# Flux –
# Eine Quadrokopter-Flugsteuerung mit dem Arduino Microcontroller

**Martin Reiche, 2017**

Version vom 26.10.2017

**1 Inhalt und Ziele ..... 4**

**2 Quadrokopter ..... 4**

2.1 Physikalische Grundlagen ..... 4

2.2 Außenperspektive der Steuerung und Bewegungen ..... 5

2.3 Prinzipielle Realisierung der Bewegungsteuerung ..... 7

**3 Systemarchitektur ..... 8**

3.1 Blockdiagramm ..... 8

3.2 Die Teilsysteme ..... 9

3.2.1 Flugsteuerung ..... 9

3.2.2 Fernsteuerempfänger ..... 9

3.2.3 Drehstromregler ..... 9

3.2.4 Inertiale Messeinheit ..... 9

3.2.5 LiPo Warner ..... 9

3.3 Elektrische Schaltung ..... 9

3.4 Die Motorsteuerung ..... 11

**4 Das Flux-Programm ..... 11**

4.1 Basis-Design ..... 11

4.2 Anforderungen und Hardware-Ressourcen ..... 12

4.3 Timing-Übersicht ..... 12

4.4 Strukturierung des Quellcodes ..... 13

4.5 Funktionsübersicht ..... 13

4.5.1 Start ..... 14

4.5.2 Hauptschleife ..... 15

4.5.3 Einstellen der Inertialen Messeinheit ..... 15

4.5.4 Lesen der Gyroskop/Accelerometer-Werte ..... 15

4.5.5 Einlesen der Fernsteuerkommandos ..... 16

4.5.6 Berechnung des Systemzustandes ..... 16

4.5.7 PID-Regelung Pitch/Roll ..... 16

4.5.8 PI-Regelung Yaw ..... 18

4.5.9 Gas-Steuerung ..... 18

4.5.10 Ansteuerung der Motoren ..... 18

4.5.11 Kalibration der Gyroskope ..... 19

4.5.12 EEPROM ..... 19

**5 Wartungsfunktionen ..... 19**

5.1 Start der Wartungsfunktionen ..... 20

5.2 Menüsteuerung der Wartungsfunktionen ..... 20
5.3 Test und Kalibration der Fernsteuerung ..... 20
5.4 IMU Test ..... 21
5.5 Horizontal-Kalibrierung* ..... 21
5.6 Einstellung der PID-Parameter* ..... 21
5.7 ESC-Kalibration ..... 21
5.8 Propeller-Balancierung ..... 22
5.9 Anzeige des EEPROM-Puffers ..... 22
5.10 Schreiben des EEPROMs ..... 23

**6 Vermischtes ..... 23**
6.1 EEPROM Verwaltung ..... 23
6.2 IMU Konfiguration ..... 24

**7 Mögliche Änderungen Erweiterungen ..... 24**
7.1 Verbesserung der Lageregelung ..... 24
7.2 Piepser zum Auffinden des Quadkopters ..... 24
7.3 Verlängerung der LOOP_PERIOD ..... 24
7.4 Notlandung ..... 24
7.5 Schutz vor Ausfall der Fernsteuerungsdaten ..... 25
7.6 Schutz vor Ausfall der Fernsteuerungssignale ..... 25
7.7 Erweiterung der Start-Prozedur ..... 25
7.8 Kalibration der Gyros als Wartungsfunktion ..... 25
7.9 Optimierung per Telemetrie ..... 25
7.10 Erweiterung der Navigation ..... 25

**8 Quellen ..... 25**

# 1 Inhalt und Ziele

Dieses Dokument beschreibt die Aufgaben, das Design, sowie die Implementierung der Quadrokopter-Flugsteuerung *Flux*, bei der ein Arduino UNO Mikrocontroller Verwendung findet. Sein Einsatz zeigt, dass auch ambitioniertere Projekte mit Standardkomponenten bewältigt werden können. Weiterhin demonstriert er die Leistungsfähigkeit und Flexibilität dieser Plattform. (Siehe https://de.wikipedia.org/wiki/Arduino_(Plattform) )

Die Flux-Software lehnt sich an die Arbeiten von Joop Brokking [JB] an. Er schreibt: „The purpose of the YMFC-AL is to provide a simple and understandable code that is needed to build an Arduino based auto-level quadcopter flight controller."

Sowohl das vorliegende Papier als auch der Flux Quellcode will einen Schritt weiter gehen:

> Ausgehend von grundsätzlichen und speziellen Überlegungen werden die Aufgaben und Strukturen der Flux-Software nachvollziehbar dokumentiert. Teilfunktionen werden in getrennten Dateien realisiert; keine Zeile C-Code soll wie vom Himmel gefallen im Quelltext erscheinen. Dies schafft die Möglichkeit für Leser, das Programm nach ihren Vorstellungen abzuändern bzw. zu erweitern.

Dieser Ansatz erforderte den Neu-Aufbau des – beinahe – gesamten Quellcode. Dabei wurde auf eine durchgängige und mit der Dokumentation übereinstimmende Terminologie geachtet.

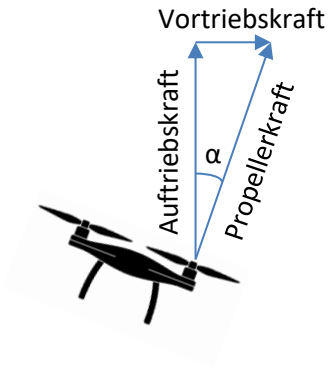
# 2 Quadrokopter

Dieses Kapitel bringt nur einen knappen Abriss der Physik und Technik eines Quadrokopters.. Der Wikipedia-Artikel https://de.wikipedia.org/wiki/Quadrocopter ist mit vielen anderen Quellen verlinkt und kann daher gut als Startpunkt für eine Vertiefung verwendet werden.

## 2.1 Physikalische Grundlagen

Warum kann ein Quadrokopter eigentlich fliegen? Die naheliegende – und grundsätzlich richtige – Antwort ist, dass die vier Propeller ihn gegen die Schwerkraft nach oben ziehen. Erzeugen alle vier Motoren denselben Schub von einem Viertel des Gesamtgewichtes, kann der Quadrokopter waagerecht in der Luft stehen, vorausgesetzt, das Fluggerät ist vollkommen symmetrisch aufgebaut. Steigert man die Drehfrequenz aller Motoren gleichmäßig, wird der Quadrokopter aufsteigen oder bei Verminderung entsprechend absinken und schließlich landen.

Weil der Quadrokopter nicht über verstellbare Flächen verfügt, die ihn im Fluge leiten könnten, müssen sämtliche Manöver durch gezielte Ansteuerung seiner Motoren erfolgen. Drehen z.B. die Propeller auf der linken Seite etwas schneller als die auf der rechten, entsteht ein Drehmoment, welches den Quadrokopter etwas zur Seite neigt. Dies bewirkt eine seitliche Beschleunigung, welche daher rührt, dass ein Teil der Propellerkraft nun parallel zur Erdoberfläche wirkt, wie man dem folgenden Diagramm zur Kräftezerlegung entnehmen kann:




Der Idealfall, dass alle Motoren genau denselben Schub liefern und dass der Schwerpunkt des Quadrokopter genau in der Mitte zwischen den Propellern liegt, wird praktisch nie erreicht. Das hat zur Folge, dass ohne weitere Maßnahmen keine stabile Position in der Waagerechten erzielt werden kann. Es braucht eine aktive Gleichgewichtsregelung, welche das Fluggerät dadurch stabilisiert, dass ein Teil der Motoren geringfügig etwas beschleunigt bzw. gebremst werden.

Damit der Quadrokopter sich nicht um seine senkrechte Achse (=Gierachse) dreht, rotieren benachbarte Propeller gegenläufig, wodurch sich deren Reaktionskräfte annullieren. Beim konventionellen Hubschrauber erledigt diese Aufgabe der Heckrotor. Auch hier ist ein exakter Ausgleich aller Motoren gefragt, der nur durch stetiges Nachjustieren erzielt werden kann.

Man erkennt, dass die Bestimmung der richtigen Drehzahl jedes einzelnen Motors die zentrale Aufgabe beim Flug eines Quadrokopter ist. Somit können wir festhalten:

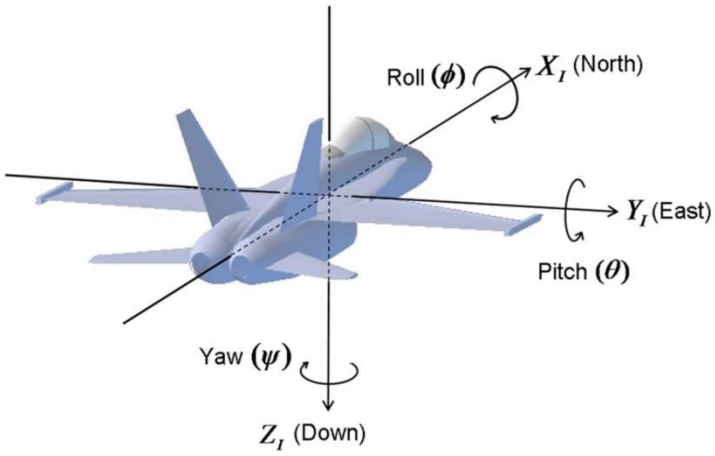
Die wesentliche Aufgabe[1] einer Flugsteuerung ist die Festlegung der Drehzahlen aller 4 Motoren, um den Quadrokopter waagerecht in der Luft zu halten bzw. entsprechend den Fernsteuerbefehlen in alle Raumrichtungen zu bewegen. Das schließt auch ein Drehen um die Gierachse (Yaw) ein.

Dabei besteht die Herausforderung an die Realisierung einer Flugsteuerung darin, alle Anforderungen an die Drehzahl jedes einzelnen Motors unter einen Hut zu bekommen, damit das Fluggerät als Ganzes nicht rotiert oder kippt, sondern sich entsprechend der Fernsteuerung fortbewegt bzw. stillsteht.

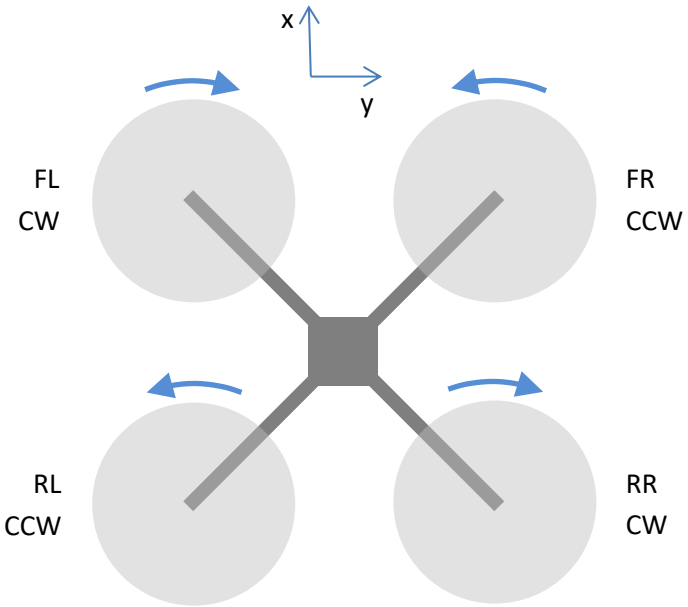
## 2.2 Außenperspektive der Steuerung und Bewegungen

Eine in sich schlüssige Beschreibung der Flugsteuerung erfordert die Definition von Richtungen und Drehsinn. Dem Vorschlag in [JB] folgend, verwenden wir folgende Vereinbarungen aus der konventionellen Luftfahrt:

[1] Natürlich sind beliebige Erweiterungen denkbar!




Siehe auch hier und hier. In der Aufsicht legen wir für unseren Quadrokopter fest:



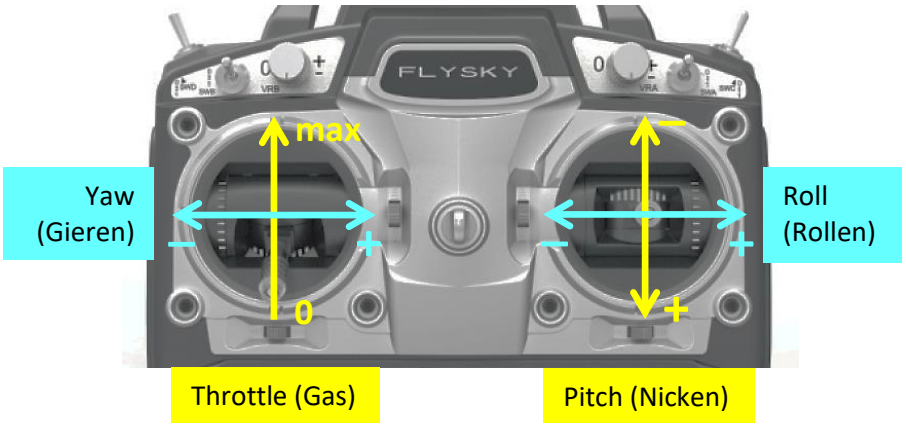

Aus der Symmetrie des Aufbaus folgt, dass ein Quadrokopter keine physikalisch bevorzugte Bewegungsrichtung in der x-y-Ebene aufweist. Anders als ein Starrflügler kann er genauso gut in y-Richtung wie in x-Richtung fliegen und sich im Flug beliebig um seine Hochachse drehen.

Bei allen Fachbegriffe arbeiten wir sowohl mit den englischen als auch deutschen Bezeichnungen:

| Englisch | Deutsch | Kommentar |
|---|---|---|
| pitch | Nicken | Winkel θ (theta) |
| roll | Rollen | Winkel ϕ (phi) |
| yaw | Gieren | Winkel ψ (psi) |
| throttle | Gas | Zwischen Null und maximal |
| FR (front right) | vorne rechts | |
| FL (front left) | vorn links | |

| | | |
|---|---|---|
| RL (rear left) | hinten links | |
| RR (rear right) | hinten rechts | |
| CW (clockwise) | Im Uhrzeigersinn | Drehrichtung der Propeller |
| CCW (counter-clockwise) | Gegen den Uhrzeigersinn | Drehrichtung der Propeller |

Auf dieses Richtungssystem beziehen sich auch die Funktionen der Steuerknüppel oder -hebel auf der Fernsteuerkonsole. Sie wird im sogenannten „Mode 2“ betrieben, welcher folgende Zuordnung vorschreibt:




Durch eingebaute Federn bewegen sich die Hebel für Nicken, Rollen und Gieren immer in die mittlere Neutralstellung, wenn sie losgelassen werden. Einzig das Gas verharrt in der zuletzt eingestellten Position.

Ausreichend Gas vorausgesetzt, soll der Quadrokopter bewegungslos in der Waagerechten schweben, wenn die Hebel für Nicken, Rollen und Gieren losgelassen werden.

## 2.3 Prinzipielle Realisierung der Bewegungsteuerung

Schaut man auf die oben beschriebenen physikalischen Grundlagen, wird klar, was bei der Bewegung der Steuerhebel zu geschehen hat:

**Pitch (Nicken)**: Schiebt man den Hebel nach vorne, erhöht man die Drehzahl der beiden hinteren Propeller und erniedrigt die der beiden vorderen. Dadurch kippt auch das Fluggerät nach vorne und nimmt Fahrt in positiver x-Richtung auf. Entsprechend gegensätzliche Bewegung und Motorsteuerung beschleunigt den Quadrokopter in negative x-Richtung.

**Roll (Rollen):** Es geschieht prinzipiell dasselbe wie beim Nicken. Bewegt man den Hebel nach rechts, erhöhen sich die Drehzahlen der beiden linken Propeller, die der beiden rechten sinken. Dadurch kippt der Quadrokopter nach rechts und nimmt Fahrt in positive y-Richtung auf. Auch hier kann man durch entsprechend gegensätzliche Bewegung das Fluggerät nach links d.h. in negative y-Richtung steuern.

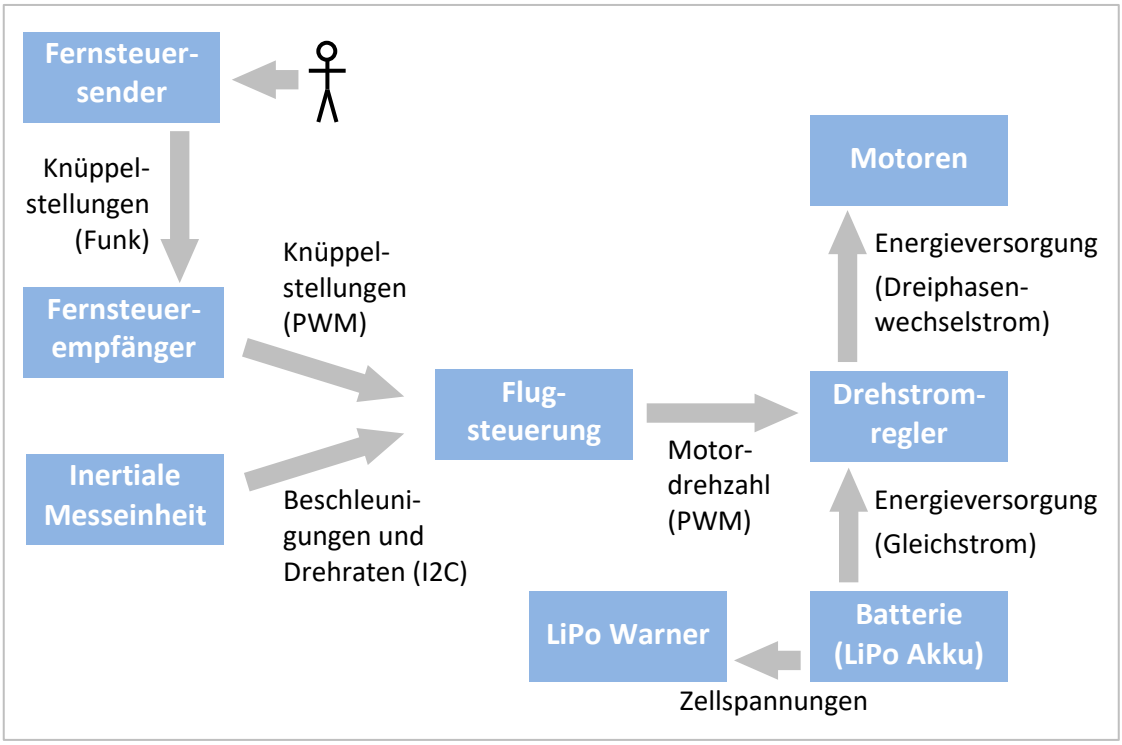
**Throttle (Gas)**: Je weiter der Hebel nach vorne geschoben wird, umso mehr steigen die Drehzahlen aller Motoren gleichartig an. Das Fluggerät beschleunigt nach oben.

**Yaw (Gieren)**: Damit das Fluggerät nicht giert, muss die Summe der Drehzahlen der rechtsläufigen (CW) Motoren mit der Summe der Drehzahlen der linksläufigen (CCW) Motoren übereinstimmen. Schafft man durch Betätigung des Yaw-Hebels hier gezielt ein Ungleichgewicht, setzt eine Drehung um die Gierachse ein. Zum Beispiel bewirkt eine Auslenkung nach links eine Beschleunigung der linksdrehenden Motoren, welche eine Drehung des Quadrokopters im Uhrzeigersinn hervorruft.

# 3 Systemarchitektur

## 3.1 Blockdiagramm

Das folgende Bild zeigt die elektrischen Baugruppen (blaue Kästen) und ihre Wirkungsbeziehungen (graue Pfeile). In Klammern werden die technischen Realisierungen benannt.




Wegen der gebotenen Zweisprachigkeit hier die wichtigsten Begriffe in Deutsch und Englisch:

| Deutsch | Englisch | Kommentar |
|---|---|---|
| Fensteuersender | transmitter, radio | Den bedient der Pilot |
| Fernsteuerempfänger | receiver | Am Quadrokopter montiert |
| PWM = Pulsweitenmodulation | pulse width modulation | Pulsdauer: 1 – 2 ms |
| Inertiale Messeinheit | inertial measurement unit (IMU) | Der Gleichgewichtssinn des Quadrokopters („Innenohr“), enthält ein Gyroskop sowie drei Akzelerometer |
| Flugsteuerung | flight controller | Ein Microcontroller (Arduino) |
| Drehstromregler | electronic speed control (ESC) | enthält je einen eigenen Microcontroller |
| LiPo Warner | LiPo warner | Warnt vor schädlicher Tiefentladung des Akkus |

## 3.2 Die Teilsysteme

### 3.2.1 Flugsteuerung

Das Flux-Programm läuft auf einem einen Atmel ATmega238P Prozessor, der auf einer Arduino UNO R3 Leiterplatte sitzt.

### 3.2.2 Fernsteuerempfänger

Der Fernsteuerempfänger übermittelt die Positionen der vier Steuerknüppel als separate Pulsweiten-modulierte Signale an die Flugsteuerung.

### 3.2.3 Drehstromregler

Die ESCs empfangen Pulsweiten-modulierte Signale von der Flugsteuerung und regeln damit die Drehzahl der jeweils angeschlossenen Motoren.

### 3.2.4 Inertiale Messeinheit

Als Inertiale Messeinheit[2] wird die „Inertial Measurement Unit“, kurz IMU vom Typ MPU 6050 eingesetzt, welche 3 Beschleunigungssensoren (Akzelerometer) und ein Gyroskop enthält.

Beachte: Gemäß dem Äquivalenzprinzip[3] können die Akzelerometer nicht zwischen Erdanziehung und Beschleunigung des Quadrokopters unterscheiden!

Kurz gesagt liefert das Akzelerometer Beschleunigungswerte in den drei Raumachsen x, y, und z; das Gyroskop hingegen liefert Winkelgeschwindigkeiten der Drehungen um dieselben Raumachsen x, y, und z.

Voreinstellungen der IMU werden über den I2C-Bus geschrieben. Details zur Programmierung finden sich in [MPU-6050-1] und [MPU-6050-2].

Die MPU-6050 wird mit der Bestückungsseite nach oben und den Anschlusspins nach hinten in den QC eingebaut. Die x-Achse des flugfesten Koordinatensystems zeigt nach vorn, y nach rechts und z nach unten.

### 3.2.5 LiPo Warner

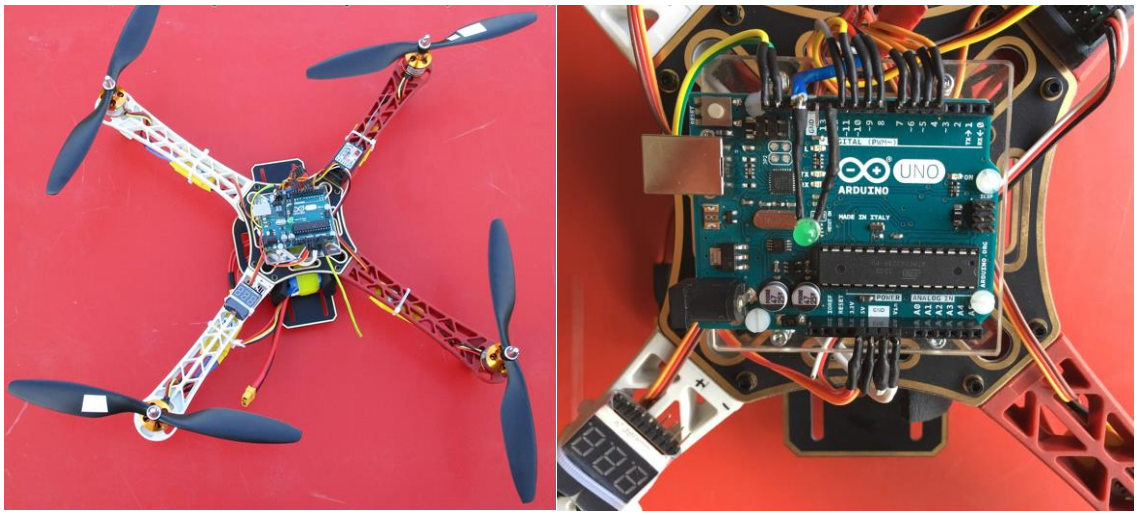
Dieses Gerät warnt während des Fluges vor einer ungewollten Tiefentladung des Akkus, welche nicht nur dessen Lebensdauer verkürzt, sondern auch den Quadrokopter gefährdet (Absturz!). Der LiPO-Warner gibt ein lautes Pfeifsignal, sobald die Spannung an irgendeiner Zelle einen Grenzwert unterschreitet. Gemäß https://www.youtube.com/watch?v=G2YONpDK108 (bei 3:45) wähle ich eine Spannung von 3,4 V. Der LiPo-Warner hat keine Verbindung zur Flugsteuerung.

## 3.3 Elektrische Schaltung

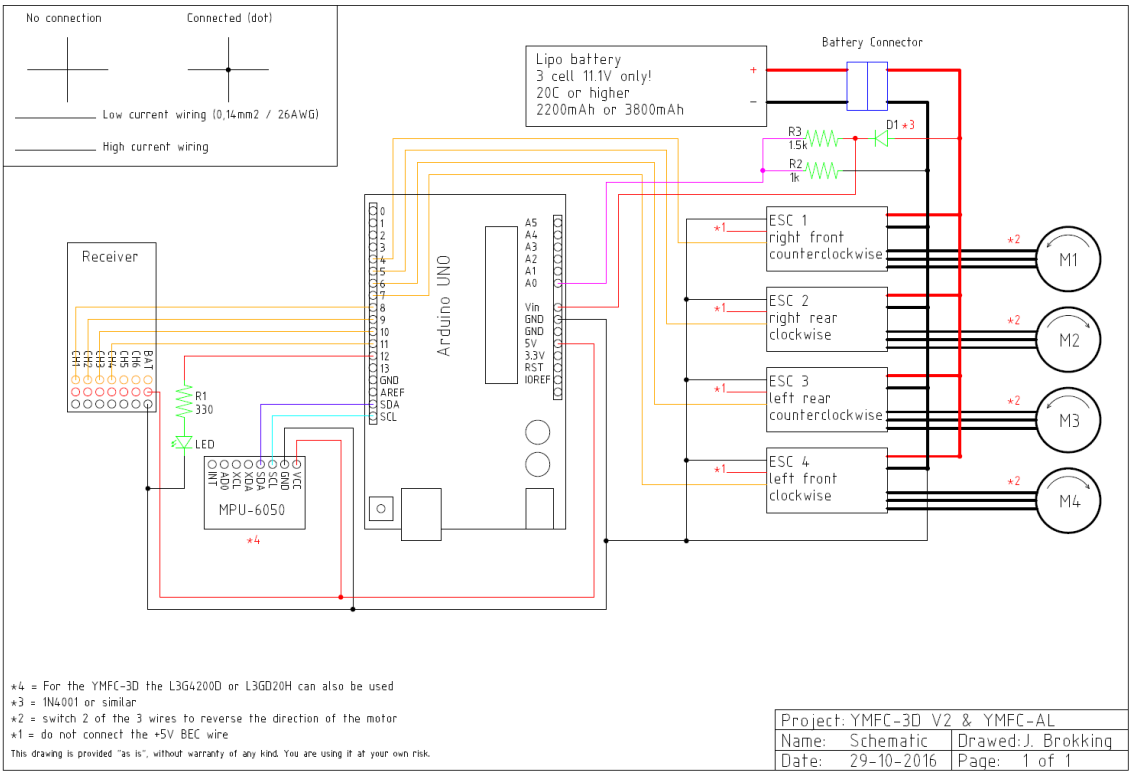
Abgesehen von minimalen Details folgt der Hardware-Aufbau dem Design von Joop Brokking [JB]. Folgende Fotos dokumentieren die Ähnlichkeit im mechanischen Aufbau:

[2] Siehe https://de.wikipedia.org/wiki/Inertiale_Messeinheit

[3] https://de.wikipedia.org/wiki/%C3%84quivalenzprinzip_(Physik)



Mit Ausnahme der Elemente zur Vermessung der Batteriespannung (R2, R3, D1 – nicht verwendet) ist die elektrische Schaltung identisch:

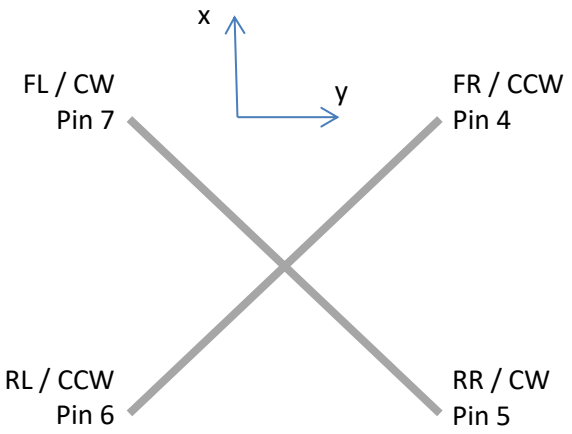



Kanalzuordnung der Fernsteuerung:

| Kanal | Funktion | Kommentar |
|---|---|---|
| 1 | Roll | |
| 2 | Pitch | |
| 3 | Throttle | |
| 4 | Yaw | |
| 5 | VRB | ungenutzt |
| 6 | VRA | ungenutzt |

## 3.4 Die Motorsteuerung

Die 4 Motoren werden von je einem ESC getrieben, der von einem PWM-Pin des Arduino angesteuert wird. Dabei drehen sich benachbarte Propeller gegenläufig, um kein Gesamtdrehmoment zu erzeugen, wenn sich alle Motoren gleich schnell drehen. Die Zuordnung der Koordinaten, Motoren, Drehrichtung und Arduino-Pins ist in der Aufsicht wie folgt:




# 4 Das Flux-Programm

## 4.1 Basis-Design

Wie dem Blockdiagram zu entnehmen ist, muss die Flugsteuerung kontinuierlich d.h. für einen Computer: periodisch aus den Stellungen der Fernsteuerknüppel und den Signalen der inertialen Messeinheit (IMU) die Drehzahl aller 4 Motoren berechnen und diese an die Drehstromregler (ESCs) weitergeben. Flux vollzieht diese Berechnung 250-mal pro Sekunde.

Die Verwendung handelsüblicher Elemente, d.h. Fernsteuerempfänger und ESCs basierend auf Pulsweitenmodulation, hat großen Einfluss auf die Programmstruktur. Will man neben dem Arduino ohne weitere digitale Hardware auskommen, muss dieser sowohl die Pulse des Fernsteuerempfängers vermessen als auch die Steuerpulse an die ESCs programmatisch generieren. Dabei sollten sich die Fehler (Jitter) in beiden Fällen klein halten (z.B. < 10 µs).

Das Programm ließe sich also wesentlich vereinfachen, könnte die Kommunikation des Arduino mit Fernsteuerung und ESC z.B. über die I2C Schnittstelle abgewickelt werden.

Der Programmablauf vollzieht sich unabhängig vom Eintreffen der Fernsteuerpulse, was eine asynchrone Verarbeitung dieser Pulse nahelegt. Bei der gewählten Fernsteuerung treffen deren Werte ca. alle 16 ms ein, während an die Motoren alle 4 ms neue Werte gesendet werden sollen. Fällt die Fernsteuerung aus, sollte der Quadrokopter nach einer Wartezeit von wenigen Sekunden eine waagerechte Position einnehmen und einen maßvoll schnellen Abstieg vollziehen. Diese Funktion ist allerdings zurzeit noch nicht implementiert.[4]

Mit dieser Kenntnis lässt sich ein erster Ablaufplan einer einfachen Flugsteuerung skizzieren:

- Start-up, einmalig, nach Einschalten. Danach fortlaufend im Wechsel:
- Lesen der Gyroskop/Akzelerometer-Werte

[4] Verfügte der Quadrokopter über ein GPS, könnte er in diesem Falle auch automatisch zum Startpunkt zurückkehren

- Daraus Berechnung der aktuellen Lage und Drehung im Raum
- Einlesen der Fernsteuerkommandos
- Berechnung der Motordrehzahlen (PID-Regelung)
- Ansteuerung der Motoren durch Pulsweitenmodulation

Diese Funktionen sind im Folgenden weiter ausgeführt.

## 4.2 Anforderungen und Hardware-Ressourcen

Folgende Eckwerte bestimmen das Design:

1. Alle PWM-Signale dauern minimal 1ms und maximal 2ms.
2. Fernsteuerkommandos werden ca. alle 16 ms empfangen (≙ 62,5 Hz).
3. Die ESCs sollen alle 4 ms (≙250 Hz) einen Steuerbefehl (PWM Puls) erhalten.
4. Die ESC PWM Pulse sollen einen akzeptablen Jitter aufweisen (wenige µs).
5. Der Atmel328-Prozessor verfügt nur über 3 Timer.

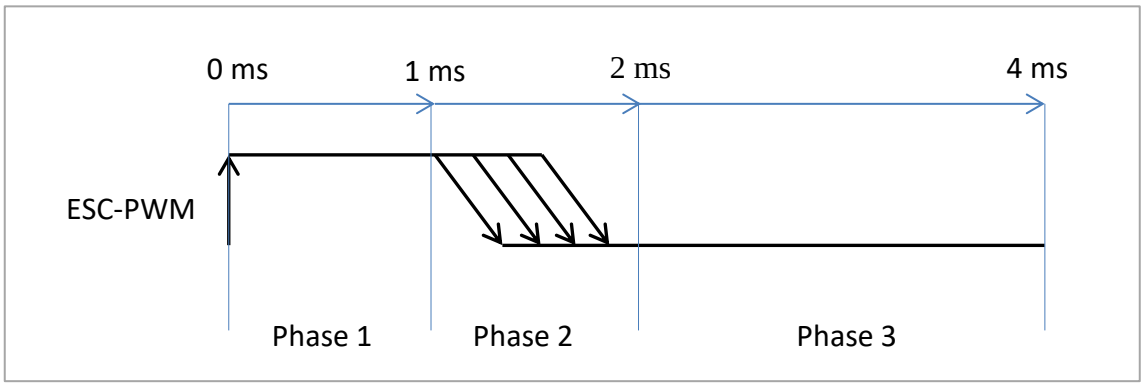
## 4.3 Timing-Übersicht

Nach erfolgreichem Start-up geht das Programm in eine Schleife:

loop {

- Ausgabe der steigenden Flanken für alle ESCs (t = 0 ms, start ESC-PWM )
- Phase 1: Abarbeiten von Funktionen, die zusammen garantiert weniger als 1 ms dauern (= minimale PWM-Zeit)
- Phase 2: Danach über einen Zeitraum von 1ms generieren der absteigenden Flanken der ESC-PWMs. Hier wird aktiv gewartet, um den Jitter zu minimieren. Es können in dieser Phase also keine anderen Aktivitäten erfolgen.
- Phase 3: Die verbleibenden minimal 2 ms werden für Aufgaben unterschiedlicher Dauer verwendet.
- Aktives warten, bis 4 ms vorbei sind.

}




Bestückung von Phase 1: Berechnung der PID-Stellwerte und der Motoransteuerung.

Bestückung von Phase 3: Auslesen und verarbeiten der IMU-Daten

Fernsteuerkommandos werden asynchron zu diesem Raster per Interrupt empfangen. Jitter entsteht, wenn Flux die abfallende Flanke eines ausgehenden ESC-Pulses nicht rechtzeitig generieren kann, weil gerade die Interrupt-Routine läuft.

## 4.4 Strukturierung des Quellcodes

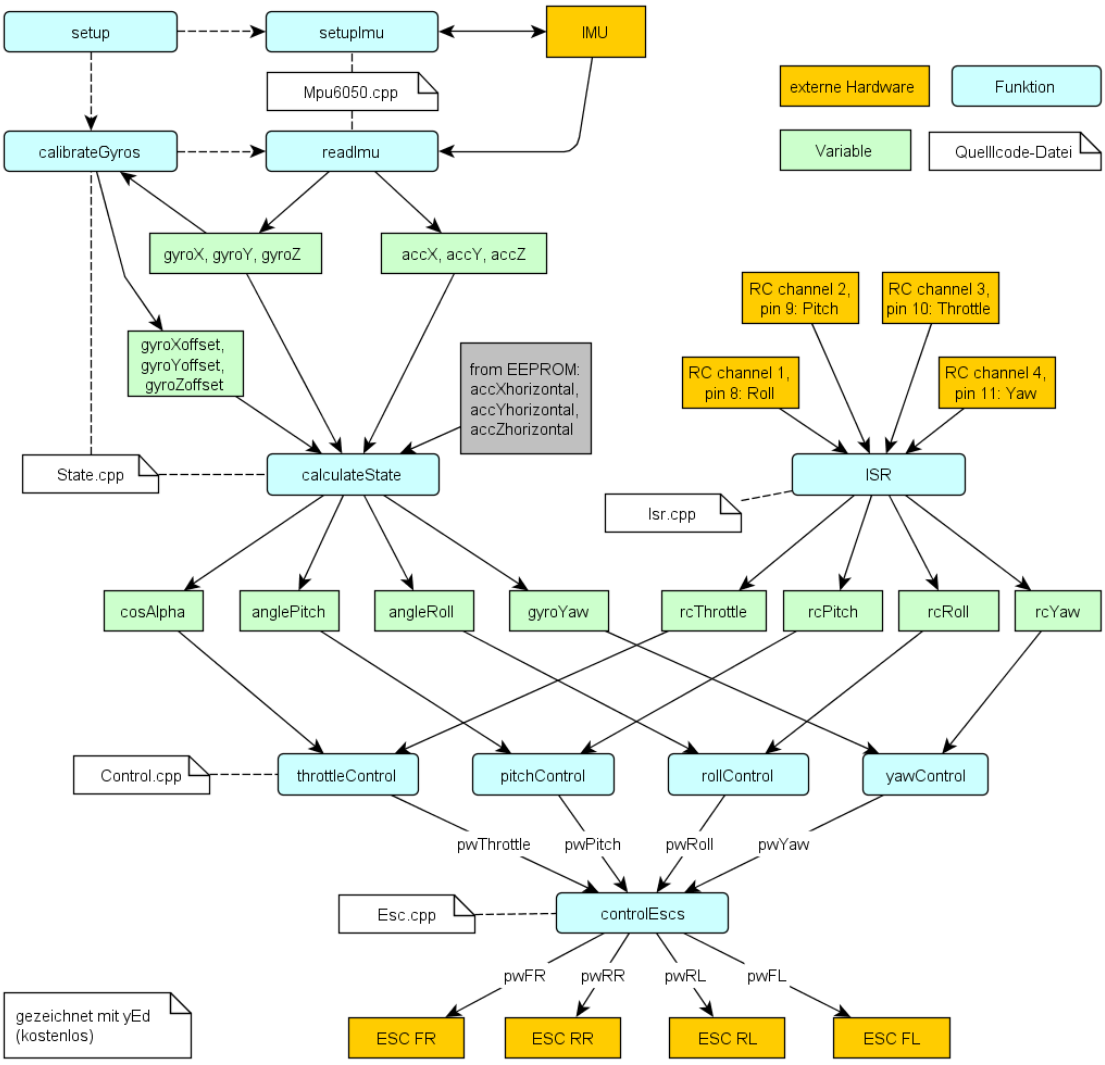
Um die Übersichtlichkeit des Programms zu erhöhen und das Auffinden spezieller Funktionen zu erleichtern, wurde der Quellcode (englisch: source code) auf mehrere Dateien verteilt. Die Arduino-IDE zeigt den Tab für die .ino Datei immer ganz links, gefolgt von den anderen Dateien des Sketches in alphabetischer Reihenfolge. Diese wird in folgender Tabelle beibehalten:

| **Datei** | **Inhalt** |
|---|---|
| Flux.ino | Die Basisfunktionen jedes Arduino-sketches: *setup* und *loop* |
| Control.cpp | Steuerungsfunktion *throttleConntrol* bzw die PID-Regelungsfunktionen *pitchControl*, *rollControl* und *yawControl* |
| Eeprom.h | Beschreibt die logische Struktur des verwendeten EPROM-Bereichs |
| Esc.c | *controlEscs* erzeugt die ESC-PWM-Pulse. |
| Flux.hpp | Definiert globale Konstanten und Strukturen. |
| Isr.cpp | *ISR*: Interrupt-Routine zur Bestimmung der Fernsteuerungs-PWM-pulse |
| Mpu6050.cpp | Mit *setupImu* wird die IMU konfiguriert, mit *readImu* deren Daten gelesen. |
| Service1.cpp | Enthält u.a. das zentrale Menü der Wartungsfunktionen |
| Service2.cpp | Hier sind die Wartungsfunktionen selbst realisiert. |
| State.cpp | In *calculateState* werden die aktuellen Zustandsgrößen ermittelt. |
| Util.cpp | Hilfsfunktionen für die Nutzung des Arduini EEPROMs |

## 4.5 Funktionsübersicht

Im Folgenden sind die wichtigsten C-Funktionen kurz beschrieben, um den Einstieg in Flux zu erleichtern. Hat man den einmal erworben, sollte der Quellcode ausreichend für sich sprechen, denn hier werden nur die Fakten beschrieben, die nicht unmittelbar dem Quellcode zu entnehmen sind.

Eine Übersicht über den Fluss der Daten durch die einzelnen C-Funktionen liefert folgende Grafik. Beachte, dass viele Daten über globale Variablen transportiert werden.




### 4.5.1 Start

Name: *setup* in *Flux.ino*

Input: -

Output: tune (=Tune::OFF für normalen Betrieb, sonst Wartungsfunktion)

Um die Flux-Programmstruktur einfach zu halten, wurde auf eine besondere Startup-Sicherung verzichtet. Wenn man sich an folgende feste Reihenfolge hält, ist m.E. ausreichend Sicherheit gegeben:

- Einsetzen des Akkus in den Rahmen
- Verbinden des LiPo-Warners mit dem Akku
- Einschalten der Fernsteuerung. Diese gibt Warntöne von sich, falls sich der der Gashebel nicht in Nullstellung befindet oder Schalter nicht auf „Aus“ stehen. Erst wenn alle Warntöne verstummen, kann der nächste Schritt erfolgen:
- Verbindung des Akkus (XT60-Stecker) d.h. Herstellung der Stromversorgung. Achtung: Jetzt muss der Quadrokopter in Ruhe liegen, weil die Gyroskope kalibriert werden!

- Abwarten, bis alle Töne der Motoren verklungen sind und die Lampe am Arduino blinkt. (Falls sie es nicht tut, liegt ein Fehler vor!)[5]
- Jetzt kann man Gas geben und abheben!

Falls die Prüfung der EEPROM-Inhalte (siehe unten) beim Startup einen Fehler erkennt, blinkt die LED langsam und der Quadrokopter kann nicht starten!

### 4.5.2 Hauptschleife

Name: *loop* in *Flux.ino*

Input: -

Output: tune (=Tune::OFF für normalen Betrieb, sonst Wartungsfunktion)

Jede Schleife, welche alle LOOP_PERIOD Millisekunden ausgeführt wird, verläuft nach demselben Muster (vgl. 4.3 Timing-Übersicht):

1. Die Startzeit der aktuellen Schleife wird gelesen, um zu wissen, wann sie später endet.
2. Berechnung und Ausgabe der ESC-Pulse
3. Einlesen und Auswerten der IMU-Daten
4. Aktives Warten bis LOOP_PERIOD abgelaufen ist.

Wird die Schleife wie erwartet in der geplanten Zeit durchlaufen, blinkt die LED schnell, d.h. etwa 4 Mal pro Sekunde. Falls die Laufzeit des Programms die LOOP_PERIOD überschreiten sollte, erlischt die LED dauerhaft. Falls man also Erweiterungen am Programm vornehmen sollte, ist auf das Blinken der LED zu achten.

### 4.5.3 Einstellen der Inertialen Messeinheit

Name: *setupImu* in *Mpu6050.cpp*

Input: -

Output: -

### 4.5.4 Lesen der Gyroskop/Accelerometer-Werte

Name: *readImu* in *Mpu6050.cpp*

Input: Registerinhalte der IMU

Output: *gyroX, gyroY, gyroZ, accX, accY, accZ, temperature*

Maßeinheiten:

[*gyroX*] = [*gyroY*] = [*gyroZ*] = 1 / (65,5 °/s) d.h. 1 LSB entspricht 1/65 Winkelgrad pro Sekunde.

Die Vorzeichen der ausgegebenen Gyro-Signale ist wie folgt:

- Eine positive Drehung um die x-Achse bewirkt positive Werte von *gyroX*.
- Eine positive Drehung um die y-Achse bewirkt negative Werte von *gyroY*.
- Eine positive Drehung um die z-Achse bewirkt negative Werte von *gyroZ*.

Die Einheiten von *accX, accY, accZ* sind ohne Belang, da nur deren Verhältnis zueinander betrachtet wird. Die Daten werden über den I2C-Bus gelesen.

[5] Eine mögliche Erweiterung von Flux könnte hier den Empfang der Fernsteuerpulse prüfen und beim Ausbleiben ein besonderes Blinkmuster aktivieren.

### 4.5.5 Einlesen der Fernsteuerkommandos

Name: ISR (Interrupt service routine) in *isr.cpp*

Input: Zustandsänderungen an den Pins des Fernsteuerempfängers = PWM

Output: globale Variable *rcThrottle, rcPitch, rcRoll, rcYaw*. (Werte zwischen 1000-2000)

Die vier Kanäle der Fernsteuerung liefern PWM-Signale, deren Dauer vermessen werden muss, um daraus die Positionen der Steuerknüppel zu erfassen.

Weil die Fernsteuerkommandos asynchron empfangen werden, bietet sich eine Vermessung der Pulsbreiten durch Hardware-Interrupts an. Der Code wurde weitgehend von [JB] übernommen.

### 4.5.6 Berechnung des Systemzustandes

Name: *calculateState* in *State.cpp*

Input: *gyroX, gyroXOffset , gyroY, gyroYOffset , gyroZ, gyroZOffset , accX, accXhorizontal , accY, accYhorizontal , accZ, accZhorizontal*

Output: *anglePitch, gyroPitch, angleRoll, gyroRoll, gyroYaw, cosAlpha*

Unter Systemzustand verstehen wir aktuell die aktuellen Neigungswinkel pitch und roll. Sie werden durch Verschmelzung der Gyro- und Akzelerometerdaten gewonnen (Sensor-Fusion). Aus den Akzelerometerdaten wird noch *cosAlpha* berechnet, der Kosinus der Neigung des QC gegen die Senkrechte. Hinzu kommt die aktuelle Drehung um die Hochachse als Winkelgeschwindigkeit[6].

Maßeinheiten:

[*anglePitch*], [*angleRoll*] = grad

[*gyroPitch*] = [*gyroRoll*] = [*gyroYaw*] = grad/s

Eine Sensor-Fuison (s.o.) wird benötigt, weil die Gyroskop- und Akzelerometer-Sensoren komplememtäre Eigenschaften haben. Mithilfe der Akzelerometer lässt sich die Lage (pitch, roll) des Quadrokopters genau berechnen, jedoch werden dessen Einzelmessungen stark durch Vibrationen der Propeller gestört. Man muss also erst über viele Werte mitteln, bevor man eine Aussage über die genaue Lage machen kann. Dies verträgt sich aber nicht mit den Anforderungen der Regelung, welche zeitnahe (wenige Millisekunden Verzögerung) Informationen verlangt. Die Gyroskopsensoren hingegen sind relativ unempfindlich gegen diese Vibrationen. Will man aber die genaue Lage ermitteln, summieren sich Rechenfehler bei der Integration und eine genaue Positionierung auf z.B. die Waagerechte wird unmöglich. Als Kompromiss geht man so vor, dass man schnelle Winkeländerungen auf Basis der Gyroskopsensoren berechnet, deren Drift aber mit den gemittelten Werten der Akzelerometersensoren kompensiert.

Genauere Informationen entnimmt man [DAA], Gleichung 9.38.

### 4.5.7 PID-Regelung Pitch/Roll

Name: *pitchControl* bzw. *rollControl* in *Control.cpp*

Input: Daten der Fernsteuerung: *rcPitch, anglePitch* bzw. *rcRoll, angleRoll*

[6] Später könnte man noch Positions- und Flugrichtungsdaten hinzufügen, welche z.B. mit einem Magnetometer, Barometer und GPS-Empfänger bestimmt werden.

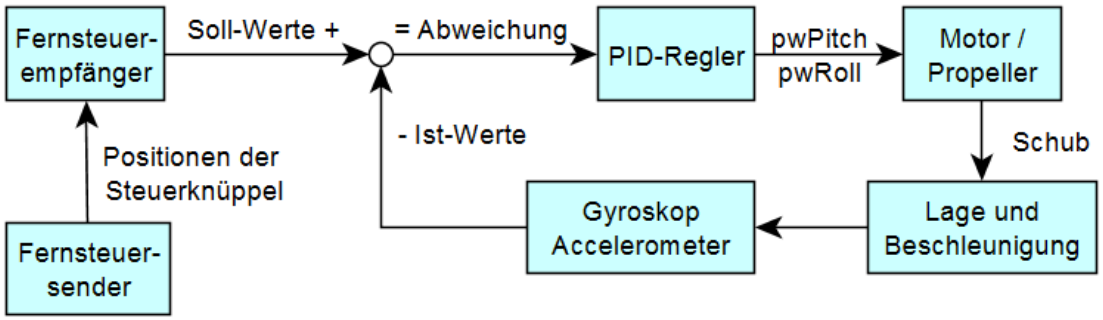
Output: Steuergrößen für die Motoren als Rückgabewerte: *pwPitch, pwRoll*

Bei der Steuerung der Nick-und Rollwinkel treffen zwei gegensätzliche Wirkungen aufeinander:

1. Um den Quadrokopter in horizontaler Richtung zu steuern, muss er sich entsprechend neigen, um Fahrt aufzunehmen. (Wirkung der externen Fernsteuerung)
2. Das automatische Leveling d.h. die Regelung auf die horizontale Lage hingegen versucht, jede Neigung zu vermeiden. (Wirkung der internen Lageregelung)

Dieser Konflikt wird dadurch aufgelöst, dass durch die Fernsteuerung ein Neigungswinkel als Führungsgröße vorgegeben wird. Befindet sich der rechte Steuerknüppel in Mittelstellung d.h. in der Neutralposition, liegen die Soll-Werte (=Führungsgrößen) der Winkel bei null.

Das folgende Diagramm zeigt die Wirkungsflüsse der Regelung:




Für Pitch und Roll kommt dieselbe PID-Regelung[7] zum Einsatz, daher werden die zwei Funktionen gemeinsam betrachtet. Der Ausgang des Reglers ist eine Summe aller drei Bestandteile:

Output = kp * Abweichung + ki * Summe der Abweichungen + kd * Änderung der Abweichung

Die Konstanten kp, ki und kd werden aus dem EEPROM gelesen (*eeprom.**).

| Die Wahl dieser Konstanten bestimmt die Lage-Stabilität des Quadrokopters! |
|---|

Achtung wegen der Vorzeichen:

- Bewegt man den Pitch-Knüppel nach vorne, wird die Pulsbreite am Empfänger größer. Jedoch soll dies eine Verringerung des Pitch-Winkels erwirken. Daher muss am Eingang von *pitchControl* das Vorzeichen vertauscht werden. Am Ausgang gilt: Will der Regler den Nickwinkel vergrößern, müssen die vorderen Propeller schneller laufen, die hinteren langsamer.
- Bewegt man den Roll-Knüppel nach rechts, wird die Pulsbreite am Empfänger größer. Dies soll eine Vergrößerung des Pitch-Winkels erwirken. Keine Vorzeichenumkehr erforderlich. Am Ausgang gilt: Will der Regler den Rollwinkel vergrößern, müssen die linken Propeller schneller laufen, die rechten langsamer.

Der I-Anteil des Reglers summiert Regelungsfehler. Um sicherzustellen, dass Mess- und Rechenfehler sich nicht langfristig anhäufen (akkumulieren), werden sie mit einer Zeitkonstante von T = 4 Sekunden auf 1/e d.h. 0.37 reduziert anhand folgender Rechnung:

Sei y die Summe der Regelungsfehler. Wird sie pro Durchlauf mit dem Faktor $\alpha < 1$ multipliziert, so hat sie nach n Schritten den Wert 1/e erreicht:

$$y(n) = \alpha * y(n-1) \qquad (1)$$

[7] Siehe z.B. https://de.wikipedia.org/wiki/Regler#PID-Regler

$y(n) = \alpha^n * y(0)$ (2)

$y(n) = 1/e * y(0)$ (3)

$n = T/\Delta t$ (4)

Aus (2) und (3) folgt:

$1/e = \alpha^n$ (5)

oder $e^{-1} = e^{n* \ln \alpha}$ (6)

somit $-1 = n* \ln \alpha$ (7)

$\alpha = e^{-1/n}$ (8)

$\alpha = e^{-\Delta t/T}$ (9)

Für $\Delta t = 4$ ms ergibt sich $\alpha = 0{,}999$.

### 4.5.8 PI-Regelung Yaw

Name: *yawControl* in *Control.cpp*

Input: *rcYaw*, *gyroYaw*

Output: Steuergröße für die Motoren als Rückgabewerte: *pwYaw*

Diese Funktion liefert eine Steuergröße für die Motoren als Ergebnis einer PI-Regelung. Achtung: Hier wind nicht der Gierwinkel geregelt, sondern die Gierwinkelgeschwindigkeit. Wenn der Yaw-Knüppel in der Neutralstellung steht, soll sie Null sein.

### 4.5.9 Gas-Steuerung

Name: *throttleControl* in *Control.cpp*

Input: *rcThrottle*, *cosAlpha*

Output: *pwThrottle*

Diese Funktion liefert eine Steuergröße für die Motoren ohne Regelung. Eigentlich ist dieser Wert direkt proportional zur Stellung es entsprechenden Steuerknüppels zu sehen. Um aber den Verlust an Auftrieb durch erwünschte Neigung des Quadrokopters zu kompensieren, muss entsprechend mehr Gas gegeben werden (* 1/cos α).[8]

### 4.5.10 Ansteuerung der Motoren

Name: *controlEscs* in *Esc.cpp*

Input: *pwThrottle, pwPitch, pwRoll, pwYaw, tune*

Output: entsprechend lange Pulse an die ESCs

Die Variablen wurden so gewählt, dass sich die Steuerung der Einzelmotoren sehr übersichtlich aus folgender Matrixmultiplikation ergibt:

$$\begin{pmatrix} pwFR \\ pwRR \\ pwRL \\ pwFL \end{pmatrix} = \begin{pmatrix} + & + & - & + \\ + & - & - & - \\ + & - & + & + \\ + & + & + & - \end{pmatrix} \cdot \begin{pmatrix} pwThrottle \\ pwPitch \\ pwRoll \\ pwYaw \end{pmatrix}$$

… wobei ´+´ für eine 1 und ´–´ für -1 steht.

[8] Später könnte hier auch eine Höhenregelung per Barometer oder GPS erfolgen.

Die Ausgabe eines ESC-PWM Signals dauert maximal 2ms. Sollen alle 4 ESCs innerhalb von 4 ms aktualisiert werden, so muss deren Ansteuerung also zeitlich überlappend erfolgen! Siehe das Timing-Diagramm oben.

Der Abfall der Batteriespannung wird bei Flux nicht mit mehr Gas kompensiert. Bei [JB] lag die Übersteuerung lediglich bei 12,5%. Es erschien mir eher wünschenswert, dass der Verlust der Spannung den Piloten auf das Ende der Flugzeit hinweist: Er muss mehr Gas geben, um die Höhe zu halten.

Die Variable *tune* dient lediglich dazu, Wartungsfunktionen zu realisieren, siehe unten.

Für die Beantwortung folgender Frage lohnt sich ein kleiner Abstecher in die Mathematik:

*Wie kann es sein, dass sich die vier Steuerungen bzw. Regelungen von throttle, pitch, roll und yaw, von denen jede gleichzeitig Einfluss auf alle vier Motoren nimmt, nicht gegenseitig behindern?*

Dass es tatsächlich funktioniert, erscheint zunächst verblüffend. Doch man kann sich das dadurch erklären, dass die vier Spaltenvektoren obiger Matrix sämtlich senkrecht aufeinander stehen d.h. ihr Skalarprodukt gleich Null ist. Was bedeutet, dass die Verlängerung oder Verkürzung des einen Vektors (durch die Eingangsvariablen = rechter Vektor) keinen Einfluss auf die Länge der anderen Vektoren hat. Man kann die gegenseitigen Beeinflussungen natürlich auch einzeln prüfen. So sieht man z.B., dass der Gesamtdrehimpuls nicht durch das Kippen oder Gasgeben nicht verändert wird. Eine Konsequenz dieser Überlegungen ist, dass man keinen Trikopter (mit starren Motoren) bauen kann, denn dann hätte die Matrix nur 3 Zeilen und die vier Spaltenvektoren könnten nicht alle senkrecht aufeinander stehen.

### 4.5.11 Kalibration der Gyroskope

Name: *calibrateGyros* in *State.cpp*

Input: *gyroX, gyroY, gyroZ*

Output: *gyroXOffset, gyroYOffset, gyroZOffset*

Für unsere Zwecke reicht eine Einpunkt-Kalibration (offset correction) aus. Sie wird vor jedem Start vorgenommen.

### 4.5.12 EEPROM

Das im Arduino-Microcontroller verfügbare EEPROM speichert Daten außerhalb des eigentlichen Programms. Sie können mit den Wartungsfunktionen (s.u.) verändert und angezeigt werden. Die logische Struktur des EEPROMs ist in der Datei *Eeprom.h* hinterlegt.

# 5 Wartungsfunktionen

Alle wesentlichen Prüfungen und Einstellungen können direkt mit dem Flux-Programm vorgenommen werden. Eine Neuprogrammierung ist nicht erforderlich. Der seriellen Monitor der Arduino-IDE (Baudrate = 57600) dient als Benutzerschnittstelle.

Die Wartungsfunktionen werden aktiviert, wenn beim Reset die Arduino-Kontakte A0 und A1 elektrisch verbunden sind.

| Zu diesem Zweck fertigt man sich z.B. aus Pin-Headers ein kleines Werkzeug an. |
|---|

Beachte: Die im Folgenden mit einem Stern (*) gekennzeichneten Funktionen modifizieren den EEPROM-Puffer.

## 5.1 Start der Wartungsfunktionen

Name: *service* in *Service1.cpp*

Input: EEPROM-Daten

Output: Ergebnis der EEPROM-Prüfung

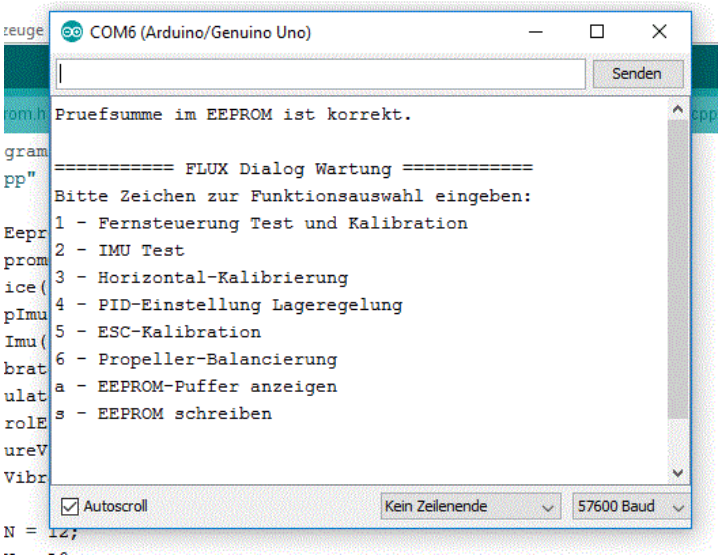
Vor jedem Start muss sichergestellt sein, dass die EEPROM-Daten fehlerfrei ausgelesen werden können. Andernfalls versagt die Lageregelung. Eine Prüfung erfolgt durch den Vergleich der berechneten mit der gespeicherten Prüfsumme (checksum). Wartungsfunktionen, welche andere Werte als *Tune::OFF* zurückliefern, benötigen die Ansteuerung der ESCs in der Hauptschleife. Sie müssen also durch einen Reset des Arduino beendet werden.

## 5.2 Menüsteuerung der Wartungsfunktionen

Name: *menu* in *Service1.cpp*

Input: -

Output: Menütext auf der Konsole




Hier kann man die gewünschten Wartungsfunktion auswählen.

## 5.3 Test und Kalibration der Fernsteuerung

Name: *remoteControlService* in *Service2.cpp*

Input: -

Output: aktuell empfangene Fernsteuerwerte

Zweck dieser Kalibration ist, dass der Quadrokopter beim Loslassen aller Hebel in der Waagerechten schwebt und sich nicht um die Hochachse dreht (ausreichend Gas vorausgesetzt). Zusätzlich soll von Flux sicher erkannt werden, wenn der Gashebel auf Null steht. Dazu werden die in Echtzeit die empfangenen Pulsweiten der Kanäle in Mikrosekunden angezeigt. Mithilfe der Trimmfunktionen auf der Fernsteuerung verfährt man wie folgt:

1. Den Gashebel auf null stellen und beide Steuerknüppel loslassen.
2. Die empfangene Pulsbreite für *throttle* auf 1000 einstellen.
3. Die Pulsbreiten der anderen Kanäle (*yaw, pitch, roll*) auf 1500 einstellen.

## 5.4 IMU Test

Name: *imuTest* in *Service2.cpp*

Input: -

Output: die berechneten Zusatndsgrößen *anglePitch*, *angleRoll*, *cosAlpha* und *gyroYaw*

## 5.5 Horizontal-Kalibrierung*

Name: *horizontalCalibration* in *Service2.cpp*

Input: -

Output: *eeprom.acc?horizontal* mit *? = X, Y, Z*

Der Quadrokopter wird auf eine exakt horizontale Fläche gestellt und steht dort in absoluter Ruhe. Die hier gemessenen Beschleunigungswerte (gemittelt über 10 sec) in x- und y-Richtung werden als Referenz für die Lageregelung im EEPROM gespeichert. Diese Werte werden dann im Flugmodus von den jeweiligen Messwerten abgezogen.

## 5.6 Einstellung der PID-Parameter*

Name: *pidGainAdjust* in *Service2.cpp*

Input: -

Output: *eeprom.pPitchRoll, eeprom.iPitchRoll, eeprom.dPitchRoll* sowie *eeprom.pYaw*, *eeprom.iYaw, eeprom.dYaw.*

Hier können nacheinander alle PID Parameter eingestellt werden.

## 5.7 ESC-Kalibration

Name: *calibrateESCs* in *Service2.cpp*

Input: -

Output: -

**Achtung 1**: Nicht alle erhältlichen ESC-Typen werden auf die nachfolgende Weise kalibriert. Besteht Zweifel, muss man sich die Betriebsanleitung des ESC-Typs besorgen.

**Achtung 2**: Auch wenn sich während der Kalibration die Motoren nicht drehen sollten, ist dringend empfohlen, vor der Ausführung alle Propeller zu demontieren!

**Achtung 3**: Diese Kalibration sollte erst erfolgen, nachdem die Kalibration der Fernsteuerung abgeschlossen ist.

Um sicher zu stellen, dass alle vier ESCs über den Bereich der Gas-Steuerung (ungefähr[9]) dieselben Drehzahlen produzieren, müssen sie kalibriert werden. Die ESCs speichern selbständig ihre Kalbrationswerte ohne Zeitbegrenzung, man muss sich also nichtweiter kümmern.

Dies ist der Ablauf: Zunächst legt man den maximalen Steuerwert d.h. 2000 µs Pulsbreite an den Eingang der ESCs und schaltet dann ihre Stromversorgung ein. Diesen Vorgang quittieren die Motoren mit ein paar Tönen. Anschließend reduziert man die Pulsbreite auf 1000 µs, was von ihnen wiederum mit Tönen quittiert wird. Danach entfernt man die Stromversorgung – und fertig!

Mit Flux gehe folgendermaßen vor:

[9] Kleinere Abweichungen werden von der Regelung kompensiert.

1. Stelle sicher, dass die Batterie nicht mit dem Quadrokopter verbunden ist und die Propeller nicht montiert sind.
2. Bringe den QC in Service-Modus und verbinde ihn mit dem Arduino-Monitor-Programm.
3. Im Flux-Service-Menü wähle die ESC-Kalibration.
4. Schalte die Fernsteuerung ein und gebe Vollgas (linker Knüppel ganz nach vorn).
5. Schließe nun die Batterie an. Falls sich jetzt ein Motor dreht, ist etwas faul im Aufbau und die Prozedur muss beendet werden.
6. Nachdem alle ESCs ihre Töne abgegeben haben, gehe mit dem Gashebel auf Null.
7. Warte, bis alle ESCs dieses Manöver mit Tönen quittiert haben, dann entferne die Batterie und schließe den Arduino-Monitor. Die ESCs sind jetzt kalibriert.

Siehe auch https://www.youtube.com/watch?v=OOplk52R4no.

## 5.8 Propeller-Balancierung

Name: *balanceProps* in *Service2.cpp*

Input: -

Output: Zahlenwerte, welche die durch Propellerunwucht erzeugten Vibrationen zeigen.

Siehe auch http://www.brokking.net/ymfc-al_main.html

Mit diesem Programm kann die Drehzahl eines zuvor ausgewählten Motors mit der Fernsteuerung eingestellt werden. Das Programm summiert laufend die von den Akzelerometern gemessenen Beschleunigungen und gibt sie als dimensionslose Zahlen aus.

Man gehe wie folgt vor:

1. Alle Propeller montieren
2. Fernsteuerung einschalten
3. Akku anschließen
4. Für jeden Motor:
    a. Balancierungsprogramm starten
    b. Motor auswählen
    c. Den Gashebel auf etwa die Mitte schieben (Achtung: der Quadrokopter darf noch abheben!)
    d. Den angezeigten Zahlenwert notieren
    e. Das Monitorfenster schließen (Beim Neustart wird der Arduino resettet.)
5. Zahlenwerte unter 100 galten bei mir als akzeptabel. Man hört auch, ob und wie ruhig ein Propeller läuft.
6. Jetzt die „lauten“ Propeller behandeln. Dazu beklebt man die eine oder andere Oberseite des Propellers mit einem größeren oder kleineren Stück haltbarem Klebeband. Anschließend zeigt das Balancierungsprogramm Erfolg oder Misserfolg. So kämpft man sich schrittweise durch alle Propeller – soweit notwendig.

## 5.9 Anzeige des EEPROM-Puffers

Name: *dumpEeprom* in *Service1.cpp*

Input: -

Output: Inhalt und Prüfsumme des EEPROMs.

In meinem Quadrokopter befinden sich folgende Werte im EEPROM (Originalausgabe des Programms):

```
Inhalt des EEPROM-Puffers:
--------------------------
Signatur = FLUX, Revision 1
-- Accelerometer zero values (values while in horizontal position)
accXhorizontal = 19
accYhorizontal = 26
accZhorizontal = 3704
-- PID values for leveling control
pPitchRoll = 2.00
iPitchRoll = 0.02
dPitchRoll = 200.00
-- PID values for yaw control
pYaw = 5.00
iYaw = 0.20
dYaw = 0.00

-- Checksum
checksum = 6DA4
```

## 5.10 Schreiben des EEPROMs

Name: *writeEeprom* in *Util.cpp*

Input: -

Output: -

Funktion: Schreibt den Inhalt des EEPROM-Puffers in das EEPROM, berechnet die Prüfsumme und schreibt auch diese in das EEPROM.

# 6 Vermischtes

## 6.1 EEPROM Verwaltung

Es steht mit 1024 Byte weit mehr Speicherplatz zur Verfügung als benötigt. Die genaue Definition des Layouts liegt der Datei *Eeprom.h*.

Layout:

| Offset | Bytes | Inhalt | Kommentar |
|---|---|---|---|
| 0 | 4 | 4 chars: „FLUX“ | Kennung |
| 4 | 1 | Version | Version des Layouts |
| 5 | | Konfig-Daten | Entsprechend der FluxEeprom-Struktur |
| Länge - 2 | 2 | Prüfsumme (Checksum) | Dient der Konsistenzprüfung |

Weil die Korrektheit der EEPROM-Daten Voraussetzung für die Flugfunktion ist, wird bei jedem Start vor dem Auslesen der Werte die Prüfsumme berechnet und mit der gespeicherten verglichen. Im Fehlerfall bleibt die Start-LED an und die Flugfunktion blockiert. Dies ist

erforderlich, weil viele Kalibrationsdaten im EEPROM hinterlegt sind und die Lageregelung mit falschen Werten „verrückt spielen“ würde und somit ein Flug unmöglich wäre.

## 6.2 IMU Konfiguration

Die MPU-6050 lässt sich in vielerlei Hinsicht konfigurieren, siehe [MPU-6050-2].

Für unsere Anwendung wählen wir mit [JB] folgende Sensitivitäten:

| Sensor | Full Scale Range | LSB Sensitivity | siehe Seite |
|---|---|---|---|
| Akzelerometer | +/- 8g | 4096 LSB/g | 15, 30 |
| Gyroskop | +/- 500 °/s | 65.5 LSB/ °/s | 14, 32 |

Die Messwerte werden als vorzeichenbehaftete 16-bit integer Zahlen ausgegeben.

Weiterhin wählen wir einen Tiefpass, um hochfrequente Störungen, wie sie von den Motoren erzeugt werden, zu unterdrücken. Dabei müssen wir einen Kompromiss eingehen: Bei tiefer Grenzfrequenz dämpfen wir zwar diese Störungen, doch verzögert sich damit auch die Signalübertragung d.h. Änderungen der Lage und Drehbewegung werden später erkannt, was zu größeren Schwankungen des QC führt.

Wir wählen mit [JB] eine Grenzfrequenz von 44 Hz, siehe [MPU-6050-2] Seite 13.

# 7 Mögliche Änderungen Erweiterungen

## 7.1 Verbesserung der Lageregelung

Aktuell hält Flux den Quadokopter stabil in der Luft und lässt ihn auch gut steuern, doch lassen sich hier noch Verbesserungen vorstellen. Dies reicht von einer Anpassung der PID-Verstärkungswerte bis hin zur Anwendung einer kaskadierten Regelstruktur. (Siehe die Ardupilot-Software, z.B. hier http://ardupilot.org/dev/docs/apmcopter-code-overview.html )

## 7.2 Piepser zum Auffinden des Quadkopters

Fliegt man auf einem Gelände mit höherem Pflanzenbewuchs, kann es schwierig werden, den Quadkopter nach Landung oder nach Absturz wieder zu finden. Daher bietet sich an, das Gas-Signal zu überwachen. Ist es für X Sekunden kleiner als z.B. THROTTLE_SAFE, beginnt ein Piezo-Summer den Standort mitzuteilen.

## 7.3 Verlängerung der LOOP_PERIOD

Dies bedeutet eine Erniedrigung der Frequenz, mit der neue Steuerdaten an die ESCs gegeben werden. Werte von 5ms oder gar 10ms sind denkbar, hat der Autor aber noch nicht ausprobiert. Der Nutzen wäre ein Gewinn an Rechenzeit für weitergehende Funktionen. Dabei könnte man die Aktivitäten aus der heutigen Phase 3 vor die Phase 1 verlegen, damit Zustandsmessung und –berücksichtigung nahe beieinander liegen.

## 7.4 Notlandung

Aktuell gibt es wenig Situationen, welche einen automatischen Abstieg des Quadkopters sinnvoll erscheinen lassen. Will man ihn programmieren, muss man u.a. einen brauchbaren

Schub (throttle/Gas) für die Motoren festlegen. Vermutlich wird man die Werte von pitch, roll und yaw zu Null setzten.

## 7.5 Schutz vor Ausfall der Fernsteuerungsdaten

Aktuell ist unerforscht, was passiert, wenn die Funkverbindung ausfällt. Vermutlich wird der Empfänger eine Zeit lang (wie lange?) die zuletzt empfangenen Werte weiterleiten. Dann würde der Quadrokopter u.U. unerwünscht weiterfliegen. Man könnte z.B. programmieren, dass wenn X Sekunden (weitgehend) identische Werte empfangen werden, eine Notlandung eingeleitet wird.

## 7.6 Schutz vor Ausfall der Fernsteuerungssignale

Falls die Interrupts der Fernsteuerung aussetzen (was auch durch Kontaktprobleme hervorgerufen werden könnte), könnte eine Notlandung initiiert werden.

## 7.7 Erweiterung der Start-Prozedur

Mehr Sicherheit beim Starten kann eine Logik bringen, welche z.B. vor jedem Start ein bestimmtes Manöver des Gashebels erfordert, z.B. einmal Vollgas und wieder Null.

## 7.8 Kalibration der Gyros als Wartungsfunktion

Will man den Quadrokopter aus einer bewegten Situation heraus starten, kann man die Gyroskope nicht jedes Mal beim Start-up (Nullpunkt-) kalibrieren. Denn dazu müssten sie in Ruhe sein. Also bietet sich eine Wartungsfunktion an – in der Hoffnung, dass der Nullpunkt nicht zu stark mit der Temperatur oder anderen Einflussgrößen variiert.

## 7.9 Optimierung per Telemetrie

Leider kann man nicht verfolgen, wie die diversen Zustands- und Steuergrößen während des Fluges verlaufen. Dies wäre aber notwendig, um Schwachstellen z.B. der Regelung aufzuspüren und zu beseitigen. Hier könnte eine Funkübertragung vom Quadrokopter z.B . über die serielle Schnittstelle zu einer Datenerfassung in einem Laptop helfen. Z.B verbindet man letzteren mit einem zweiten Arduino, der als Empfänger fungiert. Eine mögliche Lösung zeigt dieses Video: https://www.youtube.com/watch?v=vqRqtgvltOI.

## 7.10 Erweiterung der Navigation

Schließlich ist noch denkbar, mit einem Barometer, einem Magnetometer und GPS-Empfänger die Steuerungs- und Regelungsfunktionen zu erweitern. Insbesondere das Barometer kann den Komfort erhöhen, weil eine Stabilisierung der Flughöhe (altitude hold) doch besondere Anforderungen an den Piloten stellt. Die anderen Sensoren helfen, den Quadkopter aus Situtionen zurückzuholen, bei der der Pilot dessen Orientierung nicht mehr (richtig) ausmachen kann (return to home).

# 8 Quellen

[JB] http://www.brokking.net/ymfc-al_main.html

[MPU-6050-1] MPU-6000/MPU-6050 Product Specification (MPU-6050_DataSheet_V3 4.pdf)

[MPU-6050-2] MPU-6000/MPU-6050 Register Map and Descriptions (RM-MPU-6000A.pdf)

[DAA] “Dokumentation Avionic Architecture” der Projektgruppe “Avionic Architecture” der Fakultät II - Informatik, Wirtschafts- und Rechtswissenschaften, Masterstudiengang Informatik an der Carl-von-Ossietzky-Universität Oldenburg vom 15. Mai 2015.