Studienarbeit T3200

Erweiterung bestehender Drohnen um eine Autonomflugfähigkeit

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

eingereicht von

Markus Rein

Abgabedatum: 2. 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

1. 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.

1.1. Verwendete Technologien

1.1.1. WLAN

1.1.2. GPS

1.2. Beschreibung der Drohne in den einzelnen Phasen

1.2.1. 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 1.1 aufgelistet.

Tabelle 1.1.: Systemübersicht Drohne und Bodenstation

	Drohne		
	Flugcontroller	Bordcomputer	Bodenstation
Funktion	Autopilot-Software liest Sensoren und steuert Motoren der Drohne	Bereitstellung WLAN-Netzwerk zur Verbindung von Autopilot und Bodenstation	Parametrierung und Steuerung der Drohne
Hardware	Pixhawk 4	Raspberry Pi 3B+	PC und/oder Smartphone
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 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 2.1. 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 Inertial Measurement Unit (IMU)) und Barometer, diese sind im Flugcontroller integriert, und der GPS-Antenne zur Verfügung.

1.2.2. 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 Raspberry Pi (Einplatinencomputer) (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

1.2.3. 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 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 4 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.

2. Stand der Technik

In diesem Kapitel werden die Grundlagen verwendeter Software erläutert.

2.1. 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 Mindesflughöhe 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.

2.2. ROS und Avoidance

Das Projekt „Obstacle Detection and Avoidance“[4], auf GitHub verfügbar als PX4-Avoidance[1], 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 2.3) 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 leisstungsstarke 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.

2.3. 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.

2.3.1. 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

[1] https://github.com/PX4/PX4-Avoidance

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 Markov-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.

2.3.2. Stereokamera

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

2.3.3. Optical Flow

In Bewegungsabläufen werden Objekten 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.

2.3.4. Punktwolkenformat

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 Punktwolke muss jedes Pixel abgetastet werden um als Koordinate im 3D-Raum dargestellt werden zu können. Das ROS beinhaltet sowohl

Progamme zur Stereoverarbeitung basierend auf OpenGL[2], als auch die Erzeugung von Punktwolken[3].

[2]siehe http://wiki.ros.org/stereo_image_proc
[3]siehe http://wiki.ros.org/depth_image_proc

3. Einführende Flugtests: Misson-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 3.1 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.

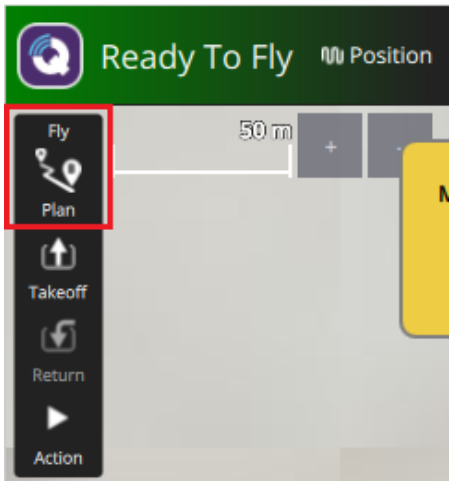


Abbildung 3.1.: QGroundControl Missionplaner-Menu wird durch Klick auf Rot umrahmten Button geöffnet

3.1. Flug auf gerade Linie

Für den einfachsten Fall lassen sich im Missionsplaner eine Liste von Wegpunkten anlegen, siehe dazu Bild 3.2. 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 3.3.

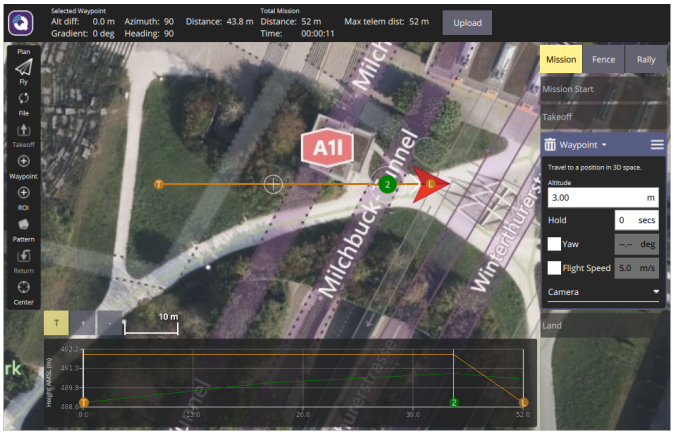


Abbildung 3.2.: QGroundControl Missionplaner-Wegpunkte

3.2. Rally-Kartierung

3.3. Flug mit GeoFence

3.4. Flug mit ROI

3.5. Zusammenfassung

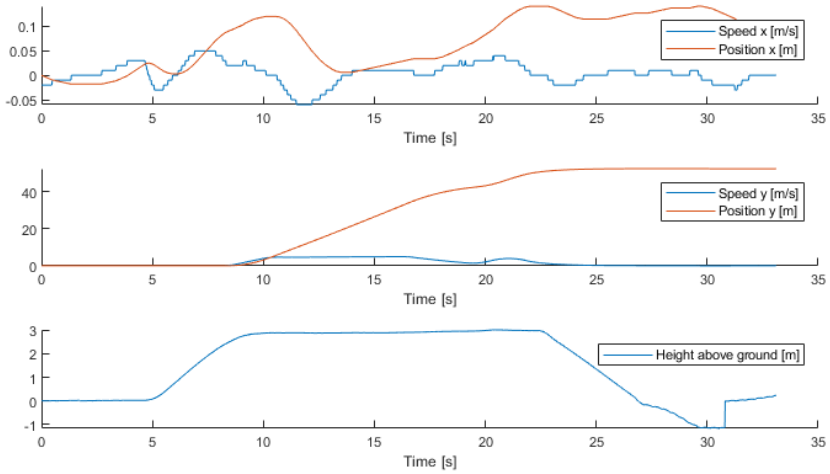


Abbildung 3.3.: Auswertung Missionsplaner-Wegpunkte: Diagramme geplottet von Matlab. In den oberen Diagrammen zu sehen sind die jeweilige x-/y-Geschwindigkeit und -Position während des Fluges. Positive x-Richtung zeigt nach Norden, positive y-Richtung nach Osten. Im unteren Diagramm die Flughöhe. Da der Simulator von einer flachen Welt ausging, die GPS-Karte aber eine Position in Östterreich annahm, wurde der Landepunkt tiefer als der eigentliche Boden berechnet. Die Drohne konnte diesen Fehler beim Landen bei ca. 30s ausgleichen und blieb sicher stehen.

4. 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.

4.1. Avoidance als Simulation in Software

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)

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 4.1 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 1.2.3 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.

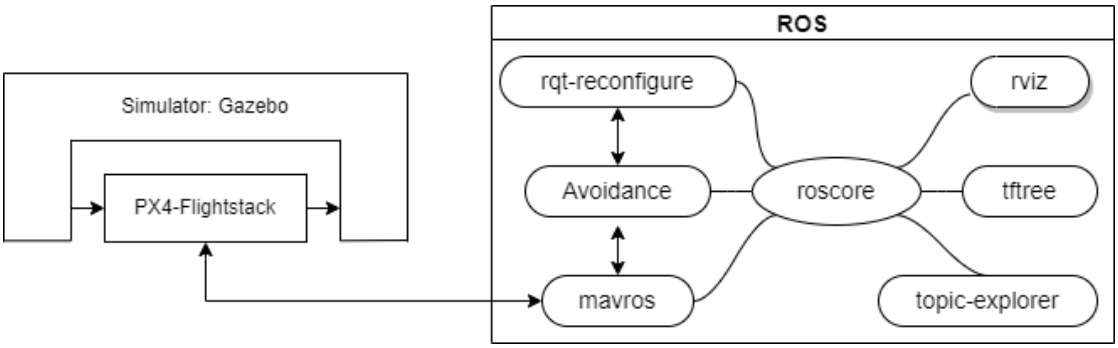


Abbildung 4.1.: Systemaufbau der Simulation von Avoidance als Software-Simulatio: Links Simulator (Gazebo) und Flugcontroller (reine Software), Rechts Bestandteile des ROS

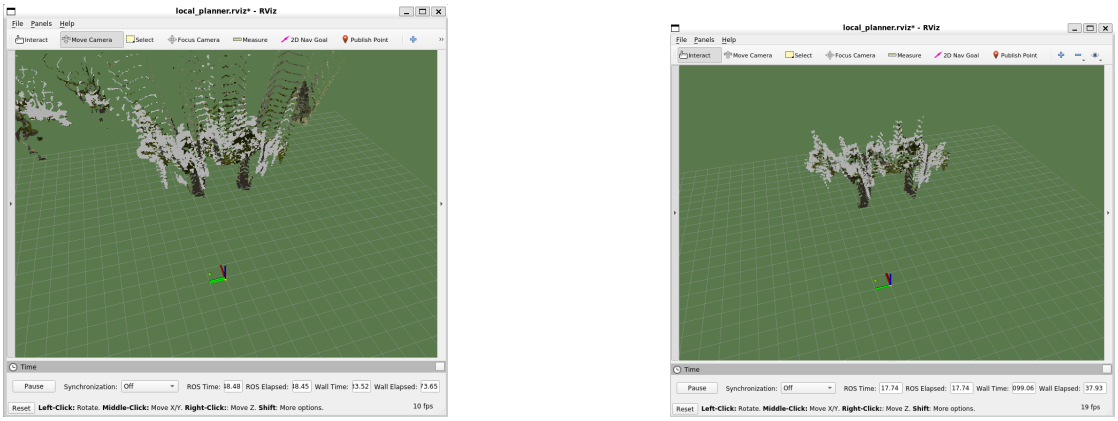
Eine erste Überprüfung des Algorithmus wird die Simulation wie in der Anleitung[1] beschrieben durchgeführt. Einige Paramameter wurden zur besseren Übersichtlichkeit angepasst, sodass:

- *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 4.2 links dargestellt. Der Algorithmus zur Erzeugung der Punktwolke ist anfällig für Rauschen verursacht durch minimale Bewegungen in den Kamerabildern, sodass im Bild Artefakte enthalten sind. Deshalb wurde die Punktwolke noch einmal mit einem VoxelGrid[2] gefiltert, abgebildet in 4.2 rechts. Für die Verwendung der Punktwolke

[1] https://github.com/PX4/PX4-Avoidance[4]
[2] http://wiki.ros.org/pcl_ros/Tutorials/VoxelGrid%20filtering[6]

mit Avoidance macht das Filtern keine Auswirkung, der *Local Planner* kommt auch mit den ursrpünglichen Daten der Punktwolke zurecht.



(a) Erzeugte Punktwolke

(b) Erzeugte Punktwolke, gefiltert

Abbildung 4.2.: Punktwolkendarstellung in *rviz* generiert aus simulierten Bildern

Zuletzt ist der Transformationsbaum in Bild 4.3 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 wiederrum 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[3]. Sowohl *mavros* als auch *Avoidance* können mit dem bestehenden Transformationsbaum arbeiten. Wird ein neues Objekt (bspw. Darstellung einer Punktwolke) 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.

4.2. 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

[3] https://www.ros.org/reps/rep-0105.html#base-link

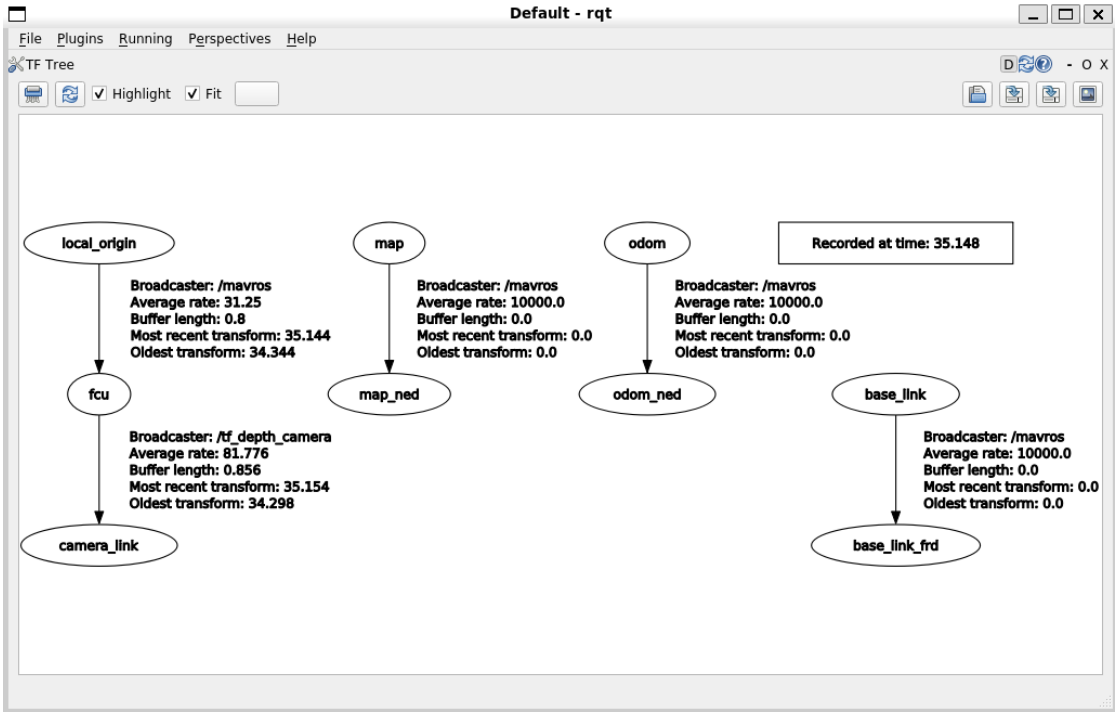


Abbildung 4.3.: Transformationsbaum tftree bei Simulation von *Local Planner*

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 4.4 rot markiert, die Funktionen der Bestandteile bleiben die gleichen wie in Kapitel 4.1. 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.
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 aus Tiefenbild von *AirSim*

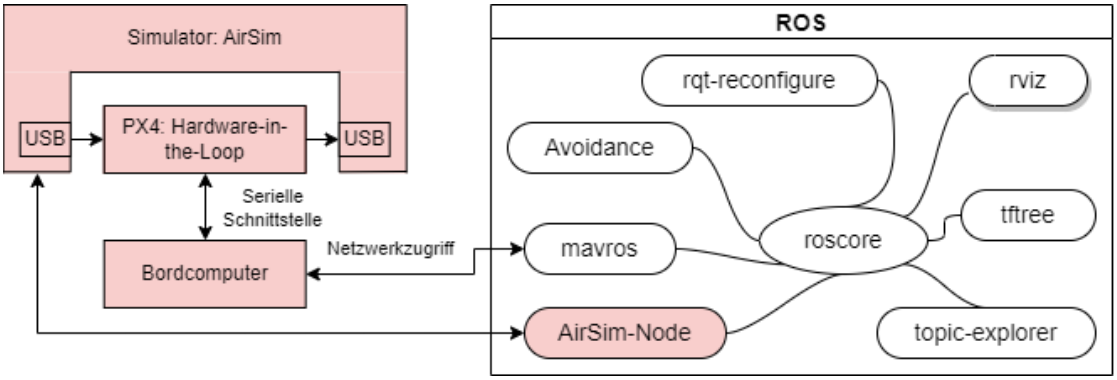


Abbildung 4.4.: Systemaufbau der Simulation von Avoidance als HIL-Simulation: Links Simulator (AirSim) und Flugcontroller (HIL), Rechts Bestandteile des ROS

- Entfernen der Transformation von „fcu“ auf „camera_link“, da 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. Die erfasste Punktwolke der Ausgangssituation ist zu sehen in Bild . Anschließende Anpassungen des Transformationsbaums ergeben eine Ansicht wie in . Der finale Transformation ist in Bild abgebildet.
Das ganze funktioniert so nicht... wer hätts gedacht

4.3. Machbarkeitsstudie Stereokamera mit Raspberry Pi

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

- 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.
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.

Tabelle 4.1.: Benchmark zum Programm *stereo_image_proc* auf dem RPI

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

4.4. Einrichtung der Ultraschallsensoren

5. Erweiterte Flugtests: Avoidance

5.1. Tests als Simulation

5.2. Anschluss der Sensoren

5.3. Verwendung Local Planner mit echter Hardware

6. Einrichtung der Stereokamera

A. Anhang

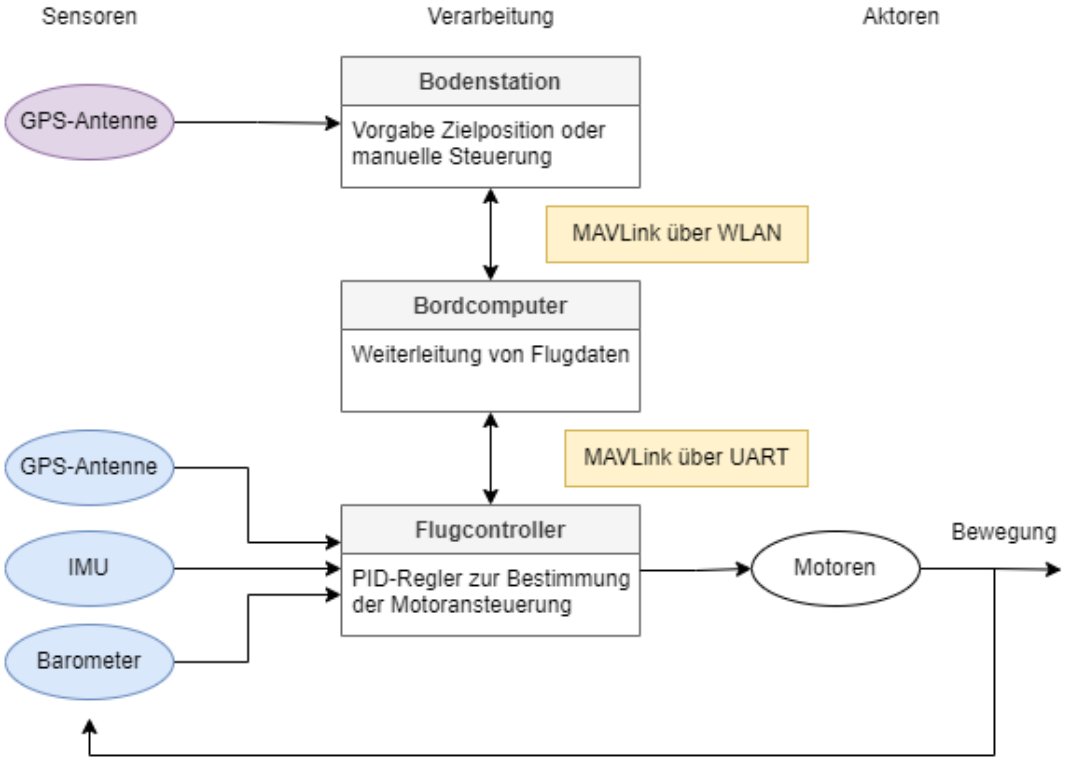


Abbildung A.1.: Beschreibung des Systems Drohne-Bodenstation: Grau hinterlegt die Bestandteile aus Tabelle 1.1; Gelb hinterlegt das jeweilige Kommunikationsprotokoll; Blau hinterlegt vorhandene Sensoren; Lila hinterlegt optionale Sensoren

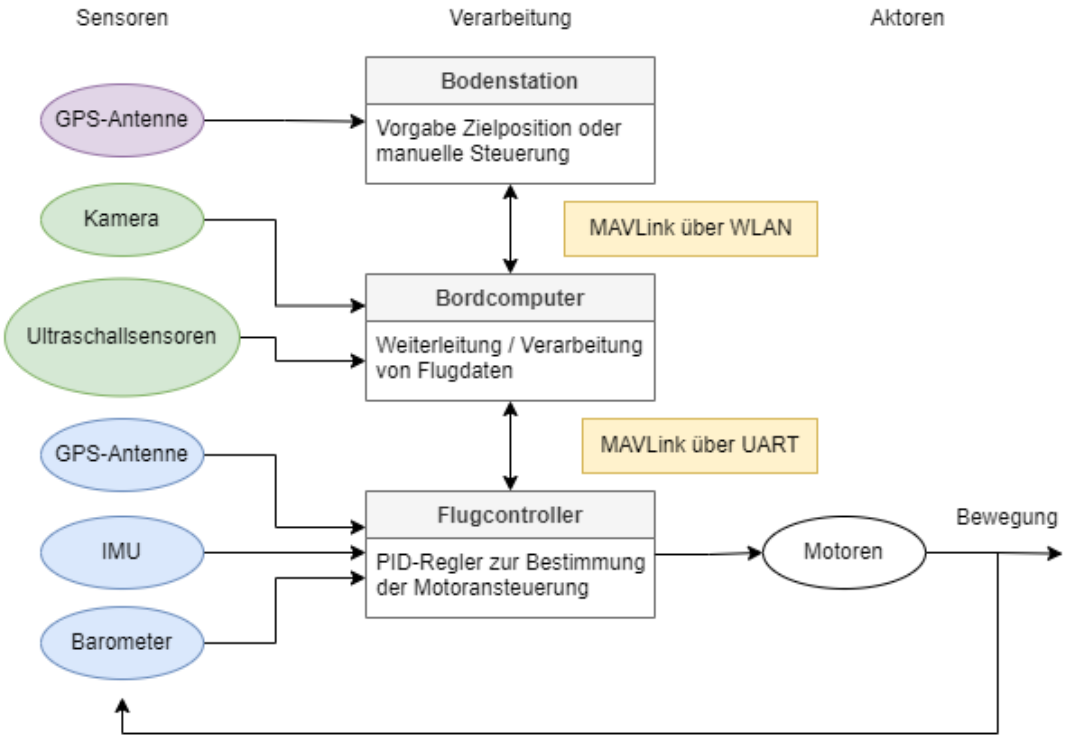


Abbildung A.2.: Beschreibung des erweiterten Systems Drohne-Bodenstation: Grün hinterlegt neu vorgesehene Sensoren

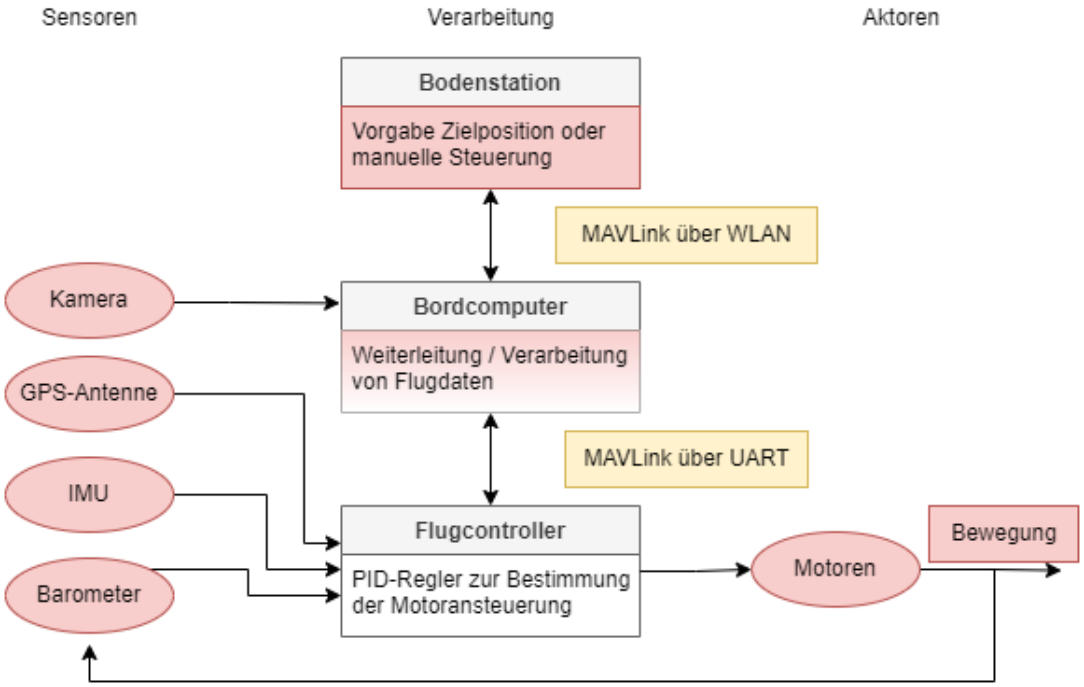


Abbildung A.3.: Beschreibung des Systems zur Simulation HIL: Hell-rot dargestellt die von der Simulation übernommenen Aufgaben

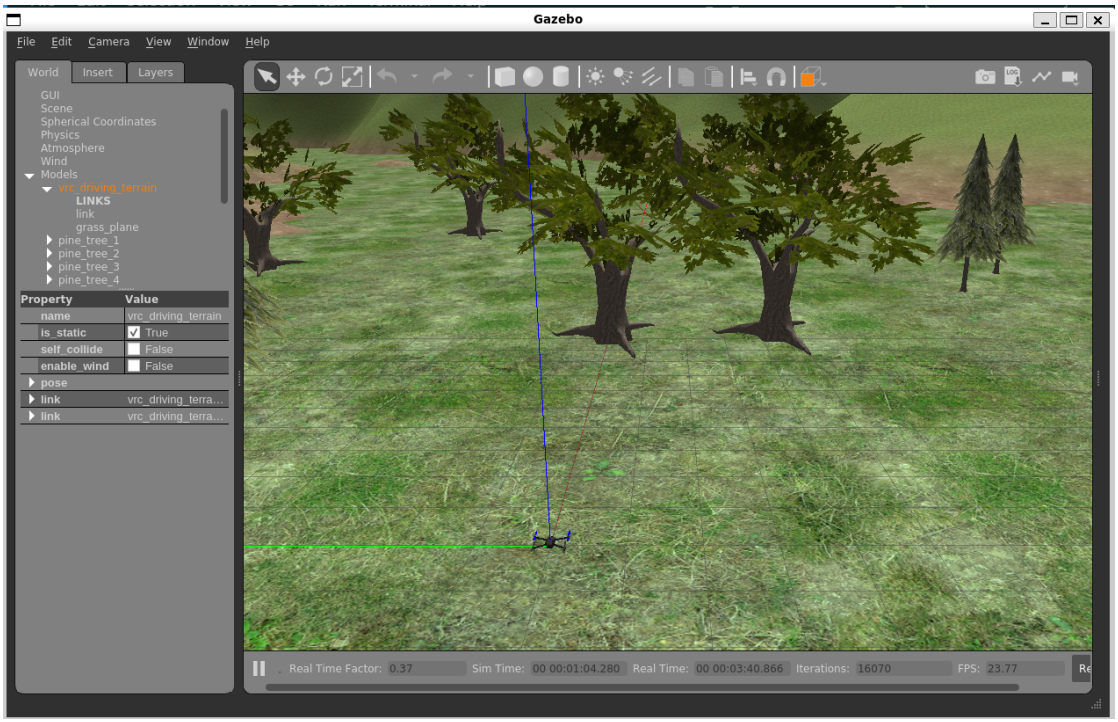


Abbildung A.4.: Erste Ansicht von Gazebo bei der Simulation von Avoidance: Relativ klein im Vordergrund die simulierte Drohne mit Koordinatenachsen (Rot, Grün, Blau), im Hintergrund die Welt mit Bäumen, auf linker Seite ein Konfigurationsmenü von Gazebo

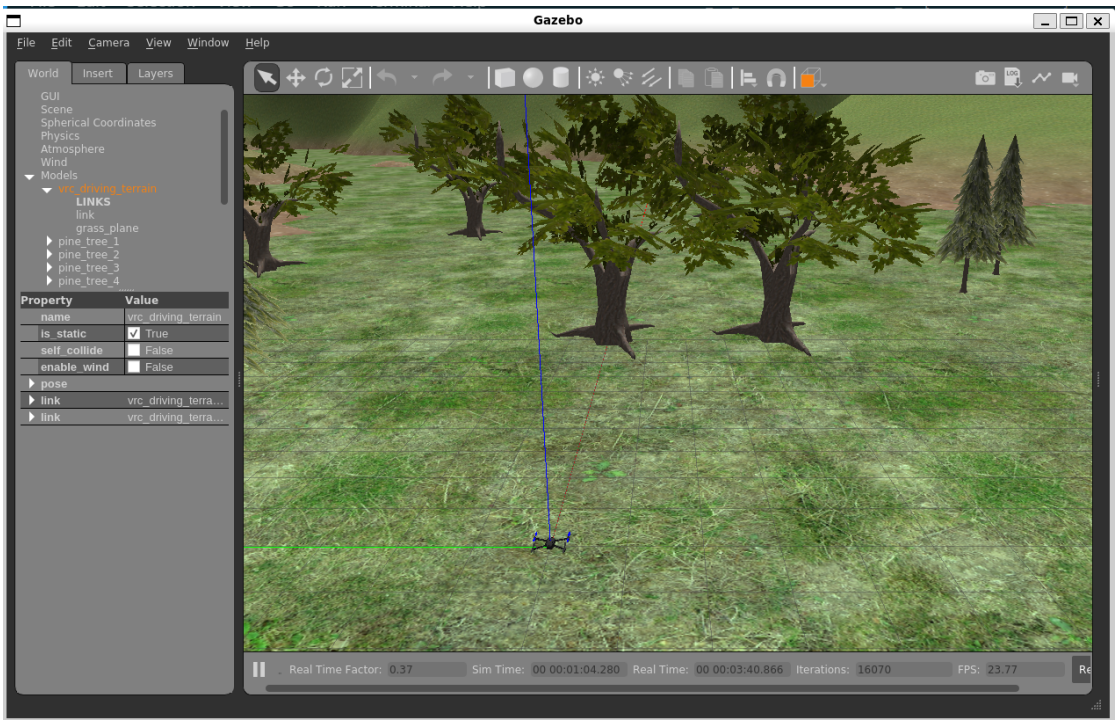


Abbildung A.5.: Erste Ansicht von AirSim bei der Blocks-Simulation: im Vordergrund die simulierte Drohne, im Hintergrund ein großer Quader und eine Kugel

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

Tabellenverzeichnis

Akronyme

FPS Frames per Second. 17
GPS Global Positioning System. 5, 9
HIL Hardware-In-the-Loop. 3
IMU Inertial Measurement Unit. 2
MAVLink Binäres Übertragungsprotokoll für ressourcenbeschränkte Umgebungen, bevorzugt auf Robotern und Drohnen eingesetzt. 5, 11
ROI Region Of Interest. 4
ROS Robot Operating System, Framework mit Schnittstellen für Sensor/Aktor-Verknüpfungen. 2, 3, 5, 6, 11, 12, 15, 17
RPI Raspberry Pi (Einplatinencomputer). 2
SLAM Simultane Lokalisierung und Kartierung der Umgebung. 5
UART Universal Asynchronous Receiver Transmitter. 1
USB Universal Serial Bus. 3
WLAN Wireless LAN. 1, 3