

**HÖHERE TECHNISCHE BUNDESLEHRANSTALT
KLAGENFURT, MÖSSINGERSTRASSE**

**ABTEILUNG ELEKTRONIK UND TECHNISCHE
INFORMATIK**

DIPLOMARBEIT

CYCLO TEST-BENCH CTB

JAHRGANG 5AHEL & 5AHET

eingereicht von

Matteo Müller

Lucas Lenarcic

Tim Sperle

Projektbetreuer

Dipl.-Ing. Harald Grünanger

Dipl.-Ing. Harald Huber

Dr. Wolfgang Granig

Diese Diplomarbeit entspricht den Standards gemäß dem Leitfaden zur Umsetzung der Reife- und Diplomprüfung des BMBWF in der letztgültigen Fassung.

Klagenfurt, am 05.04.2024

EIDESSTATTLICHE ERKLÄRUNG

Ich erkläre an Eides statt, dass ich die vorliegende Diplomarbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche erkenntlich gemacht habe.

Verfasser / Verfasserin

Matteo Müller

Lucas Lenarcic

Tim Sperle

Klagenfurt, am 05.04.2024

Kurzbeschreibung

Das Projekt Cyclo Test-Bench CTB umfasst die Realisierung eines neuartigen Drohnenantriebs, der auf dem Voith-Schneider-Prinzip basiert. Die Diplomarbeit befasst sich dabei mit der Steuerung eines Prüfstandes zur Messung nicht vorhandener Vergleichsdaten im Bereich der Luftfahrt, wie z.B. der Schubkraftmessung.

Aufgabenstellung

Müller Matteo:

Programmierung der µC des Prüfstandes → Sicherheitsvorkehrungen beachten, Auslesen der Sensordaten, Programmierung der Website, Leiterplatten

Lenarcic Lucas:

Auswertung der Sensordaten, Webserver-, Datenbank-Design & Eintragung bzw. Verarbeitung, Programmierung der Website und Schnittstelle

Tim Sperle:

Design, Planung und Bau des Rotors, Prüfstandes und des Schutzkäfigs, Test der Hardwarekomponenten auf Stabilität

Realisierung

Drohnen gibt es mittlerweile in fast jeder Bauform und Baugröße. Dabei erfolgt der grundsätzliche Antrieb immer nach demselben Prinzip. Für eine geradlinige Vorwärtsbewegung müssen allerdings alle diese Drohnen eine Neigungsänderung aufweisen. Für entsprechend größere Drohnen, welche zum Transport von empfindlichen Gegenständen genutzt werden, könnte diese Eigenschaft zu einem Problem führen. Aus diesem Grund beschäftigen wir uns im Rahmen unserer Diplomarbeit mit der Entwicklung eines alternativen Antriebssystems. Wir entwickeln, programmieren und testen einen Cyclorotor, um die Grundlagen für die Konstruktion einer Drohne zu schaffen, die dazu in der Lage wäre, sich ohne Neigungsänderung vorwärtszubewegen.

Ergebnisse

Im Laufe der Bearbeitung der Diplomarbeit ist im Durchlauf von mehreren Prototypen der Soft- und Hardware ein fertiges Endprodukt entstanden, dass die vorgegebenen Anforderungen und Ziele sinngemäß erfüllt.

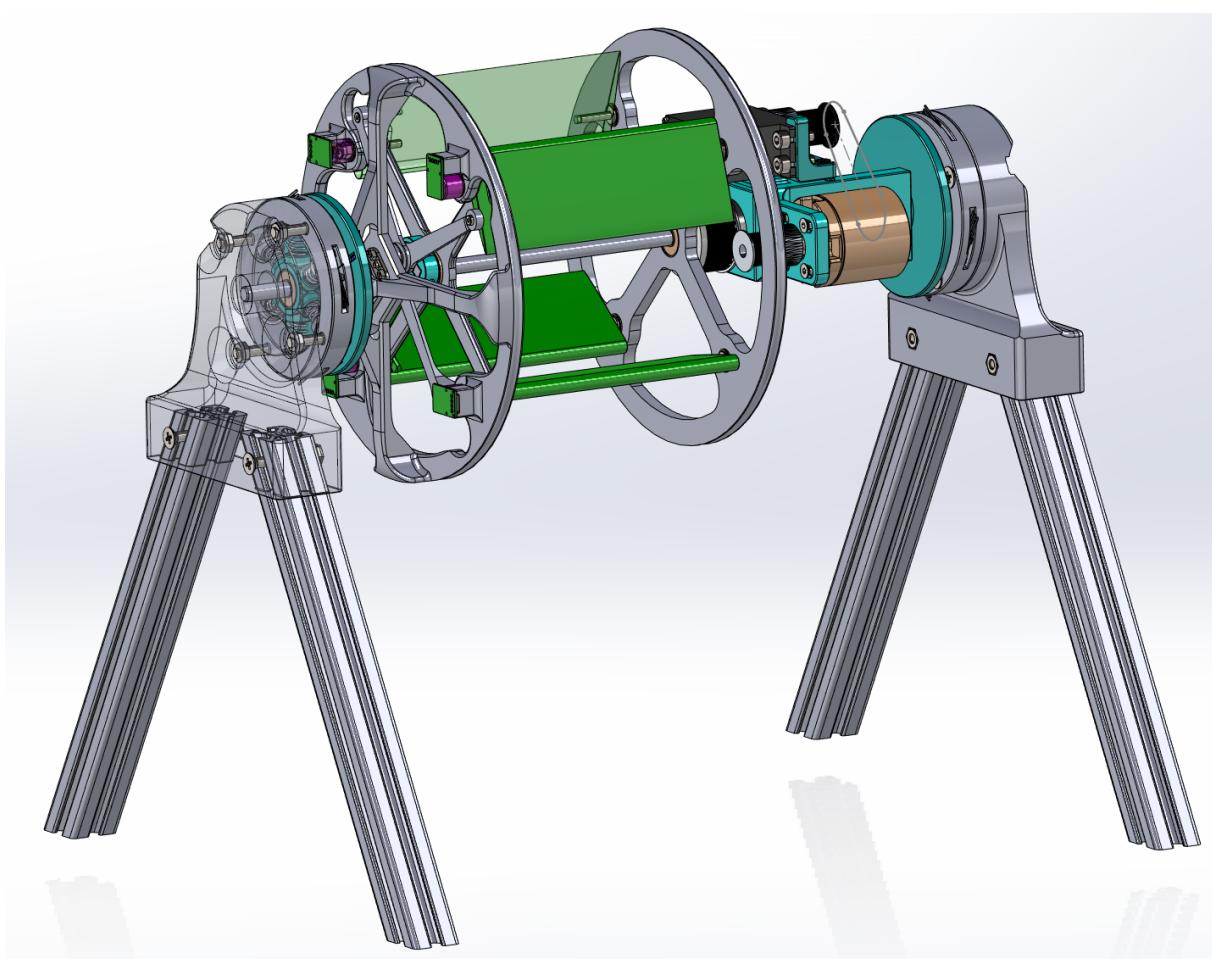


Abbildung 1: Rotorprinzip

Abstract

The Cyclo Test-Bench CTB project involves the realisation of a new type of drone propulsion system based on the Voith-Schneider principle. The thesis deals with the control of a test bench for measuring non-existent comparative data in the field of aviation, such as thrust measurement.

Assignment of Tasks

Müller Matteo:

Programming the µC of the test bench → Observing safety precautions, reading out sensor data, programming the website, circuit boards

Lenarcic Lucas:

Evaluation of the sensor data, web server, database design & entry or processing, programming of the website

Tim Sperle:

Design, planning and construction of the rotor, test stand and protective cage, testing the hardware components for stability

Implementation

Drones are now available in almost every shape and size. The basic drive is always based on the same principle. However, in order to move forwards in a straight line, all of these drones must have a change in inclination. For larger drones, which are used to transport sensitive objects, this characteristic could lead to a problem. For this reason, we are working on the development of an alternative drive system as part of our diploma thesis. We are developing, programming and testing a cyclorotor in order to lay the foundations for the construction of a drone that would be able to move forwards without changing its inclination.

Results

In the course of working on the diploma thesis, several prototypes of the software and hardware were run through to create a finished end product that meets the specified requirements and objectives.

Short title: CTB

Keywords: measurements, test bench, drone drive, frontend, hardware

Inhaltsverzeichnis

| | |
|--|----|
| 1. Einleitung..... | 11 |
| 2. Grundlagen und Methoden..... | 13 |
| 2.1. Ist-Situation / Stand der Technik | 13 |
| 2.2. Lösungsmethoden / Lösungsansatz..... | 14 |
| 2.3. Grundlagen für das Gesamtprojekt | 14 |
| 2.4. Schnittstellen zwischen den Modulen..... | 15 |
| 2.5. Produktstrukturplan (PdSP)..... | 16 |
| 2.6. Scrum-Projektplan (Epics und Stories) | 16 |
| 3. Backend und Sensoreinbindung (Matteo Müller) | 20 |
| 3.1. Individuelle Aufgabenstellung | 20 |
| 3.2. Grundlagen / Methoden..... | 20 |
| 3.2.1. Werte für Propellerprüfstände..... | 20 |
| 3.2.2. Schubkraftmessung | 21 |
| 3.2.3. Drehmomentmessung..... | 23 |
| 3.2.4. Drehzahl- und Geschwindigkeitsmessung..... | 25 |
| 3.2.5. Rotorposition/Flügelposition | 28 |
| 3.2.6. Schutz-/Sicherheitskäfig für den Prüfstand..... | 29 |
| 3.2.7. Ansteuerung des Offsets mittels Servo | 31 |
| 3.2.8. Mini ESP32 zur Messung der Flügelposition | 33 |
| 3.2.9. Ansteuerung der Sensoren für die Messung | 35 |
| 3.2.10. SPI (Serial Peripheral Interface)..... | 39 |
| 3.2.11. 3-Wire SPI | 40 |
| 3.2.12. Messung TLE4998P3C | 41 |
| 3.2.13. Messung TLE4964-3M | 42 |
| 3.3. Realisierung / Implementierung | 44 |
| 3.3.1. Allgemein | 44 |
| 3.3.2. Steuerung des Prüfstandes | 44 |
| 3.3.2.1. Ermitteln des ESC-Signals | 44 |
| 3.3.2.2. ESP32 – Programmimplementierung | 46 |
| 3.3.2.3. Ausgangsstufe (3.3V auf 5V)..... | 47 |
| 3.3.2.4. Empfangen der Steuerdaten..... | 51 |
| 3.3.3. Messung der Flügelposition | 52 |

| | |
|--|----|
| 3.3.3.1. Interfaces der Winkelmessung | 52 |
| 3.3.4. Programmimplementierung..... | 52 |
| 3.3.4.1. Library-Überarbeitung für ESP32..... | 52 |
| 3.3.4.2. Einbindung in Programm | 53 |
| 3.3.4.3. Auslesen der Daten → Funktionstest (LogicAnalyser)..... | 53 |
| 3.3.4.4. Senden der Messdaten..... | 55 |
| 3.3.4.5. Schaltung + PCB TLE5012B | 56 |
| 3.3.4.6. Erkenntnisse | 56 |
| 3.3.5. Messung der Rotordrehzahl | 57 |
| 3.3.5.1. Funktionsweise TLE4964-3M (Hall-Switch)..... | 57 |
| 3.3.5.2. Programmimplementierung | 57 |
| 3.3.5.3. Schaltung + PCB TLE4964-3M | 58 |
| 3.3.6. Messung der Schubkraft | 59 |
| 3.3.6.1. Funktionsweise TLE4998P3C | 59 |
| 3.3.6.2. Programmimplementierung | 59 |
| 3.3.6.3. Schaltung + PCB TLE4998P3C | 61 |
| 3.3.6.4. Testaufbau..... | 62 |
| 3.3.6.5. Erkenntnis..... | 62 |
| 3.4. Ergebnisse / Konklusion..... | 63 |
| 4. Datenverarbeitung und Visualisierung (Lucas Lenarcic)..... | 64 |
| 4.1. Individuelle Aufgabenstellung | 64 |
| 4.2. Grundlagen / Methoden | 64 |
| 4.2.1. Allgemein | 64 |
| 4.2.2. Was ist Serverless und ein Custom Back-End (Serverfull) ? | 64 |
| 4.2.3. Vergleich Back-End Database | 64 |
| 4.2.4. Kriterien..... | 64 |
| 4.2.5. MongoDB | 65 |
| 4.2.6. MySQL | 66 |
| 4.2.7. Firebase | 66 |
| 4.2.8. MQTT..... | 67 |
| 4.2.9. Auswertung | 67 |
| 4.2.10. Vergleich Interface Back-End | 68 |
| 4.2.11. Kriterien..... | 68 |

| | |
|---|----|
| 4.2.12. Rest API | 68 |
| 4.2.13. Websockets | 69 |
| 4.2.14. Auswertung | 70 |
| 4.2.15. Hardware oder Webserver | 70 |
| 4.2.16. Raspberry Pi | 70 |
| 4.2.17. Google Cloud Platform | 70 |
| 4.2.18. Auswertung | 70 |
| 4.3. Realisierung / Implementierung | 71 |
| 4.3.1. Server Design | 71 |
| 4.3.1.1. Raspberry PI Image | 71 |
| 4.3.1.2. Verbinden zum Server | 72 |
| 4.3.1.3. Option 1: Anpassen der WLAN-Konfiguration | 72 |
| 4.3.1.4. Option 2: Verbinden mithilfe von Netzwerkkabel | 72 |
| 4.3.1.5. Verbinden über SSH | 72 |
| 4.3.1.6. Installieren der benötigten Services | 73 |
| 4.3.1.7. Einstellen der Startup Services | 74 |
| 4.3.1.8. Aufsetzen des Frontend | 75 |
| 4.3.1.9. Aufsetzen des Backend | 76 |
| 4.3.2. Backend – Schnittstelle Design | 77 |
| 4.3.2.1. Ordner Struktur | 78 |
| 4.3.2.2. API-Socket | 79 |
| 4.3.2.3. Models | 80 |
| 4.3.2.4. Endpoints / Routes | 81 |
| 4.3.2.5. Get Endpoint | 82 |
| 4.3.2.6. Post Endpoint | 82 |
| 4.3.2.7. PUT Endpoint: | 83 |
| 4.3.2.8. Testen der API | 85 |
| 4.3.3. Frontend – Visualisierung | 86 |
| 4.3.3.1. Frontend Dashboard Panel | 86 |
| 4.3.3.2. Chart Typen | 87 |
| 4.3.3.3. Line Chart | 87 |
| 4.3.3.4. Pie-Chart | 89 |
| 4.3.3.5. Bar-Chart | 90 |

| | |
|--|-----|
| 4.3.3.6. Display-Grid | 91 |
| 4.3.3.7. API-Einbindung | 93 |
| 4.3.3.8. Frontend Control Panel | 93 |
| 4.3.3.9. Control Panel Slider und Button. | 94 |
| 4.3.3.10. Test-Routine Konfigurator | 96 |
| 4.3.3.11. Status-Anzeigen | 101 |
| 4.4. Ergebnisse / Konklusion..... | 102 |
| 5. Mechanik – Entwicklung & Aufbau (Tim Sperle) | 103 |
| 5.1. Individuelle Aufgabenstellung | 103 |
| 5.2. Zielsetzung der Arbeit (Mechanik): | 103 |
| 5.3. Aufbau | 103 |
| 5.4. Erster Prototyp - Rotor | 104 |
| 5.4.1. Grundkonzeptentwicklung | 104 |
| 5.4.2. Komponenten..... | 104 |
| 5.4.2.1. Komponentenliste | 105 |
| 5.4.3. Zeichnungen | 105 |
| 5.4.4. Mechanischer Aufbau | 106 |
| 5.4.5. Funktionstest der Grundfunktion (Rotor) | 107 |
| 5.5. Zweiter Prototyp – Rotor..... | 107 |
| 5.5.1. Erweiterung des ersten Prototyps..... | 107 |
| 5.5.1.1. Verbindungen / Befestigungen..... | 107 |
| 5.5.1.2. Motor..... | 109 |
| 5.5.1.3. Servo | 110 |
| 5.5.1.4. ESC | 111 |
| 5.5.1.5. Zahnriemen / Zahnräder..... | 112 |
| 5.5.2. Weiterentwicklung des Rotors – Konzeptentwicklung | 112 |
| 5.5.3. Zeichnungen | 113 |
| 5.5.4. Mech. Aufbau | 115 |
| 5.5.5. Funktions- und Belastungstest | 115 |
| 5.6. Erster Prototyp – Prüfstand (Schubmessung)..... | 116 |
| 5.6.1. Konzept der Schubmessung | 116 |
| 5.6.2. Zeichnungen | 118 |
| 5.6.3. Mech. Umsetzung - Komplikation | 119 |

| | |
|--|-----|
| 5.7. Finales Endprodukt..... | 119 |
| 5.7.1. Rotor..... | 120 |
| 5.7.1.1. Technische Zeichnung..... | 120 |
| 5.7.1.2. Mech. Aufbau | 122 |
| 5.7.1.3. Zusammenbau | 123 |
| 5.7.2. Prüfstand | 124 |
| 5.7.2.1. Schubmessung..... | 124 |
| 5.7.2.2. Drehzahlmessung | 127 |
| 5.7.2.3. Tragflächen-Winkelmessung | 128 |
| 5.7.3. Produkteigenschaften | 130 |
| 5.7.3.1. Testungen | 130 |
| 5.7.3.2. Ergebnisse..... | 131 |
| 5.7.3.3. Zukünftige Weiterentwicklung | 132 |
| 6. Anhang..... | 133 |
| 6.1. Projektmanagement..... | 133 |
| 6.1.1. Aufgabenstellung des Gesamtprojekts | 133 |
| 6.1.2. Scrum-Projektplan..... | 135 |
| 6.2. Inbetriebnahme | 136 |
| 6.2.1. Server | 136 |
| 6.2.2. Mikrocontroller (Steuerung / Messung)..... | 136 |
| 6.3. Kostenaufstellung | 138 |
| 6.4. Besprechungsprotokolle Elektronik und Technische Informatik | 139 |
| 6.5. Besprechungsprotokolle Elektrotechnik..... | 143 |
| 6.6. Arbeitszeitnachweis..... | 146 |
| 6.7. Wettbewerbe | 155 |
| 7. Literaturverzeichnis | 156 |
| 8. Abbildungsverzeichnis | 162 |
| 9. Tabellenverzeichnis | 166 |

1. Einleitung

Ein neuartiger Antrieb in der Dronentechnik könnte die gesamte Luftfahrtindustrie revolutionieren. Denn das Prinzip des Voith-Schneider-Propellers ist aufgrund seiner Bauweise nicht nur im Bereich der Luftfahrt, sondern auch in Kombination mit der Fahrzeugs- und Schiffstechnik als Antrieb einsetzbar. Von daher ist die Themenstellung von großem Interesse, wie man die drei Sparten sinnvoll und vor allem ressourcensparend kombinieren kann und ob die Fliegenschaften und Daten ähnlich oder sogar besser sind als jene des herkömmlichen Dronenantriebs.

Um diese Daten für das Prinzip des Voith-Schneider-Propellers messbar zu machen, wird zu dem Bau des Rotors ein Prüfstand zum Messen verschiedenster wichtiger Parameter wie Drehzahl, Winkel der Flügel und die daraus resultierende Richtung wie auch Kraft des Schubs.

Als Ausgangslage liegt der bereits funktionierende Antrieb der Firma CycloTech und deren Ergebnisse vor, auf denen das Projekt aufbaut. Die Problemstellung bzw. die gesetzte Aufgabe der Diplomarbeit lautet, dass ein funktionsfähiger Rotor, nach Prinzip des Voith-Schneider-Propellers, über Luftverdrängung eine Schubkraft erzeugt. Die Steuerung der Drehzahl wie auch die Steuerung der Flügelposition soll gegeben sein und über eine Applikation anwendbar gemacht werden. Die Parameter des Rotors sollen mittels Sensoren von Infineon über einen Prüfstand ausgelesen und an eine Datenbank zur Visualisierung gesendet werden. Dadurch sollen wichtige Erkenntnisse über die Funktionalität und Effizienz des Rotors gewonnen werden.

Wie bereits erwähnt stellt Infineon Technologies Austria AG die Hall-Effekt Sensoren für die verschiedenen Messungen zur Verfügung. Die Implementierung der Sensoren erfolgt mit Expertise von Infineon durch Dr. Wolfgang Granig, der ebenso wie Dipl. -Ing Harald Grünanger für die Abteilung Elektronik und Technische Informatik und Dipl. -Ing Harald Huber für die Abteilung Elektrotechnik, als Projektbetreuer für die gesamte Diplomarbeit fungiert.

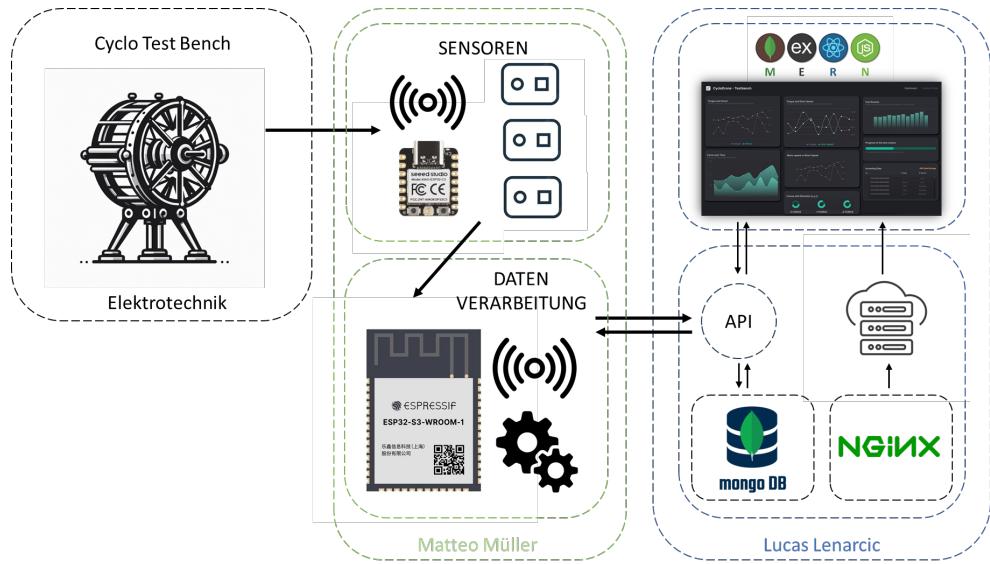


Abbildung 2: Systemstrukturplan

Der Rotor wird anhand des Voith-Schneider Prinzip entwickelt und hergestellt. Für die Durchführung von Messungen am Rotor ist es erforderlich, geeignete Sensoren auszuwählen und auf der Testbank zu montieren, die den Rotor auch fixiert. Zudem muss ein System zur Erfassung, Auswertung und Weiterverarbeitung der Daten am Rotor sowie zur Steuerung des Rotors entwickelt werden. Dabei muss eine zentrale Steuereinheit für die Sensoren sowie der Ansteuerung entwickelt werden. Die erfassten Daten sollen anschließend visualisiert und gespeichert werden, um wichtige Erkenntnisse zu gewinnen.

2. Grundlagen und Methoden

2.1. Ist-Situation / Stand der Technik

Drohnen gibt es mittlerweile in fast jeder Bauform und Baugröße. Dabei erfolgt der grundsätzliche Antrieb, egal ob mit vier oder mit acht Propellern, immer nach demselben Prinzip. Für eine geradlinige Vorwärtsbewegung müssen allerdings alle diese Drohnen eine Neigungsänderung aufweisen. Für entsprechend größere Drohnen, welche zum Transport von empfindlichen Gegenständen genutzt werden, könnte diese Eigenschaft zu einem Problem führen. Aus diesem Grund beschäftigen wir uns im Rahmen unserer Diplomarbeit mit der Entwicklung eines alternativen Antriebssystems. Wir entwickeln und testen einen Cyclorotor, um die Grundlagen für die Konstruktion einer Drohne zu schaffen, die dazu in der Lage wäre, sich ohne Neigungsänderung vorwärtszubewegen.

Ein weiterer großer Vorteil der Konstruktion eines solchen Antriebs ist, dass ein Vehikel, das mit vier dieser Rotoren ausgestattet ist, nicht nur fliegen könnte. Mit der horizontalen Ausrichtung der drehenden Achse der Konstruktion, wäre es auch in der Lage, sich an Land vorwärtszubewegen und so wie ein herkömmliches Auto zu fahren.

Zudem kann der Rotor die Schubrichtung um die ganze Achse einstellen, daher ist es auch möglich Luft oder auch ein anderes Medium nach oben zu verdrängen. Somit würde sich der Antrieb auch für ein Fahrzeug eignen, welches sich unter Wasser in alle Richtungen bewegen möchte.

Das österreichische Unternehmen CycloTech arbeitet seit dem Jahr 2017 an einem Fahrzeug mit dieser Antriebsart. Die folgende Abbildung zeigt einen dieser hochkomplexen Rotoren der eine Spannweite von 42cm aufweist und in der Lage ist, eine maximale Schubkraft von 253N zu generieren. Dabei besitzt dieses Konstrukt fünf Tragflächen, die sich horizontal um die Achse drehen. Im Betrieb erreicht dieser Rotor eine Drehzahl von bis zu 3100U/min. Die Ingenieure/innen dieser Firma entwickelten ein Fahrzeug, welches mit vier dieser Rotoren ausgestattet ist. Im Jahr 2023 erreichte dieses Unternehmen die ersten Ergebnisse mit diesem Flugkörper. Ein Outdoor-Flug mit dem insgesamt 83kg schweren Cyclo-Copters konnte erfolgreich durchgeführt werden. [1]

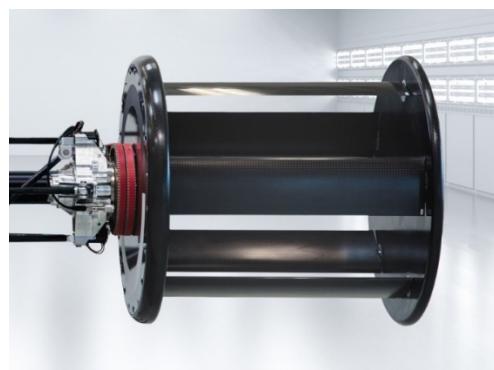


Abbildung 3: CycloTech Rotor [1]

2.2. Lösungsmethoden / Lösungsansatz

Nachdem es außer diesem oben genannten Unternehmen keine weiteren Organisationen oder Institute gibt, welche an der Entwicklung derartiger Antriebe forschen, mussten die meisten Konzepte des Rotors selbst entworfen werden. Einzelne YouTuber wie beispielsweise Jude Schauer, der selbst einen Prototyp eines solchen Fahrzeugs konstruiert hat, lieferte uns für die mechanische Funktionalität einige Inspirationen. Da auch unser Projekt in einem modellbauhaften Maßstab gebaut werden soll, wenden wir uns an dafür ausgelegte Komponenten.

Um den Rotor zu Fertigen werden mittels SolidWorks Skizzen erstellt. Diese werden wiederum in weiterer Folge bearbeitet und mit dem 3D-Drucker gefertigt.

Komponenten des Rotors, welche eine höhere Festigkeit aufweisen müssen, um den bei Rotation entstehenden Fliehkräften, standzuhalten, werden aus Aluminium gefräst. Dieser Fräsprprozess wurde über die CNC-Fräsmaschine der Schule möglich gemacht.

Als Antrieb für die Rotation der Konstruktion wird ein modellbauspezifischer Brushlessmotor verwendet und auch ein ähnlich konfigurierter Servo steuert die Schubrichtung des Rotors.

Zur Steuerung dieser Komponenten sollen vorwiegend Mikrocontroller verwendet werden, die die einzelnen Komponenten durch Befehle von einer separaten selbst konstruierten API und Datenbank steuern. Diese Daten sollen gesendet bzw. geholt werden und somit den Rotor je nach Eingabe steuern.

Ebenso werden zur Messung verschiedenster wichtiger Vergleichsdaten Hall Sensoren von Infineon herangezogen. Diese werden auch mittels Mikrocontroller implementiert und die gemessenen Daten an die Datenbank gesendet.

Es muss ein Server mit einer geeigneten Datenbank zur Speicherung der Messdaten ausgewählt und konfiguriert werden. Zudem muss eine Schnittstelle für die Messstation sowie das Front-End zur Visualisierung geplant, entwickelt und getestet werden.

Ein Front-End zur Visualisierung sowie der Steuerung soll entwickelt werden, um bedeutende Erkenntnisse und Muster aus dem Verhalten des Rotors abzuleiten.

2.3. Grundlagen für das Gesamtprojekt

Die Grundlagen des Projektes bauen bzw. schließen sich aus dem Prinzip des Voith-Schneider-Propellers.

Wie zuvor erwähnt wird dieses Prinzip von der Firma Cyclotech angewendet. Es wurde allerdings schon einige Jahre früher entwickelt. Der österreichische Erfinder Ernst Leo Schneider hat 1927 den sogenannten Voith-Schneider-Propeller konstruiert. Dieser Propeller dient bis heute als effektiver Antrieb von Schiffen. Dabei besteht dieser aus einer Scheibe, die an der Unterseite von Schiffen befestigt ist, aus welcher vier bis sechs Flügel herausragen. Der große

Vorteil dieses Schiffsantriebs ist, dass durch das Variieren des Anstellwinkels jedes einzelnen Flügels (Tragfläche) die Schubrichtung um 360° verändert werden kann. Die Änderung ist zudem sehr schnell möglich. [2]

2.4. Schnittstellen zwischen den Modulen

Wie in Abbildung 4 zu erkennen ist, gibt es zwischen den einzelnen Aufgabenbereichen mehrere verschiedene Schnittstellen.

Unter anderem wären dies zwischen der Datenbank bzw. API und der Steuerung bzw. der Messung des Prüfstandes mit Mikrocontrollern die Übertragung und der Erhalt von Daten durch HTTP Requests. Diese werden auch bei der Schnittstelle zwischen dem Front-End also der Visualisierung der Daten und der API, also die Schnittstelle zur Datenbank verwenden.

Auf der anderen Seite gibt es zwischen der Steuerung bzw. der Messung des Prüfstandes mit Mikrocontrollern und dem mechanischen Aufbau des Prüfstandes Schnittstellen zwischen der Steuerung des Rotors und zwischen den Messungen und Messaufbauten. Dies wäre bei der Schubkraftmessung ein PWM-Signal von 300Hz, bei der Messung der Drehzahl ein digitales HIGH- bzw. LOW-Signal und bei der Messung der Flügelposition 3-Wire-SPI.

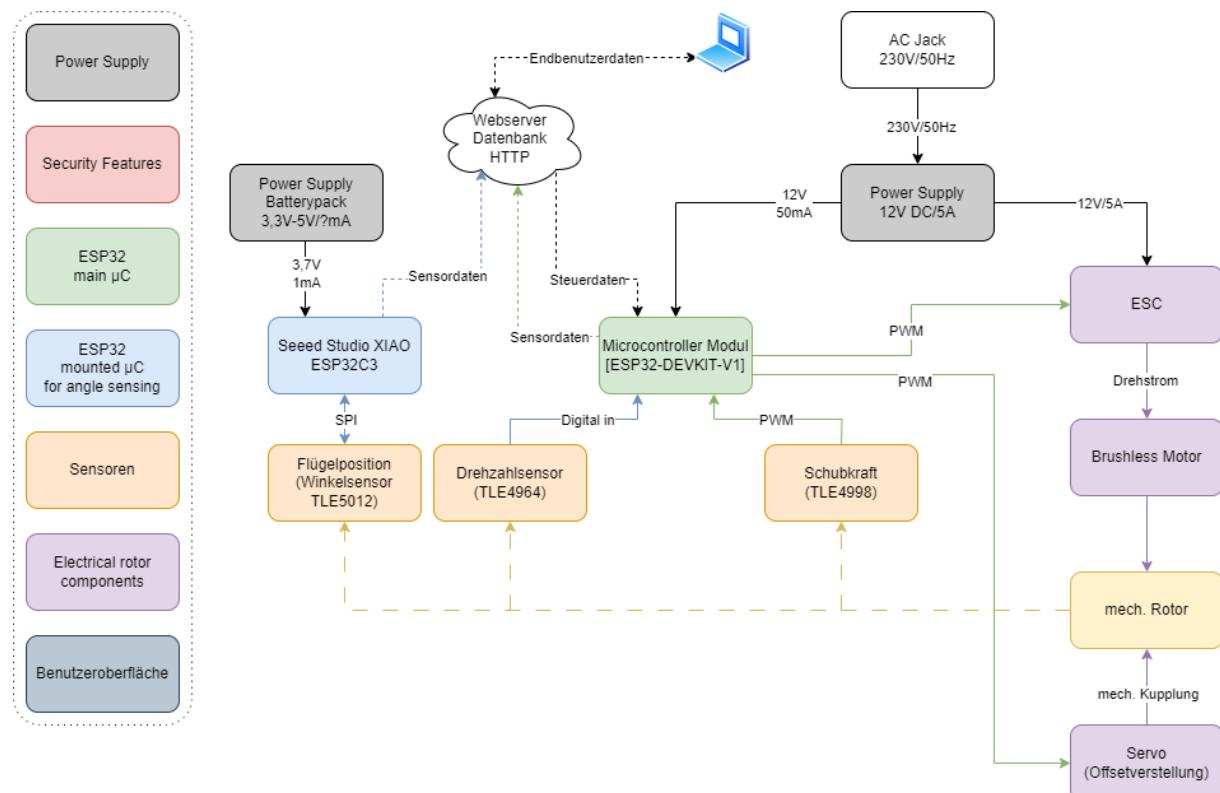


Abbildung 4: Blockschaltbild

2.5. Produktstrukturplan (PdSP)

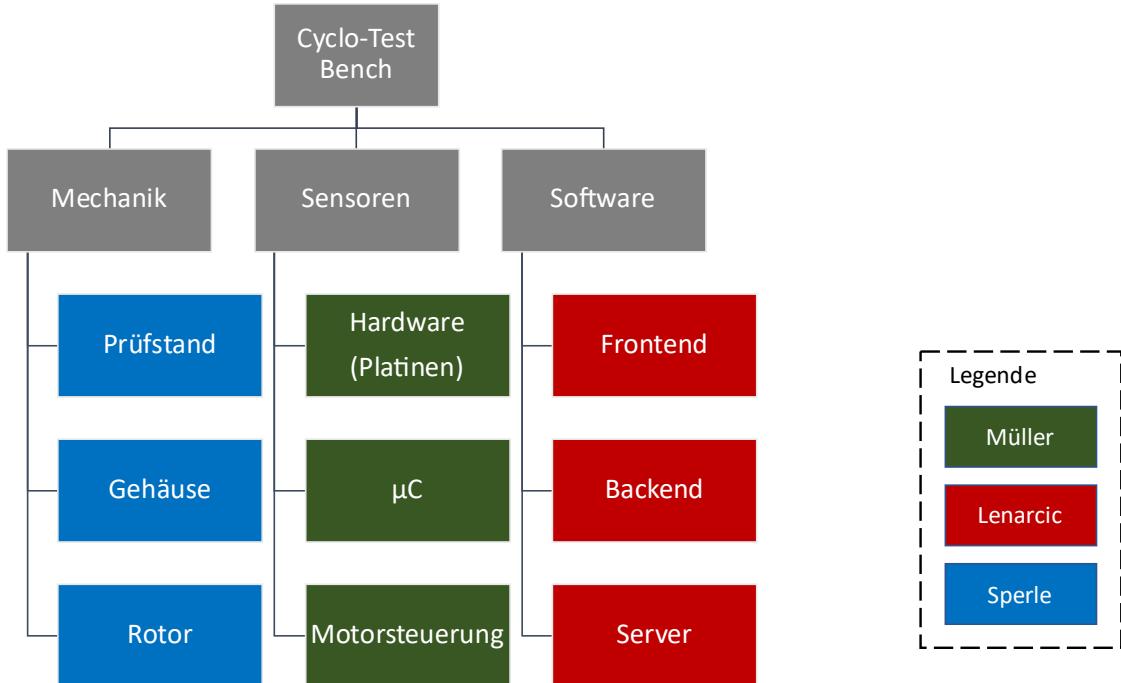


Abbildung 5: Produktstrukturplan

2.6. Scrum-Projektplan (Epics und Stories)



Abbildung 6: Scrum Projektplan Dokumentation

CTB – Diplomarbeit [2023/24]

2 Messdaten & Steuerung

21 Der Auftraggeber will eine genaue Auflistung und die ersten Schaltungen der verschiedenen verwendeten Sensoren.

22 Der Auftraggeber möchte für jeden der verwendeten Sensoren ein geeignetes Programm haben, mit dem die Daten der Sensoren situationsbezogen verarbeitet werden.

23 Der Auftraggeber verlangt für alle Sensoren eine Platine zum Auslesen der Daten für die effiziente Übertragung an die Datenbank.

24 Der Auftraggeber möchte eine Applikation zum Senden und Empfangen der Daten in und an eine API haben. Aus dieser sollen Controldataen gelesen und Messdaten hochgeladen werden.

25 Der Auftraggeber fordert eine Steuerung eines BLDC-Motors mittels eines Mikrocontroller zur Kontrolle der Drehzahl.

26 Der Auftraggeber verlangt eine finale Version zum Übertragen der gesamten Sensordaten an die API, sowie zum Auslesen der Controldataen für die Steuerung des Motors und des Servos.

27 Der Auftraggeber will ein Programm zur theoretischen Messung des Schubs haben, welches den Duty Cycles des dafür zu verwendenden Sensors misst verarbeitet und ausgibt.

Abbildung 7: Scrum Projektplan Messdaten und Steuerung Matteo Müller

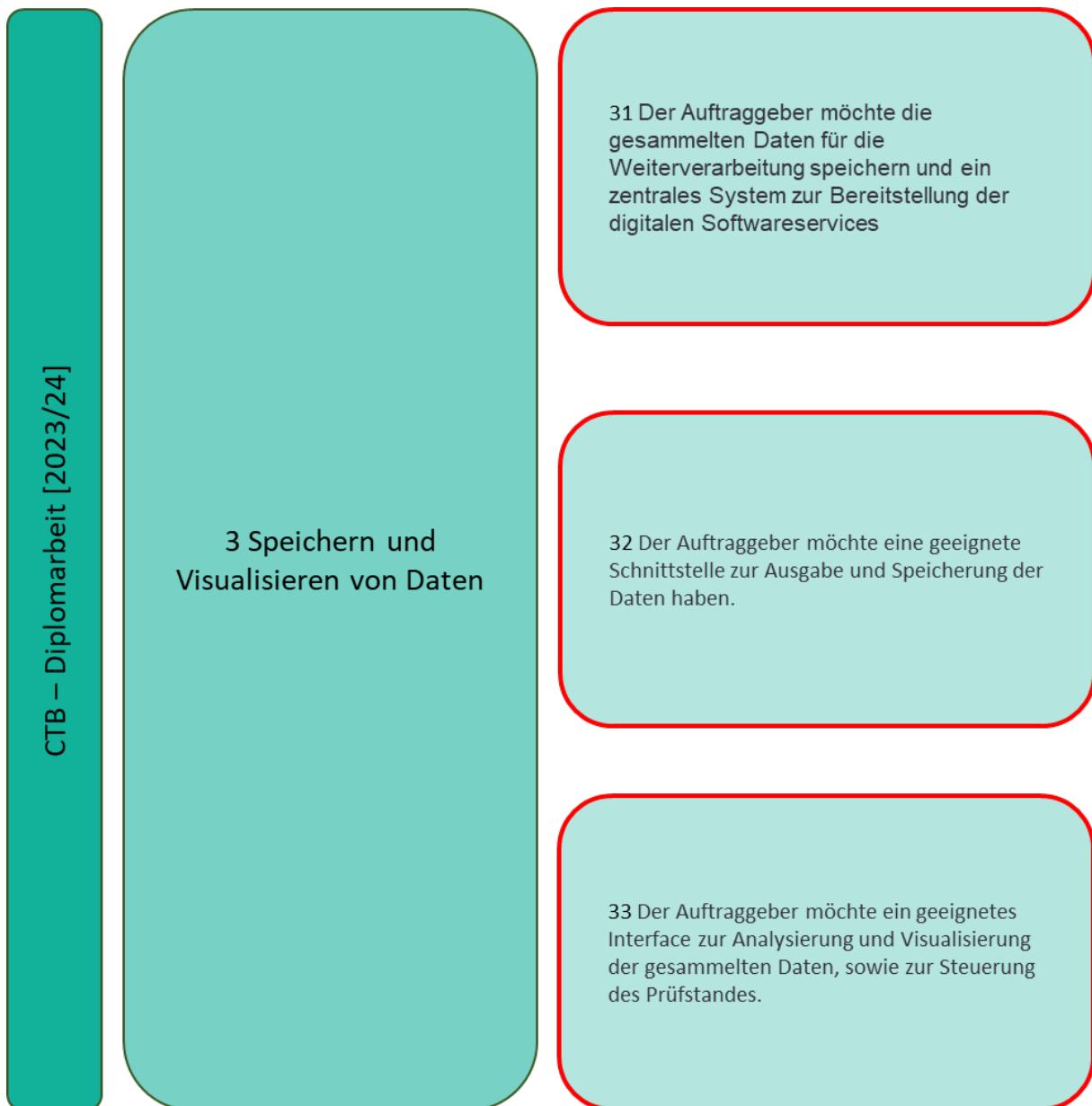


Abbildung 8: Scrum Projektplan Speichern und Visualisieren von Daten Lucas Lenarcic

3 Mechanischer Aufbau

31 Der Auftraggeber will ein simulierte mechanisches Modell des Void-Schneider-Propellers.

32 Der Auftraggeber möchte einen mechanischen Aufbau dieses Void-Schneider-Propellers als Rotor.

33 Der Auftraggeber verlangt eine Ansteuerung des Cyclo-Rotors über einen Elektromotor.

34 Der Auftraggeber will eine funktionsfähige Ansteuerung der Winkel der Tragflächen über einen Servomotor.

35 Der Auftraggeber fordert ein Gerüst, welches den Rotor in einer gewissen Höhe hält und eine dem Gewicht entsprechende Basis.

36 Der Auftraggeber verlangt methodische Konstruktionen zur Tragflächenwinkelmessung und Schubkraftmessung.

37 Der Auftraggeber fordert einen Schutzkäfig um den Rotor als Sicherheitsmaßnahme vor fliegenden Teilen im eventuellen Notfall im Betrieb.

Abbildung 9: Scrum Projektplan Mechanischer Aufbau Tim Sperle

4. Datenverarbeitung und Visualisierung (Lucas Lenarcic)

4.1. Individuelle Aufgabenstellung

Umfasst den Aufbau eines Servers, der die benötigten Services hosten sowie effektiv die Daten verarbeiten kann, die Konfiguration einer geeigneten Datenbanklösung für unsere Bedürfnisse, die Auswahl eines Webhosting-Services, die Entwicklung einer Schnittstelle für das Frontend und die Programmierung des Frontends zur Visualisierung der Daten.

4.2. Grundlagen / Methoden

4.2.1. Allgemein

Es ist zu evaluieren, wie das Backend aufgebaut sein muss, um den spezifischen Eigenschaften und Belastungen des Projekts standzuhalten. Da die Diplomarbeit einen experimentellen Ansatz verfolgt, ist es im Vorfeld nur möglich, grobe Schätzungen hinsichtlich des erwarteten Datenvolumens abzugeben. Mögliche Erweiterungen, genauere Messungen und die Implementierung in andere wissenschaftliche Projekte müssen bei der Auswahl berücksichtigt werden, um die Integration reibungslos zu gestalten und mögliche Fehlerquellen vorzubeugen.

4.2.2. Was ist Serverless und ein Custom Back-End (Serverfull) ?

Serverless Computing ist ein Modell, bei dem die gesamte Infrastruktur in der Cloud von einem Anbieter verwaltet wird. Daher ist es sehr einfach, das Backend zu verwalten, zu skalieren und Ressourcen verschiedenen Diensten zuzuweisen. Auch das schnelle Deployen neuer Versionen von Diensten ist ein Vorteil. Die Nachteile von Serverless sind vor allem die Kosten und die eingeschränkte Kontrolle, die bei bestimmten Anforderungen zu Problemen führen können. Bei einem Custom Backend wird eine eigene Datenbank, API (Application Programming Interfaces) oder Ähnliches sowie ein Website-Hosting-Server auf einem eigenen Server gestartet. Dadurch entstehen nur wenige laufende Kosten, und die Infrastruktur kann leicht angepasst werden. [34]

4.2.3. Vergleich Back-End Database

4.2.4. Kriterien

Bei der Auswahl der Datenbank ist es wichtig, Kriterien wie Datenstruktur und -modell, Skalierbarkeit, Kosten und Aufwand zu beachten. Da die Datensammlung, Evaluation und Präsentation des Antriebs im Vordergrund stehen, muss das Speichern und Übertragen von Daten reibungslos passieren. Aufgrund der Vielzahl von Sensoren und der Menge an Daten ist zu erwarten, dass eine entsprechend große Datenlast entsteht.

4.2.5. MongoDB

MongoDB ist eine nichtrelationale Datenbank. Sie wird für Big Data-Anwendungen verwendet, also für Daten, die schnell und in großen Mengen anfallen. Durch ihre Struktur kann MongoDB besser als SQL-Lösungen skaliert werden. Mit MongoDB sind komplexe Datenanalysen und effiziente als auch schnelle Abfragen wie Filterung, Gruppierung oder Transformation. Flexible Schemas erleichtern die Anpassung des Schemas, wenn sich die Anforderungen ändern. MongoDB unterstützt mittlerweile die meisten Programmiersprachen, wodurch sie vielseitig einsetzbar und einfach zu integrieren ist. Joins können in MongoDB nur schwer implementiert werden, obwohl wir wahrscheinlich keine Joins benötigen werden. [35] [36]

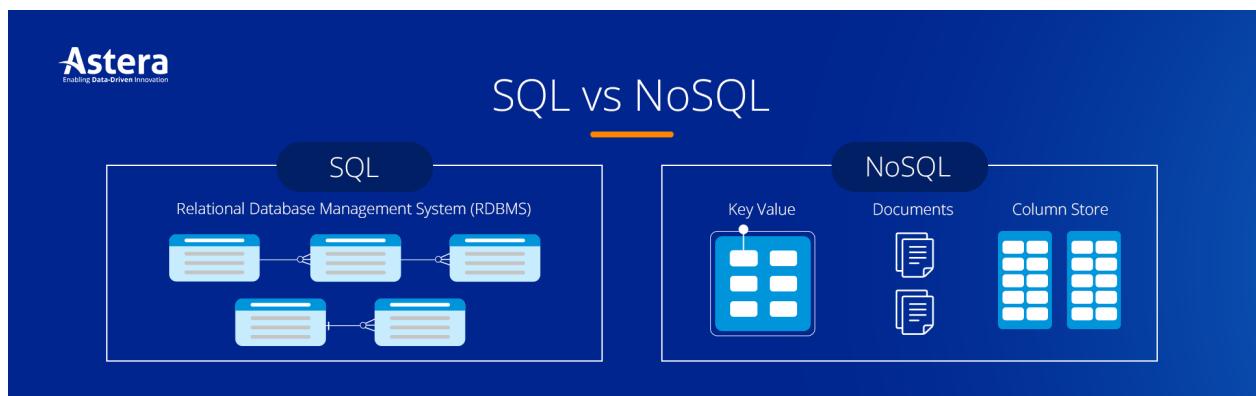


Abbildung 55: Vergleich SQL und NoSQL [37]

Vorteile:

- schnelles Abfragen und Analysieren von Daten
- einfache Anpassung des Datenbankschemas
- einfache Implementation und verwendbar mit vielen Programmiersprachen
- einfach skalierbar

Nachteile:

- Joins sind schwer zu Implementierung
- hoher Speicherverbrauch
- mögliche duplizierte Daten

4.2.6. MySQL

MySQL ist eine relationale Datenbank. MySQL wird für Anwendungen verwendet, bei denen Daten strukturiert und in tabellenähnlichen Strukturen organisiert sind. Dadurch lassen sich Joins einfach implementieren. Ein weiterer Vorteil von MySQL ist der niedrige Speicherverbrauch durch das strukturierte Speichern und Deduplizieren von Daten. MySQL ist in der Industrie weit verbreitet, es hat sich auch über die Zeit bewährt und Stabilität gezeigt. Während MySQL viele Programmiersprachen unterstützt, kann die Integration in einigen von ihnen komplexer als bei anderen Lösungen sein. [38] [39]

Vorteile:

- relationale Struktur
- SQL-Sprache – Leichtes Bearbeiten und Abfragen
- niedriger Speicherverbrauch
- bewährte Zuverlässigkeit

Nachteile:

- starres Schema
- Komplexität bei der Implementation
- mangelnde Skalierbarkeit für Big Data

4.2.7. Firebase

Firebase ist ein sogenanntes Backend-as-a-Service, welches eine Realtime Database, Datenspeicher, Nutzer-Authentifizierung und vieles mehr in der Cloud verbindet. Firebase hat Developer SDKs für die meisten Plattformen, Frameworks und Programmiersprachen und reduziert dadurch den Backend-Code-Aufwand. Da Firebase serverlos ist, muss außerdem kein Server aufgesetzt und gewartet werden. Firebase in seiner Grundkonfiguration ist jedoch leider nicht darauf ausgelegt, viele Daten schnell hintereinander zu verarbeiten. Da Firebase ein Service ist, können auch Kosten bei starker Nutzung anfallen. [40] [41]

Vorteile:

- einfache Nutzung und Implementation
- verschiedene zusätzliche Services
- Serverless und skalierbar

Nachteile:

- mögliche Kosten
- limitierte Daten-Bandbreite
- umständliches Filtern und Suchen
- Daten in Cloud

4.2.8. MQTT

MQTT ist ein Nachrichtenprotokoll für Netzwerke mit geringer Bandbreite und IoT-Geräte. Es wird in Umgebungen mit hoher Latenz und niedriger Bandbreite oder zwischen zwei Maschinen eingesetzt. MQTT verbraucht nur eine minimale Menge an Code und minimale Ressourcen. Außerdem verwaltet MQTT die Datenbank und der MQTT-Broker die Verbindung zum Front-End und Mikrocontroller. [42] [43]

Vorteile:

- effiziente und kleine Messages
- minimaler Aufwand am Mikrocontroller

Nachteile:

- kleine Message Datengröße
- eingeschränkte Funktionen
- großer Aufwand beim MQTT-Broker aufsetzten

4.2.9. Auswertung

Dadurch dass wir schnell Daten schreiben und auslesen möchten, fallen alle Serverless Lösungen wie Firebase weg. Die Verzögerung beim Schreiben und Auslesen sowie die möglichen Kosten, die beim ständigen Nutzen und Testen entstehen können, sind starke Nachteile dieser Lösung. Bei MQTT ist das Risiko, dass es bei vielen Daten in kleinen Zeitabständen zu Verzögerungen und Überlastungen kommt. Außerdem kann der Aufwand des Aufsetzens eines MQTT-Brokers nicht vernachlässigt werden. Damit bleiben noch Datenbanken die Lokal aufgesetzt werden, wie MySQL und MongoDB. Zwischen den beiden gibt es nur kleinere Unterschiede, jedoch scheint es, als wäre MongoDB flexibler und teilweise auch schneller als MySQL. Daher sollte MongoDB als Datenbank im Back-End verwendet werden.

4.2.10. Vergleich Interface Back-End

4.2.11. Kriterien

Wir brauchen eine schnelle Verbindung zwischen Frontend und der Datenbank. Da es sich um eine Diplomarbeit und keine produktionsfertige Anwendung handelt, benötigen wir fast kein bis kein Error Handling. Mögliche Anfragen an den Server wären das Auslesen von Sensordaten aus der Datenbank, das Abrufen von im Backend berechneten Werten sowie das Einstellen von Abläufen/Tests und Rotor/Motor-Steuerungen. Die Abfragerate und die Geschwindigkeit der Anfragen und Antworten sind das Wichtigste bei der Auswahl des Interfaces. Auch muss die Schnittstelle flexibel sein, um mögliche Erweiterungen schnell implementieren zu können.

4.2.12. Rest API

Eine REST-API ist ein Interface, das es unter definierten Regeln ermöglicht, Daten zu senden oder von einer Datenbank zu empfangen. Mit einer REST-API können unter festgelegten Regeln HTTP-Anfragen bearbeitet werden. REST-APIs erleichtern das Lesen und Schreiben in die Datenbank und sind leicht und effizient zu handhaben. Außerdem lassen sie sich einfach bearbeiten und modifizieren. Eine Rest-API nutzt standardisierte HTTP-Methoden wie GET, POST und DELETE, während eine Standard-API oft durch individuelle Methoden definiert ist. [44]

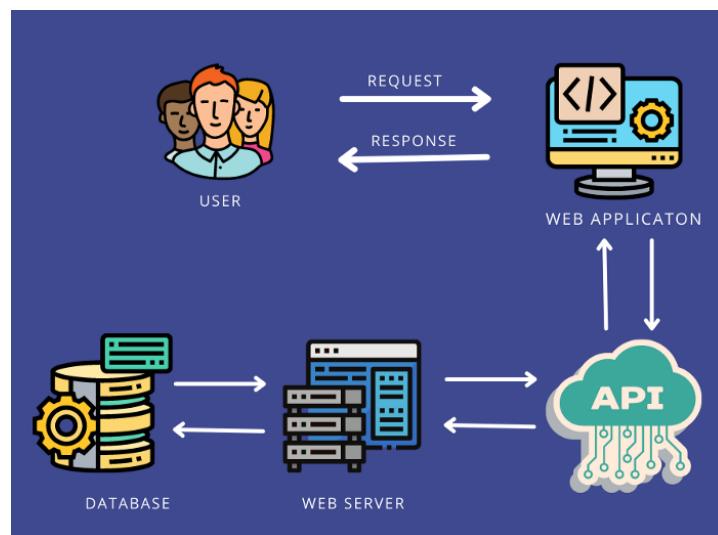


Abbildung 56: API-Prinzip [45]

Vorteile:

- einfache Verwendung am Front-End und Mikrocontroller
- plattformunabhängig
- flexibel

Nachteile:

- komplex in der Entwicklung
- schlechte Echtzeitkommunikation
- separate http-Anfragen für jede Aktion

4.2.13. Websockets

Ein WebSocket ist eine "live" Verbindung zwischen zwei Geräten. Dabei können Daten so lange gesendet werden, bis eine der beiden Parteien die Verbindung trennt. Dies ermöglicht es beiden Seiten, Daten in Echtzeit auszutauschen, ohne wiederholte Anfragen und Antworten. WebSockets werden für Live-Anwendungen wie Online-Chats oder Online-Spiele verwendet, bei denen die Geschwindigkeit im Vordergrund steht. [46]

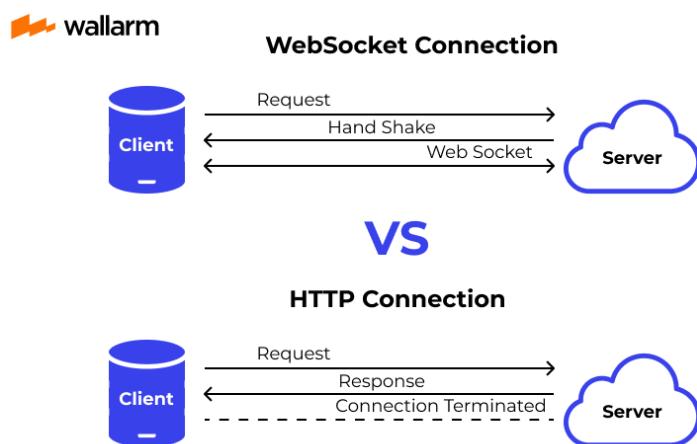


Abbildung 57: Prinzip Websockets [47]

Vorteile:

- Echtzeitaktualisierung
- kleinere Datenpakete
- niedrige Latenz

Nachteile:

- komplex am Frontend und Mikrocontroller zu implementieren
- Kompatibilität

4.2.14. Auswertung

Um eine "saubere" Schnittstelle zu gewährleisten, wäre eine REST-API die beste Wahl. Dadurch könnte unsere Technik leicht von anderen Entwicklerteams integriert werden, und die Daten, die von unserem Rotor produziert werden, könnten verarbeitet werden. Die leicht erhöhte Latenz gegenüber Websockets kann ebenfalls ignoriert werden. Durch eine REST-API wäre es auch sehr einfach, weitere Services an die Rotor-Infrastruktur anzuschließen.

4.2.15. Hardware oder Webserver

4.2.16. Raspberry Pi

Ein Raspberry Pi ist ein kostengünstiger, kleiner Computer, der verschiedenste Dienste ausführen kann. Durch die weite Verbreitung und den Open-Source Ansatz gibt es für die meisten Probleme schon vorgefertigte Software-Pakete, welche den Aufbau eines Custom Backends vereinfachen können. Aufgrund der begrenzten Leistung und der Skalierbarkeit muss man sich bereits im Vorhinein Gedanken über die maximalen Rechen- und Speicherressourcen machen. Ein weiterer kleiner Vorteil wäre, dass alles lokal am Raspberry Pi läuft und Daten darauf auch lokal gespeichert werden. [48]

4.2.17. Google Cloud Platform

Google Cloud Platform ist eine skalierbare Cloud-Computing-Plattform, die umfassende Lösungen für das Hosting eines Backends mit Datenbank, die Verbindung zum Frontend und das Hosting eines Webservers bietet. Dienste wie Firebase und Google Cloud Storage ermöglichen das einfache Speichern und Verarbeiten von Daten. Daneben ist das Hosting einer Webseite einfach und flexibel. Die Möglichkeit zur Erweiterung und zum umfassenderen Nutzen der Dienste ist ebenfalls unkompliziert umsetzbar. Dadurch können jedoch neben den laufenden Kosten auch schnell zusätzliche Kosten anfallen. [49]

4.2.18. Auswertung

Dadurch, dass der Teststand mit erhöhter Wahrscheinlichkeit auf einer Messe stehen wird und dabei so schnell und reibungslos wie möglich aufgebaut werden soll, wäre ein Serversystem mit einem oder mehreren Raspberry Pi-Geräten in einem Cluster die einfachste Lösung. Die Nutzung von Google Cloud bringt zwar einige Vorteile mit sich, jedoch überwiegen die negativen Aspekte wie die möglichen Kosten und die zentralisierte Serverstruktur die Vorteile. Daher ist es sinnvoll, ein Raspberry Pi-Cluster zu verwenden, um die Datenbank und die Website-Hosting-Server lokal und zentralisiert zu betreiben.

4.3. Realisierung / Implementierung

In diesem Abschnitt wird beschrieben, wie die ausgewählten Technologien, darunter die REST-API realisiert mit Node.js, MongoDB als Datenbank, React als Frontend-Framework und einen Raspberry Pi als Server, in das Projekt implementiert wurden und wie das Visualisieren, Speichern und Verarbeiten der Messdaten realisiert wurde.

4.3.1. Server Design

Es ist notwendig, einen Server einzurichten, um die Weboberfläche zu hosten, die gemessenen Daten zu speichern sowie eine Schnittstelle zu betreiben, über die sowohl die Weboberfläche zur Visualisierung als auch die Messstation geregt Zugriff auf die Datenbank haben. Zudem musste auch der Raspberry Pi konfiguriert werden, um eine einfache Verbindung, automatisches Starten der Services sowie die korrekte Konfiguration im Netzwerk zu gewährleisten.

4.3.1.1. Raspberry PI Image

Um den Raspberry Pi verwenden zu können, muss zuerst ein Betriebssystem auf eine SD-Karte übertragen werden. Dabei kann man, bei Verwendung des offiziellen Raspberry Pi Imagers, schon jetzt das einfache Verbinden zum Server über die SSH-Konfiguration einstellen. Auch das Einstellen eines Hostnamens, welches ähnlich wie DNS funktioniert, ist jetzt schon möglich und erlaubt die sofortige Nutzung in verschiedenen Netzwerken. Es vereinfacht außerdem auch das Kommunizieren der Schnittstelle mit dem Server und dem Frontend. Als letztes wird das WLAN-Passwort für ein Netzwerk ausgewählt, wobei das System auch ohne WLAN und nur mit einem Ethernet-Kabel betrieben werden kann.



Abbildung 59: Raspberry PI Imager

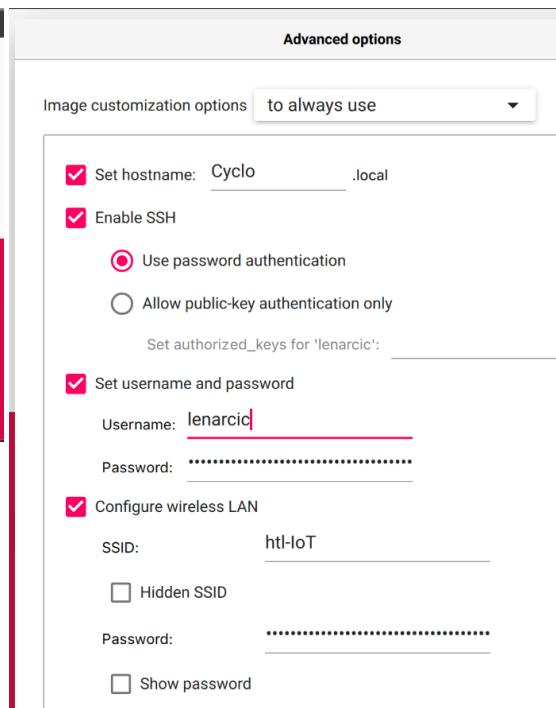


Abbildung 58: Raspberry PI Imager Settings

4.3.1.2. Verbinden zum Server

Das Verbinden zum Server über SSH ist sehr einfach. Davor muss jedoch noch entweder die WLAN-Konfigurationsdatei des Raspberry-Images angepasst oder der Raspberry mit einem Ethernet-Kabel an das Netzwerk angeschlossen werden.

4.3.1.3. Option 1: Anpassen der WLAN-Konfiguration

Es muss eine neue Datei auf der SD-Karte mit dem Namen `wpa_supplicant.conf` erstellt werden. In diese kommt folgender Code, wobei die `ssid` und das Passwort eingestellt werden müssen.

```
ctrl_interface=DIR=/var/run/wpa_supplicant GROUP=netdev
update_config=1
country=<Country Code>

network={
    ssid=<SSID>
    psk=<PASSWORD>
    scan_ssid=1}
```

4.3.1.4. Option 2: Verbinden mithilfe von Netzwerkabel

Der Raspberry muss hierbei nur mit Strom versorgt und sein Ethernet-Port mit dem Netzwerk verbunden sein.

4.3.1.5. Verbinden über SSH

Zuletzt kann der Raspberry über den eingestellten `hostname` und `username` erreicht werden. Dabei wird der command: `ssh username@hostname` verwendet. Nach eingeben des eingestellten Passworts hat man sich erfolgreich mit dem Server verbunden.

```
Last login: Sat Mar  2 10:38:00 on ttys003
(base) lemme@lemmes-MBP ~ % ssh lenarcic@Cyclo.local
[ lenarcic@cyclo.local's password:
Linux Cyclo 6.1.21-v8+ #1642 SMP PREEMPT Mon Apr  3 17:24:16 BST 2023 aarch64

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Sat Mar  2 10:38:28 2024 from fe80::cd3:86f2:cf09:efb1%eth0
[ lenarcic@Cyclo:~ $ uname -a
Linux Cyclo 6.1.21-v8+ #1642 SMP PREEMPT Mon Apr  3 17:24:16 BST 2023 aarch64 GN
U/Linux
[ lenarcic@Cyclo:~ $ 
[ lenarcic@Cyclo:~ $ ]
```

Abbildung 60: Terminal - SSH Verbindung aufgebaut

4.3.1.6. Installieren der benötigten Services

Die Services, die für den Betrieb des Servers benötigt werden, sind die folgenden:

- MongoDB
- Nginx
- NodeJS

Diese sind im bereitgestellten Image bereits vorinstalliert. Die Dienste müssen nicht sofort konfiguriert werden. Die einzelnen Anpassungen erfolgen bei der Installation des Backend- und Frontend-Codes.

```
sudo apt update
sudo apt install nginx
curl -fsSL https://deb.nodesource.com/setup_14.x | sudo -E bash -
sudo apt install -y nodejs
```

Da MongoDB die Raspberry Pi-Plattform nicht direkt unterstützt, müssen die angepassten Binaries von GitHub heruntergeladen werden. Beim Testen ergaben sich keine großen Unterschiede zur herkömmlichen Version.

```
$ mkdir ~/mdb-binaries && cd ~/mdb-binaries
$ wget https://github.com/themattman/mongodb-raspberrypi-binaries/releases/download/r6.0.13-rpi-unofficial/mongodb.ce.pi4.r6.0.13.tar.gz
$ tar xzvf mongodb.ce.pi4.r6.0.13.tar.gz # Decompress tarball

$ mkdir -p /data/db/test_db
$ touch /data/db/test_db/mongod.log
$ sudo chown -R ${USER}:${USER} /data

$ ./mongod --dbpath /data/db/test_db --fork --logpath /data/db/test_db/mongod.log --port 28080
$ ./mongo --port 28080 # run queries!
```

Nach dem Herunterladen können die Services gestartet und getestet werden. Dafür können folgende Befehle verwendet werden. Zuerst kann man überprüfen, ob die Services richtig installiert wurden, und danach, ob man sie starten kann.

```
mongod --version; nginx -v; node -v; npm -v
sudo systemctl start mongodb
sudo systemctl start nginx
```

```

db version v5.0.5
Build Info: {
  "version": "5.0.5",
  "gitVersion": "d65fd89df3fc039b5c55933c0f71d647a54510ae",
  "openSSLVersion": "OpenSSL 1.1.1n 15 Mar 2022",
  "modules": [],
  "allocator": "tcmalloc",
  "environment": {
    "distarch": "aarch64",
    "target_arch": "aarch64"
  }
}
nginx version: nginx/1.18.0
v16.20.2
8.19.4
  
```

Abbildung 61: Installierte Services Informationen

4.3.1.7. Einstellen der Startup Services

Die Services, wie die Datenbank, das Webhosting und die API-Schnittstelle, sollen alle automatisch beim Booten des Betriebssystems starten. Dafür muss ein Start-up-Skript erstellt und ausführbar gemacht werden. Dieses ist bereits beim Herunterladen der bereitgestellten Image-Datei vorkonfiguriert.

`nano startup_script.sh`

Danach muss folgender Code in das Skript eingegeben und der Pfad zum Backend angepasst werden:

```

#!/bin/bash
sudo service nginx start
sudo mongod -bind_ip 0.0.0.0
cd /home/<User>/Cyclo-Bench-Backend
npm start
  
```

Das Skript startet zuerst den Webhosting-Server Nginx, dann die Datenbank und öffnet alle Ports der Datenbank. Dann wechselt es in das Backend-Verzeichnis und startet die API.

Zuletzt muss das Skript ausführbar gemacht und zu den systemd-Services hinzugefügt werden.

```

chmod +x startup_script.sh
ExecStart=/home/<User>/startup_script.sh
  
```

Anschließend kann der Raspberry Pi optional zum Testen neu gestartet werden.

```
[lenarcic@Cyclo:~ $ cat startup_script.sh
#!/bin/bash

sudo service nginx start
sudo mongod --bind_ip 0.0.0.0
cd /home/lenarcic/Cyclo-Bench-Backend
npm start

[lenarcic@Cyclo:~ $ reboot ]
```

Abbildung 62: Startup Script und reboot command

Beim erneuten Verbinden sollten alle Services laufen.

```
[lenarcic@Cyclo:~ $ service nginx status
● nginx.service - A high performance web server and a reverse proxy server
  Loaded: loaded (/lib/systemd/system/nginx.service; enabled; vendor preset: enabled)
  Active: active (running) since Sat 2024-03-02 10:17:11 CET; 1h 7min ago
    Docs: man:nginx(8)
  Process: 527 ExecStartPre=/usr/sbin/nginx -t -q -g daemon on; master_process on; (code=
  Process: 610 ExecStart=/usr/sbin/nginx -g daemon on; master_process on; (code=exited,
```

Abbildung 63: nginx Service Status

4.3.1.8. Aufsetzen des Frontend

Das Frontend ist bereits auf dem bereitgestellten Image kompiliert und läuft auf Nginx. Um eine neue Version oder die Website an sich neu aufzusetzen, muss zuerst das Frontend von GitHub heruntergeladen werden.

```
git clone https://github.com/Lemme-lab/Cyclo-Bench-Frontend.git
```

Dann müssen alle Libraries und Plugins im Ordner installiert werden, damit anschließend die Website kompiliert werden kann.

```
cd Cyclo-Bench-Frontend/
npm install
npm run dev build dev
```

Als Letztes muss die kompilierte Website in den Nginx-Ordner verschoben und der Nginx-Service neu gestartet werden.

```
sudo cp -r dist/* /var/www/html/
sudo service nginx restart
```

Nach kurzer Wartezeit ist die Website über die IP-Adresse des Raspberry Pi oder durch den Hostnamen über Port 80 erreichbar.



Abbildung 64: Website Zugriff über Hostname

Abbildung 65: Website Zugriff über IP-Adresse

4.3.1.9. Aufsetzen des Backend

Wie auch das Frontend ist das Backend bereits vorkonfiguriert auf dem bereitgestellten Image. Um das Backend jedoch neu aufzusetzen, muss es zuerst von GitHub geladen werden.

```
git clone https://github.com/Lemme-lab/Cyclo-Bench-Backend.git
```

Dann muss die API nur mehr im Ordner gestartet werden.

```
cd Cyclo-Bench-Backend/
npm start
```

Nach erfolgreichem Start gibt die API verschiedene Systemnachrichten auf der Konsole aus. Zum Testen können HTTP-Anfragen an die IP-Adresse des Raspberry Pi oder an den Hostnamen gesendet werden.

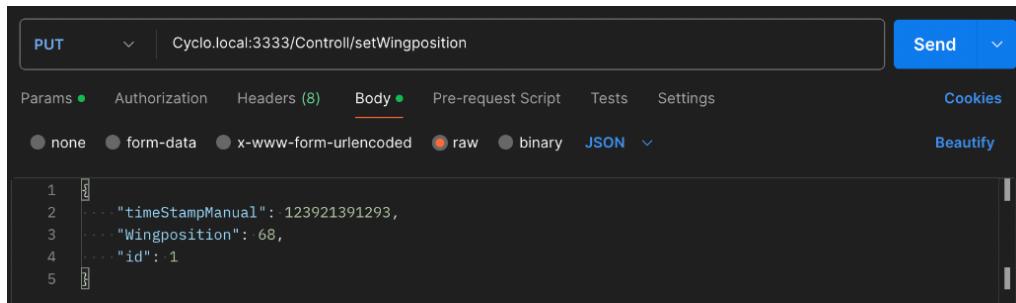


Abbildung 66: Postman PUT Method Output

```
[lenarcic@Cyclo:~/Cyclo-Bench-Backend $ npm start
> backend-cyclo-bench@1.0.0 start
> node index.js

(node:979) [MONGODB DRIVER] Warning: useNewUrlParser is a deprecated option
next major version
(Use `node --trace-warnings ...` to show where the warning was created)
(node:979) [MONGODB DRIVER] Warning: useUnifiedTopology is a deprecated option
next major version
App Connected to Database
MongoDB URL: mongodb://cyclo.local:27017/CycloDB
On Port: 3333
App is listening
Server started at: 02/03/2024, 12:09:52
```

Abbildung 67: Starten der API am Server

4.3.2. Backend – Schnittstelle Design

Die Schnittstelle ist notwendig, um eine Verbindung zwischen Frontend und Datenbank, Messstation und Datenbank sowie eine indirekte Verbindung zwischen Frontend und Messstation zu gewährleisten. Als Schnittstelle wurde eine REST-API ausgewählt, die mit Node.js und Express.js realisiert wird. Der API-Server muss konfiguriert und die Endpoints für das Auslesen und Speichern der Messdaten sowie das Einstellen des Testbench erstellt werden. Außerdem müssen Modelle für die zu verarbeitenden Daten konzipiert und erstellt werden.

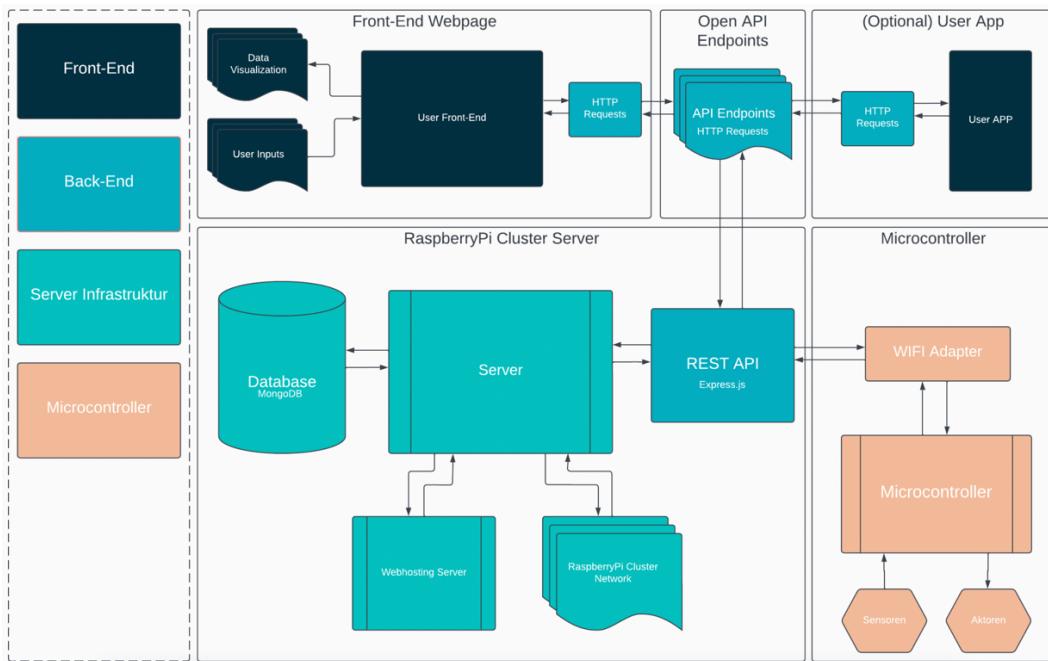


Abbildung 68: Software Block Diagramm

4.3.2.1. Ordner Struktur

Models sind die Datenstrukturen, die über die API verarbeitet werden können. Unter "routes" findet man sowohl die "Sensordata" als auch die "Controldata" Endpoints, auf die unter bestimmten Regeln Daten gesendet und erhalten werden können. In der "config.js" sind verschiedene Konfigurationsvariablen enthalten. Darunter zählen der API-Port und die lokale Datenbankadresse. In "index.js" befindet sich der Code für den API-Server.

```

  \ models
    JS ControllStates.js
    JS SensorData.js
  > node_modules
  \ routes
    JS ControllStateRouter.js
    JS SensorDataRouter.js
  JS config.js
  JS index.js
  {} package-lock.json
  {} package.json
  ⓘ README.md

```

Abbildung 69: Backend Ordner Struktur

4.3.2.2. API-Socket

In index.js ist der API-Socket definiert. Dieser startet nach dem „npm run“ Befehl einen Express.js-Websocket mit der eingestellten Konfiguration, der die definierten Routen/Endpoints sowie verschiedene Debug-Services bereitstellt. Dabei wird zuerst ein Express-Service gestartet. Express ermöglicht das einfache Programmieren einer API, wobei die einzelnen Routes effektiv implementiert werden können. Danach wird CORS aktiviert. CORS steht für „Cross-Origin Resource Sharing“ und verhindert das Aufrufen der Ressourcen durch Unbefugte.

```
const app = express();

app.use(express.json());
app.use(cors());

app.use('/Data', routerSensorData);
app.use('/Controll', routerControll);
```

Dann werden Funktionen erstellt und dem Express-Service hinzugefügt, welche Metadaten beim Starten der API ausgegeben werden.

```
app.use((req, res, next) => {
  console.log(`[${new Date().toLocaleString()}] ${req.method} ${req.url}`);
  next();
});

app.use((err, req, res, next) => {
  console.error(err.stack);
  res.status(500).send('Something broke!');
});
```

Als Letztes wird eine Verbindung mit der Datenbank erstellt, wodurch der API-Socket gestartet wird. Es werden erneut Metadaten ausgegeben und ein Error-Handling für einen besetzten Port sowie für einen Fehler mit der Datenbankverbindung erstellt.

```
mongoose.connect(mongoDBURL, { useNewUrlParser: true, useUnifiedTopology: true })
  .then(() => {
    console.log('App Connected to Database');
    console.log('On Port:', PORT);
    app.listen(PORT, '0.0.0.0', () => {
      console.log("App is listening");
      console.log("Server started at:", new Date().toLocaleString());
    });
  });
});
```

```

}).catch((error) => {
  if (error.code === 'EADDRINUSE') {
    console.log(`Port ${PORT} is already in use`);
    setTimeout(() => {
      startServer();
    }, 1000);
  } else {
    console.error('Error connecting to MongoDB:', error);
  }
});

```

Der Port sowie mongoDBURL Variablen sind unter der config.js definiert.

```

export const PORT = 3333;
export const mongoDBURL = 'mongodb://cyclo.local:27017/CycloDB'

```

4.3.2.3. Models

Die Models sind Strukturen, in denen man die zu verarbeitenden Variablen in einem JSON definiert. Diese Modelle werden dann sowohl beim Senden von Daten als auch zur Speicherung verwendet und dienen auch als Kommunikation zwischen Datenbank und API. Als „Kommunikator“ zwischen Datenbank und API wird Mongoose verwendet. Die Models teilen sich in die Gruppen SensorData und ControlData ein.

Die SensorData Modelle beinhalten die Motorgeschwindigkeit, Rotorgeschwindigkeit, Drehmoment, Schub und Flügelposition. Zuerst wird ein Schema erstellt, welches später zu einem Model umgewandelt wird. Es wird definiert, welchen Namen, Datentyp und ob es benötigt wird. Bei allen SensorData Modellen wurde jeweils ein manueller Timestamp neben dem automatischen hinzugefügt. Auch eine ID zur Sortierung wird verwendet. Die Schemas wurden so entworfen, dass neue Features in Bezug auf Visualisierung, Auswertung und Sortierung einfach implementiert werden können.

```

const rotorSpeed = new mongoose.Schema({
  rotorSpeed: {
    type: Number,
    required: true,
  },
  timeStampManual: {
    type: Number,
    required: true,
  },
},

```

```

  id: {
    type: Number,
    required: true,
  }
}, {
  timestamps: true,
});
  
```

Am Ende werden die einzelnen Schemas zu einem Mongoose-Model umgewandelt und als const. Variable exportiert.

```

export const SetSpeed = mongoose.model('setSpeed', setSpeed);
export const MotorSpeed = mongoose.model('motorSpeed', motorSpeed);
export const RotorSpeed = mongoose.model('rotorSpeed', rotorSpeed);
export const Torque = mongoose.model('torque', torque);
  
```

Das ControlData beinhaltet nur ein großes Model mit den verschiedenen Control Parameters in einem zusammengefügten. Dazu zählt eine ID, welche immer 1 ist, damit nur ein Objekt abgefragt werden kann. Ob der Motor läuft, in welche Richtung der Rotor Schub leisten soll, eine maximale Motorgeschwindigkeit, welche die derzeitig eingestellte begrenzt, die derzeitige Flügelposition sowie Testroutinen und gespeicherte Einstellungen. Auch ob es eine Verbindung zum Testbench gibt, ob der Schutzkäfig geöffnet ist oder ob der Emergency Shutdown betätigt wurde, wird in diesem Control Parameter Model gespeichert.

```

const controlStates = new mongoose.Schema({
  id: {
    type: Number,
    required: true,
  },
  motorRun: {
    type: Boolean,
    required: true,
  },
  direction: {
    type: [Number],
    default: [0, 0, 0],
    required: true,
  },
  ...
  
```

4.3.2.4. Endpoints / Routes

Auf den Endpoints, auch Routen genannt, können HTTP-Anfragen gemacht werden. Auf diesen werden dann unter bestimmten Regeln die Daten entweder angenommen und gespeichert, abgelehnt oder auch die Abfrage von Daten wird angenommen oder abgelehnt.

Die Routen sind unter der Adresse des API-Sockets + der Unterteilung + der Adresse des Endpoints erreichbar. Die Routen sind in diesem Projekt noch einmal in /data und /Control unterteilt. Außerdem können die Routen noch in GET, POST und PUT Methods eingeteilt werden. GET Methods werden verwendet, um Daten von der Datenbank abzufragen. POST Methods sind zum Senden und PUT Methods zum Aktualisieren von Daten auf der Datenbank zuständig.

Ein Endpoint hat im Methodenkopf eine Adresse und eine der oben genannten HTTP-Methods und wird asynchron verarbeitet.

```
routerSensorData.get('/getSensorData/rotorSpeeds', async (request, response)  
=> {...})
```

4.3.2.5. Get Endpoint

Bei einem Get-Endpoint werden die letzten 100 Daten von der Datenbank gesucht und in die Response gespeichert. Dabei werden die Daten so sortiert, dass die letzten Daten als letztes gesendet und damit die ersten Elemente sind. Außerdem werden Debug Nachrichten ausgegeben und ein Error Handling implementiert.

```
try {  
    console.log("Retrieving Rotor Speeds data...");  
    const count = parseInt(request.query.count) || 100;  
    const rotorSpeedData = await RotorSpeed.find().sort({ _id: -1  
}).limit(count);  
    return response.status(200).send(rotorSpeedData);  
} catch (error) {  
    console.log("Error encountered while retrieving Rotor Speeds data: " +  
error.message);  
    response.status(500).send({  
        message: 'Internal server error: ' + error.message  
    });  
}
```

4.3.2.6. Post Endpoint

Bei einem POST-Endpoint wird zuerst überprüft, ob alle gesendeten Daten im richtigen Format vorliegen. Wenn nicht, wird ein Fehler ausgegeben. Wenn alle gesendeten Daten im richtigen Format auf der API ankommen, werden sie in ein Objekt des entsprechenden Models umgewandelt, welches dann in der Datenbank gespeichert wird.

Hierbei wird erkannt, ob das Drehmoment gesendet wurde.

```
if (!request.body.torque) {  
    return response.status(400).send({  
        message: 'Please provide the required field: "torque"',  
    });  
}
```

Dann wird erkannt, ob der gesendete Wert eine Ganzzahl (Integer) ist. Das Gleiche wird für die anderen Felder in diesem Endpoint durchgeführt.

```
if (!Number.isInteger(request.body.torque)) {  
    return response.status(400).send({  
        message: 'Invalid data type for "torque". Must be integer.',  
    });  
}
```

Als Letztes wird das Objekt erstellt und in die Datenbank gespeichert. Bei einem Fehler wird ein Fehlercode als Antwort gesendet.

```
const torque = {  
    torque: request.body.torque,  
    timeStampManual: request.body.timeStampManual,  
    id: request.body.id,  
};  
  
const newTorque = await Torque.create(torque);  
  
return response.status(201).send(newTorque);  
} catch (error) {  
    console.log("Error encountered while setting Torque data: " + error.message);  
    response.status(500).send({  
        message: 'Internal server error: ' + error.message  
    });  
}
```

4.3.2.7. PUT Endpoint:

PUT-Methoden werden in diesem Projekt eingesetzt, um die gespeicherten Controls zu ändern. Da wir nur ein Steuerungsobjekt in der Datenbank speichern, ist es am einfachsten, nur eine bestimmte Variable zu ändern.

Zuerst wird wieder der Boolean-Wert für den Motor aus der PUT-Anfrage geholt. Dann wird erneut erkannt, ob die Variable nicht leer ist.

```
const { boolMotor } = request.body;

console.log("Motor State: " + boolMotor);

if (boolMotor === undefined) {
  return response.status(400).send({
    message: 'Send Motor bool',
  });
}
```

Dann wird nach dem gespeicherten Control Objekt gesucht, und das Feld "motorRun" mit der Variable überschrieben. Das Keyword „New“ gibt dann das neu gespeicherte Objekt zurück.

```
const result = await ControllStates.findOneAndUpdate(
  { id: 1 },
  { $set: { motorRun: boolMotor } },
  { new: true }
);
```

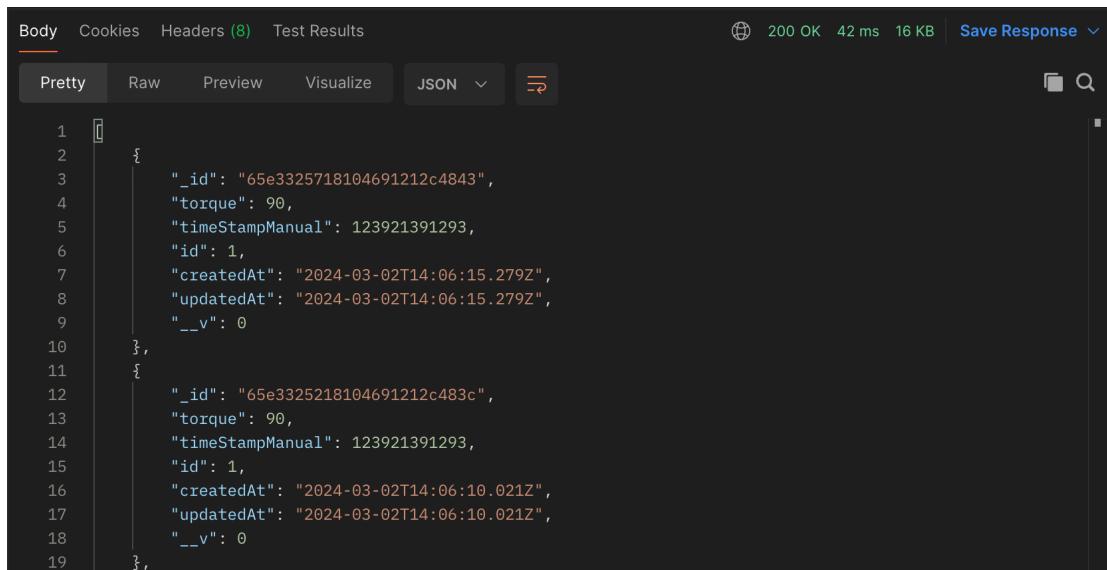
Am Ende gibt es wieder Debug Messages und ein Error Handling.

```
if (!result) {
  return response.status(404).json({
    message: 'Something went horribly wrong'
});
}

return response.status(200).send({
  message: 'Set Motor State successfully',
  updatedDocument: result
});
} catch (error) {
  console.log(error.message);
  response.status(500).send({
    message: error.message
});
}
```

4.3.2.8. Testen der API

Die API kann man durch Senden von HTTP-Anfragen testen und validieren. Durch eine Software wie Postman ist dieser Prozess auch viel einfacher. Dabei muss man die richtige Adresse, die richtige Anfragemethode und den richtigen Anfragekörper senden.



```

1 [
2   {
3     "_id": "65e3325718104691212c4843",
4     "torque": 90,
5     "timeStampManual": 123921391293,
6     "id": 1,
7     "createdAt": "2024-03-02T14:06:15.279Z",
8     "updatedAt": "2024-03-02T14:06:15.279Z",
9     "__v": 0
10   },
11   {
12     "_id": "65e3325218104691212c483c",
13     "torque": 90,
14     "timeStampManual": 123921391293,
15     "id": 1,
16     "createdAt": "2024-03-02T14:06:10.021Z",
17     "updatedAt": "2024-03-02T14:06:10.021Z",
18     "__v": 0
19   }
]
  
```

Abbildung 70: Postman Get Method Output

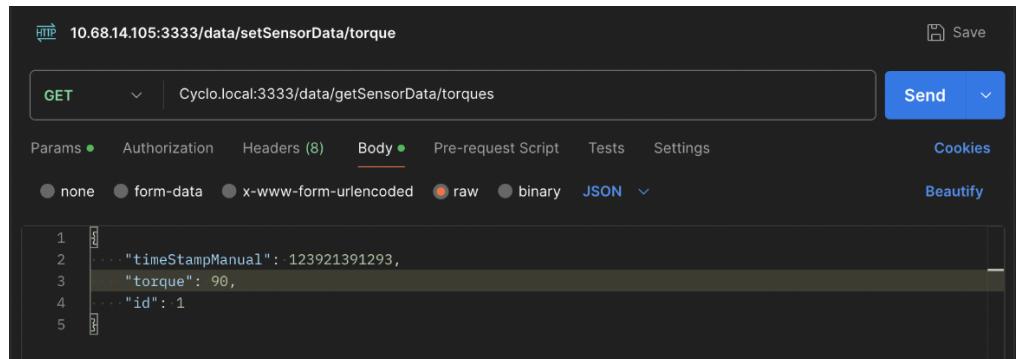


Abbildung 71: Postman Get Method Input Body

4.3.3. Frontend – Visualisierung

Das Ziel der Diplomarbeit ist die Analyse der am Teststand gemessenen Daten. Daher wird eine Oberfläche benötigt, auf der diese Daten angezeigt werden können. Der Teststand soll außerdem über diese Oberfläche kontrolliert und eingestellt werden können. Die gemessenen Daten wie Rotorgeschwindigkeit und Motorgeschwindigkeit sollen graphisch ansprechend dargestellt und einfach miteinander verglichen werden können. Die verschiedenen Parameter, Voreinstellungen und Testroutinen sollen auf der Website intuitiv einstellbar sein. Das Frontend wurde mit dem Framework React + Vite realisiert, wobei auch Material-UI-Komponenten verwendet wurden.



Abbildung 72: Front-End Website

4.3.3.1. Frontend Dashboard Panel

Das Frontend-Dashboard-Panel bietet eine übersichtliche Darstellung wichtiger Messdaten und Informationen in Echtzeit. Auf dem Dashboard-Panel werden die Messdaten visuell ansprechend angezeigt. Es besteht aus verschiedenen Chart-Typen, die unter anderem die verschiedenen Messdaten sowie wichtige Informationen auf einen Blick liefern. Das Dashboard ermöglicht es dem Nutzer, die Leistung sowie verschiedene Einstellungen des Rotors zu überwachen und zu analysieren. Auf dem Dashboard befinden sich verschiedene Chart-Typen, welche eine Vielzahl an Informationen auf einen Blick liefern.

4.3.3.2. Chart Typen

Die Charts zeigen verschiedene Daten an und ermöglichen es dabei, diese auch mit anderen direkt zu vergleichen. Messdaten wie Drehmoment und Schub, Rotorgeschwindigkeit und Schub oder Motor- und Rotorgeschwindigkeit können direkt miteinander verglichen werden. Dabei werden Linien- und Flächendiagramme, Balkendiagramme sowie Kreisdiagramme eingesetzt, um in Echtzeit Erkenntnisse über die Integrität, den Zustand und die Leistung des Rotors zu erhalten. Durch diese Kombination können Benutzer die Daten leicht interpretieren und dadurch Muster und wichtige Erkenntnisse aus den Messdaten ziehen.

4.3.3.3. Line Chart

Line-Charts werden auf dem Dashboard eingesetzt, um Daten über die Zeit im Vergleich zu anderen darzustellen. Dabei werden Schub und Rotor-Drehzahl, Schub und Flügelposition sowie Kräfte über die Zeit dargestellt. Der Schub- und Rotor-Drehzahl-Chart zeigt die Beziehung zwischen dem Schub, den der Rotor produziert, und der eingestellten Drehzahl. Der Schub- und Flügel-Positions-Chart veranschaulicht die Beziehung zwischen der erzeugten Schubkraft und der Position der Flügel. Es ermöglicht, den unterschiedlichen Schub in Abhängigkeit von der Position der Flügel zu betrachten. Dadurch kann nicht nur veranschaulicht werden, in welche Richtung, sondern auch mit welcher Effizienz der Rotor mit unterschiedlich eingestellten Flügelpositionen funktioniert.

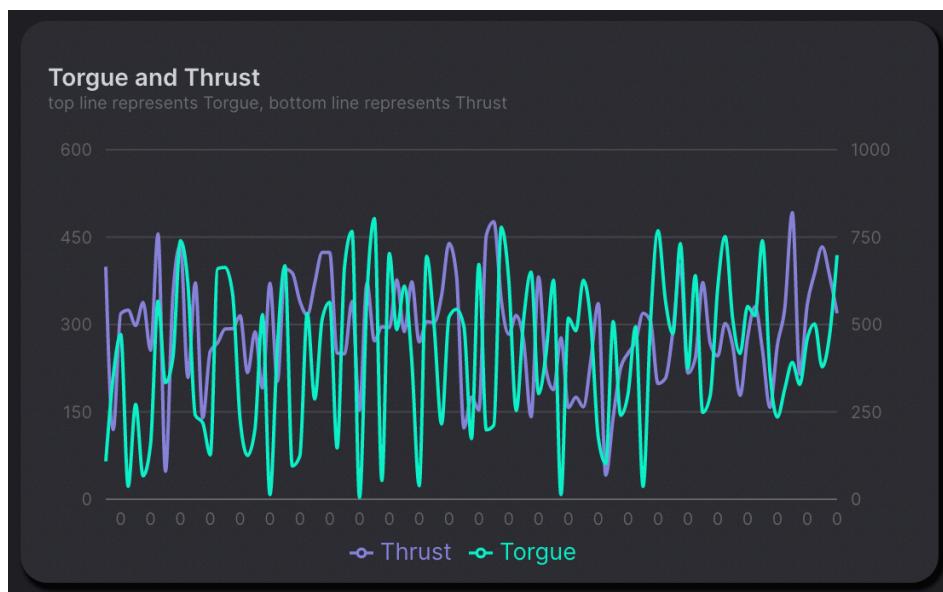


Abbildung 73: Torque and Thrust Linechart

Zuerst wird ein Liniendiagramm definiert, konfiguriert und das CSS-Styling festgelegt.

```
<LineChart
    width={500}
    height={400}
    data={combinedData2}
    margin={{{
        top: 20,
        right: 0,
        bottom: 55
    }}}
    >
</LineChart>
```

In diesem Linechart werden dann zweimal die X- und Y-Achsen für die dargestellten Messdaten definiert. Dabei wird festgelegt, welche Parameter der Messdaten auf den Achsen dargestellt werden sollen, und es wird die Schriftart für die Achsenbeschriftungen eingestellt.

```
<XAxis
    dataKey="id"
    axisLine={false}
    tickLine={false}
    style={{ fontSize: "10px" }}
/>
<YAxis
    axisLine={false}
    tickLine={false}
    style={{ fontSize: "10px" }}
/>
/>>
```

Dann wird die Linie definiert und eingestellt, welcher Messdatenparameter angezeigt werden soll. Es wird auch definiert, auf welcher Seite die Y-Achse angezeigt wird, welche Farbe und Breite die Linie hat, welche Breite sie hat und ob es eine Animation beim Laden geben soll.

```
<Line
    yAxisId="left"
    type="monotone"
    strokeWidth={2}
    dataKey="Thrust"
    stroke={palette.tertiary[500]}
    isAnimationActive={false}
    dot={false}
/>>
```

Zuletzt wird noch eine Legende, welche sich unter dem Chart befindet, sowie ein Tooltip welcher es ermöglicht mit der Maus den genauen Wert abzufragen, erstellt und konfiguriert.

```

<Tooltip />
<Legend
  height={20}
  wrapperStyle={{
    margin: "0 0 10px 0"
  }}
/>

```

4.3.3.4. Pie-Chart

In den Kreisdiagrammen werden die aktuellen X-, Y- und Z-Kräfte des Rotors in Echtzeit dargestellt, wobei jeweils nur der letzte Zustand angezeigt wird. Diese Darstellung ermöglicht es, schnell zu erkennen, in welche Richtung der Rotor derzeit Schub ausübt. Jedes Kreisdiagrammelement repräsentiert dabei eine Kraftkomponente. Die Größe des Kreisausschnitts entspricht dem relativen Anteil dieser Kraft im Vergleich zur Gesamtkraft.

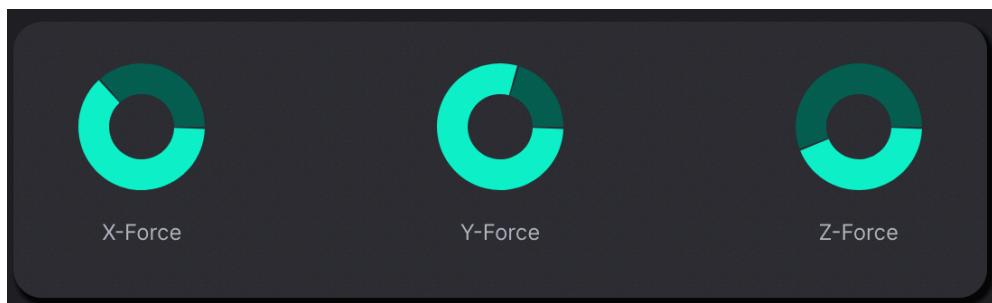


Abbildung 74: Kräfte Pie Chart

Als erstes wird der PieChart genau wie beim Linechart definiert und konfiguriert. Es wird jedoch schon in dem Box Container definiert das nur der letzte Wert angezeigt werden soll. Ein Großteil des Codes ist für das Visuelle zuständig.

```

<Box key={`${data[0].name}-${i}`}>
  <PieChart width={110} height={100}>
    <Pie
      stroke="none"
      data={data}
      innerRadius={18}
      outerRadius={35}
      paddingAngle={2}
      dataKey="value"
    >

```

```

  {data.map((entry, index) => (
    <Cell key={`cell-${index}`} fill={pieColors[index]} />
  )))
  </Pie>
</PieChart>
<Typography variant="h5">{data[0].name}</Typography>
</Box>
  
```

4.3.3.5. Bar-Chart

Im Bar-Chart werden die zu erreichenden Geschwindigkeiten der Testroutine angezeigt. Jeder Balken im Diagramm repräsentiert einen Testschritt, wobei die Höhe des Balkens die maximale Geschwindigkeit in diesem Schritt angibt. Dadurch erhält der Benutzer Einblicke in die Testroutine und kann sich auf die bevorstehenden Testschritte einstellen.



Abbildung 75: Testroutine Bar Chart

Der Bar-Chart wird genau wie die anderen Diagramme definiert und konfiguriert.

```

<BarChart
  width={500}
  height={300}
  data={realDataTestRoutine}
  margin={{
    top: 17, ...
  }}
>
<Bar dataKey="rotorSpeed" fill="url(#colorRevenue)" />
</BarChart>
  
```

4.3.3.6. Display-Grid

Die Diagramme sind in einem Grid angeordnet. Dadurch können die Diagramme organisiert und leicht modifiziert werden. Außerdem ist es sehr einfach, das Grid für verschiedene Bildschirmgrößen anzupassen.



Abbildung 76: Front-End Charts

Zuerst wird der Gridaufbau definiert. Dabei entspricht jeder Buchstabe einer Box oder in unserem Fall einem Diagramm.

```
const gridTemplateLargeScreens = `

  "a b j"
  "a b j"
  "a b j"
  "a b j"
  "d e c"
  "d e c"
  "d e c"
  "d h c"
  "d h c"
`
```

Danach wird die Dashboard-Komponente erstellt und eine Variable gesetzt, die bestimmt, welches Grid verwendet werden soll. Anschließend wird das CSS-Styling definiert und das Grid gestaltet. Als nächstes werden die drei Reihenkomponenten, welche die Diagramme beinhalten, eingefügt.

```
const isAboveMediumScreens = useMediaQuery("(min-width: 1000px)");
return (
  <Box
    width="100%"
    height="100%"
    display="grid"
    gap="1.5rem"
    sx={
      isAboveMediumScreens
      ?
      {
        gridTemplateColumns: "repeat(3, minmax(370px, 1fr))",
        gridTemplateRows: "repeat(10, minmax(60px, 1fr))",
        gridTemplateAreas: gridTemplateLargeScreens,
      }
      :
      {
        gridAutoColumns: "1fr",
        gridAutoRows: "80px",
        gridTemplateAreas: gridTemplateSmallScreens,
      }
    }
  >
  <Row1 />
  <Row2 />
  <Row3 />
</Box>
);
};
```

Abschließend wird jeweils eine Box um die Charts mit einem Identifier hinzugefügt, der mit Hilfe des zuvor festgelegten Rasters bestimmt, an welche Position welches Diagramm kommt.

```
<DashboardBox gridArea="a">
  <ResponsiveContainer width="100%" height="100%">
    <LineChart
    </LineChart>
  </ResponsiveContainer>
</DashboardBox>
```

4.3.3.7. API-Einbindung

Die API-Einbindung wurde größtenteils mithilfe von ReduxJS Toolkit realisiert. Dieses ermöglicht eine sehr einfache, saubere, effiziente und skalierbare Verarbeitung von HTTP-Requests. Alle HTTP-Methoden außer PUT wurden mit ReduxJS realisiert. Die PUT-Methoden führten bei der Implementierung mit ReduxJS zu Performanceproblemen. Zuerst wird ein API-Objekt erstellt, in dem die Endpoints definiert sind.

```
export const api = createApi({
  baseQuery: fetchBaseQuery({ baseUrl: import.meta.env.VITE_BASE_URL }),
  reducerPath: "main",
  tagTypes: ["torque", "thrust", "rotorSpeed", "motorSpeed", ...],
  endpoints: (build) => ({
    },
  });
});
```

Für jeden API-Endpoint wurde eine eigene Query und dazu passendes Objekt erstellt. Danach kann die API-Abfrage sehr einfach im Code implementiert und oft wiederverwendet werden.

```
export interface TorqueType {
  torque: number;
  timeStampManual: number;
  id: number;
}

getTorque: build.query<Array<TorqueType>, void>({
  query: () => "data/getSensorData/torques",
  providesTags: ["torque"],
});
```

Am Ende werden alle Querys unter dem API-Objekt exportiert.

```
export const { useGetTorqueQuery,
  useGetThrustQuery,
  useGetControlDataQuery
} = api;
```

4.3.3.8. Frontend Control Panel

Das Control Panel dient zur Steuerung und Überwachung des Teststandes. Dabei können verschiedene Parameter wie die Rotorgeschwindigkeit sowie die Flügelposition über Schieberegler eingestellt, der Rotor gesteuert und überwacht sowie Testroutinen erstellt, gespeichert und gestartet werden. Die konfigurierten Parameter werden in der Datenbank gespeichert, sobald sie eingestellt sind, während die zu überwachenden Parameter von der Website abgerufen werden.

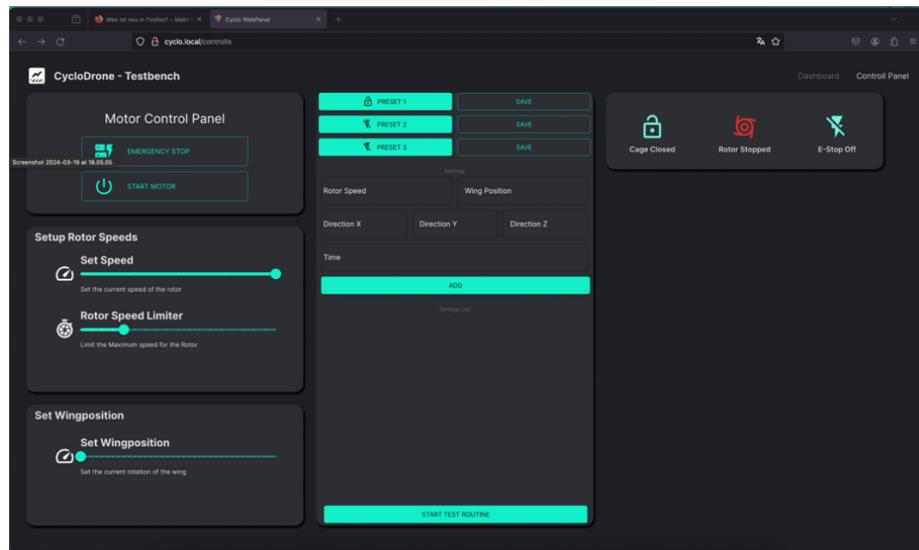


Abbildung 77: Front-End Control Panel

4.3.3.9. Control Panel Slider und Button.

Die Rotorgeschwindigkeit und die Flügelposition können jeweils über einen Schieberegler eingestellt werden. Der Rotor kann über einen Button gestartet und gestoppt werden, wodurch beim Stoppen noch der letzte Schritt in der Testroutine ausgeführt wird. Beim Drücken des Emergency Stops wird der Rotor, auch während einer Testroutine, sofort gestoppt. Beide Elemente wurden mithilfe von Material-UI-Komponenten umgesetzt. Diese Komponenten sind bereits vorgefertigt und können einfach in die Website eingebunden und mit wenigen Einstellungen an das Website-Design angepasst werden.

Der Button kann mithilfe von MUI sehr einfach implementiert und angepasst werden. Dafür muss MUI nur installiert und dann importiert werden.

```
npm install @mui/material @emotion/react @emotion/styled
import { Slider, Button } from '@mui/material';
```

Dann kann die Material-UI Button-Komponente verwendet werden. Dabei wird, ähnlich wie bei den Standard-JavaScript-Buttons, eine Callback-Methode verwendet, um den Zustand zu setzen.

```
<Button
  size="large"
  variant="outlined"
  style={{
    width: '60%',
    marginBottom: '10px',
    display: 'flex',
    alignItems: 'center',
    justifyContent: 'flex-start',
  }}
  onClick={ startStopMotor }
>
<DynamicFormIcon sx={{ mr: 3, fontSize: 40 }} />
  Start Motor
</Button>
```

Die Callback-Methode, die durch den Button ausgelöst wird, macht eine PUT API Request, welche den Motor Status auf der Datenbank updatet.

```
const startStopMotor = async () => {
  try {
    const response = await fetch(baseUrl + '/Controll/startMotor', {
      method: 'PUT',
      headers: {
        'Content-Type': 'application/json',
      },
      body: JSON.stringify({
        boolMotor: !motorStatus,
      }),
    });
    if (response.ok) {
      setMotorStatus(!motorStatus);
      console.log('Motor status toggled successfully!');
    } else {
      console.error('Failed to toggle motor status');
    }
  } catch (error) {
    console.error('Error toggling motor status:', error);
  }
};
```

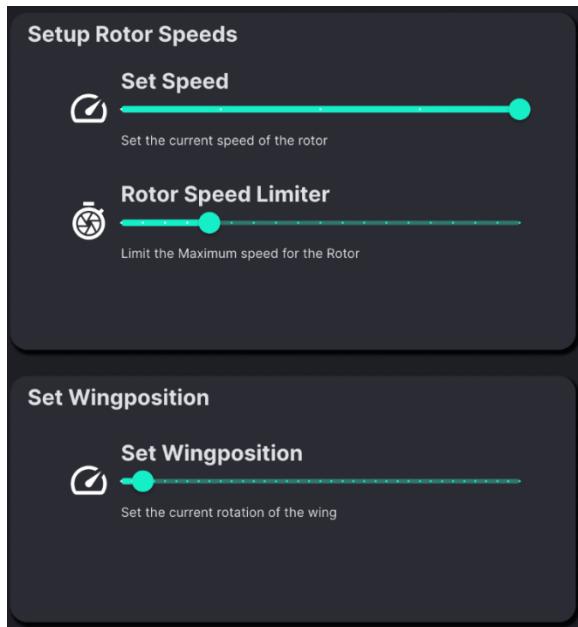


Abbildung 79: Rotor Geschwindigkeit und Flügelposition Slider

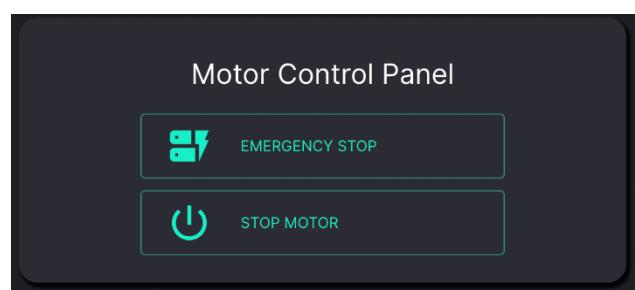


Abbildung 78: Motor Control Buttons

4.3.3.10. Test-Routine Konfigurator

Dieser Konfigurator ermöglicht die Erstellung von Testroutinen, bei denen verschiedene Parameter am Rotor über eine bestimmte Anzahl von Testschritten eingestellt werden können. Eine Routine besteht immer aus mehreren Testschritten. Die erstellten Einstellungen werden dann nacheinander beim Starten der Testroutine angewendet und durchgeführt. Parameter wie Geschwindigkeit, Position der Flügel und die Zeit, die der Rotor mit diesen Parametern verbringen soll, können angepasst werden. Die Routinen können auch als Presets gespeichert werden, um verschiedene mechanische Varianten des Rotors zu validieren und zu vergleichen. Für die Buttons und Textfelder wurden Material-UI-Komponenten verwendet.

Die Preset-Buttons lösen einen Button-Handler aus, der dann einen API-GET-Request macht, um die Presets auf der Datenbank zu beschaffen.

```

<Button
  variant="contained"
  color="primary"
  startIcon={<LockOpenIcon />}
  onClick={handlePreset1Click}
  style={{ flex: 1, marginRight: "5px", marginLeft: "5px" }}
>
  Preset 1
</Button>
  
```

Über die Preset-Nummer, die zwischen 1 und 3 für die drei Presets liegt, kann dann das Preset aus der Datenbank abgefragt und mithilfe eines useState-Hooks in eine Variable gespeichert werden.

```
const [presets, setPresets] = useState([]);

const fetchPresets = async (presetNumber) => {
  try {
    const response = await fetch(baseUrl + `/Controll/getAllPresets`);
    const data = await response.json();
    const selectedPreset = data.presets.find((preset) => preset.id === presetNumber);

    if (selectedPreset) {
      setPresets(selectedPreset.routine);
    }
  } catch (error) {
    console.error("Error fetching presets:", error);
  }
};
```

Bei Eingabe in den Textfeldern wird eine onChange-Methode aufgerufen, welche dann den eingegebenen Wert in die passende Variable mit dem übergebenen Namen mithilfe eines useState-Hooks speichert. Dabei wird die gleiche Methode für alle Textfelder verwendet, und nur der in der Methode übergebene Name entscheidet darüber, in welche Variable des useState der Wert gespeichert wird.

```
<TextField
  label="Wing Position"
  variant="outlined"
  color="primary"
  value={settings.wingPosition}
  onChange={handleInputChange("wingPosition")}
/>

const [settings, setSettings] = useState({
  rotorSpeed: "",
  wingPosition: "",
  directionMatrix: ["", "", ""],
  time: "",
});
```

```

const handleInputChange = (fieldName, index) => (event) => {
  if (fieldName === "directionMatrix") {
    const updatedDirectionMatrix = [...settings.directionMatrix];
    updatedDirectionMatrix[index] = event.target.value;
    setSettings((prevSettings) => ({
      ...prevSettings,
      directionMatrix: updatedDirectionMatrix,
    }));
  } else {
    setSettings((prevSettings) => ({
      ...prevSettings,
      [fieldName]: event.target.value,
    }));
  }
};
  
```

Über den „Add“ - Button kann dann ein neuer Testschritt erstellt und zu den schon bestehenden Schritten hinzugefügt werden.

```

<Button
  variant="contained"
  color="primary"
  onClick={handleAddButtonClick}
>
  Add
</Button>
  
```

Es wird das useState-Settings-Objekt an die Presets-Liste angehängt und erneut mithilfe eines useState-Hooks gespeichert. Dann werden die Einstellungsfelder wieder geleert.

```

const handleAddButtonClick = () => {
  const updatedPresets = [...presets, { ...settings }];
  setPresets(updatedPresets);
  setSettings({
    rotorSpeed: "",
    wingPosition: "",
    directionMatrix: ["", "", ""],
    time: "",
  });
};
  
```

Über den "Save"-Button können alle derzeitig eingestellten Testschritte als Preset gespeichert werden. Dabei werden die einzelnen Daten der Routine zuerst von Strings in Integers und Floats konvertiert und dann mit einem PUT-API-Request an die Datenbank geschickt.

```

const handleSaveClick = (presetNumber) => {
  const routineData = presets.map((preset) => ({
    rotorSpeed: parseInt(preset.rotorSpeed),
    wingPosition: parseInt(preset.wingPosition),
    directionMatrix: [
      parseFloat(preset.directionMatrix[0]),
      parseFloat(preset.directionMatrix[1]),
      parseFloat(preset.directionMatrix[2]),
    ],
    time: parseInt(preset.time),
  }));
}

fetch(baseUrl + `/Controll/addRoutine`, {
  method: 'PUT',
  headers: {
    'Content-Type': 'application/json',
  },
  body: JSON.stringify({
    id: presetNumber,
    routine: routineData,
  }),
})
  .then((response) => response.json());
};
  
```

Beim Drücken des "Start Test Routine"-Buttons werden wieder die Daten von Strings zu Floats und Ints konvertiert. Zuerst wird die Testroutine über einen PUT-API-Request gespeichert, und anschließend wird der Rotor auf dieselbe Weise gestartet.

```

const handleStartTestClick = async() => {
  const testRoutineData = {
    TestRoutine: presets.map((preset) => ({
      rotorSpeed: parseInt(preset.rotorSpeed),
      wingPosition: parseInt(preset.wingPosition),
      directionMatrix: preset.directionMatrix.map(parseFloat),
      time: parseInt(preset.time),
    })),
  };
}

fetch(baseUrl + "/Controll/setTestRoutine", {
  method: 'PUT',
  headers: {
    'Content-Type': 'application/json',
  },
  body: JSON.stringify(testRoutineData),
})
  .then((response) => response.json());
  
```

```

try {
  const response = await fetch(baseUrl + '/Controll/startMotor', {
    method: 'PUT',
    headers: {
      'Content-Type': 'application/json',
    },
    body: JSON.stringify({
      boolMotor: true,
    }),
  });
} catch (error) {

}
};


```

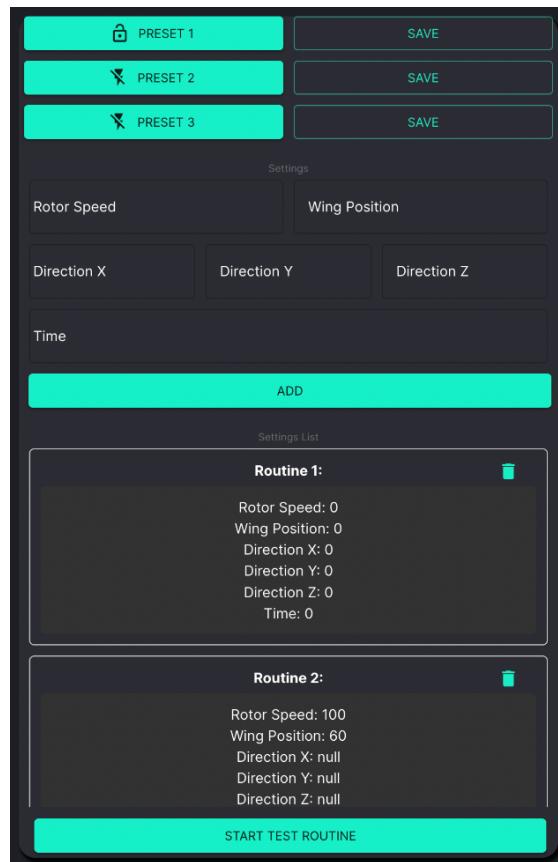


Abbildung 80: Test-Routinen Konfigurator

4.3.3.11. Status-Anzeigen

Die Statusanzeigen zeigen verschiedene Parameter an, darunter ob der Käfig geöffnet ist, ob der Rotor läuft oder ob der E-Stop aktiviert wurde. Diese Informationen werden periodisch von der Datenbank über die API abgefragt. Anschließend wird anhand der Parameter die Farbe sowie der Text unter den Icons geändert.

```

<LockOpenIcon
  sx={{ color: getIconColor2(controlData?.parameters?.cageOpen) }}
/>
<Typography marginTop={1} marginLeft={0} variant="h4">
  Cage {controlData?.parameters?.cageOpen ? 'Open' : 'Closed'}
</Typography>

```

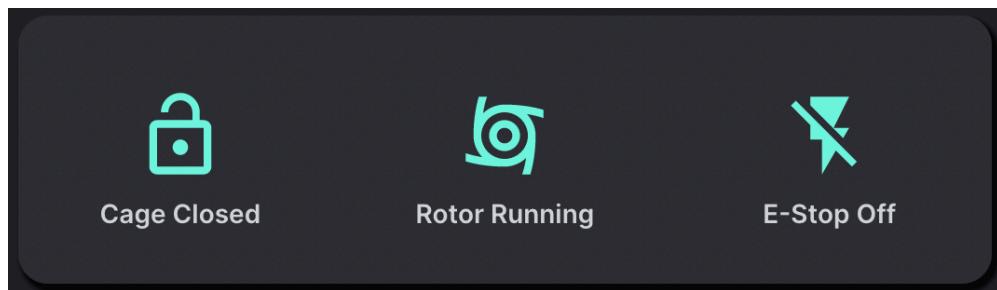


Abbildung 81: Teststand Status Anzeigen

Über einen useEffect-Hook wird ein Intervall eingerichtet, das zu Beginn und dann jede Sekunde Daten von der API abruft und in eine Variable speichert. Anschließend wird das Intervall ordnungsgemäß zurückgesetzt und beendet.

```

const {
  data: controlData,
  error: controlError,
  refetch: refetchControl,
} = useGetControlDataQuery();

useEffect(() => {
  const id = setInterval(() => {
    refetchControl();
  }, 1000);
  setIntervalId(id);

  return () => clearInterval(id);
}, [refetchControl]);

```

4.4. Ergebnisse / Konklusion

Die Datenverarbeitung und Visualisierung funktionierten größtenteils wie am Anfang geplant. Daten werden in Echtzeit über die API gespeichert und am Frontend angezeigt. Die Datenvizualisierung ist äußerst effektiv. Trotz der begrenzten Live-Daten, die am Rotor gemessen wurden, lässt sich bereits erkennen, dass die Analyse erfolgreich sein wird.

Die Kontrolle des Teststandes über das Frontend funktioniert auch einwandfrei. Der Rotor kann gestartet und die Flügelposition sowie die Geschwindigkeit eingestellt werden. Dabei funktioniert die Kette vom User Input bis zum tatsächlichen Einstellen am Rotor schneller als erwartet. Beim Testen traten keine Engpässe auf, weder beim Senden großer Datenmengen an die API noch beim Abfragen von Daten. Der konfigurierte "Server" Raspberry kann schnell aufgesetzt werden und startet automatisch alle Dienste, die zum Betreiben des Teststandes notwendig sind. Dadurch kann schon nach kurzer Wartezeit die Website in Betrieb genommen, Daten an das Frontend gesendet und der Rotor gesteuert werden. Features wie Testroutinen, Überwachung des Rotors sowie die Skalierbarkeit und einfache Implementierung der Funktionskette in anderen Projekten sind schon jetzt im Code bereitgestellt, jedoch noch nicht am Rotor selbst implementiert.

Das Implementieren unserer Softwarekette sowie die Schnittstelle zur Datenspeicherung können sehr einfach in anderen Projekten weiterverwendet, weiterentwickelt und adaptiert werden. Dadurch kann auch unser Wirtschaftspartner den Rotor einfach in zukünftigen Arbeiten weiterentwickeln. Wir konnten verschiedene Technologien einsetzen, ohne an einen Dienstanbieter gebunden zu sein, da wir unsere gesamte Infrastruktur selbst entwickelt, betrieben und skaliert haben. Zudem setzen wir stark auf Open-Source-Services, die für uns kostenlos sind.

Es ist noch offen, das Softwaresystem auf vier Rotoren zu skalieren, weitere Control Parameter für den Drohnenbetrieb sowie weitere Zwischenmodule zum ordentlichen Betrieb hinzuzufügen. Obwohl der Server auf einem Raspberry Pi-Cluster betrieben werden kann, um mehr Leistung und Speicher zu erreichen, empfiehlt es sich, bei der Skalierung auf den Drohnenbetrieb einen leistungsstärkeren Server zu verwenden. Die Dienste, die auf dem Raspberry Pi genutzt wurden, können jedoch trotzdem weiterverwendet werden.

6.7. Wettbewerbe

Im Lauf des Erstellens der Diplomarbeit wurde an einigen für das Projekt relevanten Wettbewerben teilgenommen.

Jugend Innovativ

Beworben am 29.11.2023

- Erfolgreiche Pre-Checks
- Halbfinale
- Projektbonus in Höhe von 300€

Ideenwettbewerb innovation@school

Beworben am 11.12.2023

- erfolgreiche Einreichung
- Aufstieg in die 10 geförderten Projekte
- 2000€ Projektförderung

Bosch Innovationspreis 2024

Registriert am 17.01.2024

Abgabe bzw. Einreichung der Diplomarbeit bis zum 11.04.2024

7. Literaturverzeichnis

- [1] CycloTech R.evolution of motion, „cyclotech.at,“ CycloTech R.evolution of motion, 2024. [Online]. Available: <https://www.cyclotech.at/rotor/>. [Zugriff am 22. Mai 2024].
- [2] Futurezone GmbH, „futurezone.at,“ Futurezone GmbH, 2024. [Online]. Available: <https://futurezone.at/produkte/legendaerer-schiffsantrieb-voith-schneider-propeller-elektrifiziert-evsp/402213588>. [Zugriff am 22. Mai 2024].
- [3] H. Schenk, „Fortschrittsgrad&Co,“ Oktober 2007. [Online]. Available: <http://www.drivecalc.de/PropCalc/PCHelp/Fortschrittsgrad&Co.pdf>. [Zugriff am 04. Juli 2023].
- [4] Deutsches Zentrum für Luft- und Raumfahrt (DLR), „DLR_next,“ DLR, 13. Dezember 2021. [Online]. Available: https://www.dlr.de/next/desktopdefault.aspx/tabcid-6621/10878_read-24681/. [Zugriff am 04. Juli 2023].
- [5] Magnetic Sense GmbH, „Der Kraftsensor - Magnetic Sense Solutions,“ Magenetic Sense GmbH, 2019. [Online]. Available: <https://magnetic-sense.com/der-kraftsensor-grosser-effekt-mit-vielen-messmethoden/>. [Zugriff am 04. Juli 2023].
- [6] ME-Messsysteme, „K6D-Mehrkomponenten-Sensoren,“ 2023. [Online]. Available: <https://www.me-systeme.de/de/loesungen/messgroesse/k6d-info>. [Zugriff am 24. Juli 2023].
- [7] induux international gmbh, „Drehmomentmessung,“ induux international gmbh, 19. Juli 2022. [Online]. Available: <https://wiki.induux.de/Drehmomentmessung>. [Zugriff am 04. Juli 2023].
- [8] S. Kunze, „Echte berührungslose Drehmomentmessung,“ 24. März 2014. [Online]. Available: <https://www.elektrotechnik.vogel.de/echte-beruehrungslose-drehmomentmessung-a-439250/>. [Zugriff am 24. Juli 2023].
- [9] tr-electronic.de, „Magnetostriktion,“ [Online]. Available: <https://www.tr-electronic.de/produkte/lineargeber/magnetostriktion/funktionsbeschreibung>. [Zugriff am 24. Juli 2023].
- [10] MAGTROL, „Drehmomentaufnehmer,“ [Online]. Available: <https://www.magtrol.com/germany/product/torque-transducers/>. [Zugriff am 24. Juli 2023].

- [11] ATTESTEO GmbH & Co. KG, „Drehmomentmessflansche für große Dimensionen von ATTESTEO,“ 11. Dezember 2018. [Online]. Available: <https://www.atesteo.com/drehmomentmessflansche-fuer-grosse-dimensionen-von-atesteo/>. [Zugriff am 24. Juli 2023].
- [12] S. Lasogga, „TU Berlin,“ 2015. [Online]. Available: https://service.projektlabor.tu-berlin.de/wordpress/wp-content/uploads/sites/14/2015/12/Drehzahl_Handout.pdf. [Zugriff am 05. Juli 2023].
- [13] EETech Media, LLC., „Tachogenerators,“ [Online]. Available: <https://www.allaboutcircuits.com/textbook/direct-current/chpt-9/tachogenerators/>. [Zugriff am 24. Juli 2023].
- [14] ResearchGate, „Abbildung 4.16: Wenn einzelne Magnete auf eine drehbaren Scheibe...“ [Online]. Available: https://www.researchgate.net/figure/Abbildung-416-Wenn-einzelne-Magnete-auf-eine-drehbaren-Scheibe-montiert-sind-kann-mit_fig9_305443082. [Zugriff am 24. Juli 2023].
- [15] Infineon Technologies AG, „TLE4964-3M,“ 20. Dezember 2019. [Online]. Available: https://www.infineon.com/dgdl/Infineon-TLE4964-3M-DataSheet-v01_20-EN.pdf?fileId=db3a30433cabdd35013cc94b7c0f64f9. [Zugriff am 10. Juli 2023].
- [16] Infineon Technologies AG, „TLE4976-1K,“ 12. Februar 2009. [Online]. Available: https://www.infineon.com/dgdl/Infineon-TLE4976L-DS-v02_00-EN.pdf?fileId=db3a304319c6f18c0119cd9980a57ae7. [Zugriff am 24. Juli 2023].
- [17] Infineon Technologies AG, „TLE4964-1M,“ 20. Dezember 2019. [Online]. Available: https://www.infineon.com/dgdl/Infineon-TLE4964-1M-DataSheet-v01_02-EN.pdf?fileId=db3a3043397219b601397256e86f0059. [Zugriff am 24. Juli 2023].
- [18] König GmbH Kunststoffprodukte, „www.koenig-kunststoffe.de,“ Januar 2016. [Online]. Available: <https://www.koenig-kunststoffe.de/produkte/pc/technisches-datenblatt-pc.pdf>. [Zugriff am 05. Juli 2023].
- [19] KHP Kunststofftechnik e. K., „Werkstoffdatenblatt: PMMA,“ [Online]. Available: <https://www.khp-kunststoffe.de/wb-data/datenblatt-100-pmma.pdf>. [Zugriff am 05. Juli 2023].
- [20] KST Servo GmbH, „BLS815 Digital Metal Gear Brushless Servo 20kg.cm 0.07sec/60degree for 550-700-class Helicopters,“ KST Servo GmbH, [Online]. Available: <https://kstservos.com/collections/standard-servos/products/bls815-digital-20kg->

metal-gear-0-07sec-rc-brushless-standard-servo?_pos=3&_fid=0438a0641&_ss=c.
[Zugriff am 06. Juli 2023].

- [21] JACOB Elektronik GmbH, „Amewi Standard-Servo 6221MG Digital-Servo Getriebe-Material: Metall Stecksystem: JR (28918),“ JACOB Elektronik GmbH, [Online]. Available: <https://www.jacob.de/produkte/amewi-standard-servo-6221mg-28918-artnr-5514148.html>. [Zugriff am 06. Juli 2023].
- [22] reichelt elektronik GmbH & Co. KG, „GRABIT SERVO,“ reichelt elektronik GmbH & Co. KG, [Online]. Available: <https://www.reichelt.at/at/de/pwm-servomotor-bis-20-kg-cm-ersatzservo-fuer-grab-it-roboterarm-grabit-servo-p219023.html?r=1>. [Zugriff am 06. Juli 2023].
- [23] Mouser, „www.mouser.at/datasheet,“ Mouser, 2024. [Online]. Available: https://www.mouser.at/datasheet/2/891/esp32_c3_mini_1_datasheet_en-2578671.pdf. [Zugriff am 29. März 2024].
- [24] Seeed Studio, „Seeed Studio XIAO ESP32S3,“ Seeed Studio, 2024. [Online]. Available: <https://www.seeedstudio.com/XIAO-ESP32S3-p-5627.html>. [Zugriff am 28. März 2024].
- [25] Infineon Technologies AG, „TLE5012B E1000,“ 20. Juni 2018. [Online]. Available: https://www.infineon.com/dgdl/Infineon-TLE5012B_Exxx-DataSheet-v02_01-EN.pdf?fileId=db3a304334fac4c601350f31c43c433f. [Zugriff am 10. Juli 2023].
- [26] Infineon Technologies AG, „TLE4998P3C,“ September 2009. [Online]. Available: https://www.infineon.com/dgdl/Infineon-TLE4998P3C-DataSheet-v01_01-en.pdf?fileId=db3a30431ce5fb52011d3ea4f5e225b8. [Zugriff am 10. Juli 2023].
- [27] RS Components Handelsges.m.b.H., „Alles über Hall-Effekt-Sensoren,“ [Online]. Available: <https://at.rs-online.com/web/content/discovery-portal/produktatgeber/hall-sensoren-leitfaden>. [Zugriff am 11. Juli 2023].
- [28] TDK Corporation, „Hall Switches,“ 2022. [Online]. Available: <https://product.tdk.com/en/products/sensor/switch/hall-switch/index.html>. [Zugriff am 11. Juli 2023].
- [29] FastBitLab, „SPI bus configuration: Full duplex, Half-duplex, and Simplex,“ 12. Juli 2019. [Online]. Available: <https://fastbitlab.com/spi-bus-configuration-discussion-full-duplex-half-duplex-simplex/>. [Zugriff am 12. Juli 2023].

- [30] Total Phase Inc., „Interfacing with 3 wire SPI,“ 2023. [Online]. Available: <https://www.totalphase.com/support/articles/200350046-interfacing-with-3-wire-spi/>. [Zugriff am 13. Juli 2023].
- [31] Premium-Modellbau, „Premium-Modellbau BULL TEC Multicopter Brushless Regler 30A, 2S - 6S (7,4-22,2V),“ Premium-Modellbau, 2024. [Online]. Available: <https://www.premium-modellbau.de/bull-tec-multicopter-brushless-regler-30a-2s-6s-lipo-opto-blheli>. [Zugriff am 26. März 2024].
- [32] C. Grieger, „ESC mit Arduino ansteuern,“ 22. Februar 2020. [Online]. Available: <https://elektro.turanis.de/html/prj204/index.html>. [Zugriff am 26. März 2024].
- [33] Conrad Electronic GmbH & Co KG, „Sol Expert L180 Micro-LiPo-Akku 3.7 V (max) (L x B x H) 20 x 25 x 5 mm,“ Conrad Electronic GmbH & Co KG, 2024. [Online]. Available: <https://www.conrad.at/de/p/sol-expert-l180-micro-lipo-akku-3-7-v-max-l-x-b-x-h-20-x-25-x-5-mm-245072.html>. [Zugriff am 28. März 2024].
- [34] Redhat, „Redhat,“ Redhat, 02 August 2023. [Online]. Available: <https://www.redhat.com/de/topics/cloud-native-apps/what-is-serverless>. [Zugriff am 05 09 2023].
- [35] „OPC Router,“ INRAY INDUSTRIESOFTWARE GMBH, [Online]. Available: <https://www.opc-router.de/was-ist-mongodb/>. [Zugriff am 04 09 2023].
- [36] „knowledgenile,“ [Online]. Available: <https://www.knowledgenile.com/blogs/pros-and-cons-of-mongodb>. [Zugriff am 04 09 2023].
- [37] Astera Analytics-Team, „Astera,“ 03 Oktober 2023. [Online]. Available: <https://www.astera.com/de/knowledge-center/sql-vs-nosql/>. [Zugriff am 23 10 2023].
- [38] R. B., „Hostinger,“ 16 8 2023. [Online]. Available: <https://www.hostinger.com/tutorials/what-is-mysql>. [Zugriff am 05 09 2023].
- [39] „Oracle,“ [Online]. Available: <https://www.oracle.com/mysql/what-is-mysql/#:~:text=MySQL%20is%20a%20relational%20database%20management%20system,-Databases%20are%20the&text=The%20database%20structure%20is%20organized,offers%20a%20flexible%20programming%20environment..> [Zugriff am 05 09 2023].
- [40] „Altexsoft,“ 06 11 2019. [Online]. Available: <https://www.altexsoft.com/blog/firebase-review-pros-cons-alternatives/#:~:text=The%20cons%20of%20Firebase,-Firebase%20Realtime%20Database&text=One%20of%20the%20main%20problems,difficult%20to%20make%20complex%20queries..> [Zugriff am 05 09 2023].

- [41] S. Ivanov, „back4app,“ [Online]. Available: <https://blog.back4app.com/de/was-ist-firebase/>. [Zugriff am 05 09 2023].
- [42] „MQTT,“ [Online]. Available: <https://mqtt.org>. [Zugriff am 05 09 2023].
- [43] „OPC Router,“ [Online]. Available: <https://www.opc-router.de/was-ist-mqtt/>. [Zugriff am 05 09 2023].
- [44] RedHat, „redhat,“ 8 Mai 2020. [Online]. Available: <https://www.redhat.com/en/topics/api/what-is-a-rest-api>. [Zugriff am 05 09 2023].
- [45] N. Parmar, „Medium,“ 03 Juni 2022. [Online]. Available: <https://itznihal.medium.com/api-rest-api-and-restful-api-7767d9997854>. [Zugriff am 23 10 2023].
- [46] m. stevewhims, „Microsoft,“ Microsoft, 13 07 2023. [Online]. Available: <https://learn.microsoft.com/de-de/windows/uwp/networking/websockets>. [Zugriff am 05 09 2023].
- [47] Wallarm, „Wallarm,“ [Online]. Available: <https://www.wallarm.com/what/a-simple-explanation-of-what-a-websocket-is>. [Zugriff am 23 10 2023].
- [48] Raspberry PI Foundation, „raspberrypi,“ raspberrypi, [Online]. Available: <https://www.raspberrypi.org/help/what-%20is-a-raspberry-pi/>. [Zugriff am 05 09 2023].
- [49] K. K., „pluralsight,“ pluralsight, 08 Juni 2023. [Online]. Available: <https://www.pluralsight.com/resources/blog/cloud/what-is-google-cloud-platform-gcp#:~:text=GCP%20is%20a%20public%20cloud%20vendor%20that%20offers%20a%20suite,pay-per-use%20basis..> [Zugriff am 05 09 2023].
- [50] Modellsport-Team Handels GmbH, „d-m-t.at,“ Modellsport-Team Handels GmbH, 2024. [Online]. Available: <https://www.d-m-t.at/motoren-zubehoer/elektro-flug-motoren/elektro-flug-motoren-631/brushless-motor-extron-2814-16-990u-v-von-1000-1600g-detail>. [Zugriff am 23. Mai 2023].
- [51] Conrad Electronic SE , „conrad.de,“ Conrad Electronic SE , 2024. [Online]. Available: <https://www.conrad.de/de/p/az-delivery-mg995-micro-digital-servo-motor-fuer-rc-roboter-hubschrauber-flugzeug-850037144.html?refresh=true>. [Zugriff am 15. Februar 2024].

-
- [52] Premium-Modellbau, „premium-modellbau.de,“ Premium-Modellbau, 2024. [Online]. Available: <https://www.premium-modellbau.de/bull-tec-multicopter-brushless-regler-30a-2s-6s-lipo-opto-simonk-n-fet>. [Zugriff am 26. Juni 2020].
 - [53] P. Schnabel, Elektronik-Fibel, 7 Hrsg., Elektronik-Kompendium, 2014.
 - [54] Wikipedia, „Lichtstärke (Photometrie),“ [Online]. Available: [https://de.wikipedia.org/w/index.php?title=Lichtst%C3%A4rke_\(Photometrie\)&oldid=155515936](https://de.wikipedia.org/w/index.php?title=Lichtst%C3%A4rke_(Photometrie)&oldid=155515936). [Zugriff am 10. August 2016].
 - [55] Datasheet - Kingbright Corporation, „Low Current LED lamps (3mm),“ [Online]. Available: https://cdn-reichelt.de/documents/datenblatt/A500/LED3MM2MAGE_LED3MM2MAGN_LED3MM2MART%23KIN.pdf. [Zugriff am 10 August 2016].
 - [56] Infineon Technologies AG, „Market-News TLE4966V,“ Infineon Technologies AG, 2013. [Online]. Available: <https://www.infineon.com/cms/de/about-infineon/press/market-news/2013/INFATV201312-014.html>. [Zugriff am 05. Juli 2023].
 - [57] ITWissen.info, „Dehnungsmessstreifen (DMS),“ 2019. [Online]. Available: <https://shopde.climate-kic-proposal.org/content?c=dehnmessstreifen&id=7>. [Zugriff am 24. Juli 2023].
 - [58] Data Base Camp, „Data Base Camp,“ 18 12 2021. [Online]. Available: <https://databasecamp.de/daten/mongodb>. [Zugriff am 04 09 2023].
 - [59] „Paessler,“ [Online]. Available: <https://www.paessler.com/it-explained/mqtt>. [Zugriff am 05 09 2023].
 - [60] „Talend,“ [Online]. Available: <https://www.talend.com/resources/what-is-mysql/>. [Zugriff am 05 09 2023].

8. Abbildungsverzeichnis

| | |
|---|----|
| Abbildung 1: Rotorprinzip | 4 |
| Abbildung 2: Systemstrukturplan | 12 |
| Abbildung 3: CycloTech Rotor [1] | 13 |
| Abbildung 4: Blockschaltbild | 15 |
| Abbildung 5: Produktstrukturplan | 16 |
| Abbildung 6: Scrum Projektplan Dokumentation | 16 |
| Abbildung 7: Scrum Projektplan Messdaten und Steuerung Matteo Müller | 17 |
| Abbildung 8: Scrum Projektplan Speichern und Visualisieren von Daten Lucas Lenarcic | 18 |
| Abbildung 9: Scrum Projektplan Mechanischer Aufbau Tim Sperle | 19 |
| Abbildung 10: Veranschaulichung DMS-Prinzip | 22 |
| Abbildung 11: Messsignale eines 6-Achsen Mehrkomponentensensors [6] | 22 |
| Abbildung 12: Prinzip Aufbau Mehrkomponentensensor – Messung auf Rotorwelle | 23 |
| Abbildung 13: Prinzip Magnetostriktion [9] | 24 |
| Abbildung 14: Prinzip der indirekten Drehmomentmessung | 24 |
| Abbildung 15: Drehmomentaufnehmer [10] | 25 |
| Abbildung 16: Drehmomentmessflansch [11] | 25 |
| Abbildung 17: Tachogeneratorprinzip [13] | 26 |
| Abbildung 18: Prinzip der Drehzahlmessung mit Hall-Sensor [14] | 26 |
| Abbildung 19: Stromverbrauch ESP32 [23] | 34 |
| Abbildung 20: Stromverbrauch SEEED ESP32C3 [24] | 34 |
| Abbildung 21: SSC-Interface (open-drain configuration) [25] | 35 |
| Abbildung 22: SSC-Datentransfer [25] | 36 |
| Abbildung 23: Pin Configuration TLE4998P3C [26] | 37 |
| Abbildung 24: Signal Processing Flow TLE4998P3C [26] | 38 |
| Abbildung 25: Timing-Diagramm TLE4964-3M [15] | 39 |
| Abbildung 26: Ausgangssignal TLE4964-3M [15] | 39 |
| Abbildung 27: SPI-Bus configuration: half-duplex [29] | 40 |
| Abbildung 28: Beispiel 3-Wire SPI [30] | 40 |
| Abbildung 29: Pin Configuration TLE4998P3C [26] | 41 |
| Abbildung 30: TLE4998P3C Duty Cycle 90% | 41 |
| Abbildung 31: Pin Configuration TLE4964-3M [15] | 42 |
| Abbildung 32: Outputsignal des TLE4964-3M ohne Pull-Up Widerstand | 42 |
| Abbildung 33: Outputsignal des TLE4964-3M mit Pull-Up Widerstand | 43 |
| Abbildung 34: Schaltung mit Pull-Up Widerstand (TLE4964-3M) [15] | 43 |
| Abbildung 35: ESC mit Arduino/ESP32 ansteuern [32] | 45 |
| Abbildung 36: Transistorschaltstufe der ESC-Ausgangsstufe | 48 |
| Abbildung 37: Simulation der Ausgangsstufe mit R_1 um Faktor 10 kleiner | 49 |
| Abbildung 38: ESC-Ausgangsstufe (3.3V auf 5V) Board-Layout | 50 |
| Abbildung 39: ESC-Ausgangsstufe Platine | 50 |

| | |
|--|-----|
| Abbildung 40: Position der Winkelmesseinheit..... | 52 |
| Abbildung 41: SSC-Interface/3-Wire SPI Taktung des Datentransfers [25] | 53 |
| Abbildung 42: SPI-Kommunikationsstart..... | 54 |
| Abbildung 43: TLE5012B SPI-Kommunikation mit Analogsignalen..... | 54 |
| Abbildung 44: TLE5012B Testaufbau mit LogicAnalyser..... | 54 |
| Abbildung 45: Schaltplan TLE5012B | 56 |
| Abbildung 46: TLE5012B Winkelmessung Board-Layout | 56 |
| Abbildung 47: Schaltplan TLE4964-3M | 58 |
| Abbildung 48: TLE4964-3M Drehzahlmessung Board-Layout | 58 |
| Abbildung 49: Schaltplan TLE4998P3C | 61 |
| Abbildung 50: TLE4998P3C Schubmessung Board-Layout | 61 |
| Abbildung 51: Testaufbau Schubmessungsprogramm | 62 |
| Abbildung 52: Testdaten Programm Schubmessung 4,5% Duty Cycle | 62 |
| Abbildung 53: Testdaten Programm Schubmessung 41% Duty Cycle | 62 |
| Abbildung 54: finales Endprodukt | 63 |
| Abbildung 55: Vergleich SQL und NoSQL [37] | 65 |
| Abbildung 56: API-Prinzip [45]..... | 68 |
| Abbildung 57: Prinzip Websockets [47] | 69 |
| Abbildung 58: Raspberry PI Imager Settings | 71 |
| Abbildung 59: Raspberry PI Imager | 71 |
| Abbildung 60: Terminal - SSH Verbindung aufgebaut | 72 |
| Abbildung 61: Installierte Services Informationen | 74 |
| Abbildung 62: Startup Script und reboot command | 75 |
| Abbildung 63: nginx Service Status..... | 75 |
| Abbildung 64: Website Zugriff über Hostname | 76 |
| Abbildung 65: Website Zugriff über IP-Adresse | 76 |
| Abbildung 66: Postman PUT Method Output..... | 77 |
| Abbildung 67: Starten der API am Server | 77 |
| Abbildung 68: Software Block Diagramm..... | 78 |
| Abbildung 69: Backend Ordner Struktur | 78 |
| Abbildung 70: Postman Get Method Output | 85 |
| Abbildung 71: Postman Get Method Input Body | 85 |
| Abbildung 72: Front-End Website | 86 |
| Abbildung 73: Torque and Thrust Linechart | 87 |
| Abbildung 74: Kräfte Pie Chart | 89 |
| Abbildung 75: Testroutine Bar Chart | 90 |
| Abbildung 76: Front-End Charts | 91 |
| Abbildung 77: Front-End Control Panel..... | 94 |
| Abbildung 78: Motor Control Buttons..... | 96 |
| Abbildung 79: Rotor Geschwindigkeit und Flügelposition Slider..... | 96 |
| Abbildung 80: Test-Routinen Konfigurator..... | 100 |

| | |
|--|-----|
| Abbildung 81: Teststand Status Anzeigen | 101 |
| Abbildung 82: Konzept Grundaufbau | 104 |
| Abbildung 83: Offsetprinzip..... | 106 |
| Abbildung 84: Grundaufbau | 106 |
| Abbildung 85: erster Prototyp – mechanischer Aufbau | 106 |
| Abbildung 86: Verbindung VAI – Achse | 108 |
| Abbildung 87: Reibungskritische Stellen - Kugellager | 108 |
| Abbildung 88: Kräfteveranschaulichung - Zeichnung | 108 |
| Abbildung 89: BRUSHLESS MOTOR EXTRON 2814/16 990U/V VON 1000-1600G..... | 109 |
| Abbildung 90: AZ-Delivery MG995 Micro Digital Servo Motor..... | 110 |
| Abbildung 91: BULL TEC Multicopter Brushless Regler 30A 2S - 6S Lipo Opto SimonK N-FET | 111 |
| Abbildung 92: ESC - Beschaltung | 111 |
| Abbildung 93: Einbau - Zahnriemen | 112 |
| Abbildung 94: Prototyp 2 - Konzept | 113 |
| Abbildung 95: Prototyp 2 – 3D-Zeichnung 1..... | 113 |
| Abbildung 96: Prototyp 2 – 3D-Zeichnung 2..... | 113 |
| Abbildung 97: Motor-Servo-Halterung..... | 114 |
| Abbildung 98: Prototyp 2 – 3D-Zeichnung 3..... | 114 |
| Abbildung 99: Prototyp 2 – mech. Aufbau Seitenansicht..... | 115 |
| Abbildung 100: Prototyp 2 – mech. Aufbau Frontansicht | 115 |
| Abbildung 101: Schubmessung v1 Grundkonzept..... | 116 |
| Abbildung 102: Magnetsensor - Messprinzip | 117 |
| Abbildung 103: Schubmessung v1 Prinzip | 118 |
| Abbildung 104: Schubmessung v1 Zeichnung | 119 |
| Abbildung 105: Einzelteile - 3. Prototyp | 120 |
| Abbildung 106: Prototyp 3 - Explosionsansicht | 121 |
| Abbildung 107: Prototyp 3 - Konzept | 124 |
| Abbildung 108: Schubmessung v2 - Konzept..... | 125 |
| Abbildung 109: Schubmessung v2 - Zeichnung | 126 |
| Abbildung 110: Schubmessung v2 | 126 |
| Abbildung 111: Drehzahlmessung - Zeichnung | 127 |
| Abbildung 112: Tragflächenwinkelmessung - Konzept..... | 128 |
| Abbildung 113: Tragflächenwinkelmessung - Zeichnung | 129 |
| Abbildung 114: Finaler Aufbau - Zeichnung | 131 |
| Abbildung 115:Scrum Projektplan Dokumentation mit Tasks..... | 135 |
| Abbildung 116: Scrum Projektplan Messdaten und Steuerung mit Tasks Matteo Müller | 135 |
| Abbildung 117: Scrum Projektplan Speichern und Visualisieren mit Tasks Lucas Lenarcic... | 135 |
| Abbildung 118: Scrum Projektplan Mechanischer Aufbau mit Tasks Tim Sperle | 136 |
| Abbildung 119: ESC-Ausgangsstufe (Darstellung für 5V-Ausgang) | 137 |
| Abbildung 120: Besprechungsprotokoll ELTI Vorprojektphase vom 13.06.2023 | 139 |

| | |
|--|-----|
| Abbildung 121: Besprechungsprotokoll ELTI Iteration 1 vom 26.11.2023 | 140 |
| Abbildung 122: Besprechungsprotokoll ELTI Iteration 3 vom 30.01.2024 | 141 |
| Abbildung 123: Besprechungsprotokoll ELTI Iteration 4 vom 18.03.2024 | 142 |
| Abbildung 124: Besprechungsprotokoll ET Vorprojektphase vom 08.06.2023 | 143 |
| Abbildung 125: Besprechungsprotokoll ET Infineon vom 03.07.2023 | 144 |
| Abbildung 126: Besprechungsprotokoll ET Iteration 3 vom 06.02.2024 | 145 |

9. Tabellenverzeichnis

| | |
|---|-----|
| Tabelle 1: Drehzahlmessung mit Hall-Sensor - Parametervergleich [15] [16] [17]..... | 28 |
| Tabelle 2: Prototyp 1 – Komponentenliste..... | 105 |
| Tabelle 3: Kugellager – Rotor..... | 122 |
| Tabelle 4: Schrauben - Rotor | 122 |
| Tabelle 5: Muttern – Rotor | 122 |
| Tabelle 6: Beilagscheiben – Rotor | 122 |
| Tabelle 7: Zahnräder – Rotor | 123 |
| Tabelle 8: Zahnriemen – Rotor | 123 |
| Tabelle 9: gedruckte Teile – Rotor..... | 123 |
| Tabelle 10: gedruckte Teile – Prüfstand | 129 |
| Tabelle 11: Magnete - Prüfstand | 129 |
| Tabelle 12: Druckfedern – Prüfstand..... | 129 |
| Tabelle 13: Kugellager – Prüfstand..... | 130 |
| Tabelle 14: Schrauben – Prüfstand..... | 130 |
| Tabelle 15: Muttern – Prüfstand | 130 |
| Tabelle 16: Kostenaufstellung | 138 |
| Tabelle 17: Arbeitszeitnachweis von Matteo Müller..... | 149 |
| Tabelle 18: Arbeitszeitnachweis von Lucas Lenarcic | 151 |
| Tabelle 19: Arbeitszeitnachweis von Tim Sperle | 154 |