
DIPLOMARBEIT

Gesamtprojekt

FPV-Drohne

Elektronik FPV-Drohne

Marcel Bieder 5AHEL

Softwareentwicklung FPV-Drohne

Maximilian Lendl 5AHEL

CAD-Entwicklung & Datenübertragung

Ben Heinicke 5AHEL

Entwicklung einer App für Android & Videoübertragung

Sebastian Hinterberger 5AHEL

Betreuer: Dipl.-Ing. Josef Reisinger

Schuljahr 2023/24

Abgabevermerk:

Datum: 02.04.2024

übernommen von:

Höhere Technische Bundeslehranstalt Hollabrunn

Höhere Lehranstalt für Elektronik und Technische Informatik

EIDESSTATTLICHE ERKLÄRUNG

Ich erkläre an Eides statt, dass ich die vorliegende Diplomarbeit selbständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche erkenntlich gemacht habe.

Marcel Bieder

Maximilian Lendl

Ben Heinicke

Sebastian Hinterberger

HINWEISE

Die vorliegende Diplomarbeit wurde in Zusammenarbeit mit der Firma **Dronetech Austria** ausgeführt.

Die in dieser Diplomarbeit entwickelten Prototypen und Software-Produkte dürfen ganz oder auch in Teilen von Privatpersonen oder Firmen nur dann in Verkehr gebracht werden, wenn sie diese selbst geprüft und für den vorgesehenen Verwendungszweck für geeignet befunden haben.

Es wird keinerlei Haftung übernommen für irgendwelche Schäden, die aus der Nutzung der hier entwickelten oder beschriebenen Bestandteile des Projekts resultieren.

Für alle Entwicklungen gilt die GNU General Public License [<http://www.gnu.org/licenses/gpl.html>] der Free Software Foundation, Boston, USA in der Version 3.

Die Diplomarbeit erfüllt die "Standards für Ingenieur- und Technikerprojekte" entsprechend dem Rundschreiben Nr. 60 aus 1999 des BMBWK (GZ.17.600/101-II/2b/99).
[https://www.bmb.gv.at/ministerium/rs/1999_60.html]

SCHLÜSSELBEGRIFFE

Open Source
Hardwaredesign
Softwaredesign
CAD-Entwicklung
App-Entwicklung
Datenübertragung
Videoübertragung
HAL

DANKSAGUNGEN

Besonderer Dank gilt unserem stets hilfsbereiten Betreuer Dipl. -Ing. Josef Reisinger, welcher uns während unseres gesamten Entwicklungsprozesses unterstützt hat. Weiters möchten wir uns bei BEd Wolfgang Kauer und Dipl. -Ing Mag. Michael Wihsböck bedanken, die uns durch ihre Expertise bei der Platinen-Entwicklung unterstützen konnten. Des Weiteren möchten wir uns bei Dipl. -Ing Gerald Stoll bedanken, der uns beim Netzwerkaufbau unterstützen konnte. Außerordentlicher Dank gebührt unserem Sponsor Dronetech Austria, besonders Daniel Stoiber, für die tatkräftige Unterstützung bei der Entwicklung der FPV-Drohne.

DIPLOMARBEIT

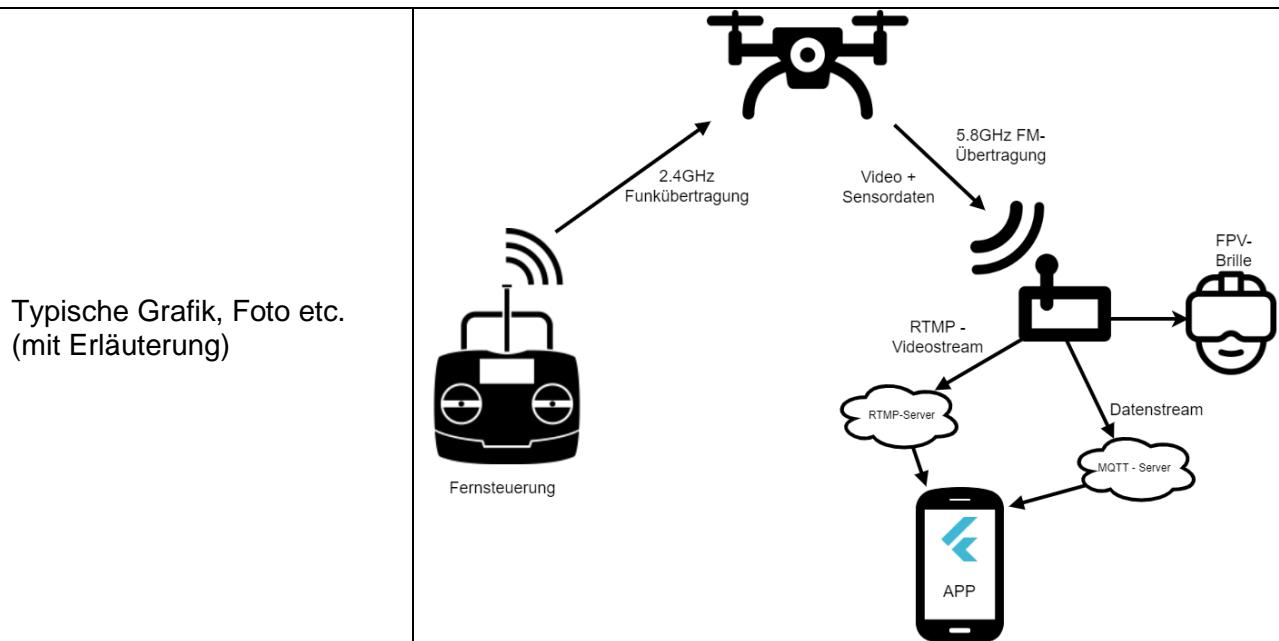
DOKUMENTATION

Namen der Verfasser/innen	Marcel Bieder Ben Heinicke Sebastian Hinterberger Maximilian Lendl
Jahrgang Schuljahr	5AHEL
Thema der Diplomarbeit	FPV-Drohne
Kooperationspartner	Dronetech Austria

Aufgabenstellung	Das Ziel der Diplomarbeit ist die Realisierung einer leistungsstarken FPV-Drohne (first person view), die mittels einer Fernsteuerung gesteuert wird, während man durch eine, an der Drohne, installierten Kamera das aktuelle Bild anschauen kann. Dieses Livebild soll auf einer FPV-Brille und in unserer selbstprogrammierten App dargestellt werden. Die gesamte Steuerelektronik und Software wird selbst entworfen und entwickelt.
------------------	---

Realisierung	Der Drohnenrahmen, sowie der ESC (Electronic Speed Controller), Motoren, VTX (5,8GHz Video Transmitter) und die Kamera wurden zugekauft. Zusätzlicher Rotorschutz und Stützen wurden mithilfe von Fusion360 designt und mit Hilfe eines 3D-Druckers gefertigt. Sämtliche Steuerelektronik wurde selbstständig in Altium Designer 22 designt und entwickelt. Dazu zählen die Sensorplatine und die Hauptplatine mit Mikrocontroller, der die Signale der Fernsteuerung einliest, dem ESC die gewünschte Motordrehzahl sendet und wichtige Sensordaten, wie Batteriespannung, Lagewinkel, Temperatur und Flughöhe, einliest. Die dazugehörige Software des Mikrocontrollers wurde selbstständig in Keil µVision5 entwickelt und in der Sprache C mithilfe von HAL (Hardware Abstract Layer) ausprogrammiert. Die gesamte Drohne wird mit einem 6s-Akku angetrieben. Diese 25,2V werden mit Fixspannungsreglern auf niedrigere Spannungen heruntergeregt, um den Mikrocontroller und die Sensoren zu versorgen. Die installierte Kamera sendet ein Signal zur VTX, um das Livebild auf der FPV-Brille und in der Visualisierungsapp darzustellen. Ebenso werden die vom Mikrocontroller eingelesenen Messdaten über die VTX mitgeschickt, um diese in einer Datenbank zu speichern und in der Visualisierungsapp mithilfe von Zeigerinstrumenten darzustellen.
--------------	---

Ergebnisse	Das Ergebnis der Diplomarbeit ist eine voll funktionsfähige, leistungsstarke FPV-Drohne, die über eine Fernsteuerung gesteuert wird. Der Flug kann mithilfe der installierten Kamera und einer FPV-Brille verfolgt werden. Die Sensordaten werden in einer Datenbank gespeichert und in einer Visualisierungsapp grafisch dargestellt. Es besteht außerdem die Möglichkeit, eine weitere Kamera auf die Drohne zu montieren, um hochauflösende Videos während des Fluges aufzunehmen.
------------	---



Teilnahme an Wettbewerben,
Auszeichnungen

-

Möglichkeiten der
Einsichtnahme in die Arbeit

HTL Hollabrunn
Anton-Ehrenfriedstraße 10
2020 Hollabrunn

Approbation (Datum / Unterschrift)	Prüfer/Prüferin	Direktor/Direktorin Abteilungsvorstand/Abteilungsvorständin
---------------------------------------	-----------------	--

DIPLOMA THESIS

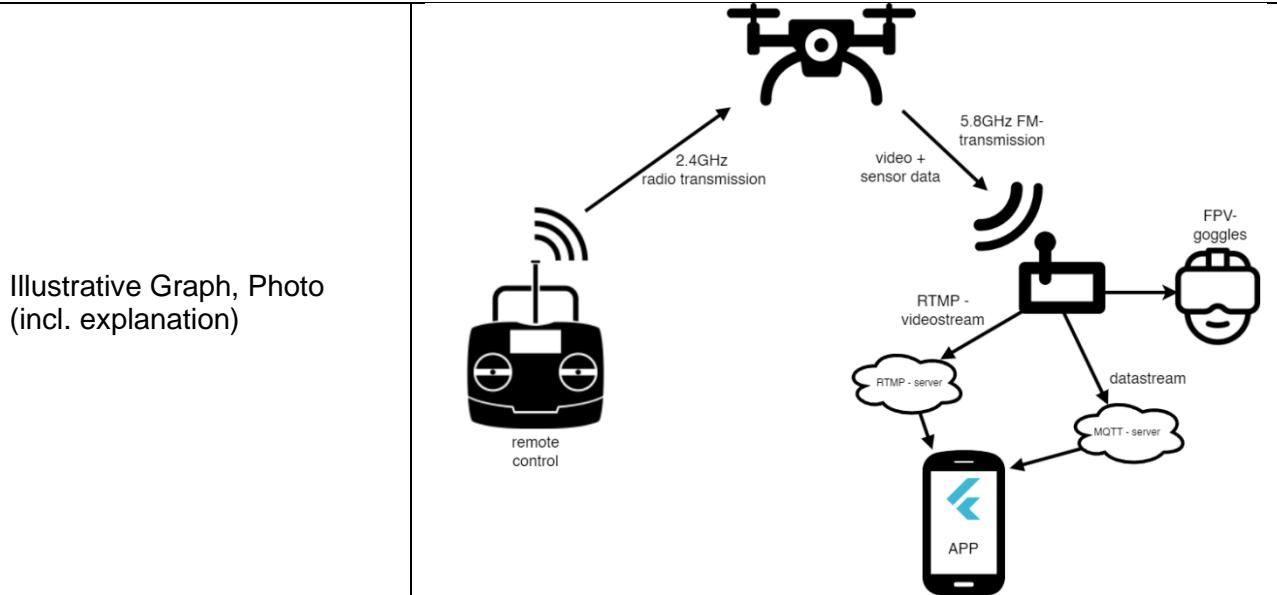
Documentation

Author(s)	Marcel Bieder Ben Heinicke Sebastian Hinterberger Maximilian Lendl
Form Academic year	5AHEL
Topic	FPV-Drone
Co-operation Partners	Dronetech Austria

Assignment of Tasks	The aim of the diploma thesis is to realize a high-performance FPV-drone (first person view drone) that is controlled remotely, while you are able to experience the live feed of the camera, installed on the drone. This live image is to be displayed on FPV goggles and in our self-programmed app. The entire control electronics and software are designed and developed in-house.
---------------------	--

Realisation	The drone frame, ESC (Electronic Speed Controller), motors, VTX (5.8GHz video transmitter) and camera were purchased. Additional rotor protection and props were designed using Fusion360 and manufactured using a 3D printer. All control electronics were designed and developed independently in Altium Designer 22. These include the sensor board and the main board with microcontroller, which reads the signals from the remote control, sends the desired motor speed to the ESC, and reads important sensor data such as battery voltage, position angle, temperature, and altitude. The associated microcontroller software was developed independently in Keil µVision5 and programmed in C using HAL (Hardware Abstract Layer). The entire drone is powered by a 6S battery. The 25.2V are regulated down to lower voltages using fixed voltage regulators to supply the microcontroller and the sensors. The installed camera sends a signal to the VTX to display the live image on the FPV goggles and in the visualisation app. The measurement data read in by the microcontroller is also sent via the VTX to save it in a database and display it in the visualisation app using gauges.
-------------	--

Results	The result of the diploma thesis is a fully functional, high-performance FPV drone that is controlled via a remote control. The flight can be tracked using the installed camera and FPV goggles. The sensor data is stored in a database and displayed graphically in a visualisation app. It is also possible to mount an additional camera on the drone to record high-resolution videos during the flight.
---------	--



Participation in Competitions
Awards

-

Accessibility of
Diploma Thesis

HTL Hollabrunn
Anton-Ehrenfriedstraße 10
2020 Hollabrunn

Approval
(Date / Sign)

Examiner

Head of College / Department

Erklärung

Die unterfertigten Kandidaten/Kandidatinnen haben gemäß § 34 Abs. 3 Z 1 und § 37 Abs. 2 Z 2 des Schulunterrichtsgesetzes in Verbindung mit den Bestimmungen der „Prüfungsordnung BMHS, Bildungsanstalten“, BGBl. II Nr. 177/2012 i.d.g.F. die Ausarbeitung einer Diplomarbeit/Abschlussarbeit mit folgender Aufgabenstellung gewählt:

FPV Drohne (Gesamtprojekt)

Individuelle Aufgabenstellungen im Rahmen des Gesamtprojektes:

- Ben Heinicke (5AHEL_24): **Datenübertragung (Bild+Messdaten) und 3D Design**
- Marcel Bieder (5AHEL_24): **Hardware FPV Drohne**
- Maximilian Lendl (5AHEL_24): **Software FPV Drohne**
- Sebastian Hinterberger (5AHEL_24): **Visualisierungs App**

Die Kandidaten/Kandidatinnen nehmen zur Kenntnis, dass die Diplomarbeit/Abschlussarbeit in eigenständiger Weise und außerhalb des Unterrichtes zu bearbeiten und anzufertigen ist, wobei Ergebnisse des Unterrichtes mit einbezogen werden können, die jedenfalls als solche entsprechend kenntlich zu machen sind.

Die Abgabe der vollständigen Diplomarbeit/Abschlussarbeit hat in digitaler und in zweifach ausgedruckter Form bis spätestens **02.04.2024** beim zuständigen Betreuer/der zuständigen Betreuerin zu erfolgen.

Die Kandidaten/Kandidatinnen nehmen weiters zur Kenntnis, dass ein Abbruch der Diplomarbeit/Abschlussarbeit nicht möglich ist.

Kandidaten/Kandidatinnen:

Ben Heinicke (5AHEL_24)

Marcel Bieder (5AHEL_24)

Maximilian Lendl (5AHEL_24)

Sebastian Hinterberger (5AHEL_24)

Datum und Unterschrift bzw. Handysignatur:

18.09.2023 Ben Heinicke

18.09.2023 Bieder Marcel

18.09.2023 Lendl Maximilian

18.09.2023 Hinterberger Sebastian

FPV – Drohne

Verlauf

- 21.09.2023: Thema wurde eingereicht
- 22.09.2023: Thema wurde von Betreuer akzeptiert
- 22.09.2023: Thema wurde von AV akzeptiert
- 28.09.2023: Thema wurde von Direktion akzeptiert
- 07.11.2023: Wurde aus dem Vorsystem übernommen.

Schule

Höhere Technische Bundeslehranstalt HOLLABRUNN

Abteilung

Hauptverantwortlich: Elektronik und technische Informatik

AV

Hauptverantwortlich: Dipl.-Ing Wilfried Trollmann

Abschließende Prüfung

2024

Betreuer/innen

Hauptverantwortlich: Dipl.-Ing Josef Reisinger

Ausgangslage

Da in der Vergangenheit bereits mehrere Drohnen im Rahmen einer Diplomarbeit gebaut wurden, wollten wir etwas Neues bauen. Eine FPV-Drohne, eine Drohne mit Live-Video-Übertragung, um das Gefühl zu bekommen, als würde man live mitfliegen.

Projektteam (Arbeitsaufwand)

Name	Individuelle Themenstellung	Klasse	Arbeitsaufwand
Marcel Bieder	Elektronik FPV-Drohne	5AHEL_24	>180 Stunden
Maximilian Lendl	Softwareentwicklung FPV-Drohne	5AHEL_24	>180 Stunden
Ben Heinicke	CAD-Entwicklung & Datenübertragung	5AHEL_24	>180 Stunden
Sebastian Hinterberger	Entwicklung einer App für Android & Videoübertragung	5AHEL_24	>180 Stunden

Inhaltsverzeichnis

1	Einleitung	19
1.1	Projektziel	19
1.2	Gesamtüberblick	19
2	Grundlagen Drohnenflug	20
2.1	Inertial Measurement Unit (IMU)	20
2.1.1	Motion Processing Unit (MPU).....	20
2.1.1.1	Gyroskop.....	20
2.1.1.2	Accelerometer.....	20
2.1.2	Barometer - BMP280	21
2.2	Lagewinkel.....	21
2.2.1	Throttle	21
2.2.2	Pitch	21
2.2.3	Roll	22
2.2.4	Yaw	22
2.2.5	Komplementärfilter.....	23
2.3	PID-Regler	25
2.3.1	Proportionalglied (P-Glied).....	26
2.3.2	Integralglied (I-Glied)	26
2.3.3	Differenzialglied (D-Glied).....	27
2.3.4	PID-Regler.....	27
3	Mechanischer Aufbau.....	28
3.1	FPV – Drohne Gesamtaufbau	28
3.2	Drohnen-Rahmen Innenaufbau.....	29
3.3	3D – Modelle	30
3.3.1	Rotoren-Schutz.....	30
3.3.2	Groundstation	31
4	Elektronik FPV-Drohne	33
4.1	Allgemeines	33
4.1.1	Grundlegendes Hardwarekonzept	33
4.2	Akku.....	33
4.2.1	Allgemeines	33
4.2.2	Akkuauswahl	34
4.2.2.1	6S 1300mAh 120C LIPO.....	34
4.2.2.2	5S 2200mAh 95C LIPO.....	35
4.2.3	Aufbau eines LIPO-Akkus.....	35
4.2.4	Ladekurve einer LIPO-Zelle	36
4.2.5	Entladekurve einer LIPO-Zelle	37
4.2.6	Entladezeiten.....	37
4.2.6.1	6S 1300mAh 120C LIPO.....	37
4.2.6.2	5S 2200mAh 95C LIPO.....	38
4.2.6.3	Fazit	38
4.2.6.4	Ergebnis.....	38
4.3	Flight Controller.....	38

4.3.1 Allgemeines	38
4.3.2 Spannungsversorgungskonzept.....	39
4.3.2.1 12V Step Down Converter.....	40
4.3.2.2 5V Step Down Converter.....	41
4.3.2.3 3,3V Step Down Converter.....	42
4.3.2.4 Versorgungsanschlüsse	43
4.3.2.5 Spannungsüberwachung.....	43
4.3.2.6 Störfilter für Mikrocontroller	44
4.3.2.7 Störfilter für VDDA-Pin	45
4.3.3 Mikrocontroller	45
4.3.3.1 Allgemeines	45
4.3.3.2 STM32H7A3RGT6	45
4.3.3.3 Schematic	48
4.3.4 Altium PCB Design	53
4.3.4.1 Allgemeines	53
4.3.4.2 Gesamtübersicht.....	53
4.3.4.3 Top Layer.....	54
4.3.4.4 Bottom Layer.....	54
4.3.4.5 Altium 3D Ansicht.....	55
4.3.4.6 Platzierung der Bauteile	56
4.3.5 Pinbelegung.....	63
4.3.6 Bestückungsplan	64
4.3.6.1 Top Layer.....	64
4.3.6.2 Bottom Layer.....	65
4.3.7 Betriebsmodi.....	66
4.3.7.1 5V Betrieb	66
4.3.7.2 Akkubetrieb	66
4.3.8 Testen	67
4.3.8.1 Spannungswandler	67
4.3.8.2 Programmierung	67
4.3.8.3 Terminal.....	67
4.3.8.4 Weitere Komponenten.....	67
4.4 Motoren.....	68
4.4.1 Allgemeines	68
4.4.2 Aufbau	68
4.4.3 Mechanische Daten	68
4.4.4 Elektrische Daten	69
4.4.5 Testdaten im Betrieb.....	69
4.4.6 Zugkraft des Motors.....	69
4.5 Electronic Speed Controller (ESC)	70
4.5.1 Allgemeines	70
4.5.2 Aufbau	70
4.5.3 Funktionsweise	71
4.5.4 Elektrische Daten	71
4.5.5 Pinbelegung.....	72

4.5.5.1 Top Layer.....	72
4.5.5.2 Bottom Layer.....	72
4.6 Inertial Measurement Unit (IMU)	73
4.6.1 Allgemeines	73
4.6.2 Mechanische Daten	73
4.6.3 Schaltplan.....	74
4.6.3.1 Spannungsversorgung	75
4.6.3.2 IMU-Interface	75
4.6.3.3 BMP280	76
4.6.3.4 MPU9250.....	76
4.6.4 Pinbelegung.....	77
4.7 Videotransmitter (VTx)	78
4.7.1 Allgemeines	78
4.7.2 Funktionsweise.....	78
4.7.3 Elektrische Daten	78
4.7.4 Mechanische Daten.....	78
4.7.5 Pinbelegung.....	79
4.8 Livekamera	79
4.8.1 Allgemeines	79
4.8.2 Funktionsweise.....	80
4.8.3 Elektrische Daten	80
4.8.4 Pinbelegung.....	80
4.9 Receiver.....	80
4.9.1 Allgemeines	80
4.9.2 Funktionsweise.....	81
4.9.3 Elektrische Daten	81
4.9.4 Pinbelegung.....	81
5 Steuerungssoftware.....	82
5.1 Hauptprogramm	82
5.1.1 Umgang mit Initialisierungsfehler	86
5.1.1.1 Übersicht Fehlercodes	87
5.2 Bestimmen der Akkuspannung - DS2438.....	89
5.2.1 One-Wire Protokoll	89
5.2.1.1 One-Wire Schreibzyklen.....	90
5.2.1.2 One-Wire Lesezyklen	93
5.2.1.3 Initialisierungssequenz	95
5.2.1.4 ROM-Funktionsbefehl	96
5.2.1.5 Memory-Funktionsbefehl.....	96
5.2.2 Registerübersicht DS2438	97
5.2.3 Initialisierung DS2438.....	98
5.2.4 Spannungsüberwachung	99
5.3 Real Time System Interrupt (MAIN_ISR).....	101
5.4 Einlesen der Daten von Fernsteuerung	103
5.4.1 Konfiguration Fernsteuerung	105
5.4.2 Unterstützte Protokolle	107

5.4.2.1 PPM (Pulse Position Modulation)	107
5.4.2.2 S.Bus	107
5.4.2.3 I.Bus.....	109
5.4.3 Initialisierung Empfangssoftware	111
5.4.4 Empfangssoftware	115
5.5 Inertial Measurement Unit (IMU)	123
5.5.1 I ² C Protokoll.....	123
5.5.1.1 Schreibzyklus IMU	125
5.5.1.2 Lesezyklus IMU.....	126
5.5.2 IMU-Verbindungstest	128
5.5.3 Bestimmen der Lagewinkel - MPU9250	130
5.5.3.1 Registerübersicht MPU9250.....	130
5.5.3.2 Initialisierung Accelerometer und Gyroskop	131
5.5.3.3 Einlesen der Accelerometer- und Gyroskop-Daten.....	132
5.5.3.4 Berechnen der Lagewinkel.....	134
5.5.4 Bestimmen der Höhe - BMP280	136
5.5.4.1 Registerübersicht BMP280.....	136
5.5.4.2 Initialisierung Barometer.....	137
5.5.4.3 Einlesen der Barometer-Daten	139
5.5.4.4 Berechnung der Höhe	141
5.6 Motorregelalgorithmus	142
5.7 Motoransteuerung	144
5.7.1 DShot Protokoll.....	144
5.7.2 Initialisierung Motoransteuerung	145
5.7.3 Motoransteuerung Software.....	146
5.8 PID-Regler	150
5.8.1 Initialisierung PID-Regler	150
5.8.2 PID-Algorithmus	152
5.9 Terminal Übertragung und Status LEDs.....	155
5.10 Terminal Kommunikation.....	157
5.11 Status – LEDs	159
6 Datenübertragung der Mess- und Videodaten	161
6.1 Überblick Datenübertragung	161
6.2 Kommunikation: Flight Controller und Sender	162
6.2.1 Blockschaltbild.....	162
6.2.2 Kamera.....	162
6.2.2.1 Bildkonfiguration.....	162
6.2.3 Sender – VTx.....	164
6.2.3.1 VTx Einstellungen	165
6.2.3.2 VTx – Funkkanal	166
6.2.4 ASK – Modulation	167
6.2.5 Datenübertragung – Programm auf Cortex µC.....	170
6.2.5.1 Verwendete Methode der Datenübertragung.....	170
6.2.5.2 Gleitkommadarstellung (memcpy)	170
6.2.5.3 UART	171

6.2.5.4 Programm	173
6.2.6 Testen der Übertragung.....	178
6.3 Kommunikation: Sender und Empfänger.....	182
6.3.1 Blockschaltbild.....	182
6.3.2 Aufbau Empfängermodul	182
6.3.3 Verbindungsaufbau.....	182
6.3.4 Testen der Übertragung.....	183
6.3.5 Videoübertragung zu FPV – Brille.....	183
6.4 Kommunikation: Empfänger und Raspberry Pi.....	184
6.4.1 Blockschaltbild.....	184
6.4.2 Anschlussdiagramm	185
6.4.3 Raspberry Pi GPIO	186
6.4.4 Datenrückgewinnung	187
6.4.5 CVBS-to-USB-Converter	189
6.4.6 Empfangsprogramm auf Raspberry Pi	189
6.4.6.1 UART auf dem Raspberry Pi	189
6.4.6.2 MQTT-Server	190
6.4.6.3 Programm	191
6.5 Testen der Datenübertragungskette.....	198
7 Visualisierungs-App	200
7.1 Applikation	200
7.1.1 Einführung – Dart / Flutter - Framework.....	200
7.1.1.1 Allgemeines	200
7.1.1.2 Pub Dev / Package Installer	200
7.1.1.3 Dart	200
7.1.1.4 State Management.....	202
7.1.1.5 StreamBuilder	203
7.1.2 Allgemeines zur App	204
7.1.2.1 Usersystem	205
7.1.2.2 Datenvisualisierung.....	205
7.1.2.3 3D-Model-Viewer	206
7.1.2.4 Livestream-Viewer	206
7.1.3 UI-Konzept	207
7.1.3.1 Farbkombinationen	208
7.1.3.2 8px-Rasterprinzip (8-Point Grid)	209
7.1.4 Projektstruktur und -umgebung	209
7.1.4.1 Editor – Visual Studio Code	209
7.1.4.2 Flutter Installation.....	210
7.1.4.3 Projekterstellung	211
7.1.4.4 Projektstruktur	211
7.1.4.5 Versionsmanagement / Git + GitHub	212
7.1.4.6 Packages	212
7.1.4.7 App Icon.....	215
7.1.4.8 Splash + Willkommensscreen	216
7.1.4.9 Native Splash Screen.....	216

7.1.5 Startbereich	218
7.1.5.1 Initialisierung der App.....	218
7.1.5.2 Willkommensbildschirm.....	220
7.1.5.3 Login- u. Registrierung-Screens.....	221
7.1.5.4 Autorisierungsfehlermeldungen.....	228
7.1.5.5 Adaptiertes Eingabefeld (StdInputField)	230
7.1.6 Homepage	232
7.1.6.1 Aufzeichnung starten und stoppen.....	232
7.1.6.2 Serverdatendialog	232
7.1.6.3 Aufzeichnung stoppen.....	233
7.1.6.4 Bottom Navigation Bar / GNav-Bar.....	234
7.1.6.5 Flugdatenvisualisierung.....	236
7.1.6.6 3D-Model-Viewer	246
7.1.6.7 Live-View	250
7.1.7 Sidemenü / Drawer	254
7.1.7.1 Userprofil.....	255
7.1.7.2 Credits / Mitwirkende.....	257
7.1.7.3 Aufgezeichneten Flüge.....	258
7.1.7.4 Einstellungen	262
7.1.7.5 Help / FAQ	264
7.1.7.6 Logout.....	265
7.2 Firebase-Backend.....	266
7.2.1 Installation via Firebase CLI.....	267
7.2.1.1 Firebase CLI – Setup	267
7.2.1.2 Firebase-Login am Computer.....	268
7.2.2 Erstellung eines Firebase-Projekts	269
7.2.2.1 Services aufsetzen.....	269
7.2.3 Einbindung in Flutter.....	270
7.2.3.1 Flutterfire Configure	270
7.2.3.2 Firebase in Flutter-Code initialisieren	271
7.2.3.3 Firebase Dienste hinzufügen.....	272
7.2.4 Nutzung der verschiedenen Datenbanksysteme	272
7.2.4.1 Auth	273
7.2.4.2 Firestore Database.....	273
7.2.4.3 Realtime Database.....	276
7.2.4.4 Storage	277
8 Videostreaming	278
8.1 Allgemeiner Aufbau.....	278
8.1.1 Anforderungen.....	278
8.2 CADDXFPV Analog Kamera	278
8.2.1 Produktinformationen.....	279
8.2.2 Verwendungsgrund.....	279
8.2.3 Verbindungstest via USB-Camera App	279
8.2.3.1 Verbindungsinitialisierung	279
8.2.3.2 Verbindungstest	279

8.3 USB2.0 VHS Video Grabber	280
8.3.1 Produktinformationen.....	280
8.3.1.1 lsusb	281
8.3.1.2 Formatinformation von Treiber	281
8.3.1.3 Vorteile eines gleichen Chipsatzes.....	282
8.3.2 Problem der Digitalisierung.....	283
8.3.3 Treiberkompatibilität	283
8.3.4 Testen des USB-Grabbers.....	283
8.3.4.1 Honestech VHS to DVD 2.0 SE.....	283
8.3.4.2 Latenz des Videograbbers	284
8.3.5 Berechnungen	284
8.3.5.1 Theoretische Datenrate.....	285
8.4 RTMP-Server via NGINX aufsetzen	286
8.4.1 RTMP Allgemein.....	286
8.4.2 NGINX	286
8.4.2.1 Allgemein	286
8.4.2.2 Installation.....	287
8.4.2.3 RTMP-Konfiguration.....	287
8.4.2.4 Server starten / neustarten.....	288
8.4.2.5 Serverstatus einsehen.....	288
8.5 Videostream erzeugen	289
8.5.1 FFMPEG Allgemein	289
8.5.2 Installation	289
8.5.3 Transkodierung.....	290
8.5.3.1 Allgemein	290
8.5.3.2 Ablauf der Konvertierung / Transkodierung	290
8.5.3.3 Simple Konvertierung.....	291
8.5.3.4 Optimierungen	291
8.5.3.5 Videoserver auf Desktop-Laptop	298
8.5.4 Problembehebung bei Konvertierung.....	300
8.5.4.1 Problem („Broken Pipe“).....	300
8.5.4.2 Problemlösung	301
8.5.5 video_stream.sh	301
8.5.5.1 Skript-Autostart	302
8.6 Videoserver-Portweiterleitung	304
8.6.1 Allgemein.....	304
8.6.2 Router.....	304
8.6.2.1 Allgemein	304
8.6.2.2 Router Website	304
8.6.2.3 Portweiterleitung	305
8.6.3 Kontakt mit Routerfirma zur Freischaltung	306
8.6.4 Testen der Portweiterleitung	307
8.7 Videostream testen	308
8.7.1 VLC-Player	308
8.7.1.1 Netzwerkstreams öffnen.....	308

8.7.2 ffplay.....	309
8.7.3 Testverfahren	309
8.7.4 Groundstation als Videoserver.....	310
8.7.4.1 Test auf Videoserver (localhost).....	310
8.7.4.2 Test in gleichem Netz.....	310
8.7.4.3 Test von externem Netz	311
8.7.4.4 Flutter-App	312
8.7.4.5 Testergebnisse	312
8.7.5 Laptop als Videoserver.....	313
8.7.5.1 Testen im gleichen Netz.....	313
8.7.5.2 Testen in Visualisierungsapp.....	313
8.7.5.3 Testergebnisse	313
8.7.6 Videoübertragung von der Drohne.....	314
9 Ergebnisse	315
9.1 Gesamtergebnisse Elektronik	315
9.2 Gesamtergebnisse Mechanik.....	315
9.3 Gesamtergebnisse Embedded.....	315
9.4 Gesamtergebnisse Visualisierungsapp	316
10 Anhang.....	317
10.1 Verwendete Software	317
10.1.1 Altium Designer 22	317
10.1.2 Fusion 360.....	317
10.1.3 Visual Studio Code	317
10.1.4 STM32CubeMX	317
10.1.5 Keil µVision5.....	317
10.1.6 Blender	317
10.1.7 Affinity Photo	317
10.2 Einführung Entwicklungsumgebung - Steuerungssoftware.....	318
10.3 Kurzeinführung CAD – Software (Fusion 360).....	325
10.3.1 UI und Projekterstellung.....	325
10.3.2 Skizze anfertigen	325
10.3.3 Körper erstellen	326
10.3.4 Schrift und Bilder einfügen.....	327
10.4 3D – Druck.....	328
10.4.1 3D – Drucker Software (UltiMaker Cura).....	328
10.5 Inbetriebnahme Anleitung	329
10.5.1 Aufladen der Akkus.....	329
10.5.1.1 Ladegerät.....	329
10.5.2 Inbetriebnahme der FPV-Drohne	334
10.6 Projektpläne	338
10.6.1 Bieder & Lendl	338
10.6.2 Heinicke & Hinterberger.....	339
10.7 Zeitaufwand	340
10.7.1 Bieder	340
10.7.2 Lendl.....	340

10.7.3 Heinicke.....	341
10.7.4 Hinterberger.....	341
10.8 Projektkosten	342
11 Quellen.....	343
11.1 Gedruckte Medien.....	343
11.2 Online	343
12 Verzeichnis der Abbildungen.....	350

1 Einleitung

1.1 Projektziel

Das Ziel der Diplomarbeit ist die Realisierung einer leistungsstarken FPV-Drohne (First Person View), die mittels einer Fernsteuerung gesteuert wird, während man durch eine, an der Drohne installierten Kamera das aktuelle Bild anschauen kann. Dieses Livebild soll auf einer FPV-Brille und in unserer selbstprogrammierten App dargestellt werden. Die gesamte Steuerelektronik und Software wird selbst entworfen und entwickelt.

1.2 Gesamtüberblick

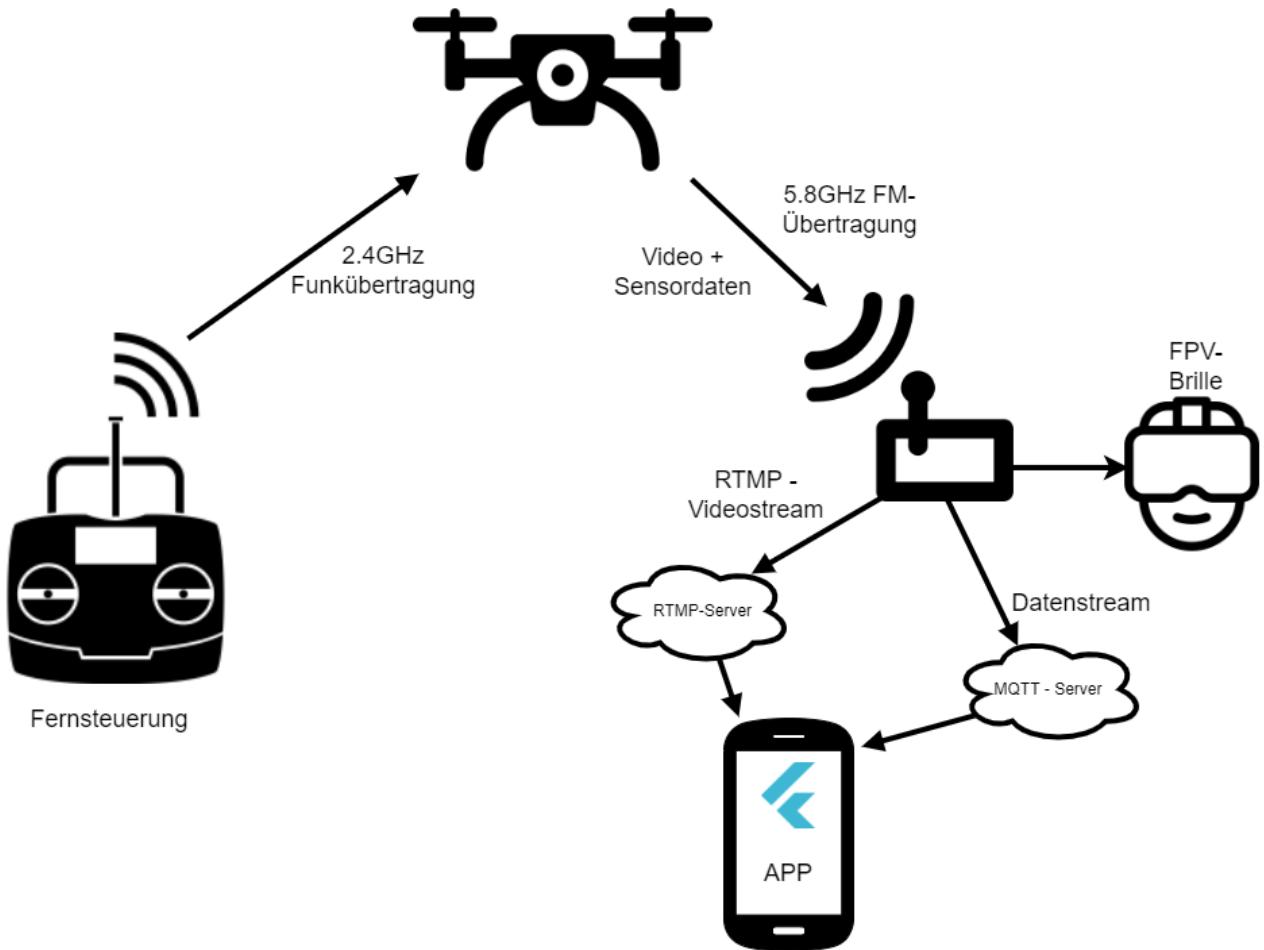


Abbildung 1: Gesamtüberblick

2 Grundlagen Drohnenflug

2.1 Inertial Measurement Unit (IMU)

Die Inertial Measurement Unit ist eine Kombination aus Sensoren, mit denen die Beschleunigung und die Rotation der Drohne gemessen werden kann und Positionsdaten berechnet werden können. Diese Positionsdaten stellen die Grundlage des Flugreglers dar und werden für die Visualisierung übertragen. Der IMU, der für unsere Drohne verwendet wird, ist eine Kombination aus dem MPU9250 und dem Barometer BMP280.

[GRL1]

2.1.1 Motion Processing Unit (MPU)

Der MPU9250 beinhaltet ein Gyroskop, ein Accelerometer und auch ein Magnetometer. Das Magnetometer ist im Grunde ein Messgerät mit dem magnetischen Flussdichten, also zum Beispiel das Erdmagnetfeld gemessen werden kann. Daraus kann dann die Ausrichtung des Sensors bestimmt werden. Da das Magnetometer in unserem Fall für die Lagebestimmung nicht verwendet wird, wird hier nicht weiters darauf eingegangen.

[GRL2]

2.1.1.1 Gyroskop

Ein Gyroskop ist in seiner einfachsten Form ein sich um seine eigene Achse drehendes Rad, das so gelagert ist, dass sich die Rotationsachse in alle Richtungen bewegen und ausrichten kann. Damit das Rad konstant rotiert, wird es von einem Motor angetrieben. Wenn es sich schnell genug dreht, bleibt das rotierende Rad aufgrund der Drehimpulserhaltung in seiner ursprünglichen Lage, egal in welcher Lage sich der Sensor, also die Drohne gerade befindet. In unserem MPU befindet sich ein MEMS-Gyroskop (Micro-Elektromechanisches System), das auch auf einem ähnlichen Prinzip funktioniert, aber kein rotierendes Rad und auch keinen Motor beinhaltet. Stattdessen befinden sich darin mikroskopische Siliziumstrukturen, die schwingfähig sind und es ermöglichen mechanische Veränderungen, wie Lage und Rotation um die eigene Achse in elektrische Signale umzuwandeln und dann vom Sensor in entsprechende Daten verarbeitet werden können. Eine wichtige Eigenschaft, die bei Gyroskopen zu berücksichtigen ist, ist der sogenannte Drift. Umso länger sie in Betrieb sind, ohne dass eine Kalibrierung vorgenommen wird, desto größer weicht deren ausgegebene Lage von der eigentlichen Lage ab.

[GRL3]

2.1.1.2 Accelerometer

Das eingebaute Accelerometer (ein Beschleunigungssensor) ist ebenfalls ein mikromechanisches System (MEMS-Accelerometer) und misst, durch bewegbare mikroskopische Strukturen, die Beschleunigung, die der Sensor erfährt. Bei einer Beschleunigung bewegen sich diese Strukturen aufgrund des Trägheitsprinzips entsprechend der dabei wirkenden Kraft entgegengesetzt der Richtung der Beschleunigung. Durch diese Bewegung werden elektrische Signale erzeugt, die der Sensor in brauchbare Daten umwandelt.

[GRL4]

2.1.2 Barometer - BMP280

Das verwendete Barometer (BMP280) benutzt ein piezoresistives Prinzip. Dabei reagiert eine Silizium-Membran auf den äußeren Druck und bildet Verzerrungen, die grob gesagt den elektrischen Widerstand verändern. Durch eine, vom variablen Widerstand abhängige Spannung kann somit also der Luftdruck und dadurch auch die aktuelle Höhe berechnet werden. Diese Höhenberechnung ist jedoch sehr ungenau.

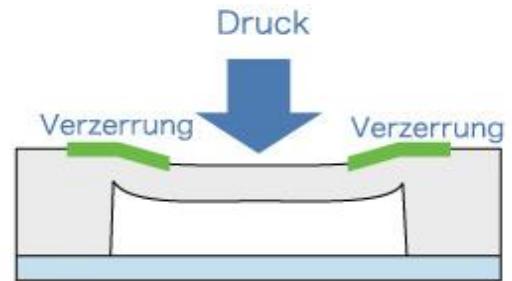


Abbildung 2: Barometer Innenaufbau [GRL5]

[GRL5]

2.2 Lagewinkel

Mit den aus der IMU gewonnenen Daten können verschiedene Lagewinkel berechnet werden, die letztendlich Auskunft über die Lage der Drohne geben. Dadurch kann eine Lagekorrektur auf Basis eines PID-Reglers (siehe: [Kapitel 2.3](#)) implementiert werden, damit die Drohne von allein „schweben“ kann und ohne Eingreifen des Piloten nicht direkt abstürzt.

2.2.1 Throttle

Die Auf- und Abbewegung der Drohne wird auch mit Throttle (dt. Gas) bezeichnet. Um die Drohne in die Höhe steigen zu lassen, muss man die Motoren gleichmäßig schneller laufen lassen. Um wieder runterzufliegen, dreht man die Motorgeschwindigkeit wieder zurück.

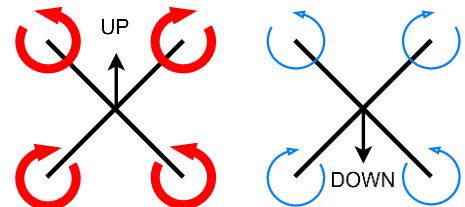


Abbildung 3: Throttle Visualisierung

[GRL6]

2.2.2 Pitch

Eine Bewegung um die Pitch-Achse, also eine Neigung nach vorne oder hinten resultiert in einer Vorwärtsbewegung bzw. einer Rückwärtsbewegung. Dies erzielt man durch schnelleres Drehen der zwei hinteren Motoren für eine Vorwärtsbewegung oder durch schnellere Geschwindigkeit der vorderen zwei Motoren für eine Rückwärtsbewegung der Drohne.

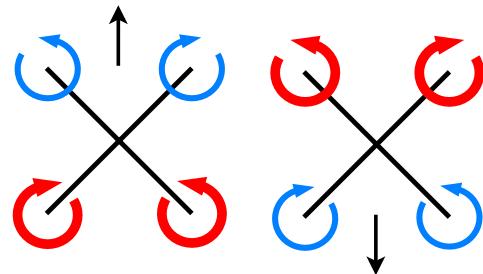


Abbildung 4: Pitch Visualisierung

[GRL6]

2.2.3 Roll

Eine Bewegung um die Roll-Achse, also eine Neigung nach links oder rechts resultiert in einer Linksbewegung bzw. einer Rechtsbewegung. Das erzielt man durch eine schnellere Geschwindigkeit der zwei linken Motoren für eine Bewegung nach rechts oder durch schnelleres Drehen der rechten zwei Motoren für eine Bewegung nach links.

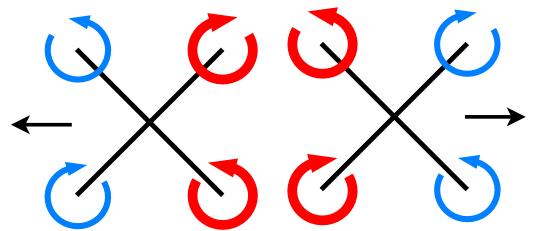


Abbildung 5: Roll Visualisierung

[GRL6]

2.2.4 Yaw

Bei der Drehung um die Yaw-Achse wird das Ganze ein bisschen spannender. Angenommen wir hätten auf der Drohne nur einen Rotor in der Mitte, dann würde sich die gesamte Drohne aufgrund der Drehimpulserhaltung auf der Yaw-Achse in die entgegengesetzte Richtung des Rotors drehen, was einen kontrollierten Flug unmöglich machen würde. Genau deswegen benötigen Helikopter auch einen zweiten kleineren Heckrotor, der dieser Drehbewegung entgegensteuert. Auf der Drohne würde genau dasselbe Problem entstehen, wenn sich alle Rotoren in dieselbe Richtung drehen würden. Genau deswegen müssen Quadrocopter so angesteuert werden, dass sich immer 2 Rotoren, die gegenüberliegenden, in die eine Richtung drehen und die anderen zwei in die andere Richtung.

Genau nach diesem Prinzip funktioniert auch das absichtliche Drehen um die Yaw-Achse. Dabei lässt man, um die Drohne nach links zu rotieren, die beiden gegenüberliegenden Rotoren schneller drehen, die sich im Uhrzeigersinn drehen und um sie nach rechts rotieren zu lassen die beiden Rotoren, die sich gegen den Uhrzeigersinn drehen schneller laufen. Dabei müssen aber immer die 2 anderen Motoren langsamer laufen, da die Drohne sonst steigt.

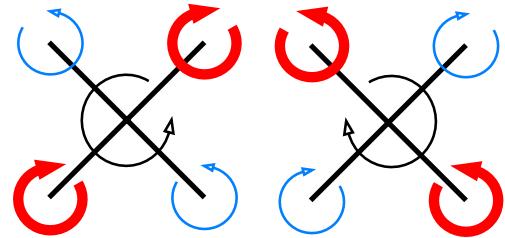


Abbildung 6: Yaw Visualisierung

[GRL6]

2.2.5 Komplementärfilter

Der Komplementärfilter hat die Aufgabe, die Messwerte von einem Beschleunigungssensor und Gyroskop in die drei Lagewinkel umzuwandeln.

Accelerometer Winkel:

Durch die Kombination der terrestrischen Gravitation und Trigonometrie ermöglicht es die Accelerometer-Werte in Neigungswinkel umzuwandeln:

$$\text{accelerometerPitch} = \text{atan2}(\text{accelerometerY}, \text{accelerometerZ})$$
$$\text{accelerometerRoll} = \text{atan2}(\text{accelerometerX}, \text{accelerometerZ})$$

accelerometerPitch/Roll ... jeweiliger berechneter Winkel in Radian (rad)
accelerometerX/Y/Z ... Accelerometer Messwert der jeweiligen Achse in g
(Erdbeschleunigung, ungefähr 9,81m/s²)

Der Yaw-Winkel kann nicht berechnet werden, da die Yaw-Bewegung in der Ebene, um die Z-Achse, stattfindet und daher unabhängig von den Werten der X- und Y-Achse ist.

Im Gegensatz zum Gyroskop besitzt ein Accelerometer keinen Drift in der Messung und kann daher für längerfristige Messungen verwendet werden.

Für den Filter wird der berechnete Winkel in Grad statt in Radian benötigt. Daher müssen die Ergebnisse mit einem Faktor von $\frac{180^\circ}{\pi} \approx 57,296$ multipliziert werden.

Problem:

Wenn der Accelerometer bewegt wird, kann dieser nicht zwischen den Bewegungsbeschleunigungen und der Erdbeschleunigung unterscheiden und liefert daher verfälschte Werte.

Gyroskop Winkel:

Durch die Integration der Winkelbeschleunigungswerte ist es möglich, die drei Lagewinkel zu bestimmen:

$$\text{gyroscopePitch} = \int \text{gyroscopeX} dt$$
$$\text{gyroscopeRoll} = \int \text{gyroscopeY} dt$$
$$\text{gyroscopeYaw} = \int \text{gyroscopeZ} dt$$

gyroscopePitch/Roll/Yaw ... jeweiliger berechneter Winkel in Radianen (rad)
gyroscopeX/Y/Z ... Gyroskop Messwert der jeweiligen Achse in Grad pro Sekunde (°/s)
dt ... Zeitbereich in Sekunden (s)

Problem:

Durch die Integration werden Messungenauigkeit aufsummiert, die zu einem Wertedrift führen. Daher kann das Gyroskop nur für kurzfristige Messungen verwendet werden.

Sensorfusion:

Der KomplementärfILTER kann als Kombination von zwei Filter gesehen werden: Ein Hochpassfilter für das Gyroskop und ein Tiefpassfilter für den Accelerometer. Das Accelerometer liefert eine gute Indikation der Orientierung bei konstanten Bedingungen, und das Gyroskop liefert eine gute Indikation bei schnellen Neigungsänderungen.

$$\text{winkel} = \alpha * (\text{winkel} + \text{gyroscopeData} * dt) + (1 - \alpha) * \text{accelerometerWinkel}$$

Winkel ... Pitch/Roll-Winkel in Grad (°)

α ... Filterkoeffizient

gyroscopeData ... Gyroskop Messwerte der jeweiligen Achse in Grad pro Sekunde (°/s)

dt ... Abtastzeit in Sekunden (s)

accelerometerWinkel ... berechneter Roll/Pitch nur mit Accelerometerdaten in Grad (°)

Der Wert für α ist typischerweise 0,98. Das bedeutet, dass die Gyroskop-Messung zu 98% und die Accelerometer-Messung zu 2% gewichtet wird. Daraus folgt:

$$\text{pitch} = 0,98 * (\text{pitch} + \text{gyroscopeX} * dt) + 0,02 * \text{accelerometerPitch}$$

$$\text{roll} = 0,98 * (\text{roll} + \text{gyroscopeY} * dt) + 0,02 * \text{accelerometerRoll}$$

$$\text{yaw} = \text{yaw} + \text{gyroscopeZ} * dt$$

Dadurch, dass mit dem Accelerometer kein Yaw-Winkel bestimmt werden kann, wird der Winkel nur mit dem Gyroskop berechnet. Wegen dem Gyroskop-Drift wird dieser Winkel über die Zeit immer ungenauer und muss für einen genauen Wert nach einer gewissen Zeit zurückgesetzt werden.

2.3 PID-Regler

Im Allgemeinen wird zwischen einer Steuerung und einer Regelung unterschieden. Bei einer Steuerung wird der Sollwert direkt in einen Steuerungsalgorithmus geleitet, der immer nach demselben Schema abläuft. Bei einer Regelung wird der Istwert des Systems rückgekoppelt und mit dem Sollwert verglichen. Dieser Vergleich ergibt eine Regelabweichung, die in den Regelalgorithmus geleitet wird. Das Ziel einer Regelung ist, die Regelabweichung zum Verschwinden zu bringen – Soll- und Istwert auf denselben Wert regeln.

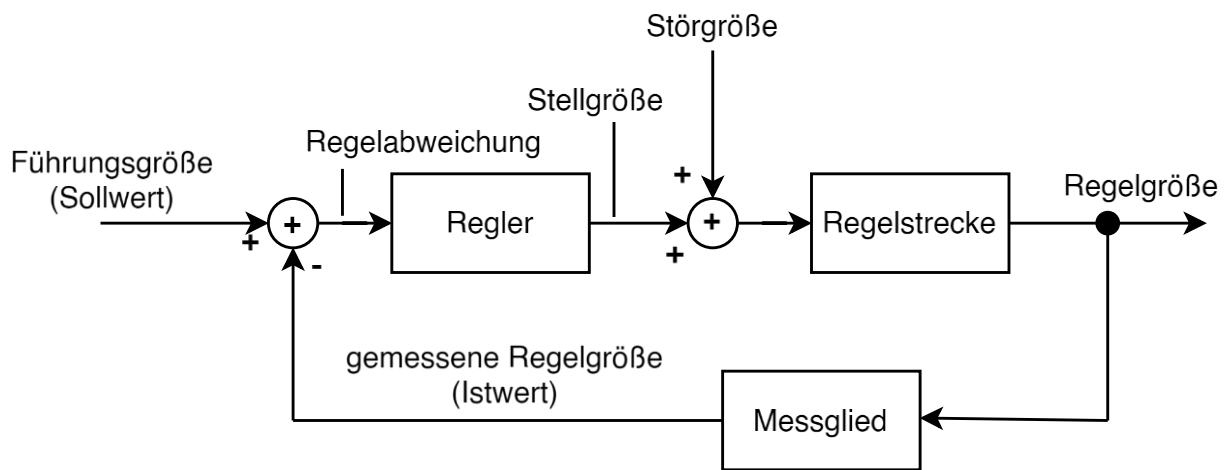


Abbildung 7: Regler-Blockschaltbild

Führungsgröße ... eingestellter Sollwert, der vom Regler erreicht werden soll

Regelabweichung ... Differenz zwischen Führungsgröße und gemessener Regelgröße

Regler ... erstellt Stellgröße proportional von der Regelabweichung

Stellgröße ... Reglerausgang

Störgröße ... externe Einflüsse, die die Stellgröße beeinflussen

Regelstrecke ... wandelt Stellgröße in die Regelgröße um

Regelgröße ... gemessener Istwert, meistens durch Sensoren bestimmt

Es gibt verschiedene Arten für Regler, die für unterschiedliche Anwendungen Vor- und Nachteile liefern. Für den Quadrokopter wurde ein PID-Regler verwendet. Dieser hält während des Fluges die Drohne stabil. Bei Störgrößen, wie zum Beispiel Windstößen, soll die Drohne sich automatisch wieder in die richtige Lage ausrichten, um weiter einen stabilen Flug zu gewährleisten.

Ein PID-Regler besteht aus drei verschiedenen Gliedern (Proportional-, Integral-, Differenzialglied), die unterschiedliche Aufgaben erfüllen und Funktionsweisen vorweisen.

In den folgenden Kapiteln werden diese Bezeichnungen verwendet:

$e(t)$ / $e[n]$... Regelabweichung für analoge / digitale Komponente

$a(t)$ / $a[n]$... Reglerausgang für analoge / digitale Komponente

$s(t)$... Führungsgröße (Sollwert)

$i(t)$... Regelgröße (Istwert)

2.3.1 Proportionalglied (P-Glied)

Das Proportionalglied setzt das Ausgangssignal proportional zum Eingangssignal.

Analoges P-Glied: $a(t) = k_p * e(t)$

Digitales P-Glied: $a[n] = k_p * e[n]$

Das analoge und digitale P-Glied funktioniert gleich, außer, dass beim digitalen Glied die Abtastzeit t_s bewirkt, dass nicht direkt auf Änderungen reagiert werden kann.

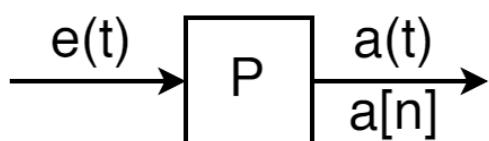


Abbildung 8: P-Glied Schaltsymbol

Vor- (+) und Nachteile (-):

- + kann Regelabweichung zum Verschwinden bringen
- + schnelles Anregeln
- + keine Phasenverschiebung
- kann Regelabweichung = 0 nicht halten

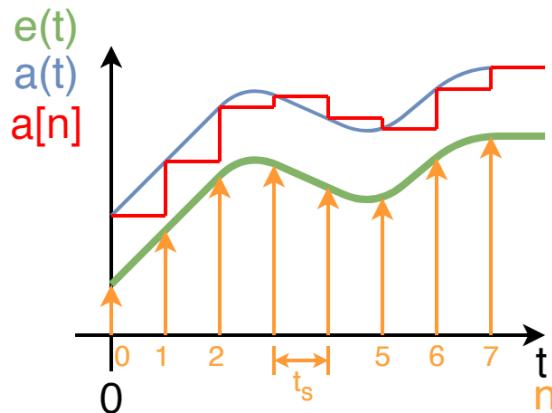


Abbildung 9: P-Glied Ein/Ausgang

2.3.2 Integralglied (I-Glied)

Das Integralglied setzt das Ausgangssignal zum Integral vom Eingangssignal.

Analoges I-Glied: $a(t) = k_I * \int_0^t e(\tau) d\tau$

Digitales I-Glied: $a[n] = a[n - 1] + k_I * e[n] * t_s$

Beim analogen I-Glied wird für jede Regelabweichung das Integral gebildet, während beim digitalen I-Glied die Abtastzeit t_s bewirkt, dass das Integral nur angenähert werden kann (Quantisierungsfehler). Außerdem ist es notwendig, das vorherige Ergebnis abzuspeichern.

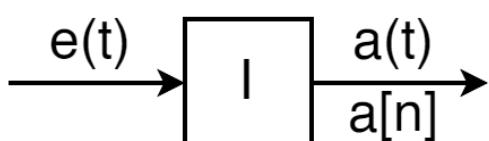


Abbildung 11: I-Glied Schaltsymbol

Vor- (+) und Nachteile (-):

- + kann Regelabweichung zum Verschwinden bringen
- langsames Anregeln
- verringert die Stabilität ($\varphi = -90^\circ$)
- neigt zum Überschwingen

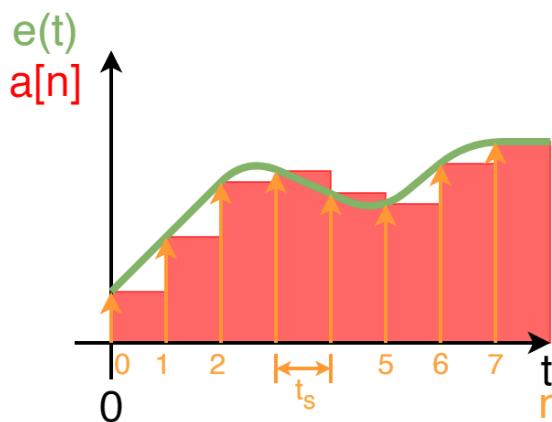


Abbildung 10: I-Glied Ein/Ausgang

2.3.3 Differenzialglied (D-Glied)

Das Differenzialglied setzt das Ausgangssignal zur momentanen Änderung des Eingangssignals.

$$\text{Analogen D-Glied: } a(t) = k_D * \frac{de(t)}{dt}$$

$$\text{Digitales D-Glied: } a[n] = k_D * \frac{e[n] - e[n-1]}{t_s}$$

Beim analogen Glied kann auf die Änderung der Regelabweichung direkt reagiert werden, während beim digitalen Glied durch die Abtastzeit t_s die Änderung nur angenähert werden kann (Quantisierungsfehler). Außerdem ist es notwendig, das vorherige Ergebnis abzuspeichern.

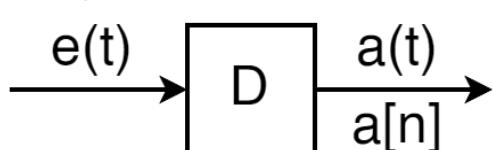


Abbildung 12: P-Glied Schaltsymbol

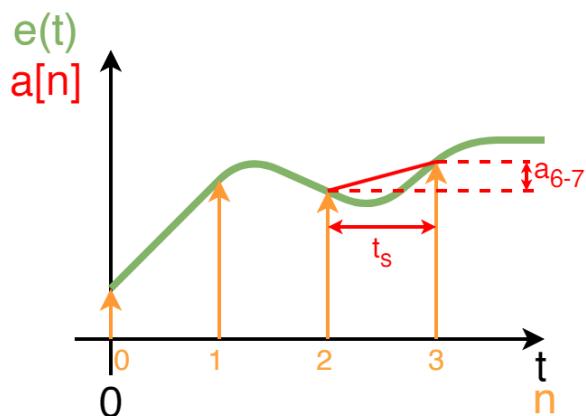


Abbildung 13: P-Glied Ausgangsbestimmung

Vor- (+) und Nachteile (-):

- kann allein nicht Regeln

In Kombination mit anderen Gliedern:

- + schnelles Anregeln

- + verbessert die Stabilität ($\varphi = +90^\circ$)

2.3.4 PID-Regler

Der PID-Regler kombiniert alle Ausgangssignale der einzelnen Glieder.

$$\text{Analoger Regler: } a(t) = k_p * e(t) + k_I * \int_0^t e(\tau) d\tau + k_D * \frac{de(t)}{dt}$$

$$\text{Digitaler Regler: } a[n] = k_p * e[n] + a[n-1] + k_I * e[n] * t_s + k_D * \frac{e[n] - e[n-1]}{t_s}$$

Durch das Einstellen der Regler-Koeffizienten (k_p , k_I , k_D) kann die Stärke des Reglers festgelegt werden. Große Werte können zu einem überschwingenden Verhalten, während kleine Werte zu einem zu langsamem Regeln führen.

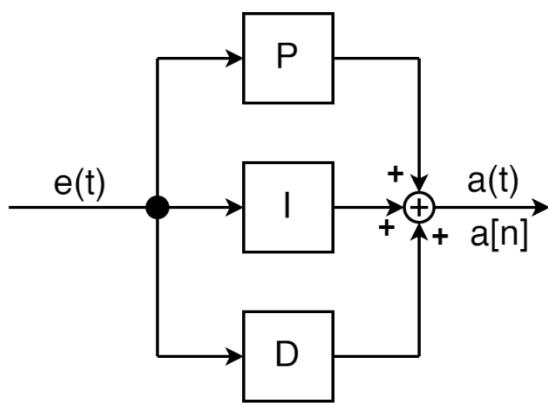


Abbildung 15: PID-Regler Schaltsymbol

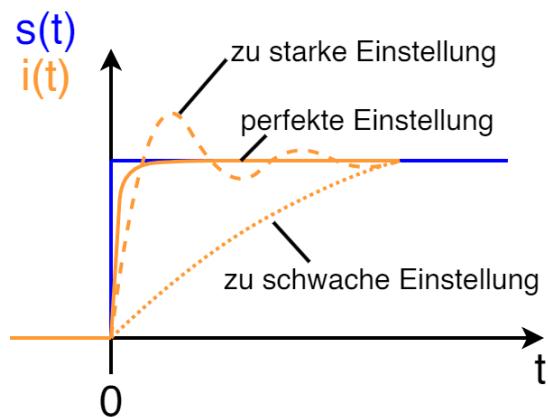


Abbildung 14: PID-Regler Regelung zum Sollwert

3 Mechanischer Aufbau

3.1 FPV – Drohne Gesamtaufbau



Abbildung 16: FPV-Drohne Gesamtaufbau Seitenansicht

Die Drohne besteht aus dem Rahmen (siehe: [Kapitel 3.2](#)), der die Grundlage des Aufbaus bildet. Auf diesem wird die gesamte Hardware, wie der Microcontroller, die Motorsteuerung, die Kamera, die Motoren, der Akku, sowie Sender und Empfänger montiert. Zusätzlich werden Sensoren, wie die IMU (Kombination von Lagesensoren), am Rahmen montiert. Zum Schutz der Rotoren wurden Rotorenschützer (siehe: [Kapitel 3.3.1](#)) designt und montiert.



Abbildung 17: FPV-Drohne Gesamtaufbau Topansicht

3.2 Drohnen-Rahmen Innenaufbau

Im Kern, dem Hauptkörper der Drohne, befinden sich alle wichtigen Steuerungs- und Regelungseinrichtungen, sowie Funkempfänger, Sender und die Kamera. In der Mitte liegt der Flight Controller, auf dem die Steuerungs- und Regelungssoftware läuft. Darunter sitzt der Electronic Speed Controller (ESC), der die Regelgröße vom Flight Controller empfängt und damit die Motoren ansteuert. Ganz vorne befindet sich die IMU, eine Platine bestehend aus mehreren Orientierungssensoren und die Kamera, die das Live-Video aufnimmt und dann an den Videotransmitter weiterleitet. Dieser sitzt ganz hinten und sendet das Video über ein 5,8GHz FM-Signal. Gleichzeitig werden die Sensordaten vom Flight Controller auch an den Videotransmitter geschickt, von dem sie über einen zweiten Übertragungskanal ebenfalls geschickt werden. Über dem Videotransmitter sitzt der Funkempfänger, der die Fernsteuerungsdaten empfängt. Oben auf dem Hauptkörper wird der Akku montiert, der dann direkt mit der ESC verbunden wird und das Gesamtsystem mit Spannung versorgt.

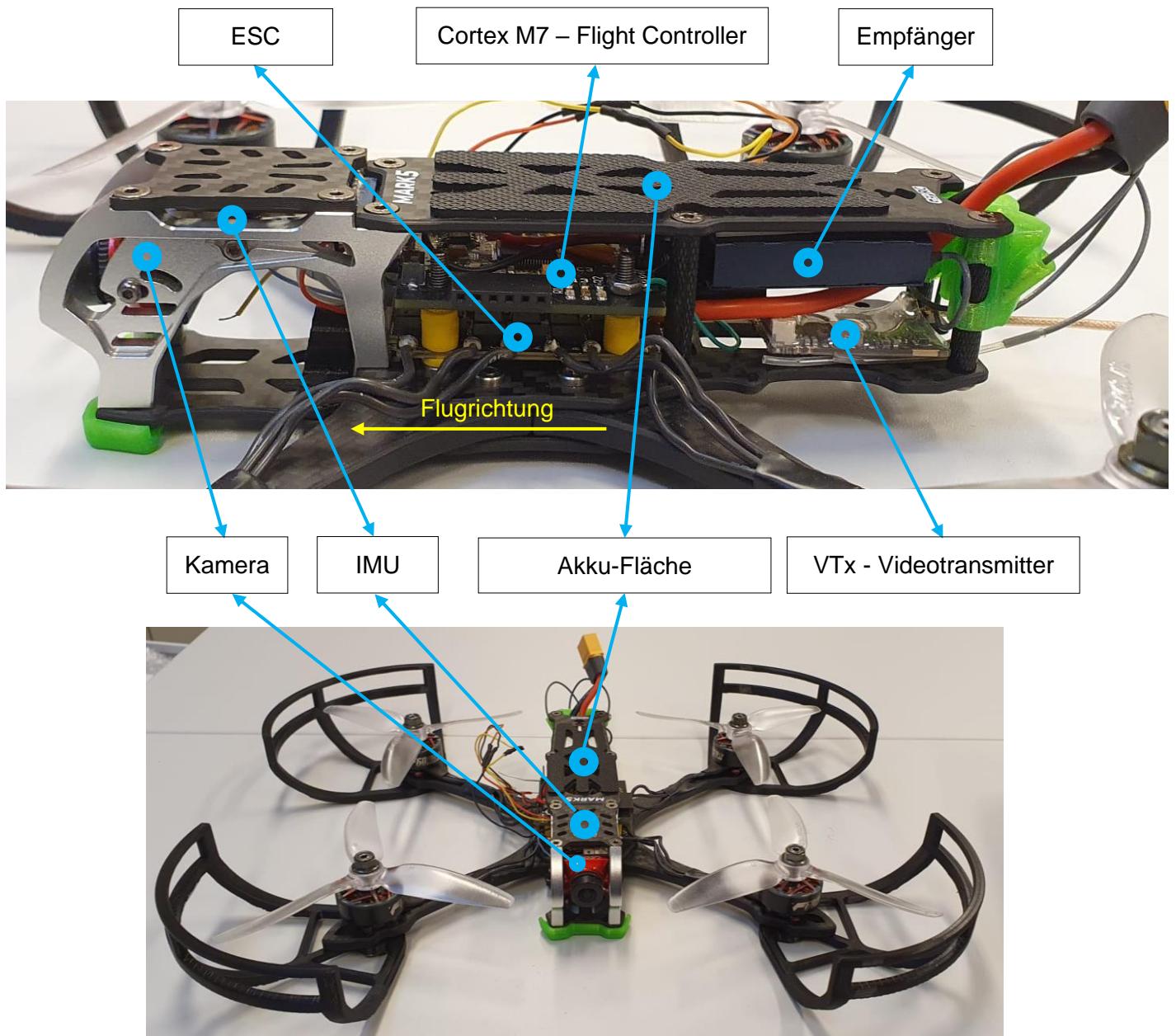


Abbildung 18: FPV-Drohne Innenaufbau

3.3 3D – Modelle

Alle 3D-Modelle wurden mithilfe der CAD-Software Fusion 360 (siehe: [Kapitel 10.3](#)) entworfen und designt und in den, von der Schule bereitgestellten, 3D-Druckern Ultimaker S5 und Ultimaker 2 Extended+ mit der dazugehörigen Software Ultimaker Cura 3D-gedruckt (siehe: [Kapitel 10.4](#)).

Im Laufe der Diplomarbeit wurden noch weitere 3D-Modelle, wie ein Platinen-Gehäuse und Abstandhalter zum Landen designt und gedruckt, die aber bei der finalen Version der Drohne nicht mehr benötigt werden und hier somit auch nicht weiter beschrieben werden.

3.3.1 Rotoren-Schutz

Der Rotoren-Schutz hat die Aufgabe erstens, die Rotoren vor Beschädigungen zu schützen, falls man zum Beispiel gegen Objekte fliegt oder die Drohne beim Testen abstürzt. Zweitens soll eine gewisse Sicherheit gewährleistet sein, sodass man in die Rotoren, während des Betriebs, nicht zu leicht reingreifen kann. Um jedoch ein geringes Gewicht, eine kompakte Größe aber auch eine gute Optik zu gewährleisten, wurde das Design schlicht gehalten.



Abbildung 19: Rotorenschutz Design

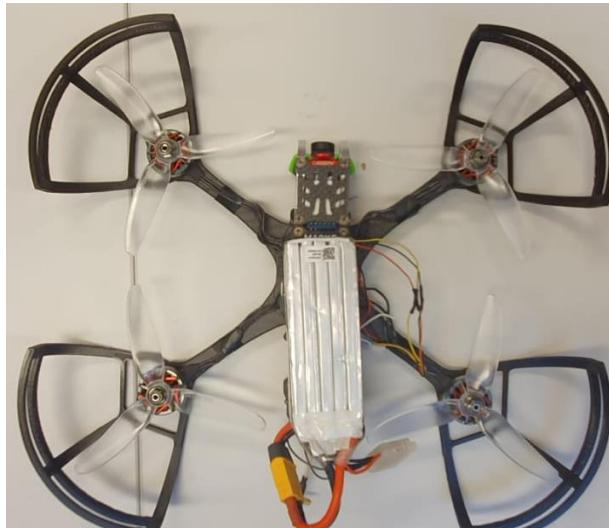


Abbildung 20: FPV-Drohne mit Rotorenschutz

3.3.2 Groundstation

Die Groundstation ist eine Box in der alle benötigten Geräte, die für den Datenempfang und den Videoempfang sowie deren Verarbeitung und Weiterleitung benötigt werden, untergebracht sind. Die Idee dafür ist erst im Laufe der Diplomarbeit entstanden, weil beim Testen zuvor immer alle benötigten Geräte und die Hardware verteilt auf dem Tisch lagen, wodurch man sehr schnell den Überblick verloren hat. Die Box beinhaltet den Raspberry Pi, den Empfänger der Video- und Messdaten, den CVBS-to-USB-Converter und die Komparator-Schaltung zur Aufbereitung der Messdaten. Zusätzlich können wichtige Ereignisse anhand der 3 LEDs angezeigt werden, wie zum Beispiel, ob Daten empfangen werden oder ob es Übertragungsfehler gab (siehe: [Kapitel 6.4.6](#)).

Grundsätzlich ist die Groundstation so konzipiert, dass man einmal alles einbaut und befestigt und danach nicht mehr herausnehmen muss. Das heißt, man kann alle benötigten Geräte noch problemlos an den GPIO-Pins des Raspberry Pi anschließen, aber auch externe Geräte können über die Aussparungen angesteckt werden.

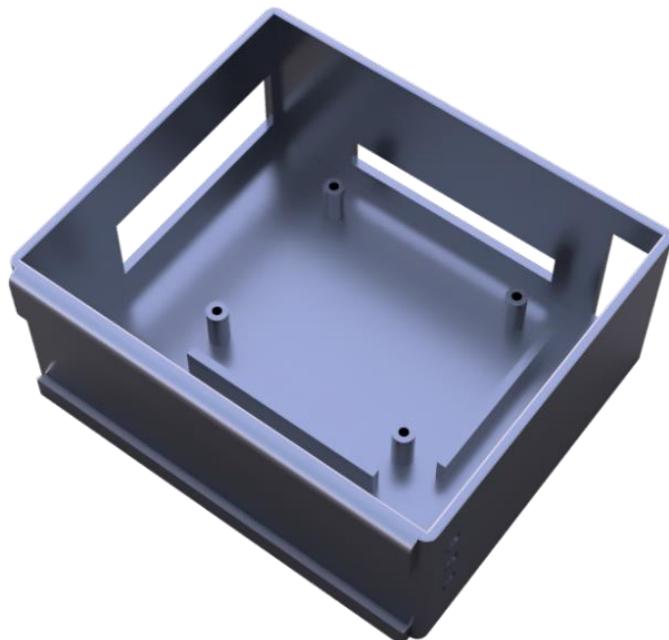


Abbildung 21: Groundstation Design



Abbildung 22: Groundstation Design mit Deckel

Als Dank für die Zusammenarbeit mit **DRONETECH Austria** wurde auch ein Schild designt, das vorne auf der Box zu sehen ist.

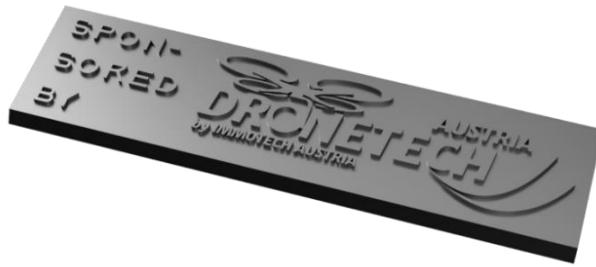


Abbildung 23: Sponsor-Shield

Um Schriftzüge, sowie unser Logo auf der Box darzustellen, war anfangs die Idee, diese einfach etwas weiter herausstehen zu lassen. Nach dem die Modelle jedoch ausgedruckt wurden, waren Schriftzüge trotzdem sehr schwer lesbar. Deswegen kam mir die Idee auf, die Modelle wie einen Stempel auf ein mit Farbe bestrichenes Papier zu drücken. Somit wurde die Sichtbarkeit von Schrift und dem Logo stark verbessert.



Abbildung 24: Deckel Ausdruck



Abbildung 25: Groundstation Ausdruck

4 Elektronik FPV-Drohne

4.1 Allgemeines

Im folgenden Kapitel werden die Hardwarekomponenten im Detail erklärt.

4.1.1 Grundlegendes Hardwarekonzept

Die Elektronik der FPV-Drohne basiert auf einem eigens entwickelten Flight Controller (siehe: [Kapitel 4.3](#)), der die vom Receiver (siehe: [Kapitel 4.9](#)) empfangenen Daten einliest und in Flugmanöver dekodiert. Um diese Flugmanöver auszuführen, werden die gewünschten Werte an den Electronic Speed Controller (ESC) (siehe: [Kapitel 4.5](#)) gesendet, der die Geschwindigkeit der Motoren (siehe: [Kapitel 4.4](#)) weiters regelt. Um sicherzustellen, dass die FPV-Drohne stabil fliegen kann, wurde eine Inertial Measurement Unit (IMU) (siehe: [Kapitel 4.6](#)) verwendet, die Lagewinkel und Höhe messen kann. Weiters wird ein Video-Transmitter (VTx) (siehe: [Kapitel 4.7](#)) verwendet, der das analoge Kamerasignal der Livekamera (siehe: [Kapitel 4.8](#)) einliest und zusammen mit den gemessenen Daten vom Flight Controller an die Groundstation (siehe: [Kapitel 3.3.2](#)) und VR-Brille (siehe: [Kapitel 6.3.5](#)) sendet. Betrieben wird alles durch einen Lithium-Polymer Akku (siehe: [Kapitel 4.2](#)), der ebenfalls auf der FPV-Drohne sitzt.

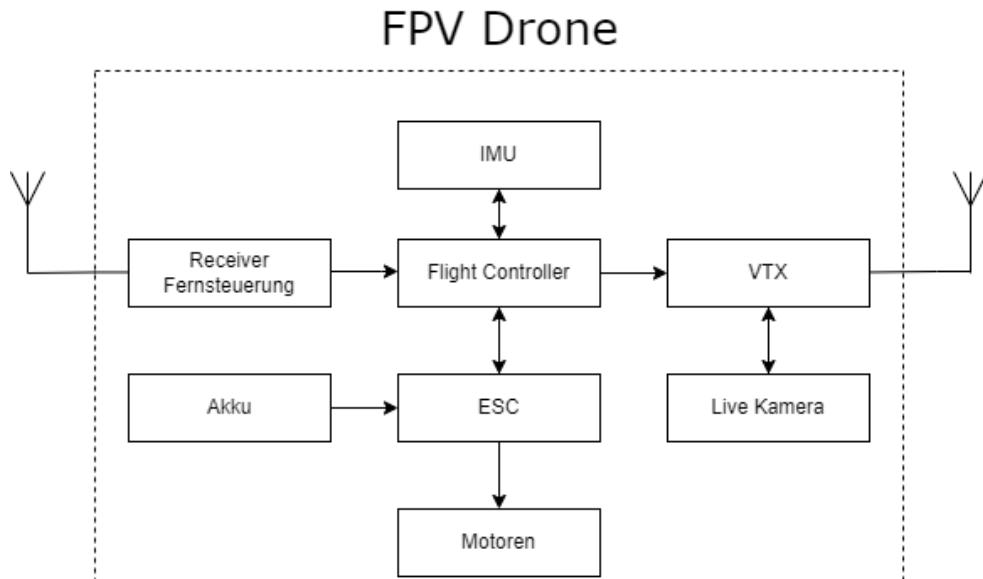


Abbildung 26: Elektronik Gesamtaufbau FPV-Drohne

4.2 Akku

4.2.1 Allgemeines

Der Akku dient für die Spannungsversorgung aller Komponenten der FPV-Drohne während des Fluges. Die Leistungsfähigkeit und Zuverlässigkeit des Akkus bestimmen maßgeblich die Flugdauer und die Fähigkeiten der Drohne, was die Wahl des richtigen Akkus zu einer kritischen Entscheidung macht. Wichtige Parameter sind daher der maximale Strom, den der Akku liefern kann, um die leistungsfähigen Motoren zu betreiben. An zweiter Stelle steht die Kapazität des Akkus, welche ausschlaggebend für die Flugdauer der Drohne ist. Zusätzlich steigt mit der Kapazität aber auch die Größe und das Gewicht des Akkus, dass die

Wendigkeit deutlich einschränken könnte. Somit muss ein Kompromiss zwischen Flugdauer und Gewicht gefunden werden.

4.2.2 Akkuauswahl

Letztendlich haben wir uns für zwei LIPO-Akkus entschieden, da diese eine hohe Energiedichte aufweisen, was bedeutet, dass selbst kleine LIPOS enorme Ströme liefern können, um unsere Motoren bei voller Leistung betreiben zu können. Für den Normalbetrieb wird ein 6S (22,2V – 25,2V) 1300mAh 120C LIPO verwendet, da dieser dank seiner geringeren Kapazität leichter ist und die Drohne ihre Flugmanöver wendiger ausführen kann. Für das Testen wird ein 5S (18,5V – 21V) 2200mAh 95C LIPO verwendet, da man durch die höhere Kapazität länger testen kann, ohne dazwischen aufladen zu müssen. Jedoch ist durch die höhere Kapazität der Akku auch schwerer, weshalb wendige Flugmanöver etwas eingeschränkt sind.

Ein weiterer wichtiger Parameter eines Akkus ist die C-Rate. Diese beschreibt, wie schnell der Akku entladen werden kann. So kann man sich mit der Kapazität und der C-Rate den maximalen Strom ausrechnen, den der Akku wiedergeben kann:

$$\text{Maximaler Strom (A)} = \text{Kapazität (Ah)} * \text{C-Rate}$$

Zusätzlich kann man sich die maximale Leistung, die der Akku wiedergeben kann, wie folgt ausrechnen:

$$\text{Maximale Leistung (W)} = \text{Spannung (V)} * \text{Maximaler Strom (A)}$$

Diese Werte sind von Wichtigkeit, um zu überprüfen, ob die Akkus die nötige Leistung aufweisen, um den ESC und die Motoren bei voller Auslastung betreiben zu können.

4.2.2.1 6S 1300mAh 120C LIPO



Abbildung 27: 6S 1300mAh 120C LIPO

Maximaler Strom:

$$I_{max} = 1,3Ah * 120C = 156A$$

Maximale Leistung:

$$P_{max} = 25,2V * 156A = 3931,2W$$

4.2.2.2 5S 2200mAh 95C LIPO



Abbildung 28: 5S 2200mAh 95C LIPO

Maximaler Strom:

$$I_{max} = 2,2Ah * 95C = 209A$$

Maximale Leistung:

$$P_{max} = 21V * 209A = 4\,389W$$

4.2.3 Aufbau eines LIPO-Akkus

Der Aufbau eines LIPO-Akkus wird anhand eines 6S LIPO-Akkus beschrieben.

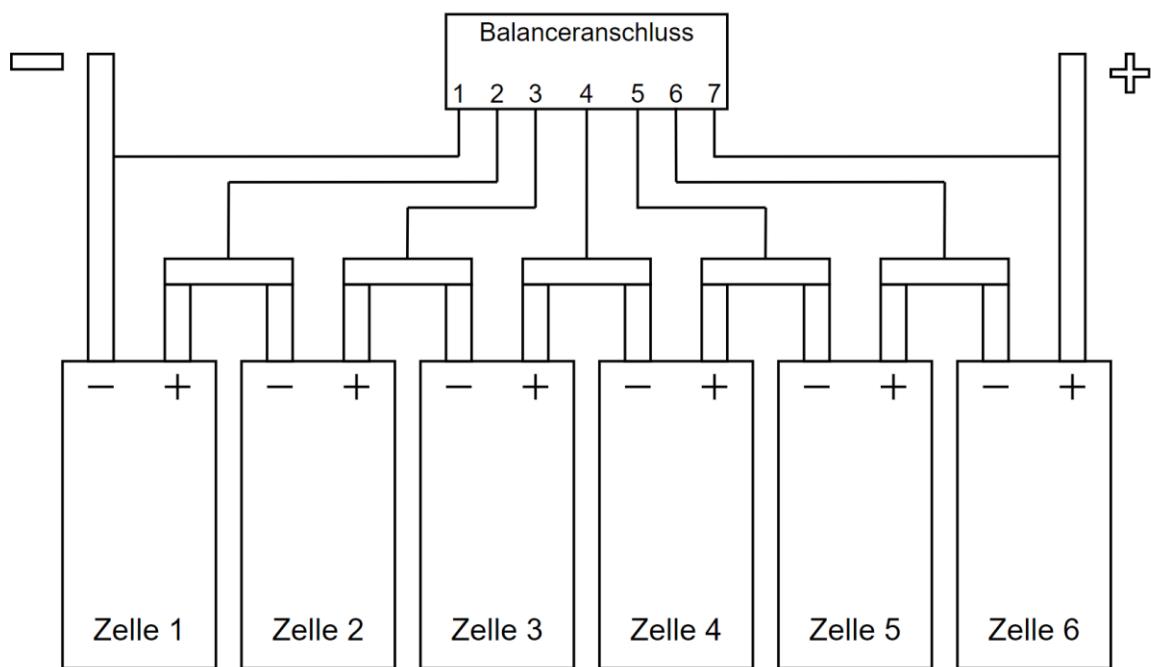


Abbildung 29: LIPO-Akku Aufbau

Die 6S im Namen des LIPO-Akkus bedeuten, dass dieser 6 LIPO-Zellen verbaut hat. Jede dieser LIPO-Zellen hat eine Nennspannung von 3,7V und voll aufgeladen eine Spannung von 4,2V. Um nun höhere Spannungen zu erreichen, werden diese 6 LIPO-Zellen in Serie geschalten, was eine Nennspannung von 22,2V und voll aufgeladen eine Spannung von

25,2V ergibt. Wichtig zu beachten ist, dass eine Zelle nie unter 3,3V fallen darf, da diese sonst tiefentladen ist und unbrauchbar wird. Entladen wird dieser durch den Ladeport, der aus zwei dicken Kabeln besteht. Aufgeladen wird ein LIPO-Akku durch den Ladeport und den Balancer Anschluss. Da sich die einzelnen Zellen nicht gleichmäßig über den Ladeport aufladen, wird der Balancer Anschluss benötigt. Dieser misst die einzelne Spannung jeder Zelle und führt ein wenig Strom hinzu oder entnimmt ein wenig Strom. Dadurch kann jede Zelle des LIPO-Akkus gleich aufgeladen werden. Das ist wichtig, da der LIPO-Akku mehr Leistung erbringen kann, wenn alle Zellen auf die gleiche Spannung aufgeladen sind.

4.2.4 Ladekurve einer LIPO-Zelle

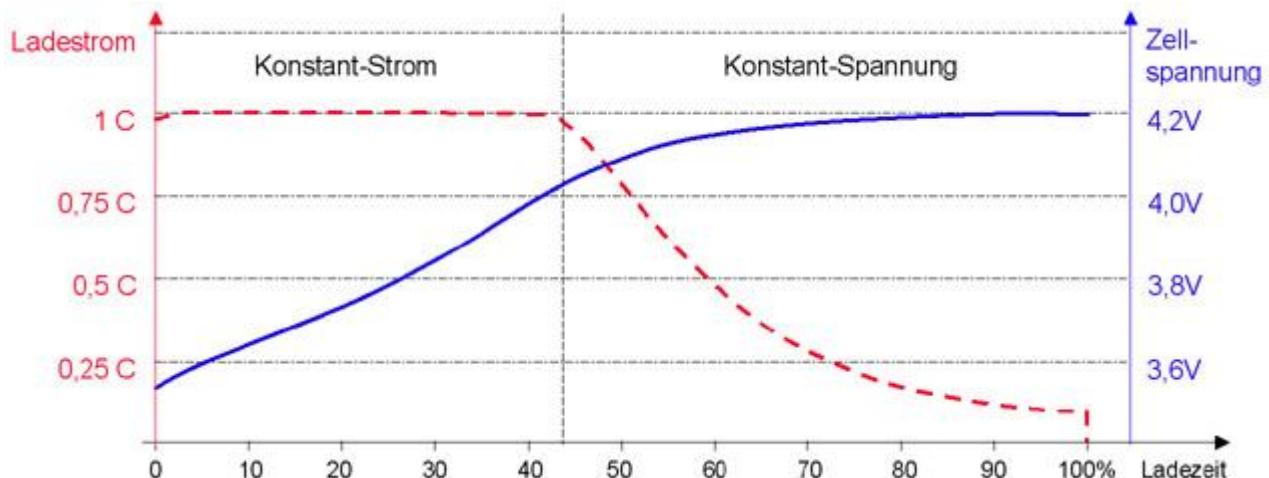


Abbildung 30: Aufladeverhalten einer LIPO-Zelle [AKKU]

Wie man anhand des Diagramms erkennen kann, werden die LIPO-Zellen von 0% bis etwa 45% der Ladezeit mit einem konstanten Strom von 1C (C-Rate) aufgeladen, was einen steilen Anstieg der Zellenspannung zur Folge hat. Nach den 45% Ladezeit, wird der Strom immer weiter reduziert, da nun das Ausgleichen (Balancen) der LIPO-Zellen beginnt. Wenn jede Zelle des LIPO-Akkus auf eine Spannung von 4,2V aufgeladen ist, gilt der Ladevorgang als beendet.

4.2.5 Entladekurve einer LIPO-Zelle

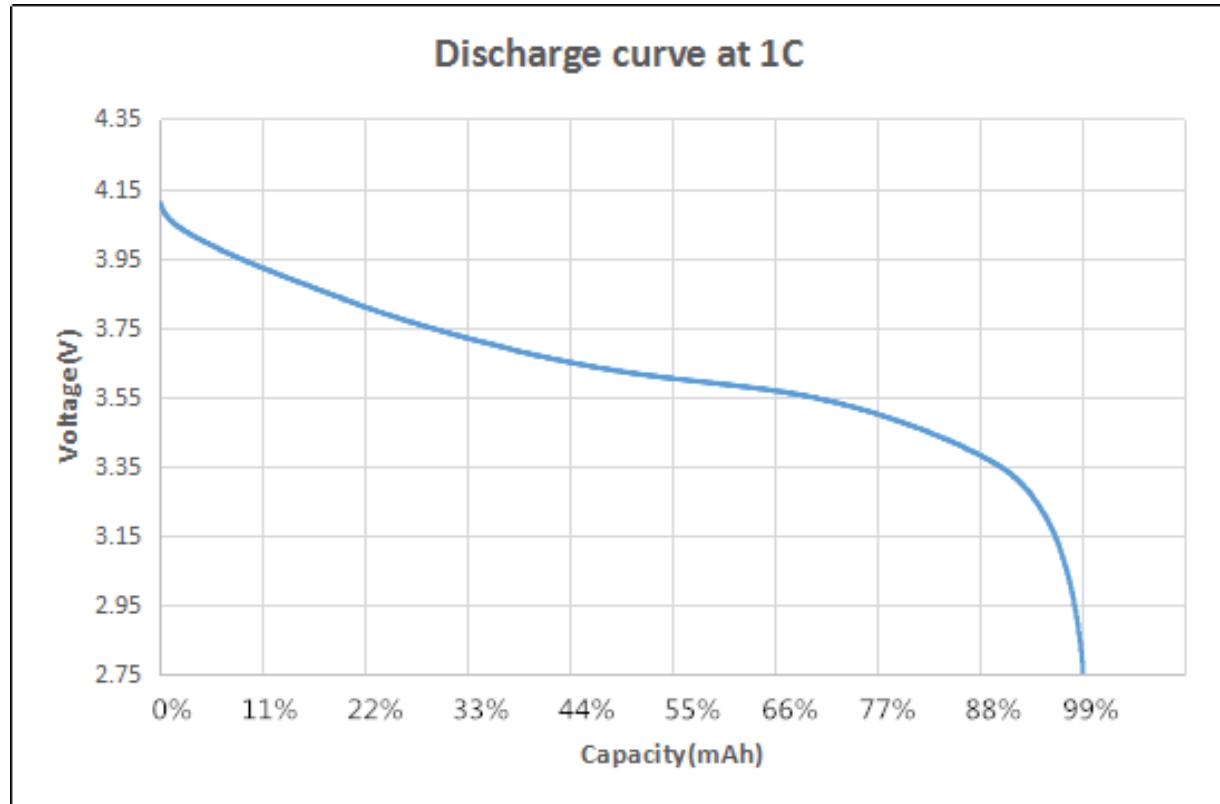


Abbildung 31: Entladeverhalten einer LIPO-Zelle [AKKUE]

Wie man anhand der Entladekurve sehen kann, entlädt sich eine LIPO-Zelle am Anfang schnell. Über die Zeit kann die LIPO-Zelle ihre Spannung einigermaßen halten, bis die Spannung in etwa bei 77% stark sinkt. Bei 88% Prozent beginnt dann der sogenannte Breakdown, wo die Zellenspannung rapide sinkt und die Zelle tiefenentladen ist.

4.2.6 Entladezeiten

Die theoretischen Entladezeiten der LIPO-Akkus wurden für 3 verschiedene Fälle berechnet:

1. Drohne im Stillstand (0% Gas: Stromverbrauch 700mA)
2. Drohne beim Abheben (20% Gas: Stromverbrauch 11,5A)
3. Drohne bei voller Leistung (100% Gas: Stromverbrauch 213,1A)

4.2.6.1 6S 1300mAh 120C LIPO

1. Drohne im Stillstand:

$$\text{Entladedauer} = \frac{1,3Ah}{0,7A} = 1,8h$$

2. Drohne beim Abheben:

$$\text{Entladedauer} = \frac{1,3Ah}{11,5A} = 6,7min$$

3. Drohne bei voller Leistung (da dieser Akku keine 213,1A liefern kann, wurde mit dem maximal lieferbaren Strom des Akkus gerechnet):

$$\text{Entladedauer} = \frac{1,3Ah}{156A} = 30s$$

4.2.6.2 5S 2200mAh 95C LIPO

1. Drohne im Stillstand:

$$\text{Entladedauer} = \frac{2,2\text{Ah}}{0,7\text{A}} = 3,1\text{h}$$

2. Drohne beim Abheben:

$$\text{Entladedauer} = \frac{2,2\text{Ah}}{11,5\text{A}} = 11,4\text{min}$$

3. Drohne bei voller Leistung (da dieser Akku keine 213,1A liefern kann, wurde mit dem maximal lieferbaren Strom des Akkus gerechnet):

$$\text{Entladedauer} = \frac{2,2\text{Ah}}{209\text{A}} = 37,9\text{s}$$

4.2.6.3 Fazit

Hierbei handelt es sich jedoch nur um theoretische Werte. In der Praxis werden 100% Gas nur selten erreicht. Realistisch ist eine Entladedauer zwischen dem Abhebewert und dem Vollgaswert.

4.2.6.4 Ergebnis

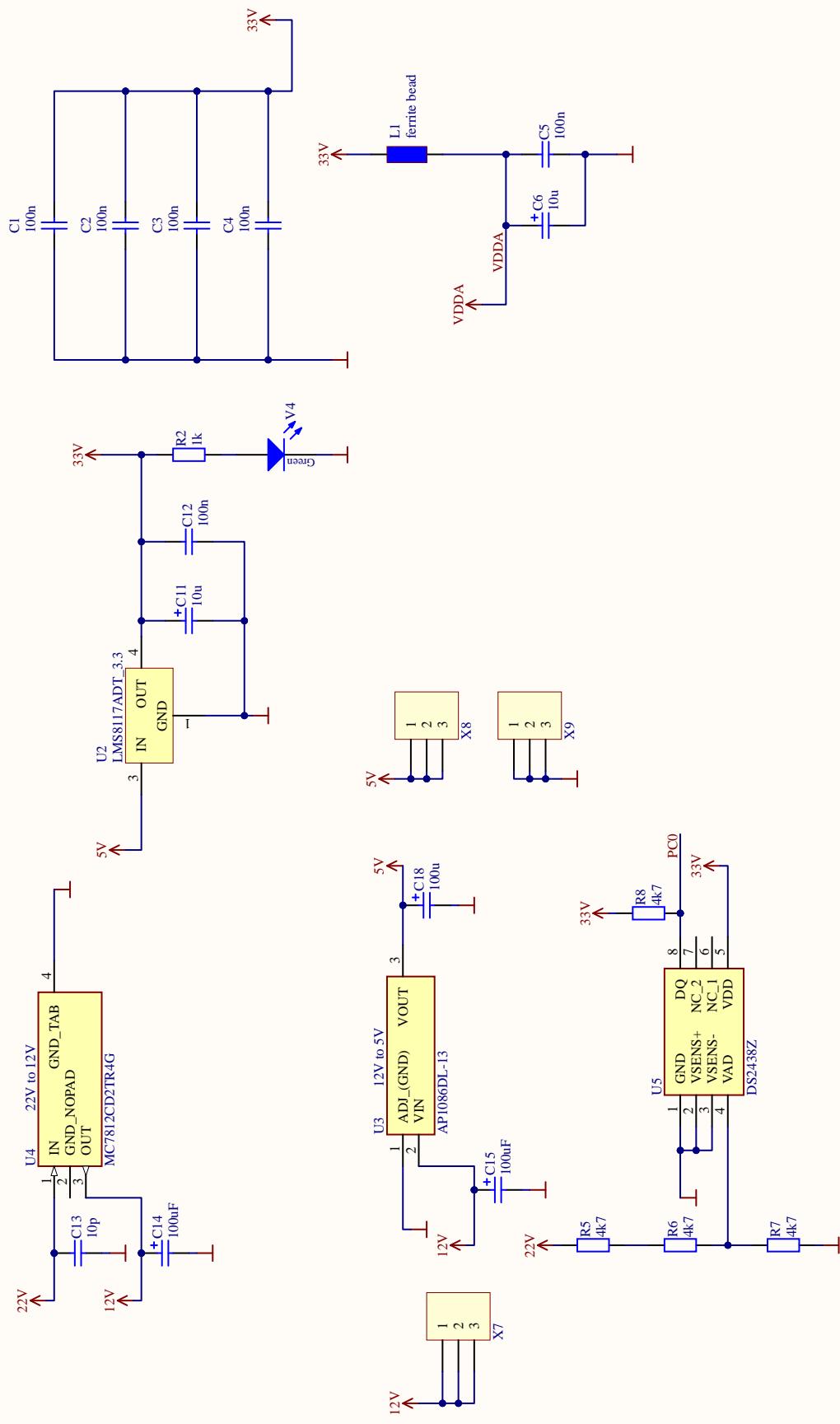
In der Realität wurden bei einem ruhigen Flug Zeiten zwischen 4 und 5 Minuten mit dem 6S 1300mAh 120C LIPO erreicht. Mit dem 5S 2200mAh 95C LIPO wurden Flugzeiten zwischen 7 und 8 Minuten gemessen.

4.3 Flight Controller

4.3.1 Allgemeines

Auf der Flight Controller Platine sitzt unser Mikrocontroller, der für die Steuer- und Regelaufgaben auf der Drohne zuständig ist. Des Weiteren befindet sich ein Smart Battery Monitor IC auf der Platine der ständig die Akkuspannung misst und vom Hauptprozessor eingelesen werden kann. Zusätzlich besitzt die Platine wichtige Schnittstellen um mit Receiver, IMU, ESC und VTx zu kommunizieren. Um die nötige Spannung für den Hauptprozessor und Sensoren bereitzustellen, wurden Fixspannungsregler verwendet. Der Flight Controller ist die Kombination und Weiterentwicklung von zwei einzelnen Platinen, der Mikrocontroller- und Sensorplatine.

4.3.2 Spannungsversorgungskonzept



Title: MCU PowerSupply			
Size: A4	Number: 1	Revision: V1.0	
Date: 11.02.2024	Time: 18:22:24	Sheet 2 of 3	
File: PowerSupply.SchDoc		Drawn By: Bieder Marcel	

3

2

1

Abbildung 32: Schematic Spannungsversorgung Flight Controller

4.3.2.1 12V Step Down Converter

Der 12V Step Down Converter hat die Aufgabe die Akkuspannung auf 12V abzusenken, da diese für den Video-Transmitter (VTx) gebraucht werden. Hierfür wurde der Fixspannungsregler MC7812CD2TR4G verwendet, da dieser einen stabilen 12V Pegel ausgeben kann und ein Stromlimit von 2,2A mit sich bringt, welches für unsere Zwecke ausreicht.

Gehäuse

Bei dem Gehäuse handelt es sich um das TO-263-3 Gehäuse, welches sich perfekt für unsere Verwendung eignet, da es SMD bestückt wird und es eine hohe Verlustleistung an die Umgebung abgeben kann.

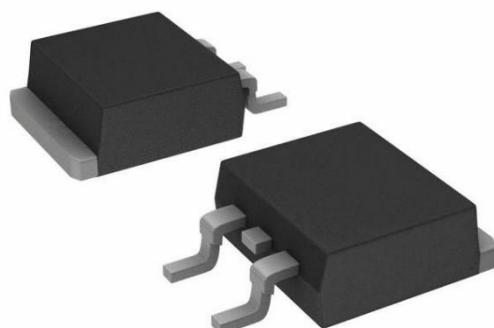


Abbildung 33: TO-263-3 Gehäuse [TO263]

Schematic

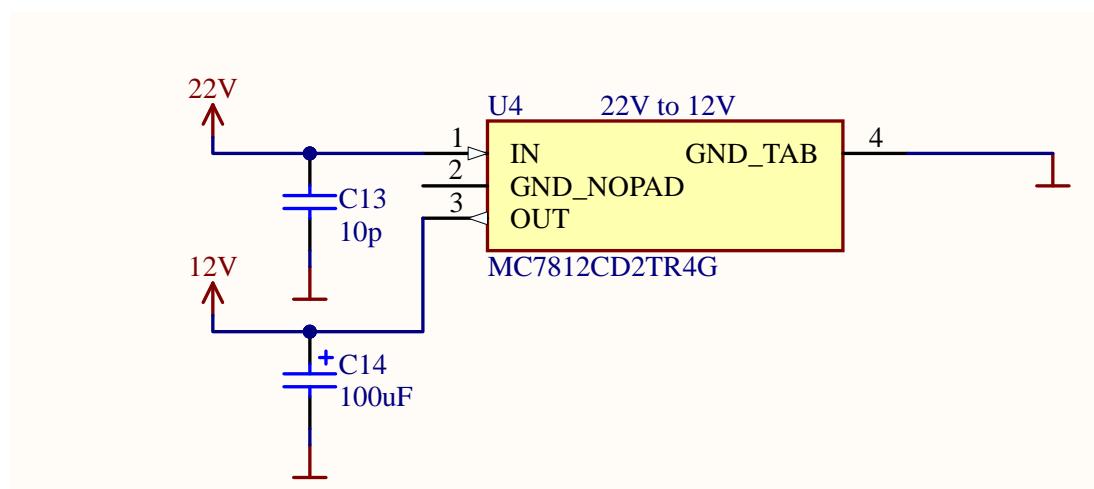


Abbildung 34: Schematic 12V Step Down Converter

Auf Pin 1 wird die Akkuspannung angelegt. Der Kondensator C13 leitet hierbei hochfrequente Störungen gegen Masse ab, damit sie nicht in den Fixspannungsregler gelangen. Auf Pin 3 wird der Ausgang geschalten, wo die heruntergeregelten 12V abgegriffen werden können. Der Kondensator C14 dient hierbei als Stützkondensator um die Ausgangsspannung zu glätten. Pin 4 wird auf Masse verbunden und Pin 2 wird nicht verbunden.

Eigenschaften

Maximale Eingangsspannung	35V
Maximaler Ausgangsstrom	2,2A
Betriebstemperatur	-40° bis 150°

4.3.2.2 5V Step Down Converter

Der 5V Step Down Converter hat die Aufgabe die 12V des vorherigen Fixspannungsreglers auf 5V abzusenken, da der IMU und der Receiver diese benötigen. Hierfür wurde der Fixspannungsregler AP1086DL-13 verwendet, da dieser einen konstanten 5V Pegel ausgibt und einen Strom von 1,5A liefern kann, welcher für unsere Anwendung ausreichend ist.

Gehäuse

Bei dem Gehäuse handelt es sich um das TO-252-3 Gehäuse, welches sich gut für unser Vorhaben eignet, da es SMD bestückt wird und sehr flach ist.

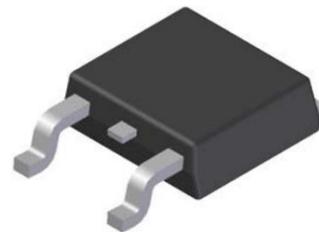


Abbildung 35: TO-252-3 Gehäuse [TO252]

Schematic

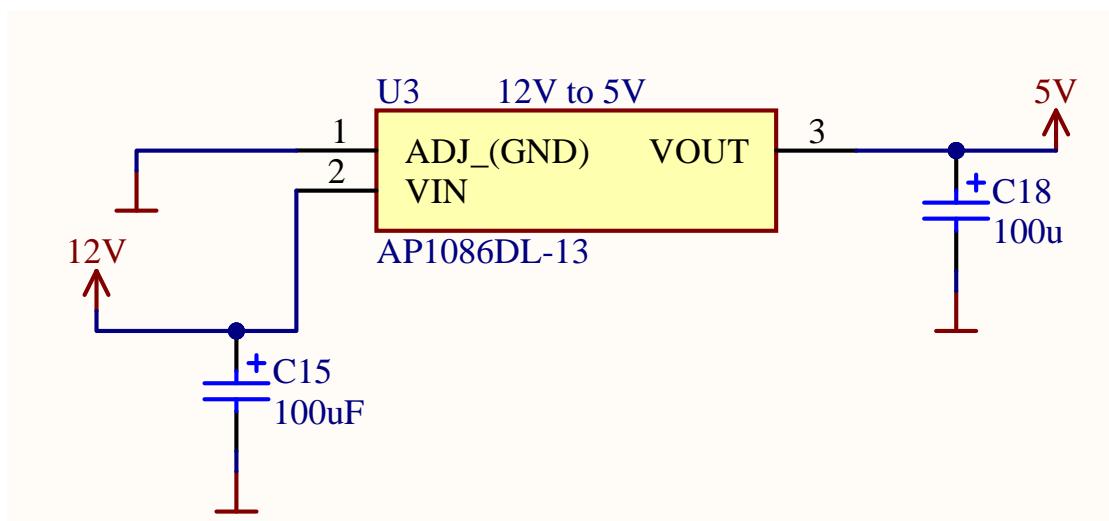


Abbildung 36: Schematic 5V Step Down Converter

Auf Pin 2 wird die Eingangsspannung von 12V angeschlossen. Der Kondensator C15 dient hierbei als zusätzlicher Stützkondensator um die Spannung zu glätten. Auf Pin 3 werden die heruntergeregelten 5V abgegriffen. Ebenfalls wurde am Ausgang der Kondensator C18 zur Spannungsglättung hinzugefügt. Pin 1 wird auf Masse verbunden.

Eigenschaften

Maximale Eingangsspannung	12V
Maximaler Ausgangsstrom	1,5A
Betriebstemperatur	0° bis 125°C

4.3.2.3 3,3V Step Down Converter

Der 3,3V Step Down Converter hat die Aufgabe die 5V des vorherigen Fixspannungsreglers auf 3,3V abzusenken, da diese für alle digitalen Bauteile gebraucht wird. Hierfür wurde der Fixspannungsregler LMS8117ADT_3.3 verwendet, da dieser einen konstanten 3,3V Pegel ausgibt und einen Strom von 1A liefern kann, welcher ohne Probleme ausreichen sollte.

Gehäuse

Bei dem Gehäuse handelt es sich um das TO-252-3 Gehäuse, welches sich gut für unser Vorhaben eignet, da es SMD bestückt wird und sehr flach ist.



Abbildung 37: TO-252-3 Gehäuse [TO252]

Schematic

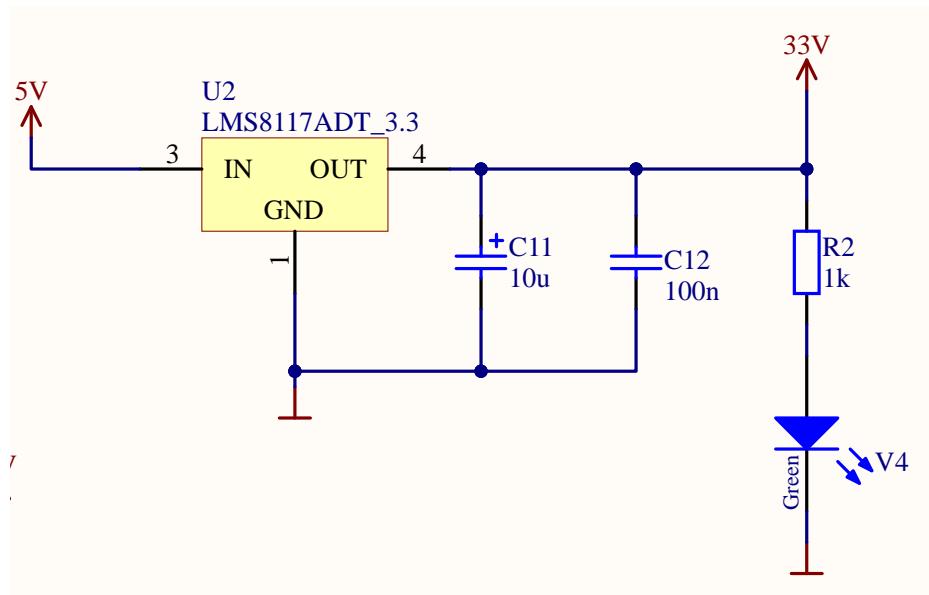


Abbildung 38: Schematic 3,3V Step Down Converter

Auf Pin 3 wird die Eingangsspannung von 5V angeschlossen. Auf Pin 4 wird der Ausgang mit den 3,3V angeschlossen. Der Kondensator C11 dient als Stützkondensator um die 3,3V zu glätten. Der Kondensator C12 dient als Entkoppelkondensator, um hochfrequente Störungen gegen Masse abzuleiten, damit diese nicht die restliche Schaltung beeinflussen. Pin 1 wird auf Masse angeschlossen. Zusätzlich wurde eine grüne LED V4 und ein Vorwiderstand R2 eingebaut, um eine visuelle Wiedergabe zu haben, ob Spannung anliegt.

Eigenschaften

Maximale Eingangsspannung	15V
Maximaler Ausgangsstrom	1A
Betriebstemperatur	0° bis 125°C

4.3.2.4 Versorgungsanschlüsse

Für alle externen Platinen und Bauteile wurden Versorgungsanschlüsse nach außen geführt.

Schematic

Zur Verfügung stehen:

- 3 * 12V Anschluss
- 3 * 5V Anschluss
- 3 * Masse Anschluss

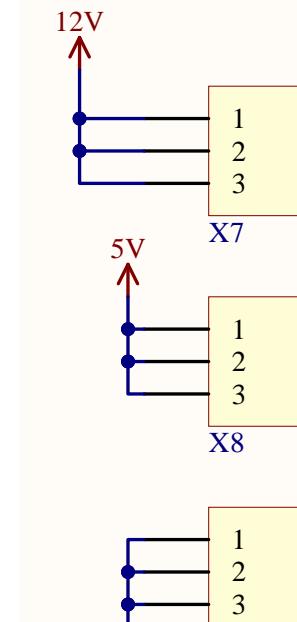


Abbildung 39: Schematic Versorgungsanschlüsse

4.3.2.5 Spannungsüberwachung

Die Aufgabe der Spannungsüberwachung ist es, stets die Akkuspannung der Drohne zu überwachen, damit früh genug gelandet werden kann, bevor der Akku sich vollständig entladen hat. Hierfür wurde der Smart Battery Monitor IC DS2438 verwendet.

DS2428

Der DS2438 ist ein Smart Battery Monitor, mit dem es möglich ist, Akkuspannung, Strom und Umgebungstemperatur zu messen. Dieser kommuniziert über eine Leitung mittels OneWire Protokoll mit dem Mikrocontroller. Bei dem Gehäuse handelt es sich hierbei um das 8-SIOC Gehäuse.

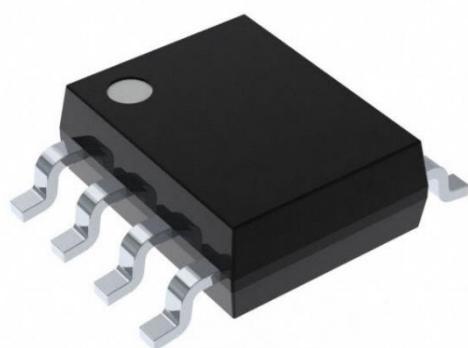


Abbildung 40: DS2438 Smart Battery Monitor [8SIOC]

Schematic

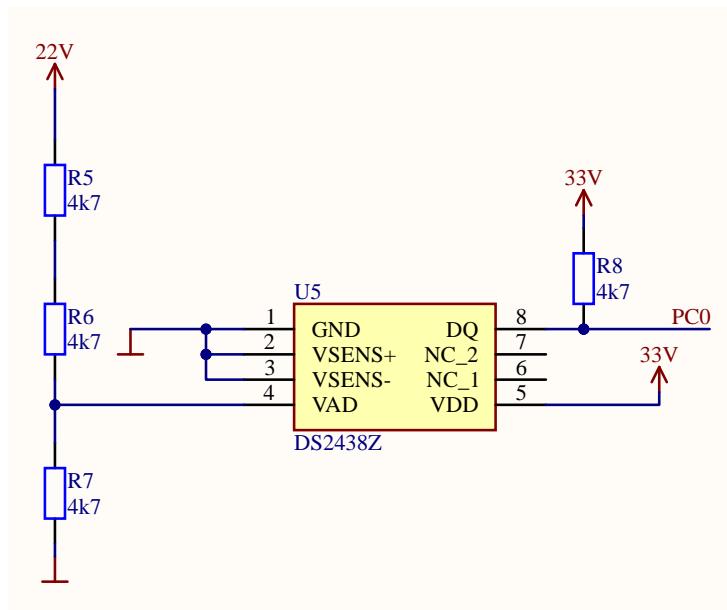


Abbildung 41: Schematic DS2438 Smart Battery Monitor

Pin 1 wird auf Masse verbunden. Pin 2 und 3 wären normalerweise für die Strommessung zuständig, da der DS2438 aber nur für die Spannungsüberwachung zuständig ist, werden die Pins auch auf Masse angeschlossen. Bei Pin 4 handelt es sich um den Pin für die Spannungsmessung. Da der DS2438 aber nur maximal 10V messen kann, wurde mit den Widerständen R5, R6 und R7 ein Spannungsteiler gebaut, damit die Grenze nicht überschritten werden kann. Pin 5 ist die externe Versorgung des DS2438. Pin 6 und 7 werden nicht verbunden. Bei Pin 8 handelt es sich um die Datenleitung zwischen DS2438 und Mikrocontroller mittels One-Wire Protokoll (siehe: [Kapitel 5.2.1](#)). Da das One-Wire Protokoll als Idle – Zustand einen High Pegel benötigt, dient der Widerstand R8 als Pullup-Widerstand.

4.3.2.6 Störfilter für Mikrocontroller

Zusätzlich wurden Maßnahmen getroffen, um Störungen bei den Versorgungspins des Mikrocontrollers zu minimieren.

Die vier Kondensatoren C1, C2, C3 und C4 dienen als Entkoppelkondensatoren, um hochfrequente Störungen an jedem Hauptprozessorversorgungspin gegen Masse abzuleiten. Zusätzlich können diese für kurze Zeit auch Strom liefern, wenn der Hauptprozessor plötzlich mehr braucht.

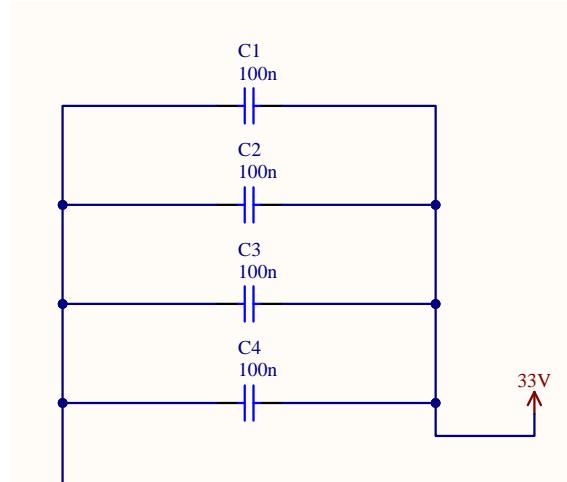


Abbildung 42: Schematic Störfilter für Mikrocontroller

4.3.2.7 Störfilter für VDDA-Pin

Der VDDA-Pin des Hauptprozessors dient zur Versorgung der ADCs innerhalb des Hauptprozessors. Dieser Eingang wurde zusätzlich nochmal gefiltert, um die bestmögliche Auflösung der ADCs zu gewährleisten.

Die Schaltung dient als LC-Tiefpass für den VDDA-Eingang des Hauptprozessors. Die Spule L1 sorgt dafür, dass nur tiefe Frequenzen passieren können. Zusätzlich leitet der Kondensator C5 hohe Frequenzen gegen Masse ab. Der Kondensator C6 dient hierbei als Stützkondensator um die gefilterte Spannung zu glätten.

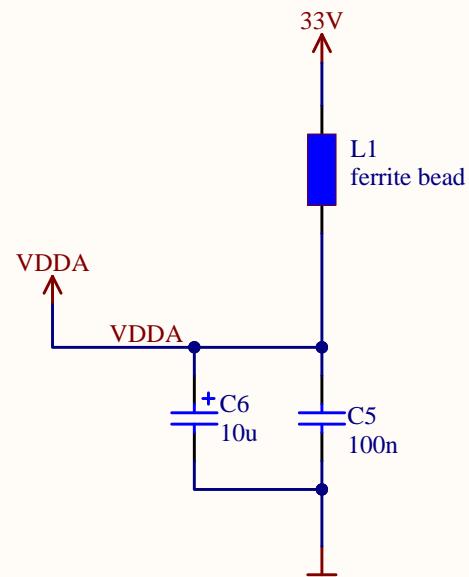


Abbildung 43: Schematic Störfilter für VDDA-Pin

4.3.3 Mikrocontroller

4.3.3.1 Allgemeines

Als Mikrocontroller wurde der STM32H7A3RG6 von ST Microelectronics gewählt. Dieser Mikrocontroller gehört zur ARM Cortex M7 Familie und besitzt eine 32Bit RISC-Rechenarchitektur. Mit seiner Taktfrequenz von 280MHz bietet er ebenso schnelle Verarbeitung seiner Steuer- und Regelaufgaben. Zusätzlich besitzt er 2MB Flash Speicher und 1,4MB SRAM.

4.3.3.2 STM32H7A3RG6

Gehäuse

Bei dem Gehäuse handelt es sich um das 64-LQFP Gehäuse. Dieses Gehäuse bietet 64 aus dem Gehäuse rausgeföhrte Pins, welche mit den Mitteln der HTL Hollabrunn gut lötbar sind.

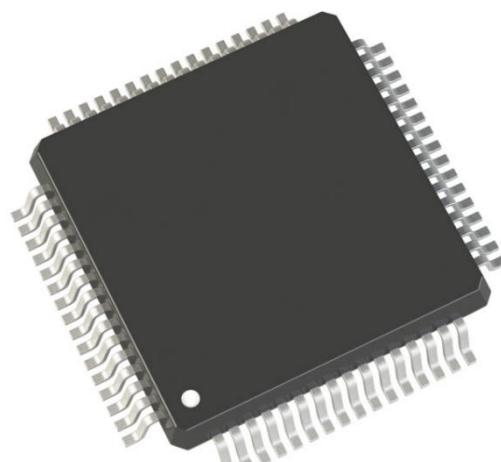


Abbildung 44: 64-LQFP Gehäuse [LQFP]

Schnittstellen

Schnittstelle	Anzahl
USART	5
SPI	6
I ² C	4
I ² S	1
CAN	2
USB OTG	1

Mikrocontroller Pinout

Der Punkt am Gehäuse markiert den 1er-Pin.

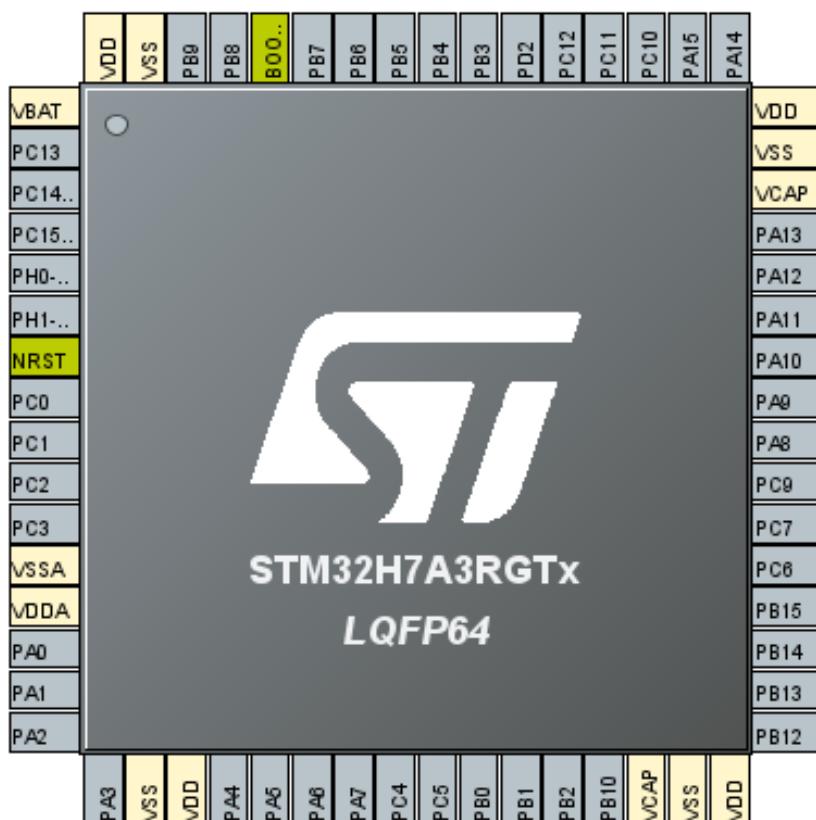


Abbildung 45: Mikrocontroller Pinout

Interner Aufbau des Mikrocontrollers

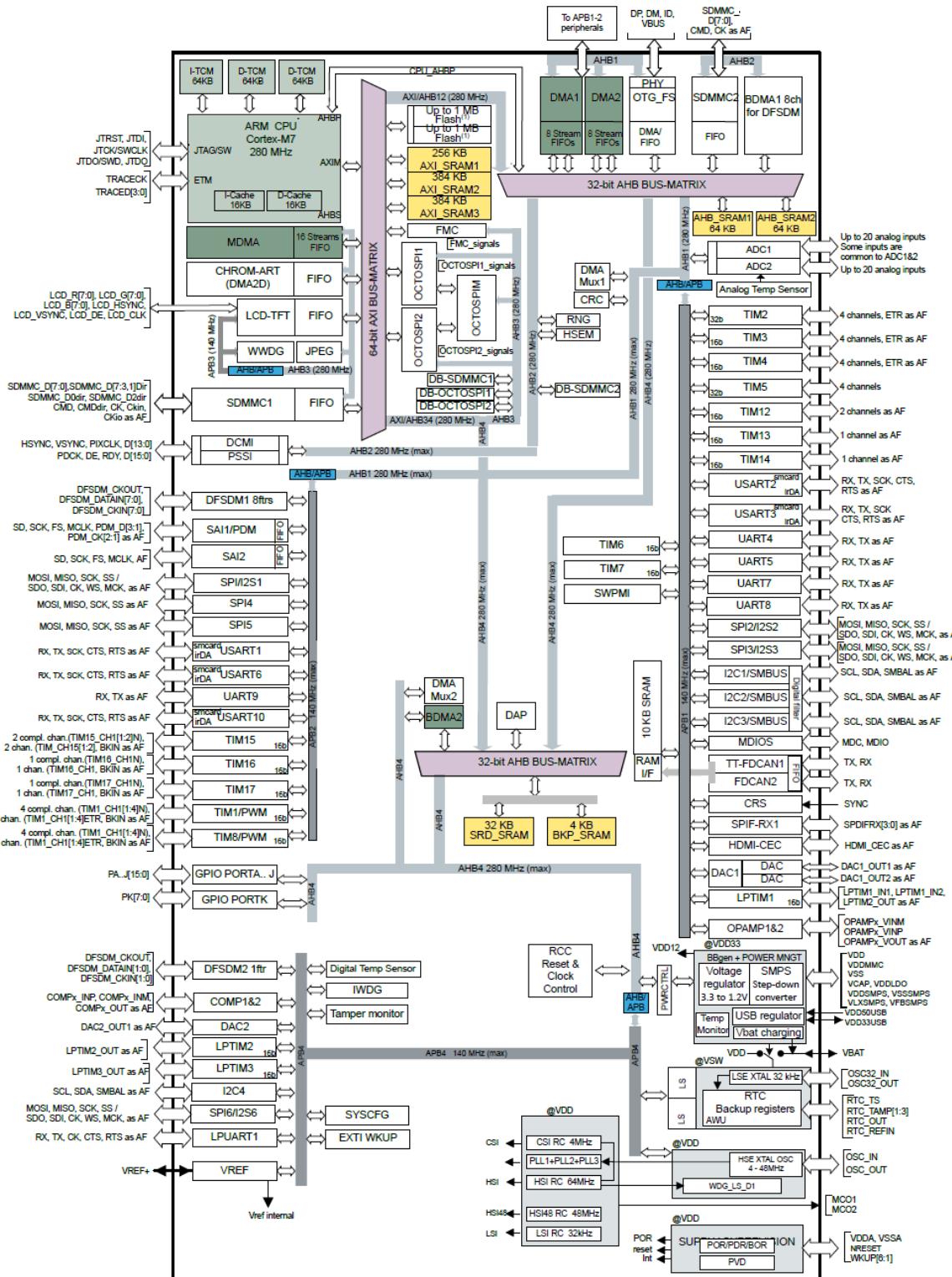


Abbildung 46: Interner Aufbau des Mikrocontrollers

Der interne Aufbau des Mikrocontrollers wurde aus dem Datenblatt entnommen. Hier werden alle GPIO-Ports, Speicher und Busse dargestellt.

4.3.3.3 Schematic

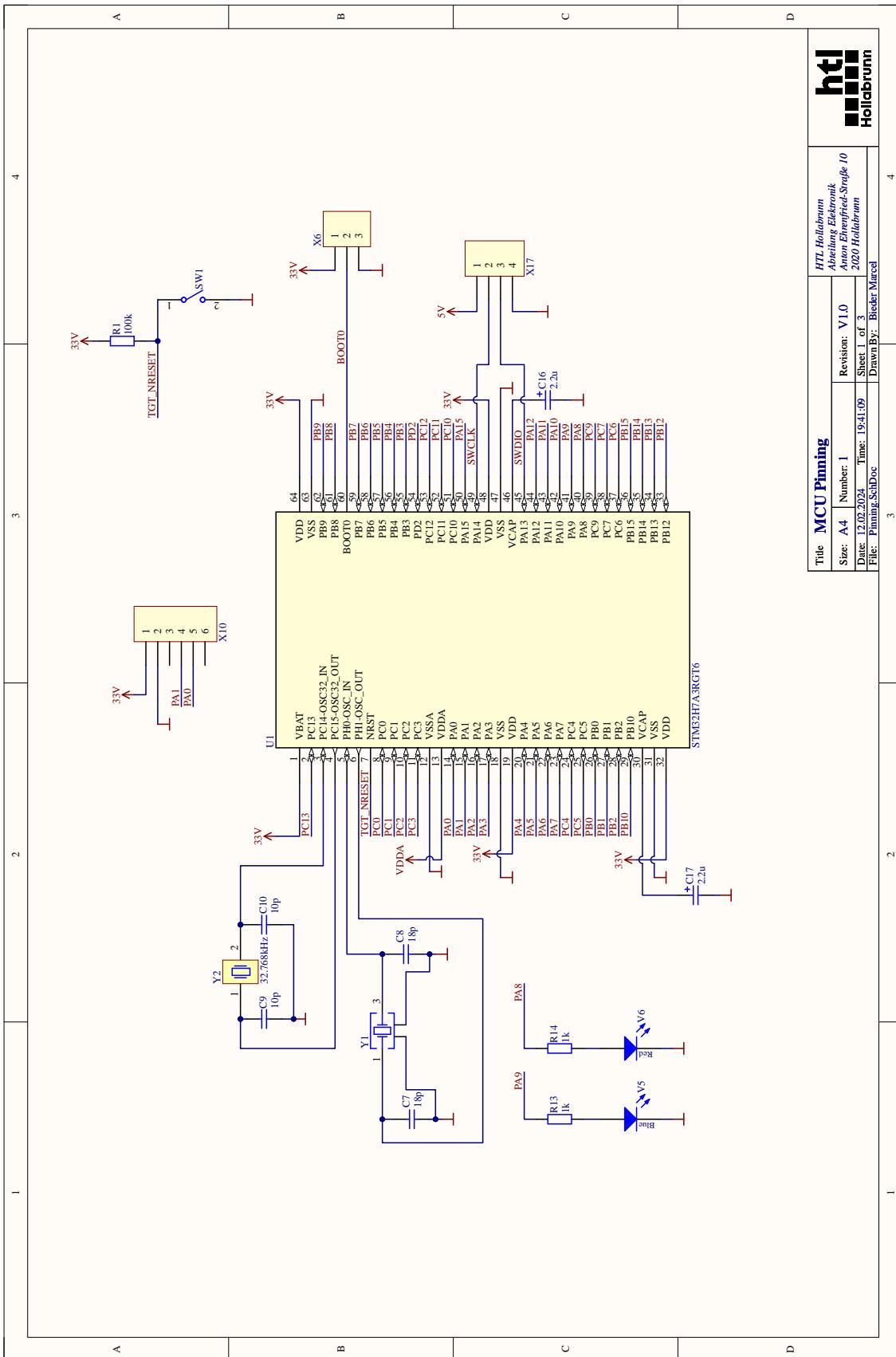


Abbildung 47: Schematic Pinning Flight Controller

Mikrocontroller Pinbelegung

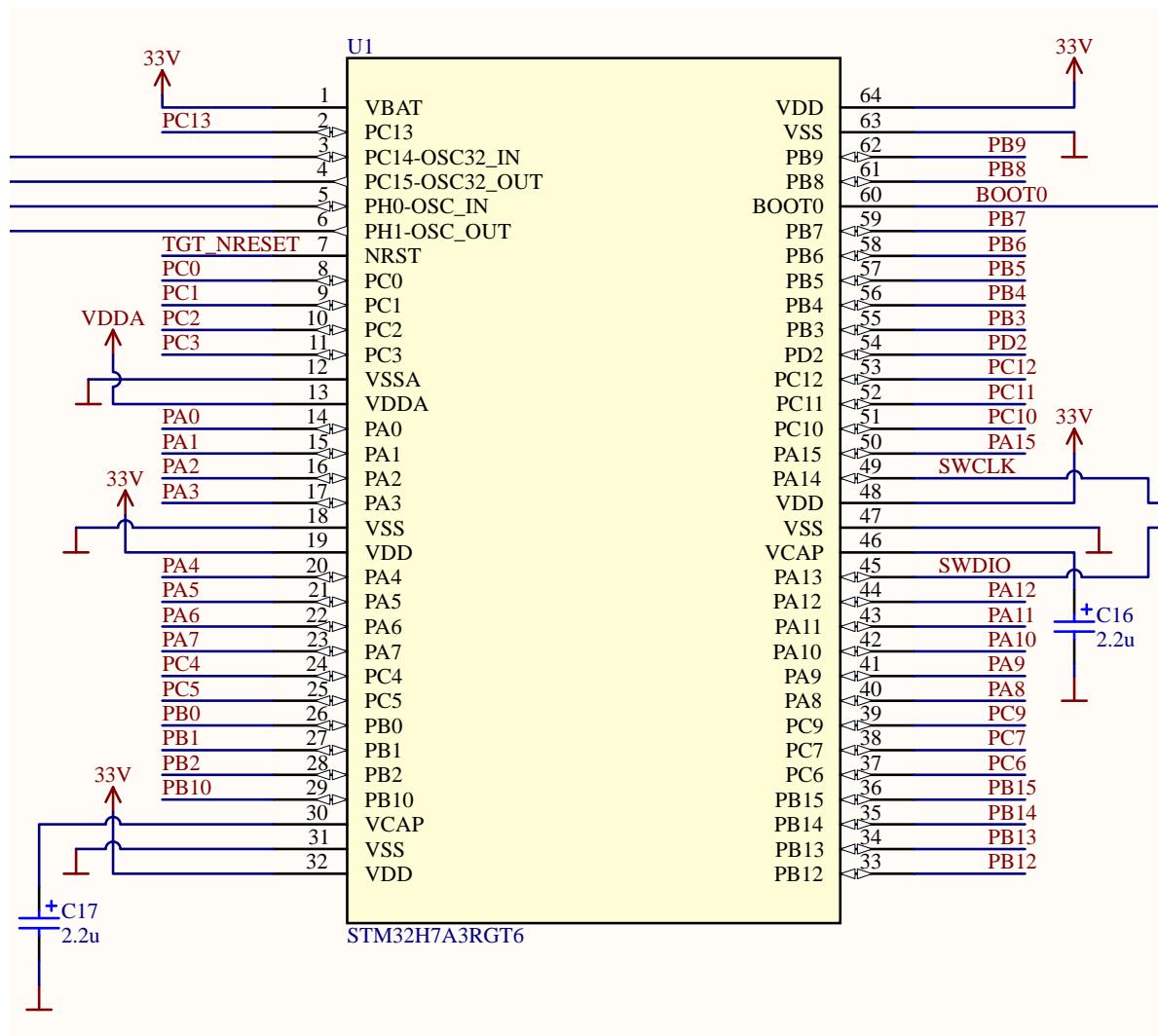


Abbildung 48: Schematic Mikrocontroller Pinbelegung

Die Versorgungspins des Mikrocontrollers (VDD) werden alle mit 3,3V verbunden. Dazu werden alle Masseleitungen des Mikrocontrollers (VSS) mit Masse verbunden. An den Ports PC14, PC15, PH0 und PH1 werden die externen Oszillatoren angeschlossen. An den NRST-Pin des Mikrocontrollers wird ein Reset Button angeschlossen. An den BOOT0 Pin des Mikrocontrollers wird eine Stiftleiste angeschlossen, um den Bootloader zu konfigurieren. Die Ports PA14 und PA15 werden für die Programmierung des Mikrocontrollers verwendet. An den zwei VCAP-Pins wurden Stützkondensatoren angeschlossen, die jederzeit zusätzlichen Strom liefern können, wenn der Mikrocontroller diesen benötigt. Die restlichen Ports sind frei belegbar und werden für verschiedene Sensoren und Aktoren verwendet.

Oszillatoren

Für die Taktversorgung des STM32H7A3RGT6 werden zwei externe Quarzoszillatoren verwendet.

Für den normalen Betrieb wird ein 8MHz Oszillator an Port PH0 (Oscillator IN) und Port PH1 (Oscillator OUT) angeschlossen. Diese 8MHz werden intern mittels einer PLL (Phase locked Loop) auf 280MHz erhöht. Um eine konstante Schwingung zu garantieren, wurden zwei 18pF Kondensatoren eingebaut, um wie im Datenblatt empfohlen, die Stabilität zu verbessern.

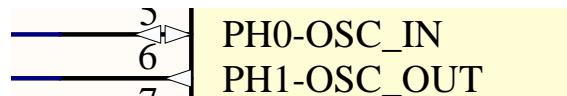


Abbildung 49: Schematic HSE Oszillator Mikrocontroller Pinning

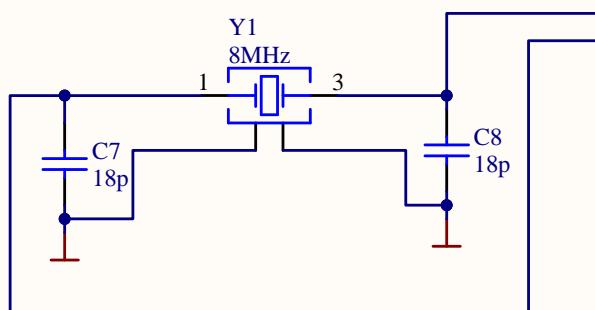


Abbildung 50: Schematic HSE Oszillator

Zusätzlich wurde ein 32,768kHz Oszillator für die interne RTC-Einheit bereitgestellt, die für speziell langsame Aufgaben genutzt werden kann. Dieser Oszillator wird an den Port PC14 (Oscillator 32 IN) und Port PC15 (Oscillator 32 OUT) angeschlossen. Ebenfalls wurden hier, wie im Datenblatt empfohlen, zwei 10pF Kondensatoren verbaut, um die Stabilität der Schwingung zu verbessern.

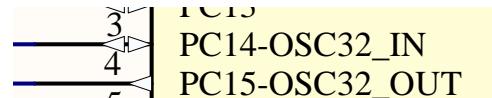


Abbildung 51: Schematic LSE Oszillator Mikrocontroller Pinning

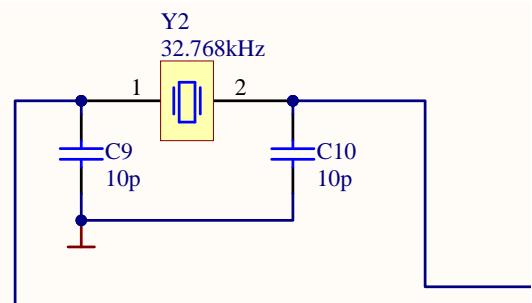


Abbildung 52: Schematic LSE Oszillator

Bootloader

Um den Bootloader von außerhalb konfigurieren zu können, wurde eine Stiftleiste mit dem BOOT0-Pin des Mikrocontrollers verbunden. Mit einem Jumper kann eingestellt werden, ob ein HIGH- oder LOW Pegel am BOOT0-Pin anliegen soll. Je nachdem, was angeschlossen wird, bootet der Mikrocontroller von einer anderen Speichereinheit.

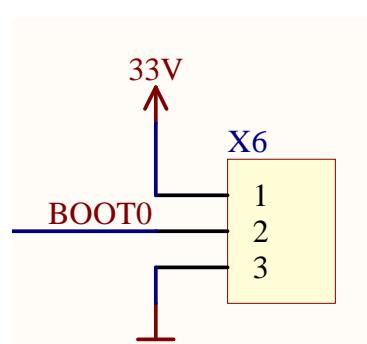


Abbildung 53: Schematic Bootloader

Reset Button

Der Reset Button wird verwendet, um einen Reset Impuls zu generieren, der den Mikrocontroller resettet. Das ist notwendig, um den Mikrocontroller nach dem flashen oder einem Fehler wieder in Betrieb zu nehmen. Bei dem Reset handelt es sich um einen low-aktiven Reset, was bedeutet, dass dieser durchgeführt wird, wenn die Leitung auf Masse gezogen wird. Der Widerstand R1 dient als Pull-Up Widerstand, damit die Leitung während des Betriebes auf einen High Pegel gezogen wird.

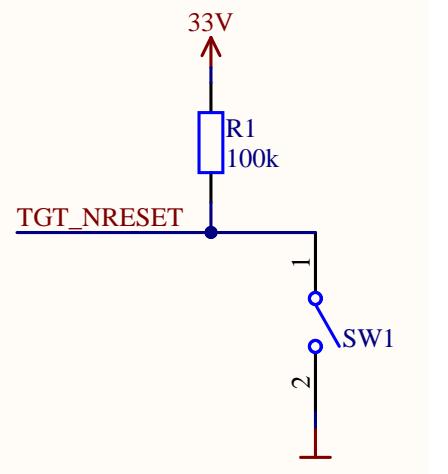


Abbildung 54: Schematic Reset Button

Programmierung

Um den Flight Controller so klein wie möglich zu bauen, wurde entschieden, das DAPLINK Modul nicht auf der Platine zu platzieren, sondern mit einem externen Modul zu flashen.

Programmierschnittstelle:

Um über das externe DAPLINK Modul zu flashen, müssen 4 Leitungen verbunden werden. An erster Stelle der Stiftleiste befindet sich der 5V Pin. Dieser ist für die Versorgung während des Flashvorganges gedacht, wenn kein Akku an der Drohne angeschlossen ist. Der nächste Pin ist die SWCLK (Serial Wire Clock) Leitung, die für den Takt während des Flashvorgangs vorgesehen ist. Diese wird vom Mikrocontroller benötigt, um am richtigen Zeitpunkt die Daten einzulesen. An Pin 3 finden wir die SWDIO (Serial Wire Debug Input/Output) Leitung, welche für die Datenübertragung zwischen DAPLINK Modul und Mikrocontroller zuständig ist. Der letzte Pin ist Masse.

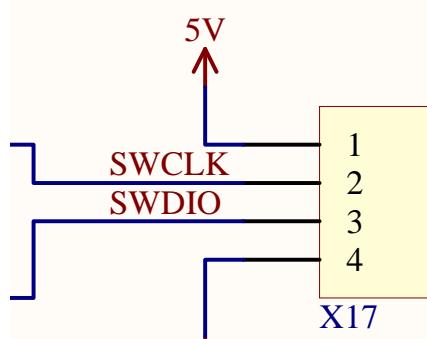


Abbildung 55: Programmierschnittstelle Mikrocontroller

Terminal

Um Messwerte und Fehlercodes während des Testens besser auszulesen, wurde eine Terminal Schnittstelle entworfen. Diese ist kompatibel mit dem UART/USB Converter der HTL Hollabrunn. An Pin 1 liegt die Versorgung von 3,3V für die Adapter Platine, daneben liegt Masse. Bei Pin 4 handelt es sich um UART4 Rx und bei Pin 5 um UART4 Tx. Diese Pins sind für die Kommunikation mit dem Terminal.

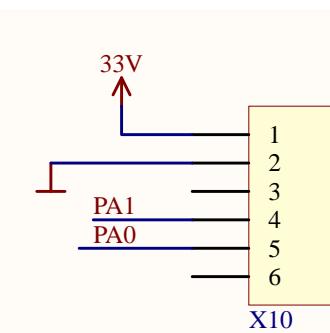


Abbildung 56: Schematic Terminalschnittstelle

UART/USB-Converter:

Der UART-USB-Converter wandelt die per UART gesendeten Daten um, damit diese von einem PC mittels USB erfasst werden können.

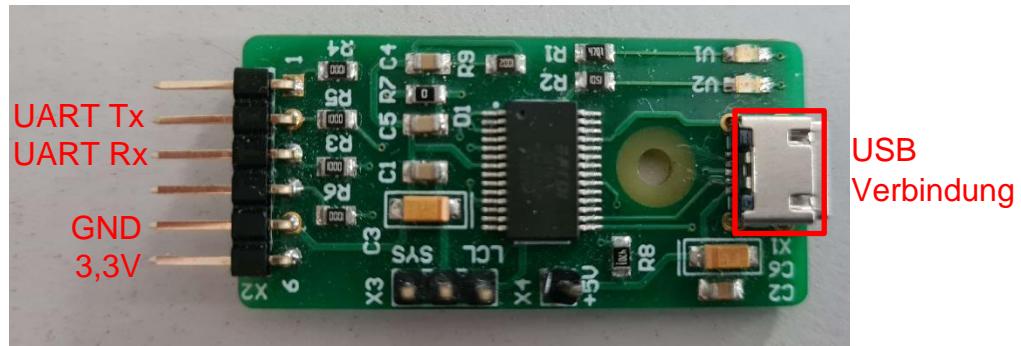


Abbildung 57: UART /USB-Converter

Status LEDs

Um verschiedene Status oder Fehler während der Initialisierung zu erkennen, werden zwei Status LEDs eingebaut. An Port PA9 des Mikrocontrollers wurde eine blaue LED angeschlossen. Diese fängt an zu blinken, wenn die Software startet und alle Sensoren initialisiert werden. Wenn die Initialisierung abgeschlossen ist, fängt die blaue LED an zu leuchten. An Port PA8 des Mikrocontrollers wurde eine rote LED angeschlossen. Diese fängt an zu leuchten, wenn während der Initialisierung ein Fehler aufgetreten ist. Zusätzlich beginnt diese zu blinken, wenn während des Fluges die Verbindung zur Fernsteuerung abgebrochen ist.

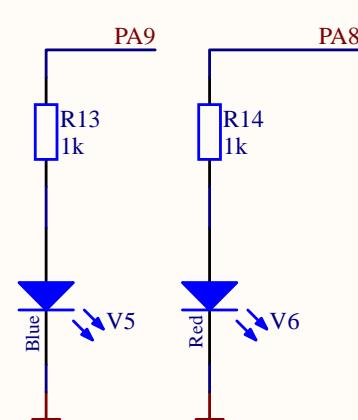


Abbildung 58: Schematic Status LEDs

Anschlüsse zu externen Komponenten

Die externen Anschlüsse zu Komponenten wie ESC, IMU, VTx und Receiver wurden erst auf dem PCB designiert.

4.3.4 Altium PCB Design

Im folgenden Kapitel werden die entworfenen PCBs im Detail erklärt.

4.3.4.1 Allgemeines

Damit die Flight Controller Platine in die Mitte der Drohne reinpasst, durfte diese bestimmte Maße nicht überschreiten. Um den vollen Platz auf der Drohne auszunutzen, wurde entschieden, ein quadratisches Design mit den Maßen 43,5mm * 43,5mm zu designen.

4.3.4.2 Gesamtübersicht

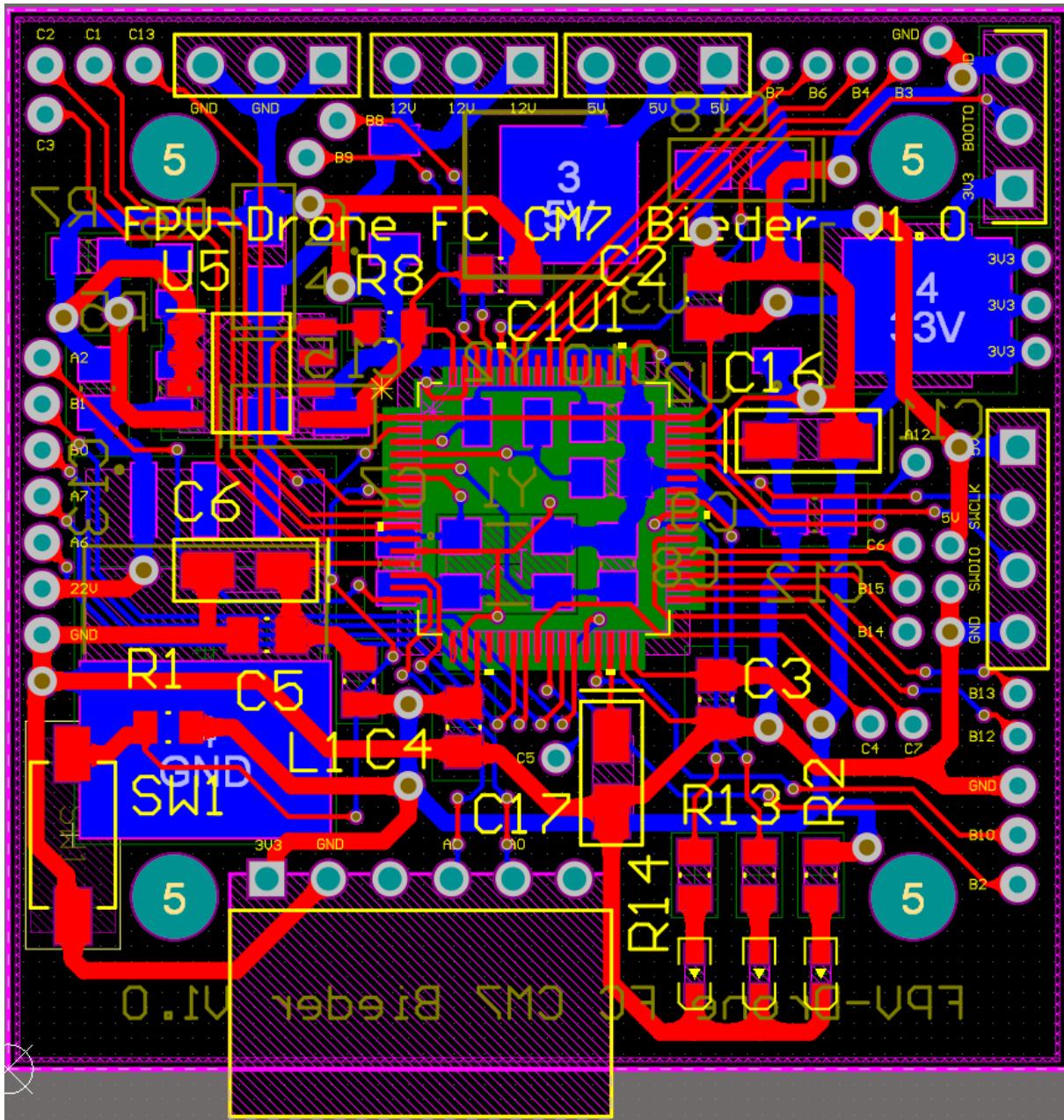


Abbildung 59: PCB Ansicht Gesamtdesign Flight Controller

Zur vereinfachten Ansicht werden das Top Layer und Bottom Layer der Flight Controller Platine auch separat dargestellt:

4.3.4.3 Top Layer

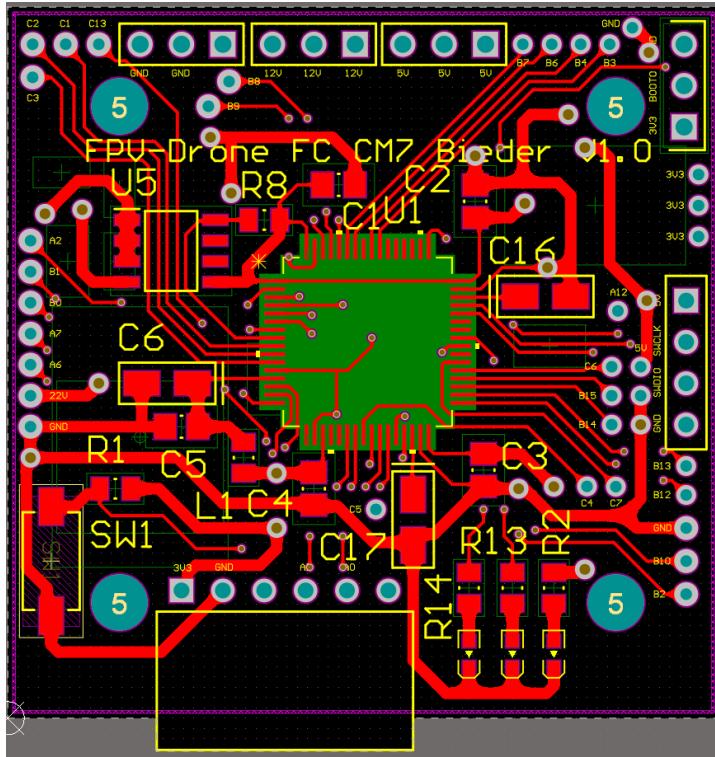


Abbildung 60: PCB Ansicht Top Layer Flight Controller

4.3.4.4 Bottom Layer

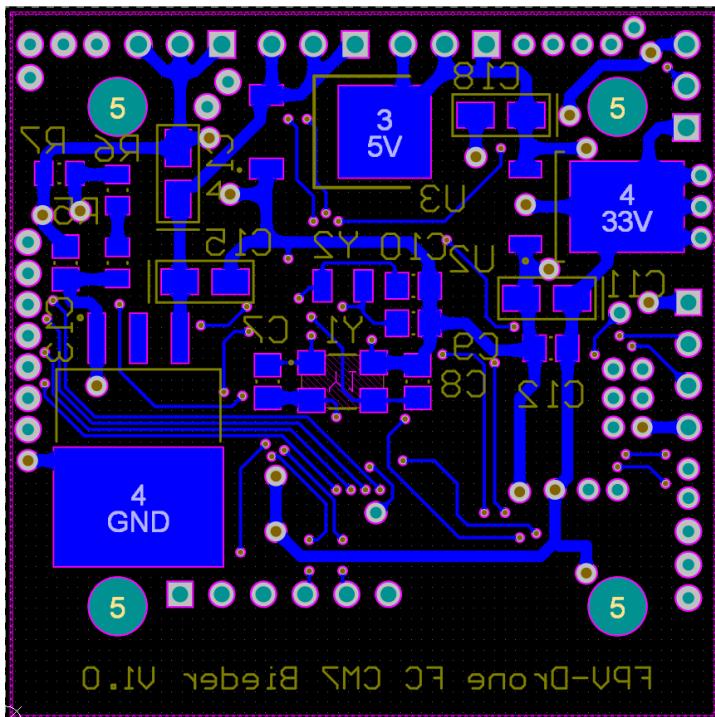


Abbildung 61: PCB Ansicht Bottom Layer Flight Controller

4.3.4.5 Altium 3D Ansicht

Top Layer

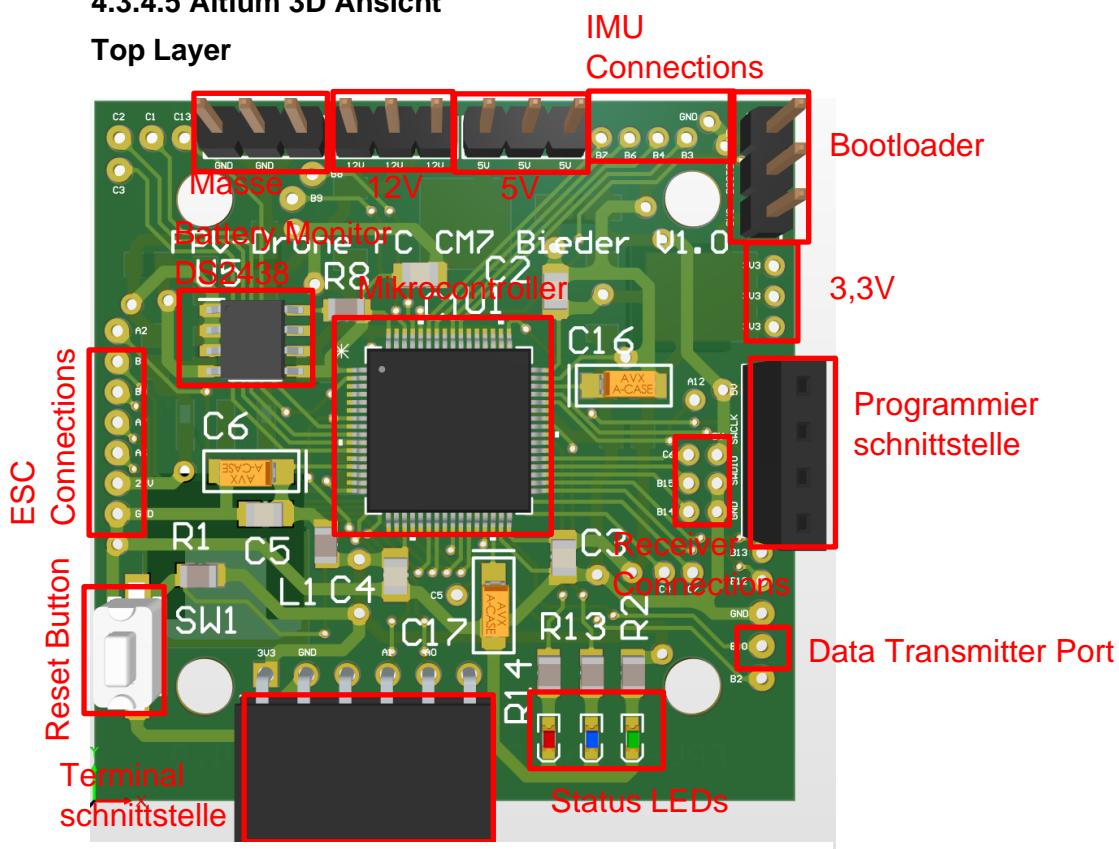


Abbildung 62: PCB 3D Ansicht Flight Controller

Bottom Layer

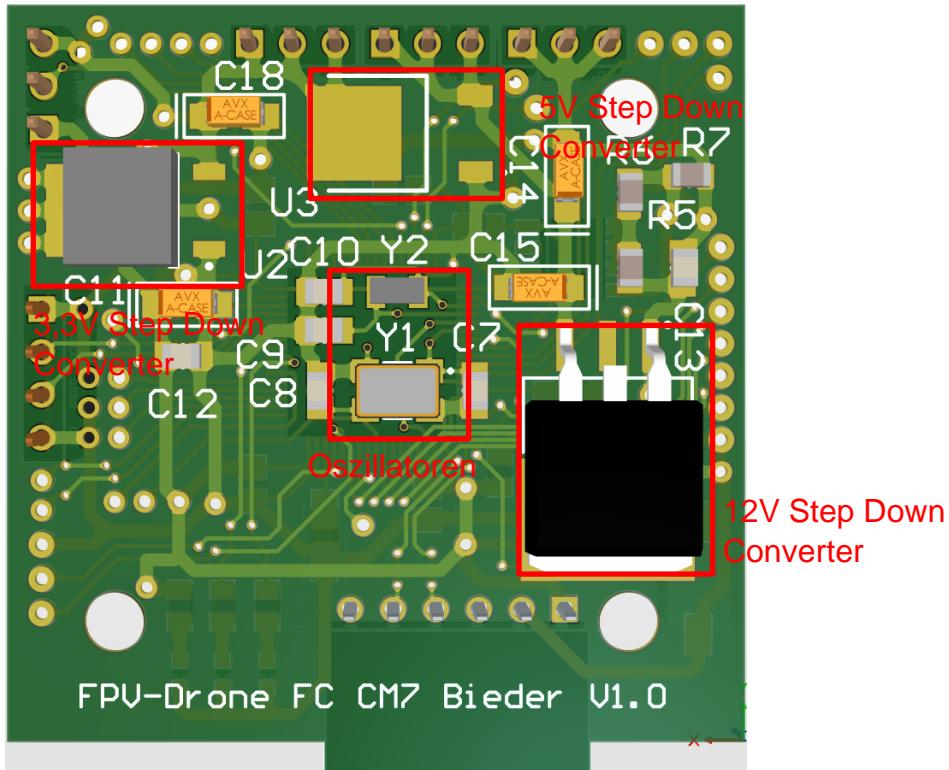
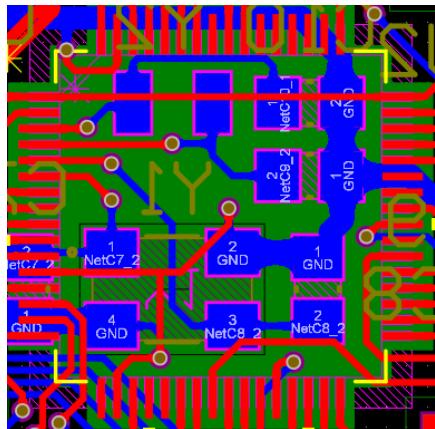


Abbildung 63: PCB 3D Ansicht Flight Controller

4.3.4.6 Platzierung der Bauteile

Durch die begrenzte Größe der Flight Controller Platine wurden die Bauteile sehr nah zusammen und auf Ober- und Unterseite platziert. Außerdem wurde sich dazu entschieden, alle Step Down Converter auf die Unterseite der Platine zu platzieren, um die ganze Spannungswandlung rein auf der Unterseite zu belassen. Auf der Oberseite der Platine wurden alle digitalen Bauteile, wie Mikrocontroller und Battery Monitor platziert. Der Reset Button wurde am Rand der Platine platziert. Alle Verbindungen zu externen Komponenten wurden ebenfalls am Rand der Platine platziert, um diese gut erreichbar zu machen.

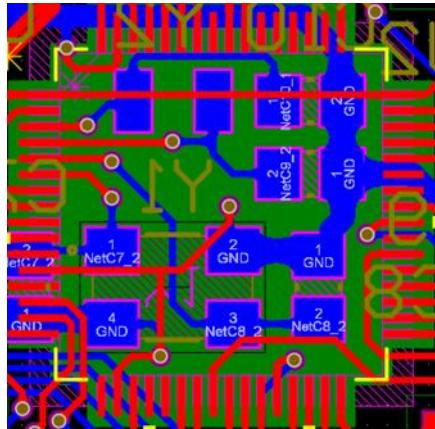
Mikrocontroller



Der Mikrocontroller wurde auf der Oberseite, in der Mitte der Platine platziert, um alle Pins gut zugänglich zu designen. Dieser wurde im 64-LQFP Gehäuse verbaut. Der Punkt am Gehäuse markiert den 1er Pin und muss mit dem Stern auf der Platine übereinstimmen.

Abbildung 64: Platzierung Mikrocontroller

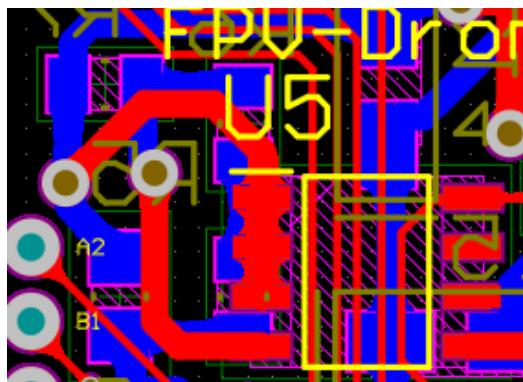
Oszillatoren



Die externen Oszillatoren für den Mikrocontroller wurden auf der Unterseite der Platine, direkt unter dem Mikrocontroller platziert. Dadurch können die Takteleitungen zum Controller sehr kurz gehalten werden und sind weniger störanfällig. Auf die korrekte Ausrichtung der Oszillatoren muss nicht geachtet werden, da die Pinbelegung korrekt ist, egal wie man sie platziert.

Abbildung 65: Platzierung Oszillatoren

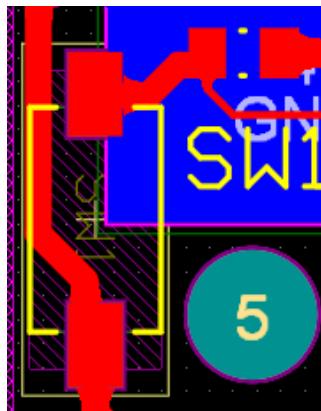
Smart Battery Monitor DS2438



Der Smart Battery Monitor DS2438 wurde direkt neben der Verbindung zur ESC platziert, wo auch die Akkuspannung herkommt. Dieser wurde auf der Oberseite platziert. Der Spannungsteiler für die Spannungsmessung liegt auf der Unterseite der Platine, direkt unter dem DS2438, um die Leiterbahnen kurz zu halten. Der 1er Pin am Gehäuse des DS2438 wird mit einem Punkt markiert. Dieser muss mit dem Strich auf der Platine übereinstimmen.

Abbildung 66: Platzierung DS2438

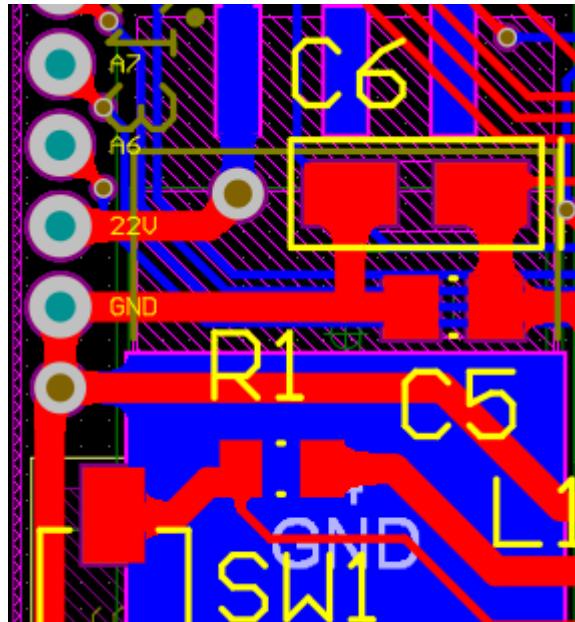
Reset Button



Der Reset Button wurde für bessere Zugänglichkeit auf der Drohne an der Seite der Platine platziert. Der dazugehörige Pull Up Widerstand wurde direkt neben dem Reset Button platziert.

Abbildung 67: Platzierung Reset Button

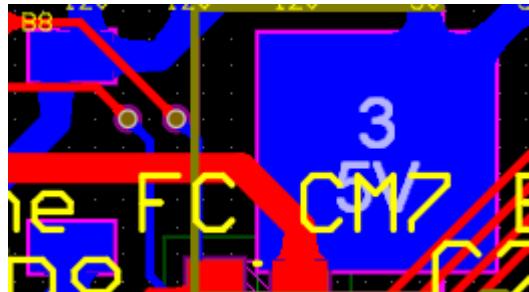
12V Step Down Converter



Der 12V Step Down Converter wurde auf der Unterseite der Platine, direkt neben der Verbindung zur ESC platziert, da hier auch die Akkuspannung herkommt. Die große Fahne des Converters, welche mit GND verbunden ist, dient als Kühlfläche.

Abbildung 68: Platzierung 12V Step Down Converter

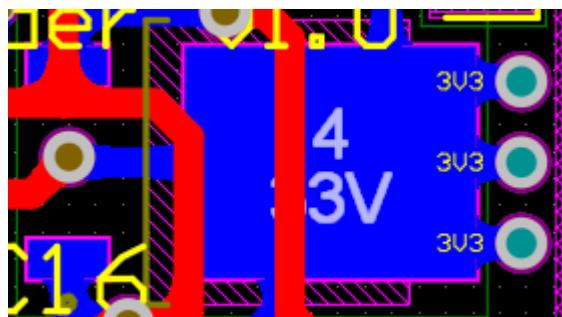
5V Step Down Converter



Der 5V Step Down Converter wurde auf der Unterseite am Rand der Platine, direkt neben den 5V Connector platziert, um die Leiterbahnen kurz zu halten. Die Fahne des Gehäuses des Converters, welche mit 5V verbunden ist, dient als Kühlfläche.

Abbildung 69: Platzierung 5V Step Down Converter

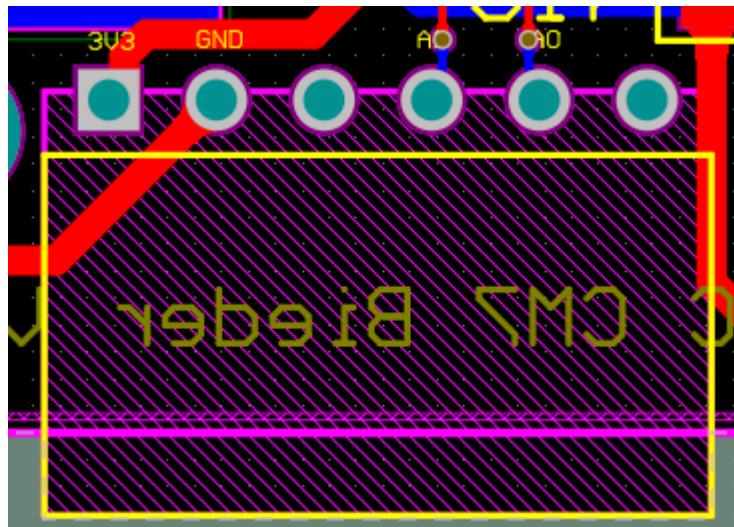
3,3V Step Down Converter



Der 3,3V Step Down Converter wurde ebenfalls auf der Unterseite am Rand der Platine platziert, um 3,3V Anschlüsse am Rand designen zu können, ohne lange Leiterbahnen. Die Fahne des Gehäuses des Converters, welche mit 3,3V verbunden ist, dient als Kühlfläche.

Abbildung 70: Platzierung 3,3V Step Down Converter

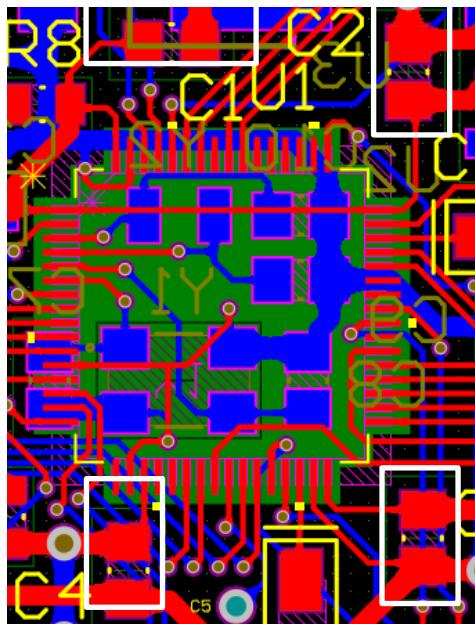
Terminal Connector



Der Terminal Connector wurde auf der Oberseite am Rand der Platine platziert, um auch einen Terminalzugang zu gewähren, wenn die Drohne zusammengebaut ist. Als Connector wurde eine 6 polige 90°-angewinkelte Buchsenleiste verwendet.

Abbildung 71: Platzierung Terminal Verbindung

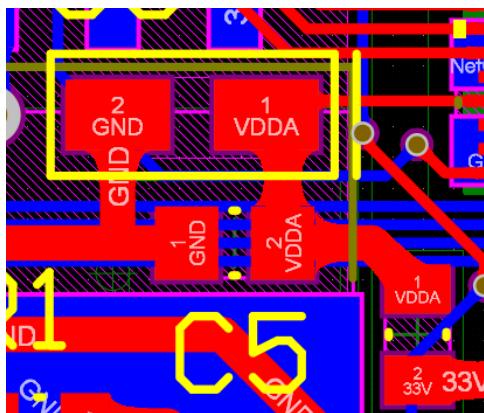
Störfilter für Mikrocontroller



Jeder der 4 Kondensatoren (weiß markiert) wurde auf der Oberseite der Platine an einen anderen Eingang des Mikrocontrollers angeschlossen und dienen als Entkoppelkondensatoren. Um die Störungen so gut wie möglich gegen Masse abzuleiten, ist es wichtig, die Kondensatoren so nah wie möglich am Mikrocontroller zu platzieren. Außerdem können diese Kondensatoren zusätzlichen Strom liefern, wenn der Mikrocontroller plötzlich mehr braucht.

Abbildung 72: Platzierung Störfilter für Mikrocontroller

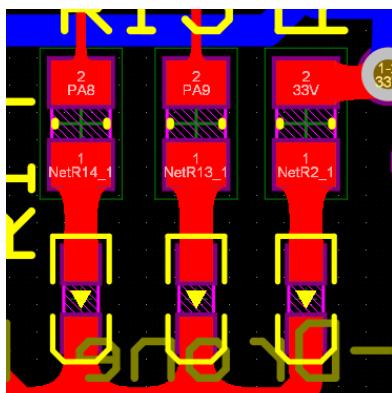
Störfilter für VDDA-Pin



Der Tiefpass, bestehend aus einer Spule und zwei Kondensatoren, wurde auf der Oberseite der Platine möglichst nah am VDDA-Pin des Mikrocontrollers platziert. Dadurch werden hochfrequente Störungen in der Versorgungsspannung bestmöglich rausgefiltert und gegen Masse abgeleitet.

Abbildung 73: Platzierung Störfilter für VDDA-Pin

Status LEDs



Die Status LEDs wurden auf der Oberseite am Rand der Platine platziert. Dadurch kann man die LEDs, auch wenn die Drohne im zusammengebauten Zustand ist, immer noch gut erkennen. Ganz rechts befindet sich die grüne LED, die leuchtet, wenn 3,3V am System anliegen. In der Mitte liegt die blaue LED, die für die Anzeige des Initialisierungsfortschritts ist. Ganz links befindet sich die rote LED, die zu leuchten beginnt, wenn ein Fehler aufgetreten ist.

Abbildung 74: Platzierung Status LEDs

Versorgungsanschlüsse

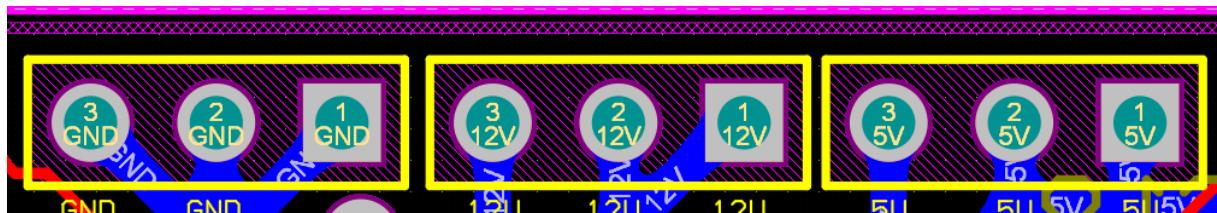


Abbildung 75: Platzierung Versorgungsanschlüsse 1

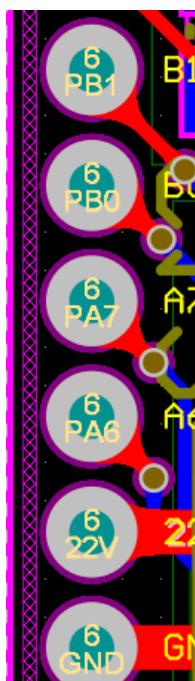
Die 12V und 5V, sowie die Masseanschlüsse befinden sich oben am Rand der Platine. Zur Verfügung stehen drei externe Masseverbindungen, sowie drei externe 12V und drei externe 5V Anschlüsse.



Auf der rechten Seite der Platine befinden sich zusätzlich noch drei externe 3,3V Anschlüsse.

Abbildung 76: Platzierung Versorgungsanschlüsse 2

ESC-Connections



Die Verbindung zum ESC (Electronic Speed Controller) wurde ganz links platziert, da das der Vorderseite der Drohne entspricht und sich dort der Stecker des ESC befindet. Ganz unten befindet sich die Masseverbindung zwischen ESC und Flight Controller Platine. Darüber befindet sich die Versorgungsspannung, die direkt vom Akku auf der Drohne kommt. Die vier Anschlüsse darüber bilden die Signalleitungen für die Ansteuerung der Motoren. Dabei ist Port PA6 die Signalleitung für Motor 1, Port PA7 für Motor 2, Port PB0 für Motor 3 und Port PB1 für Motor 4.

Abbildung 77: Platzierung ESC-Verbindungen

IMU-Connections

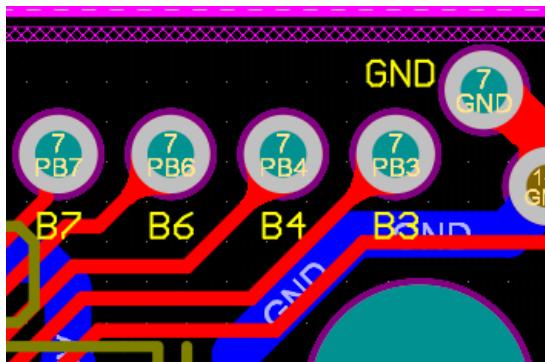


Abbildung 78: Platzierung IMU-Verbindungen

Die IMU-Connections wurden oben am Rand der Platine platziert. Ganz rechts befindet sich die Masseverbindung zwischen Flight Controller und IMU. Links daneben befindet sich der Mikrocontroller Port PB3, welcher mit dem F_SYNC-Port des IMU verbunden wird. Der Port PB4 ist für den Interrupt Port des IMU. Die Ports PB6 und PB7 sind für die I2C Datenübertragung zwischen Flight Controller und IMU, wobei PB6 die Takteleitung und PB7 die Datenleitung ist. Die vom IMU benötigte Versorgungsspannung von 5V wird von einem 5V Versorgungsanschluss genommen.

VTx-Connections



Abbildung 79: Platzierung VTx-Verbindungen

Die VTx (Videotransmitter) Verbindungen wurden auf der rechten Seite der Platine platziert, da sich hier die Rückseite der Drohne befindet und die VTx in der Rückseite verbaut ist. Der Mikrocontroller Port PB10 wird für die Datenübertragung von Flight Controller zu VTx verwendet. Darüber befindet sich die Masseverbindung. Versorgt wird die VTx direkt von der Akkuspannung, da dies im Datenblatt so empfohlen ist.

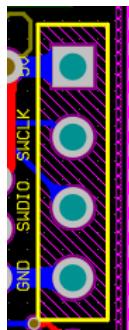
Receiver-Connections



Die Receiver Verbindungen wurden auf der rechten Seite der Platine platziert, da sich hier das Heck der Drohne mit dem Receiver befindet. Die 5V bilden die Versorgungsspannung des Receivers. Direkt darunter befindet sich die Masseverbindung zwischen Flight Controller und Receiver. Der Port PC6 wird zum Einlesen des PPM Signals des Receivers verwendet. Am Port PB14 wird das S-Bus/I-Bus Signal des Receivers eingelesen.

Abbildung 80: Platzierung Receiver-Verbindungen

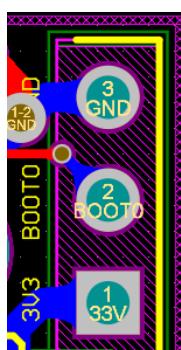
Programmierschnittstelle



Die Stifteleiste für die Programmierung wurde an der linken Seite der Platine platziert. Ganz oben befindet sich der 5V Anschluss für den 5V Betrieb. Darunter befindet sich der SWCLK Anschluss für die externe Taktleitung für die Programmierung. Darunter befindet sich der SWDIO Anschluss, wobei es sich um die Datenübertragung der Programmierung handelt. Ganz unten wird Masse verbunden.

Abbildung 81: Platzierung Programmierschnittstelle

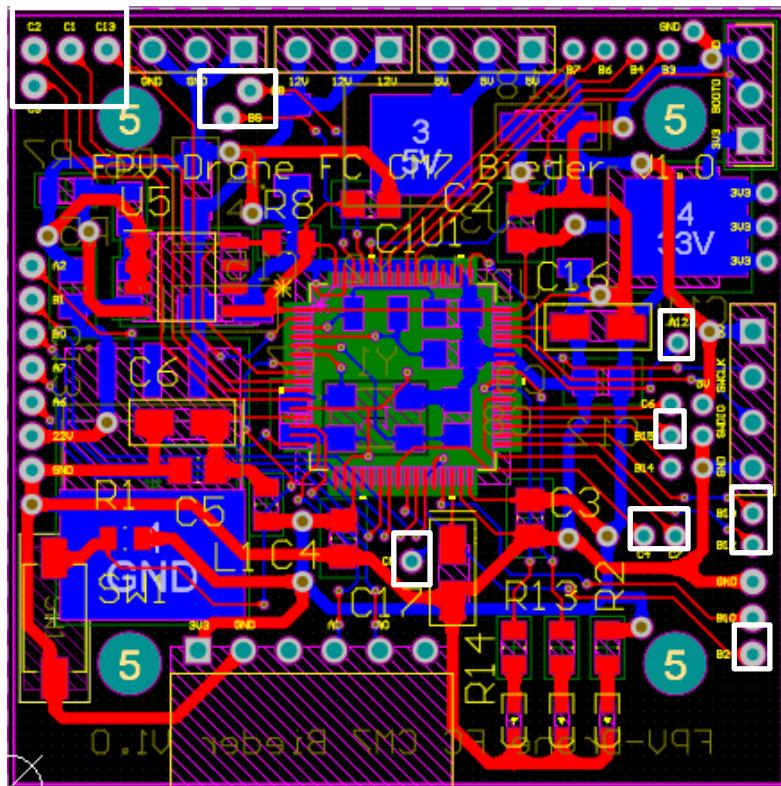
Boot Loader



Die Stifteleiste für den Bootloader wurde in der oberen, rechten Ecke platziert. In der Mitte befindet sich der eigentliche Boot Pin. Darunter befindet sich ein 3,3V Pin und darüber ein Masse Pin. Um den Bootvorgang zu ändern, kann der Boot Pin mittels eines Jumpers zu einem anderen Pin verbunden werden.

Abbildung 82: Platzierung Bootloader

Freie Pins



Zusätzlich wurden ein paar freie Pins (weiß markiert) des Mikrocontrollers rausgeführt, um für den späteren Gebrauch Optionen offen zu halten.

Abbildung 83: Platzierung freie Pins

4.3.5 Pinbelegung

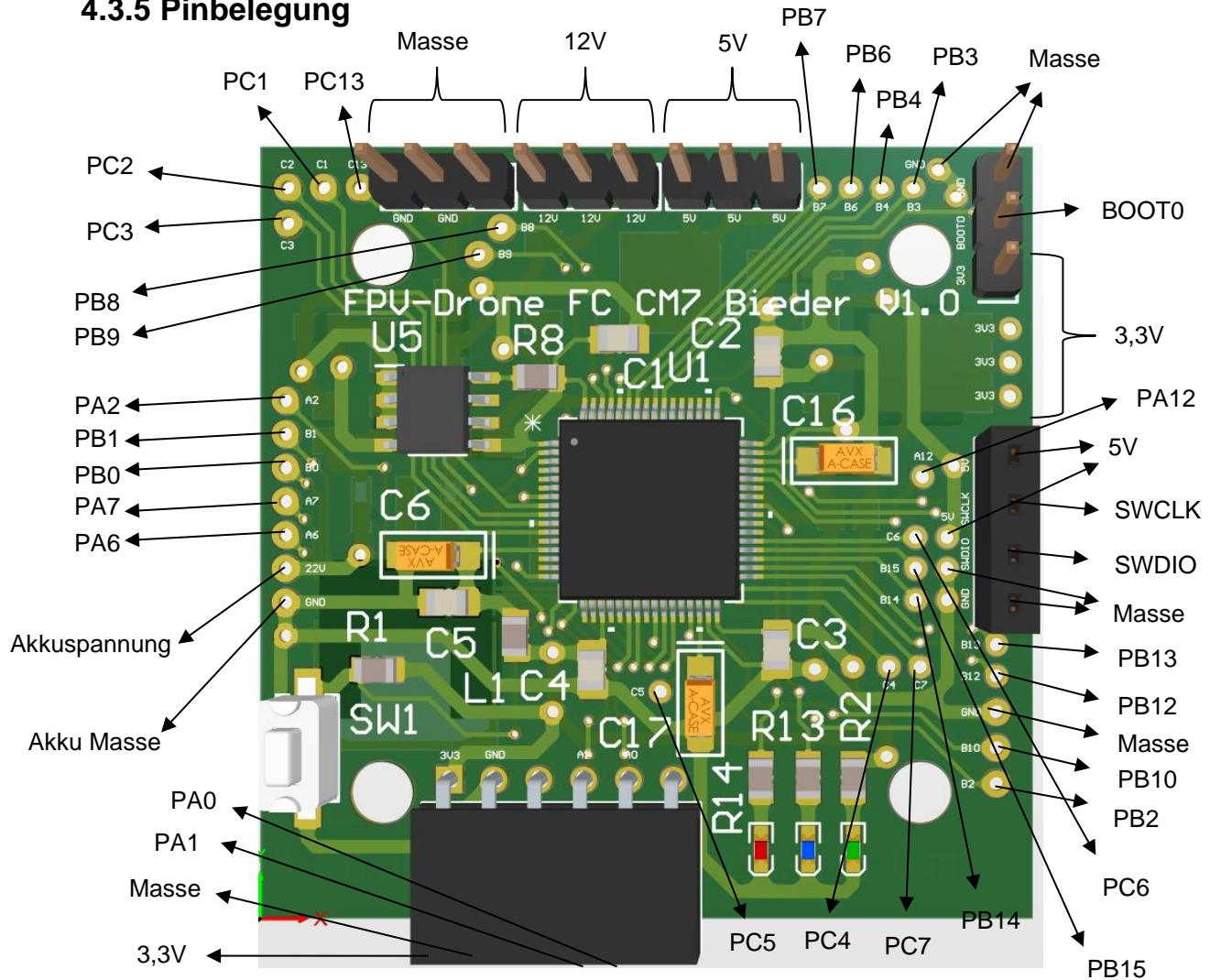


Abbildung 84: Pinbelegung Flight Controller

	PA	PB	PC
0	Terminal Tx	ESC Motor 3	/
1	Terminal Rx	ESC Motor 4	Frei
2	Frei	Frei	Frei
3	Frei	IMU FSYNC	Frei
4	/	IMU INT	Frei
5	/	/	Frei
6	ESC Motor 1	IMU I ² C SCL	Receiver PPM
7	ESC Motor 2	IMU I ² C SDA	Frei
8	/	/	/
9	/	/	/
10	/	Data Transmitter Port	/
11	/	/	/
12	/	Frei	/
13	/	Frei	Frei
14	/	Receiver S-Bus	/
15	/	Frei	/

4.3.6 Bestückungsplan

4.3.6.1 Top Layer

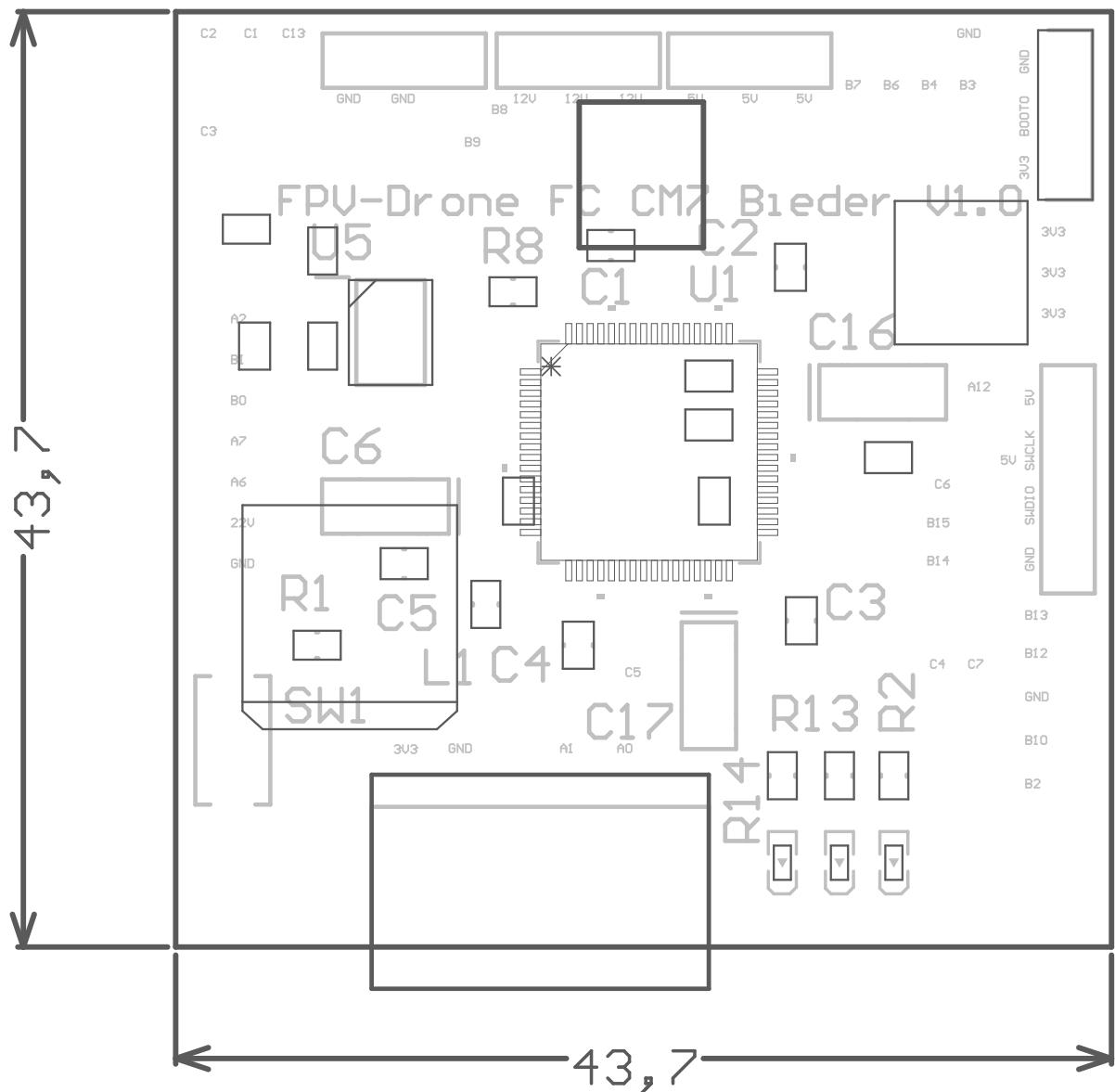


Abbildung 85: Bestückungsplan Top Layer

4.3.6.2 Bottom Layer

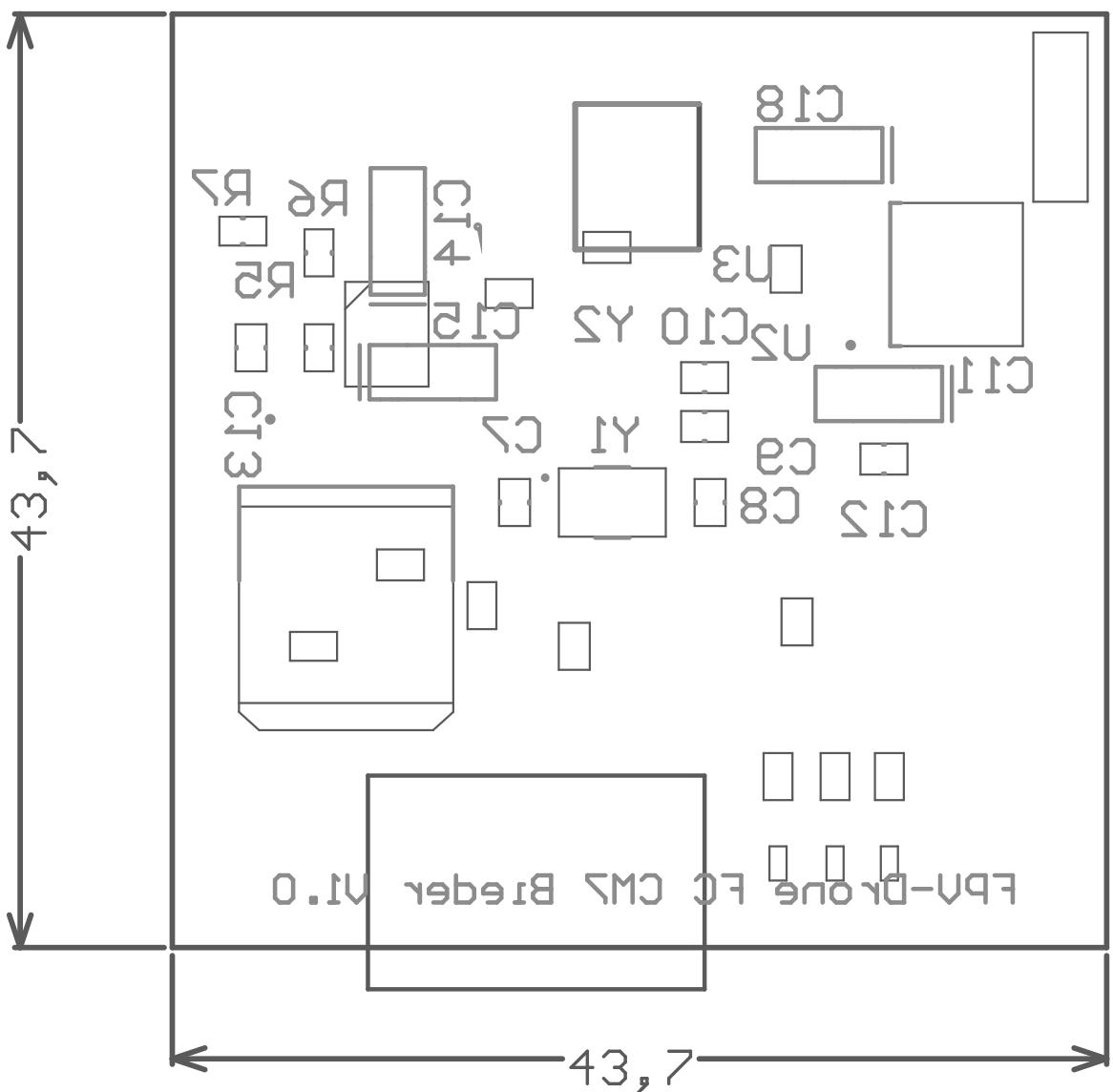


Abbildung 86: Bestückungsplan Bottom Layer

4.3.7 Betriebsmodi

4.3.7.1 5V Betrieb

Es besteht die Möglichkeit neben dem Akkubetrieb, die FPV-Drohne auch nur mit 5V zu versorgen. In diesem Modus können aber nur neue Programme auf den Mikrocontroller geflasht werden und mit den Sensoren kommuniziert werden. Die Motoren können sich in diesem Modus nicht drehen. Dieser Modus ist perfekt geeignet, um neue Programme auf ihre Funktion zu überprüfen, ohne dass sich die Motoren bewegen können. Hierbei wird kein Akku angeschlossen, sondern an der Programmierschnittstelle folgende vier Anschlüsse:

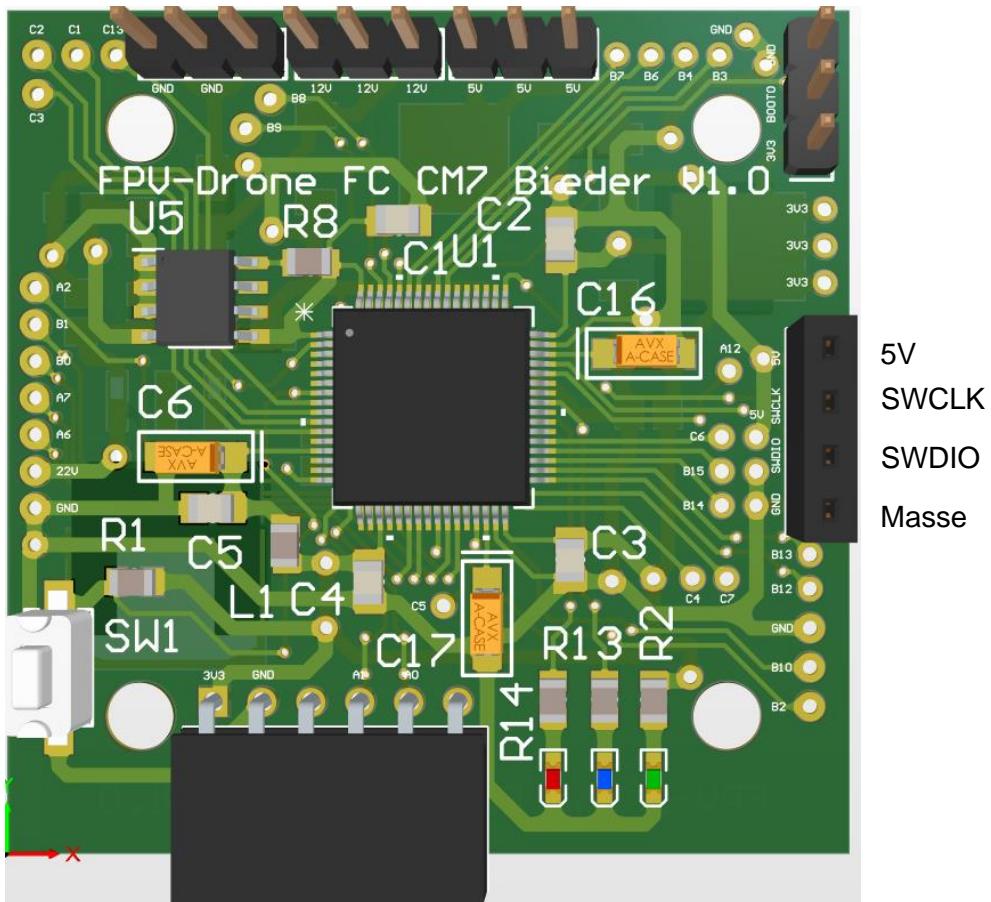


Abbildung 87: 5V Betrieb

4.3.7.2 Akkubetrieb

Im Akkubetrieb ist die FPV-Drohne voll einsatzfähig. In diesem Modus kann programmiert werden, mit Sensoren kommuniziert werden und die Motoren gedreht werden. In diesem Modus ist die Drohne flugfähig.

4.3.8 Testen

Nachdem die Platine gefertigt wurde, musste diese natürlich auch getestet werden, ob alle wesentlichen Komponenten und Übertragungen funktionieren.

4.3.8.1 Spannungswandler

Zuallererst wurde überprüft, ob alle Spannungswandler korrekt funktionieren und die richtige Spannung ausgeben. Dafür wurde zuerst die 3,3V Ebene ins Visier genommen und mit einem externen Netzteil mit Strombegrenzung versorgt. Hierbei sollte die grüne LED auf der Oberseite der Platine leuchten. Zusätzlich wurden alle 3,3V Verbindungen zu Bauteilen überprüft, ob diese korrekt ankommen.

Als nächsten Schritt hat man sich die 5V Ebene der Flight Controller Platine angesehen und überprüft ob alle Spannungen korrekt sind. Hier wurde ebenfalls ein externes Netzteil mit Strombegrenzung verwendet, auf 5V eingestellt und auf einen 5V Pin verbunden. Wichtig hierbei ist darauf zu achten, ob der 3,3V Step Down Converter die Spannung von 5V korrekt auf 3,3V senkt. Wenn dies der Fall ist, sollte die grüne LED wieder leuchten.

Als nächstes wurde die 12V Ebene auf Fehler überprüft. Hier wurde wieder ein externes Netzteil mit Strombegrenzung verwendet, auf 12V eingestellt und auf einen 12V Pin angeschlossen. Hier ist es wichtig zu überprüfen, ob der 5V Step Down Converter richtig die 12V auf 5V absenkt. Die grüne LED sollte wieder leuchten.

Als letzten Schritt wurde die Ebene der Akkuspannung überprüft, welche zwischen 11,1V und 25,2V liegen kann. Getestet wurde mit der maximalen Spannung von 25,2V mit einem externen Netzteil mit Strombegrenzung. Wichtig zu überprüfen ist, ob der 12V Step Down Converter die Akkuspannung korrekt auf die 12V senkt. In diesem Betrieb sollte die grüne LED ebenfalls leuchten.

4.3.8.2 Programmierung

Als Nächstes wurde der Flashvorgang des Programms überprüft. Um flashen zu können, wurde die Flight Controller Platine mit einer Spannung von 5V versorgt. Zusätzlich müssen die zwei Debugleitungen SWCLK und SWDIO des Controllers mit dem externen DAPLINK Modul verbunden werden. Wenn dies abgeschlossen ist, kann geflasht werden und man bekommt eine Rückmeldung der Entwicklungsumgebung ob der Flashvorgang erfolgreich war.

4.3.8.3 Terminal

Zusätzlich wurde überprüft, ob die Terminal Eingabe und Ausgabe korrekt funktioniert. Hierfür wurde der UART/USB Converter der HTL Hollabrunn angeschlossen und mit dem PC verbunden. Um die Übertragung auf ihre Funktionsfähigkeit zu überprüfen, wurde ein Testprogramm geschrieben.

4.3.8.4 Weitere Komponenten

Der letzte Schritt beinhaltet das Testen der externen Empfänger, Sensoren und Aktoren wie IMU, Smart Battery Monitor, ESC, Receiver und VTx. Um diese zu testen, wurden die entsprechenden Signale gesendet bzw. empfangen und die Werte per Terminal ausgegeben und auf ihre Richtigkeit überprüft. Hierfür wurden ebenfalls Testprogramme geschrieben.

4.4 Motoren

4.4.1 Allgemeines

Bei den Motoren fiel unsere Wahl auf den T-Motor F60PROV-LV 2020KV, da diese eine enorme Leistung und hohe Geschwindigkeit für unsere FPV-Drohne bieten können. Weiters überzeugen diese Motoren mit ihrem stabilen und ansprechendem Aluminiumdesign.



Abbildung 88: T-Motor F60PROV-LV [TMF60]

4.4.2 Aufbau

Beim Aufbau handelt es sich um einen BLDC-Motor (Brushless Direct Current Motor). Bei bürstenlosen Motoren müssen die Spulen innerhalb des Motors selbst angesteuert werden. Bei diesem Motor handelt es sich um einen 3-phasigen Motor, was bedeutet, dass dieser drei Spulen verbaut hat. Die Ansteuerung der drei Phasen werden zusammen mit dem Electronic Speed Controller (ESC) (siehe: [Kapitel 4.5](#)) erläutert.

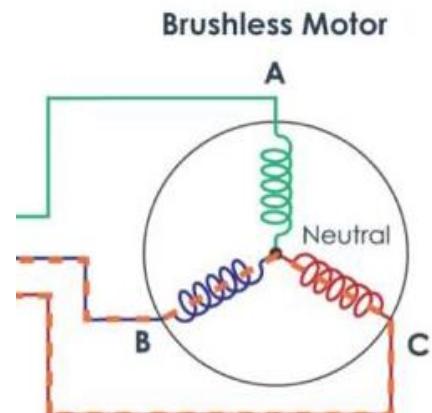


Abbildung 89: Motor Innenaufbau [BMIA]

4.4.3 Mechanische Daten

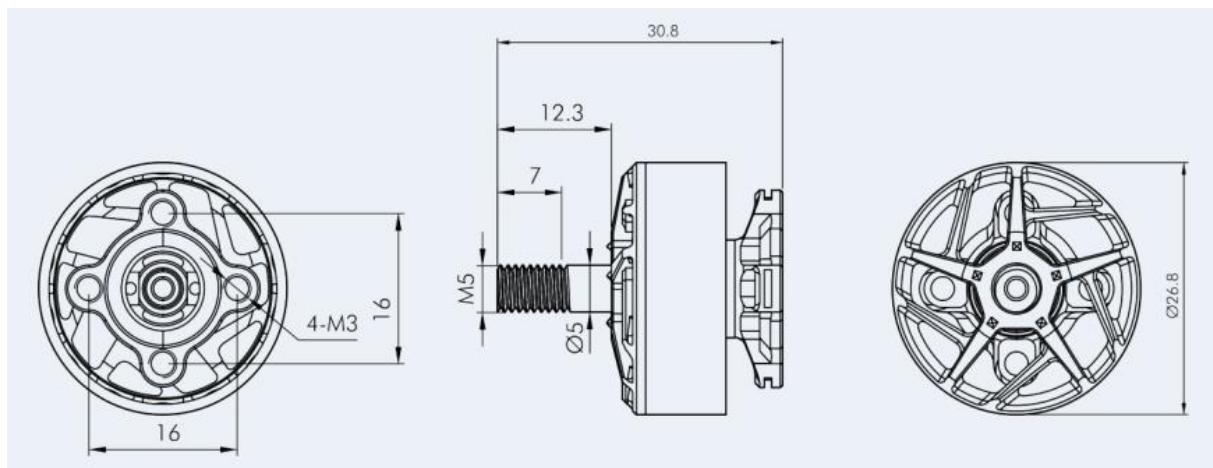


Abbildung 90: Mechanische Daten Motor [F60MD]

Alle Maße sind in mm angegeben.
Das Gewicht des Motors beträgt 32,1g.

4.4.4 Elektrische Daten

Nennspannung	25,2V (6s)
Eingangsspannung	11,1V – 25,2V (3s – 6s)
Leerlaufstrom (10V)	1,19A
Maximaler Strom (10s)	53,1A
Maximale Leistung (10s)	1306W

4.4.5 Testdaten im Betrieb

Gaspedal drücken	Spannung (V)	Strom (A)	Drehzahl (U/min)	Zugkraft (g)	Leistung (W)	Kraftwirkung (g/W)
20%	25.2	2.7	12034	282.2	68.3	4.13
40%	25.1	9.6	18423	696.5	241.6	2.88
60%	25.0	20.3	22649	1076.1	508.5	2.12
80%	24.8	34.0	27834	1669.3	844.4	1.98
100%	24.6	53.1	31409	2083.5	1306.1	1.60

Abbildung 91: Motortestdaten im Betrieb [F60MD]

4.4.6 Zugkraft des Motors

Im folgenden Diagramm kann man die Zugkraft über das gedrückte Gaspedal erkennen. Die Zugkraft wurde mit verschiedenen Propellern getestet, wie man anhand der verschiedenen farbigen Kennlinien im Diagramm erkennen kann. Wir haben uns hierbei für die GF51466-3 entschieden.

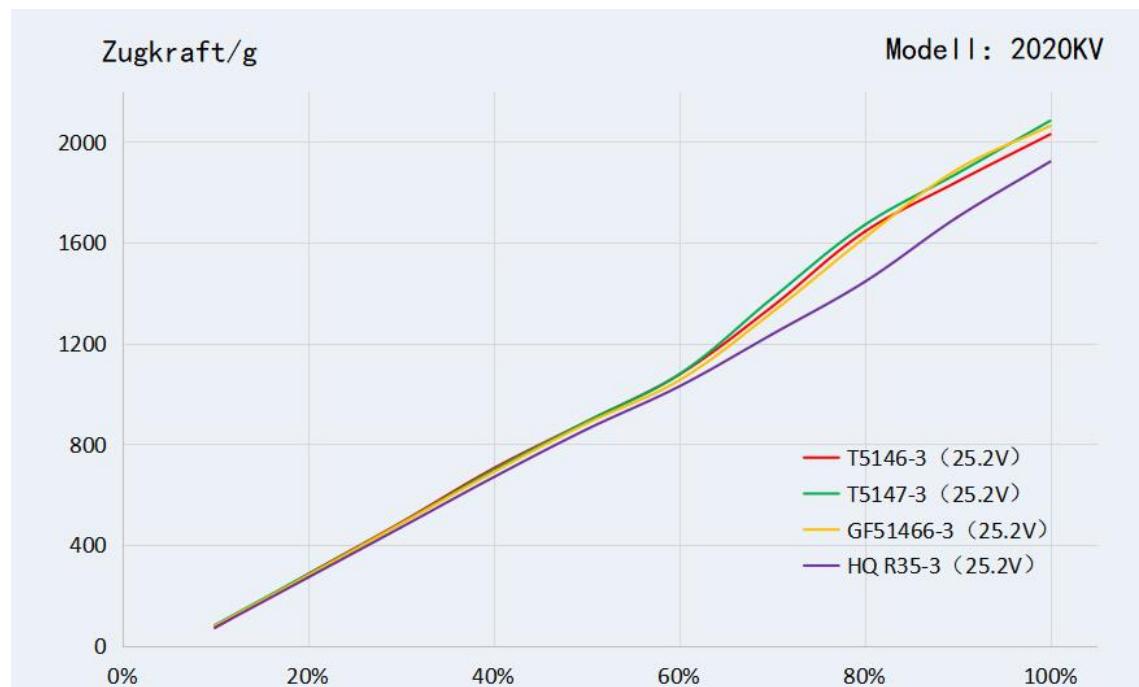


Abbildung 92: Zugkraft des Motors [F60MD]

4.5 Electronic Speed Controller (ESC)

4.5.1 Allgemeines

Der Electronic Speed Controller (ESC) ist zuständig für die Ansteuerung der Motoren (siehe: [Kapitel 4.4](#)) auf der Drohne. Je nachdem, welche Daten der ESC vom Flight Controller geliefert bekommt, steuert dieser die Motoren schneller oder langsamer an. Hierbei haben wir uns für die T Motor V45A LITE 6S entschieden, da diese genug Leistung bereitstellt, um unsere leistungsstarken Motoren anzusteuern. Ein weiterer Vorteil von diesem ESC ist, dass dieser ein 4in1 ESC ist, was bedeutet, dass alle vier Motoren angeschlossen werden können und nicht nur einer.

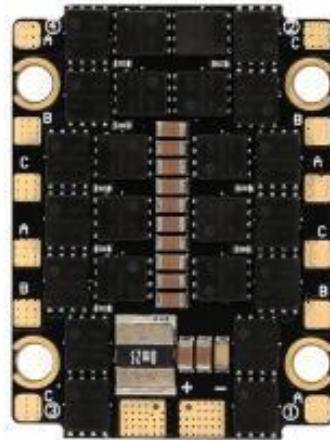


Abbildung 93: T Motor V45A LITE [V24AD]

4.5.2 Aufbau

Der ESC besitzt drei Halbbrücken pro Motor, eine Halbbrücke pro Phase, welche dazu verwendet werden, um die Spulen des Motors anzusteuern. Eine Halbbrücke besteht aus meist zwei MOSFETs, wobei einer gegen VCC (High-Side Switch) und der andere gegen Masse (Low-Side Switch) geschalten ist. In der Mitte dieser zwei MOSFETs befindet sich die Last, wie in unserem Fall der Motor.

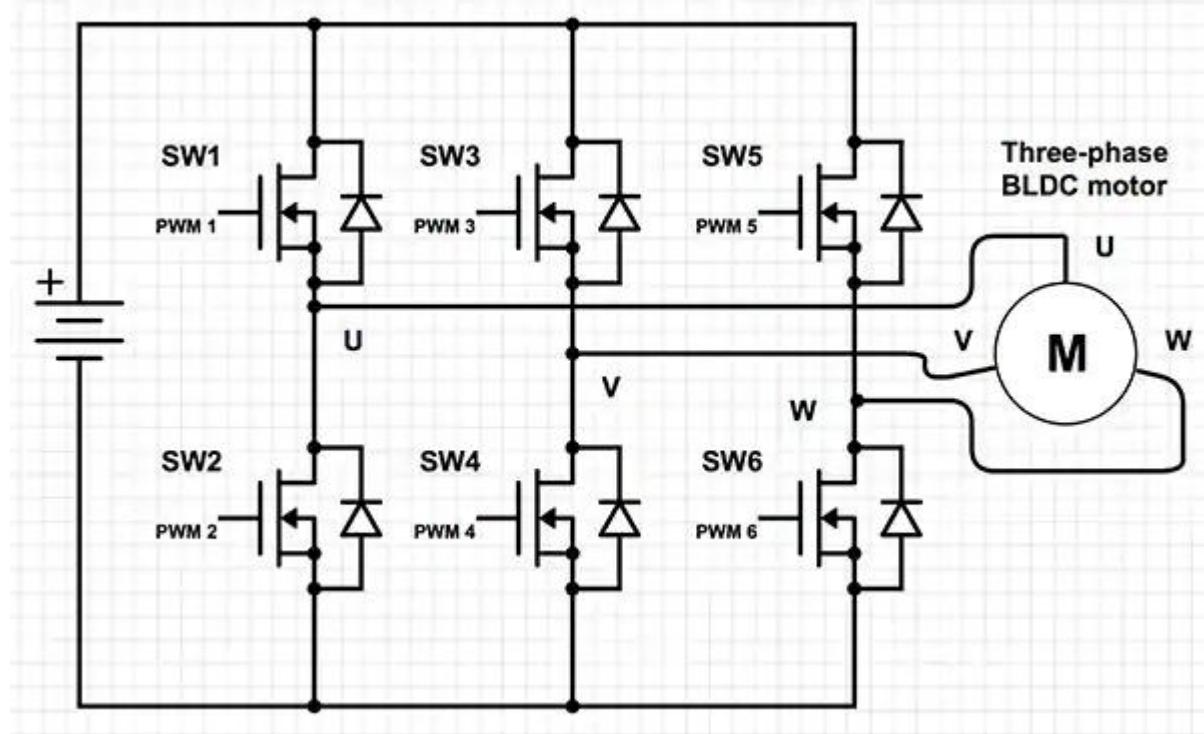


Abbildung 94: ESC Aufbau [ESCAB]

4.5.3 Funktionsweise

Um nun den Motor zum Drehen zu bringen, werden nacheinander immer zwei MOSFETs von verschiedenen Phasen angesteuert:

Schritt	VCC-Verbindung	Masse Verbindung	Angesteuerte Spulen
1	SW1	SW6	UW
2	SW3	SW6	VW
3	SW3	SW2	VU
4	SW5	SW2	WU
5	SW5	SW4	WV
6	SW1	SW4	UV

Wenn man diese Schritte nun immer wieder schnell nacheinander ausführt, dreht sich der Motor. Um die Leistung des Motors regeln zu können, werden die einzelnen MOSFETs mittels PWM-Signalen angesteuert. Um die Geschwindigkeit der Motoren zu regeln, wird die Schrittfolge einfach schneller oder langsamer abgearbeitet. Angesteuert werden die MOSFETs mittels Mikrocontrollern auf der Rückseite der Platine. Damit diese Mikrocontroller die MOSFETs mit der richtigen Geschwindigkeit ansteuern, bekommen diese für jeden Motor Daten mithilfe des DShot Protokolls (siehe: [Kapitel 5.7.1](#)) vom Flight Controller (siehe: [Kapitel 4.3](#)).

4.5.4 Elektrische Daten

Eingangsspannung	11,1 – 25,2V (3s – 6s)
Strom	45A (pro Motor)
Maximaler Strom (10s)	55A (pro Motor)
Unterstützte Protokolle	DShot150, DShot300, DShot600

4.5.5 Pinbelegung

4.5.5.1 Top Layer

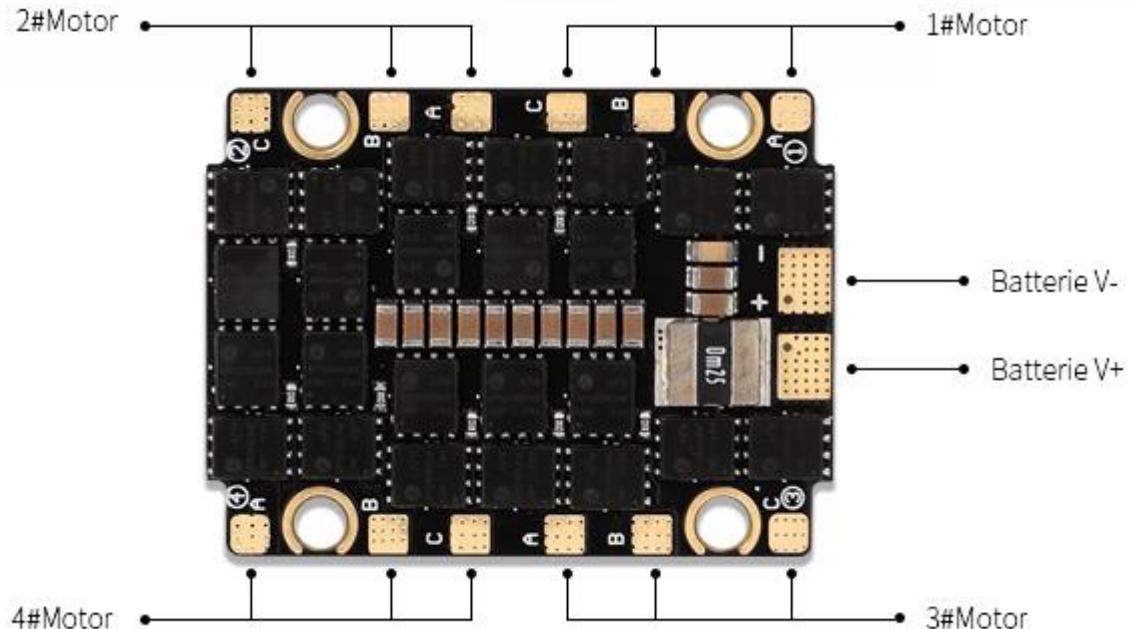


Abbildung 95: ESC Pinbelegung Top Layer [V24AD]

4.5.5.2 Bottom Layer

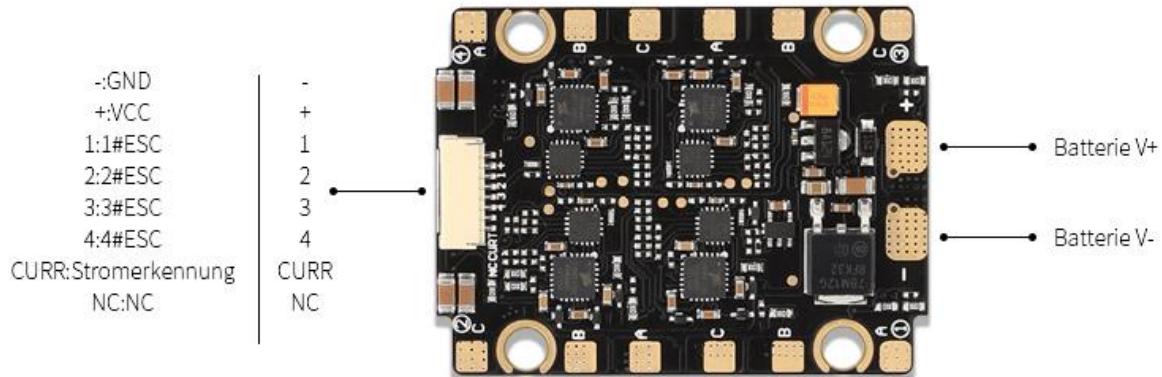


Abbildung 96: ESC Pinbelegung Bottom Layer [V24AD]

4.6 Inertial Measurement Unit (IMU)

4.6.1 Allgemeines

Die Inertial Measurement Unit (IMU) ist ein Breakoutboard mit einem Gyroskop, Beschleunigungssensor, Magnetometer und einem Barometer. Das Gyroskop, der Beschleunigungssensor und das Magnetometer befinden sich alle in einem Chip, dem MPU9250. Dieser wird verwendet, um die Winkelbeschleunigungen zu messen und in Lagewinkel umzurechnen, um eine Lageregelung für die Drohne zu entwickeln. Mit dem Barometer, dem AK8963C, werden der Luftdruck und die Temperatur gemessen, um sich die ungefähre Höhe der Drohne ausrechnen zu können.

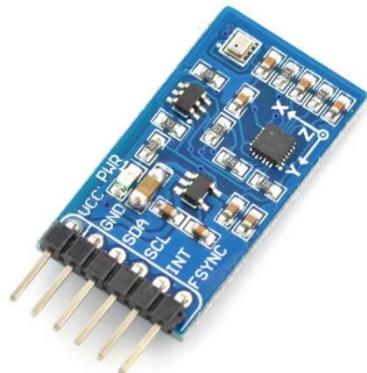


Abbildung 97: IMU Breakoutboard [IMUD]

4.6.2 Mechanische Daten

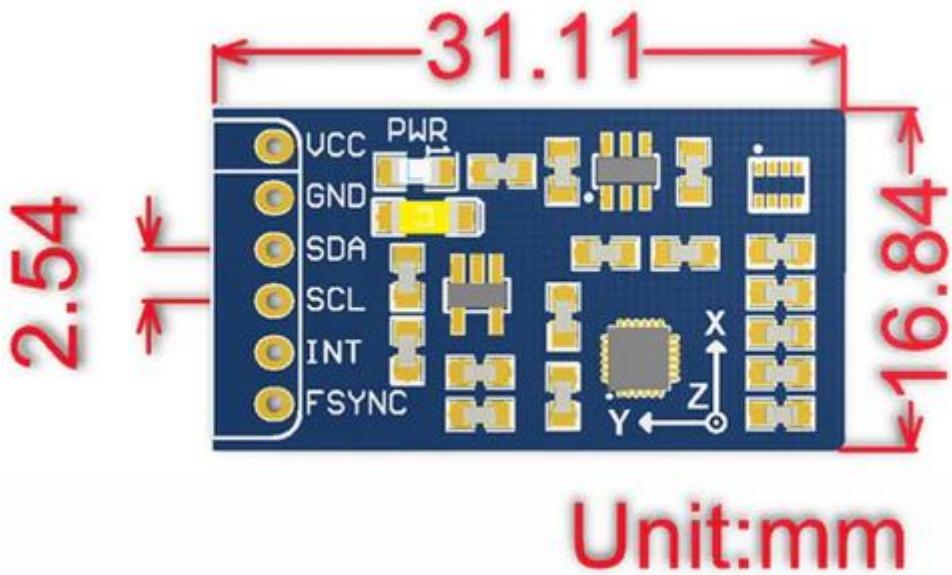


Abbildung 98: Mechanische Daten IMU [IMUD]

4.6.3 Schaltplan

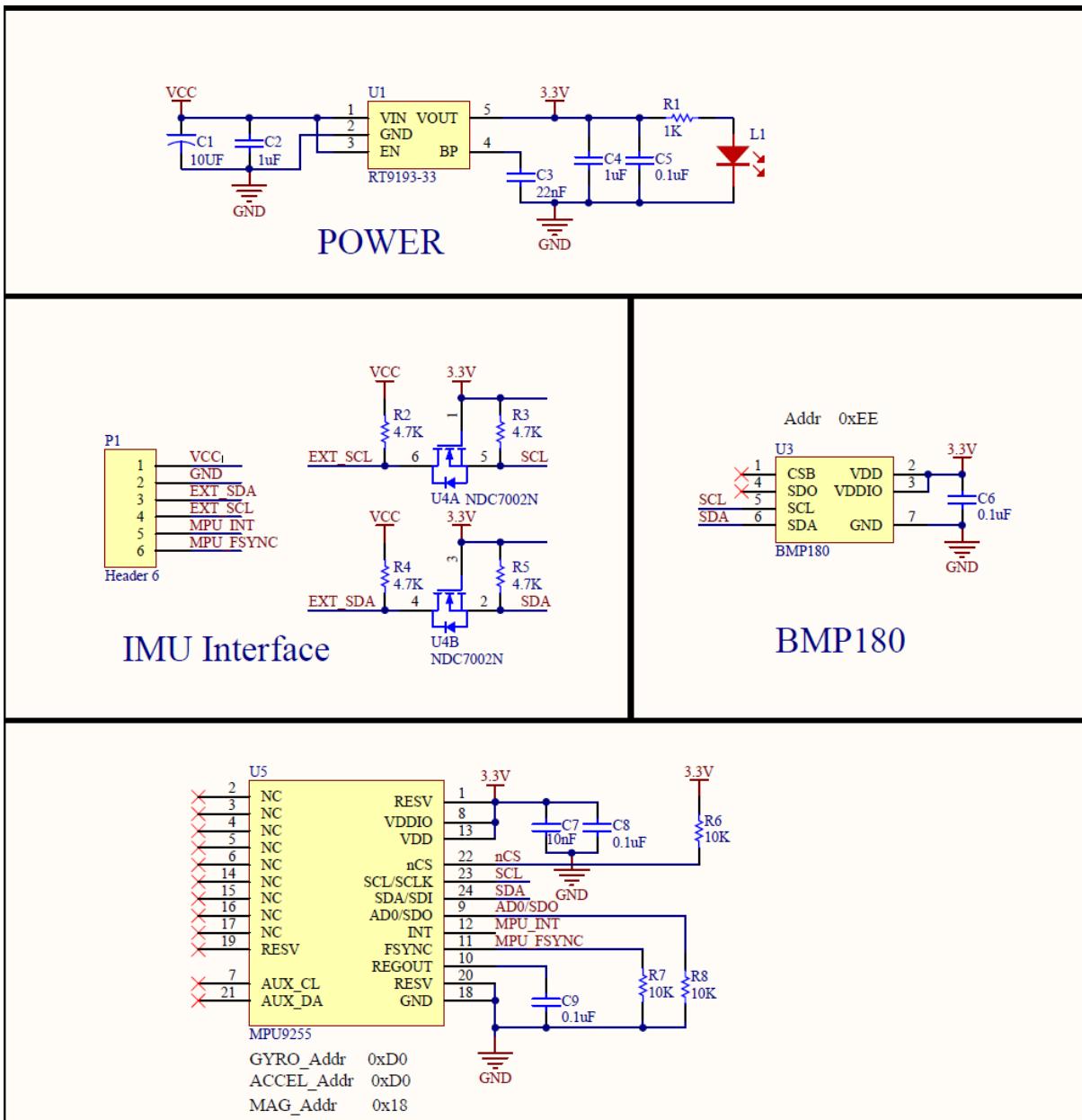


Abbildung 99: Schematic IMU

Anmerkung: Auf unserem Breakoutboard ist der BMP280 und der MPU9250 verbaut. Die Schaltung bleibt allerdings dieselbe.

4.6.3.1 Spannungsversorgung

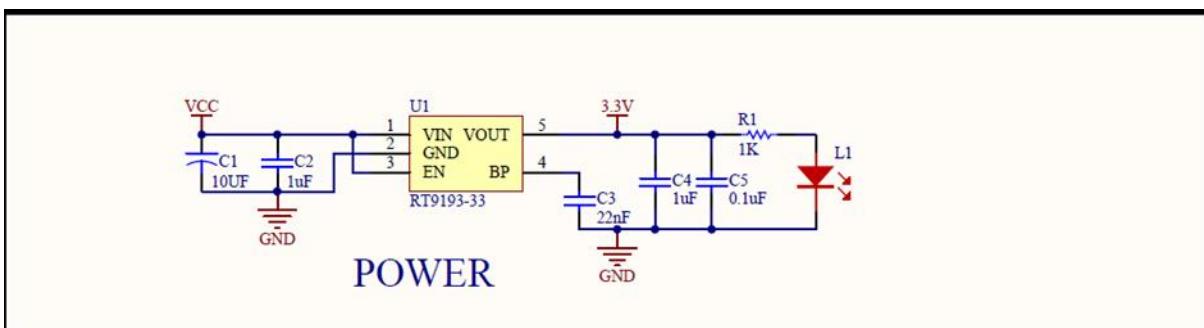


Abbildung 100: Schematic Spannungsversorgung IMU

Für die Spannungsversorgung der IMU-Platine kommt der Fixspannungsregler RT9193-33 zum Einsatz, welcher die Eingangsspannung auf 3,3V runterregeln soll, um die Sensoren versorgen zu können. Die Eingangsspannung wird dabei auf VIN (Pin 1) des Spannungsreglers angeschlossen. An GND (Pin 2) wird Masse angeschlossen. Zwischen VIN und GND werden zusätzlich noch die Kondensatoren C1 und C2 geschalten, die als Stützkondensatoren dienen, um die Welligkeit in der Versorgungsspannung zu verringern. Der EN-Pin (Pin 3) stellt den Enable-Pin dar, welcher verwendet wird, um den Spannungsregler ein oder auszuschalten. Dieser liegt dauerhaft auf einem High-Pegel, weshalb der Spannungsregler dauerhaft in Betrieb ist. An VOUT kann man die heruntergeregelten 3,3V abgreifen. Der Kondensator C4 dient ebenfalls als Stützkondensator, um die 3,3V zu glätten. Weiters wird der Kondensator C5 als Entkoppelkondensator verwendet, um hochfrequente Störungen gegen Masse abzuleiten. Um ein visuelles Feedback zu haben, ob die Spannungsversorgung funktioniert, wird eine rote LED L1 über einen 1kOhm Vorwiderstand eingebaut, die leuchtet, wenn 3,3V anliegen. Der BP-Pin (Pin 4) des Reglers ist der Reference Noise Bypass Pin, welcher verwendet wird, um die interne Referenzspannung von Störungen zu befreien. Laut Hersteller ist es ratsam einen 22nF Kondensator zum Filtern zu verwenden, wie es hier mit dem Kondensator C3 gemacht wird.

4.6.3.2 IMU-Interface

Über die 6-polige Stiftleiste des IMUs wird die Platine versorgt und weiters kann mit den Sensoren kommuniziert werden. An Pin 1 wird die Versorgungsspannung angelegt. Über Pin 2 wird Masse verbunden. Bei Pin 3 und 4 handelt es sich um die Kommunikation zwischen Flight Controller und Sensoren mittels I²C-Bus, wobei Pin 3 die Datenleitung und Pin 4 die Taktleitung ist. An Pin 5 befindet sich der digitale Interrupt für den MPU9250. An Pin 6 kann ein Synchronisationssignal an den MPU9250 gesendet werden. Die zwei Schaltungen mit jeweils einem N-Kanal MOSFET und

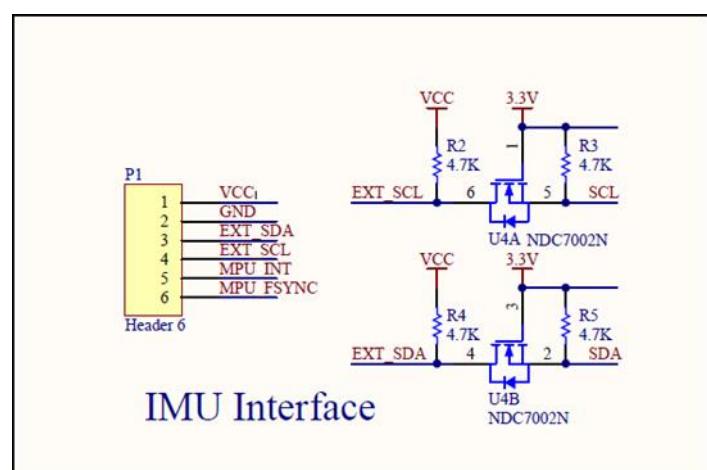


Abbildung 101: Schematic Interface IMU

zwei Widerständen hat die Aufgabe, die Pegel der I²C-Datenleitung und I²C-Taktleitung auf 3,3V anzupassen. Somit ist es egal, ob der externe Controller mit 3,3V oder 5V arbeitet.

4.6.3.3 BMP280

Der BMP280 ist ein digitaler Luftdruck- und Temperatursensor, der verwendet wird, um die aktuelle Höhe der Drohne zu bestimmen. Versorgt wird dieser durch 3,3V am VDD Pin (Pin 2). Der VDDIO Pin ist die separate Versorgung der I/O-Einheit des Barometers. Am GND Pin (Pin 7) wird Masse verbunden. Zwischen den 3,3V Versorgung und Masse wurde der Kondensator C6 eingebaut. Dieser dient als Entkoppelkondensator um hochfrequente Störungen aus der Versorgung gegen Masse abzuleiten. Außerdem kann dieser zusätzlichen Strom liefern, wenn der BMP280 mehr braucht. Der SDA Pin ist für die Datenübertragung und der SCL Pin für den Takt der I²C Kommunikation. Die zwei übrigen Pins (Pin 1 und 4) werden nicht benötigt, da diese für eine SPI Kommunikation ausgelegt sind.

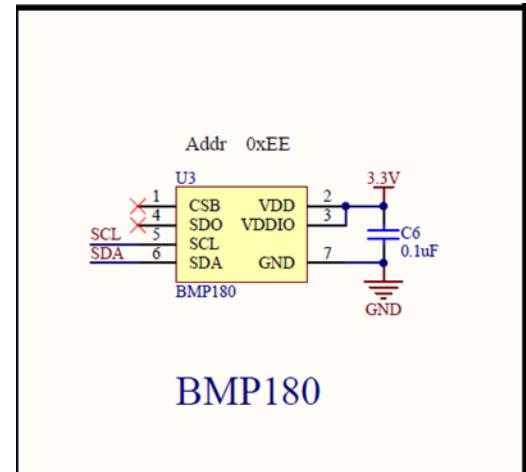


Abbildung 102: Schematic BMP280 IMU

Elektrische Daten BMP280

Eingangsspannung	1,71V – 3,6V
Interface Eingangsspannung	1,2V – 3,6V
Stromverbrauch	2,7µA
Arbeitstemperaturbereich	-40°C - +85°C
Luftdruckmessbereich	300 – 1100hPa
Relative Genauigkeit	+/- 0.12hPa
Absolute Genauigkeit	+/- 1hPa

4.6.3.4 MPU9250

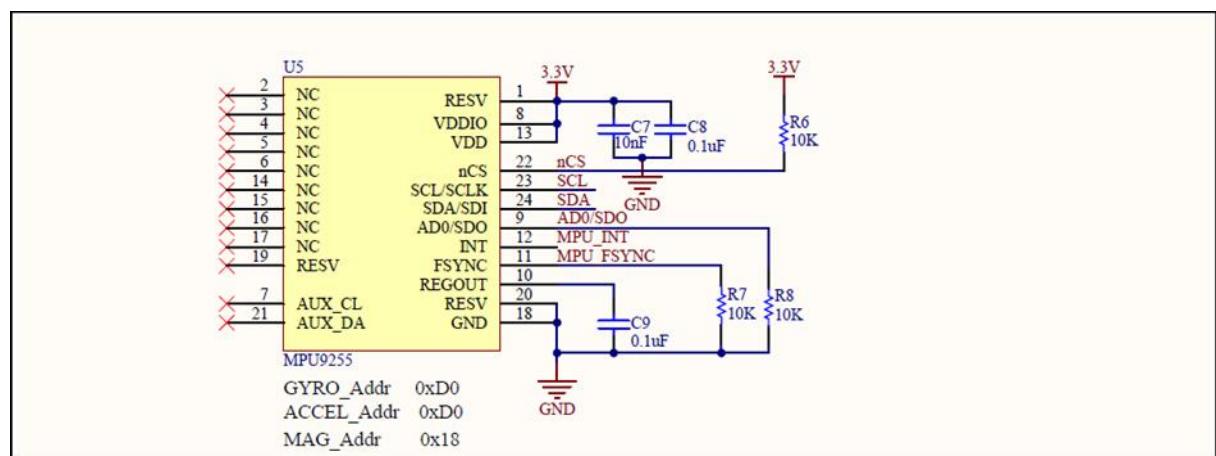


Abbildung 103: Schematic MPU9250 IMU

Der MPU9250 ist ein, aus einem Gyroskop, Beschleunigungssensor und Magnetometer bestehender IC. Dieser wird verwendet, um die Winkelbeschleunigung zu messen und damit

eine Lageregelung für die Drohne zu realisieren. Versorgt wird der MPU9250 durch die 3,3V Spannungsversorgung am VDD-Pin (Pin 13). Die I/O Einheiten werden ebenfalls durch 3,3V am VDDIO-Pin (Pin 8) versorgt. Bei Pin 1 handelt es sich um einen Reserved Pin, welcher laut Datenblatt bei der Versorgungsspannung angeschlossen werden soll. Der nCS Pin ist der Chip Select Pin, welcher aber nur bei der SPI Kommunikation notwendig ist, jedoch wird die I²C Kommunikation verwendet. Bei dem SCL Pin (Pin 23) handelt es sich um die Taktverbindung und bei dem SDA Pin (Pin 24) um die Datenverbindung für die I²C Kommunikation. Der AD0 Pin (Pin 9) wird verwendet, um das LSB der I²C Slave Adresse zu ändern. In diesem Fall wird der AD0 Pin über einen 10kOhm Widerstand auf Masse geschalten. Der INT Pin (Pin 12) ist der digitale Interrupt des MPU9250 und ist mit der 6-poligen Stifteleiste verbunden. An den FSYNC Pin (Pin 11), welcher mit einem Pull-Down Widerstand gegen Masse geschalten ist, um kein ungewolltes Antennenverhalten zu erzeugen, wenn dieser nicht angeschlossen ist, kann ein digitales Synchronisationssignal gesendet werden. Der Regout Pin (Pin 10) ist für einen Entkopplungskondensator (Kondensator C9) gedacht, welcher interne Störfrequenzen gegen Masse ableiten soll. An GND (Pin 18) wird Masse angeschlossen. Bei Pin 20 handelt es sich um einen weiteren Reserved Pin, welcher laut Datenblatt, gegen Masse geschalten werden soll. Die Pins AUX_CL (Pin 7) und AUX_DA (Pin 21) können für eine I²C Kommunikation verwendet werden, um mehrere Sensoren anzuschließen. Bei Pin 19 handelt es sich um einen weiteren Reserved Pin, welcher laut Datenblatt nicht verbunden werden soll. Die restlichen Pins werden nicht verbunden, da diese nicht benutzt werden.

Elektrische Daten MPU9250

Eingangsspannung	2,4V – 3,6V
Interface Eingangsspannung	1,71V - VDD
Stromverbrauch (Gyroskop)	2mA
Stromverbrauch (Beschleunigungssensor)	450µA
Stromverbrauch (Magnetometer)	280µA
Arbeitstemperaturbereich	-40°C - +85°C
Einstellbereich (Gyroskop)	+/- 250, +/- 500, +/- 1000, +/- 2000 °/s
Einstellbereich (Beschleunigungssensor)	+/- 2g, +/- 4g, +/- 8g, +/- 16g
Bereich (Magnetometer)	+/- 4800µT
Auflösung (Gyroskop)	16 Bit
Auflösung (Beschleunigungssensor)	16 Bit
Auflösung (Magnetometer)	14 oder 16 Bit

4.6.4 Pinbelegung

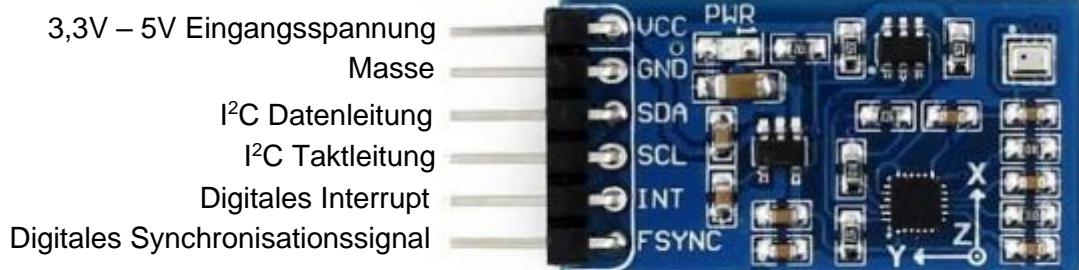


Abbildung 104: Pinbelegung IMU [IMUD]

4.7 Videotransmitter (VTx)

4.7.1 Allgemeines

Bei dem Videotransmitter für das Live Video haben wir uns für die Blacksheep TBS UNIFY PRO HV 5,8GHz entschieden. Diese bietet eine hohe Datenrate für unsere Liveübertragung und hohe Sendeleistungen für eine hohe Reichweite.



Abbildung 105: Blacksheep TBS UNIFY PRO HV [VTXD]

4.7.2 Funktionsweise

Die VTx wird verwendet, um das Livebild der Livekamera und Daten des Flight Controllers an die Groundstation und die FPV-Brille zu senden. Versorgt wird die VTx durch eine Eingangsspannung im Bereich von 7V bis 26V (2S – 6S Akku). Die VTx liest die analogen Signale der Livekamera ein und sendet diese über eine Funkübertragung mit 5,8GHz direkt an die FPV-Brille. Außerdem liest sie die per Flight Controller gesendeten Daten ein und sendet diese mit. Die Daten werden von der Groundstation empfangen und zusammen mit dem Livebild in der Smartphone Applikation angezeigt.

4.7.3 Elektrische Daten

Eingangsspannung	7V – 26V (2S – 6S)
Sendeleistung	25mW, 200mW, 500mW, 800mW
Stromverbrauch (je nach Leistung)	250mA, 320mA, 460mA, 600mA
Reichweite	Bis zu 4km
Kanäle	Band A(8), B(8), E(5), Fatshark(8), Race Band(8)
Kamera Versorgung	5V / 500mA
Zusätzliche Protokolle	Smart Audio 2.0

4.7.4 Mechanische Daten

Anschlussstecker	JST-GH, 7 Pin
Antennenanschluss	SMA
Gewicht	5g ohne Antenne

4.7.5 Pinbelegung

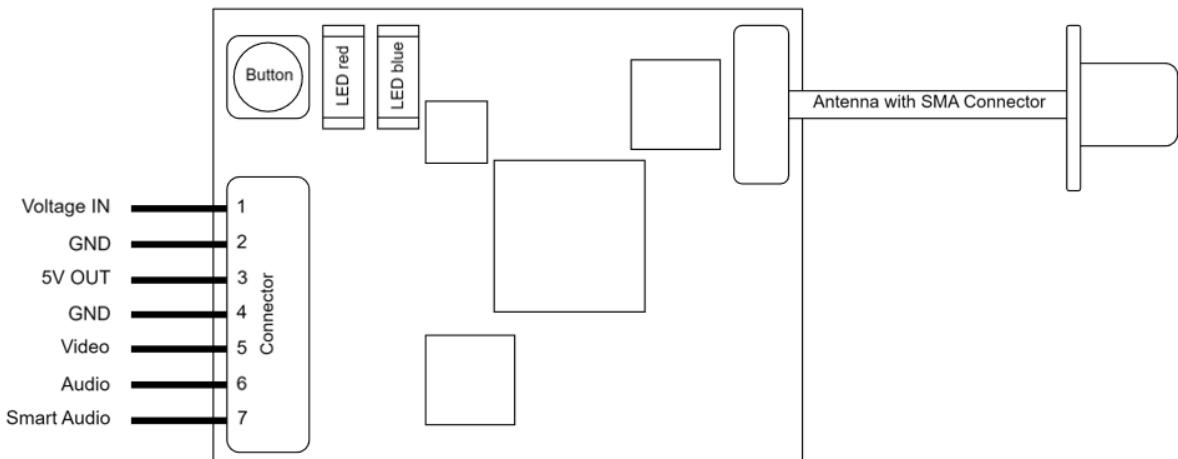


Abbildung 106: VTx Pinbelegung

An Pin 1 des JST-GH, 7 Pin Connectors wird die Versorgungsspannung zwischen 7V – 26V verbunden. Es wird empfohlen, die Spannungsversorgung der VTx direkt an die Akkuspannung anzuschließen. Daneben an Pin 2 wird die Masse der VTx mit der Masse des Flight Controllers verbunden. Bei Pin 3 handelt es sich um den 5V Output Pin, der als Spannungsversorgung für die Livekamera dient. Bei Pin 4 wird der Masseanschluss der Livekamera mit der Masse der VTx verbunden. Das analoge Videosignal, das die Livekamera sendet, wird über den Pin 5 eingelesen. Bei Pin 6 handelt es sich um den eigentlichen Audioanschluss. Da wir aber kein Audio senden, wird dieser Pin zur Datenübertragung verwendet und mit dem Data Transmission Port des Flight Controllers verbunden. Bei Pin 7 kann ein Smart Audio Protokoll verwendet werden, wobei es sich um eine UART-Schnittstelle zur Datenübertragung handelt.

4.8 Livekamera

4.8.1 Allgemeines

Bei der Livekamera haben wir uns für die Caddx Ratel 2 entschieden, wobei es sich um eine analoge FPV (First Person View) - Kamera handelt. Diese sitzt an der Front der Drohne und soll wirken, als würde man live mitfliegen, wenn man die FPV-Brille aufsetzt.



Abbildung 107: Caddx Ratel 2 [RATE2]

4.8.2 Funktionsweise

Die Livekamera wird mit dem 5V / 500mA Output der VTx versorgt. Zusätzlich wird die Masse der Kamera direkt an die Masse der VTx angeschlossen. Das analoge Kamerasignal wird direkt auf den Videoeingang der VTx verbunden. Hierbei ist aber wichtig, dass man die Leitung möglichst kurz macht und nicht an störreichen Komponenten wie Akku, ESC oder Flight Controller vorbeiführt. Dies kann nämlich zu Verzerrungen des Bildes führen.

4.8.3 Elektrische Daten

Eingangsspannung	4,5V – 36V
Sensor	1/1.8 Inch Starlight Sensor
Auflösung	1200TVL
Arbeitstemperaturbereich	-20°C - +60°C

4.8.4 Pinbelegung

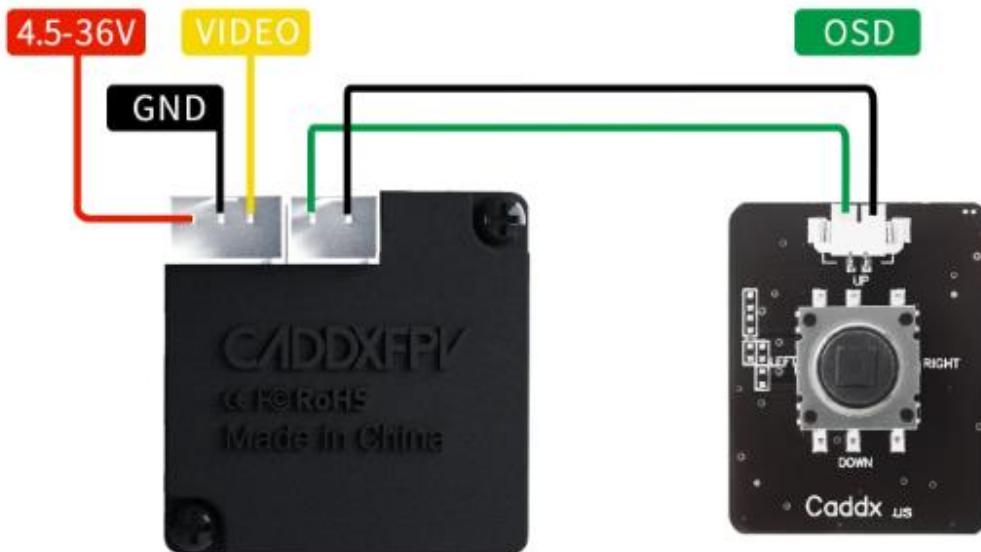


Abbildung 108: Kamera Pinbelegung [RATE2]

4.9 Receiver

4.9.1 Allgemeines

Bei dem Receiver haben wir uns für den Turnigy FS-iA6C entschieden, da dieser eine solide Reichweite bietet. Die Aufgabe des Receivers ist es, die Signale der Fernsteuerung zu empfangen und diese an den Flight Controller weiterzuleiten.



Abbildung 109: Turnigy iA6C [TIA6C]

4.9.2 Funktionsweise

Die Hauptaufgabe des Receivers ist es, die Signale der Fernsteuerung zu empfangen und in Daten zu dekodieren, die anschließend vom Flight Controller verarbeitet werden. Gesendet werden die Signale der Fernsteuerung durch eine Funkverbindung mit dem Receiver auf einer Frequenz von 2,4GHz, auf acht unterschiedlichen Frequenzkanälen. Der Receiver wandelt die acht empfangenen parallelen Datenleitungen dann auf eine serielle Leitung um, da dies leichter ist mit unserem Flight Controller einzulesen. Der Receiver bietet drei verschiedene Möglichkeiten die Daten auszulesen. Die einfachste Lösung ist, die Daten per PPM (Puls-Pausen Modulation) einzulesen, wobei es sich um ein einfaches Codierungsverfahren für analoge Werte handelt. Die zweite Möglichkeit ist, serielle Protokolle wie S-Bus oder I-Bus zu verwenden, die zwar komplexer sind, aber wesentlich schneller.

4.9.3 Elektrische Daten

Eingangsspannung	4V – 6,5V
Empfangsfrequenz	2,4GHz
Empfangsfrequenzbereich	2,408GHz – 2,475GHz
Kanäle	8
Demodulationsverfahren	GFSK

4.9.4 Pinbelegung



PPM Signal
 S-Bus / I-Bus Protokoll
 Eingangsspannung (4V – 6,5V)
 Masse

Abbildung 110: TGY iA6C Pinbelegung [TIA6C]

5 Steuerungssoftware

Die Software des Flight Controllers hat die Aufgabe, die Daten der Fernsteuerung über einen Receiver und die Messwerte der Sensoren einzulesen und diese umzuwandeln, damit diese an die ESC, VTX, Status-LEDs und Terminal senden zu können.

Die Architektur ist sehr zeitkritisch und benötigt genaue Interrupts, damit ein stabiler und sicherer Flug möglich ist. Daher wird das gesamte System von einem Real Time System Interrupt gesteuert, welcher in einem Zeitbereich von ein paar Millisekunden auslöst und mithilfe der Sensorik die gesamte FPV-Drohne steuert.

5.1 Hauptprogramm

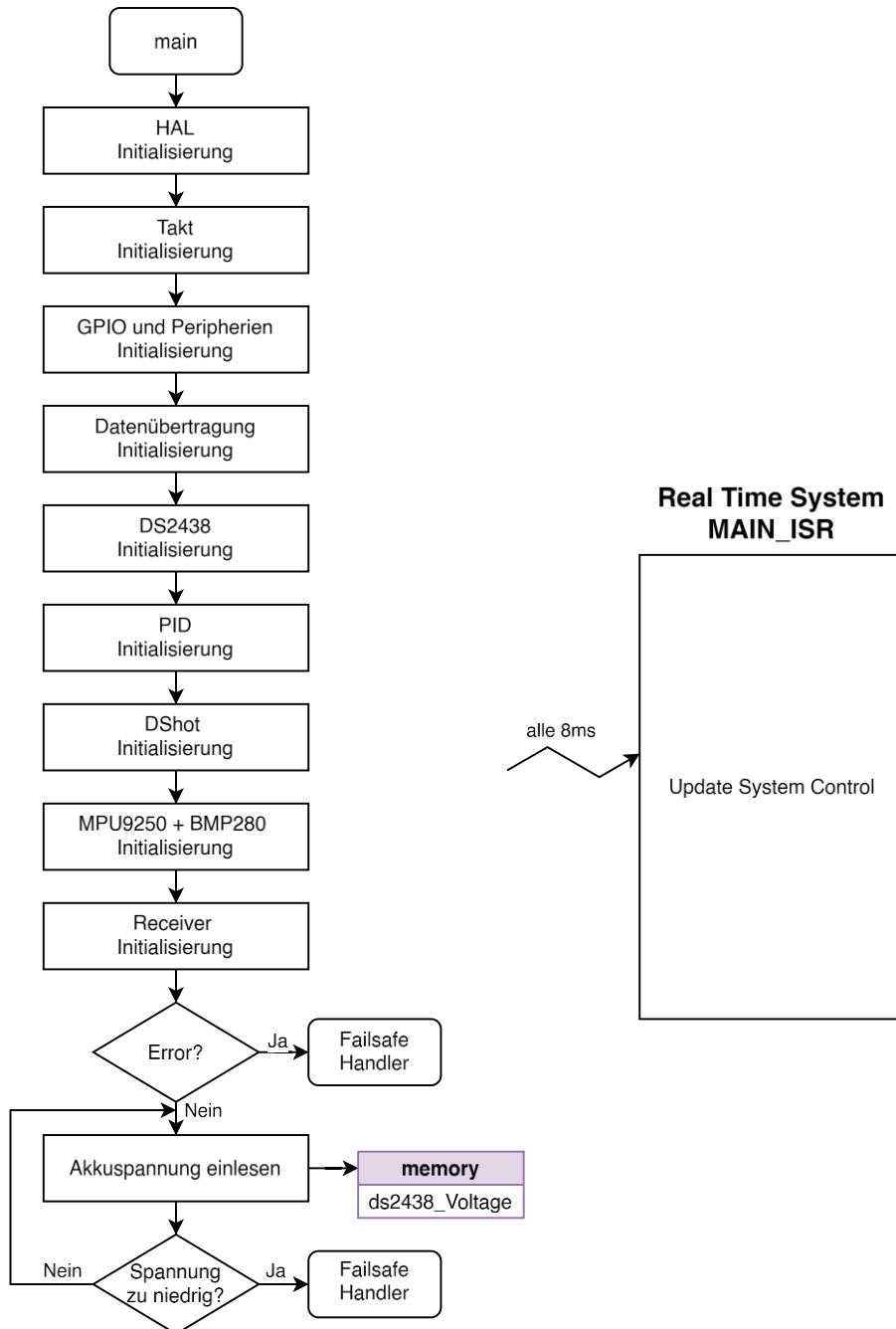


Abbildung 111: Flussdiagramm Programmablauf

Am Beginn des Hauptprogrammes (main) wird die gesamte HAL-Struktur (hardware abstract layer) initialisiert. Danach wird die Taktversorgung, GPIO-Pins und die einzelnen Peripherien mit den gewünschten Angaben aus STM32CubeMX eingestellt. Der nachfolgende Ausschnitt wird von STM32CubeMX automatisch generiert.

```
Dateiname: main.c | Ausschnitt aus main() Funktion

/**
 * @brief The application entry point.
 * @retval int
 */
int main(void)
{
    /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
    HAL_Init();

    /* Configure the system clock */
    SystemClock_Config();

    /* Initialize all configured peripherals */
    MX_GPIO_Init();
    MX_DMA_Init();
    MX_TIM3_Init();
    MX_USART1_UART_Init();
    MX_USART3_UART_Init();
    MX_I2C1_Init();
    MX_UART4_Init();
    MX_TIM4_Init();
    MX_TIM15_Init();
    MX_TIM16_Init();
    MX_TIM17_Init();
    MX_TIM14_Init();
    MX_TIM2_Init();
    MX_TIM1_Init();

    ...
}
```

Bemerkung:

Die Punkte ... bedeuten, dass sich nach dem Ausschnitt noch weiterer Programmteile befinden. Im weiteren Verlauf werden die Punkte immer für diesen Kontext verwendet.

Als nächsten Schritt werden alle Sensoren, Ein- und Ausgänge mit den angegebenen Einstellungen initialisiert.

```
Dateiname: main.c | Ausschnitt aus main() Funktion
int main(void)
{
    ...

    int8_t errorCode;
    Terminal_Print("Program start\n\r");

    // initialize data transmission
    DATA_INIT(&huart3);

    // initialize DS2438 (battery monitoring)
    Terminal_Print("DS2438 start ... ");
    errorCode = DS2438_Init(&htim16, GPIOC, GPIO_PIN_0);
    if(errorCode != DS2438_OK)
        Sensor_ErrorHandler(DS2438, errorCode);
    Terminal_Print("DS2438 OK\n\r");

    // initialize PID
    Terminal_Print("PID start ... ");
    PID_Init(&huart4);
    Terminal_Print("PID OK\n\r");

    // initialize output via DSHOT protocol
    Terminal_Print("DShot start ... ");
    errorCode = DShot_Init(&htim3, DSHOT300, &htim14);
    if(errorCode != DSHOT_OK)
        Sensor_ErrorHandler(DSHOT, errorCode);
    Terminal_Print("DShot OK\n\r");

    // initialize IMU 10DOF
    Terminal_Print("IMU start ... ");
    IMU_InitTypeDef imuInit;
    imuInit.hi2c = &hi2c1;
    imuInit.gyroFS = GYRO_500DPS;
    imuInit.accelFS = ACCEL_16G;
    imuInit.gyroDLPF = GYRO_DLDPF_10HZ;
    imuInit.accelDLPF = ACCEL_DLDPF_10HZ;
    imuInit.baroCoeff = IMU_BARO_FILTER_COEFF_16;
    imuInit.baroTempOS = BARO_TEMP_OS_2X;
    imuInit.baroPressOS = BARO_PRESS_OS_16X;
    imuInit.baroSBT = BARO_STANDBY_0P5MS;
    imuInit.htim = &htim17;
    errorCode = IMU_Init(&imuInit);
    if(errorCode != IMU_OK)
        Sensor_ErrorHandler(IMU, errorCode);
    Terminal_Print("IMU OK\n\r");
```

```

// initialize receiver reception with DMA
Terminal_Print("Receiver start ... ");
errorCode = Receiver_Init(SBUS, &huart1);
if(errorCode != RECEIVER_OK)
    Sensor_ErrorHandler(RECEIVER, errorCode);
Terminal_Print("Receiver OK\n\r");

Terminal_Print("Initialisation finished -> Start Real Time System ... \n\r");

...
}

```

Bei jeder Initialisierungsfunktion werden bestimmte Parameter übergeben. Diese bestimmen spezielle Eigenschaften der jeweiligen Komponente (zum Beispiel: welche UART-Peripherie für die Datenübertragung verwendet werden soll). In der Funktion *Receiver_Init()* wird zusätzlich das Real Time System Interrupt (MAIN_ISR) gestartet, das alle 8ms das komplette System aktualisiert und die Steuerung der Drohne ermöglicht.

Am Terminal wird vor jeder Initialisierung der Text „<Komponente> start ...“ und nach erfolgreicher Initialisierung der Text „<Komponente> OK“ ausgegeben. Im Fall eines Initialisierungsfehler wird die Funktion *Sensor_ErrorHandler()* aufgerufen (*siehe: Kapitel 5.1.1*).

Als letzten Schritt wird in einer Endlosschleife die Akkuspannung eingelesen (*siehe: Kapitel 5.2.4*). Diese Funktion befindet sich nicht in der MAIN_ISR, weil der Einlesevorgang mittels dem One-Wire-Protokolls ungefähr 26ms dauert, was zu lange für die Updaterate von 8ms der ISR ist.

Dateiname: main.c Ausschnitt aus main() Funktion
<pre> int main(void) { ... /* Infinite loop */ while(1) { errorCode = DS2438_ReadVoltage(); if(errorCode == DS2438_VOLTAGE_ERROR) Receiver_FailsafeHandler(); } } </pre>

Wenn die Akkuspannung einen kritischen Wert unterschritten hat, wird die Funktion *Receiver_FailsafeHandler()* aufgerufen. Diese bewirkt, dass alle Motoren ausgeschaltet werden, und die Drohne eine Warnung an die Groundstation schickt.

5.1.1 Umgang mit Initialisierungsfehler

```
Dateiname: status_handling.c

/**
 * @brief This function completely stops the program
 * @param sens what sensor has the error
 * @param errorCode
 * @retval None
 */
void Sensor_ErrorHandler(Sensors sens, int8_t errorCode)
{
    char txt[100];

    // choose error source
    switch(sens)
    {
        case DATA_TRANSMIT:
            sprintf(txt, "DATA TRANSMIT ERROR | Code: %d\n\r", errorCode);
            break;

        case DS2438:
            sprintf(txt, "DS2438 ERROR | Code: %d\n\r", errorCode);
            break;

        case IMU:
            sprintf(txt, "IMU ERROR | Code: %d\n\r", errorCode);
            break;

        case RECEIVER:
            sprintf(txt, "RECEIVER ERROR | Code: %d\n\r", errorCode);
            break;

        case DSHOT:
            sprintf(txt, "DSHOT ERROR | Code: %d\n\r", errorCode);
            break;

        case PID:
            sprintf(txt, "PID ERROR | Code: %d\n\r", errorCode);
            break;

        default:
            sprintf(txt, "wrong sensor ERROR | Code: %d\n\r", errorCode);
            break;
    }

    // output error message
    Terminal_Print(txt);

    // turn red LED on and the blue LED off
    __HAL_TIM_SET_COMPARE(LED_TIM, LED_RED_CHANNEL, 10000);
    __HAL_TIM_SET_COMPARE(LED_TIM, LED_BLUE_CHANNEL, 0);
}
```

```
// disable all interrupts
__disable_irq();

// infinite loop
while(1);
}
```

Diese Funktion gibt über das Terminal die Errorquelle mit dem Errorcode aus.

Beispiel: RECEIVER Error | Code 13

Für eine Fehlercodeerklärung – siehe: [Kapitel 5.1.1.1](#)

Weiters wird die rote LED durchgehend eingeschalten, und die blaue LED ausgeschalten. Alle Interrupts werden deaktiviert, und das Programm wird durchgehend mit einer Endlosschleife pausiert. Um den Initialisierungsprozess neu zu starten, muss der Mikrocontroller zurückgesetzt werden.

5.1.1.1 Übersicht Fehlercodes

Für jede Verbindung gibt es eigene Fehlercodes, die sich in enum-Objekte befinden. Diese werden in den Header-Dateien der einzelnen Komponente als Datentyp mit den Namen *<Komponente>_Status* definiert. Zusätzlich hat jedes Objekt einen Wert, der angibt, dass kein Fehler aufgetreten ist: diese sind mit *<Komponente>_OK* definiert.

IMU-Fehlercodes:

```
Dateiname: IMU_10DOF.h

typedef enum IMU_Status
{
    IMU_OK = 0,
    IMU_ADDRESS_ERROR = 1,      // wrong I2C slave address
    IMU_I2C_ERROR = 2,         // no I2C typedef set
    IMU_TIM_ERROR = 3,         // no TIM typedef set

    IMU_MPU_WHOAMI_ERROR = 10, // MPU9250 who am i value wrong
    IMU_MAG_WHOAMI_ERROR = 11, // AK8963 who am i value wrong
    IMU_BARO_CHIPID_ERROR = 12, // BMP280 who am i value wrong

    IMU_BARO_INIT_ERROR = 13   // BMP280 init timeout
} IMU_Status;
```

DS2438 Fehlercodes:

```
Dateiname: DS2438.h

typedef enum DS2438_Status
{
    DS2438_OK = 0,

    DS2438_ERROR = 1,        // sensor not found or initialisation error
    DS2438_VOLTAGE_ERROR = 2 // battery voltage too low
} DS2438_Status;
```

DShot Fehlercodes:

```
Dateiname: dshot.h

typedef enum DShot_Status
{
    DSHOT_OK = 0,
    DSHOT_TIM_ERROR = 100 // no TIM typedef set
} DShot_Status;
```

Receiver Fehlercodes:

```
Dateiname: receiver.h

typedef enum Receiver_Status
{
    RECEIVER_OK = 0,                      // receiver ok
    RECEIVER_UART_ERROR = 1,                // uart configuration doesnt match selected protocol
    RECEIVER_PWM_ERROR = 2,                 // pwm timer not set
    RECEIVER_PPM_ERROR = 3,                 // IBUS selected: PPM not configured correctly
    RECEIVER_TIMEOUT = 4,                  // no data signal found
    PROTOCOL_ERROR = 5,                   // selected protocol wrong

    IBUS_ERROR = 6,                       // IBUS UART DMA not starting
    IBUS_HEADER_ERROR = 7,                // IBUS header is wrong
    IBUS_CHECKSUM_ERROR = 8,              // IBUS checksum is wrong
    IBUS_SIGNAL_LOST_ERROR = 9,           // IBUS signal lost

    SBUS_ERROR = 10,                     // SBUS UART DMA not starting
    SBUS_HEADER_ERROR = 11,               // SBUS header is wrong
    SBUS_FOOTER_ERROR = 12,              // SBUS footer is wrong
    SBUS_SIGNAL_LOST = 13,                // SBUS signal lost flag is set
    SBUS_SIGNAL_FAILSAFE = 14,             // SBUS signal failsafe flag is set
} Receiver_Status;
```

5.2 Bestimmen der Akkuspannung - DS2438

Für die Diplomarbeit wird der Smart Battery Monitor DS2438 für die Spannungsüberwachung des Akkus verwendet, um zu verhindern, dass die Akkuspannung unter einen kritischen Grenzwert fällt und die Drohne abstürzt. Die Kommunikation zwischen dem Flight-Controller und dem DS2438 findet über das One-Wire Protokoll statt.

Als Vorlage für die Programmierung wurde ein in der 4. Klasse erstelltes Projekt verwendet. Nach Absprache mit den Autoren (Lukas Lindmair und Marcel Bieder) wurde die Erlaubnis für die Verwendung und Anpassung an die FPV-Drohne Diplomarbeit erteilt.

5.2.1 One-Wire Protokoll

Der Flight Controller kommuniziert mit dem DS2438 über das One-Wire Protokoll. Dieses besteht aus einer einzelnen halbduplexen, bidirektionalen Leitung. Der Cortex-M7 hat keine One-Wire Peripherie, deswegen muss das Protokoll selbstständig implementiert werden. Die Kommunikation mit dem DS2438 beginnt immer aus einer Initialisierungssequenz. Als nächstes wird ein ROM-Funktionsbefehl ausgeführt. Danach wird ein Memory-Funktions-Befehl gesendet, und am Ende werden die Daten ausgetauscht.

GPIO-Einstellungen in STM32CubeMX:

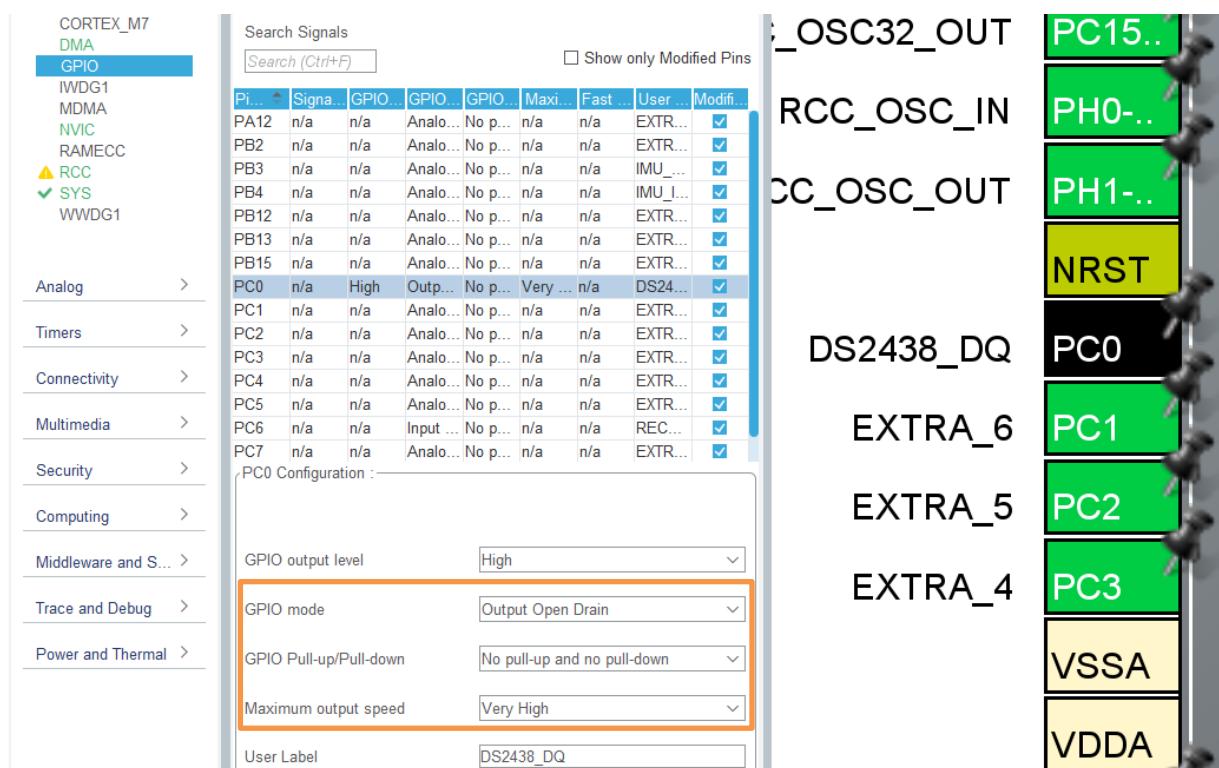


Abbildung 112: STM32CubeMX Einstellungen DS2438

Damit die Kommunikation funktionieren kann, müssen in STM32CubeMX der *GPIO mode* auf *Output Open Drain* und der *GPIO Pull-up/Pull-down* auf *No pull-up and no pull-down* gesetzt werden.

Der Pin PC0 wird am Flight-Controller für DS2438_DQ (Datenleitung) verwendet. Dieser Pin kann bei dem Funktionsaufruf von *DS2438_Init()* bestimmt werden. (siehe: [Kapitel 5.2.3](#))

5.2.1.1 One-Wire Schreibzyklen

Bei der Kommunikation wird das niederwertigste Bit immer zuerst gesendet.

Um das Bit 1 zu senden, müssen folgende Zeitbedingungen eingehalten werden:

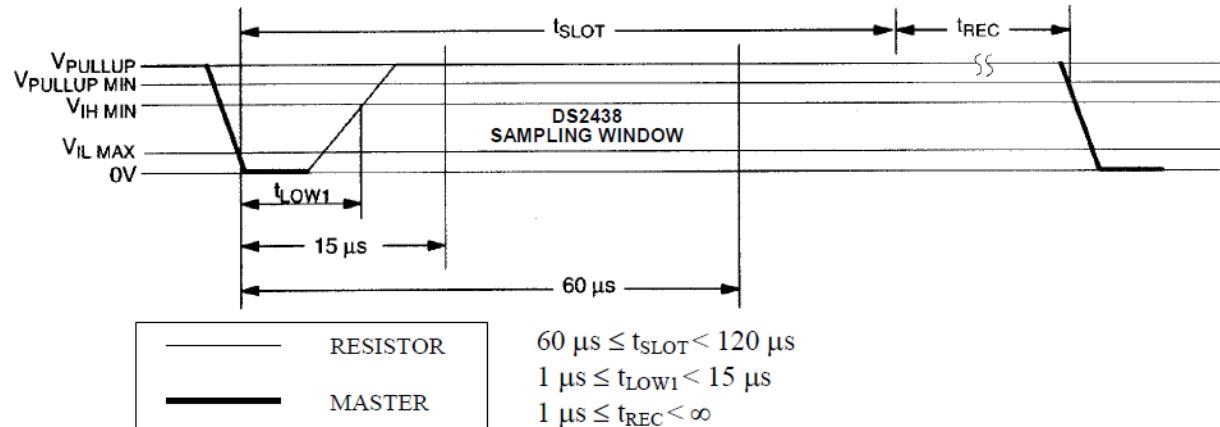


Abbildung 113: One-Wire Bit 1 senden

Um das Bit 0 zu senden, müssen andere Zeitbedingungen eingehalten werden:

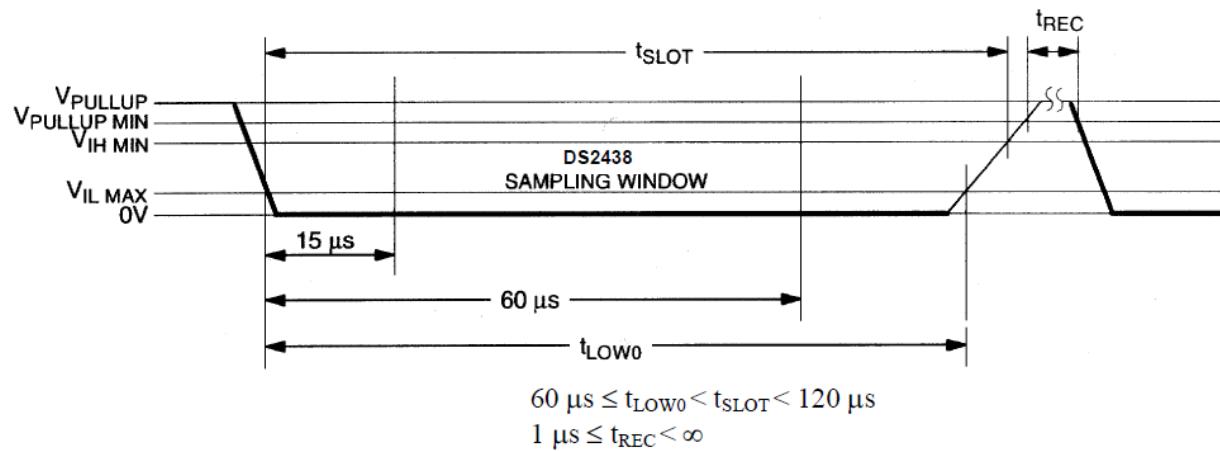


Abbildung 114: One-Wire Bit 0 senden

Ein Byte zum DS2438 senden:

Dateiname: DS2438.c

```
/** 
 * @brief This function writes one byte to the DS2438
 * @param byte byte to write
 * @retval None
 */
void DS2438_WriteByte(uint8_t byte)
{
    for(int8_t i = 0; i < 8; i++)
    {
        DS2438_WriteBit(byte & 0x01);
        byte >>= 1;
    }
}

/** 
 * @brief This function writes one bit to the DS2438
 * @param bit bit to write
 * @retval None
 */
void DS2438_WriteBit(int8_t bit)
{
    if(bit == 1)
    {
        HAL_GPIO_WritePin(ds2438_GPIOPort, ds2438_GPIOPin, GPIO_PIN_RESET);
        DS2438_DelayUs(10);
        HAL_GPIO_WritePin(ds2438_GPIOPort, ds2438_GPIOPin, GPIO_PIN_SET);
        DS2438_DelayUs(70);
    }
    else
    {
        HAL_GPIO_WritePin(ds2438_GPIOPort, ds2438_GPIOPin, GPIO_PIN_RESET);
        DS2438_DelayUs(60);
        HAL_GPIO_WritePin(ds2438_GPIOPort, ds2438_GPIOPin, GPIO_PIN_SET);
        DS2438_DelayUs(10);
    }
}
```

Um auf eine bestimmte Page Daten schreiben zu können, müssen diese zuerst auf das Scratchpad geschrieben werden. Danach muss das Scratchpad in den ROM oder RAM kopiert werden.

Zuerst wird die Initialisierungssequenz ausgeführt, und danach wird der Skip-ROM- (0xCC) und der Write-Scratchpad-Befehl (0x4E) ausgeführt. Anschließend muss die gewünschte Page-Nummer übertragen werden. Darauf folgen die Daten und eine zweite Initialisierungssequenz. Damit die Daten vom Scratchpad auch in den Speicher des Sensors übertragen werden, muss noch ein Skip-ROM- (0xCC) und ein Copy-Scratchpad-Befehl (0x48), gefolgt von der Page-Nummer, ausgeführt werden.

Zu einer Page Daten senden:

Dateiname: DS2438.c

```
/**  
 * @brief This function writes the data to one page of the DS2438  
 * @param page page number (0 - 7)  
 * @param pageData data of page  
 * @return DS2438_Status  
 */  
DS2438_Status DS2438_WritePage(uint8_t page, int16_t *pageData)  
{  
    // reset + presence pulse  
    if(DS2438_Reset() == DS2438_ERROR)  
        return DS2438_ERROR;  
  
    // copy current data to scratchpad  
    DS2438_WriteByte(DS2438_SKIP_ROM);  
    DS2438_WriteByte(DS2438_WRITE_SP);  
    DS2438_WriteByte(page);  
  
    for(uint8_t i = 0; i < 9; i++)  
        DS2438_WriteByte(pageData[i]);  
  
    // reset + presence pulse  
    if(DS2438_Reset() == DS2438_ERROR)  
        return DS2438_ERROR;  
  
    DS2438_WriteByte(DS2438_SKIP_ROM);  
    DS2438_WriteByte(DS2438_COPY_SP);  
    DS2438_WriteByte(page);  
  
    return DS2438_OK;  
}
```

5.2.1.2 One-Wire Lesezyklen

Genau wie bei den Schreibzyklen gibt es auch spezielle Zeitkriterien, die bei den Lesezyklen eingehalten werden müssen. Es wird auch das niederwertigste Byte zuerst empfangen.

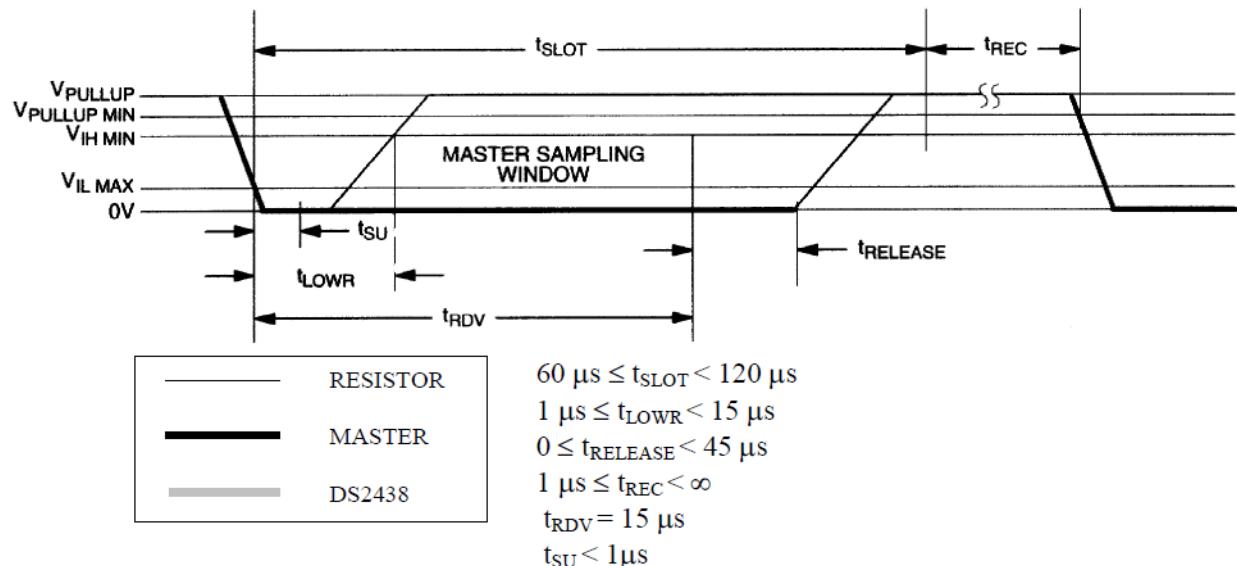


Abbildung 115: One-Wire Bit empfangen

Ein Byte vom DS2438 auslesen:

Dateiname: DS2438.c

```
/*
 * @brief This function reads one byte from the DS2438 (LSB first)
 * @return uint8_t
 */
uint8_t DS2438_ReadByte(void)
{
    uint8_t byte = 0;

    for(int8_t i = 0; i < 8; i++)
        byte |= (DS2438_ReadBit() << i);

    return byte;
}

/*
 * @brief This function reads one bit from the DS2438
 * @return int8_t
 */
int8_t DS2438_ReadBit(void)
{
    int8_t bit = 0;

    HAL_GPIO_WritePin(ds2438_GPIOPort, ds2438_GPIOPin, GPIO_PIN_RESET);
    DS2438_DelayUs(10);
    HAL_GPIO_WritePin(ds2438_GPIOPort, ds2438_GPIOPin, GPIO_PIN_SET);
    DS2438_DelayUs(10);
}
```

```

// read current pin level
bit = HAL_GPIO_ReadPin(ds2438_GPIOPort, ds2438_GPIOPin) == GPIO_PIN_SET;

DS2438_DelayUs(60);

return bit;
}

```

Um von einer bestimmten Page Daten lesen zu können, müssen diese zuerst vom Sensor auf das Scratchpad geschrieben werden. Danach müssen die Daten vom Scratchpad eingelesen werden.

Zuerst wird die Initialisierungssequenz ausgeführt und danach wird der Skip-ROM- (0xCC) und der Recall-Memory-Befehl (0xB8) ausgeführt. Anschließend muss die gewünschte Page-Nummer übertragen werden. Darauf folgt eine zweite Initialisierungssequenz. Um jetzt die Daten vom Scratchpad zu lesen, muss ein Skip-ROM- (0xCC) und ein Read-Scratchpad-Befehl (0xBE), gefolgt von der Page-Nummer, ausgeführt werden. Danach können die Daten auf der One-Wire-Leitung empfangen werden.

Von einer Page Daten auslesen:

```

Dateiname: DS2438.c


```

/**
 * @brief This function reads the data from one page of the DS2438
 * @param page page number (0 - 7)
 * @param pageData data of page
 * @return DS2438_Status
 */
DS2438_Status DS2438_ReadPage(uint8_t page, int16_t *pageData)
{
 // reset + presence pulse
 if(DS2438_Reset() == DS2438_ERROR)
 return DS2438_ERROR;

 // copy current data to scratchpad
 DS2438_WriteByte(DS2438_SKIP_ROM);
 DS2438_WriteByte(DS2438_RECALL_MEM);
 DS2438_WriteByte(page);

 // reset + presence pulse
 if(DS2438_Reset() == DS2438_ERROR)
 return DS2438_ERROR;

 // read scratchpad data
 DS2438_WriteByte(DS2438_SKIP_ROM);
 DS2438_WriteByte(DS2438_READ_SP);
 DS2438_WriteByte(page);

 for(int8_t i = 0; i < 9; i++)
 pageData[i] = DS2438_ReadByte();
}

```


```

```

    return DS2438_OK;
}
}
```

5.2.1.3 Initialisierungssequenz

Die Initialisierungssequenz besteht immer aus einen Reset- und einen Presence-Puls, die die folgenden zeitlichen Anforderungen einhalten müssen:

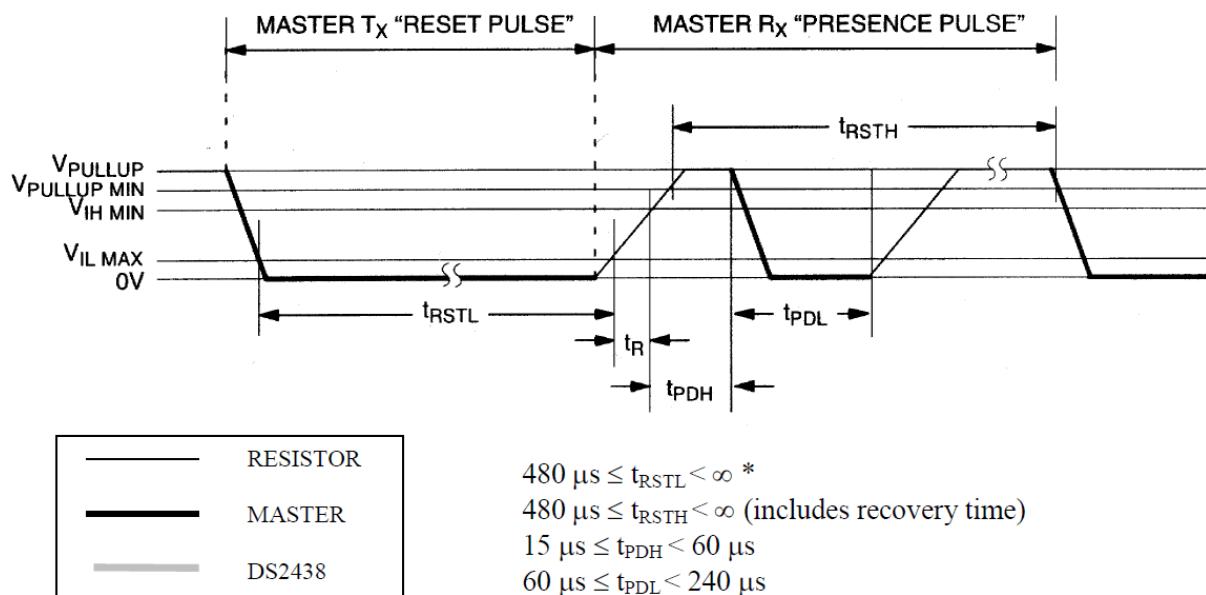


Abbildung 116: One-Wire Initialisierungssequenz

Wenn der One-Wire Slave bei dem Presence-Puls nicht nach $60\mu\text{s}$ bis $240\mu\text{s}$ die Leitung in den high-Zustand setzt, besteht ein Problem bei der Kommunikation zwischen Master und Slave.

Initialisierungssequenz DS2438:

```

Dateiname: DS2438.c

/**
 * @brief This function resets / checks device presence
 * @return DS2438_Status
 */
DS2438_Status DS2438_Reset(void)
{
    // reset DS2438
    // send reset pulse (min 480us)
    HAL_GPIO_WritePin(ds2438_GPIOPort, ds2438_GPIOPin, GPIO_PIN_RESET);
    DS2438_DelayUs(480);
    // release line -> change to receive mode
    HAL_GPIO_WritePin(ds2438_GPIOPort, ds2438_GPIOPin, GPIO_PIN_SET);

    // wait until slave sends presence pulse
    DS2438_DelayUs(70);
}
```

```
// read current pin state
int8_t pin = HAL_GPIO_ReadPin(ds2438_GPIOPort, ds2438_GPIOPin);
DS2438_DelayUs(410);

// check pin state (0 -> found, 1 -> not found)
if(pin == GPIO_PIN_SET)
    return DS2438_ERROR;

return DS2438_OK;
}
```

5.2.1.4 ROM-Funktionsbefehl

Es gibt viele verschiedene ROM-Funktionsbefehle. Für das Spannungsmonitoring ist nur der SKIP-ROM-Befehl (0xCC) wichtig. Dieser Befehl erlaubt es dem Master, direkt einen Memory-Funktionsbefehl zu senden, ohne einen ROM-Code senden zu müssen. Dadurch wird Zeit gespart.

5.2.1.5 Memory-Funktionsbefehl

Der einzige Memory-Funktionsbefehl, der wichtig für das Spannungsmonitoring ist, ist der CONVERT-V (0xB4) Befehl. Diese Anweisung startet die Analog-Digital-Spannungswandlung des Sensors. Wenn die Umwandlung beendet ist, wird das ADB-Flag (A/D Conversion Busy Flag) vom Sensor auf 0 gesetzt.

5.2.2 Registerübersicht DS2438

Der DS2438 ist unterschiedliche Register (Pages) unterteilt, die alle notwendigen Einstellungen und Messdaten des Sensors beinhalten.

PAGE	BYTE	CONTENTS	R/W	NV
0	0	STATUS/ CONFIGURATION	R/W	YES
	1	TEMPERATURE LSB	R	NO
	2	TEMPERATURE MSB	R	NO
	3	VOLTAGE LSB	R	NO
	4	VOLTAGE MSB	R	NO
	5	CURRENT LSB	R	NO
	6	CURRENT MSB	R	NO
	7	THRESHOLD	R/W	YES

Abbildung 117: Registerübersicht DS2438

Für die Spannungsüberwachung ist nur die Page 0 wichtig, da hier alle Einstellungen und Messwerte gesetzt beziehungsweise ausgelesen werden.

Byte	Verwendungszweck
0	Einstellungen setzen
3 und 4	Nieder- und hochwertiges Byte des Spannung-Messwertes

5.2.3 Initialisierung DS2438

```
Dateiname: DS2438.c

/**
 * @brief This function initializes the DS2438
 * @param htim pointer to TIM_HandleTypeDef (timer for us delay)
 * @param gpio_Port GPIOx (port of DQ Pin)
 * @param gpio_Pin GPIO_PIN_x (pin of DQ Pin)
 * @return DS2438_Status
 */
DS2438_Status DS2438_Init(TIM_HandleTypeDef *htim, GPIO_TypeDef *gpio_Port, uint16_t
gpio_Pin)
{
    if(htim == NULL)
        return DS2438_ERROR;

    DS2438_DelayTimer = htim;
    HAL_TIM_Base_Start(DS2438_DelayTimer); // start timer for DS2438_DelayUs

    ds2438_GPIOPort = gpio_Port;
    ds2438_GPIOPin = gpio_Pin;

    if(DS2438_Reset() == DS2438_ERROR)
        return DS2438_ERROR;

    // set Vad as A/D converter input
    int16_t pageData[9] = {0x00};

    if(DS2438_ReadPage(0x00, pageData) == DS2438_ERROR)
        return DS2438_ERROR;

    // pageData[0] |= 0x08; // supply voltage
    pageData[0] &= 0xF7; // external input

    if(DS2438_WritePage(0x00, pageData) == DS2438_ERROR)
        return DS2438_ERROR;

    // check current voltage
    int8_t errorCode = DS2438_ReadVoltage();
    if(errorCode != DS2438_OK)
        return errorCode;

    return DS2438_OK;
}
```

Der/Die BenutzerIn kann sich aussuchen, ob die Versorgungsspannung oder die Spannung am Eingang vom A/D-Wandler gemessen werden soll. Dies funktioniert mit dem AD-Bit in der Page 0 im Byte[0]. Wenn das AD-Bit auf 1 gesetzt wird, wird die Versorgungsspannung gemessen. Wenn das Bit auf 0 gesetzt wird, misst der Sensor die Eingangsspannung vom A/D-Wandler.

5.2.4 Spannungsüberwachung

Die eingelesenen Spannungswerte werden in folgendem Format gespeichert:

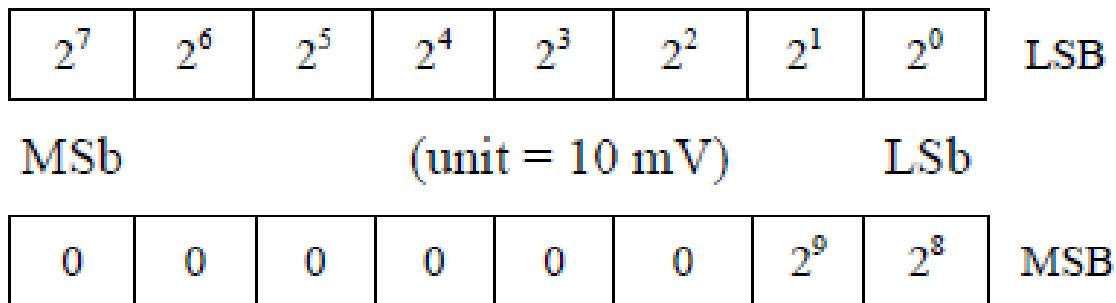


Abbildung 118: DS2438 Spannungsregister Format

Um mit den eingelesenen Werten einen Spannungswert zu bilden, müssen MSB und LSB zusammengefügt werden. Da die Werte im Spannungsregisters eine Auflösung von 10mV pro LSB haben, muss der Registerwert mit 10mV multipliziert oder durch 100 gerechnet werden, um den eigentlichen Spannungswert zu bestimmen.

Spannung einlesen:

```
Dateiname: DS2438.c

/*
 * @brief This function reads the current voltage value of the DS2438
 * @attention the voltage gets stored in the global variable 'ds2438_Voltage'
 * @return DS2438_Status
 */
DS2438_Status DS2438_ReadVoltage(void)
{
    // start measurement (send CONVERT T command)
    if(DS2438_StartVoltageMeasurement() == DS2438_ERROR)
        return DS2438_ERROR;

    // wait for measurement to be complete (ADB flag: 1 = busy, 0 = ready)
    while(DS2438_ReadControlVoltageFlag());

    int16_t pageData[9] = {0x00};

    // read data
    if(DS2438_ReadPage(0x00, pageData) == DS2438_ERROR)
        return DS2438_ERROR;

    // extracting voltage bytes
    int16_t voltageLSB = pageData[3];
    int16_t voltageMSB = pageData[4];

    ds2438_Voltage = (((voltageMSB & 0x3) << 8) | (voltageLSB)) / 100.0;
    ds2438_Voltage *= 3; // times 3 because of resistor voltage divider
}
```

```
if(ds2438_Voltage <= DS2438_MIN_VOLTAGE)
    return DS2438_VOLTAGE_ERROR;

return DS2438_OK;
}
```

Da der A/D-Wandler des DS2438 nur einen Spannungsbereich von 0-10V hat, muss die Eingangsspannung vor dem Sensor geteilt werden. Auf der Platine des Flight-Controllers ist vor dem Eingang des A/D-Wandlers des DS2438 ein 3:1 Spannungsteiler eingebaut. Daher wird der Spannungswert mit einem Faktor von drei multipliziert, um den eigentlichen Messwert zu bestimmen.

Der endgültige Spannungswert wird in der globalen Variable *ds2438_Voltage* gespeichert. Wenn die Spannung einen kritischen Wert unterschreitet (standardmäßig 22.2V), liefert die Funktionen einen DS2438_VOLTAGE_ERROR.

Wenn der Error geschickt wird, sollte der Flug der Drohne so schnell wie möglich beendet werden, damit der Akku nicht beschädigt wird.

5.3 Real Time System Interrupt (MAIN_ISR)

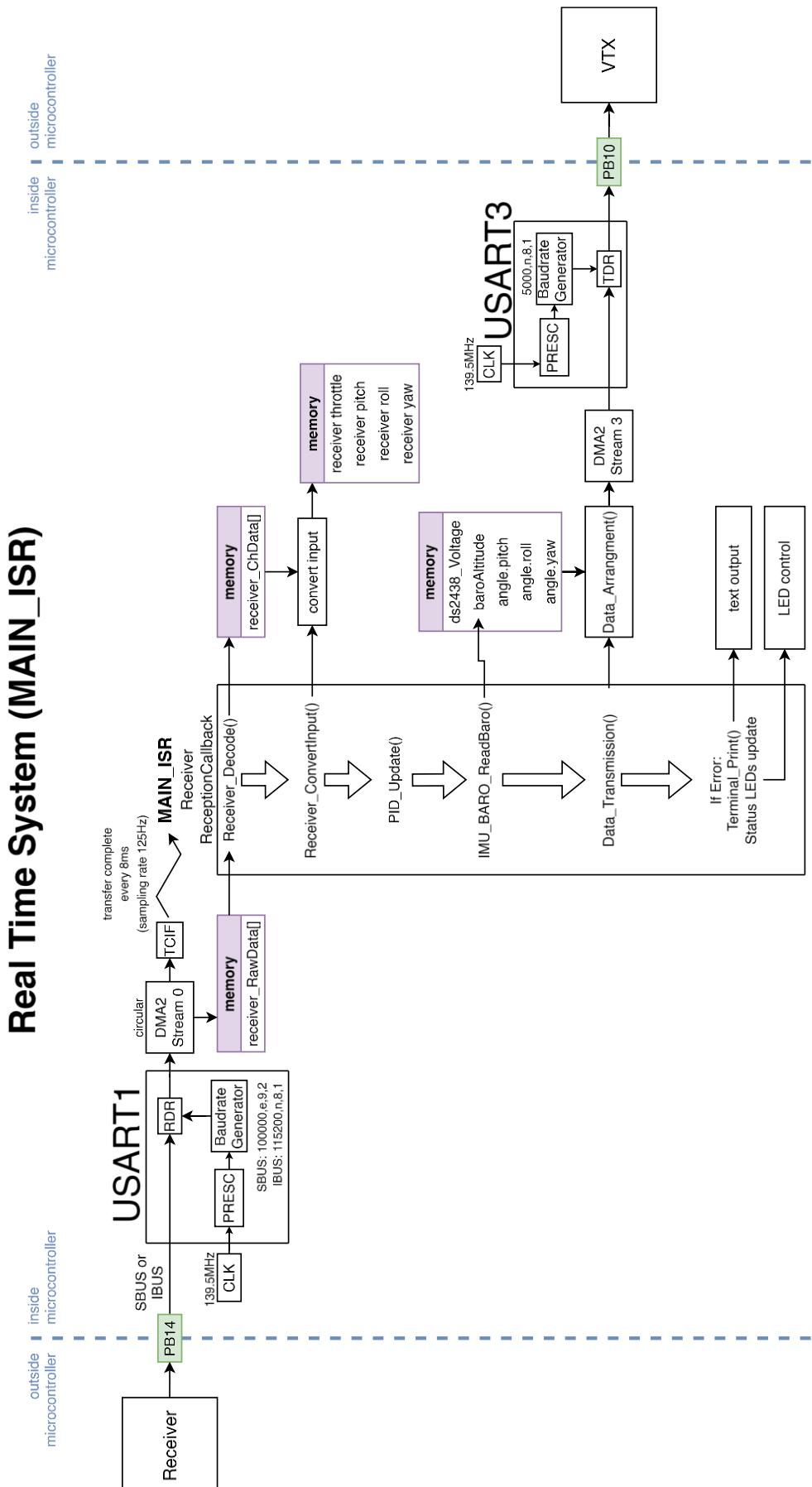


Abbildung 119: Real Time System Interrupt Architektur

Durch das Einlesen der Receiver-Daten mit dem DMA2-Controller über die USART1-Peripherie, wird alle 8ms ein transfer complete interrupt ausgelöst. Die interrupt service routine (ISR) ist mit der MAIN_ISR (*Receiver_ReceptionCallback()*) realisiert (siehe: [Kapitel 5.4.4](#)).

Auch bei Verbindungsverlust schickt der Receiver weiterhin alle 8ms ein Datenpaket. Dadurch ergibt sich eine konstante Abtastrate von 125Hz aller Sensoren.

Die empfangenen Rohdaten werden mit der Funktion *Receiver_Decode()* in weiterverwertbare Werte der einzelnen Kanäle dekodiert (siehe: [Kapitel 5.4.4 Teil #1](#)). Die Kanäle beinhalten die Position der Joysticks und ausgewählten Schaltern der Fernsteuerung.

Mit der Funktion *Receiver_ConvertInput()* werden die Kanalwerte in Throttle-, Pitch-, Roll- und Yaw-Werte umgewandelt (siehe: [Kapitel 5.4.4 Teil #2](#)). Dabei beschreibt der Throttle-Wert von 0% bis 100% die Motorgeschwindigkeit aller Motoren und die anderen Werte die gewünschte Drehung um die jeweilige Achse in Grad.

Mit der Funktion *PID_Update()* werden diese Werte als Führungsgrößen (Sollwerte) an den PID-Regler übergeben (siehe: [Kapitel 5.8.2](#)). Dieser hat die Aufgabe, die Drohne in die richtige Lage zu bringen, indem er die Geschwindigkeiten der Motoren mit Hilfe der Sollwerte verändert (siehe: [Kapitel 2.2, 5.6](#)).

Die Höhe der Drohne kann mit dem Barometer bestimmt werden. Der Luftdruck und die Temperatur werden mit der I2C1-Peripherie mit der Funktion *IMU_BARO_ReadBaro()* eingelesen und in die aktuelle Flughöhe umgewandelt (siehe: [Kapitel 5.5.4.3, 5.5.4.4](#)).

Als letzten Schritt werden die Sensordaten (Akkuspannung, Höhe, Lagewinkel) zur Groundstation über einen video transmitter (VTx) mit Hilfe der USART3-Peripherie und dem DMA2-Controller geschickt (siehe: [Kapitel 6.2.5](#)).

Wenn ein Fehler während der Laufzeit der MAIN_ISR aufgetreten ist, wird dieser über das Terminal (text output) und den eingebauten Status-LEDs (LED control) angezeigt (siehe: [Kapitel 5.9, 5.10, 5.11](#)).

5.4 Einlesen der Daten von Fernsteuerung

Um eine FPV-Drohne steuern zu können, muss mindestens ein 4-Kanal Sender und Empfänger verwendet werden, um die gewünschten Steuerungsdaten, Throttle, Pitch, Roll und Yaw, zu senden. Mit weiteren Kanälen können zusätzliche Funktionen, wie zum Beispiel ON/OFF-Switch, realisiert werden.

Für das Projekt wurde der Turnigy 9X 9Ch Mode 2 Transmitter mit dem TGY-iA6C Receiver ausgewählt. Diese Kombination bietet eine große Auswahl von programmierbaren Schaltern, deren Stellungen mittels eines schnellen, digitalen seriellen Protokolls übertragen wird.



Abbildung 121: Receiver

Abbildung 120: Fernsteuerung

In der Abbildung 122 wird die verwendete Tastenbelegung auf der Fernsteuerung dargestellt:

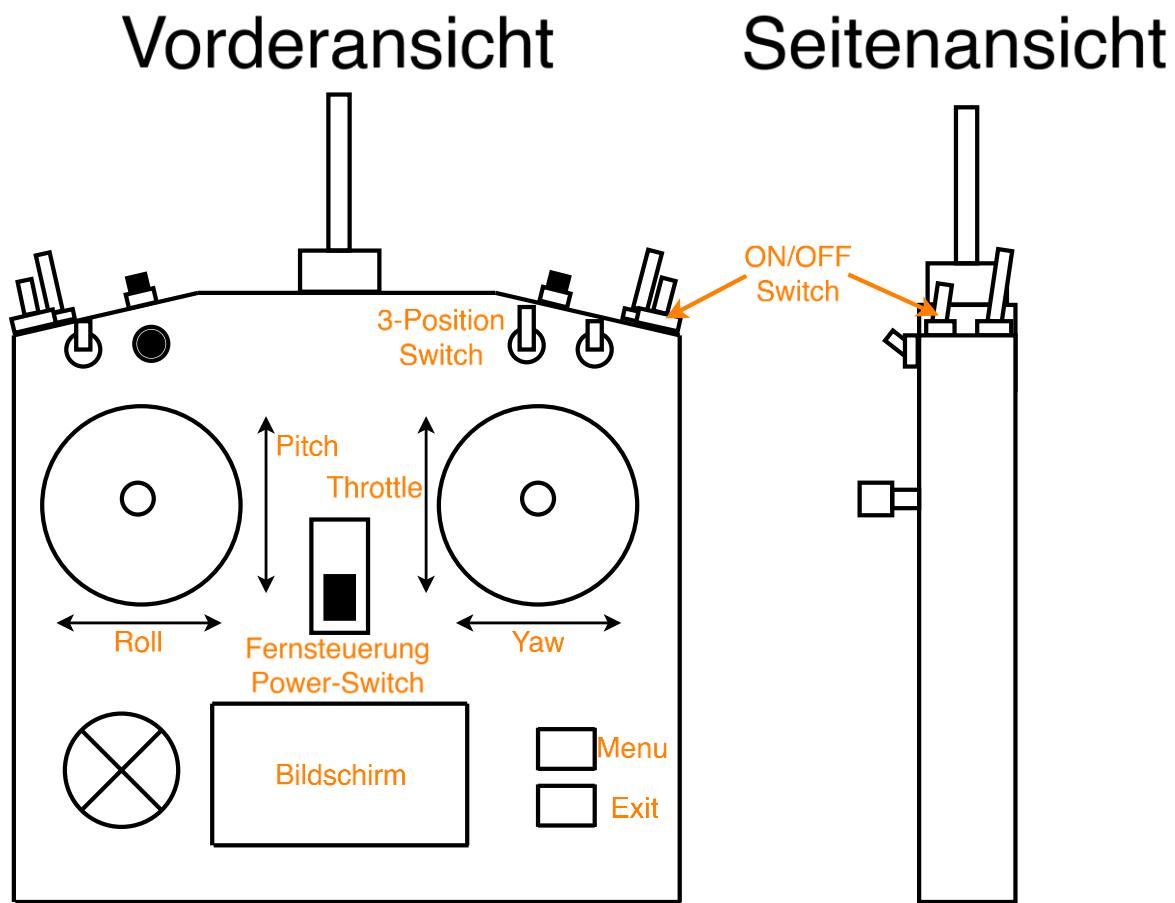


Abbildung 122: Fernsteuerung Tastenbelegung

Die zwei großen Joysticks reichen aus, um die Drohne steuern zu können. Für eine bessere Bedienung werden zwei zusätzlich Schalter verwendet.

Der ON/OFF-Switch ermöglicht das Ein- und Ausschalten der Fernsteuerungseingaben.

Mit dem 3-Position-Switch kann zwischen mehreren Flugmodi unterschieden werden.

Um die richtige Funktionalität der Schalter zur gewährleisten - siehe: [Kapitel 5.4.1](#)

Wenn der ON/OFF-Switch nach unten zeigt, schalten sich die Motoren aus und alle Fernsteuerungsbefehle werden übernommen. Wenn der Schalter nach oben zeigt, wirkt der 3-Position-Switch als Modus Auswahl.

In der oberen Stellung befindet sich die Drohne im „Safe-Mode“. Das bedeutet, dass der maximale Throttle-Wert, der zur ESC geschickt werden kann, begrenzt wird.

In der mittleren Stellung schaltet wird der „Normal-Mode“ eingeschaltet. Dieser Modus verhält sich gleich, wie der „Safe-Mode“, nur erlaubt er, höhere Motordrehzahl.

Die untere Position stellt einen Hover-Mode ein. Die Drohne versucht jetzt, sich selbst gerade in der Ebene mit Sensoren- und Throttlewerten zu halten.

Wichtig: Wenn die Versorgungsspannung der Fernsteuerung weniger als 8,5V beträgt, beginnt ein Buzzer alle 5 Sekunden den Benutzer zu alarmieren. Die aktuelle Spannung kann auf dem Bildschirm der Fernsteuerung überprüft werden.

Switch Error:

Beim Starten der Fernsteuerung kann es vorkommen, dass am Bildschirm „switch-error!“ angezeigt wird. Dieser Fehler wird angegeben, wenn die Schalter nicht auf High-Position sind.

Das bedeutet: Beim Starten der Fernsteuerung müssen alle Schalter nach oben zeigen beziehungsweise die Schalter, die auf der Oberseite montiert worden sind, müssen nach hinten zeigen, wie in Abbildung 122 dargestellt ist.

5.4.1 Konfiguration Fernsteuerung

Damit die ausgewählte Fernsteuerung mit unserem System funktioniert, müssen grundlegende Vorbereitungen vor der Inbetriebnahme getroffen und kontrolliert werden.

Verbindung mit Empfänger:

Um die Verbindung zu testen, schaltet man die Fernsteuerung und den Receiver ein. Die eingebaute Status-LED des Receivers fängt schnell zum Blinken an. Wenn die LED nach ein paar Sekunden anfängt durchgehend zu leuchten, ist die Fernsteuerung mit dem Receiver verbunden.

Wenn die LED anfängt langsamer zu blinken, wurde die Verbindung nicht gefunden und die Fernsteuerung muss mit dem Receiver neu gekoppelt werden:

1. Schalte die Fernsteuerung und den Receiver aus
2. Halte den Bind-Knopf auf der Rückseite der Fernsteuerung, während diese angeschaltet wird
3. Schalte den Receiver ein
4. Warte, bis die Status-LED auf dem Receiver durchgehend leuchtet
5. Lasse den Bind-Knopf los und starte die Fernsteuerung neu, um die Koppelung zu testen

Einstellungen in Fernsteuerung:

Damit die Signale richtig von der Flugsoftware interpretiert werden können, müssen folgende Einstellungen in der Fernsteuerung getroffen werden:

Beim längeren gedrückt halten der MENU-Taste wird eine Auswahl zwischen „System Setting“ und „Function Setting“ angezeigt.

Die folgenden Tabellen zeigen die Einstellung, die für die richtige Funktionsweise notwendig sind:

System Setting	
Type Select	ACRO
Modeuat	PPM
Stick Set	MODEL 1
Output Select	PPM s-BUS

Function Setting		
AUX-CH (für ON/OFF Switch)	CH5 Gear	
PROG. MIX (für 3-Position-Switch)	MIX1	STATE: ACT MASTER: GYR SLAVE: FLP OFFSET: 000 UPRATE: -100 DNRATE: 100 SW: NOR
	MIX2	STATE: ACT MASTER: GYR SLAVE: FLP OFFSET: 000 UPRATE: 000 DNRATE: 000 SW: ID1
	MIX3	STATE: ACT MASTER: GYR SLAVE: FLP OFFSET: 000 UPRATE: 100 DNRATE: -100 SW: ID2
DISPLAY	Kontrolle der Kanal-Ausgänge	

Diese Einstellungen sind notwendig, damit der ON/OFF Switch und der 3-Position-Switch richtig dekodiert werden, und eine richtige Funktionalität nachweisen können.

Zur Kontrolle werden in Balkendiagrammen die einzelnen Kanalwerte unter DISPLAY angezeigt.

5.4.2 Unterstützte Protokolle

Der Receiver unterstützt eine gleichzeitige Signalausgabe von einem analogen und digitalen Signal. Der Pin PPM gibt immer ein PPM-Signal aus, während der zweite Pin S.BUS entweder ein S.Bus oder I.Bus Signal ausgibt. Die Ausgabe kann in den Fernsteuerungseinstellungen festgelegt werden ([siehe: Kapitel 5.4.1](#)).

5.4.2.1 PPM (Pulse Position Modulation)

Das PPM-Signal ist das einzige analoge Signal, das der Receiver ausgibt. Die Daten können mit der Timer-Peripherie eingelesen werden, indem die Länge der Periodendauer bestimmt wird. Die Daten kommen in 9 Kanälen mit einem 8,1ms low-aktiven Ruhezustand. Die Kanalwerte befinden sich in den Bereich 0,5ms (0% Throttle) bis 1,5ms (100% Throttle).

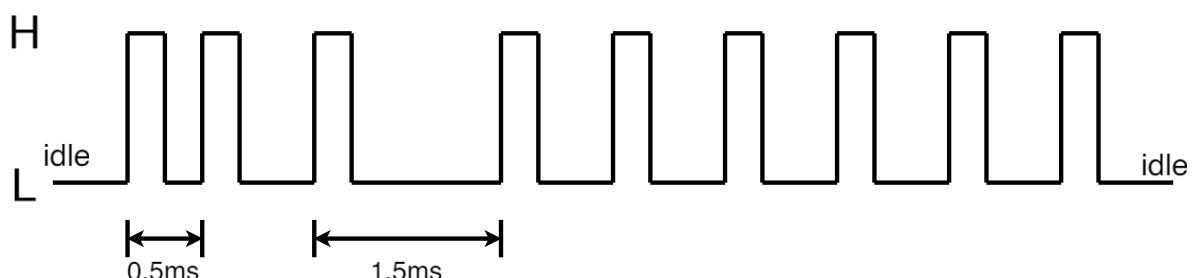


Abbildung 123: Beispiel PPM-Signal

Dadurch, dass die digitalen Protokolle um ein Vielfaches schneller und nicht sehr fehleranfällig sind, unterstützt die Steuerungssoftware das PPM-Protokoll nicht. Das PPM-Signal wird nur bei der Initialisierung für einen Verbindungstest mit GPIO-Input und nicht für das Fliegen der Drohne verwendet.

5.4.2.2 S.Bus

Das S.Bus-Protokoll ist das empfohlene Protokoll für die Benutzung der Drohne. Es handelt sich um ein digitales Protokoll, das mittels der UART-Peripherie eingelesen wird. Das Protokoll ist schneller als das analoge Signal PPM und weniger fehleranfällig als das I.Bus-Protokoll, da das Signal mit invertierten Pegeln übertragen wird. Die Datenpakete werden kontinuierlich gesendet, sind ungefähr 3ms lang und besitzen einen ungefähr 4,7ms langen low-aktiven Ruhezustand. Der digitale Wertebereich befindet sich zwischen 350 und 1680.

UART-Konfiguration in STM32CubeMX:

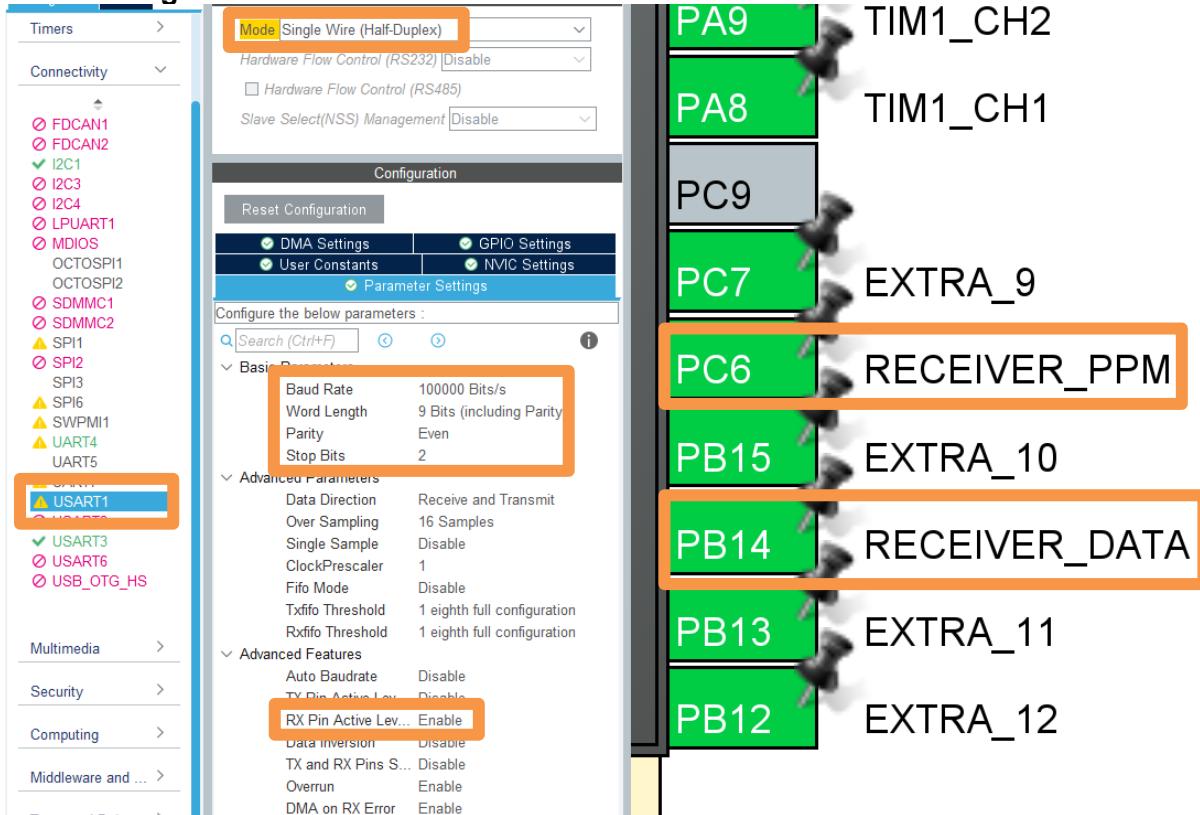


Abbildung 124: STM32CubeMX Einstellungen S.Bus

Die wichtigsten UART-Einstellungen sind in der folgenden Tabelle dargestellt:

Einstellung	Wert
Mode	Single Wire (Half-Duplex)
Baud Rate	100000 Bits/s
Word Length	9 Bits (including Parity)
Parity	Even
Stop Bits	2
RX Pin Active Level Inversion	Enable

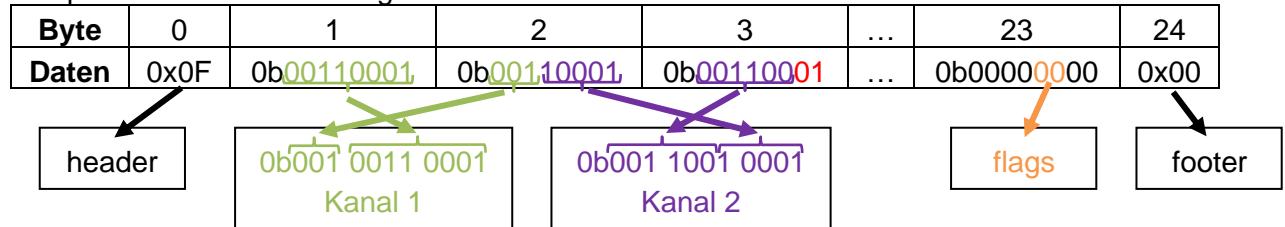
Protokolldaten:

Das S.Bus Protokoll besteht aus 25 Bytes:

Byte[0]	S.Bus header, 0x0F			
Byte[1...22]	Daten, 16 Kanäle			
Byte[23]	Bit[0] (0x01) Kanal 17	Bit[1] (0x02) Kanal 18	Bit[2] (0x04) Frame Lost Flag	Bit[3] (0x08) Failsafe Flag
Byte[24]	S.Bus footer, 0x00			

Die Bytes[1...22] beinhalten die Daten der einzelnen Kanäle. Jeder Kanal besteht aus 11 Bits, die nacheinander gesendet werden, wobei die ersten Bits des Kanals den niedrigsten Stellenwert haben.

Beispiel für eine Dekodierung:



5.4.2.3 I.Bus

Das I.Bus-Protokoll ist das zweite, digitale Protokoll, welches der Receiver ausgeben kann. Die Daten werden über die UART-Peripherie eingelesen. Die Datenpakete werden kontinuierlich gesendet, sind ungefähr 3ms lang und besitzen einen ungefähr 4,7ms langen high-aktiven Ruhezustand. Im Gegensatz zum S.Bus Protokoll kann mit I.Bus kein Verbindungsverlust festgestellt werden. Der digitale Wertebereich befindet zwischen 1070 und 1920.

UART-Konfiguration in STM32CubeMX:

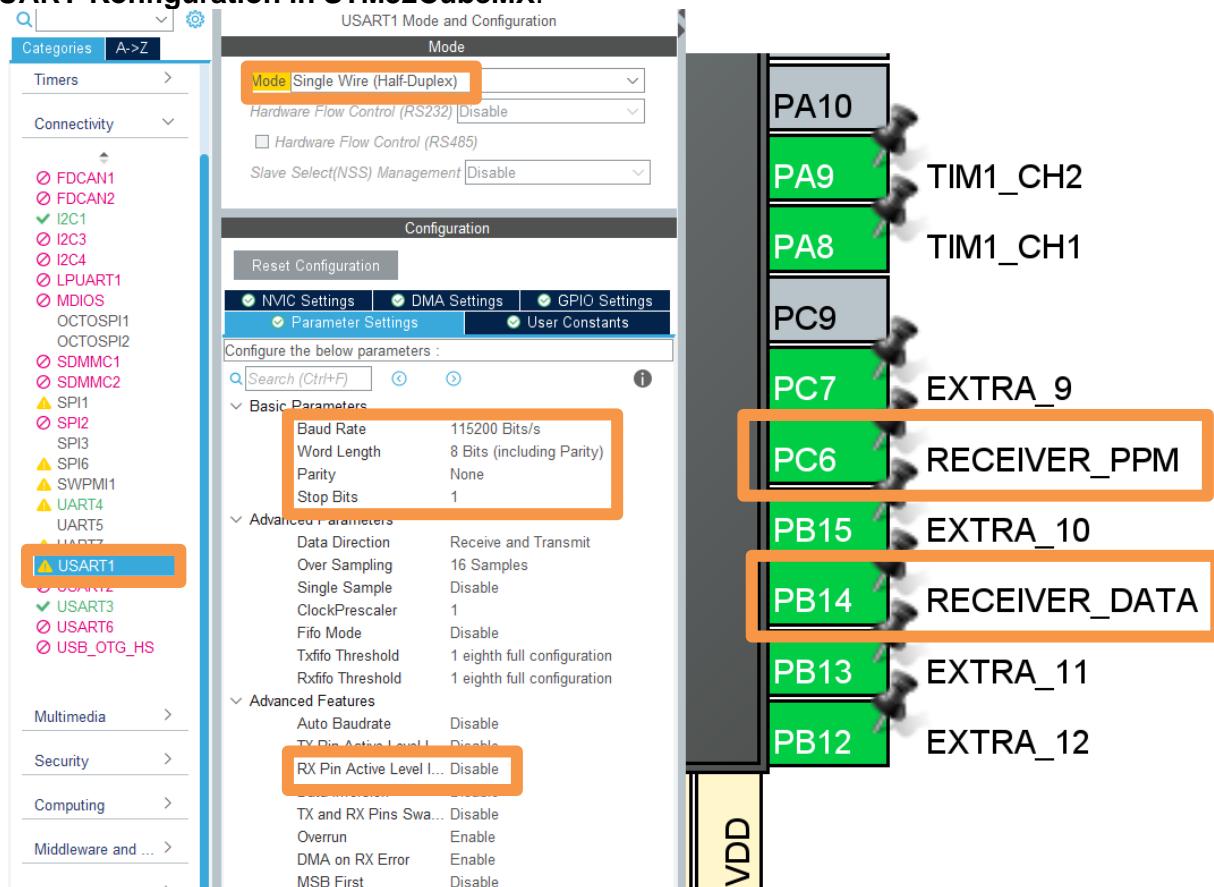


Abbildung 125: STM32CubeMX Einstellungen I.Bus

Die wichtigsten UART-Einstellungen sind in der folgenden Tabelle dargestellt:

Einstellung	Wert
Mode	Single Wire (Half-Duplex)
Baud Rate	115200 Bits/s
Word Length	8 Bits (including Parity)
Parity	None
Stop Bits	1
RX Pin Active Level Inversion	Disable

Protokolldaten:

Das I.Bus Protokoll besteht aus 32 Bytes:

Byte[0]	Protokolllänge, 0x20
Byte[1]	Command Code, 0x40
Byte[2...29]	Daten, 14 Kanäle
Byte[30...31]	Checksumme

Die Bytes[2...29] beinhalten die Daten der einzelnen Kanäle. Jeder Kanal besteht aus 2 Bytes, welche in der little-endian-byte-order geschickt werden. Die Checksumme besteht auch aus 2 Byte, welche in der little-endian-byte-order gesendet werden. Die Checksumme berechnet sich aus 0xFFFF minus der Summe der ersten 30 Bytes.

Beispiel für die Dekodierung:

Byte	0	1	2	3	...	30	31
Daten	0x20	0x40	0xDC	0x05	...	0x12	0x34
Protokolllänge		Command Code		0x05DC Kanal 1		0x3412 Checksumme	

Beispiel Checksumme: $0xFFFF - \langle \text{Summe aller Kanalwerte} \rangle = 0x3412$

5.4.3 Initialisierung Empfangssoftware

Dadurch, dass der Receiver die Daten kontinuierlich sendet, wird ein DMA (direct memory access) - Controller zum Einlesen verwendet, der durchgehend auf die Werte wartet und diese einliest.

DMA-Konfiguration in STM32CubeMX:

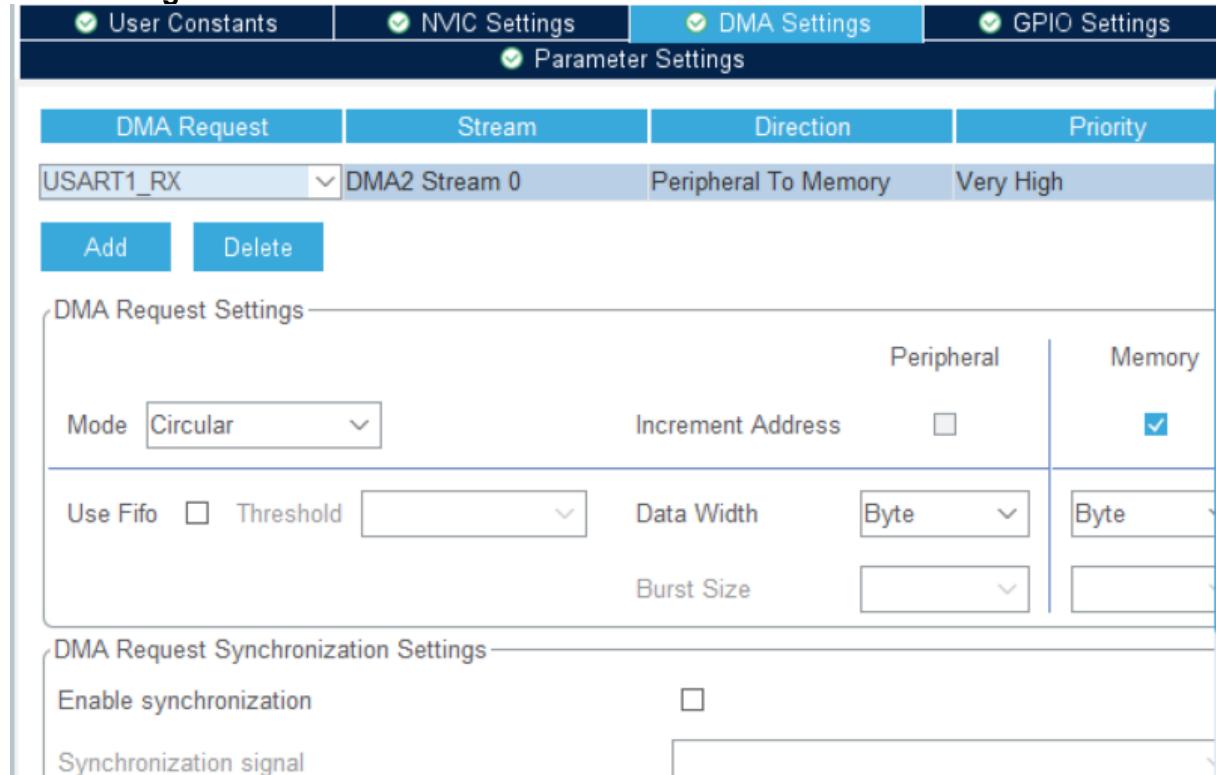


Abbildung 126: Einstellungen DMA für Receiver in STM32CubeMX

Die wichtigste Einstellung ist, dass der Modus als *Circular* festgelegt wird. Dadurch wird der DMA-Stream automatisch neu gestartet, wenn dieser den Einlesevorgang abgeschlossen hat. Dies ermöglicht ein durchgehendes Einlesen.

```

Dateiname: receiver.c

/** 
 * @brief This function calibrates and starts uart receive dma with selected protocol
 * @param proto protocol to use (SBUS / IBUS)
 * @param huart pointer to a UART_HandleTypeDef structure (input u(s)art)
 * @return Receiver_Status
 */
Receiver_Status Receiver_Init(Receiver_Protocol proto, UART_HandleTypeDef *huart)
{
    receiver_InputUART = huart;           // set input uart
    receiver_SelectedProtocol = proto;   // set serial protocol

    // set custom reception complete ISR
    HAL_UART_RegisterCallback(receiver_InputUART, HAL_UART_RX_COMPLETE_CB_ID,
    Receiver_ReceptionCallback);

    switch(receiver_SelectedProtocol)
    {
        /**
         * 115200 baud
         * 8 data bits, 1 stop bit, no parity
         * LSB first, not inverted
         * 32 Bytes:
         *      Byte[0]: protocol length, 0x20
         *      Byte[1]: command code, 0x40
         *      Byte[2-29]: channel data, 14 channels, 2 byte each, little endian
         *      Byte[30-31]: checksum, little endian, 0xFFFF - sum of other 30 bytes = checksum
         */
        case IBUS:
        {
            // check if uart is configured via baudrate
            if(receiver_InputUART->Init.BaudRate != 115200)
                return RECEIVER_UART_ERROR;

            // check if transmitter is connected (ppm signal reception)
            uint8_t timeout = 0;
            int8_t tmp_PinState = HAL_GPIO_ReadPin(RECEIVER_PPM_GPIO_Port, RECEIVER_PPM_Pin);
            while(tmp_PinState == HAL_GPIO_ReadPin(RECEIVER_PPM_GPIO_Port, RECEIVER_PPM_Pin))
            {
                // if the ppm signal doesn't change in 10ms -> error
                if(timeout++ > 10)
                    return RECEIVER_PPM_ERROR;
                HAL_Delay(1);
            }

            uint8_t tmp[2] = {0};
            timeout = 0;

            // calibrate reception to begin of protocol
            while(!(tmp[0] == 0x20 && tmp[1] == 0x40))

```

```

{
    // if the header is wrong 100x -> error
    if(timeout++ > 100)
        return RECEIVER_TIMEOUT;

    HAL_UART_Receive(receiver_InputUART, tmp, 2, 3);
}

HAL_Delay(4); // wait to sync to next data packet

// start DMA read i.bus signal
if(HAL_UART_Receive_DMA(receiver_InputUART, receiver_RawData, 32) != HAL_OK)
    return IBUS_ERROR;

// set min/max values of receiver input data
receiver_InputLimits.min = 1070;
receiver_InputLimits.max = 1920;
break;
}

/***
 * 100000 baud
 * 9 data bits, 2 stop bit, even parity
 * LSB first, inverted
 * 25 Bytes:
 *      Byte[0]: protocol header, 0x0F
 *      Byte[1-22]: channel data, 16 channels, 11 bits each
 *      Byte[23]:
 *          bit[4]: signal failsafe flag
 *          bit[5]: signal lost flag
 *          bit[6]: digital channel 18
 *          bit[7]: digital channel 17
 *      Byte[24]: protocol footer, 0x00
 */
case SBUS:
{
    // check if uart is configured via baudrate
    if(receiver_InputUART->Init.BaudRate != 100000)
        return RECEIVER_UART_ERROR;

    // check if transmitter is connected (ppm signal reception)
    uint8_t timeout = 0;
    int8_t tmp_PinState = HAL_GPIO_ReadPin(RECEIVER_PPM_GPIO_Port, RECEIVER_PPM_Pin);
    while(tmp_PinState == HAL_GPIO_ReadPin(RECEIVER_PPM_GPIO_Port, RECEIVER_PPM_Pin))
    {
        // if the ppm signal doesn't change in 10ms -> error
        if(timeout++ > 10)
            return RECEIVER_PPM_ERROR;
        HAL_Delay(1);
    }
}

```

```

    uint8_t tmp = 0;
    timeout = 0;

    // calibrate reception to begin of protocol
    while(tmp != 0x0F)
    {
        // if the header is wrong 100x -> error
        if(timeout++ > 100)
            return RECEIVER_TIMEOUT;

        HAL_UART_Receive(receiver_InputUART, &tmp, 1, 4);
    }
    HAL_Delay(4); // wait to sync to next data packet

    // start DMA read s.bus signal
    if(HAL_UART_Receive_DMA(receiver_InputUART, receiver_RawData, 25) != HAL_OK)
        return SBUS_ERROR;

    // set min/max values of receiver input data
    receiver_InputLimits.min = 350;
    receiver_InputLimits.max = 1680;
    break;
}

// wrong or no protocol selected
case NO_PROTO:
default:
    return PROTOCOL_ERROR;
break;
}

// set value range and half value
receiver_InputLimits.delta = receiver_InputLimits.max - receiver_InputLimits.min;
receiver_InputLimits.half = (receiver_InputLimits.max + receiver_InputLimits.min) / 2;

return RECEIVER_OK;
}

```

In den Programm *Receiver_Init()* wird je nach ausgewählten Empfangsprotokoll (SBUS oder IBUS) die UART-Peripherie mit den Datenpaketen synchronisiert und dann mit dem DMA-Controller durchgehend eingelesen.

Der Receiver schickt auch bei Verbindungsverlust alle 8ms ein Datenpaket. Bei Empfangsabschluss wird ein transfer complete Interrupt ausgelöst. Die Interrupt Service Routine ist mit der Funktion *Receiver_ReceptionCallback()* realisiert. In dieser Funktion erfolgt die gesamte Echtzeitregelung der Drohne (siehe: [Kapitel 5.4.4](#)).

5.4.4 Empfangssoftware

Wenn der DMA-Controller ein Packet fertig eingelesen hat, wird ein Interrupt mit der Funktion *Receiver_ReceptionCallback()* aufgerufen:

```
Dateiname: receiver.c

/*
 * @brief This function is the ISR for DMA receiver reception complete (called every 8ms)
 * @details all data gets decoded, PID updated and data send to groundstation
 * @param huart
 */
void Receiver_ReceptionCallback(UART_HandleTypeDef *huart)
{
    *****
    ----- part #1: receiver input -----
    *****
    uint8_t errorCode;
    float throttle = 0, pitch = 0, roll = 0, yaw = 0;

    errorCode = Receiver_Decode(); // decode raw data to channel data

    // check for decode errors
    if(errorCode != RECEIVER_OK)
    {
        // check connection lost
        if(errorCode == IBUS_SIGNAL_LOST_ERROR || errorCode == SBUS_SIGNAL_LOST || errorCode == SBUS_SIGNAL_FAILSAFE)
            Receiver_FailsafeHandler();

        // output error code
        sprintf(txt, "Receiver Error %d\n\r", errorCode);
        Terminal_Print(txt);
    }
    else
    {
        *****
        ----- part #2: motor control -----
        *****
        // convert to throttle speed and stick positions to angles
        Receiver_ConvertInput(throttle, pitch, roll, yaw);

        // update target values
        PID_Update(throttle, pitch, roll, yaw);
    }

    *****
    ----- part #3: get IMU data -----
    *****
    IMU_GetAngles();      // get pitch, roll, yaw
    IMU_BARO_ReadBaro(); // get altitude
```

```

***** part #4: data transmission to groundstation *****
***** ****
static int8_t dataTransmitDelay = 0;

// insert time delay between packets
if(dataTransmitDelay++ >= 60)
{
    static int8_t packetSelect = 0;

    // check what packet to send
    if(packetSelect == 0)
        DATA_TRANSMISSION_1(ds2438_Voltage, baroAltitude, 0x00);
    else
        DATA_TRANSMISSION_2(angle.pitch, angle.roll, angle.yaw);

    packetSelect = packetSelect == 0;
    dataTransmitDelay = 0;
}
}

```

Teil #1 – receiver input, Receiver_Decode():

Zu Beginn werden die rohen Empfangsdaten (Joystick- und Schalterwerte) mit der Funktion *Receiver_Decode()* in Kanalwerte dekodiert und in den globalen Array *receiver_ChData[]* gespeichert.

Dateiname: receiver.c
<pre> /** * @brief This function decodes the receiver raw data depending on the protocol * @details * i.bus channel values from 1070 - 1920 * s.bus channel values from 350 - 1680 * * what channel does what depends on the defines found in receiver.h: * - RECEIVER_YAW_CHANNEL * - RECEIVER_PITCH_CHANNEL * - RECEIVER_THROTTLE_CHANNEL * - RECEIVER_ROLL_CHANNEL * - RECEIVER_ONOFF_SWITCH_CHANNEL * - RECEIVER_MODESEL_SWTICH_CHANNEL * @return Receiver_Status */ Receiver_Status Receiver_Decode(void) { switch(receiver_SelectedProtocol) { case IBUS: { // if reception input start at the last byte -> reorder for correct order (rotate left) if(receiver_RawData[1] == 0x20 && receiver_RawData[2] == 0x40) </pre>

```

{
    uint8_t tmp = receiver_RawData[0];
    for(int8_t i = 1; i < 32; i++)
        receiver_RawData[i - 1] = receiver_RawData[i];
    receiver_RawData[31] = tmp;
}

// check if protocol header is correct
if(receiver_RawData[0] != 0x20 || receiver_RawData[1] != 0x40)
    return IBUS_HEADER_ERROR;

// check if checksum is correct (0xFFFF - sum of other 30 bytes = checksum)
uint16_t sum = 0;
for(int8_t i = 0; i < 30; i++)
    sum += receiver_RawData[i];

uint16_t checksum = (receiver_RawData[31] << 8) | receiver_RawData[30];
if((0xFFFF - sum) != checksum)
    return IBUS_CHECKSUM_ERROR;

// decode channel data (14 channels, 2 bytes each, little endian byte order)
for(int8_t i = 0, j = 0; i < 14; i++, j += 2)
    receiver_ChData[i] = (receiver_RawData[j + 3] << 8) | receiver_RawData[j + 2];

// check disconnection
Receiver_IBusFailsafeCheck();
if(receiver_SameDataCounter > 250)
    return IBUS_SIGNAL_LOST_ERROR;

break;
}

case SBUS:
{
// if reception input start at the last byte -> reorder for correct order (rotate left)
if(receiver_RawData[0] == 0x00 && receiver_RawData[1] == 0x0F)
{
    uint8_t tmp = receiver_RawData[0];
    for(int8_t i = 1; i < 25; i++)
        receiver_RawData[i - 1] = receiver_RawData[i];
    receiver_RawData[24] = tmp;
}

// check if protocol header is correct
if(receiver_RawData[0] != 0x0F)
    return SBUS_HEADER_ERROR;

// check if protocol footer is correct
if(receiver_RawData[24] != 0x00)
    return SBUS_FOOTER_ERROR;
}

```

```

// check signal lost flags
if(receiver_RawData[23] & 0x04)
    return SBUS_SIGNAL_LOST;

// check signal failsafe flag
if(receiver_RawData[23] & 0x08)
    return SBUS_SIGNAL_FAILSAFE;

// decode channel data (16 channels, 11 bits each, lsb first)
for(int8_t i = 0, j = 0; i < 16; i += 8, j += 11)
{
    receiver_ChData[i + 0] = ((receiver_RawData[j + 1] >> 0) | (receiver_RawData[j + 2]
<< 8)) & 0x7FF;
    receiver_ChData[i + 1] = ((receiver_RawData[j + 2] >> 3) | (receiver_RawData[j + 3]
<< 5)) & 0x7FF;
    receiver_ChData[i + 2] = ((receiver_RawData[j + 3] >> 6) | (receiver_RawData[j + 4]
<< 2) | (receiver_RawData[5] << 10)) & 0x7FF;
    receiver_ChData[i + 3] = ((receiver_RawData[j + 5] >> 1) | (receiver_RawData[j + 6]
<< 7)) & 0x7FF;
    receiver_ChData[i + 4] = ((receiver_RawData[j + 6] >> 4) | (receiver_RawData[j + 7]
<< 4)) & 0x7FF;
    receiver_ChData[i + 5] = ((receiver_RawData[j + 7] >> 7) | (receiver_RawData[j + 8]
<< 1) | (receiver_RawData[9] << 9)) & 0x7FF;
    receiver_ChData[i + 6] = ((receiver_RawData[j + 9] >> 2) | (receiver_RawData[j +
10] << 6)) & 0x7FF;
    receiver_ChData[i + 7] = ((receiver_RawData[j + 10] >> 5) | (receiver_RawData[j +
11] << 3)) & 0x7FF;
}

break;
}

// wrong or no protocol selected
case NO_PROTO:
default:
    return PROTOCOL_ERROR;
    break;
}

return RECEIVER_OK;
}

```

Diese Funktion wandelt die empfangenen Daten in S.Bus/I.Bus Kanalwerte um. Es wird kontrolliert, ob keine Protokollfehler, wie zum Beispiel Header- oder Footer-Error, vorkommen. Im Fall eines Fehlers wird der dazugehörige Fehlercode von der Funktion zurückgegeben – siehe: [Kapitel 5.1.1.1](#)

Da das I.Bus-Protokoll keine Failsafe-Flags besitzt, wird der Verbindungsstatus mit der Funktion *Receiver_IBusFailsafeCheck()* überprüft.

```
Dateiname: receiver.c

/***
 * @brief This function saves the current channel data and check if its the same as before
 * @param huart pointer to UART_HandleTypeDef
 * @retval None
 */
void Receiver_IBusFailsafeCheck(void)
{
    // only used for IBUS because SBUS does have a signal lost / failsafe flag
    if(receiver_SelectedProtocol != IBUS)
        return;

    static uint16_t receiver_OldChData[16] = {0};    // previous channel data

    // when off -> don't check
    if(receiver_ChData[RECEIVER_ONOFF_SWITCH_CHANNEL] < receiver_InputLimits.half)
    {
        receiver_SameDataCounter = 0; // reset counter
    }
    else
    {
        int8_t same = 1; // data is same flag
        for(int8_t i = 0; i < 14 && same == 1; i++)
        {
            // check if the old channel data is not the same as the current
            if(receiver_OldChData[i] != receiver_ChData[i])
            {
                receiver_SameDataCounter = 0; // reset channel data check
                same = 0;
            }

            // increment receiver_SameDataCounter when data is the same
            else if(i == 14 - 1)
                (receiver_SameDataCounter == UINT16_MAX - 10) ? receiver_SameDataCounter = 260 :
                receiver_SameDataCounter++;
        }
    }

    // save current channel data
    for(int8_t i = 0; i < 14; i++)
        receiver_OldChData[i] = receiver_ChData[i];
}
```

Im Falle des Verbindungsverlust sendet der Receiver durchgehend die exakt selben Kanalwerte. Daher zählt diese Funktion, wie oft das exakt selbe Paket mit denselben Werten gesendet worden ist. Dafür wird der Zähler *receiver_SameDataCounter* verwendet. Wenn der Wert über 250 ist, liefert das Dekodierungsprogramm *Receiver_Decode()* einen Failsafe-Error.

Teil #2 – motor control, Receiver_ConvertInput():

Die dekodierten Joystickpositionen (Kanalwerte) werden dann mit der Funktion *Receiver_ConvertInput()* in Prozentwerte für Throttle und in Gradwerte für Pitch, Roll und Yaw umgewandelt. Diese sind in weiteren Schritten für den PID-Regler wichtig.

Dateiname: receiver.c

```
/*
 * @brief This function converts the input from the receiver to thorttle and angles
 * @details
 * The max throttle values per mode can be changed in receiver.h with:
 * - ESC_SAFEMODE_THR_MAX
 * - ESC_NORMALMODE_THR_MAX
 * - ESC_OFFMODE_THR
 * - ESC_TURN_OFFSET_MAX
 * @param throttle percent of throttle speed
 * @param pitch stick position in degrees
 * @param roll stick position in degrees
 * @param yaw stick position in degrees
 * @return None
 */
void Receiver_ConvertInput(float throttle, float pitch, float roll, float yaw)
{
    // turn red LED off
    __HAL_TIM_SET_COMPARE(LED_TIM, LED_RED_CHANNEL, 0);

    *****
    ----- check ON / OFF switch -----
    *****
    static uint8_t droneOffModeFlag = 1;

    // top position (< half) = turn drone off
    if(receiver_ChData[RECEIVER_ONOFF_SWITCH_CHANNEL] < receiver_InputLimits.half)
    {
        droneOffModeFlag = 1; // set off flag
        throttle = 0;        // turn motors off
    }

    // check if throttle stick in the lowest postiion after turning on/off switch to on
    else if(droneOffModeFlag == 1 && receiver_ChData[RECEIVER_THROTTLE_CHANNEL] >
receiver_InputLimits.min + 10)
    {
        throttle = 0; // turn motors off
    }

    // normal control
    else
    {
        droneOffModeFlag = 0;
    }
}
```

```

*****  

----- check 3 position switch (mode select) -----  

*****  

uint16_t esc_MaxThr = ESC_SAFEMODE_THR_MAX;  

uint8_t hoverModeFlag = 0;  
  

// top position (< half) = safemode  

if(receiver_ChData[RECEIVER_MODESEL_SWTICH_CHANNEL] < receiver_InputLimits.half - 10)  

    esc_MaxThr = ESC_SAFEMODE_THR_MAX; // set max throttle value  
  

// middle position (half +- 10) = normalmode  

else if(receiver_ChData[RECEIVER_MODESEL_SWTICH_CHANNEL] >= receiver_InputLimits.half -  
10 && receiver_ChData[RECEIVER_MODESEL_SWTICH_CHANNEL] <= receiver_InputLimits.half + 10)  

    esc_MaxThr = ESC_NORMALMODE_THR_MAX; // set max throttle value  
  

// down position = extra mode hover mode  

else  

    hoverModeFlag = 1;  
  

*****  

----- calculate throttle input (up / down) -----  

*****  

// get joystick position  

throttle = (float)(receiver_ChData[RECEIVER_THROTTLE_CHANNEL] -  
receiver_InputLimits.min) / receiver_InputLimits.delta;  

throttle *= esc_MaxThr; // get real thorttle value  
  

*****  

----- calculate pitch input (forwards / backwards) -----  

*****  

// get joystick position  

pitch = (float)(receiver_ChData[RECEIVER_PITCH_CHANNEL] - receiver_InputLimits.min) /  
receiver_InputLimits.delta;  

pitch = (pitch - 0.5f) * 2; // check forwards or backwards position  

pitch *= ESC_TURN_OFFSET_MAX; // get stick position in degrees  
  

*****  

----- calculate roll input (left / right) -----  

*****  

// get joystick position  

roll = (float)(receiver_ChData[RECEIVER_ROLL_CHANNEL] - receiver_InputLimits.min) /  
receiver_InputLimits.delta;  

roll = (roll - 0.5f) * 2; // check left or right position  

roll *= ESC_TURN_OFFSET_MAX; // get stick position in degrees

```

```

***** -----
----- calculate yaw input (rotate left / rotate right) -----
***** /***** /****

// get joystick position
yaw = (float)(receiver_ChData[RECEIVER_YAW_CHANNEL] - receiver_InputLimits.min) /
receiver_InputLimits.delta;
yaw = (yaw - 0.5f) * 2; // check left or right position

***** -----
----- check values -----
***** /***** /****

// check if hovermode
if(hoverModeFlag == 1)
{
    pitch = 0;
    roll = 0;
    yaw = 0;
}
}

}

```

Die berechneten Werte (Throttle, Pitch, Roll, Yaw) werden mit call-by-reference wieder zurückgeschickt.

Wichtig: Diese Funktion berechnet nur die Werte und aktualisiert den PID-Regler nicht. Um die Sollwerte auf den aktuellen Stand zu bringen, muss die Funktion *PID_Update()* aufgerufen werden (siehe: [Kapitel 5.8.2](#)).

Teil #3 – get IMU data, IMU_GetAngles() + IMU_BARO_GetBaro():

Für die Erklärung und Funktionsweise der Befehle

– siehe: [Kapitel 5.5.3.4, 5.5.4.3, 5.5.4.4](#)

Teil #4 – data transmission to groundstation, DATA_TRANSMISSION_1/2():

Für die Erklärung und Funktionsweise der Befehle

– siehe: [Kapitel 6.2.5](#)

5.5 Inertial Measurement Unit (IMU)

Inertial Measurement Unit (IMU) ist ein Sammelbegriff für alle Sensoren, die die Kräfte auf einen Körper messen. Typischerweise besteht ein IMU aus einem Gyroskop und einen Beschleunigungsmesser (Accelerometer). In komplexeren Systemen sind weitere Sensoren, wie ein Kompass (Magnetometer) und ein Luftdrucksensor (Barometer) zusätzlich vorhanden.

Der Sensor wird meistens mit dem Begriff „Degrees Of Freedom / DOF“ beschrieben. Dieser Parameter gibt an, wie viele Achsen der Sensor messen kann.

Das heißt, dass zum Beispiel ein 6DOF-IMU, der aus einem Gyroskop und einen Accelerometer besteht, um sechs Achsen messen kann – pro Sensor die x-, y- und z-Achse.

Für die FPV-Drohne wird ein 10DOF-IMU mit einem Gyroskop, Accelerometer, Barometer und Magnetometer verwendet, die auf dem IMU-Breakout vorhanden sind. Mit diesen Sensoren werden die Lagewinkel, die Flughöhe, Temperatur und Regelgrößen für PID-Regler gemessen.

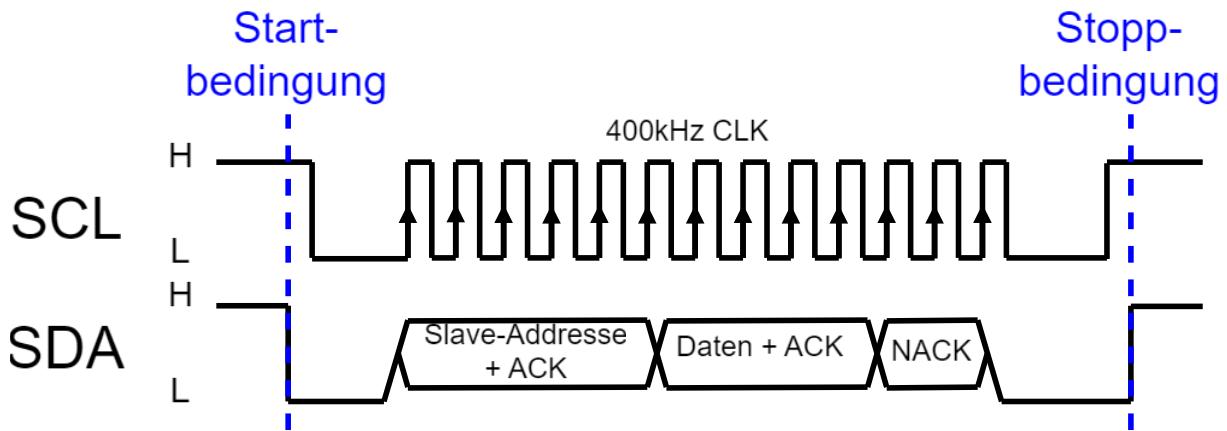
5.5.1 I²C Protokoll

Das IMU-Breakout auf der FPV-Drohne kann nur mit dem I²C-Protokoll angesprochen werden. Dieses Protokoll besteht aus drei Leitungen: SDA (Datenleitung), SCL (Taktleitung) und GND (Massoleitung) und basiert auf dem Master-Slave-Prinzip.

Um die Datenübertragung zu starten, muss eine Startbedingung erfüllt werden: während die SCL-Leitung im high-Ruhezustand ist, wechselt die SDA-Leitung auf einen low-Zustand.

Nach der Startbedingung schickt der Master im System (Flight Controller) ein 400kHz Rechtecksignal mit einem Duty Cycle von 50% auf die SCL-Leitung. Zuerst wird auf der SDA-Leitung wird die I²C-Adresse des I²C-Slaves geschickt. Danach werden gleichzeitig die Daten gesendet beziehungsweise empfangen und bei einer steigenden Flanke der SCL-Leitung interpretiert. Nach jedem Datenbyte schickt der Empfänger der Daten ein Acknowledge-Bit (ACK / 0), um den Empfang der Daten zu bestätigen. Wenn das Acknowledge-Bit einen high-Pegel hat (not Acknowledge, NACK), trat bei der Übertragung ein Fehler auf.

Am Ende jeder Übertragung wird ein not-Acknowledge-Bit (NACK / 1) empfangen. Darauf folgt eine Stoppbedingung, die eine zeitlich invertierte Startbedingung ist. Während die SCL-Leitung sich in den high-Ruhezustand befindet, wechselt die SDA-Leitung von einem low-Zustand auch in den high-Ruhezustand.

Abbildung 127: I²C Datentransfer**I²C Einstellung in STM32CubeMX:**

The screenshot shows the STM32CubeMX software interface for configuring the I²C1 port. The left sidebar lists various peripheral components. The 'Connectivity' section is expanded, showing options like FDCAN1, FDCAN2, I2C1, I2C2, I2C3, I2C4, LPUART1, MDIOS, OCTOSPI1, OCTOSPI2, and SDMMC1. The 'I2C1' option is selected and highlighted. The main configuration window is titled 'I2C1 Mode and Configuration' under the 'Mode' tab, with 'I2C [I2C]' selected. In the 'Configuration' tab, the 'Parameter Settings' section is active. A specific configuration for 'Timing configuration' is highlighted with an orange border, showing 'I2C Speed Mode' set to 'Fast Mode' and 'I2C Speed Frequency (KHz)' set to '400'. To the right, two green pins are labeled 'IMU_SDA' and 'IMU_SCL', corresponding to PB7 and PB6 respectively.

Abbildung 128: STM32CubeMX Einstellungen IMU

Um die schnellste Kommunikation mit dem IMU-Breakout festzulegen, muss die Einstellung *I²C Speed Mode* auf *Fast Mode* und *I²C Speed Frequency (KHz)* auf 400 gesetzt werden.

Um zwischen mehreren I²C Geräten zu unterscheiden, hat jedes Gerät eine eigene I²C-Adresse. Die Adresse besteht aus einer 7-Bit-Zahl. Während der Kommunikation muss nach der Sensoradresse ein achtes Bit (R/W Bit) zur Adresse hinzugefügt werden, welches bei Schreibzyklen auf 0 gesetzt und bei Lesezyklen auf 1 gesetzt werden muss.

Sensor	Slave Adressen + R/W Bit	
	Schreiben	Lesen
MPU9250	0xD0	0xD1
BMP280	0xEE	0xEF
AK8963	0x18	0x19

Damit mit den Sensoren des IMUs kommuniziert werden kann, müssen bestimmte Lese- und Schreibzyklen eingehalten werden:

5.5.1.1 Schreibzyklus IMU

Master	S	AD+W		RA		DATA		P
Slave			ACK		ACK		ACK	

Abbildung 129: I²C Schreibzyklus

Signal	Beschreibung
S	Startbedingung
AD+W	Slave-Adresse + Write Bit
ACK	Acknowledge Bit
RA	Register Adresse
P	Stoppbedingung

Es können mehrere Byte direkt hintereinander geschickt werden. Dabei können beliebig viele Datenbyte mit Acknowledge-Bit vor der Stoppbedingung gesendet werden. Ein interner Zeiger erhöht die Register Adresse um eine Stelle pro Datenbyte.

Schreiben zu IMU-Register:

```
Dateiname: IMU_10DOF.c

/** 
 * @brief This function writes an amount of bytes to registers from the IMU
 * @param sensor MPU9250, AK8963 (MAG), BMP280 (BARO)
 * @param regAddr register address
 * @param data data to write
 * @return IMU_Status
 */
IMU_Status IMU_WriteRegister(IMU_Sensor sensor, uint8_t regAddr, uint8_t data)
{
    // determine the I2C device address
    uint16_t devAddress;
    switch(sensor)
    {
        case MPU9250:
            devAddress = IMU_MPU_I2C_ADDR;
            break;

        case AK8963:
            devAddress = IMU_MAG_I2C_ADDR;
    }
}
```

```

        break;

    case BMP280:
        devAddress = IMU_BARO_I2C_ADDR;
        break;

    default:
        return IMU_ADDRESS_ERROR;
    }

    // read register(s)
    HAL_I2C_Mem_Write(imu_ComI2C, devAddress, regAddr, I2C_MEMADD_SIZE_8BIT, &data, 1, 1000);

    return IMU_OK;
}

```

Die Funktion *HAL_I2C_Mem_Write()* sendet automatisch die angegebenen Daten und Befehle an den I²C-Peripheriebaustein, die für die Kommunikation notwendig sind. Wenn nach einer Sekunde die Kommunikation noch nicht abgeschlossen ist, liefert die Funktion einen Timeout-Error.

5.5.1.2 Lesezyklus IMU

Master	S	AD+W		RA		S	AD+R			NACK	P
Slave			ACK		ACK			ACK	DATA		

Abbildung 130: I²C Lesezyklus

Signal	Beschreibung
S	Startbedingung
AD+W	Slave-Adresse + Write Bit
ACK	Acknowledge Bit
RA	Register Adresse
AD+R	Slave-Adresse + Read Bit
NACK	not Acknowledge Bit
P	Stopptbedingung

Wie beim Schreibablauf können auch mehrere Byte direkt hintereinander eingelesen werden. Dabei wird die Kommunikation mit NACK vom Master im System beendet. Ein interner Zeiger erhöht die Register Adresse um eine Stelle pro Datenbyte.

Lesen von IMU-Register/n:

```
Dateiname: IMU_10DOF.c

/** 
 * @brief This function reads an amount of bytes from registers from the IMU
 * @param sensor MPU9250, AK8963 (MAG), BMP280 (BARO)
 * @param regAddr register address
 * @param data data pointer
 * @param rxBytes amount of bytes to read
 * @return IMU_Status
 */
IMU_Status IMU_ReadRegister(IMU_Sensor sensor, uint8_t regAddr, uint8_t *data, uint8_t rxBytes)
{
    // determine the I2C device address
    uint16_t devAddress;
    switch(sensor)
    {
        case MPU9250:
            devAddress = IMU_MPU_I2C_ADDR;
            break;

        case AK8963:
            devAddress = IMU_MAG_I2C_ADDR;
            break;

        case BMP280:
            devAddress = IMU_BARO_I2C_ADDR;
            break;

        default:
            return IMU_ADDRESS_ERROR;
    }

    // read register(s)
    HAL_I2C_Mem_Read(imu_ComI2C, devAddress, regAddr, I2C_MEMADD_SIZE_8BIT, data, rxBytes,
1000);

    return IMU_OK;
}
```

Die Funktion `HAL_I2C_Mem_Read()` sendet automatisch die Befehle an den I²C-Peripheriebaustein, die für die Kommunikation notwendig sind, und speichert die Daten in der angegeben Variable. Wenn nach einer Sekunde die Kommunikation noch nicht abgeschlossen ist, liefert die Funktion einen Timeout-Fehler.

5.5.2 IMU-Verbindungstest

Alle drei Sensoren haben ein Who-Am-I-Register. Diese Register haben festgelegte Werte, die nicht geändert werden können.

Es werden alle Sensoren überprüft, obwohl der Magnetometer nicht verwendet wird, damit eine vollständige Verbindungsüberprüfung mit dem gesamten IMU-Breakout durchgeführt werden kann.

Sensor	Who Am I Register Adresse	Registerinhalt
MPU9250	0x75	0x71
BMP280	0xD0	0x58
AK8963	0x00	0x48

Wenn in den angegebenen Registern nicht die vorgegebenen Werte stehen besteht ein Problem bei dem Datenaustausch mit dem I²C-Bus oder der Sensor ist beschädigt.

Verbindungstest aller IMU-Sensoren:

```
Dateiname: IMU_10DOF.c
/***
 * @brief This function checks the connection of all sensors on the IMU
 * @attention This function enables the bypass mode in the MPU9250
 * @return IMU_Status
 */
IMU_Status IMU_CheckConnection(void)
{
    uint8_t regVal[3] = {0x71, 0x48, 0x58};
    uint8_t regAddr[3] = {IMU_MPU_WHOAMI_ADDR, IMU_MAG_WHOAMI_ADDR, IMU_BARO_CHIPID_ADDR};
    uint8_t sensor[3] = {MPU9250, MAG, BARO};
    uint8_t timeout;
    uint8_t data = 0x00;

    for(uint8_t i = 0; i < 3; i++)
    {
        timeout = 0;
        while(data != regVal[i])
        {
            if(IMU_ReadRegister(sensor[i], regAddr[i], &data, 1) != IMU_OK)
                return IMU_ADDRESS_ERROR;

            if(timeout++ > 100)
                return IMU_MPU_WHOAMI_ERROR + i;
        }

        if(sensor[i] == MPU9250)
        {
            // enable bypass mode
            if(IMU_WriteRegister(MPU9250, IMU_MPU_INT_PIN_CFG_ADDR, 0x02) != IMU_OK)
                return IMU_ADDRESS_ERROR;
            IMU_DelayUs(1000);
        }
    }
}
```

```
    }
    return IMU_OK;
}
```

Damit auf das Magnetometer zugegriffen werden kann, muss im MPU9250 im bypass enable Register das BYPASS_EN – Bit gesetzt werden.

Dadurch werden die I²C-Leitung durch den Sensor durchgeführt, und es kann direkt auf das Magnetometer zugegriffen werden.

Wenn das Bit nicht gesetzt wird, müssen im MPU9250 komplexe Einstellungen gesetzt werden, damit er als I²C-Master zu externen Sensoren fungieren kann.

Wenn nach 100 Lesezyklen der Registerwert immer noch falsch ist, liefert die Funktion einen *IMU_<Sensor>_WHOAMI_ERROR*.

5.5.3 Bestimmen der Lagewinkel - MPU9250

5.5.3.1 Registerübersicht MPU9250

In der folgenden Abbildung wird die Registerübersicht der verwendeten Registern des MPU9250 dargestellt, auf die mit der I²C-Adresse 0x68 zugegriffen werden kann:

Addr (Hex)	Addr (Dec.)	Register Name	Serial I/F	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
19	25	SMPLRT_DIV	R/W	SMPLRT_DIV[7:0]							
1A	26	CONFIG	R/W	-	FIFO_MODE	EXT_SYNC_SET[2:0]			DLPF_CFG[2:0]		
1B	27	GYRO_CONFIG	R/W	XGYRO_Ct_en	YGYRO_Ct_en	ZGYRO_Ct_en	GYRO_FS_SEL [1:0]		-	FCHOICE_B[1:0]	
1C	28	ACCEL_CONFIG	R/W	ax_st_en	ay_st_en	az_st_en	ACCEL_FS_SEL[1:0]				
1D	29	ACCEL_CONFIG 2	R/W					ACCEL_FCHOICE_B		A_DLPF_CFG	
37	55	INT_PIN_CFG	R/W	ACTL	OPEN	LATCH_INT_EN	INT_ANYRD_2CLEAR	ACTL_FSYNC	FSYNC_INT_MODE_EN	BYPASS_EN	-
3B	59	ACCEL_XOUT_H	R	ACCEL_XOUT_H[15:8]							
3C	60	ACCEL_XOUT_L	R	ACCEL_XOUT_L[7:0]							
3D	61	ACCEL_YOUT_H	R	ACCEL_YOUT_H[15:8]							
3E	62	ACCEL_YOUT_L	R	ACCEL_YOUT_L[7:0]							
3F	63	ACCEL_ZOUT_H	R	ACCEL_ZOUT_H[15:8]							
40	64	ACCEL_ZOUT_L	R	ACCEL_ZOUT_L[7:0]							
43	67	GYRO_XOUT_H	R	GYRO_XOUT_H[15:8]							
44	68	GYRO_XOUT_L	R	GYRO_XOUT_L[7:0]							
45	69	GYRO_YOUT_H	R	GYRO_YOUT_H[15:8]							
46	70	GYRO_YOUT_L	R	GYRO_YOUT_L[7:0]							
47	71	GYRO_ZOUT_H	R	GYRO_ZOUT_H[15:8]							
48	72	GYRO_ZOUT_L	R	GYRO_ZOUT_L[7:0]							
6B	107	PWR_MGMT_1	R/W	H_RESET	SLEEP	CYCLE	GYRO_STANDBY	PD_PTAT	CLKSEL[2:0]		
6C	108	PWR_MGMT_2	R/W	-	DIS_XA	DIS YA	DIS_ZA	DIS_XG	DIS_YG	DIS_ZG	
75	117	WHO_AM_I	R	WHOAMI[7:0]							

Abbildung 131: Registerübersicht MPU9250

Register	Verwendungszweck
SMPLRT_DIV	dividiert die interne Abtastrate
CONFIG	digitaler Tiefpassfilter für Gyroskop
GYRO_CONFIG	Messbereich für Gyroskop
ACCEL_CONFIG	Messbereich für Accelerometer
ACCEL_CONFIG 2	digitaler Tiefpassfilter für Accelerometer
INT_PIN_CFG	direkter I ² C-Zugriff auf Magnetometer
ACCEL_<Achse>OUT_H/L	Messwerte von Accelerometer <Achse>: X/Y/Z - Messachse H/L: hoch- oder niedervwertiges Byte
GYRO_<Achse>OUT_H/L	Messwerte von Gyroskop <Achse>: X/Y/Z - Messachse H/L: hoch- oder niedervwertiges Byte
PWR_MGMT_1	Sensor reset und Taktquelle
PWR_MGMT_2	Sensoren aktivieren
WHO_AM_I	Überprüfung der Verbindung

5.5.3.2 Initialisierung Accelerometer und Gyroskop

```
Dateiname: IMU_10DOF.c | Ausschnitt aus IMU_Init() Funktion

/** 
 * @brief This function initializes the 10DOF IMU (accel, gyro, mag, baro)
 * @param imuInit pointer to IMU_InitTypeDef
 * @return IMU_Status
 */
IMU_Status IMU_Init(IMU_InitTypeDef *imuInit)
{
    ...

    // reset MPU
    IMU_WriteRegister(MPU9250, IMU_MPU_PWR_MGMT_1_ADDR, 0x00);
    IMU_DelayUs(10000);
    // auto select best clk source
    IMU_WriteRegister(MPU9250, IMU_MPU_PWR_MGMT_1_ADDR, 0x01);
    // enable gyro and accel
    IMU_WriteRegister(MPU9250, IMU_MPU_PWR_MGMT_2_ADDR, 0x00);
    // select full scale range for gyro and accel
    IMU_WriteRegister(MPU9250, IMU_MPU_ACCEL_CONFIG_ADDR, imuInit->accelFS << 3);
    IMU_WriteRegister(MPU9250, IMU_MPU_GYRO_CONFIG_ADDR, imuInit->gyroFS << 3);
    // select digital low pass filter for gyro and accel
    IMU_WriteRegister(MPU9250, IMU_MPU_ACCEL_CONFIG_2_ADDR, imuInit->accelDLPF);
    IMU_WriteRegister(MPU9250, IMU_MPU_CONFIG_ADDR, imuInit->gyroDLPF);
    // select fastest sample rate
    IMU_WriteRegister(MPU9250, IMU_MPU_SMPLRT_DIV_ADDR, 0x00);
    // calculate sensitivity scale factor (LSB/g and LSB/(°/s))
    accelSens = IMU_ACCEL_RES_MAX / (1 << imuInit->accelFS);
    gyroSens = IMU_GYRO_RES_MAX / (1 << imuInit->gyroFS);

    // calibrate gyro
    uint16_t amount = 2000;
    IMU_RegCoordinates tempGyro = {0};
    int32_t tempX = 0, tempY = 0, tempZ = 0;
    for(uint16_t i = 0; i < amount; i++)
    {
        tempGyro = IMU_MPU_ReadGyro();
        tempX += tempGyro.x;
        tempY += tempGyro.y;
        tempZ += tempGyro.z;
        IMU_DelayUs(3000);
    }
    gyroOffset.x = (float)tempX / (float)amount;
    gyroOffset.y = (float)tempY / (float)amount;
    gyroOffset.z = (float)tempZ / (float)amount;

    ...

    return IMU_OK;
}
```

Die Messwerte eines Gyroskops driften bei Bewegung in eine Richtung (siehe: [Kapitel 2.1.1.1](#)). Um dem dagegen zu wirken, wird ein Gyroskop Offset bestimmt. Es wird nach 2000 Messung der durchschnittliche Messwert bestimmt, der als Offsetwert verwendet wird. Im Programm wird das Ergebnis in der globalen Variable *gyroOffset* gespeichert.

5.5.3.3 Einlesen der Accelerometer- und Gyroskop-Daten

Die Messdaten des Accelerometers werden in den Registern 59 ACCEL_XOUT_H bis zum Register 64 ACCEL_ZOUT_L gespeichert. XOUT, YOUT und ZOUT bestimmt die einzelne Messachse und _H und _L bestimmt das high- und low-Byte des Messwertes.

Auslesen der Accelerometer-Daten:

```
Dateiname: IMU_10DOF.c

/**
 * @brief This function reads accelerometer register data (x,y,z)
 * @return IMU_RegCoordinates
 */
IMU_RegCoordinates IMU_MPU_ReadAccel(void)
{
    uint8_t buffer[6] = {0};
    IMU_ReadRegister(MPU9250, IMU_MPU_ACCEL_XOUT_H_ADDR, buffer, 6);

    IMU_RegCoordinates accelData = {0};
    accelData.x = ((int16_t)buffer[0] << 8) | buffer[1];
    accelData.y = ((int16_t)buffer[2] << 8) | buffer[3];
    accelData.z = ((int16_t)buffer[4] << 8) | buffer[5];

    return accelData;
}
```

In dem Programm werden die Daten beginnend mit dem Register 59 nacheinander bis zum Register 64 eingelesen und danach zusammengefügt.

Die Messung des Gyroskops funktioniert gleich wie beim Accelerometer. Die Messdaten werden in den Registern 67 GYRO_XOUT_H bis zum Register 72 GYRO_ZOUT_L gespeichert. XOUT, YOUT und ZOUT bestimmen die einzelne Messachse und _H und _L bestimmen das high- und low-Byte des Messwertes.

Auslesen der Gyroskop-Daten:

```
Dateiname: IMU_10DOF.c

/**
 * @brief This function reads gyroscope register data (x,y,z)
 * @return IMU_RegCoordinates
 */
IMU_RegCoordinates IMU_MPU_ReadGyro(void)
{
    uint8_t buffer[6] = {0};
    IMU_ReadRegister(MPU9250, IMU_MPU_GYRO_XOUT_H_ADDR, buffer, 6);

    IMU_RegCoordinates gyroData = {0};
    gyroData.x = ((int16_t)buffer[0] << 8) | (int16_t)buffer[1];
    gyroData.y = ((int16_t)buffer[2] << 8) | (int16_t)buffer[3];
    gyroData.z = ((int16_t)buffer[4] << 8) | (int16_t)buffer[5];

    return gyroData;
}
```

Bemerkung:

Bei beide Programmen werden nur die Registerwerte zurückgeliefert. Um die eigentlichen Messwerte zu bekommen, müssen die Werte mit dem sensitivity scale factoren des Accelerometers und des Gyroskops gerechnet werden. Diese Faktoren werden im Initialisierungsprogramm berechnet und in den globalen Variablen `accelSens` und `gyroSens` gespeichert (*siehe: [Kapitel 5.5.3.2](#)*).

Um die Drift vom Gyroskop entgegenzuwirken, wird der Registerwert noch mit dem Gyroskop-Offset gerechnet. Diese Werte werden auch im Initialisierungsprogramm bestimmt und in der globalen Variable `gyroOffset` gespeichert (*siehe: [Kapitel 5.5.3.2](#)*).

Berechnung der eigentlichen Messwerte:

```
IMU_RegCoordinates gyroData = IMU_MPUs_ReadGyro();
IMU_RegCoordinates accelData = IMU_MPUs_ReadAccel();

gyro.x = (gyroData.x - gyroOffset.x) / gyroSens;
gyro.y = (gyroData.y - gyroOffset.y) / gyroSens;
gyro.z = (gyroData.z - gyroOffset.z) / gyroSens;

accel.x = (accelData.x / accelSens) - 0.01f;
accel.y = (accelData.y / accelSens) - 0.02f;
accel.z = (accelData.z / accelSens) - 0.1f;
```

Die Offsetwerte bei der Beschleunigungsberechnung (0,01; 0,02 und 0,1) müssen händisch bestimmt werden. Wenn der Sensor auf einer ebenen Fläche gerade liegt, muss der Accelerometer eine Beschleunigung von 1g auf der z-Achse messen.

Der Offset ergibt sich aus der eigentlichen Messung auf einer ebenen Fläche von 1,01g. Dieser Vorgang muss mit 90° Drehungen für allen Achsen wiederholt werden, um die restlichen Offsetwerte zu bestimmen.

5.5.3.4 Berechnen der Lagewinkel

Um die Lagewinkel Pitch, Roll und Yaw zu bestimmen, wird ein Komplementärfilter auf die Messwerte (Erd-, Winkelbeschleunigung) des MPU9250 angewendet (siehe: [Kapitel 2.2.5](#)).

Anwendung des Komplementärfilters:

```
Dateiname: IMU_10DOF.c

/** 
 * @brief This function calculates pitch, roll and yaw
 * @details data gets stored in the global variable 'angle'
 * @retval None
 */
void IMU_GetAngles(void)
{
    // read counter value since last function call
    uint16_t tmpTime = __HAL_TIM_GET_COUNTER(imu_DelayTIM);
    // reset timer
    __HAL_TIM_SET_COUNTER(imu_DelayTIM, 0);

    // start first time call
    static int8_t firstTimeFlag = 0;
    if(firstTimeFlag == 0)
    {
        firstTimeFlag = 1;
        return;
    }

    // calculate delta time
    imu_DeltaTime = (float)tmpTime / 1E6f;
```

```

// read sensors
IMU_RegCoordinates gyroData = IMU_MPUMPU_ReadGyro();
IMU_RegCoordinates accelData = IMU_MPUMPU_ReadAccel();

// calculate actual values
gyro.x = (gyroData.x - gyroOffset.x) / gyroSens;
gyro.y = (gyroData.y - gyroOffset.y) / gyroSens;
gyro.z = (gyroData.z - gyroOffset.z) / gyroSens;

accel.x = (accelData.x / accelSens) - 0.01f;
accel.y = (accelData.y / accelSens) - 0.02f;
accel.z = (accelData.z / accelSens) - 0.1f;

// invert axis because the sensor is upside down
accel.z = -accel.z;

// apply complementary filter to calc angles
float accelPitch = atan2(accel.y, accel.z) * RAD2DEG;
float accelRoll = atan2(accel.x, accel.z) * RAD2DEG;

angle.roll = 0.98f * (angle.roll + gyro.y * imu_DeltaTime) + 0.02f * accelRoll;
angle.pitch = 0.98f * (angle.pitch - gyro.x * imu_DeltaTime) + 0.02f * accelPitch;
angle.yaw += gyro.z * imu_DeltaTime;
}

```

Die Funktion liest die Registerwerte vom Accelerometer und Gyroskop ein. Danach werden die eigentlichen Messwerte ausgerechnet. Mittels Komplementärfilter werden die Lagewinkel ausgerechnet, die in der Variable *angle* gespeichert werden.

Dadurch, dass der Sensor verkehrt auf der Drohne montiert ist, muss der Messwert um die z-Achse des Accelerometers invertiert werden.

5.5.4 Bestimmen der Höhe - BMP280

5.5.4.1 Registerübersicht BMP280

In der folgenden Abbildung wird die Registerübersicht, der verwendeten Registern des BMP280 dargestellt, auf die mit der I²C-Adresse 0x77 zugegriffen werden kann:

Register Name	Address	bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0	Reset state
temp_xlsb	0xFC		temp_xlsb<7:4>		0	0	0	0	0	0x00
temp_lsb	0xFB			temp_lsb<7:0>						0x00
temp_msb	0xFA				temp_msb<7:0>					0x80
press_xlsb	0xF9		press_xlsb<7:4>		0	0	0	0	0	0x00
press_lsb	0xF8			press_lsb<7:0>						0x00
press_msb	0xF7				press_msb<7:0>					0x80
config	0xF5	t_sb[2:0]			filter[2:0]		spi3w_en[0]			0x00
ctrl_meas	0xF4	osrs_t[2:0]			osrs_p[2:0]		mode[1:0]			0x00
status	0xF3			measuring[0]			im_update[0]			0x00
reset	0xE0			reset[7:0]						0x00
id	0xD0			chip_id[7:0]						0x58
calib25...calib00	0xA1...0x88			calibration data						individual

Registers:	Reserved registers	Calibration data	Control registers	Data registers	Status registers	Revision	Reset
Type:	do not write	read only	read / write	read only	read only	read only	write only

Abbildung 132: Registerübersicht BMP280

Register	Verwendungszweck
temp_xlsb	Temperatur Messwerte
temp_lsb/msb	
press_xlsb	Luftdruck Messwerte
press_lsb/msb	
config	Zeitkonstanten
ctrl_meas	Oversampling- und Modus-Auswahl
status	Status überprüfen
reset	Sensor zurücksetzen
id	Überprüfung der Verbindung
calib25...calib00	Kompensationsparameter auslesen

5.5.4.2 Initialisierung Barometer

Dateiname: IMU_10DOF.c Ausschnitt aus IMU_Init() Funktion
<pre>/** * @brief This function initializes the 10DOF IMU (accel, gyro, mag, baro) * @param imuInit pointer to IMU_InitTypeDef * @return IMU_Status */ IMU_Status IMU_Init(IMU_InitTypeDef *imuInit) { ... IMU_WriteRegister(BMP280, IMU_BARO_RESET_ADDR, 0xB6); // reset barometer // check if status bit for device = ready uint8_t timeout = 0, status = 1; while(status != 0x00) { IMU_ReadRegister(BMP280, IMU_BARO_STATUS_ADDR, &status, 1); // check 100 times max if(timeout++ > 100) return IMU_BARO_INIT_ERROR; } IMU_BARO_ReadCompensationValues(); // set standby time and time constant of IIR filter uint8_t config = ((imuInit->baroSBT << 5) (imuInit->baroCoeff << 2)); IMU_WriteRegister(BMP280, IMU_BARO_CONFIG_ADDR, config); // set oversampling settings for temperature and pressure measurement and set normal mode uint8_t ctrl = (imuInit->baroTempOS << 5) (imuInit->baroPressOS << 2) 0x03; IMU_WriteRegister(BMP280, IMU_BARO_CTRL_MEAS_ADDR, ctrl); // get current altitude level IMU_DelayUs(UINT16_MAX - 1); float baroSum = 0; for(uint16_t i = 0; i < amount; i++) { IMU_BARO_ReadBaro(); baroSum += baroAltitude; IMU_DelayUs(1000); } // calc offset with the average baroAltitudeOffset = baroSum / amount; return IMU_OK; }</pre>

Das Barometer wird immer im Normal-Mode betrieben.
 Für die richtige Luftdruck- und Höhenbestimmung müssen die Registerwerte kompensiert werden. Diese Kompensationswerte werden vom Sensor zur Verfügung gestellt und müssen vor Beginn der Werteberechnung eingelesen werden. Das Einlesen wird mit der Funktion *IMU_Baro_ReadCompensationValues()* realisiert.

Kompensationswerte für Barometermessung auslesen:

```
Dateiname: IMU_10DOF.c

/** 
 * @brief This function reads the temperature and pressure compensation values
 * @details values get stored in variable "baroCompensation"
 * @retval None
 */
void IMU_BARO_ReadCompensationValues(void)
{
    uint8_t buffer[24];
    IMU_ReadRegister(BMP280, IMU_BARO_DIG_T1_L_ADDR, buffer, 24);

    baroCompensation.T1 = (buffer[1] << 8) | buffer[0];
    baroCompensation.T2 = (buffer[3] << 8) | buffer[2];
    baroCompensation.T3 = (buffer[5] << 8) | buffer[4];

    baroCompensation.P1 = (buffer[7] << 8) | buffer[6];
    baroCompensation.P2 = (buffer[9] << 8) | buffer[8];
    baroCompensation.P3 = (buffer[11] << 8) | buffer[10];
    baroCompensation.P4 = (buffer[13] << 8) | buffer[12];
    baroCompensation.P5 = (buffer[15] << 8) | buffer[14];
    baroCompensation.P6 = (buffer[17] << 8) | buffer[16];
    baroCompensation.P7 = (buffer[19] << 8) | buffer[18];
    baroCompensation.P8 = (buffer[21] << 8) | buffer[20];
    baroCompensation.P9 = (buffer[23] << 8) | buffer[22];
}
```

Die Daten werden vom Hersteller angegeben und sind konstant in den Registern gespeichert. Die zusammengefügten Werte werden in der globalen Variable *baroCompensation* gespeichert.

5.5.4.3 Einlesen der Barometer-Daten

Die Messdaten vom BMP280 werden in den Registern 0xF7 press_msb bis 0xFC temp_xlsb gespeichert.

Einlesen der Barometerwerte:

```
Dateiname: IMU_10DOF.c | Ausschnitt aus IMU_BARO_ReadBaro() Funktion

/** 
 * @brief This function reads the barometer values and calculates temperature, pressure and
altitude
 * @details values gets stored in global variables 'baroTemperature', 'baroPressure' and
'baroAltitude'
 * @retval None
 */
void IMU_BARO_ReadBaro(void)
{
    uint8_t buffer[6] = {0};
    IMU_ReadRegister(BMP280, IMU_BARO_PRESS_ADDR, buffer, 6);

    int32_t adcPress = ((int32_t)buffer[0] << 12) | ((int32_t)buffer[1] << 4) |
((int32_t)buffer[2] >> 4);
    int32_t adcTemp = ((int32_t)buffer[3] << 12) | ((int32_t)buffer[4] << 4) |
((int32_t)buffer[5] >> 4);

    int32_t fineTemp;
    int32_t temp = IMU_BARO_CompensateTemp(adcTemp, &fineTemp);
    uint32_t press = IMU_BARO_CompensatePress(adcPress, fineTemp);

    // convert register values to real temp and pressure values
    baroTemperature = (float)temp / 100.0;
    baroPressure = (float)press / 256.0;

    ...
}
```

Für die Berechnung müssen die Temperatur und der Luftdruck kompensiert werden. Dafür werden die Funktionen *IMU_BARO_CompensateTemp()* und *IMU_BARO_CompensatePress()* verwendet. Diese Funktionen sind im Datenblatt des Sensors vom Hersteller vorgegeben und wurden für die Nutzung in der Steuerungssoftware angepasst und übernommen.

Barometer Temperatur- und Luftdruckwerte kompensieren:

```
Dateiname: IMU_10DOF.c

/** 
 * @brief This function compensates the temperature according to the datasheet
 * @details
 * Returns temperature in DegC, resolution is 0.01 DegC. Output value of "5123" equals
51.23 DegC.
 * @param adcTemp measured temperature
 * @param fineTemp
 * @return int32_t (temperature)
*/
int32_t IMU_BARO_CompensateTemp(int32_t adcTemp, int32_t *fineTemp)
{
    int32_t var1, var2, T;
    var1 = (((adcTemp >> 3) - ((int32_t)baroCompensation.T1 << 1))) *
((int32_t)baroCompensation.T2)) >> 11;
    var2 = (((((adcTemp >> 4) - ((int32_t)baroCompensation.T1)) * ((adcTemp >> 4) -
((int32_t)baroCompensation.T1))) >> 12) * ((int32_t)baroCompensation.T3)) >> 14;
    *fineTemp = var1 + var2;
    T = (*fineTemp * 5 + 128) >> 8;
    return T;
}

/** 
 * @brief This function compensates the pressure according to the datasheet
 * @details
 * Returns pressure in Pa as unsigned 32 bit integer in Q24.8 format (24 integer bits and 8
fractional bits).
 * Output value of "24674867" represents 24674867/256 = 96386.2 Pa = 963.862 hPa
 * @param adcPress measured pressure
 * @param fineTemp
 * @return uint32_t (pressure)
*/
uint32_t IMU_BARO_CompensatePress(int32_t adcPress, int32_t fineTemp)
{
    int64_t var1, var2, p;
    var1 = ((int64_t)fineTemp) - 128000;
    var2 = var1 * var1 * (int64_t)baroCompensation.P6;
    var2 = var2 + ((var1 * (int64_t)baroCompensation.P5) << 17);
    var2 = var2 + (((int64_t)baroCompensation.P4) << 35);
    var1 = ((var1 * var1 * (int64_t)baroCompensation.P3) >> 8) + ((var1 *
(int64_t)baroCompensation.P2) << 12);
    var1 = (((((int64_t)1) << 47) + var1)) * ((int64_t)baroCompensation.P1) >> 33;
    if(var1 == 0)
    {
        return 0; // avoid exception caused by division by zero
    }
    p = 1048576 - adcPress;
    p = (((p << 31) - var2) * 3125) / var1;
    var1 = (((int64_t)baroCompensation.P9) * (p >> 13) * (p >> 13)) >> 25;
```

```

var2 = (((int64_t)baroCompensation.P8) * p) >> 19;
p = ((p + var1 + var2) >> 8) + (((int64_t)baroCompensation.P7) << 4);
return (uint32_t)p;
}

```

5.5.4.4 Berechnung der Höhe

Mit dem gemessenen Luftdruck kann die aktuelle Höhe über dem Meeresspiegel ausgerechnet werden. Dafür wird eine Höhenberechnungsformel aus dem Datenblatt übernommen:

$$\text{altitude} = 44330 * \left(1 - \left(\frac{p}{p_0} \right)^{\frac{1}{5.255}} \right)$$

Abbildung 133: Formel Berechnung Höhe über Meeresspiegel

altitude ... Höhe über den Meeresspiegel in Meter (m)

p ... gemessener Luftdruck in Hektopascal (hPa)

p_0 ... typischer Luftdruck am Meeresspiegel, normalerweise 1013,25hPa

Höhe über Meeresspiegel bestimmen:

IMU_10DOF.c Ausschnitt aus IMU_BARO_ReadBaro() Funktion
<pre> void IMU_BARO_ReadBaro(void) { ... // convert pressure to altitude according to datasheet float presshPa = baroPressure / 100; baroAltitude = (44330.0 * (1.0 - pow(presshPa / 1013.25, 1.0 / 5.255))) - baroAltitudeOffset; } </pre>

Dadurch, dass in der Berechnung die Variable *baroAltitudeOffset* verwendet wird, berechnet sich die relative Höhe der Drohne zum Boden. Wenn dieser Teil der Berechnung entfernt wird, kann die absolute Höhe bestimmt werden.

5.6 Motorregelalgorithmus

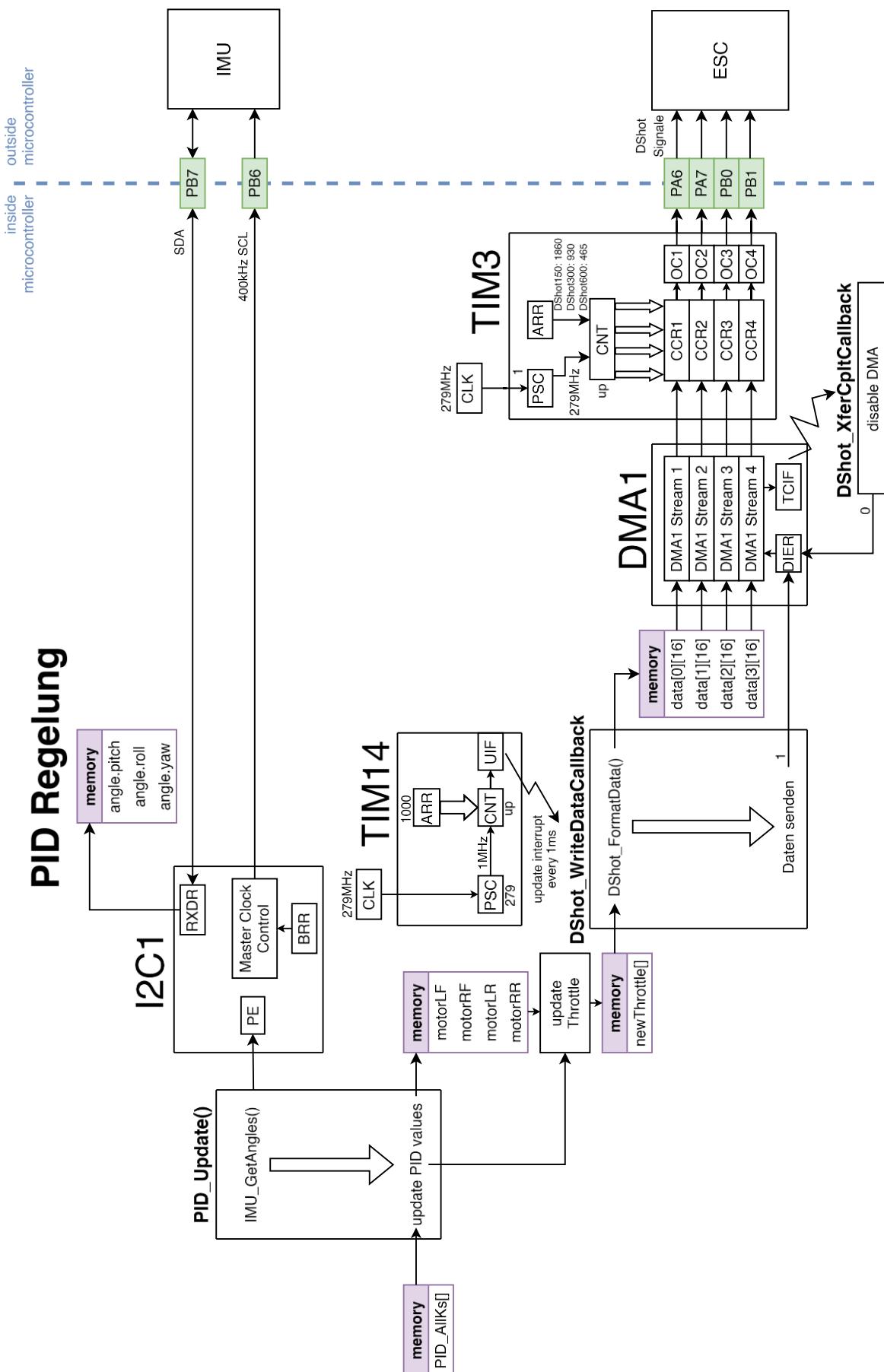


Abbildung 134: Motorregelalgorithmus Architektur

Am Beginn des PID-Regelalgorithmus wird die Regelgröße (Istwert) mit Hilfe des IMU bestimmt. Dafür werden die MPU9250-Daten vom Accelerometer und Gyroskop mit der I2C1-Peripherie eingelesen, in Lagewinkel (Pitch, Roll und Yaw) umgewandelt und in der globalen Variable *angle* gespeichert (siehe: [Kapitel 5.5.3.3, 5.5.3.4](#)).

Als nächsten Schritt bestimmt der Regler die Motorgeschwindigkeiten, um die Drohne in die gewünschte Lage zu bringen (Geschwindigkeiten von 0-100%). Diese werden dann mit der Funktion *DShot_SendThrottle()* an den Ausgangsalgorithmus übergeben, der die Daten mittels DShot-Protokolls an den electronic speed controller (ESC) mit der TIM3-Peripherie überträgt (siehe: [Kapitel 5.7.3](#)).

Damit der ESC nicht in einen Standby-Modus übergeht, müssen durchgehend Werte geschickt werden. Die TIM14-Peripherie löst alle 1ms ein update Interrupt *DShot_WriteDataCallback()* aus. In der ISR wird ein DShot-Paket aus den Throttle-Werten gebildet und dann mit Hilfe des DMA1-Controllers zu den PWM-Ausgängen der TIM3-Peripherie geschickt (siehe: [Kapitel 5.7.3](#)).

Damit alle vier Timer-Kanäle gleichzeitig das Ausgangssignal schicken, wird nach Beendung des Datentransfers ein transfer complete Interrupt ausgelöst, indem der DMA1-Controller deaktiviert wird. Damit im nächsten Zyklus alle vier Kanäle gleichzeitig senden können, werden zuerst in *DShot_WriteDataCallback()* alle vier Übertragungen vorbereitet und danach der Transfer der einzelnen DMA1-Controller-Kanäle gestartet (siehe: [Kapitel 5.7.3](#)).

Dadurch herrscht bei jedem Zyklus die bestmögliche parallele Übertragung der Motordaten.

5.7 Motoransteuerung

5.7.1 DShot Protokoll

Zum Ansteuern der Motoren müssen Befehle an dem Electronic Speed Controller (ESC) geschickt werden. Dafür wird das DShot-Protokoll verwendet. Im Vergleich zur analogen PWM-Ansteuerung der ESC, werden die DShot-Signale um ein Vielfaches schneller gesendet und dadurch, dass es sich um ein digitales Protokoll handelt, wird keine Kalibrierungssequenz der Throttle Minimal- und Maximalwerte benötigt.

Das Protokoll besteht aus 16 Bits. Um zwischen 0 und 1 zu unterscheiden, wird der PWM Duty-Cycle des Signals verändert. Ein Duty-Cycle von 75% entspricht einer „1“ und ein Duty-Cycle von 37,5% entspricht einer „0“. Das Protokoll gibt es in mehreren Geschwindigkeiten (DShot150, DShot300, DShot600, DShot1200), wobei die Zahl angibt, in welcher Frequenz (f_{DShot} in Kiloherz) ein Bit gesendet wird. Der/Die BenutzerIn kann sich aussuchen mit welcher Geschwindigkeit die Bits gesendet werden soll – *siehe: [Kapitel 5.7.2](#)*

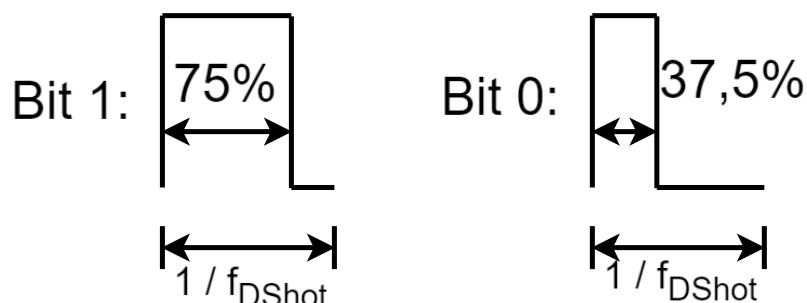


Abbildung 135: DShot Bit 0/1 Duty Cycle + Geschwindigkeit

Die ersten 11 Bits beinhalten den Throttle-Wert in Prozent, wobei $48 = 0\%$ Throttle und $2048 = 100\%$ Throttle. Die Zahlenwerte 0 bis 47 sind für spezielle Befehle reserviert.

Mit dem nächsten Bit kann eine Telemetrie-Anfrage gesendet werden, die bei einer erweiterten Version des Protokolls (bidirektionalen DShot) verwendet werden kann. Die ESC, die für die Diplomarbeit verwendet wird, unterstützt diese Variante nicht.

Die letzten 4 Bits bilden eine Checksumme, die aus der XOR-Verknüpfung der ersten 12 Bits in jeweils Viererblöcken gebildet wird. Die Ruhezeit zwischen Paketen ist im low-Zustand.



Abbildung 136: DShot Beispiel Übertragung

In der Beispielübertragung wird der Zahlenwert 0b10000011000 (1048) geschickt. Dieser entspricht einen Throttle-Wert von 50%. Die Telemetry Request ist ausgeschaltet, und die Berechnung der Checksumme ergibt den Wert 0b1011, der am Ende des Pakets ausgegeben wird.

5.7.2 Initialisierung Motoransteuerung

```
Dateiname: dshot.c

/**
 * @brief This function initializes the output ESC DShot signal
 * @param htim pointer to TIM_HandleTypeDef (output timer)
 * @param protocol DSHOT150, DSHOT300, DSHOT600
 * @param updateTim pointer to TIM_HandleTypeDef (executes 1ms interrupt)
 * @return DShot_Status
 */
DShot_Status DShot_Init(TIM_HandleTypeDef *htim, ESC_OutputProtocol protocol,
TIM_HandleTypeDef *updateTim)
{
    // set and check timer pointer
    DShot_OutputTim = htim;
    if(DShot_OutputTim == NULL)
        return DSHOT_TIM_ERROR;

    // set the right timer frequency, 279MHz / (prescaler * autoreload)
    __HAL_TIM_SET_PRESCALER(DShot_OutputTim, 1 - 1);
    __HAL_TIM_SET_AUTORELOAD(DShot_OutputTim, protocol - 1);

    oneDC = ceil((float)protocol * 0.75);    // calculate duty cycle for sending bit 1
    zeroDC = ceil((float)protocol * 0.375); // calculate duty cycle for sending bit 0

    // define custom transfer complete ISR
    DShot_OutputTim->hdma[ESC_LF_DMA_ID]->XferCpltCallback = DShot_DMA_XferCpltCallback;
    DShot_OutputTim->hdma[ESC_RF_DMA_ID]->XferCpltCallback = DShot_DMA_XferCpltCallback;
    DShot_OutputTim->hdma[ESC_LR_DMA_ID]->XferCpltCallback = DShot_DMA_XferCpltCallback;
    DShot_OutputTim->hdma[ESC_RR_DMA_ID]->XferCpltCallback = DShot_DMA_XferCpltCallback;

    // set output low
    __HAL_TIM_SET_COMPARE(DShot_OutputTim, ESC_LF_TIM_CH, 0);
    __HAL_TIM_SET_COMPARE(DShot_OutputTim, ESC_RF_TIM_CH, 0);
    __HAL_TIM_SET_COMPARE(DShot_OutputTim, ESC_LR_TIM_CH, 0);
    __HAL_TIM_SET_COMPARE(DShot_OutputTim, ESC_RR_TIM_CH, 0);

    // start all timers in pwm output mode
    HAL_TIM_PWM_Start(DShot_OutputTim, ESC_LF_TIM_CH);
    HAL_TIM_PWM_Start(DShot_OutputTim, ESC_RF_TIM_CH);
    HAL_TIM_PWM_Start(DShot_OutputTim, ESC_LR_TIM_CH);
    HAL_TIM_PWM_Start(DShot_OutputTim, ESC_RR_TIM_CH);

    // set custom ISR for 1ms interrupt + start timer
    HAL_TIM_RegisterCallback(updateTim, HAL_TIM_PERIOD_ELAPSED_CB_ID,
DShot_WriteDataCallback);
    HAL_TIM_Base_Start_IT(updateTim);

    // at beginning send 0% throttle to every motor
    if(DShot_SendThrottle(0, 0, 0, 0) != DSHOT_OK)
        return DSHOT_TIM_ERROR;
}
```

```

    HAL_Delay(5000);

    return DSHOT_OK;
}

```

Das Protokoll wird mit der Timer-Peripherie gesendet, welche ein PWM-Signal ausgibt. Damit das DShot-Protokoll erstellt wird, schickt der DMA-Controller die richtigen Werte für die Änderung des Duty-Cycle der Signale.

5.7.3 Motoransteuerung Software

Dadurch, dass sich Motoren nicht automatisch ausschalten, wird ein DShot-Packet jede Millisekunde gesendet. Dafür wird der *updateTim* verwendet. Wenn dieser Timer einen Überlauf ergibt, wird ein Interrupt ausgelöst.

Dateiname: dshot.c
<pre> /** * @brief This function is the call for a 1ms interrupt, to send every 1ms * @retval None */ static void DShot_WriteDataCallback(TIM_HandleTypeDef *htim) { static uint16_t prevThrottle[4] = {1,1,1,1}; // stores previous throttle values static uint16_t data[4][18] = {0}; // stores each bit (duty cycle value) for every motor // check change of throttle values if(newThrottle[0] != prevThrottle[0] newThrottle[1] != prevThrottle[1] newThrottle[2] != prevThrottle[2] newThrottle[3] != prevThrottle[3]) { DShot_FormatData(newThrottle, 0, data); for(int8_t i = 0; i < 4; i++) prevThrottle[i] = newThrottle[i]; } // start dma transfer to the capture compare register HAL_DMA_Start_IT(DShot_OutputTim->hdma[ESC_LF_DMA_ID], (uint32_t)&data[0][0], ESC_TIM_GET_CCR_ADDR(ESC_LF_TIM_CH), 18); HAL_DMA_Start_IT(DShot_OutputTim->hdma[ESC_RF_DMA_ID], (uint32_t)&data[1][0], ESC_TIM_GET_CCR_ADDR(ESC_RF_TIM_CH), 18); HAL_DMA_Start_IT(DShot_OutputTim->hdma[ESC_LR_DMA_ID], (uint32_t)&data[2][0], ESC_TIM_GET_CCR_ADDR(ESC_LR_TIM_CH), 18); HAL_DMA_Start_IT(DShot_OutputTim->hdma[ESC_RR_DMA_ID], (uint32_t)&data[3][0], ESC_TIM_GET_CCR_ADDR(ESC_RR_TIM_CH), 18); // reset counter to get rid of delay between channels __HAL_TIM_SET_COUNTER(DShot_OutputTim, 0); } </pre>

```
// enable dma / start sending
__HAL_TIM_ENABLE_DMA(DShot_OutputTim, TIM_DMA_CC1);
__HAL_TIM_ENABLE_DMA(DShot_OutputTim, TIM_DMA_CC2);
__HAL_TIM_ENABLE_DMA(DShot_OutputTim, TIM_DMA_CC3);
__HAL_TIM_ENABLE_DMA(DShot_OutputTim, TIM_DMA_CC4);
}
```

Diese Funktion bereitet die Daten für die Übertragung vor und startet den DMA-Controller. Wenn die Datenübertragung des DMA-Controllers abgeschlossen hat, wird ein Interrupt ausgelöst.

Dateiname: dshot.c
<pre>/** * @brief This function is the ISR for DMA transmit complete * @param hdma * @retval None */ static void DShot_DMA_XferCpltCallback(DMA_HandleTypeDef *hdma) { // disable DMA to get rid of the delay between channels if(hdma == DShot_OutputTim->hdma[TIM_DMA_ID_CC1]) { __HAL_TIM_DISABLE_DMA(DShot_OutputTim, TIM_DMA_CC1); } else if(hdma == DShot_OutputTim->hdma[TIM_DMA_ID_CC2]) { __HAL_TIM_DISABLE_DMA(DShot_OutputTim, TIM_DMA_CC2); } else if(hdma == DShot_OutputTim->hdma[TIM_DMA_ID_CC3]) { __HAL_TIM_DISABLE_DMA(DShot_OutputTim, TIM_DMA_CC3); } else if(hdma == DShot_OutputTim->hdma[TIM_DMA_ID_CC4]) { __HAL_TIM_DISABLE_DMA(DShot_OutputTim, TIM_DMA_CC4); } }</pre>

Diese Funktion schaltet den DMA-Controller für alle Kanäle ab. Dies ermöglicht, dass alle Pakete zur selbe Zeit gesendet werden.

Um den Throttlewert der Motoren zu ändern, muss folgende Funktion ausgeführt werden:

Dateiname: dshot.c

```
/** 
 * @brief This function formats the motor data for the DShot protocol
 * @param motorLF percent of throttle value of left front motor (0-100)
 * @param motorRF percent of throttle value of right front motor (0-100)
 * @param motorLR percent of throttle value of left rear motor (0-100)
 * @param motorRR percent of throttle value of right rear motor (0-100)
 * @retval DShot_Status
 */
DShot_Status DShot_SendThrottle(double motorLF, double motorRF, double motorLR, double
motorRR)
{
    if(DShot_OutputTim == NULL)
        return DSHOT_TIM_ERROR;

    /**
     * timer channel 1 = left front motor
     * timer channel 2 = right front motor
     * timer channel 3 = left rear motor
     * timer channel 4 = right rear motor
     */

    // convert to dshot throttle format (48 = 0% throttle, 2047 = 100% throttle)
    newThrottle[0] = 48 + 20 * motorLF;
    newThrottle[1] = 48 + 20 * motorRF;
    newThrottle[2] = 48 + 20 * motorLR;
    newThrottle[3] = 48 + 20 * motorRR;

    return DSHOT_OK;
}
```

Diese Funktion speichert die neuen Throttlewerte in der Variable *newThrottle*, die in dem *DShot_WriteDataCallback()* auf Änderung überprüft wird.

Um diese Daten in das richtige Format zu ändern, wird die Funktion *DShot_FormatData()* ausgeführt. Das Format besteht aus einem Array (für jedes Bit) den jeweiligen Duty Cycle → für Bit 1 – 75% und für Bit 0 – 37,5%.

Dateiname: dshot.c

```
/** 
 * @brief This function converts the throttle value to duty cycle bits
 * @param throttle Throttle values (0-2047)
 * @param telemetry telemetry request bit
 * @param data formatted data by the function, array of 75% or 37.5%
 * @retval None
 */
void DShot_FormatData(uint16_t *throttle, int8_t telemetry, uint16_t data[4][18])
{
    uint16_t withoutCS, complete, div;

    // format the data to packets
    for(int8_t i = 0; i < 4; i++)
    {
        // first 12 bits (without Checksum)
        withoutCS = (throttle[i] << 1) | telemetry;

        // format whole data frame
        complete = withoutCS << 4 | ((withoutCS ^ (withoutCS >> 4)) ^ (withoutCS >> 8)) & 0x0F;

        // convert each bit to the specific duty cycle length
        div = 0x8000;
        for(int8_t j = 0; j < 16; j++)
        {
            data[i][j] = (complete & div) ? oneDC : zeroDC;
            div >>= 1;
        }
    }
}
```

Die Datenpakete werden zusammengestellt und danach in den richtigen Duty-Cycle-Wert umgewandelt.

5.8 PID-Regler

5.8.1 Initialisierung PID-Regler

Damit die Filterkoeffizienten während der Programmlaufzeit geändert werden können, wird der DMA-Controller verwendet. Dafür wird die UART4-Peripherie, die auch für die Terminal Ausgabe (siehe: [Kapitel 5.10](#)) verwendet wird.

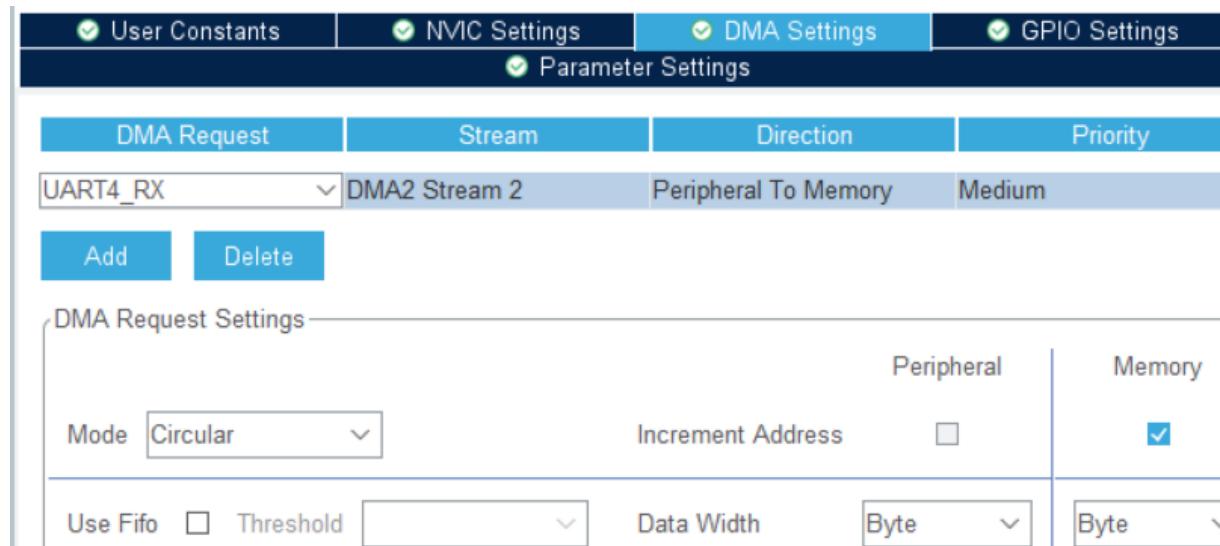


Abbildung 137: STM32CubeMX Einstellung PID DMA Empfang

Zum Initialisieren wird die Funktion *PID_Init()* ausgeführt.

```
Dateiname: PID.c
/*
 * @brief This function initializes the change Ks system
 * @param phuart pointer to UART_HandleTypeDef (input uart)
 * @retval None
 */
void PID_Init(UART_HandleTypeDef *phuart)
{
    // start reception with uart
    HAL_UART_RegisterCallback(phuart, HAL_UART_RX_COMPLETE_CB_ID, PID_ChangeKs);

    // read 10 Byte (for 10 chars for format "0 0.123456")
    HAL_UART_Receive_DMA(phuart, (uint8_t *)&receiveData, 10);
}
```

Diese Funktion startet den DMA-Controller, der auf 10 Bytes (10 Zeichen) von der UART-Peripherie wartet und in dem Array *receiveData[]* speichert. Dadurch können Werte im Format „0 0.123456“ gesendet werden. Die erste Zahl gibt den Array-Index vom globalen Array *PID_AllKs[9]* und die zweite Zahl den Wert des Index' an.

Index	0	1	2	3	4	5	6	7	8
Wert	KP Roll	KI Roll	KD Roll	KP Pitch	KI Pitch	KD Pitch	KP Yaw	KI Yaw	KD Yaw

Nachdem ein Datensatz eingelesen worden ist, wird ein Interrupt mit der Funktion *PID_ChangeKs()* ausgeführt.

```
Dateiname: PID.c
/*
 * @brief This function changes a PID controller coefficients via uart
 * @param huart
 */
void PID_ChangeKs(UART_HandleTypeDef *huart)
{
    // get index
    int index = atoi(&receiveData[0]);

    // check if index is in range
    if(index < 0 || index > 8)
    {
        Terminal_Print("index out of range\n\r");
        return;
    }

    // get value
    char value[9];
    sprintf(value, "%c%c%c%c%c%c%c", receiveData[2], receiveData[3], receiveData[4],
receiveData[5], receiveData[6], receiveData[7], receiveData[8], receiveData[9]);

    // check if value is in range
    double tmp = atof(value);
    if(tmp > 5 || tmp < 0)
    {
        Terminal_Print("value out of range\n\r");
        return;
    }

    // write value to array
    PID_AllKs[index] = tmp;

    // write confirmation to terminal (eg: 0 = 0.250000)
    sprintf(txt, "[%d] = %f\n\r", index, PID_AllKs[index]);
    Terminal_Print(txt);
}
```

Diese Funktion wandelt die Daten in Zahlen um und schreibt den Wert in den richtigen Index im Array *PID_AllKs[]*.

Danach wird ein Bestätigungstext mit dem Arrayindex und Zahlenwert ausgegeben. Wenn dieser Text nicht zurückgesendet wird, hat es ein Problem bei der Datenübertragung gegeben. Wenn der Index oder die Zahl außerhalb des verwendeten Zahlenbereichs ist, wird der Text „out of range“ ausgegeben.

5.8.2 PID-Algorithmus

```
Dateiname: PID.c

/**
 * @brief This function controls the flight PID controller
 * @details
 * The max throttle value can be altered with the define PID_MAX_TURN
 * @param inputThrottle throttle value from joysticks
 * @param inputPitch pitch value from joysticks
 * @param inputRoll roll value from joysticks
 * @param inputYaw yaw value from joysticks
 * @return PID_Status
 */
PID_Status PID_Update(float inputThrottle, float inputPitch, float inputRoll, float
inputYaw)
{
    static double I_Roll = 0, errorRollPrev = 0;
    static double I_Pitch = 0, errorPitchPrev = 0;
    static double I_Yaw = 0, errorYawPrev = 0;

    /*****
    ----- get current angles + dt -----
    *****/
    IMU_GetAngles();

    /*****
    ----- check off mode -----
    *****/
    if(inputThrottle < 5)
    {
        I_Roll = 0;
        errorRollPrev = 0;

        I_Pitch = 0;
        errorPitchPrev = 0;

        I_Yaw = 0;
        errorYawPrev = 0;

        DShot_SendThrottle(0, 0, 0, 0);

        return PID_OK;
    }

    /*****
    ----- calc roll output -----
    *****/
    double errorRoll = -inputRoll - angle.roll;

    I_Roll += (errorRoll * imu_DeltaTime) * PID_AllKs[1];
}
```

```

// limit I value to +/- max throttle addition
if(I_Roll > PID_MAX_TURN) I_Roll = PID_MAX_TURN;
else if(I_Roll < -PID_MAX_TURN) I_Roll = -PID_MAX_TURN;

double rollOutput = PID_AllKs[0] * errorRoll + I_Roll + PID_AllKs[2] * ((errorRoll -
errorRollPrev) / imu_DeltaTime);

// limit output value to +/- max throttle addition
if(rollOutput > PID_MAX_TURN) rollOutput = PID_MAX_TURN;
else if(rollOutput < -PID_MAX_TURN) rollOutput = -PID_MAX_TURN;

errorRollPrev = errorRoll;

/***** -----
----- calc pitch output -----
----- *****/
double errorPitch = inputPitch - angle.pitch;

I_Pitch += (errorPitch * imu_DeltaTime) * PID_AllKs[4];

// limit I value to +/- max throttle addition
if(I_Pitch > PID_MAX_TURN) I_Pitch = PID_MAX_TURN;
else if(I_Pitch < -PID_MAX_TURN) I_Pitch = -PID_MAX_TURN;

double pitchOutput = PID_AllKs[3] * errorPitch + I_Pitch + PID_AllKs[5] * ((errorPitch -
errorPitchPrev) / imu_DeltaTime);

// limit output value to +/- max throttle addition
if(pitchOutput > PID_MAX_TURN) pitchOutput = PID_MAX_TURN;
else if(pitchOutput < -PID_MAX_TURN) pitchOutput = -PID_MAX_TURN;

errorPitchPrev = errorPitch;

/***** -----
----- calc yaw output -----
----- *****/
static double controlYaw = 0;
static int8_t changeYawFlag = 1;

double yawOutput = inputYaw * PID_MAX_TURN;

if(yawOutput >= -1 && yawOutput <= 1)
{
    if(changeYawFlag == 1)
        controlYaw = angle.yaw;

    changeYawFlag = 0;

    double errorYaw = controlYaw - angle.yaw;
}

```

```

I_Yaw += (errorYaw * imu_DeltaTime) * PID_AllKs[7];

// limit I value to +/- max throttle addition
if(I_Yaw > PID_MAX_TURN) I_Yaw = PID_MAX_TURN;
else if(I_Yaw < -PID_MAX_TURN) I_Yaw = -PID_MAX_TURN;

yawOutput = PID_AllKs[6] * errorYaw + I_Yaw + PID_AllKs[8] * ((errorYaw - errorYawPrev)
/ imu_DeltaTime);

// limit output value to +/- max throttle addition
if(yawOutput > PID_MAX_TURN) yawOutput = PID_MAX_TURN;
else if(yawOutput < -PID_MAX_TURN) yawOutput = -PID_MAX_TURN;

errorYawPrev = errorYaw;
}

else
{
    changeYawFlag = 1;
}

/*****
----- check min/max values + output -----
*****/
// calc throttle value
float motorLF = inputThrottle - pitchOutput - rollOutput - yawOutput;
float motorRF = inputThrottle - pitchOutput + rollOutput + yawOutput;
float motorLR = inputThrottle + pitchOutput - rollOutput + yawOutput;
float motorRR = inputThrottle + pitchOutput + rollOutput - yawOutput;

// if value is less then 5 -> turn motors off to hinder motor tremble
if(motorLF < 5) motorLF = 0;
if(motorRF < 5) motorRF = 0;
if(motorLR < 5) motorLR = 0;
if(motorRR < 5) motorRR = 0;

// check if value is higher than max value
if(motorLF > inputThrottle + PID_MAX_TURN) motorLF = inputThrottle + PID_MAX_TURN;
if(motorRF > inputThrottle + PID_MAX_TURN) motorRF = inputThrottle + PID_MAX_TURN;
if(motorLR > inputThrottle + PID_MAX_TURN) motorLR = inputThrottle + PID_MAX_TURN;
if(motorRR > inputThrottle + PID_MAX_TURN) motorRR = inputThrottle + PID_MAX_TURN;

DShot_SendThrottle(motorLF, motorRF, motorLR, motorRR);

return PID_OK;
}

```

Der gesamte PID-Regler besteht aus drei PID-Reglern. Jeweils einer für die Pitch-, Roll- und Yaw-Achse. Es wird mit dem IMU-Lagewinkel und den übergebenen Variablen die Regelabweichungen gebildet, die dann weiter zu Throttle-Werten umgerechnet werden.

5.9 Terminal Übertragung und Status LEDs

Terminal Übertragung und Status LEDs

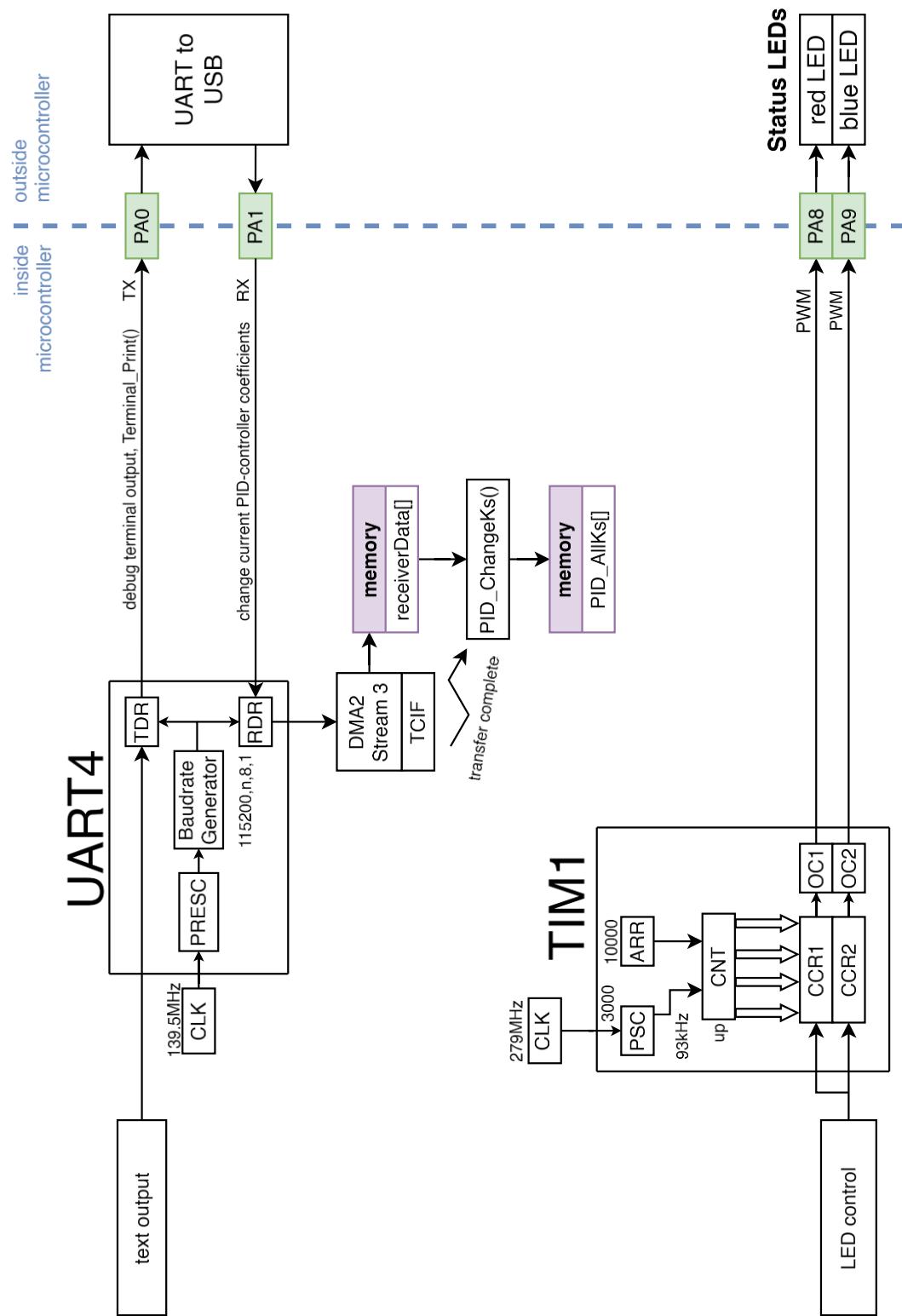


Abbildung 138: Terminal Übertragung und Status LEDs Architektur

Um am Terminal einen Text für eine Status-Anzeige auszugeben, wird die Funktion *Terminal_Print()* verwendet. Diese formatiert den übergebenen String und schickt ihn mit der UART4-Peripherie an einen UART/USB-Converter, der mit einem PC verbunden ist (siehe: [Kapitel 5.10](#)).

Während der Initialisierungssequenz wird die Frequenz und der Duty-Cycle des PWM-Ausgangs der TIM1-Peripherie geändert. Dadurch können die Status-LEDs in unterschiedlichen Frequenzen blinken.

Während der Initialisierung ändert die blaue LED ihre Blinkgeschwindigkeit je nach Komponente, die initialisiert wird. Im Fall eines Errors schaltet sich die blaue LED aus, und die rote LED beginnt zu leuchten. Bei erfolgreichem Abschluss der Initialisierungssequenz beginnt die blaue LED durchgehend zu leuchten (siehe: [Kapitel 5.11](#)).

Mit der UART4-Peripherie wird eine weitere wichtige Funktion realisiert. Über den UART/USB-Converter können Texte an den Mikrocontroller gesendet werden. Mit ihnen können die PID-Regelkoeffizienten während der Laufzeit verändert werden.

Dafür empfängt der DMA2-Controller durchgehend alle ankommenden Daten. Nach dem Empfang wird ein transfer complete interrupt mit der Funktion *PID_ChangeKs()* ausgeführt, der den String zu Zahlen umformt und in dem Array *PID_AllKs[]* für die weitere Verwendung im Motorregelalgorithmus speichert (siehe: [Kapitel 5.8](#)).

5.10 Terminal Kommunikation

Die Terminal Ausgabe ist essenziell für die Programmkontrolle, da der aktuelle Inhalt von Variablen direkt ausgegeben werden kann. Im Vergleich zum Debuggen wird das Programm dabei nur kurz unterbrochen, was bei Interrupt gesteuerten System zu Problemen führen kann.

Die Kommunikation findet mithilfe der UART-Peripherie statt. Diese ist mit einer Zusatzplatine verbunden, um den Datenaustausch mit einem externen System zu ermöglichen.

UART-Einstellungen in STM32CubeMX:

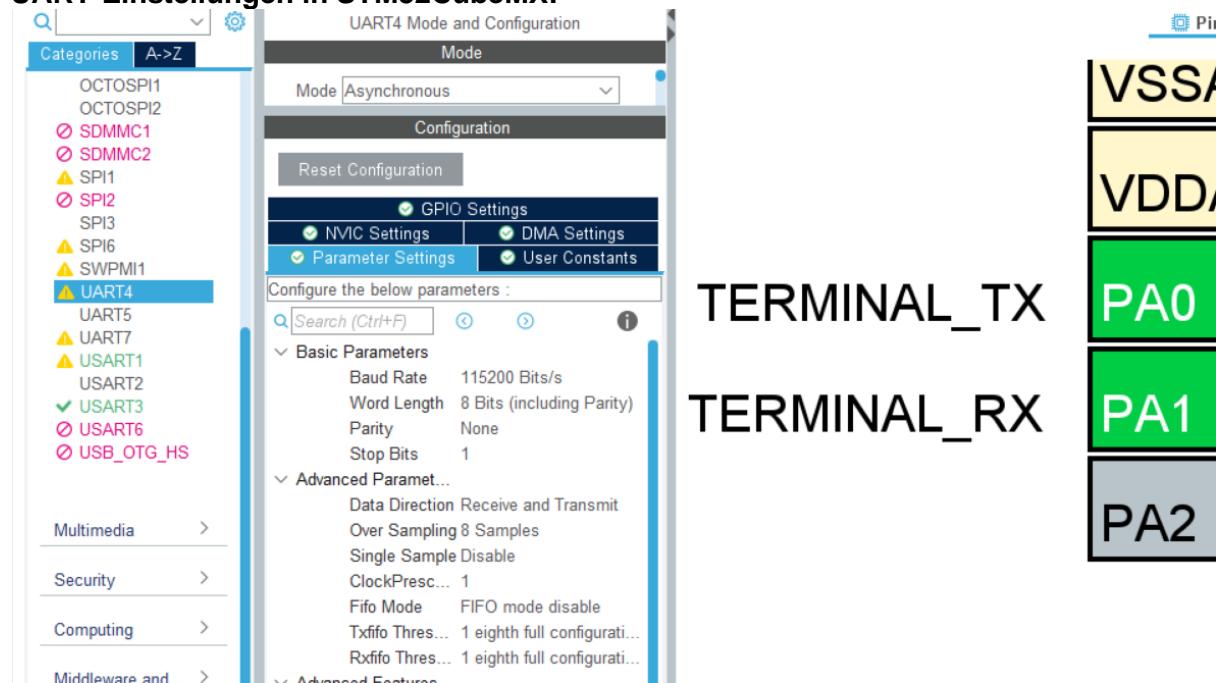
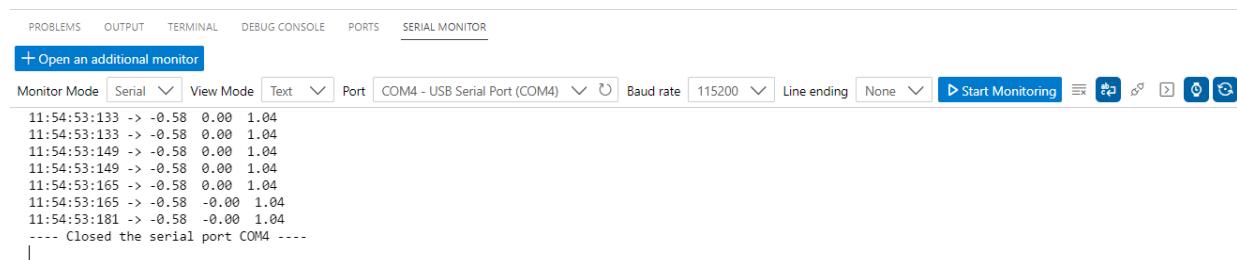


Abbildung 139: STM32CubeMX Einstellungen Terminal

Für den Empfang der Daten wird die Erweiterung „Serial Monitor“ [SERM] für Visual Studio Code verwendet.



Wichtig:

Die Baudrate, die für die Datenübertragung verwendet wird, ist 115200 Bits/s.

Text zum Terminal senden:

```
Dateiname: status_handling.c

/** 
 * @brief This function prints a string to the terminal
 * @param string
 * @retval none
 */
void Terminal_Print(char *string)
{
    HAL_UART_Transmit(&huart4, (uint8_t *)string, strlen(string), HAL_MAX_DELAY);
}
```

Beispiel Programmaufruf:

```
// with only text
Terminal_Print("wichtige Daten\n\r");

// with variable
char text[] = "wichtige Daten\n\r";
Terminal_Print(text);
```

5.11 Status – LEDs

Auf der Flight-Controller Platine befinden sich eine rote und eine blaue LED. Die blaue wird zum Initialisierungsstatus, und die rote LED wird für zur Anzeige von Errors verwendet.

Blaue LED	leuchtet nicht blinkt alle 0,5 Sekunden blinkt alle 0,4 Sekunden blinkt alle 0,3 Sekunden blinkt alle 0,2 Sekunden blinkt alle 0,1 Sekunden leuchtet durchgehend	Problem bei Programmstart DS2438 initialisiert IMU initialisiert PID initialisiert DShot initialisiert Receiver initialisiert Initialisierungen abgeschlossen
Rote LED	leuchtet nicht leuchtet durchgehend blinkt alle 0,1 Sekunden	kein Error Error während Initialisierungen Failsafe von Receiver oder DS2438 ausgelöst

Einstellungen STM32CubeMX:

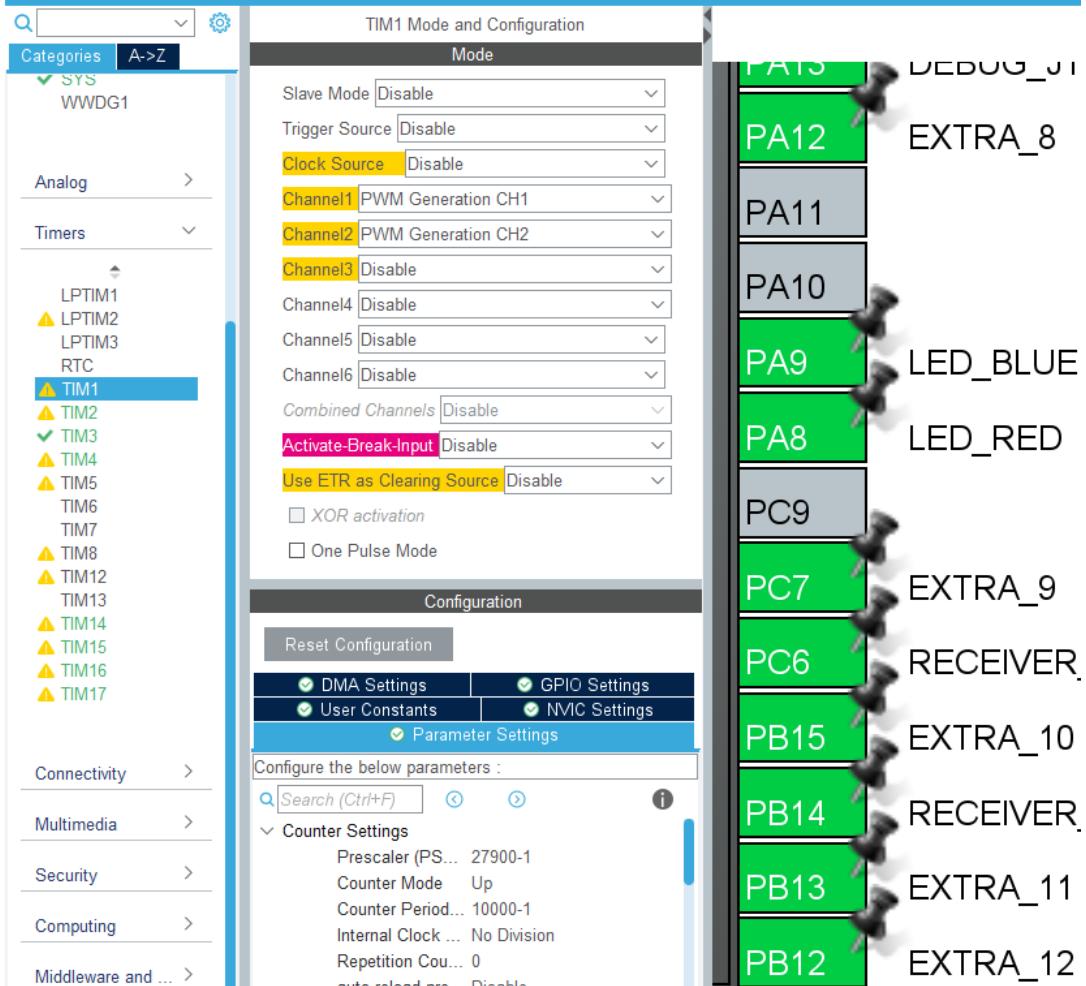


Abbildung 141: STM32CubeMX Einstellungen LEDs

Für die Änderung der Frequenz des PWM-Signals wird folgende Formel verwendet:

$$f_{Blink} = \frac{f_{Timer}}{(PSC + 1) * (ARR + 1)}$$

f_{Blink} ... Ausgangsfrequenz

f_{Timer} ... Timer Eingangs frequenz, für Diplomarbeit 279MHz

PSC ... prescaler Registerwert

ARR ... auto-reload Registerwert, für Diplomarbeit fixiert auf 10000-1

Wenn der Wert im Counter-Register (CNT) unter dem Wert im capture/compare Register (CCR) ist, wird ein low-Zustand ausgegeben. Wenn der CNT-Wert größer ist als der CCR-Wert ist, dann wird ein high-Zustand ausgegeben. Durch die Änderung des CCR-Wertes kann der Duty Cycle verändert werden.

Für die Ansteuerung der LEDs wird ein Duty-Cycle von 50% verwendet. Dafür muss der Wert im CCR-Register die Hälfte des Wertes im ARR-Register gesetzt werden – *siehe Abbildung 142*

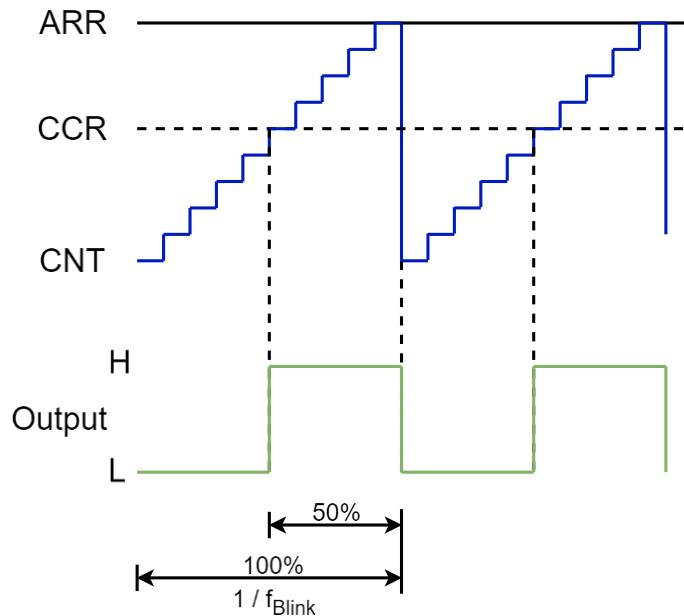


Abbildung 142: Beispiel PWM-Signalverlauf

Um die Werte während der Programmlaufzeit ändern zu können, verwendet man folgende Funktionen:

```
// set duty cycle for LEDs
__HAL_TIM_SET_COMPARE(&htim1, TIM_CHANNEL_1, 0);
__HAL_TIM_SET_COMPARE(&htim1, TIM_CHANNEL_2, 5000);

// set frequency ~2Hz
__HAL_TIM_SET_PRESCALER(&htim1, 14000 - 1);

// start timer for LEDs
HAL_TIM_PWM_Start(&htim1, TIM_CHANNEL_1);
HAL_TIM_PWM_Start(&htim1, TIM_CHANNEL_2);
```

Hinweis:

Die Funktionen erlauben es, auch die Registerwerte zu ändern, ohne die Signal-Ausgabe unterbrechen zu müssen.

6 Datenübertragung der Mess- und Videodaten

6.1 Überblick Datenübertragung

Um das Video und die Messdaten der Sensoren, von der Drohne, vom Boden aus anzuschauen und diese zu visualisieren, benötigen wir einen Sender, der die Signale an einen geeigneten Sender über eine Funkverbindung schickt. Das Kameravideo und die Messdaten werden von diesem Sender also gleichzeitig gesendet, dabei empfängt die FPV-Brille nur das Videosignal und der FM-Receiver empfängt beide Signale, also Bild und Messdaten. In der FPV-Brille kann das Video dann Live mitverfolgt werden. Vom FM-Receiver aus werden beide Signale dann an den Raspberry Pi weitergeleitet, der die Signale verarbeitet und die verarbeiteten Daten auf einem Server ablegt. Von dort aus kann somit jeder Zuschauer über die Handy-App den Flug, also Video und Sensordaten, live verfolgen (siehe: [Kapitel 7](#)).

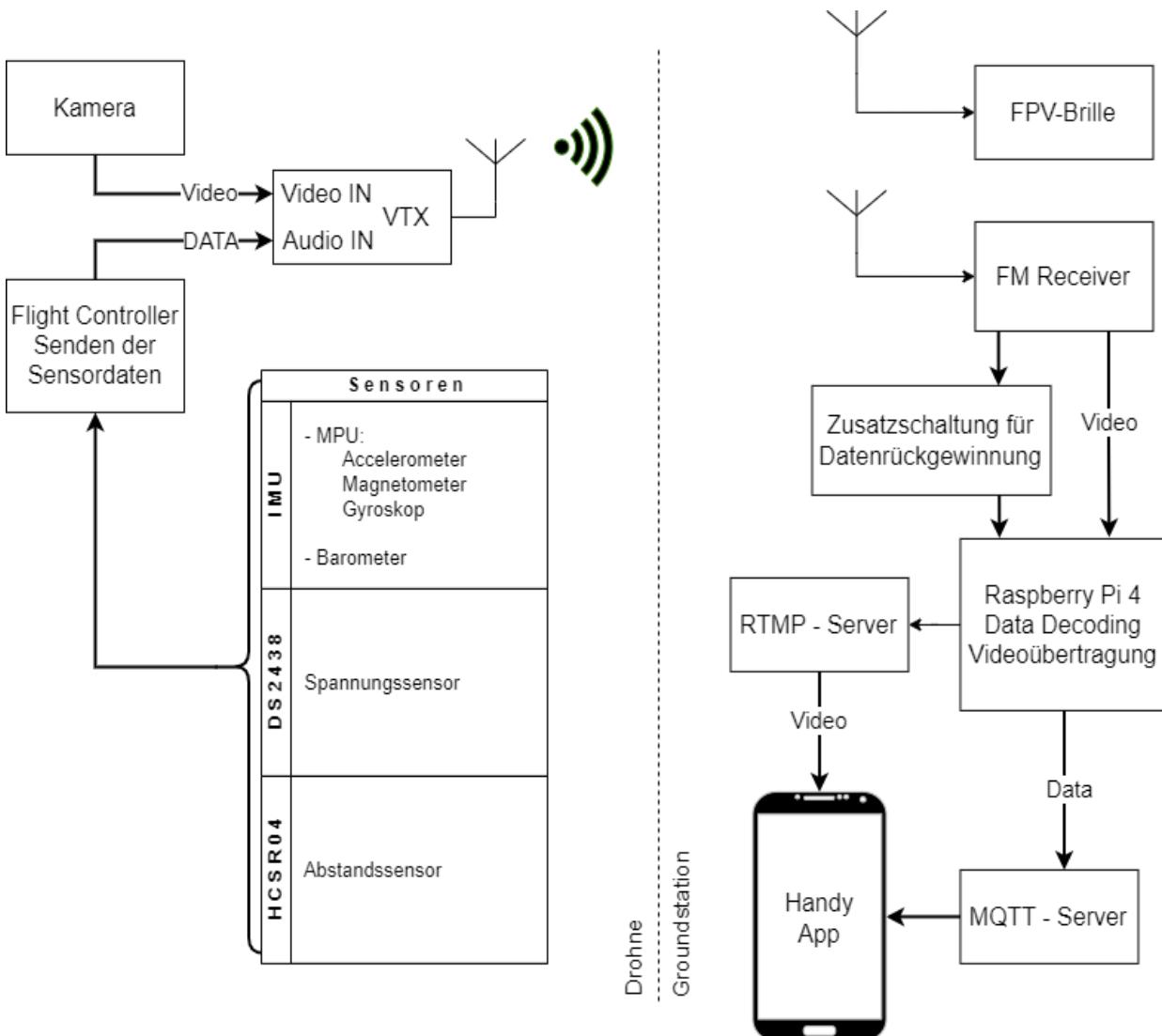


Abbildung 143: Blockschaltbild Datenübertragung

6.2 Kommunikation: Flight Controller und Sender

6.2.1 Blockschaltbild

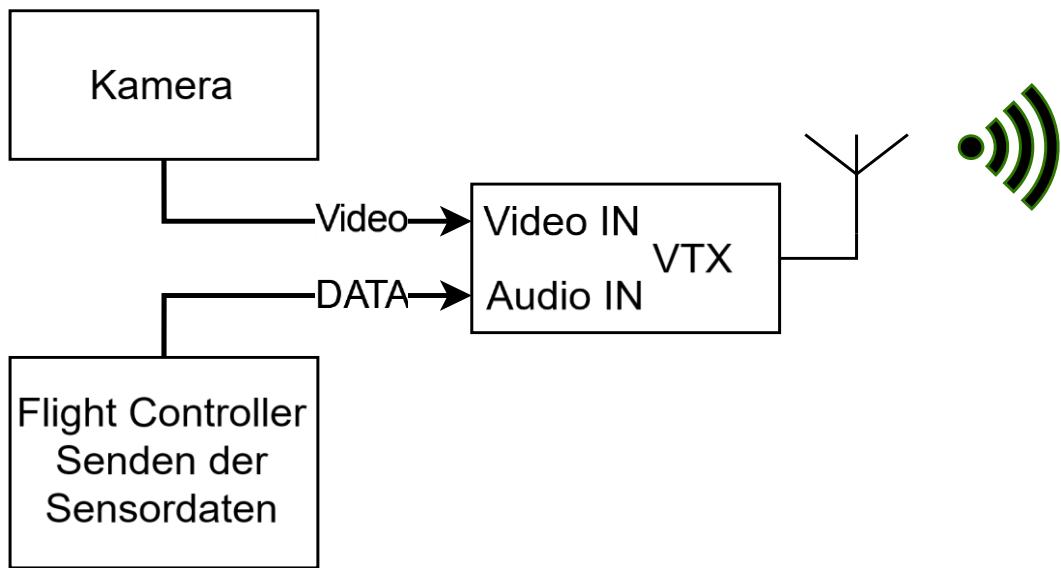


Abbildung 144: Teilblockschaltbild - Sender

6.2.2 Kamera

Die in der Drohne verbaute Kamera ist die Caddx Ratel 2, eine analoge kleine und leistungsfähige Kamera (siehe: [Kapitel 4.8](#)).

6.2.2.1 Bildkonfiguration

Um an der Kamera Einstellungen, wie Auflösung, Codierungsverfahren oder Farbeinstellungen vorzunehmen gibt es einen zusätzlichen OSD-Chip, der mitgeliefert wurde. Damit wird ein OSD-Menü im Video der Kamera angezeigt, wo dann mit Hilfe des Joy-Sticks auf dem Chip Bild-Einstellungen vorgenommen werden können. Diese Einstellungen können zum Beispiel in der FPV-Brille angeschaut werden.

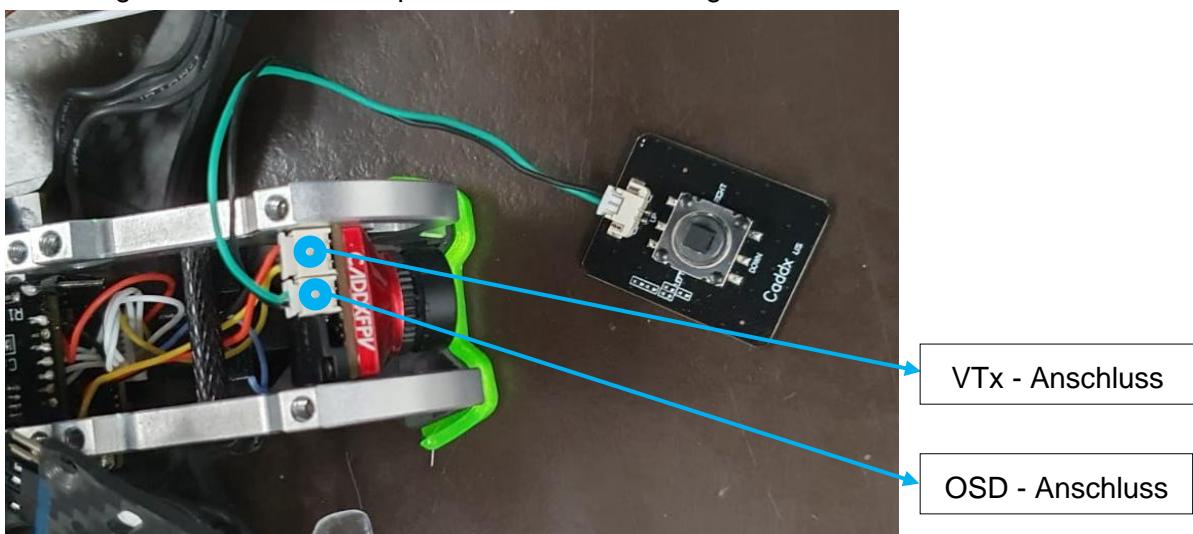


Abbildung 145: Kamera Anschlüsse

Die wichtigste Einstellung dabei ist die des Videostandards. Da man natürlich sende – und empfangsseitig denselben Codec verwenden sollte haben wir uns auf PAL geeinigt. Im gleichen Untermenü könnte auch das Format eingestellt werden, wobei wir es auf 4:3 gelassen haben.



Abbildung 146: OSD - Video-Settings

Weiters könnte man noch die Bildkorrekturen, wie Kontrast, Schärfe und Farbkorrektur einstellen, was in unserem Fall aber nicht nötig war.



Abbildung 147: OSD - Image-Settings

6.2.3 Sender – VTx

Der in der Drohne verbaute Sender: TBS UNIFY PRO 5G8 HV (FM – Sender mit 5.8 GHz) oder auch genannt VTx (Videotransmitter) ist grundsätzlich dafür gedacht das Videosignal von der Kamera an die FPV – Brille und den Empfänger zu senden. Er hat aber auch die Option Audio mitzusenden, für den Fall, dass man eine Kamera mit eingebautem Mikrofon verwendet. Da unsere Kamera kein Mikrofon hat und wir deswegen auch kein Nutzen für den Audiokanal haben, kam die Idee auf, die Messdaten über diesen Audiokanal zu übertragen, um sich einen zusätzlichen Sender und somit Platz auf der Drohne zu sparen (*Aufbau VTx, siehe: [Kapitel 4.7](#)*).



Abbildung 148: VTx

Abbildung 149: VTx - Antenne

Die Antenne wird über den SMA-Connector verbunden, dabei handelt es sich um eine Clover-Leaf-Antenne, die aus geschwungenen Kleeblattähnlichen Blättern besteht. Diese Bauform ist zirkular polarisiert, in unserem Fall LHCP (Left Hand Circular Polarized). Diese Art von Antenne hat ein sehr großflächiges Abstrahlungsverhalten, das Donut-förmig aufgebaut ist. Somit strahlt sie, bis auf den blinden Fleck über und unter der Antenne sehr gut in alle Richtungen.

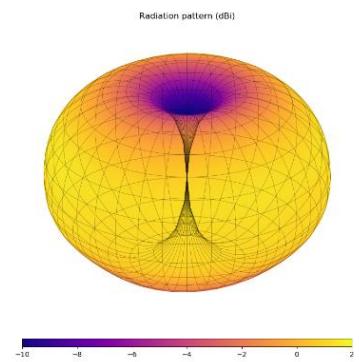


Abbildung 150: Antenne
Richtcharakteristik [RIC1]

[ANT1]

Anmerkung: Im weiteren Verlauf wird der Sender oft mit VTx abgekürzt.

6.2.3.1 VTx Einstellungen

Um Einstellungen an der VTx vorzunehmen, ist es die einfachste Methode, den Button und die LEDs als Anzeige auf der Platine zu verwenden. Dabei gibt es folgendes Untermenü:

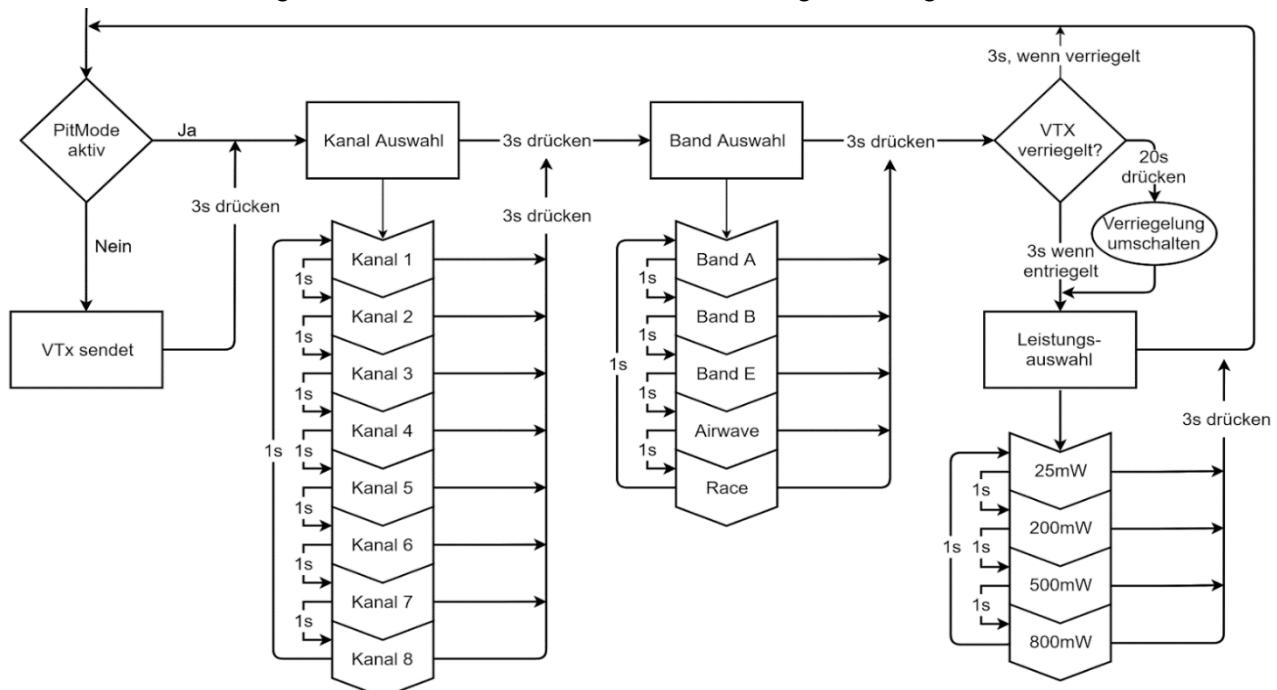


Abbildung 151: VTx – Menü [MEN1]

Um in den PitMode, also den Einstellungsmodus zu starten, muss man, nachdem die VTx mit Strom versorgt wurde, den Button für 3s halten. Um nun zu wissen in welchem Untermenü man sich gerade befindet helfen einem die 2 LEDs die, während man Einstellungen vornimmt, blinken. Blinkt die rote LED einmal in regelmäßigen Abständen, dann ist man in der Kanal-Auswahl. Mithilfe der blauen LED kann man jetzt den Kanal einstellen. Blinkt sie ebenfalls einmal in regelmäßigen Abständen, dann befindet man sich gerade im ersten Kanal. Um auf den nächsten Kanal (Kanal 2) zu wechseln, muss man den Button für 1s drücken. Will man noch einen Kanal weiter hält man den Button wieder 1s gedrückt, bis die LED 3-mal blinkt. Wenn man mit der Auswahl zufrieden ist, hält man den Button wieder 3s gedrückt und befindet sich im Band-Auswahl Menü, die rote LED sollte jetzt 2-mal blinken. Auch hier funktionieren die Einstellungen wie zuvor, in dem man anhand der Anzeige an der blauen LED die einzelnen Menüpunkte durchwandert, bis man zufrieden ist. Im letzten Untermenü könnte man die Leistung des Senders umstellen, da wir aber die benötigte Lizenz nicht haben, weil das Senden über 25mW bei 5,8GHz in Österreich illegal ist, können wir hier nichts umstellen. Beim nächsten Starten der VTx ist der PitMode wieder deaktiviert und die Einstellungen sind übernommen.

[MEN1]

Also grundsätzlich:

- 3s halten: Untermenü auswählen (rote LED)
- 1s halten: Einstellungen im Untermenü umstellen (blaue LED)

Folgende Betriebseinstellungen wurden für den Betrieb der Drohne ausgewählt:

- Kanal: 1
- Band: A
- Leistung: 25mW

6.2.3.2 VTx – Funkkanal

Der Audiokanal wurde anfangs mit einem Funktionsgenerator getestet, um Datengeschwindigkeit zu testen. Dazu wurde ein Funktionsgenerator mit einem Rechtecksignal am Audiokanal der VTx angeschlossen und der Audioausgang des Empfängers mit einem Oszilloskop dargestellt. Durch diesen Test wurde herausgefunden, dass ein Rechtecksignal, über den Audiokanal gesendet, mit steigender Frequenz am Ausgang kein Rechteck mehr darstellt, sondern immer mehr an die Form eines Sinussignals herankommt. Diese Abrundung fängt bereits bei wenigen kHz an, bei 10 kHz ist im Prinzip nur noch ein Sinussignal übrig.

Bei den folgenden Bildern ist CH1 (gelb) das Eingangssignal und CH2 (blau) das Ausgangssignal.



Abbildung 153: Rechteck mit 2kHz



Abbildung 152: Rechteck mit 10kHz

Im linken Bild (Abb. 153) kann man erkennen, dass bei einem 2kHz Rechtecksignal als Eingangssignal am Ausgang auch noch ein Rechteck wiederzuerkennen ist. Doch bereits bei 10kHz Rechtecksignal am Eingang (Abb. 152) kommt am Ausgang definitiv kein Rechteck mehr heraus.

Wenn man aber ein reines Sinussignal am Eingang anlegt und die Frequenz erhöht, wird mit steigender Frequenz aus dem Sinussignal langsam ein DC – Signal. Somit kommt man aber jedoch, statt bis 10 kHz mit einem Rechteck, bis ca. 25 kHz.

Das liegt daran, dass irgendwo zwischen Sender und Empfänger das Band begrenzt wird, in dem Fall, dass ein Tiefpass höhere Frequenzen herausfiltert. Da der Datenkanal normalerweise als Audiokanal verwendet wird, wird vermutet, dass Frequenzen ab ca. 20 kHz mit einem Tiefpass aus dem Signal gefiltert werden, da wir nur bis ~20kHz hören können und höhere Frequenzen somit unnötig sind und zusätzliche Bandbreite benötigen würden. Weil ein Rechteck im Spektrum Oberwellen, also zusätzliche unerwünschte Frequenzen vor allem über der Signalfrequenz aufweist, wird dieser auch schon früher von diesem Tiefpass begrenzt als der Sinus.

Somit gab es 2 Möglichkeiten, die Daten zu übertragen.

Erstens könnte man die Daten als Sinus in den Audioeingang der VTx senden, um mit einer höheren Frequenz Daten senden zu können, ohne dass diese durch den Kanal beeinträchtigt werden. Dafür könnte man das Datensignal ASK- oder FSK – Modulieren. Das hat jedoch den Nachteil, dass man auf der Drohne eine Zusatzschaltung benötigt, oder den ADC des Microcontrollers benutzt, um das Datensignal zu modulieren, bevor die Daten in die VTx gesendet werden.

Die Zweite Möglichkeit ist es, den Datenausgang des Flight Controllers direkt am Audioeingang der VTx anzuschließen. Somit fällt die Zusatzschaltung auf der Drohne weg, man bräuchte aber eine Zusatzbeschaltung, um aus dem empfangenen, abgerundeten, ursprünglich rechteckigen und somit verfälschtem Datensignal wieder die Daten rückzugewinnen. Die kann aber zum Beispiel mit einer einfachen Komparator-Schaltung realisiert werden.

6.2.4 ASK – Modulation

Die ASK – Modulation (Amplitude - Shift - Keying) ist ein Modulationsverfahren, bei dem die Amplitude des Trägers abhängig vom Datensignal verändert wird. OOK (On - Off - Keying) ist eine spezielle Variante der ASK – Modulation, bei der es nur 2 Zustände („0“ und „1“) gibt. Dabei repräsentiert zum Beispiel eine logische Null im Datensignal 0V im modulierten Signal. Eine logische Eins repräsentiert einen Sinus mit fixer Frequenz.

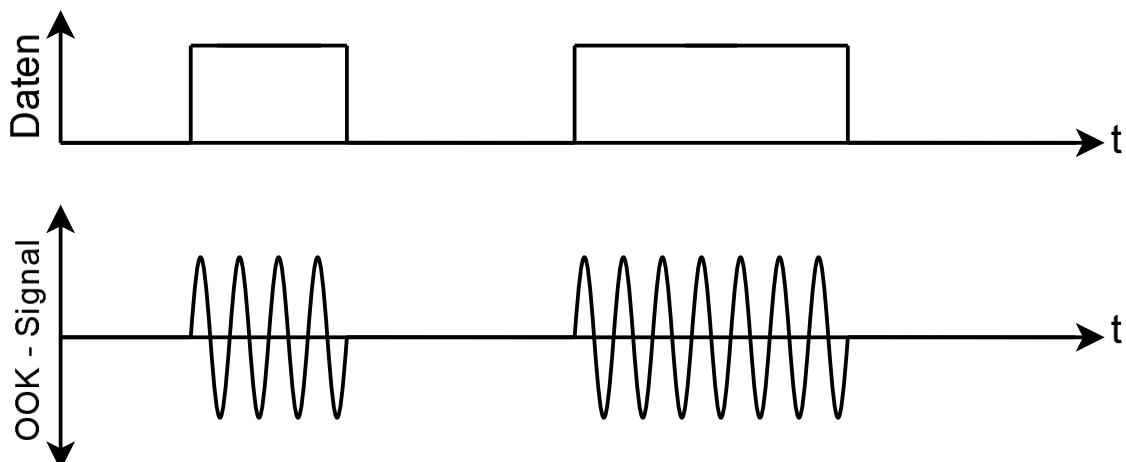


Abbildung 154: On Off Keying

Hüllkurvendemodulator

Um dieses Signal dann wieder zu demodulieren, ist eine der einfachsten Lösungen, einen Hüllkurvendemodulator zu verwenden. Diese Schaltung schneidet den negativen Anteil des Signals weg, glättet das Signal und nimmt den DC – Offset weg.

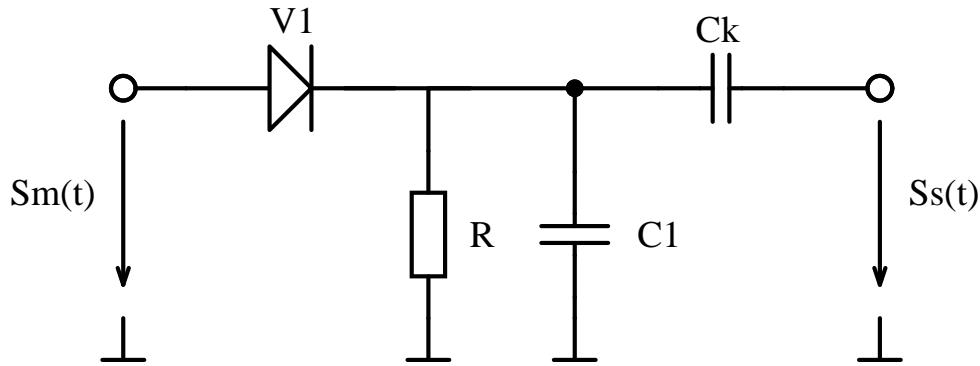


Abbildung 155: Hüllkurvendemodulator

Idealisiert schaut das resultierende Signal dann wie folgt aus (rote Linie):

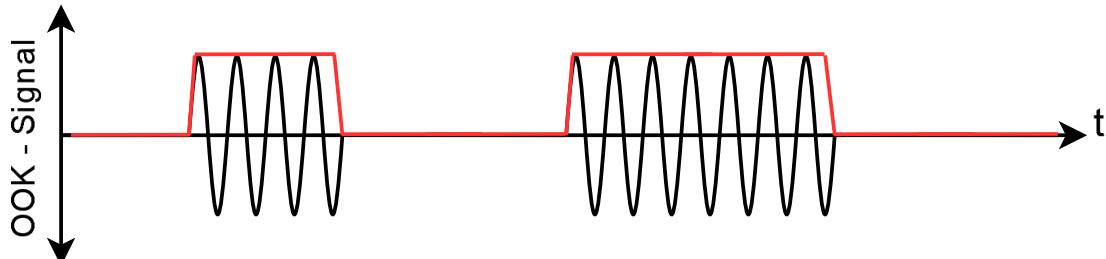


Abbildung 156: Hüllkurve

Da wir aufgrund des Tiefpassverhaltens die Frequenz des Sinussignals nicht sehr hoch ansetzen können würde das OOK – modulierte Signal dann ca. so aussehen:

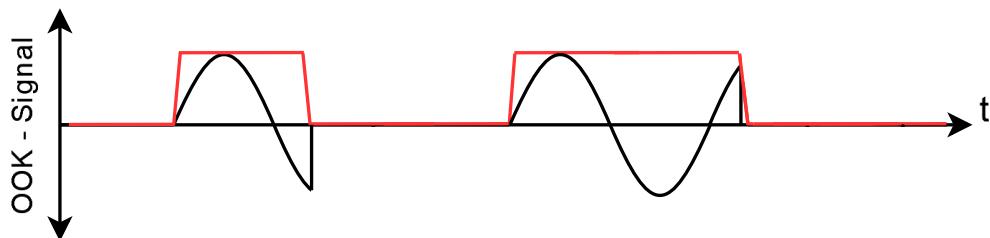


Abbildung 157: On Off Keying, Frequenz zu tief

Dies wurde auch mit einer ASK-Modulationsschaltung auf NE555-Basis getestet, die aufgrund dieses Erkenntnisses aber nicht weiterverwendet wurde.

Würde man z.B.: mindestens 5 Periodendauern pro Bit benötigen und eine Modulationsfrequenz von 10kHz verwenden:

$$\text{ASK - Bitlänge} = 5 * \frac{1}{20\text{kHz}} = 250\mu\text{s}$$

Somit käme man auf eine maximale Frequenz von: $\frac{1}{250\mu\text{s}} = 4\text{kHz}$

Wenn man das Signal jedoch direkt in den Sender hineinschickt, kann man diese 5 Periodendauern pro Bit weglassen, muss aber mit einer kleineren Baudrate senden. Nach einem kurzen Übertragungstest kam ich auf die Erkenntnis, dass man trotzdem mit einer Frequenz von 5kHz, oder sogar höher, senden kann.

Theoretisch wäre es eine noch bessere Lösung eine FSK – Modulation zu machen, bei der es den „Nullzustand“ nicht gibt, und eine logische Null ebenfalls für ein Sinussignal, nur mit anderer Frequenz, steht. Das würde ein noch schöneres Spektrum und somit eine bessere Nutzung der gegebenen Bandbreite ergeben, weil man den schlagartigen Übergang von Gleichspannung zu Sinussignal nicht hat. Bei der FSK – Modulation hätte man aber dasselbe Problem, wie bei der ASK – Modulation und hätte zusätzlich einen höheren Schaltungsaufwand und Platzverbrauch.

Letztendlich habe ich mich also für die etwas unkonventionelle Art der Datenübertragung entschieden in dem die „rohen“ Daten direkt in den Sender geschickt werden.

6.2.5 Datenübertragung – Programm auf Cortex µC

6.2.5.1 Verwendete Methode der Datenübertragung

Die Sensordaten werden zuerst in Gleitkommadarstellung (siehe: [Kapitel 6.2.5.2](#)) und dann mit UART (siehe: [Kapitel 6.2.5.3](#)) ausgegeben. Dieser Daten-Stream wird dann direkt in den Audiokanal der VTx geleitet, wo er dann gesendet wird.

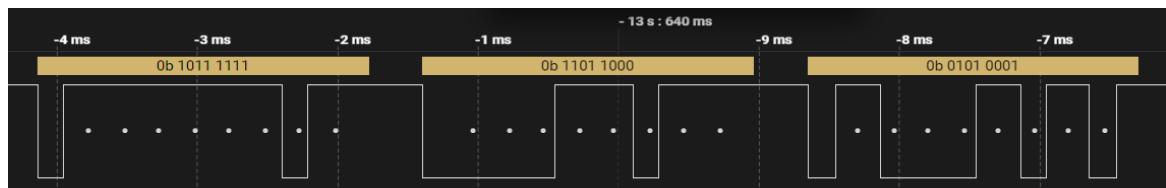


Abbildung 158: Beispielhafter Datenstream

6.2.5.2 Gleitkommadarstellung (memcpy)

Die Messdaten die Übertragen werden sind Kommazahlen (float).

Das sind zum Beispiel:

- Lagewinkel der Drohne (Pitch, Roll, Yaw)
- Akkusspannung
- Entfernung zum Boden

Um diese aber übertragen zu können, müssen die Werte binär, also nur mit 1 oder 0 (HIGH / LOW) dargestellt werden. Dabei gibt es zwei Möglichkeiten, die Zahlen darzustellen, nämlich mit der Fixkommadarstellung und der Gleitkommadarstellung.

Die Fixkommadarstellung hat eine bestimmte Anzahl an Bits vor dem Komma und eine bestimmte Anzahl an Bits nach dem Komma. Das hat zwar den Vorteil, dass die Fixkommadarstellung sehr leicht realisierbar ist, dafür kann durch die fixe Kommaposition die Genauigkeit nicht verändert werden, was sich je nach Zahl unterschiedlich stark auf das Ergebnis auswirkt.

Bei der Gleitkommadarstellung hingegen kann die Kommaposition je nach Anforderung und für jede Zahl so angepasst werden, dass die Genauigkeit nicht so stark unter der Umwandlung leidet.

Man teilt die Zahl für die Umwandlung in Exponent und Mantisse ein.

$$\text{Zahl} = \text{Mantisse} * \text{Basis}^{\text{Exponent}} \quad \text{Basis}_{\text{Binär}} = 2$$

Dabei steht die Mantisse für die Zahl, nachdem das Komma verschoben wurde und der Exponent für die Zahl, wie oft das Komma verschoben wurde. Die Basis ist in unserem Fall 2, weil wir die Zahlen in die binäre Schreibweise umwandeln. Zur Darstellung von negativen Zahlen wird normalerweise ein zusätzliches Sign-Bit verwendet, bei dem eine „0“ einer positiven Zahl und eine „1“ einer negativen Zahl entspricht.

Der einfacheren Handhabung wurde für das Umwandeln der Zahlen die memcpy-Funktion (32-Bit Gleitkommadarstellung) in C verwendet. Diese verwendet den IEEE-754 Standard, der die Gleitkommadarstellung für 32-Bit und 64-Bit festlegt.

32-Bit Gleitkommadarstellung nach dem IEEE-754 Standard:

Sign-Bit (1-Bit)	Exponent (8-Bit)	Mantisse (23-Bit)
------------------	------------------	-------------------

Somit kann ein Zahlenbereich von $1,4 * 10^{-45}$ bis $3,403 * 10^{38}$ dargestellt werden.

Da der Exponent aus 8 Bit besteht, muss noch ein „Bias“ oder Offset von $2^7 - 1 = 127$ zum Exponenten addiert werden.

[GKD1]

Die memcpy Funktion ist grundsätzlich eine Funktion, mit der man zum Beispiel Daten aus einer Variable in eine andere Zielvariable kopieren kann. Sie kann aber auch zwischen verschiedenen Datentypen kopieren. Und kopiert man eine Float-Zahl in eine Int-Variable, dann wird sie automatisch in Gleitkommadarstellung umgerechnet.

Syntax:

memcpy(&Ziel, &Quelle, Anzahl der Bytes);

Programmbeispiel:

```
float buffer_float = 1.23;
int32_t buffer_int;
memcpy(&buffer_int, &buffer_float, sizeof buffer_int);
```

In buffer_int steht somit die 32-Bit Gleitkommadarstellung der Zahl 1,23.

6.2.5.3 UART

UART (Universal Asynchronous Receiver Transmitter) ist eine Hardwarekomponente, die eine serielle Kommunikation zwischen Komponenten ermöglicht und das Übertragen der Daten übernimmt. Für die UART-Kommunikation wird die von UART bereitgestellte V24-Schnittstelle verwendet, die die Verbindung der Teilnehmer und die Kommunikation der Daten festlegt. Für die Kommunikation werden 2 Leitungen benötigt, nämlich eine Tx-Leitung (Transmit) und eine Rx-Leitung (Receive). Dabei muss der Tx-Port des Senders am Rx-Port des Empfängers angeschlossen werden und umgekehrt, damit die Daten empfangen werden können. Im Gegensatz zu USART unterstützt UART nicht die Möglichkeit die Daten synchron zu übertragen, deswegen gibt es auch keine Takt-Leitung.

Während keine Daten übertragen werden, liegt die Daten-Leitung standardmäßig auf HIGH. Damit sich Sender und Empfänger synchronisieren können, wird ein Start-bit („0“) gesendet. Danach kommen die Datenbits. In den meisten Fällen und auch bei uns sind das 8 Datenbits. Zum Ende der Übertragung wird dann ein Stop-Bit gesendet, dass das Ende eines Bytes markiert. Man könnte vor dem Stop-Bit noch ein Parity-Bit senden, dass anzeigt, ob die Summe der übertragenen Bits gerade oder ungerade ist. Dadurch kann man Überprüfung ob ein Bit bei der Übertragung umgefallen ist. Dies funktioniert jedoch nur bei 1-Bit Fehlern.

Für die Datenübertragung wurde USART3 im asynchronen Modus verwendet (siehe: [Kapitel 5.3](#))

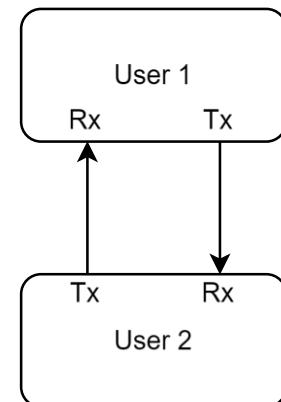


Abbildung 159: UART - Kommunikation

Um nach dem Start-Bit nun wirklich genau die Daten auszulesen, die auch gesendet wurden, müssen Sender und Empfänger mit der gleichen Geschwindigkeit schreiben/lesen. Diese Geschwindigkeit wird mit der Baudrate angegeben, die im Prinzip die Frequenz des Signals angibt.

Protokollaufbau:



Abbildung 160: V24 - Protokoll

Somit liegt ein Bit für $\frac{1}{\text{Baudrate}}$ s an.

Mit 8 Bit und keinem Parity-Bit würde ein Byte $\frac{10}{\text{Baudrate}}$ s zum Übertragen benötigen.

Bei der Übertragung von Binärzahlen unterscheidet man auch noch zwischen LSB (Least Significant Bit) first und MSB (Most Significant Bit) first. Mit MSB first werden die Bits von links nach rechts nacheinander gesendet, man fängt also mit dem höchstwertigen Bit an. Bei LSB first verläuft die Übertragung genau anders herum.

Bsp.: 57 → 0011 1001_b

MSB first: 0011 1001_b

LSB first: 1001 1100_b

Bei dieser Beispielhaften UART-Übertragung wird die Zahl 156 mit LSB first gesendet.

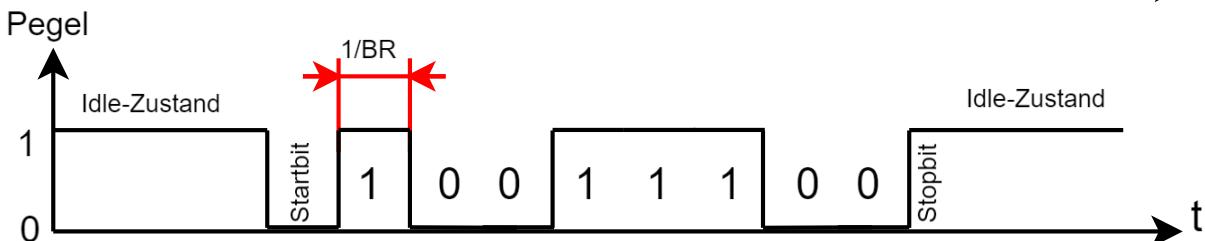


Abbildung 161: UART - Beispiel

UART-Einstellungen bei der Datenübertragung:

- 8 Daten-Bits
- Kein Parity-Bit
- 1 Stop-Bit
- LSB first
- Baudrate = 4800

6.2.5.4 Programm

Flussdiagramm

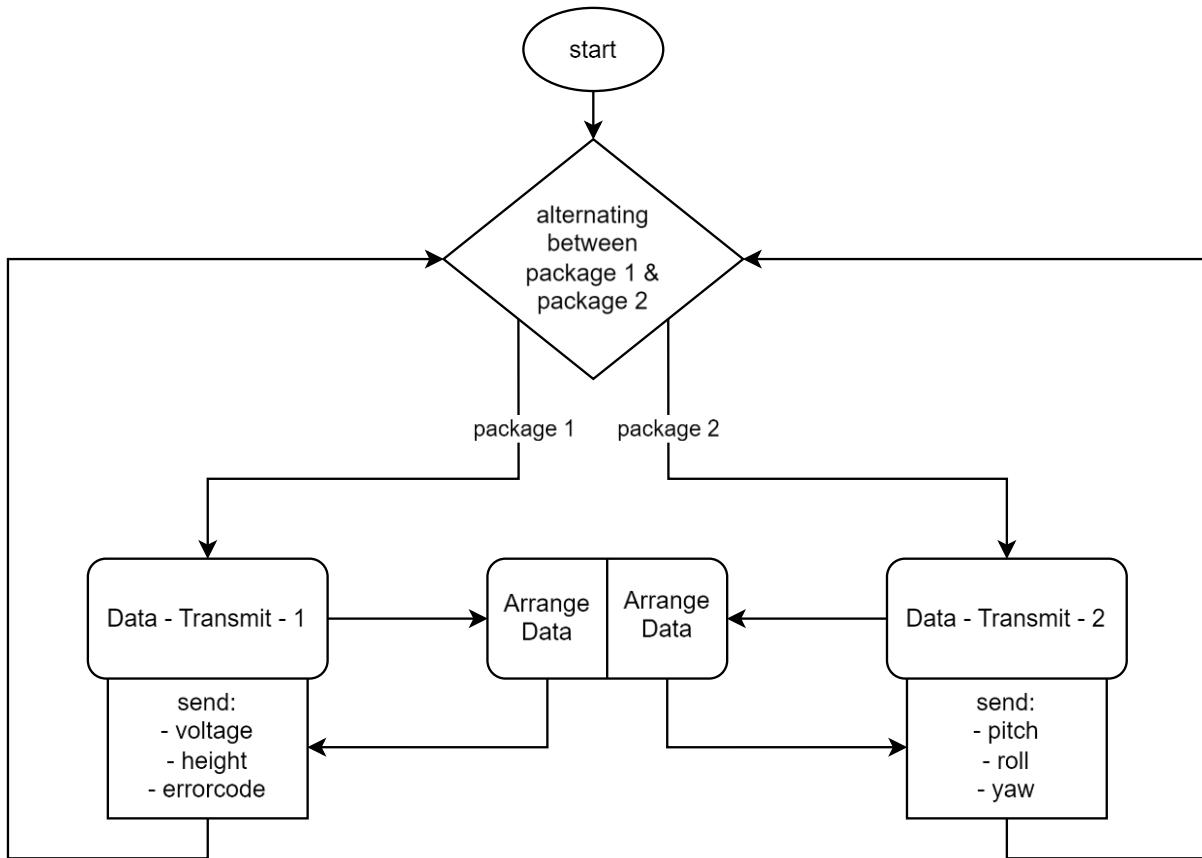


Abbildung 162: Flussdiagramm Senden der Daten

Beim Senden der Daten werden abwechselnd Packet 1 und Packet 2 gesendet. Dafür werden die Daten in Gleitkommadarstellung umgewandelt und über UART gesendet.

Code

Datei: datatransmission.c – Funktion: DATA_INIT()

```

...
UART_HandleTypeDef *DATA_OUTPUT;

void DATA_INIT(UART_HandleTypeDef *huart)
{
    DATA_OUTPUT = huart;
}
...
  
```

In der DATA_INIT-Funktion wird UART3 (wird beim Aufruf aus dem Hauptprogramm übergeben) als Pointer übergeben und in DATA_OUTPUT gespeichert. Somit kann DATA_OUTPUT (global) in den anderen Funktionen zum Senden für UART verwendet werden.

Datei: datatransmission.c – Funktion: DATA_ARRANGEMENT()

```

...
uint8_t transmission_buffer[20];

...
/** 
 * @brief This function prepares the data for transmission^
 * @retval None
 * @param value the float number being converted
 * @param j 4-Byte-number, e.g. 1 --> first for bytes (after sync-byte), 2 - -> second 4 bytes
 */
void DATA_ARRANGEMENT(float value, int8_t j)
{
    int32_t datatosend;

    j--;

    // convert int into floating point number with 32 bits
    memcpy(&datatosend, &value, sizeof(datatosend));

    // arrange buffer: separate bytes, save them in single buffer positions
    transmission_buffer[1+4*j] = (int8_t)(datatosend >> 24);
    transmission_buffer[2+4*j] = (int8_t)((datatosend >> 16) & 0xFF);
    transmission_buffer[3+4*j] = (int8_t)((datatosend >> 8) & 0xFF);
    transmission_buffer[4+4*j] = (int8_t)(datatosend & 0xFF);
}

```

Die Funktion DATA_ARRANGEMENT bereitet die Daten auf das Senden vor. Dabei werden die Kommazahlen zuerst in ihre Gleitkommadarstellung umgewandelt und dann an den durch die Variable j zugeordneten Stellen hintereinander in einem Array (transmission_buffer) gespeichert. Diese Funktion wird dann für jede Zahl aufgerufen, bis der Array für ein Datenpacket befüllt ist.

Datei: datatransmission.c – Funktion: DATA_TRANSMISSION_1()

```

/**
 * @brief This function transmits the general data to the port where Audio-In (VTx) is connected, from Uart3 PORT PB10
 * @retval None
 */
void DATA_TRANSMISSION_1(float voltage, float grounddistance, int8_t error_code)
{
    int8_t syncbyte = 0x00;

    // add synchronisation byte
    transmission_buffer[0] = syncbyte;
    // arrange battery-voltage

```

```

DATA_ARRANGEMENT(voltage, 1);
// arrange ground distance value
DATA_ARRANGEMENT(grounddistance, 2);
// add errorcode
transmission_buffer[9] = error_code;

// send package 1 (sync-byte, battery voltage, ground distance,
errorcode)
HAL_UART_Transmit_DMA(DATA_OUTPUT, transmission_buffer, 10);
}

```

Die Funktion DATA_TRANSMISSION_1 ordnet die Daten im Array (transmission_buffer), der letztendlich gesendet wird, an. Das erste Datenpacket beinhaltet die Akku-Spannung, die Flughöhe und einen Error-Code. Um das Packet beim Empfangen zu identifizieren, wird ein Synchronisations-Byte vor den Daten mit dem Inhalt: 00000000b gesendet.

Aufbau Packet 1:

buff[0]	buff[1]	buff[2]	buff[3]	buff[4]	buff[5]	buff[6]	buff[7]	buff[8]	buff[9]
Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8	Byte 9	Byte 10
Sync-Byte	Akkuspannung					Flughöhe			Error-Byte

Nachdem der Array richtig gefüllt ist, werden die Daten mit UART gesendet. Diese Aufgabe wird an den DMA-Controller übergeben, der parallel zum Prozessor arbeiten kann. Somit kann sich der Prozessor auf die wesentliche Aufgabe, nämlich das Fliegen der Drohne konzentrieren.

Datei: datatransmission.c – Funktion: DATA_TRANSMISSION_2()

```

/** 
 * @brief This function transmitts the MPU data to the port where Audio-In
(VTx) is connected, from Uart3 PORT PB10
 * @retval None
 */
void DATA_TRANSMISSION_2(float pitch, float roll, float yaw)
{
    int8_t syncbyte = 0xFF;

    // add synchronisation byte
    transmission_buffer[0] = syncbyte;
    // send pitch value
    DATA_ARRANGEMENT(pitch, 1);
    // send roll value
    DATA_ARRANGEMENT(roll, 2);
    // send yaw value
    DATA_ARRANGEMENT(yaw, 3);

    // send package 2 (sync-byte, pitch value, role value, yaw value)
    HAL_UART_Transmit_DMA(DATA_OUTPUT, transmission_buffer, 13);
}

```

In der Funktion DATA_TRANSMISSION_2 werden die Daten im Array (transmission_buffer), der letztendlich gesendet wird, angeordnet. Das zweite Datenpacket beinhaltet den Pitch-, den Roll- und den Yaw-Wert. Um das Packet beim Empfangen zu identifizieren, wird ein Synchronisations-Byte vor den Daten mit dem Inhalt: 11111111b gesendet.

Aufbau Packet 2:

buff[0]	buff[1]	buff[2]	buff[3]	buff[4]	buff[5]	buff[6]	buff[7]	buff[8]	buff[9]	buff[10]	buff[11]	buff[12]
Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8	Byte 9	Byte 10	Byte 11	Byte 12	Byte 13
Sync-Byte	Pitch				Roll				Yaw			

Der verwendete Port PB10 mit UART, sowie der DMA wurden mit STM32CubeMX initialisiert.

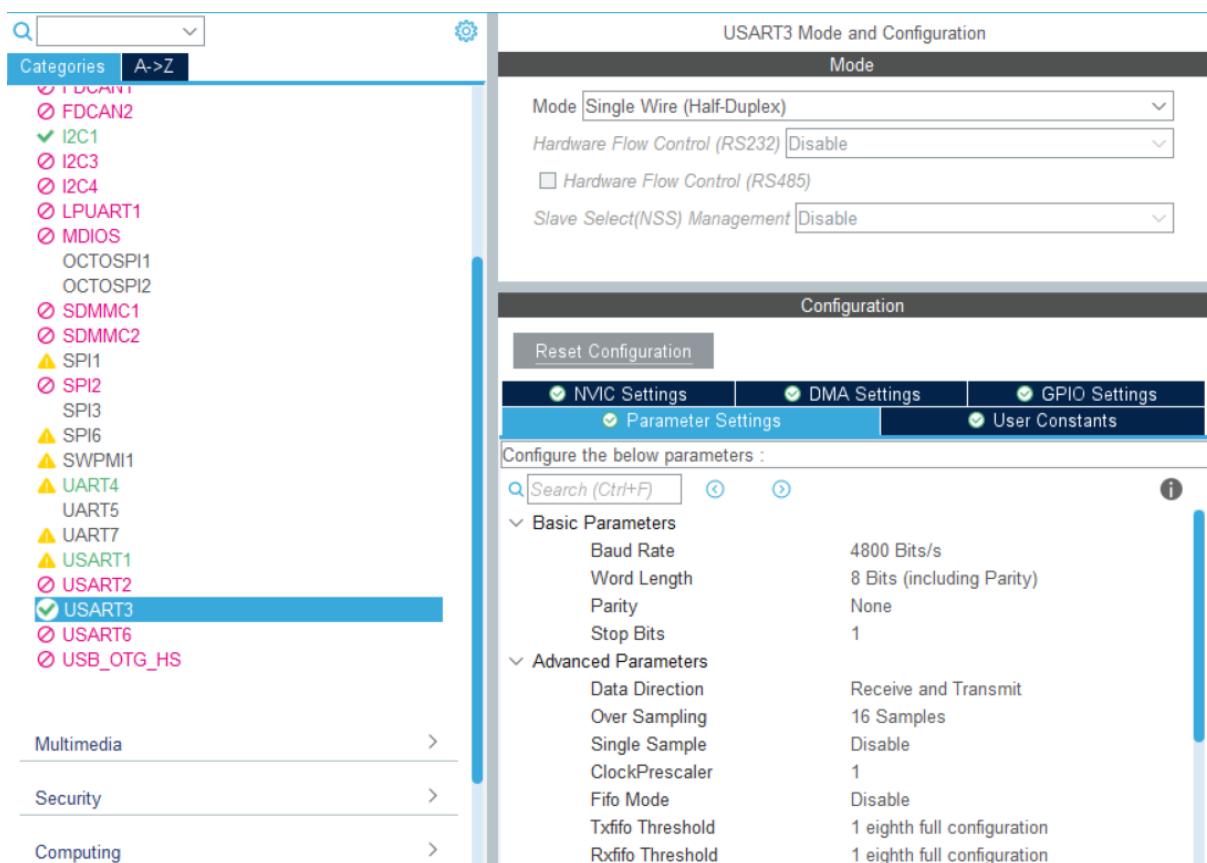


Abbildung 163: CubeMX USART3 Einstellungen

UART wurde also mit einer Baudrate von 4800, 8 Datenbits und keinem Parity-Bit initialisiert. Somit muss das Empfangsprogramm dieselben Einstellungen vorweisen, damit die Daten richtig empfangen werden.

Zunächst muss der DMA hinzugefügt werden, damit das Datensenden weniger Zeit in Anspruch nimmt

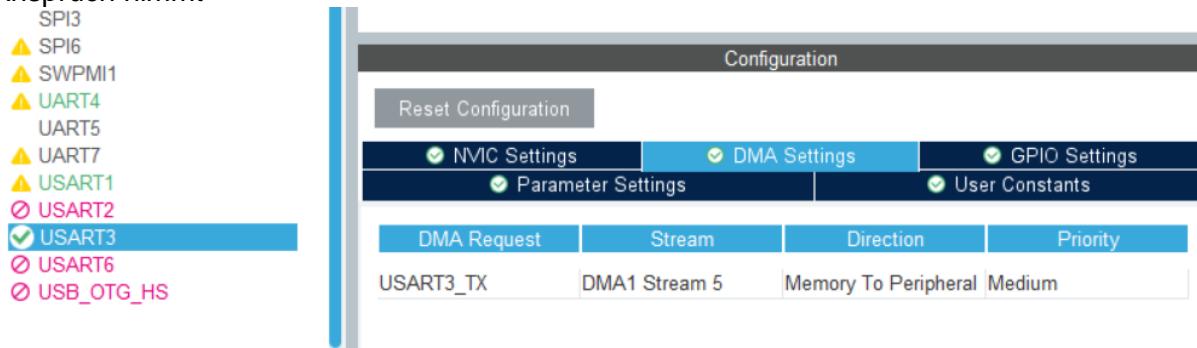


Abbildung 164: USART3 DMA Einstellungen

Zusätzlich muss man noch den Global Interrupt für USART3 aktivieren:

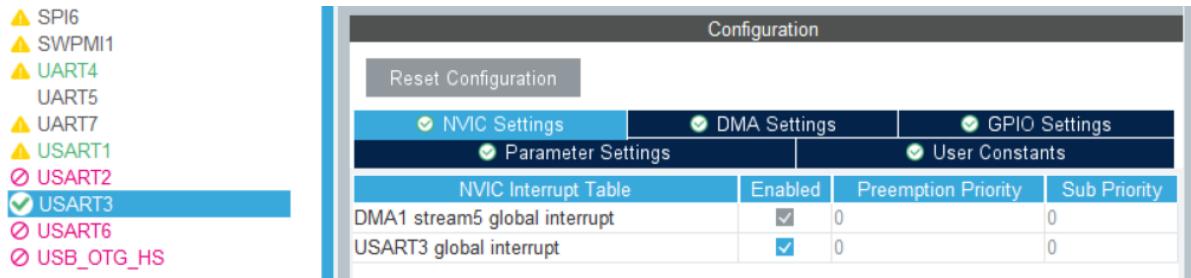


Abbildung 165: USART3 Global Interrupt aktivieren

Die Funktionen zum Senden der Daten werden dann im Interrupt ausgeführt, wo auch die Sensordaten übergeben werden.

Datei: main.c – Funktion: RealTimeSystemCallback

```
// data transfer
static int8_t dataTransmitDelay = 0;

if(dataTransmitDelay++ >= 60)
{
    static int8_t packetSelect = 0;
    if(packetSelect == 0)
        DATA_TRANSMISSION_1(ds2438_Voltage, baroAltitude, 0x00);
    else
        DATA_TRANSMISSION_2(angle.pitch, angle.roll, angle.yaw);

    packetSelect = packetSelect == 0;
    dataTransmitDelay = 0;
}
```

Hier werden abwechselnd Packet 1 und Packet 2 gesendet. Da der Interrupt öfter aufgerufen wird (alle ~8ms), als wir Daten senden möchten, wurde mit dataTransmitDelay eine Verzögerung eingebaut (wird jedes 60. mal ausgeführt). Durch dieses Delay kommt die gewünschte Sendepause zwischen den Paketen zustande, damit das Schwingverhalten die Daten nicht stört (siehe: [Kapitel 6.2.6](#)).

6.2.6 Testen der Übertragung

Der Code wurde nach mehreren Versionen und Tests so geschrieben, wie er jetzt endgültig ist. Anfangs war die Idee die Daten als ununterbrochenen Datenstream zu senden, wobei aber sehr schnell Probleme aufgetreten sind. Dadurch, dass in die VTx Signale geschickt werden, die sehr schlagartige Signaländerungen aufweisen (Rechtecksignale), beginnt das übertragene Signal zu schwingen. Zusätzlich wird das Signal beim Sender oder Empfänger invertiert, was aber eigentlich kein Problem ist, weil man bei der Datenrückgewinnung lediglich die Daten wieder zurück invertieren muss.

In Blau wird das Eingangssignal der VTx und in Gelb das Ausgangssignal des Empfängers dargestellt:

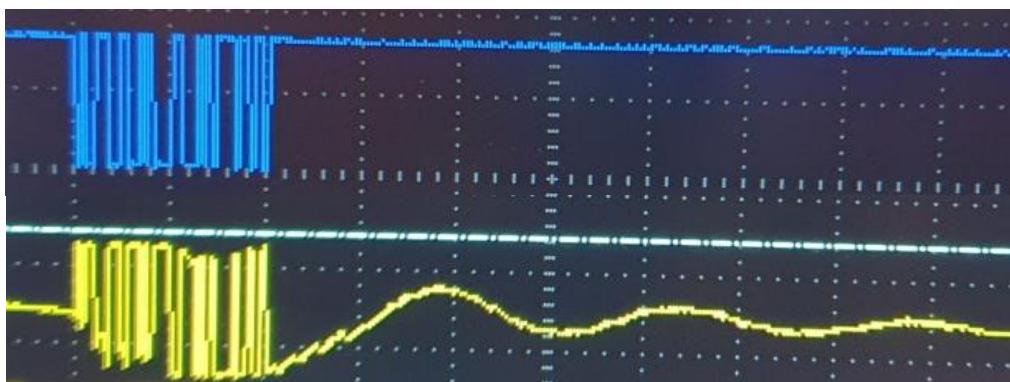


Abbildung 166: Datenempfangsmessung

Durch dieses Schwingverhalten, das aufgrund der Art der Übertragung der Daten zustande kommt, kann man die Daten also nicht wie gewollt in einem ununterbrochenen Datenstream senden, sondern muss das Signal „ausschwingen“ lassen, da ansonsten die Amplitude der Schwingung immer größer wird, bis das Signal letztendlich zu stark davon verfälscht wird. Das bedeutet im Klartext einfach, dass man, nachdem ein Datenpaket gesendet wurde, eine gewisse Zeit warten muss, bis das Signal nicht mehr schwingt. Darum werden in der jetzigen Version der Datenübertragung die Sensordaten in 2 zusammengefassten Paketen gesendet zwischen denen eine Wartezeit von 50 – 200ms gewartet werden sollte (Abhängig von Übertragungsrate). Um die Pakete auseinanderzuhalten, wird ein Synchronisation-Byte am Anfang des Pakets gesendet.

Paket 1 (10 Byte):

- 1 Byte: Sync-Byte: 0x00
- 4 Byte: Akku-Spannung
- 4 Byte: Höhe
- 1 Byte: Error-Codes

Paket 2 (13 Byte):

- 1 Byte: Sync-Byte: 0xFF
- 4 Byte: Pitch-Wert
- 4 Byte: Roll-Wert
- 4 Byte: Yaw-Wert

Folgende Bilder wurden am Ausgang des Empfängers gemessen.

Baudrate = 4800

Abstand zwischen Paketen: ~ 170ms

In der nachstehenden Abbildung (Abb. 167) erkennt man sehr gut das Schwingverhalten des Signals, dass nachdem nicht mehr gesendet wird langsam abklingt.

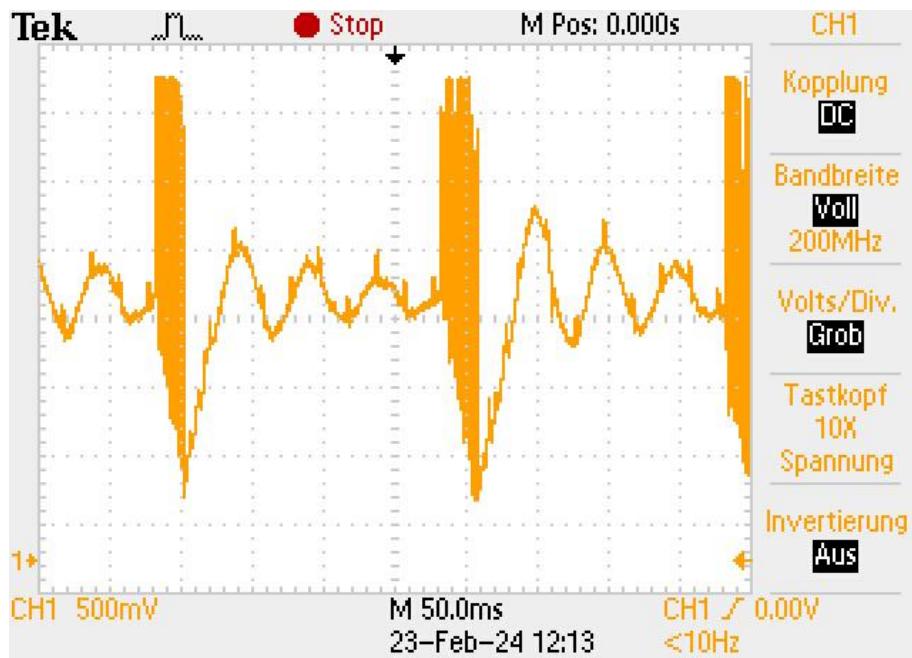


Abbildung 167: Schwingverhalten des Receivers

Auch muss beachtet werden, dass die Signale durch die Übertragung stark verfälscht werden (siehe: [Kapitel 6.2.3.2](#)), weshalb zwischen dem Empfänger und dem Raspberry Pi auch eine OPV-Stufe zur Datenrückgewinnung benötigt wird (siehe: [Kapitel 6.4.4](#)).



Abbildung 168: Datenverfälschung der empfangenen Daten

Zum Schreiben und Testen des Datenübertragungsprogramms wurde auf dem Cortex M3 (STM32f103RB) programmiert. Dabei hat alles auch gut funktioniert. Jedoch bin ich bei der Implementierung des Codes auf den wirklichen Flight-Controller Cortex M7 (STM32H7A3RGT6) auf ein merkwürdiges Problem gestoßen. Wenn dieser an der VTx angeschlossen und das System mit Spannung versorgt war, wurde das UART-Signal noch vor dem Senden verfälscht. Dabei wurden die Daten zwar richtig ausgegeben, der Idle Zustand wurde aber immer auf Masse gezogen. Weil das Start-Bit ebenfalls ein LOW-Bit ist und somit von Idle-LOW auf das Startbit keine Flanke erkannt wird, wird das erste Byte falsch oder überhaupt nicht eingelesen. Das obere Signal ist eine Aufnahme direkt aus dem Cortex M3 und das untere eine Messung, direkt aus dem Ausgang des Cortex M7. Bei beiden war die VTx angeschlossen.

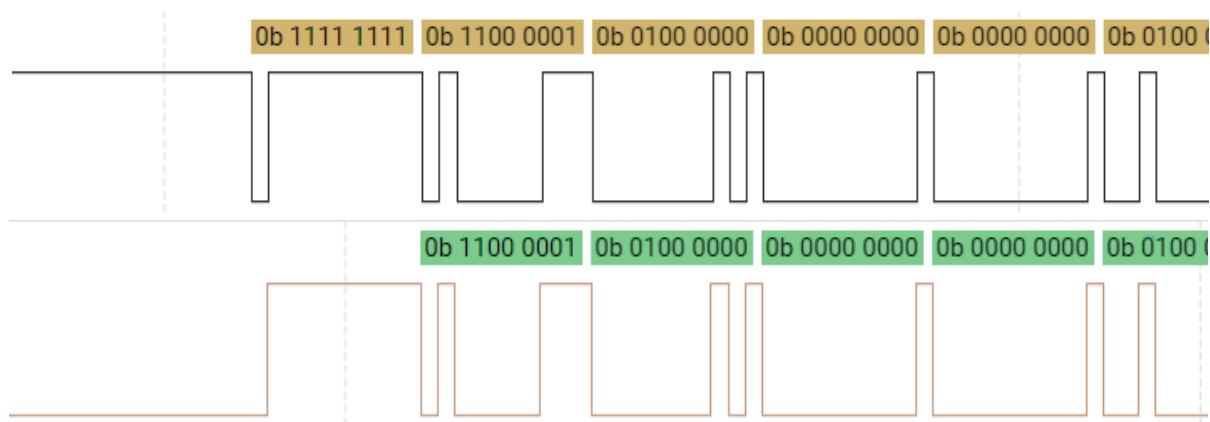


Abbildung 169: Idle Low Fehler

Zuerst wurde vermutet, dass auf der Flight-Controller-Platine ein Schaltungsfehler vorliegt, nach längerem Recherchieren, kamen ich aber zu einer Programm-technischen Lösung. Nach einer Person im Internet, die mit genau demselben Problem gekämpft hat, musste man in dem von Cube-MX generierten UART-INIT zwei Zeilen ändern.

[UAR1]

Einmal wechselt man Von `UART_MODE_TX` auf `UART_MODE_TX_RX`. Man schaltet also den Receive-Kanal auch an, obwohl er nicht benutzt wird. Zweitens muss in der ersten Error-Abfrage die Funktion `HAL_HalfDuplex_Init` gegen `HAL_UART_Init` ausgetauscht werden.

main.c (MX_USART3_UART_Init) - old

```

...
huart3.Instance = USART3;
huart3.Init.BaudRate = 5000;
huart3.Init.WordLength = UART_WORDLENGTH_8B;
huart3.Init.StopBits = UART_STOPBITS_1;
huart3.Init.Parity = UART_PARITY_NONE;
huart3.Init.Mode = UART_MODE_TX;
huart3.Init.HwFlowCtl = UART_HWCONTROL_NONE;
huart3.Init.OverSampling = UART_OVERSAMPLING_16;
huart3.Init.OneBitSampling = UART_ONE_BIT_SAMPLE_DISABLE;
huart3.Init.ClockPrescaler = UART_PRESCALER_DIV1;
huart3.AdvancedInit.AdvFeatureInit = UART_ADVFEATURE_NO_INIT;
if (HAL_HalfDuplex_Init(&huart3) != HAL_OK)
{
    Error_Handler();
}
...

```

main.c (MX_USART3_UART_Init) - new

```

...
huart3.Instance = USART3;
huart3.Init.BaudRate = 5000;
huart3.Init.WordLength = UART_WORDLENGTH_8B;
huart3.Init.StopBits = UART_STOPBITS_1;
huart3.Init.Parity = UART_PARITY_NONE;
huart3.Init.Mode = UART_MODE_TX_RX;
huart3.Init.HwFlowCtl = UART_HWCONTROL_NONE;
huart3.Init.OverSampling = UART_OVERSAMPLING_16;
huart3.Init.OneBitSampling = UART_ONE_BIT_SAMPLE_DISABLE;
huart3.Init.ClockPrescaler = UART_PRESCALER_DIV1;
huart3.AdvancedInit.AdvFeatureInit = UART_ADVFEATURE_NO_INIT;
if (HAL_UART_Init(&huart3) != HAL_OK)
{
    Error_Handler();
}
...

```

Danach war das Problem, dass der Idle-Zustand auf Masse gezogen wurde, beseitigt.

Anmerkung: Der UART-Modus: UART_MODE_TX_RX kann direkt in CubeMX eingestellt werden und muss daher nicht händisch geändert werden. Die Funktion HAL_UART_Init muss jedoch händisch abgeändert werden.

6.3 Kommunikation: Sender und Empfänger

6.3.1 Blockschaltbild

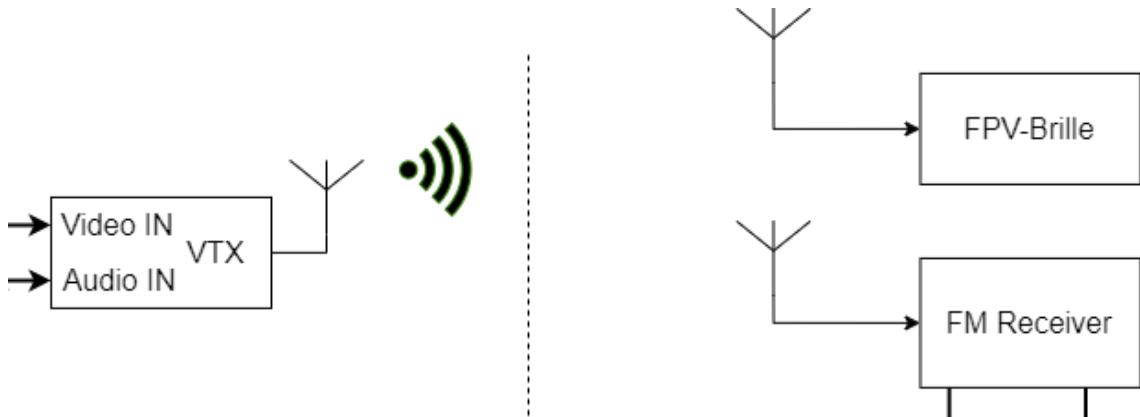


Abbildung 170: Blockschaltbild Sender - Empfänger

6.3.2 Aufbau Empfängermodul

Mit dem SKYDROID Empfängermodul können Video- und Datensignal an der Groundstation wieder empfangen werden. Die Daten werden im Modul bereits demoduliert und können an den dafür vorgesehenen Pins (Audio und Video) abgegriffen werden. Dafür muss es mit 5V versorgt werden. Grundsätzlich wird es wie im Anschlussdiagramm (siehe: [Kapitel 6.4.2](#)) angeschlossen.



Abbildung 171: Empfängermodul

6.3.3 Verbindungsauflaufbau

Um den Receiver auf den richtigen Channel einzustellen, auf dem übertragen wird, kann man entweder den Button auf der Vorderseite drücken, um jeden Channel einzeln durchzugehen, bis man den richtigen gefunden hat, oder man hält ihn einfach gedrückt, um eine automatische Suche zu starten, mit der der Channel mit dem stärksten Signal gefunden werden kann.

6.3.4 Testen der Übertragung

Zum Testen der Übertragung wurde anfangs die Handy-App „USB-Camera“ verwendet, mit der man das empfangene Videosignal anzeigen kann. Somit wurde die Videoübertragungskette (Kamera – VTx – Empfänger) grundsätzlich getestet. Dafür muss man sich einfach die App herunterladen und sein Handy mit einem USB-Kabel mit dem Empfänger verbinden.

Später wurde die Videoübertragung sowie die Datenübertragung mit Oszilloskop-Messungen und über den Raspberry Pi getestet.

[APP1]

6.3.5 Videoübertragung zu FPV – Brille

Damit der Pilot während des Fluges die Drohne besser steuern kann und immer sieht, wo die Drohne gerade ist, wird das Kamera-Bild direkt an die FPV-Brille gesendet. Diese empfängt das analoge Signal über eine eingebaute Antenne (Patch-Antenne) und ist somit unabhängig vom Empfänger. Dadurch ist das Video, das in der FPV-Brille angezeigt wird, auch von Störungen und Verzögerungen, die nach dem Empfängermodul entstehen nicht betroffen. Deswegen kommt das Video schnell an und für das Auge ist keine Verzögerung zu spüren. Deswegen ist sie für den Piloten ideal um den Flug mitzuverfolgen.

Um den richtigen Channel zum Empfangen auszuwählen, drückt man die S-Taste auf der Brille und kann dann mit den Pfeiltasten durch die Kanäle wechseln und den gewünschten auswählen.



Abbildung 172: FPV-Brille Vorderansicht



Abbildung 173: FPV-Brille Seitenansicht

6.4 Kommunikation: Empfänger und Raspberry Pi

6.4.1 Blockschaltbild

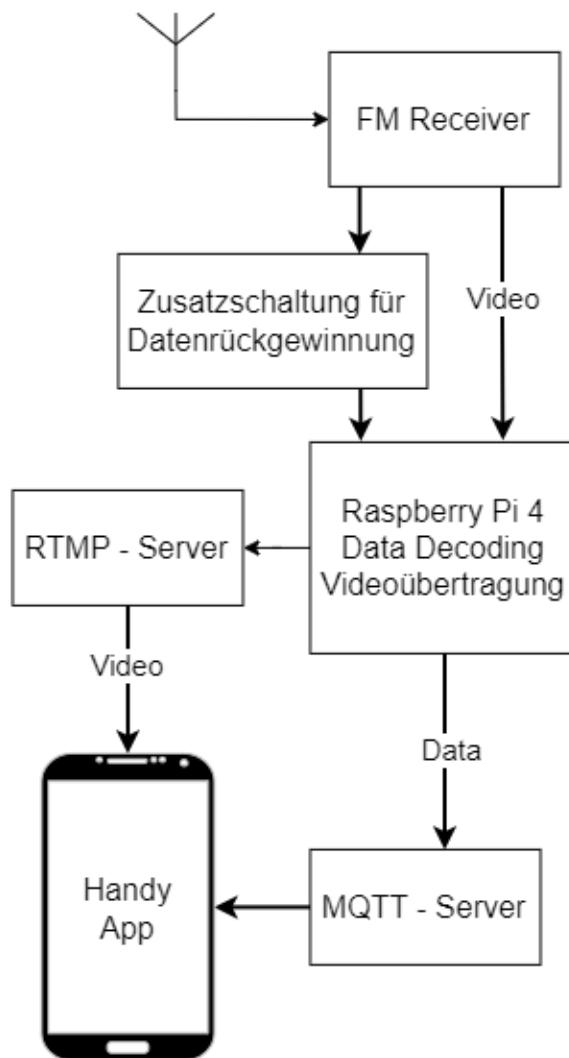


Abbildung 174: Blockschaltbild Empfänger

Die empfangenen Video- und Sensordaten werden vom FM-Receiver an den Raspberry Pi weitergesendet. Dazwischen müssen die Daten noch aufbereitet werden.

Auf dem verwendeten Raspberry Pi 4 (8GB) läuft das Raspberry Pi OS 6.1 64-Bit als Desktop Version. Darauf laufen verschiedene Prozesse, die die Videodaten an den RTMP - Server weiterleiten, sowie die Messdaten auf einem MQTT-Server speichern. Von dort aus können dann alle Daten von einem Zuschauer auf der Handy-App abgerufen und angezeigt werden.

6.4.2 Anschlussdiagramm

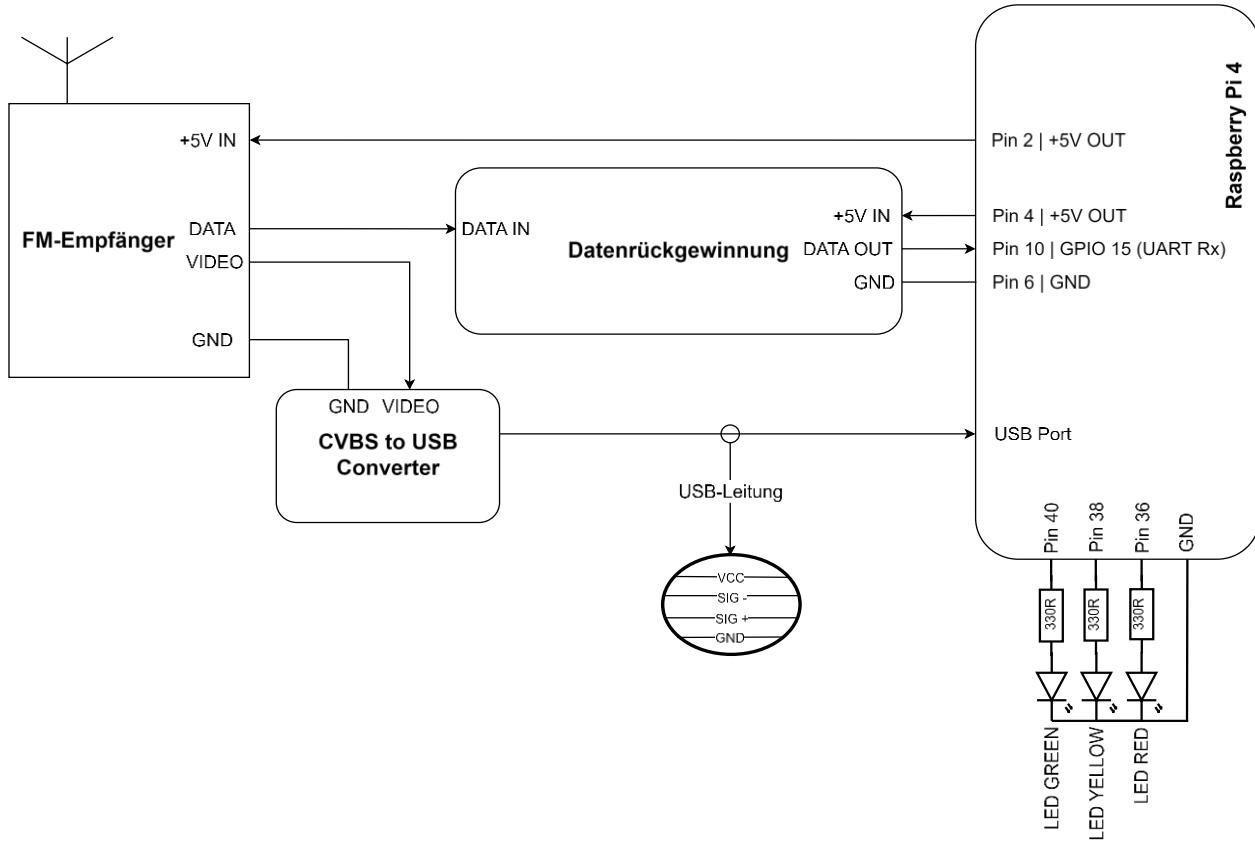


Abbildung 175: Anschlussdiagramm Hardware mit Raspberry Pi

Die empfangenen Video- & Sensordaten müssen, bevor sie in den Raspberry Pi kommen, wo die Daten verarbeitet werden, noch aufbereitet werden.

Die Sensordaten werden in der Datenrückgewinnung so aufbereitet, dass danach wieder ein schönes Datensignal mit klar erkennbaren Flanken herauskommt, dass der Raspberry Pi über UART einlesen kann.

Auch für das Videosignal benötigen wir einen Converter, der das einzelne Signal auf ein symmetrisches USB-Signal umwandelt. Der Empfänger und die Schaltung zur Datenrückgewinnung werden direkt mit 5V vom Raspberry Pi versorgt. Der CVBS-to-USB-Converter wird über den USB-Port auch vom Raspberry Pi versorgt. Vom Empfänger muss keine Masse-Verbindung zur Datenrückgewinnungsschaltung gemacht werden, weil der Empfänger bereits mit der Masse des CVBS-to-USB-Converter verbunden ist und sich die Masse somit über den Raspberry Pi schließt.

6.4.3 Raspberry Pi GPIO

Beim Raspberry Pi gibt es 2 Arten von Pin-Bezeichnungen. Die verschiedenen Ports können entweder mit deren Board-Nummer bezeichnet werden. Das sind einfach alle GPIO-Pins von 1 – 40 durchnummeriert und bezeichnet die physische Pin-Nummer. Die zweite Art der Bezeichnung ist über die BCM (Broadcom SOC Channel) Bezeichnung. Das ist quasi die Benennung, unter der das Betriebssystem die GPIO-Pins kennt. Auf welche Art die Pins bezeichnet werden kann man bei der Initialisierung auswählen.

```
# GPIO config (BCM)
GPIO.setmode(GPIO.BCM)

# GPIO config (BOARD)
GPIO.setmode(GPIO.BOARD)
```

Raspberry Pi Portbelegung:

Physischer Pin	BCM	Funktion
1	-	Power 3.3V
2	-	Power 5V
3	2	SDA1 (I2C Data)
4	-	Power 5V
5	3	SCL1 (I2C Clock)
6	-	Ground
7	4	GPIO4
8	14	UART0_TX (TXD)
9	-	Ground
10	15	UART0_RX (RXD)
11	17	GPIO17
12	18	GPIO18
13	27	GPIO27
14	-	Ground
15	22	GPIO22
16	23	GPIO23
17	-	Power 3.3V
18	24	GPIO24
19	10	SPI_MOSI
20	-	Ground

Physischer Pin	BCM	Funktion
21	9	SPI_MISO
22	25	GPIO25
23	11	SPI_SCLK
24	8	SPI_CE0_N
25	-	Ground
26	7	SPI_CE1_N
27	0	ID_SD (I2C ID EEPROM)
28	1	ID_SC (I2C ID EEPROM)
29	5	GPIO5
30	-	Ground
31	6	GPIO6
32	12	GPIO12
33	13	GPIO13
34	-	Ground
35	19	GPIO19
36	16	GPIO16
37	26	GPIO26
38	20	GPIO20
39	-	Ground
40	21	GPIO21

[RPI1]

Grundsätzlich ist es egal, welche Bezeichnung man verwendet. Wir haben uns somit für die BCM – Bezeichnung entschieden.

6.4.4 Datenrückgewinnung

Durch das bereits angesprochene Problem (siehe: [Kapitel 6.5](#)) wird für die Datenrückgewinnung eine Hardwareschaltung aufgebaut, die erstens das verfälschte UART-Signal wieder rückgewinnen soll und zweitens das gesamte Signal invertiert. Die Invertierung wäre auch softwaremäßig möglich, stellt aber hardwaretechnisch im Prinzip keinen Aufwand dar.

Signal nach Übertragung:

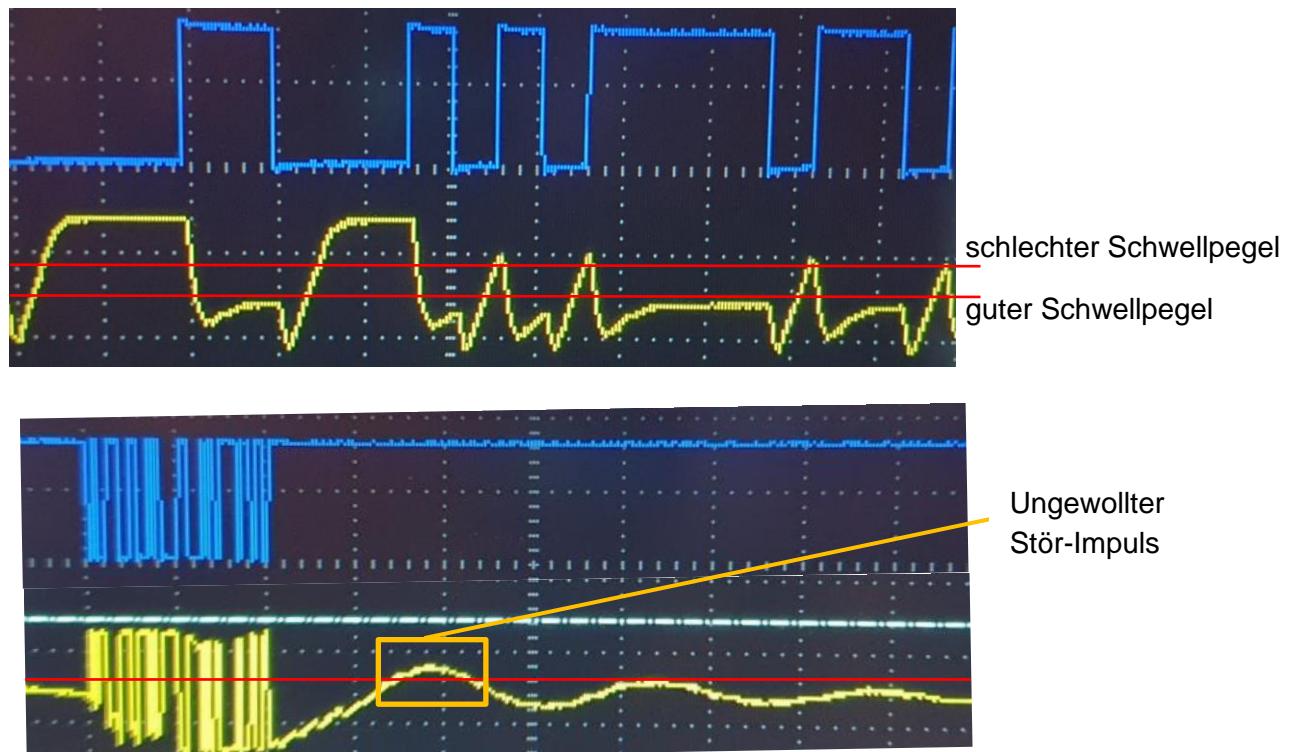


Abbildung 176: Signal nach der Datenübertragung

Dadurch, dass der Vergleichs-Pegel so niedrig sein muss wie möglich, da ansonsten das Timing des Signals nicht mehr stimmt (Abb. 176) und wegen des Schwingverhaltens (siehe: [Kapitel 6.2.3.2](#)), werden Störimpulse miterfasst, die später in der Empfangssoftware übersprungen werden müssen.

Dafür wurde eine invertierende Komparator-Schaltung auf Basis eines OPV aufgebaut. Man hätte dafür auch direkt einen Komparator verwenden können, da ich die Schaltung aber in den Ferien gebaut habe und zu der Zeit nur OPVs zur Verfügung hatte, wurde die Schaltung als OPV-Schaltung aufgebaut und auch so gelassen. Nachgemessen sollte die Schaltschwelle ca. 2,2V betragen.

Widerstandsberechnungen:

Für den Pegel der Schaltschwelle wurde ein einfacher Spannungsteiler verwendet, der durch ein Poti einstellbar ist. Als Versorgungsspannung für die Schaltung wird $U_B = 5V$ verwendet. Um aus $5V \rightarrow \sim 2,2V$ zu machen benötigt man ungefähr einen 1:1 Spannungsteiler. Da ein Poti verwendet wird, ist das Verhältnis nicht ganz so wichtig, denn die $2,2V$ sind ja nur eine Vermutung. Im Endeffekt wird der Pegel dann sowieso mit dem Poti eingestellt.

Weil ich gerade ein $20k\Omega$ Poti zur Hand hatte, wurde der zweite Widerstand wie folgt berechnet:

$$R2 = \frac{UR2 * R1}{UB - UR2} = \frac{2,2V * 15k\Omega}{5V - 2,2V} = 11,7k\Omega = 12k\Omega$$

Für das Poti wurde nicht $20k\Omega$, sondern $15k\Omega$ eingesetzt, weil ja noch ein Spielraum nach oben und unten zum Einstellen bleiben soll.

Der Ausgangsspannungsbereich von OPVs reicht nicht bis an die Versorgungsspannung heran. Bei den meisten OPVs liegt dieser „Offset“ bei $\sim 2V$. Um diesen herauszufinden wurden eine einfache Messung durchgeführt, bei der am Signaleingang 5 V angelegt wurden. Am Ausgang wurde dadurch eine Spannung von 3,9V gemessen.

Um daraus ein 3,3V-Pegel zu machen, wurde der Einfachheit halber wieder ein Spannungsteiler verwendet. Da der Strom aus dem OPV nicht zu stark begrenzt werden soll, wurde R3 mit $3,3k\Omega$ eher klein angenommen.

$$R4 = \frac{UR4 * R3}{UB - UR4} = \frac{3,3V * 3,3k\Omega}{3,9V - 3,3V} = 18,2k\Omega = 22k\Omega$$

Da die Ausgangsspannung eher über 3,3V als unter 3,3V liegen soll wurde der Widerstandswert aufgerundet.

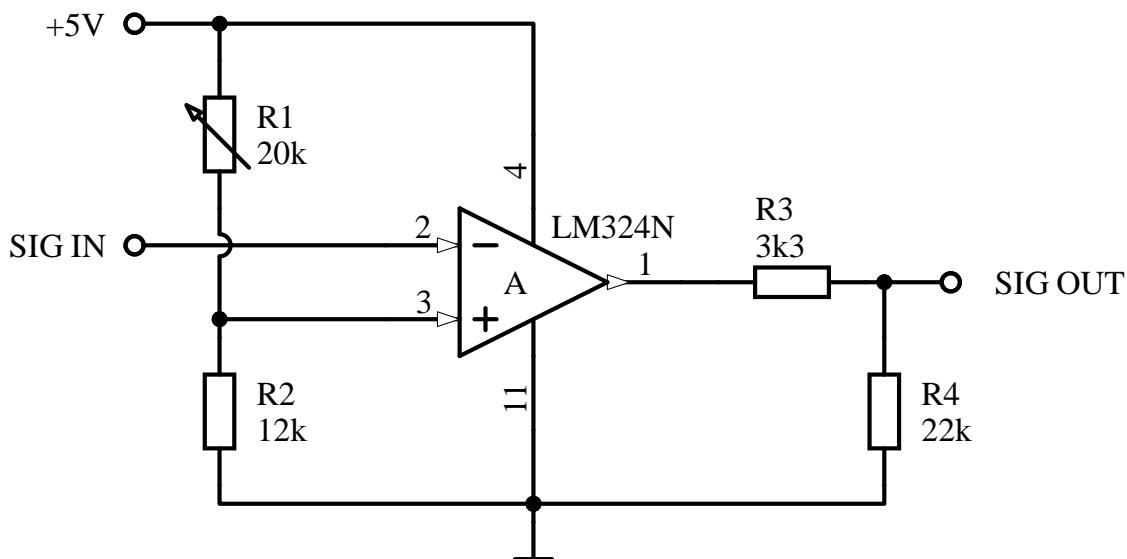


Abbildung 177: Schaltung zur Datenrückgewinnung

6.4.5 CVBS-to-USB-Converter

Das Videosignal, dass durch die Kamera erzeugt wird, ist ein analoges CVBS-Signal, mit dem es möglich ist, Farbe und Helligkeit in einem einzigen Signal zu übermitteln. Jedoch kann die Funktion auf dem Raspberry Pi, die das Video aus dem CVBS-Signal digitalisiert und weiterverarbeitet, das Eingangssignal nicht von den GPIO-Leitungen, sondern nur von den USB-Ports einlesen. Deswegen muss das Videosignal zuerst mit einem Converter, der dazu gekauft wurde zu einem USB-Signal konvertiert werden, damit es dann an einem USB-Port angesteckt und verarbeitet werden kann (*siehe: [Kapitel 8.3](#)*).

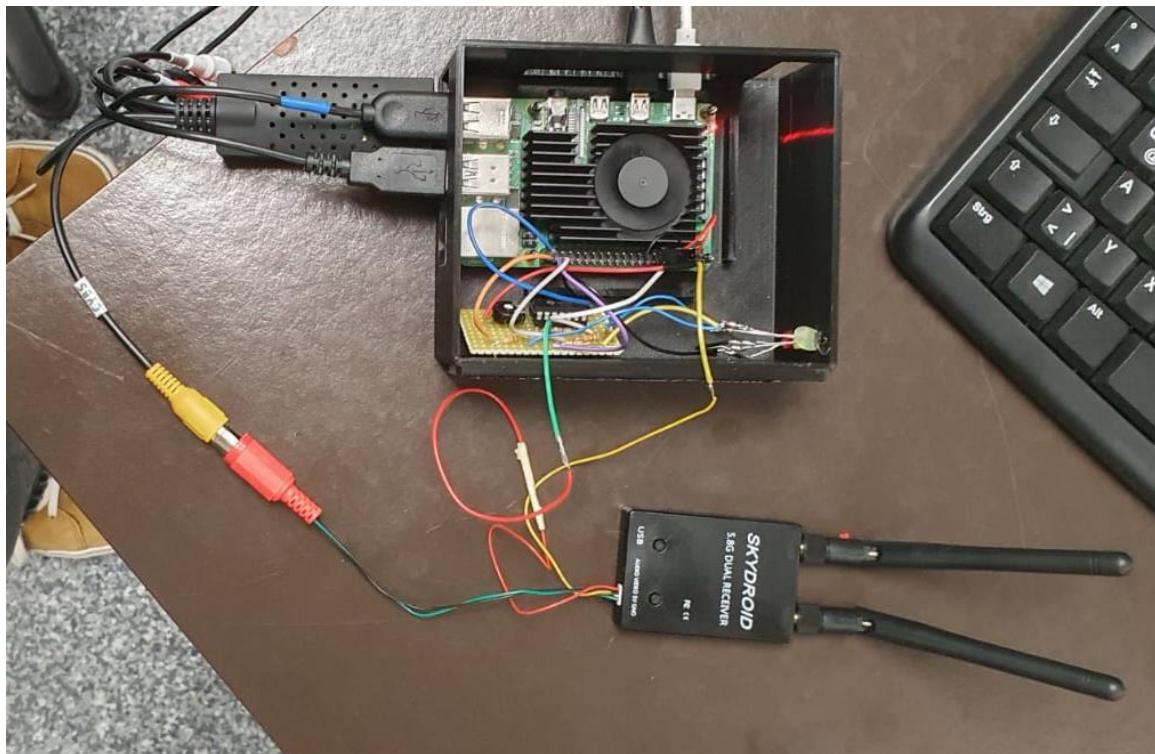


Abbildung 178: CVBS zu USB - Converter

6.4.6 Empfangsprogramm auf Raspberry Pi

Das Empfangsprogramm wurde in der Programmiersprache Python entwickelt und dient der Datendecodierung, indem es die Daten aus der Gleitkommadarstellung zurück in Float-Zahlen wandelt und diese dann an den MQTT-Server sendet.

6.4.6.1 UART auf dem Raspberry Pi

Wie die UART-Hardwarekomponente grundsätzlich funktioniert wurde schon erklärt (*siehe: [Kapitel 6.2.5.3](#)*). Für die Implementierung der seriellen Datenübertragung über die V24-Schnittstelle auf dem Raspberry PI wurde die Library: pyserial verwendet, die es ermöglicht serielle Daten-Streams einzulesen. Es gibt jedoch einen großen Unterschied, wenn man serielle Datenübertragung auf dem Raspberry Pi mit der eines Microcontrollers vergleicht. Wenn man auf dem Cortex Microcontroller ein Signal über die UART-Komponente einliest, dann liest man immer das Signale ein, was in diesem Moment anliegt. Somit kann man zum Beispiel bei einem konstanten Daten-Stream nur alle paar Sekunden Daten einlesen und bekommt somit in regelmäßigen Zeitabständen den aktuellen Wert, der an der Leitung anliegt.

Auf dem Raspberry Pi mit der Kombination der pyserial-Library funktioniert das Ganze ein bisschen anders. Wenn man hier versucht in gewissen Zeitabständen Daten einzulesen, liest man standardmäßig nicht die aktuellen Werte, die im Moment anliegen, sondern zuerst die ein, die übersprungen wurden. Dieses Problem hat mich anfangs bei der Programmierung des Datenempfangs sehr aus der Bahn gebracht. Denn erst nach längerem Testen bin ich darauf gekommen, dass das Betriebssystem des Raspberry Pi alle Daten, die am GPIO-Pin für UART ankommen buffert. Wenn man also versucht, gezielt Störimpulse zu überspringen, so wie es in unserem Fall notwendig ist (*siehe: Kapitel 6.4.4*), ist dieses automatische Speicherverhalten der letzten Daten sehr problematisch. Denn somit würde man, nachdem die Störimpulse übersprungen wurden und man eigentlich das nächste Paket einlesen will, noch die Störimpulse einlesen, weil sie automatisch gespeichert wurden.

Nach längerem Testen und Recherchieren habe ich dann einen Befehl gefunden, mit dem man den Inhalt des Buffers löschen kann.

[FLU1]

```
ser.flushInput()
```

6.4.6.2 MQTT-Server

Auf dem MQTT-Server werden unsere Messdaten gesammelt. Von dort können sie dann auch von der App abgerufen werden. Der MQTT-Server wird auf dem Raspberry Pi aufgesetzt. Für die Installation wurde ein Guide aus dem Internet benutzt.

[MQT1]

Installation:

1. Im Terminal MQTT-Broker herunterladen:

```
sudo apt-get install mosquitto mosquitto-clients -y
```

2. Im Config-File (unter: /etc/mosquitto/mosquitto.conf) können Einstellungen, zum Beispiel, welcher Port verwendet wird durchgeführt werden. In unserem Fall wurde alles auf den Standardwerten gelassen.
3. Um den Server nun zu starten und Autostart für den nächsten Boot zu aktivieren, müssen folgende Befehle im Terminal eingegeben werden:

```
sudo systemctl start mosquitto  
sudo systemctl enable mosquitto
```

6.4.6.3 Programm

Flussdiagramm

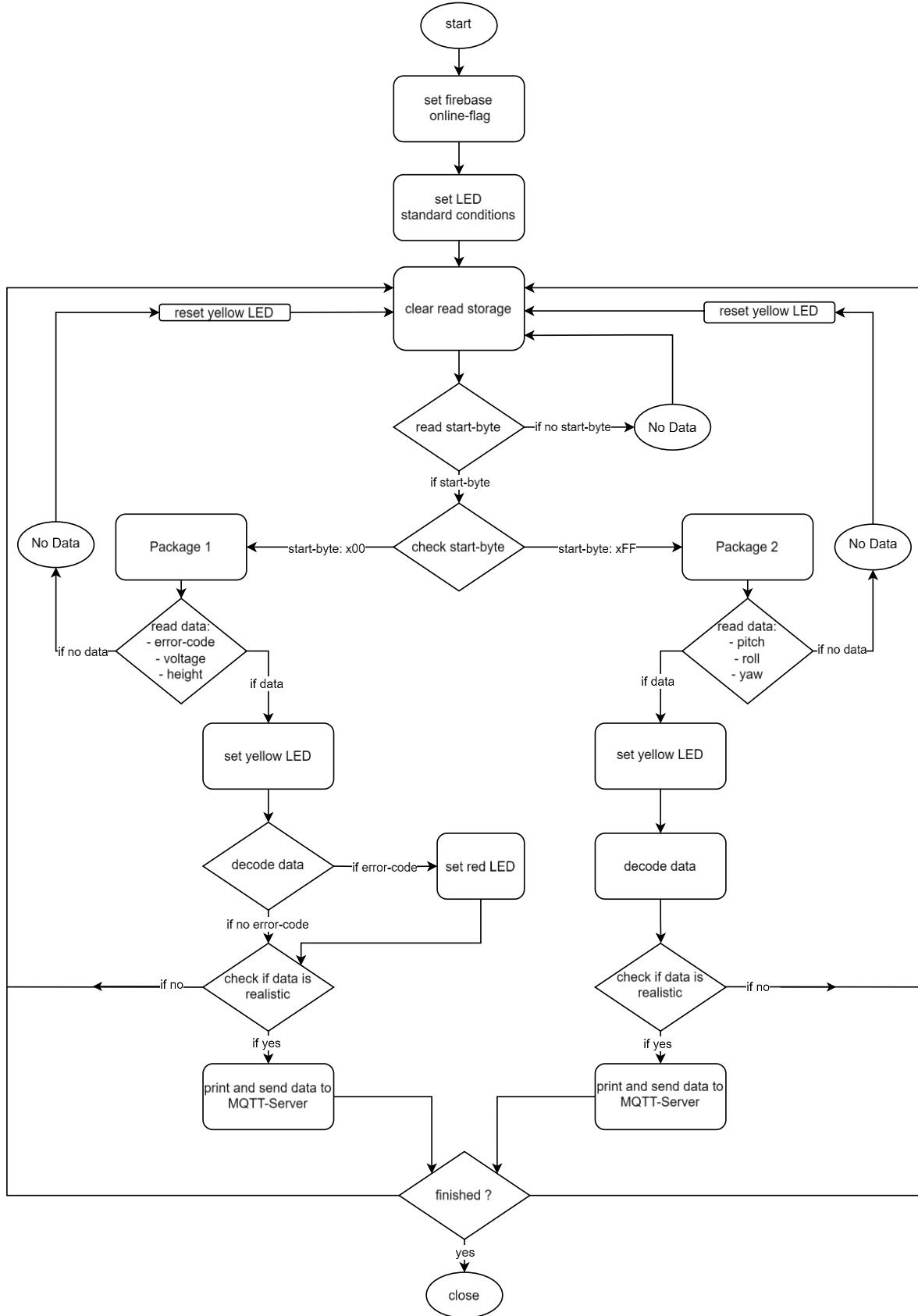


Abbildung 179: Flussdiagramm Empfangen der Daten

Code

Datei: receiver_new.py – GPIO inits
--

```

...
import serial
import RPi.GPIO as GPIO

...
# global definitions
RX_PIN = "/dev/ttyS0" # GPIO15 uart rx port
baud = 4800

# GPIO-serial config
ser = serial.Serial(RX_PIN, baud, timeout=0.5)

# LED config
GPIO.setmode(GPIO.BCM)
LED_Green = 21
LED_Yellow = 20
LED_Red = 16
GPIO.setup(LED_Green, GPIO.OUT)
GPIO.setup(LED_Yellow, GPIO.OUT)
GPIO.setup(LED_Red, GPIO.OUT)

...

```

Um den seriellen PIN zum Datenempfangen zu aktivieren, muss man im Terminal am auf dem Raspberry Pi folgendes eingeben: **sudo raspi-config**

Darauf öffnet sich ein Dialogfenster, in dem man den Unterpunkt **Interfacing-Options** auswählt und auf **Select** drückt. Im nächsten Fenster wählt man den Unterpunkt **Serial** und wählt diesen wieder mit **Select** aus. Im nächsten Fenster wird man gefragt ob auf die Login Shell über die Serielle Schnittstelle zugegriffen werden kann. Dabei wählt man **No**. Im letzten Fenster wählt man mit **Yes** aus, ob man die serielle Schnittstelle aktivieren möchte. Zuletzt wählt man noch **Ok** und beendet die Einstellungen mit **Finish**.

Im Code kann der UART Rx Port (BCM: GPIO 15) dann mit „/dev/ttyS0“ angesprochen werden. Bei der Initialisierung des UART RX-Pins kann auch noch ein Timeout angegeben werden, welches die Zeit definiert, wie lange beim Einlesen auf Daten gewartet werden soll. Wird diese Zeit überschritten, dann bricht der Einlese-Vorgang ab und das Programm läuft weiter. Zuletzt wurden noch die GPIO-Pins zum Ansteuern der LEDs initialisiert.

[COF1]

Datei: receiver_new.py – firebase init

```

...
import firebase_admin
...
```

```

from firebase_admin import db

# connect to database
cred_obj =
firebase_admin.credentials.Certificate('/home/fpv/Documents/receiver/fpv.json')
default_app = firebase_admin.initialize_app(cred_obj, {
    'databaseURL': "https://fpv-drohne-default-rtbd.firebaseio-europe-
west1.firebaseio.database.app/"
})

...

```

Um eine Connection mit dem Firebase-Server aufzubauen, benötigt man ein Zertifikat. Dies kann man einfach unter seinem Firebase - Account erstellen und herunterladen. Der Speicherort wird dann in der Initialisierung angegeben. Außerdem wird die Adresse des Firebase-Servers angegeben.

Datei: receiver_new.py – MQTT init

```

...
import paho.mqtt.client as mqtt
...

def mqtt_init() -> mqtt.Client:
    #-----MQTT init-----

    # MQTT broker configuration
    broker_address = "192.168.0.105" # IP of the Raspberry Pi
    port = 1883 # Default MQTT port

    # Create an MQTT client instance
    client = mqtt.Client()

    # Connect to the MQTT broker
    client.connect(broker_address, port=port)

    # Start the loop
    client.loop_start()

    return client

...

```

Um sich mit dem, auf dem Raspberry Pi laufenden, MQTT-Server zu verbinden, muss man beim Client Connect die IP-Adresse des Raspberry Pi und den MQTT-Default Port angeben.

Anmerkung: Der folgende Code geht über mehrere Seiten.

Datei: receiver_new.py – Hauptprogramm

```

...
import struct
import time
import sys

...
# ---- Main Code -----
try:
    if __name__ == '__main__':
        client = mqtt_init()

        # send data to firebase
        ref = db.reference("is_online")
        ref.set(True)

        # set led standard condition
        GPIO.output(LED_Green, GPIO.HIGH)
        GPIO.output(LED_Yellow, GPIO.LOW)
        GPIO.output(LED_Red, GPIO.LOW)

        # set timestamp for data syncronisation
        timestamp = 0.0

    while 1:
        # clear read storage
        ser.flushInput()

        # read recognition byte, error if no data
        recognition_byte = ser.read(1)
        if not recognition_byte:
            print("No Data!")
            continue

        # counter --> how many 1's in recognition byte
        int_byte = int.from_bytes(recognition_byte, "big")
        counter = bin(int_byte).count('1')

        # package 1 (sync_byte = 0x00)
        if counter <= 4:

            # read following data, error if no data
            data_voltage = ser.read(4)
            data_groundeddistance = ser.read(4)
            data_error = ser.read(1)

```

```

        if not data_voltage or not data_groundeddistance or not
data_error:
    print("No Data!")
    GPIO.output(LED_Yellow, GPIO.LOW)
    continue

    # Yellow-LED --> data received
    GPIO.output(LED_Yellow, GPIO.HIGH)

    # convert floating point numbers back to float number
    float_voltage = struct.unpack('!f',data_voltage)[0]
    float_groundeddistance = struct.unpack('!f',
data_groundeddistance)[0]
    # if errorcode, set Red-LED
    int_errorbyte = int.from_bytes(data_error, "big")
    counter = bin(int_errorbyte).count('1')
    if counter >= 1:
        GPIO.output(LED_Red, GPIO.HIGH)
    else:
        GPIO.output(LED_Red, GPIO.LOW)

    # Ausgabe am Terminal und Senden zu MQTT-Server
    print("Voltage: ")
    if float_voltage < 30 and float_voltage > 1:
        print(float_voltage)
        client.publish("data/voltage", "B " + str(float_voltage)
+ " " + str(timestamp))
    else:
        print("Transmission - Error!")

    print("Distance: ")
    if float_groundeddistance < 0:
        print(float_groundeddistance)
        client.publish("data/height", "H " +
str(float_groundeddistance) + " " + str(timestamp))
    else:
        print("Transmission - Error!")

    print("errorcode: ")
    print(data_error)

    # package 2 (sync_byte = 0xFF)
    elif counter > 4:

        # read following data, error if no data
        data_pitch = ser.read(4)
        data_roll = ser.read(4)
        data_yaw = ser.read(4)

        if not data_pitch or not data_roll or not data_yaw:

```

```

        print("No Data!")
        GPIO.output(LED_Yellow, GPIO.LOW)
        continue

    # Yellow-LED --> data received
    GPIO.output(LED_Yellow, GPIO.HIGH)

    # convert floating point numbers back to float number
    float_pitch = struct.unpack('!f', data_pitch)[0]
    float_roll = struct.unpack('!f', data_roll)[0]
    float_yaw = struct.unpack('!f', data_yaw)[0]

    # Ausgabe am Terminal und Senden zu MQTT-Server
    print("Pitch: ")
    if float_pitch <= 180 and float_pitch >= -180:
        print(float_pitch)
        client.publish("data/pitch", "P " + str(float_pitch) +
" " + str(timestamp))

    else:
        print("Transmission - Error!")

    print("Roll: ")
    if float_roll <= 180 and float_roll >= -180:
        print(float_roll)
        client.publish("data/roll", "R " + str(float_voltage) +
" " + str(timestamp))

    else:
        print("Transmission - Error!")

    print("Yaw: ")
    if float_yaw <= 180 and float_yaw >= -180:
        print(float_yaw)
        client.publish("data/yaw", "Y " + str(float_voltage) +
" " + str(timestamp))

    else:
        print("Transmission - Error!")

    time.sleep(.14) # wait for 140ms
    timestamp += 0.2 # increase time stamp
...

```

Dieser Code-Abschnitt empfängt die Daten, wandelt sie um und schickt sie an den MQTT-Server. Zuerst wird die „is_online“ – Flag in Firebase gesetzt, damit die App abrufen kann, ob die Datenübertragung läuft oder nicht. Die Überprüfung, welches Start-Byte gesendet wurde, wurde so umgesetzt, dass überprüft wird, ob mehr als 4 oder weniger als 4 Bits im Start-Bit gesetzt sind.

Somit schließt man falsche Entscheidungen aus, bei denen 1 oder 2 Bits umgefallen sind. Nach dieser Entscheidung werden die restlichen Daten eingelesen und decodiert. Auch wird hier die gelbe LED als Anzeige, dass Daten übertragen wurden, gesetzt. Die grüne LED ist standardmäßig gesetzt, um zu zeigen, dass das Programm läuft und die rote LED zeigt Fehler an. Zuletzt wird überprüft, ob die Daten realistisch sein können. Sind sie das, dann werden die Daten am Terminal ausgegeben und an den MQTT-Server gesendet. Dafür wurde für jede Zahl ein Identifier dazu gesetzt (z.B: P). Außerdem wird zu jedem Wert ein Zeitstempel dazugesetzt, damit man die Sensordaten, auf einer Zeitachse auftragen kann.

Beispiel:

```
client.publish("data/pitch", "P " + str(float_pitch) + " " + str(timestamp))
```

Datei: receiver_new.py – close program

```
...
# send good-bye-message
except(KeyboardInterrupt):
    print(" Bye!")

# clean-ups
finally:
    ser.close
    GPIO.cleanup()
    client.disconnect()
    client.loop_stop()
    ref = db.reference("is_online")
    ref.set(False)

...
```

Zuletzt wird noch die „is_online“ – Flag zurückgesetzt, damit die App keine Daten mehr erwartet und es werden alle offenen Schnittstellen und Verbindungen geschlossen.

6.5 Testen der Datenübertragungskette

Zuletzt musste die gesamte Übertragung, vom Microcontroller bis zum Raspberry Pi getestet werden. Dafür wurde die gesamte Kette wie folgt aufgebaut. Die Kamera und die Datenleitung des Flight-Controllers wurde an der VTx angeschlossen. Diese wurde an einer 12V Spannungsquelle angeschlossen. Der Receiver und die Datenrückgewinnungsschaltung wurden wie in: [Kapitel 6.4.2](#) beschrieben angeschlossen. Danach wurde überprüft, ob die gleichen Channel auf Sender, sowie Empfänger ausgewählt sind (siehe: [Kapitel 6.2.3.1](#) und [Kapitel 6.3.3](#)). Zuletzt wurde der Flight Controller eingeschaltet und das Programm auf dem Raspberry Pi zum Empfangen der Daten gestartet.

```

PROBLEMS   OUTPUT   DEBUG CONSOLE   TERM
Roll:
-0.0206600334495306
Yaw:
1.2639144659042358
Voltage:
12.180000305175781
Distance:
1.2245924472808838
errorcode:
b'\x00'
Pitch:
-0.6456024646759033
Roll:
-0.018446099013090134
Yaw:
1.265078067779541
Voltage:
12.180000305175781
Distance:
1.1591655015945435
errorcode:
b'\x00'
Pitch:
-0.6436699032783508
Roll:
-0.015581033192574978
Yaw:
1.2682578563690186
Pitch:
5.993298881277134e-39
Roll:
Transmission - Error!
Yaw:
-1.5407907916128293e-33

```

Abbildung 180: Empfangen von Messdaten

Wie man erkennen kann, sind die Daten richtig angekommen. Der letzte Roll-Wert ist irgendwo zwischen VTx und Raspberry Pi gestört worden und lag somit außerhalb der Toleranzgrenze, die überprüft, ob die Werte realistisch sein können. Somit wurde statt dem Zahlenwert: „Transmission – Error!“ ausgegeben.

Dadurch, dass pro Zahl 4 Bytes für die Übertragung benötigt werden (Gleitkommadarstellung) und zwischen den einzelnen Datenpaketen ein Sendeabstand eingebaut wurde (siehe: [Kapitel 6.2.6](#)) ist die Datenrate nicht sonderlich schnell.

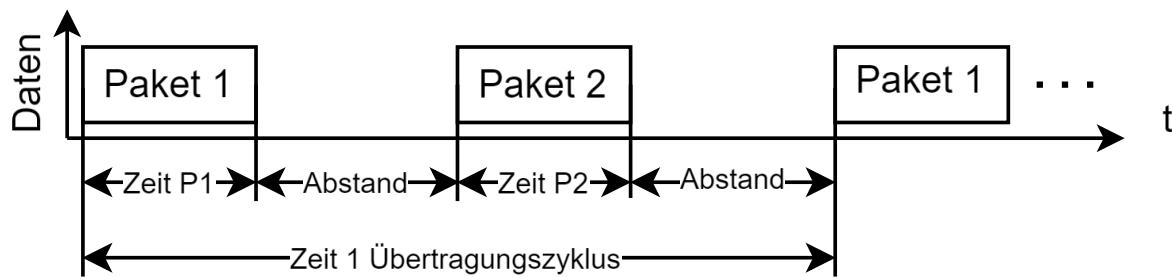


Abbildung 181: Sendeabfolge Zeitmessung

Berechnungsgrundlagen:

Packet 1: 10 Byte	Packet 2: 13 Byte
Sendeabstand: 150ms	Baudrate: 4800

$$\text{Datenrate} = \frac{\text{Übertragene Bits / Zyklus}}{\text{Zeit pro Zyklus}} = \frac{8 * (\text{Bytes Paket 1} + \text{Bytes Paket 2})}{2 * \text{Sendeabstand} + \text{Zeit / Byte} * (\text{Bytes Paket 1} + \text{Bytes Paket 2})}$$

$$= \frac{8 * (10 + 13)}{2 * 150\text{ms} + \frac{10}{4800} * (10 + 13)} = \frac{184}{0,348 \text{ s}} = 528,86 \text{ bit/s} \approx 0,5 \text{ kbit/s}$$

Oder anders gesagt wird jeder Sensorwert ca. 3-mal in der Sekunde übertragen. Diese Datenrate reicht natürlich nicht aus, um das Flugverhalten der Drohne auf das genaueste nachzuvollziehen und zum Beispiel Schlüsse aus den IMU-Daten zu ziehen. Für die allgemeine Visualisierung reicht die Übertragungsgeschwindigkeit jedoch aus.

Im Nachhinein wäre es eine bessere Entscheidung gewesen, entweder ein FSK-Moduliertes Signal zu verwenden oder gleich ein zweites Sendemodul zu kaufen. Da gibt es schon für einen sehr geringen Preis Module wie zum Beispiel ein HC-12 Funkmodul, das mit einer Übertragungsgeschwindigkeit von 5 kbit/s Datenraten um einen Faktor 10 höher ermöglicht als mit meiner Variante.

Das direkte Senden eines UART-Signals zieht nämlich viele Nachteil mit sich. Denn neben einer sehr Störanfälligen Datenübertragung sieht das Ausgangsspektrum des Senders deutlich anders aus als vom Hersteller vorgesehen. Eigentlich sollten nämlich nur Audiosignale über diesen Kanal übertragen werden. Durch die unkonventionelle Nutzung des Senders können ungewollte Oberwellen entstehen, die zu stark von der vorgegebenen Mittenfrequenz abweichen und andere Teilnehmer im selben Frequenzband stören könnten.

7 Visualisierungs-App

7.1 Applikation

7.1.1 Einführung – Dart / Flutter - Framework

7.1.1.1 Allgemeines

Flutter ist ein Open-Source-Framework von dem Unternehmen Google und wird seit Erstveröffentlichung im Jahr 2017 stetig mit Updates versorgt. Mit diesem Framework ist es möglich Native-Apps für Linux und Windows, Web-Apps aber auch Smartphone-Applikationen für Android und iOS zu machen.

Unter dem Framework Flutter versteckt sich jedoch die Programmiersprache namens „Dart“, womit man die vorgefertigten UI-Elemente / Widgets einfach zusammenfügen und miteinander logisch verbinden.



Abbildung 182: Flutter Framework Logo [FFLO]

Vorteile:

- Mehrere Plattformen mit nur einer Codebasis unterstützt
- Einfaches Erstellen von Layouts mit dem Prinzip der Widgets
- Zustandsloses und zustandsbehaftete UI-Elemente, was bei korrekter Benutzung die Effizienz der Apps immens erhöht
- Riesige Auswahl an Plugins, womit viele Aufgaben an bewährte und von der Community getesteten Packages übernommen werden können

7.1.1.2 Pub Dev / Package Installer

Ein weiterer wichtiger Teil des Flutter Frameworks ist der Package Manager namens *pub.dev*. Mit jenem lassen sich von der Community oder von Firmen erstellte Packages / Erweiterungen in sein eigenes Projekt einfügen. Somit können schwere spezifische UI-Elemente von bereits getesteten und meist relativ effizienten Plugins übernommen werden.

Ein Package kann über die offizielle Pub Dev Seite angeschaut werden. Die Plugins auf der Website können je nach Kriterien wie Titel, unterstützte Plattformen oder Lizenz gefiltert werden. Viele der Plugins besitzen zusätzlich meist ein kleines Beispiel, Installationsinstruktionen, einen Change Log der verschiedenen Versionen und eine Readme-Sektion, in der kurz das Package beschrieben wird.

7.1.1.3 Dart

Dart ist eine ursprünglich von Google im Jahre 2011 entwickelte objektorientierte Programmiersprache und ist die Hauptsprache des Flutter-Frameworks. Sie wird verwendet, um die Logik und Funktionalität innerhalb der Anwendungen zu implementieren. Neben den üblichen Kontrollstrukturen wie If-else, for oder while, gibt es noch das Konzept der „Nullable Types“ mit dem „?“-Zeichen und dem Import von jeglichem Package mittels des „import“-

Keywords. Unter „Nullable Types“ versteht man Variablen, die optional auch „null“ als Wert, unabhängig derer Datentypen annehmen können.

Syntax Beispiel

Um die Syntax von Dart kurz und prägnant widerzuspiegeln, habe ich ein kurzes Beispielprogramm geschrieben, indem die wichtigsten Komponenten einer modernen Programmiersprache vorkommen:

```
import 'dart:math';

void main() {
    // Variablendeclaration und Initialisierung
    int zahl = 42;

    // "dynamic" kann jeder Typ zugewiesen werden
    List<dynamic> namensListe = ['Thomas', zahl, false];

    // Ausgabe von Text und Variablenwerten
    print("sin(1) = ${sin(1)}");

    // Bedingte Anweisungen (if-else)
    if (namensListe.length == 3) {
        print("Die Liste hat 3 Werte");
    } else {
        print("Die Liste hat NICHT 3 Werte");
    }

    // Schleifen (for-Schleife)
    for (int i = 0; i < 3; i++) {
        print('Schleifendurchlauf Nr. $i');
    }

    halloName("Josef");
}

// Funktionen
void halloName(String name) {
    print("Hallo $name");
}
```

Programmoutput:

```
sin(1) = 0.8414709848078965
Die Liste hat 3 Werte
Schleifendurchlauf Nr. 0
Schleifendurchlauf Nr. 1
Schleifendurchlauf Nr. 2
Hallo Josef
```

7.1.1.4 State Management

Unter dem Begriff „State Management“, versteht man die Änderung von Daten innerhalb einer Applikation und wie diese Änderungen auf der UI wiedergespiegelt werden.

Stateful vs. Stateless Widgets

In Flutter ohne weitere Plugins gibt sind Stateless-/Stateful-Widgets die einzige Methode, um State-Management umzusetzen. Bei der Erstellung eines neuen Widgets kann man daher ein Stateless-Widget erstellen, wobei keine Echtzeitänderungen umgesetzt werden können, und ein Stateful-Widget. Stateful-Widgets unterteilen sich in zwei Blöcke. Dem Stateful-Widget selbst worin nur die Daten sind die „*immutable*“ bzw. unveränderbar sind, sowie dem „State“-Objekt, welches die UI-Elemente dieses Widgets beschreibt und auch ein Update jener Elemente mit der „`setState`“-Funktion zulässt. Innerhalb dieser Funktion lassen sich direkt auch Variablen setzen wessen Änderungen dann direkt beim „Rebuild“/Update des Widgets wiedergespiegelt werden, sofern jene Variable auch benutzt wird.

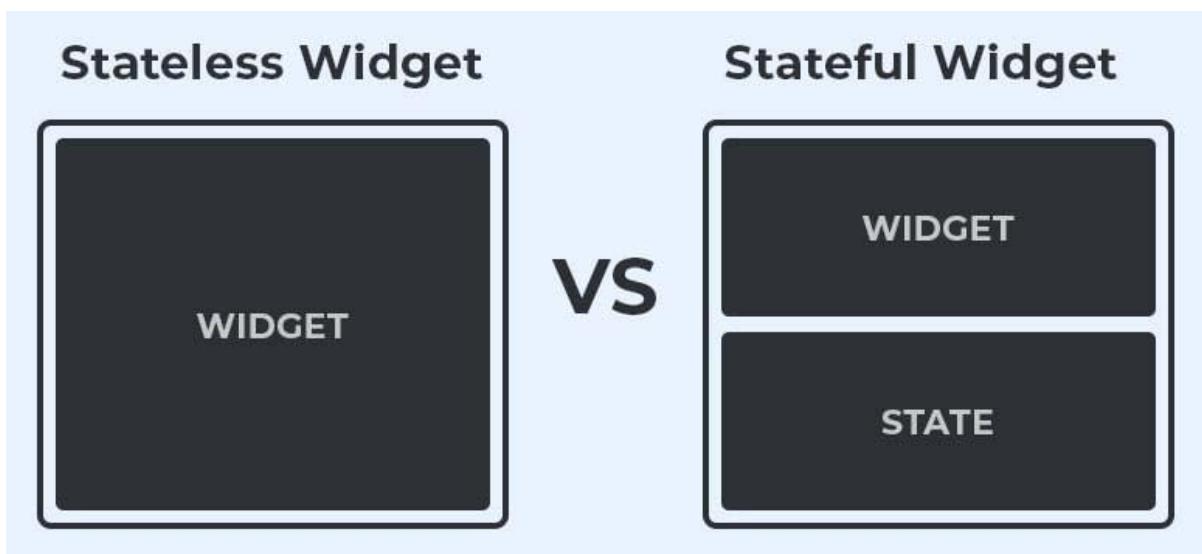


Abbildung 183: Stateful vs. Stateless Widget [SLSF]

Provider

Provider ist eines der bekanntesten State-Management Packages in Flutter und erlaubt es sogenannte „Provider“-Klassen zu erstellen und jene zu Start des Programmes zu definieren und Eigenschaften (Variablen in einer Klasse) überall im Fluss der Applikation zu benutzen und automatisch bei Änderung jener innerhalb der benutzten Widgets umzusetzen.

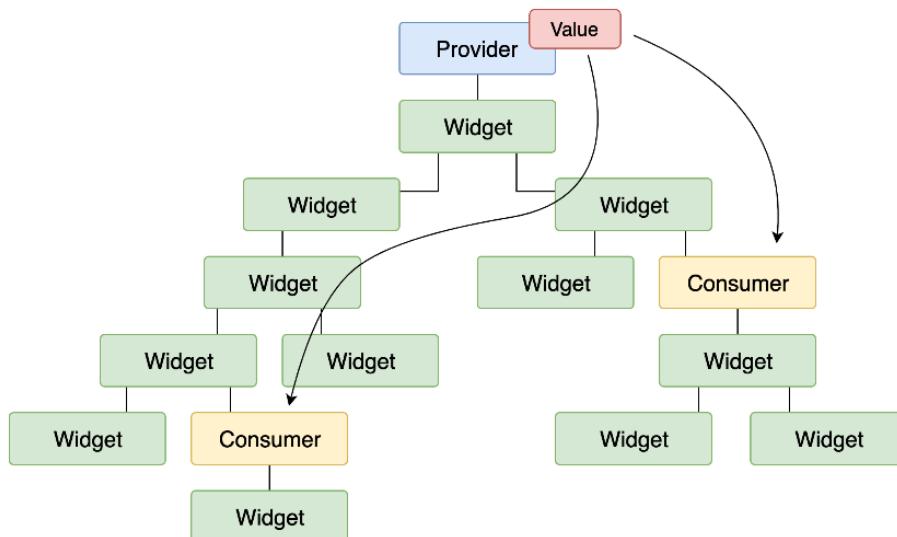


Abbildung 184: Provider State Management Aufbau [PROV]

Im Aufbau zeigt sich die Funktion eines Providers. Eine Providerklasse wird zum Start der Applikation bzw. an der Wurzel des „Widget Trees“ definiert. Diese Klasse kann dann in den Kindern dieser Klasse bzw. überall in der Applikation über ein „Consumer“-Widget oder alternativ auch direkt über den BuildContext aufgerufen werden. Wird nicht explizit angegeben, dass Änderungen ignoriert werden sollen, wird jede Änderung eines bestimmten Wertes bzw. Eigenschaft in der Providerklasse überall automatisch aktualisiert. Um die UI-Elemente von den Änderungen in der Klasse zu notifizieren muss nach Änderung einer Eigenschaft lediglich die „notifyListener“-Funktion in der Providerklasse aufgerufen werden.

Es gibt noch viele weitere dieser Packages, aber es ist das Einzige, dass neben den „Stateful Widgets“ innerhalb der Visualisierungsapp benutzt wird.

7.1.1.5 StreamBuilder

Der StreamBuilder ist eine Art von Wrapper für Widgets. Hierbei kann eine Funktion mit einem kontinuierlichen Datenstream als Rückgabewert definiert werden („stream“). Die Daten von dem Stream können schließlich innerhalb der Widgets im StreamBuilder verwendet werden, um somit z.B. Daten in Echtzeit anzuzeigen oder die UI auf die Änderungen einer Variable auf einer Datenbank reagieren zu lassen. Es ist ein essenzieller Aspekt, um insbesondere Sensordaten in Echtzeit zu Empfangen und innerhalb der UI widerzuspiegeln.

7.1.2 Allgemeines zur App

Die Visualisierungsapp für die FPV-Drohnen-Diplomarbeit wurde in Flutter realisiert, mit dem Ziel sowohl die Live-Videodaten als auch die Flugdaten (Höhe, Lage, Temperatur,...) in Echtzeit zu empfangen und möglichst benutzerfreundlich zu präsentieren. Sie wurde primär für Android als Endgerät entwickelt. Eine theoretische Konvertierung für z.B. iOS oder auch Windows, sollte aufgrund der plattformübergreifenden Natur von Flutter kein Problem darstellen.

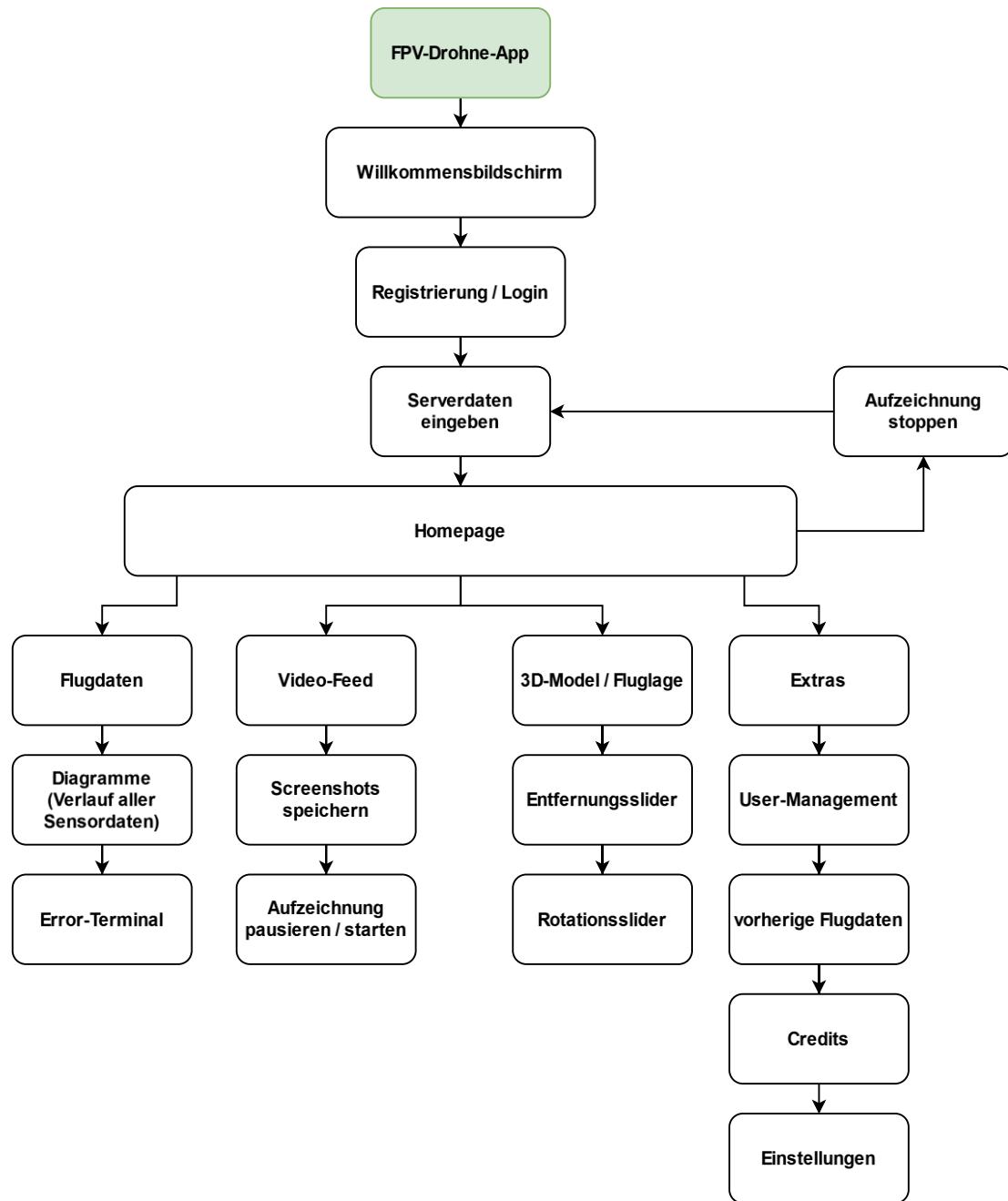


Abbildung 185: Allgemeiner Aufbau der Visualisierungsapp

Die obige Abbildung zeigt den generellen Ablauf und die Hauptfunktionen innerhalb der Visualisierungsapp. Vom Start der Applikation mit dem Loginbereich, bis hin zur Homepage mit den wichtigsten Optionen und Bildschirmen.

7.1.2.1 Usersystem

Neben der Hauptaufgabe der Echtzeitvisualisierung der Daten, war es ein Ziel die aufgezeichneten Daten für die Zukunft zu speichern und wieder aufrufbar zu machen. Hierfür bot sich ein Usersystem mit Profilen und eigenen Logindaten an. Ein Userprofil soll auf einer Registrierungsseite mit einer E-Mail und einem Password angelegt werden und dann direkt, als Login verwendet, werden können. Abgesehen von dem Login via E-Mail + Passwort sollen auch simplere alternative Anmeldemöglichkeiten wie z.B. mittels eines bereits existierenden Google-Accounts angeboten werden.

Als Backend / Datenbank für die Realisierung der Autorisierung und Speicherung aller Userdaten innerhalb der, werden einige Bestandteile der Cloud-Datenbank von Google namens Firebase benutzt.

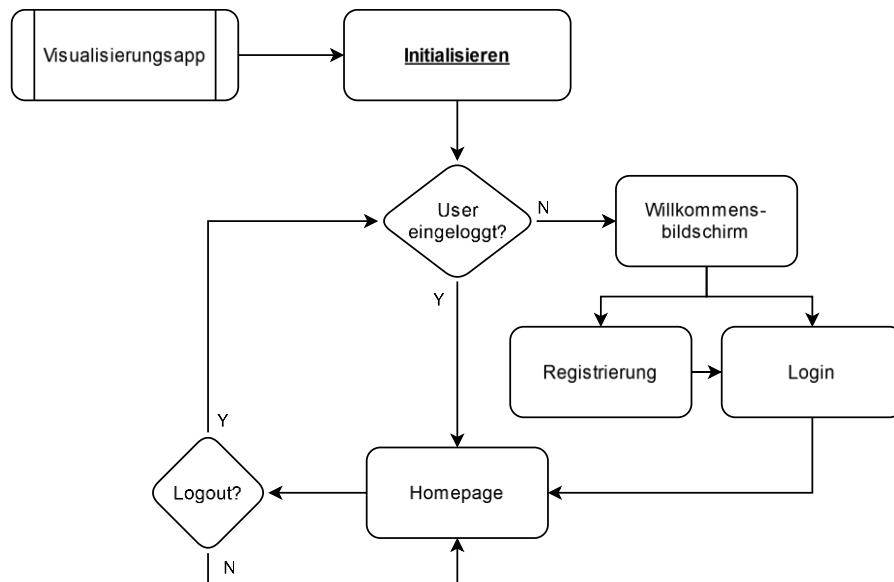


Abbildung 186: Flussdiagramm, Userstatus (Autologin, Logout)

In der obigen Abbildung zeigt sich der Ablauf des Logins zum Start der Applikation. Der Ablauf inkludiert den potenziellen automatischen Login, falls bereits ein eingeloggter User gespeichert ist, aber auch den Ablauf innerhalb der verschiedenen Anmeldebildschirme (Willkommensbildschirm, Login, Registrierung) bis hin zu einer erfolgreichen Anmeldung und der Weiterleitung zur Homepage.

7.1.2.2 Datenvisualisierung

Die Visualisierung der Daten findet mittels des MQTT-Protokolls statt, welches ein Standard für die Übermittlung von Sensordaten in der modernen Welt ist. Die vom MQTT-Broker übertragenen Daten sollen von einem selbstgeschriebenen MQTT-Client in Flutter empfangen und in nutzbare Daten umgewandelt werden. Die Visualisierung soll einerseits anhand von einem Diagramm, aber auch einfach mit einem einfachen Textfeld verfahren, um sowohl den aktuellen Wert direkt ablesen zu können und die zeitliche Entwicklung zu zeigen.

Syncfusion-Flutter

Syncfusion ist im Allgemeinen einen Anbieter von vordefinierten UI-Elementen für jede Form von Applikationen. Die UI-Elemente die Syncfusion anbietet reichen von vielen File-Viewern und Editoren (PDF, Microsoft Excel,....), bis hin zu jeder erdenklichen Art von Diagrammen.

Neben dieser riesigen Auswahl an UI-Elementen bietet Syncfusion seine Produkte für eine Vielzahl an Web- und App-Frameworks, inklusive dem Flutter Framework zur Verfügung. Damit die App den modernen Standards entspricht und um die Entwicklungszeit zu verkürzen, entschied ich mich dazu deren Produkte in der Visualisierungsapp zur Darstellung von Daten zu integrieren.



Abbildung 187: Syncfusion Logo

7.1.2.3 3D-Model-Viewer

Die Lage der Drohne soll anhand eines 3D-Modells dargestellt werden, indem die Rotationsdaten (Pitch, Roll und Yaw) der Drohne in Echtzeit über MQTT empfangen werden und als Rotationsdaten auf den jeweiligen Achsen im 3D-Raum gesetzt werden. Zusätzlich wird unter dem 3D-Modell einige Optionen mittels Buttons gegeben, mit dem die Entfernung der Drohne und einen Offset der Rotation einzustellen, um das Modell von der gewollten Perspektive beobachten zu können.

7.1.2.4 Livestream-Viewer

Im Livestream-Viewer soll das aufgezeichnete Video der Kamera auf der Drohne, so gut wie möglich in Echtzeit angezeigt werden. Dieser RTMP-Stream (Real Time Messaging Protocol) wird von der Groundstation erzeugt und über einen NGINX-RTMP-Server verbreitet. Dieser kann zuletzt mit einem VLC-Plugin und der Livestream-URL in der App angezeigt werden. In diesem Fall sieht die URL des Livestreams wie folgt aus:

```
rtmp://<IP-Adresse der Groundstation>:<Port>/<Livestream Zusatz>
```

- Die IP-Adresse der Groundstation kann vor Start eines Fluges eingegeben werden, um auch Kompatibilität mit anderen RTMP-Streams sicherzustellen
- Der Port ist zwar meistens „1935“, kann aber abhängig von den Einstellungen beim RTMP-Server abweichen und ist deswegen auch optional einstellbar
- Der Livestream-Zusatz ist ein Zusatz, der beim Streamen via RTMP oft benutzt wird, sodass zwischen mehreren Streams auf dem gleichen Server unterschieden werden kann. Dieser Zusatz startet mit dem sogenannten „Application Name“ der in der Konfiguration des Servers definiert wird, gefolgt vom einem „Stream Key“, welcher bei Start einer Übertragung willkürlich gewählt werden kann. Mit dieser Methode kann am gleichen Server mit der gleichen Host-URL ein Stream mit dem Zusatz XX/YY aber auch mit einem anderen Zusatz wie z.B. AA/BB aufgerufen werden. In unserem Fall wird jedoch nur ein Stream benötigt und der verwendete Zusatz beschränkt daher in jeden Fall auf „live/stream“

7.1.3 UI-Konzept

Das Ziel mit dem Design des User Interfaces war es sowohl ein professionelles als auch ein intuitives Design zu schaffen, um möglichst simpel zwischen verschiedenen Optionen in der App navigieren zu können.

Das Prinzip, welches ich mit dieser App verfolgte, kann man leicht in drei Hauptaspekte unterteilen:

1. Startbereich / Loginbereich

Der Startbereich ist der Bereich der App mit dem ein User als erstes in Kontakt kommt. Dessen Ziel ist es dem User mehrere leichte und klare Optionen zu geben, um sich entweder ein Userkonto zu erstellen oder sich mit einem bereits erstellten Konto anzumelden, um rasch in das Herz der App zu gelangen. In diesem Bereich besteht die Navigation auf Buttons, der automatischen Weiterleitung (z.B. nach Erstellung eines Accounts), oder den „Zurück“-Buttons am Header des Bildschirms, womit man ohne Probleme zum vorherigen Bildschirm gelangt.

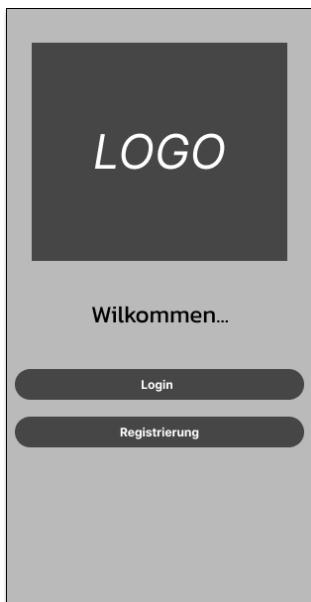


Abbildung 188: Konzept des Willkommensscreens

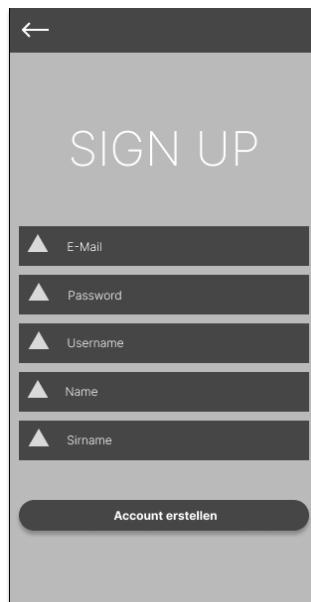


Abbildung 190: Konzept der Registrierungsseite

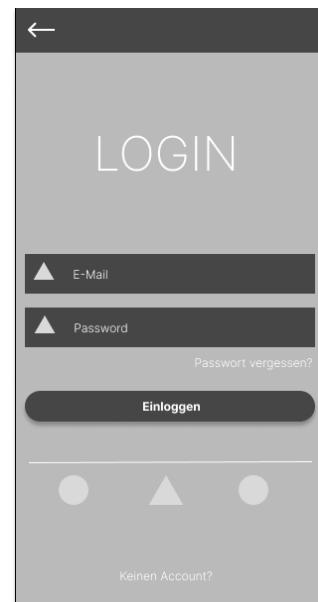


Abbildung 189: Konzept der Loginseite

Hinweis: Die UI-Konzepte, die hier dargestellt sind, wurden in „Figma“ designt und stellen lediglich die Ideen hinter den Hauptprinzipien einer Seite dar. Daher weichen sie auch teilweise mehr oder weniger von der fertigen Umsetzung in der Applikation ab.

2. Homepage / Aufzeichnungsbereich

Die Homepage kann als das Herzstück der App angesehen werden und ist der Ort, wo man den Flug der Drohne in Echtzeit bestaunen kann. Die Navigation in diesem Bereich beschränkt sich dabei auf Navigation-Bars am unteren Ende des Smartphones, womit man zwischen den einzelnen Optionen innerhalb der App jederzeit wechseln kann.

3. Side Menu / Extraoptionen

Das Side Menu, oder auch „Drawer“ genannt, ist eine typische Form des Menüs in modernen Apps, die über die Hauptseite einer Applikation geschoben wird und dem Nutzer weitere meist eher nebensächliche Optionen abseits der Hauptfunktionen der App gibt. In diesem Menü sollen alle Optionen bezüglich der Usersystems (Profil anschauen, Logout,...), Flugdaten anschauen, Einstellungen, FAQ und weitere kleinere Sachen gegeben werden. Dieses Menü soll sowohl über eine Wischbewegung nach rechts als auch über einen Button in der oberen linken Ecke des Hauptmenüs aufrufbar sein.



Abbildung 192: Konzept der Abbildung 191: Konzept
Homepage / Hauptseite des Sidemenüs /
Drawers

7.1.3.1 Farbkombinationen

Um die Nutzererfahrung während der App zu verbessern, wird in vielen modernen Apps und Webseiten darauf geachtet, dass alle benutzen Farben (Text, Hintergründe, Schatten,...) stimmig für das menschliche Auge sind. Zwar gibt es mit der Farbenlehre (Color Theory) einen ganzen Bereich, der sich mit diesem Thema beschäftigt, in unserem Fall gibt es aber bereits Webseiten, die dem User eine Palette an Farben zusammenstellt, die gut zueinander passen und für die bestimmten Nutzungszwecke in einer App abgestimmt sind. Ein Beispiel für so eine Webseite ist <https://www.realtimecolors.com/>, welche auch für die Visualisierungsapp benutzt wurde. Diese Webseite gibt einem die Möglichkeit stimmige Farben für die Nutzung in einem Softwareprojekt bietet.

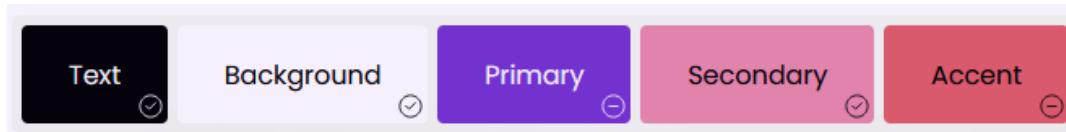


Abbildung 193: Beispiel einer generierten Farbpalette von Realtime Colours [RTEX]

Um diese Farben für sein Projekt zu nutzen, gibt es einerseits die Option die Hex-Farbencodes jeder dieser generierten Farben anzuzeigen, aber es gibt auch eine Exportfunktion, die automatisch Code für viele typische App- und Webseitenframeworks generiert. Unter diesem Framework ist auch Flutter, wobei es die Farben in einem Flutter-spezifischen Objekt zur Setzung von Farben generiert.

Enter custom code with variables. [Learn how.](#)

Presets: [DaisyUI](#) [DaisyUI + Themes](#) [shadcn](#) [NextUI](#) [Flutter](#) [LESS](#) [Bootstrap](#) [Material UI](#)
[Material UI Shades](#) [Chakra UI](#) [Skeleton UI](#) [Vuetify](#)

```
const textColor = Color(${text.hexMtrl});
const backgroundColor = Color(${bg.hexMtrl});
const primaryColor = Color(${primary.hexMtrl});
const primaryFgColor = Color(${primaryFg.hexMtrl});
const secondaryColor = Color(${secondary.hexMtrl});
```

```
const textColor =
Color(0xFF06020b);
const backgroundColor =
Color(0xFFFF6f0fc);
const primaryColor =
Color(0xFF7332cf);
const primaryFgColor =
Color(0xFFFF6f0fc);
const secondaryColor =
Color(0xFFe283ac);
```

Abbildung 194: Flutter Code in Realtime Colors Export [RTEX]

Hinweis: Das Objekt, welches für Flutter generiert wurde, ist **rechts**.

7.1.3.2 8px-Rasterprinzip (8-Point Grid)

Um die Layouts der verschiedenen Komponenten in der Visualisierungsapp so ansprechend wie möglich zu machen, bemühte ich mich für die Platzierung und Größe der verschiedenen Widgets, als auch für die Abstände zwischen den Widgets, das Prinzip namens „8-Point Grid“ anzuwenden. Es ist ein Prinzip im Web- und App-Design welches besagt, dass die Anordnung und Größe von Objekten immer in einem Vielfachen von 8 stattfindet sollte, um ein möglichst schönes Gesamtbild zu erzeugen. Dies inkludiert unter anderem auch die Größe von Texten, Bildern und Icons.

7.1.4 Projektstruktur und -umgebung

7.1.4.1 Editor – Visual Studio Code

Visual Studio Code ist ein plattformübergreifender Texteditor entwickelt von Microsoft und stellt im Jahre 2024 einen der populärsten Code-Editoren auf dem Markt dar. Die Grundinstallation ist zwar recht simple und kann auch wenige typische Aufgaben eines guten Code-Editors erfüllen (z.B. Syntax-Highlighting). Deswegen gibt es im mitgelieferten Extensions-Tab bzw. Marketplace eine riesige Anzahl an Plugins für die meisten Programmiersprachen und Frameworks oder auch anderen diversen Plugins. Unter der Vielzahl von Plugins gibt es auch Viele für die Entwicklung in Flutter:



Abbildung 195: VS-Code Logo [VSCL]

Benutzte Extensions / Erweiterungen:

- **Dart:** Allgemeine Unterstützung für die Programmiersprache Dart
- **Flutter:** Offizielle Unterstützung des Flutter Frameworks in VS-Code. Bietet Code-Formatter, Starten von Apps, Debugging-Tools (Hot-Reload)
- **Awesome Flutter Snippets:** Erstellt automatisch das Skelett eines bestimmten Widgets durch eine Vielzahl an vordefinierten Kürzeln (z.B. statelessW -> erstellt StatelessWidget)

7.1.4.2 Flutter Installation

Für die Installation kann man die Installationsschritte von der offiziellen Webseite befolgen (<https://docs.flutter.dev/get-started/install>, Letzter Aufruf: 28.02.24). In meinem Fall findet die Entwicklung auf einem Windows-Betriebssystem statt, weswegen ich auch nur diese Schritte genauer erläutern werde.

Flutter SDK herunterladen + Pfad hinzufügen

Es gibt 2 Methoden, um die Flutter SDK auf Windows zu installieren. Einerseits über ein „.zip“-File und über das VS-Code Plugin von Flutter. Die Methode, die ich wählte, war die über VS-Code. Diese verlangt einerseits die Installation des Flutter-Plugins und die Erstellung eines neuen Projekts. Dies funktioniert über die Kommandopalette (Strg + Shift + P) durch die Option „Flutter: New Project“. Wird die SDK nicht gefunden, so öffnet sich ein kleiner Dialog, mit den Optionen „Locate SDK“ und „Download SDK“. Drückt man auf die Downloadoption, muss man einen geeigneten Installationsverzeichnis wählen (erhöhte Berechtigungen und keine besonderen Zeichen wie Leerzeichen oder Ähnlichem) (in meinem Fall „C:/src/“) und es startet der automatische Download der SDK. Wurden die benötigten Files installiert, so wird noch gefragt, ob die SDK zum Pfad / PATH hinzugefügt werden soll, was mit einem Klick auf „Add SDK to PATH“ akzeptiert werden kann. Mit Zustimmung kann nun jedes Flutter-Kommando in der Kommandozeile ausgeführt werden.

Android SDK bzw. Android Studio installieren

Um innerhalb von Flutter Android Applikationen entwickeln zu können ist eine Version der Android SDK notwendig. Die leichteste Methode diese zu installieren ist über die Android Studio IDE, wobei die aktuellen SDKs für die Entwicklung mit Android gleich mitinstalliert werden. Die Installation von Android Studio kann über die offizielle Webseite (<https://developer.android.com/studio>) durch Download und Ausführung der Setup-Datei abgeschlossen werden.

Installation überprüfen

Um die Installation zu überprüfen, gibt es für Windows den sogenannten „Flutter doctor“. Dieser kann in einem Terminal mit folgendem Kommando ausgeführt werden:

```
flutter doctor
```

Bei einer fehlerfreien Installation schaut der Output folgendermaßen aus:

```
C:\Users\hinte>flutter doctor
Doctor summary (to see all details, run flutter doctor -v):
[✓] Flutter (Channel stable, 3.16.8, on Microsoft Windows [Version 10.0.19045.4046], locale en-AT)
[✓] Windows Version (Installed version of Windows is version 10 or higher)
[✓] Android toolchain - develop for Android devices (Android SDK version 31.0.0)
[✓] Chrome - develop for the web
[✓] Visual Studio - develop Windows apps (Visual Studio Community 2022 17.5.1)
[✓] Android Studio (version 2020.3)
[✓] VS Code (version 1.86.2)
[✓] Connected device (4 available)
[✓] Network resources

• No issues found!
```

Abbildung 196: Flutter Installation mit "flutter doctor" überprüfen

7.1.4.3 Projekterstellung

Um in VS-Code ein Flutter-Projekt zu erstellen, muss erstmals entweder durch Auswahl des Textfeldes in der oberen Mitte des Fensters oder durch die Tastenkombination Strg+Umschalttaste+P (Ctrl+Shift+P) die Kommandopalette öffnen. In diesem Textfeld lassen sich durch die Eingabe des „>“-Zeichens eine Vielzahl von Kommandos für VS-Code, aber auch für Erweiterungen anzeigen. Mit dem Kommando „Flutter: New Project“ kann nun ein Projekt erstellt werden.

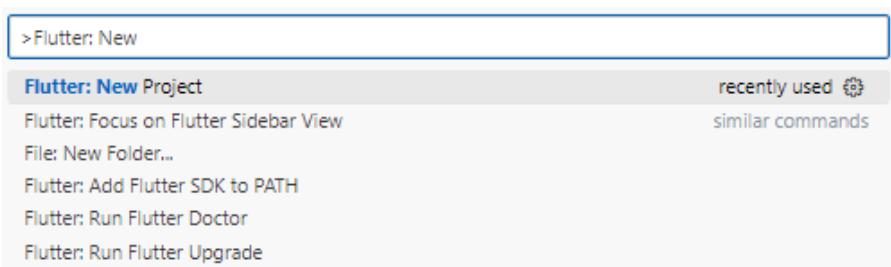


Abbildung 197: VS-Code, neues Flutter-Projekt erstellen

Nach Ausführung dieses Kommandos gibt es nun die Auswahl zwischen einer leeren Applikation (Empty Application) oder zwei Optionen, wo schon etwas Beispielscode als Erklärung bzw. Leitfaden geschrieben ist (Application, Skeleton Application). Zuletzt muss noch ein Verzeichnis und ein Name für das Projekt ausgewählt werden und schon werden alle Files für das Projekt initialisiert.

7.1.4.4 Projektstruktur

Ein Projekt in Flutter besteht aus vielen Ordnern im Verzeichnis, in dem man es erstellt hat. Der Ordner in dem man die eigentlichen Files erstellt und von wo auch das Projekt über das „main.dart“-File startet, ist der „lib“-Ordner. In diesem Ordner bzw. in einem Subordner müssen alle weiteren „.dart“-Files gespeichert werden.

Die Ordnerstruktur der Codefiles wurde im Falle der Visualisierungsapp in mehrere Ordner geteilt, wo alle passenden Files abgelegt wurden:

- **/data:** Provider und allgemeine Klassen, die lediglich zur Datenverarbeitung existieren (Usermodel, Logging, Abspeichern in lokale Files)

- **/extensions:** Extensions, um bereits definierte Klassen zu erweitern (z.B. erleichterter Abruf von Designeigenschaften)
- **/screens:** Alle angezeigten Bildschirme bzw. Ansammlung an mehreren angeordneten Widgets (z.B. Welcome Screen, Login, Homepage, Video Overlay)
- **/widgets:** Alle Widget-Files die ein bereits existierendes Widget für spezielle Anwendungsfälle erweitern (z.B. Nummern-Eingabefeld, Lade-Indikator,...)
- **/service:** Klassen die Methoden für die einfache Kommunikation mit externen Datenbanken, Datenstreams oder Files handeln (z.B. MQTT-Manager, Storage Service, Auth Service,....)
- **/themes:** Alle Files die lediglich designtechnischen Code beeinhalten (z.B. Light- / Dark-Mode, Konstanten für Design)

7.1.4.5 Versionsmanagement / Git + GitHub

Um ein Projekt auf mehreren Geräten entwickeln zu können und eine zentrale Verwaltung dieses Projekts mit Zugriff auf alle Änderungen zu ermöglichen benutzt die Visualisierungsapp das Versionsmanagementtool „Git“. In Kombination mit Git wird zusätzliche noch GitHub verwendet, um das Projekt auf einem externen Server hochzuladen zu können und somit auch von überall darauf Zugriff zu haben

7.1.4.6 Packages

Pubspec.yaml

Das „pubspec.yaml“-File stellt in Flutter die Konfigurationsdatei für ein Projekt dar. In diesem File werden, Informationen über das Projekt gespeichert, es fast alle über pub.dev installierte Packages zusammen und es müssen alle im Projekt verwendeten Assets (Bilder, Videos, Icons, Fonts) mit dem relativen Pfad (vom Projektverzeichnis aus) definiert werden.

Installieren neuer Packages

Um neue Packages zu installieren, nutzt man den Package-Installer von Dart (pub.dev) im Terminal des Projektverzeichnisses:

```
flutter pub add <package_name>
```

Der Name eines Pakets (<package_name>) ist für jedes Paket einzigartig. Die Pakete sind auf der Webseite <https://pub.dev/> gesammelt und können nach einer Vielzahl an Kriterien gefiltert werden (Name, Bewertungen, Kompatibilität,...).



Abbildung 198: pub.dev Webseite

Finales pubspec.yaml-File

Im fertigen Puspec.yaml-File zeigen sich die einzigen Komponenten des fertigen Konfigurationsfiles der Visualisierungsapp. Es unterteilt sich in folgende Hauptkomponenten:

- **name:** Name der Applikation, automatisch vergeben bei Projektkreation
- **description:** Kurze Beschreibung des Projekts
- **publish_to:** Bei einer normalen Applikation "none". Bei Erstellung eines Plugins für pub.dev nicht vorhanden.
- **version:** Version des Projekts.
- **environment:** Definiert den Bereich der möglichen Flutter Versionen, mit denen dieses Projekt abspielbar ist.
- **dependencies:** Block zur Definition von benutzten Plugins, inklusive Version.
- **flutter_native_splash / flutter_launcher_icons:** Blöcke zur Einstellung von 2 benutzten Plugins.
- **flutter:** In diesem Block wird das benutzte Design, die relativen Pfade zu den Assets und den Fonts definiert.

```
Dateiname: pubspec.yaml
name: drone_2_0
description: A visualisation App for the FPV-Drone diploma project
publish_to: 'none' # Remove this line if you wish to publish to pub.dev
version: 1.0.0+1
environment:
  sdk: '>=3.0.2 <4.0.0'

dependencies:
  flutter:
    sdk: flutter

  firebase_auth: ^4.9.0
  firebase_core: ^2.15.1
  syncfusion_flutter_gauges: ^22.2.12
  cloud_firestore: ^4.9.1
  provider:
    logger: ^2.0.2
  google_nav_bar: ^5.0.6
  firebase_storage: ^11.2.7
  file_picker: ^5.5.0
  cached_network_image: ^3.2.3
  firebase_database: ^10.2.7
  syncfusion_flutter_charts: ^22.2.12
  async: ^2.11.0
  flutter_native_splash: ^2.3.3
  google_sign_in: ^6.1.5
  path_provider: ^2.1.1
  flutter_keyboard_visibility: ^5.4.1
  flutter_animate: ^4.3.0
  mqtt_client: ^10.0.1
  flutter_vlc_player: ^7.2.0
```

```

flutter_cube: ^0.1.1
shared_preferences: ^2.2.2
image_picker: ^1.0.7
image_gallery_saver: ^2.0.3

dev_dependencies:
  flutter_test:
    sdk: flutter
  flutter_lints: ^2.0.0
  flutter_launcher_icons: ^0.13.1

flutter_native_splash:
  image: "assets/images/logo_dark.png"
  color: "#e3e9ed"
  image_dark: "assets/images/logo_light.png"
  color_dark: "#12181c"
  fullscreen: true

flutter_launcher_icons:
  image_path: "assets/images/icon.png"
  android: true
  ios: true
  remove_alpha_ios: true

flutter:
  uses-material-design: true
  assets:
    - assets/
    - assets/images/
    - assets/images/company_logos/
    - assets/images/dronetech/
    - assets/images/weather_icons/dark/
    - assets/images/weather_icons/light/
    - assets/loading/
    - assets/fonts/
    - assets/videos/
    - assets/models/
  fonts:
    # Differentiation of Weights based on naming convention
    - family: Raleway
      fonts:
        - asset: assets/fonts/raleway/Raleway-ExtraLight.ttf
    - family: VCR_OSD_Mono
      fonts:
        - asset: assets/fonts/vcr_osd_mono/vcr_osd_mono.ttf
    - family: Moderne-Sans
      fonts:
        - asset: assets/fonts/moderne_sans.ttf
    - family: Switzer
      fonts:

```

```

# regular
- asset: assets/fonts/switzer/Switzer-Regular.otf
- asset: assets/fonts/switzer/Switzer-Light.otf
- asset: assets/fonts/switzer/Switzer-Semibold.otf
- asset: assets/fonts/switzer/Switzer-Bold.otf
- asset: assets/fonts/switzer/Switzer-Extrabold.otf

# italic
- asset: assets/fonts/switzer/Switzer-Italic.otf
- asset: assets/fonts/switzer/Switzer-LightItalic.otf
- asset: assets/fonts/switzer/Switzer-SemiboldItalic.otf
- asset: assets/fonts/switzer/Switzer-BoldItalic.otf
- asset: assets/fonts/switzer/Switzer-ExtraboldItalic.otf

```

7.1.4.7 App Icon

Eines der kleinstes aber wichtigsten Aspekte einer modernen App ist ein Icon auf dem Homescreen des Smartphones der User, um möglichst gut im Gedächtnis der Benutzer zu bleiben. Zwar bietet Flutter von Haus aus ein Standardlogo, dieses ist aber sehr ineffektiv in der oben beschriebenen Aufgabe und ist allgemein kein professioneller Weg seine App zu vermarkten. Deswegen haben wir Teil unseres Logos zu einem kleinen Icon umfunktioniert und benutzen jenes anhand des „flutter_launcher_icons“-Package als Icon.

Nach Installation des Packages müssen die Eigenschaften als auch ein Pfad zu einem Icon im pubspec.yaml-File definiert werden. Die Konfiguration für Smartphones schaut in unserem Fall wie folgt aus:

```

flutter_launcher_icons:
  image_path: "assets/images/icon.png"
  android: true
  ios: true
  remove_alpha_ios: true

```

Abbildung 199: Konfiguration von "flutter_launcher_icons"

Zuallerletzt muss das Package noch ausgeführt werden damit die verschiedenen großen Bilder für alle möglichen Situationen erstellt werden. Dies passiert im Projektverzeichnis mit folgendem Terminal-kommando:

```
dart run flutter_launcher_icons
```

Bei richtiger Installation und Konfiguration, gibt das Terminal folgenden Text aus:

```
Building package executable... (11.8s)
Built flutter_launcher_icons:flutter_launcher_icons.

FLUTTER LAUNCHER ICONS (v0.13.1)

• Creating default icons Android
• Overwriting the default Android launcher icon with a new icon
• Overwriting default iOS launcher icon with new icon
No platform provided
```

Abbildung 200: Ausführung des "flutter_launcher_icons"-Packages im Projektverzeichnis

Das finale Icon der App sieht folgendermaßen aus, auch wenn es aufgrund von bestimmten Einstellungen Betriebssystems teilweise etwas zugeschnitten wird:

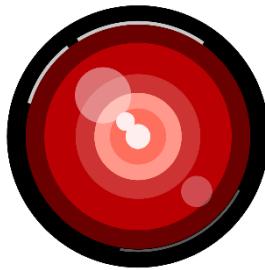


Abbildung 202: App-Icon
der Visualisierungsapp



Abbildung 201: Abgeschnittenes Icon
durch Android

7.1.4.8 Splash + Willkommensscreen

Startet man die Applikation erstmals so trifft man auf den Splash Screen. Ein Bildschirm, welcher angezeigt wird während anderer Bildschirme der App im Hintergrund laden. Dieser Bildschirm ist nicht interaktiv und dient lediglich einem schöneren und kompletten Design, da ansonsten nur ein 1-farbiger Bildschirm oder ungeladene Elemente angezeigt werden würden.

7.1.4.9 Native Splash Screen

Native Splash Screen oder „flutter_native_splash“ ist ein Plugin, mit dem es möglich ist, einen Standard „Splash Screen“ der Applikation vor dem Laden der Applikation anzuzeigen Zwar bietet dies Plugin nur die Möglichkeit nur die Option für statische Bilder als Splash-Screen, diese können aber mit einem einzigen Kommando im Terminal für alle gewollten Zielplattformen generiert werden.

Um angezeigtes Bilder und Hintergründe dieses Bildschirmes zu definieren muss ein Block im pubspec.yaml-File definiert



Abbildung 203: Splash Screen der
Visualisierungsapp

werden, wo ein Pfad zu einem Bild und andere Eigenschaften eingestellt werden können.

```
flutter_native_splash:
  image: "assets/images/logo_dark.png"
  color: "#e3e9ed"
  image_dark: "assets/images/logo_light.png"
  color_dark: "#12181c"
  fullscreen: true
```

Abbildung 204: Konfiguration des Packages "flutter_native_splash" im pubspec.yaml-File

Um die Splash Screens zu erzeugen oder Änderungen an den Einstellungen beim Starten der App zu sehen, muss folgender Command im Terminal des Projektordners ausgeführt werden:

dart run flutter_native_splash:create

Bei erfolgreicher Installation des Plugins und einer korrekten Definition von Eigenschaft im pubspec.yaml-File, wird folgender Output am Terminal angezeigt:

```
Building package executable... (41.3s)
Built flutter_native_splash:create.
[Android] Creating default splash images
[Android] Creating dark mode splash images
[Android] Updating launch background(s) with splash image path...
[Android] - android/app/src/main/res/drawable/launch_background.xml
[Android] - android/app/src/main/res/drawable-night/launch_background.xml
[Android] - android/app/src/main/res/drawable-v21/launch_background.xml
[Android] - android/app/src/main/res/drawable-night-v21/launch_background.xml
[Android] Updating styles...
[Android] - android/app/src/main/res/values-v31/styles.xml
[Android] - android/app/src/main/res/values-night-v31/styles.xml
[Android] - android/app/src/main/res/values/styles.xml
[Android] - android/app/src/main/res/values-night/styles.xml
[iOS] Creating images
[iOS] Creating dark mode images
[iOS] Updating ios/Runner/Info.plist for status bar hidden/visible
Web folder not found, skipping web splash update...

✓ Native splash complete.
Now go finish building something awesome! 🚀 You rock! 🎉
Like the package? Please give it a 👍 here: https://pub.dev/packages/flutter\_native\_splash
```

Abbildung 205: Ausführung von "flutter_native_splash" bzw. Generierung der Splash Screens

7.1.5 Startbereich

7.1.5.1 Initialisierung der App

Bevor dem User eine interaktive UI angezeigt wird und während des Splash-Screens, werden im „main.dart“-File bzw. am Startpunkt des Projekts eine Vielzahl an essenziellen Funktionen geladen. Die wichtigste darunter für den User bezieht sich auf das Laden eines vorherigen eingeloggten Accounts bzw. dem automatischen Login. Der allgemeine Ablauf der Initialisierung ist im unteren Flowchart beschrieben.

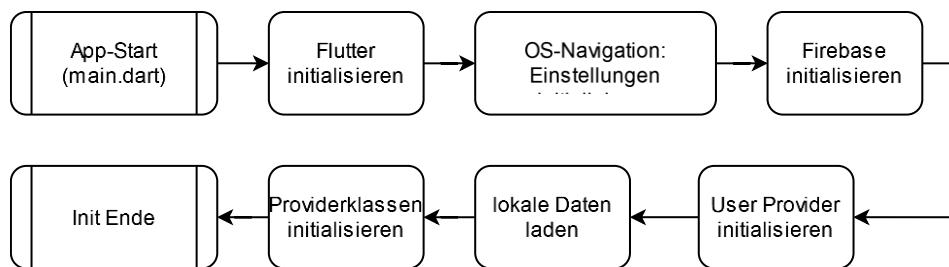


Abbildung 206: Teile der Initialisierung der Visualisierungsapp

Die äquivalenten Funktionen im eigentlichen Programmcode sind im unteren Code mit den grünen Kommentaren beschrieben:

Dateiname: main.dart; Funktion: main
<pre> void main() async { // Flutter Initialisieren WidgetsFlutterBinding.ensureInitialized(); // OS-Einstellungen defaultAppSettings(); // Initialize App await Firebase.initializeApp(options: DefaultFirebaseOptions.currentPlatform); AuthenticationProvider authProvider = AuthenticationProvider(); User? user = authProvider.currentUser; await authProvider.initUser(); final ThemeManager themeManager = ThemeManager(); await themeManager.initThemeSettings(user?.uid ?? ""); // Initialize colors after theme loaded appNavigationColorSettings(themeManager.isDark); // Init cached data final DataCache dataCache = DataCache(); await dataCache.initData(); // Defintion of Provider-Classes + Running of actual Application runApp(MultiProvider(providers: [ChangeNotifierProvider(</pre>

```

        create: (_) => authProvider,
),
ChangeNotifierProvider(
    create: (_) => themeManager,
),
ChangeNotifierProvider(
    create: (_) => dataCache,
),
],
child: MyApp(
    user: user,
),
));
}
}

```

Unter diesen Prozessen bei der Initialisierung fallen:

- Allgemeine App-Einstellung initialisieren:
 - Smartphone Orientierung fixieren (setPreferredOrientations)
 - Overlay des Betriebssystems mit Dark Mode initialisieren (colorSettings)
 - Navigationsleisten verstecken (SystemUiMode.edgeToEdge)

```

Dateiname: app_settings.dart; Funktion: defaultAppSettings
void defaultAppSettings({bool darkMode = true}) {
    // App Navigation Colors
    appNavigationColorSettings(darkMode);

    // Bottom and Top Navigation Bar hidden -> available via Swipe
    SystemChrome.setEnabledSystemUIMode(SystemUiMode.edgeToEdge);

    // Default App Orientation -> independent from Phone Orientation
    SystemChrome.setPreferredOrientations([
        DeviceOrientation.portraitUp,
        DeviceOrientation.portraitDown,
    ]);
}

```

- Firebase Plugins für spezifische Plattform initialisieren (Firebase.initializeApp)
- User- / Authentication-Provider initialisieren: User angemeldet? Userdaten initialisieren
- Userspezifische Designeinstellungen initialisieren (ThemeManager): Einstellungen von Userdokument laden und setzen (nur falls User schon angemeldet ist) (initThemeSettings)
- Datencache initialisieren: lokal gespeicherte Daten laden (Flugdaten und Alter der Daten lesen via „readFile“-Funktion,..)

```

Dateiname: data_cache.dart

```

```

Future<void> initData() async {
    final LocalStorageService localStorage = LocalStorageService();

    // Reading local data
    final data = await localStorage.readFile("flight_records.dat") ?? [];
    _previousFlights = data;

    // Reading age of data
    Map? dataAges = await localStorage.readFile("data_ages.dat");

    if (dataAges != null) {
        _dataAges = dataAges;
    }
}

```

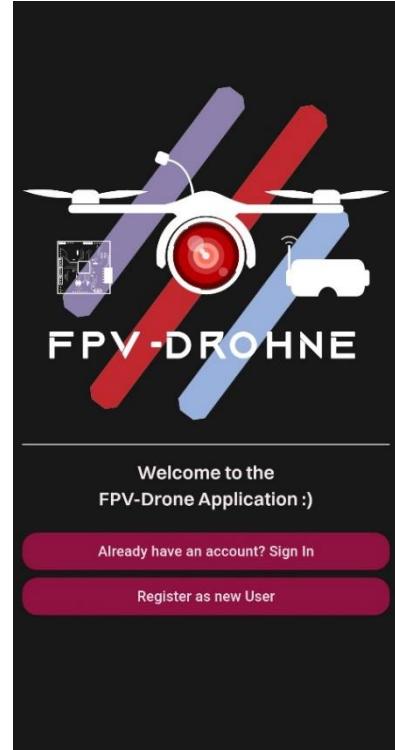
- Provider initialisieren (MultiProvider....)
- Initialisierte Userobjekt an das Hauptwidget („MyApp“) übergeben (user: user)

7.1.5.2 Willkommensbildschirm

Der Willkommensbildschirm ist der erste Interaktive Teil der Applikation und dient als Weggabel zwischen der Registrierung eines neuen Useraccounts und des Logins für User mit einem bestehenden Account. Zusätzlich zu dieser Option wird noch das Logo der Diplomarbeit und eine kleine Willkommensnachricht angezeigt.

Logo-Design

Um ein einzigartiges Logo für unsere Diplomarbeit zu haben ohne irgendwelche Lizenzen oder Ähnlichem, entschieden wir unser eigenes Logo zu designen. Das Logo wurde von Grund auf mit dem Programm „Affinity Photo“ designt und hatte als Ziel unsere Diplomarbeit mit einem kurzen Blick zu beschreiben. Das Logo ist einer der Hauptaspekte des Willkommensbildschirms, aber auch in anderen Teilen der App, wie auch der Diplomarbeit im Allgemeinen zu finden.



*Abbildung 207:
Willkommensbildschirm*

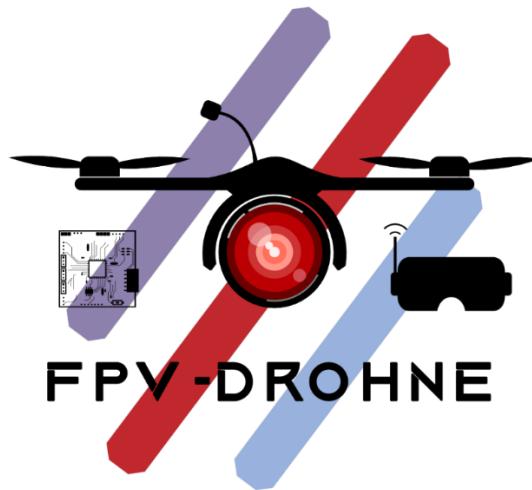


Abbildung 208: Logo der Visualisierungsapp

7.1.5.3 Login- u. Registrierung-Screens

Login Flussdiagramm

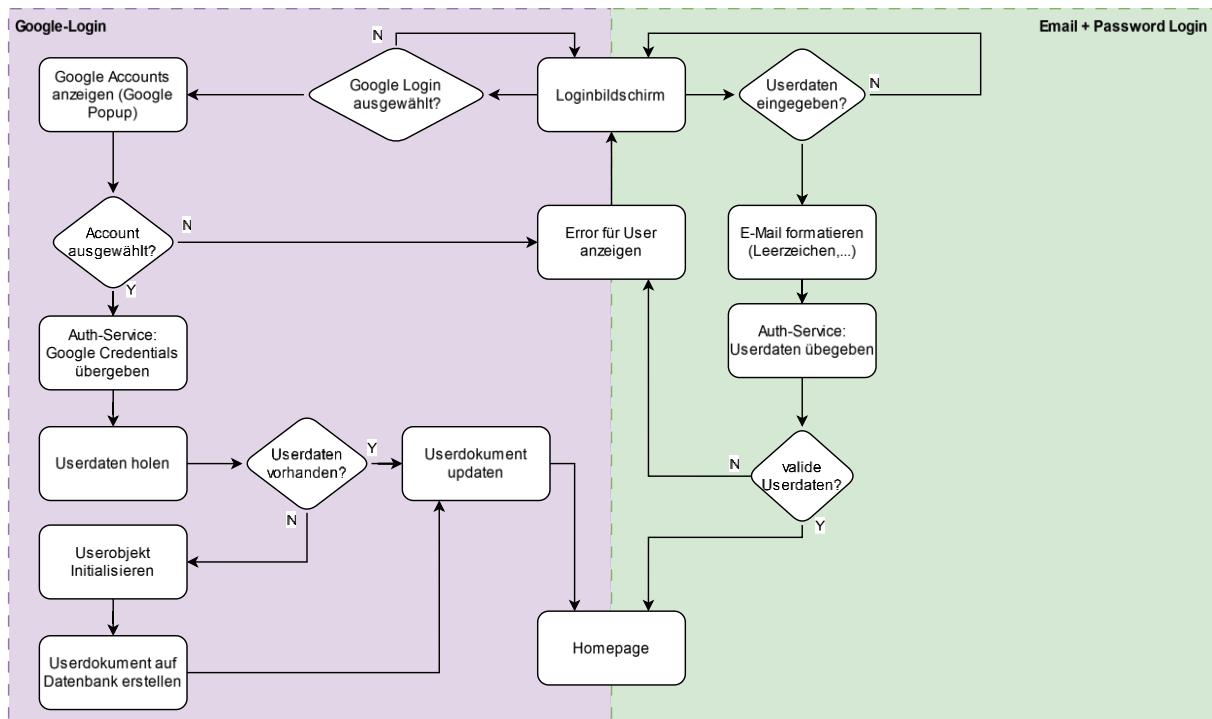


Abbildung 209: Flussdiagramm Loginbildschirm

Im obigen Flussdiagramm zeigen sich die 2 Loginmethoden die innerhalb der Visualisierungsapp unterstützt werden und die Schritte, welche im Hintergrund passieren, um diese umzusetzen. Einerseits der über ein Google Account (links) und klassisch über E-Mail + Passwort (rechts). Nach dem eigentlichen Prozess des Logins gibt es keine Unterschiede für den angemeldeten User.

Login Screen Widget

Aufgrund des Aufbaus einer App innerhalb des Flutter Frameworks und die zusammenhängende Syntax, um eine schöne UI umzusetzen, werden die Files ziemlich lang. Deswegen habe ich mich entschieden nur einen einzigen Bildschirm, den Login-Screen, als Ganzes in diese Dokumentation zu inkludieren. Es soll dienen zu zeigen, wie ein ganzes Widget mit all dem nötigem Code drumherum aussieht.

```
Dateiname: login.dart;
// ignore_for_file: use_build_context_synchronously

import 'package:drone_2_0/data/providers/auth_provider.dart';
import 'package:drone_2_0/data/providers/logging_provider.dart';
import 'package:drone_2_0/screens/login/registration.dart';
import 'package:drone_2_0/screens/login/reset_password.dart';
import 'package:drone_2_0/service/auth/auth_error_handler.dart';
import 'package:drone_2_0/service/auth/auth_service.dart';
import 'package:drone_2_0/screens/homepage/homepage.dart';
import 'package:drone_2_0/themes/theme_constants.dart';
import 'package:drone_2_0/widgets/animations/animation_routes.dart';
import 'package:drone_2_0/widgets/loading_icons.dart';
import 'package:drone_2_0/widgets/utils/error_bar.dart';
import 'package:drone_2_0/widgets/utils/helper_widgets.dart';
import 'package:drone_2_0/extensions/extensions.dart';
import 'package:firebase_auth/firebase_auth.dart';
import 'package:flutter/material.dart';
import 'package:flutter/services.dart';
import 'package:provider/provider.dart';
import '.../widgets/input.dart';

class LoginScreen extends StatefulWidget {
  const LoginScreen({Key? key}) : super(key: key);

  static String id = 'LoginScreen';

  @override
  LoginScreenState createState() => LoginScreenState();
}

class LoginScreenState extends State<LoginScreen> {
  final TextEditingController _emailController = TextEditingController();
  final TextEditingController _passwordController = TextEditingController();

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      resizeToAvoidBottomInset: false,
      appBar: AppBar(
        title: const Text('Return'),
      ),
      body: Container(
        margin: const EdgeInsets.all(Margins.stdMargin),
        child: Column(
          crossAxisAlignment: CrossAxisAlignment.center,
          mainAxisAlignment: MainAxisAlignment.center,
          children: [
            const Spacer(),
            Text(
              "LOGIN",
              style: context.textTheme.headlineLarge,
            ),
            const VerticalSpace(height: 128),
            StdInputField(
              width: MediaQuery.sizeOf(context).width,
              hideText: false,
            ),
          ],
        ),
      ),
    );
  }
}
```

```

        controller: _emailController,
        hintText: "Email",
        icon: Icons.person,
        onSubmitFunction: (String text) {},
),
const VerticalSpace(height: 8),
StdInputField(
    width: MediaQuery.sizeOf(context).width,
    hideText: true,
    controller: _passwordController,
    hintText: "Password",
    icon: Icons.password,
    onSubmitFunction: (String text) {},
),
TextButton(
    onPressed: () {
        Navigator.of(context).pushReplacement(
            pageRouteAnimation(const ResetPassword()),
        );
    },
    child: Align(
        alignment: Alignment.bottomRight,
        child: Text(
            "Forgot Password?",
            style: context.textTheme.labelSmall,
        ),
    ),
),
const VerticalSpace(height: 8),
SizedBox(
    width: double.infinity,
    child: ElevatedButton(
        onPressed: () async {
            _loginWithEmailAndPassword(
                context, _emailController.text, _passwordController.text);
        },
        child: const Text('Login'),
    ),
),
const VerticalSpace(height: 64),
Row(
    children: [
        Expanded(
            child: Divider(
                thickness: 1,
                color: context.colorScheme.onBackground,
            )),
        Padding(
            padding: const EdgeInsets.symmetric(horizontal: 8.0),
            child: Text(
                "Or Sign in with",
                style: context.textTheme.displaySmall,
            ),
        ),
        Expanded(
            child: Divider(
                thickness: 1,
                color: context.colorScheme.onBackground,
            ))
    ],
),
const VerticalSpace(),
Row(
    mainAxisAlignment: MainAxisAlignment.spaceEvenly,
    children: [
        GestureDetector(
            onTap: () {
                _googleLogin(context);
            }
        )
    ],
)
);

```

```

        },
        child: const Image(
            image: AssetImage(
                "assets/images/company_logos/google_dark_normal.png"),
            width: 32,
            height: 32,
        ),
    ),
),
],
),
const Spacer(),
TextButton(
    onPressed: () {
        Navigator.of(context).push(
            pageRouteAnimation(const Registration()),
        );
    },
    child: Text("Don't have an account? Sign up",
        style: context.textTheme.bodySmall),
),
),
),
),
);
}

@Override
void dispose() {
    _emailController.dispose();
    _passwordController.dispose();
    super.dispose();
}
}

```

E-Mail + Passwort – Login

Beim Login mit E-Mail und einem Passwort wird ein bereits erstelltes Konto vorausgesetzt. Nach Eingabe der Daten werden die Daten mit Firebase-Auth gecheckt, ob die E-Mail-Adresse als Account existiert und ob das Passwort dazu passt. Gibt es irgendeinen Error so gibt es ein paar definierte, als auch eine allgemeine Error-Nachricht, welche dem User das Problem aufzeigen sollen.

Dateiname: auth_service.dart; Funktion: registration
<pre> Future<AuthStatus?> registration({ required String email, required String password, required String username, required String name, required AuthenticationProvider authProvider, }) async { try { // Create User with Email and Password UserCredential userCredentials = await FirebaseAuth.instance.createUserWithEmailAndPassword(email: email, password: password,); // Set Additional Value for new Auth User Account } } </pre>

```

User? user = userCredentials.user;
//user?.sendEmailVerification();
await authProvider.createUser(
    user!,
    UserDataModel(
        email: email,
        userId: user.uid,
        fullName: name,
        storagePath: '',
        username: username,
    ));
}

return AuthStatus.successful;
} on FirebaseAuthException catch (e) {
    return AuthExceptionHandler.handleAuthException(e);
} catch (e) {
    return AuthStatus.unknown;
}
}
}

```

Google-Login

Beim Login mit einem existierenden Google-Account, wird ein Dialog über der App aufgerufen, mit dem ein bereits existierender Google-Account zum Login benutzt werden kann. Dies macht den Prozess des Logins um einiges schneller und simpler, da eine Mehrheit der Benutzer insbesondere auf Android bereits so einen Account besitzen und lediglich ihren gewünschten Account auswählen müssen. Innerhalb der App gibt es keine Unterschiede im Vergleich zum Login mit E-Mail + Passwort.

Dateiname: login.dart; Funktion: _googleLogin
<pre> void _googleLogin(BuildContext context) async { final GlobalKey<State> loaderDialogue = GlobalKey<State>(); LoaderDialog.showLoadingDialog(context, loaderDialogue); try { final UserCredential? credentials = await AuthService().signInWithGoogle(); if (context.mounted) { if (credentials != null) { // initializing document for user User? user = credentials.user; context.read<AuthenticationProvider>().createAltLoginDocument(user!); _navigate(context); } else { Navigator.pop(context); ScaffoldMessenger.of(context).showSnackBar(defaultSnackbar("Google Sign In Error"),); } } } } </pre>

```
        }
    } on PlatformException catch (e) {
        Logging.error(e);
    }
}
```

Passwort vergessen

In dem Fall, dass ein User das Passwort seines Accounts vergessen hat, kann dieser jenes mit seiner E-Mail-Adresse wieder zurücksetzen. Hierfür wird nach Eingabe der jeweiligen E-Mail des betroffenen Accounts eine automatisierte E-Mail ins Postfach des Users geschickt. Innerhalb dieser Nachricht gibt es schließlich einen Link, mit dem es möglich ist, das vergessene Passwort zu ändern.

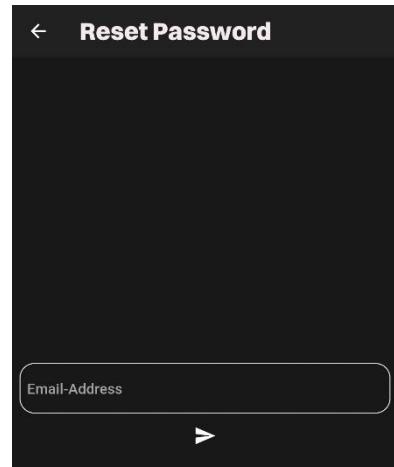


Abbildung 210: Ausschnitt des "Passwort vergessen"-Bildschirms

Dateiname: auth provider.dart; Funktion: resetPassword

```
Future<String> resetPassword({required String email}) async {
    String message = "Sent Email";
    await _auth.sendPasswordResetEmail(email: email).then((value) {
        return true;
    }).catchError((e) {
        Logging.error("Reset Password Exception: $e");
        message = e.toString();
        return false;
    });
    return message;
}
```

Im Code wird beim Abschicken der E-Mail die „`resetPassword`“-Funktion im AuthProvider ausgeführt. Diese übergibt die eingegebene E-Mail an die vordefinierte „`sendPasswordResetEmail`“-Funktion des Authentication Objekts und das restliche Prozedere läuft unabhängig von der Visualisierungsapp bzw. meinem Code. Für das Projekt wurde die automatisierte E-Mail von Firebase-Authentication personalisiert. Um dies zu tun kann man in seinem Firebase-Projekt unter *Authentication > Vorlagen > Passwortzurücksetzung* das reine HTML der versendeten E-Mail, aber auch z.B. auch den Namen des Absenders nach seinem Verlangen anpassen.

Die E-Mail die schließlich im Postfach des Users landet, beeinhaltet einen Link zu einer von Firebase bereitgestellten Webseite auf welcher der User sein neues Passwort eingeben kann.

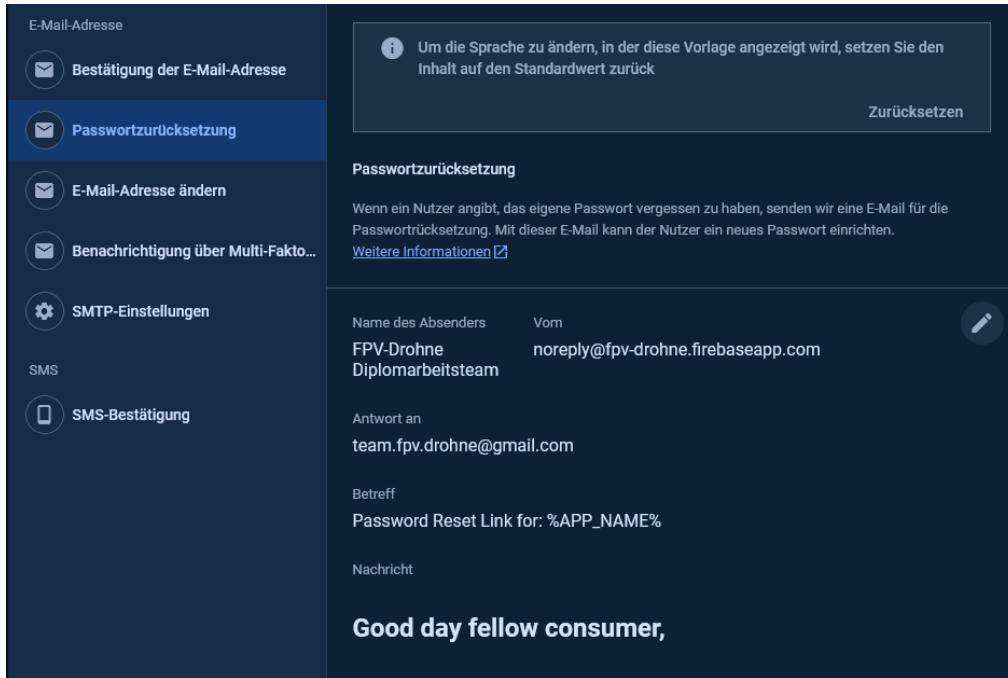


Abbildung 211: Anpassung der E-Mail zur Passwortzurücksetzung in Firebase

Good day fellow consumer,

Follow this link to reset your FPV-Drone-App password.

...

[Reset your Password!!](#)



If you didn't ask to reset your password, you can ignore this email.

Thanks,

Your FPV-Drohne-DA-2024 team

Reset your password

for hinterbergersebastian@gmail.com

New password

SAVE

Abbildung 213: E-Mail zur Rücksetzung des Userpasswords

Abbildung 212: Passwort zurücksetzen
Webseite

Registrierung

Bei der Registrierung kann ein neuer Account mittels eines Namens, E-Mail und einem Passwort erstellt werden. Bei erfolgreicher Anmeldung wird im Hintergrund zusätzlich ein Userdokument von Firestore initialisiert, um Userdaten abzuspeichern.

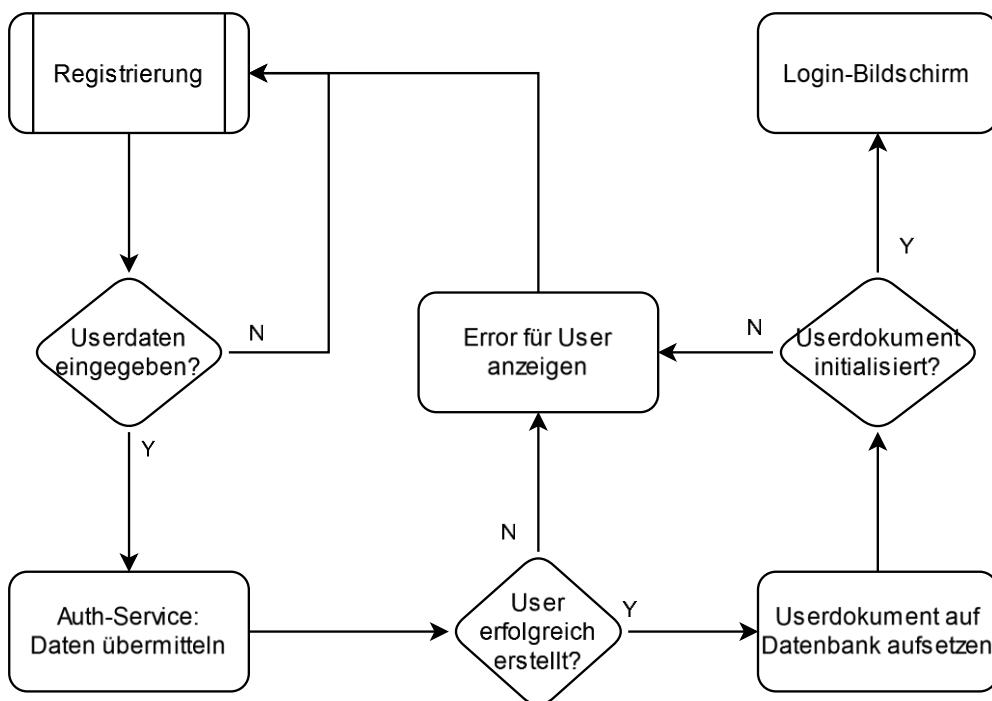


Abbildung 214: Flussdiagramm der Registrierung

Die einzigen Beschränkungen bei Registrierungsprozess sind:

- Die E-Mail-Adresse muss richtig formatiert sein
- Die E-Mail-Adresse darf nicht bereits von einem vorhandenen Account benutzt werden
- Das Passwort muss mindestens aus 6 Charakteren bestehen
- Die restlichen Felder dürfen NICHT leer sein

Werden diese Kriterien nicht eingehalten so kommt eine dazugehörige Fehlermeldung in der Form einer Snackbar am unteren Teil des Bildschirms.

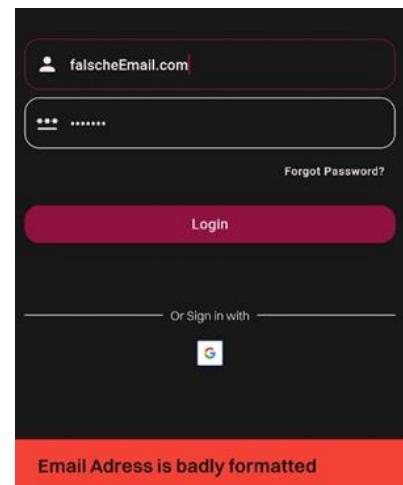


Abbildung 215: Autorisierungsfehler, invalides E-Mail-Format

7.1.5.4 Autorisierungsfehlermeldungen

Werden die Kriterien beim Login- / Registrierungsprozess nicht eingehalten, so werden bestimmte vordefinierte Fehlermeldungen angezeigt, um den User Feedback zu geben:

- Ungültige E-Mail (Formatierung der E-Mail-Adresse fehlerhaft)
- Falsches Passwort (Passwort inkorrekt zu eingegebenem Account)
- Zu schwaches Passwort (Passwort hat weniger als 6 Zeichen)

- E-Mail schon in Benutzung (E-Mail gehört bereits einem anderen Account)
- Unbekannter Fehler

Die häufigsten Fehler werden dem User direkt angezeigt. Bei anderen Problemen wie z.B. eines fehlerhaften Serverstatus oder einer fehlenden Internetverbindung wird lediglich ein „Unbekannter Fehler“ angezeigt.

Die Anzeige der Fehlermeldungen erfolgt in all diesen Fällen wie in anderen Apps mithilfe einer Snackbar (*siehe Hinweis*), die bei Auftreten eines Fehlers am unteren Bildschirmrand mit der jeweiligen Nachricht aufpoppt.

Hinweis: Eine Snackbar ist eine kleine Box mit einer Nachricht, die nach einem bestimmten Event am unteren Rand einer Applikation aufpoppt, um den User Feedback über den Status bzw. Auskunft jenes Events zu geben (z.B. Fehlermeldungen, erfolgreicher Login,..)

Dateiname: auth_error_handler.dart

```
import 'package:firebase_auth/firebase_auth.dart';

enum AuthStatus {
    successful,
    wrongPassword,
    emailAlreadyExists,
    invalidEmail,
    weakPassword,
    unknown,
}

class AuthExceptionHandler {
    static AuthStatus handleAuthException(FirebaseAuthException e) {
        AuthStatus status;
        switch (e.code) {
            case "invalid-email":
                status = AuthStatus.invalidEmail;
                break;
            case "wrong-password":
                status = AuthStatus.wrongPassword;
                break;
            case "weak-password":
                status = AuthStatus.weakPassword;
                break;
            case "email-already-in-use":
                status = AuthStatus.emailAlreadyExists;
                break;
            default:
                // any other error ->
                https://firebase.google.com/docs/reference/js/auth?hl=de#autherrorcodes
                status = AuthStatus.unknown;
        }
        return status;
    }
}
```

```

}

static String generateErrorMessage(error) {
    String errorMessage;
    switch (error) {
        case AuthStatus.invalidEmail:
            errorMessage = "Email Adress is badly formatted";
            break;
        case AuthStatus.weakPassword:
            errorMessage = "Your Password must be atleast 6 characters";
            break;
        case AuthStatus.wrongPassword:
            errorMessage = "Wrong password or email";
            break;
        case AuthStatus.emailAlreadyExists:
            errorMessage = "Email Adress already in use";
            break;
        default:
            errorMessage = "An unknown error occured";
    }
    return errorMessage;
}
}
}

```

7.1.5.5 Adaptiertes Eingabefeld (StdInputField)

Um überall innerhalb der Visualisierungsapp aber insbesondere im Startbereich ein gleichmäßiges Verfahren für die Eingabe von Daten via Textfelder zu haben, entschied ich mich für das von Flutter bereitgestellte „Textfield“-Widget einen Wrapper in Form eines maßgeschneiderten Widgets zu programmieren. Dieses Stateless-Widget hat den Namen „StdInputField“ und dient dazu dem standardmäßigen Textfeld eine simple Initialisierung und ein standardisiertes Design in allen Teilen der App zu ermöglichen. Unter den im Konstruktor einstellbaren Parametern gibt es folgende:

- **hintText (String):** Der teils durchsichtige Text, der angezeigt wird, wenn das Feld leer ist.
- **width (double):** Die breite des Textfeldes.
- **hideText (bool):** Erlaubt den eingegebenen Text zu verstecken und nur per repräsentierenden Zeichen anzuzeigen. Wird normalerweise für Passwortfelder oder ähnlichen kritischen Eingaben verwendet.
- **controller (TextEditingController):** Der „controller“ des Textfeldes ist vom Typen vordefinierten Typen „TextEditingController“ und muss für jede Art von Textfeld in Flutter definiert sein. Er dient dazu mit dem jeweiligen Textfeld programmiertechnisch zu kommunizieren, um z.B. Zugriff auf dem eingegebenen Text per Code zu haben, oder allgemein den Text im Feld steuern zu können.
- **icon (IconData):** Falls verlangt wird ein Icon links von dem Textfeld anzuzeigen, kann man ein von Flutter vordefiniertes Icon (z.B. Icons.abc,..) übergeben.
- **onSubmitFunction (Function(String)):** Eine Funktion aus einem anderen File kann übergeben werden solange die Übergabeparameter (1 String) übereinstimmen. Diese

Funktion wird schließlich automatisch aufgerufen, falls der „onSubmitted“-callback des Textfeldes eintritt (Enter gedrückt, fertige Eingabe,...).

```
Dateiname: input.dart
import 'package:drone_2_0/extensions/extensions.dart';
import 'package:flutter/material.dart';
import 'package:flutter/services.dart';

class StdInputField extends StatelessWidget {
  const StdInputField({
    super.key,
    required this.hintText,
    required this.width,
    required this.hideText,
    required this.controller,
    this.icon,
    required this.onSubmitFunction,
  });

  final String hintText;
  final bool hideText;
  final double width;
  final TextEditingController controller;
  final IconData? icon;
  final Function(String) onSubmitFunction;

  @override
  Widget build(BuildContext context) {
    bool hasIcon = icon != null;
    return SizedBox(
      width: width,
      child: TextField(
        controller: controller,
        cursorColor: context.colorScheme.primary,
        decoration: InputDecoration(
          contentPadding: EdgeInsets.only(
            top: 16.0,
            bottom: 16.0,
            right: 8.0,
            left: hasIcon ? 0 : 8,
          ),
          hintText: hintText,
          prefixIcon: hasIcon ? Icon(icon) : null,
          isCollapsed:
              true, // fixing hintText not being centered correctly when
prefixIcon exists
          border: const OutlineInputBorder(
            borderRadius: BorderRadius.all(
              Radius.circular(16),
            ),
        ),
    
```

```

        ),
        ),
        obscureText: hideText,
        textInputAction: TextInputAction.next,
        onEditingComplete: () =>
            {SystemChrome.setEnabledSystemUIMode(SystemUiMode.edgeToEdge)},
        textAlignVertical: TextAlignVertical.center,
        autocorrect: false,
        style: context.textTheme.labelLarge,
        onSubmitted: onSubmitFunction,
    ),
),
);
}
}

```

7.1.6 Homepage

Die Homepage bzw. Hauptseite ist das Herzstück der Visualisierungsapp und dient als eine Art Grundstück für alle Hauptfunktionen dieser Applikation. Die meistens der Funktionen bezüglich der Flugaufzeichnung und Visualisierung sind jedoch erst nach Start der Ground-Station startbar. Auf dieser Seite wartet die App ständig auf den Start der Aufzeichnung bzw. auf die Änderung der Boolean Variable in der Realtime Datenbank (siehe: [Kapitel 7.2.4.3](#)). Ansonsten ist lediglich die Sidebar mit seinen Optionen nutzbar.

7.1.6.1 Aufzeichnung starten und stoppen

Um die Aufzeichnung eines Fluges zu starten, gibt es im unteren Abschnitt der Homepage einen Button mit, der die Aufnahme initiiert werden kann. Dieser ist jedoch nur im Zustand einer aktiven Datenübertragung von der Ground-Station aus sichtbar. Findet gerade eine Aufnahme statt so dient derselbe Button dazu, um die Aufnahme manuell zu stoppen.

7.1.6.2 Serverdatendialog

Der Serverdatendialog ist ein Bildschirm, der auf der Homepage aufpoppt, wenn die Aufzeichnung eines Fluges gestartet wurde. Dieser dient dem Zweck die benötigten IP-Adresse des Servers (Ground-Station) und die Ports (MQTT und RTMP) für die Aufnahme zu definieren. Somit ist die App auch deutlich flexibler beim Empfang der Daten, da theoretisch jeder Server und Port als Quelle definiert werden kann, solange die Daten richtig formatiert sind bzw. wie im Fall von RTMP das richtige Protokoll verwenden.

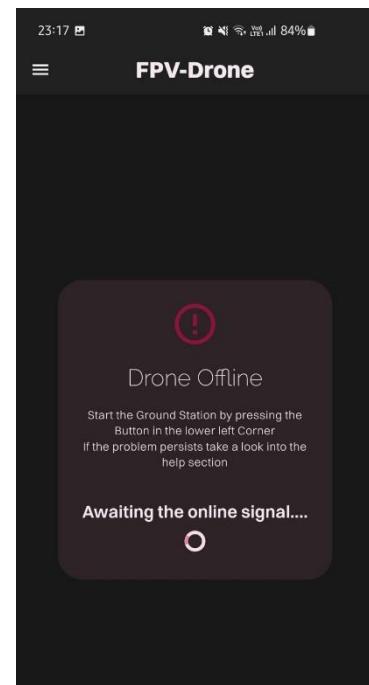


Abbildung 216: Homepage im Offline-Status

Die Eingabe kann mit dem Button unter den Eingabefeldern bestätigt werden, wobei die eingegebenen Daten an die spezifischen Widgets übergeben werden, um die Aufzeichnung nun endlich zu starten.

7.1.6.3 Aufzeichnung stoppen

Nachdem eine laufende Aufzeichnung über den zugehörigen Button auf der Homepage gestoppt wurde, öffnet sich ein Bildschirm auf der Homepage mit dem der Flug personalisiert und schlussendlich für späteres Anschauen gespeichert werden kann. Unter dieser Personalisierung fallen bisher die Eingabe eines Titels und die Auswahl eines Icons die den Flug repräsentiert. Obwohl der Titel vom User geändert werden kann, wird dieser mit dem Datum und der Uhrzeit des Fluges initialisiert.

Mit dem Drücken auf den „Save Flight“-Button werden die gesammelten Flugdaten automatisch über den „Flight Records“-Punkt im Side Menü aufrufbar. Zusätzlich gibt es noch die Option die gesammelten Flugdaten zu verwerfen, indem man auf den Button links oben des Bildschirmes, direkt neben dem „Discard“ tippt.

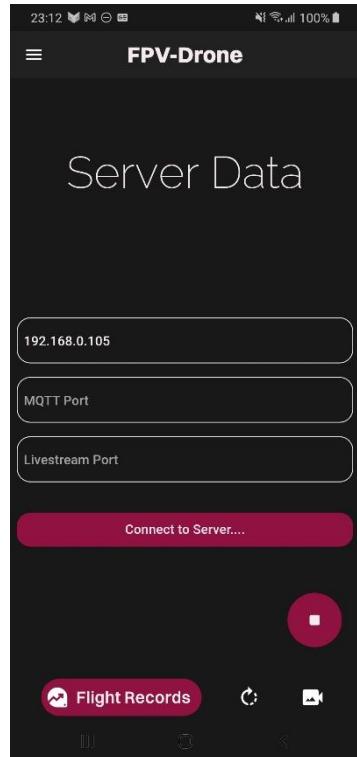


Abbildung 217: Serverdatendialog mit eingegebener IP-Adresse

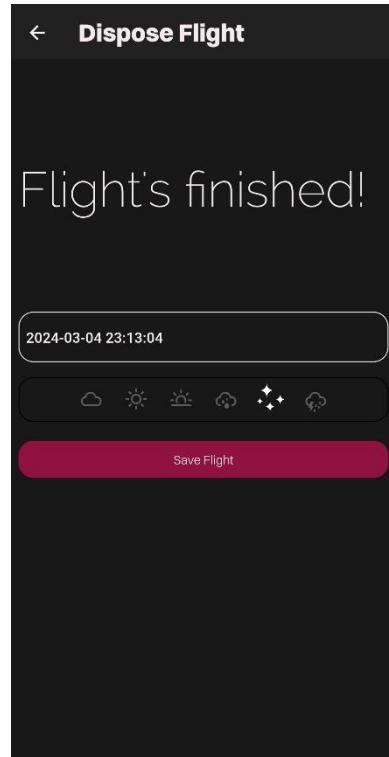


Abbildung 218: Bildschirm nach beendetem Flug

7.1.6.4 Bottom Navigation Bar / GNav-Bar

Um zwischen den verschiedenen Optionen auf der Homepage zu navigieren, befindet sich am unteren Rand des Bildschirmes eine sogenannte „*Bottom Navigation Bar*“ oder auch Navigationsleiste. Auf dieser sind alle Hauptfunktionen der Flugaufzeichnung aufgelistet und können durch einen simplen Knopfdruck ausgewählt werden.

Um das Design der Navigationsleiste zu realisieren, entschied ich mich gegen eine standardmäßige Version von Flutter selbst, aber für ein Package, welches den Stil dieser Leiste von modernen Google Apps nachahmt. Dieses Package hat den Namen „*google_nav_bar*“ und ganz einfach über pub.dev installierbar.

```
Dateiname: Homepage.dart; Ausschnitt: „bottomNavigationBar“ des Scaffolds
import 'package:google_nav_bar/google_nav_bar.dart';
.....
bottomNavigationBar: StreamBuilder<dynamic>(
    stream: rtdbService.listenToValue("server_selected"),
    builder: (context, snapshot) {
        bool visibility = ((snapshot.data?.snapshot.value ?? false) as
bool);
        return Visibility(
            visible: visibility,
            child: GNav(
                // Style
                style: GnavStyle.google,
                iconSize: 28,
                tabBackgroundColor: context.colorScheme.secondary,
                tabMargin: const EdgeInsets.symmetric(
                    vertical:
                        5), // setting the space between buttons and end of bar
                padding: const EdgeInsets.symmetric(
                    horizontal: 5,
                    vertical:
                        8), // setting thickness of button and bar in general
                mainAxisAlignment: MainAxisAlignment.spaceEvenly,
                gap: 8,
                // Function
                selectedIndex: currentPageIdx,
                tabs: [
                    GButton(
                        icon: Icons.data_exploration,
                        text: "Flight Records",
                        iconColor: context.colorScheme.onBackground,
                        iconActiveColor: context.colorScheme.background,
                    ),
                    GButton(
                        icon: Icons.rotate_right_outlined,
                        text: "3D-Space",
                        iconColor: context.colorScheme.onBackground,
                    ),
                ],
            ),
        );
    },
);
```

```

        iconActiveColor: context.colorScheme.onPrimary,
    ),
    GButton(
        icon: Icons.video_camera_back,
        text: "Live View",
        iconColor: context.colorScheme.onBackground,
        iconActiveColor: context.colorScheme.background,
    ),
    ],
    ,

    onTabChange: (value) {
        setState(() {
            currentPageIdx = value;
        });
    },
),
);
},
),
),
),
),
),
.... .

```

Im Code ist abgesehen von der eigentlichen Navigationsleiste (**GNav**) noch ein „*StreamBuilder*“ (siehe: [Kapitel 7.1.1.5](#)) und ein „*Visibility*“-Widget (Widget um Teile der App mit einer Boolean Variable zu zeigen oder zu verstecken) zu sehen, die diese umwickeln. Diese haben lediglich den Sinn die Navigationsleiste bei noch ausstehender Auswahl eines Servers zu verstecken.

Die Navigationsleiste selbst wird mit dem Keyword „GNav“ definiert und grundlegendsten Einstellungen lauten wie folgt:

- **style:** Kann entweder mit GnavStyle.google oder GnavStyle.oldSchool gesetzt werden und definiert den grundlegenden Style der Navigationsleiste
- **selectedIndex:** Definiert mit welcher ausgewählten Option die Navigationsleiste gestartet wird
- **tabs:** Dieser Punkt nimmt eine Liste an „GButton“-Widgets, welche die eigentlich sichtbaren Buttons darstellen. Diese Buttons können mit einem Icon und Text personalisiert werden
- **onTabChange:** Sobald der User eine Option auswählt, wird die hier definierte Funktion aufgerufen. In meinem Fall wird lediglich eine Variable mit dem ausgewählten Index der Option gesetzt

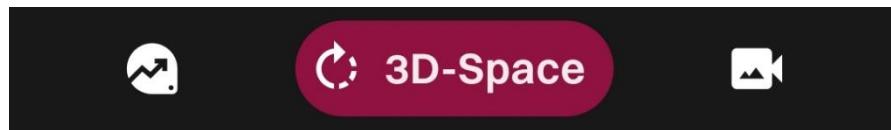


Abbildung 219: GNav / Bottom Navigation Bar auf Homepage

7.1.6.5 Flugdatenvisualisierung

Allgemeines

Eine der Hauptgründe, weshalb diese App eigentlich entwickelt wurde, ist die Visualisierung der Flugdaten. Hierfür benutzen wir mit MQTT (Message Queuing Telemetry Transport) ein weitverbreitetes Protokoll, um Sensordaten von der Ground-Station zur App zu transportieren. Dieses erlaubt es einem Daten einfache in Form von Text über das Netzwerk an die App mit nur geringer Latenz zu speisen.

Der primäre Vorteil an MQTT liegt am Publish/Subscribe-Prinzip. Darunter versteht man, dass eine Quelle Daten sendet („publish“) und diese Daten dann von mehreren Empfängern („subscriben“) empfangen werden können. Durch dieses explizite Abonnementverfahren von Quelle zu Empfänger ist die Datenrate und somit die Latenz ziemlich niedrig.

Eine simple Verbindung über Sockets ist aufgrund der Möglichkeit von mehreren Empfängern bzw. parallelen Nutzern der Visualisierungsapp nicht in Frage gekommen.

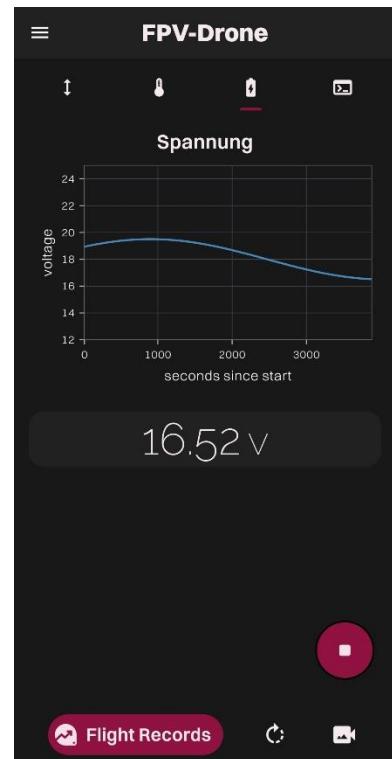


Abbildung 220: Empfangene Spannungsdaten via MQTT

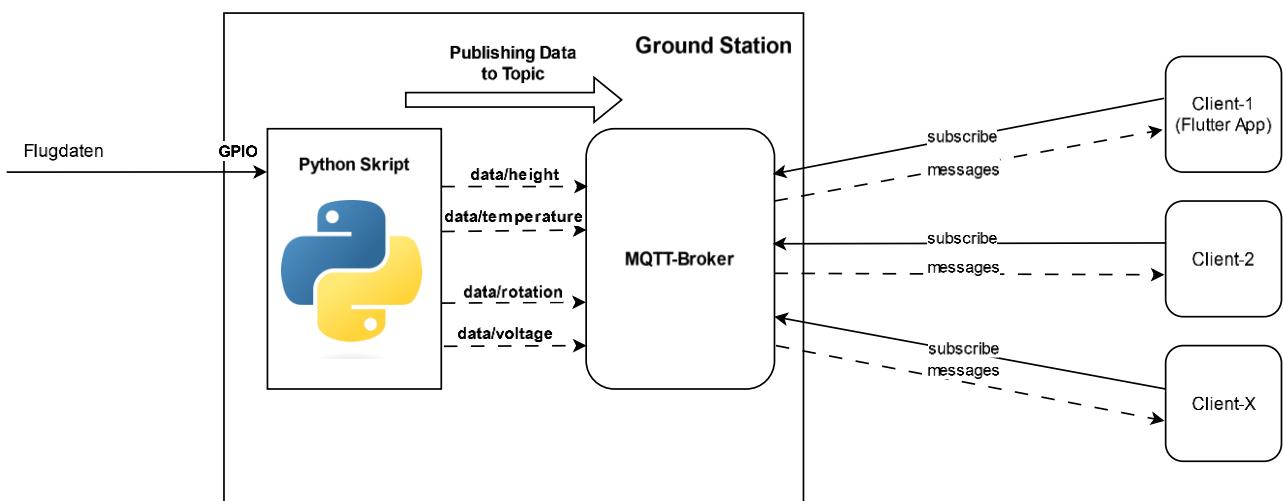


Abbildung 221: MQTT-Broker zu Client Blockschaltbild [PYLO]

MQTT-Manager

Um eine zentralisierte Klasse zu haben, welche alle notwendigen Funktionen von in Verbindung mit MQTT beeinhaltet, schrieb ich einen MQTT-Manager. Dieser beinhaltet Funktionen, wie den Verbindungsaufbau und die Verbindung zu verschiedenen Datenstreams (Topics).

```
Dateiname: mqtt_manager.dart
import 'package:drone_2_0/data/providers/logging_provider.dart';
import 'package:mqtt_client/mqtt_client.dart';
import 'package:mqtt_client/mqtt_server_client.dart';
import 'dart:async';

class MQTTManager {
    late MqttServerClient _client;
    late final StreamController<Map<String, String>> _messageStreamController;
    Stream<Map<String, String>> get messageStream =>
        _messageStreamController.stream;

    late final String _serverIp;
    late final int _port;
    late final String _clientIdentifier;

    MQTTManager(this._serverIp, this._port, this._clientIdentifier) {
        _messageStreamController =
            StreamController<Map<String, String>>.broadcast();
    }

    Future<bool> connect() async {
        _client = MqttServerClient(_serverIp, _clientIdentifier,
            maxConnectionAttempts: 3);
        _client.port = _port;
        _client.logging(on: false);

        _client.onDisconnected = onDisconnected;
        _client.onConnected = onConnected;

        final MqttConnectMessage connMess = MqttConnectMessage()
            .withClientIdentifier(_clientIdentifier)
            .startClean();

        _client.connectionMessage = connMess;

        try {
            await _client.connect();

            // listening to received messages from the connected broker
            _client.updates?.listen((List<MqttReceivedMessage<MqttMessage>> event) {
                for (MqttReceivedMessage<MqttMessage> message in event) {
                    if (message.payload is MqttPublishMessage) {
                        final MqttPublishMessage publishMessage =
                            message.payload as MqttPublishMessage;
                        // convert databytes to string
                        final String payload = MqttPublishPayload.bytesToStringAsString(
                            publishMessage.payload.message);
                        final String topic = message.topic;
                    }
                }
            });
        } catch (e) {
            print('Error connecting to MQTT broker: $e');
        }
    }

    void onConnected() {
        print('Connected to MQTT broker');
    }

    void onDisconnected() {
        print('Disconnected from MQTT broker');
    }
}
```

```

        // Add the message to the stream
        _messageStreamController.add({topic: payload});
    }
}

// listen to connection errors
return true;
} catch (e) {
    // Exception during connection or listening attempt
    Logging.error('MQTT EXCEPTION: $e');
}
return false;
}

void subscribeToTopic(String topic) {
    if (_client.connectionStatus!.state == MqttConnectionState.connected) {
        _client.subscribe(topic, MqttQos.atMostOnce);
    }
}

void publishMessage(String topic, String message) {
    final MqttClientPayloadBuilder builder = MqttClientPayloadBuilder();
    builder.addString(message);
    _client.publishMessage(topic, MqttQos.atLeastOnce, builder.payload());
}

void onDisconnected() {
    Logging.info("MQTT: CLIENT DISCONNECTED");
}

void onConnected() {
    Logging.info("MQTT: CLIENT CONNECTED TO BROKER -> $_serverIp:$_port");
}

void disconnect() {
    _client.disconnect();
    _messageStreamController.close();
}
}

```

Diese Klasse setzt sich aus den folgenden Komponenten zusammen:

- **Konstruktor (MQTTManager):** Der Konstruktor wird aufgerufen, sobald ein Objekt dieser Klasse erstellt wird und nimmt als Parameter den Port, die IP-Adresse und eine Client-Identifikation (Ein String, um mehrere Clients zu unterscheiden). Zusätzlich wird der „_messageStreamController“ mit einem StreamController initialisiert, der den Datentyp <Map<String, String>> als „broadcast“ liefert. Die Definition als broadcast dient lediglich dazu, dass die Nachrichten vom StreamController von mehreren Teilen im Code abgegriffen werden können.

- **`connect()`-Methode:** Initialisiert die vordefinierte `_client`-Variable mit der `MqttServerClient`-Klasse des Plugins und initialisiert alle wichtigen Werte. Bei erfolgreichen Verbindungsaufbau hört die Klasse auch auf alle Änderungen auf verbundenden Topics, konvertiert diese zu einem String und fügt jede neue Nachricht zum „`_messageStreamController`“ hinzu
- „**`_messageStreamController`**“: Der `_messageStreamController` ist eine Variable vom Typen der `StreamController`-Klasse von Flutter die als eine Art Schnittstelle zwischen der Managerklasse und anderem Code fungiert, der lediglich die empfangenen Daten abgreifen möchte. Sie dient dazu die empfangenen Nachrichten in der Form eines Streams anzubieten. Dieser Stream kann durch die getter-Funktion `messageStream` von außen externen Files abgegriffen werden für meist die Nutzung innerhalb eines `StreamControllers`.
- **`subscribeToTopic()`-Methode:** Verbindet bzw. Abonniert den Client mit einem gegebenen Topic, solange dieser noch nicht verbunden ist.
- **`onConnected()`-Methode und `onDisconnected ()`-Methode:** Simple Klassen die nur eine Nachricht ans Terminal und das Logfile schreiben sobald der Client sich verbunden hat oder die Verbindung abgebrochen ist
- **`disconnect()`-Funktion:** Schließt die Verbindung des Clients und den `StreamController` ordnungsgemäß

MQTT-Datenstreams

Die 3 Datenstreams (Höhe, Temperatur und Batteriespannung) die empfangen werden, treffen alle über einem Stream ein. Die Initialisierung der MQTT-Klasse und die Verbindung bzw. das Abonnieren zu den verschiedenen Topics findet in der „`initDataConnection`“-Funktion statt.

```
Dateiname: flight_records.dart; Funktion: initDataConnection
Future<bool> initDataConnection() async {
  chartData = Map.from(_emptyChartData); // reset data on init
  bool connection = await mqttManager.connect();

  // Subscribe to topics
  if (connection) {
    mqttManager.subscribeToTopic("data/height");
    mqttManager.subscribeToTopic("data/temperature");
    mqttManager.subscribeToTopic("data/voltage");
    mqttManager.subscribeToTopic("data/messages");
    Logging.info("Subscribed to Flight Records MQTT Topics");
  }

  return connection;
}
```

Um die Daten in Echtzeit empfangen zu können wird ein sogenannter StreamBuilder benutzt, wobei die „Stream“-Klasse definiert werden kann, die kontinuierlichen Daten empfängt und dem Widget bzw. den Kindern des „StreamBuilder“ als ein „snapshot“ darbietet:

```
Dateiname: flight_records.dart; Ausschnitt: StreamBuilder + Datenübergabe an Widgets
StreamBuilder(
  stream: _dataStreams(),
  builder: (context, snapshot) {
    if (snapshot.hasError) {
      return const Expanded(
        child: ConnectionError(),
      );
    }

    if (!snapshot.hasData) {
      return const Expanded(
        child: AwaitingConnection(),
      );
    }

    // Extract data from the snapshot
    snapData = snapshot.data;

    if (flightStartTimestamp == -1) {
      flightStartTimestamp = snapData[2];
    }
    timeAxisValue = (snapData[2] - flightStartTimestamp);

    switch (snapData[MQTTData.identifier.index]) {
      case "V":
        chartData["voltage"] =
          setData("voltage", timeAxisValue);
      case "H":
        setData("height", timeAxisValue);
      case "T":
        setData("temperature", timeAxisValue);
    }

    Logging.info(chartData["height"]!.length.toString());
    return Expanded(
      child: Padding(
        padding: const EdgeInsets.all(8.0),
        child: TabBarView(
          children: [
            StreamDisplay(
              title: "Height",
              xibName: "meters",
              yibName: "seconds since start",
            ),
          ],
        ),
      ),
    );
  },
);
```



```
Dateiname: flight_records.dart; Funktion _dataStreams
Stream<dynamic> _dataStreams() {
  try {
    final mqttStream = mqttManager.messageStream;
    final combinedStream = StreamGroup.merge([mqttStream]);

    return combinedStream.map((event) {
      List<String> data = event.values.first.split(" ");
      List<dynamic> convertedData = [data[MQTTData.identifier.index]];
      convertedData.add(double.parse(data[MQTTData.value.index]));
      convertedData.add(int.parse(data[MQTTData.timestamp.index]));
      return convertedData;
      // ignore: unnecessary_null_comparison
    }).where((data) => data != null); // Filter out null values
  } on SocketException catch (e) {
    Logging.error(e.toString());
    return const Stream.empty();
  }
}
```

Hinweis: Die Kombinierung bzw. StreamGroup.merge ist noch ein Artefakt aus einer älteren Version in den mehrere Streams gebündelt werden mussten und hat in dieser Konstellation keinen Nutzen

Wie in der Empfangsfunktion zu sehen ist wird ein einziger Stream benutzt, um die Daten an den StreamBuilder zu übergeben, wo eine empfangene Nachricht nach seinem Thema zugeordnet wird. Dies funktioniert aufgrund von folgender Struktur einer Nachricht:

```
<Identifier> <Datenwert> <Timestamp / Zeitstempel>
```

- **Identifier:** Ist ein einziger Buchstabe, der signalisiert um welche Daten es sich handelt
 - **H = Höhe**
 - **V = Spannung**
 - **T = Temperatur**
- **Datenwert:** Der Eigentliche Messwert
- **Timestamp / Zeitstempel:** Ein Zeitstempel, der benötigt wird, um festzulegen wann ein Datenwert gemessen bzw. gesendet wurde

In der „dataStreams“-Funktion wird dieser String lediglich in seinen einzelnen Bestandteil aufgeteilt und in die korrekten Datentypen umgewandelt, da die ursprüngliche Nachricht ein reiner Text ist. Diese konvertierten Daten werden wieder als Stream an den StreamBuilder durch das „return“-Keyword übergeben.

Im StreamBuilder wird schlussendlich abhängig vom Identifier das jeweilige Datenarray in der „setData“-Funktion aktualisiert und an das „StreamDisplay“-Widget übergeben, wobei diese als Diagramm und als simpler Text visualisiert werden.

```
Dateiname: flight_records.dart; Funktion: setData
setData(String key, int xValue) {
    chartData[key]?.add(ChartData(xValue, snapData[MQTTData.value.index]));
    widget.flightData.addDataPoint(key, snapData[MQTTData.value.index], xValue);
    lastMeasurement = snapData[1];
    limitDataPoints(key);
}
```

Hinweis: MQTTData in den oben gezeigten Codeausschnitten ist eine eigens definierte Enumeration (Enum), die den Index des Identifiers, Datenwertes und Zeitstempels im Array definiert

Begrenzung der Daten

Damit das Diagramm nicht im Laufe einer Aufzeichnung mit Daten überfüllt wird, habe ich mich entschieden das Datenarray für die Diagramme auf ein Maximum von 500 Datenwerten zu begrenzen. Dies geschieht innerhalb der „setData“-Funktion mit der „limitDataPoints“-Funktion, welche das Array auf die letzten 500 Datenpunkte begrenzt:

```
Dateiname: flight_records.dart; Funktion: limitDataPoints
void limitDataPoints(String key) {
    // limitiert eine Flugdaten-Map auf 500 Datenwerte -> Scrollende Diagramme
    int maxDataPoints = 500;
    chartData[key] = (chartData[key]!.length > maxDataPoints
        ? chartData[key]!.sublist(chartData[key]!.length - maxDataPoints)
        : chartData[key])!;
}
```

Echtzeitdaten + Diagramme

Die eigentliche Visualisierung der Daten funktioniert einerseits durch die Darstellung des letzten Messwertes als Text, aber viel wichtiger auch durch die Diagramme des Syncfusion-Plugins (siehe: [Kapitel 7.1.2.2](#)). Diese Visualisierung findet in einem wiederverwendbaren Widget namens „StreamDisplay“ statt. Für das XY-Diagramm wird ein „CartesianChart“ von der Syncfusion Library benutzt um den Verlauf der letzten 500 Datenwerte (siehe: [Kapitel 7.1.6.5](#)) anzuzeigen.

```
Dateiname: stream_displayer.dart; Ausschnitt XY-Diagramm / Cartesian Chart Definition
SfCartesianChart(
    // Chart title
    title: ChartTitle(text: title, textStyle: context.textTheme.bodyMedium),

    // Style
    enableAxisAnimation: false,
    primaryXAxis: NumericAxis(
        name: xibName,
        title: AxisTitle(text: xibName),
    ),
    primaryYAxis: NumericAxis(
```

```
        minimum: minY,
        maximum: maxY,
        name: yAxisName,
        title: AxisTitle(text: yAxisName),
    ),
    // Data
    series: <CartesianSeries>[
        LineSeries<ChartData, num>(
            dataSource: dataArray,
            xValueMapper: (ChartData time, _) => time.x,
            yValueMapper: (ChartData yValue, _) => yValue.y,
            markerSettings: const MarkerSettings(
                isVisible: false,
            ),
            ),
        ],
    ],
),
```

Unter dem Diagramm wird zusätzlich noch der zuletzt gemessene Datenwert als einfacher Text dargestellt:

```
Dateiname: stream_displayer.dart; Ausschnitt: Darstellung des letzten  
Messwertes
```

```
Container(  
    width: double.infinity,  
    decoration: BoxDecoration(  
        color: Colors.grey.shade900,  
        shape: BoxShape.rectangle,  
        borderRadius: BorderRadius.circular(16)),  
    child: Center(  
        child: RichText(  
            text: TextSpan(  
                style: context.textTheme.headlineLarge,  
                children: <TextSpan>[  
                    TextSpan(  
                        text: dataArray.isNotEmpty  
                            ? dataArray[dataArray.length - 1].y.toStringAsFixed(2)  
                            : "-"),  
                    TextSpan(  
                        text: " $unit",  
                        style: context.textTheme.headlineMedium,  
                    ),  
                ],  
            ),  
        ),  
    ),  
,),
```

Der Datenwert wird fix mit 2 Nachkommastellen durch die Nutzung von der „`toStringAsFixed`“-Funktion begrenzt. Der Grund, weshalb kein normales Text-Widget, sondern ein RichText-Widget verwendet wird, ist aufgrund der Einheit, die nach dem eigentlichen Datenwert angezeigt wird. Diese kann in einigen Fällen Hochzahlen oder andere besondere Unicode-Zeichen beinhalten, die nur in dieser Form dargestellt werden können.

Speicherung der Daten

Neben der Visualisierung der Daten in Echtzeit werden empfangenen Daten auch in einer „`FlightData`“-Klasse abgespeichert, welche die eigentlichen Flugdaten, aber auch generelle Parameter eines Fluges wie Titel, Wetter, Startzeit oder Endzeit abspeichert. Zusätzlich bietet diese Klasse eine Vielzahl an Methoden, mit denen diese Daten am Ende des Fluges ohne Probleme abgespeichert werden können.

Datencache

Da die Daten der Flüge einen etwas größeren Ausmaß haben können, entschied ich mich dazu die Daten eines Fluges neben der eigentlichen Abspeicherung auf der Datenbank auch lokal abzuspeichern.

Auf der Datenbank wird ein Zeitstempel gespeichert, wann diese Flugdaten zuletzt aktualisiert wurden. Lokal wird auch ein File erstellt, indem das Alter der Flüge bzw. ein Zeitstempel abgespeichert wird.

```
Dateiname: finished_flight.dart; Funktion: _saveFlightProperties
```

```
void _saveFlightProperties(BuildContext context) {
    String newTitle = textFieldController.text;

    // update title on db
    UserProfileService().updateFlightDataProperty(
        context.read<AuthenticationProvider>().userId,
        endTimeStamp,
        "title",
        newTitle);

    // update title in local storage
    Provider.of<DataCache>(context, listen: false)
        .updateFlightProperty(endTimeStamp, "title", newTitle);

    // update weather icon on db
    UserProfileService().updateFlightDataProperty(
        context.read<AuthenticationProvider>().userId,
        endTimeStamp,
        "weather",
        _weatherIcon);

    // update weather in local storage
    Provider.of<DataCache>(context, listen: false)
        .updateFlightProperty(endTimeStamp, "weather", _weatherIcon);
```

```
// hide dialogue
Navigator.pop(context);
}
```

Bei Endung eines Fluges werden die neuen Daten auf die Datenbank hochgeladen und im lokale Cachefile gesetzt. Neben den Daten werden auch die Zeitstempel aktualisiert, da die Daten nun als aktualisiert und synchronisiert gelten.

Im Falle das eine Aufzeichnung auf einem anderen Gerät abgespeichert wurde, würden diese Zeitstempel nicht mehr übereinstimmen bzw. der Zeitstempel auf der Datenbank wäre aktueller und die Daten müssen auf dem jeweils anderem Gerät neu heruntergeladen werden.

Dies hat zur Folge das die Daten auf der Datenbank nur bei einem Wechsel des Geräts heruntergeladen werden müssen, aber trotzdem auf jedem Gerät eines spezifischen Users synchronisiert sind.

7.1.6.6 3D-Model-Viewer

Der 3D-Model-Viewer ist eine der Hauptfunktionen der Visualisierungsapp und hat den Zweck die Lage der Drohne in Echtzeit anhand eines 3D-Modells darzustellen.

Flutter Cube

Flutter Cube („*flutter_cube*“) ist ein Plugin für Flutter, welches die Einbindung von 3D-Modellen einfach macht. Die Installation findet wieder über pub.dev statt:

```
flutter pub add flutter_cube
```

Um die Funktionen dieses Plugins zu benutzen, muss man ein 3D-Modellfile als Asset definieren und folgendermaßen im Code definieren:

```
.....
child: Cube(
    onSceneCreated: (Scene scene) {
        scene.world.add(Object(fileName: 'assetPfad/filename'));
    },
),
....
```

Mit diesen gezeigten Funktionen lassen sich alle möglichen 3D-Modelle in Flutter anzeigen. Um weitere Einstellungen zu tätigen wie Lichteinstrahlung oder Position des Modells zu definieren, geht das innerhalb der „onSceneCreated“-Funktion, bevor die Modelle zu einer Szene hinzugefügt werden.

3D-Model Asset hinzufügen

Grundsätzlich funktioniert die Nutzung eines 3D-Modells so wie jedes andere Asset in Flutter, durch die Definition des Pfades im „`pubspec.yaml`“-File (siehe: [Kapitel 7.1.4.6](#)). Im Falle eines „.obj“-Files (Wavefront OBJ), welches ein typisches Dateiformat für 3D-Modelle ist, müssen zwei Files mit demselben Namen und Pfad definiert werden. Einerseits das „.obj“-File, dass Informationen über die Geometrie beinhaltet, und ein „.mtl“-File (Wavefront Material Template Library), welches die Farbinformationen hält. Wurden diese beiden Files inkludiert, so benutzt Flutter Cube automatisch das jeweilige „.mtl“-File für die Farben.

Wird wie in meinem Fall nur das „.obj“-File zur Verfügung gestellt, so kann man in z.B. Blender, die Farben der jeweiligen Flächen definieren und schließlich als „.obj“-File exportieren, wobei das zugehörige „.mtl“-File automatisch mitexportiert wird.

MQTT-Datenstream

Um das 3D-Modell in Echtzeit mit den wirklichen Rotationsdaten (Pitch, Roll, Yaw) der Drohne zu rendern wird auch in diesem Teil der App ein MQTT-Datenstream verwendet (siehe: [Kapitel 7.1.6.5](#)). Anders als die anderen Flugdaten werden diese 3 Daten in einem einzigen zusammengesetzten Paket mit folgender Formatierung gesendet:

```
<PITCH> <ROLL> <YAW>
```

Der Empfang der Daten findet ebenfalls, wie bei den Flugdaten über der eigens kreierten „`MQTTManager`“-Klasse statt (siehe: [Kapitel 7.1.6.5](#)).

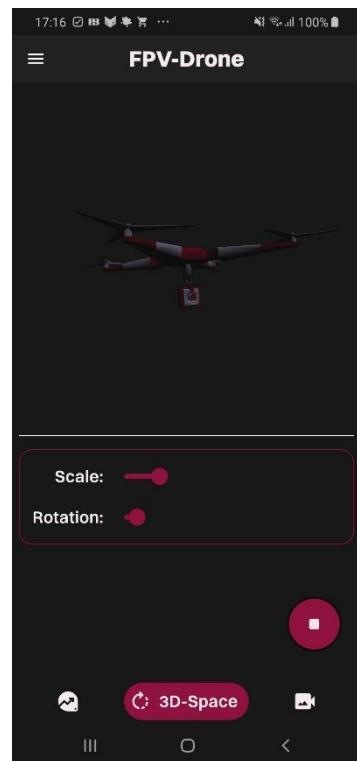


Abbildung 222: 3D-Modell auf Homepage

```
Dateiname: drone_model_viewer.dart; Funktion: _rotationManagerInit
Future<bool> _rotationManagerInit() async {
    bool connection = await mqttManager.connect();

    if (connection) {
        mqttManager.subscribeToTopic("data/rotation");
        Logging.info("Subscribed to Rotation Topic");
        mqttManager.messageStream.listen((event) {
            List<double> rotationData = [];
            event.values.first.split(" ").forEach((element) {
                // converting split list of strings to the rotation values as
                floating point
                rotationData.add(double.parse(element)));
            });
            Logging.info(rotationData);
            _cube?.rotation.z = rotationData[ModelAxis.roll.index];
            _cube?.rotation.y = rotationData[ModelAxis.yaw.index] + yawOffset;
            _cube?.rotation.x = rotationData[ModelAxis.pitch.index] + pitchOffset;
        });
    }
}
```

```

        updateModel();
    });
}
return connection;
}

```

In der oben gezeigten Funktion ist eine Kombination der Init-Funktion des MQTT-Managers und der Funktion zum Empfang der Daten zu sehen. Bei der Initialisierung wird lediglich die Verbindung zum Topic, wo die Rotationsdaten übertragen werden, aufgebaut (*data/rotation*). Bei erfolgreicher Verbindung wird auf die Daten vom MQTT-Broker bzw. von der Groundstation gewartet. Diese Daten werden schließlich dekodiert in die 3 verschiedenen Rotation und den jeweiligen Rotationen des Modells gesetzt.

Dateiname: drone_model_viewer.dart; Funktion: updateModel
<pre> void updateModel() { _cube?.updateTransform(); _model.update(); } </pre>

Anders als bei herkömmlichen Widgets reicht nicht das Setzen der Variablen im Objekt des 3D-Modells innerhalb der „*setState*“-Funktion, sondern es müssen die im Code gezeigten Funktionen aufgerufen werden, sodass das Modell neu gerendert wird und die Änderungen auch sichtbar werden.

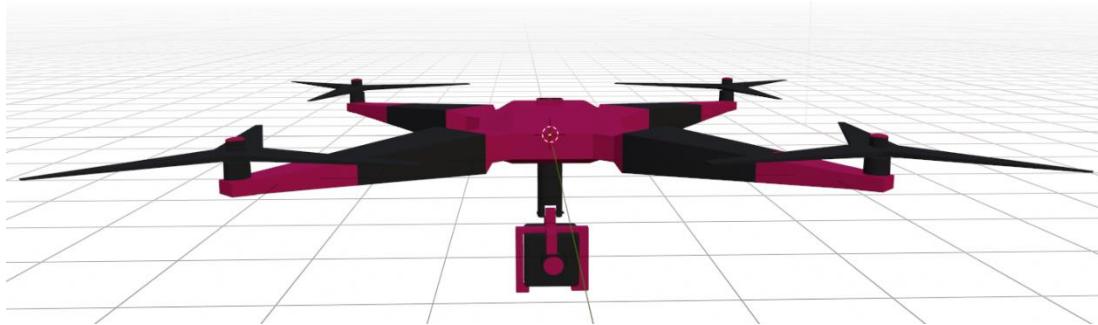


Abbildung 223: Eingefärbtes Modell der Drohne in Blender

3D-Dronenmodell

Das 3D-Modell selber ist ein frei benutzbare 3D-Modell einer normalen Drohne und wird im Code als Kind eines „*SizedBox*“-Widgets mit dem „*Cube*“-Widget definiert, um die Größe in einem fixen Bereich zu halten. Die eigentliche Kreation des Modells findet jedoch erst in der „*_onSceneCreated*“-Funktion definiert und schlussendlich in die sogenannte Szene platziert.

Dateiname: `drone_model_viewer.dart`; Ausschnitt: Erstellung des 3D-Modell im State

```
SizedBox(
  width: MediaQuery.sizeOf(context).width,
  height: modelWidgetHeight,
  child: Cube(
    interactive:
      false, // only interaction through MQTT -> Pitch, Roll, Yaw
  )
  onSceneCreated: _onSceneCreated,
),
```

Extrabedienungen

Unter dem eigentlichen 3D-Modell befinden sich einige Slider mit denen der User die Darstellung des Modells auf ein paar Weisen beeinflussen kann:

- **Scale / Skalierung:** Ändert den FOV-Wert der Kamera, um die Entfernung vom Modell zu simulieren

Dateiname: `drone_model_viewer.dart`; Ausschnitt: Scale Slider

```
Slider(
  label: 'Scale',
  max: 150,
  min: 50,
  value: fov,
  onChanged: (value) {
    setState(() {
      fov = value;
      _model.camera.fov = fov;
      updateModel();
    });
  },
),
```

- **Yaw-Offset:** Der Yaw-Offset rotiert die Drohne unabhängig von den empfangenen Yaw-Daten um die eigene Achse des Modells. Somit lässt sich beeinflussen, ob die Drohne zu einem hin- oder wegschaut

Dateiname: `drone_model_viewer.dart`; Ausschnitt: Yaw-Offset Slider

```
Slider(
  label: 'Yaw-Offset',
  max: 180,
```

```
    min: -180,  
    value: yawOffset,  
    onChanged: (value) {  
        setState(() {  
            yawOffset = value;  
            _cube?.rotation.y = yawOffset;  
            updateModel();  
        });  
    },  
),
```

7.1.6.7 Live-View

Der Teil des Hauptmenüs, indem man die Bild der Drohne in „Echtzeit“ mitverfolgen kann, oder auch „Live View“, ist die 3te und letzte Hauptfunktion der Visualisierungsapp.

VLC-Plugin

Um den gesendeten RTMP-Stream vom Server empfangen zu können, benutze ich ein Plugin, das bestimmte Funktionen des VLC-Players in Flutter verfügbar macht, darunter auch Netzwerkstreams. Dieses Plugin hat den Namen „`flutter_vlc_player`“ und kann über `pub.dev` installiert werden:

```
flutter pub add flutter_vlc_player
```

Darstellung eines Videostreams

Um jegliche Form eines Videos über das VLC-Plugin anzeigen zu können sind 2 Hauptkomponenten im Code verlangt:

- **VlcPlayer-Widget:** dient als Container innerhalb der UI und für die Allgemeinen Einstellungen. Beinhaltet den *VlcPlayerController*, einen Platzhalter für die Anzeige bei Problemen und definiert das Seitenverhältnis des Videos:

```
Dateiname: live_view.dart; Ausschnitt: VlcPlayer-Widget
.....
@Override
void initState() {
    super.initState();
    Logging.debug("LiveView Initializing");
    _videoPlayerController = VlcPlayerController.network(
        autoInitialize: true,
        "rtmp://${widget.ipAdress}:${widget.port}/${widget.streamName}",
        hwAcc: HwAcc.auto,
        autoPlay: true,
        options: _controllerOptions,
    );
.....
}
```

- **VlcPlayerController:** Der VlcPlayerController ist wie alle anderen Controller eine Klasse die für ein vordefiniertes Widget, in diesem Fall das „VlcPlayer“-Widget, existieren muss damit es möglich ist mit diesem Widget über Code zu interagieren. Zusätzlich wird dieser Controller in mehrere Typen unterteilt, je nachdem welche Form von Video man abspielen möchte:
 - VlcPlayerController.network (Netzwerk-URLs die Videos übertragen)
 - VlcPlayerController.file (File auf lokalem Gerät)
 - VlcPlayerController.asset (Asset aus den App-Dateien)
 Bei den eigentlichen Parametern muss eine Adresse bzw. der Pfad des Videos definiert werden. Die Optionen, Autoinitialisierung, Hardware-Akzeleration und Autoplay sind alle optional wurden jedoch in meinem Falle gesetzt:

```
Dateiname: live_view.dart; Ausschnitt: VLC-Player Controller
	videoPlayerController = VlcPlayerController.network(
		autoInitialize: true,
		"rtmp://${widget.ipAdress}:${widget.port}/${widget.streamName}",
		hwAcc: HwAcc.auto,
		autoPlay: true,
		options: _controllerOptions,
	);
```

- **VlcPlayerOptions:** Diese Optionen sind zwar nicht unbedingt notwendig, erlauben es eine jedoch genauere Einstellung zu tätigen. Da dieses Plugin im Hintergrund lediglich eine Art Kommandozeilentool von VLC ausführt, können hier sogar alle Einstellungen, die im ursprünglichen Tool möglich sind, als String gesetzt werden, selbst wenn diese vom Plugin als Klasse nicht eigens als Funktion ausprogrammiert wurden.

```
Dateiname: live_view.dart; Klasse: _controllerOptions
VlcPlayerOptions _controllerOptions = VlcPlayerOptions(
advanced: VlcAdvancedOptions([
  VlcAdvancedOptions.networkCaching(300),
]),
rtp: VlcRtpOptions([
  // feeling that with it, it runs smoother
  VlcRtpOptions.rtpOverRtsp(true),
]),
video: VlcVideoOptions([
  VlcVideoOptions.dropLateFrames(false), // better late than never
  VlcVideoOptions.skipFrames(
    true), // never drop any image received -> else sometimes playing
audio without video
]),
);
```

Videostream initialisieren

Da im „VlcPlayerController“ definiert wurde, dass der Stream sich automatisch initialisiert und dann auch automatisch startet, müssen keine weiteren Schritte durchgeführt werden, sodass das Video bzw. der Stream startet.

Video-Overlay

Das Overlay für das Video soll in der Zukunft ähnlich wie die Flugdatenvisualisierung (*siehe: Kapitel 7.1.6.5*) anhand von MQTT die Daten von der Drohne empfangen und dem Benutzer über dem eigentlichen Livestream darstellen.



Abbildung 224: Video-Overlay mit Balken und Platzhalterwerten

Unter diesen Werten fallen zum Beispiel der Pitch und Roll Wert über die Balken an den Rändern aber auch einfache Werte in den Ecken, wie die Batteriespannung, Temperatur oder Höhe.

Extrafunktionen

Einige der Knöpfe im Video-Overlay interagieren und beeinflussen direkt den Status des wiedergegebenen Videos. Um dies zu überhaupt zu ermöglichen, gibt es die Möglichkeit eine Funktion zu definieren die aufgerufen wird, sobald sich irgendetwas an dem Status des Videoplayers ändert. Diese Funktion wird auch als „listener“ bezeichnet und muss direkt nach Initialisierung des eigentlichen Controllers definiert werden:

Dateiname: live_view.dart; Ausschnitt: Defintion einer „listener“-Funktion in der „initState“-Funktion
.... @override void initState() { // Defining the Function that is called as soon as anything changes on the // Status of the Player _videoPlayerController.addListener(listener); }

Der „listener“-Funktion wird keine Art von Hinweis bzw. Parameter gegeben, was sich im Objekt verändert hat, sodass die Funktion aus sehr vielen Bedingungen besteht. Die Hauptart wie man den aktuellen Status des Videoplayers im Code auslesen kann ist über die „value“-Methode innerhalb des Controllers. Diese kann anhand vieler Statusflaggen (Boolean Flags) den ausgeben, was im Videoplayer gerade vor sich hergeht:

```
_videoPlayerController.value.hasError
	videoPlayerController.value.isPlaying
	videoPlayerController.value.isInitialized
...
```

Die Funktionen innerhalb der „listener“-Funktion sind noch in Arbeit und konnten aufgrund des aktuellen Standes der Videoübertragung mit hoher Latenz und Instabilität nur schwer getestet werden. Neben den Funktionen im „listener“ gibt es noch zwei Funktionen des Videoplayers, die direkt als Funktion innerhalb des Controllers zur Verfügung stehen und daher direkt nach Drücken der Buttons unter dem Video ausgeführt werden können. Darunter fallen die Pausierung des Videos und die Aufnahme eines Screenshots.

Dateiname: live_view.dart; Ausschnitt: Screenshot- und Pause-Button UI

```
Row(
  mainAxisAlignment: MainAxisAlignment.spaceEvenly,
  children: [
    IconButton(
      icon: _videoPlayerController.value.isPlaying
        ? const Icon(Icons.pause_circle_outline)
        : const Icon(Icons.play_circle_outline),
      onPressed: () async {
        await _togglePlaying();
      },
    ),
    IconButton(
      iconSize: 24,
      onPressed: () async {
        Logging.info("Taking Screenshot");
        await takeScreenshot();
      },
      icon: const Icon(Icons.screenshot_monitor_rounded),
    ),
  ],
)
```

Dateiname: live_view.dart; Funktion: takeScreenshot

```
Future<void> takeScreenshot() async {
  Uint8List imageData = await _videoPlayerController.takeSnapshot();
  await ImageGallerySaver.saveImage(imageData);
}
```

Bei der Aufnahme eines Screenshots gibt die vorgefertigte Funktion des Video-Controllers, die Bilddaten des Streams als Liste der Pixeldaten. Das Plugin namens ImageGallerySaver

(image_gallery_saver in pub.dev) kann diese Pixeldaten mit der „`saveImage`“-Funktion direkt in die Galerie des Geräts speichern.

```
Dateiname: live_view.dart; Funktion: _togglePlaying
Future<void> _togglePlaying() async {
    _videoPlayerController.value.isPlaying
        ? await _videoPlayerController.pause()
        : await _videoPlayerController.play();
}
```

Um das Video zu pausieren und wieder fortzusetzen, nutze ich die „`pause`“- und „`play`“-Funktionen des Video-Controllers in Kombination mit einer Abfrage, ob das Video gerade läuft oder nicht.

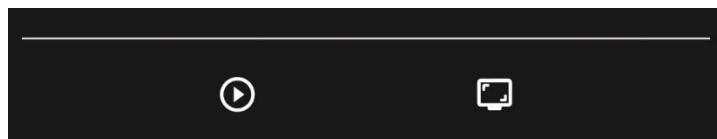


Abbildung 225: Pausieren- und Screenshot-Button unter Live-View

Drohnenvideo anzeigen

Die grundsätzliche Funktion des Players wurde einerseits durch Tests mit einem normalen Videofile aber auch Videos aus dem Netzwerk getestet und diese funktioniert auch grundsätzlich. Wegen der von Haus aus hohen Latenz bei der Konvertierung und Übertragung des Videos (siehe: [Kapitel 8.7.5.4](#)) und den zusätzlichen Schritten, um diesen Stream am Smartphone zu empfangen, ist die Latenz noch einmal höher (~5-10s) als bei Tests mit Geräten, die den Stream auf direktem Wege wie mit der VLC-App abgreifen.

7.1.7 Sidemenü / Drawer

Das Sidemenü auch „Drawer“ genannt, ist das Menü, das sich öffnet, wenn man im Hauptmenü der App nach rechts wischt oder auf den Menübutton in der oberen linken Ecke drückt. Es beinhaltet alle wichtigen Optionen und Informationen für einen eingeloggten User.

```
Dateiname: homepage.dart; Ausschnitt: Definition des Sidemenüs auf Homepage
@Override
Widget build(BuildContext context) {
    return Scaffold(
        drawer: const NavDrawer(),
```

Hinweis: „`NavDrawer`“ ist ein selbstgeschriebenes Widget, welches ein Drawer mit aller Optionen als Kinder definiert

7.1.7.1 Userprofil

An oberste Stelle des Sidemenüs steht das Userprofil. Einerseits in der Form eines runden Icons mit dem Profilbild des Users und darunter direkt als erste Option. Wird auf eines von Beiden gedrückt, so öffnet sich ein neuer Bildschirm mit einer genaueren Ansicht des Userprofils.

Darstellung der Userdaten

Im Userbildschirm werden alle Daten des Users angezeigt. Der Username über dem großen Profilbild und die restlichen Daten wie Name und E-Mail direkt darunter.

Profilbildungsauswahl

Um ein Profilbild auszuwählen, muss man lediglich im Userbildschirm auf das Profilbild klicken, welches unter dem Username angezeigt wird. Dabei wird das Filesystem des Users aufgerufen, wobei der User ein Bild eines typischen Fileformats aussuchen kann, welches nach Bestätigung automatisch zum Profil hinzugefügt wird.

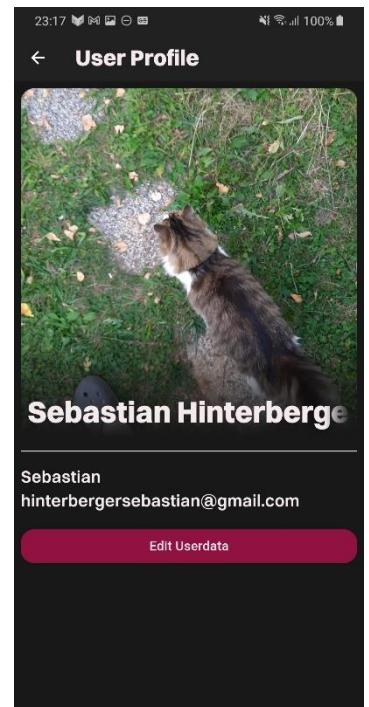


Abbildung 226: Userprofil mit definiertem Profilbild

Dateiname: user_profile.dart; Funktion: _selectProfileImage

```
import 'package:image_picker/image_picker.dart';
.....
Future<void> _selectProfileImage({required BuildContext context}) async {
    try {
        Logging.info(
            Provider.of<AuthenticationProvider>(context, listen: false)
                .currentUser
                ?.photoURL,
        );
        final results = await FilePicker.platform.pickFiles(
            allowMultiple: false,
            type: FileType.image,
        );
        if (results == null) {
            // no file selected by user
            ScaffoldMessenger.of(context).showSnackBar(
                defaultSnackbar("No File Selected", color: Colors.deepOrange),
            );
            // ignore: duplicate_ignore
        } else {
            final path = results.files.single.path;
            final filename = results.files.single.name;
            const folder = "user_profiles";
            await StorageService().uploadFile(folder, path!, filename);

            // delete old profile image from Storage
            var oldStorageURL = context.read<AuthenticationProvider>().storageUrl;
```

```

if (oldStorageURL != "") {
    await StorageService().deleteFile(oldStorageURL);
    Logging.info("Deleted Old Profile Image");
}

// set new Storage URL in Auth User
final newStorageURL = "$folder/$filename";
await Provider.of<AuthenticationProvider>(context, listen: false)
    .updatePhotoURL(newStorageURL);

Logging.info("Local Image File: $path");
}
} on PlatformException catch (error) {
// Permission denied by user most likely
Logging.error(error.toString());
}
}
.....

```

Der Zugriff auf das Dateisystem des jeweiligen Geräts wird durch das sogenannte "FilePicker"-Plugin („file_picker“ in pub.dev) gehandhabt. Dieses Plugin kann mit den gegebenen Dateitypen (Bild, Video, Text,...) und der Angabe ob mehrere Files gewählt werden dürfen, diesen ganzen Prozess in Kombination mit dem Betriebssystem übernehmen. Bei erfolgreicher Auswahl gibt es ein Objekt zurück, das die relevanten Daten der ausgewählten Datei in Bezug auf das Dateisystem enthält (Pfad, Dateiname usw.).

Abschließend wird das Profilbild auf Firebase Storage hochgeladen. Wenn bereits ein altes Profilbild existiert, wird dieses gelöscht. Die Referenz zum Profilbild im entsprechenden Benutzerdokument in Firestore wird aktualisiert und mit dem neuen Pfad des hochgeladenen Bildes aktualisiert.

Änderung der Userdaten

Mit einem Druck auf den Button „Edit Userdata“ öffnet sich ein Menü, in welchem alle Daten des Users anhand von Textfeldern editiert werden können und schlussendlich abgespeichert werden können.

Dateiname: user_profile_options.dart; Funktion: _saveuserdata
<pre> Future<String> _saveuserdata(AuthenticationProvider userProvider, String newEmail, String newUsername, String newName) async { // Updating Email await userProvider.updateEmail(newEmail); //--- This operation is sensitive and requires recent authentication. Log in again before retrying this request. // Updating UserData in UserObject await userProvider.updateName(newName); await userProvider.updateUsername(newUsername); // Updating UserData in FireStore </pre>

```

return UserProfileService().updateMultipleUserValues(
  userId: userProvider.userId,
  newUserData: {
    "username": newUsername,
    "fullName": newName,
  },
);
}

```

Beim Abspeichern der Daten wird die obere Funktion aufgerufen, die die jeweiligen Werte auf der Datenbank, aber auch in der Klasse für den User von Firebase-Auth updatet.

7.1.7.2 Credits / Mitwirkende

Die Credits dienen dazu, dem Benutzer einen kurzen Einblick hinter das UI-Design dieser Diplomarbeit zu gewähren und die Namen sowie die Aufgabenbereiche jedes Teammitglieds in unserer Diplomarbeit zu beschreiben. Zusätzlich wird das Logo der Diplomarbeit und das unseres Sponsors präsent angezeigt.

Dateiname: app_info.dart; Ausschnitt: Anzeige der Logos + Erste Zeile der Tabelle

```

body: SingleChildScrollView(
  child: Padding(
    padding: const EdgeInsets.symmetric(horizontal: 32),
    child: Column(
      children: [
        Image.asset(
          context.read<ThemeManager>().getLogoPath(),
          fit: BoxFit.fitWidth,
        ),
        const VerticalSpace(),
        Text(
          "Sponsored by:",
          textAlign: TextAlign.center,
          style: context.textTheme.bodyLarge,
        ),
        Image(
          image: Provider.of<ThemeManager>(context).isDark
            ? const AssetImage("assets/images/dronetech/logo_light.png")
            : const AssetImage("assets/images/dronetech/logo_dark.png"),
          fit: BoxFit.fitWidth,
        ),
        Table(
          defaultVerticalAlignment: TableCellVerticalAlignment.middle,
          columnWidths: const {
            0: FlexColumnWidth(),
            1: FlexColumnWidth(),
            2: FlexColumnWidth(),
          },
          children: const [

```

```
TableRow(children: [
    TableCell(
        child: AppInfoText(text: "Hardware Conception + Design:"),
    ),
    TableCell(
        child: AppInfoText(text: "Marcel Bieder"),
    ),
]),
```

Wie im Code zu sehen ist, befindet sich der ganze Inhalt dieser Menüpunkts in einem „*SingleChildScrollView*“-Widget, sodass alle der Daten durch das Wischen nach oben und unten erreichbar sind.

Die Anordnung der Texte bei der Nennung des Teams wurde mit einem „*Table*“-Widget und dem damit zusammenhängenden „*TableCell*“-Widget realisiert.

7.1.7.3 Aufgezeichneten Flüge

Die aufgezeichneten Flüge können unter „*Previous Flights*“ im Sidemenü aufgerufen werden. Hierbei werden alle gespeicherten Flüge des jeweiligen Nutzers mit dem Namen und der Aufnahmedauer aufgelistet. Zusätzlich wird noch das vom User ausgewählte Icon angezeigt, um mögliche Wetterbedingungen oder Uhrzeiten bildlich zu markieren.

Laden der Flüge

Um die Flüge zu laden, wird eine Kombination eines Datencaches (lokal) und dem Herunterladen der Daten von der Datenbank benutzt. Sind die lokalen Daten auf dem Stand der Daten am Server, so werden diese benutzt, ansonsten müssen die Daten neu heruntergeladen werden (siehe: [Kapitel 7.1.6.5](#)).

←	Previous Flights		
	Title	Duration	Date
	2024-01-19 00:19:42	Duration: 104 seconds	
	2024-01-19 00:03:09	Duration: 82 seconds	
	2024-01-28 14:36:12	Duration: 57 seconds	
	2024-01-19 00:23:48	Duration: 55 seconds	
		Duration: 46 seconds	
		Duration: 25 seconds	
		Duration: 19 seconds	

Abbildung 227: Liste der aufgezeichneten Flüge

```
Previous_fligths.dart; Funktion: _getRecords
Future<List?> _getRecords(BuildContext context) async {
    // fetching data:
    // if empty records -> read from db
    // if local timestamp (flightRecordsAge) is older (<) than the one in the
    database
    final dataCache = Provider.of<DataCache>(context);
    final userId = context.read<AuthenticationProvider>().userId;
    final bool emptyFlightRecords =
        Provider.of<DataCache>(context).previousFlights.isEmpty;
```

```
// checking if data needs to be reloaded
if (dataCache.dataAges["previousFlights"] <
    (await UserProfileService().fetchDataAge(
        userId: userId, timestampKey: "flight_data_age")) ??
    -1) ||
emptyFlightRecords) {
// data either older than on db or non-existent
return UserProfileService().getFlightDataSets(userId);
}
```

Sortierungsmöglichkeiten

Bei der Liste der aufgezeichneten Flüge gibt es anhand von 3 Buttons am oberen Rand des Bildschirms die Aufzeichnungen nach seinem Belieben zu sortieren. Die Optionen umfassen:

- Title / Titel / Name
- Duration / Länge der Aufzeichnung
- Date / Datum

Bei einem Klick auf einer dieser 3 Optionen werden die Aufzeichnung automatisch sortiert.

Innerhalb des Codes funktioniert die Sortierung, indem alle der Daten in der Liste in einem Array mit dem Namen „data“ abgelegt sind, welches in einem „ListView.builder“ in dem gezeigten Format gerendert wird. Dadurch, dass der gesamte Aufbau der Liste lediglich durch die sequenzielle Struktur der Daten basiert, kann man die Daten im je nach Einstellung im Array sortieren, den Bildschirm neu rendern und die Daten werden als sortiert angezeigt.

Dateiname: previous_flights.dart; Ausschnitt: Aufruf der `_sortData`-Funktion + `ListView.builder`

```
// sorting data
_sortData();

// displaying data
return ListView.separated(
    physics: const BouncingScrollPhysics(),
    itemCount: data!.length + 1,
    separatorBuilder: (context, index) {
        return const VerticalSpace(
            height: 0,
        );
    },
    itemBuilder: (context, index) {
        if (index == 0) {
            // building header
            return _getListHeader();
        }

        index = index - 1; // setting index back to normal value
        String weatherIcon = data?[index]["weather"] ?? "wi-day-sunny";
    }
)
```

```

return Column(
  children: [
    Divider(
      height: 8,
      thickness: 1,
      color: Colors.grey.shade900,
      endIndent: 8,
      indent: 8),
    ListTile(
      leading: Image.asset(
        "${context.read<ThemeManager>().getWeatherIconPath()}$weatherIcon.png",
      ),
      title: Text(data?[index]["title"] ?? ""),
      subtitle: Text(
        "Duration: ${_getDurationInSeconds(index)} seconds",
        style: context.textTheme.bodyMedium,
      ),
      onTap: () {
        Navigator.of(context).push(MaterialPageRoute(
          builder: (context) => PreviousFlight(
            flightData: data?[index] ?? {},
          ),
        )));
      },
    ),
  ],
].animate(interval: 200.ms).fade(duration: 100.ms));
},
);

```

Als erstes Item in der Liste wird der Header mit den verschiedenen Sortierungsoptionen gerendert. Dies hat nebenbei zur Folge das der Index von der Liste und dem Datenarray den Offset 1 haben. Der Header besteht aus einer Reihe an 3 simplen Buttons mit dem Text, die die Art der Sortierung ändern und mit „setState“ den Bildschirm neu laden.

Dateiname: previous_flights.dart; Funktion: _getHeaderButton
<pre>Expanded _getHeaderButton(String text, SortingTypes type) { return Expanded(child: TextButton(child: Text(text), onPressed: () { setState(() { sortType = type; }); },),); }</pre>

Die eigentliche Funktion, die das Datenarray vor dem neu laden des Bildschirms sortiert sieht wie folgt aus:

```
Dateiname: previous_flights.dart; Funktion: _sortData
void _sortData() {
    switch (sortType) {
        case SortingTypes.title:
            data?.sort((a, b) => b["title"].compareTo(a["title"]));
            break;
        case SortingTypes.duration:
            data?.sort((a, b) => (b["endTimestamp"] - b["startTimestamp"])
                .compareTo((a["endTimestamp"] - a["startTimestamp"])));
            break;
        case SortingTypes.date:
            data?.sort((a, b) => b["endTimestamp"].compareTo(a["endTimestamp"]));
            break;
        default:
            Logging.error("Error, Invalid Sorting Type");
            break;
    }
}
```

Um die Daten zu sortieren, wird eine vorgefertigte Funktion für alle Typen von Listen in Flutter benutzt. Diese erlaubt es einem die Daten in einem Array mit einer bestimmten Bedingung zu sortieren, solange jedes Element in der List jene diese Bedingung erfüllt.

Einsicht in vorherigen Flug

Bei Auswahl eines Fluges werden die aufgezeichneten Daten sowie bei der eigentlichen Aufzeichnung in mehreren Tabs angezeigt. Zu den Flugdaten gibt es noch eine Seite mit allgemeinen Informationen zum Flug, die Fluglänge, Datum, Wetter und Name beinhaltet. Alle dieser Bildschirme sind durch eine Navigationsleiste am unteren Rand des Bildschirmes zugänglich.

Eine Extrafunktion bei der Anzeige der Daten ist die Addition von bestimmten Werten, die anhand der Messdaten berechnet wurden. Darunter fallen Maximalwert, Minimalwert und Durchschnittswert.

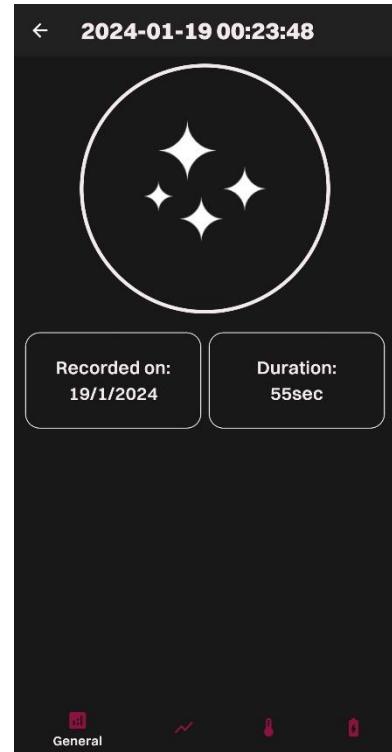


Abbildung 228: Allgemeiner Tab eines vorherigen Fluges

7.1.7.4 Einstellungen

Die Einstellungen oder „Settings“ innerhalb der App, dienen als Menü indem verschiedene Einstellungen für jegliche Funktion der App getätigter werden können.

Dark- / Light-Mode

Mit dem Kippschalter, bzw. „toggle switch“, lässt sich das allgemeine Design der App zwischen einem hellen und dunklen Design wechseln. Hierfür wird bei Änderung des Schalters in den Einstellungen eine Funktion aufgerufen, die alle notwendigen Änderungen vornimmt.

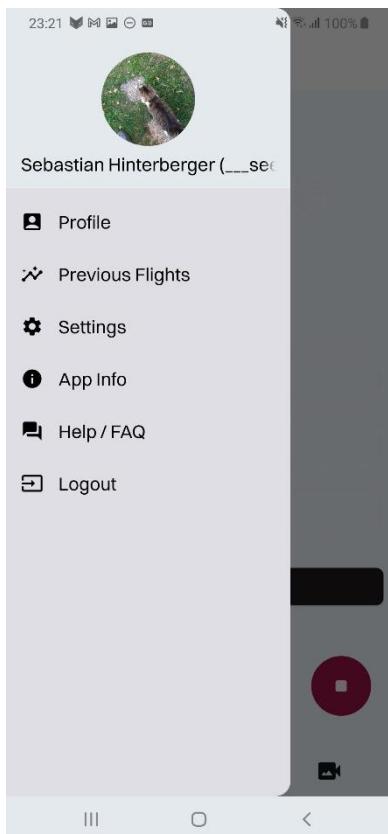


Abbildung 230: Sidemenü mit hellem Design

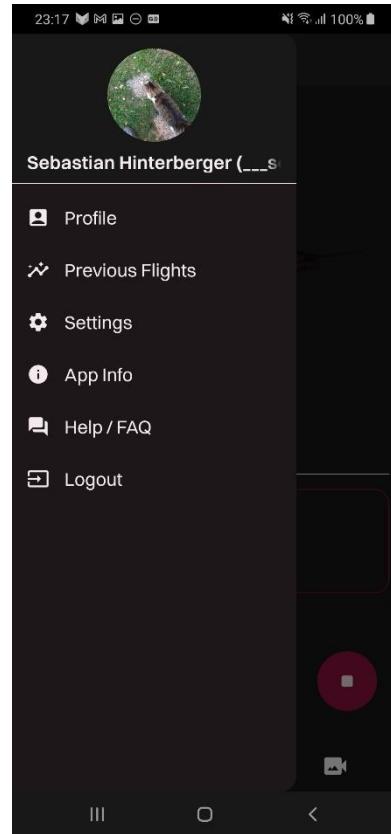


Abbildung 229: Sidemenü mit dunklem Design

Dateiname: <code>settings.dart</code> ; Ausschnitt: Toggle-Switch für Light-/Dark Mode
<code>SwitchListTile.adaptive(activeColor: Colors.white, inactiveTrackColor: Colors.black, value: themeManager.isDark, title: Text("Theme Mode", style: context.textTheme.bodyLarge?.copyWith(color: themeManager.isDark ? Colors.white : Colors.black),), subtitle: Text("Switch to change the theme", style: context.textTheme.bodyMedium?.copyWith(color: themeManager.isDark ? Colors.white : Colors.black),),)</code>

```
'Light/Dark Mode Toggle',
style: context.textTheme.bodySmall?.copyWith(
    color: themeManager.isDark ? Colors.white : Colors.black),
),
controlAffinity: ListTileControlAffinity.trailing,
onChanged: ((value) {
    themeManager.setTheme(value);
    appNavigationColorSettings(value);
    SettingsService().addNewSettings(
        userId: Provider.of<AuthenticationProvider>(context,
            listen: false)
        .userId,
        settings: {
            "isDark": themeManager.isDark,
        },
    );
}),
),
```

- Aufruf des Providers mit dem Namen „ThemeManager“ indem das allgemeine Design gehandelt wird. Durch die Setzung der _themeMode-Variable, wird das Design automatisch adaptiert, da diese Variable zu Start der App zwischen einem hellen und einem dunklen Design unterscheidet.

Dateiname: settings.dart; Funktion: setTheme

```
setTheme(bool toggle) {
    _themeMode = toggle ? ThemeMode.dark : ThemeMode.light;
    _isDark = toggle;
    notifyListeners();
}
```

Dateiname: main.dart; Ausschnitt: Definition des Themes bei Erstellung der MaterialApp

```
.....
@Override
Widget build(BuildContext context) {
    cachingImages(context);
    return MaterialApp(
        debugShowCheckedModeBanner: false,
        theme: AppTheme.lightTheme,
        darkTheme: AppTheme.darkTheme,
        themeMode: Provider.of<ThemeManager>(context).themeMode,
        title: 'FPV-Drone-Application',
    ....
```

Durch die Änderung der Variable im Provider und dem Aufruf der Funktion „notifyListeners“, wird das Design augenblicklich in der ganzen App geändert.

Hinweis: In diesem Bild wurde ausnahmsweise die Navigation des Betriebssystems (Leiste oben + Navigation unten) um die veränderten Farben in beiden Einstellungen darzustellen

7.1.7.5 Help / FAQ

Der FAQ-Teil dient dazu dem Benutzer eine erste Anlaufstelle bei Problemen zu geben, wobei man die am häufigsten gestellten Fragen vorweg beantwortet und mögliche Fehlermeldungen zu erklären.

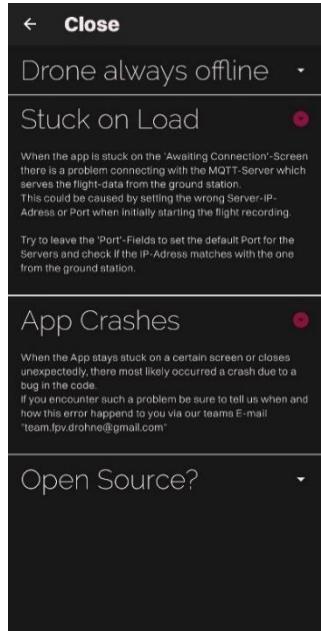


Abbildung 231: Help- / FAQ-Bildschirm

Dateiname: help.dart; Ausschnitt: UI-Code der ausfahrbaren Tabs

```
body: ListView.builder(
  itemCount: data.length,
  itemBuilder: (context, index) {
    return ExpansionTile(
      title: Text(
        data.keys.elementAt(index),
        style: context.textTheme.headlineMedium,
      ),
      trailing: Icon(getIcon(index)),
      children: <Widget>[
        ListTile(
          title: Text(
            data.values.elementAt(index),
            style: context.textTheme.bodySmall,
          )),
      ],
      onExpansionChanged: (bool isExpanded) {
        setState(() {
          expandedTiles[index] = isExpanded;
        });
      },
    );
  },
);
```

Die einzelnen Elemente die anhand eines Touch-Inputs ausgefahren und eingefahren werden können, sind das sogenannte „*ListTile*“-Widget. Die Texte, die innerhalb der einzelnen Tabs angezeigt werden, sind mit der Überschrift in einer simple „Map“ fix definiert.

```
Dateiname: help.dart; Ausschnitt: Erstes Element der angezeigten Texte
.....
Map<String, String> data = {
    "Drone always offline":
        """This error occurs when the specified 'isOnline'-Flag hasn't been
set by the Groundstations script.
....
```

7.1.7.6 Logout

Die unterste Funktion im Sidemenü ist der Logout-Button, der neben einer abgelaufenen Session in Firebase-Auth oder einem Netzwerkproblem, die einzige Möglichkeit darstellt von der Homepage in den Startbereich der App zu gelangen.

```
Dateiname: side_menu.dart; Funktion: logout
void logout() async {
    // Google Sign Out
    try {
        await GoogleSignIn().disconnect();
    } catch (e) {
        Logging.debug('Failed to disconnect on signout');
    }

    // Clear Datacache -> preventing seeing false previousFlights
    Provider.of<DataCache>(context, listen: false).previousFlights.clear();

    // Auth Sign Out
    await FirebaseAuth.instance.signOut();
    Navigator.pushReplacementNamed(
        context,
        WelcomeScreen.id,
    );
}
```

Wird der Button gedrückt, so wird die „*logout*“-Funktion aufgerufen. Diese loggt sowohl einen mit Google angemeldeten User als auch einen „normalen“ User von der App mit den gegebenen Funktionen der jeweiligen Auth-Klassen aus.

Damit bei einem neuen Login keine Flüge eines vorherigen Users angezeigt werden, muss zusätzlich noch die Liste der vorherigen Flüge im Cache geleert werden.

7.2 Firebase-Backend

Das Backend der Visualisierungsapp besteht im Wesentlichen aus Firebase, dem Backend von Google. Dieses hat eine Vielzahl an Services von Hosting, Cloud-Storage, Echtzeitdatenbanken, Autorisierung aber noch viele Weitere. Der Grund für die Nutzung dieses Services ist die simple und komplett dokumentierte Einbindung mit Flutter, dem Framework der Visualisierungsapp. Um diese Produkte zu nutzen, müssen vier Installationsschritte befolgt werden:

1. Installieren der Firebase-CLI
2. Flutterfire für das jeweilige Projekt konfigurieren
3. Firebase in der App initialisieren
4. Plugins hinzufügen

Zwar gibt es auch andere Installationsmethoden, wobei einige Konfigurationsfiles editiert werden müssten. Aufgrund der Simplizität der oben beschriebenen Variante und der offiziellen Empfehlung des Herstellers wird jedoch die andere Methode benutzt.

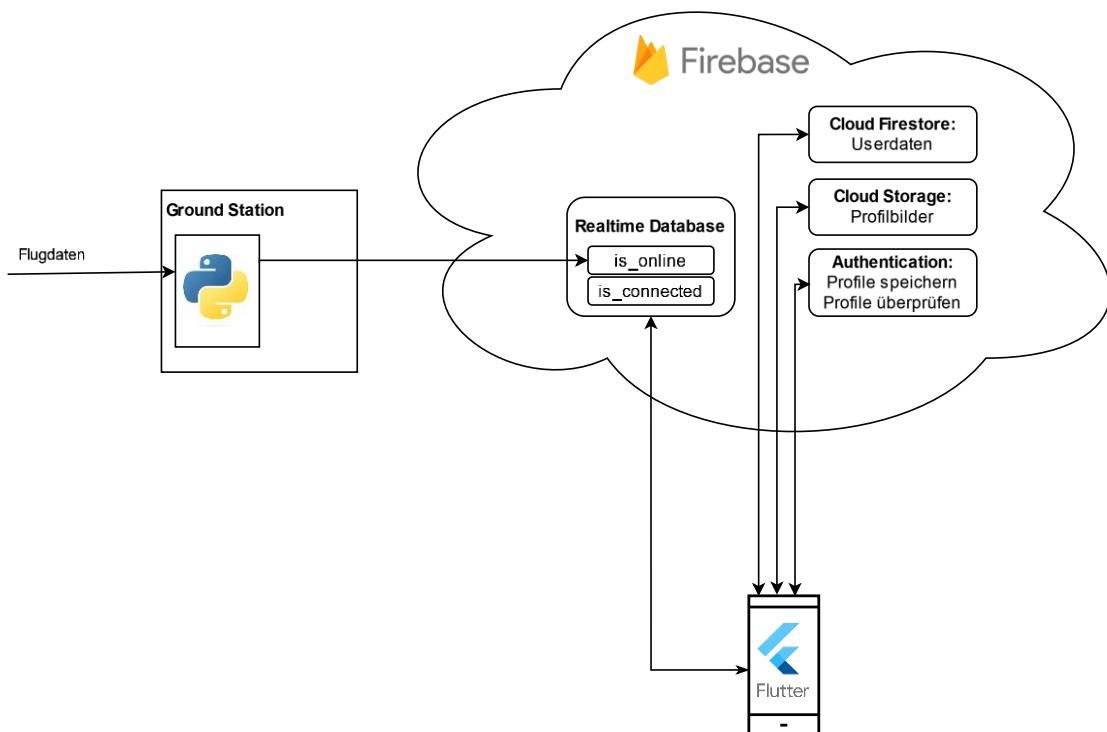


Abbildung 232: Übersicht der Firebase Services in Verbindung mit der App und Ground-Station [PYLO] [FFLO]

7.2.1 Installation via Firebase CLI

7.2.1.1 Firebase CLI – Setup

Es gibt mehrere Methoden das Kommandozeilentool zu installieren. Eine der Varianten ist über den Package Manager namens npm von Node.js. Ist Node.js nicht bereits mit dem npm-Tool installiert, so muss dieses über deren Website heruntergeladen und installiert werden.

```
npm install -g firebase-tools
```

Wird der Befehl erfolgreich ausgeführt so ist der „firebase“-Befehl global in der Kommandozeile des Computers verfügbar.

Node.js und Node Package Manager (npm) – Installation

Um Node.js zu installieren kann man auf die Downloadseite auf der Website von Node.js (<https://nodejs.org/en/download/>) gehen und einen Installer für sein gewünschtes Betriebssystem herunterladen.

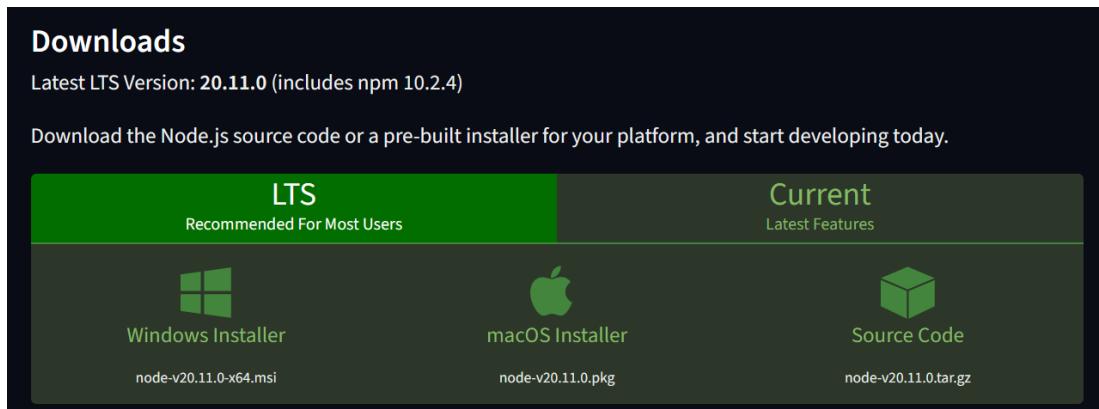


Abbildung 233: Downloadoptionen von Node.js auf der Produktseite [NJSO]

Führt man die heruntergeladene .msi-Datei aus, so öffnet sich ein Install-Wizard für Node.js mit dem man alles grafisch einstellen kann.

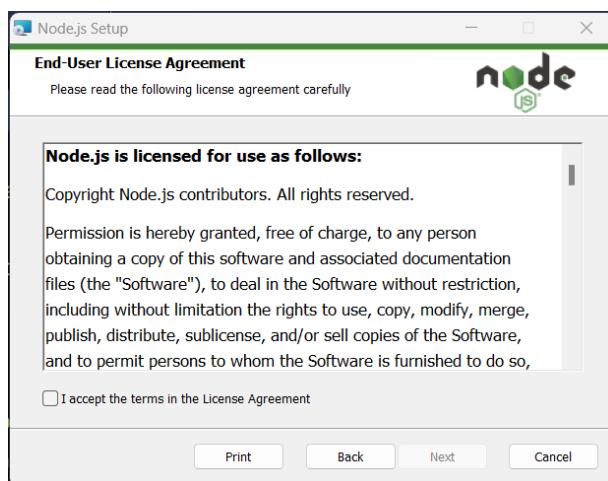


Abbildung 234: Ausschnitt des Setup.exe Files von Node.js

7.2.1.2 Firebase-Login am Computer

Um auf die Projekte von Firebase auf der Cloud lokal auf der jeweiligen Maschine zugreifen zu können muss man mit seinem Firebase-Account am Terminal eingeloggt sein. Dies geht mit folgendem Kommando:

```
firebase login
```

Bei Ausführung muss zuerst die Entscheidung getroffen werden, ob bestimmte Daten von Google gesammelt werden dürfen. Nach Eingabe von entweder „Y/N“ öffnet sich ein Browserfenster, in welchem ein Google-Account ausgewählt werden muss. Falls bereits Firebase-Projekte erstellt wurden, sollte man den jeweiligen Account auswählen und weiters die Berechtigungsanfragen annehmen.

```
C:\Users\hinte>firebase login
i  Firebase optionally collects CLI and Emulator Suite usage and error reporting information to help improve our product
s. Data is collected in accordance with Google's privacy policy (https://policies.google.com/privacy) and is not used to
identify you.

? Allow Firebase to collect CLI and Emulator Suite usage and error reporting information? Yes
i  To change your data collection preference at any time, run `firebase logout` and log in again.

Visit this URL on this device to log in:
https://accounts.google.com/o/oauth2/auth?client\_id=563584335869-fgrhgmd47bqnekij5i8b5pr03ho849e6.apps.googleusercontent.com&scope=email%20openid%20https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fcloudplatformprojects.readonly%20https%3A%2F%2Fwww.googleapis.com%2Fauth%2Ffirebase%20https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fcloud-platform&response\_type=code&state=266494253&redirect\_uri=http%3A%2F%2Flocalhost%3A9005

Waiting for authentication...
```

Abbildung 235: Firebase CLI Login + Informationsanfrage

Nach einem erfolgreichen Login und Zustimmung zu den benötigten Berechtigungen öffnet sich im gleichen Browser ein Popup mit einer Bestätigung des Logins.

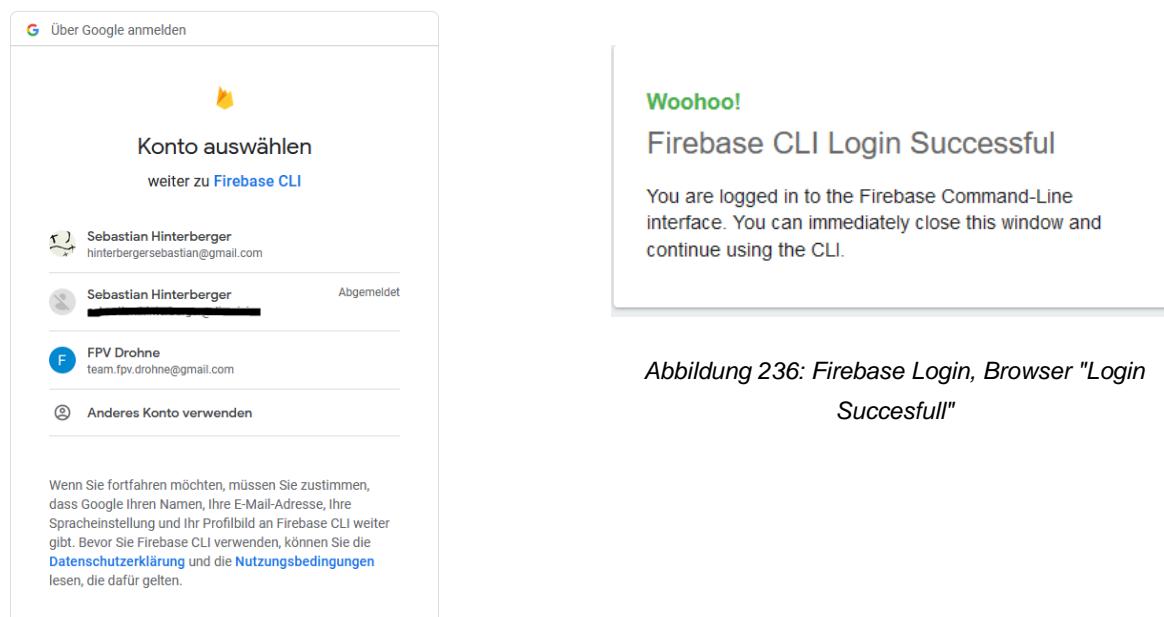


Abbildung 237: Google-Account Auswahl bei Firebase CLI Login

Zusätzlich zu dem Fenster im Browser, sollte der „firebase login“ Befehl in der Kommandozeile mit einer Erfolgsmeldung beendet worden sein:

```
Waiting for authentication...
+ Success! Logged in as hinterbergersebastian@gmail.com
```

Abbildung 238: Firebase CLI erfolgreicher Login

7.2.2 Erstellung eines Firebase-Projekts

Um ein neues Firebase-Projekt zu erstellen, muss man sich nach einer Anmeldung auf der Firebase-Webseite mit einem Google-Account, auf die Konsole von Firebase gehen, wo man zur Projektübersicht kommt. Die Adresse für die Projektübersicht auf der Firebase-Konsole ist: <https://console.firebaseio.google.com> (Letzter Aufruf: 29.02.24) Mit einem Klick auf die Fläche mit der Bezeichnung „*Projekt hinzufügen*“, und der Befolgung der Schritte wird automatisch ein neues leeres Projekt erstellt. Dieses Projekt kann nach Erstellung geöffnet werden und ist auch in der Konsole neben den Projekterstellungsbutton aufrufbar.

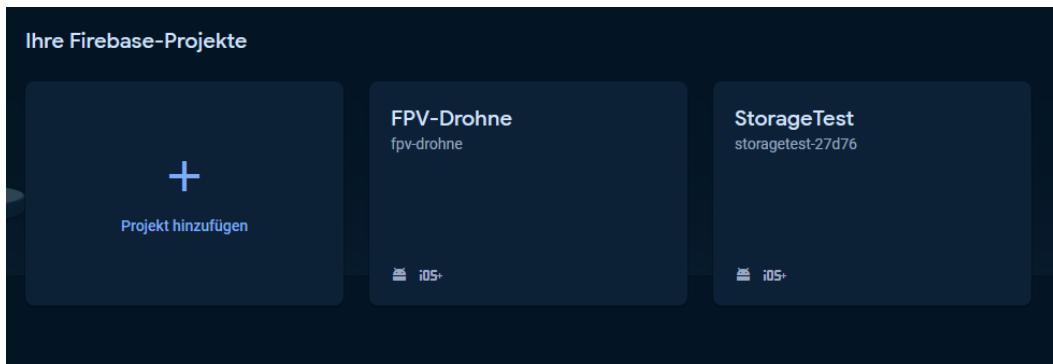


Abbildung 239: Projektübersicht in Firebase Konsole

7.2.2.1 Services aufsetzen

Um innerhalb eines leeren Projekts eine Datenbank zu erstellen oder jeglichen anderen Service aufzusetzen muss man innerhalb des Projekts auf den Tab mit dem Titel „Entwicklung“ drücken, womit sich dann eine Liste mit allen vorhandenen Services öffnet. In dieser Liste kann man einen Service auswählen und es tut sich die Seite für den jeweiligen Service auf. Der Service lässt sich durch Befolgung der Schritte aktivieren.

Diese Schritte unterscheiden sich je nach Service mehr oder weniger. So muss bei der Erstellung einer „Cloud-Firestore“ oder „Realtime Database“-Datenbank der Standort festgelegt werden oder bei der Authentication die unterstützten Anbieter ausgewählt werden.

Die Services, die im Projekt aufgesetzt wurden, sind einerseits auf der linken Seite der Projektseite verlinkt. Zusätzlich werden noch Statistiken zu den benutzten Services auf der Projektübersicht angezeigt.

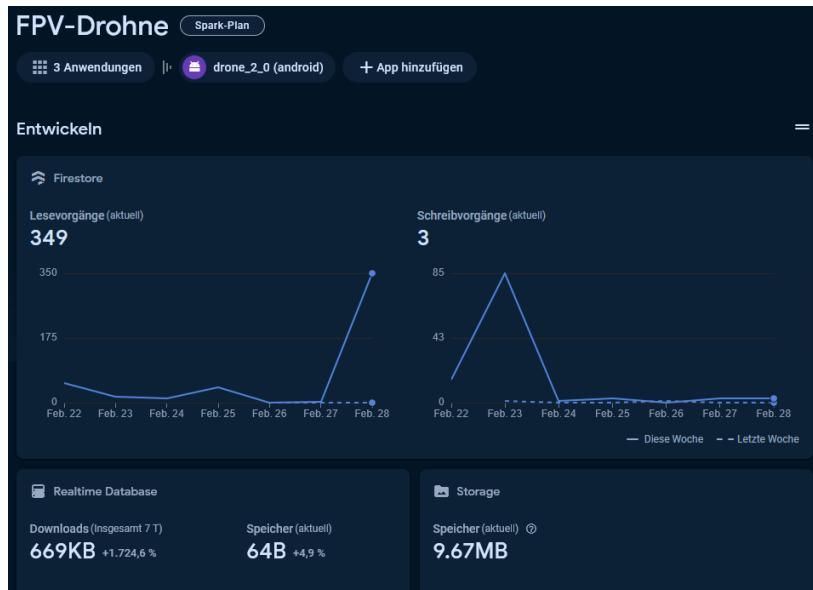


Abbildung 240: Projektübersicht mit autogenerierten Statistiken

7.2.3 Einbindung in Flutter

7.2.3.1 Flutterfire Configure

Im Verzeichnis eines bereits erstellten Flutter-Projekts kann man die benötigten Konfigurationen automatisch hinzufügen, indem man im Terminal folgende Kommandos ausführt:

```
dart pub global activate flutterfire_cli
```

Dieses Kommando muss bei einer erfolgreichen Installation von Flutterfire nicht mehr ausgeführt werden, kann aber später noch benutzt zu werden, um den Status der Installation zu überprüfen und installierte Pakete zu aktualisieren. Bei erfolgreicher Ausführung und Updates endet der Output dieses Kommandos mit folgendem Text:

```
Building package executables... (10.1s)
Built flutterfire_cli:flutterfire.
Installed executable flutterfire.
Activated flutterfire_cli 0.2.7.
```

Um Flutterfire nun für sein Projekt zu Konfigurieren und mit einem Firebase-Projekt zu verbinden, muss folgendes Kommando ausgeführt werden:

```
flutterfire configure
```

Bei richtiger Konfiguration lädt Flutterfire jetzt alle bereits erstellten Projekte des eingeloggten Accounts und bittet eines auszuwählen:

```
PS C:\Users\hinte\OneDrive\Dokumente\temp> flutterfire configure
i Found 2 Firebase projects.
? Select a Firebase project to configure your Flutter application with >
> fpv-drohne (FPV-Drohne)
storage-test-27d76 (StorageTest)
<create a new project>
```

Abbildung 241: Auswahl eines Firebase-Projekts für Verbindung mit Flutter-Projekt

Nach Auswahl eines Projekts müssen lediglich die Plattformen ausgewählt werden für die man beabsichtigt zu entwickeln:

```
i Found 2 Firebase projects.
✓ Select a Firebase project to configure your Flutter application with - storage-test-27d76 (StorageTest)
? Which platforms should your configuration support (use arrow keys & space to select)? >
✓ android
✓ ios
✓ macos
✓ web
```

Abbildung 242: Auswahl von Plattformen für Firebase-Projekt

Hinweis: In diesem Beispiel wird ein Beispielprojekt namens StorageTest zur Veranschaulichung des Installationsprozesses benutzt

Zuletzt konfiguriert Flutterfire das Firebase-Projekt für die ausgewählten Plattformen und erstellt benötigte Verzeichnisse und Files in den jeweiligen Ordnern der Plattformen.

```
✓ Select a Firebase project to configure your Flutter application with - storage-test-27d76 (StorageTest)
✓ Which platforms should your configuration support (use arrow keys & space to select)? - android, ios, macos, web
i Firebase android app com.example.temp is not registered on Firebase project storage-test-27d76.
i Registered a new Firebase android app on Firebase project storage-test-27d76.
i Firebase ios app com.example.temp is not registered on Firebase project storage-test-27d76.
i Registered a new Firebase ios app on Firebase project storage-test-27d76.
i Firebase macos app com.example.temp.RunnerTests is not registered on Firebase project storage-test-27d76.
i Registered a new Firebase macos app on Firebase project storage-test-27d76.
i Firebase web app temp (web) is not registered on Firebase project storage-test-27d76.
i Registered a new Firebase web app on Firebase project storage-test-27d76.

Firebase configuration file lib.firebaseio_options.dart generated successfully with the following Firebase apps:

Platform  Firebase App Id
web      1:680391817427:web:283efee218541080dfffc26
android   1:680391817427:android:1aec834da44e1f36dfffc26
ios       1:680391817427:ios:f7e308c696622948dfffc26
macos     1:680391817427:ios:36c96c4e815c3292dfffc26
```

Abbildung 243: Flutterfire Plattformkonfiguration für alle Plattformen

7.2.3.2 Firebase in Flutter-Code initialisieren

Um Firebase innerhalb des Codes zu nutzen, muss zu Beginn der Applikation die Funktion `Firebase.initializeApp` aufgerufen werden, nachdem mit der `WidgetsFlutterBinding.ensureInitialized`-Funktion die Verwendung des `async`-Keywords in der `main`-Funktion erlaubt wurde. Dabei muss die jeweilige Plattform als Parameter übergeben werden (`DefaultFirebaseOptions.currentPlatform` wählt automatisch die Plattform

aus). Hierfür müssen das Paket namens „firebase“ und die von Flutterfire erstellte Datei namens „firebase_options.dart“ importiert werden.

```
...
import "package:firebase_core/firebase_core.dart";
...
import 'firebase_options.dart';
...

void main() async {
  WidgetsFlutterBinding.ensureInitialized();
  ...
  await Firebase.initializeApp(options:
DefaultFirebaseOptions.currentPlatform);
  ...
}
```

Hinweis: Die Punkte im Code sollen anderen Code im main.dart-File der App repräsentieren die NICHT für die Initialisierung von Firebase notwendig sind

7.2.3.3 Firebase Dienste hinzufügen

Um alle benötigten Dienste in Flutter nutzen zu können, kann man diese so wie jedes andere Package über „pub add <name>“ im Terminal des Projekts automatisch hinzufügen. Die von der Visualisierungsapp verwendeten Dienste können wie folgt hinzugefügt werden:

```
flutter pub add cloud_firestore
flutter pub add firebase_auth
flutter pub add firebase_storage
flutter pub add firebase_database
```

7.2.4 Nutzung der verschiedenen Datenbanksysteme

Firebase besteht nicht nur aus einem Produkt, sondern aus einer Sammlung an Produkten, wobei jedes ihren eigenen Nutzen hat. Diese Produktvielfalt reicht von Services zur Sammlung von Nutzerdaten bis hin zur simplen Datenbank. Die Beschreibung jeder dieser Services finden man auf der offiziellen Webseite von Firebase und wie diese mit Flutter interfacen auf der offiziellen Flutterfire-Webseite (<https://firebase.flutter.dev/docs/overview>, Letzter Aufruf: 29.02.24). [FBPR]

7.2.4.1 Auth

„Auth“ ist ein Service, wobei man Zugriff auf eine Vielzahl von Autorisierungsmethoden für seine App bekommt. Von dem simplen Login mit E-Mail und Password, bis hin zum Login mit verschiedenen weit verbreiteten Accounts, wie Google, Microsoft, Apple, usw. Zusätzlich erlaubt dieser Service es einem leicht den Loginstatus eines Users zu speichern und bei späteren Sessions abzurufen. Dies ermöglicht es einem, ohne der direkten Nutzung von Cookies oder anderen ähnlichen Methoden einem User automatisch einzuloggen.

Kennung	Anbieter	Erstellt ↓	Angemeldet	Nutzer-UID	
kinder.lendl@gmail.com		30.01.2024	30.01.2024	RztCCBkwpzZPN6njVmIpl7G7...	...
team.fpv.drohne@gmail...		11.01.2024	23.01.2024	Z38JWgTK8MfeHnSssUBGUM...	...
jesser@blue.man		10.11.2023	10.11.2023	1YutrlM93cQiuqeylEWvxjnRuS...	...
maximilian.lendl@gmai...		10.11.2023	10.02.2024	XZnP0Ef4ATWp8o1wSSrPHm...	...

Abbildung 244: Ausschnitt beispielhafter Useraccounts (Google, E-Mail + Password) in Auth [FBAU]

E-Mail + Passwort – Login

Beim Login mit E-Mail und Passwort wird nur mit der Übergabe einer E-Mail und eines Passworts, ein Useraccount mit User-ID erstellt. Der erstellte User wird bis zur manuellen Löschung des Accounts bestehen bleiben und kann zur Verifizierung benutzt werden.

Google Login

Der einzige Login über Drittanbieter ist über einen Google-Account. Dieser kann beim Loginbildschirm über ein Icon ausgewählt werden, wobei dann ein Popup mit einer Liste an all den registrierten Google-Accounts am Gerät erscheint. Diese Loginmethode ist aufgrund der Sicherheit eines Google-Account, deutlich sicherer und dazu noch effizienter als die Anmeldung über E-Mail und Passwort.

7.2.4.2 Firestore Database

Firestore ist eine dokumentenbasierte NoSQL Datenbank und wird innerhalb der App genutzt, um die benutzerspezifischen Daten eines Users zu speichern. Diese Daten reichen von bestimmten Einstellungen, allgemeinen Userdaten wie Name oder E-Mail, einem Link zum Profilbild in Firebase-Storage und den gespeicherten Flugdaten. Aufgrund der Struktur dieses Services ist dieser nur geeignet zum Speichern von einfacherem Text, Booleans oder Zahlen, weswegen das Profilbild auch nur als Link und nicht als Ganzes abgespeichert wird.

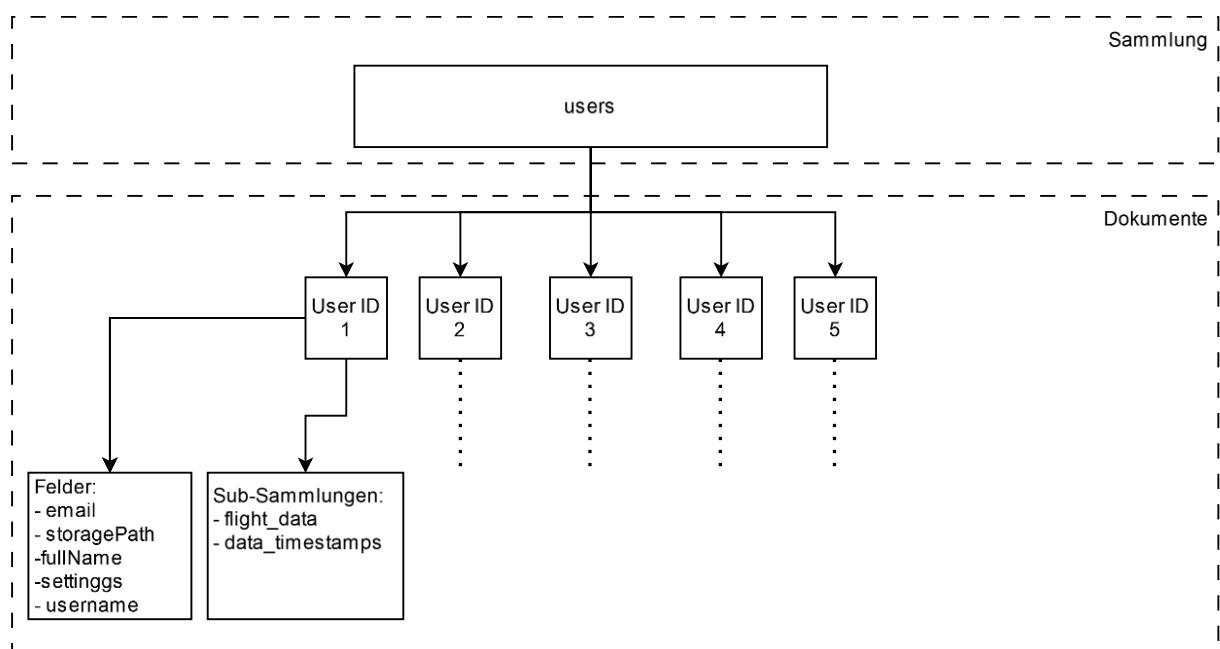
Struktur:

Abbildung 245: Aufbau Cloud Firestore Userdaten

- **Hauptsammlung („users“):** Die Hauptsammlung namens „users“ beeinhaltet alle User die sich jemals angemeldet haben. Der Name eines dieser Userdokumente setzt sich aus der einzigartigen User-ID zusammen, die bei einer Anmeldung mit Auth automatisch generiert wird.
- **Userdokumente:** Im Userdokument werden alle userspezifischen Daten gespeichert, mit der Ausnahme des Profilbildes, welches lediglich als Link zu Firebase-Storage abgelegt wird. Die „Felder“ innerhalb des Dokuments sind einfache Variablen (Texte, Zahlen, Booleans) wie die E-Mail-Adresse oder der Username. Abgesehen von den Feldern gibt es zudem noch die Möglichkeit weitere „Sub-Sammlungen“ innerhalb dieses Dokuments zu erstellen.
- **Dokumentsammlungen / „Sub-Sammlungen“:** In einem Userdokument können eigene Sammlungen erstellt werden. Diese werden genutzt, um Flugdaten abzuspeichern. Hierfür werden 2 Sammlungen angelegt. Eines für die eigentlichen Flugdaten („flight_data“) und das Zweite („flight_data_age“) um lediglich Zeitstempel für den lokalen Cache abzuspeichern. Diese Zeitstempel haben den Sinn das Alter der Flugdaten vergleichen zu können und Folgedessen Datenrate zu Firebase zu sparen.

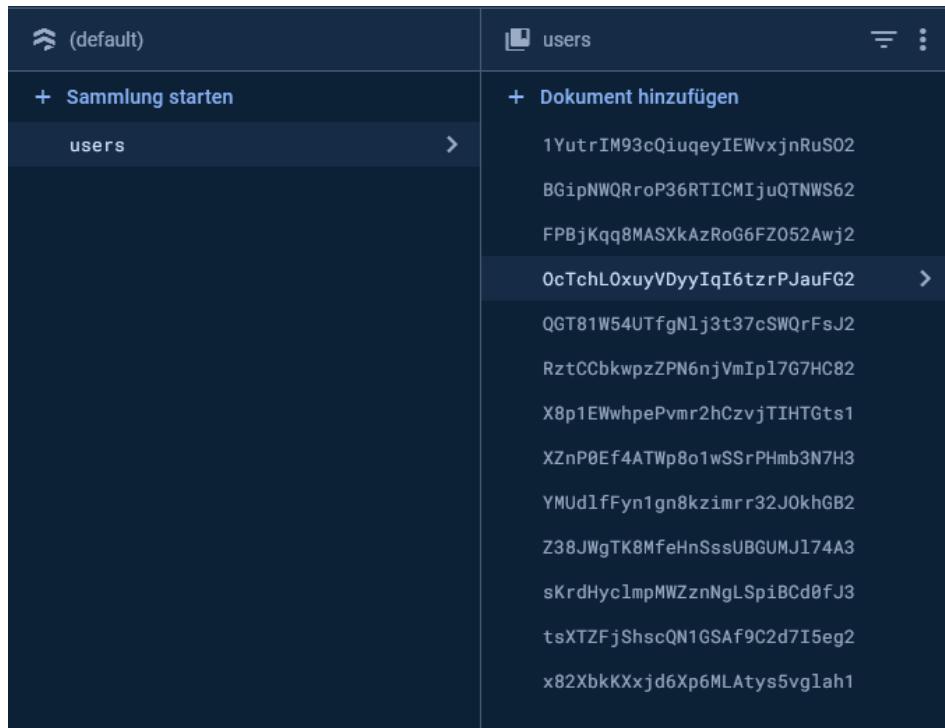


Abbildung 246: Hauptsammlung und Userdokumente in Firestore

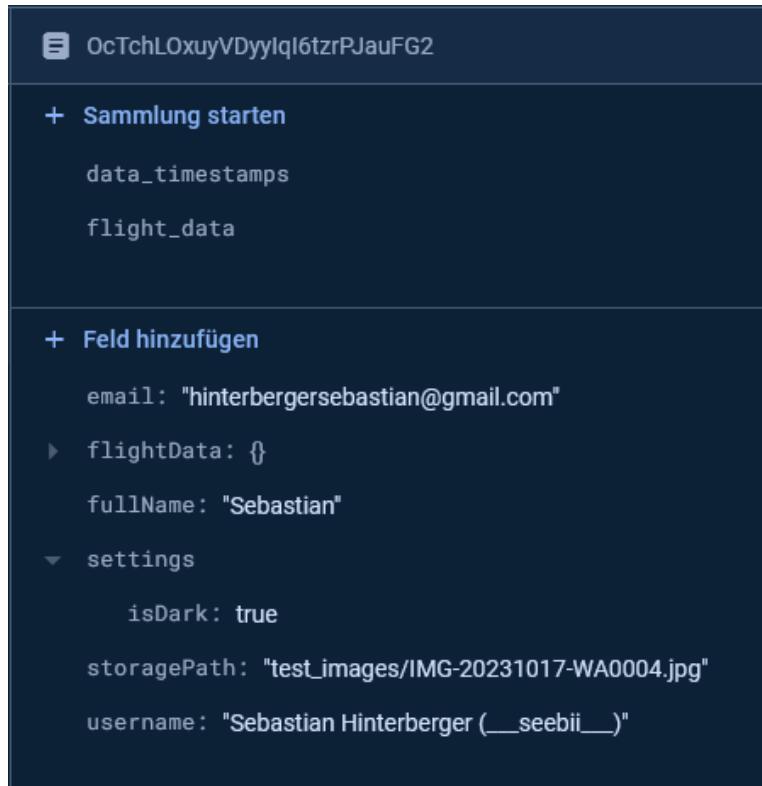


Abbildung 247: Userdokument in Firestore

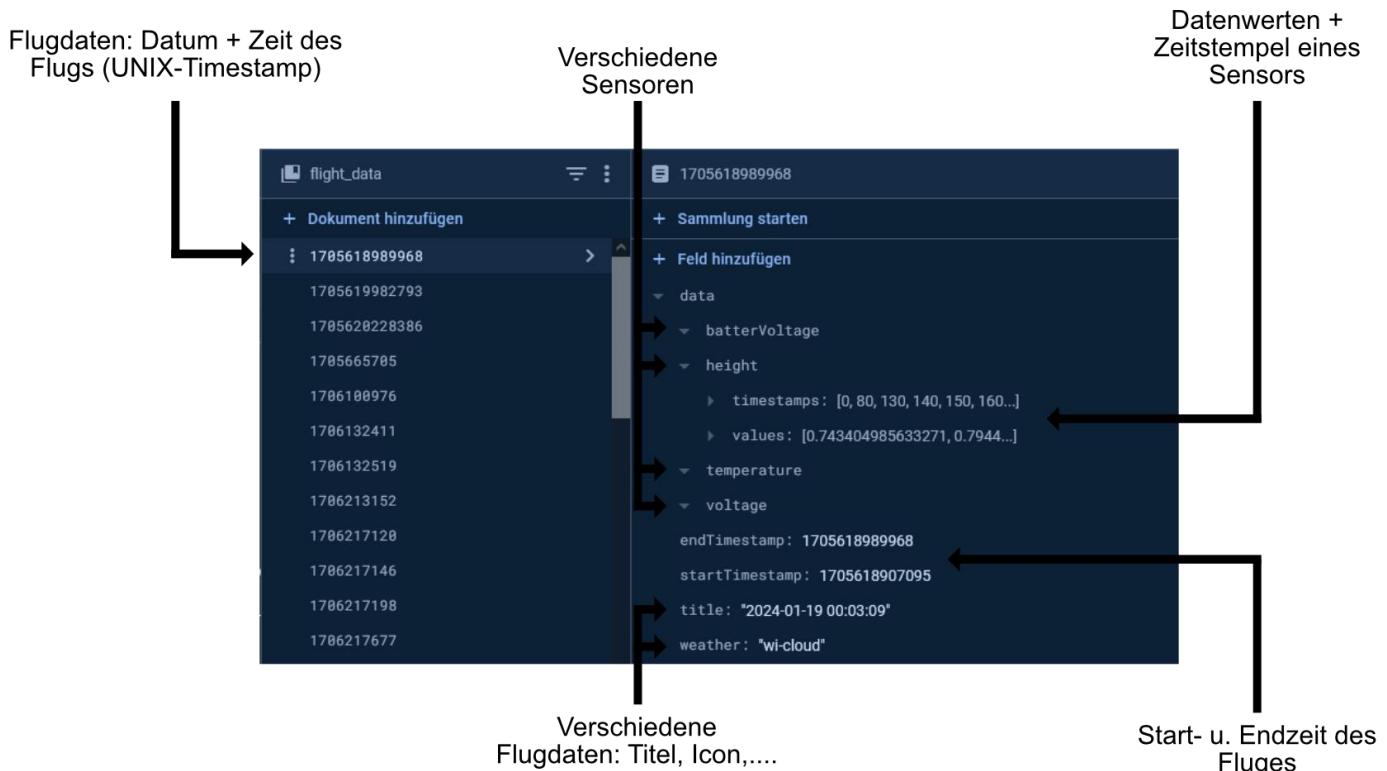


Abbildung 248: gespeicherte Flugdaten eines Users

7.2.4.3 Realtime Database

Allgemein

Obwohl Firebase Realtime Database zwar eigentlich eine Art Vorversion von Firestore darstellt, wird diese für die Visualisierungsapp benutzt. Realtime Database ist so wie Firestore eine NoSQL-Datenbank und der Nutzungszweck bezieht sich auf bestimmte Boolean Flags, die einen bestimmten Status zwischen der Groundstation und der Visualisierungsapp in Echtzeit synchronisieren können. Für diesen Zweck bietet Realtime Database eine deutlich geringere Latenz (~1s schneller als Firestore) bei der Aktualisierung von Daten.

Nutzung der Boolean-Flags

Die zwei Boolean-Flags, die benutzt werden, haben folgenden Nutzen:

- **is_online:** Wird von einem Skript auf der Ground-Station gesetzt, um zu signalisieren, dass die Übertragung der Daten gestartet hat und eine Flugaufzeichnung starten kann
- **is_connected:** Dieser Wert wird von der Visualisierungsapp gesetzt und bedeutet, dass gerade eine Flugaufzeichnung stattfindet.

Beiden dieser Werte wird in der Visualisierungsapp zugehört, um möglichst schnell auf jegliche Änderungen dieser zu reagieren. Zum Beispiel endet die Datenübertragung von der Ground-Station, so wird eine aktive Aufzeichnung eines Fluges innerhalb von maximal 1s automatisch beendet.

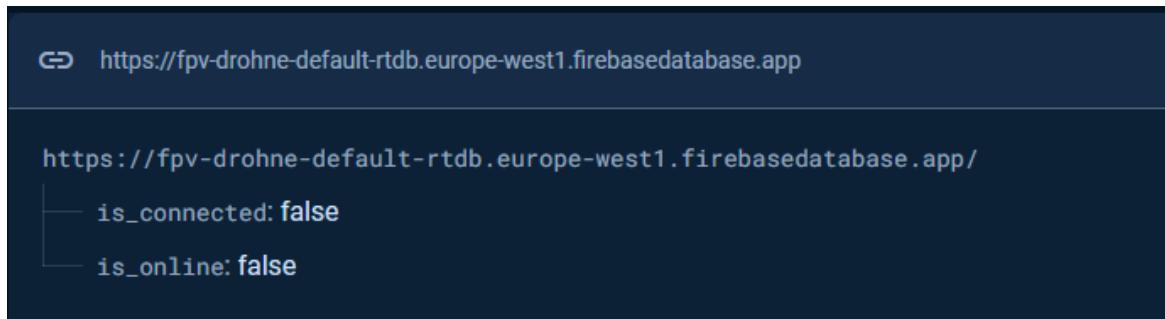


Abbildung 249: Aufgesetzte Realtime Database mit Boolean Flags

7.2.4.4 Storage

Allgemein

Der Firebase Storage ist der Cloud-Storage Service innerhalb von Firebase und erlaubt Videos, Bilder oder andere Formen von Files abzuspeichern. Die Struktur der Datenbank erinnert sehr stark an die eines typischen Betriebssystems, da es Ordner gibt in denen Files abgelegt werden.

Innerhalb der Visualisierungsapp wird der Cloud-Storage lediglich benutzt, um die Profilbilder der User abzuspeichern. Diese Bilder werden in einem Ordner namens *user_profiles* abgelegt.

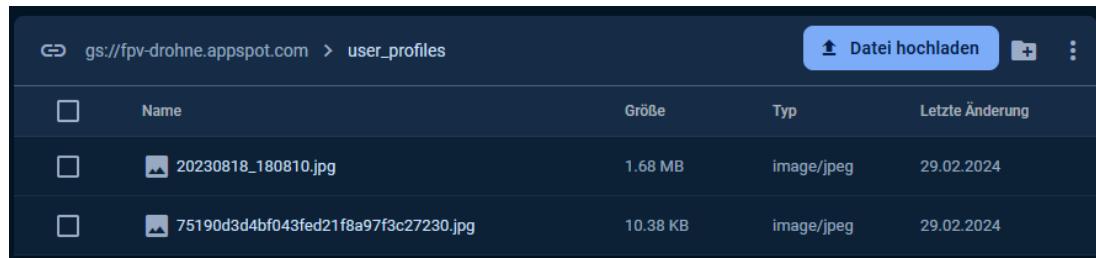


Abbildung 250: Ausschnitt des Cloud-Storage in Firebase

Referenz zu User in Firestore

Um zu wissen, welcher der abgespeicherten Bilder zu welchem User gehört, wird ein neues Profilbild automatisch mit dem Ordner und dem Filename im Firestore-Dokument des Users eingetragen. Bei Änderung des Userprofils wird das alte gelöscht und die Referenz in Firestore geupdated.

8 Videostreaming

Die Daten der Kamera auf der Drohne werden vom VTX (Videotransmitter) zur Groundstation gesendet und von einem Empfänger empfangen. Von der Ground-Station aus müssen die Bilddaten der Kamera in einen aufrufbaren Stream umgewandelt werden. Aus Gründen der Kostenersparnis, werden nur die Bilddaten verarbeitet und der Audiokanal zur Übertragung anderer Daten verwendet. Somit war kein weiter Transmitter und Receiver für die Datenübertragung notwendig.

8.1 Allgemeiner Aufbau

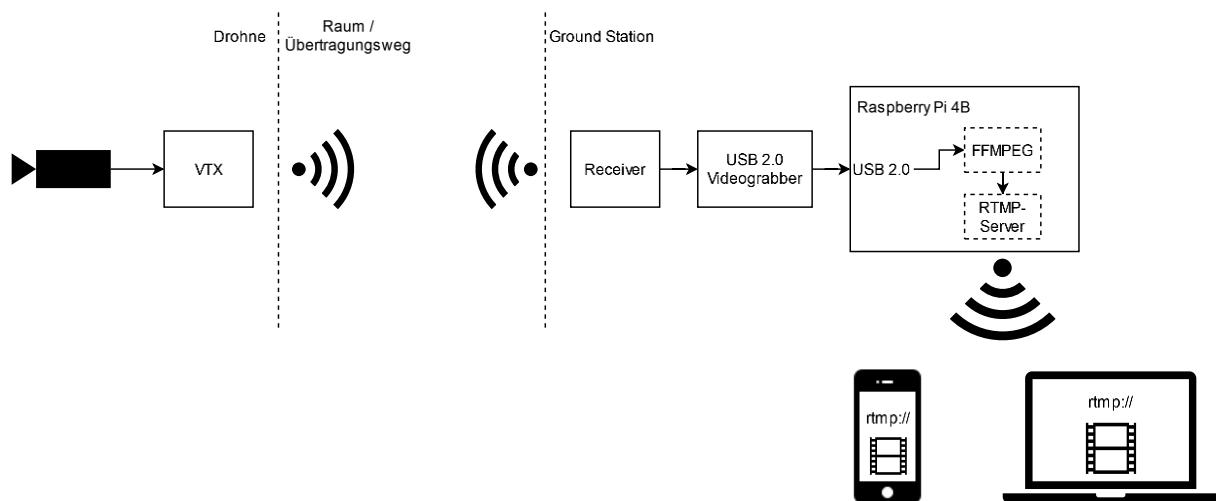


Abbildung 251: Blockschaltbild Videostreaming

8.1.1 Anforderungen

- Aufrufbarer Videostream der Kamera auf Endgeräten im gleichen Netzwerk
- Möglichst geringe Latenz
- Stabiler Videofeed
- Kein Verlust an Videoqualität (Bildrate, Bildqualität)

8.2 CADDXFPV Analog Kamera

Die Kamera, die für die Aufnahme der Livebilder verwendet wird, ist eine analoge Kamera von CADDXFPV.



Abbildung 252: Bild der CADDXFPV Ratel 2 analog Kamera [RATE]

8.2.1 Produktinformationen

Sensor	1/1,8"
Resolution	1200TVL
FOV	165°
Linse	2.1mm
TV-System	NTSC & PAL (umstellbar -> PAL ist Standard)
Videoformat	CVBS
Eingangsspannung	4.5-36V
Gewicht	5.9g

[CAXF]

8.2.2 Verwendungsgrund

Die Kamera wird verwendet, um ein Livevideo auf der Drohne aufzunehmen. Der Grund für die Benutzung einer analogen Kamera ist einerseits der Preisfaktor, VTX für Analoge Videodaten ist deutlich billiger, das leichte Gewicht mit gerade einmal 5.9g und die vermeintlich einfachere Übertragung mit nur einem Transmitter und Receiver.

8.2.3 Verbindungstest via USB-Camera App

Eine Applikation, die für den ursprünglichen Test der Videokamera und der Übertragungskette empfohlen wird, ist die USB-Camera App. Diese kann nach einer einfachen Suche im Google-Play-Store für Android-Geräte heruntergeladen werden kann.

8.2.3.1 Verbindungsinitialisierung

Um das Video der Kamera über die App wiederzugeben, muss erstens die Kamera und die VTX aufgebaut werden, um schließlich die Videokamera über ein Mikro-USB zu USB-C Kabel mit dem Handy zu verbinden. Wurde dieser Aufbau korrekt getätigigt, so sollte ein Dialog am unteren Rand des Bildschirms auftauchen, wo nach den Zugriffsberechtigungen der USB-Camera App gefragt wird.

8.2.3.2 Verbindungstest

Wurde dem Zugriffsdialog der USB-Camera-App zugestimmt, öffnet sich die Applikation automatisch und das übertragene Bild des angesteckten Receivers sollte angezeigt werden.

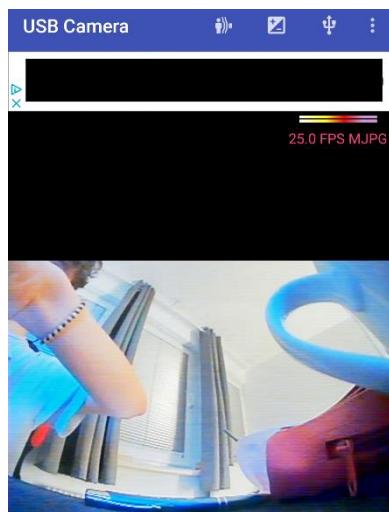


Abbildung 253: Verbindungstest mit USB-Camera App

8.3 USB2.0 VHS Video Grabber

8.3.1 Produktinformationen

Der USB2.0 Videograbber ist eine Hardware, die es möglich macht, die analogen Videodaten einer VHS-Kamera oder jeglicher anderen analogen Kamera als digitale Daten via einen USB2.0 Port auszugeben.



Abbildung 254: Bild des USB2.0 Videograbbers [CAPU]

Hinweis: gelbes Kabel: CVBS (Video); rotes u. weißes Kabel: Audio;
schwarze Buchse: S-Video

In unserem Fall benutzen wir die lokalisierte Version dieser Hardware der Firma „CSL“. Viele dieser Video-Converter haben denselben Videochip von „Syntek Semiconductor Co. Ltd“ namens „STK1160“, der die Videodaten für einen USB-A Port konvertiert.

Je nach Vertreiber dieses Videograbbers, bekommt man diesen zwischen einem Preis von 15 bis maximal 25€ (Stand: Ende 2023).

Da diese Hardware nicht implizit für ein Echtzeitanwendung wie in unserem Fall vermarktet wird, ist die Latenz nicht innerhalb der Datenblätter gegeben und muss später bei den ersten Funktionstests geprüft werden (siehe: [Kapitel 8.3.4.2](#)).

8.3.1.1 lsusb

Diese Herkunft des Geräts zeigt sich eindeutig, sobald man die Produktinformationen dieser Hardware über das Terminal mit folgendem Linux-Kommando ausgeben lässt.

```
lsusb
```

Dieses Kommando gibt alle Information der verbundenen USB-Geräte am Computer aus. Ist der Videograbber korrekt verbunden, so sollte am Terminal folgende Information angezeigt werden.

```
fpv@raspberrypi:~ $ lsusb
Bus 002 Device 001: ID 1d6b:0003 Linux Foundation 3.0 root hub
Bus 001 Device 004: ID 258a:1006 Gaming KB Gaming KB
Bus 001 Device 003: ID 046d:c53f Logitech, Inc. USB Receiver
Bus 001 Device 005: ID 05e1:0408 Syntek Semiconductor Co., Ltd STK1160
Video Capture Device
Bus 001 Device 002: ID 2109:3431 VIA Labs, Inc. Hub
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
```

Dieser Befehl gibt Auskunft über alle verbundenen Geräte an den Bluetooth Schnittstellen. Der Videograbber ist in der oberen Abbildung fett hervorgehoben und unterstrichen.

8.3.1.2 Formatinformation von Treiber

Um Informationen bezüglich des Videoformats und anderen Eigenschaften zu erfahren kann man folgenden Befehl im Terminal ausführen, insofern V4L2 (Video 4 Linux, Version 2) vorhanden ist. V4L2 ist eine vorinstallierte Kollektion an Treibern für Videogeräte auf Linux Betriebssystemen:

```
v4l2-ctl --all -d /dev/video0
```

Dieser Befehl nutzt eine Programmierschnittstelle für Video in Linux namens V4L (Video4Linux) um Informationen eines angeschlossenen Videogeräts an einer Schnittstelle zu bekommen. Ist der Videograbber das einzige Videogerät und somit an der Schnittstelle /dev/video0, so wird folgende Information ausgegeben:

```
fpv@raspberrypi:~ $ v4l2-ctl --all -d /dev/video0
Driver Info:
  Driver name      : stk1160
  Card type        : stk1160
  Bus info         : usb-0000:01:00.0-1.1
  Driver version   : 6.1.63
  Capabilities    : 0x85200001
    Video Capture
    Read/Write
    Streaming
    Extended Pix Format
    Device Capabilities
  Device Caps     : 0x05200001
    Video Capture
    Read/Write
    Streaming
    Extended Pix Format
Priority: 2
Video input : 0 (Composite0: ok)
Video Standard = 0x00001000
```

```

NTSC-M
Format Video Capture:
  Width/Height      : 720/480
  Pixel Format     : 'UYVY' (UYVY 4:2:2)
  Field            : Interlaced
  Bytes per Line   : 1440
  Size Image       : 691200
  Colorspace        : SMPTE 170M
  Transfer Function : Default (maps to Rec. 709)
  YCbCr/HSV Encoding: Default (maps to ITU-R 601)
  Quantization     : Default (maps to Limited Range)
  Flags             :

Streaming Parameters Video Capture:
  Frames per second: 29.970 (30000/1001)
  Read buffers      : 2

User Controls

  brightness 0x00980900 (int)      : min=0 max=255 step=1
default=128 value=128 flags=slider
  contrast 0x00980901 (int)       : min=0 max=127 step=1
default=64 value=64 flags=slider
  saturation 0x00980902 (int)    : min=0 max=127 step=1
default=64 value=64 flags=slider
  hue 0x00980903 (int)          : min=-128 max=127
step=1 default=0 value=0 flags=slider
  chroma_agc 0x0098091d (bool)   : default=1 value=1
flags=update
  chroma_gain 0x00980924 (int)    : min=0 max=127 step=1
default=40 value=15 flags=inactive, slider, volatile

```

Die relevantesten Eigenschaften, die für unseren Anwendungszweck aus diesem Output gelesen werden können, lauten wie folgt:

- **Chipsatz:** stk1160
- **Videoeigenschaften:**
 - **Videoformat:** NTSC-M
 - **Höhe / Breite:** 720/480
 - **Pixelformat:** 'UYVY' (UYVY 4:2:2)
 - **Bildgröße:** 691200 Bytes
 - **Bildrate:** 29.970 (30000/1001) Bilder pro Sekunde (FPS)
- **Steuerungseinstellungen („User Controls“):** Helligkeit, Kontrast, Sättigung, Farbton, Chroma AGC und Chroma-Gain (Aspekte des Bildes die vom User adaptierbar sind)

Hinweis: Obwohl das ausgegebene Videoformat hier als NTSC-M definiert wird, wurden bei der späteren Videotranskodierung deutlich bessere Eigenschaften (z.B. Farbvideo) mit bestimmten PAL-Standardformaten erreicht

8.3.1.3 Vorteile eines gleichen Chipsatzes

Da in allen der Vertriebenen Videograbber mit gleichem Aussehen, das gleiche Innenleben beinhaltet ist (insbesondere durch den Digitalisierungschip STK1160), sind alle Postings und Informationen im Internet von anderen Vertreibern auf unseren Videograbber anwendbar. Dies erlaubt es uns die geringe Menge an Informationen durch unseren Vertreiber (CSL) umgehen zu können.

8.3.2 Problem der Digitalisierung

Eines der Hauptprobleme bei der Übertragung des Videos war die Digitalisierung der Videodaten um diese als Stream mit irgendeiner Art von Computer verarbeiten zu können. Nach Recherche kam dann ein VHS zu USB2.0-Adapter auf, der als Input ein analoges CVBS-Signal oder alternativ auch ein S-Video Signal aufnehmen kann und dieses digital über eine USB2.0 Schnittstelle ausgibt, wovon es sich dann an die meisten handelsüblichen Computer anstecken lässt.

8.3.3 Treiberkompatibilität

Ein Gerät wie dieses basiert meistens auf der Tatsache, dass es eine Art von Unterstützung für ein bestimmtes Betriebssystem oder Hardware im Allgemeinen in Form eines Treibers oder Ähnlichem vorliegt.

Offiziell wird der oben genannten Videograbber nur von den letzten Windows Versionen (Vista, 7, 8, 10, 11) offiziell unterstützt. Auf einem Forum von Ubuntu Enthusiasten wird jedoch beschrieben, dass diese Kombination von Videograbber und Chipsatz ganz einfach mit der Kernel Schnittstelle „v4l2“ (Video4Linux2) funktioniert.

[VGWI]

8.3.4 Testen des USB-Grabbers

8.3.4.1 Honestech VHS to DVD 2.0 SE

Die empfohlene Software des Herstellers, um die Funktion des Videograbbers zu testen wurde zuerst das Video direkt mit der empfohlenen Software durchgeführt. Diese Software hat den Namen „Honestech VHS to DVD 2.0 SE“ und bietet dem User die Tools analoge Videodaten von alten Kameras oder Videorekordern zu digitalisieren, zu schneiden oder zu brennen. Zusätzlich lassen sich die Videodaten des Videograbbers direkt abgreifen und Einstellungen durchführen, was sie ideal zum Testen macht.

Wird die Software geöffnet, so zeigt sich nach einem kurzen Ladescreen das Hauptmenü mit 3 Menüoptionen. Standardmäßig ist der Menüpunkt „1. Capture“ ausgewählt, wo man eine Vorschau der ausgewählten Quelle anschauen kann.

Konfiguration

Um das Video der angesteckten Kamera sehen zu können muss der Videograbber als Eingangsquelle definiert werden. Dieser hat normalerweise den Namen „USB2.0 Grabber“ und kann in der oberen linken Seite des ersten Menüpunkts ausgewählt werden.

Um die Videoeinstellungen zu öffnen, muss man das Zahnradicon direkt neben dem Videoquellenfeld drücken. Innerhalb der Videoeinstellungen gibt es noch einen Untermenüpunkt mit der Bezeichnung „Steuerung“, unter dem erweiterte Videoeinstellungen getätigkt werden können.

Um die besten Einstellungen für ideale Videokonvertierung zu finden, probierten wir einige der Videoeinstellungen durch. Die folgenden Einstellungen wurden hier nur aufgrund ihrer simplen Menüanordnung getestet und werden für spätere Konvertierung gebraucht:

Eingangsquelle	Composite , das Videokabel der Kamera liefert ein CVBS-Signal, welches das gelbe Kabel einer Composite-Übertragung darstellt.
Image / Video-Standard	PAL/BDGHI , das Datenblatt der Kamera behauptet zwar es unterstütze sowohl NTSC als auch PAL-Videostandards und lasse sich umstellen. Jedoch zeigt die Vorschau nur ein farbiges Bild mit diesem exakten PAL-Standard, weswegen es auch übernommen wurde,

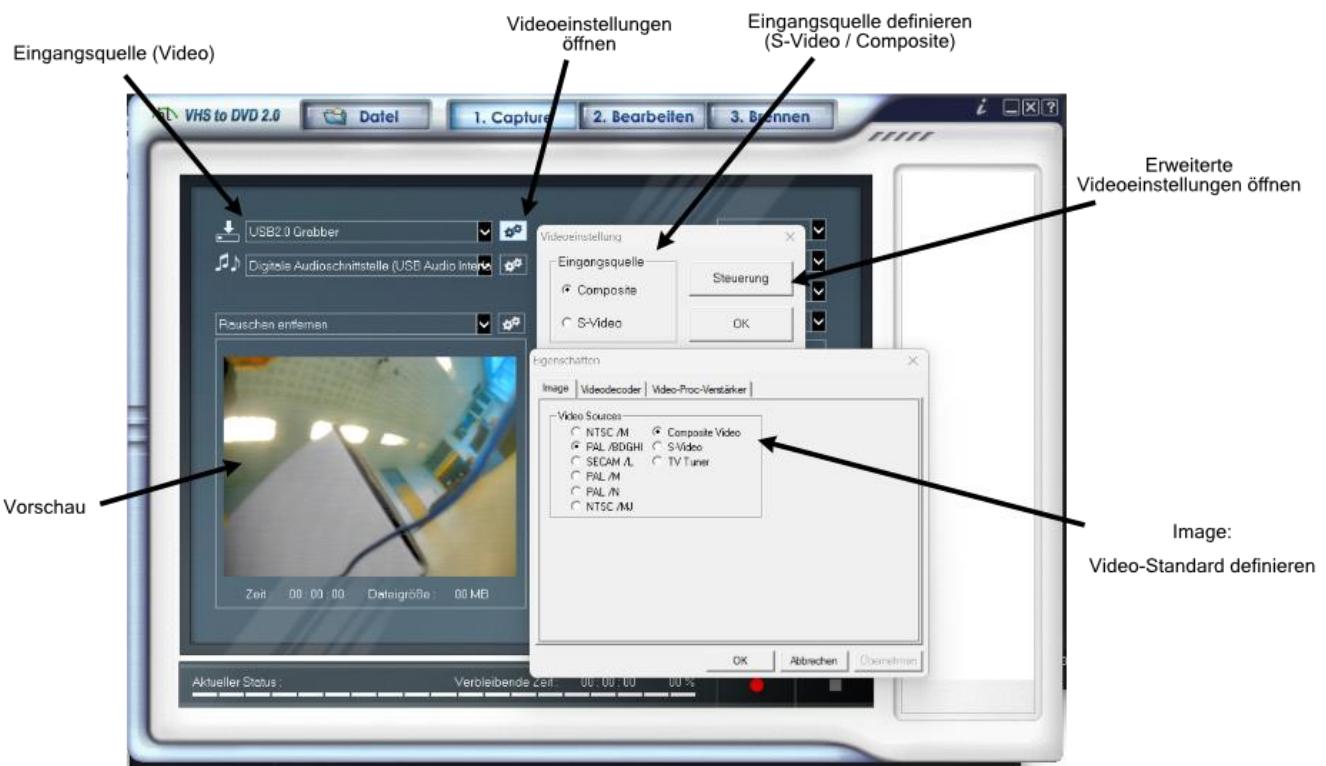


Abbildung 255: Honestech VHS to DVD 2 Software, Video-Capture Quellen und Einstellungen definieren

8.3.4.2 Latenz des Videograbbers

Wenn man davon ausgeht, dass die Übertragung der Daten über den Raum ohne jegliche Latenz stattfindet, so kann man sagen das die gesamte Latenz zum Zeitpunkt des Abgriffes der USB-Schnittstelle von dem Videograbber und dessen Konvertierung zusammenhängt. Bei dieser Messung kommt heraus, dass der Videograbber in Kombination mit der Verarbeitung des Anzeigeprogramms (Honestech, VLC-Player) eine ungefähre Latenz von einer halben Sekunde hinzufügt, was für eine Echtzeitanwendung definitiv nicht ideal, aber dennoch akzeptabel ist.

8.3.5 Berechnungen

Bevor der Konvertierungsprozess von den analogen Kameradaten zu einem digitalen Videostream startet, möchte ich kurz ein paar Berechnungen durchführen, um die einerseits

den Aufwand für diesen Prozess zu quantifizieren und auch um mögliche Probleme besser angehen zu können.

8.3.5.1 Theoretische Datenrate

Um die notwendige Datenrate zu berechnen, benötigen wir lediglich spezifische Parameter der Videodaten die der Videograbber liefert. Diese wurden schon ausgelesen (siehe: [Kapitel 8.3.1.2](#)). Mit den Angaben von:

- „rawVideo“: unkomprimiertes Video
- „720x480“: 720pixel Breite und 480pixel Höhe
- UYVY 4:2:2: Farbenformat -> jeder Pixel 2 Bytes
- USB2.0: maximale Bandbreite von ca. 480Mbps

Und der Annahme das wir einen flüssigen Stream mit 30 Bildern pro Sekunde (FPS) haben möchten lässt sich folgende benötigte Bandbreite berechnen:

$$\text{Größe eines Frames} = 720 \text{ Pixel} * 480 \text{ Pixel} * 2 \text{ Bytes pro Pixel} = 691kB \text{ pro Frame}$$

$$\frac{\text{Bytes}}{\text{Sekunde}} = \text{Größe eines Frames} * 30 \text{ FPS} = 691kB * 30FPS = 20MBps$$

$$\frac{\text{Bits}}{\text{Sekunde}} = \frac{\text{Bytes}}{\text{Sekunde}} * 8 \frac{\text{bits}}{\text{Byte}} = 165,888Mbps$$

Anhand dieser Berechnung lässt sich schließen, dass eine Datenrate von 165,888Mbps notwendig ist, um diese Daten mit 30 Bildern in der Sekunde zu verarbeiten. Somit lässt sich auch gleich eine Begrenzung durch die USB2.0 Buchse ausschließen, da diese mit 480Mbps deutlich darüber liegt. Anzumerken ist jedoch, dass bei gleichzeitiger Nutzung mehrerer Geräte an der USB2.0 Schnittstelle, die Grenze erreicht werden kann, weswegen kein weiteres Gerät parallel an der 2ten Schnittstelle der Ground-Station benutzt werden sollte.

8.4 RTMP-Server via NGINX aufsetzen

8.4.1 RTMP Allgemein

RTMP (Real Time Streaming Protocol) ist ein Streamingprotokoll, welches primär für die Übertragung von Videodaten über das Internet benutzt wird. Es ist proprietär und wurde ursprünglich von Adobe Inc. entwickelt.

Von diesem Protokoll gibt es 3 Varianten [RMWK]:

- RTMP, auf Basis von TCP/IP-Port 1935
- RTMPT, auf Basis von http
- RTMPS, auf Basis von https

In dieser Diplomarbeit haben wir uns der Einfachheit halber für die Nutzung von RTMP entschieden.

8.4.2 NGINX

8.4.2.1 Allgemein

Um die Videodaten auf anderen Geräten zugreifbar zu machen ist ein Server notwendig der diese Daten zum Client vermittelt. NGINX ist für diesen Fall geeignet, da es sich um eine weit verbreitete, sowie effiziente Open-Source-Webserver-Software handelt. Sie ist vollständig kompatibel mit Linux und kann daher auch ohne Probleme mit unserem Raspberry Pi-Server verwendet werden.



Abbildung 256: NGINX Logo [NGXL]

8.4.2.2 Installation

Zur Installation von NGINX wurde der Package Manager „apt“ im Linux Terminal mit folgendem Kommando benutzt:

```
apt install nginx -y
```

Hinweis: Wird bei einem dieser Installationsschritte ein Zugriffsfehler (z.B. Permission Denied...) ausgegeben, so kann man sich die nötigen Berechtigungen mit dem „sudo“-Keyword am Anfang eines Befehls geben. Alternativ kann man sich mit „sudo su“ auch allgemein zum „Superuser“ mit allen Berechtigungen machen

Bei einer erfolgreichen Installation schaut der Output am Terminal folgendermaßen aus:

```
Processing triggers for man-db (2.11.2-2) ...
fpv@raspberrypi:~ $ sudo apt install nginx -y
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following NEW packages will be installed:
  nginx
0 upgraded, 1 newly installed, 0 to remove and 16 not upgraded.
Need to get 0 B/494 kB of archives.
After this operation, 1,374 kB of additional disk space will be used.
Selecting previously unselected package nginx.
(Reading database ... 129672 files and directories currently installed.)
Preparing to unpack .../nginx_1.22.1-9_arm64.deb ...
Unpacking nginx (1.22.1-9) ...
Setting up nginx (1.22.1-9) ...
Processing triggers for man-db (2.11.2-2) ...
fpv@raspberrypi:~ $
```

Abbildung 257: NGINX Installation via apt

8.4.2.3 RTMP-Konfiguration

Um den Server für RTMP zu konfigurieren, muss einerseits das spezifische NGINX-RTMP-Package installiert werden und ein Konfigurationsfile von NGINX abgeändert werden.

NGINX-RTMP installieren

Da NGINX-RTMP ein Ubuntu-Standardpaket ist, lässt es sich genauso wie NGINX selbst mit „apt“ im Terminal installieren:

```
apt install libnginx-mod-rtmp -y
```

Konfigurationsfile (nginx.conf) editieren

Das Konfigurationsfile von NGINX liegt bei richtiger Installation mit dem Filename „nginx.conf“ im /etc/nginx/-Verzeichnis. Da zur Editierung dieses Files Superuser-Rechte notwendig sind, ist es entweder zu empfehlen vor Änderung des Fileinhalts die Rechte anzupassen oder das File einfach mit Superuser-Rechten via z.B. „sudo nano“ zu öffnen:

```
sudo nano /etc/nginx/nginx.conf
```

Die Änderungen, die in diesem File vorgenommen werden müssen, beziehen sich auf einen Block, welcher an einer freien Stelle im File eingefügt werden muss:

```

rtmp {
    server {
        listen 1935;
        chunk_size 4096;
        allow publish all;

        application live {
            live on;
            record off;
        }
    }
}

```

Dieser Textblock dient dazu einen der ohne jegliche Art von Filter einen gegebenen RTMP-Stream auf dem Port 1935 bereitstellt.

[RTSE]

Konfigurationen verifizieren

Um die Einträge im Konfigurationsfile zu überprüfen, existiert folgendes Kommando im Terminal:

```
nginx -t
```

Bei richtiger Konfiguration wird im Terminal folgender Text ausgegeben:

```
nginx: the configuration file /etc/nginx/nginx.conf syntax is ok
nginx: configuration file /etc/nginx/nginx.conf test is successful
```

Abbildung 258: NGINX-Serverkonfiguration überprüfen

8.4.2.4 Server starten / neustarten

Um den konfigurierten Server starten bzw. neu zu starten kann man den NGINX-Service in Linux mit folgenden Kommandos aufrufen.

Starten:

```
systemctl start nginx
```

Neustart:

```
systemctl restart nginx
```

Wurde NGINX richtig installiert, sollte der Server beim Startup des Raspberry Pis automatisch starten ohne weitere Eingriffe zu benötigen. Ein Neustart ist lediglich erforderlich, wenn Änderungen am Konfigurationsfile des Servers vorgenommen wurden.

8.4.2.5 Serverstatus einsehen

Um die Funktion des Servers zu testen kann man die TCP-Sockets anzeigen die auf dem gewählten Port laufen. Dies funktioniert mit einem standardmäßig installierten Kommandozeilentool namens „ss“, womit die Sockets des Systems analysiert werden.

```
ss -antpl | grep 1935
```

Die Flags nach Spezifizierung des Tools „ss“ filtern den Output nachfolgenden Kriterien:

a	Alle Sockets anzeigen
n	Numerische Adressen anzeigen
t	Nur TCP-Ports (RTMP)
p	Prozessinformationen anzeigen
l	Nur lauschende (listening) Sockets anzeigen

Der Zusatz „grep 1935“ dient dazu den Output des ersten Kommandos so zu filtern, dass nur die Einträge zu dem Port „1935“ ausgegeben werden. Dies ist der Port, der im Konfigurationsfile eingestellt wurde.

8.5 Videostream erzeugen

8.5.1 FFMPEG Allgemein

FFMPEG ist eine Sammlung an Software, mit der es möglich ist Audio- und Videomaterial in einer nahezu unbegrenzten Weise zu bearbeiten (Transkodieren, Overlays, Komprimierung, Stream erzeugen). Unter FFMPEG versteht man grundsätzlich 3 Kommandozeilenprogramme:

- **ffmpeg**: Zur Konvertierung, Aufnahme oder Editierung von Audio und Video
- **ffplay**: Ein simpler Medioplayer mit dem sich Files oder Netzwerkstreams wiedergeben lassen
- **ffprobe**: Ein Tool mit dem man Metadaten von vielen File- / Containerformaten anzeigen



Abbildung 259: FFmpeg Logo [FFML]

Zur Realisierung der Videoübertragung benutzen wir aber lediglich den FFMPEG-Teil um die Videodateien zu Transkodieren ([siehe: Kapitel 8.5.3](#)) [FFWK].

8.5.2 Installation

Um FFMPEG auf Linux zu installieren kann man das jeweilige Paket einfach via dem „apt“-Package Manager installieren. Dies geht im Terminal mit folgendem Kommando:

```
sudo apt install ffmpeg
```

8.5.3 Transkodierung

8.5.3.1 Allgemein

Transkodierung beschreibt den Prozess, bei dem man digitale Daten von einem Quellenformat in ein Zielformat umwandelt, meistens im Kontext von Audio- und Videoformaten. Hierbei muss Rücksicht auf die verschiedenen Codecs, Dateitypen und Kompressionsmethoden genommen werden, um die richtige Interpretation der Daten sicherzustellen. Oftmals wird dieser Prozess unternommen, um die Dateigröße anzupassen oder die Qualität dem spezifischen Nutzungszweck anzupassen. Alltägliche Transkodierungen wären zum Beispiel die Umwandlung der Audiodaten einer MP4-Datei in ein pures Audioformat mit z.B. MP3.

8.5.3.2 Ablauf der Konvertierung / Transkodierung

Die Konvertierung bzw. Transkodierung von Videodaten von einem Ausgangsformat bzw. einem Eingangsfile haben einen bestimmten Ablauf, welcher in der einfachsten Version wie folgt abläuft:



Abbildung 260: Genereller Ablauf Transkodierung FFMPEG

- **Inputfile / Schnittstelle:** Am Anfang der Kette steht ein Inputfile oder auch eine Schnittstelle, von der man die Ausgangsdaten bezieht.
- **Kodierte Datenpakete:** Diese Datenpakete entstehen nach dem Prozess des Demultiplexen und enthalten die aufgeteilten Komponenten eines Videos (Video, Audio, Untertitel). Dies dient der einzelnen Weiterverarbeitung der einzelnen Bestandteile.
- **Dekodierte Frames:** dekoderter und unkomprimierte Frame von dem ursprünglichen Frame
- **Output-File:** Das Output File wird erzeugt, nachdem die rohen Daten verarbeitet wurden und die einzelnen Bestandteile des Files (Audio, Video, Untertitel,...) je nach gewähltem Fileformat wieder enkodiert und zusammengefügt werden.

Die Prozesse, um bei einer Transkodierung eines Videos die Daten zu verarbeiten und wieder in ein fertiges File zu bringen lauten wie folgt:

- Demultiplexen: Bei diesem Verfahren wird das Inputfile in seine Bestandteile zerlegt (Audio, Video,...), um diese unabhängig voneinander bearbeiten zu können.
- Dekodierung: Die meisten Fileformate komprimieren die Inhalte eines Files. Diese komprimierten Daten können jedoch in dieser komprimierten Fassung nicht bearbeitet werden und müssen in eine unkomprimierte Rohfassung dekodiert werden.
- Encoding: Je nach definiertem Ausgangsfileformat werden die unkomprimierten Daten wieder komprimiert
- Multiplexen: Fügt die einzelnen Bestandteile wieder in ein Output File zusammen

Hinweis: Die oben genannten Abläufe beschreiben lediglich die simpelsten und allgemeinsten Abläufe bei einer Konvertierung. Gibt man Parameter zum Filtern oder andere Änderungen zu den Daten, so können weitere Prozesse wie De-Interlacing hinzukommen.

[FFDD]

8.5.3.3 Simple Konvertierung

Um die Videodaten des Videograbbers zu einem RTMP-Stream zu konvertieren müssen theoretisch nur wenige Parameter angegeben werden:

```
ffmpeg -i /dev/video0 -c:v libx264 -f flv rtmp://<Server-IP-  
Adresse>/Streamkey
```

- **ffmpeg**: Muss lediglich am Start des Kommandos stehen, um anzugeben, dass mit diesem Kommandozeilentool gearbeitet wird
- **-i /dev/video0**: Definiert die erste Video-Schnittstelle des Linux-Servers als Inputfiles. Da nur ein Videogerät an den Raspberry Pi angesteckt wird, kann man davon ausgehen das die erste Schnittstelle (./video0) immer der Videograbber ist.
- **-c:v libx264**: Gibt das Videocodec an womit spezifisch das Video enkodiert werden soll. „libx264“ ist die populärste und auch empfohlene Lösung als Codec für einen Livestream
- **-f flv rtmp://<Server-IP-Adresse>/Streamkey**:
 - *flv*: Das Format in dem die Daten am Ende enkodiert werden
 - *rtmp://<Server-IP-Adresse>/Streamkey*: Die Ziel-URL zu der das enkodierte Video gesendet wird. Die IP-Adresse muss mit der des Servers / des Computers übereinstimmen. Der „Streamkey“ am Ende kann frei gewählt werden und dient lediglich, um eine Mehrzahls an Videostreams zu unterscheiden

Mit diesen Einstellungen wird zwar ein Livestream erfolgreich erzeugt, aber dieser ist nicht ansatzweise ideal (hohe Latenz [10-50s], Instabilität, Schwarz-Weiß, niedrige Bildrate, schlechte Bildqualität).

8.5.3.4 Optimierungen

Damit die Videoübertragung verbessert werden kann muss man nun eine Vielzahl an Optimierungen an bei der Konvertierung vornehmen. Bei der Auswahl dieser Optimierungsoptionen beruhete ich mich auf die Erfahrungen anderer User, aber auch auf die eigene Testung der dokumentierten Optionen. [FFMP] [FFMC]

Messverfahren

Wir führten Tests durch, um die Transkodierung mit FFMPEG zu vergleichen. Dieses Verfahren wurde einerseits bei dem Vergleich der verschiedenen Codecs als auch beim finalen Test der Optimierungseinstellungen verwendet. Die verwendeten Vergleichsparameter lauten wie folgt:

- **Ladezeit**: Wie lange benötigt die Transkodierung, bis ein schaubarer Stream generiert wurde
- **Bildrate (FPS)**: Wie flüssig läuft das Video bzw. wie schnell funktioniert die Konvertierung (max. 25 FPS der Videoquelle) (ohne Hintergrundprozesse)

- **Bildqualität:** Schärfe, Klarheit und Farbgenauigkeit des Videostreams
- **Stabilität:** Kommt es zu abstürzen (Ja/Nein/Wie häufig)?
- **Latenzstabilität:** Bleibt die Latenz vom Start des Livestreams konstant oder ändert sie sich? Messung ~1min
- **Latenz:** Wie hoch ist die Latenz zwischen der Echtzeit (z.B. Bewegung, oder Stoppuhr) und dem Livestream

Die Latenz und Ladezeit wurden mithilfe eines Vergleichs zwischen zwei gestarteten Uhren im Video und in der echten Welt gemessen. Die anderen Parameter wurden hauptsächlich durch die subjektive Beobachtung verglichen und konnten nicht quantifiziert werden

Vergleich der Videocodecs

Um die Effizienz und Qualität der verschiedenen Videocodecs zu vergleichen, testete ich alle verfügbaren Codecs, die innerhalb von FFmpeg verfügbar waren. Um diese am Terminal auszugeben, gibt es den folgenden Befehl:

```
ffmpeg -encoders
```

Die folgende Liste, die ausgegeben wurde, unterteilt sich in die verschiedenen Arten von Codecs (Video, Audio,...). Für meine Tests waren jedoch nur die Videocodecs von Bedeutung. Diese werden am Anfang der Definition mit einem V gekennzeichnet. Nach langer Prüfung der meisten Videocodecs in dieser Liste mit der USB2.0-Schnittstelle des Videograbbers als Input und FLV bzw. RTMP als Ausgangsformat kam ich zu folgenden Ergebnissen:

Codec-Bezeichnung	Ladezeit	FPS	Bildqualität	Stabilität	Latenz
h263	rot	gelb	gelb	gelb	gelb
libx264	gelb	gelb	grün	gelb	rot
h261	lila	lila	lila	lila	lila
h264	gelb	gelb	rot	gelb	rot
flashsv	grün	gelb	gelb	gelb	grün
libxvid	rot	rot	rot	rot	rot
flashsv2	gelb	rot	gelb	gelb	gelb
flv	gelb	gelb	rot	gelb	gelb
Weitere...	Fehler oder nicht geladen				

Fehler	Nicht geladen	Schrecklich	Akzeptabel	Gut	Ideal
jeglicher Error	Wartezeit >3min	Unbenutzbar, aber funktioniert	Noch verwendbar	Offensichtliche Fehler aber in Ordnung	Im Kontext ideal (<2s)

Wenn man sich die oberen Ergebnisse anschaut, muss man beachten, dass es sich hier um einen lokalen Stream handelt. Das bedeutet, dass die oberen Ergebnisse nur die Geschwindigkeit der Transkodierung widerspiegeln und nicht die Geschwindigkeit des Netzwerks. Somit kann ein gutes Ergebnis zwar eine schnelle Transkodierung der Frames

bedeuten, aber die Komprimierung der einzelnen Frames und benötigte Bitrate bei einer Netzwerkübertragung wurde komplett missachtet.

Bitrate und Videogröße der Codecs

Um die Größe bzw. Komprimierung der nutzbaren Codecs miteinander zu vergleichen, lassen sich bei einer Übertragung die Bitrate und Größe eines konvertierten Frames vom Terminal ablesen:

```
frame= 4139 fps= 25 q=-0.0 size= 10212kB time=00:02:48.68 bitrate= 495.9kbits/
frame= 4152 fps= 25 q=-0.0 size= 10238kB time=00:02:49.20 bitrate= 495.7kbits/
frame= 4164 fps= 25 q=-0.0 size= 10266kB time=00:02:49.72 bitrate= 495.5kbits/
frame= 4177 fps= 25 q=-0.0 size= 10303kB time=00:02:50.20 bitrate= 495.9kbits/
```

Abbildung 261: Kennwerte während einer Transkodierung mit FFmpeg

Vergleicht man alle Videocodecs, die mit FLV und RTMP kompatibel sind, so ergeben sich folgende Bitraten:

Codec	Bitrate (unbegrenzt)	Bitratenvergleich zu libx264
libx264	1-3Mbps	-
flashsv	>100Mbps	~30-100x
flashsv2	20-35Mbps	~10x
flv	1-3Mbps	-

Hinweis: Das „unbegrenzt“ bezieht sich lediglich darauf, dass die Transkodierung nicht künstlich eingeschränkt wird durch eine vorgegebene Bitrate, was die Qualität deutlich mindert.

Testergebnisse der Codecs

Beim Vergleich der Codecs zeigt sich eine problematische Situation in Hinsicht zur Nutzung der Groundstation mit einem Raspberry Pi 4 als Videoserver. Die Videocodecs, welche am häufigsten empfohlen werden („libx264“, „h264“,...), haben oft eine relativ hohe Komprimierung der Videodaten und benötigen daher auch mehr Leistung bei der Transkodierung, die der Raspberry Pi 4 einfach nicht hat. Im Vergleich ein Format wie „flashsv“, welches bei der Testung via eines lokalen Streams die besten allgemeinen Ergebnisse lieferte, erzeugt Bilder, die deutlich schlechter komprimiert sind, und daher auch unrealistische Bitraten für das Streaming eines Videos mit geringer Auflösung benötigen. Um dies in Perspektive zu setzen wird für das flüssige Streaming eines FULL-HD Streams auf modernen Videoplattformen lediglich eine Bitrate von 5Mbps empfohlen, was deutlich geringer im Vergleich zu den ~80Mbps von „flashsv“ bei einer deutlich geringeren Auflösung von gerade einmal 720x576p ist.

Abschließend lässt sich sagen, dass die Hauptprobleme des Videostreamings bei der langsamen Geschwindigkeit der Groundstation liegt. Aufgrund der Tatsache, dass nicht einmal eine hardwarebeschleunigte Transkodierung möglich ist, wegen der Abwesenheit der dafür notwendigen Hardware.

Weitere Optimierungseinstellungen

FFMPEG hat aufgrund seiner riesigen Bandbreite an Optionen auch eine große Anzahl an Einstellungen und Flags, die genutzt werden können. Diese sind zwar teilweise auf der offiziellen Website beschrieben, aber viele sind auch etwas versteckter und eher schlechter dokumentiert. Die hier beschriebenen Flags zu jedem Teil der Videokonvertierung stellt in keiner Weise die idealen Einstellungen dar, sondern sind lediglich Versuche mit den besten Empfehlungen und Tests die besten Ergebnisse zu erzielen.

Einstellung	Dokumentation	Verwendete Einstellung
<code>-c:v <Videocodec></code>	Definiert „flashsv“ als Videocodec	flashsv
<code>-b:v <Bitrate></code>	Einstellung der Bitrate	50kbps-1000kbps
<code>-fflags <fileflag></code>	Definiert wie FFMPEG die Daten des Inputstreams handelt. (Bei Livestreaming / Echtzeitanwendungen wird „nobuffer“ empfohlen)	nobuffer od
<code>-standard <Videostandard></code>	Definiert den Videostandard und setzt dazugehörige Einstellungen	pal
<code>-preset <veryslow-ver, slower, slow, fast, ultrafast,...></code>	Encoding-Preset wird definiert. Von <i>veryslow</i> bis <i>ultrafast</i>	slow (schnellere Varianten führen zu schlechteren Ergebnissen)
<code>-live <true / false></code>	Input wird definiert als Livestream	true
<code>-tune <setting></code>	Latenz wird versucht möglichst gering zu halten	zerolatency
<code>-r <fps></code>	Setzt die Bildrate (FPS) für den Output auf den gegebenen Wert	10
<code>-an</code>	Blockiert alle Audiostreams vor dem Output	-an

Hinweis: Alle verfügbaren Optionen sind auf der offiziellen Dokumentationsseite oder können auch mit dem „FFMPEG -h <Encoder/Codec>“ am Terminal angeschaut werden [FFMP] weitere Einstellungen sind in FFMPEG selbst oder über [GGFP] einsichtbar.

Messungen der Videocodecs

Um die verschiedenen Videocodecs miteinander zu vergleichen, benutze ich einerseits die Angaben von FFmpeg während der Transkodierung (Bitrate, Größe), aber auch eine Stoppuhr auf einem unabhängigen Smartphone, welches nur als Vergleichswert zwischen den Werten auf den verschiedenen Streams gilt.

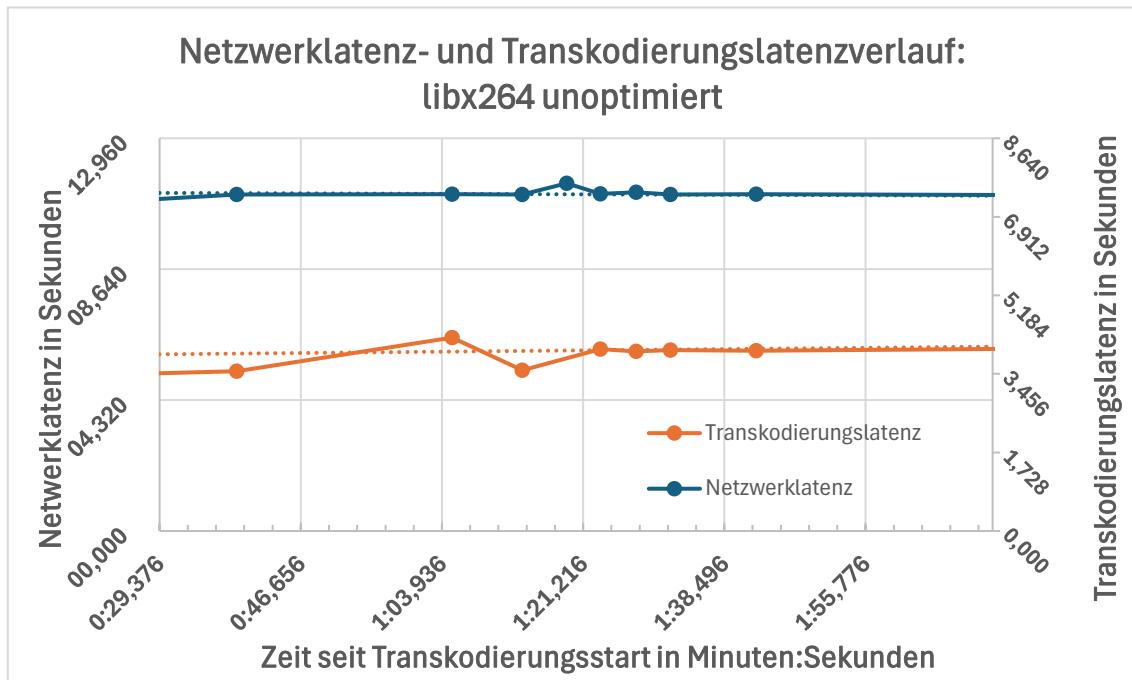
Zeit auf Stoppuhr ... tr
Zeit auf Serverstream ... t, s
Zeit auf Appstream ... t, app
Latenz (Serverstream) ... l, server
Latenz (Visualisierungsapp) ... l, app
Transkodierungslatenz ... l, cod
Netzwerklatenz ... l, netz

$$l, \text{server} \approx l, \text{cod} = \text{tr} - t, s$$

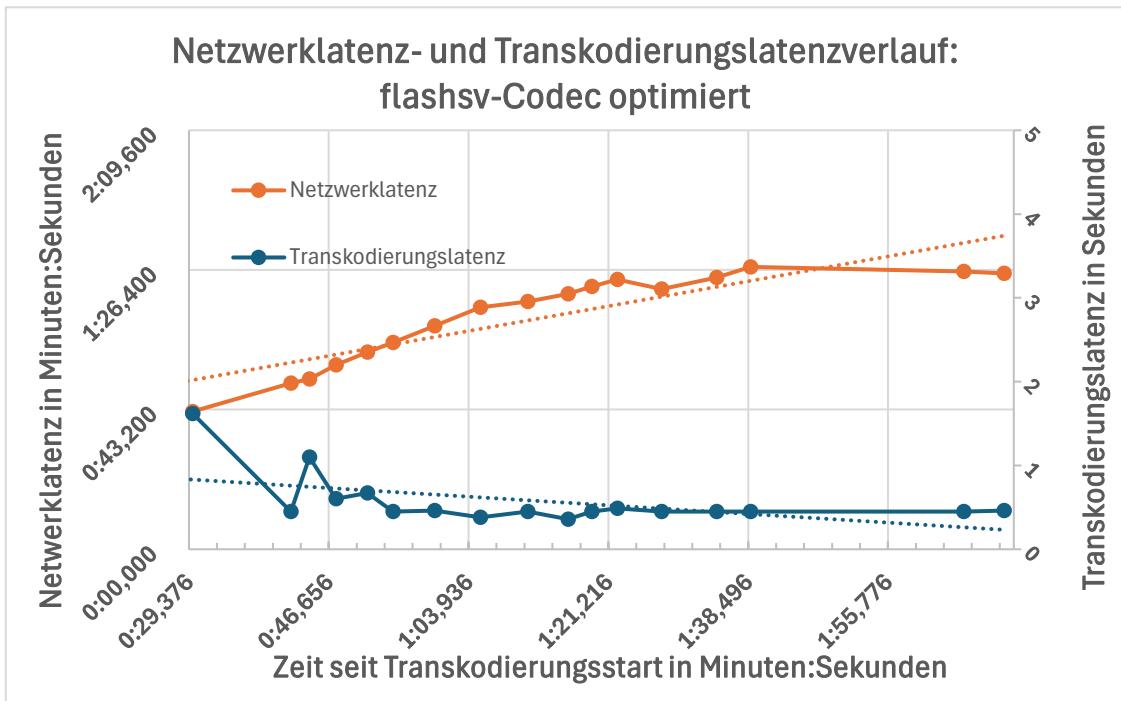
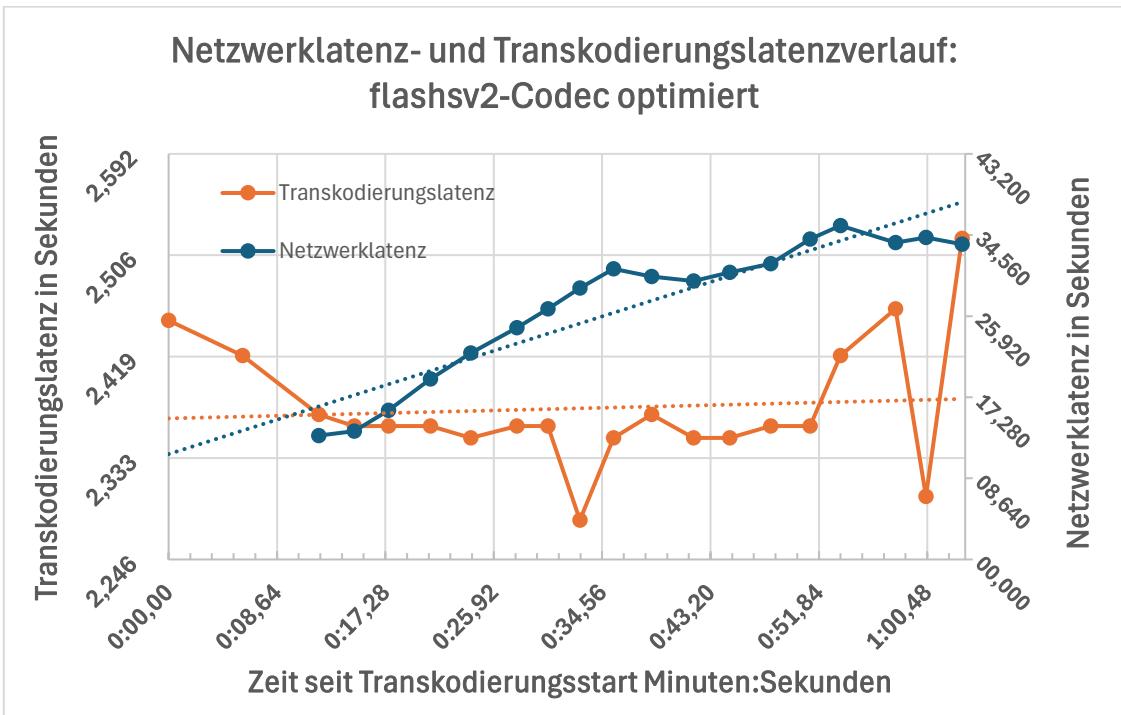
$$l, \text{app} = \text{tr} - t, \text{app}$$

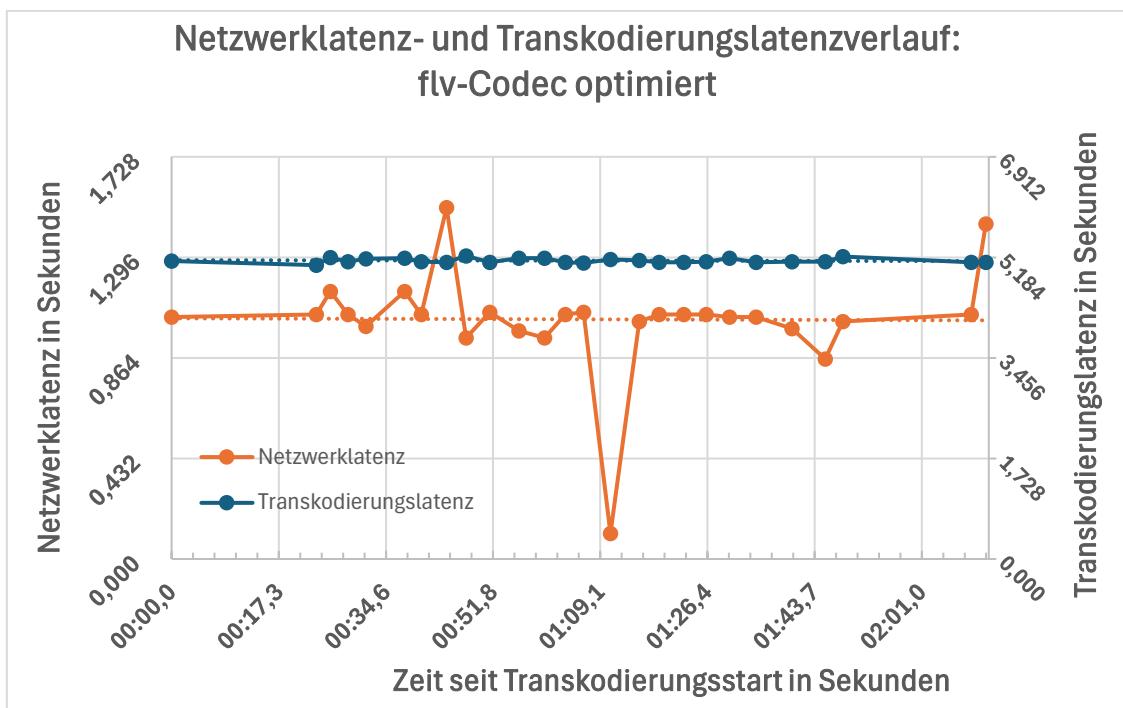
$$l, \text{netz} = l, \text{server} - l, \text{app}$$

libx264 ohne Optimierungen:



Hinweis: Das sind die einzigen Messungen, die ohne jegliche Form von Optimierungsflags durchgeführt wurden

flashsv:**flashsv2:**

flv:

Messergebnisse

Wie in den oberen Diagrammen abzulesen ist zeigt sich bei jedem der Codecs ein anderes Verhalten der Latenz auch über die Zeit. Wie zu erwarten hat libx264 zwar eine sehr hohe Transkodierungszeit (3s-5s) aber eine relativ überschaubare und stabile Netzwerklatenz, was von einer guten Komprimierung der Daten aufweist.

Anders zeigt sich das „flashsv“-Codec, was mit seiner extrem kurzen Transkodierungszeit von ~0,5s. Dieses produziert Daten, die relativ schlecht komprimiert sind, weswegen die Latenz von der Übertragung im Netz in die Höhe rast.

Ein ähnliches Verhalten zeigt sich bei „flashsv2“, mit einer niedrigeren Latenz in beiden gemessenen Kategorien. Auch wenn es hier als eine bessere Alternative zu „flashsv“ scheint, wurde es wegen seiner sehr schlechten Stabilität und Bildqualität für weitere Übertragungen ignoriert.

„flv“ scheint hier zwar der klare Gewinner zu sein mit einer guten Mischung aus relativ geringer Transkodierungs- und Netzwerklatenz, produzierte jedoch mit Abstand das schlechteste Bild mit vielen Fragmenten, sodass vor allem abrupte Bewegung schwer gehandelt werden konnten.

Zum Schluss zeigen sich „libx264“ und „flashsv“ als klare Sieger. Wobei „libx264“ durch die Geschwindigkeit der Groundstation beschränkt ist, profitiert „flashsv“ durch seine primitive Komprimierung bei der Transkodierung, was jedoch mit der benötigten Netzbänderbreite direkt wieder an Netzwerklatenz dazukommt.

Finaler FFmpeg-Befehl

Nach einer Vielzahl an Tests und Recherche bezüglich aller möglichen Optimierungseinstellungen endete die Optimierung des FFmpeg-Befehls wie folgt:

```
ffmpeg -standard pal -i /dev/video0 -codec:v flashev -preset slow -live
true -fflags nobuffer -tune zerolatency -r 10 -an -f flv rtmp://<IP-
Adresse>:1935/live/stream
```

Es ist wichtig zu betonen, dass die vorgenommenen Optimierungen keineswegs als die ideale Transkodierung darstellen. Sie basieren vielmehr auf einer Kombination aus eigenen Tests und den uns bekannten Optimierungen, die für unseren spezifischen Anwendungsfall relevant sind.

8.5.3.5 Videoserver auf Desktop-Laptop

Allgemeines + Begründung

Aufgrund der Probleme der Groundstation bei der Konvertierung von rohen Videodaten vom Videograbber in einer akzeptablen Zeit, überlegte ich zur Umgehung dieser Einschränkungen den Videoserver auf einem typischen Desktop-Laptop mit Dual-Boot Linux aufzusetzen. Auf dem Laptop sind alle benötigten Schnittstellen, insbesondere USB-A für den Videograbber vorhanden, und die Hardwarespezifikationen sollten keine grundlegenden Probleme mehr verursachen.

Hardwarespezifikationen

Um die Fähigkeiten des verwendeten Laptops festzuhalten, möchte ich hier kurz ein paar der wichtigen Hardwareinfos auflisten:

Name:	Acer Aspire 5 A515 43
Prozessor:	AMD Ryzen 5 3500U
Grafikkarte:	AMD Radeon Vega Mobile Gfx 2.10GHz
RAM:	16GB
Systemtyp:	64-Bit

Installationen

Um NGINX, FFmpeg und alle weiteren notwendigen Pakete und Services zu installieren die für das Streaming von Videodaten notwendigen sind befolgte ich die gleichen Schritte und Befehle wie auf der Groundstation (siehe: [Kapitel 8.4.2.2](#), [Kapitel 8.5.2](#)).

Hardwarebeschleunigte Transkodierung (AMF)

Da auf dem Laptop eine AMD Radeon Grafikkarte zur Verfügung steht, gibt es die Möglichkeit einen die hardwarebeschleunigten Versionen mancher Codecs auszunutzen. Dabei wird die parallele Natur von Grafikkarten ausgenutzt, um einerseits Last von der CPU zu nehmen, weniger Leistung zu verbrauchen, aber auch um eine deutlich schnellere Transkodierung durchzuführen.

Ob und was für ein hardwarebeschleunigter Videocodec benutzt werden kann, hängt von dem Hersteller und Typen der Grafikkarte ab. Für Grafikkarten vom Hersteller AMD und der Radeon-Produktserie gibt es die API-Implementierung mit dem Namen AMF. Diese ist in Form eines fertigen Codecs über „h264_amf“ verwendbar.

Um diese innerhalb von FFmpeg verwenden zu können müssen folgende Blöcke in einen Befehl eingefügt werden:

- **-vaapi_device /dev/dri/renderD128**: Gibt das VA-API-Gerät an, das für die Hardwarebeschleunigung verwendet werden soll
- **-vf 'format=nv12,hwupload'**: Definiert mit „nv12“ des verwendete Format und mit „hwupload“, dass der Filter hardwarebeschleunigt umgesetzt werden soll.
- **-c:v h264_vaapi**: Definiert das hardwarebeschleunigte Codec

Eine Sache, die jedoch zu beachten ist, sind die Beschränkung von AMF. In unserem Anwendungsfall ist lediglich die ausstehende Implementierung von Filtern via AMF ein Problem, was bedeutet, dass das Video in keiner Weise skaliert oder in anderer Weise bearbeitet werden kann.

Vergleich libx264 & h264_vaapi

Um die schnellere Transkodierungsgeschwindigkeit zwischen einer hardwarebeschleunigten und einer klassischen Transkodierung zu messen, transkodierte ich eine typische mp4-Datei mit FULL-HD Videodaten. Mit den exakt gleichen Einstellungen schaffte die klassische Methode max. 130FPS und die beschleunigte um die 180FPS, was eine Steigerung von ~25% darstellt. Wichtig zu erwähnen ist, dass bei der Nutzung von stärkeren GPUs deutlich höhere Werte erreicht werden können, die um ein Vielfaches schneller sind.

Streaming über HTTP (DASH)

Eine weitere Komponente von modernem Streaming die durch die neuere Hardware des Laptops möglich ist, sind http-basierte Streamingprotokolle wie DASH (Dynamic Adaptive Streaming over HTTP) von MPEG oder HLS (HTTP Live Streaming) ursprünglich von Apple. Diese zwei Kommunikationsprotokolle verfolgen so ziemlich das gleiche Prinzip, und zwar das Streaming von Bilddaten über HTTP und der Aufteilung eines Videostreams in viele kleine Segmente bzw. Fragmente.

Dieser Prozess hat jedoch sowie Vor- als auch Nachteile. Einerseits ist Streaming über HTTP deutlich moderner als Streaming über reines RTMP und aufgrund des Aufbaus wird die Bitrate dynamisch je nach Netzqualität angepasst, aber andererseits führt dieser extra Konvertierungsschritt in die einzelnen Segmente zu einer zusätzlichen Latenz im Bereich von 1-3s.

Aufsetzen von DASH mit NGINX

Um DASH innerhalb von NGINX aufzusetzen, müssen ähnlich wie beim Aufsetzen von RTMP ein paar Zeilen im Konfigurationsfile (/etc/nginx/nginx.conf) hinzugefügt bzw. abgeändert werden (siehe: [Kapitel 8.4.2.3](#)).

Finaler FFmpeg-Befehl Laptop Desktop

Aufgrund der grundsätzlich anderen Voraussetzungen des Laptops wird auch von anderen Codecs und Optionen nutzen gemacht. Der finale Befehl, der auf dem Laptop zur Erzeugung des Videostreams erzeugt wird, schaut daher wie folgt aus:

```
ffmpeg -vaapi_device /dev/dri/renderD128 -re -i /dev/video2 -vf
'format=nv12,hwupload' -bufsize 10M -b:v 500k -c:v h264_vaapi -c:a aac -
tune zero_latency -live true -preset fast -f flv
rtmp://192.168.8.111:1935/live/bbb
```

Hinweis: Auf dem Laptop ist die Schnittstelle des ersten externen Videogeräts anscheinend immer /dev/video2, da die ersten zwei Schnittstellen bereits reserviert sind und von anderer interner Hardware benutzt wird.

Fazit

Allgemein funktioniert der Videoserver über den Laptop nahezu fehlerfrei und um weiten besser als mit einem Raspberry Pi 4. Durch die Vervielfachung der Geschwindigkeit, war es möglich moderne Codecs inklusive hardwarebeschleunigte Codecs zu nutzen, welche die Transkodierung der Daten des Videograbbers mit relativer Leichtigkeit handeln können.

Das Streaming über DASH bietet zwar eine gute Latenz, wenn auch langsamer als reines RTMP, dynamische Bitraten und ein modernes Fundament mit HTTP, jedoch ist diese Variante noch immer weit weg von einem wahren Stream in Echtzeit (<0,5s). Deswegen wäre es vielleicht die allerbeste Option gewesen WebRTC versuchen aufzusetzen. WebRTC ist ähnlich zu den anderen zweien HTTP-Streamingprotokollen, fokussiert sich aber grundsätzlich auf die kleinstmögliche Latenz, auch wenn die Videoqualität darunter leidet.

Zuallerletzt lässt sich sagen, dass zwischen einer hardwarebeschleunigten und einer traditionellen mit CPU (libx264), kein Unterschied mehr zu sehen ist, da die Transkodierung der Videos ohne komplizierte Filter über beide Methoden schnell genug läuft. Der größte Unterschied findet sich in der Bitrate bzw. der verfügbaren Bandbreite des Netzwerks. Um das Video mit einer schönen Qualität und der originalen Auflösung halbwegs ruckelfrei zu übertragen ist mindestens eine konstante Bitrate von 500kbps notwendig, welche das Netz meistens nicht hergibt, weswegen es öfters zu höheren Latenzen bis zu 10s kommt. Wird die Bitrate auf Werte wie 50kbps reduziert, so zeigt sich zwar eine recht schlechte Videoqualität, aber die Latenz beträgt weniger als eine Sekunde.

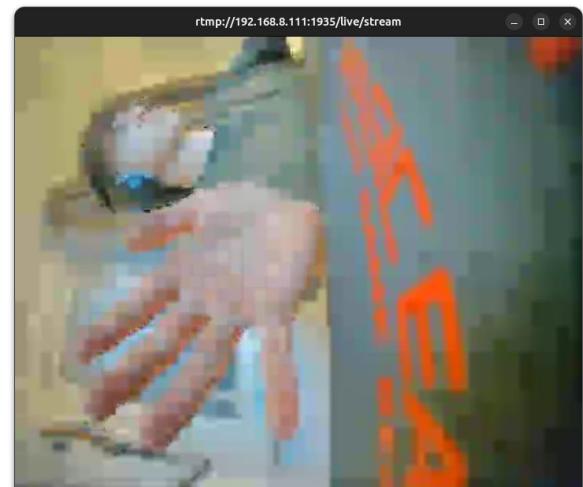


Abbildung 262: Livestreaminglatenz unter 1s mit niedriger Bitrate (50kbps)

8.5.4 Problembehebung bei Konvertierung

8.5.4.1 Problem („Broken Pipe“)

Aufgrund bisher unbekannter Gründe kommt es in unregelmäßigen Abständen zu Abstürzen bei der Konvertierung des Streams. Eine häufig auftretende Fehlermeldung in diesem Kontext ist der „Broken Pipe“-Error. Aufgrund des unregelmäßigen Auftretens konnte ein offensichtlicher Grund für diesen Fehler gefunden werden. Der einzige Zusammenhang, der bisher festgestellt werden konnte, war eine erhöhte Häufigkeit dieser Fehlermeldung bei

höherer Belastung der Groundstation durch die lokale Wiedergabe des Streams oder anderen unabhängigen Prozessen.

```
frame= 79 fps= 10 q=-0.0 size= 65089kB time=00:00:07.64 bitrate=69783.1kbits
[video4linux2,v4l2 @ 0x556f644780] Dequeued v4l2 buffer contains 827396 bytes, b
ut 829440 were expected. Flags: 0x00002001.
[video4linux2,v4l2 @ 0x556f644780] Dequeued v4l2 buffer contains 825352 bytes, b
ut 829440 were expected. Flags: 0x00002001.
[video4linux2,v4l2 @ 0x556f644780] Dequeued v4l2 buffer contains 827396 bytes, b
ut 829440 were expected. Flags: 0x00002001.
frame= 85 fps= 10 q=-0.0 size= 70592kB time=00:00:08.00 bitrate=72277.3kbits
[video4linux2,v4l2 @ 0x556f644780] Dequeued v4l2 buffer contains 825352 bytes, b
ut 829440 were expected. Flags: 0x00002001.
[video4linux2,v4l2 @ 0x556f644780] Dequeued v4l2 buffer contains 827396 bytes, b
ut 829440 were expected. Flags: 0x00002001.
[video4linux2,v4l2 @ 0x556f644780] Dequeued v4l2 buffer contains 825352 bytes, b
ut 829440 were expected. Flags: 0x00002001.
av_interleaved_write_frame(): Connection reset by peer
Error writing trailer of rtmp://192.168.0.105:1935/live/stream: Connection reset
by peer
frame= 87 fps= 10 q=-0.0 Lsize= 72578kB time=00:00:08.20 bitrate=72497.9kbit
s/s speed=0.97x
video:72584kB audio:0kB subtitle:0kB other streams:0kB global headers:0kB muxing
overhead: unknown
Error closing file rtmp://192.168.0.105:1935/live/stream: Broken pipe
Conversion failed!
fpv@raspberrypi:~ $
```

Abbildung 263: FFmpeg Transkodierung, "Broken Pipe"-Error

8.5.4.2 Problemlösung

Da der oben genannte Fehler die Konvertierung der Videodateien stoppt wurde zur Handhabung dieses Errors ein Shell-Skript geschrieben, das einen Fehler abfängt, die Konvertierung nach ein paar Sekunden automatisch wieder startet.

8.5.5 video_stream.sh

„videostream.sh“ ist ein Shell-Skript, das in Reaktion einiger unerklärlicher Fehlermeldungen im Verlauf der Erzeugung des Videostreams geschrieben wurde. Dieses Skript wird zum Start der Groundstation automatisch ausgeführt und dient dem Zweck mögliche Fehlermeldung von „FFmpeg“ abzufangen und den Stream ohne notwendige Handlungen durch die Hand eines Menschen neuzustarten.

```
Dateiname: video_stream.sh
#!/bin/bash

while true; do
    ffmpeg -err_detect ignore_err -standard pal -i /dev/video0 -pix_fmt
yuv420p -codec:v flashev -preset slow -live true -fflags nobuffer -flags
low_delay -fflags discardcorrupt -tune zerolatency -threads 1 -r 25 -an -f flv
-flvflags no_duration_filesize rtmp://192.168.0.105:1935/live/stream

# Check the exit code of ffmpeg -> if not 0 (Error), retry connection
if [ $? -eq 0 ]; then
    # If the exit code is 0 -> successfully completed -> should not happen
    break
else
    echo "Reconnecting in 3 seconds..."
fi
```

```

    sleep 3 # waiting 3 seconds before restart
fi
done

```

- **#!/bin/bash:** Dies ist die sogenannte „Shebang“-Zeile. Sie gibt an, welche Shell verwendet werden soll, um das Skript auszuführen. In diesem Fall wird die Bash-Shell verwendet.
- **while true; do:** Endlosschleife, die nur mit einem `break` beendet werden kann
- **ffmpeg ...:** Der eigentliche Befehl zur Transkodierung der Videodaten von der Schnittstelle zum RTMP-Stream
- **if [\$? -eq 0]; then ... fi:** Falls der FFMPEG-Befehl aufgrund von einem Fehler oder menschlichen Eingriff beendet wird, checkt diese Bedingung den Exit-Code (Jeder Code != 0 wird als Fehler interpretiert)
- **sleep 3:** Wartet die gegebene Zahl in Sekunden

8.5.5.1 Skript-Autostart

Damit das Skript fürs Livestreaming automatisch ausgeführt wird gibt es eine Vielzahl von Methoden. Die Methode für die wir uns entschieden haben war es das „`rc.local`“-File im `/etc`-Verzeichnis zu editieren. Die Kommandos, die in diesem Skriptfile definiert sind, werden bei jedem Bootprozess automatisch ausgeführt.

Um das Skript nun zu diesem File hinzuzufügen muss man das „`rc.local`“-File mit folgendem Befehl öffnen:

```
sudo nano /etc/rc.local
```

Hat sich File nun geöffnet muss man lediglich den Pfad zu dem Skriptfile mit einem „&“ am Ende vor dem „exit 0“-Kommando am Ende des Files hinzufügen. Das „&“ dient dazu, dass das Skript als Hintergrundprozess ausgeführt wird. In unserem Fall sieht die hinzugefügte Zeile so aus:

```
sudo /home/fpv/Documents/receiver/video_stream.sh &
```

Wird der Raspberry Pi jetzt neugestartet, so wird das spezifizierte File mit dem Skript automatisch ausgeführt.

8.5.5.1.1 Autostart des Skripts überprüfen

Um zu überprüfen, ob die automatische Ausführung unseres Skripts ohne Probleme funktioniert hat, kann man in der Prozessliste von Linux nach dem gefragten File filtern:

```
ps aux | grep video_stream.sh
```

Gab es beim Autostart keine Probleme so wird am Terminal folgendes ausgegeben:

```
fpv      2293  300  0.0  11204  4420 pts/0    R+   15:13   0.00 0s aux
fpv@raspberrypi:~ $ ps aux | grep video_stream.sh
root     1144  0.0  0.0  10676  3936 ?        S    15:05   0:00 sudo /home/fpv/Documents/receiver/video_stream.sh
root     1173  0.0  0.0  6804  3160 ?        S    15:05   0:00 /bin/bash /home/fpv/Documents/receiver/video_stream.sh
fpv      2301  0.0  0.0  6228  1968 pts/0    S+   15:13   0:00 grep --color=auto video_stream.sh
fpv@raspberrypi:~ $ ps aux | grep video_stream.sh
```

Abbildung 264: Autostart video_stream.sh Nachweis

Statische IP-Adresse zuweisen

Damit die Video- und Datenübertragung über eine fixe Adresse innerhalb des Netzes aufgerufen werden kann, haben wir uns entschieden der Ground-Station eine statische IP-Adresse zuzuweisen. Im Normalfall wird einem Gerät durch DHCP automatisch eine freie IP-Adresse im Netz zugewiesen, diese kann aber auch statisch definiert werden, sodass bei jedem Neustart ein bestimmte IP-Adresse garantiert werden kann.

DHCPD Service aktivieren:

DHCPD ist ein Client, mit dem es möglich ist mit den DHCP-Servern seines Routers zu kommunizieren. Dieser ist für Linux eine Art von Service mit einer Konfigurationsdatei. Um diesen Service zu aktivieren, muss man im Terminal folgenden Befehl eingeben:

```
sudo systemctl enable dhcpcd
```

Ist wie in unserem Fall dieser Service nicht installiert, so kommt es zu einem Error und der Service muss manuell installiert werden. Wie viele Pakete lässt sich auch dieser Teil mit einem „apt install“ installieren:

```
sudo apt-get install dhcpcd5
```

Bei einer erfolgreichen Installation muss man den ersten Befehl zum Aktivieren des Services erneut ausführen. Nun sollte dieser ohne Probleme ausgeführt werden. Eine erfolgreiche Aktivierung dieses Servers zeigt sich durch die Abwesenheit jeglicher Outputs auf die Konsole.

Um nun die statische IP-Adresse zu definieren, muss man im Konfigurationsfile mit:

```
sudo nano /etc/dhcpcd.conf
```

folgende Zeilen hinzufügen:

```
...
interface eth0
static ip_address=192.168.0.5/24
static routers=192.168.0.1
static domain_name_servers=192.168.0.1

interface wlan0
static ip_address=192.168.0.5/24
static routers=192.168.0.1
static domain_name_servers=192.168.0.1
....
```

Diese 2 Blöcke weisen den Schnittstellen mit dem Namen „eth0“ (Ethernet-Port) und „wlan0“ (Wlan-Chip) die darunter definierte IP-Adresse zu. Wurden diese Zeilen an eine beliebige Stelle im File hinzugefügt, so muss man die Änderungen speichern und das File schließen.

Test der statischen IP-Adresse

Um die Änderungen zu übernehmen kann man einfach das System neustarten und die IP-Adresse der betroffenen Netzwerkschnittstellen anzeigen lassen. Eine Option hierfür ist das Kommando „*ifconfig*“, wobei die Information für alle Netzwerkschnittstellen unter Linux ausgegeben werden, unter anderem auch die IP-Adresse.

Wird die statischen IP-Adresse erfolgreich gesetzt, und dem Raspberry Pi wird jetzt bei jedem Neustart dieselbe fixe Adresse zugewiesen.

8.6 Videoserver-Portweiterleitung

8.6.1 Allgemein

Ein Ziel beim Aufrichten des Videostreams war es, die Wiedergabe des Streams auch auf einem externen Gerät zu ermöglichen, das weit entfernt von der eigentlichen Groundstation liegt. Dafür ist jedoch eine Portweiterleitung für den Port der Videoübertragung erforderlich. Außerdem ist der handelsübliche Router, den wir für unsere Groundstation im Netzwerk verwenden, standardmäßig nicht für diesen Verwendungszweck konfiguriert und muss hierfür eingestellt werden.

8.6.2 Router

8.6.2.1 Allgemein

Um die Drohne und die damit verbundene Video- bzw. Datenübertragung testen zu können, entschieden wir uns dafür, einen handelsüblichen Router von der Firma „HoT“, genauer gesagt den ZTE M253V, in ein persönliches Netzwerk umzufunktionieren. Dies hat den Vorteil, dass wir unabhängig von Beschränkungen, insbesondere hinsichtlich der Firewall (z.B. im Schulnetzwerk) oder durch einen mobilen Hotspot (z.B. von einem Smartphone), Flugdaten sammeln können. Diese blockieren nämlich oft bestimmte Ports oder Streams, vor allem RTMP- und MQTT-Datenströme sind in solchen Netzen meist nur schwer möglich, da sie häufig blockiert werden.

8.6.2.2 Router Website

Die Webseite des Routers ist über die Gateway-IP-Adresse im Browser verfügbar, wenn man mit ihm verbunden ist. Die IP-Adresse lautet wie folgt:

192.168.0.1

Ruft man diese Webseite auf, so sieht man erstmals eine Aufforderung zur Anmeldung via Passwort. Dieses lautet bei einer standardmäßigen Konfiguration lediglich „admin“. Nach erfolgreicher Anmeldung landet man auf der Hauptseite des Routers, von wo man aus allen möglichen Einstellungen konfigurieren kann.

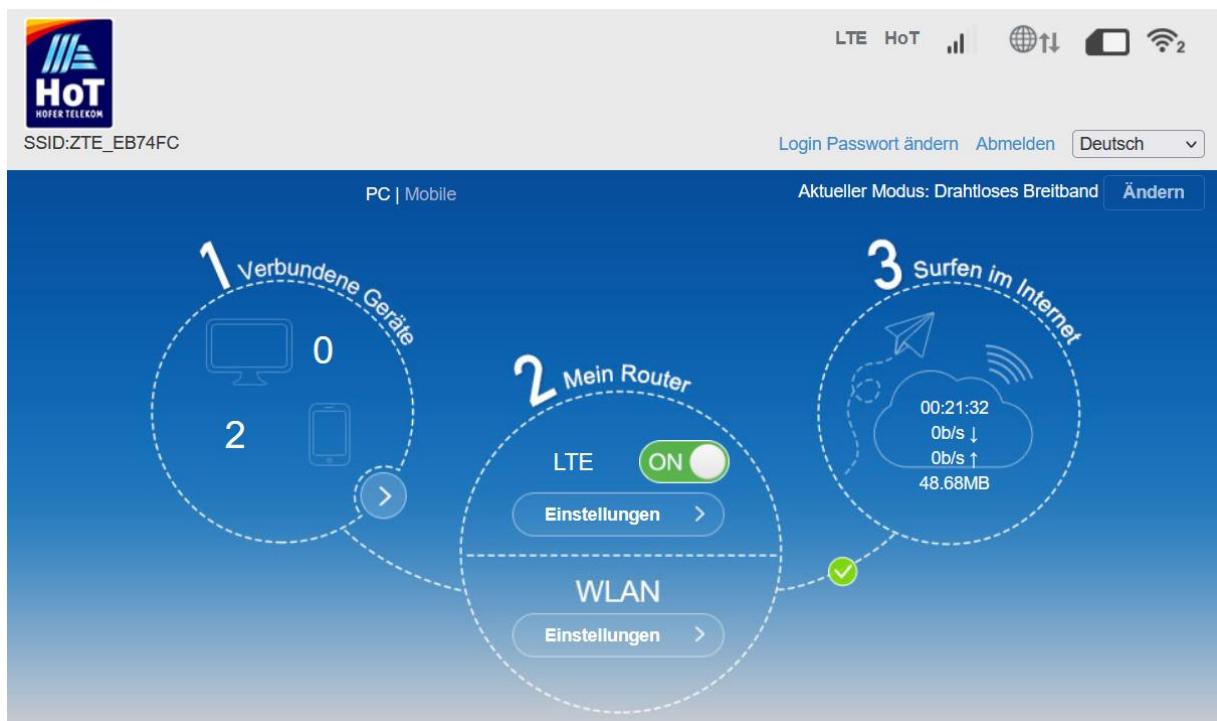


Abbildung 265: Hauptseite des Routers

8.6.2.3 Portweiterleitung

Um von der Hauptseite des Routers in die Einstellungen für die Portweiterleitung zu gelangen, muss man auf der oben gezeigten Hauptseite ganz herunterscrollen und auf die Option namens „Erweiterte Einstellungen“ klicken. Nachdem das Menü geöffnet wurde, muss man auf das Banner "Firewall" klicken und letztendlich noch auf den Punkt "Portweiterleitung".



Abbildung 266: Firewalleinstellungen öffnen

In dem nun offenen Menü lassen sich nun spezifische Ports für bestimmte IP-Adressen freischalten und bereits freigeschaltete Ports verwalten. Bei jeder Freischaltung eines Ports müssen eine IP-Adresse, ein Port, ein Protokoll und eine Bemerkung angegeben werden.

Port-Weiterleitung-Einstellungen

IP-Adresse *	192.168.0.105		(z.B. 192.168.0.101)
Portbereich *	1935	-	1935 (1-65535)
Protokoll	TCP		
Bemerkung *	RTMP-Port		
Anwenden			

Abbildung 267: Neue Portweiterleitung freischalten

Aktueller virtueller Server im System

<input type="checkbox"/>	IP-Adresse	Portbereich	Protokoll	Bemerkung
<input type="checkbox"/>	192.168.0.105	554 - 554	TCP	RaspberryPi-RTMP-Alt
<input type="checkbox"/>	192.168.0.105	1935 - 1935	TCP	RaspberryPi-RTMP-Main

Abbildung 268: Freigeschaltete RTMP-Ports für den Raspberry Pi

Die verwendete IP-Adresse „192.168.0.105“ Pis stellt die definierte statische IP-Adresse des Raspberry Pis dar. Als Protokoll wurde TCP festgelegt, da RTMP im Hintergrund auf diesem Protokoll basiert.

8.6.3 Kontakt mit Routerfirma zur Freischaltung

Aufgrund der Tatsache, dass die Portweiterleitung bei dem von uns benutzten Router nicht von Haus aus freigeschaltet ist, musste die Routerfirma (Hot) über die Kontakt-Hotline kontaktiert werden ([+43 677 6001 6770](tel:+4367760016770), 26.02.24).

Beim Anruf war es mit Übergabe der Daten des Routers und der verbundenen SIM-Karte (PUK1) möglich die Portweiterleitung für unser spezifisches Gerät freischalten zu lassen.

Zuletzt mussten nur noch die APN-Einstellungen des Routers bestimmt konfiguriert werden. Diese Konfiguration steht in einer von der Service-Hotline vermittelten PDF-Datei, sind unter anderem aber auch unter <https://cloudbox.ull.at/wiki/zte-mf253v-hot-hofer-telecom-public-ip-port-forwarding> (Letzter Aufruf: 29.02.24) beschrieben. Hierbei muss ein manuelles Profil mit den folgenden Einstellungen konfiguriert und am Ende noch übernommen werden.

Aktuelle APN	webipaut	
Modus	<input type="radio"/> Auto	<input checked="" type="radio"/> Manuell
Profil	<input type="text" value="webipaut"/> <input type="button" value="Neu hinzufügen"/>	
PDP-Typ	<input type="text" value="IPv4"/> <input type="button"/>	
Profilname *	<input type="text" value="webipaut"/>	
APN *	<input type="text" value="webipaut"/>	
Authentifizierung	<input type="text" value="PAP"/> <input type="button"/>	
Benutzername	<input type="text"/>	
Passwort	<input type="text"/>	

Um diese Einstellung zu ändern, muss die Verbindung zuerst getrennt werden.

Abbildung 269: APN-Einstellung im Router

Hinweis: Um diese APN-Einstellungen vorzunehmen muss der Schalter mit „LTE“ auf aus geschalten werden und nach Änderung der Einstellung wieder angestellt werden.

8.6.4 Testen der Portweiterleitung

Um die Freischaltung der jeweiligen Ports zu testen, benutzen wir eine externe Webseite namens „portchecker.co“ (<https://portchecker.co/>). Diese ermöglicht einem nach Eingabe eines beliebigen Ports festzustellen, ob ein externes Gerät Zugriff darauf hat bzw. ob dieser Port erfolgreich weitergeleitet wird.

The screenshot shows the "Port Checker" interface. It has fields for "Your IP Address" (86.62.33.160) and "Port Number" (1935). A "Check" button is present. Below the form, a green box displays the result: "Port 1935 is open."

Abbildung 270: Erfolgreiche Portweiterleitung auf portchecker.co

Nach all den Einstellungen kann ein externes Gerät auf unsere Ground-Station, über die freigeschalteten Ports Daten abgreifen und somit auch Video- bzw. Flugdaten empfangen.

8.7 Videostream testen

8.7.1 VLC-Player

Der VLC-Player ist ein freier und bewiesener Multimediaplayer, mit dem eine Vielzahl an Videoformaten abgespielt werden können. Abgesehen davon lassen sich auch Videostreams von einer Schnittstelle, aber auch von einem Netzwerk, wie einem RTMP- oder RTSP-Stream anzeigen, was sie ideal zum Testen des Videoservers macht.

Der VLC-Player existiert in vielen Formen auf einer Vielzahl von Geräten. Von einer Desktop-App, einer Smartphone-App und auch einem Command-Line-Tool. Zum Testen nutzen wir die offizielle App des Entwicklers, die von deren Website heruntergeladen werden kann. [VLCW]

8.7.1.1 Netzwerkstreams öffnen

Öffnet man die Desktop-App des VLC-Players, so sieht man über einer leeren Vorschau eine Menüleiste. Drückt man in dieser Leiste auf den Menüpunkt namens „Medien“, so öffnet sich ein Untermenü, wobei jede Option zum Abspielen von Medien aufgelistet ist. Drückt man hier auf den Menüpunkt „Netzwerkstream öffnen ...“, so poppt ein Fenster auf, wo sich eine beliebige URL eines Netzwerkstreams angeben lässt.

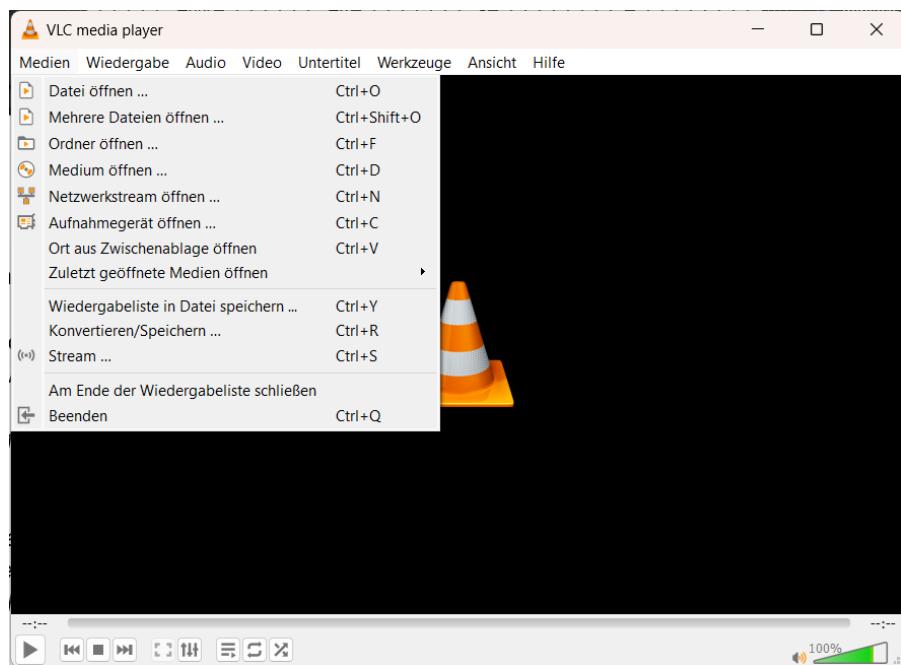


Abbildung 271: VLC-Player Hauptmenü + Medien Menüpunkt

In diesem aufgepoppten Menü kann man nun eine URL eines Videostreams angeben. VLC unterstützt eine Vielzahl von Netzwerkstreams, wie HTTP, MMS, RTP, RTSP und unter anderem auch RTMP, was in unserem Fall wichtig ist, da der Server mit RTMP aufgesetzt wurde.

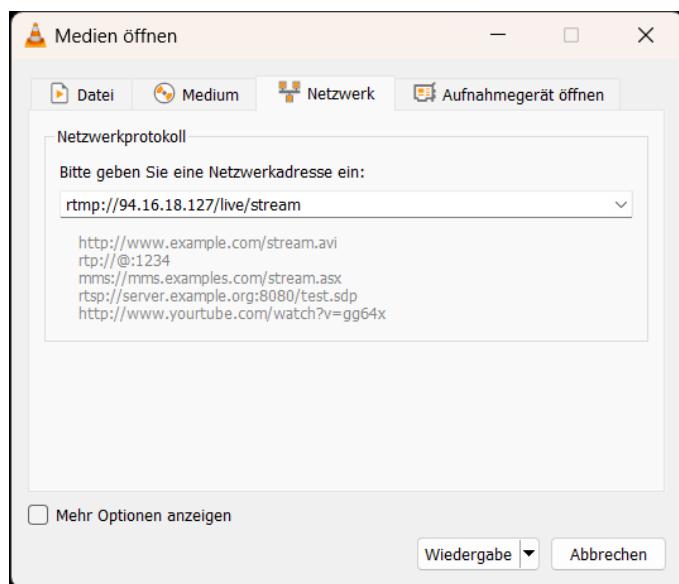


Abbildung 272: Beispiel RTMP-Netzwerkstream öffnen

8.7.2 ffplay

ffplay ist eine der 3 Hauptfunktionen von FFMPEG (siehe: [Kapitel 8.2.5.1](#)) und kann innerhalb des Terminals aufgerufen werden, um jegliche Form von File oder Videostream aufzurufen. Aufgrund der besseren Dokumentation und der schnelleren Ladezeiten, benutzte ich bei den Tests auf dem Desktop-Laptop fast ausschließlich ffplay.

8.7.3 Testverfahren

Um die folgenden Tests durchzuführen, wurde die Kamera der Drohne auf eine Stoppuhr gezeigt die stetig hochzählt. Anhand der Differenz der Stoppuhr in der echten Welt und am Livestream konnte eine ungefähre Latenz ermittelt werden. Weitere Parameter wie Bildqualität und die Stabilität des Streams wurden lediglich anhand von visuellen Faktoren wie Fragmenten, Rauschen oder Bildfehlern festgestellt.



Abbildung 273: Messverfahren Latenz zwischen Kamera und Stream

8.7.4 Groundstation als Videoserver

8.7.4.1 Test auf Videoserver (localhost)

Um die simpelste Art von Videoübertragung unseres Servers zu testen, lässt sich der Videostream direkt am Server selbst über die richtige URL wiedergeben. So lassen sich theoretisch alle Störungen durch das durch die Netzwerkqualität oder einer Firewall ausschießen.



Abbildung 274: Videostream auf Groundstation via VLC abgreifen; FLV-Codec

8.7.4.2 Test in gleichem Netz

Der nächste Schritt beim Testen beläuft sich auf das Abspielen des Videostreams auf einem Gerät, welches mit dem gleichen Netz wie der Server verbunden ist. Hierzu kann ein beliebiges Gerät benutzt werden, solange es eine Option hat einen Netzwerkstream zu öffnen (z.B. VLC-Player).

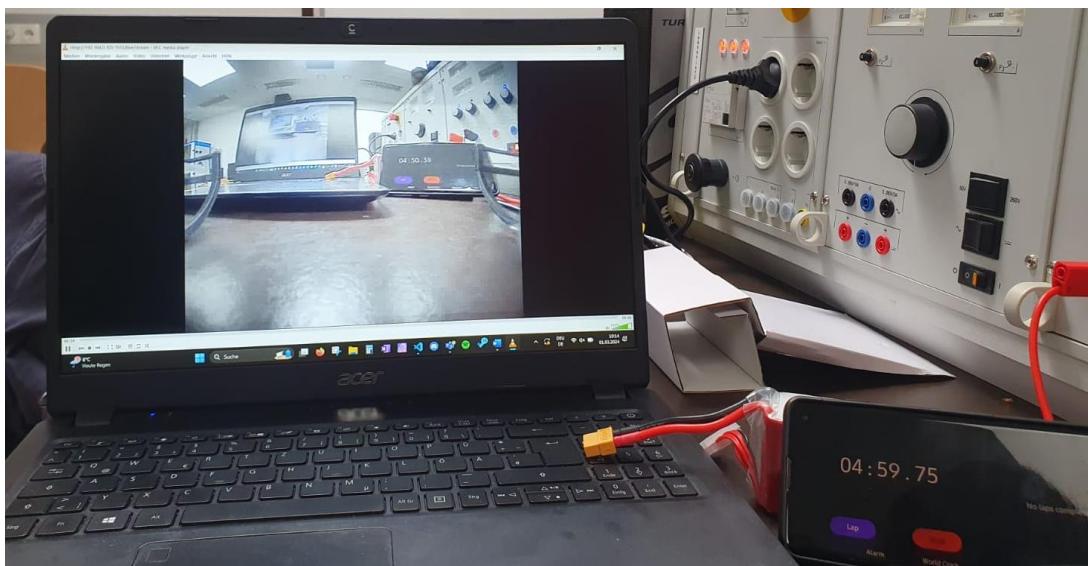


Abbildung 275: Abgreifen des RTMP-Streams von Gerät in lokales Netz + Latenzmessung

Hinweis: Mittlerweile (~2017) lassen sich RTMP- / RTSP-Streams nicht mehr im Browser wiedergeben. Zu Zeiten des Adobe-Flash-Players war dies noch möglich, da dieser RTMP als Form von Streamingprotokoll unterstützte

8.7.4.3 Test von externem Netz

Sind die spezifischen Ports des Routers freigeschalten und es wurden alle anderen Maßnahmen getroffen, um Port-Weiterleitung mit dem benutzten Router zu ermöglichen, steht nichts im Wege den gleichen Prozess zur Wiedergabe des Livestreams wie im lokalen Netz durchzuführen, nur jetzt mit der öffentlichen IP-Adresse.



Abbildung 276: Test über externes Netz; VLC-Player auf Android

Öffentliche IP-Adresse abrufen

Um die öffentliche IP-Adresse des Servers abzurufen, gibt es einige Webseiten, die diese anzeigen können. In unserem Fall benutzen wir die Webseite namens <https://whatismyipaddress.com/>, welche die öffentliche IP-Adresse gleich bei Aufruf groß anzeigt. Diese IP-Adresse ändert sich aufgrund der Natur der meisten Router nach zufälligen Intervallen, aber meist nach einem Neustart.

The screenshot shows the homepage of whatismyipaddress.com. At the top, there is a search bar with the placeholder "Enter Keywords or IP Address..." and a blue "Search" button. Below the search bar are navigation links for "ABOUT", "PRESS", "BLOG", and "SUPPORT". A horizontal menu bar includes "MY IP", "IP LOOKUP", "HIDE MY IP", "VPNS", "TOOLS", and "LEARN". The main content area displays the user's IP information: "My IP Address is: IPv4: 89.144.195.167" and "IPv6: Not detected". To the right, there is a map of Europe with a callout box highlighting Liechtenstein. The callout box contains the text "Click for more details about 89.144.195.167". Below the map, it says "Location not accurate? Update My IP Location". On the left side, there is a sidebar with "My IP Information" showing: ISP: Mobilkom Austria AG, City: Feldkirch, Region: Vorarlberg, Country: Austria. There is also a red button labeled "HIDE MY IP ADDRESS NOW" and a link "Show Complete IP Details".

Abbildung 277: öffentliche IP-Adresse herausfinden mit whatismyipaddress.com [WMIP]

8.7.4.4 Flutter-App

Innerhalb der eigens programmierten Flutter-App muss man im angemeldeten Zustand lediglich die IP-Adresse des Servers angegeben. Anschließend muss man zum Menüpunkt "Live View" wechseln, wo der Livestream angezeigt werden sollte. Je nach Netzwerkqualität und im Allgemeinen kann es jedoch zu einer kurzen Wartezeit von (5-10s) kommen bevor der Livestream geladen bzw. gebuffert hat und richtig angezeigt wird.

Zudem ist das die einzige Testmethode, die nicht auf einem PC oder einer offiziellen Software des Entwicklers, sondern auf der eigenen Android-App stattfindet.

Hinweis: Die Daten im Overlay des Videos sind lediglich Platzhalter, da zum Testen nur die Videoübertragung von der Kamera aufgebaut wurde.

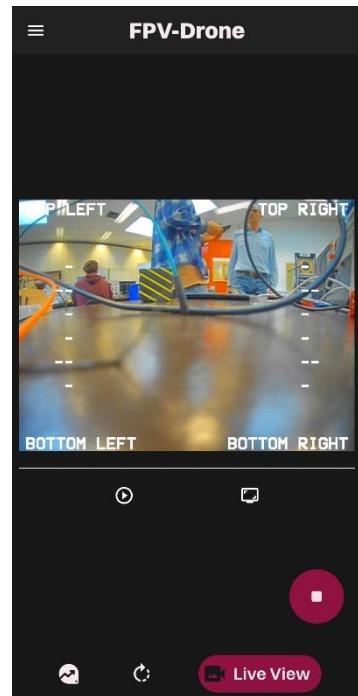


Abbildung 278: Videostream in der Visualisierungsapp mit Platzhalterwerten im Video-Overlay

8.7.4.5 Testergebnisse

Um die verschiedenen Zugriffsmethoden auf den vom Server generierten RTMP-Stream zu vergleichen wurde nach bestimmten Faktoren verglichen:

	USB-Schnittstelle	Direkt auf Server	Im gleichen Netz	Externer Zugriff	Flutter App (lokal)
Latenz	<1s	1-4s	3-7s	>1min	10-15s
Stabilität	perfekt	gut	ok	unbenutzbar	akzeptabel
Bildqualität	perfekt	sehr gut	gut	akzeptabel	gut

Anhand der Ergebnisse zeigt sich eine relative hohe Latenz, insbesondere beim Abgriff des Streams. Durch die gestiegene Latenz und den allgemeinen Verlust an Qualität, lässt sich das Problem der Videoübertragung auf die Transkodierung von FFmpeg auf der Groundstation oder durch fehlende Bandbreite im Netzwerk (siehe: [Kapitel 8.3.5.1](#)) zurückführen.

8.7.5 Laptop als Videoserver

8.7.5.1 Testen im gleichen Netz

Greift man das Video direkt vom Videoserver ab (localhost), so ist in den meisten Fällen eine Latenz von unter 2s bei stabilem Bild, und sehr guter Bildqualität möglich.

8.7.5.2 Testen in Visualisierungsapp

Mit der extra Übertragung über das Netz zeigt sich eine generelle Latenz von ungefähr 5s bei einer fehlerfreien Qualität.

8.7.5.3 Testergebnisse

Bei den Tests zeigt sich das alle der Varianten zum Mitschauen des Streams einwandfrei funktionieren, sowohl über RTMP als auch über DASH.

DASH fügt mit seiner deutlich komplexeren Struktur und Aufarbeitung der verschiedenen Fragmente eine extra Latenz im Vergleich zu rohem RTMP hinzu. Selbst bei einer guten Netzwerkqualität ist eine Übertragung der Daten selbst in einem lokalen Netz unter 2s nur selten möglich.

Die eigentliche Qualität des Bildes ist nun durch den Laptop nahezu ideal und kann jede Form von schneller Bewegung gut und mit nur kleineren Fragmenten gut mitverfolgen.

8.7.6 Videoübertragung von der Drohne

Um die Funktionalität auch auf der eigentlichen Drohne und ohne die Hilfe von irgendwelchen separaten Netzteilen zu beweisen, möchte ich zum Schluss noch einmal ein paar Bilder aus der Sicht der Drohne zeigen:

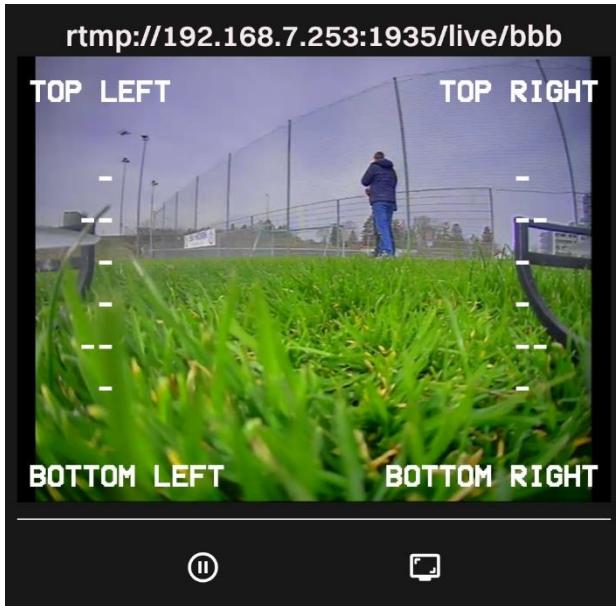


Abbildung 280: RTMP-Stream von Drohne am Boden



Abbildung 279: RTMP-Stream von Drohne in der Luft

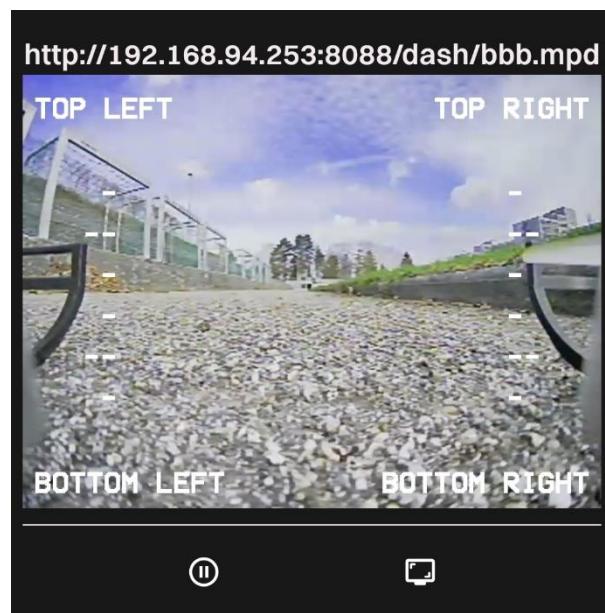


Abbildung 281: HTTP/DASH-Stream von Drohne am Boden

9 Ergebnisse

9.1 Gesamtergebnisse Elektronik

Durchgeführte Arbeiten

- Schaltung für Flight Controller Platine
- PCB Layout für Flight Controller Platine
- Bauteilbeschaffung für Flight Controller Platine
- Flight Controller Platine gefertigt
- Einzelkomponenten auf Drohne verbunden
- Datenrückgewinnungsschaltung entworfen und gebaut
- Funkübertragung aufgebaut

9.2 Gesamtergebnisse Mechanik

Durchgeführte Arbeiten

- Rotorenschutz designt und gefertigt
- Groundstation designt und gefertigt
- Drohne zusammenbauen

Nicht durchgeführte Arbeiten

- Drohne Flugtestsystem bauen (um sicher PID-Regler zu testen)

9.3 Gesamtergebnisse Embedded

Durchgeführte Arbeiten

- Akkuspannung bestimmen
- Fernsteuerungsdaten auswerten
- Verschiedene Flugmodi + Ein/Ausschalter realisiert
- Lagewinkel und Flughöhe bestimmen
- ESC-Ansteuerung mit DShot-Protokoll
- Echtzeit Flugregelung mit PID-Regler
- Kontrolle auf Verbindungsverlust
- Status Ausgabe an Terminal und LEDs
- Sendeprogramm auf Cortex M7 geschrieben
- Empfangsprogramm auf Raspberry Pi geschrieben

Nicht durchgeführte Arbeiten

- Magnetometer auslesen für verbesserte Lagewinkelbestimmung
- PID-Koeffizienten feinjustieren
- Datenübertragung mit HC-12 Funkmodul umsetzen

9.4 Gesamtergebnisse Visualisierungsapp

Durchgeführte Arbeiten

- Entwicklung einer MQTT-Klasse für den Datenempfang in Flutter
- Echtzeitvisualisierung der Flugdaten (Spannung, Temperatur, Höhe) über MQTT mit geringer Latenz (max. 1s)
- Usersystem für die Visualisierungsapp mit Abspeicherung der userspezifischen Daten in Firebase
- Aufsetzen eines Videoservers via NGINX für das Streaming über RTMP und DASH (HTTP) auf der Groundstation
- Erstellung einer übersichtlichen und interaktive UI mit dem Flutter Framework
- Darstellung von Netzwerkstreams in der Visualisierungsapp über VLC -Plugin (Latenz je nach Netzwerkqualität 1-7s)
- Aufsetzen eines MQTT-Brokers auf Groundstation

Nicht durchgeführte Arbeiten

- Zufällige und seltene Abstürze der Visualisierungsapp auf bestimmten Android-Versionen
- Darstellung der Fehlercodes in der Visualisierungsapp

10 Anhang

10.1 Verwendete Software

10.1.1 Altium Designer 22

Altium Designer ist eine Entwicklungsumgebung für integrierte Elektronik und wurde für die Erstellung von der Drohnenelektronik in Form von Schaltungen, Simulationen und Layouts verwendet.

10.1.2 Fusion 360

Fusion 360 ist ein CAD-Tool, mit dem effizient und übersichtlich 3D-Objekte erstellt werden können. Das Programm ist sehr umfassend und bietet von der Zeichnung bis hin zur Konstruktion, als auch Simulation und Rendering so gut wie alle Funktionen, die zur Erstellung von 3D-Körpern benötigt werden. Dabei ist eine übersichtliche Projektstruktur möglich.

10.1.3 Visual Studio Code

Visual Studio Code ist ein plattformübergreifender Code-Editor, der sich durch seine Benutzerfreundlichkeit und Erweiterbarkeit auszeichnet. Mit seiner intuitiven Oberfläche und zahlreichen Erweiterungen ist es eine beliebte Wahl für Entwickler weltweit.

10.1.4 STM32CubeMX

STM32CubeMX ist ein grafisches Konfigurationswerkzeug, das die Konfiguration und Initialisierung von STM32-Mikrocontroller-Projekten vereinfacht, indem es eine benutzerfreundliche Oberfläche für die Konfiguration auf Basis der STM32-Familie bietet.

10.1.5 Keil µVision5

Keil µVision5 ist eine integrierte Entwicklungsumgebung für die Programmierung von Embedded-Systemen, die eine benutzerfreundliche Plattform für das Schreiben, Kompilieren, Debuggen und Testen von Code bietet.

10.1.6 Blender

Blender ist eine Open Source 3D-Grafiksoftware für die Erstellung, Editierung und Animierung von 3D-Modellen jeglicher Art. Durch die Unterstützung der Entwickler und der wachsenden Community, hat sich Blender zu einem der Marktführer im Bereich der 3D-Bearbeitung entwickelt.

10.1.7 Affinity Photo

Affinity Photo ist eine leistungsstarke Bildbearbeitungssoftware mit einer Vielzahl an professionellen Tools und einer benutzerfreundlichen Oberfläche, die präzise Bearbeitung, sowie Erstellung, von Fotos ermöglicht.

10.2 Einführung Entwicklungsumgebung - Steuerungssoftware

Um den Mikroprozessor STM32H7A3RG6 auf der Flight-Controller Platine zu programmieren, wird die Entwicklungsumgebung Keil µVision5 in der Version V5.38.0.0 in Kombination mit STM32CubeMX in der Version 6.10.0 und Vision Studio Code verwendet. Programmiert wird dieser mithilfe eines DAPLink (CMSIS-DAP) – Interface über die SWD-Schnittstelle.

Mit STM32CubeMX können die Grundeinstellungen des Mikrocontrollers, wie die Peripherie- und Taktversorgungseinstellungen, mit einer grafischen Oberfläche einfach getätigt und automatisch eine Initialisierungssoftware in der Programmiersprache C generiert werden. Diese Software wird mithilfe von HAL (hardware abstraction layer) erstellt. Dieses System bietet eine Menge APIs, die es ermöglichen mit einfachen Funktionen komplexe Befehle und Einstellungen in einen STM32 Mikrocontroller zu tätigen.

Keil µVision5 bietet eine integrierte Entwicklungsumgebung mit eingebautem Assembler, Compiler und Debugger. Es gibt eine beschränkte kostenlose Version im Internet, die aber nicht für die Programmgröße der Diplomarbeit ausreicht. Daher muss eine kostenlose Community-Lizenz von Keil aktiviert werden [KLIZ].

Für die Programmierung des C-Sourcecodes wird Visual Studio Code verwendet. Mit der Erweiterung „Keil Assistent“ [KASS] kann Keil µVision5 direkt von einer Benutzeroberfläche gesteuert werden. Visual Studio Code bietet eine Vielfalt von Funktionen, die den Arbeitsablauf produktiver und einfacher gestalten.

STM32CubeMX Projekt erstellen:

Nach dem Starten von der Software muss der zu programmierende Mikrocontroller ausgewählt werden.

Bei vorgefertigten Entwicklungsplatinen, kann „access to board selector“ oder „access to example selector“ verwendet werden.

Für die Diplomarbeit wird eine eigenerstellte Platine mit Mikrocontroller verwendet, daher wird die Option „access to MCU selector“ verwendet.

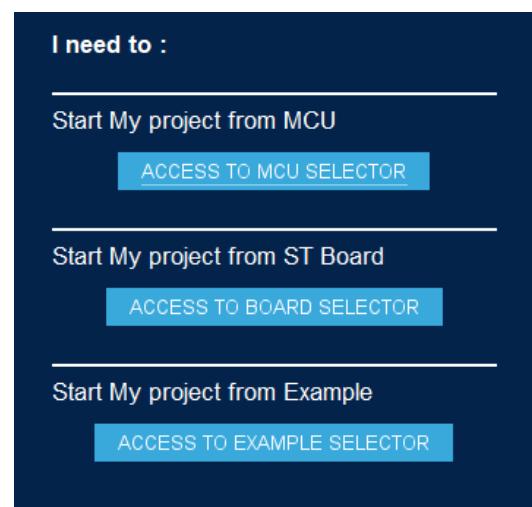


Abbildung 282: STM32CubeMX access to MCU

Nach der Auswahl werden die aktuellen Informationen der Mikrocontroller und Entwicklungsplatinen von der STM32 Datenbank runtergeladen.

MCUs/MPUs List: 3930 items

	Commercial P...	Part No	Reference	Marketing S...	Unit Price for 10k...	Board	Package	Flash	RAM	I/O	Freque...	Export
★	STM32C011D6Y...	STM32C011D6	STM32C0...	Coming soon	NA		WLCSP 12 1.7x1.42x0.6 P...	32 kBytes	6 kBBytes	10	48 MHz	
★	STM32C011D6Y...		STM32C0...	Active	0.3213		WLCSP 12 1.7x1.42x0.6 P...	32 kBytes	6 kBBytes	10	48 MHz	
★	STM32C011F4P3		STM32C0...	Coming soon	NA		TSSOP-20	16 kBBytes	6 kBBytes	18	48 MHz	
★	STM32C011F4P6		STM32C0...	Active	0.3116		TSSOP-20	16 kBBytes	6 kBBytes	18	48 MHz	
★	STM32C011F4P7		STM32C0...	Active	0.3335		TSSOP-20	16 kBBytes	6 kBBytes	18	48 MHz	
★	STM32C011F4P...	STM32C011F4	STM32C0...	Active	0.3335		TSSOP-20	16 kBBytes	6 kBBytes	18	48 MHz	
★	STM32C011F4U3		STM32C0...	Coming soon	NA		UFQFPN 20 3x3x0.6 mm	16 kBBytes	6 kBBytes	18	48 MHz	
★	STM32C011F4U...		STM32C0...	Active	0.3583		UFQFPN 20 3x3x0.6 mm	16 kBBytes	6 kBBytes	18	48 MHz	
★	STM32C011F4U...		STM32C0...	Active	0.3116		UFQFPN 20 3x3x0.6 mm	16 kBBytes	6 kBBytes	18	48 MHz	
★	STM32C011F6P3		STM32C0...	Active	0.4144		TSSOP-20	32 kBBytes	6 kBBytes	18	48 MHz	
★	STM32C011F6P...		STM32C0...	Active	0.4144		TSSOP-20	32 kBBytes	6 kBBytes	18	48 MHz	
★	STM32C011F6P6		STM32C0...	Active	0.3604		TSSOP-20	32 kBBytes	6 kBBytes	18	48 MHz	
★	STM32C011F6P...		STM32C0...	Active	0.3604		TSSOP-20	32 kBBytes	6 kBBytes	18	48 MHz	
★	STM32C011F6P7	STM32C011F6	STM32C0...	Active	0.3856		TSSOP-20	32 kBBytes	6 kBBytes	18	48 MHz	
★	STM32C011F6P...		STM32C0...	Active	0.3856		TSSOP-20	32 kBBytes	6 kBBytes	18	48 MHz	

Abbildung 283: STM32CubeMX Liste von Mikrocontroller

Mit Hilfe von Filtern und Suchoptionen kann der gewünschte Mikrocontroller ausgewählt werden. Bevor das Projekt erstellt wird, können die Eigenschaften der Auswahl kontrolliert werden. Um das Projekt zu erstellen, muss der Knopf „Start Project“ gedrückt werden.

Features Block Diagram Docs & Resources CAD Resources Datasheet Buy Start Project

STM32H7 Series

Features

- Core
 - 32-bit Arm® Cortex®-M7 core with double-precision FPU and L1 cache: 16 Kbytes of data and 16 Kbytes of instruction cache allowing to fill one cache line in a single access from the 128-bit embedded Flash memory, frequency up to 280 MHz, MPU, 599 DMIPS/ 2.14 DMIPS/MHz (Dhrystone 2.1), and DSP instructions
- Memories
 - Up to 2 Mbytes of Flash memory with read while write support, plus 1 Kbyte of OTP
 - ~1.4 Mbytes of RAM: 192 Kbytes of TCM RAM (inc. 64 Kbytes of ITCM RAM + 128 Kbytes of DTCM RAM for time critical routines), 1.18 Mbytes of user SRAM, and 4 Kbytes of SRAM in Backup domain
 - 2x Octo-SPI memory interfaces with on-the-fly decryption, I/O multiplexing and support for serial PSRAM/NOR, Hyper RAM/Flash frame formats, running up to 140 MHz in SRD mode and up to 110 MHz in DTR mode
 - Flexible external memory controller with up to 32-bit data bus:
 - SRAM, PSRAM, NOR Flash memory clocked up to 125 MHz in Synchronous mode
 - SDRAM/LPSDR SDRAM,
 - 8/16-bit NAND Flash memory

MCUs/MPUs List: 1 item

*	Commercial P...	Part No	Reference	Marketing S...	Unit Price for 10k...	Board	Package	Flash	RAM	I/O	Freque...	Export
★	STM32H7A3RG	STM32H7A3RG	STM32H7...	Active	6.1984		LQFP 64 10x10x1.4 mm	1024 kB...	1184 kB...	49	280 MHz	

Abbildung 284: STM32CubeMX Auswahl Mikrocontroller

Mit der grafischen Oberfläche können die gewünschten Peripherie-, Pin-, Takt- und Projekteinstellungen getätigt werden.

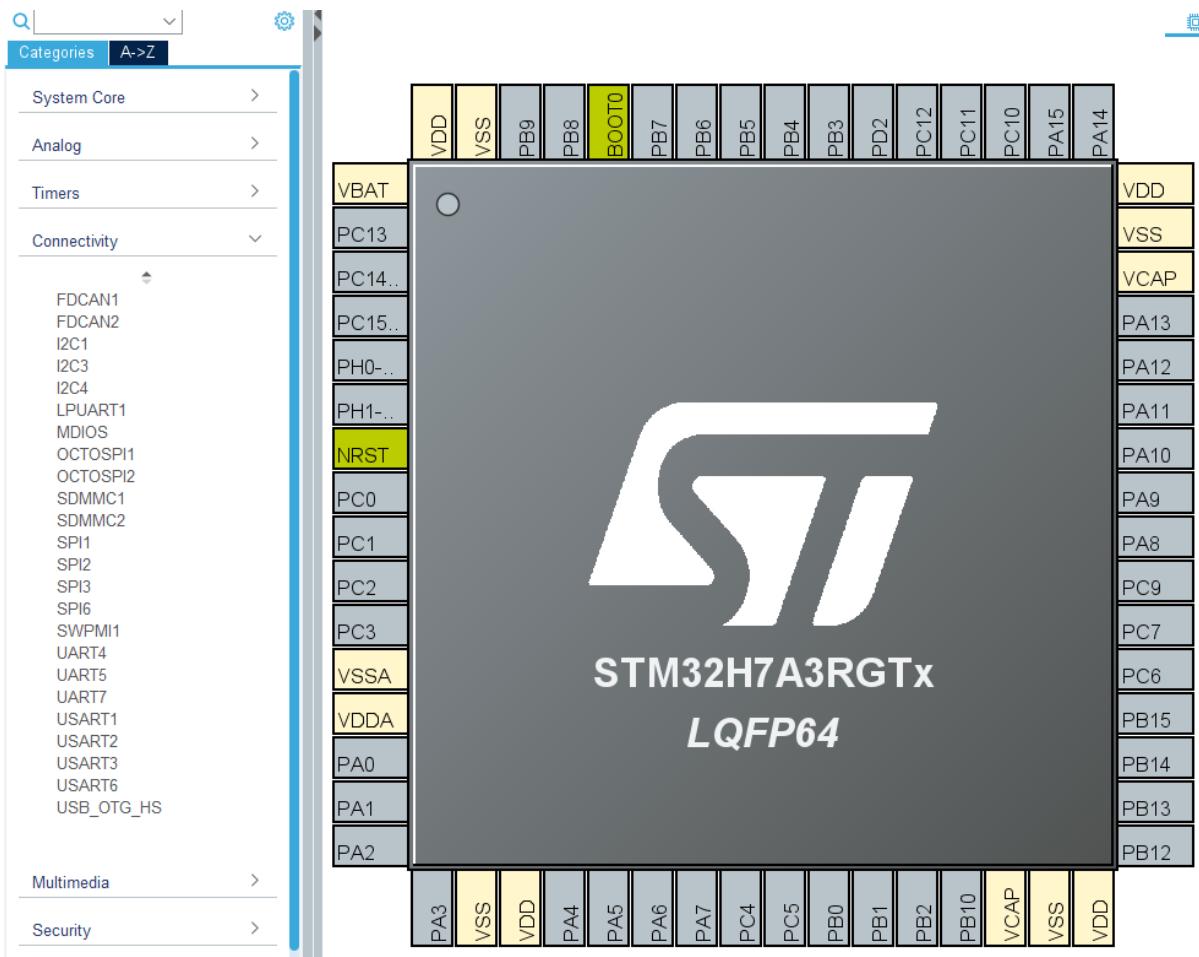


Abbildung 285: STM32CubeMX grafische Oberfläche

Um die effizienteste Programmstruktur für die Diplomarbeit zu erstellen, müssen wichtige Einstellungen getätigt werden:

Als Erstes muss die Taktquelle ausgewählt werden. Da externe Quarzoszillatoren verwendet werden, muss in der Kategorie RCC für „High Speed Clock (HSE)“ und „Low Speed Clock (LSE)“ die Einstellung „Crystal/Ceramic Resonator“ ausgewählt werden.

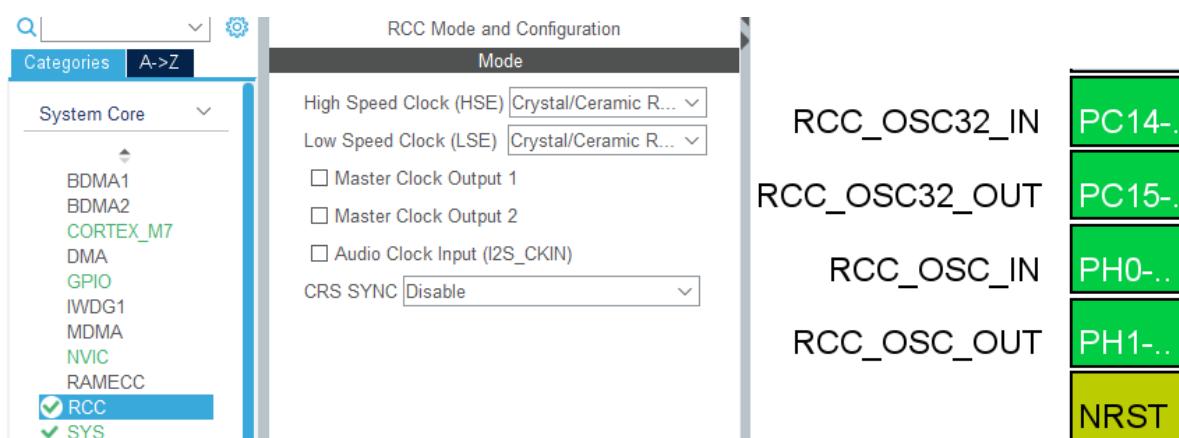


Abbildung 286: STM32CubeMX HSE/LSE Einstellung

Mit der „Clock Configuration“ kann die gesamte Taktstruktur nach Anforderungen verändert werden:

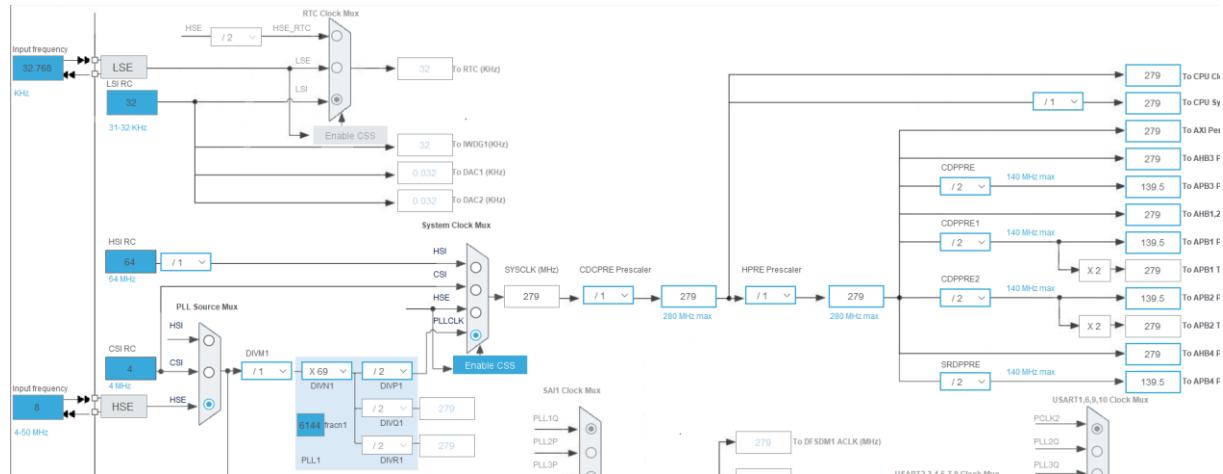


Abbildung 287: STM32CubeMX clock configuration

Wichtig: Wenn externe Oszillatoren verwendet werden, muss auf der linken Seite, die Einstellung „Input frequency“ auf die vorhandene Frequenz gesetzt werden.

Für die Diplomarbeit wird ein 8MHz Quarz Oszillator, der auf eine Systemfrequenz von 279MHz erhöht wird, verwendet.

Um den generierten Code mit der Keil µVision5 Entwicklungsumgebung verwenden zu können, müssen unter Project Manager gewisse Einstellungen getätigert werden:

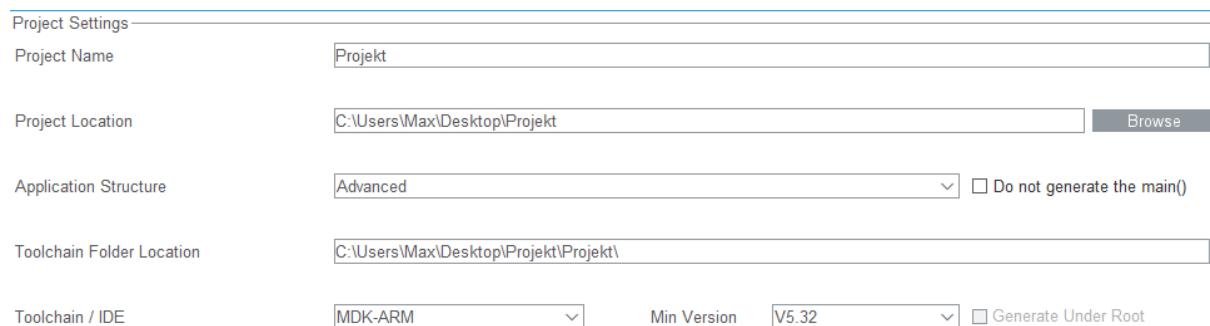


Abbildung 288: STM32CubeMX project settings

Eine für die Drohne notwendige Einstellung ist unter Project Manager → Advanced Settings → Register Callback. In diesen Bereich können für einzelne Peripherien eigenerstellte Callback-Funktionen aktiviert werden.

Für die FPV-Drohne ist die Einstellung für die TIM-, UART- und USART-Peripherie aktiviert werden.

Eine Callback-Funktion wird beim Auslösen eines Interrupts von der ISR (Interrupt Service Routine) ausgelöst.

Register CallBack	
<input type="button" value="Search (Ctrl+F)"/>	<input type="button" value="<"/>
SMBUS	DISABLE
SPI	DISABLE
SWPMI	DISABLE
TIM	ENABLE
UART	ENABLE

Abbildung 289: STM32CubeMX register callback

Zum Beispiel: `void HAL_TIM_PeriodElapsedCallback (TIM_HandleTypeDef * htim)` wird jedes Mal ausgeführt, wenn eine Timer-Peripherie einen Überlauf-Interrupt auslöst. Diese Funktion ist in der Datei `stm32h7xx_hal_tim.c` mit dem Präfix `_weak` definiert. Das bedeutet, wenn die Funktion in einer anderen Datei ausprogrammiert wird, wird diese mit der neuen Version ersetzt.

Die Einstellung von Register Callback erlaubt es, die Funktion mit einer eigenen zu ersetzen. Dafür muss die Funktion `HAL_TIM_RegisterCallback()` aufgerufen werden.

Zum Beispiel:

```
HAL_TIM_RegisterCallback(&htim15, HAL_TIM_PERIOD_ELAPSED_CB_ID, RealTimeSystemCallback);
```

Von der Timer15-Peripherie wird das Überlauf-Interrupt-Callback mit der Funktion `RealTimeSystemCallback()` ersetzt.

Nach der fertigen Einstellung des Projekts, können die Programmdateien mit dem Knopf „Generate Code“ erstellt werden. Daraufhin wird folgende Ordnerstruktur erstellt:

- Core: alle Programm- und Headerdateien
- Drivers: alle HAL-Dateien, die für das Projekt notwendig sind
- MDK-ARM: Keil µVision5 Projektdateien

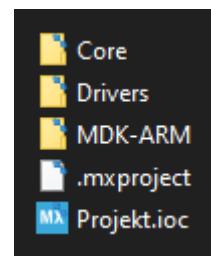


Abbildung 290: STM32CubeMX

Ordnerstruktur

Keil µVision5 Einstellungen:

Die Einstellungen des Projekts können mit dem Zauberstab „Options for Target...“ eingestellt werden.

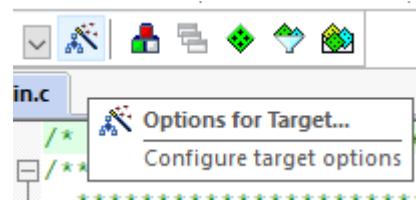


Abbildung 291: µVision Zauberstab

Die wichtigsten Einstellungen sind die Target- und Debug-Einstellungen:



Abbildung 292: µVision Target Einstellungen

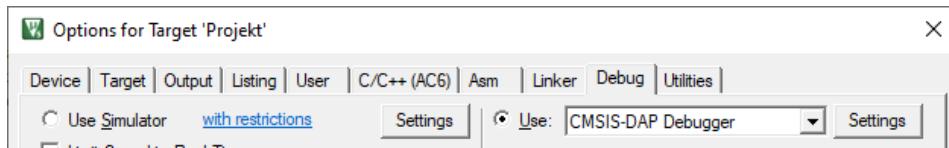


Abbildung 293: µVision Debug Einstellungen

Für die Diplomarbeit wird der default compiler version 6 und der CMSIS-Dap Debugger verwendet.

Das von STM32CubeMX erstellte Programm muss nicht geändert werden, und eigener Code soll zwischen den Kommentaren „USER CODE BEGIN“ und „USER CODE END“ eingefügt werden, damit, wenn ein neuer Code generiert wird, der eigene nicht überschrieben wird.

Um das Programm auf den Mikrocontroller zu spielen, müssen folgende Knöpfe verwendet werden:



Abbildung 294: µVision
Build/Flash Knöpfe

Auf der linken Seite kann das Projekt mit virtuellen Ordnern strukturiert werden.

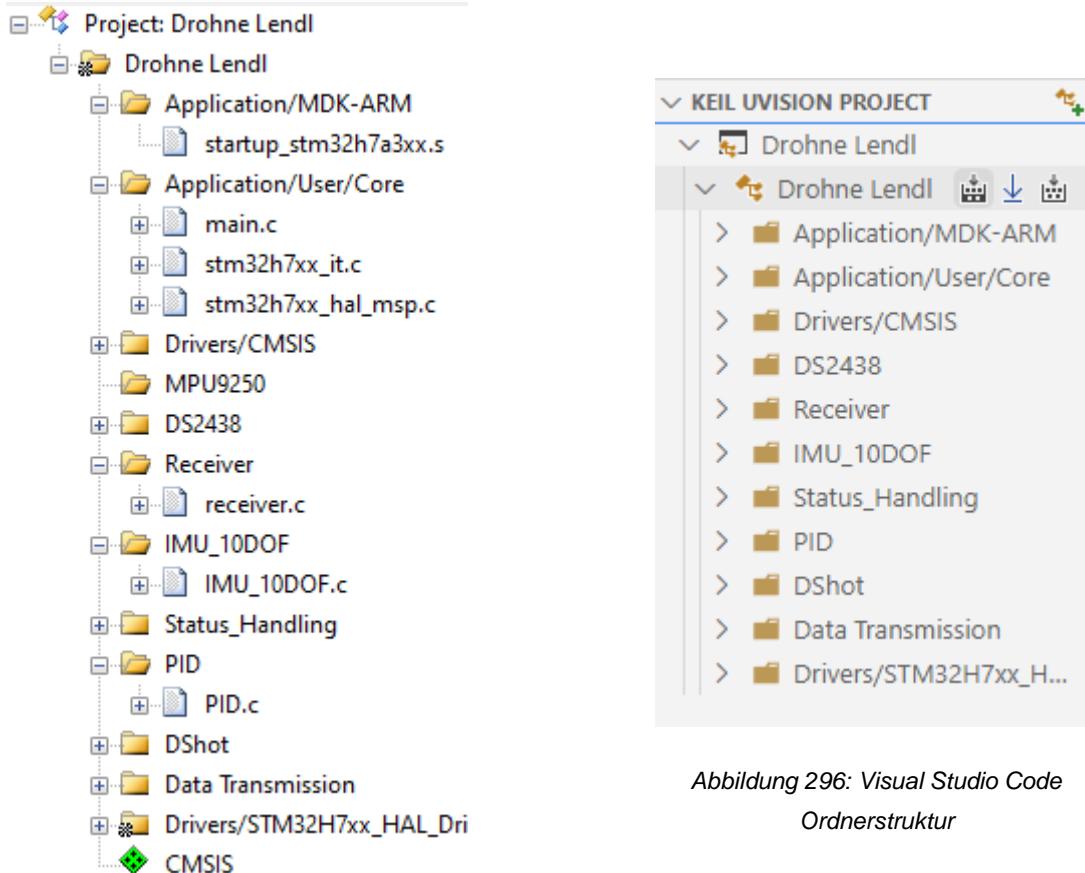


Abbildung 295: µVision Ordnerstruktur

Abbildung 296: Visual Studio Code
Ordnerstruktur

Visual Studio Code:

Mit Visual Studio Code kann dieselbe Ordnerstruktur des Keil µVision5 Projekts geöffnet und dessen Funktionen gesteuert werden (siehe: Abbildung 295 und 296).

10.3 Kurzeinführung CAD – Software (Fusion 360)

Für alle 3D-Körper, die im Rahmen dieser Diplomarbeit entworfen wurden, wurde die CAD-Software Fusion 360 von Autodesk verwendet. Die Software kann, neben dem Zeichnen und Designen der Körper, auch Simulationen berechnen, wie zum Beispiel die Stabilität eines Objekts, und 3D-Render der 3D-Designs.

10.3.1 UI und Projekterstellung

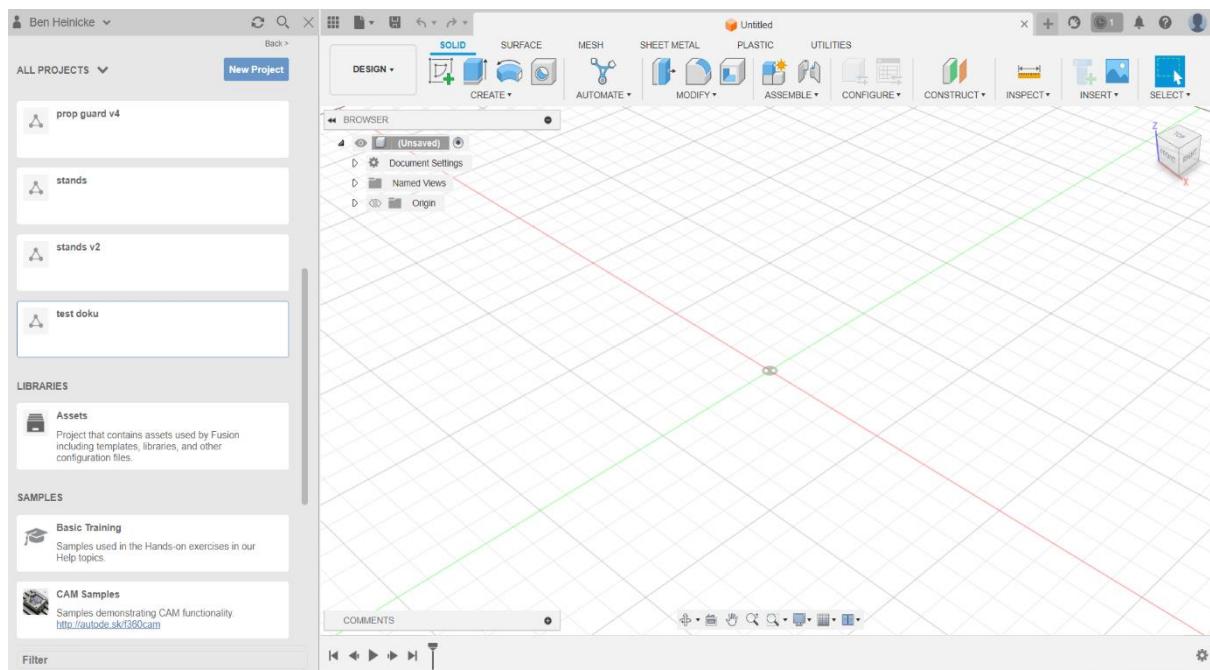


Abbildung 297: UI Fusion 360

Auf der linken Seite ist die Project-Library. Hier sieht man einen Überblick über all seine Projekte. Von dort aus können über den Button [New Project] auch neue Projekte erstellt werden. Auf der rechten Seite ist die Zeichenfläche, wo Skizzen und Körper erstellt werden können. Darüber hat man eine Schaltfläche, wo alle Tools und Funktionen, die für die Erstellung benötigt werden, zu finden sind. Dafür muss aber ganz links in der Schaltfläche der Punkt [Design] ausgewählt sein.

10.3.2 Skizze anfertigen

Um eine neue Skizze zu erstellen, klickt man auf den ersten Menüpunkt in der Tool-Schaltfläche. Danach muss man eine Fläche auswählen, auf der die Skizzen gezeichnet werden sollen. Dies können entweder die 3 Grundflächen oder eigens erstellte Flächen (Planes) sein.

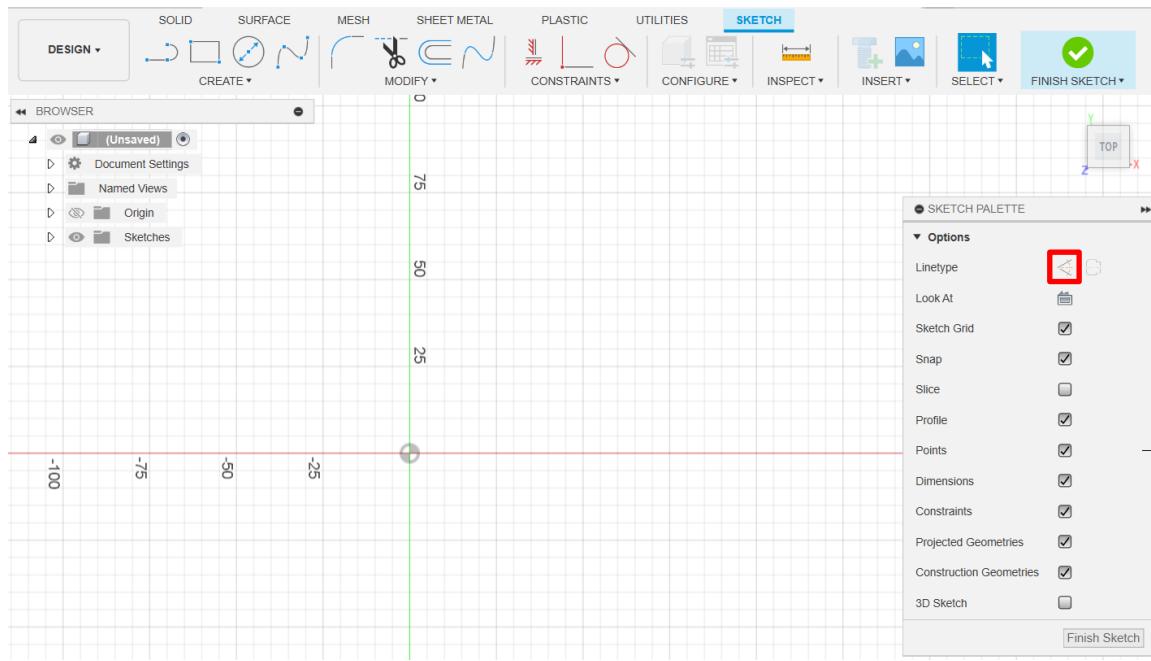


Abbildung 298: Skizze anfertigen

Danach befindet man sich im Zeichenmodus. Hier können in der oberen Schaltfläche alle möglichen Linienformen und Zeichentools gefunden werden. Im rechten Fenster kann der Linientyp „Construction“ ausgewählt werden, mit dem man Hilfslinien erstellen kann, die später in der 3D-Ansicht nicht angezeigt werden. Ist die Skizze fertig, dann drückt man oben rechts auf [Finish Sketch].

10.3.3 Körper erstellen

Um aus dieser Skizze jetzt einen Körper zu erstellen, wählt man die Fläche aus, aus der man einen Körper erstellen möchte und klickt dann in der oberen Schaltfläche auf den zweiten Button [Extrude].

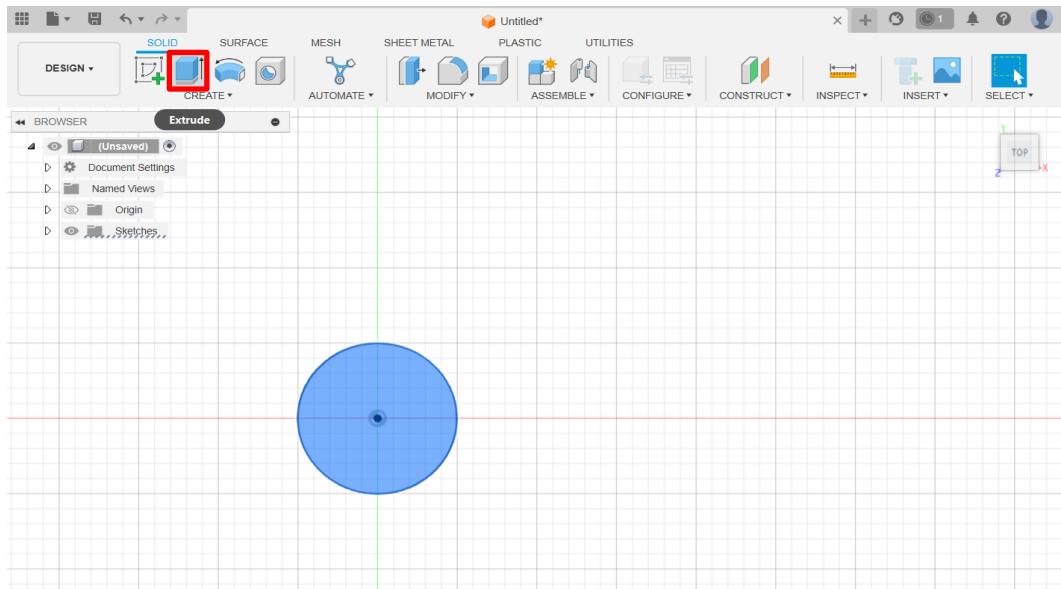


Abbildung 299: Extrude

Unter dem Fenster, dass dann aufgeht kann die Höhe des Körpers, der aus der Fläche erstellt wird, eingestellt werden.

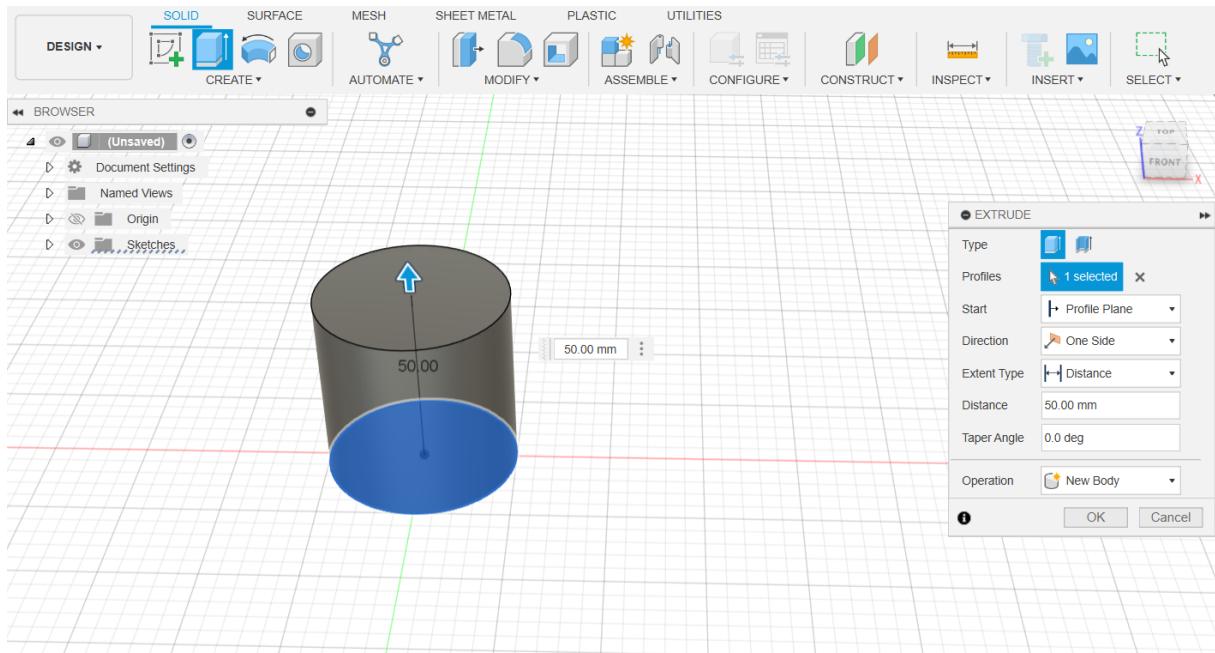


Abbildung 300: Extrude Einstellungen

10.3.4 Schrift und Bilder einfügen

Will man wie Schriften oder Logos einfügen, dann kann man das ganz einfach über die [Insert]-Schaltfläche machen. Dafür ist es fast am einfachsten, wenn man die Bilder oder Schriften als Vektorgrafik (.svg) abspeichert.

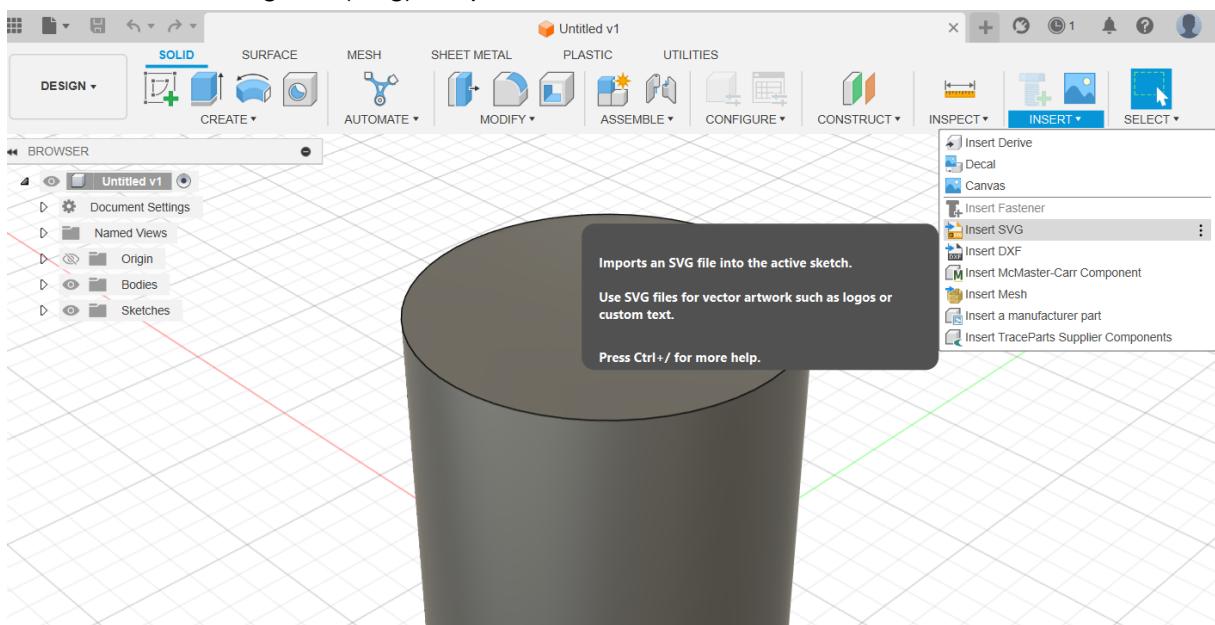


Abbildung 301: Grafiken Einfügen

Dadurch wird die Vektorgrafik in Sketch-Linien konvertiert. Diese können dann auch beliebig als Fläche ausgewählt und zu einem 3D-Körper gemacht werden [Extrude]. Um die fertigen Objekte für den 3D-Druck zu exportieren, muss man sie als STL-File exportieren. (Datei → exportieren)

10.4 3D – Druck

Um die fertigen 3D-Objekte dann auszudrucken, haben wir die von unserer Schule zur Verfügung gestellten 3D-Drucker verwendet. Dabei handelt es sich um die Modelle Ultimaker 2 Extended+ und Ultimaker S5.

10.4.1 3D – Drucker Software (UltiMaker Cura)

Um aus den STL-Files dann letztendlich für den 3D-Drucker benutzbare Files zu erstellen, benötigt man die dazugehörige Druckersoftware, auch genannt Slicer. In unserem Fall ist das UltiMaker Cura. Man kann die Files ganz einfach importieren und dann auf den Druck vorbereiten.

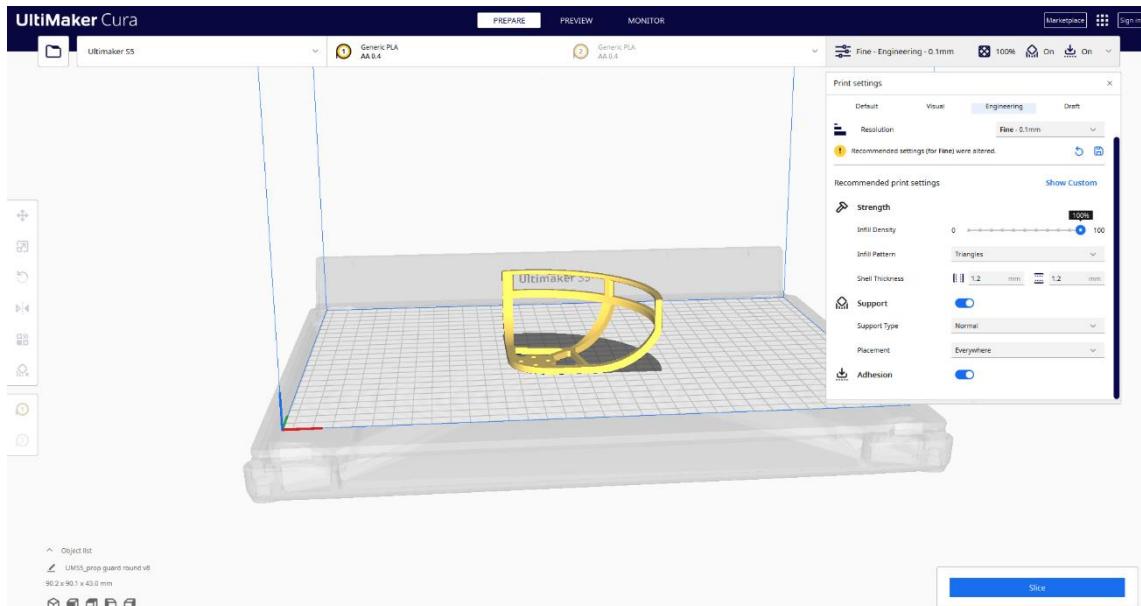


Abbildung 302: UltiMaker Cura UI

Oben links muss der benutzte Drucker ausgewählt werden. Rechts daneben kann das Filament und der Durchmesser der Düse eingestellt werden. Für alle Modelle wurde PLA-Filament zum Drucken verwendet. Im Einstellungsmenü auf der rechten Seite können nun alle zusätzlichen Druckeinstellungen vorgenommen werden. Als Druckauflösung wurde immer mit Fine gedruckt. Das dauert zwar länger, ergibt aber ein schöneres Endergebnis. Als Infill-Pattern wurden Dreiecke verwendet, da Dreiecke eine sehr stabile Form darstellen. Außerdem wurde meistens mit einer Infill-Dichte von 100% gedruckt, um die Stabilität zu maximieren. Alle anderen Einstellungen wurden auf ihren Standardwerten gelassen.

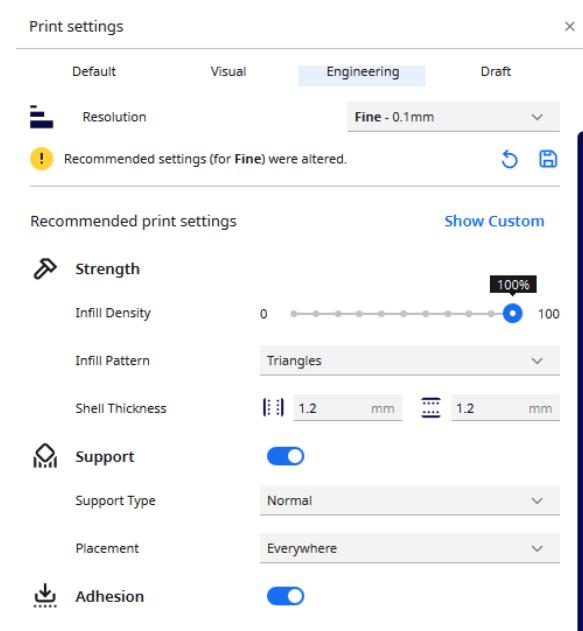


Abbildung 303: Einstellungen

Ist alles wie gewünscht eingestellt, muss man zuletzt auf [Slice] (unten rechts) drücken und kann das fertige Druckerfile abspeichern.

10.5 Inbetriebnahme Anleitung

In diesem Kapitel wird beschrieben, welche Schritte notwendig sind, um die FPV-Drohne betriebsbereit zu machen. Diese Anleitung sollte man sich vor jedem Flug sorgfältig durchlesen, um Beschädigungen oder Verletzungen zu vermeiden.

10.5.1 Aufladen der Akkus

Der Akku kümmert sich um die Spannungsversorgung auf der Drohne. Hierfür kann ein 3S – 6S LIPO-Akku verwendet werden.

Im folgenden Beispiel wird ein 6S 1300mAh LIPO-Akku mit einem XT60 Stecker verwendet.



Abbildung 304: 6S LIPO Akku Sideview

10.5.1.1 Ladegerät

Für die Aufladung des Akkus wird ein HOTA D6 Pro AC 200W DC 650W 15A*2 Dual Channel LiPo-Ladegerät verwendet.



Abbildung 305: Ladegerät

Das Ladegerät besitzt zwei Ladeports mit dem XT60 Standard und 2 Balancer Ports. Damit wäre es sogar möglich, zwei Akkus gleichzeitig aufzuladen.



Abbildung 306: Ladegerät Ports

Ladevorgang

1. Zuerst wird der Akku an den Ladeport des Ladegeräts angeschlossen



Abbildung 307: Akku an Ladeport anschließen

2. Danach wird der Balancer Stecker des Akkus an den Balancer Port des Ladegeräts angeschlossen



Abbildung 308: Akku an Balancer Port anschließen

- Nach dem Anstecken des Ladeports und Balancer Ports sollte der Ladegeräts Bildschirm folgendermaßen aussehen:



Abbildung 309: Ladegerät Bildschirm nach Akku anstecken

- Nun wird auf den Channel 1 des Ladegeräts gewechselt, um den Akku aufzuladen. Dies erreicht man indem man links auf den schwarzen CH-Button drückt, um die Channel zu wechseln.



Abbildung 310: Ladegerät Bildschirm Ladedaten 1

In dieser Ansicht kann man links oben den aktuellen Ladestrom und darunter die aktuelle Spannung erkennen. Rechts oben werden die schon aufgeladenen mAh des Akkus angezeigt. Darunter wird die Zeit gemessen, wie lange der Akku schon lädt. Ganz unten am Bildschirm werden auch die einzelnen Zellspannungen des Akkus angezeigt, die mittels Balancer Port gemessen werden.

5. Es können jedoch auch andere Daten angezeigt werden. Dies erreicht man mittels des roten Reglers auf der rechten Seite des Ladegeräts.



Abbildung 311: Ladegerät Bildschirm Ladedaten 2

Bei dieser Ansicht kann man sich zusätzliche Daten anzeigen lassen, jedoch wechseln wir für den Ladevorgang wieder auf die Zellspannungen zurück.

6. Wenn man nun auf den roten Regler drückt, kommt man in die Task Settings.

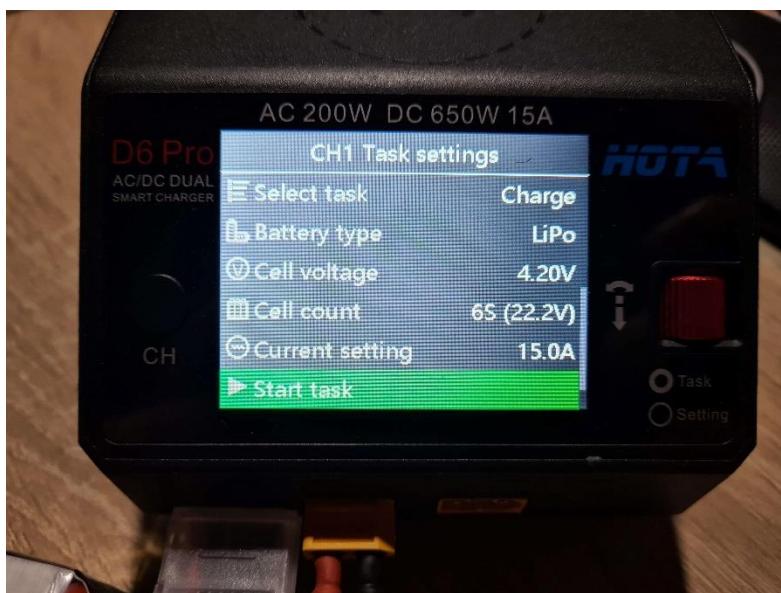


Abbildung 312: Ladegerät starte Ladevorgang

Hier können wir verschiedene Einstellungen wählen. Da wir unseren Akku aufladen wollen, wählen wir die Aufgabe „Charge“. Darunter müssen wir unseren Akku-Typ angeben, was in unserem Fall ein LIPO Akku ist. Danach haben wir noch die Möglichkeit die maximale Zellspannung anzugeben, damit das Ladegerät jede Zelle auf diese Spannung auflädt. Darunter muss die Anzahl der Akkuzellen eingegeben werden, damit das Ladegerät weiß, um welchen Akku es sich handelt. Das Ladegerät hat aber die Möglichkeit die Zellspannung und die Zellenanzahl automatisch zu erkennen. Außerdem haben wir die Möglichkeit, den maximalen Ladestrom

einzustellen. Ganz unten bei „Start task“ kann der Ladevorgang nun gestartet werden, indem man auf den roten Regler drückt.

7. Nun wird der Akku aufgeladen



Abbildung 313: Akku Ladevorgang

Während des Ladevorgangs kann nun der aktuelle Ladestrom links oben abgelesen werden. Darunter kann man die aktuelle Gesamtspannung des Akkus ablesen. Auf der rechten Seite des Bildschirms sieht man außerdem die bereits aufgeladenen mAh des Akkus und wie lange der Akku schon lädt. Besonders wichtig zu beachten sind die einzelnen Zellspannungen des Akkus, da sich alle ungefähr gleich aufladen müssen. Wenn dies nicht der Fall ist, ist eine Zelle wahrscheinlich nicht mehr funktionsfähig.

8. Wenn der Ladevorgang abgeschlossen ist, leuchtet der Bildschirm grün und das Gerät piepst. Nun kann der voll aufgeladene Akku abgesteckt und verwendet werden.

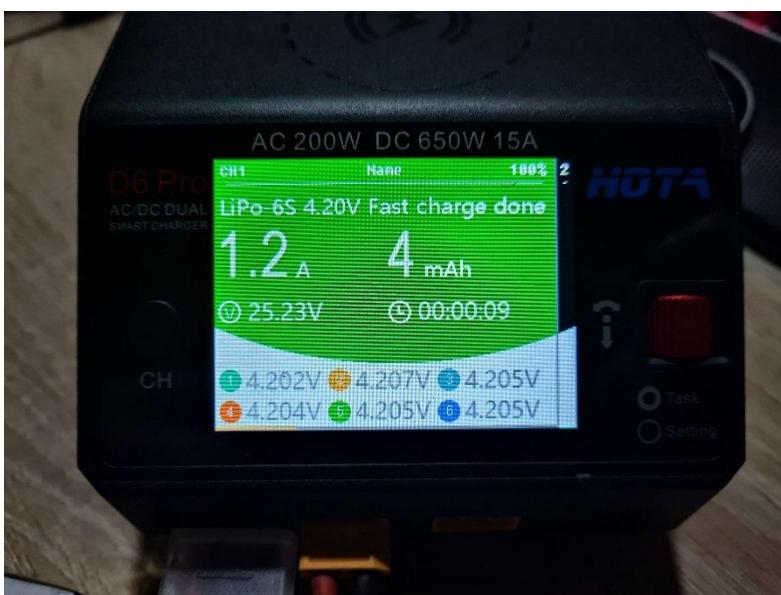


Abbildung 314: Ladegerät Ladevorgang abgeschlossen

10.5.2 Inbetriebnahme der FPV-Drohne

Im folgenden Kapitel werden die notwendigen Schritte erläutert, wie man die FPV-Drohne flugbereit macht.

1. Ausgangssituation: Um die Drohne flugbereit zu machen, braucht man die FPV-Drohne und einen beliebigen 3S – 6S Akku

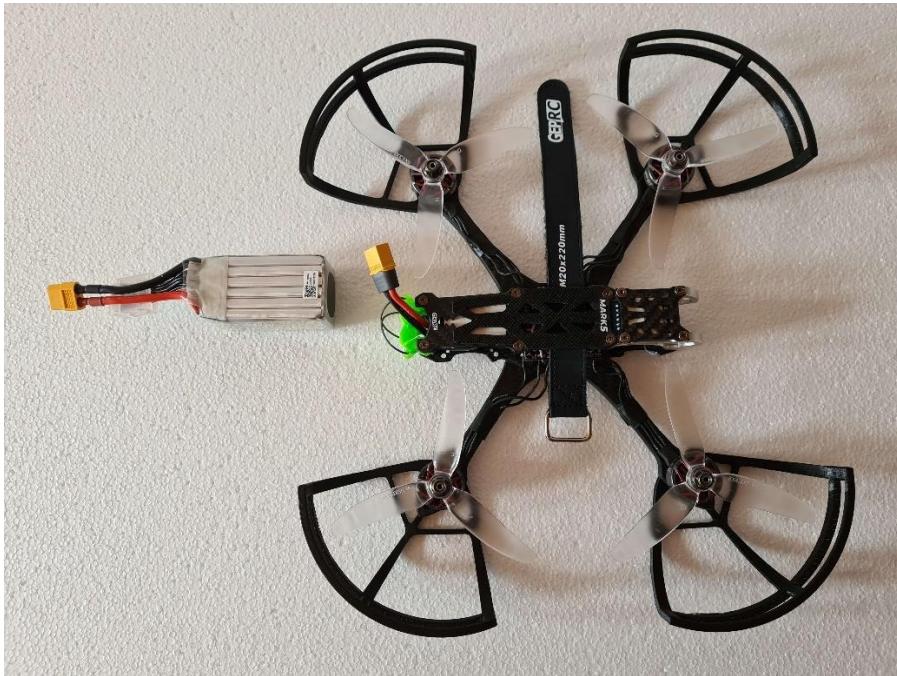


Abbildung 315: FPV-Drohne Inbetriebnahme Ausgangssituation

2. Danach wird der Akku auf der FPV-Drohne platziert und mittels Battery-Strap montiert



Abbildung 316: FPV-Drohne Inbetriebnahme Akku montiert

3. Anschließend wird der Akku an den ESC mittels XT60 Steckverbindung angeschlossen. Hierbei sollte die grüne LED zu leuchten beginnen



Abbildung 317: FPV-Drohne Inbetriebnahme Akku angeschlossen

4. Anschließend muss der Reset Button (rot markiert) auf dem Flight Controller gedrückt werden, um die Startsequenz der FPV-Drohne einzuleiten

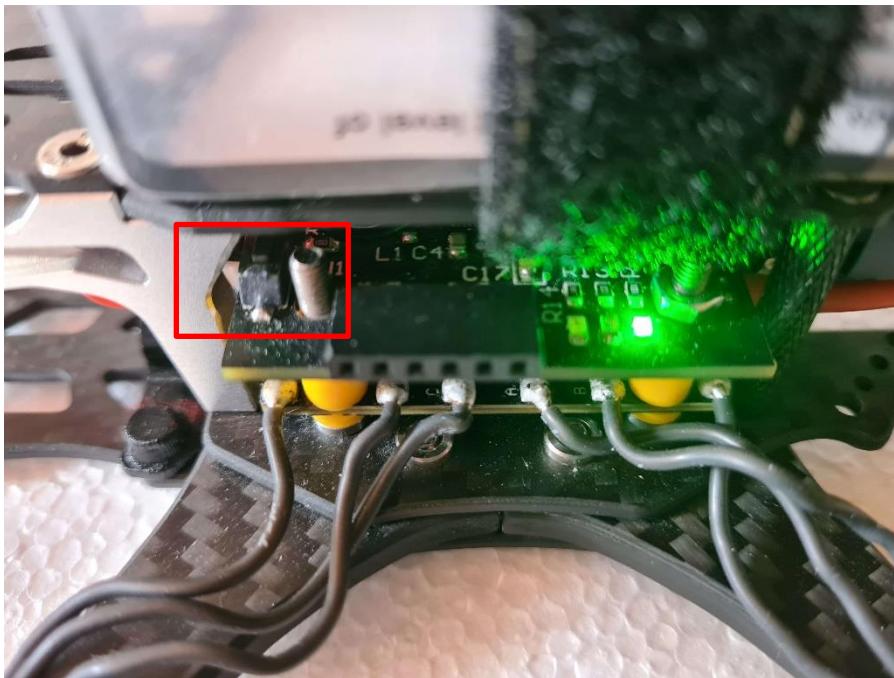


Abbildung 318: FPV-Drohne Inbetriebnahme Reset Button

5. Nun sollte der Initialisierungsvorgang gestartet sein und die blaue LED sollte zu blinken beginnen



Abbildung 319: FPV-Drohne Inbetriebnahme Initialisierungsvorgang

6. Wenn die Initialisierung fehlgeschlagen ist, sollte nun die rote LED leuchten. Der Fehlercode kann am Terminal angezeigt werden.

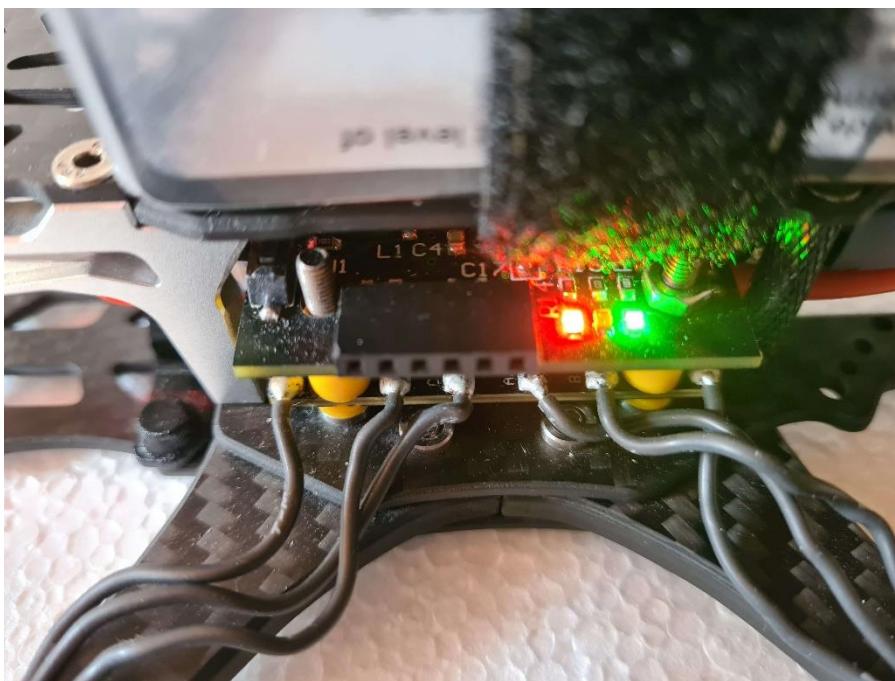


Abbildung 320: FPV-Drohne Inbetriebnahme Initialisierung fehlgeschlagen

7. Wenn die Initialisierung erfolgreich war, sollte nun die blaue LED leuchten. Die Drohne ist jetzt flugbereit.



Abbildung 321: FPV-Drohne Inbetriebnahme Initialisierung erfolgreich

10.6 Projektpläne

10.6.1 Bieder & Lendl

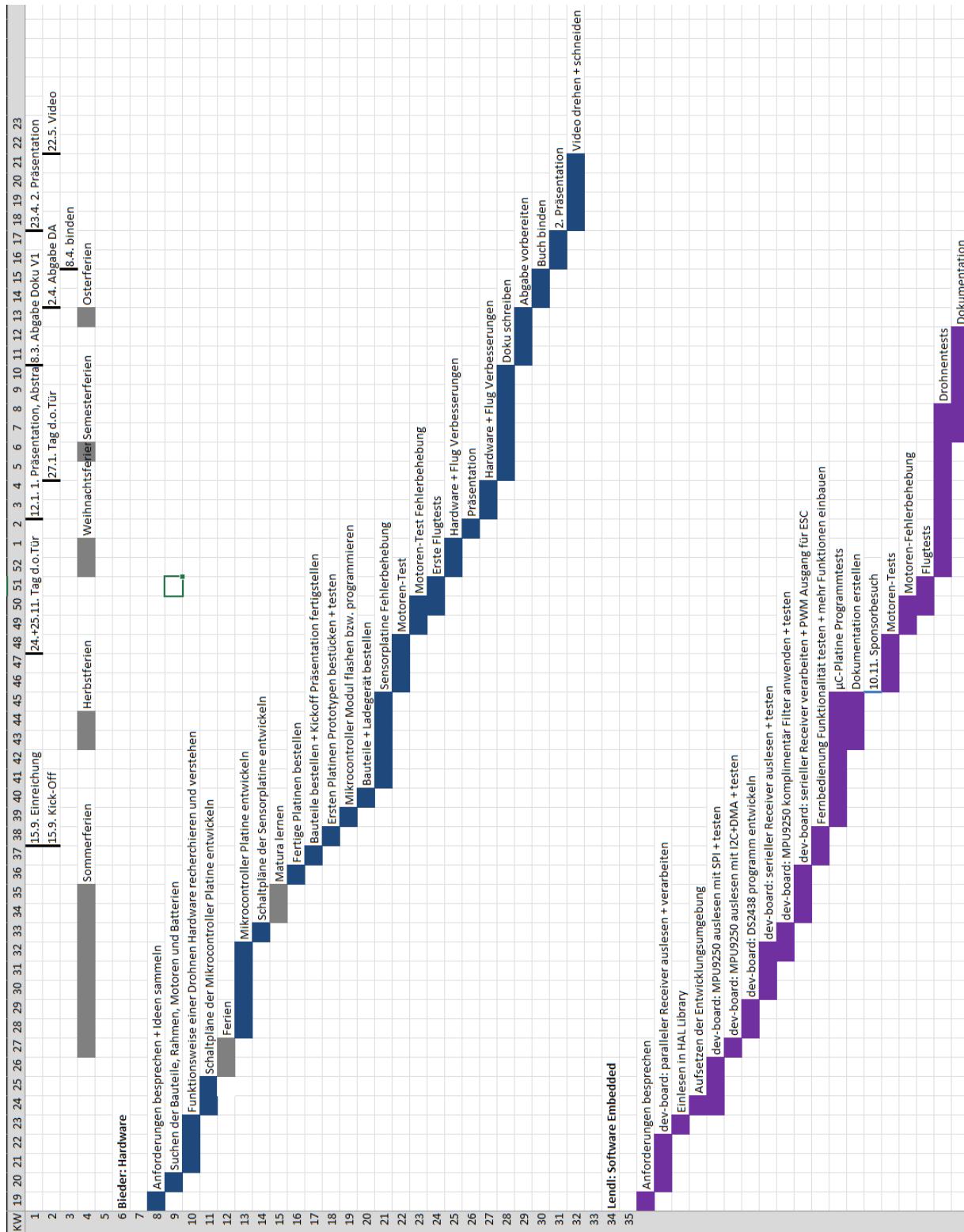


Abbildung 322: Projektplan Bieder & Lendl

10.6.2 Heinicke & Hinterberger

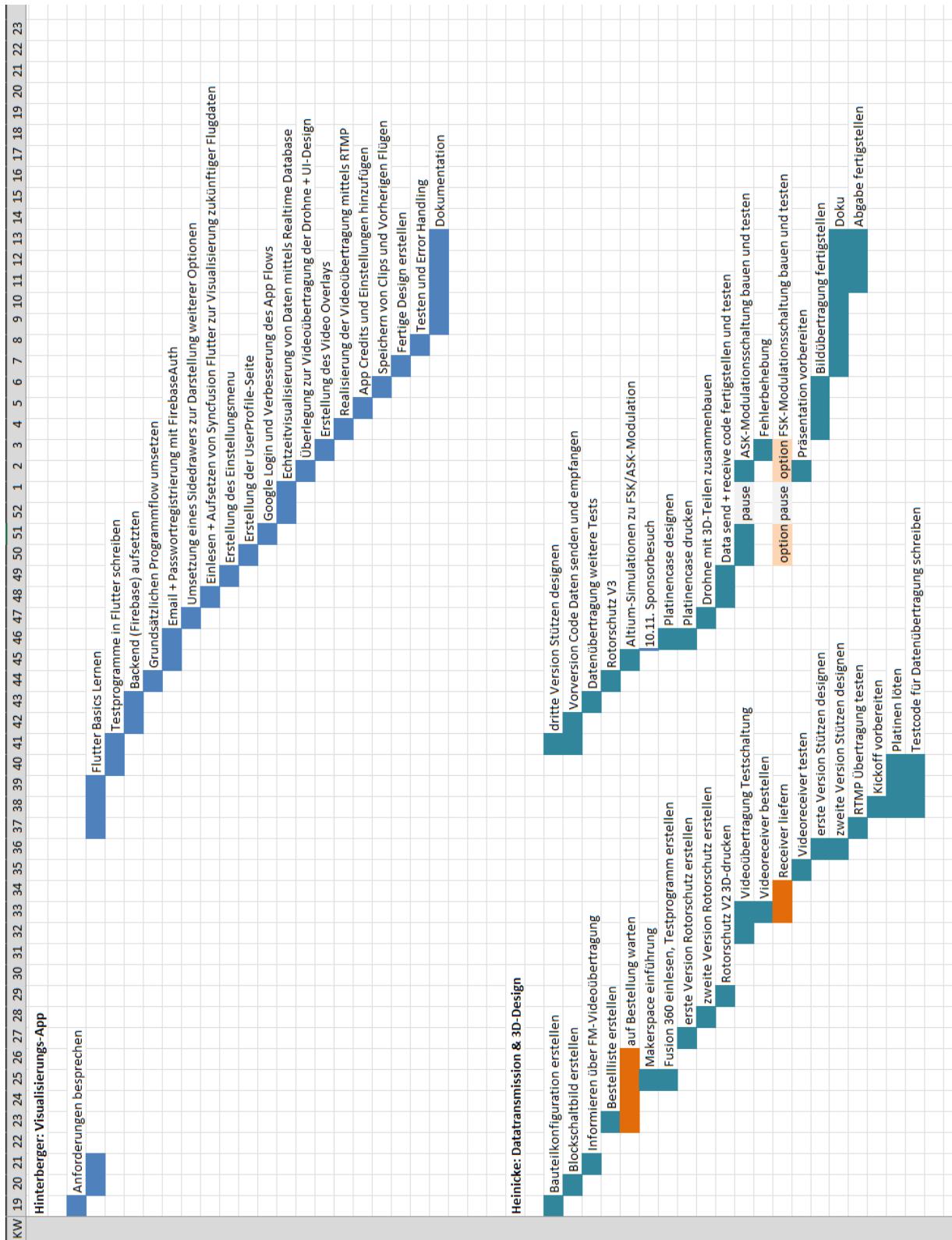
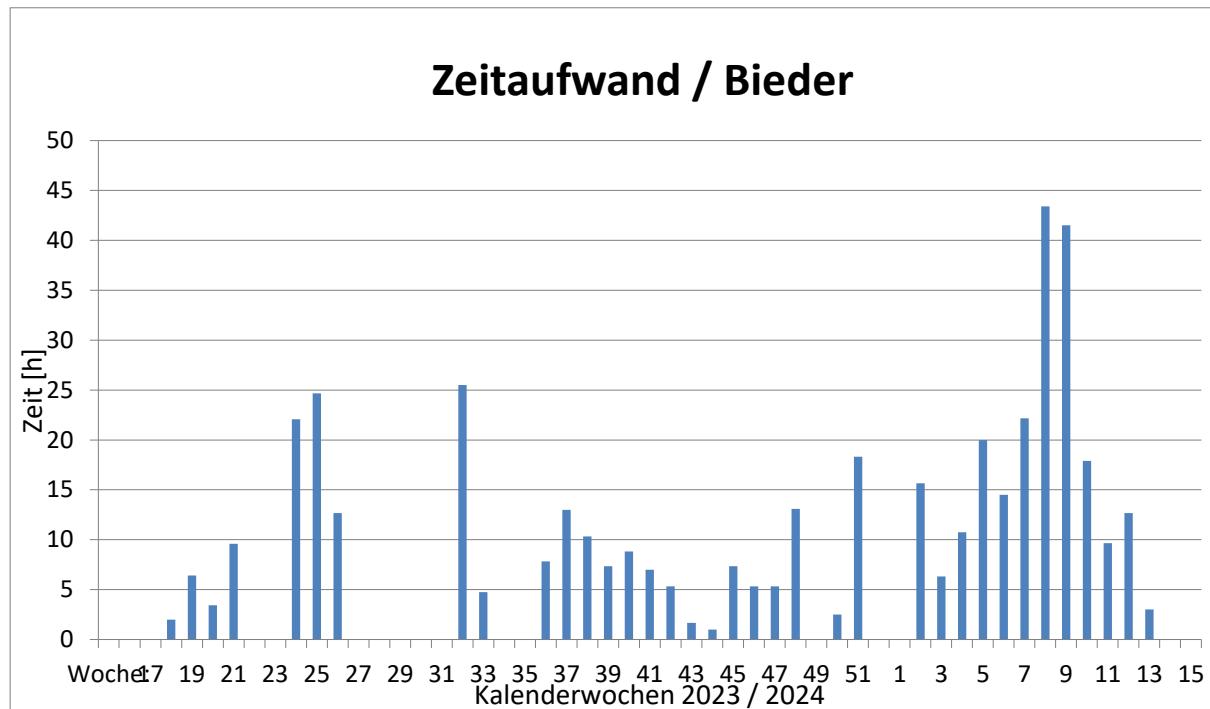


Abbildung 323: Projektplan Heinicke & Hinterberger

10.7 Zeitaufwand

10.7.1 Bieder

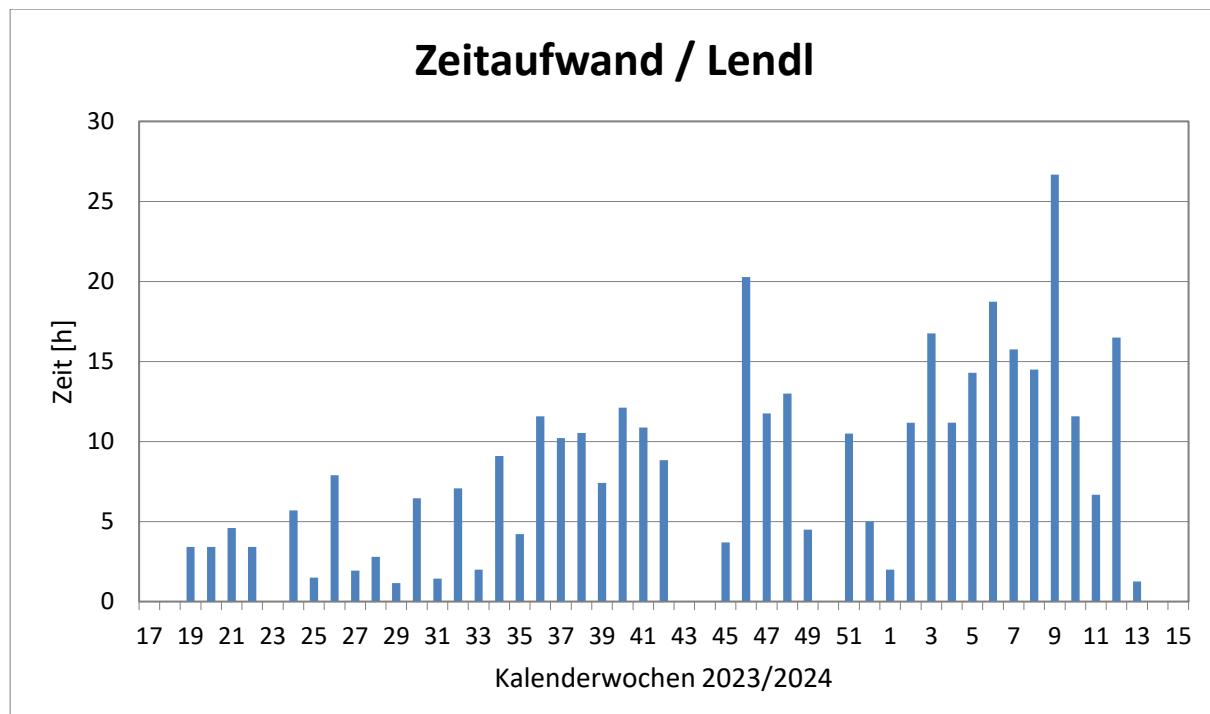


Schule: 170,38 Stunden

Freizeit: 272,54 Stunden

Summe: 442,92 Stunden

10.7.2 Lendl

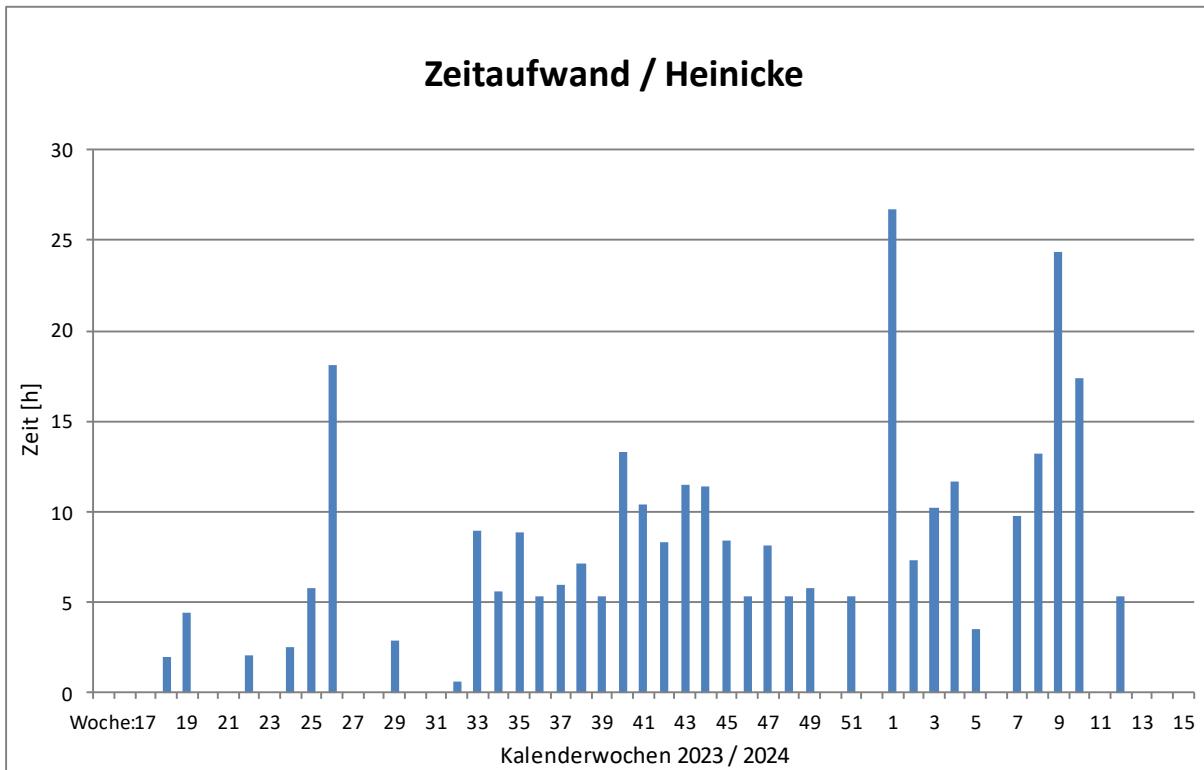


Schule: 124,70 Stunden

Freizeit: 248,68 Stunden

Summe: 373,38 Stunden

10.7.3 Heinicke

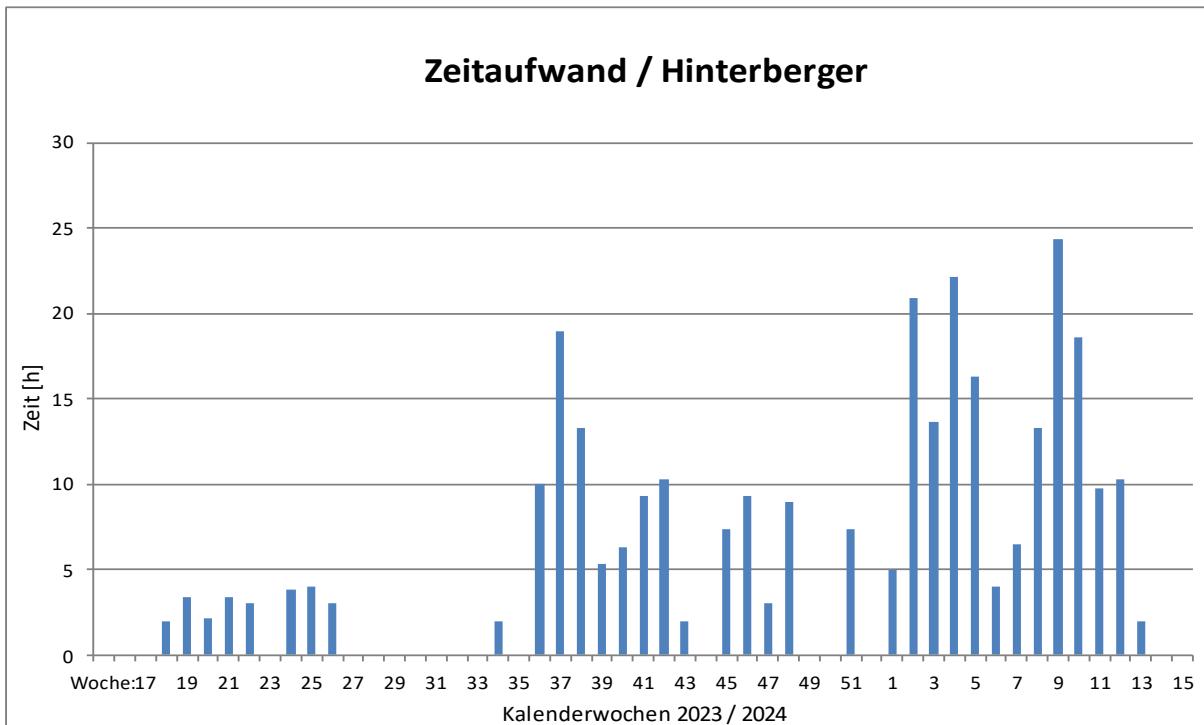


Schule: 108,83 Stunden

Freizeit: 199,35 Stunden

Summe: 308,18 Stunden

10.7.4 Hinterberger



Schule: 117,45 Stunden

Freizeit: 188,02 Stunden

Summe: 305,47 Stunden

10.8 Projektkosten

Einnahmen:	Ben	Max	Sebastian	Marcel	DroneTech	Reisinger	Gesamt:
12.06.2023							1115,07
14.06.2023	100	100	100	101			
23.06.2023					70		
28.06.2023	100	100	100	30			
13.10.2023	50	50	50	50			
19.01.2024						74,76	
14.02.2024						39,31	
	250	250	250	251	114,07	0	

Ausgaben:	Teil	Einzelstückpreis	Stückanzahl	Preis	Gesamt:
16.06.2023	Rahmen	99,9	1	99,9	964,42
16.06.2023	Motoren	27,9	4	111,6	
16.06.2023	ESC	60,9	1	60,9	
16.06.2023	Propeller	2,99	4	11,96	
16.06.2023	Kamera	51,26	1	51,26	
16.06.2023	FPV-Sender	34,9	1	34,9	
16.06.2023	SMA-Antenne	22,9	1	22,9	
16.06.2023	Propeller für REJ	1,99	1	1,99	
16.06.2023	Motor für REJ	10,9	1	10,9	
16.06.2023	ESC für REJ	28,9	1	28,9	
12.09.2023	Bauteile für Platinen	50	1	50	
08.09.2023	Platinen	50	1	50	
22.09.2023	Neue Bauteile	40	1	40	
06.10.2023	Ladegerät	107,53	1	107,53	
06.10.2023	MPU	29,9	1	29,9	
29.12.2023	3 Motoren	29,9	3	89,7	
29.12.2023	ESC+ FC	94,9	1	94,9	
13.02.2024	M3 Schrauben	0,43	16	6,88	
14.02.2024	Akku	39,31	1	39,31	
16.02.2024	Klebeband	8,99	1	8,99	
26.03.2024	Propeller		3	4	12

Versand/Zoll:	Art	Kosten	Gesamt:	Gesamtausgaben:
16.06.2023	Versand	6,9	121,43	1085,85
22.06.2023	Zoll	56,03		
08.09.2023	Versand	50		
26.03.2024	Versand	8,5		
				Übriges Budget:
				29,22

11 Quellen

11.1 Gedruckte Medien

- [GKD1] Dipl. -Ing Mag. Michael Wihsböck: DIC-Skriptum 2. Jahrgang
- [PID1] Dipl. -Ing Erwin Dobart: MTRS-Skriptum 4. Jahrgang
- [DRREJ] Dr.-Ing. Erwin Samal und Prof. Dr.-Ing Wilhelm Becker: Grundriß der praktischen Regelungstechnik
21. Auflage, Oldenbourg Verlag München Wien, 2013
ISBN: 978-3-486-27583-4

11.2 Online

- [TO263] TO-263-3 Gehäuse
<https://blog.mbedded.ninja/pcb-design/component-packages/to-263-component-package/d2pak-to-263-component-package-3d-render.jpg>
(Letzter Aufruf: 12.02.2024)
- [TO252] TO-252-3 Gehäuse
<https://encrypted-tbn0.gstatic.com/images?q=tbn:ANd9GcTNcOS0T4eLI2fj4mMriWprZCFIgKpEF-L10A&usqp=CAU>
(Letzter Aufruf: 12.02.2024)
- [8SOIC] DS2438 Smart Battery Monitor Gehäuse
<https://mm.digikey.com/Volume0/opasdata/d220001/medias/images/192/175%7E21-0041%7ESA%7E8.JPG>
(Letzter Aufruf: 13.02.2024)
- [LQFP] 64-LQFP Gehäuse
https://mm.digikey.com/Volume0/opasdata/d220001/medias/images/4832/497%7E5_W_ME%7ET%7E64.JPG
(Letzter Aufruf: 14.02.2024)
- [TMF60] T Motor F60PROV
https://store.tmotor.com/images/202208/thumb_img/1659575787597547885.jpg
(Letzter Aufruf: 23.02.2024)
- [BMIA] BLDC Motor Innenaufbau
<https://www.ato.com/Content/Images/uploaded/blog/how-does-ESC-work.jpg>
(Letzter Aufruf: 21.02.2024)
- [F60MD] T Motor F60PROV Daten
<https://store.tmotor.com/product/f60prov-lv-fpv-motor.html>
(Letzter Aufruf: 21.02.2024)

- [V24AD] V45A LITE 6S Daten
<https://store.tmotor.com/product/v45a-lite-4in1-velox-esc.html>
(Letzter Aufruf: 22.02.2024)
- [ES CAB] ESC Aufbau
<https://www.digikey.de/de/articles/how-to-power-and-control-brushless-dc-motors>
(Letzter Aufruf: 28.02.2024)
- [IMUD] IMU Breakout Daten
<https://botland.de/9dof-imu-sensoren/8298-imu-10dof-mpu9255-bmp280-3-achsenbeschleunigungsmesser-gyroskop-und-magnetometer-und-barometer-waveshare-12476-5904422375805.html>
(Letzter Aufruf: 28.02.2024)
- [VTXD] TBS Unify Pro 5.8 GHz Daten
<https://www.fpv24.com/de/team-blacksheep/tbs-unify-pro-5g8-hv-race-sma>
(Letzter Aufruf: 28.02.2024)
- [RATE2] CADDXFPV Ratel2 Analog Camera Daten
<https://caddxfpv.com/products/ratel-2-1-1-8inch-starlight-sensor-freestyle-fpv-camera>
(Letzter Aufruf: 28.02.2024)
- [TIA6C] Turnigy iA6C
https://hobbyking.com/en_us/turnigy-ia6c-ppm-sbus-receiver.html?_store=en_us
(Letzter Aufruf: 29.02.2024)
- [AKKUA] LIPO-Akku Aufladeverhalten
<https://www.fly2air.com/tipps/Akkus/main-Akkus.htm>
(Letzter Aufruf: 01.03.2024)
- [AKKUE] LIPO-Akku Entladeverhalten
<https://betaflight.de/docs/knowledge-base/bauplanung-und-bauteile/kleine-lipo-kunde/>
(Letzter Aufruf: 03.03.2024)
- [KLIZ] ARM Keil: Community Lizenz Anleitung
<https://www.keil.arm.com/mdk-community/>
(Letzter Aufruf: 28.02.2024)
- [KASS] Visual Studio Marketplace: Keil Assistant Erweiterung
<https://marketplace.visualstudio.com/items?itemName=CL.keil-assistant>
(Letzter Aufruf: 28.02.2024)
- [SERM] Visual Studio Marketplace: Serial Monitor Erweiterung
<https://marketplace.visualstudio.com/items?itemName=ms-vscode.vscode-serial-monitor>
(Letzter Aufruf: 28.02.2024)

- [SBDW] digitalwire: Futaba S-Bus Protokoll
<https://digitalwire.ch/de/projekte/futaba-sbus/>
(letzter Aufruf: 28.02.2024)
- [IBDSP] The FlySky iBus protocol
<https://blog.dsp.id.au/posts/2017/10/22/flysky-ibus-protocol/>
(letzter Aufruf: 28.02.2024)
- [IBGH] STM32 HAL iBUS
https://github.com/mokhwasomssi/stm32_hal_ibus
(letzter Aufruf: 28.02.2024)
- [SBGH] STM32 RadioLink SBUS DMA
https://github.com/osos11-Git/STM32_RadioLink_SBUS_DMA/tree/main
(letzter Aufruf: 02.03.2024)
- [DSBW] Brushless Whoop: DSHOT – the missing handbook
<https://brushlesswhoop.com/dshot-and-bidirectional-dshot/>
(letzter Aufruf: 28.02.2024)
- [DSBF] Betaflight: DSHOT
<https://betaflight.com/docs/development/Dshot>
(letzter Aufruf: 02.03.2024)
- [DSGH] stm32 hal dshot
https://github.com/mokhwasomssi/stm32_hal_dshot/tree/main
(letzter Aufruf: 02.03.2024)
- [IMUWS] Waveshare: 10 DOF IMU Sensor
[https://www.waveshare.com/wiki/10_DOF_IMU_Sensor_\(C\)](https://www.waveshare.com/wiki/10_DOF_IMU_Sensor_(C))
(letzter Aufruf: 28.02.2024)
- [M7] STM32H7A3RG7 Datenblätter
<https://www.st.com/en/microcontrollers-microprocessors/stm32h7a3rg.html>
(letzter Aufruf: 28.02.2024)
- [M7HAL] STM32H7 HAL Datenblätter
<https://www.st.com/en/embedded-software/stm32cubeh7.html#documentation>
(letzter Aufruf: 28.02.2024)
- [DS2438] DS2438 Datenblatt
<https://www.analog.com/media/en/technical-documentation/data-sheets/DS2438.pdf>
(letzter Aufruf: 28.02.2024)
- [COFIL] Complementary filter and relative orientation with MPU9250
<https://www.hackster.io/hibit/complementary-filter-and-relative-orientation-with-mpu9250-d4f79d>
(letzter Aufruf: 28.02.2024)

- [FFLO] Flutter Framework Logo
<https://upload.wikimedia.org/wikipedia/commons/thumb/4/44/Google-flutter-logo.svg/2560px-Google-flutter-logo.svg.png>
(Letzter Aufruf: 25.02.24)
- [SLSF] Stateful vs. Stateless Widget
https://www.nitorinfotech.com/wp-content/uploads/2022/06/Stateless-and-stateful-widgets_Nitor-Infotech.jpg
(Letzter Aufruf: 26.02.24)
- [PROV] State Management via Provider
https://koenig-media.raywenderlich.com/uploads/2019/11/provider_tree.png
(Letzter Aufruf: 27.02.24)
- [NGXL] NGINX-Logo
https://live.staticflickr.com/5481/12252140713_bf663db1ef_b.jpg
(Letzter Aufruf: 27.02.24)
- [RTEX] Realtime Colors: Farbpaletten generieren und exportieren
<https://www.realtimecolors.com/>
(Letzter Aufruf: 27.02.24)
- [NJSD] Nodejs Download Webseite
<https://nodejs.org/en/download/>
(Letzter Aufruf: 27.02.24)
- [FFML] FFmpeg-Logo
<https://upload.wikimedia.org/wikipedia/commons/thumb/4/4b/FFmpeg-Logo.svg/2560px-FFmpeg-Logo.svg.png>
(Letzter Aufruf: 27.02.24)
- [WMIP] Öffentliche IP-Adresse herausfinden
<https://whatismyipaddress.com/>
(Letzter Aufruf: 27.02.24)
- [CAPU] USB2.0 Videograbber
https://m.media-amazon.com/images/I/71I8NIQmLQL._AC_UF894,1000_QL80_.jpg
(Letzter Aufruf: 27.02.24)
- [RATE] CDDXFPV Ratel 2 analog Camera
https://cdn-v2.getfpv.com/media/catalog/product/c/a/caddx_ratel_2_fpv_camera_1.jpg
(Letzter Aufruf: 27.02.24)
- [VSCL] VS-Code Logo
https://upload.wikimedia.org/wikipedia/commons/thumb/9/9a/Visual_Studio_Code_1.35_icon.svg/2048px-Visual_Studio_Code_1.35_icon.svg.png
(Letzter Aufruf: 27.02.24)

- [PYLO] Python Logo
<https://upload.wikimedia.org/wikipedia/commons/thumb/c/c3/Python-logo-notext.svg/800px-Python-logo-notext.svg.png>
(Letzter Aufruf: 29.02.24)
- [FFMP] FFMPEG Dokumentation
<https://ffmpeg.org/ffmpeg.html>
(Letzter Aufruf: 27.02.24)
- [GGFP] Liste an Optionen in FFMPEG
<https://gist.github.com/tayvano/6e2d456a9897f55025e25035478a3a50>
(Letzter Aufruf: 27.02.24)
- [RMWK] RTMP-Wikipedia
https://de.wikipedia.org/wiki/Real_Time_Messaging_Protocol#Protokollvarianten,
(Letzter Aufruf: 23.02.24)
- [FFWK] FFMPEG-Wikipedia
<https://de.wikipedia.org/wiki/FFmpeg>
(Letzter Aufruf: 01.03.2024)
- [GRL1] UST: Selecting an Inertial Measurement Unit (IMU) for UAV Applications
<https://www.unmannedsystemstechnology.com/feature/selecting-an-inertial-measurement-unit-imu-for-uav-applications/>
(Letzter Aufruf: 05.03.2024)
- [GRL2] InvenSense: MPU9250 Datenblatt
<https://invensense.tdk.com/wp-content/uploads/2015/02/PS-MPU-9250A-01-v1.1.pdf>
(Letzter Aufruf: 01.03.2024)
- Wikipedia: Magnetometer
<https://de.wikipedia.org/wiki/Magnetometer>
(Letzter Aufruf: 01.03.2024)
- [GRL3] StudySmarter: Gyroskop
<https://www.studysmarter.de/schule/physik/mechanik/gyroskop/>
(Letzter Aufruf: 01.03.2024)
- [GRL4] StudySmarter: Beschleunigungssensor
<https://www.studysmarter.de/studium/ingenieurwissenschaften/messtechnik/beschleunigungssensor/>
(Letzter Aufruf: 01.03.2024)
- [GRL5] ROHM Semiconductors: Barometrischer Drucksensor
<https://www.rohm.de/electronics-basics/sensor/barometric-pressure-sensor>
(Letzter Aufruf: 01.03.2024)

- [GRL6] Austro Control: Grundlagen des Fliegens
<https://online-kurs.dronespace.at/online-kurs/lehrmaterial/allgemeine-uas-kunde/grundlagen-des-fliegens/>
(Letzter Aufruf: 01.03.2024)
- [ANT1] Nils Waldmann: Überblick: Welche FPV-Antennen gibt es und wie nutzt du sie?
https://www.drone-zone.de/ueberblick-welche-fpv-antennen-gibt-es-und-wie-nutzt-du-sie/#FPV-Antenne_Bauformen_im_Vergleich
(Letzter Aufruf: 01.03.2024)
- [RIC1] Clover-Leaf-Antenne Image
<https://www.google.com/url?sa=i&url=https%3A%2F%2Flaboratorio.blau.com.ve%2F%3Fo%3Dcloverleaf-antennas-qq-28b9BmF4&psig=AOvVaw1JLkulKRNrauf1XoWA1-W&ust=1709410423889000&source=images&cd=vfe&opi=89978449&ved=0CBIQjRxqFwoTCJC35uXx04QDFQAAAAAdAAAAABAE>
(Letzter Aufruf: 01.03.2024)
- [MEN1] The Black Sheep: TBS Unify Pro 5.8 Video Transmitters
<https://www.team-blacksheep.com/media/files/tbs-unify-pro-5g8-manual.pdf>
(Letzter Aufruf: 01.03.2024)
- [GKD1] Igal Zeifman: memcpy C Function – Syntax, Examples, and Security Best Practices
<https://sternumiot.com/iot-blog/memcpy-c-function-examples-and-best-practices/>
(Letzter Aufruf: 01.03.2024)
- [UAR1] ST Microelectronics Community – JU: STM32G0 USART2 TXINV / DATAINV
<https://community.st.com/t5/stm32-mcus-products/stm32g0-usart2-txinv-datainv-not-having-effect-line-stays-active/td-p/141017>
(Letzter Aufruf: 01.03.2024)
- [APP1] 沈垚 / ShenYao China: USB Camera APP
https://play.google.com/store/apps/details?id=com.shenyaoch.android.usbcamera&hl=de_AT&gl=US&pli=1
(Letzter Aufruf: 01.03.2024)
- [RPI] anon10691195: USB-Kabel Belegung
<https://community.homenetgear.com/t/usb-buchse-defekt/29747/21?page=2>
(Letzter Aufruf: 01.03.2024)
- Understanding Raspberry Pi 4 GPIO Pinouts
<https://toptechboy.com/understanding-raspberry-pi-4-gpio-pinouts/>
(Letzter Aufruf: 01.03.2024)
- [FLU1] pySerial: pySerial API
https://pyserial.readthedocs.io/en/latest/pyserial_api.html
(Letzter Aufruf: 01.03.2024)

- [MQT1] Christoph Burgdorfer: [Tutorial] How to Set Up a Mosquitto MQTT Broker Securely—Using Client Certificates
<https://medium.com/gravio-edge-iot-platform/how-to-set-up-a-mosquitto-mqtt-broker-securely-using-client-certificates-82b2aaaef9c8>
(Letzter Aufruf: 01.03.2024)
- [COF1] Novellus: Raspberry PI 3 - Enable Serial Communications to TtyAMA0 to BCM GPIO 14 and GPIO 15
<https://www.instructables.com/Raspberry-PI-3-Enable-Serial-Communications-to-Tty/>
(Letzter Aufruf: 01.03.2024)
- [VLCW] VLC-Webseite
<https://www.videolan.org/vlc/index.de.html>
(Letzter Aufruf: 29.02.24)
- [FBPR] Firebase Services aufgelistet
<https://firebase.google.com/products-build>
(Letzter Aufruf: 29.02.24)
- [FFMC] FFMPEG-Codec-Dokumentation
<https://www.ffmpeg.org/ffmpeg-codecs.html>
(Letzter Aufruf: 24.02.24)
- [FFDD] FFMPEG-Detailbeschreibung
<https://ffmpeg.org/ffmpeg.html#toc-Detailed-description>
(Letzter Aufruf: 24.02.24)
- [RTSE] RTMP-Server aufsetzen Ubuntu
<https://www.howtoforge.com/how-to-set-up-a-video-streaming-server-using-nginx-rtmp-on-ubuntu-22-04/>
(Letzter Aufruf: 23.02.24)

12 Verzeichnis der Abbildungen

Abbildung 1: Gesamtüberblick.....	19
Abbildung 2: Barometer Innenaufbau [GRL5].....	21
Abbildung 3: Throttle Visualisierung	21
Abbildung 4: Pitch Visualisierung	21
Abbildung 5: Roll Visualisierung	22
Abbildung 6: Yaw Visualisierung	22
Abbildung 7: Regler-Blockschaltbild	25
Abbildung 8: P-Glied Schaltsymbol	26
Abbildung 9: P-Glied Ein/Ausgang	26
Abbildung 10: I-Glied Ein/Ausgang.....	26
Abbildung 11: I-Glied Schaltsymbol.....	26
Abbildung 12: P-Glied Schaltsymbol	27
Abbildung 13: P-Glied Ausgangsbestimmung.....	27
Abbildung 14: PID-Regler Regelung zum Sollwert.....	27
Abbildung 15: PID-Regler Schaltsymbol.....	27
Abbildung 16: FPV-Drohne Gesamtaufbau Seitenansicht	28
Abbildung 17: FPV-Drohne Gesamtaufbau Topansicht	28
Abbildung 18: FPV-Drohne Innenaufbau	29
Abbildung 19: Rotorschutz Design	30
Abbildung 20: FPV-Drohne mit Rotorschutz	30
Abbildung 21: Groundstation Design	31
Abbildung 22: Groundstation Design mit Deckel.....	31
Abbildung 23: Sponsor-Shield.....	32
Abbildung 24: Deckel Ausdruck.....	32
Abbildung 25: Groundstation Ausdruck	32
Abbildung 26: Elektronik Gesamtaufbau FPV-Drohne	33
Abbildung 27: 6S 1300mAh 120C LIPO	34
Abbildung 28: 5S 2200mAh 95C LIPO	35
Abbildung 29: LIPO-Akku Aufbau	35
Abbildung 30: Aufladeverhalten einer LIPO-Zelle [AKKU].....	36
Abbildung 31: Entladeverhalten einer LIPO-Zelle [AKKUE]	37
Abbildung 32: Schematic Spannungsversorgung Flight Controller.....	39
Abbildung 33: TO-263-3 Gehäuse [TO263]	40
Abbildung 34: Schematic 12V Step Down Converter	40
Abbildung 35: TO-252-3 Gehäuse [TO252]	41
Abbildung 36: Schematic 5V Step Down Converter	41
Abbildung 37: TO-252-3 Gehäuse [TO252]	42
Abbildung 38: Schematic 3,3V Step Down Converter	42
Abbildung 39: Schematic Versorgungsanschlüsse	43
Abbildung 40: DS2438 Smart Battery Monitor [8SIOC].....	43
Abbildung 41: Schematic DS2438 Smart Battery Monitor	44
Abbildung 42: Schematic Störfilter für Mikrocontroller	44
Abbildung 43: Schematic Störfilter für VDDA-Pin.....	45
Abbildung 44: 64-LQFP Gehäuse [LQFP]	45
Abbildung 45: Mikrocontroller Pinout	46
Abbildung 46: Interner Aufbau des Mikrocontrollers	47

Abbildung 47: Schematic Pinning Flight Controller	48
Abbildung 48: Schematic Mikrocontroller Pinbelegung	49
Abbildung 49: Schematic HSE Oszillator Mikrocontroller Pinning	50
Abbildung 50: Schematic HSE Oszillator	50
Abbildung 51: Schematic LSE Oszillator Mikrocontroller Pinning	50
Abbildung 52: Schematic LSE Oszillator	50
Abbildung 53: Schematic Bootloader	51
Abbildung 54: Schematic Reset Button	51
Abbildung 55: Programmierschnittstelle Mikrocontroller	51
Abbildung 56: Schematic Terminalschnittstelle	52
Abbildung 57: UART /USB-Converter	52
Abbildung 58: Schematic Status LEDs	52
Abbildung 59: PCB Ansicht Gesamtdesign Flight Controller	53
Abbildung 60: PCB Ansicht Top Layer Flight Controller	54
Abbildung 61: PCB Ansicht Bottom Layer Flight Controller	54
Abbildung 62: PCB 3D Ansicht Flight Controller	55
Abbildung 63: PCB 3D Ansicht Flight Controller	55
Abbildung 64: Platzierung Mikrocontroller	56
Abbildung 65: Platzierung Oszillatoren	56
Abbildung 66: Platzierung DS2438	57
Abbildung 67: Platzierung Reset Button	57
Abbildung 68: Platzierung 12V Step Down Converter	57
Abbildung 69: Platzierung 5V Step Down Converter	58
Abbildung 70: Platzierung 3,3V Step Down Converter	58
Abbildung 71: Platzierung Terminal Verbindung	58
Abbildung 72: Platzierung Störfilter für Mikrocontroller	59
Abbildung 73: Platzierung Störfilter für VDDA-Pin	59
Abbildung 74: Platzierung Status LEDs	59
Abbildung 75: Platzierung Versorgungsanschlüsse 1	60
Abbildung 76: Platzierung Versorgungsanschlüsse 2	60
Abbildung 77: Platzierung ESC-Verbindungen	60
Abbildung 78: Platzierung IMU-Verbindungen	61
Abbildung 79: Platzierung VTx-Verbindungen	61
Abbildung 80: Platzierung Receiver-Verbindungen	61
Abbildung 81: Platzierung Programmierschnittstelle	62
Abbildung 82: Platzierung Bootloader	62
Abbildung 83: Platzierung freie Pins	62
Abbildung 84: Pinbelegung Flight Controller	63
Abbildung 85: Bestückungsplan Top Layer	64
Abbildung 86: Bestückungsplan Bottom Layer	65
Abbildung 87: 5V Betrieb	66
Abbildung 88: T-Motor F60PROV-LV [TMF60]	68
Abbildung 89: Motor Innenaufbau [BMIA]	68
Abbildung 90: Mechanische Daten Motor [F60MD]	68
Abbildung 91: Motortestdaten im Betrieb [F60MD]	69
Abbildung 92: Zugkraft des Motors [F60MD]	69
Abbildung 93: T Motor V45A LITE [V24AD]	70
Abbildung 94: ESC Aufbau [ESCAB]	70

Abbildung 95: ESC Pinbelegung Top Layer [V24AD]	72
Abbildung 96: ESC Pinbelegung Bottom Layer [V24AD]	72
Abbildung 97: IMU Breakoutboard [IMUD].....	73
Abbildung 98: Mechanische Daten IMU [IMUD].....	73
Abbildung 99: Schematic IMU	74
Abbildung 100: Schematic Spannungsversorgung IMU	75
Abbildung 101: Schematic Interface IMU.....	75
Abbildung 102: Schematic BMP280 IMU.....	76
Abbildung 103: Schematic MPU9250 IMU.....	76
Abbildung 104: Pinbelegung IMU [IMUD]	77
Abbildung 105: Blacksheep TBS UNIFY PRO HV [VTXD].....	78
Abbildung 106: VTx Pinbelegung	79
Abbildung 107: Caddx Ratel 2 [RATE2].....	79
Abbildung 108: Kamera Pinbelegung [RATE2].....	80
Abbildung 109: Turnigy iA6C [TIA6C].....	80
Abbildung 110: TGY iA6C Pinbelegung [TIA6C].....	81
Abbildung 111: Flussdiagramm Programmablauf	82
Abbildung 112: STM32CubeMX Einstellungen DS2438	89
Abbildung 113: One-Wire Bit 1 senden.....	90
Abbildung 114: One-Wire Bit 0 senden.....	90
Abbildung 115: One-Wire Bit empfangen	93
Abbildung 116: One-Wire Initialisierungssequenz.....	95
Abbildung 117: Registerübersicht DS2438	97
Abbildung 118: DS2438 Spannungsregister Format.....	99
Abbildung 119: Real Time System Interrupt Architektur.....	101
Abbildung 120: Fernsteuerung	103
Abbildung 121: Receiver	103
Abbildung 122: Fernsteuerung Tastenbelegung	104
Abbildung 123: Beispiel PPM-Signal	107
Abbildung 124: STM32CubeMX Einstellungen S.Bus.....	108
Abbildung 125: STM32CubeMX Einstellungen I.Bus	109
Abbildung 126: Einstellungen DMA für Receiver in STM32CubeMX.....	111
Abbildung 127: I²C Datentransfer	124
Abbildung 128: STM32CubeMX Einstellungen IMU	124
Abbildung 129: I²C Schreibzyklus.....	125
Abbildung 130: I²C Lesezyklus.....	126
Abbildung 131: Registerübersicht MPU9250	130
Abbildung 132: Registerübersicht BMP280	136
Abbildung 133: Formel Berechnung Höhe über Meeresspiegel	141
Abbildung 134: Motorregelalgorithmus Architektur	142
Abbildung 135: DShot Bit 0/1 Duty Cycle + Geschwindigkeit.....	144
Abbildung 136: DShot Beispiel Übertragung.....	144
Abbildung 137: STM32CubeMX Einstellung PID DMA Empfang	150
Abbildung 138: Terminal Übertragung und Status LEDs Architektur	155
Abbildung 139: STM32CubeMX Einstellungen Terminal	157
Abbildung 140: Terminal Bild.....	157
Abbildung 141: STM32CubeMX Einstellungen LEDs	159
Abbildung 142: Beispiel PWM-Signalverlauf.....	160

Abbildung 143: Blockschaltbild Datenübertragung.....	161
Abbildung 144: Teilblockschaltbild - Sender	162
Abbildung 145: Kamera Anschlüsse	162
Abbildung 146: OSD - Video-Settings.....	163
Abbildung 147: OSD - Image-Settings.....	163
Abbildung 148: VTx.....	164
Abbildung 149: VTx - Antenne.....	164
Abbildung 150: Antenne Richtcharakteristik [RIC1]	164
Abbildung 151: VTx – Menü [MEN1]	165
Abbildung 152: Rechteck mit 10kHz	166
Abbildung 153: Rechteck mit 2kHz	166
Abbildung 154: On Off Keying	167
Abbildung 155: Hüllkurvendemodulator.....	168
Abbildung 156: Hüllkurve	168
Abbildung 157: On Off Keying, Frequenz zu tief.....	168
Abbildung 158: Beispielhafter Datenstream.....	170
Abbildung 159: UART - Kommunikation	171
Abbildung 160: V24 - Protokoll	172
Abbildung 161: UART - Beispiel	172
Abbildung 162: Flussdiagramm Senden der Daten.....	173
Abbildung 163: CubeMX USART3 Einstellungen.....	176
Abbildung 164: UART3 DMA Einstellungen.....	177
Abbildung 165: UART3 Global Interrupt aktivieren	177
Abbildung 166: Datenempfangsmessung.....	178
Abbildung 167: Schwingverhalten des Receivers	179
Abbildung 168: Datenverfälschung der empfangenen Daten.....	179
Abbildung 169: Idle Low Fehler	180
Abbildung 170: Blockschaltbild Sender - Empfänger	182
Abbildung 171: Empfängermodul	182
Abbildung 172: FPV-Brille Vorderansicht.....	183
Abbildung 173: FPV-Brille Seitenansicht	183
Abbildung 174: Blockschaltbild Empfänger.....	184
Abbildung 175: Anschlussdiagramm Hardware mit Raspberri Pi	185
Abbildung 176: Signal nach der Datenübertragung	187
Abbildung 177: Schaltung zur Datenrückgewinnung.....	188
Abbildung 178: CVBS zu USB - Converter	189
Abbildung 179: Flussdiagramm Empfangen der Daten.....	191
Abbildung 180: Empfangen von Messdaten	198
Abbildung 181: Sendeabfolge Zeitmessung	199
Abbildung 182: Flutter Framework Logo [FFLO].....	200
Abbildung 183: StatefulWidget vs. StatelessWidget Widget [SLSF].....	202
Abbildung 184: Provider State Management Aufbau [PROV]	203
Abbildung 185: Allgemeiner Aufbau der Visualisierungsapp	204
Abbildung 186: Flussdiagramm, Userstatus (Autologin, Logout).....	205
Abbildung 187: Syncfusion Logo	206
Abbildung 188: Konzept des Willkommensscreens	207
Abbildung 189: Konzept der Loginseite	207
Abbildung 190: Konzept der Registrierungsseite	207

Abbildung 192: Konzept des Sidemenüs / Drawers	208
Abbildung 191: Konzept der Homepage / Hauptseite	208
Abbildung 193: Beispiel einer generierten Farbpalette von Realtime Colours [RTEX].....	208
Abbildung 194: Flutter Code in Realtime Colors Export [RTEX].....	209
Abbildung 195: VS-Code Logo [VSCL].....	209
Abbildung 196: Flutter Installation mit "flutter doctor" überprüfen.....	211
Abbildung 197: VS-Code, neues Flutter-Projekt erstellen	211
Abbildung 198: pub.dev Webseite.....	212
Abbildung 199: Konfiguration von "flutter_launcher_icons"	215
Abbildung 200: Ausführung des "flutter_launcher_icons"-Packages im Projektverzeichnis	216
Abbildung 201: Abgeschnittenes Icon durch Android.....	216
Abbildung 202: App-Icon der Visualisierungsapp.....	216
Abbildung 203: Splash Screen der Visualisierungsapp.....	216
Abbildung 204: Konfiguration des Packages "flutter_native_splash" im pubspec.yaml-File.....	217
Abbildung 205: Ausführung von "flutter_native_splash" bzw. Generierung der Splash Screens.....	217
Abbildung 206: Teile der Initialisierung der Visualisierungsapp	218
Abbildung 207: Willkommensbildschirm	220
Abbildung 208: Logo der Visualisierungsapp.....	221
Abbildung 209: Flussdiagramm Loginbildschirm.....	221
Abbildung 210: Ausschnitt des "Passwort vergessen"-Bildschirms	226
Abbildung 211: Anpassung der E-Mail zur Passwortzurücksetzung in Firebase	227
Abbildung 212: Passwort zurücksetzen Webseite	227
Abbildung 213: E-Mail zur Rücksetzung des Userpasswords	227
Abbildung 214: Flussdiagramm der Registrierung	228
Abbildung 215: Autorisierungsfehler, invalides E-Mail-Format	228
Abbildung 216: Homepage im Offline-Status	232
Abbildung 217: Serverdatendialog mit eingegebener IP-Adresse	233
Abbildung 218: Bildschirm nach beendetem Flug	233
Abbildung 219: GNav / Bottom Navigation Bar auf Homepage	235
Abbildung 220: Empfangene Spannungsdaten via MQTT	236
Abbildung 221: MQTT-Broker zu Client Blockschaltbild [PYLO]	236
Abbildung 222: 3D-Modell auf Homepage	247
Abbildung 223: Eingefärbtes Modell der Drohne in Blender.....	248
Abbildung 224: Video-Overlay mit Balken und Platzhalterwerten.....	252
Abbildung 225: Pausieren- und Screenshot-Button unter Live-View	254
Abbildung 226: Userprofil mit definiertem Profilbild.....	255
Abbildung 227: Liste der aufgezeichneten Flüge	258
Abbildung 228: Allgemeiner Tab eines vorherigen Fluges	261
Abbildung 229: Sidemenü mit dunklem Design	262
Abbildung 230: Sidemenü mit hellem Design	262
Abbildung 231: Help- / FAQ-Bildschirm	264
Abbildung 232: Übersicht der Firebase Services in Verbindung mit der App und Ground-Station [PYLO] [FFLO].....	266
Abbildung 233: Downloadoptionen von Node.js auf der Produktseite [NJSD]	267
Abbildung 234: Ausschnitt des Setup.exe Files von Node.js.....	267
Abbildung 235: Firebase CLI Login + Informationsanfrage	268
Abbildung 236: Firebase Login, Browser "Login Succesfull"	268
Abbildung 237: Google-Account Auswahl bei Firebase CLI Login	268
Abbildung 238: Firebase CLI erfolgreicher Login.....	269

Abbildung 239: Projektübersicht in Firebase Konsole	269
Abbildung 240: Projektübersicht mit autogenerierten Statistiken.....	270
Abbildung 241: Auswahl eines Firebase-Projekts für Verbindung mit Flutter-Projekt	271
Abbildung 242: Auswahl von Plattformen für Firebase-Projekt	271
Abbildung 243: Flutterfire Plattformkonfiguration für alle Plattformen.....	271
Abbildung 244: Ausschnitt beispielhafter Useraccounts (Google, E-Mail + Password) in Auth [FBAU]	273
Abbildung 245: Aufbau Cloud Firestore Userdaten.....	274
Abbildung 246: Hauptsammlung und Userdokumente in Firestore	275
Abbildung 247: Userdokument in Firestore.....	275
Abbildung 248: gespeicherte Flugdaten eines Users	276
Abbildung 249: Aufgesetzte Realtime Database mit Boolean Flags.....	277
Abbildung 250: Ausschnitt des Cloud-Storage in Firebase	277
Abbildung 251: Blockschaltbild Videostreaming	278
Abbildung 252: Bild der CADDFPX Ratel 2 analog Kamera [RATE]	278
Abbildung 253: Verbindungstest mit USB-Camera App	280
Abbildung 254: Bild des USB2.0 Videograbbers [CAPU].....	280
Abbildung 255: Honestech VHS to DVD 2 Software, Video-Capture Quellen und Einstellungen definieren	284
Abbildung 256: NGINX Logo [NGXL].....	286
Abbildung 257: NGINX Installation via apt.....	287
Abbildung 258: NGINX-Serverkonfiguration überprüfen	288
Abbildung 259: FFMPEG Logo [FFML].....	289
Abbildung 260: Genereller Ablauf Transkodierung FFMPEG	290
Abbildung 261: Kennwerte während einer Transkodierung mit FFMPEG	293
Abbildung 262: Livestreaminglatenz unter 1s mit niedriger Bitrate (50kbps)	300
Abbildung 263: FFMPEG Transkodierung, "Broken Pipe"-Error.....	301
Abbildung 264: Autostart video_stream.sh Nachweis	303
Abbildung 265: Hauptseite des Routers	305
Abbildung 266: Firewalleinstellungen öffnen.....	305
Abbildung 267: Neue Portweiterleitung freischalten.....	306
Abbildung 268: Freigeschaltete RTMP-Ports für den Raspberry Pi.....	306
Abbildung 269: APN-Einstellung im Router	307
Abbildung 270: Erfolgreiche Portweiterleitung auf portchecker.co	307
Abbildung 271: VLC-Player Hauptmenü + Medien Menüpunkt	308
Abbildung 272: Beispiel RTMP-Netzwerkstream öffnen.....	309
Abbildung 273: Messverfahren Latenz zwischen Kamera und Stream.....	309
Abbildung 274: Videostream auf Groundstation via VLC abgreifen; FLV-Codec	310
Abbildung 275: Abgreifen des RTMP-Streams von Gerät in lokales Netz + Latenzmessung	310
Abbildung 276: Test über externes Netz; VLC-Player auf Android.....	311
Abbildung 277: öffentliche IP-Adresse herausfinden mit whatismyipaddress.com [WMIP].....	311
Abbildung 278: Videostream in der Visualisierungsapp mit Platzhalterwerten im Video-Overlay.....	312
Abbildung 279: RTMP-Stream von Drohne in der Luft	314
Abbildung 280: RTMP-Stream von Drohne am Boden.....	314
Abbildung 281: HTTP/DASH-Stream von Drohne am Boden.....	314
Abbildung 282: STM32CubeMX access to MCU selector	318
Abbildung 283: STM32CubeMX Liste von Mikrocontroller	319
Abbildung 284: STM32CubeMX Auswahl Mikrocontroller	319
Abbildung 285: STM32CubeMX grafische Oberfläche	320
Abbildung 286: STM32CubeMX HSE/LSE Einstellung	320

Abbildung 287: STM32CubeMX clock configuration	321
Abbildung 288: STM32CubeMX project settings.....	321
Abbildung 289: STM32CubeMX register callback.....	322
Abbildung 290: STM32CubeMX Ordnerstruktur	322
Abbildung 291: µVision Zauberstab.....	323
Abbildung 282: µVision Target Einstellungen	323
Abbildung 293: µVision Debug Einstellungen	323
Abbildung 294: µVision Build/Flash Knöpfe	323
Abbildung 295: µVision Ordnerstruktur.....	324
Abbildung 296: Visual Studio Code Ordnerstruktur.....	324
Abbildung 288: UI Fusion 360	325
Abbildung 289: Skizze anfertigen	326
Abbildung 290: Extrude	326
Abbildung 291: Extrude Einstellungen.....	327
Abbildung 292: Grafiken Einfügen	327
Abbildung 293: UltiMaker Cura UI	328
Abbildung 294: Einstellungen	328
Abbildung 297: 6S LIPO Akku Sideview	329
Abbildung 298: Ladegerät	329
Abbildung 299: Ladegerät Ports	330
Abbildung 300: Akku an Ladeport anschließen.....	330
Abbildung 301: Akku an Balancer Port anschließen	330
Abbildung 302: Ladegerät Bildschirm nach Akku anstecken.....	331
Abbildung 303: Ladegerät Bildschirm Ladedaten 1.....	331
Abbildung 304: Ladegerät Bildschirm Ladedaten 2.....	332
Abbildung 305: Ladegerät starte Ladevorgang	332
Abbildung 306: Akku Ladevorgang	333
Abbildung 307: Ladegerät Ladevorgang abgeschlossen	333
Abbildung 308: FPV-Drohne Inbetriebnahme Ausgangssituation.....	334
Abbildung 309: FPV-Drohne Inbetriebnahme Akku montiert.....	334
Abbildung 310: FPV-Drohne Inbetriebnahme Akku angeschlossen	335
Abbildung 311: FPV-Drohne Inbetriebnahme Reset Button.....	335
Abbildung 312: FPV-Drohne Inbetriebnahme Initialisierungsvorgang	336
Abbildung 313: FPV-Drohne Inbetriebnahme Initialisierung fehlgeschlagen	336
Abbildung 314: FPV-Drohne Inbetriebnahme Initialisierung erfolgreich	337
Abbildung 315: Projektplan Bieder & Lendl	338
Abbildung 316: Projektplan Heinicke & Hinterberger	339