

# **RaHM-Lab: Positionsregelung eines Quadrocopters zur Risikominimierung von studentischen Laborversuchen**

**T3100**

für die Prüfung zum  
Bachelor of Science

im Studiengang Informatik  
an der DHBW Karlsruhe

von

**Michael Maag**

17.05.2022

Bearbeitungszeitraum

6 Monate

Matrikelnummer

6170558

Gutachter der DHBW Karlsruhe

Prof. Dr. Marcus Strand

Michael Maag  
Freidorfstraße 14  
97957 Wittighausen

# Eigenständigkeitserklärung

Ich versichere hiermit, dass ich meine Ausarbeitung T3100 mit dem Thema “RaHM-Lab: Positionsregelung eines Quadrocopters zur Risikominimierung von studentischen Laborversuchen“ selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Ich versichere zudem, dass die eingereichte elektronische Fassung mit der gedruckten Fassung übereinstimmt.

---

Wittighausen, 17.05.2022

---

Unterschrift

# Inhaltsverzeichnis

Abbildungsverzeichnis .....	I
Abkürzungsverzeichnis .....	II
Tabellenverzeichnis .....	III
Code-Verzeichnis .....	V
1 Einleitung .....	1
2 Problemstellung.....	2
3 Methodik .....	3
3.1 Begriffe.....	3
3.2 Eingesetzte Software .....	3
3.2.1 ROS .....	3
3.2.2 Ubuntu - Betriebssystem .....	4
3.2.3 Visual Studio - IDE .....	4
4 Quadrokopter als System.....	5
4.1 Orientierungsmerkmale.....	5
4.2 Geometrien .....	5
4.3 Freiheitsgrade.....	7
4.4 Pose .....	7
4.4.1 lokale Pose .....	7
4.4.2 globale Pose .....	7
5 Regelsysteme.....	9
5.1 Regelkreis .....	9
5.2 Arten von Reglergliedern .....	10
5.2.1 P-Glied .....	10
5.2.2 I-Glied .....	10
5.2.3 D-Glied.....	11
5.2.4 PID-Regler .....	12
5.2.5 PT-Glied.....	12
5.2.6 Tt-Glied.....	12
5.3 Stabilität .....	12
6 Positionsregelung von Quadrokoptern .....	14
6.1 Positionsregelung von Quadrokoptern mittels Pose.....	14
6.2 Erzeugung von Posen aus Beschleunigungsdaten .....	14
6.2.1 Berechnung .....	14
6.2.2 Signalaufarbeitung .....	14

7	Eingesetzte Hardware.....	20
7.1	COEX Drohne.....	20
7.1.1	Control Stack .....	20
7.1.2	Funkfernsteuerung .....	20
7.1.3	Sensorik .....	20
7.1.4	Aufbau des Bausatzes.....	21
7.1.5	Inbetriebnahme.....	21
7.1.6	Mögliche Lösung der Aufgabenstellung.....	22
7.1.7	Troubleshooting.....	23
7.2	Parrot Drohne .....	24
7.2.1	Geometrie .....	24
7.2.2	Sensorik .....	24
7.2.3	Interaktion mittels <i>ROS</i> .....	25
7.2.4	Inbetriebnahme auf separatem Rechner.....	25
7.2.5	Troubleshooting.....	26
8	Software-Architektur.....	27
8.1	Architektur Konzept .....	27
8.2	<i>Domain Package</i> .....	28
8.3	<i>DroneController Package</i> .....	28
8.4	<i>Adapter Package</i> .....	28
8.5	<i>Controller Package</i> .....	28
8.6	<i>parrot Package</i> .....	28
8.7	<i>PosControl Package</i> .....	28
9	Implementierung.....	29
9.1	Domain Layer.....	29
9.2	Application Layer - <i>DroneController Package</i> .....	30
9.3	Adapter Layer .....	30
9.4	PlugIn Layer .....	30
9.4.1	<i>parrot Package</i> .....	30
9.4.2	<i>Controller Package</i> .....	31
9.4.3	<i>PosControl Package</i> .....	32
9.4.4	<i>calling Package</i> .....	32
9.4.5	<i>threading Package</i> .....	32
9.4.6	<i>coex Package</i> .....	34
10	Ergebnis .....	36
10.1	Analyse realer Flugdaten.....	36
10.1.1	Verlauf.....	36
10.1.2	Analyse.....	39

10.2 Signalverarbeitung .....	39
10.2.1 Ermittlung der dauerhaften Nullpunkt-Abweichung.....	39
10.2.2 Kalibrierung der Posenberechnung.....	40
11 Erweiterungen .....	42
11.1 Dedektion der Umgebung .....	42
11.1.1 Vorwärts-Berechnung .....	42
11.1.2 Rückwärts-Berechnung bei Ringschluss .....	42
11.2 Sensoriken.....	42
11.2.1 Abstandssensoren.....	42
11.2.2 Magnetometer .....	42
11.2.3 Kamera.....	42
11.2.4 GPS .....	42
12 Fazit und Ausblick.....	43
Literaturverzeichnis	
Anhang	

# Abbildungsverzeichnis

1	Orientierungsmerkmale eines Flugobjekts [15] . . . . .	5
2	wirkende Kräfte am Quadrokopter in <b>x</b> -Anordnung (vgl. [2]) . . . . .	6
3	Geometrien von Quadrooptern [1] . . . . .	6
4	wirkende Kräfte für gerollten Quadrokopter (vgl. [13]) . . . . .	8
5	Allgemeines Schema eines Regelkreises [6] . . . . .	9
6	Sprungantwort eines P-Glieds . . . . .	10
7	Sprungantwort eines I-Glieds . . . . .	11
8	Sprungantwort eines D-Glieds . . . . .	11
9	Sprungantwort eines PID-Glieds . . . . .	12
10	Sprungantwort eines PT-Glieds . . . . .	12
11	Originalkurve des Szenarios . . . . .	16
12	Einfluss von Signalrauschen . . . . .	17
13	Einfluss von Ausreißern . . . . .	18
14	Einfluss von konstanter Nullpunkt-Abweichung . . . . .	19
15	Aufbau des Bausatzes für die Drohne <i>Clover 4.20</i> . . . . .	21
16	Architektur des Regelungssystems . . . . .	27
17	Klassendiagramm des <i>calling Packages</i> . . . . .	33
18	Testflug: StatusID . . . . .	37
19	Testflug: Höhenverlauf . . . . .	38
20	Testflug Signalverarbeitung: Aufarbeitung $a_z$ . . . . .	40
21	Testflug Signalverarbeitung: Kalibrierung . . . . .	41

## **Abkürzungsverzeichnis**

<b>ADC</b>	Analog Digital Converter
<b>GPIO</b>	General Purpose Input Output
<b>HAL</b>	Hardware Abstraction Layer
<b>I/O</b>	Input/Output
<b>IDE</b>	Integrated Development Environment
<b>IEEE</b>	Institute of Electrical and Electronics Engineers
<b>mm</b>	Millimeter
<b>POST</b>	Power-on Self-Test
<b>PWM</b>	Puls Width Modulation
<b>UART</b>	Universal Asynchronous Receiver Transmitter
<b>USB</b>	Universal Serial Bus

## **Tabellenverzeichnis**

## **Formelverzeichnis**

1	Übertragungsfunktion des P-Glieds . . . . .	10
2	Übertragungsfunktion des I-Glieds . . . . .	11
3	Übertragungsfunktion des D-Glieds . . . . .	11
4	Übertragungsfunktion des PID-Glieds . . . . .	12
5	Übertragungsfunktion des PT-Glieds . . . . .	12
6	Übertragungsfunktion des I-Glieds . . . . .	41

## **Code-Verzeichnis**

1	Befehl zum Öffnen des <i>OverrideRCIn</i> -Headers . . . . .	23
2	Definition des Struct <b>MD5Sum</b> für das Template <i>OverrideRCIn</i> . . . . .	23

# 1 Einleitung

“Über den Wolken muss die Freiheit wohl grenzenlos sein.“

aus dem Lied *Über den Wolken* von Reinhard Mey, 1973 [7]

Menschen sehnten sich seit jeher, fliegen zu können. **BESCHREIBUNG!** *irgendwas mit Ikarus*

**BESCHREIBUNG!** *Erste motorisierte Flüge - 1903 Gebrüder Wright*

Dronen als zukünftiges Verkehrsmittel für kurzstreckentransporte

Dronen auf dem Mars, Ingenuity Feb 2021

*Anmerkung:* Für diese Ausarbeitung werden fachliche Begrifflichkeiten vorausgesetzt, sofern diese nicht innerhalb der Ausarbeitung erklärt werden. Sind Begriffe für Lesende unklar, sind diese an geeigneter Stelle nachzuschlagen. Auf eine voranstehende Erklärung aller genutzten und nicht näher erklärten Begriffe wird in dieser Ausarbeitung verzichtet, um den Rahmen dieser Arbeit einhalten zu können.

## 2 Problemstellung

Fluggeräte jeder Ausführung können durch Umwelteinflüsse von ihrer Position abgetrieben werden (vgl. [8]). Während der Versuchsdurchführung von Studierenden der DHBW Karlsruhe an einem Quadrokopter hat sich gezeigt, dass sich das Halten einer Position für Piloten mit geringer Erfahrung als schwierig erweist. Beschädigungen der in Laborversuchen eingesetzten Hardware ist zu vermeiden. Die Versuchsdurchführung *Höhenregelung*<sup>1</sup> soll für die Studierenden dahingehend vereinfacht werden, alsdass der eingesetzte Quadrokopter die horizontale Bewegung selbstständig regelt.

Zu entwickeln ist eine Positionsregelung auf Basis der verfügbaren Beschleunigungswerte der Drohne. Optional kann die Regelung um ein bildgestütztes System erweitert werden.

Für die Positionsregelung können unterschiedliche Modi entwickelt werden:

- Halten der Position nach einer manuellen Positionsänderung
- Anfliegen von vorgegebenen Positionen. Hierbei ist ein Überschwingen möglichst zu vermeiden.

Durch die Anschaffung eines neuen Quadroopters erweitert sich die Aufgabenstellung um den Aufbau des Bausatzes und die Inbetriebnahme des Quadroopters, an dem die Positionsregelung implementiert werden soll. Die Anpassung der Versuchsbeschreibung an die geänderte Hardware ist gewünscht.

Eine tabellarisch angelegtes Lastenheft ist **BESCHREIBUNG!** *im Kapitel Anhang...* einzusehen.

---

<sup>1</sup>Bei dem Laborversuch *Höhenregelung* sollen Studierende eine ROS-Node erstellen, welche eine konstante Flughöhe des Quadroopters ermöglicht. Hierzu wird als Rückführungsgröße des Regelkreises die Abstandsmessung zwischen Quadroopter und der darunterliegenden Ebene eingesetzt.

## 3 Methodik

### BESCHREIBUNG!

Analyse des Systems „Drohne“ (Geometrie, Regel- und Messgrößen) Analyse möglicher Messgrößen (Beschleunigungssensorik, evtl. Bilddaten) Analyse realer Messdaten aus Flugversuchen Modellierung des Regelsystems Entwurf eines geeigneten Reglers Implementierung des entworfenen Reglers Testen und optimieren der Regelparameter

### 3.1 Begriffe

In diesem Kapitel sollen Begrifflichkeiten definiert werden, welche im allgemeinen Sprachgebrauch mehrdeutig belegt sind.

### BESCHREIBUNG!

Drohne	Quadrokopter
--------	--------------

### 3.2 Eingesetzte Software

In diesem Kapitel sollen die für diese Projektarbeit eingesetzten Softwares genannt, um eine Reproduktion der Ergebnisse gewährleisten zu können.

*Anmerkung:* Die Ausführungen der eingesetzten Software beziehen sich auf den Umgang mit der Drohne *ArDrone 2.0*. Für die Interaktion mit der Drohne *Clover 4.20*, welche zum Projektbeginn eingesetzt wurde, wurde aktuellere Software eingesetzt.

#### 3.2.1 ROS

Das *Robot Operating System (ROS)* ist eine *Open Source* Bibliothek, welche dem Nutzer eine modulare Architektur ermöglicht. Hierbei kommunizieren *Nodes* mittels *Messages* miteinander.(vgl. [11])

Die *ROS* Versionen werden jeweils mit Namen versehen, wobei die Anfangsbuchstaben der Versionen der alphabetischen Nummerierung entsprechen. In diesem Projekt wurde die *ROS*-Version *Indigo* eingesetzt. Diese Version wird auf Grund der Anforderungen des *Driver-Package ardroner\_autonomy* eingesetzt. Der Einsatz der aktuellsten *ROS*-Version *noetic* in Zusammenspiel mit *Ubuntu 20.04* konnte das gewünschte Ergebnis nicht erzielen.

Die *ROS-Nodes* wurden mittels *catkin* kompiliert. Für eine korrekte Kompilierung müssen *CMakeList.txt*-Dateien den Befehl `add_compile_options(-std=c++11)` beinhalten

ten.

### **3.2.2 Ubuntu - Betriebssystem**

Als Betriebssystem wird das *UNIX*-basierte Betriebssystem *Ubuntu* auf einer *virtuelle Maschine* in der Version 14.04 eingesetzt. Die Auswahl dieser Version gründet auf den Anforderungen der *ROSIndigo*-Version.(vgl. [12])

Die *virtuelle Maschine* wird durch die Software *VMWare Workstation 15 Pro* virtualisiert.

### **3.2.3 Visual Studio - IDE**

Aus der Präferenz des Autors heraus wurde der Code mittels *Visual Studio 2019 (Community Editon)* erstellt, getestet und anschließend in den *catkin workspace* migriert.

## 4 Quadroopter als System

### 4.1 Orientierungsmerkmale

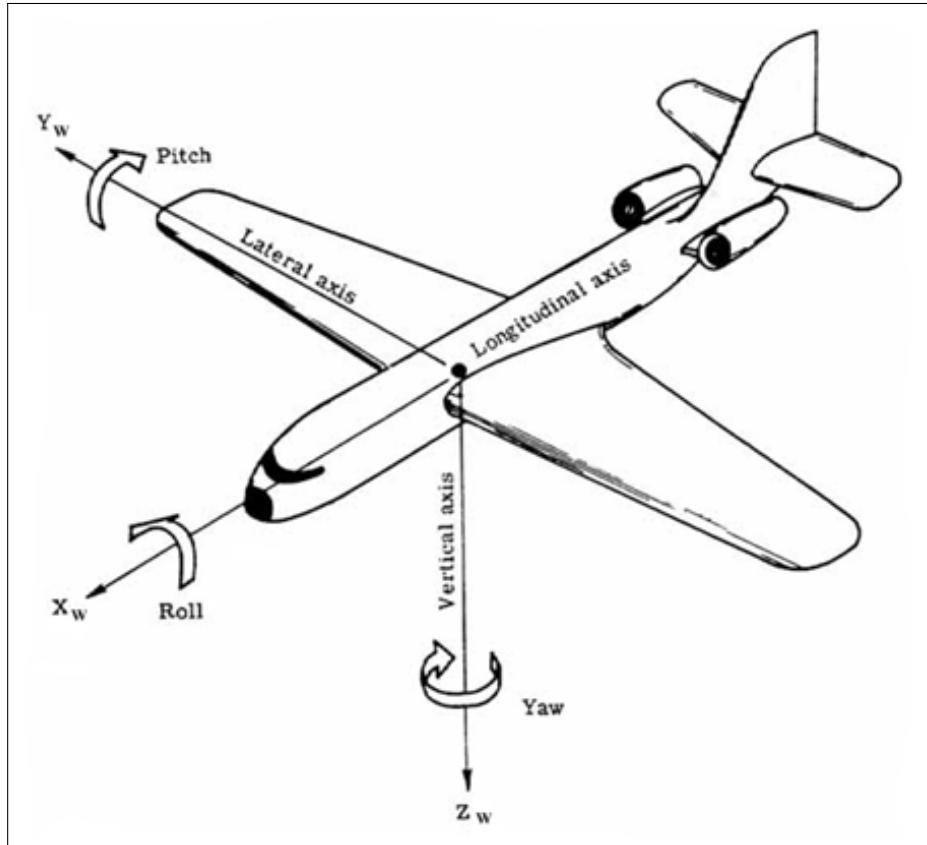


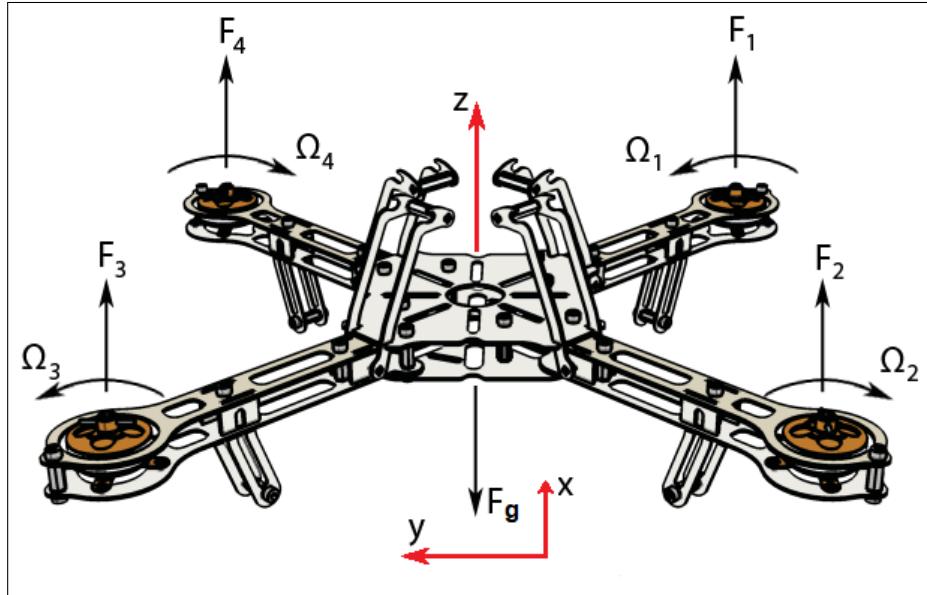
Abbildung 1: ORIENTIERUNGSMERKMALE EINES FLUGOBJEKTS [15]

Abbildung 1 zeigt die Orientierung eines beliebigen Flugobjektes am Beispiel eines Flugzeugs. Die Bezeichner beziehen sich auf ein kartesisches Koordinatensystem und beschreiben jeweils die Rotation um eine Raumachse.

### 4.2 Geometrien

Bei Quadrooptern handelt es sich um Fluggeräte mit vier Rotoren, welche horizontal angebracht sind. Der Auftrieb wird somit unmittelbar durch die Rotoren induziert.

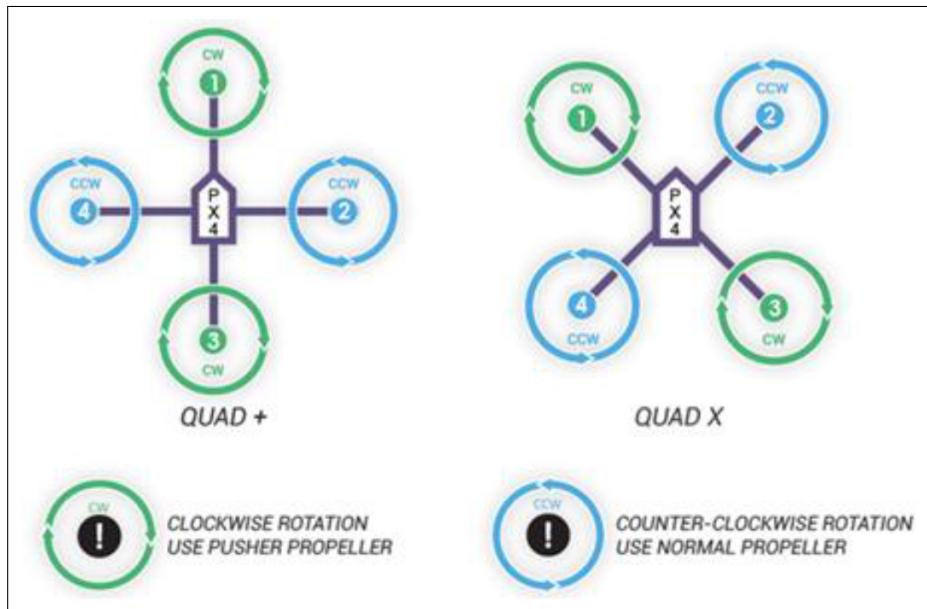
Das Drehmoment der Rotoren des Quadroopters muss sich aufheben können, um eine unkontrollierbare Rotation um die z-Achse vermeiden zu können. Um die Agilität des Quadroopters beizubehalten zu können, sind die Drehrichtungen der Rotoren zu alternieren. (vgl. [1]) Abbildung 2 (Seite 6) deutet die Kräfte und aus der Rotationsgeschwindigkeit der Rotoren ableitbare Momente eines Quadroopters an. Dies soll als Einleitung für das Kapitel 4.3 *Freiheitsgrade* (Seite 7) dienen und wird an dieser Stelle nicht näher vertieft.



**Abbildung 2:** WIRKENDE KRÄFTE AM QUADROKOPTER IN x-ANORDNUNG (VGL. [2])

Abbildung 2 zeigt die wirkenden Kräfte an einem Quadrocopter in **x**-Anordnung. Die Achsen des Koordinatensystems sind in **rot** markiert. Alle mit *F* markierten Pfeile deuten Kräfte an. Die mit  $\Omega$  markierten gebogenen Pfeile zeigen die Rotationsgeschwindigkeit der einzelnen Rotoren.

Die Rotoren von Quadrokoptern können in zwei verschiedenen Anordnungen angebracht werden (siehe Abbildung 3 (Seite 6)).



**Abbildung 3:** GEOMETRIEN VON QUADROKOPTERN [1]

### +-Anordnung

In der +-Anordnung befinden sich die Rotoren auf den Objekt-Achsen des Quadroko-

pters. Die Nummerierung der Rotoren erfolgt gemäß Abbildung 3 (Seite 6) (links) im Uhrzeigersinn beginnend mit dem Rotor auf der positiven x-Achse.

### **x-Anordnung**

Die **x**-Anordnung scheint nach online Recherchen weiter verbreitet. Eine mögliche Begründung findet sich in der weniger verdeckten Sicht für Frontkameras oder andere Anbaugeräte.

## **4.3 Freiheitsgrade**

Einem Quadrokopter können sechs Freiheitsgrade zugewiesen werden, jeweils drei der Position und der Orientierung im Raum.

Nachfolgend wird physikalisch begründet, weshalb nur vier der genannten sechs Freiheitsgrade unabhängig regelbar sind.

Wird eine Kraft in der horizontalen Ebene induziert, beschleunigt der Quadrokopter entlang dieser Kraft. Eine solche Kraft wird durch eine Neigung um die x- beziehungsweise y-Achse hervorgerufen (siehe Abbildung 4 (Seite 8)). Aus den genannten Umständen lässt sich ableiten, dass sich eine geänderte Orientierung um die x- beziehungsweise y-Achse auf die Position des Quadroopters auswirkt.

## **4.4 Pose**

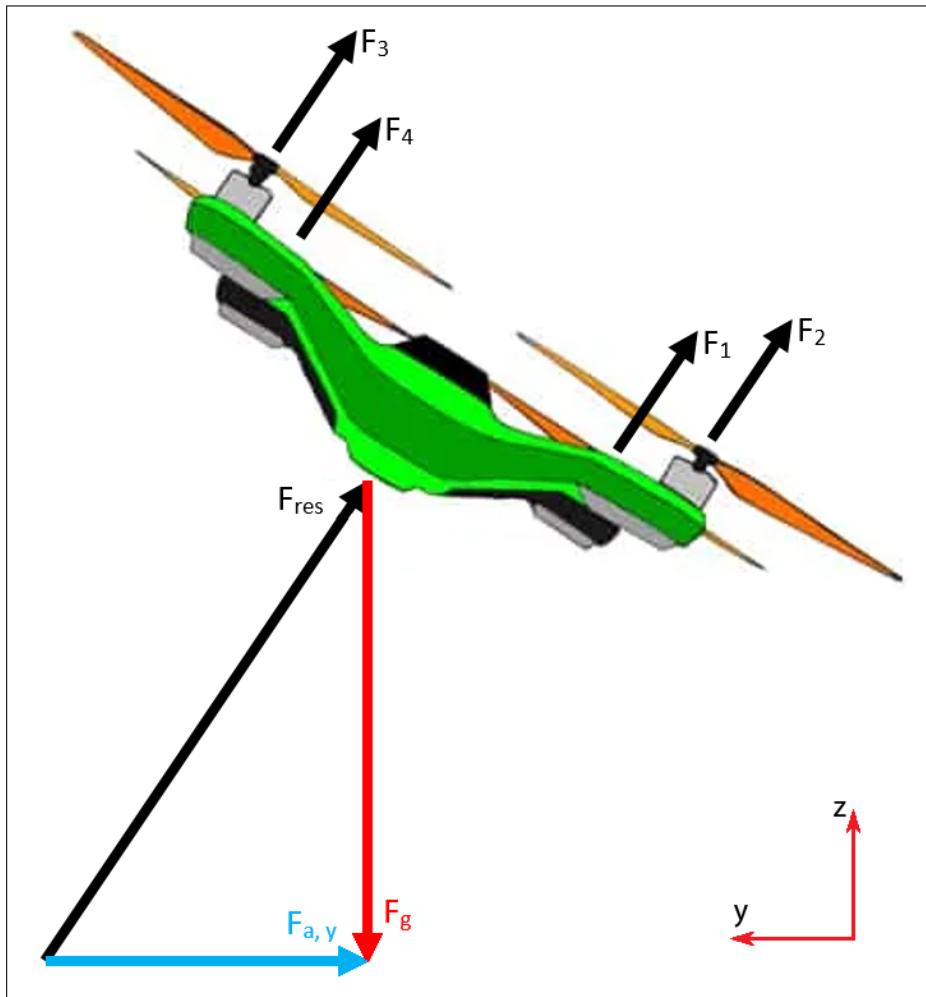
Eine Pose ist die Positions- und Lagebeschreibung in einem Raum. **BESCHREIBUNG!**  
*quelle*

### **4.4.1 lokale Pose**

**BESCHREIBUNG!**

### **4.4.2 globale Pose**

**BESCHREIBUNG!**



**Abbildung 4:** WIRKENDE KRÄFTE FÜR GEROLLTEN QUADROKOPTER (VGL. [13])

Abbildung 4 zeigt die auf einen Quadrokopter wirkenden Kräfte, wenn sich dieser in einer gerollten Orientierung befindet. Hier entspricht der Betrag der Kraft entlang der z-Achse der Gewichtskraft.

Die Kraft  $F_{res}$  wird mit der Gewichtskraft  $F_g$  überlagert. Die hieraus resultierende Kraft kann in drei Kräfte aufgeteilt werden, welche parallel zu den kartesischen Achsen angeordnet sind. Hieraus berechnet sich die Beschleunigung, welche auf den Quadrokopter wirkt.

## 5 Regelsysteme

In der Norm *DIN IEC 60050-351 (Internationales Elektrotechnisches Wörterbuch – Teil 351: Leittechnik)*, welche Begriffe der Regelungstechnik definiert, wird der Begriff *Regelung* wie folgt beschrieben: [6]

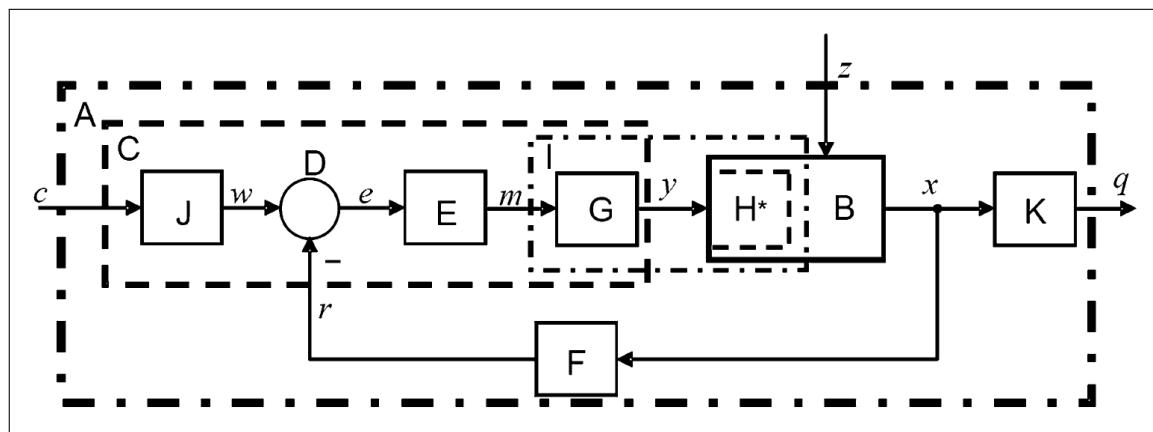
„Vorgang, bei dem fortlaufend eine variable Größe, die Regelgröße, erfasst, mit einer anderen variablen Größe, der Führungsgröße, verglichen und im Sinne einer Angleichung an die Führungsgröße beeinflusst wird.“

*Anmerkung:* Kennzeichen für das Regeln ist der geschlossene Wirkungsablauf, bei dem die Regelgröße im Wirkungsweg des Regelkreises fortlaufend sich selbst beeinflusst.“

In diesem Kapitel sollen regelungstechnische Grundlagen beschrieben werden.

### 5.1 Regelkreis

Im Allgemeinen grenzen sich die Konzepte *steuern* und *regeln* durch die Eigenschaft der Rückkopplung des Systems ab. Gesteuerte Systeme wirken lediglich auf Aktoren, wobei geregelte Systeme die Abweichung des *Ist*-Zustandes vom *Soll*-Zustand ermitteln und eine geeignete Veränderung erzielen sollen. (vgl. [6])



A	Regelungssystem	K	Bildung der Aufgabengröße
B	Regelstrecke	c	Zielgröße
C	Regeleinrichtung	w	Führungsgröße
D	Vergleichsglied	e	Regeldifferenz
E	Regelglied	m	Reglerausgangsgröße
F	Messglied	y	Stellgröße
G	Steller	z	Störgröße
H*	Stellglied	x	Regelgröße
I	Stelleinrichtung	q	Aufgabengröße
J	Führungsgrößenbildner	r	Rückführgröße

Abbildung 5: ALLGEIMEINES SCHEMA EINES REGELKREISES [6]

Abbildung 5 zeigt **BESCHREIBUNG!**

## 5.2 Arten von Reglergliedern

In diesem Kapitel sollen grundlegende Bausteine der Regelungstechnik beschrieben werden. Diesbezüglich erhebt dieses Kapitel keinen Anspruch auf Vollständigkeit. Auf eine detaillierte Beschreibung, welche unter anderem das Zeitverhalten betrachten, soll hier außen vor gelassen werden. Als weitere Einschränkung soll sich die Betrachtung der Bausteine ausschließlich auf zeitdiskrete Systeme beziehen.

### 5.2.1 P-Glied

Die Abkürzung P in der Bezeichnung *P-Glied* steht für *proportional*. Hierbei wird eine die Eingangsgröße um einen Faktor  $k_P$  verstärkt. **BESCHREIBUNG!** *Elektrotechnik oder sehr einfacher Regler* **BESCHREIBUNG!** *dauerhafte Regelabweichung*.

$$y_k = k_P * x_k \quad (1)$$

ÜBERTRAGUNGSFUNKTION DES P-GLIEDS

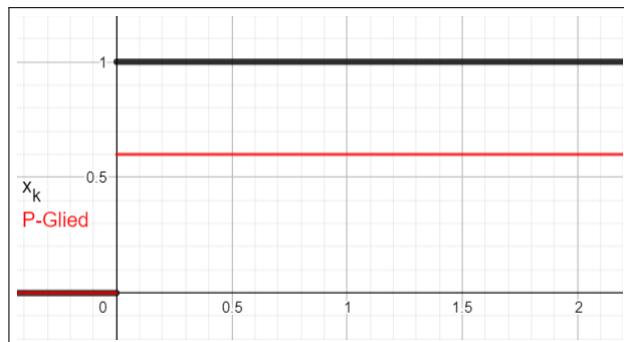


Abbildung 6: SPRUNGANTWORT EINES P-GLIEDS

Abbildung 9 zeigt charakteristischen Sprungantwort des *P-Glieds*. Hier wurde der Parameter  $k_P$  mit dem Wert 0,6 gewählt, um die Sichtbarkeit in der Abbildung zu erhöhen.

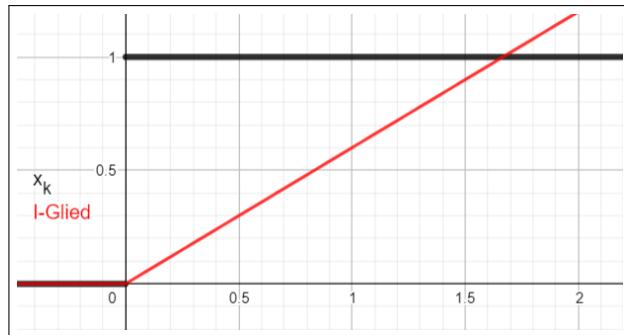
**BESCHREIBUNG!** [3] Seite 220

### 5.2.2 I-Glied

Wie sich aus dem Namen des *Integral-Glieds* ableiten lässt, bildet dieser Baustein ein Integral über dem Eingangssignal. Hierdurch können physikalische Umrechnungen oder Prozesse abgebildet werden. Als Beispiele kann an dieser Stelle der Füllstand eines Tanks oder die Ermittlung einer Geschwindigkeit aus Beschleunigungsdaten genannt werden.

$$y_k = y_{k-1} + k_I * x_k * \Delta t \quad (2)$$

### ÜBERTRAGUNGSFUNKTION DES I-GLIEDS



**Abbildung 7:** SPRUNGANTWORT EINES I-GLIEDS

### 5.2.3 D-Glied

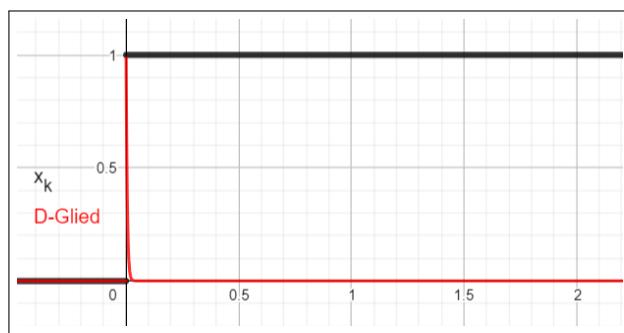
In der betrachteten Literatur spielen D-Glieder keine signifikante Rolle. (vgl. [3] und [4]) Dennoch sollen sie hier als Grundlage für *PID*-Glieder beschrieben werden.

Bei einem *D-Glied* handelt es sich um ein differentielles Glied. Somit lässt sich hiermit die Veränderung des Eingangssignals ermitteln. (vgl. [9]) **BESCHREIBUNG!** *Diese Quelle iO?*

Das Ausgangssignal des *D-Glieds* kann durch folgende Formel berechnet werden, welche sich aus der Diskretisierung der Differenzationsfunktion ergibt.

$$y_k = k_D * \frac{x_k - x_{k-1}}{\Delta t} \quad (3)$$

### ÜBERTRAGUNGSFUNKTION DES D-GLIEDS



**Abbildung 8:** SPRUNGANTWORT EINES D-GLIEDS

Abbildung 8 zeigt die Sprungantwort eines *D-Glieds*. Diese ist ein Ausschlag, welcher für den Zeitschritt anhält, in dem sich die ansteigende Flanke des Sprungs befindet.

$$y_k = y_{k(P)} + y_{k(I)} + y_{k(D)} \quad (4)$$

ÜBERTRAGUNGSFUNKTION DES PID-GLIEDS

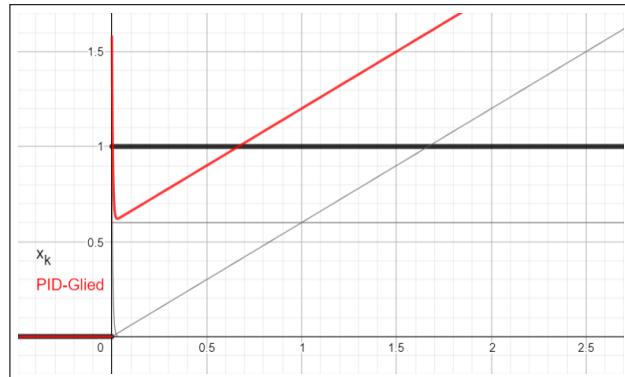


Abbildung 9: SPRUNGANTWORT EINES PID-GLIEDS

Abbildung 9 zeigt die Sprungantwort eines *PID-Glieds*. Hierbei setzt sich das Signal als Summe des P-, des I- und des D-Glieds zusammen, welche in grau angedeutet sind.

#### 5.2.4 PID-Regler

#### 5.2.5 PT-Glied

**BESCHREIBUNG!** *Dient der Abbildung realer Systeme, welche sich an einen Zustand anpassen müssen. Drehzahlen, Temperaturen etc.*

$$y_k = k_D * \frac{x_k - x_{k-1}}{\Delta t} \quad (5)$$

ÜBERTRAGUNGSFUNKTION DES PT-GLIEDS

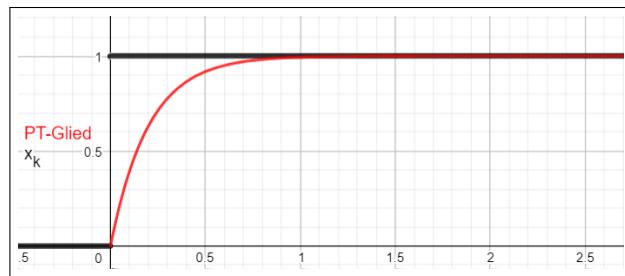


Abbildung 10: SPRUNGANTWORT EINES PT-GLIEDS

Abbildung 10 zeigt **BESCHREIBUNG!**

#### 5.2.6 Tt-Glied

### 5.3 Stabilität

Die Stabilität eines Regelkreises beschreibt, ob der Regelkreis zum Schwingen neigt. Hierfür wird die Übertragungsfunktion des offenen Regelkreises analysiert. Das Schwin-

gen eines zu regelnden Systems ist zu vermeiden. (vgl. [3] und [10])

An dieser Stelle soll auf eine Beschreibung der Analyse der Stabilität von Regelkreisen verzichtet werden, um den Rahmen dieser Projektarbeit einhalten zu können. Als Schlagworte für interessierte Lesende sei hier das *Bode-Diagramm* und die *Ortskurve* genannt (vgl. [3] und [10]).

# 6 Positionsregelung von Quadrooptern

## 6.1 Positionsregelung von Quadrooptern mittels Pose

Die Freiheitsgrade einer Drohne können grundsätzlich separat geregelt werden. Jedoch ist die Kraft in z-Richtung abhängig von der Ausrichtung (Roll und Pitch) des Quadroopters. Dies ist aus Abbildung 4 (Seite 8) ersichtlich. Der Quadroopter wird als Folge einer Veränderung der Roll- und Pitch-Winkel an Höhe verlieren. Der Höhenregler wird diese Veränderung nachführen. Um ein Nachregeln des Quadroopters bei einer Orientierungsänderung in der Horizontalen abschwächen zu können, kann der veränderte Schubbedarf aus den gemessenen oder berechneten Roll- und Pitch-Winkeln abgeleitet und zu dem Schubsignal aufsummiert werden.

Es ist zu beachten, dass sie Parallelität des Quadroopter-Koordinatensystem zum lokalen beziehungsweise globalen Koordinatensystem bei einer Rotation um die z-Achse verloren geht. Hierzu ist die translativen Bewegung entgegengesetzt dieser Rotation zu transformieren. Somit kann die Position bezogen auf das lokale beziehungsweise globale Koordinatensystem ermittelt werden. **BESCHREIBUNG!** *Quelle? evtl Vorlesung?*

## 6.2 Erzeugung von Posen aus Beschleunigungsdaten

### 6.2.1 Berechnung

Für die nachfolgend beschriebene Berechnung der Position einer Pose wird angenommen, dass die Orientierung des Quadroopters bekannt ist. Es ist zu berücksichtigen, dass die Beschleunigungssensorik fest im Quadroopter verbaut ist. Der resultierende Kraft-Vektor ist in ein Koordinatensystem zu transformieren, dessen xy-Ebene parallel zu der Start-Orientierung des Quadroopters ausgerichtet ist. **BESCHREIBUNG!** *Quelle?*

Anschließend kann der Kraft-Vektor entlang der Achsen des Quadroopter-Koordinatensystems aufgeteilt werden und die Beschleunigungen auf den Quadroopter zweifach integriert werden. Dieses Vorgehen gelingt lediglich bei idealen Daten. Da in Störungen auf das Signal wirken können, sind geeignete Methoden der Signalverarbeitung anzuwenden. **BESCHREIBUNG!** *Quelle?*

### 6.2.2 Signalaufarbeitung

Nachfolgend soll auf die Einflüsse einer Messung eingegangen werden, welche sich die Qualität der ermittelten Pose auswirken. Hierzu werden verschiedene Störgrößen am Beispiel eines Szenarios betrachtet und mögliche Gegenmaßnahmen gezeigt.

An dieser Stelle sei angemerkt, dass bei dem Vorgehen der Ansatz für online-Datenverarbeitung

genutzt wurde. Hierbei fließen ausschließlich Daten ein, welche maximal den aktuell betrachteten Zeitstempel aufweisen.

### veranschaulichendes Szenario

Im gewählten Szenario wird die z-Achse beziehungsweise Flughöhe als Beispiel betrachtet. Die Kurve setzt sich aus idealisierten Teilstücken zusammen (siehe Abbildung 11 (Seite 16), oben):

- Initialisierung: horizontale Gerade
- Abflug: negativer Cosinus ( $1/2$  Schwingung)
- Schweben: horizontale Gerade
- Landung: zwei entgegengesetzte Parabeln<sup>2</sup>
- Abschalten: horizontale Gerade

Die Beschleunigungsdaten wurden aus der genannten Kurve mittels zweifacher zeitdiskreter Differentiation gebildet (siehe Abbildung 11 (Seite 16), unten).

Durch dieses Vorgehen ist sichergestellt, dass ein Vergleich zwischen originaler Kurve und den rekonstruierten Verläufen gebildet werden kann. In diesem Zusammenhang wurde sichergestellt, dass eine Rekonstruktion ohne aufgeprägte Störgrößen möglich ist. (vgl. [4]) (Seite 123)

### Rauschen

Rauschen eines Signals kann durch die Quantisierung von analogen Daten auftreten (vgl. [4]) (Seite 100, Abbildung 4.5) oder in dem Anwendungsfall einer Beschleunigungsmessung auch von Vibrationen des Systems beeinflusst werden. Es ist darauf zu achten, dass Überläufe von digitalen Speichertypen vermieden werden, da diese einen signifikanten Anteil an rauschenden Signalen ausmachen können. (vgl. [4]) (Seite 118)

**BESCHREIBUNG!** *was dagegen tun? Eigentlich Mittelwert bilden. Aber Mittelwert glättet auch einzelne Werte. Also Höhere Abtastrate!*

### Ausreißer

**BESCHREIBUNG!** *Erklären, wie diese entstehen*

1977 wurde von Tukey beschrieben, dass ein gleitender Median für die Glättung einer Datenreihe eingesetzt werden kann (vgl. [5]) (Seite 210). Im Buch wird ein Median über

---

<sup>2</sup>Die Parabeln sind so angeordnet, dass der Berührungszeitpunkt zeitlich in der Mitte der Landungsphase liegt. Da die Parabeln einen identischen Streckfaktor besitzen, entspricht der Übergang zwischen den Parabeln einer knickfreien Kurve.

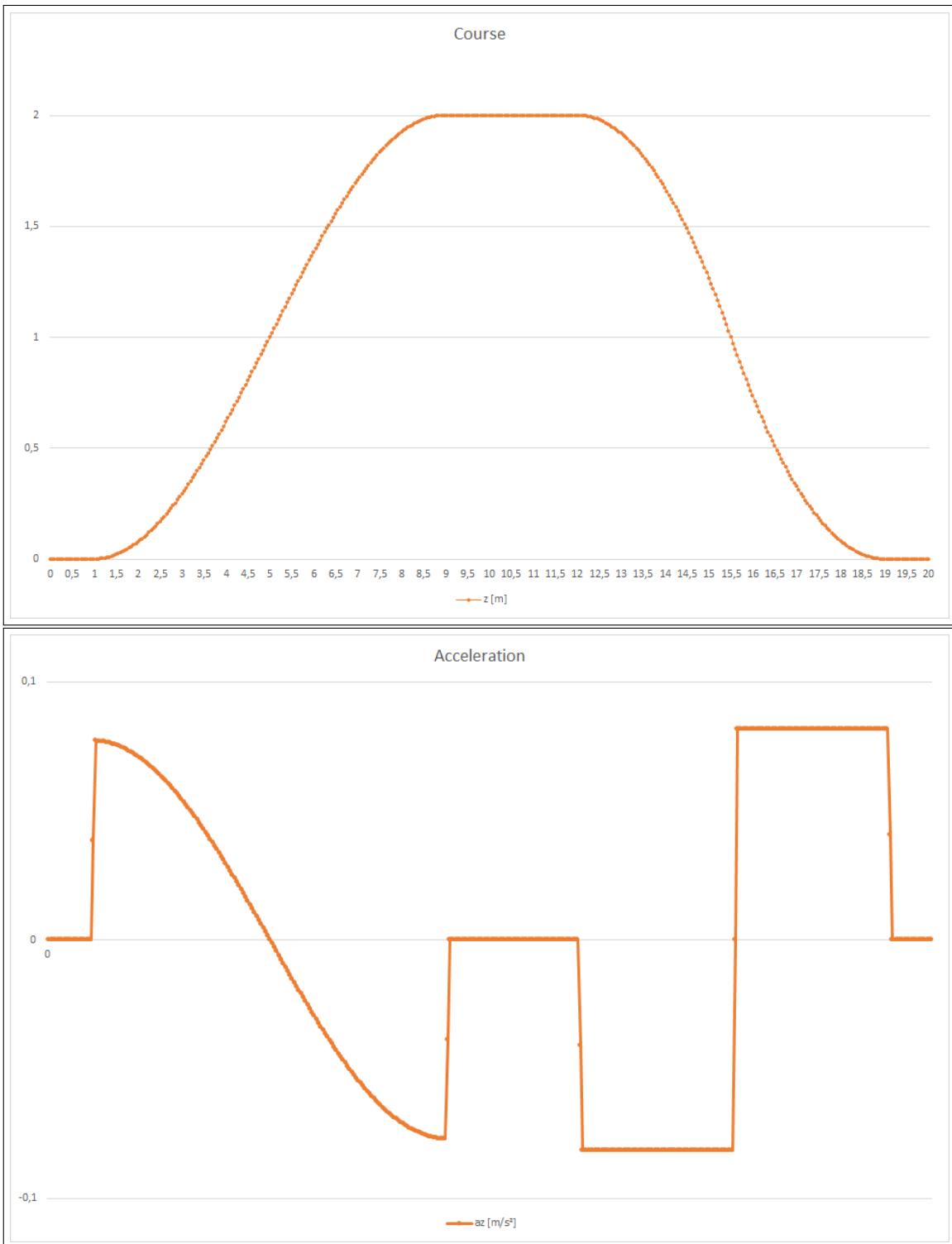
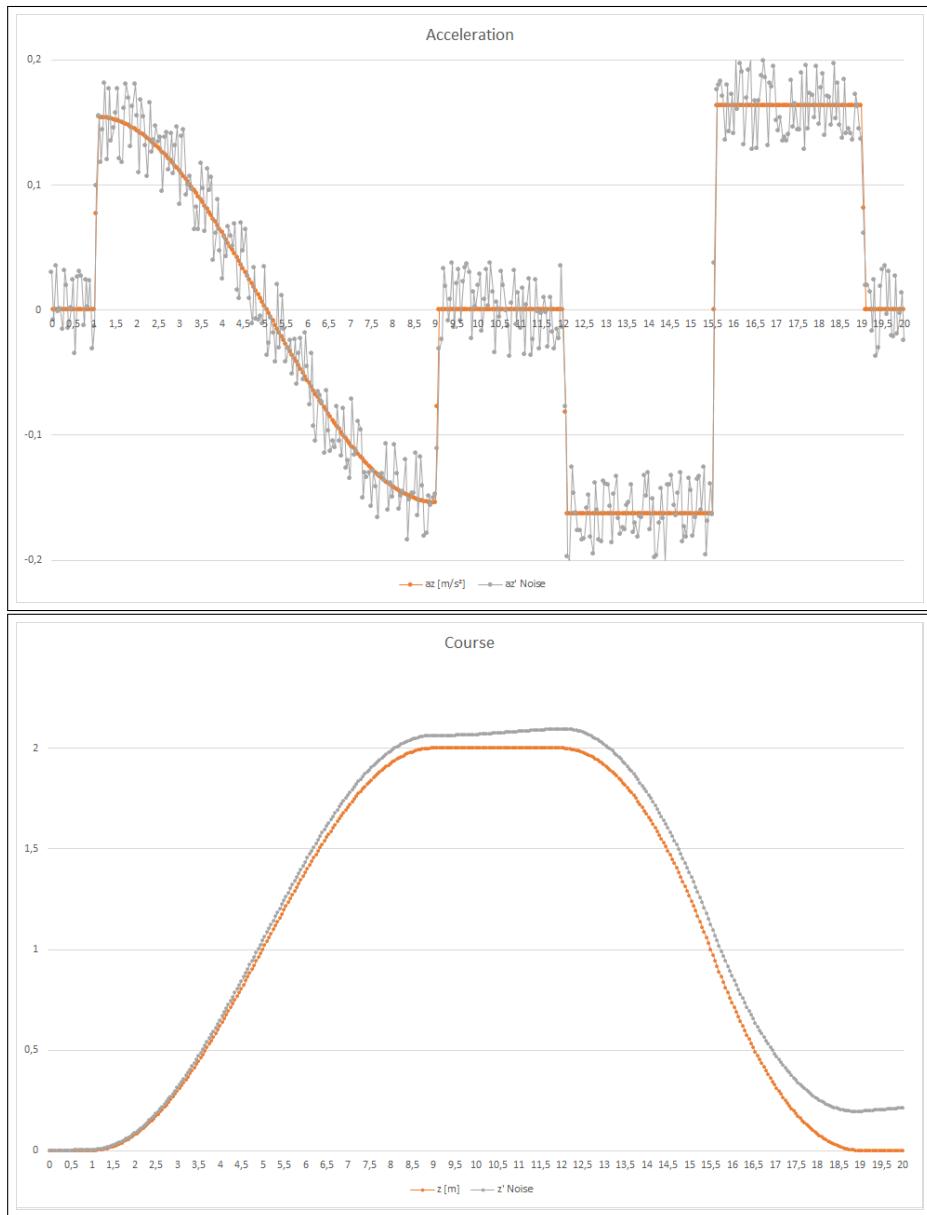


Abbildung 11: ORIGINALKURVE DES Szenarios

In Abbildung 11 sind der idealisierte Höhneverlauf (oben) und die daraus abgeleitete Beschleunigung (unten) aufgetragen. Sofern nicht anderweitig beschrieben, wird im Folgenden eine Abtastrate von 20 Datenpunkten pro Sekunde angenommen.

drei Werte vorgeschlagen. Eine Erweiterung der Anzahl der einbezogenen Werte ist möglich, wobei zu berücksichtigen ist, dass die Median-Funktion bei einer geraden Anzahl

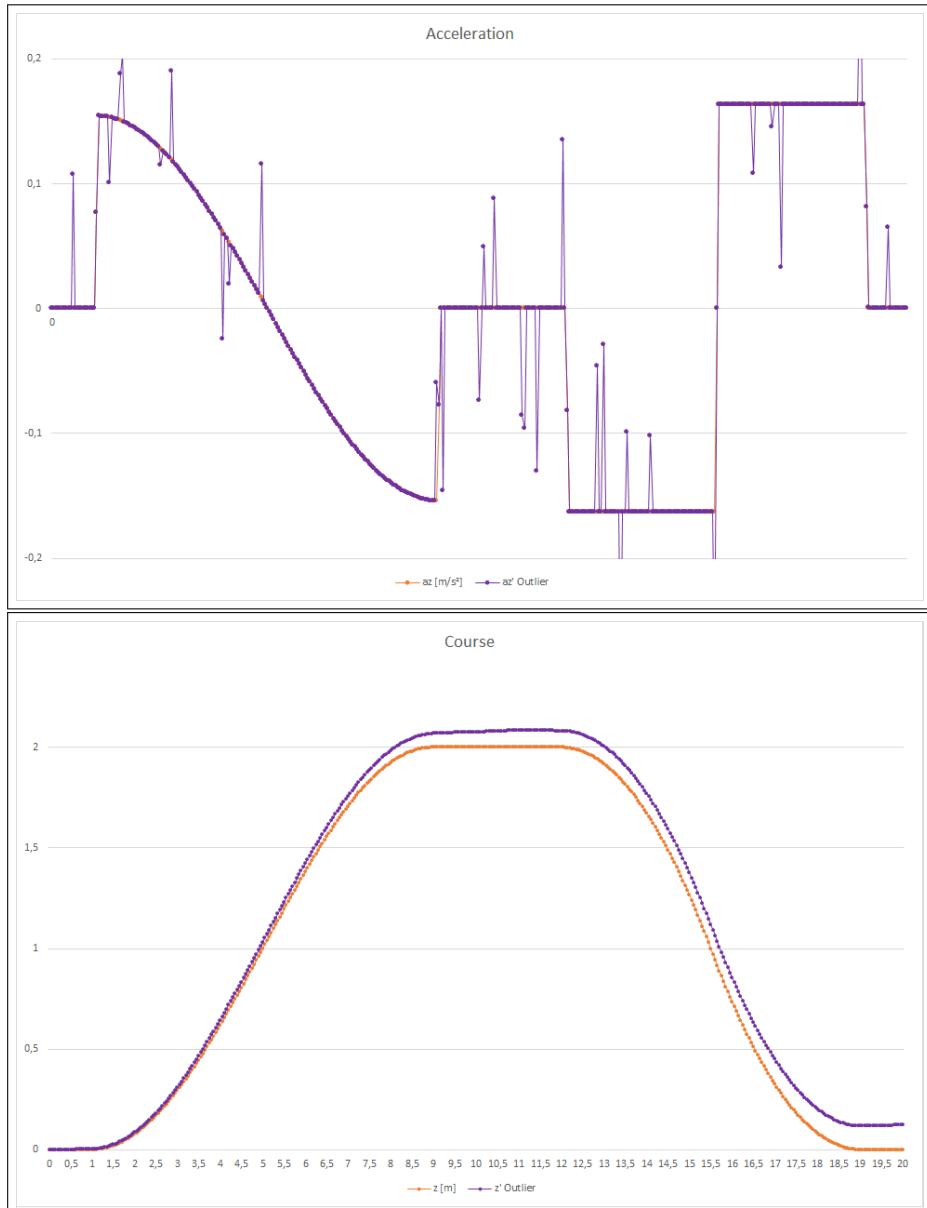


**Abbildung 12:** EINFLUSS VON SIGNALRAUSCHEN

Um das aufgeprägte Rauschen zu erzeugen wurden Zufallszahlen im Bereich  $[-1, 1]$  erzeugt und diese mit dem Faktor einer angenommenen absoluten Genauigkeit (0.375% für einen Wertebereich  $[-15 \text{ m/s}^2, 15 \text{ m/s}^2]$ ) und einer angenommenen relativen Genauigkeit (0.5%) verrechnet.

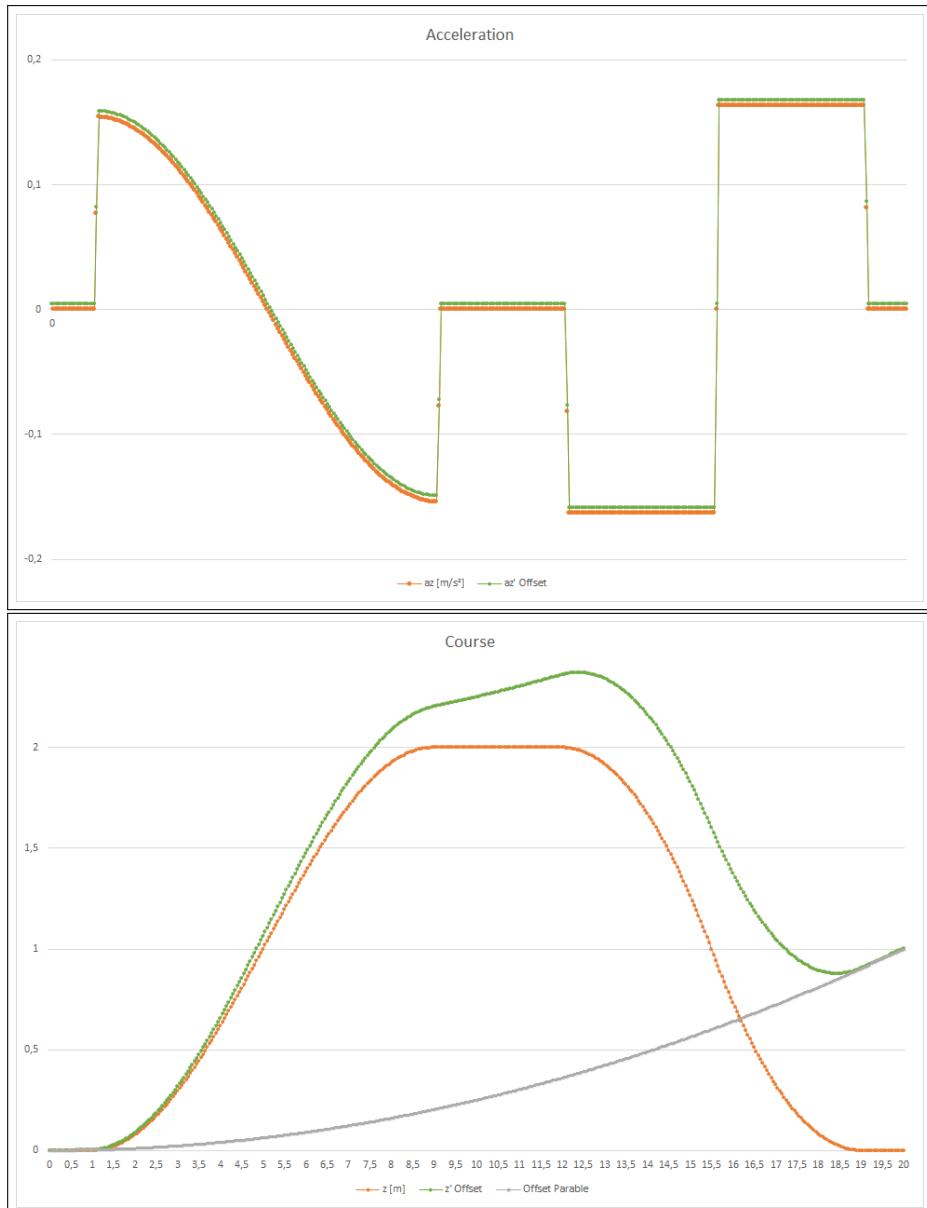
an Werten den Durchschnitt der mittleren Werte bildet. **BESCHREIBUNG!** *Das ist allgemeinbildung in Mathe. Es gibt zwar ein Buch (H. Cramér, "Mathematical methods of statistics", Princeton Univ. Press (1946) –Info von [https://encyclopediaofmath.org/wiki/Median\\_\(in\\_statistics\)](https://encyclopediaofmath.org/wiki/Median_(in_statistics))), aber da kommt man extrem schwer ran. reicht es, hier nur das Buch zu nennen, ohne explizite Seitenzahl?*

### konstante Nullpunkt-Abweichung



**Abbildung 13:** EINFLUSS VON AUSREISSERN

Die Höhe der aufgeprägten Ausreißer wurde mittel einer Datenreihe von Zufallszahlen im Bereich  $[-1, 1]$ , verrechnet mit dem beliebigen Faktor 0.125, generiert. Für die zeitliche Lage wurde eine Datenreihe von Zufallszahlen im Bereich  $[0, 1]$  mit einem Schwellwert von 0.925 verglichen. Daraus folgt, dass etwa 7.5% der Datenwerte einen zusätzlichen Wert aufgeprägt bekommen.



**Abbildung 14:** EINFLUSS VON KONSTANTER NULLPUNKT-ABWEICHUNG

Die hier gezeigte Abweichung der Beschleunigungsdaten beträgt  $0.0025 \text{ m/s}^2$ . Die Superposition der rekonstruierten Kurve entspricht einer in grau dargestellten Parabel mit einem Streckfaktor von  $\frac{\Delta a}{2}$ .

## 7 Eingesetzte Hardware

In dieser Projektarbeit wurden zwei unterschiedliche Quadrokopter eingesetzt.

Die zu Beginn der Projektarbeit eingesetzte Quadrokopter des Herstellers *COEX* wird in diesem Kapitel ebenfalls beschrieben. Vorrangig sollen hier Erkenntnisse über das Verhalten dieser Drohne und mögliche Lösungswege der Problemstellung erläutert werden.

Nach Austausch der Hardware wurde die *ArDrone 2.0* des Herstellers *Parrot* genutzt. Dieser Quadrokopter wurde bereits erfolgreich durch die DHBW Karlsruhe im Zuge der Labor-Vorlesung *Robotik* eingesetzt.

### 7.1 COEX Drohne

Bei dem für die DHBW Karlsruhe neu angeschafften Quadrokopter handelt es sich um das Modell *Clover 4.20* des Unternehmens *Copter Express (COEX)*.

Das Modell *Clover 4.20* wurde vom Hersteller zur Ausbildung und Forschung an Quadrokoptern entwickelt. Das Modell besitzt einen Rahmen, welche die Rotoren bei Kollisionen schützen soll.

Interner Flight Controller, ROS Kommunikation via Pie 4.

Probleme bei Inbetriebnahme - Beispielprogramm des Herstellers bringt nicht das erwartete Ergebnis.

#### 7.1.1 Control Stack

Als *Flight Controller* wird das Modell *PX4 Racer* genutzt. Die Firmware, sowie weitere Software zur Interaktion mit der Drohne *Clover 4.20* werden von *Dronecode Foundation* bereitgestellt.

Die Anbindung an *ROS* wird durch einen *Raspberry Pie 4* realisiert. Hierbei wird der On-Board Computer als *roscore* genutzt.

#### 7.1.2 Funkfernsteuerung

s-Bus ??

#### 7.1.3 Sensorik

BESCHREIBUNG! *EIn bisschen Einleitung*.

- Gyroskop
- Laser-Abstandsmessung zum Boden
- GPS
- Bodenkamera

#### 7.1.4 Aufbau des Bausatzes

##### Aufbau

BESCHREIBUNG! *Bilder vom Aufbau*



**Abbildung 15:** AUFBAU DES BAUSATZES FÜR DIE DROHNE *Clover 4.20*

Abbildung 15 zeigt etappenweise den Aufbau des Bausatzes für die Drohne *Clover 4.20*. Hierbei ist die zeitliche Anordnung in der Collage zeilenweise zu interpretieren.

#### 7.1.5 Inbetriebnahme

##### Konfiguration des Flight Controllers

##### Testflug

(vgl. [14]) **BESCHREIBUNG!** *Verweis auf Example Code*

### 7.1.6 Mögliche Lösung der Aufgabenstellung

#### COEX-Package

Für die Interaktion mit der *COEX*-Drohne wurden diverse Klassen erstellt, um einzelne Aspekte der Interaktion mit der Drohne umsetzen zu können (siehe Kapitel 9.4.6 *coex Package* (Seite 34)).

Nach dem Wechsel auf die andere Drohne wurde die Aktualisierung dieses Package nicht weiter verfolgt. Sofern eine Einbindung der *COEX*-Drohne in die Ergebnisse dieser Projektarbeit durchgeführt werden soll, muss dieses Package entsprechend angepasst werden.

#### geeignete Topics

Mit dem *ROS-Topic* **blablab BESCHREIBUNG!** kann eine Regelung in der XY-Ebene umgesetzt werden. Die Höhenregelung kann mit `set_attitude/thrust` eingeführt werden. Somit wäre der Versuch für nachfolgende Studierende sehr viel sicherer und so.

#### BESCHREIBUNG!

Das *ROS-Topic* **/mavros/rc/override** erlaubt das Überschreiben der RC-Kanäle, somit können einzelne Eingaben durch den Controller übernommen werden. **BESCHREIBUNG!**

#### hilfreiche Literatur

Nachfolgend sollen Internetseiten genannt werden, welche die Einarbeitung in den Umgang mit der *COEX*-Drohne vereinfachen können.

- <https://clover.coex.tech/en/wifi.html>
- [https://clover.coex.tech/en/simple\\_offboard.html](https://clover.coex.tech/en/simple_offboard.html)
- [https://docs.px4.io/master/en/ros/mavros\\_offboard.html](https://docs.px4.io/master/en/ros/mavros_offboard.html)
- [https://docs.px4.io/master/en/flight\\_modes/offboard.html](https://docs.px4.io/master/en/flight_modes/offboard.html)
- [https://mavlink.io/en/services/manual\\_control.html](https://mavlink.io/en/services/manual_control.html)
- [http://wiki.ros.org/mavros#mavros.2FPlugins.manual\\_control](http://wiki.ros.org/mavros#mavros.2FPlugins.manual_control)
- [https://mavlink.io/en/messages/common.html#SET\\_POSITION\\_TARGET\\_LOCAL\\_NED](https://mavlink.io/en/messages/common.html#SET_POSITION_TARGET_LOCAL_NED)

Spezifische Verweise sind im Quellcode des *COEX*-Package hinterlegt.

### 7.1.7 Troubleshooting

**BESCHREIBUNG!** *Achtung: der Regler VRA muss so eingestellt sein, dass "manual Flight" verfügbar ist. Andersfalls ist ein Start der Drohne nicht möglich.*

#### Platinenfehler

**BESCHREIBUNG!** *Hier anmerken, dass das Problem bisher nicht behoben wurde => Wirkt sich nur auf LED-Streifen aus.*

#### Bus-System des RC Empfängers

**BESCHREIBUNG!** *RC-Empfänger gibt per default i-Bus aus, PX4 erwartet s-Bus.*

**BESCHREIBUNG!** *Bild von Oszilloskop*

#### Lösung

**BESCHREIBUNG!** *i-Bus und s-Bus werden durch Halten des Knopfes getauscht. Verweis auf Homepage angeben?*

#### md5-Sum des Topics OverrideRCIn

Nach erfolgreicher Kompilierung wird nachfolgender Laufzeitfehler ausgegeben, wenn sich ein `ros::Subscriber` oder ein `ros::Publisher` auf das *ROS-Topic /mavros/rc/override* anmeldet:

[ERROR] [1643616625.226584828]: Client [/mavros] wants topic /mavros/rc/override to have datatype/md5sum [mavros\_msgs/OverrideRCIn/73b27a463a40a3eda1f9fb1fc86d6f3], but our version has [mavros\_msgs/OverrideRCIn/fd1e1c08fa504ec32737c41f45223398].  
Dropping connection.

#### Lösung

Definition von

```
struct MD5Sum<::mavros_msgs::OverrideRCIn_<ContainerAllocator>>
aus
```

```
sudo nano /opt/ros/noetic/include/mavros_msgs/OverrideRCIn.h
```

#### Code 1: BEFEHL ZUM ÖFFNEN DES OVERRIDERCIN-HEADERS

```
template<class ContainerAllocator>
struct MD5Sum<::mavros_msgs::OverrideRCIn_<ContainerAllocator>>
{
    static const char* value()
    {
        return "73b27a463a40a3eda1f9fb1fc86d6f3";
    }

    static const char* value(const ::mavros_msgs::OverrideRCIn_<ContainerAllocator>&)
    {
```

```

        return value();
    }

    static const uint64_t static_value1 = 0x73b27a463a40a3edULL;
    static const uint64_t static_value2 = 0xa1f9fbb1fc86d6f3ULL;
};

```

**Code 2:** DEFINITION DES STRUCT `MD5SUM` FÜR DAS TEMPLATE `OVERRIDERCIN`

von dem *Raspberry Pie 4* kopieren und in der Definition des lokalen *ROS OverrideRCIn*-Headers ersetzen. Ein Versuch, den *Raspberry Pie 4* einem Update zu unterziehen, ist fehlgeschlagen. Somit ist eine unmittelbare Synchronisation der Nachrichten-Typen aufwändig.

## 7.2 Parrot Drohne

Um die Problematik der COEX Drohne zu umgehen, steigt diese Projektarbeit auf die Drohne um, welche die Idee für diese Studienarbeit ergeben hat. Hierbei handelt es sich um die Drohne *ArDrone 2.0*. Nachfolgend soll die Drohne und die eingebaute Sensorik näher beschrieben werden.

### 7.2.1 Geometrie

#### Anordnung der Rotoren

BESCHREIBUNG!

X

#### Abmessungen

BESCHREIBUNG!

### 7.2.2 Sensorik

- Beschleunigungssensorik
- Magnetometer
- Ultraschall-Abstandsmessung zum Boden
- Frontkamera
- Bodenkamera

### 7.2.3 Interaktion mittels ROS

#### Treiber *ardrone\_autonomy*

Für die Ansteuerung der Drohne *ArDrone 2.0* existiert eine Treiber, welche die Initialisierung und die Kommunikation mit der Drohne anbietet. Die *ArDrone 2.0* entspricht hierbei einem Client. Der *ROS*-Core wird auf dem Host-Rechner ausgeführt. Als *ROS*-seitige Schnittstelle werden verschiedene *ROS-Topics* und *ROS-Services* angeboten, welche nachfolgend näher beschrieben werden sollen.

#### Topics

BESCHREIBUNG! *Bild von rqt-graph*

##### *ROS-Topic NavData*

Das *ROS-Topic NavData* fasst alle für die verfügbaren Daten des Quadroopters in einem Nachricht zusammen. Hieraus können geeignete Informationen entnommen werden.

##### *ROS-Topic cmd\_vel*

Nur Geschwindigkeits-Daten. Keine tatsächliche Übergabe von Neigungswinkeln und so ODER?? BESCHREIBUNG!

Entsprechen alle Attribute der `geometry_msgs::Twist` Nachricht dem Wert 0, schaltet die *ArDrone 2.0* in einen Hover-Modus um und versucht, die aktuelle Position zu halten. Zu beachten ist, dass diese Funktion nicht für die Aufgabenstellung herangezogen werden kann, da für den studentischen Laborversuch *Höhenregelung* Daten abweichen des Werts 0 über das *ROS-Topic cmd\_vel* übermittelt werden.

#### *Video-Streams*

Die Übertragung der Video-Daten der beiden Kameras soll an dieser Stelle lediglich erwähnt werden.

#### Services

#### Parameter

### 7.2.4 Inbetriebnahme auf separatem Rechner

Der in Kapitel 7.2.3 *Interaktion mittels ROS* (Seite 25) Treiber kann entsprechend folgender Anleitung installiert werden:

<https://ardrone-autonomy.readthedocs.io/en/latest/installation.html>

Für das System werden folgende *ROS*-Knoten benötigt:

- roslaunch ardrone\\_autonomy ardrone.launch
- rosrun PosControl AutoController  
*alternativ:* rosrun PosControl ManualController
- rosrun keyboard KeyReader

### 7.2.5 Troubleshooting

#### Akkus

Die mit der *ArDrone 2.0* zur Verfügung gestellten Akkus konnten selbst in vollständig geladenen Zustand keine zufriedenstellende Flugdauer garantieren. Hierfür wurden zwei neue Akkus von einem Dritthersteller bezogen. Die Lieferung der bestellten Akkus erfolgte in geringem zeitlichen Abstand zum Projektende.<sup>3</sup>

---

<sup>3</sup> „geringer zeitlicher Abstand“ entspringt in diesem Kontext einem Zeitraum von drei Wochen. In diesem Zeitraum lag die Priorität des Autors auf der schriftlichen Ausarbeitung dieser Projektarbeit.

# 8 Software-Architektur

In diesem Kapitel soll die Architektur des Projekts umrissen werden.

## 8.1 Architektur Konzept

Der Aufbau der Architektur entspricht den Konzept der *Clean Architecture*. Diesem Prinzip folgend zeigen sämtliche Abhängigkeiten des Codes auf *weiter innen liegende* Schichten. Die Farbgebung in Abbildung 16 (Seite 27) orientiert sich an den Inhalten der Vorlesung *Advanced Software-Engineering* der DHBW Karlsruhe<sup>4</sup>.

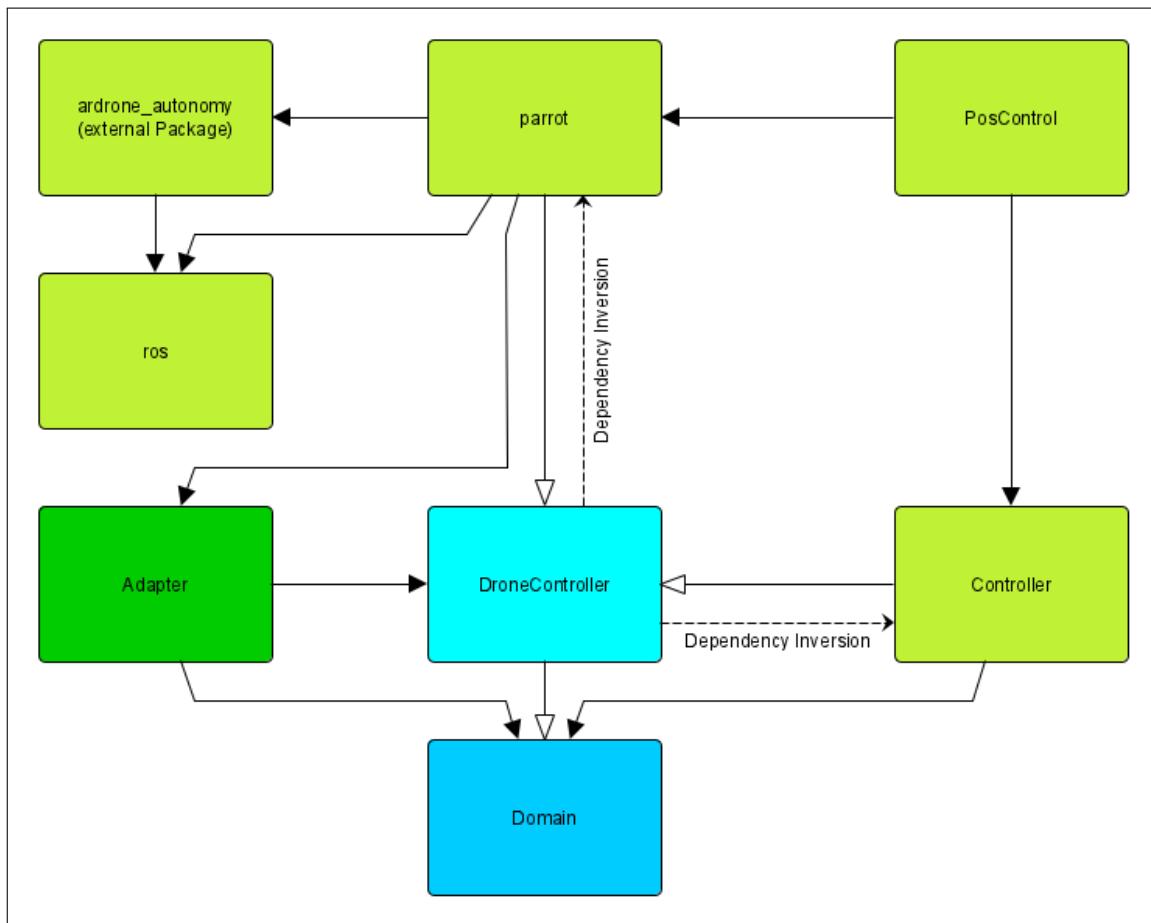


Abbildung 16: ARCHITEKTUR DES REGELUNGSSYSTEMS

Abbildung 16 zeigt das Konzept der Architektur. Die Bedeutungen der Pfeile orientieren sich an UML-Klassendiagrammen. Die Farbgebung unterscheidet die verschiedenen Schichten der *Clean Architecture*: *Domain-Layer* (blau), *Application-Layer* (hellblau), *Adapter-Layer* (grün), *PlugIn-Layer* (hellgrün).

<sup>4</sup>An dieser Stelle sei darauf hingewiesen, dass Abbildung 16 (Seite 27) in der Ausarbeitung für die genannte Vorlesung auftaucht.

## **8.2 Domain Package**

Im *Domain Package* sind grundlegende Klassen abgelegt. Diese werden von den nachfolgend beschriebenen Schichten eingebunden. Gemeinsam mit dem *DroneController Package* bildet es das Herzstück der Anwendung. Eine eindeutige Zuordnung der Klassen in diese beiden *Packages* ist diskutabel.

## **8.3 DroneController Package**

Das Entwurfsmuster des *Application-Layer* entspricht einer *Bridge*. Ziel ist es, sowohl die eingesetzte Drohne, als auch die Implementierung des Reglers mit überschaubarem Aufwand austauschen zu können.

Hierin werden die grundlegenden Aufgaben für die Regelung einer Drohne übernommen. Eine detailliertere Beschreibung findet sich im Kapitel Kapitel 9.2 *Application Layer - DroneController Package* (Seite 30).

## **8.4 Adapter Package**

Im *Adapter Package* wird die Signalverarbeitung der Rohdaten übernommen und anschließend in einem aktuellen Zustand zur Weiteren Verarbeitung bereitgestellt.

## **8.5 Controller Package**

Das *Controller Package* einspricht beinhaltet Regelungsglieder, welche für eine beliebige Regelung im Allgemeinen genutzt werden können. Die Interaktion wird durch die Fassade `PoseController` an das Aufgabenfeld dieser Projektarbeit angepasst.

## **8.6 parrot Package**

Die Interaktion mit der in dieser Projektarbeit eingesetzten Hardware wird vom gleichnamigen *parrot Package* übernommen. Hier sind verschiedene Klassen enthalten, welche mittels *ROS ROS-Topics* mit der Drohne kommunizieren und sowie Messdaten von der Drohne zur Verarbeitung bereitstellen, sowie Steuerungsbefehle an die Drohne übersenden.

## **8.7 PosControl Package**

Das *PosControl Package* kann als Dach-Software angesehen werden. hierin wird die `main()`-Methode aufgerufen und die übergeordnete verwaltende Klasse initialisiert.

# 9 Implementierung

## 9.1 Domain Layer

Der *Domain Layer* stellt als Teil der *Clean Architecture* die Ebene dar, in der allgemein gültige Typen oder Definitionen abgelegt werden können. Nachfolgend werden die Klassen beschrieben, die in diesem Projekt zu dem *Domain Layer* gehören. Sofern nicht anders betitelt, handelt es sich bei den Klassen um Klassen vom Typ *Value-Object*.

### Optional

Die Klasse `Optional` soll dem Ringbuffer ermöglichen, das Nichtvorhandensein von Einträgen darstellen zu können. Diese Klasse ist als *template* implementiert und enthält neben dem Speicherplatz für die generische Instanz einen bool'schen Wert als Validierung. Hiermit wird das Arbeit mit *Null-pointern* im Kontext der Klasse `Ringbuffer` vermieden. *Anmerkung:* Mit der Verwendung von c++17 ist eine vergleichbare Klasse Teil der Standard-Bibliothek. Die Sprachversion des Projekts ist c++11.

### Ringbuffer

Die Klasse `Ringbuffer` soll einen Ringspeicher abbilden. Hier wird nicht -wie der Name vermuten lässt- ein Ringspeicher im Sinne einer ringförmig verketteten Liste implementiert. Diese Klasse Kapselt eine *Standard Template Library (STL)* vom Typ `std::vector<T>`, wobei bei Überschreiten der maximalen Anzahl an Elementen das vordere Elemente entfernt wird.

*Anmerkung:* Die Implementierung der Klasse `std::vector<T>` sieht vor, neue Elemente an das Ende anzuhängen.

### TimedValue

Die Klasse `TimedValue` soll einen mit einem Zeitstempel versehenen Wert abbilden. Dies wird durch das Erben von den Klassen `Timestamp` und `Value` umgesetzt.

### Timestamp

Mit der Klasse `Timestamp` wird ein Zeitstempel eingeführt. Alle *ROS*-Nachrichten beinhalten durch die Kapselung der `std_msgs::Header`-Klasse einen Zeitstempel.

### Unit

Mit `Unit` werden Einheiten umgesetzt, um eine korrekte Übergabe von `Value`-Instanzen zur Laufzeit zu gewährleisten.

### Value

Die Klasse `Value` bildet einen Wert ab. Sie besteht aus einer `Unit` und einem dazugehörigen Zahlenwert.

## Vector3D

Bei der Klasse `Vector3D` handelt es sich um die Abbildung eines Vektors im dreidimensionalen Raum. Zusätzlich wird dem Vektor eine Einheit zugewiesen. Zudem werden in dieser Klasse grundlegende mathematische Operationen für Vektoren implementiert.

BESCHREIBUNG! *Safety-Kram*

## 9.2 Application Layer - *DroneController Package*

`AccelToPos`

`State`

`StateTranslator`

## 9.3 Adapter Layer

`StateBuilder`

Innerhalb der `StateBuilder` werden große Teile der Signalverarbeitung übernommen. Hierzu werden verschiedene Instanzen der `StateHandler` mit entsprechenden Aufgaben implementiert. Als Funktionalität bietet die `StateBuilder` vor Allem das Glätten von Signalen, sowie die Anpassung einer dauerhaften Nullpunkt-Abweichung an.

`StateHandler`

Die `StateHandler` erbt von der `Ringbuffer` für das *Template* der `IMUState`. Zudem werden verschiedene Methoden als Grundlage für die Signalverarbeitung angeboten (`IMUState getAvgState()`, `IMUState getMedianState()` und `IMUState getVariance()`).

## 9.4 PlugIn Layer

### 9.4.1 *parrot Package*

Die Klassen in diesem *Package* entsprechen der vollständigen Implementierung der virtuellen Klassen des *DroneController-Package*. Weitere Details sind dem Code zu entnehmen.

Folgende Klassen sind verfügbar:

- `parrotBattery`
- `parrotControl`
- `parrotIMU`
- `parrotStatus`
- `parrotTransmitter`

#### **9.4.2 Controller Package**

Die in dieser *Package* umfangreiche Vererbungshirarchie wurde umgesetzt, um das parallel hierzu auszuarbeitende *Software Engineering 2*-Projekt ausführlicher umsetzen zu können. Eine schlankeres *Package* wäre möglich.

Nachfolgend sollen die implementierten Klassen dieses *Package* näher erläutert werden:

##### **Controllable**

Die virtuelle `Controllable` hält als Attribut eine Ausprägung der Enumeration `ControllerType` und deklariert zudem die Methoden für die Funktionalität, einen Regelparameter  $k$  verwalten zu können. Diese Funktionalität wird in der `Controller_Basic` definiert.

##### **ControlledOutput**

Mit der `ControlledOutput` werden die Funktionalitäten der Klassen `Controllable` und `Output` vereint.

##### **Controller\_Basic**

Die `Controller_Basic` bildet die Grundlage für die in der Implementierung vorhandenen Regelbausteine, ausgenommen der Klasse `ControllerPID`.

##### **Controller\_Input**

*Lagacy Code BESCHREIBUNG! please delete!*

##### **Controller\_x**

Klassen, welche mit „`Controller_`“ beginnen und einen Baustein (vgl. Kapitel 5.2 *Arten von Reglergliedern* (Seite 10)) benennen, implementieren die Funktion des jeweiligen Bausteins.

##### **ControllerSystem**

Instanzen der `ControllerSystem` kapseln Instanzen verschiedener Regelbausteine (`Controller_`).

Hierbei können diverse Regelbausteine als Reihenschaltung zusammengefasst werden.

*Anmerkung:* In der Implementierung wird diese Klasse nicht aktiv genutzt.

### **ControllerType**

Bei `ControllerType` handelt es sich um eine Enumeration. Hier können Parameter der Regelbausteine über eine kapselnde Instanz der `ControllerSystem` angepasst werden.

*Anmerkung:* Im Projektfortschritt hat sich keine Möglichkeit ergeben, dieses Funktionalität einzusetzen.

### **Inputable und Input**

Die beiden Klassen ermöglichen einen Daten-Eingang für ein Objekt. Hierbei Besitzt die `Input` einen Pointer auf eine Instanz der `Outputable`.

### **Outputable und Output**

Die beiden Klassen bilden den Output eines Objekts an. Hierbei ist die `TimedValue getOutput()` eine virtuelle Methode und muss in einer von der `Outputable` erbenden Klasse definiert werden.

### **TimedDifference**

Die `TimedDifference` wurde eingeführt, um ein Datum und einen Zeitstempel zu ver-einen. Dies ist für die Berechnung der Regelbausteine notwendig.

## **9.4.3 PosControl Package**

### **9.4.4 calling Package**

Dieses *Package* wurde eingeführt, um die Erzeugung neuer *ROS-Messages* zu umgehen. Ferner bildet die Struktur dieser Klassen einen *pull Observer* ab. Die aufgerufene Methode erhält den Pointer auf die aufrufende Klasse und kann hiermit eine Entscheidung über weitere Aktionen treffen.

*Anmerkung:* Dieses *Package* entspricht nicht den Grundsätzen einer *ROS*-Programmierung und ist daher nicht weiter zu nutzen. Die Beschreibung dient an dieser Stelle dem Verständnis des Einsatzes dieser Klassen im *coex Package* und wird darüber hinaus nicht weiter eingesetzt.

### **9.4.5 threading Package**

Das Paket `threading` bietet die Möglichkeit wiederkehrende Aufgaben, abseits von separaten *ROS-Nodes*, bearbeiten zu können. Dieser Zusatz erlaubt ein monolithisches

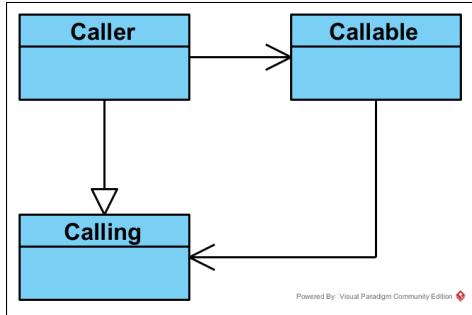


Abbildung 17: KLASSENIDIAGRAMM DES CALLING PACKAGES

Programm zu entwerfen.

### Thread

Diese Klasse bietet die Basis für die Thread-Implementierung, indem die Klasse `std::thread` gekapselt wird. Hier wird mit Aufruf der `start()`-Methode eine neue Instanz auf den zugehörigen *Pointer* initialisiert. Zusätzlich implementiert die Klasse `Thread` einen Sperr-Mechanismus, um synchrones Schreiben zu vermeiden. **BESCHREIBUNG!** *dirty read und so beschreiben?*

### rosThread

Die Klasse `rosThread` erweitert die Klasse `Thread` um eine generische Variable `T Payload`, welcher von den erbenden Klassen zum Versand genutzt werden kann. Die Methode `T runOnce(T Payload)` ist als *full virtual* implementiert, somit wird ein Überschreiben durch erbende Klassen erzwungen. Die Klasse `ros::Rate` ermöglicht eine frequentiell pausierte Abarbeitung des der auszuführenden Methode `T runOnce()`.

*Anmerkung:* Idealerweise sollte eine Instanz der Klasse `ROS::NodeHandler` ebenfalls in dieser Klasse integriert sein. Tests während der Entwicklung zeigten, dass dies nicht umsetzbar ist. Eine Begründung hierfür konnte nicht gefunden werden.

### AutoPublisher

Wie der Name der Klasse `AutoPublisher` erahnen lässt, wird hier ein `ros::Publisher` implementiert. Mit dem Aufruf der `runOnce()`-Methode wird der in Basisklasse `rosThread` gespeicherte `T Payload` mit der Instanz des `ros::Publisher` versandt.

### AutoClient

Die Klasse `AutoClient` kapselt eine Instanz der Klasse `ros::ServiceClient` und bietet somit die Option, Service-Anfragen regelmäßig senden zu können.

## **9.4.6 coex Package**

Dieses Kapitel beschreibt die Klassen, welche zur Interaktion mit der zuerst eingesetzten Hardware genutzt wurden.

*Anmerkung:* Die Implementierung dieses *Packages* wurde vor der konzeptionellen Änderung des *Application-Layers* umgesetzt. Die Beschreibung der Klassen dieses Paketes dient der Einarbeitung nachfolgender Studienarbeiten. Auf Grund der Änderungen im Code kann dieses *Package* nicht Kompiliert werden.

**BESCHREIBUNG!** *Hier evtl den Abhängigkeitsgraph für coex zur Veranschaulichung?*

### **coexBattery**

Wie der Klassename vermuten lässt überwacht diese Klasse die Batteriestand des Quadroopters.

Diese Klasse entspricht der Klasse `parrotBattery` der *ArDrone 2.0*.

### **coexControl**

Die Klasse `coexControl` ist als Fassade eingesetzt und bietet Anwendenden Zugriff auf diverse Funktionen der gekapselten Klassen.

Diese Klasse entspricht der Klasse `parrotControl` der *ArDrone 2.0*.

### **coexMC**

Die Abkürzung *MC* im Klassennamen steht für *Manual Control* und deutet hiermit sowohl das genutzte *ROS-Topic* als auch den zugrundeliegenden Nachrichtentyp an. Laut Entwickler-Literatur (siehe Kapitel 7.1.6 *Mögliche Lösung der Aufgabenstellung* (Seite 22)) werden Steuerungsdaten der vier Freiheitsgrade an den Quadroopter gesendet.

Diese Klasse erbt von der Klasse `coexTransitable`

Diese Klasse entspricht der Klasse `parrotTransmitter` der *ArDrone 2.0*.

### **coexOrientation**

Es war geplant, die Pose des Quadroopters in dieser Klasse berechnen zu lassen. Im weiteren Projektverlauf hat sich ergeben, hierfür eine separate Klasse zu erstellen.

Diese Klasse entspricht der Klasse `parrotIMU` der *ArDrone 2.0*.

### **coexRC**

`coexRC` ist als Fassade für die beiden nachfolgenden implementiert.

#### **coexRC\_Receiver**

Die Klasse `coexRC_Receiver` liest vom *ROS-Topic* `/mavros/rc/in` und somit die Eingaben der Funkfernsteuerung. Eine nähere Erläuterung, welche Daten übermittelt werden, findet sich als Kommentar im Code.

### **coexRC\_Transmitter**

Als *RC-Transmitter* wird die Klasse bezeichnet, welche Nachrichten an das *ROS-Topic* `/mavros/rc/override` veröffentlicht (vgl. Kapitel 7.1.6 *Mögliche Lösung der Aufgabenstellung* (Seite 22)).

Diese Klasse erbt von der Klasse `coexTransitable`

### **coexState**

Die Klasse `coexState` übernimmt die Interaktion mit dem Zustandsautomaten, welcher im *Flight Controller* des Quadroopters realisiert ist. Anfragen zur Änderung eines Zustand beziehen sich maßgeblich auf den Start und die Landung eines Flugs.

Diese Klasse entspricht der Klasse `parrotStatus` der *ArDrone 2.0*.

### **coexTransitable**

Diese Klasse erbt von der Klasse `Transitable`. Hierin wird die Steuerung innerhalb dieses *Packages* als eine normierte *Manual Control*-Nachricht mit dem Wertebereich [-1, 1] definiert.

### **Joystick**

Hier wird ein Hilfskonstrukt für die Transformation von Daten der Funkfernsteuerung eingeführt. Die Klasse implementiert vier Instanzen der Klasse `JoystickAxis`.

### **JoystickAxis**

Hier werden die Rohdaten der Kanäle der Funkfernsteuerung in normierte Daten zu überführt. Zudem können normierte Werte in den Wertebereich der Funkfernsteuerung transformiert werden.

# 10 Ergebnis

Um ein Ergebnis bewerten zu können, wird nachfolgend einer Auswahl aussagekräftiger Daten eines Fluges gezeigt.

Hierbei wurden zwei Starts in dem Datendatzen durchgeführt. Zur besseren Interpretation wurden Daten entfernt, welche entsprechend der Implementierung irrelevant sind.

Ein Mitschnitt, welcher mittels *rosbag* entstand, ist verfügbar. Die genannten Daten wurden mittels geeigneter Software (`rosrun BagPlotter parrot` und `rosbag play <File>.bag`) in menschenlesbare *.txt* Dateien überführt. Diese Daten wurden anschließend zur Ermittlung der von der Implementierung geschätzten Pose genutzt. Dieser Vorgang wurde aus Sicherheitsgründen in einer Simulation (*PosControl/Simulant.cpp*) durchgeführt.

Im Zuge dieses Projekts wurden diverse Sicherheitsfunktionen zum Quadrokopter hinzugefügt. Da sich diese Projektarbeit aus interne Sensoren beschränkt, werden diese als Basis der Sicherheitsfunktionen eingesetzt. Aus den übertragenen Daten kann ein Timeout<sup>5</sup>, sowie ein Zusammenstoß mit einem Objekt ermittelt werden. Als weitere Funktionalität wird ein Absturz auf Grund einer zu geringen Akku-Ladung verhindert. Jedes Sicherheitsfunktion löst das Landemanöver der *ArDrone 2.0* aus.

## 10.1 Analyse realer Flugdaten

In diesem Kapitel werden die Daten eines Testflugs mit der *ArDrone 2.0* analysiert.

### 10.1.1 Verlauf

Dieses Kapitel zeigt die verschiedenen Daten des Flugs, um diese in der nachfolgenden Analyse einordnen zu können.

**BESCHREIBUNG!** *Analyse mit Daten der parrot Drohne*

---

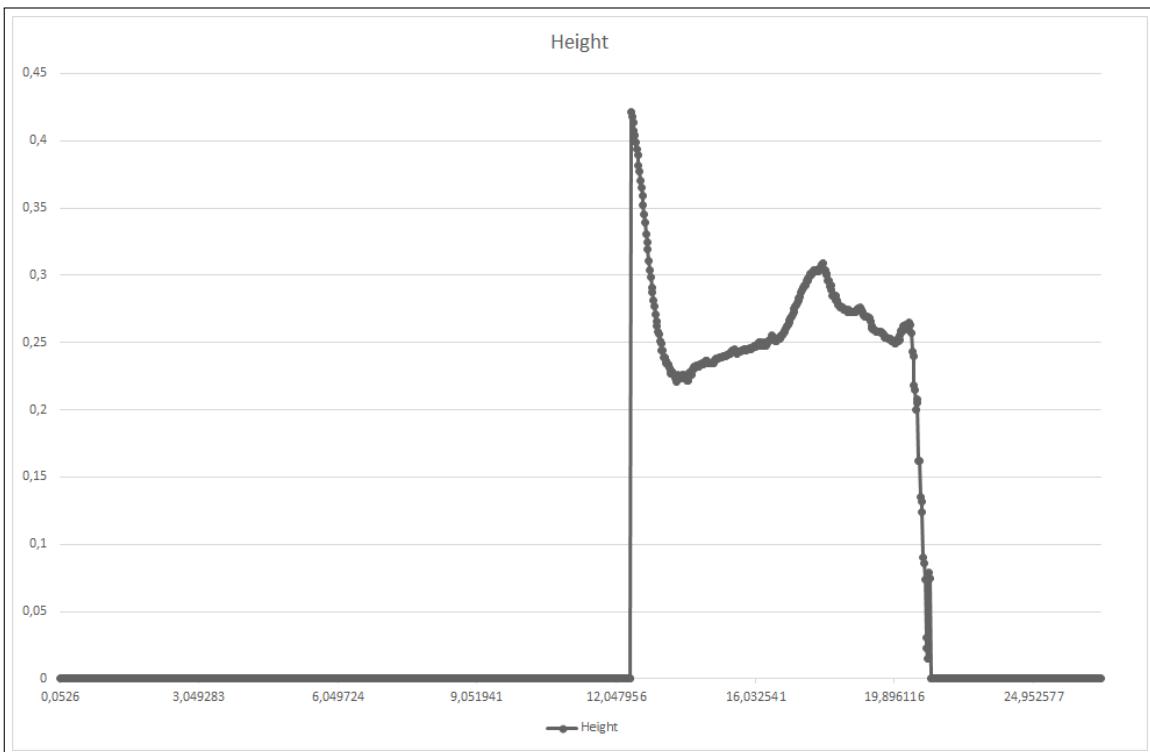
<sup>5</sup>Der *Timeout* wird in diesem Projekt mit dem Eingang einer neuen Nachricht ermittelt. Geeigneter wäre das Starten eines Timers beim Eingang einer Nachricht. Verstreicht die Zeit des Timers soll das Landemanöver initialisiert werden.



**Abbildung 18:** TESTFLUG: STATUSID

#### BESCHREIBUNG! *override!!*

Der Testflug wird mit der Aufforderung der Status-Änderung bei etwa 9.3 s initialisiert. Davor befindet sich der Quadrokopter ruhend auf einer ebenen Oberfläche. Die hier eingenommene StatusID = 7 entspricht ‚Goto Fix Point‘, wobei hier vermutlich eine Höhe von 0.4 m angenommen werden soll. Zwischen etwa 12.3 s und 13.3 s wird keine Nachricht zum aktualisieren der Daten gesendet. Anschließend wechselt der Status zu ‚Flying.. Die Status-Änderung bei etwa 20.6 s entspricht einer Notlandung aufgrund niedriger Batterieladung.



**Abbildung 19:** TESTFLUG: HÖHENVERLAUF

Äquivalent zu Abbildung 18 (Seite 37) zeigt sich der Verlauf des Höhenprofils des Fluges. Die Daten wurden von der Ultraschall-Messung abgegriffen. Eine Transformation der Einheiten in [m] wurde vorgenommen.

## 10.1.2 Analyse

### Lecks der Datenübertragung

**BESCHREIBUNG!** *Beim Start etwa 1s, etwas später nochmal ein bisschen Probleme...*

### Offset der Beschleunigungswerte

Wie Abbildung ?? (Seite ??) entnommen werden kann, sind die Werte für die Beschleunigungen entlang der x- und y-Achsen nicht auf den Wert 0 kalibriert. Eine Kalibrierung des Quadroopters wird unmittelbar vor dem Start gesendet und kann auch unabhängig durch eine User-Eingabe ausgelöst werden.

Die Gravitation ist in Abbildung ?? (Seite ??) ablesbar.

Entsprechend Kapitel 6.2.2 *Signalaufarbeitung* (Seite 17) sind diese Abweichungen zu korrigieren. Als Ansatz kann hier eine Mittlung der Werte bis zum Senden der Start-Aufforderung gewählt werden. Aus Abbildung ?? (Seite ??) und Abbildung ?? (Seite ??) ist ersichtlich, dass hier keine signifikanten Schwankungen auftreten.

### Rekonstruktion der Flughöhe

Bei beiden Filtern ist eine deutliche Abweichung ersichtlich.

**BESCHREIBUNG!** *was ist hier los?* entsprechend Kapitel 6.2.2 *Signalaufarbeitung* (Seite 17) um

Aus der Rekonstruktion in Abbildung ?? (Seite ??) lässt sich zeigen, dass markante Verläufe abgebildet werden können.

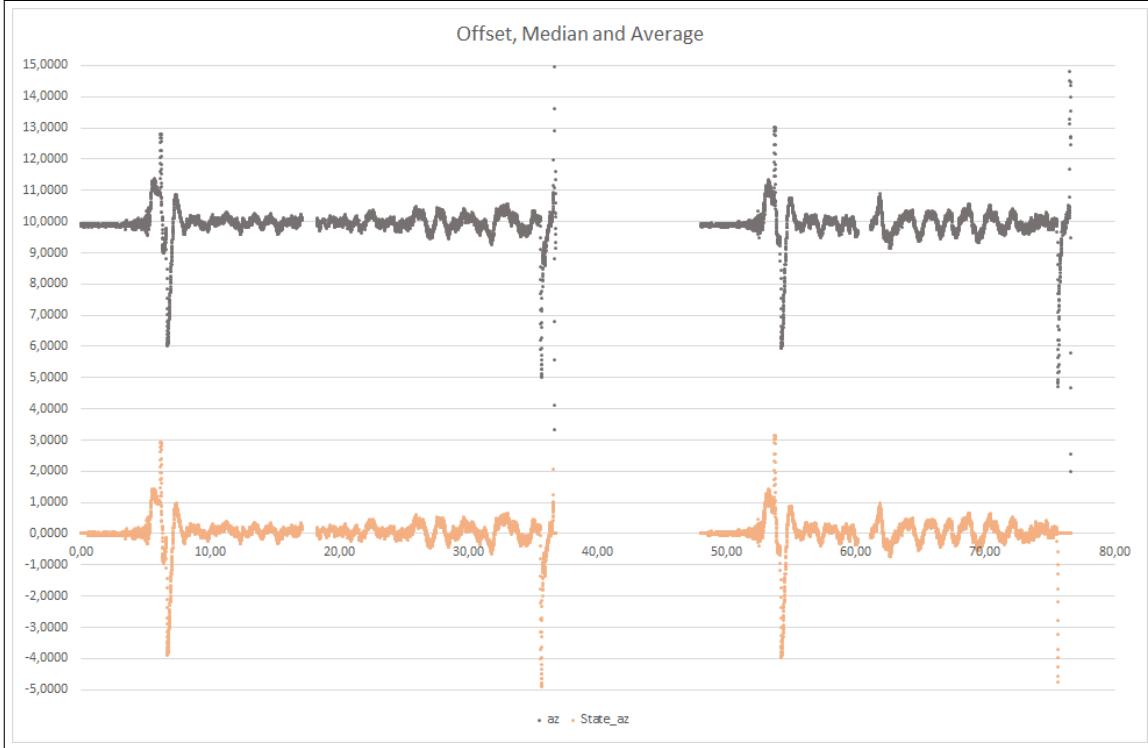
Die rekonstruierte Kurve ist jedoch in den negativen Bereich verschoben. Dies lässt sich mit der fehlenden Datenübertragung beim Start des Quadroopters begründen. Darüber hinaus ist eine konstante Geschwindigkeit nach der Landung anhand der konstanten Steigung der rekonstruierten Kurve in diesem Bereich ersichtlich.

## 10.2 Signalverarbeitung

### 10.2.1 Ermittlung der dauerhaften Nullpunkt-Abweichung

In Abbildung 14 (Seite 19) wird eine Abweichung der Beschleunigungswerte vom Nullpunkt deutlich. Die eingesetzte Software wirkt dem in der `StateBuilder` entgegen, indem über eine definierte Anzahl an Eingangswerten der `IMUState` ein Mittelwert gebildet wird. Um an dieser Stelle aussagekräftigere Ergebnisse zu erhalten, wird zudem die Varianz über den zwischengespeicherten Datensatz gebildet. Der Mittelwert mit

der geringste Varianz aus einer definierten Anzahl an Stichproben wird als Nullpunkt-Abweichung angenommen. Hierbei ist zu beachten, dass die Attribute der [IMUState](#) separat betrachtet werden.



**Abbildung 20:** TESTFLUG SIGNALVERARBEITUNG: AUFARBEITUNG  $a_z$

Es zeigt sich, dass die Nullpunkt-Abweichung aus den Eingangsdaten der Quadrokopter-IMU berechnet und zur Aufarbeitung der Daten eingesetzt werden kann. Als Parameter werden für diese Berechnung jeweils ein Median-Fenster und ein Mittelwert-Fenster der Größe 1 angesetzt. Somit findet keine Veränderung durch diese Filter statt. Es zeigte sich in den Auswertungen, dass ein Median-Fenster bis zur Größe von 15 Werten keinen signifikanten Einfluss auf die berechnete Pose besitzt. Jedoch wirkt sich eine Vergrößerung des Mittelwert-Fensters negativ auf die berechnete Pose aus. Die Abweichung von erwarteten Werten nimmt deutlich zu.

### 10.2.2 Kalibrierung der Posenberechnung

Aus Simulationen mit der in Kapitel 9 *Implementierung* (Seite 29) beschriebenen Software zeigte sich, dass eine einfache Abschätzung der Nullpunkt-Abweichung unach Kapitel 10.2.1 *Ermittlung der dauerhaften Nullpunkt-Abweichung* (Seite 39) nicht vollständig ausreicht. Der Einfluss bereits kleiner Nullpunkt-Abweichung wurde in Kapitel 6.2.2 *Signalaufarbeitung* (Seite 17) erläutert. Aus Erkenntnissen, aufgezeigt in Abbildung 14 (Seite 19), lässt sich nachfolgender mathematische Zusammenhang ableiten:

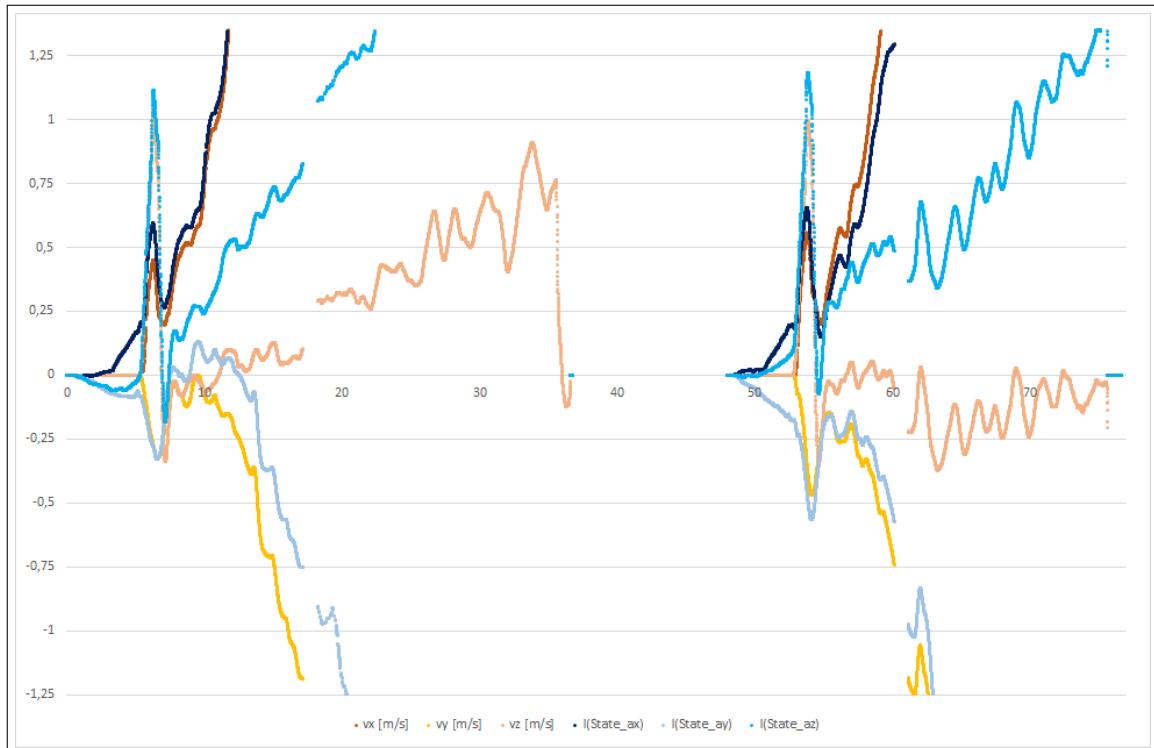
Entsprechend Formel ?? berechnet die [PoseBuilder](#) eine nach der Aufarbeitung durch die [StateBuilder](#) verbleibende Nullpunkt-Abweichung der Beschleunigungswerte. Die-

$$\Delta P_{Direction} = \frac{\Delta a_{Direction}}{2} * \Delta t^2 \quad (6)$$

### ÜBERTRAGUNGSFUNKTION DES I-GLIEDS

se werden zur Berechnung der Pose von den Eingangdaten subtrahiert.

In Abbildung 21 (Seite 41) kann gezeigt werden, dass sich eine zusätzliche Kalibrierung der Beschleunigungsdaten positiv auf die Berechnung der Pose<sup>6</sup> auswirken kann.



**Abbildung 21:** TESTFLUG SIGNALVERARBEITUNG: KALIBRIERUNG

BESCHREIBUNG!

---

<sup>6</sup>In der genannten Abbildung werden die berechneten Geschwindigkeiten aufgetragen.

# 11 Erweiterungen

Dieses Kapitel soll beschreiben, welche weiteren Sensoren eingesetzt werden können, um die Genauigkeit der Pose zu erhöhen und damit die Regelung der Drohne zu stabilisieren.

## 11.1 Dedektion der Umgebung

BESCHREIBUNG! *Auswertung von Objekten und Speicherung der zugehörigen Metadaten*

### 11.1.1 Vorwärts-Berechnung

### 11.1.2 Rückwärts-Berechnung bei Ringschluss

BESCHREIBUNG! *Position und Genauigkeit der vorherig ermittelten Objekte berechnen, wenn ein bekanntes Objekt erneut gesichtet wurde.*

Aufzeichnung des Weges erforderlich

## 11.2 Sensoriken

### 11.2.1 Abstandssensoren

Punktuelle Abstandsmessung

Linienförmige Abstandsmessung

3D Abstandsmessung

### 11.2.2 Magnetometer

### 11.2.3 Kamera

Bodenkamera

Frontkamera

Stereokamera

externe Stereokamera

### 11.2.4 GPS

globale Ausrichtung

## 12 Fazit und Ausblick

Im Zuge der Analyse der Flugdaten (siehe Kapitel 10.1 *Analyse realer Flugdaten* (Seite 36)) in Kombination mit der Rekonstruktion eines idealisierten Flugverlaufs (siehe Kapitel 6.2 *Erzeugung von Posen aus Beschleunigungsdaten* (Seite 14)) zeigte sich, dass sich eine Rekonstruktion der Pose aus Beschleunigungsdaten als schwierig herausstellt.

Um äußerer Einflüssen entgegenwirken zu können, wurden diversere Verfahren der Signalverarbeitung herangezogen (vgl. Kapitel 6.2.2 *Signalaufarbeitung* (Seite 14)).

## Literaturverzeichnis

- [1] Pozo D., Romero L., Rosales J., Quadcopter stabilization by using PID controllers, veröffentlicht 2014
- [2] Fresk E., Nikolakopoulos G., Full Quaternion Based Attitude Control for a Quadrotor, veröffentlicht 19.07.2013
- [3] Heinrich B. (Hrsg.), et al., Kaspers/KOfner Messen - Steuern - Regeln, 8., überarbeitete und ergänzte Auflage. Auflage veröffentlicht 2009 ISBN 978-3-8348-0006-0
- [4] Wendemuth A., Grundlagen der digitalen Signalverarbeitung, veröffentlicht 2005 ISBN 3-540-21885-8
- [5] Tukey J., Exploratory Data Analysis, veröffentlicht 1977 ISBN 0-201-07616-0
- [6] Internationales Elektrotechnisches Wörterbuch – Teil 351: Leittechnik (IEC 60050-351:2006), veröffentlicht 06/2009
- [7] Mey R., Reinhard Mey Textsammlung 14.Auflage, online, <https://www.reinhard-mey.de/texte-fuer-alle/> veröffentlicht 13.12.2017, abgefragt 08.02.2022
- [8] ellwangen2010, Ballon steuern?, online, <https://www.ballonfahrten.com/ballon-steuern/> veröffentlicht 20.02.2010, abgefragt 07.11.2021
- [9] D-Glied, online, [http://testcon.info/FB\\_DE\\_D-Glied.html](http://testcon.info/FB_DE_D-Glied.html) veröffentlicht -unbekannt-, abgefragt 13.04.2022
- [10] Stabilität von Regelkreisen (Frequenzkennlinienverfahren), online, <https://homepages.thm.de/hg13555/Datenbank/aat/index.php/grundlagen-regelungstechnik/45-stabilitaet-von-regelkreisen-frequenzkennlinienverfahren.html> veröffentlicht -unbekannt-, abgefragt 15.04.2022
- [11] ROS - Robot Operating System, online, <https://www.ros.org> veröffentlicht -unbekannt-, abgefragt 28.11.2021
- [12] ROS Indigo Igloo,

online, <http://wiki.ros.org/indigo>  
veröffentlicht -unbekannt-, verändert 08.01.2018, abgefragt 16.03.2022

- [13] How quadcopters work & fly: An intro to multirotors,  
online, <https://www.droneybee.com/how-quadcopters-work/>  
veröffentlicht -unbekannt-, verändert 20.11.2017, abgefragt 01.04.2022
- [14] MAVROS Offboard control example,  
online, [https://docs.px4.io/master/en/ros/mavros\\_offboard.html](https://docs.px4.io/master/en/ros/mavros_offboard.html)  
veröffentlicht -unbekannt-, verändert 02.02.2021, abgefragt 16.03.2022
- [15] AR.Drone Developer Guide,  
Kapitel *AR.Drone 2.0 Overview*, Seite 5 ff. online, <https://jpchanson.github.io/ARdrone/ParrotDevGuide.pdf>  
veröffentlicht 21.05.2012, abgefragt 17.03.2022
- [16] Saks D., Better even at the lowest levels,  
online, <https://www.embedded.com/better-even-at-the-lowest-levels/>  
veröffentlicht 01.11.2008, verändert 05.12.2020, abgefragt 28.07.2021
- [17] Application Note Object-Oriented Programming in C,  
online, [https://www.state-machine.com/doc/AN\\_OOP\\_in\\_C.pdf](https://www.state-machine.com/doc/AN_OOP_in_C.pdf)  
veröffentlicht 06.11.2020, abgefragt 28.07.2021
- [18] Kirk N., How do strings allocate memory in c++?,  
online, <https://stackoverflow.com/questions/18312658/how-do-strings-allocate-memory-in-c>  
veröffentlicht 19.08.2013, abgefragt 17.08.2021
- [19] Bansal A., Containers in C++ STL (Standard Template Library),  
online, <https://www.geeksforgeeks.org/containers-cpp-stl/>  
veröffentlicht 05.03.2018, verändert 12.07.2020, abgefragt 17.08.2021
- [20] Automatic Storage Duration,  
online, <https://www.oreilly.com/library/view/c-primer-plus/9780132781145/ch09lev2sec2.html>  
veröffentlicht -unbekannt-, abgefragt 17.08.2021
- [21] Noar J., Orda A., Petruschka Y., Dynamic storage allocation with known durations,  
online, <https://www.sciencedirect.com/science/article/pii/S0166218X99001754>  
veröffentlicht 30.03.2000, abgefragt 17.08.2021

*Anmerkung:* Wird hier ein Veröffentlichungsdatum als “-unbekannt-“ markiert, so konnte diese Angabe

weder auf der entsprechenden Webseite, noch in deren Quelltext ausfindig gemacht werden.

# Anhang