



**Hochschule
Bonn-Rhein-Sieg**
University of Applied Sciences



Fraunhofer
IAIS

Zustandsregelung für ein Mikroflugsystem zur Ansteuerung vorgegebener Wegpunkte in Innenräumen

Studiengang Elektrotechnische Systementwicklung
Fachbereich 03 Elektrotechnik, Maschinenbau und Technikjournalismus

Masterarbeit

zur Erlangung des akademischen Grades
Master of Engineering

vorgelegt von

Kalle Knipp
Matr. Nr. 9019317

Windhausen 3a
51491 Overath
+49 157/81676016
kalle.knipp@gmail.com

Erstprüfer: Prof. Dr. rer. nat. Wolfgang Joppich
Zweitprüfer: Prof. Dr. rer. nat. Irene Rothe

April 2018

Erklärung zur Master-Thesis

„Ich versichere hiermit, die von mir vorgelegte Arbeit selbstständig verfasst zu haben. Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten oder nicht veröffentlichten Arbeiten anderer entnommen sind, habe ich als entnommen kenntlich gemacht. Sämtliche Quellen und Hilfsmittel, die ich für die Arbeit benutzt habe, sind angegeben. Die Arbeit hat mit gleichem Inhalt bzw. in wesentlichen Teilen noch keiner anderen Prüfungsbehörde vorgelegen.

Mir ist bewusst, dass sich die Hochschule vorbehält, meine Arbeit auf plagiierte Inhalte hin zu überprüfen und dass das Auffinden von plagiierten Inhalten zur Nichtigkeit der Arbeit, zur Aberkennung des Abschlusses und zur Exmatrikulation führen kann.“

Ort, Datum

Unterschrift

Danksagungen

An dieser Stelle möchte ich mich besonders bei Herrn Dipl.-Inform. Rainer Worst und Herrn Prof. Dr. Wolfgang Joppich bedanken, die mir im Laufe meiner Arbeit immer mit gutem Rat zur Seite standen, und es mir ermöglicht haben, meine Masterarbeit am Fraunhofer Institut für intelligente Analyse- und Informationssysteme (IAIS) zu schreiben. Ebenfalls möchte ich mich bei Erik Zimmermann sowie Prof. Hartmut Surmann bedanken. Sie haben sich bei sämtlichen Fragen Zeit für mich genommen und mich mit guten Ratschlägen unterstützt. Ein besondere Dank geht an Kerstin Dolinski, die mit großer Sorgfalt und Geduld diese Arbeit Korrektur gelesen hat.

Abschließend möchte ich mich bei meinen Eltern bedanken, die mich während meines gesamten Studiums unterstützen. Ihr bietet mir die Möglichkeit, erfolgreich meinen Weg zu gehen.

Vielen Dank dafür!

Kurzfassung

In der Masterarbeit *Zustandsregelung für ein Mikroflugsystem zur Ansteuerung vorgegebener Wegpunkte in Innenräumen* wird die Entwicklung einer Positionsregelung für ein Mikroflugsystem vorgestellt. Damit ist es möglich, sowohl in einer bekannten als auch unbekannten Umgebung vorgegebene Wegpunkte automatisch anzusteuern. Die Lokalisation des Flugsystems findet mit interner Sensorik sowie mithilfe von zwei Laserscannern statt. Steht bereits eine Karte der Umgebung zur Verfügung, ist es möglich, einen Pfad zu einem vorgegebenen Zielpunkt zu berechnen und diesen Pfad automatisch abzufliegen.

Diese Arbeit wurde teilweise von der europäischen Kommission durch das Projekt TRADR unter der Nummer FP7-609763 gefördert.

Schlüsselwörter: #ROS, #3D-Lokalisierung, #Flugsystem, #MAV, #Laserscanner
#Zustandsregler, #Positionsregler

Abstract

The master-thesis *State-controller for a micro-aerial-vehicle to move to given waypoints in indoor-environments* presents the development of a position-controller for a micro-aerial-vehicle. With this controller it is possible to fly to given waypoints in known and unknown environments. Required position information is received by using two laserscanners and the internal sensors of the micro-aerial-vehicle. With existing maps of the environment the path to a given position can be calculated and automatically executed.

This thesis was partly funded by the European Commission from the project TRADR at the number FP7-609763.

Keywords: #ROS, #3D-localization, #micro-aerial-vehicle, #MAV, #laserscanner
#state-controller, #position-controller

Inhaltsverzeichnis

1	Einleitung	1
1.1	Motivation	1
1.2	Problemstellung	2
1.3	Aufbau der Arbeit	3
2	Stand der Technik	4
2.1	Mikroflugsysteme	4
2.2	Robot Operating System	5
2.3	Sensorik zur Positionsbestimmung	5
2.3.1	Inertial Measurement Unit	5
2.3.2	Laserscanner	6
2.3.3	Kameras	7
2.4	Pfadplanung	7
2.5	Regelungsverfahren für Mikroflugsysteme	8
2.5.1	Regelalgorithmen	8
2.5.2	Positionsregelung	10
3	Positionsregelung	12
3.1	Zustandsraumdarstellung	12
3.1.1	Mathematisches Darstellung eines Quadrocopters	13
3.1.2	Zustandsraummodell	17
3.2	Zustandsregelung	24
3.2.1	Zustandsrückführung	24
3.2.2	Führungsgrößenvorfilter	28
3.2.3	Zustandsregler mit Integrator im Stellgrößenpfad	30
3.2.4	Zustandsregelung für einen Hexacopter	39
3.2.5	Ermittlung von Zustandsgrößen unter Verwendung realer Sensorik	42
3.2.6	Zweistufenregelung	45
4	Systemdesign	48
4.1	Software	48
4.1.1	ROS	48
4.1.2	Rotors Simulator	50
4.2	Hardware	52
4.2.1	Asctec Neo	52
4.2.2	Hokuyo Laserscanner	53
4.2.3	Gesamtconfiguration	55

5	Implementierung	57
5.1	Zustandsregler als ROS-Node	57
5.2	Trajektorienregler	59
5.3	Konfiguration von RotorS Simulator	61
5.3.1	3D-Umgebung	61
5.3.2	Plugins	62
5.4	Inbetriebnahme verwendeter Hardwarekomponenten	65
5.4.1	Hokuyo UST-20LX	65
5.4.2	Savöx Servomotor	66
6	Ergebnisse und Evaluierung	68
6.1	Simulation in RotorS Simulator	68
6.1.1	VTOL und Rotation	68
6.1.2	Ansteuerung vorgegebener Wegpunkte	72
6.2	Evaluation mit dem realen Flugsystem Asctec Neo	81
6.2.1	VTOL	81
6.2.2	Vorgabe von Wegpunkten	82
7	Abschlussbetrachtung	84
7.1	Zusammenfassung	84
7.2	Ausblick	85
8	Anhang	86
8.1	Datenblätter	86
8.1.1	Datenblatt des Asctec Neo	87
8.1.2	Ausschnitt aus dem Datenblatt des Hokuyo UST-20LX	88
8.2	Verwendung der implementierten Software	89
8.3	Dateianhang	90

Abbildungsverzeichnis

1.1	Tradr Evaluation 2017 in Rotterdam	2
2.1	Flugmodell der Firma DJI mit der Bezeichnung <i>Matrice 600 Pro</i> für Anwendungen im industriellen Bereich [6]	4
2.2	Hokuyo UST-20LX. 130 Gramm	6
2.3	Zusammenhang zwischen Karte der Umgebung und Graph [11]	8
2.4	Blockschaltbild eines Standardregelkreises	9
2.5	Schematische Darstellung einer Landung mithilfe verschiedener Lokalisationsverfahren [23]	11
3.1	Schematische Darstellung eines Quadrocopters	12
3.2	Allgemeine Darstellung eines Zustandsraummodells (MiMo-System)	18
3.3	Darstellung der Eigenstabilität des Zustandsraummodells eines Flugsystems	23
3.4	Signalflussplan eines Zustandsraummodells mit Zustandsrückführung	24
3.5	Zusammenhang der Eigenwerte und des Verhaltens eines dynamischen Systems	25
3.6	Einregelverhalten des Zustandsraummodells eines Flugsystems mit Zustandsrückführung	26
3.7	Signalflussplan eines Zustandsraummodells mit Zustandsrückführung und Führungsgrößenvorfilter	28
3.8	Einregelverhalten des Zustandsraummodells eines Flugsystems mit Zustandsrückführung und Führungsgrößenvorfilter.	29
3.9	Signalflussplan eines Zustandsreglers mit I-Anteil im Stellgrößenpfad	30
3.10	Einregelverhalten eines Zustandsreglers für ein MAV ohne optimierte Eigenwerte.	32
3.11	Programmablaufplan zur Bestimmung geeigneter Eigenwerte für einen Zustandsregler mithilfe von Testsimulationen	33
3.12	Einregelverhalten eines Zustandsreglers für ein MAV mit optimierten Eigenwerten.	34
3.13	Resultierende Rotordrehzahlen bei einem Steigflug in z-Richtung um 2,0 Meter	36
3.14	Einregelverhalten eines Zustandsreglers für ein MAV mit optimierten Eigenwerten unter Berücksichtigung der Rotordrehzahlen	37
3.15	Resultierende Rotordrehzahlen bei einem Steigflug in z-Richtung um 2,0 Meter unter Berücksichtigung der Drehzahlgrenzen eines realen Flugmodells	38
3.16	Schematische Darstellung der Anordnung der einzelnen Rotoren des Flugmodells <i>Asctec Neo</i>	39
3.17	Einregelverhalten eines Zustandsreglers für das Flugsystem <i>Asctec Neo</i> mit optimierten Eigenwerten	41

3.18	Rotordrehzahlen des Flugsystems Asctec Neo bei einem Steigflug auf zwei Meter	42
3.19	Berechnung der Geschwindigkeit aus der Position mithilfe der einfachen Differenzierung und dem Savitzky–Golay-Algorithmus	44
3.20	Bestimmung der Geschwindigkeit durch Kombination der Position und der Beschleunigung	45
3.21	Schematischer Aufbau einer Positionsregelung als Zweistufenregler	46
4.1	Pfadplanung innerhalb einer Karte mithilfe des ROS-Nodes <i>Global Planner</i> [25]. Standard Pfadplanung oben, Pfadplanung mit A*-Algorithmus [17] unten	49
4.2	MAV ausgestattet mit einer Stereo Kamera im rotorS Simulator. Anzeige des resultierenden Kamerabildes in RViz	52
4.3	Propellerschutz für das Flugsystem Asctec Neo	53
4.4	Vorrichtung zur Montage eines Laserscanners an der Unterseite des Flugsystems Asctec Neo in Verbindung mit einem Servomotor	54
4.5	Gesamtconfiguration aller Hardwarekomponenten an dem Flugsystem Asctec Neo	55
5.1	Funktionsweise des Zustandsreglers für das Flugsystem Asctec Neo als ROS-Node	57
5.2	Funktionsweise des Trajektorienreglers zur Vorgabe von Wegpunkten	59
5.3	Vorgabe einer Zielkoordinate in einer vorhanden zweidimensionalen Karte mithilfe von RViz	60
5.4	3D Umgebung des RotorS Simulators mit dem Flugsystem Asctec Neo und verschiedenen frei wählbaren Objekten	62
5.5	Anschlussmöglichkeiten externer Komponenten an dem Flugsystem Asctec Neo	66
5.6	Steuersignal für Standard Servomotoren und die daraus resultierende Hebelstellung	67
6.1	Verhalten des Flugsystems Asctec Neo bei einem Steig- und Sinkflug	69
6.2	Aufteilung des Rotationswinkels um die z -Achse in drei Abschnitte	70
6.3	Signalflussplan eines Zustandsreglers mit Reset-Funktion	70
6.4	Verhalten des Flugsystems Asctec Neo bei Rotation um die z -Achse	71
6.5	Testumgebung für Versuche im RotorS Simulator	73
6.6	Verhalten des Flugsystems Asctec Neo bei einer Bewegung in translatorischer Richtung	74
6.7	Vorgabe einer Zielkoordinate in RViz mit anschließender Ansteuerung vorgegebener Wegpunkte	75
6.8	Darstellung des Flugsystems Asctec Neo in dem RotorS Simulator und den resultierenden Laserscandaten in Rviz	76
6.9	Darstellung der vorgegebenen Trajektorie im Vergleich zu den angesteuerten Wegpunkten. Rot → vorgegebene Trajektorie. Blau → abgeflogene Trajektorie	77
6.10	Vorgabe einer Zielkoordinate in RViz in einer unbekannten Umgebung mit anschließender Ansteuerung vorgegebener Wegpunkte	78

6.11	Verhalten des Flugsystems Asctec Neo bei einem Steig- und Sinkflug unter Verwendung realitätsnaher Sensorik	79
6.12	Darstellung der vorgegebenen Trajektorie im Vergleich zu den angesteuerten Wegpunkten unter Verwendung realitätsnaher Sensorik. Rot → vorgegebene Trajektorie. Blau → abgeflogene Trajektorie	80
6.13	VTOL-Verhalten der Asctec Neo in einer realen Umgebung	81
6.14	Ansteuerung eines vorgegebenen Wegpunktes innerhalb eines Raumes	82
6.15	Ansteuerung eines vorgegebenen Wegpunktes im Außenbereich	83
7.1	GUI zur Übergabe von Befehlen an den Positionsregler	85

Tabellenverzeichnis

3.1	Symboldefinitionen für Differentialgleichungen eines Quadrocopters	16
3.2	Bedeutung einzelner Parameter eines Zustandsraummodells	18
3.3	Parameter des Quadrocopters Pelican	23
3.4	Parameter des Hexacopters Asctec Neo	40

Abkürzungsverzeichnis

UAV	Unmanned arial vehicle
UGV	Unmanned ground vehicle
MAV	Micro aerial vehicle
IMU	Inertial Measurement Unit
TRADR	Long-Term Human-Robot Teaming for Robot-Assisted Disaster Response
SLAM	Simultaneous localization and mapping
ROS	Robot Operating System
GPS	Global Positioning System
ICP	Iterative Closest Point
VTOL	Vertical Take-Off and Landing
PWM	Pulsweitenmodulation
SDK	Software Development Kit
VTOL	Vertical Take-Off and Landing
GUI	Graphical User Interface
DGL	Differentialgleichung

1 Einleitung

1.1 Motivation

Mobile Roboter wie Mikroflugsysteme werden immer häufiger zur Unterstützung von Rettungskräften eingesetzt. So verfügt z.B. die Feuerwehr in Dortmund über ein Flugsystem, welches mit einer Wärmebildkamera ausgestattet ist. Damit ist es möglich, bei Einsätzen einen Überblick der Umgebung zu erhalten. Weiterhin sind mobile Roboter auch nutzbar, um gefährliche Umgebungen wie einsturzgefährdete Gebäude zu untersuchen. Dadurch können erste Informationen gesammelt werden, ohne dass sich Rettungskräfte einer Gefahr aussetzen müssen.

In dem EU Projekt TRADR (Long-Term Human-Robot Teaming for Robot-Assisted Disaster Response) beschäftigt man sich schwerpunktmäßig mit der Zusammenarbeit zwischen Mensch und Robotern. Dazu sind verschiedene Arten von mobilen Robotern wie UGVs (Unmanned ground vehicles) oder auch UAVs (Unmanned arial vehicles) vorgesehen. Diese werden mit den verschiedensten Arten von Sensorik wie Kameras oder Gas Detektoren ausgestattet. Mit dem Projekt soll ermöglicht werden, Rettungskräfte bei der Bergung und Ortung von Personen in Gefahrensituationen zu unterstützen. Zusätzlich können unterschiedliche Proben mithilfe von Robotern eingesammelt werden, um diese später zu untersuchen [31]. Mehrmals im Jahr findet ein Treffen der verschiedenen Projektpartner statt, um das Gesamtsystem zu testen. In Abbildung 1.1 ist ein Testgelände dargestellt, welches für Übungen innerhalb des TRADR-Projekts genutzt wird. Es handelt sich dabei um ein Trainingsgelände in Rotterdam. Damit können Einsatzszenarien für das System realitätsnah untersucht werden.

Auch in einem echten Einsatz konnte das TRADR-System bereits erfolgreich eingesetzt werden. So wurden nach dem schweren Erdbeben in Amatrice im September 2016 Roboter verwendet, welche innerhalb des TRADR Projekts entwickelt wurden. Diese machten Aufnahmen von zwei vom Einsturz gefährdeten Kirchen. Mithilfe der von den Robotern aufgenommenen Daten kann so das weitere Vorgehen geplant werden.



Abbildung 1.1: Tradr Evaluation 2017 in Rotterdam

1.2 Problemstellung

Um die Zusammenarbeit zwischen Mensch und Roboter möglichst flexibel und effizient gestalten zu können, gibt es eine Reihe an Anforderungen. Die Steuerung der Roboter sollte möglichst einfach und stressfrei ablaufen, um Rettungskräfte nicht zusätzlich zu belasten. Es ist wünschenswert, dass möglichst viele Aufgaben bereits automatisch bearbeitet werden. Betrachtet man speziell Flugsysteme, so erfordert die Steuerung ein hohes Maß an Konzentration, damit eine Kollision mit Hindernissen vermieden wird. Daher ist es sinnvoll, dass solche Systeme einen möglichst hohen Grad an Autonomie besitzen. Dadurch kann erreicht werden, dass eine manuelle Steuerung eines Flugsystems nicht oder nur teilweise erforderlich ist. Anstatt also das Flugsystem manuell steuern zu müssen, ist es deutlich effizienter, dem System lediglich einen Zielpunkt vorzugeben. Dieser wird dann automatisch angesteuert. Damit ein solch autonomes Flugverhalten gewährleistet werden kann, ist es allerdings notwendig, dass sich das Flugsystem selbst innerhalb einer bestimmten Umgebung lokalisieren kann. Dazu kann GPS genutzt werden. Wird das System allerdings innerhalb eines Gebäudes verwendet, kann es sein, dass GPS Signale stark abgeschwächt werden, bzw. nicht mehr vorhanden sind. Daher müssen andere Verfahren genutzt werden, um die Lokalisation eines mobilen Roboters innerhalb eines Gebäudes zu gewährleisten. Dazu können Laserscanner verwendet werden, welche an einem Flugsystem befestigt sind.

Ziel dieser Arbeit ist die Entwicklung einer Positionsregelung. Damit soll ein Flugsystem einen vorgegebenen Wegpunkt automatisch ansteuern. Zusätzlich wird davon ausgegangen, dass kein GPS zur Verfügung steht. Zur Ermittlung benötigter Positionsinformationen sollen daher u.a. Laserscanner verwendet werden.

1.3 Aufbau der Arbeit

In dieser Arbeit wird in Kapitel 2 zunächst der aktuelle Stand der Technik dargelegt. Dabei wird schwerpunktmäßig auf die Regelung von Mikroflugsystemen eingegangen. Dafür werden Regelverfahren vorgestellt und Beispiele für Positionsregelungen genannt. Zusätzlich werden mögliche Lokalisationsverfahren für Flugsysteme präsentiert, da diese die Voraussetzung für eine stabile Funktionsweise eines Positionsreglers sind. In Kapitel 3 wird die Entwicklung eines Zustandsreglers dargestellt, damit ein Flugsystem einen definierten Punkt automatisch ansteuern kann. Dafür wird zunächst auf die Herleitung eines Zustandsraummodells eingegangen. Dieses stellt die Grundlage zur Auslegung eines Zustandsreglers mit guter Regelgüte dar. Um das Regelverhalten zu prüfen, werden mithilfe von Matlab/Simulink einige Simulationen durchgeführt. Im darauf folgenden Kapitel (Kapitel 4) wird auf die verwendete Software eingegangen. Es wird u.a. das 3D-Simulationsprogramm rotorS Simulator vorgestellt sowie weitere benötigte Softwarekomponenten. Außerdem wird die verwendete Hardware genauer betrachtet. Die Anordnung von Laserscannern an dem verwendeten Flugmodell wird beschrieben und technische Details von verschiedenen Hardwarekomponenten dargestellt. Kapitel 5 befasst sich mit der Implementierung von entwickelter Software. Das ROS Framework wird genutzt und die dafür entsprechenden Nodes erstellt. Anschließend wird der entwickelte Positionsregler in Kapitel 6 evaluiert. Dazu wird zunächst der rotorS Simulator verwendet, um das Flugverhalten zu untersuchen. Anschließend wird das System in der Realität mit dem Mikroflugsystem *Asctec Neo* getestet. Die Ergebnisse dieser Arbeit werden in Kapitel 7 zusammengefasst. Des Weiteren werden Ideen und Hypothesen für nachfolgende Projekte vorgestellt. Anhänge wie Datenblätter der verwendeten Hardware finden sich schlussendlich im letzten Kapitel (Kapitel 8).

2 Stand der Technik

2.1 Mikroflugsysteme

Als Mikroflugsysteme oder auch Micro aerial vehicles (MAVs) werden in dieser Arbeit Flugsysteme bezeichnet, welche mehrere senkrecht nach unten wirkende Rotoren besitzen und ein Gesamtgewicht von fünf Kilogramm nicht überschreiten. Die Anzahl sowie die Anordnung der einzelnen Rotoren kann dabei variieren. So gibt es z.B. Micro aerial vehicles (MAVs) mit vier (Quadrocopter), sechs (Hexacopter) oder auch acht (Octocopter) Rotoren. Bereits für unter 50 Euro können solche Flugsysteme erworben werden. Hersteller wie DJI bieten nicht nur Flugsysteme für den Verbraucherbereich an, sondern auch für die industrielle Nutzung [6]. In Abbildung 2.1 ist ein Flugsystem von DJI mit der Bezeichnung *Matrice 600 Pro* dargestellt. Diese Plattform bietet die Möglichkeit, zusätzliche Hardware zu montieren.



Abbildung 2.1: Flugmodell der Firma DJI mit der Bezeichnung *Matrice 600 Pro* für Anwendungen im industriellen Bereich [6]

In dem EU-Projekt TRADR werden u.a. MAVs der Firma Ascending Technologies verwendet [2]. So wird das Flugmodell *AscTec Falcon* bereits von der Feuerwehr in Dortmund für Einsätze genutzt. Auch in dieser Arbeit wird für praktische Versuche ein Flugmodell der Firma Ascending Technologies verwendet. Dieses trägt die Bezeichnung *AscTec Neo* und wird im Abschnitt 4 noch genauer beschrieben [3]. Das MAV zeichnet sich vor allem dadurch aus, dass zusätzliche Komponenten wie Laserscanner oder Computer an das Flugsystem montiert werden können.

2.2 Robot Operating System

Unter ROS oder auch Robot Operating System ist ein Framework zu verstehen, um Software für die verschiedensten Arten von Robotern zu entwickeln. Es handelt sich dabei um eine Sammlung von verschiedenen Werkzeugen, Bibliotheken und Konventionen [25]. Es existieren sehr viele Programme, um spezielle Hardware wie Laserscanner ansteuern zu können. Diese Programme werden in ROS auch als *Nodes* bezeichnet. Das Robot Operating System (ROS) Framework wird kostenlos angeboten und ist somit für verschiedene Institute, Unternehmen oder Privatpersonen verfügbar. Wird neue Software von verschiedenen Institutionen entwickelt, kann diese wiederum der Öffentlichkeit zur Verfügung gestellt werden. Die Anwendungsbereiche von ROS erhöhen sich dadurch immer weiter. Auch in dieser Arbeit wird das ROS Framework genutzt, um Hardware ansteuern zu können und weitere Software zu entwickeln.

2.3 Sensorik zur Positionsbestimmung

Um die absolute Position im Raum eines MAVs bestimmen zu können, gibt es mehrere Arten von Sensorik, welche für eine solche Aufgabe geeignet sind.

2.3.1 Inertial Measurement Unit

Der wichtigste Sensor in einem Flugsystem ist dabei die Inertial Measurement Unit (IMU). Dieses Bauteil misst Rotationen und ist in jedem MAV verbaut, um eine stabile Fluglage zu gewährleisten. Die IMU setzt sich aus mehreren einzelnen Sensoren, wie einem Beschleunigungssensor und einem Gyroskop zusammen. Durch Sensordatenfusion ist es möglich, die Rotation um die x - und y -Achse zu messen. Oft ist auch ein Magnetfeldsensor, bzw. Kompass verbaut, um zusätzlich die Rotation um die z -Achse bestimmen zu können. Zur Fusion der einzelnen Sensordaten gibt es bereits viele verschiedene Ansätze. Die einfachste Art ist die Nutzung eines Komplementärfilters. Dieser Filter hat die folgende Form:

$$angle = 0,98 \cdot (angle + gyrData \cdot dt) + 0,02 \cdot accData \quad (2.1)$$

Dabei werden die Sensorwerte des Gyroskops (*gyrData*) und die Werte des Beschleunigungssensors (*accData*) miteinander verrechnet. Das Gyroskop liefert Rotationsgeschwindigkeiten. Dieser Sensor reagiert dabei schnell auf Änderungen, weist aber im stationären Zustand einen

Drift auf, d.h. auch wenn das Flugsystem nicht rotiert, gibt der Sensor einen Wert ungleich null aus. Der Beschleunigungssensor dagegen liefert oft ein größeres Rauschen bei schnellen Bewegungen, ist allerdings im stationären Zustand genauer. Mit dem in Gleichung 2.1 dargestellten Filter wird nun zur Bestimmung des endgültigen Winkels die Werte der Sensoren miteinander verrechnet und so die jeweiligen Vorteile beider Sensoren genutzt. Mit dieser Art von Filterung ist es allerdings nur möglich, die Rotation um die x - und y -Achse zu bestimmen, da kein Kompass mit in die Berechnung einbezogen wird. Sebastian Madgwick stellt ein Verfahren zur Fusion von drei-Achsen Gyroskopsensoren und Beschleunigungssensoren sowie drei-Achsen Magnetfeldsensoren vor [21]. Der Filter weist dabei einen geringen Rechenaufwand sowie hohe Effektivität bei geringer Abtastrate auf.

2.3.2 Laserscanner

Zur Erfassung von Umgebungsinformationen sind Laserscanner besonders gut geeignet. Einige Hersteller, wie *SICK* oder *Hokuyo*, bieten unterschiedliche Typen von Laserscannern an. Dabei stehen Modelle zur Verfügung, welche ein geringes Gewicht aufweisen, wodurch sich diese gut für die Montage an MAVs eignen. In Abbildung 2.2 ist ein Laserscanner von Hokuyo mit der Bezeichnung *UST-20LX* dargestellt. Dieser hat lediglich ein Gewicht von 130 Gramm und eine Reichweite von bis zu 20 Metern. Um mithilfe von Laserdaten eine Positionsbestim-



Abbildung 2.2: Hokuyo UST-20LX. 130 Gramm

mung zu erhalten, können SLAM-Verfahren genutzt werden. SLAM bedeutet Simultaneous localization and mapping und beschreibt, wie ein mobiler Roboter eine Karte der Umgebung erstellt und sich gleichzeitig in dieser Karte lokalisiert. Dieses Verfahren eignet sich daher besonders, um Positionsinformationen für ein Regelverfahren zu liefern. H. Durrant-Whyte und T. Bailey stellen in einem Artikel den grundlegenden Ablauf eines Simultaneous localization and mapping (SLAM)-Verfahrens [7] dar. Dabei wird die Bewegung und die Position des Roboters ohne Kenntnisse der Umgebung geschätzt. Bewegt sich ein mobiler Roboter durch eine unbekannte Umgebung kann dieser, mit der entsprechenden Sensorik, Orientierungspunkte in der Umgebung detektieren. Verändert der Roboter anschließend die Position, können wiederum neue Orientierungspunkte erfasst werden. Daraus lässt sich die Bewegung des Roboters ermitteln.

2.3.3 Kameras

Nicht nur Laserscanner eignen sich, um Umgebungsinformationen zu erfassen, sondern auch Kameras. Diese können mit entsprechenden Verfahren eine dreidimensionale Darstellung der Umgebung liefern. Es stehen dabei Verfahren zu Verfügung, welche eine (Mono Vision) oder zwei (Stereo Vision) Kameras nutzen. Mithilfe von Stereo Vision oder auch stereo-optischen Verfahren ist es möglich, die Entfernung zu einem Objekt per Triangulation zu messen. Es kann dabei eine größere Fläche mit nur zwei Sensoren abgedeckt werden. Wird dieses Verfahren zur Entfernungsmessung genutzt, ergibt sich daraus allerdings ein erhöhter Rechenaufwand. Dieser hängt auch stark von der Auflösung der beiden Kameras ab. Steht nicht genug Rechenleistung zur Verfügung, ist ein Echtzeitbetrieb nicht mehr gewährleistet. Durch aktuellen Lichtverhältnisse ergeben sich außerdem Abweichungen in der Genauigkeit des Verfahrens [16].

Mono Vision dagegen nutzt lediglich eine Kamera. Um eine dreidimensionale Darstellung zu ermöglichen, werden kontinuierliche Aufnahmen der Umgebung erstellt. Die Entfernung der aufgenommenen Objekte kann nicht wie bei Stereo Vision über Triangulation berechnet werden, da nur eine Kamera zur Verfügung steht. Daher ist es nötig, die Trajektorie zwischen den einzelnen Bildern zu schätzen. Christian Forster et al. beschreiben in einem Artikel von 2014 ein mono visuelles Verfahren [8]. Der vorgestellte Algorithmus arbeitet direkt mit den Intensitäten der einzelnen Pixel. Daraus ergeben sich Subpixel und die Genauigkeit wird erhöht. Das Verfahren wird auf MAVs angewendet und arbeitet dabei mit einer Rate von 55 Bildern pro Sekunde.

2.4 Pfadplanung

Damit ein mobiler Roboter möglichst effizient eine vorgegebene Position ansteuern kann, ist es notwendig, zunächst einen Pfad zu dem entsprechenden Ziel zu planen. Dabei wird unterschieden, ob bereits eine Karte der Umgebung zur Verfügung steht oder diese erst erstellt werden muss. Um einen Pfad zu planen, wird die Umgebung als Graph dargestellt. Dieser Vorgang wird in Abbildung 2.3 verdeutlicht. Es wird zunächst die Karte betrachtet und an einem Punkt gestartet. Gibt es Verzweigungen oder Sackgassen, werden diese in dem Graph eingetragen. Ist bereits die gesamte Umgebung bekannt, kann auch der Graph vollständig erstellt werden. Sind allerdings nur Teile der Umgebung bekannt, wird zunächst der vorhandene Teil des Graphen erstellt und nachträglich aktualisiert. Zur Planung eines Pfades kann nun im einfachsten Fall der erstellte Graph in der Tiefe oder Breite abgesucht werden. Bei der Tiefensuche wird bei der Wurzel begonnen und den Kanten bis zu einer Sackgasse gefolgt. Anschließend wird wieder die letzte Verzweigung betrachtet, bei der noch unbesuchte Äste vorhanden sind. Dieser Vorgang wiederholt sich solange bis der Graph vollständig untersucht ist. Die Tiefensuche kann allerdings dazu führen, dass Lösungen gefunden werden, welche deutlich längere Pfade zum Ziel zur Folge haben. Um den schnellsten Pfad zu finden, muss immer der gesamte Baum abgesucht und die gefundenen Pfade miteinander verglichen werden. Bei der Breitensuche hingegen wird jede horizontale Ebene nacheinander abgearbeitet.

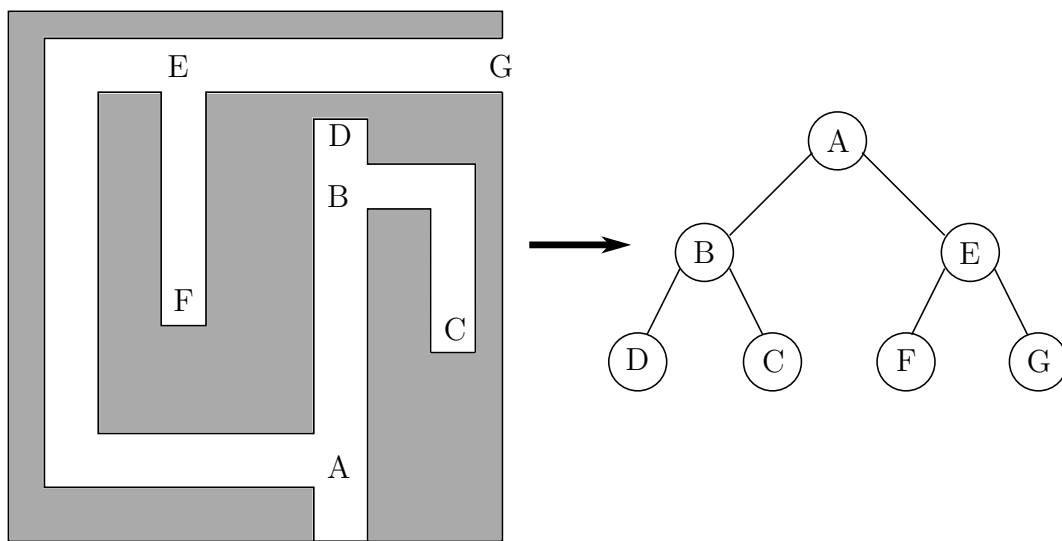


Abbildung 2.3: Zusammenhang zwischen Karte der Umgebung und Graph [11]

Wird hier ein möglicher Pfad gefunden, kann die Berechnung beendet werden und es müssen somit keine weiteren Ebenen abgesucht werden [11]. Eine spezielle Variante, um einen Pfad zu planen, wird von Patrick Lester dargestellt [17]. Dabei wird der A*-Algorithmus beschrieben. Es handelt sich dabei um ein heuristisches Verfahren, welches genutzt wird, um mithilfe von gewichteten Graphen den kürzesten Pfad zu einem Zielpunkt zu finden.

Auch in ROS wird bereits ein Node für die Pfadplanung angeboten. Dieser trägt die Bezeichnung *global_planner* und bietet einige Einstellmöglichkeiten. In Abschnitt 5 wird dieser ROS-Knoten verwendet und genauer auf die Konfiguration eingegangen.

2.5 Regelungsverfahren für Mikroflugsysteme

Im Folgenden wird auf verschiedene Verfahren eingegangen, um die Lage sowie die Position eines MAVs zu regeln. Dabei wird hier zwischen Lageregelung und Positionsregelung unterschieden. Lageregelung beschreibt im Folgenden die Regelung um die einzelnen Rotationsachsen. Also die Variante eines Reglers, welche genutzt wird, damit das Flugsystem eine bestimmte Neigung annimmt. Die Positionsregelung beschreibt zusätzlich die Möglichkeit, eine definierte Position in der Umgebung anzusteuern.

2.5.1 Regelalgorithmen

Jedes MAV verfügt bereits über eine Lageregelung, um eine stabile Fluglage zu gewährleisten. Als Sensorik ist für diese Art der Regelung lediglich eine IMU nötig, welche in jedem Flugsystem verbaut ist. Mithilfe der IMU können alle benötigten Sensorwerte wie die Rotation um die x - oder y -Achse ermittelt werden. Steht zusätzlich noch ein Magnetometer zur Verfügung, kann auch die Rotation um die z -Achse geregelt werden. In Abbildung 2.4 ist

ein Standardregelkreis dargestellt, um den Ablauf eines Regelverfahrens für ein MAV zu verdeutlichen. Zunächst wird dem System ein Sollwert $w(t)$ vorgegeben. Bei einem Flugsystem

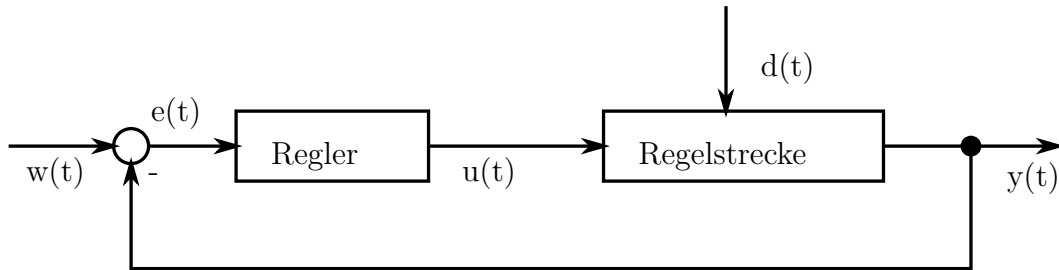


Abbildung 2.4: Blockschaltbild eines Standardregelkreises

sind das meist die einzelnen Rotationswinkel. Der aktuelle Istwert $y(t)$ wird anschließend von dem Sollwert subtrahiert. Daraus resultiert die Regeldifferenz $e(t)$. Die Istwerte $y(t)$ sind hierbei die entsprechenden Sensorwerte der IMU. Mithilfe eines entsprechenden Regelalgorithmus ergibt sich die Stellgröße $u(t)$. Die Stellgröße beschreibt bei einem MAV die einzelnen Rotordrehzahlen und wird an die Regelstrecke bzw. das Flugsystem übergeben. Als Resultat ergeben sich wiederum die aktuellen Istwerte.

Im einfachsten Fall kann ein PID-Regler genutzt werden, um eine stabile Fluglage gewährleisten zu können. Der Vorteil von solchen Regelverfahren besteht darin, dass die Regelstrecke nicht genau bekannt sein muss, um eine ausreichend gute Regelgüte zu erreichen. Es können die einzelnen Parameter für P, I und D einzeln gesetzt und das Flugverhalten anschließend geprüft werden. Das P-Glied beschreibt den proportional wirkenden Anteil. Wird ein Sollwert vorgegeben, wird dieser ohne Verzögerung mit einem Verstärkungsfaktor auf die Stellgröße geschaltet. Das I-Glied beschreibt hingegen das integrierende Verhalten des Reglers und dient hauptsächlich zum Ausgleich von Regeldifferenzen. Diese Differenzen können bei einem MAV auftreten, wenn das Flugsystem nicht symmetrisch aufgebaut ist und somit unterschiedliche Drehzahlen der einzelnen Rotoren im eingeschwungenen Zustand benötigt werden. Das D-Glied beschreibt den differenzierenden Teil eines PID-Reglers. Dieser reagiert auf die Änderungsgeschwindigkeit der Regeldifferenz $e(t)$. Bei einem MAV bewirkt dieser, dass bei einer Drehung um eine Achse vor Erreichen des Sollwertes eine Gegenbewegung ausgeführt wird. Dadurch wird ein Überschwingen des System reduziert bzw. vermieden. Ist die Regelstrecke des Flugsystems bekannt, können die benötigten Regelparameter auch mit dem Wurzelortskurvenverfahren ermittelt werden [32]. Damit lassen sich Vorgaben für die Überschwingweite und Einregelzeiten festlegen. Eine weitere Variante der Reglerauslegung ist die Verwendung eines Zustandsreglers. Dabei handelt es sich um ein Regelverfahren, welches nicht nur die Ausgangsgrößen mit den Eingangsgrößen vergleicht, sondern auch Zwischen- bzw. Zustandsgrößen des Systems nutzt, um entsprechende Stellgrößen zu bestimmen. Voraussetzung für die Entwicklung eines Zustandsregler ist die Bestimmung eines Zustandsraummodells. Weiterhin ist zu beachten, dass ein Zustandsregler gegenüber eines PID-Reglers einen höheren Rechenaufwand besitzt. Es wird demnach entsprechende Hardware benötigt, um die Stellgrößen berechnen zu können. In Kapitel 3 wird noch genauer

auf die Auslegung eines Zustandsreglers für ein MAV eingegangen. Dabei wird ausführlich die Ermittlung benötigter Parameter sowie die Regelgüte untersucht. Ein Vergleich zwischen verschiedenen Regelalgorithmen ist in mehreren Arbeiten zu finden. So wird z.B. in [29] und [33] die Regelgüte eines PID-Reglers gegenüber eines Zustandsreglers verglichen und die Ergebnisse mithilfe von Matlab/Simulink dargestellt. Die Ermittlung eines mathematischen Modells für einen Quadrocopter wird in [34], [5] und [20] erläutert. Dabei wird zunächst das Kräftegleichgewicht des Flugsystems aufgestellt. Anschließend resultieren daraus sechs Differentialgleichungen, welche die drei Rotationen sowie die drei Translationen des Quadrocopters beschreiben. Stehen Flugsysteme zur Verfügung, welche eine andere Anzahl an Rotoren besitzen, können die ermittelten Gleichungen für den entsprechenden Fall angepasst werden. Carlos Arellano-Muro et al. beschreibt in einem Artikel, wie ein Zustandsraummodell für einen Hexacopter entwickelt wird [1].

2.5.2 Positionsregelung

Im Folgenden werden einige Arbeiten kurz vorgestellt, welche sich mit dem Thema der Positions- bzw. Trajektorienregelung für MAVs beschäftigen.

Die korrekte Funktionsweise sowie Genauigkeit eines Positionsreglers hängt stark von dem verwendeten Lokalisationsverfahren ab. So kann im Außenbereich z.B. GPS genutzt werden, wohingegen für den Einsatz innerhalb eines Gebäudes SLAM-Verfahren eine gute Lösung sind.

Lokalisation mithilfe von Laufzeitmessungen

Tim Puls stellt in seiner Dissertation eine Methode zur Lokalisation- und Regelung für einen 4-Rotor-Helikopter vor [23]. Als Lokalisationsverfahren wird eine Kombination aus GPS, Funk und Ultraschall genutzt. In Abbildung 2.5 ist die Landung eines Flugsystems schematisch dargestellt. Ist das MAV weiter von dem Landepunkt entfernt, wird zur Lokalisation GPS genutzt, da in diesem Bereich noch nicht so eine hohe Genauigkeit benötigt wird. Kommt das Flugsystem dem Ziel näher, findet die Lokalisation mithilfe von einer Laufzeitmessung über Funk statt. Die Genauigkeit dieses Verfahrens ist abhängig von der Umgebung und kann leicht variieren. Die präziseste Positionsbestimmung wird hier mithilfe von Ultraschall erreicht. Um die Verfahren mit Funk oder Ultraschall nutzen zu können, sind allerdings entsprechende Sensoren erforderlich. Diese müssen sich im Landegebiet sowie an dem Flugsystem selbst befinden. Daher ist es mit diesem Verfahren nicht möglich, eine präzise Landung an einer beliebigen Stelle durchzuführen.

Autonome Navigation in Innenräumen mit Laserscanner und Kamera

Eine Anwendung für eine autonome Navigation mit einem MAV wird von Shaojie et. al. vorgestellt. Die Anwendung soll dabei in Innenräumen genutzt werden. Der Fokus liegt dabei auf der Kartierung und Lokalisation. Um eine vollständige Autonomie des Flugsystems gewährleisten zu können, werden alle erforderlichen Berechnungen direkt auf dem MAV durchgeführt [30]. Die Sensorik umfasst neben der integrierten IMU einen Laserscanner und

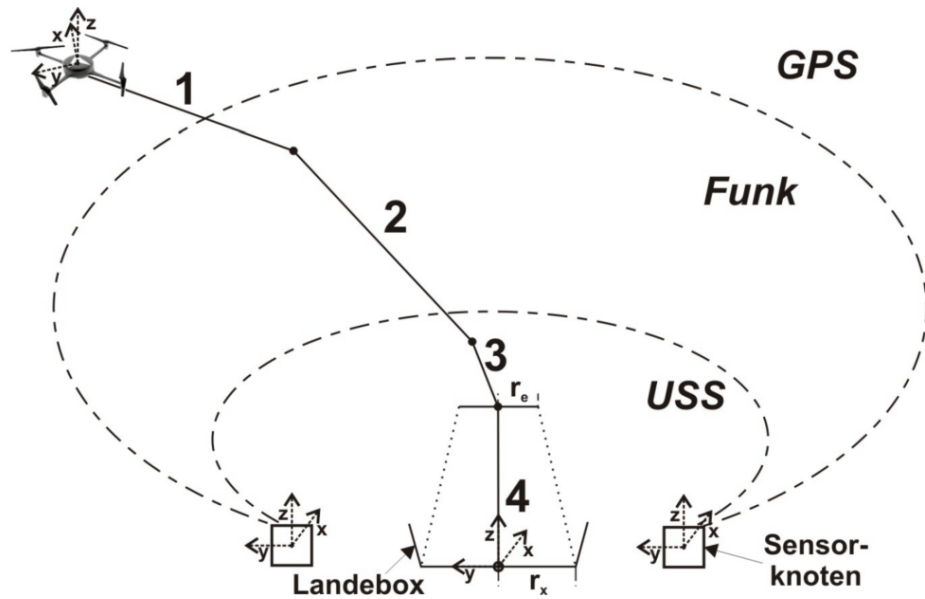


Abbildung 2.5: Schematische Darstellung einer Landung mithilfe verschiedener Lokalisationsverfahren [23]

eine Kamera. Der Scanner ist dabei so montiert, dass die horizontale Ebene erfasst wird. Um zusätzlich noch den Abstand zum Boden und der Decke ermitteln zu können, sind zwei Spiegel so angebracht, dass einige Strahlen des Scanners nach unten bzw. oben abgeleitet werden. Zur Positionsbestimmung wird ein ICP (Iterative Closest Point) Algorithmus verwendet [28].

Autonome Navigation in einem Lagerhaus

In einem Artikel von 2017 stellt Beuel et. al. ein Verfahren zur autonomen Navigation in einem Lagerhaus vor. Dazu wird ein MAV genutzt, welches mit diverser Sensorik ausgestattet ist. Darunter ist ein 3D Laserscanner, drei Stereo Kamerapaare, ein RFID-Lesegerät und ein leistungsfähiger Computer [4]. Bei den Kameras handelt es sich um sechs omnidirektionale Sensoren. Diese werden für die visuelle Odometrie genutzt. Dabei können jeweils zwei Kameras parallel montiert werden, um drei Stereo-Kameras zu erhalten oder jeweils radial vom Zentrum des Flugsystems. Durch die radiale Anordnung ergibt sich ein erhöhter Sichtbereich. Bei dem 3D Laserscanner handelt es sich um zwei einzelne 2D Scanner, welche an einer Rotationsvorrichtung angebracht sind. Es wird das ROS Framework genutzt, um Software zu erstellen und Sensorik anzusteuern. So wird z.B. für die visuelle Odometrie das ROS Packet *viso2* genutzt, welche die *LIBVISO2* Bibliothek verwendet [10].

3 Positionsregelung

In diesem Abschnitt wird eine Positionsregelung in Form eines Zustandsreglers, für ein Mikroflugsystem entwickelt. Dabei ist gewünscht, dass Translationen sowie die Rotation um die z-Achse vorgegeben werden können und das MAV diese vorgegebenen Punkte automatisch ansteuert. Zustandsregler bieten die Möglichkeit, zur Berechnung der Regelgrößen mehrere Zustandsgrößen zu nutzen. Durch diese Eigenschaft kann diese Form des Reglers besonders gut für Flugsysteme genutzt werden, da bei einer Regelung der einzelnen Rotoren mehrere Zustände des Systems berücksichtigt werden müssen, um eine stabile Fluglage zu gewährleisten. Im Folgenden wird genauer auf die genutzten Zustandsgrößen sowie die Bestimmung geeigneter Regelparameter eingegangen.

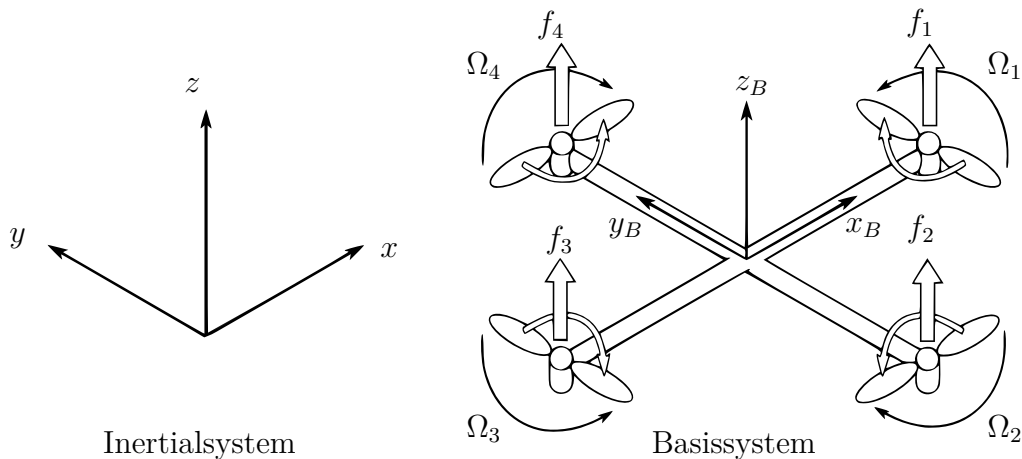


Abbildung 3.1: Schematische Darstellung eines Quadrocopters

3.1 Zustandsraumdarstellung

Um einen Zustandsregler gut realisieren zu können, ist es notwendig, ein Zustandsraummodell des zu regelnden System zu ermitteln. Dazu wird das MAV mithilfe von Differentialgleichungen beschrieben. Zunächst wird der in Abbildung 3.1 dargestellte Quadrocopter genauer untersucht. Das Flugsystem ist dabei in der sogenannten *Plus-Anordnung* aufgebaut, d.h. es liegen sich zwei Propeller gegenüber und die Achsen des Flugsystems liegen genau auf den Koordinatenachsen.

3.1.1 Mathematisches Darstellung eines Quadrocopters

Zur Ermittlung des mathematischen Modells eines Quadrocopters wird untersucht, welche Kräfte auf das Flugsystem wirken. Dabei sind u.a. äußere Einflüsse wie die Schwerkraft sowie Kräfte zu beachten, welche sich durch die Rotation der einzelnen Rotoren ergeben.

Translation

Wird dabei zunächst die translatorischen Bewegungen des Flugsystems betrachtet, ergibt sich der in Gleichung 3.1 dargestellte Zusammenhang.

$$m\dot{\vec{V}}_B = \mathbf{R}^T \vec{G} + \vec{T}_B \quad (3.1)$$

Die einzelnen Faktoren haben folgende Bedeutung:

- $m\dot{\vec{V}}_B$ Lineare Beschleunigung
- $\mathbf{R}^T \vec{G}$ Gravitation
- \vec{T}_B Schub der Rotoren

In translatorischer Richtung sind also drei Kräfte genauer zu betrachten, welche auf das MAV wirken. Die erste Kraft ist durch die lineare Beschleunigung vorgegeben, welche sich aus der Masse m sowie der Beschleunigung des Systems in x , y und z Richtung ergibt. Die zweite Kraft ergibt sich aus der Gravitation, welche auf das System in linearer Richtung wirkt. Diese wird mit dem Vektor \vec{G} beschrieben. Der in Gleichung 3.1 dargestellte Zusammenhang beschreibt die wirkenden Kräfte auf das Systems im lokalen Koordinatensystem des MAVs, daher wird die Gravitation mit der inversen Rotationsmatrix \mathbf{R}^T verrechnet [20]. Die Rotationsmatrix stellt dabei den Zusammenhang zwischen dem Inertialsystems und dem lokalen System dar und ist wie folgt definiert:

$$\mathbf{R} = \begin{pmatrix} C_\psi C_\theta & C_\psi S_\theta S_\phi - S_\psi C_\phi & C_\psi S_\theta C_\phi + S_\psi S_\phi \\ S_\psi C_\theta & S_\psi S_\theta S_\phi + C_\psi C_\phi & S_\psi S_\theta C_\phi - C_\psi S_\phi \\ -S_\theta & C_\theta S_\phi & C_\theta C_\phi \end{pmatrix} \quad (3.2)$$

Für eine übersichtlichere Darstellung der Rotationsmatrix \mathbf{R} gilt $S_x = \sin(x)$ und $C_x = \cos(x)$. Die griechischen Buchstaben ϕ , θ und ψ beschreiben die Rotation um die x , y und z -Achse.

Die letzte Kraft ergibt sich aus dem linearen Schub der einzelnen Rotoren und wird mit \vec{T}_b bezeichnet. Wird nur die Translation des Flugsystems betrachtet, entsteht durch die Rotoren zunächst nur eine Kraft in Richtung der z -Achse, da eine Bewegung in eine andere Richtung zusätzlich eine Rotation des Systems um eine Achse zu Folge hat. Der durch die

einzelnen Rotoren resultierende Schub ergibt sich aus der Summe der Rotationsenergien. Allgemein ist die Rotationsenergie wie in Gleichung 3.3 definiert.

$$E_{rot} = \frac{1}{2} \cdot J_x \cdot \Omega^2 \quad (3.3)$$

J_x beschreibt das Trägheitsmoment und Ω die Winkelgeschwindigkeit des Körpers. Bezieht man diesen Zusammenhang auf die vier Rotoren eines Quadrocopters resultiert daraus die folgende Darstellung:

$$\vec{T}_b = \begin{bmatrix} 0 \\ u_1 \end{bmatrix}$$

$$u_1 = \sum_{i=1}^4 f_i = b \sum_{i=1}^4 \Omega_i^2 \quad (3.4)$$

Der Schub in z Richtung wird hier mit u_1 definiert. Die Drehgeschwindigkeiten der einzelnen Rotoren werden mit Ω_i angegeben und b beschreibt den sogenannten Schubfaktor, welcher ein Maß für die Kraft der einzelnen Rotoren darstellt.

Nun kann die in 3.1 beschriebene Gleichung in Bezug auf das Inertialsystem beschrieben werden. Dafür wird ein neuer Vektor $\vec{\xi}$ definiert, welcher die translatorischen Koordinaten des Flugsystems im Inertialsystem beschreibt. Durch Transformation der Gleichung 3.1 mithilfe der Rotationsmatrix \mathbf{R} ergeben sich die in 3.5 dargestellten Differentialgleichungen (DGLs).

$$m\ddot{\vec{\xi}} = \vec{G} + \mathbf{R}\vec{T}_b$$

$$\begin{bmatrix} \ddot{x} \\ \ddot{y} \\ \ddot{z} \end{bmatrix} = -g \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} + \frac{u_1}{m} \begin{bmatrix} C_\psi S_\theta C_\phi + S_\psi S_\phi \\ S_\psi S_\theta C_\phi - C_\psi S_\phi \\ C_\theta C_\phi \end{bmatrix} \quad (3.5)$$

Rotation

Nachdem der Quadrocopter durch die drei DGLs für translatorische Bewegungen vollständig beschrieben ist, wird nun die Rotation des Systems genauer betrachtet. Dafür wird auch hier zunächst geprüft, welche Kräfte auf das Flugsystem wirken und ein Rotationsmoment erzeugen. Das Verhalten wird dabei durch den folgenden Zusammenhang beschrieben:

$$\mathbf{I}\dot{\vec{v}} + \vec{v} \times (\mathbf{I}\vec{v}) + \Gamma = \vec{\tau} \quad (3.6)$$

Die einzelnen Faktoren haben dabei folgende Bedeutung:

- $\mathbf{I}\dot{\vec{v}}$ Angulare Beschleunigung
- $\vec{v} \times (\mathbf{I}\vec{v})$ Zentripetalkraft
- Γ Gyroskopische Kräfte
- $\vec{\tau}$ Externe Momente durch die Rotoren

Die angulare Beschleunigung setzt sich aus den Rotationsbeschleunigungen $\dot{\vec{v}}$ sowie den Trägheitsmomenten der einzelnen Achsen zusammen. Die Trägheit der Rotation des Systems wird mithilfe der Trägheitsmatrix \mathbf{I} beschrieben, welche wie in 3.7 definiert ist [20].

$$\mathbf{I} = \begin{pmatrix} I_x & 0 & 0 \\ 0 & I_y & 0 \\ 0 & 0 & I_z \end{pmatrix} \quad (3.7)$$

Die Trägheitsmatrix hat Diagonalstruktur. Daraus lässt sich erkennen, dass die einzelnen Trägheiten nur in Richtung der einzelnen Achsen wirken und sich nicht gegenseitig beeinflussen, d.h. bei einer Rotation um die x -Achse wirkt nur die Trägheit I_x und keine andere. Rotiert das MAV um eine Achse, nimmt auch die Zentripetalkraft Einfluss auf das System. Diese Kraft ist durch $\vec{v} \times (\mathbf{I}\vec{v})$ definiert. Zusätzlich werden auch noch die Gyroskopischen Kräfte berücksichtigt, die durch die einzelnen Rotoren entstehen. Diese werden als Γ bezeichnet und setzen sich aus den Trägheitsmomenten J_r der Rotoren sowie den Rotationsgeschwindigkeiten \vec{v} zusammen. Daraus ergibt sich die in 3.8 dargestellte Gleichung.

$$\Gamma = J_r \cdot \vec{v} \times \Omega_\tau = J_r \begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} \times \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \Omega \quad (3.8)$$

Die letzte Kraft, welche Einfluss auf das Rotationsverhalten des MAVs hat, ergibt sich aus dem erzeugten Schub der einzelnen Rotoren. Dieser wird hier allgemein mit $\vec{\tau}$ bezeichnet. Um zu ermitteln, wie der Schub der einzelnen Propeller die Rotation des Systems beeinflusst, wird der Aufbau des Quadropters, wie in Abbildung 3.1 dargestellt, betrachtet. Daraus lässt sich erkennen, dass eine Rotation in positiver Richtung um die x -Achse erreicht wird, wenn der vierte Rotor schneller dreht als der zweite. Bei Betrachtung der y -Achse muss entsprechend der dritte Rotor schneller drehen als der erste. Zusätzlich ist bei einer Drehung um die x - oder y -Achse auch der Hebelarm zu betrachten, welcher Einfluss auf das Drehmoment durch die Rotoren hat. Die Länge des Hebelarms wird mit l bezeichnet und gibt die Entfernung vom Mittelpunkt des Flugsystems zum Propeller an. Bei einem Quadrocopter rotieren zwei Propeller im Uhrzeigersinn und zwei gegen den Uhrzeigersinn. Lässt man die links- und rechtsdrehenden Propeller mit unterschiedlicher Drehzahl rotieren, ergibt sich ein Moment, welches ermöglicht, das Flugsystem um die z -Achse zu drehen. Darus folgt für $\vec{\tau}$ der folgende Zusammenhang:

$$\begin{aligned}
 \vec{\tau} &= \begin{bmatrix} u_2 \\ u_3 \\ u_4 \end{bmatrix} \cdot l \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix} \\
 u_2 &= b(\Omega_4^2 - \Omega_2^2) \\
 u_3 &= b(\Omega_3^2 - \Omega_1^2) \\
 u_4 &= d(\Omega_2^2 + \Omega_4^2 - \Omega_1^2 - \Omega_3^2)
 \end{aligned} \tag{3.9}$$

Der Faktor b steht auch hier für den Schubfaktor, der ein Maß für die Schubkraft der einzelnen Rotoren ist. Der Faktor d gibt an, welches Moment die Rotoren bei einer Drehung um die z -Achse, erzeugen. Wird nun wieder der in Gleichung 3.6 beschriebene Zusammenhang betrachtet und die entsprechenden Werte eingesetzt, ergeben sich die folgenden DGLs, welche die Rotation des Flugsystems beschreiben:

$$\begin{aligned}
 \dot{\vec{v}} &= \mathbf{I}^{-1} \left(- \begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} \times \begin{bmatrix} I_x \cdot \dot{\phi} \\ I_y \cdot \dot{\theta} \\ I_z \cdot \dot{\psi} \end{bmatrix} - J_r \begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} \times \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \Omega + \vec{\tau} \right) \\
 \begin{bmatrix} \ddot{\phi} \\ \ddot{\theta} \\ \ddot{\psi} \end{bmatrix} &= \begin{bmatrix} \dot{\theta}\dot{\psi}(I_y - I_z)/I_x \\ \dot{\phi}\dot{\psi}(I_z - I_x)/I_y \\ \dot{\phi}\dot{\theta}(I_x - I_y)/I_z \end{bmatrix} - J_r \begin{bmatrix} \dot{\theta}/I_x \\ -\dot{\phi}/I_y \\ 0 \end{bmatrix} \Omega + \begin{bmatrix} u_2 \cdot l/I_x \\ u_3 \cdot l/I_y \\ u_4 \cdot 1/I_z \end{bmatrix}
 \end{aligned} \tag{3.10}$$

Gesamtbetrachtung

Zur Übersicht sind im Folgenden nochmals alle sechs Differentialgleichungen aufgeführt, welche das Verhalten des Flugsystem für die Translation und Rotation beschreiben [5]. Die entsprechenden Bedeutungen der einzelnen Symbole sind in Tabelle 3.1 dargestellt.

Symbol	Definition
Φ	Rotationswinkel x-Achse (roll)
Θ	Rotationswinkel y-Achse (pitch)
Ψ	Rotationswinkel z-Achse (yaw)
$I_{x,y,z}$	Trägheitsmoment Körper
J_r	Trägheitsmoment Rotor
Ω	Rotorgeschwindigkeit
l	Entfernung zwischen Rotor und Mittelpunkt
b	Schubfaktor
d	Zugfaktor

Tabelle 3.1: Symboldefinitionen für Differentialgleichungen eines Quadrocopters

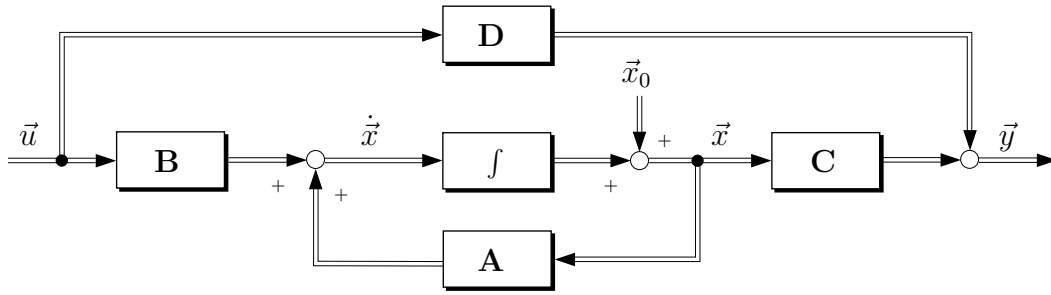
$$\begin{aligned}
 \ddot{x} &= (\cos\Phi \cdot \sin\Theta \cdot \cos\Psi + \sin\Phi \cdot \sin\Psi) \frac{1}{m} \cdot u_1 & u_1 &= b(\Omega_1^2 + \Omega_2^2 + \Omega_3^2 + \Omega_4^2) \\
 \ddot{y} &= (\cos\Phi \cdot \sin\Theta \cdot \sin\Psi - \sin\Phi \cdot \cos\Psi) \frac{1}{m} \cdot u_1 & u_2 &= b(\Omega_4^2 - \Omega_2^2) \\
 \ddot{z} &= -g + (\cos\Phi \cdot \cos\Theta) \frac{1}{m} \cdot u_1 & u_3 &= b(\Omega_3^2 - \Omega_1^2) \\
 & & u_4 &= d(\Omega_2^2 + \Omega_4^2 - \Omega_1^2 - \Omega_3^2) \\
 \ddot{\Phi} &= \dot{\Theta} \dot{\Psi} \left(\frac{I_y - I_z}{I_x} \right) - \frac{J_r}{I_x} \cdot \dot{\Theta} \Omega + \frac{l}{I_x} \cdot u_2 \\
 \ddot{\Theta} &= \dot{\Phi} \dot{\Psi} \left(\frac{I_z - I_x}{I_y} \right) + \frac{J_r}{I_y} \cdot \dot{\Phi} \Omega + \frac{l}{I_y} \cdot u_3 \\
 \ddot{\Psi} &= \dot{\Phi} \dot{\Theta} \left(\frac{I_x - I_y}{I_z} \right) + \frac{1}{I_z} \cdot u_4
 \end{aligned}$$

Durch genauere Betrachtung der resultierenden DGLs können erste Aussagen über das Verhalten des Flugsystems getroffen werden. So ist anhand der ersten zwei Gleichungen zu erkennen, dass sich das Flugsystem nur dann in eine Richtung bewegt, wenn der Winkel ϕ oder θ ungleich null ist. Zusätzlich wird das Verhalten maßgeblich von den Eingangsgrößen u_i beeinflusst. Diese Eingangsgrößen setzen sich dabei aus den Kräften der einzelnen Propeller zusammen. Dadurch, dass die Rotordrehzahlen nicht direkt als Eingangsgröße für das System definiert worden sind, können die ermittelten DGLs flexibler für verschiedene MAV-Typen genutzt werden. Steht z.B. ein Flugsystem mit mehr als vier Propellern zur Verfügung, müssen nur die Gleichungen für die Eingangsgrößen u_i angepasst werden. Die ermittelten DGLs ändern sich allerdings nicht. Auf diesen Aspekt wird noch genauer in Abschnitt 3.2.4 eingegangen.

3.1.2 Zustandsraummodell

Mit den in Abschnitt 3.1.1 ermittelten Differentialgleichungen kann nun ein Zustandsraummodell für ein MAV entwickelt werden, welches für die Entwicklung eines Zustandsreglers Voraussetzung ist. In Abbildung 3.2 ist die allgemeine Form eines solchen Modells als Signalflussplan dargestellt. Es handelt sich dabei um ein so genanntes *Multiple Input, Multiple Output System* oder auch MiMo-System. Damit wird also ein System beschrieben, welches von mehreren Eingangsgrößen abhängig ist und zusätzlich mehr als eine Ausgangsgröße aufweist.

Das Verhalten des Systems wird durch die Matrizen **A**, **B**, **C** und **D** beschrieben. Die entsprechenden Bedeutungen der einzelnen Parameter ist in Tabelle 3.2 dargestellt. Die Dynamikmatrix **A** beschreibt das allgemeine Systemverhalten, wenn keine Einflüsse von außen wirken. Die Eingangsmatrix **B** gibt an, wie Stellgrößen das Systemverhalten beeinflussen. Die Ausgangsmatrix **C** stellt den Zusammenhang zwischen den vorhandenen Zustandsgrößen



$$\dot{\vec{x}} = \mathbf{A}^{(n,n)} \cdot \vec{x} + \mathbf{B}^{(n,p)} \cdot \vec{u}$$

$$\vec{y} = \mathbf{C}^{(q,n)} \cdot \vec{x} + \mathbf{D}^{(q,p)} \cdot \vec{u}$$

Abbildung 3.2: Allgemeine Darstellung eines Zustandsraummodells (MiMo-System)

und dem Ausgang des System dar. Für manche Systeme muss auch eine Durchgangsmatrix **D** berücksichtigt werden. Diese gibt an, ob und in welcher Form Eingangsgrößen direkt auf den Ausgang wirken. Für die Entwicklung eines Zustandsraummodells für ein MAV muss allerdings keine Durchgangsmatrix berücksichtigt werden, da das hier betrachtete Flugsystem eine gewisse Trägheit aufweist und somit Eingangsgrößen, welche auf das System geschaltet werden, nicht ohne Beeinflussung direkt auf den Ausgang wirken [18].

Symbol	Definition
\vec{u}	Eingangsgrößen
\vec{y}	Ausgangsgrößen
\vec{x}_0	Initialwerte des Systems
\vec{x}	Zustandsgrößen
n	Anzahl Zustandsgrößen
p	Anzahl Eingangsgrößen
q	Anzahl Ausgangsgrößen
A	Dynamikmatrix (n,n)
B	Eingangsmatrix (n,p)
C	Ausgangsmatrix (q,n)
D	Durchgangsmatrix (q,p)

Tabelle 3.2: Bedeutung einzelner Parameter eines Zustandsraummodells

Definition geeigneter Zustandsgrößen

Um für ein spezielles System ein Zustandsraummodell entwickelt zu können, müssen zunächst Zustandsgrößen definiert werden. Zustandsgrößen sind in einem realen System normalerweise Messwerte, welche durch geeignete Sensorik zur Verfügung gestellt werden. Bei einem MAV sind das z.B. Neigungsinformationen von der IMU. Für die in Abschnitt 3.1.1 hergeleiteten DGLs werden die Zustandsgrößen \vec{x} wie folgt definiert:

$$\begin{aligned}
 x_1 &= x & x_7 &= \Phi \\
 x_2 &= \dot{x}_1 = \dot{x} & x_8 &= \dot{x}_7 = \dot{\Phi} \\
 x_3 &= y & x_9 &= \Theta \\
 x_4 &= \dot{x}_3 = \dot{y} & x_{10} &= \dot{x}_9 = \dot{\Theta} \\
 x_5 &= z & x_{11} &= \Psi \\
 x_6 &= \dot{x}_5 = \dot{z} & x_{12} &= \dot{x}_{11} = \dot{\Psi}
 \end{aligned} \tag{3.11}$$

Setzt man nun die einzelnen Zustandsgrößen in die zuvor ermittelten DGLs ein und ermittelt die erste Ableitung der einzelnen Zustandsgrößen, erhält man die folgenden Zustandsgleichungen.

$$\begin{aligned}
 \dot{x}_1 &= x_2 \\
 \dot{x}_2 &= (\cos x_7 \cdot \sin x_9 \cdot \cos x_{11} + \sin x_7 \cdot \sin x_{11}) \frac{1}{m} \cdot u_1 \\
 \dot{x}_3 &= x_4 \\
 \dot{x}_4 &= (\cos x_7 \cdot \sin x_9 \cdot \sin x_{11} - \sin x_7 \cdot \cos x_{11}) \frac{1}{m} \cdot u_1 \\
 \dot{x}_5 &= x_6 \\
 \dot{x}_6 &= -g + (\cos x_7 \cdot \cos x_9) \frac{1}{m} \cdot U_1 \\
 \dot{x}_7 &= x_8 \\
 \dot{x}_8 &= x_{12} x_{10} \left(\frac{I_y - I_z}{I_x} \right) - \frac{J_r}{I_x} \cdot x_{10} \Omega + \frac{l}{I_x} \cdot u_2 \\
 \dot{x}_9 &= x_{10} \\
 \dot{x}_{10} &= x_{12} x_8 \left(\frac{I_z - I_x}{I_y} \right) + \frac{J_r}{I_y} \cdot x_8 \Omega + \frac{l}{I_y} \cdot u_3 \\
 \dot{x}_{11} &= x_{12} \\
 \dot{x}_{12} &= x_{10} x_8 \left(\frac{I_x - I_y}{I_z} \right) + \frac{1}{I_z} \cdot u_4
 \end{aligned} \tag{3.12}$$

Durch diesen Schritt wird erreicht, dass die DGLs zweiter Ordnung in DGLs erster Ordnung überführt werden, was eine Voraussetzung für die Entwicklung des Zustandsraummodells ist.

Linearisierung der Zustandsgleichungen

Bei dem in Abbildung 3.2 dargestellten Zustandsraummodell handelt es sich um ein lineares System. Die in 3.12 dargestellten Zustandsgleichungen sind allerdings nicht linear und müssen somit zunächst linearisiert werden. Wird ein nichtlineares System linearisiert, findet

dieser Vorgang immer um einen bestimmten Arbeitspunkt statt. Dieser Arbeitspunkt wird normalerweise durch die Ruhelage des betrachteten Systems bestimmt. Ein System ist immer dann in Ruhelage, wenn alle Geschwindigkeiten und Beschleunigungen den Wert null annehmen. Stellt man sich ein Flugsystem vor, so tritt die Ruhelage dann ein, wenn das MAV in der Luft schwebt und sich in keine Richtung bewegt. Daraus folgt, dass alle Zustandsgrößen gleich null gesetzt werden können, außer die drei Translationen x_1 , x_2 und x_3 , da ein Flugsystem auch an einer anderen x , y oder z -Koordinate bewegungslos in der Luft stehen kann. Werden jetzt allerdings erneut die in 3.12 beschriebenen Gleichungen betrachtet, so ist zu erkennen, dass die Zustandsgrößen x_1 , x_2 und x_3 keinen Einfluss auf die Gleichungen haben. Somit können diese Werte ebenfalls gleich null gesetzt werden und es gilt bei Ruhelage des Systems für alle Zustandsgrößen $\vec{x}_0 = \vec{0}$. Werden die Eingangsgrößen \vec{u} betrachtet, darf nur die erste Größe einen Wert ungleich null annehmen, da diese den Gesamtschub der Rotoren beschreibt. Damit das Flugsystem die Höhe nicht ändert, muss der erzeugte Schub genau so groß sein wie die Kraft welche in z -Richtung auf das Flugsystem wirkt. Daraus folgt $u_{1_0} = m \cdot g$, wobei m für die Masse des Flugsystems steht und g für die Erdanziehungskraft. Die Ruhelage des System ist somit wie folgt definiert:

$$\vec{x}_0 = \vec{0} \quad \vec{u}_0 = \begin{pmatrix} m \cdot g \\ 0 \\ 0 \\ 0 \end{pmatrix} \quad (3.13)$$

Ist die Ruhelage des Systems bekannt, können die zuvor hergeleiteten Zustandsgleichungen nun linearisiert werden, um ein Zustandsraummodell der Form $\dot{\vec{x}} = \mathbf{A} \cdot \vec{x} + \mathbf{B} \cdot \vec{u}$, zu erhalten. Dazu wird jede in 3.12 definierte Gleichung nach jeder einzelnen Zustandsgröße x und nach jeder einzelnen Eingangsgröße u differenziert und anschließend die Ruhelage des Systems eingesetzt [18]. Dieser Vorgang wird mithilfe der *Jacobi-Matrix* beschrieben. Das linearisierte Zustandsraummodell wird somit durch den folgenden Ausdruck definiert:

$$\begin{aligned}\Delta \dot{\vec{x}} &= \mathbf{A} \cdot \Delta \vec{x} + \mathbf{B} \cdot \Delta \vec{u} \\ \Delta \vec{y} &= \mathbf{C} \cdot \Delta \vec{x} + \mathbf{D} \cdot \Delta \vec{u}\end{aligned}$$

$$\begin{aligned}\Delta \dot{\vec{x}} &= \begin{pmatrix} \left. \frac{\partial f_1}{\partial x_1} \right|_{\substack{\vec{x}=\vec{x}_0 \\ \vec{u}=\vec{u}_0}} & \left. \frac{\partial f_1}{\partial x_2} \right|_{\substack{\vec{x}=\vec{x}_0 \\ \vec{u}=\vec{u}_0}} & \dots & \left. \frac{\partial f_1}{\partial x_n} \right|_{\substack{\vec{x}=\vec{x}_0 \\ \vec{u}=\vec{u}_0}} \\ \left. \frac{\partial f_2}{\partial x_1} \right|_{\substack{\vec{x}=\vec{x}_0 \\ \vec{u}=\vec{u}_0}} & \left. \frac{\partial f_2}{\partial x_2} \right|_{\substack{\vec{x}=\vec{x}_0 \\ \vec{u}=\vec{u}_0}} & \dots & \left. \frac{\partial f_2}{\partial x_n} \right|_{\substack{\vec{x}=\vec{x}_0 \\ \vec{u}=\vec{u}_0}} \\ \vdots & \vdots & \ddots & \vdots \\ \left. \frac{\partial f_n}{\partial x_1} \right|_{\substack{\vec{x}=\vec{x}_0 \\ \vec{u}=\vec{u}_0}} & \left. \frac{\partial f_n}{\partial x_2} \right|_{\substack{\vec{x}=\vec{x}_0 \\ \vec{u}=\vec{u}_0}} & \dots & \left. \frac{\partial f_n}{\partial x_n} \right|_{\substack{\vec{x}=\vec{x}_0 \\ \vec{u}=\vec{u}_0}} \end{pmatrix} \cdot \Delta \vec{x} + \begin{pmatrix} \left. \frac{\partial f_1}{\partial u_1} \right|_{\substack{\vec{x}=\vec{x}_0 \\ \vec{u}=\vec{u}_0}} & \left. \frac{\partial f_1}{\partial u_2} \right|_{\substack{\vec{x}=\vec{x}_0 \\ \vec{u}=\vec{u}_0}} & \dots & \left. \frac{\partial f_1}{\partial u_p} \right|_{\substack{\vec{x}=\vec{x}_0 \\ \vec{u}=\vec{u}_0}} \\ \left. \frac{\partial f_2}{\partial u_1} \right|_{\substack{\vec{x}=\vec{x}_0 \\ \vec{u}=\vec{u}_0}} & \left. \frac{\partial f_2}{\partial u_2} \right|_{\substack{\vec{x}=\vec{x}_0 \\ \vec{u}=\vec{u}_0}} & \dots & \left. \frac{\partial f_2}{\partial u_p} \right|_{\substack{\vec{x}=\vec{x}_0 \\ \vec{u}=\vec{u}_0}} \\ \vdots & \vdots & \ddots & \vdots \\ \left. \frac{\partial f_n}{\partial u_1} \right|_{\substack{\vec{x}=\vec{x}_0 \\ \vec{u}=\vec{u}_0}} & \left. \frac{\partial f_n}{\partial u_2} \right|_{\substack{\vec{x}=\vec{x}_0 \\ \vec{u}=\vec{u}_0}} & \dots & \left. \frac{\partial f_n}{\partial u_p} \right|_{\substack{\vec{x}=\vec{x}_0 \\ \vec{u}=\vec{u}_0}} \end{pmatrix} \cdot \Delta \vec{u} \\ \Delta \vec{y} &= \begin{pmatrix} \left. \frac{\partial g_1}{\partial x_1} \right|_{\substack{\vec{x}=\vec{x}_0 \\ \vec{u}=\vec{u}_0}} & \left. \frac{\partial g_1}{\partial x_2} \right|_{\substack{\vec{x}=\vec{x}_0 \\ \vec{u}=\vec{u}_0}} & \dots & \left. \frac{\partial g_1}{\partial x_n} \right|_{\substack{\vec{x}=\vec{x}_0 \\ \vec{u}=\vec{u}_0}} \\ \left. \frac{\partial g_2}{\partial x_1} \right|_{\substack{\vec{x}=\vec{x}_0 \\ \vec{u}=\vec{u}_0}} & \left. \frac{\partial g_2}{\partial x_2} \right|_{\substack{\vec{x}=\vec{x}_0 \\ \vec{u}=\vec{u}_0}} & \dots & \left. \frac{\partial g_2}{\partial x_n} \right|_{\substack{\vec{x}=\vec{x}_0 \\ \vec{u}=\vec{u}_0}} \\ \vdots & \vdots & \ddots & \vdots \\ \left. \frac{\partial g_q}{\partial x_1} \right|_{\substack{\vec{x}=\vec{x}_0 \\ \vec{u}=\vec{u}_0}} & \left. \frac{\partial g_q}{\partial x_2} \right|_{\substack{\vec{x}=\vec{x}_0 \\ \vec{u}=\vec{u}_0}} & \dots & \left. \frac{\partial g_q}{\partial x_n} \right|_{\substack{\vec{x}=\vec{x}_0 \\ \vec{u}=\vec{u}_0}} \end{pmatrix} \cdot \Delta \vec{x} + \begin{pmatrix} \left. \frac{\partial g_1}{\partial u_1} \right|_{\substack{\vec{x}=\vec{x}_0 \\ \vec{u}=\vec{u}_0}} & \left. \frac{\partial g_1}{\partial u_2} \right|_{\substack{\vec{x}=\vec{x}_0 \\ \vec{u}=\vec{u}_0}} & \dots & \left. \frac{\partial g_1}{\partial u_p} \right|_{\substack{\vec{x}=\vec{x}_0 \\ \vec{u}=\vec{u}_0}} \\ \left. \frac{\partial g_2}{\partial u_1} \right|_{\substack{\vec{x}=\vec{x}_0 \\ \vec{u}=\vec{u}_0}} & \left. \frac{\partial g_2}{\partial u_2} \right|_{\substack{\vec{x}=\vec{x}_0 \\ \vec{u}=\vec{u}_0}} & \dots & \left. \frac{\partial g_2}{\partial u_p} \right|_{\substack{\vec{x}=\vec{x}_0 \\ \vec{u}=\vec{u}_0}} \\ \vdots & \vdots & \ddots & \vdots \\ \left. \frac{\partial g_q}{\partial u_1} \right|_{\substack{\vec{x}=\vec{x}_0 \\ \vec{u}=\vec{u}_0}} & \left. \frac{\partial g_q}{\partial u_2} \right|_{\substack{\vec{x}=\vec{x}_0 \\ \vec{u}=\vec{u}_0}} & \dots & \left. \frac{\partial g_q}{\partial u_p} \right|_{\substack{\vec{x}=\vec{x}_0 \\ \vec{u}=\vec{u}_0}} \end{pmatrix} \cdot \Delta \vec{u}\end{aligned}\tag{3.14}$$

Der in den Gleichungen 3.14 verwendete Δ -Operator dient hier lediglich dazu, um zu verdeutlichen, dass das so beschriebene System sich von der in 3.12 beschriebenen nichtlinearen Zustandsraumdarstellung unterscheidet. Es wird nicht mehr das Verhalten des gesamten Flugsystems beschrieben, sondern nur noch der Bereich um die Ruhelage. Je weiter sich also das Flugsystem von der in 3.13 definierten Ruhelage entfernt, umso ungenauer beschreibt das linearisierte Zustandsraummodell das Verhalten des Systems. Wird für solch ein System allerdings ein Zustandsregler entwickelt, so ist die Hauptaufgabe dieses Reglers das System nahe der Ruhelage zu halten. Dadurch ist gewährleistet, dass eine lineare Beschreibung eines Flugsystems ausreicht, um einen Zustandsregler mit einer ausreichenden Regelgüte zu realisieren. In Abschnitt 3.2 wird auf die Entwicklung eines Zustandsreglers für ein MAV näher eingegangen.

Werden für das in 3.14 dargestellte Zustandsraummodell nun Werte eingesetzt, ergibt sich die in 3.15 gezeigte Zustandsraumdarstellung.

$$\Delta \dot{\vec{x}} = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ & & 0 & 0 & 0 & 0 & 0 & 0 & g & 0 & 0 & 0 \\ & & & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ & & & & 0 & 0 & g & 0 & 0 & 0 & 0 & 0 \\ & & & & & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ & & & & & & 0 & 0 & 0 & 0 & 0 & 0 \\ & & & & & & & 1 & 0 & 0 & 0 & 0 \\ & & \underline{0} & & & & & & 0 & 0 & 0 & 0 \\ & & & & & & & & & 1 & 0 & 0 \\ & & & & & & & & & & 0 & 0 \\ & & & & & & & & & & & 1 \\ & & & & & & & & & & & & 0 \end{pmatrix} \cdot \Delta \vec{x} + \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ \frac{1}{m} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & \frac{l}{I_x} & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{l}{I_y} & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{I_z} \end{pmatrix} \cdot \Delta \vec{u}$$

$$\Delta \vec{y} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix} \cdot \Delta \vec{x} \quad (3.15)$$

Es ist zu erkennen, dass das Verhalten des linearisierten System hauptsächlich von den Trägheitsmomenten I_x , I_y und I_z , der Masse m und der Entfernung zwischen Rotor und Mittelpunkt l abhängt. Für das MAV sollen die Translationen sowie die Rotation um die z -Achse vorgegeben werden können. Dies wird durch die Matrix \mathbf{C} definiert. Es werden also die vier entsprechenden Zustandsgrößen ohne Veränderung zum Ausgang des Systems weitergeleitet. Dadurch weißt die Ausgangsmatrix \mathbf{C} lediglich an vier Stellen einen Wert von eins auf, wohingegen die restlichen Werte null sind.

Stabilität

Da nun das gesamte Zustandsraummodell bekannt ist, wird zunächst geprüft, ob das System eine gewisse Eigenstabilität aufweist. Dazu wird das Programm Matlab/Simulink genutzt, um entsprechende Simulationen durchzuführen. Für das Flugmodell werden die in Tabelle 3.3 dargestellten Parameter verwendet. Diese wurden aus einem Simulationsprogramm entnommen, welches von der ETH-Zürich für verschiedene Flugmodelle angepasst wurde [9]. Dieser Simulator trägt die Bezeichnung *rotorS Simulator* und wird im späteren Verlauf dieser Arbeit noch verwendet und genauer dargestellt. Die genutzten Parameter beschreiben einen Quadrocopter mit der Bezeichnung *Pelican*. Dabei handelt es sich um ein MAV, welches von der Firma Ascending Technologies [2] entwickelt und gebaut wurde und auch im Projekt TRADR [31] für Flugtests genutzt wird.

Das Flugmodell ist dabei symmetrisch aufgebaut und der Schwerpunkt befindet sich genau im Zentrum des Systems. Die Trägheitsmomente I_x und I_y haben demnach den gleichen

Paramter		Wert	Einheit
Trägheitsmoment x -Achse	I_x	0,01	$kg \cdot m^2$
Trägheitsmoment y -Achse	I_y	0,01	$kg \cdot m^2$
Trägheitsmoment z -Achse	I_z	0,02	$kg \cdot m^2$
Masse	m	1,0	kg
Entfernung Rotor Mittelpunkt	l	0,21	m
Schubfaktor	b	$9,50346 \cdot 10^{-6}$	$kg \cdot m$
Zugfaktor	d	$5,06428 \cdot 10^{-7}$	$kg \cdot m^2$

Tabelle 3.3: Parameter des Quadrocopters Pelican

Wert. Um das System nun auf Stabilität zu prüfen, wird zunächst der Signalflussplan, wie in Abbildung 3.2 dargestellt, in Matlab/Simulink aufgebaut und die ermittelten Matrizen eingetragen. Anschließend wird die Eingangsgröße u_1 auf den Wert 3,0, u_2 auf den Wert 2,0, u_3 auf den Wert 0,0 und u_4 auf den Wert 0,5 gesetzt. Diese Werte sind beliebig gewählt und dienen zur Untersuchung der Eigenstabilität des Flugsystems.

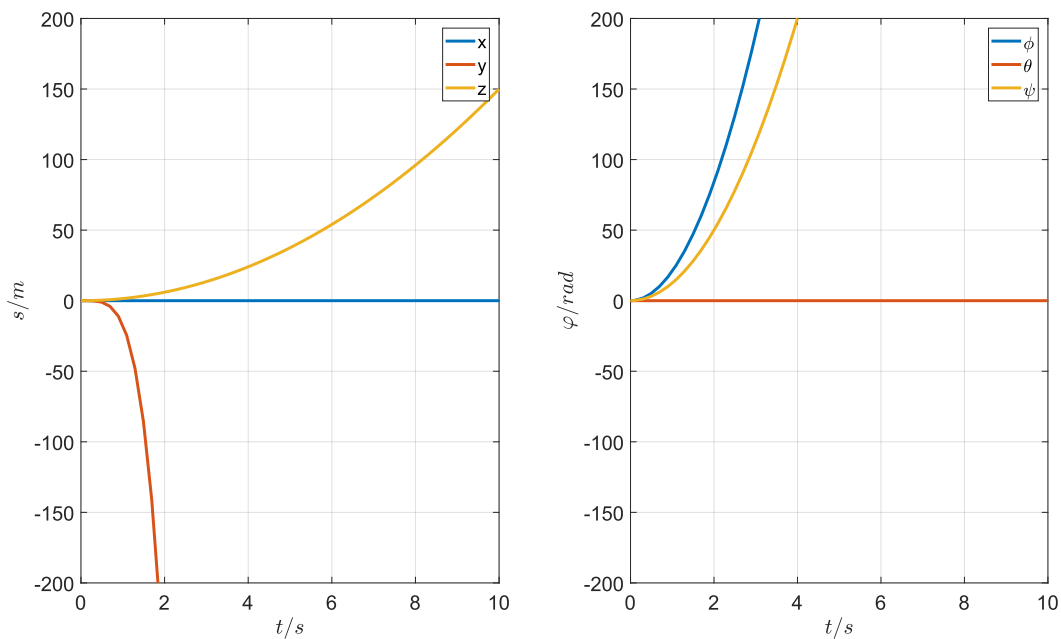


Abbildung 3.3: Darstellung der Eigenstabilität des Zustandsraummodells eines Flugsystems

In Abbildung 3.3 ist die Bewegung des MAVs nach der Zeit aufgetragen. Die linke Seite repräsentiert die translatorischen Bewegungen in x -, y - sowie z -Richtung und die rechte Seite stellt die Rotationen um die einzelnen Achsen dar. Es wird ersichtlich, dass das Flugsystem keinerlei Eigenstabilität aufweist, da kein stationärer Zustand angenommen wird. Laut der durchgeführten Simulation würde sich das Flugsystem in weniger als zwei Sekunden bereits um 200 Meter in y -Richtung bewegt haben. Dies entspricht einer Geschwindigkeit von mehr

als 360km/h . Das instabile Verhalten wird noch deutlicher bei Betrachtung der Rotation des Flugsystems. Das durch die Simulation festgestellte Verhalten kann durch das Flugsystem in der Realität nicht erreicht werden. Das System muss demnach aktiv geregelt werden.

3.2 Zustandsregelung

Um eine stabile Fluglage gewährleisten zu können, wird im Folgenden eine Zustandsregelung für das in 3.1.2 hergeleitete Zustandsraummodell entwickelt.

3.2.1 Zustandsrückführung

Zunächst werden die einzelnen Zustandsgrößen auf den Eingang des System zurückgeführt. Daraus ergibt sich der in Abbildung 3.4 dargestellte Signalflussplan. Es zeigt sich, dass alle Zustandsgrößen mit einer Rückführungsmatrix \mathbf{K} verrechnet und auf den Eingang zurückgeführt werden [19, S. 144 ff.]. Durch Einsetzen der Rückführungsmatrix in die allgemeine Zustandsgleichung $\dot{\vec{x}} = \mathbf{A} \cdot \vec{x} + \mathbf{B} \cdot \vec{u}$ ergibt sich der folgende Zusammenhang:

$$\dot{\vec{x}} = \mathbf{A} \cdot \vec{x} + \mathbf{B}(-\mathbf{K}\vec{x}) = (\mathbf{A} - \mathbf{BK})\vec{x} \quad (3.16)$$

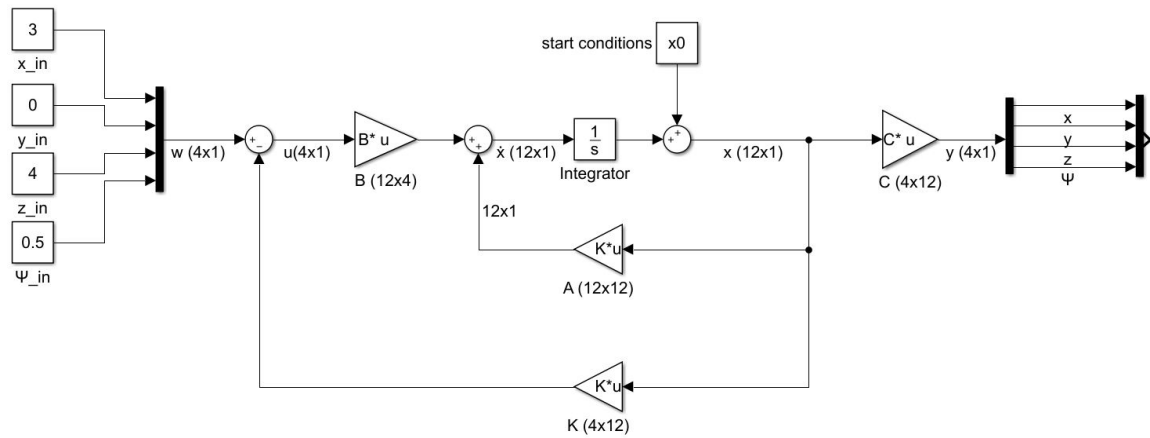


Abbildung 3.4: Signalflussplan eines Zustandsraummodells mit Zustandsrückführung

Um die entsprechenden Werte für die Matrix \mathbf{K} ermitteln zu können, werden zunächst die Eigenwerte bzw. Polstellen für das System gewählt. Diese sind maßgebend für das dynamische Verhalten des geschlossenen Regelkreises. Für eine geeignete Wahl der Eigenwerte sind einige Kriterien zu beachten. In Abbildung 3.5 ist der Zusammenhang zwischen den Eigenwerten λ und dem Verhalten eines dynamischen Systems dargestellt.

Werden nur reelle Eigenwerte gewählt, ergibt sich ein überschwingfreies Einregelverhalten des Systems, wohingegen konjugiert komplexe Eigenwerte dazu führen, dass das System zunächst überschwingt bevor der eingeschwungene Zustand erreicht wird. Weiterhin ist es

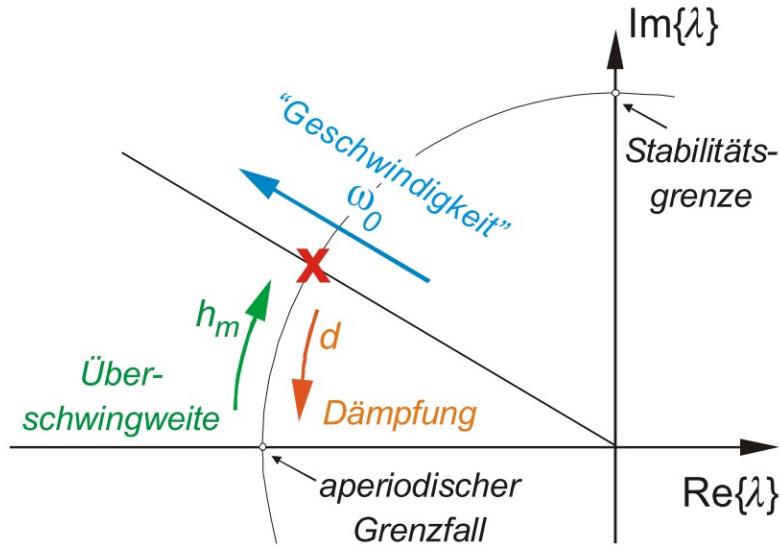


Abbildung 3.5: Zusammenhang der Eigenwerte und des Verhaltens eines dynamischen Systems

notwendig, dass alle Eigenwerte negativ gewählt werden, um ein stabiles Einregelverhalten zu gewährleisten. Zusätzlich kann durch entsprechende Platzierung der Eigenwerte die Geschwindigkeit des System beeinflusst werden. Allgemein gilt, je kleiner die Eigenwerte sind, umso schneller erreicht das System den eingeschwungenen Zustand. Dennoch sollten die Eigenwerte nicht zu klein gewählt werden, da ein reales System wie ein MAV nicht in der Lage ist, beliebig schnell einen bestimmten Punkt anzusteuern. Es muss also bei der Wahl geeigneter Eigenwerte darauf geachtet werden, welches System in der Realität geregelt werden soll. Zunächst ist es also sinnvoll, relativ kleine negative Eigenwerte zu wählen und das Ausregelverhalten des Systems anhand eine Simulation zu betrachten. Anschließend können die Eigenwerte schrittweise verkleinert werden, bis das gewünschte Ausregelverhalten des Systems erreicht wird. Für kleiner Systeme können die Rückföhrungsfaktoren bestimmt werden, indem die Eigenwerte mithilfe der Determinante des geschlossenen Regelkreises wie folgt dargestellt werden:

$$\det(\lambda \mathbf{E} - (\mathbf{A} - \mathbf{B}\mathbf{K})) \quad (3.17)$$

Durch Einsetzen der in Abschnitt 3.1.2 definierten Matrizen \mathbf{A} und \mathbf{B} sowie gewählter Eigenwerte λ , kann die entsprechende Matrix \mathbf{K} bestimmt werden [19, S. 243 ff.]. Für größere Systeme ist es sinnvoll, Computerprogramme wie Matlab zu nutzen, um die Rückföhrungsfaktoren \mathbf{K} zu ermitteln. Dazu gibt es verschiedene Funktionen wie *acker()* oder *place()*. Die Funktion *acker()* ist allerdings nur für Eingrößensysteme geeignet. Bei dem Zustandsraummodell eines MAV handelt es sich allerdings um ein Mehrgrößensystem. Daher wird zur Berechnung der Rückföhrungsmatrix \mathbf{K} die Matlab-Funktion *place()* genutzt.

Das Kommando `place()` ist in der *Control System Toolbox* von Matlab enthalten. Da das hier entwickelte Zustandsraummodell eines MAVs zwölf Zustandsgrößen aufweist, müssen dementsprechend auch zwölf Eigenwerte definiert werden, welche das dynamische Verhalten des Systems beschreiben. Diese werden zunächst beliebig gewählt und zusammen mit der Systemmatrix \mathbf{A} und der Eingangsmatrix \mathbf{B} dem Matlab-Kommando `place()` übergeben, um die entsprechende Rückführungsmatrix \mathbf{K} zu erhalten. In Abschnitt 3.2.3 wird noch genauer auf eine zweckmäßige Auswahl von genutzten Eigenwerten eingegangen. Das Kommando `place()` nutzt den in [13] beschriebenen Algorithmus. Ist die Rückführungsmatrix \mathbf{K} bekannt, kann nun eine Simulation in Matlab/Simulink durchgeführt werden. Die Eingangsgrößen wurden wie bei der Stabilitätsprüfung wieder auf die gleichen Werte gesetzt. Also $x_{in} = 3,0$, $y_{in} = 2,0$, $z_{in} = 0,0$ und $\psi_{in} = 0,5$. Die Eingangsgrößen werden nun nicht mehr mit u_1 bis u_4 bezeichnet, da nun die Werte nicht mehr direkt als Stellgröße vorgegeben werden, sondern diese vorher mit der Rückführungsmatrix \mathbf{K} verrechnet werden. Die Ergebnisse sind in Abbildung 3.6 dargestellt.

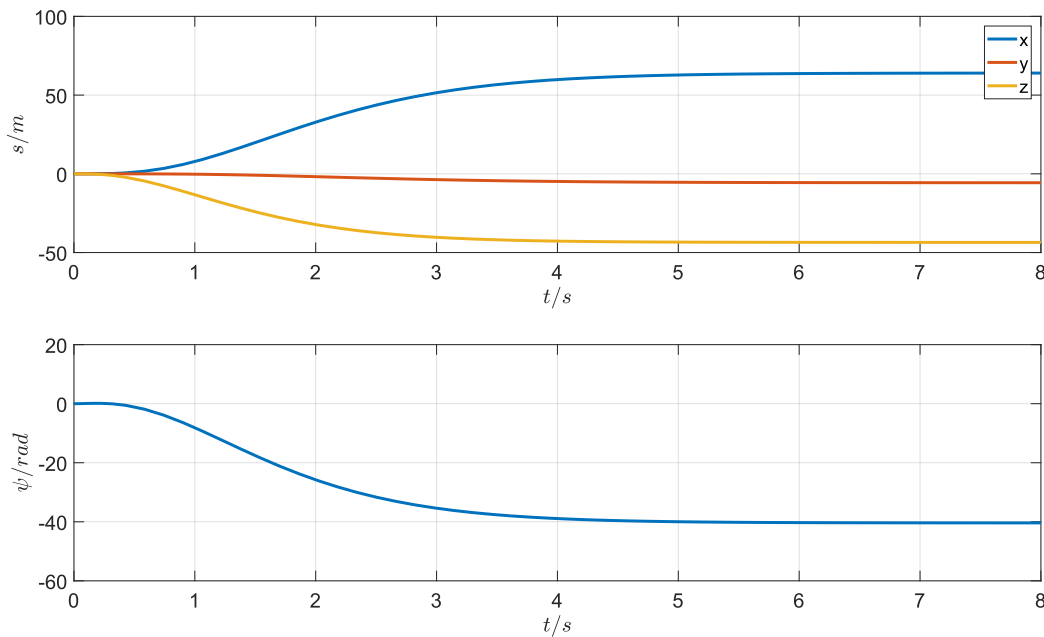


Abbildung 3.6: Einregelverhalten des Zustandsraummodells eines Flugsystems mit Zustandsrückführung

Werden zunächst die translatorischen Bewegungen des Flugsystems (Abbildung 3.6 oben) betrachtet, ist zu erkennen, dass das System nach kurzer Zeit einen stationären Wert annimmt. Die resultierenden Ausgangsgrößen entsprechen allerdings nicht den zuvor gewählten Eingangsgrößen. Das Flugsystem nimmt also nach einiger Zeit eine stabile Fluglage an, diese weicht aber von den erwarteten Werten deutlich ab. So erreicht z.B. die Bewegung in

x -Richtung nach ca. fünf Sekunden einen Wert von über 60 Meter. Bei Betrachtung der Rotation um die z -Achse wird erst ein stationärer Wert nach ca. $-40rad$ angenommen. Das würde bedeuten, dass sich das Flugsystem zunächst über sechs mal um die eigene Achse dreht bevor eine stabile Fluglage eingenommen wird.

3.2.2 Führungsgrößenvorfilter

Um die Differenzen zwischen den Eingangs- und den Ausgangswerten zu minimieren, ist es möglich, die einzelnen Eingangswerte zunächst mit einem Faktor zu verrechnen bevor diese auf das System aufgeschaltet werden. Dazu wird ein Führungsgrößenvorfilter \mathbf{K}_W definiert [19, S. 180 ff.]. Dieser wird in den Stellgrößenpfad des Zustandsraummodells eingefügt. Der daraus resultierende Signalflussplan ist in Abbildung 3.7 dargestellt. Damit das System im

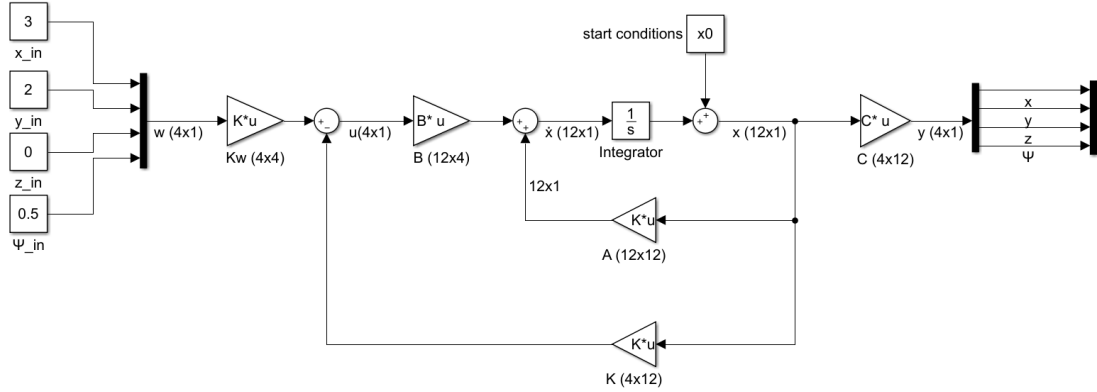


Abbildung 3.7: Signalflussplan eines Zustandsraummodells mit Zustandsrückführung und Führungsgrößenvorfilter

eingeschwungenen Zustand stationär genau ist, müssen die folgenden Bedingungen gelten:

$$\begin{aligned} \lim_{t \rightarrow \infty} \vec{y}(t) &\stackrel{!}{=} \vec{w}_0 \\ \dot{\vec{x}} &= 0 \quad \vec{x} = \vec{x}_\infty \end{aligned} \quad (3.18)$$

Nach einer bestimmten Zeit müssen also die Ausgangsgrößen \vec{y} des Systems die Werte der Eingangsgrößen \vec{w}_0 annehmen. Nach dieser Zeit darf sich der Wert der Zustandsgrößen nicht mehr ändern, bis neue Eingangsgrößen definiert werden. Die in 3.16 definierte Gleichung für ein Zustandsraummodell mit Zustandsrückführung kann nun mit den Bedingungen für stationäre Genauigkeit erweitert werden. Durch Berücksichtigung des Führungsgrößenvorfilters ergibt sich der folgende Zusammenhang:

$$\begin{aligned} \mathbf{A} \cdot \vec{x}_\infty + \mathbf{B}(\mathbf{K}_W \cdot \vec{w}_0 - \mathbf{K} \cdot \vec{x}_\infty) &= 0 \\ \vec{y} = \mathbf{C} \cdot \vec{x} \rightarrow \vec{y}_\infty = \mathbf{C} \cdot \vec{x}_\infty &\stackrel{!}{=} \vec{w}_0 \\ \Rightarrow \mathbf{K}_W &= \left[-\mathbf{C} \cdot (\mathbf{A} - \mathbf{B} \cdot \mathbf{K})^{-1} \cdot \mathbf{B} \right]^{-1} \end{aligned} \quad (3.19)$$

Mit der in der in 3.19 definierten Gleichung für den Führungsgrößenvorfaktor können nun die Werte für die \mathbf{K}_W bestimmt werden. Daraus ergibt sich eine Matrix der Größe 4×4 . Es wird eine weitere Simulation mit Matlab/Simulink durchgeführt, um die stationäre Genauigkeit des Systems zu prüfen. Auch hier werden die Eingangsgrößen auf den gleichen Wert wie in den vorherigen Simulationen gesetzt. Also $x_{in} = 3,0$, $y_{in} = 2,0$, $z_{in} = 0,0$ und $\psi_{in} = 0,5$. Die Ergebnisse sind in Abbildung 3.8 dargestellt. Es ist zu erkennen, dass sowohl die Translationen als auch die Rotation um die z -Achse stationär genau ist. Alle Ausgangsgrößen regeln sich auf den Wert der Eingangsgrößen ein. Bei genauerer Betrachtung der

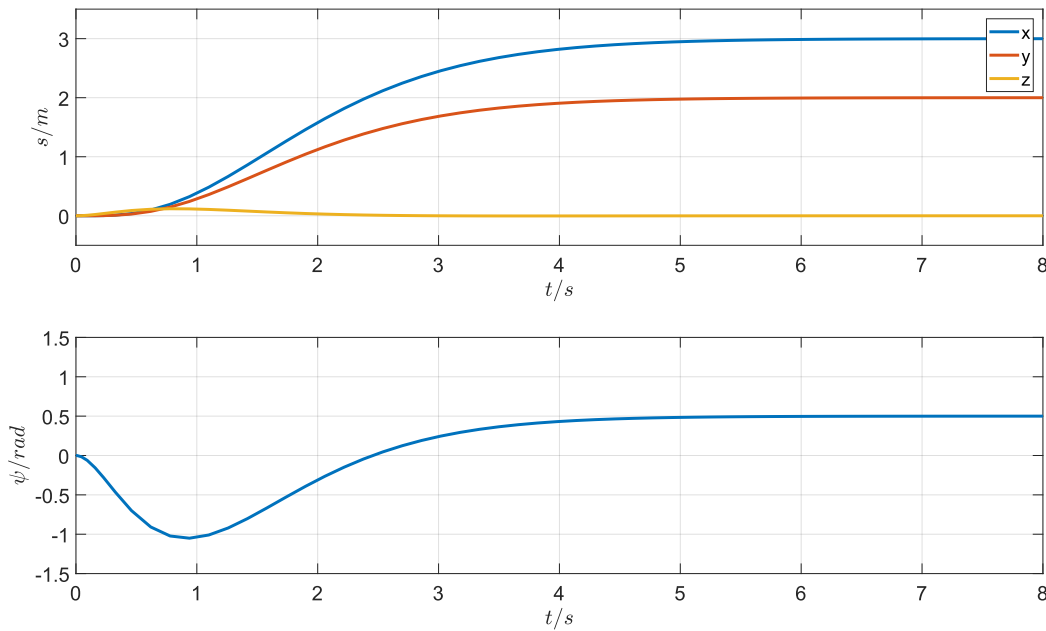


Abbildung 3.8: Einregelverhalten des Zustandsraummodells eines Flugsystems mit Zustandsrückführung und Führungsgrößenvorfilter.

Translation in z -Richtung (Abbildung 3.7 oben) ist am Anfang ein leichter Anstieg zu erkennen. Nach ca. zwei Sekunden regelt sich dieser Wert allerdings wieder auf die Vorgabe

von $z_{in} = 0,0$ ein. Auch bei der Rotation um die z -Achse (Abbildung 3.7 unten) braucht das System einige Zeit, um den Vorgabewert anzunehmen. Das liegt daran, dass alle Eingangsgrößen zum selben Zeitpunkt $t = 0$ auf das System geschaltet wurden und sich somit gegenseitig beeinflussen. Das System braucht etwas Zeit, damit sich alle Ausgangsgrößen auf die Werte der Eingangsgrößen einstellen. Dieses Verhalten kann durch eine bessere Vorgabe der Eigenwerte λ noch verbessert werden. In Abschnitt 3.2.3 wird auf eine sinnvolle Wahl von Eigenwerte noch genauer eingegangen, um ein gutes Regelverhalten des Systems zu erreichen.

Ändern sich die Parameter **A**, **B**, **C** der Regelstrecke, arbeitet der Regelkreis nicht mehr stationär genau, da es sich bei Benutzung eines Führungsgrößenvorfaktors nur um eine rein steuernde Maßnahme handelt. Es wird also nicht geprüft, welche Werte sich am Ausgang des Systems einstellen. Auch, wenn die Regelstrecke nicht genau dem realen System entspricht oder Störungen von außen auf das System wirken, ist eine stationäre Genauigkeit nicht mehr gewährleistet. Wird ein Flugsystems betrachtet, so ergeben sich bereits Änderungen des Zustandsraummodells, wenn kleine Änderungen an der Hardwarekonfiguration des Systems vorgenommen werden, also wenn z.B. zusätzliche Sensorik angebracht wird.

3.2.3 Zustandsregler mit Integrator im Stellgrößenpfad

Zum Ausgleich kleiner Änderungen des Systems wird der Ausgang des Systems auf den Eingang zurückgeführt, um einen geschlossenen Regelkreis zu erhalten. Zusätzlich wird ein Integrator in den Stellgrößenpfad des Systems hinzugefügt. Ergibt sich eine Differenz zwischen den Eingangs- und Ausgangsgrößen, werden diese nun mithilfe des Integrators ausgeglichen und dadurch eine stationäre Genauigkeit des Systems gewährleistet [19, S. 214 ff.]. Der daraus resultierende Signalflussplan ist in Abbildung 3.9 dargestellt. Die Rückführungsfaktoren werden hier wieder als Matrix **K** bezeichnet und die Vorfaktoren mit \mathbf{K}_i

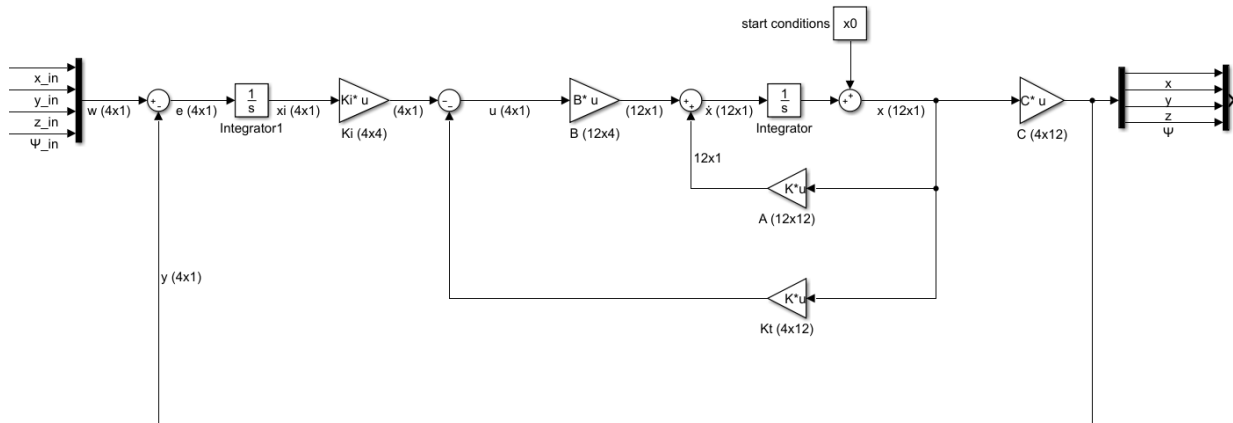


Abbildung 3.9: Signalflussplan eines Zustandsreglers mit I-Anteil im Stellgrößenpfad

Mithilfe des Signalflussplans kann nun wieder die allgemeine Zustandsgleichung $\dot{\vec{x}} = \mathbf{A} \cdot \vec{x} + \mathbf{B} \cdot \vec{u}$

sowie die Ausgangsgleichung $\vec{y} = \mathbf{C} \cdot \vec{x}$ angepasst werden. Dafür wird zunächst eine Zwischengröße \vec{x}_i eingeführt. Diese stellt die Differenz zwischen den Eingangs- und Ausgangsgrößen dar und ist somit wie folgt definiert:

$$\begin{aligned}\dot{\vec{x}}_i &= \vec{w} - \vec{y} \\ &= \vec{w} - \mathbf{C} \cdot \vec{x}\end{aligned}\tag{3.20}$$

Zusammen mit der Zustandsgleichung ergibt sich die folgende Darstellung:

$$\begin{bmatrix} \dot{\vec{x}} \\ \dot{\vec{x}}_i \end{bmatrix} = \underbrace{\begin{pmatrix} \mathbf{A} & \mathbf{0} \\ -\mathbf{C} & \mathbf{0} \end{pmatrix}}_{\mathbf{A}^*} \begin{bmatrix} \vec{x} \\ \vec{x}_i \end{bmatrix} + \underbrace{\begin{pmatrix} \mathbf{B} \\ \mathbf{0} \end{pmatrix}}_{\mathbf{B}^*} \vec{u} + \begin{bmatrix} \mathbf{0} \\ 1 \end{bmatrix} \vec{w}\tag{3.21}$$

Die Gesamtordnung des Systems steigt um eins und das lineare Regelgesetz $\vec{u} = -\mathbf{K} \cdot \vec{x}$ kann somit erweitert werden zu:

$$\vec{u} = -\underbrace{\begin{pmatrix} \mathbf{K} & \mathbf{K}_i \end{pmatrix}}_{\mathbf{K}^*} \cdot \begin{bmatrix} \vec{x} \\ \vec{x}_i \end{bmatrix}\tag{3.22}$$

Mit dem in 3.22 dargestellten linearem Regelgesetz können nun in der selben Weise wie in Abschnitt 3.2.1 die Rückföhrungsfaktoren bestimmt werden. Es werden allerdings nicht mehr die Systemmatrix \mathbf{A} und die Eingangsmatrix \mathbf{B} für die Berechnung genutzt, sondern die erweiterten Matrizen \mathbf{A}^* und \mathbf{B}^* , welche sich, wie in Gleichung 3.21 dargestellt, zusammensetzen. Dadurch, dass die Gesamtordnung des Systems um eins angestiegen ist, ergibt sich für die Matrix \mathbf{A}^* nun eine Größe von 16×16 . Dadurch ist es nötig, 16 Eigenwerte zu definieren, welche das dynamische Verhalten des Systems bestimmen. Zur Berechnung der Rückföhrungsfaktoren \mathbf{K} , bzw. der Vorfaktoren \mathbf{K}_i , kann auch hier wieder das Matlab-Kommando `place()` genutzt werden. Als Ergebnis ergibt sich die Matrix \mathbf{K}^* , welche sich, wie in Gleichung 3.22 dargestellt, zusammensetzt. Die 16 benötigten Eigenwerte werden auch hier erstmal zufällig gewählt und das Ergebnis mithilfe von Matlab/Simulink überprüft. Um das Einregelverhalten der einzelnen Translationen x , y und z besser darstellen zu können, werden diese zu unterschiedlichen Zeitpunkten auf einen Wert von 2,0 Metern gesetzt. Die Resultate sind in Abbildung 3.10 dargestellt. Zunächst wird auf das System ein Sprung in z -Richtung geschaltet. Nach 15 Sekunden wird zusätzlich eine Translation in x -Richtung definiert und nach weiteren 15 Sekunden eine Bewegung in y -Richtung. Der Sollwert für die Rotation um die z -Achse bleibt durchgehend auf dem Wert 0,0. Es ist zu erkennen, dass sich das System auf alle vorgegebenen Werte einregelt, dennoch ergeben sich während der Simulation unerwünschte Effekte. Bewegt sich das Flugsystem in x - oder y -Richtung verringert sich die Höhe des MAVs um bis über einen Meter. Nach kurzer Zeit wird allerdings der vorgegebene Wert von $z = 2,0m$ wieder erreicht. Bei Betrachtung eines MAVs in der Realität, neigt sich dieses in eine Richtung, um eine horizontale Bewegung auszuführen. Um dabei nicht an Höhe zu verlieren, müssen alle Rotordrehzahlen erhöht werden, da nicht mehr

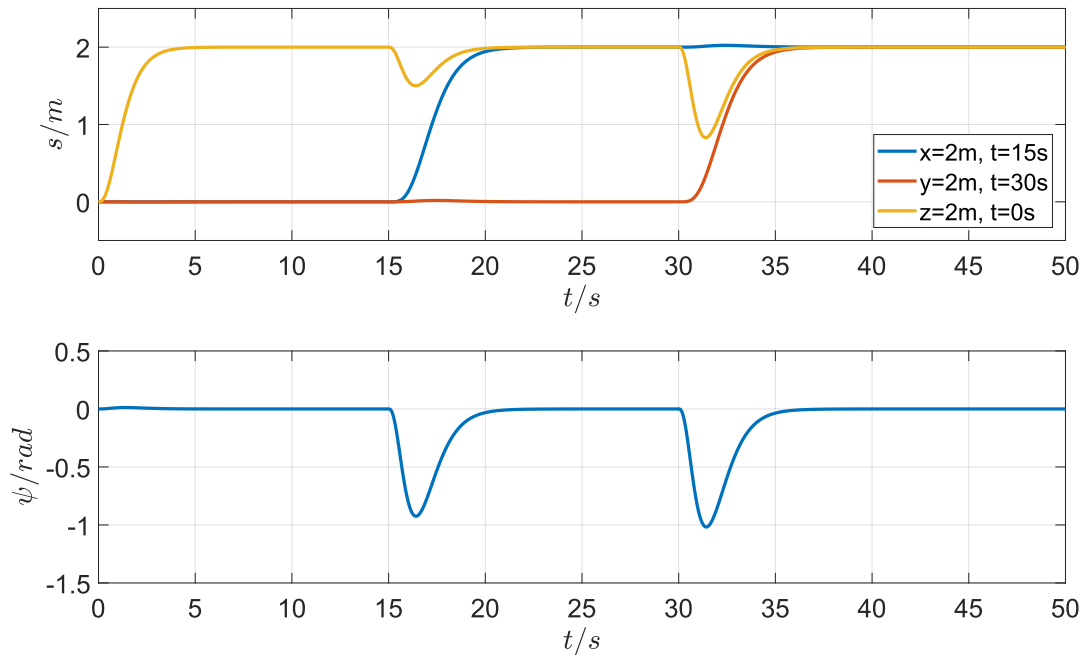


Abbildung 3.10: Einregelverhalten eines Zustandsreglers für ein MAV ohne optimierte Eigenwerte.

die gesamte Schubkraft, welche durch die Propeller erzeugt wird, senkrecht auf das System wirkt. Der Verlust an Höhe, welcher in Abbildung 3.10 zu erkennen ist, lässt sich somit dadurch erklären, dass die Regelung nicht schnell genug arbeitet und etwas Zeit braucht, um den Höhenverlust zu kompensieren. Zusätzlich dreht sich das Flugsystem bis zu 57° um die z -Achse, obwohl keinerlei Rotation vorgegeben wurde. Da das dynamische Verhalten der Regelung von der Wahl der Eigenwerte maßgeblich abhängt, können diese unerwünschten Effekte durch Vorgabe anderer Eigenwerte minimiert werden.

Ermittlung sinnvoller Eigenwerte

In Abschnitt 3.2.1 wurde bereits erläutert, wie die Wahl bestimmter Eigenwerte das Verhalten des geschlossenen Regelkreises beeinflussen. Darüber hinaus beeinflusst jeder einzelne Eigenwert das Verhalten von allen Zustandsgrößen. Es ist also nicht möglich, nur bestimmte Eigenwerte zu verändern, um z.B. nur das Verhalten in x -Richtung zu beeinflussen. Wird nur ein Eigenwert verändert, kann sich ein deutlich anderes Einregelverhalten des gesamten Systems ergeben. Um dennoch sinnvolle Werte bestimmen zu können, wird mithilfe von Matlab ein Programm erstellt, welches mehrere tausend Simulationen mit unterschiedlichen Eigenwerten durchführt. Zusätzlich können Vorgaben gemacht werden, um bestimmte Eigenschaften für das Einregelverhalten zu definieren. So kann z.B. festgelegt werden, dass sich die Höhe nur um einen bestimmten Wert ändern darf, wenn sich das Flugsystem in horizontaler Richtung bewegt. Werden von dem Programm Eigenwerte gefunden, welche die Vorgaben

erfüllen, werden diese ausgegeben. In Abbildung 3.11 ist der Ablauf des Programms als Programmablaufplan dargestellt. Zur Bestimmung der Matrizen \mathbf{K} und \mathbf{K}_i wird auch hier das

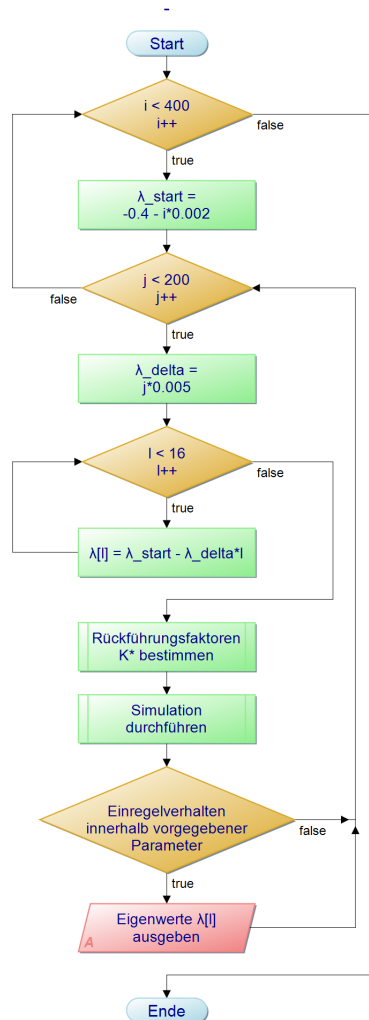


Abbildung 3.11: Programmablaufplan zur Bestimmung geeigneter Eigenwerte für einen Zustandsregler mithilfe von Testsimulationen

Matlab-Kommando `place()` genutzt. Es ist dabei aber nicht möglich, mehrere Eigenwerte auf den gleichen Wert zu setzen. Es müssen also 16 unterschiedliche Werte definiert werden. Zunächst wird dazu ein erster Eigenwert festgelegt. Dieser wird in dem Programmablaufplan in Abbildung 3.11 mit λ_{start} bezeichnet. Danach wird der Abstand zwischen den einzelnen Eigenwerten berechnet. Im Programmablaufplan ist dieser mit λ_{delta} bezeichnet. Daraufhin können alle 16 erforderlichen Werte bestimmt werden. Die Eigenwerte haben dadurch immer denselben Abstand zueinander. Anschließend werden diese Werte zusammen mit der Systemmatrix \mathbf{A}^* sowie der Eingangsmatrix \mathbf{B}^* an die Matlab-Funktion `place()` übergeben. Daraus ergibt sich die Matrix $\mathbf{K}^* \triangleq (\mathbf{K} \quad \mathbf{K}_i)$. Mithilfe der ermittelten Werte wird nun ei-

ne Simulation durchgeführt, um das Einschwingverhalten des geschlossenen Regelkreises zu überprüfen. Entspricht dieses Verhalten den vorgegebenen Parametern, werden die entsprechenden Eigenwerte ausgegeben. In diesen Parametern kann u.a. festgelegt werden, wie weit sich das MAV um die z -Achse maximal drehen oder wie viel das MAV absinken darf, sobald sich das Flugsystem in horizontaler Richtung bewegt. Anschließend wird λ_delta bzw. λ_start verändert und eine neue Simulation durchgeführt. Die Anzahl an Wiederholungen kann frei gewählt werden. Es sollte jedoch berücksichtigt werden, dass die Durchführung des in Abbildung 3.11 dargestellten Programms mehrere Stunden in Anspruch nehmen kann, wenn sehr viele Simulationen durchgeführt werden.

Ergeben sich Eigenwerte, welche die vorgegebenen Parameter erfüllen, können diese Anschließend genauer untersucht werden. Dazu wird wieder in Matlab/Simulink eine Simulation mit den entsprechenden Werten durchgeführt und das Einregelverhalten betrachtet. In Abbildung 3.12 sind die Ergebnisse dargestellt. Werden die optimierten Eigenwerte zur

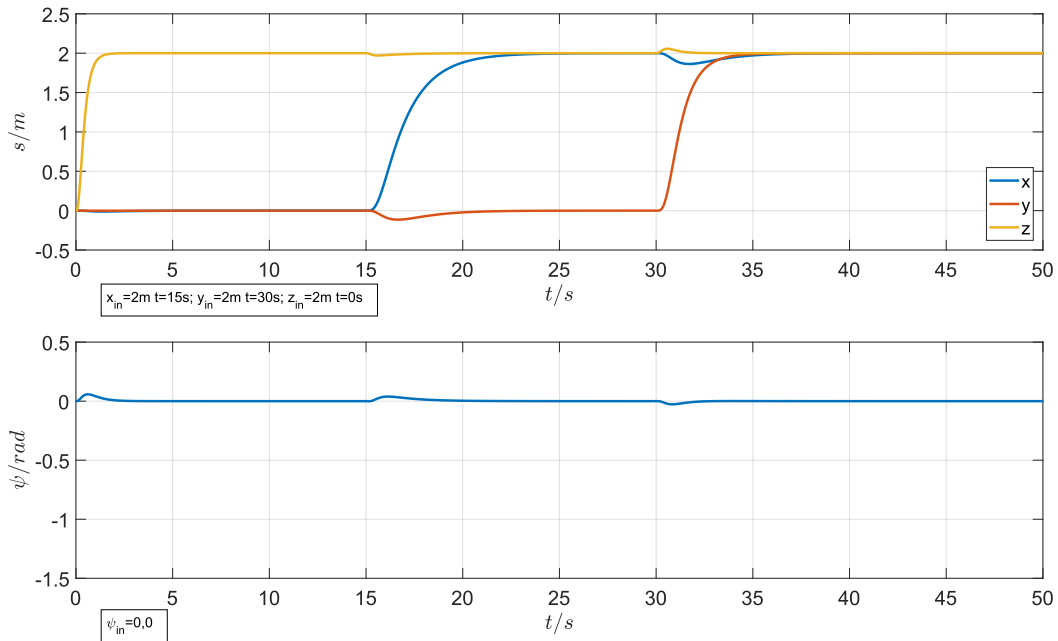


Abbildung 3.12: Einregelverhalten eines Zustandsreglers für ein MAV mit optimierten Eigenwerten.

Berechnung der Matrix \mathbf{K}^* genutzt, ergibt sich ein deutlich besseres Einregelverhalten des geschlossenen Regelkreises. Bewegt sich das MAV in x - oder y -Richtung, verändert sich die Höhe nur noch um wenige Zentimeter und nicht, wie in Abbildung 3.10 zu sehen, um über einen Meter. Auch rotiert das Flugsystem nur noch minimal um die z -Achse. Bei der Nutzung nicht optimierter Eigenwerte drehte sich das Flugsystem noch um bis zu 57° . Mit optimierten Eigenwerten verringert sich dieser Wert auf ca. 5° . Auch, wenn sich das Einregelverhalten deutlich verbessert hat, ist noch der Bezug auf ein reales Flugsystem zu betrachten. So ist zu

erkennen, dass das MAV sehr schnell die vorgegebene Höhe erreicht. Bereits nach ca. einer Sekunde ist das Flugsystem auf einem Wert von $z = 2,0$ Metern angestiegen.

Ermittlung der Rotordrehzahlen für einen Quadrocopter aus den resultierenden Stellgrößen

Um einen besseren Eindruck zu erhalten, ob ein System in der Realität so schnelle Änderungen annehmen kann, ist es sinnvoll, die resultierenden Stellgrößen u_1 bis u_4 zu betrachten. Damit kann der Stellaufwand, welcher auf das System wirkt, besser beurteilt werden. Da es sich bei diesen Werten allerdings nicht direkt um die einzelnen Rotordrehzahlen handelt, sondern um Zwischengrößen, sind zunächst die entsprechenden Werte zu ermitteln. Wie bereits in Abschnitt 3.1.1 dargestellt, ist der Zusammenhang zwischen den Stellgrößen und den einzelnen Rotordrehzahlen Ω_1 bis Ω_4 des hier betrachteten Quadrocopters wie folgt definiert:

$$\begin{aligned} u_1 &= b(\Omega_1^2 + \Omega_2^2 + \Omega_3^2 + \Omega_4^2) \\ u_2 &= b(\Omega_4^2 - \Omega_2^2) \\ u_3 &= b(\Omega_3^2 - \Omega_1^2) \\ u_4 &= d(\Omega_2^2 + \Omega_4^2 - \Omega_1^2 - \Omega_3^2) \end{aligned} \quad (3.23)$$

Die in 3.23 dargestellten Zusammenhänge können als Gleichungssystem mit vier Unbekannten betrachtet werden. Da auch vier Gleichungen zur Verfügung stehen, ist es möglich, z.B. mit dem Gauß-Algorithmus ein eindeutiges Ergebnis für die einzelnen Rotordrehzahlen zu erhalten. Zunächst wird das Gleichungssystem als Matrix dargestellt:

$$\begin{pmatrix} u_1 + u_{0,1} \\ u_2 \\ u_3 \\ u_4 \end{pmatrix} = \begin{pmatrix} \Omega_1^2 & \Omega_2^2 & \Omega_3^2 & \Omega_4^2 \\ b & b & b & b \\ 0 & -b & 0 & b \\ -b & 0 & b & 0 \\ -d & d & -d & d \end{pmatrix} \quad u_{0,1} = m \cdot g \quad (3.24)$$

Auf die Stellgröße u_1 wird hier zusätzlich die in Gleichung 3.13 definierte Ruhelage addiert. Dies ist notwendig, da durch die in Abschnitt 3.1.2 durchgeführte Linearisierung des Zustandsraummodells die Erdanziehungskraft vernachlässigt wurde, welche auf das Flugsystem wirkt. Die Stellgröße u_1 gibt den durch die Rotoren erzeugten Schub in z -Richtung an. Schwebt das MAV in der Luft, ohne die Höhe zu ändern, muss also die Schubkraft in vertikaler Richtung genau so groß sein wie die auf das System wirkende Erdanziehungskraft.

Anschließend kann nach den quadratischen Rotordrehzahlen aufgelöst werden:

$$\begin{pmatrix} \Omega_4^2 \\ \Omega_3^2 \\ \Omega_2^2 \\ \Omega_1^2 \end{pmatrix} = \begin{pmatrix} \frac{u_1+u_{0,1}}{4 \cdot b} + \frac{u_2}{2 \cdot b} + \frac{u_4}{4 \cdot d} \\ \frac{u_1+u_{0,1}}{2 \cdot b} + \frac{u_2}{2 \cdot b} + \frac{u_3}{2 \cdot b} - \Omega_4^2 \\ -\frac{u_1+u_{0,1}}{b} - \frac{2 \cdot u_2}{b} - \frac{u_3}{b} + 2 \cdot \Omega_3^2 + 3 \cdot \Omega_4^2 \\ \frac{u_1+u_{0,1}}{b} - \Omega_2^2 - \Omega_3^2 - \Omega_4^2 \end{pmatrix} \quad (3.25)$$

In Abbildung 3.13 sind die resultierenden Drehzahlen der einzelnen Rotoren nach der Zeit aufgetragen. Diese werden hier in Radiant pro Sekunde (rad/s) angegeben. Die benötigten Werte für den Schubfaktor b und den Zugfaktor d können der Tabelle 3.3 entnommen werden. Wird zunächst der eingeschwungene Zustand betrachtet, kann die Drehzahl der einzelnen

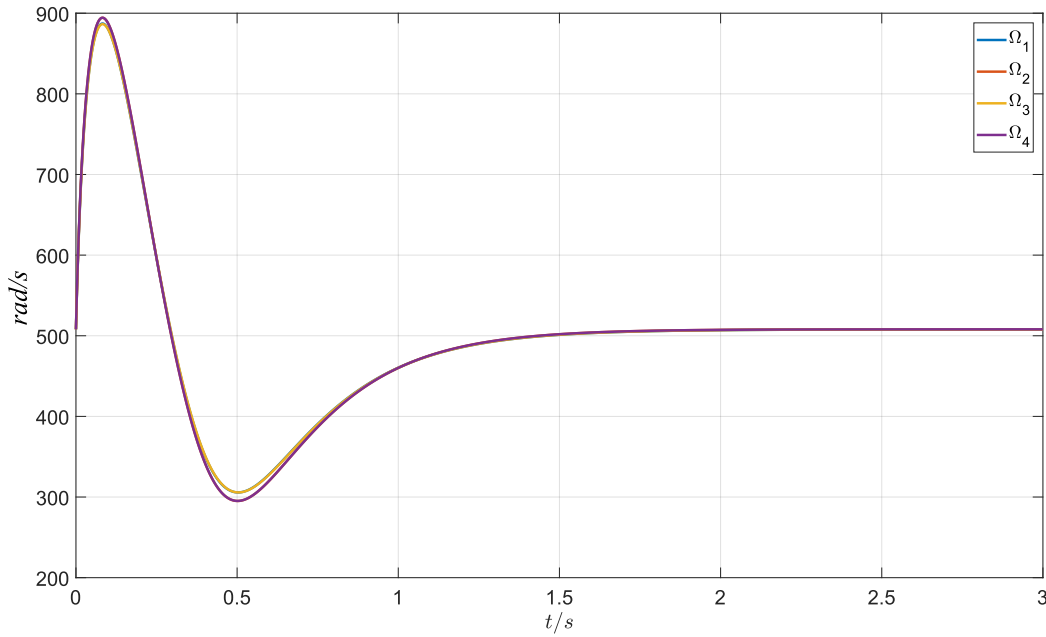


Abbildung 3.13: Resultierende Rotordrehzahlen bei einem Steigflug in z-Richtung um 2,0 Meter

Rotoren abgelesen werden bei dem das Flugsystem in einer konstanten Höhe schwebt. Dieser Wert liegt bei $508 rad/s$. Nun wird der Steigflug des MAVs genauer untersucht. Nach ca. 1,5 Sekunden erreicht das Flugsystem die vorgegebene Höhe von zwei Metern. Zunächst steigen dafür die Rotordrehzahlen nach etwa 0,1 Sekunden auf fast $900 rad/s$. Das bedeutet, dass die Drehgeschwindigkeiten der Propeller in kürzester Zeit um fast 80% ansteigen. Wie in Abschnitt 3.1.2 beschrieben, werden für die durchgeführten Simulationen die Eigenschaften des Quadcopters mit der Bezeichnung *Pelican* zugrunde gelegt. Die entsprechenden Parameter

für das Flugmodell wurden dabei dem Simulationsprogramm *rotorS Simulator* entnommen [9]. Zusätzlich zu den Parametern für den Schubfaktor und den einzelnen Trägheitsmomenten des Flugsystems ist noch ein Wert aufgeführt, welcher die maximale Drehgeschwindigkeit der einzelnen Rotoren angibt. Dieser Wert liegt bei 838 rad/s . Mithilfe dieser Angabe wird deutlich, dass das Flugsystem nicht so hohe Drehzahlen liefern kann, wie durch den Regler vorgegeben werden. Es ist also bei Betrachtung des Einregelverhaltens des geschlossenen Regelkreises wichtig, nicht nur das Verhalten der Ausgangsgrößen x , y , z und ψ zu untersuchen, sondern auch die entsprechenden Stellgrößen genauer zu betrachten.

Durch das in 3.11 beschriebene Programm, ergeben sich meist mehrere mögliche Eigenwerte, welche die vorgegebenen Parameter für das gewünschte Einregelverhalten des geschlossenen Regelkreises erfüllen. Um also eine Regelung entwickeln zu können, welche besser für ein reales Flugsystem geeignet ist, werden zunächst andere mögliche Eigenwerte vorgeben und wieder die resultierenden Stellgrößen betrachtet. Die Ergebnisse sind in Abbildung 3.14 dargestellt. Das resultierende Einregelverhalten ist dem in Abbildung 3.12 sehr ähnlich. Auch

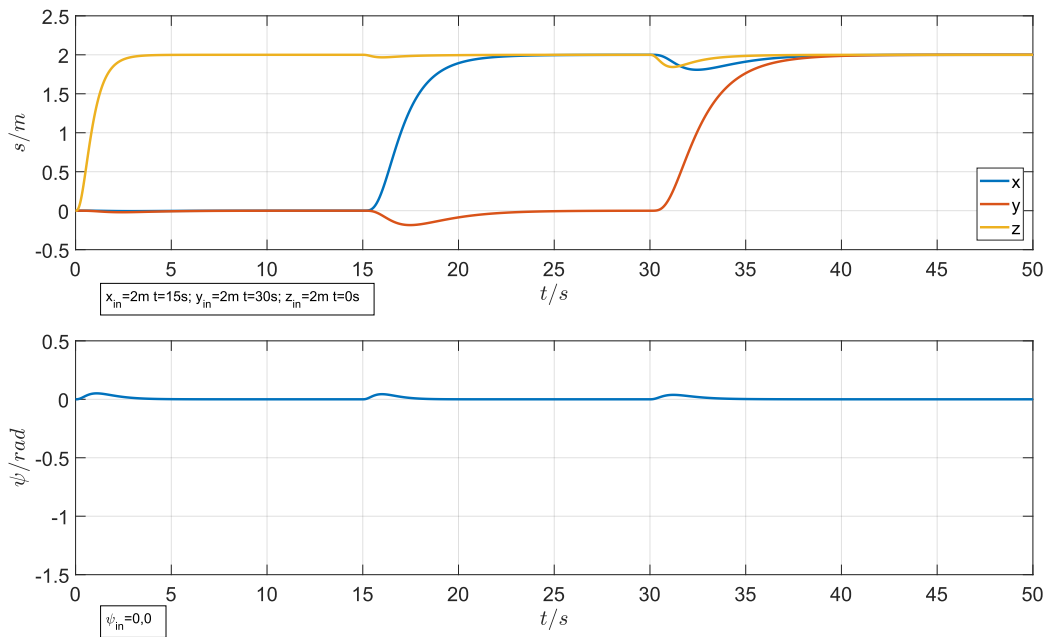


Abbildung 3.14: Einregelverhalten eines Zustandsreglers für ein MAV mit optimierten Eigenwerten unter Berücksichtigung der Rotordrehzahlen

hier erreicht das Flugsystem die vorgegebenen Sollwerte, ohne dass sich während des Fluges starke Abweichungen ergeben. So dreht sich das MAV nur noch weniger als $4,0^\circ$ um die z -Achse bei Bewegungen in translatorischer Richtung. Auch verringert sich die Höhe des Flugsystems nur noch um wenige Zentimeter, wenn eine Bewegung in x - oder y -Richtung durchgeführt wird. Der Unterschied im Vergleich zu dem Einregelverhalten in Abbildung 3.12 wird allerdings deutlich bei Betrachtung der Anstiegszeit des Flugsystems zum Zeit-

punkt $t = 0$. Die Höhe von zwei Metern wird hier etwas langsamer erreicht. Daraus lässt sich schließen, dass auch die Rotordrehzahlen einen geringeren Wert aufweisen. Auch hier werden wieder die resultierenden Stellgrößen \vec{u} mit den in 3.25 ermittelten Gleichungen in die entsprechenden Drehzahlen umgerechnet. Die Resultate sind in Abbildung 3.15 dargestellt. Der

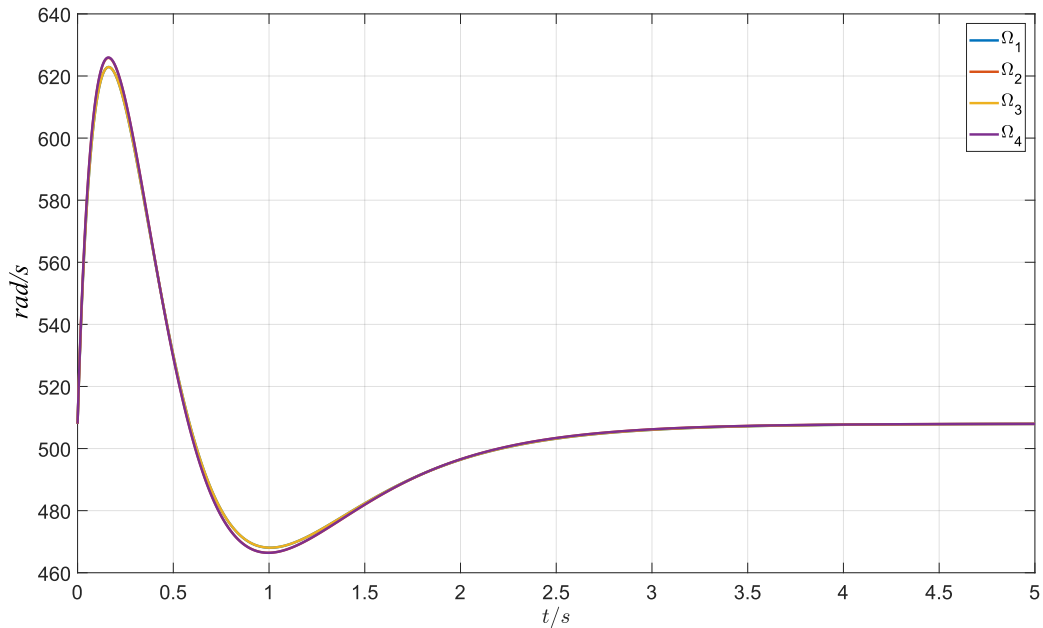


Abbildung 3.15: Resultierende Rotordrehzahlen bei einem Steigflug in z-Richtung um 2,0 Meter unter Berücksichtigung der Drehzahlgrenzen eines realen Flugmodells

Maximalwert liegt hier lediglich bei ca. 625 rad/s . Die Rotordrehzahlen steigen somit nicht mehr um fast 80% an wie in Abbildung 3.13, sondern nur noch um ca. 23%. Somit wird auch die maximal mögliche Geschwindigkeit von 838 rad/s nicht überschritten. Wird das Flugsystem später im Simulator getestet, sind demnach die zuletzt ermittelten Eigenwerte zu bevorzugen.

3.2.4 Zustandsregelung für einen Hexacopter

Bei dem Flugsystem, welches in der Realität zur Verfügung steht, handelt es sich um einen Hexacopter der Firma Ascending Technologies mit der Bezeichnung *Neo*. Dieses Flugmodell wird auch noch in Abschnitt 4.2.1 im Detail vorgestellt. Da dieses MAV nicht nur über vier, sondern sechs Rotoren verfügt, ist der zuvor entwickelte Zustandsregler geringfügig anzupassen. Es müssen allerdings nur die in 3.23 dargestellten Gleichungen verändert werden. Das Zustandsraummodell hingegen verändert die Form nicht. Es sind lediglich die Werte der Trägheitsmomente sowie der Masse des Flugsystems entsprechend anzupassen. Um nun die Gleichungen, welche den Zusammenhang zwischen den Stellgrößen \vec{u} und den Rotordrehzahlen $\vec{\Omega}$ beschreiben, für das Flugsystem *Asctec Neo* entwickeln zu können, ist die Anordnung der einzelnen Rotoren genauer zu betrachten. Dazu wird die schematische Darstellung in Abbildung 3.16 betrachtet. Dieses zeigt das Flugsystem in der Ansicht von oben.

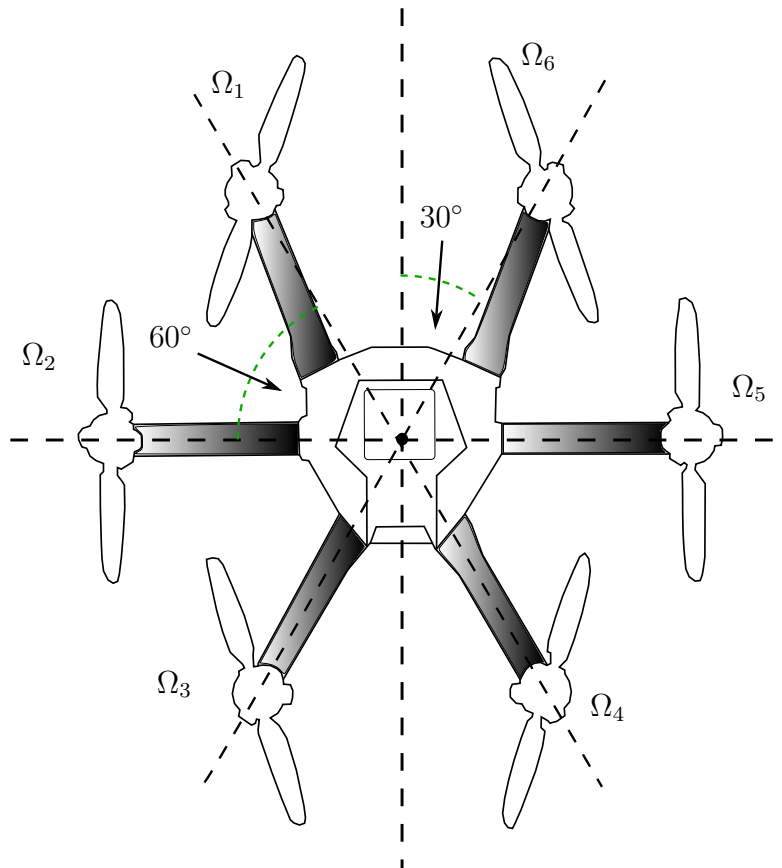


Abbildung 3.16: Schematische Darstellung der Anordnung der einzelnen Rotoren des Flugmodells *Asctec Neo*

Um nun die in 3.25 ermittelten Gleichungen entsprechend anzupassen, werden die Achsen des Flugsystems einzeln betrachtet. Damit sich das MAV z.B. vorwärts bewegt, bzw. um y -Achse dreht, muss der dritte und vierte Rotor eine höhere Drehzahl aufweisen als der erste

und sechste. Zusätzlich ist zu beachten, dass die Schubkraft dieser Rotoren nicht vollständig auf die y -Achse wirken. Die Schubkräfte beeinflussen demnach die Rotation nur anteilig. Die Rotation um die x -Achse wird hingegen von allen sechs Rotoren beeinflusst. So wirken die Rotoren zwei und fünf mit der vollständigen Schubkraft auf die x -Achse, da diese direkt auf dieser Achse liegen. Die restlichen Rotoren beeinflussen das Rotationsverhalten hier auch wieder nur anteilig. Daraus resultieren die folgenden Gleichungen:

$$\begin{aligned}
 u_1 &= b(\Omega_1^2 + \Omega_2^2 + \Omega_3^2 + \Omega_4^2 + \Omega_5^2 + \Omega_6^2) \\
 u_2 &= b(-\frac{1}{2}\Omega_1^2 - \Omega_2^2 - \frac{1}{2}\Omega_3^2 + \frac{1}{2}\Omega_4^2 + \Omega_5^2 + \frac{1}{2}\Omega_6^2) \\
 u_3 &= b(-\cos(30^\circ) \cdot \Omega_1^2 + \cos(30^\circ) \cdot \Omega_3^2 + \cos(30^\circ) \cdot \Omega_4^2 - \cos(30^\circ) \cdot \Omega_6^2) \\
 u_4 &= d(-\Omega_1^2 + \Omega_2^2 - \Omega_3^2 + \Omega_4^2 - \Omega_5^2 + \Omega_6^2)
 \end{aligned} \tag{3.26}$$

Auch hier ergeben sich wieder vier Gleichungen, welche den Zusammenhang zwischen den Stellgrößen und den Rotordrehzahlen darstellen. Allerdings verfügt das Flugsystem nicht mehr über vier, sondern über sechs Rotoren. Es existieren somit mehr Unbekannte als Gleichungen zur Verfügung stehen. Es ist dadurch nicht mehr möglich, eine eindeutige Lösung zu finden, da dieses Gleichungssystem unterbestimmt ist. Dennoch existieren verschiedene Lösungsverfahren, um unterbestimmte Gleichungssysteme zu lösen. In Matlab kann für so einen Fall die *LSQR*-Methode genutzt werden [22].

Um für das Flugsystem *Asctec Neo* geeignete Eigenwerte bestimmen zu können, wird zunächst das Zustandsraummodell angepasst, indem einige Parameter wie die Maße oder die Trägheitsmomente des MAVs verändert werden. Auch hier werden die Parameter dem Simulationsprogramm *rotorS Simulator* entnommen und sind in der Tabelle 3.4 zu finden. Zur

Paramter		Wert	Einheit
Trägheitsmoment x -Achse	I_x	$6,0887 \cdot 10^{-2}$	$kg \cdot m^2$
Trägheitsmoment y -Achse	I_y	$6,87913 \cdot 10^{-2}$	$kg \cdot m^2$
Trägheitsmoment z -Achse	I_z	$1,48916 \cdot 10^{-1}$	$kg \cdot m^2$
Masse	m	3,42	kg
Entfernung Rotor Mittelpunkt	l	0,2895	m
Schubfaktor	b	$1,269 \cdot 10^{-5}$	$kg \cdot m$
Zugfaktor	d	$2,0673 \cdot 10^{-4}$	$kg \cdot m^2$

Tabelle 3.4: Parameter des Hexacopters Asctec Neo

Bestimmung geeigneter Eigenwerte kann nun in der gleichen Weise wie für den Quadrocopter *Pelican* vorgegangen werden. Dazu wird auch hier das in Abbildung 3.11 dargestellte Programm genutzt. Das Einregelverhalten des Flugsystems *Asctec Neo* ist in Abbildung 3.17 zu finden. Zum Zeitpunkt $t = 0s$ steigt das Flugsystem zunächst auf eine Höhe von zwei Metern. Anschließend wird eine Translation in x - sowie in y -Richtung durchgeführt. Dabei ist zu erkennen, dass das Flugsystem ein paar Zentimeter an Höhe verliert. Nach kurzer Zeit

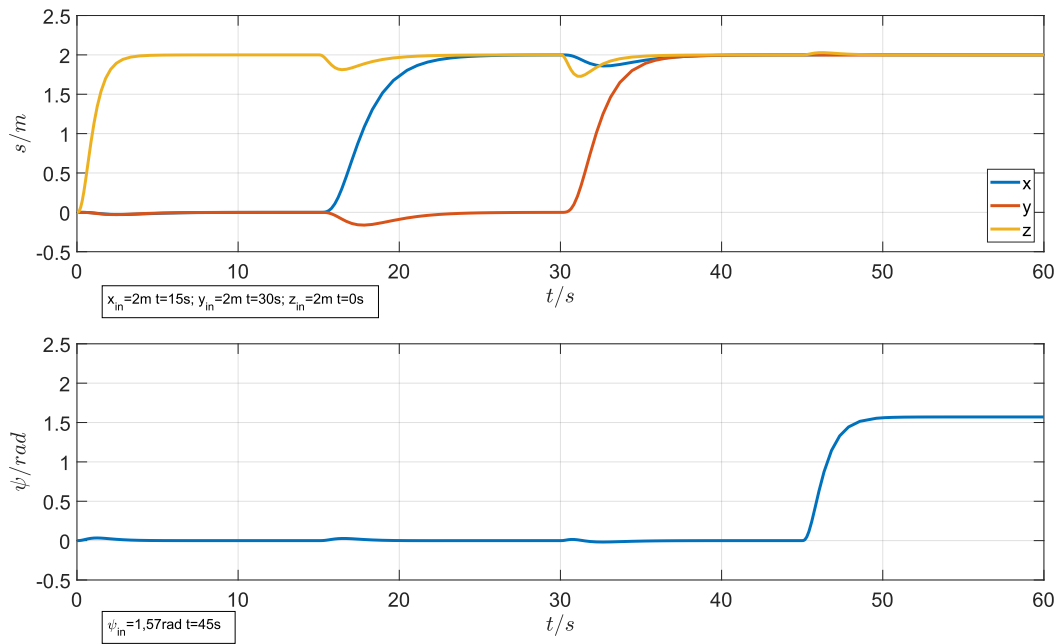


Abbildung 3.17: Einregelverhalten eines Zustandsreglers für das Flugsystem Asctec Neo mit optimierten Eigenwerten

wird allerdings die vorgegebene Höhe von zwei Metern wieder erreicht. Nach 45 Sekunden wird zusätzlich eine Drehung um 90° ($1,57rad$) durchgeführt, um zu prüfen, welchen Einfluss die Rotation des Flugsystems auf die Translationen hat. Es ist zu erkennen, dass der resultierende Einfluss auf die Bewegung in x -, y - und z -Richtung nur minimal ist. Es findet lediglich ein Anstieg in z -Richtung von unter fünf Zentimetern statt.

Wie bereits bei dem Quadrocopter *Pelican* muss auch für den Hexacopter *Asctec Neo* geprüft werden, ob das in Abbildung 3.17 dargestellte Verhalten auch in der Realität umsetzbar ist. Dazu werden wieder die resultierenden Rotordrehzahlen betrachtet. Diese werden dann in Hinblick auf die maximal mögliche Drehzahl der einzelnen Rotoren des Flugsystems bewertet. Die maximal mögliche Drehzahl der einzelnen Rotoren für das Flugsystem *Asctec Neo* kann auch hier wieder dem *rotorS Simulator* entnommen werden. Dieser Wert liegt bei $1047,2 \text{ rad/s}$. Abbildung 3.18 zeigt die resultierenden Rotordrehzahlen des Flugsystems *Asctec Neo* bei einer Bewegung in z -Richtung um zwei Meter. Wird zunächst der

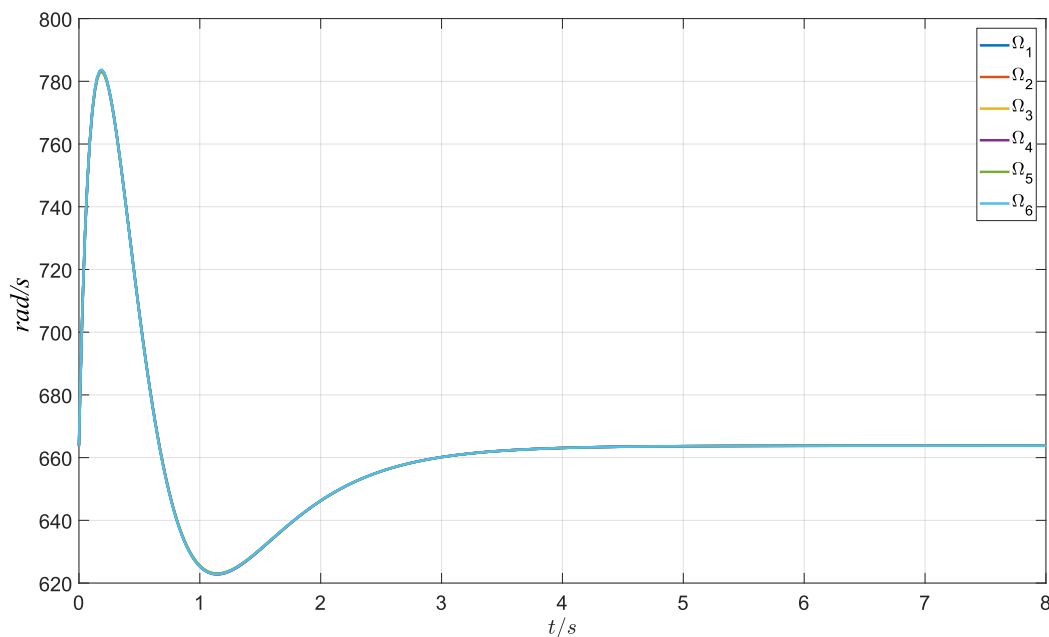


Abbildung 3.18: Rotordrehzahlen des Flugsystems *Asctec Neo* bei einem Steigflug auf zwei Meter

eingeschwungene Zustand betrachtet, lässt sich die Rotordrehzahl in Ruhelage ablesen. Diese liegt bei 664 rad/s . Zu Beginn steigt die Drehzahl der einzelnen Rotoren auf einen Wert von ca. 780 rad/s an. Das entspricht einer Erhöhung von etwa 17%. Die maximal mögliche Geschwindigkeit wird hier nicht überschritten.

3.2.5 Ermittlung von Zustandsgrößen unter Verwendung realer Sensorik

Damit der hier entwickelte Positionsregler ordnungsgemäß funktioniert, ist es notwendig, dass alle in Abschnitt 3.1.2 definierten Zustandsgrößen zur Verfügung stehen. In der Realität werden diese Größen von unterschiedlicher Sensorik bereitgestellt.

Rotationen und Rotationsgeschwindigkeiten

Die Rotationswinkel sowie die entsprechenden Rotationsgeschwindigkeiten um die x -, y - und z -Achse können mithilfe der IMU ermittelt werden (siehe dazu auch Abschnitt 2.3.1). Die Genauigkeit von diesem Sensor ist auch oft in der Realität sehr hoch und die Messwerte können somit direkt als Zustandsgröße verwendet werden.

Translationen und lineare Geschwindigkeiten

Um zusätzlich die Translation eines Flugsystems ermitteln zu können, wird zusätzliche Sensorik benötigt. In dieser Arbeit werden dazu Laserscanner verwendet sowie ein SLAM-Verfahren mit der Bezeichnung *hector_mapping*. In Kapitel 4 sind die verwendeten Software- und Hardware-Komponenten genauer dargestellt. Mithilfe von *hector_mapping* kann nun die Translation in x - und y -Richtung ermittelt werden. Die Höhe z wird durch eine Abstandsmessung zwischen dem Boden und dem Flugsystem erhalten. Dafür wird auch ein Laserscanner verwendet. Bei der Ermittlung der Translationen können kleinere Sprünge in den Messgrößen auftreten. Um solche Sprünge kompensieren zu können, werden die Messgrößen zunächst mit einem Tiefpass in folgender Form gefiltert:

$$y_i = a \cdot x_i + (1 - a) \cdot y_{i-1} \quad (3.27)$$

Durch das Filter können hochfrequente Störungen oder Sprünge der Messgröße kompensiert werden. Der Faktor a muss zwischen 0 und 1 liegen und gibt dabei die Stärke des Filters an. Der neue Wert ergibt sich aus dem aktuellen Messwert x_i , welcher mit dem alten gefilterten Wert y_{i-1} verrechnet wird. Eine andere Möglichkeit Sprünge der Messgröße zu verhindern ist die Verwendung eines Median-Filters. Dabei werden mehrere Messwerte aufgenommen und nur der mittlere Wert als Zustandsgröße verwendet.

Bei den letzten drei benötigten Zustandsgrößen handelt es sich um die linearen Geschwindigkeiten in x -, y - und z -Richtung. Das hier verwendete Flugsystem verfügt leider nicht über entsprechende Sensorik um diese Größe direkt messen zu können. Daher ist es notwendig, diese Zustandsgrößen aus anderen Messgrößen zu ermitteln. Die linearen Geschwindigkeiten können aus der aktuellen Position des Flugsystems bestimmt werden. Dazu wird die Translation nach der Zeit differenziert. Da die Position des Flugsystems allerdings nur als momentaner Messwert zur Verfügung steht, muss diese Größe numerisch differenziert werden. Die einfachste Form der Differenzierung ist durch die folgende Gleichung definiert:

$$\dot{x}_i = \frac{x_i - x_{i-1}}{\Delta t} \quad (3.28)$$

Um also die Geschwindigkeit \dot{x} in x -Richtung zu erhalten, wird die alte Position x_{i-1} von der aktuellen Position x_i subtrahiert und durch die vergangene Zeit Δt geteilt. Dadurch ergibt sich allerdings ein sehr stark verrauschtes Ergebnis und kann daher nicht als Zustandsgröße verwendet werden. Ein deutlich besseres Ergebnis ergibt sich durch die Nutzung des Savitzky-Golay Algorithmus [35]. Dabei werden zunächst mehrere Messpunkte von der Position verwendet und durch diese Punkte eine quadratische Funktion gelegt. Anschließend wird ein

gewichteter Mittelwert dieser Punkte berechnet und wieder durch die Zeit Δt geteilt. Die entsprechende Gleichung, für die Geschwindigkeit in x -Richtung ist wie folgt definiert:

$$\dot{x}_i = \frac{4 \cdot x_i + 3 \cdot x_{i-1} + 2 \cdot x_{i-2} + 1 \cdot x_{i-3} + 0 \cdot x_{i-4} - 1 \cdot x_{i-5} - 2 \cdot x_{i-6} - 3 \cdot x_{i-7} - 4 \cdot x_{i-8}}{60 \cdot \Delta t} \quad (3.29)$$

In Abbildung 3.19 ist ein Vergleich zwischen der einfachen Differenzierung und dem Savitzky–Golay-Algorithmus dargestellt. Die blaue Linie in Abbildung 3.19 unten, stellt die tatsächliche

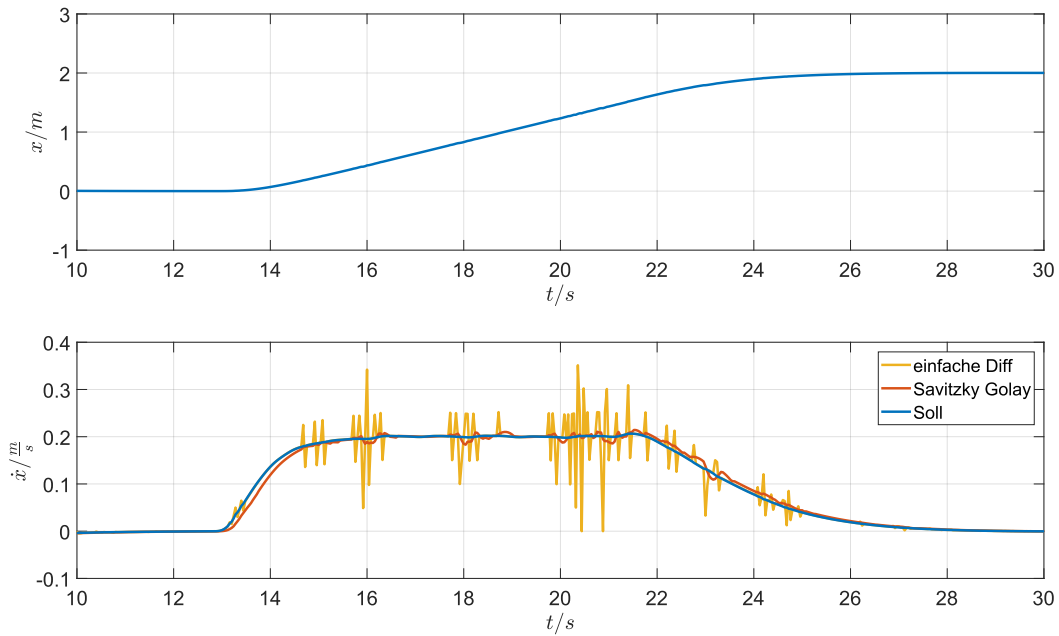


Abbildung 3.19: Berechnung der Geschwindigkeit aus der Position mithilfe der einfachen Differenzierung und dem Savitzky–Golay-Algorithmus

Geschwindigkeit des Flugsystems dar und dient als Referenz, um zu prüfen wie genau die Geschwindigkeit aus der Position ermittelt werden kann. Wird die Position wie in Gleichung 3.28 dargestellt, differenziert, ergibt sich ein deutlich verrauschtes Signal. Wird allerdings der in Gleichung 3.29 dargestellte Savitzky–Golay-Algorithmus verwendet, ergibt sich eine deutlich verbesserte Darstellung. Aber auch diese Methode weist kleinere Abweichungen von dem Sollwert auf. Um nun das Ergebnis noch weiter zu verbessern, kann noch ein anderer Sensor verwendet werden, welcher auf jedem Flugsystem vorhanden ist. Es handelt sich dabei um den Beschleunigungssensor, welcher ein Teil der IMU ist. Dieser misst die Beschleunigungen des Flugsystems in alle drei translatorischen Richtungen. Durch Integration dieser Beschleunigungen lässt sich somit wieder die Geschwindigkeit des Flugsystems ermitteln. Zur Integration wird das Trapezverfahren verwendet, welches durch Gleichung 3.30 definiert ist.

$$\dot{x}_i = \dot{x}_{i-1} + \frac{(\ddot{x}_{i-1} + \ddot{x}_i)}{2} \cdot \Delta t \quad (3.30)$$

Im stationären Zustand weist der Beschleunigungssensor allerdings einen Wert ungleich null auf. Wird dieser Messwert daraufhin integriert, weicht der integrierte Wert dadurch immer weiter vom realen Wert ab. Um diesen Drift zu vermeiden und einen sinnvollen Wert für die Geschwindigkeit des Flugsystems zu erhalten, wird die Geschwindigkeit, welche aus der Differenzierung der Position ermittelt wird, mit der Geschwindigkeit, welche durch Integration aus dem Beschleunigungssensor ermittelt wird, kombiniert. Zur Verknüpfung der beiden Signale wird ein Komplementärfilter verwendet.

In Abbildung 3.20 ist das kombinierte Signal dargestellt. Die blaue Linie stellt hier wie-

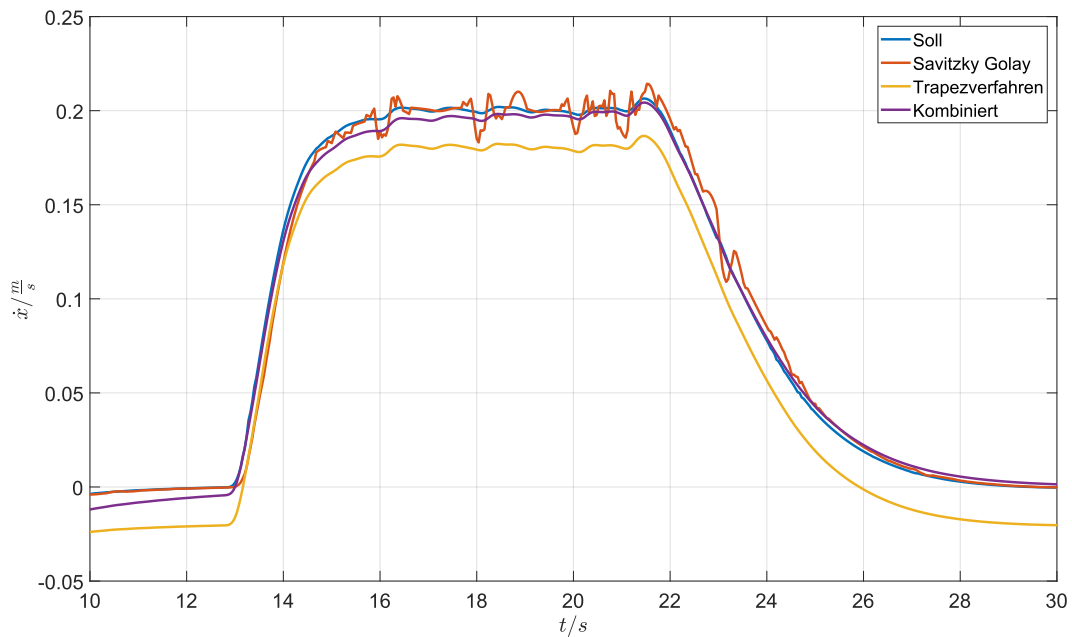


Abbildung 3.20: Bestimmung der Geschwindigkeit durch Kombination der Position und der Beschleunigung

der die tatsächliche Geschwindigkeit des Flugsystems dar. Die gelbe Linie ist die integrierte Beschleunigung des Flugsystems und die rote Linie ist die differenzierte Position. Die Kombination der resultierenden Geschwindigkeiten ist durch die violette Linie dargestellt. Es ist zu erkennen, dass das kombinierte Signal deutlich näher an dem Sollwert liegt. Das Rauschen des differenzierten Signals und die Abweichungen des integrierten Signals werden somit verringert.

3.2.6 Zweistufenregelung

In der Realität sind die Sensoren zur Messung der Rotationen eines Flugsystems meist deutlich genauer und robuster als die Sensoren zur Messung der Translationen. Rotationen werden mithilfe der internen IMU des Flugsystems gemessen, wohingegen die Translationen oft mit

optischen Verfahren bestimmt werden. Dabei können z.B. Kameras oder Lasercanner verwendet werden, um die Position des Flugsystems zu ermitteln. Äußere Einflüsse wie unterschiedliche Lichtverhältnisse oder dynamische Umgebungen haben allerdings großen Einfluss auf die Genauigkeit von optischen Verfahren. Es ist daher oft sinnvoll, den Positionsregler für ein reales Flugsystem in zwei einzelne Regler zu unterteilen. Ein Regler wird genutzt, um nur die Lage bzw. die Rotation des Flugsystems zu regeln. Als Sensorik benötigt dieser dann nur die IMU des MAVs. Mit dem zweiten Regler wird anschließend die Translation des Flugsystems geregelt. In Abbildung 3.21 ist dieses zweistufige Regelverfahren schematisch dargestellt. Die gesamte Regelung wird wie zuvor als Zustandsregler entwickelt. Daher werden auch hier

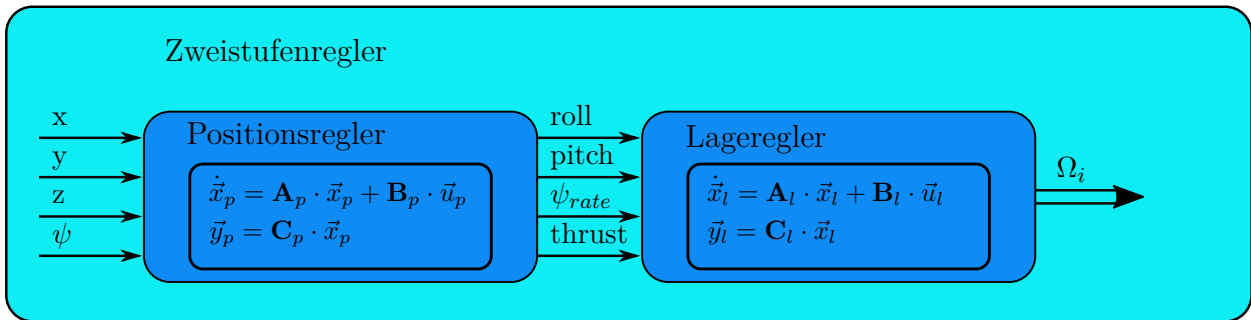


Abbildung 3.21: Schematischer Aufbau einer Positionsregelung als Zweistufenregler

die in Abschnitt 3.1.1 hergeleiteten DGLs verwendet. Bei der Entwicklung eines Zweistufenreglers werden die DGLs für die Rotation und die Translation des Flugsystems zunächst unabhängig voneinander betrachtet. Dabei ist zu erkennen, dass die Gleichungen 3.10 für die Rotationen nicht von den Gleichungen 3.5 für die Translationen abhängig sind. Es kann daher zunächst ein geeigneter Lageregler dimensioniert werden. Dieser ist dafür verantwortlich, dass das Flugsystem eine vorgegebene Rotation annimmt. Als Eingangsgröße kann dem Lageregler die Rotation um die x - und y -Achse ($roll$, $pitch$) sowie die Rotationsgeschwindigkeit um die z -Achse (ψ_{rate}) vorgegeben werden. Bei der vierten Eingangsgröße, welche dem Lageregler vorgegeben werden kann, handelt es sich um den Schub ($thrust$). Durch Vorgabe dieser vier Eingangsgrößen ist es möglich, das Flugsystem in jede Richtung zu bewegen. Soll ein Flugsystem manuell gesteuert werden, reicht diese Art der Regelung bereits aus, da die vier Eingangsgrößen $roll$, $pitch$, ψ_{rate} und $thrust$ über eine Fernbedienung vorgegeben werden können. Um allerdings eine definierte Position im Raum automatisch anzusteuern, wird ein weiterer Regler benötigt, welcher den Lageregler entsprechend ansteuert. Dieser zweite Regler ist dann der eigentliche Positionsregler. Zur Entwicklung dieses Positionsreglers werden nun die drei DGLs verwendet, welche die Translationen des Flugsystems beschreiben (siehe Gleichungen 3.5). Die Eingangsgrößen dieses Reglers sind dann die Translationen in x -, y - und z -Richtung sowie die Rotation um die z -Achse. Der Positionsregler steuert dann nicht mehr, wie der Regler in Abschnitt 3.2.3, die Rotoren des Flugsystems direkt an, sondern berechnet aus den vorgegebenen Eingangsgrößen die entsprechenden Stellgrößen zur Ansteuerung des Lagereglers.

Wenn die Verfahren zur Positionsbestimmung des Flugsystems schlechte Werte liefern, wird bei dem zweistufigen Regelverfahren nur der Positionsregler negativ beeinflusst. Der Lage-

regler hingegen kann weiterhin für ein stabiles Flugverhalten sorgen. Einige Flugmodelle verfügen bereits über einen Lageregler, welcher mit externen Systemen angesteuert werden kann. Steht so ein Flugsystem zur Verfügung, kann der Positionsregler aufbauend auf dem vorhandenen Lageregler entwickelt werden.

4 Systemdesign

In diesem Abschnitt werden die verwendeten Software- sowie Hardwarekomponenten genauer beschrieben. Einige dieser Bestandteile, wie die Hokuyo Laserscanner oder das Flugsystem *Neo*, wurden bereits in meiner Bachelorarbeit *Selbstlokalisierung eines Mikroflugsystems zu 3D-Kartierung von Innenräumen* [14] verwendet und werden daher nur noch ansatzweise dargestellt.

4.1 Software

Im Folgenden werden die verwendeten Softwarekomponenten beschrieben.

4.1.1 ROS

Bei dem Robot Operating System (ROS) handelt es sich nicht um ein Betriebssystem im eigentlichen Sinne, sondern um eine Sammlung von Bibliotheken und Programmen. Diese können genutzt werden, um Software für die verschiedensten Arten von Robotern zu entwickeln. Um Software zu entwickeln, werden sogenannte Nodes erstellt. Diese Nodes sind Programme, welche für verschiedene Arten von Roboterhardware erstellt werden. Viele Hersteller solcher Hardware bieten bereits fertige Nodes an. Von Hokuyo gibt es beispielsweise einen Node mit der Bezeichnung *urg_node* zur Ansteuerung eines Laserscanners. Mehrere einzelne Nodes können zusätzlich kommunizieren, um Daten auszutauschen. Dies geschieht über Topics. Mithilfe dieser Topics, werden verschiedene Arten von Daten benannt. Dadurch muss ein Node lediglich diesen Namen kennen, um die entsprechenden Daten abzurufen. Dadurch ergibt sich die Möglichkeit, viele kleine Programme zu erstellen, welche untereinander kommunizieren [26]. Es gibt auch die Möglichkeit verschiedene Arten von Sensordaten zu visualisieren. Dazu kann das Programm RViz verwendet werden, welches im ROS Framework integriert ist. Es ist damit z.B. möglich, Kamerabilder sowie Laserscans anzuzeigen.

Hector Mapping

Hector Mapping ist ein ROS-Node, welcher zur Lokalisierung von Robotern sowie Kartierung genutzt werden kann. Das Verfahren verwendet Laserdaten, um die Position eines Roboters innerhalb einer Umgebung schätzen zu können. Die Anwendung kombiniert ein 2D-SLAM-Verfahren mit einem 3D Navigationssystem. Dadurch wird eine zweidimensionale Karte der Umgebung erstellt. Zur genaueren Schätzung der Position werden allerdings nicht nur die Laserdaten genutzt, sondern auch andere Sensorwerte. Damit ist es z.B. möglich, auch die Neigungsinformationen einer IMU in die Berechnungen mit einzubeziehen [15]. Das Verfahren

wird in dieser Arbeit genutzt, um die Position eines MAVs bestimmen zu können. Auf die Konfiguration wird noch genauer in dem Kapitel 5 eingegangen.

Global Planner

Bei dem Node *Global Planner* handelt es sich um eine Anwendung, um einen Pfad innerhalb einer vorhandenen Karte zu planen [25]. Zunächst muss dafür die Position des mobilen Roboters bekannt sein, damit der Startpunkt des Pfades definiert ist. Zusätzlich wird eine Karte der Umgebung benötigt. Diese kann z.B. vorher mithilfe eines SLAM-Verfahrens erstellt werden. Anschließend wird ein Ziel vorgegeben. Befindet sich das Ziel innerhalb der Karte und sind keine Hindernisse vorhanden, welche den Weg versperren, kann erfolgreich ein Pfad erstellt werden. Ein Beispiel für solch eine Planung ist in Abbildung 4.1 dargestellt. Zunächst wird geprüft, welches Gebiet für den Roboter erreichbar ist, bzw. innerhalb

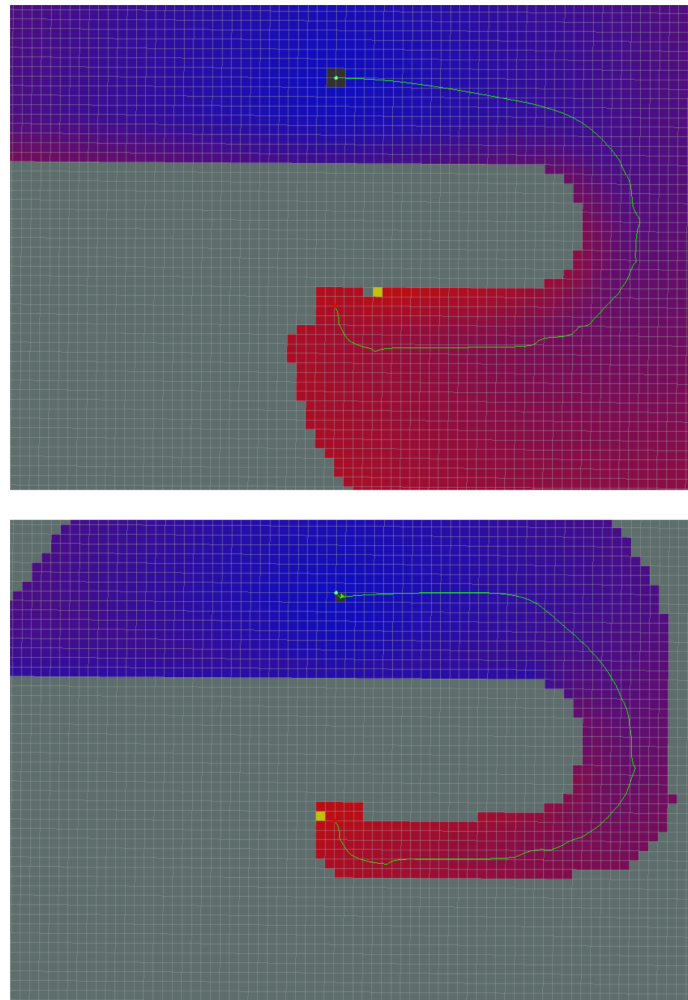


Abbildung 4.1: Pfadplanung innerhalb einer Karte mithilfe des ROS-Nodes *Global Planner* [25]. Standard Pfadplanung oben, Pfadplanung mit A*-Algorithmus [17] unten

welchen Bereichs ein Pfad geplant werden kann. Dieses Gebiet ist hier farblich dargestellt. Je nach verwendeten Planungsalgorithmus kann dieser Bereich größer oder kleiner sein. Die unterschiedlichen Farben stellen hier zusätzlich die Entfernung vom Roboter zum Zielpunkt dar. Werden die Standardeinstellungen für die Pfadplanung genutzt, ergibt sich das in Abbildung 4.1 (oben) dargestellte Verhalten. In Abbildung 4.1 (unten) wird dagegen der A* Algorithmus genutzt [17]. Damit wird ein deutlich kleinerer Bereich definiert, bei dem eine Pfadplanung möglich ist. Dadurch wird die Berechnungszeit minimiert. Es können sich so allerdings unterschiedliche Pfade ergeben, auch wenn der gleiche Zielpunkt vorgegeben wurde.

4.1.2 Rotors Simulator

Bei dem rotorS Simulator handelt es sich um ein modulares Simulationsprogramm für MAVs. Damit ist es möglich, für verschiedene Arten von Flugsystemen Software zu entwickeln. Der Simulator basiert auf Gazebo und wurde von der ETH Zürich angepasst, um die Nutzung mit MAVs zu vereinfachen [9].

Um Algorithmen für z.B. Pfadplanung oder Lokalisation erfolgreich entwickeln zu können, sind oft viele kleinere Tests erforderlich. Solche Versuche in der Realität erfordern viel Zeit, da das gesamte Flugsystem für Testflüge vorbereitet werden muss. Bestehen zu dem Zeitpunkt noch Fehler in den zu testenden Algorithmen, können Abstürze des Flugsystems die Folge sein. Dies verzögert die weitere Entwicklung deutlich. Mithilfe des rotorS Simulators ist es hingegen möglich, solche Versuche in einer kontrollierten virtuellen Umgebung durchzuführen. Treten während der Simulation Fehler auf, können diese besser nachgebildet werden, was zur Fehlerkorrektur beiträgt. Zudem ist es mit nur wenig Aufwand möglich, einige Flugmodelle zu integrieren. Hierfür sind lediglich einige Parameter in vorgegebenen Konfigurationsdateien anzupassen. Im Folgenden ist ein Ausschnitt aus der Konfigurationsdatei für das MAV mit der Bezeichnung *Neo* dargestellt, welches in Abschnitt 4.2.1 genauer vorgestellt wird.

```

1 <robot name="neo11" xmlns:xacro="http://ros.org/wiki/xacro">
2   [...]
3
4   <xacro:property name="use_mesh_file" value="true" />
5   <xacro:property name="mesh_file" value="package://rotors_description/
6     meshes/neo11.dae" />
7   <xacro:property name="mass" value="3.42" /> <!-- [kg] -->
8   <xacro:property name="body_width" value="0.2" /> <!-- [m] -->
9   <xacro:property name="body_height" value="0.234" /> <!-- [m] -->
10  <xacro:property name="mass_rotor" value="0.005" /> <!-- [kg] -->
11  <xacro:property name="arm_length" value="0.2895" /> <!-- [m] -->
12  [...]
13  <!-- Property Blocks -->
14  <xacro:property name="body_inertia">
15    <inertia ixx="6.08870e-02" ixy="0.0" ixz="0.0" iyy="6.87913e-02" iyz="
16      0.0" izz="1.48916e-01" /> <!-- [kg m^2] [kg m^2] [kg m^2] [kg m^2]
17      [kg m^2] [kg m^2] -->
18  </xacro:property>
19
20  [...]
21 </robot>

```

In diesen Dateien sind alle Parameter aufgeführt, um die Eigenschaften des genutzten Flugsystems zu beschreiben. Darunter finden sich Angaben wie das Gewicht ("*mass*") oder die Abmessungen ("*body_width*", "*body_height*") des Flugsystems. Zusätzlich kann auch ein 3D Modell ("*mesh_file*") hinzugefügt werden. Unter der Angabe "*body_inertia*" sind die Trägheitsmomente zu finden, welche die Trägheit um die einzelnen Achsen des Flugsystems, beschreiben.

Alle Komponenten, welche sich auf realen Flugsystemen finden lassen, können auch in dem Simulator mithilfe von Plugins genutzt werden. Dadurch ist es möglich, zunächst mithilfe des Simulators Software zu entwickeln und diese dann mit nur wenigen oder keinen Änderungen an realen Systemen zu nutzen. Das Programm ist dabei vollständig in ROS integriert und es können somit jegliche Sensorwerte mithilfe von ROS-Topics empfangen werden. Auch kann jeder Rotor des simulierten MAVs einzeln angesteuert werden [9].

Jedes Flugsystem, welches im rotorS Simulator zu Verfügung steht, verfügt bereits über grundlegende Sensorik wie ein Gyroskop oder Beschleunigungssensoren. Zusätzlich ist es möglich, weitere Sensorik zu nutzen. Dazu können die Konfigurationsdateien entsprechend angepasst werden. Einige zusätzliche Komponenten, wie Kameras oder Laserscanner stehen bereits zu Verfügung. Werden andere Arten von Sensorik benötigt, können auch eigene Plugins erstellt und integriert werden. Abbildung 4.2 zeigt beispielhaft ein MAV im rotorS Simulator, welches mit einer Stereo Kamera ausgestattet ist. Das MAV schwebt ca. einen Meter über dem Boden. Die Kamera filmt in Richtung eines Hauses. Es können auch andere Objekte ohne viel Aufwand in den Simulator integriert werden. Dazu stehen bereits viele verschiedene Gegenstände wie z.B. Regale oder Kisten zur Verfügung. So kann schnell eine Umgebung nachgebaut werden, welche die gewünschten Anforderungen erfüllt. Abbildung

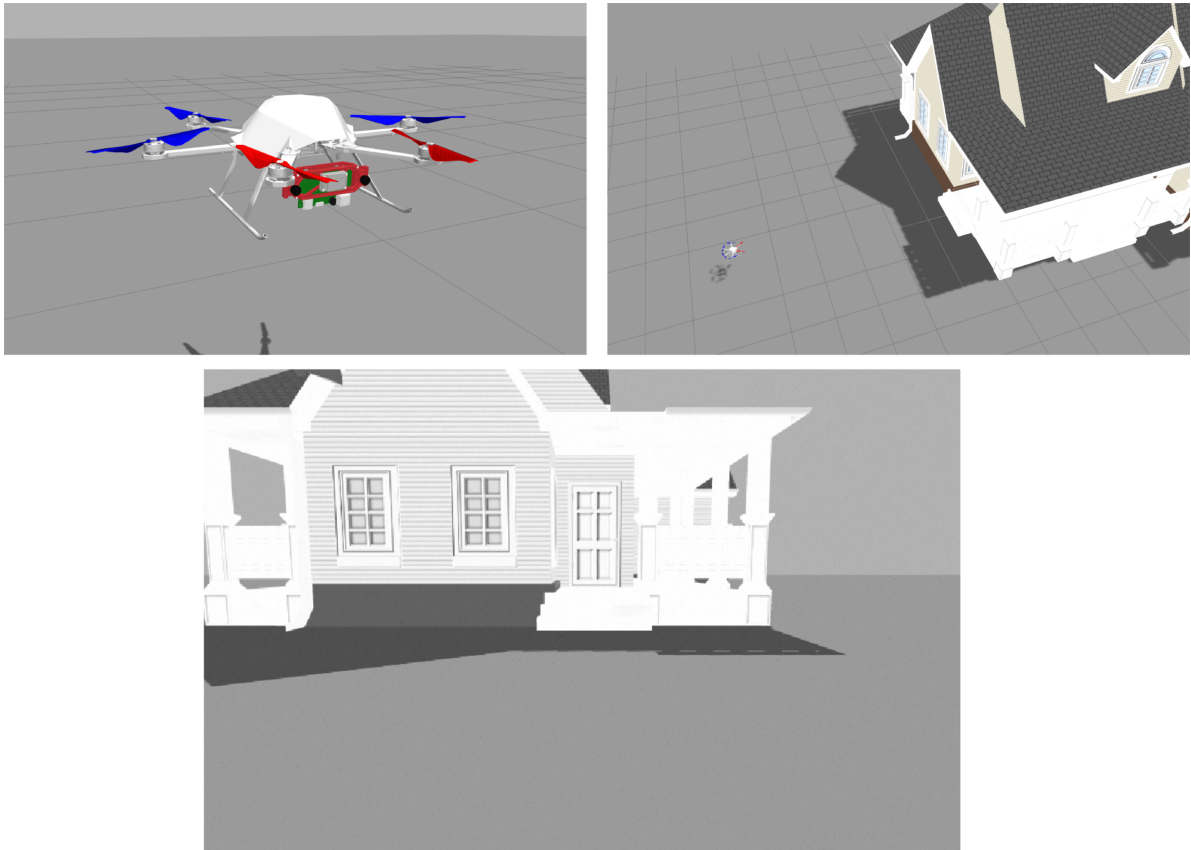


Abbildung 4.2: MAV ausgestattet mit einer Stereo Kamera im rotorS Simulator. Anzeige des resultierenden Kamerabildes in RViz

4.2 unten zeigt das resultierende Kamerabild. Dieses kann direkt mithilfe von z.B. RViz angezeigt werden. Die Bilddaten liegen dabei in derselben Form wie in der Realität vor. Werden also mithilfe dieser Kamerabilder Algorithmen entwickelt, können diese ohne Anpassung auf einem realen System verwendet werden.

4.2 Hardware

Der folgende Abschnitt befasst sich genauer mit den verwendeten Hardwarekomponenten.

4.2.1 Asctec Neo

Das hier verwendete Flugsystem ist von der Firma Ascending entwickelt und gebaut worden und trägt die Bezeichnung *Asctec Neo*. Der Vorteil dieser Plattform ist u.a. die kompakte Bauweise sowie die Möglichkeit, zusätzliche Hardware zu montieren. Das Flugsystem kann dabei bis zu zwei Kilogramm Zuladung aufnehmen [3]. Zur Befestigung zusätzlicher Hardware sind an der Ober- und Unterseite vier M3-Gewinde vorgesehen, welche in einem Abstand

von 80mm quadratisch angeordnet sind. Weitere Informationen sind auch in dem Datenblatt der Asctec Neo im Anhang in Abschnitt 8.1.1 zu finden.

Neo SDK

Das Flugsystem *Asctec Neo* verfügt über drei Mikrocontroller. Diese sind u.a. verantwortlich für die Steuerung des MAVs sowie zur Übertragung von Sensorwerten an den Linux Rechner, welcher mit dem Flugsystem verbunden ist. Von Ascending Technologies wird ein Software Development Kit (SDK) zur Verfügung gestellt, welches es ermöglicht, eigene Software für einen der drei Mikrocontroller zu entwickeln. In Abschnitt 5.4.2 wird diese Möglichkeit noch genutzt, um einen Servomotor anzusteuern.

Propellerschutz

Serienmäßig verfügt die Asctec Neo nicht über einen Schutz für die Rotoren. Um die Sicherheit bei Testflügen zu erhöhen und das Absturzrisiko zu verringern, wird noch ein Schutz für die einzelnen Propeller erstellt. Dazu wird ein sechs Millimeter starkes Aluminiumrohr genutzt und in die entsprechende Form gebogen (siehe Abbildung 4.3).

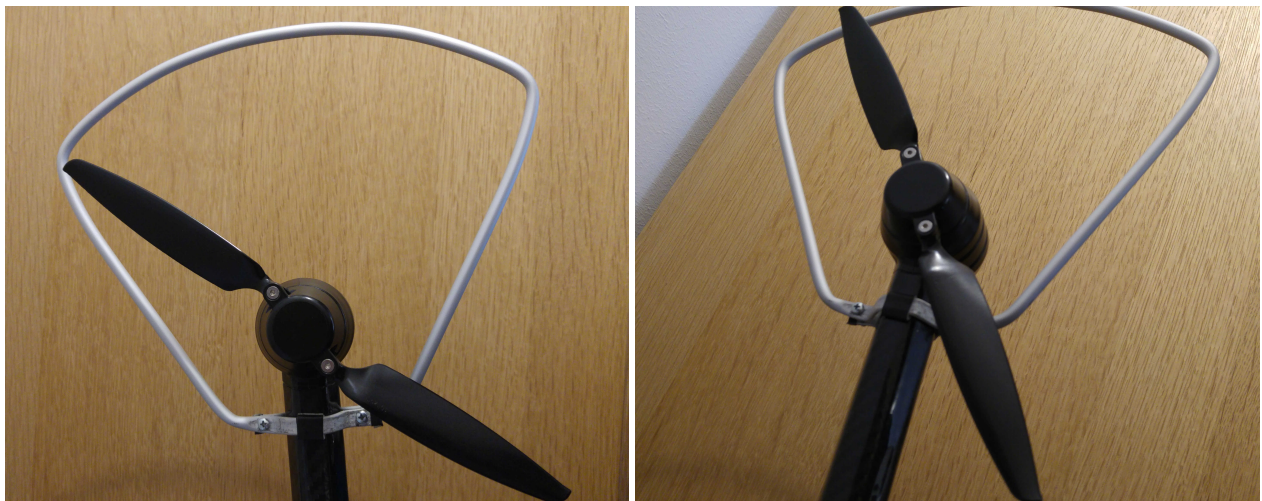


Abbildung 4.3: Propellerschutz für das Flugsystem Asctec Neo

4.2.2 Hokuyo Laserscanner

Das Flugsystem wird mit zwei Laserscannern ausgestattet, um sich mithilfe von SLAM lokalisieren und eine Karte der Umgebung erstellen zu können. Die Scanner sind von der Firma Hokuyo und tragen die Bezeichnung *UST-20LX*. Die Datenübertragung findet über Ethernet mit einer Geschwindigkeit von 40Hz statt. Das Gewicht beträgt lediglich 130 Gramm (ohne Verkabelung). Durch das geringe Gewicht bieten sich diese Scanner für die Nutzung auf Flugsystemen an [12]. Weiter Informationen sind ebenso im Anhang in Abschnitt 8.1.2 zu finden.

Montagevorrichtung

Zur Montage beider Laserscanner an der Asctec Neo, werden zwei Montagevorrichtungen konstruiert. Um das Gewicht nicht zu sehr zu erhöhen und dennoch eine stabile Konstruktion zu erhalten, wird drei Millimeter starkes Aluminiumblech genutzt. Der erste Scanner wird oben an der Drohne in horizontaler Ausrichtung befestigt. Mithilfe von Hector SLAM kann so eine zweidimensionale Karte der Umgebung erstellt werden und in dieser Karte eine Lokalisation stattfinden [15]. Der zweite Scanner hingegen wird in vertikaler Ausrichtung an der Unterseite des MAVs befestigt. Zusätzlich wird ein Servomotor genutzt, um den Scanner in einem bestimmten Bereich drehen zu können. Dadurch wird erreicht, dass der Bereich unter dem Flugsystem detailliert erfasst werden kann, ohne dass sich das MAV selber bewegen muss. Da mithilfe des oberen Laserscanners nur eine zweidimensionale Lokalisation stattfinden kann, wird die Höhe des Flugsystems anhand der Sensordaten des unteren Laserscanners bestimmt. Es können so die Translationen in alle Richtungen erfasst werden. Abbildung 4.4 zeigt eine bemaßte Darstellung der Montagevorrichtung für den unteren Laserscanner. Die

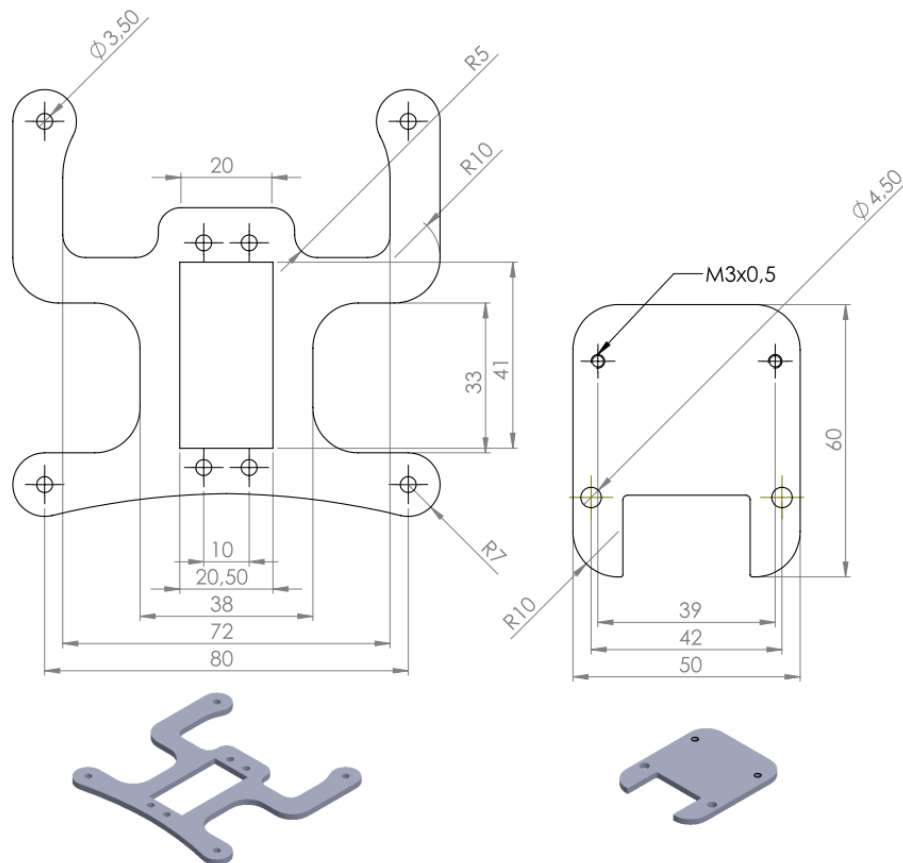


Abbildung 4.4: Vorrichtung zur Montage eines Laserscanners an der Unterseite des Flugsystems Asctec Neo in Verbindung mit einem Servomotor

Komponente auf der linken Seite ist fest mit der Asctec Neo verbunden. Dazu sind an dem

Flugsystem vier Verschraubungen vorgesehen, welche in einem Abstand von 80mm quadratisch angeordnet sind. In der Mitte der Halterung befindet sich eine rechteckige Aussparung. Diese dient zur Befestigung des Servomotors. Da sich der Scanner durch den Servomotor in einem Bereich von ca. -45° bis $+45^\circ$ dreht, ist sicherzustellen, dass bei Drehung des Laserscanners die Verkabelung nicht mit der Halterung kollidiert. Dafür sind an der Oberseite der Halterung entsprechende Aussparungen vorgesehen. Die Aussparungen an der linken und rechten Seite dienen lediglich der Gewichtsreduktion. Zur Verbindung des Laserscanners mit der Drehachse des Servomotors ist zusätzlich noch eine Adapterplatine erforderlich. Diese ist in Abbildung 4.4 auf der rechten Seite dargestellt und wird mithilfe von zwei M4 Gewindeschrauben mit dem Scanner verbunden. An der oberen Seite der Adapterplatte sind zwei M3 Gewindebohrungen vorhanden. Diese werden über einen Servohebel mit der Rotationsachse des Servomotors verbunden.

4.2.3 Gesamtkonfiguration

Im Folgenden wird der gesamte Aufbau des Flugsystems mit allen Hardwarekomponenten betrachtet. Dieses ist in Abbildung 4.5 dargestellt. An der Oberseite des MAVs wird ein



Abbildung 4.5: Gesamtkonfiguration aller Hardwarekomponenten an dem Flugsystem Asctec Neo

Laserscanner fest in horizontaler Ausrichtung montiert. Dieser dient zur Erstellung einer zweidimensionalen Karte der Umgebung sowie zur Lokalisation innerhalb dieser Karte. An der Unterseite befindet sich ein zweiter Laserscanner, welcher in vertikaler Ausrichtung montiert wird. Zusätzlich wird ein Servomotor verwendet, um diesen Scanner in Drehung zu versetzen. Dadurch kann der Bereich unterhalb des Flugsystems detailliert erfasst werden. Die durch die Laserscanner erfassten Daten werden an den Ubuntu- Rechner weitergeleitet,

welcher ebenfalls fest an der Unterseite des MAVs befestigt ist. Auch alle anderen Softwarekomponenten werden auf dieser Recheneinheit ausgeführt. Die Implementierung zusätzlicher Software wird in Kapitel 5 genauer betrachtet. An den Rotoren sind Schutzvorrichtungen montiert. Diese erhöhen bei ersten praktischen Versuchen die Sicherheit und sollen das Absturzrisiko verringern. Je nach Flugverhalten des MAVs können die Schutzvorrichtungen im späteren Testverlauf entfernt werden, um das Gewicht des Flugsystems zu verringern und damit die Flugzeit zu steigern.

5 Implementierung

Im folgenden Kapitel wird auf die Konfiguration sowie Implementierung einzelner Softwarekomponenten eingegangen. Die hier erstellte Software und die Konfigurationsdateien sind auch auf der beiliegenden CD im Anhang zu finden.

5.1 Zustandsregler als ROS-Node

Um den in Abschnitt 3.2 entwickelten Zustandsregler in dem 3D Simulator *rotorS Simulator* in der Realität nutzen zu können, wird dieser als ROS-Node implementiert. Der Regelalgorithmus wird dabei für das Flugsystem *Asctec Neo* optimiert. Zunächst wird ein ROS Packet mit der Bezeichnung *mav_position_control* erstellt. In diesem Packet wird der Node *neo_state_controller* implementiert, welcher den Zustandsregler für das Flugmodell *Asctec Neo* enthält. Der grundlegende Aufbau sowie die Funktionsweise des Nodes ist in Abbildung 5.1 dargestellt. Für eine korrekte Funktionsweise benötigt der Node *neo_state_controller*

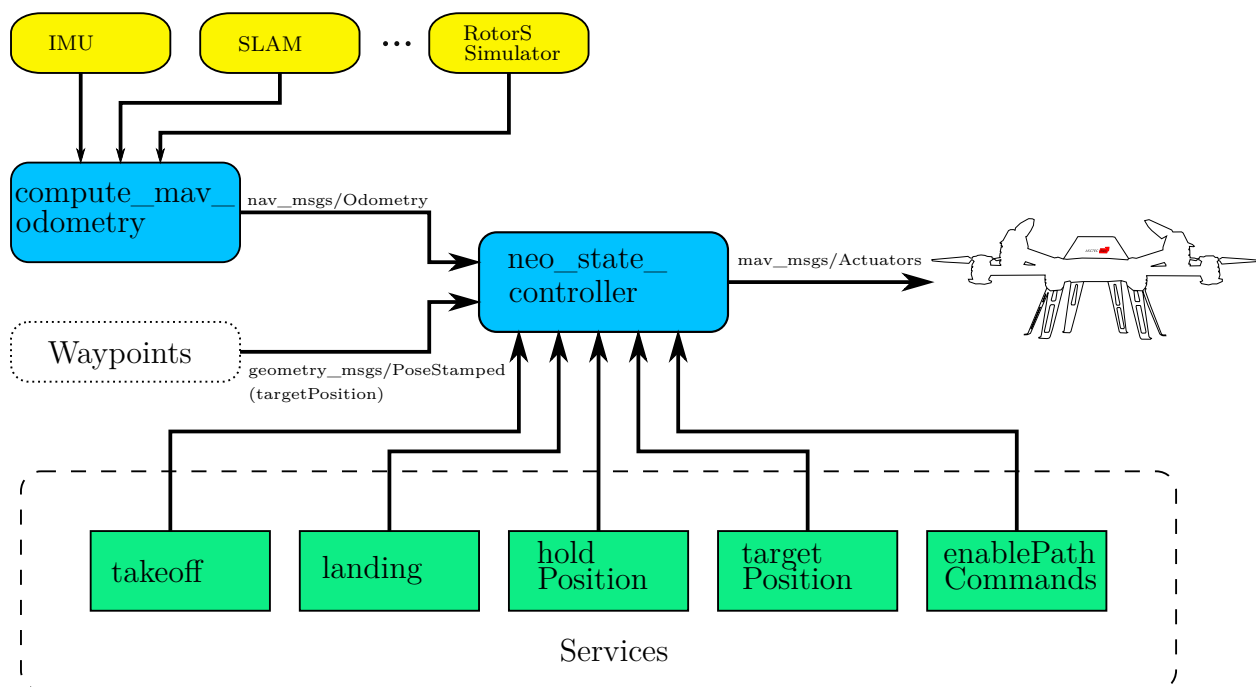


Abbildung 5.1: Funktionsweise des Zustandsreglers für das Flugsystem Asctec Neo als ROS-Node

zunächst die Position des Flugsystems. Dafür wird das Topic vom Typ *nav_msgs/Odometry*

genutzt. Dieses Topic enthält die folgenden Informationen:

```

1 std_msgs/Header header
2 string child_frame_id
3 geometry_msgs/PoseWithCovariance pose
4 geometry_msgs/TwistWithCovariance twist

```

Besonders wichtig sind dabei die Variablen *pose* und *twist*. Die Variable *pose* enthält die Translation des Flugsystems in *x*-, *y*- und in *z*-Richtung sowie die Rotation um alle drei Achsen. Die Variable *twist* hingegen beinhaltet die Linear- sowie die Winkelgeschwindigkeit des Flugsystems. Damit sind die in Abschnitt 3.1.2 ermittelten Zustandsgrößen, welche vom Zustandsregler benötigt werden, voll definiert. Die Quelle dieser Positionsinformationen ist für den Zustandsregler nicht relevant. Es ist lediglich wichtig, dass diese Informationen vom Typ *nav_msgs/Odometry* sind. Um die Positionsinformationen in dieser Art zur Verfügung zu stellen, wird der Node *compute_mav_odometry* verwendet. Dieser ist dafür verantwortlich, dass von verschiedenen Quellen Positionsinformationen dem Zustandsregler in der richtigen Form zur Verfügung gestellt werden. Dabei kann z.B. die IMU eines Flugsystems in Kombination mit einem geeigneten SLAM-Verfahren verwendet werden, oder, wenn Versuche im Simulator durchgeführt werden, die Positionsinformationen vom RotorS Simulator. Aber auch andere Quellen sind denkbar, wie externe Lokalisationsverfahren über Funk oder Ultraschall wie sie z.B. in der Arbeit von Tim Puls vorgestellt werden [23]. Dadurch, dass die Positionsinformationen für den Zustandsregler mit einem separaten ROS-Node ermittelt werden, kann der Zustandsregler flexibler eingesetzt werden, ohne diesen anpassen zu müssen, wenn Positionsinformationen von anderen Quellen zur Verfügung stehen. Weiterhin kann dem Node *neo_state_controller* über ein Topic vom Typ *geometry_msgs/PoseStamped* eine Zielkoordinate vorgegeben werden. Das benötigte Topic enthält dabei die folgenden Informationen:

```

1 std_msgs/Header header
2 geometry_msgs/Pose pose
3     —> geometry_msgs/Point position
4     —> geometry_msgs/Quaternion orientation

```

Die Variable *pose* enthält die Position (*position*) in *x*- *y*- und *z*-Richtung als Vektor sowie die Rotation (*orientation*) des Flugsystems als Quaternion. Quaternionen sind dabei lediglich eine andere Darstellung von Euler Winkeln und können ineinander umgerechnet werden. Somit kann mit der Variable *pose* die vollständige Position des Flugsystems im Raum vorgegeben werden. Als Sollwerte nimmt der Zustandsregler die Translation in *x*- *y*- und *z*-Richtung sowie die Rotation ψ um die *z*-Achse entgegen. Werden darüber hinaus noch Rotationen um die *x*- oder *y*-Achse vorgeben, werden diese ignoriert. Die Vorgabe von Zielkoordinaten kann nun mit einem weiteren Node erfolgen. Dieser kann z.B. dem Zustandsregler nacheinander Wegpunkte vorgeben, um einen definierten Pfad abzufliegen. Dieser Node wird noch genauer in Abschnitt 5.2 vorgestellt. Stehen alle benötigten Informationen zur Verfügung, werden die entsprechenden Rotordrehzahlen von dem Node *neo_state_controller* berechnet und an das Flugsystem *Asctec Neo* weitergegeben.

Zusätzlich ist es möglich, verschiedene ROS-Services zu nutzen, um das Flugsystem zu steuern. Diese sind in Abbildung 5.1 in grün dargestellt. Bei diesen Services handelt es sich um zusätzliche Befehle, welche dem *neo_state_controller* übergeben werden können. Die Services *takeoff* und *landing* können verwendet werden, um das Flugsystem automatisch starten und landen zu lassen. Wird eine Landung durchgeführt, werden auch die Rotoren abgeschaltet, wenn das MAV den Boden erreicht hat. Wird der Service *holdPosition* aufgerufen, bleibt das Flugsystem an der aktuellen Stelle stehen. Weitere Wegpunkte, welche durch einen separaten Node vorgegeben werden, werden dann ignoriert. Mithilfe des Services *targetPosition* kann manuell eine Zielkoordinate vorgegeben werden, welche daraufhin automatisch angesteuert wird. Dabei muss nicht, wie bei dem Datentyp *geometry_msgs/PoseStamped*, die Rotation als Quaternion angegeben werden, sondern kann direkt als Euler Winkel vorgegeben werden. Der Service *releasePathCommands* kann verwendet werden, um die Vorgabe von Wegpunkten, welche von dem Flugsystem angesteuert werden sollen, wieder freizugeben.

5.2 Trajektorienregler

In ROS steht bereits ein Node für die Planung eines Pfades innerhalb einer zweidimensionalen Karte zur Verfügung. Aufbauend auf diesem Node wird ein Trajektorienregler entwickelt, um dem in Abschnitt 5.1 vorgestellten Zustandsregler Wegpunkte vorzugeben. Der Zusammenhang zwischen dem Trajektorienregler und dem Pfadplaner ist in Abbildung 5.2 dargestellt. Um nun automatisch einen bestimmten Pfad abfliegen zu können, wird zunächst eine Ziel-

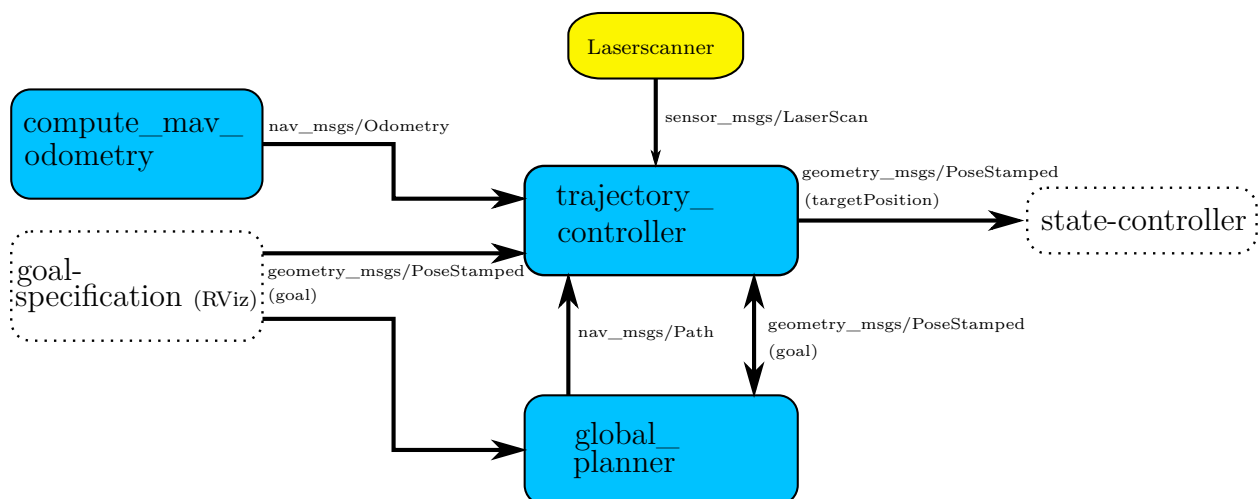


Abbildung 5.2: Funktionsweise des Trajektorienreglers zur Vorgabe von Wegpunkten

koordinate (*goal*) definiert. Diese kann entweder direkt über ein Linux Terminal vorgegeben werden oder mithilfe einer grafischen Oberfläche. Dazu bietet sich besonders das Visualisierungsprogramm RViz an, welches standardmäßig in ROS zur Verfügung steht. In Abbildung 5.3 ist die Vorgabe einer Zielkoordinate sowie der daraus resultierende Pfad beispielhaft dargestellt. RViz verfügt bereits über eine Möglichkeit, eine zweidimensionale Zielkoordinate

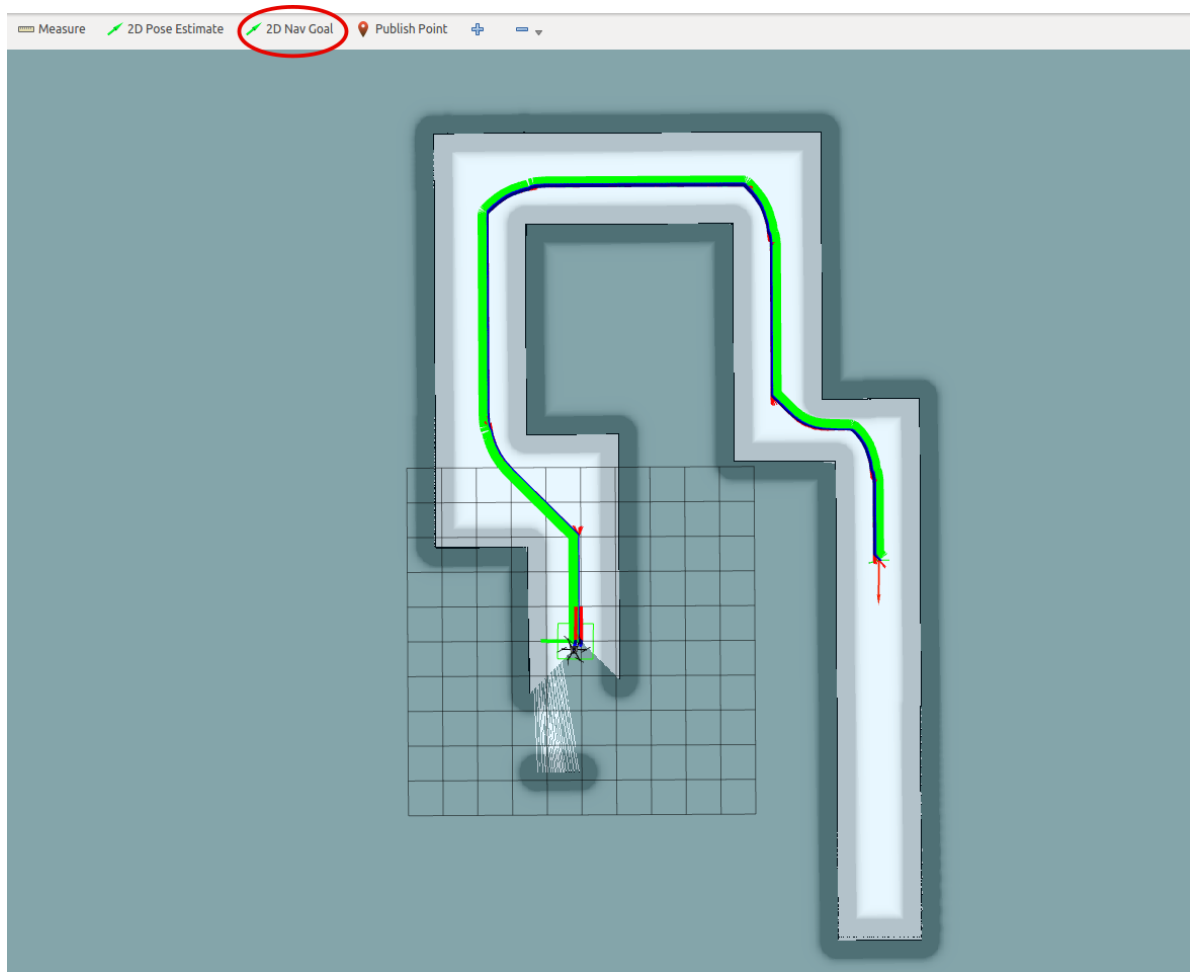


Abbildung 5.3: Vorgabe einer Zielkoordinate in einer vorhandenen zweidimensionalen Karte mithilfe von RViz

vorzugeben. Dazu wird die Schaltfläche *2D NAV Goal* am oberen Rand von RViz verwendet. Anschließend kann an eine beliebige Stelle auf der Karte geklickt werden. Die daraus resultierende Koordinate wird anschließend an den Pfadplaner *global_planner* übergeben. Daraufhin wird der entsprechende Pfad geplant und an den Trajektorienregler *trajectory_controller* übermittelt. Die in dem Pfad enthaltenen Wegpunkte werden anschließend nacheinander an den Zustandsregler weitergeleitet, welcher die entsprechende Koordinate ansteuert. Bevor der aktuelle Wegpunkt von dem Flugsystem erreicht wird, wird bereits der nächste Wegpunkt gesendet, um einen möglichst unterbrechungsfreien Flug zu gewährleisten. Der Pfad, welcher von dem Node *global_planner* berechnet wird, ist für Bodenroboter optimiert. Dabei wird davon ausgegangen, dass ein Bodenroboter nicht seitwärts fahren kann und die in dem Pfad enthaltenen Wegpunkte somit immer in Richtung des Pfades ausgerichtet sind. Ein MAV dagegen bietet auch die Möglichkeit, nicht nur vorwärts zu fliegen, sondern auch in andere translatorische Richtungen. Es ist daher nicht sinnvoll, die Vorderseite des Flugsystems immer genau zum Pfad auszurichten. Aus diesem Grund gibt der Trajektorienregler nicht

jede Rotation um die z -Achse an den Zustandsregler weiter. Das Flugsystem soll nur dann rotieren, wenn sich die Rotation in dem aktuellen Wegpunkt um einen größeren Wert ändert. Dieser Wert ist variable und kann somit frei in dem Node *trajectory_controller* eingestellt werden. Somit dreht sich das MAV beispielsweise um die z -Achse, wenn eine 90° Kurve angefliegen wird. Zusätzlich gibt es die Möglichkeit, dem Trajektorienregler Daten von einem Laserscanner zur Verfügung zu stellen. Damit kann der lokale Bereich um das Flugsystem überwacht werden. Befindet sich ein Hindernis auf dem Weg, welches zuvor nicht in der Karte vorhanden war, wird die zuvor definierte Zielkoordinate erneut an den Pfadplaner gesendet. Daraufhin wird ein neuer Pfad geplant, um das Hindernis zu umfliegen.

5.3 Konfiguration von RotorS Simulator

Um den entwickelten Zustandsregler umfangreich testen und evaluieren zu können, wird für die ersten Versuche der RotorS Simulator genutzt. Zunächst ist dieser dafür entsprechend zu konfigurieren, um die benötigten Anforderungen zu erfüllen. Es sind bereits einige verschiedene Flugmodelle verfügbar. Darunter ist auch das Flugmodell *Asctec Neo*. Wird der folgende Befehl ausgeführt, öffnet sich der Simulator mit dem Flugmodell *Asctec Neo* in einer leeren dreidimensionalen Umgebung:

```
roslaunch rotors_gazebo mav.launch mav_name:=neo11 world_name:=basic
```

Bereits jetzt liefert das MAV Sensorwerte von der IMU und kann wie ein reales System angesteuert werden. Dazu werden lediglich die gewünschten Rotordrehzahlen an ein Topic vom Typ *mav_msgs/Actuators* gesendet. Dazu kann der folgende Befehl in einem Linux Terminal ausgeführt werden:

```
rostopic pub /neo11/command/motor_speed mav_msgs/Actuators
'{'angular_velocities': [100, 100, 100, 100, 100, 100]}'
```

Dadurch rotiert jeder der sechs Rotoren der Asctec Neo mit einer Geschwindigkeit von 100 rad/s . Damit nun das Flugsystem von dem entwickelten Zustandsregler gesteuert werden kann, sendet der Node *neo_state_controller* die berechneten Rotordrehzahlen an das Topic mit dem Namen */neo11/command/motor_speed*. Sollen die Rotordrehzahlen nicht an den Simulator, sondern an das reale Flugsystem gesendet werden, ist lediglich der Name des Topics anzupassen.

5.3.1 3D-Umgebung

Um die dreidimensionale Umgebung innerhalb des RotorS Simulators zu individualisieren, können verschiedene Gegenstände eingefügt werden. In Abbildung 5.4 ist das 3D-Fenster des RotorS Simulator mit dem Flugsystem *Asctec Neo* sowie weiteren Objekten dargestellt. Es sind bereits viele verschiedene Gegenstände vorhanden, welche in den RotorS Simulator eingefügt werden können. Dazu kann auf der linken Seite des Simulators der Reiter *Insert*

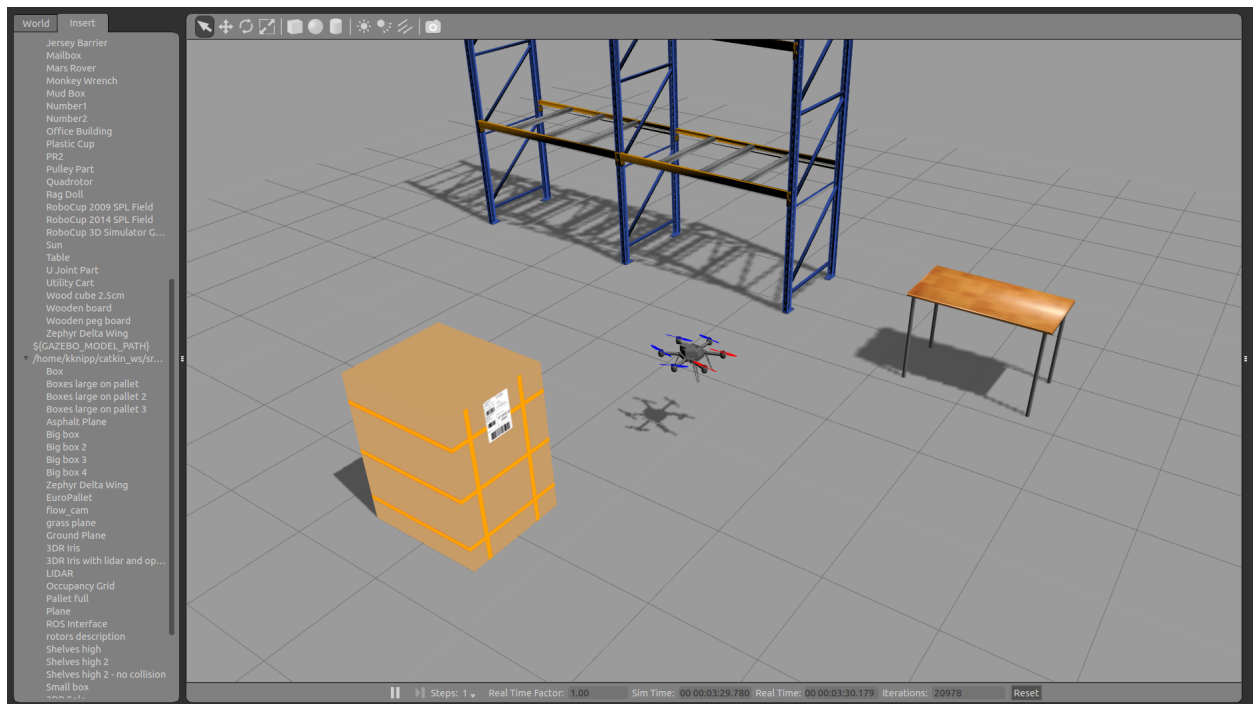


Abbildung 5.4: 3D Umgebung des RotorS Simulators mit dem Flugsystem Asctec Neo und verschiedenen frei wählbaren Objekten

ausgewählt werden. Anschließend wird das gewünschte Objekt, beispielsweise ein Regal oder ein Tisch, in das 3D-Fenster gezogen. So können schnell verschiedene Testumgebungen aufgebaut werden, um das System gut evaluieren zu können.

5.3.2 Plugins

Zusätzlich zu bereits vorhandenen Komponenten, können auch eigene Plugins für den RotorS Simulator erstellt werden. Um den in Abschnitt 4.2.3 beschriebenen Aufbau möglichst genau in der Simulation nachbilden zu können, wird noch ein Plugin für die beiden Hokuyo Laserscanner sowie für den Servomotor benötigt. Ein Plugin setzt sich aus zwei Teilen zusammen. Die erste Komponente, das eigentliche Plugin, ist ein ROS-Node, welcher definiert, wie sich ein Objekt im Simulator verhält, also z.B. ob ein Gegenstand rotiert oder sich linear bewegt. Bei der zweiten Komponente handelt es sich um eine Konfigurationsdatei. Darin können verschiedene Parameter definiert werden, beispielsweise der Name des Objekts oder die Position im 3D-Raum.

Hokuyo UST-20LX Plugin

In dem RotorS Simulator ist bereits ein Plugin für einen Laserscanner vorhanden. Die technischen Eigenschaften stimmen allerdings nicht mit dem hier verwendeten Laserscanner von Hokuyo überein. Es müssen daher einige Parameter wie der Scanbereich oder die Abtastrate

angepasst werden. Im Folgenden ist ein Ausschnitt aus der entsprechenden Konfigurationsdatei dargestellt:

```

1  [...]
2  <visual>
3    <origin xyz="0 0 0" rpy="0 0 1.5707"/>
4    <geometry>
5      <mesh filename="package://mav_state_controller_sim/meshes/
6        hokuyo_ust_20lx.dae" scale="0.9 0.9 0.9"/>
7    </geometry>
8  </visual>
9  [...]
10 <gazebo reference="${namespace}/${device_name}_link">
11   <sensor name="laser" type="ray">
12     <pose>0 0 0 0 0 0</pose>
13     <ray>
14       <scan>
15         <horizontal>
16           <resolution>1</resolution>
17           <max_angle>2.3562</max_angle> <!-- 135 Degree -->
18           <min_angle>-2.3562</min_angle> <!-- -135 Degree -->
19           <samples>1081</samples>
20         </horizontal>
21       </scan>
22       <range>
23         <min>0.15</min>
24         <max>20</max>
25         <resolution>0.04</resolution>
26       </range>
27     </ray>
28     <always_on>1</always_on>
29     <update_rate>40</update_rate>
30     <visualize>true</visualize>
31   </sensor>
32 </gazebo>
33 [...]
```

Anhand des Datenblattes des Hokuyo UST-20LX (siehe Anhang Abschnitt 8.1.2) können die Parameter ermittelt werden, welche in die Konfigurationsdatei einzutragen sind. Der Laserscanner hat einen Scanbereich von -135° bis 135° . Dieser Wert ist in Radiant unter dem Punkt *min_angle* bzw. *max_angle* einzutragen. Der Punkt *samples* beschreibt die Anzahl der Abtastpunkte pro Messung. Dieser Wert liegt bei dem Hokuyo UST-20LX bei 1081. Eine weitere wichtige Angabe ist die Wiederholrate von einzelnen Messungen. Diese wird unter dem Punkt *update_rate* eingetragen und beträgt 40Hz. Dem Plugin kann zusätzlich unter dem Punkt *mesh filename* ein 3D-Objekt zugewiesen werden. Dieses wird dann im Simulator angezeigt.

Servo Plugin

Für einen Servomotor ist noch kein Plugin in dem RotorS Simulator vorhanden. Es muss daher ein entsprechender ROS-Node entwickelt werden. Das Plugin führt dabei hauptsächlich die folgende Funktion aus:

```

1  [...]
2  this->ROS_subscriber = this->n.subscribe("/neo_servo_states", 10, &ModelPush::
    ROSCallback, this);
3  [...]
4  public: void ROSCallback(const sensor_msgs::JointState::ConstPtr& data)
5  {
6      this->jointBottom->SetAngle(0, data->position[1]);
7  }
8  [...]
```

Es wird zunächst mit der Funktion *ROSCallback* geprüft, ob auf dem Topic mit dem Namen *neo_servo_states*, Informationen zur Verfügung stehen. Es handelt sich dabei um den Winkel, welcher von dem Servomotor angefahren werden soll. Diese Informationen können z.B. direkt von dem Flugsystem stammen oder für Testzwecke von einem separaten ROS-Node erzeugt werden. Sind Daten verfügbar, wird *jointBottom* auf den entsprechenden Winkel gesetzt. Die Variable *jointBottom* enthält den Namen des Objektes, welches im RotorS Simulator rotiert werden soll. Anschließend können wieder die entsprechenden Konfigurationsdateien erstellt werden. Der Servomotor besteht dabei aus zwei Teilen. Der erste Teil ist der Motor selber und der zweite Teil ist der Hebelarm, welcher an dem rotierenden Teil des Servomotors befestigt ist. Der Servomotor ist daher fest mit dem Flugsystem verbunden und nur der Hebelarm wird mithilfe des Servo Plugins rotiert. Dazu wird in einer Konfigurationsdatei die entsprechenden Abhängigkeiten zwischen den verwendeten Komponenten definiert. Ein Ausschnitt dieser Datei ist im Folgenden dargestellt:

```

1  [...]
2  <!--Scanner Top-->
3
4  <xacro:laser
5      parent_link="base_link"
6      device_name="hokuyoTop">
7      <origin xyz="0.0 0.0 0.12" rpy="0 0 0" />
8  </xacro:laser>
9
10 <!--Scanner Bottom-->
11
12 <xacro:servo
13     parent_link="base_link"
14     device_name="servoBottom">
15     <origin xyz="0.0 0.0 -0.08" rpy="3.14159 0 1.5707" />
16 </xacro:servo>
17
18 <xacro:servo_lever
19     parent_link=${servoBottom_link}
20     device_name="leverBottom">
21     <origin xyz="0.0 0.0 0.0" rpy="0 -1.5707 3.141592" />
22 </xacro:servo_lever>
23
24 <xacro:laser
25     parent_link="leverBottom_link"
26     device_name="hokuyoBottom">
27     <origin xyz="0.026 0.0 0.0" rpy="0 0 0" />
28 </xacro:laser>
29 [...]
```

Der obere Laserscanner ist fest mit dem Flugsystem verbunden. Mit dem Parameter *parent_link* wird festgelegt, an welchem Objekt der Scanner befestigt ist. Der untere Laserscanner hingegen ist nicht direkt mit dem Flugsystem verbunden. Es wird also zunächst der Laserscanner mit dem Hebelarm (*servo_lever*) verbunden und dieser wiederum an den Servomotor (*servo*). Anschließend wird der Servomotor mit dem Flugsystem selbst verbunden.

5.4 Inbetriebnahme verwendeter Hardwarekomponenten

Im Folgenden wird auf die Inbetriebnahme der verwendeten Hardwarekomponenten eingegangen, welche an dem Flugsystem *Asctec Neo* angebracht sind.

5.4.1 Hokuyo UST-20LX

Die Datenübertragung der beiden Laserscanner vom Typ Hokuyo UST-20LX findet über Ethernet statt. Diese werden daher über einen Switch mit dem Rechner verbunden, welcher an der Asctec Neo angebracht ist. Die IP-Adresse beide Scanner ist zunächst auf eine Standardadresse konfiguriert. Um diese Adresse ändern zu können, wird jeder Scanner zunächst einzeln mit einem Linux Rechner verbunden und der folgende Befehl ausgeführt:


```
roslaunch urg_node set_urg_ip.py 192.168.0.11 0.0.0.0
```

Mit diesem Befehl wird die Standard IP-Adresse des Laserscanners auf den Wert *192.168.0.11* gesetzt. Dadurch wird ermöglicht, dass mehrere Laserscanner an einem Rechner betrieben werden können. Über ein ROS launch-file können weitere Parameter, wie der Name des Scanners oder der Scanbereich, eingestellt werden [27]. Die Versorgungsspannung der beiden Scanner liegt zwischen $10V - 30V$. Der Stromverbrauch von einem Laserscanner beträgt beim Start bis zu $450mA$. Bei normalem Betrieb liegt der Stromverbrauch bei etwa $150mA$. Zur Stromversorgung verfügt das Flugsystem *Asctec Neo* über verschiedene Anschlussmöglichkeiten (siehe Abbildung 5.5). Zum Betrieb der beiden Laserscanner werden die $12V/2A$

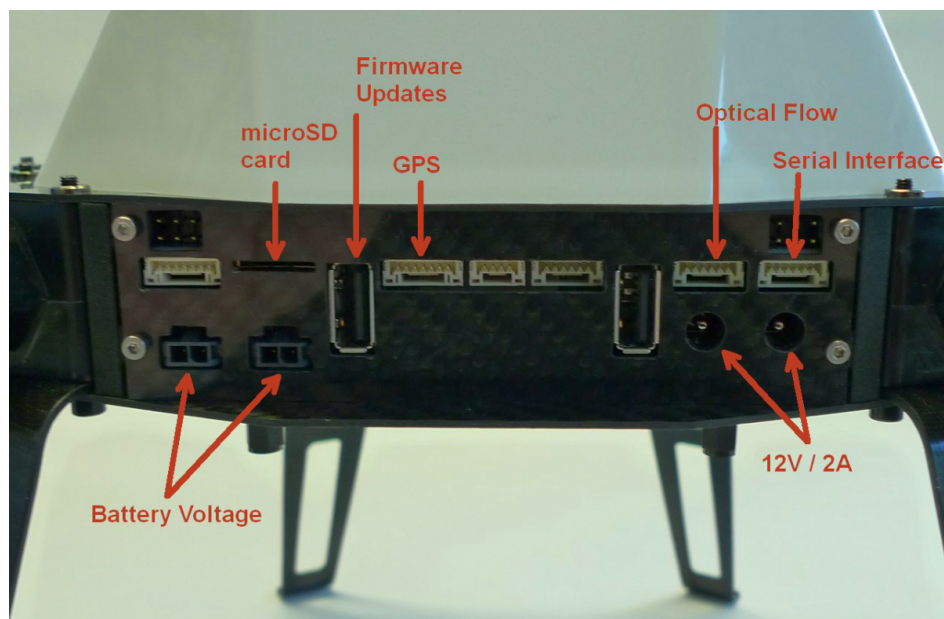


Abbildung 5.5: Anschlussmöglichkeiten externer Komponenten an dem Flugsystem Asctec Neo

Anschlüsse verwendet.

5.4.2 Savöx Servomotor

Um den Laserscanner, welcher an der Unterseite des Flugsystems angebracht ist, rotieren zu können, wird ein Servomotor der Firma *Savöx* verwendet. Dieser Servomotor zeichnet sich durch ein hohes Stellmoment von $70Ncm$ sowie einer geringen Bauhöhe von $25,4mm$ aus. Zur Ansteuerung eines Servomotors wird ein PWM (Pulsweitenmodulations (PWMs)) Signal benötigt. Die Periodendauer des Signals beträgt dabei $20ms$. Der Winkel des Servomotors wird über die Dauer des High-Pegels bestimmt. Also die Zeit, in der das PWM Signal einen Wert von über null Volt annimmt. Abbildung 5.6 zeigt ein Steuersignal für Standard Servomotoren und die daraus resultierende Hebelstellung.

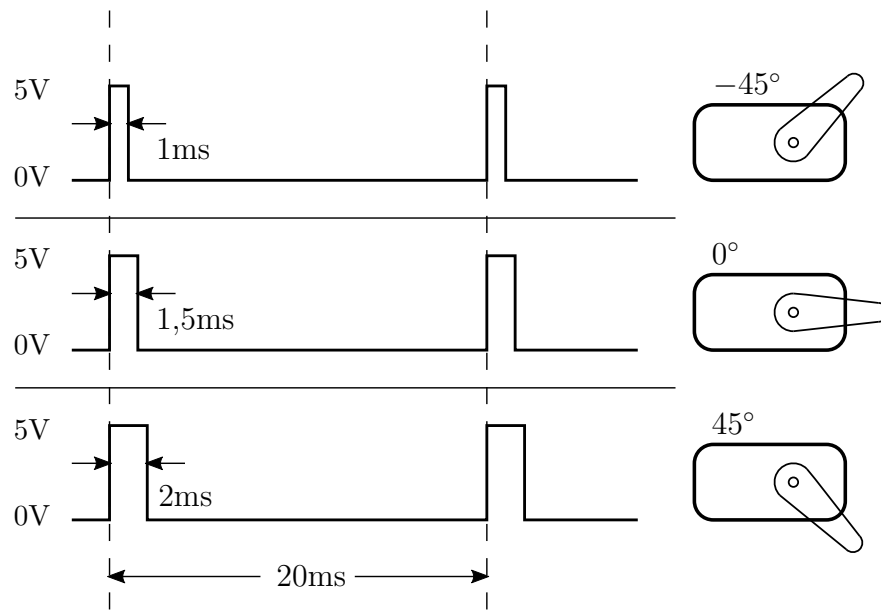


Abbildung 5.6: Steuersignal für Standard Servomotoren und die daraus resultierende Hebelstellung

Um nun einen bestimmten Winkel mit dem Servomotor anzusteuern, wird die Dauer des High-Pegels auf einen Wert zwischen 1ms und 2ms eingestellt. Damit kann ein Winkel zwischen -45° und 45° angesteuert werden. Unter Verwendung des SDK von Ascending Technologies (siehe Abschnitt 4.2.1), wird mithilfe des Mikrocontrollers auf der Asctec Neo ein Steuersignal für einen Servomotor erzeugt. An der Vorderseite der *Asctec Neo* befinden sich Anschlüsse, um einen Servomotor zu verbinden und so das generierte Signal entsprechend zu nutzen (siehe Abbildung 5.5).

6 Ergebnisse und Evaluierung

Im folgenden Abschnitt wird der in Kapitel 3 entwickelte Positionsregler sowie die in Kapitel 5 implementierte Software evaluiert. Dafür werden zunächst unterschiedliche Versuche im RotorS Simulator durchgeführt. Für die Evaluierung wird ein Hexacopter mit der Bezeichnung *Asctec Neo* verwendet.

6.1 Simulation in RotorS Simulator

Damit der entwickelte Positionsregler ordnungsgemäß funktioniert, ist es erforderlich, dass alle in Abschnitt 3.1.2 definierten Zustandsgrößen zur Verfügung stehen. Die zwölf Zustandsgrößen beinhalten die Translationen und die linearen Geschwindigkeiten in x -, y -, und z -Richtung sowie die Rotationen und die Rotationsgeschwindigkeiten um die x -, y - und z -Achse. Diese Angaben werden direkt von dem RotorS Simulator zur Verfügung gestellt und können so für die ersten Versuche verwendet werden. In der Realität müssen diese Zustandsgrößen allerdings anders ermittelt werden. Im späteren Verlauf der Evaluation wird der Simulator daher so angepasst, dass die benötigten Zustandsgrößen mit Sensorik bzw. Verfahren ermittelt werden, welche auch in der Realität zur Verfügung stehen.

6.1.1 VTOL und Rotation

Der erste Versuch stellt das VTOL-Verhalten des Flugsystems genauer dar. VTOL bedeutet Vertical Take-Off and Landing und soll hier zeigen, wie sich die Asctec Neo bei einem Start oder einer Landung verhält. Zusätzlich wird die Bewegung des Flugsystems bei einer Drehung um die z -Achse betrachtet. In Abbildung 6.1 ist das Verhalten der Asctec Neo bei einem Steig- und Sinkflug abgebildet. Zusätzlich zu der Höhe wird auch die Position in x - und y -Richtung sowie die Rotation um die z -Achse dargestellt. Die in dem Diagramm dargestellte Höhe gibt die Entfernung vom Boden zum Mittelpunkt des Flugsystems an. Durch das Landegestell der Asctec Neo befindet sich der Mittelpunkt des MAVs bereits zu Anfang ein paar Zentimeter über dem Boden. Daher wird in dem Diagramm in Abbildung 6.1 bereits zu Anfang eine Höhe angegeben, welche einen Wert größer null aufweist. Nach etwas weniger als fünf Sekunden beginnt das Flugsystem auf einen Wert von einem Meter anzusteigen und erreicht diesen nach etwa drei Sekunden. Bei dem Steigflug bewegt sich das Flugsystem nur minimal in eine andere translatorische Richtung. Die Rotation um die z -Achse ist auch sehr gering und steigt knapp über $0,01rad$, was einer Drehung von etwa $0,6^\circ$ entspricht. Zum Zeitpunkt $t = 13,8s$ beginnt der Sinkflug, welcher nach etwa zwei Sekunden beendet ist. Die Bewegung in eine andere translatorische Richtung ist noch geringer als bei dem zuvor ausgeführten Steigflug. Das Flugsystem rotiert dabei um ca. $0,017rad$ ($-0,97^\circ$).

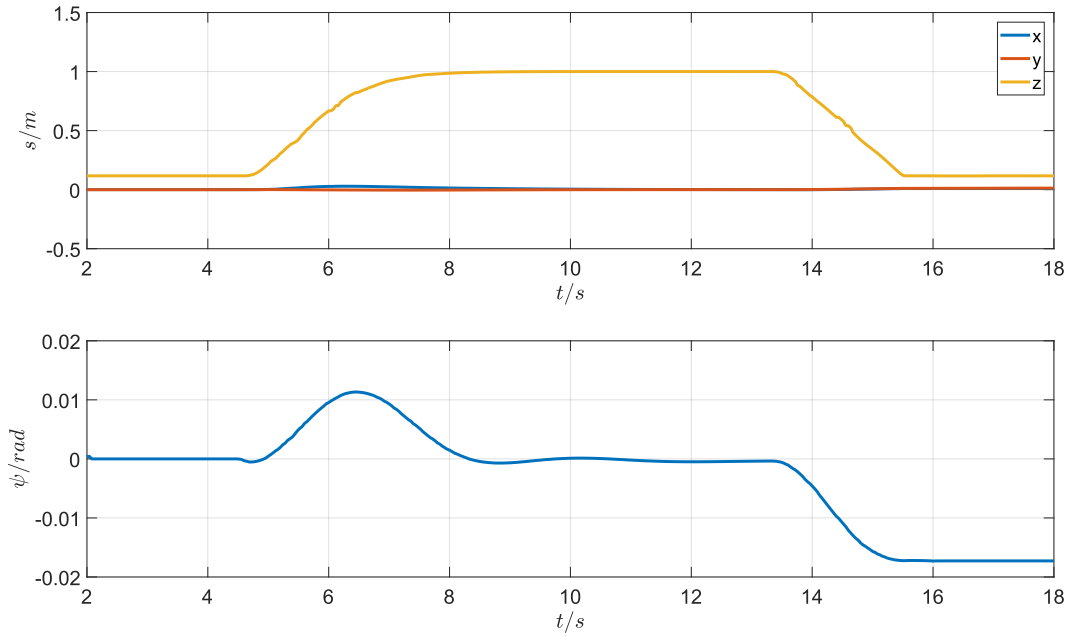
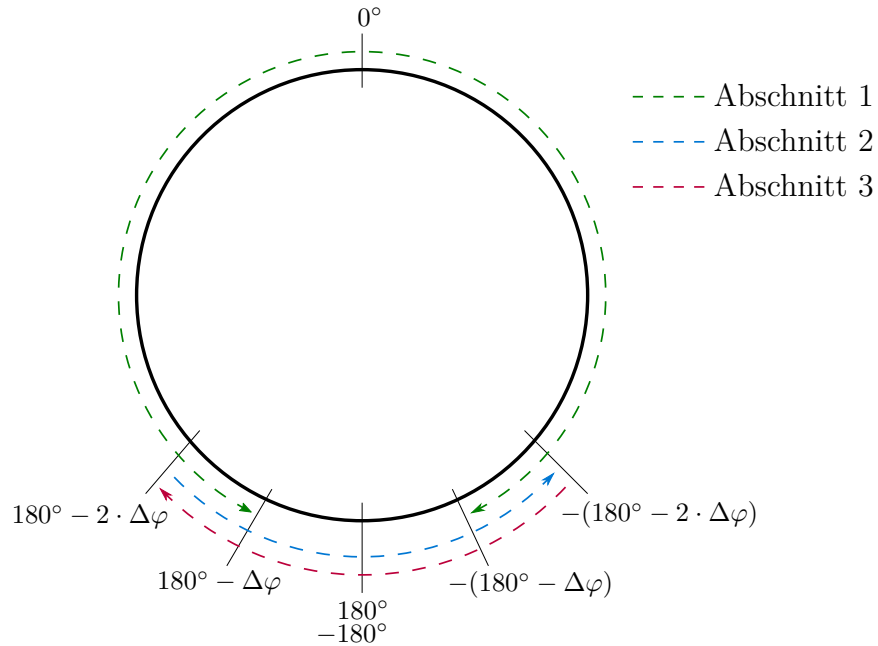


Abbildung 6.1: Verhalten des Flugsystems Asctec Neo bei einem Steig- und Sinkflug

Ein weiterer Versuch soll zeigen, wie sich die Asctec Neo bei einer größeren Rotation um die z -Achse verhält. Dabei ist zu beachten, dass die aktuelle Rotation des Flugsystems um die z -Achse von dem Simulator in einem Bereich von -180° bis 180° übergeben wird. Dreht sich also das MAV um mehr als 180° um die z -Achse, ergibt sich somit im Übergangsbereich ein Sprung von 360° . Der Rotationswinkel um die z -Achse stellt eine der zwölf Zustandsgrößen des Positionsreglers dar. Diese Zustandsgrößen sind, wie in Abschnitt 3.2 beschrieben, über Rückföhrungsfaktoren mit den Stellgrößen, bzw. den Rotordrehzahlen des Flugsystems fest gekoppelt. Verändert sich nun eine Zustandsgröße über einen größeren Wert, ergibt sich somit auch ein Sprung der Rotordrehzahlen, was zu einem Absturz des Flugsystems föhren kann. Zur Vermeidung einer Sprungstelle bei dem Übergangspunkt von 180° zu -180° wird die Rotation um die z -Achse in drei Abschnitte eingeteilt. Diese sind in Abbildung 6.2 dargestellt. Dreht sich das Flugsystem in positiver Richtung wechselt der Abschnitt von eins nach zwei bei $180^\circ - \Delta\varphi$. Ein Wechsel von Abschnitt zwei zurück nach Abschnitt eins hingegen findet erst bei $180^\circ - 2 \cdot \Delta\varphi$ statt. Dadurch ergibt sich eine Hysterese mit der Größe $\Delta\varphi$. Durch diese Hysterese wird erreicht, dass die Sprungstelle des Rotationswinkels um die z -Achse nicht nur an einer Stelle liegt, sondern abhängig davon ist, von welcher Seite der Übergang von 180° nach -180° stattfindet. Beträgt die Rotation des Flugsystems etwa 180° wird durch die Hysterese auch verhindert, dass bereits bei kleinen Drehungen des Flugsystems sich der gemessene Winkel um einen größeren Wert ändert. Es ergeben sich allerdings weiterhin Sprungstellen, wenn ein Übergang in einen anderen Abschnitt stattfindet. Diese müssen abgefangen werden, um einen Absturz des Flugsystems zu vermeiden.


 Abbildung 6.2: Aufteilung des Rotationswinkels um die z -Achse in drei Abschnitte

Hierfür wird eine Funktion entwickelt, um den Zustandsregler zurückzusetzen. Damit auch hier die Zustandsgrößen nicht springen, muss gewährleistet werden, dass sich der Wert der Stellgrößen im Moment des Zurücksetzens des Zustandsreglers nicht ändern. Dafür wird der Zustandsregler um eine Reset-Funktion erweitert. Diese ist in Abbildung 6.3 in Form eines Signalflussplanes dargestellt. Wird nun ein Reset durchgeführt, werden zunächst die aktuellen

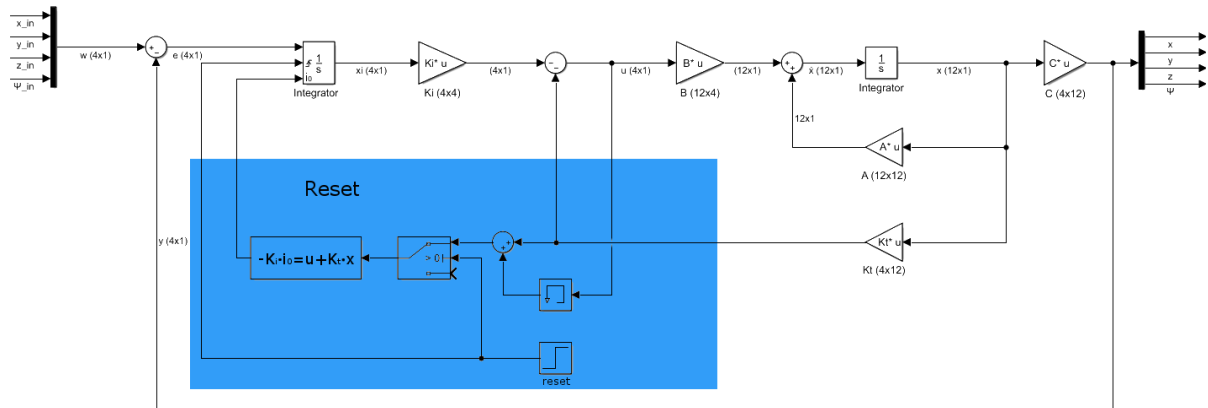


Abbildung 6.3: Signalflussplan eines Zustandsreglers mit Reset-Funktion

Rückführungsgrößen ausgelesen. Diese ergeben sich durch die Multiplikation der Zustandsgrößen \vec{x} und den Rückföhrungsfaktoren \mathbf{K}_t . Die Rückföhrungsgrößen werden anschließend mit den Stellgrößen addiert. Daraufhin wird der Integrator im StellgröÖenpfad zurückgesetzt und mit definierten Werten initialisiert. Diese Werte werden mit \vec{i}_0 bezeichnet und ergeben

sich aus dem folgenden Gleichungssystem:

$$-\mathbf{K}_i^{4 \times 4} \cdot \vec{i}_0 = \vec{u} + \mathbf{K}_t^{4 \times 12} \cdot \vec{x} \quad (6.1)$$

Dieses Gleichungssystem beinhalten vier Gleichungen mit vier Unbekannten. Die Werte für \vec{i}_0 können somit eindeutig ermittelt werden. Die Dauer eines Resets beträgt genau ein Rechendurchgang. Arbeitet also der Regler z.B. mit einer Geschwindigkeit von $100Hz$, benötigt ein Reset des Reglers 10 Millisekunden und die Stellgröße ändert sich für diese Zeit nicht. Durch die geringe Dauer eines Resets wird das Flugsystem in der Simulation nicht merkbar beeinflusst.

Im folgenden Versuch wird dem Flugsystem eine Rotation um die z -Achse vorgegeben. Die Ergebnisse sind in Abbildung 6.4 dargestellt. Zunächst steigt das Flugsystem wieder auf eine

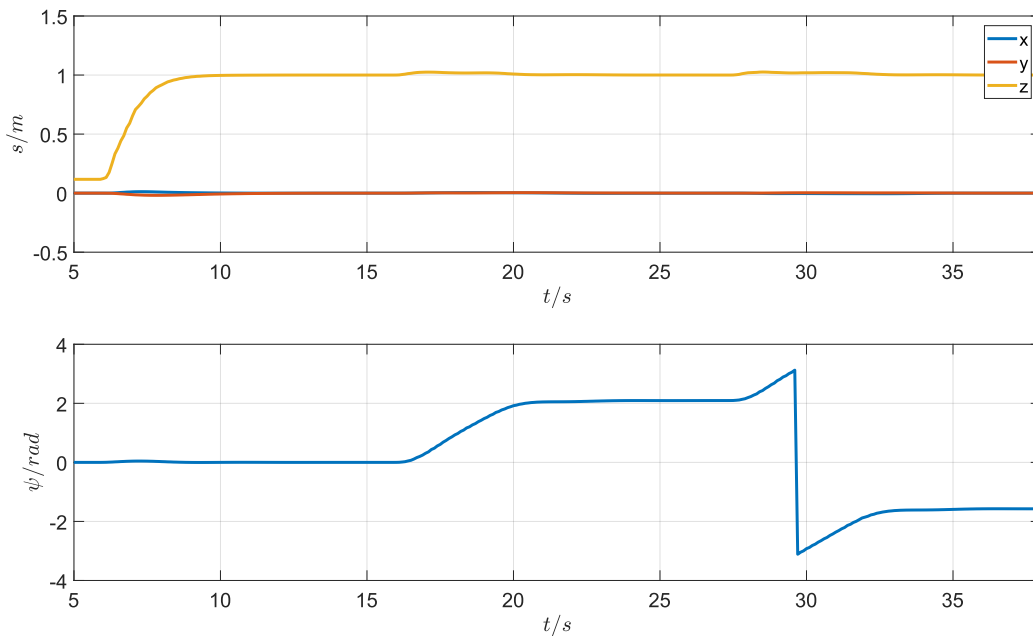


Abbildung 6.4: Verhalten des Flugsystems Asctec Neo bei Rotation um die z -Achse

Höhe von zwei Meter. Nach etwa 16 Sekunden wird eine Drehung in positiver Richtung um $2,1rad$ (120°) durchgeführt. Der Winkel wird in unter vier Sekunden erreicht. Zum Zeitpunkt $t = 18s$ wird ein weiterer Wert von $\varphi = -1,57rad$ (-90°) vorgegeben. Es ist zum Zeitpunkt $t = 29,5s$ ein deutlicher Sprung zu erkennen. Dieser Sprung resultiert aus der Messweise des entsprechenden Sensors, da dieser den Rotationswinkel im Bereich von $-\pi$ (-180°) bis π (180°) ausgibt und das Flugsystem bei einer Rotation von $2,1rad$ (120°) nach $-1,57rad$ (-90°) diesen Punkt überschreitet. Damit dieser Sprung keinen Effekt auf die Stellgröße hat, wird diese Sprungstelle wie in Abbildung 6.2 verändert und die in Abbildung 6.3 dargestellte Reset-Funktion ausgeführt. Werden die Translationen in Abbildung 6.4 an der Sprungstelle

betrachtet, ist auch hier zu erkennen, dass der Sprung des Rotationswinkels keinen Einfluss auf die aktuelle Position des Flugsystems hat.

6.1.2 Ansteuerung vorgegebener Wegpunkte

Im Folgenden werden dem Flugsystem unterschiedliche Wegpunkte vorgegeben, um das Flugverhalten genauer zu untersuchen. Es können dabei die Translationen in x - y - und z -Richtung sowie die Rotation um die z -Achse vorgegeben werden. Ein Wegpunkt ist somit definiert durch $P = (x, y, z, \psi)$. Für die einzelnen Versuche wird zunächst eine virtuelle Umgebung in dem RotorS Simulator aufgebaut. Diese ist in Abbildung 6.5 dargestellt.

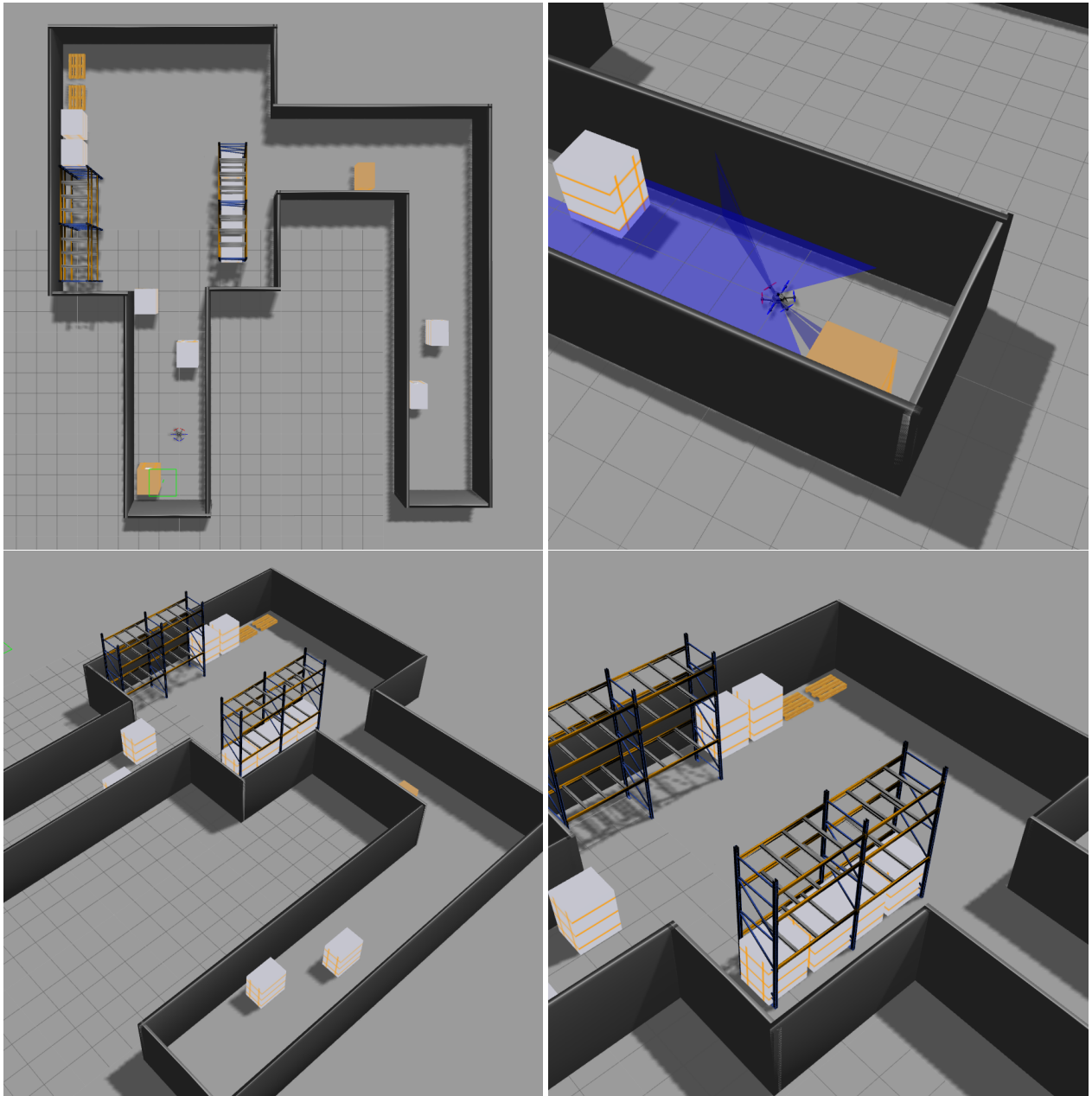


Abbildung 6.5: Testumgebung für Versuche im RotorS Simulator

In der Testumgebung sind mehrere Hindernisse sowie Abzweigungen vorhanden. Dadurch kann das Flugsystem nicht jeden Punkt auf direktem Weg erreichen und es ist somit oft notwendig, einen Pfad zum vorgegebenen Ziel zu bestimmen und abzufliegen. Das Flugsystem wird bereits in der Simulation mit zwei Laserscannern ausgestattet. Diese werden so angebracht wie es in Abschnitt 4.2.3 beschrieben ist. Der Scanbereich ist im Simulator an der leicht transparenten blauen Fläche zu erkennen.

manuelle Vorgabe von Wegpunkten

Dem Flugsystem wird zunächst ein Wegpunkt manuell vorgegeben. Dadurch kann untersucht werden, wie schnell und genau ein gegebenes Ziel erreicht wird. Wird ein Wegpunkt manuell vorgegeben, fliegt das MAV diesen auf direktem Wege an. Es muss somit darauf geachtet werden, dass kein Hindernis den Weg zum Zielpunkt versperrt. Die Ergebnisse sind in Abbildung 6.6 dargestellt. Auch hier steigt das Flugsystem zunächst auf eine Höhe von $z = 1m$ an. Zum Zeitpunkt $t = 14s$ wird eine Translation von $x = 3m$ und $y = 1,5m$ vorgegeben. Es ist zu erkennen, dass das Flugsystem zu diesem Zeitpunkt etwa 20 Zentimeter an Höhe verliert. Nach kurzer Zeit wird allerdings wieder die vorgegebene Höhe von $z = 1m$ erreicht. Dieses Verhalten hat folgenden Grund. Während sich das Flugsystem in eine Richtung bewegt, ergibt sich eine Neigung des MAVs um die x - oder y -Achse. Dadurch wirkt nicht mehr der gesamte Schub in z -Richtung. Um nun nicht an Höhe zu verlieren, muss also der Gesamtschub in einem bestimmten Verhältnis zur Neigung erhöht werden. Es benötigt allerdings etwas Zeit, um den richtigen Wert für den Schub zu bestimmen und das Flugsystem sinkt daher kurzzeitig um ein paar Zentimeter ab.

Nach etwa fünf Sekunden erreicht das Flugsystem den vorgegebenen Wert in y -Richtung und innerhalb von ca. 12 Sekunden den Wert in x -Richtung. Das Flugsystem erreicht den Sollwert mit einer hohen Genauigkeit und bleibt an dem Zielpunkt stehen, bis ein neuer Wert vorgegeben wird. Auch die Rotation um die z -Achse ist nur minimal. Diese steigt nicht über einen Wert von $0,05rad$ ($2,9^\circ$).

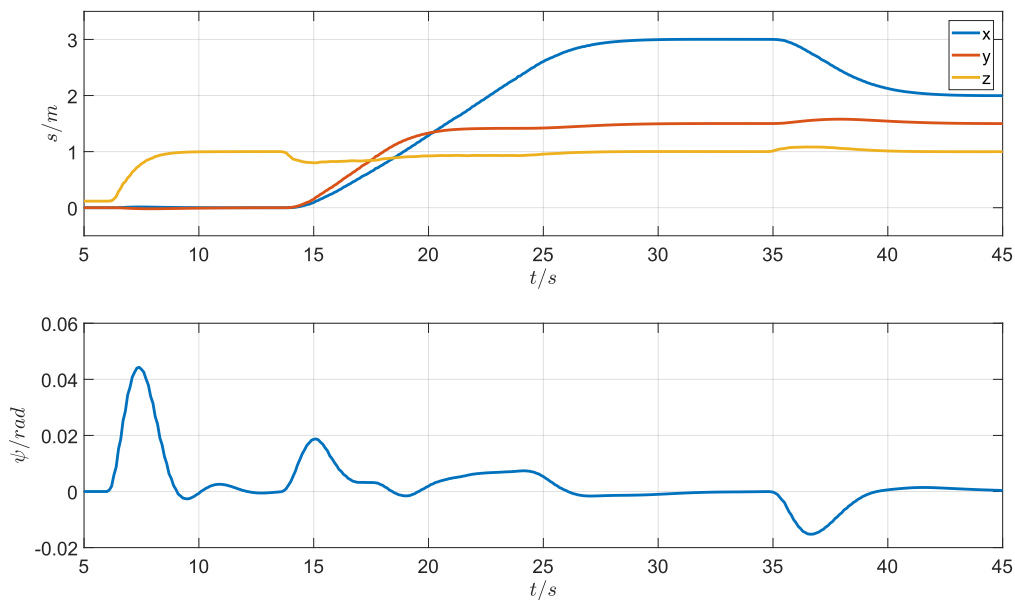


Abbildung 6.6: Verhalten des Flugsystems Asctec Neo bei einer Bewegung in translatorischer Richtung

Wegpunktvorgabe durch Trajektorienregler in bekannter Umgebung

Da das Flugsystem manuell vorgegebene Wegpunkte mit einer hohen Genauigkeit ansteuert, werden nun im nächsten Versuch mehrere Wegpunkte hintereinander an das MAV übergeben. Dazu wird der in 5.2 beschriebene Trajektorienregler verwendet. Es wird zunächst davon ausgegangen, dass bereits eine Karte der Umgebung vorhanden ist. Diese kann z.B. von einem Bodenroboter stammen, welcher bereits durch das Gebiet gefahren ist oder von einem anderen Flugsystem, welches manuell gesteuert wurde. Um nun ein Ziel vorzugeben, wird die Karte der Umgebung in dem Visualisierungsprogramm *RViz* angezeigt. Anschließend wird an eine beliebige Stelle in der Karte geklickt (siehe dazu auch Abschnitt 5.2). Daraufhin wird ein Pfad zum Zielpunkt berechnet und das Flugsystem beginnt, die einzelnen Wegpunkte anzusteuern. In Abbildung 6.7 ist dieser Vorgang dargestellt.

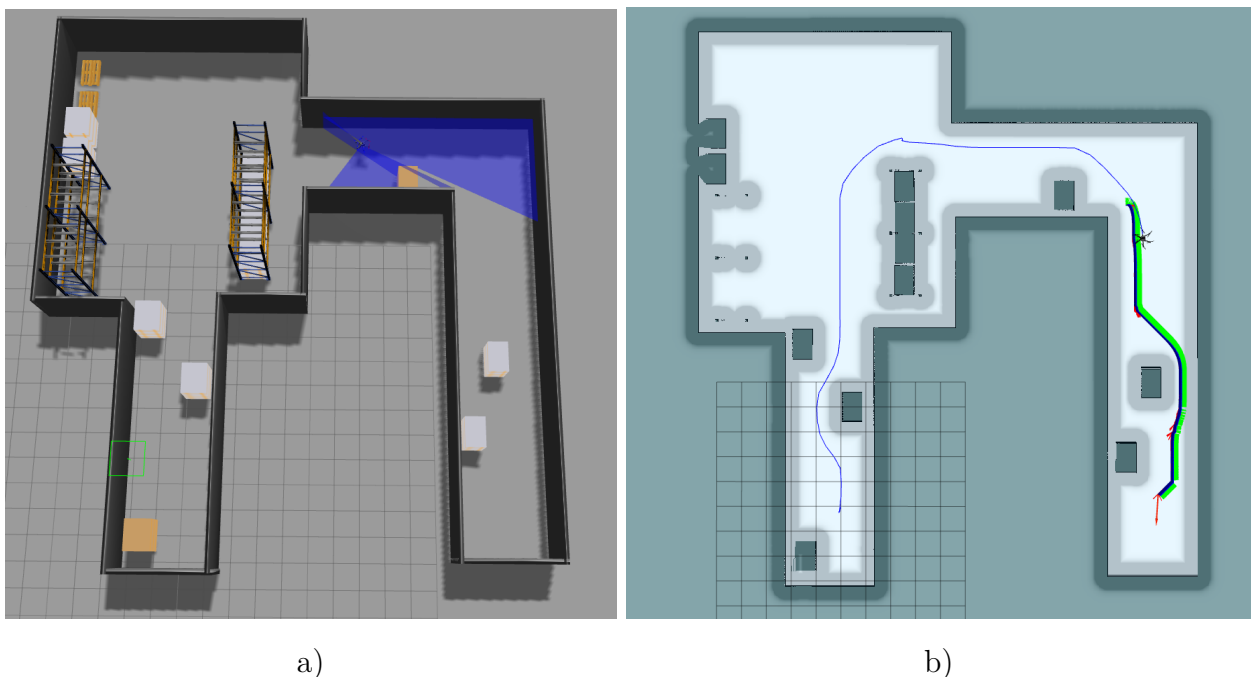


Abbildung 6.7: Vorgabe einer Zielkoordinate in *RViz* mit anschließender Ansteuerung vorgegebener Wegpunkte

In Abbildung 6.7 b) ist die Oberfläche von *RViz* zu sehen. Die dünne Blaue Linie gibt den bisher zurückgelegten Weg des Flugsystems an und die grüne Linie stellt den restlichen Weg zum Ziel dar. Es ist zu erkennen, dass ein Pfad zum Zielpunkt berechnet wurde und das Flugsystem erfolgreich die Hindernisse umflogen hat. Existiert also bereits eine Karte der Umgebung, kann zuverlässig ein beliebiger Punkt angesteuert werden. Am Rand von Hindernissen ist ein breiterer leicht transparenter grauer Rand zu erkennen. Durch diesen Rand wird ein Sicherheitsabstand definiert, in dem kein Pfad geplant werden soll. Dadurch wird verhindert, dass das Flugsystem zu nahe an ein Hindernis fliegt.

Zusätzlich ist es möglich in *RViz* die Scanbereiche der simulierten Laserscanner darzustellen.

Dadurch können einige Objekte wie Kisten oder Wände erkannt werden. In Abbildung 6.8 a) ist das Flugsystem *Asctec Neo* in dem RotorS Simulator dargestellt und Abbildung 6.8 b) zeigt die entsprechenden Scandaten, welche von RViz angezeigt werden.

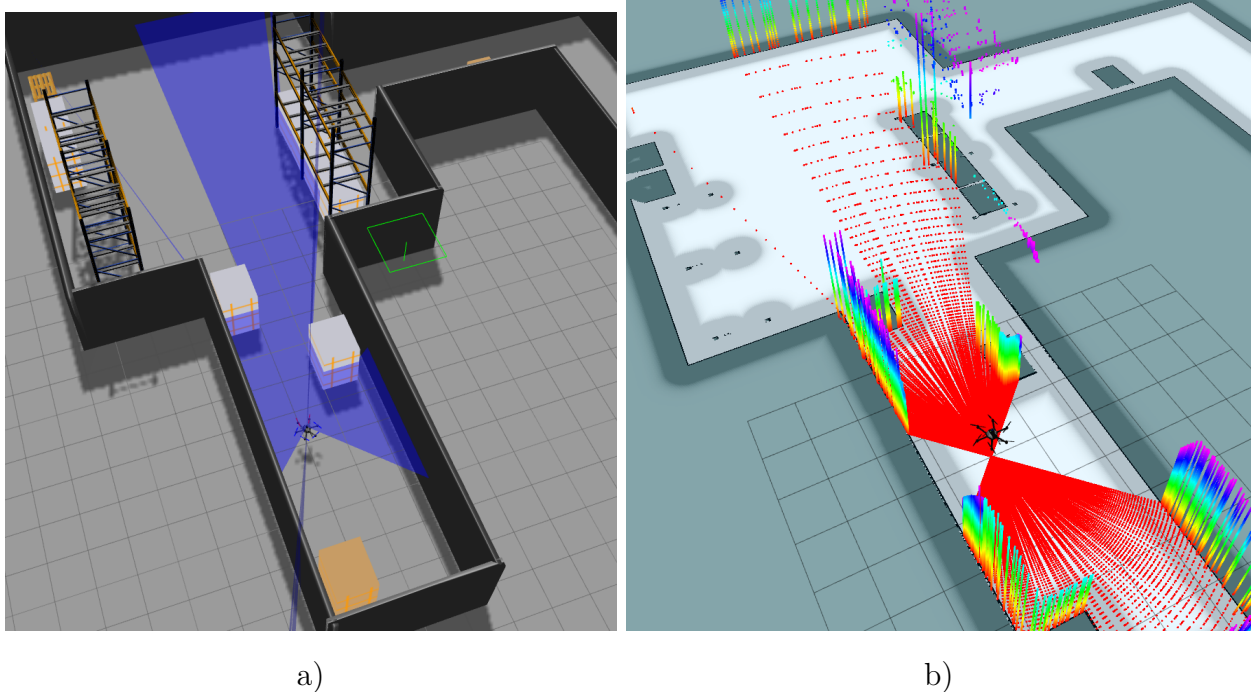


Abbildung 6.8: Darstellung des Flugsystems *Asctec Neo* in dem RotorS Simulator und den resultierenden Laserscandaten in RViz

Da der untere Laserscanner wie in Abschnitt 4.2.3 beschrieben rotiert, kann der Bereich unter dem Flugsystem detailliert dargestellt werden, auch wenn sich das MAV nicht bewegt. In RViz können deutlich die beiden Kisten erkannt werden, welche sich in der Nähe des Flugsystems befinden. Auch weiter entfernte Objekte wie ein Regal werden bereits teilweise vom Laserscanner erfasst.

Die Genauigkeit mit der die einzelnen Wegpunkte angesteuert werden, ist in Abbildung 6.9 zu sehen. In Rot ist die vorgegebene Trajektorie dargestellt und in Blau die abgeflogene Trajektorie. Das Flugsystem folgt demnach sehr genau den Vorgabewerten. Die kleineren Abweichungen ergeben sich hauptsächlich daraus, dass dem Flugsystem immer ein Wegpunkt vorgegeben wird der etwas weiter entfernt ist, bevor der aktuelle Wegpunkt erreicht wird. Dieser Effekt macht sich besonders an Eckpunkten bemerkbar. An diesen Stellen steuert das Flugsystem oft einen direkteren Weg zum nächsten Wegpunkt an. Um diesen Effekt zu verringern, ist es möglich, dem MAV näher liegende Wegpunkte vorzugeben. Dadurch wird allerdings auch die Fluggeschwindigkeit verringert und der Abflug einer Trajektorie verlängert sich dementsprechend. Zusätzlich ist z.B. an der Stelle $x = 14m$, $y = -2m$ eine kleine Schwankung zu erkennen. An dieser Stelle rotiert das Flugsystem um die z -Achse, um sich zum Pfad auszurichten.

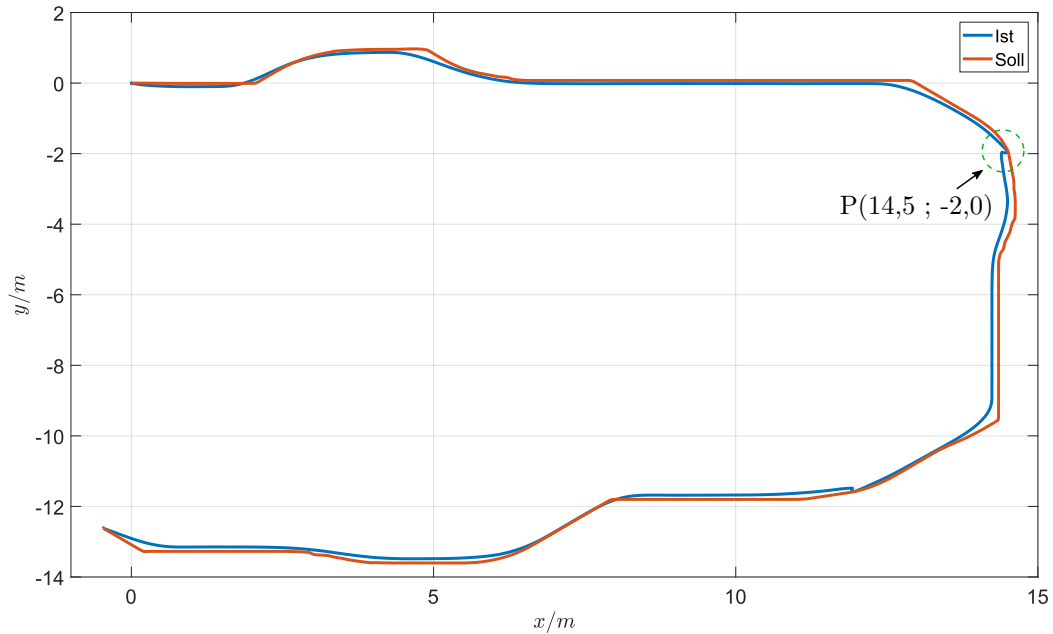


Abbildung 6.9: Darstellung der vorgegebenen Trajektorie im Vergleich zu den angesteuerten Wegpunkten. Rot → vorgegebene Trajektorie. Blau → abgeflogene Trajektorie

Wegpunktvorgabe durch Trajektorienregler in unbekannter Umgebung

In der Realität ist es oft der Fall, dass keine Karte der Umgebung zur Verfügung steht und diese erst erstellt werden muss. Im folgenden Versuch wird das Verhalten des Flugsystems geprüft, wenn noch keine Karte zur Verfügung steht. Dem MAV wird dazu ein ähnlicher Zielpunkt wie im vorherigen Versuch vorgegeben, um den abgeflogenen Pfad gut vergleichen zu können. Die Karte wird hierbei Schritt für Schritt mithilfe von dem Node *hector_mapping* (siehe Abschnitt 4.1.1) erstellt. Bewegt sich das Flugsystem zu nah an ein Hindernis, wird der Pfad erneut mit der aktuellen Karte geplant. Dadurch bewegt sich das Flugsystem immer weiter zum vorgegebenen Zielpunkt. In Abbildung 6.10 wird der aktuell abgeflogene Pfad zu unterschiedlichen Zeitpunkten dargestellt.

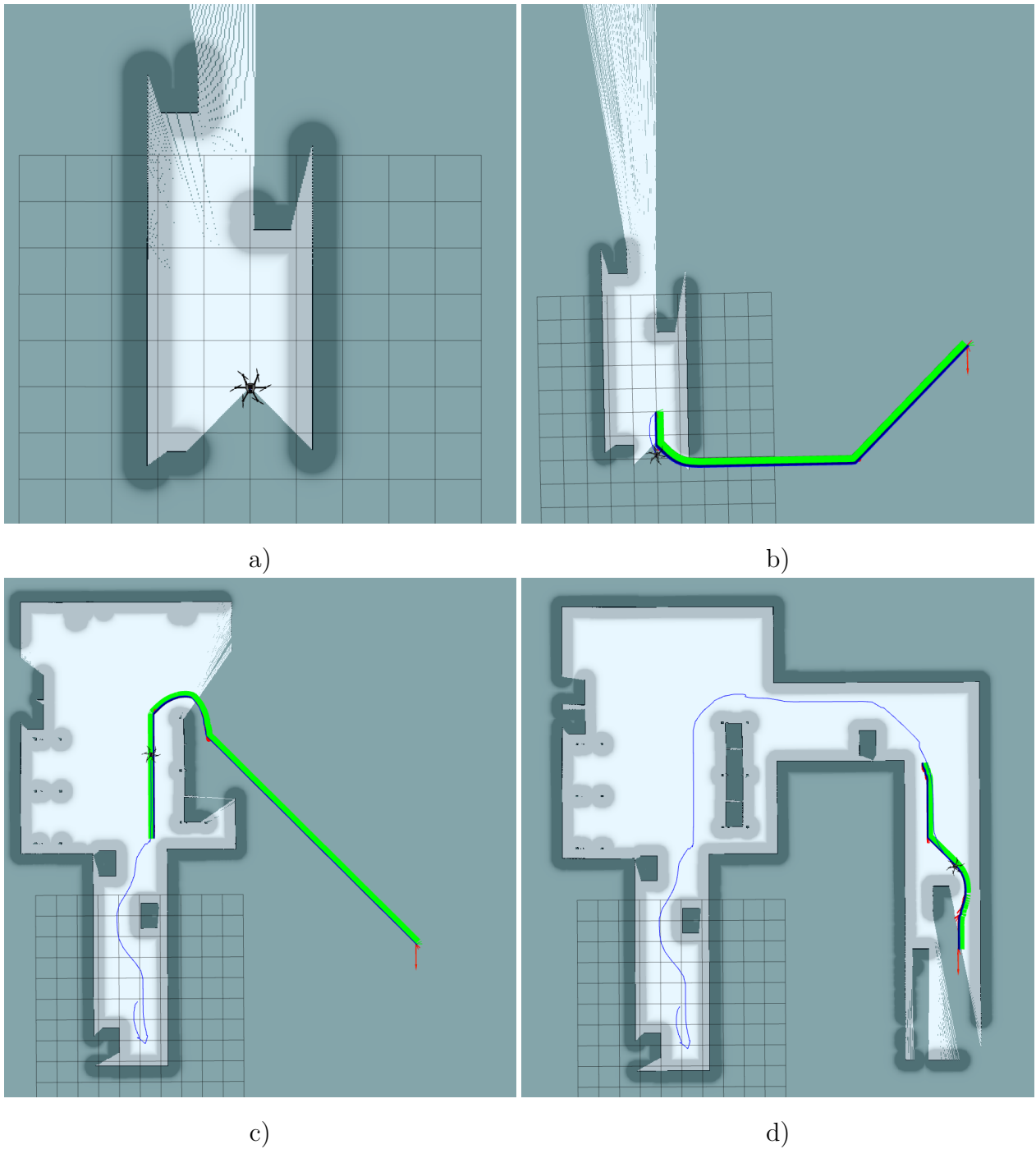


Abbildung 6.10: Vorgabe einer Zielkoordinate in RViz in einer unbekannten Umgebung mit anschließender Ansteuerung vorgegebener Wegpunkte

Zunächst steht nur eine Karte der lokalen Umgebung zur Verfügung, da der weitere Bereich nicht von dem Laserscanner erfasst werden kann (siehe Abbildung 6.10 a). Bei der Vorgabe einer Zielkoordinate wird zunächst versucht, diese möglichst direkt anzufliegen. Dieses

Verhalten ist in Abbildung 6.10 b) zu sehen. Es ist am Anfang nicht bekannt, dass sich hinter dem Flugsystem eine Wand befindet, da der Laserscanner diese nicht direkt erfasst. Daraufhin bewegt dreht sich das MAV und versucht, diesen Pfad abzufliegen. Sobald dieses Hindernis allerdings vom Scanner erfasst wird und das Flugsystem nah genug an der Wand ist, wird ein neuer Pfad bestimmt und das MAV beginnt in die richtige Richtung zu fliegen (siehe Abbildung 6.10 c). Das Flugsystem bewegt sich dadurch nach und nach immer näher zum Zielpunkt. Sind auf dem Weg keine unüberwindbaren Hindernisse, kann das Ziel, wie in Abbildung 6.10 d) dargestellt, erfolgreich erreicht werden.

Ansteuerung von Wegpunkten mit realitätsnaher Sensorik zur Positionsbestimmung

Bisher arbeitete der Positionsregler mit den Zustandsgrößen, welche direkt von dem RotorS Simulator zur Verfügung gestellt werden. Diese Größen geben sehr genau die Position und Bewegung des Flugsystems wieder und bieten sich daher sehr gut an, um die Funktionsweise des Positionsreglers testen zu können. Um jetzt allerdings die Simulation möglichst realitätsnah durchzuführen, werden die Zustandsgrößen nun von Sensorik zur Verfügung gestellt, welche auch in der Realität zur Verfügung stehen. Die einzelnen Messgrößen ergeben sich dabei wie in Abschnitt 3.2.5 dargestellt.

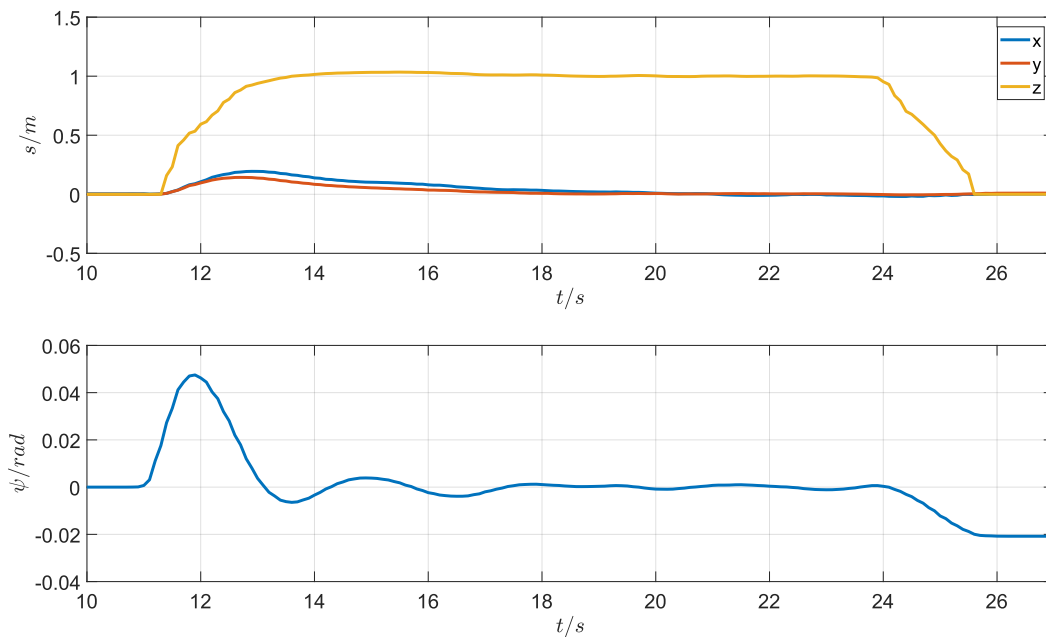


Abbildung 6.11: Verhalten des Flugsystems Asctec Neo bei einem Steig- und Sinkflug unter Verwendung realitätsnaher Sensorik

Um nun das Flugverhalten des MAVs mit realitätsnaher Sensorik zu testen, wird zunächst wieder das VTOL Verhalten untersucht. In Abbildung 6.11 ist das Ergebnis dargestellt. Im Vergleich zum vorherigen Versuch, welcher in Abbildung 6.1 dargestellt ist, ist das Ver-

halten des Flugsystems allgemein etwas unruhiger. Steigt das MAV auf eine Höhe von einen Meter an, findet auch eine Bewegung in x - und y -Richtung statt. Diese ist allerdings sehr gering und nach kurzer Zeit bewegt sich das Flugsystem wieder auf die vorgegebene Position. Auch die Rotation um die z -Achse ist etwas höher als im vorherigen Versuch (siehe Abschnitt 6.1.1). Die Abweichung steigt auf einen Wert von ca. 0.05rad ($2,86^\circ$). Diese Abweichungen lassen sich durch die Benutzung von der realitätsnahen Sensorik erklären. Diese ist nicht mehr so genau wie die Werte, welche direkt von dem RotorS Simulator zu Verfügung gestellt werden. Das Flugsystem braucht somit etwas länger, um den stationären Zustand anzunehmen.

Im nächsten Versuch wird dem MAV wieder ein Zielpunkt in einer bekannten Umgebung vorgegeben. Die Ergebnisse sind in Abbildung 6.12 dargestellt. Das Flugsystem folgt auch

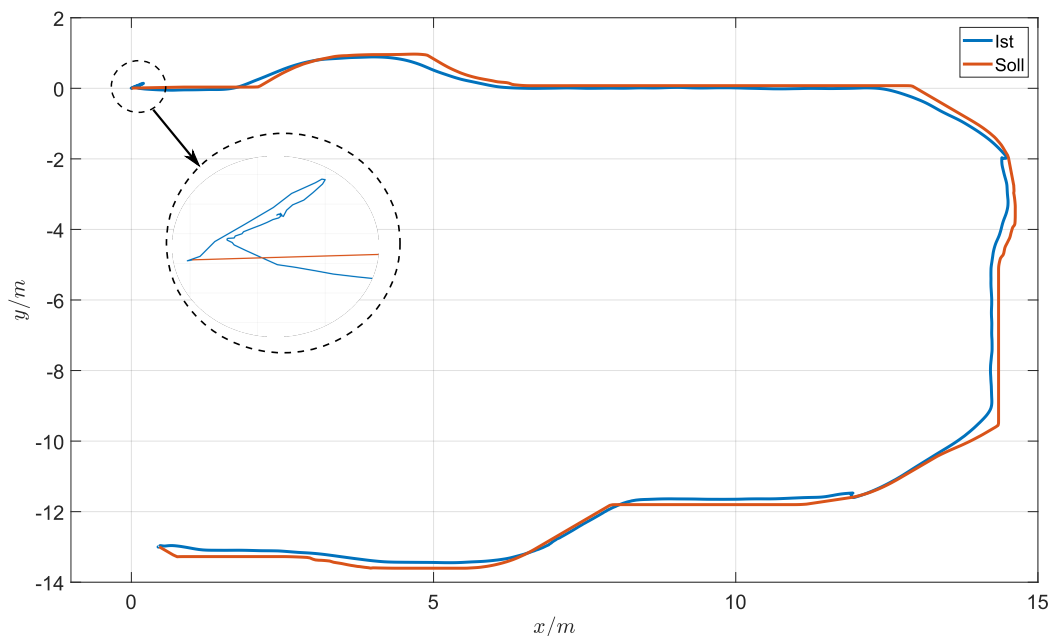


Abbildung 6.12: Darstellung der vorgegebenen Trajektorie im Vergleich zu den angesteuerten Wegpunkten unter Verwendung realitätsnaher Sensorik. Rot → vorgegebene Trajektorie. Blau → abgeflogene Trajektorie

bei Verwendung realitätsnaher Sensorik dem vorgegebenen Pfad mit einer hohen Genauigkeit. Lediglich am Anfang ist eine kleinere Abweichung zu erkennen. Dieses Verhalten hat sich auch bereits bei dem VTOL-Versuch in Abbildung 6.11 gezeigt und lässt sich auch hier durch die realitätsnahe Sensorik erklären, welche etwas ungenauer ist als die Werte, welche direkt von dem RotorS Simulator zu Verfügung gestellt werden.

6.2 Evaluation mit dem realen Flugsystem Asctec Neo

Das Flugsystem *Asctec Neo* verfügt bereits über einen Lageregler, welcher extern angesteuert werden kann. Daher wird aufbauend auf diesen Regler ein Positionsregler, wie in Abschnitt 3.2.6 beschrieben, verwendet.

6.2.1 VTOL

Auch in dem praktischen Versuch wird zunächst, wie in der Simulation, geprüft, wie sich die *Asctec Neo* bei einem Start und bei einer Landung verhält. Die Ergebnisse sind in Abbildung 6.13 dargestellt. Im Vergleich zu den Simulationsergebnisse, sind deutliche Unterschiede zu

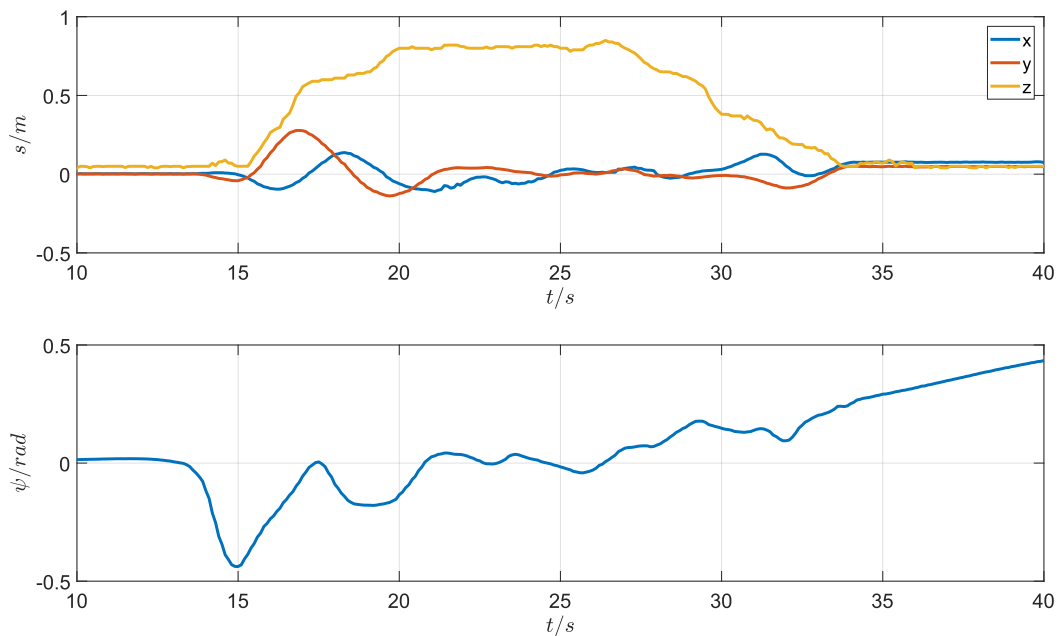


Abbildung 6.13: VTOL-Verhalten der *Asctec Neo* in einer realen Umgebung

erkennen. Zu Beginn wird dem Flugsystem eine Höhe von $z = 0,8\text{m}$ vorgegeben. Die *Asctec Neo* bewegt sich beim Start zunächst um ein paar Zentimeter in x - und y -Richtung. Nach ein kurzer Zeit bewegt sich das Flugsystem wieder auf den Sollwert von $x = 0,0$ und $y = 0,0$. Allgemein ist das Verhalten allerdings etwas unruhiger als in der Simulation. Auch bei der Rotation um die z -Achse ergeben sich besonders bei dem Steig- und Sinkflug der *Asctec Neo* etwas höhere Abweichungen. Das Flugsystem rotiert um etwa $0,43\text{rad}$ ($24,6^\circ$). Die Abweichungen und das etwas unruhigere Verhalten ergeben sich hauptsächlich durch Windverwirbelungen, welche durch die Rotoren des Flugsystems entstehen. Besonders, wenn sich das Flugsystem nah an einer Wand befindet, ergibt sich ein unruhigeres Flugverhalten. Dieses Verhalten wird auch in den folgenden Versuchen noch deutlicher.

6.2.2 Vorgabe von Wegpunkten

Der Asctec Neo wird nun ein Wegpunkt vorgegeben, welcher automatisch angesteuert werden soll. Das Flugsystem befindet sich zunächst in einem geschlossenen Raum mit einem Abstand von mindestens 1,2 Metern zur Wand. In Abbildung 6.14 ist das resultierende Verhalten dargestellt. Das Flugsystem steigt zunächst wieder auf eine Höhe von etwa 0,8 Meter

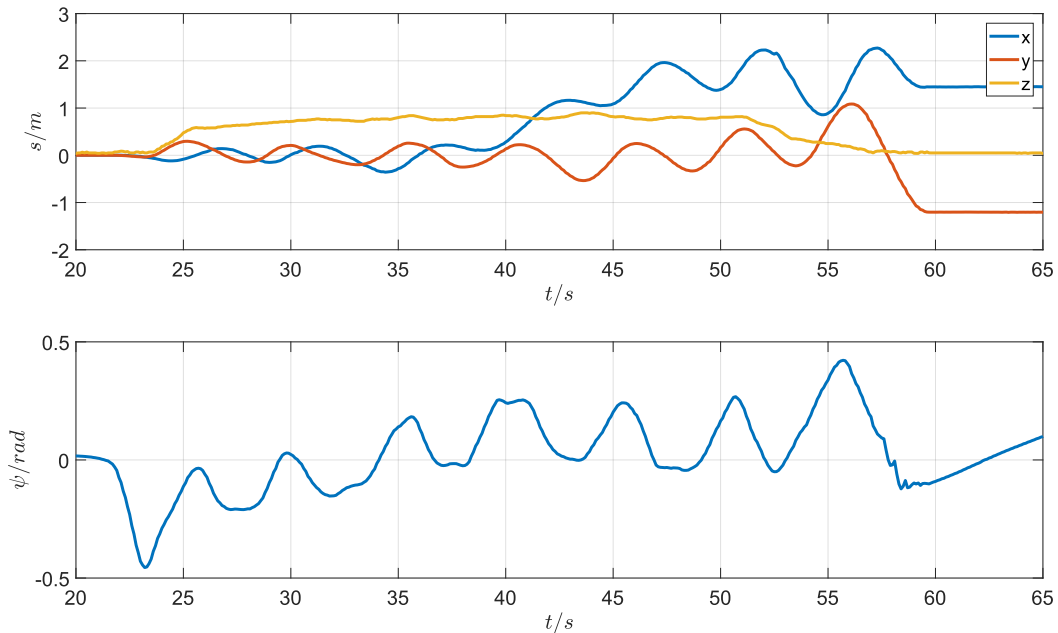


Abbildung 6.14: Ansteuerung eines vorgegebenen Wegpunktes innerhalb eines Raumes

an. Anschließend wird eine Translation von 1,5 Meter in x -Richtung vorgegeben. Ab einem Zeitpunkt von $t = 40s$ ist zu erkennen, dass sich die Asctec Neo in x -Richtung bewegt. Dabei gerät das Flugsystem allerdings in Schwingungen und die Abweichungen in x - und y -Richtung vergrößern sich immer weiter. Der vorgegebene Sollwert wird dementsprechend nicht genau erreicht. Auch dieses instabile Verhalten ergibt sich vermutlich aus Turbulenzen. Der Wind, welcher von den Rotoren erzeugt wird, kann nicht ungehindert aus dem Raum entweichen und wird von den Wänden abgeleitet. Daraus entstehen wiederum Verwirbelungen, welche das Flugverhalten der Asctec Neo weiter beeinflussen.

Um zu prüfen, ob sich das Flugverhalten verbessert, wenn die Turbulenzen verringert werden, wird ein weiterer Versuch im Außenbereich durchgeführt. Dadurch wird der durch die Rotoren erzeugte Wind nicht mehr von nahestehenden Wänden abgeleitet und das Flugsystem wird nicht mehr so stark von Verwirbelungen beeinflusst. Die Ergebnisse sind in Abbildung 6.15 dargestellt. Die Asctec Neo steigt zunächst zum Zeitpunkt $t = 31s$ auf eine Höhe von 1,0 Meter an. Anschließend wird eine Translation von $x = 5,0m$ vorgegeben. Im Vergleich zum vorherigen Versuch ergibt sich ein deutlich ruhigeres Flugverhalten und

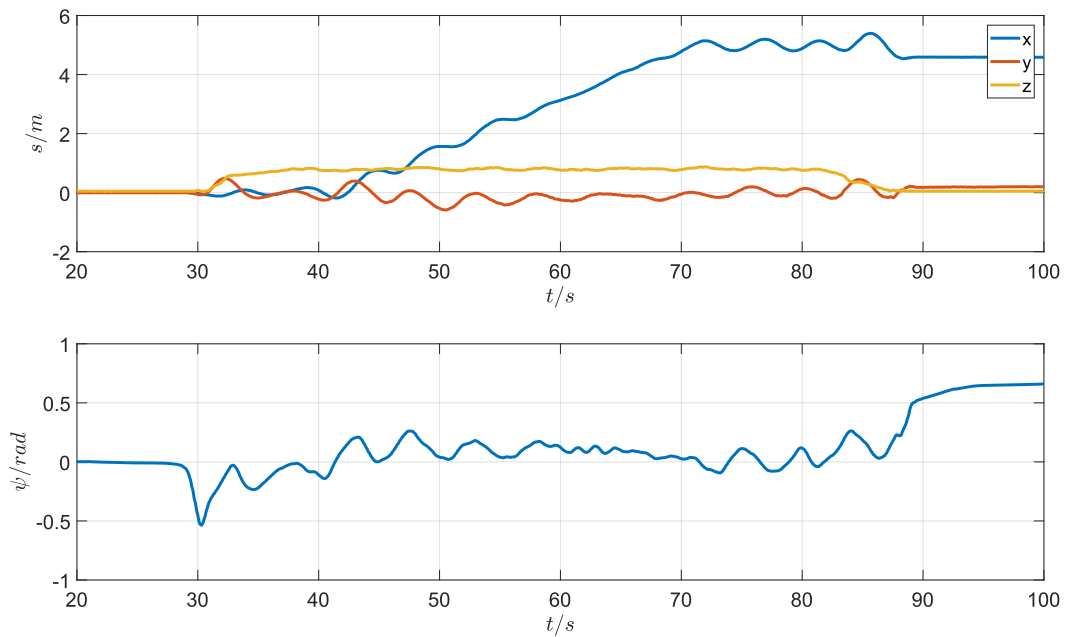


Abbildung 6.15: Ansteuerung eines vorgegebenen Wegpunktes im Außenbereich

der vorgegebene Wegpunkt kann erfolgreich angesteuert werden. Die Abweichungen bei der Rotation um die z -Achse hingegen weichen nicht sehr stark von den vorherigen Versuchen ab. Auch hier sind die Abweichungen beim Steig- und Sinkflug des Flugsystems am größten. Diese Abweichungen steigen auf einen Wert von über $0,5rad$ ($28,6^\circ$) an. Diese Differenzen lassen sich auch hier auf Turbulenzen zurückführen, da diese in Bodennähe am größten sind. Während des Fluges sind die Abweichungen allerdings deutlich geringer.

7 Abschlussbetrachtung

In dem folgenden Kapitel wird der Inhalt zusammenfassend dargestellt. Ein abschließender Ausblick bietet Ideen für weitere Arbeiten und Projekte.

7.1 Zusammenfassung

In dieser Arbeit wurde eine Positionsregelung für ein Mikroflugsystem entwickelt. Diese ermöglicht es, vorgegebene Wegpunkte automatisch anzusteuern. Als Regelalgorithmus wurde hier eine Zustandsregelung gewählt, da bei einem Flugsystem viele verschiedene Zustandsgrößen berücksichtigt werden müssen. Zur Entwicklung des Zustandsreglers wurden zunächst die benötigten Differentialgleichungen ermittelt. Anschließend wurden diese linearisiert und das Zustandsraummodell aufgestellt. Daraufhin konnte der eigentliche Regler berechnet werden.

Das Flugsystem wurde mit zwei Laserscannern vom Typ Hokuyo UST-20LX ausgestattet, um alle benötigten Zustandsgrößen zu erhalten. Ein Scanner wurde dafür in horizontaler Richtung an der Oberseite des Flugsystems montiert. Mithilfe von *hector_mapping* konnte daraus die Translation in x - und y - Richtung ermittelt werden. Der zweite Scanner wurde an der Unterseite des Flugsystems angebracht. Durch Verwendung eines Servomotors wurde dieser Scanner zusätzlich rotiert. Dadurch konnte der Bereich unter dem Flugsystem detailliert erfasst werden.

Zusätzlich wurde der Node *global_planner* verwendet und auf dieser Basis ein Trajektorienplaner implementiert. Existiert eine Karte der Umgebung, kann so ein Pfad zu einem frei wählbaren Zielpunkt berechnet werden. Daraufhin wird dieser Pfad von dem Flugsystem abgeflogen. Auch wenn keine Karte der Umgebung vorhanden ist, ist es möglich, einen Zielpunkt vorzugeben. Dabei wird zunächst ein Pfad direkt zum Ziel mit den aktuell verfügbaren Informationen geplant. Bewegt sich das Flugsystem, wird anschließend eine Karte der Umgebung erstellt. Befindet sich das MAV zu nah an einem Hindernis, wird ein neuer Pfad berechnet und das Flugsystem bewegt sich so immer näher an den Zielpunkt.

Die Regelgüte des entwickelten Positionsreglers wurde durch mehrere Simulationen in Matlab/Simulink und dem RotorS Simulator überprüft. Zusätzlich wurde das System in der Realität mit dem Flugsystem *Asctec Neo* getestet. Dabei wurde deutlich, dass das Flugverhalten sehr stark von Turbulenzen beeinflusst wird, welche durch die Rotoren entstehen. Besonders in Innenräumen wird das Flugsystem durch Windverwirbelungen beeinflusst. Im Außenbereich hingegen ergibt sich ein deutlich besseres Flugverhalten.

7.2 Ausblick

GUI

Damit das hier entwickelte System möglichst einfach verwendet werden kann, ist eine grafische Oberfläche sinnvoll. Dadurch kann das Flugsystem einfach und intuitiv gesteuert werden. Um dem MAV grundlegende Befehle zu übermitteln, wurde dafür bereits eine einfache GUI erstellt. Diese ist in Abbildung 7.1 dargestellt.

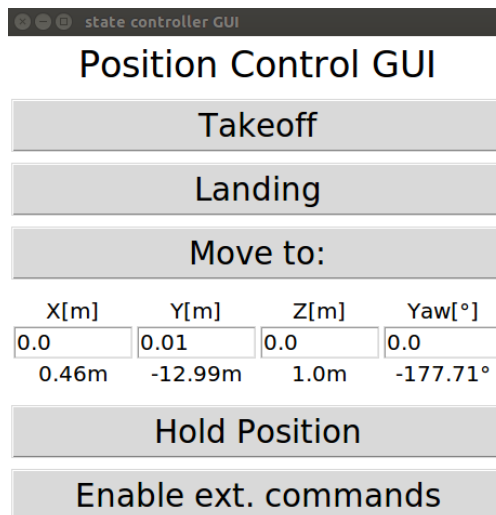


Abbildung 7.1: GUI zur Übergabe von Befehlen an den Positionsregler

Für die Darstellung von Sensorwerten und zur Übergabe von Zielpunkten wurde zusätzlich RViz verwendet.

Um die Steuerung des Flugsystems für den Anwender weiter zu vereinfachen ist es sinnvoll, eine grafische Oberfläche zu entwickeln, welche alle benötigten Funktionen sowie die Anzeige von benötigten Daten in einem Fenster darstellt.

3D-Pfadplanung

Bei der hier verwendeten Pfadplanung handelt es sich um ein zweidimensionales Verfahren. Befindet sich ein Hindernis auf dem Weg, wird somit immer versucht, dieses zu umfliegen. Da es mit einem Flugsystem allerdings auch möglich ist, über ein Hindernis zu fliegen, ist es denkbar, eine dreidimensionale Pfadplanung zu entwickeln. Da es bereits möglich ist, mit dem Positionsregler auch die Höhe des Flugsystems vorzugeben, kann dieser auch direkt Wegpunkte verarbeiten, welche von einer dreidimensionalen Pfadplanung ermittelt werden.

8 Anhang


8.1 Datenblätter

In dem folgenden Abschnitt ist das Datenblatt des Asctec Neo Flugroboters sowie ein Ausschnitt aus dem Datenblatt des Hokuyo UST-20LX Laserscanners aufgeführt.

8.1.1 Datenblatt des AscTec Neo


ASCENDING TECHNOLOGIES
 Amazing Technology!

ASCTEC Neo



The cutting-edge research UAV!

Fully redundant UAV!
 Payloads up to 2 kg!
 Optional 360° depth camera
 with Intel® RealSense™!



AscTec Trinity
 A new era of flight control.
 3X REDUNDANT
 MADE IN GERMANY



**Developed & Made
in Germany**

BY ASCENDING TECHNOLOGIES

Ascending Technologies GmbH /// Konrad-Zuse-Bogen 4 /// 82152 Krailling, Germany /// T +49 89 89556079-0 /// F +49 89 89556079-19 /// team@ascotec.de /// www.ascotec.de

Advanced research opportunities: Powerful, safe & versatile.

AscTec® Neo

- New platform developed & optimized within EuRoC, TRADR & AEROWORKS
- Configurations:
 - Hex
 - 9" propellers
 - 11" propellers
- AscTec® Trinity flight controller
- Pre-order Q2 2016
- Available Q3 2016

Key Features

- Payloads up to 2 kg
- Total weight below 5 kg
- Flight time: up to 26 mins
- Space-saving transport
 - Folding propellers
 - Detachable motor booms
- Efficient (and redundant) propulsion system
- High-performance motor control with FOC (Field Oriented Control)
- Active braking and energy recovery system
- Triple redundant flight controller (with stock firmware)
- Redundant power supply with smart batteries
- AscTec® Trinity with user-programmable module (STM32F4)
- Standardized mechanical interface

Flight controller

AscTec® Trinity

- 3 x fully equipped controllers + all sensors
- Redundant flight control at all times
- Real-time safety recovery:
 - Safe evaluation of your algorithms on one controller
 - Switch back to two redundant controllers is always possible (even in flight)

User Interfaces

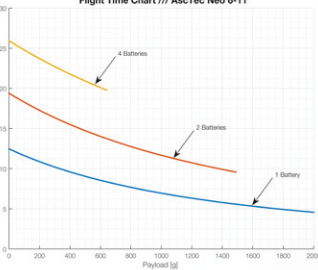
- One AscTec® Trinity module fully user-programmable (Cortex-M4 + FPU @ 180MHz)
- SDK provided free of charge with open-source tools
- Receive IMU data at high rates (raw + filtered)
- Various control commands available
- Electrical interfaces: UART (up to 2), I2C (up to 2), SPI, CAN, PWM (2x), USB (host or client)
- Multiple power outlets (battery voltage, 12V, 5V)

Optional Payloads

- Intel® NUC with Core i7-5557U (4x3.1 GHz, 8 GB RAM, 128 GB M.2 SSD, USB3.0, Iris Graphics 6100)
- 360° depth camera with Intel® RealSense™ technology
- AscTec® Atomboar v3 (4x1.91 GHz Intel Atom, 4 GB RAM, 64 GB SSD, USB3.0)
- Various camera mount options
- Propeller protection
- Laser scanner, optical flow sensor etc.



Flight Time Chart /// AscTec Neo 6-11





www.ascotec.de



Ascending Technologies is partner of EU FP7 – EuRoC (www.euroc-project.eu),
 EU FP7 – TRADR (www.tradr-project.eu), EU Horizon 2020 – AEROWORKS (www.aeroworks2020.eu)

Ascending Technologies GmbH /// Konrad-Zuse-Bogen 4 /// 82152 Krailling, Germany /// T +49 89 89556079-0 /// F +49 89 89556079-19 /// team@ascotec.de /// www.ascotec.de

8.1.2 Ausschnitt aus dem Datenblatt des Hokuyo UST-20LX

Date: 2014.6.12

Scanning Laser Range Finder

Smart-URG mini

UST-20LX (UUST004)

Specification

Symbol	Amended Reason				Pages	Date	Amended by	Ref.No
Approved by	Checked by	Drawn by	Designed by	Title	UST-20LX (UUST004) Specification			
Kamitani	Utsugi	Kamon	Yamamoto	Drawing No.	C-42-04078			1/6

HOKUYO AUTOMATIC CO.,LTD

4. Specifications

Product name	Scanning Laser Range Finder
Model	UST-20LX
Supply voltage	12VDC/24VDC (Operation range 10 to 30V ripple within 10%)
Supply current	150mA or less (during start up 450mA is necessary.)
Light source	Laser semiconductor (905nm) Laser class 1 (IEC60825-1:2007)
Detection range	0.06m to 20m (white Kent sheet) 0.06m to 8m (diffuse reflectance 10%) Max. detection distance : 60m
Accuracy	±40mm (*1)
Repeated accuracy	±50mm (*1)
Scan angle	270°
Scan speed	25ms (Motor speed 2400rpm)
Angular resolution	0.25°
Start up time	Within 10 sec (start up time differs if malfunction is detected during start up)
Input	JP reset input, photo-coupler input (current 4mA at ON)
Output	Synchronous Output, photo coupler open collector output 30VDC 50mA MAX.
Interface	Ethernet 100BASE-TX
LED display	Power supply LED display (Blue): Blinks during start up and malfunction state. Less than 15,000h.
Surrounding intensity	Note : Avoid direct sunlight or other illumination sources as it may cause sensor malfunction
Ambient temperature and humidity	-10°C to +50°C, below 85%RH (without dew, frost)
Storage temperature and humidity	-30°C to +70°C, below 85%RH (without dew, frost)
Vibration resistance	10 to 55Hz double amplitude of 1.5mm for 2hrs in each X, Y, and Z direction 55 to 200Hz 98m/s ² sweep of 2min for 1hr in each X, Y and Z direction
Vibration resistance (Operating)	55 to 150Hz 19.6m/s ² sweep of 2min for 30min in each X, Y and Z direction
Shock resistance	196m/s ² (20G) X, Y and Z direction 10 times.
EMC standards	(EMI) EN61326-1:2013 EN55011:2009 + A1:2010 (EMS) EN61326-1:2013 EN61000-4-2:2009 EN61000-4-3:2006 + A1:2008 + A2:2010 EN61000-4-4:2012 EN61000-4-6:2009 EN61000-4-8:2010
Protective Structure	IP65
Weight	130g (Excluding cable)
Material	Front case: Polycarbonate, Rear case: Aluminum
Dimensions (W×D×H)	50×50×70mm (sensor only)

(*1) Under the factory standard testing conditions using white Kent sheet.

Title	UST-20LX Specification	Drawing No.	C-42-04078	3/6
-------	------------------------	-------------	------------	-----

HOKUYO AUTOMATIC CO.,LTD

1. General

This sensor uses a laser source to scan 270° field of view. Positions of objects in the range are calculated with step angle and distance. Sensor outputs these data through communication channel

2. Structure

2-1. Structure diagram

2-2. Laser scanning image

3. Important notes

- (1) This sensor is not a safety device/tool
- (2) This sensor cannot be used for human body detection as per the machinery directives.
- (3) Hokuyo products are not developed and manufactured for the use in weapons, equipments or related technologies intended for destroying human lives or causing mass destruction. If such possibilities or usages are revealed, the sales of Hokuyo products to those customers might be halted by the laws of Japan such as Foreign Exchange Law, Foreign Trade Law or Export Trade control order. In addition, Hokuyo products are for the purpose of maintaining the global peace and security in accordance with the above law of Japan.

Title	UST-20LX Specification	Drawing No.	C-42-04078	2/6
-------	------------------------	-------------	------------	-----

HOKUYO AUTOMATIC CO.,LTD

5. Measurement Data

Distance Value (s)	Meaning
x < 21	Output numerical number "4" as Measurement error
21 ≤ x ≤ 60000	Valid distance [mm]
x > 60000	Output numerical number "65533" as Measurement error (object does not exists or object has low reflectivity)

6. Connection

6-1. Power source, I/O cable

Cable length: 1000mm Flying lead cable (AWG28)

Color	Signal
Red	COM Input +
Gray	COM Output -
Light Blue	JP Reset Input
Orange	Synchronous Output
Brown	+VIN (12VDC/24VDC)
Blue	-VIN

Note: Direction of Inputs and Outputs are mentioned from the sensor's side.

6-2. Ethernet cable

Cable length: 300mm

Color	Signal
Blue	TX +
White	TX -
Orange	RX +
Yellow	RX -

7. LED display

Title	UST-20LX Specification	Drawing No.	C-42-04078	4/6
-------	------------------------	-------------	------------	-----

HOKUYO AUTOMATIC CO.,LTD

8.2 Verwendung der implementierten Software

Zur Verwendung der implementierten Software zusammen mit dem RotorS Simulator können die folgenden Befehle verwendet werden:

Start RotorS Simulator mit den entsprechenden Konfigurationen

```
roslaunch mav_state_controller_sim neo_hokuyo_hector.launch  
  world_name:=basic8
```

Start Positionsregler

```
roslaunch mav_position_control neo_position_control_global_planner.launch
```

Start Trajektorienregler

```
roslaunch mav_position_control trajectory_controller
```

Start GUI

```
roslaunch mav_position_control_gui mpc_gui.py
```

Zur Ausführung der dargestellten Kommandos ist eine ROS Installation Voraussetzung. Auf der Homepage von ROS ist eine entsprechende Anleitung zu finden [24]. Das erste Kommando startet den RotorS Simulator mit dem Flugsystem *Asctec Neo*. Über den Parameter *world_name* wird die Simulationsumgebung definiert. Das zweite Kommando startet den Positionsregler. Mithilfe einer GUI, welche mit dem letzten Kommando gestartet wird, können dem Regler Positionen vorgegeben werden. Mit dem dritten Kommando wird der Trajektorienregler gestartet. Dieser übergibt Wegpunkte an den Positionsregler, um vorgegebene Zielpunkte anzusteuern.

8.3 Dateianhang

Dieser Arbeit liegt eine CD bei, welche u.a. eine digitale Version der Masterarbeit *Zustandsregelung für ein Mikroflugsystem zur Ansteuerung vorgegebener Wegpunkte in Innenräumen*, sowie implementierte Software enthält. Im Folgenden sind die Pfade aufgeführt unter denen die entsprechenden Daten auf der CD zu finden sind.

Digitale Version der Masterarbeit

/Masterarbeit_Kalle_Knipp.pdf

Latex Sourcecode der erstellten Masterarbeit

/Latex/

Implementierte ROS Pakete

/ROS/mav_position_control/
/ROS/mav_position_control_gui/
/ROS/mav_state_controller_sim/
/ROS/neo_startup/
/ROS/hokuyo_startup/

zusätzlich benötigte ROS Pakete

/ROS/catkin_simple/
/ROS/rotors_simulator/
/ROS/trinity_comm/
/ROS/trinity_msgs/
/ROS/trinity_ros/

Matlab/Simulink Simulationsdateien

/Matlab/

Bilder

/Latex/img/

Datenblätter

/Latex/Datenblaetter/

Verwendete Literatur

/Literatur/

Literaturverzeichnis

- [1] Carlos A. Arellano-Muro, Luis F. Luque-Vega, Bernardino Castillo-Toledo, and Alexander G. Loukianov. Backstepping control with sliding mode estimation for a hexacopter. In *Electrical Engineering, Computing Science and Automatic Control (CCE), 2013 10th International Conference on*, pages 31–36, 2013.
- [2] Ascending Technologies GmbH. Ascending projektseite, 2017. <http://www.asctec.de/> [Online: Stand 23.12.2017].
- [3] Ascending Technologies GmbH. Asctec neo datenblatt, 2017. <http://www.asctec.de/downloads/public/AscTec-Neo-flight-robot-research-uav-uas-drone-hexacopter-intel-realsense-iro-s-aeroworks.pdf> [Online: Stand 23.12.2017].
- [4] Marius Beul, Nicola Krombach, Matthias Nieuwenhuisen, David Droeschel, and Sven Behnke. Autonomous navigation in a warehouse with a cognitive micro aerial vehicle. In *Robot Operating System (ROS)*, pages 487–524. Springer, 2017.
- [5] Samir Bouabdallah, Pierpaolo Murrieri, and Roland Siegwart. Design and control of an indoor micro quadrotor. In *Robotics and Automation, 2004. Proceedings. ICRA '04. 2004 IEEE International Conference on*, volume 5, pages 4393–4398, 2004.
- [6] DJI. Dji, 2018. <https://www.dji.com/> [Online: Stand 31.01.2018].
- [7] Hugh Durrant-Whyte and Tim Bailey. Simultaneous localization and mapping: Part i. *IEEE Robotics & Automation Magazine*, 13(2):99–110, 2006.
- [8] Christian Forster, Matia Pizzoli, and Davide Scaramuzza. Svo: Fast semi-direct monocular visual odometry. In *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, pages 15–22, 2014.
- [9] Fadri Furrer, Michael Burri, Markus Achtelik, and Roland Siegwart. Rotors—a modular gazebo mav simulator framework: Robot operating system (ros): The complete reference (volume 1). pages 595–625. Springer International Publishing, Cham, 2016.
- [10] Andreas Geiger, Julius Ziegler, and Christoph Stiller. Stereoscan: Dense 3d reconstruction in real-time. In *IEEE Intelligent Vehicles Symposium*, Baden-Baden, Germany, 2011.
- [11] Martin Helmreich, Sebastian Prokop, and Chris Schwemmer. Algorithmen zur pfadplanung, 21.06.2005.

- [12] Hokuyo-UST-20LX. Hokuyo ust-20lx datenblatt. \url(https://en.manu-systems.com/HOK-UST-20LX_technical_specifications.pdf) [Online: Stand 05.04.2018].
- [13] Jaroslav Kautsky, Nancy K. Nichols, and Paul van Dooren. Robust pole assignment in linear state feedback. *International Journal of control*, 41(5):1129–1155, 1985.
- [14] Kalle Knipp. Selbstlokalisierung eines Mikroflugsystems mit Laserscannern zur 3D-Kartierung von Innenräumen. 2017.
- [15] S. Kohlbrecher, J. Meyer, O. von Stryk, and U. Klingauf. A flexible and scalable slam system with full 3d motion estimation. In *Proc. IEEE International Symposium on Safety, Security and Rescue Robotics (SSRR)*, 2011.
- [16] Helge Andreas Lauterbach and Dipl-Ing Nils Gageik. Stereo-optische abstandsmessung für einen autonomen quadcopter. *Bachelorarbeit, Universität Würzburg*, 2013.
- [17] Patrick Lester. A* pathfinding for beginners. *online*. *GameDev WebSite*. <http://www.gamedev.net/reference/articles/article2003.asp> (Acesso em 08/02/2009), 2005.
- [18] Jan Lunze. Beschreibung linearer systeme im zeitbereich. In *Regelungstechnik 1*, pages 55–117. Springer, 2014.
- [19] Jan Lunze. *Regelungstechnik*. Springer-Lehrbuch. Springer Vieweg, Berlin, 8., überarb. aufl. edition, 2014.
- [20] Teppo Luukkonen. Modelling and control of quadcopter. *Independent research project in applied mathematics, Espoo*, 22, 2011.
- [21] Sebastian Madgwick. An efficient orientation filter for inertial and inertial/magnetic sensor arrays. *Report x-io and University of Bristol (UK)*, 25, 2010.
- [22] Matlab. Matlab_lsqr, 2018. https://de.mathworks.com/help/matlab/ref/lsqr.html?s_tid=gn_loc_drop [Online: Stand 03.04.2018].
- [23] Tim Puls. *Lokalisations-und Regelungsverfahren für einen 4-Rotor-Helikopter*. Verlag Dr. Hut, 2011.
- [24] ROS. about ros. <http://www.ros.org/about-ros> [Online: Stand 30.01.2018].
- [25] ROS. global_planner. http://www.ros.org/global_planner [Online: Stand 30.01.2018].
- [26] ROS. Ros topics. <http://wiki.ros.org/Topics> [Online: Stand 05.04.2018].
- [27] ROS-URG-Node. Ros-urg-node. http://wiki.ros.org/urg_node [Online: Stand 05.04.2018].

- [28] Szymon Rusinkiewicz and Marc Levoy. Efficient variants of the icp algorithm. In *3-D Digital Imaging and Modeling, 2001. Proceedings. Third International Conference on*, pages 145–152, 2001.
- [29] Francesco Sabatino. Quadrotor control: Modeling, nonlinear control design, and simulation, 2015.
- [30] Shaojie Shen, Nathan Michael, and Vijay Kumar. Autonomous multi-floor indoor navigation with a computationally constrained mav. In *Robotics and automation (ICRA), 2011 IEEE international conference on*, pages 20–25, 2011.
- [31] TRADR. Tradr project, 2017. <http://www.tradr-project.eu/> [Online: Stand 23.12.2017].
- [32] Heinz Unbehauen. Das wurzelortskurven-verfahren. *Regelungstechnik I: Klassische Verfahren zur Analyse und Synthese linearer kontinuierlicher Regelsysteme, Fuzzy-Regelsysteme*, pages 169–184, 2008.
- [33] Patrick Vogl. *Design und Implementierung verschiedener Regelungskonzepte zur Lageregelung eines Quadropters*. PhD thesis, HAW Hamburg, Berliner Tor 5, 20099 Hamburg, 2014.
- [34] Holger Voos. Entwurf eines flugreglers für ein vierrotoriges unbemanntes fluggerät control systems design for a quadrotor uav. *at - Automatisierungstechnik*, 57(9):3936, 2009.
- [35] Wikipedia contributors. Savitzky–golay filter, 2018. https://en.wikipedia.org/wiki/Savitzky-Golay_filter [Online: Stand 03.04.2018].