

# Studienarbeit T3200

## **Erweiterung bestehender Drohnen um eine Autonomflugfähigkeit**

des Studienganges Eletrotechnik  
an der Dualen Hochschule Baden-Württemberg, Stuttgart

eingereicht von  
*Markus Rein*

Abgabedatum: 14. Mai 2023

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

# Eigenständigkeitserklärung

Studienarbeit T3200 von Markus Rein (B. Sc. Elektrotechnik)

Anschrift            Alexanderstraße 146, 70180 Stuttgart

Matrikelnummer    6983030, TEL20GR5

Deutscher Titel    *Erweiterung bestehender Drohnen um eine Autonomflugfähigkeit*

Hiermit erkläre ich,

- dass ich die vorliegende Arbeit selbstständig verfasst habe,
- dass keine anderen als die angegebenen Quellen benutzt und alle wörtlich oder sinngemäß aus anderen Werken übernommenen Aussagen als solche gekennzeichnet sind,
- dass die eingereichte Arbeit weder vollständig noch in wesentlichen Teilen Gegenstand eines anderen Prüfungsverfahrens gewesen ist,
- dass ich die Arbeit weder vollständig noch in Teilen bereits veröffentlicht habe und
- dass ich mit der Arbeit keine Rechte Dritter verletze und die Universität von etwaigen Ansprüchen Dritter freistelle.

---

Stuttgart, den 21.10.22 – 22.01.2023

# Inhaltsverzeichnis

<b>1. Einleitung</b>	<b>1</b>
<b>2. Einführung in Projekt</b>	<b>3</b>
2.1. Verwendete Technologien . . . . .	3
2.1.1. WLAN . . . . .	3
2.1.2. GPS . . . . .	3
2.1.3. Serielle Verbindungsprotokolle . . . . .	3
2.2. Beschreibung der Drohne in den einzelnen Phasen . . . . .	3
2.2.1. Beschreibung des Ausgangsystems Drohne . . . . .	3
2.2.2. Beschreibung des Zielsystem Drohne mit autonomer Flugfähigkeit	4
2.2.3. Beschreibung der Simulation . . . . .	4
<b>3. Stand der Technik</b>	<b>6</b>
3.1. Erweiterte Flugmodi der Drohne . . . . .	6
3.2. ROS und Avoidance . . . . .	7
3.3. Hinderniserkennung und ROS Punktwolken . . . . .	7
3.3.1. SLAM Algorithmus . . . . .	7
3.3.2. Stereokamera . . . . .	8
3.3.3. Optical Flow . . . . .	8
3.3.4. Punktwolkenformat . . . . .	8
<b>4. Einführende Flugtests: Mission-Mode</b>	<b>10</b>
4.1. Flug auf gerade Linie . . . . .	11
4.2. Rally-Punkte . . . . .	11
4.3. Flug mit Survey . . . . .	12
4.4. Flug mit GeoFence . . . . .	13
4.5. Flug mit Region of Interest . . . . .	13
4.6. Zusammenfassung . . . . .	14
<b>5. Vorbereitung zur Umsetzung von Avoidance</b>	<b>16</b>
5.1. Avoidance als Simulation in Software . . . . .	16
5.2. Avoidance als Simulation mit Hardware . . . . .	20
5.3. Machbarkeitsstudie Stereokamera mit Raspberry Pi . . . . .	22
5.4. Beschaffung Hardware Stereokamera . . . . .	23
5.4.1. Halterung für Platine und Kameras . . . . .	24
5.4.2. Software auf dem Compute Module 4 . . . . .	25

*Inhaltsverzeichnis*

5.5.	Einrichtung der Ultraschallsensoren . . . . .	25
5.5.1.	Erzeugen einer Punktwolke . . . . .	25
5.5.2.	Software der Sensoren . . . . .	27
5.5.3.	Einbinden der Sensoren . . . . .	29
5.5.4.	Verbesserung der Aufnahmegenauigkeit der Sensoren . . . . .	30
<b>6.</b>	<b>Erweiterte Flugtests: Avoidance</b>	<b>31</b>
<b>A.</b>	<b>Anhang</b>	<b>32</b>
	<b>Literatur</b>	<b>41</b>
	<b>Abbildungsverzeichnis</b>	<b>42</b>
	<b>Tabellenverzeichnis</b>	<b>43</b>

# 1. Einleitung

Dieses Projekt dient der Verwirklichung Autonomen Fliegens.

## Ziele der Arbeit:

- Erweiterung der Sensorfähigkeit der Drohne um Tiefenkamera
- Verwendung von Tiefenbildern und Ultraschallsensordaten für Hinderniserkennung
- Autonomer Flug der Drohne um große, flächige Hindernisse wie Häuser oder Automobile unter Verwendung der Software *Avoidance*

## Optionale Erweiterungen:

- Verfeinerungen und Anpassungen der Software *Avoidance*, sodass kleinere, strukturreiche Hindernisse (bspw. Bäume) erkannt und umflogen werden
- Einführung kamerabasierte Zielerkennung als Alternative zur GPS-gestützten Zielführung

## Geplantes Vorgehen:

1. Vervollständigen der Simulation aus vorhergehendem Projektteil: die simulierte Drohne soll im Flug allein mit Kamerabildern und der Software *Avoidance* Hindernissen ausweichen
2. Beschaffung und Installation Tiefenkamera an realer Drohne
3. Aufspielen der Software *Avoidance* auf reale Drohne
4. Flugtests mit Tiefenkamera
5. Einbinden von Ultraschallsensordaten in Softwareverarbeitung von *Avoidance*
6. Flugtests mit Tiefenkamera und Ultraschallsensordaten
7. mehr Details: was funktioniert in der Endanwendung, was sind die Schritte in der Simulation, ,was wird simuliert

## 1. Einleitung

Im weiteren Verlauf der Arbeit spielen folgende Begriffe eine wichtige Rolle:

**Bodenstation:**

**Inertial Measurement Unit (IMU):** Zusammenfassung der Sensoren Beschleunigungssensor, Kompass und Gyroskop

**Raspberry Pi (Einplatinencomputer) (RPI):** Einplatinencomputer, eingesetzt als Bordcomputer

**ROS:** Metabetriebssystem zur Vernetzung von Roboterbestandteilen (Aktor-Sensor-Verknüpfung), Überwachung und Steuerung von Prozessen

**Docker:** Containerisierung von Betriebssystemumgebungen. Beispielsituation:

- auf dem RPI läuft Raspberry Pi OS (Linux Debian), genannt Host-Betriebssystem
- unabhängig davon kann innerhalb eines Containers Linux Ubuntu installiert werden
- im Container sind speziell festgelegte Bibliotheken, Umgebungsvariablen, Speichergeräte und Netzwerke zum Programmbetrieb vorhanden, es können Programme gestartet werden, die nicht auf dem Host-Betriebssystem lauffähig sind
- im Container gestartete Programme laufen parallel zu Programmen auf dem Host-Betriebssystem, sie sind trotzdem im Prozessmanager (*top*) von Raspberry Pi OS sichtbar

**PX4:** Offizielle Software der Dronecode Stiftung mit der verschiedene  $\mu$ C, embedded System, oder PC zum Steuern von Robotern, Fahrzeugen oder Drohnen ausgestattet werden können

**Avoidance:** Projekt im Umfeld von PX4 zur autonomen Steuerung von Drohnen in unbekanntem Umfeld. Aufgeteilt in 3 Unterprojekte:

- Local Planner
- Global Planner
- Safe Landing Planner

Im derzeitigen Projekt wird der Local Planner umgesetzt.

...

## 2. Einführung in Projekt

Auf physikalische Zusammenhänge wird in der Arbeit nicht eingegangen, da am bestehenden System Drohne keine Änderungen vorgenommen werden. Es wird lediglich die Software angepasst, was die Flugfähigkeit aber nicht beeinflusst.

### 2.1. Verwendete Technologien

#### 2.1.1. WLAN

#### 2.1.2. GPS

#### 2.1.3. Serielle Verbindungsprotokolle

### 2.2. Beschreibung der Drohne in den einzelnen Phasen

#### 2.2.1. Beschreibung des Ausgangsystems Drohne

Die Drohne kann mit Methoden der Regelungstechnik als adaptives System beschrieben werden. Dabei dient der Flugcontroller als Regler, die Motoren als Steuerstrecke, und Bordcomputer sowie Bodenstation zur Identifikation und Modifikation der Parameter. Zu Beginn wird die Drohne in ihrer Ausgangslage, ohne automone Flugfähigkeiten beschrieben. Die Bestandteile des Systems sind zunächst in Tabelle 2.1 aufgelistet.

Tabelle 2.1.: Systemübersicht Drohne und Bodenstation

	Drohne		Bodenstation
Funktion	Flugcontroller	Bordcomputer	
Funktion	Autopilot-Software liest Sensoren und steuert Motoren der Drohne	Bereitstellung WLAN- Netzwerk zur Verbin- dung von Autopilot und Bodenstation	Parametrierung und Steuerung der Drohne
Hardware	Pixhawk 4	Raspberry Pi 3B+	PC und/oder Smart- phone
Software	PX4	MAVLink-Router	QGroundControl

Zusammen bilden sie ein System wie in Bild A.1 gezeigt. Im Flugbetrieb werden von der Bodenstation Flugbefehle (feste Zielkoordinaten, relative Koordinaten, oder manuelle

## 2. Einführung in Projekt

Motoransteuerung) per WLAN-Verbindung an den Bordcomputer, von diesem per Serieller Schnittstelle (UART) an den Flugcontroller, geschickt. Mehr Details zur Steuerung mit der Bodenstation in Kapitel 3.1. Der Flugcontroller steuert anschließend die Motoren um die gewünschte Position zu erreichen. Als Eingabegrößen stehen dem Flugcontroller Sensordaten von Beschleunigungssensor, Kompass (erstere bilden zusammen die IMU) und Barometer, diese sind im Flugcontroller integriert, und der GPS-Antenne zur Verfügung.

### 2.2.2. Beschreibung des Zielsystem Drohne mit autonomer Flugfähigkeit

Zusätzliche Sensoren stellen Daten für Berechnungen auf dem Bordcomputer bereit. Diese müssen ausgewertet und die Ergebnisse dem Flugcontroller zugespielt werden. Das Auswerten und Zuspielen der Daten ist zeitkritisch denn es beeinflusst direkt den Flug der Drohne. Im Idealfall sollten Berechnungen direkt auf dem Flugcontroller oder in unmittelbarem Zusammenhang mit diesem durchgeführt werden. Zu den Aufgaben zählt:

- Erfassen von Ultraschall-Daten zum Detektieren von Hindernissen
- Erfassen von Bildern
- Verarbeiten von Bildern zur Detektieren von Hindernissen
- Berechnung alternativer Flugbahn zur Umgehung von Hindernissen

Die Umsetzung der Flugplanung soll mittels der Software *Avoidance* erfolgen. Diese arbeitet auf dem Metabetriebssystem ROS, welches wiederum auf *Ubuntu* aufsetzt. (Im Anwendungsfall ROS Noetic auf Ubuntu 20.04.) Die Software kann vollständig auf dem RPI betrieben werden. Somit ergeben sich für den RPI weitere Aufgabenbereiche, wie nachfolgend dargestellt. Das erweiterte System ist dargestellt in Bild A.2.

- Ubuntu 20.04 als Betriebssystem (nur indirekt möglich innerhalb eines Containers möglich)
- ROS-Noetic innerhalb des Docker-Containers
- Einlesen der Ultraschallsensordaten, erzeugen von Tiefenkarte und publizieren entsprechender Topic
- Einlesen der Kamerabilder, erzeugen von Tiefenkarte und publizieren entsprechender Topic
- Ausführung von Avoidance

### 2.2.3. Beschreibung der Simulation

Die Entwicklung von Software für und im Zusammenhang mit dem Flugcontroller sieht vor, in einer Simulation getestet zu werden. Von der Dronecode-Stiftung wird als offizielle Umgebung dazu der Simulator *Gazebo* empfohlen[1]. Sowohl die *PX4*- als die *Avoidance*-Software können vollständig in diesem betrieben werden. Zum Betrieb des

## 2. Einführung in Projekt

Simulators wird wieder das Betriebssystem Linux Ubuntu benötigt. Deshalb werden für das Projekt einige Einschränkungen aufgenommen. Die weiteren Ausführungen hier beschreiben die Ausgangslage zur Simulation, in Kapitel 5 wird ROS eingeführt und eingerichtet.

Zum Vorgehen wird der Hardware-In-the-Loop (HIL)-Aufbau mit dem Simulator *AirSim* von Microsoft[2], wie in [3, Kapitel 3.4.1] beschrieben, verwendet. Bild A.3 zeigt, in Anlehnung an Bild A.1, die durch die Simulation übernommenen Funktionen. Die Komponenten sind im Betrieb wie folgt verbunden:

**PC-Flugcontroller:** USB-Kabel, wird von *AirSim* zur direkten Kommunikation mit dem Flugcontroller verwendet

**PC-Bordcomputer:** WLAN, erlaubt Verbindung Bodenstation mit Flugcontroller

Die Aufgaben der Komponenten während der Entwicklung sind:

**Simulation:** In der Simulation werden sowohl alle physikalischen Effekte berechnet als auch die virtuelle Umgebung der Drohne dargestellt. Die Sensordaten werden der Drohne direkt von der Simulation eingespeist. Eine resultierende Ansteuerung der Motoren wird in die Simulation übernommen. Somit kann sich die Drohne in der Simulation wie in realer Umgebung bewegen. Außerdem werden von der Drohne aufgenommene, simulierte Kamerabilder bereitgestellt.

**Drohne:** Der Flugcontroller auf der Drohne wird im HIL-Modus betrieben. Alle Ein- und Ausgänge zum Controller werden durch virtuelle Schnittstellen der Simulation ersetzt. Die Kommunikation mit dem Bordcomputer bleibt dieselbe wie zuvor, sodass die Drohne per Bodenstation gesteuert werden kann.

**Bordcomputer:** Wird weiterhin nur zur Kommunikation zwischen Bodenstation und Flugcontroller verwendet. Erweiterte Funktionen werden auf einem separaten Rechner entwickelt und getestet um anschließend auf den Bordcomputer überspielt zu werden.

# 3. Stand der Technik

In diesem Kapitel werden die Grundlagen verwendeter Software erläutert.

## 3.1. Erweiterte Flugmodi der Drohne

Bei Verwendung der Drohne in Verbindung mit einer Bodenstation, wird die aktuelle Position auf einer Karte eingezeichnet. Von diesem Punkt aus kann der Drohne eine Wegvorgabe eingespielt werden, der „Mission-Mode“. Dabei enthält die Karte Informationen zur ungefähren Beschaffenheit der Umgebung, sodass die Drohne nicht Tiefer fliegen würde als der Boden der Karte. Gleichzeitig sind die standardmäßigen Sicherheitsmaßnahmen eingestellt, bspw. eine Mindestflughöhe von  $4m$  einzuhalten. Neben der Wegvorgabe können dem Flugcontroller weiterhin verbotene Zonen mitgeteilt werden, die nicht durchflogen werden dürfen, genannt „Geo-Fence“. Der Algorithmus sieht derartige Zonen als Hindernis an. Bei Kontakt mit ihnen wird ein Failsafe ausgelöst. PX4 kennt zwei derartige Modi:

**Failsafe GeoFence:** Ein Zylinder dessen Durchmesser von der Funkreichweite der Fernbedienung und maximaler Flughöhe beschränkt ist. Bei Durchbruch verfällt die Drohne standardmäßig in den „Return-Mode“ und kehrt zu ihrer Ausgangsposition zurück.

**GeoFence Plan:** Kreise oder Polygone auf Karte die nicht durchflogen oder verlassen werden dürfen (je nach Einstellung). Bei Bruch der Bedingung verfällt die Drohne in den „Hold-Mode“ und bleibt schlicht stehen.

Es ist also bereits mit Bordmitteln möglich das Flugverhalten zu beeinflussen. Für das Vorgehen mit *Avoidance* kommt der „Offboard-Mode“ zum Einsatz. In diesem Modus werden dem Flugcontroller ständig neue Anweisungen, als nächster Wegpunkt, eingespeist.

Weiterhin können der Drohne im Missionsmodus sogennante Region Of Interest (ROI) mitgeteilt werden. Ist eine Kamera an der Drohne vorhanden, wird diese gezielt auf die Positionen gerichtet. Ist keine Kamera explizit definiert richtet sich die Drohne mit dem Bug in Richtung der ROI aus. Da die Drohne sowohl vorwärts als auch seitwärts fliegen kann, hält sie durchgehend auf den Punkt zu.

## 3.2. ROS und Avoidance

Das Projekt „Obstacle Detection and Avoidance“[4], auf GitHub verfügbar als PX4-Avoidance<sup>1</sup>, hier nur *Avoidance* genannt, entstand in enger Zusammenarbeit mit der Dronecode Stiftung an der ETH Zürich, dem Ursprungsort aller *PX4*-Software. Es arbeitet innerhalb einer ROS-Umgebung.

Es stehen im Projekt 3 Algorithmen zur Verfügung, die unabhängig voneinander zu betrachten sind. Alle dienen der Anpassung der Flugbahn in unbekannter Umgebung:

**Local Planner:** Navigiert um Hindernisse in der direkten Umgebung

**Global Planner:** Speichert nahezu vollständige Karte der Umgebung und erlaubt Navigation durch Labyrinth-artige Umgebung

**Safe Landing Planner:**

Die Software von *Avoidance* erhält die Daten des Flugcontrollers über das Zwischenprogramm *mavros* (MAVLink-zu-ROS-Übersetzung, siehe [3, Kapitel 5.2/5.4]). Es sind die Soll-Trajektorie und Sensordaten vom Flugcontroller bekannt. Außerdem wird zur Navigation eine *Punktwolke* (siehe Kapitel 3.3) der Umgebung eingespeist. Falls das Programm ein Hindernis in der Flugbahn erkennt, wird eine angepasste Trajektorie an den Flugcontroller ausgegeben.

Die Software kann nicht direkt auf dem Flugcontroller ausgeführt werden, da die Berechnungen sehr viele Ressourcen (Rechenkapazität, Speicher) benötigen. Weiterhin empfehlen die Entwickler, zuerst den Local Planner zu implementieren, da dieser am besten funktioniert. Offizielle Empfehlungen der Entwickler verwenden leistungsstarke Hardware wie Nvidia Jetson (Hardware-Unterstützung für Bildverarbeitung) oder Intel RealSense (Kamera mit Tiefenerkennung).

Im Zusammenhang mit der Software sind letztere bereits erprobt. Aufgrund des hohen Preises können sie nicht in diesem Projekt verwendet werden, siehe [5, Kapitel 4.3.8]. Als Alternative können auch Stereokameras verwendet werden. Beispielcode zur Einbindung von Tiefenbildern ist unter Github (siehe Fußnote 1) vorhanden.

## 3.3. Hinderniserkennung und ROS Punktwolken

Als Verschiedene Prinzipien stehen zur Hinderniserkennung zur Verfügung. Als Eingabegröße für *Avoidance* müssen die verarbeiteten Bilder im Punktwolkenformat als ROS-Topic vorliegen. Nachfolgend vorgestellt werden die grundlegenden Techniken der Bilderkennung.

### 3.3.1. SLAM Algorithmus

SLAM Techniken entstanden bereits in den 1980-1990 Jahren und werden bspw. bei Robotern eingesetzt, die in Hallen navigieren (für die kein GPS verfügbar ist). Zum Einsatz

---

<sup>1</sup><https://github.com/PX4/PX4-Avoidance>

### 3. Stand der Technik

kommen Kamerasysteme in Verbindung mit Entfernungssensoren (Sonar, Radar, Lidar). Die Ergebnisse von SLAM können nicht garantiert werden und sind nicht reproduzierbar, weshalb es in keinen kritischen Umgebungen (bspw. wenn Verletzungsrisiko besteht) eingesetzt werden kann.

Allgemein wird SLAM durch einen modularen Prozess beschrieben:

**Lokalisierung:** per Motorfortschritt, IMU, Kamera, etc.

**Kartengenerierung:** durch einen der Algorithmen

- 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 Eingangsdaten
- Monte-Carlo-Lokalisierung (Partikelfilter): nimmt Wahrscheinlichkeiten für jeden Ort an; genauer als Kalman-Filter; lineare Komplexität; Weniger Speicher als Kalman-Filter; Nachteil: Stillstand ohne sich ändernde Sensordaten

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

**Visual SLAM**, kurz vSLAM, stellt eine Unterform des SLAM dar, bei der ausschließlich Kameras zur Erfassung der Umgebung eingesetzt werden. Algorithmen verwenden zumeist zusätzlich die Daten der IMU, um die Bewegung der Kamera in die Berechnung der Position einzubeziehen.

#### 3.3.2. Stereokamera

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

#### 3.3.3. Optical Flow

In Bewegungsabläufen werden Objekte verfolgt und können somit relativ zur Kamera bestimmt werden. Das Prinzip wird auch von Lebewesen im Gehirn angewandt. Dabei kann schlecht zwischen der Bewegung der Kamera und der Bewegung von Objekten unterschieden werden. Ungenau, da Kameras immer eine Verzerrung besitzen.

#### 3.3.4. Punkt wolkenformat

Die Möglichkeiten Optical Flow und Stereokamera erzeugen jeweils Tiefenkarten. In diesen Bildern sind, zumeist als Graustufen, Pixel je nach Entfernung zur Kamera gekennzeichnet. Zur Umwandlung als Punkt wolke muss jedes Pixel abgetastet werden um als Koordinate im 3D-Raum dargestellt werden zu können. Das ROS beinhaltet sowohl

### *3. Stand der Technik*

Programme zur Stereoverarbeitung basierend auf OpenGL<sup>2</sup>, als auch die Erzeugung von Punktwolken<sup>3</sup>.

---

<sup>2</sup>siehe [http://wiki.ros.org/stereo\\_image\\_proc](http://wiki.ros.org/stereo_image_proc)  
<sup>3</sup>siehe [http://wiki.ros.org/depth\\_image\\_proc](http://wiki.ros.org/depth_image_proc)

## 4. Einführende Flugtests: Mission-Mode

Einführend wird die Verwendung von Bodenstation und Drohne im Missionsmodus beschrieben. Zum Einsatz kommt die Software *QGroundcontrol* auf dem PC.

Der Missionsplaner kann jederzeit oben Links in *QGroundcontrol*, wie in Bild 4.1 dargestellt, aufgerufen werden. Es öffnet sich ein erweitertes Menü. Nach dem Anlegen des jeweiligen Planes muss dieser zur Drohne hochgeladen werden (nur während eine Verbindung hergestellt ist). Anschließend kann der Missions-Planer verlassen und die jeweilige Mission über das wechseln in den Mission-Mode gestartet werden.

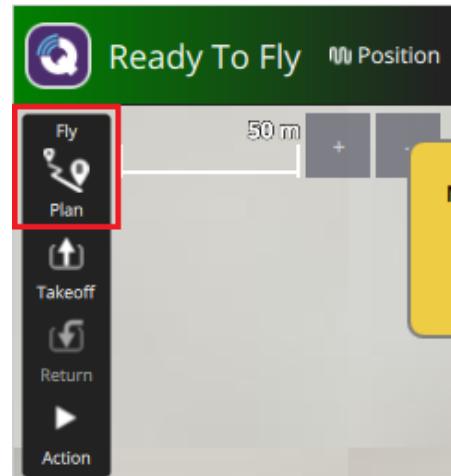


Abbildung 4.1.: QGroundControl  
Missionplaner-Menu  
wird durch Klick auf  
Rot umrahmten Button  
geöffnet

#### 4. Einführende Flugtests: Mission-Mode

## 4.1. Flug auf gerade Linie

Für den einfachsten Fall lassen sich im Missionsplaner eine Liste von Wegpunkten anlegen, siehe dazu Bild 4.2. Nachdem am linken Rand „Waypoint“ ausgewählt wurde lassen sich diese beliebig auf der Karte platzieren. Der erste Punkt beschreibt den Start, der letzte kann als Landepunkt dienen. Andernfalls kann die Drohne zum Startpunkt zurückkehren oder in letzter Position stehen bleiben.

Jeder Wegpunkt beschreibt eine Position im Raum. Die Parameter Höhe, Bewegungsgeschwindigkeit und eine relative Drehung zur Vorwärtsrichtung (Yaw) können für jeden Punkt separat am rechten Rand festgelegt werden.

Im unteren Bildbereich ist das Höhenprofil über den Weg eingezeichnet: in Orange gefärbt die jeweilige Flughöhe der Drohne, in Grün der Boden laut Global Positioning System (GPS) Karte.

Mit den im Bild gezeigten Einstellungen wird die Drohne starten, auf gerade Linie nach vorn fliegen und anschließend landen. Dieses Verhalten wurde in der Simulation überprüft. Dazu wurde ein Hilfsprogramm geschrieben, welches die derzeitige Geschwindigkeit, Höhe und Orientierung der Drohne mitschreibt. Eine Auswertung als Diagramm ist zu sehen in Bild 4.3.

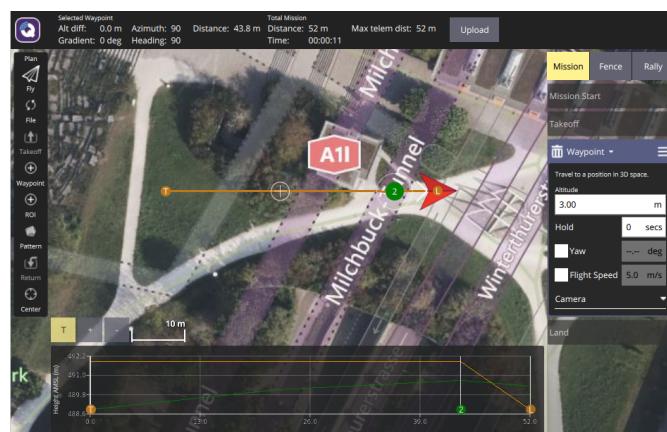


Abbildung 4.2.: QGroundControl Missionplaner-Wegpunkte

## 4.2. Rally-Punkte

Neben den Wegpunkten können Rally-Punkte beliebig auf der Karte verteilt werden. Diese dienen als alternative Landepunkte. Wird während der Mission ein „Return to Land“ ausgelöst (bspw. als letzter Punkt einer Mission, durch GeoFence, oder im Fehlerfall), fliegt die Drohne den nächsten Rally-Punkt an und landet. Das Verhalten ist in Bild 4.4 dargestellt. Die Funktion kann zur Fehlervermeidung verwendet werden.

#### 4. Einführende Flugtests: Mission-Mode

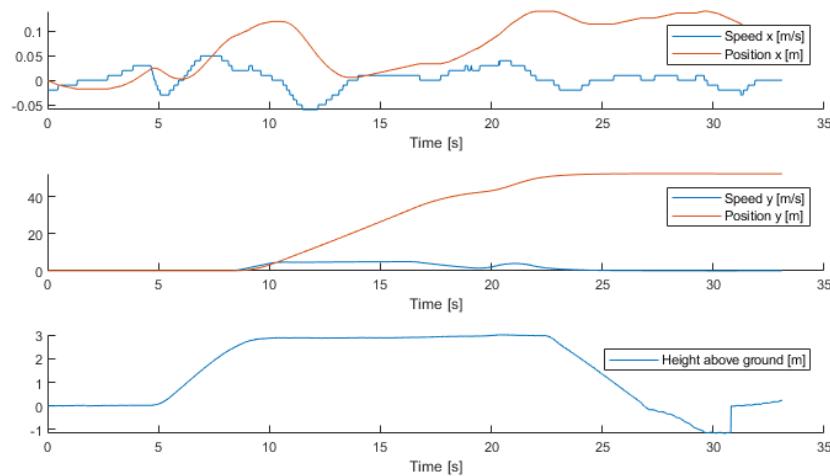


Abbildung 4.3.: Auswertung Missionsplaner-Wegpunkte: Diagramme geplottet von Matlab. In den oberen Diagrammen zu sehen sind die jeweilige x-/y-Geschwindigkeit und -Position während des Fluges. Positive x-Richtung zeigt nach Norden, positive y-Richtung nach Osten. Im unteren Diagramm die Flughöhe. Da der Simulator von einer flachen Welt ausging, die GPS-Karte aber eine Position in Österreich annahm, wurde der Landepunkt höher als der eigentliche Boden angedacht. Durch verfrühtes Abschalten der Motoren "fiel" die Drohne auf den simulierten Boden, was zu Fehlern in der beschriebenen Höhe führte. Die Drohne konnte diesen Fehler beim Landen bei ca. 30s ausgleichen, hob erneut ab, um schließlich langsam zu sinken und blieb sicher stehen.

### 4.3. Flug mit Survey

Anstatt Wegpunkte manuell anzulegen, können mit „Pattern“<sup>1</sup> verschiedene Routen automatisch angelegt werden. Mit „Survey“ kann ein Gebiet eingestellt werden, welches von der Drohne im überflogen wird. Im Auswahlmenü stehen verschiedene Formen wie Polygon oder Kreis zur Verfügung. Innerhalb dieser wird ein Zick-Zack Pfad aus mehreren Wegpunkten angelegt. Innerhalb des Konfigurationsmenü können Abstand und Winkel der Bahnen eingestellt werden. Die Drohne während der Abarbeitung einer Survey ist in Bild 4.5 gezeigt. Neben Survey können noch weitere Planungsmodi „Corridor-Scan“ und „Structure-Scan“ angelegt werden. Bei ersterem fliegt die Drohne kontinuierlich eine gerade Strecke hin und her. Mit letzterem kann in verschiedenen Höhen um Objekte wie Gebäude geflogen werden, um Aufnahmen von der Beschaffenheit zu machen.

---

<sup>1</sup>siehe <https://docs.qgroundcontrol.com/master/en/PlanView/Pattern.html>

#### 4. Einführende Flugtests: Mission-Mode

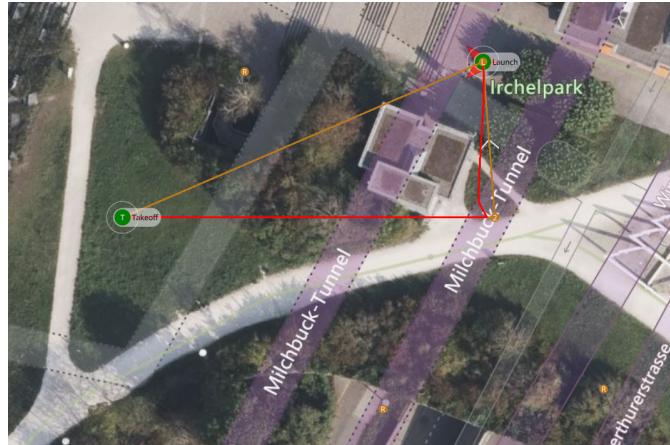


Abbildung 4.4.: QGroundControl Missionplaner-Rallypunkte: 4 Rally-Punkte verteilt auf Karte um Flugbahn der Drohne. Nach Abschluss der Mission wurde der obere rechte Punkt angeflogen und gelandet

### 4.4. Flug mit GeoFence

Ein GeoFence kann unabhängig einer Mission angelegt werden. Im Missionsmodus wird von *QGroundControl*, vor Hochladen der Mission, überprüft, ob Punkte im betroffenen Bereich liegen. Trifft dies zu wird eine Fehlermeldung ausgegeben und das hochladen des Missionsplanes verhindert. Es gibt 2 Verhaltensmuster: Einschlüsse (die nicht Verlassen werden dürfen) und Ausschlüsse (die nicht Betreten werden dürfen).

Wird ein manueller Flug durchgeführt, kommt zuerst eine Warnung, dann stoppt die Drohne. Ein Beispiel für einen Einschluss ist in Bild 4.6 gezeigt. Selbst bei mehrfachem Anflug war es nicht möglich die gekennzeichnete Zone zu erreichen. Die Drohne verfiel jeweils in den *Hold*-Modus und musste zurückgestellt werden.

### 4.5. Flug mit Region of Interest

ROIs dienen der Orientierung der Drohne während einer Mission. Es können ROIs-Punkte gleich den Wegpunkten auf der Karte gesetzt werden. Um die Ausrichtung zu verdeutlichen wurde Bild 4.7 aufgenommen. Es zeigt die Drohne als roten Pfeil, in Richtung des mit „R“ markierten Punktes schauend. Ein Punkt ist solange gültig, bis er abgewählt oder ersetzt wird.

Das ROI-Kommando welches von MAVLink übertragen wird, kann später verwendet werden, um die Drohne gezielt auf Hindernisse zeigen zu lassen, falls diese sich außerhalb des Sichtbereiches der Sensoren aber noch innerhalb des Gefahrenbereiches der Drohne befinden.

#### 4. Einführende Flugtests: Mission-Mode

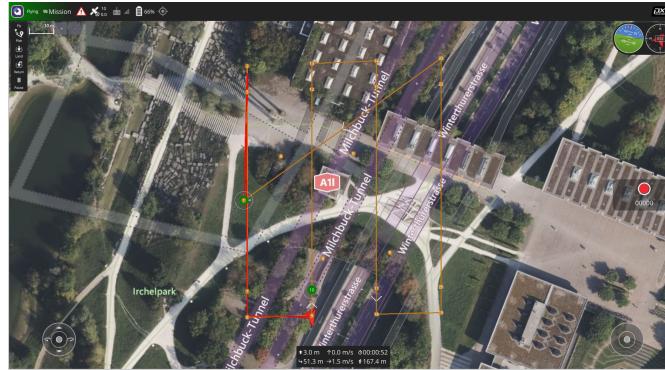


Abbildung 4.5.: QGroundControl Missionplaner-Survey: Automatisch angelegte Wegpunkte werden angeflogen. Nach Start bei Punkt „T“ flog die Drohne nach Norden zu Punkt „3“ und befindet sich im Moment der Aufnahme in Drehung bei Punkt „9“

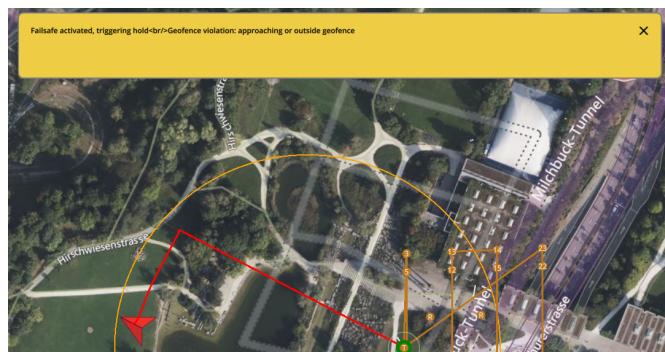


Abbildung 4.6.: QGroundControl GeoFence: Gelber Kreis stellt den GeoFence dar, den die Drohne nicht verlassen kann. Rote Linie zeigt den zurückgelegten Weg

## 4.6. Zusammenfassung

Für den angedachten autonomen Flugmodus eignet sich der Mission-Mode der Drohne nicht. Trotzdem sind Eigenheiten in der Flugsteuerung betrachtet wurden, die später wiederverwendet werden können. Ein Flug sollte grundsätzlich nur innerhalb eines GeoFence gestartet werden, um ein Entkommen der Drohne zu verhindern. Auch sind ROI hilfreich bei der Detektierung von Hindernissen.

#### 4. Einführende Flugtests: Misson-Mode



Abbildung 4.7.: QGroundControl Missionplaner-ROI: Während einer Mission zeigt die Drohne mit ihrem Bug in Richtung der mit „R“ markierten ROI

# 5. Vorbereitung zur Umsetzung von Avoidance

Um die Avoidance-Software umzusetzen, wird diese vorher in verschiedenen Modi erprobt. Dazu zählt eine Simulation. Aus dieser können die Strukturen zum Einsatz mit realer Hardware übernommen werden. Als Zwischenschritt wird die HIL-Simulation aus dem vorhergehenden Projektteil überarbeitet.

## 5.1. Avoidance als Simulation in Software

Das Kapitel spaltet sich auf in die Beschreibung und nachfolgend die programmiertechnischen Schritte zur Umsetzung.

### Beschreibung der Umsetzung von Avoidance

Das Avoidance-Modul arbeitet eng mit der Simulationssoftware „Gazebo“ zusammen. In der Dokumentation [4] ist die Simulation umfangreich dokumentiert, die Umsetzung des Projektes auf reale Hardware jedoch kurz gehalten.

Ein vollständiges Linux Ubuntu 20.04 wird benötigt, um die Software per Simulation verwenden zu können. Die Installation beinhaltet eine komplette ROS-Distribution mit graphischen Tools. Mithilfe mehrerer, sich gegenseitig aufrufenden, „Launch-Files“ wird die Simulation instrumentiert und, aufeinander abgestimmt, gestartet. Ohne weiteres Zutun muss nur das Hauptscript zur jeweiligen Simulation aufgerufen werden. Folgend aufgelistet die vier primären Software Bestandteile:

**Simulator:** *Gazebo*, erzeugt virtuelle Umgebung in der sich eine simulierte Drohne befindet, kann direkt mit dem Flugcontroller kommunizieren

**Flugcontroller:** PX4-Flightstack, Software zur Steuerung einer Drohne, ohne echte Hardware wird ein Flugcontroller simuliert der eine Konsole und Netzwerkanbindungspunkte (via MAVLink) bereitstellt

**mavros:** MAVLink-zu-ROS-Übersetzung, siehe [3, Kapitel 5.2]

**Avoidance:** Je nach gewähltem Programm wird der *Local Planner*, *Global Planner*, oder *Safe Landing Planner* gestartet. In diesem Projekt wird der *Local Planner* erprobt.

Zusätzliche Programme, die beim Ausführen des *Local Planner* gestartet werden, sind:

**rqt-reconfigure:** Feineinstellungen zum Algorithmus *Local Planner* (bspw. bevorzugte Richtungskorrektur, minimale Größe erkannter Hindernisse)

## 5. Vorbereitung zur Umsetzung von Avoidance

**rviz:** Visualisierung der Drohne im 3-Dimensionalen Raum, dargestellt wird Position der Drohne, ihres Ursprungs, Zielposition, geplante Flugroute, Punktwolkendarstellung von Hindernis, eventuell Blaupause von Hindernis (Abhängig von gewählter Umgebung)

Für die Untersuchung der gegebenen Situation wird zusätzliche Software verwendet:

**topic-explorer:** Detaillierte Informationen zu Nachrichten im ROS-Umfeld

**tftree:** Verknüpfungen im Transformationsbaum zur Bestimmung der Position der Drohne und derer Peripherie im Bezug zum Ursprung

Die Interaktion der Bestandteile ist in Abbildung 5.1 noch einmal dargestellt. Auf rechter Seite gezeichnet sind die Bestandteile in der „ROS-Umgebung“, welche in die Endanwendung übernommen werden können. Die vollen Aufgaben des Simulators wurden eingekürzt, sie sind zu lesen in Kapitel 2.2.3 und [3, Kapitel 3.4.1]. Zusätzlich stellt der Simulator Kamerabilder bereit, je nach gewähltem Modus direkt Tiefenbilder oder Stereobilder, welche weiter verarbeitet werden müssen.

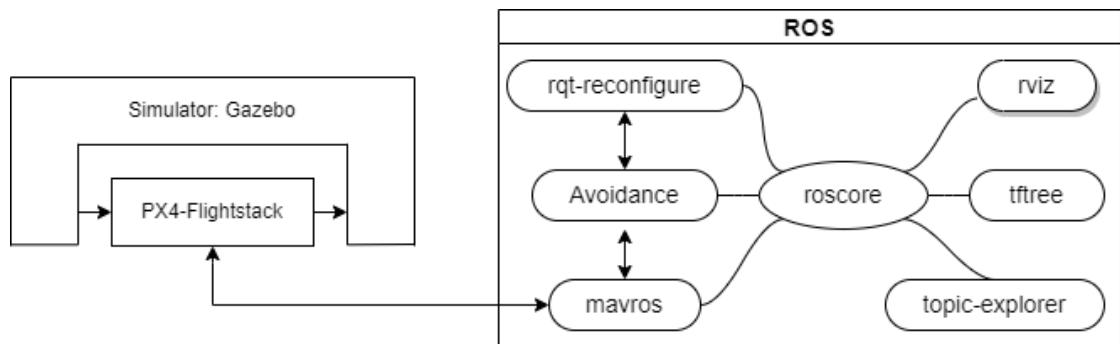


Abbildung 5.1.: Systemaufbau der Simulation von Avoidance als Software-Simulation:  
Links Simulator (Gazebo) und Flugcontroller (reine Software), Rechts Bestandteile des ROS

Eine erste Überprüfung des Algorithmus wird die Simulation wie in der Anleitung<sup>1</sup> beschrieben durchgeführt. Einige Parameter wurden zur besseren Übersichtlichkeit angepasst, sodass:

- *Gazebo* mit graphischer Oberfläche startet
- Die korrekte Punktwolken-Topic vom *Local Planner* abonniert wird
- Die Punktwolke der Kamerabilder noch einmal gefiltert wird

Die Anfangsbedingungen der Simulation sind eine vor Bäumen stehende Drohne. Das Bild aus der Simulation ist abgedruckt im Anhang unter A.4. Von der Drohne werden 2 virtuelle Kamerabilder erzeugt, aus welchen eine Punktwolke gebildet wird. Die Punktwolke ist in Bild 5.2 links dargestellt. Der Algorithmus zur Erzeugung der Punktwolke

<sup>1</sup><https://github.com/PX4/PX4-Avoidance>[4]

## 5. Vorbereitung zur Umsetzung von Avoidance

ist anfällig für Rauschen verursacht durch minimale Bewegungen in den Kamerabildern, sodass im Bild Artefakte enthalten sind. Deshalb wurde die Punktfolke noch einmal mit einem VoxelGrid<sup>2</sup> gefiltert, abgebildet in 5.2 rechts. Für die Verwendung der Punktfolke mit Avoidance macht das Filtern keine Auswirkung, der *Local Planner* kommt auch mit den ursprünglichen Daten der Punktfolke zurecht.

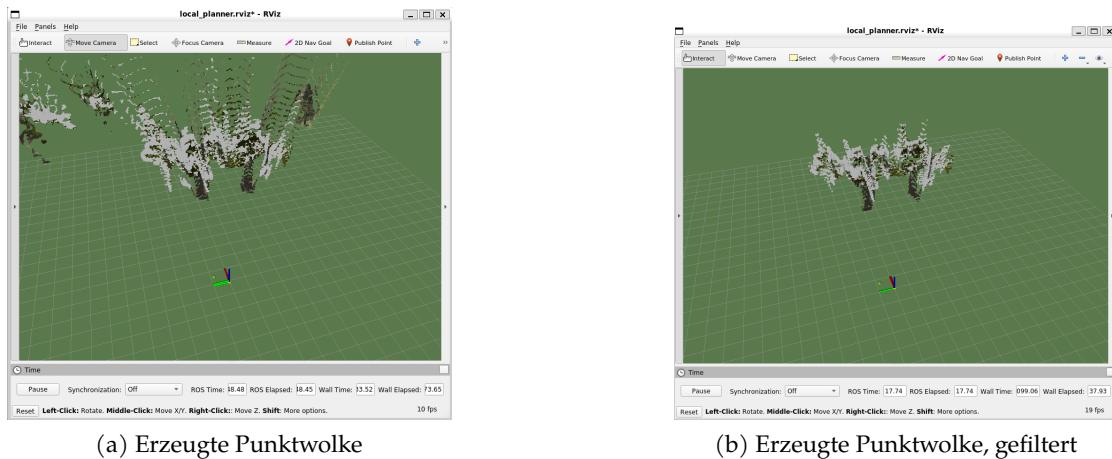


Abbildung 5.2.: Punktfolkendarstellung in *rviz* generiert aus simulierten Bildern

Zuletzt ist der Transformationsbaum in Bild 5.3 abgedruckt. Eine Transformation ist notwendig, um die Position von Objekten (bspw. der Drohne im Raum, Kameras an der Drohne) relativ zu ihrem Bezugspunkt/ Ursprung darzustellen. Die Position der Drohne wird gekennzeichnet durch den Knoten „fcu“ und ist relativ zum Knoten „local\_origin“. Um alle bekannten Punkte im Raum anpeilen zu können (von einem bekannten Punkt aus), sollten alle Knoten miteinander verknüpft sein. Jedoch besteht der Transformationsbaum hier aus 4 einzelnen Strängen. Dies ist der Konfiguration von *mavros* geschuldet, die wiederum in *Avoidance* integriert ist. Die Themen „map“ und „odom“ werden von *mavros* zur Navigation genutzt. Die Transformation „base\_link“ wurde in einer neueren Version von *mavros* eingeführt und soll den Mittelpunkt eines Roboters darstellen, wird aber im Projekt nicht genutzt<sup>3</sup>. Sowohl *mavros* als auch *Avoidance* können mit dem bestehenden Transformationsbaum arbeiten. Wird ein neues Objekt (bspw. Darstellung einer Punktfolke) im Koordinatensystem erzeugt, muss auch eine entsprechende Transformation angelegt werden. Für die Tiefenbilder kommt hier das Thema „camera\_link“, relativ zur Position der Drohne, zum Einsatz.

---

<sup>2</sup>[http://wiki.ros.org/pcl\\_ros/Tutorials/VoxelGrid%20filtering](http://wiki.ros.org/pcl_ros/Tutorials/VoxelGrid%20filtering)[6]

<sup>3</sup><https://www.ros.org/reps/rep-0105.html#base-link>

## 5. Vorbereitung zur Umsetzung von Avoidance

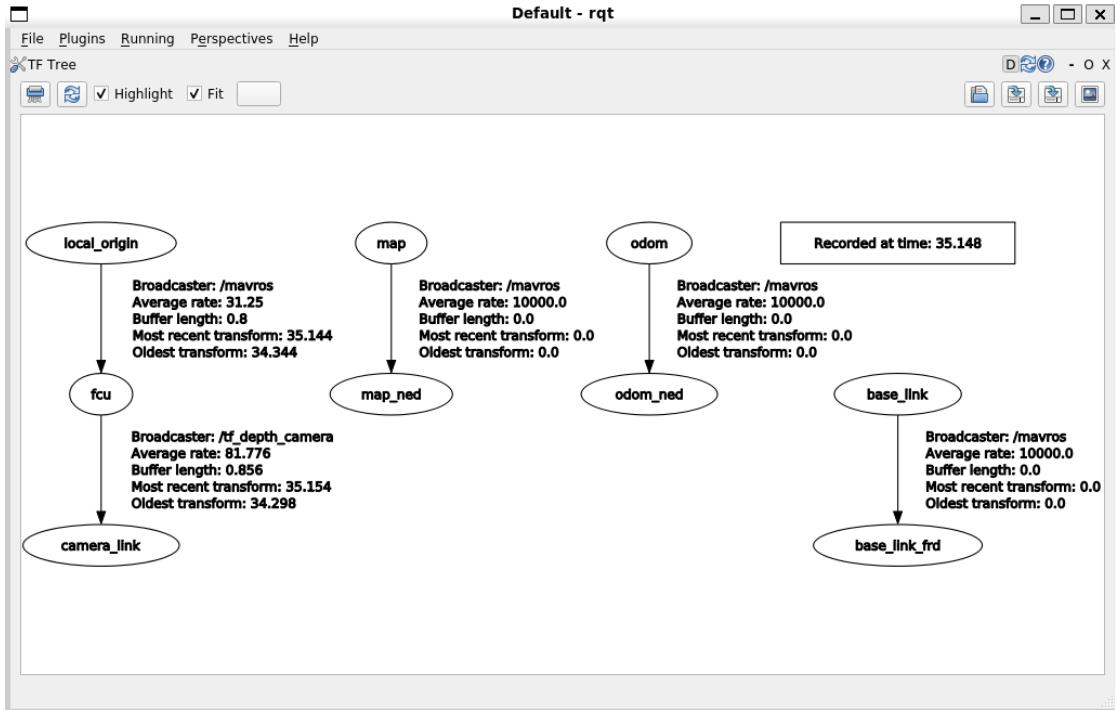


Abbildung 5.3.: Transformationsbaum tftree bei Simulation von *Local Planner*

## Programmierung der Umsetzung von Avoidance

Im GitHub[4] ist eine Anleitung zur Installation von *Avoidance* gegeben. Diese sieht nicht vor in einem Container betrieben zu werden, auch ist die Performance meist nicht ausreichend. Stattdessen wird ein separater Computer benötigt. Im Zweifallsfall funktioniert *Avoidance* doch auch mit den in GitHub bereitgestellten Containern. Es müssen Pakete mit graphischen Anwendungen (bspw. *rviz*, *rqt-reconfigure*) jedoch noch manuell installiert werden, wie in den Dockerfiles von [3, Kapitel 5]<sup>4</sup> gezeigt.

Anschließend muss der PX4-Flightstack<sup>5</sup> heruntergeladen werden (bisher im ersten Teil der Arbeit umgangen). Es sind weitere Schritte notwendig, um die Umgebungsvariablen für *Gazebo* und diesen zu setzen. Im Rahmen dieser Arbeit wurden die Schritte als Script zusammengefasst, abgedruckt in Listing A.2. Das Script muss aus dem Verzeichnis */catkin\_ws/* ausgeführt werden und der PX4-Flightstack muss sich im Verzeichnis */catkin\_ws/src/PX4-Autopilot* befinden.

<sup>4</sup>siehe [https://github.com/aur20/T3000-autonomous\\_drone/tree/airsim\\_avoidance](https://github.com/aur20/T3000-autonomous_drone/tree/airsim_avoidance)

<sup>5</sup><https://github.com/PX4/avoidance.git>

## 5.2. Avoidance als Simulation mit Hardware

Als nächste Stufe der Simulation kommt der Simulator „AirSim“ von Microsoft zum Einsatz. Mit *AirSim* stehen die gleichen Möglichkeiten zur Simulation wie mit *Gazebo* zur Verfügung. Allerdings, kann *AirSim* für das Projekt nur mit einem Windows PC zum verwendet werden. Dadurch ergeben sich Beschränkungen, wodurch *AirSim* keine Verbindung zum simulierten Flugcontroller aufbauen kann. Jedoch kann der reale Flugcontroller wie in [3, Kapitel 5] als HIL-Simulation betrieben werden. Die Veränderungen im Aufbau für die Simulation sind in Bild 5.4 rot markiert, die Funktionen der Bestandteile bleiben die gleichen wie in Kapitel 5.1. Hinzu kommt der Bordcomputer, der eine netzwerkgebundene Kommunikation mit dem Flugcontroller erlaubt und der Knoten „*AirSim-Node*“, durch den Bilder aus der Simulation ausgelesen werden können. Dabei entfällt das Zusammenfügen zweier Stereobilder, denn die Simulation liefert direkte Tiefenbilder.

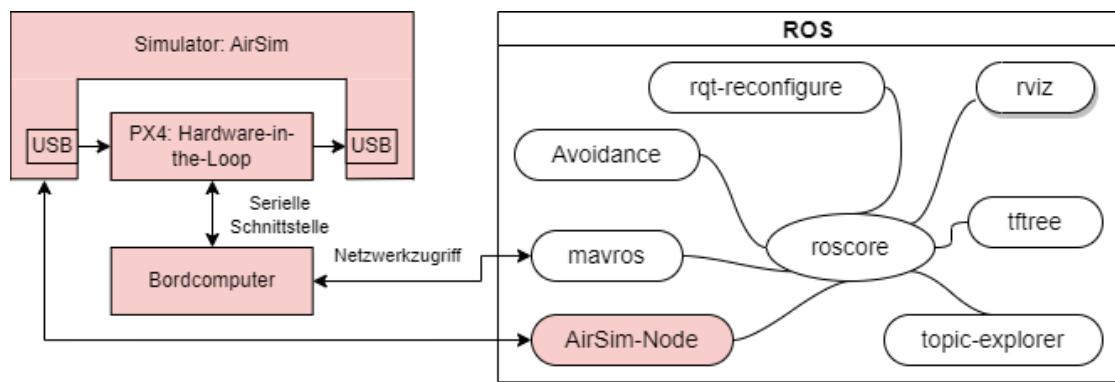


Abbildung 5.4.: Systemaufbau der Simulation von Avoidance als HIL-Simulation: Links Simulator (AirSim) und Flugcontroller (HIL), Rechts Bestandteile des ROS

Zur Änderung der Funktionalität werden die *Launch-Files* und „*airsim\_ros\_pkgs*“ (Bibliothek zur Drohnensteuerung im *AirSim*-Simulator) nacheinander angepasst:

1. *local\_planner\_stereo.launch*:
  - Umbenannt zu *local\_planner\_hitl\_airsim.launch*
  - Entfernen der Funktion von Disparitätsbildern, da Anwendung keinen ersichtlichen Zweck erfüllt
  - Unteraufruf von *avoidance\_hitl\_airsim.launch* anstatt *avoidance\_sitl\_stereo.launch*
2. *avoidance\_sitl\_stereo.launch*:
  - Umbenannt zu *avoidance\_hitl\_airsim.launch*
  - Entfernen der Stereo-Bildverarbeitung, stattdessen Berechnung der Punktwolkendarstellung durch Aufruf von eigens kompiliertem *depth\_image\_proc* aus Tiefebild von *AirSim*

## 5. Vorbereitung zur Umsetzung von Avoidance

- Entfernen der Transformation von „fcu“ auf „camera\_link“, da Transformation von Punktwolkendarstellung noch nicht bekannt
- Unteraufruf von *avoidance\_hitl\_airsim\_mavros.launch* anstatt *avoidance\_sitl\_mavros.launch*

### 3. *avoidance\_sitl\_mavros.launch*:

- Umbenannt zu *avoidance\_hitl\_airsim\_mavros.launch*
- Entfernen Startvorgang des „PX4 SITL“ (Software Simulation des Flugcontrollers)
- Ändern des Parameters „fcu\_url“ auf lokale IP-Adresse des Onboard-Computers
- Ändern des Parameters „gcs\_url“ auf lokale IP-Adresse des Simulations-PC
- Entfernen Startvorgang von *Gazebo*
- Einfügen Startvorgang der „AirSim-Node“ (Bestandteil der *airsim\_ros\_pkgs*) zur Darstellung eines simulierten Gefährts, dabei noch wichtige Anpassungen: das erzeugte Tiefenbild muss im Untertopic wie die zugehörige „camera\_info“ vorzufinden sein; der Ursprung aller Ortstransformationen, gennant „world\_frame\_id“ wird entsprechend an *local\_origin* angeknüpft

### 4. Einstellungen zu *AirSim*:

- Anlegen eines „vehicle“, hier eine Drohne *PX4* die über USB mit der realen Hardware verbunden ist
- Anlegen einer Kamera innerhalb der Drohne, hier vom Typ „DepthPlanar“, gennant „mk1“

### 5. Einstellungen zu *airsim\_ros\_pkgs*:

- Aufbau einer Verbindung zum Windows-Rechner mit *AirSim-Simulator*
- Anhängen des Transformationsbaum von *AirSim* an Transformationsbaum von *mavros* um Transformation der Punktwolke (aus Kamerabild) zur Drohne zu ermöglichen
- Ausgliedern generierter Tiefenbilder aus Transformationsbaum von *AirSim*
- Eingliedern generierter Tiefenbilder in Transformationsbaum von *mavros*, dazu wurde die Punktwolke noch entsprechend transformiert, siehe Bild

Als Simulation von *AirSim* wurde die „Blocks“-Umgebung gewählt, eine einfache Welt mit wenigen Objekten. Die Ausgangssituation ist dargestellt im Anhang unter A.5. Dabei steht die Drohne vor einem großen Quader. Rechts vom Quader befindet sich eine Kugel. Probleme die bei der Durchführung durchlaufen wurden, umfassen unter anderem, dass *AirSim* von einem Koordinatensystem im Format „ENU“ (East-North-Up) als die Interpretation der Welt, in der sich die Drohne befindet, ausgeht. Von *mavros* wird die Drohne hingegen im Format „NED“ (North-East-Down) beschrieben. Somit ist das erfasste Kamerabild nicht vor, sondern rechts der Drohne und steht Kopfüber.

## 5. Vorbereitung zur Umsetzung von Avoidance

Die weitere Entwicklung wurde an diesem Punkt aufgrund mangelnder Zeit eingestellt. Eine Simulation ist bereits vorhanden und lauffähig, der Zwischenschritt *AirSim* wird übersprungen. Bisherige Entwicklungen sind hinterlegt im GitHub<sup>6</sup>.

### 5.3. Machbarkeitsstudie Stereokamera mit Raspberry Pi

Vor der Anschaffung eines Kameramoduls für den Raspberry Pi, soll die Nutzbarkeit bewertet werden. Die Akzeptanzkriterien sind:

- flüssige Ausgabe von Punktwolke (Avoidance benötigt 10 – 20 Frames per Second (FPS))
- Erfassung großer Gegenstände, kleine Texturen können je nach Kamera und Lichtverhältnissen nicht erfasst werden
- Erfassung von Gegenständen im Nahbereich vor der Kamera, weitläufiger Hintergrund kann vernachlässigt werden

ROS stellt eine Beispiel-Videoaufnahme mit einer Auflösung von 640x480 Pixeln und 15 FPS Bildwiederholrate bereit, die von 2 Kameras als linke und rechte Perspektive aufgenommen wurde. Diese wird als Referenz für die Tests verwendet.

#### Durchführung

ROS stellt die notwendige Software zur Verfügung. Mit dem Programm *stereo\_image\_proc* können 2 Bilder (Linkes und Rechtes Bild) zu einem Tiefenbild zusammengeführt werden.

Die Software ROS kann als komplettes Packet von Docker<sup>7</sup> bezogen werden. Für den RPI steht sie in der Version Noetic sowohl für *arm32* als auch *arm64v8* zur Verfügung, die Version wird entsprechend des Betriebssystems ausgewählt. Allerdings sind die Images zur allgemeinen Verwendung bestimmt. Somit enthält keines derer, kompilierte Software mit Optimierungen, die bspw. die Grafikbefehle des RPI ausnutzen und so die Bildverarbeitung beschleunigen.

Optimierung steht im Kontext hier für eigene Kompilierung mit den C++-Compiler Flags `-march=native -O3`. Durch diese wird eine Anpassung auf den aktuell verwendeten Prozessor aktiviert. Speziell die Bildverarbeitung mittels OpenCV kann durch parallele Verarbeitung mit sog. Streaming-Befehlssätzen, ähnlich einer Grafikkarte, beschleunigt werden.

Um die Rechenkapazität des RPI zu bewerten, wurden verschiedene Packete manuell kompiliert. Gleichzeitig wurden verschiedene Videoauflösungen zur Berechnung getestet. Verwendet wurde ein RPI Model 4 B mit offiziellem 64-Bit Betriebssystem (Raspberry Pi OS). Der Quellcode zum Aufsetzen des Tests ist hinterlegt im GitHub<sup>8</sup>. Das Vorgehen

<sup>6</sup>[https://github.com/aur20/T3000-autonomous\\_drone/tree/airsim\\_avoidance](https://github.com/aur20/T3000-autonomous_drone/tree/airsim_avoidance)

<sup>7</sup>[https://hub.docker.com/\\_/ros/tags](https://hub.docker.com/_/ros/tags)

<sup>8</sup>[https://github.com/aur20/T3000-autonomous\\_drone/tree/rpi\\_ros\\_bench\\_stereo](https://github.com/aur20/T3000-autonomous_drone/tree/rpi_ros_bench_stereo)

## 5. Vorbereitung zur Umsetzung von Avoidance

zum Durchführen der Tests ist beschrieben unter<sup>9</sup>. Tabelle 5.1 zeigt die Ergebnisse in Bildrate je Einstellung und Auflösung.

Tabelle 5.1.: Benchmark zum Programm *stereo\_image\_proc* auf dem RPI

Versuch	Auflösung normal (640x480)	Auflösung halbiert (320x240)
Standard Pakete	6.3	14.7
Optimiertes OpenCV	6.9	14.7
Optimiertes OpenCV, vision_opencv und image_pipeline	6.7	14.6

### Auswertung

Selbst die Standardpakete erreichen gute Ergebnisse. Die internen Mechanismen beinhalten anscheinend bereits schon die meisten Optimierungen<sup>10</sup>. Zusätzlich wurden von den ursprünglichen Entwicklern weitere Optimierungen im Zusammenhang mit ROS vorgenommen, die den Transfer von Bildinformationen verbessern.

Die Leistung des RPI ist nicht ausreichend um hochauflösende Bilder zu verarbeiten, aber dies wird auch für das Projekt nicht benötigt. Mit dem verwendeten Format wird das Maximum an Bildrate erreicht, es könnte also ein noch wenig besseres Format verwendet werden.

Der entstandene Aufwand steht in keinem Verhältnis zum Gewinn, in der finalen Anwendung können Standardpakete verwendet werden.

## 5.4. Beschaffung Hardware Stereokamera

Der RPI 4 hätte hardware-technische Unterstützung mehrere Kameras anzuschließen. Allerdings besitzen die Modelle A und B nur immer einen notwendigen Header. Alternative Boards wurden von folgenden Projekten entwickelt:

- Stereopi<sup>11</sup>
- Arducam<sup>12</sup>

Entscheidender Faktor bei der Beschaffung im Rahmen dieses Projektes ist der Preis, weshalb ein Stereopi gewählt wurde. Zusätzlich wird in jedem Fall ein leistungsstarker

<sup>9</sup>[http://wiki.ros.org/stereo\\_image\\_proc/Tutorials/ChoosingGoodStereoParameters](http://wiki.ros.org/stereo_image_proc/Tutorials/ChoosingGoodStereoParameters)  
<sup>10</sup>siehe „Advanced SIMD“<https://en.wikipedia.org/wiki/AArch64>

<sup>11</sup><https://stereopi.com/>

<sup>12</sup><https://www.arducam.com/product-category/stereo-vision-cameras/>

## 5. Vorbereitung zur Umsetzung von Avoidance

RPI benötigt. Der aktuelle *Stereopi v2* benötigt ein *Compute Module 4 (CM4)*, eine kleine Platine die als Aufsatz gedacht ist. Beide Gerät sind in 5.5 abgebildet.



(a) Raspberry Pi Compute Module 4  
(<https://www.raspberrypi.com/products/compute-module-4/?variant=raspberry-pi-cm4001000>)



(b) Stereopi v2 ([https://wiki.stereopi.com/index.php?title>Main\\_Page](https://wiki.stereopi.com/index.php?title>Main_Page))

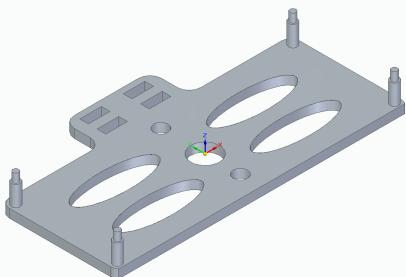
Abbildung 5.5.: Compute Module 4 und Stereopi

Da auf zur Zeit der Entwicklung keine CM4 auf dem Markt verfügbar sind, wird ein alternatives Board, welches Kompatibel zum RPI sein soll, verwendet. Zum Einsatz kommt ein *Radxa Compute Module 3*<sup>13</sup>.

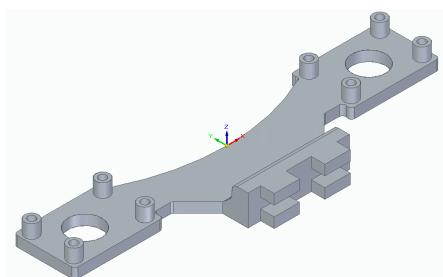
Zur Verbindung mit dem *Stereopi v2* wurden 2 Kameras vom Typ *Raspberry Pi Camera Module 2* besorgt. Diese bieten hochauflösende Bilder und Videos.

### 5.4.1. Halterung für Platine und Kameras

Um den *Stereopi v2* und die Kameras zu verbinden, wurde eine Halterung entworfen und per 3D-Druck gefertigt. Die Modelle werden in Bild 5.6 vorgestellt. Diese kann später an der Drohne montiert werden.



(a) Hauptkörper für Stereopi-Platine mit RPI Compute Module



(b) Steckaufsatz zur Halterung von 2 Kamera-Modulen

Abbildung 5.6.: Halterung Platine Stereopi und 2 Kameras

<sup>13</sup><https://wiki.radxa.com/Rock3/CM3>

### 5.4.2. Software auf dem Compute Module 4

Das CM4 enthält einen verlöteten Flash-Chip mit 32GB Festspeicher. Das Betriebssystem kann mithilfe eines weiteren Computers geschrieben werden<sup>14</sup>. Weitere Schritte zur Einrichtung wurden als Script im Anhang unter A.3 hinterlegt.

Um die Kameraanbindung des Stereopi zu aktivieren, ist eine speziell angepasste Software auf einem RPI notwendig. Diese ist nicht mit dem Compute Module 3 von Radxa kompatibel.

Versuche im Rahmen dieser Arbeit haben gezeigt, dass das gesamte Linux Betriebssystem bereits einen veralteten Kernel verwendet. Es wurde zur Entwicklung minimaler Aufwand betrieben und kein weiterer Support bereitgestellt. Es kann offiziell noch zur letzten Version des 4er-Kernels geupdatet werden. Inoffiziell war es im Rahmen der Arbeit möglich auch den 5er-Kernel zu installieren, jedoch entfielen sämtliche Hardware-Funktionen (z.B. war Wireless LAN (WLAN) nicht mehr verfügbar).

Für Anpassungen an Hardware-Funktionen muss bei Verwendung des 4er-Kernels die Datei „config.txt“ in der Boot-Partition editiert, und anschließend das Script „update\_extlinux.sh“ ausgeführt werden. Mit dem 5er-Kernel funktioniert dies nicht mehr, aber es gibt auch keine Dokumentation diesbezüglich.

Um die Kameraanbindung zu aktivieren, werden 2 Kanäle Inter-Integrated Circuit (I<sup>2</sup>C) und 2 Kamerainterfaces benötigt. Diese sind in den Konfigurationsdateien des Compute Module 3 enthalten, müssen aber noch entsprechenden Pins zugewiesen werden.

Das Betriebssystem-Image von Radxa enthält weiterhin keinen (aktivierten) Grafiktreiber um etwaige Kameras auslesen zu können. Entweder muss das offizielle CM3-IO Board verwendet werden (welches auch mehrere Kameraanschlüsse besitzt) oder diese tauchen nach Erkennen einer Kamera automatisch auf.

Im Rahmen dieser Arbeit war es nicht möglich die Kameras zu verwenden, zum Einsatz kommen nur die Ultraschallsensoren.

## 5.5. Einrichtung der Ultraschallsensoren

In diesem Kapitel wird die Verwendung der Ultraschallsensoren in Verbindung mit ROS beschrieben.

### 5.5.1. Erzeugen einer Punktwolke

Ein Beispiel zur Erzeugung von Punktwolken mit Python ist gegeben unter<sup>15</sup>, es erzeugt das Bild A.6. (Dieses bewegt sich gleichzeitig noch wellenartig.)

Das ROS-Punktwolkenformat definiert eine Klasse mit den Eigenschaften:

<sup>14</sup>siehe <https://wiki.radxa.com/Rock3/installusb-install-radxa-cm3-rpi-cm4-io>

<sup>15</sup>[https://docs.ros.org/en/noetic/api/rospy\\_tutorials/html/publish\\_\\_pointcloud2\\_8py\\_source.html](https://docs.ros.org/en/noetic/api/rospy_tutorials/html/publish__pointcloud2_8py_source.html)

## 5. Vorbereitung zur Umsetzung von Avoidance

### Header:

- stamp - aktueller Zeitstempel
- frame\_id - Kennzeichnung zur Transformation

**Fields:** Liste von Eigenschaften der Punkte, beinhaltet x-,y-,z-Position sowie Farbattribute

**Points:** Liste von Punkten, jeder Punkt im Format wie „Fields“

Der Zeitstempel wird mit jeder publizierten Nachricht auf die aktuelle Zeit angepasst. Für die Anwendung im Projekt ist muss die „frame\_id“ eine entsprechende Transformation von der lokalen Position der Drohne besitzen. Im einfachsten Fall wird die Drohne selbst als Ursprungspunkt des Bildes angenommen sodass keine Transformation benötigt wird. Somit entspricht die verwendete „frame\_id“ vorerst immer „fcu“.

Als Fields kommen die 3 Raumkoordinaten und ein Farbfeld (Achtung hier Abweichung vom Beispiel) zum Einsatz. Das Farbfeld dient der Darstellung in *rviz*, spielt aber für den Einsatzzweck keine Rolle.

Zur Erprobung wird eine Liste von Punkten angelegt. Diese befinden sich in 4 Ecken um den Koordinatenursprung („fcu“) und einmal direkt darüber. Die Abstände sind jeweils 1m in x- und y-Richtung sowie 1m in z-Richtung für den z-Punkt. Die Topic der Nachricht wird mit 4Hz veröffentlicht, was der Update Rate der Sensoren entspricht [3, Kapitel 4.4]. Das Script ist im Anhang A.1 abgedruckt, das Ergebnis zu sehen hier in Bild 5.7. Zur Zeichnung im Bild gibt es noch keine Transformation. Das Koordinatensystem „fcu“ wird daher als Ursprung angenommen (die Einstellung ist oben links in *rviz* zu sehen).

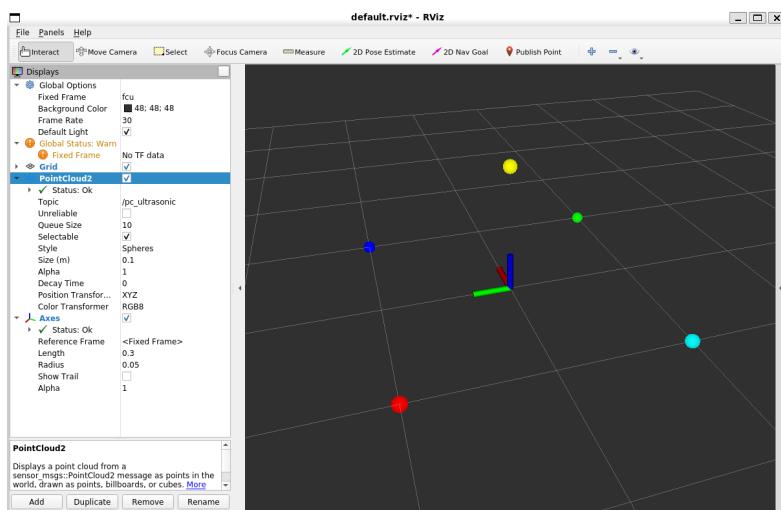


Abbildung 5.7.: Erprobung von Punktwolkenformat in *rviz*: die Punkte jeweils im Abstand von 1m zum Ursprung. Als Referenz wurde noch ein Koordinatensystem am Ursprung eingezeichnet, bei diesem entspricht x-Achse=rot, y-Achse=grün, z-Achse=blau.

## 5. Vorbereitung zur Umsetzung von Avoidance

Zur weiteren Erprobung wird die Simulation gestartet. Die 5 erzeugten Punkte sollten sich gemeinsam mit der Drohne bewegen. In Bild 5.8 wurde die Software-Simulation mit Stereokamera gestartet. Anschließend wurde das Script zur Erzeugung von Punktfolgen gestartet und in *rviz* hinzugefügt. Zu sehen ist die „Drohne“ (Ursprung der Drohne) mit Punkten, welche sich mitbewegen. Die Bewegung der Punkte ist durch die geringe Update Rate immer etwas später als die der Drohne. Dies sollte aber kein Problem sein, denn es spiegelt das reale Verhalten (die Punkte bleiben stehen aber die Drohne bewegt sich auf diese zu/weg) wieder.

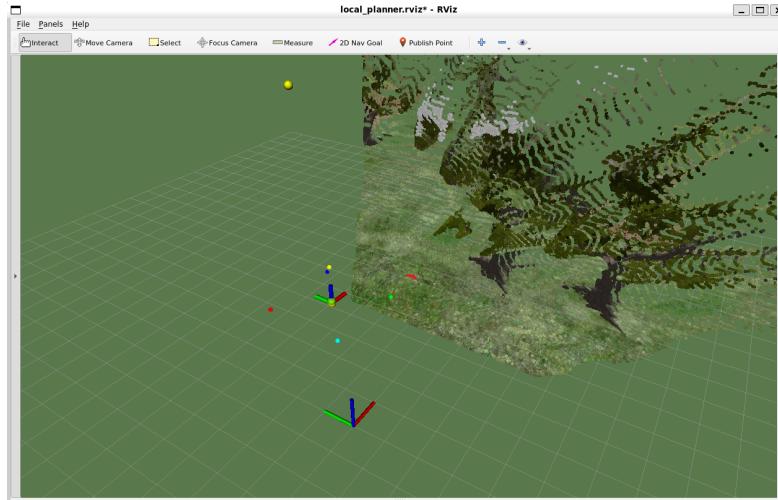


Abbildung 5.8.: Erprobung von Punktfolgenformat mit Simulator: die Drohne befindet sich in der Luft vor den Bäumen, um sie herum sind die erzeugten Punkte eingezeichnet.

### 5.5.2. Software der Sensoren

Der Algorithmus zum Auslesen der Sensoren wurde bereits für [3, Kapitel 4.4] entworfen, aber die Anwendung noch nicht dokumentiert. Der Code des Projektteils ist auf GitHub unter<sup>16</sup> hinterlegt. Er besteht aus 2 Teilen:

Die **Arduino Firmware** liest die Sensoren aus und stellt Daten per I<sup>2</sup>C zur Verfügung. Das Auslesen der Sensoren ist mit der notwendigen Verzögerung verbunden, um eventuellen Echos von Ultraschall vorzubeugen. In 5.1 dargestellt ist ein Ausschnitt aus der `loop()`-Schleife, die fortwährend immer durchlaufen wird. Der gezeigte Code kommt 4-mal vor, jeweils mit geänderten Pins der Sensoren (Zeile 187) und geänderten Indizes (Zeile 188-190). In der ersten gezeigten Zeile wird der jeweilige Sensor ausgelesen. Dazu wird mittels einer Arduino-internen Funktionen bis zum empfangenen Echo gewartet und die Zeit zurückgegeben. Anschließend wird der gefilterte Sensorwert berechnet.

<sup>16</sup>[https://github.com/aur20/T3000-autonomous\\_drone/tree/arduino\\_sensors](https://github.com/aur20/T3000-autonomous_drone/tree/arduino_sensors)

## 5. Vorbereitung zur Umsetzung von Avoidance

Der ungefilterte und gefilterte Wert werden jeweils in ein Array eingetragen. Zuletzt erfolgt das Warten, mit einer groben Annäherung: das Makro *PAUSE\_MEAS* steht für *60ms*. Davon abgezogen wird die Zeit, die bereits auf das Echo gewartet wurde in Millisekunden. Arduino selbst stellt eine Verzögerungsfunktion in Mikrosekunden bereit, dieses sollte jedoch nicht für Verzögerungen länger als „*a few thousand microseconds*“<sup>17</sup> verwendet werden, und funktioniert auch nur bis zu *16ms* Verzögerung. Um die Zeit von Mikrosekunden in Millisekunden umzurechnen, muss durch 1000 geteilt werden. Im Programm wird stattdessen 10 mal nach rechts geschoben, was einer Division durch 1024 entspricht. Die Verarbeitung von Schiebeoperationen ist wesentlich schneller als eine Division. Letztere kann im benötigten Zahlenspektrum von kleiner 30.000 zwar mit Integer Variablen umgesetzt werden, benötigt dann aber nach<sup>18</sup> immer noch bis zu *15ms*, was die Messfrequenz beeinflussen würde. Außerdem hat der Prozessor bereits Zeit mit der Berechnung des Filters verbracht, sodass die Wartezeit lediglich eine Annäherung an *60ms* ist, für das Ergebnis spielt dies keine Rolle.

---

```
187 t = measure_distance(A_TRIGGER, A_ECHO);
188 d = filter_median(0, t);
189 *(i2cResponseRaw) = t;
190 *(float*)(i2cResponseBuffer) = d;
191 delay(PAUSE_MEAS - (t >> 10));
```

---

Listing 5.1: Ausschnitt der Arduino Firmware: loop()-Schleife

Neben dem Auslesen der Sensoren stellt sich der Arduino als I<sup>2</sup>C-Slave zur Verfügung. Er reagiert auf einzelne zugesendete Buchstaben und Zahlen und sendet eine entsprechende Antwort. Derzeit implementiert sind die Funktionen wie in 5.2 aufgezeigt.

Tabelle 5.2.: Verfügbare Kommandos zum Auslesen der Sensordaten auf Arduino

Kommando	Anwort
eine der Zahlen 1-4	jeweiliger Sensorwert, gefiltert
Buchstabe „a“	alle Sensorwerte, gefiltert
Buchstabe „r“	alle Sensorwerte, ungefiltert

Das **Python Script** auf dem RPI gibt Messwerte aus oder speichert diese als CSV-Datei ab.

Derzeit stehen die Scripte *ultrasonic\_i2c\_reader.py* und *ultrasonic\_i2c\_csvwriter.py* zur Verfügung. Sie verhalten sich nahezu gleich. Bei ersterem kann durch die Eingabe eines

<sup>17</sup><https://www.arduino.cc/reference/en/language/functions/time/delaymicroseconds/>

<sup>18</sup><https://forum.arduino.cc/t/speed-of-math-operations-particularly-division-on-arduino/90726/5>

## 5. Vorbereitung zur Umsetzung von Avoidance

Zeichens ein bestimmter Wert an den Arduino gesendet werden. Die Antwort wird entsprechend auf der Konsole ausgegeben.

Beim zweiten Script kommt nur der Buchstabe „a“ zum Einsatz. Es wird die Antwort geparsst und mit aktuellem Zeitstempel in die Datei eingetragen. 5.2 zeigt einen Ausschnitt aus der Schleife des zweiten Scriptes. In Zeile 25 werden die Daten per I<sup>2</sup>C-Verbindung gelesen. Um die 16 Byte Binärdaten als Fließkommazahlen zu interpretieren wird das Packet „struct“ verwendet. Im Beispiel werden die Daten in Zeile 28 als Fließkommazahlen im Little-Endian Format interpretiert. Die Funktion *struct.iter\_unpack* liefert anschließend ein iterierbares Objekt („Each iteration yields a tuple as specified by the format string.“<sup>19</sup>). Um den Inhalt zu extrahieren wird eine Lambda-Funktion innerhalb der Funktion *map* verwendet, dies hat zur Folge dass jedes Tupel einzeln betrachtet werden kann. Es wird die jeweilige Zahl extrahiert und in einer Liste angelegt. Die Liste wird zusammen mit der derzeitigen Zeit jeweils ausgegeben und in die Datei geschrieben.

---

```
25     req = bus.read_i2c_block_data(addr, ord('a'), 16) # get all
26         → sensors
27     #print("Length of received data: %s Type: %s Data: %s" %
28         → (len(req), type(req), req))
29     t = time.time() - start_time
30     r = list(map(lambda a: a[0], struct.iter_unpack('<f',
31         → bytearray(req)))) # cast tuple to useful data
32     print([t] + r)
33     csvwriter.writerow([t] + r)
```

---

Listing 5.2: Ausschnitt des Python Scriptes zum Auslesen der Sensordaten

### 5.5.3. Einbinden der Sensoren

Als nächster Schritt wurden die vorliegenden Programme miteinander vereinigt. Vom letzten Script werden jeweils alle 4 Sensoren ausgelesen. Aufgrund der Ungenauigkeit der Messwerte, wurde der Wertebereich nochmals eingeschränkt und beträgt hier 3m. Größere Werte (beinhaltet Werte außerhalb der Reichweite, die mit 4m gesendet werden) werden schlachtweg ignoriert. Die Angaben für die Punktwolke werden auf Meter umgerechnet.

Das Projekt liegt separat auf GitHub<sup>20</sup> und kann innerhalb zukünftiger Container direkt eingebunden werden.

---

<sup>19</sup><https://docs.python.org/3/library/struct.html>

<sup>20</sup>[https://github.com/aur20/T3000-autonomous\\_drone/tree/rpi\\_ros\\_ultrasonic](https://github.com/aur20/T3000-autonomous_drone/tree/rpi_ros_ultrasonic)

## *5. Vorbereitung zur Umsetzung von Avoidance*

### **5.5.4. Verbesserung der Aufnahmegenauigkeit der Sensoren**

Um die Aufnahmegenauigkeit der Sensoren zu verbessern, könnte eine Interrupt gesteuerte Auslösung des Messvorgangs zum Einsatz kommen. Ein Timer könnte nach jeweils 60ms auslösen um die Messung und Berechnungen des Filters durchzuführen.

## 6. Erweiterte Flugtests: Avoidance

Für die Flugtests mit realer Hardware kommt wieder ein RPI Model 3B+ zum Einsatz. Das Einbinden der Ultraschallsensoren aus dem letzten Kapitel in das *Avoidance* Umfeld erfolgt durch das Ändern der Topic in der Startdatei „local\_planner.launch“

# A. Anhang

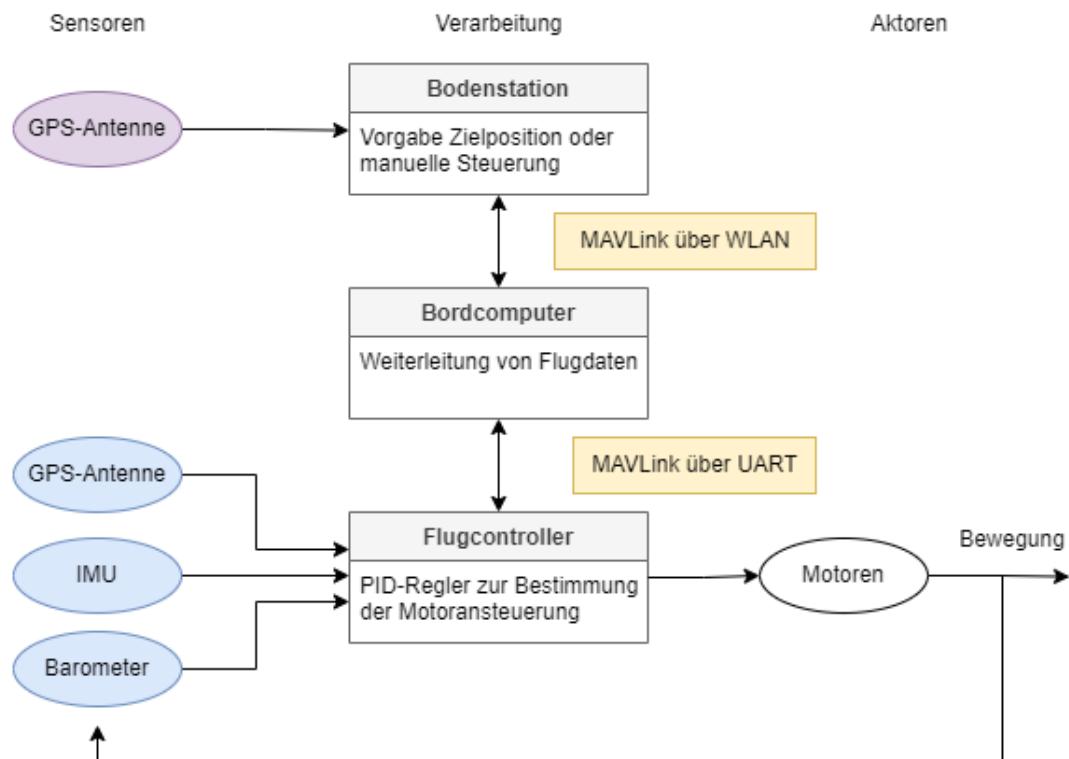


Abbildung A.1.: Beschreibung des Systems Drohne-Bodenstation: Grau hinterlegt die Bestandteile aus Tabelle 2.1; Gelb hinterlegt das jeweilige Kommunikationsprotokoll; Blau hinterlegt vorhandene Sensoren; Lila hinterlegt optionale Sensoren

## A. Anhang

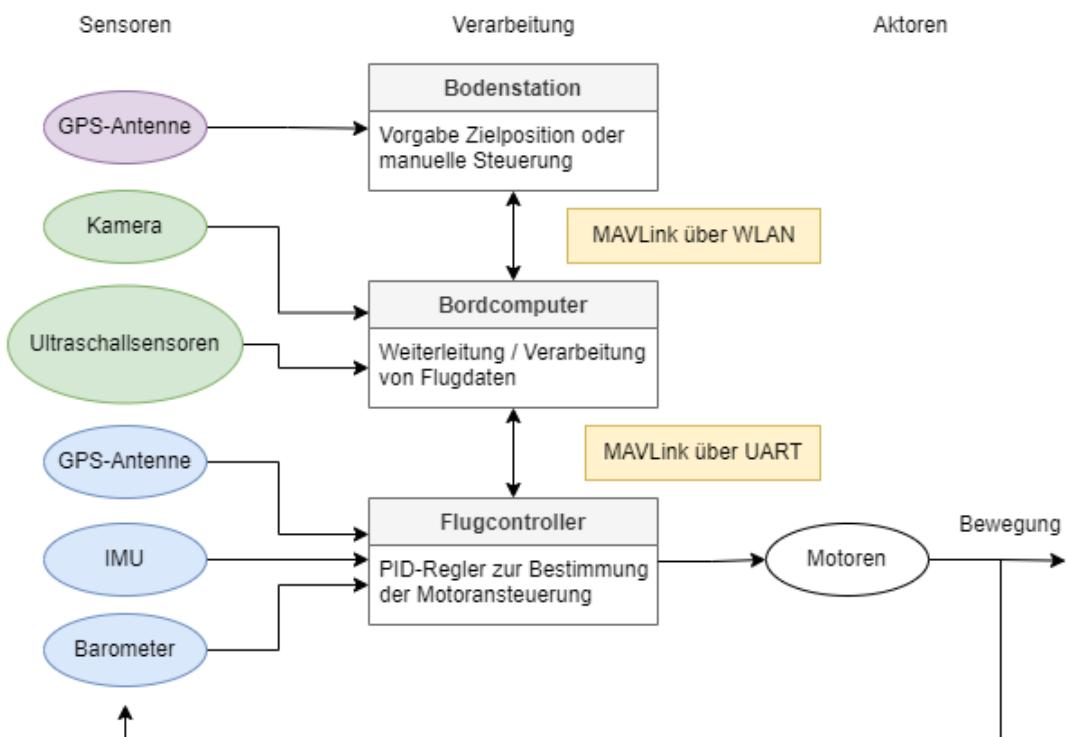


Abbildung A.2.: Beschreibung des erweiterten Systems Drohne-Bodenstation: Grün hinterlegt neu vorgesehene Sensoren

## A. Anhang

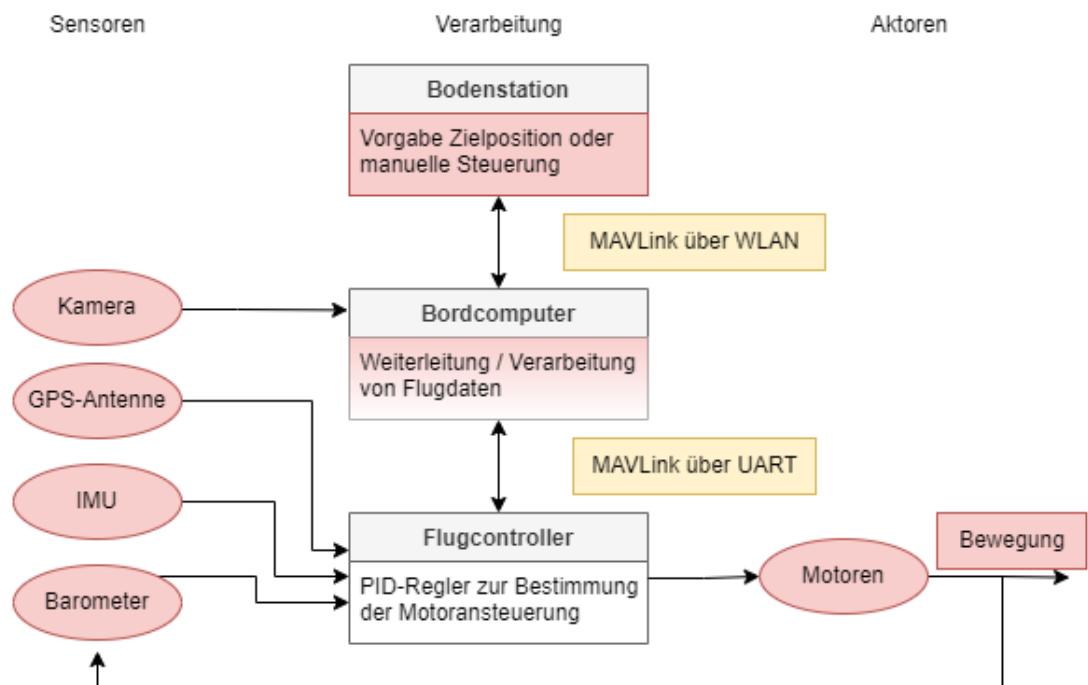


Abbildung A.3.: Beschreibung des Systems zur Simulation HIL: Hell-rot dargestellt die von der Simulation übernommenen Aufgaben

## A. Anhang

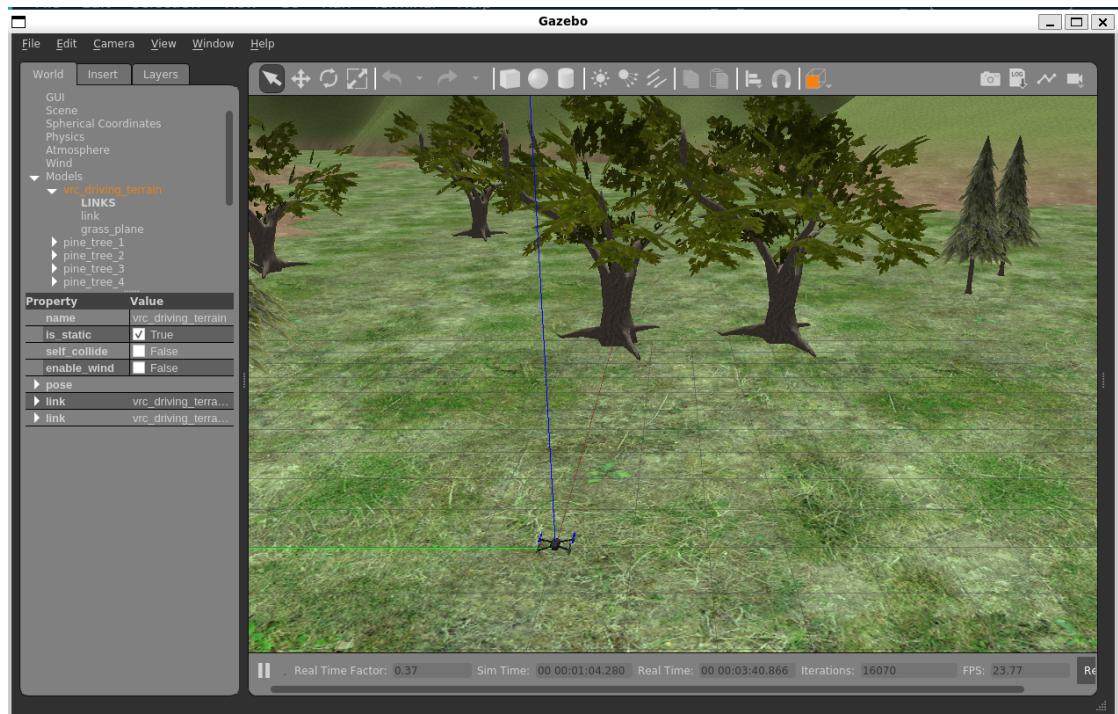


Abbildung A.4.: Erste Ansicht von Gazebo bei der Simulation von Avoidance: Relativ klein im Vordergrund die simulierte Drohne mit Koordinatenachsen (Rot, Grün, Blau), im Hintergrund die Welt mit Bäumen, auf linker Seite ein Konfigurationsmenü von Gazebo

## A. Anhang

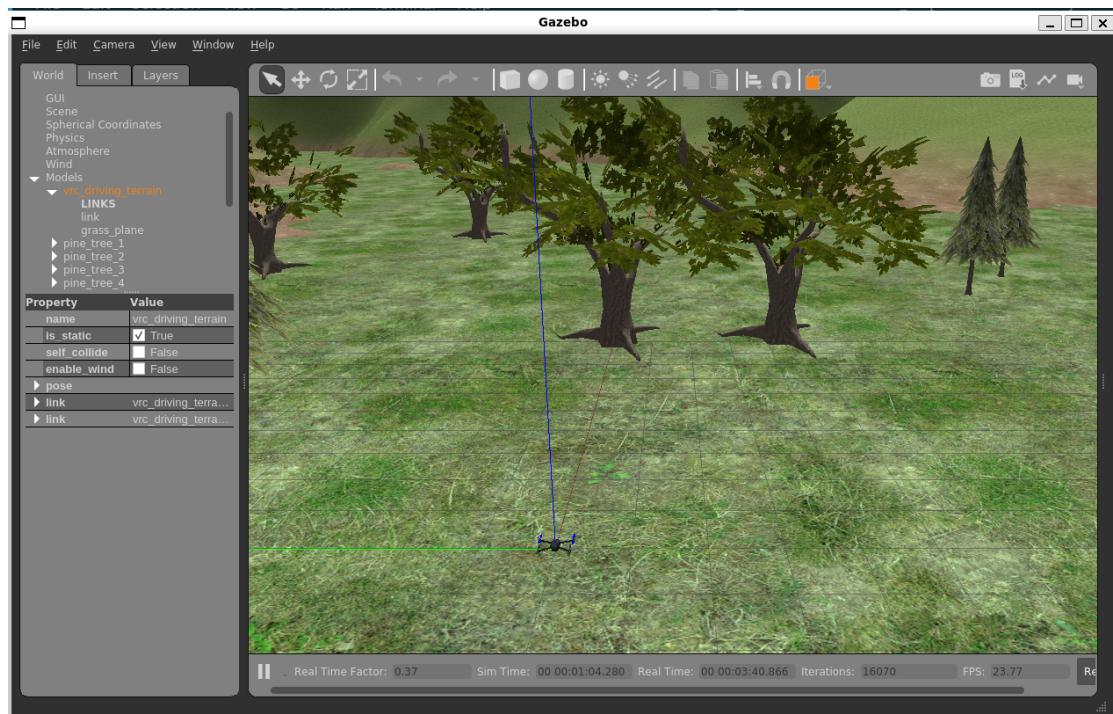


Abbildung A.5.: Erste Ansicht von AirSim bei der Blocks-Simulation: im Vordergrund die simulierte Drohne, im Hintergrund ein großer Quader und eine Kugel

## A. Anhang

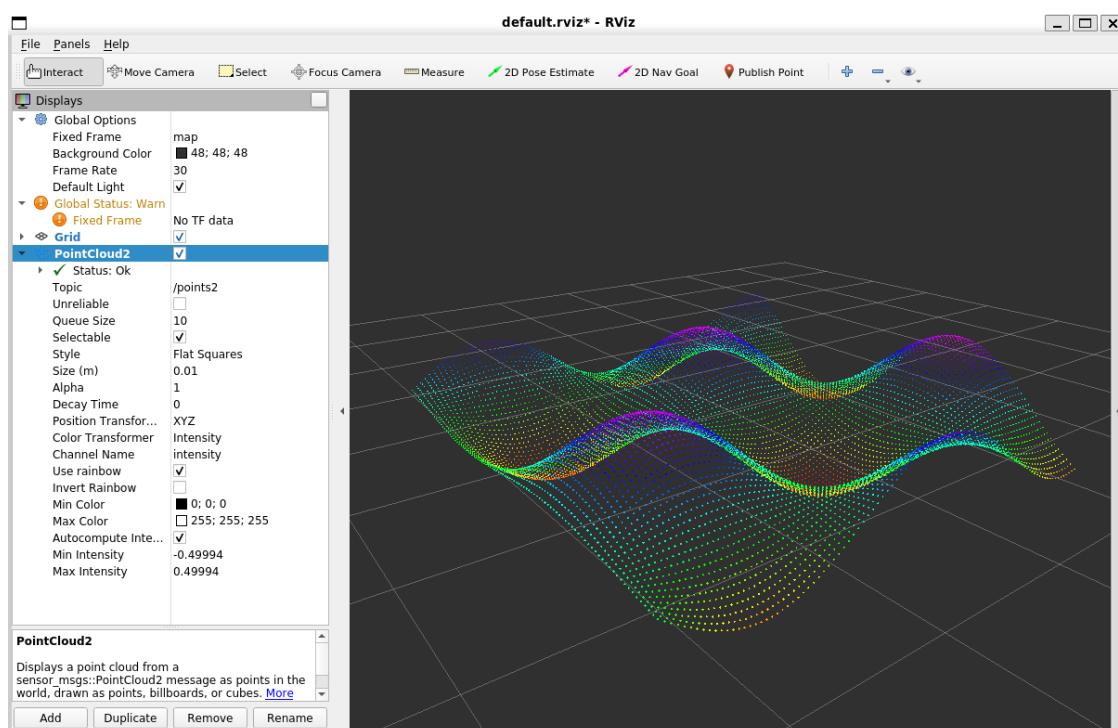


Abbildung A.6.: Vorgefertigtes Beispiel zur Punktfolke

```

1 #!/usr/bin/env python
2 import rospy
3 from sensor_msgs import point_cloud2
4 from sensor_msgs.msg import PointCloud2, PointField
5 import std_msgs.msg
6
7 if __name__ == '__main__':
8     rospy.init_node("ultrasonic_sensor_to_pointcloud")
9     cloud_pub = rospy.Publisher("pc_ultrasonic", PointCloud2,
10                                queue_size=4)
11     rate = rospy.Rate(4)
12
13     frame_id = rospy.get_param("frame_id", "fcu")
14     offsets = rospy.get_param("offsets", [0, 0, 0, 0])
15
16     header = std_msgs.msg.Header()
17     header.frame_id = frame_id
18     fields = [
19         PointField(name='x', offset=0, datatype=PointField.FLOAT32,
20                    count=1),
21         PointField('y', 4, PointField.FLOAT32, 1),
22         PointField('z', 8, PointField.FLOAT32, 1),
23         #PointField('rgb', 12, PointField.UINT32, 1),
24         PointField('rgb', 16, PointField.UINT32, 1),
25     ]
26
27     points = [
28         [1, 1, 0, 0xFF], # blue
29         [1, -1, 0, 0xFF00], # green
30         [-1, 1, 0, 0xFF0000], # red
31         [-1, -1, 0, 0x00FFFF], # cyan
32         [0, 0, 1, 0xFFFF00] # yellow
33     ]
34     while not rospy.is_shutdown():
35         header.stamp = rospy.Time.now()
36         pc = point_cloud2.create_cloud(header, fields, points)
37         cloud_pub.publish(pc)
38         rate.sleep()
39
40     rospy.spin()

```

---

Listing A.1: Eigener Beispielcode zur Veröffentlichung von 5 Punkten um Koordinatenursprung

---

```
1 THISD=${PWD}
2 cd src/PX4-Autopilot
3 DONT_RUN=1 make px4_sitl_default gazebo
4 # source ~/catkin_ws/devel/setup.bash      # (optional)
5 source Tools/simulation/gazebo-classic/setup_gazebo.bash $(pwd)
   ↳ $(pwd)/build/px4_sitl_default
6 export ROS_PACKAGE_PATH=$ROS_PACKAGE_PATH:$(pwd)
7 export
   ↳ ROS_PACKAGE_PATH=$ROS_PACKAGE_PATH:$(pwd)/Tools/simulation/gazebo-classic/sitl_gazebo
8 export GAZEBO_MODEL_PATH=${GAZEBO_MODEL_PATH}: \
   ↳ ~/catkin_ws/src/avoidance/avoidance/sim/models: \
10    ~/catkin_ws/src/avoidance/avoidance/sim/worlds
11 cd ${THISD}
```

---

Listing A.2: Setup-Script für Gazebo Simulation

---

```
1 #!/bin/bash
2
3 # execute:
4 # sudo bash ./setup.sh
5
6 # add radxa repo
7 export DISTRO=focal-stable
8 echo "deb http://apt.radxa.com/$DISTRO/ ${DISTRO%-*} main" | sudo tee -a
   ↳ /etc/apt/sources.list.d/apt-radxa-com.list
9 wget -O - apt.radxa.com/$DISTRO/public.key | sudo apt-key add -
10 # make sure we use radxa packages over Ubuntu packages
11 echo "Package: *"
12 Pin: release n=repo
13 Pin-Priority: 50" | sudo tee /etc/apt/preferences.d/radxa-conf
14 # install basics
15 sudo apt-get update
16 sudo apt-get install -y linux-4.19-rock-3-latest
17 sudo apt-get upgrade
18 sudo apt-get install rockchip-overlay
19
20 # install commons
21 sudo apt install curl screen
22
23 # install docker
24 curl -fsSL https://get.docker.com -o get-docker.sh
25 sh get-docker.sh
26 sudo usermod -aG docker $USER
```

---

Listing A.3: Setup-Script auf Stereopi

# Literatur

- [1] Dronecode Stiftung. (). Simulation | PX4 User Guide, Adresse: <https://docs.px4.io/main/en/simulation/> (besucht am 25.04.2023).
- [2] Microsoft Corporation, *Welcome to AirSim*, Microsoft, 14. Jan. 2023.
- [3] Markus Rein, „Erweiterung Bestehender Drohnen Um Eine Autonomflugfähigkeit,“ DHBW Stuttgart, 22. Jan. 2023.
- [4] Dronecode Stiftung, *Obstacle Detection and Avoidance*, PX4 Autopilot for Drones, 15. Jan. 2023.
- [5] H. Wirth und D. Helfenstein, „Erweiterung Einer Bestehenden Drohne Um Eine Autonomflugfähigkeit,“ DHBW Stuttgart, 23. Jan. 2022.
- [6] OpenRobotics. (). Documentation - ROS Wiki, Adresse: <https://wiki.ros.org/> (besucht am 10.03.2023).

# Abbildungsverzeichnis

4.1.	QGroundControl Missionplaner-Menu . . . . .	10
4.2.	QGroundControl Missionplaner-Wegpunkte . . . . .	11
4.3.	Auswertung Missionsplaner-Wegpunkte . . . . .	12
4.4.	QGroundControl Missionplaner-Rallypunkte . . . . .	13
4.5.	QGroundControl Missionplaner-Survey . . . . .	14
4.6.	QGroundControl GeoFence . . . . .	14
4.7.	QGroundControl Missionplaner-ROI . . . . .	15
5.1.	Systemaufbau der Simulation von Avoidance als Software-Simulation . . . . .	17
5.2.	Punktwolkendarstellung in <i>rviz</i> . . . . .	18
5.3.	Transformationsbaum tftree bei Simulation von <i>Local Planner</i> . . . . .	19
5.4.	Systemaufbau der Simulation von Avoidance als HIL-Simulation . . . . .	20
5.5.	Compute Module 4 und Stereopi . . . . .	24
5.6.	Halterung Platine Stereopi und 2 Kameras . . . . .	24
5.7.	Erprobung von Punktwolkenformat . . . . .	26
5.8.	Erprobung von Punktwolkenformat mit Simulator . . . . .	27
A.1.	Beschreibung des Systems Drohne-Bodenstation . . . . .	32
A.2.	Beschreibung des erweiterten Systems Drohne-Bodenstation . . . . .	33
A.3.	Beschreibung des Systems zur Simulation HIL . . . . .	34
A.4.	Ansicht Gazebo Simulator . . . . .	35
A.5.	Ansicht Gazebo Simulator . . . . .	36
A.6.	Vorgefertigtes Beispiel zur Punktwolke . . . . .	37

# Tabellenverzeichnis

2.1. Systemübersicht Drohne und Bodenstation . . . . .	3
5.1. Benchmark zum Programm <i>stereo_image_proc</i> auf dem RPI . . . . .	23
5.2. Verfügbare Kommandos zum Auslesen der Sensordaten auf Arduino . . . . .	28

# Akronyme

**CM4** Compute Module 4. 24

**CSV** Comma Separated Values, Textdatei mit Tabelleninhalt. 28

**FPS** Frames per Second. 22

**GPS** Global Positioning System. 7, 11

**HIL** Hardware-In-the-Loop. 5

**I<sup>2</sup>C** Inter-Integrated Circuit. 25

**IMU** Inertial Measurement Unit. 2

**MAVLink** Binäres Übertragungsprotokoll für ressourcenbeschränkte Umgebungen, bevorzugt auf Robotern und Drohnen eingesetzt. 7, 13, 16

**ROI** Region Of Interest. 6

**ROS** Robot Operating System, Framework mit Schnittstellen für Sensor/Aktor-Verknüpfungen. 2

**RPI** Raspberry Pi (Einplatinencomputer). 2

**SLAM** Simultane Lokalisierung und Kartierung der Umgebung. 7

**UART** Universal Asynchronous Receiver Transmitter. 4

**USB** Universal Serial Bus. 5

**WLAN** Wireless LAN. 3, 4, 5, 25