OSTBAYERISCHE TECHNISCHE HOCHSCHULE REGENSBURG

Masterarbeit

Hard- und Softwareentwicklungen für ein Flugrobotersystem

Ausgeführt am
Sensorik-Applikationszentrum

unter der Anleitung von
Prof. Dr. rer. nat. Rudolf Bierl, Dipl.-Phys.

durch
Dipl.-Ing. (FH) Andreas Gschoßmann
Am Protzenweiher 7
93059 Regensburg

Regensburg, den 01.09.2015

Diese Arbeit wird online zur Verfügung gestellt. Der Zugriff ist passwortgeschützt.

http://hps.othr.de/gsa39665
andreas.gschossmann@oth-regensburg.de

Abstract

This thesis is part of the research on Unmanned Aerial Systems at the Sensorik-Applikationszentrum (SappZ) at the University of Applied Sciences Regensburg. One of the main purposes of this research is to meet the great potential for Unmanned Aerial Systems (UAS) to augmented desaster relieve. These drones will be used by local fire and rescue departments. Optimal design leads to Unmanned Aerial Vehicles (UAV) that hover for long periods of time and deliver HD video back to the ground where immediate decisions can be made by command personnel. Further more, since being used mainly in tense mission scenarios, the cognitive activity of controlling the drone has to be facile and effortless. A quadrotor can serve these requirements. Since command controls are explored to facilitate the flight for the pilot it is important to have a flightcontrol which allows full access to its flight algorithms. To that purpose, a flightcontrol for quadrotors is developed.

Being a research center for sensor technology another application for UAVs at SappZ is using UAVs as platforms to carry a wide range of different sensors.

This thesis covers four basic tasks on the way towards a flexible hard- and software control for a quadrotor. The flightcontrol electronics hardware was developed. It is capable of both, running flight algorithms in real time and serving sensor interfaces and drivers on a high level Linux system. A GUI-based software simulation for UAVs was developed in order to test flight algorithms for in- and outdoor scenarios. Algorithms for attitude and position estimation, based on Kalman filtering, were developed. This has been solved with great emphasis on numerical stability and runtime. Finally, a suitable data structure, shared by ground tools and the UAV, based on available Open Source protocols for communication was found.

Danksagung

Zunächst möchte ich allen Kollegen des Sensorik-Applikationszentrums danken, die mir stets mit Rat und Tat zur Seite standen. Eine orientalische Weisheit besagt, wer alleine arbeitet addiert, wer zusammenarbeitet multipliziert. Ich bin dankbar jene Erfahrung in einer so gedeihlichen Gesellschaft ehrbarer Menschen machen zu dürfen. Sie ist Keim so mancher Idee, die zu schöpferischem Tun, nicht nur fachlicher Natur anregt.

Ein herzliches Dankeschön gilt Thorsten Reitmeier, der mit leidenschaftlichem Einsatz und ohne Selbstzweck seine langjährige Erfahrung und sein feines Gespür im Bereich Regelungstechnik eingebracht hat und hierfür bereit war seine Freizeit zu opfern. Auch Wolfgang Högele möchte ich für die fachlichen Ratschläge und fruchtbaren Diskussionen zum Thema Kalman-Filter danken. Er stand jeder Fachdiskussion zur praktischen Anwendung des Kalman-Filters mit Freude offen und es machte Spass, mit ihm verschiedene Aspekte der Thematik zu entschleiern.

Außerdem bedanke ich mich bei Martin Hofmann und Thomas Rück für die moralische Unterstützung und die aufmunternden Worte in unkreativen Momenten. Auch Ignaz Laepple möchte ich für seine kritische und ehrliche Meinung, die mich häufig zur Reflexion, sowie zur Neuordnung meiner Gedanken angeregt hat, danken.

Auch bedanke ich mich bei Rafaela Gonetz, Matthias Strobl, Elisabeth Keck, Ignaz Laepple, Tina Bradschi, Simon Jobst und Florian Olbrich für das Korrekturlesen. Ich werde mich mit einem Bier revanchieren.

Bei Waldemar Sessler und Johannes Fischer bedanke ich mich für die kameradschaftliche Zusammenarbeit. Im Zweifel konntet Ihr mir immer einen Ratschlag geben und es macht Spass, mit Euch zu arbeiten.

Mein besonderer Dank gilt Prof. Dr. Rudolf Bierl, der uns vertraut, uns ermöglicht an dem Projekt zu arbeiten und uns die Freiheit gibt, unsere Visionen umzusetzen.

Zuletzt möchte ich meiner Familie für die Unterstützung in allen Lebenslagen danken.

Meinem Vater Nikolaus Gschossmann und seinem Freund und Kollegen Hans Horneck, die mir stets ein Vorbild waren und mich nicht nur fachlich inspiriert haben.

Erklärung

1. Mir ist bekannt, dass dieses Exemplar der Masterarbeit als Prüfungsleistung in das Eigentum des Freistaates Bayern übergeht.

2. Ich erkläre hiermit, dass ich diese Masterarbeit selbstständig verfasst, noch nicht anderweitig für andere Prüfungszwecke vorlege, keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie wörtliche und sinngemäße Zitate als solche gekennzeichnet habe.

Regensburg, den _____ _____
Ort, Datum ***Unterschrift***

Abkürzungsverzeichnis

SappZ	Sensorik-Applikationszentrum
UAV	Unmanned Aerial Vehicle
UAS	Unmanned Aerial System
GUI	Graphical User Interface
QR	Quick Response
CBRNE	Chemical, Biological, Radiological, Nuclear and High-Yield Explosive
LuftVG	Luftfahrgesetz
DLR	Deutsches Luft- und Raumfahrtzentrum
PID	Proportional Integral Derivative
GPS	Global Positioning System
FAA	Federal Aviation Administration
NAS	National Airspace System
RPAS	Remotely Piloted Aircraft Systems
SESAR	Single European Sky ATM Research
ATM	Airtraffic Control
I2C	Inter IC Bus
SPI	Serial Peripheral Interface
UART	Universal Asynchronous Receiver Transceiver
USB	Universal Serial Bus
RISC	Reduced Instruction Set Computer
ARM	Acorn RISC
DSP	Digitaler Signal Prozessor
IMU	Inertial Measurement Unit
IC	Integrated Circuit
EOMA	Embedded Open Modular Architecture
PPM	Puls-Position Modulation
SSH	Secure Shell
OTG	On-The-Go
GPIO	General Purpose I/O
GND	Ground

PHY	Physical Layer
LSB	Least Significant Bit
DDR	Double Data Rate
SDRAM	Synchronous Dynamic Random Access Memory
SOM	System on Module
TTL	Transistor-Transistor-Logik
MDIO	MANAGEMENT DATA I/O
MII	Media Independent Interface
SD	Secure Digital
PMIC	Power Management IC
PLL	Phase Locked Loop
RTC	Real Time Clock
ESD	Electro Static Discharge
LED	Light Emitting Diode
EMU	Emulation
JTAG	Joint Test Action Group
TI	Texas Instruments
MMC	Multimedia Card
WP	Write Protect
CD	Card Detect
PCB	Printed Circuit Board
pwt	Power
CU	Cuprum, Kupfer
LQ	Linear Quadratic
KF	Kalman-Filter
EKF	Extended Kalman-Filter
API	Application Programming Interface
XML	Extensible Markup Language
SDF	Simulator Description Format
TCP	Transmission Control Protocol
IP	Internet Protocol
MT	Momentum Theorie

3D	Dreidimensional
SIL	Software in the Loop
IEEE	Institute of Electrical and Electronics Engineers
GNSS	Globales Navigationssatellitensystem
INS	Inertiale Sensorik
UKF	Unscented Kalman-Filter
DKF	Distributed Kalman-Filter
MARG	Magnetic, Angular Rate, Gravity
AHRS	Attitude Heading Reference System
RTK	Realtime-Kinematics
DGPS	differential GPS
QZSS	Quasi Zenit Satelliten System
GLONASS	Globalnaja Nawigazionnaja Sputnikowaja Sistema
MEMS	Microelectromechanical System
GPL	Gnu General Public Lizenz
SAS	Stabilized Augmented Systems
JSON	JavaScript Object Notation
BSD	Berkeley Software Distribution
LGPL	Lesser General Public License
I/O	Input/Output
MIT	Massachusetts Institute of Technology
UTF-8	8-Bit Universal Character Set Transformation Format
ASF	Apache Software Foundation
CAN	Controller Area Network
GSM	Global System for Mobile Communications
USAR	Urban Search and Rescue

Abbildungsverzeichnis

1.1 Link zur Website der Arbeit . . . 2
1.2 Quadrocopter Merlin des Sensorik-Applikationszentrums . . . 2
1.3 Anwendung von Flugrobotern . . . 3
1.4 Link zur Präsentation Luftverkehrsrechtliche Betrachtungen zu UAVs . . . 4
1.5 Anforderungen der FlightConrol . . . 7

2.1 Grundsätzliche Struktur der Flugsteuerung . . . 10
2.2 Blockdiagramm der Flugsteuerung . . . 11
2.3 Tag-Connect Stecker, Links: TC2050-IDC-NL, Rechts: TC-2000-M Adapter . . . 13
2.4 Pinout des Breakout-Connectors . . . 13
2.5 Pinout des EOMA-68-Connectors . . . 14
2.6 Orientierung der inertialen Messeinheit auf der Flugsteuerung . . . 18
2.7 Außenabmessungen und Position der Bohrungen der Flugsteuerung . . . 19
2.8 Schaltung der Spannungsversorgung der Flugsteuerung . . . 24
2.9 Resetschaltung . . . 25
2.10 XDS100v2 von Spectrum Digital [22] . . . 26
2.11 Molex SD-Kartenhalter (Art.Nr.49225-0821) . . . 28
2.12 Aufbau der Lagen der Flugsteuerung . . . 29
2.13 Schichtdicken einer vierlagigen Leiterplatte von *Hofmann Leiterplatten* . . . 30
2.14 Footprint des OMAP-L138 SOM-M1 . . . 31
2.15 Footprint des Tag-Connect Debug-Steckers [12] . . . 32
2.16 Layoutvorlagen von Tag-Connect [12] . . . 32

3.1 Simulations Framework [74] . . . 38
3.2 GUI von DeadalusSim . . . 39
3.3 Koordinatensysteme und Drehrichtung der Rotoren . . . 40
3.4 Messungen der Leistung und des Schubs über der Drehzahl für Robbe Roxxy BL2827-34 . . . 42
3.5 Systemkonstanten der Motordynamik im stationären Fall . . . 42
3.6 Simulations Framework [74] . . . 43
3.7 Ablauf der Berechnung des Dynamischen Verhaltens des Quadrocopters . . . 44
3.8 PlayStation 3 Controller . . . 45

4.1 Rolle der Sensordatenfusion im Gesamtsystem [95] . . . 48

4.2 Magnetfeld der Erde [66] . . . 52
4.3 Grundprinzip Komplementärfilter . . . 54
4.4 Komplementärfilter nach [101] . . . 55
4.5 Nominale Appollo Trajektorien, skizziert ohne die Berücksichtigung der orbitalen Geschwindigkeit des Mondes um die Erde (1 km/h). Jene orbitale Geschwindigkeit des Mondes macht die Berechnung der Trajektorien noch komplizierter. [56] . . . 56
4.6 R.E. Kalman bei der Überreichung des Charles Stark Draper Preises [110] . . . 56
4.7 Ablauf Kalman-Filter [127] . . . 57
4.8 Ablauf des Kalman-Filters: (a) Schätzung (a priori) anhand des Systemmodells, (b) Erhebung der Messung, (c) Korrektur der Schätzung (a priori) durch die Messung (a posteriori) [122] [46] . . . 57
4.9 Kalman-Filtergleichungen [46] . . . 58
4.10 Extended Kalman-Filtergleichungen . . . 59
4.11 Apollo Guidance Computer (15-bit Fixpoint-Arithmetik) und Display/Keyboard-Einheit [130] 61
4.12 Verwendete Koordinatensysteme [129] . . . 62
4.13 ZYX-Drehung **(a)** Yaw ϕ um z-Achse **(b)** Pitch ψ um y-Achse **(C)** Roll θ um x-Achse . . . 64
4.14 Roll-, Pitch- und Yaw-Winkel am Luftfahrzeug . . . 64
4.15 Vereinfachter Strapdown-Algorithmus [93] . . . 66
4.16 Umsetzung des Komplementärfilters nach [101] **a)** ohne Driftkompensation und **b)** mit Driftkompensation des Drehratensensors . . . 68
4.17 Trigonometrie des Beschleunigungssensors exemplarisch für den Roll-Winkel. . . . 69
4.18 Flussdiagramm des Kalman-Filteralgorithmus zur Fluglageschätzung in sieben Schritten . . 70
4.19 Konzept Fluglageregler . . . 71
4.20 Alternatives Reglerkonzept . . . 72
4.21 Extended Kalman-Filteralgorithmus zur Positionsschätzung nach [105] . . . 76
4.22 Konzept Positionsregelung nach [93] . . . 77

5.1 Funkkonzept des UAS . . . 81

B.1 Link zur Klassendokumentation der DeadalusSim-Software [59] . . . 110

Tabellenverzeichnis

2.1 Pinout Receiver-Eingang . . . 15
2.2 Pinout I2C-Motortreiber . . . 15
2.3 Pinout UART-Debug-Port . . . 15
2.4 Pinout der Analogen Eingänge . . . 16
2.5 Barometrische Drucksensoren im Vergleich . . . 16
2.6 Sensitivität Drehraten MPU-9150 und BMX055 . . . 17
2.7 Sensitivität Beschleunigung MPU-9150 und BMX055 . . . 17
2.8 Sensitivität MPU-9150 und BMX055 . . . 17
2.9 Berechnung der Impedanz von Differenziellen Leitungen . . . 30

4.1 Nomenklatur des Systemmodells des Kalman-Filters zur Positionsschätzung . . . 73

5.1 Vergleich von Bibliotheken zur Serialisierung strukturierter Daten . . . 82
5.2 Sprachanbindung ausgewählter Bibliotheken zur Serialiserung . . . 83

Abstract . . . iii
Danksagung . . . v
Erklärung . . . ix
Abkürzungsverzeichnis . . . x
Abbildungsverzeichnis . . . xiv
Tabellenverzeichnis . . . xv

1 Einleitung **1**
1.1 Aufbau der Arbeit . . . 1
1.2 Was ist ein Quadrocopter . . . 2
1.3 Anwendungen . . . 2
1.4 Geschichte . . . 4
1.5 Rechtliches . . . 4
1.6 Stand der Technik . . . 4
1.7 Forschung und Entwicklung von UAS-Technik im Sensorik-Applikationszentrum . . . 5
1.8 Ziel der Arbeit . . . 6

2 Hardwareentwicklung der Flugsteuerung **9**
2.1 Einleitung . . . 9
2.1.1 Ziel der Arbeit . . . 9
2.1.2 Stand der Technik . . . 10
2.1.3 Rahmenbedingungen . . . 10
2.2 Systemübersicht . . . 11
2.2.1 Schematische Struktur . . . 11
2.2.2 Mechanischer Aufbau . . . 12
2.3 Aufbau . . . 13
2.3.1 Pinbelegung . . . 13
2.3.2 Inertiale Messeinheit . . . 16
2.3.3 Abmessungen . . . 19
2.4 Technische Daten . . . 19
2.4.1 Features . . . 19
2.4.2 Specs . . . 20
2.5 Schaltungsdesign . . . 20
2.5.1 Signalübersicht und Überprüfung des Pinmultiplexing . . . 20

2.5.2 Versorgung . . . 22
2.5.3 Clocks . . . 23
2.5.4 Reset . . . 24
2.5.5 Testpunkte . . . 24
2.5.6 User LEDs . . . 25
2.5.7 Debug Interface . . . 26
2.5.8 Sensoren IMU . . . 27
2.5.9 USB . . . 27
2.5.10 Micro SD-Card . . . 28
2.6 Layout . . . 29
2.6.1 Aufbau der Lagen . . . 29
2.6.2 Toleranzen . . . 29
2.6.3 USB . . . 30
2.6.4 Footprint OMAP-L138 SOM-M1 . . . 31
2.6.5 Footprint JTAG Debugger . . . 31
2.7 Ausblick . . . 31

3 Simulation **35**
3.1 Einleitung . . . 35
3.1.1 Ziel der Arbeit . . . 35
3.1.2 Stand der Technik . . . 36
3.1.3 Rahmenbedingungen . . . 37
3.1.4 Gazebo . . . 38
3.2 Quadrocopter Modell . . . 39
3.2.1 Kinematik . . . 39
3.2.2 Dynamik . . . 40
3.2.3 Identifikation der Systemkonstanten aus Messungen . . . 41
3.3 Simulation . . . 43
3.3.1 Aufbau . . . 43
3.3.2 Gazebo-Plugin Klassen . . . 43
3.3.3 Quadrocopter Dynamik Klassen . . . 44
3.3.4 Software in the Loop (SIL) . . . 44
3.4 Regelung . . . 45
3.4.1 Backstepping-Regler . . . 45
3.4.2 Steuerung . . . 45
3.4.3 Mixer . . . 46
3.5 Ausblick . . . 46

4 Sensordatenfusion **47**
4.1 Einleitung . . . 47
4.1.1 Ziel der Arbeit . . . 47
4.1.2 Stand der Technik . . . 48
4.1.3 Rahmenbedingungen . . . 51
4.2 Theorie . . . 54
4.2.1 Komplementärfilter . . . 54
4.2.2 Kalman-Filter . . . 56
4.2.3 Koordinatentransformation . . . 61
4.3 Sensorfusionsalgorithmen . . . 67

4.3.1 Fluglageschätzung . . . 67
4.3.2 Positionsschätzung . . . 72
4.4 Ausblick . . . 76

5 Protokolle **79**
5.1 Einleitung . . . 79
5.1.1 Ziel der Arbeit . . . 79
5.1.2 Stand der Technik . . . 80
5.1.3 Rahmenbedingungen . . . 81
5.2 Vergleich der Bibliotheken . . . 82
5.2.1 Diskussion . . . 82
5.2.2 Fazit . . . 83
5.3 Implementierung . . . 84
5.4 Ausblick . . . 84

6 Schluss **85**
6.1 Zusammenfassung . . . 85
6.2 Ausblick . . . 86

Anhang **86**

A Fertigungsunterlagen Flightcontrol **87**
A.1 Stückliste . . . 87
A.2 Schaltplan . . . 91
A.3 Layout . . . 101

B DeadalusSim **109**
B.1 Installation . . . 109
B.1.1 Gazebosim installieren . . . 109
B.1.2 Playstation 3 Controller Treiber installieren . . . 109
B.1.3 DeadalusSim kompilieren und starten . . . 110
B.2 Class Documentation . . . 110

C Codes Sensordatenfusion **111**
C.1 Komplementär-Filter zur Lageschätzung . . . 111
C.1.1 Code Ansatz Mahony . . . 111
C.1.2 Code Ansatz Oliviera/Pascoal/Kaminer . . . 113
C.2 Kalman-Filter zur Lageschätzung . . . 114
C.2.1 Code Kalman-Filter zur Lagestützung . . . 114
C.2.2 Matrix-Berechnungen . . . 119

D Codes Protokolle **121**
D.1 MAVlink Samples . . . 121
D.1.1 MAVlink installieren . . . 121
D.1.2 MAVlink deinstallieren . . . 121
D.1.3 Ausführen des Sample Codes . . . 122
D.1.4 Troubleshooting . . . 122

Literaturverzeichnis **123**

KAPITEL 1

Einleitung

Eine Flugmaschine zu erfinden bedeutet wenig; sie zu bauen schon mehr; aber sie zu fliegen, das ist das Entscheidende.

Ferdinand Ferber [1]

[1] Jene Worte werden häufig fälschlicherweise dem Luftfahrtpionier Otto von Lilienthal zugeschrieben. Diese Fehlannahme stellt Ferdinand Ferber in seinem Werk «l'Aviation, ses débuts, son développement» 1908 richtig.

Inhalt: Zunächst wird der Aufbau der Arbeit beschrieben. Danach wird darauf eingegangen was ein Quadrocopter ist, welche Anwendungen ein solches Gerät findet und in groben Zügen auf die Geschichte von Quadrocoptern eingegangen. Außerdem wird Grundlegendes zur rechtlichen Lage bezüglich des Betriebs von Unbemannten Luftfahrzeugen (*Unmanned Aerial Vehicle*, UAVs) eingegangen und auf weitere Literatur verwiesen. Es wird ein Überblick über den Stand der Technik der Forschung und Entwicklung im Bereich Unbemannte Luftfahrtsysteme (*Unmanned Aerial System*, UAS) gegeben und die Forschungen am Sensorik-Applikationszentrum beschrieben. Abschließend wird das Ziel dieser Arbeit im Gesamten beschrieben.

1.1 Aufbau der Arbeit

Die vorliegende Arbeit setzt sich aus vier Hauptthemen zusammen: Hardwareentwicklung der Flightcontrol, Simulation, Sensordatenfusion sowie Protokolle. Jedes dieser Themen ist Teil der Entwicklung der Flugsteuerung eines Quadrocopters und wurde für sich eigens behandelt und dokumentiert. Diese Aufteilung spiegelt sich im Aufbau der Arbeit wider. Jedem der vier Hauptthemen wird ein Kaptitel gewidmet und zu jedem Kapitel korrespondiert jeweils ein eigener Anhang. Dieser Aufbau beruht darauf, dass jedes dieser vier Themen in seiner Ordnerstruktur als eigenständiges Projekt angelegt wurde. Die jeweiligen Arbeiten können im Download-Bereich [58] der Website dieser Arbeit, neben der Arbeit im Gesamten, auch einzeln

heruntergeladen und betrachtet werden. Der Link zur Website selbst, in Form eines QR-Codes, findet sich in Abbildung 1.1.

Abbildung 1.1: Link zur Website der Arbeit

1.2 Was ist ein Quadrocopter

Quadrocopter sind vierrotorige Fluggeräte, die mittlerweile vor allem in Form von Drohnen eingesetzt werden. Jeder der vier Rotoren wird von einer Elektronik in einer Weise angesteuert, dass der Flugroboter die Balance in der Luft hält. Dadurch fliegt das Gerät sehr ruhig und eignet sich ideal um Filmaufnahmen und Bilder mit einer Boardkamera zu machen. Die Regelungselektronik kann das Fluggerät auch in einen bestimmten Winkel im Raum anstellen. Dadurch fliegt das Gerät in die gewünschte Richtung. Es gibt auch Drohnen mit mehr als vier Rotoren, dann spricht man von Multicoptern.

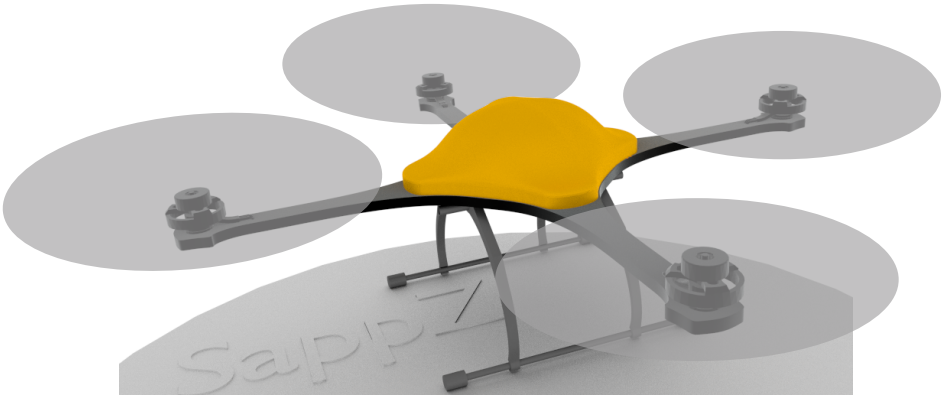


Abbildung 1.2: Quadrocopter Merlin des Sensorik-Applikationszentrums

1.3 Anwendungen

Die Anwendungen von Flugrobotern sind zahlreich und gerade in der letzten Zeit sprießen fast täglich neue innovative Einsatzideen aus dem Boden. Beispielsweise kann man Außeninspektionen von großen technischen Anlagen wie Stromleitungen, Schornsteinen oder Windkraftanlagen durchführen. So kann auf Hubschrauber und Industriekletterer verzichtet werden. Das erhöht die Sicherheit und macht diese Einsätze kostengünstiger. Auch die Inspektion von Solarfeldern in Verbindung mit Wärmebildanlagen ist eine häufige Anwendung von Multirotor-Flugsystemen. Defekte Module können durch deren Erwärmung aufgespürt werden. Wenn man bedenkt, dass jedes defekte Modul wie ein Widerstand wirkt und somit die Effektivität

des Gesamtsystems verschlechtert, sieht man, dass eine rasche Auffindung des defekten Modules bares Geld spart.

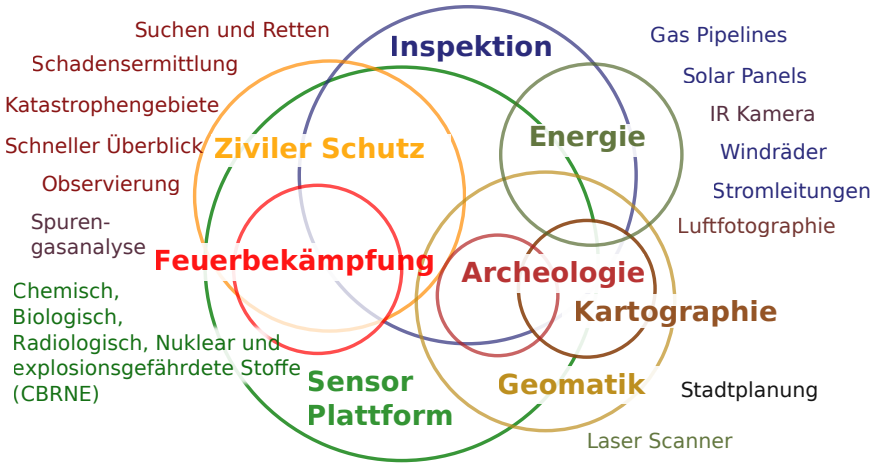


Abbildung 1.3: Anwendung von Flugrobotern

In [112] wird eine Möglichkeit vorgestellt, Walfischen mit einer Drohne autonom zu folgen. Dies wird mithilfe von Bilderkennungsalgorithmen umgesetzt. Ein Video unter der Rubrik *weiterführende Links* im Downloadsbereich der Website dieser Arbeit [58] verdeutlicht, wie schwierig es ist, Walfische und Delphine von einem Boot aus mit einer Drohne manuell zu filmen. Dies zeigt das Potential dieser Technologie für Meeresbiologen.

Eine sehr vielversprechende praktische Anwendung ist der Einsatz von Flugrobotern in der Landwirtschaft. Wird in der Landwirtschaft mit Sensorik gearbeitet um bestimmte Parameter zu ermitteln, spricht man von Präzisionslandwirtschaft (*Precision Agriculture, PA*). Bestimmte Werte, wie beispielsweise der Nitratgehalt von Pflanzen, können durch die Auswertung von Satellitenbildern erhoben werden. Da diese Variante jedoch sehr teuer ist, werden Möglichkeiten erforscht, diese Bilddaten mithilfe von Flugrobotern zu sammeln [69], [106].

Im Sensorik-Applikationszentrum wird besonderes Augenmerk auf den Einsatz von Flugrobotern für Umwelt- und Katastrophenschutz gelegt. Hier finden sich zahlreiche Einsatzmöglichkeiten. Es wird am Anfang des Einsatzes ein Gerät mit Kamera in die Luft gebracht und Bilder werden an eine Bodenstation gesendet. So hat die Einsatzleitung zügig einen Überblick über das Geschehen und kann schnell Entscheidungen treffen. Auch Schadensanalysen nach Bränden sind möglich. Denkbar sind auch automatisierte Flüge über Katastrophengebieten oder Vermissten- Suchmissionen mit Wärmebildkameras. Mithilfe von geeigneter Sensorik, wie CBRNE-Detektoren [1] können Daten in Bereichen gesammelt werden, die für Rettungskräfte nur schwer zugänglich sind.

[1] CBRNE: **engl.** *chemical, biological, radiological, nuclear and high-yield explosives*, **deutsch** *chemische, biologische, radiologische, nukleare und explosive Gefahrstoffe*

1.4 Geschichte

Das Konzept eines Luftfahrzeuges mit vier Rotoren wurde erstmals 1907 von Brequet und Richet untersucht. Deren schweres und großes Luftfahrzeug hob nur wenige Meter über dem Boden ab und flog nur kurze Zeit [83]. Erst 1920 wurde wieder ein Fluggerät mit vier Rotoren entwickelt (Étienne Oehmichen, seit 1920). Der erste Prototyp war zunächst zu schwer und benötigte einen zusätzlichen Wasserstoffballon um fliegen zu können [82]. Die Fluggeräte mit vier Rotoren wurden schließlich durch die Erfindung der Taumelscheibe gänzlich verdrängt. In den sechziger Jahren gab es noch einmal einige Entwicklungen von Fluggeräten mit vier Rotoren, die meisten kamen jedoch über das Konzeptstadium nicht hinaus [29].

Durch die Verfügbarkeit von immer genaueren und billigeren Gyroskop- und Beschleunigungssensoren haben in den letzten Jahren Quadrocopter Einzug in den Bereich der sogenannten (*Unmanned Aerial Vehicles, UAV*), in Form von kleinen unbemannten Fluggeräten, gehalten.

1.5 Rechtliches

Aufgrund der steigenden Popularität von Flugdrohnen wurde für sie das Luftverkehrsgesetz (LuftVG) abgeändert. Kommerziell eingesetzte Flugroboter bekommen jetzt eine eigene Kategorie: *unbemannte Luftfahrsysteme*. Es bedarf einer Aufstiegsgenemigung für ihren Betrieb. Hobbygeräte fallen weiterhin unter die Kategorie *Flugmodelle* und sind nicht genehmigungspflichtig, wenn sie unter 5kg wiegen. Sowohl der autonome Flug als auch der Flug außerhalb des Sichtbereiches des Piloten ist derzeit weiterhin gesetzeswidrig.

Abbildung 1.4: Link zur Präsentation Luftverkehrsrechtliche Betrachtungen zu UAVs

Für genauere Informationen zur Gesetzeslage wurde online eine Präsentation hinterlegt [62]. Dort wurden Gesetze mit Relevanz für unbemannte Luftfahrzeuge unter 5 kg in Deutschland zusammengetragen und mit Quellen belegt.

1.6 Stand der Technik

Die Fluglage- und Positionsregelung von Quadrocoptern wurde bereits vielfach untersucht. Die ersten Arbeiten mit hohem Impact-Faktor sind wohl jene der ETH-Zürich [34] und des Deutschen Luft und Raumfahrtzentrums (*DLR*) [63], beide aus dem Jahr 2007. Hier werden gewöhnliche PID-Regler verwendet. Im Laufe der Zeit wurden auch weitere Regelungskonzepte vorgestellt. So kommen Fuzzy-Regler [93], Backstepping- [87] [129] oder Sliding-mode-Regler [81] [36] zum Einsatz. Auch neuronale Netze zur Höhenregelung [107] oder - in Verbindung mit einem Backstepping Regler - zur Fluglageregelung [86] wurden vorgeschlagen. In [114] wird ein neuronales Netz zur Regelung des Quadcopters vorgelegt.

Um notwendige Werte für die Positions- und Fluglageregelung zu erhalten werden barometrischer Höhendruck-, Magnetfeld-, Beschleunigungs-, Drehraten- und GPS-Sensoren in geeigneter Weise fusioniert. Diese Themen wurden bereits zahlreich behandelt. Die Fluglageschätzung wird in [93] [129] [91] [90] [89] [88] [115] [28] [25], die Positionsschätzung mithilfe von Globaler Navigation Satelliten Systemen wird in [93] [129] [96] [134] [47] behandelt. Als Werkzeug der Sensorfusion wird häufig der Kalman-Filter verwendet. Dieser wird in [78] [102] [117] [85] [46] [127] beschrieben. Für eine detailliertere Einführung in den Stand der Technik zum Thema Sensorfusion sei auf das Kapitel *Sensordatenfusion* verwiesen.

Viele Open Source Flugsteuerungen wurden in den letzten Jahren umgesetzt. Einige von ihnen werden in [84] vorgestellt und verglichen.

Auch bezüglich der Energieversorgung des unbemannten Luftfahrzeuges (UAV) gibt es Forschungen. In [100] wurde beispielsweise eine Akkuüberwachung basierend auf einem Kalman-Filter vorgestellt. In [27] wird ein Quadrocopter mithilfe einer speziellen hocheffektiven Solarzelle [79] der Firma LaserMotive [21] über einen Infrarotlaser mit Energie versorgt.

Für die Untersuchung der Aerodynamik der Rotoren sei zunächst auf ein Standardwerk über Hubschraubertechnik der Cambridge Universität, von J. Gordon Leishman, verwiesen [82]. Hier wird in die Grundlagen der aerodynamischen Untersuchung von Rotoren eingegangen. Leishman hat auch interessante Themen, wie beispielsweise das Schubverhalten überlappender Rotoren [57] untersucht. Die Behandlung von verschiedensten Aerodynamik-Themen, die Quadrocopter im Speziellen betreffen finden sich in [43] [126] [120] [75] [103]. Das Thema Simulation von Quadrocoptersystemen wird in [74] [48] [53] [54] [55] behandelt. Der Stand der Technik bezüglich der Themen Aerodynamik und Simulation wird neben dem physikalischen Bewegungsmodell eines Quadrocopters, in Kapitel *Simulation* noch einmal feingliedriger zusammengetragen.

Die Bestrebungen, unbemannte Luftfahrzeuge in die jeweiligen nationalen Lufträume zu integrieren, ergeben völlig neue Forschungsfelder. Im November 2013 veröffentlichte die Bundesluftfahrtbehörde der USA (*Federal Aviation Administration FAA*) die sogenannte *Integration of Civil Unmanned Aircraft Systems (UAS) in the National Airspace System NAS Roadmap* [23], ein Plan, unbemannte Luftfahrzeuge sicher in den amerikanischen Luftraum einzubringen. Eine der größten Herausforderungen dieses Vorhabens ist wohl die Fähigkeit anderen Luftfahrzeugen zuverlässig auszuweichen. Bisher gilt die Regelung des Ausweichens bei Sichtung (*See and Avoid*). Dies muss von einem unbemanntem Luftfahrzeug automatisiert geschehen (*Sense and Avoid*).

Auch EUROCONTROL hat Bestrebungen, ferngelenkte Luftfahrzeuge (*Remotely Piloted Aircraft Systemes, RPAS*) im Zeitfenster von 2020 bis 2028 in den allgemeinen Luftraum zu integrieren [24]. Forschungen zur Umsetzung dieses Vorhabens finden als Teil des SESAR-Projekts (*Single European ATM Research*) statt [39], ein pan-europäisches Programm zur Modernisierung und Vereinheitlichung des europäischen Flugverkehrsmanagements (*Airtraffic Control, ATM*) mit dem Ziel, die Sicherheit zu erhöhen und Kosten zu senken.

Da sowohl der amerikanische als auch der europäische Ansatz, UAVs in den Luftraum zu integrieren, auf alle Größen von UAVs abzielen, sollten die resultierenden gesetzlichen und technischen Entwicklungen vorsichtig verfolgt werden.

1.7 Forschung und Entwicklung von UAS-Technik im Sensorik-Applikationszentrum

Für das Sensorik-Applikationszentrum (SappZ) ist der Quadrocopter in vielerlei Hinsicht interessant. Zum einen besteht die Elektronik aus zahlreichen Sensoren: Beschleunigungssensoren, Gyroskopsensoren und Magnetfeldsensoren für die Ermittlung der Neigung des Quadrocopters im Raum, sowie barometrische Drucksensoren zur Höhenbestimmung werden verwendet. Hier kann das Know-How des Sensorik-Applikationszentrums im Bereich der Sensorfusions- und Filteralgorithmik einfließen. Zum anderen kann der Quadrocopter als Sensorplattform dienen. Es können eigene Sensoren sowie Fremdsensoren angeschlossen werden. Ebenso kann man GPS gestützt bestimmte Messpunkte abfahren und den Quadrocopter als Werkzeug zur Messwertaufnahme in der Luft verwenden. Besonderes Augenmerk bei der Entwicklung wird dabei auf die Vereinfachung der Bedienung, den autonomen Flug und ein durchdachtes Gesamtkonzept gelegt.

Im Sensorik-Applikationszentrum wird derzeit das unbemannte Luftahrzeug (Unmanned Aerial Vehicle, UAV) **merlin UAS** entwickelt. Die Hauptanwendungen dieses Systems sind folgende:

- Inspektion von Solarfeldern, Pipelines
- Feuerwehr und Katastrophenschutz
- Sensor-Trägerplattform

Dabei wird nicht nur das UAV, basierend auf einem Quadrocopter, entwickelt sondern das komplette System mit Bodenstation, (*Unmanned Aerial System, UAS*).

1.8 Ziel der Arbeit

Eines der Hauptthemen der UAS-Forschung im Sensorik-Applikationszentrum ist die Vereinfachung der Bedienung. Gerade in den Anwendungen des Katastrophenschutzes ist dies wichtig, da im Einsatzfall wenig Zeit bleibt, sich mit der Bedienung des Gerätes auseinanderzusetzen. Deshalb ist es notwendig, eine eigene Flugsteuerung zu entwickeln, deren Flugalgorithmen beeinflusst werden können. Nur so können die Flugeigenschaften an die gewünschten vereinfachten Bedienkonzepte angepasst werden.

Da ein Quadrocopter ein instabiles System ist, müssen dessen vier Rotoren stets in einer Weise angesteuert werden, dass die Fluglage des Quadrocopters korrigiert wird. Dabei ist die Regelung der bürstenlosen Motoren nicht die Aufgabe der Flugsteuerung. Hierfür wurde eine eigene Motortreiber-Hardware [2] entwickelt. Es muss also eine Verbindung zu jenen Motortreibern bestehen. Außerdem muss geeignete Sensorik vorhanden sein, um die inertiale Orientierung bestimmen zu können.

Da der Quadrocopter am Sensorik-Applikationszentrum als Sensor-Trägerplattform verwendet wird, ist es desweiteren notwendig, dass benötigte Schnittstellen (I2C, *SPI*, *UART*, *Ethernet*, *USB*) vollständig nach außen geführt werden.

Die Flugregelung ist zeitkritisch und sollte in Echtzeit und in wenigen Millisekunden stattfinden. Für die Anbindung der Sensoren ist ein Betriebssystem sinnvoll, da hier der Integrationsaufwand zur Entwicklung von Treibersoftware wesentlich geringer ist. Deshalb fiel die Entscheidung, eine Recheneinheit zu verwenden, welche sowohl aus einem ARM-Core als auch einem DSP-Core besteht.

[2] Das Motortreiberboard (*FireMicroControl*) wurde von Johannes Fischer entwickelt

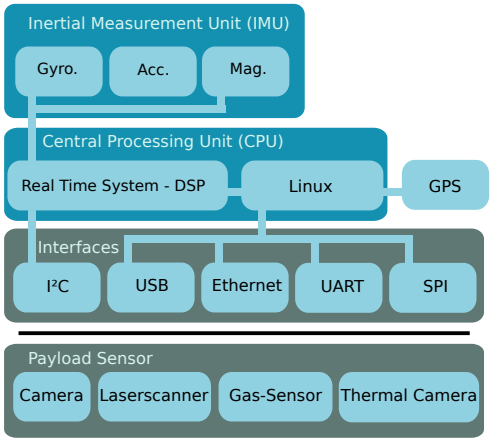


Abbildung 1.5: Anforderungen der FlightConrol

Im Rahmen dieser Arbeit wurden folgende Aufgaben zur Entwicklung jener *Flugsteuerung* durchgeführt:

- *Hardwareentwicklung der Flugsteuerung*: Es wurde basierend auf dem OMAP-L138 von Texas Instruments eine Flughardware aufgebaut. Deren Ausgang hat die Möglichkeit, mit den Motortreibern zu kommunizieren. Außerdem verfügt die Hardware über die notwendige inertiale Sensorik zur Bestimmung der Fluglage. In dieser Arbeit wurde die Hardware entwickelt, jedoch nicht getestet.
- *Simulation*: Basierend auf der Open-Source Software Gazebo wurde eine Simulation für den Test von Flugalgorithmen entwickelt. Mithilfe jener Simulation kann in Realzeit und rundenbasierend das Verhalten des Quadrocopters unter Einfluss eines Stimulus simuliert werden. Auch eine grafische Oberfläche zur visuellen Darstellung des Flugverhaltens wurde entwickelt.
- *Sensordatenfusion*: Für die Ermittlung von Flughöhe, Orientierung und Position des Quadrocopters ist es notwendig, die Werte von verschiedenen Sensoren in geeigneter Weise zu fusionieren. Hier werden sowohl Komplementär- als auch Kalmanfilter verwendet.
- *Protokolle*: Da es notwendig ist, dass verschiedene Systeme unter verschiedenen Programmiersprachen unter Verwendung von Funksystemen mit begrenzter Bandbreite mit der Flugplattform kommunizieren, ist eine Möglichkeit zur einheitlichen Serialisierung und Deserialisierung jener Daten notwendig. Auch das Speichern und Lesen von Daten muss auf verschiedenen Plattformen einheitlich ablaufen. Deshalb wurden verschiedene Bibliotheken, welche diese Anforderungen erfüllen, zusammengetragen, untersucht und verglichen.

KAPITEL 2

Hardwareentwicklung der Flugsteuerung

> Die Natur entfaltet gerade in diesen Bewegungsformen des Vogelflügels eine Harmonie der Kräftewirkungen, welche uns so mit Bewunderung erfüllen muss, dass es uns nur nutzlos erscheinen kann, wenn auf anderen Wegen versucht wird zu erreichen, was die Natur auf ihrem Wege so schön und einfach erzielt.
>
> ***Otto von Lilienthal,***
> ***Der Vogelflug als Grundlage der Fliegekunst,***
> ***Berlin 1889***

Inhalt: In diesem Kapitel wird die Entwicklung einer Flughardware für einen Quadrocopter beschrieben. Dabei wird auf die inertiale Sensorik, Pinbelegungen, technische Daten sowie Abmessungen und Schaltpläne eingegangen. Schließlich werden wesentliche Aspekte des Schaltungsdesigns sowie des Layouts geschildert.

2.1 Einleitung

2.1.1 Ziel der Arbeit

Es soll eine Flughardware für einen Quadrocopter entwickelt werden die im Wesentlichem zwei Aufgaben erfüllen soll.

- Sie dient als **Flugsteuerung** eines Quadrocopters und soll Lage- und Positionsregelung in Echtzeit übernehmen. Dafür dienen Sensorwerte inertialer Sensorik als Eingangswerte für die Bestimmung der Orientierung im Raum. Daraus werden die Ausgangswerte errechnet um die Motoren anzusteuern.

Diese werden an die bereits bestehende Motorreglerhardware über CAN- oder I2C-Bus weitergegeben.

- Desweiteren soll sie als **Sensorplattform** dienen und Treiber- sowie Steueraufgaben übernehmen. Für diese Aufgabe ist die Möglichkeit der Anbindung an ein Betriebssystem, der Echtzeitfähigkeit vorangestellt. Dadurch können Treiber für Payloadsensoren rascher entwickelt werden.

2.1.2 Stand der Technik

Die bekanntesten kommerziellen Quadrocopterhersteller sind wohl *microdrones* [8] und *Ascending Technologies* [10]. *Ascending* bedient dabei eher den Forschungsmarkt und *MicroDrones* bietet Lösungen für den industriellen Einsatz. Eine Neuerscheinung ist der *ScaraBot* der Firma *DaVinci Copters* [17]. Dieser weißt ähnlich lange Flugzeiten wie das derzeit größte Modell von *MicroDrones* (*MD-1000*) auf (mit 600 Gramm Gewicht 30 Minuten). Es existieren derzeit mehrere Closed und Open Source Flugsteuerungen. Einige der bekanntesten Open Source Flugsteuerungen wurden in [84] zusammengetragen und verglichen.

2.1.3 Rahmenbedingungen

Der Grundbaustein der Hardware ist der Dual-Core Prozessor *OMAP-L138* von Texas Instruments. Er beherbergt einen DSP- sowie einen ARM-Core. Der DSP-Core soll alle Echtzeitanwendungen ausführen. Er ist gut geeignet für die rechenintensiven Flugregelungs- und Positionsschätzungsalgorithmen. Dabei kommuniziert die ineritiale Sensorik mit diesem Teil des Kerns.

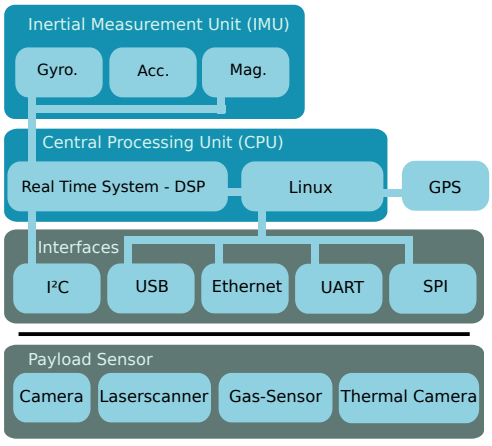


Abbildung 2.1: Grundsätzliche Struktur der Flugsteuerung

Auf dem ARM-Controller läuft ein Linux-Betriebssystem. Hier wird High-Level-Software, wie Netzwerkkommunikation oder Treibersoftware für die Payload-Sensoren ausgeführt werden. Der wesenhafte, strukturelle Aufbau der Hardware wird in Abbildung 2.1 gezeigt.

2.2 Systemübersicht

2.2.1 Schematische Struktur

Herzstück der Flugsteuerung ist das System on Module (OMAP-L138 SOM-M1) von LogicPD mit dem Prozessor OMAP-L138 von Texas Instruments. Dieses Board stellt 64MB Arbeitsspeicher zur Verfügung und läuft bei einer Frequenz von bis zu 375 MHz. Der benutzte Prozessor OMAP-L138 beherbergt zwei Prozessorkerne - einen ARM9 Prozessor und einen digitalen Signalprozessor (Fix- und Floatingpoint Arithmetik). Auf dem ARM9 läuft ein Linux-System, welches die Ansteuerung von externer Peripherie und einen Teil der Auswertung des GPS-Signals übernimmt. Der Digitale Signalprozessor ist für alle zeitkritischen Aufgaben zuständig. Hier werden hauptsächlich die Algorithmen für die Positionsregelung laufen.

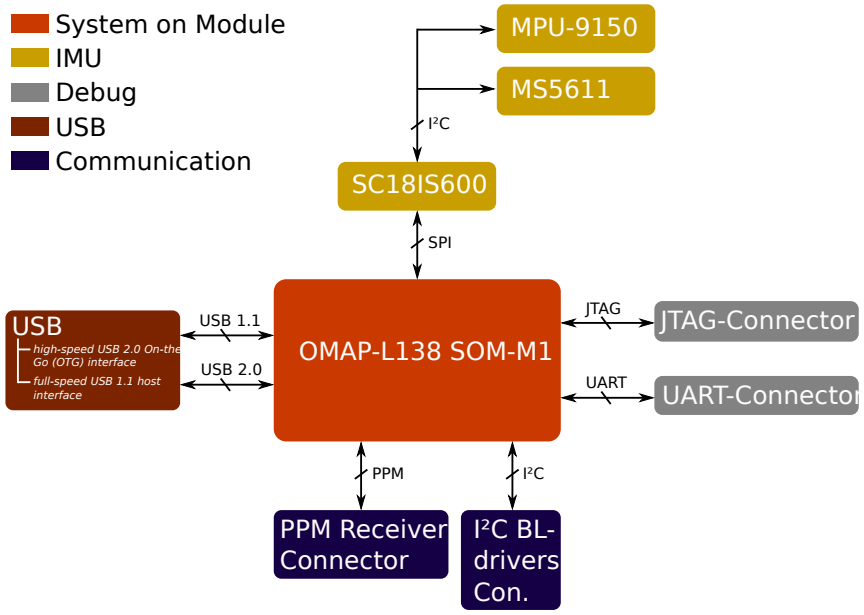


Abbildung 2.2: Blockdiagramm der Flugsteuerung

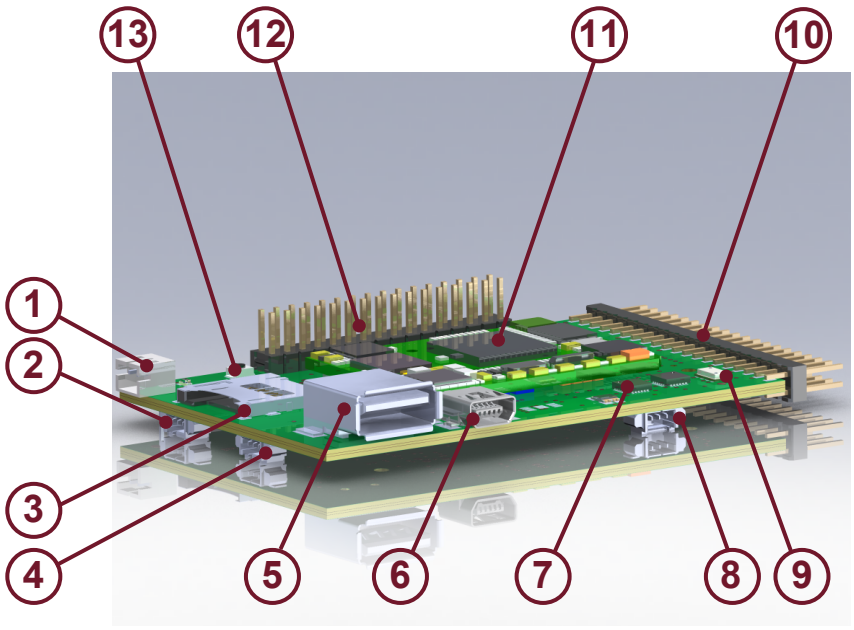
Die inertiale Messeinheit (*Inertial Measurement Unit*, IMU) besteht zum einen aus den Sensoren MPU9150, sowie MS5611. Der MPU9150 [1] beherbergt mehrere Sensoren in einem Gehäuse. Dieser IC enthält sowohl einen dreiachsigen Magnetfeld-, einen dreiachsigen Beschleunigungs- und einen dreiachsigen Drehratensensor, als auch einen Temperatursensor. Seine Ausrichtung und Position wird in Grafik 2.6 dargestellt. Der MS5611 misst den barometrischen Höhendruck. Die IMU selbst wird in Kapitel 2.3.2 genauer beschrieben. Da die inertialen Sensoren zwar beide über I2C angesteuert werden, jedoch kein freier I2C Port, aber dafür ein SPI Port für sie zur Verfügung steht, wird der SC18IS600 verwendet. Dieser Chip stellt einen I2C Master Port zur Verfügung, welcher über SPI (Slave) angesteuert werden kann.

[1] MPU9150 ist abgekündigt und sollte in zukünftigen Designs durch MPU9250 ersetzt werden. Die beiden ICs sind pinkompatibel, Treiber müssen für die Nutzung des MPU9250 angepasst werden.

2.2.2 Mechanischer Aufbau

In der Mitte des Boards befindet sich das System on Module (OMAP-L138 SOM-M1) ⑪, welches den Prozessor enthält. Es wird über drei Hirose-Stecker (DF40C-100DS-0.4V(51)) auf die Flugsteuerung gesteckt. Mehr Infos zu den technischen Daten des Prozessors und der Speicher auf dem SOM finden sich in Kapitel 2.4.

Über den Breakout-Connector ⑫ werden viele Signale für die freie Verwendung auf eine Pfostenleiste geführt. Ein weiterer Breakout-Connector ⑩ ist teilweise mit der EOMA-68 Richtlinie [11] konform. Hier kann später über ein Adapterboard ein leistungsfähiges Prozessorboard, welches der EOMA-68 Richtlinie entspricht, angeschlossen werden.



Der Stecker ① dient zur Versorgung des Boards mit 5V. In den SD-Card Stecker ③ kann die Mikro-SD-Card gesteckt werden, von welcher das Betriebssystem gebootet wird. Der Stecker ② dient als Debug-UART-Port für das Linux-System. Das Linux-System wird so eingerichtet, dass auf diesem Port die Bootmeldungen, sowie eine Standardkonsole ausgegeben werden. Auf den Stecker ④ ist der I2C-Port herausgeführt, welcher für die Ansteuerung der Motortreiber vorgesehen ist. Der Stecker ⑧ ist für den Anschluss des PPM-Receivers vorgesehen.

Der full-speed USB 1.1 Host Port ⑤ wird vor allem dafür verwendet, einen USB-Ethernet-Converter anzuschließen. Über diesen kann vom Host-Rechner aus über eine SSH-Konsole auf die Flugsteuerung zugegriffen werden. So kann die Flugsteuerung konfiguriert und programmiert werden. Der High-Speed USB 2.0 OTG Port ⑥ steht zur freien Nutzung zur Verfügung. Er kann beispielsweise später zur Konfiguration des Boards an einem Rechner über eine Benutzeroberfläche benutzt werden. Außerdem können hier große Datenmengen übertagen werden (480 Mbit/s).

Die IMU-Sensoren MPU-9150 ⑦ und MS5611 ⑨ befinden sich rechts unten auf der Flugsteuerung. Auf der Rückseite der Flugsteuerung befinden sich ein Stecker mit vier analogen Eingängen (X6) und Kontakte für einen TC2050-IDC-NL JTAG-Connector (Abbildung 2.2.2), welcher in Kombination mit dem TC-C2000-M Adapter von Tag-Connect benutzt wird. Über diesen kann man das Board mit dem XDS100v2 JTAG-Emulator von Texas Instruments debuggen und programmieren.

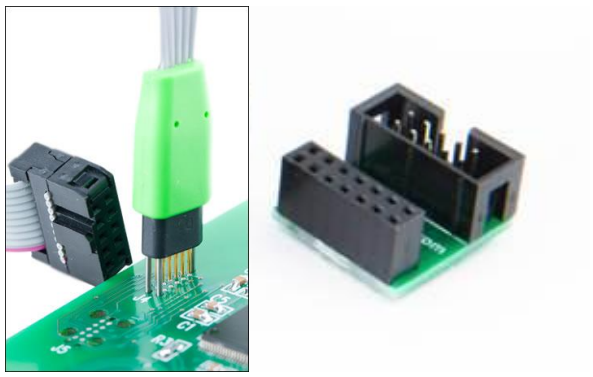


Abbildung 2.3: Tag-Connect Stecker, Links: TC2050-IDC-NL, Rechts: TC-2000-M Adapter

Die Position der Koordinatensysteme der inertialen Messung auf der Flugsteuerung werden in Kapitel 2.3.2 beschrieben. Die Pinbelegungen der einzelnen Stecker finden sich in Kapitel 2.3.1 und die Abmessungen des Boards in Kapitel 2.3.3. Die technischen Daten werden in Kapitel 2.4 angesprochen.

2.3 Aufbau

2.3.1 Pinbelegung

Breakout-Connector

Zusätzlich zu USB-Schnittstellen und den Anschlüssen für die Peripherie stellt die Flugsteuerung I2C, SPI, UART, GPIOs und Timer auf einem Breakout-Connector zur Verfügung. Abbildung 2.4 zeigt die Pinbelegung des Breakout-Connectors.

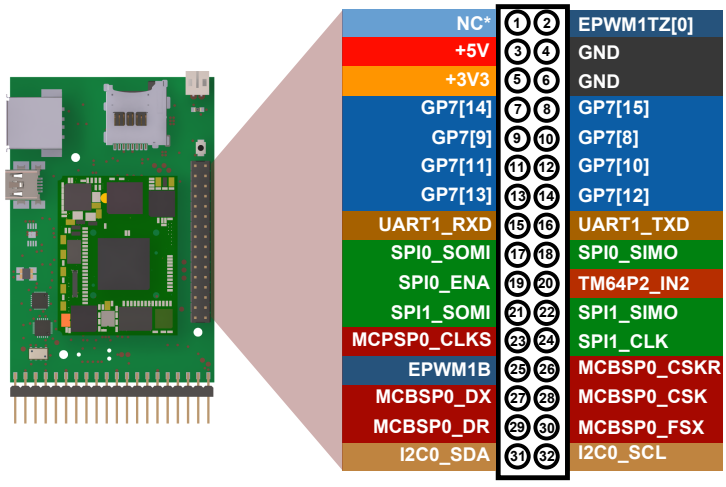


Abbildung 2.4: Pinout des Breakout-Connectors

Außerdem befinden sich 5V-, 3,3V und GND-Pins auf dem Breakout-Connector. Die Pegel-

spannungen aller Ein- und Ausgänge betragen 3,3V.

EOMA-68-Connector

Seitlich an der Flugsteuerung befindet sich ein Stecker, der soweit wie notwendig den EOMA-68 Richtlinien genügt. Die Pinbelegung wird in Abbildung 2.5 dargestellt. Hier kann bei Bedarf über ein kleines Adapterboard ein leistungsfähiges Prozessorboard, welches ebenfalls die EOMA-68 Richtlinien erfüllt angeschlossen werden. Für die Kommunikation stehen in diesem Fall eine I2C-Schnittstelle und einige GPIOs zur Verfügung.

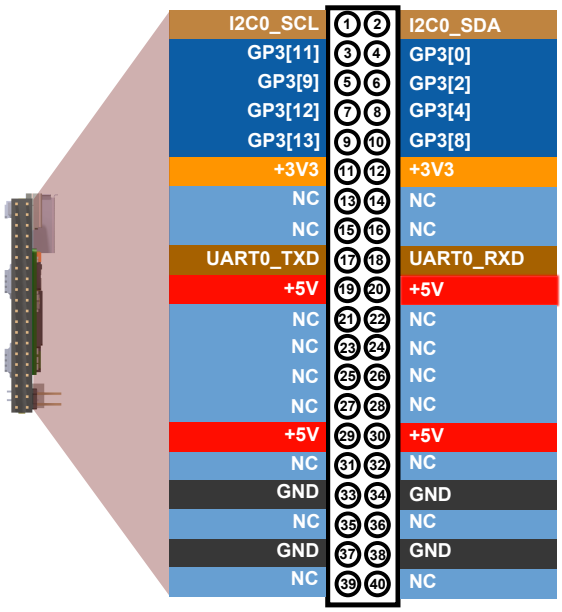


Abbildung 2.5: Pinout des EOMA-68-Connectors

Auf einige Schnittstellen, welche in der EOMA-68 Richtlinie vorgesehen sind musste verzichtet werden. Da beispielsweise der Ethernet PHY des OMAP-L138 SOM-M1 entfernt werden musste konnten keine Ethernet Signale auf den Stecker gelegt werden. Falls hohe Datenraten bei der Kommunikation zu erwarten sind, kann auf den high-speed USB 2.0 OTG Port der Flugsteuerung zurückgegriffen werden. Auch hier beträgt die Pegelspannung aller Ein- und Ausgänge 3,3V.

Receiver-Eingang

Tabelle 2.1 zeigt das Pinout des Receivers-Eingangs (X9). Da der Graupner-Receiver mit 5 V Pegel arbeitet wurden seine 3,3 V Eingänge durch einen Spannungsteiler 5V-verträglich gemacht.

Tabelle 2.1: Pinout Receiver-Eingang

Pin	Signalname
1	GND
2	+5V
3	TM64P0_IN12

I2C-Motortreiber

In Tabelle 2.2 wird das Pinout des I2C-Ports für die Motortreiber (X3) beschrieben. Dieser Stecker ist dafür vorgesehen die Motortreiber zu verbinden.

Tabelle 2.2: Pinout I2C-Motortreiber

Pin	Signalname
1	GND
2	I2C1_SCL
3	I2C1_SDA

UART-Debug-Port

Tabelle 2.3 zeigt das Pinout des UART-Debug-Ports (X1). Standardmäßig ist das Linux-System so konfiguriert, dass UART2 als Standard-Debug-Port verwendet wird [2] [1]. Da dieser Port jedoch für andere Zwecke verwendet wird, musste auf UART1 als Debug-Port ausgewichen werden. Das Linux-System muss so umkonfiguiert werden, dass die Bootmeldungen, sowie die Standardkonsole auf UART1 ausgegeben werden.

Tabelle 2.3: Pinout UART-Debug-Port

Pin	Signalname
1	UART1_RXD
2	UART1_TXD
3	GND

Analoge Eingänge

Tabelle 2.4 zeigt das Pinout des Steckers X6. Er befindet sich auf der Rückseite der Flugsteuerung und enthält vier analoge Eingänge. Die Eingangsspannung sollte 2.25 V nicht überschreiben. [AN 470]

[2] Abfragen der Bootargumente von *uBoot* mit `printenv` ergibt im Default-Zustand

```
...
bootargs=mem=32M console=ttyS2,115200n8 root=/dev/mmcblk0p2 rw rootwait ip=off
...
```

Tabelle 2.4: Pinout der Analogen Eingänge

Pin	Signalname
1	AI0
2	AI1
3	AI2
4	GND

2.3.2 Inertiale Messeinheit

Funktionsweise

Die inertiale Messeinheit (*Ineritial Measurement Unit*, IMU) wird zur Fluglagebestimmung des Fluggerätes. Sie besteht aus folgenden Sensoren:

- 3 Achsen Beschleunigungssensor
- 3 Achsen Drehratensensor
- 3 Achsen Magnetfeldsensor
- Barometrischer Drucksensor

Die Messwerte der Sensoren der IMU werden zusammen mit jenen eines GPS-Moduls zur Bestimmung von Orientierung und Position der Flugsteuerung herangezogen. Hierin werden optimale Orientierungswerte durch eine kluge Fusion der jeweiligen Sensorwerte durch numerische Algorithmen erzielt. Dabei werden Ansätze wie beispielsweise der Kalman Filter aus [96] verwendet.

Wahl der Sensoren

Drucksensoren

Um die Höhe des Fluggerätes zu bestimmen, wird der Druckunterschied zwischen dem Höhendruck und dem Druck am Boden herangezogen. Dafür wird ein barometrischer Drucksensor verwendet. Tabelle 2.5 zeigt einige barometrische Drucksensoren im Vergleich [14] [4] [7]. Alle Sensoren haben als zusätzliches Feature einen Temperatursensor integriert, mit dem die Sensoren kompensiert werden können.

Tabelle 2.5: Barometrische Drucksensoren im Vergleich

Sensor	Hersteller	Auflösung	Relativdruck Genauigkeit	Messbereich
MS5611-01BA03	MEAS-SPEC	0.012 mbar	$\pm$ 0.5 / $\pm$ 2.5 mbar	450 - 1100 mbar
BMP180	Bosch	0.02 mbar	$\pm$ 0.12 mbar	300 - 1000 mbar
MPL3115A2	Freescale	$\approx$ 0.01 mbar	$\pm$ 0.5 mbar	500 - 1100 mbar

Da die Höhe gemessen werden soll, ist die Auflösung von besonderem Interesse. Diese ist bei dem MS5611 und dem MPL3115A2 am besten und liegt auf den messbaren Höhenunterschied umgerechnet bei etwa 10 cm. Da der MS5611 bereits getestet wurde und verlässliche Werte liefert, fiel die Entscheidung auf diesen.

Beschleunigungs-/Drehraten-/Magnetfeldsensoren

Es gibt mittlerweile Sensor-ICs, welche 10 Sensor-Achsen in einem Gehäuse beherbergen:

- 3 Achsen Beschleunigungssensor
- 3 Achsen Drehratensensor
- 3 Achsen Magnetfeldsensor
- Temperatursensor

Hier werden die Sensoren BMX085 von Bosch-Sensortec [6] und MPU-9150 [13] von Invensense verglichen um eine Wahl zu treffen. Beide Sensoren können über I2C angesteuert werden und deren Gehäuse sind sehr kompakt gebaut. Die Tabellen 2.6, 2.7 und 2.8 zeigen die Sensitivität des Drehratensensors, des Beschleunigungssensor und des Magnetfeldsensors bezogen auf das LSB der jeweiligen Sensoren. Die Senssitivität der Drehratensensoren ist dabei bei dem BMX055 und dem MPU-9150 gleich gut. Jedoch ist die Sensitivität des Beschleunigungssensors und des Magnetfeldsensors des MPU-9150 höher. Da bereits Kalman-Filteransätze mit dem MPU-9150 erfolgreich getestet wurden und dabei gute Ergebnisse erzielt werden fiel die Wahl auf diesen Sensor.

Tabelle 2.6: Sensitivität Drehraten MPU-9150 und BMX055

Gyro Range	MPU-9150	BMX055
± 250 °/s	131 LSB/(°/s)	131.2 LSB/(°/s)
± 500 °/s	65.5 LSB/(°/s)	65.6 LSB/(°/s)
± 1000 °/s	32.8 LSB/(°/s)	32.8 LSB/(°/s)
± 2000 °/s	16.4 LSB/(°/s)	16.4 LSB/(°/s)

Tabelle 2.7: Sensitivität Beschleunigung MPU-9150 und BMX055

Acc Range	MPU-9150	BMX055
± 2 g	16384 LSB/g	1024 LSB/g
± 4 g	8192 LSB/g	512 LSB/g
± 8 g	4096 LSB/g	256 LSB/g
± 16 g	2048 LSB/g	128 LSB/g

Tabelle 2.8: Sensitivität MPU-9150 und BMX055

MPU-9150	BMX055
0.314 µT/LSB	1 µT/µT

Laut einem Gespräch mit Bosch-Sensortec, ist die Driftstabilität des Drehratensensor des BMX055 besser als jene des MPU-9150. Das kann bedeuten, dass der Bosch-Sensor, trotz der schlechteren

Auflösung des Beschleunigungssensors, nach der Stabilisierung des Gyro-Drifts durch die Sensorfusionsalgorithmen, gute Werte liefert. Es wurden bereits Samples geordert um dies zu testen. Zusätzlich zu dem BMX055 erscheint der BNO055, welcher zusätzlich zu den 10 Sensorachsen einen Mikrocontroller enthält. Der Flashspeicher dieses Mikrocontrollers ist teilweise mit Sensorfusionssoftware beschrieben. Diese wurde seit zwei Jahren mit verbessert. Sobald die neuen Sensorfusionsalgorithmen erscheinen, wird der neue Sensor von Bosch getestet und mit dem MPU-9150 verglichen.

Falls die Alternativen zu dem MPU-9150 wesentlich bessere Werte liefern, werden diese in zukünftigen Designs bevorzugt verwendet. Der BMX055 wird ohnehin bereits als redundanter Sensor auf dem Motortreiber-Board verbaut.

Orientierung der IMU-Sensoren

Folgende Grafik zeigt, die Orientierung des Beschleunigungs-, Drehraten und Magnetfeldsensors (MPU-9150). Außerdem ist die Position des MPU-9150 (IMU-Sensor) bezogen auf die Mitte bemaßt.

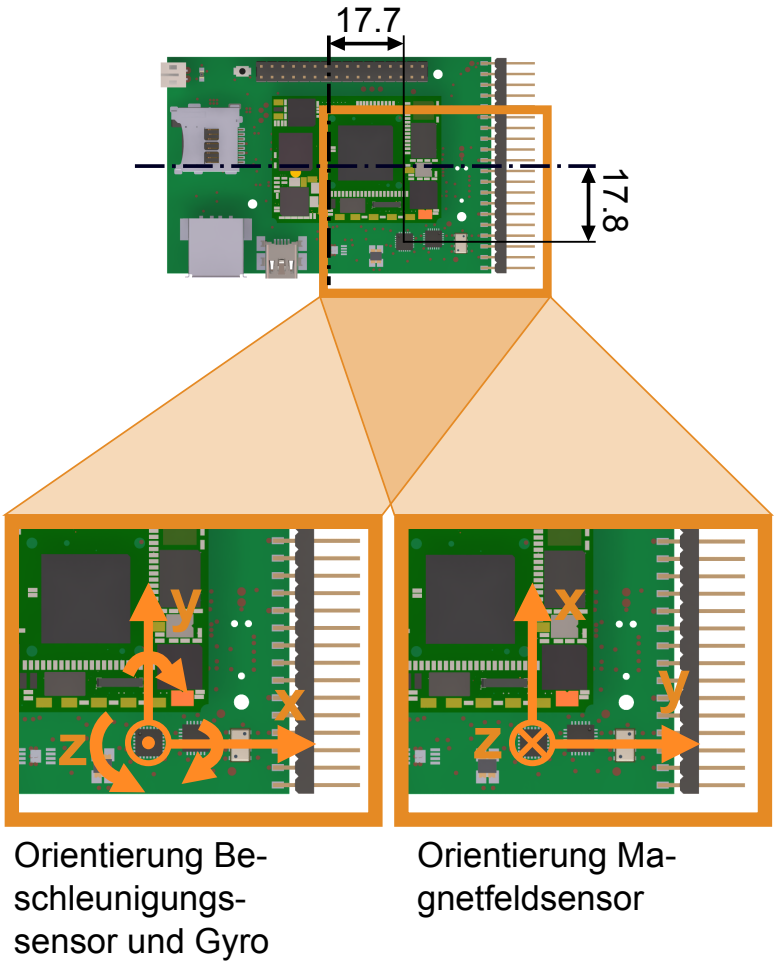


Abbildung 2.6: Orientierung der inertialen Messeinheit auf der Flugsteuerung

2.3.3 Abmessungen

Abbildung 2.7 zeigt die Abmessungen der Flugsteuerung, bemaßt in der Draufsicht. Außerdem sind die Positionen der Befestigungsbohrungen eingezeichnet und bezogen auf die linke untere Ecke bemaßt.

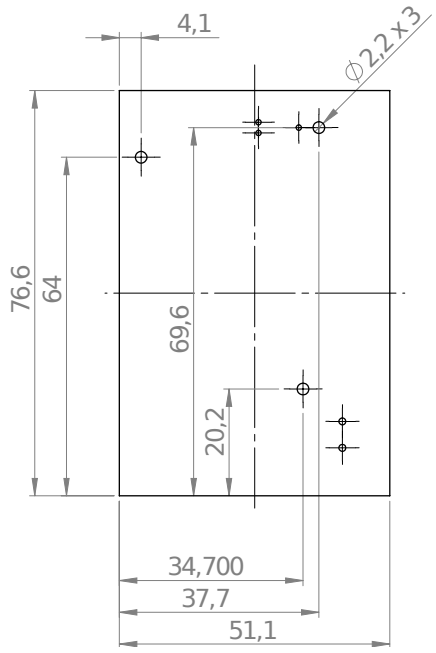


Abbildung 2.7: Außenabmessungen und Position der Bohrungen der Flugsteuerung

2.4 Technische Daten

2.4.1 Features

- Dual-Core DSP und ARM-Prozessor in einem Gehäuse
- Realtime-Betriebssystem für zeitkritische Aufgaben auf DSP
- Linux Board Support Package auf ARM-Controller beispielsweise für Schnittstellentreiber
- USB 2.0 high-speed On-The-Go Interface
- USB 1.1 full-speed host Interface
- I2C-, SPI-, UART-, GPIO-Ports, GND, 5V und 3,3V auf Breakout-Connector
- Teilweise Unterstützung der EOMA-68 Richtlinie [11]
- Vier analoge Eingänge

2.4.2 Specs

- Abmessungen: **LxBxH: 75,6x51,1x50,0 mm**
- Gewicht: **30g**
- Spannungsversorgung: **5 V**
- Stromaufnahme: **200 mA**
- Prozessor: **OMAP-L138 Dual-Core ARM 926EJ-S und C6848 VLIW DSP, 375 MHz** [19]
- Speicher: **8MB Flash, 166 MHz DDR SDRAM, 64 oder 128 MB** [19]
- IMU: **MPU-9150, MS5611** [13], [14]

2.5 Schaltungsdesign

2.5.1 Signalübersicht und Überprüfung des Pinmultiplexing

TIs Pinmultiplexing Utility

Der OMAP-L138 Prozessor benutzt Pinmultiplexing um die Ausgangspins mit mehreren wählbaren Funktionen zu versehen. Um Kollisionen konkurierender Signale durch Mehrfachbelegung von Pins zu vermeiden, stellt Texas Instruments ein Software-Tool namens "Pin Multiplexing Utility" [3] [15] zum Design der Pinbelegungen zu Verfügung. Dieses Tool wurde verwendet um zu ermitteln, welche Pin-Funktionen verwendet werden können und ob es Überschneidungen gibt. Dabei musste auch darauf geachtet werden, dass keine Pins verwendet werden, welche vom OMAP-L138 SOM-M1 intern vorgesehen sind. Dafür wurden hauptsächlich die Designunterlagen von LogicPD - *OMAP-L138 SOM SOM-M1 Hardware Spec*, sowie der *Schaltplan des OMAP-M1 SOM-M1* - beide verfügbar im User-Bereich der LogicPD Homepage [9], herangezogen.

Warum die Entscheidung auf die jeweiligen Pin-Funktionen fiel, wird in den folgenden Kapiteln beschrieben. Es werden auch Signale berücksichtigt, die nur intern auf dem SOM genutzt werden. Hier werden jedoch nur Signale berücksichtigt, die für die Verwendung der Flugsteuerung relevant sind.

I2C

Es werden auf der Flugsteuerung zwei I2C-Ports benötigt. Ein Port wird für die Ansteuerung der Motoren verwendet. Der andere **I2C**-Port ist für das Auslesen der IMU-Sensoren, die beide über **I2C** angesteuert werden, notwendig. Auf dem Board befinden sich zwei **I2C** Ports:

- **I2C0:** Wird auf dem OMAP-L138 SOM-M1 für die Kommunikation mit dem Powermanagment IC verwendet. Dieser ist für systemkritische Funktionen wie Reset und das Powersequencing auf dem SOM zuständig.

[3] In diesem Rahmen wurde mit der *Pin Multiplexing Utility* in der Version 1.0 gearbetet.

- **I2C1:** Wird auf dem OMAP-L138 SOM-M1 in der Pinmultiplexing-Konfiguration des Linux Systems standardmäßig als UART2 festgelegt. Die Signale dienen als Standard-Terminal.

I2C1 wird für die Kommunikation mit den Motortreibern verwendet. Hierfür wird das Signal auf einen Stecker gelegt. Die Signal-Pins von **I2C1** haben bei dem OMAP-L138 zusätzlich die Funktion von **UART2**. Dieser Port dient dem Linux-System des OMAP-L138 SOM-M1 als Standard-Konsole. Deshalb ist es hier notwendig zum einen die Pinmultiplexing-Funktionen des Linux Systems auf **I2C** umzustellen und zum anderem, in U-Boot [4] einen anderen seriellen Port für die Ausgabe der Boot-Sequenz einzustellen.

I2C0 ist für die Kommunikation zwischen dem OMAP-L138 und dem Power-Management IC auf dem SOM zuständig. Diese Kommunikation ist systemkritisch. Da sowohl für die Kommunikation mit den Motortreibern, als auch für jene mit den Sensoren viel Buslast erwartet wird, wird auf die Nutzung des **I2C0** Ports für diesen Zweck verzichtet. Es soll verhindert werden, dass die systemkritische Kommunikation auf dem SOM durch zu viel Kommunikation auf dem Bus blockiert wird. Da für die Kommunikation mit den IMU-Sensoren noch ein **I2C**-Bus benötigt wird, jedoch alle verfügbaren Busse in Verwendung sind, wird der IC SC18IS600 von NXP verwendet. Dieser wird über SPI angesteuert und fungiert hier als Slave. Er stellt einen **I2C**-Master-Port zur Verfügung.

SPI

Für den SC18IS600 wird ein SPI-Port benötigt. Auf dem OMAP-L138 SOM-M1 stehen zwei SPI-Ports zur Verfügung:

- **SPI0:** Konkuriert mit den MII-Signalen und wird auf dem SOM für die Kommunikation zwischen OMAP-L138 Prozessor und LAN8710 Ethernet PHY verwendet.
- **SPI1:** Von SPI1 ist als Default-Boot Quelle für den OMAP-L138 SOM-M1 definiert. Ein serieller Flash-Baustein ist an SPI1_CS0 angeschlossen

Da **SPI0** Standard Boot-Quelle ist und während des Bootvorganges von Linux verwendet wird, wird darauf verzichtet diesen für den SC18IS600 zu verwenden. Es soll verhindert werden, dass der SPI-Port für den DSP während des Bootvorganges des Linux-Systems nicht zugänglich ist. Dies hat sicherheitstechnische Gründe, da über diesen **SPI**-Port die Sensordaten erhalten werden. Falls das Linux System während des Fluges einen Software-Reboot durchführt, soll der DSP dadurch nicht gehindert werden den SPI Bus zu verwenden. Die Sensorwerte werden für die Fluglageregelung benötigt und haben damit systemkritische Funktion.

Stattdessen fiel die Entscheidung auf den **SPI1** Bus. Dieser wird für die Kommunikation zwischen OMAP-L138 Prozessor und dem LAN8710 Bus verwendet. Da auf die Verwendung der Ethernet Funktion auf dem Board verzichtet wird, kann der **SPI1** Port verwendet werden. Da der Ethernet PHY auf dem SOM nicht vom SPI-Port abgetrennt werden kann, wurde er abgelötet. Alternativ ist es möglich den IC durch einige 0 Ohm Widerstände von der IO-, Core-, und Versorgungsspannung abzutrennen, dies muss jedoch erst getestet werden. Es ist unklar, wie sich die TTL Ein-, und Ausgänge im unbestromten Zustand am Bus verhalten.

[4] Auf dem OMAP-L138 SOM ist standardmäßig der U-Boot Bootloader installiert um das Linux-System zu booten. U-Boot (*Universal Bootloader*) ist ein Bootloader, der vorwiegend auf eingebetteten Systemen zum Einsatz kommt, um den Kernel des verwendeten Betriebssystem in den Arbeitsspeicher zu laden. [5]

UART

UART2 ist standardmäßig der Debug Port des Linux Systems. Da dieses Signal mit dem I2C1 Port des OMAP-L138 Prozessor gemultiplext wird, muss hier auf **UART1** ausgewichen werden. Hierfür müssen die Einstellungen für U-Boot (Ausgabe der Boot-Meldungen), sowie die des Linux Systems (Standard-Konsole) angepasst werden.

Timer

Die Timer TM64P2_IN12 und TM64P3_IN12 werden auf einen Breakout-Connector gelegt. Bei beiden Signalen handelt es sich um Timer, die auf eine Event an den jeweiligen Pins reagieren können. Der Timer TM64P0_IN12 ist für die Auswertung des PPM-Signals des Receivers vorgesehen. Da genug Platz zur Verfügung steht wird zusätzlich der Timer TM64P1_IN12 auf den Stecker gelegt. Dieser kann zusätzlich wahlweise als Watchdog Timer verwendet werden.

Ähnlich wie die SPI0-Leitung werden auch die Timer-Pins mit den MDIO/MII-Signalen gemultiplext. Der Ethernet PHY vom Board entfernt wurden und kann somit nicht mehr mit den jeweiligen Pins interagieren. Ob es alternativ ausreicht den Ethernet PHY-IC nur von der Spannungsversorgung abzutrennen muss noch getestet werden.

MMC/SD

Für das Booten des Linuxsystems wird eine SD Flashspeicher Karte verwendet. Hierfür wird der MMC/SD0 Port verwendet. Die Überprüfung des Pinmultiplexing hat ergeben, dass die Verwendung dieses Signals nicht mit anderen kollidiert.

2.5.2 Versorgung

ICs, Sensoren, sonstiges:

Alle ICs, sowie die Micro-SD-Card, werden mit 3,3V versorgt und arbeiten mit einer Pegelspannung von 3,3V. Sie können dabei wahlweise über die Pegelspannung (M_3V3_IO) des SOM oder über einen auf dem Board befindlichem Spannungsregler versorgt werden, je nachdem ob die 0-Ohm Widerstände R10, R11 und R41 oder R40 bestückt werden. Es wird jedoch nicht empfohlen das M_3V3_IO Netz sehr zu belaseten. Bei der Wahl des 3,3 V Spannungsreglers wurde darauf geachtet, dass dessen Spannung nicht zu weit von jenen 3,3 V des 3.3V_or_1.8V Netzes des Omap-L138 SOM-M1 abweicht. Die Schaltung wird in Abbildung 2.8 dargestellt.

OMAP-L138 SOM-M1

Das SOM-L138-Modul hat folgende Spannungseingänge:

- **5V:** Der 5V Eingang ist der Hauptspannungsanschluss. Sobald 5V angeschlossen sind, wird diese Spannungsquelle allen anderen Eingängen bevorzugt. Wenn die hier die 5V-Spannung ordnungsgemäß vorhanden ist, werden der TPS65070 PMIC und der OMAP-L138 Prozessor sofort hochfahren und laufen. Über diese 5V wird die Flugsteuerung über die 5V am Eingang der Flugsteuerung versorgt.

- **USB0_VBUS:** USB0_VBUS ist eine optionale Spannungsquelle. Sobald eine Spannung an USB0_VBUS-Eingang vorhanden ist, wird das Power-Management IC TPS65070 diese Spannung bevorzugt verwenden. Alternativ zu den 5V am Eingang kann die Flugsteuerung über USB0_VBUS versorgt werden. Hierfür muss der 0 Ohm Widerstand R42 bestückt werden (siehe Abbildung 2.8). Dies ist nützlich, wenn die Flugsteuerung für Debugzwecke am Schreibtisch verwendet wird.
- **MAIN_BATT_IN:** Der MAIN_BATT_IN-Eingang ist dafür vorgesehen eine Lithium-Ionen Batterie anzuschließen. Der TPS65070 PMIC wird das Board von dieser Spannung nur dann versorgen, wenn weder am 5V-Eingang, noch am USB0_VBUS-Eingang eine Spannung anliegt. Wenn eine passende Spannung am MAIN_BAT_IN-Eingang anliegt, werden der TPS65070 PMIC und der OMAP-L138 Prozessor nicht sofort hochgefahren. Erst muss am PMIC_PB_IN Pin kurzzeitig eine Spannung angelegt werden. Der TPS65070 PMIC kann die an MAIN_BATT_IN angeschlossene Batterie sowohl über den 5V- als auch über den USB0_VBUS-Eingang laden. **Dieser Spannungseingang wird in der Flugsteuerung nicht benutzt.**
- **VRTC_BATT:** Der VRTC_IN-Eingang versorgt das onboard RTC (Real-Time-Clock) Modul. Um den Systemclock zu erhalten, sollte dieser Anschluss immer angeschlossen werden. Er wird mit 3,3 V versorgt.

Außerdem verfügt die Flugsteuerung über den Spannungsausgang **3.3V_or_1.8V**. Es wird empfohlen diesen Ausgang für Pegelspannungen zu verwenden. Auf der Flugsteuerung wird dieser Ausgang zum **M_IO_3V3**. Da die Sensoren auf der Flugsteuerung sehr wenig Strom brauchen, ist es denkbar, dass diese anstatt über das 3.3V Netz, über die Spannung **M_IO_3V3** versorgt werden. Dafür müssen die 0-Ohm Widerstände R10, R11 und R41 bestückt und der 0-Ohm Widerstand R40 entfernt werden (siehe Abbildung 2.8).

Verbrauch:

Laut Datenblatt sollte das OMAP-L138 SOM-M1 im Betrieb 220 mA benötigen. Um den Verbrauch zu testen, wurde das Evaluationsboard ohne Displayboard betrieben. Dabei war nur ein Schnüffelstück angeschlossen um den Bootnachrichten in einem Terminalprogramm sehen zu können. Dabei ergab sich jedoch ein Verbrauch von 300 mA.

2.5.3 Clocks

Das OMAP-L138 SOM-M1 ist mit zwei Quarzen ausgestattet. Einer der Quarze generiert den Systemtakt für den Prozessor und die Peripherie. Der zweite Quarz ist für das Real Time Clock Modul des L138 zuständig.

- **24 MHz:** Der Omap-L138 Prozessor beinhaltet on-chip Phase Locked Loops (acsppll), die aus dem 24.000 MHz Quarz die Taktfrequenz für den Prozessorkern und die Peripherie erzeugen. Die maximale Prozessorfrequenz beträgt 300 MHz. Die PLL0 ist über **J3.18: uP_OBSCLK** vom SOM herausgeführt. Dieser Pin wird mit einem Testpunkt versehen (TP*).
- **32,768 MHz:** Der Omap-L138-Prozessor verfügt über ein Real Time Clock Modul (RTC), welches über einen 32.768 MHz Quartz versorgt wird. Diese dient als Zeitreferenz auf dem

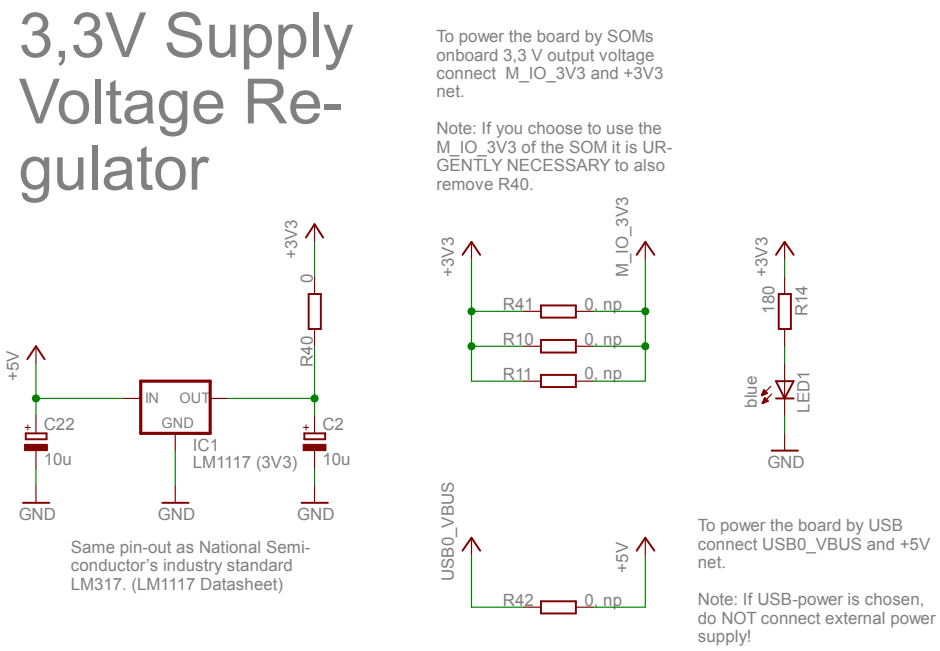


Abbildung 2.8: Schaltung der Spannungsversorgung der Flugsteuerung

Omap-L138. Das RTC Modul wird über **J2.64: VRTC_IN** versorgt. Auf der Flugsteuerung heisst dieser Pin **uP_RTC_BATT**. Er wird mit 3,3 V versorgt. Wahlweise kann man einen 0 Ohm Widerstand auslöten und das Board über Testpunkte mit einer Lithium-Ionen-Knopfzelle versorgen.

Da das Omap-L138 SOM-M1 bereits mit allen notwendigen Taktquellen ausgestattet ist und für die Peripherie keine zusätzlichen Taktquellen notwendig sind, befinden sich keine Quarze oder Quarz-Oszillatoren auf der Flugsteuerung.

2.5.4 Reset

Das OMAP-L138 SOM-M1 Modul hat bereits einen internen Pull-Up Widerstand am **uP_RESETn** Signal. Deshalb wird hier kein Pull-Up Widerstand mehr benötigt. Es wird empfohlen einen 0.1 µF Kondensator in der Nähe des Reset Pins zu platzieren um durch ESD verursachte Resets zu reduzieren [3]. Außerdem wird ein Software Reset durch einen Transistor, der durch einen I/O (**GP8_[11]**) des OMAP-L138 geschalten werden kann, ermöglicht. Die Schaltung wird in Abbildung 2.9 dargestellt.

2.5.5 Testpunkte

Die Flugsteuerung stellt folgende Testpunkte zur Verfügung:

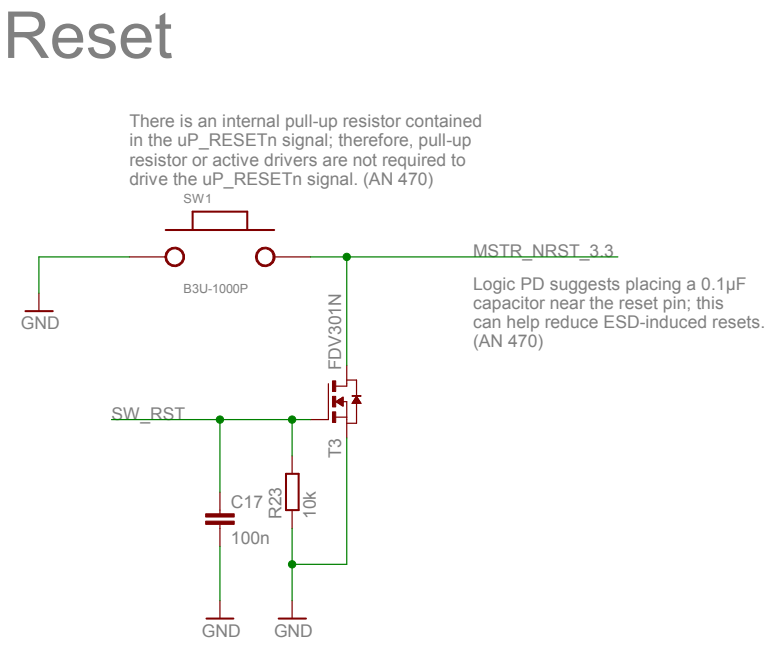


Abbildung 2.9: Resetschaltung

- *BOOT[1], BOOT[2], BOOT[3], BOOT[4]*
- *I2C_MS5611_SCL, I2C_MS5611_SDA*
- *TM64P1_IN12*
- *3.3_1.8_IO_VOLT_SEL*
- *UP_OBSCLK*
- *I2C0_SDA, I2C0_SCL*
- *SPI1_CLK, SPI1_SOMI, SPI1_SIMO*
- *RESETOUT_3.3_1.8*
- *M_IO_3V3*
- *SPI1_SCS[4]/UART2_TXD/I2C1_SDA, SPI1_SCS[5]/UART2_RXD/I2C1_SCL*
- *ES_DA, ES_CL*
- *~EN1 und ~EN2 (TPS2042, IC7)*
- *GND*

2.5.6 User LEDs

Die Flugsteuerung verfügt über zwei User-LEDs:

- **LED1:** LED1 ist eine blaue LED, welche anzeigt dass die 3,3V Spannung anliegt.

- **LED2:** LED2 ist eine zweifarbig LED (rot und grün). Damit lassen sich drei Farben darstellen: Rot, grün - und durch das Mischen beider Farben gelb. Diese LED kann als Statusanzeige verwendet werden.

2.5.7 Debug Interface

Über das Debug-Interface kann ein korrupter Flash-Speicher wiederhergestellt, eine Echtzeitanwendung debugged, oder ein DSP-Projekt entwickelt werden. Das JTAG-Interface besteht aus den Signalen **TDI**, **TMS**, **TCK**, **TDO**, **nTRST**, **RTCK**, **EMU0** und **EMU1**. Diese Signale werden auf Pads gelegt, welche über einen speziellen Stecker von Tag-Connect angeschlossen werden können. Es wurde der 10-polige Stecker **TC2050-IDC-NL** in Verbindung mit dem Adapter **TC-C2000** gewählt (siehe Abbildung 2.2.2). Der Adapter TC-C2000 zieht EMU0 und EMU1 durch Pull-Ups auf VCC. Außerdem konnten durch den Adapter einige Pins eingespart werden indem einige GND-Leitungen auf eine einzelne zusammengefasst werden und das Signal EMU_STS auf GND gelegt wurden. Dadurch ergibt sich ein sehr kleines Footprint. Durch den Debugstecker kann der OMAP-L138 Prozessor über die JTAG Schnittstelle mit dem XDS100v2-Debugger angesprochen werden.

Der XDS100v2 ist ein kostengünstiger JTAG-Emulator, der von Zulieferfirmen von Texas Instruments hergestellt wird. Er ist eine Weiterentwicklung des XDS100 von Spectrum Digital [22]. Abbildung 2.10 zeigt den Debugger. Mit ihm lassen sich JTAG-Interfaces von Texas Instruments Hardware ansprechen. Es ist kompatibel mit der Code Composer Studio Entwicklungsumgebung. Eigentlich handelt es sich bei dem XDS100 JTAG Emulator um ein Reference Design, das man auf der eigenen Hardware bestücken kann, oder als verwendbare Hardware kaufen kann. In diesem Fall wird eine fertige Hardware verwendet.

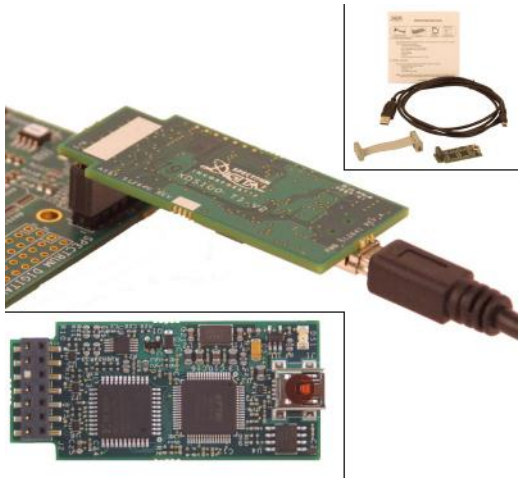


Abbildung 2.10: XDS100v2 von Spectrum Digital [22]

Der XDS100 ist etwas langsamer als die XDS510 und XDS560 Emulatoren. Das heißt, dass Programm- und Datendownloads sowie das Steppen durch C-Code beim Debuggen länger dauert. Dafür ist der XDS100 kostengünstiger. [22]

2.5.8 Sensoren IMU

Wie in Kapitel 2.3.2 beschrieben besteht die IMU aus folgenden Sensoren

- **MPU-9150:** 3 Achsen Beschleunigungs-, 3 Achsen Drehraten-, 3 Achsen Magnetfeld- und Temperatursensor
- **MS5611:** Hochpräzises Drucksensormodul

Beide Sensoren werden über I2C versorgt. Da jedoch kein freier I2C-Port des OMAP-L138 SOM-M1 zur Verfügung steht (siehe Kapitel 2.5.1) wurde ein freier SPI Port in Verbindung mit einem SPI-I2C Converter verwendet (**SC18IS600, NXP**). An dem von diesem IC zur Verfügung gestelltem I2C-Port werden der **MPU-9150** und der **MS5611** betrieben. Alternativ kann das Signal des MS5611 auch direkt an die I2C-Ausgabe des MPU-9150 angehängt werden. Hierfür müssen die 0-Ohm Widerstände R19 und R24 entfernt und die 0-Ohm Widerstände R30 und R31 bestückt werden.

2.5.9 USB

Das Omap-L138 SOM-M1 unterstützt einen USB 1.1 full-speed Port (USB1) und einen USB 2.0 OTG Port (USB0). Deshalb befinden sich auf der Flugsteuerung

- ein **USB 2.0 OTG Port** Kann sowohl als Master als auch als Host fungieren. Außerdem kann über ihn die Flugsteuerung versorgt werden (z.B. bei der Programmierung).
- ein **USB 1.1 full-speed Port:** Hier können USB-Geräte and die Flugsteuerung angeschlossen werden. Beispielsweise kann hier ein USB-Ethernet Converter angeschlossen werden um über *telnet* auf das Board zuzugreifen.

Der TPS2042 Controller von TI schaltet die 5V Spannung auf die USB-Ports, je nachdem ob das Board gerade als Host fungiert. Die Singale USB0_PWR_EN und USB1_PWR_EN (USB0_DRVVBUS, am OMAP L138 Prozessor) schalten die Spannungen entsprechend auf die USB-Ports. (Siehe: TMS320C674x/OMAP-L1x Processor Universal Serial Bus 2.0 (USB2.0) Controller - USER GUIDE) Das Signal **UHPI_HRDY**, wird von den Over-Current Ausgängen des TPS2042 geschaltet und regelt den Zugriff auf das Host Prozessor Interface. Sobald der Strom über einen bestimmten Wert steigt, begrenzt der TPS2042 auf einen Maximalstrom. Wenn ein Kurzschluss auftritt und die Temperatur über einen bestimmten Wert steigt, schaltet der TPS2042 ab und schaltet erst wieder durch, wenn die Temperatur unter einen bestimmten Wert sinkt.

Da an USB0 ein Host angeschlossen werden kann, kann die Schaltung über USB0 versorgt werden. Deshalb wird die Spannung an USB0 auf den Eingang **USB_VBUS** gelegt. Dies ist ein optionaler Spannungseingang, über den das SOM-L138 versorgt wird, wenn am 5V-Eingang nichts anliegt. Um USB0 als Spannungsversorgung benutzen zu können müssen das 5V Netz und das **USB_VBUS** Netz verbunden werden. Hierfür muss muss der 0 Ohm Widerstand R42 bestückt werden (siehe Abbildung 2.8). Wird das Board über USB versorgt, ist darauf zu achten, dass keine externe Spannungsversorgung angeschlossen wird.

Die Leitungen D+/D- Signale der jeweiligen USB-Ports sind differentielle Signale und deren Leiterbahnen sollten deshalb parallel geroutet werden. Außerdem sollten sie mit einer Impedanz von

90 Ohm über eine Ground Plane geroutet und möglichtst kurz gehalten werden. [USB_AN], [3]. Dies wird in Kapitel 2.6.3 beschrieben.

2.5.10 Micro SD-Card

Die SD-Karte ist an den **Multimedia Card (MMC)/Secure Digital SD Card Controller** des OMAP L138 angeschlossen [20]. Entsprechend dem nativem SD-Karten-Protokoll [18] sind die Signale wie folgt angeschlossen:

- **CLK:** SD0_CLK
- **CMC:** SD0_CMD
- **DAT:** SD0_DATA0; SDO_DATA1; SD0_DATA2 SD0_DATA3

Ist das Signal **Write Protect (WP)** vorhanden, muss es an **uP_EMIFA_A17** angeschlossen werden. Wird das **Card Detect (CD)** Signal verwendet, muss es an **uP_EMIFA_A16** angeschlossen werden. [3]

- Da hier eine Micro SD-Card verwendet wird, ist das **Write Protect** Signal nicht vorhanden und wird deshalb auf GND gelegt
- Der verwendete SD-Card Halter unterstützt das Signal **Card Detect** (siehe Abbildung 2.11). Ein Kontakt zwischen zwei Pins wird geschlossen sobald eine SD-Karte eingesteckt wird. Einer dieser beiden CD-Pins wird auf Masse gelegt und der Andere wird mit **uP_EMIFA_A16** verbunden, damit dieses auf Low gezogen wird sobald eine SD-Karte eingesteckt wurde.

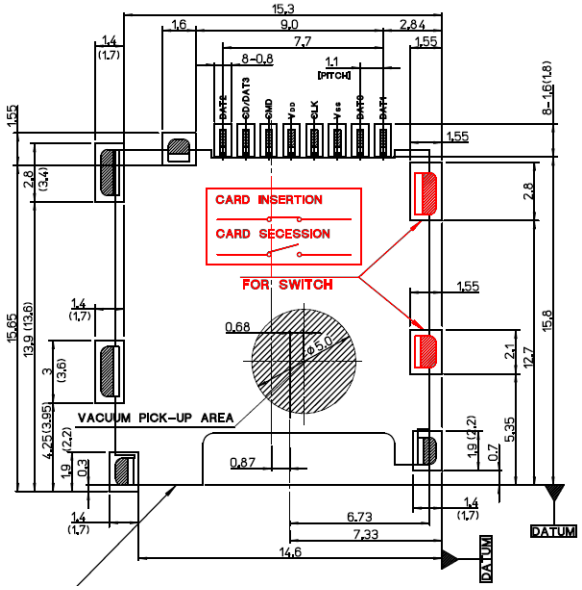


Abbildung 2.11: Molex SD-Kartenhalter (Art.Nr.49225-0821)

2.6 Layout

2.6.1 Aufbau der Lagen

Das PCB der Flugsteuerung besteht aus 4 Lagen. Deren Belegung wird in Abbildung 2.12 dargestellt.

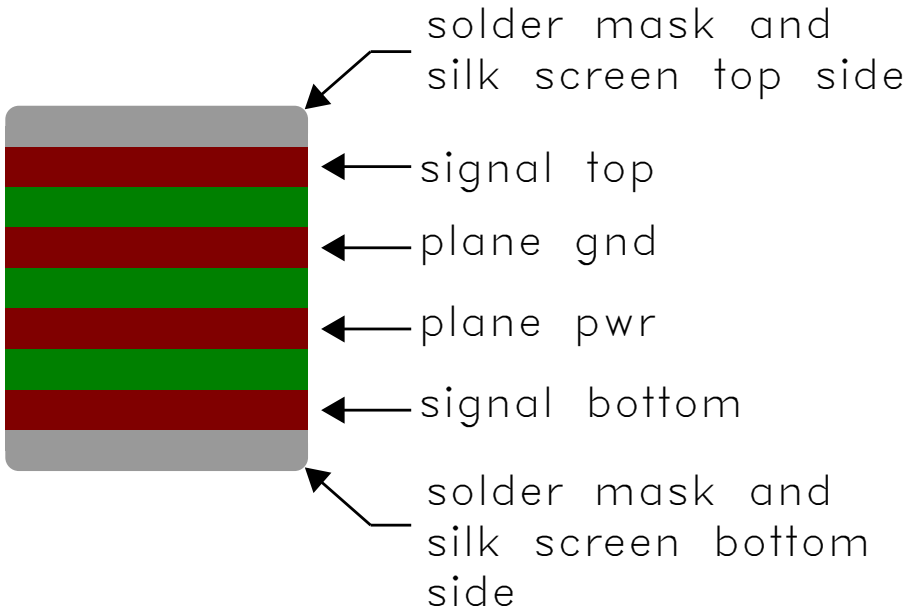


Abbildung 2.12: Aufbau der Lagen der Flugsteuerung

Die Signallagen wurden auf die äußeren Lagen gelegt. Dies hat den Vorteil dass sie noch erreichbar sind und somit Layoutfehler ausgebessert werden können. Außerdem gibt es Leitungen auf dem TOP-Layer welche impedanzangepasst verlegt werden (siehe Kapitel 2.6.3). Hier ist ein möglichst geringer Abstand des Signallayers zum GND-Layer ohne Zwischenlayer nötig [2].

2.6.2 Toleranzen

Die **Leiterbahnbreite** und die **Abstände** zwischen Leiterbahnen wurde auf mindestens **5 mil** festgesetzt. Laut dem Leiterplattenhersteller (*Hofmann Leiterplatten*) ist diese Toleranz bei einer **CU-Auflage** von **25 µm** machbar. Abbildung 2.13 zeigt den Aufbau einer von *Hofmann Leiterplatten* hergestellten Leiterplatte.

Mit dem Wissen, dass die Dicke der Leiterplatte insgesamt 1,5 mm, sowie die Dicke der CU-Auflage 0,25 µm beträgt, kann aus den Maßen von Abbildung 2.13 der Abstand zwischen Signal- und GND-Layer berechnet werden:

$$H = \frac{(1,5 - (0.025 \cdot 2 + 0.035 \cdot 2 + 0,710))}{2} mm \approx 0,32mm \hat{=} 12,6mil$$

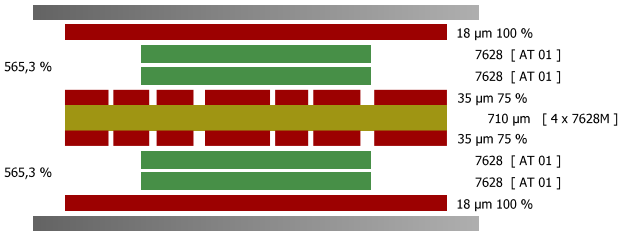


Abbildung 2.13: Schichtdicken einer vierlagigen Leiterplatte von *Hofmann Leiterplatten*

Die **Innenbohrdurchmesser der Vias** beträgt mindestens **0.26 mm**. Diese kleinen Durchmesser mussten gewählt werden um genügend Platz unter den Steckern des SOMs zu haben. Wären größere Durchmesser gewählt worden, hätte es Überschneidungen gegeben. Laut dem Leiterplattenhersteller, *Hofmann Leiterplatten*, stellen jedoch Bohrdurchmesser von bis zu 0.2 mm kein Problem dar.

2.6.3 USB

Die USB-Leitungen müssen als differentielle Leitungspaare mit einer Impedanz von 90 Ohm verlegt werden [2]. Folgende Formel zeigt wie sich die Impedanz aus den Abmessungen der Leiterplatte berechnet:

$$Z_{diff} = 2Z_0(1 - 0.48e^{-0,96S/H})$$
$$Z_0 = (87/(\epsilon_r + 1,41)^{0,5})ln(5,98H/0,8W + T)$$

Tabelle 2.9: Berechnung der Impedanz von Differenziellen Leitungen

Variable	Beschreibung
Z_{diff}	Differenzielle Impedanz von zwei parallelen Leiterbahnen auf einer Signallage
Z_0	Impedanz einer Leiterbahn auf einer Signallage
W	Breite einer Leiterbahn
H	Abstand der Leiterbahn zur GND-Lage
T	Trace Thickness
S	Abstand zwischen den differenziellen Leiterbahnen
ϵ	Relative Permeabilität des Platinenmaterials (FR4 $\approx$ 4.5)

Setzt man die Werte aus Kapitel 2.6.2 in diese Formel ein Berechnet sich die Impedanz zu:

$$Z_{diff} = 128\Omega$$

2.6.4 Footprint OMAP-L138 SOM-M1

Abbildung 2.14 zeigt die Vorgaben für die Platzierung der Stecker für das OMAP-L138 SOM-M1. Diese wurden im Layout eingehalten.

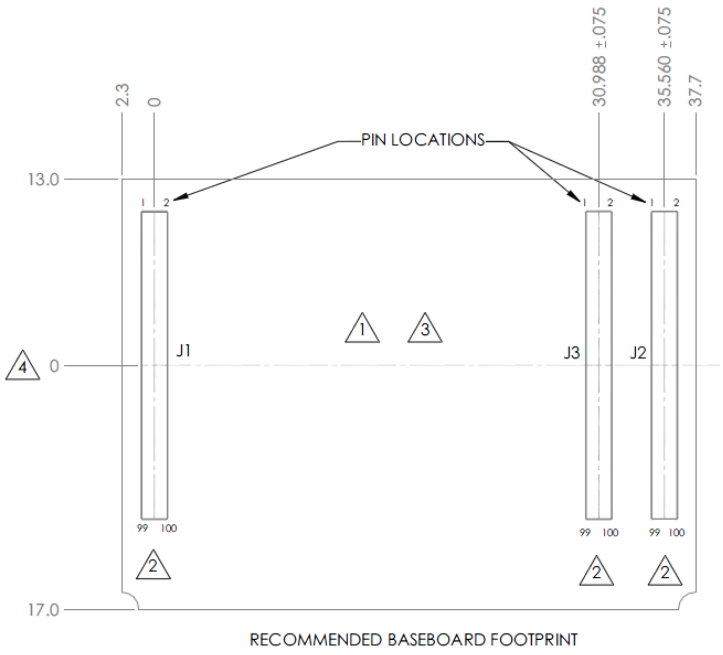


Abbildung 2.14: Footprint des OMAP-L138 SOM-M1

Um sicherzugehen, dass die Stecker während der Layout-Phase nicht versehentlich verschoben werden, wurde eine Eagle-Library für das OMAP-L138 SOM-M1 erstellt, welche alle drei Stecker mit den in Abbildung 2.14 dargestellten Abständen enthält.

2.6.5 Footprint JTAG Debugger

Abbildung 2.15 zeigt die Abmessungen des Tag-Connect Steckers *TC2050-IDC-NL*. Diese wurden beim Layout eingehalten. Außerdem wurde darauf geachtet, dass keine Bauteile zu nah an dem Stecker platziert wurden. Die Kontakte befinden sich aus Platzgründen auf der Rückseite des Boards. Im Vergleich zu anderen Steckern, zeichnet sich das Footprint von Tag-Connect durch seine geringen Abmessungen aus.

Außerdem wurde beim Layout des Steckers darauf geachtet, dass die Layoutvorgaben von Tag-Connect, welche in Abbildung 2.16 dargestellt sind, eingehalten werden.

2.7 Ausblick

Es lies sich bei Tests nicht verhindern, dass ein neues Booten des Linux-Systems dazu führt, dass ebenfalls das Echtzeitsystem neu bootet. Das bedeutet, dass ein Abstürzen des Linux-Systems

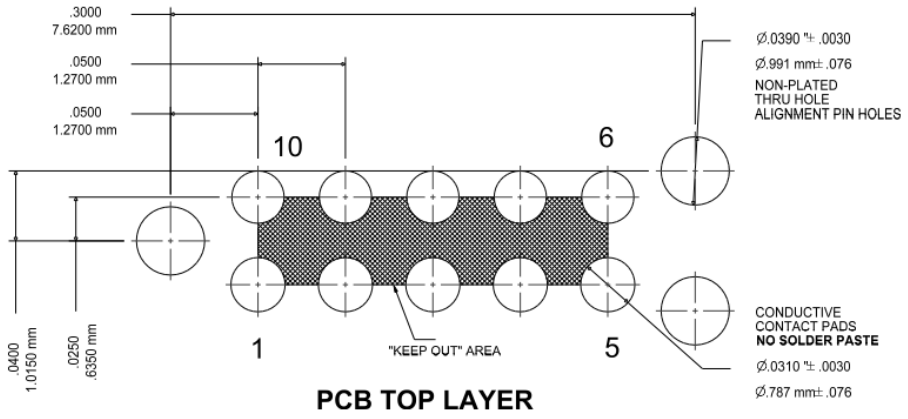


Abbildung 2.15: Footprint des Tag-Connect Debug-Steckers [12]

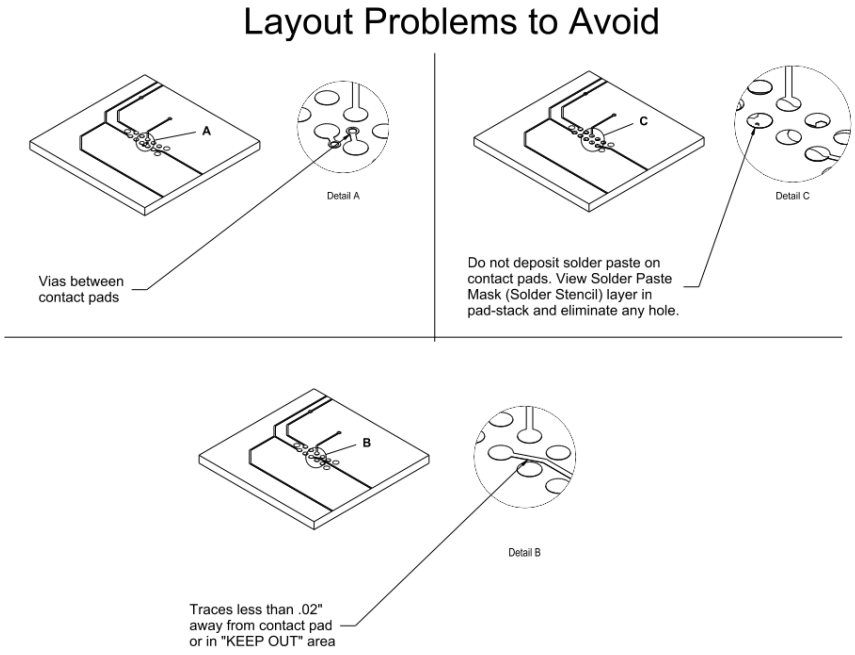


Abbildung 2.16: Layoutvorlagen von Tag-Connect [12]

dazu führen würde, dass die zeitkritischen Anwendungen des Echtzeitsystems ebenfalls abstürzen würden. Es ist jedoch aus Sicherheitsgründen nicht akzeptabel, dass das Echtzeitsystem durch einen Fehler im Linuxsystem abstürzt. Das würde zu einem Absturz des acspuav führen.

Da es nicht möglich war, die Linux-Anwendung komplett von der DSP-Anwendung zu trennen, wurde entschieden alle zeitkritischen Algorithmen der Flugregelung auf die Motortreiber auszulagern. Hier steht ein 32-Bit-Controller zur Verfügung, dessen Resourcen ausreichen um jene Algorithmen zu ausführen zu können.

KAPITEL 3

Simulation

Ich ermahne Dich, dass Du auf der mittleren Bahn fliegst, damit nicht die Welle Deine Federn beschwert, wenn Du zu tief fliegst und nicht das Feuer sie versenkt, wenn Du zu hoch fliegst. Fliege zwischen beiden!

Deadalus warnt seinen Sohn Ikarus (griechische Mythologie)[1]

[1] Einer Übersetzung aus dem Lateinischem, unbekannten Ursprungs, der griechischen Sage «Deadalus und Ikarus» aus der Schriftenreihe «Metamorphoses» des römsichen Dichters Ovid entnommen.

Inhalt: Um Flugalgorithmen testen und vergleichen zu können wurde eine GUI-basierte Simulation in C++ geschrieben. Als Grundlage wurde die Open Source Software-Framework Gazebo verwendet. Hierfür wird zunächst das dynamische Modell hergeleitet. Anschließend werden Messungen vorgestellt, welche für die Herleitung von Konstanten benötigt werden, die in den Gleichungen zum Schubverhalten von Rotoren bedeutend sind. Am Ende wird die Grundstruktur der Simulation beschrieben. Außerdem wird der Inhalt der entwickelten Klassen der Simulation, sowie deren Zusammenspiel geschildert.

3.1 Einleitung

3.1.1 Ziel der Arbeit

Bei der Entwicklung von Flugalgorithmen für den autonomen Flug eines Quadrocopters ist eine Simulation unvermeidbar. Es kann der Code für riskante Flugmanöver getestet und solange ausgereift werden bis dessen Grundfunktionen nachgewiesen werden können. Außerdem besteht die

Möglichkeit verschiedene Verfahren für Flugregelung, Sensorfusion und andere Algorithmen verglichen werden. Damit gelingt es die Algorithmen durch einen gezielten Stimulus zu beeinflussen. Die erhobenen Daten können aufgenommen werden und im Anschluss direkt verglichen werden. Ausgehend vom zu testenden Algorithmus kann ausgewählt werden, ob ideale Werte oder verrauschte Sensordaten verwendet werden. Eine Simulation stellt zwar keinen Ersatz von Praxistests dar, jedoch kann der Entwicklungszyklus für die Firmware des UAVs erheblich verkürzt werden.

3.1.2 Stand der Technik

Um einen Quadrocopter simulieren zu können, muss für jeden Simulationsschritt die Lage und Position im Raum aus den am Rahmen anliegenden Kräften und Momenten berechnet werden. In vielen Arbeiten wird hierfür das dynamische Model des Quadrocopters hergeleitet. Meist werden die Bewegungsleichungen mit der Newton-Euler Methode hergeleitet. [80] zieht neben der Newton-Euler Methode den Euler-Lagrange Formalismus heran um die Bewegungsgleichungen aufzustellen.

Die Kräfte auf den Rahmen resultieren aus den von den Rotoren erzeugtem Schub. Das Yaw-Moment resultiert aus den Reaktionsmomenten die durch Lagerreibung und vor allem durch Luftwiderstand der Rotoren (Drag) entstehen. Eine Methode diese mechanischen Einflüsse aus den Drehzahlen der Rotoren zu gewinnen ist die Impuls Theorie (Momentum Theory) wie sie beispielsweise in [82] beschrieben ist. Der einfachste Ansatz ist dabei den Schwebezustand anzunehmen und die dafür benötigten Konstanten über Schubmessungen eines fest eingespannten Motors zu gewinnen [74], [98].

Andere Simulationsansätze berücksichtigen zur Schubberechnung einen Ansatz nach [103], welcher durch die Verbindung von Impuls-Theorie und Blattelement Theorie eine Berücksichtigung der Relativgeschwindigkeit des Quadrocopters in vertikaler Richtung zulässt [74]. [68] nimmt den in [103] genannten Ansatz auf und erweitert ihn so, dass er neben den Effekten von vertikaler Geschwindigkeit auch den Einfluss von Vorwärtsflug berücksichtigt. Auch [31] behandelt aerodynamische Effekte im Vorwärtsflug. In [103] wird der Ground- und Deckeneffekt untersucht und in [67] wird auch der Flapping-Effekt von Rotoren behandelt. Die Theorie überlappender Motoren wird in [57] beschrieben. Wind wird in [43] und [126] behandelt und ein Controller zum Ausgleich des Windes vorgestellt. Auch [75] und [120] beschäftigt sich mit dem Ausgleich von Positionsdrift, hervorgerufen durch Windturbulenzen.

In [33] wird ein PID und ein LQ-Regler zur Regelung der Fluglage vorgeschlagen und verglichen. In [93] werden neben PID und LQ-Regler zusätzlich ein Fuzzy Regler vorgestellt und verglichen. Eine Regelung basierend auf Quaternionen wird in [123] vorgestellt. In [121] werden ebenfalls auf Quaternionen basierende Algorithmen vorgestellt, welche Coriolis- und gyroskopische Momente kompensiert.

In [93] wird ein mehrstufiges Kalman Filter System vorgeschlagen, welches durch die Werte von Magnetfeld und IMU die Lage des Quadrocopters stützt. Die Zustandsmatrizen dieses Filters werden bei Verfügbarkeit von GPS-Daten online erweitert und es werden nun auch Position und Geschwindigkeit mitgeschätzt. Ein Filter für reine Lagestützung wird in [90] vorgestellt. Der Ansatz zur Lagestützung von [96] schätzt mit einem Ansatz basierend auf einem extended Kalman Filter (EKF) die Geschwindigkeit mit.

Eine kommerzielle Simulationsumgebung ist RotorLib [109]. Diese ist kostenpflichtig. [53] wählt einen sehr schlichten Ansatz über Matlab. Da eine grafische Rückmeldung der Flugbewegung des UAVs ratsam ist, werden in [54] die Positionswerte über Simulink berechnet und zur grafischen Anzeige in den 3D-Flugsimulator FlightGear eingespeist. Die 3D-Darstellung von FlightGear verbraucht jedoch sehr viel Ressourcen. Deshalb müssen hier zwei Rechner verwendet werden. Ein anderer Ansatz ist [55], [72], welcher ebenfalls FlightGear verwendet. Hier wird jedoch die von FlightGear verwendete Aerodynamik-Simulation JSBsim verwendet. JSBsim ist dabei eine eigenständige und quelloffene C++ API, welche getrennt von FlightGear betrieben werden kann [48]. Simulationsumgebungen im Überblick:

- **FlightGear:** Open Source Flugsimulator mit der Möglichkeit Plugins zur programmieren. Wahlweise werden JSBSIM oder YAsim als Aerodynamik API genutzt.
- **YAsim, JSBsim:** Quelloffene Aerodynamik API.
- **Rotorlib:** Kommerzielle Simulationssoftware.
- **USARSim:** Quelloffene Simulationsumgebung, basierend auf der Unreal Tournament Game Engine.
- **Gazebo:** Quelloffene Simulationsumgebung mit vielen Features.

3.1.3 Rahmenbedingungen

Anforderungen

Im Folgenden, werden die wichtigsten Anforderungen an die Simulationsumgebung [1] zusammengetragen:

- Berechnung von Fluglage und Position im Raum aus den Kräften und Momenten auf den Rahmen
- Simulation von Sensoren (IMU, Magnetfeld, GPS)
- Berechnung der aerodynamischen Kräfte und Momente aus den Motordrehzahlen (Impulstheorie)
- Berechnung der Kräfte und Momente resultierend aus den Massen und Trägheitsmomenten des Körpers (Trägheit, gyroskopischer Effekt)
- Grafische Darstellung
- Möglichkeit Simulationsdaten in Dateien zu speichern
- Modularer und zugänglich dokumentierter Code um Erweiterungen möglich zu machen

Das Ablaufdiagram aus Abbildung 3.1 zeigt dabei einen Teil der Anforderungen.

Links (grün hinterlegt), befindet sich der simulierte Code, rechts (rot hinterlegt) der Ablauf, der Simulation selbst.

[1] Die hier entwickelte Simulationssoftware für Quadrocopter wurde *DeadalusSim* genannt.

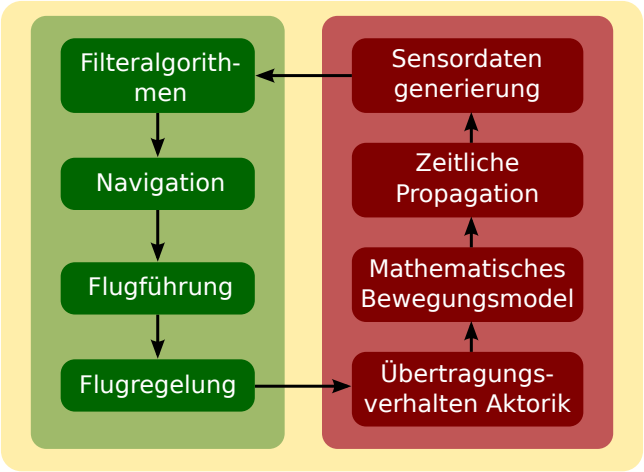


Abbildung 3.1: Simulations Framework [74]

Vereinfachungen

Folgende Vereinfachungen werden angenommen:

- Der Rahmen wird als starr angenommen.
- Der Rahmen ist symmetrisch.
- Der geometrische Mittelpunkt des Rahmens und Schwerpunkt fallen zusammen.
- Die Rotoren werden als starr angenommen.
- Sowohl Ground- und Deckeneffekte als auch Auftriebseffekte durch Vorwärts- oder Vertikalflug werden zunächst vernachlässigt.

Wahl der Simulationsumgebung

Da die 3D-Darstellung von Gazebo ressourcenschonend ist und viele Features, wie beispielsweise die Simulation von Sensoren bietet, fiel die Wahl auf diese Simulationsumgebung. Es wurde ein vergleichbarer Ansatz wie er in [74] und [98] vorgestellt wurde, gewählt.

3.1.4 Gazebo

Gazebo ist eine rundenbasierende Open Source Robotik-Simulationsumgebung mit Visualisierung. Sie berechnet zyklisch aus den Kräften und Momenten, welche an einem starren Körper anliegen, dessen Lage und Position im Raum. Dafür zieht sie eine der unterstützten Physik-Engines heran. Sie bietet dabei folgende Features:

- Unterstützung von derzeit vier Physik-Engines (ODE, Bullet, Simbody, Dart)
- Kollisionsmodell

- Simulation von Sensoren (Beschleunigungssensor, Kameras, GPS), konfigurierbar mit Rauschen
- Räumliche Darstellung der Gazebo-World durch sogenannte Digital Elevation Models (DEM), welche online verfügbar sind [70]
- Ressourcenschonende grafische Darstellung, falls der Rechenaufwand noch vertretbar ist auch in Echtzeit

Gazebo kann modular durch programmierte Plugins, programmiert in C++, in einer CMake Umgebung erweitert werden. Dabei sind vier Plugin-Arten möglich (World, Model, Sensor, System). Die API von Gazebo ist dabei gut dokumentiert. Modelle, die in die Gazebo-World geladen werden, werden in XML-Files, im von Gazebosim vorgegebenem SDF-Format, definiert. Hier werden dessen Eigenschaften, wie beispielsweise Trägheitsmomente, Gewicht, visuelle Darstellung, Abmessungen des Kollisionsmodells und vieles mehr festgelegt. Auch das SDF-Format ist gut auf der Website von Gazebosim dokumentiert.

Die Gazebo Kommunikation funktioniert mit TCP/IP Sockets. Messages werden auf Kanälen, sogenannten Topics via Publisher versandt. Empfangen werden sie durch einen sogenannten Subscriber. Der Code hierfür wird durch Überschreiben der jeweiligen Funktionen der Basisklassen in die Gazebo-Plugins eingefügt.

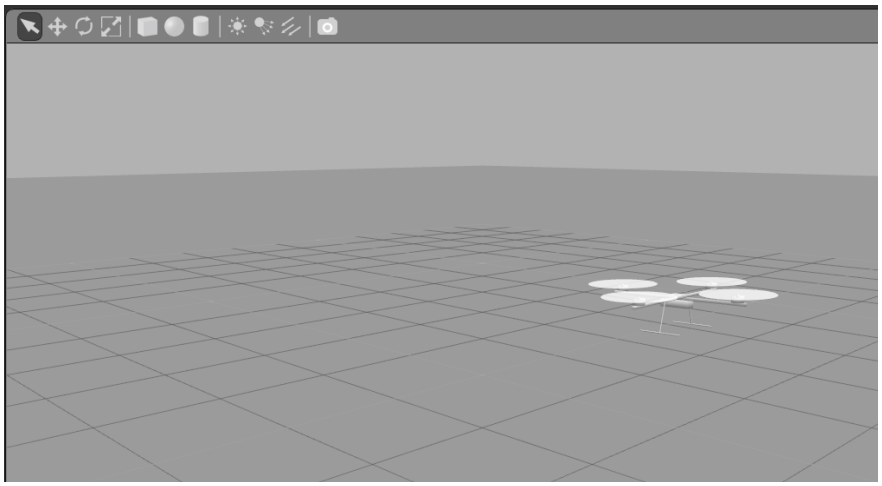


Abbildung 3.2: GUI von DeadalusSim

3.2 Quadrocopter Modell

3.2.1 Kinematik

Abbildung 3.3 zeigt die verwendeten Koordinatensysteme. Das Inertialkoordinatensystem (N, E, D) ist fest mit der Erde verbunden. Dessen Achsen zeigen in Richtung der Himmelsrichtungen ((N)orth, (E)east) und des Erdursprungs ((D)own). Das körperfeste Koordinatensystem (x, y, z) ist fest mit dem Rahmen des UAVs verbunden. Die Vorwärtsflugrichtung stimmt dabei mit der x-Achse überein.

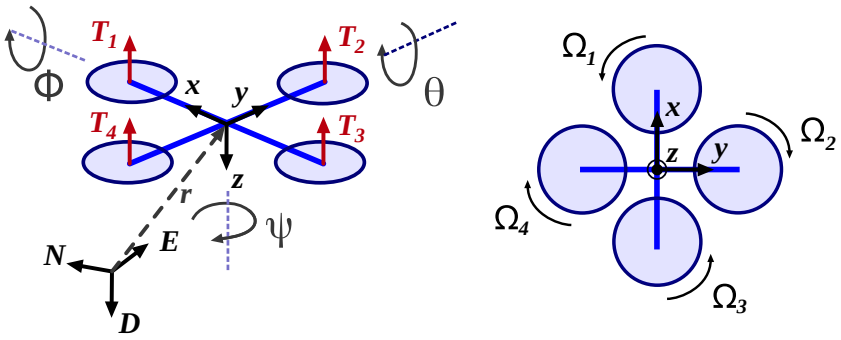


Abbildung 3.3: Koordinatensysteme und Drehrichtung der Rotoren

Für die Umrechnung der Orientierung des körperfesten Koordinatensystems (x, y, z) zum Inertialkoordinatensystem (N, E, D) wird die Richtungskosinusmatrix [74], [81], [93] verwendet:

$$R = \begin{bmatrix} c\theta c\psi & s\phi s\theta c\psi - c\phi s\psi & c\phi s\theta c\psi + s\phi s\psi \\ c\theta s\psi & s\phi s\theta s\psi + c\phi c\psi & c\phi s\theta s\psi - s\phi c\psi \\ -s\theta & s\phi c\theta & c\phi c\theta \end{bmatrix}$$

Die Position des UAVs im Inertialkoordinatensystem wird durch den Vektor r dargestellt:

$$r = [N\ E\ D]^T \tag{3.1}$$

3.2.2 Dynamik

Aerodynamik

Entsprechend der Impulstheorie (Momentum Theory, MT) gelten im Schwebeflug folgende Zusammenhänge für den Schub T_i und das vom Rotor verursachte Lastmoment Q_i [82]:

$$T_i = \underbrace{C_T \rho A R^2}_{k_T} \Omega_i^2, \quad M_i = \underbrace{C_M \rho A R^3}_{k_M} \Omega_i^2 \tag{3.2}$$

Dabei ist ρ die Dichte der Luft, A die vom Rotor überstrichene Fläche und R ist der Radius des Rotors. Kombiniert man die Impulstheorie und die Blattelement Theorie, lässt sich ein Zusammenhang für den Schub herleiten, der eine Berücksichtigung der Relativgeschwindigkeit zulässt [74].

$$T = C_{T,0}\Omega^2 + C_{T,1} v_1 \Omega^2 + C_{T,2} v_1^2 \tag{3.3}$$

$(v_1)_i$ ist dabei die Geschwindigkeit in Richtung des Kraftvektors des Motors i. Unter Einbezug der Winkelgeschwindigkeit des Rahmens ω^b lässt sie sich nach Gleichung (3.4) berechnen. l_M ist dabei der Abstand der Motor-Achse zum Zentrum des Rahmens [74].

$$\begin{aligned} (v_1)_1 &= v_z + \omega_y^b \cdot l_M, \quad & (v_1)_3 &= v_z - \omega_y^b \cdot l_M \\ (v_1)_2 &= v_z - \omega_x^b \cdot l_M, \quad & (v_1)_4 &= v_z + \omega_x^b \cdot l_M \end{aligned} \tag{3.4}$$

Kräfte und Momente auf den Rahmen

Durch die Gleichungen (3.2) können die Kräfte und Momente auf den Rahmen berechnet werden. Diese werden durch den Schub und das Lastmoment der Motoren, hervorgerufen durch Lagerreibung und Luftwiderstand, berechnet.

$$F_M^b = \begin{bmatrix} 0 \\ 0 \\ k_T(\Omega_1^2 + \Omega_2^2 + \Omega_3^2 + \Omega_4^2) \end{bmatrix} \qquad M_M^b = \begin{bmatrix} k_T l_M(\Omega_4^2 - \Omega_2^2) \\ k_T l_M(\Omega_1^2 - \Omega_3^2) \\ k_M(\Omega_2^2 + \Omega_4^2 - \Omega_1^2 - \Omega_3^2) \end{bmatrix} \tag{3.5}$$

Außerdem resultiert aus dem gyroskopischen Effekt, hervorgerufen durch die Drehzahl der Motoren und den durch sie zu überwindendnen Trägheitsmomenten J_r, das Moment M_G^b:

$$M_G^b = \begin{bmatrix} J_r \omega_y(\Omega_1 + \Omega_3 - \Omega_2 - \Omega_4) \\ J_r \omega_x(\Omega_2 + \Omega_4 - \Omega_1 - \Omega_3) \\ 0 \end{bmatrix} \tag{3.6}$$

Bewegungsgleichungen nach der Newton-Euler Methode

Durch die in Kapitel *Kräfte und Momente auf den Rahmen* aufgestellten Gleichungen, lassen sich unter Verwendung der Newton-Euler Methode die Bewegungsgleichungen für Translation [2] und Rotation aufstellen:

$$\begin{aligned} J\dot{\omega^b} + \omega \times J\omega &= M_M^b - M_G^b \\ m\ddot{r} &= [0\ 0\ mg]^T - RF_M^b \end{aligned} \tag{3.7}$$

Motor Dynamik

Es hat sich herausgestellt, dass es zweckmäßig ist, das Verhalten des Motors als Differentialgleichung erster Ordnung darzustellen. Dafür wird eine Gleichung verwendet, wie sie ebenfalls in [35] hergeleitet wurde. Dabei ist u die Stellgröße des Motors.

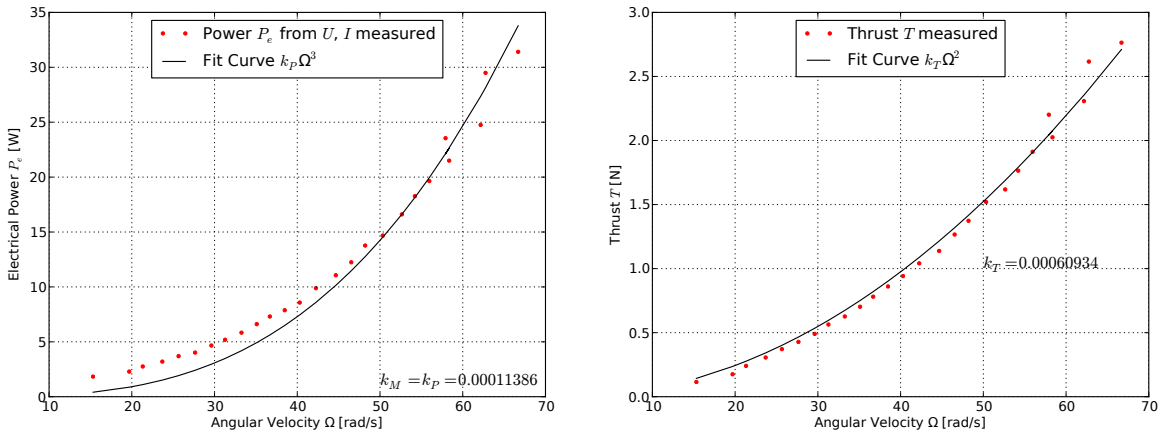
$$K \cdot u + C = T \cdot \dot{\omega} + \omega \tag{3.8}$$

3.2.3 Identifikation der Systemkonstanten aus Messungen

Um die in Kapitel *Dynamik* vorgestellten Gleichungen in der Simulation verwenden zu können, müssen die darin enthaltenen Konstanten aus Messungen ermittelt werden. Die Konstanten k_M und k_T aus den Gleichungen (3.2) wurden durch die in Abbildung 3.4 dargestellten Messungen ermittelt. Zunächst wurde die Simulation auf einen Quadrocopter mit 440 mm Achsabstand und Robbe Roxxy BL2827-34 Motoren mit 10 Zoll Motoren ausgelegt.

Da entsprechend Gleichung (3.2) eine quadratische Funktion zu erwarten ist, wurde die Konstante des Schubbeiwerts k_T in Abbildung 3.4 (b) durch die Least-Square-Fit Methode angenähert. Aus

[2] May the Mass times Acceleration be with you.



(a) (b)

Abbildung 3.4: Messungen der Leistung und des Schubs über der Drehzahl für Robbe Roxxy BL2827-34

[111] ist bekannt, dass der Leistungsbeiwert k_P und der Momentbeiwert k_M identisch sind. Da für das Moment des Motors keine Messungen vorliegen, musste der Momentbeiwert in Abbildung 3.4 (a) über den Leistungsbeiwert hergeleitet werden. Dabei wurden Messungen der elektrischen Eingangsleistung des Motortreibers verwendet. Hier muss darauf geachtet werden, dass für die tatsächlich abgegebene Leistung der Wirkungsgrad berücksichtigt werden muss. Dieser wird hier auf etwa 80 Prozent geschätzt.

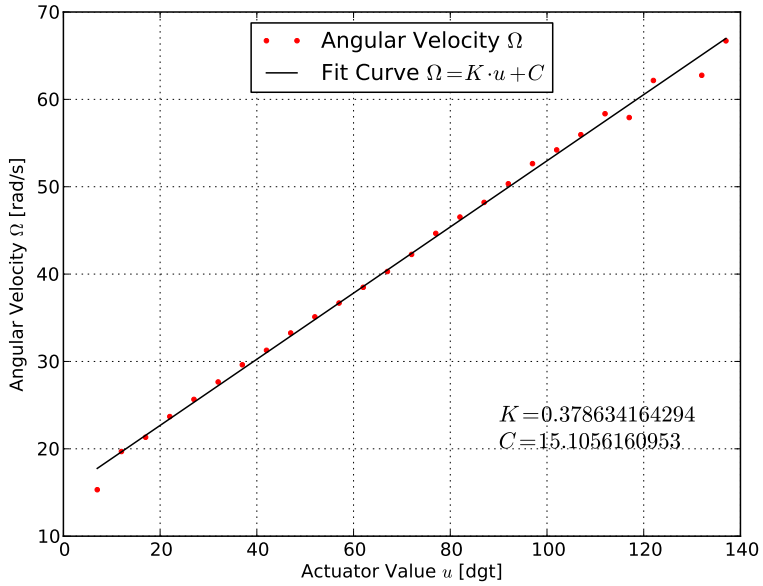


Abbildung 3.5: Systemkonstanten der Motordynamik im stationären Fall

In Gleichung (3.8) wird das dynamische Verhalten des Motors vorgestellt. Da für das zeitliche Verhalten derzeit keine Messungen vorliegen wurde zunächst nur das stationäre Verhalten identifiziert.

In Abbildung 3.5 werden die Konstanten K und C ermittelt. In der Simulation wird die fehlende Größe T zunächst auf Null gesetzt.

Die hier identifizierten Werte können jederzeit analog für andere Ausleger/Motor/Rotor-Kombinationen berechnet werden.

3.3 Simulation

3.3.1 Aufbau

Abbildung 3.6 zeigt den Aufbau der Simulation. Gazebo übernimmt dabei die Lösung der Bewegungsgleichungen aus den Kräften und Momenten zur Ermittlung der Fluglage und Position. Dafür wird von Gazebo eine Physik-Engine verwendet. Auch die 3D-Darstellung wird von Gazebosim übernommen. Da Gazebosim jedoch keine Aerodynamik kennt, werden die in Kapitel *Dynamik* vorgestellten Gleichungen für die Berechnung der Schubkräfte und der daraus resultierenden Momente auf den Rahmen verwendet. Wie bereits in Kapitel *Identifikation der Systemkonstanten aus Messungen* erwähnt, wird aufgrund von fehlenden Messungen, zunächst vereinfachend nur der Schwebeflug bei der Schubberechnung berücksichtigt.

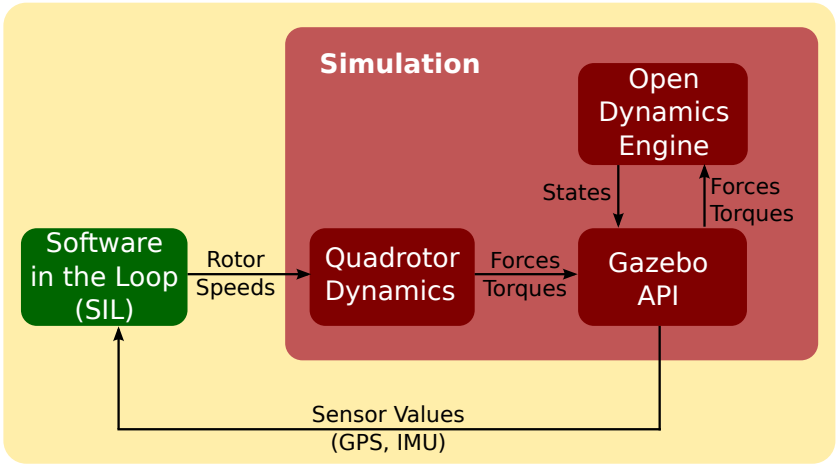


Abbildung 3.6: Simulations Framework [74]

3.3.2 Gazebo-Plugin Klassen

Es wurden zwei Plugins für Gazebo programmiert. Ein Model-Plugin und ein Sensor-Plugin:

- **GazeboModelPlugin:** Diese Klasse enthält eine Instanz von *SIL* (siehe Kapitel *Software in the Loop (SIL)*) und *QuadRotorDynamics* (Kapitel *Quadrocopter Dynamik Klassen*). Es werden die aus der Instanz von *SIL* gewonnenen Stellwerte an die Instanz von *QuadRotorDynamics* weitergegeben. Hier werden die Kräfte und Momente berechnet. Diese Kräfte und Momente werden durch Befehle der Gazebo API an den Quadrocopter-Rahmen angelegt. Außerdem werden die simulierten Sensor-Werte der IMU an die Instanz von *SIL* übergeben.

- **GazeboIMUPlugin:** Diese Klasse veröffentlicht die simulierten Sensor-Werte der IMU, damit auf diese in der Klasse *GazeboModelPlugin* zugegriffen werden kann.

Diese Klassen stellen die Schnittstelle zur Simulationsumgebung Gazebosim dar. Hierfür werden die Befehle der Gazebo API verwendet. Mehr Informationen können auf der Website von Gazebo gefunden werden [51].

3.3.3 Quadrocopter Dynamik Klassen

Da die API von Gazebo derzeit keine Aerodynamik Funktionalität anbietet, wurde die Klasse **QuadRotorDynamics** programmiert, um diese nachzubilden. Aufgabe dieser Klasse ist, durch die Gleichungen aus Kapitel *Dynamik*, aus den Stellwerten zunächst durch ein PT1-Verhalten die Drehzahlen zu berechnen. Aus den Drehzahlen werden schließlich durch die Formeln aus Gleichung (3.2) die Schubkräfte und das Yaw-Moment berechnet und aus den Formeln der Gleichungen (3.5) das Roll und Pitch Moment. Da sich die Momente des gyroskopischen Effekts im Schwebeflug auflösen und oft vernachlässigt werden, können diese wahlweise aktiviert werden.

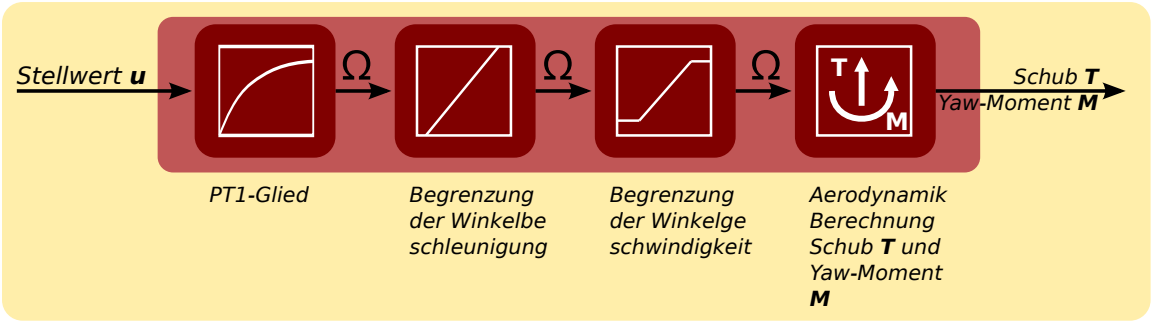


Abbildung 3.7: Ablauf der Berechnung des Dynamischen Verhaltens des Quadrocopters

Da in der Realität die Motordrehzahl und die Winkelbeschleunigung der Motoren begrenzt ist, können in *QuadRotorDynamics* die Grenzwerte für die Simulation eingestellt werden. Werden diese überschritten, werden sie in der Simulation auf die maximal möglichen Werte begrenzt. Der gesamte Ablauf der Klasse *QuadRotorDynamics* wird in Abbildung 3.7 gezeigt.

3.3.4 Software in the Loop (SIL)

Alle Funktionalitäten um die zu testende Firmware (Software in the Loop, SIL) des QuadroCopters in das Gazeboframework einzubinden befinden sich in der Klasse *SIL*. Es werden hier die simulierten Sensorwerte, die Simulationszeit und die realen Euler-Winkel zur Verfügung gestellt. Die berechneten Motor-Stellwerte werden von dieser Klasse an die Simulation zurückgeliefert. Außerdem besteht die Möglichkeit Eingaben eines PlayStation 3 [3] Controllers auszulesen. Es gibt auch Funktionen um gewünschte Stimuli an die Simulation zu liefern.

[3] Sony PlayStation 3 - DualShock 3 Wireless Controller

Die wichtigste Aufgabe dieser Klasse ist es, in jedem Simulationsschritt, die zu testende Firmware auszuführen. Zu diesem Zweck wird die Update-Funktion der Instanz dieser Klasse in *GazeboModelPlugin* ausgeführt.

3.4 Regelung

3.4.1 Backstepping-Regler

Um schnell Simulationsergebnisse erhalten zu können wurde ein Backstepping-Regler für die Lageregelung implementiert. Dabei wird der Regelausdruck, aus Gleichung (3.9) verwendet (hier für den Rollwinkel ϕ dargestellt). Für die Herleitung dieses Ausdrucks sei auf [93] verwiesen.

$$u = \frac{J_x}{l_M}((1 + a_1 \cdot a_2)(\phi_{set} - \phi) - (a_1 + a_2)\omega^b_{ib,x}) \tag{3.9}$$

ϕ_{set} stellt dabei den Stellwert des Anstellwinkels der Rollachse, ϕ, den tatsächlichen Wert des Rollwinkels, l_M den Abstand der Rotorachse zur Rahmenmitte, J_x das Massenträgheitsmoment um die x-Achse und $\omega^b_{ib,x}$ die x-Komponente der Werte des Drehratensensors. Über die Einstellungen der Parameter a_1 und $a2$ kann das Verhalten des Reglers beeinflusst werden.

Die Reglerparameter werden dabei zunächst empirisch gewählt und sind nicht optimiert. Die Winkel werden zunächst noch nicht durch Filteralgorithmen aus den Sensorwerten gewonnen. Statt dessen werden vorerst die realen Winkelwerte der Simulation verwendet.

3.4.2 Steuerung

Es besteht die Möglichkeit den Quadrocopter der Simulation über einen PlaySation 3 Controller zu steuern.

Abbildung 3.8: PlayStation 3 Controller

Der Controller kann drahtlos über Bluetooth genutzt werden. Hierfür wird der sogenannte *QtSixA*-Treiber [131] verwendet. Dies ist ein angepasster Joystick-Treiber für Linux-Systeme, der den Playstation 3 Controller unterstützt. Details zu Installation und Verwendung dieses Treibers finden sich im Anhang unter Kapitel *installqtsixa*.

3.4.3 Mixer

Mit dem Mixer wird festgelegt, in welcher Weise die Motor-Stellwerte auf die einzelnen Motoren verteilt werden. In diesem Fall wurde die T-Flug-Konfiguration aus Gleichung (3.10) gewählt.

$$\begin{aligned} u_1 &= ps3_{gas} + y_{yaw} + y_{pitch} \\ u_2 &= ps3_{gas} + y_{yaw} - y_{pitch} \\ u_3 &= ps3_{gas} - y_{yaw} + y_{roll} \\ u_4 &= ps3_{gas} - y_{yaw} - y_{roll} \end{aligned} \tag{3.10}$$

3.5 Ausblick

Mit DeadalusSim wurde hier eine Simulationsumgebung vorgestellt mit der man bereits das Flugverhalten eines Quadrocopters untersuchen kann. Diese kann nun stetig weiterentwickelt werden.

Einige der Arbeiten welche noch geplant sind:

- *Sensor Plugin für Magnetfeldsensor schreiben:* Da Gazebo derzeit noch keine Magnetfeldsensor nativ angeboten wird, muss hier noch Code geschrieben werden. Am einfachsten ist es, eine Klasse zu schreiben, welche die Werte eines fixen 3D-Vektors im körperfesten Koordinatensystem zunächst in den Ursprung des Ineritalkoordinatensystems des simulierten UAVs verschiebt und ihn dann durch Quaternionenrotationen in jenes Inertialkoordinatensystem umrechnet.
- *Erweiterung um GPS Sensor:* Gazebo bietet die Klassen für die Simulation eines Global Positioning System (GPS)-Sensors. Diese müssen noch implementiert werden.
- *Verbesserung des Aerodynamik Modells:* Das Aerodynamik-Modell kann durch die Berücksichtigung des Ground-Effekts verbessert werden. Desweiteren kann der Einfluss von Vorwärtsflug und vertikaler Geschwindigkeit auf das Flugverhalten einbezogen werden. Hierfür sind jedoch Windkanalmessungen notwendig.
- *Schätzung des Flugverhaltens durch ein neuronales Netz:* Anstatt die aerodynamischen Werte von den Rotoren durch Messungen zu ermitteln, könnte man die nichtlinearen Zusammenhänge des Flugverhaltens des Quadrocopters, durch Anlernen eines neuronalen Netzes, mit Messungen aus realen Flügen erhalten. Hier könnte ein ähnlicher Ansatz wie in [41] gefunden werden.

KAPITEL 4

Sensordatenfusion

> The Kalman Filter in its various forms is clearly established as a fundamental tool for analyzing and solving a broad class of estimation problems.
>
> ***Leonhard McGee and Stanley Schmidt, Ames Research Center, NASA***

Inhalt: Es werden Algorithmen hergeleitet um aus inertialer Sensorik (INS) und globalen Navigationssystemen (GNSS) Lage- und Positionsinformationen zu berechnen. Hierfür wird auf Komplementär- und Kalman-Filter zurückgegriffen. Zunächst werden theoretische Grundlagen beschrieben und anschließend die Algorithmen selbst vorgestellt.

4.1 Einleitung

4.1.1 Ziel der Arbeit

Für **Fluglageregelung**, **Positionsregelung** sowie **Höhenregelung** des Flugroboters werden stabile und genaue Werte unterschiedlicher Größen wie **Fluglage**, **Flughöhe**, **Geschwindigkeit**, oder **Position** des Flugroboters benötigt.

Diese Größen können nicht immer direkt durch Sensormessungen in ausreichender Genauigkeit erhalten werden. Ziel der Arbeit ist es durch Sensordatenfusion aus fehlerbehafteten Messwerten verschiedener Sensoren präzise Werte jener benötigten Größen zu schätzen. Dabei werden Werte des globalen Navigationssatellitensystems (GNSS) sowie inertiale Größen wie Beschleunigung, Drehraten und weitere Größen wie barometrischer Höhendruck oder Magnetfeld herangezogen.

Unter Sensordatenfusion versteht man die Kombination von Sensordaten mit dem Ziel die Quali-

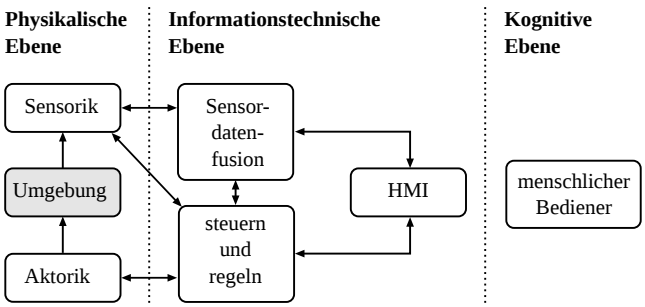


Abbildung 4.1: Rolle der Sensordatenfusion im Gesamtsystem [95]

tät der erhaltenen Information im Vergleich zu Einzelmessungen zu verbessern. Dabei müssen die verwendeten Sensoren nicht zwingend die selben Größen liefern und auch keiner gemeinsamen Zeitbasis unterliegen. Es reicht, wenn Größen gemessen werden, welche in systematischem Zusammenhang stehen. Durch dieses Konzept können aus einer Fülle von mannigfaltigen Messungen eines Systems, Größen abgeleitet werden, die nicht direkt gemessen werden. Auch können, durch die Synergie von mehreren Sensorwerten und Systemverständnis, Messfehler und Ausreißer reduziert, sowie das Wissen über die Plausiblität der Größen verbessert werden. Die Rolle der Sensordatenfusion im Gesamtsystem wird in Abbildung 4.1 dargestellt. Zur genaueren Lektüre von Multi-Sensordatenfusion und Sensorvernetzung können [26] und [95] herangezogen werden.

Die Sensordatenfusion bedient sich vieler Werkzeuge unterschiedlicher Disziplinen, wie beispielsweise neuronaler Netze, Fuzzy-Logic, bayesscher Filter oder lernender Systeme. In diesem Rahmen kommen vor allem Konzepte wie Komplementär- oder Kalman-Filterung zur Anwendung.

4.1.2 Stand der Technik

Kalman-Filter

Das Kalman-Filter [78] ist bereits seit den 1960er Jahren bekannt und daher in zahlreicher Literatur gut dokumentiert. Ein Klassiker, ebenfalls aus den sechziger Jahren, ist die Arbeit [102], dessen Titel eine humoristische Anlehnung an Stanley Cubrick's satirischen Film über den Kalten Krieg und die nukleare Abschreckung - *Dr. Strangelove* - ist. Andere intuitive und anwendungsbezogene Einführungen zum Thema Kalman-Filter finden sich in [117] und [46]. Eine Heranführung an die Grundidee der Herleitung des Kalman-Filters wird in [47] geliefert. Für die Lektüre einer vollständigen Herleitung des Filters eignet sich [122] besonders gut. Auch [85] geht auf die Herleitung des Kalman-Filters ein und liefert eine umfassende Diskussion der Filteralgorithmen für Ingenieure. Für ein Grundverständnis der Anwendung des Kalman-Filters kann jedoch zunächst die Herleitung übersprungen werden und es genügt eine pragmatische Herangehensweise an das Thema wie sie in [117] und [46] vorliegt. Vertiefungen bezüglich numerischer Optimierung und Anwendung der Filteralgorithmen finden sich in [52], [118] und [47]. Hier werden anwendungsbezogene Themen, wie numerische Integrationsmethoden oder numerische Vereinfachungen der Kalman-Filteralgorithmen mit Berücksichtigung von Rechenaufwand, Speicherauslastung und numerischer Stabilität untersucht. Ein Studium dieser Werke lohnt sich daher sehr, vor allem wenn das Filter auf eingebetteten Systemen mit beschränkten Ressourcen Anwendung findet.

Bereits in den sechziger Jahren wurde mit dem extended Kalman-Filter (EKF) eine Lösung auch für nichtlineare Systeme gefunden, welches ebenfalls sehr anschaulich in [122] beschrieben wird. Die Grundidee ist die kontinuierliche Linearisierung um den aktuellen Punkt mithilfe einer Taylorreihe. Mitte der neunziger Jahre wurde mit dem Unscented Kalman-Filter (UKF) ein weiteres abgewandeltes Kalman-Filter für nichtlineare Systeme vorgestellt. Auch dieses findet sich in [122]

Auch für Sensornetzwerke gibt es Ansätze für Kalman-Filter. Wenn jeder Knoten in einem Sensornetzwerk einen lokalen Kalman-Filter ausführt und die jeweiligen Schätzungen von Knoten zu Knoten zur Weiterverarbeitung weitergereicht werden, spricht man vom sogenannten Decentralized Kalman-Filter [26]. Da diese Art von Kalman-Filter eine Kommunikation von allen Knoten mit allen anderen Knoten erfordert, ist sie nicht förderlich, wenn die Knoten batteriebetrieben sind, da eine drahtlose Kommunikation Leistung kostet [26]. Für diesen Zweck ist der sogenannte Distributed Kalman-Filter (DKF) besser geeignet, da dieser nur eine Kommunikation von Knoten zu Knoten benötigt. [26]

Fluglageschätzung

Eine inertiale Messeinheit (*Inertial Measurement Unit*, acsimu) besteht aus Drehraten- und Beschleunigungssensoren. Mit ihnen kann Rotation und Translation eines Körpers gemessen werden. Ein MARG (*Magnetic, Angular Rate, Gravity*) ist eine IMU, die zusätzlich einen Magnetfeldsensor beinhaltet. Damit kann die Orientierung zum Magnetfeld der Erde, sowie zum Erdschwerevektor bestimmt werden. Ein solches System zur Schätzung der Orientierung bezeichnet man als AHRS (*Attitude Heading Reference System*). Die wohl bekanntesten Ansätze sind jene von Madgwick [89], [88] und Mahony [90]. Beide Filter verwenden die Quaternionendarstellung und leiden daher nicht unter Singularitäten, wie sie unter bestimmten Winkeln bei der Euler-Winkeldarstellung auftreteten. Diese Algorithmen zeichnen sich vor allem durch ihre Performanz, trotz verhältnismäßig wenigen Floating-Point-Berechnungen, aus. Sie können wahlweise mit einer IMU oder einem MARG-System betrieben werden und wurden in Bezug auf Floating-Point-Berechnungen zulasten des Speicherplatzes in C optimiert. Unter [132] werden die genannte C-Codes, neben weiteren Formulierungen in C# und Matlab, zum Download angeboten. Da sie auch bei niedrigen Samplingraten noch gute Schätzungen liefern, sind sie auch ohne weiteres auf schwachen 8-Bit Controllern noch lauffähig, wenn man nicht an die Grenzen des Speicherplatzes stößt.

Sehr häufig werden Ansätze zur Schätzung der Orientierung mit Kalman-Filtern formuliert. In [91] wird ein auf Quaternionen basierender Ansatz vorgestellt. Der Zustandsvektor setzt sich aus den Komponenten der Winkelgeschwindigkeit und dem Quaternion zur Darstellung der Orientierung zusammen. Es werden zwei alternative Messvektoren vorgestellt. Zunächst besteht der Messvektor aus den neun Messungen des MARG-Systems. Da diese Darstellung zu vielen Matrixberechnungen führt wird ein alternativer Messvektor vorgestellt. Dieser besteht aus den Komponenten der Winkelgeschwindigkeit und dem Quaternion welches optimal die Messungen von Magnetfeld und Beschleunigung in den Folgezustand überführt. Dieses Quaternion wird mithilfe des Gauss-Newton-Verfahrens ermittelt. Dies führt dazu, dass sich der Messvektor von neun Elementen auf sieben reduziert.

In [93] wird ein sogenannter Error Space State Kalman-Filter zur Fluglageschätzung formuliert. Hier werden im Zustandsvektor nicht die absoluten Werte, sondern die Fehler geschätzt. Im Speziellen werden die Fluglagewinkel und der Biasdrift des Drehratensensors geschätzt. In [115] werden

kinematische Formeln für das Systemmodell verwendet. Auch hier wird der Drift des Drehratensensors mitgeschätzt.

Einfachere und damit weniger rechenintensive Algorithmen sind die in [113] und [101] vorgestellten Komplementärfilter. Diese Algorithmen sind jedoch nicht so robust gegen Fehler durch Beschleunigungen, die den Erdvektor verfälschen.

In [25] werden die Algorithmen von Magwick [89], [88] und Mahony [90] mit einem Ansatz mit extended Kalman-Filter experimentell verglichen. Es wird auch auf Genauigkeit und Laufzeit der Algorithmen eingegangen. In [28] wird der nichtlineare Komplementärfilter von Mahony [90] und der auf einer Optimierungsrechnung mithilfe des gradient descent Algorithmus basierende Filter von Madgwick [89], [88] verglichen. Für diesen Vergleich werden eine Simulation, sowie Messungen von realen Sensorwerten herangezogen. Es werden die Fehler der jeweiligen Algorithmen miteinander verglichen. Außerdem wird die Performanz der jeweiligen Filter bei unterschiedlichen Filterparametern verglichen.

Positionsschätzung

Am Fraunhofer-Institut in Stuttgart wurden 2004 in [64] Kalman-Filteralgorithmen zur rein inertialen Positionsschätzung für Indoor-Navigation, basierend auf Drehraten-, Beschleunigungs- und Magnetfeldsensoren, vorgestellt. Auch an der ETH-Zürich [42] sowie an der Universität von Pennsylvania [94] wurden Indoor-Navigationslösungen basierend auf einem kommerziellem Motion Capture System der Firma Vicon vorgestellt. Der Zweck dieser Lösung ist vor allem die Erforschung von Kontrollalgorithmen für Quadrocopter. In [134] wird ein Lokalisierungssystem, basierend auf Ultrabreitband-Funktechnik, untersucht, welches für Umgebungen ohne GPS-Empfang, wie beispielsweise in Gebäuden, eingesetzt werden soll.

Die Genauigkeit von Globalen Navigationssystemen (GNSS) allein reicht nicht aus um die Position des Quadrocotpers im Freien zu regeln. Deshalb wurden zahlreiche Lösungen vorgeschlagen, die Genauigkeit durch die Fusion von GNSS-Systemen und inertialer Messtechnik zu verbessern. Der Zustandsvektor des Error Space State Kalman-Filter aus [93] wird bei Verfügbarkeit von GPS zur Positionsschätzung erweitert. Dieser Filter stammt ursprünglich aus [129]. Dies ist die Fortsetzung der Forschungen an einem direkten Kalman-Filteransatz aus [128], von Jan Wendel, dem Autor von [129]. In [96] wird ein direkter Kalman-Filter zur Positionsschätzung, basierend auf Quaternionen und der Lösung des Strapdownalgorithmus, vorgestellt. In dieser Arbeit wird ein sehr ähnlicher Ansatz verwendet.

Eine weitere Möglichkeit die Genauigkeit von GNSS-Systemen zu verbessern ist differential GPS (**DGPS**) in Verbindung mit Realtime-Kinematics (RTK). Hier werden durch ein ortsfestes GPS-Modul, das seine Position kennt, sogenannte Pseudorange-Daten berechnet. Pseudorange-Daten beinhalten die Differenz der gemessenen Werte dieses Moduls zur bekannten Position, hervorgerufen durch den Ionosphären- und Stratosphärenfehler des GPS-Signals. Diese Daten können zur Korrektur des GPS-Signals verwendet werden. Außerdem wird das Rohsignal der Trägerphase des GPS-Signals ausgewertet, um die Genauigkeit zu verbessern (**RTK**).

Seit etwa einem Jahr ist ein kommerzielles GPS-Modul der Firma *Swift Navigation* lieferbar, welches **RTK**-Technologie bietet [119]. Im Gegensatz zu den industriellen Lösungen ist es leicht und preiswert. Die Module wurden in einem Kickstarter-Projekt entwickelt. Deren Software ist Open

Source und deren Preis beträgt um die tausend Euro. Dies ist zwar immer noch teuer, aber wesentlich billiger als industrielle Lösungen.

Navigation durch RTK in Verbindung mit DGPS wurde im Rahmen der Applied Reseach Conference in [60] getestet. Hier hat sich gezeigt, dass mit dieser Methode eine ausreichende Genauigkeit für die Positionsregelung eines UAVs erreicht werden kann, jedoch haben sich auch einige Nachteile herausgestellt. Es kann unter Umständen mehrere Sekunden dauern, bis ein positionsgenaues Signal erhalten wird. Diese Wartezeit kann sich auch bei Verlust des GPS-Signales wiederholen. Außerdem wurden die Tests bei weitgehend wolkenlosem Wetter durchgeführt. Es sind weitere Tests bei schlechtem Wetter notwendig.

Aus diesem Grund wird in diesem Rahmen auf einen Kalman-Filter, als unabhängige Lösung, zurückgegriffen und differential GPS als Ergänzung in Situationen, wo eine absolut genaue Ortung notwendig ist.

4.1.3 Rahmenbedingungen

Sensoren

GNSS-Systeme: Mit einem GNSS-Empfänger lassen sich Signale von Satellitensystemen wie GPS, QZSS, GLONASS, sowie Galileo [1] empfangen. Die Genauigkeit liegt im Bereich von einigen Metern. GPS und Galileo sind im Bezug auf die Übertragungsfrequenz und Modulierungsverfahren weitgehend kompatibel, herkömmliche Empfänger können den sogenannten C/A-Code auslesen, welches auf der L1-Frequenz (1575,42 MHz) gesendet wird und das typische Zeitsignal enthält. Es gibt auch Empfänger, die zusätzlich das L2-Signal (1227,6 MHz) auswerten können. Das L2-Signal ist zwar nur für militärische Zwecke vorgesehen und daher mit dem sogenanntem Y-Code verschlüsselt. Durch spezielle Auswertealgorithmen, welche sich die Rohwerte der Trägerphasen der L1- und L2-Frequenz zunutze machen, können jedoch trotzdem die Ionosphärenfehler kompensiert und somit das Signal verbessert werden. Diese sogenannten Zweifrequenzempfänger benötigen jedoch Elektronik eines zusätzlichen Empfängers und sind damit schwerer, teurer und haben eine höhere Leistungsaufnahme [129]. In dieser Arbeit werden daher Algorithmen beschrieben, welche die Positionsinformation von herkömmlichen GPS-Empfängern verbessern.

Weiter Information zum Thema GNSS wurden im Rahmen einer Literaturrecherche in [61] zusammengetragen.

Beschleunigung: Der Beschleunigungssensor dient zur Berechnung von Geschwindigkeit und Position mithilfe des Strapdownalgorithmus. Die Güte aktueller MEMS-Beschleunigungssensoren hat sich in den letzten Jahren sehr verbessert. Die Bias-Fehler, wie sie in [129] beschrieben werden, sind nicht mehr besonders ausgeprägt und können vernachlässigt werden. Damit ist die einzige Fehlerquelle des Beschleunigungssensors sein Rauschen.

Drehraten: Mit dem Drehratensensor kann durch Integration der Winkelgeschwindigkeit auf die aktuellen Fluglagewinkel geschlossen werden. Sind diese bekannt, so kann mithilfe des Strapdownalgorithmus durch Integration der Beschleunigung auf Geschwindigkeit und Position zurückge-

[1] Bis 2016 sollen die ersten Galileo-Dienste zur Verfügung stehen, das gesamte Netz von 30 Satelliten soll bis 2020 fertiggestellt werden. Die Signale von Galileo sind kompatibel zu GPS. Es werden jedoch im Vergleich zu GPS noch weitere Dienste, wie etwa Echtzeit-Ortung von Notrufen. [40] angeboten.

schlossen werden. Für gewöhnlich ist der Messwert der Drehraten jedoch mit einem konstanten Bias-Fehler behaftet. Das führt zu Integrationsfehlern in der Winkelberechnung und muss bei den folgenden Fusionsalgorithmen berücksichtigt werden.

Magnetfeld: Die Methode das Erdmagnetfeld für die Navigation zu nutzen ist bereits fast tausend Jahre alt und dessen Entdeckung hatte dramatische Auswirkung auf die Geschichte der modernen Zivilisation [32]. Die Feldlinien des Erdmagnetfeldes treten an der Südhalbkugel aus der Erde aus und am Nordpol wieder in die Erde ein. Dies wird in Abbildung 4.2 dargestellt. Die Inklination [2] beträgt hierzulande in etwa zwischen $62°$ und $70°$ [66]. Der Betrag der magnetischen Flussdichte variiert auf der Erde, je nach Ort, zwischen $22\ \mu T$ und $67\ \mu T$ [93].

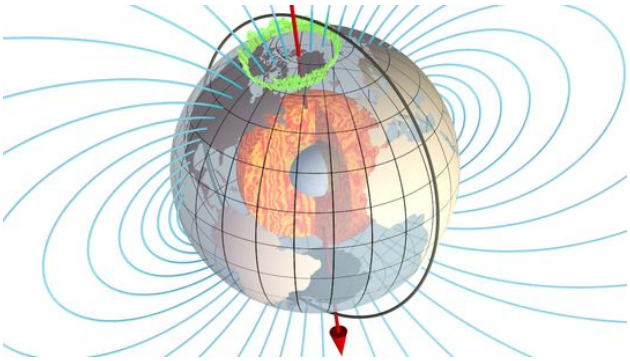


Abbildung 4.2: Magnetfeld der Erde [66]

Durch die Messung des Richtungsvektors des Magnetfeldes kann eine Orientierung des Flugroboters bezüglich der Erde bestimmt werden.

Barometrischer Höhendruck: Für die Schätzung der Höhe des Flugroboters steht ein barometrischer Höhensensor zur Verfügung. Über inertiale Sensorik allein lässt sich die Höhe nicht bestimmen. Die Positionsmessung des GNSS-Systems beinhaltet zwar ebenfalls die Höheninformation, diese ist jedoch dessen ungenauster Wert. Außerdem ist es sinnvoll auch bei Ausfällen der Satellitennavigation, in der Lage zu sein die Höheninformation zuverlässig zu stützen. Dies kann durch die Fusion der Drucksensorwerte mit den Werten von Beschleunigung und Drehraten gewährleistet werden.

Anforderungen

Ein Quadrocopter ist ein instabiles System, das nur durch eine permanente und zeitlich performante Regelung der Aktoren in Balance gehalten werden kann. Für diesen Zweck gibt es folgende Regelschleifen:

- **Fluglageregelung:** Aufgrund der naturgemäßen Instabilität des Systems werden in einer inneren, schnellen Regelschleife die Fluglagewinkel geregelt und so der Quadrocopter in Balance gehalten. Für die Regelgröße dieses Reglers werden die Fluglagewinkel benötigt.

[2] Inklination bezeichnet den Neigungswinkel zwischen Horizontalebene der Erde und dem Richtungsvektor des Erdmagnetfeldes [93]. Unter [66] finden sich Deklinationskarten.

- **Positions- sowie Höhenregelung und Navigation:** Flughöhen- und Positionsregelung werden in langsameren, die Fluglageregelung umschließenden Reglern realisiert. Für diese Regler werden Geschwindigkeit und Position benötigt.

Fluglageschätzung: Für die Fluglageregelung ist es notwendig qualitativ hochwertige Fluglagewerte in 6 Freiheitsgraden zur Verfügung zu haben. Die direkte Ermittlung der Fluglage aus den einzelnen Sensoren ist aufgrund deren Fehlercharakteristik nicht möglich. Beispielsweise würde eine Integration des Drehratensensors durch dessen Fehler-Bias schnell zu großen Driftfehlern führen. Auch Rückschlüsse auf den Lagewinkel des Flugroboters durch die Messung des Gravitationsvektors, führen ohne Mittelung nur im ruhenden Zustand zu wahren Ergebnissen. Um ausreichend genaue Werte zu erhalten, ist es deshalb notwendig verschieden Sensoren zu fusionieren. Algorithmen zur Fluglagebestimmung mithilfe der inertialen Messeinheit, bestehend aus Beschleunigungs- und Drehratensensor, werden in Kapitel 4.3.1 untersucht.

Positionsschätzung: Die Genauigkeiten von GNSS-Systemen wie GLONASS, Galileo oder GPS genügen, durch deren hohe Standardabweichung von mehreren Metern, nicht den Anforderungen einer Positionsbestimmung des Flugroboters. Auch die Lösung des Strapdownalgorithmus würde innerhalb kurzer Zeit zu großen Integrationsfehlern führen. Da die **GNSS-Systeme** langzeitgenaue Positionswerte und **Trägheitsnavigation** [3] kurzzeitgenaue Positionswerte liefern, müssen sie in geeigneter Weise klug fusioniert werden, dass sie über alle Zeitbereiche gute Werte liefern. Diese Kombination des GNSS-Systems mit Trägheitsnavigation bezeichnet man als Loosely Coupled. Zusätzlich wird diese Lageschätzung durch den Magnetfeld- und den barometrischen Höhendrucksensor gestützt. Hierfür wird in Kapitel 4.3.2 ein extended Kalman-Filter entworfen.

Als Tightly Coupled bezeichnet man die Stützung der GNSS-Werte durch Pseudorange-Daten. Da ein solches System jedoch erheblich mehr Integrationsaufwand mit sich bringt und rechenintensiver ist wird meistens ein Loosely-Coupled-System bevorzugt. [129], [93]

Eine weitere Möglichkeit genauere Positionswerte zu erhalten, ist es mithilfe einer stationären Referenzstation Pseudorange- und Trägerphasenmessungen vorzunehmen. Da die Position der Referenzstation bekannt ist, können systematische Fehler, wie Ionosphären- oder Stratosphärenfehler herausgerechnet werden. Dieses Verfahren wird als differential GPS (DGPS) bezeichnet. Als Referenzstationen können geostationäre Satelliten oder ein fest montiertes GPS-Modul, mit über längere Zeit gemitteltem GPS-Signal, dienen. Mit zweiterer Methode können Genauigkeiten im Dezimeter-Bereich erreicht werden. In [60] wurde getestet ob sich ein solches System zur Lokalisierung eines Flugroboters eignet. Es hat sich gezeigt, dass es, nach Verlust der Verbindung zu GPS-Satelliten, sehr lange dauern kann bis wieder ein stabiles Korrektursignal berechnet wird. Aus diesem Grund ist diese Technik für eine Positionsregelung nur bedingt geeignet. Vielmehr ist es ein Zusatzfeature, wenn eine positionsgenaue Landvermessung vorgenommen werden soll, oder wenn Punkte sehr exakt angeflogen werden sollen. Aus diesem Grund wird in diesem Rahmen auf ein Loosely Coupled System in Form eines extended Kalman-Filters gesetzt.

[3] Trägheitsnavigation (Inerital Navigation) ist die Schätzung von Orientierung und Position einzig durch inertiale Sensorik (Beschleunigung, Drehraten). Der aktuelle Wert wird basierend auf zurückliegender Werte berechnet (Dead-Reckoning).

4.2 Theorie

4.2.1 Komplementärfilter

Hintergrund

Oft hat man Werte von unterschiedlichen Sensoren, eines zuverlässig in niedrigen und ein anderes in höheren Frequenzbereichen, welche für die Bestimmung einer einzelnen Größe geeignet sind. Um jene Größe zu schätzen bietet sich dann ein Komplementärfilter an.

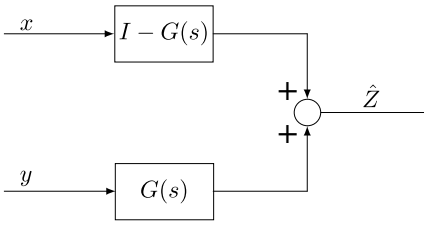


Abbildung 4.3: Grundprinzip Komplementärfilter

Beim Komplementärfilter werden die Sensorsignale mit einem Hochpass und einem Tiefpass entsprechend obiger Abbildung vereint. Die Sensorsignale, die in niedrigen Frequenzbereichen verlässlich sind werden durch einen Tiefpass und jene, welche in hohen Frequenzbereichen verlässlich sind durch einen Hochpass gefiltert.

Ansatz nach Colton

In [113] wird ohne Herleitung ein Filter zur Schätzung eines Winkels $\hat{\theta}$ aus der Messung der Winkelgeschwindigkeit ω_{gyro} und dem aus den Messungen eines Beschleunigungssensors berechneten Winkels θ_{acc} nach der Grundgleichung (4.1) vorgestellt.

$$\hat{\theta} = a \cdot (\hat{\theta} + \omega_{gyro} dt) + (1 - a) \cdot \theta_{acc} \tag{4.1}$$

Dabei lässt sich die Zeitkonstante τ für die Trennfrequenz des Hoch- und Tiefpassfilters aus der Konstanten a durch $\tau = \frac{a \cdot dt}{1-a}$ berechnen, wobei dt die Periodendauer der Frequenz, mit welche der Filter druchgeführt wird, darstellt.

Ansatz nach Oliviera/Pascoal/Kaminer

Im Folgenden wird ein Komplementärfilter anhand eines Beispiels beschrieben, wie es in [101] vorgestellt wird.

Sei ψ der Steuerkurs eines Fahrzeuges. Er soll durch die Sensorwerte eines Drehratensensors r_ω und eines Magnetfeldsensors ψ_ω gemessen werden. Daraus ergibt sich der Zusammenhang $r = \dot{\psi}$.

Beide Messungen sind naturgemäß störungsbehaftet. Nimmt man die Laplace-Transformierten $\psi(s)$ und $r(s)$ von ψ und r kann man für $k > 0$ folgende Gleichung aufstellen [101]:

$$\psi(s) = \frac{s+k}{s+k}\psi(s) = T_1(s)\psi(s) + T_2(s)\psi(s) \tag{4.2}$$

Der Bruch aus Gleichung (4.2) lässt sich zerlegen zu $T_1(s) = k/(s+k)$ und $T_2(s) = s/(s+k)$. Außerdem muss die Gleichung $T_1(s) + T_2(s) = I$ erfüllt werden. Durch den Zusammenhang $r(s) = s\psi(s)$ kann man die Gleichung (4.2) umschreiben zu $\psi(s) = F_\psi\psi(s) + F_r(s)r(s)$, mit $F_\psi(s) = T_1(s) = k/(s+k)$ und $F_r(s) = 1/(s+k)$. Multipliziert man diese Gleichung aus kann sie auf den Term $s\psi(s)$ aufgelöst werden. Damit lässt sich der Filterterm nach Gleichung (4.3) im Zustandsraum darstellen [101]:

$$\dot{\hat{\psi}} = -k\hat{\psi} + k\psi_m + r_m = r_m + k(\psi_m - \hat{\psi}) \tag{4.3}$$

Diese Filterberechnung wird in folgender Abbildung grafisch dargestellt [101].

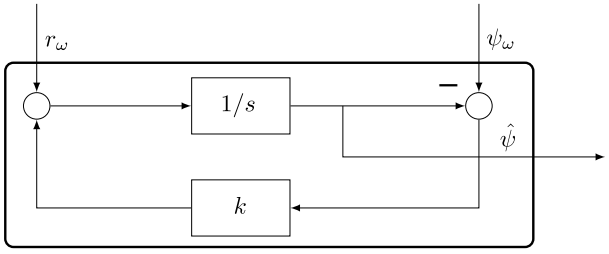


Abbildung 4.4: Komplementärfilter nach [101]

Zur Erläuterung sollen noch folgende Zusammenhänge genannt werden:

- $T_1(s)$ ist ein Tiefpassfilter, angewendet auf die Messungen des Magnetfeldsensors. Hochfrequenten Messungen wird hier nicht vertraut.
- $T_2(s)$ ist ein Hochpassfilter, angewendet auf die Werte des Drehratensensors. Er bildet den komplementären Bereich, in welchem der Magnetfeldsensor keine Werte liefert, ab.
- Der Bereich der Grenzfrequenz wird durch die Wahl des Parameters k gewählt

4.2.2 Kalman-Filter

Hintergrund

1960 stellte R.E. Kalman [4] seine berühmte Arbeit [78] über das Kalman-Filter, eine rekursive Lösung des diskreten linearen Filter Problemes, vor. Es gewann seither durch die Verbesserung der Rechenleistung immer mehr Anwendungen. Vor allem in der autonomen Navigation wurden viele Lösungen basierend auf dem Kalman-Filter vorgestellt. Die bekannteste Anwendung ist wohl das Apollo Programm der NASA in den sechziger Jahren, die Neil Armstrong als ersten Mann auf den Mond brachte. [56]

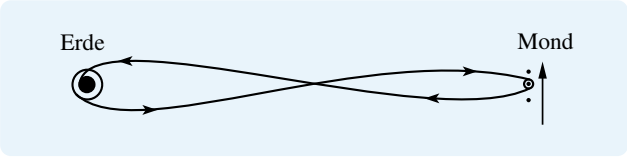


Abbildung 4.5: Nominale Appollo Trajektorien, skizziert ohne die Berücksichtigung der orbitalen Geschwindigkeit des Mondes um die Erde (1 km/h). Jene orbitale Geschwindigkeit des Mondes macht die Berechnung der Trajektorien noch komplizierter. [56]

Das Kalman-Filter ist eine Reihe von Gleichungen, welche auf effektive Weise rekursiv den Zustand eines Prozesses schätzt und dabei den mittleren quadratischen Fehler minimiert. Für eine erste Schätzung wird ein gewisses Systemverständnis verlangt. Diese erste Schätzung wird dann durch eine Messung korrigiert. Sowohl das Systemmodell, als auch das Messmodell werden als gestörte statistische Prozesse gesehen, welche mit System- und Messstörungen in Form von weißem gaußverteiltem Rauschen betroffen sind. *Das Kalman-Filter ist kein Filter im klassischem Sinne, welches ein Signal im Frequenzbereich beschneidet. Es schätzt für ein lineares, dynamisches Systemmodel mit stochastischer Erregung den zeitlich veränderlichen Zustand aus den gestörten Beobachtungen des Sytems.* [85]

Abbildung 4.6: R.E. Kalman bei der Überreichung des Charles Stark Draper Preises [110]

Der ursprüngliche Ansatz setzte ein lineares System- und Messmodell voraus. Im Zuge der Apollo Missionen wurde jedoch eine Version für nichtlineare Systeme, das extended Kalman-Filter

[4] R.E. Kalman gilt nicht nur als einer der einflussreichsten Forscher der Systemtheorie, sondern auch als einer der Wegbereiter dieses Forschungsgebietes. [30] Seine Leistungen wurden mehrmals durch Auszeichnungen honoriert, zuletzt 2008 mit dem Charles Stark Draper Preis, überreicht durch den amtierenden Präsidenten der USA, Barack Obama. Dieser Preis ist für die Ingenieurswissenschaften ähnlich bedeutend wie der Nobelpreis für die Naturwissenschaften. [110]

(EKF), vorgestellt [92]. Es wurde durch den Vergleich mit Monte Carlo Simulationen gezeigt, dass der Schätzfehler einer Linearisierung des Problems mithilfe einer Taylorreihe klein genug ist um immer noch ausgezeichnete Ergebnisse zu liefern.

Gleichungen Kalman-Filter

Das zeitdiskrete Kalman-Filter setzt voraus, dass der Zustand $\mathbf{x_t}$ durch ein lineares zeitdiskretes Systemmodell nach Gleichung (4.4) zu einem Zeitpunkt t, für den aus dem vorhergehenden Zustand zum Zeitpunkt $t-1$ berechnet werden kann:

$$\mathbf{x_t} = \mathbf{F_t}\mathbf{x_{t-1}} + \mathbf{B_t}\mathbf{u_t} + \mathbf{w_t} \tag{4.4}$$

Wobei $\mathbf{x_t}$ den Zustandsvektor, $\mathbf{u_t}$ den Vektor für die Eingangswerte des Systems, $\mathbf{F_t}$ die Systemmatrix, $\mathbf{B_t}$ die Kontrollmatrix und $\mathbf{w_t}$ den Vektor, der das Systemrauschen enthält darstellen.

Außerdem muss durch eine Messung, dessen Werte im Messvektor $\mathbf{z_t}$ enthalten sind, mithilfe des Zustandsvektor $\mathbf{x_t}$ ein lineares System entsprechend Gleichung (4.5) herleitbar sein:

$$\mathbf{z_t} = \mathbf{H_t}\mathbf{x_t} + \mathbf{v_t} \tag{4.5}$$

Wobei $\mathbf{H_t}$ die Messmatrix und $\mathbf{v_t}$ das Messrauschen jeweils zum Zeitpunkt t darstellen.

Das Kalman-Filter arbeitet rekursiv in zwei Stufen. Zunächst wird mithilfe des Systemmodells der Zustandsvektor geschätzt. Anschließend wird die Schätzung durch die Erfassung der Messwerte korrigiert:

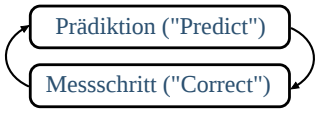


Abbildung 4.7: Ablauf Kalman-Filter [127]

Liegen keine Messungen vor, kann auch mehrmals der Prädiktionsschritt ausgeführt werden, solange bis wieder Messungen verfügbar sind. Folgende Abbildung zeigt jeweils die Wahrscheinlichkeitsdichtefunktionen des Systemmodells (a), des Messmodells (b) und schließlich der Schätzung des Kalman-Filters für einen Durchlauf von Prädiktion und Messschritt.

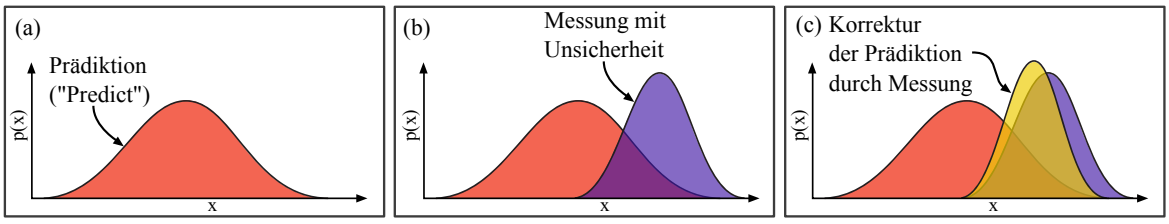


Abbildung 4.8: Ablauf des Kalman-Filters: (a) Schätzung (a priori) anhand des Systemmodells, (b) Erhebung der Messung, (c) Korrektur der Schätzung (a priori) durch die Messung (a posteriori) [122] [46]

Die Breite (Kovarianz) der Wahrscheinlichkeitsdichtefunktion kann als Maß der Vertrauenswürdigkeit des Wertes aufgefasst werden. In obiger Abbildung (c) sieht man, dass die Gaußsche Verteilung der Schätzung des Kalman-Filters sowohl schmaler ist als jene des propagierten und des gemessenen Wertes als auch, dass sie zwischen den beiden liegt.

Folgende Abbildung zeigt die Gleichungen des Kalman-Filteralgorithmus nach [46] im Detail [5]. Die Grundidee ist, dass der geschätzte Zustand durch eine Wahrscheinlichkeitsdichtefunktion dargestellt wird. Hierfür werden ein Vektor für den geschätzten Zustand $\hat{\mathbf{x}}_\mathbf{t}$ sowie eine Kovarianzmatrix $\mathbf{P_t}$ für den Schätzfehler benötigt.

Prädiktion ("Predict")

(1) $\hat{x}_{t|t-1} = F_t \hat{x}_{t-1|t-1} + B_t u_t$

(2) $P_{t|t-1} = F_t P_{t-1|t-1} F_t^T + Q_t$

Messschritt ("Correct")

(3) $K_t = P_{t|t-1} H_t^T (H_t P_{t|t-1} H_t^T + R_t)^{-1}$

(4) $\hat{x}_{t|t} = \hat{x}_{t|t-1} + K_t(z_t - H_t \hat{x}_{t|t-1})$

(5) $P_{t|t} = P_{t|t-1} - K_t H_t P_{t|t-1}$

Abbildung 4.9: Kalman-Filtergleichungen [46]

Prädiktion: Zunächst wird mithilfe des Systemmodells aus Gleichung (4.4) der aktuelle Zustand $\hat{\mathbf{x}}_\mathbf{t}$ des Filters über die Zeit propagiert **(1)** [6] [93]. Außerdem wird die Kovarianzmatrix der Schätzfehler $\mathbf{P_t}$ rekursiv aus dessen vorhergehenden Werten, sowie der Kovarianzmatrix der Unsicherheit des Systemmodells $\mathbf{Q_t}$ bestimmt **(2)**.

Messschritt: Im Messschritt werden die Schätzung $\hat{\mathbf{x}}_{\mathbf{t|t-1}}$, sowie die Kovarianzmatrix der Schätzfehler $\mathbf{P_t}$ korrigiert. Hierfür wird die Messung $\mathbf{z_t}$ herangezogen. Zunächst wird zu diesem Zweck die sogenannte Kalman-Gain-Matrix $\mathbf{K_t}$ berechnet **(3)**. Sie bestimmt die Gewichtung inwieweit die Messung Einfluss auf den Systemzustand hat. Anschließend wird mithilfe jener Kalman-Gain-Matrix der Systemzustand durch die Messung $\mathbf{z_t}$ korrigiert **(4)**. Schließlich wird auch die Kovarianzmatrix der Schätzfehler $\mathbf{P_t}$ korrigiert **(5)**. [93], [122]

Die Kovarianzmatrix der Unsicherheit des Systems $\mathbf{Q_t}$ beinhaltet das Vertrauen des Systems [46]. Sie berechnet sich aus dem Rauschvektor des Systems $\mathbf{w_t}$ nach Gleichung (4.6):

$$\mathbf{Q_t} = \mathbf{E}(\mathbf{w_t} \mathbf{w_t^T}) \tag{4.6}$$

Die Kovarianzmatrix der Messunsicherheit $\mathbf{R_t}$ beinhaltet das Vertrauen in die Messung [46]. Sie berechnet sich aus dem Rauschvektor der Messung $\mathbf{v_t}$ nach Gleichung (4.7):

$$\mathbf{R_t} = \mathbf{E}(\mathbf{v_t} \mathbf{v_t^T}) \tag{4.7}$$

Das Vertrauen in das Systemmodell $\mathbf{Q_t}$ sowie das Vertrauen in die Messung $\mathbf{R_t}$ können bei Bedarf zu jeder Zeit angepasst werden. Sind beispielsweise bei einer GPS-Messung wenig Satelliten

[5] In $\hat{\mathbf{x}}_{\mathbf{t-1|t-1}}$ steht das Dach exemplarisch für einen Schätzwert, der erste Index steht für den Schätzschritt und der zweite Index für den Updateschritt (a priori/a posteriori).

[6] Die fett gedruckten Nummern in Klammern beziehen sich auf die Nummerierung in Abbildung 4.9.

vorhanden, kann das Vertrauen in die Messung so angepasst werden, dass den GPS-Messungen weniger Gewichtung zukommt, indem die Werte für die GPS-Messung im Rauschvektor $\mathbf{v_t}$ entsprechend geändert werden. Dadurch wird in diesem Fall der Einfluss des Systemmodells verstärkt. Sind wieder mehr Satelliten vorhanden, kann diese Änderung wieder rückgängig gemacht werden.

Gleichungen extended Kalman-Filter

Für die korrekte Funktion des Kalman-Filters ist es zwingend notwendig, dass das Systemmodell linear ist, denn nur dann resultiert die Beobachtung einer Gaußschen Zufallsvariable wieder in einer Gaußschen Zufallsvariable [122]. Das extended Kalman-Filter (EKF) ist die nichtlineare Version des Kalman-Filters. Beim EKF wird eine Linearisierung durch eine Taylorreihe, die nach dem erstem Glied abgeschnitten wird, vorgenommen.

Prädiktion ("Predict")

(1) $\hat{x}_{t|t-1} = g(u_t, x_{t-1|t-1})$

(2) $P_{t|t-1} = \Psi_t P_{t-1|t-1} \Psi_t^T + Q_t$

Messschritt ("Correct")

(3) $K_t = P_{t|t-1} \Gamma_t^T (\Gamma_t P_{t|t-1} \Gamma_t^T + R_t)^{-1}$

(4) $\hat{x}_{t|t} = \hat{x}_{t|t-1} + K_t(z_t - h(\hat{x}_{t|t-1}))$

(5) $P_{t|t} = P_{t|t-1} - K_t \Gamma_t P_{t|t-1}$

Abbildung 4.10: Extended Kalman-Filtergleichungen

Der Algorithmus des EKFs ähnelt sehr jenem des Kalman-Filters. Folgende Tabelle zeigt die Hauptunterschiede [122]:

	Kalman-Filter	EKF
Systemmodell Zeile **(1)**	$F_t \hat{x}_{t-1\|t-1} + B_t u_t$	$g(u_t, x_{t-1})$
Messmodell Zeile **(4)**	$H_t \hat{x}_{t\|t-1}$	$h(x_t)$

Die linearen Schätzungen des Kalman-Filters werden durch die nichtlinearen Funktionen des EKFs ersetzt. Außerdem werden im EKF statt der linearen Systemmatrizen $\mathbf{F_t}$, $\mathbf{B_t}$ und $\mathbf{H_t}$ die Jacobi-Matrizen $\mathbf{\Phi_t}$ und $\mathbf{\Gamma_t}$ verwendet. Dabei korrespondiert $\mathbf{\Phi_t}$ mit $\mathbf{F_t}$ und $\mathbf{B_t}$ und $\mathbf{\Gamma_t}$ korrespondiert mit $\mathbf{H_t}$. [122]

Rechenaufwand und Präzision

Rechenaufwand

Der größte Rechenaufwand des Filters liegt in der Inversion der Matrix S aus Gleichung **(3)** des Kalman-Filteralgorithmus aus Abbildung 4.9:

$$P = PH^T(\overbrace{HPH^T + R}^{S})^{-1}$$

Hier muss eine $r \times r$-Matrix invertiert werden, wobei r der Anzahl der Elemente des Messvektors z entspricht. Da man zeigen kann, dass die Matrix S immer symmetrisch positiv definit ist lässt sich deren Inverse durch das Lösen folgender Gleichung mithilfe des Cholewsky-Verfahrens finden:

$$SX = I \text{ mit der Lösung } X = S^{-1}$$

Diese Methode ist numerisch stabil und ihr Rechenaufwand beträgt $O(1/3 r^3)$ [104]. [7] Neben dem Cholewsky-Verfahren gibt es schnellere Methoden die Matrixinversion durchzuführen. Laut [122] basieren die derzeit schnellsten Inversionsalgorithmen auf den schnellen Matrixmultiplikationen von Coppersmith und Winograd [38] und weisen damit einem Rechenaufwand von $O(r^{2,4})$ auf. In diesem Rahmen wurde jedoch nicht geprüft, ob diese Verfahren praxistauglich sind.

Eine Möglichkeit den Rechenaufwand zu reduzieren ist es die Inversion jener Matrix massiv zu vereinfachen, indem man Messungen nacheinander und einzeln verarbeitet (Sequential Filtering) [118]. Damit reduziert sich jene Inversion einer $r \times r$-Matrix zur Inversion einer 1×1-Matrix und damit zu einer schlichten skalaren Division. Der Rechenaufwand ist dann nur noch proportional zu r.

Wenn die Matrizen R und Q, sowie das zugrunde liegende Systemmodell zeitinvariant sind, kann es vorkommen, dass die Kovarianzmatrix des Schätzfehlers P und damit auch die Kalman-Gain-Matrix K nach einigen Filterdurchläufen zu konstanten Werten konvergieren. Dann besteht die Möglichkeit diese vorher zu berechnen. Am einfachsten geht dies indem man den Kalman-Filteralgorithmus offline für eine bestimmte Zahl von Durchgängen durchlaufen lässt. Dadurch kann man jene Rechenschritte im Filteralgorithmus, die zu deren Berechnung dienen, einsparen. Man spricht von Steady-State Filtering [118].

Präzision

Auf eingebetteten Systemen mit geringer Wortbreite kann es unter bestimmten Umständen dazu kommen, dass die Kovarianzmatrix aus der Riccati-Gleichung [8] eine schlechte Konditionszahl erhält und damit die Präzision nicht mehr ausreichend ist. Dieses Problem trat erstmalig auf dem Apollo Guidance Computer, konstruiert zur Berechnung der Trajektorie von der Erde zum Mond, auf.

[7] Um numerische Matrix-Operationen in C/C++ durchzuführen lohnt es sich einen Blick auf die GNU Scientific Library (GSL) [50] zu werfen. Beispielsweise wird eine Funktion für das Cholewski-Verfahren angeboten.

[8] Die Gleichungen im Kalman-Filter, die zur Berechnung der Kovarianzmatrix führen werden zusammen Riccati-Gleichung genannt [118]

Abbildung 4.11: Apollo Guidance Computer (15-bit Fixpoint-Arithmetik) und Display/Keyboard-Einheit [130]

Da sich Änderungen häufig nur in den letzten Nachkommastellen niederschlagen, wird für das Kalman-Filter prinzipiell Auflösung in der Arithmetik benötigt [9]. In [118] werden zwei Alternativen zu den Riccati-Gleichungen vorgestellt, die auf Kosten des Rechenaufwandes die Genauigkeit erhöhen:

- Square-Root Filtering [10]
- U-D Filtering

Diese beiden Methoden erhöhen die Präzision sowie die Stabilität des Kalman-Filters, auf Kosten des Rechenaufwandes. Da jene Methoden in diesem Rahmen keine Anwendung finden, wird hier lediglich auf deren Existenz hingewiesen. Der geneigte Leser sei bei näherem Interesse auf die Lektüre von [118] verwiesen.

4.2.3 Koordinatentransformation

Koordinatensysteme für Erdnahe Navigation

In Abbildung 4.12 werden die hier verwendeten Koordinatensysteme gezeigt:

- **b-frame:** Das **körperfeste Koordinatensystemem** ist fest mit dem Luftfahrzeug verbunden. Dabei entspricht die Ausrichtung der Achsen jenen aus Abbildung 4.14. Markant ist hier, dass die z-Achse nach unten weist. Der Ursprung befindet sich im Luftfahrzeug. Die Komponenten der Messwerte der IMU werden in diesem Koordinatensystem angegeben. [129]
- **i-frame:** Das **inertiale Koordinatensystem** ist auf Fixsterne ausgerichtet und damit ein ruhendes Koordinatensystem. Dessen z^i-Achse fällt mit der Rotationsachse der Erde zusammen. Die Achsen x^i und y^i befinden sich in der Äquatorebene. Die Messwerte der IMU sind Drehraten und Beschleunigung des körperfesten Koordinatensystems bezüglich des inertialen Koordinatensystems. [129]
- **e-frame:** Der Ursprung und die z^e-Achse des **erdfesten Koordinatensystems** fallen mit jenem des Inertialkoordinatensystems zusammen. Das **erdfeste Koordinatensystem** ist fest

[9] Aussage von Wolfgang Högele

[10] Diese Methode fand auf dem 15-Bit Appollo Guidance Computer Anwendung. Der Appollo Guidance Computer wurde konstruiert um die Trajektorie von der Erde zum Mond zu berechnen. Eine schlechte Kondition der Kovarianzmatrix führte auf der 15-Bit Maschine jedoch zu Problemen. Die Lösung fand der MIT Student J.E. Potter, der als Student Teilzeit am Instrumentation Laboratory der NASA in Mountain View arbeitete. Der Trick war, mithilfe des Cholewski-Verfahrens die Kovarianzmatrix zu zerlegen. Man bezeichnete diese Methode als *Square-Root Filtering*. [56]

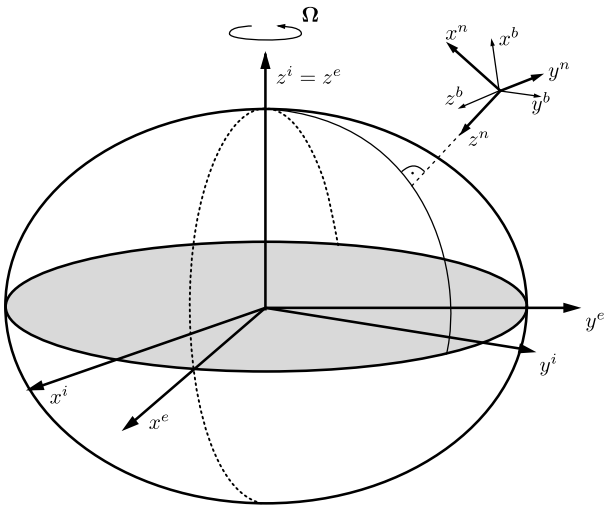


Abbildung 4.12: Verwendete Koordinatensysteme [129]

mit der Erde verbunden. Es dreht sich, mit der Erdrotationsgeschwindigkeit Ω, mit der Erde mit. Die Achsen x^e und y^e befinden sich in der Äquatorebene. [129]

- **n-frame:** Der Ursprung des **Navigationskoordinatensystems** fällt mit jenem des körperfesten Koordinatensystems zusammen. Die x^n-Achse weist in Nord- und die y^n-Achse weist in Ostrichtung. Beide liegen tangential zum Erdellipsoid. Die z^n-Achse weist nach unten. Sie enthält den Gravitationsvektor. [129]

Damit die Ausrichtung von Nord, Ost und unten bezüglich des Navigationskoordinatensystems bei einer Drehung des Erdkoordinatensystems erhalten bleibt, muss dieses ständig nachgedreht werden. Die resultierende Drehrate wird als Transportdrehrate bezeichnet.

Nomenklatur

Die Komponenten jedes Vektors haben ihre Gültigkeit in den Koordinatensystemen aus Kapitel *Koordinatensysteme für Erdnahe Navigation*. Auch gelten für jeden Vektor Beziehungen zu den anderen Koordinatensystemen. Die Schreibweise des Vektors ist deshalb so gestaltet, dass dies erkennbar wird. Am Beispiel des Geschwindigkeitsvektors soll exemplarisch die hier verwendete Nomenklatur beschrieben werden.

$$\vec{v}_{eb}^{\,n}$$

Der obere Index gibt an in welchem Koordinatensystem die Größe gegeben ist. In diesem Fall handelt es sich um die Geschwindigkeit im Navigationskoordinatensystem. Die beiden unteren Indizes geben die Beziehungen der Koordinatensysteme untereinander an. Hier ist beispielsweise die Geschwindigkeit des körperfesten Koordinatensystems bezüglich des erdfesten Koordinatensystems angegeben.

Mit Rotationsmatrizen kann ein Vektor aus einem Koordinatensystem durch Rotation in ein anderes Koordinatensystems übergeführt werden. Dabei wird ein Vektor vom Koordinatensystem des

unteren Index zum Koordinatensystem des oberen Index gedreht.

$$C_b^n$$

Im obigen Beispiel handelt es sich um die Rotationsmatrix vom körperfesten Koordinatensystem in das Navigationskoordinatensystem. Diese kann beispielsweise dazu verwendet werden die Beschleunigung $\vec{a}_{ib}^{\,n}$ im Navigationskoordinatensystem aus der Beschleunigung $\vec{a}_{ib}^{\,b}$ aus dem körperfesten Koordinatensystems zu berechnen. Da die Sensoren fest mit dem körperfesten Koordinatensystem verbunden sind, kann man die Komponenten von $\vec{a}_{ib}^{\,b}$ den Werten des Beschleunigungssensors entnehmen. Damit ergibt sich Gleichung (4.8).

$$\vec{a}_{ib}^{\,n} = C_b^n \vec{a}_{ib}^{\,b} \tag{4.8}$$

Die Rotationsmatrizen werden im Detail in Kapitel *Rotation* beschrieben.

Rotation

In den vorgestellten Algorithmen ist es häufig notwendig Vektoren eines Koordinatensystems in ein Anderes zu überführen. Hierfür werden Vektorrotationen verwendet. Folgend werden knapp jene Formeln beschrieben, die zum Verständnis dieser Algorithmen benötigt werden. Eine Einführung in dieses Thema ist ebenfalls in [129], [49] zu finden.

Richtungskosinusmatrix

Die Fluglage kann anschaulich durch die Eulerwinkel Roll θ, Pitch ψ und Yaw ϕ beschrieben werden. Die räumliche Zuordnung der Eulerwinkel in einem Luftfahrzeug wird in Abbildung 4.14 dargestellt. Sie geben die Winkellage zwischen körperfestem Koordinatensytem des Fluggerätes und Navigationskoordinatensystem an. Eine hintereinander ausgeführte und nicht kommutative Drehung eines Vektors um jene Eulerwinkel kann durch Multiplikation mit der Richtungskosinusmatrix aus Gleichung (4.9) durchgeführt werden. Dabei wird in diesem Fall zunächst um die z-Achse (Yaw, ϕ), anschließend um die y-Achse (Pitch, ψ) und zuletzt um die x-Achse (Roll, θ) gedreht. [93] Eine räumliche Darstellung dieser Drehungen findet sich in Abbildung 4.13.

$$C_b^n = \begin{pmatrix} \cos(\theta)\cos(\psi) & -\cos(\phi)\sin(\psi) + \sin(\phi)\sin(\theta)\cos(\psi) & \sin(\phi)\sin(\psi) + \cos(\phi)\sin(\theta)\cos(\phi) \\ \cos(\theta)\sin(\psi) & \cos(\phi)\cos(\psi) + \sin(\phi)\sin(\theta)\sin(\psi) & -\sin(\phi)\cos(\psi) + \cos(\phi)\sin(\theta)\sin(\psi) \\ -\sin(\theta) & \sin(\phi)cos(\theta) & \cos(\phi)\cos(\theta) \end{pmatrix} \tag{4.9}$$

Mittels der Richtungskosinusmatrix kann entsprechend Gleichung (4.10) auch die zeitliche Änderung der Eulerwinkel angegeben werden [49].

$$R = \begin{pmatrix} \dot{\theta} \\ \dot{\psi} \\ \dot{\phi} \end{pmatrix}_{nb}^{b} = \begin{pmatrix} (\omega_{nb,y}^b \sin(\theta) + \omega_{nb,x}^b \cos(\theta)) \tan(\psi) + \omega_{nb}^b \\ \omega_{nb}^b \cos(\theta) - \omega_{nb,z}^b \sin(\theta) \\ \omega_{nb,y}^b \sin(\theta) + \omega_{nb,z}^b \cos(\theta) / \cos(\psi) + \omega_{nb,x}^b \end{pmatrix} \tag{4.10}$$

In Gleichung (4.10) sieht man, dass bei einem Pitch-Winkel von $\pm\psi = 90°$ durch eine Division durch Null eine Singularität auftritt. Räumlich gesehen entspricht dies dem Aufeinanderfallen von

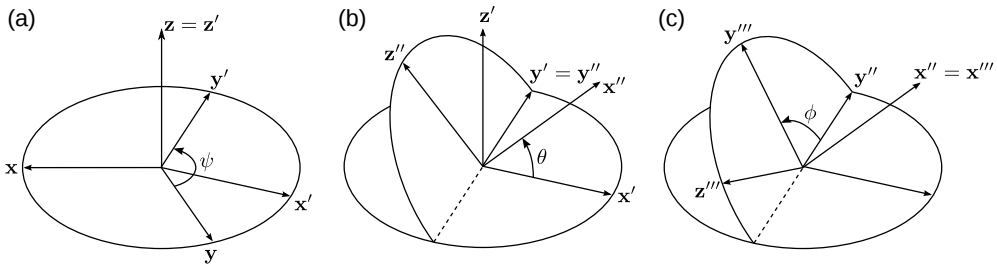


Abbildung 4.13: ZYX-Drehung **(a)** Yaw ϕ um z-Achse **(b)** Pitch ψ um y-Achse **(C)** Roll θ um x-Achse

Drehachsen und damit dem Ausfall von Freiheitsgraden in der Drehbewegung. Man bezeichnet dieses Problem der Richtungskosinusmatrix als Gimbal-Lock.

Links zu Animationen und weiterführenden Informationen, sowie Code zur Veranschaulichung dieser Zusammenhänge finden sich unter [58].

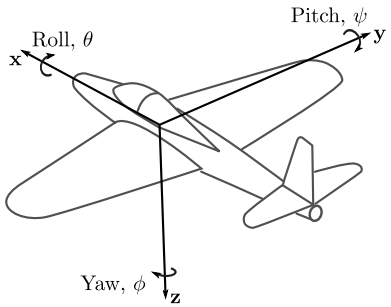


Abbildung 4.14: Roll-, Pitch- und Yaw-Winkel am Luftfahrzeug

Quaternionen

Die Lage zweier Koordinatensysteme kann mithilfe des Orientierungsvektors $\vec{\sigma}$ beschrieben werden:

$$\vec{\sigma} = (\sigma_x, \sigma_y, \sigma_z)^T \tag{4.11}$$

Ein Quaternion [11] beschreibt eine Rotation durch eine einzige Drehung. Dies geschieht durch eine Quaternionenmultiplikation, welche näher in Gleichung (4.13) beschrieben wird. Es können auch mehrere Drehungen durch die Aneinanderreihung von Quaternionenmultiplikation durchgeführt werden. Dabei tritt das Problem der Mehrdeutigkeit der Eulerwinkel nicht auf.

[11] In den 1950er Jahren führte die Angst zwischen den Großmächten USA und Russland und weiteren Nationen zu einem Wettrüsten. Dies löste einen Technologieboost im Bereich Luft- und Raumfahrt aus. In diesem Zuge wurden für Anwendungen, wie beispielsweise Fernlenkraketen, die von Hamilton 1844 vorgestellten Quaternionen [65] für die Beschreibung der Orientierung eines Körpers wiederentdeckt. (Es sei an dieser Stelle vom Autor angemerkt, dass zivile Programme, wie die Internationale Raumstation ISS beweisen, dass nicht nur Krieg und Misstrauen, sondern Forscherdrang, Neugier und internationale Zusammenarbeit zu herausragenden technischen Leistungen, ebenfalls im zivilem Bereich, führen können.)

Ein Quaternion ist ein vierdimensionaler Vektor [12], dessen Betrag auf eins normiert ist, wie in Gleichung (4.12) dargestellt.

$$q_b^n = \begin{pmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \end{pmatrix} = \begin{pmatrix} \cos(\sigma/2) \\ \sigma_x/\sigma \ \sin(\sigma/2) \\ \sigma_y/\sigma \ \sin(\sigma/2) \\ \sigma_z/\sigma \ \sin(\sigma/2) \end{pmatrix} \tag{4.12}$$

Es kann durch Quaternionenmultiplikation ein Vektor eines Koordinatensystems in ein anderes überführt werden. [13] Hierfür muss der Vektor mit einer führenden Null erweitert werden. Dies ist entsprechend [49] in Gleichung (4.13) dargestellt.

$$\begin{pmatrix} 0 \\ x^n \end{pmatrix} = q_b^n \bullet \begin{pmatrix} 0 \\ x^b \end{pmatrix} \bullet q_n^b$$
$$\text{mit } \ q_b^n \bullet (\ldots) = \begin{pmatrix} q_0 & -q_1 & -q_2 & -q_3 \\ q_1 & q_0 & -q_3 & q_2 \\ q_2 & q_3 & q_0 & -q_1 \\ q_3 & -q_2 & q_1 & q_0 \end{pmatrix} \cdot (\ldots) \tag{4.13}$$

Alternativ kann nach [129] aus der Richtungskosinusmatrix die sogenannte Euler-Rodriguez-Darstellung, welche das Navigationskoordinatensystem in das körperfeste Koordinatensystem überführt, [116] berechnet werden. Sie ist in Gleichung (4.14) [14] dargestellt.

$$C_b^n = \begin{pmatrix} (q_0^2 + q_1^2 - q_2^2 - q_3^2) & 2(q_1q_2 - q_0q_3) & 2(q_1q_3 + q_0q_2) \\ 2(q_1q_2 + q_0q_3) & (q_0^2 - q_1^2 + q_2^2 - q_3^2) & 2(q_2q_3 - q_0q_1) \\ 2(q_1q_3 - q_0q_2) & 2(q_2q_3 + q_0q_1) & (q_0^2 - q_1^2 - q_2^2 + q_3^2) \end{pmatrix} \tag{4.14}$$

In Gleichung (4.15) wird gezeigt wie eine Hintereinanderschaltung von Drehungen vollzogen wird. Diese Operation ist nicht kommutativ [49].

$$q_b^n = q_e^n \bullet q_b^e \tag{4.15}$$

Die Quaternionen-Differentialgleichung (4.16) stellt die zeitliche Änderung eines Quaternions aufgrund von Drehraten dar [49]. Eine Herleitung für diese Gleichung liegt in [73] vor.

$$\dot{q_b^n} = \frac{1}{2} q_b^n \bullet \begin{pmatrix} 0 \\ \omega_{nb}^b \end{pmatrix} = \frac{1}{2} \begin{pmatrix} q_0 & -q_1 & -q_2 & -q_3 \\ q_1 & q_0 & -q_3 & q_2 \\ q_2 & q_3 & q_0 & -q_1 \\ q_3 & -q_2 & q_1 & q_0 \end{pmatrix} \begin{pmatrix} 0 \\ \omega_{nb}^b \end{pmatrix} \tag{4.16}$$

Nach Gleichung (4.17) können Quaternionen in Euler-Winkel umgewandelt werden [49].

$$\begin{pmatrix} \theta \\ \psi \\ \phi \end{pmatrix} = \begin{pmatrix} \arcsin(2(q_0q_2 - q_3q_1)) \\ \arctan(\frac{2(q_0q_3 + q_1q_2)}{q_0^2 + q_1^2 - q_2^2 - q_3^2}) \\ \arctan(\frac{2(q_0q_1 + q_2q_3)}{q_0^2 - q_1^2 - q_2^2 + q_3^2}) \end{pmatrix} \tag{4.17}$$

[12] Zur Verbesserung der Lesbarkeit wird bei der Darstellung des Quaternionenvektors $\vec{q}$ auf den Vektorpfeil verzichtet.
[13] Für die Quaternionenmultiplikation wird das Symbol $\bullet$ verwendet.
[14] Die Erfüllbarkeit dieser Gleichung setzt voraus, dass das Quaternion in normalisierter Form $q_{norm} = \frac{q}{|q|}$ vorliegt [91].

Strapdown-Algorithmus

Unter Strapdownalgorithmus versteht man die Propagation von Lage, Geschwindigkeit und Position durch die Integration der Werte des Drehratensensors und des Beschleunigungssensors. Da hier sowohl Corioliskräfte, welche aufgrund der Erdrotation auftreten, als auch Erd- [15] und Transportdrehrate [16] vernachlässigt werden können, wird hier eine vereinfachte Form des Strapdownalgorithmus vorgestellt, dargestellt in Abbildung 4.15. Eine detailliertere Beschreibung kann in [129] nachgelesen werden.

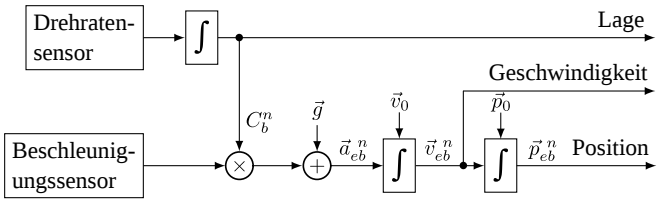


Abbildung 4.15: Vereinfachter Strapdown-Algorithmus [93]

Der Vektor der Drehrate ω_{nb}^{b} im Navigationskoordinatensystem kann nach Gleichung (4.18) aus dem körperfesten Koordinatensystem berechnet werden. Dabei werden Transportdrehrate und Erddrehrate berücksichtigt [93].

$$\vec{\omega}_{nb}^{b} = \vec{\omega}_{ib}^{b} - \underbrace{\vec{\omega}_{ie}^{b}}_{\substack{\text{Erddreh-}\\\text{rate}}} - \underbrace{\vec{\omega}_{en}^{b}}_{\substack{\text{Transport-}\\\text{drehrate}}} \tag{4.18}$$

Da die Geschwindigkeit in der hier gestellten Anforderung gering ist und nur geringe Wege zurückgelegt werden, fällt die Transportdrehrate nicht ins Gewicht. Auch die Erddrehrate kann im Vergleich zum Fehler des Drehratensensors vernachlässigt werden. Unter Berücksichtigung dieser Vereinfachungen ergeben sich die Differentialgleichungen für Lage, Geschwindigkeit und Position aus den gemessenen Werten von Drehrate und Beschleunigung der IMU nach Gleichung (4.19).

$$\begin{aligned} \dot{\vec{\sigma}} &= \vec{\omega}_{ib}^{\,b} \\ \dot{\vec{v}}_{eb}^{\,n} &= C_n^b \vec{a}_{ib}^{\,b} + \vec{g}_l^n \\ \dot{\vec{p}}_{eb}^{\,n} &= \vec{v}_{eb}^{\,n} \end{aligned} \tag{4.19}$$

mit $\vec{g}_l^n = (0, 0, g_0)^T$. Dabei kann der Betrag von g_0 vereinfacht zu $g_0 = 9,80665 m/s^2$ angenommen werden. Aufgrund der Fehlerbeiträge des Beschleunigungssensors und der geringen zurückgelegten Wegstrecken kann hier auf ein genaueres Gravitationsmodell, wie beispielsweise *WGS84*, verzichtet werden. [93]

[15] Die Erde dreht sich in 24 Stunden einmal um ihre Achse. Dies entspricht einer Winkelgeschwindigtkeit von $|\vec{\omega}_{ie}^{b}| = 7,292115 \cdot 10^{-5}\ rad/s$. Dieser Drehrate ist auch der Drehratensensor ausgesetzt.

[16] Aufgrund der Erdrotation und der Erdkrümmung muss sich bei der Zurücklegung von weiten Strecken das Navigationskoordinatensystem in einer Weise drehen, dass dessen z-Achse immer in Richtung der lokalen Vertikale weist. Dies wird in der Transportdrehrate berücksichtigt [129].

4.3 Sensorfusionsalgorithmen

4.3.1 Fluglageschätzung

Komplementärfilter

Ansatz nach Madgwick/Mahony

Madgwick hat im Rahmen seiner Dissertation einen AHRS (Attitude Heading Reference System) Algorithmus zur Lagebestimmung entwickelt. [88] [89] Auch Mahony hat einen Fusionsalgorithmus vorgestellt. [90] Die Algorithmen fusionieren jeweils 3-Achsen Beschleunigungs- sowie Drehraten- und wahlweise auch Magnetfeldsensoren. Nimmt man den Magnetfeldsensor mit hinzu werden die Algorithmen rechenintensiver, aber auch stabiler. Außerdem ist dann die Stützung des Yaw-Winkels zuverlässig.

Sowohl Madgwick als auch Mahony haben Code für ihre Algorithmen zur Verfügung gestellt. Dieser Code wird auf [132] unter der *Gnu General License* (GPL) für Matlab, C# und C zur Verfügung gestellt. Auf [71] ist auch eine Labview Version vorhanden.

Eine einfache Abwandlung des von Mahony vorgestellten Codes [132], welche auf Rechenleistung getrimmt wurde befindet sich im Anhang in Kapitel *Code Ansatz Mahony*.

Ansatz nach Oliviera/Pascoal/Kaminer

Für die Schätzung der Fluglage kann das in Kapitel *Ansatz nach Oliviera/Pascoal/Kaminer* beschriebene Filter herangezogen werden. So kann die Winkelgeschwindigkeit ω des Lagewinkels θ nach Gleichung (4.20) geschätzt werden [84]. Dabei sind ω_{gyro} die Messung der Winkelgeschwindigkeit durch den Drehratensensor und ω_{acc} die Schätzung der Winkelgeschwindigkeit durch den Beschleunigungssensor.

$$\dot{\hat{\theta}} = \omega_{gyro} + k_p(\theta_{acc} - \hat{\theta}) \tag{4.20}$$

Da jedoch nach Gleichung (4.20) der Bias des Drehratensensors nicht berücksichtigt wird, wird sie so erweitert, dass dieser kompensiert wird. [84]

$$\begin{aligned} \dot{\hat{\theta}} &= \omega_{gyro} + k_p(\theta_{acc} - \hat{\theta}) \\ \dot{\hat{b}} &= -k_I(\theta_{acc} - \hat{\theta}) \end{aligned} \tag{4.21}$$

Folgende Grafik zeigt das Blockdiagramm des Filters sowohl für Gleichung (4.20) als auch (4.21).

Der Code für die Umsetzung dieses Filters, exemplarisch für den Roll-Winkel, befindet sich im Anhang in Kapitel *Ansatz nach Oliviera/Pascoal/Kaminer*.

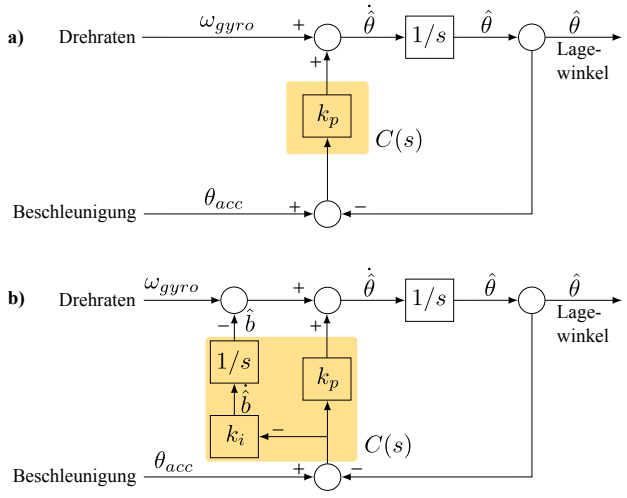


Abbildung 4.16: Umsetzung des Komplementärfilters nach [101] **a)** ohne Driftkompensation und **b)** mit Driftkompensation des Drehratensensors

Ansatz nach Colton

In Kapitel *Ansatz nach Colton* wird ein Filter beschrieben, wie es von Shane Colton in [113] dargelegt wird. Folgender Code zeigt dieses Filter exemplarisch für den Roll-Winkel.

```
1  #define DELTA_T     0.002 // Cycle Time 2 ms
2  #define FACTOR_A    0.98
3
4  void Complement2Update(float ax, float ay, float az, float gx, float gy, float gz)
5  {
6
7      float y_gyro_roll = gx;
8      float y_acc_roll = atan2(ay,az);
9
10     y_acc_roll = y_acc_roll/PI * 180.0;
11
12     roll = FACTOR_A*(roll+y_gyro_roll*0.005) + (1-FACTOR_A)*y_acc_roll;
13 }
```

Dabei lässt sich entsprechend den Formeln aus Kapitel *Ansatz nach Colton* die Zeitkonstante τ folgendermaßen berechnen:

$$\tau = \frac{a \cdot dt}{1-a} = \frac{0.98 \cdot 0.002\ sec}{0.02} \approx 0.1\ sec$$

Für Zeitperioden unter $0.1\ sec$ überwiegt der Einfluss der Winkelwerte, berechnet durch die Integration des Drehratensensors, und jene des Beschleunigungssensors werden weggefiltert. Für Zeitperioden über $0.l\ sec$ verhält es sich umgekehrt und den Werten des Beschleunigungssensors kommt mehr Gewichtung zu.

Ansatz mit Kalman-Filter

Für die Fluglageschätzung wurde basierend auf einem Kalman-Filter ein ähnlicher Ansatz wie in [115] gefunden. Im Prädiktionsschritt wird der Lagewinkel θ durch Integration des Drehratensensors propagiert. Da die Integration durch den Bias-Fehler des Drehratensensors mit einem

Drift behaftet ist, würde dies dazu führen, dass das Rauschverhalten des Systemmodells nicht rein normalverteilt ist. Abhilfe kann dadurch geschaffen werden, dass der Drift b_θ mitgeschätzt wird und damit dessen Eigenschaften im Modellverhalten enthalten sind. Dadurch wird der Drift des Drehratensensors automatisch korrigiert. Der Fluglagewinkel θ lässt sich durch Gleichung (4.22) beschreiben.

$$\theta_t = \theta_{t-1} - (\omega_{gyro} - b_\theta)dt \tag{4.22}$$

Aus dieser Gleichung lässt sich mit dem Zustandsvektor $x_t = \begin{bmatrix}\theta & b_\theta\end{bmatrix}^{\mathrm{T}}$ das lineare System (4.23) ableiten.

$$\begin{bmatrix}\theta \\ b_\theta\end{bmatrix}_t = \begin{bmatrix}1 & -dt \\ 0 & 1\end{bmatrix} \begin{bmatrix}\theta \\ b_\theta\end{bmatrix}_{t-1} + \begin{bmatrix}dt \\ 0\end{bmatrix} \omega_{t-1} \tag{4.23}$$

Als Messung z wird der Lagewinkel, berechnet durch die Komponenten des Beschleunigungssensors, herangezogen. Der Messschritt führt in diesem Ansatz dazu, dass der Drift des Drehratensensors korrigiert wird. Damit ergibt sich das Messmodell nach Gleichung (4.24).

$$z_t = \begin{bmatrix}1 & 0\end{bmatrix} x_t \tag{4.24}$$

Im Unterschied zum Ansatz nach [115] wird hier der Messschritt nicht in jedem Filterdurchgang ausgeführt. Während im Prädiktionsschritt alle *2 ms* propagiert wird, werden die Werte des Beschleunigungssensors über *200 ms* gemittelt. Erst nach dieser Zeit wird aus diesen gemittelten Komponenten des Beschleunigungssensors der Lagewinkel mithilfe der atan2-Funktion für die Messung berechnet und der durch das Systemmodell propagierte Schätzwert mit dem Messmodell korrigiert.

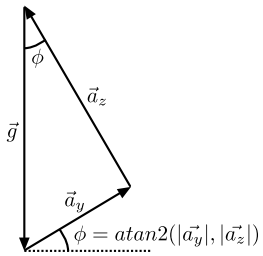


Abbildung 4.17: Trigonometrie des Beschleunigungssensors exemplarisch für den Roll-Winkel.

Da einzig eine Messung herangezogen wird, entfällt die Matrixinversion des Filteralgorithmus und wird zu einer skalaren Division. Dadurch ist die Anzahl der Floating-Point-Berechnungen gering. Noch mehr Floating-Point-Berechnungen können durch die Vorausberechnung der Kalman-Gain gespart werden. Wie in Kapitel *Rechenaufwand und Präzision* beschrieben, kann es dazu kommen, dass diese konvergiert, wenn die Rauschvektoren von System- und Messmodell konstant bleiben [118]. Dies ist hier der Fall. Deshalb kann im hier vorgestellten Ansatz durch die Wahl der Update-Funktion gewählt werden, ob man das Filter herkömmlich berechnet, oder eine vorausberechnete Kalman-Gain verwendet werden soll:

- `KalmanVelUpdate`: Berechnet den Kalman-Filteralgorithmus komplett nach den Formeln wie sie in Kapitel *Gleichungen Kalman-Filter* beschrieben werden.

- `KalmanVelUpdateFast`: Verzichtet im Kalman-Filteralgorithmus auf die Berechnung der Kovarianzmatrix des Schätzfehler P und die Kalman-Gain K. Stattdessen werden vorausberechnete Werte für die Kalman-Gain verwendet. Dabei kann, durch Einfügen des Defines `#define PRECOMP_KP` im Headerfile, zur Compile-Zeit entschieden werden, ob die Kalman-Gain im Initialisierungsschritt berechnet werden soll. Ansonsten können deren Werte in den Membervariablen `pre_comp_k11` und `pre_comp_k12` des Structs `KalmanVelObject` gesetzt werden.

Folgende Grafik zeigt den Ablauf der Funktion `KalmanVelUpdate` als Flussdiagramm. Der komplette Code des hier vorgestellten Ansatz findet sich im Anhang in Kapitel *Code Kalman-Filter zur Lagestützung*.

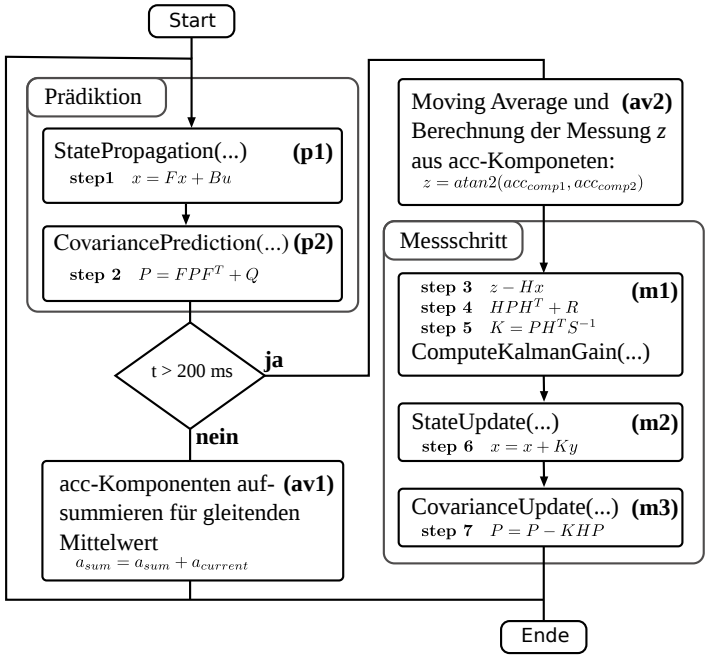


Abbildung 4.18: Flussdiagramm des Kalman-Filteralgorithmus zur Fluglageschätzung in sieben Schritten

Da viele Matrizen dünn besetzt sind fallen viele Berechnungen weg. Außerdem fällt beim Ausmultiplizieren des Algorithmus auf, dass sich manche Berechnungen wiederholen. Dazu kommt, dass die Kovarianzmatrix des Schätzfehlers P symmetrisch ist und es daher genügt, nur eine Seite der Diagonale zu berechnen. Deshalb können die Rechenschritte zur Durchführung des Filters optimiert und Floating-Point-Berechnungen eingespart werden, indem man sie entsprechend zusammenfasst. Die Berechnungen, welche zur Herleitung des Codes geführt haben, finden sich im Anhang in Kapitel *Matrix-Berechnungen*.

Vergleich der Lagefilter und Anwendung im Regler

Vergleich

Sowohl bei den Algorithmen nach Mahony/Madgwick, als auch beim Kalman-Filter muss eine $x \times x$-Matrix invertiert werden. Beide Ansätze sind zwar genau und auch bei Vibrationen noch robust, jedoch sind sie auch rechenintensiv. Ein Vorteil von Mahony/Madgwick ist, dass diese Algorithmen im Gegensatz zum Kalman-Ansatz über einen Winkel von 180 Grad hinaus funktionieren.

Wegen der hohen Rechenintensität wurden die etwas simpleren Komplementärfilteralgorithmen getestet. Diese Filter waren etwas schwerer zu parametrisieren und sind weniger genau als die Ansätze nach Mahony/Madgwick und dem Ansatz mit Kalman-Filter. Dafür ist deren Laufzeit geringer.

Einsatz im Regler

Es hat sich gezeigt, dass die hier vorgestellten Filter zur Fluglageschätzung für manche Anwendungen sehr gut funktionieren (Schlaf-Sensor, Gimbal), für andere Anwendungen (Regelung) jedoch, wegen deren hoher Laufzeit, nur mit Einschränkungen geeignet sind.

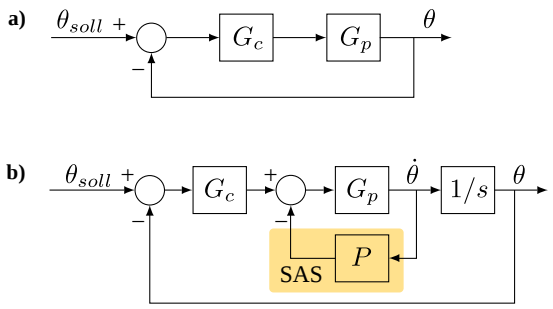


Abbildung 4.19: Konzept Fluglageregler

Desweiteren hat sich gezeigt, dass ein reiner PID-Regler einzig nach Fluglagewerten, wie in Abbildung 4.19 **(a)** dargestellt, sehr instabil ist. Aus diesem Grund wurde der Regler um eine Rückkopplung in Form eines sogenannten Stabilized Augmented Systems (SAS) erweitert wie es in Abblildung 4.19 **(b)** dargestellt ist. Dies ist ein schneller Regelkreis der den Flugroboter über die Drehratensensoren stabilisiert. In der Luftfahrzeugtechnik ist diese Form der Lageregelung über Drehratensensoren gängige Praxis [99]. Deshalb entstand im Rahmen dieser Arbeit die Idee eines zweistufigen Reglers, wie er in Abbildung 4.20 dargestellt ist. Ein innerer Regelkreis sollte in Form einer SAS umgesetzt werden und sehr schnell ablaufen. Da die Drehraten jedoch driften, sollte dieser innere Regelkreis durch einen langsameren äußeren Regelkreis unterstützt werden, der direkt auf die Fluglage regelt. Da dieser Regelkreis langsamer abläuft ist hier genug Zeit um den rechenintensiven Kalman-Filteralgorithmus durchführen zu lassen und mit dessen geschätzem Bias die Drehratensensoren zu korrigieren.

Die Idee nach Abbildung 4.20 wurde jedoch im Rahmen dieser Arbeit nicht mehr getestet, da

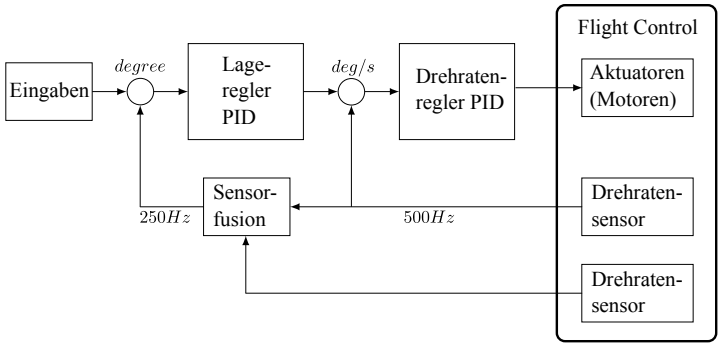


Abbildung 4.20: Alternatives Reglerkonzept

zuvor ein anderer Ansatz sehr gute Ergebnisse zeigte. Die Idee dieses Ansatzes ist es, ebenfalls mit einem schnellen Regler auf die Drehraten zu regeln. Für diesen Regler wurde eine sehr kurze Zykluszeit von nur *2 ms* gewählt. Über einen sehr langen Zeitraum von fast einer Sekunde werden nebenher die Werte des Beschleunigungssensors durch einen gleitenden Mittelwert erfasst und mithilfe dessen Werten die Bias-Werte des Drehratensensors korrigiert. [17]

4.3.2 Positionsschätzung

Da der Betrag der Standardabweichung von GNSS-Sensoren im Bereich von einigen Metern liegt, sind dessen Messwerte nicht geeignet für die Verwendung in der Positionsregelung des Flugroboters. Um ein besseres Signal zu erhalten wird ein extended Kalman-Filter verwendet, die Werte des globalen Navigationssatellitensystems Globales Navigationssatellitensystem (GNSS), mit den Werten der Inertialen Messeinheit (IMU), zu fusionieren. Dafür werden die Positionswerte mithilfe des Strapdownalgorithmus propagiert und mit den Werten von Magnetfeld und barometrischem Drucksensor, sowie den Positions- und Geschwindigkeitswerten des GPS-Sensors korrigiert.

Im OpenPilot-Projekt [105] wird ein extended Kalman-Filter zur Positionsschätzung vorgestellt, welches sehr jenem aus [96] ähnelt. Im Unterschied zum Ansatz in [96] wird hier der Drift des Drehratensensors $\vec{b}$ nicht als Random-Walk-Prozess modelliert, sondern im Systemmodell mitgeschätzt.

Zunächst wird das Systemmodell in der Form der nichtlinearen Zustandsgleichung nach Gleichung (4.25) hergeleitet.

$$\dot{\vec{x}} = f(\vec{x}, \vec{u}, \vec{w}) \tag{4.25}$$

mit dem Zustandsvektor $\vec{x} = (\vec{p}, \vec{v}, \vec{q}, \vec{b})^T$, dem Eingangsvektor $\vec{u} = (\vec{a}, \vec{\omega})^T$ sowie dem Rauschvektor $\vec{w} = (w_a, w_\omega)^T$. Die Nomenklatur der Elemente jener Vektoren $\vec{x}$, $\vec{u}$ und $\vec{u}$ wird in nachfolgender Tabelle gezeigt. [18]

[17] Dieser Regler ist nicht als Teil dieser Arbeit entstanden und stamt von Thorsten Reitmeier.

[18] Es sei hier nocheinmal daran erinnert, dass das erdfeste Koordinatensystem und das Inertialkoordinatensystem näherungsweise aufeinanderfallen.

Tabelle 4.1: Nomenklatur des Systemmodells des Kalman-Filters zur Positionsschätzung

Variable	Beschreibung
$\vec{p}$	Position im Navigationskoordinatensystems $\vec{p}_{eb}^{\,n}$
$\vec{v}$	Geschwindigkeit im Navigationskoordinatensystem $\vec{v}_{eb}^{\,n}$
$\vec{q}$	Fluglagevektor in Form eines Quaternions
$\vec{b}$	Bias des Drehratensensors $\vec{b}_\omega$
$\vec{a}$	Beschleunigung $\vec{a}_{ib}^{\,n}$ im Koordinatensystem
$\vec{\omega}$	Drehraten im körperfesten Koordinatensystem $\vec{\omega}_{ib}^{\,b}$
$\vec{w}_a$	Messrauschen des Beschleunigungssensors
$\vec{w}_\omega$	Messrauschen des Drehratensensors

Lässt man in Gleichung (4.16) alle Elemente weg, welche mit Null multipliziert werden und damit unerheblich sind und berücksichtigt zudem den Bias des Drehratensensors, wird sie zum dynamischen Modell für die Drehraten nach Gleichung (4.26).

$$\dot{q} = \frac{1}{2}\Omega(q)(\vec{\tilde{\omega}}_{ib}^{\,b} - \vec{b}_\omega) \tag{4.26}$$

mit

$$\Omega(q) = \begin{pmatrix} -q_1 & -q_2 & -q_3 \\ q_0 & -q_3 & q_2 \\ q_3 & q_0 & -q_1 \\ -q_2 & q_1 & q_0 \end{pmatrix}$$

Damit ergeben sich die Bewegungsgleichungen eines starren Körpers für 6 Freiheitsgrade, mithilfe des Strapdownalgorithmus aus Gleichung (4.19), aus den Werten der IMU. Diese hängen nicht von der Dynamik des bewegten Körpers ab und sind damit ohne Anpassung für jedes Fahrzeug anwendbar.

$$\begin{aligned} \dot{\vec{p}}_{eb}^{\,n} &= \vec{v}_{eb}^{\,n} \\ \dot{\vec{v}}_{ib}^{\,n} &= C_n^b \vec{\tilde{a}}_{ib}^{\,b} + g_l^n \\ \dot{\vec{q}} &= \frac{1}{2}\Omega q(\vec{\tilde{\omega}}_{ib}^{\,n} - \vec{b}_\omega) \end{aligned}$$

Nun lässt sich das **Systemmodell** entsprechend der nichtlinearen Zustandsgleichung $\vec{f}(\vec{x}, \vec{u}, \vec{w})$ nach Gleichung (4.27) aufstellen.

$$\vec{f}(\vec{x}, \vec{u}, \vec{w}) = \begin{pmatrix} \dot{\vec{p}} \\ \dot{\vec{v}} \\ \dot{\vec{q}} \\ \dot{\vec{b}}_\omega \end{pmatrix} = \begin{pmatrix} \vec{v} \\ C_n^b \cdot (\vec{\tilde{a}} + \vec{w}_a) + g_l^n \\ \frac{1}{2}\Omega(\vec{q} \cdot (\vec{\tilde{\omega}} + \vec{w}_\omega - \vec{b}_\omega) \\ \vec{b}_\omega \end{pmatrix} \tag{4.27}$$

Im Gegensatz zum Ansatz nach [105] wird hier der Bias des Drehratensensors $\vec{b}_\omega$ in das Systemmodell aufgenommen, um im Kalman-Filteralgorithmus mitgeschätzt zu werden.

Für die Messung werden Positions- und Geschwindigkeitswerte des GPS-Sensors, sowie Messwerte des Magnetfeldsensors und des barometrischen Drucksensors herangezogen. Damit lässt sich das **Messmodell** als Funktion des Zustandsvektors $\vec{h}(\vec{x})$ entsprechend Gleichung (4.28) formulieren.

$$\vec{z} = \vec{h}(\vec{x}) = \begin{pmatrix} \vec{p} \\ \vec{v} \\ \vec{h}^b \\ p \end{pmatrix} = \begin{pmatrix} \vec{p} \\ \vec{v} \\ C_b^n(\vec{q})\vec{h}^e \\ p \end{pmatrix} \tag{4.28}$$

mit dem Positionsvektor $\vec{p}$, dem Geschwindigkeitsvektor $\vec{v}$, dem Magnetfeldvektor $\vec{h}$ und dem skalarem Druckwert des barometrischem Höhensensors p (nicht zu verwechseln mit dem Positionsvektor $\vec{p}$).

Der extended Kalman-Filteralgorithmus erfordert eine Linearisierung des Systemmodells. Hierfür werden die Jacobi-Matrizen aus Gleichung (4.29) benötigt.

$$\begin{aligned} F &= \frac{\partial \vec{f}}{\partial \vec{x}} \\ G &= \frac{\partial \vec{f}}{\partial \vec{w}} \\ H &= \frac{\partial \vec{h}}{\partial \vec{x}} \end{aligned} \tag{4.29}$$

Die Partielle Ableitung von F ergibt sich nach Gleichung (4.30).

$$F = \left(\begin{array}{c|c|cc} 0_{3\times3} & I_{3\times3} & \multicolumn{2}{c}{0_{3\times3}} \\ \hline & & F_{vq} & 0_{3\times3} \\ 0_{10\times6} & & F_{qq} & F_{qb} \\ & & 0_{3x4} & I_{3\times3} \end{array}\right) \tag{4.30}$$

mit

$$\begin{aligned} F_{vq} &= \begin{pmatrix} F_{vq0} & F_{vq1} & F_{vq2} & F_{vq3} \\ -F_{vq3} & -F_{vq2} & F_{vq1} & F_{vq0} \\ F_{vq2} & -F_{vq3} & F_{vq0} & F_{vq1} \end{pmatrix} \\ F_{vq0} &= 2(q_0\tilde{a}_x - q_3\tilde{a}_y + q_2\tilde{a}_z) \\ F_{vq1} &= 2(q_1\tilde{a}_x + q_2\tilde{a}_y + q_3\tilde{a}_z) \\ F_{vq2} &= 2(-q_2\tilde{a}_x + q_1\tilde{a}_y + q_0\tilde{a}_z) \\ F_{vq3} &= 2(-q_3\tilde{a}_x - q_0\tilde{a}_y + q_1\tilde{a}_z) \end{aligned}$$

$$F_{qq} = \frac{1}{2}\begin{pmatrix} 0 & -(\tilde{\omega}_x - b_{\omega x}) & -(\tilde{\omega}_y - b_{\omega y}) & -(\tilde{\omega}_z - b_{\omega z}) \\ (\tilde{\omega}_x - b_{\omega x}) & 0 & (\tilde{\omega}_z - b_{\omega z}) & -(\tilde{\omega}_y - b_{\omega y}) \\ (\tilde{\omega}_y - b_{\omega y}) & -(\tilde{\omega}_z - b_{\omega z}) & 0 & (\tilde{\omega}_x - b_{\omega x}) \\ (\tilde{\omega}_z - b_{\omega z}) & (\tilde{\omega}_y - b_{\omega y}) & -(\tilde{\omega}_x - b_{\omega x}) & 0 \end{pmatrix}$$

und

$$F_{qb} = -\frac{1}{2}\Omega(\vec{q})$$

Die partielle Ableitung der Matrix G ergibt sich nach Gleichung (4.31).

$$G = \left(\begin{array}{c|c} 0_{3\times6} & \\ \hline 0_{3\times3} & C_b^n(\vec{q}) \\ \hline \Omega(\vec{q})/2 & 0_{4\times3} \\ \hline 0_{3\times6} & \end{array}\right) \tag{4.31}$$

Die partielle Ableitung der Messmatrix H ergibt sich nach Gleichung (4.32).

$$H = \left(\begin{array}{c|cc} I_{6\times6} & 0_{6\times7} & \\ \hline 0_{3\times6} & H_{hq} & 0_{3\times3} \\ \hline 0\ 0\ -1 \mid 0_{1\times3} & 0_{1\times7} & \end{array}\right) \tag{4.32}$$

mit

$$H_{hq} = \begin{pmatrix} H_{hq0} & H_{hq1} & H_{hq2} & H_{hq3} \\ H_{hq3} & -H_{hq2} & H_{hq1} & -H_{hq0} \\ -H_{hq2} & -H_{hq3} & H_{hq0} & H_{hq1} \end{pmatrix}$$

$$H_{hq0} = 2(q_0\tilde{h}_x + q_3\tilde{h}_y - q_2\tilde{h}_z)$$
$$H_{hq1} = 2(q_1\tilde{h}_x + q_2\tilde{h}_y + q_3\tilde{h}_z)$$
$$H_{hq2} = 2(-q_2\tilde{h}_x + q_1\tilde{h}_y - q_0\tilde{h}_z)$$
$$H_{hq3} = 2(-q_3\tilde{h}_x + q_0\tilde{h}_y + q_1\tilde{h}_z)$$

Nun lässt sich schließlich das extended Kalman-Filter herleiten. Zunächst muss das **System-** und das **Messmodell** der nichtlinearen Form aus den Gleichungen (4.25) und (4.28) in die linearisierte und diskretisierte Form nach Gleichung (4.33) geführt werden.

$$\begin{aligned} \vec{x}_t &= \Phi\vec{x}_{t-1} + \Gamma\vec{w}_{t-1} \\ \vec{z}_t &= H\vec{x}_t + \vec{v}_t \end{aligned} \tag{4.33}$$

Da das Systemmodell aus Gleichung (4.25) in Form einer kontinuierlichen Differentialgleichung $\vec{f}(\vec{x}, \vec{u}, \vec{w})$ vorliegt muss dieses diskretisiert werden. Hierfür werden die Näherungen aus Gleichung (4.34) verwendet.

$$\begin{aligned} \Phi &\cong I + FT \\ \Gamma &\cong GT \end{aligned} \tag{4.34}$$

Damit steht der komplette extended Kalman-Filteralgorithmus für die Positionsschätzung fest. Er wird in Abbildung 4.21 dargestellt. Die Integration zur Berechnung der Zustandsgleichung im ersten Filterschritt **(p1)**, wird mithilfe des Runge-Kutta-Algorithmus vierter Stufe [19] durchgeführt. Dabei wird Gleichung (4.27) herangezogen. Die partiellen Ableitungen zur Linearisierung des Systemmodells aus Gleichung (4.30) und (4.31) werden in **(p3)** durchgeführt. In **(p4)** wird, wie

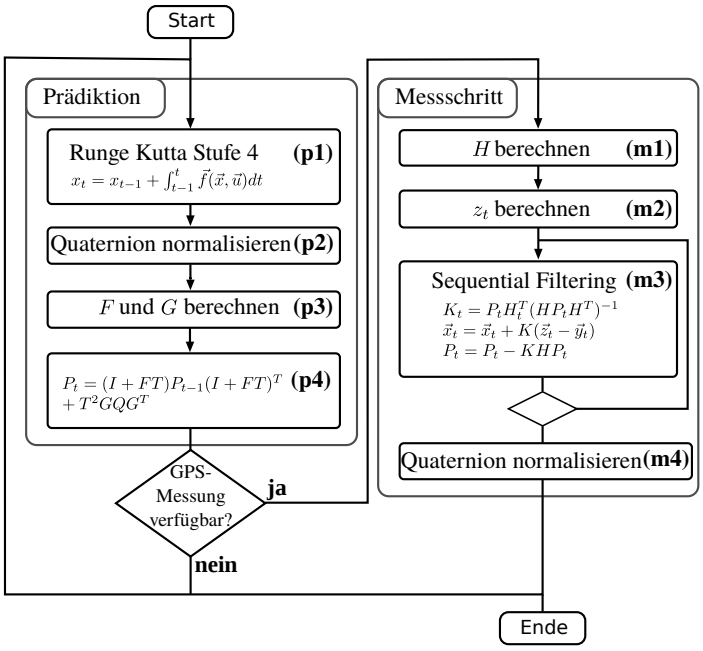


Abbildung 4.21: Extended Kalman-Filteralgorithmus zur Positionsschätzung nach [105]

im Kalman-Filter üblich, der aufgrund des Messrauschens anwachsende Schätzfehler in Form der Kovarianzmatrix P berechnet.

Für die Berechnung von H in **(m1)** wird Gleichung (4.32) herangezogen. Zur Berechnung des Ausgangswertes $\vec{z}_t$ in Schritt **(m2)** wird Gleichung (4.28) herangezogen. In Schritt **(m3)** wird ein wesentlicher Trick für die Reduktion des Rechenaufwandes angewendet: Um die im Messschritt des Kalman-Filteralgorithmus notwendige Matrixinversion zu verhindern, werden wie in [52] und [47] vorgestellt, die Messungen einzeln verarbeitet. Damit reduziert sich die besagte Matrixinversion zu skalaren Divisionen, einzeln durchgeführt für jede Messung. Dies ist zulässig, da das Messrauschen als korrelationsfrei angenommen wird.

4.4 Ausblick

Die Fluglagefilter für die Schätzung der Orientierung des UAVs, wie sie in dieser Arbeit vorgestellt wurden, kamen für die Fluglageregelung nicht zum Einsatz. Die Gründe hierfür werden in Kapitel *Einsatz im Regler* dargelegt. Trotzdem wurden aus deren Umsetzung wertvolle Informationen für die numerische Umsetzung des Kalman-Filters zur Positionsschätzung gewonnen.

Auch können die gewonnenen Erfahrungen im Bereich Kalman-Filterung durch dezentrale Konzepte wie Descentralized Kalman-Filter und Distributed Kalman-Filter, beschrieben in [26], in weiteren Anwendungen der Sensorvernetzung zukünftig ausgebaut werden.

Zu den in Kapitel *Positionsschätzung* gefundenen Algorithmen zur Positionsschätzung mit GNSS-Systemen und Trägheitsnavigation finden sich im Anhang noch keine Codes, da diese im zeitlichen

[19] Runge-Kutta-Integration vierter Stufe wird in [118] bündig beschrieben (S. 32ff).

Rahmen dieser Arbeit nicht mehr auf der Flughardware getestet werden konnten. Der nächste Schritt wird sein, diese auf der Flughardware umzusetzen.

Sobald gute Positionswerte gefunden werden, kann eine Positionsregelung implementiert werden. Der häufigste Ansatz für eine Regelung in X/Y-Richtung, ist ein zweistufiger Regler von Position und Geschwindigkeit, wie er beispielsweise in [93] oder [63] beschrieben wird. Im inneren Regelkreis wird die Geschwindigkeit und im äußerem Regelkreis die Position des UAVs geregelt. Die Eingangsgröße des Reglers ist die Position und die Ausgangsgröße sind die Stellwerte, die zu den gewünschten Fluglagewerten führen.

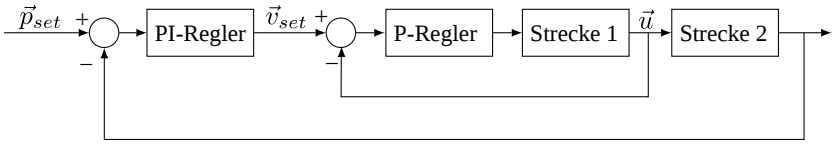


Abbildung 4.22: Konzept Positionsregelung nach [93]

Auch zur Höhenregelung sowie Höhen-Estimation finden sich Algorithmen in [93] und [63]. Die Update-Rate von Höhen- und Positionsregelung sollte weit unter jener der Fluglageregelung liegen.

KAPITEL 5

Protokolle

The human voice cannot mount up into these boundless solitudes.

Alberto Santos-Dumont,
My Airships - The Story of my Life
London, 1904

Inhalt: Es werden verschiedene Bibliotheken für den Austausch und das Abspeichern von Daten in kompakter, binärer Form zusammengetragen und verglichen. Zunächst werden Anforderungen beschrieben, die für den Datenaustausch in der UAS-Entwicklung am SappZ benötigt werden. Anschließend werden die Eigenschaften der geeigneten Bibliotheken für den Nachrichtenaustausch beschrieben und Vor- und Nachteile diskutiert. Am Ende wird eine Bibliothek als optimale Wahl für die Verwendung in diesem Rahmen ausgewählt und die Entscheidung begründet.

5.1 Einleitung

5.1.1 Ziel der Arbeit

Für den Austausch sowie das Speichern und Lesen von Daten müssen diese in binärer Form, kompakt vorliegen. Beispielsweise ist dies bei der drahtlosen Übertragung von Daten über - in der Bandbreite beschränkte - Systeme notwendig. Es wird auf unterschiedlichen Systemen gearbeitet. Auf dem UAV befindet sich ein eingebettetes System, während auf am Boden im Webbrowser oder auf Betriebssystemen gearbeitet wird. Deshalb sind für die jeweiligen Systeme Anbindungen an unterschiedliche Programmiersprachen notwendig. Die Codierung der Daten nennt man Serialisierung und die Rückgewinnung der einzelnen Datentypen aus den binären Daten Deserialisierung. Um für dieses wiederkehrende Problem der Handhabung strukturierter Sensor- oder Steuerdaten eine möglichst generische Lösung zu schaffen, wird hier auf frei verfügbare Standards gesetzt. In

dieser Arbeit soll ein Überblick über die zahlreichen verfügbaren Bibliotheken geschaffen werden. Hierfür werden eine selektierte Untermenge geeigneter Kandidaten untersucht und verglichen und deren jeweilige Stärken und Schwächen herausgestellt. Abschließend wird die Entscheidung getroffen, welche der verfügbaren Bibliotheken am besten geeignet ist.

5.1.2 Stand der Technik

Hintergrund

Für den Datenaustausch gibt es zahlreiche Formate, die für verschiedene Zwecke geeignet sind. **JSON** ist ein weit verbreitetes Dateiformat in Textform zum Datenaustausch. Es wird von vielen Systemen in nahezu allen Sprachen unterstützt. Da dieses Protokoll in Unicode codiert ist, ist das Footprint der übertragenen Daten sehr groß. Dies ist abträglich, wenn große Datenmengen mit niedriger Latenz übertragen werden sollen. Deshalb wurden von verschiedenen Stellen Formate entwickelt, deren Hauptentwurfskriterium Performanz ist (**Google Protocol Buffers**, **Cap'n Proto**, **MAVlink**, **Apache Thrift**).

Untersuchte Bibliotheken

Google Protocol Buffers wurde von Google für die eigene Infrastruktur entwickelt und erstmals 2008, in der BSD-Lizenz, der Open Source Community zur Verfügung gestellt. [37]

Cap'n Proto ist ein binäres Protokoll mit der selben Zielsetzung wie Google Protocol Buffers. Es wurde vom Hauptautor von Google Protocol Buffers, in der Version 2 - Kenton Varga - entwickelt. Laut Kenton Varga ist Cap'n Proto wesentlich schneller als Google Protocol Buffers. [125]

MAVlink ist eine leichtgewichtige Header-Only Bibliothek, die von Lorenz Meier zusammen mit Andrew Tridgell und James Goppert, im Rahmen des Pixhawk-Projekts [45] der ETH-Zürich entwickelt wurde. Die Software wurde 2009 erstmals unter der LGPL-Lizenz veröffentlicht. [135]

Apache Thrift ist ein Kommunikationsprotokoll, dessen Ursprung bei der Social-Network-Platform Facebook liegt. Es unterstützt sowohl binäre als auch textbasierte Protokolle und bietet die umfangreichste Sprachanbindung aller untersuchten Bibliotheken. Sogar *Erlang*, eine Websprache welche hohe Skalierbarkeit bietet (die derzeit häufig genutzte Alternative ist *JavaScript* in Verbindung mit *node.js*) wird unterstützt. [124]

Einige nicht betrachtete Bibliotheken

Einige Bibliotheken sind zwar verbreitet, doch auf deren Betrachtung wurde gänzlich verzichtet:

- Apache Etch
- **JSON**

Apache Etch [44] scheidet aus, da es nicht für den Einsatz auf einem eingebettetem System ausgelegt ist. **JSON** [76] überträgt Unicode-Zeichen. Dies macht den I/O-Stream zwar für Menschen

unmittelbar lesbar, dabei ist dieser unnötig speicherintensiv. Da die Bandbreite in Funkübertragungssystemen begrenzt ist und viele Daten möglichst schnell und fehlerfrei übertragen werden sollen, werden in diesem Rahmen stattdessen nur binäre Protokolle untersucht.

5.1.3 Rahmenbedingungen

Hintergrund

Abbildung 5.1 [1] zeigt den Aufbau der Funkkommunikation wie er derzeit für das UAS im Sensorik-Applikationszentrum geplant ist. Hier wird ersichtlich, dass verschiedenste Systeme miteinander kommunizieren. Auf dem UAS, findet ein Embedded System, auf der Host-Seite ein Tablet oder ein PC Verwendung. Zwischen allen Systemen müssen Nachrichten austauschen werden. Auf den verschiedenen Plattformen werden unterschiedliche Programmiersprachen auf unterschiedlichem Software-Level verwendet. Deshalb ist es essentiell, dass eine sinnvolle und einheitliche Gestalt der Daten für die Kommunikation gefunden wird. Nur so kann die Kommunikation von vorneherein nachhaltig gestaltet und hoher Zeitaufwand einer nachträglichen Anpassung jener Systeme verhindert werden.

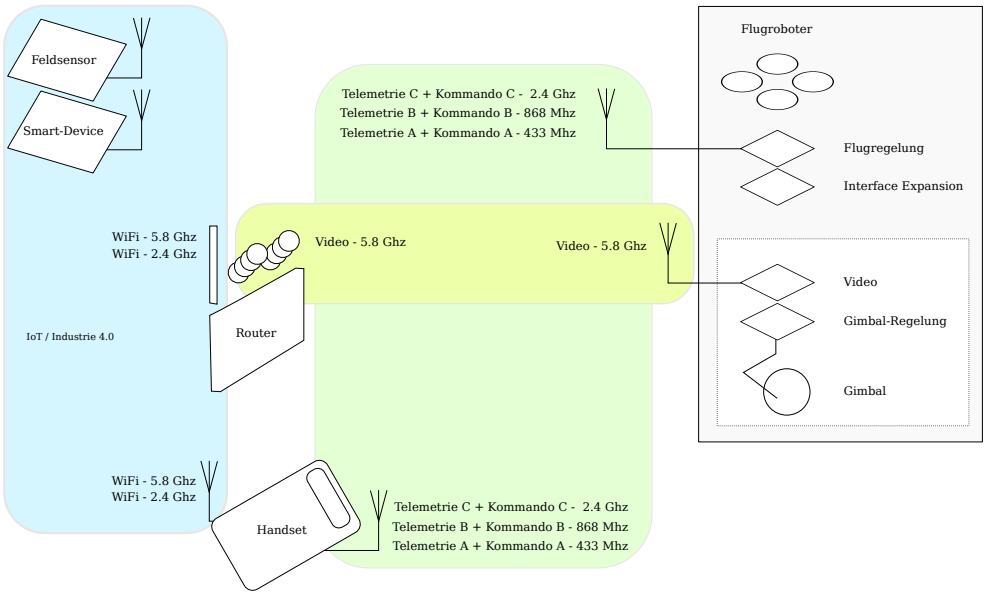


Abbildung 5.1: Funkkonzept des UAS

Anforderungen

Eine Anforderung ist die Möglichkeit der Serialisierung und Deserialisierung von Protokolldaten im Browser. Geplant ist eine Kommunikation über Websockets. Auf der Server-Seite wird un-

[1] Diese Abbildung stammt von Waldemar Sessler.

ter Umständen auf *JavaScript* in Verbindung mit Technologien wie *node.js* gesetzt [2]. Auch ist es zweckmäßig, dass die Bibliothek an einem Host-PC betrieben werden kann. Außerdem muss sie auf einem eingebettetem System (FlightControl) lauffähig sein. Hierfür muss die Bibliothek plattformunabhängig sein und verschiedene Programmiersprachen unterstützen.

Die Bandbreite von Funkübertragungssystemen ist begrenzt. Deshalb ist es wichtig, dass trotz eines kleinen Footprints des serialisierten, binären Datenstreams Funktionen zur Serialisierung und Deserialisierung möglichst geringe Laufzeiten aufweisen.

Alle Anforderungen sind in folgender Liste zusammengefasst:

- kompaktes binäres Potokoll
- schnelle Serialisierung/Deserialisierung
- generische Lösung
- flexible Protokollanpassung (Änderungen/Erweiterungen)
- schlanke Bibliothek, auch geeignet für den Betrieb auf eingebetteten Systemen
- Portabilität (Plattformunabhängigkeit)
- Verschiedene Sprachanbindungen (*C*, *C++*, *Python*, *JavaScript*)
- Webtechnologien (JavaScript, node.js)
- Lizenz erlaubt Nutzung

Auf diese Anforderungen werden die jeweiligen Bibliotheken im Folgenden untersucht und verglichen.

5.2 Vergleich der Bibliotheken

5.2.1 Diskussion

In folgender Tabelle sind die wichtigsten Eigenschaften der einzelnen Bibliotheken zusammengefasst. Neben den möglichen Sprachanbindungen aus Tabelle 5.2, wird auf diese Eigenschaften im Folgenden eingegangen.

Tabelle 5.1: Vergleich von Bibliotheken zur Serialisierung strukturierter Daten

	JSON	google protobufs	Cap'n Proto	MAVlink	Apache Thrift
Human Readable	*ja*	*nein*	*nein*	*nein*	*nein/JSON*
Protokoll	*UTF-8*	*binär*	*binär*	*binär*	*binär/JSON*
Lizenz	*MIT*	[77]	*BSD*	*LGPL*	*ASF 2.0*
Website	[76]	[37]	[125]	[135]	[124]
Checksumme	*keine*	*keine*	*keine*	*ITU X.25*	*keine*

[2] Eine endgültige Entscheidung über die verwendete Technologie ist an dieser Stelle noch nicht gefallen. Diese Problematik, wird umfassender in der Abschlussarbeit von Waldemar Sessler untersucht.

Google Protocol Buffers unterstützt zwar keine Checksummen, doch können diese bei Bedarf selbst als eigene Nachricht implementiert werden. In der neuesten Version werden Google Protocol Buffers von *Ruby* unterstützt, was dessen Einsatz mit Webtechnologien ermöglicht. In diesem Rahmen ist jedoch die Verwendung von *JavaScript* gefordert, was von Google Procol Buffers nicht nativ unterstützt wird. Ein Vorteil von Google Protocol Buffers ist, dass es sogenannte *optional fields* unterstützt, das sind Daten, die nur bei Bedarf im Binärstream mitgesendet werden können. Außerdem wird in der neuesten Version *Go* unterstützt, dessen Verwendung für zukünftige Webprojekte von Vorteil sein könnte.

Cap'n Proto ist eine verbesserte Version von Google Protocol Buffers, welche von einem der Entwickler von Google Protocol Buffers geschrieben wurde. Es besticht durch die gute Performanz bei der (De-)Serialisierung und die vielen unterstützten Programmiersprachen. Es wird nur von privaten Quellen gepflegt, was ein Nachteil sein kann.

Apache Thrift besticht durch seine hohe Anzahl an Features und die untersützung von vielen Programmiersprachen. Es könnte beispielsweise durch die Nutzung von *Erlang* oder *node.js* hervorragend in ein skalierbares Webprojekt eingebunden werden.

MAVlink wurde im Gegensatz zu den restlichen Protokollen speziell für die Benutzung in eingebetteten Systemen mit Funksystemen mit eingeschränkter Bandbreite entwickelt. Der binäre Daten-Stream ist dabei sehr schlank und enthält eine Prüfsumme [3]. Außerdem wird im Gegensatz zu Google Protocol Buffers auf dynamische Speicherverwaltung verzichtet, was es gut anwendbar auf Mikrocontrollern macht. Zwar unterstützt MAVlink *JavaScript* nicht nativ, trotzdem könnte der Vorteil der guten Einsetzbarkeit auf eingebetteten Systeme gegenüber den anderen Protokollen entscheidend sein.

Tabelle 5.2: Sprachanbindung ausgewählter Bibliotheken zur Serialiserung

	JSON	google protobufs	Cap'n Proto	MAVlink	Apache Thrift
C	○	○	○	●	●
C++	○	●	●	●	●
Python	○	●	○	●	●
Java	○	●	○	○	●
JavaScript	○	○	○	○	●

● Native Unterstützung in vollem Umfang
○ Nur bedingte Unterstützung durch verschiedene Autoren mit ungewisser Maintainance, möglicherweise mit eingeschränktem Funktionsumfang

Apache Thrift unterstützt neben den in Tabelle 5.2 genannte Programmiersprachen noch *PHP*, *Ruby*, *Erlang*, *Perl*, *Haskell*, *C#*, *Cocoa*, *node.js*, *Smalltalk*, *OCaml* und *Delphi*.

5.2.2 Fazit

Wie aus den Tabellen und den Ausführungen aus Kapitel *Diskussion* hervorgeht, decken alle Bibliotheken große Teile der Anforderungen ab. Einzig MAVlink sticht jedoch dadurch hervor, dass

[3] Die Prüfsumme ist die selbe, wie sie in den Standards *ITU X.25* und *SAE AS-4* (CRC-16-CCITT) verwendet wird. Sie wird in *SAE AS5669A* dokumentiert. [135]

es für den Gebrauch auf eingebetteten Systemen optimiert ist. Da letztlich alle anderen Anforderungen, wenn auch manche mit etwas Integrationsaufwand, mit MAVlink umgesetzt werden können, ist dies der entscheidende Vorteil, den MAVlink gegenüber den anderen Bibliotheken bietet. Alle anderen Bibliotheken sind auf die Umsetzung auf Betriebssystemen ausgelegt. Häufig werden zwar Lösungen für eingebettete Systeme angeboten, jedoch nie nativ und mit eingeschränkter Funktionalität, so dass diese eher als Notlösung erscheinen.

Das einzige Feature, das MAVlink nicht unterstützt ist eine reine *JavaScript*-Sprachanbindung. Zwar können *JavaScript*-Sources erzeugt werden, diese basieren jedoch auf *node.js* und sind damit für die serverseitige Nutzung vorgesehen. Dieser Nachteil kann jedoch mit vertretbarem Aufwand beseitigt werden. Es können entweder die vorhandenen *JavaScript*-Klassen so umgeschrieben werden, dass sie auch ohne *node.js* im Browser lauffähig sind, oder C-Funktionen nach *JavaScript* portiert werden. Dies kann beispielsweise mit Emscripten geschehen. Die Attraktivität dieser Lösung liegt darin, dass der Nachteil fehlender Unterstützung von 64-bit Integer in *JavaScript* umgangen würde.

Außerdem ist MAVlink durch **CAN** und Luftfahrtstandards aus *SAE AS-4* (*CRC-16-CCITT*) inspiriert. Auch das macht es für UAV-Anwendugnen attraktiv.

5.3 Implementierung

Zu Testzwecken wurden einfache Beispiele zur Anwendung von MAVlink in verschiedenen Sprachen implementiert. Es wurde ein Python-Skript geschrieben, mit dem MAVlink und einige notwendige Abhängigkeiten installiert werden können. Danach können die Beispiel-Codes kompiliert und ausgeführt werden. Die Installation und Ausführung jener Codes wird im Anhang in den Kapiteln *MAVlink installieren* und *Ausführen des Sample Codes* beschrieben.

Alle weiteren Infos, zur Benutzung und Funktionsweise von MAVlink, finden sich unter [135].

5.4 Ausblick

Für einen letzten Vergleich könnte noch der Vollständigkeit halber ein einfacher Benchmarktest in Bezug auf Serialisierungsgeschwindigkeit und Footprintgröße der Binärdatei durchgeführt werden. Hierfür war in diesem Rahmen keine Zeit. Trotzdem hat sich MAVlink, durch seine gute Integratierbarkeit in eingebette Systeme, als am besten geeignet herausgestellt.

KAPITEL 6

Schluss

6.1 Zusammenfassung

Kapitel *Hardwareentwicklung der Flugsteuerung*: Es wurde eine intelligente Flugsteuerung entwickelt. Diese bietet ein Echtzeitsystem für die Umsetzung von Flugalgorithmen und ein Linuxsystem für die Ansteuerung von Payload-Sensoren. Es hat sich gezeigt, dass ein Absturz des Linux-Systems dazu führt, dass auch das Echtzeitsystem abstürzt. Da es nicht möglich war, die sicherheitsrelevanten Algorithmen der Flugsteuerung vom störungsanfälligen Linux-System zu isolieren, wurden weitere Tests dieser Hardware übersprungen und es wurde dazu übergegangen die Flugalgorithmen auf den Motortreibern zu implementieren.

Kapitel *Simulation*: Eine GUI-basierende Simulationsumgebung für Quadrocopter wurde entwickelt. Die Entwicklung wurde so weit vorangetrieben, dass die grundsätzliche Physik eines Quadrocopters simuliert werden kann und diese nun für weitere Tests zur Verfügung steht. Um auch die Positionsregelung eines Quadrocopters testen zu können müssen noch Magnetfeld und GPS integriert werden.

Kapitel *Sensordatenfusion*: Hier werden Algorithmen für die Schätzung von Orientierung und Position hergeleitet. Dafür wird zunächst ein fundierter Literaturüberblick geliefert. Anschließend werden alle theoretischen Grundlagen die für die Herleitung der Algorithemen benötigt werden beschrieben. Anschließend werden zunächst verschiedene Algorithmen für die Orientierungsschätzung des Quadrocopters gezeigt. Diese werden verglichen und deren Einsatz im Regler wird diskutiert. Abschließend wird ein extended Kalman-Filter zur Positionsschätzung hergeleitet. Dieser konnte im Rahmen dieser Arbeit nicht mehr auf der Flugsteuerung umgesetzt und getestet werden.

Kapitel *Protokolle*: Da unterschiedlichste Systeme häufig über - in der Bandbreite beschränkte Funksysteme - miteinander kommunizieren müssen, wurde eine Methode gesucht den Austausch und das Speichern von Daten in geeigneter Weise zu gestalten. Hierfür wurden frei verfügbare Bibliotheken zusammengetragen und verglichen. Dabei wurde auf verschiedene Aspekte, wie Lauffähigkeit auf eingebetteten Systemen oder Unterstützung verschiedener Programmiersprachen eingegangen. Am Ende wurde jene Bibliothek ausgewählt, die am besten geeignet ist. Es wurden einige Sample-Codes mit Installationsskript geschrieben, um den Einstieg in die Bibliothek zu erleichtern.

6.2 Ausblick

Ein wesentlicher Punkt, der Verbesserungspotential verspricht, ist die funktionale Sicherheit des Flugroboters. Für die Verwendung von Fluggeräten mit sechs oder mehr Rotoren spricht, dass sie auch bei Ausfall eines Rotors immer noch stabil fliegen. Da im Rahmen dieser Arbeit jedoch festgestellt wurde, dass Fluggeräte mit nur vier Rotoren längere Flugzeiten aufweisen wird hier auf mehr Rotoren verzichtet. Eine Möglichkeit trotzdem einen Absturz zu verhindern ist die Anbringung eines Fallschirmes. Hier ist jedoch häufig eine relativ hohe Flughöhe notwendig, da kommerziell erhältliche Fallschirme bis zu 200 ms benötigen, bis diese sich öffnen. Eine weitere Möglichkeit wurde von der ETH Zürich in [97] vorgestellt. Die Funktionalität dieser Methode konnte für den Ausfall von einem oder zwei gegenüberliegenden Rotoren experimentell nachgewiesen werden. In der Simulation, konnte die Funktionalität dieser Methode sogar für den Ausfall von drei Rotoren nachgewiesen werden. Es wird bei Rotorausfall auf die Regelung des Yaw-Winkels verzichtet. Da durch den Rotorausfall das Drehmoment der Yaw-Axe nicht mehr ausgeglichen wird, beginnt der Quadrocopter in dieser Axe zu rotieren. Indem man immer dem Rotor mehr Schub zuweist, der bei der Rotationsbewegung des Quadrocopters gerade den tiefsten Punkt passiert. Regelt man auf diese Weise einen bestimmten Neigungswinkel, kann der Flugroboter gelenkt werden. Unter Umständen führt jedoch der Schubverlust eines oder mehrerer Rotoren dazu, dass sich der Quadrocopter zwangsweise im Sinkflug befindet. Hierzu findet sich ein Video der ETH Zürich unter der Rubrik *weiterführende Links* im Downloadsbereich der Website dieser Arbeit [58]. Diese beiden Möglichkeiten sollten evaluiert, erforscht und erprobt werden.

Desweiteren kann der Flugroboter als Sensorplatform in weiteren Versuchen für den Einsatz im Katastrophenschutz getestet werden. Neben der bereits bekannten Methode der Vermisstensuche mit Wärmebildkamera können innovative Technologien, wie sie beispielsweise im Rahmen einer Übung zum Thema *Suchen und Retten in Städten* (USAR, *Urban Search and Rescue*) des Technischen Hilfswerks praktisch erprobt wurden. Hier wurden mittels einer, an der Friedrich-Alexander Universität Erlangen-Nürnberg entwickelten Technologie, Verschüttete über die GSM-Signale von deren Mobiltelefonen geortet [133]. Eine weitere Methode, Verschüttete aufzuspüren ist die Detektion von Brustkorbbewegung durch Radarmessungen (Bioradar) [108], wie sie an der Universität Freiburg erforscht wurden. Dies sind vielversprechende Technologien, deren Nutzen für den Katastrophenschutz durch den mobilen, autonomen Einsatz zusammen mit dem Flugroboter gesteigert werden kann.

Die Flugzeit eines Quadrocopters ist durch den Akkubetrieb naturgemäß endlich. Da eine der Hauptanwendungen des Flugroboters im Katastrophenschutz der Schwebeflug zur Lageübersicht mit Kamera ist, ist es sinnvoll, für diesen Anwendungsfall über eine Verlängerung der Flugzeit, durch die Versorgung vom Boden aus nachzudenken. Eine Möglichkeit bieten hocheffektive Solarzellen, wie sie beispielsweise von den Firmen LaserMotive [21] oder L2W Eneregy [16] angeboten werden. Die Versorgung kann dann entweder drahtlos, über einen gerichteten Laser oder über die Einkopplung eines Lasers in leichte Glasfaserkabel erfolgen.

ANHANG A

Fertigungsunterlagen Flightcontrol

A.1 Stückliste

SENSORIK-APPLIKATIONSZENTRUM

Bill of Materials

Flight Control for Quadrocopter

Model Number & Rev	**Author**
FlightControl Rev A	Andreas Gschossmann

Contact
Andreas Gschossmann
Sensorik-Applikationszentrum (SappZ)
Telefon: +49 (941) 943 9848
Web: http://www.sappz.de

Part Number	Rev	Qty	Ref Des	Description	Distributor	Distributor Part Number
		1	C1	CAP, CERAMIC, 2n2, 50V, X7R, 0603	RS-COMPONENTS	464-6492
		2	C2, C22	CAP, TANTALUM, 10u, 6,3V	RS-COMPONENTS	464-9069
		1	C18	CAP, CERAMIC, 4u7, 25V, X5R, 0805	RS-COMPONENTS	723-6057
		1	C4	CAP, TANTALUM, 22u, 6,3V	RS-COMPONENTS	464-9249
		14	C3, C5, C6, C7, C11, C12, C13, C15, C16, C17, C19, C20, C21, C23	CAP, CERAMIC, 100n, X7R, 0603	RS-COMPONENTS	698-3263
		2	C8, C14	CAP, TANTALUM, 150UF, 10V, 6032	RS-COMPONENTS	547-9546
		1	C9	CAP, CERAMIC, 10n, X7R, 0603	RS-COMPONENTS	698-3257
		1	D1	TVS Diode, 3.3 V, SOD323	RS-COMPONENTS	770-4813
		1	IC1	LM1117, 800mA LDO Linear Regulator, 3.3V	RS-COMPONENTS	535-8635
		1	IC2	MPU-9150, 9 axis IMU Sensor	SCANTEC	MPU-9150
		1	IC3	SC18IS600IBS, SPI to I^2C-bus interface	MOUSER	771-SC18IS600IBS,128
		1	IC4	MS5611, pressure sensor	QUANTEC-NETWORKS	12637
		2	IC5, IC6	TPD2E001,low capacitance 2-channel ±15-kV, ESD-protection array for high speed data interfaces	RS-COMPONENTS	709-9050
		1	IC7	TPS2042, DUAL POWER-DISTRIBUTION SWITCHES	RS-COMPONENTS	423-090
		1	JP1, JP2, JP3			
		1	LED1	LED, blue, clear, 0805	RS-COMPONENTS	692-0953
		1	LED2	LED, bicolor, red/green, clear	RS-COMPONENTS	419-053
		1	P1	JST B2B-PH-K-S(LF)(SN), CONNECTOR HEADER, MALE	RS-COMPONENTS	546-8883

Part Number	Rev	Qty	Ref Des	Description	Distributor	Distributor Part Number
		14	R1, R2, R3, R4, R7, R8, R9, R17, R18, R20, R23, R26, R27, R43	RESISTOR, 10k Ohm, 0.1W, ±0.1%, 0603	RS-COMPONENTS	566-890
		6	R5, R6, R12, R19, R21, R24	RESISTOR, 0 Ohm, 0.1W, ±5%, 0603	RS-COMPONENTS	378-205
		7	R10, R11, R13, R30, R31, R33, R41	RESISTOR, 0 Ohm, 0.1W, ±5%, 0603, **np**	RS-COMPONENTS	378-205
		3	R14, R44, R45	RESISTOR, 180 Ohm, 0.1W, ±5%, 0603	RS-COMPONENTS	721-8097
		4	R15, R16, R25, R29	RESISTOR, 100 Ohm, 0.1W, ±0.1%, 0603	RS-COMPONENTS	566-339
		1	R22	RESISTOR, 0.020 OHM, 1/4W, 1%, 1210	RS-COMPONENTS	566-339
		1	R28	RESISTOR, 470 OHM, 0.1W, 0.1%, 0603	RS-COMPONENTS	566-547
		1	R32	RESISTOR, 100k OHM, 0.1W, 0.1%, 0603	RS-COMPONENTS	661-8767
		4	R34, R35, R36, R37	RESISTOR, 4k7 OHM, 0.1W, 0.1%, 0603	RS-COMPONENTS	661-8714
		1	R40	RESISTOR, 0 Ohm, 0.1W, 1%, 0805	RS-COMPONENTS	223-0146
		1	R42	RESISTOR, 0 Ohm, 0.1W, 1%, 0805, **np**	RS-COMPONENTS	223-0146
		1	SW1	B3U1000P, Ultra Small Tactile Switch	RS-COMPONENTS	419-867
		2	T1, T2, T3	FDV301N, N-Channel MOSFET, 25V, 0.5A, SOT-23	RS-COMPONENTS	354-4907
		1	T3	FDV301N	RS-COMPONENTS	354-4907
		3	J1, J2, J3	Hirose DF40C-100DS-0.4V(51)	MOUSER	798-DF40C100DS0.4V51
		1	U1	TPS2042BDGN	RS-COMPONENTS	423-090
		3	X1, X3, X9	53261-03	RS-COMPONENTS	542-7107
		1	X2	MICROSD_SD_MODE	RS-COMPONENTS	756-9578
		1	X2	Molex 49225-0821, Micro SD-Card Connector	RS-COMPONENTS	756-9578
		1	X4	USB-AB Connector	RS-COMPONENTS	515-2027
		1	X7	USB-A Connector	RS-COMPONENTS	718-7214

A.2 Schaltplan

SENSORIK-APPLIKATIONSZENTRUM

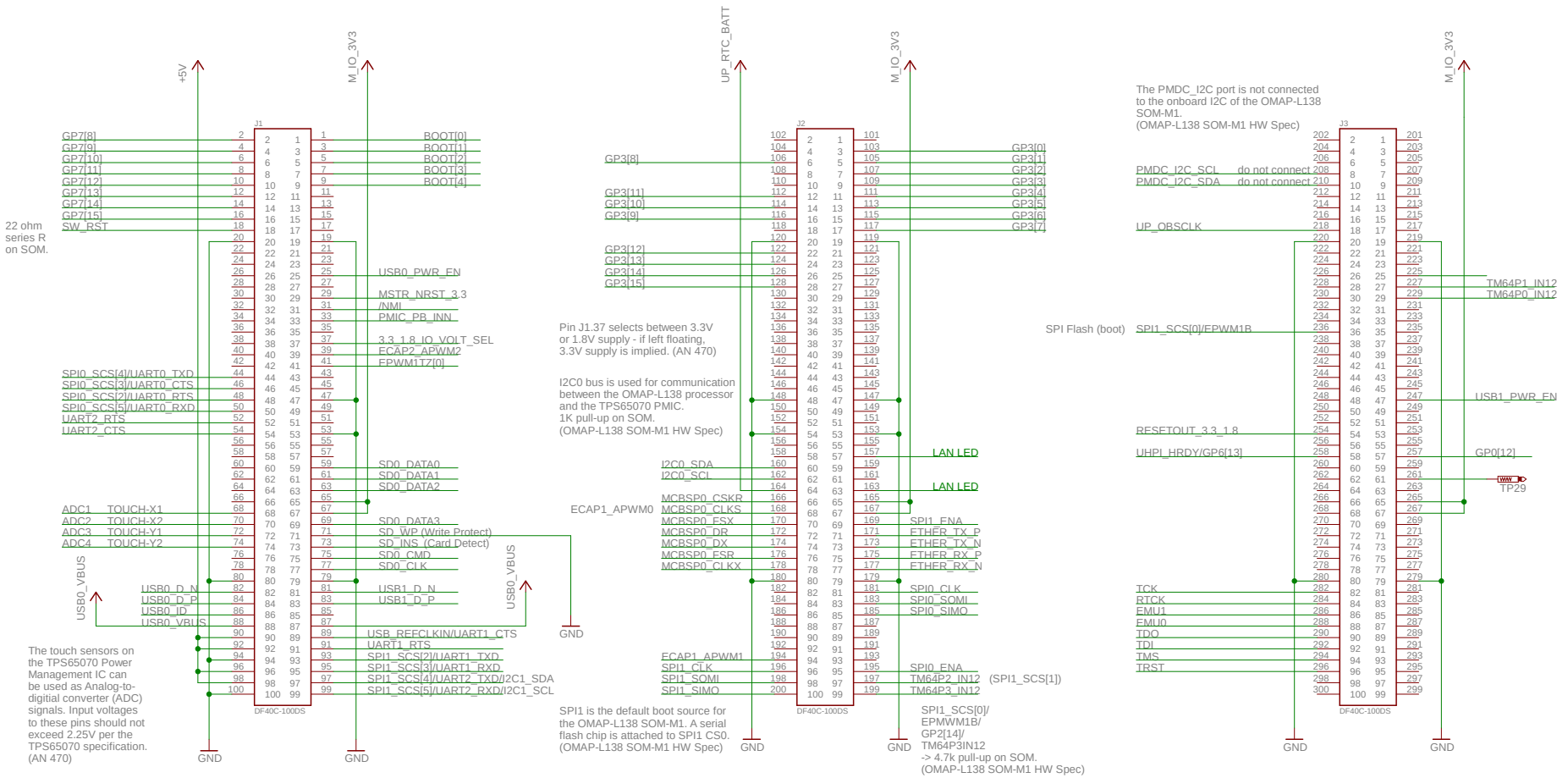
Schematic

Flight Control for Quadrocopter

Model Number & Rev	**Author**
FlightControl Rev A	Andreas Gschossmann

Contact
Andreas Gschossmann
Sensorik-Applikationszentrum (SappZ)
Telefon: +49 (941) 943 9848
Web: http://www.sappz.de

L138-SOM-CONNECTORS

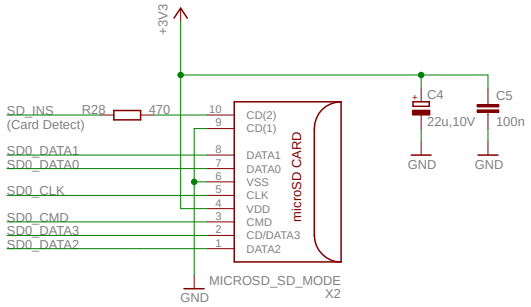


NOTE: Ethernet PHY must be removed from OMAP-L138 SOM-M1. Only then it can be used in this application.

	FC_REV_A
	12.12.2013 20:25:49
	Sheet: 1/8

SD/MMC

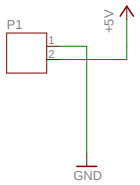
Micro SD-Card Connector



	FC_REV_A
	12.12.2013 20:25:49
	Sheet: 2/8

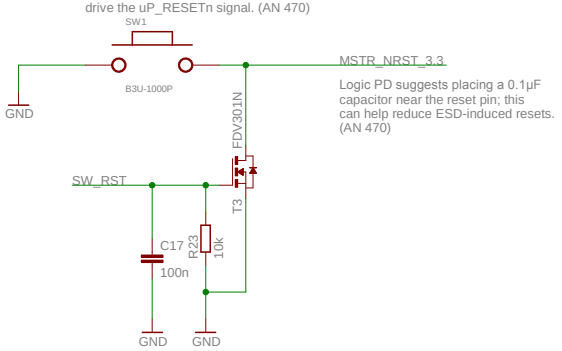
POWER

5V Input Connector



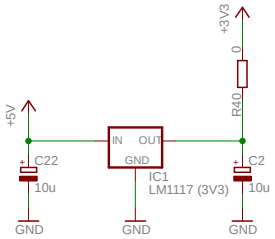
Reset

There is an internal pull-up resistor contained in the uP_RESETn signal; therefore, pull-up resistor or active drivers are not required to drive the uP_RESETn signal. (AN 470)



Logic PD suggests placing a 0.1µF capacitor near the reset pin; this can help reduce ESD-induced resets. (AN 470)

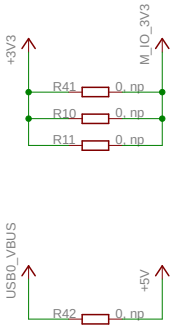
3,3V Supply Voltage Re-gulator

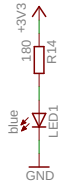


Same pin-out as National Semi-conductor's industry standard LM317. (LM1117 Datasheet)

To power the board by SOMs onboard 3,3 V output voltage connect M_IO_3V3 and +3V3 net.

Note: If you choose to use the M_IO_3V3 of the SOM it is URGENTLY NECESSARY to also remove R40.



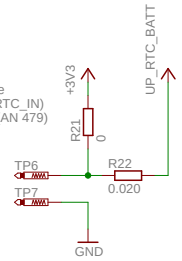


To power the board by USB connect USB0_VBUS and +5V net.

Note: If USB-power is chosen, do NOT connect external power supply!

Real-Time Clock (RTC)

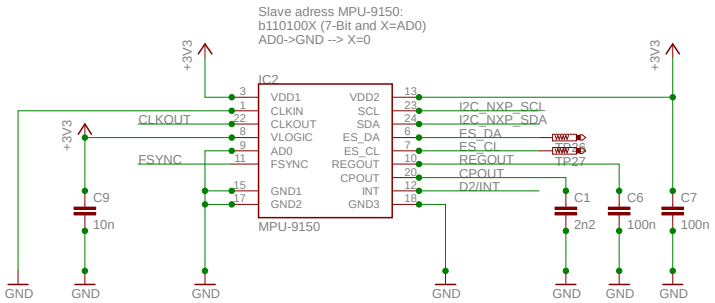
Ensure the real-time clock (RTC) rail (VRTC_IN) is always powered (AN 479)



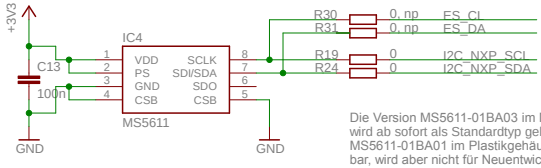
	FC_REV_A
	12.12.2013 20:25:49
	Sheet: 3/8

INERTIAL MEASUREMENT UNIT

Gyro, Accelerometer, Compass



Pressure Sensor

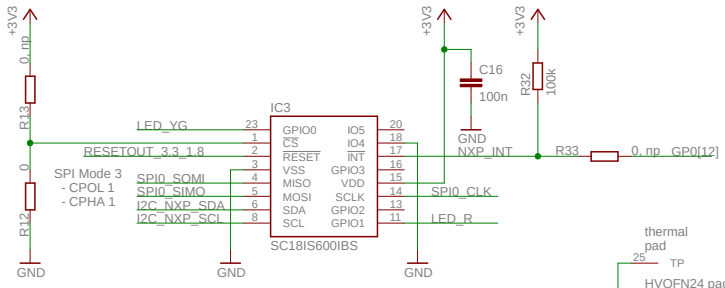


CSB in I2C mode defines the last bit in I2C address.

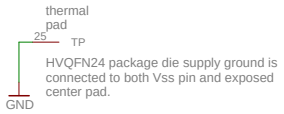
Protocol Select
PS high (VDD) -> I²C
PS low (GND) -> SPI

Die Version MS5611-01BA03 im Metallgehäuse wird ab sofort als Standardtyp geliefert. Der MS5611-01BA01 im Plastikgehäuse ist noch lieferbar, wird aber nicht für Neuentwicklungen empfohlen. Die technischen Parameter sind bei beiden Sensoren identisch.
(http://www.sander-electronic.de/be00067.html)

I²C-SPI Switch

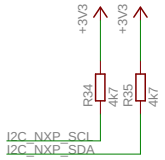


$\overline{\text{WAKEUP}}$/IO4: Wake up the SC18IS600 from the power-down mode. Pulled LOW by the host to wake-up from low power state. (datasheet SC18IS600)



HVQFN24 package die supply ground is connected to both Vss pin and exposed center pad.

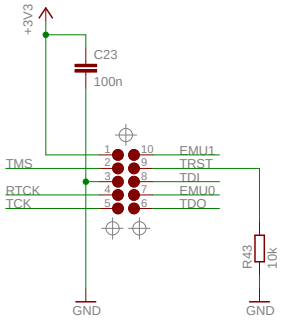
I²C Pull-Up Resistors



	FC_REV_A
	12.12.2013 20:25:49
	Sheet: 4/8

JTAG

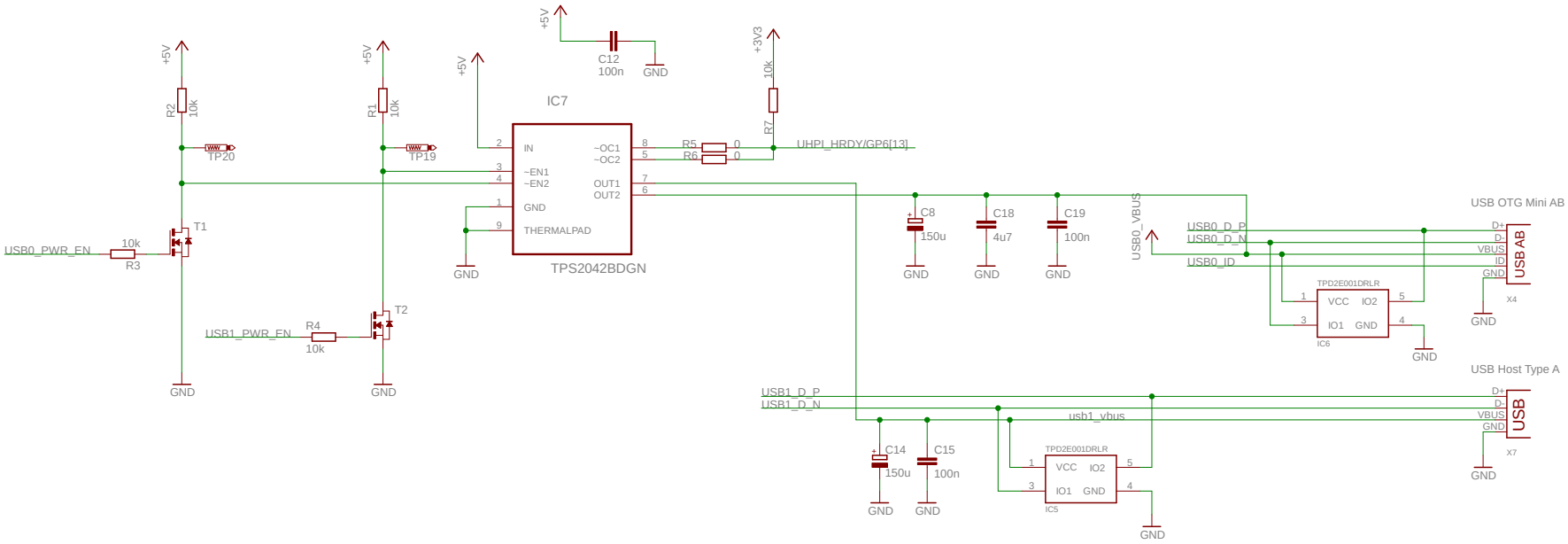
Tag-Connect JTAG Contact Pads



JTAG Emulator XDS100v2: Use TC2050-IDC-NL in combination with TC-C2000-M adaptor.

	FC_REV_A
	12.12.2013 20:25:49
	Sheet: 5/8

USB



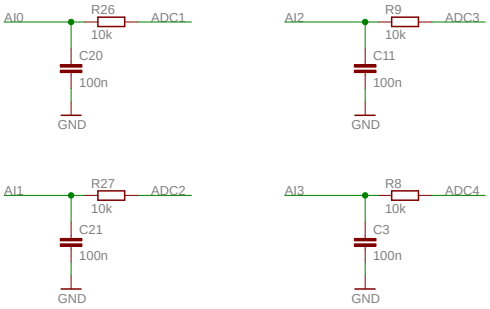
Verify that the USB differential pairs listed below have an impedance match of 90 ohms

- uP_USB0_DP/uP_USB0_DP
- uP_USB1_DP/uP_USB1_DP

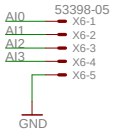
	FC_REV_A
	12.12.2013 20:25:49
	Sheet: 6/8

Connectors

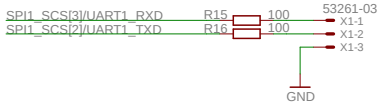
ADC-Inputs



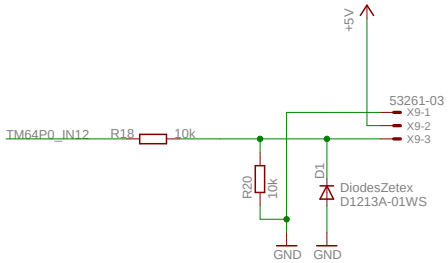
Analog Inputs



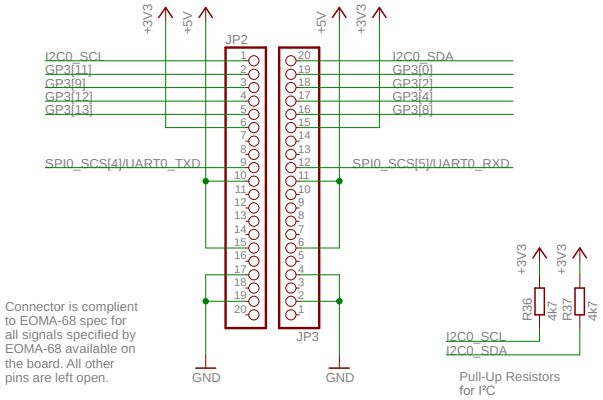
Debug Uart-Port



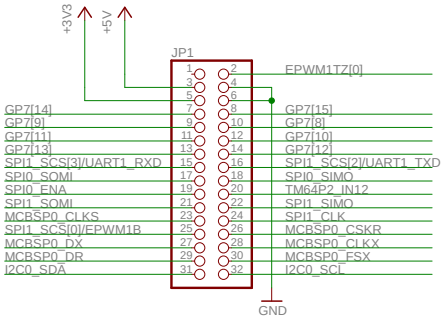
PPM Receiver



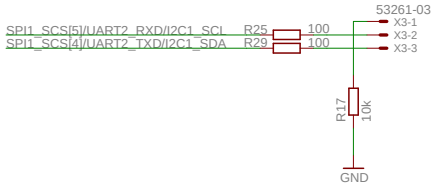
EOMA-68-Connector



Breakout-Connector



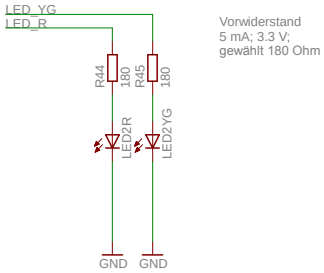
I2C Motor-driver



	FC_REV_A
	12.12.2013 20:25:49
	Sheet: 7/8

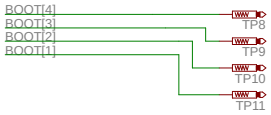
LEDs & Testpins

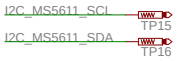
USER LED (Bi-Color)

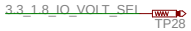


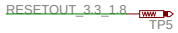
Vorwiderstand
5 mA; 3.3 V;
gewählt 180 Ohm

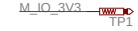
Boot-Optionen

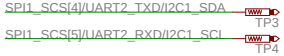


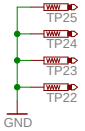












	FC_REV_A
	12.12.2013 20:25:49
	Sheet: 8/8

A.3 Layout

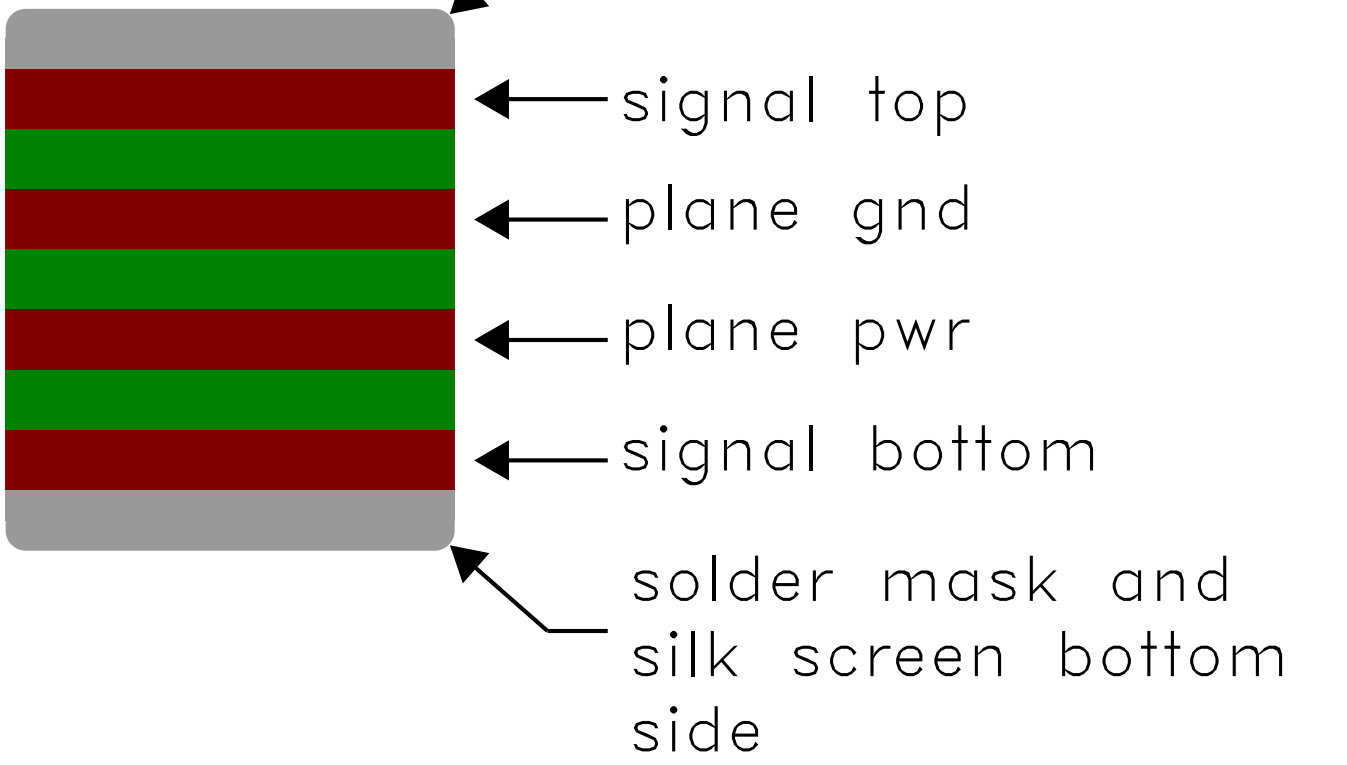
SENSORIK-APPLIKATIONSZENTRUM

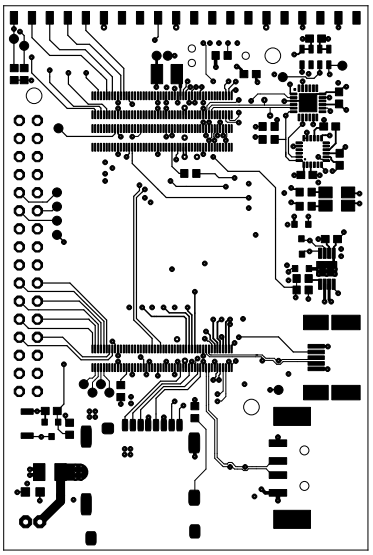
Layout

Flight Control for Quadrocopter

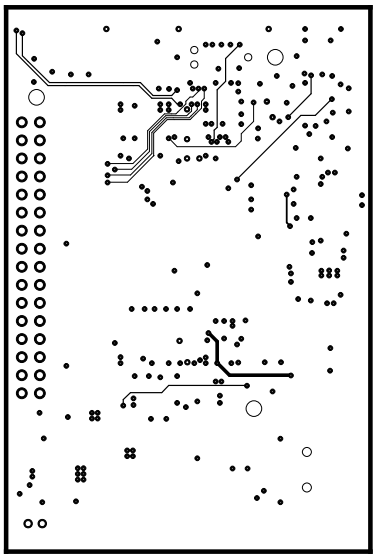
Model Number & Rev	Author
FlightControl Rev A	Andreas Gschossmann

Contact
Andreas Gschossmann
Sensorik-Applikationszentrum (SappZ)
Telefon: +49 (941) 943 9848
Web: http://www.sappz.de

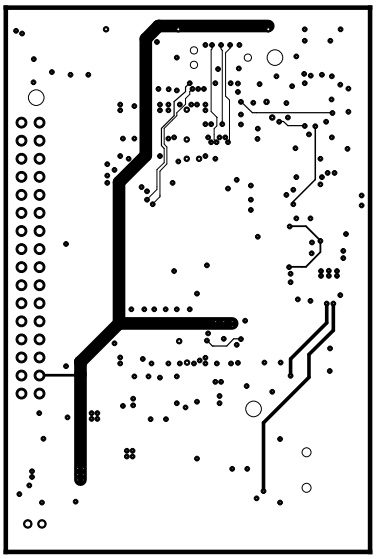

solder mask and silk screen top side
signal top
plane gnd
plane pwr
signal bottom
solder mask and silk screen bottom side



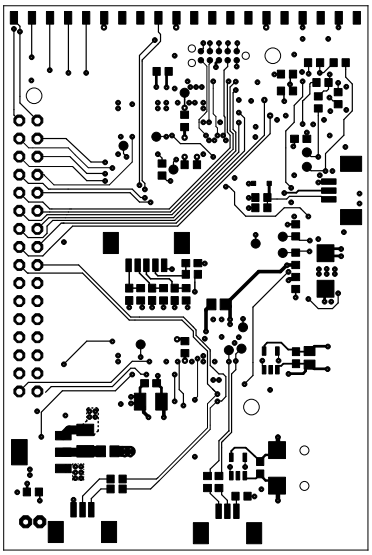
signal top



plane gnd

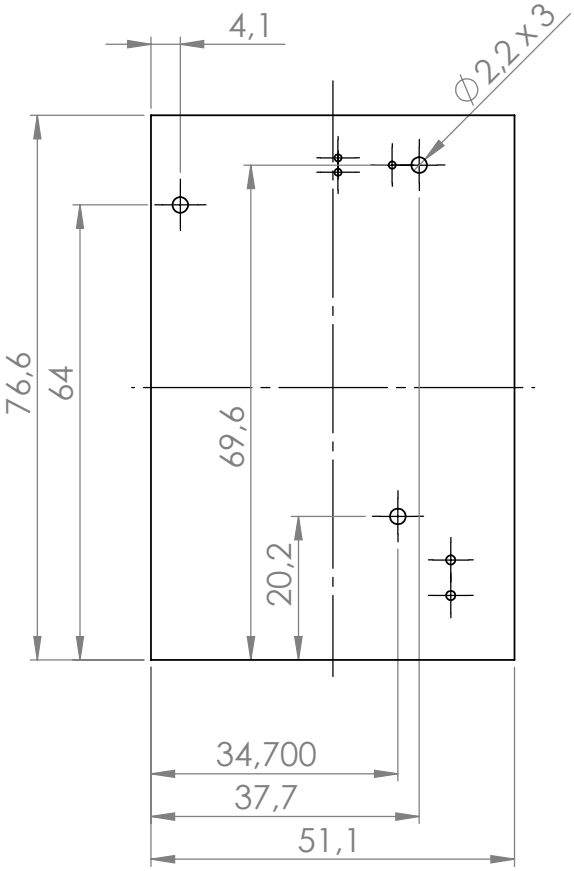


plane pwr



signal bottom

SENSORIK-APPLIKATIONSZENTRUM



drill template

ANHANG B

DeadalusSim

B.1 Installation

B.1.1 Gazebosim installieren

Gazebo kann entweder über die Paketquellen installiert werden [1].

```
sudo sh -c 'echo "deb http://packages.osrfoundation.org/gazebo/ubuntu precise
main" > /etc/apt/sources.list.d/gazebo-latest.list'
wget http://packages.osrfoundation.org/gazebo.key -O - | sudo apt-key add -


sudo apt-get update
sudo apt-get install gazebo3
```

oder von gazebo.org heruntergeladen werden. Die hier vorgestellten Codes wurden mit Version 3.0 von Gazebo getestet.

B.1.2 Playstation 3 Controller Treiber installieren

Zunächst muss der *QtSixA*-Treiber [131] installiert werden. Dieser bietet die Möglichkeit PS3 Hardware an eine Linux-Maschine anzuschließen:

```
sudo add-apt-repository ppa:falk-t-j/qtsixa
sudo apt-get update
sudo apt-get install qtsixa
```

Nun kann der PS3 Controller bentutzt werden:

```
# plug controller into machine (via wire)
sudo sixpair
# unplug controller
sixad --start
# hold PS button
```

Wird ein PS3 Controller angeschlossen, kann man das simulierte UAV damit steuern. Falls kein PS3 Controller gefunden wird, wird eine Funktion mit vordefinierten Stimuli aufgerufen.

[1] Alle Schritte, die für die Kompilierung und Installation, der hier entwickelten Simulation notwendig sind wurden unter Ubuntu 12.04 getestet.

B.1.3 DeadalusSim kompilieren und starten

Herunterladen und enpacken:

```
mkdir mysim && cd mysim
scp -P 4444 gsa39665@hps.hs-regensburg.de:html/files/deadalussim_v0.1.2015.tar.bz2 .
tar xvjf deadalussim_v0.1.2015.tar.bz2
```

Kompilieren mit

```
cd DeadalusSim/source
cd build
sudo cmake ..
sudo make
cd ..
```

Ausfuehren mit

```
cd source
gazebeo gazebo_models/quadrotor.sdf
```

oder einfach durch aufruf des Skripts in diesem Ordner

```
sudo ./startup_sim.sh
```

Falls die Steuerung mit Playstation3 Controller gewünscht ist müssen vorher in einer eigenen Konsole folgende Befehle ausgeführt werden:

```
# plug controller into machine (via wire)
sudo sixpair
# unplug controller
sixad --start
# hold PS button
```

B.2 Class Documentation

Um Seiten zu sparen wurde die Klassendokumentation nur online abgelegt, der QR-Code aus Abbildung B.1 führt zu den Online-Quellen.

Abbildung B.1: Link zur Klassendokumentation der DeadalusSim-Software [59]

ANHANG C

Codes Sensordatenfusion

C.1 Komplementär-Filter zur Lageschätzung

C.1.1 Code Ansatz Mahony

Header-Datei

```
#ifndef _MAHONY_H
#define _MAHONY_H

typedef struct
{
  float twoKp;
  float twoKi;

  float q0;    //< set point
  float q1;    //< error
  float q2;    //< previous error
  float q3;    //< integral

  float integralFBx;
  float integralFBy;
  float integralFBz;  // integral error terms scaled by Ki

} MahonyObject;

void mahonyInit(MahonyObject* mahony);
void mahonyUpdate(MahonyObject* mahony, float gx, float gy, float gz, float ax, float ay, float az, float dt);
#endif
```

C-Datei

```
#include "Mahony_Lage.h"
#include "mainvars.h"

#include <math.h>

#define M_PI 3.14159265358979323846

#define TWO_KP_DEF  (2.0f * 0.4f) // 2 * proportional gain
#define TWO_KI_DEF  (2.0f * 0.001f) // 2 * integral gain

float invSqrt(float x);

float ConvertToDegree(float value)
{
    return (value/PI * 180.0);
}

//---------------------------------------------------------------------------------------------------
```

```
20  const float Epsilon = 0.0009765625f;
21  const float Threshold = 0.5f - Epsilon;
22
23  void CalculateAngles(MahonyObject* mahony)
24  {
25      float XY = mahony->q0 * mahony->q1;
26      float ZW = mahony->q2 * mahony->q3;
27
28      float TEST = XY + ZW;
29
30      if ( (TEST < -Threshold) || (TEST > Threshold) )
31      {
32          float sign;
33          if (TEST < 0.0) sign=-1.0; else sign=1.0;
34          pitch = sign * 2.0 * (float)atan2(mahony->q0, mahony->q3);
35          yaw   = sign * PI/2;
36          roll  = 0;
37      }
38      else
39      {
40
41          float XX = mahony->q0 * mahony->q0;
42          float XZ = mahony->q0 * mahony->q2;
43          float XW = mahony->q0 * mahony->q3;
44
45          float YY = mahony->q1 * mahony->q1;
46          float YW = mahony->q1 * mahony->q3;
47          float YZ = mahony->q1 * mahony->q2;
48
49          float ZZ = mahony->q2 * mahony->q2;
50
51          pitch = atan2(2 * YW - 2 * XZ, 1 - 2 * YY - 2 * ZZ);
52          //yaw   = atan2(2 * XW - 2 * YZ, 1 - 2 * XX - 2 * ZZ);
53          roll  = asin(2 * TEST);
54
55      }//if
56
57      yaw   = ConvertToDegree(yaw);
58      pitch = ConvertToDegree(pitch);
59      roll  = ConvertToDegree(roll);
60
61  }
62
63  void mahonyInit(MahonyObject* mahony)
64  {
65      mahony->twoKp = TWO_KP_DEF;     // 2 * proportional gain (Kp)
66      mahony->twoKi = TWO_KI_DEF;     // 2 * integral gain (Ki)
67
68    mahony->q0 = 0.0001f;
69    mahony->q1 = 0.0001f;
70    mahony->q2 = 0.0001f;
71      mahony->q3 = 0.0001f;
72  }
73
74  void mahonyUpdate(MahonyObject* mahony, float gx, float gy, float gz, float ax, float ay, float az, float dt)
75  {
76    float recipNorm;
77    float halfvx, halfvy, halfvz;
78    float halfex, halfey, halfez;
79    float qa, qb, qc;
80
81    // Compute feedback only if accelerometer measurement valid (avoids NaN in accelerometer normalisation)
82    if(!((ax == 0.0f) && (ay == 0.0f) && (az == 0.0f)))
83    {
84      // Normalise accelerometer measurement
85      recipNorm = invSqrt(ax * ax + ay * ay + az * az);
86      ax *= recipNorm;
87      ay *= recipNorm;
88      az *= recipNorm;
89
90      // Estimated direction of gravity and vector perpendicular to magnetic flux
91      halfvx = mahony->q1 * mahony->q3 - mahony->q0 * mahony->q2;
92      halfvy = mahony->q0 * mahony->q1 + mahony->q2 * mahony->q3;
93      halfvz = mahony->q0 * mahony->q0 - 0.5f + mahony->q3 * mahony->q3;
94
95      // Error is sum of cross product between estimated and measured direction of gravity
96      halfex = (ay * halfvz - az * halfvy);
97      halfey = (az * halfvx - ax * halfvz);
98      halfez = (ax * halfvy - ay * halfvx);
99
100     // Apply proportional feedback
101     gx += mahony->twoKp * halfex;
102     gy += mahony->twoKp * halfey;
103     gz += mahony->twoKp * halfez;
104   }
105
106   // Integrate rate of change of quaternion
107   gx *= (0.5f * dt);   // pre-multiply common factors
108   gy *= (0.5f * dt);
109   gz *= (0.5f * dt);
110   qa = mahony->q0;
```

```
111     qb = mahony->q1;
112     qc = mahony->q2;
113     mahony->q0 += (-qb * gx - qc * gy - mahony->q3 * gz);
114     mahony->q1 += (qa * gx + qc * gz - mahony->q3 * gy);
115     mahony->q2 += (qa * gy - qb * gz + mahony->q3 * gx);
116     mahony->q3 += (qa * gz + qb * gy - qc * gx);
117
118     // Normalise quaternion
119     recipNorm = invSqrt(mahony->q0 * mahony->q0 + mahony->q1 * mahony->q1 + mahony->q2 * mahony->q2 + mahony->q3 * mahony->q3);
120     mahony->q0 *= recipNorm;
121     mahony->q1 *= recipNorm;
122     mahony->q2 *= recipNorm;
123     mahony->q3 *= recipNorm;
124
125     // Calculate angles from quaternions
126     CalculateAngles(mahony);
127 }
128
129 //---------------------------------------------------------------------------------------------------
130 // Fast inverse square-root
131 // See: http://en.wikipedia.org/wiki/Fast_inverse_square_root
132 float invSqrt(float x)
133 {
134     float halfx = 0.5f * x;
135     float y = x;
136     long i = *(long*)&y;
137     i = 0x5f3759df - (i>>1);
138     y = *(float*)&i;
139     y = y * (1.5f - (halfx * y * y));
140     return y;
141 }
```

C.1.2 Code Ansatz Oliviera/Pascoal/Kaminer

```
1  #define K_P         0.02
2  #define K_I         0.001
3
4  #define DELTA_T     0.002 // Cycle Time 2 ms
5
6  void Complement1Update(float ax, float ay, float az, float gx, float gy, float gz)
7  {
8      float dotRoll;
9      float dotBiasRoll;
10
11     float y_gyro_roll = 0.0;
12     float y_acc_roll = 0.0;
13
14     // measurement of gyro
15     y_gyro_roll = gx;
16
17     // measurment of acc
18 #ifdef  SMALL_ANGLE_APPROX
19     y_acc_roll = ay/az;
20 #else // no small angle approximation
21     y_acc_roll = atan2(ay,az);
22 #endif
23
24     // Convert angle calculated
25     // by acc to degrees
26     y_acc_roll = y_acc_roll/PI * 180.0;
27
28     dotRoll = y_gyro_roll - biasRoll + K_P*(y_acc_roll - roll);
29
30     dotBiasRoll = -K_I*(y_acc_roll - roll);
31     biasRoll = biasRoll + dotBiasRoll*DELTA_T;
32
33     roll = roll + dotRoll*DELTA_T;
34 }
```

C.2 Kalman-Filter zur Lageschätzung

C.2.1 Code Kalman-Filter zur Lagestützung

Header-Datei

```
/* ----------------------------------------------------------------------------
 *
 *
 *  ____                  _____     _
 * / ___|  __ _ _ __  _ __|__  /  __| | ___
 * \___ \ / _` | '_ \| '_ \ / /  / _` |/ _ \
 *  ___) | (_| | |_) | |_) / /_ | (_| |  __/
 * |____/ \__,_| .__/| .__/____(_)__,_|\___|
 *             |_|   |_|
 *
 *
 *
 * Author: Andreas Gschossmann
 * Email:  andreas.gschossmann@hs-regensburg.de
 *
 * kalman_vel.h - Kalman Filter for attitude estimation
 * The algorithm estimates the attitude and the bias of
 * the gyro using the gyro for the system model and the
 * accelerometer for the measurement model.
 *
 * ----------------------------------------------------------------------------*/

#ifndef KalmanVel_h
#define KalmanVel_h

#define Q1 0.0f//0.003f
#define Q2 0.1f//0.001f
#define R1 1.0f//0.008f

#define DT 0.002

#define MSR_CNT 20

#define MOVING_AVG_ACC
//#define PRECOMP_KP
//#define SMALL_ANGLE_APPROX

typedef struct
{
    // public member variables
    float x1, x2;

    // private member variables
    float p11, p12, p21, p22;           // Covariance Matrix
    float q1,q2;                                // Believe in System
    float r1;                                       // Believe Measurement
    float dt;                                       // Delta t

    float y;                                        // Innovation
    float k11, k12;                             // Kalman Gain

    float acc_comp1_sum;
    float acc_comp2_sum;

    float pre_comp_k11;
    float pre_comp_k12;

    int measure_cnt;

} KalmanVelObject;

// public functions
void KalmanVelInit(KalmanVelObject *data);
void KalmanVelUpdate(KalmanVelObject *data, float acc_comp1, float acc_comp2, float omega);
void KalmanVelUpdateFast(KalmanVelObject *data, float acc_comp1, float acc_comp2, float omega);

// private functions
// step 1
void StatePropagation(KalmanVelObject *data, float u);
// step 2
void CovariancePrediction(KalmanVelObject *data);
// step 3-5
void ComputeKalmanGain(KalmanVelObject *data, float z);
// step 6
void StateUpdate(KalmanVelObject *data);
// step 7
void CovarianceUpdate(KalmanVelObject *data);

void PreCompKP(KalmanVelObject *data);
```

```
80
81
82  #endif
```

C-Datei

```
1   /* ----------------------------------------------------------------------------
2    *
3    *
4    *  ____                    _____     _
5    * / ___|  __ _ _ __  _ __ |__  /  __| | ___
6    * \___ \ / _` | '_ \| '_ \  / /  / _` |/ _ \
7    *  ___) | (_| | |_) | |_) |/ /_ | (_| |  __/
8    * |____/ \__,_| .__/| .__/____(_)__,_|\___|
9    *             |_|   |_|
10   *
11   *
12   *
13   * Author: Andreas Gschossmann
14   * Email:  andreas.gschossmann@hs-regensburg.de
15   *
16   * kalman_vel.c - Kalman Filter for attitude estimation
17   * The algorithm estimates the attitude and the bias of
18   * the gyro using the gyro for the system model and the
19   * accelerometer for the measurement model.
20   *
21   * ----------------------------------------------------------------------------*/
22
23  #include "kalman_vel.h"
24  #include "mainvars.h"
25  #include "FlashDtaValues.h"
26  #include <math.h>
27
28  void KalmanVelInit(KalmanVelObject *data)
29  {
30      int i;
31
32      data->x1 = 0.0f;
33      data->x2 = 0.0f;
34
35      data->p11 = 1000.0f;
36      data->p12 = 0.0f;
37      data->p21 = 0.0f;
38      data->p22 = 1000.0f;
39
40      data->q1 = Q1;
41      data->q2 = Q2;
42
43      data->r1 = R1;
44      data->dt = DT;
45
46      data->acc_comp1_sum = 0.0f;
47      data->acc_comp2_sum = 0.0f;
48
49      data->pre_comp_k11 = 0.393326f;
50      data->pre_comp_k12 = 0.0f;
51
52      data->measure_cnt = 0;
53
54  #ifdef PRECOMP_KP
55      for (i=0; i<PRECOMP_KP_CNT; i++) {
56          PreCompKP(data);
57      }
58  #endif
59  }
60
61  void KalmanVelUpdate(KalmanVelObject *data, float acc_comp1, float acc_comp2, float omega)
62  {
63      float z;
64
65      // Step 1
66      // x(k) = Fx(k-1) + Bu + w:
67      StatePropagation(data, omega);
68
69      // Step 2
70      // P = FPF'+Q
71      CovariancePrediction(data);
72
73  ////////////////////////////////////
74  // here  moving average of acc for
75  // 200 ms and then measurement step
76  if (data->measure_cnt < MSR_CNT) {
77      data->measure_cnt ++;
78  #ifdef MOVING_AVG_ACC
79          data->acc_comp1_sum += acc_comp1;
80          data->acc_comp2_sum += acc_comp2;
```

```
81  #endif
82  } else {
83      data->measure_cnt = 0;
84  #ifdef MOVING_AVG_ACC
85      acc_comp1 = (data->acc_comp1_sum)/MSR_CNT;
86      acc_comp2 = (data->acc_comp2_sum)/MSR_CNT;
87  #endif MOVING_AVG_ACC
88
89      data->acc_comp1_sum = 0.0f;
90      data->acc_comp2_sum = 0.0f;
91
92      ////////////////////////////////////
93      // do measurement step of kalman Filter
94
95      // measurment of acc
96  #ifdef  SMALL_ANGLE_APPROX
97    z = acc_comp1/acc_comp2;
98  #else // no small angle approximation
99      z = atan2(acc_comp1, acc_comp2);
100 #endif
101     // convert to degrees
102     z = z/PI * 180.0;
103
104     // Step 3
105     // y = z(k) - Hx(k)
106     // Step 4
107     // S = HPT' + R
108     // Step 5
109     // K = PH'*inv(S)
110     ComputeKalmanGain(data, z);
111
112     // Step 6
113     // x = x + Ky
114     StateUpdate(data);
115
116     // Step 7
117     // P = (I-KH)P
118     CovarianceUpdate(data);
119 }
120 }
121
122 void KalmanVelUpdateFast(KalmanVelObject *data, float acc_comp1, float acc_comp2, float omega)
123 {
124     float y, z;
125
126     // Step 1
127     // x(k) = Fx(k-1) + Bu + w:
128     StatePropagation(data, omega);
129
130     ////////////////////////////////////
131 // here  moving average of acc for
132 // 200 ms and then measurement step
133 if (data->measure_cnt < MSR_CNT) {
134     data->measure_cnt ++;
135 #ifdef MOVING_AVG_ACC
136         data->acc_comp1_sum += acc_comp1;
137         data->acc_comp2_sum += acc_comp2;
138 #endif
139 } else {
140     data->measure_cnt = 0;
141 #ifdef MOVING_AVG_ACC
142     acc_comp1 = (data->acc_comp1_sum)/MSR_CNT;
143     acc_comp2 = (data->acc_comp2_sum)/MSR_CNT;
144 #endif MOVING_AVG_ACC
145
146     data->acc_comp1_sum = 0.0f;
147     data->acc_comp2_sum = 0.0f;
148
149     ////////////////////////////////////
150     // do measurement step of kalman Filter
151
152     // measurment of acc
153 #ifdef  SMALL_ANGLE_APPROX
154   z = acc_comp1/acc_comp2;
155 #else // no small angle approximation
156     z = atan2(acc_comp1, acc_comp2);
157 #endif
158     // convert to degrees
159     z = z/PI * 180.0;
160
161     // Step 3
162     // y = z(k) - Hx(k)
163     data->y = z - data->x1;
164
165     // Step 6
166     // x = x + Ky
167     data->x1 = data->x1 + data->pre_comp_k11*data->y;
168     data->x2 = data->x2 + data->pre_comp_k11*data->y;
169 }
170 }
171
```

```
172 // private functions
173 // step 1
174 void StatePropagation(KalmanVelObject *data, float u)
175 {
176     // Step 1
177     // x(k) = Fx(k-1) + Bu + w:
178     data->x1 = data->x1 + u*data->dt - data->x2*data->dt;
179     //x2 = x2;
180 }
181
182 // step 2
183 void CovariancePrediction(KalmanVelObject *data)
184 {
185     float a, b;
186
187     // Step 2
188     // P = FPF'+Q
189     a = data->p12*data->dt; // = data->21*data-dt
190     b = data->p22*data->dt;
191     data->p11 = data->p11 - a - a + b*data->dt + data->q1;
192     data->p12 = data->p12 - b;
193     data->p21 = data->p12; // Symmetry of Covariance Matrix
194     data->p22 = data->p22 + data->q2;
195 }
196
197 // step 3-5
198 void ComputeKalmanGain(KalmanVelObject *data, float z)
199 {
200     float s;
201
202     // Step 3
203     // y = z(k) - Hx(k)
204     data->y = z - data->x1;
205
206     // Step 4
207     // S = HPT' + R
208     s = data->p11 + data->r1;
209
210     // Step 5
211     // K = PH'*inv(S)
212   data->k11 = data->p11/s;
213     data->k12 = data->p21/s;
214 }
215
216 // step 6
217 void StateUpdate(KalmanVelObject *data)
218 {
219     // Step 6
220     // x = x + Ky
221     data->x1 = data->x1 + data->k11*data->y;
222     data->x2 = data->x2 + data->k12*data->y;
223 }
224
225 // step 7
226 void CovarianceUpdate(KalmanVelObject *data)
227 {
228     // Step 7
229     // P = (I-KH)P
230     data->p11 = data->p11 - data->k11*data->p11;
231     data->p12 = data->p12 - data->k11*data->p12;
232     data->p21 = data->p12; // symmetry of covariance matrix
233     data->p22 = data->p22 - data->k12*data->p12;
234 }
235
236 void PreCompKP(KalmanVelObject *data)
237 {
238     float a, b;
239     float s;
240     float k11, k12;
241
242     // Step 2
243     // P = FPF'+Q
244     a = data->p12*data->dt; // = data->21*data-dt
245     b = data->p22*data->dt;
246     data->p11 = data->p11 - a - a + b*data->dt + data->q1;
247     data->p12 = data->p12 - b;
248     data->p21 = data->p12; // Symmetry of Covariance Matrix
249     data->p22 = data->p22 + data->x2*data->q2;
250
251 if (data->measure_cnt < MSR_CNT) {
252     data->measure_cnt ++;
253 } else {
254     data->measure_cnt = 0;
255
256     // Step 4
257     // S = HPT' + R
258     s = data->p11 + data->r1;
259
260     // Step 5
261     // K = PH'*inv(S)
262     data->pre_comp_k11 = data->p11/s;
```

```
263         data->pre_comp_k12 = data->p21/s;
264         //printf ("k11: %f k12: %f\n", data->k11, data->k12);
265
266         // Step 7
267         // P = (I-KH)P
268         data->p11 = data->p11 - data->pre_comp_k11*data->p11;
269         data->p12 = data->p12 - data->pre_comp_k11*data->p12;
270         data->p21 = data->p12; // symmetry of covariance matrix
271         data->p22 = data->p22 - data->pre_comp_k12*data->p12;
272     }
273     }
```

C.2.2 Matrix-Berechnungen

Im Folgenden werden alle Matrixmultiplikationen gezeigt, die für die Herleitung der Codezeilen des Kalman-Filters zur Lageschätzung aus Kapitel *Code Kalman-Filter zur Lagestützung* im Anhang notwendig sind gezeigt.

$$\begin{bmatrix}1 & -dt\\ 0 & 1\end{bmatrix}\begin{bmatrix}x_1\\ x_2\end{bmatrix}+\begin{bmatrix}dt\\ 0\end{bmatrix}u=\begin{bmatrix}x_1-x_2dt+udt\\ x_2\end{bmatrix}$$

Step 1:
$x = Fx + Bu$

$$\begin{bmatrix}1 & -dt\\ 0 & 1\end{bmatrix}\begin{bmatrix}p_{11} & p_{12}\\ p_{21} & p_{22}\end{bmatrix}\begin{bmatrix}1 & 0\\ -dt & 1\end{bmatrix}+\begin{bmatrix}q_1 & 0\\ 0 & q_2\end{bmatrix}$$
$$=\begin{bmatrix}p_{11}-p_{21}dt & p_{12}-p_{22}dt\\ p_{21} & p_{22}\end{bmatrix}\begin{bmatrix}1 & 0\\ -dt & 1\end{bmatrix}+\dots$$
$$=\begin{bmatrix}p_{11}-p_{21}dt-\overbrace{(p_{12}-\underbrace{p_{22}dt}_{a})}^{b}dt & \overbrace{p_{12}-\underbrace{p_{22}dt}_{a}}^{b}\\ \overbrace{p_{21}-\underbrace{p_{22}dt}_{a}}^{b} & p_{22}\end{bmatrix}+\begin{bmatrix}q_1 & 0\\ 0 & q_2\end{bmatrix}$$
$$=\begin{bmatrix}p_{11}-p_{21}dt-b\cdot dt+q_1 & b\\ b & p_{22}+q_2\end{bmatrix}\quad\text{mit}\quad\begin{matrix}a=p_{22}dt\\ b=p_{12}-p_{22}dt\end{matrix}$$
$$\text{Symmetrie}\begin{cases}=p_{12}-a\\ =p_{21}-a\end{cases}$$

Step 2:
$P = FPF^T + Q$

$$y=z-\begin{bmatrix}1 & 0\end{bmatrix}\begin{bmatrix}x_1\\ x_2\end{bmatrix}=z-x_1$$

Step 3:
$y = z - Hx$

$$\begin{bmatrix}1 & 0\end{bmatrix}\begin{bmatrix}p_{11} & p_{12}\\ p_{21} & p_{22}\end{bmatrix}\begin{bmatrix}1\\ 0\end{bmatrix}+R$$
$$=\begin{bmatrix}p_{11} & p_{12}\end{bmatrix}\begin{bmatrix}1\\ 0\end{bmatrix}+R$$
$$=p_{11}+r_1$$

Step 4:
$S = HPH^T + R$

$$\begin{bmatrix}p_{11} & p_{12}\\ p_{21} & p_{22}\end{bmatrix}\begin{bmatrix}1\\ 0\end{bmatrix}s_{11}^{-1}=\begin{bmatrix}p_{11}\\ p_{12}\end{bmatrix}s_{11}^{-1}$$
$$=\begin{bmatrix}p_{11}/s_{11}\\ p_{12}/s_{11}\end{bmatrix}$$

Step 5:
$K = PH^T S^{-1}$

$$\begin{bmatrix}x_1\\ x_2\end{bmatrix}+\begin{bmatrix}k_{11}\\ k_{12}\end{bmatrix}y=\begin{bmatrix}x_1+k_{11}y\\ x_2+k_{12}y\end{bmatrix}$$

Step 6:
$x = x + Ky$

$$\begin{bmatrix}p_{11} & p_{12}\\ p_{21} & p_{22}\end{bmatrix}-\begin{bmatrix}k_{11}\\ k_{12}\end{bmatrix}\begin{bmatrix}1 & 0\end{bmatrix}\begin{bmatrix}p_{11} & p_{12}\\ p_{21} & p_{22}\end{bmatrix}=\dots\begin{bmatrix}k_{11} & 0\\ k_{12} & 0\end{bmatrix}\begin{bmatrix}p_{11} & p_{12}\\ p_{12} & p_{22}\end{bmatrix}$$
$$=\begin{bmatrix}p_{11} & p_{12}\\ p_{21} & p_{22}\end{bmatrix}-\begin{bmatrix}k_{11}p_{11} & k_{11}p_{12}\\ k_{12}p_{11} & k_{12}p_{12}\end{bmatrix}=\begin{bmatrix}p_{11}-k_{11}p_{11} & p_{12}-k_{11}p_{12}\\ p_{21}-k_{12}p_{11} & p_{22}-k_{12}p_{12}\end{bmatrix}$$
$$\text{Symmetrie}\begin{cases}=p_{21}-\frac{p_{21}}{p_{11}+r}\cdot p_{11}\\ =p_{12}-\frac{p_{11}}{p_{11}+r}\cdot p_{12}\end{cases}$$
bei Einsetzen von K wird Symmetrie erkennbar.

Step 7:
$P = P - KHP$

ANHANG D

Codes Protokolle

D.1 MAVlink Samples

Bemerkung: Alle nachfolgenden Schritte wurden unter Ubuntu 14.04 getestet.

D.1.1 MAVlink installieren

Herunterladen und entpacken:

```
mkdir myMAVtools && cd myMAVtools
scp -P 4444 gsa39665@hps.hs-regensburg.de:html/files/ganymedtools_v0.1.2015.tar.bz2 .
tar xvjf ganymedtools_v0.1.2015.tar.bz2
```

Für die lokale Installation von MAVlink und einiger Abhängigkeiten wurde ein Skript geschrieben. Dies kann mit folgenden Befehlen ausgeführt:

```
cd GanymedTools/mavlink
./setup.py --install
```

Nun muss noch die Pfadtvariable PYTHONPATH gesetzt werden, damit die Python-Tools von MAVlink systemweit ausführbar werden. Wird dieser Schritt vergessen, kompilieren unter umständen einige Codes nicht, da die nötigen Headers vom Makefile nicht erzeugt werden können. Auch dies kann mit dem genannten Installationskript gemacht werden:

```
./setup.py --setpath
```

D.1.2 MAVlink deinstallieren

Die Deinstallation von MAVlink wird mit folgendem Befehl durchgeführt:

```
./setup --uninstall
```

D.1.3 Ausführen des Sample Codes

Bemerkung: Folgende Befehle setzen vorraus, dass die Schritte aus Kapitel *MAVlink installieren* erfolgreich durchgeführt wurden.

Die Sample-Codes zur Implementierung von MAVlink in verschiedenen Sprachen befinden sich in den jeweiligen Unterordnern des Ordners *GanymedTools/src*. In allen Unterordnern befindet sich eine Datei namens *readme.rst*. Darin befinden sich Anweisungen, die jeweiligen Codes zu kompilieren und auszuführen.

Mit folgenden Befehlen können die C-Codes unter Linux kompiliert und ausgeführt werden.

Headers von MAVlink erzeugen:

```
cd GanymedTools/src/linux
make headers
```

Und schließlich können die Codes im selben Verzeichnis mit folgendem Befehl kompiliert werden:

```
make binary
```

D.1.4 Troubleshooting

Problem: Will man die JavaScript Sources mit *mavgenerate.py* erzeugen kann folgender Fehler erscheinen:

```
Errors occoured in mavgen:
[Errno 2] No such file or directory:
'./javascript/lib/jspack/jspack.js'
```

Lösung: Damit das Submodule *jspack* nachgeladen wird, müssen folgende Befehle ausgeführt werden:

```
git submodule init
git submodule update
```

Zwar tritt dann die Fehlermeldung immer noch auf, kann aber dann ignoriert werden. Dies geht aus folgendem Thread hervor:

https://groups.google.com/forum/#!topic/mavlink/iDKpDrqL9AI

Außerdem werden hier Möglichkeiten diskutiert, *JavaScript* ohne die Abhängigkeit von *node.js* durchzuführen.

Literaturverzeichnis

[1] Am1808/omap-l138 linux user guide.

[2] An1168 - high-speed usb pcb layout recommendations. http://www.cypress.com/?docID=29107. [Online; accessed 30-Juni-2015].

[3] An470 - omap-l138 som-m1 design checklist. file:///home/usappz/Downloads/1017728D_AN470_OMAP-L138_SOM-M1_Design_Checklist.pdf. [Online; accessed 30-Juni-2015].

[4] Bmp180 - digital, barometric pressure sensor. https://ae-bst.resource.bosch.com/media/downloads/pressure/bmp180/Flyer_BMP180_08_2013_web.pdf. [Online; accessed 30-Juni-2015].

[5] Das u-boot – the universal boot loader. http://www.denx.de/wiki/U-Boot/. [Online; accessed 30-Juni-2015].

[6] Data sheet - bmx055 - small, versatile 9-axis sensor module. http://ae-bst.resource.bosch.com/media/products/dokumente/bmx055/BST-BMX055-DS000-02.pdf. [Online; accessed 30-Juni-2015].

[7] Datasheet:technical data - xtrinsic mpl3115a2 i2c precision altimeter. http://cache.freescale.com/files/sensors/doc/data_sheet/MPL3115A2.pdf. [Online; accessed 30-Juni-2015].

[8] Drone solutions. http://www.microdrones.com/. [Online; accessed 26-Juni-2015].

[9] Embedded solutions. http://www.logicpd.com/. [Online; accessed 30-Juni-2015].

[10] Entwickler und hersteller von mikro-uavs. http://www.asctec.de/. [Online; accessed 26-Juni-2015].

[11] Eoma-68 specification. http://elinux.org/Embedded_Open_Modular_Architecture/EOMA-68. [Online; accessed 26-Juni-2015].

[12] Footprint for tc2050-idc plug-of-nails cable. http://kurzurl.net/mzKTb. [Online; accessed 30-Juni-2015].

[13] Mpu9150 product specification - revistion 4.3. http://www.inertialelements.com/docs/PS-MPU-9150A-00v4_3.pdf. [Online; accessed 30-Juni-2015].

[14] Ms5611-01ba01 variometer module, with lcp cap. http://www.amsys.de/sheets/amsys.en.ms5611_01ba01.pdf. [Online; accessed 30-Juni-2015].

[15] Omap-l132/l138, tms320c6742/6/8 pin multiplexing utility. http://www.ti.com/lit/an/sprab63b/sprab63b.pdf. [Online; accessed 30-Juni-2015].

[16] Optical power isolator. http://l2wenergy.com. [Online; accessed 28-Mai-2015].

[17] Scarabot. http://davinci-copters.com/innovative-akkutechnik/. [Online; accessed 26-Juni-2015].

[18] Sd simplified specification. https://www.sdcard.org/downloads/pls/. [Online; accessed 30-Juni-2015].

[19] Sprs586i - omap-l138 c6000™ dsp+ arm® processor. http://www.ti.com/lit/ds/symlink/omap-l138.pdf. [Online; accessed 30-Juni-2015].

[20] Sprufm2b - tms320c674x/omap-l1x processor multimedia card (mmc)/secure digital (sd) card controller. http://www.ti.com/lit/ug/sprufm2b/sprufm2b.pdf. [Online; accessed 30-Juni-2015].

[21] Wireless power delivery systems. http://lasermotive.com. [Online; accessed 28-Mai-2015].

[22] Xds100v2. http://tiexpressdsp.com/index.php/Xds100v2. [Online; accessed 30-Dezember-2014].

[23] Integration of civil unmanned aircraft systems (uas) in the national airspace system (nas) roadmap. faa.gov/uas/media/UAS_Roadmap_2013, 2013. [Online; accessed 28-Mai-2015].

[24] Roadmap for the integration of civil remotely-piloted aircraft systems into the european aviation system. http://ec.europa.eu/enterprise/selectors/aerospace/files/rpas-roadmap_en.pdf, 2013. [Online; accessed 28-Mai-2015].

[25] Cavallo A., Cirillo A., Cirillo P., De Maria G., Falco P., Natale C., and Pirozzi S. Experimental comparison of sensor fusion algorithms for attitude estimation. *The International Federation of Automatic Control*, 2014.

[26] Ahmed Abdelgawad and Magdy Bayoumi. *Resource-Aware Data Fusion Algorithms for Wireless Sensor Networks*. Springer, 2012.

[27] Michael C. Achtelik, Jan Stumpf, Daniel Gurdan, and Klaus-Michael Doth. Design of a flexible high performance quadcopter platform breaking the mav endurance record with llaser power beaming. *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2011.

[28] Fakhri Alam, ZhaiHe Zhou, and JiaJia Hu. A comparative analysis of orientation estimation filters using mems based imu. *2nd International Conference on Research in Science, Engineering and Technology*, 2014.

[29] Seth B. Anderson. Historical overview of v/stol aircraft technology. *NASA Technical Memorandum 81280*.

[30] ac Antoulas. *Mathematical System Theory - The Influence of R. E. Kalman*. Springer, 1991.

[31] Stjepan Bogdan and Matko Orsag. *Recent Advances in Aircraft Technology*. Intech, 2012.

[32] David A. Boruchoff. The three greatest inventions of modern times: An idea and its public. *Entangled Knowledge: Scientific Discourses and Cultural Difference*, 2012.

[33] André Bouabadallah and Roland Siegwart. Pid vs lq control techniques applied to an indoor micro quadrotor. *IEEE International Conference Intelligent Robots and Systems*, 2004.

[34] S. Bouabdallah and R. Siegwart. Full control of a quadrotor. *Intelligent Robots and Systems*, 2007.

[35] Samir Bouabdallah, Pierpaolo Murrieri, and Roland Siegwart. Design and control of an indoor micro quadrotor. *IEEE International Conference on Robotics and Automation*, 2004.

[36] Samir Bouabdallah and Roland Siegwart. Backstepping and sliding-mode techniques applied to an indoor micro quadrotor. *International Conference on Robotics Automation*, 2005.

[37] Google Code. Protocol buffers. http://code.google.com/p/protobuf. [Online; accessed 15-März-2014].

[38] Don Coppersmith and Shmuel Winograd. Matrix multiplication via arithmetic progression. *J. Symbolic Computation*, 1990.

[39] Ricardo Román Cordón, Javier Sáez Nieto, and Cristina Cuerno Rejado. Rpas integration in non-segregated airspace: the sesar approach. *Fourth SESAR Innovation Days*.

[40] Offizielle Website der Europäischen Union. Kommentar eu-kommissarin bieńkowska, März 2015. zuletzt aufgerufen März 2015, http://ec.europa.eu/deutschland/press/pr_releases/13213_de.htm.

[41] Travis Dierks and Sarangapani Jagannathan. Output feedback control of a quadrotor uav using neural networks. *IEEE Transactions on Neural Networks*, 2010.

[42] Guillaume Ducard and Raffaello D'Andrea. Autonomous quadrotor flight using a vision system and accommodating frames misalignment. *IEEE International Symposium Industrial Embedded Systems*, 2009.

[43] Escareño, Salazar, Romero, and Lozano. Trajectory control of a quadrotor subject to 2d wind disturbances. *Journal of Intelligent and Robotic Systems*, 2013.

[44] Apache Etch. Apache etch. http://etch.apache.org. [Online; accessed 15-März-2014].

[45] ETH-Zuerich. Pixhawk. https://pixhawk.org/start. [Online; accessed 15-März-2014].

[46] Ramsey Faragher. Understanding the basis of the kalman filter via a simple and intuitive derivation. *IEEE Signal Processing Magazine*, 2012.

[47] Jay Farrell and Barth Matthew. *The Global Positioning System and Inertial Navigation*. McGraw-Hill, 1998.

[48] flight dynamics and control software library in C++. JSBsim. www.jsbsim.org, 2012. [Online; accessed 23-Mai-2012].

[49] Holger Flühr. *Flugsicherungstechnik*. Springer, Graz, Österreich, 2010.

[50] Free Software Foundation. Gnu scientific license (gsl). zuletzt aufgerufen März 2015, http://www.gnu.org/software/gsl.

[51] GAZEBO. Robot Simulation Framework. http://gazebosim.org. [Online; accessed 25-Mai-2012].

[52] Mohinder S. Gewal and Angus P. Andrews. *Kalman Filtering - Theory and Practice Using MATLAB*. WILEY, Hoboken, New Jersey, 2001.

[53] Andrew Gibiansky. Quadrotor Dynamics and Simulation. andrew.gibiansky.com/blog/physics/quadcopter-dynamics, 2012. [Online; accessed 23-Mai-2012].

[54] Rahul Goel, Sapan M. Shah, Nitin K. Gupta, and N. Ananthkrishnan. Modeluing, simulation and flight testing of an autonomous quadrotor.

[55] James M. Goppert. *AN ADAPTABLE, LOW COST TEST-BED FOR UNMANNED VEHICLE SYSTEMS RESEARCH*. PhD thesis, Purdue University, 2011.

[56] Mohinder S. Grewal and Angus P. Andrews. Application of kalman filtering in aerospace 1960 to the present. *IEEE Control Systems Magazine*, 2010.

[57] Daniel A. Griffith and A. Gordon Leishman. A study of dual-rotor interference and ground effect using a free-vortex wake model. *In Proc. 58th Annual Forum and Technology Display of the American Helicopter Assoc.*, 2002.

[58] Andreas Gschossmann. Download-links. zuletzt aufgerufen April 2015, https://hps.othr.de/gsa39665/downloads.html.

[59] Andreas Gschossmann. Doxygen-dokumentation von deadalussim. zuletzt aufgerufen April 2015, https://hps.hs-regensburg.de/gsa39665/files/doxygen/index.html.

[60] Andreas Gschossmann. Evaluation of an open source realtime-kinematic gps software library. zuletzt aufgerufen April 2015, https://hps.othr.de/gsa39665/files/paper_diffgps.pdf.

[61] Andreas Gschossmann. Satellitennavigation mit dem global positioning system (gps). zuletzt aufgerufen April 2015, http://hps.othr.de/gsa39665/files/Bericht_Breitbandige_Zugangsnetze.pdf.

[62] Andreas Gschossmann. Luftverkehrsrechtliche Betrachtungen zu UAVs. hps.othr.de/gsa39665/files/rechtliches_embed.svg, 2013. [Online; accessed 30-März-2015].

[63] Daniel Gurdan, Jan Stumpf, Michael Achtelik, Klaus-Michael Doth, Gerd Hirzinger, and Daniela Rus. Energy-efficient autonomous four-rotor flying robot controlled at 1 khz. *IEEE International Conference on Robotics and Control*, 2007.

[64] Markus Haid. *Verbesserung der referenzlosen inertialen Objektverfolgung zur Low-cost Indoor-Navigation durch Anwendung der Kalman-Filterung*. Fraunhofer IRB Verlag, 2004.

[65] William Rowan Hamilton. Quaternions, or on a new system of imagineries in algebra. *Philosophical Magazine*, 1844.

[66] Helmholtz-Zentrum Potsdam. Department of Transportation (GFZ). http://www.gfz-potsdam.de.

[67] Grabriel M. Hoffmann, Haomiao Huang, Steve Wasl, and Er Claire J. Tomlin. Quadrotor helicopter flight dynamics and control: Theory and experiment. *AIAA Guidance, Navigation and Control Conference*, 2007.

[68] Haomiao Huang, Gabriel M. Hoffmann, Steven L. Waslander, and Claire J. Tomlin. Aerodynamics and control of autonomous quadrotor helicopters in aggressive maneuvering. *Robotics and Automation*, 2009.

[69] Raymond E. Hunt, Michel Cavigelli, Craig S.T. Daughtry, James McMurtrey, and Charles L. Walthall. Evaluation of digital photography from model aircraft for remote sensing of crop biomass and nitrogen status. *Precision Agriculture*, 2005.

[70] Earth Science Data Interface. ESDI. http://glcfapp.glcf.umd.edu:8080/esdi/index.jsp. [Online; accessed 23-Mai-2012].

[71] Roger Isakson. labview-quaternion-ahrs. zuletzt aufgerufen März 2014, https://code.google.com/p/labview-quaternion-ahrs/.

[72] jgoppert. Mavsim (github). zuletzt aufgerufen April 2015, https://github.com/jgoppert/mavsim.

[73] Yan-Bin Jia. Quaternions and rotations*. zuletzt aufgerufen April 2015, https://www.cs.iastate.edu/~cs577/handouts/quaternion.pdf.

[74] Meyer Johannes, Sendorbry Alexander, Kohlbrenner Stefan, Klingauf Uwe, and Stryk Oskar. Comprehensive simulation of quadrotor uavs using ros and gazebo. *Simulation, Modeling, and Programming for Autonomous Robots*, 2013.

[75] Kamran Joyo, Ahmed Faiz, Tanveer Hassen, Warsi Faizan, and A. T. Hussain. Altitude and horizontal motion control of quadrotor uav in the presence of air turbulence. *IEEE Conference on Systems, Process and Control*, 2013.

[76] JSON. Introducing JSON. json.org. [Online; accessed 15-März-2014].

[77] JSON. The json license. http://www.json.org/license.html. [Online; accessed 15-März-2014].

[78] R.E. KALMAN. A new approach to linear filtering and prediction problems. *Journal of Basic Engineering*, 2012.

[79] J. Kare and T. Nugent.

[80] Nonami Kenzo, Kendoul Farid, Suzuki Satoshi, Wang Wei, and Nakazawa Daisuke. *Autonomous Flying Robots*. Springer, 2010.

[81] Sastry Shankar Lee Deawon, Jin Kim H. Feedback linearization vs. adaptive sliding mode control for a quadrotor helicopter. *International Journal of Control, Automation, and Systems*, 2009.

[82] J. Gordon Leishman. *Principles of Helicopter Aerodynamics*. CAMBRIDGE UNVERSITY PRESS, 2000.

[83] J. Gordon Leishman. The bréquaet-richet quadrotor helicopter of 1907. *Vertiflite Vol. 47*, 2002.

[84] Hyon Lim, Jaemann Park, Deawon Lee, and H.J. Kim. Build your own quadrotor: Open-source projects on unmanned aerial vehicles. *Robotics and Automation Magazin*, 2012.

[85] O. Loeffield. *Estimationstherie II*. Oldenburg Verlag, 1990.

[86] T. Madani and A. Benallegue. Adaptive control via backstepping technique and neural networks of a quadrotor helicopter. *Proceedings of the 17th World Congress The International Federation of Automatic Control*, 2008.

[87] Tarek Madani and Abdelaziz Benallegue. Backstepping control for a quadrotor helicopter. *Proceedings of the 2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2006.

[88] O.H. Sebastian Madgwick. An efficient orientation filter for inertial and inertial/magnetic sensor arrays. *Internal Report*, 2010.

[89] Sebastian O.H. Madgwick, Andrew J.L Harrison, and Ravi Vaidyanathan. Estimation of imu and marg orientation using a gradient descent algorithm. *IEEE International Conference on Rehabilitation Robotics*, 2011.

[90] R. Mahony, Tarek Hamel, and Jean-Michel Pflimlin. Nonlinear complementary filters on special orthogonal group. *HAL archives-ouverts*, 2010.

[91] Jao Luis Marins, Xiaoping Yun, Eric R. Bachmann, Robert B. McGhee, and Michael J. Zyda. An extended kalman filter for quaternion-based orientation estimation using marg sensors. *Proceedings of the 2001 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2001.

[92] Leonard A. McGee and Stanley F. Schmidt. Discovery of the kalman filter as a practical tool for aerospace and industry. *NASA Technical Memorandum 86847*, 1985.

[93] Oliver Meister. *Entwurf und Realisierung einer Aufklährungsplattform auf Basis eines unbemannten Minihelikopters mit autonomen Flugfähigkeiten*. dissertation, Karlsruher Institut für Technologie (KIT), 2010.

[94] Nathan Michael, Daniel Mellinger, Quentin Lindsey, and Vijay Kumar. The grasp multiple micro uav testbed. *Robotics and Automation Magazine, IEEE*, 2010.

[95] H.B. Mitchell. *Multi-Sensor Data Fusion: An Introduction*. Springer, 2007.

[96] Micheal R. Molt, Dale E. Schinstock, and Robert M. Caplinger. Gps-aided ins solution with photogrammetry validation. *Aerotech Congress and Exhibition*, 2005.

[97] Mark W. Mueller and D'Andrea Raffaello. Stability and control of a quadrocopter despite the complete loss of one, two, or three propellers. *IEEE International Conference on Robotics and Automation (ICRA)*, 2014.

[98] Amr Nagaty, Sajad Saeedi, Carl Thibault, Mae Seto, and Howard Li. Control and navigation framework for quadrotor helicopters. *Journal of Intelligent & Robotic Systems*, April 2013.

[99] R. Nelson. *Flight Stability and Automatic Control*. McGraw-Hill, 1989.

[100] Maral Partovibakhsh and Guangjun Liu. An adaptive unscented kalman filtering approach for online estimation of model parameters and state-of-charge of lithium-ion batteries for autonomous mobile robots. *IEEE Transactions on Control Systems Technology*, 2015.

[101] Oliveira P. Pascoal A., Kaminer I. Navigation system design using time-varying complementary filters. *International Conference on Control Applications*, 1998.

[102] Roger M. du Piessis. Poor man's explenation of kalman filtering or how i stopped worrying and learned to love matrix inversion. *North American Rockwell Electronics Group*, 1967.

[103] Caitlin Powers, Daniel Mellinger, Aleksandr Kushleyev, Bruce Kothmann, and Vijay Kumer. Influence of aerodynamics and proximity effects in quadrotor flight. *13th International Symposium on Experimental Robotics*, 2012.

[104] H. William Press, Brian P. Flannery, Saul A. Teukolsky, and William T. Vertterling. *Numerical Recipes in C: The Art of Scientific Computing*. Camebridge University Press, 1992.

[105] OpenPilot Project. Open source uav autopilot. zuletzt aufgerufen April 2015, https://www.openpilot.org/.

[106] J. Rasmussen, J. Nielsen, F. Garcia-Ruiz, S. Christensen, and J.C. Streibig. Potential uses of small unmanned aircraft systems (uas) in weed research. *Weed Research*, 2013.

[107] F. Rinaldi. Study on the use of neural networks in control systems.

[108] O. Rogall, S. Fisun, F. Schilling, M. Loschonky, R. Schimko, and Reindl L.M. Radargestützte personenortung. *Sensoren und Messsysteme*, 2010.

[109] RTdynamics. Rotorlib fdm - a helicopter dynamics simulation library. zuletzt aufgerufen April 2015, http://www.rtdynamics.de/.

[110] Peter Rüegg. Medal presented personally by obama, 2009. zuletzt aufgerufen März 2014, http://www.ethlife.ethz.ch/archive_articles/091008_kalman_per/index_EN.

[111] J. Seddon. *Basic Helicopter Aerodynamics*. BSP Professional Books, 1990.

[112] William Selby, Peter Corke, and Daniela Rus. Autonomous aerial navigation and tracking of marine animals. *Proceedings of Australasian Conference on Robotics and Automation*, 2011.

[113] Colton Shane. The balance filter, 2007. zuletzt aufgerufen März 2014, http://robottini.altervista.org/wp-content/uploads/2014/04/filter.pdf.

[114] Jack F. Shepherd and Kagan Tumer. Robust neuro-control for a micro quadrotor.

[115] Wang Shoahua and Ying Yang. Quadrotor aircraft attitude estimation and control based on kalman filter. *Control Conference (CCC)*, 2012.

[116] David J. Siminovitch. *Rotations in NMR: Part I. Euler-Rodrigues Parameters and Quaternions*. WILEY, Lethbridge, Canada, 1997.

[117] Dan Simon. Kalman filtering. *Embedded Systems Programming*.

[118] Dan Simon. *Optimal State Estimation: Kalman, H Infinity, and Nonlinear Approaches*. WILEY, Hoboken, New Jersey, 2006.

[119] SWIFT NAVIGATION. Piksi low-cost high-performance GPS receiver with Real-Time Kinematics (RTK). http://www.swiftnav.com/piksi.html.

[120] Nitin Sydney, Brendan Smyth, and Derek A. Paley. Dynamic control of autonomous quadrotor flight in an estimated. *52nd IEEE Conference on Decision and Control*, 2013.

[121] A. Tavebi and S. McGilvray. Attitude stabilization of a four-rotor aerial robot. *43rd IEEE Conference on desicion and control*, 2004.

[122] Fox Dieter Thrun Sebastian, Burgard Wolfram. *Propabilistic Robotics*. MIT Press, 2006.

[123] John Ting-Yung Wen and Kenneth Kreutz-Delgado. The attitude control problem. *IEEE Transactions on automatic Control*, 1991.

[124] Apache TLP. Apache thrift. https://thrift.apache.org/. [Online; accessed 15-März-2014].

[125] Kenton Varda. Cap'n proto cerealization protocol. https://capnproto.org. [Online; accessed 15-März-2014].

[126] Steven Waslander and Carlos Wang. Wind disturbance estimation and rejection for quadrotor position control. *Aerospace Conference*, 2013.

[127] Greg Welch and Bishop Gary. An introduction to the kalman filter. *Department of Computer Science University of North Carolina*, 2012.

[128] J. Wendel, C. Schlaile, and Trommer F. Direct kalman filtering of gps/ins for aerospace applications. *IEEE Transactions on Aerospace and Electric Systems*, 2002.

[129] Jan Wendel. *Integrierte Navigationssysteme*. Oldenburg Verlag, München, 2011.

[130] Wikimedia. Apollo guidance computer ansicht nasa. zuletzt aufgerufen März 2015, commons.wikimedia.org/wiki/File:Agc_view.jpg.

[131] Written and maintained by falkTX. Qtsixa sixaxis joystick manager. zuletzt aufgerufen April 2015, http://qtsixa.sourceforge.net/.

[132] x-io Technologies. Open source imu and ahrs algorithms, 2012. zuletzt aufgerufen März 2014, www.x-io.co.uk/open-source-imu-and-ahrs-algorithms.

[133] Stefan Zorn, Matthias Maser, Alexander Goetz, Richard Rose, and Robert Weigel. A power saving jamming system for e-gsm900 and dcs1800 cellular phone networks for search and rescue applications. *IEEE Topical Conference on Wireless Sensors and Sensor Networks (WiSNet)*, 2011.

[134] Lukasz Zwirello. *Realization Limits of Impulse-Radio UWB Indoor Localization Systems*. Scientific Publishing, 2013.

[135] Lorenz Meier (ETH Zürich). Mavlink - micro air vehicle communication protocol. http://qgroundcontrol.org/mavlink. [Online; accessed 15-März-2014].