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

2. Juli 2025

Inhaltsverzeichnis

1. Stromversorgung

1.1. Analyse des Aufbaus und der Komponenten des vorherigen Projekts (Koch)

Zu Beginn wurde die bestehende Stromversorgung und die dafür genutzten Komponenten eines früheren Semesterprojekts analysiert, um anhand dessen bestimmen zu können, welche Teile wiederverwendet werden können, sowie ob das gegebene Layout in etwa für das eigene Projekt genutzt werden kann.
Essentiell bestand die Stromversorgung aus zwei Step-Down-Wandlern, die aus einer Eingangsspannung eine 8V und eine 5V Ausgangsspannung erzeugten, was ebenfalls für unser eigenes Projekt benötigt wird. Außerdem wurden zwei Verteiler genutzt, um die Spannungen auf die verschiedenen Sensoren und Aktoren zu verteilen. Das vorhandene Layout auf dem Lochrastergerüst war für uns jedoch nicht geeignet, da wir einen übersichtlicheren Aufbau und ein sinnvolles Color-Coding der Kabel für die verschiedenen Anschlüsse und für einen besseren Überblick anstrebten.

1.2. Aufbau der eigenen Stromversorgung (Koch)

Nachdem die vorhandenen Teile analysiert wurden, wurde die Entscheidung getroffen nur die Step-Down-Wandler, da der Rest nicht relevant für unser Projekt war. Lediglich die Verteiler brauchten wir auch, mussten allerdings ersetzt werden, da die Schraubverbindungen kaputt waren. Die Step-Down-Wandler waren so aufgebaut, dass ein Modul die Eingangsspannung erhielt und am Ausgang ein selbstangefertigtes Y-Kabel hatte, welches dann jeweils in einen Verteiler und in den anderen Step-Down-Wandler ging. Diese Kombination sollte auch so für unser Projekt übernommen werden, allerdings mussten dafür die Kabel erneuert werden, da die alten Kabel nicht dem geplanten Color-Coding entsprachen und zu kurz waren. Dabei stellte sich heraus, dass der entstandene Durchmesser, durch die Kombination aus zwei Kabeln zu einem Y-Kabel, zu groß war, um in die Steckverbindung zu passen. Aus diesem Grund entstand das alternative Konzept die ausgehenden Kabel des ersten Step-Down-Wandlers mit dem ersten Verteiler zu verbinden. Das war vor allem dadurch leicht realisierbar, da jeder Verteiler 12 Ports besitzt und 8V lediglich für die Motoren zum fahren benötigt werden. Somit konnte eine Verbindung vom 8V-Verteiler zum zweiten Step-Down-Wanlder hergestellt werden ohne dabei die Steckverbindungen zu beschädigen.

Das Color-Coding der Kabel wurde wie folgt eingeführt:

- **Rot:** Versorgungsspannung
- **Schwarz:** Masse
- **Gelb:** PWM-Verbindung für Motoren
- **Weiß:** Direction Pin für Motoren

Des Weiteren wurde darauf geachtet, dass die Kabel so kurz wie möglich gehalten werden und wenn möglich unter der Platte verlegt werden, um eine bessere Übersicht zu gewährleisten.
Als Eingangsspannung wurde zu Beginn ein 6V-Batterieverbund genutzt, der im Laufe des Projekts durch einen 12V-Batterieverbund ausgetauscht wurde, da beim Testing der Motortreiber festgestellt wurde, dass die Motoren eine höhere Spannung benötigen, um korrekt zu funktionieren. Außerdem wurde versucht den Raspberry Pi 5 über den 5V-Verteiler zu versorgen, was jedoch nicht funktionierte, da die Stromstärke zu niedrig war, wenn der Pi aufwendigere Aufgaben erledigen musste. Aus diesem Grund wurde eine Powerbank genutzt, die den Pi mit Strom versorgt und somit die 5V-Verteilung entlastet.
Der Gesamtaufbau der Stromversorgung sieht dabei wie folgt aus:

- **12V-Batterieverbund:**
  - Step-Down-Wandler (8V) → 8V-Verteiler
    - 2 PWM Boards für Motoren
    - Step-Down-Wandler (5V) → 5V-Verteiler
      - ESP32
      - 2 PWM Boards für die Flywheel Motoren
      - Servo-Motor für die Geschützplattform
      - Servo-Motor für den Geschützarm
- **Powerbank:**
  - Raspberry Pi 5
    - Pi-Camera
    - MPU6050 Gyrosensor
    - SRF02 Ultraschallsensor

2. CAD-Konstruktion

2.1. 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.

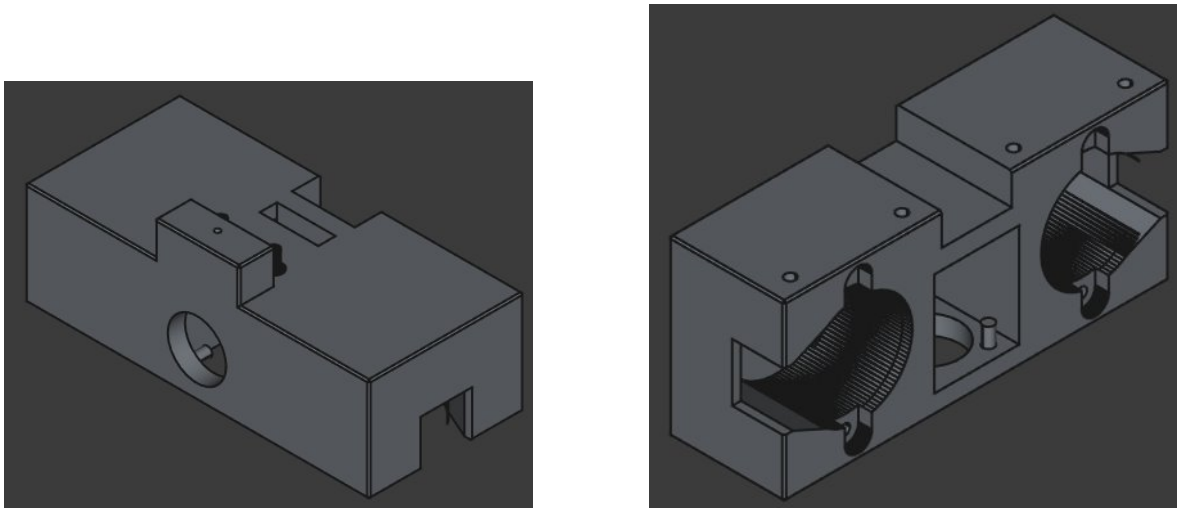
2.2. Geschützarm Version 1 (Specht)

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

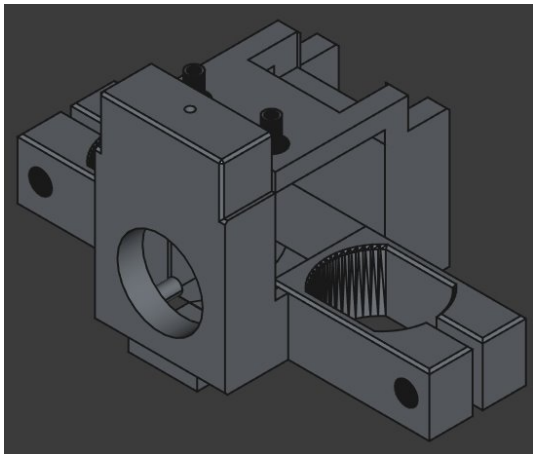
Abbildung 2.1: Geschützarm Version 1

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 2.1 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.

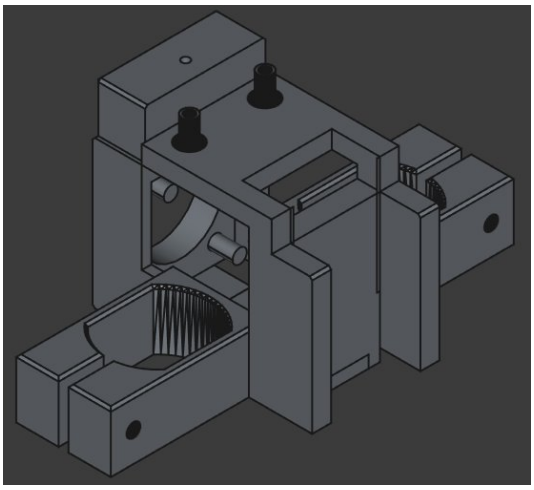
2.3. Geschützarm Version 2 (Specht)

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



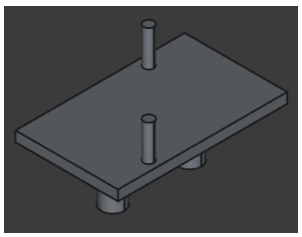
(b) Geschützarm Version 2 - Rückansicht

Abbildung 2.2: Geschützarm Version 2

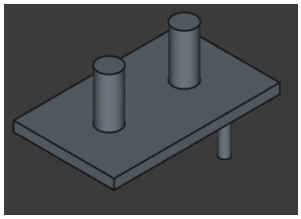
Im Gegensatz zur Version 1 wird der Ultraschallsensor nun seitlich eingeführt anstatt von unten, siehe 2.2. Problematisch waren dabei die beschränkten Platzverhältnisse, da die Schrittmotoren näher am Kanonenrohr angebracht wurden als im vorherigen Entwurf. Außerdem wurden die zuvor angedachten Montagepunkte am Magazin wieder entfernt. Die Zusammenführung des Geschützarms mit dem Magazin erfolgte deshalb mittels Modellbaukleber.

2.4. 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 2.3: 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 2.3 zu sehen, kann die Halterung direkt auf der Montageplatte des Fahrzeugs geklippt werden.

2.5. Geschützplattform (Becker)

Als Geschützplattform wird der Unterbau des Geschützes bezeichnet, welcher den Geschützarm mit der Lochplatte des Fahrzeugs verbindet. Diese Plattform besteht aus zwei, 3D-gedruckten Komponenten.

2.5.1. Unterbau

Der erste Teil des Objekts ist der Unterbau, welcher eine zylindrische Form aufweist und mit einer Bodenplatte versehen ist. Diese ist mit Schraublöchern ausgestattet, welche dazu dienen, den Aufbau mit dem Fahrzeug zu verbinden. Das vorliegende Bauteil wurde aus dem vorherigen Projekt übernommen, da die Konstruktion bereits auf dem Fahrzeug verbaut war und eine Eigenkonstruktion sehr ähnlich aufgebaut wäre.
Der Unterbau ist so konstruiert, dass er Platz für folgende Komponenten bietet:

- Eine **PCA9685 PWM-Treiberplatine** zur Ansteuerung der Servomotoren auf dem Geschütz. Die erforderlichen Befestigungsbohrungen für die Platine waren bereits im Design integriert.
- Einen **MG996R Servomotor**, der für die Rotation des darüberliegenden Aufbaus verantwortlich ist.

2.5.2. Abdeckung

Die zweite Komponente ist die Abdeckung des Unterbaus. Diese verfügt über Bohrungen zur Verbindung mit dem im Unterbau positionierten Servomotor sowie über Montagepunkte für den Geschützarm.

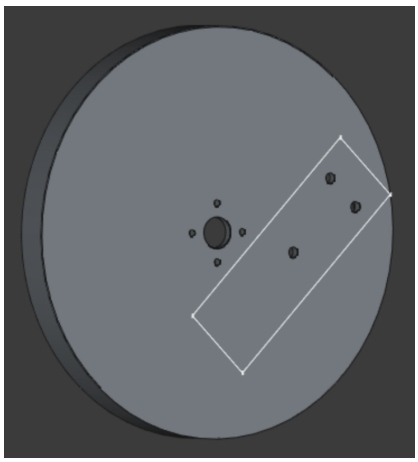


Abbildung 2.4: Abdeckung Plattform mit Position des Verbindungsstücks

Obwohl auch diese Abdeckung bereits vorhanden war, wurde sie exakt vermessen und in FreeCAD rekonstruiert. Ziel dieser Rekonstruktion war es, wie in Abbildung 2.4 ersichtlich, die Bohrlöcher für die Verbindung mit dem Geschützarm so zu positionieren, dass das Verbindungsstück und das Magazin ohne weitere Anpassungen direkt montiert werden konnten.

2.6. Magazin und Verbindungsstück (Becker)

Das Magazin, sowie das Verbindungsstück wurden auf Basis einer bestehenden Konstruktionsvorlage [1] 3D-gedruckt. Das Verbindungsstück ist so ausgelegt, dass es einen weiteren MG996R Servomotor aufnimmt, welcher die vertikale Neigung des Geschützes steuert.
Die Struktur des Magazins umfasste einen linken und einen rechten Teil, die in der Vorlage zusammengeklebt wurden. Wie bereits im Abschnitt 2.2 dargelegt, erfolgte für die Verbindung mit dem Geschützarm der ersten Version die Konstruktion von Verbindungsstücken an beiden Enden des Magazins, um mittels Schrauben eine Verbindung zwischen beiden Teilen zu gewährleisten. Darüber hinaus wurden Schraublöcher in beide Teile des Magazins integriert, um eine Verbindung beider Teile mittels Schrauben zu ermöglichen.
In der zweiten Version des Geschützarmes wurden die Verbindungsstücke entfernt, da der Geschützarm nun direkt mit dem Magazin verbunden wird. Die vorgenommene Änderung resultierte aus der Tatsache, dass es aufgrund der signifikant geringeren Dimension des Geschützarmes unmöglich war, Verbindungsstücke mit Schraublöchern zu versehen.
Zuletzt wurde auch die Länge des Laufs vergrößert, um eine bessere Stützung des Geschützarms zu gewährleisten.

2.7. Magazingewicht (Becker)

Das Magazin des Geschützes verfügt über eine Kapazität für sechs Nerf-Darts sowie ein Magazingewicht. Letzteres dient dazu, einerseits bei hoher Vibration des Fahrzeugs das Ausfallen der Darts zu verhindern und andererseits sicherzustellen, dass nach einem Schuss das nächste Geschoss nachrutscht.
Zunächst wurde die Vorlage [1] für das Magazingewicht angepasst, indem der Projektname eingraviert wurde.

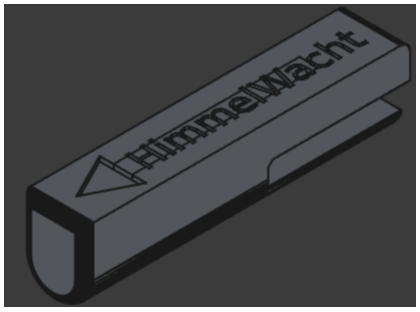


Abbildung 2.5: Magazingewicht

Das Gewicht besitzt außerdem eine Aussparung, die dafür sorgt, dass das Bauteil die Bewegung des Servo Motors nicht einschränkt, welcher die Darts bei der Schussabgabe in die Flywheel Motoren befördert. Ohne entsprechende Aussparung würde der Servo versuchen, das Gewicht in den Lauf zu schieben, was zu Materialschäden, entweder am Motor oder am Geschütz führen würde.
Die vorliegende Aussparung führte allerdings zu Problemen. Wird die Platform sehr weit nach hinten geneigt, so kam es vor, dass der hintere Teil des Gewichts nicht schwer genug war, um

die Nerfs-Darts in den Lauf zu drücken. Die Lösung für dieses Problem bestand im Einsatz selektiver Infill-Technik. Die übrigen Teile wurden standardmäßig mit einem Infill-Gehalt von 15% gedruckt. Im hinteren Teil des Gewichts wurde jedoch eine 100% Infill eingesetzt. Dieses Vorgehen resultierte in der Behebung des Problems.

2.8. Geschütztrigger (Becker)

Der Auslösemechanismus initiiert den Schussvorgang. Durch die Aktivierung eines MG92B Servomotors wird ein Dart in die laufenden Flywheel-Motoren geschoben, welche ihn beschleunigen und abfeuern.
Der Mechanismus ist eine zweiteilige Konstruktion, die auf einer bestehenden Vorlage [1] basiert, deren Schraublöcher jedoch für den spezifischen Anwendungsfall angepasst wurden.
Am unteren Teil wurde eine Öffnung für eine M4-Schraube konstruiert, welche in das Magazin hineinreicht und Kraft auf den Dart ausübt. Bei der Verbindung der beiden Teile wurde darauf geachtet, dass das Schraubloch des ersten Teils etwas größer ist, so dass die Schraube hier nicht greift. Diese wird lediglich im zweiten Teil festgeschraubt, wodurch eine Art Gelenk entsteht. Abschließend erfolgt noch die Verbindung des ersten Teils mit dem Servomotor.

2.9. Halterungen Stormversorgung (Becker)

Da einige Komponenten der Stromversorgung ebenfalls in ein früheres Projekt integriert wurden, wurde auch untersucht, wie die Vorgängergruppe diese Teile montiert hat. Zu diesem Zweck hat die Gruppe Komponenten mit Stelzen gefertigt, die in die Lochplattform des Fahrzeugs eingesetzt werden konnten.
Dieses System wurde unter anderem für die Stromverteiler verwendet. Für dieses Projekt wurde ein weiteres Teil nach gleichem Prinzip für die Step-Down-Module erstellt, um auch diese in gleicher Weise befestigen zu können.

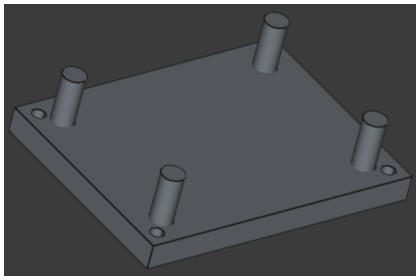


Abbildung 2.6: Halterung DC Step-Down mit Stecksystem

2.10. Halterungen Flywheel Motortreiber (Becker)

Des Weiteren wurde die Halterung für die Motortreiber der Flywheel-Motoren mit den gleichen Stempen ausgestattet. Die Halterung stellt eine modifizierte Version einer Vorlage [2] dar. Die ursprüngliche Konstruktion dieser Vorlage beinhaltete seitliche Schraublöcher, welche jedoch im Zuge der Implementierung des Stecksystems entfernt wurden.

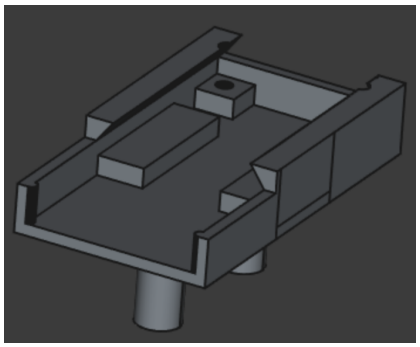


Abbildung 2.7: Halterung für Flywheel Motortreiber

2.11. Mikrocontroller-Case (Becker)

Die Mikrocontroller-Case bietet Platz für einen ESP-32 auf einem Freenove Breakout Board und für einen Raspberry Pi 5 mit aktivem Luftkühler. Beide Controller sind in diesem Fall übereinander angeordnet, da auf der Lochplattform ansonsten nicht genügend Platz zur Verfügung stehen würde, um alle weiteren erforderlichen Komponenten, wie beispielsweise jene für die Stromversorgung, unterzubringen. Aufgrund der Verwendung fester Kabel, wie beispielsweise des Kamerakabels für den Raspberry PI, und der allgemeinen Kabellänge müssen beide Controller in unmittelbarer Nähe zur Geschützplattform platziert werden.
Der ESP-32 ist im unteren Teil des Gehäuses untergebracht. Im ersten Entwurf wurde lediglich eine Vorlage [3] gedruckt. Es stellte sich jedoch heraus, dass es von dem Freenve-Steckbrett mehrere Varianten in unterschiedlichen Größen gibt. Die gewählte Vorlage erwies sich als zu klein, um unser konkretes Modell darin zu platzieren. Daher wurde auf Basis der Vorlage eine Version mit passenden Dimensionen erstellt. Darüber hinaus wurden einige Änderungen vorgenommen. So wurde ein zusätzliches Schraubloch hinzugefügt, um eine externe Antenne anzuschließen und somit die Bluetooth- und WLAN-Abdeckung zu optimieren. Im nächsten Schritt wurden im Deckel Kühllöcher in Form des Textes HimmelWacht integriert, um den Mikrocontroller mit zusätzlicher Luft zu versorgen.

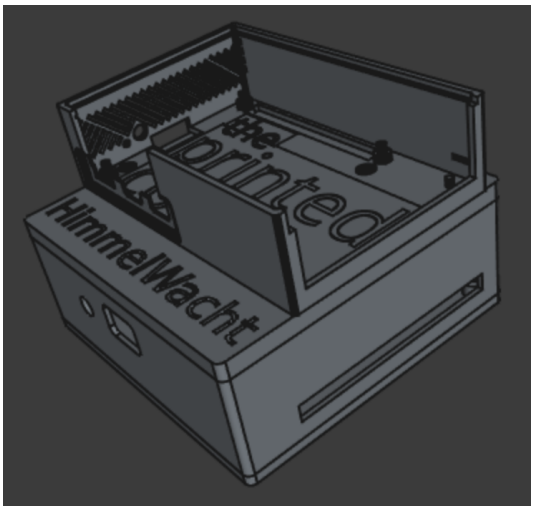


Abbildung 2.8: Mikrocontroller Case

Zuletzt wurde die Unterseite der Case noch mit dem modularen Stecksystem ausgestattet, welches auch für Teile der Stromversorgung verwendet wird. Die Montage kann somit ohne den Einsatz von Klebstoff oder Schrauben durchgeführt werden.

2.12. Halterung Lautsprecher (Becker)

Auch für den ursprünglich vorgesehenen Lautsprecher wurde eine Befestigung konstruiert. Die vorliegende Halterung wurde konzipiert, um sowohl den Lautsprecher als auch das zugehörige Verstärkerboard unterzubringen, womit kurze Kabellängen und eine damit einhergehend einfachere Verkabelung gewährleistet werden. Diese Halterung ist ebenfalls mit dem modularen Stecksystem ausgestattet.

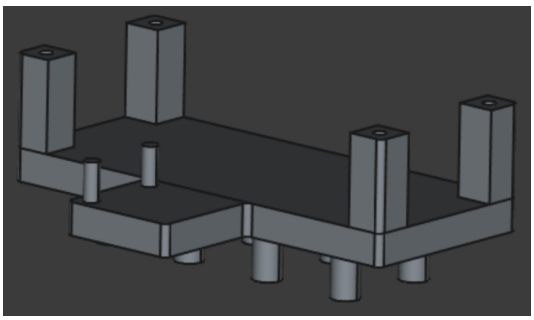


Abbildung 2.9: Lautsprecherhalterung

3. ESP32 Programmierung

3.1. Einführung (Becker, Specht)

Die Programmierung der zeitkritischen Funktionalitäten erfolgt unter Verwendung eines ESP-32, da bestimmte Funktionen strengere zeitliche Anforderungen aufweisen, als dies mit einem General-Purpose-Betriebssystem realisierbar wäre.
Nach Abwägung aller relevanten Faktoren wurde sich dazu entschieden, den klassischen ESP-32 Chip zu wählen, da dieser den Bluetooth Classic Standard unterstützt. Dieses Protokoll wird für die Ansteuerung des Dualshock4-Controllers benötigt. Darüber hinaus trägt das weite Verbreitungsgebiet dieser Serie dazu bei, dass eine Vielzahl an Material zur Verfügung steht, was wiederum die Fehlersuche erheblich vereinfacht.
Es wurde jedoch ausdrücklich darauf verzichtet, den Controller unter Verwendung des Arduino Frameworks zu programmieren. Die Ursache hierfür war insbesondere die Größe der entstehenden Kompilate. Es sei darauf hingewiesen, dass das System über zahlreiche Funktionalitäten verfügt, was eine begrenzte Speicherverfügbarkeit zur Folge hat. Daher erfolgte die Programmierung des Controllers unter Verwendung des *Espressif IoT Development Frameworks* (ESP-IDF) [4]. Dieses Framework stellt ebenfalls eine Vielzahl von Treibern für unterschiedliche Funktionen bereit, ist jedoch gezielt für Espressif-Controller konzipiert und zeichnet sich dadurch aus, dass es einen geringen Speicherplatzbedarf aufweist.
Die Programmierung erfolgte in der Programmiersprache C.

3.2. 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.

3.2.1. 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 abstrahierte 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.

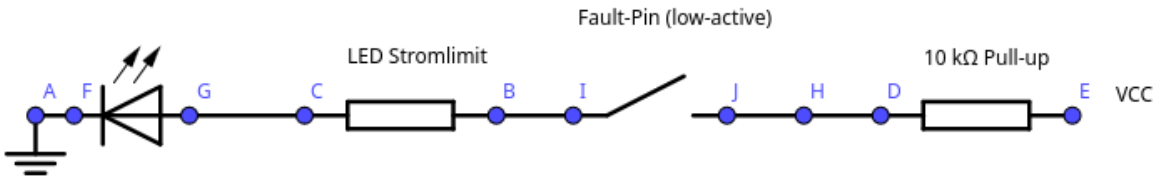


Abbildung 3.1: Testaufbau des Fault-Pins des Pololu-Motor-Treibers

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.

3.2.2. 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.

3.2.3. Differential Drive

3.2.4. Tests

3.2.5. Integration

3.3. MQTT-Anbindung (Specht)

3.3.1. WiFi-Stack

3.3.2. MQTT-Stack

3.4. Dualshock4 Treiber (Becker)

3.4.1. Übertragung

3.4.2. Version 1: eigens entwickelter Treiber

3.4.3. Version 2: Treiber basierend auf der Bluepad32 Bibliothek

3.4.4. Testing

3.5. Platformsteuerung (Becker)

Die sogenannte Plattformsteuerung bezeichnet alle technischen Komponenten, die erforderlich sind, um die Plattform in ihrer Rotation, vertikalen Neigung sowie für die Abgabe eines Schusses zu steuern.
Für den genannten Zweck werden folgende Komponenten benötigt:

- **Drehung und Neigung**
  - zwei MG996R Servo Motoren
- **Schussabgabe**
  - zwei DC-Motoren (Flywheels)
  - MG92B Servo Motor

Da die maximale Ausgangsstärke eines GPIO-PINs des ESP-32 mit 40mA [5, S. 53] für die benötigten Motoren nicht ausreicht [6–8], wurden entsprechende Treiberbaords verwendet. Konkret handelt es sich hierbei um ein PCA9685 PWM-Treiberboard für die Servomotoren und um per PWM steuerbare MOSFET-Module für die Flywheel-Motoren.
In der nachfolgenden Sektion wird der Entwurf des Codes erörtert, der erforderlich ist, um die genannten Teile anzusteuern.

3.5.1. PWM Board Treiber (Becker)

Das PCA9685 PWM-Treiberboard gestattet die gleichzeitige Anbindung von bis zu 16 Servomotoren. Für die Stromversorgung steht ein Eingang mit einer Spannung von 5 Volt zur Verfügung.
Die Steuerung des Boards erfolgt durch das Schreiben verschiedener Werte in Konfigurationsregister, wobei das I²C-Protokoll zum Einsatz kommt. Der vorliegende Treiber wurde aus der Portierung eines bereits bestehenden Treibers [9] entwickelt, welcher in der Programmiersprache C++ implementiert war. Es wurde bewusst nur die Funktionalität portiert, die für den Umfang des Projekts von Relevanz war. Der Treiber umfasst demzufolge lediglich drei Funktionen:

- **pca9685_init**: Die Funktion erhält die gewünschte Konfiguration für das Board (beispielsweise die Bus-Adresse, die SDA- und SCL-Ports für den I²C-Bus) und initialisiert den I²C-Bus. Im Anschluss registriert sie das Treiberboard und konfiguriert schließlich das Board mit der gewünschten PWM-Frequenz.

- **pca9685_set_pwm_on_off**: Mithilfe dieser Funktion besteht nun die Möglichkeit, einen Motor auf einem der 16 Kanäle zu steuern. Der Parameter *ON* ist eine 12-Bit Zahl beschreibt hierbei den Zeitpunkt in der Phase, an welchem der Ausgang auf 5 Volt geschalten wird. *OFF*, ebenfalls eine 12-Bit Zahl bezeichnet den Zeitpunkt, zu welchem der Ausgang wieder auf 0 Volt geregelt wird. Eine grafische Veranschaulichung ist in Abbildung 3.2 ersichtlich. Da für die Ansteuerung der Servo Motoren keine symmetrischen PWM-Signale benötigt werden, wird der Parameter *ON* im Folgenden immer den Wert 0 annehmen.

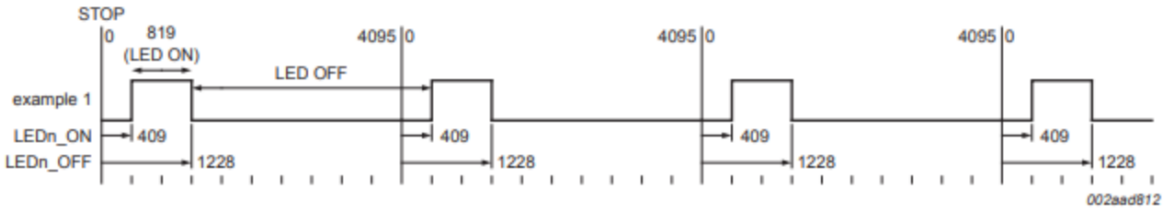


Abbildung 3.2: Erklärung ON/OFF Parameter für PCA9685 aus [10, S. 17]

- **pca9685_set_off**: Mittels dieser Funktion kann das PWM-Signal auf einem bestimmten Kanal deaktiviert werden.

Auf Basis dieses Treiber wurden im nächsten Schritt zwei Interfaces programmiert: Einerseits das Interface zur Plattform-Kontrolle und andererseits das Interface zur Schusskontrolle, welches zusätzlich der Logik zur Ansteuerung der Flywheel-Motoren enthält.

3.5.2. Ansteuerung der Servo-Motoren (Becker)

Um nun die Platform sowohl mit manuellen Modus mit dem Dualshock4 Controller, als auch im semi-automatischen Modus mit künstlicher Intelligenz über MQTT präzise Steuern zu

können, wurde nun ein Interface entwickelt, welches die Ansteuerung eines Motors an eine gewisse Position erlaubt.
Gearbeitet wird hier mit der Einheit Grad, ein Blick in das Datenblatt der Servo Motoren zeigt, das durch die Einstellung der Duty-Cycle Länge des PWN-Signals eine Drehung auf eine gewisse Grad-Position erreicht wird. Im ersten Schritt wurden nun durch manuelles Testen die *OFF* Werte für die Punkte $-90°$ (max. Drehung nach links), $0°$ und $90°$ (max. Drehung nach rechts) ermittelt. Der Wert für die Drehung auf $0°$ wird im Folgenden als $value_{zero}$ bezeichnet. Um nun den Zielwert $value_{off}$ für den *OFF* Parameter des PWM Board Treibers zu berechnen, wird Formel 3.1 verwendet:

$$
\begin{aligned}
&\text{Sei } \theta \text{ der gewünschte Winkel,} \\
&|\theta| = \text{Betrag von } \theta \\
&n_3 = \left\lfloor \frac{|\theta|}{3} \right\rfloor \\
&n_2 = |\theta| - n_3 \\
&s = 2 \cdot n_2 + 3 \cdot n_3 \\
&\tilde{s} = \begin{cases} s, & \theta > 0 \\ -s, & \theta \leq 0 \end{cases} \\
&value_{off} = value_{zero} + \tilde{s}
\end{aligned}
\tag{3.1}
$$

3.5.3. Ansteuerung der Flywheel Motoren (Becker, Koch, Wohlrab)

3.6. Integration (Becker, Specht)

4. Raspberry Pi Programmierung

4.1. Lautsprecher (Becker)

4.2. Gyrosensor Programmierung (Koch)

Zu Beginn des Projekts war geplant den Gyrosensor MPU6050 zu nutzen, um die Position der Geschützplattform zu bestimmen, da eine kontinuierliche Bewegung um die eigene Achse aufgrund der Kabel nicht möglich ist. Dabei sollte der Sensor über den I2C-Bus mit dem Raspberry Pi 5 verbunden werden, um die Daten auszulesen und zu verarbeiten. Der MPU6050 ist dabei ein 3-Achsen-Gyroskop und 3-Achsen-Beschleunigungssensor, welcher entsprechende Drehbewegungen und Beschleunigungen entlang der Raumachsen messen kann, welche für eine genaue Winkelbestimmung notwendig sind.
Nach kurzem Einlesen in die Dokumentation waren erste Rohdaten leicht auszulesen. Diese Rohdaten liegen in Form von 16 Bit in zwei Registern bereit und haben die Einheit LSB/g für die Beschleunigungswerte und $LSB/°s$ für die Gyroskop-Werte. Dies gilt es in tatsächliche physikalische Größen umzuwandeln, was bei unserem Projekt letztlich einem Winkel entspricht. Um die Beschleunigungswerte nutzen zu können, muss dafür mittels des Skalierungsfaktors die Fallbeschleunigung g errechnet werden, indem man den erhaltenen Wert $x/16384$ rechnet. Die 16384 ergeben sich aus der Dokumentation und entsprechen den LSB bei einem Messbereich von $\pm 2g$, welches der Standardauflösung entspricht und auch die höchste Auflösung des MPU6050 für ist. Ähnlich wird nun auch Winkelgeschwindigkeit ($°s$) errechnet. Hierbei beträgt der Teiler standardmäßig 131. [11, S. 12-13]
Nach der Umrechnung der Rohdaten in physikalische Größen können nun die Neigungswinkel (Roll- und Pitch-Winkel) des Sensors berechnet werden. Diese ergeben sich aus der Richtung der Erdbeschleunigung relativ zum Sensor.
Dazu wird die Erweiterung des Arkustangens genutzt, genauer gesagt die Funktion `atan2`, da sie im Gegensatz zum gewöhnlichen Arctangens auch die Orientierung in allen vier Quadranten berücksichtigt und somit stabile Winkelwerte über den gesamten Bereich von $-180°$ bis $+180°$ liefert. [12]
Die Winkelberechnung erfolgt nach der Formel 4.1:

$$\begin{aligned} \varphi_{\text{pitch}} &= \arctan 2\left(a_x, \sqrt{a_y^2 + a_z^2}\right) \cdot \frac{180°}{\pi} \\ \varphi_{\text{roll}} &= \arctan 2\left(a_y, a_z\right) \cdot \frac{180°}{\pi} \end{aligned} \tag{4.1}$$

Hierbei sind a_x, a_y und a_z die normierten Beschleunigungswerte in g-Einheiten (berechnet aus den Rohwerten durch Division durch 16384).
Die $atan2$-Funktion liefert den Winkel zwischen der positiven x-Achse und dem Punkt (x, y) in der Ebene, wodurch Sprünge oder Mehrdeutigkeiten bei 90° vermieden werden. [12]
Nachdem der Term im Code implementiert wurde, konnte ein starkes Rauschen beobachtet werden, was zunächst auf natürliche Schwankungen des Sensors zurückgeführt wurde, weshalb sich dazu entschieden wurde zuerst einen Komplementärfilter zu implementieren, welcher allerdings das Problem nur bedingt beheben konnte. Deshalb wurde auch noch der Kalman-Filter ausgetestet, wodurch auch eine starke Rauschunterdrückung festgestellt werden konnte, doch auch hier zeichnete sich ein überdurchschnittliches Rauschverhalten ab, weshalb der Fehler nicht mehr auf ein natürliches Rauschen zurückzuführen war. Daraufhin wurde auch ein zweiter MPU6050 getestet, welcher ebenfalls dieses Verhalten aufwies, wodurch klar wurde, dass es sich hierbei um einen Programmierfehler handeln muss. Dieser konnte nach einiger Zeit auch herausgefunden werden und lag an der Interpretation der Rohdaten, welche fälschlicherweise als `int16_t` statt `uint16_t` interpretiert wurden.
Aufgrund der aufgewendeten Zeit für die Implementierung des Kalman-Filters sollte dieser aber trotzdem Anwendung im Projekt finden und wird in 4.2.1 behandelt.
Wie Eingangs erwähnt, sollte der Gyrosensor MPU6050 genutzt werden, um die Position der Geschützplattform zu bestimmen, wofür der Winkel der Drehung um die eigene Achse benötigt wird. Dabei stellte sich heraus das der MPU6050 für diesen Wert zu einem starken Drift neigt, weshalb empfohlen wird diesen mit einem Magnetometer zu kombinieren [11, S. 26]. Zur gleichen Zeit stellte sich heraus, dass diese Funktionalität nicht benötigt wird, da über die Servo-Motoren bereits ein Nullpunkt definiert werden konnte, weshalb der Gyrosensor letztlich nur noch für die Neigung der Geschützplattform genutzt wird, um diese als Debug-Information auf dem Webserver 5 anzuzeigen.

4.2.1. Kalman Filter Implementierung (Koch)

Der Kalman-Filter ist ein Algorithmus zur Schätzung des Zustands eines dynamischen Systems. Er nutzt Messwerte mit einem mathematischen Modell, um aus verrauschten Daten optimale Schätzungen zu erzeugen und zu filtern, was insbesondere bei Sensoren mit Rauschen, wie dem MPU6050, von Bedeutung ist [13]. In der Abbildung 4.1 ist sowohl für den Roll, als auch den Pitch ein signifikanter Unterschied zwischen den Rohdaten und den gefilterten Daten zu erkennen. Die ersten Schwankungen der gefilterten Werte sind vermutlich darauf zurückzuführen, dass der Kalmanfilter mit wenig Referenzwerten nur ungenaue Schätzungen liefern kann, weshalb die Schätzungen auch erst mit steigender Anzahl an Werten genauer werden. Der Pitch konnte mithilfe des Kalmanfilters eine Rauschreduktion von 59.4% erzielen und für den Roll 37.4% bei jeweils 100 Testwerten.
Aufgrund der Art wie der Gyrosensor auf dem Geschützarm montiert ist, ist für die Neigung allerdings nur der Roll-Wert von Bedeutung.

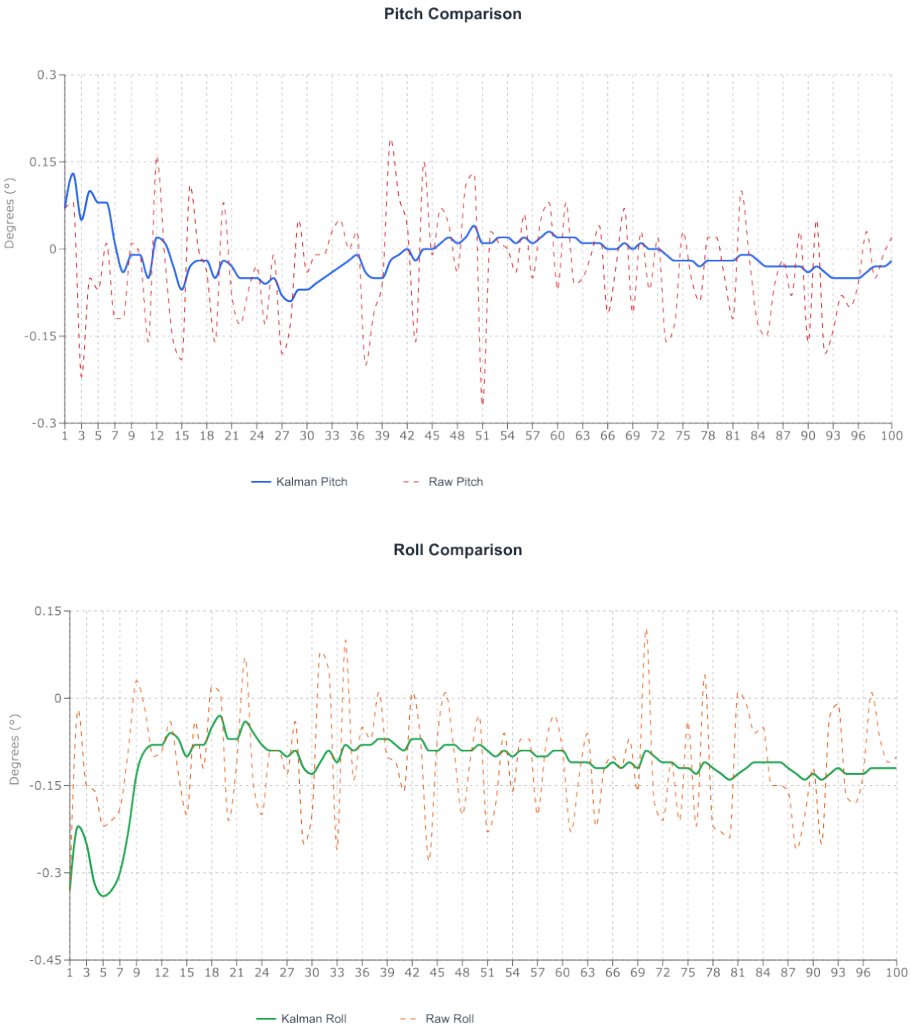


Abbildung 4.1: Vergleich der Rohdaten und Kalman-Filter geschätzten Werte für Pitch und Roll

5. Webserver (Koch)

5.1. Funktion und Anforderung des Webservers

Der Webserver des Projekts sollte die zentrale Schnittstelle zwischen verschiedenen Sensoren werden und sowohl Videomaterial mit bereits verarbeiteten KI-Informationen darstellen können und wissenswerte Daten wie die Entfernung des Fliegers oder die Geschützneigung anzeigen. Als zentrales Problem stand dabei das Videomaterial mit entsprechenden Bounding Boxes anzuzeigen im Vordergrund, da bereits über das ganze Projekt hinweg die KI die leistungsintensivste Aufgabe war und entsprechende Latenzprobleme bereits auf das Minimum verringert wurden. Der Grundlegende Aufbau der KI-Verarbeitung mitsamt der Videoausgabe ist in **??** erklärt.

5.2. Lösungsansatz und Umsetzung

Ein Lösungsansatz diesbezüglich war deshalb, dass die Sensordaten vom Raspberry Pi per MQTT an den PC geschickt werden, welcher die KI-Verarbeitung übernimmt, um die Daten direkt im Videostream abzubilden. Das Video mit den Bounding Boxes und den Sensordaten sollte dann zurück auf den MediaMTX-Server geschickt werden, worüber man dann das Video sehen kann. Dieser Ansatz war allerdings keine geeignete Lösung, da die Verarbeitung zu lange dauerte und somit eine zu geringe Framerate erreicht wurde, wodurch dies keine annehmbare Lösung darstellte.
Deshalb wurde der alternative Ansatz gewählt, dass der Raspberry Pi selbst den Webserver bereitstellt, da dieser bereits die Sensordaten und den MediaMTX-Server bereitstellt. Das Problem hierbei war jedoch, dass somit zwar der Videostream und die Sensordaten dargestellt werden konnten, allerdings noch keine Bounding Boxes, welche eine harte Anforderung an den Webserver waren. Es musste also eine Lösung gefunden werden, den verarbeiteten Videostream mit den Bounding Boxes vom PC zum Raspberry Pi zu übertragen, doch das senden des Videostreams ist nicht ohne Verluste möglich, wie bereits erwähnt wurde.
Die hierbei effizienteste Lösung war, die Koordinaten der Bounding Boxes in Form von JSON-Daten über einen Websocket zu übertragen, welche dann auf dem Raspberry Pi in den Videostream eingebettet werden.

Stundenliste

Name: Fabian Becker
Gruppe: 1 HimmelWacht

Pos.	Bezeichung	Beschreibung	Stunden in h
1	Vorbereitung	Gruppenbildung, Projektfindung, erste Recherche	4
2	Projektorganisation	Erstellung Projektbeschreibung, Terminplan & Plakat	8
3	CAD Design	Einarbeitung FreeCAD	5
4	CAD Design	Entwurf Platform, Anpassung Magazin & Schussarm	11
5	Ps4 Controller Treiber	Einarbeitung ESP-IDF, erster Entwurf mit Bluetooth HID	3
6	Ps4 Controller Treiber	Debugging PS4 Treiber aufgrund ESP Freeze	10
7	Ps4 Controller Treiber	Erstellung ESP32 Projekt mit Bluepad32 Dependencies	2
8	Ps4 Controller Treiber	Erstellung Custom Bluepad32 Plattform	4
9	Ps4 Controller Treiber	Implementierung Datenabgriff, Vibration, Farbwechsel	8
10	Ps4 Controller Treiber	Treiber auf Multiprozessor portiert	6
11	Ps4 Controller Treiber	Low Battery Warning durch rote blinkende LED	3
12	CAD Design	Design + Druck für Stromversorgung und Motortreiber	6
13	PWM Treiber	Portierung PCA9685 für Servosterung	4
14	Plattformsteuerung	Gradweise Ansteurung für Servomotoren implementiert	6
15	CAD Design	Redesign + Druck Schussarm und Magazin	5
16	Lautsprecher	Treiberboard gelötet und Test Lautsprecher	5
17	Flywheel Motoren	Test Flywheelmotoren + Implementierung Treibercode	5
18	Geschütz	Zusammenbau, Kalibrierung Servos	10
19	Geschütz	Integration PS4 Treiber	15
20	CAD Design	Druck & Design Mikrocontroller Cases	10
21	Stromversorgung	Finale Verkabelung ESP32	6
22	Gesamttests	Integriertes Fahrzeug getestet (Controller & Platform)	4
23	Plattformsteuerung	Implementierung Moduswechsel (manuel <-> KI)	2
24	Ps4 Controller Treiber	Tiefpassfilter zur Unterdrückung von Jitter implementiert	3
25	Abschlusspräsentation	Präsentationsfolien erstellt & Bilder aufgenommen	3
26	Objekttracking	Implementierung & Testen des Objekttrackings	3
27	Dokumentation	Erstellung der wissenschaftlichen Dokumentation	20
		Gesamt:	**171**

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

Literaturverzeichnis

[1] Elephant333. „Nerf Turret - Modular Arduino Tank (M.A.T),“ besucht am 1. Juli 2025. Adresse: https://www.thingiverse.com/thing:4870102/files.

[2] Higany. „Holder for XY-MOS D4184 Power MOSFET breakout module,“ besucht am 1. Juli 2025. Adresse: https://www.printables.com/model/355368-holder-for-xy-mos-d4184-power-mosfet-breakout-modu.

[3] A. Whizzbizz. „Case and camera clip for Freenove Breakout Board for ESP32 / ESP32-S3,“ besucht am 1. Juli 2025. Adresse: https://www.printables.com/model/723594-case-and-camera-clip-for-freenove-breakout-board-f.

[4] „Getting Started with ESP-IDF,“ besucht am 1. Juli 2025. Adresse: https://idf.espressif.com/.

[5] Espressif. „ESP32 Series Datasheet. “Adresse: https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf.

[6] „DC Toy / Hobby Motor - 130 Size. “Adresse: https://www.adafruit.com/product/711.

[7] „TowerPro MG92B Servo. “Adresse: https://servodatabase.com/servo/towerpro/mg92b.

[8] „TowerPro MG996R Servo. “Adresse: https://servodatabase.com/servo/towerpro/mg996r.

[9] J. Supcik. „ESP/IDF PCA9685 I2C PWM Driver. “Adresse: https://github.com/supcik/idf-pca9685.

[10] N. Semiconductors. „PCA9685 Product data sheet. “Adresse: https://cdn-shop.adafruit.com/datasheets/PCA9685.pdf.

[11] I. InvenSense. „MPU-6000 and MPU-6050 Product Specification. “Adresse: https://invensense.tdk.com/wp-content/uploads/2015/02/MPU-6000-Datasheet1.pdf.

[12] MathWorks. „atan2. “Adresse: https://www.mathworks.com/help/matlab/ref/double.atan2.html.

[13] R. Aachen. „Kalmanfilter. “Adresse: https://www.irt.rwth-aachen.de/global/show_document.asp?id=aaaaaaaacdvoafc.