sheet: 1/1

Qty	Value	Device	Package	Parts
2		1,6/0,8	1,6/0,8	SERVO_GND, SERVO_PWM
1		10-XX	B3F-10XX	S1
11		2,54/1,0	2,54/1,0	7V4, GND, SA1, SA2, SA3, SB1, SB2, SB3, SC1, SC2, SC3
1		CPOL-EUE5-10.5	E5-10,5	C24
2		D-SOD123	SOD123	D1, D2
1		GPS_FGPMMOPA6C	FGPMMOPA6C	PA6C
1		JP2W	JP2W	JP1
1		R-EU_R0805	R0805	BEAD
1	0R	R-EU_R0603	R0603	R11
11	100n	C-EUC0603	C0603	C1, C4, C5, C9, C11, C12, C15, C20, C21, C22, C23
2	10k	R-EU_R0603	R0603	R4, R7
3	10m	R-EU_R1206	R1206	R8, R9, R10
2	10n	C-EUC0603	C0603	C7, C26
1	10µ	C-EUC0603	C0603	C28
4	10µ	C-EUC0805	C0805	C3, C17, C18, C19
1	15µ	ELL6G	ELL6G	L1
2	1k	R-EU_R0603	R0603	R13, R16
1	1n	C-EUC0603	C0603	C25
1	1µ	C-EUC0805	C0805	C16
1	2.2n	C-EUC0603	C0603	C6
2	220n	C-EUC0603	C0603	C2, C10
2	22p	C-EUC0603	C0603	C13, C14
4	2k2	R-EU_R0603	R0603	R5, R6, R14, R15
1	31k6	R-EU_R0603	R0603	R3
2	330	R-EU_R0603	R0603	R1, R2
1	3k	R-EU_R0603	R0603	R12
1	4.7µ	C-EUC0603	C0603	C27
1	4.7µ	CPOL-EUB	PANASONIC_B	C8
1	8.000 MHz	CRYSTALHC49S	HC49/S	Q1
3	A4963	A4963	TSSOP20	MOT1, MOT2, MOT3
1	B32522	B32522	B32522	C29
9	FDS9934C	FDS9934C	SOIC8	PHASEA1, PHASEA2, PHASEA3, PHASEB1, PHASEB2, PHASEB3, PHASEC1, PHASEC2, PHASEC3
1	HC_05	HC_05	BLUETOOTH_SMD	HC-06
1	HMC5883L	HMC5883L	LPCC16	U$2
1	HTL-A4-HA4V	HTL-A4-HA4V	HTL-A4-V	FRAME1
1	LP3878-ADJ#	LP3878-ADJ#	SO-POWERPAD	LP3878
1	MCP16331	MCP16331	SOT-23-6	MCP16331
1	MPU-6000	MPU-6000	QFN.24.4X4	MPU6050
1	SIM800LDEV	SIM800LDEV	SIM800LPACK	U$4
1	STM32F103C8	STM32F103C8	LQFP48-7X7	STM

Tabelle 17: Mainboard Stückliste

17 Danksagung

Wir möchten uns bei folgenden Personen und Unternehmen für ihre Unterstützung bedanken:

- *DI Christoph Stüttler*, für die umfangreiche Unterstützung als Projektbetreuer.
- *MSFC Rheintal*, für die Unterstützung beim mechanischen Aufbau des Luftschiffes.
- *Air Liquide GmbH*, für die materielle Unterstützung
- *DI Christoph Büsel*, für die Unterstützung beim Auffinden von HF-Einwirkungen
- *DI Franz Lauritsch*, für das zu Verfügung stellen zahlreicher elektronischer Komponenten.
- *Lucas Kneissl*, für das Drucken der mechanischen Bauteile.
- *Korrekturlesern*, für deren genaues Kontrollieren der Dimplomarbeit.
- *HTL Rankweil*, für die zur Verfügung gestellten Räumlichkeiten in der unterrichtsfreien Zeit.