Studienarbeit T3200

Erweiterung bestehender Drohnen um eine Autonomflugfähigkeit

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

eingereicht von

Markus Rein

Abgabedatum: 20. März 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

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*1.2

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

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

Bodenstation:

Inertial Measurement Unit (IMU): Zusammenfassung der Sensorn 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 μC, 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.

...

1.1. Beschreibung des Systems

Zunächst in Tabelle **??** aufgelistet, der Aufbau des Systems Drohne und Bodenstation in der einfachen Ausführung ohne jegliche automone Fähigkeiten. Sie bilden zusammen eine Verwaltungseinheit, denn der Drohne müssen Fluganweisungen von der Bodenstation eingespeist werden.

Bestandteil der Drohne sind weitere Komponenten (Sensoren und Aktoren), wovon die meisten im Flugcontroller integriert sind bzw. in engen Zusammenhang mit diesem

Tabelle 1.1.: Systemübersicht Drohne und Bodenstation

	Drohne		
	Flugcontroller	Bordcomputer	Bodenstation
Funktion	Autopilot-Software liest Sensoren und steuert Aktoren 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

stehen, wie in Bild 1.1 dargestellt. Grau hinterlegt sind die Bestandteile aus Tabelle **??**. Im Rahmen dieser Projekte hinzugefügte Sensoren sind grün hinterlegt. Diese werden direkt am Bordcomputer angeschlossen, der mit den gewonnenen Daten Berechnungen zum Flug durchführen soll. Der Bordcomputer erhält kabelgebunden über UART-Verbindung Daten vom Flugcontroller und sendet diese kabellos per WLAN-Verbindung weiter zur Bodenstation (gelb hinterlegt).

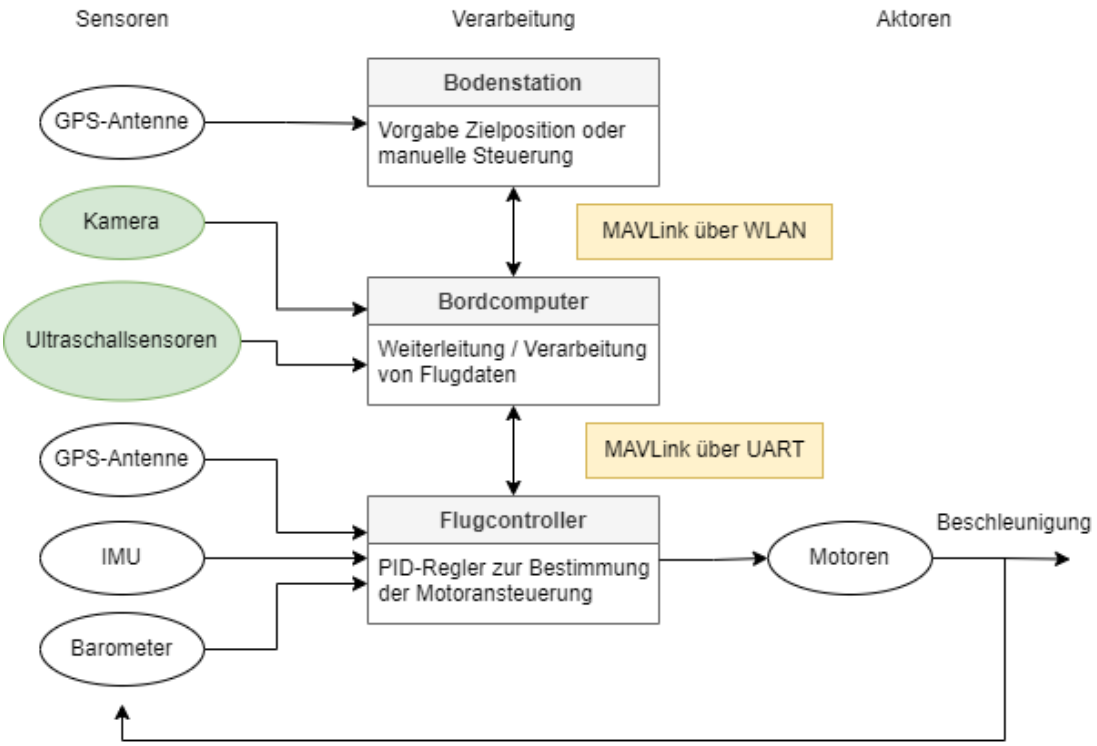


Abbildung 1.1.: Beschreibung des Systems Drohne-Bodenstation

1.2. Beschreibung des Zielsystems

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 Erkennung von Hindernissen
- Berechnung alternativer Flugbahn zur Umgehung von Hindernissen

Avoidance Zur Umsetzung der Flugplanung soll zuerst

1.3. Beschreibung der Simulation

Zur Entwicklung wurde ein Hardware-In-the-Loop (HIL)-Aufbau aufgesetzt. Bild 1.2 zeigt, in Anlehnung an Bild Bild 1.1, die durch die Simulation übernommenen Funktionen in hell-rot. Die Aufgaben der Komponenten während der Entwicklung sind:

Simulation: In der Simulation werden sowohl alle physikalischen Effekte als auch die virtuelle Umgebung der Drohne berechnet. Die Sensordaten werden der Drohne direkt von der Simulation per USB-Anschluss 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.

Erweiterte Funktionalität wird auf dem Entwicklungsrechner in Containern, in Verbindung mit der Simulation, erprobt. Derartige fertige Anwendungen können dann direkt auf dem Bordcomputer eingesetzt werden. Bestandteile der Software sind:

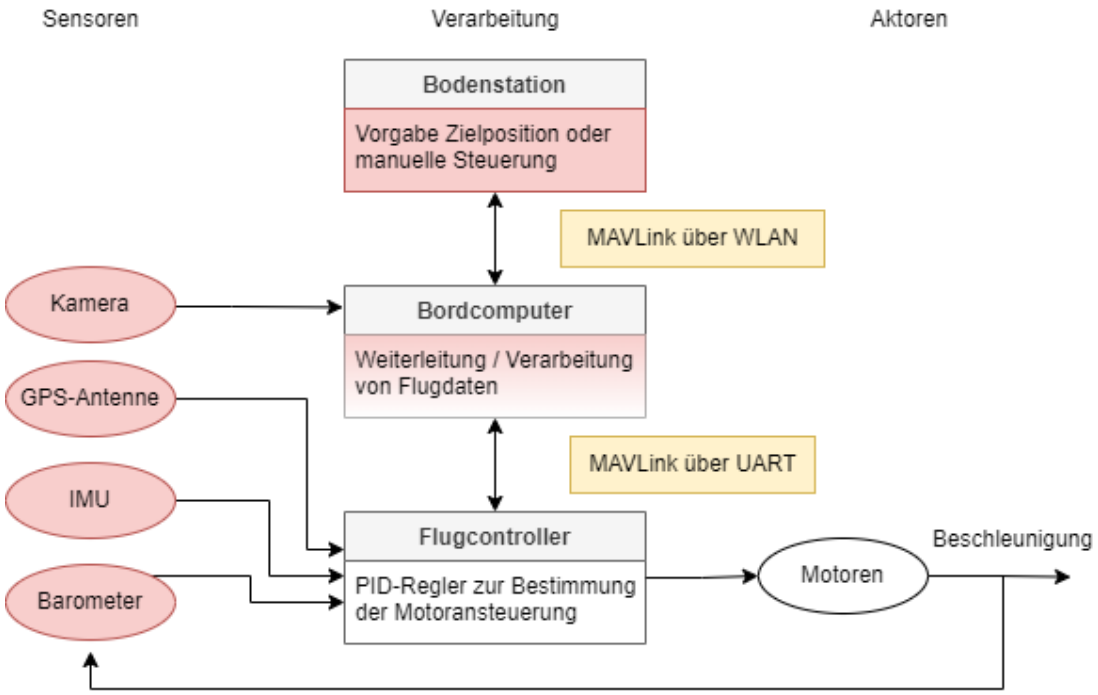


Abbildung 1.2.: Beschreibung des Systems zur Simulation der Drohne

ROS

***mavros*:** auf gleicher Ebene angesiedelt wie eine Bodenstation, erlaubt Protokollübersetzung zwischen MAVLink- und ROS-Nachrichten für ROS-internen Datenaustausch, empfängt Daten der Drohne und sendet neue Anweisungen zur Drohne

Tiefenverarbeitung: arbeitet direk mit Tiefenbildern aus dem Simulator um eine „Punktwolke der Umgebung“ zu generieren

***Avoidance*:** setzt neue Zielpunkte für Drohne anhand von Sensordaten der Drohne und Punktwolke von Kamera

***AirSim-Wrapper*:** nicht für Endanwendung benötigt, kommuniziert direkt mit dem Simulationsprogramm und stellt Tiefenbild bereit

2. Stand der Technik

2.1. Hinderniserkennung

Verschiedene Prinzipien stehen zur Bilderkennung zur Verfügung. Nachfolgend vorgestellt werden die grundlegenden Techniken der Bilderkennung.

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 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

Eine Sonderform des SLAM ist Visual SLAM (vSLAM). Bei diesem werden ausschließlich Kameras zur Erkennung der Umgebung eingesetzt. Algorithmen verwenden zumeist zusätzlich die Daten der IMU, um die Bewegung der Kamera in die Berechnung der Position einzubeziehen.

Stereokamera

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

Optical Flow

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.

2.2. Avoidance

Das Projekt „Obstacle Detection and Avoidance“[1], auf GitHub verfügbar als PX4-Avoidance[1], hier nur *Avoidance* genannt, entstand in enger Zusammenarbeit mit der Dronecode Stiftung an der ETH Zürich.
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:

Es arbeitet mit der Software des Flugcontrollers (PX4) und dem ROS-Metabetriebssystem. Die Software Avoidance erhält per ROS-Nachrichten die Soll-Trajektorie und Sensordaten vom Flugcontroller. Außerdem wird zur Navigation eine Punktwolke der Umgebung benötigt. Falls notwendig wird eine angepasste Trajektorie an den Flugcontroller gesendet. Die Software kann nicht direkt auf dem Flugcontroller ausgeführt werden, da die Berechnungen sehr enorm viele Ressourcen 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, Intel RealSense.
Sprich Stereokamera ist erprobt
Stereokamera liefert genaue Karte der Umgebung, ähnlich einem Lidar.
Andere Methoden arbeiten eventuell nicht mit Avoidance zusammen.

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

3. 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.

3.1. Avoidance als Simulation in Software

Das Avoidance-Modul arbeitet eng mit der Simulationssoftware „Gazebo“ zusammen. In der Dokumentation [1] 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 [2, 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 3.1 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 1.3 und [2, Kapitel 3.4.1]. Zusätzlich stellt der Simulator Kamerabilder bereit, je nach gewähltem Modus direkt Tiefenbilder oder Stereobilder welche weiter verarbeitet werden müssen.

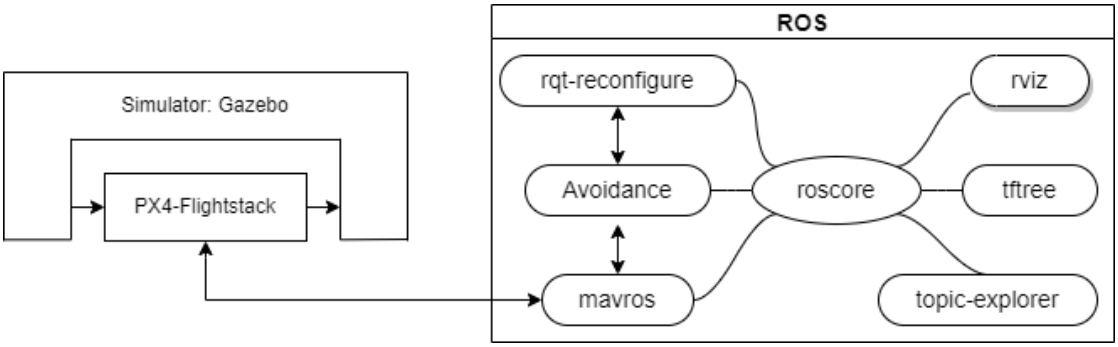


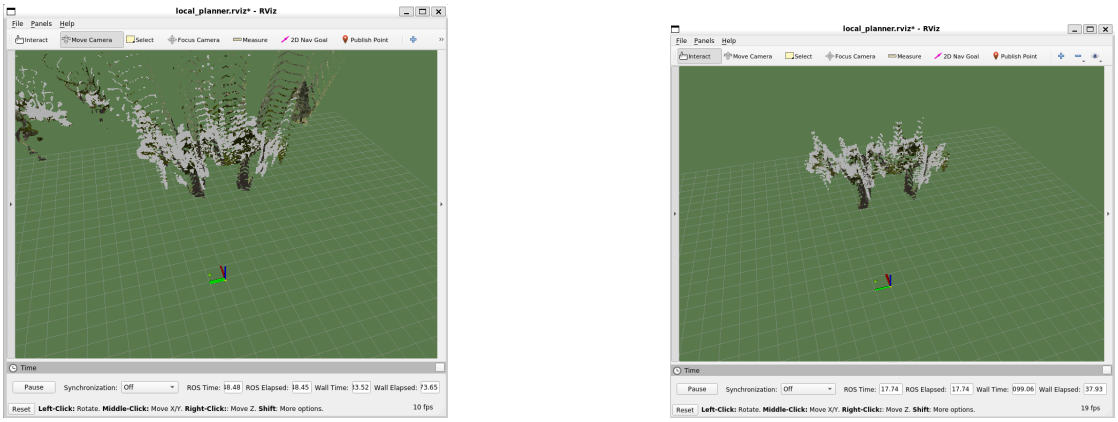
Abbildung 3.1.: Systemaufbau der Simulation von Avoidance: Links Simulator (Gazebo) und Flugcontroller (reine Software), Rechts Bestandteile des ROS

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.1. Von der Drohne werden 2 virtuelle Kamerabilder erzeugt, aus welchen eine Punktwolke gebildet wird. Die Punktwolke ist in Bild 3.2 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 3.2 rechts. Für die Verwendung der Punktwolke mit Avoidance macht das Filtern keine Auswirkung, der *Local Planner* kommt auch mit den ursrpünglichen Daten der Punktwolke zurecht.

[1] https://github.com/PX4/PX4-Avoidance[1]
[2] http://wiki.ros.org/pcl_ros/Tutorials/VoxelGrid%20filtering[3]



(a) Erzeugte Punktwolke

(b) Erzeugte Punktwolke, gefiltert

Abbildung 3.2.: Punktwolkendarstellung in *rviz* generiert aus simulierten Bildern

Zuletzt ist der Transformationsbaum in Bild 3.3 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.

3.2. 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 [2, Kapitel 5] als HIL-Simulation betrieben werden. Die Veränderungen im Aufbau für die Simulation sind in Bild 3.4 rot markiert, die Funktionen der

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

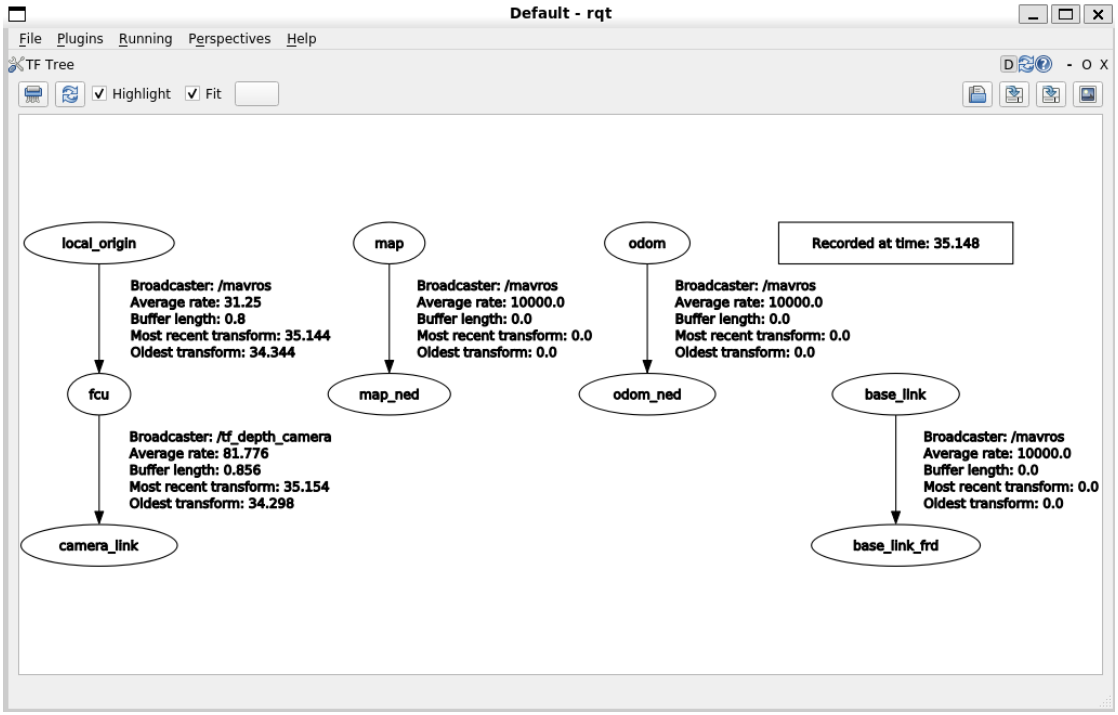


Abbildung 3.3.: Transformationsbaum tftree bei Simulation von *Local Planner*

Bestandteile bleiben die gleichen wie in Kapitel 3.1. Hinzu kommt der Bordcomputer, der eine netzwerkgebundene Kommunikation mit dem Flugcontroller erlaubt und der Knoten /enquoteAirSim-Node, durch den Bilder aus der Simulation ausgelesen werden können. Dabei entfällt das Zusammenfügen zweier Stereobilder, denn die Simulation liefert direkte Tiefenbilder.

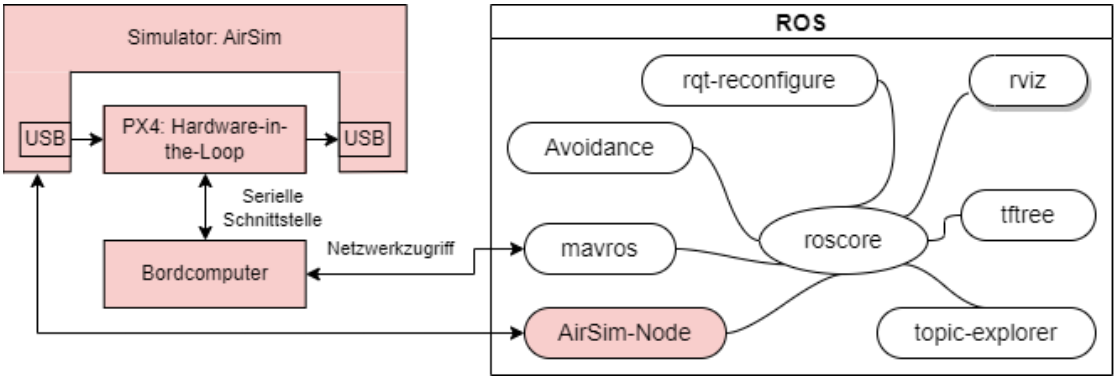


Abbildung 3.4.: Systemaufbau der Simulation von Avoidance: Links Simulator (AirSim) und Flugcontroller (HIL), Rechts Bestandteile des ROS

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_stereo_airsim.launch*
   - Entgernen der Funktion von Disparitätsbildern, da Anwendung keinen ersichtlichen Zweck erfüllt
   - Unteraufruf von *avoidance_sitl_stereo_airsim.launch* anstatt *avoidance_sitl_stereo.launch*
2. *avoidance_sitl_stereo.launch*:
   - Umbenannt zu *avoidance_sitl_stereo_airsim.launch*
   - Entfernen der Stereo-Bildverarbeitung, stattdessen Berechnung der Punktwolkendarstellung aus Tiefenbild von *AirSim*
   - Entfernen der Transformation von „fcu“ auf „camera_link“, da Punktwolkendarstellung noch nicht bekannt
   - Unteraufruf von *avoidance_sitl_mavros_airsim.launch* anstatt *avoidance_sitl_mavros.launch*
3. *avoidance_sitl_mavros.launch*:
   - Umbenannt zu *avoidance_sitl_mavros_airsim.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 *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“

Das ganze funktioniert so nicht... wer hätts gedacht

3.3. 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 3.1.: 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

A. Anhang

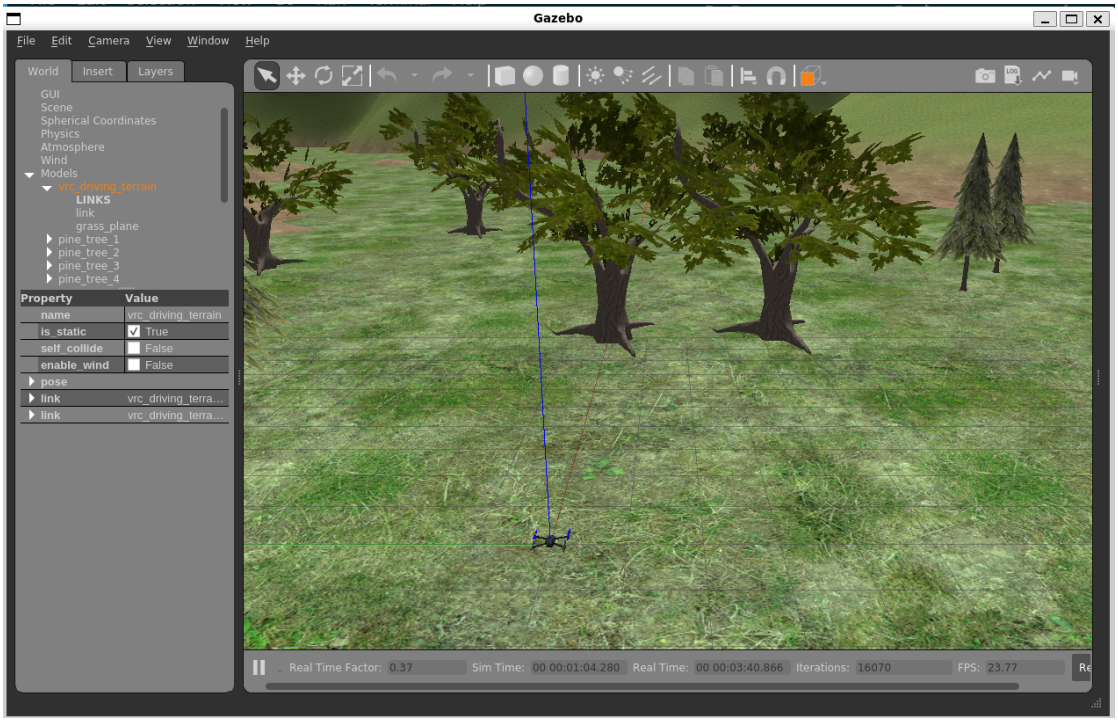


Abbildung A.1.: Erste Ansicht von Gazebo bei der Simulation von Avoidance: Relativ klein im Vordergrund die simulierte Drohne mit Koordinatenachsen (Rot, Grün, Blau), im Hintergrund die Welt mit Bäumen, auf linker Seite ein Konfigurationsmenü von Gazebo

Literatur

[1] Dronecode Stiftung, *Obstacle Detection and Avoidance*, PX4 Autopilot for Drones, 15. Jan. 2023.

[2] Markus Rein, „Erweiterung Bestehender Drohnen Um Eine Autonomflugfähigkeit," DHBW Stuttgart, 22. Jan. 2023.

[3] OpenRobotics. (). Documentation - ROS Wiki, Adresse: https://wiki.ros.org/ (besucht am 10. 03. 2023).

Abbildungsverzeichnis

Tabellenverzeichnis

Akronyme

FPS Frames per Second. 13
GPS Global Positioning System. 6
HIL Hardware-In-the-Loop. 4
IMU Inertial Measurement Unit. 2
MAVLink Binäres Übertragungsprotokoll für ressourcenbeschränkte Umgebungen, bevorzugt auf Robotern und Drohnen eingesetzt. 5
ROS Robot Operating System, Framework mit Schnittstellen für Sensor/Aktor-Verknüpfungen. 2
RPI Raspberry Pi (Einplatinencomputer). 2
SLAM Simultane Lokalisierung und Kartierung der Umgebung. 6