**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 Smartphone & 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**

Hollabrunn, am 02.04.2024

**HINWEISE**

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

oder

Die vorliegende Diplomarbeit wurde für die Abteilung Elektronik und Technische Informatik der HTL Hollabrunn 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

DANKSAGUNGEN

Reisinger, Wihsböck, Kauer, Stoll, Dronetech Austria (Daniel Stoiber),

**DIPLOMARBEIT**

**DOKUMENTATION**

|  |  |
| --- | --- |
| Namen der  Verfasser/innen |  |
| Jahrgang  Schuljahr |  |
| Thema der Diplomarbeit |  |
| Kooperationspartner |  |

|  |  |
| --- | --- |
| Aufgabenstellung |  |

|  |  |
| --- | --- |
| Realisierung |  |

|  |  |
| --- | --- |
| Ergebnisse |  |

|  |  |
| --- | --- |
| Typische Grafik, Foto etc.  (mit Erläuterung) |  |

|  |  |
| --- | --- |
| 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) |  |
| Form  Academic year |  |
| Topic |  |
| Co-operation partners |  |

|  |  |
| --- | --- |
| Assignment of tasks |  |

|  |  |
| --- | --- |
| Realisation |  |

|  |  |
| --- | --- |
| Results |  |

|  |  |
| --- | --- |
| Illustrative graph, photo  (incl. explanation) |  |

|  |  |
| --- | --- |
| Participation in competitions  Awards |  |

|  |  |
| --- | --- |
| Accessibility of  final project thesis | HTL Hollabrunn  Anton Ehrenfriedstraße 10  2020 Hollabrunn |

|  |  |  |
| --- | --- | --- |
| Approval  (Date / Signature) | Examiner/s | Head of Department / College |

DA Antrag und unterschriebene Erklärung aus der Diplomarbeitsdatenbank einfügen

Inhaltsverzeichnis

[1 Einleitung 16](#_Toc159094522)

[1.1 Projektziel 16](#_Toc159094523)

[1.2 Gesamtüberblick 16](#_Toc159094524)

[2 Grundlagen Drohnenflug 17](#_Toc159094525)

[2.1 Inertial Measurement Unit (IMU) 17](#_Toc159094526)

[2.1.1 Gyroskop 17](#_Toc159094527)

[2.1.2 Accelerometer 17](#_Toc159094528)

[2.1.3 Magnetometer 17](#_Toc159094529)

[2.1.4 Barometer 17](#_Toc159094530)

[2.2 Lagewinkel 17](#_Toc159094531)

[2.2.1 Throttle 17](#_Toc159094532)

[2.2.2 Pitch 17](#_Toc159094533)

[2.2.3 Roll 17](#_Toc159094534)

[2.2.4 Yaw 17](#_Toc159094535)

[2.2.5 Komplementärfilter 18](#_Toc159094536)

[2.3 PID-Regler 19](#_Toc159094537)

[2.3.1 P-Glied 20](#_Toc159094538)

[2.3.2 I-Glied 20](#_Toc159094539)

[2.3.3 D-Glied 20](#_Toc159094540)

[2.3.4 PID-Glied 20](#_Toc159094541)

[2.3.5 Einstellen der Regelparameter 20](#_Toc159094542)

[3 Mechanischer Aufbau 21](#_Toc159094543)

[3.1.1 FPV – Drohne Gesamtaufbau 21](#_Toc159094544)

[3.1.2 3D – Modelle 21](#_Toc159094545)

[3.1.2.1 Propellerschutz 21](#_Toc159094546)

[3.1.2.2 Groundstation 21](#_Toc159094547)

[4 Elektronik FPV-Drohne (BIE) 22](#_Toc159094548)

[4.1 Allgemeines 22](#_Toc159094549)

[4.1.1 Grundlegendes Hardwarekonzept 22](#_Toc159094550)

[4.1.2 Anforderungen 22](#_Toc159094551)

[4.1.3 Komponentenauswahl 22](#_Toc159094552)

[4.2 Flight Controller 22](#_Toc159094553)

[4.2.1 Allgemeines 22](#_Toc159094554)

[4.2.2 Spannungsversorgungskonzept 22](#_Toc159094555)

[4.2.3 Mikrocontroller 22](#_Toc159094556)

[4.2.4 Altium PCB Design 22](#_Toc159094557)

[4.2.5 Pinbelegung 22](#_Toc159094558)

[4.3 Electronic Speed Controller (ESC) 22](#_Toc159094559)

[4.3.1 Übersicht 22](#_Toc159094560)

[4.3.2 Aufbau 22](#_Toc159094561)

[4.3.3 Funktionsweise 22](#_Toc159094562)

[4.3.4 Technische Daten 22](#_Toc159094563)

[4.4 Motoren 22](#_Toc159094564)

[4.4.1 Übersicht 22](#_Toc159094565)

[4.4.2 Aufbau 22](#_Toc159094566)

[4.4.3 Funktionsweise 22](#_Toc159094567)

[4.4.4 Technische Daten 22](#_Toc159094568)

[4.5 Receiver Fernsteuerung 22](#_Toc159094569)

[4.5.1 Übersicht 22](#_Toc159094570)

[4.5.2 Funktionsweise 22](#_Toc159094571)

[4.5.3 Technische Daten 22](#_Toc159094572)

[4.6 Video Transmitter (VTx) 22](#_Toc159094573)

[4.6.1 Übersicht 22](#_Toc159094574)

[4.6.2 Funktionsweise 22](#_Toc159094575)

[4.6.3 Technische Daten 22](#_Toc159094576)

[4.7 Live Kamera 22](#_Toc159094577)

[4.7.1 Übersicht 23](#_Toc159094578)

[4.7.2 Funktionsweise 23](#_Toc159094579)

[4.7.3 Technische Daten 23](#_Toc159094580)

[4.8 VR-Brille 23](#_Toc159094581)

[4.8.1 Übersicht 23](#_Toc159094582)

[4.8.2 Funktionsweise 23](#_Toc159094583)

[5 Steuerungssoftware 24](#_Toc159094584)

[5.1 Einführung HAL-Library und STM32CubeMX 24](#_Toc159094585)

[5.2 Softwarearchitektur 24](#_Toc159094586)

[5.3 Fernsteuerung 24](#_Toc159094587)

[5.3.1 Konfiguration 24](#_Toc159094588)

[5.3.2 Unterstützte Protokolle 26](#_Toc159094589)

[5.3.2.1 PPM 26](#_Toc159094590)

[5.3.2.2 S.Bus 26](#_Toc159094591)

[5.3.2.3 I.Bus 27](#_Toc159094592)

[5.3.3 Empfängersoftware 28](#_Toc159094593)

[5.4 Inertial Measurement Unit (IMU) 29](#_Toc159094594)

[5.4.1 I²C Protokoll 29](#_Toc159094595)

[5.4.2 Verbindungstest 33](#_Toc159094596)

[5.4.3 Gyroskop und Accelerometer - MPU9250 34](#_Toc159094597)

[5.4.3.1 Initialisierung 34](#_Toc159094598)

[5.4.3.2 Einlesen der Daten 36](#_Toc159094599)

[5.4.3.3 Komplementärfilter 38](#_Toc159094600)

[5.4.4 Luftdrucksensor - BMP280 38](#_Toc159094601)

[5.4.4.1 Initialisierung 38](#_Toc159094602)

[5.4.4.2 Einlesen der Daten 39](#_Toc159094603)

[5.4.4.3 Berechnung der Höhe 39](#_Toc159094604)

[5.4.5 Magnetometer - AK8963 39](#_Toc159094605)

[5.4.5.1 Initialisierung 39](#_Toc159094606)

[5.4.5.2 Einlesen der Daten 39](#_Toc159094607)

[5.5 Smart Battery Monitor - DS2438 39](#_Toc159094608)

[5.5.1 One-Wire Protokoll 41](#_Toc159094609)

[5.5.1.1 One-Wire Schreibzyklen 41](#_Toc159094610)

[5.5.1.2 One-Wire Lesezyklen 44](#_Toc159094611)

[5.5.1.3 Initialisierungssequenz 46](#_Toc159094612)

[5.5.1.4 ROM-Funktionsbefehl 47](#_Toc159094613)

[5.5.1.5 Memory-Funktionsbefehl 47](#_Toc159094614)

[5.5.2 Spannungsüberwachung 47](#_Toc159094615)

[5.6 Abstandssensor - HCSR04 (Ben) 50](#_Toc159094616)

[5.6.1 Funktionsweise 50](#_Toc159094617)

[5.6.2 Code 50](#_Toc159094618)

[5.7 Motoransteuerung 50](#_Toc159094619)

[5.7.1 Motorstrom auslesen 50](#_Toc159094620)

[5.7.2 DShot Protokoll 50](#_Toc159094621)

[5.8 PID-Regler 50](#_Toc159094622)

[5.8.1 Hover Modus 50](#_Toc159094623)

[5.8.2 Failsafe 50](#_Toc159094624)

[5.9 Debugging 51](#_Toc159094625)

[5.9.1 Terminal Ausgabe 51](#_Toc159094626)

[5.9.2 Status – LEDs 51](#_Toc159094627)

[6 Datenübertragung der Mess- und Videodaten (Ben) 52](#_Toc159094628)

[6.1 Überblick Datenübertragung 52](#_Toc159094629)

[6.2 Kommunikation: Flight Controller und Sender 52](#_Toc159094630)

[*6.2.1 Aufbau Sendermodul (VTx)* 52](#_Toc159094631)

[6.2.2 ASK – Modulation 52](#_Toc159094632)

[6.2.3 Transmit – Programm auf Cortex µC 52](#_Toc159094633)

[6.2.3.1 Gleitkommadarstellung (memcpy) 52](#_Toc159094634)

[6.2.3.2 Code 52](#_Toc159094635)

[*6.2.4* Testen der Übertragung 52](#_Toc159094636)

[6.3 Kommunikation: Sender und Empfänger 52](#_Toc159094637)

[6.3.1 Aufbau Empfängermodul 52](#_Toc159094638)

[6.3.2 Verbindungsaufbau 52](#_Toc159094639)

[6.3.3 Testen der Übertragung 52](#_Toc159094640)

[6.4 Kommunikation: Empfänger und Raspberry Pi 52](#_Toc159094641)

[6.4.1 Empfangsprogramm auf Raspberry Pi 52](#_Toc159094642)

[6.4.1.1 UART auf dem Raspberry Pi 52](#_Toc159094643)

[6.4.1.2 Code 52](#_Toc159094644)

[6.4.1 Senden der Daten auf dem MQTT-Server 52](#_Toc159094645)

[6.5 Testen der Datenübertragungskette 52](#_Toc159094646)

[7 Visualisierungs-App 53](#_Toc159094647)

[7.1 Applikation 53](#_Toc159094648)

[7.1.1 Einführung – Dart / Flutter - Framework 53](#_Toc159094649)

[7.1.1.1 Allgemeines 53](#_Toc159094650)

[7.1.1.2 Pub Dev / Package Installer 53](#_Toc159094651)

[7.1.1.3 Dart Syntax 53](#_Toc159094652)

[7.1.1.4 State Management 53](#_Toc159094653)

[7.1.1.4.1 Stateful vs. Stateless Widgets 53](#_Toc159094654)

[7.1.1.4.2 Provider 53](#_Toc159094655)

[7.1.2 Allgemeines zur App 53](#_Toc159094656)

[7.1.2.1 Usersystem 53](#_Toc159094657)

[7.1.2.2 Datenvisualiserung 53](#_Toc159094658)

[7.1.2.3 3D-Model-Viewer 53](#_Toc159094659)

[7.1.2.4 Livestream-Viewer 53](#_Toc159094660)

[7.1.3 UI-Konzept 53](#_Toc159094661)

[7.1.4 Projektstruktur und -umgebung 53](#_Toc159094662)

[7.1.4.1 Editor – Visual Studio Code 53](#_Toc159094663)

[7.1.4.2 Flutter Installation 53](#_Toc159094664)

[7.1.4.3 Projekterstellung 53](#_Toc159094665)

[7.1.4.4 Projektstruktur 53](#_Toc159094666)

[7.1.4.5 Packages 53](#_Toc159094667)

[7.1.4.5.1 Pubspec.yaml 53](#_Toc159094668)

[7.1.4.5.2 Installieren neuer Packages 53](#_Toc159094669)

[7.1.5 Splash + Willkommensscreen 54](#_Toc159094670)

[7.1.5.1 Native Splash Screen 54](#_Toc159094671)

[7.1.5.2 Willkommensbildschirm 54](#_Toc159094672)

[7.1.5.3 Logo-Design 54](#_Toc159094673)

[7.1.6 Login + Registrierung 54](#_Toc159094674)

[7.1.6.1 E-Mail-Login 54](#_Toc159094675)

[7.1.6.2 Google-Login 54](#_Toc159094676)

[7.1.6.3 Passwort vergessen 54](#_Toc159094677)

[7.1.6.4 Registrierung 54](#_Toc159094678)

[7.1.7 Homepage 54](#_Toc159094679)

[7.1.7.1 Bottom Navigation Bar / GNav-Bar 54](#_Toc159094680)

[7.1.7.2 Serverdatendialog 54](#_Toc159094681)

[7.1.7.3 Flugdatenvisualisierung 54](#_Toc159094682)

[7.1.7.3.1 MQTT-Datenstreams 54](#_Toc159094683)

[7.1.7.3.2 Syncfusion Cartesian Charts 54](#_Toc159094684)

[7.1.7.3.3 Echtzeitdaten + Animierte Diagramme 54](#_Toc159094685)

[7.1.7.3.4 Speicherung der Daten 54](#_Toc159094686)

[7.1.7.4 3D-Model-Viewer 54](#_Toc159094687)

[7.1.7.4.1 Flutter Cube 54](#_Toc159094688)

[7.1.7.4.2 MQTT-Datenstream 54](#_Toc159094689)

[7.1.7.4.3 3D-Dronenmodell 54](#_Toc159094690)

[7.1.7.4.3.1 Optimierungen in Blender 54](#_Toc159094691)

[7.1.7.4.4 Extrabedienungen 54](#_Toc159094692)

[Live-View 54](#_Toc159094693)

[7.1.7.4.5 VLC-Plugin 54](#_Toc159094694)

[7.1.7.4.6 Darstellung eines Netzwerkstreams 54](#_Toc159094695)

[7.1.7.4.7 Video-Overlay 54](#_Toc159094696)

[7.1.8 Sidemenu / Drawer 54](#_Toc159094697)

[7.1.8.1 Userprofil 54](#_Toc159094698)

[7.1.8.1.1 Darstellung der Userdaten 54](#_Toc159094699)

[7.1.8.1.2 Profilbildauswahl 54](#_Toc159094700)

[7.1.8.1.3 Änderung der Userdaten 55](#_Toc159094701)

[7.1.8.2 Credits 55](#_Toc159094702)

[7.1.8.3 Vorherige Flüge 55](#_Toc159094703)

[7.1.8.3.1 Darstellung aller gespeicherten Flüge 55](#_Toc159094704)

[7.1.8.3.2 Sortiermöglichkeiten 55](#_Toc159094705)

[7.1.8.3.3 Einsicht in vorherigen Flug 55](#_Toc159094706)

[7.1.8.4 Einstellungen 55](#_Toc159094707)

[7.1.8.4.1 Dark- / Light-Mode 55](#_Toc159094708)

[7.1.8.4.2 3D-Model Ausrichtung 55](#_Toc159094709)

[7.1.8.5 Logout´ 55](#_Toc159094710)

[7.2 Firebase Backend 55](#_Toc159094711)

[7.2.1 Installation via Firebase CLI 55](#_Toc159094712)

[7.2.1.1.1 Firebase CLI – Setup 55](#_Toc159094713)

[7.2.1.1.2 Firebase für Dart / Flutter aktivieren 55](#_Toc159094714)

[7.2.1.1.3 Flutter App mit Flutterfire konfigurieren 55](#_Toc159094715)

[7.2.2 Einbindung in Flutter 55](#_Toc159094716)

[7.2.2.1.1 Flutterfire Configure 55](#_Toc159094717)

[7.2.2.1.2 Benutzten Firebase Packages hinzufügen 55](#_Toc159094718)

[7.2.3 Nutzen der verschiedenen Datenbanksysteme 55](#_Toc159094719)

[7.2.3.1 Auth 55](#_Toc159094720)

[7.2.3.1.1 E-Mail + Passwort – Login 55](#_Toc159094721)

[7.2.3.1.2 Google Login 55](#_Toc159094722)

[7.2.3.2 Firestore Database 55](#_Toc159094723)

[7.2.3.2.1 Struktur 55](#_Toc159094724)

[7.2.3.2.2 Allgemeine Userdaten 55](#_Toc159094725)

[7.2.3.2.3 Flugdaten 55](#_Toc159094726)

[7.2.3.2.4 Einstellungen 55](#_Toc159094727)

[7.2.3.3 Realtime Database 55](#_Toc159094728)

[7.2.3.3.1 Allgemein 55](#_Toc159094729)

[7.2.3.3.2 Sinn der verschiedenen Flags 56](#_Toc159094730)

[7.2.3.4 Storage 56](#_Toc159094731)

[7.2.3.4.1 Allgemein 56](#_Toc159094732)

[7.2.3.4.2 Referenz zu User in Firestore 56](#_Toc159094733)

[8 Videostreaming 56](#_Toc159094734)

[8.1 Allgemeiner Aufbau 56](#_Toc159094735)

[8.1.1 Anforderungen 56](#_Toc159094736)

[8.1.2 Übertragungskette 56](#_Toc159094737)

[8.2 CADFPX Analog Kamera 56](#_Toc159094738)

[8.2.1 Allgemein 56](#_Toc159094739)

[8.2.2 Produktinformationen 56](#_Toc159094740)

[8.2.3 Verwendungszweck 56](#_Toc159094741)

[8.2.4 Verbindungstest via USB-Camera App 56](#_Toc159094742)

[8.3 USB2.0 VHS Video Grabber 56](#_Toc159094743)

[8.4 RTMP-Server via NGINX aufsetzen 56](#_Toc159094744)

[8.4.1 RTMP Allgemein 56](#_Toc159094745)

[8.4.2 NGINX 56](#_Toc159094746)

[8.4.2.1 Allgemein 56](#_Toc159094747)

[8.4.2.2 Installation 56](#_Toc159094748)

[8.4.2.3 RTMP-Konfiguration 56](#_Toc159094749)

[8.4.2.4 Serverstatus einsehen 56](#_Toc159094750)

[8.5 RTMP-Stream erstellen 56](#_Toc159094751)

[8.5.1 FFMPEG Allgemein 56](#_Toc159094752)

[8.5.2 Aufbau der Konvertierung 56](#_Toc159094753)

[8.5.2.1 Simple Konvertierung 56](#_Toc159094754)

[8.5.2.2 Optimierungen 56](#_Toc159094755)

[8.5.2.2.1 Encodervergleich 56](#_Toc159094756)

[8.5.2.2.2 Optimierungsflags 56](#_Toc159094757)

[8.5.2.2.3 Unoptimiert vs. Optimiert 57](#_Toc159094758)

[9 Videoserver-Port Forwarding 57](#_Toc159094759)

[9.1 Allgemein 57](#_Toc159094760)

[9.2 Routereinstellungen 57](#_Toc159094761)

[9.3 Kontakt mit Routerfirma zur Freischaltung 57](#_Toc159094762)

[9.4 Testen der Ports 57](#_Toc159094763)

[9.4.1 Portchecker.co 57](#_Toc159094764)

[9.5 Testen des Videostreams 57](#_Toc159094765)

[9.5.1 VLC-Player 57](#_Toc159094766)

[9.5.1.1 Devicestream direkt testen 57](#_Toc159094767)

[9.5.1.2 Netzwerkstream im selben Netzwerk 57](#_Toc159094768)

[9.5.1.3 Netzwerkstream via Public IP 57](#_Toc159094769)

[10 Ergebnisse 57](#_Toc159094770)

[11 Anhang 58](#_Toc159094771)

[11.1 Einführung CAD – Software (Fusion 360) (Ben) 58](#_Toc159094772)

[*11.1.1* UI und Projekterstellung 58](#_Toc159094773)

[*11.1.2* Skizze anfertigen 58](#_Toc159094774)

[*11.1.3* Körper erstellen 58](#_Toc159094775)

[*11.1.4* Schrift und Bilder einfügen 58](#_Toc159094776)

[11.2 3D – Druck (Ben) 58](#_Toc159094777)

[*11.2.1* 3D – Drucker 58](#_Toc159094778)

[*11.2.2* Filamente 58](#_Toc159094779)

[11.2.3 3D – Drucker Software (Ultimaker Cura) 58](#_Toc159094780)

[11.3 Inbetriebnahme Anleitung 58](#_Toc159094781)

[11.4 Projektplan 58](#_Toc159094782)

[11.5 Projektkosten 58](#_Toc159094783)

[11.6 Projekttagebuch 58](#_Toc159094784)

[12 Quellen 59](#_Toc159094785)

[12.1 Gedruckte Medien 59](#_Toc159094786)

[12.2 Online 59](#_Toc159094787)

[13 Verzeichnis der Abbildungen 60](#_Toc159094788)

[14 Begleitprotokoll 61](#_Toc159094789)

[15 Anhang 62](#_Toc159094790)

[15.1 — Projektdokumentation (Kostendarstellung, Besprechungsprotokolle etc.) 62](#_Toc159094791)

[15.2 — Technische Dokumentation (technische Beschreibungen, Berechnungen, 62](#_Toc159094792)

[15.3 Konstruktionszeichnungen, Versuchsberichte, betriebswirtschaftliche Kalkulationen etc.) 62](#_Toc159094793)

[15.4 Schaltungen, Zeichnungssätze, sonstiges 62](#_Toc159094794)

# Einleitung

## Projektziel

## Gesamtüberblick

# Grundlagen Drohnenflug

## Inertial Measurement Unit (IMU)

### Gyroskop

### Accelerometer

### Magnetometer

### Barometer

## Lagewinkel

### Throttle

Throttle beschreibt die Geschwindigkeit aller vier Motoren. Ein anderer Begriff für Throttle ist der Motorschub. Bei der Angabe von zum Beispiel 80% Throttle ist gemeint, dass alle Motoren sich mit 80% Geschwindigkeit drehen. Wenn der Wert hoch ist, fliegt die Drohne nach oben. Bei einem niedrigen Wert fliegt die Drohne nach unten.

Die Motoren und Propellern müssen genug Schub erzeugen, damit das Gesamtgewicht der Drohne angehoben und manövriert werden kann. Die Throttleangabe ist vor allem wichtig bei einem Hover-Modus. Dabei regelt der Quadrokopter die einzelnen Geschwindigkeiten der Motoren, um sich selber aufrecht zu halten.

### Pitch

Bei Pitch handelt es sich um den Neigungswinkel nach vorne und hinten, wenn die Drohne von hinten angeschaut wird. Mit der Angabe vom Pitch kann der Quadrokopter nach vorne und nach hinten gelenkt werden.

Bild Drohne mit Pitch winkel

Vielleicht mehr physikalisch erklären

### Roll

Bei Roll handelt es sich um den Neigungswinkel nach rechts und links, wenn die Drohne von hinten angeschaut wird. Mit der Angabe vom Roll kann der Quadrokopter nach rechts und links gesteuert werden.

Bild Drohne mit Roll winkel

Vielleicht mehr physikalisch erklären

### Yaw

Bei Yaw handelt es sich um die Drehung um die eigene Achse nach links oder rechts. Mit der Angabe vom Yaw kann der Quadrokopter nach links und rechts drehen.

Bild Drohne mit Yaw winkel

Vielleicht mehr physikalisch erklären

### Komplementärfilter

**Accelerometer Winkel**:

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

accelerometerPitch/Roll … jeweiliger berechneter Winkel in Radiant

accelerometerX/Y/Z … Accelerometer Messwert der jeweiligen Achse in g

Der Yaw-Winkel kann nicht berechnet werden, da die Yaw-Bewegung in der Ebene, um die Z-Achse, stattfindet und daher unabhängig von dem 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 Radiant, benötigt. Daher müssen die Ergebnisse mit einem Faktor von multipliziert werden.

Problem:

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

**Gyroskop Winkel**:

Durch die Integration der Winkelbeschleunigungswerte über der Zeit ist es möglich Lagewinkel zu bestimmen:

gyroscopePitch/Roll/Yaw … jeweiliger berechneter Winkel in Radiant

gyroscopeX/Y/Z … Gyroskop Messwert der jeweiligen Achse in Grad pro Sekunde

dt … Zeitbereich in Sekunden

Problem:

Durch die Integration werden Messungenauigkeit aufsummiert, die zu einen 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 einen 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.

Winkel … Pitch/Roll-Winkel in Grad

α … Filterkoeffizient

gyroscopeData … Gyroskop Messwerte der jeweiligen Achse in Grad pro Sekunde

dt … Abtastzeit in Sekunden

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% gewichtet wird und die Accelerometer-Messung zu 2%. Daraus folgt:

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

## 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 wieder zurückgekoppelt und mit dem Sollwert verglichen. Dieser Vergleich ergibt eine Regelabweichung, die das in den Regelalgorithmus geleitet wird. Das Ziel einer Regelung ist, die Regelabweichung zu verschwinden zu bringen.

In einer Drohne wird der PID-Regler verwendet, um die Drohne während des Fluges stabil zu halten. Damit soll die Drohne bei zum Beispiel Windstößen sich wieder selbst richtig ausrichten und den normalen Flug weiterführen.

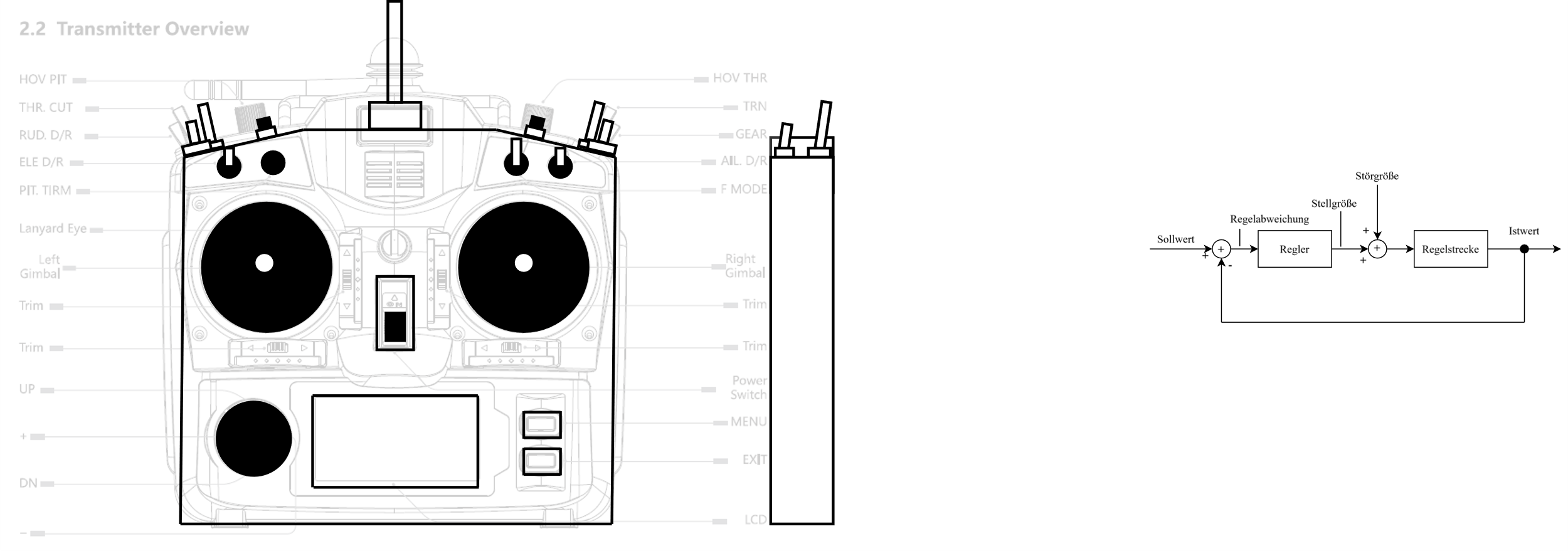


Abbildung 1: Regler-Blockschaltbild

Sollwert … die gewünschte Einstellung, die von Regler gehalten werden soll

Regelabweichung … Differenz zwischen Soll- und Istwert

Regler …

Stellgröße …

Störgröße …

Regelstrecke …

Istwert … die aktuelle Einstellung, 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:

### P-Glied

Blockschaltbild + Formel + Diagramm bei Sprungantwort + Vor/Nachteile

### I-Glied

Blockschaltbild + Formel + Diagramm bei Sprungantwort + Vor/Nachteile

### D-Glied

Blockschaltbild + Formel + Diagramm bei Sprungantwort + Vor/Nachteile

### PID-Glied

Blockschaltbild + Formel + Diagramm bei Sprungantwort + Vor/Nachteile

### Einstellen der Regelparameter

Ablauf der Einstellung der Bestimmung der Parameter

# Mechanischer Aufbau

### FPV – Drohne Gesamtaufbau

### 3D – Modelle

#### Propellerschutz

#### Groundstation

# Elektronik FPV-Drohne (BIE)

## Allgemeines

### Grundlegendes Hardwarekonzept

### Anforderungen

### Komponentenauswahl

## Flight Controller

### Allgemeines

### Spannungsversorgungskonzept

### Mikrocontroller

### Altium PCB Design

### Pinbelegung

## Electronic Speed Controller (ESC)

### Übersicht

### Aufbau

### Funktionsweise

### Technische Daten

## Motoren

### Übersicht

### Aufbau

### Funktionsweise

### Technische Daten

## Receiver Fernsteuerung

### Übersicht

### Funktionsweise

### Technische Daten

## Video Transmitter (VTx)

### Übersicht

### Funktionsweise

### Technische Daten

## Live Kamera

### Übersicht

### Funktionsweise

### Technische Daten

## VR-Brille

### Übersicht

### Funktionsweise

# Steuerungssoftware

## Einführung HAL-Library und STM32CubeMX

Erklärung was ist HAL und STM32CubeMX

Projekterstellung

Setup mit µVision

## Softwarearchitektur

Übersicht Blockschaltbild

+ Erklärung

## 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.

Bild Tranmitter und Receiver

Für das Projekt wurde der Turnigy 9X 9Ch Transmitter mit dem TGY-iA6C Receiver ausgewählt, da diese eine große Auswahl an Kanälen, Schalter und digitalen Signalausgang ermöglichen.

Erklärung Layout der Knöpfe + LED an Receiver

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!“ steht. Dieser Fehler wird angegeben, wenn die Schalter nicht auf ihrer 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.

Bild das das erklärt

### Konfiguration

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, schalte 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 dies nicht der Fall ist, muss die Fernsteuerung mit dem Receiver neu gekoppelt werden:

1. Schalte Fernsteuerung und 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.

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

Die dargestellten Einstellungen sind die empfohlenen Werte. Die Einstellungen Stick Set und Output Select kann vom Benutzer in der Software festgelegt werden. *siehe Stick Set Änderung* und *Output Select Änderung*

|  |  |  |
| --- | --- | --- |
| **Function Setting** | | |
| AUX-CH  (für Ein-/Ausschalter) | CH5 Gear | |
| PROG. MIX  (für 3-Wege-Schalter) | 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 | |

Die restlichen Einstellungen haben die Standardwerte und müssen nicht verändert werden.

### Unterstützte Protokolle

Der Receiver unterstützt eine parallele digitale und analoge Signalausgabe. Ein Pin gibt immer ein PPM-Signal aus, während der andere Pin entweder ein S.Bus oder I.Bus Signal ausgibt. Die Ausgabe kann in den Fernsteuerungseinstellungen festgelegt werden. *(*[*siehe: 5.3.1 Konfiguration*](#_Konfiguration)*)*

#### PPM

Das PPM-Signal ist das einzige analoge Signal, das der Receiver ausgibt. Die Daten werden mit der Timer-Peripherie eingelesen, indem die Länge des high-Pegels bestimmt wird. Die Daten kommen in 12ms langen Paketen mit einer 8,1ms high-aktiven Ruhezustand. Die Daten befinden sich in den Bereich 0,5ms bis 1,5ms.

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 die Steuerung verwendet.

#### 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 Protokoll PPM und weniger fehleranfällig als das I.Bus-Protokoll, da das Signal invertiert ist. 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:

Bilder I guess oder Tabelle von Einstellungen

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:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Byte** | 0 | 1 | 2 | 3 | … | 23 | 24 |
| **Daten**  header  0b001 0011 0001  Kanal 1  0b001 1001 0001  Kanal 2  flags  footer | 0x0F | 0b00110001 | 0b00110001 | 0b00110001 | … | 0b00000000 | 0x00 |

#### 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 keine Signalverlust festgestellt werden, da dieses keine Flags besitzt. Der digitale Werteberiech befindet zwischen 1070 und 1920.

UART-Konfiguration mit STM32CubeMX:

Bilder I guess oder Tabelle von Einstellungen

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 weniger der Summe der ersten 30 Bytes.

Beispiel für die Dekodierung:

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

### Empfängersoftware

Programm oder Programm + Programmerklärung

## Inertial Measurement Unit (IMU)

### I²C Protokoll

Das I²C Protokoll besteht aus drei Leitungen: SDA (Datenleitung), SCL (Taktleitung) und GND.

Bevor jeder Datenübertragung 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 einen Duty Cycle von 50% auf die SCL-Leitung. Zuerst wird auf der SDA-Leitung wird die I²C-Addresse des I²C-Slaves geschickt. Danach werden gleichzeitig die Daten gesendet beziehungsweise empfangen und bei einer aktiven high-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, kam 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 einen low-Zustand auch in den high-Ruhezustand.

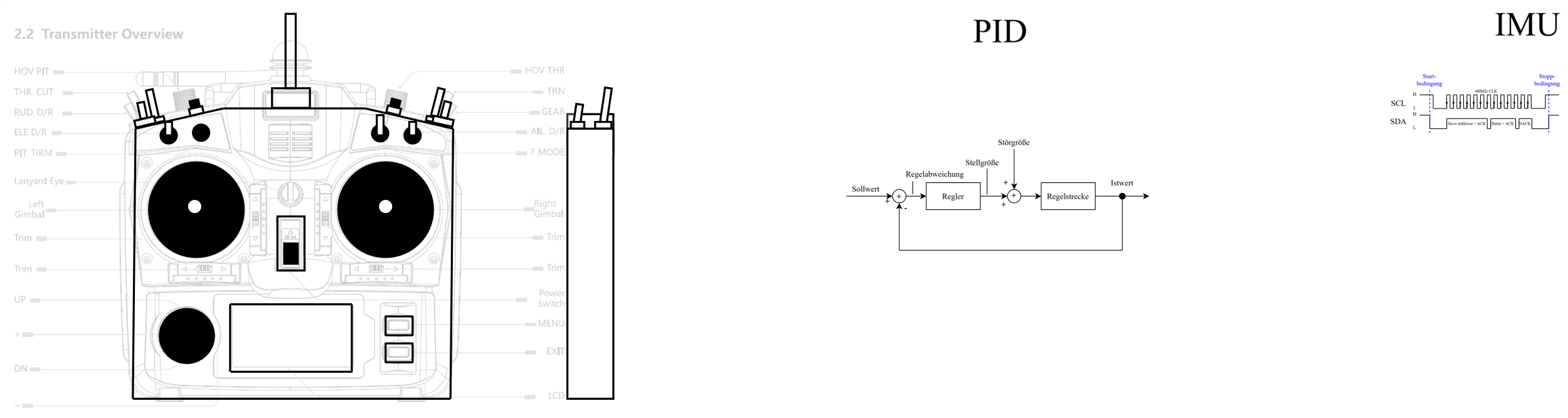


Abbildung 2: I²C Datentransfer

I²C Einstellung in STM32CubeMX:

Ein Bild, das Text, Screenshot, Software, Webseite enthält.

Automatisch generierte Beschreibung

Abbildung 3: STM32CubeMX Einstellungen IMU

Auf der Flight-Controller Platine ist der Pin PB7 für SDA-Leitung und der Pin PB6 für die SCL-Leitung festgelegt.

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. Bei Kommunikation mit Sensoren muss ein achtes Bit (R/W Bit) zur Adresse hinzugefügt werden, welches bei Schreibzyklen auf 0 gesetzt werden muss 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:

Schreibzyklus:

Ein Bild, das Text, Screenshot, Reihe, Schrift enthält.

Automatisch generierte Beschreibung

Abbildung 4: 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:

|  |
| --- |
| 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()* schickt automatisch den richtigen Ablauf für die Kommunikation.

Lesezyklus:

Ein Bild, das Text, Reihe, Screenshot, Schrift enthält.

Automatisch generierte Beschreibung

Abbildung 5: 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 | Stoppbedingung |

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.

|  |
| --- |
| 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()* schickt automatisch den richtigen Ablauf für die Kommunikation.

### Verbindungstest

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

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

Mit der Funktion *IMU\_CheckConnection()* werden alle drei Sensoren auf diese Werte überprüft und wenn ein Wert nicht übereinstimmt, wird ein *IMU\_<Sensor>\_ADDRESS\_ERROR* ausgegeben.

|  |
| --- |
| 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(10000);          }      }      return IMU\_OK;  } |

Damit der Magnetometer und auf das Barometer 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 auf den anderen Sensoren kann zugegriffen werden.

### Gyroskop und Accelerometer - MPU9250

#### Registerübersicht

Bild von allen wichtigen Registern + vielleicht Beschreibung

#### Initialisierung

Initialisierung vom Accelerometer und Gyroskop im Programm:

|  |
| --- |
| IMU\_10DOF.c | Ausschnitt aus IMU\_Init() Funktion |
| /\*\*   \* @brief This function initialzes 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;  } |

Bemerkung:

Die Punkte ... bedeuten, dass vor und nach dem Ausschnitt noch Programmcode steht.

Die Messwerte eines Gyroskops driften bei Bewegung in einer Richtung *(siehe: Gyroskop Erklärung vom Drift)*. 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.

Beispiel Aufruf vom Programm:

|  |
| --- |
| main.c |
| // init IMU 10DOF  IMU\_InitTypeDef imuInit;  imuInit.hi2c = &hi2c1;  imuInit.gyroFS = GYRO\_500DPS;  imuInit.accelFS = ACCEL\_16G;  imuInit.gyroDLPF = GYRO\_DLPF\_10HZ;  imuInit.accelDLPF = ACCEL\_DLPF\_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;  imuInit.dt = dt;  errorCode = IMU\_Init(&imuInit);  if(errorCode != IMU\_OK)      Sensor\_ErrorHandler(IMU, errorCode); |

#### Einlesen der 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:

|  |
| --- |
| 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];        // invert z-axis because the sensor is upside down      accelData.z = -accelData.z;      return accelData;  } |

In dem Programm werden die Daten beginnend mit dem Register 59 nacheinander bis zum Register 64 eingelesen und danach zusammengefügt. Die z-Achse des Accelerometers muss invertiert werden, weil das IMU-Breakout auf der FPV-Drohne umgedreht angebracht ist.

Die Messung des Gyroskops funktioniert gleich mit dem Accelerometer. Die Messdaten des werden in den Registern 67 GYRO\_XOUT\_H bis zum Register 72 GYRO\_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 Gyroskop-Daten:

|  |
| --- |
| 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: 5.4.3.2 Initialisierung)*](#_Initialisierung).

Um die Drift vom Gyroskop entgegenzuwirken wird der Registerwert noch mit dem Gyroskop-Offset gerechnet werden. Diese Werte werden auch im Initialisierungsprogramm bestimmt und in der globalen Variable ‚gyroOffset‘ gespeichert [*(siehe: 5.4.3.2 Initialisierung)*](#_Initialisierung).

Berechnung der eigentlichen Messwerte:

|  |
| --- |
| IMU\_RegCoordinates gyroData = IMU\_MPU\_ReadGyro();  IMU\_RegCoordinates accelData = IMU\_MPU\_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) - 0.020f;  accel.y = (accelData.y / accelSens) - 0.021f;  accel.z = (accelData.z / accelSens) - 0.140f; |

Die Offsetwerte bei der Beschleunigungsberechnung (0,020; 0,021 und 0,140) müssen händisch bestimmt werden. Wenn der Sensor auf einer ebenen Fläche gerade liegt, muss der Sensor eine Beschleunigung von 1g auf der z-Achse messen. Der Offset ergibt sich aus der eigentlichen Messung auf einer ebenen Fläche von 1,14g. Dieser Vorgang muss mit 90° Drehungen mit allen Achsen wiederholt werden, um die restlichen Offsetwerte zu bestimmen.

#### Komplementärfilter

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

|  |
| --- |
| 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)  {      IMU\_RegCoordinates gyroData = IMU\_MPU\_ReadGyro();      IMU\_RegCoordinates accelData = IMU\_MPU\_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) - 0.020f;      accel.y = (accelData.y / accelSens) - 0.021f;      accel.z = (accelData.z / accelSens) - 0.140f;      // Complementary filter      float accelPitch = atan2(accel.y, accel.z) \* RAD2DEG;      float accelRoll = atan2(accel.x, accel.z) \* RAD2DEG;      angle.roll = 0.98 \* (angle.roll - gyro.y \* dt) + (1.0 - 0.98) \* accelRoll;      angle.pitch = 0.98 \* (angle.pitch + gyro.x \* dt) + (1.0 - 0.98) \* accelPitch;      angle.yaw += gyro.z \* dt;  } |

Bemerkung:

Die aktuellen Werte für Pitch, Roll und Yaw werden in der globalen Variable ‚angle‘ gespeichert.

### Luftdrucksensor - BMP280

#### Registerübersicht

Bild von allen wichtigen Registern + vielleicht Beschreibung

#### Initialisierung

Initialisierung vom Barometer:

|  |
| --- |
| IMU\_10DOF.c |
| /\*\*   \* @brief This function initialzes 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 status of device      uint8\_t timeout = 0, status = 1;      while(status != 0x00)      {          IMU\_ReadRegister(BMP280, IMU\_BARO\_STATUS\_ADDR, &status, 1);          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);      }      baroAltitudeOffset = baroSum / amount;      return IMU\_OK;  } |

Das Barometer wird immer im Normal-Mode betrieben. Für die Werteberechnung müssen die Kompensationswerte mit der Funktion *IMU\_Baro\_ReadCompensationValues()* bestimmt werden.

|  |
| --- |
| 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 befinden sich in den Registern 0x88 dig\_T1 bis zu den Register 0x9F dig\_P9 [*(siehe: 5.4.4.1 Registerübersicht)*](#_Registerübersicht). Die Werte werden zusammengefügt und in der globalen Variable ‚baroCompensation‘ gespeichert.

#### Einlesen der Daten

Die Messdaten vom BMP280 werden in den Registern 0xF7 press\_msb bis 0xFC temp\_xlsb [*(siehe: 5.4.4.1 Registerübersicht)*](#_Registerübersicht).

|  |
| --- |
| IMU\_10DOF.c |
| /\*\*   \* @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);      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.

|  |
| --- |
| 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;  } |

#### Berechnung der Höhe

Mit dem gemessenen Luftdruck kann die aktuelle Seehöhe abgeschätzt werden.

Formelerklärung + Formel im ReadBaro() Progamm

Problem mit Höhenmessung mit Barometer (langsam und so)

### Magnetometer - AK8963

#### Registerübersicht

Bild von allen wichtigen Registern + vielleicht Beschreibung

#### Initialisierung

Init Mag Programm

#### Einlesen der Daten

ReadMag Programm

Fehlererklärung

## Smart Battery Monitor - DS2438

In der Diplomarbeit wird der DS2438 für die Spannungsüberwachung des Akkus verwendet. Die Kommunikation zwischen dem Flight-Controller und dem DS2438 findet mit dem One-Wire Protokoll statt.

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

Ein Bild, das Text, Karte Menü, Zahl, Dokument enthält.

Automatisch generierte Beschreibung

Abbildung 6: DS2438 Memory Map

Für die Spannungsüberwachung ist nur die Page 0 wichtig, da im Byte[0] Einstellungen gesetzt werden können und in den Byte[3..4] die gemessenen Spannungswerte gespeichert werden.

GPIO-Einstellungen in STM32CubeMX:

Ein Bild, das Text, Screenshot, Software, Zahl enthält.

Automatisch generierte Beschreibung

Abbildung 7: STM32CubeMX Einstellungen von DS2438

Der Pin PC0 wird am Flight-Controller für DS2438\_DQ (Datenleitung) verwendet. Bei anderen Systemen kann der Pin beim Aufruf von *DS2438\_Init()* bestimmt werden. *(siehe: 5.5.2 Spannungsmonitoring)*

Damit die Kommunikation funktionieren kann, muss 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.

### 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-Funktionsbefehl gesendet, und am Ende werden die Daten ausgetauscht.

#### One-Wire Schreibzyklen

Bei der Kommunikation wird das niederwertigste Bit immer zuerst gesendet.

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

Ein Bild, das Text, Diagramm, Reihe, Schrift enthält.

Automatisch generierte Beschreibung

Abbildung 8: One-Wire Bit 1 senden

Wenn das Bit 0 gesendet werden soll, müssen andere Zeitbedingungen eingehalten werden:

Ein Bild, das Text, Reihe, Diagramm, Quittung enthält.

Automatisch generierte Beschreibung

Abbildung 9: One-Wire Bit 0 senden

|  |
| --- |
| 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. Anschießend 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.

|  |
| --- |
| 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;  } |

#### One-Wire Lesezyklen

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

Ein Bild, das Text, Diagramm, Reihe, Schrift enthält.

Automatisch generierte Beschreibung

Abbildung 10: One-Wire Bit empfangen

|  |
| --- |
| 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 bestimmen Page Daten lesen zu können, müssen diese zuerst vom Sensor auf das Scratchpad geschrieben werden. Danach muss man die Daten vom Scratchpad einlesen.

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.

|  |
| --- |
| 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;  } |

#### Initialisierungssequenz

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

Ein Bild, das Text, Diagramm, Reihe, parallel enthält.

Automatisch generierte Beschreibung

Abbildung 11: One-Wire Initialisierungssequenz

Wenn der One-Wire Slave bei dem Presence-Puls nicht nach 60µs bis 240µs die Leitung in den high-Zustand setzt, besteht ein Problem bei der Kommunikation zwischen Master und Slave.

Umsetzung im Programmcode:

|  |
| --- |
| 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;  } |

#### 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.

Vllt Liste von allen Befehlen mit Beschreibung

#### Memory-Funktionsbefehl

Der einzige Memory-Funktionsbefehl, der wichtig für das Spannungsmonitoring wichtig 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) auf 0 gesetzt.

Vllt Liste von allen Befehlen mit Beschreibung

### Spannungsüberwachung

Initialisierung von DS2438:

|  |
| --- |
| 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;  } |

Man kann sich aussuchen, ob man die Versorgungsspannung oder die Spannung am Eingang vom A/D-Wandler messen will. 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.

Beispiel Aufruf vom Programm:

|  |
| --- |
| main.c |
| errorCode = DS2438\_Init(&htim16, GPIOC, GPIO\_PIN\_0);  if(errorCode != DS2438\_OK)  Sensor\_ErrorHandler(DS2438, errorCode); |

Die eingelesen Spannungswerte werden in folgendem Format gespeichert:

Ein Bild, das Text, Schrift, Zahl, Reihe enthält.

Automatisch generierte Beschreibung

Abbildung 12: 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.

|  |
| --- |
| 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. Um die richtige Akkuspannung zu bekommen, muss der eingelesen Spannungswert mit dem Faktor drei multipliert werden.

Beispiel Aufruf vom Programm:

|  |
| --- |
| main.c |
| errorCode = DS2438\_ReadVoltage();  if(errorCode != DS2438\_OK)  {    sprintf(txt, "DS2438 Error %d\n\r", errorCode);      Terminal\_Print(txt);  } |

## Abstandssensor - HCSR04 (Ben)

### Funktionsweise

### Code

## Motoransteuerung

### Motorstrom auslesen

ADC Erklärung

ADC Einlesen Erklärung

### DShot Protokoll

Aufbau Protokoll

Geschwindigkeiten

Init() Aufruf + Programm selber

## PID-Regler

### Hover Modus

Schalter auf Fernsteuerung Erkennung im Programm

PID-Algorithmus Programm

### Failsafe

Wann?

Wie?

Programm

## Debugging

### Terminal Ausgabe

Wofür?

Wie?

USB UART Platine

Einstellungen CubeMX

Einstellungen Terminal (VSCode, X-CTU)

### Status – LEDs

Was bedeuten Pattern

Blaue rote LED

# Datenübertragung der Mess- und Videodaten (Ben)

## Überblick Datenübertragung

Blockschaltbild und schaltung

## Kommunikation: Flight Controller und Sender

### *Aufbau Sendermodul (VTx)*

### ASK – Modulation

Überblick und wieso nicht verwendet

### Transmit – Programm auf Cortex µC

#### Gleitkommadarstellung (memcpy)

#### Code

### Testen der Übertragung

+ probleme

## Kommunikation: Sender und Empfänger

### Aufbau Empfängermodul

### Verbindungsaufbau

### Testen der Übertragung

## Kommunikation: Empfänger und Raspberry Pi

### Empfangsprogramm auf Raspberry Pi

#### UART auf dem Raspberry Pi

#### Code

### Senden der Daten auf dem MQTT-Server

## Testen der Datenübertragungskette

# Visualisierungs-App

## Applikation

### Einführung – Dart / Flutter - Framework

#### Allgemeines

#### Pub Dev / Package Installer

#### Dart Syntax

#### State Management

##### Stateful vs. Stateless Widgets

##### Provider

### Allgemeines zur App

#### Usersystem

#### Datenvisualiserung

#### 3D-Model-Viewer

#### Livestream-Viewer

### UI-Konzept

### Projektstruktur und -umgebung

#### Editor – Visual Studio Code

#### Flutter Installation

#### Projekterstellung

#### Projektstruktur

#### Packages

##### Pubspec.yaml

##### Installieren neuer Packages

### Splash + Willkommensscreen

#### Native Splash Screen

#### Willkommensbildschirm

#### Logo-Design

### Login + Registrierung

#### E-Mail-Login

#### Google-Login

#### Passwort vergessen

#### Registrierung

### Homepage

#### Bottom Navigation Bar / GNav-Bar

#### Serverdatendialog

#### Flugdatenvisualisierung

##### MQTT-Datenstreams

##### Syncfusion Cartesian Charts

##### Echtzeitdaten + Animierte Diagramme

##### Speicherung der Daten

#### 3D-Model-Viewer

##### Flutter Cube

##### MQTT-Datenstream

##### 3D-Dronenmodell

###### Optimierungen in Blender

##### Extrabedienungen

#### Live-View

##### VLC-Plugin

##### Darstellung eines Netzwerkstreams

##### Video-Overlay

### Sidemenu / Drawer

#### Userprofil

##### Darstellung der Userdaten

##### Profilbildauswahl

##### Änderung der Userdaten

#### Credits

#### Vorherige Flüge

##### Darstellung aller gespeicherten Flüge

##### Sortiermöglichkeiten

##### Einsicht in vorherigen Flug

#### Einstellungen

##### Dark- / Light-Mode

##### 3D-Model Ausrichtung

#### Logout´

## Firebase Backend

### Installation via Firebase CLI

##### Firebase CLI – Setup

##### Firebase für Dart / Flutter aktivieren

##### Flutter App mit Flutterfire konfigurieren

### Einbindung in Flutter

##### Flutterfire Configure

##### Benutzten Firebase Packages hinzufügen

### Nutzen der verschiedenen Datenbanksysteme

#### Auth

##### E-Mail + Passwort – Login

##### Google Login

#### Firestore Database

##### Struktur

##### Allgemeine Userdaten

##### Flugdaten

##### Einstellungen

#### Realtime Database

##### Allgemein

##### Sinn der verschiedenen Flags

#### Storage

##### Allgemein

##### Referenz zu User in Firestore

# Videostreaming

## Allgemeiner Aufbau

### Anforderungen

### Übertragungskette

## CADFPX Analog Kamera

### Allgemein

### Produktinformationen

### Verwendungszweck

### Verbindungstest via USB-Camera App

## USB2.0 VHS Video Grabber

## RTMP-Server via NGINX aufsetzen

### RTMP Allgemein

### NGINX

#### Allgemein

#### Installation

#### RTMP-Konfiguration

#### Serverstatus einsehen

## RTMP-Stream erstellen

### FFMPEG Allgemein

### Aufbau der Konvertierung

#### Simple Konvertierung

#### Optimierungen

##### Encodervergleich

##### Optimierungsflags

##### Unoptimiert vs. Optimiert

# Videoserver-Port Forwarding

## Allgemein

## Routereinstellungen

## Kontakt mit Routerfirma zur Freischaltung

## Testen der Ports

### Portchecker.co

## Testen des Videostreams

### VLC-Player

#### Devicestream direkt testen

#### Netzwerkstream im selben Netzwerk

#### Netzwerkstream via Public IP

# Ergebnisse

Entweder Gesamt System irgendwelche Bilder

Oder wenn nicht, dann einzelne Teile die funktionieren zeigen / beschreiben

Das Kapitel ist, um die Ergebnisse zu verkaufen, basically alles was geht

Name könnte sich noch ändern

# Anhang

## Einführung CAD – Software (Fusion 360) (Ben)

### UI und Projekterstellung

### Skizze anfertigen

### Körper erstellen

### Schrift und Bilder einfügen

## 3D – Druck (Ben)

### 3D – Drucker

### Filamente

### 3D – Drucker Software (Ultimaker Cura)

## Inbetriebnahme Anleitung

## Projektplan

## Projektkosten

## Projekttagebuch

# Quellen

## Gedruckte Medien

Für die Quellenangabe bei Fachbüchern ist, wie nachfolgend dargestellt, vorzugehen (löschen):

Vorname Nachname: Titel. Untertitel. - Verlagsort: Verlag, Jahr.

Vorname Nachname: Titel. Untertitel. Auflage - Verlagsort: Verlag, Jahr.

z.B.:

[REI13] Jürgen Reichhardt: Lehrbuch Digitaltechnik - Eine Einführung mit VHDL  
 3. Auflage, München: Oldenbourg Verlag, 2013  
 ISBN: 978-3-486-72765-4

## Online

Für die Quellenangabe von Suchergebenisse in Internet ist, wie nachfolgend dargestellt, vorzugehen (löschen):

Vorname Nachname des Autors: Titel. Online in Internet: URL: www – Adresse, Datum.

(Autor und Titel wenn vorhanden, Online in Internet: www-Adresse, Datum auf jeden Fall )

z.B.:

[APA17] Apache Foundation: The Maven Project

<http://maven.apache.org/>

(Letzter Aufruf: 17.12.2017)

[TOR91] Linus Torvalds: Linux announcement

<https://web.archive.org/web/20100104211620/http://www.linux.org/people/linus_post.html>

(Erstellt: 25.08.1991, Letzter Aufruf: 17.12.2017)

# Verzeichnis der Abbildungen

# Begleitprotokoll

# Anhang

## — Projektdokumentation (Kostendarstellung, Besprechungsprotokolle etc.)

## — Technische Dokumentation (technische Beschreibungen, Berechnungen,

## Konstruktionszeichnungen, Versuchsberichte, betriebswirtschaftliche Kalkulationen etc.)

## Schaltungen, Zeichnungssätze, sonstiges