Aufbau eines Simulationsframeworks zur Validierung von Quadrotortrajektorien

Christian Maier, Longfei Li, Yingyi Zhang, Fengyun Shao

***Zusammenfassung*— Der Inhalt dieses Artikels beschreibt den Aufbau eines Simulationsframeworks zur Validierung von Quadrotortrajektorien. Es wird beschrieben, wie 3D Daten von Gebäuden über OpenStreetMap in Gazebo importiert werden. Außerdem wird die Funktionsweise eines selbsterstellten Python-Skripts erklärt, das die Simulation und das Programm für die Drohnen vor dem Start der Trajektorienvalidierung vorbereitet. Danach wird gezeigt, wie eine Validierung einer Quadrotortrajektorie gestartet wird. Hierbei wird eine triviale Bahnberechnung als Beispiel genommen. Zum Schluss wird die Logik für die Erkennung einer Kollision erklärt und wie die Daten gespeichert werden.**

I. Einführung

Quadrotoren werden vermehrt in verschieden Gebieten eingesetzt. Besonders in Bereichen, wo viele Hindernisse in der Umgebung sind oder auch andere UAVs (Unmanned Aerial Vehicles) autonom in der Luft sind, ist ein kollisionsfreier Flugpfad besonders wichtig. Einen Flugpfad in der realen Welt zu validieren ist sehr kostspielig und zeitaufwendig. Deshalb werden solche Aufgaben in einer Simulation erledigt, die im Vergleich schneller und kostengünstiger ist. Um dies zu ermöglichen wird ein Programmgerüst gebraucht, das mit kleinen und einfachen Änderungen einer beliebige Trajektorien mit beliebigen Anzahl an Quadrotoren zulässt. Außerdem soll es möglich sein, die Simulationswelt mit gewünschten Hindernissen aus der realen Welt auszustatten. Ziel dieses Projekts ist ein solches Simulationsframework aufzubauen.

Als Fundament und Einstiegspunkt wurde das Tutorial von Intelligent Quads genommen. [1] Genau wie im Tutorial wurde das Simulationsprogramm Gazebo auf der Linux-Distribution Ubuntu 20.04 installiert. Die Robotersimulationssoftware Gazebo 11 wird für das Erstellen der Welt, das ArduPilot Plugin für Gazebo wird für die SITL (Software-In-The-Loop) Simulation benötigt. Mit dem Bodenkontrollstationsprogramm MAVProxy verbinden sich alle SITL Simulatoren von den Drohnen und dient als Kommunikationsschnittstelle für andere Programme. In dem Tutorial sind außerdem Beispielprogramme enthalten, die die Kommunikationsmöglichkeiten mit den Drohnen zeigen. Außerdem ist eine Beispielwelt mit drei Drohnen in diesem Tutorial aufgebaut. Das Beispielprogramm und die Beispielwelt wird über das Roboter Software Framework ROS (Robot Operation System) Noetic gestartet.

Die Arbeit ist in folgende Sektionen unterteilt. In der Sektion II werden die Grundlagen vermittelt und die grobe Struktur dieses Projekt vorgestellt. Hierbei wird detailierte auf die drei Software Gazebo, ROS und ArduPilot eingegangen und welche Beziehung sie miteinander haben. Wie statische 3D Daten in die Simulationsumgebung importiert werden, wird in der Sektion III näher erklärt. Anschließend wird in Sektion **??** vorgestellt, welche Methode die Drohnen für die Berechnung des Pfades benutzen. In Sektion V wird beschrieben, wie eine Kollision in der Simulationswelt erkannt, gefiltert und gespeichert wird. Die Vorbereitung der Validierung übernimmt ein selbstgeschriebenes Python-Skript und wird in der Sektion VI eingegangen. Zum Schluss kommt eine Zusammenfassung des Projekts mit möglichen Ausbaumöglichkeiten in Sektion VII.

Diese Arbeit wurde von Nikolas Hohmann, M.Sc. unterstützt.

II. Grundlage

A. Simulationssoftware Gazebo

Die gesamte Simulationswelt ist in Gazebo aufgebaut. Gazebo ist eine Multi-Roboter Simulationssoftware, die Quadrotoren und Gebäude in einer dreidimensionalen Umgebung simulieren kann. Es hat eine leistungsstarke Physik-Engine, gute Grafikqualität und eine sehr einfache und schnelle Benutzeroberfläche. [2] Das Simulationsprogramm hat seinen Schwerpunkt in Robotik und bietet eine große Anzahl an Bibliotheken und Plugins wie in diesem Fall für ArduPilot. Sie hat eine Client/Server-Architektur mit themenbasiertem Publish/Subscribe-Modell für die Kommunikation zwischen verschiedenen Prozessen. Jedes gesteuerte Objekt kann mit einem oder mehreren Controllern verbunden werden. Der Client sendet die Steuerdaten, die Koordinaten des simulierten Objekts an den Server, der das simulierte Objekt in Echtzeit an den Server überträgt, um eine verteilte Steuerung zu erreichen.

Die Hardware in der Gazebo-Simulation soll das entsprechende Verhalten in der Realität widerspiegeln. Durch Einsatz solcher Simulationen soll Zeit und Geld für Tests direkt an der Hardware gespart werde, ohne das tatsächliche Verhalten der Hardware in der physischen Welt zu wissen. Auf der Abbildung 1 ist ein Quadrotor in der Simulation dargestellt.

Der Vorteil bei der Verwendung der Gazebo-Software ist, die Möglichkeit verschiedene Arten von Positionssensoren wie Lidar, Sonar und Global Positioning System (GPS) zu simulieren. [3] Dieser Vorteil soll als Fundament komplexere Software für die Trajektorienplanung dienen. Darüber hinaus können realistische Simulationen der Starrkörperphysik durchgeführt werden. Damit steht ein effektives Simulationsprogramm zum Testen zur Verfügung. In diesem Projekt wurde Gazebo 11 gewählt.

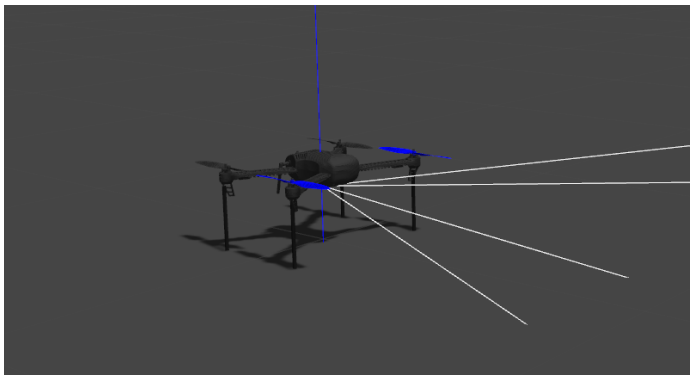

Abb. 1: Quadrotor in Gazebo

B. ArduPilots

Das ArduPilot Projekt bietet eine fortgeschrittene und vollwertige Open Source Autopilot-Softwaresystem. [4] Das Projekt wird von einer globalen Gemeinschaft weiterentwickelt und kann deswegen eine große Anzahl an verschiedene Fahrzeugsysteme steuern. Hierunter fällt auch Quadrotoren und andere UAVs. ArduPilot wird von großen Institutionen und Unternehmen wie NASA oder Intel verwendet.

In diesem Projekt ist unter anderen die SITL Simulator, die Bodenkontrollstation MAVProxy und das Kommunikationsprotokoll MAVLink. Die Bodenkontrollstation MAVProxy basiert auf das Kommunikationsprotokoll MAVLink und ist somit kompatibel mit der SITL Simulator von ArduPilot. Sie wird häufig von Entwickler für das Testen von neuen Software für UAVs verwendet, weil einer der Schlüsselfunktion der Software erlaubt, Nachrichten von den UAVs an andere Bodenkontrollstationen zu senden.

Das Kommunikationsprotokoll MAVLink kann über jede serielle Schnittstelle, z.B WiFi, übertragen werden. Da das Protokoll nicht garantiert, dass die Nachricht an den Empfänger ankommt, muss eine ständige Verbindung zwischen den Bodenkontrollstation und den UAVs bestehen, damit die Bodenkontrollstationen anhand der Zustände von den UAVs den Empfang der Nachricht bestätigen kann.

SITL Simulatoren sind eine weit verbreitete Methode zum Testen komplexe Systeme mit wiederholbaren und kontrollierbaren Komponenten. In diesem Fall simultiert die ArduPilot SITL Simulator eine virtuelle Drohne. Da das Programm in der Programmiersprache C++ geschrieben wurde, verhält sich die Software ähnlich als würde die Software auf einem Hardware laufen.

Das Gesamtsystem mit der Bodenkontrollstation MAVProxy und SITL Simulator sieht folgendermaßen aus. Der SITL Simulator wird mit der virtuelle Drohne in einer Simulationsumgebung verknüpft und bekommt über diese dann die Sensordaten. Dies Daten werden verarbeitet und dann an die Bodenkontrollstation gesendet. Die Bodenkontrollstation empfängt die Daten und leitet sie gegebenenfalls weiter. Falls Befehle für die Drohne gesendet werden soll, geschieht dies über die Bodenkontrollstation. Der SITL Simulator verarbeitet die Befehle und gibt die Steuerdaten an die Simulationsumgebung, die anschließend das Verhalten der virtuelle Drohne verändert.

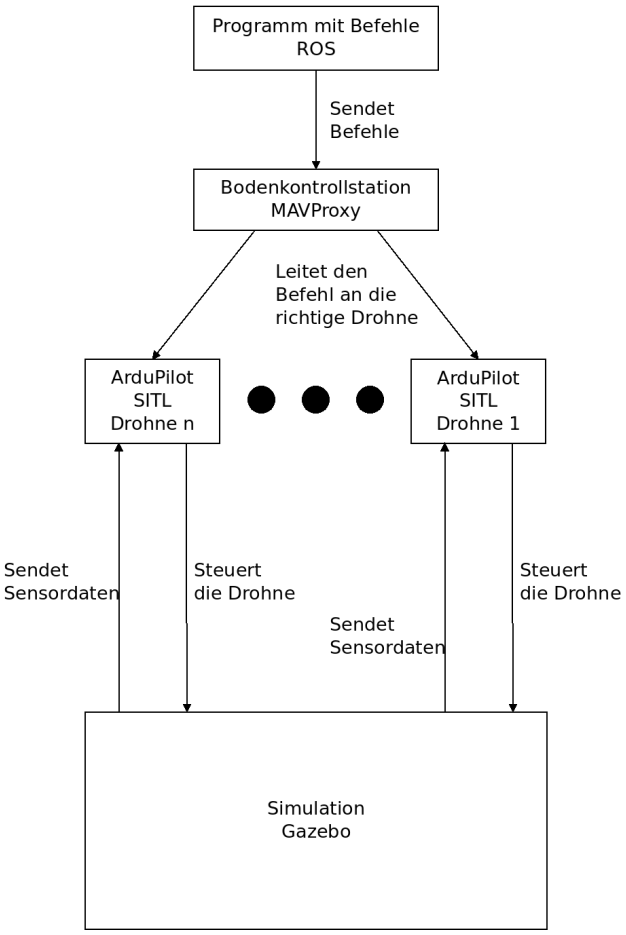

Abb. 2: Kommunikation zwischen den verschieden Programme

Die Abbildung 2 zeigt die Kommunikation zwischen den verschiedenen Systeme in diesem Simulationsframework. Die Befehle kommen von einem C++ Programm, dass über ROS gestartet wird.

C. ROS Umgebung

ROS (Robot Operation System) ist eine Sammlung von Bibliotheken und Werkzeuge , wie z.B. Hardwareabstraktion, Messaging zwischen Prozessen, Paketmanagement und Simulationsvisualisierung, und ist mit vielen Programmiersprachen kompatibel, z.B C++ und Python. [5] Der Hauptgrund für ROS in diesem Projekt ist, dass mit Hilfe der Werkzeuge und Bibliotheken vom Framework Programme gleichzeitig gestartet werden können und die Kommunikation zwischen den Programmen erleichtern wird.

Das Programmgerüst kommuniziert zwischen Dateien und Simulationssoftware (z.B. Gazebo) durch Knoten (nodes), die ein oder mehrere Themen(Topics) veröffentlicht und abonniert werden können. Jeder Knoten stellt einen Prozess dar, der eigenständig ein Softwareprogramm ausführt. Die Kommunikation zwischen den Knoten kann synchron mit einem Service oder asynchron mit einem Topic geschehen. Anhang eines Beispiels soll die Kommunikation über einen Topic verdeutlicht werden. Konten A sei ein Publisher und Knoten

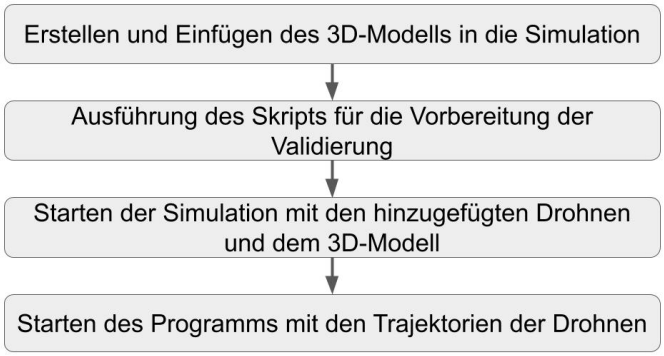

Abb. 3: Schritte für die Ausführung einer Trajektorienvalidierung mit der simulationsframework

B sei ein Subscriber. Um Daten vom Knoten A zum Knoten B zu senden, muss Knoten B den Topic von Knoten A abonnieren. Falls A eine Nachricht in den abonnierten Topic veröffentlicht, bekommt B die Nachricht sofort nach dem Senden der Nachricht.

Des weiteren unterstützt die verwendete ROS-Distribution die MAVROS-Erweiterung (Micro Air Vehicle ROS). [6] [7] Das Paket beschreibt den Treiber für die Kommunikation mit ArduPilot über das MAVLink Kommunikationsprotokoll. Außerdem ist ROS mit diesem Protokoll in der Lage, Befehle an den SITL Simulator über die Bodenkontrollstation MAVProxy zu senden. Die Funktion von ROS in unserer Projekt besteht darin, Gazebo mit den Modellen zu erzeugen, die Bereitstellung eines Schnittstelle für die Modelle in Gazebo und die Ausführung der Befehle für die Drohnen in der Simulation.

D. Struktur des Projekts

Für die Struktur des Projekts wurde folgender Ansatz erstellt. Es soll ein Skript erstellt werden, welches die nötigen Vorbereitungen für die Simulation bereitstellen soll. Dazu zählt das Erstellen einer beliebigen Anzahl an Drohnen in die Simulation und einer Datei, die das Programm für die Bahnberechnung der Drohne mit ihren jeweiligen Parametern ausführt. Um die Anzahl der Drohnen zu ermitteln, wird diese über das Einlesen einer YAML-Datei realisiert. Diese Datei enthält die Namen der Drohnen und deren zugehörigen Parametern. Ein weiterer Schritt ist die Änderung des Programms für die Übergabe der Befehle an die Bodenkontrollstation. Die Startesquenz und die Landung der Drohne ist für alle Drohnen gleich, jedoch unterscheiden sich die Wegpunkte der Drohnen, die sie abfliegen sollen. Also fehlt in der jetzigen Form die Skalierbarkeit des Programms. Für das Simulationsframework fehlt ein Programm, das Kollisionen in der Simulation erkennt und diese niederschreibt. Das 3D-Modell für die Simulation wird manuell in die simulierte Welt hinzugefügt.

Abbildung 3 zeigt die Schritte, die für das Ausführen einer Validierung eines Quadrotor in der gewünschten Simulationswelt gemacht werden müssen.

Abb. 4: Der Bereich von 3D Modell [8]

III. 3D-Modell in der Simulationswelt

Die Simulationswelt besteht aus statischen 3D Daten von Gebäuden der Technischen Universität Darmstadt und einer beliebigen Anzahl an dynamischen Drohnen. Wichtig ist der gleichzeitige Start der beiden Modelle und die Stabilität des Modells.

A. 3D-Modell erstellen

3D-Daten werden von OpenStreetMap (OSM) heruntergeladen. [8] Der Vorteil von OpenStreetMap gegenüber anderen wie z.B Google Maps ist, dass die Daten offen sind und kostenlos zur Verfügung stehen. Zusätzlich werden die Daten kontinuierlich aktualisiert, womit die Zuverlässigkeit von Simulationen mit aktuellen Bedingungen erhöht wird. Für das Projekt wurden die Gebäude des zentralen Campus der Technischen Universität Darmstadt als Simulationsmodell gewählt. Der genaue Bereich ist in der Abbildung 4 dargestellt. Wie in der Abbildung 4 gezeigt wird, sind die Hauptgebäude wie das Universitätszentrum, die Mensa Stadtmitte und das Elektrotechnik und Informationstechnik Institute enthalten.

Die exportierten 3D-Daten (osm-Datei) werden in Blender optimiert und für die letztendliche Verwendung in eine dae-Datei umgewandelt werden. Blender ist eine 3D-Animationssoftware, die einen Modelloptimierungsansatz aus Modellierung, Animation, Materialien, Rendering und mehr anbietet. [9] In Blender wird zunächst das Gebäude in der Mitte platziert. Auch die Umrandung wird zur besseren Erkennbarkeit gerendert. Rendering beschreibt Auflösung, Schattierungsmethode, Abtastrate, Performance. Um die Größe (Dimension) in der Gazebo Simulation anzupassen, wurde in Blender besonders Wert auf die Größe des Modells gelegt. Die Abbildung 5 zeigt das 3D-Modell in Gazebo nachdem die Modelle eingefügt werden. Nachdem die 3D-Daten in Blender angepasst wurden, werden die Daten weiter mit MeshLab verarbeitet. MeshLab ist ein Netzverarbeitungssystem, das bei der Bearbeitung von Modellen ohne spezifische Struktur hilft, die in 3D-Scans erfasst wurden. [10]

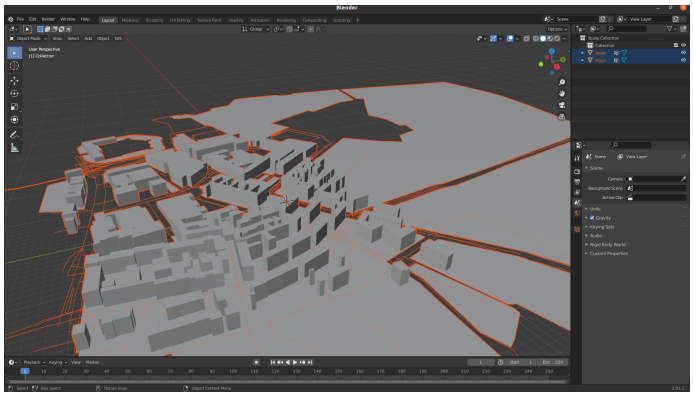

Abb. 5: 3D Modell von Gebäude der TU Darmstadt in Blender

B. 3D-Modell in Gazebo importieren

Einige Dateien müssen geschrieben werden, um das Modell vollständig zu beschreiben. Es ist wichtig, die Struktur des Simulationspakets in ROS zu verstehen, d.h. welche Datei wo platziert werden soll. Die Modell-Datei umfasst drei Teile, der Ordner Meshes, die config-Datei und die sdf-Datei. Meshes stellen die COLLADA-Dateien(dae) aus MeshLab dar. COLLADA ist Speicherformat und dae ist die Dateiendung. [11] Die dae-Datei enthält das Aussehen von des 3D-Modell der Gebäude. Die config-Datei enthält Informationen wie z.B. den Modellnamen, eine kurze Beschreibung zum Modell und die Modellversion. Die sdf-Datei bietet Modell-Informationen wie Modellposition, verknüpfte Kollisionselemente, Geometrie, Größe, Materialinformationen, Masse, Rotationsträgheit, usw. Die drei Datei, d.h. dae-Datei, config-Datei und sdf-Datei sind im XML-Dateiformat.

Die Stabilität des Modells ist von großer Bedeutung. Beim Importieren von Modellen in die Simulationswelt sind die häufigsten Probleme, dass die Modelle nach oben fliegen, von einer Seite zur anderen wackeln,usw. Mögliche Ursachen sind eine zu hohe Einstellung der Masse, eine falsche Einstellung der Rotationsträgheit von Link und eine falsche Einstellung von statischen Parameter der Modellen.

Die Gazebo-Datei enthält Informationen darüber, welches Modell der Welt erzeugt werden soll. Es hat zwei Hauptordner:

Der Ordner launch: Dieser Ordner enthält die Datei runway.launch, die zum Starten der Welt, zur Erzeugung der Gebäudemodelle in der Umgebung und das Einbinden der erforderlichen Plugins verwendet wird.

Der Ordner world: Dieser Ordner enthält die Weltdateien der Simulationen, die von der oben genannten Startdatei aufgerufen werden. Diese Datei enthält Informationen über die Simulationsumgebung. Damit die Gebäude und die Quadrotoren in der Simulation zu sehen sind, müssen die Modelle von den Gebäuden und der Drohnen in die world-Datei eingefügt werden. In diesem Fall wurden die Modelle in die runway.world hinzugefügt.

Wie in Abbildung 6 gezeigt wird, ist die Struktur der Modell-Datei und Gazebo-Datei klar dargestellt. Die Abbildung 7 zeigt das 3D-Modell in Gazebo nachdem die Modelle eingefügt wurden. So können wir statische und dynamische Modelle gleichzeitig simulieren.

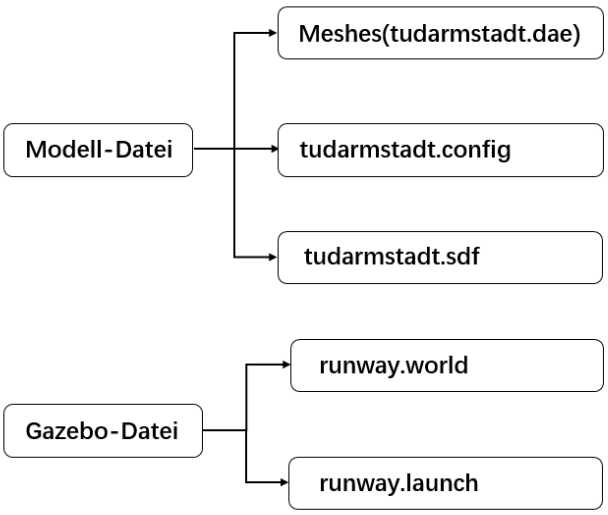

Abb. 6: Struktur von Modell-Datei und Gazebo-Datei

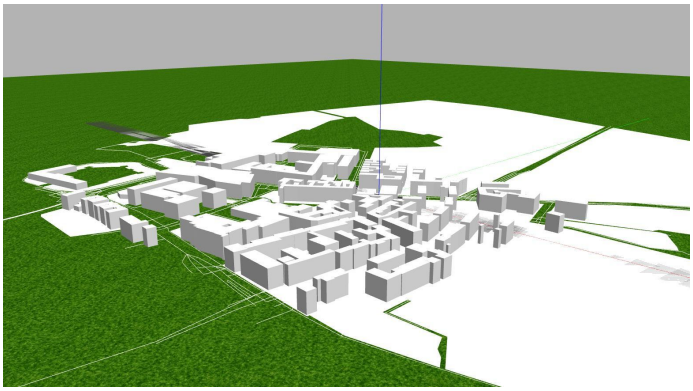

Abb. 7: 3D Modell von Gebäuden der TU Darmstadt und Drohnen in Gazebo

IV. Berechnung und Implenmierung des Pfades

A. Ablauf der Pfadsgenerierung

Der Pfad der Quadrotoren wird durch die Interpolation realisiert. In diesem Projekt werden drei Koordinaten vorgegeben:

$$\vec{X}_{\text{Data}} = (x_s, x_z, x_e)^T$$
$$\vec{Y}_{\text{Data}} = (y_s, y_z, y_e)^T$$
$$\vec{Z}_{\text{Data}} = (z_s, z_z, z_e)^T$$

Sie stellen jeweils die Komponenten der X, Y und Z-Achse des Startpunkts, des Zwischenpunkts und des Zielpunkts dar. Mit diesen drei Punkten als Parameter wird die gewünschte Kurve durch Interpolation berechnet. Die daduch berechneten, auf der Kurve liegenden Punkte werden in der “Waypointlist” gespeichert. Die so generierten Punkte dienen als Sollposition an das Steuersystem des Quadrotors übergeben. Für ein intuitives Verständnis hilft das Abbildung 8 und 9.

Folgende Zeichen sind zu erläutern:

$$x_i = Waypointlist[i].x \tag{1}$$

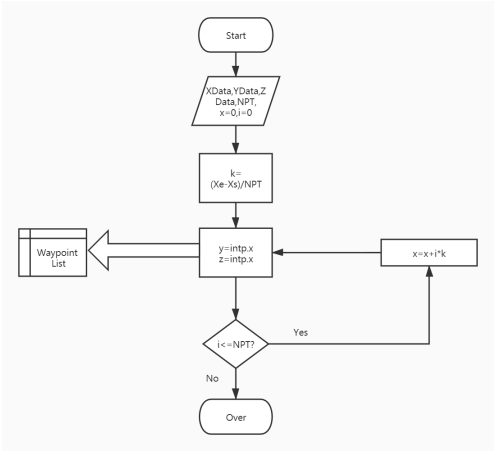

Abb. 8: Das Ablaufdiagramm für die Generierung des Pfades

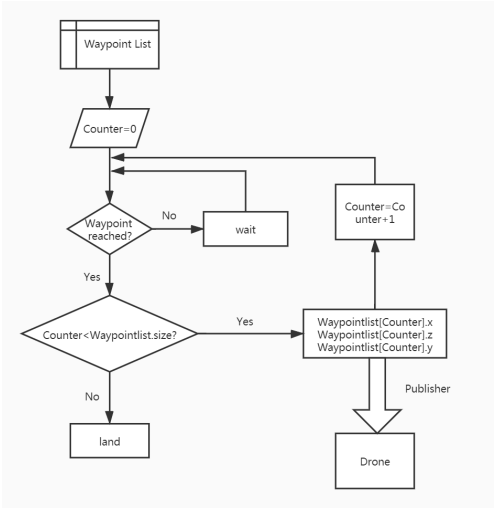

Abb. 9: Die in "Waypointlist" gespeicherten Punkte werden an die Drohne als Sollposition übergeben."Waypointlist" ist hier eine Matrix, die mit dem Buchstabe W dargestellt wird, deren Spalte jeweils der X, Y und Z-Komponente der vorher generierten Punkte entsprechen.

$$y_i = Waypointlist[i].y \tag{2}$$

$$z_i = Waypointlist[i].z \tag{3}$$

$$W = \begin{pmatrix} x_0 & y_0 & z_0 \\ x_1 & y_1 & z_1 \\ \vdots & \vdots & \vdots \\ x_{NPT-1} & y_{NPT-1} & z_{NPT-1} \end{pmatrix} = \left(\vec{X}, \vec{Y}, \vec{Z}\right) \tag{4}$$

Dabei gilt:

$$\vec{X} = (x_0, x_1, \cdots, x_{NPT-1})^T \tag{5}$$

$$\vec{Y} = (y_0, y_1, \cdots, y_{NPT-1})^T \tag{6}$$

$$\vec{Z} = (z_0, z_1, \cdots, z_{NPT-1})^T \tag{7}$$

Der in der Abbildung 8 gezeigte "intp" Teil ist die Interpolationsfunktion, die im Folgenden in Detail erklärt wird. Die hier verwendeten Interpolationsmethoden sind die lineare Interpolation und die Newtonsche Interpolation.

B. Lineare Interpolation

Hier wird die Pfad zwischen den gegebenen Koordinatenpunkten (Startpunkt, Zwischenpunkt und Zielpunkt) als Gerade betrachtet, d. h. die durch diese lineare Interpolation erzeugten Punkte liegen alle auf einer Linie. Hier ist ein Beispiel für die Interpolation in der xy-Ebene.

Hier seien zwei Parameter

$$X_{\text{Data}} = (x_s, x_z, x_e)^T$$
$$Y_{\text{Data}} = (y_s, y_z, y_e)^T$$

vorgegeben. Das bedeutet, die drei Koordinatenpunkte lauten

$$Startpunkt : \vec{A} = (x_s, y_s)^T \tag{8}$$

$$Zwischenpunkt : \vec{B} = (x_z, y_z)^T \tag{9}$$

$$Endpunkt : \vec{C} = (x_e, y_e)^T \tag{10}$$

Für die Punkte zwischen A und B gilt

$$dydx = (y_z - y_s)/(x_z - x_s) \tag{11}$$

$$y = y_s + m(x - x_s) \tag{12}$$

Analog gilt für die Punkte zwischen B und C

$$dydx = (y_e - y_z)/(x_e - x_z) \tag{13}$$

$$y = y_z + m(x - x_z) \tag{14}$$

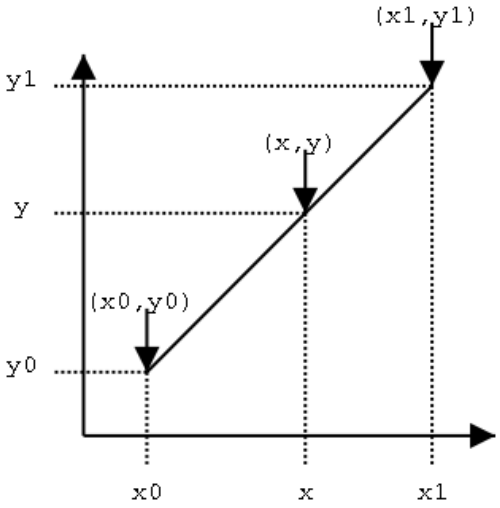

Abb. 10: lineare Interpolation

Die Abbildung 10 zeigt den Zusammenhang zwischen der X Koordinate und der Y Koordinate in 2D-Koordinaten. Auf die gleiche Weise wird die X- und Z-Koordinaten interpoliert, so dass eine Trajektorie im 3D-Raum abgeleitet werden kann.

C. Newtonsche Interpolation

Obwohl eine lineare Interpolation zwischen den gegebenen Koordinatenpunkten leicht möglich ist, kann der am Ende erhaltene Pfad nur eine gerade Linie sein. Die lineare Interpolation kann nicht verwendet werden, wenn die gewünschte Trajektorie eine Kurve sein soll. Daher wird die Newton-Polynom-Interpolation zur Lösung des Problems betrachtet.

Als Ansatz wird die Newtonsche Darstellung ausgewählt

$$p_3 = \gamma_0 + \gamma_1(x - x_s) + \gamma_2(x - x_s)(x - x_z) + \gamma_3(x - x_s)(x - x_z)(x - x_e) = intpl.x \tag{15}$$

mit

$$\gamma_0 = f_{[x_s]} \tag{16}$$

$$\gamma_1 = f_{[x_0,x_1]} \tag{17}$$

$$\gamma_2 = f_{[x_0,x_1,x_2]} \tag{18}$$

$$\gamma_3 = f_{[x_0,x_1,x_2,x_3]} \tag{19}$$

Zur Berechnung der Koeffizienten

$$\gamma_i = f_{[x_0,\ldots,x_i]}, i = 0, \cdots, 3.$$

sollte die folgende Vorschrift

$$f_{[x_i]} = y_i, i = 0, \cdots, 3. \tag{20}$$

$$f_{[x_i,\ldots,x_{i+k}]} = \frac{f_{[x_{i+1},\ldots,x_{i+k}]} - f_{[x_i,\ldots,x_{i+k-1}]}}{x_{i+k} - x_i} \tag{21}$$

für $i = 0, \cdots, 3-k$, und $k = 1, \cdots, 3$ berechnet.

Dann wird das Newtonsche-Interpolationspolynom durch das Schema erhalten. In der Abbildung 11 [12] beziehen sich x_0, x_1, x_2 jeweils auf x_s, x_z, x_e. Analogisch stehen y_0, y_1, y_2 jeweils für y_s, y_z, y_e.

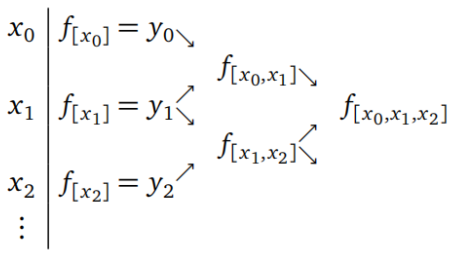

Abb. 11: Newtonsches Schema

Mit dem Newtonsche-Interpolationspolynom lässt sich den gewünschten Pfad ermitteln.

Es wird in Abbildung 12 dargestellt, dass bei der Newton-Interpolationsmethode nur die Koordinaten der Stützstelle benötigt wird, um eine Kurve anzupassen.

Darüber hinaus zeigt die Abbildung, dass mit Hilfe einer Kurve und einer X Komponente den zu berechnenden Wert der Y Komponente berechnet werden kann.

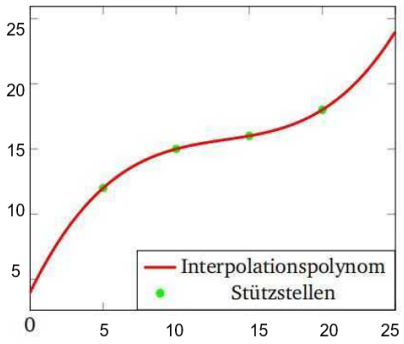

Abb. 12: Newtonsche Interpolation

V. Kollisionserkennung

Als Kollision wird das Zusammenstoßen zweier Objekte verstanden. Es keine vorhandene Schnittstelle, die Kollisionen zwischen den Objekten direkt detektieren kann. Deshalb wird eine Methode vorgestellt, die eine Kollisionserkennung durchführen kann.

A. Informationen für die Kollisionserkennung

Bevor eine Implementierung der Kollisionserkennung stattfinden kann, müssen zwei Fragen beantwortet werden. Wie kann eine Kollision erkannt werden? Ist die Kollision mit einer Drohne passiert?

Eine Kollision kann mit Hilfe der Simulation ermittelt werden. Hier bietet Gazebo Nachrichten über Objekte, die miteinander in Berührung stehen. Diese Nachrichten sind über den Topic /gazebo/default/physics/contacts erreichbar. Damit erhalten wir Informationen über die Zeit der Berührung, welche Objekte sich berühren, welche Teile der Objekte sich berühren und die Position im Raum, an der die Berührung stattgefunden hat. Es kann vorkommen, dass Nachrichten gesendet werden, die einen Wert für die Zeit haben, aber keinen für Berührungen. Deshalb sollten die Nachrichten vor der Verarbeitung gefiltert werden.

Ob die Kollision mit einer Drohne passiert ist oder nicht, wird mit Hilfe der Name der Drohnen realisiert. Diese werden mit dem Programm übergeben und haben den denselben Namen wie in der Simulation. Wie genau der Name der Drohne entsteht, wird in Sektion VI beschrieben.

Das ist die Basis, um eine Kollision zu erkennen. Im Programm wird ein Client für Gazebo erstellt und das Kollisionstopic mit einem Subscriber abonniert. Als nächstes wird die Logik für die Erkennung einer Kollision erläutert.

B. Erkennung einer Kollision

Das Programm verwendet folgende Logik für die Erkennung einer Kollision. Zuerst bekommt das Programm eine Nachricht mit einer Berührung. Es wird überprüft, ob es eine Drohne ist und ob die Drohne überwacht werden soll. Diese Informationen werden dem Programm vor dem Start übergeben. Danach wird überprüft, wann die letzte Berührung stattgefunden hat. Die Überprüfung soll verhindern, dass eine bestehende Berührung zwischen den Drohnen oder der Drohne

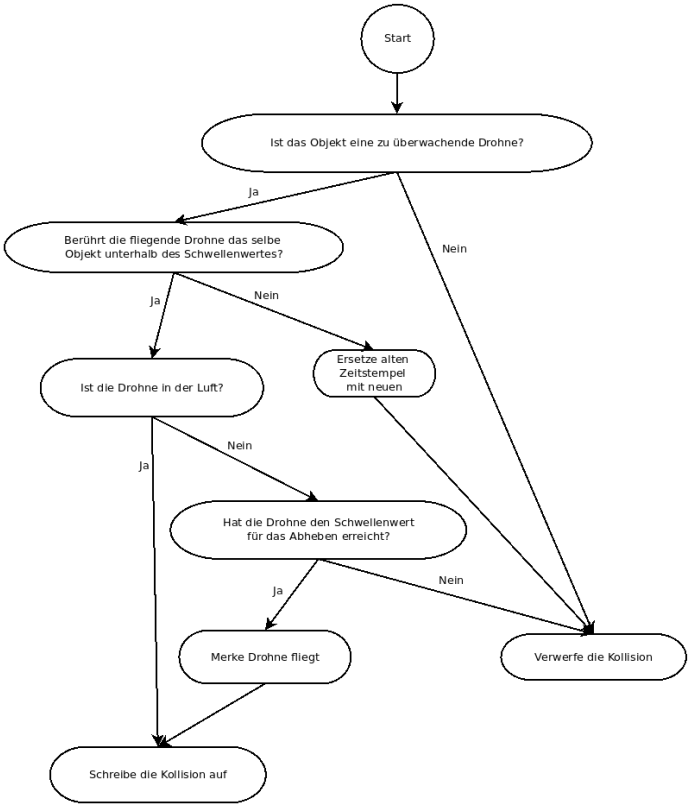

Abb. 13: Überprüfung für eine gültige Kollision

und einem Objekt als erneute Kollision detektiert wird. Zusätzlich wird hier überprüft, ob die Drohne schon in der Luft ist, da beim Simulationsbeginn alle Drohnen mit dem Boden Kontakt haben, was eine dauerhafte Kollisionsdetektion zur Folge hat. Zum Schluss wird überprüft, ob die Drohnen schon in der Luft sind oder nicht. Falls sie in der Luft sind, wird die Berührung in die Datei geloggt. Falls sie sich noch auf dem Boden befinden, wird eine finale Überprüfung durchgeführt. Diese vergleicht die aktuelle Position der Drohne mit einem Schwellenwert, der definiert, wann eine Drohne sich in der Luft befindet. In diesem Fall wurde 0,1 von der Z-Achse gewählt.

Die Abbildung 13 visualisiert die oben genannte Schritte.

C. Speicherung der Kollisionsdaten

Nachdem die Nachricht überprüft wurde, wird diese in einer Datei gespeichert. Der Name der Datei ist aus dem Prefix ”Collision” und dem Datum mit der Zeit zusammemgesetzt. Das Programm wird durch eine beliebige Eingabe beendet. Am Ende des Programms wird die Gesamtanzahl der Kollisionen ausgegeben.

Folgende Tabelle I soll die gespeicherte Werte und deren Bedeutung erklären. Dabei ist die Position nicht der Mittelpunkt der Drohne, sondern die Position von einem Teil der Drohne mit dem kollidierten Objekt.

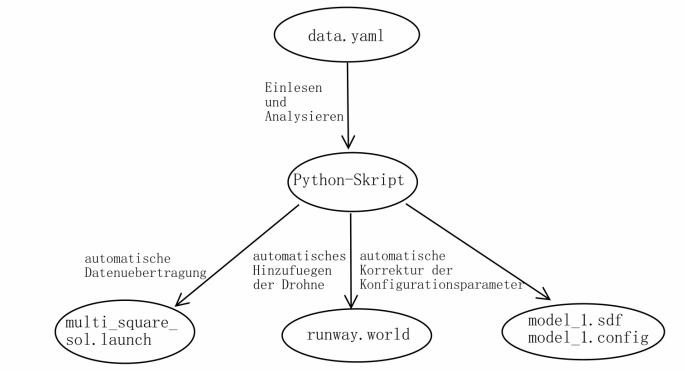

Abb. 14: die Struktur des Python-Skripts

Bezeichnung	Bedeutung
Count	Nummer der Kollision
Collision 1	Name des ersten Kollisionsobjekts
Collision 2	Name des zweiten Kollisionsobjekts
X	x-Position der Kollisionsposition
Y	y-Position der Kollisionsposition
Z	z-Position der Kollisionposition
Sec	Kollisionszeitpunkt in Sekunde
Nsec	Kollisionszeitpunkt in Nanosekunde

TABELLE I: Bezeichnung und Erklärung der gespeicherten Kollision

VI. Skript für die Vorbereitung der Simulation

Um eine kompliziertere Flugevalidierung zu realisieren, müssen mehrere Drohnen in die Simulationswelt hinzugefügen und jeweils kontrollieren werden. Ein Skript in der Programmiersprache Python soll diese Aufgaben erledigen. Im Vergleich zu anderen Programmiersprachen wie C oder mit der Skriptsprache Shell ermöglicht Python eine schnelle Implementierung eines Skript mit Hilfe von Bibliotheken. Zusätzlich ist die Programmiersprache für Personen mit geringen Programmierkenntnisse einfach zu erlernen. Die Abbildung 14 zeigt die Struktur des Python-Skripts.

A. Extrahieren der Daten aus der Konfigurationsdatei

Eine Konfigurationsdatei soll als Schnittstelle zwischen Anwender und Simulationsframework dienen und soll mit der Auszeichnungssprache YAML geschrieben werden. YAML ermöglicht eine menschenfreundliche Serialisierung der Daten. [13] Daher könnnen die Daten auch durch den Menschen schnell und einfach geändert werden. Außerdem hat Python ein fertiger Programmgerüst für YAML, das beim Parsen der Daten hilft. In der Konfigurationsdatei werden die Namen der Drohnen sowie die Anfangs-, Zwischen- und Endpunkte auf den jeweiligen Flugrouten gespeichert. Dabei dient der Name der Drohne als Namensraum.

B. Erstellung und Änderung der ROS Porgramm

Die eingelesenen Daten aus der Konfigurationsdatei sollen in den verschiedenen Dateien an den richtigen Stellen vom Skript eingefügt werden.

Zuerst wird die gewünschte Drohnenanzahl in der Simulationswelt hinzugefügt. Die gewünschte Drohnenanzahl wird aus den verschiedenen Namen der Drohnen ermittelt. Die Drohnen müssen auf die selbe Weise wie die Gebäude aus Sektion III in die world-Datei hinzugefügt werden. Für das Einfügen einer Drohne werden die Informationen über die Namen des Modells, die gewünschte Position in der Simulation und den Namen der Drohne gebraucht. Die gewünschten Positionen und die Namen stehen in der YAML-Konfigurationsdatei, wobei die Orientierung der Drohne für alle gleich null gesetzt wird. Der Name des Modells ist der Ordnername, wo das Aussehen, Größe und Ausstattung der Drohne in der Simulation beschrieben wird. In dem Skript wurde eine Vorlage für das Einfügen neuer Drohnen intern gespeichert. Das Skiprt öffnet die world-Datei und schreibt die angepassten Vorlagen in die Datei rein.

In diesem Projekt haben alle Drohne das selbe Modell. Jedoch braucht jede Drohne seinen eigenen Ordner, die die angepassten Konfigurationsdateien enthalten. In den Konfigurationsdateien des Modells wird der Name und die Ports der Drohne definiert, die das Skript ändern muss, damit externe Programme verschiedene Befehle and die verschiedene Drohnen senden können. Hierbei wurde eine Kopiervorlage mit Platzhaltern erstellt. Das Skript liest die Vorlagen ein, fügt die passende Werte in den Platzhaltern ein und erstellt den Ordner mit den angepassten Konfigurationsdateien des Modells.

Damit alle Drohnen gleichzeitig gestartet werden können, erstellt das Skript eine launch-Datei. Diese launch-Datei startet die Drohnen mit dem gemeinsamen Pfadplanungsprogramm, aber unterschiedlichen Parametern. In der selben launch-Datei wird das Kollisionsprogramm hinzugefügt, damit das Programm automatisch mit den Drohnen gestartet werden kann. Die Abbildung 16 zeigt das Ergebnis, wenn die launch-Datei gestartet wird.

Die Abbildung 15 zeigt die zuvor erklärte Schritte, die das Skript durchläuft.

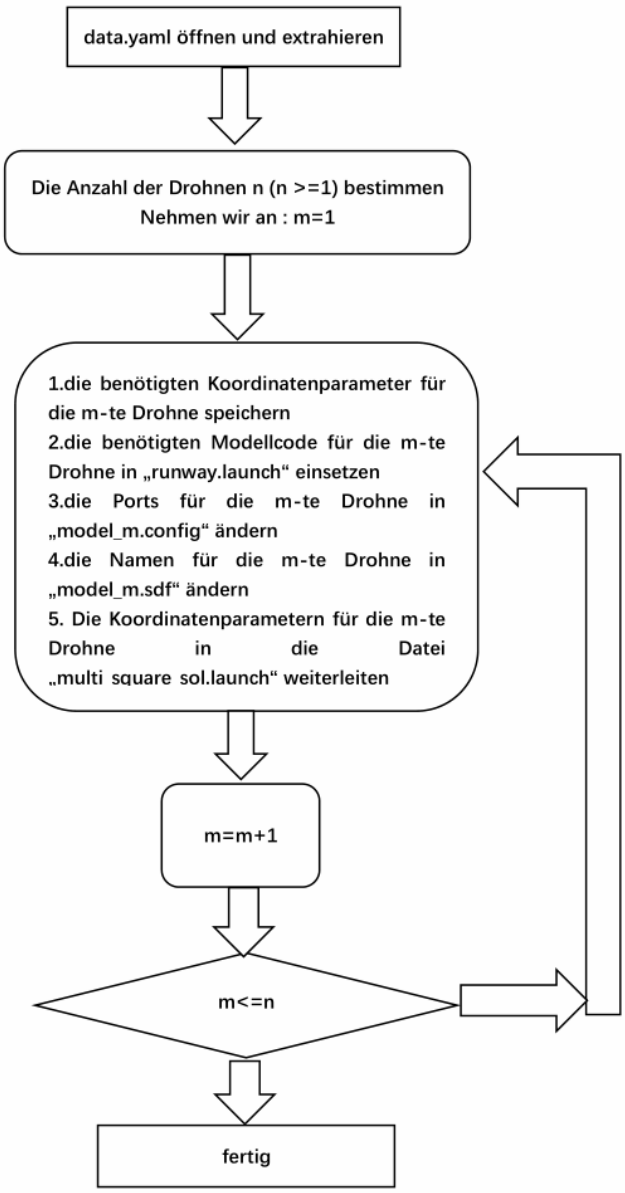

Abb. 15: Programmschritte des Skripts

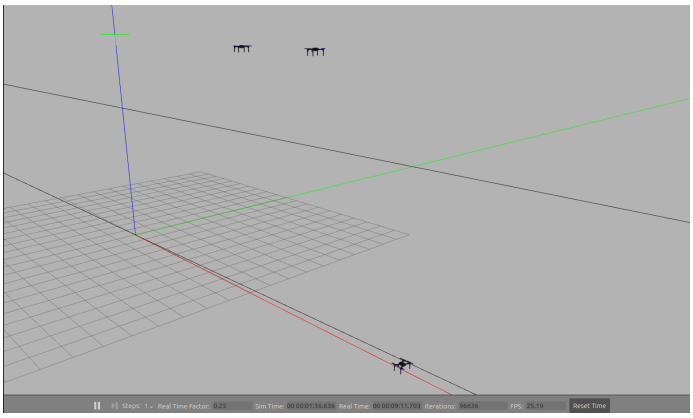

Abb. 16: Zwei Drohnen in der Simulation starten lassen

VII. Zusammenfassung

Nach dem Abschluss des Projekts kann eine Simulation mit importieren 3D-Daten aus OpenStreetMaps und einer beliebigen Anzahl an Drohnen gestartet werden. Zusätzlich ist es möglich mit Hilfe von ROS mehrere Drohnen in der Simulationswelt fliegen zu lassen und gleichzeitig deren Kollision zu loggen.

Für die zukünftige Ausarbeitung kann die Bahnberechnung in einem separaten Programm erstellt werden und das zentrale Programm verteilt die Wegpunkte an die entsprechende Drohnen. Des Weiteren kann die Schnittstelle zwischen Befehlsprogramm und Bodenkontrollstation erweitert werden und somit kompliziertere Drohnenprogramme ermöglichen. Nachdem die Schnittstelle erweitert wurde, kann anstelle der Bahnberechnung die Trajektorie der Drohnen validiert werden, die in diesem Projekt nicht implementiert wurde. Die Simulationswelt hat nur Drohnen als dynamische Objekte und durch das Einfügen von andere dynamische Objekte wie Vögel würde das Ergebnis der Simulation mehr der Realität

entsprechen. Außerdem kann das Skript erweitert werden, dass auch die importierte 3D-Modelle automatisch in die Simulationswelt eingefügt werden. Die Kollisionserkennung und die Zuverlässigkeit des Kollisionsprogramm kann optimiert werden.

DANKSAGUNG

Wir danken unserem Betreuer Nikolas Hohmann, M.Sc. für den einfachen Einstieg in unserem Projekt.

LITERATURVERZEICHNIS

[1] I. Quads, "Intelligent quads tutorials," aufgerufen: 24.01.2021. [Online]. Available: https://github.com/Intelligent-Quads/iq_tutorials
[2] O. S. R. Foundation, "Gazebo - robot simulation made easy," aufgerufen: 24.01.2021. [Online]. Available: http://gazebosim.org/
[3] J. M. M. F. I Alonso, M. Fernandez.
[4] ArduPilot, "Ardupilot versatile, trusted, open," aufgerufen: 28.01.2021. [Online]. Available: https://ardupilot.org/
[5] O. S. R. Foundation, "Ros documentation," aufgerufen: 24.01.2021. [Online]. Available: http://wiki.ros.org/
[6] C. Sciortino and A. Fagiolini, "Ros/gazebo-based simulation of quadcopter aircrafts," 2018.
[7] "mavros ros package," aufgerufen: 28.01.2021. [Online]. Available: http://wiki.ros.org/mavros#mavros.2BAC8-Plugins.sys_status
[8] O. Foundation, "Openstreetmap," aufgerufen: 28.01.2021. [Online]. Available: https://www.openstreetmap.org/
[9] B. Foundation, "Blender," aufgerufen: 28.01.2021. [Online]. Available: https://www.blender.org/
[10] V. C. Lab, "Meshlab," aufgerufen: 24.01.2021. [Online]. Available: https://www.meshlab.net/
[11] T. K. G. Inc, "Collada overview," aufgerufen: 29.01.2021. [Online]. Available: https://www.khronos.org/collada/
[12] P. D. S. Ulbrich, *Vorlesungsskript Mathematik IV für Elektrotechnik Mathematik III für Informatik*. TU Darmstadt, 2019.
[13] "Yaml: Yaml ain't markup language," aufgerufen: 29.01.2021. [Online]. Available: https://yaml.org/

Christian Maier studiert Autonome Systeme an der Technischen Universität Darmstadt, wo er voraussichtlich 2022 seinen Masterabschluss machen wird. Davor hat er den Bachelorstudiengang Technische Informatik an der Hochschule Augsburg gemacht.

Longfei Li studiert Elektro- und Informationstechnik mit dem Schwerpunkt Automatisierungstechnik an der Technischen Universität Darmstadt, wo er voraussichtlich 2022 seinen Masterabschluss machen wird. Im gleichen Studiengang hat er 2018 seinen Bachelor-Abschluss gemacht.

Yingyi Zhang studiert Elektro- und Informationstechnik mit dem Schwerpunkt Automatisierungstechnik an der Technischen Universität Darmstadt, wo sie voraussichtlich 2022 ihren Masterabschluss machen wird. Im gleichen Studiengang hat sie 2019 ihren Bachelor-Abschluss gemacht.

Fengyun Shao studiert Elektro- und Informationstechnik mit dem Schwerpunkt Automatisierungstechnik an der Technischen Universität Darmstadt, wo er voraussichtlich 2022 seinen Masterabschluss machen wird. Im gleichen Studiengang hat er 2018 seinen Bachelor-Abschluss gemacht.