

Bachelorarbeit

# **Optimierung eines Zustandsreglers für einen Quadrocopter**

im Studiengang Technische Informatik  
der Fakultät Informationstechnik

**Matthias Kapche**

**Zeitraum:** 3. Okt 2011 – 10. Feb 2012  
**Prüfer:** Prof. Dr. Jörg Friedrich

## **Eidestattliche Erklärung**

Hiermit versichere ich, die vorliegende Arbeit selbstständig und unter ausschließlicher Verwendung der angegebenen Literatur und Hilfsmittel erstellt zu haben.

Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungsbehörde vorgelegt und auch nicht veröffentlicht.

Esslingen, den 22. Februar 2012

-----

Unterschrift

## **Danksagung**

Ich möchte mich gerne an dieser Stelle bei allen Beteiligten für das Zustandekommen dieser Bachelor-Thesis bedanken.

An erster Stelle möchte ich mich bei Herrn Prof. Dr. Jörg Friedrich für die fachliche Betreuung und Unterstützung bei der Entstehung meiner Bachelor-Thesis bedanken.

Ein weiterer Dank geht an Udo Schlesinger, Robert Zins, Onur Ozun, Jian Wu, Peter Dück und Daniel Freymeyer, die mich bei auftretenden Problemen durch ihre konstruktiven Ideen stets unterstützt haben und für eine gute Motivation gesorgt haben.

Ein besonderer Dank geht an die Professoren der Fakultät Informationstechnik, die mir beim erfolgreichen Absolvieren meines Studiums geholfen haben.

Ganz herzlich will ich mich bei meinen Eltern, meinem Bruder und meiner Freundin bedanken, dafür dass sie mich immer unterstützt und aufgebaut haben.

# Inhaltsverzeichnis

<b>Eidestattliche Erklärung .....</b>	<b>2</b>
<b>Danksagung .....</b>	<b>3</b>
<b>Inhaltsverzeichnis.....</b>	<b>4</b>
<b>Abbildungsverzeichnis.....</b>	<b>6</b>
<b>Tabellenverzeichnis.....</b>	<b>8</b>
<b>Abkürzungen.....</b>	<b>9</b>
<b>Kurzfassung.....</b>	<b>11</b>
<b>1     <b>Einleitung</b> .....</b>	<b>12</b>
1.1    Motivation .....	12
1.2    Aufbau dieser Arbeit .....	13
<b>2     <b>Projektumfeld</b>.....</b>	<b>14</b>
2.1    Ist-Zustand .....	14
2.2    Soll-Zustand .....	14
2.3    Entwicklungsumgebung .....	14
2.4    Zielumgebung .....	15
<b>3     <b>Serielle Sensordaten-Funkübertragung</b>.....</b>	<b>16</b>
3.1    Aufbau der Funkübertragung .....	16
3.2    Paketaufbau .....	18
3.3    Parameter .....	20
3.4    Implementierung in Matlab .....	23
3.5    Nutzung.....	24
<b>4     <b>Der Quadrocopter</b>.....</b>	<b>26</b>
4.1    Einführung.....	26
4.1.1   Earth- und Bodyframe .....	27
4.1.2   Auftrieb .....	28
4.1.3   Nicken.....	29
4.1.4   Rollen.....	29
4.1.5   Gieren .....	30
4.2    Platine der Fluglageregelung .....	31
4.2.1   Sensoren.....	31
4.2.2   Die alte Platine der Fluglageregelung.....	32
4.2.3   Die neue Platine der Fluglageregelung.....	33
<b>Matthias Kapche</b>	<b>4</b>

4.3	Sensorkalibrierung.....	36
4.3.1	Feste Sensorwerte.....	36
4.3.2	Kalibrierung mit flüchtigem Speicher.....	36
4.3.3	Kalibrierung mit EEPROM-Speicher .....	36
<b>5</b>	<b>Der Zustandsregler .....</b>	<b>40</b>
5.1	Einführung.....	40
5.1.1	Einfaches System .....	41
5.1.2	Steuerbarkeit.....	42
5.1.3	Beobachtbarkeit .....	42
5.1.4	Zustandsregler .....	43
5.1.5	Pole-Placement.....	43
5.1.6	Zustandsregler mit Beobachter.....	46
5.2	Quadrokopter in MATLAB-Simulink .....	47
5.2.1	Gesamtes MATLAB-Simulink-Modell .....	48
5.2.2	Remote-Control-Block.....	49
5.2.3	Control-Block.....	50
5.2.4	Dynamics-Block.....	52
5.2.5	Sensor-Block.....	54
5.3	Zustandsregler des Quadropters .....	54
5.4	Optimierung des Zustandsreglers des realen Quadropters .....	56
<b>6</b>	<b>Zusammenfassung und Ausblick .....</b>	<b>63</b>
<b>7</b>	<b>Literaturverzeichnis.....</b>	<b>64</b>

## Abbildungsverzeichnis

Abbildung 1: XBee-Modul .....	11
Abbildung 2: Der hochschulinterne Quadrokofter .....	15
Abbildung 3: Datenübertragung im Sampling-Mode .....	17
Abbildung 4: Allgemeiner Paketaufbau .....	18
Abbildung 5: Parameter-ID .....	18
Abbildung 6: Antwort-Paket des Quadrokofters .....	20
Abbildung 7: Sensordatenstream in der Matlab-Konsole, v. r. Zeitstempel, Parameter-ID, Sensorwert .....	23
Abbildung 8: MATLAB-Konsole mit ausgeführtem Programm.....	25
Abbildung 9: Beschleunigung in z-Richtung geplottet in MATLAB .....	25
Abbildung 10: Quadrokofterschema.....	26
Abbildung 11: Das Earth- (grün) und Bodyframe (rot) .....	27
Abbildung 12: links der Quadrokofter mit ausgerichtetem Body- und Earthframe, rechts der um den Winkel $\psi$ gedrehte Quadrokofter.....	28
Abbildung 13: Auftrieb .....	28
Abbildung 14: Nicken .....	29
Abbildung 15: Rollen .....	29
Abbildung 16: Gieren .....	30
Abbildung 17: Gyroskop.....	31
Abbildung 18: Beschleunigungssensor.....	31
Abbildung 19: Alte Flight-Control-Platine .....	32
Abbildung 20: Neue Flight-Control-Platine .....	33
Abbildung 21: Hebelstellung für den Kalibriervorgang .....	37
Abbildung 22: Trimmung .....	37
Abbildung 23: Ablauf der Main-Funktion .....	39
Abbildung 24: Einfaches SISO-System dritter Ordnung .....	41
Abbildung 25: Einfaches SISO-System mit Zustandsregler .....	43
Abbildung 26: s-Ebene: links stabil, rechts nicht stabil.....	44
Abbildung 27: Prozess im „open-loop“-Betrieb mit direktem IO-Durchgriff.....	44
Abbildung 28: Prozess im „open-loop“-Betrieb ohne direkten IO-Durchgriff ....	45
Abbildung 29: Einfaches SISO-System dritter Ordnung mit Zustandsregler und Beobachter.....	47
Abbildung 30: Gesamtsystem des Quadrokofters in MATLAB-Simulink .....	48
Abbildung 31: Remote-Control-Block.....	49
Abbildung 32: Remote-Control-Stimuli .....	50
Abbildung 33: Control-Block .....	51
Abbildung 34: Dynamics-Block.....	52
Abbildung 35: Sensor-Block.....	54
Abbildung 36: Zustandsregler des Quadrokofters.....	55
Abbildung 37: Sollwerte der Fersteuerung .....	56

Abbildung 38: Gemessene Zustandsvariablen des Quadropters .....	58
Abbildung 39: Sollwerte mit berechneten Kv-Faktoren .....	59
Abbildung 40: Zustandsrückführung mit den berechneten .....	61
Abbildung 41: Sollwertberechnung der Motoren .....	62

## Tabellenverzeichnis

Tabelle 1: Parameter-ID Zusammensetzung .....	19
Tabelle 2: Fehler-Codes .....	19
Tabelle 3: Parameter-ID's der Beschleunigungswerte in x-, y- und z- Richtung .....	20
Tabelle 4: Parameter-ID's der Winkelgeschwindigkeiten, Temperatur, Luftdruck und der Batteriespannung.....	21
Tabelle 5: Parameter-ID's der Motorkräfte und Motorsollwerte.....	22
Tabelle 6: Parameter-ID's der Eingangssignale der Fernsteuerung .....	22
Tabelle 7: Gemessene Sensorwerte.....	34
Tabelle 8: Zustandsmatrizen .....	40
Tabelle 9: Der Vektor pos_vel_acc .....	53



## Abkürzungen

accX	Beschleunigung in x-Richtung (C-Code)
accX_Raw	Rohwerte der Beschleunigung in x-Richtung (C-Code)
accY	Beschleunigung in y-Richtung (C-Code)
accY_Raw	Rohwerte der Beschleunigung in y-Richtung (C-Code)
accZ	Beschleunigung in z-Richtung (C-Code)
accZ_Raw	Rohwerte der Beschleunigung in z-Richtung (C-Code)
airPressureRaw	Rohwert des Luftdrucks (C-Code)
angP	Winkel theta (C-Code)
angR	Winkel phi (C-Code)
angVelP	Winkelbeschleunigung des Winkels theta (C-Code)
angVelPRaw	Rohwert der Winkelbeschleunigung des Winkels theta (C-Code)
angVelR	Winkelbeschleunigung des Winkels phi (C-Code)
angVelRRaw	Rohwert der Winkelbeschleunigung des Winkels phi (C-Code)
angVelY	Winkelbeschleunigung des Winkels psi (C-Code)
angVelYRaw	Rohwert der Winkelbeschleunigung des Winkels psi (C-Code)
ax	Beschleunigung des Body-Frames in x-Richtung
ay	Beschleunigung des Body-Frames in y-Richtung
battery	Batteriespannung (C-Code)
batteryRaw	Rohwert der Batteriespannung (C-Code)
EEPROM	(electrically erasable programmable read-only memory) nichtflüchtiger Speicher der nur gelesen werden kann
F_prop	Resultierende Kraft der Propeller
MIMO	Multiple Input Multiple Output
MU	Messumformer (Sensor)
phi	Winkel um die x-Achse des Body-Frames

pos-vel-acc	(positions-velocities-accelerations) Vektor der die Werte für die Position, die Geschwindigkeiten und die Beschleunigungen enthält
psi	Winkel um die z-Achse des Body-Frames
r_phi	Winkelgeschwindigkeit vom Winkel phi
r_psi	Winkelgeschwindigkeit vom Winkel psi
r_theta	Winkelgeschwindigkeit vom Winkel theta
SISO	Single Input Single Output
Temp	Temperatur (C-Code)
tempRaw	Rohwert der Temperatur (C-Code)
theta	Winkel um die y-Achse des Body-Frames
U1	Summe aller Propellerkräfte
U2	Winkelbeschleunigung des Winkels phi
U3	Winkelbeschleunigung des Winkels theta
U4	Winkelbeschleunigung des Winkels psi

## Kurzfassung

Heutzutage werden Flugroboter immer gefragter, da sie in schwierigem Gelände oder unter unmenschlichen Gegebenheiten verwendbar sind. Sie werden z.B. bei der Feuerwehr oder der Polizei zu Überwachungszwecken oder für Kontrollflüge eingesetzt. Eine Art der Flugroboter sind Quadrocopter. Diese eignen sich hervorragend durch ihre einfache Mechanik und präzise Steuerung.

Diese Arbeit befasst sich mit der Fluglageregelung des hochschulinternen Quadrocopters. Diese Lageregelung wurde in einer vorangegangenen Studienarbeit um einen Zustandsregler erweitert, der dem Quadrocopter das Fliegen ermöglicht. Zustandsregler werden heutzutage in sehr vielen Regelsystemen eingesetzt, da man mit ihnen vermeintlich unregelbare Systeme regeln kann. Zu diesen schwer regelbaren Systemen gehört auch der Quadrocopter, der nur durch die Drehzahländerung der vier einzelnen Rotoren stabil in der Luft stehen bzw. fliegen kann. Dabei werden alle sechs Freiheitsgrade geregelt. Allerdings ist dieser Zustandsregler noch nicht optimal und wirkt etwas träge. Um den Zustandsregler zu optimieren, werden über ein ZigBee-Modul die Telemetriedaten zu einem Rechner geschickt und anhand dieser Sensordaten ausgewertet.



Die Aufzeichnung der Telemetriedaten und das Quadrocopter Simulationsmodell selbst, erfolgen in Matlab. Anhand von Simulationen in Matlab, kann der Zustandsregler schließlich in C implementiert werden. Die Matlab-Simulationen beruhen auf den realen Hardware-Daten des Quadrocopters und werden als 3D-Modell dargestellt.

Abbildung 1: Xbee-Modul. [2]

Ziel dieser Arbeit ist es, dass der Quadrocopter trotz Fremdeinwirkung stabil in der Luft gehalten und der Matlab-Simulation angeglichen wird.

Als Fremdeinwirkung zählen unter anderem Seitenwinde oder auch zusätzliches Gewicht wie z.B. das Funk-Kamerasystem mit dem der Quadrocopter auch auf weitere Entfernung sicher gesteuert werden kann.

# 1 Einleitung

## 1.1 Motivation

Mittlerweile gibt es Flugroboter schon als günstige Spielzeuge, aber nicht nur in diesem Bereich werden sie eingesetzt. Ein großer Markt geht in Richtung autonome Überwachung. So auch der Quadrokopter, dieses mechanisch einfache Gerät wird heutzutage immer interessanter, da mit ihm neue Möglichkeiten der Landschaftsfotografie oder Videografie und Überwachung, kostengünstig möglich werden. Gerade in der heutigen Zeit in der regenerative Energiequellen immer wirtschaftlicher werden, müssen diese auch kontrolliert und gegebenenfalls repariert werden. In einigen Pilotanlagen und kommerziell genutzten Anlagen werden mittlerweile Quadrokopter als Kontroll- und Überwachungsgerät eingesetzt. Diese sind deutlich kostengünstiger als Flugzeuge oder ausgebildetes Personal. Aber nicht nur in diesem Bereich, auch in Katastrophengebieten wie z.B. Fukushima fliegen Quadrokopter zur Überwachung über die Reaktoren des havarierten Atomkraftwerks. Für diese Einsätze sind Quadrokopter bestens geeignet da sie sehr präzise steuerbar aber trotzdem sehr robust sind.

Zukünftig werden die Quadrokopter auch in Schwärmen fliegen können, um damit z.B. mobile W-LAN-Netzwerke oder Mobilfunknetzwerke aufzubauen und so Events kostengünstig oder Katastrophengebiete schnell zu versorgen.

## **1.2 Aufbau dieser Arbeit**

Die vorliegende Bachelorarbeit ist in 6 Kapitel unterteilt. Dieses Kapitel gibt einen kurzen Einblick in die Thematik und im Anschluss werden die Details weiter erörtert.

### **Kapitel 2**

Das 2. Kapitel beschreibt die Entwicklungsumgebung und das Umfeld dieser Arbeit. Es erläutert die Aufgaben und Ziele des Projektes, und beschreibt den Quadrocopter.

### **Kapitel 3**

In diesem Kapitel wird die Sensordatenübertragung über das vorhandene Xbee-Modul, mit dem Paketaufbau und der verfügbaren Parameter beschrieben.

### **Kapitel 4**

Dieses Kapitel befasst sich mit den Grundlagen des Quadrocopters und seiner vorhandenen Hardware. Es stellt die Probleme mit der neuen Platine der Fluglageregelung und deren Lösung dar. Des Weiteren wird die Sensorkalibrierung erläutert.

### **Kapitel 5**

In diesem Kapitel wird der vorhandene Zustandsregler zuerst in der Matlab-Simulation und dann auf der realen Hardware erläutert. Es zeigt die Probleme und Lösungen des Zustandsreglers.

### **Kapitel 6**

Das letzte Kapitel fasst die Ergebnisse dieser Arbeit zusammen und bietet einen Ausblick auf potentiell zukünftige Weiterentwicklungen.

## 2 Projektumfeld

### 2.1 Ist-Zustand

Der Quadrokopter ist flugfähig, allerdings mit einem PID-Regler, der im Verlauf einer Masterarbeit mitentwickelt wurde. Dieser verhält sich sehr unruhig und man muss ständig über die Fernsteuerung gegensteuern. In einer Studienarbeit wurde der vorhandene PID-Regler durch einen Zustandsregler ausgetauscht, dadurch hatte der Quadrokopter in der Matlab-Simulation verbesserte Regeleigenschaften. Allerdings konnten diese Ergebnisse nicht auf dem realen Quadrokopter umgesetzt werden. Desweiteren können die Sensordaten zur Auswertung des Quadrokopterhaltens, per serieller Funkübertragung gesendet und am Notebook empfangen werden.

### 2.2 Soll-Zustand

Die Thesis befasst sich mit der Optimierung des vorhandenen Zustandsreglers. Dieser soll soweit optimiert werden, dass der reale Quadrokopter stabil fliegt. Um das Optimieren zu vereinfachen wird die vorhandene Sensor-Funkübertragung um die grafische Darstellung in MATLAB erweitert. Desweiteren soll eine Sensorkalibrierung implementiert werden, mit der der Quadrokopter über eine Hebelstellung an der Fernbedienung kalibriert werden kann. Diese Kalibrierungswerte werden anschließend im EEPROM gespeichert. Diese Kalibrierungswertspeicherung wird benötigt, um den Quadrokopter auch in einer Umgebung verwendet zu können, in der es keine ebenen Flächen zum kalibrieren gibt.

### 2.3 Entwicklungsumgebung

Für die Softwareentwicklung wurde die Entwicklungsumgebung (IDE) CodeWarrior Development Studio 5.9.0 der Firma Freescale verwendet. Die Simulation des Quadrokopters und die Sensorwert-Darstellung werden in Matlab 2010 ausgeführt bzw. implementiert. Für die Kompilierung des Programms „Valiquad“, das für die Funkübertragung benötigt wird, muss zusätzlich die Entwicklungsumgebung Netbeans DIE 7.0.1 installiert sein.

## 2.4 Zielumgebung

Die Entwicklung des Fluglagereglers erfolgt auf dem Quadrokopter selbst. Die Platine des Quadrokopters ist mit dem Mikrocontroller MC9S12XDT256CAL der Firma Freescale bestückt. Dieser bekommt die Werte der aktuellen Lage von den 4 Sensoren, die in Kapitel 5.2.1 genauer beschrieben werden.



Abbildung 2: Der hochschulinterne Quadrokopter

## 3 Serielle Sensordaten-Funkübertragung

Die serielle Sensordaten-Funkübertragung, genannt „Valiquad“ ist eine Studienarbeit aus dem Wintersemester 2010/2011 von Herrn Stübler [4]. Diese Arbeit dient zur Überwachung aller Sensorwerte und Fernsteuerungswerte des Quadropters. Als Hardware zur Übertragung der Daten werden zwei XBee-Module verwendet, eines das direkt auf dem Quadropter sitzt und eines das zum Empfangen der Daten am Notebook angeschlossen ist.

Mit Hilfe dieser konsolenbasierten Software wird der Zustandsregler des Quadropters im späteren Abschnitt optimiert.

Die Software ist so aufgebaut, dass man verschiedene Parameter (Sensorwerte) vom Quadropter abfragen kann. Dazu wird der Quadropter mit den sogenannten Parameter-IDs konfiguriert. Der Quadropter schickt dann die konfigurierten Sensordaten an das Notebook.

In dem Unterkapitel 3.2 wird zuerst der grundsätzliche Paketaufbau mit der Übertragung erläutert, danach werden alle abfragbaren Sensorwerte aufgelistet und erläutert.

Der Nachteil des Programms „Valiquad“ ist die nicht vorhandene grafische Darstellung der empfangenen Sensordaten. Dieser Nachteil wird durch die zusätzliche Implementierung in MATLAB behoben.

### 3.1 Aufbau der Funkübertragung

Mit dem Programm „Valiquad“ ist es möglich die Sensordaten auf zwei Arten zu schicken bzw. zu empfangen. Die erste Möglichkeit ist das zyklische Abfragen, das sogenannte „Polling“, bei dem das Notebook über die ganze Zeit, in der man die Sensorwerte benötigt, dem Quadropter zyklisch Abfragen schickt. Der Nachteil dabei, es entsteht sehr viel Daten-Overhead. Die zweite Möglichkeit ist das ununterbrochene Senden nach nur einer Anforderung, das sogenannte „Sampling“, bei dem man über das Notebook den Quadropter am Anfang der Kommunikation konfiguriert und zum Starten des Bytestreams eine Anforderung schickt.

In dieser Arbeit wurde das Sampling-Verfahren des Programms Valiquad verwendet. Dieses Sampling-Verfahren wird nun genauer erläutert.

Beim Sampling-Verfahren werden die Sensorwerte nicht andauernd durch Abfragen (polling) des Masters, in diesem Fall das Notebook gesendet, sondern der Quadropter wird einmalig auf die zu sendenden Sensorwerte konfiguriert. Nach der Konfiguration kann die Datenübertragung gestartet werden. Der



Quadrokopter sendet solange, bis vom Master (Notebook) das Beenden-Signal geschickt wird.

Die Übertragung im Sampling-Mode erfolgt in sogenannten Frames. Von diesen Frames gibt es vier Stück A, B, C und D. Jedes Frame kann bis zu 60 Bytes Nutzdaten übertragen und wird mit jeweils einer Parameterliste konfiguriert. Außerdem lässt sich mit der Konfiguration der Frames die Sample-Rate steuern, das heißt, dass wenn ein Parameter mit allen vier Frames A, B, C und D geschickt wird, man die höchste Sample-Rate hat. Will man nur die halbe Sample-Rate, überträgt man den Parameter mit nur zwei Frames, d.h. entweder das Frame A und C, oder das Frame B und D. Eine ungerade Zahl zu wählen sowie andere Framekombinationen ergeben keinen Sinn, da man sonst keine gleichmäßige Sample-Rate hat. Allerdings kann man auch nur ein Frame nehmen, was einem Viertel der höchsten Sample-Rate entspricht.

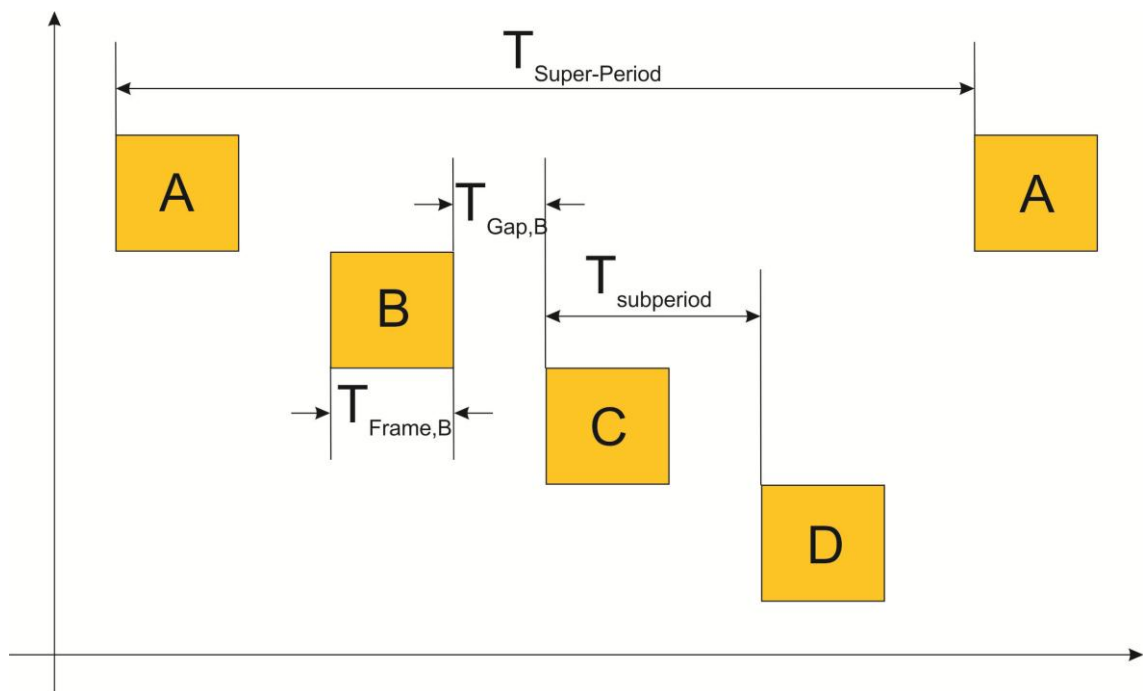


Abbildung 3: Datenübertragung im Sampling-Mode. [4]

Die Periode, bis alle Frames durchlaufen sind, nennt man  $T_{Super-Period}$  und entspricht 4 mal der Subperiode  $T_{Subperiod}$ . Die Sub-Periode entspricht genau der Zeit, die von Anfang eines Frames bis zum nächsten Frame verstreicht. Sie ist bei allen vier Frames dieselbe. Dann gibt es noch die Frame-Zeit  $T_{Frame,x}$  und die Zeit, die zwischen den einzelnen Frames,  $T_{Gap,x}$ . Beide Zeiten können bei jedem Frame unterschiedlich sein, je nachdem wie die einzelnen Frames konfiguriert wurden. Nach einer Super-Periode beginnt das Spiel von neuem. Dabei muss beachtet werden, dass man die Frames nicht von Super-Periode zu Super-Periode neu konfigurieren kann.

## 3.2 Paketaufbau

Die ganze Kommunikation läuft über das Senden von Paketen ab, die, wie in Abbildung 4 dargestellt, aufgebaut sind. Das Paket besitzt einen Header, in dem sich die Message-ID „MsgID“ und die Länge der Nutzdaten befinden. Darauf folgen die Nutzdaten selbst und eine Checksumme. Die maximale Länge mit 64 Bytes wurde gewählt, um alle käuflichen Varianten der XBee-Module zu verwenden, die Pakete mit einer Paketlänge zwischen 70 und 100 Bytes senden können.

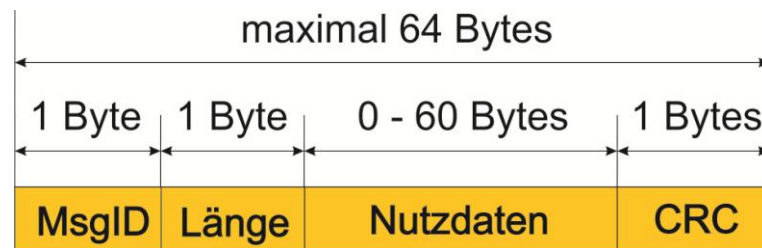


Abbildung 4: Allgemeiner Paketaufbau. [4]

Um die Sensordaten unterscheiden zu können, besitzt jeder Sensorwert eine eigene Parameter-ID „ParID“. Das erste Bit der Parameter-ID beschreibt ob es sich um eine Observable oder einen normalen Parameter handelt. Observables sind die Parameter, die im Sampling-Mode, d.h. mit nur einer Anfrage ununterbrochen gesendet werden. Bei normalen Parametern, die in dieser Arbeit aber nicht verwendet wurden, muss jeder Wert zyklisch abgefragt werden. Die nächsten 3 Bits beinhalten den Datentyp des Observables, alle unterstützten Datentypen werden in Tabelle 1 aufgeführt. Die nächsten 4 Bits enthalten die Gruppe, in der sich die Observables befinden, es stehen bis zu 16 Gruppen zu Verfügung und dienen dazu z.B. Observables auch mit anderen Datentypen zu senden. Alle Observables, die in dieser Arbeit verwendet wurden, befinden sich in der Gruppe „0“. Die letzten 8 Bits enthalten die Sequenz-Nummer, mit der die einzelnen Sensorwerte identifiziert werden.

Alle verfügbaren Sensorwerte werden in Kapitel 3.3 mit ihren Parameter-IDs aufgeführt.

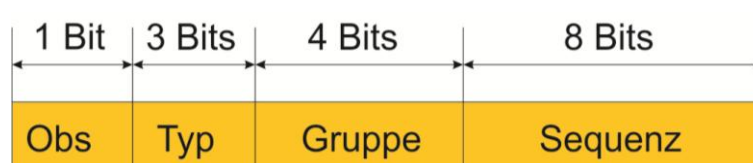


Abbildung 5: Parameter-ID. [4]

Name	Bereich
Obs	0x0 ... 0x1
Typ	0x0 ... 0x7
Gruppe	0x0 ... 0xF
Sequenz	0x00 ... 0xFF

Tabelle 1: Parameter-ID Zusammensetzung. [4]

Die Quadrokopter-Antwort-Message kann eine Größe von maximal 60 Bytes haben. Das erste Byte enthält den Fehler-Code, ist kein Fehler aufgetreten wird der Wert „0“ zurückgegeben. Die weiteren Fehler-Codes sind in Tabelle 2 beschrieben.

Fehler-Code	Beschreibung
0x00	Es ist kein Fehler aufgetreten.
0x01	Format-Fehler: Die Nachricht, die vom Quadrokopter empfangen wurde ist fehlerhaft.
0x02	Overflow-Fehler: in einem Frame wurden zu viele Observables konfiguriert.
0x03	Parameter-Fehler: Die gesendete Parameter-ID ist falsch, oder nicht vorhanden.
0x04	Sub-Perioden-Fehler: Die gewünschte Sub-Periode kann nicht gesetzt werden.
0x05	Motor-Fehler: Nach dem die Motoren angelaufen sind, können keine Konfigurationen mehr unternommen werden.

Tabelle 2: Fehler-Codes. [4]

Die restlichen 59 Bytes der Antwort-Message enthalten die konfigurierten Sensordaten.



Abbildung 6: Antwort-Paket des Quadropters. [4]

### 3.3 Parameter

Alle aufgeführten Parameter sind Observables und in der Gruppe „0“ untergebracht.

Beschleunigungssensor		
Parameter-ID	Datentyp	Parameter
0xD000	int16	accXRaw
0xD001	int16	accX
0xD002	int16	accYRaw
0xD003	int16	accY
0xD004	int16	accZRaw
0xD005	int16	accZ

Tabelle 3: Parameter-ID's der Beschleunigungswerte in x-, y- und z-Richtung

<b>Gyroskope, Temperatur, Luftdruck und Batterie-Spannung</b>		
<b>Parameter-ID</b>	<b>Datentyp</b>	<b>Parameter</b>
<b>0xD006</b>	int16	angVelRRaw
<b>0xD007</b>	int16	angVelR
<b>0x8008</b>	float32	angR
<b>0xD009</b>	int16	angVelPRaw
<b>0xD00A</b>	int16	angVelP
<b>0x800B</b>	float32	angP
<b>0xD00C</b>	int16	angVelYRaw
<b>0xD00D</b>	int16	angVelY
<b>0xD00E</b>	int16	tempRaw
<b>0xD00F</b>	int16	Temp
<b>0xD010</b>	int16	airPressureRaw
<b>0xD011</b>	int16	batteryRaw
<b>0xD012</b>	int16	battery

Tabelle 4: Parameter-ID's der Winkelgeschwindigkeiten, Temperatur, Luftdruck und der Batteriespannung

Motoren Informationen		
Parameter-ID	Datentyp	Parameter
0xD013	int16	forceFront
0xA014	uint8	setpointFront
0xD015	int16	forceRear
0xA016	uint8	setpointRear
0xD017	int16	forceRight
0xA018	uint8	setpointRight
0xD019	int16	forceLeft
0xA01A	uint8	setpointLeft
0xD01B	int16	forceTotal

Tabelle 5: Parameter-ID's der Motorkräfte und Motorsollwerte

Fernsteuerungs-Werte		
Parameter-ID	Datentyp	Parameter
0xA020	uint8	remoteConnected
0xA021	uint8	remoteMotorsOn
0xA022	uint8	remoteForceRaw
0xD023	int16	remoteForce
0xA024	uint8	remoteYawRaw
0xD025	int16	remoteYaw
0xA026	uint8	remotePitchRaw
0xD027	int16	remotePitch
0xA028	uint8	remoteRollRaw
0xD029	int16	remoteRoll

Tabelle 6: Parameter-ID's der Eingangssignale der Fernsteuerung

### 3.4 Implementierung in Matlab

Erste Versuche den Java-Code des Programms „Valiquad“ in MATLAB-Code umzusetzen, zeigten, dass MATLAB in der Lage ist eine Verbindung mit dem Quadrokofter aufzubauen. Der empfangene Bytestream besteht aus Hexadezimalzahlen, jedoch arbeitet MATLAB nur mit Dezimalzahlen. Es gibt zwar die MATLAB-Funktion `hex2dec`, mit der man die Hexadezimalzahlen in Dezimalzahlen umrechnen kann, allerdings wird der Wert „0“ ausgeschnitten. Diese lassen sich nicht wieder rekonstruieren, weshalb nach einer neuen Lösung gesucht werden musste.

Da MATLAB auch Java-Code ausführen kann, wurden die einzelnen Funktionen des Programms „Valiquad“ in MATLAB aufgerufen.

Um die einzelnen Sensordaten aus dem Bytestream, der vom Quadrokofter gesendet wird, herauszuholen, werden zusätzlich die Parameter-IDs ausgegeben. Anhand dieser Parameter-IDs kann man den Sensorwert zuordnen.

```
10:34:28 : -12283 : -98
10:34:28 : -12283 : -98
10:34:28 : -12283 : -98
10:34:28 : -12283 : -98
10:34:28 : -12283 : -98
10:34:29 : -12283 : -98
10:34:29 : -12283 : -98
10:34:29 : -12283 : -98
```

Abbildung 7: Sensordatenstream in der Matlab-Konsole, v. r. Zeitstempel, Parameter-ID, Sensorwert

Dies ist für den nächsten Schritt, für das Plotten in MATLAB sehr wichtig. So ist es möglich die Sensorwerte über einfache Switch-Case-Abfragen anhand der Parameter-IDs auszuschneiden und mit einer Plot-Funktion zu plotten.

Im Moment funktioniert das Zeichnen der Diagramme nur bedingt, d.h. bei einem gesendeten Sensorwert funktioniert das Plotten flüssig und in Echtzeit. Bei zwei gesendeten Sensorwerten hat man aber schon Sprünge in der Darstellung, d.h. die Sensorwerte ändern sich schneller als die Plot-Funktion zeichnet. Bei drei gesendeten Sensorwerten werden nur noch zwei davon geplottet.

## 3.5 Nutzung

Nach dem Öffnen des MATLAB-Files muss dieses erst richtig konfiguriert werden.

1. Schritt: Der Pfad der ausführbaren Datei „Valiquad.jar“ muss auf den aktuellen Speicherort geändert werden.

```
javaaddpath('C:\Users\Benutzer\Documents  
\NetBeansProjects\Valiquad\dist\Valiquad.jar');
```

2. Schritt: Um eine serielle Verbindung mit MATLAB herzustellen, muss der richtige COM-Port an dem sich das XBee-Device befindet ausgewählt werden. Am besten schaut man dazu im Gerätemanager nach.

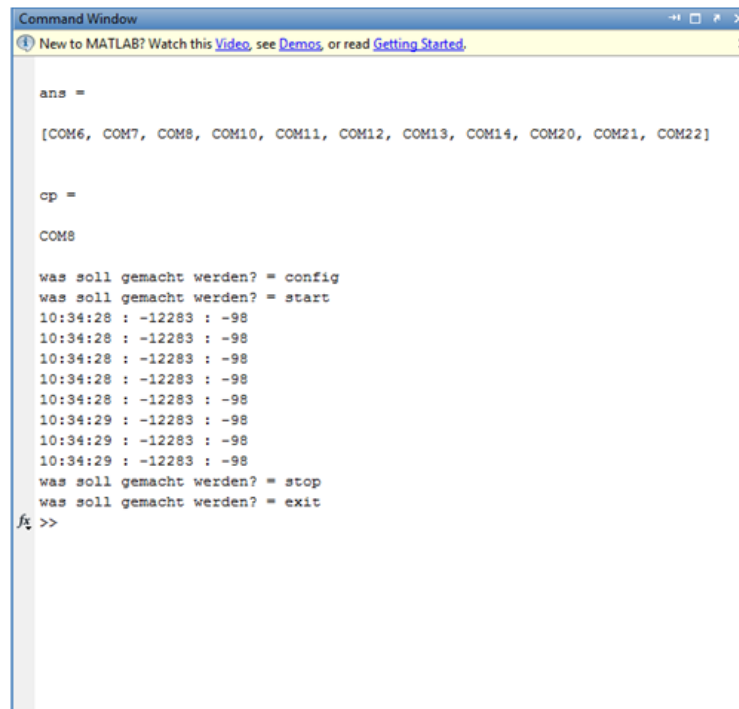
```
cp = 'COM8' % COM-Port der genutzt wird
```

3. Schritt: Im unteren Bereich des Files ist das if-Konstrukt indem man unter „config“ die Observables konfigurieren kann. Da MATLAB mit Hexadezimalzahlen nicht umgehen kann, müssen die Parameter-IDs (rot gekennzeichnet) erst in Dezimalzahlen gewandelt werden.

```
elseif (str.startsWith('config'))  
    conf.clearObservables();  
    % observable, int16, group 0, seq. 1, factor 1  
    % -> accXRaw  
    conf.addObservable(cast(53249, 'uint16'), 1);  
    % accYRaw  
    conf.addObservable(cast(53251, 'uint16'), 1);  
    % accZRaw  
    conf.addObservable(cast(53253, 'uint16'), 4);  
    %  
    % angR  
    conf.addObservable(cast(32776, 'uint16'), 1,  
        Configuration.FRAME_D);  
    % angP  
    conf.addObservable(cast(32779, 'uint16'), 1,  
        Configuration.FRAME_D);  
    conf.sendConfiguration();  
end
```



4. Schritt: Wird das m-File ausgeführt, kann man über den Befehl „config“ den Quadrokofter mit den gewünschten Observables konfigurieren und Kommunikation mit dem Befehl „start“ starten.



```
Command Window
New to MATLAB? Watch this Video, see Demos, or read Getting Started.

ans =

[COM6, COM7, COM8, COM10, COM11, COM12, COM13, COM14, COM20, COM21, COM22]

cp =

COM8

was soll gemacht werden? = config
was soll gemacht werden? = start
10:34:28 : -12283 : -98
10:34:28 : -12283 : -98
10:34:28 : -12283 : -98
10:34:28 : -12283 : -98
10:34:28 : -12283 : -98
10:34:29 : -12283 : -98
10:34:29 : -12283 : -98
10:34:29 : -12283 : -98
was soll gemacht werden? = stop
was soll gemacht werden? = exit
f1 >>
```

Abbildung 8: MATLAB-Konsole mit ausgeführtem Programm

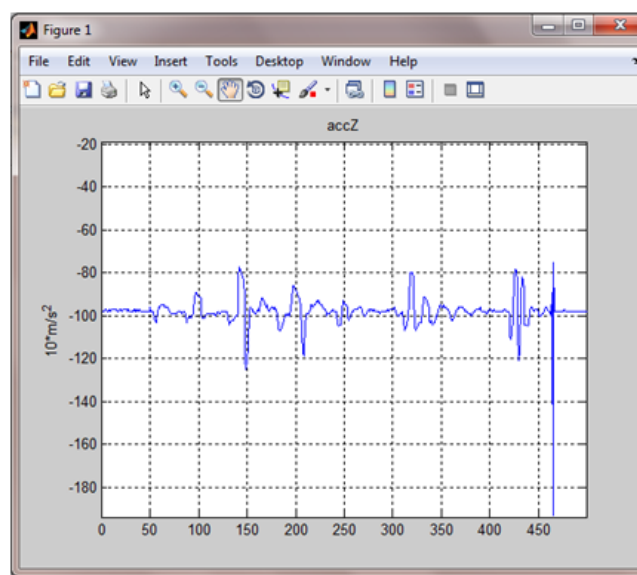


Abbildung 9: Beschleunigung in z-Richtung geplottet in MATLAB

## 4 Der Quadrokofter

### 4.1 Einführung

Der Quadrokofter ist eine Art des Helikopters, der im Gegensatz zum einrotorigen Helikopter vier starre Rotoren besitzt. Er wird nur durch die unterschiedlichen Rotordrehzahlen gesteuert bzw. stabil in der Luft gehalten. Im Weiteren wird erläutert welche Drehzahl-Verhältnisse, welche Lenk- oder Steuerungsbewegung ausführt [1].

Der Quadrokofter ist symmetrisch aufgebaut und die vier Rotoren sind jeweils in zwei 2er Paare aufgeteilt, d.h. ein Rotorpaar dreht im Uhrzeigersinn und das andere Rotorpaar dreht gegen den Uhrzeigersinn. Das hat den Grund, dass sich der Quadrokofter nicht dreht.

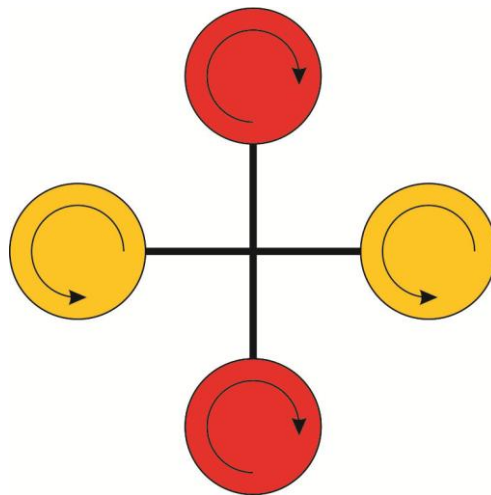


Abbildung 10: Quadrokofterschema

### 4.1.1 Earth- und Bodyframe

Die Beschreibung des Quadrokoopters erfolgt in zwei Koordinatensystemen. Wie in Abbildung 11 ersichtlich gibt es ein Koordinatensystem in dem der Quadrokoopter fliegt, das sogenannte Earthframe (grün) und ein zweites, das die direkten Bewegungen und Stellungen des Quadrokoopters aufzeigt, das sogenannte Bodyframe (rot). Ab hier ist der vordere Motor dunkelgrün gekennzeichnet.

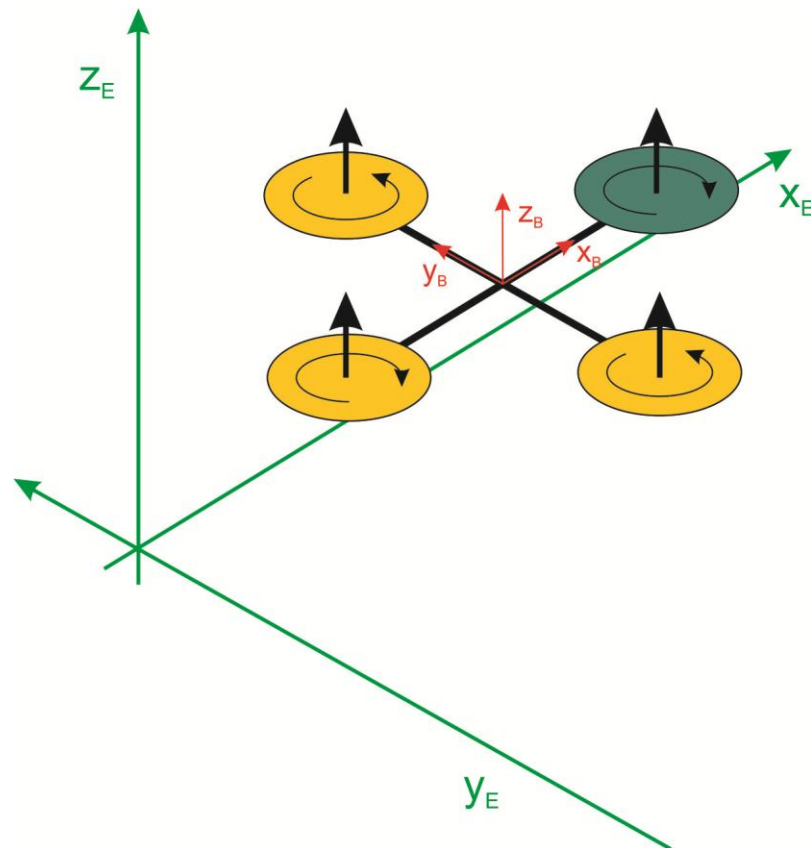


Abbildung 11: Das Earth- (grün) und Bodyframe (rot)

Das Earthframe beschreibt die Position des Quadrokoopters im Raum in x-, y- und z-Richtung (Punkt im Koordinatensystem). Allerdings sagt es nichts über die genaue Ausrichtung des Quadrokoopters aus. Diese Informationen liefert das Bodyframe, welches auf dem Quadrokoopter sitzt, in Kombination mit dem Earthframe. Die Bewegung des Quadrokoopters wird über Winkel, Drehgeschwindigkeiten und lineare Geschwindigkeiten der einzelnen Achsen beschrieben, die sich in Relation des Earthframes und Bodyframes zueinander berechnen lassen. Dies wird in Abbildung 12 verdeutlicht.

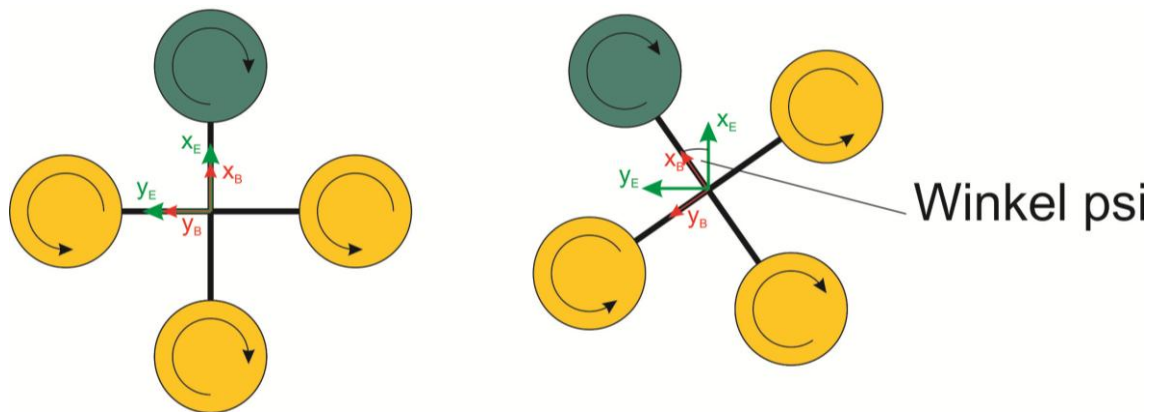


Abbildung 12: links der Quadrokopter mit ausgerichtetem Body- und Earthframe, rechts der um den Winkel  $\psi$  gedrehte Quadrokopter

Im weiteren Verlauf werden die Bewegungsmöglichkeiten des Quadropters dargestellt [1].

### 4.1.2 Auftrieb

Haben alle Motoren die gleiche erhöhte Drehzahl und somit die gleiche Winkelgeschwindigkeitsänderung  $\Delta\omega$ , steigt der Quadrokopter gleichmäßig in die Luft. Ab einem gewissen Punkt kann der Quadrokopter auch in der Luft stehen bleiben, das trifft dann ein, wenn die Rotoren genau die entgegengesetzte Beschleunigung zur Erdbeschleunigung von  $9,81 \text{ m/s}^2$  aufwenden.

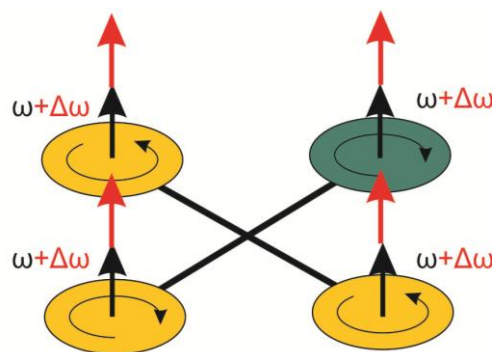


Abbildung 13: Auftrieb. [1]

### 4.1.3 Nicken

Um den Quadrokopter nach vorn oder nach hinten fliegen zu lassen, muss er nach vorn oder nach hinten geneigt werden, das heißt, dass die Drehzahl des vorderen oder des hinteren Motors erhöht oder gesenkt werden muss. In Abbildung 14 wird der Fall des Vorwärts-Fluges beschrieben.

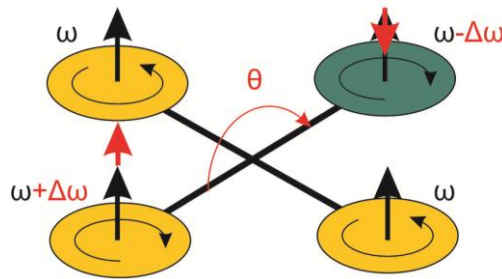


Abbildung 14: Nicken. [1]

### 4.1.4 Rollen

Der Seitwärts-Flug ist ähnlich dem Vorwärts-Flug, der Unterschied besteht darin, dass nun nicht mehr die Drehzahl des vorderen und hinteren Motors geändert wird, sondern die des linken und rechten Motors. In Abbildung 15 wird der Fall des Seitwärts-Fluges nach links dargestellt.

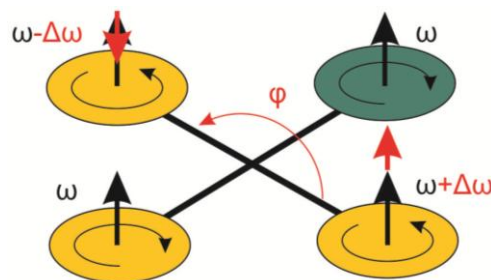


Abbildung 15: Rollen. [1]

### 4.1.5 Gieren

Beim Gieren wird die Quadrokofter-Bauweise ausgenutzt, d.h. die jeweilig entgegengerehenden Rotorpaare erhöhen die Drehzahl. Soll der Quadrokofter nach rechts drehen, wird die Drehzahl des rechten und linken Motors erhöht, dabei erhöht sich das Drehmoment im Uhrzeigersinn und der Quadrokofter dreht sich im Uhrzeigersinn. Soll der Quadrokofter entgegengesetzt des Uhrzeigersinnes drehen, wird die Drehzahl des vorderen und hinteren Motors erhöht. Dies wird in Abbildung 16 veranschaulicht.

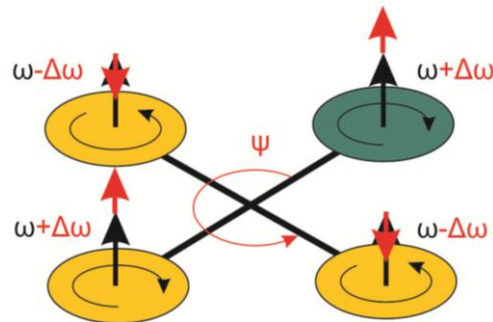


Abbildung 16: Gieren. [1]

## 4.2 Platine der Fluglageregelung

Die gesamte Steuerung des Quadrocopters erfolgt mit Hilfe der Fluglageregelung. Auf dieser Platine befinden sich alle nötigen Sensoren um die Winkel und Winkelgeschwindigkeiten zu messen, die zum Berechnen der Fluglage benötigt werden.

### 4.2.1 Sensoren

Zu den benötigten Sensoren auf der Fluglageregelung gehören 3 Gyroskope und 1 Beschleunigungssensor.

*Gyroskope:* Auf der Platine sind 3 Gyroskope für die x-, y- und z-Achse vorhanden. Sie liefern die Winkelgeschwindigkeit der Winkel  $\theta$  (nicken),  $\phi$  (rollen) und  $\psi$  (gieren).



Abbildung 17: Gyroskop

*Beschleunigungssensor:* Der Beschleunigungssensor ist einmal vorhanden, da dieser die lineare Beschleunigung sowohl in x-, y- als auch in z-Richtung messen kann.



Abbildung 18: Beschleunigungssensor

Die gemessenen rotatorischen Geschwindigkeitswerte der Gyroskope werden direkt zur Berechnung verwendet. Zur Berechnung der Winkel  $\theta$  und  $\phi$  werden allerdings noch die translatorischen Beschleunigungswerte des linearen Beschleunigungssensors benötigt. Der Winkel von  $\psi$  wird nicht benötigt, da dieser für die Regelung nicht relevant ist und vom Anwender gesteuert wird.

### 4.2.2 Die alte Platine der Fluglageregelung

Die erste Version der Platine wurde seit Bestehen des Projekts verwendet. Man bemerkte allerdings einen Layout-Fehler, der durch fehlerhafte Bauteilmerkmale in der Layout-Software entstanden ist.

Die Layoutänderung war allerdings zu spät, weshalb man den Fehler manuell auf der Platine mit Drähten ausgebessert hat.

Nach dieser Korrektur war der Quadrocopter flugfähig. Die Software des Quadrocopters entstand in einer Master-Thesis, die von 4 Master-Studenten entwickelt wurde. Sie besaß einen PID-Regler für die Lageregelung.

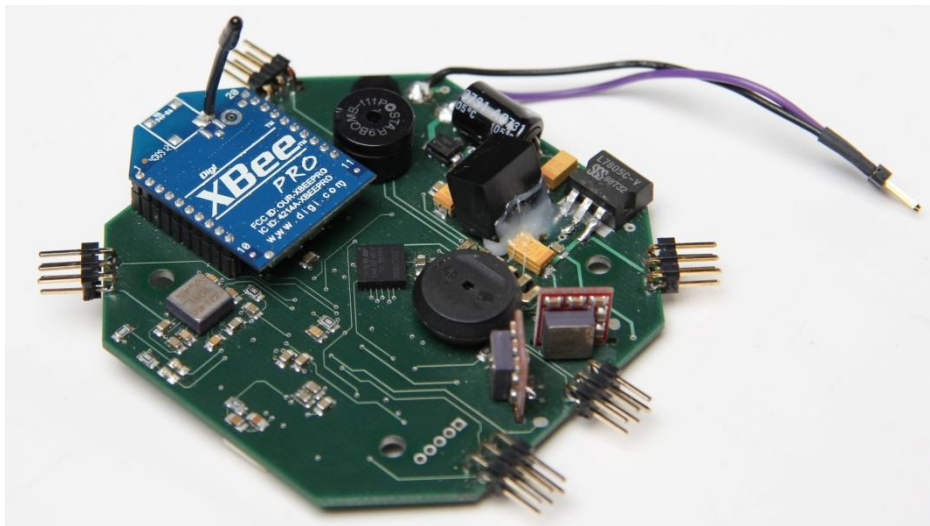


Abbildung 19: Alte Flight-Control-Platine



### 4.2.3 Die neue Platine der Fluglageregelung

Im Laufe dieser Arbeit traf die neue, verbesserte Version der Platine der Fluglageregelung ein. Im Folgenden werden die Unterschiede der alten und der neuen Version der Platine erläutert. Der größte Unterschied besteht darin, dass die neue Version nur noch auf einer Seite bestückt ist. Die alte Platine in Abbildung 19 wurde noch auf beiden Seiten bestückt. Außerdem wurde der Layoutfehler behoben, sodass es keine zusätzliche Verdrahtung nötig ist. Die Drähte, die man in Abbildung 20 erkennen kann, dienen nur der Messung der Gyroskope, mit einem Voltmeter.



Abbildung 20: Neue Flight-Control-Platine

Vergleich der alten Version zur neuen Version		
Sensorwerte	Alte Platine	Neue Platine
accXRaw	1	4
accX	0	0
accYRaw	-5	-12
accY	0	0
accZRaw	348	360
accZ	-97	-98
angVelRRaw	521	506
gyroROffset	521	506
angVelR	0	0
angR	-0,002921282	0
angVelPRaw	518	501
gyroPOffset	518	501
angVelP	0	0
angP	0,00283049	0
angVelYRaw	562	6
gyroYOffset	562	1
angVelY	0	0
tempRaw	514	604
temp	26	81
airPressureRaw	14	488

Tabelle 7: Gemessene Sensorwerte

In Tabelle 7 werden alle Sensorwerte als Rohdaten und berechnete Daten dargestellt. Die Beschleunigungswerte *accX*, *accY*, *accZ* werden jeweils in  $10 \cdot \text{m/s}^2$  angegeben. Die Winkelgeschwindigkeitswerte *angVelP*, *angVelR*, *angVelY* werden jeweils in  $100 \cdot \text{rad/sec}$  und die Winkel *angP* und *angR* in  $10000 \cdot \text{rad}$  angegeben.

Alle Werte scheinen bei beiden Platinen, mal abgesehen von kleinen Abweichungen, gleich zu sein. Allerdings sieht man im unteren Drittel der Tabelle, dass die Werte des Gyroskops der z-Achse (Gieren), in Tabelle 7 rot gekennzeichnet, deutlich abweichen. Aber auch die Temperatur weicht deutlich ab, das liegt daran, dass die Temperatur ebenfalls vom Gyroskop der z-Achse ausgelesen wird. Weitere Messungen mit dem Voltmeter ergaben bei einem funktionierenden Sensor, im ruhigen Zustand eine Spannung von 1,6V, was dem Mittelwert der Referenzspannung von 3,3V entspricht. Die Messung am defekten Sensor ergab eine Spannung von 0V weder im ruhigen noch aktiven Zustand. Das bestätigte, dass es keine Unterbrechung zwischen Mikrokontroller und Sensor gibt. Allerdings konnte man den Fehler noch nicht weiter eingrenzen, da es noch keine weitere Platine der neuen Version gibt. Aus diesem Grund wurde für den weiteren Verlauf dieser Arbeit die alte Version der Platine verwendet.

### 4.3 Sensorkalibrierung

Unter der Sensorkalibrierung versteht man das Messen und Festlegen der Abweichungen der Sensorwerte, das für die anschließende Regelung sehr wichtig ist. Die Sensoren sind fertigungstechnisch nicht 100 prozentig genau auf der Platine ausgerichtet, was durch die Kalibrierung ausgeglichen wird.

Beim kalibrieren der Sensoren muss der Quadrokofter auf einer ebenen Fläche stehen, denn das ist die Lage, die der Quadrokofter während des Fluges halten muss. Der eigentliche Kalibriervorgang ist das Abspeichern der Sensorwerte. In der Regelung werden diese Werte dann als Bezugswerte benutzt.

#### 4.3.1 Feste Sensorwerte

In der ersten Version der Software wurden die Sensorwerte auf einer ebenen Fläche gemessen und fest in den Code geschrieben. Diese Variante funktioniert nur bei der Platine, bei der auch gemessen wurde, da jede Platine etwas andere Werte besitzt.

#### 4.3.2 Kalibrierung mit flüchtigem Speicher

Die zweite Version der Quadrokoftersoftware besaß eine Kalibrierung, die bei jedem Einschalten des Quadrokofters ausgeführt wurde. Das heißt der Quadrokofter musste bei jedem Einschaltvorgang auf einer ebenen Fläche stehen, was zu Problemen führte, wenn man keine ebene Fläche hatte. Der große Vorteil dieser Variante im Vergleich zur Version mit festen Sensorwerten, es konnten auch mit verschiedenen Platinen ohne Probleme kalibriert werden.

#### 4.3.3 Kalibrierung mit EEPROM-Speicher

Die aktuelle Version besitzt eine Kalibrierung, bei der man den Quadrokofter optimal ausrichtet und die gemessenen Werte in dem nicht flüchtigen EEPROM-Speicher abspeichert. Diese Version hat den großen Vorteil, dass man nur einmal auf eine ebene Fläche angewiesen ist.

Der Kalibriervorgang wird durch die in Abbildung 21 dargestellte Hebelstellung der Fernbedienung gestartet. Dies entspricht im C-Code einer einfachen if-Abfrage:

```
if((state.remoteMotorsOn == FALSE) &&  
   (state.remoteForceRaw >= 245) &&  
   (state.remoteYawRaw <= 30) &&  
   (state.remotePitchRaw <= 30) &&  
   (state.remoteRollRaw <= 30))
```



Abbildung 21: Hebelstellung für den Kalibriervorgang

Dabei muss darauf geachtet werden, dass die Trimmung wie in Abbildung 22 jeweils auf die Mittelstellung gebracht wird.

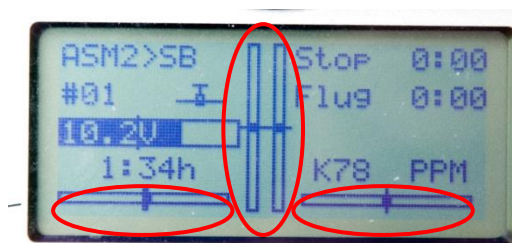


Abbildung 22: Trimmung

Die Kalibrierung läuft wie folgt ab:

1. Der Quadrocopter wird mit einer Libelle optimal ausgerichtet
2. Die Kalibrierhebelstellung wird auf der Fernbedienung eingestellt
3. Warten bis der Quadrocopter piept
4. Den Quadrocopter nochmals AUS und wieder EIN schalten

Ab diesem Zeitpunkt sind die Werte solange im EEPROM gespeichert, bis das EEPROM beim nächsten Software-Flash-Vorgang gelöscht wird.

**Nach einem Software-Flash-Vorgang muss neu kalibriert werden!**

### 4.3.3.1 Implementierung

Bei der Kalibrierung mit flüchtigem Speicher wurde der Kalibrierungsvorgang noch vor dem Hauptprogramm mit der Endlosschleife ausgeführt. Dies funktioniert aber nur nach jedem Einschalten, bei dem der Quadrokopter initialisiert wird.

Bei der Kalibrierung mit dem EEPROM-Speicher, kann deshalb nicht außerhalb der Endlosschleife des Hauptprogramms, kalibriert werden.

Das Hauptprogramm ist in verschiedene Zeitbereiche gegliedert, d.h. es gibt Funktionen, die oft aufgerufen werden müssen und welche die zeitunkritischer sind. Zu den zeitkritischen Funktionen gehören *copterActualizeState()*, *myFilter()*, *flightControl()* und *copterSetRpms()*. Diese Funktionen beinhalten die Lageregelung sowie die Steuerung der Motoren. Sie werden alle 5 Millisekunden aufgerufen, um den Quadrokopter sicher in der Luft halten zu können. Zu den weniger zeitkritischen Funktionen gehört die Übertragung der Sensorwerte über das XBee-Modul. Diese wird alle 20 Millisekunden aufgerufen. Zu den zeitunkritischen Funktionen zählen die *copterOnBoardDiagnosis()*, die z.B. die Batteriespannung überwacht, und die Kalibrierung mit fester Speicherung mit der Funktion *allCalibrate()*. Diese werden alle 1000 Millisekunden aufgerufen.

Bei der Kalibrierung kommt es nicht auf Performance an, deshalb kann diese Funktion in den zeitunkritischen Bereich der Endlosschleife.

Ein Nachteil der festen Speicherung ist, dass die Regelung beim ersten Einschalten nach einem Software-Flash-Vorgang noch keine kalibrierten Werte hat. In dieser Zeit sind die Abweichungen sehr groß, was sich beim Starten der Motoren durch die maximale Drehzahl äußert.

Aus diesem Grund muss der Quadrokopter nach dem ersten Kalibrieren nach einem Flash-Vorgang neu gestartet werden, um mit den richtig kalibrierten Werten zu regeln.

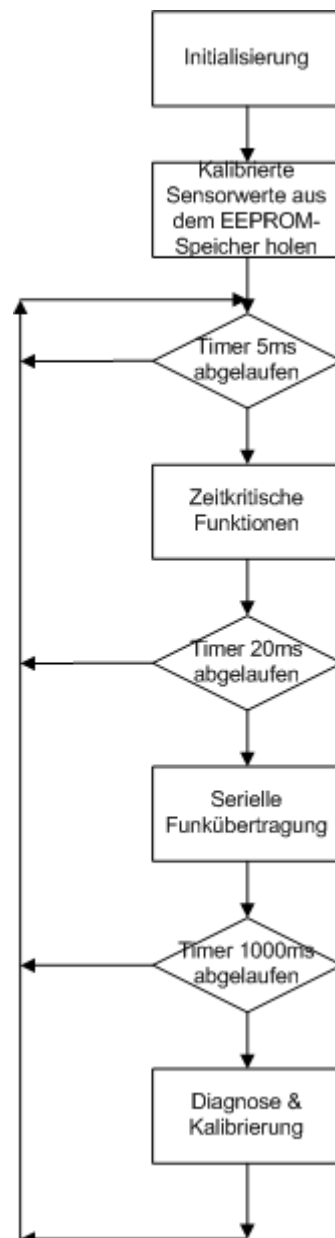


Abbildung 23: Ablauf der Main-Funktion

## 5 Der Zustandsregler

### 5.1 Einführung

Zustandsregler sind eine Möglichkeit Prozesse in der Zustandsraumdarstellung zu regeln. Bei diesem Verfahren wird der Zustand des Prozesses durch Messung oder einen Beobachter des Prozesses zurückgeführt. Deshalb wird dieses Verfahren auch Regelung durch vollständige Zustandsrückführung genannt. Es eignet sich zur Regelung nichtlinearer, zeitvariante Systeme sowie Mehrgrößensysteme.

Die mathematische Darstellung des Systems erfolgt in 4 Zustands-Matrizen.

<b>A</b>	Systemmatrix
<b>B, b</b>	Eingangsmatrix, Eingangsvektor
<b>C, c<sup>T</sup></b>	Ausgangsmatrix, Ausgangsvektor
<b>D, d</b>	Durchgangsmatrix, Durchgangsvektor

Tabelle 8: Zustandsmatrizen

Die Zustands-Matrizen eines SISO-Systems zweiter Ordnung werden so dargestellt:

$$\begin{array}{c}
 \begin{array}{c} x1' \\ x2' \\ y \end{array}
 \begin{bmatrix}
 & x1 & x2 & u \\
 A & A & b \\
 A & A & b \\
 c & c & d
 \end{bmatrix}
 \end{array}$$

Die Systemmatrix  $A$  enthält alle Koeffizienten der Zustandsvariablen. Diese Matrix beschreibt das System und seine Charakteristik, außerdem lässt sich feststellen ob das System stabil ist. Die Stabilität des Systems lässt sich prüfen indem man die Eigenwerte der Matrix berechnet.

Die Eingangsmatrix  $B$  oder der Eingangsvektor  $b$  beschreibt die Eingänge des Systems.

Die Ausgangsmatrix  $C$  oder der transponierte Ausgangsvektor  $c^T$  beschreibt die Ausgänge des Systems.



Die Durchgangsmatrix  $D$  oder der Durchgangsvektor zeigt ob das System eine direkte Verbindung zwischen Ein- und Ausgang besitzt. Bei den meisten besteht keine direkte Verbindung zwischen dem Ein- und Ausgang, deshalb ist diese Matrix „0“.

### 5.1.1 Einfaches System

Im Folgenden wird ein einfaches SISO-System [1], dritter Ordnung beschrieben. SISO bedeutet Single Input Single Output, d.h. das System hat nur einen Eingang und einen Ausgang.

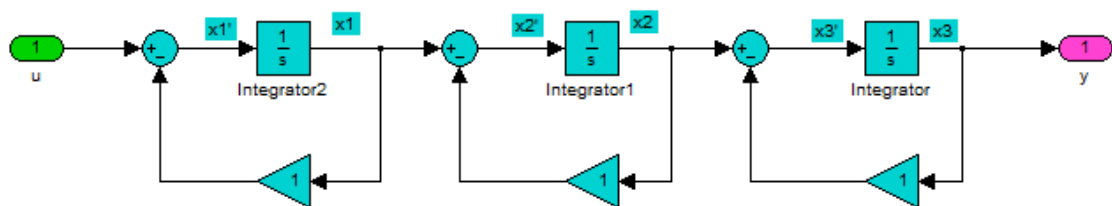


Abbildung 24: Einfaches SISO-System dritter Ordnung. [1]

In Abbildung 24 sehen Sie ein einfaches SISO-System mit 3 Integratoren, d.h. das System ist dritter Ordnung. Das System wird mathematisch in Differenzialgleichungen dargestellt.

Die Differenzialgleichungen lauten:

$$\dot{x}_1 = -x_1 + u$$

$$\dot{x}_2 = x_1 - x_2$$

$$\dot{x}_3 = x_2 - x_3$$

$$y = x_3$$

Das ergibt die Matrix :

	$x_1$	$x_2$	$x_3$	$u$
$\dot{x}_1$	-1	0	0	1
$\dot{x}_2$	1	-1	0	0
$\dot{x}_3$	0	1	-1	0
$y$	0	0	1	0

### 5.1.2 Steuerbarkeit

Der Begriff der Steuerbarkeit spielt in der Regelkreisanalyse eine entscheidende Rolle, denn Zustandsregelungen sind nur dann realisierbar, wenn die Stellgröße auf alle Zustandsvariablen wirkt und diese beeinflusst.

Ob ein System steuerbar ist oder nicht, kann man mit Hilfe der Steuerbarkeitsmatrix  $Q_S$  berechnen.

$$Q_S = [b \quad A * b \quad A^2 * b \quad \dots \quad A^{n-1} * b] \quad (1)$$

Die Steuerbarkeit ist nur dann gegeben, wenn die Steuerbarkeitsmatrix  $Q_S$  den vollen Rang besitzt.

#### MATLAB-Befehl:

Zum berechnen der Steuerbarkeitsmatrix

$$Q_S = \text{ctrb}(\text{Sys}) \quad \text{oder} \quad Q_S = \text{ctrb}(A, b)$$

Rang überprüfen

$$\text{rang} = \text{rank}(Q_S)$$

### 5.1.3 Beobachtbarkeit

Bei einem Zustandsregler, der ein reales System regelt kommt es häufig vor, dass nicht alle Zustandsgrößen gemessen werden. Das heißt die fehlenden Zustandsgrößen müssen im Rechner simuliert (nachgebildet) werden. Um zu testen ob das System mit einem solchen Beobachter versehen werden kann, muss man das System auf Beobachtbarkeit testen. Dies berechnet man mit der sogenannten Beobachtbarkeitsmatrix  $Q_B$ .

$$Q_B = [c^T \quad c^T * A \quad c^T * A^2 \quad \dots \quad c^T * A^{n-1}]^T \quad (2)$$

Die Beobachtbarkeit ist nur dann gegeben, wenn die Beobachtbarkeitsmatrix  $Q_B$  den vollen Rang besitzt.

**MATLAB-Befehl:**

Zum berechnen der Beobachtbarkeitsmatrix

$$Q_B = \text{obsv}(\text{Sys}) \text{ oder } Q_B = \text{obsv}(A, b)$$

Rang überprüfen

$$\text{rang} = \text{rank}(Q_B)$$

**5.1.4 Zustandsregler**

In diesem Unterkapitel wird das einfache System [1] aus Unterkapitel 5.1.1 mit einem Zustandsregler geregelt. Dazu wird von jeder Zustandsvariablen eine Rückführung mit einem Rückföhrfaktor angefügt. Um bleibende Regelabweichungen entgegenzuwirken wird zwischen dem Systemeingang und der Summationstelle der Rückföhrzweige ein einfaches Proportional-Glied als Vorfilter eingefügt.

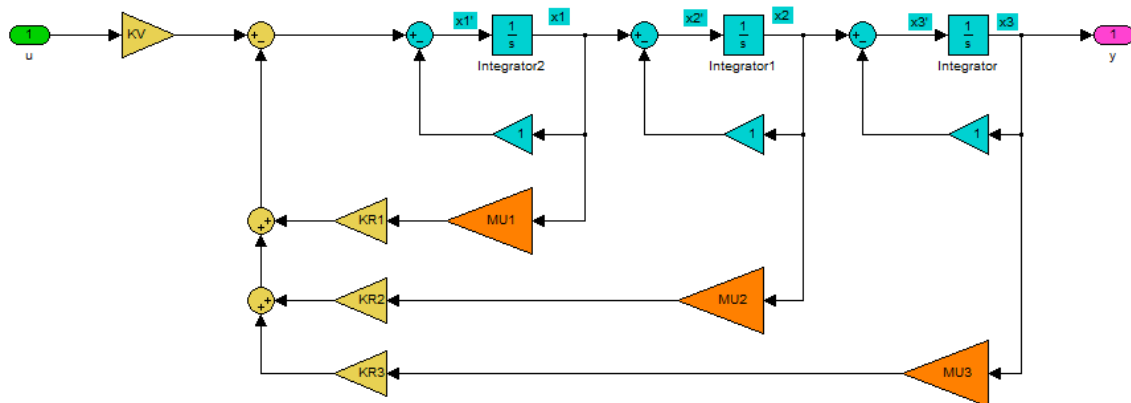


Abbildung 25: Einfaches SISO-System mit Zustandsregler. [1]

**5.1.5 Pole-Placement**

Das Pole-Placement-Verfahren ist eine Möglichkeit einen Zustandsregler auszu-legen. Dabei werden die Pole des Systems nach eigenem Ermessen in den stabilen Bereich der s-Ebene (siehe Abb. 26) verschoben. Je weiter die Pole in die linke s-Halbebene verschoben werden umso schneller wird das System. Allerdings muss auch beachtet werden, dass die Pole nicht zu weit links liegen, denn dadurch müsste die Stellgröße eine hohe Verstärkung aufweisen. Dies kann so weit gehen, dass man unendlich Energie in das System stecken müsste. Im Fall des Quadropters sind die Motoren der begrenzende Faktor. Diese können nur begrenzt schnell auf die gewünschte Drehzahl gebracht werden.

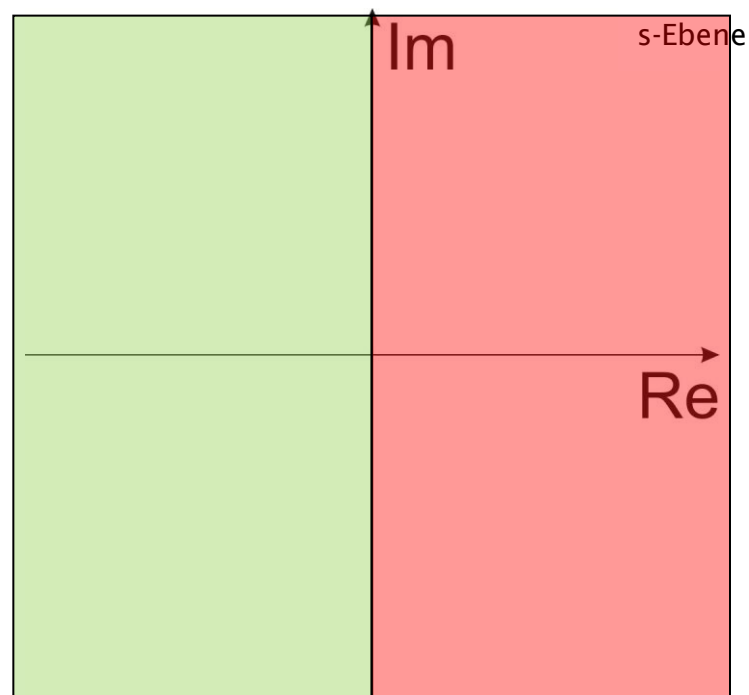


Abbildung 26: s-Ebene: links stabil, rechts nicht stabil

Im Folgenden wird die Anwendung von Pole-Placement erläutert.

In Abbildung 27 sieht man einen Prozess, der nicht geregelt wird also im „open-loop“ Betrieb ist.

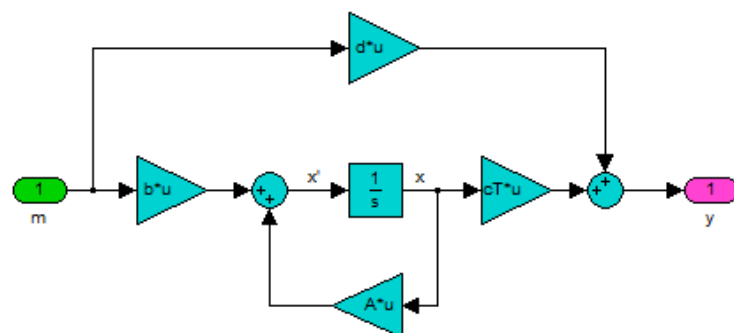


Abbildung 27: Prozess im „open-loop“-Betrieb mit direktem IO-Durchgriff . [3]

Da in den meisten Fällen kein direkter Ein-Ausgangs-Durchgriff vorhanden ist wird  $d = 0$  gesetzt.

Daraus entsteht das in Abbildung 28 abgebildete System.

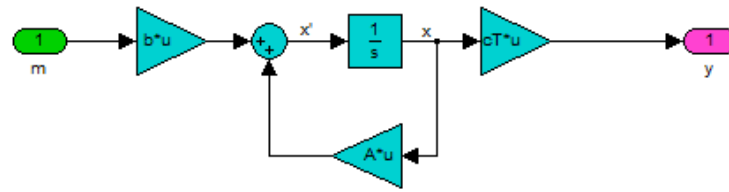


Abbildung 28: Prozess im „open-loop“-Betrieb ohne direkten IO-Durchgriff. [3]

Dieses „open-loop“-System, d.h. ohne Regelung, besitzt folgende Gleichung:

$$\underline{x}'(t) = \underline{A} * \underline{x}(t) + b * m(t) \quad (3)$$

$$y(t) = \underline{c}^T * \underline{x}(t) \quad (4)$$

Wählt man den Rückführparameter  $\underline{k} = [K_1 \ K_2 \ K_3]^T$ , so ergibt das, folgende Gleichung für den Eingang des Systems:

$$m(t) = Kv * w(t) - \underline{k}^T * \underline{x}(t) \quad (5)$$

Daraus ergeben sich die Gleichungen des „closed loop“-System, d.h. mit Regelung:

$$\underline{x}'(t) = (\underline{A} - b * \underline{k}^T) * \underline{x}(t) + b * Kv * w(t) \quad (6)$$

$$y(t) = \underline{c}^T * \underline{x}(t) \quad (7)$$

Durch die Rückführung und den Vorfilter entsteht ein neues System mit der neuen Systemmatrix  $\underline{A}_{CL} = \underline{A} - b * \underline{k}^T$ .

Deren Pole können nun durch den Zustandsrückführungsvektor  $\underline{k}^T$  nach Belieben verändert werden.

### **MATLAB-Befehl:**

Zur Berechnung des Zustandsrückführungsvektors  $k^T$ :

$$\mathbf{k} = \text{place}(\mathbf{A}, \mathbf{b}, \mathbf{p\_soll})$$

Die bleibende Reglerabweichung wird mit dem Vorfilter  $K_v$  ausgeglichen. Dieser wird wie folgt berechnet:

$$K_v = -[c^T * (A - b * k^T)^{-1} * b]^{-1} \quad (10)$$

### **MATLAB-Code:**

In MATLAB wird diese Formel wie folgt eingegeben:

$$K_v = -\text{inv}(c^T * \text{inv}(A - b * k') * b)$$

### **5.1.6 Zustandsregler mit Beobachter**

Ist es in einem System nicht möglich alle Zustandsgrößen zu messen oder sind Sensoren zur Messung durch Kosteneinsparung nicht vorhanden, kann man zur Regelung einen Zustandsregler mit einem Beobachter verwenden.

Dabei simuliert der Beobachter die Zustände, die nicht gemessen werden können. Der Beobachter selbst ist quasi eine Nachbildung des realen Systems. Er benötigt selbst zur Simulation des Systems dessen Eingang und Ausgang.

Dieses Prinzip wurde von Herrn David Luenberger entwickelt, weshalb es auch seinen Namen Luenberger-Zustandsbeobachter trägt.

Im Folgenden wird das System aus Unterkapitel 5.1.1 mit der Erweiterung eines Beobachters beschrieben.

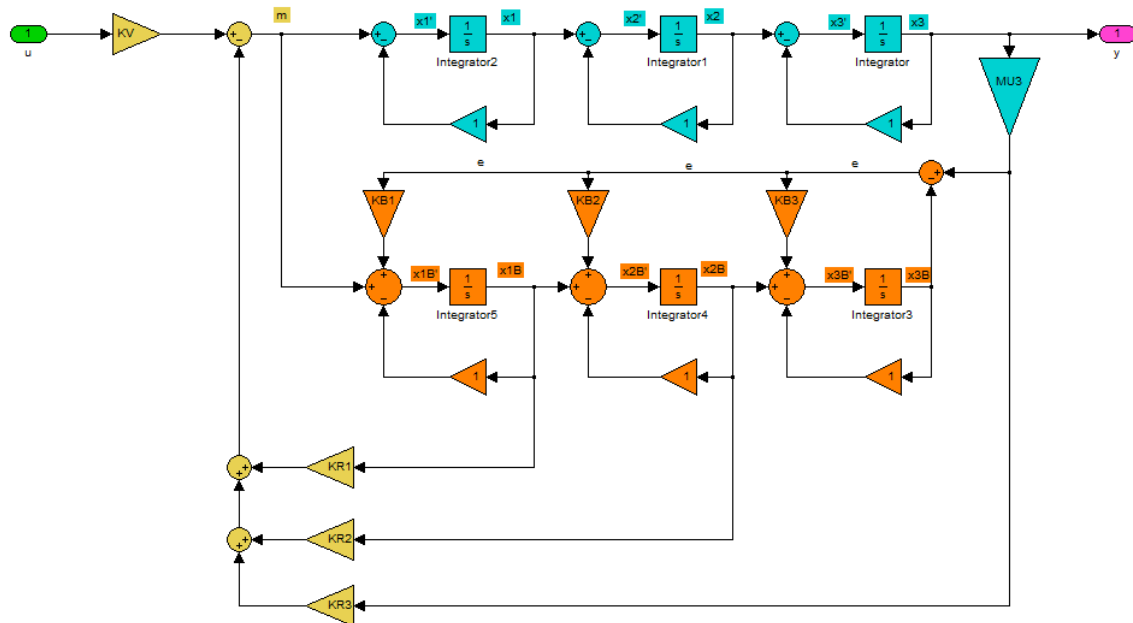


Abbildung 29: Einfaches SISO-System dritter Ordnung mit Zustandsregler und Beobachter. [1]

In Abbildung 29 besitzt das SISO-System nur noch einen Messumformer *MU3* (Sensor), wird aber trotzdem mit dem gleichen Zustandsregler geregelt wie im Unterkapitel 5.1.4. Dies ist nur möglich, wenn ein Zustands-Beobachter eingefügt wird. Dieser Zustands-Beobachter ist orange eingefärbt und bildet das reale System nach. Wie man erkennen kann muss der System-Ein- und Ausgang an den Beobachter geleitet werden.

Der Quadrocopter benötigt aber keinen Zustands-Beobachter, da alle Zustandsvariablen mit Sensoren messbar sind. Deshalb wird dieser auch nicht weiter erläutert.

## 5.2 Quadrocopter in MATLAB-Simulink

Ein wichtiger Bestandteil bei der Entwicklung des Quadrocopters ist die parallele Entwicklung und Simulation in MATLAB-Simulink. Der komplette Quadrocopter ist als MATLAB-Simulink-Modell entworfen worden. An diesem Modell wird unter anderem der Zustandsregler berechnet, implementiert und getestet.

### 5.2.1 Gesamtes MATLAB-Simulink-Modell

Das komplexe MATLAB-Simulink-Modell ist in fünf Bereiche unterteilt (Abb. 30), der Fernsteuerungsteil mit grün gekennzeichnet, der Regelungsblock mit gelb gekennzeichnet, der Dynamics-Block mit cyan gekennzeichnet, der Copter-Animation-Block mit magenta gekennzeichnet und der Sensor-Block mit orange gekennzeichnet.

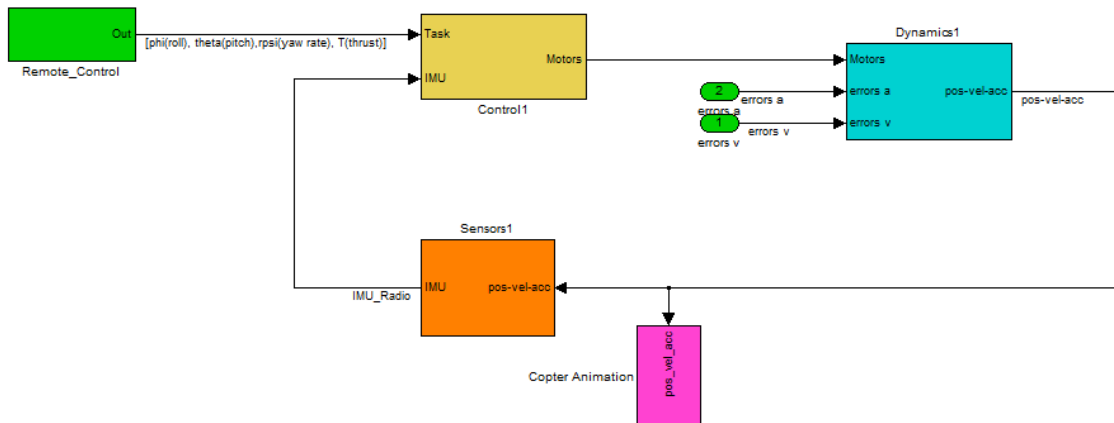


Abbildung 30: Gesamtsystem des Quadropters in MATLAB-Simulink. [1]

Der Remote-Block erzeugt einen Vektor, der die vier Fernsteuersignale, die sogenannten Sollwerte, enthält. Der Vektor besteht aus:

1. Dem Winkel  $\phi$  (roll)
2. Dem Winkel  $\theta$  (pitch)
3. Der Winkelgeschwindigkeit  $r\psi$  (yaw rate)
4. Der durchschnittliche Auftriebskraft (thrust)

Diese Sollwerte werden dann, im sogenannten Control-Block, weiter verarbeitet. In diesem Control-Block befinden sich unter anderem der zu optimierende Zustandsregler und eine Umwandlung in Pseudo-Kräfte, die dann weiter an den Dynamics-Block geleitet werden.

Im Dynamics-Block befindet sich das physikalische Modell des Quadropters mit den Eigenschaften des realen Quadropters. Außerdem können sämtliche Störgrößen aufgeschaltet werden. Am Ausgang des Dynamic-Blocks wird ein größerer Vektor Namens „pos-vel-acc“ ausgegeben, der für die Quadropter-Animation und für den Sensor-Block verwendet wird. Dieser Vektor enthält alle Koordinaten, Winkelbeschleunigungs- und Geschwindigkeitswerte, sowie die Winkel selbst, die für die Animation und weitere Berechnung nötig sind. Im Sensor-Block werden diese Werte dann quantisiert und wenn nötig mit einem zuschaltbaren Sensorrauschen versehen. Diese Sensorwerte werden dann wie-



der in den Control-Block geleitet um damit eine reale Reglerumgebung zu simulieren.

Im Weiteren werden die einzelnen Blöcke genauer erläutert.

### 5.2.2 Remote-Control-Block

In diesem Unterkapitel wird der Remote-Control-Block genauer betrachtet. Wie man in Abbildung 31 erkennen kann, ist es möglich die Simulation mit einer 3D-Maus die an den Computer angeschlossen wird (in gelb eingefärbt), oder mit vordefinierten Steuersignalen (in cyan eingefärbt) zu steuern.

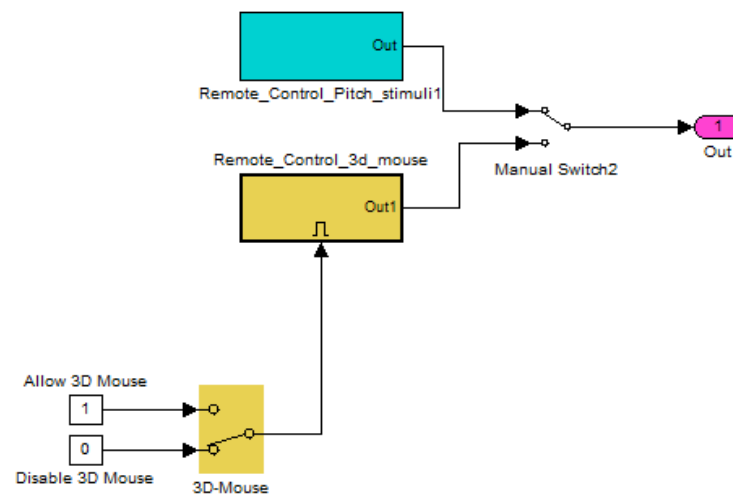


Abbildung 31: Remote-Control-Block. [1]

In dieser Arbeit wurde nur der Block mit den vordefinierten Steuersignalen verwendet, deshalb wird dieser Block in Abbildung 32 genauer betrachtet.

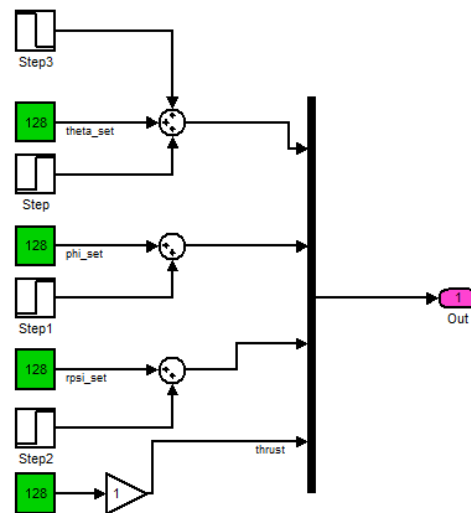


Abbildung 32: Remote-Control-Stimuli. [1]

Die grün gefärbten Blöcke entsprechen den Werten, die die Fernsteuerung sendet. Der erste Wert steht für den Soll-Winkel  $\theta$ , der zweite Wert steht für den Soll-Winkel  $\phi$ , der dritte Wert steht für die Soll-Winkelgeschwindigkeit  $r_{\psi}$  und der letzte Wert steht für die Drehzahl. Die restlichen Blöcke dienen der Zu- und Abschaltung der einzelnen Werte, indem sie nach unterschiedlichen Zeiten wirken. Der jeweilige Wert von 128 entspricht jeweils der Mittelstellung der Fernsteuerung deren Werte von 0 bis 255 annehmen können.

### 5.2.3 Control-Block

Die nachfolgende Abbildung 33 zeigt die möglichen Regler, die durch die Kippschalter ausgewählt werden können. Als Eingangssignale bekommen alle Blöcke dieselben.

Vom Sensor (IMU):

1.  $a_x$ : Beschleunigung in x-Richtung
2.  $a_y$ : Beschleunigung in y-Richtung
3.  $r_{\phi}$ : Winkelgeschwindigkeit des Winkels  $\phi$  (Rollen)
4.  $r_{\theta}$ : Winkelgeschwindigkeit des Winkels  $\theta$  (Nicken)
5.  $r_{\psi}$ : Winkelgeschwindigkeit des Winkels  $\psi$  (Gieren)

Von der Fernsteuerung (Task):

1.  $\phi$ : Sollwert des Winkels  $\phi$  (Rollen)
2.  $\theta$ : Sollwert des Winkels  $\theta$  (Nicken)
3.  $\dot{\psi}$ : Sollwert der Winkelgeschwindigkeit des Winkels  $\psi$  (Gieren)
4. T: Sollwert der Motorenschubkraft

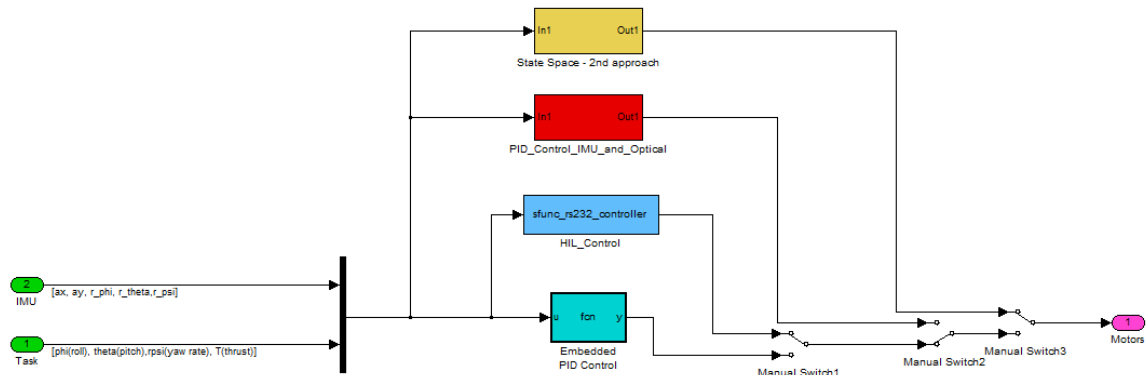


Abbildung 33: Control-Block

**Vorhandene Reglerblöcke:**

1. **State Space Controller (gelb):** Beinhaltet den Zustandsregler, der auch auf dem Quadrokofter realisiert ist. Dieser wird in Kapitel 5.3 genauer betrachtet.
2. **PID\_Control\_IMU\_and\_Optical (rot):** Beinhaltet den PID-Regler aus der Masterarbeit, welcher aber in dieser Arbeit nicht weiter erläutert wird.
3. **HIL\_Control (hellblau):** Mit Hilfe dieses Blocks ist es möglich den Quadrokofter mit dem Hardware-in-the-Loop zu testen. Dabei wird eine serielle Funkverbindung aufgebaut, mit der der Quadrokofter vom Rechner aus gesteuert wird.
4. **Embedded PID Control (cyan):** Enthält den PID-Regler als C-Funktion, die auch im realen Quadrokofter verwendet werden kann.

Am Ausgang jedes Regler-Blocks werden die pseudo-Kräfte für jeden Rotor ausgegeben.

### 5.2.4 Dynamics-Block

Der Dynamics-Block enthält das physikalische Modell des Quadropters. Abbildung 34 zeigt den ganzen zu regelnden Prozess. Dieser wird zur Erläuterung in vier Bereiche unterteilt und im Folgenden genauer beschrieben.

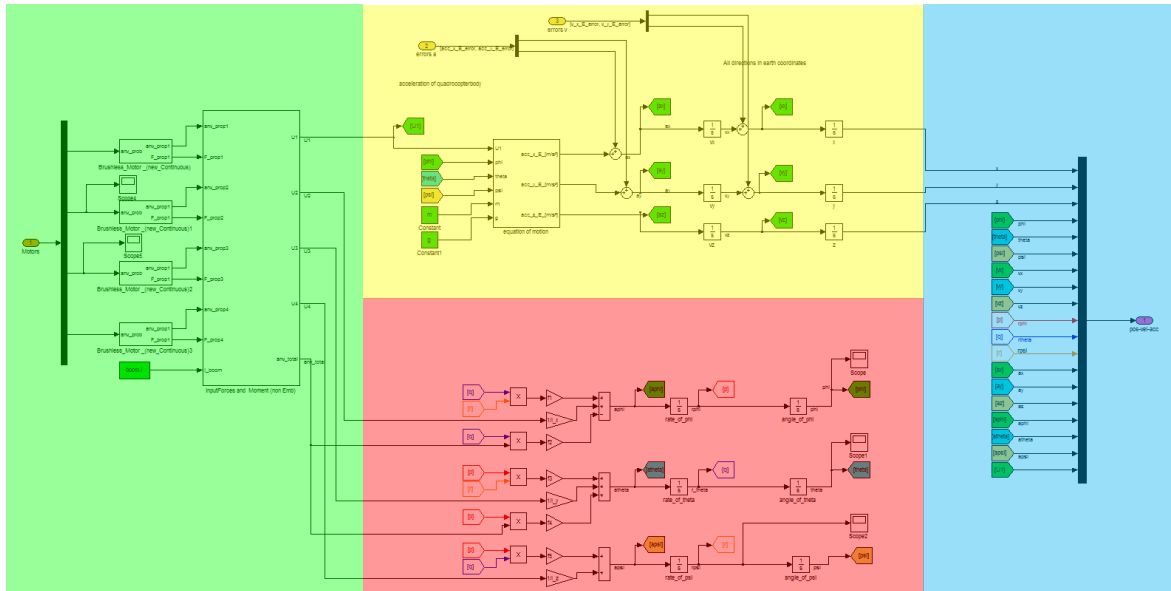


Abbildung 34: Dynamics-Block. [1]

#### Dynamikmodell der Motoren (grüner Bereich)

Der grüne Bereich enthält die Dynamik der Motoren und erhält als Eingangsvektor die Pseudo-Kräfte aus dem vorangegangenen Control-Block. Diese Pseudo-Kräfte werden auf die jeweiligen Brushless-Motoren geleitet und mit Hilfe des „*setpoint\_thrust\_prop*“-Blocks in die charakteristische reale Kraft umgesetzt. Der erste Ausgang „*F\_prop*“ entspricht der berechneten Kraft. Der Wert des zweiten Ausgangs wird im „*thrust\_speed\_prop*“-Block berechnet und entspricht der Winkelgeschwindigkeit des Propellers.

#### Earthframe des Quadropters (gelber Bereich)

Der gelbe Bereich enthält die Berechnung des Earthframe des Quadropters. Am Eingang des Bereichs werden in dem Block „*equations of motion*“ alle Werte des vorangegangenen grünen Bereichs verarbeitet. Dazu gehören die Summe aller Propeller-Kräfte  $U1$ , der Winkel  $\phi$ , der Winkel  $\theta$ , der Winkel  $\psi$  sowie die Masse des Quadropters  $m$  und die Erdbeschleunigung  $g$ . Dieser Block berechnet die Beschleunigung in x-, y- und z-Richtung des Quadropters im Earthframe. Aus diesen Beschleunigungswerten kann man durch das einmalige integrieren, die Geschwindigkeit, und durch nochmaliges integrieren, die Position des Quadropters berechnet werden. Desweiteren besteht die Möglichkeit Störgrößen aufzuschalten um so z.B. Wind zu simulieren.

**Bodyframe des Quadropters (roter Bereich)**

Der rote Bereich enthält die Berechnung des Bodyframes des Quadropters. Zum Eingang dieses Bereichs führen die Winkelbeschleunigungen der Winkel  $\phi$  (U2),  $\theta$  (U3) und  $\psi$  (U4) und die Winkelgeschwindigkeit aller Propeller.

Den roten Bereich könnte man nochmals in 3 Bereiche unterteilen. Im ersten Teilbereich wird die Winkelbeschleunigung vom Winkel  $\phi$  berechnet. Im zweiten Teilbereich wird die Winkelbeschleunigung vom Winkel  $\theta$  und im dritten Teilbereich wird die Winkelbeschleunigung vom Winkel  $\psi$  berechnet.

Betrachtet man den mittleren Zweig (Berechnung der Winkelbeschleunigung von  $\theta$ ) genauer, sieht man, dass hierfür die Variablen  $U3$ ,  $r\phi$ ,  $r\psi$  sowie die Winkelgeschwindigkeit aller Propeller  $anv\_total$  benötigt werden.

Der Vektor  $pos\_vel\_acc$  enthält folgende Variablen:

<b>x</b>	Position x des Earth-Frames
<b>y</b>	Position y des Earth-Frames
<b>z</b>	Position z des Earth-Frames
<b>phi</b>	Winkel phi des Body-Frames
<b>theta</b>	Winkel theta des Body-Frames
<b>psi</b>	Winkel psi des Body-Frames
<b>vx</b>	Lineare Geschwindigkeit in x-Richtung
<b>vy</b>	Lineare Geschwindigkeit in y-Richtung
<b>vz</b>	Lineare Geschwindigkeit in z-Richtung
<b>rphi</b>	Winkelgeschwindigkeit des W. phi
<b>rtheta</b>	Winkelgeschwindigkeit des W. theta
<b>rpsi</b>	Winkelgeschwindigkeit des W. psi
<b>ax</b>	Lineare Beschleunigung in x-Richtung
<b>ay</b>	Lineare Beschleunigung in y-Richtung
<b>az</b>	Lineare Beschleunigung in z-Richtung
<b>aphi</b>	Winkelbeschleunigung des W. phi
<b>atheta</b>	Winkelbeschleunigung des W. theta
<b>apsi</b>	Winkelbeschleunigung des W. psi
<b>U1</b>	Summe aller Propellerkräfte

Tabelle 9: Der Vektor  $pos\_vel\_acc$

### 5.2.5 Sensor-Block

Im Sensor-Block werden alle 5 Sensoren simuliert, die auch auf dem Quadrokofter vorhanden sind. Als Eingang bekommt dieser Block den kompletten *pos-vel-acc*-Vektor, dieser wird aber durch einen sogenannten Selector aufgesplittet. Das heißt es werden nur die Werte rausgegriffen, die auch wirklich benötigt werden. In diesem Fall sind es die Werte  $r\_phi$ ,  $r\_theta$ ,  $r\_psi$ ,  $phi$  und  $theta$ , die auch beim realen Quadrokofter gemessen werden und für die Regelung wichtig sind. Die Werte  $r\_phi$ ,  $r\_theta$  und  $r\_psi$  entsprechen der Winkelgeschwindigkeit und die Werte  $phi$  und  $theta$  der Winkel. Die zwei Sensor-Blöcke „*acc\_ax\_sensor*“ und „*acc\_ay\_sensor*“ messen die lineare Beschleunigung in x- und y-Richtung des Bodyframes. Diese Werte sind im *pos-vel-acc*-Vektor nicht vorhanden und müssen in diesen Blöcken berechnet werden. Dies erfolgt mit dem Winkel und einer trigonometrischen Funktion. Die Winkelgeschwindigkeiten der drei Winkel sind allerdings im *pos-vel-acc*-Vektor vorhanden und müssen deshalb nicht berechnet werden. Um reale Sensoren zu simulieren kann man bei allen Sensor-Blöcken ein Rauschen aufschalten. Alle gemessenen Werte werden wieder durch einen Selector in der Reihenfolge *ax*, *ay*,  $r\_phi$ ,  $r\_theta$ ,  $r\_psi$  zu einem Vektor namens „IMU“ kombiniert.

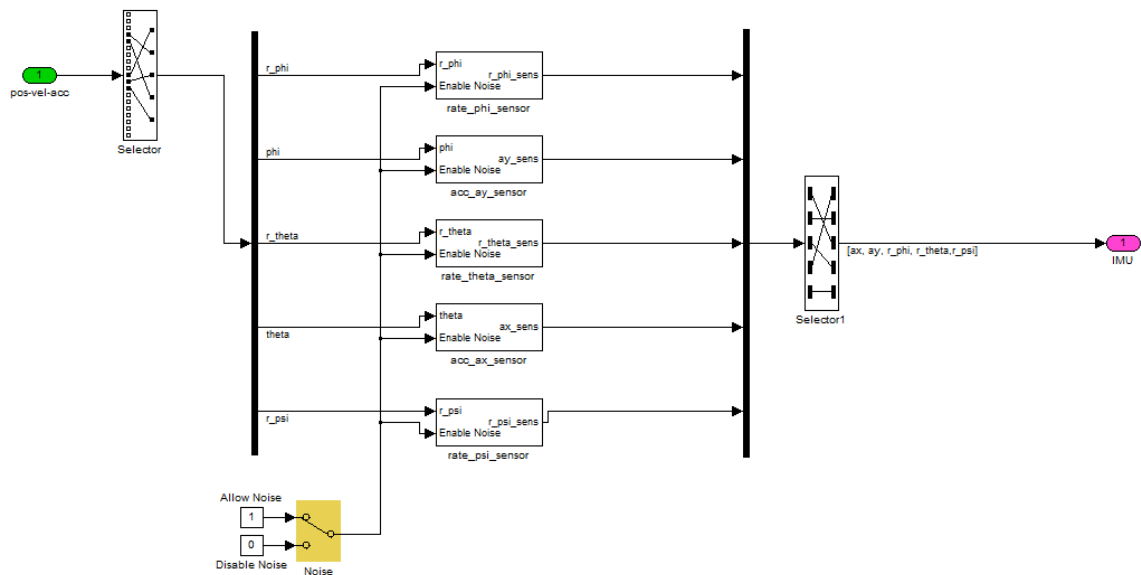


Abbildung 35: Sensor-Block. [1]

## 5.3 Zustandsregler des Quadrokofters

Der Zustandsregler für den Quadrokofter wurde in einer Studienarbeit von Herrn B. Jaißle und Herrn A. Stoltz entwickelt [1]. Dabei wurde der Zustandsregler erst in Matlab-Simulink entworfen und dann in C implementiert.

Bei diesem Zustandsregler handelt es sich um einen MIMO-Regler, d.h. er hat 3 Eingänge und 3 Ausgänge. Die drei Eingänge entsprechen den Sollwertvorga-

ben, die von der Fernsteuerung aus gesendet werden. Die drei Sollwerte sind der Winkel  $\phi$  für das Rollen, der Winkel  $\theta$  für das Nicken und die Winkelgeschwindigkeit  $\dot{\psi}$  für das Gieren.

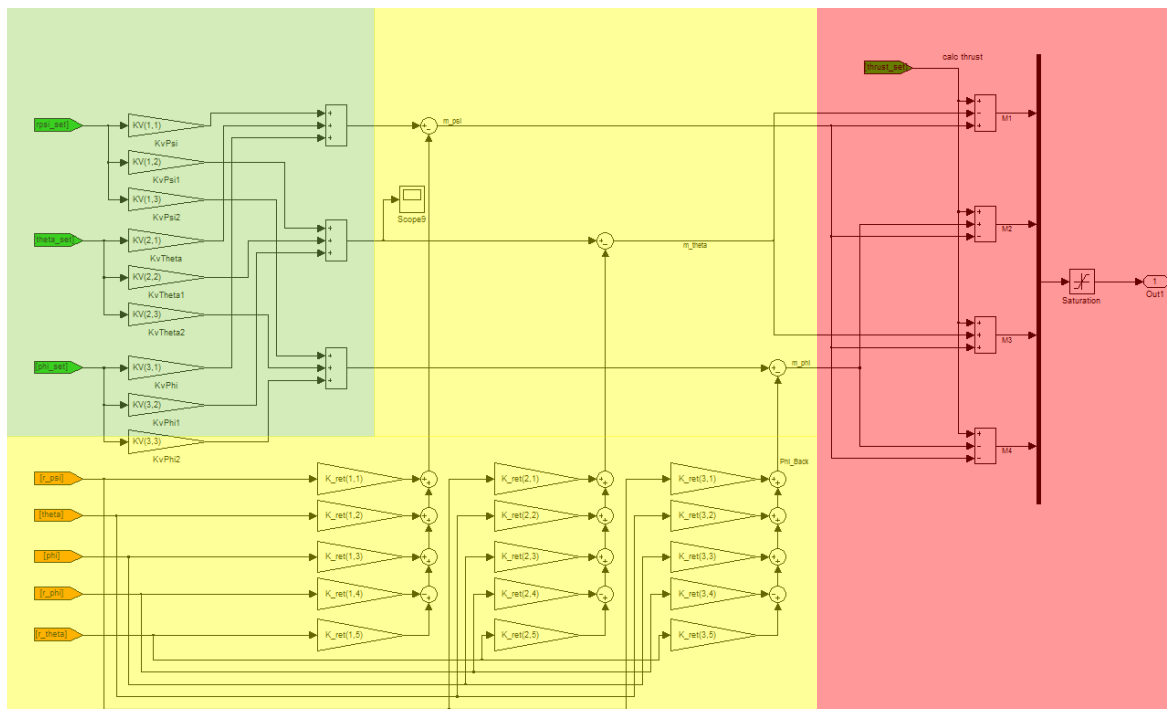


Abbildung 36: Zustandsregler des Quadropters. [1]

Die Berechnung eines MIMO-Systems ist dieselbe, wie die bei einem SISO-System. Allerdings lässt sich so ein MIMO-System nur noch sehr schwer von Hand ausrechnen, deshalb wird für die weitere Berechnung das Tool MATLAB-Simulink verwendet. Dabei kann man das Quadropters-Modell in Simulink erstellen und mit Hilfe von MATLAB die Rückföhrfaktoren  $K_r$  und den Vorfilter  $K_v$  berechnen.

Im grünen Bereich sehen Sie die  $K_v$ -Faktoren des Zustandsreglers. Sie sind pro Eingang dreifach vorhanden, da der Quadropters ein zusammenhängendes System ist und die Auswirkungen z.B. des Sollwerts des Winkels  $\phi$  auch die anderen Regelkreise beeinflusst.

Im gelben Bereich werden die Zustandsrückföhrungen mit den  $K_r$ -Faktoren dargestellt. Diese sind, wie vorhin auch schon, die  $K_v$ -Faktoren mehrfach vorhanden. Sie bekommen die simulierten Sensorwerte vom Quadropters-Prozess, der in Abbildung 36 nicht gezeigt wird.

## 5.4 Optimierung des Zustandsreglers des realen Quadropters

Die MATLAB-Simulation des Quadropters zeigte, dass der Zustandsregler theoretisch funktioniert. Allerdings änderte sich dies, als der Zustandsregler in der Programmiersprache C auf dem Quadropter umgesetzt wurde.

Im Folgenden wird der C-Code des realen Quadropters mit Hilfe des MATLAB-Simulink-Modell erläutert.

Der Zustandsregler bekommt als Eingangswerte die Sollwerte der Fernsteuerung. Im C-Code lauten die Werte *remote\_Pitch\_SS*, *remote\_Roll\_SS* und *remote\_Yaw\_SS* und entsprechen den Werten in Abbildung 37 im Simulink-Modell. Die Teiler sind notwendig um die Empfindlichkeit auf die Steuersignale zu ändern. Je größer die Teiler umso unempfindlicher reagiert der Quadropter.

```
remote_Pitch_SS = ((float) copterState->remotePitch)/2;
remote_Roll_SS = ((float) copterState->remoteRoll)/2;
remote_Yaw_SS = ((float) copterState->remoteYaw)/4;
```

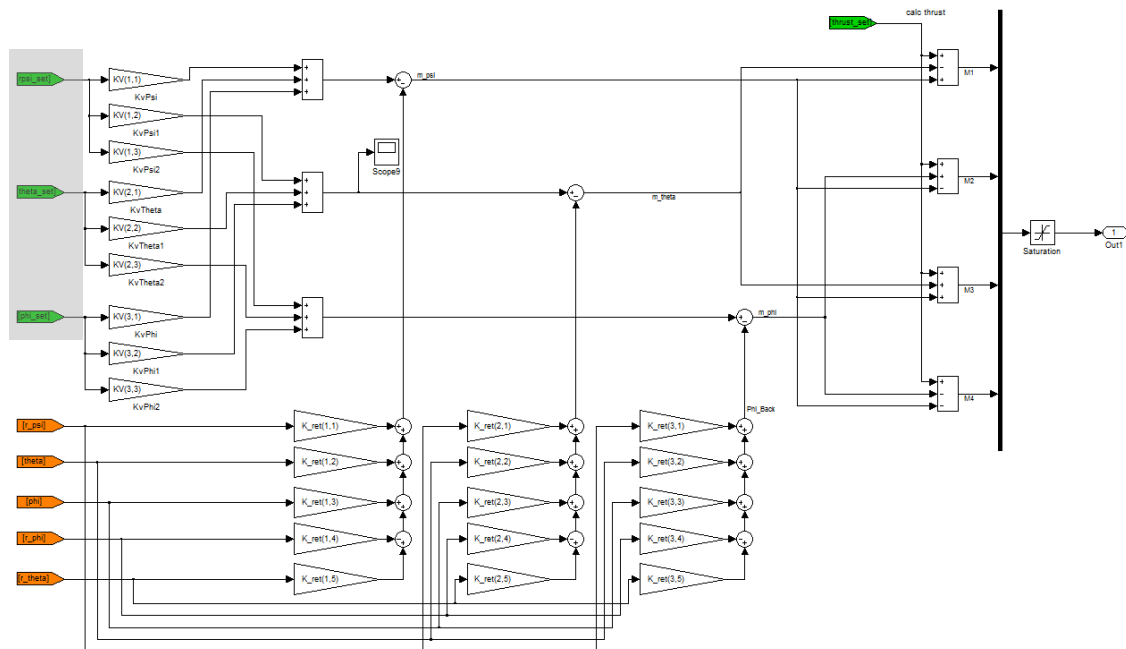


Abbildung 37: Sollwerte der Fernsteuerung



Die Zustandsvariablen des realen Quadropters sind die gemessenen Sensorwerte. Diese lauten im C-Code *theta\_sens*, *phi\_sens*, *rtheta\_sens*, *rphi\_sens* und *rpsi\_sens* und entsprechen den Werten in Abbildung 38 im Simulink-Modell. Die Teiler und Faktoren sind wie auch schon bei den Sollwerten nötig um die Empfindlichkeit des Quadropters einzustellen. Sind die Teiler zu klein eingestellt, wird der Quadropter im Flug sehr unruhig und nervös. Sind die Teiler dafür zu groß, wird der Quadropter zu träge und so unflybar.

Genau das war das Problem des vorhandenen Zustandsreglers, wie man im unteren Code-Block erkennen kann sind die Teiler bei 10 und 8. Beim vorhandenen Zustandsregler waren diese Teiler um den Faktor 50 größer, d.h. die Zustandsrückführung hatte keine Auswirkung auf die Sollwerte. Dieses Fehlverhalten wurde deutlich als man die Quadroptermotoren einschaltete und durch kippen und neigen keine Reaktion des Reglers zu erkennen war. Als die Teiler kleiner gemacht wurden, wurden die Sollwerte entsprechend geregelt. Allerdings funktionierte die Regelung immer noch nicht richtig. Beim Nicken regelte der Zustandsregler wie gewünscht, aber beim Rollen regelte er während der Drehung des Quadropters erst entgegengesetzt und hatte dann eine kurze Denkpause. Nach der Denkpause regelte er wieder richtig. Es gab noch ein weiteres Problem welches bei der Zustandsrückführung betrachtet und erläutert wird.

```

theta_sens = ((float) copterState->Pitch_filt)*40;
phi_sens = ((float) copterState->Roll_filt)*40;
rtheta_sens = ((float) copterState->angVelP)/10;
rphi_sens = ((float) copterState->angVelR)/10;
rpsi_sens = ((float) copterState->angVelY)/8;

```

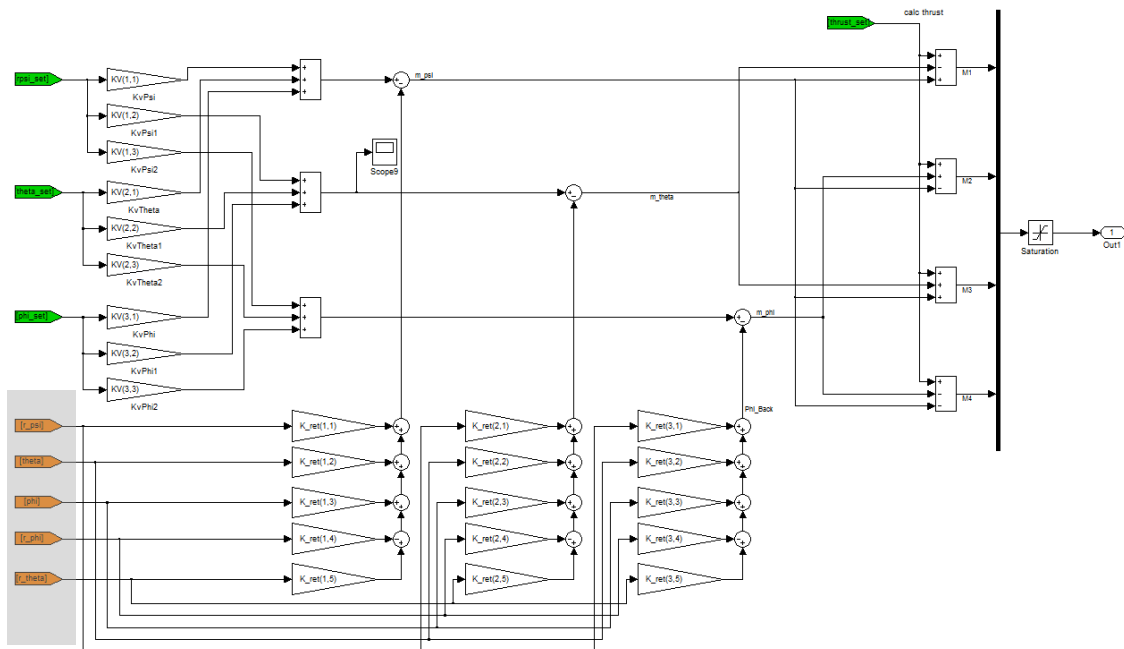


Abbildung 38: Gemessene Zustandsvariablen des Quadropters

Der nächste C-Code-Block entspricht den Sollwerten des Eingangs und der berechneten Vorfiltern den sogenannten  $K_v$ -Faktoren. Diese sind wie auch im Simulink-Modell miteinander verbunden. Das heißt, wird der Sollwert von z.B.  $rpsi\_SET$  verändert, wirkt sich das auch auf die anderen Werte  $theta\_SET$  und  $phi\_SET$  aus.

```

rpsi_SET = remote_Yaw_SS * rpsi_rpsi + remote_Pitch_SS * theta_rpsi +
remote_Roll_SS * phi_rpsi;

theta_SET = remote_Yaw_SS * rpsi_theta + remote_Pitch_SS * theta_theta +
remote_Roll_SS * phi_theta;

phi_SET = remote_Yaw_SS * rpsi_phi + remote_Pitch_SS * theta_phi +
remote_Roll_SS * phi_phi;

```

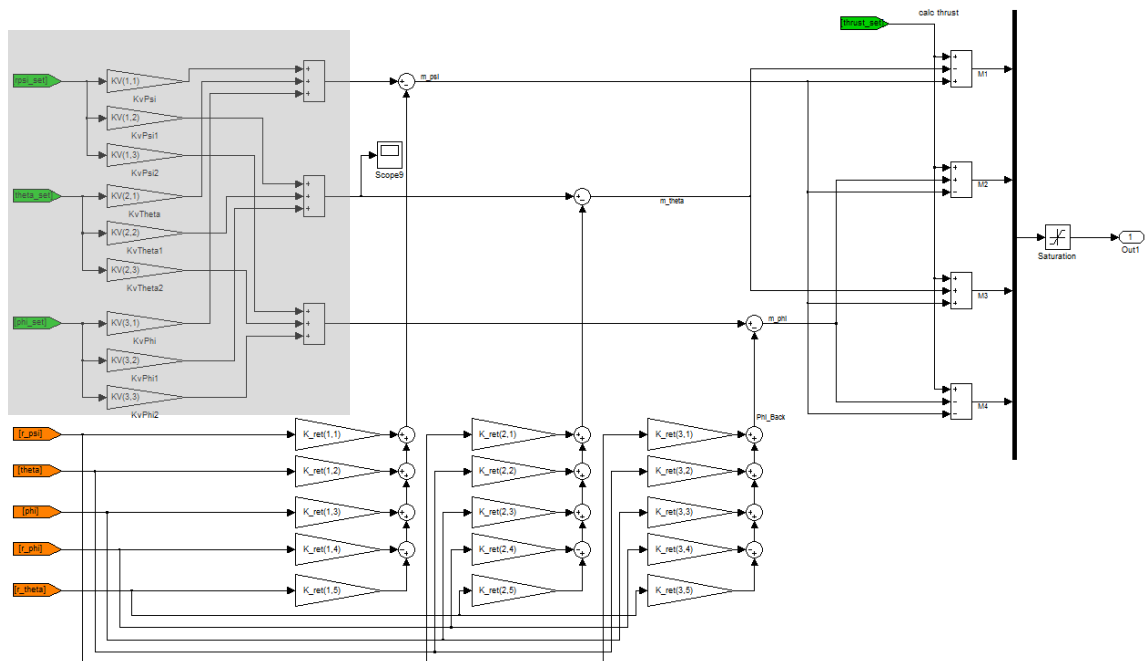


Abbildung 39: Sollwerte mit berechneten  $K_v$ -Faktoren

Der nächste C-Code-Block entspricht der Reglerrückführung, dabei werden die gemessenen Sensorwerte mit den berechneten Rückföhrfaktoren  $K_r$  jeweils multipliziert bzw. miteinander addiert und vom Sollwert abgezogen. Und hier sieht man einen kleinen aber entscheidenden Unterschied zwischen dem C-Code und dem Simulink-Modell. Im Simulink-Modell ist der Zweig von  $r\_phi$  jeweils mit einem negativen Vorzeichen behaftet. Das liegt daran, dass der Sensor der diesen Wert misst auf dem realen Quadrokofter falschherum eingebaut ist. Im C-Code wird diese Gegebenheit allerdings schon bei der Filterung des Sensorwertes, also noch vor der Regelung ausgeglichen. Würde man es wie in Simulink nochmals in der Regelung invertieren, bekommt man falsche Werte.

Genau dieser Fehler wurde beim vorhandenen Zustandsregler gemacht. Während der Drehung liefert  $r\_phi$  einen im Vorzeichen vertauschten Wert und invertiert somit die Regelung. Erst als der Quadrokofter die Schräglage eingenommen hat und es keine Winkelgeschwindigkeitsänderung mehr gab, regelte der Zustandsregler wieder richtig, da  $r\_phi$  in der Schräglage nicht geändert wird.

```
m_r_Yaw = rpsi_SET - (rpsi_sens * rpsi_Rrpsi + theta_sens * rpsi_Rtheta +  
                    phi_sens * rpsi_Rphi + rphi_sens * rpsi_Rrphi + rtheta_sens *  
                    rpsi_Rrtheta);  
  
m_r_Roll = phi_SET - ( rpsi_sens * phi_Rrpsi + theta_sens * phi_Rtheta +  
                    phi_sens * phi_Rphi + rphi_sens * phi_Rrphi + rtheta_sens *  
                    phi_Rrtheta);  
  
m_r_Pitch = theta_SET - ( rpsi_sens * theta_Rrpsi + theta_sens *  
                        theta_Rtheta + phi_sens * theta_Rphi + rphi_sens *  
                        theta_Rrphi + rtheta_sens * theta_Rrtheta);
```

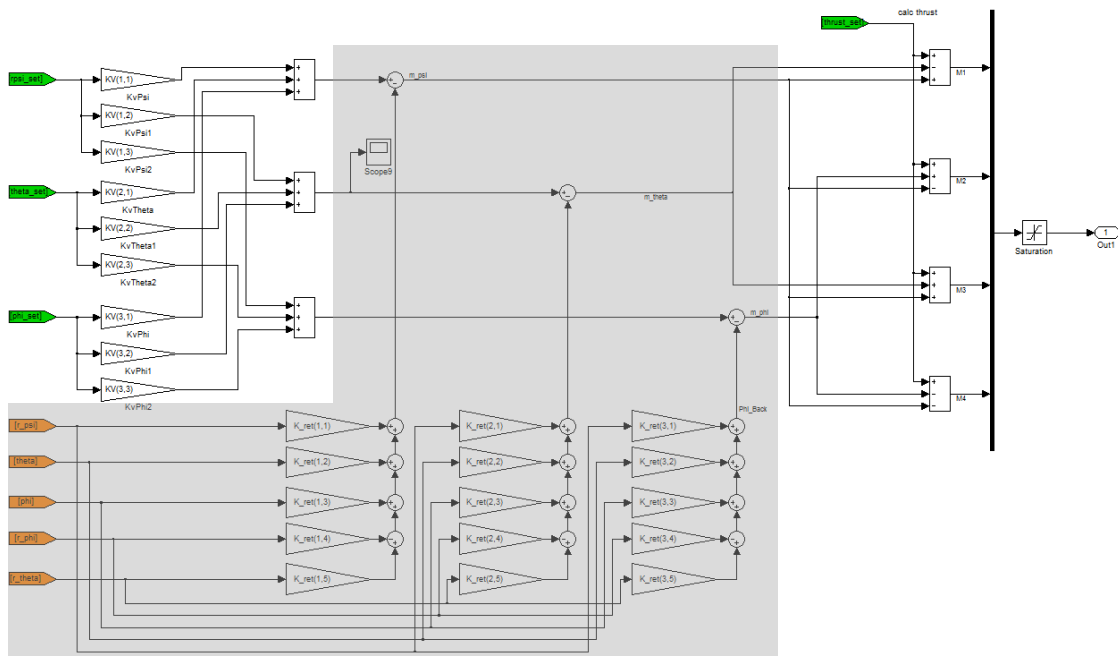


Abbildung 40: Zustandsrückführung mit den berechneten

Der letzte C-Code-Block entspricht der geregelten Sollwertberechnung für die Motoren. Diese Sollwerte steuern dann beim realen Quadrokopter die Motorregler an. Dabei entspricht M1 dem vorderen Motor, M2 dem linken Motor, M3 dem hinteren Motor und M4 dem rechten Motor. Man kann außerdem sehr gut die Ansteuerung der Motoren erkennen, *thrust* entspricht im C-Code der Rohdaten der Fernsteuerung *remoteForceRaw*, *m\_theta* entspricht im C-Code dem berechneten Sollwert *m\_r\_Pitch*, *m\_phi* entspricht im C-Code dem berechneten Sollwert *m\_r\_Roll* und *m\_psi* entspricht im C-Code dem berechneten Sollwert *m\_r\_Yaw*. Dabei werden die berechneten Stellgrößendifferenzen mit dem Gesamtschub überlagert und ergeben die benötigten Stellgrößen.

```

M1 = (float)(copterState->remoteForceRaw) - m_r_Pitch + m_r_Yaw;
M2 = (float)(copterState->remoteForceRaw) + m_r_Roll - m_r_Yaw;
M3 = (float)(copterState->remoteForceRaw) + m_r_Pitch + m_r_Yaw;
M4 = (float)(copterState->remoteForceRaw) - m_r_Roll - m_r_Yaw;

```

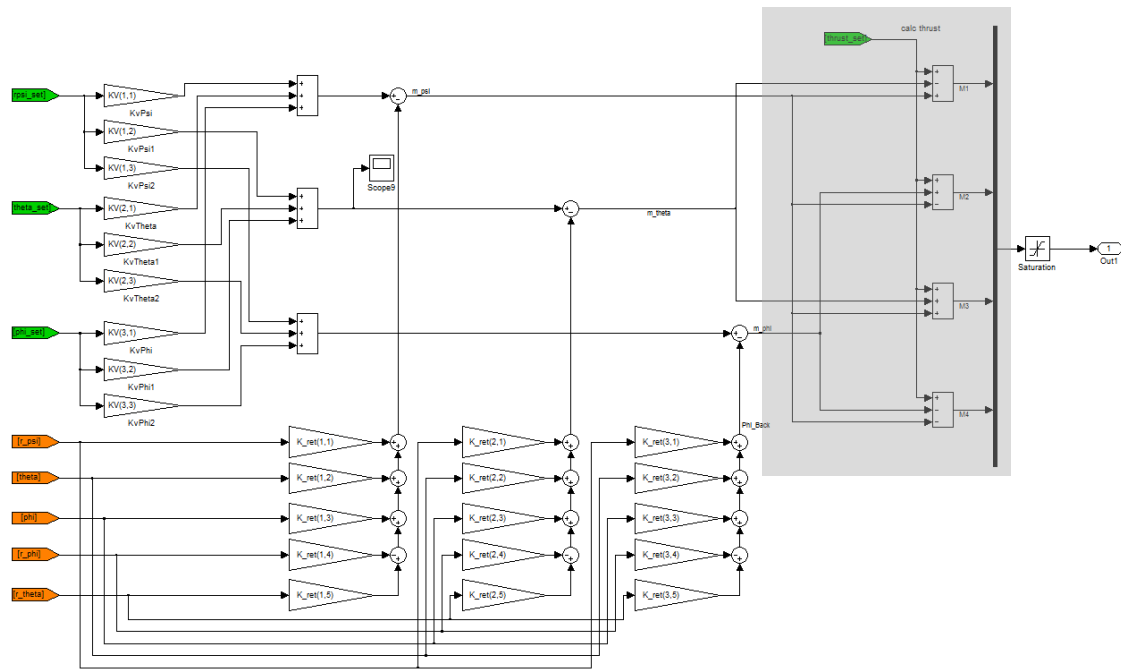


Abbildung 41: Sollwertberechnung der Motoren

## 6 Zusammenfassung und Ausblick

Der Zustandsregler wurde soweit optimiert, dass der Quadrokofter sicher und stabil fliegt. Allerdings wurde noch nicht getestet wie störanfällig der Quadrokofter mit Störgrößen wie die Kamera oder bei schlechten Windbedingungen ist. Durch die Speicherung der kalibrierten Sensorwerte kann der Quadrokofter nun auch in Umgebungen eingesetzt werden in der keine ebene Fläche zum Kalibrieren bereitsteht. Desweiteren ist der Quadrokofter in der Lage, kurzzeitig freihändig geflogen zu werden.

Als nächster Schritt muss noch untersucht werden, wie sich die Regelung verhält, wenn die Rechnung komplett auf Integer umgestellt wird. Denn die Berechnung der Flight-Control wird zurzeit mit Floating-Point-Operationen realisiert, die sehr viel Rechenzeit kosten und man nicht genau sagen kann wie stark der Mikrocontroller ausgelastet ist.

In naher Zukunft kann man die Möglichkeit der Abspeicherung von mehreren Setups in Betracht ziehen. Dadurch ist es möglich den Quadrokofter ohne großen Aufwand aggressiver oder langsamer einzustellen und ihn so an bestimmte Begebenheiten anzupassen.

## 7 Literaturverzeichnis

- [1] Alexander Stoltz, Benjamin Jaißle. Development of a state space controller for a quadrocopter. HS-Esslingen, 2011.
- [2] ELMICRO. Elektronikladen by ELMICRO. 21. 11 2011. 21. 11 2011 <elmicro.com/images/zf-xbee-pro.jpg>.
- [3] Kull, Prof. Dr.-Ing. Hermann. Systemtechnik Manuskript - Kapitel 7 Zustandsregler. HS-Esslingen, 2011.
- [4] Stübler, Manuel. Valiquad - The analyzingsoftware for the Quadrocopter. HS-Esslingen, 2010/2011.