Thema

Erweiterung bestehender Drohnen um eine Autonomflugfähigkeit

# Studienarbeit T3100

des Studienganges Eletrotechnik

an der Dualen Hochschule Baden-Württemberg, Stuttgart

von

Markus Rein

Abgabedatum: 22.01.2023

| | |
|---|---|
| Bearbeitungszeitraum | 21.10.22 – 22.01.2023 |
| Matrikelnummer, Kurs | 6983030, TEL20GR5 |
| Dualer Partner | Infineon Technologies, Neubiberg |
| Gutachter*in der Dualen Hochschule | Prof. Dr.-Ing. Johannes Moosheimer |

Erklärung

Ich versichere hiermit, dass ich meine Projektarbeit mit dem Thema: „Analyse der Anwendungsmöglichkeiten bei der Vermittlung von Datenpaketen zwischen Mikrocontrollern und Terminalcomputern“ selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.
Ich versichere zudem, dass die eingereichte elektronische Fassung mit der gedruckten Fassung übereinstimmt.

| ..... | ..... | ..... |
| --- | --- | --- |
| Ort | Datum | Unterschrift |

# Abstrakt

In diesem Projekt werden die softwareseitigen Möglichkeiten zur Datenübertragung im Kontext von Eingebetteten Systemen diskutiert. Benötigt wird eine generische Schnittstelle zur Steuerung verschiedener integrierter Geräte. Diese müssen mit Anweisungen versorgt werden und Daten zurück liefern.

Es soll ermöglicht werden, ausgehend von verschiedenen Programmierumgebungen Lösungen zu entwickeln und den Mehrfachaufwand der Implementation zu reduzieren.

Im Rahmen des Projektes werden mehrere Serialisierungs-Bibliotheken vorgestellt und getestet. Danach wird eine Architektur als Grundlage entwickelt mit deren Hilfe eine beispielhafte Implementation einer Kommunikationsanwendung durchgeführt wird.

Fokus liegt auf der ausführlichen Betrachtung der Kodierung von Daten mittels der Bibliotheken, da bei dieser Effizienz eine entscheidende Rolle für Eingebettete Systeme spielt.

# Abstract

Discussed in this project are possibilities with software for data transfer with embedded systems. Required is a generic interface for control of various integrated devices. These have to be impinged with instructions and they have to deliver data.

The project shall enable development from different framework but reduce additional efforts of development.

In the scope of the project multiple serialization-libraries are introduced and tested. Next an architecture is designed which is then used for an exemplary implementation of a communication-application.

Highlight of the project is the detailed reflection of data encoding with serialization with regard to efficiency in embedded systems.

# Kurzfassung

..... Short summary of the thesis ...

# Inhaltsverzeichnis

# Abbildungsverzeichnis

# Tabellenverzeichnis

# Verzeichnis der Listings

# Verzeichnis der Algorithmen

# Abkürzungsverzeichnis

**GCS** Ground Control Station.

**GPS** Global Positioning System.

**IMU** inertiale Messeinheit.

**LBA** Luftfahrt-Bundesamt.

**MAVLink** MAVLink.

**Pixhawk® 4** Microcontroller, mit Software und Peripherie zur Flugsteuerung.

**RAM** Arbeitsspeicher.

**Robot Operating System** Framework mit Schnittstellen für Sensor/Aktor-Verknüpfungen.

**RPI** Raspberry Pi (Einplatinencomputer).

**simultaneous localization and mapping** Simultane Lokalisierung und Kartierung der Umgebung.

**UAS** Unmanned Aircraft System.

**UAV** Unmanned Aerial Vehicle.

**UDP** User Datagram Protocol.

# 1 Problemstellung und geplantes Vorgehen

Ziel dieser Arbeit ist es, das Projekt „Erweiterung einer bestehenden Drohne um eine Autonomflugfähigkeit“ von Harald Wirth und Dominik Helfenstein, siehe [WH22a][WH22b], aufzuarbeiten und fortzuführen. Dazu steht eine Modelldrohne „Holybro S500“ zur Verfügung. Diese wurde zusätzlich mit einem Raspberry Pi (Einplatinencomputer) (RPI) ausgestattet, der eine Kommunikation mit der Drohne ermöglicht und zusätzliche Rechenaufgaben übernehmen kann. Dieser Aufbau wird verwendet, um Objekte in der Flugbahn der Drohne zu erkennen und auszuweichen. Bisher können der im Flug befindlichen Drohne über WLAN GPS-Zielkoordinaten zugespielt werden. Anschließend fliegt sie in Richtung des Ziels und verwendet währenddessen Ultraschall, um Objekte in nächster Umgebung zu detektieren. Die Umsetzung was anschließend passieren soll, ist allerdings noch nicht abgeschlossen. Es herrscht folgender Zustand:

- In den Tests flog die Drohne in Richtung eines ebenen Hindernisses. Sie hielt in sicherer Entfernung an und wurde automatisch gelandet. Bei weiteren Tests an natürlichen Strukturen (Büsche) konnten diese nicht erkannt werden und die Drohne musste manuell abgefangen werden.
- Für weitere Flüge soll das sog. Magneten-Prinzip zur Anwendung kommen (noch nicht implementiert). Bei diesem wird eine Kursänderung um sich im Weg befindliche Hindernisse durchgeführt. [schänes BILD zum Verständnis] Wird das Ziel erreicht, landet die Drohne an gegebenem Punkt. Für produktive Anwendungszwecke reicht dieses aber nicht aus denn eine Zielführung ist nicht garantiert.

Um das Projekt erfolgreich abzuschließen, wird eine verbesserte Hinderniserkennung und Routenplanung entwickelt. Dieses Projekt sieht vor, dass die Drohne sowohl statische als auch dynamische Hindernisse erkennt und als Karte dokumentiert. Somit wird Navigation durch unbekanntes Terrain ermöglicht. Weiterhin soll die Drohne gezielt Markierungen ansteuern können.

Im ersten Teil dieser Arbeit wird das bestehende Projekt tiefgründig analysiert. Grundlegende Konzepte werden aufgegriffen und erweitert. Bestandteil hiervon ist das erfolgreiche Anwenden der Tests und Funktionalität des ersten Projektes. Dazu wird die Software angepasst und auch „Robot Operating System“, was Vorgesehen war aber nie zum Einsatz kam, verwendet um eine Basis für weitere Entwicklungen zu schaffen.
Ein weiterer Meilenstein ist es, die Sicherheit für weiteres Vorgehen zu schaffen. Sämtliche

Software soll ab diesem Punkt im Simulator getestet werden, um Ausfälle und Schäden zu vermeiden. Auch soll die Software zur Hindernis- und Zielerkennung im Simulator getestet werden.

Der zweite Teil sieht vor, ein Kamerasystem zu installieren, um Hindernisse und Ziele zu erkennen. Dieses kommuniziert mittels ROS mit der Drohne selbst und steuert diese so. Die Software der Drohne dokumentiert eine notwendige grundlegende Vorgehensweise und wird als eine Basis für eigene Entwicklungen benutzt.
Für die Erkennung von Hindernissen ist ein schneller (echtzeitfähiger), zuverlässiger Algorithmus notwendig. Es sind umfangreiche Tests notwendig.
Zuletzt wird der Navigationsalgorithmus der Drohne erweitert und angepasst.

# 2 Grundlagen zum Themenbereich „autonomer Drohnenflug“

> Ein UAS besteht aus einem unbemannten Luftfahrzeug und dessen Ausrüstung. Die Steuerung kann entweder von einem Menschen oder von einem integrierten oder ausgelagerten Computer durchgeführt werden. Dadurch kann eine Drohne auch ein teil- oder vollautonomes Luftfahrzeug darstellen.
>
> *(Bundesministerium für Digitales und Verkehr[1])*

Der englische Fachbegriff für Drohnen lautet „Unmanned Aircraft System (UAS)“. Er stammt vorrangig aus dem militärischen Bereich. Heutzutage wird vermehrt der Begriff „UAV“ verwendet, so auch fortwährend in dieser Arbeit. Aus dem Modellbau sind UAV in Form von fernsteuerbaren Flugzeugen, Helikoptern und Multicoptern bekannt. Drohne wird häufig als Synonym für letztere genutzt. Eine häufige Anwendung spielt dabei der autonome Flug, bei dem Anweisungen vom Bediener kommen, die automatisiert durchgeführt werden. So kann eine Drohne auch zu gewerblichem Einsatz kommen, bspw. in den folgenden Bereichen: Landwirtschaft, Wettervorhersage, Vermessung, Bevölkerungsschutz und Transport.

Zur Einstufung des automatisierten Verhaltens von Drohnen definiert die European Cockpit Association AISBL 6 „Level“ (Klassen), veranschaulicht in Grafik 2.1. Verfügbare UAV implementieren bereits automatische Funktionen der unteren Level, wie bspw. Regelung zur Flugstabilität und dem Halten einer Flugrichtung. Interessant sind die oberen Level, beginnend bei Level 3:

Level 3 - Bedingte Automation: Das UAV kann bestimmte Teilaufgaben in definierter Umgebung selbsttätig durchführen. Ein Pilot muss immer anwesend sein, um im Störungsfall einzugreifen.

Level 4 - Hohe Automation: Die gesamte Flugsteuerung und Ausfallsicherung wird eigenständig durchgeführt. Kein Pilot wird benötigt, jedoch muss das Verhalten des UAV überwacht werden.

Level 5 - Vollständige Automation (Autonomie): Eigenständige Flugplanung von Start über Flugstrecke bis Landung. Mit derzeitigen Mitteln nicht realisierbar[2].

[1] https://www.dipul.de/homepage/de/informationen/allgemeines/was-ist-eine-drohne

[2] siehe https://twitter.com/elonmusk/status/1411280212470366213

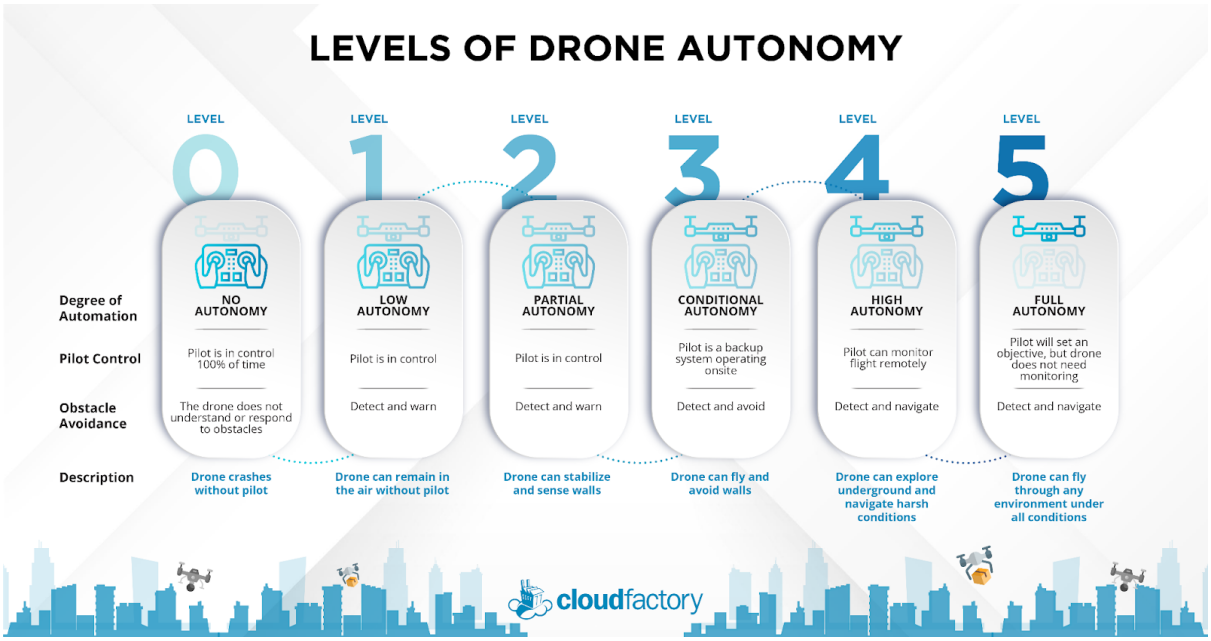



**Abbildung 2.1:** Einteilung Automation von UAV: Vom niedrigsten Level (0) zum höchsten Level (5) steigt der Automationsgrad (https://blog.cloudfactory.com/levels-of-drone-autonomy).

Das Luftfahrt-Bundesamt (LBA) erlässt zudem Regelungen, wonach ein autonomer Flug ‚[...] derzeit nicht zulässig [ist]; Fernpiloten müssen jederzeit eingreifen können. [...]‘[3].

Der bisherige Stand des Projektes steht im Übergang von Automationsklasse 3 auf Klasse 4. Allerdings fehlen generalisierte Anwendungsfälle (später gezeigt) sodass die Drohne noch nicht wie gewünscht, Hindernissen ausweicht (das bisherige Verhalten beschränkt sich auf sicheres Landen beim Erkennen von Hindernissen).

## 2.1 Die Drohne und ihr Umfeld

Aus der Ausarbeitung erschloss sich, dass nur ein Drohnen-Bausatz zur Umsetzung des Projektes genutzt werden kann. Ein Bausatz kann flexibel mit zusätzlichen Sensoren und Bordcomputer erweitert und programmiert werden. Der bereitgestellte Aufbau wird vereinfacht in Bild 2.2 dargestellt. Die wichtigsten Bestandteile sind:

**Flugcontroller:** Kontrolle der originalen Drohne
**RPI:** Bereitstellen von WLAN-Netzwerk und Verbindung mit Flugcontroller
**Ultraschallsensoren:** Erfassen von Hindernissen in der Umgebung

[3] https://lba-openuav.de/onlinekurs/lehrmaterial/allgemeine-uas-kunde/kontrolle-und-flugmodi/

**Bodenstation:** Laptop/Smartphone der Flug steuert und Drohne parametriert; zum Einsatz kommt die Software QGroundControl, siehe [WH22a, Kapitel 4.3.5]

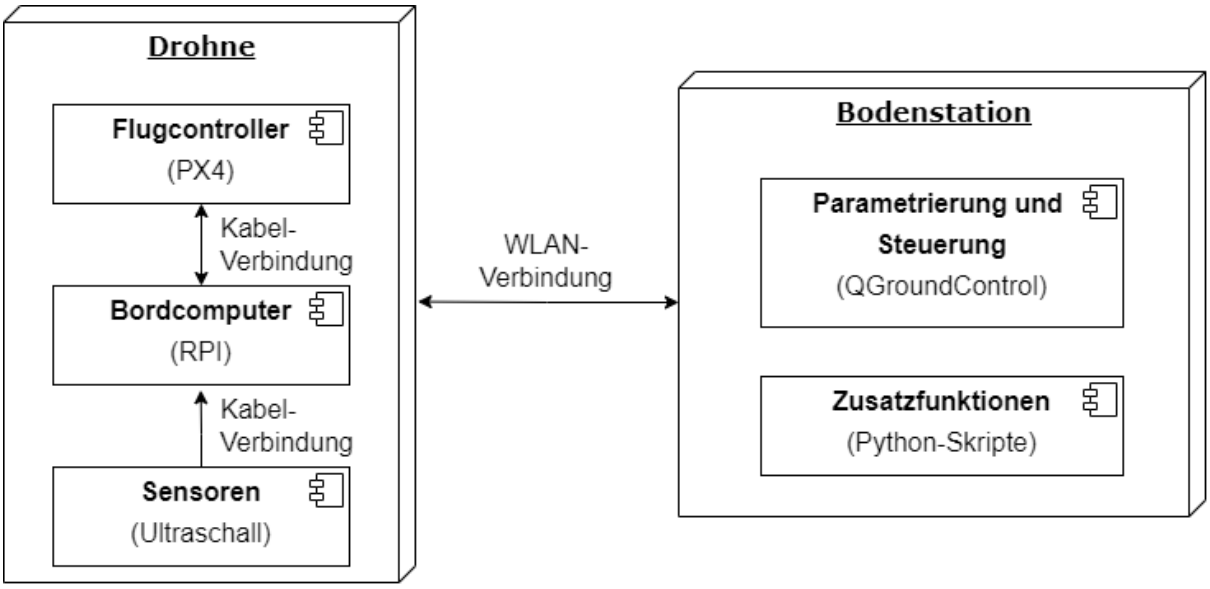



**Abbildung 2.2:** Gesamtbild Zusammenwirken der Bauteile der Drohne

Zur Entwicklung wurde das Modell Holybro S500 ausgewählt, welches in dieser Arbeit weiterhin verwendet wird. Der Quadcopter hat folgende Eigenschaften:

- Gewicht: 935g, Gesamtgewicht mit Akku, Sensoren und Bordcomputer: ?
- Traglast: 1,8kg
- Akku: Typ - LiPo; Nennspannung - 14,8V; Kapazität - 5000mAh; Capacity-Racting: 10C; Gewicht - ?
- Flugcontroller: Pixhawk® 4
- Sensoren: Gyroskop, Beschleunigung, Kompass, Barometer, GPS, Ultraschall

Um den autonomen Flug planen zu können, wären mögliche Fluggeschwindigkeit und Flugdauer notwendig. Dazu ist keine Angabe vom vorhergehenden Projekt bekannt, die Werte werden in diesem Projekt ermittelt.

Um die Modelldrohne Holybro S500 Fliegen zu dürfen, bestehen in Deutschland mehrere Vorraussetzungen[4]:

- Drohnenführerschein: Unterkategorie A2 (UAS bis $4kg$, $5m$ Abstand zu Menschen, kein Überfliegen von Menschenansammlungen, nur auf Sichtweite fliegen)
- Höhenmesser: die maximale Flughöhe von $120m$ darf nicht überschritten werden
- Fernidentifizierung: Pflicht zum Mitführen eines ADS-B Transmitters[5]

[4] https://lba-openuav.de/onlinekurs/lehrmaterial
[5] https://de.wikipedia.org/wiki/Automatic_Dependent_Surveillance

- Haftpflichtversicherung: Pflicht zum Versichern und Kennzeichnen der Drohne
- „Filmen, fotografieren oder das Anfertigen von Tonaufnahmen ist verboten, wenn: [...] die Aufnahmen für Gesichtserkennung oder andere automatisierte Prozesse verwendet werden“

Somit der Einsatz der Drohne im öffentlichen Raum streng untersagt. Tests beschränken sich auf Flüge nahe am Boden, fernab der Zivilisation.

## 2.2 Peripherie an der Drohne

Im Flugcontroller integriert sind mehrere Sensoren (Beschleunigung, Gyroskop, Magnetometer, Barometer). Neben diesen besitzt das Modell einen dedizierten Global Positioning System (GPS)-Empfänger. Nachfolgend ist dargestellt, wie diese Sensoren ausgewertet werden können.

### 2.2.1 Kommunikation mit dem Flugcontroller

Der Flugcontroller kann als eigenständige Einheit betrieben werden. Dabei kann die Drohne mittels Fernsteuerung gesteuert werden. In diesem manuellen Betrieb ist die Drohne nicht programmierbar.
Für zusätliche Funktionen stellt der Flugcontroller einen USB-Port und mehrere Telemetry-Ports bereit. Alle Daten zu und vom Flugcontroller werden mit dem MAVLink (MAVLink)[6]-Protokoll übertragen. Die Daten können von Bodenstation oder entsprechenden Anwendungen ausgewertet werden. Mit der USB-Verbindung kann eine direkte Verbindung zu einem PC hergestellt werden. Somit lässt sich die Drohne einrichten, allerdings muss die Kabelverbindung zum Flug getrennt werden. Auch wird zur allgemeinen Flugsteuerung davon abgeraten die USB-Schnittstelle zu verwenden, da es bei der Datenübertragung zu Verzögerungen kommen kann.
Der auf die Drohne montierte RPI kommuniziert mit dem Flugcontroller über den Telemetry-Port. Hierbei wird eine UART-Verbindung (bidirektionale Kommunikation über je eine Signalleitung) verwendet, da diese im Gegensatz zu USB einfacher zu programmieren ist und verzögerungsfrei arbeitet.
Das MAVLink-Protokoll zeichnet sich durch eine allgemein hohe Effizienz (geringer Overhead) aus. Nachrichten werden nach Themengebieten (z.B. Nachrichten vom Typ *HIGHRES_IMU* liefern Sensordaten) geordnet. Somit lässt sich eine bestimmte Eigenschaft von Interesse auslesen bzw. einspeisen. Um verlässliche Resultate beim Übertragen der Daten zu erhalten, enthalten so gut wie alle Nachrichten den Zeitpunkt des Absendens.
Der Flugcontroller kann konfiguriert werden nur Nachrichten bestimmter Typen zu versenden, um die Rechenlast zu verringern und höhere Übertragungsraten zu erlauben. Zu den möglichen

[6] https://mavlink.io/en/

Einstellungen zählen bspw. *Normal* (Verbindung zu Bodenstation) oder *Onboard* (Verbinfung zu mitgeführtem Computer) wobei ca. 40 respektive 50 Nachrichtentypen versendet werden. Wird die Einstellung *Normal* verwendet, so werden keine Nachrichten vom Typ *HIGHRES_IMU* versendet. Außerdem wird für jeden Nachrichtentyp die Frequenz festgelegt, sie beträgt bspw. für die *HIGHRES_IMU* $50Hz$.

### 2.2.2 Sensoren des Flugcontrollers

Ausgelesen vom Flugcontroller werden die Sensoren der *HIGHRES_IMU*. Dabei werden Messpunkte mit hoher Frequenz und Auflösung bereitgestellt. Eine Verarbeitung muss die Daten filtern um weitere Berechnungen durchführen zu können. Zur Verwendung kommt Robot Operating System,

### 2.2.3 Navigation mittels GPS

Die Navigation der Drohne erfolgt vorrangig über GPS. Mit solchem System können die aktuelle Position und Geschwindigkeit festgestellt werden. Für ein reales System muss ein eventueller Ausfall der GPS-basiertern Navigation in Betracht gezogen werden, denn GPS setzt klare Sicht zum Himmel und eine Verbindung zu mindestens 4 Satelliten voraus. [Quelle]

Fällt das GPS aus, ist die Drohne jedoch noch nicht völlig blind. Beschleunigungssensor, Gyroskope und Kompass bilden eine inertiale Messeinheit (IMU), mit deren Daten mittels eines *Extended Kalman Filter* die ungefähre Position berechnet werden kann. Das Ergebnis kann durch das Überlagern der Messwerte mehrerer Sensoren verbessert werden, bspw. indem sich an einem Kamerabild mit den bereits bekannten Abständen zu Objekten orientiert wird.

Ein weiteres Problem ist die totale Abhängigkeit von GPS in Bezug auf Zielfindung. Selbst wenn die Drohne direkt über dem Zielpunkt fliegt, aber GPS ausfällt, kann sie keine Erfolg verzeichnen. Für die Flugplanung wäre folglich ein zweiter Sensor von notwendig, der ohne GPS zum Zielpunkt navigieren kann. Für das Aufwinden des Zieles stehen mehrere Technologien zur Verfügung:

**Ortung** eines Sender (Ultraschall oder Infrarot) welcher durchgängig Signale sendet. Die Drohne bewegt sich dann immer auf den Sender zu.

**Erkennung** einer Struktur oder Bildes (Marker) mittels Kamera.

### 2.2.4 Hinderniserkennung durch Ultraschallsensoren

Mit einem Ultraschallsensor kann der Abstand zu Objekten gemessen werden.

Das Prinzip der Ultraschallortung besteht aus 3 Schritten:

1. Aussenden des Messimpulses. Daraufhin generiert der Sensor Ultraschall-Impulse.
2. Empfangen der Messimpulse. Der Sensor setzt ein Signal wenn ein Echo empfangen wird.
3. Auswerten der Signallaufzeit. Die Impulse bewegen sich mit Schallgeschwindigkeit im Medium zum Messobjekt und zurück.

Verwendet werden Ultraschallsensoren an der Drohne, welche Messimpulse in die Umgebung aussenden. Für eine exakte Auswertung der Signallaufzeit müssten Umgebungstemperatur, Luftdruck und Luftfeuchte bekannt sein. Den größten Einfluss spielt dabei die Temperatur. Druck und Feuchte wirken nur mit jeweils maximal auf das Ergebnis ein 5% bzw. 2%.
Die Formel zur Bestimmtung der Entfernung eines Objektes lautet ist gegeben in Formel 2.2. Benötigt wird die Schallgeschwindigkeit $c_{20}$ bei $20C$ und der Temperaturkoeffizient $\alpha_{20}$[Gru22, Seite 152].

$$s = \frac{1}{2} c \cdot t \tag{2.1}$$

$$= \frac{1}{2} c_{20}(1 + \alpha_{20}(\vartheta - 20)) \tag{2.2}$$

## 2.3 Bilderkennung und „Obstacle Avoidance“

Mannigfaltige Prinzipien fallen in den Bereich der Bilderkennung. Der Fokus dieses Projektes, das Erkennen und Ausweichen von Hindernissen wird „Obstacle Avoidance“ genannt. Es ist nicht zu verwechseln mit „Obstacle Detection“, dem Erkennen und Klassifizieren von Bildinhalten. Nachfolgend vorgestellt werden Techniken die hier zum Einsatz kommen könnten.

**SLAM Algorithmus** Simultaneous localization and mapping Techniken entstanden bereits in den 1980-1990 Jahren und wird bspw. bei Robotern eingesetzt, die in Hallen navigieren (kein GPS verfügbar). Dazu kommen mit Kamerasysteme in Verbindung mit Entfernungssensor (Sonar, Radar, Lidar) zur Verwendung . Die Ergebnisse von simultaneous localization and mapping können nicht garantiert werden und sind nicht reproduzierbar, weshalb es in keinen kritischen Umgebungen (bspw. wenn Verletzungsrisiko besteht) eingesetzt werden kann.

Allgemein wird simultaneous localization and mapping durch einen modularen Prozess beschrieben:

**Lokalisierung:** per Motorfortschritt, IMU, Kamera, etc.
Bei vsimultaneous localization and mapping kommen folgende Prinzipien zum Einsatz:

- Markov-Lokalisierung: Wahrscheinlichkeit des Aufenthaltsortes wird angenommen und über Zeit verfeinert. Iterativ, Ressourcenaufwendig.
- Kalman-Filter: Ermöglicht basierend auf Sensordaten schnelles wiederfinden aktueller Position. Anfällig bei Verlust von Kamerabildern

- Monte-Carlo-Lokalisierung: auch Partikelfilter genannt, nimmt Wahrscheinlichkeiten für jeden Ort an. Genauer als Markov, lineare Komplexität. Weniger Speicher als Kalman. Nachteil: Stillstand ohne sich ändernde Sensordaten

**Messung:** per Reichweite, Marker in Umgebung

**Visual SLAM** Eine Sonderform des simultaneous localization and mapping ist Visual glsslam (vSLAM). Bei diesem werden ausschließlich Kameras zur Erkennung der Umgebung eingesetzt. Der Algorithmus verwendet zusätzliche Sensordaten um die Bewegung der Kamera in die Berechnung der Position einzubeziehen.

**Stereokamera** Verwendet mehrere Kameras aus parallelverschobenen Bildern Tiefeninformationen zu gewinnen., also Abstand zu Punkten im Bild zu erkennen.

### Optical Flow

# 3 Verfügbare Technologien

Im Kapitel werden verfügbare Technologien für den autonomen Drohnenflug vorgestellt und ausgewertet. Dabei wird auf verfügbare Hard- und Software Lösungen eingegangen.

Was ist derzeit in Drohnen implementiert?

Wie macht es DJI.

Intel Kamers

[TODO: das hier benutzen https://github.com/PX4/PX4-Avoidance]

# 4 Aufarbeitung bestehendes Drohnenprojekt

In diesem Kapitel wird das bestehende Projekt aus nachvollzogen und überarbeitet. Inhalte sind das Testen bestehender Vorlagen aus [WH22a] und [WH22b], sowie das Aufsetzen einer neuen Arbeitsumgebung wie in beschrieben in [HD22], siehe **??**.

## 4.1 Inbetriebnahme

Zur Aufarbeitung des Projektes steht ist die Drohne bestehend aus Rahmen, Flugcontroller, Motoren mit Propellern, Ultraschallsensoren und Akku bereit. Nicht vorhanden sind die RC-Fernsteuerung und der auf die Drohne montierte RPI. Ursprünglich verwedet wurde ein RPI 4 Model B mit $2GB$ Arbeitsspeicher (RAM) als Bordcomputer, der mit dem Flugcontroller kommuniziert, dessen Daten über ein Netzwerk verbreitet und auch Anweisungen zum Flug geben kann. Gleiches kann auch mit dem RPI 3 Model B+ bewerkstelligt werden. Dieser bestitzt im Gegensatz nur 1GB RAM und weniger Rechenleistung. Die gelieferte SD-Karte enthält das Betriebssystem Raspbian OS in der Ausführung als 32-bit Betriebssystem. Sowohl RPI Model 4 als auch Model 3 besitzen zwar 64-bit Prozessoren, allerdings wird bei bis zu 4GB RAM das 32-bit Betriebssystem empfohlen um diesen effektiver auszunutzen. Der verwendete RPI startet somit problemlos. Beim Systemstart wird automatisch der WLAN-Hotspot aktiviert und der MAVLINK-Server (mavlink-routerd) gestartet.

#### Verbindung Flugcontroller mit RPI

Durch das automatische Setup des RPI sollte die Drohne mit dem Einstecken des Stromes bereit zum Flug sein. In [WH22a, Kapitel 4.3.6] ist der erste Schritt beschrieben als "Test der Verbindung". Der RPI wird mit dem Flugcontroller per Serieller Schnittstelle verbunden (siehe auch Abschnitt 2.2.1). Dazu wurde eigens ein Kabel entwickelt, welches die UART-Pins des RPI mit dem TELEM2-Port des Flugcontrollers verbindet. Es ist zu beachten, dass keine Dokumentation bezüglich der Belegung der Kabeladern (per Jumper an den RPI zu Stecken) gegeben ist. Ein Ausmessen des Kabels ergab:

- Schwarz: Masse -> Pin 06 des RPI
- Braun: UART-Receive (Rx) des Flugcontrollers -> Pin 08 (Tx) des RPI

- Weiß: UART-Transmit (Tx) des Flugcontrollers -> Pin 10 (Rx) des RPI

Anschließend soll die Verbindung wie in [WH22a, Kapitel 4.3.6] beschrieben, die MAVLINK-Konsole geöffnet werden. Jedoch ist das Programm *mavproxy* auf dem aktuellen Image nicht vorhanden. Um das weitere Vorgehen zu konkretisieren wurde das Programm nachinstalliert[1]. Das Starten des Konsolenprogrammes ist nur möglich, wenn der MAVLINK-Server (Prozess *mavlink-routerd*) nicht läuft (ansonsten ist die Serielle Schnittstelle blockiert). Mit dem Befehl: `mavproxy.py --master=/dev/serial0 --baudrate=921600` kann schließlich die Kommunikation des RPI mit der Drohne verifiziert werden.

Für das weitere Vorgehen wird das allzeit vorliegende Programm *mavlink-routerd* verwendet. Es arbeitet als Server im Hintergrund und verbreitet MAVLINK-Nachrichten im Netzwerk. Auch *mavproxy* stellt solch eine Funktionalität bereit, allerdings wird aufgrund von zu erwartenden Leistungseinbußen davon abgeraten[2]. Die Konfiguration des *mavlink-routerd* wird wie in 4.1 dargestellt angepasst. Dadurch kann sich jedes Netzwerkgerät (auch der RPI selbst) über das User Datagram Protocol (UDP)-Protokoll mit dem Flugcontroller verbinden. Auch kann ein PC/ Smartphone eine Verbindung zum Flugcontroller über das Netzwerk herstellen. Mit der neuen Konfiguration steht sowohl ein TCP-Server, als auch ein UDP-Server zur Verfügung. Eine Auswertung zur Verwendung des geeignetsten Protokolls folgt in Abschnitt 4.1. In Bild 4.1c ist das Vorgehen zur Verbindung über einen Netzwerkknoten und das erneute Ausführen des Funktionstests dargestellt.

### Verbindung Flugcontroller mit PC

Als Ground Control Station (GCS) zur Kommunikation mit dem Flugcontroller wird das Programm QGroundControl, wie in [WH22a, Kapitel 3.4] vorgeschlagen, verwendet. Mit diesem können beliebige Drohnen konfiguriert, parametriert und geflogen werden. Der Flugcontroller kann dazu per Micro-USB mit dem PC verbunden werden und das Programm findet selbigen automatisch.

Per Knopfdruck kann hier die Funktion *arm* ausgeführt werden und die Propeller beginnen zu Drehen. Das Programm stellt sogleich fest, dass keine Fernsteuerung verbunden ist und verfällt in den *manuellen Modus*. Nach einigen Sekunden wird die Drohne automatisch wieder *disarmed* und die Propeller gestoppt um Schäden zu vermeiden.

Um den Flugcontroller über das drahtlose Netzwerk des RPI anzusteuern, muss das Programm QGroundControl manuell konfiguriert werden. Dazu muss eine Einstellung unter *Comm Links* vorgemerkt werden, in welcher die IP-Adresse des RPI und der *TcpServerPort* des MAVLINK-Servers (siehe Bild 4.1b) eingepflegt werden.

[1] https://ardupilot.org/mavproxy/docs/getting_started/download_and_installation.html#mavproxy-downloadinstalllinux

[2] https://ardupilot.org/mavproxy/docs/getting_started/forwarding.html

```
pi@raspberrypi:~ $ cat /etc/mavlink-router/main.conf
[General]
TcpServerPort=3000
#5760
ReportStats=true
MavlinkDialect=common
DebugLogLevel=info

[UartEndpoint DROHNEuart]
Device = /dev/serial0
Baud = 921600


[UdpEndpoint DROHNEudp]
Mode = Normal
Address = 192.168.4.12
Port = 14550
```

**(a)** Alte Konfiguration von `mavlink-routerd`

```
pi@raspberrypi:~ $ cat /etc/mavlink-router/main.conf
[General]
TcpServerPort=5760
ReportStats=true
MavlinkDialect=common
DebugLogLevel=info

[UartEndpoint DROHNEuart]
Device = /dev/serial0
Baud = 921600

[UdpEndpoint DROHNEudp]
Mode = Normal
Address = 0.0.0.0
Port = 14550

[UdpEndpoint DROHNEserver]
Mode = Server
Address = 0.0.0.0
Port = 14560
```

**(b)** BILD FALSCH, das ganze funktioniert anders! Neue Konfiguration von `mavlink-routerd`

Verbindung herstellen

Drohne aktivieren

Drohne deaktivieren

```
pi@raspberrypi:~ $ mavproxy.py --master=udp:127.0.0.1:14550
Connect udp:127.0.0.1:14550 source_system=255
Log Directory:
Telemetry log: mav.tlog
Waiting for heartbeat from 127.0.0.1:14550
MAV> Detected vehicle 1:1 on link 0
online system 1
MANUAL> Mode MANUAL
fence breach
Received 848 parameters
Saved 849 parameters to mav.parm
arm throttle
MANUAL> Got COMMAND_ACK: COMPONENT_ARM_DISARM: ACCEPTED
AP: Failsafe mode activated
ARMED
UNKNOWN> Mode UNKNOWN
Arming checks disabled
disarm
UNKNOWN> Got COMMAND_ACK: COMPONENT_ARM_DISARM: ACCEPTED
AP: Disarmed by external command
AP: Failsafe mode deactivated
DISARMED
MANUAL> Mode MANUAL
```

**(c)** Verbindung mittels `mavproxy` auf dem RPI

**Abbildung 4.1:** Konfiguration und Verwendung von `mavlink-routerd`

Mithilfe des Programmes QGroundControl kann eine Drohne weiterhin ohne RC-Fernsteuerung geflogen werden. Es stehen Werkzeuge zur Planung eines „autonomen Fluges“ bereit[3]. Dazu lassen sich GPS-Wegpunkte auf einer Karte anlegen, die die Drohne später selbstständig abfliegt. Außerdem stellt QGroundControl Funktionen zur Verfügung, eine Drohne mittels Joystick (virtuell oder per angeschlossenem Konsolen-Controller) zu steuern[4].

#### Benchmark zur Verbindung von PC zum Flugcontroller

Zur Durchführung des Projektes steht keine physikalische RC-Fernsteuerung zur Verfügung, sondern es können nur Wegpunkte gesetzt oder ein Konsolen-Controller am PC verwendet werden. Für letzteren Fall ist es entscheident, dass die Daten über das mavlink-Protokoll zum Flugcontroller gesendet werden. Dabei ist die Verbindung über das Netzwerk des RPI langsamer als die Verwendung einer echten Fernsteuerung. Um trotzdem die optimale Reaktionsfähigkeit zu erreichen, soll verglichen werden, ob die Verbindung über TCP-Protokoll oder UDP-Protokoll schneller ist[5]. Die entscheidenden Kriterien sind:

- Latenz: Reaktionsfähigkeit des Flugcontrollers
- Bandbreite: Auslastung des Datenkanals

[TODO]

### 4.1.1 Einbindung von Funktionalität mittels Docker-Containern

Weitere Funktionen der Drohne sollen mit Robot Operating System bereitgestellt werden. Da dieses nicht auf dem Betriebssystem des RPI lauffähig ist, wird es als Docker Containern bereitgestellt.

Aufbauend auf den Projekten [WH22a], [WH22b] sollen bestehende Docker-Container wiederverwendet werden. Eine erste Inspektion mit `docker images` zeigt, dass lediglich das `hello-world` Beispiel bereits heruntergeladen wurde und im Programmspeicher bereit liegt. Weiterhin zeigt der Befehl `docker ps -a` dass auch nur dieses Beispiel bisher ausgeführt wurde. Im bestehenden Quellcode enthalten sind verschiedene Tests. Schon der Test 1 importiert einen Container aus „arm64v8/ros:galactic“. Das besagte Abbild ist für eine 64-bit ARM Architektur gedacht und damit auf dem gelieferten Betriebssystem nicht lauffähig. In weiteren Tests 3 und 4 wurde dieser Mangel erkannt und versucht zu korrigieren.

**Test 1** dient der Demonstration von ROS im Zusammenspiel mit Docker und wird in dieser Arbeit nicht betrachtet, da er keine erkennbare Funktion erfüllt.

[3] https://docs.qgroundcontrol.com/master/en/PlanView/PlanView.html
[4] https://docs.px4.io/main/en/config/joystick.html
[5] https://stackoverflow.com/questions/47903/udp-vs-tcp-how-much-faster-is-it

**Test 3** betrachtet das Zusammenwirken mehrer Container über Netzwerkenpunkte. Das Sende- und Empfangsverhalten wurde untersucht um möglichst effizienten Datenaustausch bereitzustellen. Der Test soll mit dem Befehl `docker-compose up` gestartet werden. Zum Zeitpunkt der Ausarbeitung dieser Arbeit schlägt dies erst einmal fehl, denn das vorgesehene Ubuntu Image für den Server („ubuntu:impish“, zu finden in Zeile 1 in ßerver/Dockerfile") aus dem Jahr 2021 ist nicht mehr verfügbar (es wurde kein Image mit "Langzeitsupport"verwendet, also war das Image nur 6 Monate verfügbar). Ein sehr ähnlicher Fehler tritt beim Aufbau des Client auf. Für den Client ist eine ROS2 Distribution vorgesehen, wobei ROS2 vornehmlich für 64-bit Betriebssysteme entwickelt wird und offiziell keine Images für die 32-bit ARM Architektur bereit stellt (die Plattform ärm"fällt in Tier 3, siehe [6]; allerdings sind 32-bit arm Images für ältere ROS1 Umgebungen verfügbar). Da der Test überhaupt nichts mit ROS zu tun hat, kann dies getrost verändert werden. Beide Dockerfiles können mit einer simplen Anpassung auf ein unterstütztes Ubuntu (bspw. übuntu:focal") lauffähig gemacht werden[7]. Als Ergebnis wird die Datei „results.json“ neu berechnet. Die Ergebnisse sind aufgeführt in Tabelle 4.1.

**Tabelle 4.1:** Ergebnisse aus [WH22b, Kapitel 6.13], größerer Zahlenwert steht für höhere Übertragungsrate

| | requests_http | requests_turbo-gears2 | requests_bottle | websocket |
|---|---|---|---|---|
| Aktuelle Messwerte | 965 | 830 | 912 | 3136 |
| Messwerte aus alter Dokumentation | 7857 | 6567 | 7516 | 28230 |

Dort ist sichtbar, dass mittels „websocket“ die meisten Daten übertragen werden können. Zustäzlich eingepflegt wurden die Messergebnisse aus dem alten Projekt, verfügbar auf GitHub[8]. Diese weisen wesentlich größere Werte auf, vermutlich wurde um die Tests durchzuführen eine leistungsfähigere Hardware verdwendet. Mit dem Wissen, dass die Container nicht für den RPI gedacht und auch nicht auf diesem ausgeführt wurden, lässt sich schlussfolgern, dass diese auf einem PC durchgeführt wurden.

Überhaupt ist es fraglich Daten über das „http“-Protokoll zu übertragen, da Robot Operating System eigene Mechanismen zum Datenaustausch besitzt.

[6] https://www.ros.org/reps/rep-2000.html

[7] die Verwendung der aktuellen Version „ubuntu:jammy“ ist nicht möglich, da dieses eine neuere Python Version verwedet, mit der die Tests nicht laufen

[8] https://github.com/dippa-1/autonomous-drone/blob/test/4-sensors-with-ros/test3-container-communication/client/results.json

**Test 4** ist unvollständig. Er ist in keiner Ausarbeitung dokumentiert. Die Dateien von Server und Client sind dieselben wie in Test 3. Es sollte wohl die Kommunikation über ROS erprobt werden, dazu kam es allerdings nicht.

**Ultraschallsensoren** benötigen weiterhin eine Schnittstelle um über ROS Daten zu verbreiten. Eine Schnittstelle wurde ansatzweise Entwickelt und steht als „ros2_ultrasonic_sensor" auf GitHub[9] zur Verfügung. Das Repository enthält einen angepassten Quellcode basierend auf der Beispielimplementation von ROS2[10]. Es wird beispielhaft ein Ultraschallsensor ausgelesen und der Messwert publiziert. [Da zum Testen eine funktionsfähige ROS Umgebung notwendig wäre, kann ich hiermit derzeit nichts anfangen.]

### 4.1.2 Experiment: Positionsbestimmung der Drohne mit GPS

In [WH22b, Kapitel 6.10, 7.4 und folgende] werden GPS-Daten von der Drohne und Bodenstation ausgelesen. Anschließend wird die Drohne angewiesen zu den Koordinaten der Bodenstation zu fliegen.

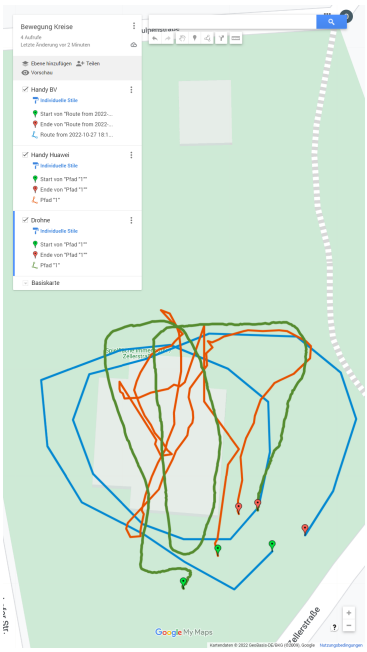
Um die Zuverlässigkeit der Navigation zu überprüfen wurde folgender Versuch durchgeführt: Die Drohne wird im eingeschaltenen Zustand manuell bewegt. Dabei wird das GPS Signal aufgezeichnet. Gleichzeitig wird ein weiteres GPS Gerät mitgeführt, welches ebenfalls die Bewegung aufzeichnet. Anschließend werden die Aufzeichnungen miteinander verglichen.

Zur Durchführung wird auf dem Bordcomputer (RPI) das Programm `mavproxy` (siehe ...) gestartet. Es legt während es aktiv ist, automatisch eine Log-Datei mit diversen Daten zur Drohne an. Die GPS Daten können nach Beenden des Programmes `mavproxy` mit dem Programm `mavtogpx` (in mavproxy-Suite enthalten) entschlüsselt werden. Neben der Drohne werden 2 Smartphones angewiesen, ihre aktuelle Position zu tracken. Anschließend wird ein kleiner Spaziergang mit allen Geräten gemacht. Um möglichst diverse Ergebnisse zu erzielen in 3 Kategorien: in der Ebene, Bergauf, Bergab.

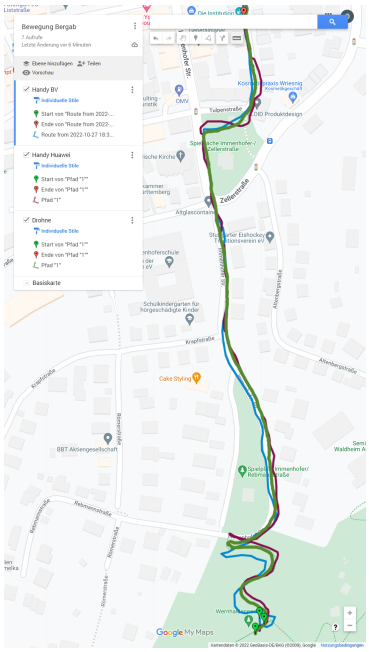
Zur Auswertung wurden die Daten in GoogleMaps hochgeladen. Die Bilder 4.2 zeigen die zurückgelegte Strecke. Auf den Karten sind jeweils farbige Spuren der Smartphones und der Drohne eingezeichnet. Bei näherer Betrachtung (reinzoomen in Maps ist möglich, hier dargestellt ist immer der größtmögliche Ausschnitt) ist zu sehen, dass die Spuren teilweise stark voneinander abweichen. Diese Diskrepanz vergrößert sich teilweise mit der zurückgelegten Strecken, was bedeutet, dass eine aus größerer Entfernung losgeschickte Drohne das Ziel unter Umständen weit verfehlt.

[9] https://github.com/dippa-1/ros2_ultrasonic_sensor

[10] https://docs.ros.org/en/galactic/Tutorials/Beginner-Client-Libraries/Writing-A-Simple-Py-Publisher-And-Subscriber.html




**(a)** Bewegung im Kreis



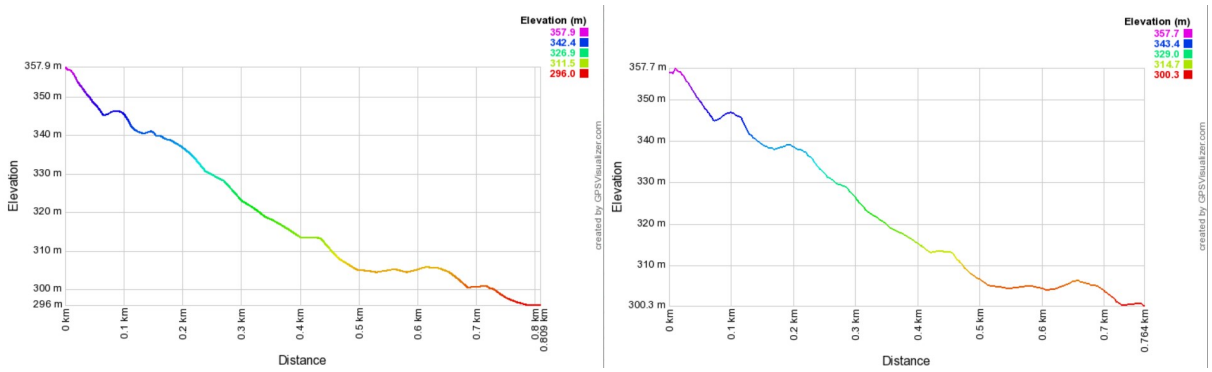

**(b)** Bewegung Bergab

**Abbildung 4.2:** Vergleich Bewegungsprofil verschiedener Geräte. GPS-Modul der Drohne in Grün, zwei Smartphones in Blau und Orange.

Weiterhin kann aus den GPS Daten Höhe, Geschwindigkeit und Neigung über die Zeit beobachtet werden. Die Drohne nimmt insgesamt mehr Messpunkte als die Smartphones auf, was ihr eine größere Genauigkeit und Zuverlässigkeit verleihen sollte. Auch gibt es bei den Smartphones teilweise Aussetzer bei der Aufzeichnung (entweder Signalverlust oder Software-Probleme). In Bild 4.3 zu sehen ist das Höhenprofil beim Bewegen der Drohne und eines Smartphones. Hätte die Drohne schwerwiegende Abweichungen während des Fluges könnte dies fatale Folgen haben. Auch ist das Profil der Drohne insgesamt ruhiger und glatter, was für gute Messwerte spricht.

Um die Genauigkeit des GPS zu verbessern gibt es verschiedene Möglichkeiten[11]. In der Praxis eingesetzt bei Landwirtschaftlichen Maschinen (Zentimetergenaue Fahrzeugführung) wird die zweite auf der Website aufgeführte Methode: über das Internet wird ein Korrektursignal abgerufen und dem GPS-Modul zugespielt. Diese Methode könnte auch einfach mithilfe des RPI und einem GSM-Modul (Mobiles Internet über 3G) implementiert werden.

[11] https://ardupilot.org/copter/docs/common-rtk-correction.html




**(a)** GPS-Modul Drohne **(b)** Huawei-Smartphone

**Abbildung 4.3:** Vergleich Höhenprofil verschiedener Geräte bei Bewegung bergab.

### 4.1.3 Auslesen von Sensor-Daten

Der Flugcontroller stellt verschiedene Sensoren bereit. Zusätzlich waren im ersten Projekt Ultraschallsensoren an den RPI angeschlossenen.

#### Ultraschallsensoren am RPI

Zuerst sollen die Ultraschall-Sensoren mit dem RPI verbunden und getestet werden, um die Wiederverwendbarkeit zu bewerten. Derzeit sind an der Drohne 4 Ultraschallsensoren verbaut. Davon sind zwei nach vorn gerichtet, und jeweils einer nach oben und unten. Somit könnnen Objekte in der Flugbahn detektiert werden, solange die Bewegung entlang einer der Raumrichtungen: „Oben“„Unten“ oder „Vorwärts“ stattfindet. Zusammengesetzte Bewegungen dürften nicht durchgeführt werden, auch darf sich die Drohne nicht rückwärts bewegen.

In [WH22a, Kapitel 4.3.9] ist beschrieben dass die Signale der Sensoren mit $5V$ anliegen, aber auf $3,3V$ herabgesetzt werden müssen um die Pins des RPI nicht zu beschädigen. Die Beschreibung sieht vor dazu einen Spannungsteiler einzusetzen. Bei genauem betrachten des Schaltplans fällt auf, dass der Spannungsteiler falsch gebaut wurde, und nicht die gewünschte Funktion erfüllt. Um die Pins des RPI nicht doch zu beschädigen muss eine alternative Lösung gefunden werden.

Weiterhin wurde ein Python-Script verwendet um die Pins am RPI zu schalten/lesen. Zum anstoßen einer Messung soll das Trigger-Signal für $1\mu s$ auf `HIGH` (logisch 1) gesetzt werden. Um die Zeitverzögerung zu implementieren, wurde die Funktion „wait“ verwendet. Eine Messung mit dem Oszilloskop zeigt, dass besagtes „wait“ im Bereich von 200us liegt.
Eine ähnliche Ungenauigkeit tritt beim Auslesen des ECHO-Signals auf. Die Bibliothek des RPI wird verwendet um die Pins zu Pollen, was den Prozessor unnötig auslastet. Messungen mit konstant eingespeister Einschaltdauer (zu messendes Signal von Signalgenerator erzeugt) ergaben, dass immer 3 von 4 Messwerten gleich, der vierte aber eine Abweichung von ca. 8%

hatte.
Somit kann mit Python keine genaue Messung der Ultraschallsensoren durchgeführt werden.

**Verwendung eines Aurduino zum Auslesen der Sensoren** Zur Lösung der Problemene kann bspw. zusätzlich ein Arduino verwendet werden, der die Sensordaten korrekt ausliest und digitalisiert an den RPI weiterleitet. Dies ist eine zuverlässige Lösung, erhöht aber auch gleichzeitig den Stromverbrauch. Um die Lösung einfach zu halten, wird diese trotzdem angewandt.

Zum Senden digitaler Messwerte muss ein Protokoll verwendet werden. Die UART-Pins des RPI sind bereits mit dem Flugcontroller verbunden. Für den Anwendungszweck bietet sich das I2C-Protokoll an. Es erlaubt eine direkte Verbindung des Arduino mit dem RPI, denn es werden vom Arduino keine $5V$ aktiv geschalten sondern nur der Leitungsbus auf Masse heruntergezogen.

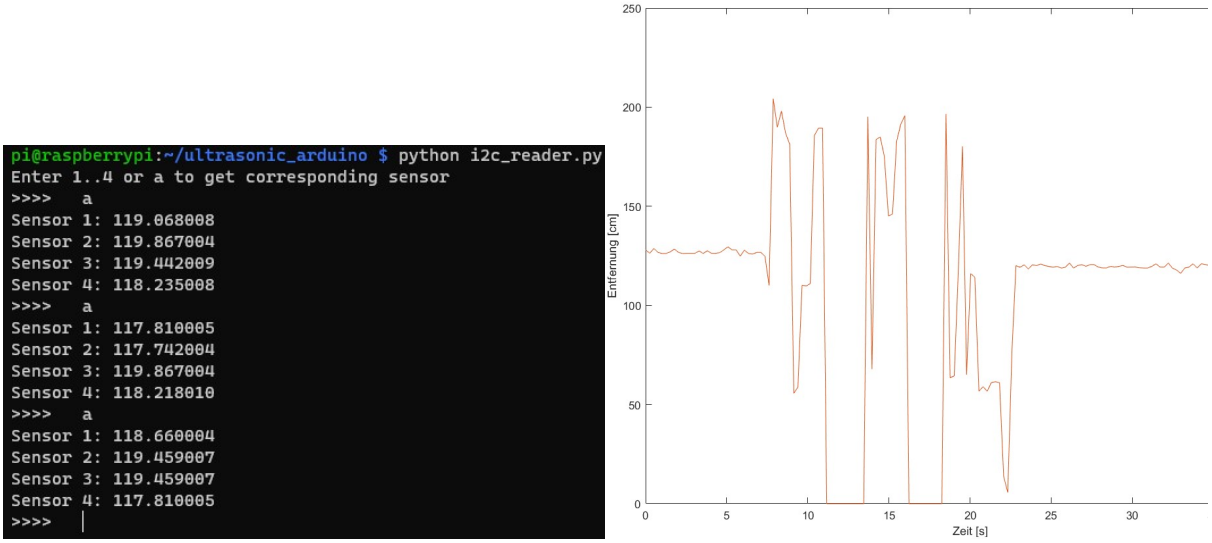
Die Ultraschallsensoren[12] haben eine Reichweite von ca. $2cm$ bis $4m$ und eine Genauigkeit von ca. $3mm$. Für die Entwicklung wird die maximale Distanz auf $3.8m$ festgelegt. Es wird ein kleinerer Wert als im Datenblatt angenommen, da der Idealwert nur bei stillstehenden glatten Flächen erreicht werden kann. Im Datenblatt wird empfohlen, zwischen aufeinanderfolgenden Messungen mindestens $60ms$ abzuwarten, um Fehleinstreuungen durch weitere Ultraschallechos zu vermeiden. Um weitere Fehler zu vermeiden wird diese Zeit vorerst auch zwischen den Messungen der 4 Sensoren eingehalten. Eine weitere Verzögerung entsteht durch das Messen selbst (maximal $\frac{380cm}{0.5\times0.034\frac{cm}{s}} \approx 22352\mu s$), diese wird vorsorglich von der jeweiligen Wartezeit abgezogen. Die minimale Periode der Sensordaten beträgt somit $4 \times 60ms = 240ms$. Grob gesagt, entspricht die Frequenz der Sensordaten somit $4Hz$.

Ein weiteres Problem ist das durch die Ultraschallsensoren verursachte Rauschen der Messwerte. Bei Messungen im Raum mit konstantem Abstand zu Obejekten ergaben sich Abweichungen zwischen 2 Messungen von ca. $1\% - 2\%$, siehe Bild 4.4a, 4.4b. Im linken Bild sind nacheinander die Messwerte mehrerer Ultraschallsensoren aufgeführt, das rechte Bild zeigt den zeitlichen Verlauf von Messdaten. Selbst bei Stillstand (Anfang und Ende der Messung) sind Schwankungen in den Messwerten zu sehen.

Auch haben die Ultraschallsensoren ein Problem mit Bewegungen. In Bild 4.4b wurde der Sensor zuerst in Richtung Decke gehalten (Beginn bei ca. $8s$ auf der x-Achse). Dann wurde er waagerecht gedreht, in den Raum zeigend. Kann der Sensor keinen Wert erfassen, kommt es auf dem Arduino zu einem Timeout und es wird $0$ zurückgegeben (ca. bei $11s$). Zu beachten sind die starken Abweichungen während den Bewegungen: es kommt immer wieder zu starken Einbrüchen und Anstiegen (bspw. bei $14s$) durch teilweisen Verlust des Echo-Signals.

Durch den Feldversuch wird ersichtlich, dass Ultraschallsensoren nur funktionieren, wenn sie nahezu senkrecht auf Flächen gerichtet werden. Somit können auch keine Hindernisse (wie

[12] https://cdn.sparkfun.com/datasheets/Sensors/Proximity/HCSR04.pdf




**(a)** Nacheinander ausgelesene Sensordaten **(b)** Verhalten bei Bewegung eines Sensors

**Abbildung 4.4:** Demonstration der Ultraschallsensoren

bspw. Wände) detektiert werden, auf die sich die Drohne schräg zubewegt. Nicht messbare Entfernungen (zu nahe oder zu große Entfernung) werden beim weiteren Vorgehen durch die Entfernung von $4m$ ersetzt und die maximal messbare Entfernung der Sensoren auf $3m$ begrenzt.

Um die Messwerte der Sensoren sinnvoll verarbeiten zu können ist weiteres Filtern notwendig. Eine einfache Implementation für einen Filter sind Moving Average Filter, oder speziell für diesen Anwendungszweck, wie in [13] beschriebene Median Filter. Bei dem zuletzt beschriebenen Verfahren werden manuell Grenzen festgelegt, welche Messwerte gefiltert werden müssen. Für die Messungen wurden Abweichungen kleiner 1% ignoriert und größer 3% gefiltert. Vom beschriebenen Vorgehen zum filtern wurde aber abgeichen und anstatt des Medians jeweils der Mittelwert der letzten Messwerte, gespeichert in einem Filter-Array, errechnet. In Bild 4.5 ist eine weitere Messung dargestellt. Die besten Resultate (wenige Sprünge, steile Anstiege) liefert der „Median Filter 1“ und soll im Projekt weiterhin verwendet werden.

[TODO: neue Bibliothek entwerfen, am besten gleich mit ROS]

### Einspeisen von Parameterwerten der Drohne für ROS

[TODO: vielliecht gibt es schon eine "direkte Einspeisung", muesste ich mich mal belesen]
[TODO: Alternativ kann der Flugcontroller auch einen EKF berechnen]

[13] https://renegaderobotics.org/filtering-sensor-data/

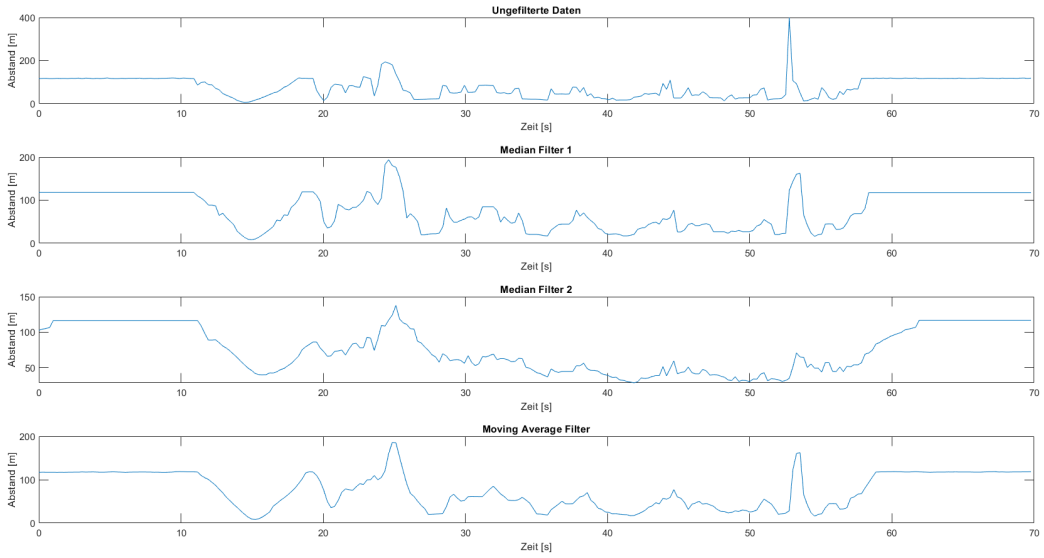



**Abbildung 4.5:** Messung mit einem Ultraschallsensor. Zu sehen: oben ungefilterte Messwerte; Median Filter 1 pflegt Sprünge in Messdaten direkt in Filter-Array ein aber berechnet Durchschnitt; Median Filter 2 pflegt berechneten Durchschnittswert in Filter-Array ein; Moving Average berechnet immer den Durchschnittswert der vergangenen Messwerte.

# Literaturverzeichnis


[Gru22] R. Gruden Prof. Dr.-Ing. *Sensorik Und Messwertverarbeitung*. 2022 (zitiert auf S. ).

[HD22] Harald Wirth, Dominik Helfenstein. *Nachfolger-Info_Studienarbeit_autonome_Drohnen*. 14. Juli 2022 (zitiert auf S. ).

[WH22a] H. Wirth, D. Helfenstein. „Erweiterung Einer Bestehenden Drohne Um Eine Autonomflugfähigkeit“. DHBW Stuttgart, 23. Jan. 2022 (zitiert auf S. ).

[WH22b] H. Wirth, D. Helfenstein. „Erweiterung Einer Bestehenden Drohne Um Eine Autonomflugfähigkeit (2/2)“. DHBW Stuttgart, 10. Juni 2022 (zitiert auf S. ).

**Erklärung**

Ich versichere, diese Arbeit selbstständig verfasst zu haben. Ich habe keine anderen als die angegebenen Quellen benutzt und alle wörtlich oder sinngemäß aus anderen Werken übernommene Aussagen als solche gekennzeichnet. Weder diese Arbeit noch wesentliche Teile daraus waren bisher Gegenstand eines anderen Prüfungsverfahrens. Ich habe diese Arbeit bisher weder teilweise noch vollständig veröffentlicht. Das elektronische Exemplar stimmt mit allen eingereichten Exemplaren überein.

---

Ort, Datum, Unterschrift