



---

**Entwicklung der Atmosphärensteuerung eines Gewächshauses für  
zukünftige Mondmissionen**

---

**BACHELORARBEIT**

für die Prüfung zum

**BACHELOR OF ENGINEERING**

des Studiengangs Informationstechnik

der Dualen Hochschule Baden-Württemberg Mannheim

von

**Adrian Roeser**

Abgabe am 27. August 2024

---

Bearbeitungszeitraum:	13.05.24 – 27.08.24
Matrikelnummer, Kurs:	5451585, TINF21IT1
Abteilung:	Avioniksysteme
Ausbildungsbetrieb:	Deutsches Zentrum für Luft- und Raumfahrt e.V.
Betreuer des Ausbildungsbetriebs:	Norbert Toth
Gutachter der Dualen Hochschule:	Joachim Wagner

# Erklärung

Ich versichere hiermit, dass ich meine Bachelorarbeit mit dem

THEMA

**Entwicklung der Atmosphärensteuerung eines Gewächshauses für zukünftige Mondmissionen**

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.\*

\* falls beide Fassungen gefordert sind

---

Bremen, den 27. August 2024

## Kurzfassung

Ein großes Ziel der Raumfahrt ist es, andere Himmelskörper zu besiedeln. Zum Fußfassen ist jedoch eine stetige Nahrungsversorgung notwendig. Um Kosten zu sparen, macht es Sinn, Kulturpflanzen vor Ort zu züchten. Der EDEN LUNA Container dient dazu, moderne, automatische Pflanzenzucht-Technologien zur Lebensmittelproduktion zu testen, um in einer zukünftigen Iteration ein raumfahrttaugliches Produkt erstellen zu können .

Diese Arbeit befasst sich mit der Entwicklung und dem Test der Software des **Atmosphere Management System** von EDEN LUNA. Diese umfasst Regelungen, Gerätetreiber und Telekommunikationselemente. Weiterhin wurden Überlegungen zur benötigten Steuerhardware angestellt.

A major goal of space travel is to colonize other celestial bodies. For a permanent stay, however, a constant supply of food is necessary. To reduce costs, it makes sense to grow crops on site. The EDEN LUNA container is used to test modern, automatic farming technologies for food production in order to be able to create a product suitable for space travel in a future iteration .

In this thesis, the software of the included **Atmosphere Management System** is developed and tested. This includes control loops, device drivers and telecommunication elements. Furthermore, considerations were made regarding the required hardware for controlling.

# Inhaltsverzeichnis

## Abbildungsverzeichnis

## Tabellenverzeichnis

## Abkürzungsverzeichnis

### 1. Einleitung

### 2. Aufgabenstellung und geplantes Vorgehen 2

### 3. Vergleich zu bestehenden Technologien 3

3.1. EDEN LAB . . . . . 3

3.2. EDEN ISS . . . . . 5

3.3. Auf der ISS eingesetzte Technologien . . . . . 6

3.4. Weitere Ansätze aus der Wissenschaft . . . . . 8

### 4. Umfeld, Rahmen und Voraussetzungen 10

4.1. Einordnung in EDEN LUNA . . . . . 10

4.2. Software Voraussetzungen . . . . . 12

4.3. Softwarebibliothek OUTPOST . . . . . 13

4.4. Hardware Voraussetzungen . . . . . 15

4.4.1. Spectra PowerBox . . . . . 15

4.4.2. Remote IO Geräte von Brainboxes . . . . . 15

4.5. Verwendete Begriffe . . . . . 16

4.5.1. Telekommando . . . . . 17

4.5.2. Telemetrie . . . . . 17

### 5. Strukturierte Aufgabenstellung 18

### 6. Anforderungen 20

6.1. EDEN LUNA Allgemein . . . . . 20

6.2. Atmosphäre Management System . . . . . 21

6.2.1. Verbesserungen aus vergangenen Iterationen . . . . . 21

6.2.2.	Umweltparameter und Regelkreise . . . . .	22
6.2.3.	Betriebsmodi der Regelkreise . . . . .	24
6.2.4.	Telekommunikation . . . . .	26
6.2.5.	Schnittstellen . . . . .	27
6.2.6.	Gewählte Reihenfolge . . . . .	28
<b>7.</b>	<b>Hardwarearchitektur</b>	<b>30</b>
7.1.	Aufbau . . . . .	30
7.2.	Auswahl der Brainboxes . . . . .	31
7.3.	Hardware der einzelnen Regelkreise . . . . .	33
7.3.1.	Temperaturregelung . . . . .	33
7.3.2.	Luftfeuchtigkeitsregelung . . . . .	33
7.3.3.	Luftgeschwindigkeitsregelung . . . . .	34
7.3.4.	CO2-Regelung . . . . .	34
7.3.5.	Sauerstoffregelung . . . . .	35
7.3.6.	Filterüberwachung . . . . .	35
7.3.7.	Überwachung sonstiger Aktoren . . . . .	35
7.4.	Weitere Komponenten . . . . .	36
7.4.1.	Weitere Sensoren . . . . .	36
7.4.2.	Motorsteuerung . . . . .	37
<b>8.</b>	<b>Softwarearchitektur</b>	<b>38</b>
8.1.	Implementierung der Komponenten . . . . .	39
8.1.1.	Hardwaretreiber . . . . .	39
8.1.2.	Value Store . . . . .	40
8.1.3.	Regelungen . . . . .	40
8.1.4.	Überwachung sonstiger Aktoren . . . . .	53
8.1.5.	Telekommunikation . . . . .	53
8.2.	Fehlerverhalten . . . . .	56
<b>9.</b>	<b>Testen</b>	<b>57</b>
9.1.	Unittests . . . . .	57
9.1.1.	Mockups . . . . .	57
9.1.2.	Temperaturregelung und Luftfeuchtigkeitsregelung . . . . .	58
9.1.3.	Luftgeschwindigkeitsregelung . . . . .	58
9.1.4.	CO2-Regelung . . . . .	59
9.1.5.	Sauerstoffregelung und Filterüberwachung . . . . .	61
9.1.6.	Überwachung sonstiger Aktoren . . . . .	63
9.1.7.	Telekommandos . . . . .	63
9.1.8.	Coverage . . . . .	64

9.2. Hardwarenahe Tests . . . . .	65
9.2.1. Tests der Treiber . . . . .	65
9.2.2. Systemtest . . . . .	69
<b>10. Bewertung der Aufgaben</b>	<b>71</b>
<b>11. Aufgetretene Probleme</b>	<b>74</b>
<b>12. Fazit</b>	<b>76</b>
12.1. Ergebnis . . . . .	76
12.2. Ausblick . . . . .	77
<b>Literatur</b>	<b>79</b>

# Abbildungsverzeichnis

3.1. EDEN LAB . . . . .	4
3.2. EDEN LAB Experimentierbereich . . . . .	4
3.3. EDEN ISS von außen[3] . . . . .	5
3.4. Veggie[8] . . . . .	7
3.5. Advanced Plant Habitat[10] . . . . .	8
4.1. Übersicht Subsysteme . . . . .	11
4.2. Brainboxes ED-538[18] . . . . .	16
6.1. Betriebsmodi und deren Übergänge . . . . .	26
7.1. Übersicht der AMS-Hardwarekomponenten . . . . .	30
7.2. Hardwaretabelle Ausschnitt . . . . .	31
7.3. Geräte - Brainboxes Zuordnung Ausschnitt . . . . .	32
7.4. Lüftersteuerung über Motorcontroller . . . . .	37
8.1. Übersicht Software und Hardware . . . . .	38
8.2. Ablaufdiagramm Temperaturregelung . . . . .	43
8.3. Ablaufdiagramm Luftfeuchtigkeitsregelung . . . . .	44
8.4. Ablaufdiagramm Luftgeschwindigkeitsregelung . . . . .	46
8.5. Ablaufdiagramm CO2-Regelung . . . . .	49
8.6. UML-Klassendiagramm CO2-Regelung . . . . .	50
8.7. Ablaufdiagramm Sauerstoffregelung . . . . .	51
8.8. Ablaufdiagramm Filterüberwachung . . . . .	52
9.1. Konsolen Ausgabe der Tests der Temperaturregelung . . . . .	58
9.2. Konsolen Ausgabe der Tests der Luftfeuchtigkeitsregelung . . . . .	58
9.3. Konsolen Ausgabe der Tests der Luftgeschwindigkeitsregelung . . . . .	59
9.4. Ausschnitt der Konsolen Ausgabe der Tests der CO2-Phasenbestimmung . . . . .	60
9.5. Konsolen Ausgabe der Tests der CO2-Regelung . . . . .	61
9.6. Konsolen Ausgabe der Tests der Filterüberwachung . . . . .	62
9.7. Konsolen Ausgabe der Tests der Sauerstoffregelung . . . . .	62
9.8. Test der Überwachung sonstiger Aktoren . . . . .	63
9.9. Ausschnitt der Konsolen Ausgabe Telekommando-Tests . . . . .	64

9.10. Coverage Report der Regelungen . . . . .	64
9.11. Coverage Report der Telekommandotests . . . . .	65
9.12. Schematischer Aufbau des Hardwaretests mit Motorsteuerung und Lüfter . . . . .	66
9.13. Physikalischer Aufbau des Hardwaretests mit Motorsteuerung und Lüfter . . . . .	66
9.14. Schematischer Aufbau des Hardwaretests mit Brainboxes ED-538 und Lüfter . . . . .	68
9.15. Physikalischer Aufbau des Hardwaretests mit Brainboxes ED-538 und Lüfter . . . . .	69



# Tabellenverzeichnis

10.1. Verifizierungsmatrix der Aufgaben . . . . .	73
---	----

# Abkürzungsverzeichnis

## **AMS**

Atmosphere Management System 3, 2, 5, 10, 18, 20–29, 38, 40, 53 f., 67, 69, 74, 76

**API** Application Programming Interface . . . . . 53

## **CAN**

Controller Area Network . . . . . 12

## **CCSDS**

Consultative Committee for Space Data Systems . . . . . 12, 26, 53

**CEL** Command Exchange Link . . . . . 10, 69

## **DHCS**

Data Handling & Control System . . . . . 10, 24, 27 f., 53 f., 56

## **DLR**

Deutsches Zentrum für Luft und Raumfahrt . . . . . 1, 20, 53, 69

## **ESA**

European Space Agency . . . . . 20

**ICS** Illumination Control System . . . . . 10

**IO** Input/ Output . . . . . 15, 30 f., 36, 69

**ISS** International Space Station . . . . . 3, 6 f.

## **NDS**

Nutrient Delivery System . . . . . 10, 27

## **OUTPOST**

Open modular software Platform for Spacecraft . . . 12 ff., 39, 53 f., 74

## **PANDO**

Packet Network Documentation model . . . . . 53 f.

<b>ppm</b>	parts per million . . . . .	34, 47
<b>PUS</b>	Packet Utilization Standard . . . . .	12, 26, 53
<b>TCS</b>	Thermal Control System . . . . .	10, 22 f., 27, 33, 41
<b>UART</b>	Universal Asynchronous Receiver/ Transmitter	12, 15, 30, 36, 39, 65, 71, 73 f., 77
<b>UDP</b>	User Datagram Protocoll . . . . .	53
<b>UML</b>	Unified Modeling Language . . . . .	50
<b>USB</b>	Universal Serial Bus . . . . .	65
<b>VOC</b>	Volatile Organic Compounds (flüchtige organische Verbindungen) . .	21, 33
<b>XML</b>	EXtensible Markup Language . . . . .	53 f.

# 1. Einleitung

Die Versorgung von Menschen, insbesondere ForscherInnen, die in extrem lebensfeindlichen Umgebungen, wie der Antarktis oder dem Weltraum, arbeiten müssen, stellt ein großes logistisches Problem dar. Vor allem bei verderblichen Lebensmitteln wie etwa Gemüse, die aufgrund des hohen Wasseranteils dazu recht schwer sind, fallen hohe Kosten für den Transport an. Gleichzeitig sind sie für das Wohlbefinden und eine gesunde Ernährung der ForscherInnen unerlässlich.

Lösungen finden sich im Bereich der Controlled Environment Agriculture, also der Landwirtschaft unter vollständig kontrollierten Bedingungen, welche das Anbauen von Pflanzen an lebensfeindlichen Orten ermöglicht. Wenn diese möglichst ferngesteuert oder gar autonom betrieben werden soll, sind fortschrittliche Softwarekonzepte nötig, um eine genaue Regulierung der Umweltparameter zu gewährleisten und gleichzeitig eine gute Überwachung sicherstellen zu können.

Das **D**eutsches Zentrum für **L**uft und **R**aumfahrt (DLR) untersucht entsprechende Konzepte und Technologien im Rahmen der EDEN Projekte.

Das Projekt EDEN ISS wurde bereits abgeschlossen. Mit den gewonnenen wissenschaftlichen Erkenntnissen werden nun Verbesserungen des Systems vorgenommen. Das mobile Gewächshaus wird komplett erneuert und unter dem Namen EDEN LUNA weiterentwickelt. Genau hierum soll es in dieser Arbeit gehen. Nach Abschluss des Projekts soll der Container in einer Mondsimulation getestet werden.[1]

## 2. Aufgabenstellung und geplantes Vorgehen

Gegenstand der Arbeit ist die Entwicklung der Software für das **A**tmosphere **M**anagement **S**ystem (AMS) von EDEN LUNA, einer Analogmission<sup>1</sup> für zukünftige Mondmissionen, bestehend aus einem automatisierten Gewächshaus und einem Arbeitsbereich für Menschen.

Bestandteil der Arbeit sind die Auswahl der Steuerhardware für Sensoren und Aktoren, die Implementierung von Hardwaretreibern, Regelkreisen und Telekommunikationsstandards und das Testen des Systems.

Das Design des AMS wird von den verantwortlichen SystemdesignerInnen entwickelt. Aus den Anforderungen soll im Rahmen dieser Arbeit funktionale Software entstehen. Die Architektur der Software wird vom fachlichen Betreuer vorgegeben. Darüber hinaus stellt dieser sicher, dass das entwickelte Subsystem kompatibel mit dem restlichen System ist.

---

<sup>1</sup>Mission unter Simulation von Raumfahrtbedingungen zur Erprobung von Technologien und Experimenten

## 3. Vergleich zu bestehenden Technologien

In diesem Kapitel werden Gemeinsamkeiten und Unterschiede zum festverbaudtem Schwesterprojekt EDEN LAB, dem Vorgängerprojekt EDEN ISS, auf der ISS eingesetzten und weiteren Technologien aufgezeigt.

### 3.1. EDEN LAB

Das erste EDEN Projekt wurde 2012 ins Leben gerufen. Das EDEN LAB ist ein festverbautes System bestehend aus einem Arbeitsbereich, einem Analysebereich, einem Versorgungsbereich und dem Experimentierbereich. Letzteres ist das Herzstück, in dem die Pflanzen wachsen. Es begleitet die Entwicklung der übrigen EDEN Projekte und dient der Erprobung und Validierung von Technologien und Verfahren des Pflanzenanbaus. Das EDEN LAB ist seit 2012 stets im Wandel und wird regelmäßig mit neuer Technik, wie beispielsweise zusätzlichen Sensoren, verbessert.[2]

In Abb. 3.1 ist der Arbeitsbereich und der Experimentierbereich zu sehen. In Abb. 3.2 wird ein Ausschnitt des Experimentierbereiches gezeigt.

### 3.1. EDEN LAB

---



Abbildung 3.1.: EDEN LAB



Abbildung 3.2.: EDEN LAB Experimentierbereich

## 3.2. EDEN ISS

Beim Vorgängerprojekt EDEN ISS handelt es sich um ein automatisiertes Gewächshaus mit einem Arbeitsbereich. Es wurde 2017 gebaut und getestet, bevor es 2018 zu seinem Einsatz in die Antarktis gebracht wurde. Der Container stellte über Jahre hinweg seine Funktionalität unter Beweis. So konnten mit dessen Hilfe Verbesserungsmöglichkeiten für die nächste Iteration gesammelt werden. EDEN ISS wurde 2023 zurück nach Bremen gebracht. Das Äußere des Containers ist in Abb. 3.3 dargestellt.



Abbildung 3.3.: EDEN ISS von außen[3]

Über den gesamten Zeitraum konnten 1014 kg Gemüse wie Salat, Gurken und Tomaten geerntet werden. Die Arbeit mit lebenden Pflanzen stellte eine willkommene Abwechslung in dieser rauen Umgebung dar und wirkte sich positiv auf die mentale Gesundheit der Besatzung aus.

Im Container waren trotz Automatisierung etwa drei Stunden menschliche Arbeitszeit pro Tag notwendig. Es konnten verschiedene Einsparpotentiale identifiziert werden, welche der Besatzung 40 % weniger Arbeitszeit im Gewächshaus verschaffen sollten. Diese Verbesserungen sollen nun von den zuständigen SystemingenieurInnen im Projekt EDEN LUNA umgesetzt werden.[4]



Beim Betrieb fiel auf, dass viele Komponenten unterdimensioniert waren. So waren beispielsweise das Waschbecken, der Arbeitsbereich und die Gänge zu klein.

Weiterhin gab es einige Probleme mit der Reinigung. Ursprünglich war es vorgesehen, die Trays, in denen die Pflanzen wachsen, in der benachbarten Neumayer-Station III zu säubern. Der Transport erwies sich jedoch als zu aufwendig, sodass die Reinigung zukünftig im Container umgesetzt werden sollte. Des Weiteren zeigte sich im Betrieb, dass auch das Verlegen der Kabel und Rohre überdacht werden muss, denn eine Reinigung um eben diese herum erwies sich als zeitaufwendig.

Darüber hinaus wurde notiert, dass in Zukunft die Konfiguration der Pflanzen-Trays modular gestaltet werden sollte. So können die einzelnen Pflanzen je nach Bedarf unterschiedlich viel Platz zum Wachsen erhalten. Besonders bei hochwüchsigen Pflanzen wie Tomaten oder Gurken erwies sich die bisherige Aufteilung als problematisch.[5] All diese und weitere Probleme sind während der Durchführung von EDEN ISS aufgetreten und sollen von den zuständigen SystemdesignerInnen beim Bau von EDEN LUNA berücksichtigt und verbessert werden.

Es ist zudem hervorzuheben, dass die Software von EDEN ISS nicht für den Einsatz im Raum ausgelegt ist, sondern von einem auf Agrarwirtschaft spezialisierten Unternehmen entwickelt wurde. Die nächste Iteration wird unter Berücksichtigung raumfahrttechnischer Standards implementiert.

Ebenso wurden einige Verbesserungsmöglichkeiten speziell beim AMS festgestellt. Diese sind in Kapitel 6.2 beschrieben.

### **3.3. Auf der ISS eingesetzte Technologien**

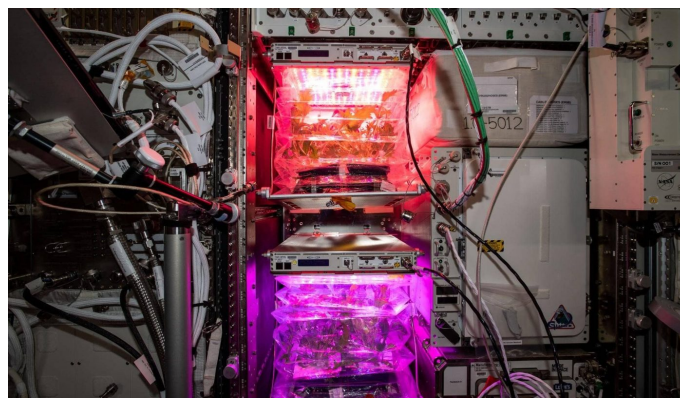
2014 wurde Veggie im Columbus Modul der ISS installiert. Dies ist ein Forschungsprojekt zur Untersuchung von Pflanzenwachstum in der Schwerelosigkeit. Entwickelt wurde es von ORBITEC mit der Unterstützung der NASA. Mit seinem kompakten Formfaktor bietet es genug Raum für sechs Pflanzen. Jede dieser Pflanzen wurde in ein lehm-basiertes "Kissen" eingebettet, um Wasser, Dünger und Luft in einem

### 3.3. Auf der ISS eingesetzte Technologien

---

gesunden Maße um die Wurzeln zu verteilen. Dies ist besonders wichtig, da Flüssigkeiten in der Schwerelosigkeit dazu tendieren, Blasen zu formen. Es wurden LEDs installiert, welche den Pflanzen unter anderem als Orientierung dienen. Dies ist im Weltraum von besonderer Bedeutung, da es keine Gravitation gibt, an der sie ihre Position festmachen können.

Mit diesem System wurden mittlerweile eine Vielzahl von verschiedenen Pflanzen angebaut, jedoch ist hier viel menschliche Arbeit notwendig.[6][7] In Abb. 3.4 ist Veggie zu sehen.



**Abbildung 3.4.:** Veggie[8]

Das Advanced Plant Habitat, welches ebenfalls auf der ISS steht, ist eine automatisierte Wachstumskammer, ausgestattet mit Kameras und über 180 Sensoren. Das System hat eine Grundfläche von 0.2 m<sup>2</sup>, eine Höhe von 0.4 m und wurde ebenfalls von ORBITEC und NASA entwickelt. [9]

Ein erster Testlauf des Systems startete 2018 und im Gegensatz zu Veggie bedarf es hier nur wenig Aufmerksamkeit der Besatzung. Die Wasserverteilung und -rückgewinnung, die Luftzusammensetzung und die Temperatur werden automatisiert geregelt. Weiterhin gibt es zeitgeschaltete LEDs, um die Pflanzen mit optimalen Lichtmengen zu versorgen. Das Projekt dient der Untersuchung der Beziehung zwischen der Schwerelosigkeit und dem Ligningehalt der Pflanzen. Lignin hat Funktionen, die am ehesten mit denen der menschlichen Knochen vergleichbar sind. Es soll also untersucht werden, wie standfest Pflanzen im All wachsen.[6] Abb. 3.5 zeigt ein Foto

### 3.4. Weitere Ansätze aus der Wissenschaft

---

des Advanced Plant Habitat.



**Abbildung 3.5.:** Advanced Plant Habitat[10]

Beide Projekte können als Vorbild für EDEN LUNA und weitere zukünftige Iterationen dienen. Da die EDEN Projekte darauf abzielen, Pflanzen auf dem Mond oder Mars zu züchten, ist es wichtig, Wissen über das Pflanzenwachstum in Gebieten mit weniger Schwerkraft zu haben und Ansätze zu kennen, welche das Wachstum verbessern.

Weiterhin besitzt das Advanced Plant Habitat ähnliche Umweltregulierungstechnologien, wie sie in EDEN LUNA verbaut werden sollen. Auch wenn es in einem deutlich kleineren Stil passiert, kann auch hier viel Wissenswertes entnommen werden.

### 3.4. Weitere Ansätze aus der Wissenschaft

Um weitere wissenschaftliche Ansätze, die für diese Arbeit relevant sein könnten, kennenzulernen, wurde Recherche zu ähnlichen Studien auf diesem Forschungsgebiet betrieben. Dabei fällt auf, dass zahlreiche Projekte in einem deutlich kleineren Maßstab durchgeführt werden.

In vielen Fällen wird die Arduino Plattform genutzt, um automatisierte Gewächshäuser für den privaten Gebrauch zu bauen. So sollen in einem Projekt Pilze und

### 3.4. Weitere Ansätze aus der Wissenschaft

---

nicht Pflanzen automatisiert gezüchtet werden.[11]

In einer Arbeit von Rajalakshmi ist die Rede davon, dass bis zu 70% Wasser beim automatisierten Anbau von Pflanzen gegenüber der herkömmlichen Landwirtschaft eingespart werden könne.[9]

Die beschriebenen Projekte haben keinen direkten Nutzen für diese Arbeit, da sie weder hinsichtlich ihres Umfangs noch ihres Konzepts mit EDEN LUNA vergleichbar sind. Im Gegensatz zu den irdischen Anwendungen, die von etablierten Umgebungen profitieren, stellt die Weltraumanwendung von EDEN LUNA eine völlig andere Herausforderung dar. Dennoch ist es interessant, wie vielfältig die Ansätze in der Wissenschaft für automatisierte Gewächshäuser sind.

## 4. Umfeld, Rahmen und Voraussetzungen

Im Folgenden wird diese Arbeit in den größeren Rahmen von EDEN LUNA eingeordnet. Weiterhin werden sowohl Software- als auch Hardwarevoraussetzungen und Grundlagen beschrieben. Zuletzt werden zentrale Begriffe erklärt.

### 4.1. Einordnung in EDEN LUNA

EDEN LUNA ist eine Analogmission, die darauf abzielt, die Funktionalität eines realen Mondmoduls nachzubilden. Um Kosten zu minimieren, wird lediglich ein Engineering-Modell erstellt, das die wesentlichen Funktionen des geplanten Systems repräsentiert. Dieses wird nicht raumfahrttauglich sein.

EDEN LUNA besteht aus einem automatisierten Gewächshaus und einem Arbeitsbereich für Menschen. Es ist vorgesehen, dass eine Besatzung im Container arbeitet, welche sich um die Pflanzen kümmert, das System überwacht und bei Bedarf steuert.

Ein Operator steuert das System im Mission Control Center, indem er mit einem grafischen Interface namens **Command Exchange Link (CEL)** interagiert. Dieses kann Telekommandos erstellen und versenden, welche dann an den zentralen Knoten, das **Data Handling & Control System (DHCS)** gesendet werden. Diese Telekommandos können beispielsweise einen neuen Ziel-CO<sub>2</sub>-Bereich definieren. Das DHCS empfängt diese Daten und verteilt sie an die entsprechenden Subsysteme. Im genannten Beispiel würde das DHCS ein Telekommando an das AMS versenden, welches diesem befiehlt,

den Ziel-CO<sub>2</sub>-Bereich entsprechend anzupassen.

Die Rückrichtung funktioniert analog dazu. Von den Subsystemen generierte Telemetrie wird an das DHCS versendet. Die Telemetrie könnte zum Beispiel Sensormesswerte und Aktorenzustände enthalten. Das DHCS empfängt sie und leitet sie an CEL weiter, wo sie gespeichert wird. Dort sind die Daten im Mission Control Center abrufbar, um wissenschaftliche Analysen vorzunehmen.

Das DHCS verteilt die Telekommandos an verschiedene Subsysteme. Neben dem AMS, welches Thema dieser Arbeit ist, gibt es das **N**utrient **D**elivery **S**ystem (NDS), das **T**hermal **C**ontrol **S**ystem (TCS), das **I**llumination **C**ontrol **S**ystem (ICS) und das Power System.

Diese und weitere Schnittstellen werden in Kapitel 6.2.5 genauer erläutert.[12]<sup>1</sup>

Die Unterteilung der Software in mehrere Subsysteme ist in Abb. 4.1 dargestellt.

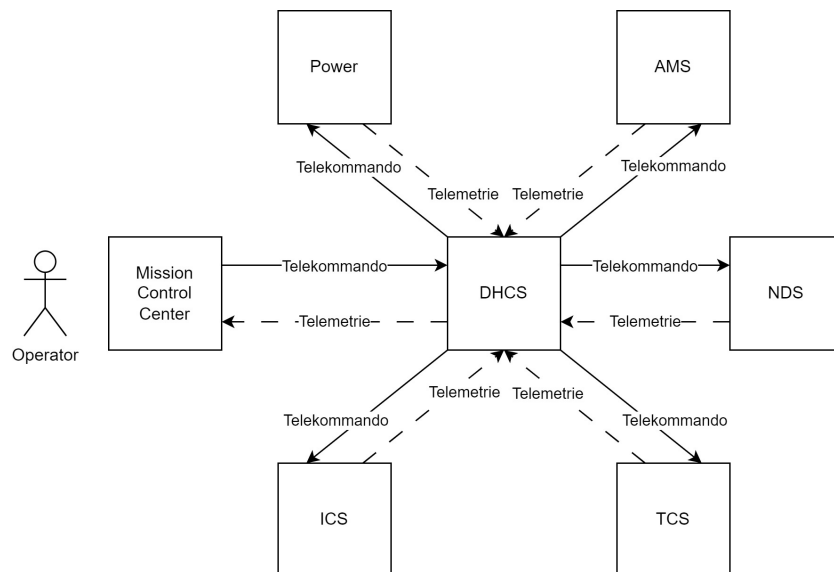


Abbildung 4.1.: Übersicht Subsysteme

---

<sup>1</sup>interne Dokumentation; nicht frei verfügbar; siehe Anhang A

## 4.2. Software Voraussetzungen

Als Programmiersprache wird C++17 in Verbindung mit dem SCons Build-System verwendet. Entwickelt wird unter Debian 11 mit dem GNU C Compiler 13.2.0. Diese Voraussetzungen sind vom fachlichen Betreuer gesetzt.

Alternativ zu SCons wäre es auch möglich gewesen CMake zu nutzen, da das verwendete Framework (siehe Kapitel 4.3) seit kurzem auch dieses Build-System unterstützt. Ein Unterschied besteht darin, dass CMake zur Konfiguration des Build-Prozesses seine eigene domänenspezifische Sprache nutzt, welche an C angelehnt ist. SCons hingegen nutzt die gängige Programmiersprache Python. Diese kann auch in ihrem vollen Umfang genutzt werden, was die Konfiguration leicht erweiterbar macht. Nachteilig allerdings ist, dass auf Plattformen, welche mit SCons bauen, Python installiert sein muss. CMake nutzt, konfiguriert und orchestriert im ersten Schritt andere Build-Tools wie Make oder Ninja und nutzt diese im zweiten Schritt für den eigentlichen Build-Prozess. Durch diese Zweiteilung kann der Build-Prozess mit CMake mehr Zeit beanspruchen als mit SCons, welches seine eigenen Tools für den Build-Prozess mitbringt.[13]

SCons bietet viele Freiräume, was jedoch nicht nur Vorteile mit sich bringt. Diese Flexibilität kann häufig zu komplexen und stark variierenden Build-Prozessen führen. So kann es sein, dass in unterschiedlichen Projekten völlig unterschiedliche Build-Verfahren zum Einsatz kommen können. Diese Variabilität erschwert insbesondere für Einsteiger die Übertragung von Wissen über den Build-Prozess von einem Projekt auf ein anderes. Im Gegensatz dazu bietet CMake weniger Freiheitsgrade, was zu einem standardisierten und oft leichter nachvollziehbaren Build-Prozess führt. Auch wenn CMake es erschwert, ungewöhnliche oder stark angepasste Build-Szenarien umzusetzen, erleichtert es durch seine strukturierte Herangehensweise und Konsistenz das Verständnis für die Anwender.

Die Benutzung von SCons war vom Projektgeber gesetzt, dennoch wäre CMake eine Alternative gewesen.

## 4.3. Softwarebibliothek OUTPOST

Um die Entwicklung missionskritischer (Flug-)Software am Institut für Raumfahrtssysteme effizienter zu gestalten, wird die intern entwickelte C++-Bibliothek OUTPOST (**O**pen **m**o**d**ular **s**o**f**tware **P**latf**O**rm for **S**pacecra**f**T) eingesetzt. Die Verwendung des Frameworks in dieser Arbeit ist vom fachlichen Betreuer vorgegeben. OUTPOST bringt zahlreiche Datenstrukturen und Funktionalitäten mit, um die Entwicklung neuer Systeme und Missionen auf die missions- und systemspezifischen Aspekte konzentrieren zu können. OUTPOST ist modular aufgebaut, so können NutzerInnen wählen, welche Teile sie verwenden möchten. Die Bibliothek ist ohne Portierung nutzbar unter allen Posix-kompatiblen Betriebssystemen und den Echtzeitbetriebssystemen FreeRTOS und RTEMS.

Die Kernsoftware besteht aus verschiedenen Modulen, welche unterschiedliche Funktionen für Raumfahrtanwendungen mit sich bringen. Dem Entwickler wird zum Beispiel im Umgang mit Zeit, Kommunikation, Missionsparametern und Tests geholfen. Außerdem gibt es eine Abstraktion für Echtzeitbetriebssysteme und viele weitere nützliche Tools, welche auf verschiedenen Ebenen Anwendung finden können. Es gibt Unterstützungen von der Treiber- bis hin zur Applikationsentwicklung. [14]

Die Kommunikation ist eine zentrale Komponente in der Entwicklung von Satelliten. Hier spielt die zuverlässige Übertragung von Daten eine Schlüsselrolle. Eine wesentliche Funktion von OUTPOST ist die Serialisierung und Deserialisierung von Daten, da diese in einen Bitstrom umgewandelt werden müssen, um eine effektive Übertragung zu gewährleisten.

In der Raumfahrt sollten strenge Standards, wie die des **C**onsultative **C**ommittee for **S**pace **D**ata **S**ystems (CCSDS) und des **P**acket **U**talization **S**tandard (PUS), eingehalten werden. Sie stellen sicher, dass Kommunikationsprotokolle und Datenformate den Anforderungen entsprechen und mit anderen Systemen kompatibel sind. Die Einhaltung solcher Standards ist jedoch komplex. OUTPOST erleichtert diesen Prozess erheblich, indem es deren Umsetzung weitgehend automatisiert.

Da Satellitensoftware eine extrem hohe Zuverlässigkeit aufweisen muss, ist umfassendes Testen unerlässlich. OUTPOST bietet hierfür wertvolle Unterstützung, indem es Hardwareabstraktionen bereitstellt, die es ermöglichen, Softwarekomponenten



unabhängig von der physischen Hardware zu testen. Für viele Schnittstellen, wie etwa CAN-Bus oder UART, stehen Stubs zur Verfügung, die die Unittests erleichtern. Auch in Kommunikationsimplementierungen können einzelne Protokollschichten isoliert getestet werden.

OUTPOST ist in verschiedene Repositories aufgeteilt: Core, Satellite und Unterstützungen für unterschiedliche Plattformen wie Posix. Dies hat den Hintergrund, dass weite Teile der Software flugspezifisch sind, für militärische Zwecke genutzt werden können und deshalb nicht veröffentlicht werden dürfen. Eine ältere Version von OUTPOST Core[14] ist als einzige frei verfügbar.

All diese Repositories werden parallel zueinander weiterentwickelt und gegeneinander getestet. Hin und wieder kommt es vor, dass Abhängigkeiten zwischen den unterschiedlichen Repositories ungewollt einseitig verändert werden, was sie inkompatibel zueinander macht.

Um nicht von derartigen Änderungen abhängig zu sein, macht es Sinn, zu Beginn eines Projektes einen kompatiblen Stand der verschiedenen benötigten Repositories abzurufen und diesen nur bei Bedarf zu ändern. Dies könnte notwendig sein, falls ein Fehler auftritt oder ein Feature fehlt, das es nur in einer neueren Version gibt.

Der Einsatz von OUTPOST bietet, insbesondere in der Raumfahrt, eine bedeutende Unterstützung für verschiedene Anwendungen. Allerdings ist die Nutzung nicht in allen Bereichen zwingend erforderlich. Beispielsweise wäre es unter Linux ausreichend, einen Thread der C++-Standard-Bibliothek anstelle eines OUTPOST-Threads zu verwenden, da beide unter POSIX-basierten Betriebssystemen auf den gleichen POSIX-Thread zurückgreifen. Der Vorteil von OUTPOST liegt jedoch darin, dass es die betriebssystemspezifische Funktionalität, in diesem Fall den Thread, betriebssystemabhängig übersetzt, sodass der Code des Nutzers betriebssystemunabhängig bleibt. Unter FreeRTOS ist die direkte Nutzung eines Standard-C++-Threads ohne eine zusätzliche Abstraktionsschicht nicht möglich[15][16], da es sich um ein leichtgewichtiges Echtzeitbetriebssystem ohne vollständige Unterstützung der C++-Standardbibliothek handelt. Der OUTPOST-Thread greift hingegen auf die betriebssystemspezifischen Mechanismen zurück und fügt weitere Abstraktionen hinzu.

Diese Fähigkeit zur betriebssystemunabhängigen Austauschbarkeit zeigt sich auch in anderen Teilen der Bibliothek, wie beispielsweise bei der UART-Unterstützung. Im

Rahmen dieses Projekts wird erwogen, zu einem späteren Zeitpunkt die einzelnen Subsysteme auf separaten Mikrocontrollern mit Echtzeitbetriebssystemen auszuführen. In einem solchen Szenario wäre die Fähigkeit zur betriebssystemübergreifenden Austauschbarkeit von entscheidender Bedeutung.

## 4.4. Hardware Voraussetzungen

Die komplette EDEN LUNA Software inklusive aller Subsysteme (siehe Abb. 4.1) wird auf einem Computer mit einer Linux Distribution als Betriebssystem (siehe Kapitel 4.4.1) ausgeführt. Für die Regelung der Umweltparameter werden Sensoren ausgelesen und Aktoren geschaltet. Im Rahmen dieser Arbeit wird die zugehörige Steuerungselektronik betrachtet. Hierfür werden einerseits Remote Input/ Output (IO) Geräte verwendet (siehe Kapitel 4.4.2). Andererseits wird bei Geräten, die eine UART Verbindung benötigen, diese direkt mit dem Computer hergestellt. All diese Vorgaben stammen vom fachlichen Betreuer.

### 4.4.1. Spectra PowerBox

Als zentrale Recheneinheit für alle Subsysteme wird ein Rechner der Spectra Power-Box 3000E Serie [17] gewählt. Dieser hat einen kompakten Formfaktor und kann problemlos in einem Schaltschrank verbaut werden. Am Rechner sind Halterungen angebracht, mit denen der Computer an Profilschienen befestigt werden kann. Weiterhin hat der erwähnte Schaltschrank keine Lüftung. Der Industrie-Computer ist in der Lage, die gegebenen Belastungen zu bewältigen, ohne dass es zu einer Überhitzung kommt. Die Verwendung dieses Rechners ist vom fachlichen Betreuer gesetzt.

### 4.4.2. Remote IO Geräte von Brainboxes

Zum Auslesen von Sensordaten und Ansteuern von Aktoren werden die Remote IO Geräte von Brainboxes (siehe Abb. 4.2) verwendet. Die Brainboxes können von einem herkömmlichen Computer aus gesteuert werden und sind deshalb für dieses

#### 4.5. Verwendete Begriffe

---

Projekt sehr nützlich. Außerdem wird kein zusätzlicher elektronischer Aufbau mit Relais benötigt, weil sie bereits enthalten sind.



Abbildung 4.2.: Brainboxes ED-538[18]

Die Brainboxes werden über eine Ethernet-Schnittstelle angesteuert. Es gibt die Geräte in verschiedenen Ausführungen, die Kombinationen aus digitalen Inputs und Outputs, analogen Inputs und Outputs, Relais, Resistance Temperature Detector Inputs und Thermocouple Inputs enthalten.

In diesem Projekt werden Geräte mit digitalen und analogen Inputs und Relais benutzt. Die verbauten Relais sind "normally open". Das bedeutet, dass sie geöffnet sind, wenn der zugehörige Pin nicht unter Spannung steht. Sie sind für den Betrieb bei Strömen bis zu 5 A bei 30 V Gleichstrom oder 250 V Wechselstrom ausgelegt. Die Brainboxes selbst werden mit 24 V Gleichstrom versorgt.[19]

## 4.5. Verwendete Begriffe

Im Folgenden werden zentrale, in dieser Arbeit verwendete Begriffe erklärt.

### **4.5.1. Telekommando**

Bei einem Telekommando wird von der Bodenstation oder dem Kontrollzentrum aus eine Nachricht über eine Telekommunikationsverbindung an ein ferngesteuertes Gerät gesendet. In der Raumfahrt werden Telekommandos verwendet, um bestimmte Aktionen auf Raumfahrzeugen auszulösen. Die Anweisungen können beispielsweise die Aktivierung bestimmter Instrumente, das Ändern von Flugbahnen oder das Einstellen von Betriebsparametern umfassen. Auf ein Telekommando wird oft<sup>2</sup> mit Telemetrie geantwortet.[20]

### **4.5.2. Telemetrie**

Telemetrie wird in der Raumfahrt genutzt, um Daten von Raumfahrzeugen zur Erde zu übertragen. Diese Daten können zum Beispiel den Status, die Position oder die Geschwindigkeit des Raumfahrzeuges oder Informationen über die Nutzlast enthalten.

Sie ermöglicht es WissenschaftlerInnen, den Zustand des Raumfahrzeugs zu überwachen, Fehler zu erkennen und bei Bedarf Anpassungen vorzunehmen. Zudem können so Messdaten zentral, zum Beispiel an einer Bodenstation, gesammelt werden.[20]

---

<sup>2</sup>immer; im Kommunikationsstandart, der in diesem Projekt verwendet wird (siehe Kapitel 6.2.4)

# 5. Strukturierte Aufgabenstellung

In diesem Kapitel werden die Aufgaben, welche in dieser Arbeit bearbeitet werden sollen, aufgezählt.

1. Die entwickelte Software für das **Atmosphäre Management System** soll in der Lage sein mit Sensoren und Aktoren zu kommunizieren.
  - 1.1 Von allen verwendeten Sensoren sollen Messwerte erhalten werden können.
  - 1.2 Es ist erforderlich, dass sämtliche verwendete Aktoren von der Software gesteuert werden können. Dies bedeutet, dass sie entsprechend ihres Typs ein- und ausgeschaltet werden können oder, beispielsweise bei Lüftern, die Drehgeschwindigkeit regulierbar ist.
2. Es sollen Hardwaretests für die entwickelten Treiber der Sensoren und Aktoren durchgeführt werden.
  - 2.1 Die Treiber der Sensoren sollen validiert werden.
  - 2.2 Die Aktoren sollen angesteuert werden, um die Funktionalität der Treiber zu bestätigen.
3. Die Software soll die Umweltparameter auf definierte Sollwerte regeln. Dies soll durch Regelkreise, welche Sensoren lesen und Aktoren schalten, realisiert werden. Die Umweltparameter sind:
  - 3.1 Temperatur
  - 3.2 Luftfeuchtigkeit
  - 3.3 Luftgeschwindigkeit
  - 3.4 CO<sub>2</sub>-Gehalt
4. Die Regelungen sollen im Rahmen von Unittests getestet werden. Es soll geprüft werden, ob diese im Bereich von gültigen und ungültigen Parametern das erwartete Verhalten zeigen.

5. Die Software des Subsystems soll vom zentralen Knoten (siehe Kapitel 6.2.5) aus über Telekommandos steuerbar sein.
  - 5.1 Es soll Telekommandos zum Ein- und Ausschalten der einzelnen Aktoren und Regelkreise geben.
  - 5.2 Auf diesem Weg sollen auch die Stellwerte geändert werden können.
6. Die Software soll in der Lage sein, Telemetrie zu versenden.
  - 6.1 Diese enthält unter anderem Housekeeping-Daten, wie Sensorwerte und Zustände der Aktoren.
  - 6.2 Warnungen und Fehler müssen an den zentralen Knoten versendet werden können.
7. Zum Schluss soll ein Systemtest stattfinden, in dem verschiedene Telekommandos von der Software empfangen und verarbeitet und dann über die entsprechenden Treiber Aktoren geschaltet werden.

## 6. Anforderungen

Im Folgenden werden zunächst Anforderungen an EDEN LUNA allgemein beleuchtet. Dann wird spezifisch auf das **A**tmosphere **M**anagement **S**ystem eingegangen.

### 6.1. EDEN LUNA Allgemein

Im Rahmen der LUNA Projekte werden vom DLR in Zusammenarbeit mit der **E**uropean **S**pace **A**gency (ESA) Technologien entwickelt und getestet. Diese Technologien sollen der Erkundung und der wirtschaftlichen Nutzung des Mondes dienen. Am DLR in Köln soll eine Mondsimulationsumgebung entstehen, welche unter anderem das EDEN LUNA Gewächshaus enthält.

Die SystemdesignerInnen stellen sicher, dass das Gewächshaus weitestgehend autonom betrieben werden kann, um die Zeit, welche die Besatzung in Wartung und Betrieb investieren muss, zu reduzieren. Weiterhin kann so der Verbrauch von wertvollen Ressource wie Wasser und Energie minimiert und der Ertrag gesteigert werden. Die Überwachung der Pflanzengesundheit durch künstliche Intelligenz ist ein weiterer Aspekt der Autonomie. Die Besatzung kann frühzeitig gewarnt werden, um Maßnahmen zur Erhaltung der Gesundheit zu ergreifen. Die Überwachung der Pflanzengesundheit ist kein Teil dieser Arbeit.

Das Gesamtsystem besteht aus mehreren Subsystemen, welche verschiedene Funktionen abdecken, wie beispielsweise Nährstoffversorgung, Wasseraufbereitung oder Temperaturkontrolle. All diese müssen entwickelt, parametrisiert und getestet werden, wenn sie in einem Weltraumgewächshaus mit ausreichender Zuverlässigkeit eingesetzt

werden sollen.[12]<sup>1</sup>

## **6.2. Atmosphere Management System**

Das AMS ist ein Subsystem von EDEN LUNA und dient der Regelung von Temperatur, Luftfeuchtigkeit, Luftgeschwindigkeit, Sauerstoffgehalt und Kohlenstoffdioxidgehalt. Des Weiteren wird der Gehalt von Ethylen, Ozon und Partikeln in der Luft überwacht.

Das grobe Systemdesign des AMS wurde bereits zu großen Teilen von den zuständigen SystemdesignerInnen entwickelt. Es wurde von EDEN ISS übernommen und angepasst mit dem Wissen, welches aus dem Projekt gewonnen wurde (siehe 6.2.1). In dieser frühen Phase, in der sich das Projekt befindet, wird zur Softwareentwicklung Prototyping genutzt, um offene Fragen im Design zu klären. Einige Bereiche sind dabei bewusst nicht präzise festgelegt, damit im Rahmen des Prototypings Anpassungen vorgenommen werden können. Dieses Vorgehen ermöglicht es, die letzten Details iterativ zu verfeinern und das Design optimal auf die Anforderungen des Projekts abzustimmen.

Die in Kapitel 6.2.2 beschriebenen Regelkreise sollen in der Lage sein, die Systemparameter auf den definierten Sollbereichen zu halten. Hierzu sollen die in Kapitel 7.3 beschriebenen Sensoren und Aktoren genutzt werden. Um diese anzusteuern, müssen Hardwaretreiber implementiert werden.

Das Subsystem soll über Telekommandos steuerbar sein und Telemetrie generieren. Diese sind genauer in Kapitel 6.2.4 beschrieben.

### **6.2.1. Verbesserungen aus vergangenen Iterationen**

Das **Atmosphere Management System** durchlief bereits mehrere Iterationen. 2012 wurde es erstmals im EDEN LAB (siehe Kapitel 3.1) und 2017 in EDEN ISS (siehe Kapitel 3.2) verbaut. Dementsprechend wurde es von den Fachexperten stetig

---

<sup>1</sup>interne Dokumentation; nicht frei verfügbar; siehe Anhang A



verbessert. Nach dem Projektende von EDEN ISS wurden Probleme, welche in Zusammenhang mit dem AMS stehen, von den ursprünglich Forschenden zusammengetragen und mögliche Lösungen diskutiert. Die notierten Resultate wurden von SystemdesignerInnen zum Entwurf des AMS-Designs von EDEN LUNA genutzt. Im Folgenden sind einige Verbesserungen dargestellt.

Ein Problem bestand darin, dass sich ein Biofilm auf dem Luftentfeuchter bildete. Eine mögliche Lösung ist es, die UV-C Lampe zum Entfernen der verantwortlichen Mikroorganismen und die verbauten Filter vor den Entfeuchter zu setzen und nicht mehr dahinter, so wie es zuvor der Fall war.

Die Sensoren, welche den Druckunterschied vor und nach den einzelnen Filtern (siehe Kapitel 6.2.2) messen sollten, um festzustellen, in welchem Zustand diese sich befinden, funktionierten nicht und wurden deshalb aus dem Regelkreis entfernt. Durch ausgiebige Hardwaretests<sup>2</sup> wird in dieser Iteration sichergestellt, dass alle Sensoren funktionieren.

Weiterhin wurde sich vorgenommen, in zukünftigen Projekten Volatile Organic Compounds (flüchtige organische Verbindungen) (VOC), Ethylen und Sauerstoff in der Luft zu messen und zu regulieren, weshalb nun entsprechende Sensoren vorgesehen sind.[21]<sup>3</sup>

### 6.2.2. Umweltparameter und Regelkreise

Zur Steuerung der Umweltparameter sollen in dieser Arbeit verschiedene Regelkreise entwickelt werden. Des Weiteren sollen die Funktionen einiger Aktoren überwacht werden.

In diesem Abschnitt werden die groben Anforderungen an die Regelkreise beschrieben. Detailliertere Erläuterungen zu den genutzten Geräten befindet sich in Kapitel 7.3.

---

<sup>2</sup>beschriebene Hardwaretests sind kein Teil dieser Arbeit

<sup>3</sup>interne Dokumentation; nicht frei verfügbar; siehe Anhang A

### **Temperaturregelung**

Der Regelkreis zur Temperaturregelung besitzt sowohl die Fähigkeit, die Temperatur zu erhöhen als auch zu senken. Er besitzt eine Schnittstelle zum TCS. Diese Schnittstelle wird genauer in Kapitel 6.2.5 beschrieben.

### **Luftfeuchtigkeitsregelung**

Die Aufgabe der Luftfeuchtigkeitsregelung besteht darin, die Luftfeuchtigkeit unter einem Schwellenwert zu halten. Sie besitzt keine Funktion, die Luftfeuchtigkeit künstlich zu erhöhen. Auch diese Regelung hat eine Schnittstelle zum TCS (siehe Kapitel 6.2.5).

### **Luftgeschwindigkeitsregelung**

Die Luftgeschwindigkeitsregelung stellt sicher, dass sich die Geschwindigkeit der Luft, welche den Behandlungstrakt<sup>4</sup> verlässt, immer zwischen einer unteren und oberen Schranke befindet.

### **CO<sub>2</sub>-Regelung**

Bei der CO<sub>2</sub>-Regelung wird zwischen Tag und Nacht unterschieden. Diese regelt dementsprechend den CO<sub>2</sub>-Gehalt der Luft zwischen zwei Schwellenwerten. Tagsüber findet eine CO<sub>2</sub>-Düngung statt, da auf diese Weise der Ernteertrag und das Wachstum der Pflanzen positiv beeinflusst werden. Da es auf dem Mond keine Tag- und Nachtphasen gibt, wird die Zeit, in der das Licht im Gewächshaus eingeschaltet ist, als Tag definiert. Weiterhin sind Mechanismen enthalten, welche die Funktionalität der Aktoren überprüfen.

---

<sup>4</sup>dieser enthält die Aktoren zum Behandeln der Luft

### **Sauerstoffregelung**

Die Sauerstoffregelung hat die Aufgabe, den Sauerstoffgehalt in der Luft zu reduzieren, sobald ein gewisser Grenzwert überschritten wird.

Diese Regelung war zu Beginn dieser Bachelorarbeit vorgesehen, gegen Ende jedoch nicht mehr, da zunächst erprobt werden soll, ob sie benötigt wird oder der Sauerstoff im Gewächshaus ohne Regelung unter einem sinnvollen Wert bleibt. Die Regelung benötigt ein Loch in der Außenwand des Containers. Dies soll vermieden werden. In dieser Arbeit wird die Sauerstoffregelung mit betrachtet, da sie bereits eingeplant und implementiert ist. Sie wird jedoch nicht in der ersten Version von EDEN LUNA enthalten sein, sondern bei Bedarf nachgerüstet.

### **Filterüberwachung**

Der Zustand der zwei verbauten Luftfilterkomplexe wird überwacht. Je nach Zustand werden unterschiedliche Meldungen generiert, die unterschiedlich schwere Folgen haben. Ein Vor- und ein HEPA<sup>5</sup>- Filter bilden den ersten und ein Nachfilter den zweiten Komplex.

### **Stromflussüberwachung**

Einige Aktoren werden nicht über einen Regelkreis geschaltet, sondern sind dauerhaft in Betrieb. Die Stromversorgung dieser wird überwacht.

### **6.2.3. Betriebsmodi der Regelkreise**

Für das AMS werden verschiedene Betriebsmodi (siehe Abb. 6.1) vorgesehen.

Dadurch, dass einige gärtnerische Tätigkeiten nach wie vor von Menschen durchgeführt werden müssen und sogar ein Arbeitsbereich für eine Besatzung vorgesehen ist, ist zu beachten, dass sich Menschen im System befinden. Dies hat den Vorteil,

---

<sup>5</sup>Partikelfilter

dass im Fehlerfall Reparaturen vorgenommen und der Modus des Systems manuell gewechselt werden kann, aber auch den Nachteil, dass immer für die Sicherheit der Besatzung gesorgt werden muss. Diese Faktoren spiegeln sich auch in den einzelnen Betriebsmodi wider.

Im "OFF" Zustand ist der Strom in allen Bereichen deaktiviert, somit sind auch alle Aktoren und Regelungen ausgeschaltet. Die Stromversorgung wird über das Power System (siehe Kapitel 6.2.5) gesteuert. Bei Ankunft des entsprechenden Telekommandos findet der Wechsel in den "STARTUP" Modus statt.

Während des "STARTUP" Zustands wird der Strom eingeschaltet und das Subsystem hochgefahren. Softwarekomponenten werden initialisiert und es finden verschiedene Überprüfungen statt. Ein Beispiel wäre testweise für wenige Sekunden CO<sub>2</sub> einzugasen und zu überprüfen, ob dies erfolgreich durchgeführt wird. So kann sichergestellt werden, dass dieser Aktor ordnungsgemäß funktioniert. Wenn eine dieser Überprüfungen fehlschlägt, geht das AMS in den "SAFE" Zustand über, ansonsten in den "NOMINAL" Zustand. In beiden Fällen wird Telemetrie versendet, um das DHCS darüber in Kenntniss zu setzen.

"NOMINAL" ist der Zustand, in dem das AMS seine vorgesehene Funktionalität ausführt. Der Strom ist in allen Sektionen aktiviert und alle Regelungen sind aktiv. Auch Aktoren, welche nicht von Regelungen abhängen, sind eingeschaltet. Dieser Modus kann verlassen werden durch ein Telekommando, welches einen Wechsel zum "OFF" oder zum "SAFE" Zustand auslöst. Wenn ein schwerwiegender Fehler auftritt, wechselt das Subsystem selbstständig in den "SAFE" Modus und versendet dabei Telemetrie.

Im "SAFE" Modus ist der Strom im Gewächshaus und dementsprechend auch dort enthaltene Regelungen und Aktoren ausgeschaltet. Um die Sicherheit der Besatzung zu gewährleisten, sind die Regelungen in den anderen Bereichen, wo sich das Personal aufhält, weiterhin aktiv. Besagte Regelungen sind im aktuellen Projektstand noch nicht klar definiert und deshalb nicht implementiert (siehe Kapitel 12.2). Der Modus kann nur durch einen Wechsel in den "STANDBY" Modus über ein Telekommando verlassen werden.

Der "STANDBY" Zustand erfüllt die Mindestbedingungen, um Pflanzen und Menschen am Leben zu halten. Deshalb ist der Strom in allen Bereichen aktiv. Er ist

## 6.2. Atmosphere Management System

vorgesehen, um Wartungen durchzuführen. Hier muss der Strom der zu wartenden Baugruppe manuell ausgeschaltet werden, um die Sicherheit der Besatzung zu gewährleisten. Mittels eines weiteren Telekommandos wird der "NOMINAL" Modus wiederhergestellt.

Es ist anzumerken, dass im Rahmen dieser Arbeit nur die Modi "OFF", "NOMINAL" und "SAFE" implementiert werden. Die übrigen sind für einen späteren Zeitpunkt vorgesehen, da noch nicht festgelegt ist, wie die Überprüfungen zum "STARTUP" im Detail und die Mindestbedingungen beim "STANDBY" aussehen.

In Abb. 6.1 sind die verschiedenen Betriebsmodi und deren Übergänge grafisch dargestellt. *TC* steht für einen Moduswechsel beim Erhalt eines Telekommando und *TM* steht für das Versenden von Telemetrie beim Moduswechsel. Mit FEG ist das Gewächshaus gemeint. *SES* und *CP* beschreiben die Bereiche in denen tendenziell sich Menschen aufhalten. Mit *RQs* werden Anforderungen abgekürzt.

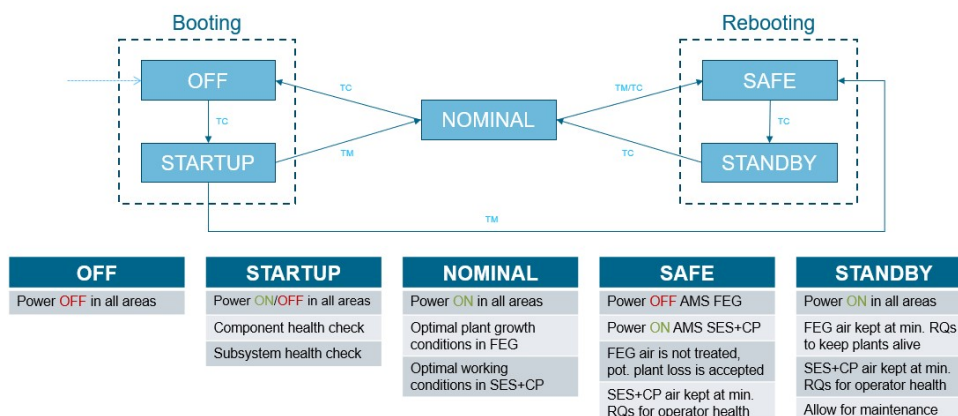


Abbildung 6.1.: Betriebsmodi und deren Übergänge

### 6.2.4. Telekommunikation

Die Telekommunikation basiert auf dem **Packet Utilization Standard** des CCSDS, dessen Ziel es ist, Formate und Prozeduren der Datenübertragung in Raumfahrtmissionen zu standardisieren. Dies ist vom fachlichen Betreuer gesetzt.

Einerseits soll es Telekommandos geben, um sowohl die einzelnen Aktoren als auch

die Regelkreise zu schalten. Andererseits sollen verschiedene Einstellungen im Subsystem, wie die zu erreichenden Sollwerte, geändert werden können. Weiterhin soll der Betriebsmodus des Subsystems änderbar sein.

Das Subsystem soll Sensordaten und die Zustände der Aktoren als Telemetrie zurückliefern. Außerdem sollen bei Problemen eventgeneriert Warnungen über den Zustand der Aktoren versendet werden können. Darüber hinaus muss gewährleistet sein, dass das Subsystem einen Moduswechsel anfragen kann, um beispielsweise in den "SAFE" Modus zu gelangen. Zudem soll es das Ein- und Ausschalten bestimmter Aktoren aus anderen Subsystemen anfordern können (siehe Kapitel 8.1.3 und 6.2.5).

### **6.2.5. Schnittstellen**

Im Folgenden werden Schnittstellen zwischen dem AMS und weiteren Subsystemen beschrieben.

#### **Data Handling & Control System**

Das DHCS ist die zentrale Schnittstelle zwischen den Subsystemen. Es versendet Telekommandos, welche die Subsysteme kommandieren. Weiterhin sammelt und speichert es die erhaltene Telemetrie.

#### **Thermal Control System**

Zur Regelung der Luftfeuchtigkeit und der Temperatur wird vom AMS ein Luftentfeuchter verwendet. Das Bauteil wurde dem **Thermal Control System** zugeordnet und wird von diesem gesteuert. Diese Einteilung wurde von den SystemdesignerInnen festgelegt und ist sinnvoll, da die Entwicklung und Steuerung dieses Bauteils viel Zeit in Anspruch nimmt. Das TCS hat weniger Komponenten als das AMS und somit mehr freie Ressourcen.

Zum Verwenden des Entfeuchters muss also Telemetrie an das DHCS gesendet werden, welches wiederum das TCS kommandiert den Aktor zu schalten.

### **Nutrient Delivery System**

Über den Dehumidifier im TCS wird bei Bedarf Feuchtigkeit abgeführt. Das gesammelte Wasser wird im NDS aufbereitet und den Pflanzen wieder in Verbindung mit Nährstoffen zugeführt.

Somit hat das AMS eine physische Verbindung mit dem NDS. Softwaretechnisch hat dies jedoch keine Auswirkungen.

### **Power System**

Alle verwendeten technischen Komponenten müssen mit Strom versorgt werden. Die Entwickler des Power Systems haben die verfügbaren Spannungen und maximalen Ströme spezifiziert. Entsprechend wurden von den SystemdesignerInnen Sensoren und Aktoren und im Rahmen dieser Arbeit Steuerhardware ausgewählt, die innerhalb dieser Vorgaben funktionieren.

### **6.2.6. Gewählte Reihenfolge**

Die tatsächliche Reihenfolge der Bearbeitung der Aufgaben weicht von der Gliederung in dieser Arbeit ab.

Zuerst wird die Hardware, welche für das Ansteuern der Sensoren und Aktoren von Bedeutung ist, ausgewählt und bestellt. Wenn diese eintrifft, wird sie zusammengebaut und, sofern möglich, direkt getestet.

Dann werden die Hardwaretreiber entwickelt und anschließend einem Test unterzogen. Dies hat den Hintergrund, dass alles Weitere auf der Steuerung der Aktoren und Sensoren aufbaut. Alternativ hätten zunächst Platzhalterklassen verwendet werden können. Das direkte Testen im Anschluss ist sinnvoll, da am Ende nicht so viele kleinere Tests auf einmal durchgeführt werden müssen. Falls es zu Fehlern kommt, können diese direkt behoben werden. Ein Nachteil besteht jedoch darin, dass zu diesem Zeitpunkt möglicherweise einige Komponenten noch nicht vorhanden sind und so nicht an der Hardware getestet werden kann.

Im Anschluss soll das System in die Lage gebracht werden, Telekommandos zu

empfangen, Telemetrie zu generieren und zu versenden. Diese grundlegende Funktionalität ist wichtig, um im nächsten Schritt vom DHCS über Telekommandos die Aktoren direkt steuerbar zu machen. Durch Abfrage per Telekommando sollen Housekeepingdaten wie Sensormesswerte und Zustände der Aktoren als Telemetrie versendet werden. Auch dies soll im Anschluss getestet werden.

Nach diesem Schritt sind die fehleranfälligen und kompliziertesten Implementierungen geschafft.

Zuletzt werden die Regelkreise implementiert. Hier gibt es erfahrungsgemäß die wenigsten Probleme. Es ist sinnvoll, sich eine solche Aufgabe zuletzt vorzunehmen. In diesem Schritt werden weitere Telekommandos und Telemetrien zur Steuerung der Regelkreise implementiert. Am Ende dieses Schrittes kann das Gesamtsystem softwareseitig getestet werden. Je nach Vorhandensein der Komponenten können auch weitere Hardwaretests durchgeführt werden.

Diese Vorgehensweise wurde vom fachlichen Betreuer vorgeschlagen und aus oben genannten Gründen so umgesetzt.

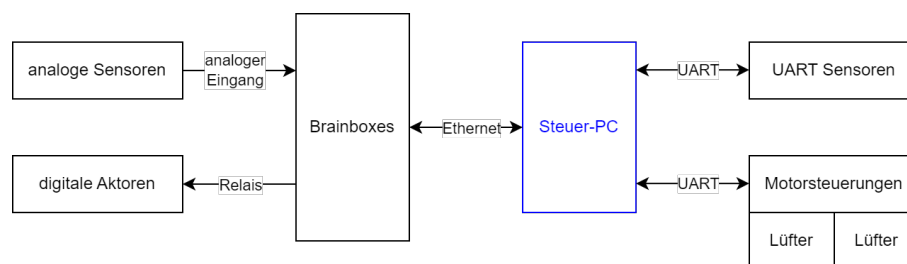


# 7. Hardwarearchitektur

In diesem Kapitel wird der Aufbau der zugrunde liegenden Hardware beschrieben. Des Weiteren wird auf Designentscheidungen eingegangen.

## 7.1. Aufbau

In Abb. 7.1 ist vereinfacht dargestellt, wie die einzelnen Hardwarekomponenten zusammenhängen. Im Zentrum steht der Steuer-PC, auf dem die Software der einzelnen Subsysteme ausgeführt wird. Über die Remote IO Devices von Brainboxes wird das analoge Signal von Sensoren ausgelesen und verschiedene Relais geschaltet, welche den jeweiligen Stromkreis schließen und auf diese Weise Aktoren mit Strom versorgen, wenn sie gebraucht werden. Über UART-Verbindungen wird direkt mit weiteren Sensoren kommuniziert. Auf diesem Weg findet auch die Verständigung mit den Motorsteuerungen (siehe Kapitel 7.4.2) statt, welche die Drehzahl von jeweils zwei Lüftern steuern.



**Abbildung 7.1.:** Übersicht der AMS-Hardwarekomponenten

## 7.2. Auswahl der Brainboxes

Voraussetzung für die Auswahl der Brainboxes ist eine Übersicht (siehe Abb. 7.2) mit den verwendeten Sensoren und Aktoren, welche auch deren Interfaces enthält. Die Komponenten waren teilweise aus dem Projekt EDEN ISS vorhanden und sollten aus Kostengründen wiederverwendet werden. Andere wurden mit den SystemdesignerInnen diskutiert und abgeändert, um möglichst oft gleiche Interfaces zu verwenden. Hintergrund dessen ist, dass dadurch ermöglicht wird, dass viele Geräte ähnliche Treiber verwenden können. Ein weiterer Aspekt ist, dass jeder Typ der Brainboxes Remote IO Devices nur ein oder zwei unterschiedliche Interfaces unterstützt (siehe Kapitel 4.4.2). Infolgedessen, dass viele Komponenten dasselbe Interface nutzen, werden weniger verschiedene Brainboxes benötigt und so Kosten eingespart. Zur Zeit der Bestellung der Hardware wurden 29 analoge Sensoren benötigt. Hinzu kommen elf Relais, welche die Aktoren schalten. Die restlichen Sensoren und Aktoren benötigen eine Verbindung über UART. Um diese einzurichten, sind keine Brainboxes, sondern lediglich Kabelverbindungen nötig.

#	Type	Parameter [unit]	Product	Amount	Comment	Digital Pins	Analog Pins
1	Oxygen sensor	O2 [0...25%]	S+S Regeltechnik AERASGARD® AO2-U	2 (SES+CP)	Input: 24V DC Output: 0...10V	0	2
2	VOC multi sensor	VOC [0...100%] CO2 [0...5000ppm], T [0...50°C], RH [0...100%].	S+S Regeltechnik AERASGARD® RFTM- LQ-CO2-W LCD	3 (FEG, SES+CP)	Input: 24V AC/DC Output: 0...10 V or 4...20mA	0	12
3	Particulate sensor	PM [0...500µg/m³]	S+S Regeltechnik AERASGARD® RPS- SD	3 (FEG, SES+CP)	Input: 24V AC/DC Output: 0...10V	0	3

Abbildung 7.2.: Hardwaretabelle Ausschnitt

Die Brainbox ED-549 besitzt acht analoge Eingänge [22], hiervon werden vier benötigt, um alle analogen Sensoren auslesen zu können. Vier Relais und vier Digitale Eingänge werden vom Typ ED-538 bereitgestellt [23], drei Geräte von diesem Typ genügen zum Schalten der Aktoren.

Es wurde jeweils noch ein weiteres Remote IO Device auf Reserve gekauft, denn die benötigten Sensoren und Aktoren sind noch nicht bestellt worden. So ist es möglich später flexibel Komponenten hinzuzugefügen oder auszutauschen, ohne erneut auf

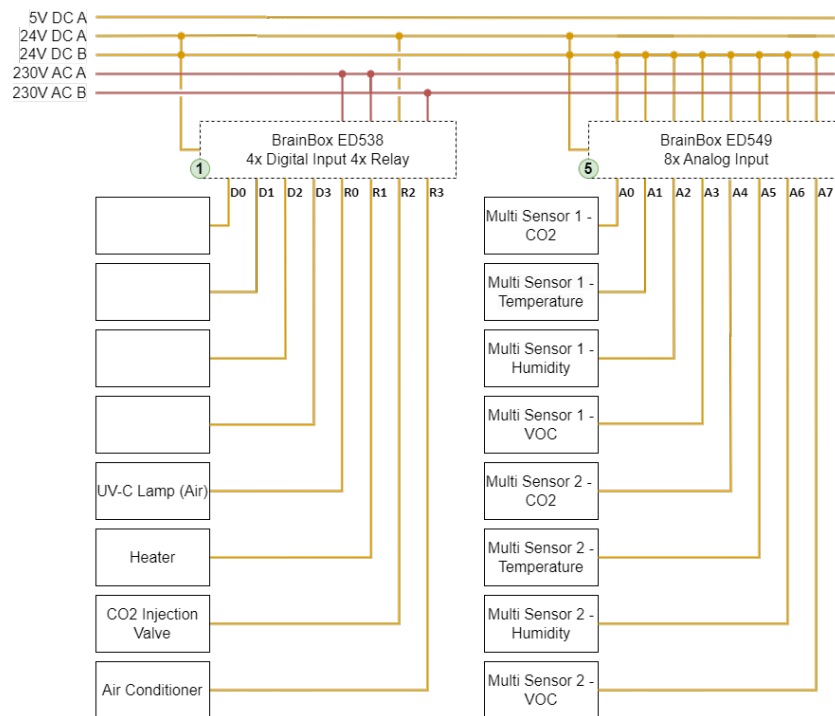
## 7.2. Auswahl der Brainboxes

---

die Bestellung warten zu müssen.

Alternativ hätten statt den drei ED-538 auch ein Gerät vom Typ ED-527 mit 16 digitalen Ausgängen genutzt werden können [24]. Diese hätte mit externen Relais verbunden werden müssen, welche die Aktoren schalten, da die maximale Spannung dieser Ausgänge für die meisten Aktoren zu klein ist. Dieser Weg wurde bewusst nicht gewählt, da, auch wenn auf diese Weise Kosten gespart werden könnten, das System komplexer werden würde. Zudem ist der Raum im Container begrenzt. Dies war ein weiterer Aspekt, der für das Verbauen der Relais direkt in den Brainboxes spricht.

Die gewählten Geräte sind in einem Diagramm zusammengefasst, in dem deren Ein- und Ausgänge den Pins der Brainboxes zugeordnet sind. Ein Ausschnitt ist in Abb. 7.3 zu sehen.



**Abbildung 7.3.:** Geräte - Brainboxes Zuordnung Ausschnitt

## 7.3. Hardware der einzelnen Regelkreise

Im Folgenden sind die in den einzelnen Regelungen verwendeten Bauteile beschrieben. Die zugehörigen Datenblätter der Geräte sind in einem Git Repository (siehe Anhang A) gesammelt.

### 7.3.1. Temperaturregelung

Der Regelkreis zur Temperaturregelung besteht aus einer 320 W Heizung<sup>1</sup> und hat zudem Kontrolle über einen Luftentfeuchter, welcher unter anderem der Kühlung dient. Dieser Luftentfeuchter ist dem TCS zugehörig. Diese Schnittstelle ist genauer in Kapitel 6.2.5 beschrieben.

Über ein Flowmeter<sup>2</sup> wird der Durchfluss durch das Kühlelement des Luftentfeuchters überprüft. Die Temperatursteuerung nutzt einen kombinierten Sensor<sup>3</sup>, welcher neben der Temperatur im Bereich von 0 bis 50 °C auch Luftfeuchtigkeit, CO<sub>2</sub>-Gehalt und VOC-Gehalt messen kann. Beide Sensoren geben ein analoges Signal aus.

### 7.3.2. Luftfeuchtigkeitsregelung

Die Luftfeuchtigkeitsregelung teilt sich mit der Temperaturregelung das Flowmeter<sup>4</sup> und den kombinierten Sensor<sup>5</sup>. Dieser wird hier zur Luftfeuchtigkeitsmessung, welche einen Bereich von 0 bis 100 % abdeckt, genutzt. Die Regelung steuert ebenfalls den dem TCS zugehörigen Luftentfeuchter an.

---

<sup>1</sup>Helios 01090009

<sup>2</sup>S+S Regeltechnik RHEASREG® KLGFT-W LCD

<sup>3</sup>S+S Regeltechnik AERASGARD® RFTM-LQ-CO<sub>2</sub>-W LCD

<sup>4</sup>S+S Regeltechnik RHEASREG® KLGFT-W LCD

<sup>5</sup>S+S Regeltechnik AERASGARD® RFTM-LQ-CO<sub>2</sub>-W LCD

### 7.3.3. Luftgeschwindigkeitsregelung

In dieser Regelung sind zwei 230 V Lüfter<sup>6</sup> mit einer maximalen Leistungsaufnahme von 750 W und einer maximalen Drehzahl von 3450 Umdrehungen pro Minute verbaut. Sie besitzen jeweils einen Volumenstrom von 1755 m<sup>3</sup>/h. Die Lüfter werden von einer Motoron M2U256 Motorsteuerung gesteuert (siehe Kapitel 7.4.2). Diese ermöglicht es, zwei Gleichstrommotoron, in diesem Fall die Lüfter, unabhängig voneinander in jede Richtung mit 800 Geschwindigkeitsstufen zu betreiben. Dabei können die Motoren mit 4.5 bis 48 V und jeweils maximal 1.8 A betrieben werden.

Weiterhin ist ein analoger Sensor<sup>7</sup> zur Messung der Luftgeschwindigkeit verbaut. Über diesen wird ermittelt, ob die Luftgeschwindigkeit zu klein oder zu groß ist. Sollte dies der Fall sein, wird die Geschwindigkeit der Lüfter über die Motorsteuerung kleinschrittig verändert.

### 7.3.4. CO<sub>2</sub>-Regelung

Über einen analogen CO<sub>2</sub>-Sensor wird in dieser Regelung der CO<sub>2</sub>-Gehalt der Luft bestimmt. Der CO<sub>2</sub>-Sensor ist Teil des kombinierten Sensors<sup>8</sup>, welcher auch in anderen Regelkreisen verwendet wird. Er hat einen Messbereich von 0 bis 5000 ppm. Wenn dieser zu niedrig ist, wird ein Ventil<sup>9</sup> geöffnet und das unter Druck stehende CO<sub>2</sub> kann aus dem Tank entweichen und reichert die Luft an. Um den eben dargestellten Vorgang zu überprüfen, ist ein weiteres analoges Flowmeter<sup>10</sup> verbaut, welches den CO<sub>2</sub>-Strom misst.

Sollte zu viel CO<sub>2</sub> in der Luft enthalten sein, wird eine 16 W Pumpe<sup>11</sup> aktiv, welche die Luft durch Atemkalk leitet. Dieses Substrat bindet das CO<sub>2</sub>[25] und verfärbt sich dabei. Zur Kontrolle dieses Vorganges gibt es einen Farbsensor<sup>12</sup>, welcher die Verfärbung überprüft. Dieser hat eine Reaktionszeit von 400 ms.

---

<sup>6</sup>Ebm papst K3G250AV29B4

<sup>7</sup>E+E Elektronik GmbH EE671

<sup>8</sup>S+S Regeltechnik AERASGARD® RFTM-LQ-CO<sub>2</sub>-W LCD

<sup>9</sup>Nadi C03I18DOP

<sup>10</sup>E+E Elektronik GmbH EE741

<sup>11</sup>Aspen Pumps mini tank pump

<sup>12</sup>Atlas Scientific EZO-RGB

### 7.3.5. Sauerstoffregelung

Die Sauerstoffregelung enthält einen analogen Sauerstoffsensoren mit einem Messbereich von 0 bis 25 % Luft-Sauerstoffgehalt<sup>13</sup> und zwei Linearmotoren, welche dem Öffnen einer Belüftungsklappe dienen. Wenn der Sauerstoffgehalt im Gewächshaus zu hoch ist, wird diese Klappe geöffnet, sodass ein Luftaustausch mit der Außenwelt stattfinden kann. Dieses Konzept funktioniert nur auf der Erde. Für die Linearmotoren fand noch keine finale Auswahl statt, da diese Regelung im aktuellen Stand nicht mehr vorgesehen ist.

### 7.3.6. Filterüberwachung

Die Zustände des Vor- und HEPA-Filterkomplexes sowie des Nachfilters werden überwacht. Jeder der zwei Filterkomplexe erhält einen Sensor<sup>14</sup> zur Messung der Druckdifferenz mit jeweils einem Fühler vor und hinter dem Filter. Der gewählte Differenzdrucksensor ist einsetzbar für Druckdifferenzen bis 1000 Pa und hat eine Messgenauigkeit von  $\pm 0,5\%$ . Mittels des gemessenen Differenzwertes kann der Zustand des Filters ermittelt werden.

### 7.3.7. Überwachung sonstiger Aktoren

Im System sind weitere Aktoren verbaut, die nicht abhängig von einer Messgröße geschaltet werden, deren Funktionalität jedoch überwacht wird. Hierbei wird geprüft, ob über die Bauteile Leistung abfällt, wenn diese in Betrieb sein sollten. Darüber hinaus sind diese auch über Telekommandos einzeln ein- und ausschaltbar.

Im System sind weiterhin zwei UV-C Lampen verbaut, eine mit 12 V Gleichspannung zur Sterilisation von Wasser<sup>15</sup> und eine mit 230 V, 60 W Wechselspannung für die Luft<sup>16</sup>.

---

<sup>13</sup>S+S Regeltechnik AERASGARD® AO2-U

<sup>14</sup>E+E Elektronik GmbH EE600

<sup>15</sup>Aquisense Pearlaqua PAQ-09D

<sup>16</sup>Light Progress UV-DUCT-FL 2/60HP-LA

Eine Pumpe<sup>17</sup> hält den CO<sub>2</sub> Tank dauerhaft unter Druck, sodass das Gas mit einer gesetzten Geschwindigkeit entweicht, sobald das Ventil geöffnet wird. Es handelt sich dabei um denselben Pumpentyp, der auch eingesetzt wird, um CO<sub>2</sub> zu entfernen.

## 7.4. Weitere Komponenten

Im Folgenden sind weitere Sensoren beschrieben, die keiner Regelung zugehörig sind. Darüber hinaus wird eine Motorsteuerung zum Festlegen der Lüfterdrehzahl vorgestellt.

### 7.4.1. Weitere Sensoren

Um die Überwachung und die Generierung wissenschaftlicher Daten zu unterstützen, kommen zusätzlich weitere Sensoren zum Einsatz. Diese Sensoren, die sowohl innerhalb als auch außerhalb der Regelkreise vorkommen, werden im Folgenden nicht noch einmal gesondert aufgeführt.

Zur Überwachung des Ethylengehalts der Luft wird ein analoger Sensor der Marke Oppermann eingesetzt<sup>18</sup> und auch ein analoger Ozonsensor desselben Herstellers ist verbaut<sup>19</sup>. Für die Messung von Feinstaub und Kleinstpartikeln wird ein analoger Sensor mit einem Ausgang von 0 bis 10 V verwendet.<sup>20</sup>

Des Weiteren sind drei verschiedene Atlas Scientific EZO Sensoren verbaut, die über UART mit dem ausführenden Rechner kommunizieren. Einer misst die Temperatur und Luftfeuchtigkeit<sup>21</sup>, einer den Sauerstoffgehalt<sup>22</sup> und einer den CO<sub>2</sub>-Gehalt<sup>23</sup> der Luft.

---

<sup>17</sup>Aspen Pumps mini tank pump

<sup>18</sup>Oppermann GMF 2.H.C2H4.30

<sup>19</sup>Oppermann Regelgeräte GMF 2.E.O3.00

<sup>20</sup>S+S Regeltechnik AERASGARD® RPS-SD

<sup>21</sup>Atlas Scientific EZO-HUM

<sup>22</sup>Atlas Scientific EZO-O<sub>2</sub>

<sup>23</sup>Atlas Scientific EZO-CO<sub>2</sub>

## 7.4.2. Motorsteuerung

Einige Lüfter sollen in der Lage sein, verschiedene Geschwindigkeiten anzunehmen. Um die Drehzahl der Lüfter einstellen zu können, werden diese an eine Motorsteuerung angeschlossen. Die verwendete Motorsteuerung von Motoron bietet den Vorteil, das lediglich ein einziges zusätzliches Bauteil für je zwei Motoren benötigt wird. Weiterhin ist die Motorsteuerung über UART ansteuerbar und bringt viele Funktionen, wie die Einstellung der maximalen Beschleunigung bereits mit.[26]

Alternativ hätte auch einen Transistor genutzt werden können, um die Spannung an den Lüftern zu regeln und somit deren Drehgeschwindigkeit zu steuern. Hier wäre es jedoch nötig gewesen, noch weitere Bauteile wie Dioden zum Schutz vor Rücklaufspannungen durch Induktion oder Kondensatoren, um etwaige Funksignale vor Störungen zu schützen, einzubauen. All dies wird in einem einzigen Bauteil, einer sogenannten H-Brücke, vereint.[27]

Wegen der zahlreichen eigenbauten Funktionen, dem integrierten Rückspannungsschutz und weil es ein einziges Bauteil ist, welches per UART angesteuert wird, also kein weiteres IO-Device benötigt, fiel die Wahl auf die Pololu Motoron M2U256 Motorsteuerung. Wie diese Steuerung angeschlossen wird, ist in Abb. 7.4 dargestellt.

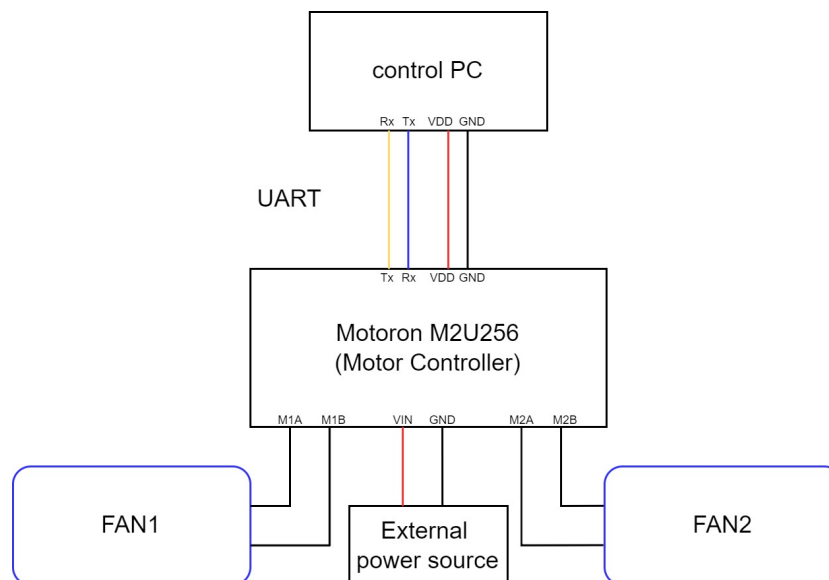


Abbildung 7.4.: Lüftersteuerung über Motorcontroller



## 8. Softwarearchitektur

Im Folgenden wird die Softwarearchitektur des AMS erklärt und auf die Implementierung einzelner Softwarekomponenten eingegangen.

In Abb. 8.1 ist beispielhaft dargestellt, wie einige Komponenten voneinander abhängen. Auf der unteren Ebene befinden sich die Hardwarekomponenten, die über verschiedene Zwischenschichten von den einzelnen Regelkreisen gesteuert werden. Darüber befinden sich die Schichten, welche die Regelkreise aufrufen.

AMS main.cpp			
Application (init & run)			
Air Speed Control Loop	Co2 Control Loop	Temperature Loop	Humidity Loop
Motor Driver	Atlas EZO Driver	Brainboxes Driver	
OUTPOST			
UART		Ethernet	
Motor Controller	Color Sensor	Brainboxes	
Fans		Heater	Multi Sensor

**Abbildung 8.1.:** Übersicht Software und Hardware

## 8.1. Implementierung der Komponenten

Im Folgenden wird die Implementierung der Hardwaredreiber, Regelungen und Telekommunikation beschrieben.

### 8.1.1. Hardwaredreiber

Zur Steuerung der Aktoren und Sensoren wurden verschiedene Hardwaredreiber implementiert.

#### **Motoron M2U256**

Die Implementierung orientiert sich an der bereits existierenden Bibliothek[28], welche für die Arduino-Plattform entwickelt wurde. Die bestehenden, für diesen Anwendungsfall relevanten, Methoden zum Erstellen der einzelnen Kommandos wurden größtenteils übernommen und mit der UART-Unterstützung der Softwarebibliothek OUTPOST verknüpft. Die Methoden stellen dabei die Basis für das Erzeugen der richtigen Bitreihenfolge zum Ansteuern der Motoren dar. Die UART-Unterstützung von OUTPOST ist so konzipiert, dass bei einer späteren Entwicklung für ein anderes Betriebssystem lediglich die Konfiguration angepasst werden muss. Die zugrunde liegenden Funktionen werden automatisch mit den für dieses Betriebssystem passenden ersetzt. Dies ist sehr hilfreich, da in Zukunft die einzelnen Subsysteme auf Mikrocontrollern mit anderen Betriebssystemen und nicht mehr unter Posix laufen sollen.

Es sind Methoden zum Setzen der Geschwindigkeit der beiden Motoren und verschiedene Abfragen, wie zum Beispiel die der Firmware oder der Eingangsspannung, implementiert.

#### **Atlas Scientific EZO**

Die Atlas Scientific EZO Sensoren lassen sich ebenfalls über UART ansteuern. Die einzelnen Kommandos und die zu erwartenden Antworten lassen sich in der zugehörigen

Dokumentation finden.[29] Auch hier wurde wieder die OUTPOST Implementierung des UART-Interfaces genutzt.

### **Brainboxes**

Da die Brainboxes bereits in Vorgängerprojekten genutzt wurden, besteht auch hier bereits eine grundlegende Implementierung. Diese wird genutzt, um einen Controller zu entwickeln, der alle setter- und getter-Methoden, welche wiederum die Ausgänge der verschiedenen Brainboxes setzen oder auslesen, enthält.

### **8.1.2. Value Store**

Der Value Store hat eine zentrale Aufgabe. Diese besteht darin, jegliche Parameter des Systems, welche von verschiedenen Klassen benötigt werden, zu speichern. Darunter fallen unter anderem alle Grenzwerte, der aktuelle Modus und die Zeiten des definierten Sonnenaufgang und -untergangs.

Bei Eingang eines Telekommandos zum Setzen der Grenzwerte werden die entsprechenden Informationen im Value Store abgelegt. Wenn die Regelungen diese benötigen, können sie aus dem Value Store abgefragt werden. Somit gibt es eine Zwischenschicht, die den Empfang von neuen Daten und die Stellen, an denen sie benötigt werden, entkoppelt.

### **8.1.3. Regelungen**

Jeder Regelkreis wird in einem separaten Thread ausgeführt. Auf diese Weise können alle Regelungen nebenläufig erfolgen. Die technische Thread-Klasse ist von der fachlichen Regelungsklasse getrennt. Diese "separation of concerns" verbessert die Testbarkeit und Wartbarkeit des Systems. Durch die isolierte Ausführung der Regelkreise wird die Fehlersuche erleichtert und die Modularität des Systems erhöht, was eine flexiblere Anpassung und Erweiterung ermöglicht.

Für jeden Regelkreis wird eine Klasse erstellt, welche eine Methode enthält, die einen

Durchlauf der Regelung darstellt. Hinzu kommen Methoden, mit denen Messwerte von außen abgefragt werden können und im Fall der CO<sub>2</sub>-Regelung eine Methode zur Bestimmung der aktuellen Tageszeit. Diese wurde aus der Hauptmethode ausgelagert, da sie so einzeln getestet werden kann und die Hauptmethode weniger komplex ist. Die Regelungsmethoden werden in verschiedene Thread-Objekte hineingegeben, welche diese dann mit einem einstellbaren Abstand ausführen.

Zum besseren Verständnis sei an dieser Stelle noch einmal wie bereits in Kapitel 6.2.5 erwähnt, dass der Entfeuchter kein Teil des AMS ist und somit per Telemetrie angefragt wird.

Zu Beginn jeder Regelung wird der aktuelle Modus (siehe Kapitel 6.2.3) aus dem Value Store abgefragt. Sollte dieser "OFF" oder "SAFE" sein, so werden alle Aktoren ausgeschaltet und es findet keine Regelung statt. Im Folgenden ist jeweils der "NOMINAL" Mode der Regelungen dargestellt.

In den folgenden Ablaufdiagrammen (siehe zum Beispiel Abb. 8.2) sind Überprüfungen enthalten, ob die Aktoren Strom verbrauchen, wenn sie aktiviert werden. Diese sind noch nicht implementiert im Rahmen dieser Arbeit, da noch unklar ist, auf welche Weise diese Überprüfung stattfinden soll.

### **Temperaturregelung**

In der Regelungsmethode dieser Klasse wird zunächst eine Methode des Hardwaretreibers der Brainboxes aufgerufen, welche die Abfrage des Temperaturmesswertes darstellt. Der analoge Wert des zugehörigen Pins wird abgefragt und zurückgegeben. Der obere und untere Temperaturgrenzwert werden aus dem Value Store (siehe Kapitel 8.1.2) abgefragt.

Der Messwert wird in Bezug auf die festgelegten Grenzwerte evaluiert, wobei drei mögliche Szenarien unterschieden werden.

Wenn der Messwert kleiner als der untere Grenzwert ist, wird die Heizung per Aufruf der entsprechenden Brainboxes-Treiber-Klasse aktiviert. Weiterhin wird Telemetrie generiert, die das TCS darüber informiert, dass der Luftentfeuchter, welcher gleichzeitig kühlt, von dieser Regelung nicht gebraucht wird. Diese Information wird ebenfalls

im Value Store abgelegt.

Im zweiten Fall liegt die gemessene Temperatur zwischen den beiden Grenzwerten. Hier wird die Heizung ausgeschaltet und wieder Telemetrie vorbereitet, die anzeigt dass der Luftentfeuchter von der Temperaturregelung gerade nicht gebraucht wird. Auch diese Information wird im Value Store abgelegt.

Im letzten Fall ist die Temperatur über dem oberen Grenzwert. Die Heizung wird ausgeschaltet und es wird Telemetrie generiert, um zu signalisieren, dass der Luftentfeuchter gebraucht wird. Der Value Store wird ebenfalls benachrichtigt. Dann ruht der Thread für eine festgelegte Zeit, um dem Entfeuchter die Möglichkeit zu geben, anzulaufen. In dieser Zeit kann die Luftgeschwindigkeitsregelung (siehe Kapitel 8.1.3) bei Bedarf die Drehgeschwindigkeit der Lüfter erhöhen, um den nötigen Luftstrom im Entfeuchter zu gewährleisten. Im Anschluss wird der Luftdurchfluss durch den Entfeuchter gemessen. Sollte dieser sich unter dem aus dem Value Store abgefragten Grenzwert befinden, wird ein Fehler generiert.

In jedem der drei Fälle wird am Ende die generierte Telemetrie weggesendet.

In Abb. 8.2 ist dieser Ablauf dargestellt.

## 8.1. Implementierung der Komponenten

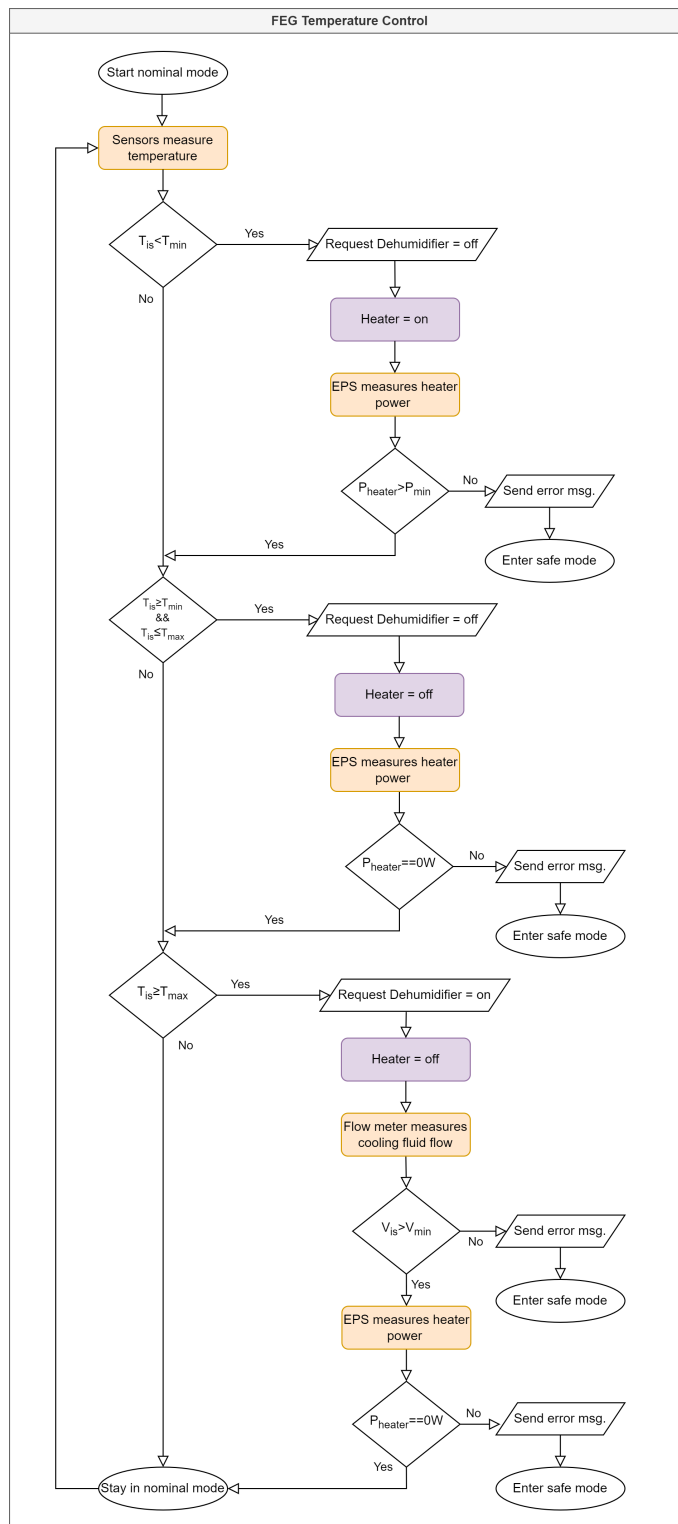


Abbildung 8.2.: Ablaufdiagramm Temperaturregelung

## Luftfeuchtigkeitsregelung

Zu Beginn dieser Regelung wird die relative Luftfeuchtigkeit über die Brainboxes gemessen. Sollte diese größer sein als der Grenzwert, welcher aus dem Value Store abgefragt wird, wird Telemetrie generiert, die anzeigt, dass der Entfeuchter von dieser Regelung benötigt wird. Auch hier wird diese Information zusätzlich im Value Store abgelegt.

Genau wie in der Temperaturkontrolle (siehe Kapitel 8.1.3) wird zunächst gewartet und gemessen, ob genug Durchfluss durch den Entfeuchter stattfindet.

Wenn der gemessene Wert unter dem Grenzwert liegt, wird per Telemetrie und im Value Store signalisiert, dass der Entfeuchter von der Luftfeuchtigkeitsregelung nicht benötigt wird.

Im letzten Schritt wird sämtliche Telemetrie übertragen.

Das Konzept ist in Abb. 8.3 in Form eines Ablaufdiagramms dargestellt.

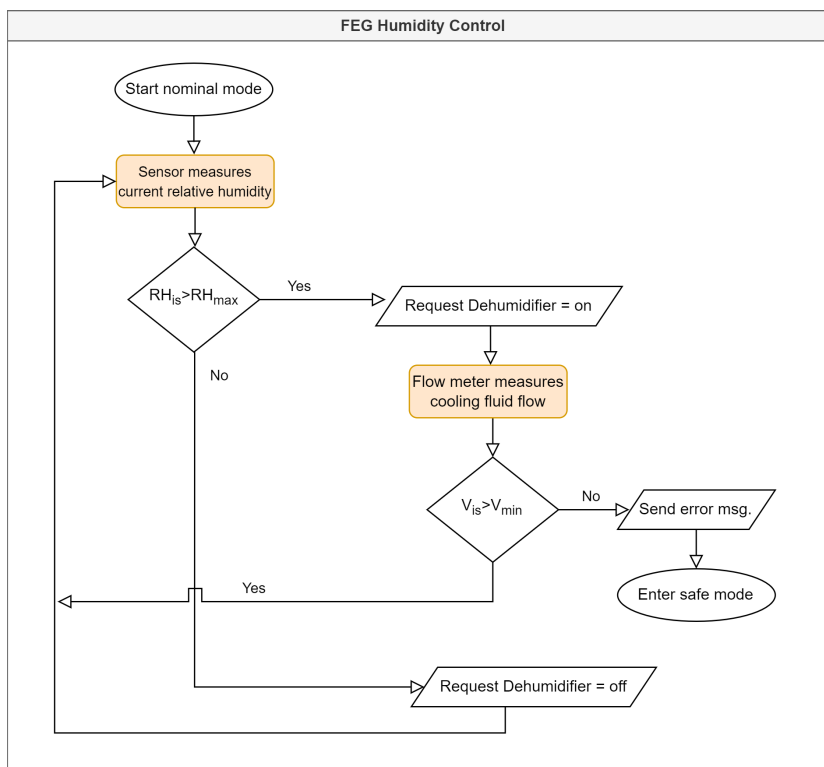


Abbildung 8.3.: Ablaufdiagramm Luftfeuchtigkeitsregelung

### **Luftgeschwindigkeitsregelung**

Für die Luftgeschwindigkeitsregelung sind zwei unterschiedliche untere Grenzwerte definiert: einer, der gilt, wenn der Entfeuchter in Betrieb ist, und ein weiterer für den Fall, dass der Entfeuchter nicht aktiv ist. Dies hat den Hintergrund, dass der Entfeuchter einen gewissen Luftstrom benötigt, um effektiv zu sein.

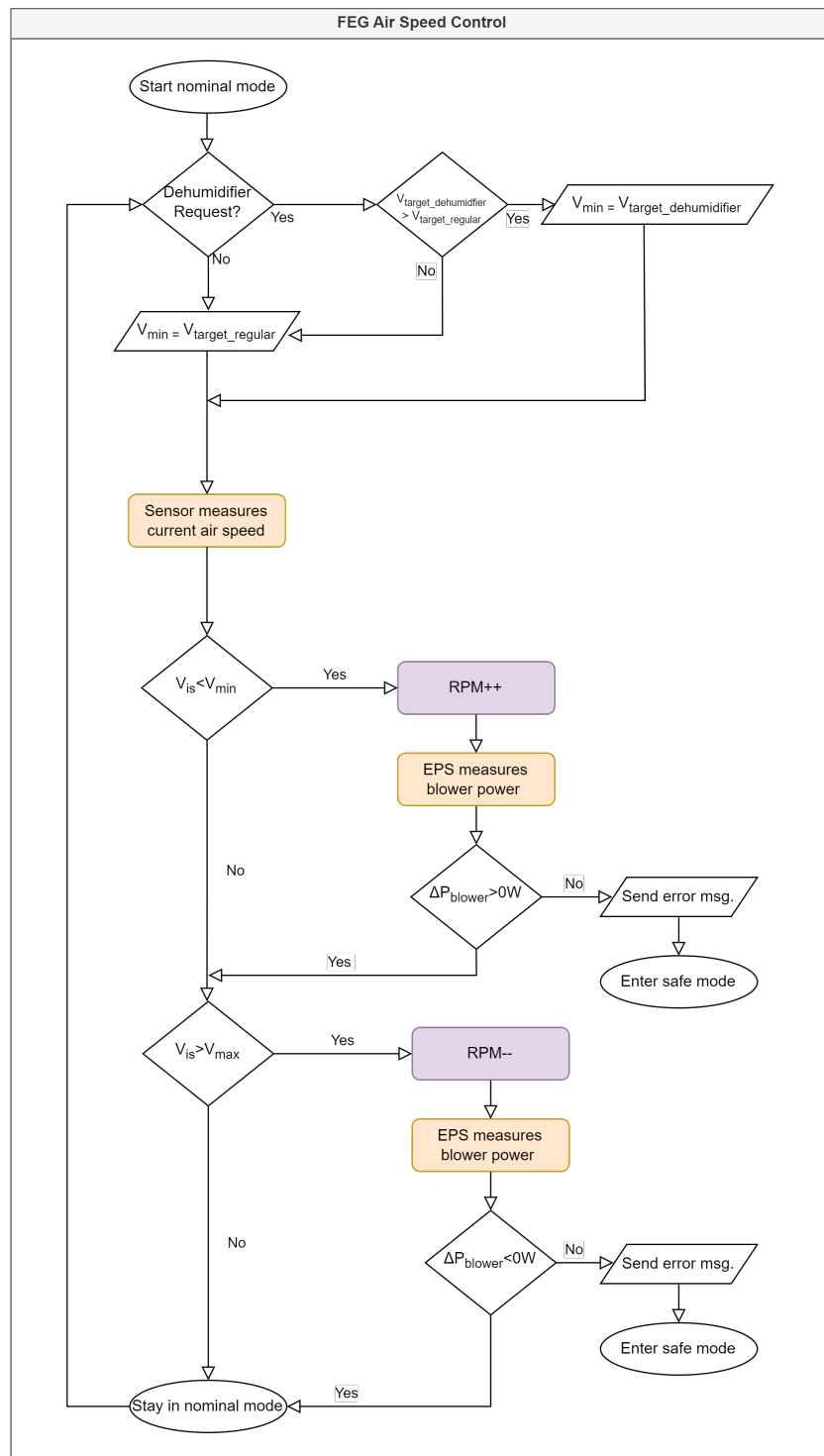
Es wird zunächst aus dem Value Store abgefragt, ob die Temperatur- oder die Luftfeuchtigkeitsregelung gerade den Entfeuchter benötigen. Wenn dieser benötigt wird, wird geprüft, ob der Grenzwert für diesen Fall höher ist als der standartmäßige. Ist dies gegeben, wird der untere Grenzwert für den Entfeuchter genutzt, ansonsten der standartmäßige.

Dann wird über die Brainboxes-Treiber-Klasse die aktuelle Luftgeschwindigkeit im Trakt gemessen. Sollte sie niedriger als der untere Grenzwert oder höher als der obere Grenzwert sein, werden 5 % von einer Variable abgezogen oder hinzugefügt. Der Wert dieser Prozent-Variable wird im Anschluss geprüft. Sollte er unter 0 sein wird er auf 0 korrigiert und sollte er über 100 sein wird er auf 100 korrigiert. Im Anschluss wird der Wert mit acht multipliziert und als Parameter in den Aufruf der Treiberklasse für die Motorsteuerung hineingegeben. Die Methode kann mit Werten von -800 bis +800 umgehen, wobei negative Werte für die umgekehrte Drehrichtung stehen, welche jedoch in dieser Regelung nicht benötigt werden. Somit wird das Kommando gesendet, um die Lüfter mit der entsprechenden Geschwindigkeit drehen zu lassen.

In Abb. 8.4 ist dieser Ablauf vereinfacht zu sehen.



## 8.1. Implementierung der Komponenten



**Abbildung 8.4.:** Ablaufdiagramm Luftgeschwindigkeitsregelung

### **CO2-Regelung**

Die CO2-Regelung enthält eine Methode zur Ermittlung der aktuellen Tageszeit. Sie unterscheidet zwischen Morgen, Tag, Nachmittag und Nacht. Je nach Tageszeit wird unterschiedlich viel CO2 eingegast. Eine vollständige CO2-Düngung ist nur während des Tages sinnvoll. Sie hat nachts keinen Einfluss, da ohne Licht keine Photosynthese stattfinden kann. Hier ergeben sich Strom- und CO2-Einsparpotenziale, da überschüssiges Kohlendioxid nicht durch die Pflanzen genutzt werden kann. Viel mehr könnte es ab einem Wert von über 1000 ppm sogar schaden. Grund hierfür ist die Wurzelatmung. Pflanzen benötigen in der Nacht zumindest im Wurzelraum Sauerstoff[30].

Am Morgen sollen die Pflanzen die Möglichkeit haben, sich langsam an die steigenden CO2-Werte anzupassen. Tagsüber wird das CO2-Düngen im vollen Ausmaß durchgeführt. Das künstliche Entfernen von CO2 ist nur für extrem hohe CO2-Werte, die 5000 ppm übersteigen, vorgesehen. Diese Werte können Menschen in der Umgebung schaden. Das Entfernen von CO2 ist nicht in täglichen Regelung vorgesehen und sollte ausschließlich in Notfällen stattfinden. Am Nachmittag wird das Zudosieren von CO2 reduziert, um bereits ein niedrigeres CO2-Level erreicht zu haben, wenn die Nacht anbricht.[31]

Aus dem Value Store werden Informationen über die Zeit des Sonnenaufgangs und Sonnenuntergangs entnommen. Hinzu kommt die Information, wie viele Stunden nach Sonnenaufgang das CO2-Düngen beginnen und wie viele Stunden vor Sonnenuntergang das CO2-Düngen enden soll.

Es wird geprüft, ob alle Stunden- und Minutenangaben in sinnvollen Bereichen liegen also zwischen 0 und 23 beziehungsweise 0 und 59. Anschließend wird überprüft, ob der Morgen vor dem Sonnenuntergang und ob der Sonnenuntergang und der Nachmittag nach dem Sonnenaufgang liegen.

Im Anschluss wird die akute Tagesphase über mehrere Vergleiche der aktuellen Systemzeit mit den oben beschriebenen Werten ermittelt.

In der Methode zur Regelung wird zunächst die gerade beschriebene Methode genutzt, um die Tageszeit zu ermitteln. Je nach Phase wird dann der untere Grenzwert aus

dem Value Store abgefragt und in einer Variable gespeichert.

Nun wird der aktuelle CO<sub>2</sub>-Wert über die Brainboxes bestimmt und mit den Grenzwerten verglichen. Sollte er niedriger als der untere CO<sub>2</sub>-Grenzwert sein, wird über den Brainboxes-Treiber das CO<sub>2</sub>-Einlass-Ventil geöffnet und für eine aus dem Value Store ermittelte Anzahl an Sekunden gewartet. Der CO<sub>2</sub>-Durchfluss wird gemessen und mit einem Referenzwert verglichen, um sicherzugehen, dass CO<sub>2</sub> eingelassen wurde. Ist der gemessene Durchfluss zu klein, wird ein Fehler generiert. Nachdem die festgelegte Zeit verstrichen ist, wird das Ventil wieder geschlossen.

Wenn sich der gemessene CO<sub>2</sub>-Wert innerhalb der Grenzen befinden, wird lediglich überprüft, ob es einen Durchfluss am CO<sub>2</sub>-Einlass-Ventil gibt. Ist dies gegeben, wird ein Fehler geworfen.

Der letzte Fall tritt ein, wenn der CO<sub>2</sub>-Wert den oberen Grenzwert überschreitet. Hier wird über den Brainboxes-Treiber die CO<sub>2</sub>-Entfernungs-Pumpe aktiviert, für eine aus dem Value Store abgefragte Zeit gewartet und diese dann deaktiviert. In dieser Zeit wird die Luft durch Atemkalk geleitet, welcher das CO<sub>2</sub> bindet und sich dabei verfärbt. Es ist vorgesehen, diese Verfärbung zu überprüfen. Die Implementierung ist jedoch kein Teil dieser Arbeit, da zunächst getestet werden muss, wie schnell sich das Substrat verfärbt. Die nötige Hardware hierfür ist noch nicht vorhanden.

Die Regelung wird im Ablaufdiagramm 8.5 gezeigt.

## 8.1. Implementierung der Komponenten

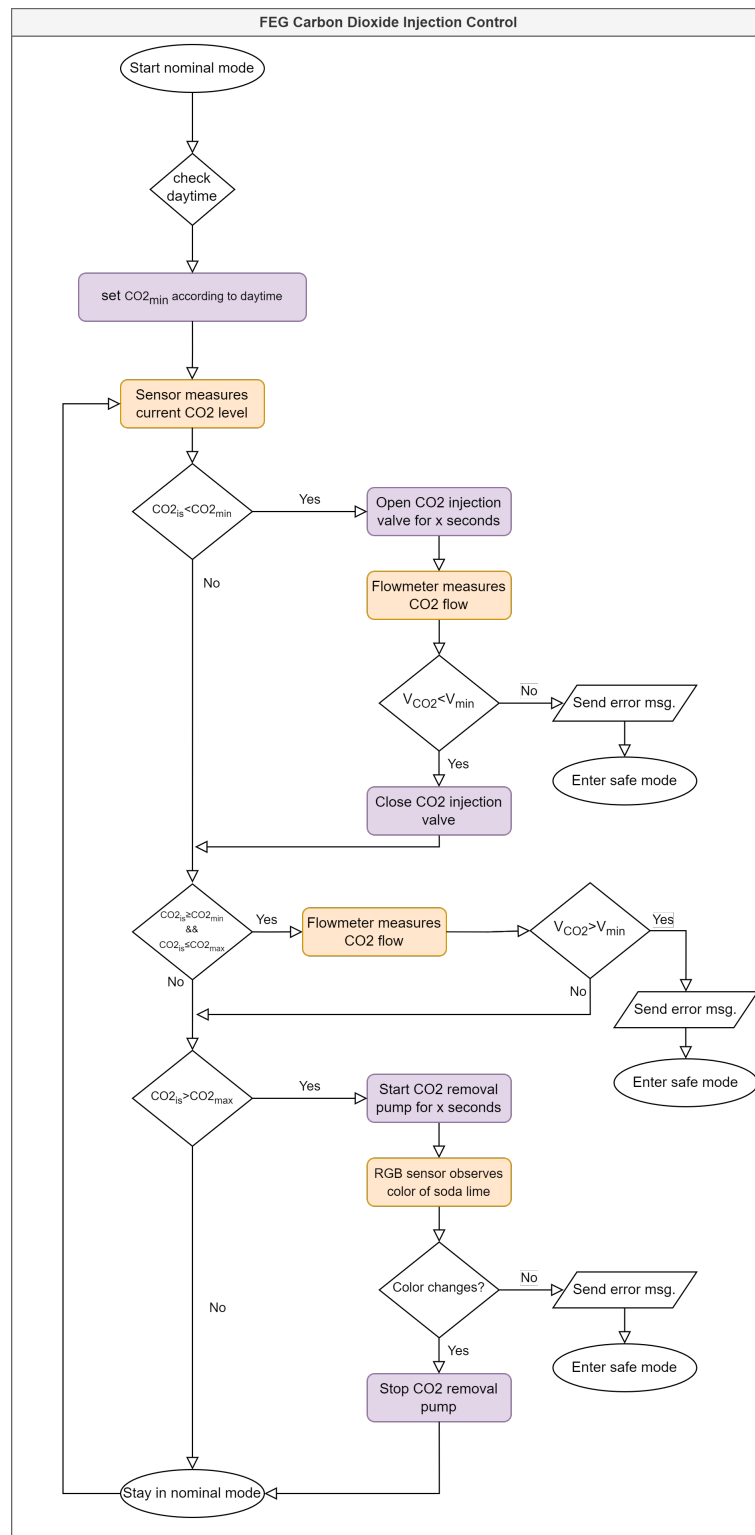


Abbildung 8.5.: Ablaufdiagramm CO<sub>2</sub>-Regelung

## 8.1. Implementierung der Komponenten

---

In Abb. 8.6 ist ein UML-Klassendiagramm der Co2Control-Klasse zu sehen. Sie hat Abhängigkeiten zu dem BrainboxesControl-Treiber, dem Value Store und einem ClockInterface, welches von der Klasse SystemClock implementiert wird.

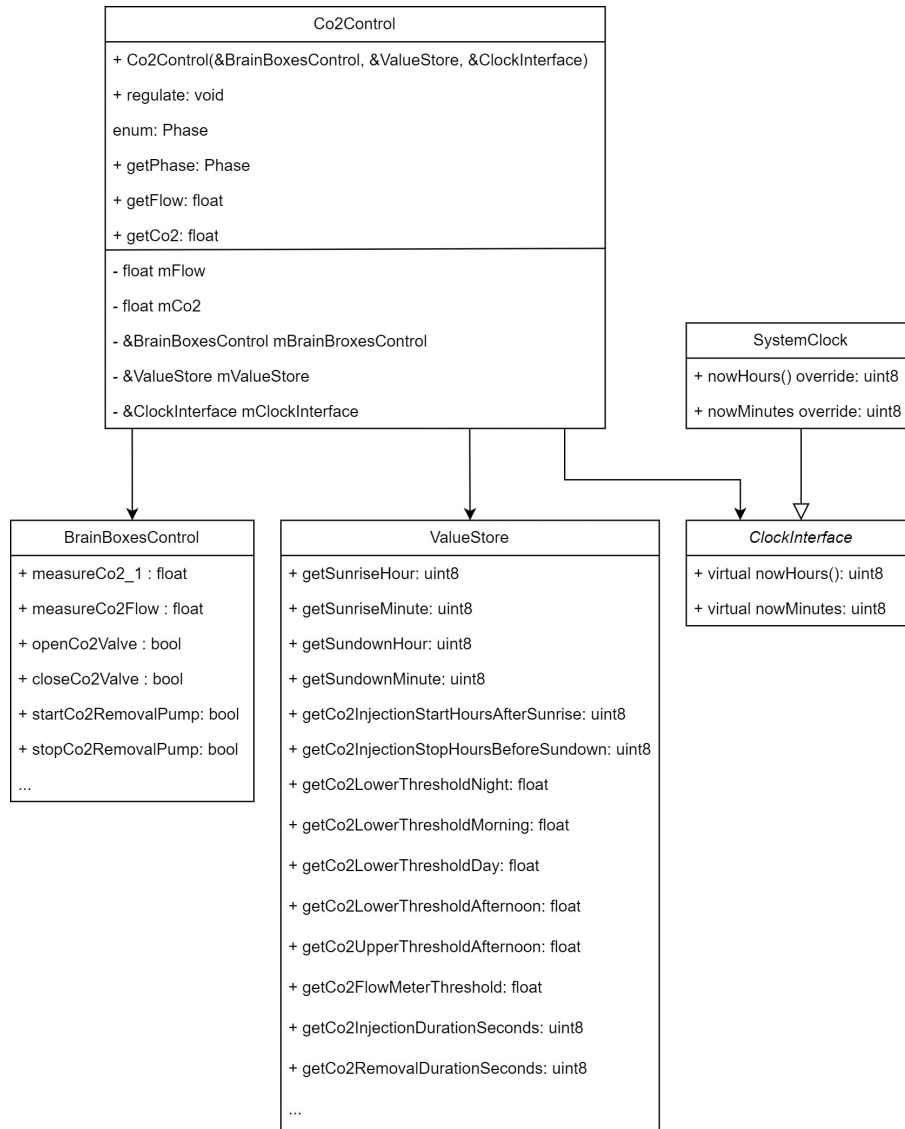


Abbildung 8.6.: UML-Klassendiagramm CO2-Regelung

## Sauerstoffregelung

Bei dieser Regelung wird über den Brainboxes Treiber der Sauerstoffgehalt der Luft gemessen. Wenn er zu hoch ist, werden über den Treiber der Motorsteuerung zwei Linearmotoren aktiviert, die eine Klappe in der Außenwand öffnen. Nach einer von der Besatzung festgelegten Zeit werden sie zurückgefahren, um die Klappe zu schließen.

In Abb. 8.7 ist dieser Ablauf dargestellt.

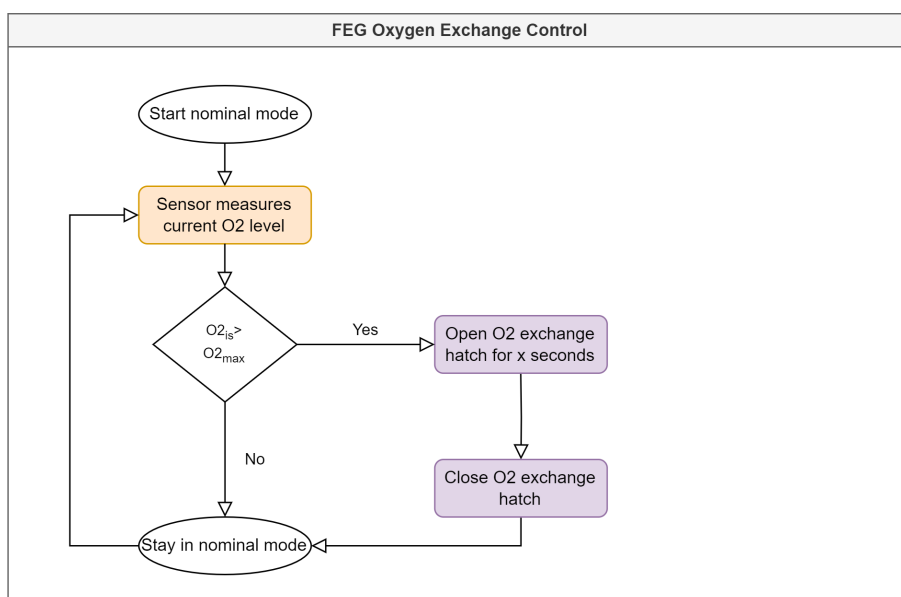


Abbildung 8.7.: Ablaufdiagramm Sauerstoffregelung

## Filterüberwachung

In dieser Kontrolle wird über den Brainboxes-Treiber an beiden Filterkomplexen die Druckdifferenz gemessen. Dann werden für jeden einzeln die folgenden vier Fälle unterschieden.

Befindet sich der gemessene Wert über dem dritten, höchsten Grenzwert, wird eine Warnung, dass ein Filterwechsel überfällig ist, in Form von Telemetrie generiert und das System in den Safemode versetzt. Der Zustand wird in einer Enumerationsvariable

## 8.1. Implementierung der Komponenten

---

abgespeichert, sodass die Housekeepingtelemetrie diesen abfragen kann.

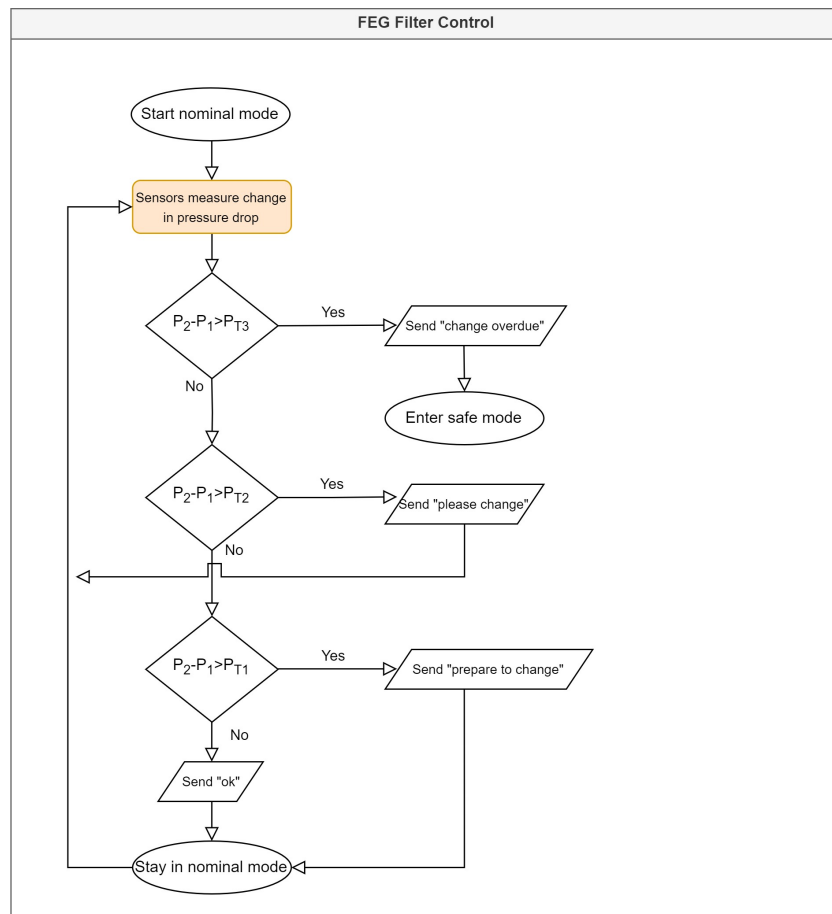
Über dem zweiten Grenzwert enthält die als Telemetrie generierte Warnung die Information, dass ein Filterwechsel stattfinden sollte, aber das System setzt seine Funktion unverändert fort. Auch hier wird der Zustand gespeichert.

Über dem ersten Grenzwert wird nur die Variable mit der Information, dass ein Filterwechsel ansteht gesetzt.

Unter dem ersten Grenzwert wird per Variable angezeigt, dass der Filter in Ordnung ist.

Im Anschluss wird jegliche Telemetrie versendet.

Das Konzept ist in Abb. 8.8 in Form eines Ablaufdiagramms zu sehen.



**Abbildung 8.8.:** Ablaufdiagramm Filterüberwachung

#### 8.1.4. Überwachung sonstiger Aktoren

Um die Funktion der sonstigen Aktoren kontrollieren zu können, wird ein Überwachungsablauf entwickelt. Dieser enthält Variablen, welche per Telekommando gesetzt werden können, die anzeigen, ob der Aktor aktiv sein soll.

#### 8.1.5. Telekommunikation

Zur Kommunikation zwischen dem **Data Handling & Control System** und dem **Atmosphere Management System** wird die OUTPOST-Implementierung des CCSDS-PUS verwendet. Sie basiert auf dem **User Datagram Protocol** (UDP). Zur Entwicklung wird das **Packet Network Documentation model** (PANDO) genutzt.

In diesem Abschnitt wird das PANDO-Tool vorgestellt und dann die Nutzung von diesem zur Implementierung der Telekommunikation des AMS beschrieben.

##### **PANDO Tool**

PANDO (**Packet Network Documentation model**) ist eine vom DLR entwickelte Software, mit deren Hilfe eine C++ **Application Programming Interface** (API) für Telekommandos und Telemetrie (siehe Kapitel 4.5) erzeugt werden kann. Es handelt sich um einen Codegenerator, der die Schnittstelle zur Telekommunikation festlegt, welche im Anschluss aber noch ausimplementiert werden sollte. Über ein Modell, dessen Beschreibungssprache XML ist, wird die Schnittstelle mit den gewünschten Informationen gefüllt. Die erzeugte Software erhält nach erfolgreicher Ausführung alle für die Kommunikation per Telekommando und Telemetrie wichtigen Klassen. Das Modell beschreibt die Struktur der Telekommandos und der Telemetriem und weist diesen eindeutige IDs zu.

Diese IDs sind von Bedeutung, da sie Gegenstand einer speziellen Missionsinformationendatenbank, welche ebenfalls von PANDO generiert wird, sind. Der Betrieb von Satelliten wird oft an externe Einrichtungen ausgelagert, welche dann über Monate oder Jahre die Kommunikation aufrecht erhalten. Die Datenbank dient als standardisiertes Referenzwerkzeug, das an diese Akteure weitergegeben werden



kann. Mit Hilfe dieser kann jede Partei ihre eigenen Technologien so einrichten, dass Telekommandos versendet und Telemetrie identifiziert werden können.

Nach der erfolgreichen Generierung kann nun in der Flugsoftware ein ebenfalls von PANDO vorbereitetes Headerfile ausimplementiert werden, um zu bestimmen, was passieren soll, wenn das entsprechende Telekommando empfangen wird. Weiterhin können definierte Telemetrien nun genutzt und versendet werden.

In der Software der Bodenstation wird das Gegenstück implementiert, welches diese in die Lage versetzt Telekommandos zu generieren und empfangende Telemetrie zu identifizieren. Die bodenstationseitige Telekommunikation soll in dieser Arbeit jedoch keine weitere Berücksichtigung finden.

### **Implementierung**

Das DHCS ist vergleichbar mit einer Bodenstation und das AMS mit dem Flugobjekt. Das AMS empfängt vom DHCS versendete Telekommandos und schickt Telemetrie an dieses zurück.

Im folgenden wird die Entwicklung der AMS-seitigen Telekommunikation beschrieben.

Das zu erstellende Modell wird in zwei XML Dateien beschrieben. Im ersten werden alle in den Telekommandos und Telemetrien benötigten Parameter mit dem zugehörigen Datentyp definiert. Dazu gehören auch Enumerationen, da einige Parameter einen spezifischen Enumerationsdatentyp erhalten sollen. Unter anderem die verschiedenen möglichen Warnungen und Fehlerfälle werden in einer Enumeration definiert. Auf diese Weise kann für Typsicherheit gesorgt werden.

Es werden zwei Telemetrien definiert. Bei der ersten handelt es sich um das Housekeeping, also die Telemetrie, die in regelmäßigen, einstellbaren Zeitabständen generiert wird und alle wichtigen Informationen über den Zustand des Systems enthält. Diese umfasst alle Sensormesswerte, aktuell eingestellte Grenzwerte, Zustände der Aktoren und Filter, Informationen zum Start der Tag- und Nachtphasen und in welchem zeitlichen Abstand die Housekeeping-Telemetrie generiert wird.

Die zweite Telemetrie hat die Aufgabe, verschiedene Warnungen und Fehler anzuzei-

gen und enthält nur einen einzigen Parameter. Dieser ist eine Enumeration, welche je nach Wert für eine von mehreren Warnungen oder Fehlern steht.

Im Anschluss werden die Telekommandos definiert, wobei zwischen zwei Arten unterschieden wird. Die erste aktiviert oder deaktiviert Elemente des Systems. Für jeden Regelkreis gibt es Telekommandos zum manuellen Ein- und Ausschalten. Dieselbe Funktionalität gibt es auch für die Aktoren. Es gilt jedoch zu beachten, dass bei Aktoren, die in einer Regelung enthalten sind, zunächst die zugehörige Regelung deaktiviert werden muss, da diese sonst den per Telekommando gesetzten Aktorenzustand direkt wieder überschreibt. Auch die Generierung der Housekeepingdaten kann ein- und ausgeschaltet werden.

Mit der zweiten Art von Telekommandos werden Systemeinstellungen gesetzt. So können etwa die Zeitabstände, mit der Housekeepingtelemetrie versendet wird, eingestellt werden. Es ist ein Telekommando enthalten, welches alle Grenzwerte gleichzeitig setzt. Es gibt auch einzelne für jede Regelung, die nur die für die Regelung spezifischen Grenzwerte setzen. Die Zeit des Sonnenauf- und untergangs kann eingestellt werden und über ein weiteres spezifisches Telekommando für die CO<sub>2</sub>-Regelung auch die Dauer des CO<sub>2</sub>-Morgens und -Nachmittags (siehe Kapitel 8.1.3) und die Dauer des Eingas- oder Filtervorgangs.

Zuletzt gibt es ein Kommando, das einen Moduswechsel hervorruft (siehe 6.2.3). Es ist eine Kombination der beiden oben genannten Kategorien, da ein Moduswechsel sowohl Regelungen aktiviert oder deaktiviert als auch eine Einstellung des Systems ändert.

Im zweiten Schritt wird ein XML File für das Mapping erstellt. Hier wird allen oben aufgezählten Elementen eine eindeutige ID zugewiesen.

Danach werden die Quellcode-Dateien aus dem Model generiert.

Dabei wird auch eine Template-Header-Datei miterzeugt, deren Methoden im Anschluss implementiert werden, um somit zu bestimmen, was beim Empfang der einzelnen Telekommandos geschehen soll. Wenn Kommandos empfangen werden, wird auf Methoden der Regelungen, Hardware Treiber oder den Value Store zugegriffen, um die gewünschte Aktion auszulösen.

Auch die Implementierung der Telemetrie wird durch PANDO vereinfacht. Bei Bedarf kann für jede Telemetrie eine von PANDO erzeugte Klasse instanziiert und die enthaltenen Methoden genutzt werden, um die Telemetrieparameter mit den gewünschten Werten zu füllen. Im Anschluss wird die Telemetrie über eine in OUTPOST existierende Funktion versendet.

## 8.2. Fehlerverhalten

Beim Versenden von Telekommandos, die neue Systemparameter setzen, findet die erste Plausibilitätsprüfung der Daten im Frontend statt, also der grafischen Oberfläche zum Versenden der Telekommandos, mit welcher die Besatzung interagiert (siehe Kapitel 9.2.2).

Zur zusätzlichen Absicherung prüfen alle Regelungen ihre benötigten Parameter erneut und generieren Fehler falls erforderlich. Ein weiterer möglicher Fehlerfall ist, dass die Überprüfung der Funktion eines Aktors fehlschlägt. Auch in diesem Fall werden von der Regelung Fehler generiert. Je nach Schwere wird nur eine Warnung als Telemetrie versendet oder das System in den "SAFE" Mode versetzt. Warnungen in Form von Telemetrie werden von den Regelungen, in denen die Fehler auftreten, direkt versendet.

Da es eine Besatzung gibt, die das Gesamtsystem überwacht, ist es wichtig, diese durch Telemetrie über aufgetretene Fehler in Kenntnis zu setzen. So kann sie entscheiden, wie weiterhin verfahren werden soll, der Fehlerursache auf den Grund gehen und gegebenenfalls Komponenten reparieren oder austauschen. Wenn eine Wartung oder eine Reparatur geschehen ist, kann die Besatzung das System wieder in seinen ursprünglichen Zustand versetzen. Die generierten Warnungen werden vom DHCS in einer Datenbank gespeichert, um sie so für wissenschaftliche Zwecke abrufbar zu machen.

Im Rahmen dieser Arbeit werden mögliche Fehlerquellen im Code vorgemerkt und gesammelt. Im weiteren Verlauf des EDEN LUNA Projekts wird die schwere der unterschiedlichen Fehler festgelegt und die Fehlerbehandlung implementiert. Dies ist nicht Teil dieser Arbeit.

# 9. Testen

An dieser Stelle werden die durchgeführten Tests vorgestellt und die erhaltenen Ergebnisse ausgewertet. Zunächst sind die Unittests der Regelungen und Telekommandos sowie die Analyse ihrer Testabdeckung beschrieben. Danach folgt die Betrachtung der hardwarenahen Tests, einschließlich der Treibertests und abschließend ein Systemtest.

## 9.1. Unittests

Im Folgenden werden die Tests vorgestellt, welche keine Hardware-Komponenten benötigen. Die sogenannten Unittests überprüfen jeweils die Funktionalität einer implementierten Klasse. Sie werden mit Hilfe des Google-Test-Frameworks durchgeführt.

### 9.1.1. Mockups

Um die Regelungen testbar zu machen, werden Mockups<sup>1</sup> für die Treiber der Brainboxes und der Motorsteuerung erstellt. Diese Mockups ermöglichen die Simulation verschiedener Rückgabewerte der Sensoren und die Überprüfung der erwarteten Aufrufe der Aktoren. Zudem wird ein Mockup für die Uhr implementiert, welches das Einstellen von Testuhrzeiten ermöglicht, um so zeitabhängige Prozesse zu simulieren. Auch für die Klasse, welche die Telemetrie versendet, wird ein Mockup erstellt, um zu überprüfen, dass der Aufruf tatsächlich stattfindet.

---

<sup>1</sup>Platzhalterobjekt, das das Verhalten einer echten Komponente simuliert

## 9.1.2. Temperaturregelung und Luftfeuchtigkeitsregelung

Die Tests für Temperatur- und Luftfeuchtigkeitsregelung versetzen die Klassen, durch Festlegen der Grenzwerte im Value Store und durch Simulation der Messwerte, in die verschiedenen möglichen Zustände. Danach wird überprüft, ob die Aktoren wie erwartet aufgerufen werden.

Zusätzlich werden Grenzfälle getestet, bei denen der Messwert genau mit dem Grenzwert übereinstimmt.

In Abb. 9.1 sind die Ausgaben des Temperaturtest dargestellt. Die Luftfeuchtigkeitsregelungen entsprechend in Abb. 9.2.

```
[-----] 5 tests from TemperatureControlTest
[ RUN   ] TemperatureControlTest.TemperatureControl__Regulate__TempTooLow__ShouldTurnOnHeaterSendDehumidifierRelease
[ OK    ] TemperatureControlTest.TemperatureControl__Regulate__TempTooLow__ShouldTurnOnHeaterSendDehumidifierRelease (0 ms)
[ RUN   ] TemperatureControlTest.TemperatureControl__Regulate__TempOkay__ShouldTurnOffHeaterSendDehumidifierRelease
[ OK    ] TemperatureControlTest.TemperatureControl__Regulate__TempOkay__ShouldTurnOffHeaterSendDehumidifierRelease (0 ms)
[ RUN   ] TemperatureControlTest.TemperatureControl__Regulate__TempTooHigh__ShouldTurnOffHeaterSendDehumidifierRequest
[ OK    ] TemperatureControlTest.TemperatureControl__Regulate__TempTooHigh__ShouldTurnOffHeaterSendDehumidifierRequest (500 ms)
[ RUN   ] TemperatureControlTest.TemperatureControl__Regulate__EdgeCaseLowerThreshold__ShouldTurnOffHeaterSendDehumidifierRelease
[ OK    ] TemperatureControlTest.TemperatureControl__Regulate__EdgeCaseLowerThreshold__ShouldTurnOffHeaterSendDehumidifierRelease (0 ms)
[ RUN   ] TemperatureControlTest.TemperatureControl__Regulate__EdgeCaseUpperThreshold__ShouldTurnOffHeaterSendDehumidifierRequest
[ OK    ] TemperatureControlTest.TemperatureControl__Regulate__EdgeCaseUpperThreshold__ShouldTurnOffHeaterSendDehumidifierRequest (0 ms)
[-----] 5 tests from TemperatureControlTest (502 ms total)
```

Abbildung 9.1.: Konsolen Ausgabe der Tests der Temperaturregelung

```
[-----] 3 tests from HumidityControlTest
[ RUN   ] HumidityControlTest.HumidityControl__Regulate__HumTooHigh__ShouldSendDehumidifierRequest
[ OK    ] HumidityControlTest.HumidityControl__Regulate__HumTooHigh__ShouldSendDehumidifierRequest (500 ms)
[ RUN   ] HumidityControlTest.HumidityControl__Regulate__HumOkay__ShouldSendDehumidifierRelease
[ OK    ] HumidityControlTest.HumidityControl__Regulate__HumOkay__ShouldSendDehumidifierRelease (0 ms)
[ RUN   ] HumidityControlTest.HumidityControl__Regulate__HumEdge__ShouldSendDehumidifierRequest
[ OK    ] HumidityControlTest.HumidityControl__Regulate__HumEdge__ShouldSendDehumidifierRequest (501 ms)
[-----] 3 tests from HumidityControlTest (1006 ms total)
```

Abbildung 9.2.: Konsolen Ausgabe der Tests der Luftfeuchtigkeitsregelung

## 9.1.3. Luftgeschwindigkeitsregelung

Die Tests zur Regelung der Luftgeschwindigkeit laufen ähnlich ab. Es muss darüber hinaus allerdings berücksichtigt werden, ob der Entfeuchter gerade im Einsatz ist. Daher wird die Durchführung der Tests zur Überprüfung der unterschiedlichen Grenzwerte davon abhängig gemacht. Die durchgeführten Tests sind in Abb. 9.3 zu sehen.

## 9.1. Unittests

---

```
[-----] 17 tests from AirSpeedControlTest
[ RUN ] AirSpeedControlTest.AirSpeedControl__CheckDehumidifierRunning__BothTrue__ShouldBeTrue
[ OK ] AirSpeedControlTest.AirSpeedControl__CheckDehumidifierRunning__BothTrue__ShouldBeTrue (0 ms)
[ RUN ] AirSpeedControlTest.AirSpeedControl__CheckDehumidifierRunning__OneTrue1__ShouldBeTrue
[ OK ] AirSpeedControlTest.AirSpeedControl__CheckDehumidifierRunning__OneTrue1__ShouldBeTrue (0 ms)
[ RUN ] AirSpeedControlTest.AirSpeedControl__CheckDehumidifierRunning__OneTrue2__ShouldBeTrue
[ OK ] AirSpeedControlTest.AirSpeedControl__CheckDehumidifierRunning__OneTrue2__ShouldBeTrue (0 ms)
[ RUN ] AirSpeedControlTest.AirSpeedControl__CheckDehumidifierRunning__BothFalse__ShouldBeFalse
[ OK ] AirSpeedControlTest.AirSpeedControl__CheckDehumidifierRunning__BothFalse__ShouldBeFalse (0 ms)
[ RUN ] AirSpeedControlTest.AirSpeedControl__Regulate__DehumidifierSpeedTooLow__ShouldIncreaseSpeed
adding 5 percent to fans
[ OK ] AirSpeedControlTest.AirSpeedControl__Regulate__DehumidifierSpeedTooLow__ShouldIncreaseSpeed (1 ms)
[ RUN ] AirSpeedControlTest.AirSpeedControl__Regulate__NoDehumidifierSpeedTooLow__ShouldIncreaseSpeed
adding 5 percent to fans
[ OK ] AirSpeedControlTest.AirSpeedControl__Regulate__NoDehumidifierSpeedTooLow__ShouldIncreaseSpeed (0 ms)
[ RUN ] AirSpeedControlTest.AirSpeedControl__Regulate__DehumidifierSpeedOkay__ShouldNotIncreaseSpeed
[ OK ] AirSpeedControlTest.AirSpeedControl__Regulate__DehumidifierSpeedOkay__ShouldNotIncreaseSpeed (0 ms)
[ RUN ] AirSpeedControlTest.AirSpeedControl__Regulate__NoDehumidifierSpeedOkay__ShouldNotIncreaseSpeed
[ OK ] AirSpeedControlTest.AirSpeedControl__Regulate__NoDehumidifierSpeedOkay__ShouldNotIncreaseSpeed (0 ms)
[ RUN ] AirSpeedControlTest.AirSpeedControl__Regulate__DehumidifierSpeedTooHigh__ShouldDecreaseSpeed
subtracting 5 percent from fans
[ OK ] AirSpeedControlTest.AirSpeedControl__Regulate__DehumidifierSpeedTooHigh__ShouldDecreaseSpeed (0 ms)
[ RUN ] AirSpeedControlTest.AirSpeedControl__Regulate__NoDehumidifierSpeedTooHigh__ShouldDecreaseSpeed
subtracting 5 percent from fans
[ OK ] AirSpeedControlTest.AirSpeedControl__Regulate__NoDehumidifierSpeedTooHigh__ShouldDecreaseSpeed (0 ms)
[ RUN ] AirSpeedControlTest.AirSpeedControl__Regulate__ThresholdsWrongSpeedTooLow__ShouldIncreaseSpeed
adding 5 percent to fans
[ OK ] AirSpeedControlTest.AirSpeedControl__Regulate__ThresholdsWrongSpeedTooLow__ShouldIncreaseSpeed (0 ms)
[ RUN ] AirSpeedControlTest.AirSpeedControl__Regulate__NoDehumidifierLowerThreshold__ShouldNotIncreaseSpeed
[ OK ] AirSpeedControlTest.AirSpeedControl__Regulate__NoDehumidifierLowerThreshold__ShouldNotIncreaseSpeed (0 ms)
[ RUN ] AirSpeedControlTest.AirSpeedControl__Regulate__NoDehumidifierUpperThreshold__ShouldNotIncreaseSpeed
[ OK ] AirSpeedControlTest.AirSpeedControl__Regulate__NoDehumidifierUpperThreshold__ShouldNotIncreaseSpeed (0 ms)
[ RUN ] AirSpeedControlTest.AirSpeedControl__Regulate__DehumidifierLowerThreshold__ShouldNotIncreaseSpeed
[ OK ] AirSpeedControlTest.AirSpeedControl__Regulate__DehumidifierLowerThreshold__ShouldNotIncreaseSpeed (0 ms)
[ RUN ] AirSpeedControlTest.AirSpeedControl__Regulate__DehumidifierUpperThreshold__ShouldNotIncreaseSpeed
[ OK ] AirSpeedControlTest.AirSpeedControl__Regulate__DehumidifierUpperThreshold__ShouldNotIncreaseSpeed (0 ms)
[ RUN ] AirSpeedControlTest.AirSpeedControl__Regulate__DehumidifierLowerThreshold1__ShouldIncreaseSpeed
adding 5 percent to fans
[ OK ] AirSpeedControlTest.AirSpeedControl__Regulate__DehumidifierLowerThreshold1__ShouldIncreaseSpeed (0 ms)
[ RUN ] AirSpeedControlTest.AirSpeedControl__Regulate__DehumidifierUpperThreshold1__ShouldIncreaseSpeed
subtracting 5 percent from fans
[ OK ] AirSpeedControlTest.AirSpeedControl__Regulate__DehumidifierUpperThreshold1__ShouldIncreaseSpeed (0 ms)
[-----] 17 tests from AirSpeedControlTest (25 ms total)
```

Abbildung 9.3.: Konsolen Ausgabe der Tests der Luftgeschwindigkeitsregelung

### 9.1.4. CO2-Regelung

Im folgenden Abschnitt wird der Test der CO2-Regelung beschrieben.

Zunächst wird die Methode zur Ermittlung der aktuellen Tageszeit getestet. Wie bereits in Kapitel 8.1.3 erwähnt, wird hierbei zwischen Morgen, Tag, Nachmittag und Nacht unterschieden.

Die Funktionalität wird getestet, indem für jede Tageszeit drei Testfälle definiert werden. Es werden jeweils die Zeiten für Sonnenaufgang, Ende des Morgens, Anfang des Nachmittags und Sonnenuntergang eingestellt und die "aktuelle" Zeit gesetzt. Im Anschluss wird die von der Methode ermittelte aktuelle Tageszeit mit der erwarteten abgeglichen.

Diese Prozedur wird für Randfälle wiederholt. Hier wird getestet, welche Tageszeit

## 9.1. Unittests

---

ausgegeben wird, wenn die Uhr genau an den Übergangszeiten steht, etwa beim Beginn, einer Minute vor dem Ende oder unmittelbar nach dem Ende des Morgens. Bei Fehleingaben, wie etwa Stundenzahlen, die kleiner als 0 oder größer als 23 sind, soll ebenfalls ein sinnvoller Wert, mit dem die CO<sub>2</sub>-Regelung arbeiten kann, zurückgeliefert werden. Dies ist gleichfalls Bestandteil der Tests.

Da die Tests bis auf unterschiedliche Werte komplett identisch sind, werden Parametertests<sup>2</sup> genutzt. Diese bestätigen, dass die Methode ihre beabsichtigte Funktionalität erfüllt.

Des Weiteren wird die Funktionalität der CO<sub>2</sub>-Regelung getestet. Es werden verschiedene Tageszeiten in Kombination mit verschiedenen CO<sub>2</sub>-Gehalten kombiniert und das erwartete Verhalten der Aktoren überwacht. Auch hier werden Randfälle, in dem der Messwert genau einem Grenzwert entspricht, überprüft.

In Abb. 9.4 ist ein Ausschnitt der Konsolenausgabe der Tests, welche die Tagesphase betreffen, zu erkennen. In Abb. 9.5 sind die entsprechenden Ausgaben für die Regelungstests zu sehen.

```
[-----] 49 tests from Co2ControlPhaseTestParameters/Co2ControlPhaseTest
[ RUN    ] Co2ControlPhaseTestParameters/Co2ControlPhaseTest.Co2Control__Phase/shouldBeMorning1
[ RUN    ] Co2ControlPhaseTestParameters/Co2ControlPhaseTest.Co2Control__Phase/shouldBeMorning2
[ RUN    ] Co2ControlPhaseTestParameters/Co2ControlPhaseTest.Co2Control__Phase/shouldBeMorning3
[ RUN    ] Co2ControlPhaseTestParameters/Co2ControlPhaseTest.Co2Control__Phase/shouldBeMorning3 (0 ms)
[ RUN    ] Co2ControlPhaseTestParameters/Co2ControlPhaseTest.Co2Control__Phase/shouldBeDay1
[ RUN    ] Co2ControlPhaseTestParameters/Co2ControlPhaseTest.Co2Control__Phase/shouldBeDay2
[ RUN    ] Co2ControlPhaseTestParameters/Co2ControlPhaseTest.Co2Control__Phase/shouldBeDay3
[ RUN    ] Co2ControlPhaseTestParameters/Co2ControlPhaseTest.Co2Control__Phase/shouldBeDay3 (0 ms)
[ RUN    ] Co2ControlPhaseTestParameters/Co2ControlPhaseTest.Co2Control__Phase/shouldBeAfternoon1
[ RUN    ] Co2ControlPhaseTestParameters/Co2ControlPhaseTest.Co2Control__Phase/shouldBeAfternoon2
[ RUN    ] Co2ControlPhaseTestParameters/Co2ControlPhaseTest.Co2Control__Phase/shouldBeAfternoon2 (0 ms)
```

**Abbildung 9.4.:** Ausschnitt der Konsolen Ausgabe der Tests der CO<sub>2</sub>-Phasenbestimmung

---

<sup>2</sup>dieselbe Testlogik wird mit verschiedenen Eingabewerten ausgeführt

## 9.1. Unittests

---

```
[-----] 20 tests from Co2ControlRegulateTest
[ RUN ] Co2ControlRegulateTest.Co2Control_Regulate_MorningCo2TooLow_ShouldOpenValve
[ OK ] Co2ControlRegulateTest.Co2Control_Regulate_MorningCo2TooLow_ShouldOpenValve (1000 ms)
[ RUN ] Co2ControlRegulateTest.Co2Control_Regulate_MorningCo2Okay_ShouldDoNothing
[ OK ] Co2ControlRegulateTest.Co2Control_Regulate_MorningCo2Okay_ShouldDoNothing (0 ms)
[ RUN ] Co2ControlRegulateTest.Co2Control_Regulate_MorningCo2TooHigh_ShouldStartRemovalPump
[ OK ] Co2ControlRegulateTest.Co2Control_Regulate_MorningCo2TooHigh_ShouldStartRemovalPump (1001 ms)
[ RUN ] Co2ControlRegulateTest.Co2Control_Regulate_DayCo2TooLow_ShouldOpenValve
[ OK ] Co2ControlRegulateTest.Co2Control_Regulate_DayCo2TooLow_ShouldOpenValve (1000 ms)
[ RUN ] Co2ControlRegulateTest.Co2Control_Regulate_DayCo2Okay_ShouldDoNothing
[ OK ] Co2ControlRegulateTest.Co2Control_Regulate_DayCo2Okay_ShouldDoNothing (0 ms)
[ RUN ] Co2ControlRegulateTest.Co2Control_Regulate_DayCo2TooHigh_ShouldStartRemovalPump
[ OK ] Co2ControlRegulateTest.Co2Control_Regulate_DayCo2TooHigh_ShouldStartRemovalPump (1000 ms)
[ RUN ] Co2ControlRegulateTest.Co2Control_Regulate_AfternoonCo2TooLow_ShouldOpenValve
[ OK ] Co2ControlRegulateTest.Co2Control_Regulate_AfternoonCo2TooLow_ShouldOpenValve (1000 ms)
[ RUN ] Co2ControlRegulateTest.Co2Control_Regulate_AfternoonCo2Okay_ShouldDoNothing
[ OK ] Co2ControlRegulateTest.Co2Control_Regulate_AfternoonCo2Okay_ShouldDoNothing (0 ms)
[ RUN ] Co2ControlRegulateTest.Co2Control_Regulate_AfternoonCo2TooHigh
[ OK ] Co2ControlRegulateTest.Co2Control_Regulate_AfternoonCo2TooHigh (1000 ms)
[ RUN ] Co2ControlRegulateTest.Co2Control_Regulate_NightCo2TooLow_ShouldOpenValve
[ OK ] Co2ControlRegulateTest.Co2Control_Regulate_NightCo2TooLow_ShouldOpenValve (1001 ms)
[ RUN ] Co2ControlRegulateTest.Co2Control_Regulate_NightCo2Okay_ShouldDoNothing
[ OK ] Co2ControlRegulateTest.Co2Control_Regulate_NightCo2Okay_ShouldDoNothing (0 ms)
[ RUN ] Co2ControlRegulateTest.Co2Control_Regulate_NightCo2TooHigh
[ OK ] Co2ControlRegulateTest.Co2Control_Regulate_NightCo2TooHigh (1001 ms)
[ RUN ] Co2ControlRegulateTest.Co2Control_Regulate_EdgeMorningCo2LowerThreshold_ShouldDoNothing
[ OK ] Co2ControlRegulateTest.Co2Control_Regulate_EdgeMorningCo2LowerThreshold_ShouldDoNothing (0 ms)
[ RUN ] Co2ControlRegulateTest.Co2Control_Regulate_EdgeMorningCo2UpperThreshold_ShouldDoNothing
[ OK ] Co2ControlRegulateTest.Co2Control_Regulate_EdgeMorningCo2UpperThreshold_ShouldDoNothing (0 ms)
[ RUN ] Co2ControlRegulateTest.Co2Control_Regulate_EdgeDayCo2LowerThreshold_ShouldDoNothing
[ OK ] Co2ControlRegulateTest.Co2Control_Regulate_EdgeDayCo2LowerThreshold_ShouldDoNothing (0 ms)
[ RUN ] Co2ControlRegulateTest.Co2Control_Regulate_EdgeDayCo2UpperThreshold_ShouldDoNothing
[ OK ] Co2ControlRegulateTest.Co2Control_Regulate_EdgeDayCo2UpperThreshold_ShouldDoNothing (0 ms)
[ RUN ] Co2ControlRegulateTest.Co2Control_Regulate_EdgeAfternoonCo2LowerThreshold_ShouldDoNothing
[ OK ] Co2ControlRegulateTest.Co2Control_Regulate_EdgeAfternoonCo2LowerThreshold_ShouldDoNothing (0 ms)
[ RUN ] Co2ControlRegulateTest.Co2Control_Regulate_EdgeAfternoonCo2UpperThreshold_ShouldDoNothing
[ OK ] Co2ControlRegulateTest.Co2Control_Regulate_EdgeAfternoonCo2UpperThreshold_ShouldDoNothing (0 ms)
[ RUN ] Co2ControlRegulateTest.Co2Control_Regulate_EdgeNightCo2UpperThreshold_ShouldDoNothing
[ OK ] Co2ControlRegulateTest.Co2Control_Regulate_EdgeNightCo2UpperThreshold_ShouldDoNothing (0 ms)
[ RUN ] Co2ControlRegulateTest.Co2Control_Regulate_EdgeNightCo2LowerThreshold_ShouldDoNothing
[ OK ] Co2ControlRegulateTest.Co2Control_Regulate_EdgeNightCo2LowerThreshold_ShouldDoNothing (0 ms)
[-----] 20 tests from Co2ControlRegulateTest (8019 ms total)
```

Abbildung 9.5.: Konsolen Ausgabe der Tests der CO2-Regelung

### 9.1.5. Sauerstoffregelung und Filterüberwachung

Die Sauerstoffregelungs- und Filterüberwachungstests laufen ähnlich ab, wie die bereits beschriebenen. Bei den Filtertests wird jeder Filter einzeln geprüft, um sicherzustellen, dass die Regelungen für beide Filter korrekt funktionieren.

Die Tests der Filterüberwachung sind in Abb. 9.6 und die der Sauerstoffregelung in Abb. 9.7 dargestellt.



## 9.1. Unittests

---

```
[-----] 22 tests from FilterControlTest
[ RUN    ] FilterControlTest.FilterControl__Regulate__HepaOkay
[ OK     ] FilterControlTest.FilterControl__Regulate__HepaOkay (0 ms)
[ RUN    ] FilterControlTest.FilterControl__Regulate__PostOkay
[ OK     ] FilterControlTest.FilterControl__Regulate__PostOkay (0 ms)
[ RUN    ] FilterControlTest.FilterControl__Regulate__HepaPrepareToChange
[ OK     ] FilterControlTest.FilterControl__Regulate__HepaPrepareToChange (0 ms)
[ RUN    ] FilterControlTest.FilterControl__Regulate__PostPrepareToChange
[ OK     ] FilterControlTest.FilterControl__Regulate__PostPrepareToChange (0 ms)
[ RUN    ] FilterControlTest.FilterControl__Regulate__HepaPleaseChange
[ OK     ] FilterControlTest.FilterControl__Regulate__HepaPleaseChange (0 ms)
[ RUN    ] FilterControlTest.FilterControl__Regulate__PostPleaseChange
[ OK     ] FilterControlTest.FilterControl__Regulate__PostPleaseChange (0 ms)
[ RUN    ] FilterControlTest.FilterControl__Regulate__HepaChangeOverdue
[ OK     ] FilterControlTest.FilterControl__Regulate__HepaChangeOverdue (0 ms)
[ RUN    ] FilterControlTest.FilterControl__Regulate__PostChangeOverdue
[ OK     ] FilterControlTest.FilterControl__Regulate__PostChangeOverdue (0 ms)
[ RUN    ] FilterControlTest.FilterControl__Regulate__HepaOkayEdge
[ OK     ] FilterControlTest.FilterControl__Regulate__HepaOkayEdge (0 ms)
[ RUN    ] FilterControlTest.FilterControl__Regulate__PostOkayEdge
[ OK     ] FilterControlTest.FilterControl__Regulate__PostOkayEdge (0 ms)
[ RUN    ] FilterControlTest.FilterControl__Regulate__HepaPrepareToChangeEdge1
[ OK     ] FilterControlTest.FilterControl__Regulate__HepaPrepareToChangeEdge1 (0 ms)
[ RUN    ] FilterControlTest.FilterControl__Regulate__HepaPrepareToChangeEdge2
[ OK     ] FilterControlTest.FilterControl__Regulate__HepaPrepareToChangeEdge2 (0 ms)
[ RUN    ] FilterControlTest.FilterControl__Regulate__PostPrepareToChangeEdge1
[ OK     ] FilterControlTest.FilterControl__Regulate__PostPrepareToChangeEdge1 (0 ms)
[ RUN    ] FilterControlTest.FilterControl__Regulate__PostPrepareToChangeEdge2
[ OK     ] FilterControlTest.FilterControl__Regulate__PostPrepareToChangeEdge2 (0 ms)
[ RUN    ] FilterControlTest.FilterControl__Regulate__HepaPleaseChangeEdge1
[ OK     ] FilterControlTest.FilterControl__Regulate__HepaPleaseChangeEdge1 (0 ms)
[ RUN    ] FilterControlTest.FilterControl__Regulate__HepaPleaseChangeEdge2
[ OK     ] FilterControlTest.FilterControl__Regulate__HepaPleaseChangeEdge2 (0 ms)
[ RUN    ] FilterControlTest.FilterControl__Regulate__PostPleaseChangeEdge1
[ OK     ] FilterControlTest.FilterControl__Regulate__PostPleaseChangeEdge1 (0 ms)
[ RUN    ] FilterControlTest.FilterControl__Regulate__PostPleaseChangeEdge2
[ OK     ] FilterControlTest.FilterControl__Regulate__PostPleaseChangeEdge2 (0 ms)
[ RUN    ] FilterControlTest.FilterControl__Regulate__HepaChangeOverdueEdge1
[ OK     ] FilterControlTest.FilterControl__Regulate__HepaChangeOverdueEdge1 (0 ms)
[ RUN    ] FilterControlTest.FilterControl__Regulate__PostChangeOverdueEdge1
[ OK     ] FilterControlTest.FilterControl__Regulate__PostChangeOverdueEdge1 (0 ms)
[ RUN    ] FilterControlTest.FilterControl__Regulate__HepaNegativeMeasurementShouldBeError
[ OK     ] FilterControlTest.FilterControl__Regulate__HepaNegativeMeasurementShouldBeError (0 ms)
[ RUN    ] FilterControlTest.FilterControl__Regulate__PostNegativeMeasurementShouldBeError
[ OK     ] FilterControlTest.FilterControl__Regulate__PostNegativeMeasurementShouldBeError (0 ms)
[-----] 22 tests from FilterControlTest (8 ms total)
```

Abbildung 9.6.: Konsolen Ausgabe der Tests der Filterüberwachung

```
[-----] 3 tests from O2ControlTest
[ RUN    ] O2ControlTest.O2Control__Regulate__O2TooHigh__ShouldStartMotors
[ OK     ] O2ControlTest.O2Control__Regulate__O2TooHigh__ShouldStartMotors (10001 ms)
[ RUN    ] O2ControlTest.O2Control__Regulate__O2Okay__ShouldDoNothing
[ OK     ] O2ControlTest.O2Control__Regulate__O2Okay__ShouldDoNothing (0 ms)
[ RUN    ] O2ControlTest.O2Control__Regulate__O2Edge__ShouldDoNothing
[ OK     ] O2ControlTest.O2Control__Regulate__O2Edge__ShouldDoNothing (0 ms)
[-----] 3 tests from O2ControlTest (10001 ms total)
```

Abbildung 9.7.: Konsolen Ausgabe der Tests der Sauerstoffregelung

### 9.1.6. Überwachung sonstiger Aktoren

Zur Validierung der Funktionalität der Überwachung der verbleibenden Aktoren wird in allen möglichen Kombinationen von Ein- und Ausschaltzuständen geprüft, ob die korrekten Aufrufe des Brainboxes-Treibers erfolgen. Die Ergebnisse der Tests sind in 9.8 zu sehen.

```
[-----] 8 tests from RemainingActorsControlTest
[ RUN   ] RemainingActorsControlTest.RemainingActorsControl__check__EverythingOff__ShouldCallMotorDriverAndBrainboxDriver
[ OK    ] RemainingActorsControlTest.RemainingActorsControl__check__EverythingOff__ShouldCallMotorDriverAndBrainboxDriver (0 ms)
[ RUN   ] RemainingActorsControlTest.RemainingActorsControl__check__UvLampAirOnRestOff__ShouldCallMotorDriverAndBrainboxDriver
[ OK    ] RemainingActorsControlTest.RemainingActorsControl__check__UvLampAirOnRestOff__ShouldCallMotorDriverAndBrainboxDriver (0 ms)
[ RUN   ] RemainingActorsControlTest.RemainingActorsControl__check__UvLampWaterOnRestOff__ShouldCallMotorDriverAndBrainboxDriver
[ OK    ] RemainingActorsControlTest.RemainingActorsControl__check__UvLampWaterOnRestOff__ShouldCallMotorDriverAndBrainboxDriver (0 ms)
[ RUN   ] RemainingActorsControlTest.RemainingActorsControl__check__Pump20nRestOff__ShouldCallMotorDriverAndBrainboxDriver
[ OK    ] RemainingActorsControlTest.RemainingActorsControl__check__Pump20nRestOff__ShouldCallMotorDriverAndBrainboxDriver (0 ms)
[ RUN   ] RemainingActorsControlTest.RemainingActorsControl__check__UvLampAirAndWaterOnRestOff__ShouldCallMotorDriverAndBrainboxDriver
[ OK    ] RemainingActorsControlTest.RemainingActorsControl__check__UvLampAirAndWaterOnRestOff__ShouldCallMotorDriverAndBrainboxDriver (0 ms)
[ RUN   ] RemainingActorsControlTest.RemainingActorsControl__check__UvLampAirAndPumpOnRestOff__ShouldCallMotorDriverAndBrainboxDriver
[ OK    ] RemainingActorsControlTest.RemainingActorsControl__check__UvLampAirAndPumpOnRestOff__ShouldCallMotorDriverAndBrainboxDriver (0 ms)
[ RUN   ] RemainingActorsControlTest.RemainingActorsControl__check__UvLampWaterAndPumpOnRestOff__ShouldCallMotorDriverAndBrainboxDriver
[ OK    ] RemainingActorsControlTest.RemainingActorsControl__check__UvLampWaterAndPumpOnRestOff__ShouldCallMotorDriverAndBrainboxDriver (0 ms)
[ RUN   ] RemainingActorsControlTest.RemainingActorsControl__check__EverythingOn__ShouldCallMotorDriverAndBrainboxDriver
[ OK    ] RemainingActorsControlTest.RemainingActorsControl__check__EverythingOn__ShouldCallMotorDriverAndBrainboxDriver (0 ms)
[-----] 8 tests from RemainingActorsControlTest (4 ms total)
```

Abbildung 9.8.: Test der Überwachung sonstiger Aktoren

### 9.1.7. Telekommandos

Es finden Tests statt, bei denen der Eingang aller möglichen Telekommandos (siehe Kapitel 8.1.5) simuliert wird. Dabei wird überprüft, ob die Software das erwartete Verhalten zeigt. Einen Ausschnitt der durchgeführten Tests ist in Abb. 9.9 gezeigt. In diesem Zusammenhang sei noch einmal darauf hingewiesen, dass keine gesonderten Telemetrie-Tests durchgeführt werden, da die Generierung von Telemetrie innerhalb der Regelungen überprüft wird. Weiterhin findet im Rahmen des Systemtests (siehe Kapitel 9.2.2) eine Überprüfung statt, dass tatsächlich Telemetrie versendet wird.

## 9.1. Unittests

```
[-----] 76 tests from ControlTest
[ RUN    ] ControlTest.Control__TurnOnUvLampAir__ShouldCallRemainingActorsControl1
[ OK     ] ControlTest.Control__TurnOnUvLampAir__ShouldCallRemainingActorsControl1 (0 ms)
[ RUN    ] ControlTest.Control__TurnOffUvLampAir__ShouldCallRemainingActorsControl1
[ OK     ] ControlTest.Control__TurnOffUvLampAir__ShouldCallRemainingActorsControl1 (0 ms)
[ RUN    ] ControlTest.Control__TurnOnHeater__ShouldCallBrainbox1
[ OK     ] ControlTest.Control__TurnOnHeater__ShouldCallBrainbox1 (0 ms)
[ RUN    ] ControlTest.Control__TurnOffHeater__ShouldCallBrainbox1
[ OK     ] ControlTest.Control__TurnOffHeater__ShouldCallBrainbox1 (0 ms)
[ RUN    ] ControlTest.Control__StartBlowerCycle__ShouldDoNothing
[ OK     ] ControlTest.Control__StartBlowerCycle__ShouldDoNothing (0 ms)
[ RUN    ] ControlTest.Control__StopBlowerCycle__ShouldDoNothing
[ OK     ] ControlTest.Control__StopBlowerCycle__ShouldDoNothing (0 ms)
[ RUN    ] ControlTest.Control__SetFanSpeed1__ShouldChangeValueStore
[ OK     ] ControlTest.Control__SetFanSpeed1__ShouldChangeValueStore (0 ms)
[ RUN    ] ControlTest.Control__SetFanSpeed2__ShouldCallRemainingActorsControl
[ OK     ] ControlTest.Control__SetFanSpeed2__ShouldCallRemainingActorsControl (0 ms)
[ RUN    ] ControlTest.Control__OpenCo2Valve__ShouldCallBrainbox1
[ OK     ] ControlTest.Control__OpenCo2Valve__ShouldCallBrainbox1 (0 ms)
```

Abbildung 9.9.: Ausschnitt der Konsolen Ausgabe Telekommando-Tests

### 9.1.8. Coverage

In Verbindung mit den durchgeführten Unittests wird auch die Testcoverage betrachtet. Das bedeutet, dass überprüft wird, wie viele Zeilen des implementierten Codes durch die Tests tatsächlich ausgeführt sind. Entsprechend erfolgt eine Anpassung der Tests, um sicherzustellen, dass möglichst alle Codezeilen mindestens einmal durchlaufen werden. Die Coverage ermittelt sich mithilfe des Tools Gcov, da dies leicht in die existierenden Tests integriert werden kann.

In Abb. 9.10 ist die Testabdeckung der Regelungstests dargestellt.


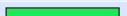
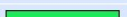
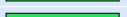

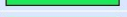
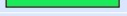
File	Line Coverage ↕			Function Coverage ↕		
	Rate	Total	Hit	Rate	Total	Hit
<a href="#">air_speed_control.cpp</a>	 100.0 %	24	24	100.0 %	2	2
<a href="#">co2_control.cpp</a>	 100.0 %	86	86	100.0 %	3	3
<a href="#">filter_control.cpp</a>	 100.0 %	37	37	100.0 %	2	2
<a href="#">humidity_control.cpp</a>	 100.0 %	18	18	100.0 %	2	2
<a href="#">o2_control.cpp</a>	 100.0 %	14	14	100.0 %	2	2
<a href="#">remaining_actors_control.cpp</a>	 100.0 %	16	16	100.0 %	2	2
<a href="#">temperature_control.cpp</a>	 100.0 %	26	26	100.0 %	2	2

Abbildung 9.10.: Coverage Report der Regelungen

Abb. 9.11 zeigt die Testabdeckung der Telekommando-Tests.

File	Line Coverage $\updownarrow$			Function Coverage $\updownarrow$		
	Rate	Total	Hit	Rate	Total	Hit
<a href="#">control.cpp</a>	 <b>100.0 %</b>	223	223	<b>100.0 %</b>	55	55

Abbildung 9.11.: Coverage Report der Telekommandotests

## 9.2. Hardwarenahe Tests

In diesem Kapitel werden Tests vorgestellt, in denen elektronische Bauteile verwendet werden.

### 9.2.1. Tests der Treiber

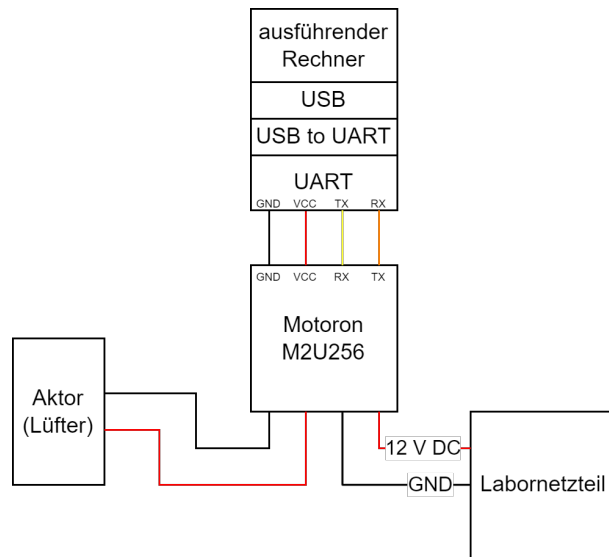
Im Folgenden sind mehrere Tests beschrieben, welche die Funktionalität der entwickelten Treiber überprüfen.

#### Motoron

Um die Funktionalität der Motoron M2U256 Motorsteuerung zu testen, müssen auf dieser zunächst Stiftleisten festgelötet werden. Mit Hilfe dieser kann dann eine zuverlässige Steckverbindung mit einem USB-to-UART-Kabel hergestellt werden. Der ausführende Rechner wird per USB mit dem UART der Motorsteuerung verbunden. Zunächst wird mit dem zugehörigen Kommando die Firmwareversion der Motorsteuerung ausgelesen. Die passende Antwort kommt per UART zurück.

Der nächste Test besteht darin, verschiedene Eingangsspannungen an die passenden Pins anzulegen und diese im Anschluss ebenfalls per Kommando abzufragen.

Zuletzt wird ein Gleichstrommotor in Form eines Lüfters und eine 12 V Spannungsversorgung an die Motorsteuerung angeschlossen. Abb. 9.12 zeigt den Testaufbau schematisch und Abb. 9.13 ist ein Foto des Aufbaus.



**Abbildung 9.12.:** Schematischer Aufbau des Hardwaretests mit Motorsteuerung und Lüfter



**Abbildung 9.13.:** Physikalischer Aufbau des Hardwaretests mit Motorsteuerung und Lüfter

Es werden verschiedene Geschwindigkeiten, auf welche die Steuerung regeln soll,

eingestellt. Um zu überprüfen, ob dies wirklich geschehen ist, wird die aktuelle Geschwindigkeit per Kommando abgefragt und zusätzlich per visueller Wahrnehmung validiert. Es werden beide Drehrichtungen getestet.

Alle Tests verlaufen erfolgreich.

### **Brainboxes**

Zum Test der Brainboxes vom Typ ED-549, welche acht analoge Eingänge besitzt, werden ein Labornetzteil mit variabler Spannungsversorgung, ein Netzkabel, ein analoger Feuchtigkeitssensor und ein analoger Distanzsensoren verwendet.

Das Brainboxes-Gerät wird über das Labornetzteil mit Spannung versorgt und mit dem Netzkabel eine Verbindung mit einem Computer hergestellt. Im ersten Test wird das Webinterface der Brainboxes aufgerufen, indem die lokale IP-Adresse aus der Brainboxes-Dokumentation in die Adresszeile eines Webbrowsers eingegeben wird. Hier erscheint eine Übersicht der angelegten Spannungen. Im ersten Schritt sagt diese aus, dass keine angelegt ist. Nun wird mit dem Labornetzteil eine Spannung an einen der Eingänge angelegt, welche auch in der Weboberfläche zu sehen ist. Die Spannung wird mehrfach variiert und geprüft, ob der am Labornetzteil eingestellte Wert mit dem Messwert aus dem Webinterface übereinstimmt. Diese Überprüfung bestätigt die Zuverlässigkeit der Messungen.

Der entwickelte Brainboxes-Treiber wird verwendet, um den analogen Wert des spezifischen Pins auszulesen, was ebenfalls erfolgreich durchgeführt werden kann.

Im Anschluss werden analoge Distanz- und Bodenfeuchtigkeitssensoren mit Spannung versorgt und an das Brainboxes-Gerät angeschlossen. Diese Sensoren sind zwar kein Teil des AMS, jedoch für den Test ausreichend, da lediglich ein von einem Sensor hervorgerufenen analoges Signal benötigt wird. Die zu messende Distanz wird variiert und es wird geprüft, ob sich der gemessene Spannungswert wie erwartet verändert. Dasselbe wird mit dem Feuchtigkeitssensor wiederholt.

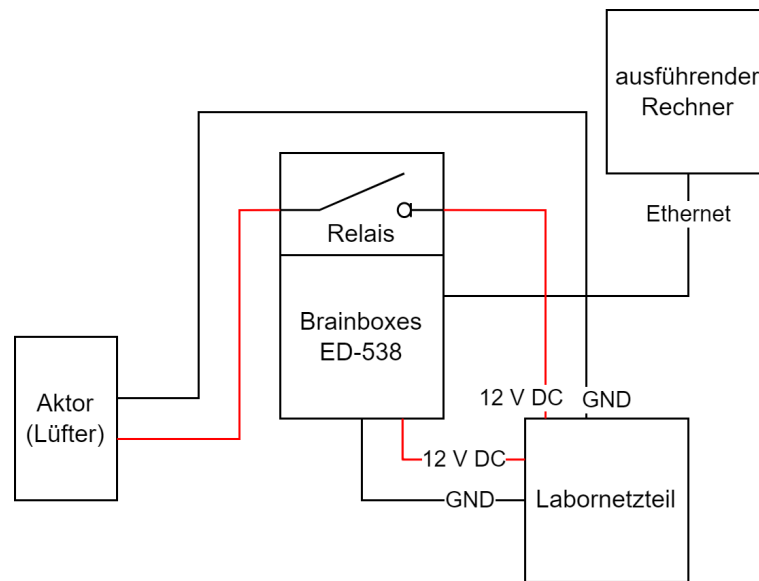
Alle durchgeführten Tests sind erfolgreich. Bei korrektem Anschluss ist der implementierte Treiber in der Lage, analoge Messwerte von den Brainboxes ED-549 zu empfangen.

Die Brainboxes ED-538 besitzt vier Relais und vier digitale Eingänge. Sie wird zu Beginn ebenfalls mit Strom versorgt und mit einem Computer verbunden. Daraufhin wird auch hier die Weboberfläche aufgerufen.

Im ersten Schritt werden die Relais über das Webinterface geschaltet und dann mit einem Multimeter der Durchgang gemessen. Dies funktioniert wie erwartet.

Im nächsten Schritt werden die Relais über den Brainboxes-Treiber geschaltet. Das Relais wird in eine elektronische Schaltung mit einem Lüfter und einer Spannungsversorgung integriert (siehe Abb. 9.14 und Abb. 9.15). Im Testprogramm wird das Relais über den Brainboxes-Treiber für zehn Sekunden aktiviert und dann deaktiviert. Es ist zu sehen, dass der Lüfter für zehn Sekunden eingeschaltet wird.

Auch bei dieser Testreihe sind alle Tests erfolgreich. Bei korrektem Anschluss können Relais vom Brainboxes-Treiber geschaltet werden.



**Abbildung 9.14.:** Schematischer Aufbau des Hardwaretests mit Brainboxes ED-538 und Lüfter

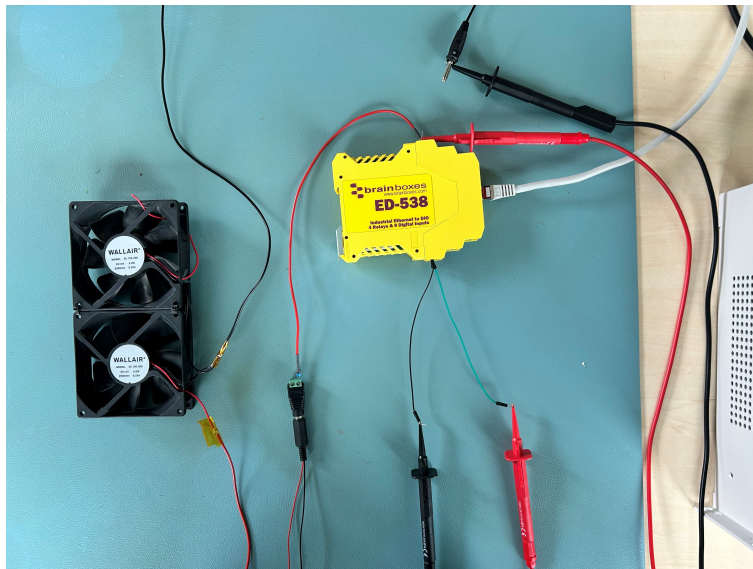


Abbildung 9.15.: Physikalischer Aufbau des Hardwaretests mit Brainboxes ED-538 und Lüfter

### 9.2.2. Systemtest

In diesem Systemtest wird getestet, ob die Software in der Lage ist, ein Telekommando zu empfangen und daraufhin einen Aktor über den Brainboxes-Treiber zu schalten.

Zum Versenden des Telekommandos wird das vom DLR entwickelte grafische Interface **Command Exchange Link** verwendet. Dies ist vom fachlichen Betreuer gesetzt. Alternativ hätte auch Homebase genutzt werden können, welches ebenfalls vom DLR entwickelt wurde und ähnliche Funktionalitäten hat. CEL befindet sich gerade in der Entwicklung. Dieser Systemtest ist eine passende Gelegenheit, um auch die Funktionalität der grafischen Schnittstelle zu testen.

Als Input nutzt CEL das entwickelte Telekommunikationsmodell (siehe Kapitel 8.1.5). Durch dieses ist die Software in der Lage, die gewünschten Telekommandos zu versenden und die eingehende Telemetrie zu identifizieren<sup>3</sup>.

---

<sup>3</sup>in aktueller Version kann keine Telemetrie empfangen werden (Software befindet sich noch in Entwicklung)



Im Test werden verschiedene Telekommandos versendet, die die AMS-Software kommandieren, den Brainboxes-Treiber aufzurufen, um unterschiedliche Relais ein- oder auszuschalten.

Das Remote IO Gerät ED-538 von Brainboxes wird über ein Ethernetkabel mit dem ausführenden Rechner verbunden. Über ein Labornetzteil erhält das Gerät eine Spannungsversorgung. Das Relais des Brainboxes-Gerätes ist in Reihe geschaltet mit einem Aktor und einer Spannungsversorgung. Der Aufbau ist identisch wie bei Kapitel 9.2.1. In Abb. 9.14 ist der Aufbau schematisch dargestellt. Abb. 9.15 zeigt ein Foto des Systemaufbaus.

Im Test werden acht verschiedene Telekommandos versendet, welche die vier Relais des angeschlossenen Brainboxes-Gerätes jeweils ein- oder ausschalten. Zunächst wird über die Weboberfläche der Brainboxes überprüft, ob der gewünschte Relaiszustand eingetreten ist. Um dies weiter zu validieren, wird der Durchgang der Relais zunächst mit einem Multimeter überprüft. Zuletzt wird der Stromkreis aufgebaut, wie in Abb. 9.14 schematisch dargestellt. Wenn nun das Relais geschlossen wird, wird der Aktor, der in diesem Test ein Lüfter ist, aktiviert. Der Lüfter ersetzt die Aktoren, welche im Realfall an den zugehörigen Relais angeschlossen sind, da diese zum Testzeitpunkt noch nicht verfügbar sind.

Zusätzlich wird im Rahmen dieser Tests überprüft, ob Housekeeping-Telemetrie generiert und versendet wird. Die Validierung erfolgt durch die Überprüfung, ob Pakete zu den erwarteten Zeitpunkten an CEL versendet werden. Da sich CEL noch in Entwicklung befindet, kann die Software die Pakete bisher nicht identifizieren. Um dieses Problem zu umgehen, wird Homepage anstelle von CEL zu diesem Zweck genutzt. Die erwarteten Pakete werden empfangen.

Mit diesem Systemtest wurde sichergestellt, dass beim Empfang eines Telekommandos das erwartete Relais aktiviert und somit im Realfall der entsprechende Aktor geschaltet wird. Weiterhin wurde validiert, dass Telemetrie versendet wird.

# 10. Bewertung der Aufgaben

An dieser Stelle wird beschrieben, ob und wie die gesetzten Aufgaben bearbeitet werden.

1. Um die Software in die Lage zu versetzen, Messwerte von den Sensoren zu empfangen und Aktoren schalten zu können, werden Treiber implementiert (siehe Kapitel 8.1.1).
  - 1.1 Analoge Sensoren werden von einem Brainboxes-Gerät und dem zugehörigen Treiber ausgelesen. Andere Sensoren kommunizieren über UART, auch für diese werden entsprechende Treiber implementiert.
  - 1.2 Zur Steuerung der meisten Aktoren wird ebenfalls der Brainboxes-Treiber verwendet. Für die Lüfter wird ein Treiber für eine Motorsteuerung implementiert, mit dessen Hilfe die Geschwindigkeit eingestellt werden kann.
2. Zur Validierung der Erreichung von Aufgabe 1. werden Hardwaretests (siehe Kapitel 9.2.1) durchgeführt.
  - 2.1 Es wird validiert, dass über die Brainboxes analoge Sensorsignale empfangen werden können. Es kann nicht getestet werden, ob die Treiber der per UART kommunizierenden Sensoren ihren Zweck erfüllen, da die Geräte nicht vorhanden sind.
  - 2.2 In den Hardwaretests wird belegt, dass Relais zum Schalten von Aktoren angesteuert werden können und dass die Geschwindigkeitssteuerung von Lüftern über eine Motorsteuerung funktioniert.
3. Es werden folgende Regelungen entwickelt (siehe Kapitel 8.1.3):
  - 3.1 Temperatur
  - 3.2 Luftfeuchtigkeit
  - 3.3 Luftgeschwindigkeit

### 3.4 CO<sub>2</sub>-Gehalt

4. Durch die Unittests in Kapitel 9.1 wird sichergestellt, dass die Regelungen die Aktoren entsprechend der gemessenen Umwelteinflüsse und gesetzten Grenzwerte schalten. In diesem Zusammenhang werden auch ungültige Eingaben überprüft. Es wird sichergestellt, dass die Software diesen Zweck erfüllt. Um zu überprüfen, dass die Regelung im physischen Gesamtsystem wie erwartet funktioniert, sind jedoch umfangreichere Tests mit der integrierten Hardware nötig, sobald diese verfügbar ist.
5. Es werden verschiedene Telekommandos entwickelt (siehe Kapitel 8.1.5). Über einen gesonderten Unittest wird gezeigt, dass bei Empfang der verschiedenen Kommandos jeweils die erwartete Aktion zuverlässig ausgeführt wird (siehe Kapitel 9.1).
  - 5.1 Telekommandos zum Schalten der einzelnen Aktoren und Regelkreise sind enthalten.
  - 5.2 Stellwerte können per Telekommando geändert werden.
6. Telemetrie kann versendet werden (siehe Kapitel 8.1.5).
  - 6.1 Über einen Systemtest wird sichergestellt, dass Housekeeping-Telemetrie versendet wird. (siehe Kapitel 9.2.2)
  - 6.2 In den Unittests der Regelungen (siehe Kapitel 9.1) wird verifiziert, dass Telemetrie an den gewünschten Stellen generiert wird.
7. Durch den abschließenden Systemtest (siehe Kapitel 9.2.2) ist sichergestellt, dass die Software Telekommandos empfangen und daraufhin Aktoren ansteuern kann. Weiterhin ist gezeigt, dass Telemetrie versendet wird.

Die einzelnen Aufgaben und deren Erreichung sind in Tabelle 10.1 dargestellt. Das Wort Telekommando ist mit *TK* und Telemetrie mit *TM* abgekürzt.

## 10. Bewertung der Aufgaben

---

Aufgabe	Beschreibung	Kapitel	Resultat
1.	Kommunikation mit Sensoren und Aktoren	8.1.1	✓
1.1	Sensoren lesen	8.1.1	✓
1.2	Aktoren steuern	8.1.1	✓
2.	Test der Kommunikation	9.2.1	teilweise
2.1	Test Sensoren lesen	9.2.1	×
2.2	Test Aktoren schalten	9.2.1	✓
3.	Umweltparameter regeln	8.1.3	✓
3.1	Temperatur regeln	8.1.3	✓
3.2	Luftfeuchtigkeit regeln	8.1.3	✓
3.3	Luftgeschwindigkeit regeln	8.1.3	✓
3.4	CO2-Gehalt regeln	8.1.3	✓
4.	Unittests der Regelungen	9.1	✓
5.	Telekommandos empfangen	8.1.5	✓
5.1	TKs Schalten Aktoren/Regelungen	8.1.5 & 9.1	✓
5.2	TKs zum Ändern der Stellwerte	8.1.5 & 9.1	✓
6.	Telemetrie versenden	8.1.5	✓
6.1	Versenden von Housekeeping TM	8.1.5 & 9.2.2	✓
6.2	Warnungen und Fehler als TM	8.1.5 & 9.1	✓
7	Systemtest	9.2.2	✓

**Tabelle 10.1.:** Verifizierungsmatrix der Aufgaben

Alle Aufgaben bis auf den Test der Sensoren, die über UART kommunizieren, wurden erfolgreich bearbeitet.

# 11. Aufgetretene Probleme

Während der Durchführung dieser Arbeit traten einige Probleme auf.

Das EDEN LUNA Projekt befindet sich noch in einem sehr frühen Entwicklungsstadium. Obwohl zu Beginn ein grobes Konzept vorhanden war, das als Orientierung diente, standen einige Entscheidungen noch aus. Um sich iterativ der optimalen Lösung zu nähern, wurde deshalb Prototyping eingesetzt. Aufgrund der hohen Dynamik nahm dieser Prozess mehr Zeit in Anspruch als ursprünglich geplant. So gab es während des Projekts immer wieder größere Änderungen wie die Entfernung der ursprünglich vorgesehenen Sauerstoffregelung. Der von der Hochschule festgelegte Zeitrahmen für diese Arbeit erwies sich als nicht ausreichend, denn kurz nach Abgabe dieser findet erst das Critical Design Review durch die SystemingenieurInnen statt. Ab diesem Meilenstein beginnt im Projektmanagement typischerweise die Fertigung, weil die wichtigsten Entscheidungen getroffen sind, die die Entwicklung maßgeblich beeinflussen.

Ein weiterer kritischer Punkt zeigte sich gegen Ende der Arbeit. Dieses fiel auf die Sommerferien und die SystemdesignerInnen des AMS und der fachliche Betreuer waren deshalb nicht zugegen. Dies hatte zur Folge, dass es immer wieder zu Verzögerungen bei der Klärung von Fragen kam, wodurch das Prototyping nicht wie geplant weitergeführt werden konnte.

Über die Implementierungsarbeiten hinweg traten immer wieder Probleme mit dem SCons Build-System auf. Zu Beginn waren nicht alle neuesten Versionen der benutzten OUTPOST Repositorien miteinander kompatibel, weshalb teilweise auf ältere Commits ausgewichen werden musste. Infolgedessen gab es jedoch das Problem, dass in manchen Teilen bereits eine neue Funktion im Build-System namens lipdep genutzt wurde und in manchen nicht. In diesem Zusammenhang ist es wichtig zu wissen,

dass libdep die Art, wie SCons baut, und die Struktur des SConstructs<sup>1</sup> grundlegend ändert. Es musste eine hybride Form genutzt werden, welche das SConstruct sehr komplex machte. Wenn es im Verlauf des Projektes zu Fehlern beim Kompilieren oder Linken kam, musste oft viel Zeit aufgewendet werden, um diese zu eliminieren. Ein weiteres Problem gab es bei der Implementierung des Treibers für die Motorsteuerung. Es trat das Phänomen auf, dass Firmware und Eingangsspannung abgefragt werden konnten und auch sinnvolle Rückmeldungen kamen, jedoch die angeschlossenen Lüfter nicht aktiviert werden konnten. Letztlich stellte sich heraus, dass eine aktive Error Flag, die sich auch nach einem Reboot nicht zurücksetzte, der Grund für das Problem war, wodurch das Starten der Lüfter verhindert wurde. Ein Großteil der vorgesehenen Sensoren und Aktoren war während der Testphase noch nicht vorhanden. Dies viel besonders bei den Sensoren, welche über UART kommunizieren, ins Gewicht, da der entwickelte Treiber somit nicht getestet werden konnte. Die anderen Sensoren und Aktoren werden über die Brainboxes oder Motorsteuerungen angesteuert, die zum Test bereitstanden. Deshalb konnten hier die zugehörigen Treiber wie geplant getestet werden.

---

<sup>1</sup>Skript, welches Bauprozess steuert

# 12. Fazit

In diesem Kapitel wird zusammengefasst, welches Ergebnis mit dieser Arbeit erreicht wird. Im Anschluss wird auf die weitere Entwicklung des AMS eingegangen. Abschließend wird die Raumfahrt-Tauglichkeit von EDEN LUNA bewertet und ein Ausblick auf zukünftige Iterationen gegeben.

## 12.1. Ergebnis

Eine Atmosphärensteuerung für das EDEN LUNA Projekt wurde erfolgreich entwickelt. Diese enthält verschiedene Regelkreise, mit welchen Temperatur, Luftfeuchtigkeit, Luftgeschwindigkeit und CO<sub>2</sub>-Gehalt geregelt werden können. Die Regelungen steuern Aktoren an und lesen Sensoren aus, für welche entsprechende Treiber implementiert wurden. Des Weiteren kann der Zustand der verbauten Filter und weitere Größen wie Ethylen- und Ozon-Gehalt überwacht werden. Die entwickelte Software zeichnet sich durch ihre Fähigkeit aus, über Telekommandos gesteuert zu werden. Sie generiert Telemetriedaten, die sowohl Sensorwerte als auch Aktorenzustände umfassen.

Diese Regelungstechnologie schafft in Verbindung mit den übrigen Subsystemen eine optimierte Umgebung, die zuverlässig die gewünschten Umweltbedingungen aufrechterhält. Dies ermöglicht es, in lebensfeindlichen Gebieten frische Nahrung anzubauen, was einen bedeutenden Fortschritt darstellt.

Darüber hinaus generiert das System wertvolle wissenschaftliche Daten, die im Kontext des Pflanzenwachstums analysiert werden können. Diese Daten eröffnen neue Erkenntnisse über die komplexen Wechselwirkungen zwischen Pflanzenwachstum und

Umweltparametern. Durch die Beobachtungen kann der automatisierte Anbau von Pflanzen hinsichtlich Ertrag und Ressourceneffizienz maximiert werden. Das EDEN LUNA Projekt setzt damit neue Maßstäbe in der automatisierten Landwirtschaft und der Erforschung extremer Lebensräume.

## 12.2. Ausblick

Der nächste Schritt ist nun, weitere Tests durchzuführen. Sobald die komplette Hardware vorhanden ist, sollte diese zunächst einzeln und dann in Kombination getestet werden. Besonders ein Test der Sensoren, welche über UART kommunizieren ist wichtig, da die zugehörigen Treiber bisher nicht erprobt werden konnten.

Auch eine erneute Umstrukturierung des Codes ist vorgesehen. Wie in Kapitel 8.1.2 beschrieben, gibt es momentan eine Value Store Klasse als zentrale Einheit zum Datenaustausch. So wird der Empfang von neuen Daten und die Stellen, an denen sie benötigt werden, entkoppelt. Zu Beginn des Projektes wurde mit dem fachlichen Betreuer erörtert, wie die Verbindung zwischen den per Telekommando erhaltenen Werten und den Regelungen, welche diese Werte benötigen, aussehen kann. Der Value Store war die genannte Lösung und gilt deswegen als vom fachlichen Betreuer gesetzt. Das Konzept ist funktional und praktisch, jedoch architektonisch nicht optimal. Es entsteht eine Art Netz- beziehungsweise Sternstruktur, weil viele Klassen vom Value Store abhängen. Eine geschichtete Struktur ohne Value Store, bei der Grenzwerte direkt in den Regelungsklassen abgelegt werden, wäre besser, da hier die Abhängigkeiten klarer definiert und besser kontrollierbar sind. Momentan könnte eine Instanz die Zugriff auf den Value Store hat, auch Werte abfragen oder ändern, welche diese nicht betreffen, sondern einer anderen zugehörig sind. Durch eine Umstrukturierung hätte jede Schicht ihre spezifischen Verantwortlichkeiten, was die Wartbarkeit und Erweiterbarkeit des Systems verbessert.

Weitere Regelungen für die Bereiche, in denen sich Menschen aufhalten, müssen von den SystemdesignerInnen klar definiert und dann umgesetzt werden. Es muss sichergestellt werden, dass sich die Besatzung gefahrlos im Container aufhalten kann. Verschiedene Lüfter und eine Klimaanlage sind zur Regelung vorgesehen.



Kurz vor Abgabe dieser Arbeit wurden die definierten Abläufe einiger Regelungen noch einmal von den SystemdesignerInnen überarbeitet. Dabei wurde insbesondere die Temperaturregelung maßgeblich verändert. Diese Änderungen müssen in den bestehenden Code aufgenommen werden.

Auch die Implementierung der Betriebsmodi "STARTUP" und "STANDBY" muss geschehen, sobald diese klar von den SystemdesignerInnen definiert sind.

Es ist denkbar, dass ursprünglich vorgesehene Komponenten wie beispielsweise die Sauerstoffregelung wieder hinzugefügt werden, wenn diese doch benötigt werden sollte.

Nach Einbau der Gewächshauskomponenten in den Container muss getestet werden, wie stark die Aktoren die Umweltparameter beeinflussen und ob das System wirklich physisch in der Lage ist, die Größen in ihrem definierten Bereich zu halten.

Die Beendigung des EDEN LUNA Projektes markiert jedoch noch lange nicht die Erreichung des eigentlichen Ziels, Nahrung auf fremden Himmelskörpern anzubauen. Der Container wird in eine Mondsimulation eingegliedert und ausgiebig im täglichen Betrieb getestet. Hierraus werden Informationen gesammelt, um ihn in der nächsten Iteration zu verbessern. Diese ist bereits in Planung unter dem Namen Lunar Agriculture Module - Ground Test Demonstrator (LAM-GTD). Sie soll raumfahrttaugliche Konzepte verfolgen und testen. Technische Standards der European Cooperation for Space Standardization sollen genutzt werden und es wird einen zylinderförmigen Formfaktor haben, um mit konventionellen Launchern transportiert werden zu können. Das System soll im Unterdruckbereich operieren können. Das heißt konkret, es soll den Gegebenheiten auf dem Mond widerstehen und im Habitat einen gewissen Druck aufrecht erhalten können. Dies bedeutet unter anderem, dass keine Löcher in der Außenwand mehr vorgesehen sind. Das System wird darauf ausgelegt, möglichst leicht und kompakt zu sein. Des Weiteren wird es verschiedene Redundanzkonzepte verfolgen, um eine ständige Funktionalität zu gewährleisten. Das LAM-GTD-Konzept wird raumfahrttauglich sein, der in diesem Rahmen entwickelte Prototyp jedoch nicht, da dies den finanziellen Rahmen des Forschungsprojekts sprengen würde. Der Gedanke bei diesem Projekt ist, dass im besten Fall die Raumfahrtindustrie das entwickelte Konzept umsetzt.

# Literatur

- [1] D. e.V. „EDEN ISS greenhouse returns to Bremen and has a new destination.“ (), Adresse: <https://www.dlr.de/en/latest/news/2023/03/eden-iss-greenhouse-returns-to-bremen-and-has-a-new-destination> (besucht am 15.05.2024).
- [2] DLR. „DLR’s EDEN laboratory.“ (), Adresse: <https://eden-iss.net/index.php/2015/07/30/dlrs-eden-laboratory/> (besucht am 10.06.2023).
- [3] P. Zabel. „NASA collaboration with EDEN ISS.“ (), Adresse: <https://eden-iss.net/index.php/category/gallery/> (besucht am 26.07.2023).
- [4] F. Dambowsky. „Gewächshaus EDEN ISS zurück in Bremen mit neuem Ziel.“ (), Adresse: <https://www.dlr.de/de/aktuelles/nachrichten/2023/03/gewaechshaus-eden-iss-zurueck-in-bremen-mit-neuem-ziel> (besucht am 10.06.2023).
- [5] V. Maiwald, V. Maiwald, V. Vrakking, V. Vrakking, P. Zabel, P. Zabel, D. Schubert, D. Schubert, R. Waclavicek, R. Waclavicek, M. Dörn, M. Dorn, L. Fiore, L. Fiore, B. Imhof, B. Imhof, T. Rousek, T. Rousek, V. Rossetti, V. Rossetti, C. Zeidler und C. Zeidler, „From ice to space: a greenhouse design for Moon or Mars based on a prototype deployed in Antarctica,“ *Ceas Space Journal*, 2020. DOI: 10.1007/s12567-020-00318-4.
- [6] D. Sempstrott. „Growing Plants in Space.“ (), Adresse: <https://www.nasa.gov/exploration-research-and-technology/growing-plants-in-space/> (besucht am 11.06.2023).
- [7] NASA. „Veggie.“ (), Adresse: [https://www.nasa.gov/wp-content/uploads/2019/04/veggie\\_fact\\_sheet\\_508.pdf](https://www.nasa.gov/wp-content/uploads/2019/04/veggie_fact_sheet_508.pdf) (besucht am 11.06.2023).
- [8] S. S. Corporation. „Gemüseanbau im Weltraum.“ (), Adresse: <https://www.sierraspace.com/blog/growing-vegetables-in-space/> (besucht am 28.07.2023).
- [9] T. S. Rajalakshmi, A. K. Menon, A. Krishnan und P. S. Arounane, „Remotely controlled automated greenhouse system,“ *AIP Conference Proceedings*, 2023. DOI: 10.1063/5.0149334.

- [10] M. McArthur(Nasa). „Growing Peppers on the ISS Is Just the Start of Space Farming.“ (), Adresse: <https://www.wired.com/story/growing-peppers-on-the-iss-is-just-the-start-of-space-farming/> (besucht am 28.07.2023).
- [11] H. Kilifarev und D. Genkov, „Development of a Microcontroller Based Automated Greenhouse Cultivation System for Mushrooms - Hardware,“ *International Conference on Applied Informatics*, 2023. DOI: 10.1109/icai58806.2023.10339014.
- [12] V. Vrakking, *EDEN LUNA Design Definition File*. 04.03.2024.
- [13] StackShare. „CMake vs SCons.“ (), Adresse: <https://stackshare.io/stackups/cmake-vs-scons#> (besucht am 25.07.2023).
- [14] DLR-RY. „OUTPOST.“ (), Adresse: <https://github.com/DLR-RY/outpost-core> (besucht am 05.08.2023).
- [15] mbedded.ninja. „Compiling FreeRTOS With C++.“ (), Adresse: <https://blog.mbedded.ninja/programming/operating-systems/freertos/compiling-freertos-with-cpp/> (besucht am 23.08.2023).
- [16] V. Lorz. „how to work with std::thread on freeRTOS?“ (), Adresse: <https://stackoverflow.com/questions/64490447/how-to-work-with-stdthread-on-freertos> (besucht am 23.08.2023).
- [17] spectra. „Spectra PowerBox 3000E-Series.“ (), Adresse: <https://industriepc.de/media/2f/71/50/1648118669/Datasheet-Spectra-PowerBox-3000E.pdf> (besucht am 26.07.2023).
- [18] Brainboxes. „ED-538.“ (), Adresse: <https://www.brainboxes.com/wp-content/uploads/2021/10/ed-538-ethernet-to-4-x-relay-8-digital-inputs.jpg> (besucht am 12.06.2023).
- [19] Brainboxes. „ED-538.“ (), Adresse: <https://www.brainboxes.com/files/catalog/product/ED/ED-538/documents/ED-538-datasheet.pdf> (besucht am 12.06.2023).
- [20] ESA. „Telemetry & Telecommand.“ (), Adresse: [https://www.esa.int/Enabling\\_Support/Space\\_Engineering\\_Technology/Onboard\\_Computers\\_and\\_Data\\_Handling/Telemetry\\_Telecommand](https://www.esa.int/Enabling_Support/Space_Engineering_Technology/Onboard_Computers_and_Data_Handling/Telemetry_Telecommand) (besucht am 18.06.2023).
- [21] V. Vrakking, *EDEN LUNA Verbesserungen der Subsysteme To Do Liste*. 2024.
- [22] Brainboxes. „ED-549.“ (), Adresse: <https://www.brainboxes.com/product/remote-io/digital/ed-549> (besucht am 22.05.2024).
- [23] Brainboxes. „ED-538.“ (), Adresse: <https://www.brainboxes.com/product/remote-io/digital/ed-538> (besucht am 22.05.2024).

- [24] Brainboxes. „ED-527.“ (), Adresse: <https://www.brainboxes.com/product/remote-io/digital/ed-527> (besucht am 22.05.2024).
- [25] P. D. med. Jan Baum. „Atemkalk: Hinweise zu korrektem Umgang und fachgerechter Nutzung.“ (), Adresse: <https://www.ai-online.info/images/ai-ausgabe/1999/06-1999/AI990651.PDF> (besucht am 27.07.2023).
- [26] P. Corporation. „Motoron M2U256 Dual Serial Motor Controller.“ (), Adresse: <https://www.pololu.com/product/5067> (besucht am 04.06.2024).
- [27] C. Grieger. „DC-Motoren steuern.“ (), Adresse: <https://elektro.turanis.de/html/prj147/index.html> (besucht am 04.06.2024).
- [28] Pololu. „Motoron Motor Controller library for Arduino.“ (), Adresse: <https://github.com/pololu/motoron-arduino> (besucht am 07.08.2023).
- [29] AtlasScientific. „EZO-HUM.“ (), Adresse: <https://files.atlas-scientific.com/EZO-HUM-P-S-Datasheet.pdf> (besucht am 27.07.2023).
- [30] F. J. Heinrich, „Trennung der Wurzelatmung und rhizomikrobiellen Atmung durch O<sub>2</sub>-Limitierung,“ 2009.
- [31] K. P. for Ontario. „Plant Damage as a Result of CO<sub>2</sub> Supplementation.“ (), Adresse: <https://www.ontario.ca/page/supplemental-carbon-dioxide-greenhouses#section-4> (besucht am 22.07.2023).

# Anhang A.

## Link zum Git Repository

Folgender Link führt zu einem Git Repository, welches den im Rahmen dieser Arbeit entwickelten Code, die Datenblätter der Sensoren, Aktoren und Steuergeräte, Ergebnisse der durchgeführten Unit- und Coveragetests und bisher unveröffentlichte Dokumente, die in dieser Arbeit referenziert werden, enthält.

<https://gitlab.com/Schnitzelhead/anhang-bachelorarbeit>