OSTBAYERISCHE
TECHNISCHE HOCHSCHULE
REGENSBURG

Datenverarbeitung in der Technik

Sommersemester 2025

Projektbericht

Teammitglieder:

Fabian Becker _____

Jendrik Jürgens _____

Nicolas Koch _____

Michael Specht _____

Jonathan Wohlrab _____

Betreuung:
Dr. Alexander Metzner, Matthias Altmann

Abgabedatum:
15.07.2025

29. Juni 2025

Inhaltsverzeichnis

1. CAD-Konstruktion

1.1. Setup und Einarbeitung (Becker, Specht)

Zu Beginn des Projekts wurde in Abstimmung mit Fabian Becker sowie im Austausch mit Andreas Wittmann (Studienkollege) entschieden, FreeCAD als CAD-Software zu verwenden. Der Grund hierfür war die einfache Kollaboration innerhalb der Projektgruppe sowie der unkomplizierte Erfahrungsaustausch mit der Arbeitsgruppe um A. Wittmann. Andere Softwarelösungen wie OnShape wurden diskutiert, aufgrund der Komplexität und der damit verbundenen Einarbeitungszeit im Hinblick auf die Projektlaufzeit jedoch verworfen. FreeCAD ist zudem eine Open-Source-Software, die neben Fedora auch auf Debian-Systemen lauffähig ist. So konnte die Software problemlos auf den Arbeitsrechnern der Teammitglieder installiert werden.
Grundsätzlich stützt sich die Konstruktion auf vorhandene STL-Vorlagen. Ein Beispielprojekt aus dem Internet diente als Grundlage für die Arbeit.

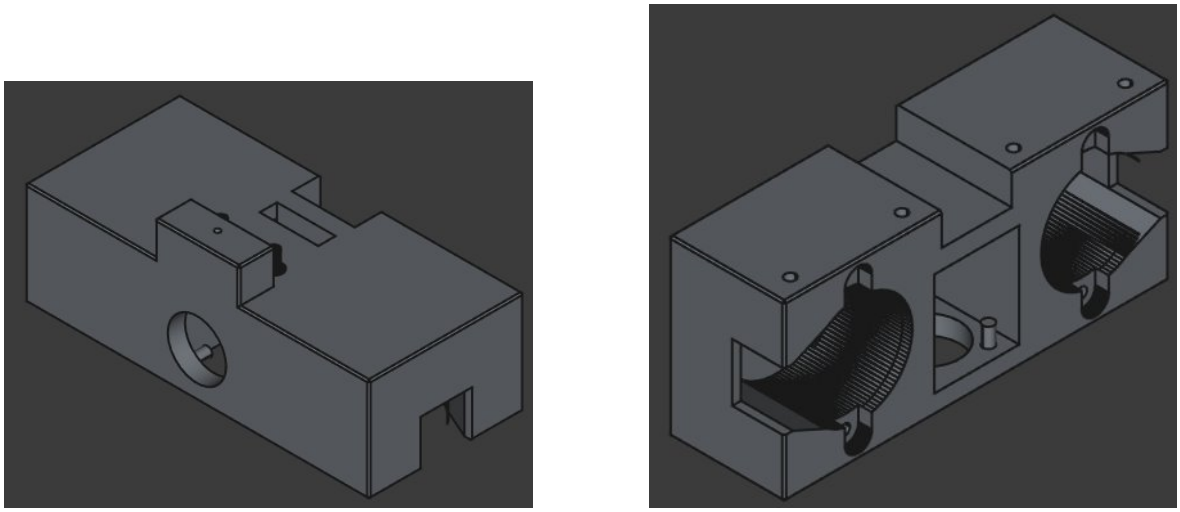
1.2. Geschützarm Version 1 (Specht)

Bevor mit der Konstruktion begonne wurde, wurde im Team besprochen, welche Komponenten nötig sind, um die Position des Flugobjekts eindeutig zu bestimmen. Die Wahl fiel auf folgende Komponenten, die aus vorherigen Studienprojekten übernommen werden konnten:

- GY-521 MPU-6050 3-Achsen-Gyroskop und Beschleunigungssensor
- SRF02 Ultraschall Entfernungssensor
- Raspberry Pi 5 Kamera Modul
- 2x 28BYJ-48 Schrittmotor

Ziel des ersten Entwurfs war es, diese kompakt auf dem Arm zu integrieren. Die angedachten Schrittmotoren wichen jedoch von der Vorlage aus dem Beispielprojekt ab, weshalb der Geschützarm von Grund auf neu konstruiert werden musste.
Alle Module sollten zentral über der Abschusseinrichtung platziert werden, um eine korrekte Berechnung der Flugbahn zu ermöglichen. Der Ultraschall-Sensor sollte dabei hochkant nach vorne gerichtet sein, um die Entfernung zum Ziel zu messen. Die Kamera sollte schräg nach oben gerichtet sein, um den Himmel zu überwachen. Der Beschleunigungssensor sollte liegend

auf dem Arm montiert werden, um die Beschleunigung des Arms zu messen. Die Schrittmotoren mussten in einem geeigneten Abstand zueinander montiert werden, sodass die Flywheel-Konstruktion des Arms funktioniert. Letzteres konnte durch das Vermessen der Vorlage aus dem Beispielprojekt realisiert werden. Für die restlichen Anforderungen waren die korrekten Maße nötig. Für die Montage der Kamera konnte eine bereits 3D-gedruckte Halterung aus einer anderen Gruppe benutzt werden. Die Haltevorrichtung für den Beschleunigungssensor wurde aus einer STL-Vorlage übernommen und angepasst. Auch für die Schrittmotoren konnte auf ein Modell aus dem Internet zurückgegriffen werden, weshalb es nicht nötig war, die komplexen Geometrie eigenständig zu ermitteln. Einzig die Maße für den Ultraschall-Sensor wurden recherchiert und durch Nachmessen validiert.



(a) Geschützarm Version 1 - Frontansicht

(b) Geschützarm Version 1 - Rückansicht

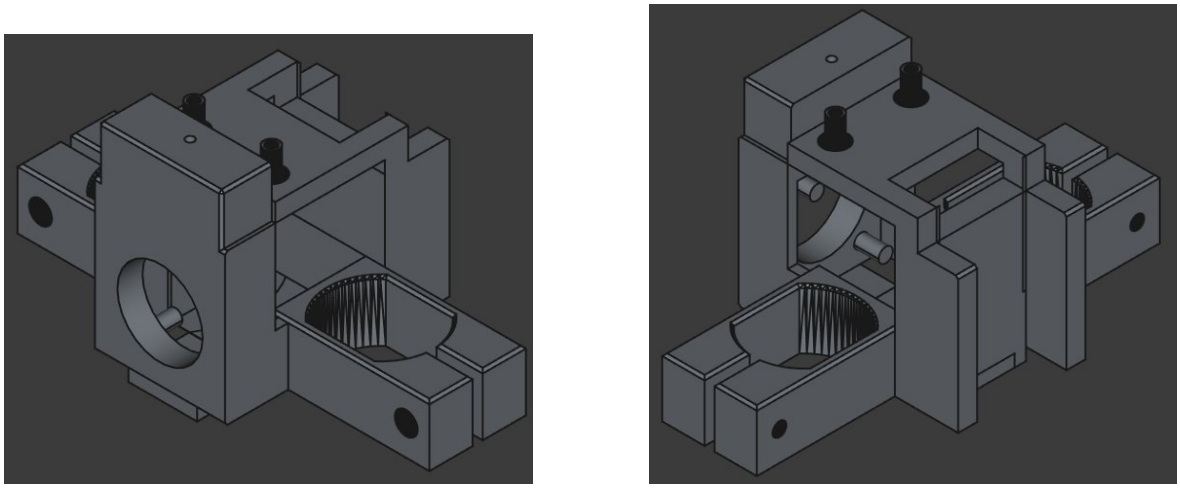
Abbildung 1.1: Geschützarm Version 1

Neben den eigentlichen Maßen der Komponenten war das Kabelmanagement ein wichtiges Thema. Alle Kabel sollten nach hinten entlang am Magazin geführt werden, um eine saubere Optik zu gewährleisten. Für Beschleunigungssensor und Kamera stellte dies kein Problem dar, da diese ganz oben angebracht sein sollten. Der Ultraschall-Sensor und die Schrittmotoren hingegen waren in das neukonstruierte Gehäuse integriert, sodass Aussparungen, wie in Abbildung 1.1 zu sehen, angebracht werden mussten, um die Kabel aus dem Gehäuse herauszuführen.
Außerdem musste sichergestellt werden, dass der Geschützarm an das Magazin montiert werden kann. Hierzu wurden vom Kollegen Fabian Becker Montagepunkte am Magazin konstruiert, die mit dem Geschützarm verschraubt werden können.

1.3. Geschützarm Version 2 (Specht)

Im Laufe des Projektes wurde in Abstimmung mit Fabian Becker klar, dass die Schrittmotoren aufgrund zu geringer Leistung nicht für die Flywheel-Konstruktion geeignet sind. Daraufhin wurde sich für die originalen Motoren aus dem Beispielprojekt entschieden. Das hatte zur Folge, dass der Geschützarm neu konstruiert werden musste, da die Maße der neuen Motoren

von den alten abwichen. Aus diesem Grund wurde der Geschützarm der Vorlage als Basis genommen und die Grundidee der Version 1 beibehalten.



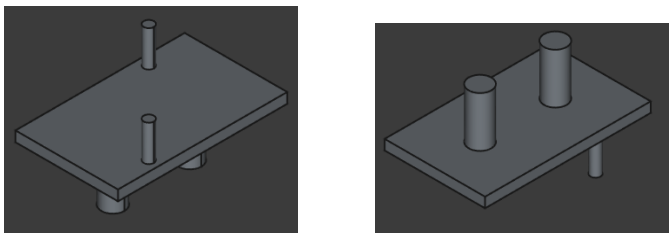
(a) Geschützarm Version 2 - Frontansicht (b) Geschützarm Version 2 - Rückansicht

Abbildung 1.2: Geschützarm Version 2

Im Gegensatz zur Version 1 wird der Ultraschallsensor nun seitlich eingeführt anstatt von unten, siehe 1.2. Problematisch waren dabei die beschränkten Platzverhältnisse, da die Schrittmotoren näher am Kanonenrohr angebracht wurden als im vorherigen Entwurf. Außerdem wurden die zuvor angedachten Montagepunkte am Magazin wieder entfernt. Die Zusammenführung des Geschützarms mit dem Magazin erfolgte deshalb mittels Modellbaukleber.

1.4. Montagehalterung Motortreiber (Specht)

Für die ersten Funktionstests wurden die Pololu-Motoren provisorisch auf Kork-Schnipseln montiert. Dieses Vorgehen ermöglichte eine zügige Inbetriebnahme, jedoch erwies es sich hinsichtlich Stabilität und Sicherheit als unzureichend. Im Rahmen des Tests kam es zum Abrutschen eines Motors von der Korkunterlage, was zu einer kurzfristigen Wärmeentwicklung und Geruchsbildung führte. Glücklicherweise wurde eine Beschädigung der Hardware vermieden.



(a) Polulu - Draufsicht (b) Polulu - Bodensicht

Abbildung 1.3: Montagehalterung für Pololu-Motortreiber

Für die finale Abnahme wurde daher ein dauerhaftes und sicheres Montagesystem umgesetzt, das ein sauberes und zuverlässiges Setup gewährleistet. Wie in Abbildung 1.3 zu sehen, kann die Halterung direkt auf der Montageplatte des Fahrzeugs geklippt werden.

2. ESP32 Programmierung

2.1. Motor-Treiber (Specht)

Wesentlich für den Einstieg in die Motor-Treiber-Programmierung waren die verwendeten Hardware-Komponenten. Folgende Teile wurden verwendet:

- 2x Pololu G2 High-Power Motor Driver 24v13 (MD31C)
- 2x MFA/Como Drills 919D501

Im ersten Ansatz wurde der Versuch unternommen, den Low-Level-Treiber (LL-Treiber) direkt mit der Differenzialantriebs-Logik (Differential Drive) zu koppeln. Nach intensiver Recherche und ersten Programmieransätzen wurde jedoch deutlich, dass eine klare Trennung beider Ebenen unter modularen Gesichtspunkten vorzuziehen ist. Diese Trennung resultiert in einer signifikanten Steigerung sowohl der Wiederverwendbarkeit als auch der Wartbarkeit des Codes. Abhilfe schafften hierbei vor allem die Verwendung von ESP-IDF-Komponenten.

Um die Ansteuerung der Motoren zu realisieren, war es von zentraler Bedeutung, die Spezifikationen der verwendeten Hardware zu berücksichtigen. Das Datenblatt der Pololu-Motor-Treiber wurde in Form einer Webseite gefunden. Die darin enthaltenen Informationen waren ausreichend, um die Logik zu implementieren. Für die verwendeten Motoren lieferte das Datenblatt insbesondere elektrische Kenngrößen, die für die Absicherung der Hardware von entscheidender Bedeutung waren. Dazu zählten maximale und nominale Ströme. Zu Beginn des Projektes konnte nur auf einen 6V-Akku zurückgegriffen werden, obwohl die Motoren bis zu 12V-Betriebsspannung zulassen. Daraufhin wurden entsprechende Widerstände auf den Treiberboards angebracht, um den maximalen Strom für 6V zu begrenzen. Des Weiteren konnte aus dem Datenblatt des Pololu-Motor-Treibers die Notwendigkeit überdimensionierter Kondensatoren abgeleitet werden, um eine gute und stabile Performance sicherzustellen.

Der Fokus der Implementierung lag vorrangig auf Modularität, Erweiterbarkeit, Clean-Code und Best-Practices. Dafür wurde wenn möglich auf globale Variablen verzichtet und stattdessen auf die Verwendung von Strukturen und Funktionen in Kombination mit Pointern gesetzt.

2.1.1. Low-Level Treiber

Die Aufgabe des LL-Treibers bestand darin, genau einen Motor anzusprechen und zu steuern. Das verwendete Framework ESP-IDF bietet eine Vielzahl an API-Funktionen, die eine ab-

strahierte und einfache Ansteuerung der Hardware ermöglichen. Für das Erzeugen von PWM-Signalen sind vor allem zwei API's von zentraler Bedeutung:

- LED Control (LEDC)
- Motor Control Pulse Width Modulator (MCPWM)

Wie den Namen zu entnehmen ist, ist LEDC für die einfache Ansteuerung von LEDs gedacht, während MCPWM speziell für Motoren entwickelt wurde. Der MCPWM-Generator besteht aus einer Reihe von Submodulen, wie bspw. einem Fault-Module und einem Brake-Operator. Die Pololu-Boards bieten ebenfalls einen Fault-Pin, weshalb im weiteren Projektverlauf der MCPWM-Generator verwendet wurde, um diese Funktionalität nutzen zu können.

Die MCPWM-API umfasst mehrere Funktionen und Strukturen. Deswegen wurde der erste Entwurf auf Basis einer Kombination aus KI-generierten Code und Beispielcode von Github erstellt. Die grundlegende Funktionalität konnte dadurch unkompliziert und schnell erfasst werden, wodurch Zeit gespart wurde. Nichtsdestotrotz war es notwendig, entsprechende Literatur zur API zu lesen und zu verstehen. Im Folgenden wurde der Code Stück für Stück angepasst, modularisiert und erweitert.

Ein Key-Konzept entstand aus dem Gedanken, was passieren würde, wenn ein Duty-Cycle von beispielsweise 100 % (Volllast) gesetzt wird und die Drehrichtung des Motors umgekehrt wird. Die Annahme war, dass der Motor in diesem Fall solange als Generator arbeitet, bis die Richtung letztendlich umgekehrt wird. Der dabei möglicherweise auftretende Rückstrom könnte eventuell die Hardware beschädigen. Aus diesem Grund wurde sich auf eine Ramping-Strategie geeinigt, die eine sichere und kontrollierte Änderung der Drehrichtung ermöglicht. Dabei soll sichergestellt werden, dass der Duty-Cycle des PWM-Signals bzw. die Richtung für den Motor nicht abrupt geändert wird. Stattdessen wird der Duty-Cycle in konfigurierbaren Schritten dem Nullbereich angenähert. In einem sicheren Hysteresebereich wird dann die Richtung geändert und der Duty-Cycle der neuen gewünschten Geschwindigkeit angepasst.

Ein weiterer wichtiger Baustein sollte das automatisierte Erkennen von Fehlern der Motoren sein. In solch einem Fall sollte der Motor sofort gestoppt werden und eine Signalleuchte angehen. Ein einfacher Testaufbau bestehend aus Pull-up Widerständen und einer LED sollte die grundlegende Funktionalität des Fault-Pins sicherstellen.

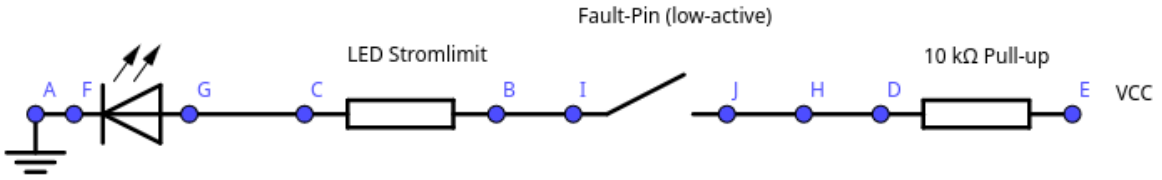


Abbildung 2.1: Testaufbau des Fault-Pins des Pololu-Motor-Treibers

Der Fault-Pin ist laut Datenblatt ein Open-Drain-Ausgang, der bei einem Fehler auf Low gezogen wird. Im Testaufbau wurde ein Pull-Up-Widerstand von 10 kΩ verwendet, um den Pin im Normalfall auf High zu halten. Folgendes Verhalten wurde erwartet: Bei einem Fehler wird der Pin auf Low gezogen und die LED leuchtet auf. Dies war jedoch nicht der Fall. Stattdessen leuchtete die LED dauerhaft, obwohl der Motor einwandfrei funktionierte. Nach Rücksprache mit dem Betreuer stellte sich heraus, dass dieser Pin in der Vergangenheit nie benutzt wurde. Um den Projektfortschritt nicht unnötig zu gefährden, wurde beschlossen, den Fault-Pin nicht weiter zu verwenden und mit der Implementierung des Differential Drive fortzufahren.

2.1.2. Low-Level Treiber Test

Das Testing wurde in mehreren Schritten durchgeführt. Für die Erstellung der Testfälle wurde dabei zum Großteil KI-generierter Code verwendet. Dadurch konnte der damit verbundene Zeitaufwand signifikant minimiert und zugleich eine minimale Testabdeckung garantiert werden.

Zunächst sollte die grundlegende Funktionalität des LL-Treibers getestet werden. Dazu wurden LEDs auf einem Breadboard angebracht, welche die Motoren simulieren sollten. Die Helligkeit der LEDs sollte dabei die Geschwindigkeit des Motors repräsentieren.

Nach erfolgreicher Validierung der Ergebnisse wurde der Testaufbau an die echte Hardware angepasst. Zudem sollte ein Wrapper erstellt werden, der beide Motoren über einzelne Treiber-Objekte anspricht. Dies stellte somit eine Vorstufe zum Differential Drive dar. Ziel war es, die beiden Motoren unabhängig voneinander ansteuern zu können. Dafür wurde das Fahrzeug auf einer Holzkonstruktion bestehend aus zwei Holzlatten platziert. Obwohl die Tests positiv verliefen, sollte sich der Test ohne Bodenkontakt im weiteren Projektverlauf als suboptimal herausstellen.

2.1.3. Differential Drive

2.1.4. Tests

2.1.5. Integration

2.2. MQTT-Anbindung (Specht)

2.2.1. WiFi-Stack

2.2.2. MQTT-Stack

Stundenliste

Name: Michael Specht
Gruppe: 1 Himmelwacht

Pos.	Bezeichung	Beschreibung	Stunden in h
1	Einführung	Gruppe, Thema, Herangehensweise	4
2	Einführung	GANTT, Lastenheft, Einteilung, Teileliste	8
3	Flieger	Fliegermodell bauen + Arm für Befestigung	2
4	CAD - Setup	Einführung in 3D-Druck, Test verschiedener Software (onshape, FreeCAD), Arbeitsaufteilung	5
5	CAD - Geschützarm v1	eigene Konstruktion des Geschützarms für Anbringen von Ultraschall- und Gyrosensor + Flywheel-Motoren + Kamera	15
6	Dokumentation lesen	Dokumentation zu Motor-Control-PWM (MCPWM), Motortreiber und Motoren suchen und lesen	5
7	HW Setup Motortreiber	Widerstände und Kondensatoren laut Datenblatt angebracht	2
8	ESP-IDF Motortreiber	Programmierung Low-Level Treiber für 919D51 Gearbox Servomotoren mit Pololu MD31C Treiberboards	15
9	ESP-IDF Differential Drive	Logik für Diff-Drive Wrapper erstellt und programmiert	10
10	Test Motortreiber	Test am Fahrzeug + Debugging	5
11	Test Differential Drive	Test am Fahrzeug + Debugging	10
12	Integration Differential Drive	Steuerung mittels PS4-Controller	5
13	Integrationstest Differential Drive	Deadzones festlegen, Diff-Drive Algorithmus anpassen	10
14	HW Setup Motortreiber	Anpassungen aufgrund zu geringer Spannung bzw. Leistung	2
15	CAD - Geschützarm v2	komplette Überarbeitung aufgrund zu schwacher Servomotoren	10
16	CAD - Pololu Mounts	Mounts für Motortreiberboards erstellt	1
17	HW Setup Fahrzeug	Diverse Arbeiten für Zusammenbau	3
18	HW Setup Fahrzeug	Keilrippenriemen-Trieb zerlegt, gereinigt, zusammengebaut	2
19	Flywheel Motoren	Test Flywheelmotoren	2
20	ESP-IDF Differential Drive Algorithmus	Umbau auf Festkommaarithmetik	3
21	ESP-IDF MQTT	Integration WiFi- + MQTT-Stack für KI-Integration	15
22	Integration MQTT	Test an Fahrzeug + Debugging	5
23	3D-Druck	Diverse Drucks begleitet (Slicer, Infill entfernen)	5
24	Dokumentation erstellen	Wissenschaftliche Arbeit verfassen	20
25	Präsentation erstellen	eigenen Teil (CAD, ESP32 Programmierung) hinzufügen	3
		Gesamt:	**167**

Abbildungsverzeichnis

Tabellenverzeichnis