

RaHM-Lab: Positionsregelung Drohne

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 Drohne“ 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	IV
1 Einleitung	1
2 Problemstellung.....	2
3 Methodik	3
3.1 Eingesetzte Software	3
3.1.1 ROS	3
3.1.2 Ubuntu - Betriebssystem	3
3.1.3 Visual Studio - IDE	3
4 Quadrocopter als System.....	4
4.1 Geometrien	4
4.1.1 +-Anordnung.....	4
4.1.2 x-Anordnung.....	4
4.2 Freiheitsgrade.....	4
5 Eingesetzte Hardware.....	5
5.1 Geometrie	5
5.2 Sensorik.....	5
5.2.1 Gyroskop	5
5.2.2 Laser-Abstandsmessung.....	5
5.2.3 Bodenkamera	5
5.2.4 GPS	5
5.3 Control Stack	5
5.3.1 Flight Controller	5
5.3.2 On Board Computer	5
5.4 Funkfernsteuerung	5
6 Regelsysteme.....	6
6.1 Arten von Reglern	6
6.1.1 P-Regler	6
6.1.2 I-Regler.....	6
6.1.3 D-Regler	6
6.1.4 PID-Regler.....	6
6.1.5 PT-Regler	6

7	Positionsregelung von Quadroköptern	7
7.1	Positionsregelung von Quadroköptern aus Beschleunigungsdaten	7
8	Aufbau des Bausatzes	8
8.1	Aufbau	8
8.2	Konfiguration des Flight Controllers	8
8.3	Troubleshooting.....	8
8.3.1	Platinenfehler	8
8.3.2	Bus-System des RC Empfängers.....	8
9	Analyse der verfügbaren ROS-Topics	9
9.1	ROS-Messages	9
9.2	mavros als Abstraktionsebene	9
9.3	ROS-Topics.....	9
9.3.1	/mavros to /simple_offboard	9
9.3.2	/simple_offboard to /mavros	9
9.3.3	unklar	9
9.3.4	externe Steuerung	10
9.3.5	Informative Topics	10
9.4	Troubleshooting.....	10
9.4.1	Clover ist faul.....	11
9.4.2	md5-Sum des Topics OverrideRCIn	11
9.4.3	Topics OverrideRCIn läuft nur mit Topic RCIn	12
9.4.4	Topics setpoint_altitude/thrust läuft nicht	12
10	Software-Architektur.....	13
10.1	Abstraction Layer	14
10.2	Domain Layer.....	15
10.3	Application Layer	15
10.4	Adapter Layer	16
10.5	Plugin Layer	16
10.5.1	Plugin Layer - <i>calling</i> Package	16
10.5.2	Plugin Layer - <i>threading</i> Package.....	16
10.5.3	Plugin Layer - <i>coex</i> Package.....	17
11	Implementierung.....	18
12	Genutzte ROS-Topics.....	19
13	Fazit und Ausblick.....	20
	Literaturverzeichnis	
	Anhang	

Abbildungsverzeichnis

1	Architektur des Positionsregelungssystems	13
---	---	----

Abkürzungsverzeichnis

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

Tabellenverzeichnis

Code-Verzeichnis

1	Freigegebene Topics	11
2	Befehl zum Öffnen des <i>OverrideRCIn</i> -Headers	12
3	Definition des Struct <code>MD5Sum</code> für das Template <i>OverrideRCIn</i>	12

1 Einleitung

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

aus dem Lied *Über den Wolken* von Reinhard Mey, 1973 [3]

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. [4]). Während der Versuchsdurchführung von Studierenden der DHBW Karlsruhe an einem Quadrokofter 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*¹ soll für die Studierenden dahingehend vereinfacht werden, alsdass der eingesetzte Quadrokofter 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 Quadrokofters erweitert sich die Aufgabenstellung um den Aufbau des Bausatzes und die Inbetriebnahme des Quadrokofters, 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.

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

3 Methodik

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

3.1 Eingesetzte Software

3.1.1 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. [5])

Die *ROS* Versionen werden jeweils mit Namen versehen, wobei die Anfangsbuchstaben der Versionen der alphabetischen Nummerierung entsprechen. In diesem Projekt wurde die aktuellste *ROS*-Version *noetic* eingesetzt.

Die *ROS*-*Nodes* wurden mittels *catkin* compiliert.

3.1.2 Ubuntu - Betriebssystem

3.1.3 Visual Studio - IDE

4 Quadrokopter als System

4.1 Geometrien

4.1.1 +-Anordnung

4.1.2 x-Anordnung

4.2 Freiheitsgrade

6 Freiheitsgrade, tatsächlich 4 individuell regelbar.

5 Eingesetzte Hardware

Bei dem für die DHBW Karlsruhe neu angeschafften Quadrocopter 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 Quadrocoptern entwickelt. Das Modell besitzt einen Rahmen, welche die Rotoren bei Kollisionen schützen soll.

5.1 Geometrie

Anordnung der Rotoren

5.2 Sensorik

5.2.1 Gyroskop

5.2.2 Laser-Abstandsmessung

5.2.3 Bodenkamera

5.2.4 GPS

5.3 Control Stack

BESCHREIBUNG! *Bezeichnung auf deutsch?*

5.3.1 Flight Controller

PX4 Racer

5.3.2 On Board Computer

Raspberry Pie 4

5.4 Funkfernsteuerung

s-Bus ??

6 Regelsysteme

Evtl Norm „IEC 60050-351 Internationales Elektrotechnisches Wörterbuch – Teil 351: Leittechnik“ zitieren, auf jeden Fall als Grundlage heranziehen. + Vorlesung Regelungstechnik von Studium MB13VT

6.1 Arten von Reglern

6.1.1 P-Regler

6.1.2 I-Regler

6.1.3 D-Regler

6.1.4 PID-Regler

6.1.5 PT-Regler

7 Positionsregelung von Quadrokoptern

7.1 Positionsregelung von Quadrokoptern aus Beschleunigungsdaten

8 Aufbau des Bausatzes

8.1 Aufbau

8.2 Konfiguration des Flight Controllers

8.3 Troubleshooting

8.3.1 Platinenfehler

8.3.2 Bus-System des RC Empfängers

9 Analyse der verfügbaren ROS-Topics

9.1 ROS-Messages

9.2 mavros als Abstraktionsebene

mavros ist eine Bibliothek, welche allgemein für Fluggeräte mit 4 Freiheitsgraden erstellt wurde. Zusätzlich können verschiedene Sensordaten übermittelt werden.

9.3 ROS-Topics

In diesem Kapitel sollen die möglicherweise für diese Studienarbeit tauglichen *ROS-Topics* beschrieben werden.

9.3.1 /mavros to /simple_offboard

/mavros/state

Übergibt den Zustand der *State Machine* des *Flight Controller*.

/mavros/battery

Übergibt die Spannung und den Ladezustand des Akkus. Da der Akku ausschließlich mit dessen Spannungsversorgung an die Drohne angeschlossen wird, sind keine weiteren Daten verfügbar. **BESCHREIBUNG!** *evtl noch Stromstärke?*

/mavros/global_position/global

Keine Informationen im Ruhezustand

/mavros/local_position/pose

Liest Rangefinder aus (z-Position wird angeasst), ändert aber sonst keine Position.
Typ: position: orientation:

/mavros/local_position/velocity_body

/mavros/manual_position/control

9.3.2 /simple_offboard to /mavros

9.3.3 unklar

/mavros/global_position/local

Keine Informationen im Ruhezustand

`/mavros/global_position/home`

Keine Informationen im Ruhezustand Evtl, weil hier nichts definiert wurde?

`/mavros/local_position/odom`

9.3.4 externe Steuerung

`/mavros/rc/in`

Signal der Fernsteuerung.

Typ: `mavros::RCIn`

`channel[0] =`

`channel[1] =`

`channel[2] =`

`channel[3] =`

`/mavros/rc/override`

Überschreibt das Signal der Fernsteuerung. Allerdings nimmt mavros vermutlich weiterhin das original-Signal `/mavros/rc/in` auf. Somit kommt es zu Störungen der gewünschten Motordrehzahlen. Typ: `mavros::OverrideRCIn` Channel-Belegung wie *ROS-Topic* `/mavros/rc/in`.

9.3.5 Informative Topics

`/mavros/imu/data`

Wird informativ vom Mavros-Knoten gesendet Typ: `orientation`: Winkel um die Achsen `[r]` `angular_velocity`: Geschwindigkeit entlang der Achsen `[m/s]` `linear_acceleration`: Beschleunigung entlang der Achsen `[m/s2]`

`/rangefinder/data`

`/mavlink/from`

`/mavros/rc/out`

Gibt die PWM-Modulation der Motoren an - oder die Drehzahl. **BESCHREIBUNG!**
Nochmal in der Doku nachschauen. Typ: `mavros::RCOut`

9.4 Troubleshooting

BESCHREIBUNG! *Topics anders benannt, als in Dokumentation (überall steht `/mavros` davor*

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

9.4.1 Clover ist faul

Um Ressourcen zu sparen werden einige *ROS-Topics* nicht vom *Raspberry Pie 4* verarbeitet beziehungsweise weitergegeben.

Lösung

Unterdrückte *ROS-Topics* müssen **BESCHREIBUNG!** *untested!*

<https://clover.coex.tech/en/mavros.html>

Freigegebene *ROS-Topics* in `mavros.launch`

```
<rosparam param="plugin_whitelist">
- altitude
- command
- distance_sensor
- ftp
- global_position
- imu
- local_position
- manual_control
# - mocap_pose_estimate
- param
- px4flow
- rc_io
- setpoint_attitude
- setpoint_position
- setpoint_raw
- setpoint_velocity
- sys_status
- sys_time
- vision_pose_estimate
# - vision_speed_estimate
# - waypoint
</rosparam>
```

Code 1: FREIGEGBENE *ROS-TOPICS* IN `MAVROS.LAUNCH`

9.4.2 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/73b27a463a40a3eda1f9fbb1fc86d6f3], 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 2: BEFEHL ZUM ÖFFNEN DES *OverrideRCIn*-HEADERS

```
template<class ContainerAllocator>
struct MD5Sum<::mavros_msgs::OverrideRCIn_<ContainerAllocator>>
{
    static const char* value()
    {
        return "73b27a463a40a3eda1f9fbb1fc86d6f3";
    }

    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 3: DEFINITION DES STRUCT *MD5Sum* FÜR DAS TEMPLATE *OverrideRCIn*

von dem *Raspberry Pie 4* kopieren und in der Definition des lokalen *ROS OverrideRCIn*-Headers ersetzen.

9.4.3 Topics *OverrideRCIn* läuft nur mit Topic *RCIn*

Lösung

9.4.4 Topics *setpoint_altitude/thrust* läuft nicht

Lösung

10 Software-Architektur

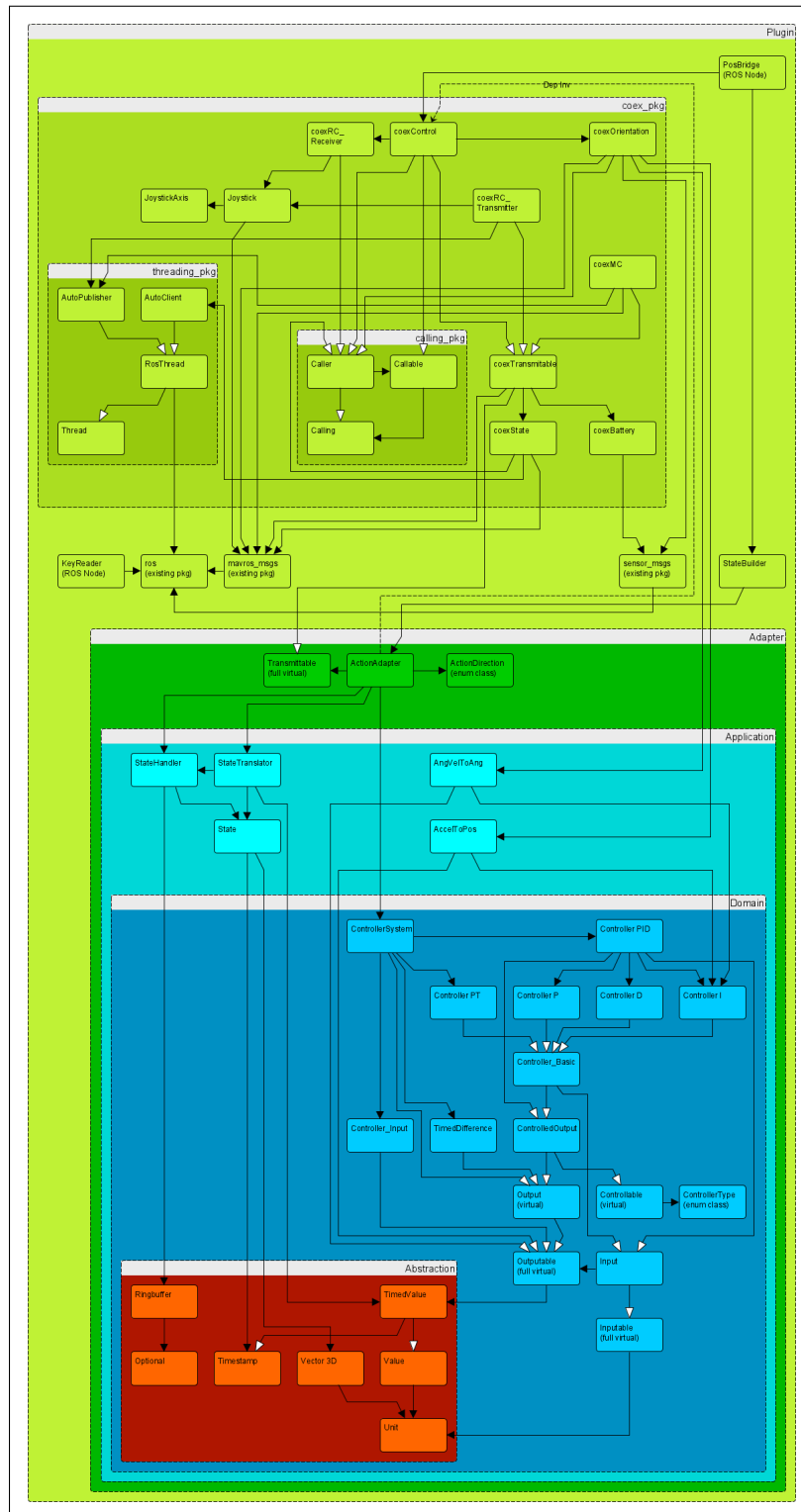


Abbildung 1: ARCHITEKTUR DES POSITIONSREGELUNGSSYSTEMS

Abbildung 1 zeigt die Architektur der Implementierung. Hierbei wurde die *Clean Architecture* zu Grunde gelegt.

10.1 Abstraction Layer

Der *Abstraction 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 *Abstraction 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 die Arbeit mit *Null-pointern* im Kontext der Klasse `Ringbuffer` vermieden.

Ringbuffer

Die Klasse `Ringbuffer` soll einen Ringspeicher abbilden. Hier wird nicht -wie man 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 Element entfernt wird.

Anmerkung: Die Implementierung der Klasse `std::vector<T>` sieht vor, neue Elemente an das Ende anzuhängen. Bei der Klasse `Ringbuffer` handelt es sich nicht um ein *Value-Object*, da **BESCHREIBUNG! bitte begründen ;).**

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.

10.2 Domain Layer

`Controllable`

`ControlledOutput`

`Controller_Basic`

`Controller_D`

`Controller_I`

`Controller_Input`

`Controller_P`

`Controller_PID`

`Controller_PT`

`ControllerSystem`

`ControllerType`

`Input`

`Inputable`

`Output`

`Outputable`

`TimedDifference`

10.3 Application Layer

`AccelToPos`

`State`

StateHandler

StateTranslator

10.4 Adapter Layer

ActionAdapter

ActionDirection

Transmittable

10.5 Plugin Layer

10.5.1 Plugin Layer - *calling* Package

Calling

Callable

Caller

10.5.2 Plugin Layer - *threading* Package

Das Paket *threading* bietet die Möglichkeit wiederkehrende Aufgaben, abseits von separaten *ROS-Nodes*, bearbeiten zu können. Dieser Zusatz erlaubt ein monolithisches 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. Für dieses Projekt ist dies notwendig, da die Anfrage für den *FlightMode* „OFFBOARD“ mit einer Frequenz von mindestens $2Hz$ als *Alive*-Nachricht genutzt wird. Wird die Zeit von $500ms$ ohne Anfrage überschritten, fällt der *Flight Controller* auf den zuvor verwendeten *FlightMode* zurück.

10.5.3 Plugin Layer - coex Package

`coexBattery`

`coexControl`

`coexMC`

`coexOrientation`

`coexRC`

`coexRC__Receiver`

`coexRC__Transmitter`

`coexState`

`coexTransmittable`

`Joystick`

`JoystickAxis`

11 Implementierung

12 Genutzte ROS-Topics

13 Fazit und Ausblick

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] Mey R., Reinhard Mey Textsammlung 14.Auflage, online, <https://www.reinhard-mey.de/texte-fuer-alle/> veröffentlicht 13.12.2017, abgefragt 08.02.2022
- [4] ellwangen2010, Ballon steuern?, online, <https://www.ballonfahrten.com/ballon-steuern/> veröffentlicht 20.02.2010, abgefragt 07.11.2021
- [5] ROS - Robot Operating System, online, <https://www.ros.org> veröffentlicht -unbekannt-, abgefragt 28.11.2021
- [6] 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
- [7] Application Note Object-Oriented Programming in C, online, https://www.state-machine.com/doc/AN_OOP_in_C.pdf veröffentlicht 06.11.2020, abgefragt 28.07.2021
- [8] 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
- [9] 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
- [10] Automatic Storage Duration, online, <https://www.oreilly.com/library/view/c-primer-plus/9780132781145/ch09lev2sec2.html> veröffentlicht -unbekannt-, abgefragt 17.08.2021
- [11] Noar J., Orda A., Petruschka Y., Dynamic storage allocation with known durati-

ons,

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