

BIP Kreativitätsgymnasium Leipzig

**BESONDERE LERNLEISTUNG
(BeLL)**

Abiturjahrgang 2024/2025

**Entwicklung einer Windensteuerung
zur Untersuchung der Sonnenkorona
mittels eines Kastendrachens**

Fachpraktische Arbeit

Max Rothe

Innenbetreuung:
Michael Kripfganz,
Fachlehrer für Mathematik und Physik

Außenbetreuung:
Benedikt Justen,
Justen Engineering Services

Abgabedatum: 20.12.2024

Inhaltsverzeichnis

1 Einleitung	3
2 Theoretische Grundlagen	5
2.1 Arduino-Entwicklungsumgebung	5
2.1.1 Hardware	5
2.1.2 Software	7
2.2 DMS-Messtechnik	8
2.2.1 Folien-DMS	9
2.2.2 Wheatstone'sche Brückenschaltung	10
2.3 Formeln zur Auslegung der Messachse	12
3 Planung der Winde	15
3.1 Anforderungen an die Winde	15
3.2 Grundlegendes Konzept	15
3.3 Entwicklung der finalen Windenversion	17
4 Entwicklung der Messachse	19
4.1 Planung der Messachse	19
4.2 Fertigung der Messachse	21
5 Entwicklung der Windensteuerung	25
5.1 Aufbau der Steuerung	25
5.2 Auswahl der Komponenten	27
5.3 Installation der Komponenten	30
5.4 Programmcode zur Steuerung	33
6 Zusammenfassung und Ausblick	48
Quellenverzeichnis	50
Literaturverzeichnis	50
Internetquellen mit Autor	51
Internetquellen ohne Autor	51
Abbildungsverzeichnis	52
Selbstständigkeitserklärung	54

1 Einleitung

Am 8. April 2024 ereignete sich über Nordamerika eine totale Sonnenfinsternis. Sie konnte sowohl vom Atlantischen Ozean als auch vom Festland (Mexiko, USA und Kanada) beobachtet werden. Ein Team von Wissenschaftlern unter der Leitung von Shadia Habal, Professorin am Institut für Astronomie (IfA) der Universität Hawaii (UH), genannt „Solar Wind Sherpas“, nutzt diese seltenen Ereignisse, um die „Atmosphäre“ der Sonne – die Korona – zu untersuchen. (Vgl. Uppal 2024)

Von ihr strömt der Sonnenwind, ein konstanter Teilchenstrom aus Protonen und Elektronen, in den Weltraum ab. Kommt es zu einer Sonneneruption, ist dieser in einem begrenzten Gebiet kurzzeitig stärker als üblich. Das stellt eine ernstzunehmende Gefahr für die Sicherheit von Astronauten dar und kann negative Auswirkungen auf die Funktionstüchtigkeit von Satelliten haben.¹

Ziel der Forscher ist es, Informationen über die Entstehung und das Verhalten des Sonnenwindes zu sammeln. Das ist jedoch nur möglich, wenn die direkte Strahlung der Sonnenoberfläche (Photosphäre) verdeckt wird, wie im Fall einer totalen Sonnenfinsternis. Stand Mai 2024 konnte das Forscherteam auf diese Weise bereits 19 totale Sonnenfinsternisse beobachten. (Vgl. Uppal 2024)

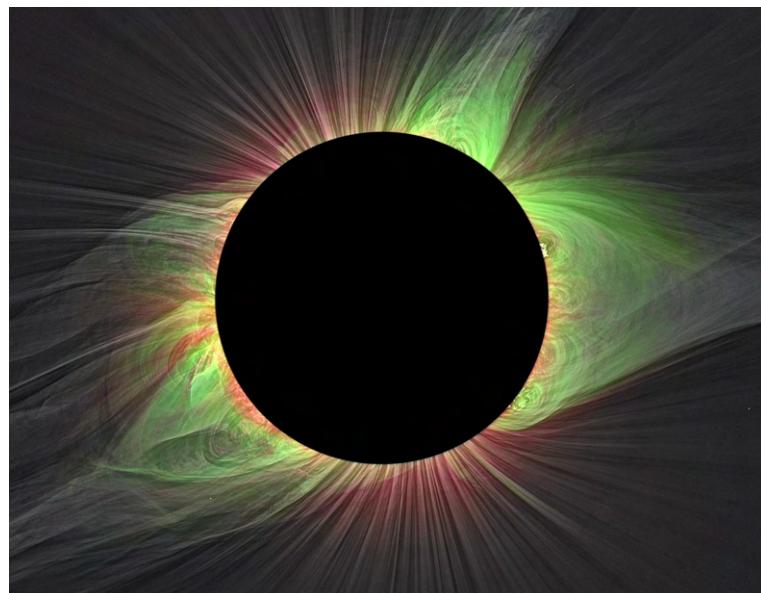


Abbildung 1: Aufnahme einer totalen Sonnenfinsternis

Jedoch konnten bei knapp der Hälfte der Expeditionen aufgrund dichter Bewölkung keine Daten gesammelt werden. Das ist nicht unerheblich, zumal die Vorbereitungen zur Beobachtung einer einzelnen, seltenen Sonnenfinsternis oft ein bis zwei Jahre in Anspruch nehmen und mit immensem Aufwand verbunden sind. (Vgl. Uppal 2024)

¹<https://www.mps.mpg.de/sonnenstuerme-sonnenaktivitaet-faq> [zuletzt überprüft am 16.07.2024]

Eine innovative und vielversprechende Lösung des Problems stellt das Projekt „KITE-BORNE PLATFORM FOR TOTAL SOLAR ECLIPSE SPECTROSCOPIC OBSERVATIONS OF THE CORONA AND THE SOURCES OF THE SOLAR WIND“ unter maßgeblicher Beteiligung von Benedikt Justen (Justen Engineering Services) dar. Dabei wird ein Spektrometer zusammen mit weiteren Messinstrumenten an einem großflächigen Kastendrachen angebracht, welcher während der Sonnenfinsternis über die niedrigen bis mittleren Wolkenschichten aufsteigt. In einer Machbarkeitsstudie gilt es zu zeigen, dass Sonnenbeobachtungen von einem Drachen aus prinzipiell möglich sind. Damit könnte die Sonnenkorona im Idealfall trotz tiefliegender Bewölkung untersucht werden, sofern keine höher liegenden Wolkenschichten vorhanden sind. (Vgl. Justen 2025 (Vorabversion))²

Das Konzept wurde bereits während der totalen Sonnenfinsternis über dem nordwestlichen Teil Australiens im Jahr 2023 erfolgreich getestet. Jedoch gab es neben technischen Schwierigkeiten mit der Objektverfolgung ebenfalls das Problem, die lange Schnur des Kastendrachens effektiv auf- und abzuwickeln. (Vgl. Uppal 2024)

Zur Lösung dieses Problems wurde im Rahmen der Expedition „Solar Eclipse 2024“ eine motorisierte Winde (siehe Abbildung 2: **Winde während der Expedition „Solar Eclipse 2024“**) unter der Leitung von Benedikt Justen geplant und angefertigt. Die Besondere Lernleistung befasst sich mit der Entwicklung der mikrocontrollerbasierten Windensteuerung und geht ebenfalls auf die Planung des Windensystems ein. Es wird der eigene Anteil am Entwicklungs- und Fertigungsprozess beschrieben und durch Bildmaterial veranschaulicht.



Abbildung 2: Winde während der Expedition „Solar Eclipse 2024“

²Die Quelle stellt eine Vorabversion des wissenschaftlichen Papers dar, welches bis zur Vollendung dieser Arbeit noch nicht publiziert wurde. Die geplante Veröffentlichung erfolgt Anfang 2025 im Astrophysical Journal Letters.

2 Theoretische Grundlagen

2.1 Arduino-Entwicklungsumgebung

Arduino ist der Name einer weit verbreiteten *Physical-Computing*-Plattform, die 2005 von den Italienern Massimo Banzi, David Cuartielles, Tom Igoe, Gianluca Martino und David Mellis veröffentlicht wurde.

Unter *Physical-Computing* versteht man eine Anwendung der Informatik, welche in direktem Kontakt mit ihrer physischen Umwelt steht. Ereignisse aus der unmittelbaren Umgebung können mithilfe von Sensoren und anderen elektronischen Steuerelementen wahrgenommen, verarbeitet und gegebenenfalls beantwortet werden. Ziel der Entwickler war es, eine möglichst simple Mikrocontroller-Plattform zu schaffen, welche dennoch vielfältige Anwendungsmöglichkeiten bietet und dabei in großem Maß für die Öffentlichkeit zugänglich ist.

Hardwareseitig wurde dies durch die Entwicklung und Veröffentlichung zahlreicher Arduino-Boards umgesetzt, softwareseitig durch die Bereitstellung der Arduino *Integrated Development Environment* (kurz *IDE*).

Das gesamte Projekt wurde unter einer *Creative-Commons*-Lizenz veröffentlicht, was eine einfache Nutzung sowohl für Privatpersonen als auch für Unternehmen ermöglicht. (Vgl. Schreiter 2019, S. 12 ff.)

Anwendung findet die Arduino-Plattform vor allem im Hobby- und Bastlerbereich, jedoch können auch komplexere Steuerungen über die Plattform realisiert werden. Konkrete Anwendungsbeispiele reichen von selbstgebauten Drum-Computern, über Lichtsteuerungen in Privathaushalten, bis hin zu ferngesteuerten Autos oder wie im Fall dieser Besonderen Lernleistung, einer komplexen Windensteuerung.

Wichtig ist außerdem zu erwähnen, dass sich Arduino seit seiner Entwicklung einer stetig wachsenden Community aus Bastlern, Studenten, Künstlern und Programmierern erfreut, die sich in zahlreichen Online-Foren organisieren. Dort können Nutzer über eigene Projekte berichten, auf Fehler anderer hinweisen und gleichzeitig selbst Hilfe erhalten. Da Beiträge im offiziellen Arduino-Forum über längere Zeit hinweg gespeichert werden, fungiert es zudem als umfassende Datenbank für Projekte und Lösungsansätze.³

2.1.1 Hardware

Im Kontext der Arduino-Entwicklungsumgebung bezieht sich der Begriff Hardware auf zahlreiche Komponenten, darunter Arduino-Boards, Shields, Kits, Erweiterungsplatten (engl. *breakout boards*) sowie weiteres Zubehör. Erstgenannte sollen in diesem Kapitel im Fokus stehen.

Ein Arduino-Board besteht aus einem Mikrocontroller als zentralem Bauelement, das über Ein- und Ausgänge mit anderen Komponenten verbunden ist. Obwohl die spezifischen Bauelemente je nach Baureihe und Version stark variieren, bleibt das Grundprinzip eines Arduino-Boards unverändert.

³<https://www.arduino.cc/en/Guide/Introduction> [zuletzt überprüft am 16.07.2024]

Über die Eingangspins des Boards gelangen elektrische Signale zum Mikrocontroller, der diese verarbeitet. Gleichzeitig können über die Ausgangspins Signale ausgegeben werden, beispielsweise um andere Bauteile anzusteuern.

An dieser Stelle ist anzumerken, dass elektronische Bauteile wie Sensoren, Motoren oder Erweiterungsplatten vollständig lötfrei über ein Stecksystem mit der Hauptplatine verbunden werden können. Dies vereinfacht das Arbeiten in der Arduino-Entwicklungs-umgebung und macht es besonders praxisnah. Für langlebigere oder robustere Projekte können Lötverbindungen zwischen den Komponenten nachgerüstet werden. (Vgl. Schreiter 2019, S. 15 ff.)

Ein Arduino ist weder auf ein herkömmliches Betriebssystem, noch auf ein Dateisystem angewiesen und reagiert dadurch zuverlässig und schnell. Ein Befall durch Viren oder andere Malware ist weitgehend ausgeschlossen. (Vgl. Schreiter 2019, S. 14 ff.)

Über die Zeit hat sich die Variante UNO R3 aus der „*Classic Family*“ als Standardmodell etabliert. Diese basiert auf dem *ATmega328P*-Mikrocontroller von „*Microchip AVR*“ und verfügt über insgesamt 14 digitale Ein- und Ausgangspins sowie 6 analoge Eingangspins. Der Arduino UNO R3 gilt als besonders benutzerfreundlich und ist aufgrund seiner umfassenden Dokumentation ideal für Anfänger geeignet. Außerdem orientieren sich viele der verfügbaren Arduino-Shields am Formfaktor der *UNO*-Reihe und auch ein Großteil der Bibliotheken (siehe Abschnitt **2.1.2 Software**) wurde speziell für sie entwickelt. Das macht ihre Nutzung besonders unkompliziert.

Neben der „*Classic Family*“ haben sich im Laufe der Jahre zahlreiche weitere Modellreihen etabliert, etwa die „*Nano Family*“ oder die „*Mega Family*“. (Vgl. Schreiter 2019, S. 15 ff.)

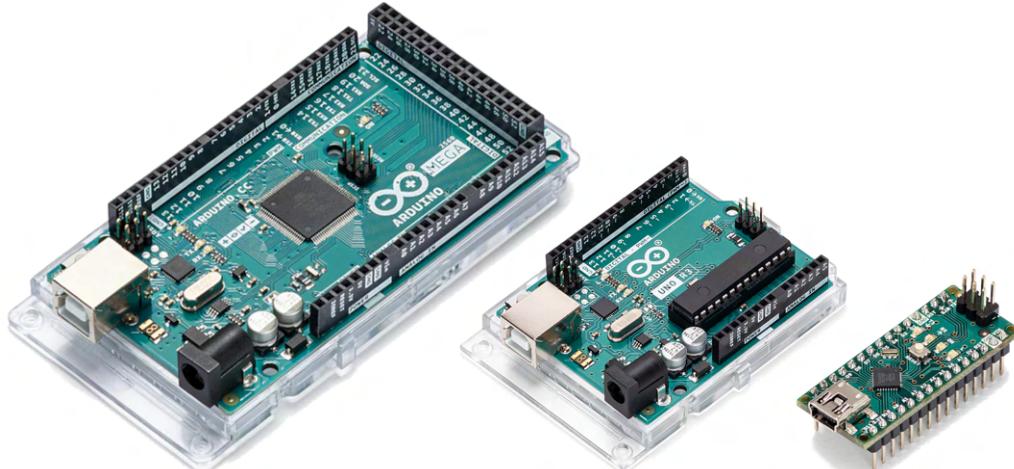


Abbildung 3: Gegenüberstellung bekannter Arduino-Modellreihen

In der Abbildung sind von links nach rechts die Arduino-Modelle **Mega 2560 Rev3** („*Mega Family*“), **UNO R3** („*Classic Family*“) und **Nano** („*Nano Family*“) abgebildet. Sie unterscheiden sich teilweise stark in Faktoren wie Rechenleistung, Formfaktor und Pinout, können jedoch alle über die Arduino *IDE* in das Projekt eingebunden und gegebenenfalls untereinander ausgewechselt werden.

Unter Arduino-Shields versteht man Erweiterungsplatinen, die beispielsweise zur Steuerung mehrerer Schrittmotoren verwendet werden und aufgrund ihres Formfaktors direkt auf den Arduino gesteckt werden können. Die Anschlusspins dienen dabei sowohl als elektrische als auch mechanische Verbindung und verhindern ein unbeabsichtigtes Ablösen des Shields. (Vgl. Schreiter 2019, S. 19 ff.)

Zuletzt sind hardwareseitig die zahlreichen *breakout boards* zu erwähnen, welche zwar nicht von *Arduino LLC* selbst entwickelt wurden, für die meisten Projekte jedoch unerlässlich sind. Diese kleinen Platinen enthalten elektronische Bauelemente wie Sensoren oder Motortreiber samt ihrer Schaltungen und lassen sich unkompliziert in Projekte integrieren. Im Gegensatz zu Arduino-Shields sind sie häufig kompakter und auf spezifische Anwendungen ausgelegt. Zudem sind sie nicht ausschließlich auf Arduino-Boards zugeschnitten, sondern können auch mit anderen Mikrocontrollern verwendet werden. Die Verbindung erfolgt oftmals per Stecksystem mithilfe eines Breadboards.⁴

2.1.2 Software

Die eigens von *Arduino LLC* entwickelte Software, die *Arduino IDE*, ermöglicht es den Nutzern, Code in den Programmiersprachen C und C++ zu schreiben und diesen auf ein Arduino-Board zu laden. Sie ist im Sinne der Entwickler benutzerfreundlich gestaltet und bietet trotzdem eine Vielzahl an nützlichen Funktionen, die das Programmieren erleichtern. So prüft sie den Programmcode auf Fehler und visualisiert empfangene Messwerte über einen seriellen Monitor.⁵

Darüber hinaus erleichtert das integrierte *Bibliotheksmanger-Tool* die Nutzung von Code anderer Entwickler im eigenen Programm. Die Kommunikation mit angeschlossenen Bauelementen oder die Auswertung empfangener Messwerte lassen sich auf diese Weise stark vereinfachen. (Vgl. Schreiter 2019, S. 27 ff.)

Ein Programm, im Arduino-Kontext auch Sketch genannt, besteht aus zwei grundlegenden Teilen: der *setup*-Funktion, welche einmalige Einstellungen bei der Initialisierung des Programms vornimmt und der *loop*-Funktion, die nach der Initialisierung kontinuierlich aufgerufen wird. In der *setup*-Funktion werden grundlegende Befehle ausgeführt, wie das Festlegen der Pins oder die Initialisierung der benötigten Module. Der eigentliche Code, der auf dem Arduino wiederholt ablaufen soll, wird in der *loop*-Funktion beschrieben. (Vgl. Schreiter 2019, S. 69 ff.)

⁴<https://www.programmingelectronics.com/what-is-a-breakout-board-for-arduino/> [zuletzt überprüft am 17.10.2024]

⁵<https://docs.arduino.cc/software/ide-v1/tutorials/Environment/> [zuletzt überprüft am 16.07.2024]

Dieser grundlegende Sketch zeigt, wie das periodische Blinken einer LED programmiert werden kann. Es handelt sich um den vielzitierten Blink-Sketch der vorinstallierten Arduino-Beispielreihe. Er soll als kurzes Praxisbeispiel für die Nutzung der Arduino *IDE* dienen:

```

1 // Pin für die interne LED des Arduino festlegen
2 int LED_PIN = 13;
3
4 void setup() {
5     // Definiere LED_PIN als Ausgang
6     pinMode(LED_PIN, OUTPUT);
7 }
8
9 void loop() {
10    // Schalte LED_PIN ein
11    digitalWrite(LED_PIN, HIGH);
12
13    // Halte das Program für 1000 Millisekunden an
14    delay(1000);
15
16    // Schalte LED_PIN aus
17    digitalWrite(LED_PIN, LOW);
18
19    // Halte das Program für 1000 Millisekunden an
20    delay(1000);
21 }
```

Programmcode 1: Arduino Blink-Sketch

Vor Beginn des eigentlichen Programms wird die LED zunächst als Pin 13 definiert. Das erlaubt den Zugriff auf die interne, fest verlötete LED der meisten Arduino-Boards. In der *setup*-Funktion kann die LED nun einmalig als Ausgang definiert werden. Dies verhindert die Neudefinition des Pins bei jeder Wiederholung des Programms und ermöglicht im folgenden Programmteil das Senden eines digitalen Signals an den Pin. Das geschieht in der *loop*-Funktion durch die Verwendung der Befehle *digitalWrite()* und *delay()*. Letzterer ist notwendig, um zu verhindern, dass die LED nach dem Ein- oder Ausschalten sofort wieder ihren Zustand ändert. Der *delay()*-Befehl pausiert das Programm für die angegebene Zeit (in Millisekunden).

2.2 DMS-Messtechnik

Ein Dehnmesstreifen (DMS) ist ein dehnbarer Widerstand, dessen Widerstandswert sich linear mit der mechanischen Verformung ändert. Anwendungsbereiche von DMS-Systemen finden sich in der experimentellen Spannungsanalyse, beispielsweise zur Dehnungsmessung an Maschinen, Konstruktionen oder anderen Bauteilen, sowie im Bereich des Messgrößenaufnehmerbaus.

Aus dieser Vielzahl von Anwendungen resultiert eine breite Auswahl an Bauformen und Abmessungen für Dehnmesstreifen. Sie lassen sich unter anderem in kapazitive, piezoelektrische und Folien-DMS unterscheiden. Letztere werden im folgenden Abschnitt ausführlich behandelt. (Vgl. Hoffmann 1987, S. 42 ff.)

2.2.1 Folien-DMS

Die verbreitetste Bauform metallischer Dehnungsmessstreifen ist der Folien-DMS. Er kann als Messgitterfolie aus Widerstandsdraht aufgefasst werden und besteht aus einer dünnen Trägerfolie, auf der eine feine Metall-Leiterbahn mäanderförmig aufgebracht ist. Wird diese Leiterbahn gestreckt, verlängert sie sich, während ihr Querschnitt abnimmt. Beide Effekte bewirken eine Erhöhung des Gesamtwiderstands R des DMS, da:

$$R = \frac{\rho \cdot l}{A} \quad [\Omega]. \quad (1)$$

Umgekehrt führt eine Stauchung zur Vergrößerung des Querschnitts und einer Verkürzung der Leiterbahn, was den Gesamtwiderstand R verringert.

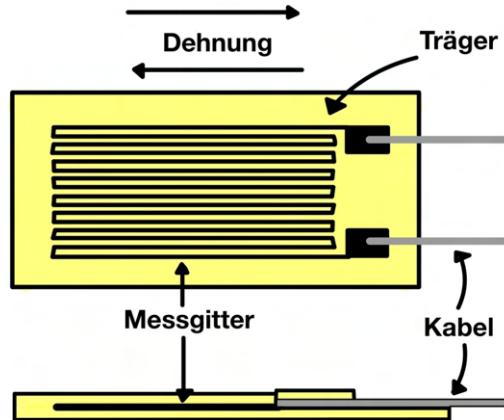


Abbildung 4: Aufbau eines Folien-DMS

Für die Dehnungsmessung wird der DMS an das entsprechende Bauteil geklebt. Formänderungen des Trägers werden dabei auf den Dehnungsmessstreifen übertragen und führen in diesem zu einer Änderung des Widerstands. (Vgl. Hoffmann 1987, S. 144 ff.)

Wird diese Widerstandsänderung ΔR berechnet, so gilt die Formel:

$$\frac{\Delta R}{R} = k \cdot \frac{\Delta l}{l} \quad (2)$$

$$\Rightarrow \Delta R = k \cdot \frac{\Delta l}{l} \cdot R \quad [\Omega]. \quad (3)$$

Dabei bezeichnet k , der sogenannte k -Faktor, die Empfindlichkeit des Dehnmessstreifens. Der Bruch $\frac{\Delta l}{l}$ gibt die relative Längenänderung der Metall-Leiterbahn an und entspricht der Dehnung ϵ des DMS. Durch diese Substitution lässt sich die Gleichung (3) vereinfachen zu:

$$\Delta R = k \cdot \epsilon \cdot R \quad [\Omega]. \quad (4)$$

Üblicherweise arbeiten die meisten Folien-DMS mit einem Basiswiderstand R von 120Ω beziehungsweise 350Ω . So ergibt sich die maximale Widerstandsänderung ΔR eines Folien-DMS mit Konstantan-Leiterbahn ($k = 2,05$ und $\epsilon_{\max} = 0,02$) von:

$$\Delta R = 2,05 \cdot 0,02 \cdot 350 = 14,35 \Omega.$$

In der Praxis tritt jedoch selten die maximale Dehnung oder Stauchung des DMS auf, weshalb die tatsächliche Widerstandsänderung meist deutlich unter dem berechneten Wert von $\Delta R = 14,35 \Omega$ liegt. Die relative Änderung des Gesamtwiderstandes $\frac{\Delta R}{R}$ ist daher sehr gering und bedarf einer Wheatstone'sche Brückenschaltung zur störungsfreien Messung. (Vgl. Ewald 2021)

2.2.2 Wheatstone'sche Brückenschaltung

Die Wheatstone'sche Brückenschaltung ermöglicht die präzise Messung von geringen relativen Widerstandsänderungen $\frac{\Delta R}{R}$ in einem Bereich von $10 \cdot 10^{-2}$ bis $10 \cdot 10^{-4}$. Sie besteht aus jeweils zwei in Reihe geschalteten Widerständen, R_1 und R_2 beziehungsweise R_3 und R_4 , die als Spannungsteiler fungieren. Diese beiden Spannungsteiler sind parallel geschaltet. Wird eine Brückenspeisespannung U_B angelegt, teilt sich diese im Verhältnis der Widerstände auf. (Vgl. Hoffmann 1987, S. 144 ff.)

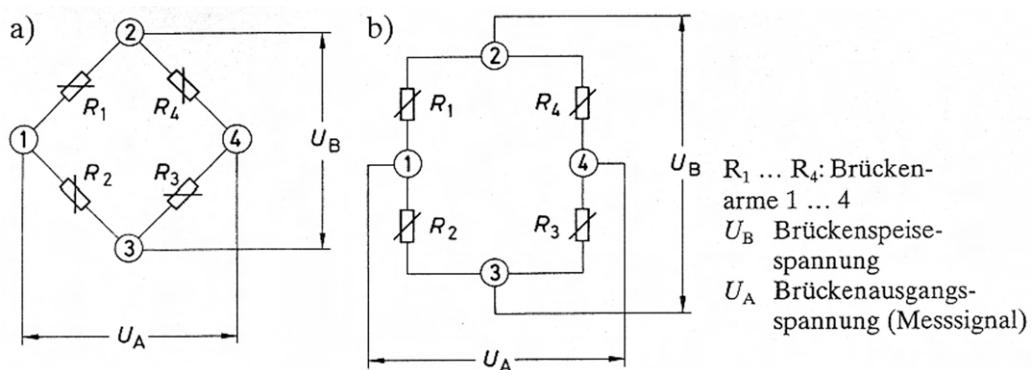


Abbildung 5: Wheatstone'sche Brückenschaltung

a) und b) zeigen unterschiedliche Darstellungsform der Wheatstone'sche Brückenschaltung

Die Brücke ist ausgeglichen, wenn die Brückenausgangsspannung $U_A = 0 \text{ V}$ beträgt. Das ist der Fall wenn:

$$\frac{R1}{R2} = \frac{R3}{R4}. \quad (5)$$

Eine Voraussetzung für die Verwendung in der DMS-Technik ist, dass die Widerstände auf jeder Brückenhälfte den gleichen Betrag besitzen. Somit gilt: $R1 = R2$ und $R3 = R4$. Setzt man für $R1$ einen zunächst unbelasteten Dehnmessstreifen ein und nutzt für $R2$ einen Ergänzungswiderstand gleich dem Betrag des verwendeten DMS-Basiswiderstands, ist die Brücke abgeglichen. Es wird lediglich die Brückenausgangsspannung gemessen, welche durch Dehnung oder Stauchung des DMS entsteht. Eine solche Brücke mit nur einem verwendeten DMS wird auch Viertelbrücke genannt.

Zur Reduktion von Grundrauschen und zur Signalverstärkung werden häufig mehrere DMS eingesetzt. Abhängig von der Anzahl der eingesetzten DMS unterscheidet man Viertel-, Halb- (Diagonal-) und Vollbrücken (siehe Abbildung 6: **Varianten der Wheatstone'schen Brückenschaltung**). Der Grundsatz lautet: Je mehr DMS eingesetzt werden, desto stärker das resultierende Signal, da alle Widerstandsänderungen addiert werden. Gleichzeitig werden bestimmte Störungen wie Temperaturänderungen besser kompensiert, da symmetrische Änderungen in den Widerständen das Messsignal U_A nicht beeinflussen. (Vgl. Hoffmann 1987, S. 144 ff.)

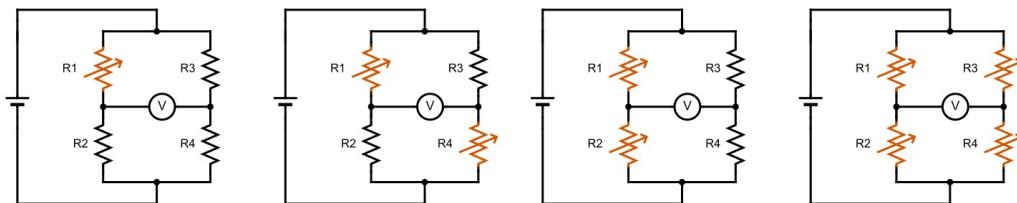


Abbildung 6: Varianten der Wheatstone'schen Brückenschaltung

2.3 Formeln zur Auslegung der Messachse

Um den erforderlichen Radius der Messachse in Abschnitt **4.1 Planung der Messachse** zu berechnen, wurde sie als einseitig eingespannter Biegebalken mit rundem Querschnitt angenähert.

Wirkt auf den einseitig eingespannten, langen, dünnen Balken aus einem linear-elastischem Material ein Biegemoment M_b quer zur Bauteilachse, führt dies aufgrund der entstehenden Biegespannungen zu einer Durchbiegung. Auf einer Seite wird das Material gedehnt, auf der anderen gestaucht. Die Zug- und Druckspannungen sind im Bereich der äußeren Bauteilränder (den sogenannten Randfasern) am größten und nehmen linear zur Mitte des Querschnitts hin ab. Genau dort, wo das Material weder gedehnt noch gestaucht wird, liegt die neutrale Faser, entlang welcher keinerlei Spannungen auftreten. Bei symmetrischen Querschnitten verläuft die neutrale Faser stets durch den Schwerpunkt der Schnittfläche. Im Falle des runden Querschnitts der Messachse ist das der Mittelpunkt der Kreisfläche beziehungsweise Grundfläche des Zylinders.⁶

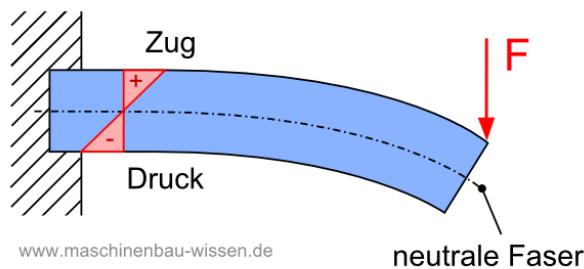


Abbildung 7: Biegeverhalten eines punktuell belasteten, einseitig eingespannten Balkens

Die Biegespannung σ_b beschreibt dabei die Spannungsverteilung im Material infolge des Biegemoments M_b . Sie wird aus dem Quotienten des Biegemoments M_b sowie dem Widerstandsmoment W des Balkens berechnet. Das Biegewiderstandsmoment gibt an, wie groß der Widerstand des Bauteils gegen die Biegung ist. Es ist allein von der Geometrie des jeweiligen Bauteils abhängig. Daraus resultiert folgende Formel:

$$\sigma_b = \frac{M_b}{W} \quad [\text{N/m}^2]. \quad (6)$$

⁶<https://www.maschinenbau-wissen.de/skript3/mechanik/balken-biegung/209-biegung-berechnen> [zuletzt überprüft am 22.11.2024]

Für den simplen Belastungsfall eines einseitig eingespannten Balken kann das Biegemoment M_b aus dem Produkt der punktuell auf den Balken aufgebrachten Kraft F und dem Abstand l zwischen Festlager und Krafteinwirkungsstelle bestimmt werden. Des Weiteren ist das Widerstandsmoment W als Quotient aus dem Flächenträgheitsmoment I und dem größten Abstand der Randfaser zur neutralen Faser a_{\max} definiert. Es gilt:

$$M_b = F \cdot l \quad [\text{Nm}] \quad (7)$$

und

$$W = \frac{I}{a_{\max}} \quad [\text{m}^3]. \quad (8)$$

Werden beide Zusammenhänge in eine Gleichung (6) gesetzt, ergibt sich eine Formel zur Berechnung der maximalen Biegespannung σ_b in den Randfasern des Balkens:

$$\sigma_b = \frac{F \cdot l \cdot a_{\max}}{I} \quad [\text{N/m}^2]. \quad (9)$$

Aufgrund des runden Querschnitts der Achse kann der Abstand zwischen Randfaser und neutralen Faser a_{\max} durch den Radius r ersetzt werden. Das Flächenträgheitsmoment eines Balkens mit rundem Querschnitt beträgt:

$$I = \frac{\pi}{4} \cdot r^4 \quad [\text{m}^4]. \quad (10)$$

Zusammengefasst ergibt sich die Formel zur Berechnung der maximalen Biegespannung σ_b eines runden, einseitig eingespannten Balkens im elastischen Bereich:

$$\sigma_b = \frac{F \cdot l \cdot r}{\frac{\pi}{4} \cdot r^4} = \frac{F \cdot l}{\frac{\pi}{4} \cdot r^3} \quad [\text{N/m}^2]. \quad (11)$$

Aus der DMS-Technik ist außerdem die Formel zur Berechnung der mechanischen Spannung σ_b aus dem Produkt des Elastizitätsmoduls E und der maximalen Dehnung des Dehnungsmessstreifens ϵ_{\max} bekannt. Diese kann mit der vorherigen Formel (11) gleichgesetzt werden.

$$\sigma_b = \frac{F \cdot l}{\frac{\pi}{4} \cdot r^3} = E \cdot \epsilon_{\max} \quad [\text{N/m}^2] \quad (12)$$

Aus diesem Zusammenhang kann der minimale Radius r der Messachse berechnet werden:

$$r = \sqrt[3]{\frac{F \cdot l}{\frac{\pi}{4} \cdot E \cdot \epsilon_{\max}}} \quad [\text{m}]. \quad (13)$$

Allerdings gibt diese Formel nur den Radius r der Messachse an, unter dem der Dehnmessstreifen bei einem Biegemoment M_b maximale Dehnung erfährt. Eine plastische Verformung der Messachse unter dieser Krafteinwirkung wird nicht ausgeschlossen. Da eine dauerhafte Verformung die Messwerte verfälschen und andere Windenteile beschädigen könnte, wird im folgenden jener Radius r der Messachse berechnet, unter dem keine plastische Verformung des Bauteils auftritt.

Die Grenzspannung eines Materials, ab welcher sich eine elastische Verformung zu einer plastischen Verformung entwickelt, wird in der Werkstofftechnik als Streckgrenze R_e bezeichnet. Bis zum Erreichen dieser Spannung bleibt die Verformung des Materials infolge einer Krafteinwirkung weitestgehend reversibel. Im Falle der Messachse kehrt das gebogene Material nach Entlastung wieder in seine ursprüngliche Form zurück. Doch gerade bei Materialien ohne ausgeprägtem Fließverhalten, beispielsweise Aluminium oder hochfestem Stahl, kann oft keine klare Streckgrenze ermittelt werden. Stattdessen wird die 0,2-Dehngrenze oder auch Ersatzstreckgrenze $R_{p0,2}$ verwendet, welche eindeutig aus dem Spannungs-Dehnungs-Diagramm ermittelt werden kann. Sie gibt die notwendige mechanische Spannung an, bei der eine plastische Dehnung des Materials um genau 0,2% zu beobachten ist.⁷

Die Dehngrenze $R_{p0,2}$ kann ebenfalls mit der Formel (11) gleichgesetzt werden. Dadurch ergibt sich der minimale Radius r' der Messachse wie folgt:

$$r' = \sqrt[3]{\frac{F \cdot l}{\frac{\pi}{4} \cdot R_{p0,2}}} \quad [\text{m}]. \quad (14)$$

⁷ <https://www.maschinenbau-wissen.de/skript3/werkstofftechnik/metall/22-streckgrenze> [zuletzt überprüft am 22.11.2024]

3 Planung der Winde

3.1 Anforderungen an die Winde

Für die Konzeption der Winde war es zunächst erforderlich, wichtige Kenngrößen und Anforderungen zu ermitteln. Aus Gesprächen mit dem Außenbetreuer der Besonderen Lernleistung, Benedikt Justen, sowie den Erfahrungswerten des Projektmitarbeiters und Drachenbauers Klemens Brumann gingen erste approximierte Werte hervor. So benötigt die Winde eine maximale Einzugsgeschwindigkeit von 15 km/h bis 20 km/h, um den Drachen auch unter ungünstigen Windverhältnissen zuverlässig starten zu können. Es ist mit einer maximalen Zugkraft des Drachens von 2000 N zu rechnen. Des Weiteren soll eine Zugkraftmessung der Schnur integriert werden, um sicherzustellen, dass nicht zu viel Kraft auf das System wirkt, was zu Schäden an der Winde oder im schlimmsten Fall zu einem Reißen der Schnur führen könnte.

In der weiteren Planung musste auch die Schnurlänge zur Dimensionierung der Trommel berücksichtigt werden. Da die Winde als erster Prototyp des „Kite-Projekts“ dient, wurde eine 5000 Meter lange Dyneema-Drachenschnur verwendet. Im Verlauf des Projektes soll diese durch eine längere Schnur ersetzt werden. Als generelle Anforderungen an das Projekt sollen die Gesamtkosten möglichst niedrig gehalten werden.

3.2 Grundlegendes Konzept

Der folgende Abschnitt entstand aus gemeinsamen Überlegungen mit Thomas Hartmann, dem Koordinator und Organisator des Projektes „Höhenflugrekord“, welches darauf abzielt, den aktuellen Höhenrekord für Fesseldrachen von 9740 m zu brechen. Im Rahmen des Projekts sollen mehrere Drachen in einer sogenannten Drachenkette fliegen, um gemeinsam eine Flughöhe von 10.000 m erreichen.⁸

Ein großes Problem stellt die hohe und inkonstante Zugkraft dar, die selbst bei niedriger Windstärke vom Drachen ausgeht und ein direktes Aufwickeln der Dyneema-Schnur auf der Trommel verhindert. Andernfalls würde es zu einem Einschneiden der Schnur auf der Trommel kommen. Es wird ein zweiter Motor benötigt, der – vor dem Trommelmotor positioniert – leistungsfähig genug sein muss, den Drachen auch unter hohem Zug zuverlässig einzuziehen. Auf diese Weise ermöglicht er dem Trommelmotor das Auf- und Abwickeln der Drachenschnur mit einer geringeren und relativ konstanten Kraft (idealerweise 30 N bis 50 N).

Da ein „Durchrutschen“ der lediglich 2 mm dicken und extrem gleitfähigen Dyneema-Schnur, insbesondere bei hoher Zugkraft, ein weiteres Problem darstellt, muss diese am besagtem Motor eine große Auflagefläche erhalten, um ausreichend Haftreibung zu erzeugen. Thomas Hartmann verwies in diesem Kontext auf zwei drehbar gelagerte und durch den Motor angetriebene Rollen, zwischen denen die Schnur mehrere Windungen tätigt und so eine ausreichend große Auflagefläche generiert. Dieses in Abbildung **8: 2005/06 genutztes Windenteil des Drachenclub Flattermann** dargestellte Konzept wurde leicht verändert und stark verkleinert in die Planung der Winde einbezogen.

⁸<https://hoehenflugrekord.de/> [zuletzt überprüft am 12. 12. 2024]

Im Verlauf der Arbeit wird der eben erwähnte Motor als Zylindermotor bezeichnet.

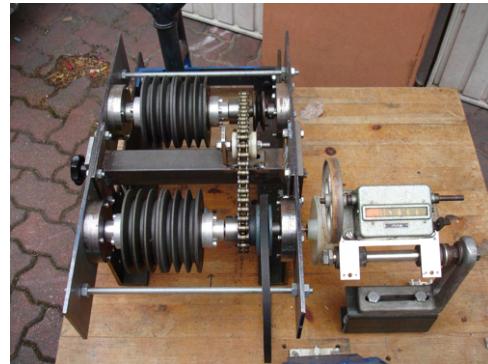


Abbildung 8: 2005/06 genutztes Windenteil des Drachenclub Flattermann

Damit die Schnur gleichmäßig auf der Trommel aufgewickelt werden kann, verläuft sie zuvor über eine Umlenkrolle, welche auf einem Schlitten gelagert ist. Dieser Schlitten befindet sich wiederum auf einer Linearführung und kann über einen Schrittmotor hin und her bewegt werden.

Die Zugkraftmessung erfolgt an einer fest verankerten Messachse im System. Dabei dient die Achse als Lagersitz einer Umlenkrolle, über welche die Dyneema-Schnur verläuft. Mit anwachsender Zugkraft tritt eine Belastung an der Messachse auf. Diese Belastung bewirkt eine Verformung der Dehnmessstreifen, welche über den Mikrocontroller des Systems ausgelesen werden kann. Nähere Informationen zu dieser Thematik sind in Abschnitt **2.2 DMS-Messtechnik** und **4 Entwicklung der Messachse** zu finden.

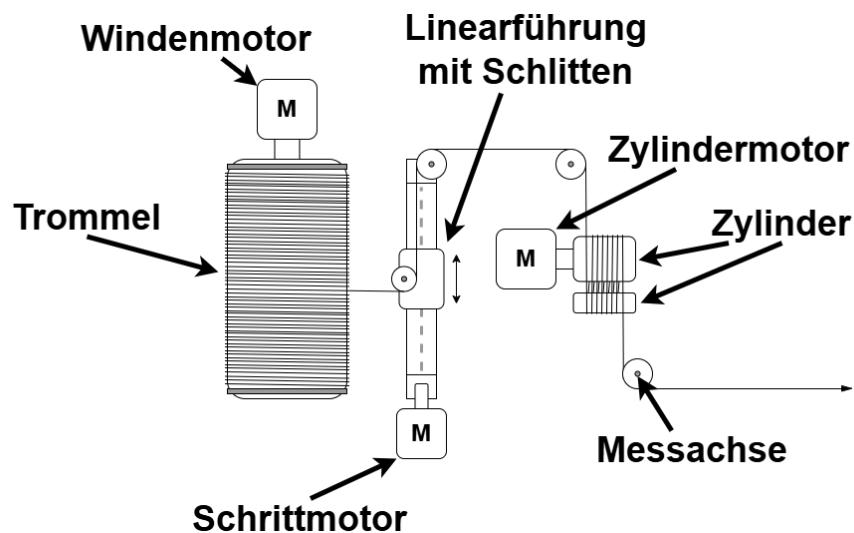


Abbildung 9: Konzept der Winde

In einer späteren Entwicklungsstufe (siehe Abschnitt **5.1 Aufbau der Steuerung**) wurde das Konzept der Winde um einen Dreharm ergänzt, über welchen die Geschwindigkeit des Windenmotors geregelt werden kann.

3.3 Entwicklung der finalen Windenversion

Für die weitere Entwicklung der Winde aus diesem grundlegenden Konzept erfolgte zunächst ein Besuch im Leibniz-Institut für Troposphärenforschung (TROPOS). Dieser diente dazu die verschiedenen, für die Verwendung mit Fesselballons entwickelten Windenausführungen zu besichtigen. Da sich die Winden trotz ihrer unterschiedlichen Anwendungsbereiche in wesentlichen Punkten ähnelten, konnten während des Besuchs verschiedene Konzepte nachvollzogen und später in die Entwicklung der eigenen Winde eingebracht werden. Die Abbildung **10: Winde des Leibniz-Instituts für Troposphärenforschung (TROPOS)** zeigt exemplarisch eine der vom Institut in Forschungsexpeditionen verwendeten Winden.

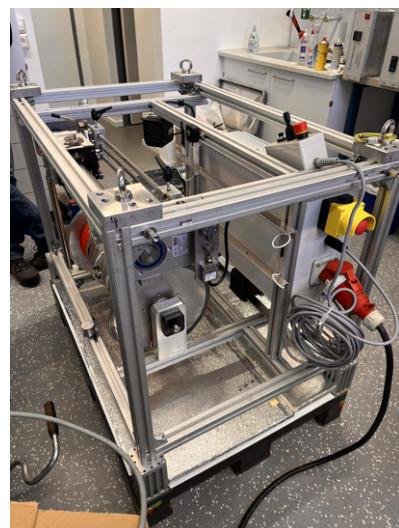


Abbildung 10: Winde des Leibniz-Instituts für Troposphärenforschung (TROPOS)

Auf diese Weise konnte beispielsweise das Konzept eines Seilfensters in die Entwicklung der eigenen Winde übernommen werden. Ein Seilfenster verhindert das Reiben der Schnur in jeglicher Richtung, unabhängig von der Stellung des Drachen. Links in der Abbildung **11: Designs der Seilfenster** ist das von TROPOS entwickelte und rechts das in der eigenen Winde verwendete Design dargestellt.



Abbildung 11: Designs der Seilfenster

Da die Winde bereits bei der Expedition „Solar Eclipse 2024“ in Texas (USA) zum Einsatz kommen sollte und ihre weitere Entwicklung sowie Fertigung ein hohes technisches Verständnis erforderten, übernahm Außenbetreuer Benedikt Justen die Gestaltung und den Bau der Winde, basierend auf den beschriebenen Grundüberlegungen. Nach Absprache ergänzte er den Dreharm zur Regelung der Geschwindigkeit des Trommelmotors (siehe Abschnitt **5.1 Aufbau der Steuerung**).

4 Entwicklung der Messachse

Wie in Abschnitt **3.2 Grundlegendes Konzept** erwähnt, sah die Windensteuerung eine periodische Messung der Zugkraft der Drachenschnur über eine Messachse mittels Dehnungsmessstreifen vor. Ziel war es, diese Zugkraft dauerhaft zu überwachen, um eine Überschreitung zu verhindern und somit ein Reißen der Schnur auszuschließen. Die Achse wurde im Rahmen der Besonderen Lernleistung speziell für diese Anwendung entwickelt, angefertigt und getestet. Darüber hinaus wurden die Auswertung der DMS am Mikrocontroller sowie die Integration der Messachse in die Windensteuerung konzipiert und vorbereitet. Obwohl die Achse aufgrund fehlender technischer Funktionalität letztlich nicht in die finale Windensteuerung eingeliefert werden konnte, wird der Prozess ihrer Planung und Fertigung in diesem Kapitel dokumentiert.

4.1 Planung der Messachse

Die Messachse wurde als einfacher DMS-Kraftaufnehmer konzipiert. Diese Art der Kraftaufnehmer finden große Verwendung in der Sensorik und Prüftechnik. Außerdem kommen DMS-Kraftaufnehmer als sogenannte Wägezellen in nahezu allen modernen Digitalwaagen vor. Um die Konstruktion der Messachse möglichst simpel zu halten, wurde sie als Biegebalken-Kraftaufnehmer angenähert.

Im nächsten Schritt erfolgte die Auslegung der Achse. Der Radius musste dabei so gewählt werden, dass unter einer maximalen Krafteinwirkung keine dauerhafte Verformung auftritt, die Biegespannung also dauerhaft unter der 0,2-Dehngrenze (siehe Abschnitt **2.3 Formeln zur Auslegung der Messachse**) liegt. Gleichzeitig wurde eine hohe elastische Verformung der Messachse gefordert, um ein Funktionieren der Dehnungsmessstreifen zu gewährleisten. Die maximale Dehnung der verwendeten DMS beträgt $\epsilon_{\max} = 0,02$. Die Maximalkraft der Schnur wurde aus dem Abschnitt **3.1 Anforderungen an die Winde** übernommen und beträgt $F_{\max} = 2000 \text{ N}$. Der Abstand l zwischen Festlager und Punkt der Krafteinwirkung wurde auf $0,05 \text{ m}$ festgelegt. Das Elastizitätsmodul für Edelstahl der Legierung 1.4305 beträgt $E = 200 \cdot 10^9 \text{ N/m}^2$ ⁹. Unter Verwendung dieser Werte wurde der Achsenradius r unter zwei verschiedenen Annahmen berechnet:

$$r = \sqrt[3]{\frac{2000 \text{ N} \cdot 0,05 \text{ m}}{\frac{\pi}{4} \cdot 200 \cdot 10^9 \text{ N/m}^2 \cdot 0,005}}$$

$$r \approx 3,17 \text{ mm}$$

Berechnung 1 erfolgte nach Formel (13) und gibt den Radius $r \approx 3,17 \text{ mm}$ der Messachse an, unter dem die Dehnungsmessstreifen bei einer Krafteinwirkung von $F_{\max} = 2000 \text{ N}$ eine maximale Dehnung erfahren.

⁹https://ucpcdn.thyssenkrupp.com/_legacy/UCPthyssenkruppBAMXAustria/assets.files/download/1.4305_07.2017.pdf [zuletzt überprüft am 22.11.2024]

Da das Erreichen dieser maximalen Dehnung mit dem Risiko einer plastischen Verformung der DMS einhergeht, dient dieser Radius r lediglich als absolute Untergrenze und sollte nicht zur Dimensionierung der Messachse genutzt werden.

$$r' = \sqrt[3]{\frac{2000 \text{ N} \cdot 0,05 \text{ m}}{\frac{\pi}{4} \cdot 190 \cdot 10^6 \text{ N/m}^2}}$$

$$r' \approx 8,75 \text{ mm}$$

Berechnung 2 erfolgte nach Formel (14) und gibt den minimalen Radius $r' \approx 8,75 \text{ mm}$ der Messachse an, unter dem die 0,2-Dehnungsgrenze der Achse nicht überschritten wird. Diese wurde im Folgenden als $R_{p0,2} = 190 \cdot 10^6 \text{ N/m}^2$ angenommen und ergab sich aus der verwendeten Edelstahllegierung 1.4305¹⁰.

Aus beiden Berechnungen folgt, dass der Radius r mindestens 8,75 mm betragen muss, damit sich die Edelstahlmessachse unter einer Krafteinwirkung von $F_{\max} = 2000 \text{ N}$ nicht dauerhaft verformt. Daraufhin fand eine Auslegung der Messachse in der CAD-Software statt.

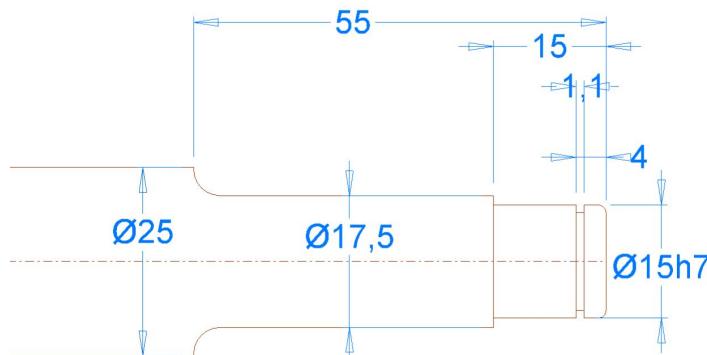


Abbildung 12: Konstruktionsskizze der Messachse

Die Messachse besteht aus einem Rundstab mit dem Durchmesser $\varnothing 25 \text{ mm}$. Im vorderen Bereich der Achse wurde der Durchmesser auf $\varnothing 17,5 \text{ mm}$ reduziert. Dieser Wert resultiert aus dem berechneten Radius $r' \approx 8,75 \text{ mm}$ und kennzeichnet den Abschnitt der Achse, an dem später die Dehnungsmessstreifen aufgeklebt werden. Am Ende der Messachse befindet sich eine $\varnothing 15h7$ Passung für das Kugellager der Umlenkrolle. Der Lagersitz wurde mit einer Nut versehen, in der später ein Sicherungsring für das Kugellager platziert wird. Die Achse wird an ihrem $\varnothing 25 \text{ mm}$ Durchmesser, direkt hinter der Verjüngung eingespannt. Auf diese Weise wird die Zugkraft der Drachenschur durch die Umlenkrolle auf die Messachse übertragen und von den Dehnungsmessstreifen erfasst.

¹⁰ https://ucpcdn.thyssenkrupp.com/_legacy/UCPthyssenkruppBAMXAustria/assets.files/download/1.4305_07.2017.pdf [zuletzt überprüft am 22.11.2024]

4.2 Fertigung der Messachse

Nach der Konstruktion in der CAD-Software wurde die Messachse aus einem Ø 25 mm Rundstab auf einer CNC-Drehmaschine gefertigt. Es wurde Edelstahl der Legierung 1.4305 verwendet. Da die genaue Einbauhöhe der Achse zu diesem Zeitpunkt noch nicht bekannt war, wurde der Rundstab während des gesamten Fertigungsprozesses möglichst lang belassen.

Zu Beginn war es nötig, die Kontur der Messachse über die werkstattorientierte Programmieroberfläche *ShopTurn* (*Siemens*) exakt nachzubilden. Auf Basis der eingegebenen Werkzeuge und Einsatzparameter wie Schnittgeschwindigkeit und Vorschub, konnte *ShopTurn* die Werkzeugbahnen selbstständig berechnen. Dabei wurde das Programm in einen Schrupp- und einen Schlichtvorgang unterteilt.

Im Schruppvorgang wird die äußere Kontur des Werkstücks ausgearbeitet, wobei gleichmäßig ein geringes Aufmaß auf dem Werkstück belassen wird. Dieses sogenannte Schlichtaufmaß wird final mit einem schärferen Werkzeug im Schlichtvorgang abgetragen. Dadurch wird eine gleichmäßigere und feinere Oberflächenstruktur geschaffen, welche im Klebebereich der Dehnungsmessstreifen sehr wichtig ist. Im Programm wurde ein Schlichtaufmaß von 0,2 mm gesetzt. In der folgenden Abbildung **13: Schritte während des Drehens der Messachse** wurde der Fertigungsprozess der Messachse Schritt für Schritt in der Programmieroberfläche *ShopTurn* dargestellt.



Abbildung 13: Schritte während des Drehens der Messachse

Daraufhin konnte die Messachse auf der CNC-Drehmaschine angefertigt werden.

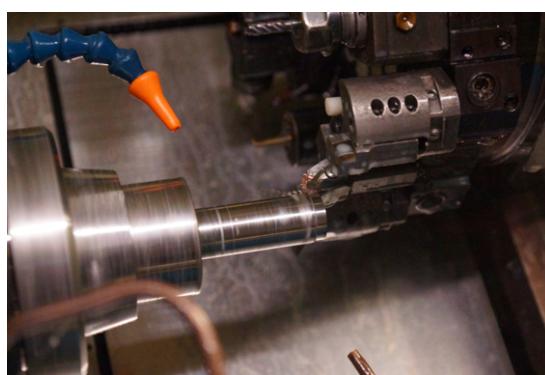


Abbildung 14: Drehen der Messachse auf der CNC-Drehmaschine

Damit die Folien-Dehnungsmessstreifen möglichst spannungsfrei und vollflächig auf der Messachse verklebt werden konnten, wurde ihre Mantelfläche auf zwei gegenüberliegenden Seiten gleichmäßig flach gefräst. Die Achse wurde dazu in einen Schraubstock gespannt und manuell auf einer Fräsmaschine bearbeitet. So konnten auf jeder Seite 0,3 mm gleichmäßig abgetragen werden. Die entstandenen Schlüsselflächen wurden anschließend mit feinem Sandpapier nachbearbeitet, um auch geringfügige Ungleichmäßigkeiten anzugleichen.



Abbildung 15: Fräsen der Schlüsselflächen für die Dehnungsmessstreifen

Im nächsten Schritt wurden die Dehnungsmessstreifen auf den vorgesehenen Klebeflächen der Messachse angebracht. Aufgrund der ausreichend großen Schlüsselflächen konnten insgesamt vier DMS jeweils paarweise verklebt werden. Dieser aufwendige und filigrane Prozess erforderte eine nahezu makellose Oberflächenstruktur des Untergrunds. Dieser musste daher im ersten Schritt des eigentlichen Klebeprozesses gereinigt und geschliffen werden. Daraufhin erfolgte eine Feinreinigung der Oberfläche mit dem Lösungsmittel Aceton, bis keinerlei Verunreinigungen mehr zu entdecken waren.



Abbildung 16: Messachse nach der Feinreinigung mit Aceton

Da die gereinigte Fläche unmittelbar nach diesem Arbeitsschritt wieder oxidierte und verschmutzte, musste der Klebeprozess schnellstmöglich durchgeführt werden. Um eine saubere Applikation zu gewährleisten, wurden mit einer Pipette mehrere kleine Tropfen Spezialkleber auf die Rückseite der Dehnungsmessstreifen aufgetragen. Nach korrekter Positionierung konnten diese unter einer Polyethylenfolie fest angedrückt werden, um die Bildung von Lufteinschlüssen zu vermeiden. Das vollständige Aushärten des Klebers nahm mehrere Tage in Anspruch. In dieser Zeit durfte die Messachse weder bewegt noch erschüttert werden. Sobald der Kleber ausgehärtet war, wurde unter die Anschlussdrähte jedes Dehnungsmessstreifens ein Kontaktplättchen platziert. Im letzten Schritt wurden die Anschlussdrähte der Dehnungsmessstreifen mit den vorgesehenen Anschlusskontakte des Lötplättchens verlötet. Abschließend wurde ein spezieller Schutzlack auf die Klebeflächen aufgetragen, um diese vor äußeren Einflüssen wie Staub, Öl oder Feuchtigkeit zu schützen.

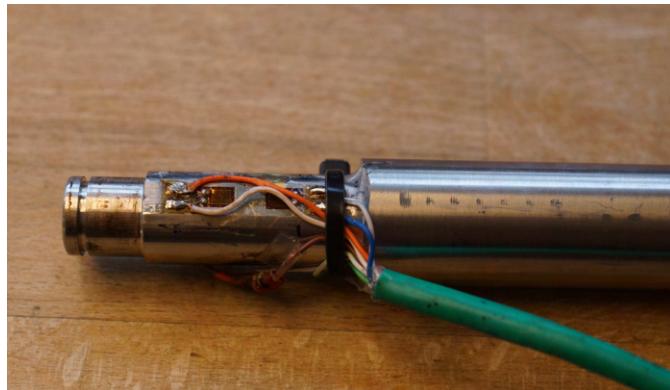


Abbildung 17: Messachse mit aufgeklebten DMS und verlöteten Anschlusskabeln

Für die Auswertung der relativen Widerstandsänderungen ΔR der Dehnungsmessstreifen am Mikrocontroller mussten diese in erster Linie in einer Wheatstone'schen Brückenschaltung verlötet werden. Aufgrund der 4 verwendeten DMS konnte eine Vollbrücke (siehe Abschnitt **2.2.2 Wheatstone'sche Brückenschaltung**) aufgebaut werden. Dabei wurden jeweils die DMS auf einer Seite angeordnet, die sich unter Last entgegengesetzt verformen. An der Messachse werden *DMS 1* und *DMS 3* bei einer Belastung gleichzeitig gestreckt. Daher werden sie im Schaubild der Wheatstone'schen Brücke auf entgegengesetzten Seiten positioniert. So bilden *DMS 1* und *DMS 2* sowie *DMS 3* und *DMS 4* jeweils einen Spannungsteiler der Brückenschaltung. Das entsprechende Schaubild ist in Abbildung **18: Messachse mit zugehöriger Brückenschaltung** dargestellt.

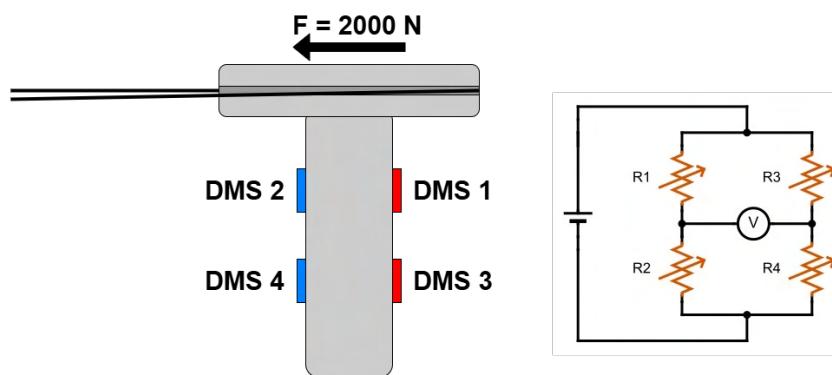


Abbildung 18: Messachse mit zugehöriger Brückenschaltung

Nach dem Lötvorgang wurde abschließend die Umlenkrolle auf die Passung der Messachse gesteckt und mit einem Sicherungsring fixiert. Es erfolgte eine Zugentlastung der Messkabel.

5 Entwicklung der Windensteuerung

5.1 Aufbau der Steuerung

Wie in Abschnitt **3.2 Grundlegendes Konzept** beschrieben, umfasst die Kernsteuerung der Winde folgende Komponenten:

- Den leistungsstarken **Zylindermotor**, der die Schnur unter einer bestimmten Geschwindigkeit einzieht oder ablässt.
- Den **Trommelmotor**, welcher die Schnur abhängig von der durch den Zylindermotor vorgegebenen Geschwindigkeit mit einer konstanten Kraft auf- beziehungsweise abwickelt.
- Den **Schrittmotor**, der den Schlitten auf der Linearführung bewegt und dadurch eine gleichmäßige Verteilung der Schnur auf der Trommel sicherstellt.

Dabei reagiert nur der Zylindermotor direkt auf die Befehle des Operators. Der Trommelmotor und der Schrittmotor passen ihre Bewegungen ausschließlich an Änderungen der Drehrichtung und -geschwindigkeit des Zylindermotors an und reagieren somit indirekt auf den Anwender.

Gesteuert wird dies durch einen beweglichen Dreharm mit gelagerter Umlenkrolle, über welche die Drachenschnur verläuft. Dieser ist wiederum mit einem Potentiometer verbunden, dessen Widerstandswert sich mit der Bewegung des Dreharms verändert. Der Widerstandswert wird daraufhin mit hoher Frequenz an einem Analogpin des Mikrocontrollers ausgelesen. Besitzt die Schnur eine zu geringe Spannung, wird der Dreharm durch eine Feder in Richtung der Nullposition gezogen. Dies tritt beispielsweise auf, wenn beim Aufwickeln mehr Schnur vom Zylindermotor bereitgestellt wird, als der Trommelmotor in gleicher Zeit auf die Trommel spult. Wickelt der Trommelmotor hingegen schneller auf, als der Zylindermotor die Schnur bereitstellen kann, erhöht sich die Spannung der Schnur und der Dreharm wird in Richtung der Extremposition gedrückt. Dieses Prinzip lässt sich auf den umgekehrten Fall – das Abwickeln – übertragen.

Die entstandene Regelsteuerung besitzt eine Sollposition, in der sich der Dreharm während des Betriebs mittig zwischen Null- und Extremposition befindet. So kann die Drehgeschwindigkeit des Trommelmotors über den relativen Abstand des Dreharms zu dieser angepasst werden. Die Drehgeschwindigkeit des Trommelmotors ist optimal geregelt, so lange sich der Dreharm in der Sollposition befindet. Dieses Prinzip wird schematisch in Abbildung **19: Konzept des Dreharms** dargestellt.

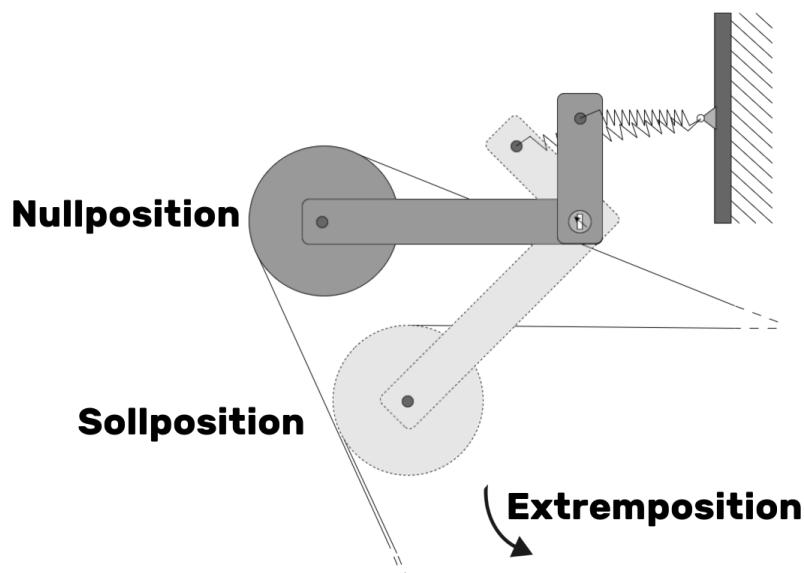


Abbildung 19: Konzept des Dreharms

Weiterhin reagiert der Schrittmotor in einer zweiten Regelstrecke auf die Geschwindigkeit der Trommel. Mit jeder vollständigen Umdrehung dieser, soll der Schlitten genau um eine Schnurstärke weiterwandern.

Das wird erreicht, indem zwei Magnete gegenüberliegend auf der Trommelwandung angebracht werden. Während des Windenbetriebs erfasst ein Hall-Sensor diese Magnete. Wie in Abbildung 20: **Hall-Sensor mit vorbeidrehenden Magneten (rot markiert)** angedeutet, sendet der Sensor ein Signal an den Mikrocontroller, sobald sich einer der Magnete in seinem Messbereich befindet.

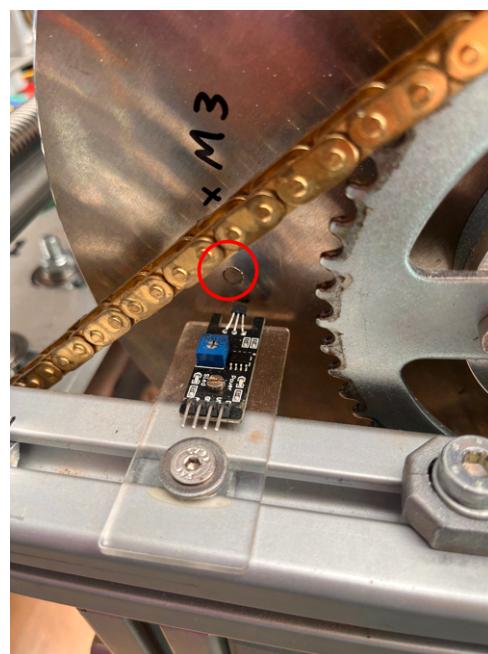


Abbildung 20: Hall-Sensor mit vorbeidrehenden Magneten (rot markiert)

Dadurch kann die Steuerung als Interrupt programmiert werden – ein Codeabschnitt – der unabhängig vom restlichen Programm ausgeführt wird und dieses unterbricht, sobald eine bestimmte Interruptbedingung erfüllt ist. Im Fall der Schrittmotorsteuerung ist dies das digitale Signal des Hall-Sensors. Für die Steuerung des Schrittmotors ist es entscheidend, die aktuelle Position des Schlittens auf der Linearführung zu kennen. Um diese zu ermitteln, wird ein Schalter am Anfang der Führung im Rahmen einer Referenzfahrt angefahren. Damit wird der Schrittzähler nach dem Systemstart auf null gesetzt, sodass die feste Anzahl an Schritten bis zur aktuell nächsten Trommelwandlung bekannt ist.

Unabhängig von der Motorsteuerung wird die Zugkraft des Drachens über die Spannung der Schnur gemessen. Wie bereits in Abschnitt **3.2 Grundlegendes Konzept** beschrieben, erfolgt diese Messung über Dehnmessstreifen an der Messachse. Sie ist gleichzeitig Lagersitz einer Umlenkrolle über welche die Schnur verläuft und so ausgerichtet, dass ihre Belastung in Messrichtung der DMS wirkt. Vergrößert sich nun beispielsweise die Zugkraft des Drachens in Folge einer Windböe, erhöht sich auch die Spannung der Drachenschnur. Die Messachse wird stärker belastet und der relative Widerstand der Wheatstone'schen Brücke steigt an. Über einen Messverstärker kann diese Widerstandsänderung am Mikrocontroller ausgelesen werden.

5.2 Auswahl der Komponenten

Der erste Schritt zur praktischen Umsetzung der Windensteuerung bestand in der Auswahl eines Motors, der die Anforderungen des Zylindermotors erfüllt. Wie in Abschnitt **3.1 Anforderungen an die Winde** beschrieben, muss dieser Motor eine Einzugsgeschwindigkeit von 15 km/h bis 20 km/h bei einer maximalen Zugkraft von 2000 N gewährleisten. Eine kurze Recherche ergab, dass diese Anforderungen aufgrund der Kosten und der Verfügbarkeit nur schwer zu erfüllen waren. Die geforderte Motorleistung in Verbindung mit der hohen Drehzahl ist zu groß.

Als Kompromiss zwischen Faktoren wie Leistung, Stromverbrauch, Preis und Bedienungsfreundlichkeit wurde für den Zylindermotor ein bürstenloser Gleichstrommotor (engl. *Brushless Direct Current*, kurz BLDC) ausgewählt. Der Motor erreicht eine hohe Leistung von 3000 W bei einer Nenndrehzahl von 4900 U/min und einer Maximaldrehzahl von 6600 U/min. Sein Nenndrehmoment beträgt 5,44 N m.

Bei einem Zylinderradius von annähernd 0,05 m und einer Übersetzung des Zylindermotors von 10,8 : 1 ergibt sich eine approximierte Maximalzugkraft von circa 1175 N. Da durch diese Übersetzung jedoch ein Kompromiss zwischen Drehmoment und Drehzahl getroffen wird, reduziert sich die Drehzahl des Zylinders von maximal 6600 U/min auf lediglich 611,11 U/min. Das entspricht einer maximalen Einzugsgeschwindigkeit von 11,52 km/h. Es ist wichtig zu beachten, dass diese Berechnungen auf den Idealwerten des Herstellers beruhen und lediglich als ungefähre Orientierung dienen. In der Praxis kann der Motor nicht dauerhaft auf maximaler Leistung betrieben werden, weshalb die zu erwartenden Werte für Drehzahl und Drehmoment unter den hier aufgeführten Idealwerten liegen dürften.

Wie der Name impliziert, verfügen BLDC-Motoren im Gegensatz zu bürstenbehafteten Gleichstrommotoren über keine Bürsten und keinen Kommutator. Dadurch wird der Verschleiß deutlich reduziert, weshalb diese Motoren annähernd wartungsfrei sind. Grundsätzlich arbeiten BLDC-Motoren sehr effizient, geräuscharm und zuverlässig, erfordern für den Betrieb jedoch einen elektronischen Drehzahlregler (engl. *Electronic Speed Control*, kurz ESC). Die Steuerung eines BLDC-Motors unterscheidet sich daher erheblich von der eines konventionellen bürstenbehafteten Motors. Ein weiterer Nachteil ist der höhere Preis bürstenloser Gleichstrommotoren. (Vgl. Ibrahim 2018, S. 22)



Abbildung 21: Verwendeter BLDC-Motor und Motorcontroller

Ursprünglich für den Betrieb von E-Bikes und E-Scootern entwickelt, erfolgt die Einstellung der Drehgeschwindigkeit über einen im Motorcontroller integrierten Drehzahlregler (rechts im Bild dargestellt). Dieser empfängt eine Analogspannung im Bereich 0,8 V bis 4,2 V, die über einen Hallgeber im Drehgriff reguliert wird. Die Drehrichtung wird über einen Schalter am Griff eingestellt.

Als Trommelmotor wurde ebenfalls ein identischer BLDC-Motor verwendet, allerdings mit einer geringeren Übersetzung von 4,91 : 1. Beide Motoren können somit über baugleiche Motorcontroller angesteuert werden. Die Arbeit an der Winde erleichtert sich dadurch erheblich. Für die Kommunikation mit dem Motorcontroller muss zunächst das digitale Signal des Schalters zur Festlegung der Drehrichtung sowie das analoge Signal des Hallgebers für die Drehzahl durch den Mikrocontroller simuliert werden. Nach ersten Testversuchen zur Ausgabe von Analogsignalen mittels Pulsweitenmodulation wurde im Verlauf dieser Arbeit mit den von *Microchip Technology* entwickelten MCP4725 Digital-Analog-Wandlern (engl. *digital-to-analog converter*, kurz DAC) gearbeitet.

Bei dem MCP4725 handelt es sich um einen Digital-Analog-Wandler mit 12-Bit-Auflösung, der die digitalen Signale des Mikrocontrollers in analoge Signale umwandelt. Die speziell für dieses Modul entwickelte Softwarebibliothek ermöglicht es, sowohl eine konstante Analogspannung in 4096 Schritten (12 Bit) als auch andere Analogsignale, wie Sinus- oder Dreieckswellen, auszugeben. Das DAC-Modul wird über die *I₂C*-Schnittstelle mit dem Mikrocontroller verbunden und verfügt zusätzlich über einen EEPROM-Speicher, der Konfigurationen im Falle eines Spannungsverlusts speichert.¹¹

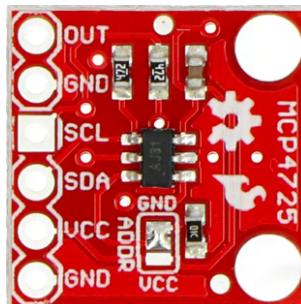


Abbildung 22: MCP4725 DAC-Modul

Für den Großteil der Testversuche, während der Entwicklung der Steuerung wurde ein Arduino UNO R3-Modell verwendet. In der finalen Windenversion kommt jedoch ein ESP32-S3-Mikrocontroller des Herstellers *Espressif Systems* zum Einsatz. Im Vergleich zum UNO R3 bietet dieser eine höhere Taktfrequenz in Verbindung mit einem größeren Arbeitsspeicher und verfügt über eine integrierte WLAN- und Bluetooth-Schnittstelle. Dies ermöglicht zukünftige Erweiterungen der Steuerung, wie die drahtlose Kommunikation mit der Winde beziehungsweise dem Messgerät am Drachen. Der *ESP32* kann ebenfalls über die Arduino-Entwicklungsramgebung programmiert werden, ist jedoch vergleichsweise weniger umfassend dokumentiert. (Vgl. Vendel 2024)

Für den Messverstärker der Wheatstone'schen Brückenschaltung wurde das von *Avia Semiconductor* entwickelte HX711-Modul ausgewählt. Dieses Modul wird häufig zur Auswertung von Wägezellen verwendet, um Gewicht oder Druck experimentell zu bestimmen, und ist daher besonders gut dokumentiert.

Der HX711 ist ein hochpräziser Analog-Digital-Wandler, der die Ausgangsspannung der Wheatstone'schen Brücke verstärken und anschließend in digitale Signale umwandeln kann. Dadurch lässt sich die Spannung mit einer Auflösung von 24 Bit präzise und verlässlich an einen Mikrocontroller übertragen. Das Modul wird mit einer Betriebsspannung von 2,6 V bis 5,5 V betrieben und ermöglicht eine Verstärkung der Messspannung mit den Faktoren 32, 64 und 128. Es verfügt über zwei separate Signaleingänge und kann die verstärkten Messwerte mit einer Frequenz von 10 Hz oder 80 Hz ausgeben. Die Schaltung des Moduls gewährleistet eine rauscharme und qualitativ hochwertige Signalverstärkung.¹²

¹¹ <https://www.microchip.com/downloads/en/devicedoc/22039d.pdf> [zuletzt überprüft am 16.07.2024]

¹² https://cdn.sparkfun.com/datasheets/Sensors/ForceFlex/hx711_english.pdf [zuletzt überprüft am 16.07.2024]

Die Auswahl der übrigen Hardware spielt für die Steuerung eine untergeordnete Rolle und wird daher nur der Vollständigkeit halber erwähnt. Aufgrund der geringen Leistungsanforderungen kam ein bereits vorhandener QMOT QSH6018-Schrittmotor zum Einsatz. Als Schrittmotortreiber diente das weit verbreitete Modell TB6600. In ersten Testversuchen wurden ebenfalls die Schrittmotorentreiber A4988 und DRV8825 genutzt. Ursprünglich aufgrund ihrer verlässlichen und präzisen Steuerung über den Mikrocontroller ausgewählt, mussten diese Modelle jedoch im Verlauf des Projekts aufgrund ihrer geringen Leistungsklasse verworfen werden. Für die Erfassung der Magneten in der Trommelwandung wurde ein KY-024-Hall-Sensor-Modul verwendet.

5.3 Installation der Komponenten

Da im Design Benedikt Justens (siehe Abschnitt **3.3 Entwicklung der finalen Windenversion**) alle wesentlichen Komponenten der Winde berücksichtigt wurden, traten bei deren Installation keine größeren Schwierigkeiten auf. Zylinder- und Trommelmotor konnten zusammen mit den beiden Motorcontrollern problemlos an den vorgesehenen Stellen montiert werden.

Für ihren Betrieb wurden die drei Phasen der Motoren über eine Kontaktschiene mit den entsprechenden Anschlusskabeln der Controller verbunden. Die Phasen erzeugen dabei ein Drehfeld und steuern die Motoren durch Variation von Frequenz und Stromfluss. Gleichzeitig wurden die Stromversorgungsanschlüsse der Motorcontroller an die Netzteile angeschlossen. Der lokale Wechselstrom wird zunächst im Netzteil in Gleichstrom umgewandelt. Anschließend transformiert der Drehzahlregler im Motorcontroller diesen Gleichstrom in dreiphasigen Wechselstrom, der die Motoren antreibt.

Die Verkabelung, in Abbildung **23: Verkabelung der BLDC-Motoren** dargestellt, wurde symmetrisch angeordnet, um das Risiko eines Kurzschlusses zu minimieren.

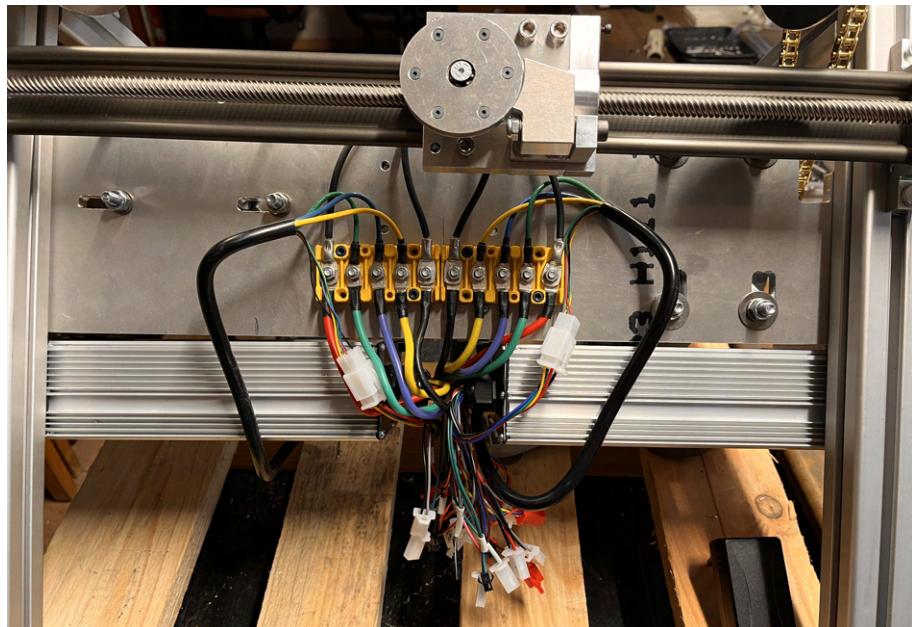


Abbildung 23: Verkabelung der BLDC-Motoren

Im nächsten Schritt wurden beide Motorcontroller mit dem Mikrocontroller verbunden. Wie bereits in Abschnitt **5.2 Auswahl der Komponenten** erwähnt, wurde der zunächst eingesetzte Arduino UNO R3 später durch den ESP32-S3 ersetzt. Im Zuge dieses Wechsels wurden die Steckverbindungen zu den Komponenten der Windensteuerung durch feste Lötverbindungen ersetzt.

Zur Regelung der Drehgeschwindigkeit mussten die entsprechenden Anschlüsse der Controller mit den Spannungsausgängen der DAC-Module verbunden werden. Zudem wurden die für die *I₂C*-Kommunikation erforderlichen *SDA*- und *SCL*-Anschlüsse mit den *I₂C*-fähigen Pins des Mikrocontrollers verbunden. Für die Ausgabe unterschiedlicher Spannungswerte an mehrere DAC-Module im selben *I₂C*-Bus benötigt jedes Modul eine eindeutige Adresse. Dafür wurde auf der Rückseite eines der MCP4725-Module an der entsprechenden Stelle ein kleiner Lötpunkt gesetzt. Dieser verändert die *I₂C*-Adresse des Moduls von standardmäßig 0x62 auf 0x63 und ermöglicht so die individuelle Ansteuerung jedes Moduls. Die folgende Abbildung **24: Schaltzeichnung zur Installation der BLDC-Motoren** zeigt die Verkabelung der BLDC-Motoren mit dem Mikrocontroller.

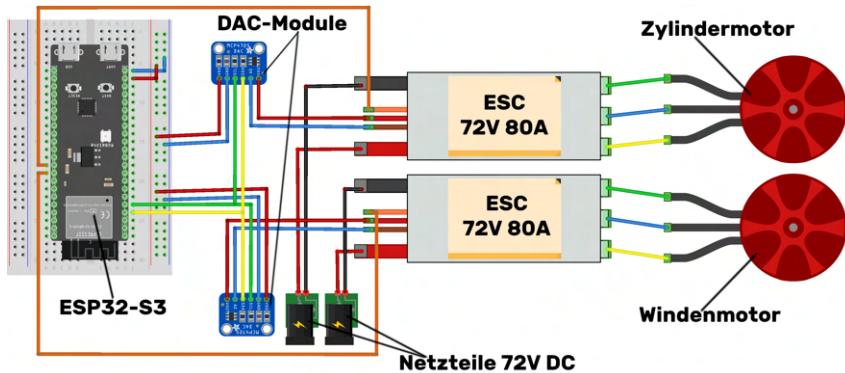


Abbildung 24: Schaltzeichnung zur Installation der BLDC-Motoren

Aufgrund der standardisierten NEMA-23-Normgröße konnte der Schrittmotor schnell und unkompliziert über Abstandshalter am Ende der Linearführung montiert werden. Als Verbindung zwischen der Welle des Motors und dem Steiggewinde der Linearführung kam eine Kupplung zum Einsatz, die einen möglichen Versatz und Unregelmäßigkeiten ausgleicht. Das sorgt für eine verbesserte Stabilität während des Betriebs.

Der TB6600-Treiber wurde seitlich am Aluminiumprofil in direkter Nähe zum Schrittmotor angebracht, um eine kompakte und effiziente Verkabelung zu gewährleisten. Die vier Phasen des Motors wurden mit den entsprechenden Anschlüssen des Treibers verbunden. Seine Steuerung erfolgt wiederum über die Signale *PULSE* und *DIRECTION*, die an die Digitalpins 13 und 14 des ESP32-S3 angeschlossen wurden.

In diesem Kontext wurde der Schalter zum Nullen des Systems am Anfang der Linearführung angebracht und mit dem Digitalpin 42 des Mikrocontrollers verbunden. Die Stromversorgung des Schrittmotors übernimmt ein 12 V-Netzteil, welches ausreichend Leistung für einen zuverlässigen Betrieb bietet. In der Abbildung **25: Schaltzeichnung zur Installation des Schrittmotors** ist anstelle des Schrittmotortreibers TB6600 der DRV8825 dargestellt (siehe Abschnitt **5.2 Auswahl der Komponenten**).

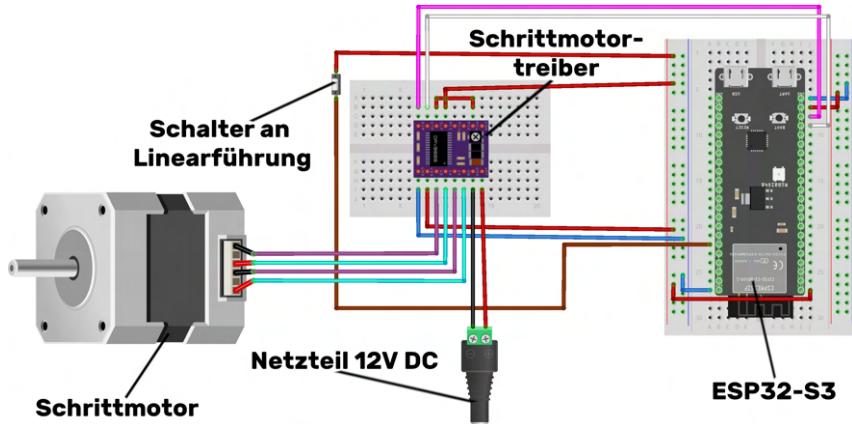


Abbildung 25: Schaltzeichnung zur Installation des Schrittmotors

Für die Installation des HX711-Messverstärkers wurden die Kontakte der Ausgangsspannung der Wheatstone'schen Brücke mit den Anschlüssen A+ und A- des Moduls verbunden. Gleichzeitig versorgen die Anschlüsse E+ und E- die Brücke mit der notwendigen Betriebsspannung, welche direkt vom Mikrocontroller bereitgestellt wurde. Da lediglich eine Brücke ausgelesen werden muss, blieben die Anschlüsse B+ und B- des Moduls ungenutzt.

Das Auslesen der Messwerte erfolgt über den DOUT-Pin des HX711-Moduls. Zusätzlich wird der SCK-Pin verwendet, um die Messwerte bitweise anzufordern. Am Mikrocontroller werden daher zwei Digitalpins für die Auswertung des Messverstärkers belegt. Die Abbildung 26: *Schaltzeichnung zur Installation des HX711-Messverstärkers* zeigt statt des ESP32-S3 einen Arduino der „Nano Family“. Das Prinzip zum Anschluss des Moduls bleibt jedoch nach wie vor unverändert.

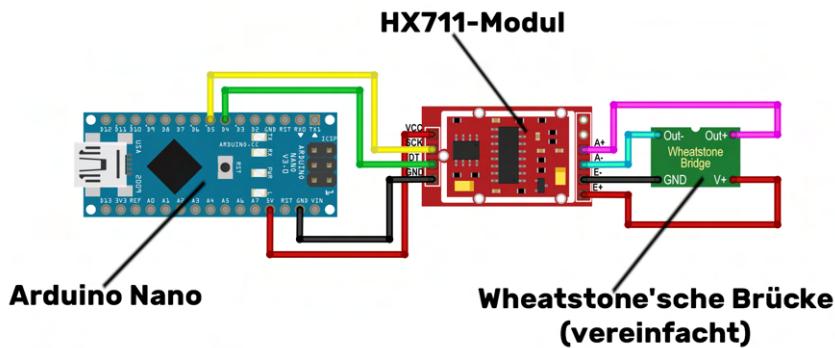


Abbildung 26: Schaltzeichnung zur Installation des HX711-Messverstärkers

Weitere Komponenten der Windensteuerung, insbesondere der Hall-Sensor und das Potentiometer des Dreharms, konnten direkt mit den Digitalpins 1 und 2 des ESP32-S3 verbunden werden. Es ergibt sich die umfassende Schaltzeichnung:

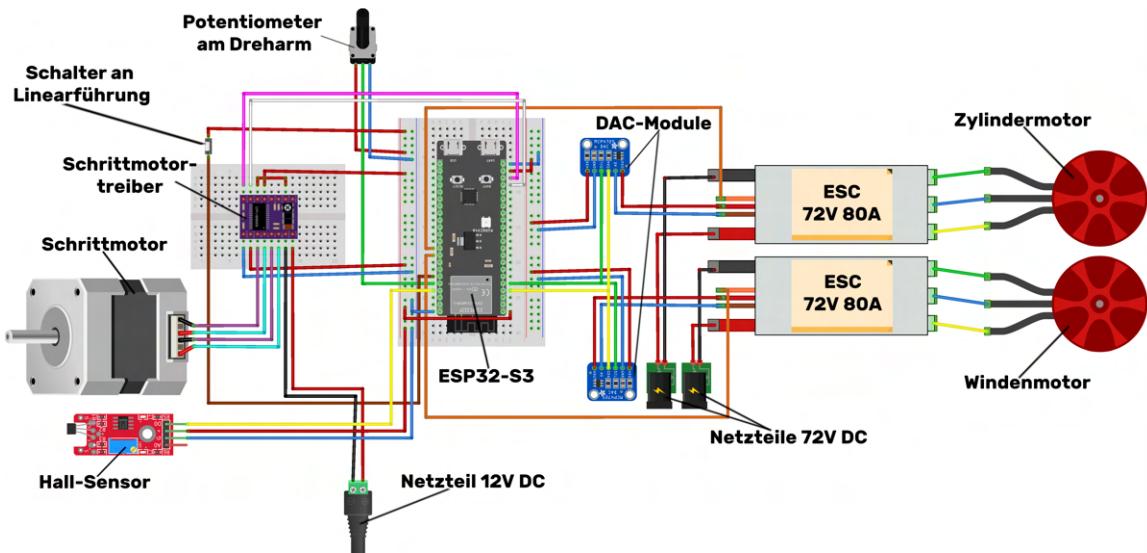


Abbildung 27: Schaltzeichnung zur Installation der Komponenten

5.4 Programmcode zur Steuerung

Als Grundpfeiler der Windensteuerung wurde die Funktion *Set_Speed()* erarbeitet, welche Drehgeschwindigkeit und -richtung der BLDC-Motoren (Zylindermotor und Windenmotor) anpasst.

Die Eingabe neuer Geschwindigkeitswerte, im Folgenden als *speed*-Werte bezeichnet, erfolgt im Wertebereich 0 bis +1000 beziehungsweise 0 bis -1000. Die Drehrichtung der Motoren wird über das jeweilige Vorzeichen angegeben. Der *speed*-Wert +1000 entspricht beispielsweise der maximalen Geschwindigkeit des BLDC-Motors in positiver Drehrichtung, der *speed*-Wert -500 hingegen der halben Geschwindigkeit in negativer Drehrichtung.

Wie in Abschnitt **5.2 Auswahl der Komponenten** erwähnt, wurde die Drehgeschwindigkeit der BLDC-Motoren ursprünglich über Analogsignale zweier Hallgeber geregelt. Darauf muss der Mikrocontroller die sich kontinuierlich verändernden Spannungen simulieren. Anstatt des „harten“ Setzens einer Geschwindigkeit bedarf es eines „weichen“ Übergangs zweier unterschiedlicher Drehgeschwindigkeiten. Es dürfen lediglich kleine Spannungsänderungen vorgenommen werden, damit der Motorcontroller des BLDC-Motors diese erfolgreich registriert. Im Gegensatz dazu kann die Drehrichtung ohne großen Aufwand über einen Digitalpin des Mikrocontrollers – als digitales Pendant eines physischen Schalters – umgeschaltet werden.

Frühe Testversuche zielten darauf ab, eine kontinuierliche Geschwindigkeitsänderung an den Motoren hervorzurufen. Das erste und einfachste Testprogramm bestand somit aus einer simplen *while*-Schleife, die den aktuellen *speed*-Wert (*current_speed*) in jedem Durchlauf um ein festgelegtes Intervall rampenartig erhöht, bis er dem vorgegebenen Wert (*new_speed*) entspricht. Die Präambel und andere wichtige Teile des Programms (beispielsweise *setup()*- und *loop()*-Funktion) wurden zur Vereinfachung ausgelassen. Der Codeabschnitt soll lediglich die gleichmäßige Veränderung des *speed*-Wertes verdeutlichen. Das Intervall der Rampe beträgt 1 *speed*-Wert. Außerdem wird eine bestimmte Zeit (*delay_time*) gewartet, um dem Motorcontroller zu ermöglichen, die Spannungsänderung wahrzunehmen.

```

1 int16_t current_speed = 0; // aktueller speed-Wert
2 int16_t new_speed = +500; // neuer speed-Wert
3 const int16_t delay_time = 10; // Wartezeit (10 ms)
4
5 // Schleife zur rampenartigen Erhöhung des current_speed:
6 while (new_speed > current_speed) {
7     current_speed++;
8
9     delay(delay_time);
10}
11
12 current_speed = new_speed;

```

Programmcode 2: Rampenartige Erhöhung des *speed*-Wertes

Da das DAC-Modul mit Werten in 12-bit-Auflösung arbeitet, mussten die *speed*-Werte (im Bereich 0 bis ± 1000) zuvor in *dac*-Werte des MCP4725 übertragen werden. Deren Wertebereich umfasst idealerweise 4096 Werte, von 0 bis 4095, die den Spannungswerten 0 bis 5 V des Mikrocontrollers zugeordnet sind. Daraus ergibt sich eine Genauigkeit von circa 1,2 mV pro Schritt. Jedoch wird nicht der gesamte Wertebereich des DAC-Moduls vom Drehzahlregler des Motorcontrollers genutzt.

Herstellerseitig ist zur Steuerung der BLDC-Motoren über die Drehzahlregler eine analoge Betriebsspannung von 0,8 bis 4,2 V angegeben. Nach durchgeföhrten Messungen schrumpft der Bereich, außerhalb welchem keine erkennbaren Änderungen in der Drehgeschwindigkeit des Motors stattfinden, auf 1,2 - 2,6 V. Dabei entsprechen die Spannungen 1,2 - 2,6 V, einem Bereich von minimal 982 und maximal 2129 *dac*-Werten des Moduls. Jedem *speed*-Wert (Wertebereich 0 bis 1000) wird also unabhängig seines Vorzeichens, ein passender *dac*-Wert von 982 bis 2129 zugeordnet.

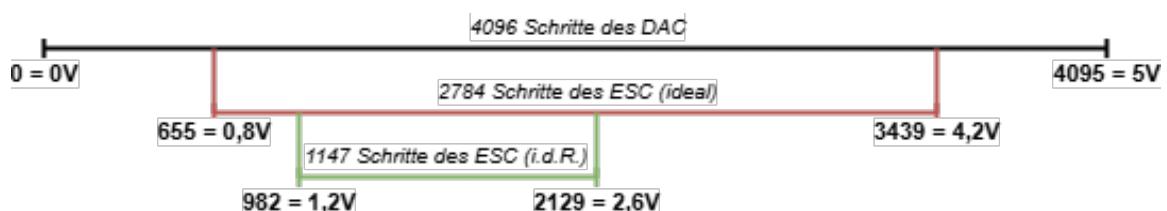


Abbildung 28: Wertebereich der nutzbaren *dac*-Werte

Im Programm wurde diese Zuordnung wie folgt umgesetzt:

```

1 // min. und max. Eingangsspannung der Drehzahlregler (in V):
2 const float min_voltage = 1.2;
3 const float max_voltage = 2.6;
4
5 // min. und max. Eingangsspannung der Drehzahlregler (dac-Wert):
6 const float dac_min_voltage = min_voltage / (5 / 4095);
7 const float dac_max_voltage = max_voltage / (5 / 4095);
8
9 int16_t current_speed = 0; // aktueller speed-Wert
10 int16_t new_speed = +500; // neuer speed-Wert
11 const int16_t delay_time = 10; // Wartezeit (10 ms)
12
13 // Schleife zur rampenartigen Erhöhung des current_speed (dac-Wert):
14 while (new_speed > current_speed) {
15     dac_value = map(abs(current_speed), 0, 1000, (int)dac_min_voltage, (int)dac_max_voltage);
16     DAC.setVoltage(dac_value, false);
17
18     current_speed++;
19
20     delay(delay_time);
21 }
22
23 DAC.setVoltage(dac_value, false);
24 current_speed = new_speed;

```

Programmcode 3: Rampenartige Erhöhung des *dac*-Wertes

Im ersten Teil des Codeabschnitts wird der eingeschränkte Wertebereich des DAC-Moduls festgelegt. Die Konstanten *min_voltage* und *max_voltage* stehen dabei für die minimale beziehungsweise maximale Eingangsspannung, die vom Drehzahlregler der Motorcontroller wahrgenommen werden können. Die Konstanten *dac_min_voltage* und *dac_max_voltage* entsprechen ihren zugeordneten *dac*-Werten. So kann dem aktuellen Geschwindigkeitswert (*current_speed*) in jedem Schleifendurchlauf ein *dac*-Wert zugeordnet werden, der zur Ausgabe der Analogspannung an den Drehzahlregler dient.

In der Erarbeitung der *Set_Speed()*-Funktion wurde eine Fallunterscheidung für die verschiedenenartigen Geschwindigkeitsänderungen hinzugefügt. Mittels *if*-Abfrage soll zunächst ermittelt werden, ob es sich um eine Zunahme oder Abnahme der Drehgeschwindigkeit handelt, bevor auf die entsprechende *while*-Schleife verwiesen wird. Zusätzlich wurde eine Variable für die Drehrichtung der Motoren eingeführt. Sie ist als boolescher Wert definiert und ändert sich genau dann, wenn der aktuelle *speed*-Wert 0 überschreitet. Dazu ist es notwendig, die *direction*-Variable in jedem einzelnen Schleifendurchgang abzufragen. Die Veränderungen werden in Programmcode **4: Vorläufige Set_Speed()-Funktion** verdeutlicht.

```

1 void Set_Win_Speed(int16_t new_speed) {
2
3     // Fallunterscheidung der Geschwindigkeitsänderung:
4     if (new_speed >= current_speed){
5
6         // rampenartigen Erhöhung des current_speed (dac-Wert):
7         while (new_speed > current_speed) {
8
9             // Abfrage ob Richtungsänderung vorgenommen werden muss:
10            if (current_speed >= 0) {
11                direction = 1;
12            }
13            else {
14                direction = 0;
15            }
16
17            dac_value = map(abs(current_speed), 0, 1000, (int)
18 dac_min_voltage, (int)dac_max_voltage);
19
20            digitalWrite(F_R_SWITCH, direction);
21            DAC.setVoltage(dac_value, false);
22
23            current_speed++;
24            delay(delay_time);
25        }
26    }
27
28    else{
29        while (new_speed < current_speed) {
30
31            if (current_speed >= 0) {
32                direction = 1;
33            }
34            else {
35                direction = 0;
36            }
37
38            dac_value = map(abs(current_speed), 0, 1000, (int)
39 dac_min_voltage, (int)dac_max_voltage);
40
41            digitalWrite(F_R_SWITCH, direction);
42            DAC.setVoltage(dac_value, false);
43
44            current_speed--;
45            delay(delay_time);
46        }
47    }
48
49    if (current_speed >= 0) {
50        direction = 1;
51    }
52    else {
53        direction = 0;
54    }

```

```

54     digitalWrite(F_R_SWITCH, direction);
55     DAC.setVoltage(dac_value, false);
56     current_speed = new_speed;
57 }

```

Programmcode 4: Vorläufige Set_Speed()-Funktion

Für die praktische Nutzung der Funktion ergibt sich das Problem, dass die *loop*-Funktion während dem Ausführen der *while*-Schleifen der *Set_Speed()*-Funktion verzögert wird. Da die *loop*-Funktion im normalen Betrieb des Mikrocontrollers mit hoher Frequenz ausgeführt wird und somit ebenfalls als Schleife fungiert, kann in der *Set_Speed()*-Funktion ausschließlich mit *if*-Abfragen gearbeitet werden. Anstatt einer *while*-Schleife, die den aktuellen *speed*-Wert solange ändert bis er dem Zielwert entspricht, wird dieser pro Durchlauf der *loop*-Funktion einmalig und geringfügig dem Zielwert angepasst. Dadurch kann im weiteren Verlauf der *loop*-Funktion die Geschwindigkeit des Windenmotors schneller und direkter auf jede der vielen geringen Geschwindigkeitsänderungen des Zylindermotors angepasst werden. Die Windensteuerung gewinnt dadurch an Stabilität und Reaktionsfähigkeit. Der finale Code zur *Set_Speed()*-Funktion ist in Programmcode **5: Finale Set_Speed()-Funktion** abgebildet.

```

1 void Set_Speed(int16_t new_speed) {
2
3     // Definition der relevanten Variablen für die Funktion:
4     static int16_t current_speed = 0;
5     bool direction = 1;
6     static uint16_t dac_value = 0;
7
8     // Fallunterscheidung nach Art der Geschwindigkeitsänderung:
9     if (new_speed > current_speed) {
10         current_speed++;
11     }
12     if (new_speed < current_speed) {
13         current_speed--;
14     }
15
16     // Abfrage ob Richtungsänderung vorgenommen werden muss:
17     if (current_speed >= 0) {
18         direction = 1;
19     }
20     else {
21         direction = 0;
22     }
23
24     dac_value = map(abs(current_speed), 0, 1000, (int)dac_min_voltage, (int)dac_max_voltage);
25
26     digitalWrite(F_R_SWITCH, direction);
27     DAC.setVoltage(dac_value, false);
28 }

```

Programmcode 5: Finale Set_Speed()-Funktion

Neben der *Set_Speed()*-Funktion bildet die *Adjust_Win_Speed()*-Funktion ein zentrales Element der Windensteuerung. Sie berechnet die neue Geschwindigkeit des Windenmotors *new_win_speed* und stellt somit die Eingabedaten für die *Set_Win_Speed()*-Funktion bereit.

Im folgenden Abschnitt wird die Herleitung einer Formel zur Berechnung von *new_win_speed* erläutert, die sowohl die Übersetzungsverhältnisse von Zylinder- und Windenmotors als auch die momentane Abweichung des Dreharms von der Sollposition berücksichtigt. Grundlegend gilt, dass die vom Zylindermotor transportierte Schnurlänge mit der des Windenmotors übereinstimmen muss, einen fehlerfreien Betrieb der Winde zu gewährleisten.

Dafür wurde zunächst das Verhältnis i der Motoren berechnet. Dieses hängt von den Drehzahlen des Zylinder- und Windenmotors sowie den Durchmessern für Trommel und Zylinder ab. Die Getriebe von Zylindermotor und Zylinder beziehungsweise Windenmotor und Trommel weisen dabei eine Übersetzung von jeweils 10,8 und 4,91 auf (siehe Abschnitt **5.1 Aufbau der Steuerung**). Daraus ergibt sich das Übersetzungsverhältnis i der Motoren von:

$$i = \frac{n_{\text{Zylindermotor}}}{n_{\text{Windenmotor}}} = \frac{10,8 \cdot \frac{1}{d_{\text{Zylinder}}}}{4,91 \cdot \frac{1}{d_{\text{Trommel}}}} \approx 2,2 \cdot \frac{d_{\text{Trommel}}}{d_{\text{Zylinder}}}.$$

Im Gegensatz zum konstanten Zylinderdurchmesser von 10 cm variiert der Trommeldurchmesser abhängig von der aufgewickelten Schnur zwischen 5 und 20 cm. Für eine 5000 m lange Schnur ergibt sich ein Wertebereich für i von 1,1 bis 4,4. Zur Vereinfachung wurde in der Windensteuerung mit einem festen Wert von $i = 2,2$ gearbeitet. Konkret bedeutet das für die Winde eine etwa 2,2-fach schnellere Bewegung des Zylindermotors pro Umdrehung des Windenmotors. Diese Vereinfachung wird in der finalen Formel (17) durch den Ausdruck d (siehe folgender Abschnitt) kompensiert.

Der Kehrwert dieser Übersetzung $\frac{1}{i} \approx 0,45$ multipliziert mit der neuen Zylindermotorgeschwindigkeit *new_zyl_speed* ergibt die neue Geschwindigkeit des Windenmotors *new_win_speed*.

Dieser Zusammenhang gilt allerdings nur unter der Annahme, dass Zylinder- und Trommeldurchmesser gleich groß sind und der Dreharm sich in Sollposition befindet. Da diese Bedingungen im Betrieb selten erfüllt sind, passt die Windensteuerung *new_win_speed* dynamisch an, um ein gleichmäßiges Aufwickeln der Schnur auf der Trommel zu gewährleisten.

Die Widerstandswerte des Dreharmpotentiometers (12-Bit-Auflösung) wurden am Mikrocontroller gemessen, wobei die Nullposition $min_poti = 1850$ und die Extremposition $max_poti = 350$ ermittelt wurden. Die Sollposition wurde mittig zwischen diesen Werten mit $opt_poti = 1100$ definiert. Auf Basis dieser Werte wurde ein mathematischer Ausdruck d entwickelt, um new_win_speed in Abhängigkeit von der momentanen Potentiometerauslenkung zu regeln:

$$d = \frac{((n - 1) + \frac{current_poti}{opt_poti})}{n}.$$

Die resultierende Formel zur Berechnung des new_win_speed lautet:

$$new_win_speed = \frac{1}{i} \cdot d \cdot new_zyl_speed \quad (15)$$

$$new_win_speed = \frac{1}{i} \cdot \frac{((n - 1) + \frac{current_poti}{opt_poti})}{n} \cdot new_zyl_speed \quad (16)$$

Die folgende Tabelle veranschaulicht den Ausdruck d mithilfe einer Fallunterscheidung:

Dreharm schwenkt in Richtung der Nullposition aus	Dreharm befindet sich in der Sollposition	Dreharm schwenkt in Richtung der Extremposition aus
$current_poti > opt_poti$	$current_poti = opt_poti$	$current_poti < opt_poti$
$\frac{((n - 1) + \frac{current_poti}{opt_poti})}{n} > 1$	$\frac{((n - 1) + \frac{current_poti}{opt_poti})}{n} = 1$	$\frac{((n - 1) + \frac{current_poti}{opt_poti})}{n} < 1$
Der Ausdruck d vergrößert new_win_speed .	Der Ausdruck d hat keine Auswirkungen auf new_win_speed .	Der Ausdruck d verkleinert new_win_speed .

Ist der Dreharm im Sollzustand, nimmt der Ausdruck $d = 1$ an und hat keinen Einfluss auf die Berechnung von new_win_speed . Wird der Dreharm während des Windenbetriebs allerdings in Richtung der Nullposition bewegt, nimmt der Ausdruck d einen Wert $d > 1$ an und wirkt verstärkend auf new_win_speed . Das ist beispielsweise der Fall, wenn new_zyl_speed beim Einholen der Schnur erhöht und daher in gleicher Zeit mehr Schnur in das Windsystem eingebracht wird. Steigt in diesem Beispiel der Zug durch den Drachen, dreht sich der Zylindermotor wieder langsamer, da er mehr Arbeit verrichten muss. Der Dreharm wird in Richtung der Extremposition ausgelenkt und der Ausdruck nimmt $d < 1$ an. Das bewirkt einen drosselnden Effekt auf new_win_speed .

Für die in Formel (16) eingesetzten Werte für n gilt: Je größer n , desto träger reagiert die Windensteuerung auf eine Abweichung des Dreharms aus der Sollposition. Gleichzeitig darf n nicht zu klein sein, da es zu einer Überregelung und damit zu einem Aufschwingen des Windensystems kommen könnte. In der folgenden Abbildung **29: Unterschiedliche Werte für n im Ausdruck d (Plot)** verdeutlichen verschiedene ganzzahlige Werte für n das jeweilige Verhalten der Windensteuerung.

Im Diagramm ist die Abhängigkeit des Verhältnisses von new_win_speed und new_zyl_speed ($\frac{new_win_speed}{new_zyl_speed} \rightarrow$ y-Achse) zur jeweiligen Stellung des Dreharms ($current_poti \rightarrow$ x-Achse) dargestellt.

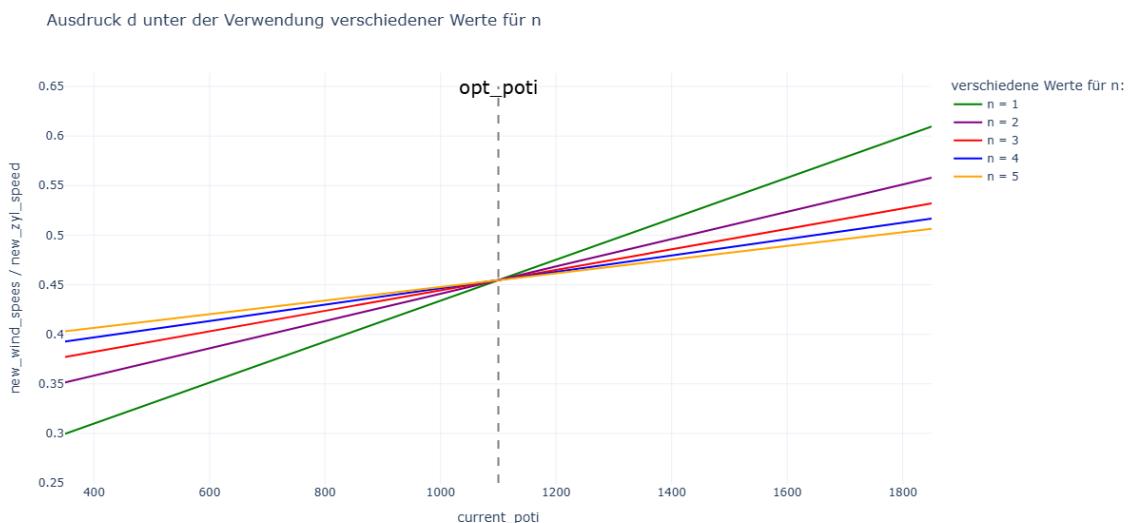


Abbildung 29: Unterschiedliche Werte für n im Ausdruck d (Plot)

Sichtbar ist, wie sich das Verhältnis zwischen new_win_speed und new_zyl_speed proportional zur Stellung des Dreharms $current_poti$ verändert. Dabei fällt der Anstieg mit wachsendem n ab. In der Sollposition des Dreharms $opt_poti = 1100$ nehmen alle Graphen den Kehrwert der Übersetzung zwischen den Motoren $\frac{1}{i}$ an.

In der Realität sind die Motoren in entgegengesetzter Drehrichtung montiert. Daher ergibt sich die endgültige Formel zur Berechnung der neuen Windenmotorgeschwindigkeit (new_win_speed):

$$new_win_speed = - \left(\frac{1}{i} \cdot \frac{((n-1) + \frac{current_poti}{opt_poti})}{n} \cdot new_zyl_speed \right) \quad (17)$$

Die Funktion *Adjust_Win_Speed()* ist entsprechend einfach gehalten und führt lediglich die Berechnung des *new_zyl_speed* aus:

```

1 const float i = 10.8 / 4.91; // Übersetzungsverhältnis Motoren
2 const float n = 2 // Wert zur Empfindlichkeit im Ausdruck d
3 const uint16_t min_poti = 1850; // Potistellung (ohne Schnurspannung)
4 const uint16_t max_poti = 350; // Potistellung (max. Schnurspannung)
5
6 // optimale Potistellung:
7 const uint16_t opt_poti = (min_poti + max_poti) / 2;
8
9 // Adjust_Win_Speed()-Funktion:
10 void Adjust_Win_Speed() {
11
12     // aktuelle Potistellung:
13     current_poti = analogRead(WIN_POT);
14
15     // Formel zur Berechnung von new_win_speed:
16     new_win_speed = -1 * (1/i * ((n-1) + (current_poti/opt_poti)) / n *
17                           new_zyl_speed);
18 }
```

Programmcode 6: Programmcode der *Adjust_Win_Speed()*-Funktion

Es verblieb, die vergleichsweise simple Steuerung des Schrittmotors umzusetzen. Der Code wurde dabei zunächst in einem eigenen Sketch programmiert und anschließend als Interrupt in das Hauptprogramm integriert.

Die Aufgabe der Steuerung ist, den Schlitten auf der Linearführung hin- und herzubewegen, sodass nach jeder vollständigen Umdrehung der Trommel genau eine Schnurstärke (2 mm) zurückgelegt wird. Das gewährleistet ein gleichmäßiges und störungsfreies Aufwickeln der Schnur. Werden zwei Magneten verwendet, die gegenüberliegend in der Trommelwandung positioniert sind, muss sich der Motor pro vorbeidrehendem Magneten um eine festgelegte Anzahl von Schritten ($part_revolution \hat{=} 1 \text{ mm}$) drehen. Die Konstante *steps_per_revolution* beschreibt die Anzahl an Schritten die notwendig ist um eine vollständige Umdrehung des Schrittmotors zu bewirken. Bei der verwendeten Ds12x25 Gewindestange ergibt sich: $steps_per_revolution \hat{=} 25 \text{ mm}$.

Für die Ansteuerung des Motors wurde die von *Arduino LLC* entwickelte *Stepper.h*-Bibliothek verwendet. Der folgende Programmausschnitt **7: Grundlegende Arbeitsweise der Schrittmotorsteuerung** zeigt die elementare Steuerung des Schrittmotors.

```

1 #include <Stepper.h>
2
3 #define HALL 1
4 #define PULSE 13
5 #define DIRECTION 14
6
7 // Schrittzahl für eine vollständige Umdrehung des Motors:
8 const uint16_t steps_per_revolution = 200 * 4;
9
10 // Schrittzahl pro vorbeidrehendem Magneten ( $\cong$  1 mm):
11 const uint16_t part_revolution = steps_per_revolution / 25;
12
13 // Drehgeschwindigkeit des Schrittmotors (in U/min):
14 const uint16_t stepper_speed = 100;
15
16 // Objektinitialisierung Schrittmotor:
17 Stepper linStepper(steps_per_revolution, PULSE, DIRECTION);
18
19 void setup() {
20
21   pinMode(HALL, INPUT);
22
23 }
24
25 void loop() {
26
27   if (digitalRead(HALL) == HIGH) {
28     linStepper.step(part_revolution);
29   }
30
31 }
```

Programmcode 7: Grundlegende Arbeitsweise der Schrittmotorsteuerung

Bereits vor Beginn des eigentlichen Programms müssen die benötigten Pins und Konstanten für den Betrieb des Schrittmotors definiert werden. Der Pin *HALL* ist dabei mit dem Hall-Sensor verbunden, während die Pins *PULSE* und *DIRECTION* zur Objektinitialisierung des Schrittmotors in Programmzeile 17 verwendet werden. Die Konstante *stepper_speed* legt die Geschwindigkeit des Motors in U/min fest.

Nachdem der *HALL*-Pin in der *setup()*-Funktion des Sketches als Eingang initialisiert wurde, sendet er ein digitales Signal an den Mikrocontroller, sobald sich einer der Magneten in seinen Messbereich bewegt. In diesem Fall nimmt der Befehl *digitalRead(HALL)* in Zeile 27 den Status *HIGH* an. Die Bedingung der *If*-Abfrage wird erfüllt und der Schrittmotor führt eine festgelegte Anzahl an Schritten (*part_revolution*) aus, die den Schlitten um exakt 1 mm auf der Linearführung bewegt.

Im nächsten Schritt wird die *stepper_count*-Variable eingeführt, um die Position des Schlittens auf der Linearführung zu bestimmen.

Dazu werden die getätigten Schritte des Motors zur *stepper_count*-Variable addiert. In Kombination mit der Konstante *max_steps*, die der Gesamtlänge der Führung (in Schritten) entspricht, lässt sich der jeweils nächste Umkehrpunkt bestimmen. Es gilt:

$$\text{stepper_count} >= \text{max_steps}.$$

Vorab muss das System jedoch an einer Begrenzung der Linearführung genutzt werden. In der *setup()*-Funktion dreht sich der Schrittmotor daher mit verminderter Geschwindigkeit in Richtung dieser Begrenzung, bis der Schlitten den am Führungsende positionierten Schalter (mit *STEPPER_ZERO* Pin verbunden) betätigt. Dies setzt den Befehl *digitalRead(STEPPER_ZERO)* auf *HIGH* und *stepper_count* auf null.

Der Schlitten pendelt im Windenbetrieb somit kontinuierlich auf der Linearführung hin und her und sorgt für ein gleichmäßiges Auf- beziehungsweise Abwickeln der Schnur auf der Trommel. Die Änderungen wurden im folgenden Programmcode **8: Vollständige Schrittmotorsteuerung** umgesetzt.

```

1 #include <Stepper.h>
2
3 #define HALL 1
4 #define PULSE 13
5 #define DIRECTION 14
6 #define STEPPER_ZERO 42
7
8 // Schrittzahl für eine vollständige Umdrehung des Motors:
9 const uint16_t steps_per_revolution = 200 * 4;
10
11 // Schrittzahl pro vorbeidrehendem Magneten ( $\cong 1$  mm):
12 const uint16_t part_revolution = steps_per_revolution / 25;
13
14 // Drehgeschwindigkeit des Schrittmotors (in U/min):
15 const uint16_t stepper_speed = 100;
16
17 volatile unsigned long stepper_count = 0; // Schrittzähler
18
19 // Länge der Linearführung in Schritten (l = 422 mm):
20 const float max_steps = (422 / 25) * steps_per_revolution;
21
22 // Objektinitialisierung Schrittmotor:
23 Stepper linStepper(steps_per_revolution, PULSE, DIRECTION);
24
25
26 void setup() {
27
28     pinMode(HALL, INPUT);
29     pinMode(STEPPER_ZERO, INPUT);
30
31     linStepper.setSpeed(stepper_speed/10);
32 }
```

```

33 // Referenzfahrt des Schrittmotors:
34 while (digitalRead(STEPPER_ZERO) == LOW) {
35   linStepper.step(-steps_per_revolution/100);
36   stepper_count += steps_per_revolution/100;
37 }
38
39 linStepper.setSpeed(stepper_speed);
40 linStepper.step(stepper_count);
41
42 }
43
44 void loop() {
45
46 if (digitalRead(HALL) == HIGH) {
47   // Richtungsänderung des Schrittmotors:
48   if (stepper_count >= max_steps) {
49     part_revolution = -part_revolution;
50     stepper_count = 0;
51   }
52   linStepper.step(part_revolution);
53   stepper_count += abs(part_revolution);
54 }
55
56 }

```

Programmcode 8: Vollständige Schrittmotorsteuerung

Wie in Programmausschnitt 7: **Grundlegende Arbeitsweise der Schrittmotorsteuerung** erfolgt erneut die Definition der benötigten Pins und Konstanten. Das Programm wird dabei um die Variable *stepper_count* und die Konstante *max_steps* erweitert. In der *setup()*-Funktion erfolgt zudem die Referenzfahrt des Systems. Zuerst wird die Geschwindigkeit des Motors in Zeile 31 deutlich reduziert, bevor der Schlitten in einer *while*-Schleife zur Begrenzung der Linearführung bewegt wird, bis er durch die Betätigung des Schalters die boolsche Variable *STEPPER_ZERO* auf *TRUE* setzt. Anschließend dreht sich der Schrittmotor, bis der Schlitten wieder seine ursprüngliche Position erreicht hat. Die Anfangsbedingungen, einschließlich *stepper_speed*, werden wiederhergestellt. Nur der *stepper_count* hat sich im Vergleich zum Zustand vor der Referenzfahrt entsprechend der Position des Schlittens angepasst. In jedem Durchlauf der *loop()*-Funktion wird nun geprüft, ob die Bedingung *stepper_count >= max_steps* erfüllt ist und eine Richtungsänderung vorgenommen werden muss (Zeile: 49). Andernfalls wird der Schlitten um die Schrittzahl *part_revolution* auf der Linearführung verschoben.

Die Schrittmotorsteuerung konnte, geringfügig angepasst, von einem eigenen Sketch in eine Funktion umgewandelt und so in die Windensteuerung eingegliedert werden.

Aufgrund der fehlenden Funktion der Messachse, wurde ihre Entwicklung zugunsten der anderen Steuerungsaspekte eingestellt. Der Code zur Auswertung der Dehnungsmessstreifen hätte mit der *HX711.h*-Bibliothek umgesetzt werden können, wobei eine Umrechnung der gemessenen Dehnung in eine Biegung der Messachse oder eine Zugkraft der Schnur erforderlich gewesen wäre. Folglich konnte der Programmcode zur Auswertung der Messachsen über den HX711-Messverstärker nicht in die Windensteuerung integriert werden.

Die zuvor beschriebene Entwicklung der *Set_Win_Speed()*-, *Adjust_Win_Speed()*- und *Stepper_Interrupt()*-Funktion bildete die Grundlage für das finale Programm zur Windensteuerung. Im Folgenden wird dieses übersichtlich erklärt und zusammengefasst.

In der Präambel des Programms werden dabei die benötigten Bibliotheken (explizit *Adafruit_MCP4725.h* und *Stepper.h*) geladen, Objekte und Pins initialisiert, sowie Konstanten und Variablen definiert. Zudem erfolgt eine Initialisierung des Schrittmotors und der im Programm benötigten Funktionen.

Nach der Präambel folgt die *setup()*-Funktion (siehe Abschnitt **2.1.2 Software**), welche die Startkonfiguration des Programms übernimmt.

```

1 void setup() {
2
3     // Initialisierung der benötigten Pins:
4     pinMode(ZYL_F_R_SWITCH, OUTPUT);
5     pinMode(WIN_F_R_SWITCH, OUTPUT);
6     pinMode(POTI, INPUT);
7     pinMode(HALL, INPUT);
8     pinMode(STEPPER_ZERO, INPUT);
9
10    // Initialisierung der seriellen und I2C-Kommunikation:
11    Serial.begin(115200);
12    Wire.begin(4, 5);
13    Wire.setClock(100000);
14    analogReadResolution(12);
15
16    // Initialisierung der MCP4725-Module:
17    if (ZYL_DAC.begin(zyl_I2C_address)) {
18        Serial.println("ZYL-DAC initialization successful!");
19    }
20    else {
21        Serial.println("ZYL-DAC initialization failed!");
22    }
23
24    if (WIN_DAC.begin(win_I2C_address)) {
25        Serial.println("WIN-DAC initialization successful!");
26    }
27    else {
28        Serial.println("WIN-DAC initialization failed!");
29    }
30
31    // Festlegen des Schrittmotor-Interrupts:
32    attachInterrupt(digitalPinToInterrupt(HALL), Stepper_Interrupt,
33                    RISING);
34 }
```

Programmcode 9: *Setup()*-Funktion der Windensteuerung

In der *setup()*-Funktion werden zunächst alle zuvor in der Präambel definierten Pins als Ausgang- beziehungsweise Eingangs-Pins des Mikrocontrollers initialisiert. Ausgenommen hiervon sind *PULSE* und *DIRECTION*, da diese bereits in der Objektinitialisierung des Schrittmotors außerhalb der *setup()*- und *loop()*-Funktion verwendet wurden.

Programmzeile 11 startet daraufhin die serielle Kommunikation mit einer Baudrate von 115.200 Bd. Die Zeilen 12 und 13 richten die *I2C*-Kommunikation ein: Die Signale *SDA* (*data*) und *SCL* (*clock*) werden auf die Pins 4 und 5 gelegt und die Taktrate wird auf 100.000 kHz gesetzt. In der nächsten Zeile erfolgt die Einstellung der Bitauflösung des *analogRead()*-Befehls. Die Angabe des Werts 12 legt fest, dass das Potentiometer des Dreharms an einem analogen Eingangspin des Mikrocontrollers mit einer Auflösung von 12 Bit (4096 Werte) gelesen wird.

Es folgt die Initialisierung beider MCP4725-Module. Je nach Ausgang dieser Initialisierung, wird die entsprechende Meldung im seriellen Monitor angezeigt. Abschließend wird der Interrupt des Schrittmotors konfiguriert. Als Interruptbedingung wird wie in Abschnitt **5.1 Aufbau der Steuerung** beschrieben, das Ausgangssignal HIGH des Hall-Sensors definiert. Gibt der Sensor durch einen vorbeidrehenden Magneten ein digitales Signal aus, wird die *Stepper_Interrupt*-Funktion aufgerufen.

```

1 void loop() {
2
3     int16_t temp_speed = 0; // temporärer Speicherplatz der Speedwerte
4
5     // Eingabe neuer Geschwindigkeitswerte über den seriellen Monitor:
6     if (Serial.available()) {
7         temp_speed = Serial.parseInt();
8
9         if (temp_speed <= 1000 && temp_speed >= -1000) {
10             new_zyl_speed = temp_speed;
11
12             Serial.print("speed = ");
13             Serial.println(new_zyl_speed);
14         }
15     else {
16         Serial.println("speed value is not within the defined range
17 (-1000 to 1000)");
18     }
19
20     // periodische Ausgabe der aktuellen Geschwindigkeit an die Motoren
21     if (millis() - speed_set_millis > speed_set_intervall) {
22         Set_Zyl_Speed(new_zyl_speed);
23         Adjust_Win_Speed();
24         Set_Win_Speed(new_win_speed);
25
26         speed_set_millis = millis();
27     }
28
29 }
```

Programmcode 10: *Loop()*-Funktion der Windensteuerung

In jedem Durchlauf *loop()*-Funktion wird zunächst eine Variable *temp_speed* definiert, welche als temporärer Speicherplatz der vom Anwender vorgegebenen *zyl_new_speed*-Werte gilt. Die Funktion prüft, ob ein neuer Wert für *zyl_new_speed* über den seriellen Monitor eingegangen ist. Ist dies der Fall, wird erneut geprüft, ob sich dieser Wert im Wertebereich 0 bis ± 1000 befindet. Erst danach wird der Wert von der Variable *temp_speed* übergeben und vom seriellen Monitor ausgegeben. Liegt der eingegebene Wert allerdings nicht im Wertebereich für *zyl_new_speed*, erfolgt eine Fehlermeldung über den seriellen Monitor.

Unabhängig davon werden die Funktionen *Set_Zyl_Speed()*, *Adjust_Win_Speed()* und *Set_Win_Speed()* aufgerufen, wenn ein bestimmtes Zeitintervall *speed_set_intervall* erfüllt ist. Die Funktion *millis()* zählt dafür die Zeit in Millisekunden, die seit dem Programmstart vergangen ist. Ist die Differenz aus *millis()* und *speed_set_millis*, einer Variablen für den Zeitpunkt des letzten *loop()*-Durchlaufs, größer als *speed_set_intervall*, werden die drei Funktionen zur Steuerung der Winde aufgerufen.

Im weiteren Verlauf der Entwicklung wurden erste Tests der Windensteuerung durchgeführt. Für den Wert $n = 2$ traten dabei vermehrt Ungleichmäßigkeiten in Betrieb und Regelung beider Motoren auf. Dies könnte zum einen an der fehlenden Kalibrierung des Systems, zum anderen an der Trägheit der Motorcontroller liegen. Aufgrund der Transportschäden (siehe Abschnitt **6 Zusammenfassung und Ausblick**) konnte die Windensteuerung nach der Expedition „Solar Eclipse 2024“ nur noch eingeschränkt getestet werden.

Der vollständige Programmcode für die Windensteuerung ist unter der Web-Adresse: <https://github.com/mxcano/Entwicklung-einer-Windensteuerung> veröffentlicht.

6 Zusammenfassung und Ausblick

Im Rahmen der Besonderen Lernleistung wurde eine Windensteuerung entwickelt und aufgebaut. Sie ermöglicht das gleichmäßige Auf- und Abwickeln der 5000 m langen Drachenschnur mit einer variabel einstellbaren Geschwindigkeit. Zusätzlich wird die Schnur gleichmäßig und platzsparend auf die Trommel gewickelt. Sie muss nicht, wie in der vorherigen „Solar Eclipse“-Expedition, aufwendig auf dem Boden ausgelegt und mit Körperkraft mehrerer Personen ausgegeben und eingezogen werden. Der Drachen kann kompakt und zuverlässig gestartet werden. Herausforderungen bestanden im Zusammenspiel der einzelnen Windenkomponenten und der Regelung der Windenmotorgeschwindigkeit.



Abbildung 30: Kastendrachen während der Expedition „Solar Eclipse 2024“

Für die fachpraktische Umsetzung erfolgte eine Einarbeitung in die Arduino-Entwicklungs-umgebung. Daneben wurde das Wissen in den Teilbereichen Mechatronik, DMS-Technik und Materialwissenschaften erweitert. Praktische Aufgaben waren beispielsweise das Verlöten, das Verkabeln oder das technische Zeichnen in der CAD-Umgebung. Eine Vielzahl von Versuchsaufbauten und Testprogrammen begleiteten die Entwicklung.

Aufgrund des Ziels, die Winde während der Expedition „Solar Eclipse 2024“ einzusetzen und dem daraus resultierenden Zeitdruck, kam es zu der pragmatischen Entscheidung, das System in einem ersten unfertigen Zustand zu verwenden. Das Forschungsteam transportierte die technische Ausrüstung, einschließlich der Winde, zum Einsatzort nach Eagle Pass (Texas, USA). Durch die amerikanische Administration wurde bis zum Tag der Sonnenfinsternis keine Flugelaubnis für den Drachen erteilt.

Parallel verfolgte ein Teil des Forschungsteams ein weiteres Projekt, in welchem zwei Imager unter Verwendung von Gimbal-Halterungen in die Tragflächen eines Flugzeugs eingebaut wurden. Während der Sonnenfinsternis flog dieses über den Wolken und folgte dabei dem Kernschatten. Dadurch wurde die Sicht nicht beeinträchtigt und die Beobachtungszeit der Finsternis verlängert. (Vgl. Coate 2024)

Diese Gründe führten zu einer niedrigeren Priorisierung des „Kite-Projekts“. Die Winde samt ihrer Steuerung kam während der Expedition „Solar Eclipse 2024“ nicht zum Einsatz und wurde auf dem Rücktransport nach Deutschland beschädigt. Eine Wiederherstellung ihrer Funktionalität konnte bis zum Zeitpunkt der Veröffentlichung dieser Arbeit nicht vollständig erreicht werden.

Unabhängig davon trägt die Entwicklung der Windensteuerung zu einem Erkenntnisfortschritt bei und bildet die Grundlage für zukünftige Entwicklungsstufen.

Erste Testversuche lieferten positive Ergebnisse beim Aufwickeln der Drachenschnur auf die Trommel. Die Ansteuerung des Schrittmotors sowie die Eingabe der Zylindermotorgeschwindigkeit über den seriellen Monitor funktionierten nahezu fehlerfrei. Nach Wiederherstellung des Systems durch den Austausch einzelner Komponenten muss die Windensteuerung noch präziser auf das System kalibriert und optimiert werden. Eine Möglichkeit zur Anpassung ist die Veränderung des Parameters n in der *Adjust_Win_Speed()*-Funktion.

Eine weitere Priorität stellt die Wiederinbetriebnahme der Messachse dar. Dazu gilt es, die Lötverbindungen der Wheatstone'schen Brücke zu überprüfen und die Funktion des HX711-Moduls an einer unabhängigen Anwendung (beispielsweise einer Wägezelle) zu gewährleisten.

In späteren Ausbaustufen kann eine Regelsteuerung zur Veränderung des Drachenanstellwinkels implementiert werden, die diesen in Abhängigkeit der von der Messachse erfassten Werte verändert. Nimmt die Zugkraft des Drachens bei starkem Wind zu, kann der Anstellwinkel verringert werden, um die Winde zu entlasten. Dies trägt zu einem stabileren Flug bei und erhöht die Sicherheit des Gesamtsystems. Umgekehrt bietet ein erhöhter Anstellwinkel dem Drachen auch bei geringem Wind eine vergrößerte Tragfläche, die dem Absinken des Drachens samt Last entgegenwirkt.

Auch in Zukunft hat die Beobachtung totaler Sonnenfinsternisse eine große wissenschaftliche Bedeutung. Die daraus gewonnenen Erkenntnisse sind unverzichtbar, um die Eigenschaften der Sonnenkorona verstehen zu können. Die Arbeit der „Solar Wind Sherpas“ wird fortgesetzt.



Abbildung 31: Kite-Team der „Solar Wind Sherpas“

Quellenverzeichnis

Literaturverzeichnis

Hoffmann, Karl: *Eine Einführung in die Technik des Messens mit Dehnungsmeßstreifen.* Darmstadt 1987.

Ibrahim, Dogan: *Motorsteuerung mit Arduino & Raspberry Pi.* Aachen 2018.

Justen, Benedikt et al.: *Kite-Borne Spectroscopic Observations of the Corona during the 2023 April 20 Total Solar Eclipse.* Leipzig 2025 (Vorabversion).

Schreiter, Danny: *Arduino-Kompendium. Elektronik, Programmierung und Projekte.* Lands-hut 2019.

Internetquellen mit Autor

Coate, Albany: *2024 Total Solar Eclipse: HSFL Support of Dr. Shadia Habbal Solar Corona Research*. In: <https://www.hsfl.hawaii.edu/2024/04/06/2024-total-solar-eclipse-hsfl-support-of-dr-shadia-habbal-solar-corona-research/> [zuletzt überprüft am 10. 12. 2024]

Ewald, Wolfgang: *Dehnungsmessstreifen*. In: <https://wolles-elektronikkiste.de/dehnungsmessstreifen#Wheatstone> [zuletzt überprüft am 22. 11. 2024]

Uppal, Anavi: *Solar Eclipse Experiment Will Fly a Kite to Avoid Cloudy Skies*. In: <https://www.scientificamerican.com/article/solar-eclipse-experiment-will-fly-a-kite-to-avoid-cloudy-skies/#:~:text=The%20scientists%20will%20launch%20a,if%20the%20skies%20are%20cloudy> [zuletzt überprüft am 10. 06. 2024]

Vendel Myles/Tan, Cherie: *Mighty Meets Mini. Esp32 vs Arduino: The Differences*. In: <https://all3dp.com/2/esp32-vs-arduino-differences/> [zuletzt überprüft am 11. 10. 2024]

Internetquellen ohne Autor

<https://www.mps.mpg.de/sonnenstuerme-sonnenaktivitaet-faq> [zuletzt überprüft am 16. 07. 2024]

<https://www.arduino.cc/en/Guide/Introduction> [zuletzt überprüft am 16. 07. 2024]

<https://www.programmingelectronics.com/what-is-a-breakout-board-for-arduino/> [zuletzt überprüft am 17. 10. 2024]

<https://docs.arduino.cc/software/ide-v1/tutorials/Environment/> [zuletzt überprüft am 16. 07. 2024]

<https://www.maschinenbau-wissen.de/skript3/mechanik/balken-biegung/209-biegung-berechnen> [zuletzt überprüft am 22. 11. 2024]

<https://www.maschinenbau-wissen.de/skript3/werkstofftechnik/metall/22-streckgrenze> [zuletzt überprüft am 22. 11. 2024]

<https://hoehenflugrekord.de/> [zuletzt überprüft am 12. 12. 2024]

https://ucpcdn.thyssenkrupp.com/_legacy/UCPthyssenkruppBAMXAustria/assets.files/download/1.4305_07.2017.pdf [zuletzt überprüft am 22. 11. 2024]

<https://ww1.microchip.com/downloads/en/devicedoc/22039d.pdf> [zuletzt überprüft am 16. 07. 2024]

https://cdn.sparkfun.com/datasheets/Sensors/ForceFlex/hx711_english.pdf [zuletzt überprüft am 16. 07. 2024]

Abbildungsverzeichnis

Abb. 1: Aufnahme einer totalen Sonnenfinsternis. In:

<https://www.scientificamerican.com/article/solar-eclipse-experiment-will-fly-a-kite-to-avoid-cloudy-skies/#:~:text=The%20scientists%20will%20launch%20a,if%20the%20skies%20are%20cloudy>

Abb. 2: Winde während der Expedition „Solar Eclipse 2024“. Eigene Abbildung.

Abb. 3: Gegenüberstellung bekannter Arduino-Modellreihen. In:

<https://www.arduino.cc/en/hardware>

Abb. 4: Aufbau eines Folien-DMS. In:

<https://www.dewetron.com/de/2022/04/dms/>

Abb. 5: Wheatstone'sche Brückenschaltung. In:

Hoffmann, Karl: Eine Einführung in die Technik des Messens mit Dehnungsmeßstreifen. Darmstadt 1987, S. 146

Abb. 6: Varianten der Wheatstone'schen Brückenschaltung. In:

<https://wolles-elektronikkiste.de/dehnungsmessstreifen#Wheatstone>

Abb. 7: Biegeverhalten eines punktuell belasteten, einseitig eingespannten Balkens. In:

<https://www.maschinenbau-wissen.de/skript3/mechanik/balken-biegung/209-biegungsberechnen>

Abb. 8: 2005/06 genutztes Windenteil des Drachenclub Flattermann. Abbildung Thomas Hartmanns.

Abb. 9: Konzept der Winde. Eigene Abbildung.

Abb. 10: Winde des Leibniz-Instituts für Troposphärenforschung (TROPOS). Eigene Abbildung.

Abb. 11: Designs der Seilfenster. Eigene Abbildung.

Abb. 12: Konstruktionsskizze der Messachse. Eigene Abbildung.

Abb. 13: Schritte während des Drehens der Messachse. Eigene Abbildung.

Abb. 14: Drehen der Messachse auf der CNC-Drehmaschine. Eigene Abbildung.

Abb. 15: Fräsen der Schlüsselflächen für die Dehnungsmessstreifen. Eigene Abbildung.

Abb. 16: Messachse nach der Feinreinigung mit Aceton. Eigene Abbildung.

Abb. 17: Messachse mit aufgeklebten DMS und verlötzten Anschlusskabeln. Eigene Abbildung.

Abb. 18: Messachse mit zugehöriger Brückenschaltung. Eigene Abbildung.

Abb. 19: Konzept des Dreharms. Eigene Abbildung.

Abb. 20: Hall-Sensor mit vorbeidrehenden Magneten (rot markiert). Eigene Abbildung.

Abb. 21: Verwendeter BLDC-Motor und Motorcontroller. In:

<https://www.amazon.de/B%C3%BCrstenloser-Gleichstrommotor-Elektro-Fahrrad-Elektroroller-Elektroautos/dp/B0928HMMPQ>

Abb. 22: MCP4725 DAC-Modul. In:

<https://botland.de/gpio-erweiterungen-fuer-raspberry-pi/2275-mcp4725-dac-i2c-konverter-sparkfun-bob-12918-modul-5903351249744.html>

Abb. 23: Verkabelung der BLDC-Motoren. Eigene Abbildung.

Abb. 24: Schaltzeichnung zur Installation der BLDC-Motoren. Eigene Abbildung.

Abb. 25: Schaltzeichnung zur Installation des Schrittmotors. Eigene Abbildung.

Abb. 26: Schaltzeichnung zur Installation des HX711-Messverstärkers In:

<https://wolles-elektronikkiste.de/dehnungsmessstreifen#Wheatstone>

Abb. 27: Schaltzeichnung zur Installation der Komponenten. Eigene Abbildung.

Abb. 28: Wertebereich der nutzbaren *dac*-Werte. Eigene Abbildung.

Abb. 29: Unterschiedliche Werte für *n* im Ausdruck *d* (Plot). Eigene Abbildung.

Abb. 30: Kastendrachen während der Expedition „Solar Eclipse 2024“. Abbildung Herbert Stehrs.

Abb. 31: Kite-Team der „Solar Wind Sherpas“. Abbildung Herbert Stehrs.

Selbstständigkeitserklärung

Hiermit erkläre ich, Max Rothe, dass ich die vorliegende Besondere Lernleistung selbstständig und ohne fremde Hilfe angefertigt und nur die im Literaturverzeichnis angeführten Quellen und Hilfsmittel benutzt habe. Insbesondere versichere ich, dass ich alle wörtlichen und sinngemäßen Übernahmen aus anderen Werken, inklusive Internetinhalten, als solche kenntlich gemacht habe.

Leipzig, den 20.12.2024

Max Rothe