DEADULUS_SIM_REV_A Dokumentation

Release 0.1

Dipl.-Ing. (FH) Andreas Gschossmann

14.07.2015

The Kalman Filter in its various forms is clearly established as a fundamental tool for analyzing and solving a broad class of estimation problems.

Leonhard McGee and Stanley Schmidt,
Ames Research Center, NASA

Abkürzungsverzeichnis

UAV	Unmanned Aerial Vehicle
GUI	Graphical User Interface
QR	Quick Response
PID	Proportional Integral Derivative
GPS	Global Positioning System
IMU	Inertial Measurement Unit
LQ	Linear Quadratic
API	Application Programming Interface
XML	Extensible Markup Language
SDF	Simulator Description Format
TCP	Transmission Control Protocol
IP	Internet Protocol
MT	Momentum Theorie
3D	Dreidimensional
SIL	Software in the Loop

KAPITEL 1

Einleitung

1.1 Ziel der Arbeit

Bei der Entwicklung von Flugalgorithmen für den autonomen Flug eines Quadrocopters ist eine Simulation unvermeidbar. Es kann der Code für riskante Flugmanöver getestet und solange ausgereift werden bis dessen Grundfunktionen nachgewiesen werden können. Außerdem besteht die Möglichkeit verschiedene Verfahren für Flugregelung, Sensorfusion und andere Algorithmen verglichen werden. Damit gelignt es die Algorithmen durch einen gezielten Stimulus zu beeinflussen. Die erhobenen Daten können aufgenommen werden und im Anschluss direkt verglichen werden. Ausgehend vom zu testenden Algorithmus kann ausgewählt werden, ob ideale Werte oder verrauschte Sensordaten verwendet werden. Eine Simulation stellt zwar keinen Ersatz von Praxistests dar, jedoch kann der Entwicklungszyklus für die Firmware des UAVs erheblich verkürzt werden.

1.2 Stand der Technik

Um einen Quadrocopter simulieren zu können, muss für jeden Simulationsschritt die Lage und Position im Raum aus den am Rahmen anliegenden Kräften und Momenten berechnet werden. In vielen Arbeiten wird hierfür das dynamische Model des Quadrocopters hergeleitet. Meist werden die Bewegungsleichungen mit der Newton-Euler Methode hergeleitet. [19] zieht neben der Newton-Euler Methode den Euler-Lagrange Formalismus heran um die Bewegungsgleichungen aufzustellen.

Die Kräfte auf den Rahmen resultieren aus den von den Rotoren erzeugtem Schub. Das Yaw-Moment resultiert aus den Reaktionsmomenten die durch Lagerreibung und vor allem durch Luftwiderstand der Rotoren (Drag) entstehen. Eine Methode diese mechanischen Einflüsse aus den Drehzahlen der Rotoren zu gewinnen ist die Impuls Theorie (Momentum Theory) wie sie beispielsweise in [21] beschrieben ist. Der einfachste Ansatz ist dabei den Schwebezustand anzunehmen und die dafür benötigten Konstanten über Schubmessungen eines fest eingespannten Motors zu gewinnen [17], [25].

Andere Simulationsansätze berücksichtigen zur Schubberechnung einen Ansatz nach [26], welcher durch die Verbindung von Impuls-Theorie und Blattelement Theorie eine Berücksichtigung der Relativgeschwindigkeit des Quadrocopters in vertikaler Richtung zulässt [17]. [14] nimmt den in [26] genannten Ansatz auf und erweitert ihn so, dass er neben den Effekten von vertikaler Geschwindigkeit auch den Einfluss von Vorwärtsflug berücksichtigt. Auch [1] behandelt aerodynamische Effekte im Vorwärtsflug. In [26] wird der Ground- und Deckeneffekt untersucht und in [13] wird auch der Flapping-Effekt von Rotoren behandelt.

Die Theorie überlappender Motoren wird in [11] beschrieben. Wind wird in [5] und [32] behandelt und ein Controller zum Ausgleich des Windes vorgestellt. Auch [18] und [29] beschäftigt sich mit dem Ausgleich von Positionsdrift, hervorgerufen durch Windturbulenzen.

In [2] wird ein PID und ein LQ-Regler zur Regelung der Fluglage vorgeschlagen und verglichen. In [23] werden neben PID und LQ-Regler zusätzlich ein Fuzzy Regler vorgestellt und verglichen. Eine Regelung basierend auf Quaternionen wird in [31] vorgestellt. In [30] werden ebenfalls auf Quaternionen basierende Algorithmen vorgestellt, welche Coriolis- und gyroskopische Momente kompensiert.

In [23] wird ein mehrstufiges Kalman Filter System vorgeschlagen, welches durch die Werte von Magnetfeld und IMU die Lage des Quadrocopters stützt. Die Zustandsmatrizen dieses Filters werden bei Verfügbarkeit von GPS-Daten online erweitert und es werden nun auch Position und Geschwindigkeit mitgeschätzt. Ein Filter für reine Lagestützung wird in [22] vorgestellt. Der Ansatz zur Lagestützung von [24] schätzt mit einem Ansatz basierend auf einem extended Kalman Filter (EKF) die Geschwindigkeit mit.

Eine kommerzielle Simulationsumgebung ist RotorLib [27]. Diese ist kostenpflichtig. [8] wählt einen sehr schlichten Ansatz über Matlab. Da eine grafische Rückmeldung der Flugbewegung des UAVs ratsam ist, werden in [9] die Positionswerte über Simulink berechnet und zur grafischen Anzeige in den 3D-Flugsimulator FlightGear eingespeist. Die 3D-Darstellung von FlightGear verbraucht jedoch sehr viel Ressourcen. Deshalb müssen hier zwei Rechner verwendet werden. Ein anderer Ansatz ist [10], [16], welcher ebenfalls FlightGear verwendet. Hier wird jedoch die von FlightGear verwendete Aerodynamik-Simulation JSBsim verwendet. JSBsim ist dabei eine eigenständige und quelloffene C++ API, welche getrennt von FlightGear betrieben werden kann [6]. Simulationsumgebungen im Überblick:

- **FlightGear:** Open Source Flugsimulator mit der Möglichkeit Plugins zur programmieren. Wahlweise werden JSBSIM oder YAsim als Aerodynamik API genutzt.
- **YAsim, JSBsim:** Quelloffene Aerodynamik API.
- **Rotorlib:** Kommerzielle Simulationssoftware.
- **USARSim:** Quelloffene Simulationsumgebung, basierend auf der Unreal Tournament Game Engine.
- **Gazebo:** Quelloffene Simulationsumgebung mit vielen Features.

1.3 Rahmenbedingungen

1.3.1 Anforderungen

Im Folgenden, werden die wichtigsten Anforderungen an die Simulationsumgebung [1] zusammengetragen:

- Berechnung von Fluglage und Position im Raum aus den Kräften und Momenten auf den Rahmen
- Simulation von Sensoren (IMU, Magnetfeld, GPS)
- Berechnung der aerodynamischen Kräfte und Momente aus den Motordrehzahlen (Impulstheorie)
- Berechnung der Kräfte und Momente resultierend aus den Massen und Trägheitsmomenten des Körpers (Trägheit, gyroskopischer Effekt)
- Grafische Darstellung

[1] Die hier entwickelte Simulationssoftware für Quadrocopter wurde *DeadalusSim* genannt.

- Möglichkeit Simulationsdaten in Dateien zu speichern
- Modularer und zugänglich dokumentierter Code um Erweiterungen möglich zu machen

Das Ablaufdiagram aus Abbildung 1.1 zeigt dabei einen Teil der Anforderungen.

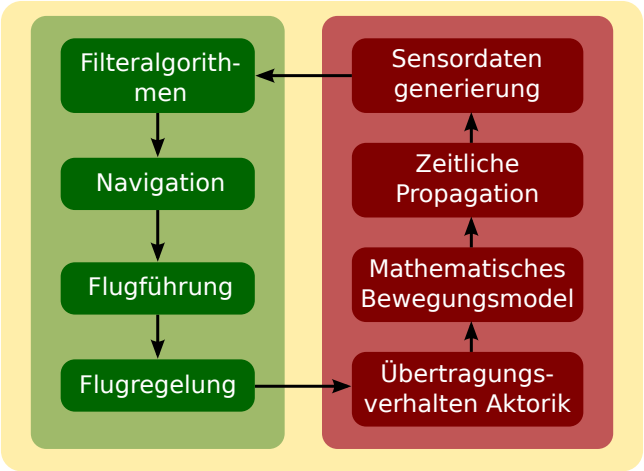


Abbildung 1.1: Simulations Framework [17]

Links (grün hinterlegt), befindet sich der simulierte Code, rechts (rot hinterlegt) der Ablauf, der Simulation selbst.

1.3.2 Vereinfachungen

Folgende Vereinfachungen werden angenommen:

- Der Rahmen wird als starr angenommen.
- Der Rahmen ist symmetrisch.
- Der geometrische Mittelpunkt des Rahmens und Schwerpunkt fallen zusammen.
- Die Rotoren werden als starr angenommen.
- Sowohl Ground- und Deckeneffekte als auch Auftriebseffekte durch Vorwärts- oder Vertikalflug werden zunächst vernachlässigt.

1.3.3 Wahl der Simulationsumgebung

Da die 3D-Darstellung von Gazebo ressourcenschonend ist und viele Features, wie beispielsweise die Simulation von Sensoren bietet, fiel die Wahl auf diese Simulationsumgebung. Es wurde ein vergleichbarer Ansatz wie er in [17] und [25] vorgestellt wurde, gewählt.

1.4 Gazebo

Gazebo ist eine rundenbasierende Open Source Robotik-Simulationsumgebung mit Visualisierung. Sie berechnet zyklisch aus den Kräften und Momenten, welche an einem starren Körper anliegen, dessen Lage und Position im Raum. Dafür zieht sie eine der unterstützten Physik-Engines heran. Sie bietet dabei folgende Features:

- Unterstützung von derzeit vier Physik-Engines (ODE, Bullet, Simbody, Dart)
- Kollisionsmodell
- Simulation von Sensoren (Beschleunigungssensor, Kameras, GPS), konfigurierbar mit Rauschen
- Räumliche Darstellung der Gazebo-World durch sogenannte Digital Elevation Models (DEM), welche online verfügbar sind [15]
- Ressourcenschonende grafische Darstellung, falls der Rechenaufwand noch vertretbar ist auch in Echtzeit

Gazebo kann modular durch programmierte Plugins, programmiert in C++, in einer CMake Umgebung erweitert werden. Dabei sind vier Plugin-Arten möglich (World, Model, Sensor, System). Die API von Gazebo ist dabei gut dokumentiert. Modelle, die in die Gazebo-World geladen werden, werden in XML-Files, im von Gazebosim vorgegebenem SDF-Format, definiert. Hier werden dessen Eigenschaften, wie beispielsweise Trägheitsmomente, Gewicht, visuelle Darstellung, Abmessungen des Kollisionsmodells und vieles mehr festgelegt. Auch das SDF-Format ist gut auf der Website von Gazebosim dokumentiert.

Die Gazebo Kommunikation funktioniert mit TCP/IP Sockets. Messages werden auf Kanälen, sogenannten Topics via Publisher versandt. Empfangen werden sie durch einen sogenannten Subscriber. Der Code hierfür wird durch Überschreiben der jeweiligen Funktionen der Basisklassen in die Gazebo-Plugins eingefügt.

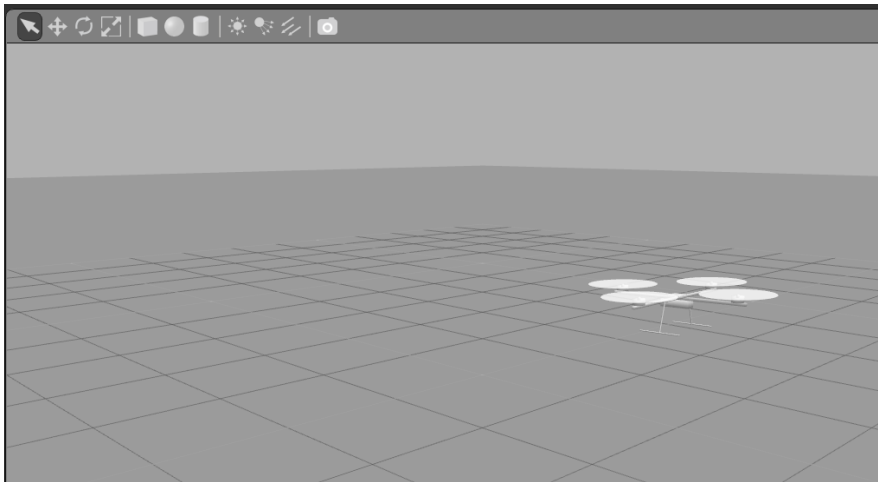


Abbildung 1.2: GUI von DeadalusSim

KAPITEL 2

Quadrocopter Modell

2.1 Kinematik

Abbildung 2.1 zeigt die verwendeten Koordinatensysteme. Das Inertialkoordinatensystem (N, E, D) ist fest mit der Erde verbunden. Dessen Achsen zeigen in Richtung der Himmelsrichtungen ((N)orth, (E)east) und des Erdursprungs ((D)own). Das körperfeste Koordinatensystem (x, y, z) ist fest mit dem Rahmen des UAVs verbunden. Die Vorwärtsflugrichtung stimmt dabei mit der x-Achse überein.

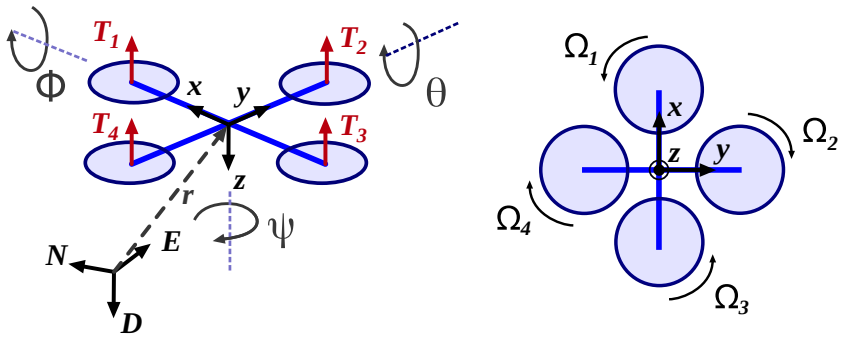


Abbildung 2.1: Koordinatensysteme und Drehrichtung der Rotoren

Für die Umrechnung der Orientierung des körperfesten Koordinatensystems (x, y, z) zum Inertialkoordinatensystem (N, E, D) wird die Richtungskosinusmatrix [17], [20], [23] verwendet:

$$R = \begin{bmatrix} c\theta c\psi & s\phi s\theta c\psi - c\phi s\psi & c\phi s\theta c\psi + s\phi s\psi \\ c\theta s\psi & s\phi s\theta s\psi + c\phi c\psi & c\phi s\theta s\psi - s\phi c\psi \\ -s\theta & s\phi c\theta & c\phi c\theta \end{bmatrix}$$

Die Position des UAVs im Inertialkoordinatensystem wird durch den Vektor r dargestellt:

$$r = [N\ E\ D]^T \tag{2.1}$$

2.2 Dynamik

2.2.1 Aerodynamik

Entsprechend der Impulstheorie (Momentum Theory, MT) gelten im Schwebeflug folgende Zusammenhänge für den Schub T_i und das vom Rotor verursachte Lastmoment Q_i [21]:

$$T_i = \underbrace{C_T \rho A R^2}_{k_T} \Omega_i^2, \quad M_i = \underbrace{C_M \rho A R^3}_{k_M} \Omega_i^2 \tag{2.2}$$

Dabei ist ρ die Dichte der Luft, A die vom Rotor überstrichene Fläche und R ist der Radius des Rotors. Kombiniert man die Impulstheorie und die Blattelement Theorie, lässt sich ein Zusammenhang für den Schub herleiten, der eine Berücksichtigung der Relativgeschwindigkeit zulässt [17].

$$T = C_{T,0}\Omega^2 + C_{T,1} v_1 \Omega^2 + C_{T,2} v_1^2 \tag{2.3}$$

$(v_1)_i$ ist dabei die Geschwindigkeit in Richtung des Kraftvektors des Motors i. Unter Einbezug der Winkelgeschwindigkeit des Rahmens ω^b lässt sie sich nach Gleichung (2.4) berechnen. l_M ist dabei der Abstand der Motor-Achse zum Zentrum des Rahmens [17].

$$\begin{aligned} (v_1)_1 &= v_z + \omega_y^b \cdot l_M, \quad & (v_1)_3 &= v_z - \omega_y^b \cdot l_M \\ (v_1)_2 &= v_z - \omega_x^b \cdot l_M, \quad & (v_1)_4 &= v_z + \omega_x^b \cdot l_M \end{aligned} \tag{2.4}$$

2.2.2 Kräfte und Momente auf den Rahmen

Durch die Gleichungen (2.2) können die Kräfte und Momente auf den Rahmen berechnet werden. Diese werden durch den Schub und das Lastmoment der Motoren, hervorgerufen durch Lagerreibung und Luftwiderstand, berechnet.

$$F_M^b = \begin{bmatrix} 0 \\ 0 \\ k_T(\Omega_1^2 + \Omega_2^2 + \Omega_3^2 + \Omega_4^2) \end{bmatrix} \qquad M_M^b = \begin{bmatrix} k_T l_M(\Omega_4^2 - \Omega_2^2) \\ k_T l_M(\Omega_1^2 - \Omega_3^2) \\ k_M(\Omega_2^2 + \Omega_4^2 - \Omega_1^2 - \Omega_3^2) \end{bmatrix} \tag{2.5}$$

Außerdem resultiert aus dem gyroskopischen Effekt, hervorgerufen durch die Drehzahl der Motoren und den durch sie zu überwindendnen Trägheitsmomenten J_r, das Moment M_G^b:

$$M_G^b = \begin{bmatrix} J_r \omega_y(\Omega_1 + \Omega_3 - \Omega_2 - \Omega_4) \\ J_r \omega_x(\Omega_2 + \Omega_4 - \Omega_1 - \Omega_3) \\ 0 \end{bmatrix} \tag{2.6}$$

2.2.3 Bewegungsgleichungen nach der Newton-Euler Methode

Durch die in Kapitel *Kräfte und Momente auf den Rahmen* aufgestellten Gleichungen, lassen sich unter Verwendung der Newton-Euler Methode die Bewegungsgleichungen für Translation [1] und Rotation aufstellen:

$$\begin{aligned} J\dot{\omega^b} + \omega \times J\omega &= M_M^b - M_G^b \\ m\ddot{r} &= [0\ 0\ mg]^T - RF_M^b \end{aligned} \tag{2.7}$$

[1] May the Mass times Acceleration be with you.

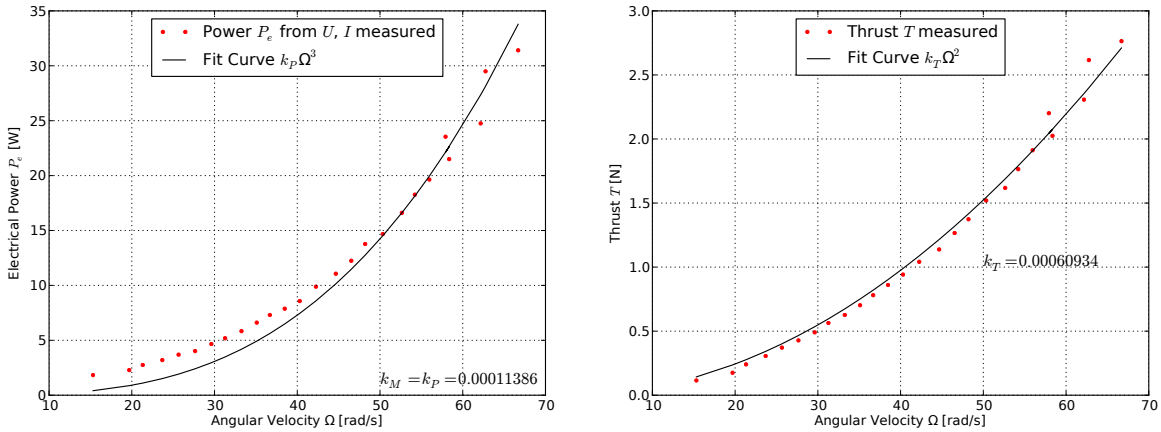
2.2.4 Motor Dynamik

Es hat sich herausgestellt, dass es zweckmäßig ist, das Verhalten des Motors als Differentialgleichung erster Ordnung darzustellen. Dafür wird eine Gleichung verwendet, wie sie ebenfalls in [3] hergeleitet wurde. Dabei ist u die Stellgröße des Motors.

$$K \cdot u + C = T \cdot \dot{\omega} + \omega \tag{2.8}$$

2.3 Identifikation der Systemkonstanten aus Messungen

Um die in Kapitel *Dynamik* vorgestellten Gleichungen in der Simulation verwenden zu können, müssen die darin enthaltenen Konstanten aus Messungen ermittelt werden. Die Konstanten k_M und k_T aus den Gleichungen (2.2) wurden durch die in Abbildung 2.2 dargestellten Messungen ermittelt. Zunächst wurde die Simulation auf einen Quadrocopter mit 440 mm Achsabstand und Robbe Roxxy BL2827-34 Motoren mit 10 Zoll Motoren ausgelegt.



(a) (b)

Abbildung 2.2: Messungen der Leistung und des Schubs über der Drehzahl für Robbe Roxxy BL2827-34

Da entsprechend Gleichung (2.2) eine quadratische Funktion zu erwarten ist, wurde die Konstante des Schubbeiwerts k_T in Abbildung 2.2 (b) durch die Least-Square-Fit Methode angenähert. Aus [28] ist bekannt, dass der Leistungsbeiwert k_P und der Momentbeiwert k_M identisch sind. Da für das Moment des Motors keine Messungen vorliegen, musste der Momentbeiwert in Abbildung 2.2 (a) über den Leistungsbeiwert hergeleitet werden. Dabei wurden Messungen der elektrischen Eingangsleistung des Motortreibers verwendet. Hier muss darauf geachtet werden, dass für die tatsächlich abgegebene Leistung der Wirkungsgrad berücksichtigt werden muss. Dieser wird hier auf etwa 80 Prozent geschätzt.

In Gleichung (2.8) wird das dynamische Verhalten des Motors vorgestellt. Da für das zeitliche Verhalten derzeit keine Messungen vorliegen wurde zunächst nur das stationäre Verhalten identifiziert. In Abbildung 2.3 werden die Konstanten K und C ermittelt. In der Simulation wird die fehlende Größe T zunächst auf Null gesetzt.

Die hier identifizierten Werte können jederzeit analog für andere Ausleger/Motor/Rotor-Kombinationen berechnet werden.

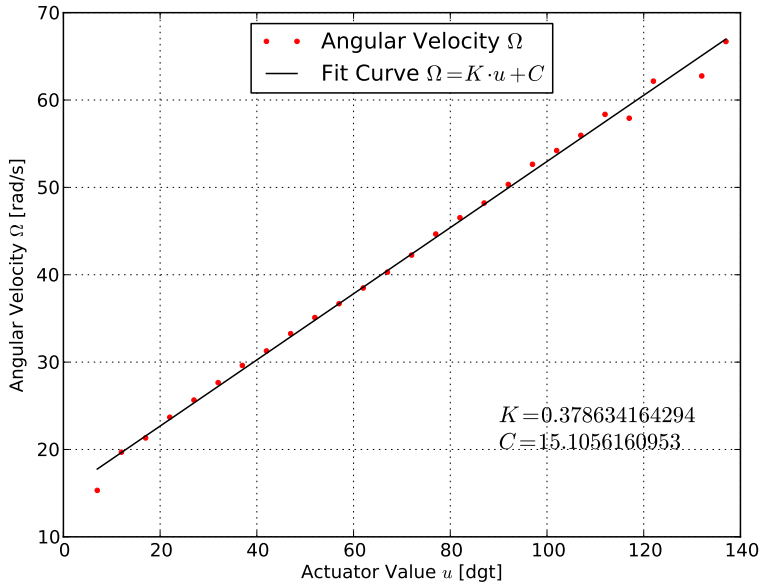


Abbildung 2.3: Systemkonstanten der Motordynamik im stationären Fall

KAPITEL 3

Simulation

3.1 Aufbau

Abbildung 3.1 zeigt den Aufbau der Simulation. Gazebo übernimmt dabei die Lösung der Bewegungsgleichungen aus den Kräften und Momenten zur Ermittlung der Fluglage und Position. Dafür wird von Gazebo eine Physik-Engine verwendet. Auch die 3D-Darstellung wird von Gazebosim übernommen. Da Gazebosim jedoch keine Aerodynamik kennt, werden die in Kapitel *Dynamik* vorgestellten Gleichungen für die Berechnung der Schubkräfte und der daraus resultierenden Momente auf den Rahmen verwendet. Wie bereits in Kapitel *Identifikation der Systemkonstanten aus Messungen* erwähnt, wird aufgrund von fehlenden Messungen, zunächst vereinfachend nur der Schwebeflug bei der Schubberechnung berücksichtigt.

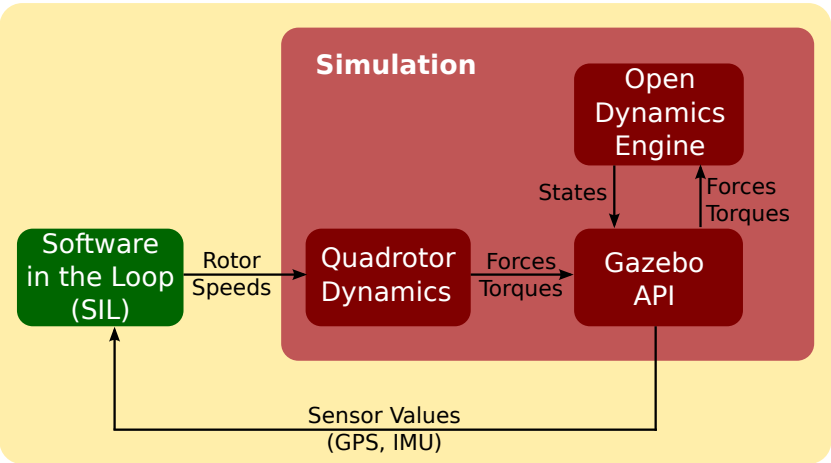


Abbildung 3.1: Simulations Framework [17]

3.2 Gazebo-Plugin Klassen

Es wurden zwei Plugins für Gazebo programmiert. Ein Model-Plugin und ein Sensor-Plugin:

- **GazeboModelPlugin:** Diese Klasse enthält eine Instanz von *SIL* (siehe Kapitel *Software in the Loop (SIL)*) und *QuadRotorDynamics* (Kapitel *Quadrocopter Dynamik Klassen*). Es werden die aus der

Instanz von *SIL* gewonnenen Stellwerte an die Instanz von *QuadRotorDynamics* weitergegeben. Hier werden die Kräfte und Momente berechnet. Diese Kräfte und Momente werden durch Befehle der Gazebo API an den Quadrocopter-Rahmen angelegt. Außerdem werden die simulierten Sensor-Werte der IMU an die Instanz von *SIL* übergeben.

- **GazeboIMUPlugin:** Diese Klasse veröffentlicht die simulierten Sensor-Werte der IMU, damit auf diese in der Klasse *GazeboModelPlugin* zugegriffen werden kann.

Diese Klassen stellen die Schnittstelle zur Simulationsumgebung Gazebosim dar. Hierfür werden die Befehle der Gazebo API verwendet. Mehr Informationen können auf der Website von Gazebo gefunden werden [7].

3.3 Quadrocopter Dynamik Klassen

Da die API von Gazebo derzeit keine Aerodynamik Funktionalität anbietet, wurde die Klasse **QuadRotorDynamics** programmiert, um diese nachzubilden. Aufgabe dieser Klasse ist, durch die Gleichungen aus Kapitel *Dynamik*, aus den Stellwerten zunächst durch ein PT1-Verhalten die Drehzahlen zu berechnen. Aus den Drehzahlen werden schließlich durch die Formeln aus Gleichung (2.2) die Schubkräfte und das Yaw-Moment berechnet und aus den Formeln der Gleichungen (2.5) das Roll und Pitch Moment. Da sich die Momente des gyroskopischen Effekts im Schwebeflug auflösen und oft vernachlässigt werden, können diese wahlweise aktiviert werden.

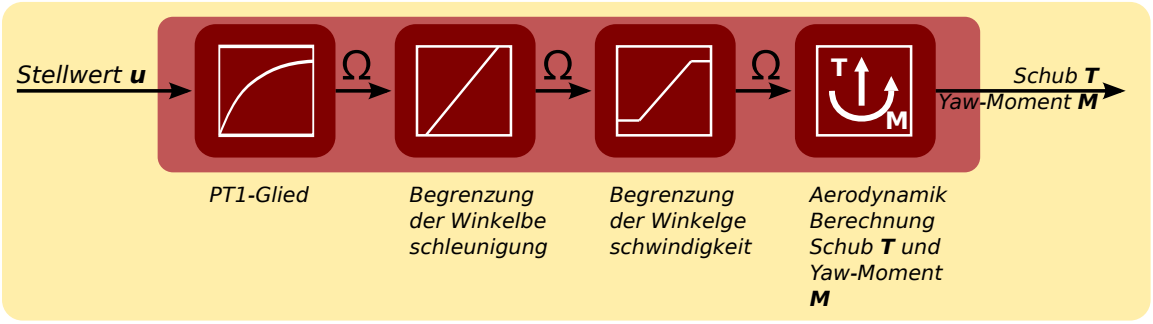


Abbildung 3.2: Ablauf der Berechnung des Dynamischen Verhaltens des Quadrocopters

Da in der Realität die Motordrehzahl und die Winkelbeschleunigung der Motoren begrenzt ist, können in *QuadRotorDynamics* die Grenzwerte für die Simulation eingestellt werden. Werden diese überschritten, werden sie in der Simulation auf die maximal möglichen Werte begrenzt. Der gesamte Ablauf der Klasse *QuadRotorDynamics* wird in Abbildung 3.2 gezeigt.

3.4 Software in the Loop (SIL)

Alle Funktionalitäten um die zu testende Firmware (Software in the Loop, SIL) des QuadroCopters in das Gazeboframework einzubinden befinden sich in der Klasse *SIL*. Es werden hier die simulierten Sensorwerte, die Simulationszeit und die realen Euler-Winkel zur Verfügung gestellt. Die berechneten Motor-Stellwerte werden von dieser Klasse an die Simulation zurückgeliefert. Außerdem besteht die Möglichkeit Eingaben

eines PlayStation 3 [1] Controllers auszulesen. Es gibt auch Funktionen um gewünschte Stimuli an die Simulation zu liefern.

Die wichtigste Aufgabe dieser Klasse ist es, in jedem Simulationsschritt, die zu testende Firmware auszuführen. Zu diesem Zweck wird die Update-Funktion der Instanz dieser Klasse in *GazeboModelPlugin* ausgeführt.

[1] Sony PlayStation 3 - DualShock 3 Wireless Controller

KAPITEL 4

Regelung

4.1 Backstepping-Regler

Um schnell Simulationsergebnisse erhalten zu können wurde ein Backstepping-Regler für die Lageregelung implementiert. Dabei wird der Regelausdruck, aus Gleichung (4.1) verwendet (hier für den Rollwinkel ϕ dargestellt). Für die Herleitung dieses Ausdrucks sei auf [23] verwiesen.

$$u = \frac{J_x}{l_M}((1 + a_1 \cdot a_2)(\phi_{set} - \phi) - (a_1 + a_2)\omega^b_{ib,x}) \tag{4.1}$$

ϕ_{set} stellt dabei den Stellwert des Anstellwinkels der Rollachse, ϕ, den tatsächlichen Wert des Rollwinkels, l_M den Abstand der Rotorachse zur Rahmenmitte, J_x das Massenträgheitsmoment um die x-Achse und $\omega^b_{ib,x}$ die x-Komponente der Werte des Drehratensensors. Über die Einstellungen der Parameter a_1 und $a2$ kann das Verhalten des Reglers beeinflusst werden.

Die Reglerparameter werden dabei zunächst empirisch gewählt und sind nicht optimiert. Die Winkel werden zunächst noch nicht durch Filteralgorithmen aus den Sensorwerten gewonnen. Statt dessen werden vorerst die realen Winkelwerte der Simulation verwendet.

4.2 Steuerung

Es besteht die Möglichkeit den Quadrocopter der Simulation über einen PlaySation 3 Controller zu steuern.

Abbildung 4.1: PlayStation 3 Controller

Der Controller kann drahtlos über Bluetooth genutzt werden. Hierfür wird der sogenannte *QtSixA*-Treiber [33] verwendet. Dies ist ein angepasster Joystick-Treiber für Linux-Systeme, der den Playstation 3 Controller unterstützt. Details zu Installation und Verwendung dieses Treibers finden sich im Anhang unter Kapitel *Playstation 3 Controller Treiber installieren*.

4.3 Mixer

Mit dem Mixer wird festgelegt, in welcher Weise die Motor-Stellwerte auf die einzelnen Motoren verteilt werden. In diesem Fall wurde die T-Flug-Konfiguration aus Gleichung (4.2) gewählt.

$$\begin{aligned} u_1 &= ps3_{gas} + y_{yaw} + y_{pitch} \\ u_2 &= ps3_{gas} + y_{yaw} - y_{pitch} \\ u_3 &= ps3_{gas} - y_{yaw} + y_{roll} \\ u_4 &= ps3_{gas} - y_{yaw} - y_{roll} \end{aligned} \tag{4.2}$$

KAPITEL 5

Ausblick

Mit DeadalusSim wurde hier eine Simulationsumgebung vorgestellt mit der man bereits das Flugverhalten eines Quadrocopters untersuchen kann. Diese kann nun stetig weiterentwickelt werden.

Einige der Arbeiten welche noch geplant sind:

- *Sensor Plugin für Magnetfeldsensor schreiben:* Da Gazebo derzeit noch keine Magnetfeldsensor nativ angeboten wird, muss hier noch Code geschrieben werden. Am einfachsten ist es, eine Klasse zu schreiben, welche die Werte eines fixen 3D-Vektors im körperfesten Koordinatensystem zunächst in den Ursprung des Ineritalkoordinatensystems des simulierten UAVs verschiebt und ihn dann durch Quaternionenrotationen in jenes Inertialkoordinatensystem umrechnet.
- *Erweiterung um GPS Sensor:* Gazebo bietet die Klassen für die Simulation eines Global Positioning System (GPS)-Sensors. Diese müssen noch implementiert werden.
- *Verbesserung des Aerodynamik Modells:* Das Aerodynamik-Modell kann durch die Berücksichtigung des Ground-Effekts verbessert werden. Desweiteren kann der Einfluss von Vorwärtsflug und vertikaler Geschwindigkeit auf das Flugverhalten einbezogen werden. Hierfür sind jedoch Windkanalmessungen notwendig.
- *Schätzung des Flugverhaltens durch ein neuronales Netz:* Anstatt die aerodynamischen Werte von den Rotoren durch Messungen zu ermitteln, könnte man die nichtlinearen Zusammenhänge des Flugverhaltens des Quadrocopters, durch Anlernen eines neuronalen Netzes, mit Messungen aus realen Flügen erhalten. Hier könnte ein ähnlicher Ansatz wie in [4] gefunden werden.

ANHANG A

Installation

A.1 Gazebosim installieren

Gazebo kann entweder über die Paketquellen installiert werden [1].

```
sudo sh -c 'echo "deb http://packages.osrfoundation.org/gazebo/ubuntu precise
main" > /etc/apt/sources.list.d/gazebo-latest.list'
wget http://packages.osrfoundation.org/gazebo.key -O - | sudo apt-key add -

sudo apt-get update
sudo apt-get install gazebo3
```

oder von gazebo.org heruntergeladen werden. Die hier vorgestellten Codes wurden mit Version 3.0 von Gazebo getestet.

A.2 Playstation 3 Controller Treiber installieren

Zunächst muss der *QtSixA*-Treiber [33] installiert werden. Dieser bietet die Möglichkeit PS3 Hardware an eine Linux-Maschine anzuschließen:

```
sudo add-apt-repository ppa:falk-t-j/qtsixa
sudo apt-get update
sudo apt-get install qtsixa
```

Nun kann der PS3 Controller bentutzt werden:

```
# plug controller into machine (via wire)
sudo sixpair
# unplug controller
sixad --start
# hold PS button
```

Wird ein PS3 Controller angeschlossen, kann man das simulierte UAV damit steuern. Falls kein PS3 Controller gefunden wird, wird eine Funktion mit vordefinierten Stimuli aufgerufen.

[1] Alle Schritte, die für die Kompilierung und Installation, der hier entwickelten Simulation notwendig sind wurden unter Ubuntu 12.04 getestet.

A.3 DeadalusSim kompilieren und starten

Herunterladen und enpacken:

```
mkdir mysim && cd mysim
scp -P 4444 gsa39665@hps.hs-regensburg.de:html/files/deadalussim_v0.1.2015.tar.bz2 .
tar xvjf deadalussim_v0.1.2015.tar.bz2
```

Kompilieren mit

```
cd DeadalusSim/source
cd build
sudo cmake ..
sudo make
cd ..
```

Ausfuehren mit

```
cd source
gazebeo gazebo_models/quadrotor.sdf
```

oder einfach durch aufruf des Skripts in diesem Ordner

```
sudo ./startup_sim.sh
```

Falls die Steuerung mit Playstation3 Controller gewünscht ist müssen vorher in einer eigenen Konsole folgende Befehle ausgeführt werden:

```
# plug controller into machine (via wire)
sudo sixpair
# unplug controller
sixad --start
# hold PS button
```

ANHANG B

Class Documentation

Um Seiten zu sparen wurde die Klassendokumentation nur online abgelegt, der QR-Code aus Abbildung B.1 führt zu den Online-Quellen.

Abbildung B.1: Link zur Klassendokumentation der DeadalusSim-Software [12]

Literaturverzeichnis

[1] Stjepan Bogdan and Matko Orsag. *Recent Advances in Aircraft Technology*. Intech, 2012.

[2] André Bouabadallah and Roland Siegwart. Pid vs lq control techniques applied to an indoor micro quadrotor. *IEEE International Conference Intelligent Robots and Systems*, 2004.

[3] Samir Bouabdallah, Pierpaolo Murrieri, and Roland Siegwart. Design and control of an indoor micro quadrotor. *IEEE International Conference on Robotics and Automation*, 2004.

[4] Travis Dierks and Sarangapani Jagannathan. Output feedback control of a quadrotor uav using neural networks. *IEEE Transactions on Neural Networks*, 2010.

[5] Escareño, Salazar, Romero, and Lozano. Trajectory control of a quadrotor subject to 2d wind disturbances. *Journal of Intelligent and Robotic Systems*, 2013.

[6] Flight dynamics and control software library in C++. JSBsim. http://www.jsbsim.org, 2012. [Online; accessed 23-Mai-2012].

[7] GAZEBO. Robot Simulation Framework. http://gazebosim.org. [Online; accessed 25-Mai-2012].

[8] Andrew Gibiansky. Quadrotor Dynamics and Simulation. http://andrew.gibiansky.com/blog/physics/quadcopter-dynamics, 2012. [Online; accessed 23-Mai-2012].

[9] Rahul Goel, Sapan M. Shah, Nitin K. Gupta, and N. Ananthkrishnan. Modeling, simulation and flight testing of an autonomous quadrotor.

[10] James M. Goppert. *AN ADAPTABLE, LOW COST TEST-BED FOR UNMANNED VEHICLE SYSTEMS RESEARCH*. PhD thesis, Purdue University, 2011.

[11] Daniel A. Griffith and A. Gordon Leishman. A study of dual-rotor interference and ground effect using a free-vortex wake model. *In Proc. 58th Annual Forum and Technology Display of the American Helicopter Assoc.*, 2002.

[12] Andreas Gschossmann. Doxygen-dokumentation von deadalussim. zuletzt aufgerufen April 2015, https://hps.hs-regensburg.de/gsa39665/files/doxygen/index.html.

[13] Grabriel M. Hoffmann, Haomiao Huang, Steve Wasl, and Er Claire J. Tomlin. Quadrotor helicopter flight dynamics and control: theory and experiment. *AIAA Guidance, Navigation and Control Conference*, 2007.

[14] Haomiao Huang, Gabriel M. Hoffmann, Steven L. Waslander, and Claire J. Tomlin. Aerodynamics and control of autonomous quadrotor helicopters in aggressive maneuvering. *Robotics and Automation*, 2009.

[15] Earth Science Data Interface. ESDI. http://glcfapp.glcf.umd.edu:8080/esdi/index.jsp. [Online; accessed 23-Mai-2012].

[16] Jgoppert. Mavsim (github). zuletzt aufgerufen April 2015, https://github.com/jgoppert/mavsim.

[17] Meyer Johannes, Sendorbry Alexander, Kohlbrenner Stefan, Klingauf Uwe, and Stryk Oskar. Comprehensive simulation of quadrotor uavs using ros and gazebo. *Simulation, Modeling, and Programming for Autonomous Robots*, 2013.

[18] Kamran Joyo, Ahmed Faiz, Tanveer Hassen, Warsi Faizan, and A. T. Hussain. Altitude and horizontal motion control of quadrotor uav in the presence of air turbulence. *IEEE Conference on Systems, Process and Control*, 2013.

[19] Nonami Kenzo, Kendoul Farid, Suzuki Satoshi, Wang Wei, and Nakazawa Daisuke. *Autonomous Flying Robots*. Springer, 2010.

[20] Sastry Shankar Lee Deawon, Jin Kim H. Feedback linearization vs. adaptive sliding mode control for a quadrotor helicopter. *International Journal of Control, Automation, and Systems*, 2009.

[21] J. Gordon Leishman. *Principles of Helicopter Aerodynamics*. CAMBRIDGE UNVERSITY PRESS, 2000.

[22] R. Mahony, Tarek Hamel, and Jean-Michel Pflimlin. Nonlinear complementary filters on special orthogonal group. *HAL archives-ouverts*, 2010.

[23] Oliver Meister. *Entwurf und Realisierung einer Aufklährungsplattform auf Basis eines unbemannten Minihelikopters mit autonomen Flugfähigkeiten*. PhD thesis, Karlsruher Institut für Technologie (KIT), 2010.

[24] Micheal R. Molt, Dale E. Schinstock, and Robert M. Caplinger. Gps-aided ins solution with photogrammetry validation. *Aerotech Congress and Exhibition*, 2005.

[25] Amr Nagaty, Sajad Saeedi, Carl Thibault, Mae Seto, and Howard Li. Control and navigation framework for quadrotor helicopters. *Journal of Intelligent & Robotic Systems*, April 2013.

[26] Caitlin Powers, Daniel Mellinger, Aleksandr Kushleyev, Bruce Kothmann, and Vijay Kumer. Influence of aerodynamics and proximity effects in quadrotor flight. *13th International Symposium on Experimental Robotics*, 2012.

[27] RTdynamics. Rotorlib fdm - a helicopter dynamics simulation library. zuletzt aufgerufen April 2015, http://www.rtdynamics.de/.

[28] J. Seddon. *Basic Helicopter Aerodynamics*. BSP Professional Books, 1990.

[29] Nitin Sydney, Brendan Smyth, and Derek A. Paley. Dynamic control of autonomous quadrotor flight in an estimated. *52nd IEEE Conference on Decision and Control*, 2013.

[30] A. Tavebi and S. McGilvray. Attitude stabilization of a four-rotor aerial robot. *43rd IEEE Conference on desicion and control*, 2004.

[31] John Ting-Yung Wen and Kenneth Kreutz-Delgado. The attitude control problem. *IEEE Transactions on automatic Control*, 1991.

[32] Steven Waslander and Carlos Wang. Wind disturbance estimation and rejection for quadrotor position control. *Aerospace Conference*, 2013.

[33] Written and maintained by falkTX. Qtsixa sixaxis joystick manager. zuletzt aufgerufen April 2015, http://qtsixa.sourceforge.net/.