

Erstellung der Hard- und Software zur Steuerung der Segel im Sailwind 4.0 Projekt

Projektarbeit

An der Fakultät Elektro- und Informationstechnik

vorgelegt von

Benedikt Horn und Nico Finkbeiner

Matrikelnummer: 300889 und 298837

Hochschule Konstanz für Technik, Wirtschaft und Gestaltung

Hochschulbetreuer:

Prof. Dr. Boris Böck

Prof. Dr. Dieter Schwechten

Konstanz den 21. Februar 2024

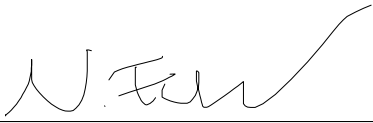
Eidesstattliche Erklärung

Wir versichern hiermit, dass wir dieses Bachelorprojekt mit dem Thema:

Erstellung der Hard- und Software zur Steuerung der Segel im Sailwind 4.0 Projekt

selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt haben.

Konstanz, 21.02.2024



Nico Finkbeiner



Benedikt Horn

Inhaltsverzeichnis

Eidesstattliche Erklärung	i
Abkürzungsverzeichnis	vi
Abbildungsverzeichnis	viii
Quellcodeverzeichnis	x
Vorbemerkung	1
1 Einleitung	2
1.1 Ausgangssituation	3
1.2 Ziel	3
2 Grundlagen zur Segeleinstellung	4
3 Hardware	6
3.1 Ziele	7
3.2 Analyse der bestehenden Hardware Elemente	7
3.2.1 Sensoren	8
3.2.2 Aktoren	9
3.2.3 Zusammenfassung und Übersicht	11
3.3 Auswahl der Platinen Bauteile	12
3.3.1 Mikrocontroller	13
3.3.2 Relais	14
3.3.3 Optokoppler	14
3.3.4 RS485 zu UART Konverter	14
3.3.5 FRAM	15
3.3.6 Operations Verstärker	15
3.3.7 DC/DC-Wandler	15
3.3.8 Stromsensor	16

3.3.9	Konnektoren	17
3.4	Gehäusebauteile	17
3.4.1	Bedienoberfläche	18
3.4.2	Kabelverschraubungen	20
3.5	Platinen Schaltplan	20
3.5.1	STM32F439zi Gruppe	21
3.5.2	Motor Gruppe	22
3.5.3	Optokoppler Gruppe	23
3.5.4	Sensoren Gruppe	23
3.5.5	Buttons und LEDs Gruppe	24
3.6	Printed Circuit Board (PCB)	25
3.7	Probleme	27
3.7.1	Schraubklemmen Löcher	27
3.7.2	Operationsverstärker	27
3.7.3	RS485 Bias Widerstände	27
3.7.4	FRAM Anschlüsse	28
3.7.5	Abstandssensor Präzision	29
3.7.6	Präzision der Analogen Signale	31
4	Software	32
4.1	IO-Modul	34
4.1.1	Digitale Pins	34
4.1.2	Analoge Pins	36
4.2	Motor-Modul	37
4.2.1	Funktionsvorgabe	38
4.2.2	Drehzahlrampe	39
4.3	Steuerung der Linearführung	41
4.3.1	Kalibrierung	41
4.3.2	Umsetzung einer Rollung/Trimmung	43
4.3.3	Optimierung der Bewegungsübergänge	45
4.3.4	Wiederherstellung der Position	48

4.3.5	Fehlerbehandlung	52
4.4	Windmessung mit dem Anemometer	53
4.5	Representational State Transfer (REST) Interface	54
4.6	Weboberfläche	56
4.6.1	Funktionalität	56
4.6.2	Webseiten	56
4.6.3	Umsetzung	57
5	Fazit und Ausblick	59
	Literaturverzeichnis	xi
A	Schaltpläne	xiii
A.1	Schaltplan Übersicht	xiii
A.2	Schaltplan STM32F439 Gruppe	xiv
A.3	Schaltplan Buttons und LED Gruppe	xv
A.4	Schaltplan Sensoren Gruppe	xvi
A.5	Schaltplan Motor Gruppe	xvii
A.6	Schaltplan Endschalter Gruppe	xviii
B	Bauteilliste	xx
C	Valide URL und JSON Formate	xxii
C.1	GET-Anfragen	xxii
C.1.1	GET /data	xxii
C.1.2	GET /data/status	xxii
C.1.3	GET /data/adjustment	xxii
C.1.4	GET /data/sensors	xxiii
C.1.5	GET /data/settings	xxiii
C.2	PUT-Anfragen	xxiii
C.2.1	PUT /data/status/error	xxiii
C.2.2	PUT /data/adjustment	xxiv

C.2.3	PUT /data/status/operating_mode	xxiv
C.2.4	PUT /data/settings	xxiv

Abkürzungsverzeichnis

API	Application Programming Interface
BLDC	Brushless DC
CGI	Common Gateway Interface
DAC	Digital Analog Converter
DC	Direct Current
DHCP	Dynamic Host Configuration Protocol
FRAM	Ferroelectric Random Access Memory
GPIO	General-Purpose Input/Output
HTTP	Hypertext Transfer Protocol
HTML	Hypertext Markup Language
IP	Internet Protocol
JSON	Java Script Object Notation
LED	Light Emitting Diode
LWIP	Lightweight IP
MSB	Most significant Bit
MISO	Master In Slave Out
MOSI	Master Out Slave In
NMEA	National Marine Electronics Association
PCB	Printed Circuit Board
REST	Representational State Transfer
SMD	Surface-Mount Device
SPI	Serial Peripheral Interface
SSI	Server Side Includes

TCP	Transmission Control Protocol
UART	Universal Asynchronous Receiver/Transmitter
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
WP	Write Protect
WREN	Write Enable

Abbildungsverzeichnis

1	Segelwindmühle auf der griechischen Insel Kos	2
2	Segeltheorie: Scheinbarer Wind und Anstellwinkel	4
3	SAILWIND Leistungskurve nach Prof. Schwechten	5
4	Simplifiziertes 3D Modell der Linerführung	6
5	Blockschaltbild der benötigten Ein- und Ausgänge der Platine . . .	12
6	Schaltplan des Texas Instruments TMCS1101 Hall Stromsensor (vgl. [7], S.24)	16
7	3D Modell der Platine innerhalb des Gehäuses	18
8	Bedienoberfläche der Segel Steuerung	19
9	Gesamtschaltplan der Platine	20
10	Schaltplan der Motorgruppe	22
11	Beispielhafte Schaltung eines Optokopplers	23
12	Schaltplan des analogen Eingangs des Abstandssensors	24
13	RC-Filter für einen am Gehäuse angebrachten Knopf	25
14	3D Ansicht des fertigen PCBs	26
15	Messreihen des ausgelesenen Spannungswert bei gleichbleibendem Ab- stand und steigenden Außerbetriebszeiten	29
16	Präzision des Abstandssensors anhand verschiedener Messwerte . .	31
17	Software Modulstruktur	33
18	Drehzahlrampe: Beschleunigen und Bremsen	40
19	Kalibrierungsprozess	42
20	Rollung und Trimmung an der Linearführung	43
21	Optimierung der Steuerbefehle	47
22	Aufbau eines NMEA Telegramm (vgl.[9], S.61)	53
23	Index Seite des Webservers	57
24	Schaltplan Übersicht	xiii
25	Schaltplan der STM32F439 Gruppe	xiv
26	Schaltplan der Buttons und LED Gruppe	xv
27	Schaltplan der Sensoren Gruppe	xvi

28	Schaltplan der Motor Gruppe	xvii
29	Schaltplan der Endschalter Gruppe	xviii

Quellcodeverzeichnis

1	Struktur für digitale Pins	34
2	Einlesen eines digitalen Pin-Zustands	35
3	Detektion einer Flanke	35
4	Struktur für analoge Sensoren	36
5	Konvertierung des rohen Analogwerts	36
6	Struktur des Motors	38
7	Einstellung der Motorfunktionen	39
8	Berechnung der Zielposition	44
9	Positionsberechnung aus dem Pulssignal des Motors	45
10	Speicherformat der Positionsdaten	49
11	FRAM Speichervorgang	50
12	Auslesen der Positionsdaten aus dem FRAM-Puffer	51
13	Empfangen einer Nachricht vom WSWD	54
14	GET /data/adjustment Antwort Paket	55
15	GET /data/adjustment Antwort Paket	58
16	GET-Request 1	xxii
17	GET-Request 2	xxii
18	GET-Request 3	xxii
19	GET-Request 4	xxiii
20	GET-Request 5	xxiii
21	PUT-Request 1	xxiii
22	PUT-Request 2	xxiv
23	PUT-Request 3	xxiv
24	PUT-Request 4	xxiv

Vorbemerkung

In der folgenden Arbeit wird aus Gründen der besseren Lesbarkeit ausschließlich die männliche Form verwendet. Sie bezieht sich auf alle Geschlechter gleichermaßen.

1 Einleitung

In der Durchführung der globalen Energiewende spielt die Windkraft eine große Rolle. Jedoch treffen Projekte zum Bau großer Anlagen immer wieder auf Widerstand der Anwohner aufgrund der Verunstaltung der Landschaft. Das SAILWIND Projekt soll dem entgegen kommen und entstand aus der Motivation heraus, kleine, griechische Segelwindmühlen, zu renovieren.



Abbildung 1: Segelwindmühle auf der griechischen Insel Kos [1]

Vor 3000 Jahren dienten diese noch zum Antrieb von Wasserrädern für den Getreidebau. Mit der Umsetzung des Projekts sollen nun zahlreiche dieser heute ungenutzten Bauwerke zur Erzeugung erneuerbarer Energie umfunktioniert werden. Dabei soll die Optik der Mühlen äußerlich möglichst unverändert bleiben. Lediglich die Segel werden erneuert und die Welle an einen Generator gekoppelt, der bei einer Windstärke von 14 m/s eine Spitzenleistung von 5 kW abgeben soll. Diese reicht aus, um z.B. ein ganzes Restaurant zu versorgen. [2] Um die Windmühle

effizient zu betreiben, soll eine voll automatisierte Steuerung integriert werden, welche die Segel der Windstärke und -Richtung entsprechend einstellt. Die Einstellung der Segel geschieht dabei über die Verbindung einer Seilführung (wie in Abbildung 1 zu sehen) mit einer motorisierten Linearführung.

1.1 Ausgangssituation

Die vorliegende Arbeit beruht auf den Erkenntnissen einer Gruppe von Studenten, die bereits einen Versuch unternahmen, einen maßstabsgetreuen Prototypen einer automatisierten Segelwindmühle zu entwickeln. Für die Umsetzung des Projekts konnte die Firma Igus als Sponsor gewonnen werden, die neben einem Betrag von 10.000 Euro wichtige Bauteile für den Bau der Windmühle, darunter die Linearführung, zur Verfügung stellt. Die Gruppe befasste sich zunächst mit der Entwicklung einer Logikeinheit zur Berechnung der optimalen Segelstellung für die maximale Ausgangsleistung aufgrund der Windbedingungen und anschließend mit dem Entwurf einer Steuerplatine. Zusätzlich musste die Kommunikation zwischen diesen beiden Systemen realisiert werden. Es stellte sich heraus, dass die Umsetzung des Logikteils weitreichende Untersuchungen und Anwendung komplexer Methoden erfordert. Mit Beendigung des Projekts wurde dazu zwar eine Lösung präsentiert, jedoch konnte die Steuerplatine aus zeitlichen Gründen nicht mehr fertiggestellt werden.

1.2 Ziel

Da sich die Aufgabenstellung der vorhergegangenen Gruppe im Rahmen des Bachelorprojekts als zu umfangreich erwies, konzentriert sich diese Arbeit auf die Neuentwicklung der Steuerplatine inklusive Implementierung der Firmware. Das System soll über eine Ethernet-Verbindung die Logikeinheit mit allen relevanten Daten versorgen, sowie umgekehrt Befehle zur Anpassung der Segelstellung empfangen und durch Ansteuerung der Linearführung umsetzen. Die Entwicklung und auch die Wiederverwendung der Logikeinheit ist nicht Teil der Arbeit, es soll lediglich eine Kommunikationsschnittstelle vorbereitet werden.

2 Grundlagen zur Segeleinstellung

In diesem Kapitel wird grob auf die Bedeutung und Funktionsweise der Segelsteuerung eingegangen. Nachfolgende Informationen basieren auf dem Wissensstand der Vorgruppe, in deren Arbeit auf genauere Details zu dieser Thematik verwiesen wird. Eine Skizze, die daraus entnommen wurde, veranschaulicht den Sachverhalt des Segelanstellwinkels:

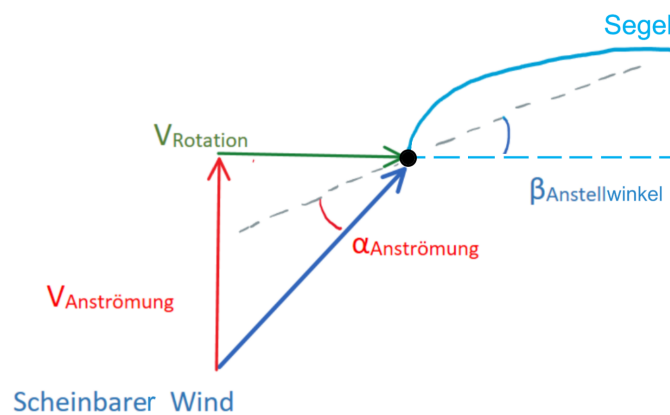


Abbildung 2: Segeltheorie: Scheinbarer Wind und Anstellwinkel

Bei einem Segelboot beeinflusst der Fahrtwind den wahren Wind (hier *Anströmung*) zu einem scheinbaren Wind, der effektiv in das Segel bläst. Ähnlich verhält es sich bei einem in diesem Fall segelartigen Rotorblatt, dass den Wind durch die erzeugte Rotation ablenkt. Verändert sich die Anströmungsgeschwindigkeit, muss der Anstellwinkel des Segels entsprechend angepasst werden, sodass der resultierende scheinbare Wind möglichst effizient vom Segel aufgenommen und in Rotation umgewandelt wird. So existiert zu jeder Windgeschwindigkeit ein optimaler Anstellwinkel. Diese Anpassung wird auch als Trimmen des Segels bezeichnet.

Wie bereits in der Einleitung erwähnt soll bei einer Windgeschwindigkeit von 14 m/s eine Leistung von 5 kW erzeugt werden. Damit diese auch bei stärkerem Wind konstant abrufbar ist, soll die Segelfläche durch Einrollen verkleinert werden. Abbildung 3 zeigt die erwartete Leistungskurve in Abhängigkeit der Windstärke

bei gezielter Trimmung und Rollung der Segel:

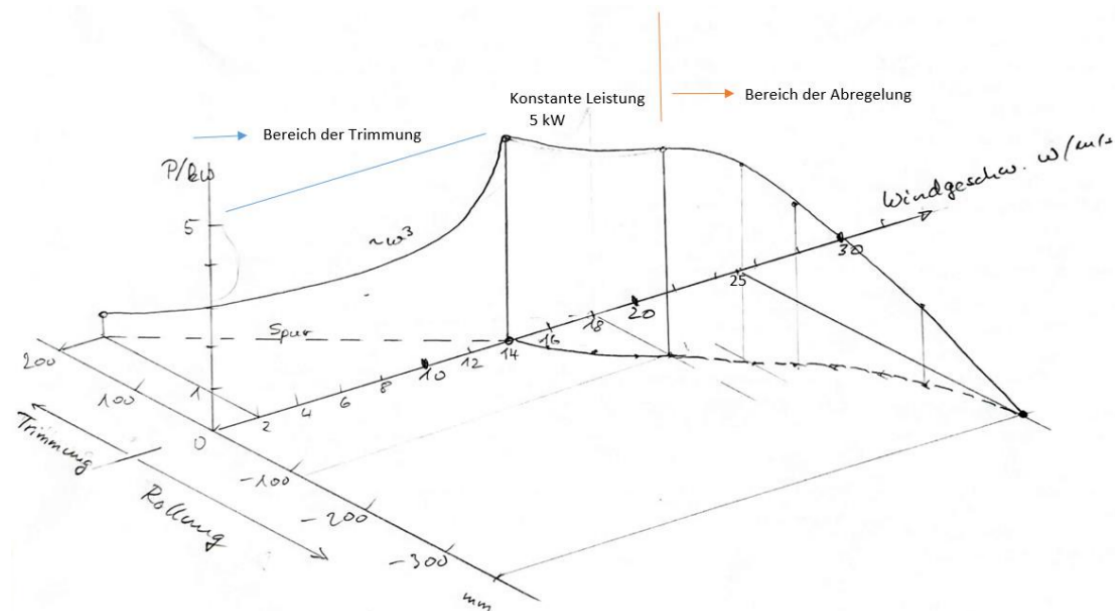


Abbildung 3: SAILWIND Leistungskurve nach Prof. Schwechten

Die Seilführung der Segel (siehe Abbildung 1) ermöglicht dabei einen fließenden Übergang zwischen Rollung und Trimmung durch eine gemeinsame Zugsbewegung aller Seile parallel zur Rotorwelle, die mithilfe der Linearführung ausgeführt werden soll. Bei einer Einschaltgeschwindigkeit von 1 bis 2 m/s sollen die Segel vollständig getrimmt und ausgerollt sein. Bis zur Nenngeschwindigkeit von 14 m/s befindet sich das System im Teillastbereich, in dem der Anstellwinkel schrittweise reduziert werden soll, um stets die optimale Leistung herauszuholen. Mit weiterem Anstieg der Windgeschwindigkeit beginnt der Volllastbereich, in dem die Segel immer weiter eingerollt werden, um die Spitzenleistung von 5 kW zu erhalten. Ab einem kritischen Wert von ca. 18 m/s soll ein Abregelungsprozess durchgeführt werden.

3 Hardware

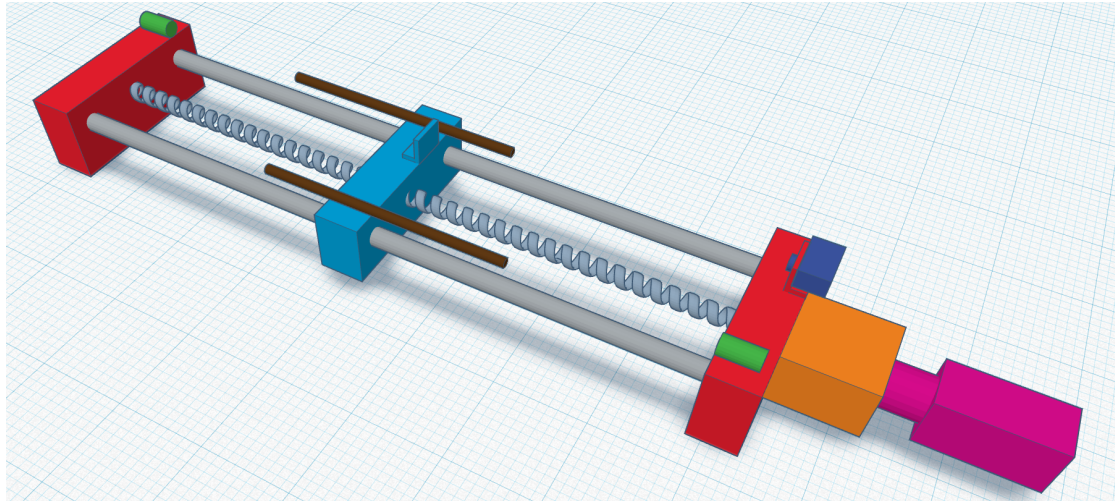


Abbildung 4: Simplifiziertes 3D Modell der Lineraführung

Da sich das Sailwind 4.0 Projekt noch in der Entwicklung befindet, existiert noch kein komplett zusammengebauter Prototyp in dem alle Teile des Projektes zusammenfinden. Die für diese Arbeit relevante Linearführung ist bereits als eigenständiges Objekt zusammengebaut und wie in Abbildung 4 zusehen, aufgebaut. Dabei wird die rotierende Bewegung eines BLDC Motors (Pink) mit der Hilfe eines Getriebes (Orange) und einer Gewindestange in eine translatorische Bewegung umgewandelt, mit welcher der Schlitten (Hellblau) zwischen den beiden stationären Punkten (Rot) hin und her bewegt werden kann. Die beiden Endschalter (Grün) dienen der Kollisionserkennung und können auch dazu genutzt werden, den Bewegungsraum einzuschränken. Dies gelingt durch das Verschieben der beiden Metallstangen (Braun), welche den Platz zwischen dem Schlitten und den stationären Elementen vorgeben. Zusätzlich kann über einen Abstandssensor (dunkel Blau) die Position des Schlittens in Relation zum Abstandssensor bestimmt werden.

Die Linearführung soll später dazu genutzt werden um die Segel des Windkraftwerkes zu trimmen und zu rollen (siehe Kapitel 2). Zusätzlich zu dem präsentierten

Aufbau soll sich auf dem Schlitten zukünftig ein Druckkraftsensor befinden, der die Kraft auf die Rotorwelle messen soll. Dieser wurde allerdings zum Zeitpunkt der Arbeit noch nicht geliefert oder auf dem Prototypen angebracht. Getrennt vom Aufbau soll ebenfalls ein Anemometer an der Kuppel angebracht werden, das die Windrichtung und Geschwindigkeit misst.

3.1 Ziele

Die Hardware der Steuerung zu der eben vorgestellten Linearführung soll dabei folgende Aufgaben übernehmen:

- Ansteuerung aller Sensoren und Aktoren
- Stromversorgung aller Komponenten
- Lokale Bedienung der Steuerung
- Ethernetkommunikation mit dem Controllino

Neben diesen Aufgaben soll die ganze Steuerung in einem zumindest Staub- und Spritzwassergeschützten Gehäuse untergebracht werden. Die Vorgängergruppe hatte sich dieser Aufgabe bereits gewidmet, allerdings wurde der resultierende Prototyp aufgrund der aufgetretenen Fehler und des zu großen Formfaktors nicht weiter verwendet. Diese Probleme sollen in dieser Iteration ebenfalls behoben werden.

3.2 Analyse der bestehenden Hardware Elemente

Um eine geeignete Steuerplatine zu entwerfen, bedarf es einer Analyse der Steuerelemente, welche bereits von der vorhergegangenen Gruppe festgelegt wurden. Diese können in die Kategorien Sensoren und Aktoren gruppiert werden. Dabei wurden die benötigte Spannung und die Schnittstellen betrachtet. Die Schnittstellen geben an, wie mit dem jeweiligen Gerät kommuniziert werden kann und welche davon benötigt werden. Über die Spannung kann die Versorgungsspannung

der Platine ausgewählt werden und bereits ein Spannungspotenzial festgelegt werden.

3.2.1 Sensoren

Zu den Sensoren zählen die folgenden Komponenten:

- Induktiver Endschalter: IFM IFS204
- Optischer Abstandssensor: IFM OGD580
- Anemometer: MESA WSWD
- Druckkraft-Sensor: Burster 8532
- Temperatursensor

Induktiver Endschalter

Zwei der IFM IFS204 Endschalter sind im Design mit eingebaut. Sie funktionieren als PNP-Schließkontakt und geben an einem Ausgang ein digitales 24V Signal aus, sobald sie ausgelöst werden (vgl. [3], S.6). Dabei hat jeder Endschalter eine Betriebsspannung von 9-30V.

Optischer Abstandssensor

Der Abstandssensor OGD580 funktioniert über einen Laser der vom Gerät ausgehend über eine Reflektive Fläche zu diesem zurückgeworfen wird. Der Abstandssensor verfügt über ein Display das den gemessenen Abstand anzeigt und über das Gerät konfigurierbar ist. Er hat ebenfalls eine Betriebsspannung von 9-30V. Der Abstand wird über einen digitalen Ausgang, dem sog. IO-Link ausgegeben. Da mit diesem nur sehr umständlich Umgegangene werden kann, wurde ein zusätzlicher IO-Link Konverter hinzugefügt. Der EIO104 konvertiert die digitale IO-Link Schnittstelle zu einer analogen 4-20mA Schnittstelle mit der einfacher Umgegangen werden kann.

Anemometer

Das WSWD Anemometer wird genutzt, um die Windrichtung und -geschwindigkeit zu messen. Dieses basiert auf einer 24V Versorgungsspannung. Es besitzt ebenfalls ein Heizelement, das die Verwendung bei sehr niedrigen Temperaturen ermöglicht. Dieses wird allerdings im Einsatzszenario und im Prototyp nicht benötigt. Die Schnittstellen des Anemometers sind abhängig von dessen Ausführung. In der verwendeten Ausführung, dem WSWD1, können die Daten neben einer analogen auch über eine Digitale RS485/RS422 Schnittstelle abgefragt werden. Die Werte können Analog entweder als 0/4-20/24mA Strom- oder als 0/2-8/10V Spannungssignal ausgegeben werden. Die Digitale Schnittstelle kann neben der Datenausgabe auch zur Konfiguration des Gerätes genutzt werden und bietet eine Vielzahl an Konfigurationsparametern und Übertragungsprotokollen an.

Druckkraft-Sensor

Der Burster Druckkraft-Sensor war der einzige Sensor der zum Zeitpunkt der Arbeit noch nicht im Prototyp mit eingebaut wurde. Dieser wurde aber dennoch mit eingeplant. Der Druckkraft Sensor kann ebenfalls mit den typischen 24V betrieben werden. Die Messwerte werden hier über eine Analoges 0-10V Spannungssignal übertragen.

Temperatursensor

Es gibt noch keine genaueren Informationen zu einem eventuell zum Einsatz kommenden Temperatursensor. Es wurde dennoch ein zusätzlicher analoger Eingang für eine entsprechende 4-20mA Schnittstelle eingeplant.

3.2.2 Aktoren

Zu den Aktoren zählen die folgenden Komponenten:

- Gleichstrommotor: Dunkermotoren BG 45x30 SI
- Externe Relais

Gleichstrommotor

Der Dunker BG 45x30 SI Gleichstrommotor ist die antreibende Kraft im Aufbau. Da der Motor eine interne Regelung besitzt, sind Leistungs- und Logikversorgung getrennt. Der Motor an sich wird dabei über den Leistungsteil bestromt, während der Logikteil für die Steuerung und die Bereitstellung von Feedback-Daten zuständig ist. Die Betriebsspannung der Leistungs- und Logikversorgung liegt bei 24V, wobei der Leistungsteil einem maximal zulässigen Dauerstrom von 3,8A ausgesetzt sein darf. Die Ansteuerung des Motors findet über vier digitale- und einen analogen Eingang statt. Der Motor stellt Feedback zum aktuellen Status über drei Ausgänge bereit. Diese geben die Drehrichtung, aktuelle Störungen und die Drehgeschwindigkeit des Motors in Impulsen an. Die Bezeichnung und Funktionen dieser Ein-/Ausgänge sind in Tabelle 1/Tabelle 2 dargestellt.

Eingang 1 (IN1)	Eingang 0 (IN0)	Funktion
0	0	Motor aus
0	1	Linkslauf
1	0	Rechtslauf
1	1	Stopp mit Haltemoment
Eingang 3 (IN3)	Eingang 2 (IN2)	
0	0	Drehzahlvorgabe Analog
0	1	Stromvorgabe Analog
1	0	Geschwindigkeit 1
1	1	Geschwindigkeit 2

Tabelle 1: Eingänge und Funktionen des BG 45x30 SI (vgl. [4], S.22)

Name	Wertebereich	Funktion
Analog In	$0-4092 \frac{U}{min}$ 0-10V	Drehzahlvorgabe/ Analogwert
Out 1	$12 \frac{Impulse}{U}$	Aktuelle Drehzahl
Out 2	1 = Störung; 0 = keine Störung	Fehlerangabe
Out 3	1 = Linkslauf; 0 = Rechtslauf	Drehrichtung

Tabelle 2: Analoger Eingang und Digitale Ausgänge des BG 45x30 SI (vgl. [4], S.23)

Externe Relais

Zum Zeitpunkt der Arbeit gab es noch keinen konkreten Verwendungszweck der zwei externen Relais, diese wurden für zusätzliche Funktionalitäten dennoch mit eingeplant und sollten über ein 24V Signal geschaltet werden. Sie könnten z.B zum Auslösen einer Motorbremse genutzt werden.

Controllino

Der Controllino ist der zentrale Microcontroller und verwaltet alle Aktoren im fertigen Sailwind 4.0 Prototyp. Dieser kann nicht zu einen der beiden Kategorien gezählt werden, sollte hier aber dennoch kurz erwähnt werden. Von diesem soll das in der Arbeit zu entwerfende System seine Befehle zur Segelausrichtung empfangen. Der Datenaustausch soll dabei über Ethernet stattfinden, um die Distanz zwischen der Kuppel, in dem sich die Linearführung befindet und Basis in der sich der Controllino befindet, zu überbrücken. Zusätzlich dazu ist es geplant die Platine vom Controllino mit Strom zu versorgen.

3.2.3 Zusammenfassung und Übersicht

Damit sind nun alle externen Komponente im System abgedeckt. Die benötigte Spannungsversorgung sowie alle Schnittstellen sind in Tabelle 3 dargestellt. Zusätzlich wurden die Verbindungen in einem Blockschaltbild in Abbildung 5 zusammengefasst, wobei alle blauen Pfeile die digitalen 24V Ein- und Ausgänge beschreiben.

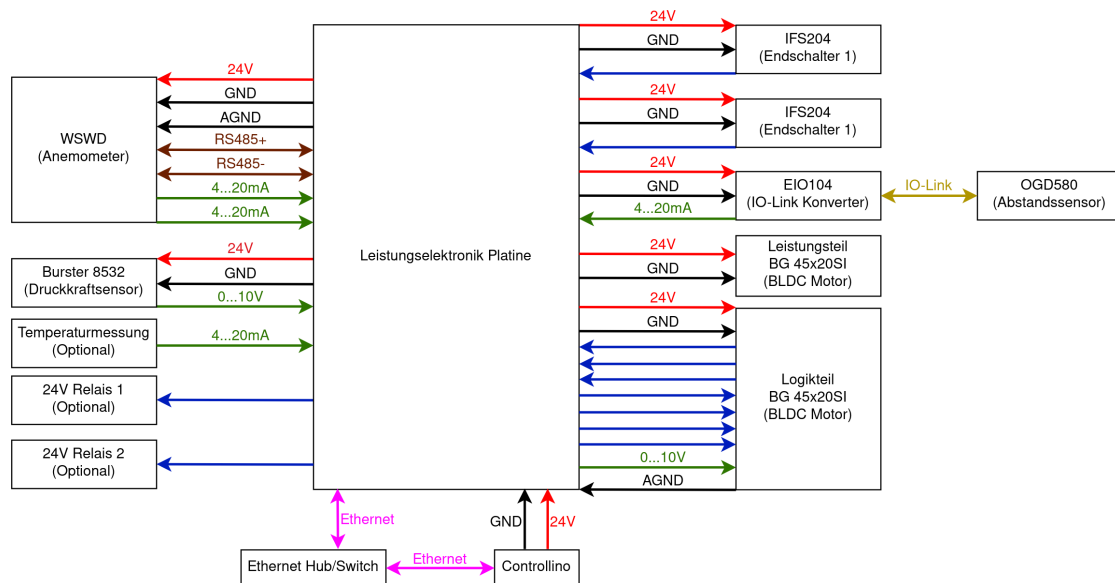


Abbildung 5: Blockschaltbild der benötigten Ein- und Ausgänge der Platine

Element	Pinout											
Endschalter 1	24V	GND	OUT									
Endschalter 2	24V	GND	OUT									
Abstandssensor	24V	GND	IO-Link									
IO-Link Konverter	24V	GND	AOUT									
Anemometer	24V	GND	AOUT	AOUT	AGND	RS485+	RS485-					
Druckkraftsensor	24V	GND	AOUT	AGND								
Motor Leistung	24V	GND										
Motor Logik	24V	GND	IN0	IN1	IN2	IN3	AIN	AGND	OUT1	OUT2	OUT3	
Externes Relais 1	OUT	GND										
Externes Relais 2	OUT	GND										
Temperatur Sensor	AOUT											
Controllino	Ethernet											

Tabelle 3: Übersicht Stromversorgung und Schnittstellen der externen Elemente

3.3 Auswahl der Platinen Bauteile

Da nun klar ist, welche Anforderungen durch die anzuschließenden Geräte bestehen, kann auf Basis dieser weiter verfahren werden. Dabei soll im folgenden ein geeigneter Mikrocontroller, sowie die nötigen Platinenkomponenten ausgewählt

werden, um mit den Sensoren und Aktoren zu kommunizieren und diese mit Strom zu versorgen.

3.3.1 Mikrocontroller

Die Auswahl des Mikrocontrollers wurde auf der Basis folgender Kriterien beschlossen:

- Benötigte Inputs und Outputs
- Softwaresupport
- Entwicklung
- Verfügbarkeit

Dabei gab es die Option, den Chip als einzelnes Element direkt in der selbst erstellten Platine einzubetten oder ein Entwicklungsboard zusammen mit einer Erweiterungsplatine zu nutzen. Schließlich fiel die Entscheidung auf die Verwendung eines Entwicklungsboards, um bereits parallel zum Hardwaredesign eine geeignete Software entwickeln und mit prototypischen Aufbauten testen zu können. Dies bot ebenfalls den Vorteil, dass ein kaputter Chip schnell ersetzt werden kann und mit der Platine ein möglicher Formfaktor für die Erweiterung vorgegeben ist. Außerdem kann durch den integrierten Debugger schneller neue Software getestet und aufgespielt werden. Zuletzt besitzen viele Entwicklungsboards bereits einen Ethernet-Port, der somit nicht mehr extra bestellt werden müsste.

Aufgrund der oben genannten Kriterien und der Entscheidung, ein Entwicklungsboard zu benutzen, fiel die Wahl auf das STM32 NUCLEO F439zi Board. Dieses war bereits an der HTWG Konstanz verfügbar und ist mit dem Softwaresupport der Firma ST, welche insbesondere eine eigene Entwicklungsumgebung anbietet, einfach zu programmieren und aufzusetzen. Außerdem enthalten Entwicklungsboards von ST einen abbrechbaren Debugger, womit das Board auch außerhalb der Entwicklungsphase eingesetzt werden kann.

3.3.2 Relais

Um das Auslösen eines Endschalters an den Mikrocontroller weiterzugeben, werden zwei 24V Relais benötigt, die bei Aktivierung der Spule ein 3,3V Signal an den Eingang des Mikrocontrollers weitergeben. Gleichzeitig soll durch die Relais der Eingang 1 oder 2 des Motors auf Null gesetzt werden. Diese geben die Drehrichtung vor und stoppen den Motor, sobald beide auf null gesetzt sind. Da dieses System unabhängig von der Software des Mikrocontrollers agiert, bietet es eine zusätzliche Sicherheit gegenüber Beschädigungen an der Linearführung aufgrund von Programm-Fehlfunktionen. Hierfür wurden die Omron G6S-2F 24DC Relais ausgewählt. Diese besitzen zwei schaltbare Ausgänge und weisen einen kompakten Formfaktor auf.

3.3.3 Optokoppler

Um die restlichen 24V Ein- und Ausgänge zu steuern oder auszulesen, kommen Optokoppler zum Einsatz. Diese isolieren ähnlich zu Relais den 24V und 3,3V Schaltkreis. Davon werden insgesamt neun benötigt, um die digitalen Ein- und Ausgänge des Motors und die beiden externen Relais zu schalten. Zusätzlich sollte wie bei den Relais wieder ein kleiner Formfaktor vorliegen. Hierfür wurden die LTV-817S ausgesucht. Deren typische Schaltzeit von $3\text{-}18\mu\text{s}$ ist für die Anwendung ausreichend und durch den Formfaktor lassen sich die Optokoppler gut auf der Platine unterbringen.

3.3.4 RS485 zu UART Konverter

Um das Anemometer über die RS485 Schnittstelle zu konfigurieren und Daten abzufragen wird ein zusätzlicher Baustein benötigt. Dieser konvertiert das differenzielle RS485-Signal zu einem nicht differenziellen Universal Asynchronous Receiver/Transmitter (UART)-Signal, welches der Mikrocontroller verarbeiten kann. Hierfür wurde der MAX3485CSA+CT-ND ausgesucht, da dieser mit einer 3,3V Versorgungsspannung betrieben werden kann und eine Datenübertragungsrate von bis zu $10\frac{\text{MB}}{\text{s}}$ unterstützt.

3.3.5 FRAM

Da die neutrale Position des Segels dauerhaft gespeichert werden soll, wird ein nicht flüchtiger Speicherstein benötigt. Dabei ist Ferroelectric Random Access Memory (FRAM) eine der billigsten Speichermöglichkeiten für diesen Zweck. Es wurde die geringste Speichergröße gewählt, da die gespeicherten Daten jederzeit, wie bei einem EEPROM Speicher, überschrieben werden können. Hier wurde der FM25L16B-GTR gewählt, welcher mit einer Speicherkapazität von 16Kbit ausgestattet ist und damit für das Speichern der Positionsdaten geeignet ist. Da er über eine Serial Peripheral Interface (SPI) Schnittstelle angesteuert wird, sind schnelle Schreib- und Lesegeschwindigkeiten möglich, was für das regelmäßige Aktualisieren der Positionsdaten erforderlich ist. Ebenfalls ist die Lebenserwartung des Speichers von 85 Jahren unter Betrieb mit einer Taktfrequenz von 20Mhz mehr als ausreichend.

3.3.6 Operations Verstärker

Um die Drehzahl des Motors vorgeben zu können, wird ein analoges Signal im Bereich von 0-10V benötigt. Da der Digital Analog Converter (DAC) des Mikrocontrollers bei einer Versorgungsspannung von 5V maximal ein analoges Signal von 3,3V ausgeben kann [5], wird ein Operationsverstärker benötigt. Dieser soll mit einer 24V Offsetspannung das Ausgangssignal auf 10V skalieren. Hier gab es bis auf die Maximalspannung keine besonderen Anforderungen an den Operationsverstärker und somit wurde der weitverbreitete LM2904QS ausgewählt.

3.3.7 DC/DC-Wandler

Um alle Komponenten zu versorgen, bedarf es insgesamt drei verschiedenen Spannungspotenzialen: 24V, 5V und 3,3V. Dabei werden alle externen Steuerelemente direkt mit 24V versorgt. Der Mikrocontroller benötigt eine 5V Spannung, damit dieser auch ohne Stromversorgung des Debuggers betrieben werden kann. Hierfür wird ein Step-Down Konverter genutzt, der die 24V Versorgungsspannung auf 5V herunterbricht. Die 3,3V müssen zusätzlich mit einem Festspannungsregler auf

3,3V-Level konvertiert werden, um die übrigen Bauteile zu versorgen. Bei der Auswahl wurde darauf geachtet, dass der Konverter einen ausreichenden Strom zu Verfügung stellt, ohne dabei dauerhaft unter Vollast zu laufen. Insbesondere muss hier der Verbrauch des Mikrocontrollers von bis zu 120mA (vgl. [6] S.103) einge-rechnet werden. Unter der Annahme, dass die restlichen Bauteile nicht mehr als zusätzliche 380mA verbrauchen, wurde der R-78E5.0-0.5 ausgewählt. Dieser stellt bis zu 0,5A zur Verfügung und kann bei Bedarf durch eine 1A Version ausgetauscht werden. Als Festspannungsregler wurde der LM3940IMP-3.3 gewählt. Dieser kann Eingangsspannung von bis zu 7.5V in eine Ausgangsspannung von 3,3V umwan-deln und hat dabei einen Durchsatz Strom von bis zu 1A. Damit genügt er den Anforderungen.

3.3.8 Stromsensor

Zur Überwachung des Motor Stroms kommt ein Hall-Stromsensor zum Einsatz. Dieser ist im Gegensatz zu einem resistiven Sensor genauer und isoliert den 24V und 3,3V Schaltkreis wie z.B im Schaltplan in Abbildung 6 zusehen ist.

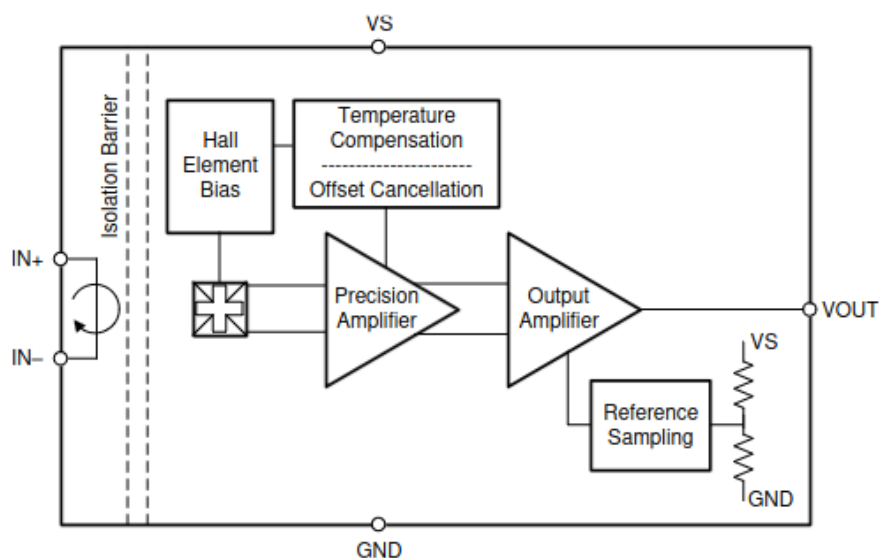


Abbildung 6: Schaltplan des Texas Instruments TMCS1101 Hall Stromsensor (vgl. [7], S.24)

Er sollte in der Lage sein, den maximal zulässigen Motorstrom messen zu können. Da die meisten Hall Sensoren einen deutlich größeren Messbereich unterstützen, wurde der TMCS1101A3BQDRQ1 gewählt. Dieser kann einen maximalen Strom von $\pm 11,25\text{A}$ messen. Der gemessene Strom wird dabei als analoges 0-3,3V Signal ausgegeben und ist damit für diese Anwendung geeignet.

3.3.9 Konnektoren

Die externen Komponenten sollen über Schraubklemmen direkt an die Platine angeschlossen werden. Je nach Querschnitt der Kabel wurde ein größeres oder kleineres Rastermaß gewählt, um Platz auf der Platine einzusparen. Dabei wurde ein Standard Rastermaß von 2,54mm festgelegt und für die Kabel mit größerem Durchmesser ein Rastermaß von 3,5mm.

3.4 Gehäusebauteile

Um die Platine, welche sich später in der Kuppel befinden soll, vor z.B Staub und Wasser zu schützen, soll diese in einem Gehäuse untergebracht werden. Das Gehäuse sollte eine Möglichkeit bieten, die Platine so zu platzieren und zu befestigen, dass im Innenraum noch genügend Platz für Kabelanschlüsse ist. Außerdem muss die Oberfläche des Gehäuses groß genug sein, um Bedienelemente und Status LEDs darauf komfortabel verteilen zu können. Den Anforderungen zufolge wurde das Modell Hammond Manufacturing RP1465 gewählt.

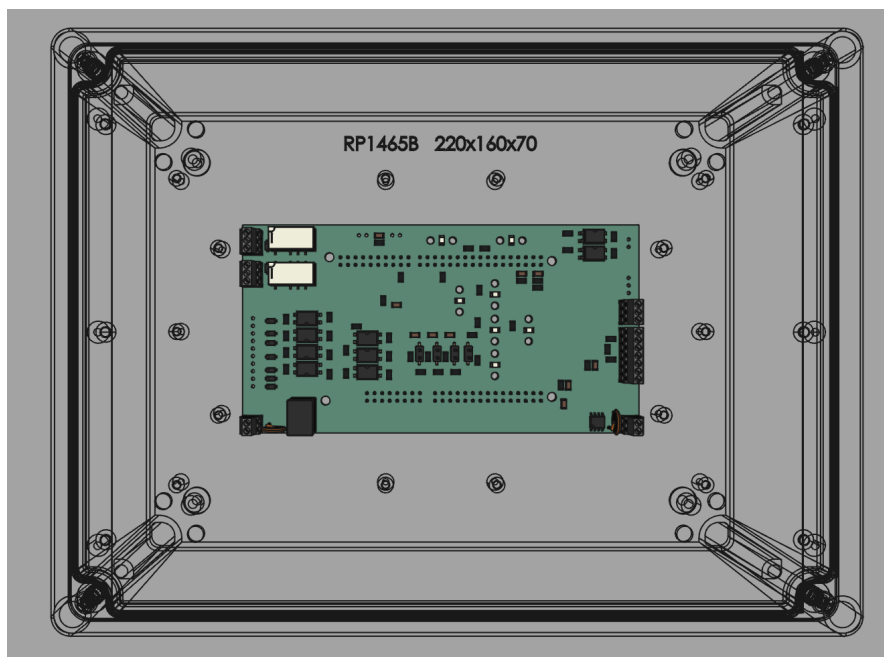


Abbildung 7: 3D Modell der Platine innerhalb des Gehäuses

Wie in Abbildung 7 zusehen ist, bietet dieses genügend Platz für die Platine und alle Kabeldurchführungen. Es ermöglicht ebenfalls den Einsatz eines zweiten Bodens auf dem die Platine mit einer 3D gedruckten Befestigung angeschraubt werden können.

3.4.1 Bedienoberfläche

Die Bedienoberfläche besteht aus einer Reihe von Kippschaltern, Knöpfen und Light Emitting Diode (LED)s. Diese sind in Abbildung 8 dargestellt. Mit dem rechten Knopf soll zur Initialisierung des Systems eine automatische Kalibrierungsfahrt eingeleitet werden. Zusätzlich kann der Benutzer anschließend über die Trimm- und Rollknöpfe die Grenze zwischen diesen Bereichen (siehe Abbildung 3) selbst anfahren und mit erneuter Betätigung des rechten Knopfes festlegen bzw. speichern. Danach kann mittels des Kippschalters in den Automatikbetrieb gewechselt wer-

den, in dem die Steuerung vom Controllino übernommen und das Betätigen der Knöpfe folglich ignoriert wird. Eine Reihe von LEDs gibt Feedback über den aktuellen Zustand der Linearführung. Dabei wird konstant der aktuelle Betriebsmodus durch zwei LEDs angegeben. Ebenfalls wird die Position relativ zur festgelegten neutralen Position durch zwei gelbe LEDs angegeben. Vom System selbst identifizierte Störungen werden durch eine rote LED signalisiert.

Die Knöpfe und Kippschalter sind jeweils mit Dichtungsringen spritzwassergeschützt am Gehäuse montiert. Die Kabel werden über Dupont-Stecker an der Platine angeschlossen. Im Gegensatz dazu sind die LEDs direkt auf der Platine platziert, wobei deren Strahlung durch flexible Lichtleiter zur Außenseite des Gehäuses geführt werden.

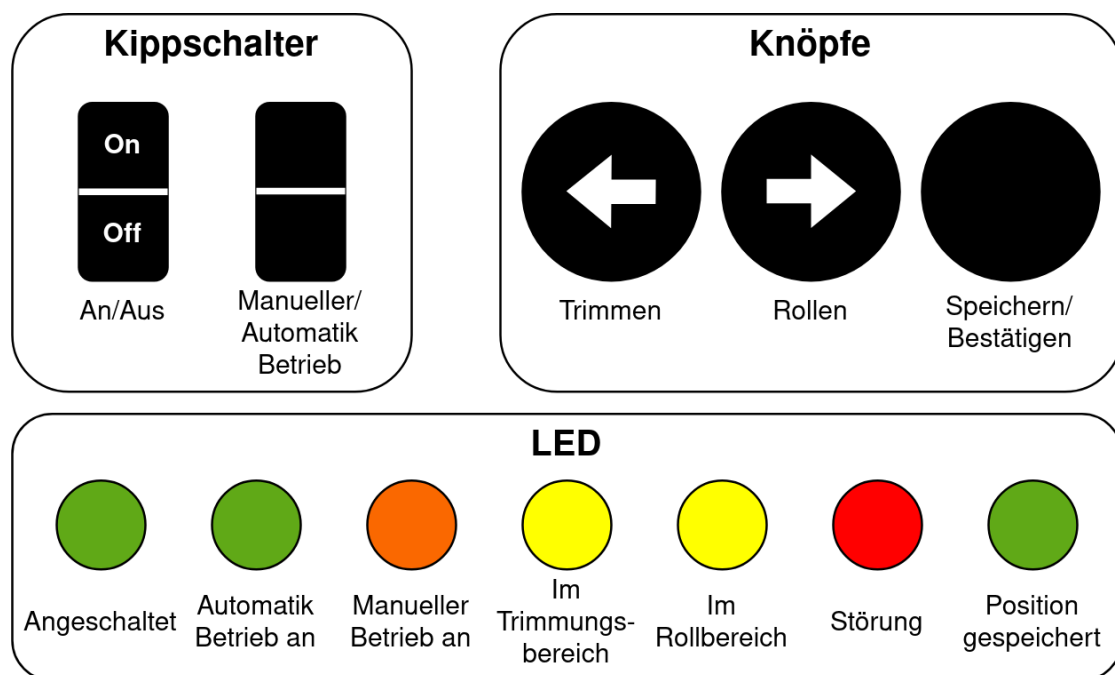


Abbildung 8: Bedienoberfläche der Segel Steuerung

3.4.2 Kabelverschraubungen

Um die benötigten Kabel zur Platine zuführen, wurden Kabelverschraubungen in die Seiten des Gehäuses eingelassen. In diese können die isolierten Kabel eingeführt und die Adernkabel an die richtige Position innerhalb des Gehäuses verlegt werden. Die Kabelverschraubungen sorgen nebenbei für eine Zugentlastung der Kabel.

3.5 Platinen Schaltplan

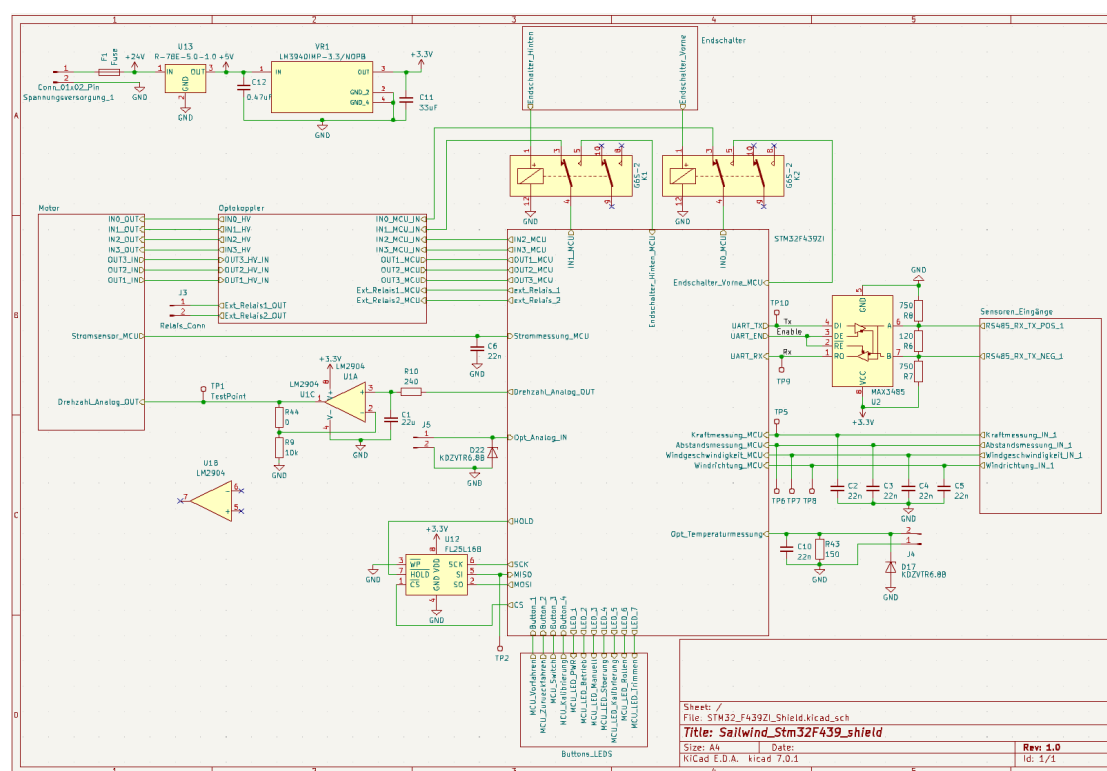


Abbildung 9: Gesamtschaltplan der Platine

Eine Übersicht des Schaltplans ist in Abbildung 9 zusehen. Der komplette Schaltplan ist als KiCad Datei in der ZIP-Datei enthalten und ebenfalls in Anhang A hinterlegt. Der Schaltplan ist in unterschiedliche Bereiche eingeteilt, welche die Gruppierung der Komponenten wiedergeben. Dabei ist die STM32-Gruppe zentral

positioniert und alle Bauteile und Komponenten um diese herum verteilt. Bauteile, die nur einfach verbaut sind wie z.B der FL25L16B FRAM (Unten Links) wurden nicht in Gruppen eingeteilt, sondern direkt in der Gesamtansicht positioniert. Oben links befindet sich die Spannungsversorgung der Platine und die Direct Current (DC)/DC Wandler. Oberhalb des STM32 befinden sich die beiden Relais, welche mit den Endschaltern verbunden sind. Links außen befindet sich der Brushless DC (BLDC) Motor, welcher mit dem Operationsverstärker und den Optokopplern verbunden ist. Rechts befinden sich mit Ausnahme des Stromsensors alle analogen Sensoren. Dort ist ebenfalls der MAX3485 UART Konvertierer abgebildet. Unterhalb des Mikrocontrollers befinden sich die Knöpfe und LEDS, die das **HMI!** (**HMI!**) bilden. Im folgenden werden exemplarisch die Schaltungen für diese Gruppen genauer betrachtet.

3.5.1 STM32F439zi Gruppe

Der STM32F439zi enthält alle Pins, auf denen später die Erweiterungsplatine auf sitzt. Hier wurden alle benötigten Pins nach außen zur Schaltplanübersicht geführt.

3.5.2 Motor Gruppe

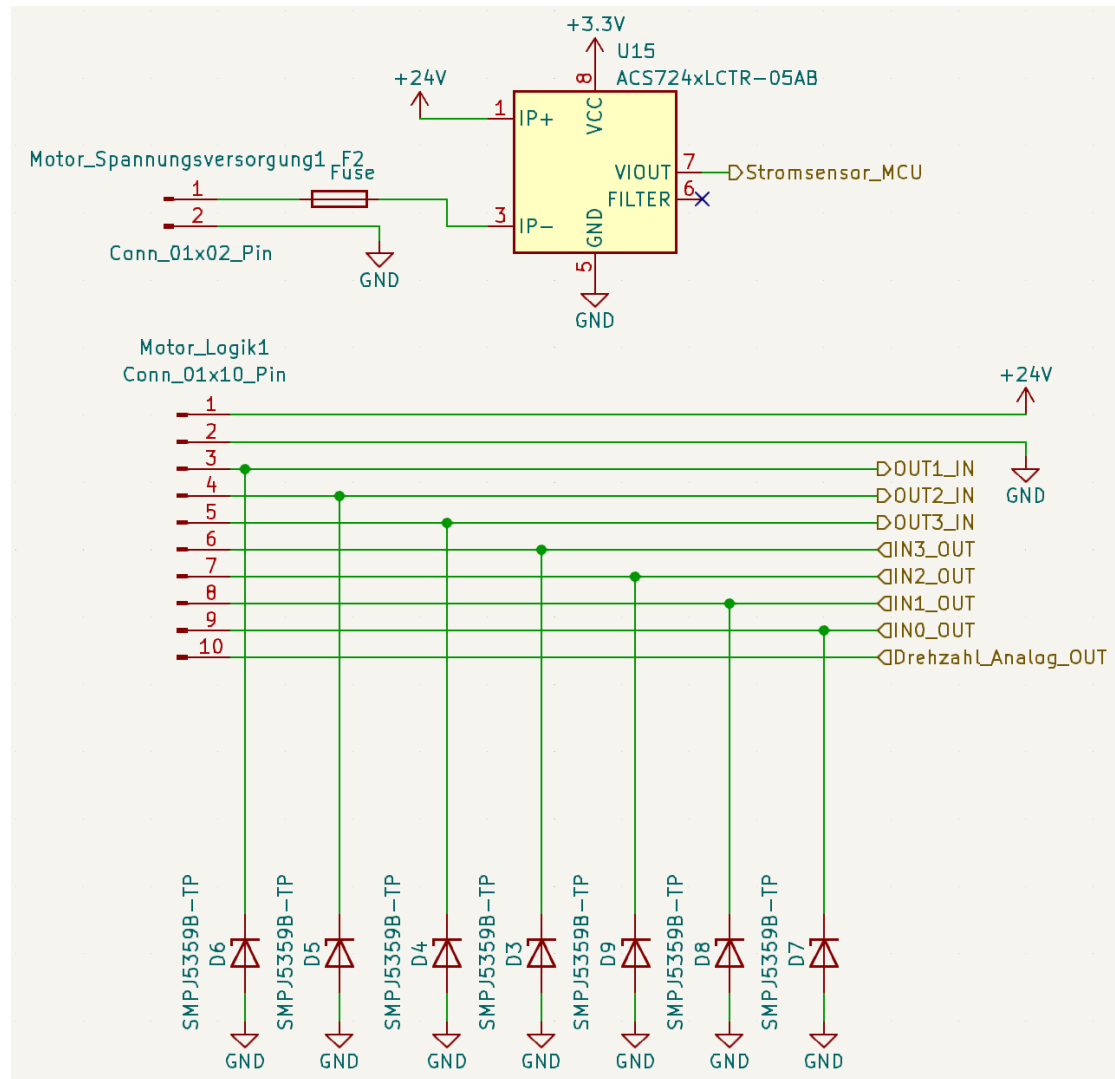


Abbildung 10: Schaltplan der Motorgruppe

In der Motorgruppe ist neben den Ein- und Ausgängen des BG 45x30 SI auch der Hall-Stromsensor abgebildet. Alle digitalen Ein- und Ausgänge auch außerhalb der Motorgruppe sind durch 24V Zener Dioden verschaltet, die als Überspannungsschutz agieren. Die Spannungsversorgungen der Platine und des Motors sind zusätzlich

durch Überstromsicherungen geschützt, wie anhand des Stromsensors in Abbildung 10 zu sehen ist.

3.5.3 Optokoppler Gruppe

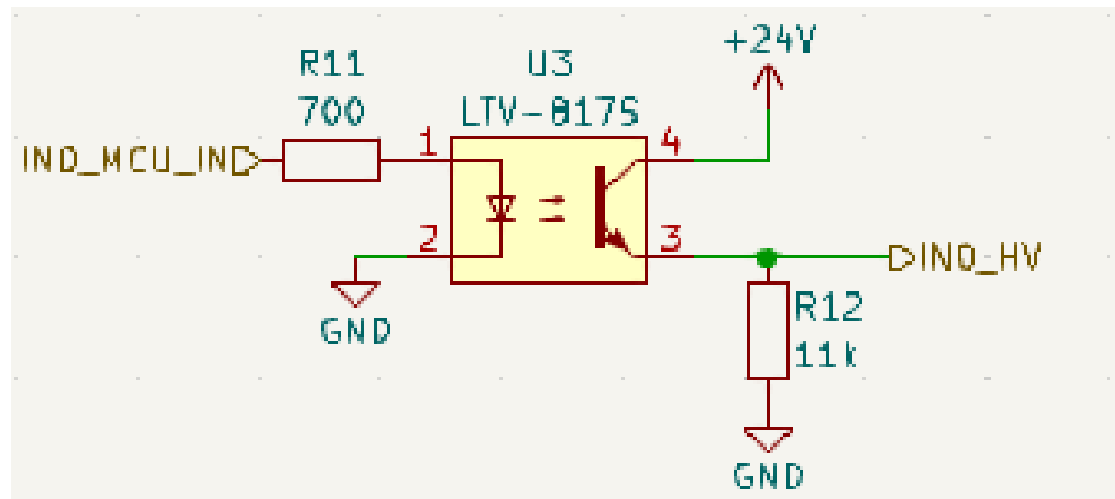


Abbildung 11: Beispielhafte Schaltung eines Optokopplers

Alle Optokopplerschaltungen der Gruppe sind gleich aufgebaut. Eine beispielhafte Schaltung ist deswegen in Abbildung 11 dargestellt. Der Eingangswiderstand (links) begrenzt den Strom, welcher in der verbauten Infrarot-LED des Optokopplers den Stromfluss auf der Ausgangsseite bestimmt. Ein Pulldown-Widerstand auf der Ausgangsseite begrenzt den maximal erlaubten Strom zum Mikrocontroller und setzt den Eingang auf ein festgelegtes Potenzial.

3.5.4 Sensoren Gruppe

In der Sensoren Gruppe sind die analogen Sensoren abgebildet. Hier ist wie beispielhaft für den Abstandssensor in Abbildung 12 gut zu erkennen, wie die Stromsignale in Spannungssignale von einem Widerstand umgewandelt werden. Für die 4-20mA Signale wurde dabei ein 150Ω Widerstand gewählt, da damit bei einem

maximalen Strom von 20mA annähernd 3,3V erreicht werden:

$$U = 20mA \times 150\Omega = 3V \quad (1)$$

Zum Überspannungsschutz ist hier ebenfalls eine Zener Diode mit einer Durchbruchspannung von 6,8V angebracht. Das Spannungssignal von dem Druckkraftsensor wird durch einen Spannungsteiler ebenfalls auf 0-3V skaliert:

$$U_{Ausgang} = \frac{10V}{2300\Omega + 1000\Omega} \times 1000\Omega = 3,03V \quad (2)$$

Um die analogen Signale stabiler zu halten und niedrige Frequenzen zu entfernen, wurden für jedes analoge Signal zusätzlich Keramikpufferkondensatoren mit einer Kapazität von 22nF hinzugefügt.

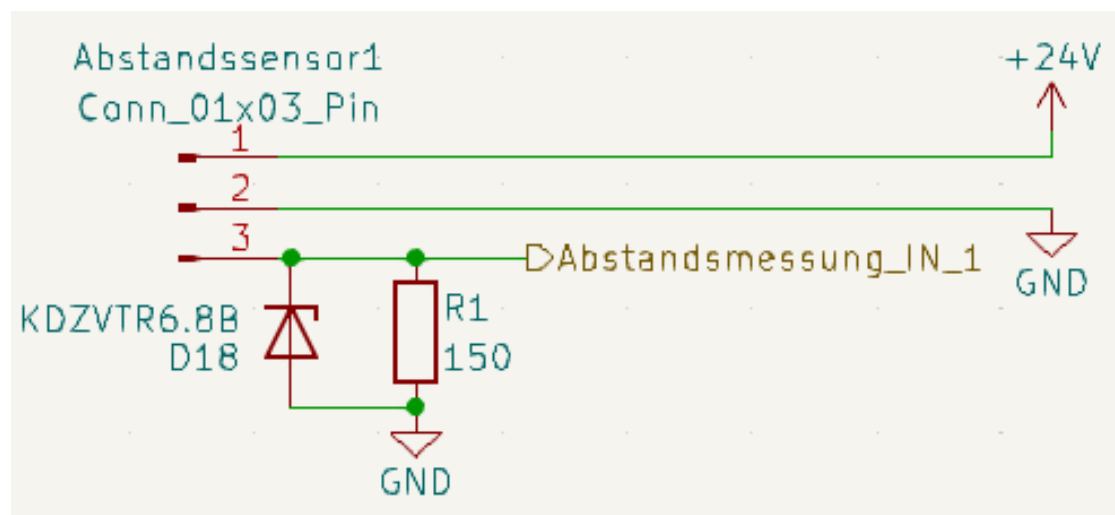


Abbildung 12: Schaltplan des analogen Eingangs des Abstandssensors

3.5.5 Buttons und LEDs Gruppe

Damit der Effekt des Debouncing nicht über die Software ausgeregelt werden muss, wurde für die Buttons eine entsprechende Schaltung integriert. Hierbei wurde ein RC-Filter, wie in Abbildung 13 zusehen, genutzt. Beim Drücken des Knopfes wird

der Kondensator zuerst entladen, was zu einer langsam abfallend Spannung führt. Wird der Knopf losgelassen, lädt sich dieser wieder auf. Somit werden jegliche unerwünschte Spannungssprünge vollständig unterdrückt. Da die Debounce-Zeit der Knöpfe typischerweise bei 0,1ms liegt (vgl. [8], S.2), wurde eine Zeitkonstante $\tau = 10\text{ms}$ für den RC Filter angenommen. Die LEDs wurden je nach zulässigem Strom mit einem entsprechenden Widerstand versehen.

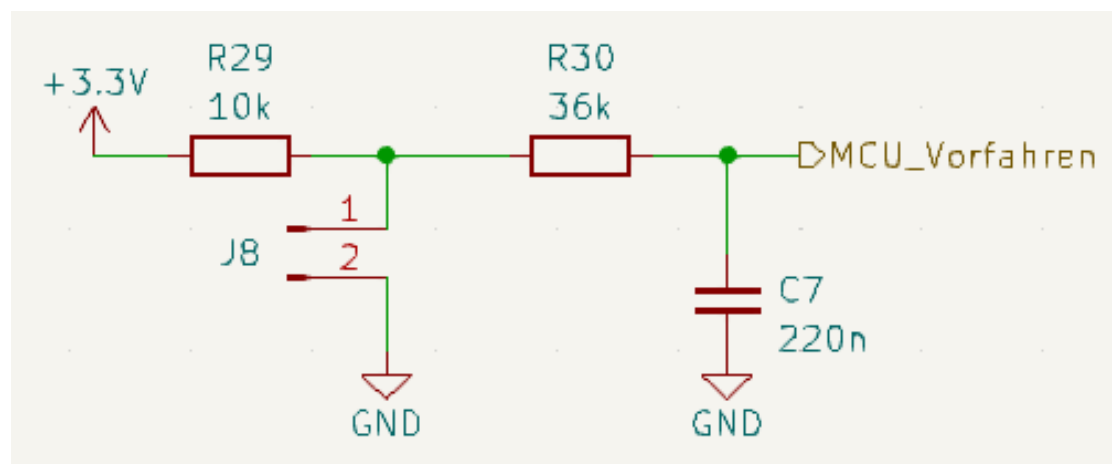


Abbildung 13: RC-Filter für einen am Gehäuse angebrachten Knopf

3.6 PCB

Das fertige PCB ist als 3D Modell in Abbildung 14 dargestellt. Beim Entwurf wurde darauf geachtet, die 24V- von den 3,3V-Komponenten zu trennen, um die Beeinflussung analoger Signale durch große Ströme oder höhere Frequenzen so weit wie möglich zu reduzieren. Aus diesem Grund befinden sich die 24V-Komponenten am Rand der Platine, während die 3,3V-Bauteile weitestgehend im Inneren der Platine positioniert sind. Die Platine sitzt mit vier unterschiedlichen Pin-Reihen am Entwicklungsboard auf und führt diese auf die Oberseite. Die Spurbreiten auf der Platine wurden aufgrund von Stromflüssen, Spannungen und Längen bestimmt. Da der Platinenhersteller eine minimale Spurbreite von 0,2mm vorgibt, reichte diese für fast alle Spuren aus. Eine Ausnahme bildet jene, welche dem Motorstrom von

bis zu 4A ausgesetzt ist und dementsprechend breiter ausgelegt werden musste.

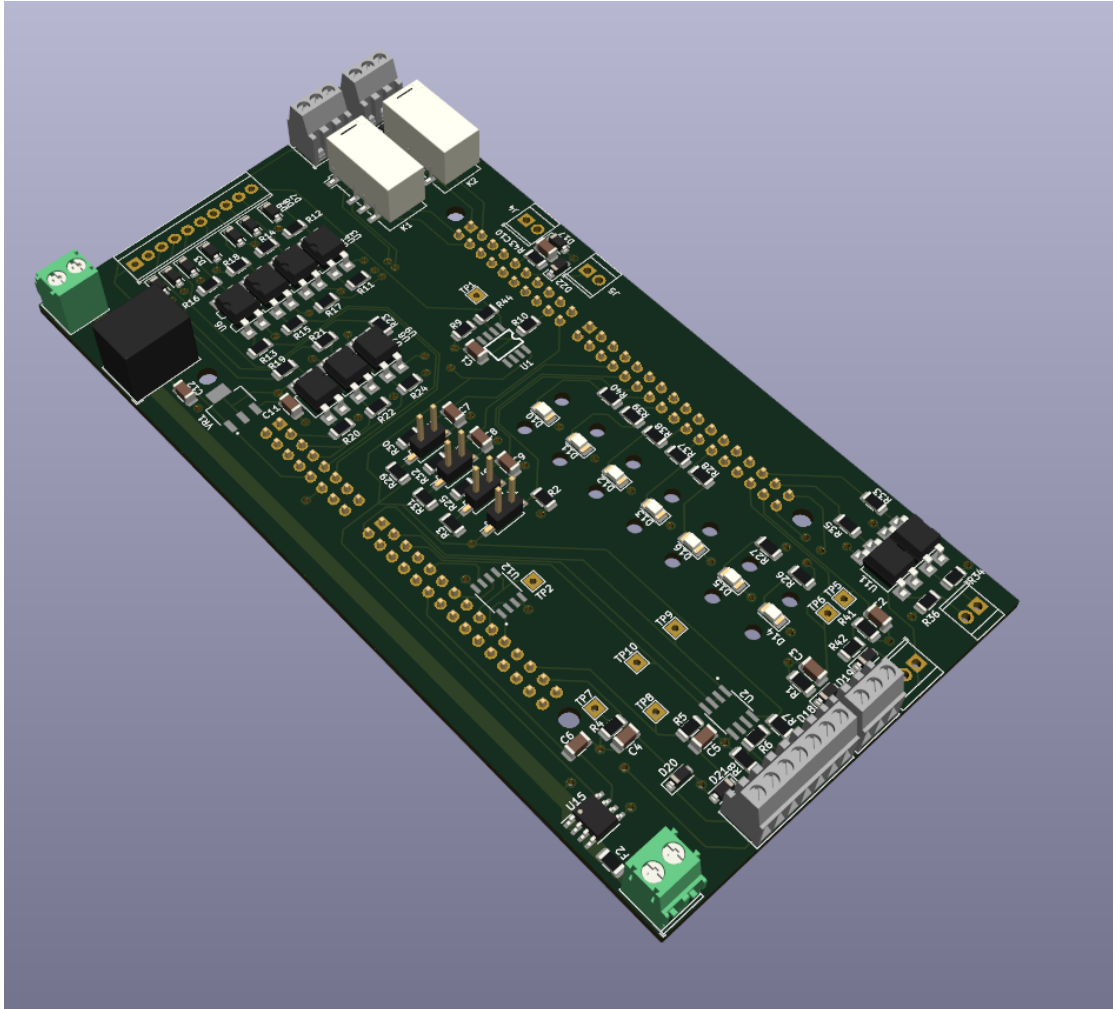


Abbildung 14: 3D Ansicht des fertigen PCBs

Der Formfaktor der Platine wurde an dem Standard Layout eines ST Nucleo 144 Boards orientiert. Aufgrund des damit verfügbaren Platzes wurden ausschließlich Surface-Mount Device (SMD) Bauteile der Größe 1206 verwendet. Diese sind groß genug, um sie unter angemessenem Aufwand von Hand zu löten und andererseits klein genug, um geeignete Abstände bei der Positionierung einhalten zu können. Eine Liste aller verwendeter Platinen- und Gehäusebauteile ist in An-

hang B zu finden.

3.7 Probleme

Während der Nutzung und Bestückung des PCBs sind einige Probleme aufgetreten, die auf das Design oder falsche Berechnungen zurückzuführen sind. Diese sollen im folgenden erklärt und korrigiert werden.

3.7.1 Schraubklemmen Löcher

Der erste Fehler, welcher zu Beginn auffiel, waren die Lochdurchmesser für die Schraubklemmen. Diese waren zwar groß genug dimensioniert, um die Pins der Schraubklemmen durchzuführen, jedoch fehlte damit der Platz, um Nieten zur elektrischen Verbindung zwischen den Pins und der Platine einzufügen. Die Löcher waren ebenfalls zu klein, um mit Drähten eine Verbindung zwischen dem Pin und den Spuren herzustellen. Dies führte zu vielen elektrischen Kontaktverlusten und minderte die horizontale Stabilität der Schraubklemmen. Die Kontaktverluste führten wiederum zur mangelnden Präzision der analogen Eingänge (siehe Unterabschnitt 3.7.6). Da die Löcher im Nachhinein nicht mehr vergrößert werden konnten, gab es hierfür leider keine Möglichkeit, diesen Fehler auszubessern.

3.7.2 Operationsverstärker

Der Operationsverstärker war zwar als Verstärkerschaltung richtig konfiguriert, allerdings musste die Verstärkerspannung von 3,3V auf 24V angehoben werden, um die 10V zu erreichen. Zusätzlich wurde der Spannungsteiler am Ausgang des OPVs entfernt und der erste Widerstand gegen einen Jumper getauscht.

3.7.3 RS485 Bias Widerstände

Die RS485 Widerstände wurden anfänglich nach den empfohlenen Werten in der WSWD Dokumentation dimensioniert. Hier wurde bei einer Versorgungsspannung von 5V ein Busabschlusswiderstand R_t von 120Ω und zwei Bias-Widerstände R_b

von 750Ω empfohlen (vgl.[9] S.21). Dies sorgte allerdings für eine zu große Differenz der beiden Spannungen und der MAX3485 konnte nicht zwischen HIGH und LOW unterscheiden. Deshalb wurde stattdessen nach der Anleitung des MAX3485 die Differenz zwischen beiden Potenzialen auf 0,2V begrenzt. Hierfür wurde auf die Berechnung in dem von Texas Instruments veröffentlichten Bericht: „RS-485 failsafe for an idle bus“ zurückgegriffen (vgl. [10]). Mithilfe der Formel:

$$R_b = \left(\frac{V_s}{V_{diff}} + 1 \right) \times 27,8\Omega \quad (3)$$

lassen sich die beiden Bias Widerstände für das RS-485 Netzwerk berechnen. Aus den eingesetzten Werten resultiert:

$$R_b = \left(\frac{3,3V}{0,2V} + 1 \right) \times 27,8\Omega = 486,5\Omega \quad (4)$$

Der nächstkleinere Widerstand der verfügbaren E24-Reihe wurde mit 470Ω gewählt. Damit konnte schließlich erfolgreich über RS485 kommuniziert werden.

3.7.4 FRAM Anschlüsse

Die SPI-Anschlüsse des FRAM Speichers wurden falsch verbunden. Dabei wurden die Master Out Slave In (MOSI)- und Master In Slave Out (MISO)-Leitung vertauscht und der Write Protect (WP)-Pin auf das Ground-Potenzial gelegt. Dies führte dazu, dass der Speicher nicht adressiert werden konnte und folglich nicht auf gesendete Befehle reagierte. Zuerst wurde der WP von dem Ground Potenzial entfernt und schwebend gehalten. Durch das Beobachten der Pegel von MISO-, MOSI- und Clock-Leitung konnte daraufhin erkannt werden, dass diese vertauscht wurden. Die PCB-Spur wurde dementsprechend getrennt und die Pins stattdessen mit Kabeln verbunden.

3.7.5 Abstandssensor Präzision

Der Abstandssensor war eine weitere, große Fehlerquelle. Der Abstandssensor hat zwar eine sehr hohe Präzision erreicht, allerdings erst nach einer Aufwärmzeit. Dieser Effekt tritt laut Datenblatt erst unter einer Umgebungstemperatur von -10°C auf (vgl. [11] S.2). Jedoch wurde in mehreren Tests festgestellt, dass bei Stillstand das Display des Sensors mit der Zeit sinkende Werte anzeigt. Um die Ursache dafür zu finden, wurden einige Messreihen durchgeführt. Hierbei wurde der Abstand zum Sensor konstant gehalten und in einem Intervall von 30s der Spannungswert an einem Messpunkt der Platine mit einem Oszilloskop gemessen. Dies wurde für inkrementierende Außerbetriebszeiten wiederholt. Die Messreihe ist in Abbildung 15 dargestellt. Hier ist klar erkennbar, dass erst nach ca. 1400s bzw. 20-25min ein gleichbleibender Spannungswert auslesbar ist. Ebenfalls geht daraus hervor, dass die Präzision mit der Temperatur des Sensors zusammenhängt, da nach kürzeren Außerbetriebszeiten die Differenz sinkt.

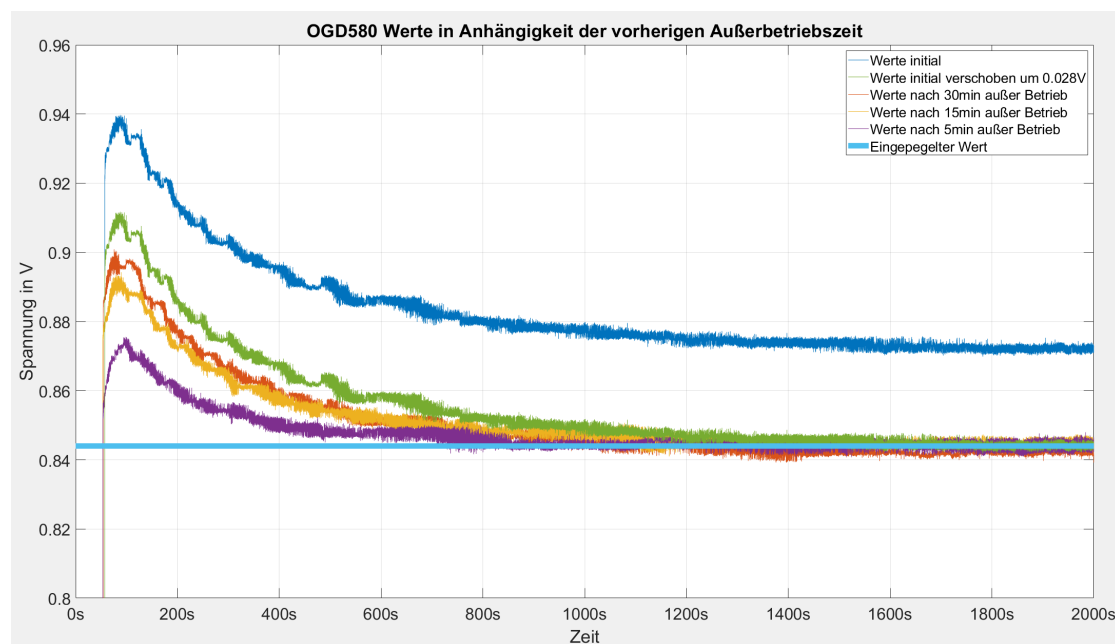


Abbildung 15: Messreihen des ausgelesenen Spannungswert bei gleichbleibendem Abstand und steigenden Außerbetriebszeiten

Das Problem blieb allerdings selbst nach einer Aufwärmzeit von 20min. bestehen. Die Abweichungen der ausgelesenen Werte über die Messpunkte von den Werten am Display lagen teilweise bei 25mm. Um die Ursache dieses Problems zu finden, wurden weitere Messreihen aufgenommen. Dabei wurde der Schlitten der Linearführung in 20mm Abständen verschoben. An jedem dieser Punkte wurden folgende Daten notiert:

- Abstand zum Sensor, gemessen mit einem Meterstab in mm
- Angezeigter Abstand auf dem Display des Sensors in mm
- Ausgegebener Strom des IO-Link Konverters gemessen mit einem Multimeter in mA
- Spannung am Testpunkt der Platine mit einem Oszilloskop in V

Dies wurde in 5 Messreihen wiederholt, um die Reproduzierbarkeit zu verifizieren. Die aufgezeichneten Daten wurden in mm umgerechnet:

$$L_{Strom} = \frac{L_{max} - L_{min}}{I_{max} - I_{min}} \times \left(\frac{I_{Position}}{1000} - I_{min} \right) + L_{min} \quad (5)$$

$$L_{Spannung} = \frac{L_{max} - L_{min}}{I_{max} - I_{min}} \times \left(\frac{U_{Position}}{153,5\Omega} - I_{min} \right) + L_{min} \quad (6)$$

und oberhalb der mit dem Meterstab gemessenen Werte geplottet. Eine dieser Messreihen ist in Abbildung 16 dargestellt.

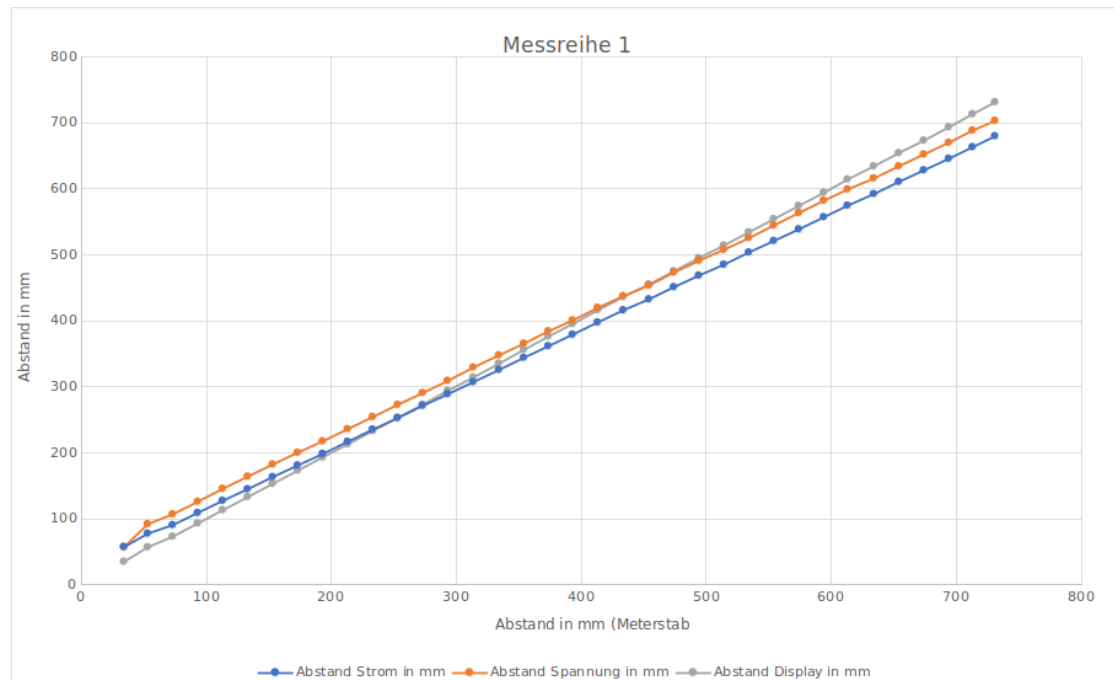


Abbildung 16: Präzision des Abstandssensors anhand verschiedener Messwerte

Hier ist zu erkennen, dass der auf dem Display abgelesene mit dem gemessenen Abstand übereinstimmt. Allerdings weichen sowohl der Strom- als auch der Spannungswert ab, da die Steigung beider Messreihen nicht gleich zu der des Abstandssensors ist. Dieser ermöglicht die Anpassung der Skalierung. Dabei kann z.B. festgelegt werden, ab welchem Messwert das Ausgangssignal 4mA/20mA betragen soll (vgl. [12], S.6). Da das Display den korrekten Wert anzeigt, aber ein falscher Strom Wert ausgegeben wird, kann davon ausgegangen werden, dass der Konverter falsch eingestellt ist. Dies konnte im Zuge dieser Arbeit nicht behoben werden, da ein entsprechendes IO-Link Master Gerät nicht verfügbar war.

3.7.6 Präzision der Analogen Signale

Neben den Problemen mit dem Abstandssensor wurden ebenfalls leicht abweichende Werte in den anderen Analogsignalen festgestellt. Beginnend mit den Stromsignalen, kann der ausgelesene Wert mit der Präzision des Widerstandes zusam-

menhängen, da diese mit 1% Genauigkeit immer noch große Temperaturabweichungen aufweisen können. Die schlechten Kontakte tragen ebenfalls zur Ungenauigkeit bei. Speziell beim Stromsensor kann auch hier die Positionierung eine Fehlerquelle sein, da die verbundenen Spuren für optimale Präzision im 45° Winkel zu den Sensorpins stehen sollten (vgl. [7], S.37). Um die Präzision zu verbessern, können zusätzlich zu den Widerständen Ferrite eingesetzt werden. Diese verhalten sich ähnlich wie Spulen und filtern hochfrequente Signale. In Kombination mit den bereits verwendeten Kondensatoren würden diese einen LC-Filter darstellen und einen Bandpass erzeugen, der zusätzlich die Signalqualität verbessert.

4 Software

Die Steuerung der Segel soll durch zwei verteilte Systeme erfolgen, die über eine Ethernet-Verbindung miteinander kommunizieren. Die Idee dahinter ist, die Berechnungen für die Ausrichtung der Segel in Abhängigkeit der Windstärke und -Richtung (Controllino) von der hardwarenahen Verarbeitung der Sensoren und Aktuatoren (STM32) logisch zu trennen.

Die vorliegende Arbeit konzentriert sich dabei lediglich auf das System der Sensoren und Aktuatoren, welches mit dem STM32 realisiert wurde. Die Aufgabe dieses Systems ist, zunächst die Linearführung zu kalibrieren, damit die aktuelle Position korrekt ermittelt werden kann. Außerdem soll über die Buttons am Gehäuse und zusätzlich über einen Webserver eine manuelle Steuerung ermöglicht werden. Während des Automatikbetriebs geschieht ein regelmäßiger Austausch aller relevanter Daten über eine REST basierte Schnittstelle. Dies beinhaltet u.A. die Information über aktuelle Windbedingung, Position der Segel, eventuelle Fehlerzustände (z.B. Motorfehler oder Überstrom) und daraus resultierende Befehle zur Anpassung der Segelstellung. Für die Implementierung wurde die Software in komponentenorientierte Module eingeteilt:

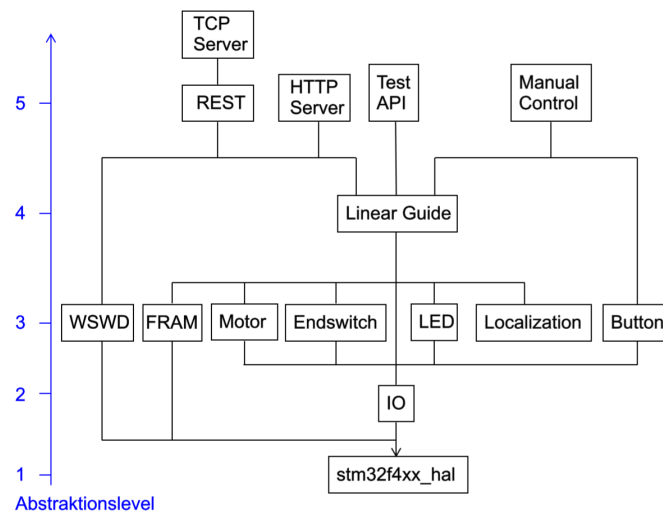


Abbildung 17: Software Modulstruktur

Abbildung 17 zeigt abwärtsgerichtet, wie die einzelnen Module aufeinander zugreifen. Auf unterster Abstraktionsebene laufen jegliche Operationen über die Standardbibliothek des Mikrocontrollers `stm32f4xx_hal`. Darüber liefert das `IO`-Modul ein Set aus Hilfsfunktionen und -Strukturen für den allgemeinen Zugriff auf die General-Purpose Input/Output (GPIO)-Pins und zum Auslesen analoger Messwerte der einzelnen Sensoren.

Ebene drei umfasst hauptsächlich Module, welche alle relevanten Funktionalitäten der physischen Teilkomponenten des Systems implementieren, wie z.B. das Anemometer `WSWD` oder der `Motor`. Einige davon greifen dabei auf das `IO`-Modul zu, wobei der allgemeine GPIO-Zugriff von einer spezifischen Funktion eingekapselt wird, wie z.B. das Einschalten einer LED im Falle des `LED`-Moduls. Eine Ausnahme ist das `Localization`-Modul, welches keine physische Komponente darstellt, sondern einige Hilfsfunktionen zur Kalibrierung und Positionsberechnung bereitstellt.

Während die Module bis Ebene drei überwiegend allgemeingültig entworfen sind, enthält das `Linear Guide`-Modul anwendungsspezifische Funktionen. Als zentrales

Element bildet dieses ein High-Level Interface zur Verwendung der Teilkomponenten **FRAM**, **Motor**, **Endswitch**, **LED**, **Localization** und **IO**.

Die Module der obersten Abstraktionsebene bilden die direkten Schnittstellen zur Außenwelt. Das **Manual Control**-Modul ermöglicht die manuelle Steuerung der Linearführung über die User-Buttons am Gehäuse und insbesondere die Umsetzung des Kalibrierungsprozesses. Auf der anderen Seite kann das System auch durch einen Hypertext Transfer Protocol (HTTP) Webserver überwacht und gesteuert werden. Außerdem werden im **REST**-Modul die Anfragen des Controllinos über die **TCP Server**-Verbindung verarbeitet, wie bereits oben erwähnt. Zuletzt wurde ein **Test**-Modul implementiert, das die Funktionalitäten der Module auf Ebene drei verifiziert. Dazu wurde ein einfaches Batch-Skript geschrieben, das die einzelnen Testcases auflistet und die Auswahl über UART an den Mikrocontroller sendet, sodass im **Test**-Modul eine entsprechende Funktion ausgeführt wird. In den nachfolgenden Abschnitten wird genauer auf einige Module eingegangen, beginnend mit dem untersten Abstraktionslevel.

4.1 IO-Modul

Dieses Modul soll den lesenden und schreibenden Zugriff auf jegliche GPIO-Pins erleichtern, die zur Kontrolle jeglicher Steuerelemente angeschlossen wurden. Allgemein besteht das System aus analogen und digitalen Sensoren bzw. Aktuatoren. Für jede Form wurde eine repräsentative Struktur definiert, in der alle relevanten Daten zusammengefasst werden.

4.1.1 Digitale Pins

Für digitale Ein- und Ausgänge wurde eine gemeinsame Struktur **IO_digitalPin_t** definiert:

```
typedef struct {  
    GPIO_TypeDef *GPIOx;  
    uint16_t GPIO_Pin;
```

```
    GPIO_PinState state;  
} IO_digitalPin_t;
```

Listing 1: Struktur für digitale Pins

Dabei ist der GPIO-Pin durch eine Nummer `GPIO_Pin` und einen Port `GPIOx` definiert. Zusätzlich wird in der Struktur der aktuelle Zustand `state` (null oder eins) gespeichert. Diese kann nun einer Funktion als Pointer übergeben werden, um z.B. den neuen Zustand auszulesen:

```
GPIO_PinState IO_digitalRead(IO_digitalPin_t *dig_IN) {  
    dig_IN->state = HAL_GPIO_ReadPin(  
        dig_IN->GPIOx, dig_IN->GPIO_Pin  
    );  
    return dig_IN->state;  
}
```

Listing 2: Einlesen eines digitalen Pin-Zustands

Dabei wird lediglich die Funktion `HAL_GPIO_ReadPin()` der `stm32f4xx_hal` Bibliothek aus Abbildung 17 aufgerufen, welche die zuvor besagten Attribute des Pins entgegennimmt und dessen Wert zurückgibt. Das Speichern dieses Werts in `state` hat den Vorteil, dass damit auf eine Änderung des Zustandes geschlossen werden kann. Dies ist z.B. für das Erfassen der steigenden und fallenden Flanke während eines Knopfdrucks sinnvoll und wurde wie folgt realisiert:

```
boolean_t IO_digitalRead_state_changed(IO_digitalPin_t *dig_IN) {  
    GPIO_PinState previous_state = dig_IN->state;  
    GPIO_PinState current_state = IO_digitalRead(dig_IN);  
    return current_state ^ previous_state;  
}
```

Listing 3: Detektion einer Flanke

Die Funktion in Listing 3 gibt durch eine logische exklusiv-oder-Verknüpfung von letztem und aktuellem Zustand in Form eines booleschen Werts an, ob sich der Zustand geändert hat oder nicht. Äquivalent zu Listing 2 ist eine Funktion

`IO_digital_write()` implementiert, die den gewünschten Pegel an einem entsprechenden Ausgangspin schaltet.

4.1.2 Analoge Pins

Etwas komplexer wird es mit dem Auslesen von analogen Sensorwerten. Dazu wird ein ADC-Kanal (`ADC_Channel`) des STM32 verwendet, der eine Spannung am Pin zwischen 0 und 3.3V in eine 12 bit Sequenz quantisiert und folglich als Integerwert `ADC_val` zwischen 0 und 4096-1 zurückgibt. Idealerweise entspricht dieser Wertebereich umgewandelt dem des Sensors, jedoch ist dies durch Ungenauigkeiten in der Hardware nicht immer der Fall, weshalb dies durch zusätzliche Grenzwerte in der Struktur für analoge Sensoren berücksichtigt wird:

```
typedef struct {
    ADC_HandleTypeDef *hadc_ptr;
    IO_SensorType_t Sensor_type;
    uint32_t ADC_Channel;
    uint16_t ADC_val;
    uint16_t max_sens_val;
    uint16_t sens_val;
    uint16_t min_sens_val;
    float max_pin_volt;
    float min_pin_volt;
} IO_analogSensor_t;
```

Listing 4: Struktur für analoge Sensoren

Die Bezeichnung `sense_val` aus Listing 4 steht dabei für den Wert in der Einheit des gewünschten Sensormesswerts und `pin_volt` für den umgewandelten Spannungswert am Pin. Damit lässt sich der Sensormesswert wie folgt berechnen:¹

```
void IO_Convert_from_ADC(IO_analogSensor_t *s) {
    float pin_volt = 3.3 * s->ADC_val / (4096-1)
    s->sens_val = (s->max_sens_val - s->min_sens_val) /
                (s->max_pin_volt - s->min_pin_volt) *
```

¹Funktion dient zur Veranschaulichung und wurde nicht exakt in dieser Form implementiert

```
(pin_volt - s->min_pin_volt) +  
s->min_sens_val  
}
```

Listing 5: Konvertierung des rohen Analogwerts

In umgekehrter Weise funktioniert die Umwandlung eines gewünschten Wertes in einen quantisierten Spannungswert für einen analogen Aktuator mithilfe einer entsprechenden Struktur `IO_analogActuator_t`.

Die Module `LED`, `Button` und `Endswitch`, repräsentieren physische Komponenten, die jeweils durch einen einfachen digitalen Pin angesteuert werden und enthalten daher lediglich Wiederverwendungen der Funktionen `IO_digitalWrite()`, um eine LED ein und auszuschalten, `IO_digitalRead_state_changed()`, um einen Knopfdruck zu erfassen oder `IO_digitalRead()`, um den Zustand eines Endschalters zu lesen. Dagegen wird der BG 45x30 SI Gleichstrommotor aufgrund seiner internen Regelung durch mehrere verschiedene Ein- und Ausgänge angesteuert.

4.2 Motor-Modul

Die Aufgabe des Motors ist, die Segelstellung entlang der Linearführung anzupassen. Dazu ermöglichen dessen digitale Eingänge gemäß Tabelle 1 eine flexible Einstellung der Drehzahl und -Richtung. Alternativ zu den konfigurierbaren fixen Geschwindigkeiten 1 und 2 kann mit dem analogen Eingang auch eine beliebige Drehzahl dynamisch vorgeben werden. Damit konnte eine Drehzahlrampe realisiert werden, die einen sanfteren Brems- und Beschleunigungsvorgang gewährleistet.

Über die digitalen Ausgänge kann der Ist-Zustand überwacht werden, darunter die aktuelle Drehrichtung und ob ein Fehler vorliegt. Zusätzlich kann durch ein Puls-Signal mit 12 Pulsen pro Umdrehung auf die Drehzahl geschlossen werden. Da der Motor in diesem Fall jedoch unter Drehzahlregelung betrieben wird, kann, sofern kein Fehlersignal vorliegt, von einer ausreichend präzisen Einhaltung der eingestellten Drehzahl ausgegangen werden. Dennoch ist das Puls-Signal von zentraler

Bedeutung, da aufgrund der Gewindesteigung der Linearführung und der Anzahl der Pulse ausgehend von einem Endpunkt die aktuelle Position und somit die Segelstellung ermittelt werden kann. Das Zählen der Pulse geschieht jedoch der Thematik entsprechend in einer Funktion des `Localization`-Moduls, die durch einen externen Interrupt am zuständigen GPIO-Pin von steigenden Flanken getriggert wird.

Orientiert an der Architektur des `IO`-Moduls dient auch hier eine Struktur als gemeinsamer Zugriffspunkt aller relevanter Daten des Motors:

```
typedef struct {
    Motor_function_t current_function;
    Motor_INs_t INs;
    IO_analogActuator_t AIN_set_rpm;
    IO_digitalPin_t OUT2_error;
    IO_digitalPin_t OUT3_rot_dir;
    uint16_t normal_rpm;
    uint16_t rpm_set_point;
    uint16_t ramp_final_rpm;
    uint32_t ramp_last_step_ms;
    boolean_t ramp_activated;
} Motor_t;
```

Listing 6: Struktur des Motors

4.2.1 Funktionsvorgabe

Grundlage zur Steuerung des Motors ist die Implementation der Funktionsvorgabe über die digitalen Eingänge, die in der Struktur nach Listing 6 durch ein Array `INs` des Typs `IO_digitalPin_t` organisiert sind. Zur einfachen Anwendung wurde ein enum `Motor_function_t` definiert, der alle Funktionen entsprechend Tabelle 1 bezeichnet und von oben beginnend von null bis sieben nummeriert. Durch eine geeignete Umrechnung in einen Binärwert zwischen 00 und 11 können aus den einzelnen Stellen direkt die passenden beiden Pins über das Array indiziert und mit

den Stellenwerten beschrieben werden. Das nachfolgende Listing zeigt die genaue Umsetzung dieses Verfahrens.

```
void Motor_set_function(Motor_t *motor, Motor_function_t func) {
    motor->current_function = func;
    uint8_t pin_offset = (func >= 4) * 2;
    int8_t function_bits = func - pin_offset * 2;
    for (int i = 0; i < 2; i++) {
        GPIO_PinState state = function_bits & 2 ? 1 : 0;
        uint8_t IN_idx = i + pin_offset;
        IO_digitalWrite(&motor->INs[IN_idx], state);
        function_bits <<= 1;
    }
}
```

Listing 7: Einstellung der Motorfunktionen

Im ersten Schritt wird überprüft, ob die Eingänge 0 & 1 oder 2 & 3 für die Funktion zuständig ist. Letzteres ist der Fall, wenn der Funktionsindex in der oberen Hälfte (≥ 4) liegt. Das Beschreiben der beiden Eingänge erfolgt iterativ über eine For-Schleife, wobei die Pins durch die Zählvariable `i` indiziert werden. Falls die Eingänge 2 & 3 zuständig sind, muss folglich ein `pin_offset` von zwei auf den Index addiert werden. Ebenfalls wird dann die Funktionsnummer mit vier subtrahiert, damit aus dem resultierenden Wert in `function_bits` der zweistellige Binärwert errechnet werden kann. Durch die bitweise und-Verknüpfung mit zwei und einer anschließenden einfachen links-shift Operation (MSB-first) werden die einzelnen Binärstellen der Reihe nach berechnet und mit `IO_digitalWrite()` als neuen Zustand `state` am passenden Pin geschaltet.

4.2.2 Drehzahlrampe

Darauf basierend wurden konkrete Funktionen zur Steuerung des Motors implementiert. Das Kernelement ist dabei die Realisierung der Drehzahlrampe innerhalb der Funktion `Motor_speed_ramp()`, welche in der Hauptschleife des Programms durchgehend ausgeführt wird. Die Rampe kann dabei durch eine Funktion

`Motor_start_moving()` aktiviert werden, die mit `Motor_set_function()` Links- oder Rechtslauf einstellt und anschließend das Flag `ramp_activated` aus Listing 6 setzt. Darauf hin wird der Drehzahlwert `rpm_setpoint` in regelmäßigen Zeitschritten um einen konstanten Wert erhöht. Nach Konfiguration auf *Drehzahlvorgabe* (Tabelle 1) kann die neue Soll-Drehzahl an dem analogen Eingang `AIN_set_rpm` mit `IO_analogWrite()` umgesetzt werden. Dies geschieht solange, bis die Zieldrehzahl `ramp_final_rpm=normal_rpm` erreicht ist und die Rampe wieder deaktiviert wird. Ebenso kann die Rampe mit `Motor_stop_moving()` für den Bremsvorgang aktiviert werden, wobei `ramp_final_rpm=0` gesetzt wird und die Drehzahl in gleicher Weise schrittweise reduziert wird. Aufgrund einiger Tests wurde für die Rampe eine Schrittdauer von 13 ms und eine Schrittweite von 10 rpm mit einer Normaldrehzahl von 1600 rpm festgelegt, sodass sich folgender zeitlicher Verlauf ergibt:

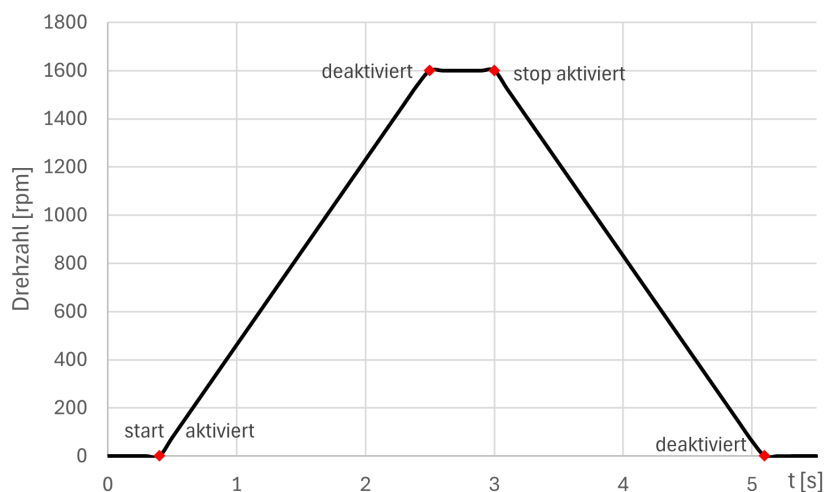


Abbildung 18: Drehzahlrampe: Beschleunigen und Bremsen

Die Stopp-Funktion bietet zusätzlich die Möglichkeit über einen booleschen Parameter `immediate` z.B. während einer Notabschaltung den Motor abrupt anzuhalten. In diesem Fall wird nicht die Rampe aktiviert sondern direkt die Motorfunktion *Motor aus* gesetzt.

4.3 Steuerung der Linearführung

Nachdem die allgemeine Steuerung des Motors implementiert ist, gilt es, diese auf die Positionierung der Segel über die Linearführung anzuwenden.

4.3.1 Kalibrierung

Damit später eine bestimmte Stellung angefahren werden kann, muss die aktuelle Position jederzeit bekannt sein. Wie bereits erwähnt, dient das Puls-Signal des Motors als Maß, wie weit sich die Position von einem Startpunkt entfernt hat. Jedoch muss dieser Punkt erst angefahren werden, was durch die Detektion der Endschalter keine Schwierigkeit darstellt. Damit kann die absolute Position anhand der Puls-Zahl und der Gewindesteigung ermittelt werden. Für die Einstellung der Segel wird vom Controllino eine Prozentzahl vorgegeben wie weit gerollt oder getrimmt werden soll. Um diesen Wert auf eine absolute Position auf der Linearführung abzubilden, muss zusätzlich die Distanz zwischen den beiden Endpunkten ermittelt werden. Außerdem soll die Grenze zwischen Roll- und Trimbereich nachträglich manuell angepasst werden können, um potenzielle Asymmetrien in der Seilführung der Segel auszugleichen. Der gesamte Ablauf der Kalibrierung wurde in Form einer Zustandsmaschine realisiert, wie in Abbildung 19 dargestellt.

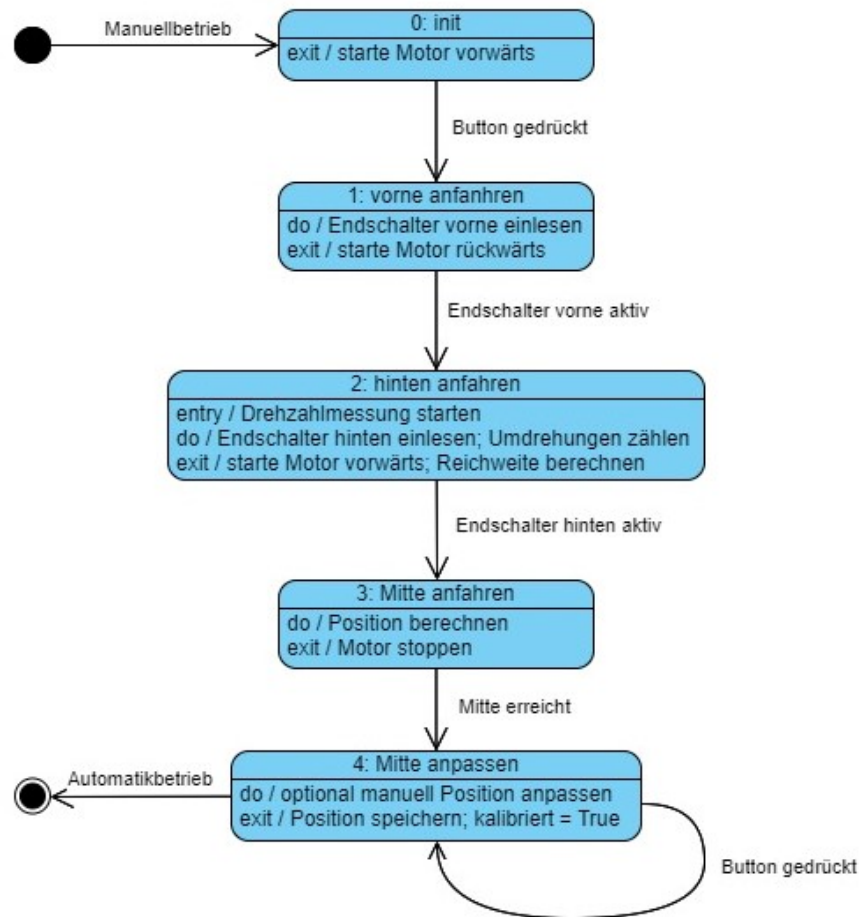


Abbildung 19: Kalibrierungsprozess

Die Eintrittsbedingung der Kalibrierung ist das Einschalten des manuellen Betriebsmodus durch den Kippschalter am Gehäuse oder alternativ über den Webserver. Aus dem Ausgangszustand heraus muss der Knopf *Speichern/Bestätigen* aus Abbildung 8 gedrückt werden, um die automatische Kalibrierung einzuleiten. Dies ist mit dem starten des Motors in Vorwärtsrichtung und dem Übergang in Zustand 1 verbunden. Anschließend wird der vordere Endschalte abgefragt, bis dieser aktiviert wird und die Endposition erreicht ist. Von dieser Position aus beginnt die Pulszählung, damit nach Ankunft am anderen Ende die zurückgelegte Distanz be-

rechnet werden kann. Aus dieser Distanz wird daraufhin die Mitte bestimmt und angefahren. Ist diese erreicht, kann zuletzt die tatsächliche Grenze manuell über die Knöpfe *Trimmen* und *Rollen* angefahren und mit dem erneuten Druck des einleitenden Knopfes bestätigt werden. Als Feedback blinkt dabei die LED *Position gespeichert*. Ebenso aktivieren bzw. deaktivieren sich nun nach der Einteilung der Bereiche je nach Positionierung die entsprechenden LEDs (*Im Trimmungsbereich* / *Im Rollbereich*).

Da die Grenze in Zustand 4 mit dem entsprechenden Knopf beliebig oft angepasst werden kann, ist es durch das selbe Signal nicht möglich, den kompletten Prozess von vorne zu starten, ohne das System neuzustarten. Daher wurde ein Timer mit einem periodischen Callback konfiguriert, der die Dauer des Knopfdrucks ermittelt und ab einer Druckdauer von 3s die Zustandsmaschine zurücksetzt, anstatt lediglich die Grenze zu überschreiben. Nach Abschluss der Kalibrierung kann in den Automatikbetrieb gewechselt werden. Im Automatikbetrieb kann der Controllino aufgrund der bereitgestellten Daten regelmäßig die optimale Segelstellung neu berechnen und an den STM32 zurückgeben. Der nächste Abschnitt beschäftigt sich damit, wie genau die gewünschte Segelstellung in eine Position auf der Linearführung umgerechnet wird.

4.3.2 Umsetzung einer Rollung/Trimmung

Der Befehl einer Segelanpassung wird vom Controllino durch einen Modus (Rollen oder Trimmen) und eine Prozentzahl definiert. Abbildung 20 zeigt graphisch die Bedeutung dieser Angabe bezüglich der Positionierung an der Linearführung.

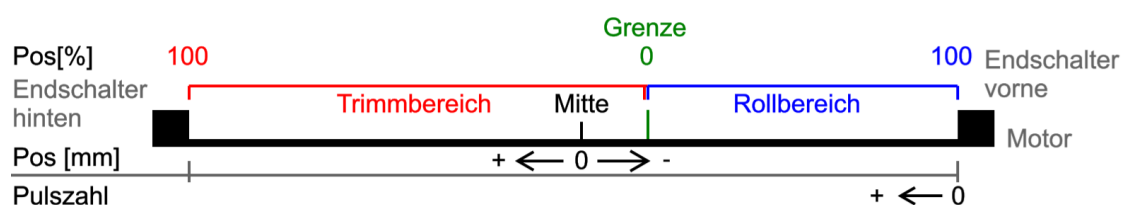


Abbildung 20: Rollung und Trimmung an der Linearführung

In diesem Beispiel weicht die Grenze zwischen den beiden Bereichen stark von der tatsächlichen Mitte ab, um die Möglichkeit der manuellen Anpassung, zu verdeutlichen. Zur besseren Lesbarkeit der aktuellen Position wird diese in einem Zwischenschritt in mm umgerechnet und die Mitte der Gesamtdistanz auf null zentriert. In dieser Einheit werden jegliche Positionsrechnungen durchgeführt.

Für die Umwandlungen zwischen Prozent, mm und Pulsen werden alle relevanten Daten in einer gleichnamigen Struktur des `Localization`-Moduls festgehalten. Listing 8 zeigt die Ermittlung der gewünschten Position in mm mithilfe dieser Struktur aufgrund eines Segelanpassungsbefehls:

```
void Linear_Guide_set_desired_roll_trim_percentage(  
    Localization_t *loc, uint8_t percentage,  
    adjustment_mode_t mode  
)  
{  
    int8_t sign = mode == adjustment_mode_roll ? 1 : -1;  
    int32_t roll_trim_distance = loc->end_pos_mm  
                                + sign * loc->center_pos_mm;  
    loc->desired_pos_mm = (int32_t)  
        (- sign * (roll_trim_distance * (percentage / 100.0F)  
            - sign * loc->center_pos_mm));  
}
```

Listing 8: Berechnung der Zielposition

Zunächst wird ein Vorzeichen `sign` festgelegt, da der Rollbereich im negativen und der Trimmbereich im positiven mm Bereich liegt. Je nachdem welche Modus gewünscht ist, berechnet sich dessen Streckenanteil an der Linearführung aus der Distanz `end_pos_mm` von der Mitte bis zu einem Endpunkt und dem Abstand `center_pos_mm * sign` von der Mitte bis zur gewählten Grenze `center_pos_mm`. Von dieser Strecke wird der gewünschte Anteil `percentage` durch das Vorzeichen auf die richtige Seite orientiert. Zuletzt muss das addieren des Abstandes `center_pos_mm * sign` wieder rückgängig gemacht werden.

Nachdem die gewünschte Position in mm vorliegt, muss diese angefahren werden. Dazu muss diese mit der aktuellen Position der Linearführung verglichen werden, welche sich nach der Kalibrierung aus der Pulszahl des Motors ergibt. Die Pulszahl wird wie bereits erwähnt durch den externen Interrupt am zuständigen GPIO-Pin regelmäßig aktualisiert und in der Struktur `Localization_t` aus Listing 8 festgehalten. Die Positionsumrechnung ist in einer Funktion des `Localization`-Moduls implementiert, wie nachfolgend gezeigt:

```
void Localization_update_position(Localization_t *loc)
{
    uint16_t abs_pos = loc->pulse_count
                    * loc->distance_per_pulse;
    loc->current_pos_mm = abs_pos - loc->end_pos_mm;
}
```

Listing 9: Positionsberechnung aus dem Pulssignal des Motors

Dabei berechnet sich die zurückgelegte Strecke pro Puls `distance_per_pulse` aus dem Quotienten der Gewindesteigung von 1.12 mm pro Umdrehung und der Pulsfrequenz des Motors von 12 Pulsen pro Umdrehung zu 0.093 mm pro Puls.

4.3.3 Optimierung der Bewegungsübergänge

Die Steuerung der Linearführung wird durch das Aktivieren von Drehzahlrampen gelöst, um sanfte Beschleunigungs- und Bremsvorgänge zu ermöglichen. Allerdings gibt es unter dem bisherigen Stand der Implementierung mehrere Situationen, in denen es trotzdem noch zu abrupten Bewegungsänderungen kommt. Eine davon stellt das Erreichen eines Endpunktes dar, denn die Aktivierung des Endschalters ist direkt mit einem hardware-geregelten Stopp des Motors verbunden. Ebenfalls kann die Rampenfunktion nicht zwischen Beschleunigung in Vorwärts- und Rückwärtsrichtung unterscheiden, weshalb bei ein Befehl in die umgekehrte Richtung sofort umgesetzt wird. Ein anderes vielleicht noch wichtigeres Problem ist, dass eine gewünschte Position nicht korrekt angefahren werden kann, da der Motor nach dem Stopp-Befehl aufgrund der Rampe noch ein kleines Stück weiterfährt.

Die Lösung dieser Probleme liegt in der Ermittlung des Bremsweges sowie der Einführung einer Warteschlange für Zielpositionen. Darauf basierend wurde ein Algorithmus zur optimierten Koordination der Steuerbefehle implementiert, wie nachfolgend in Form eines Programmablaufplans dargestellt:

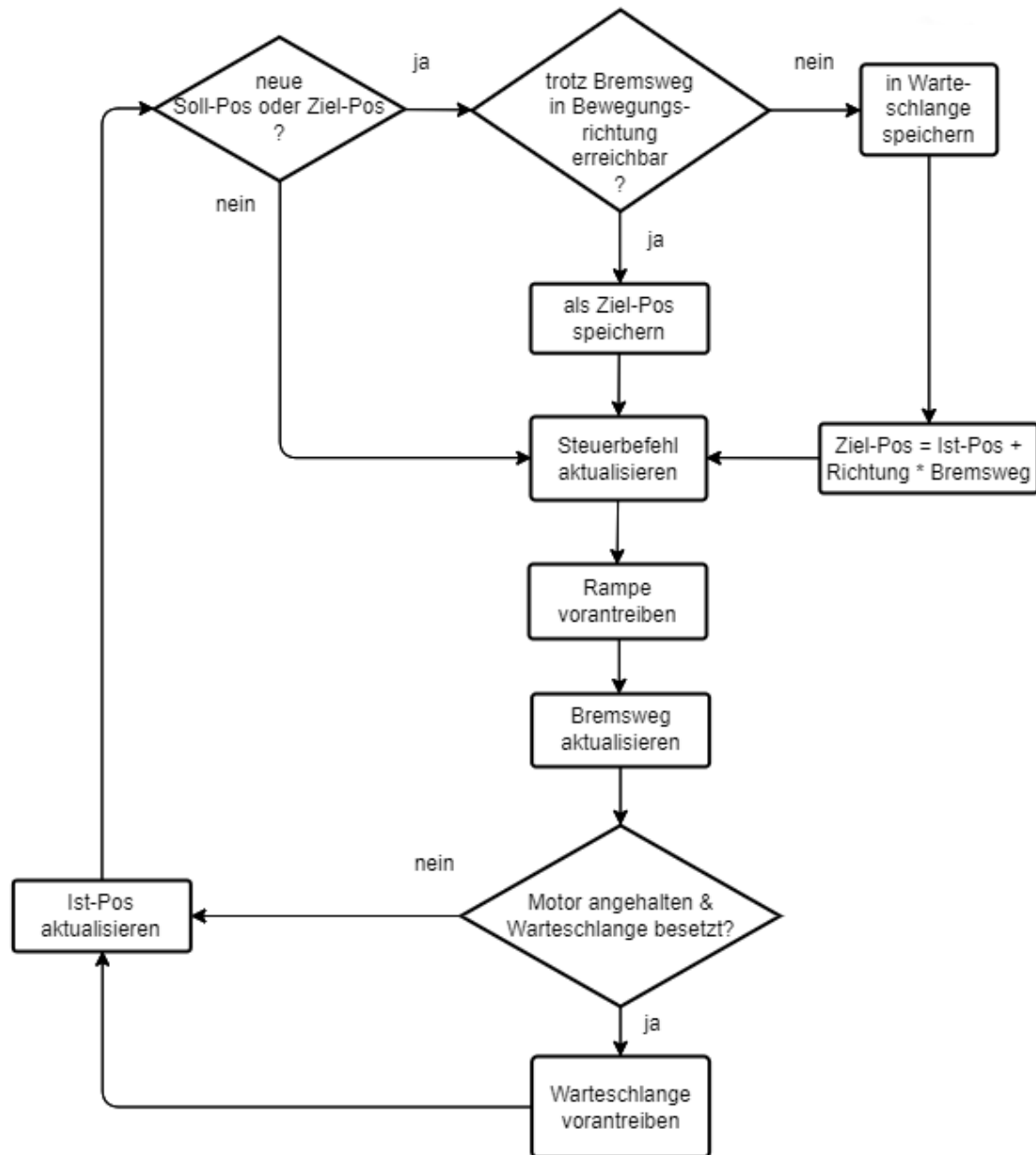


Abbildung 21: Optimierung der Steuerbefehle

Eine Soll-Position kann entweder durch die manuelle Steuerung oder durch eine gewünschte Segelstellung vom Controllino während des Automatikbetriebs vorge-

geben werden. Hat sich diese geändert, kann es sein, dass sich die Linearführung wegen eines vorhergegangenen Befehls bereits in Bewegung befindet. In diesem Fall muss überprüft werden, ob sich die Soll-Position in Bewegungsrichtung vor der aktuellen Position befindet und ggfs. ob der Abstand aufgrund des Bremsweges noch erreichbar ist. Wenn diese Bedingungen nicht erfüllt sind, wird die Soll-Position nicht direkt als neues Ziel gesetzt sondern in einer Warteschlange gespeichert, wobei diese eigentlich nur eine einzelne Variable der `Localization_t` Struktur ist. Damit die neue Soll-Position trotzdem so schnell wie möglich erreicht wird, bildet der in Fahrtrichtung nächste noch erreichbare Punkt die neue Zielposition. Danach kann der Steuerbefehl aktualisiert werden (Vorwärts, Rückwärts oder anhalten). Nach Vorantreiben der Rampe muss der Bremsweg aufgrund der aktuellen Drehzahl ebenfalls neu berechnet werden. Ist der Motor zum Stillstand gekommen, kann sofern vorliegend die Position aus der Warteschlange als neues Ziel vorgerückt werden. Zuletzt muss natürlich auch die aktuelle Position der Linearführung aktualisiert werden bevor der Kreislauf von neuem beginnt.

Sollte eine Soll-Position auch nach einem Richtungswechsel aufgrund einer Fehlberechnung des Bremsweges verfehlt werden, wird diese einfach erneut in die Warteschlange gesetzt, sodass sich die exakte Position garantiert einpendelt. Zusammenfassend werden nun mithilfe der Ziel-Warteschlange abrupte Richtungswechsel vermieden und durch die Berechnung des Bremsweges kann das System rechtzeitig anhalten, um die Soll-Position präzise anzufahren oder den Aufprall am Endschalter zu verhindern.

4.3.4 Wiederherstellung der Position

Die in Kapitel 4.3.1 beschriebene Kalibrierung ist ein zeitintensiver Prozess. Damit diese nicht bei jedem Programmneustart erneut durchgeführt werden muss, wurde ein FRAM Speicher in der Hardware verbaut, um dort alle relevanten Informationen zur Wiederherstellung der aktuellen Position hineinzuschreiben und bei Bedarf wieder auszulesen. Für diese beiden Funktionen wurde ein Benutzerinterface im

FRAM-Modul implementiert.

Das FRAM Modul steuert den FRAM über SPI an. Der FRAM kann dabei über sechs Befehle gesteuert werden. Dabei wurden die Befehle: WREN(0000 0110b), RDSR(0000 0101b), READ(0000 0011b) und WRITE(0000 0010b) genutzt. Um auf den Speicher zu schreiben wird das Write Enable (WREN) Bit gesetzt. Hierfür wird der gleichnamige Befehl genutzt. Um zu überprüfen ob das Bit gesetzt wurde, wird es mit dem Befehl RDSR ausgelesen. Um den Speicher zu beschreiben wird mit dem WRITE Befehl gefolgt von einer 12Bit Adresse der Speicherbereich ausgewählt. Die darauffolgenden Bits werden in den Speicher geschrieben. Beim Auslesen mit dem Befehl READ ist das vorgehen gleich, nur werden hier keine zusätzlichen Bits übertragen. In der Software wurden dabei die zwei Funktionen `FRAM_write` und `FRAM_read` erstellt die für einen High Level Access sorgen. Über diese kann ein Puffer in den FRAM geschrieben werden oder die Daten ausgelesen werden. Die Serialisierung und Deserialisierung, sowie das setzen des WREN Bits wird beim Aufruf übernommen. Selbes gilt für das Auswählen des Chips über die Chip Select Leitung. Der Serial Clock Takt und die Übertragung von Daten geschieht mit dem `HAL_SPI_Transmit`/`HAL_SPI_Receive` Funktionen.

Damit die Daten beim Auslesen zur Initialisierung des Systems aktuell sind, werden diese bei jeder Positionsänderung an der entsprechenden Speicheradresse überschrieben. Die für die Lokalisierung nötigen Informationen werden zur Serialisierung innerhalb einer separaten Struktur organisiert, wie nachfolgend gezeigt:

```
typedef struct {
    Loc_state_t state;
    int16_t pulse_count;
    uint16_t end_pos_mm;
    int16_t center_pos_mm;
    uint16_t start_pos_abs_mm;
} Loc_safe_data_t;
```

Listing 10: Speicherformat der Positionsdaten

Die bisher nicht eingeführte Variable `start_pos_abs_mm` entspricht dem vom Distanzsensor gemessenen Abstand der Position bei Aktivierung des vorderen, motorseitigen Endschalters. Daraus lässt sich die mit dem Puls-Signal ermittelte Position mit dem Messwert des Sensors vergleichen. Die Bedeutung der Messung wird in Kapitel 4.3.5 genauer behandelt.

Der Schreibprozess dieser Daten in das FRAM ist durch eine Funktion `Linear_Guide_safe_Localization()` gegeben. Listing 11 zeigt deren Implementierung.

```
int8_t Linear_Guide_safe_Localization(Localization_t loc)
{
    uint8_t FRAM_buffer[sizeof(Loc_safe_data_t)];
    Localization_serialize(loc, FRAM_buffer);
    if(FRAM_write(FRAM_buffer,
                  LINEAR_GUIDE_INFOS,
                  sizeof(Loc_safe_data_t)) != HAL_OK)
    {
        printf("Saving␣Position␣failed!\r\n");
        return LG_LOCALIZATION_FAILED;
    }
    return LG_LOCALIZATION_SAFED;
}
```

Listing 11: FRAM Speichervorgang

Zunächst wird ein FRAM-Puffer der Größe des Speicherformats aus Listing 10 deklariert. Dann werden die entsprechenden Daten in der Funktion `Localization_serialize()` von der Struktur `loc` in das richtige Format extrahiert und anschließend an den Puffer adressiert. Zuletzt werden die Daten innerhalb des Puffers durch `FRAM_write()` an die definierte Adresse `LINEAR_GUIDE_INFOS` des physischen Speichers transferiert.

Nachdem die Daten während des Betriebs kontinuierlich im FRAM aktualisiert werden, können diese bei einem Neustart des Systems in umgekehrter Weise wie in

Listing 11 durch `FRAM_read()` abgerufen werden. Listing 12 zeigt die Verarbeitung des resultierenden Ausgangspuffers `buffer`:

```
static int8_t Localization_deserialize(  
    Localization_t *loc,  
    uint8_t buffer[sizeof(Loc_safe_data_t)]  
)  
{  
    Loc_safe_data_t *data = (Loc_safe_data_t *)  
        malloc(sizeof(Loc_safe_data_t));  
    memcpy(data, buffer, sizeof(Loc_safe_data_t));  
    loc->state = data->state;  
    loc->pulse_count = data->pulse_count;  
    loc->end_pos_mm = data->end_pos_mm;  
    loc->center_pos_mm = data->center_pos_mm;  
    loc->start_pos_abs_mm = data->start_pos_abs_mm;  
    free(safe_data_ptr);  
    if (loc->state != Loc_state_5_center_pos_set)  
    {  
        return LOC_RECOVERY_RESET;  
    }  
    return LOC_RECOVERY_COMPLETE;  
}
```

Listing 12: Auslesen der Positionsdaten aus dem FRAM-Puffer

Im ersten Schritt wird ein Pointer `data` im Speicherformat `Loc_safe_data_t` entsprechend allokiert, damit diesem anschließend mit der C-Standard-Funktion `memcpy()` die Daten aus dem Puffer übertragen werden können. Danach werden die einzelnen Werte der Reihe nach der Hauptstruktur des `Localization`-Moduls übergeben. Zuletzt wird mit der `state`-Variable überprüft, ob die Kalibrierung beim letzten Betrieb abgeschlossen wurde. Falls nicht, muss diese wiederholt werden, was das Rückgabe-Flag `LOC_RECOVERY_RESET` signalisiert. War die Wiederherstellung der Lokalisierung erfolgreich, kann die aktuelle Position wie gehabt entsprechend Listing 9 mit den Werten `pulse_count` und `end_pos_mm` ermittelt werden. Durch das Speichern der Grenze `center_pos_mm` bleiben außerdem auch

die zuvor eingestellten Trimm- und Rollbereiche erhalten.

4.3.5 Fehlerbehandlung

Mithilfe der Sensoren und dem Fehlersignal des Motors können verschiedene Fehlerzustände identifiziert werden. Tabelle 4 listet diese auf, stellt deren Bedingung dar und wie damit umgegangen wird.

ID	Bezeichnung	Bedingung	Behandlung
0	Normal	kein Fehler identifiziert	keine
1	Distanzfehler	Toleranz von 1 cm zwischen Messwert und Pulsberechnung überschritten	Anpassung an Messwert
2	Windfehler	max. Geschwindigkeit überschritten	Segel 100 % einrollen
3	Motorfehler	Fehlersignal aktiv	Notabschaltung
4	Stromfehler	Stromsensorwert von 4 A überschritten	Notabschaltung

Tabelle 4: Fehlerzustände

Bei jedem Fehler leuchtet als Feedback die LED *Störung* aus Abbildung 8 auf. Während Fehler 1 vom System automatisch korrigiert werden kann, wird bei den anderen im Gegensatz dazu der weitere Betrieb solange blockiert, bis diese behoben sind. Die Toleranz zur maximalen Windgeschwindigkeit soll dabei vom Controllino festgelegt werden, da dort auch die Berechnungen zur optimalen Segelstellung durchgeführt werden. Daher wird der Windfehler auch nicht vom STM32 selbst gesetzt, sondern vom Controllino über das REST-Protokoll kommuniziert.

4.4 Windmessung mit dem Anemometer

Wie bereits in Unterunterabschnitt 3.7.6 aufgezeigt war die Präzision der Analogen Signale zwar ausreichend aber teilweise sehr ungenau. Aus diesem Grund wurde stattdessen die digitale RS485 Schnittstelle zur Übertragung der Windgeschwindigkeit und Windrichtung genutzt. Der WSWD bietet dabei mehrere Protokolle an, um diese Daten Formattiert zu übertragen. Dabei wurde das National Marine Electronics Association (NMEA) Telegramm ausgewählt. Die Formatierung ist in Abbildung 22 dargestellt.

Telegrammaufbau:

\$	<MSGTYPE>	,	<WR>	,	R	,	<WG>	,	<U>	,	<V>	*	<CRC>	<CR><LF>
\$	XXXXX	,	XXX.X	,	R	,	XXX.XX	,	u	,	x	*	hh	CR LF

Abbildung 22: Aufbau eines NMEA Telegramm (vgl.[9], S.61)

Hier wird neben den Daten (< WR > und < WG >) ebenfalls die Gültigkeit der Messung (< V >) und die Einheit der Geschwindigkeit (< U >) übertragen. Das versenden von Befehlen geschieht dabei aber in einem von MESA selbsterstellten Format:

<ID>	<CM>	<PARAM>	<CR><LF>
Geräte ID	Kommando	Parameter	Befehlsabschluss

Tabelle 5: Aufbau eines Kommandos zur Einstellung des WSWD (vgl. [9], S.26f.)

Dieses wird im Normal Betrieb nicht benötigt, aber kann zur einmaligen Konfiguration des Sensors genutzt werden. Zur Konfiguration des Anemometers wurde z.B dabei der Befehl 42 angewandt um die bereits Erklärten NMEA Telegramme in einem regelmäßigen Abstand gesendet zu bekommen.

Die Befehle an das Anemometer werden über UART Gesendet und Empfangen. Das Signal wird dann vom MAX3485 zu RS485 konvertiert. Da RS485 nur eine Halbduplex Kommunikation zulässt muss vorher über das Setzen eines Pins immer

zwischen Senden und Empfangen gewechselt werden. Dies wurde in zwei Funktionen: `WSWD_enable_receive` und `WSWD_enable_send` realisiert. Diese prüfen zuerst den Status des Pins und ändern diesen falls nötig. Diese werden in den Senden und Empfangen Funktionen genutzt. Diese Serialisieren/Deserialisieren eine Zeichenkette mit einem Cast zu `uint8_t*` und Versenden/Empfangen diese über die `HAL_UART_Transmit`/`HAL_UART_Receive` Funktionen:

```
uint8_t WSWD_receive
(char* receive_buffer, uint8_t size_of_receive_buffer)
{
    WSWD_enable_receive();
    if(HAL_UART_Receive(&huart2, (uint8_t*)receive_buffer,
        size_of_receive_buffer, WSWD_UART_TIMEOUT) != HAL_OK)
    {
        printf("error sending to WSWD\r\n");
    }
    return HAL_OK;
}
```

Listing 13: Empfangen einer Nachricht vom WSWD

Beim Empfangen der Daten wird das komplette Telegramm als ein String gespeichert. Die Telegramm Struktur wird danach durch String Manipulation geprüft und die gesendeten Daten über die `atof` Funktion zu einem nutzbaren `float` Datentyp Konvertiert. Dasselbe Prinzip wird auch beim Versenden eines Befehles angewandt, nur das hier der String mit `snprintf` zusammengesetzt wird.

4.5 REST Interface

Das REST Interface wird genutzt um mit dem Controllion zu kommunizieren. Zuerst wurden die hierfür benötigten HTTP Befehle festgelegt. Da der Controllino als Client agieren soll und somit immer der Initiator der Kommunikation ist wurden die Befehle: `GET` und `PUT` als ausreichend angesehen. Die dabei mitversendeten Daten sollen im Java Script Object Notation (JSON) Format versendet werden. Die

URL auf die der Controllino über die Application Programming Interface (API) zugreift wurden ebenfalls festgelegt. Diese wurden nach den unterschiedlichen Sensor- und Informationsdaten organisiert. Um z.B alle Informationen abzufragen sendet der Controllino den Befehl: `GET /data`. Sollte er speziellere Infos abfragen wollen, wie z.B die Segelposition, sendet er den Befehl: `GET /data/adjustment`. Eine Übersicht aller validen Uniform Resource Locator (URL)s und JSON Formatierungen ist in Anhang C zu finden. Ein Beispiel für ein Antwort Paket für den `GET /data/adjustment` Befehl ist in Listing 14 zu sehen.

```
{
    "sail_pos": <(int) pitch/roll [ +/- 0...100 %]>
}
```

Listing 14: `GET /data/adjustment` Antwort Paket

Über den `PUT` Befehl kann der Controllino einen Fehler setzen, eine gewünschte Position anfahren oder den Betriebsmodus der Steuerung ändern. Zusätzlich kann auch der maximale Distanz Fehler und die maximale Drehzahl des Motors festgelegt werden. Der TCP-Server für die REST Kommunikation ist über den Port 2375 erreichbar und die Standard IP-Adresse als 192.168.0.123 definiert.

Zur Netzwerkkommunikation wurde der Lightweight IP (LWIP) Stack genutzt. Dieser beinhaltet allerdings keine bereits integrierte REST API. Aus diesem Grund wurde diese selbst implementiert. Dabei wurde ein Transmission Control Protocol (TCP) Server erstellt. Dieser wird auf dem Mikrocontroller gehostet. Dieser nimmt die Rohdaten aus den Empfangen Paketen und speichert diese in einem Buffer. Der Buffer wird daraufhin an die REST API weitergegeben. Diese unterscheidet dabei zwischen den zwei möglich Befehlen und den akzeptierten URLs. Falls diese nicht stimmen wird mit dem HTTP Error code `404 Not Found` geantwortet. Falls es sich um einen `PUT` Befehl handelt, wird der mitgesendete JSON Payload auf eine Korrekte Formattierung geprüft. Hierfür wurde die `cJson` Bibliothek genutzt. Je nachdem ob die Formattierung stimmt wird mit `200 OK` oder mit `501 Not Implemented` bzw. `400 Bad Request` geantwortet.

Beim versenden einer Antwort auf eine `GET` Request wird ebenfalls mit Hilfe der `cJson` Bibliothek ein Payload zusammengebaut, der die aktuellen angeforderten Werte beinhaltet.

4.6 Weboberfläche

Der Webserver stellt eine alternative zu der physikalischen Bedienungsoberfläche dar. Diese ist in manchen Fällen ggü. der Bedienungsoberfläche zu bevorzugen, da hier schneller Anpassungen durchgeführt werden können und genauere Informationen dargestellt werden können.

4.6.1 Funktionalität

Die Weboberfläche, soll es einer Arbeitskraft erlauben den aktuellen Status, sowie Einstellungen des Gerätes abzufragen oder festzulegen. Dabei sollen dieselben Informationen, die der Controllino über die REST API abfragen kann auch auf diesem dargestellt werden. Zusätzlich dazu soll es hier auch möglich sein, dieselbe Funktionalität wie die Benutzeroberfläche zu bieten. Also sollte es möglich sein Manuell über die Weboberfläche die Segel zu Rollen und zu Trimmen. Dazu gehört auch die Auslösung einer Kalibrierungsfahrt. Die Netzwerkeinstellungen sollten ebenfalls über die Weboberfläche gesetzt werden können. Dazu zählen die Aktivierung von DHCP, das Festlegen einer IP-Adresse und das Zurücksetzen dieser.

4.6.2 Webseiten

Die Webseiten wurden dabei nach ihrem Inhalt sortiert. Es wurden insgesamt sechs Seiten implementiert:

- Index Seite
- Positionsdaten Seite
- Sensordaten Seite

- Einstellungs Seite
- Error Seite
- Login Seite

Als Beispiel ist die Index Seite in Abbildung 23 dargestellt.

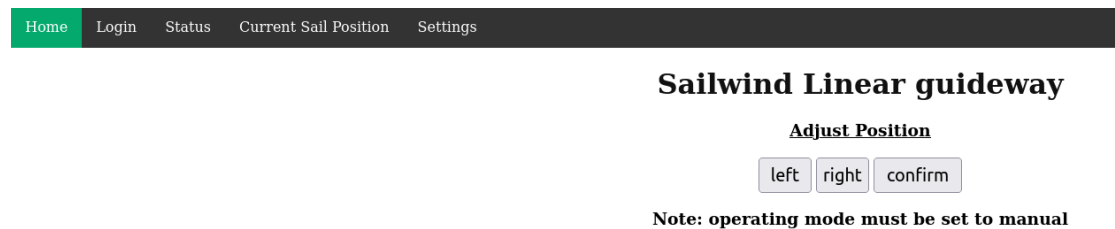


Abbildung 23: Index Seite des Webservers

Hier befindet sich die manuelle Steuerung zur Positionierung der Segel. Auf allen Seiten wird dabei am oberen Rand eine Navigations Bar angezeigt mit der zu den anderen Seiten Navigiert wird. Auf der Positionsdaten Seite ist der aktuelle Betriebsmodus und die Trimmung/Rollung der Segel zu finden. Die Sensordaten Seite zeigt alle aktuellen Werte der Sensoren an und wird regelmäßig aktualisiert. Die Einstellungsseite zeigt die aktuellen Einstellungen und macht es möglich den Betriebsmodus zu ändern, das Gerät neuzustarten und eine neue IP Adresse festzulegen oder Dynamic Host Configuration Protocol (DHCP) zu aktivieren. Die Login Seite hat aktuell keine Funktion und dient nur als Platzhalter. Hier könnten später je nach Nutzer die aufrufbaren Webseiten eingeschränkt werden. So kann z.B ein Service Mitarbeiter auf alle Einstellungen zugreifen, während ein normaler Nutzer nur z.B die Sensordaten abfragen kann. Die Error Seite wird nur bei einer Zeitüberschreitung der Anfrage angezeigt und hat keinen weiteren Nutzen.

4.6.3 Umsetzung

Die Webserver erhält dieselbe Internet Protocol (IP) Adresse wie die REST API ist aber auf dem Port 80 erreichbar. Im Gegensatz zum REST interface bietet

LWIP hier bereits schon eine fertige Webserver implementierung. Dabei können Daten über die Server Side Includes (SSI) und Common Gateway Interface (CGI) Handler Übertragen werden. Beim Aufrufen einer Webseite werden die im Hyper-text Markup Language (HTML) Code festgelegten SSI Tags mit den entsprechenden Sensor Daten oder Text mit einer Rückmeldung für den User ersetzt. Hierfür wurde ein zentrales **switch-case** festgelegt, das nach der Nummer des SSI Tags unterscheidet und den dementsprechend String einfügt. Der CGI Handler wird für Textboxen oder für Buttons genutzt. Für Buttons wurden dafür HTML Forms genutzt. Diese haben als **Action** einen Uniform Resource Identifier (URI) wie in Listing 15 zusehen ist.

```
<form action="/form_control.cgi">
  <p>
    <input type="submit"
      id="move"
      value="confirm"
      name="move"
      style="width:100px;height:40px;font-size:20px;">
  </p>
</form>
```

Listing 15: GET /data/adjustment Antwort Paket

Wird jetzt z.B auf der Index Seite der Button „Confirm“ angeklickt wird der String: `/form_control.cgi?move=confirm` zum Mikrocontroller gesendet. Auf Basis dieses String kann daraufhin eine Aktion gewählt werden und z.B die Anzeige auf dem Webserver verändert werden. Textinputs funktionieren ähnlich nur das sie den Inhalt der Textbox nach der URI beinhalten.

5 Fazit und Ausblick

Für die Steuerung der Segel des Sailwind 4 Projekts konnte eine Platine entwickelt werden, welche die Hauptsteuerelemente über geeignete Schnittstellen mit dem STM32 Mikrocontroller als zentrale Recheneinheit verbindet. Mit dem passgenauen Einbau in ein Staub- und Spritzwassergeschütztes Gehäuse sollte das System den Anforderungen äußerer Bedingungen während des Betriebs an echten Segelwindmühlen genügen. Trotz einiger Probleme beim Design der Platine konnten alle Komponenten erfolgreich angesteuert und ausgelesen werden. Die meisten Fehler im Design konnten dabei durch Analysen gefunden und eliminiert werden.

Die in den STM32 implementierte Firmware ermöglicht die manuelle Bedienung des Systems über die Schalter und Taster, die an der Oberfläche des Gehäuses angebracht sind, oder alternativ über einen Webserver, auf den über das Ethernet zugegriffen werden kann. Für visuelles Feedback sorgen mehrere LEDs, ebenfalls am Gehäuse, die z.B. über eine mögliche Störung, den Betriebsmodus oder den aktuell angefahrenen Bereich der Segelsteuerung (rollen oder trimmen) Auskunft geben.

Um die Segel über die Linearführung präzise einstellen zu können, kann über die Bedienelemente ein automatischer Kalibrierungsprozess eingeleitet werden. Dieser ermittelt zunächst aufgrund des Puls-Signals des Motors und der Gewindesteigung der Linearführung durch anfahren der beiden Endschalter die aktuelle Position des Schlittens relativ zum Gesamtsteuerbereich und ermöglicht anschließend das Anpassen der Grenze zwischen rollen und trimmen. Damit dieser Prozess nicht bei jedem Neustart durchgeführt werden muss, wird die aktuelle Position und alle weiteren Daten der Kalibrierung bei jeder Bewegung in den FRAM-Speicher geschrieben. Bei Systemneustart signalisiert dann eine der LEDs, ob die Wiederherstellung der Position erfolgreich war und somit direkt in den Automatikbetrieb übergegangen werden kann.

Zur Optimierung der Bewegungsabläufe während der Umsetzung neuer Steuerbefehle wurde eine beidseitige Drehzahlrampe für sanftere Beschleunigungs- und Bremsvorgänge des Motors implementiert. Die Verwendung der Rampe erforderte einen zusätzlichen Algorithmus, der abrupte Richtungsänderungen verhindert und gleichzeitig das exakte Anfahren einer Zielposition garantiert. Dies konnte mit der Berechnung von Bremswegen und der Koordination der Steuerbefehle mithilfe einer Zielwarteschlange gelöst werden.

Durch die Auswertung der Distanz- und Stromsensoren, sowie dem Anemometer und den digitalen Ausgängen des Motors ist das System in der Lage, selbst Störungen zu identifizieren und entsprechende Gegenmaßnahmen einzuleiten. Allerdings konnte mit dem Stromsensor keine besonders präzise Messung erreicht werden. Bei dem Distanzsensord ergab sich eine Einschaltzeit von ca. 15 min, bis sich ein stabiler Wert einpendelt, was jedoch im Rahmen der Anforderungen liegt.

Zuletzt wurde mit dem REST-Modul eine geeignete Schnittstelle implementiert, welche die Kommunikation zwischen der Steuereinheit und dem Controllino über eine Ethernet Verbindung ermöglicht. Dieser kann dadurch mit Statusinformationen der Linearführung sowie Daten zur Windgeschwindigkeit und -richtung versorgt werden, um darauf basierend mit einem Befehl zum Rollen oder Trimmen der Segel zu reagieren.

Das Endprodukt dieser Arbeit kann für die Entwicklung des Sailwind 4 Projekts wiederverwendet werden. Dabei sollten weitere Tests an der Implementierung der Firmware durchgeführt werden, um die korrekte Funktion zu garantieren.

Literatur

- [1] Markus Tröger. *Windmühle — Boarding-Time — boarding-time.de*. <https://www.boarding-time.de/reiseziele/griechenland/kos/antimachia/windmuehle-antimachia/>. [Accessed 19-02-2024]. 2017.
- [2] Kilian Müller. *Projekt Sailwind 4: Grüner Strom aus historischen Windmühlen — industr.com*. <https://www.industr.com/de/projekt-sailwind-4-gruener-strom-aus-historischen-windmuehlen-2719408>. [Accessed 19-02-2024]. 2023.
- [3] IFM Electronic GmbH. *Infocard Induktive Sensoren*. <https://www.ifm.com/us/en/product/IFS204?tab=documents>. [Accessed 15-02-2024]. 2018.
- [4] Dunkermotoren GmbH. *BG 45 SI Betriebsanleitung*. https://www.dunkermotoren.de/-/media/project/oneweb/oneweb/dunkermotoren/downloads/pdf/downloads/manuals/brushless-dc-motors/9_bg_45_si.pdf?la=de-de&revision=1b9ccf15-1d22-4953-918e-31635c338770. [Accessed 15-02-2024]. 2022.
- [5] STMicroelectronics NB. *RM0090 Reference manual*. <https://www.st.com/en/microcontrollers-microprocessors/stm32f439zi.html#documentation>. [Accessed 15-02-2024]. 2024.
- [6] STMicroelectronics NB. *DS9484 STM32F437xx STM32F439xx*. <https://www.st.com/en/microcontrollers-microprocessors/stm32f439zi.html#documentation>. [Accessed 15-02-2024]. 2023.
- [7] Texas Instruments. *TMCS1101-Q1 AEC-Q100*. <https://www.ti.com/lit/ds/symlink/tmcs1101-q1.pdf?HQS=dis-dk-null-digikeymode-dsf-pf-null-ww&ts=1708536991851>. [Accessed 20-02-2024]. 2021.
- [8] Schurter Electronic Components Inc. *MCS 18 Front Datasheet*. <https://www.digikey.com/en/htmldatasheets/production/2706140/0/0/1/mcs-18-front-datasheet>. [Accessed 15-02-2024]. 2019.

-
- [9] MESA Systemtechnik GmbH. *WSWD SONIC Anemometer Bedienungsanleitung ab Firmwareversion V3.00*. <https://mesa-systemtechnik.de/Bedienungsanleitung/WSWD-HM-Bedienungsanleitung.pdf>. [Accessed 15-02-2024]. 2017.
- [10] Thomas Kugelstadt. *RS-485: Passive failsafe for an idle bus*. <https://www.ti.com/lit/an/slyt324/slyt324.pdf>. [Accessed 15-02-2024]. 2009.
- [11] IFM Electronic GmbH. *Bedienungsanleitung Optischer Abstandssensor OGD580*. <https://www.ifm.com/us/en/product/OGD580?tab=documents>. [Accessed 15-02-2024]. 2019.
- [12] IFM Electronic GmbH. *Betriebsanleitung Konverter IO-Link Sensor – 4...20 mA EIO104*. <https://www.ifm.com/us/en/product/EIO104?tab=documents>. [Accessed 15-02-2024]. 2021.

A Schaltpläne

A.1 Schaltplan Übersicht

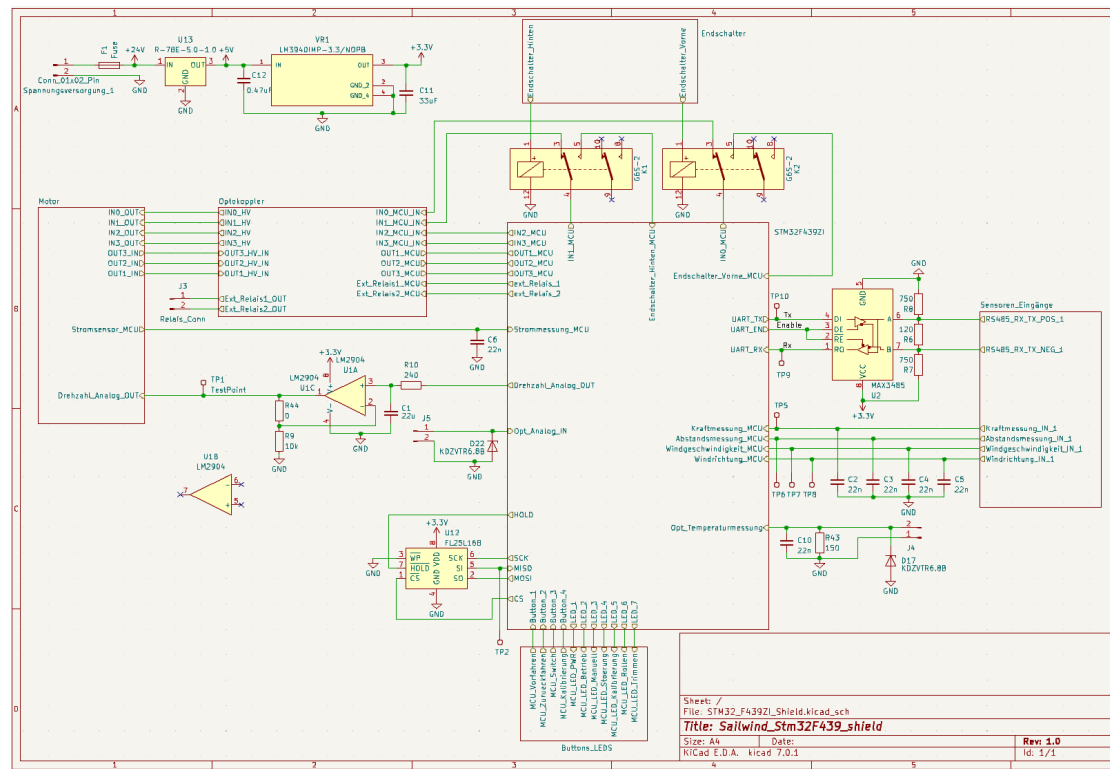


Abbildung 24: Schaltplan Übersicht

A.2 Schaltplan STM32F439 Gruppe

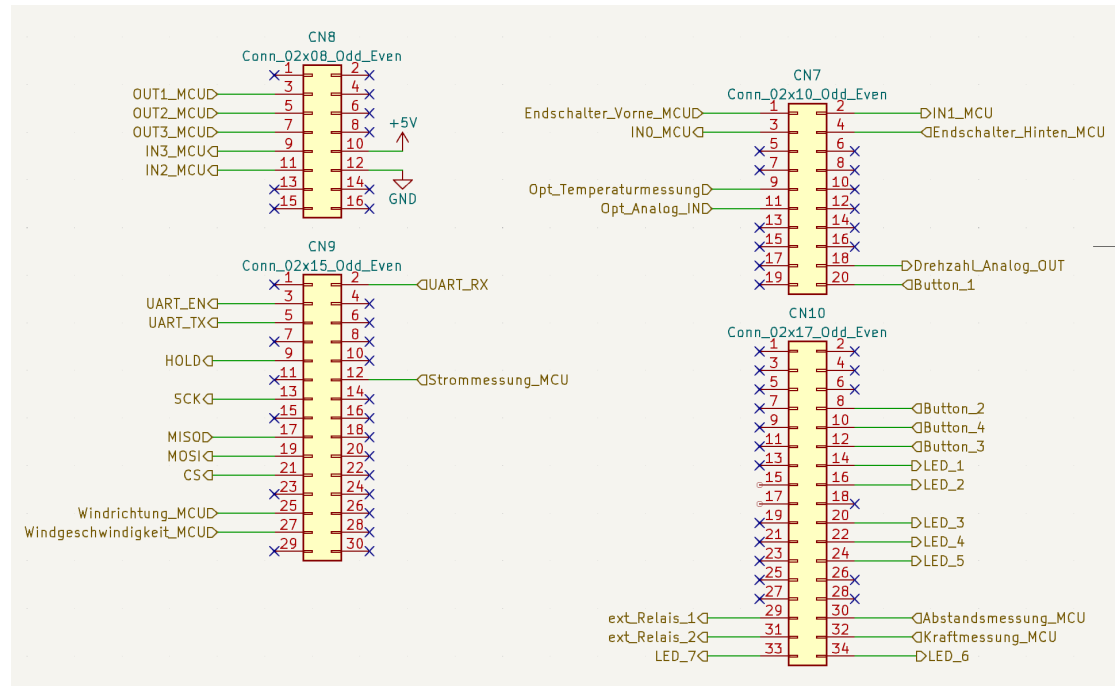


Abbildung 25: Schaltplan der STM32F439 Gruppe

A.3 Schaltplan Buttons und LED Gruppe

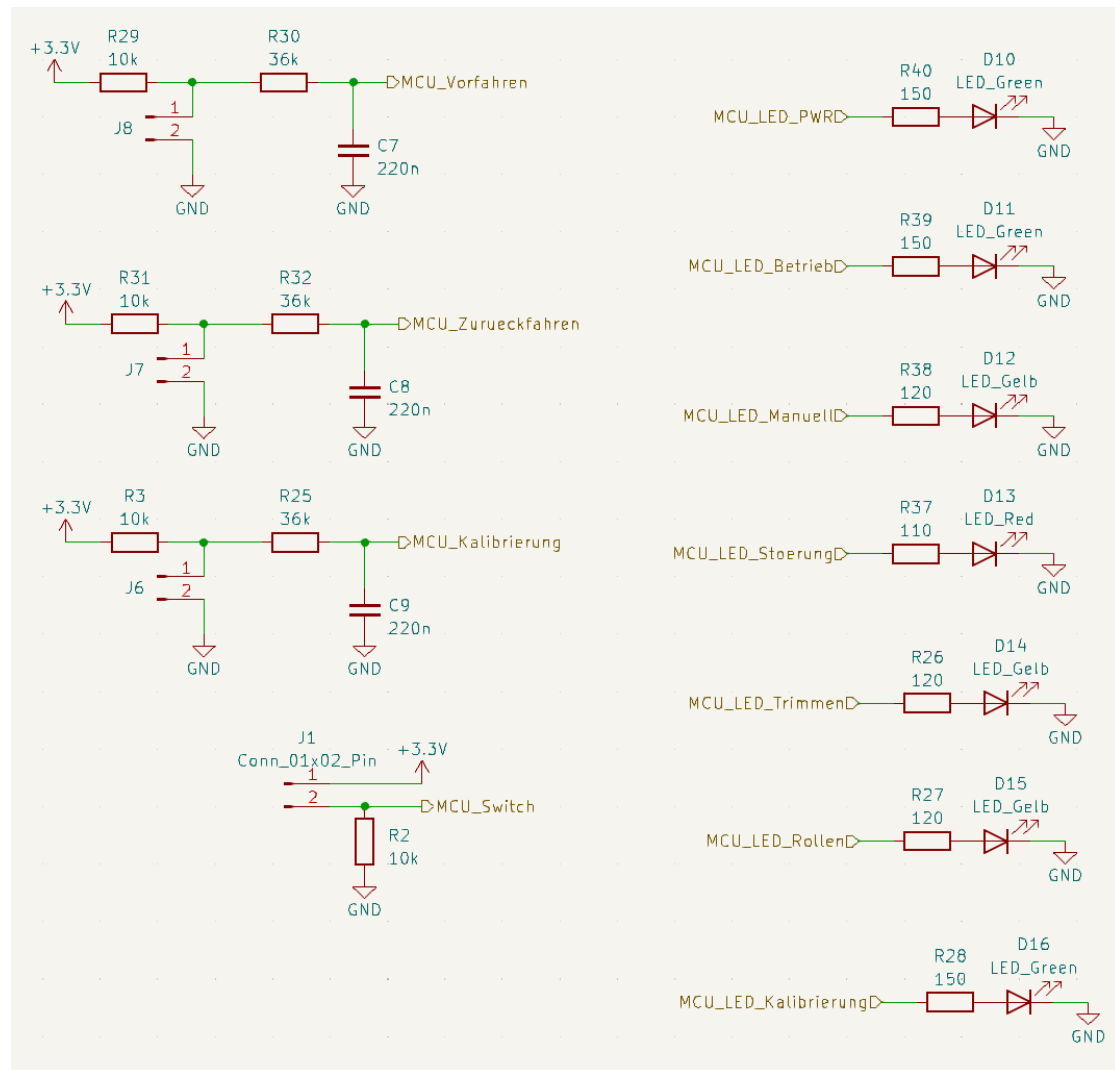


Abbildung 26: Schaltplan der Buttons und LED Gruppe

A.4 Schaltplan Sensoren Gruppe

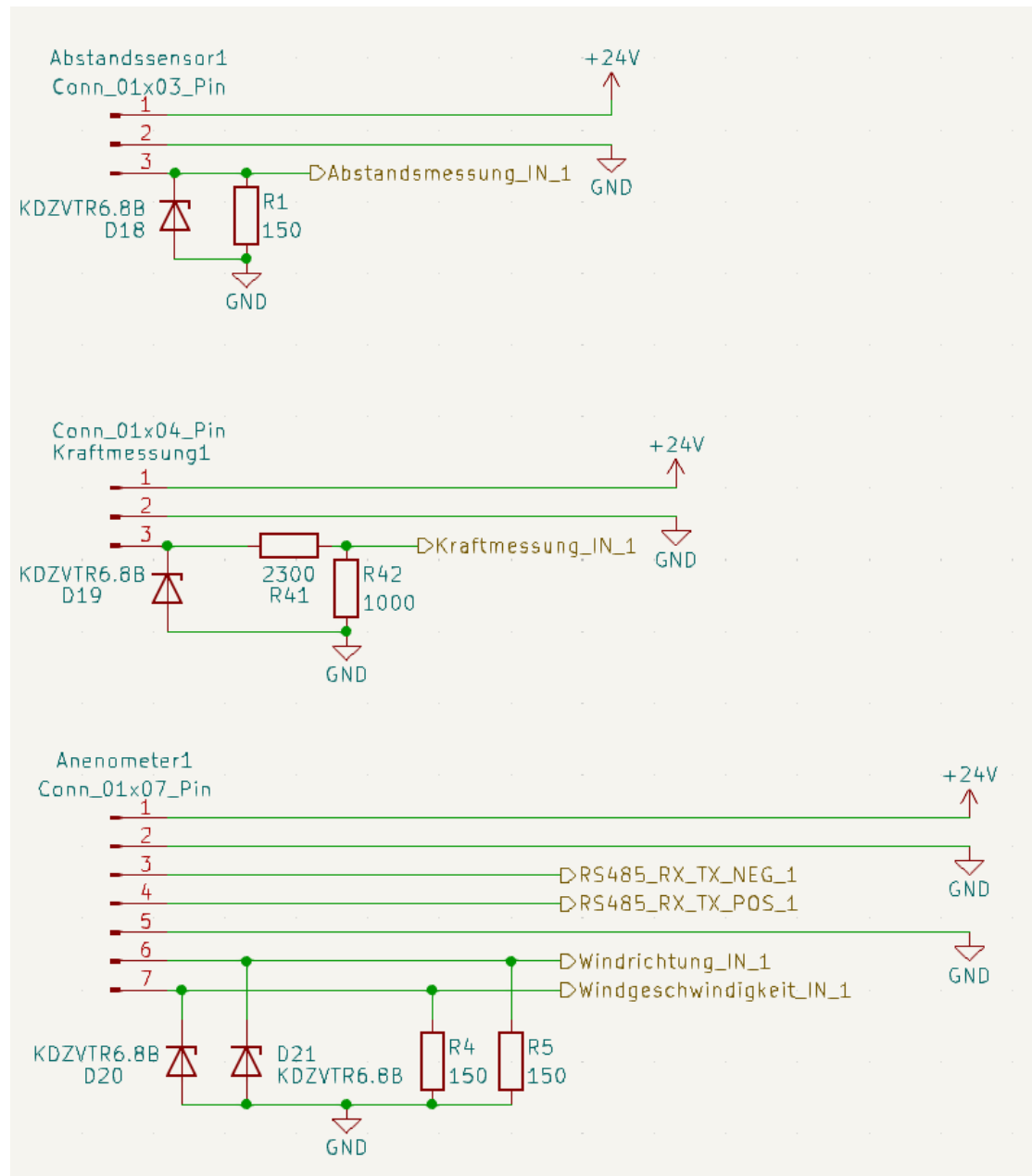


Abbildung 27: Schaltplan der Sensoren Gruppe

A.5 Schaltplan Motor Gruppe

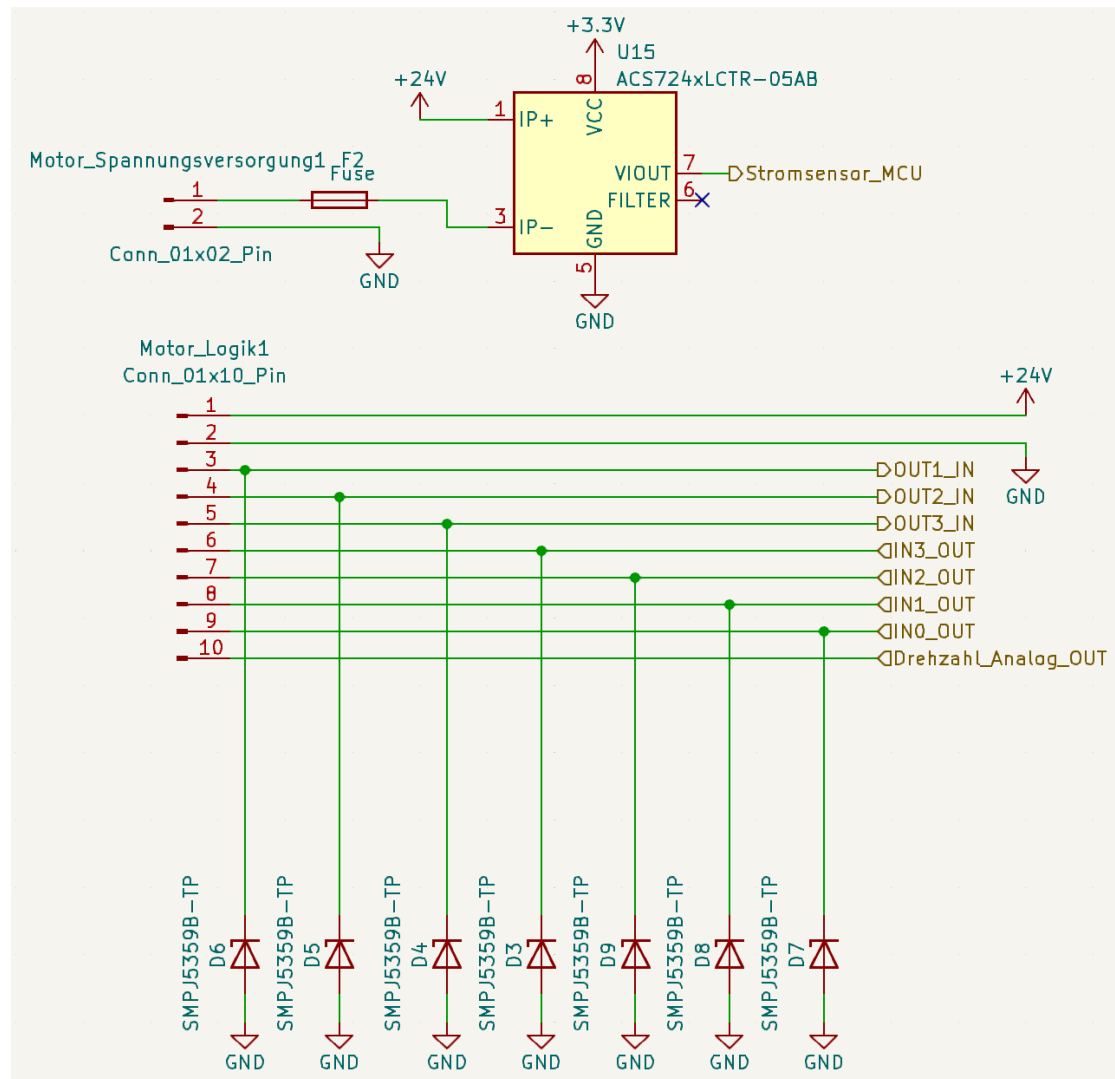


Abbildung 28: Schaltplan der Motor Gruppe

A.6 Schaltplan Endschalter Gruppe

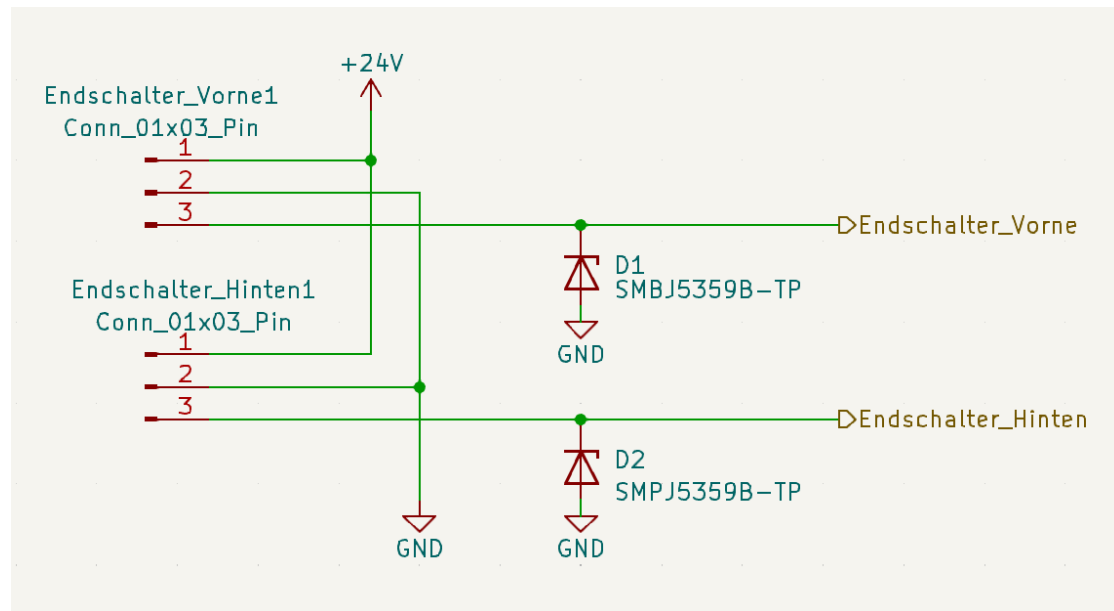


Abbildung 29: Schaltplan der Endschalter Gruppe

B Bauteilliste

Bauteilbezeichnung	Bauteilart	Anzahl
Schurter TASTER 1104 GN	Button, Grün	1
Schurter TASTER 1104 SW	Button, Schwarz	2
LM3940IMP-3.3/NOPBCT-ND	Festspannungsregler	1
FM25L16B-GTRCT-ND	F-RAM	1
C1Q 4	Fuse 1206, 4A	1
3413.0223.22	Fuse 1206, 5A	1
RP1465	Gehäuse	1
R-78E5.0-0.5	DC/DC Wandler	1
Phoenix Contact 1984617	Schraubterminal 2 Fach 3.5mm	2
OSTVN02A150	Schraubterminal 2 Fach 2.54mm	3
OSTVN03A150	Schraubterminal 3 Fach 2.54mm	4
TE Connectivity 282834-7	Schraubterminal 7 Fach 2.54mm	1
OSTVN10A150	Schraubterminal 10 Fach 2.54mm	1
LM2904QS-13DICT-ND	Operationsverstärker	1
MAX3485CSA+CT-ND	RS-485 Converter	1
TMCS1101A3BQDRQ1CT-ND	Stromsensor	1
LTST-C150GKT	SMD LED Grün	3
LTST-C150KSKT	SMD LED Gelb	3
LTST-C230KRKT	SMD LED Rot	1
LTST-C230KFKT	SMD LED Orange	1
LTV-817S	Optokoppler	9
G6S-2F 24DC	Relais, 24V	2
KDZVTR24BCT-ND	Zenerdiode, 24V	9
Schurter FRONTSATZ	Button, Montagesatz	3
C6000ALBB	Schalter	1
C6000ALBB-1229W	Schalter, gekennzeichnet	1
SL 2X10G SMD2,54	Pin-Header, SMD	1
SL 2X50G 2,54	Pin-Header	1
MEN 1216.1001	LED-Lichtleiter	8
KDZVTR6.8B	Zenerdiode, 6.8V	4
IPG-2227	Kabelverschraubung 3mm-6mm	4
IPG-222135	Kabelverschraubung 6mm-12mm	4

Tabelle 6: Bauteilliste Teil 1

Widerstände SMD1206	Anzahl
150 Ω	7
10k Ω	5
120 Ω	4
750 Ω	2
240 Ω	1
700 Ω	6
11k Ω	6
2,2k Ω	3
6,8k Ω	3
36k Ω	3
110 Ω	1
2,3k Ω	1
1k Ω	1
Kondensatoren SMD1206	
22 μ F	1
22nF	6
220nF	3
33 μ F	1
0.47 μ F	1

Tabelle 7: Bauteilliste Teil 2

C Valide URL und JSON Formate

C.1 GET-Anfragen

C.1.1 GET /data

```
{
    "error": <(int) error_code>,
    "operating_mode": <(int) 0/1 [manual/automatic]>,
    "localized": <(boolean)>,
    "sail_pos": <(int) pitch/roll [+/- 0...100 %]>,
    "wind": {
        "speed": <(int) speed[m/s]>,
        "direction": <(int) direction[degree]>
    },
    "current": <(int) current[mA]>
}
```

Listing 16: GET-Request 1

C.1.2 GET /data/status

```
{
    "error": <(int) error_code>,
    "operating_mode": <(int) 0/1 [manual/automatic]>,
    "localized": <(boolean)>,
}
```

Listing 17: GET-Request 2

C.1.3 GET /data/adjustment

```
{
    "sail_pos": <(int) pitch/roll [+/- 0...100 %]>
}
```

Listing 18: GET-Request 3

C.1.4 GET /data/sensors

```
{
  "wind": {
    "speed": <(int) speed[m/s]>,
    "direction": <(int) direction[degree]>
  },
  "current": <(int) current[mA]>
}
```

Listing 19: GET-Request 4

C.1.5 GET /data/settings

```
{
  "max_rpm": <(int) 400 - 2000>,
  "max_distance_error": <(int) 5 - 50>
}
```

Listing 20: GET-Request 5

C.2 PUT-Anfragen

C.2.1 PUT /data/status/error

```
{
  "error": <(int) error_code>,
}
```

Listing 21: PUT-Request 1

C.2.2 PUT /data/adjustment

```
{  
    "sail_pos": <(int) pitch/roll [+/- 0...100 %]>  
}
```

Listing 22: PUT-Request 2

C.2.3 PUT /data/status/operating_mode

```
{  
    "operating_mode": <(int) 0/1 [manual/automatic]>  
}
```

Listing 23: PUT-Request 3

C.2.4 PUT /data/settings

```
{  
    "max_rpm": <(int) 400 - 2000>,  
    "max_distance_error": <(int) 5 - 50>  
}
```

Listing 24: PUT-Request 4