Zürcher Hochschule
für Angewandte Wissenschaften

zh
aw

Flugregelung einer Kleindrohne für Forschungszwecke

Betreuer:	Prof. Dr. sc. techn. ETH Walter Siegl
Verfasser:	Emilio Schmidhauser
	René Chaney
Studiengang:	Maschinentechnik-Informatik
	Bachelorarbeit 2010
Datum:	13. August 2010

I. Inhaltsverzeichnis

II. Abstract

Diese Arbeit handelt von der Implementierung und Optimierung eines Open Source Autopiloten (Paparazzi) für eine Drohne zu Forschungszwecken. Es handelt sich bei der Drohne um ein Projekt welches an der ZHAW mit dem Namen UMARS (Unmanned Modular Airborne Research System) realisiert wird.

Das Ziel dieser Arbeit war es, anstelle der Infrarot-Sensoren, die für die Bestimmung der Fluglage verantwortlich sind, eine Inertial Measurement Unit (IMU) zu implementieren. Für Infrarot-Sensoren stellt sich der Himmel als kalt und die Erdoberfläche als warm dar. Aus der Temperaturdifferenz zweier Infrarotsensoren lässt sich die Lage berechnen. Probleme dieser Messemethode sind ungünstige Temperaturkonstellationen von Wolken (z.B. am Morgen), Waldgebiete oder Flüge längs von Felswänden. Mit einer IMU ist die Lageerfassung unabhängig von derartigen Einflüssen. Ausserdem wurde der Autopilot mit einer Geschwindigkeitsregelung ausgerüstet, die die Geschwindigkeit bezüglich der Luft konstant halten kann. Hauptgrund dafür sind Messkampagnen, die mit der Drohne geplant sind, welche eine konstante Luftgeschwindigkeit von ± 1 m/s erfordern.

Ungünstige Eigenschaften des Testflugzeuges und Ungenauigkeiten der Geschwindigkeitsmessung bei tiefen Fluggeschwindigkeiten machten das Einstellen der Regelparameter schwierig. Trotzdem konnten mit dem Testflugzeug sehr gute Ergebnisse erzielt werden, die die Toleranzen nur knapp nicht erfüllen.

This work deals with the implementation and optimization of an open source autopilot (paparazzi) for a drone for research purposes. The drone is a project which is implemented at the ZHAW named UMARS (Unmanned Modular Airborne Research System).
The goal of this project was to replace the infrared sensors, which are responsible for determining the attitude, with an Inertial Measurement Unit (IMU). For infrared sensors, the sky is presented as cold and the earth's surface as warm. From the temperature difference between two infrared sensors the attitude can be calculated. Problems of this method are fair unfavourable temperature constellations of clouds (e.g. in the morning), forest areas or flights along a rock face. An IMU should be selected and integrated into the existing system. In addition, the autopilot was fitted with a speed control, which can keep the speed constant compared to the air. The main reason for this, are the planned measurement campaigns with the drone, which require a constant air velocity of ± 1 m/s.
Unfavourable characteristics of the test aircraft, and inaccuracies in the velocity measurement at low airspeeds made the adjustment of control parameters difficult. Despite many problems with the test aircraft the tolerances could be almost reached.

III. Selbständigkeitserklärung

Zürcher Hochschule für Angewandte Wissenschaften

School of Engineering

Erklärung betreffend das selbständige Verfassen einer Bachelorarbeit an der School of Engineering

Mit der Abgabe dieser Bachelorarbeit versichert der/die Studierende, dass er/sie die Arbeit selbständig und ohne fremde Hilfe verfasst hat. (Bei Gruppenarbeiten gelten die Leistungen der übrigen Gruppenmitglieder nicht als fremde Hilfe.)

Der/die unterzeichnende Studierende erklärt, dass alle zitierten Quellen (auch Internetseiten) im Text oder Anhang korrekt nachgewiesen sind, d.h. dass die Bachelorarbeit keine Plagiate enthält, also keine Teile, die teilweise oder vollständig aus einem fremden Text oder einer fremden Arbeit unter Vorgabe der eigenen Urheberschaft bzw. ohne Quellenangabe übernommen worden sind.

Bei Verfehlungen aller Art treten die Paragraphen 46 (Unredlichkeit und Verfahren bei Unredlichkeit) der ZHW Prüfungsordnung sowie die Bestimmungen der Disziplinarmassnahmen der Hochschulordnung in Kraft.

Ort, Datum:	Unterschriften:
.....	.....
	.....
	.....

Das Original dieses Formulars ist bei der ZHAW-Version aller abgegebenen Bachelorarbeiten zu Beginn der Dokumentation nach dem Abstract bzw. dem Management Summary mit Original-Unterschriften und -Datum (keine Kopie) einzufügen.

Zürcher Fachhochschule

1 Einleitung

UMARS [11] ist ein Projekt des Institutes für Mechanische Systeme (IMES) und dem Zentrum für Aviatik (ZAV) an der ZHAW in Winterthur. Es handelt sich um die Entwicklung eines autonom fliegenden Messflugzeuges für die meteorologische Forschung. Die Drohne soll dabei möglichst flexibel und einfach zu bedienen sein, um aufwendige und teure Messungen in Kleinflugzeugen so weit wie möglich vermeiden zu können. Die zum Zeitpunkt der Erstellung dieses Berichts aktuelle Demonstratordrohne hat eine Spannweite von 5 m und ein maximales Abfluggewicht von 25 kg, was einer maximalen Nutzlast von 10 kg entspricht. Sie ist aerodynamisch für eine maximale Flughöhe von 5000 m.ü.M ausgelegt und hat einen Cruisespeed von 70 km/h. Weiter ist die Konstruktion einer Nachfolgedrohne geplant und wird in den nächsten Monaten in Angriff genommen. Für dieses Trägersystem soll nun ein Autopilotensystem implementiert werden, welches ermöglicht, eine zuvor programmierte Route autonom abfliegen zu können. Während der Projektarbeit im Abschlusssemester wurde ein Autopilotensystem ausgewählt, Komponenten bestellt und das System analysiert. Der Bericht dieser Projektarbeit [8] befindet sich im Anhang und beinhaltet viele Grundinformationen zum Gebrauch von Paparazzi.

Das Ziel dieser Arbeit war es, anstelle der Infrarot-Sensoren, die für die Bestimmung der Fluglage verantwortlich sind, eine Inertial Measurement Unit (IMU) zu implementieren. Für Infrarot-Sensoren stellt sich der Himmel als kalt und die Erdoberfläche als warm dar. Aus der Temperaturdifferenz zweier Infrarot-Sensoren lässt sich die Lage berechnen. Probleme dieser Messemethode sind ungünstige Temperaturkonstellationen von Wolken (z.B. am Morgen), Waldgebiete oder Flüge längs von Felswänden. Für die Drohne soll darum eine IMU ausgewählt und in das bestehende System integriert werden. Ausserdem soll die Drohne über eine Geschwindigkeitsregelung verfügen, die die Geschwindigkeit bezüglich der Luft konstant halten kann. Hauptgrund dafür sind Messkampagnen, die mit der Drohne geplant sind, welche eine konstante Luftgeschwindigkeit ($\pm$ 1 m/s) erfordern. Zusätzlich ist die Geschwindigkeit bezüglich der Luft für ein Flugzeug essentiell und muss sich immer in gewissen Grenzen bewegen. Als Testflugzeug für den Autopiloten wurde ein Flugzeug namens MAJA von der Firma borjet beschafft, mit dem alle in diesem Bericht gemachten Flugversuche durchgeführt wurden.

Zu Beginn wurden erste Flüge durchgeführt und Erfahrungen mit dem Einstellen der Regelparameter gesammelt. Als IMU wurde ein sehr preisgünstiger Sensor gewählt (ArduImu), der in einem ähnlichen Open Source Autopiloten Verwendung findet. Für die Luftgeschwindigkeit- und Höhenregelung wurden verschiedene Regler erstellt, die während des Fluges umgeschaltet werden können.

Als Messinstrumente für die Luftgeschwindigkeit und Höhe wurden ein Differenzdrucksensor (Pitot-Rohr) und ein Barometer von einer anderen Bachelorarbeit [15], die sich im Kontext der Drohne mit einem Messbaum für Anstellwinkel, Barometer und Differenzdruck beschäftigt hatten, übernommen.

Bei wenig bis mässigen Winden funktioniert die Geschwindigkeitsregelung mit allen Reglern sehr gut, bei stärkeren Winden konnten aber die Vorgaben noch nicht eingehalten werden. Wobei es sich hier aber vor allem um das Finden der richtigen Regelparameter handeln dürfte und um das Flugverhalten des Testflugzeuges. Das Flugverhalten des Testflugzeuges (MAJA) zeigte sehr ungünstige Eigenschaften bei Änderungen der Motorenleistung. So sinkt das Flugzeug sehr stark ab, wenn von niederen Drehzahlen sprunghaft auf grosse Drehzahlen beschleunigt wird. Dieses Problem könnte allenfalls über eine Lookuptable, die die Regelparameter lastfallabhängig setzt, gelöst werden. Für die Feinjustierung der Parameter bezüglich Höhe und Airspeed des Testflugzeuges wurde nicht viel Zeit investiert, da die Flugeigenschaften der Drohne nicht mit deren des Testflugzeuges übereinstimmen.

Trotzdem konnte gezeigt werden, dass mit den erstellten Regelungen auch das Testflugzeug genügend gut geregelt werden kann. Der IMU funktioniert einwandfrei bis auf einen minimalen Drift, der bei dauerhaftem Kreisen um einen Punkt auftritt. Dieser Drift rührt grösstenteils vom Wind her, der die Korrektur des Drifts um die Z-Achse beeinflusst. Bei Seitenwind hat das Flugzeug nicht den gleichen Winkel wie der GPS-Kurs vorgibt. Die Driftkorrektur der Z-.Achse wird aber mit dem GPS-Kurs gerechnet und deswegen entsteht ein Fehler, der sich über längere Zeit aufsummieren kann und Probleme verursachen kann. Dieses Problem kann mit Hilfe eines Magento-meters (Elektronischer Kompass) eliminiert werden. Dies konnte aus Zeitgründen nicht mehr realisiert werden. Aus technischer Sicht sollte die Implementierung dieses Sensors ohne grosse Schwierigkeiten möglich sein.

1.1 Was ist der Paparazzi-Autopilot?

Paparazzi beinhaltet ein komplettes System für den autonomen Flug. Darin enthalten sind die in der Luft benötigten Komponenten (exklusive RC Zubehör), sowie die Bodenstation, die mittels einer seriellen Verbindung mit der Drohe verbunden ist. Die Bodenstation wird verwendet um die Flugroute zu planen, aber auch für die Echtzeit-Verfolgung der Drohne in der Luft. Über ein Radio-Modem werden die aktuellen Daten an die Bodenstation gesendet. Auch Videosignale und andere Parameter können übermittelt werden. Paparazzi ist Open Source und läuft unter Linux.

Autopilot Control Board
Battery
Data Link Radio-Modem & Antenna
GPS Receiver
IR Sensor Board
Motor & Controller
RC Receiver & Antenna
Servos
Payload = Camera & Video Transmitter

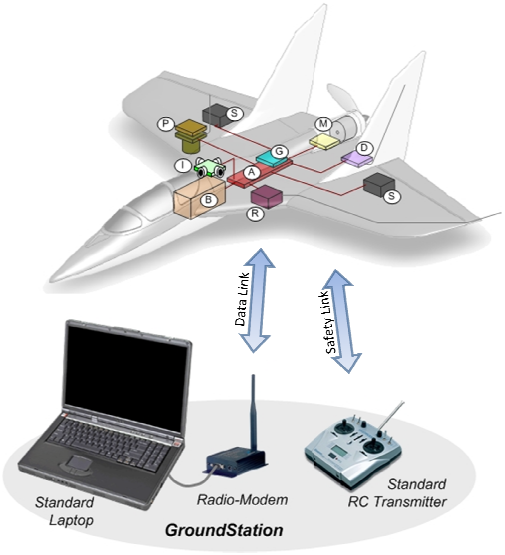

Abb. 1 Übersicht über das Paparazzi Systems

Man benötigt ein flugfähiges Modellflugzeug mit allen Komponenten wie Servos, Empfänger, Regler und Motor. Das Paparazzi-System wird übergeordnet dazwischen geschaltet. Die Fernsteuerung wird nur noch als Safety Link verwendet. Bei Problemen während des autonomen Fluges kann über einen Schalter an der Fernbedienung die Kontrolle des Flugzeuges wieder übernommen werden. Vor dem Start werden alle flugrelevanten Informationen sowie die Flugroute in den Flashspeicher des Autopiloten geladen. Das Flugzeug ist völlig autonom und kann, ohne Eingriff von aussen, den programmierten Weg abfliegen. Die Drohne regelt sich bei z.B. Seitenwind selbständig aus und steuert das Flugzeug wieder auf die angepeilten Koordinaten. Der Data Link zwischen Ground-Station und dem Flugzeug wird eigentlich nur für den Empfang von Telemetriedaten und für Änderungen an der Flugroute während des Fluges gebraucht.

Emilio Schmidhauser
René Chaney

1.2 Aufgabenstellung

MET
W. Siegl

System- & Automatisierungstechnik

Bachelorarbeit: Flugregelung für eine Drohne

Studenten: **Emilio Schmidhauser MI07**
René Chaney MI07

Beginn: **21. Juni 2010**
Abgabe: **13. August 2010**

Ausgangslage: An der ZHAW wurde im Laufe des Herbstsemesters 2009 eine Drohne konzipiert und gebaut, welche seinen ersten Flurversuch bereits erfolgreich absolviert hat. Bei diesem Flugtest wurde die Drohne von der Bodenstation her nur gesteuert.

Ziel der Arbeit: In dieser Bachelorarbeit soll nun die Implementierung einer Flugregelung realisiert werden. Als Sicherheitsgründen werden alle Tests mit einem kommerziellen Modellflugzeug Typ Maya realisiert. Die gewonnene Erfahrung wird dann in einer späteren Phase des Projektes auf die Drohne übertragen.

Aufgaben:

1. Inbetriebnahme und Flugtests mit dem Modellflugzeug Maya und die Paparazzi Hard- und Software.

2. Marktrecherche und Verfügbarkeit eines *Inertial Measuring Unit* IMU, welche sich in die vorhandene Paparazzi-Hardware integrieren lässt.

3. Konzept und Implementierung einer Geschwindigkeitsregelung, welche ein Pitot-Rohr als Geschwindigkeitssensor verwenden soll.

4. Boden- und anschließend Flugtest des gesamten Systems.

5. Dokumentation: Da praktisch alle Komponenten für die Realisierung einer Flugregelung kommerziell vorhanden sind, wird in dieser Arbeit viel Wert an der Dokumentation gelegt. Insbesondere sollen die Hardware-Integration aller Komponenten sowie die entsprechenden Software-Anpassungen gut beschrieben sein.

Kontakt: **Prof. Dr. W. Siegl**
Tel: 058 934 7534
siew@zhaw.ch

11_Aufgabenstellung 1 Siew-09.08.2010

1.3 Glossar

Begriff Übersetzung / Ausgeschrieben	Erklärung
AHRS Attitude Heading Reference System	AHRS ist im Grunde das gleiche wie eine IMU.
Arduino	Die Arduino-Plattform ist eine aus Soft- und Hardware bestehende Physical-Computing-Plattform. Beide Komponenten sind im Sinne von Open Source quelloffen. Die Hardware besteht aus einem einfachen I/O-Board mit einem Mikrokontroller, analogen und digitalen Ein- und Ausgängen.
Auto1 Mode	Lage Regelung ist eingeschalten und die Sollgrössen für den Roll- und Pitch-Regelkreis können an der Fernsteuerung eingestellt werden. Der Motor muss manuell gesteuert werden.
Auto2 Mode	Komplette Regelung der Flugbahn durch den Autopiloten.
ENAC École nationale de l'aviation civile	Die ENAC (Ecole Nationale de l'Aviation Civile) ist eine im Jahr 1948 in Orly bei Paris gegründete staatliche französische Hochschule für zivile Luftfahrt. Das Projekt Paparazzi wird von dieser Hochschule geleitet.
Flashen des Mikrokontrollers	Speichern des Programmcodes im Flash-Speicher des Mikrokontrollers
FSO Full Scale Output	Mit % FSO wird bei Sensoren der möglich Fehler/Auflösung angegeben.
GCS Ground Control Station	Prozess welcher das GUI auf der Bodenstation während dem Flug darstellt.
I²C Inter-Integrated Circuit	I²C (auch TWI) ist ein von Philips Semiconductors (heute NXP Semiconductors) entwickelter serieller Datenbus. Er wird hauptsächlich geräteintern für die Kommunikation zwischen verschiedenen Schaltungsteilen mit geringer Übertragungsgeschwindigkeit benutzt, z.B. zwischen einem Kontroller und Peripherie-ICs.
IAS Indicated Air Speed	IAS ist die Geschwindigkeit bei einem Flugzeug die in einem Pitot-Rohr gemessen wird. Sie entspricht auf Meereshöhe dem TAS.
IMU Inertial Measurement Unit	Inertialsensoren dienen der Messung translatorischer und rotatorischer Beschleunigungskräfte. Durch Kombination mehrerer Inertialsensoren in einer inertialen Messeinheit (engl.: Inertial Measurement Unit (IMU)) können die Beschleunigungen der sechs Freiheiten gemessen werden.
ISP In-System-Programmer	Die In-System-Programmierung (ISP) ermöglicht das Programmieren einer Logischen Schaltung direkt im Einsatzsystem. Dazu wird meist eine einfache serielle Verbindung genutzt.

Begriff Übersetzung / Ausgeschrieben	Erklärung
Manual Mode	Manuelle Steuerung des Flugzeugs per Fernsteuerung.
MEMS Microelectromechanical systems	Mit Hilfe dieser Mikrosystemtechnik werden Computer mit kleinen mechanischen Geräten wie Sensoren, Ventilen, Getrieben, Spiegeln und Reglern verbunden, die in Halbleiter-Chips eingebettet sind. MEMS werden beispielsweise in Airbags von Kraftfahrzeugen als Beschleunigungsmesser eingesetzt.
Pitch Nick	Roll-Nick-Gier-Winkel, englisch Roll-Pitch-Yaw-Winkel, sind eine Möglichkeit zur Beschreibung der Orientierung eines Fahrzeugs im dreidimensionalen Raum, die zunächst nur bei Luftfahrzeugen gebräuchlich war, inzwischen aber auch zur Lagebeschreibung von Land-, Wasser- und Raumfahrzeugen Verwendung findet.
Roll Roll	Roll-Nick-Gier-Winkel, englisch Roll-Pitch-Yaw-Winkel, sind eine Möglichkeit zur Beschreibung der Orientierung eines Fahrzeugs im dreidimensionalen Raum, die zunächst nur bei Luftfahrzeugen gebräuchlich war, inzwischen aber auch zur Lagebeschreibung von Land-, Wasser- und Raumfahrzeugen Verwendung findet.
RS232 Serielle Schnittstelle	Der Begriff EIA-232, ursprünglich RS-232, bezeichnet einen Standard für eine serielle Schnittstelle, die in den frühen 1960ern von einem US-amerikanischen Standardisierungskomitee (heute EIA – Electronic Industries Alliance) eingeführt wurde.
SCL serial clock line	SCL ist der Taktgeber für die SDA des I²C Protokolls.
SDA serial data line	SDA ist die Datenlinie des I²C Protokolls.
SPI Serial Peripheral Interface	Das Serial Peripheral Interface (kurz SPI) ist ein von Motorola entwickeltes Bus-System mit einem sehr lockeren Standard für einen synchronen seriellen Datenbus, mit dem digitale Schaltungen nach dem Master-Slave-Prinzip miteinander verbunden werden können.
TAS True Airspeed	Wirkliche Geschwindigkeit eines Flugzeuges gegenüber der Luft.
UMARS Unmanned Modular Airborne Research System	UMARS ist ein unbemannt und autonom fliegendes Messflugzeug für die meteorologische Forschung. Seine einfache Bedienung und sein breites Einsatzspektrum sollen die Arbeit der Umweltwissenschaft erleichtern und dabei kostengünstiger sein als die aktuell genutzten manntragenden Flugzeuge. [11]

Begriff Übersetzung / Ausgeschrieben	Erklärung
Yaw Gier	Roll-Nick-Gier-Winkel, englisch Roll-Pitch-Yaw-Winkel, sind eine Möglichkeit zur Beschreibung der Orientierung eines Fahrzeugs im dreidimensionalen Raum, die zunächst nur bei Luftfahrzeugen gebräuchlich war, inzwischen aber auch zur Lagebeschreibung von Land-, Wasser- und Raumfahrzeugen Verwendung findet.

2 Inbetriebnahme des Testflugzeuges mit dem Autopiloten

Flugversuche mit dem Flugzeug konnten während der Projektarbeit vom sechsten Semester nicht mehr durchgeführt werden. Deshalb war eine der ersten Aufgaben die Justierung der Regelparameter während Flugversuchen. Für ein erstes Kennenlernen des Autopiloten wurden keine zusätzlichen Elemente und Messinstrumente verbaut. Es wurde mit der Originalsoftware, den Infrarot-Sensoren und keinerlei zusätzlicher Hardware geflogen.

Dank Oliver Ensslin ein wissenschaftlicher Assistent an der ZHAW und Projektleiter von UMARS, der Mitglied in einem Modellflugclub in Lommis (TG) ist, konnten wir jederzeit auf einen Flugplatz gehen, der die Möglichkeit bietet Akkus zu laden und bei unerwarteten Schauern Schutz vor dem Regen im Clubhaus zu finden. Das Laden der Akkumulatoren zeigte sich als zeitraubendes Problem, welches das Testen von neuen Funktionen stark verzögerte.

2.1 Verkabelung

Vor den Flugversuchen wurde die Verkabelung nochmals überarbeitet um möglichen Verbindungsproblemen aus dem Weg zu gehen. Im weiteren Verlauf der Bachelorarbeit wurde aber die Verkabelung auf nur einen Akku im Flugzeug beschränkt.

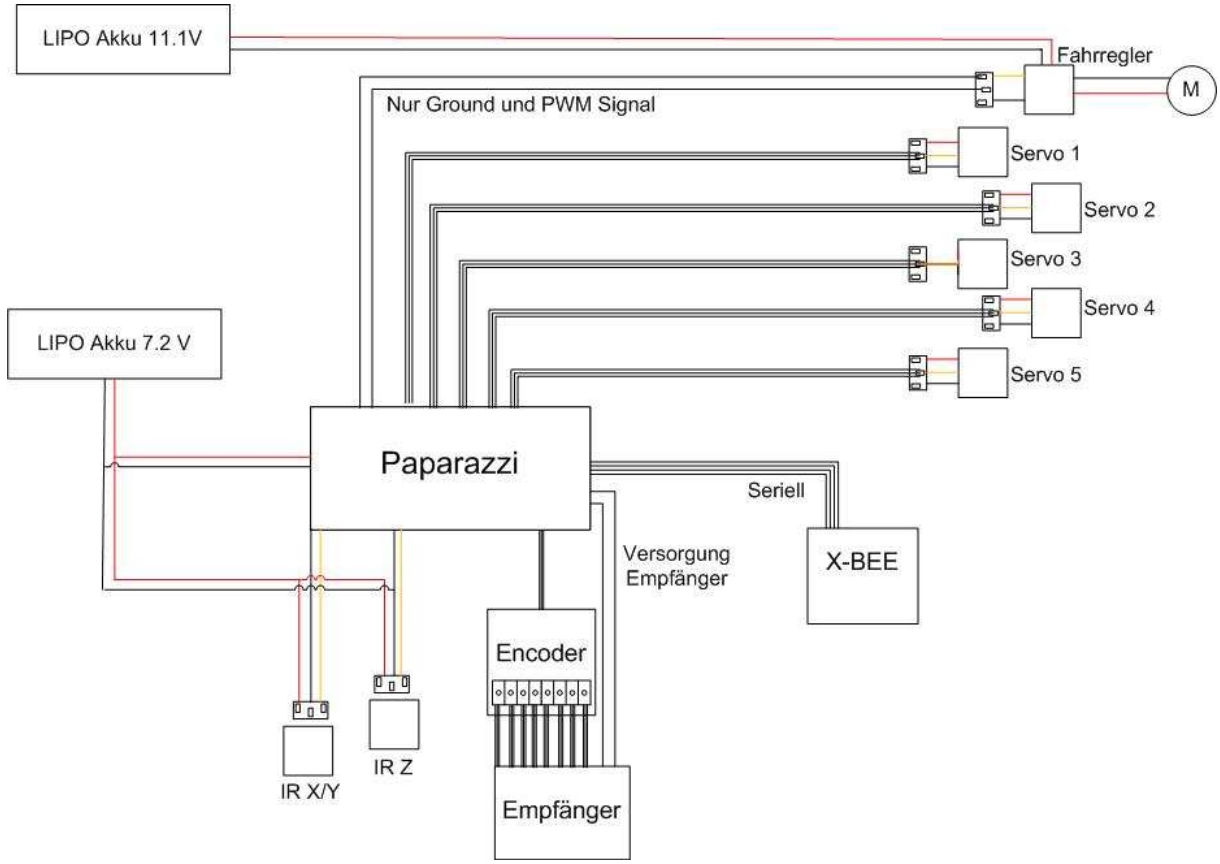

Abb. 2 Signal- und Speisungs-Schema des Autopiloten mit IR-Sensoren und zwei Akkumulatoren

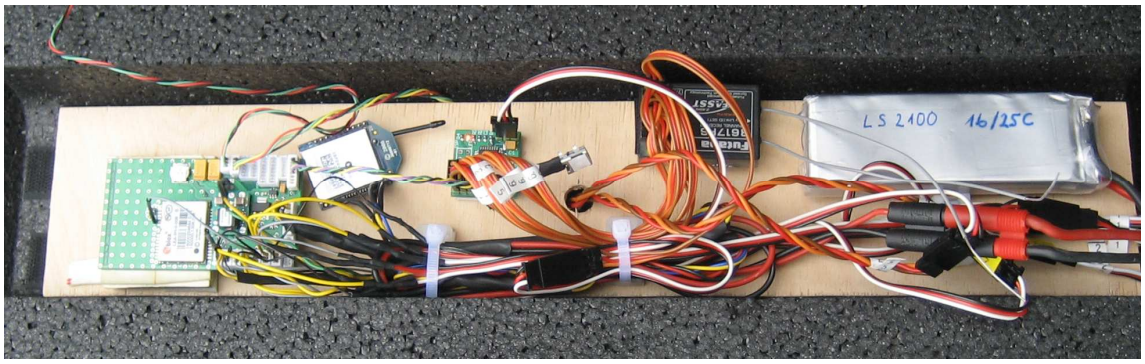

Abb. 3 Verkabelung des Autopiloten mit IR-Sensoren und zwei Akkumulatoren

2.2 Justierung der IR-Sensoren

Die Fluglage, die mit den Infrarot-Sensoren errechnet wird, kann stark von der effektiven Lage abweichen. Der Grund hierfür ist, dass die Lage aus einer Funktion der Differenzspannungen von den Ausgängen der Infrarot-Sensoren errechnet wird. Herstellungsbedingte Abweichungen und Montageplatz auf dem Flugzeug erfordern eine Justierung der Parameter.

Um die Werte möglichst genau einstellen zu können, muss ein geeigneter Platz gefunden werden, auf dem für die Sensoren möglichst die gleichen Bedingungen herrschen wie in der Luft. Hierfür entschieden wir uns für einen Aussichtsturm oberhalb von Frauenfeld (Stählibuck). Der Horizont ist dort ziemlich flach und man hat eine gute 360° Aussicht ohne Hindernisse, die die Sensoren stören können. In der Folge sollen nun kurz die wichtigsten Schritte für die Kalibrierung der Infrarot-Sensoren erläutert werden.

Abb. 4 Platzierung und Montage der IR-Sensoren an der MAJA

1) **Kaffebechertest**
   Man stülpt einen Becher oder etwas Ähnliches über die Sensoren, so dass jeder Sensor den gleichen Wert messen sollte. Wenn dies nicht der Fall ist, müssen die Werte im Airframe (*ADC_IR1_Neutral*, *ADC_IR2_Neutral*, *ADC_TOP_Neutral*) des entsprechenden Flugzeuges angepasst werden.
2) **Nullpunkt einstellen**
   Je nach Montage der Sensoren auf dem Flugzeug (z.B. asymmetrisch oder schräg), muss der Nullpunkt des künstlichen Horizontes neu eingestellt werden. Hierfür empfiehlt es sich, auf einen Platz mit guter Rundumsicht zu gehen. Die Neutrallage (kein Pitch und kein Roll) kann im GCS mit *RollNeutral* und *PitchNeutral* eingestellt werden. Diese Werte müssen aber später ins Airframe geschrieben werden.
3) **Steigung der Kennlinie**
   Der errechnete Roll- bzw. Pitch-Winkel muss noch so angepasst werden, dass er mit der Realität übereinstimmt. Dieser kann ähnlich wie der Nullpunkt im GCS über die Variablen *lateral_correction, longitunal_correction* und *vertical_correction* parametriert werden.

2.3 Parametrieren der Regelung

Dieser Abschnitt wendet sich insbesondere an Leser dieses Berichts, die sich gerade an einer Projektarbeit befinden, die mit dem Autopiloten zu tun haben. Es stellt den Ansatz dar, wie wir schnell und einfach zu relativ guten Regelparametern gekommen sind. Das Feintuning ist aber eine langwierige Angelegenheit, die je nach Ansprüchen sehr lange dauert.

Als Hilfe befindet sich im Anhang 8.1 eine Übersicht, welche die von uns verwendeten Parameter beschreibt.

1) **Grobeinstellungen im Auto1-Mode**
   Als erster Ansatz begannen wir mit der Justierung der Roll- und Pitchparameter. Je grösser sie sind, desto stärker und schneller reagieren die Klappen auf eine Sollwertränderung oder Störgrösse. Das Tuningverfahren auf der Paparazzi-Internetseite [1] empfiehlt folgendes Prozedere: Wenn das Flugzeug ins Schwingen kommt, ist der Maximalwert erreicht und muss etwas reduziert werden. Unsere Erfahrungen zeigen aber, dass wir die Parameter relativ stark reduzieren mussten um gute Flugeigenschaften im Auto2 Modus zu erzielen. Weitere Informationen zu diesem Punkt finden Sie unter Punkt 3.8 sowie 3.9.
   Die Roll- und Pitch-Eigenschaften werden über folgende Parameter justiert:

   roll_atitude_pgain

   pitch_pgain
   pitch_dgain (sollte aber relativ klein bleiben, da sonst der Servo zittert einsetzt)

2) **Schräg auf Linie anfliegen**
   Um einen ersten, gut zu überprüfenden Eindruck über das Regelverhalten des Flugzeuges zu bekommen, war der nächste Schritt, das Flugzeug schräg auf eine Gerade fliegen zu lassen, der es dann folgen sollte. Die Flugroute sollte dabei möglichst direkt sein, aber kein Überschwingen zeigen. Falls es zum Überschwingen kommt, empfiehlt es sich als Erstes den *course_pgain* zu

verringern. Er ist verantwortlich für die Position des orangen Punktes (Carrot) auf die das Flugzeug zufliegt.

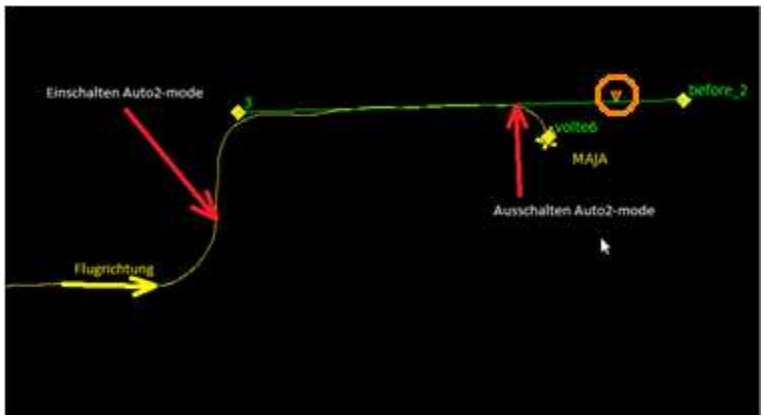

Abb. 5 Kurs Tuning:

course_pgain (*course_dgain* ist im Normalfall 0)
aber auch mit *roll_attitude_pgain* sollte gearbeitet werden

Mit der Höhe kann ähnlich vorgegangen werden wie mit dem Kurs.

altitude_pgain
Aber auch hier kann mit *pitch_pgain* experimentiert werden.

3) Kreis fliegen und sukzessive Verkleinerung des Radius

Der Navigationsradius (*nav_radius*) ist für eine gute Flugroute wichtig und sollte deshalb experimentell ermittelt und nach Wunsch verkleinert werden. Ein wichtiger Parameter für diese Einstellungen ist der *elevator_of_roll*. Er mischt bei Schräglage des Flugzeuges noch Pitch dazu, was zu engeren Kurvenradien führt aber auch das Flugzeug an Höhe gewinnen lässt. Mit zu engen Kurvenradien sollte deshalb vorsichtig umgegangen werden.

elevator_of_roll

Aber auch mit folgenden Parametern sollte wieder experimentiert werden.

roll_atitude_pgain, *pitch_pgain*, *altitude_pgain*, *course_pgain*

Wie oben schon erwähnt, stellen diese Ausführungen nicht eine komplette Anleitung dar, sondern sind sie Anhaltspunkte für erste Versuche mit dem System. Es empfiehlt sich darum auch, die restlichen Parameter anzuschauen und allenfalls zu verwenden.

3 Inertial Measurement Unit für Lageregelung

3.1 Anforderungen

Die Standardausführung des verwendeten Autopiloten Paparazzi benötigt für die Erkennung der Fluglage Infratorsensoren. Diese errechnen mittels der Temperaturdifferenz zwischen Boden und Atmosphäre die Fluglage. Dies kann bei speziellen Wettersituationen zu Problemen führen, da zum Beispiel am Morgen die Erde kälter als die Luft sein kann. Die Infrarot-Sensoren müssen ausserhalb des Flugkörpers angebracht werden, so dass diese die Umgebung erfassen können. Dies ist der zweite Schwachpunkt der Infrarot-Sensoren, da die Leiterbahnen des Sensors nicht geschützt sind. Ein weiteres Szenario, dass zu Problemen führen könnte, ist zum Beispiel ein Flug entlang einer Felswand, wo eine falsche Lage gemessen werden würde.

Durch die Integration einer IMU sollen die Schwachstellen der Infrarot-Sensoren behoben werden, so dass die Drohne unabhängig von der Topologie und der aktuellen Wetterlage die Messkampagne durchführen kann.

3.2 Theorie Einführung

Eine IMU misst die sechs Freiheitsgrade eines Körpers mittels Beschleunigungs- und Drehratensensoren (Gyroscope). Es stehen also drei Gyroscopes und drei Beschleunigungssensoren für die Bestimmung der Lage zur Verfügung. Dies bestimmt grundsätzlich die Lage des Körpers vollständig, wenn nicht der Drift der Gyroscopes wäre. Ein Gyro hat einen sogenannten Drift, der laufend überprüft und abgeglichen werden muss. Der Drift der X- und Y-Achsen wird über die Änderung des gemessenen Gravitationsbeschleunigungsvektors abgeglichen. Wenn also der Beschleunigungsvektor konstant bleibt, erkennt der IMU die Winkeländerung um die beiden genannten Achsen als Drift und kompensiert ihn. Das Problem besteht aber immer noch in der Z-Achse. Eine Rotation (langsam, Zentrifugalkraft wird vernachlässigt) um die Z-Achse führt aber keine Änderung des Gravitationsbeschleunigungsvektors an der X- und Y-Achsen herbei, was dazu führt, dass der Drift nicht erkennt werden kann und deswegen wandert (Siehe Abb. 10). Ein weiteres Problem für die Eliminierung des Drifts und eine saubere Erkennung der Lage ist in einem bewegten Körper wie der Drohne, das Kennen der Zentrifugalkräfte in Kurvenflügen und Höhenwechsel. Für den Abgleich des Drifts muss die Richtung der Erdbeschleunigung bekannt sein. Während dem Fliegen von Kurven verändert sich die Summe aus Erdbeschleunigung und Zentrifugalkraft dauernd. Dies macht es für den IMU unmöglich, ohne das Wissen über die Grösse der Zentrifugalkraft, gute Resultate zu liefern. Um diese Informationen zu beschaffen werden vor allem zwei Verfahren verwendet. Zum einen kann über GPS (oder ein ähnliches System) Informationen zu Bahnradien und deren Beschleunigungen, zum anderen über ein sogenanntes Magnetometer Informationen über die Ausrichtung des Flugzeuges beschafft werden. Der Magnetometer wird in einem IMU im Wesentlichen als elektronischen Kompass verwendet, aus dessen Daten auf Kursänderungen geschlossen werden kann und somit den Drift um die Yaw-Achse eliminieren.

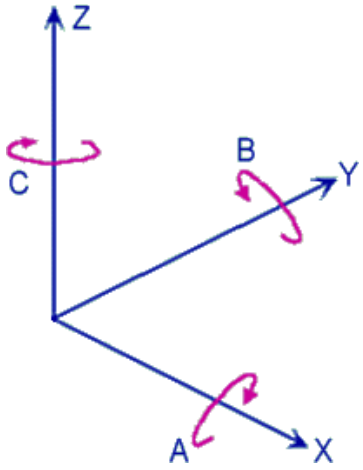

Abb. 6 Koordinatensystem

3.3 Welche IMU kommen in Frage

Für die Auswahl einer geeigneten IMU haben wir drei verschieden Preisklassen untersucht. In der teuersten Preisklasse haben wir zwei Sensoren von X-Sens und die Produktpalette von MicroStrain angeschaut. In einer zweiten Preisklasse gibt es den VN100 von VectorNav. Die deutlich billigsten Sensoren sind die Open Surce Produkte ArduImu+ V2 und die Razor IMU Familie.

Die X-Sens Produkte MTi und MTi-G sind kommerziell erhältliche Sensoren. Mit dem Development Kit kosten die Sensoren 1990 Euro (MTi) respektive 3790 Euro (MTi-G). Die grössere MTi-G ist eine GPS unterstützte, auf MEMS basierte IMU. Dem anderen Sensor fehlt die GPS-Einheit. Beide sind jedoch mit einem Magnetometer ausgestattet.

Die Sensoren von MicroStrain der Familie 3DM sind in etwa von der gleichen Güte und Preisklasse wie die oben erwähnten X-Sens. Wir denken, dass diese kommerziellen Sensoren für unseren Gebrauch zu genau sind. Auch die angebotene Schnittstelle RS232 ist für unseren Autopiloten nicht geeignet.

Der VN100 von VectorNav ist mit 500 $ bedeutend günstiger und bietet diverse Schnittstellen zur Auswahl. Darunter ist auch die SPI, welche von Autopiloten unterstützt wird.

In der untersten Preisklasse befinden sich die Open Source Produkte Razor und ArduImu. Der Razor hat drei Messsysteme, welche sich gegenseitig unterstützen. Dies sind je ein Beschleunigungssensor, ein Gyroskop und ein Magnetometer pro Achse. Der Ausgang des Razor ist jedoch lediglich ein serielles Signal.

Der Vorteil der Open Surce ArduImu ist die vorhandenen SPI und I²C Schnittstellen. Da diese IMU nur die Beschleunigungssensoren und Gyros integriert hat, müssen allenfalls Magnetometer und GPS-Signal angeschlossen werden. Die IMU hat somit Anschlüsse, mit denen der Drift um die Z-Achse kompensiert werden kann.

Hersteller	Typ	Preis [CHF]	SPI	Seriell	I²C	GPS	Magnetometer
X-Sens	MTi-G	5300		X		X	X
X-Sens	MTi	2800		X			X
Microstrain	3DM	ab 2700		X		X	X
VectorNav	VN100	500	X	X			X
Sparkfun	Razor	140		X			X
DIYDrones	ArduImu	100	X	X	X		

Tabelle 1: Verschiedene IMU-Sensoren im Vergleich

3.4 Mögliche Schnittstellen

Für die Implementierung einer IMU muss eine geeignete Schnittstelle ausgewählt werden, welche von Autopiloten und IMU unterstützt wird. Der ArduImu, sowie das Autopiloten-Board bieten verschiedene Schnittstellen an, die für die Kommunikation in Frage kämen. Aufgrund der Tatsache, dass die meisten IMU auf dem Markt ein serielles Ausgangssignal haben, wäre dies die favorisierte Schnittstelle. Das Autopiloten-Board besitzt aber nur zwei serielle Kanäle, die durch Downlink und GPS schon besetzt sind. Als Alternativen steht aber noch SPI (Serial Peripheral Interface) oder I²C (Inter-Integrated Circuit) zur Verfügung. Diese beiden Schnittstellen sind weit verbreitet und werden von beiden Mikrokontrollertypen (ArduImu (Atmega328) und Autopilot (ARM7 LPC2148)) unterstützt.

Eine weitere, aber eher aufwendige Möglichkeit wäre, weitere Anschlussmöglichkeiten mit Hilfe eines zusätzlichen Mikrokontrollers zu schaffen. Die Kommunikation zum zusätzlichen Kontroller müsste über eine der erwähnten Schnittstellen mit dem Autopiloten stattfinden.

Die Hardware des Paparazzi Autopiloten Tiny V2 stellt grundsätzlich die folgenden Schnittstellen zur Verfügung:

3.4.1 Analog

Es stehen sieben Analogeingänge zur Verfügung, wobei drei davon in der Standardausführung durch die Infrarot-Sensoren verwendet werden. Für die Anbindung einer IMU kommt diese Schnittstelle nicht in Frage, da keine IMU gefunden werden konnten die ein analoges Signal bereitstellt.

3.4.2 Seriell

Die serielle Schnittstelle dient dem Datenaustausch zwischen Computern und Peripheriegeräten. Bei einer seriellen Datenübertragung werden die Bits nacheinander (seriell) über eine Leitung übertragen. Wenn ohne nähere Kennzeichnung von einer „seriellen Schnittstelle“ gesprochen wird, ist damit fast immer die CCITT-V.24 bzw. EIA-RS-232-Schnittstelle gemeint. Die Bedeutung und Anordnung der Bits wird bei der RS-232 mit einem sogenannten UART (Motorola-Bezeichnung) oder USART (Intel, Zilog) vorgegeben. Moderne serielle Schnittstellen sind Ethernet, USB, Firewire, CAN-Bus oder RS-485, allerdings werden diese landläufig nicht als serielle Schnittstellen bezeichnet.[2]

3.4.3 SPI (Serial Peripheral Interface)

Das Serial Peripheral Interface (kurz SPI) ist ein von Motorola entwickeltes Bus-System mit einem sehr lockeren Standard für einen synchronen seriellen Datenbus, mit dem digitale Schaltungen nach dem Master-Slave-Prinzip miteinander verbunden werden können. [3]

Die wichtigsten Eigenschaften dieser Schnittstelle sind:

- SPI hat 3 Leitungen: MISO Master-In-Slave-Out
  MOSI Master-out-Slave-In
  CLK Clock
- Clock wird vom Master ausgegeben
- Vollduplexfähig
- Der Standard unterstützt nur zwei Adressen. Es ist aber grundsätzlich möglich, beliebig viele Teilnehmer an den Bus anzuschliessen, wobei es immer einen Master geben muss.
- Normalerweise für schnellen Datentransfer zwischen zwei Geräten mit bis zu 10 Mbit/s.

3.4.4 I^2C (Inter-Integrated Circuit)

Der I^2C Bus wurde von der Firma Philips für die Kommunikation zwischen integrierten Schaltungen entwickelt. Heute steht eine Vielzahl von Komponenten zur Verfügung (ca. 400), z.B. A/D-und D/A-Wandler, Temperatursensoren, parallele und serielle Schnittstellen, Prozessoren und so weiter. Der I^2C Bus kommt vor allem in einfacheren Geräten zum Einsatz, wo Durchsatz nicht höchste Priorität hat, sondern eine sichere und einfache Kommunikation zwischen den Bausteinen. Die I^2C Peripheriebausteine lassen sich zudem ohne grossen Aufwand in Software über I/O-Ports ansteuern. [4]

Die wichtigsten Eigenschaften dieser Schnittstelle sind:

- Einfaches Master/Slave Bussystem mit Adressierung
- Mehrere Master sind möglich

- Für mehrere Geräte, 7 Bit Adressraum
- 100 KHz, 400 KHz bis 1 MHz
- Zwei Drähte, SDA (Data) und SCL (Clock)

3.5 Bestehender Source Code von Paparazzi

Eine wichtige Frage, die im Vorfeld geklärt werden sollte, ist die Frage, was bereits im Source Code von Paparazzi implementiert ist. Da das Projekt Paparazzi eine grosse Community hat, gibt es bereits einige IMUs, für die Treiber bereits vorhanden sind. Die Verwendung dieser Sensoren wäre mit einem kleinen Aufwand realisierbar, deshalb lohnt es sich, diese Lösungen zu untersuchen. Es gibt hauptsächlich zwei Projekte welche publiziert sind. Diese werden im Folgenden beschrieben.

3.5.1 OSAM

Hinter dem OSAM-UAV-BFB (Open Source Autonomous Multiple-Unmanned Aerial Vehicle-Build Fly and Beyond) Projekt [5] verbirgt sich die Utah State University. Laut ihrer Internetseite ist ihr Ziel, eine low-cost Drohne zu entwickeln, welche für die Luftraumüberwachung oder für Forschungszwecke eingesetzt werden kann.

Um ihre Ziele zu erreichen, setzten sie die IMU Gx2 von MicroStrain ein. Jedoch sollten ihre Treiber auch mit der X-Sens MTi-G funktionieren. Denn beide Sensoren stellen einen seriellen Ausgang zur Verfügung. Für die Verarbeitung des Signals nutzen sie einen zusätzlichen Kontroller, der dem Paparazzi-Board über analoge Eingänge die Lage des Flugzeugs mitteilt.

Zur Anwendung ihrer Dateien gibt es eine Readme-Datei, welche die nötigen Änderungen und Einstellungen dokumentiert. Diese Datei findet man unter dem Pfad:

~/paparazzi3/doc/OSAM_IMU/readme_pprz_osam_imu.txt

3.5.2 VN100

Als Modul kann der Sensor VN100 von VectorNav eingebunden werden. Dazu muss nur die xml-Datei des Airframes angepasst werden. Als Vorlage dazu kann der Twinjet2 von ENAC konsultiert werden. Die Datei findet man unter dem Pfad:

~/paparazzi3/conf/airframes/ENAC/fixed-wing/twinjet2.xml

Die Kommunikation mit dem Paparazzi Controller läuft dabei über die SPI-Schnittstelle.

3.5.3 Gyro

Eine weitere Möglichkeit, die bereits im Quellcode vorhanden ist, stellt der Einsatz eines Gyros dar. Dieser wird mit dem Setzten des Flags „Gyro“ in der xml-Datei des Airframes eingebunden. Es ist angedacht, dass ein Gyro eingesetzt wird, welcher einen analogen Ausgang anbietet. Mit Hilfe dieses Sensors kann die Regelung nicht nur mit der Lage rechnen, sondern auch mit den Beschleunigungen um die Roll- respektive Pitch-Achse. Jedoch stellt diese Variante sicherlich keinen Ersatz für eine vollwertige IMU dar.

3.6 Wahl der IMU und der Schnittstelle

Der ArduImu ist eine „Inertial Measure Unit“ mit einem Arduino kompatiblen Prozessor, auf dem das AHRS (Attitude Heading Reference System) läuft. Die Hardware besteht aus drei Beschleunigungssensoren (X-, Y-, Z-Achse), drei Gyro-Sensoren, einen GPS-Port sowie einem Atmega 328.

Die Software ist Arduino-Kompatibel und Open Source. Ausserdem verfügt das Bord über drei verschiedene Schnittstellen (I²C, SPI und Seriell), was die Anbindung an andere Mikrokontroller, Computer oder Messmittel sehr offen hält.

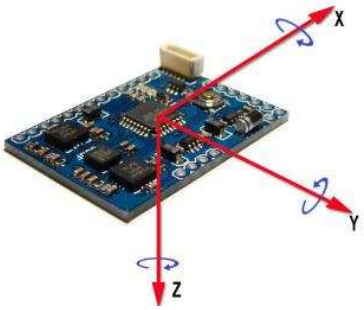

Abb. 7 ArduImu +V2 flat [6]

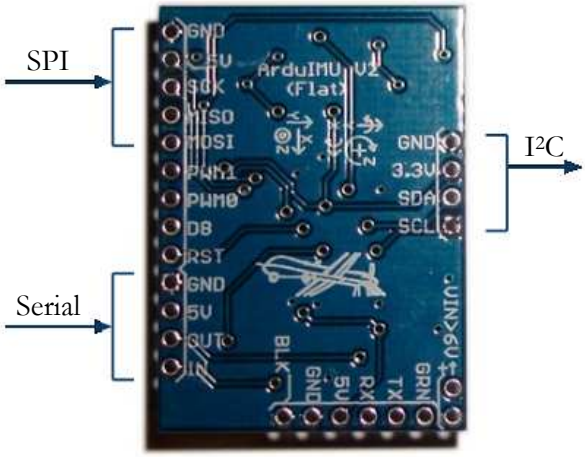

Abb. 8 ArduImu +V2 flat Grundriss [7]

Die benötigte Genauigkeit, die es zum autonomen Fliegen benötigt, wird wahrscheinlich vom ArduImu erzielt. Deshalb fiel unsere Wahl auf diese Open Source IMU. Sollte die IMU dennoch nicht unseren Erwartungen gerecht werden, so haben wir zumindest erste Erfahrungen mit der Implementierung der Schnittstelle und der Verarbeitung der Messungen sammeln können, ohne das Budget gross zu belasten.

Der ArduImu wurde ursprünglich für den ArduPilot [8] entwickelt und basiert auf der Arduino-Platform. Die momentan aktuelle Version ist der ArduImu+ V2 flat. Er ist ausgerüstet mit einem Beschleunigungssensor von Analog Devices [9], der die Beschleunigungen in drei Richtungen misst. Er kann Beschleunigungen bis zur dreifachen Erdbeschleunigung messen. Die Sensoren für die Messung der Drehraten werden von STMicroelectronics [10] hergestellt und basieren auf der MEMS-Technologie. Sie sind ausgelegt für bis zu 300 °/s zu messen. Hardwaremässig ist ein Sensor für die Roll- (X-Achse) und Nick-Rate (Y-Achse) zuständig und ein zweiter für die Gierbewegung (Z-Achse).

Dem IMU fehlt standardmässig eine Komponente für die Kompensierung des Drifts um die Z-Achse. Jedoch ist der IMU so vorbereitet, dass verschiedene GPS-Empfänger angehängt werden können. Darunter auch der von Paparazzi verwendete u-blox Empfänger. Die andere Variante zur Messung des Drifts mittels Magnetometer ist softwaremässig auch vorbereitet.

Zur Kommunikation mit anderen Geräten ist eine serielle Schnittstelle vorgesehen, jedoch kann die Verbindung auch über ein I²C oder SPI Interface erfolgen. Das Letztere ist jedoch noch nicht implementiert und muss folglich vom Nutzer selbst programmiert werden. Somit empfiehlt sich die Nutzung des I²C-Protokolls, da es für die Arduino-Familie bereits eine Bibliothek gibt, welche die Einbindung der Schnittstelle vereinfacht. Die serielle Schnittstelle, welche eigentlich den Standard der ArduImu darstellt, kann nicht genutzt werden, weil die auf dem Paparazzi-Board vorhandenen Leitungen bereits besetzt sind.

3.7 Analyse der IMU

3.7.1 Drift

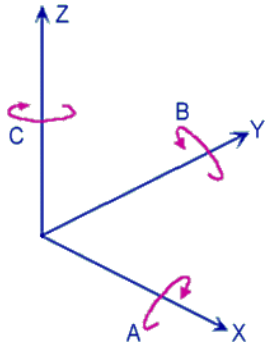

Abb. 9 Achsensystem

Ein Körper im Raum besitzt sechs Freiheitsgrade, drei in Achsrichtung sowie drei rotative um die Achsen. Mit sechs Sensoren sollten also alle Freiheitsgrade definiert sein. Das Problem liegt im Drift der Sensoren, die durch die zeitliche Integration zu Position- und Winkelfehler führen. Bei einem Gyro äussert sich dieser Fehler in einem stetig wachsenden Winkel obwohl die IMU bewegungslos bleibt. Dieser Fehler kann aber mit Hilfe der Beschleunigungssensoren auf einer anderen Achse abgeglichen und eliminiert werden. Beispielsweise kann der Drift um die X-Achse abgeglichen werden, indem man die Beschleunigungen, die durch die Gravitation auftreten, auf der Y- und Z-Achse beobachtet. Dasselbe gilt auch für die Y-Achse, aber bei der Z-Achse entfallen diese Beschleunigungen. Das bedeutet, dass für den Drift der Z-Achse ein zusätzliches Messmittel nötig ist. Es werden vor allem zwei Methoden eingesetzt um diesen Fehler zu eliminieren. Zum einen kann über GPS der COG (Course Over Ground) errechnet werden. Dazu muss man sich aber bewegen, was für viele Anwendungen nicht möglich ist. Zum anderen kann ein neigungskompensierter dreidimensionaler Kompass verwendet werden.

Um den Drift der Achsen im Stillstand aufzuzeichnen, wurde eine Messung durchgeführt. Die Resultate sind in dem nächsten Diagramm abgebildet. Es ist ersichtlich, dass, entsprechend der Theorie, der Roll- und Pitch-Winkel keinen Drift aufweisen. Wie oben beschrieben, ist der Abgleich des Yaw-Winkels problematisch und somit verfälscht sich der Messwert diese Winkels andauernd.

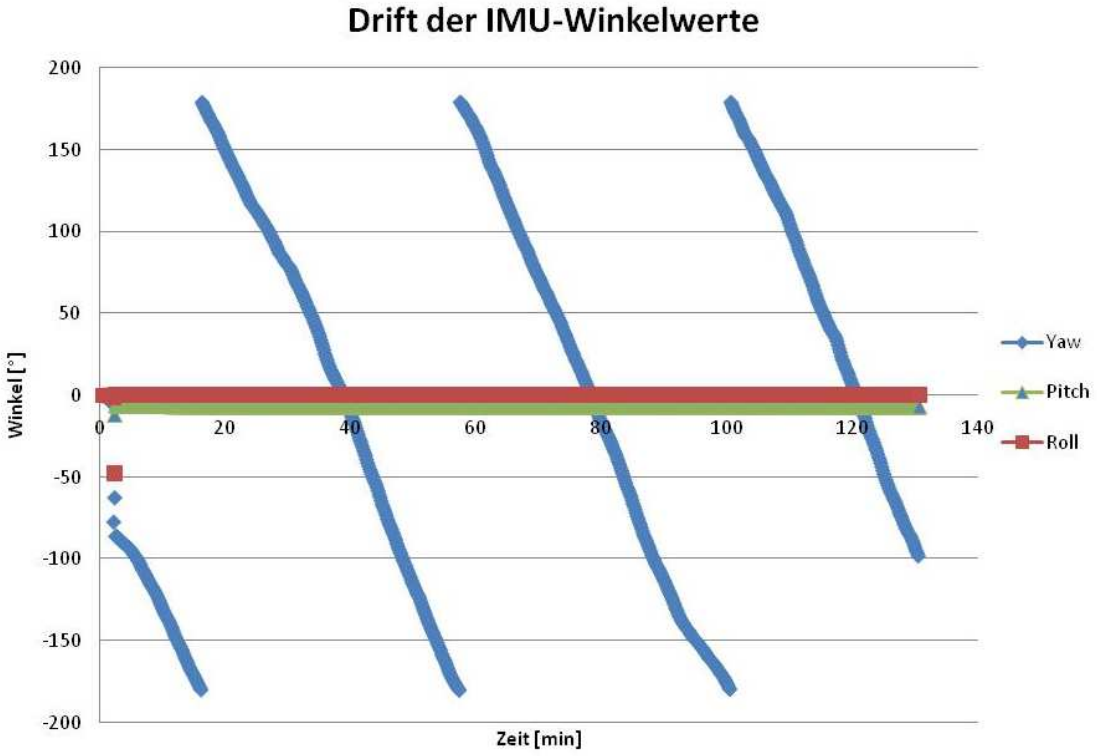

Abb. 10 Drift um die Z-Achse (Yaw) der IMU im Stillstand

3.7.2 Abgleich der Sensoren

Im bewegten System tritt nicht nur die Erdanziehungskraft auf, sondern auch Fliehkräfte, welche abhängig von der zurückgelegten Bewegung sind. Hat die IMU keine Informationen über ihre Laufbahn, so gleicht er seine Gyros mit den Beschleunigungssensoren ab, ohne zu berücksichtigen, dass eventuell die gemessenen Beschleunigungen nicht der Erdanziehungskraft entsprechen.

Um diesen Fehler zu beheben braucht die IMU Daten über ihre Bewegung. Diese Information kann zum Beispiel ein GPS-Empfänger liefern. Um die Genauigkeit zu erhöhen, kann man zusätzlich zum GPS-Signal ein elektrischer Kompass verwenden. Die Theorie über den Nutzen des Kompasses wird im Kapitel 3.7.3 beschrieben.

Der auf dem Autopiloten eingesetzten GPS-Empfänger (u-blox LEA-5) [12] hat eine Updaterate von 4 Hz. In den nachfolgenden Graphen wird aufgezeichnet, was die Updaterate für einen Einfluss auf die Genauigkeit und Stabilität hat. Als Versuch wurden eine langsame Rate von 2 Hz und eine schnelle Rate von 4 Hz einander gegenübergestellt. Es ist ersichtlich, dass die Steigung des linearisierten Wertes mit erhöhter Frequenz deutlich kleiner ist. Der Mittelwert zeigt in etwa den Fehler, welcher durch den Abgleich der Daten entsteht. Das Flugzeug misst somit je länger das Flugmanöver andauert einen grösseren Roll-Winkel, was den künstlichen Horizont immer weiter von der Realität abweichen lässt.

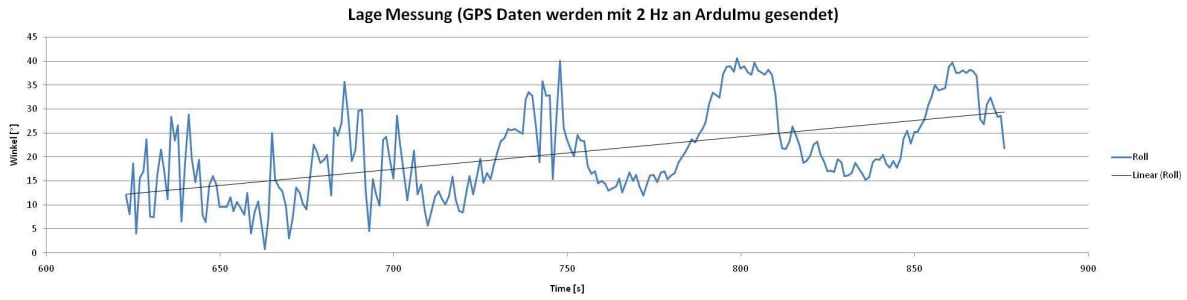

Abb. 11 Winkel-Fehler durch Abgleich der IMU mit GPS Informationen (2 Hz) während einem Flugmanöver in dem im Uhrzeigersinn gekreist wird.

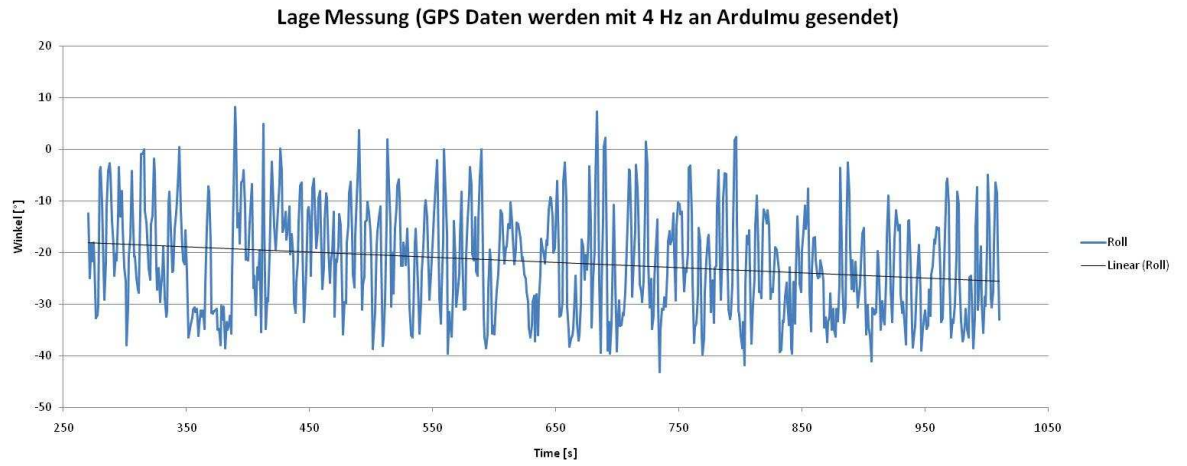

Abb. 12 Winkel-Fehler durch Abgleich der IMU mit GPS Informationen (4 Hz) während einem Flugmanöver in dem im Gegenuhrzeigersinn gekreist wird.

Bei einer Updaterate der GPS-Daten für die IMU von 2 Hz verfälscht sich die Winkelmessung so schnell, dass der Pitch-Regelkreis bereits nach vier Umdrehungen in einem Kreis mit 100 m Radius an seine Grenzen stösst. Bei einem weiteren Testflug mit erhöhter Frequenz von 4 Hz konnte in über 12 Minuten in einem Kreis mit 60 m Radius geflogen werden, ohne dass die Regelung nicht mehr funk-

tionierte. Jedoch verfälschte sich auch mit der schnelleren Updaterage der gemessene Roll-Winkel dauernd, was zu einem wachsenden Kreisradius führte.

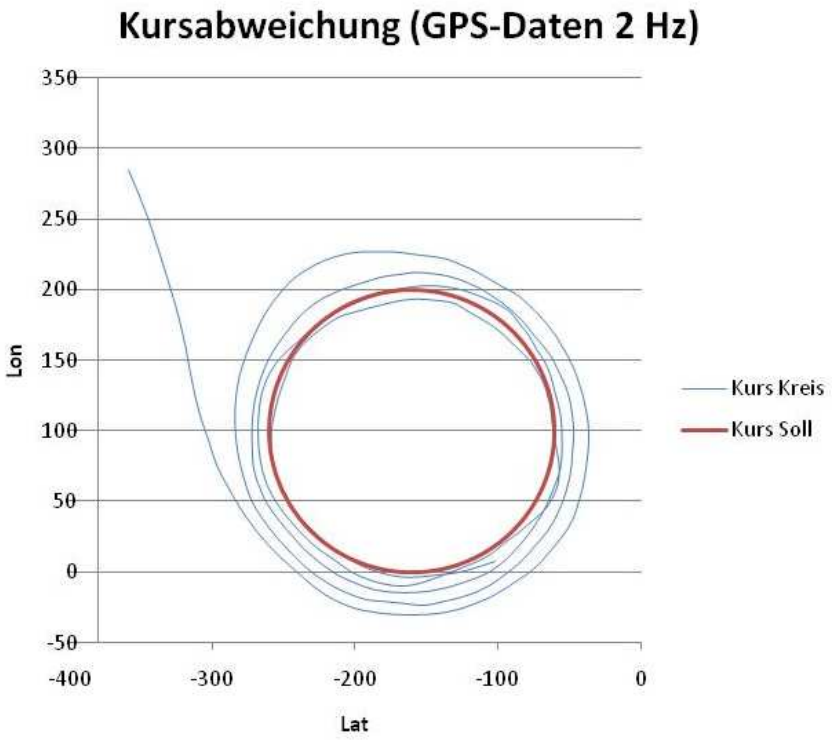

Abb. 13 Kursabweichung von einem Kreis mit 100 m Radius bei einer GPS-Daten Updatefrequenz von 2 Hz

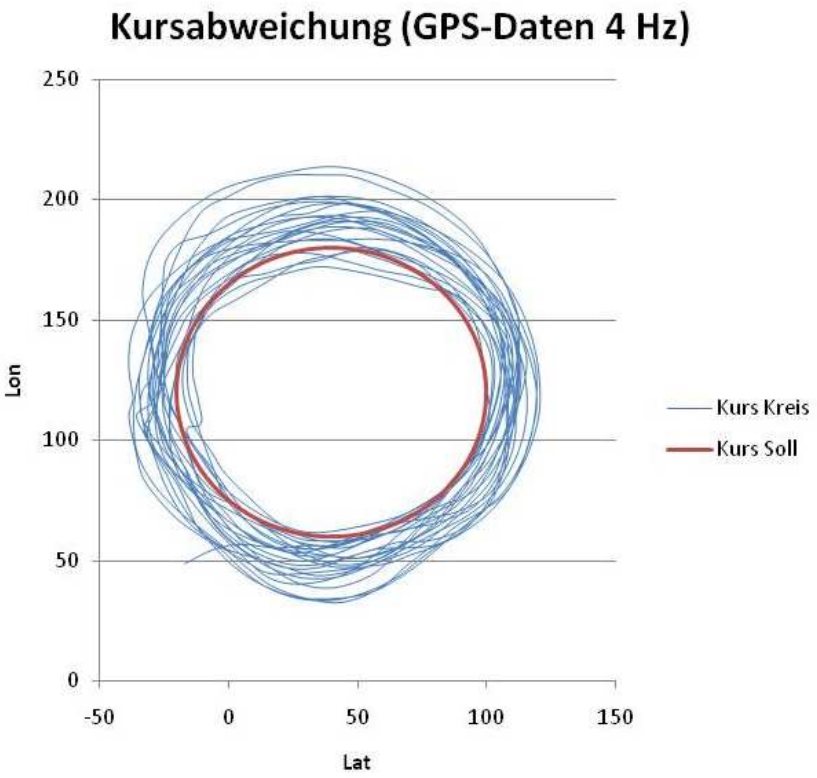

Abb. 14 Kursabweichung von einem Kreis mit 60 m Radius bei einer GPS-Daten Updatefrequenz von 4 Hz

Trägt man den geflogenen Radius gegenüber der Zeit auf, wird die geflogene Spirale ersichtlich. Siehe dazu die unten stehende Abbildung.

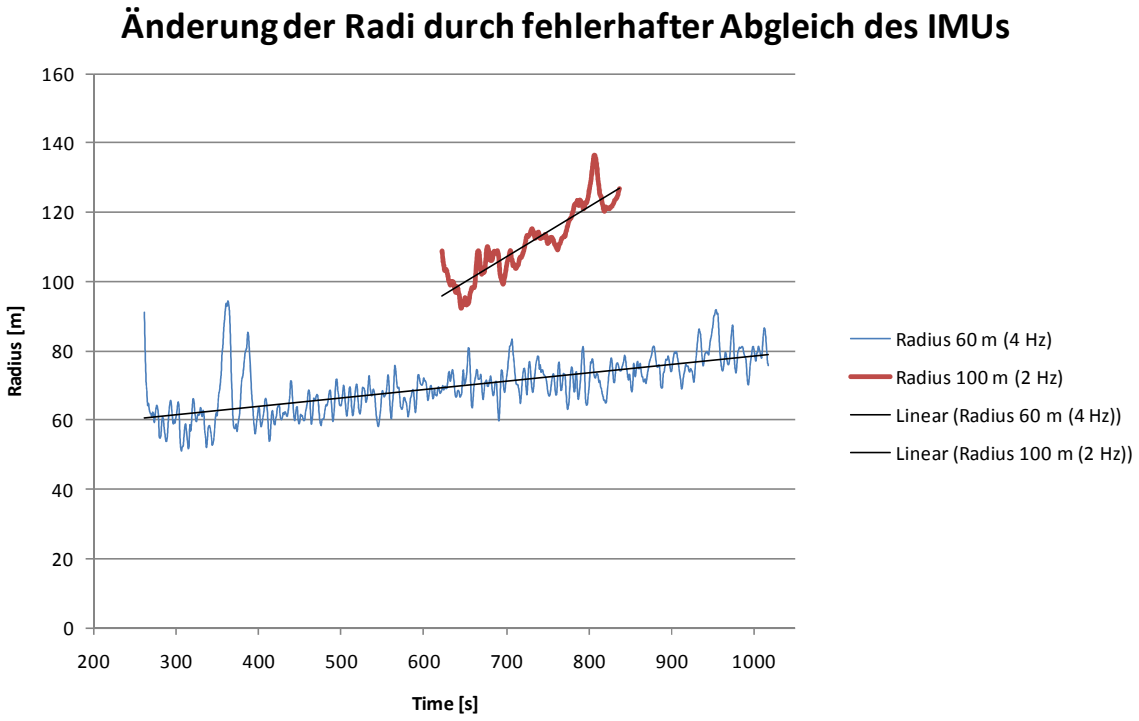

Abb. 15 Änderung der Radien durch fehlerhafter Abgleich der IMU

3.7.3 Weitere Verbesserungen für Abgleich (Kompass)

Die Versuche mit der erhöhten GPS-Updaterate haben sehr gute Resultate geliefert. Da sie aber beim GPS-Modul (u-blox Lea 5H) auf 4 Hz beschränkt ist, müssen auch andere Überlegungen mit-einbezogen werden. Wenn das Flugzeug eine Kurve fliegt, addieren sich die Erdbeschleunigung und die Zentrifugalkräfte zu einem gemeinsamen Vektor. Um die Driftkorrektur durchführen zu können, müssen die Zentrifugalkräfte bekannt sein, um sie vom gemeinsamen Vektor subtrahieren zu können. Die Zentrifugalkräfte in einer bestimmten Achse errechnet der IMU aus der GPS-Geschwindigkeit und dem Messwert des Gyros.

```
void Accel_adjust(void){
      Accel_Vector[1] += Accel_Scale(speed_3d*Omega[2]);  // Centrifugal force on Acc_y = GPS_speed*GyroZ
       Accel_Vector[2] -= Accel_Scale(speed_3d*Omega[1]);  // Centrifugal force on Acc_z = GPS_speed*GyroY
}
```

Der Roll-Winkel errechnet sich nun wie folgt aus Z- und Y-Beschleunigungssensor.

```
roll = atan2(Accel_Vector[1],Accel_Vector[2]);    // atan2(acc_y,acc_z)
```

Dies funktioniert bei Windstille, Gegen- oder Rückenwind gut. Das Flugzeug hat aber beispielsweise bei Seitenwind einen anderen Flugvektor als bei Rückenwind. Dies bedeutet, dass auch der Yaw-Winkel bei Seitenwind nicht dem errechneten GPS-Kurs entspricht und darum der errechnete Beschleunigungsvektor nicht mit der Realität übereinstimmt. Dieser Fehler fliesst dann bei der Berechnung des Roll-Winkels mit ein und ergibt einen Fehler der sich aufsummieren kann.

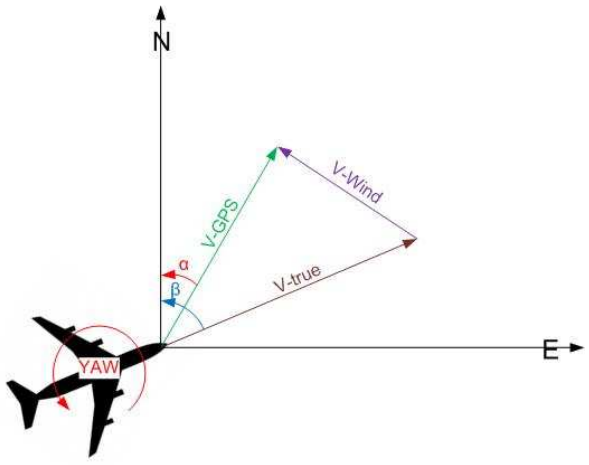

Abb. 16 Vektorfehler bei Abgleich ohne Magnetometer

Wie aus den Codeauschnitten ersichtlich ist, ist der Roll-Winkel direkt abhängig von der Winkelgeschwindigkeit der Z-Achse (Yaw). Der Drift um diese Achse kann aber mit Hilfe von GPS und Abgleich der Beschleunigungsvektoren nicht vollständig eliminiert werden und bedarf einer weiteren Messgrösse. Für das Erfassen der Ausrichtung der Drohne eignet sich ein elektronischer Kompass oder auch Magnetometer genannt. Dieser orientiert sich am natürlichen Magnetfeld der Erde und kann daraus auf die Position der Z-Achse zurückschliessen. Im Code des ArduImu ist die Integration eines Magnetometers bereits implementiert und würde voraussichtlich problemlos mit der ArduImu verbunden werden können. Ein mögliches Magnetometer währe der HMC5843 von Honeywell [27] der auch im Code der ArduImu verwendet wird. Es handelst sich um einen drei-Achsigen und relativ preisgünstigen Sensor der über I²C mit dem IMU verbunden werden könnte.

3.8 Horizontale Regelung

Die Regelung des Kurses und der Lage wird in der Datei *fw_h_ctl.c* gerechnet. Der Regelkreis des Kurses und der externe Höhen-Regelkreis stellen die äusseren Regelungen dar, welche den Sollwert der inneren Lageregelung berechnet. Die zwei Regelkreise werden im Folgenden aufgezeichnet.

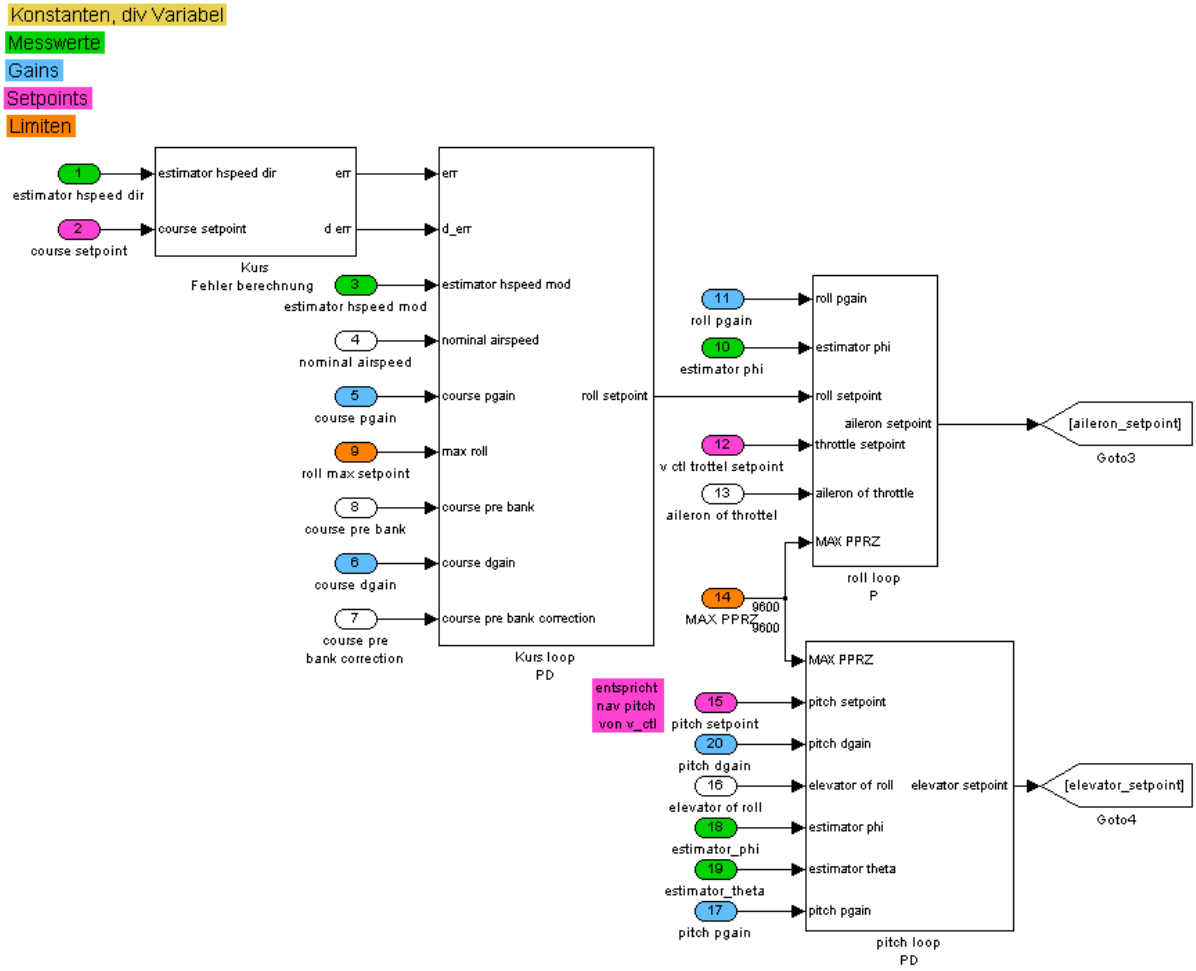

Abb. 17 Regelung des Kurses und der Lage

3.8.1 Kurs und Lage Regelung

Der Kurs Regelkreis rechnet die Kursabweichung zwischen dem gemessenen GPS-Vektor (*estimator_hspeed_dir*) und dem gewünschten Kurs (*course_setpoint*). Der Regelausgang *roll_setpoint* wird durch den Parameter *roll_max_setpoint* limitiert und als Sollwert für den Regelkreis der Querruder genutzt. In diesem inneren Regelkreis wird der Sollwert mit der gemessenen Lage der IMU (*estimator_phi*) verglichen.

Ähnlich verhält sich die Regelung der Höhenruder. Der äussere Regelkreis ist in der Datei fw_v_ctl.c programmiert und stellt dem inneren Regelkreis den Sollwert zur Verfügung (*pitch_setpoint*). Der Fehler wird ebenfalls mit der gemessenen Lage der IMU (*esimator_theta*) erechnet. Dabei wird jedoch zusätzlich der gemessene Roll-Winkel φ (*estimator_phi*) berücksichtig. Dies kommt beim Fliegen einer Kurve zum Zug, da das Flugzeug bei einem Roll-Winkel ungleich Null mit dem Höhenruder nicht nur die Höhe verändert, sondern auch den geflogene Kurs.

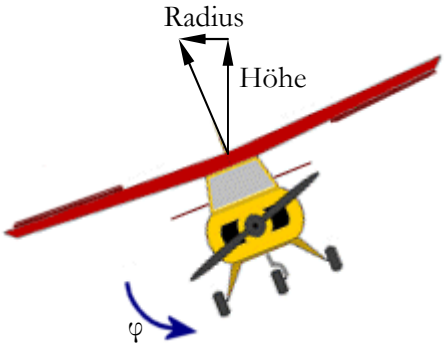

Abb. 18 Zusammenhang zwischen Quer- und Höhenruder

3.9 Parametrierung

Die Parametrierung der Lageregelung wird am Besten einzeln durchgeführt. Das heisst, dass man jeden Regelkreis nacheinander justiert, bis die gewünschte Regelgüte erreicht wird. Das Tuningverfahren, welches auf der Homepage von dem Paparazzi Projekt vorgeschlagen wird, sieht vor, dass man im Flugmodus Auto1 fliegt und anschliessend die Roll und Pitch Parameter einstellt, bis das Flugverhalten instabil wird. Anschliessend soll man die Werte verkleinern, bis die Fluglage wieder stabil ist. Es hat sich jedoch gezeigt, dass das Arbeiten mit einer gewollten Störgrösse einfacher ist. Der Pilot fliegt im Modus Auto1 und ändert mit einem Impuls an der Fernsteuerung den Sollwert. Danach kann das Verhalten beobachtet werden und Rückschlüsse auf die Regelparameter gezogen werden.

Wenn der gemessene Winkel nach einer Auslenkung sich nicht stabilisiert, ist der P-Faktor zu gross, oder wenn er sich zu langsam ausregelt, ist der Faktor zu klein eingestellt. Im Folgenden werden die Messungen aufgezeichnet, die das Verhalten der MAJA zeigen. Zum Vergleich wird jeweils auch eine Messreihe gezeigt, bei der die Regelung schwingt.

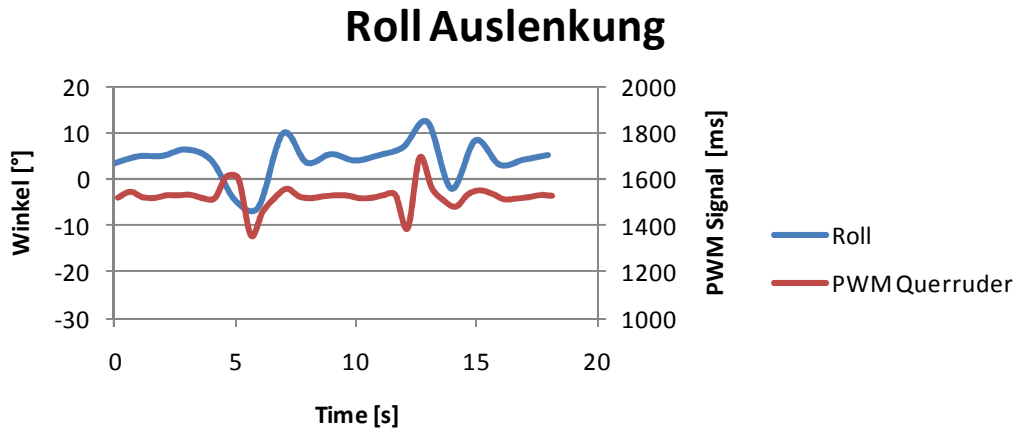

Abb. 19 Tuning der Roll Auslenkung

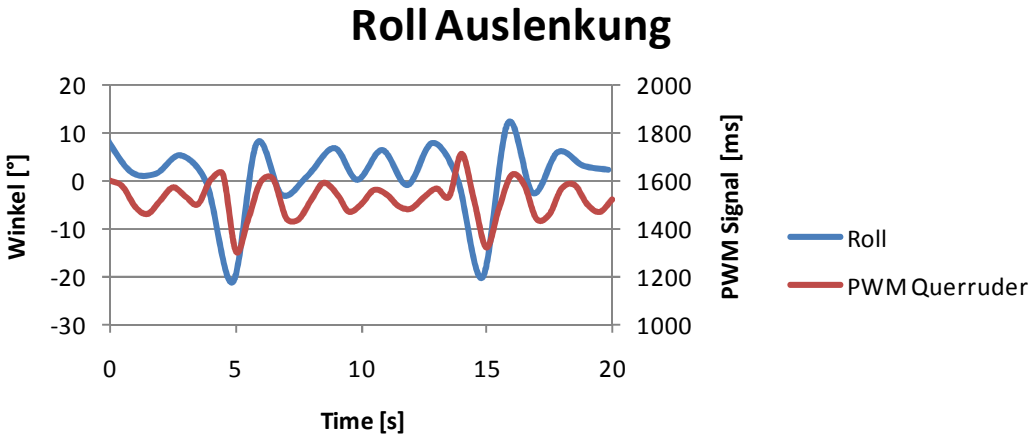

Abb. 20 Schwingung des Roll-Winkels nach einer Auslenkung des Sollwertes (roll_atitude_pgain zu gross)

Emilio Schmidhauser
René Chaney

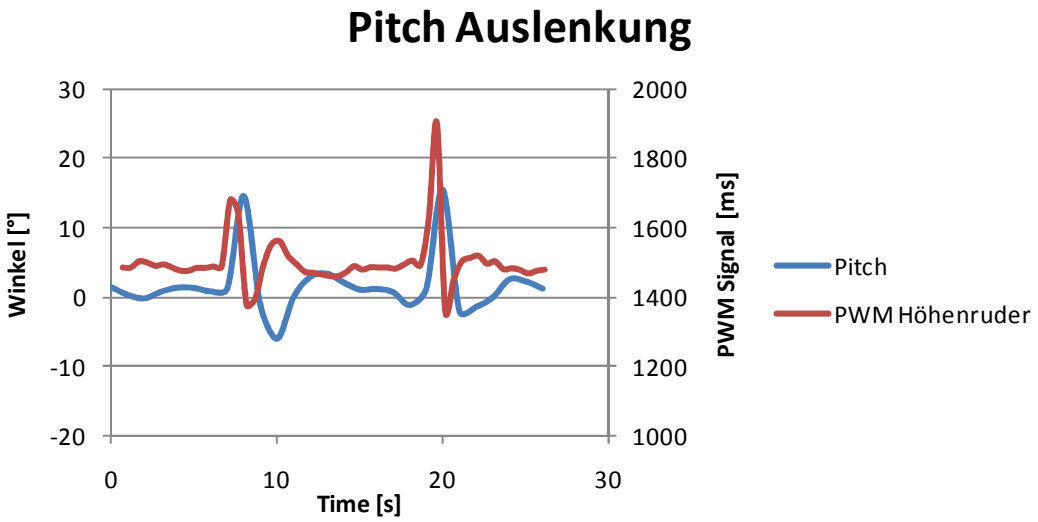

Abb. 21 Tuning der Pitch Auslenkung

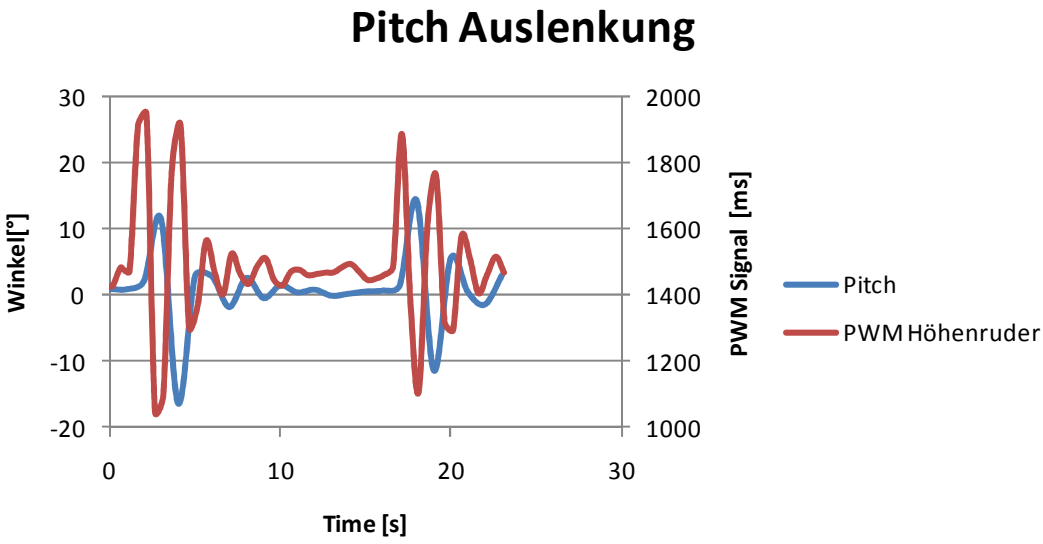

Abb. 22 Schwingung des Pitch-Winkels nach einer Auslenkung des Sollwertes (pitch_pgain zu gross)

4 Geschwindigkeits- und Höhenregelung

4.1 Anforderungen

Eine Anforderung an den Autopiloten ist, dass er die Fluggeschwindigkeit bezogen zur Luft in einem Bereich von ± 1 m/s regeln kann. Die Demonstrator-Drohne ist auf einen Cruisespeed von 70 km/h ausgelegt und wurde auf eine maximale Flughöhe von 5000 m.ü.M. ausgelegt. Der Nachfolger dieser Drohne soll aber auf ca. 130 km/h Cruisespeed ausgelegt werden. Die Geschwindigkeitsmessung muss diesen Anforderungen Rechnung tragen.

4.2 Einführung

Die Geschwindigkeit bezogen zur Luft kann mit einem Pitot-Rohr gemessen werden. Im Pitot-Rohr wird der Staudruck p_{kin} gemessen, wobei daraus mit folgender Beziehung auf die Geschwindigkeit des Fluides geschlossen werden kann.

$$p_{kin} = \frac{1}{2} \cdot \rho \cdot v^2$$

$$\dot{p}_{kin} = \rho \cdot v$$

Wobei die Dichte aber mit zunehmender Höhe abnimmt. Die Luft kann hier als ideales Gas betrachtet werden und mit der Gasbeziehung errechnet werden.

$$\rho = \frac{p(h)}{R_{Luft} \cdot T}$$

Wobei für den Druck in Abhängigkeit der Höhe die internationale Höhenformel verwendet werden kann.

$$p(h) = \left[1013.25 \left(\frac{0.0065 \cdot h}{288.15}\right)^{5.255}\right] \cdot 100 \qquad [Pa]$$

Gaskonstante trockener Luft $R_{Luft} = 287.058 \ \frac{J}{Kg \cdot K}$

Temperatur in Kelvin: $T = 273.15 \ t(°C)$

Werte der Standardatmosphäre

$$p_0 = 1013.25 \ hPa$$
$$\rho_0 = 1.225 \ \frac{kg}{m^3}$$
$$T_0 = 15 \ °C$$

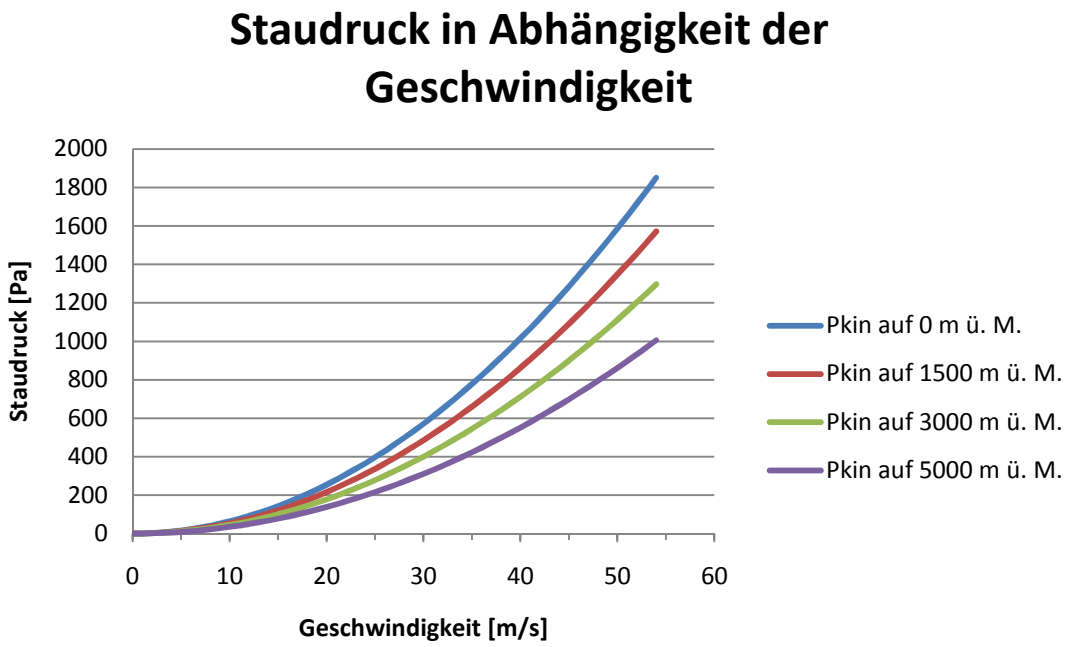

Abb. 23 Staudruck in Abhängigkeit der Geschwindigkeit

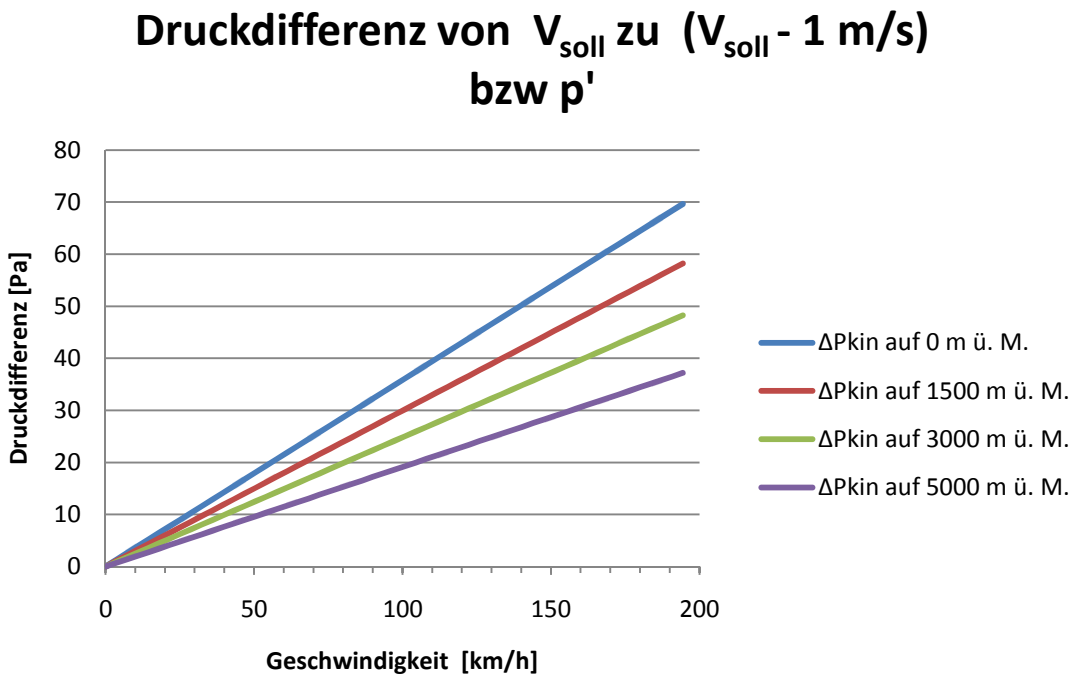

Abb. 24 Zu messende Druckdifferenz, welche einem Unterschied von 1 m/s entspricht, in Abhängigkeit der Geschwindigkeit

In den beiden Grafiken sind die Zusammenhänge von Geschwindigkeit, Flughöhe (Luftdichte) und Staudruck aufgetragen. Grundsätzlich nimmt die Dichte der Luft mit zunehmender Flughöhe ab, was einen geringeren Staudruck bewirkt. Das würde bedeuten, dass für das Bestimmen der Geschwindigkeit bezüglich der Luft eine Kennlinie hinterlegt werden müsste. Das Mass aller Dinge in der Fliegerei ist aber die IAS (Indicated Air Speed) die über den Staudruck gemessen wird. Die so

gemessene Geschwindigkeit ist abhängig von Temperatur und Flughöhe und entspricht nur bei Standardatmosphäre (15 °C und 1013,25 hPa) der Geschwindigkeit gegenüber der Luft (TAS = True Airspeed). Die meisten aerodynamischen Eigenschaften sind an die IAS gekoppelt und bestimmen die Flugeigenschaften. Bei gleicher IAS resultiert in der Höhe eine höhere TAS (True Air Speed).

4.3 Anbindung an das System

Um die Auflösung des Sensors auch verwerten zu können, muss der AD-Wandler des Autopiloten auch genügend genau sein oder die Übermittlung der Messwerte wird über eine Schnittstellen wie I²C oder SPI vorgenommen.

Grundsätzlich sollte aber davon ausgegangen werden, dass die Sensorik austauschbar sein soll. Hierfür würde sich der Datenaustausch über Analogsignale anbieten. Ausserdem könnten diese Werte über das Paparazzi-GCS sehr gut parametriert und justiert werden. Voraussetzung hierfür ist aber die Auflösung des ADC-Eingangs am Mikrokontroller (Paparazzi-Autopilot).

Die AD-Wandler des auf dem Autopiloten-Board verbauten Mikrokontrollers (Arm7, LPC2148) lösen die Eingangswerte mit 10 Bit auf. Dies entspricht:

$$2^{10} = 1024 \text{ Werten}$$

Bei den 5 V Eingängen: $\frac{5\,V}{1024} \cong 0.005\,V = 5\,mV$

Bei den 3.3 V Eingängen: $\frac{3.3\,V}{1024} \cong 0.0033\,V = 3.3\,mV$

Da nur noch ein freier AD-Wandler mit einer maximalen Eingangsspannung von 5 V zur Verfügung stand musste ein AD-Eingang entsprechend angepasst werden. Es wurde also am Eingang ADC_6 der 15 kΩ Widerstand durch einen 1.8 kΩ ersetzt.

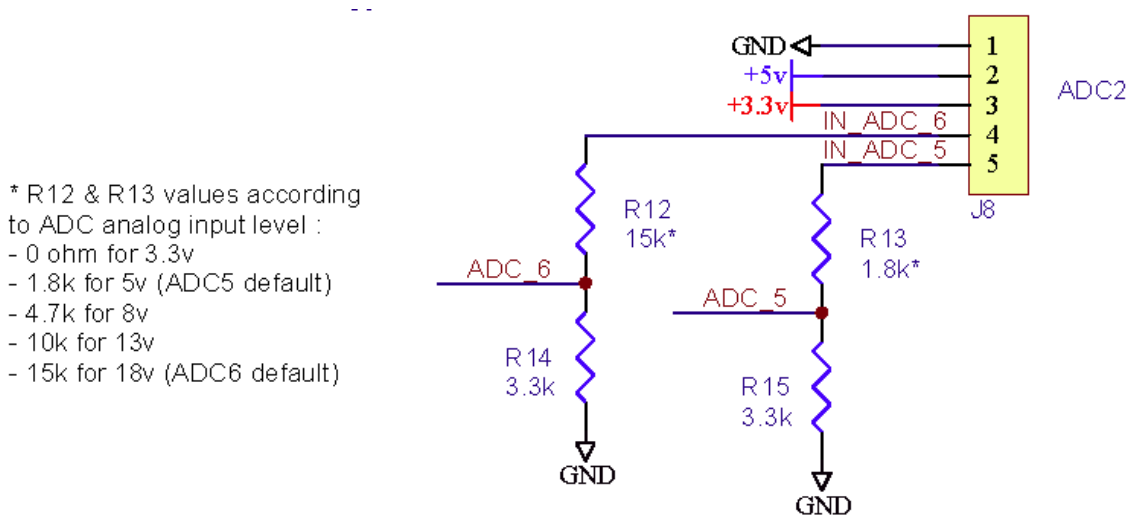

Abb. 25 Ausschnitt aus dem Elektro-Schema des Autopiloten-Board bezüglich der ADC-Eingänge [13]

Wie bei vielen analogen Schaltungen, unterliegt auch der ADC einem Rauschen. Das bedeutet, dass man nicht davon ausgehen sollte, dass der ADC bei konstanter Eingangsspannung auch immer denselben konstanten Wert ausgibt. Ein "Zittern" der niederwertigsten 2 Bits ist durchaus nicht ungewöhnlich. Besonders hervorgehoben werden soll an dieser Stelle nochmals die Qualität der Refe-

renzspannung. Diese Qualität geht in erheblichem Maße in die Qualität der Wandelergebnisse ein [14]. Die Analyse der Messwerte für ein konstantes Messsignal zeigen ein leichtes Rauschen, das in einem Rahmen von ca. 10 mV variiert. Dieses Rauschen dürfte aber für die Regelung irrelevant sein, da Messunschärfen viel stärker ins Gewicht fallen dürften.

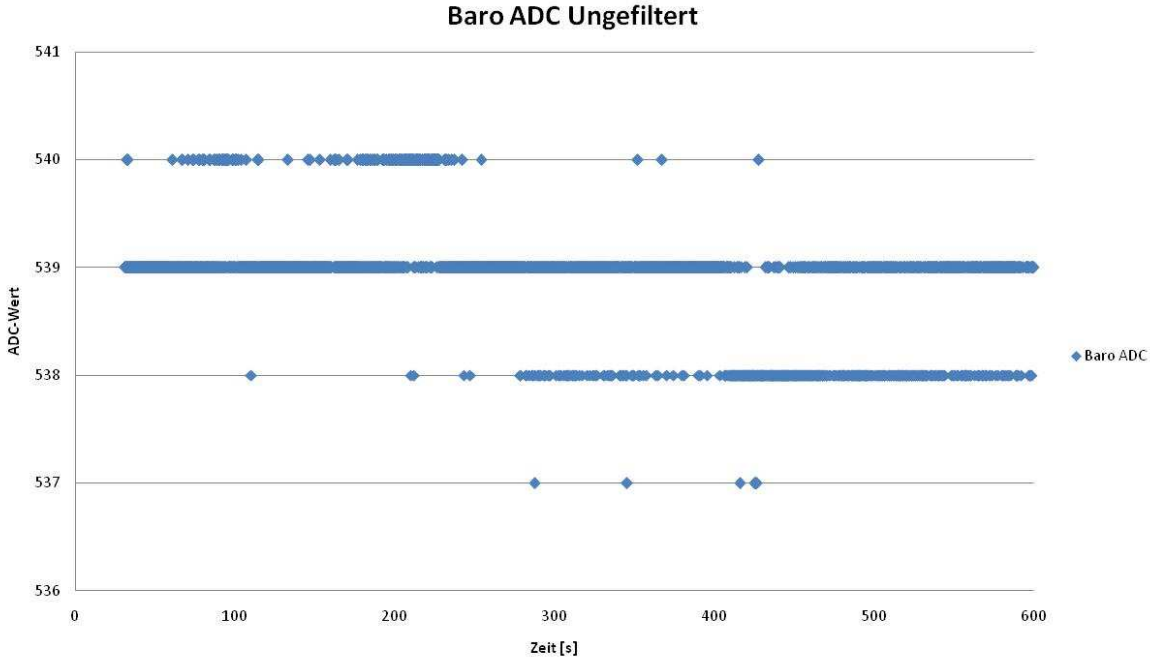

Abb. 26 Zittern des digitalisierten Analog-Signal

4.4 Wahl der Sensoren

Die Sensoren sollten die unter Abschnitt 4.1 aufgeführten Toleranzen einhalten, möglichst kompakt und leicht sein, aber auch preislich sollten sie sich in einem vertretbaren Rahmen befinden. Nachforschungen haben ergeben, dass der Databoom [15] bereits über einen Barometer und einen Differenzdrucksensor verfügt, der die genannten Anforderungen erfüllt. Es macht darum Sinn, die beiden Sensoren gemeinsam zu nutzen.

Es handelt sich um zwei Sensoren der Firma AMSYS. Einen Barometer sowie um einen Differentialdrucksensor. Das Datenblatt [16] der beiden Sensoren befindet sich im Anhang.

4.4.1 Barometer AMSYS AMS 4711 – 1200 B

Die Genauigkeit der Höhenmessung über GPS ist abhängig von der Position der aktuellen Satelliten. Wenn die erreichbaren Satelliten nahe am Horizont sind, steigt die Ungenauigkeit der Höhenmessung. Ein Barometer misst den Luftdruck aus dem die Höhe errechnet werden kann. Durch ändernde Luftdrücke ist diese Messung aber nicht absolut und muss abgeglichen werden.

Die Höhenmessung ist eine Absolutdruckmessung die gegen einen Referenzdruck gemessen wird. Im Normalfall ist dieser das Vakuum oder ein vernachlässigbar kleiner Überdruck, der für die Messung irrelevant wird. Die Sensoren arbeiten mit einem piezoresistiven Effekt der durch das durchbiegen einer Membrane hervorgerufen wird.

Parameter	Wert	
Druckbereich	700…1200 mBar	Bis ca. 3000 m.ü.M
Ausgangsignal	0…..5 V	
Gesamtfehler (-25…85°C) Typisch	± 0.7 %FSO	Siehe Abb. 27
Gesamtfehler (-25…85°C) maximal	± 1.5 %FSO	Siehe Abb. 27
Auflösung	0.05 %FSO	Siehe Abb. 27
Versorgungsspannung	8….24 V ideal sind 24V	
Eigenstromaufnahme	5mA	
Gewicht	20 g	

Tabelle 2: Daten des Barometers

Der Messbereich des Sensors entspricht nicht der geforderten Maximalhöhe von 5000 m.ü.M, was einem Druck von 540 mBar entsprechen würde. Da der Sensor aber schon verbaut ist, und es Sinn machen würde die vorhandenen Mittel zu nutzen, wurde entschieden diesen Sensor zu nutzen.

Die Auflösung des Sensors ist für die Anforderungen genügend genau in jeder Höhe respektive bei jedem geforderten Atmosphärendruck. Um den etwas grossen Fehler in der absoluten Höhe zu eliminieren kann er möglicherweise sporadisch mit der GPS-Höhe abgeglichen werden.

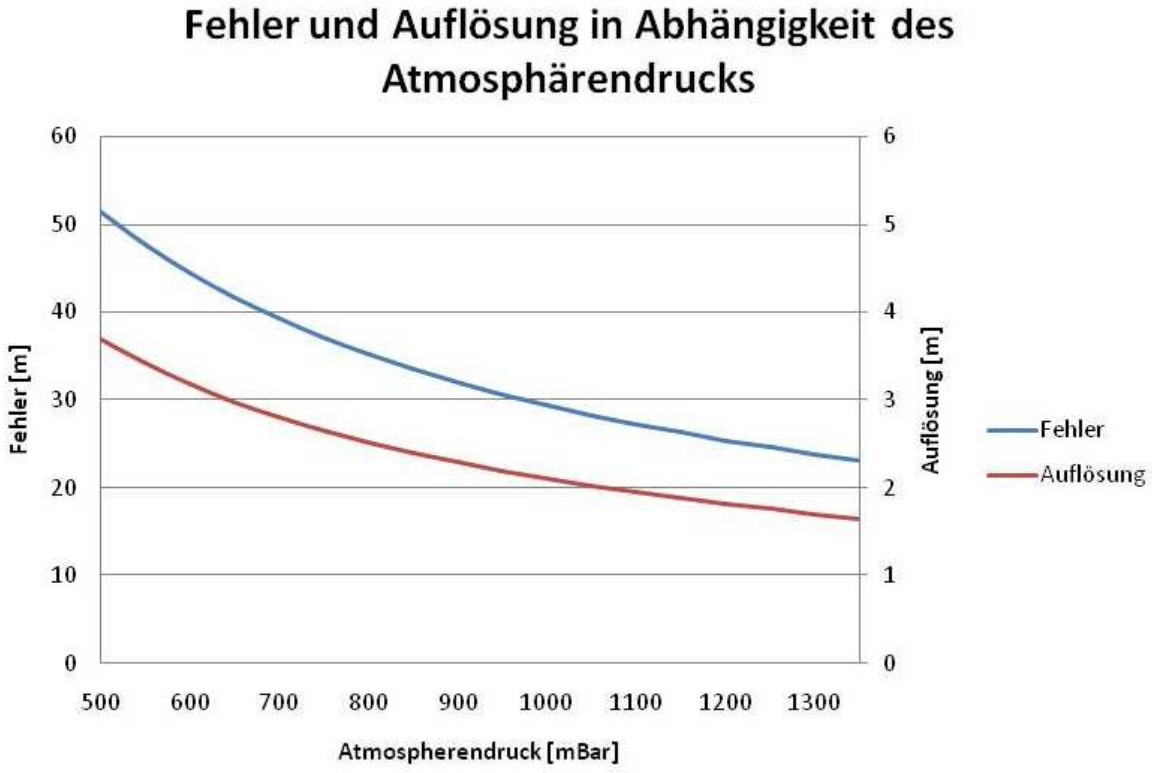

Abb. 27 Fehler und Auflösung in Abhängigkeit des Atmosphärendrucks

4.4.2 Differenzdrucksensor AMSYS AMS 4711 – 0020D

Der Differenzdrucksensor ist ein Drucksensor, der die Differenz zweier Absolutdrücke, den sogenannten Differenzdruck, misst. Der Differenzdrucksensor kann aus zwei Messkammern bestehen, die durch eine Membran hermetisch voneinander getrennt sind. Die Auslenkung der Membran ist dann ein Mass für die Größe des Differenzdruckes[17].

Parameter	Wert	
Druckbereich	0…20 mBar	Ca. 0 bis 55 m/s (200 km/h)
Ausgangsignal	0…..5 V	
Gesamtfehler (-25…85℃) Typisch	± 1 %FSO	
Gesamtfehler (-25…85℃) maximal	± 2 %FSO	
Auflösung	0.05 %FSO	
Versorgungsspannung	8….24 V ideal sind 24V	
Eigenstromaufnahme	5mA	
Gewicht	20 g	

Tabelle 3: Daten des Differenzdrucksensor

Wie der Name schon sagt, misst der Differenzdrucksensor die Druckdifferenz zwischen dem sogenannten Staudruck und dem Umgebungsdruck. Dieser Druck kann mit der folgenden Beziehung nach Bernoulli in die Geschwindigkeit umgerechnet werden.

$$v = \sqrt{\frac{2 \cdot p_{kin}}{\rho_{luft}}}$$

Um die Zusammenhänge zwischen den Messwerten, dem Staudruck und der Fluggeschwindigkeit darzustellen wurde die folgende Grafik gemacht.

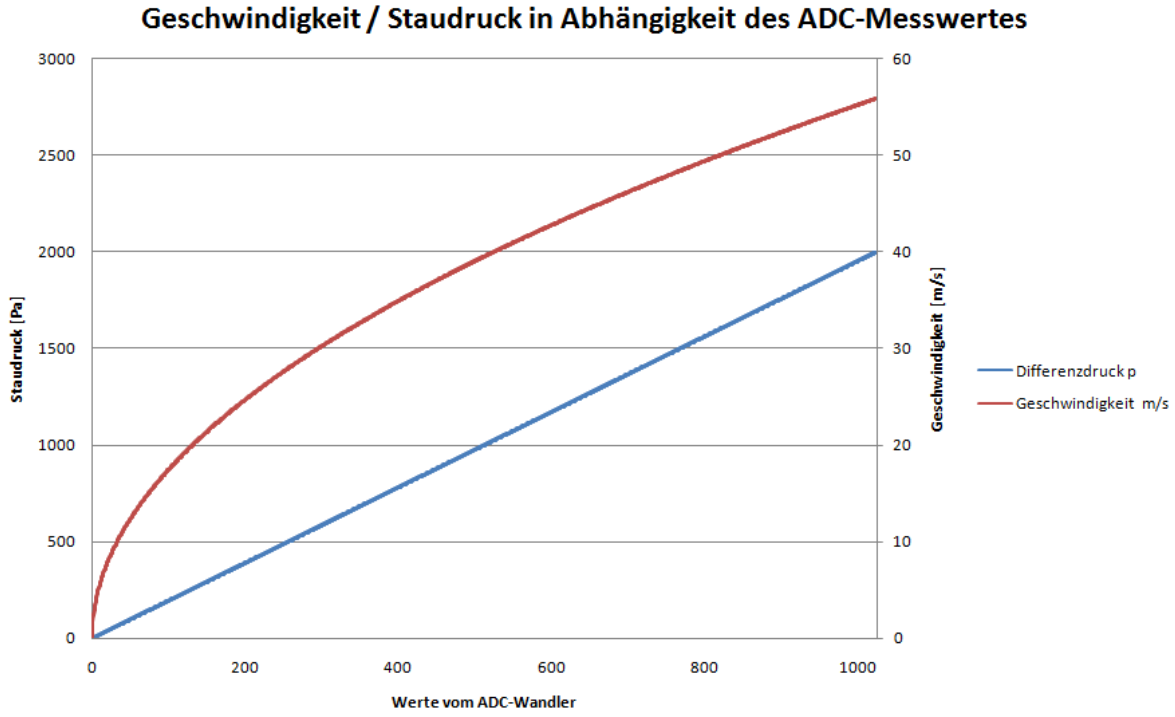

Abb. 28 Geschwindigkeit und Staudruck in Abhängigkeit des ADC-Messwertes

Aus dieser Abbildung ist gut ersichtlich, dass die Messproblematik bei tiefen Geschwindigkeiten markant ist. Der Druck steigt linear mit dem ADC-Messwert an. Die Geschwindigkeit wächst aber in einer Wurzelfunktion an. Dies führt dazu, dass die Messwerte bei tieferen Geschwindigkeiten ungenauer sind als bei hohen. Das Verhalten dieser Problematik muss genauer untersucht werden.

Für die verlangte Geschwindigkeitstoleranz von ± 1 m/s muss herausgefunden werden, wie genau die Auflösung der Sensoren und des ADC-Wandlers sein muss um die Geschwindigkeit genügend genau messen zu können.

$$Auflösung\ Sensor \qquad \Delta p = 0.05\% \cdot 2000 = 1\ Pa.$$
$$Auflösung\ ADC - Wandler \qquad \Delta p = \frac{1}{1024} \cdot 2000\ Pa = 1.96\ Pa$$

Wir haben also eine effektive Auflösung von ca. 2 Pa, die für die Regelung verwendet werden kann. Um ein Bild über die Toleranz und die dazugehörenden Drücke zu erhalten, wurde die Beziehung zwischen Geschwindigkeit und Druckdifferenz bei 1 m/s Geschwindigkeitsverlangsamung dargestellt. Daraus wird ersichtlich, dass bis ca. 5 m/s die Geschwindigkeit nicht genügend genau gemessen werden kann, um effizient zu regeln.

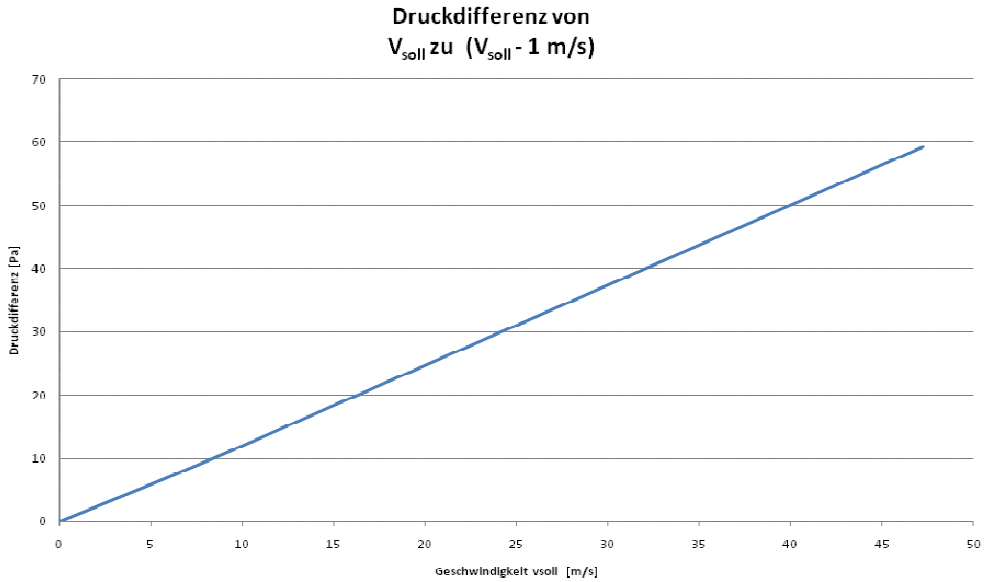

Abb. 29 Druckdifferenz, die bei einer Regelabweichung von ±1 m/s auftritt

Um diese Behauptungen zu verifizieren, wurde die Auflösung der Geschwindigkeit, die über die gemessenen Drücke errechnet werden kann, in Abhängigkeit zur Fluggeschwindigkeit aufgetragen. Darin wird aber ersichtlich, dass die Auflösung von 2^{10} Bit des AD-Wandlers das Signal des Sensors nicht vollständig auflösen kann. Das Problem ist aber nicht gravierend, da die Abweichung nur in einem Bereich bis ca. 5 m/s markant ist. Für eine Flugregelung dürfte es aber immer noch genügend genau sein, da für den Cruisespeed der Drohne von 70 km/h (20 m/s) sich schon eine Auflösung der Geschwindigkeit von 0.076 m/s ergibt.

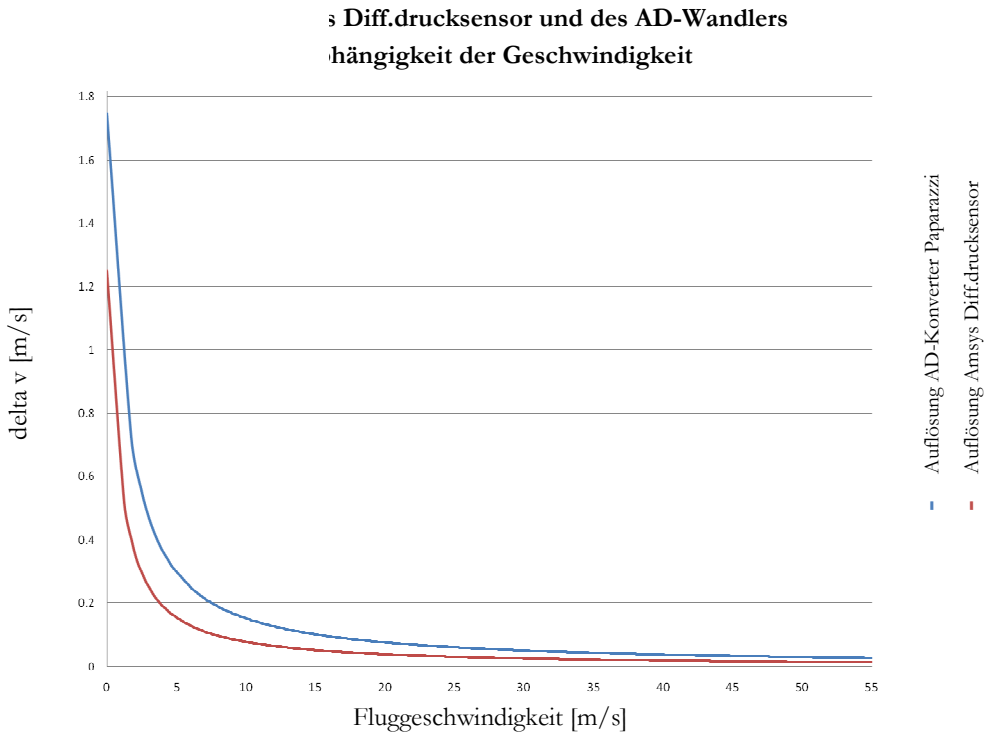

Abb. 30 messbare Geschwindigkeitsdifferenz in Abhängigkeit zur Fluggeschwindigkeit

4.5 Positionierung der Messstellen

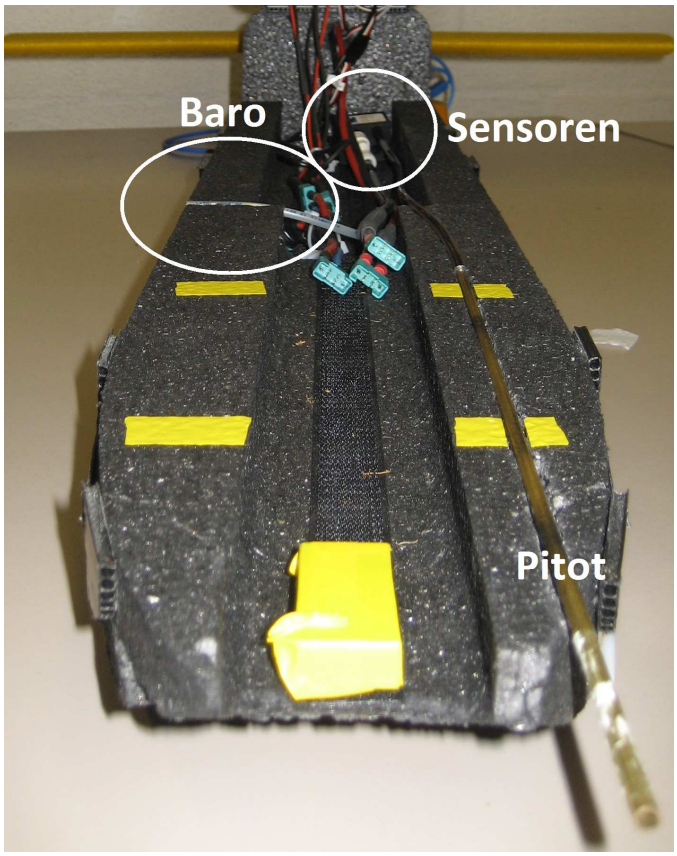

Abb. 31 Anordnung der Messinstrumente im Testflugzeug

Da in dem Testflugzeug MAJA genügend Platz vorhanden ist, konnten der Barometer und das Pitot-Rohr einfach nach aussen gezogen werden. Uns ist bewusst, dass dieses Setup nicht einwandfreie Messergebnisse liefert, da aber der Aufwand für präzise Messergebnisse sehr gross ist und in der UMARS-Drohne ein Messbaum zum Einsatz kommt, entschieden wir uns, den Aufwand für die Messungen mit dem Tesflugzeug auf einem vertretbaren Niveau zu halten. Die Flugversuche haben auch gezeigt, dass die Abweichungen, wenn vorhanden, sehr klein sein müssen. Dies ist zum Beispiel in der Abb. 32 gut sichtbar, denn die Differenz, zwischen dem GPS-Speed zum gemessenen Air-Speed, bleibt über die Beschleunigung beinahe konstant, so dass die Differenz der Windgeschwindigkeit zugesprochen werden kann.

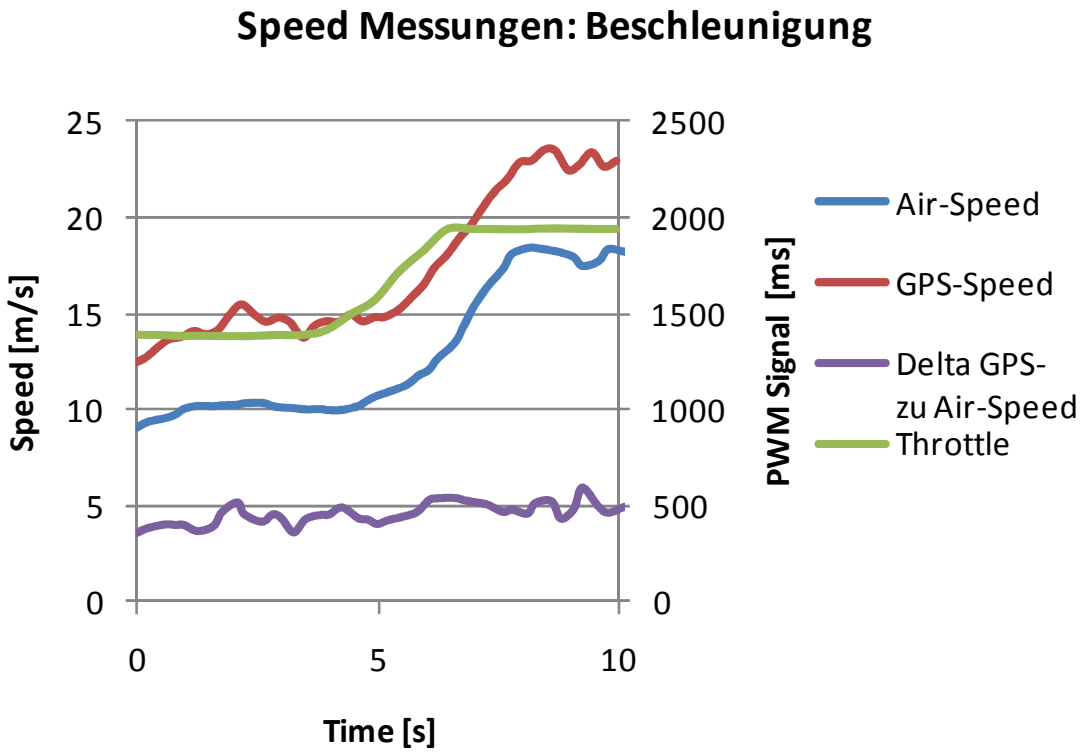

Abb. 32 Vergleich der Geschwindigkeit beim Beschleunigen

Eine statische Messung zeigt, dass der Sog von Propeller die statische Druckabnahme, also die Messung des Barometers und des Pitot-Rohrs, verfälscht. Durch die Druckabnahme an der Messstelle, werden eine zu grosse Höhe und eine zu langsame Geschwindigkeit gemessen. Der genaue Einfluss auf die Regelparameter wurde in dieser Arbeit nicht untersucht, da diese Verfälschung in dem UMARS durch den Einsatz eines optimal positionierten Pitot-Rohrs nicht mehr auftreten wird.

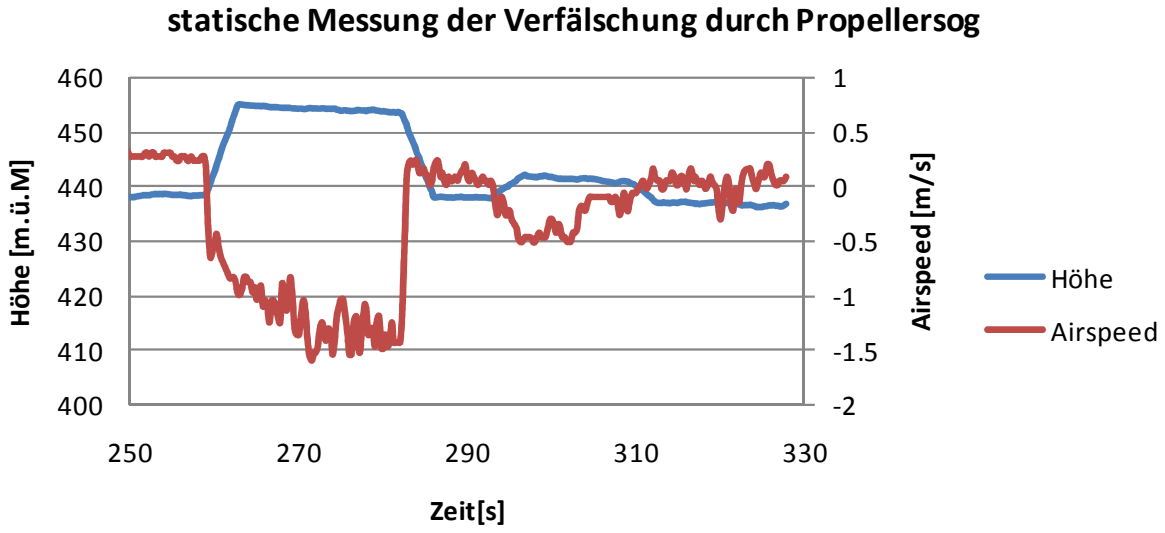

Abb. 33 statische Messung der Verfälschung durch Propellersog

4.6 Regelung

Die Sensorwerte müssen nun im Sourcecode verwertet werden. Aufgrund der mangelnden Erfahrung und der Verschiedenartigkeit der möglichen Regelungen entschieden wir uns, verschiedene Reglungen zu erstellen, die während des Fluges umschaltbar sind, um so einfach und schnell Direktvergleiche machen zu können. So sind, zu den im Autopilotencode vorhandenen Regelungen, zwei zusätzliche entstanden. Weiter wurden die Regelparameter so voneinander getrennt, dass alle regelungsspezifischen Werte unabhängig sind, um schnell zwischen den Regelsystemen umschalten zu können.

4.6.1 Standardregelung

Die Standardregelung des Autopiloten regelt die Fluggeschwindigkeit nicht. Man gibt nur einen bestimmten Sollwert der Motorenleistung (% Vollleistung) vor, um den dann geregelt wird. Abweichungen vom Sollwert entstehen bei Steig- oder Sinkflügen, wo der Autopilot das Gas zurück- oder Hochfährt. Diese Regelung ist sehr robust und bei unseren Flügen haben wir stets gute Resultate erzielt. Die Luftgeschwindigkeit wird aber nicht geregelt.

Die Differenz zwischen Soll- und Ist-Höhe, die zusätzlich mit verschiedenen Faktoren etwas skaliert werden kann, ist eine der Haupteingangsgrössen für die Standardregelung. Daraus wird zum einen der Pitch-Setpoint geregelt, aber auch die Motorenleistung (*throttle*). Der Pitch-Setpoint (*nav_pitch*) wird in der Lageregelung (*horizontal_control*) die Soll-Eingangsgrösse mit der der Winkel des Höhenruders geregelt wird. Die Motorenleistung wird direkt nach einen PID-Regler und einem Limiter als Stellsignal auf den Motor geschickt.

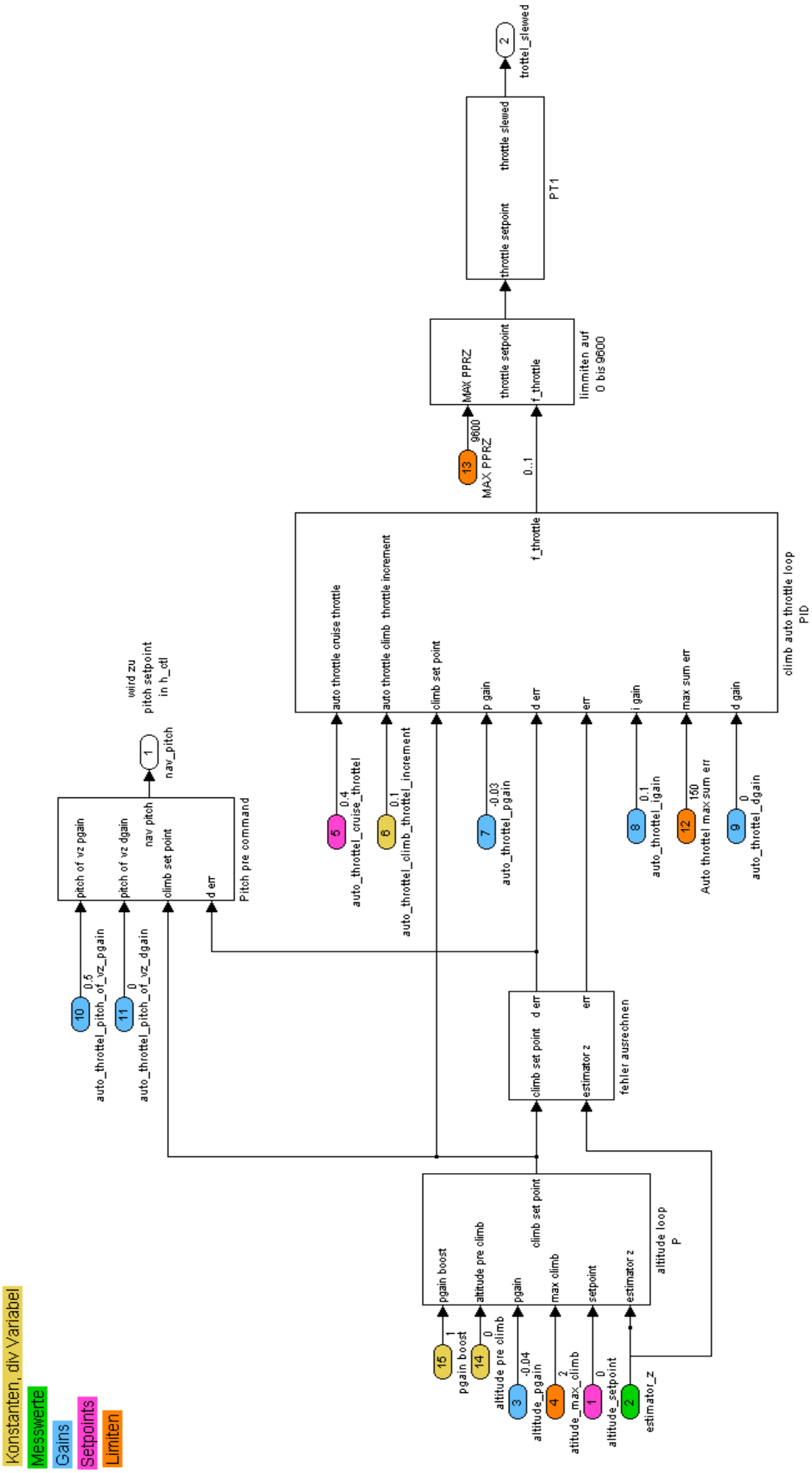

Abb. 34 Simulink Modell der Standard Höhen- und Geschwindigkeits-Regelung

4.6.2 Vassilis-Airspeed

Diese Regelung ist relativ neu im Paparazzi-Sourcecode enthalten und wurde von Vassilis Varveropoulos (Paparazzi-User) geschrieben, über die Paparazzi-Community dem Code hinzugefügt und ist jetzt Bestandteil vom Original Source Code. Das Projekt von Vassilis [18] sieht einen EagleTree [19] Sensor für die Messung des Airspeed und ein Barometer vom gleichen Hersteller für die Messung der Flughöhe vor. Dieses Konzept regelt mit dem Ansatz, dass mit Motorenleistung Geschwindigkeit erzeugt wird und mit dem Höhenruder (Pitch) die Höhe verändert wird. Um abzusichern, dass bei starken Winden der Groundspeed immer positiv ist, wurde der Airspeed-Sollwert so abgesichert, dass wenn er unter einen bestimmten Wert fällt, dieser sich proportional erhöhen würde. Dies ist von Bedeutung für die Berechnung des Kurses. Falls die Bodengeschwindigkeit negativ werden würde, könnte der Kurs nicht mehr richtig berechnet werden, da das Flugzeug auf einmal rückwärts fliegen würde.

Der Regelkreis für die Absicherung einer positiven Bodengeschwindigkeit ist ein PI-Glied, das den Fehler zwischen der Soll-Bodengeschwindigkeit (*groundspeed_setpoint*) und dem GPS-Bodengeschwindigkeit (*estimator_hspeed_mod*) verrechnet. Der Airspeed-Regelkreis basiert auch auf einem PI-Glied. Als Messgrösse dient die gemessene Luftgeschwindigkeit (*astimator_airspeed*), welche entweder mit der Soll-Luftgeschwindigkeit (*airspeed_setpoint*), oder mit der Stellgrösse vom Groundspeed Regelkreis verglichen wird. Je nachdem, ob die gemessene Bodengeschwindigkeit unter den minimalen Wert gesunken ist. Die Stellung des Höhenruders ist wieder abhängig von Sollhöhe und Ist-Höhe, die proportional verstärkt werden kann, um so die Höhe etwas „schärfer“ zu regeln. Der effektive Pitch wird dann in einem PI-Regler geregelt und danach als Sollwert in den Horizontalcontrol für die Lage der Drohne weitergeführt.

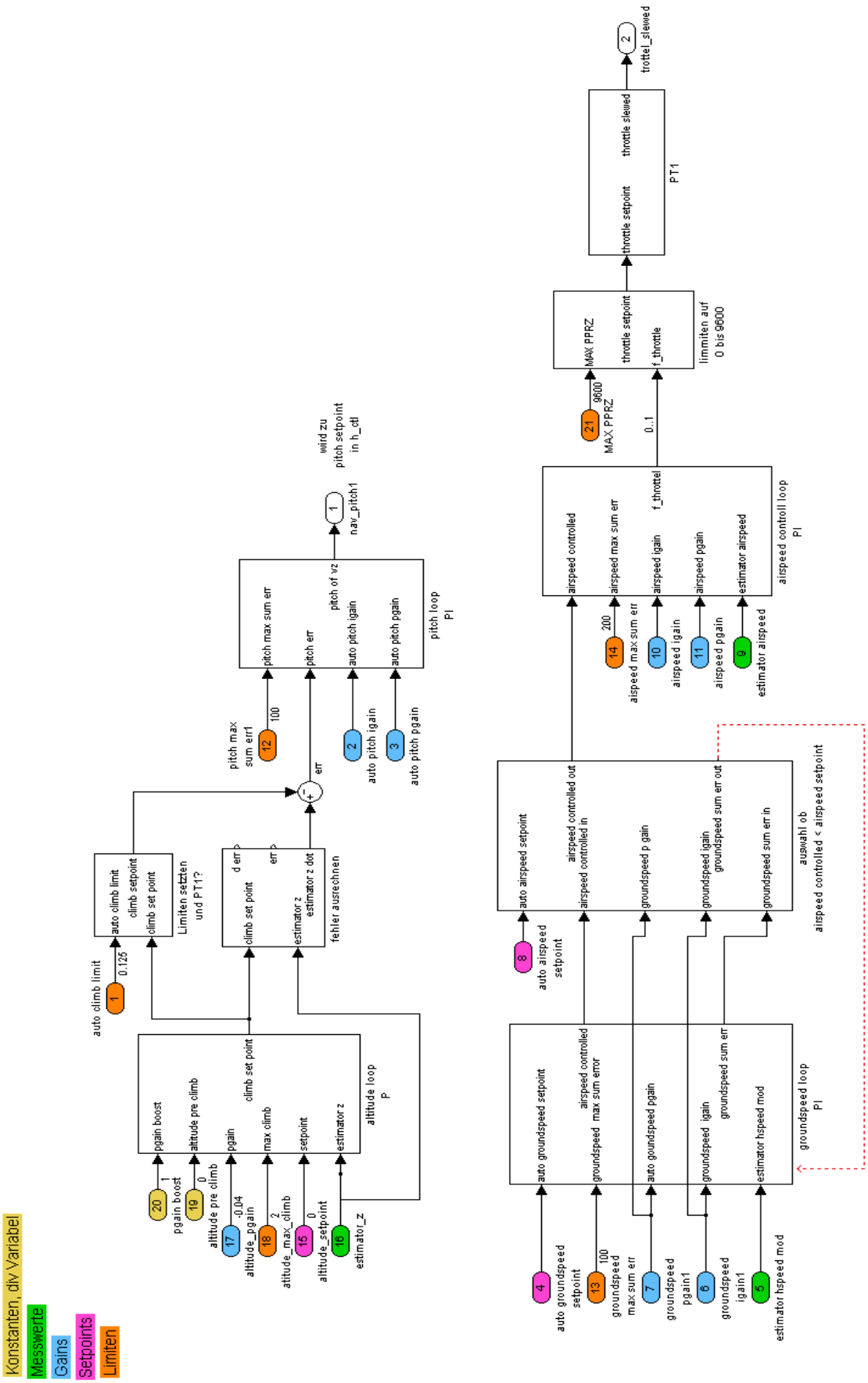

Abb. 35 Simulink Modell der Vassilis Höhen- und Geschwindigkeits-Regelung

4.6.3 Airspeed One

Die Airspeed-One-Regelung ist ähnlich zur Regelung von Vassilis. Diese Regelung wurde unabhängig zur Vassilis-Regelung erstellt, ist aber bei genauerer Betrachtung beinahe gleich aufgebaut. Pitch- und Airspeed-Loop werden mit Hilfe eines PI Regler realisiert. Auch die Abweichung der Soll-Höhe kann proportional verstärkt werden. Eine Sicherheitsschranke um zu verhindern, dass die Bodengeschwindigkeit negativ wird, ist auch vorhanden.

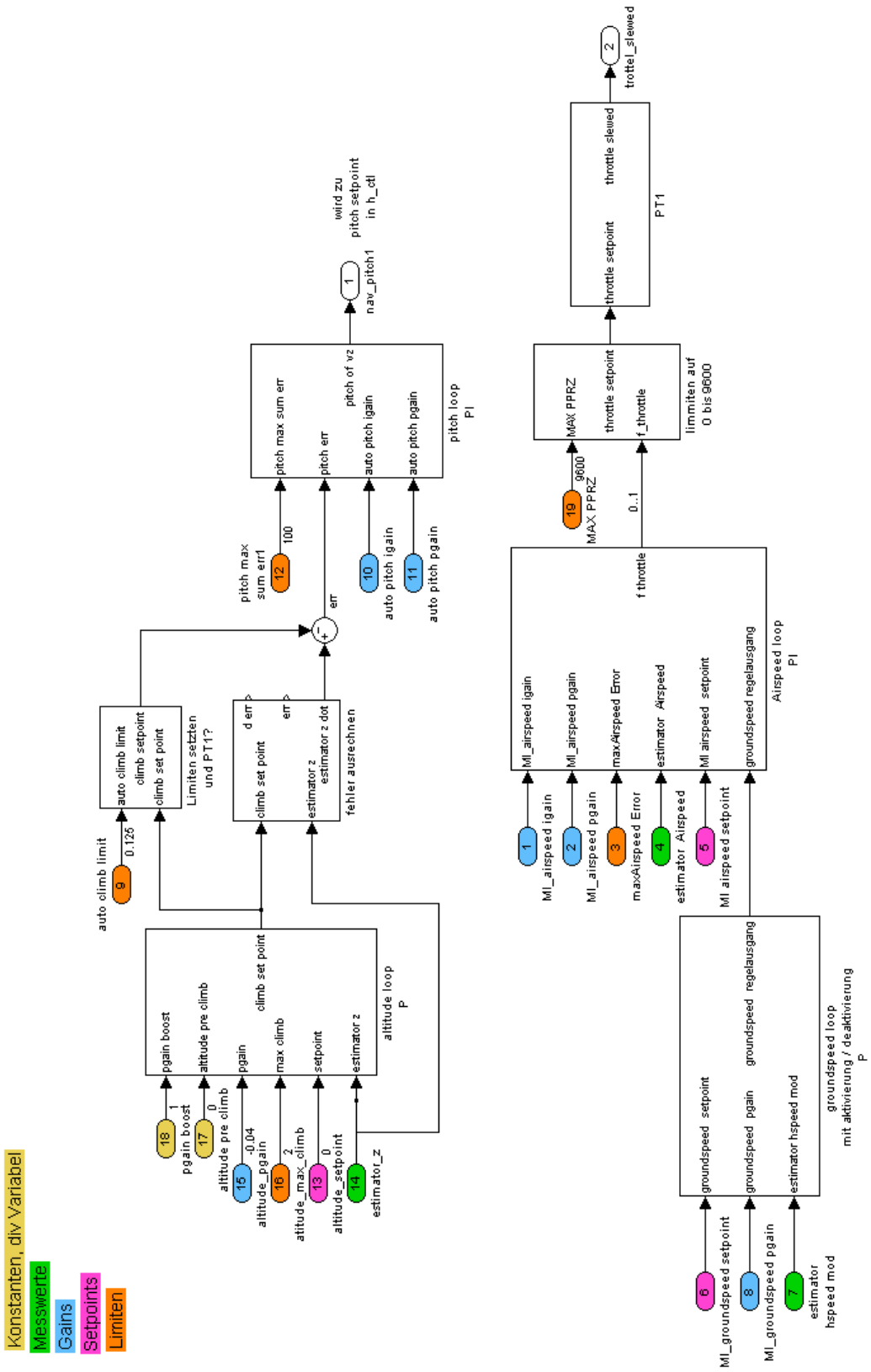

Abb. 36 Simulink Modell der Airspeed One Regelung für die Höhe und die Geschwindigkeit

4.6.4 Airspeed Two

Die Airspeed-Two-Regelung ist eine im Ansatz grundsätzlich andersartige Regelung für Höhe und Geschwindigkeit. Im Gegensatz zu den beiden letzten Geschwindigkeitsregelungen arbeitet sie nach dem Prinzip, dass für eine Geschwindigkeitsänderung die Höhe und für eine Höhenänderung die Motorenleistung verändert werden muss. Wenn das Flugzeug also zu tief ist, soll es die Motorenleistung erhöhen bis es „zu schnell“ wird und dann Pitch geben und um an Höhe zugewinnen. Diese Regelung hat verschiedene Vorteile gegenüber den anderen Systemen. Geschwindigkeit bzw. die Menge der umströmenden Luft, die um die Tragflächen fliesst ist eine essentielle Grösse für jedes Flugzeug. Bei einem Motorenausfall fliegt das Flugzeug trotzdem mit der Sollgeschwindigkeit weiter und kann optimal weitergeflogen werden.

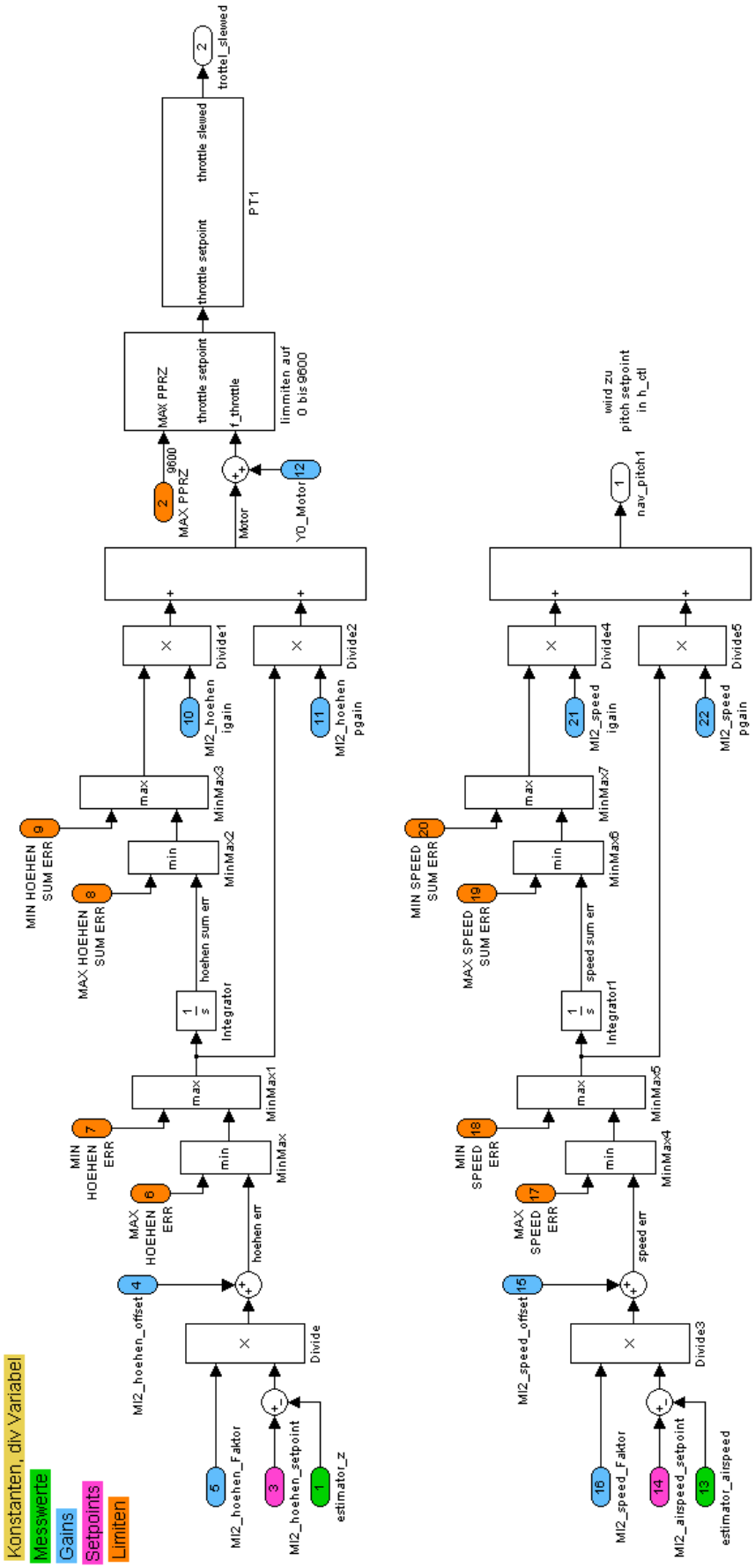

Abb. 37 Simulink Modell der Airspeed Two Regelung für die Höhe und die Geschwindigkeit

4.6.5 Vergleich der verschiedenen Regelungen

Für den Vergleich der verschiedenen Regelungen wurde ein Oval geflogen. Während dieses Fluges herrschte mässiger Seitenwind.

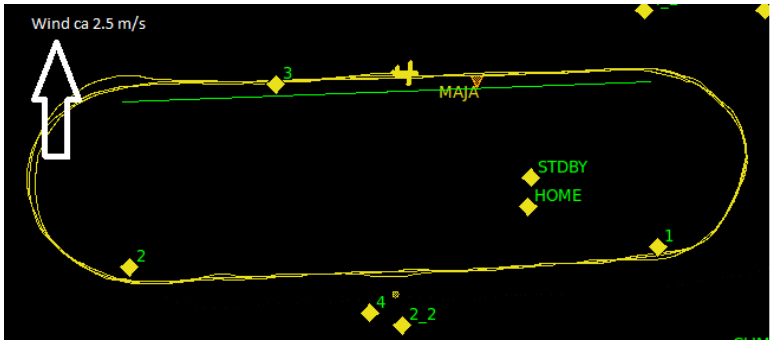

Abb. 38 Ovales Flugpattern zum Vergleich der verschiedenen Fluggeschwindigkeits-Regelung

Wir verglichen die zwei verschiedenen Systeme miteinander um die Güte der Regelung zu untersuchen. Die Regelparameter konnten aus zeitlichen Gründen nicht perfekt eingestellt werden, da das Testflugzeug (MAJA von borjet) eine ungünstige Eigenschaft bei Erhöhung der Motorenleistung zeigt. Sobald der Motor die Drehzahl erhöht, neigt sich das Flugzeug nach unten und beginnt zu sinken. Dies ergibt eine erhöhte Luftgeschwindigkeit und einen Fehler in der Soll-Höhe. Welche der Regler auszugleichen versucht, indem er Motorenleistung zurücknimmt und mit Pitch das Flugzeug an Höhe gewinnen lässt. Da diese Eigenschaft aus unseren Beobachtungen nicht linear ist, beginnt sich Geschwindigkeit und Höhe sehr schnell aufzuschaukeln. Es wäre interessant wie schwierig das Einstellen der Regler bei einem Flugzeug ist, welches diese Eigenschaft wenig bis gar nicht besitzt.

Die Regler konnten aber trotzdem so eingestellt werden, dass sie die Anforderungen fast erfüllen. Bei weiteren Justierungen der Parameter sind wir zuversichtlich, dass die Toleranz von ± 1 m/s auch mit der MAJA eingehalten werden kann.

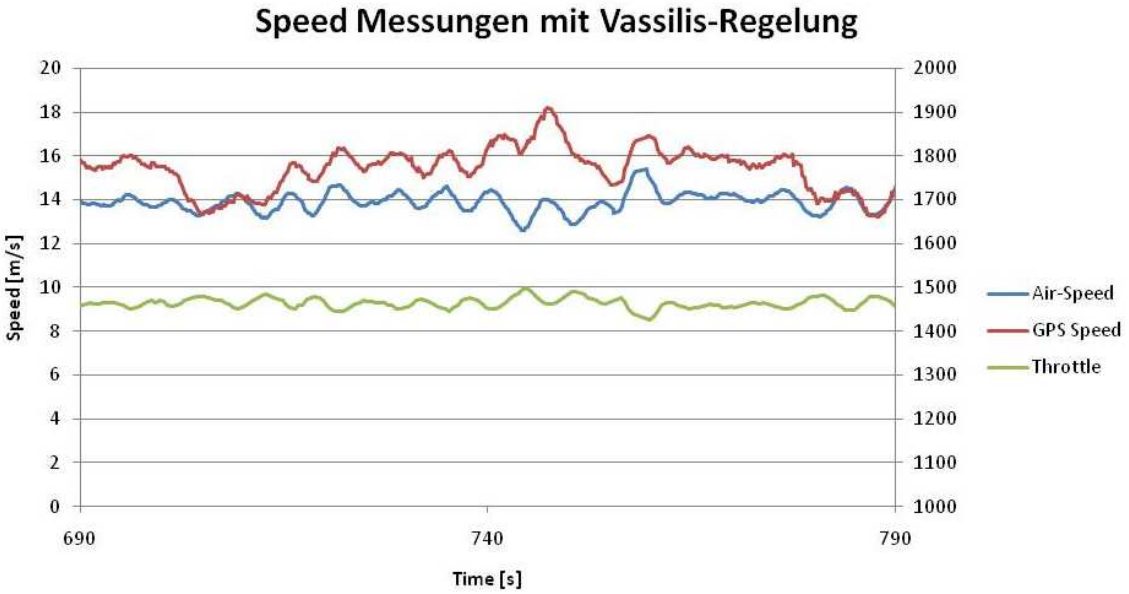

Abb. 39 Messung der Fluggeschwindigkeit während der Vassilis-Regelung

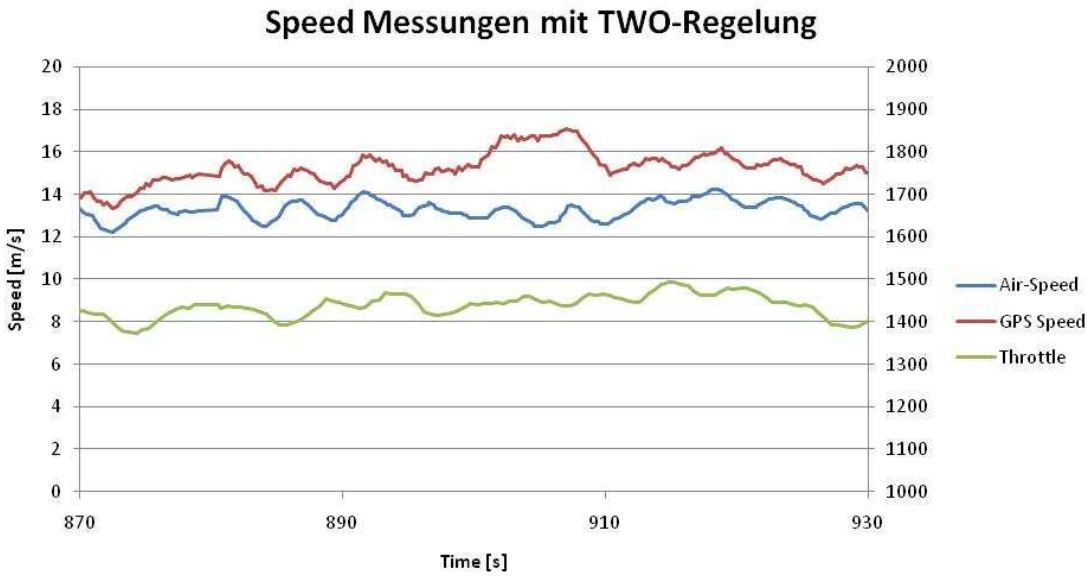

Abb. 40 Messung der Fluggeschwindigkeit während der Airspeed TWO-Regelung

4.6.6 Mögliche Verbesserung der Regelung

Die bisherige Regelung schwingt sehr schnell und es ist noch nicht klar, ob dies auch den Eigenschaften des Flugzeuges zuzuschreiben ist. Auf jeden Fall wird es durch diese begünstigt. Ausserdem ist, wie im letzten Kapitel beschrieben, die Suche nach passenden Regelparametern sehr mühsam und zeitintensiv. Wünschenswert wäre daher eine Lösung, die es ermöglichen würde, über Modelle am Computer, die passenden Regelparameter zu finden. Wir konnten diesen Ansatz aus Zeitgründen nicht mehr zu Ende bringen, wollen aber mit diesem Kapitel eine Diskussionsgrundlage bieten.

Die Idee ist, mit einer Nachbildung der Regelstrecke und Matlab-Simulink einen geeigneten Regler zu entwerfen. Dafür wurden die Daten während einer Beschleunigungsphase aufgezeichnet und in Simulink übernommen. Die Strecke konnte mit einem PT-2 Verhalten ziemlich genau nachgebildet werden.

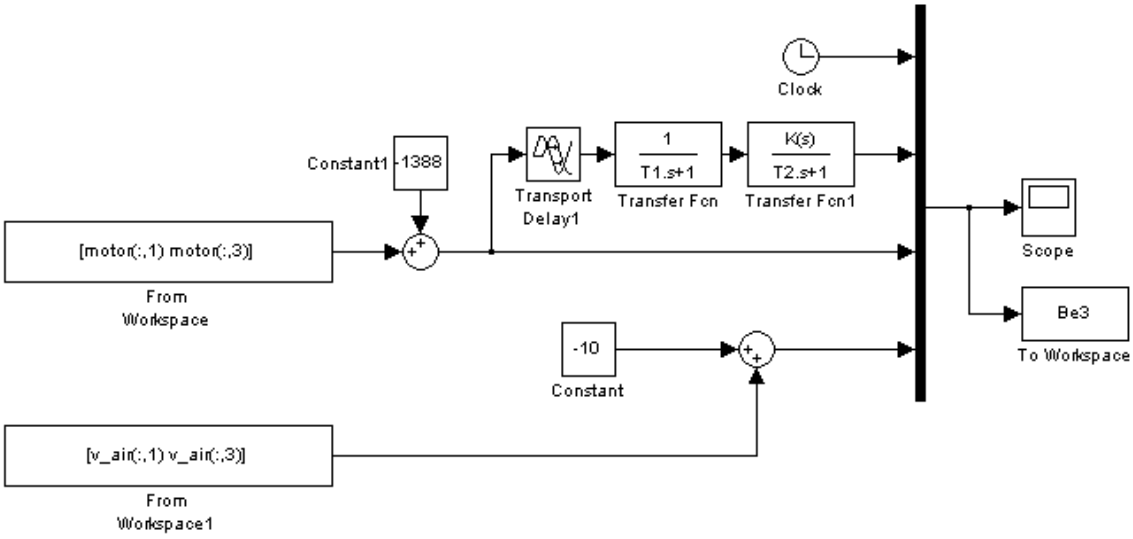

Abb. 41 Simulink Modell für die Nachbildung der Regelstrecke mittels einem PT-2 Verhalten
K = 0.01475 T1 = 0.6 T2 = 0.3

Die Nachbildung der Strecke in diesem Plot beinhaltet auch das Signal der Fernsteuerung (Soll-Wert) und die effektiv gemessene Geschwindigkeit bezogen zur Luft.

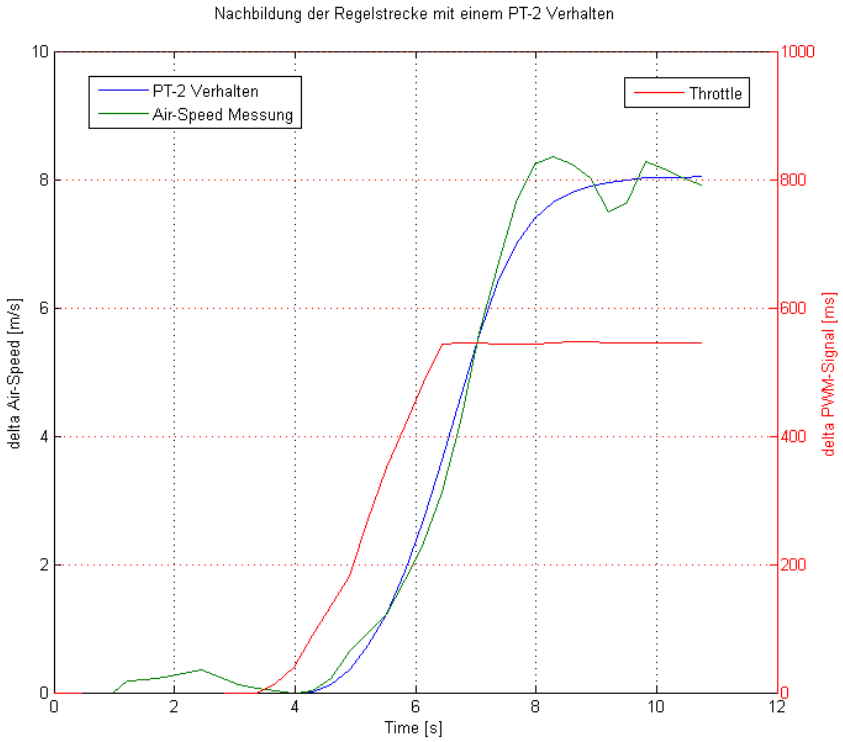

Abb. 42 Nachbildung der Regelstrecke mit einem PT-2 Verhalten

Der Ansatz zur Nachbildung der Strecke eröffnet ein grosses Feld an möglichen Reglereinstellverfahren. Zum Beispiel könnten die Strecke und Regler in Matlab-SISO-Tool übernommen werden und dort die Regler schnell und einfach parametriert werden. So müsste nicht mehr, wie bis anhin, durch aufwendigen Flugversuchen die Parameter ausprobiert werden. Weiter wäre es denkbar in die Regler der Software eine Lookuptable zu integrieren, die je nach Situation andere Regelparameter verwendet, um so ein noch günstigeres Verhalten der Regelung zu erhalten und mögliches Aufschwingen des Flugzeuges zu verhindern.

Die Daten, die für ein solches Einstellverfahren verwendet würden, müssen aber sehr präzise gemessen werden. Zum Beispiel sollte die Flughöhe nicht schwanken, was bei unseren Messungen trotz mehreren Versuchen immer etwas der Fall war. Je nach Verhalten der UMARS-Drohne muss auch die Regelung der Flughöhe in die Optimierung der Geschwindigkeitsregler miteinbezogen werden, damit das uns bekannte Verhalten von Absinken bei Leistungszuwachs eliminiert werden kann. Eine computergestützte Optimierung der Regelparameter wäre aus unserer Sicht sehr hilfreich um mühsames und zeitintensives Finden der Regelparameter zu vermeiden.

5 Jetziger Stand der Software/Hardware

Dieser Abschnitt des Berichtes soll eine Übersicht über die Software des Autopiloten geben. Das Ziel dieses Kapitels ist nicht, alle Aspekte und Details aufzugreifen, denn dafür ist die Software zu komplex. Es geht vor allem darum, dem Leser einen Überblick zu verschaffen sowie den Einstieg in die modifizierte Architektur der Software zu ermöglichen.

Zu Beginn dieser Arbeit war das Ziel alle Zusatzkomponenten so einzufügen, dass die Originalsoftware heruntergeladen werden kann (siehe dazu Trickkiste.doc im Anhang [20]) und mit wenigen Schritten die Erweiterungen eingebunden werden können. Deswegen wurde entschieden, dass alle Zusatzfunktionen mit Modulen gemacht werden. Dies ermöglicht einem die Zusatzfunktionen während des Fluges ein- und auszuschalten, die Frequenzen mit denen die Funktionen aufgerufen werden einzustellen.

5.1 Hard- und Software-Aufbau von Paparazzi

5.1.1 Aufbau der Hardware

Im Unterschied zum ersten Setup wurde der Speisungsakku für den Autopiloten entfernt, da wir mit einem starken 11.1 V Akku immer genügend Spannung haben um den Autopiloten zu speisen. Dieser stellt bei einer Mindestspannung von 4 V ab. Ein derart tiefer Wert des Hauptakkus kann fast nicht erreicht werden. Die Spannung kann zudem im GCS überwacht werden kann.

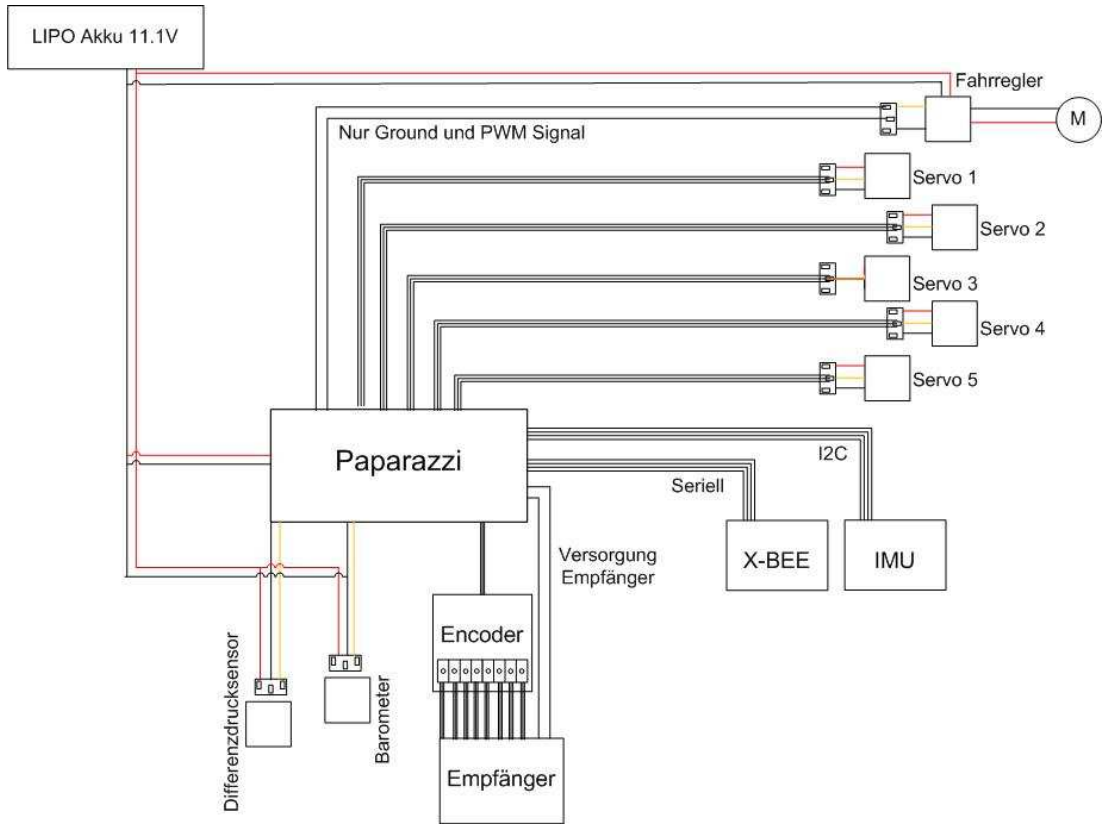

Abb. 43 Signal- und Speisungs-Schema des Autopiloten mit IMU und AMSYS-Sensoren und einem Akkumulator

5.1.2 Überblick über die wichtigsten Hard- und Softwarekomponenten

Index	Beschreibung
1	Roll-Winkel Pitch-Winkel Yaw-Winkel Beschleunigungen in X-,Y- und Z-Richtung
2	Roll-Winkel Pitch-Winkel Yaw-Winkel Beschleunigungen in X-,Y- und Z-Richtung
3	Gesamte GPS Daten
4	Selektierte GPS Daten
5	ADC 5
6	ADC 6

Tabelle 4: Variabeln Verzeichnis 1

Index	Beschreibung
7	Amsys_airspeed Amsys_Baro
8	Course_setpoint Altitude_setpoint
9	Estimator_z Estimator_phi Estimator_theta Estimator_ hspeed_dir Estimator_ hspeed_mod
10	Nav_pitch
11	GPS Daten

Tabelle 5: Variabeln Verzeichnis 2

Um die Funktion und den Aufbau der Regelung zu verstehen ist es wichtig die zentralen Komponenten der Hard- und Software für die Regelung und Navigation zu kennen. Auf dem folgenden Bild sind auf der linken Seite die Sensoren, in der Mitte ist die Software dargestellt und rechts befinden sich die Aktoren, die für die Auslenkung des Flugzeuges verantwortlich sind. Die Darstellung ist stark vereinfacht, stellt aber die wichtigsten Komponenten und Verbindungen für den Datenfluss dar.

In der obigen Tabelle ist eine Übersicht über die wichtigsten Werte und Variablen, die in der folgenden Grafik über die Beziehungen, dargestellt sind.

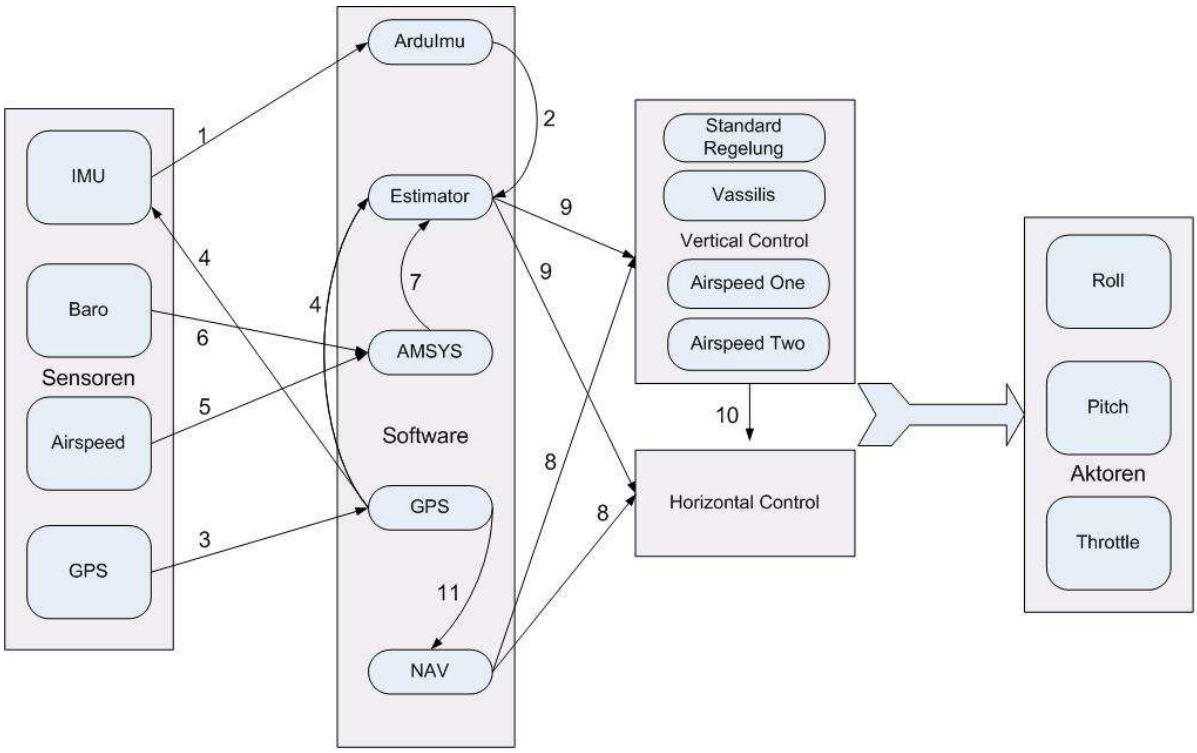

Abb. 44 Schema der wichtigsten Hard- und Softwarekomponente

5.1.3 Die Architektur von Paparazzi

Main_ap ist die Hautroutine die zyklisch durchlaufen wird. Sie ruft die wichtigsten Funktionen mit einer Frequenz von 60 Hz auf. In ihr kann auch eingestellt werden, wie schnell eine andere Funktion aufgerufen werden soll. Dies kann zum Beispiel hilfreich sein für die Einstellung der Abtastrate eines Sensors oder Ähnlichem.

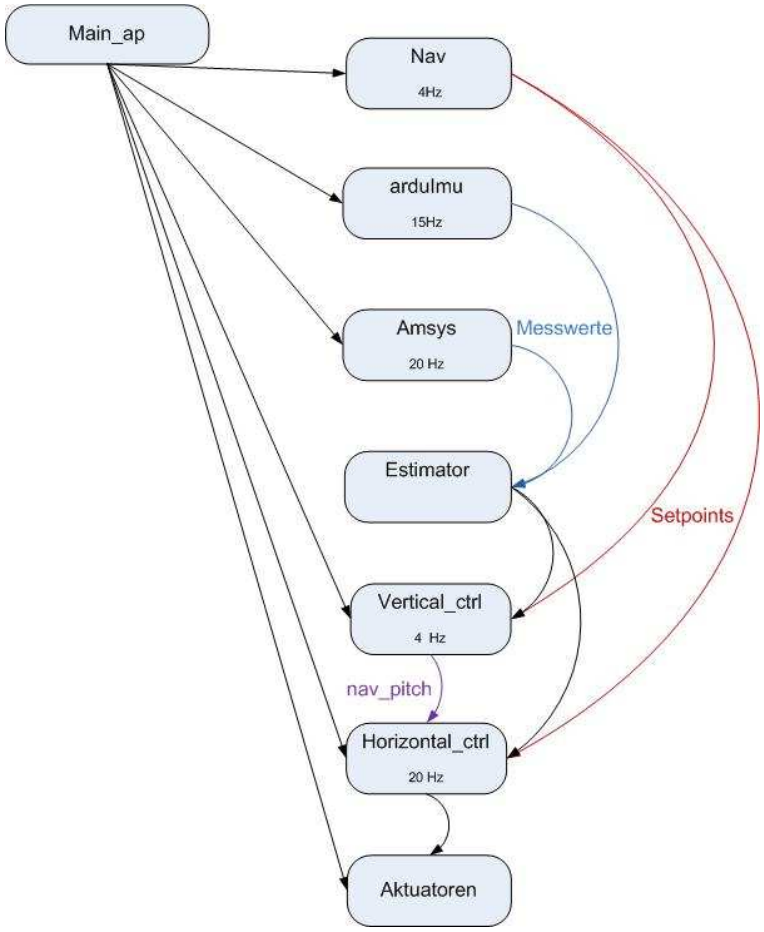

Abb. 45 Schema der wichtigsten c-Dateien der Paparazzi Architektur

5.1.4 Die wichtigsten Dateien

In diesem Abschnitt werden kurz die wichtigsten Dateien vorgestellt, um zukünftigen Projekt- oder Bachelorarbeiten eine Hilfe zu sein. Ausführlichere Informationen sind in der PA des Frühlingssemester [8] zu finden.

5.1.4.1 XML-Konfigurationsfiles

1) Airframe.xml (~/paparazzi3/conf/airframes)

Im *airframe.xml* werden alle flugrelevanten Informationen bekannt gegeben. Dazu zählen unter anderem auch die Regelparameter. Es ist gedacht um die Konfiguration des Flugzeuges festzulegen.

2) Flight_plan.xml (~/paparazzi3/conf/flight_plan)

Hier werden die verschiedenen Flugpatterns programmiert.

3) Messages.xml (paparazzi3/conf)

In der Datei *messages.xml* wird die Struktur einer Message festgelegt und ihr eine ID vergeben. Diese Datei wird durch ein „make"-Befehlt im paparazzi3 Verzeichnis in die Datei *messages.h* umgeschrieben, die dann im effektiven Code kompiliert wird.

4) **Defalut.xml** **(paparazzi3/conf/telemetry)**

Darin kann angegeben werden mit welcher Frequenz eine periodische Nachricht gesendet werden soll.

5) **Radio.xml** **(~/paparazzi3/conf/radios)**

Diese Datei ordnet den Kanälen der Fernsteuerung die Kommandos zu.

6) **Tuning.xml** **(~/paparazzi3/conf/settings)**

Unter den Settings wird festgelegt was für Parameter im GCS verfügbar sein werden.

5.1.4.2 C-Files

7) **Main_ap.c** **(paparazzi3/sw/sirborne)**

Dieses File ist das eigentliche Herzstück der Software. Es beinhaltet die Hauptroutine die zyklisch durchlaufen wird. Aus ihr werden alle Grundfunktionen aufgerufen.

8) **Ap_downlink.h** **(paparazzi3/sw/airbourne)**

Wenn eine periodische Message in diesem File deklariert wird, kann sie aus jedem anderen File aufgerufen werden, wenn dieser Header inkludiert wurde.

9) **Estimator.c** **(paparazzi3/sw/airborne)**

Im Estimator (Schätzer) befinden sich alle Variablen die irgendwie mit den gemessenen Fluglagen zu tun haben.

10) **Fw_h_ctl.c/h** **(paparazzi3/sw/airborne)**

Steht für Fixedwing horizontal Control. Diese Datei ist verantwortlich für die Regelung des Flugzeuges in der XY-Ebene (Horizontal).

11) **Fw_v_ctl.c/h** **(paparazzi3/sw/airborne)**

Steht für Fixedwing vertical Control. Diese Datei ist für die Regelung in der Höhe verantwortlich. Ausserdem regelt sie die Motorenleistung.

12) **Nav.c/h** **(paparazzi3/sw/airborne)**

Diese Datei berechnet den Sollwert für den Kurs.

13) **Amsys.c/h** **(paparazzi3/sw/airborne)**

In dieser Datei werden grundsätzlich die Analogwerte für die Barometer und Differenzdruck-Sensoren verarbeitet und in die entsprechenden Variablen geschrieben.

14) **Arduimu.c/h** **(paparazzi3/sw/airborne)**

In dieser Datei werden die Daten die vom IMU angefragt und nach Empfang in die Entsprechenden Variablen geschrieben. Dazu kommt das Versenden der GPS-Daten für den Abgleich der IMU.

5.2 Die I2C Schnittstelle

Das I²C-Protokoll ist als Master-Slave-Bus entwickelt worden. Somit stellt sich die Frage, wie die Kommunikation zwischen der ArduImu und dem Paparazzi-Board geregelt werden soll. Der Bus wird gebraucht um die Messdaten der IMU an den Autopiloten zu senden und um die GPS-Daten der ArduImu zur Verfügung zu stellen.

Zur Gestaltung der Kommunikation bieten sich zwei Varianten an. Entweder die ArduImu sendet seine Daten mit einer gewissen Frequenz, oder der Autopilot fragt den IMU nach den Daten. Um Änderungen an der Frequenz möglichst Benutzerfreundlich zu gestalten, wurde der Paparazzi Autopilot als Master an den Daten-Bus angemeldet. So kann im Treiber der ArduImu die Frequenz ohne weiteres eingestellt werden.

Zum Testen der Kommunikation wurde eine Testumgebung mit einem zusätzlichen Mikrokontroller-Board eingesetzt, die aus dem Paparazzi Autopiloten und einem Arduino besteht. Dabei generiert der Arduino bekannte Signale, die er auf Anfrage versendet. So kann die Initialisierung der I²C-Schnittstelle des Autopiloten überprüft und getestet werden. Das Ziel dieses Testes ist auch das Minimieren der möglichen Fehlerquelle, da so nur an einem Kommunikationspartner gearbeitet wird.

5.2.1 Adressierung

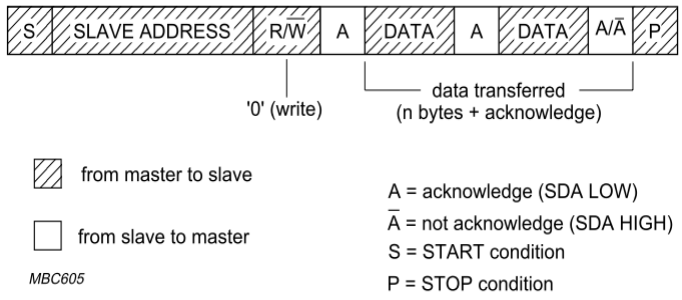

Abb. 46 I²C Adressierung nach Spezifikation [21]

Bei der Anmeldung der Kommunikationspartner ist ein besonderes Augenmerk auf die Adressierung zu setzten, da die zwei Mikrochips auf unterschiedlichen Bibliotheken aufbauen. Das Problem steckt in der Handhabung der Adresse: der ArduImu setzt hinter die 7-Bit Adresse ein zusätzliches Read/Write Bit, welches die Kommunikationsrichtung festlegt. Dies erfolgt nach der Spezifikation vom I²C-Bus. Beim Paparazzi Autopiloten gehört dieses Bit jedoch zu seiner 8-Bit Adresse dazu und wird je nach Verbindung gesetzt oder gelöscht.

	Dez	Hex	Bin
Paparazzi	34	22	0010001**0**
ArduImu	17	11	0010001

Tabelle 6: Adressierung des I²C Slaves

5.2.2 Anfrage der ArduImu-Daten

In den folgenden Abbildungen sind Ausschnitte der I²C Kommunikation dargestellt. Solange die SCL (Serial Clock Linie) konstant auf High gesetzt ist, ist der Daten-Bus frei. Bei einer Anfrage des Masters setzt er eine Nachricht mit der Adresse des gewünschten Slaves ab und hält anschliessend die Clock tief. Der Slave antwortet danach mit der Nachricht. Im vorliegenden Fall entsprechen die Daten den drei Lagewinkel und den drei Beschleunigungen zu je zwei Byte. Der ganze Anfrage Zyklus kann bis zu etwa 26 ms dauern. Dies geschieht, wenn der Prozessor von der ArduImu bei der Anfrage nicht umgehend auf den Event reagieren kann, da er beschäftigt ist.

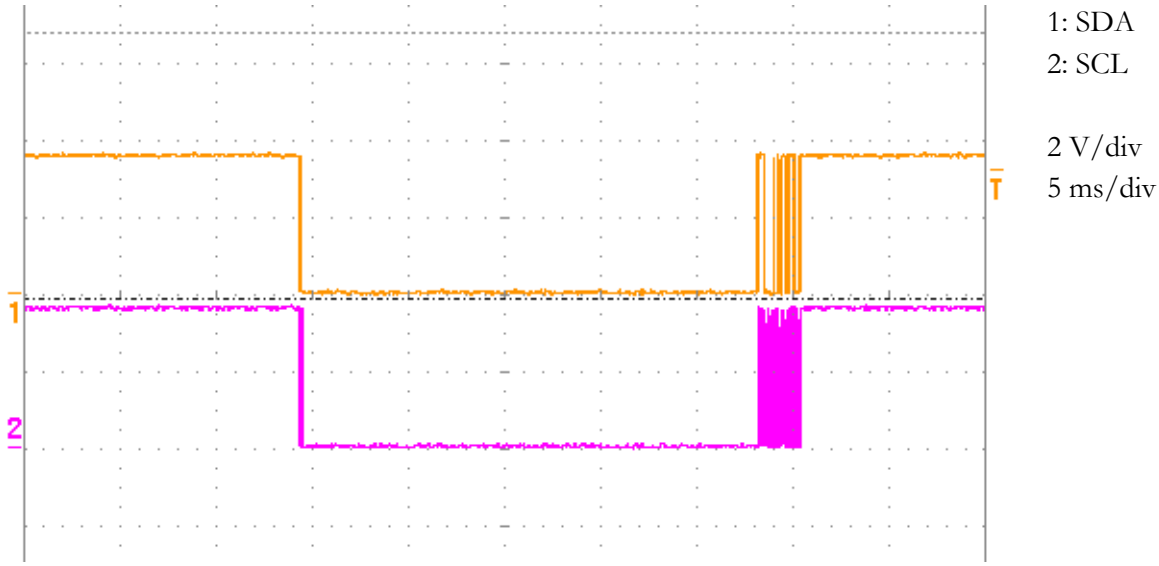

Abb. 47 I^2C Anfrage und Nachricht mit den aktuellen Lage-Daten

Die Anfrage startet mit dem Start Bit, worauf die Clock anfängt zu takten. Anschliessend folgt die 7 Bit Adresse, das R/W Bit und die Bestätigung. Diese Anfrage dauert nur 0.2 ms.

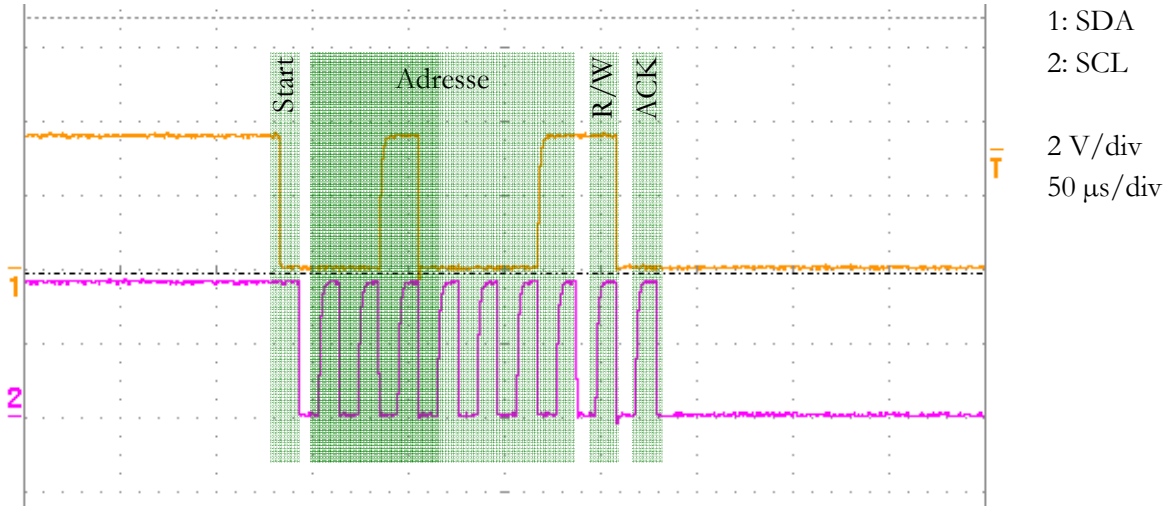

Abb. 48 I^2C Anfrage (Adresse)

Für die Übermittlung der 12 Bytes benötigten die Kommunikationspartner lediglich 2.5 ms. Die Empfangenen Daten werden byteweise in einen Buffer geschrieben. Um aus den zusammengehörigen Bytes einen Integer zu erhalten, müssen jeweils zwei Bytes durch Schifftoperatoren zusammengefügt werden.

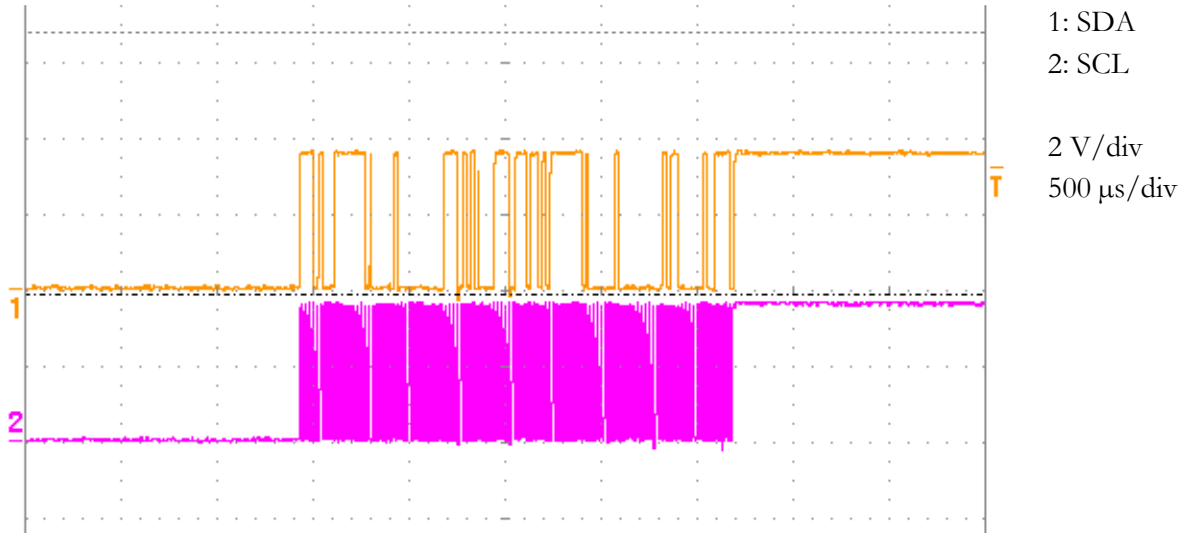

Abb. 49 I^2C Nachricht mit den aktuellen Lage-Daten

5.2.3 Versenden der GPS-Daten

Das Versenden der GPS-Daten, für den Abgleich der ArduImu, geschieht in zwei Pakete, da die notwendigen Daten zu gross für ein Paket ist, welche mit derselben Frequenz versendet werden, wie der GPS-Empfänger Daten zur Verfügung stellt. Dies bedeutet, dass die IMU alle 0.25 Sekunden die aktuellen GPS-Informationen erhält.

Die grössere Nachricht enthält die Informationen für die Position, Kurs und Geschwindigkeit. Sie besteht aus 28 Bit und benötigt in etwa 6.4 ms um versendet zu werden.

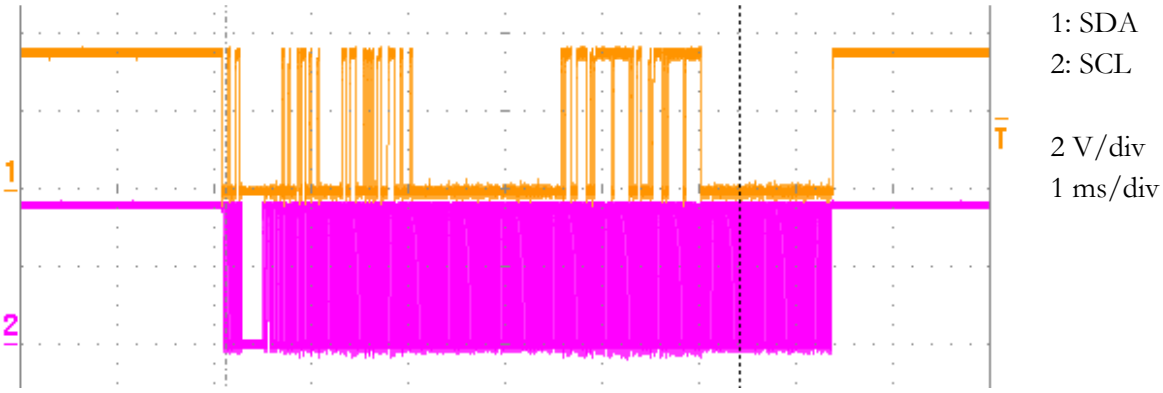

Abb. 50 I^2C Nachricht mit GPS-Daten Nr: 1

Das zweite Paket ist mit 13 Bit fast halb so gross und benötigt entsprechend weniger Zeit für die Übertragung. Diese Nachricht ist in etwa 3 ms versendet. Der Inhalt der Nachricht beschränkt sich vorwiegend auf die Güte der Daten, welche in der anderen Nachricht vorhanden sind. So wird zum Beispiel der IMU mitgeteilt, ob der GPS-Empfänger überhaupt ein brauchbares GPS Signal empfängt.

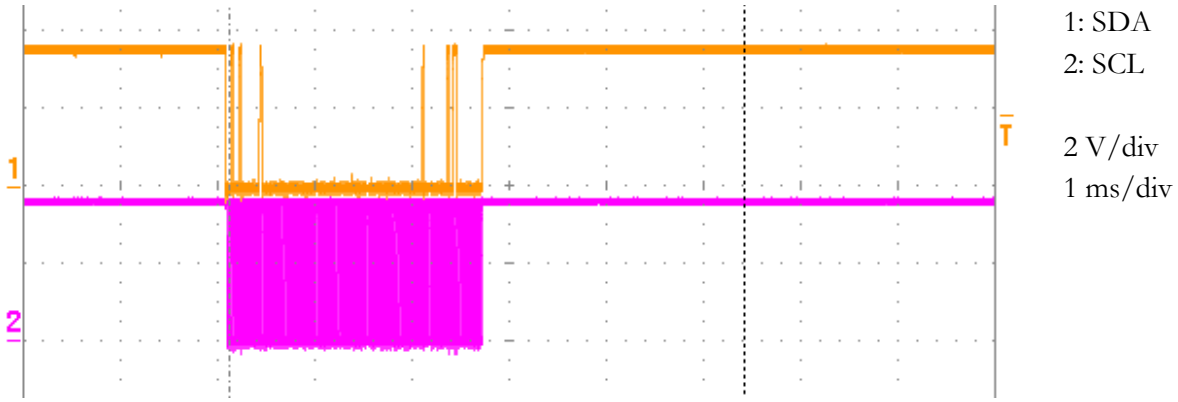

Abb. 51 I²C Nachricht mit GPS-Daten Nr: 2

5.2.4 Weitere Nachrichten

Um den kompletten Umfang der ArduImu zu nutzen, benötigt es wie im Kapitel 3.7.3 beschrieben einen elektrischen Kompass. Die Firmware der ArduImu ist vorbereitet um über I²C die Messwerte des Kompasses zu empfangen. Jedoch stellt sich somit die Frage, ob die Leitung nicht überlastet wird.

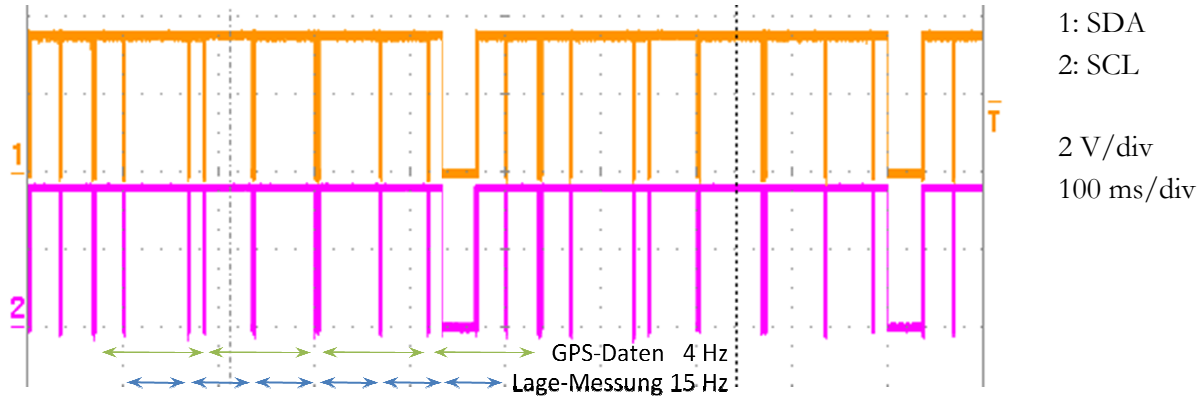

Abb. 52 Auslastung vom I²C-Bus

Wird die Leitung über eine Sekunde beobachtet, erkennt man die Frequenzen der Nachrichten. Ersichtlich ist auch, dass bei ungünstiger Konstellation die IMU kurzzeitig den Bus blockiert. Jedoch sollte die Leitung noch genügend Kapazität haben für eine weitere Nachricht in der die Daten eines Kompasses an die IMU gesendet werden. Diese Nachricht sollte 10 Hz verschickt werden.

5.3 ArduImu

Die Änderungen die am IMU-Code gemacht werden müssen, damit die Werte vom Autopiloten gelesen werden können sind sehr gering. Siehe dazu Kapitel 5.3.

Die ArduImu-Software ist kompatibel mit dem Arduino und kann sehr gut mit der Entwicklungsumgebung des Arduinos bearbeitet werden (http://www.arduino.cc/). Die Arduino-Plattform bietet eine sehr gute und übersichtliche Bibliothek für die Implementierung der I²C-Schnittstelle in einem Programm. Die Bibliothek heisst „Wire library“ und kann auf der Webseite von Arudino [22] gratis heruntergeladen werden und vereinfacht dem Benutzer die Schnittstelle soweit, dass er nur noch ein paar wenige Funktionen braucht, deren Namen für sich selber sprechen.

Das Bearbeiten der Software stellt über die Entwicklungsumgebung des Arduinos keinerlei Probleme dar. Ein grösseres Problem war das Flashen des Mikrokontrollers. Uns war es nicht möglich den

Mikrokontroller mit dem empfohlene FTDI Kabel [24] nach den Angaben der offiziellen ArduImu Seite [23] zu flashen. Um die Software auf den Mikrokontroller zu laden, mussten wir den Atmel AVR ISP MKII verwenden. Dies ist ein weit verbreiteter USB zu ISP Adapter, mit dem Mikrokontroller programmiert werden können. Um den MKII zu verwenden und den Kontroller zu programmieren haben wir das AVR-Studio verwendet, welches gratis im Internet zur Verfügung steht.

Die folgende Anleitung zeigt die wichtigsten Änderungen, die am Originalcode gemacht werden mussten.

5.3.1 Importieren der „Wire-library“

Die „wire-library“ wird im neuesten Code (Version 1.7) schon importiert. Es muss aber sichergestellt werden, dass die Library auch im entsprechenden Verzeichnis vorhanden ist. Kontrollieren Sie dazu im Verzeichnis, aus dem Sie die Arduino-Umgebung starten, ob im Pfad harware\libraries der Ordner „Wire“ vorhanden ist.

5.3.2 Senden der IMU-Daten an Paparazzi

5.3.2.1 Schalter für Output am I²C-Port

Damit am Originalprogramm nur minimale Änderungen gemacht werden müssen, wurde ein „Schalter“ eingefügt, mit dem die Schnittstelle ein- oder ausgeschaltet werden kann. Dieser Schalter ist in der Datei *arduimu.pde* bei den Deklarationen zu finden.

```
#define PRINT_I2C_Data 1     //Will print IMU-data
int I2C_Message_ar[6];                    // Array für Roll; Pitch ; Yaw ; ACCX; ACCY; ACCZ
```

5.3.2.2 I²C Output Handler

Die Idee war es, dass der Autopilot die Lagedaten beim IMU anfordert und sie dann verwertet. Der Autopilot schickt also eine Anforderung über bestimmte Daten und worauf der IMU reagieren muss. Hierfür stellt die „Wire-library“ die Funktion *on_Request()* zur Verfügung. Sie überwacht die Datenleitung und ruft bei einer Anfrage an die eigene Adresse die entsprechende Funktion auf. Diese Deklarationen werden im *arduimu.pde* in der Setuproutine bei der Initialisierung durchgeführt.

```
#if PRINT_I2C_Data == 1
        Wire.begin(17);                         // join i2c bus with address #17
        Wire.onRequest(requestEvent);                   //call requestEvent() after I²C-Request
#endif
```

5.3.2.3 Der Output

In der Funktion *RequestEvent()* wird nun definiert was bei einer I²CAnforderung alles ausgeführt werden soll. Ein Problem beim Verschicken der Daten, stellt das Bereitstellen der Daten und das Konvertieren in ein geeignetes Format dar. Die Funktion Send() der „Wire-library“ ist so definiert, dass sie nur ein Objekt in einem Send verschicken kann. Um zu umgehen, dass für jeden Winkel bzw. Beschleunigung eine neue Anfrage vom Autopiloten gesendet werden muss, kann eine Pointervariable auf einen Array mitgegeben werden. Die Funktion *send()* verschickt dann ein Paket mit allen Bytes an einem Stück. Diese Werte müssen später im Autopiloten Byteweise wieder zusammengeführt werden. Um zu verhindern, dass Floatwerte verschickt werden, die später vom Autopiloten wieder zusammengesetzt werden müssten, werden die Werte mit Hundert multipliziert um zwei Nachkomma-

stellen zu retten und in eine Integerzahl umgewandelt. So müssen keine Float-Zahlen verschickt werden. Nun muss noch ein Pointer des Datenarrays erstellt werden, dessen Inhalt von der *Send()*-funktion verschickt wird.

```
void requestEvent(){
  //MEssage Array : Roll; Pitch ; Yaw ; ACCX;ACCY;ACCZ
  // Float Number is multipited with 100 and converted to an Integer, for sending via I2C.
    I2C_Message_ar[0] = int(ToDeg(roll)*100);
    I2C_Message_ar[1] = int(ToDeg(pitch)*100);
    I2C_Message_ar[2] = int(ToDeg(yaw)*100);
    I2C_Message_ar[3] = int(read_adc(3)*100);
    I2C_Message_ar[4] = int(read_adc(4)*100);
    I2C_Message_ar[5] = int(read_adc(5)*100);

    byte* pointer;
    pointer = (byte*) &I2C_Message_ar;
    Wire.send(pointer, 12);
}
```

Hier finden Sie einige Anleitungen und auch die Firmware der ArduImu.
http://code.google.com/p/ardu-imu/wiki/HomePage?tm=6

5.3.3 Empfangen der GPS-Daten

Die ArduImu-Softwareversion 1.7 ist für die Verwendung von GPS-Modulen von u-blox vorbereitet. Es müssen also die Daten die über I²C empfangen werden, in die entsprechenden Variablen geschrieben werden und die Logik entsprechend angepasst werden.

5.3.3.1 Die Datei Arduimu.pde

Auch hier wurden wieder „*Schalter*“ definiert, die es erlauben diese Funktionen ein- oder auszuschalten.

```
#define GET_GPS_PAP
```

Es werden zusätzliche Variablen verwendet die Werte zwischenspeichern.

```
#if GET_GPS_PAP
    // übergnagsvariablen für die GPS Werte zwischen zu speichern
    long iTOW2=0; //GPS Millisecond Time of Week
    long lon2=0; // Store the Longitude from the gps to pass to output
    long lat2=0; // store the Latitude from the gps to pass to output
    long alt2=0;  //Height above Ellipsoid in millimeters
    long alt_MSL2=0; //This is the altitude in millimeters
    float speed_3d2=0; //Speed (3-D)
    float ground_speed2=0
    byte recPakOne = 0x00;
    byte Paparazzi_GPS_buffer[UBX_MAXPAYLOAD];
    int gpsDataReady = 0;  // sind neue GPS-Daten vorhanden ??
    byte stGpsFix;
    byte stFlags;
    byte solGpsFix;
    byte solFlags;
    byte messageNr;
#endif
```

Bei der Initialisierung des IMU muss in der Setuproutine der Empfangsevent für das Empfangen der GPS-Daten deklariert werden.

```
#if GET_GPS_PAP == 1
    //Wire.begin(17);
    Wire.onReceive(receiveEvent); // register event --> Output
  #endif
```

Der *recieveEvent* wurde, um alle I²C-Events an einer Stell zu haben, in die Datei *Output.pde* geschrieben. Im Wesentlichen schreibt diese Routine die empfangenen Bytes in den dafür erstellten Buffer und ruft die Routine für das Zusammenführen und Speichern in die entsprechenden Werte auf.

```
void receiveEvent(int howMany){
    for(int i=0; i < howMany; i++){
            Paparazzi_GPS_buffer[i]=Wire.receive();
    }
   parse_ubx_gps();              // Parse new GPS packet...
   GPS_timer=DIYmillis();        //Restarting timer...
   gpsDataReady=1;
}
```

5.3.3.2 Die Datei „GPS_UBLOX.pde“

Hier finden die tiefgreifendsten Änderungen am Originalcode statt. Es wird hier nicht der ganze Code vorgestellt, sondern nur die essentiellen Abschnitte sowie die Logik.

Da das Paparazzi-Board nicht fähig ist alle GPS-Daten in einem I²C-Paket zu verschicken, mussten die Daten in zwei Einzelpakete getrennt werden. Weil der Lese-/Schreibbuffer des Mikrokontrollers nicht garantiert erst wieder beschrieben wird wenn die Daten gelesen wurden, mussten die Daten zeitlich gestaffelt werden, so dass ein Überschreiben der Werte nicht mehr möglich ist. Dies erfordert eine entsprechende Logik um die Datenpakete wieder zusammen zu führen. Dazu wurden folgender Abfragen programmiert, die das Handling der Daten verwalten und die Daten wieder zusammenführen.

Setzen der MessageNr

```
messageNr = Paparazzi_GPS_buffer[0];  // ID der Nachricht ( 0,1 )
```

Schreiben der Werte in die entsprechenden Variablen. Die Kommentare am Ende der Linie sind die Bytes in denen sich der Wert im I²C-Datenpaket befindet.

```
if(messageNr == 0x00){
      //Nachricht 0
          iTOW2 = join_4_bytes(&Paparazzi_GPS_buffer[1]);                          //1,2,3,4
          lon2 = join_4_bytes(&Paparazzi_GPS_buffer[5]);                           //5,6,7,8
          lat2 = join_4_bytes(&Paparazzi_GPS_buffer[9]);                           //9,10,11,12
          alt2 = join_4_bytes(&Paparazzi_GPS_buffer[13]);                          //13,14,15,16
          alt_MSL2 = join_4_bytes(&Paparazzi_GPS_buffer[17]);                      // 17,18,19,20
          speed_3d2 = (float)join_4_bytes(&Paparazzi_GPS_buffer[21])/100.0;    //21,22,23,24
          ground_speed2 = (float)join_4_bytes(&Paparazzi_GPS_buffer[25])/100.0;          // 25,26,27,28
          recPakOne=0x01;                                                  //29,30 31
      }
```

```
if(messageNr == 0x01 && recPakOne==0x01){
        // Nachricht 1
        ground_course = (float)join_4_bytes(&Paparazzi_GPS_buffer[1])/100000.0;  // 1,2,3,4
        ecefVZ=(float)join_4_bytes(&Paparazzi_GPS_buffer[5])/100;                // 5,6,7,8
        numSV=Paparazzi_GPS_buffer[9];
        stGpsFix=Paparazzi_GPS_buffer[10];
        stFlags=Paparazzi_GPS_buffer[11];
        solGpsFix=Paparazzi_GPS_buffer[12];
        solFlags=Paparazzi_GPS_buffer[13];

        iTOW = iTOW2;
        lon = lon2;
        lat = lat2;
        alt = alt2;
        alt_MSL = alt_MSL2;
        speed_3d = speed_3d2;
        ground_speed = ground_speed2;

        messageNr=messageNr+1; //2
}
```

Um dem Programm Informationen über die Qualität der GPS-Daten zu geben werden noch einige Vergleiche und Kontrollen gemacht.

```
if(messageNr == 0x02){
      if((stGpsFix >= 0x03)&&(stFlags&0x01)){
        gpsFix=0; //valid position
        digitalWrite(6,HIGH);  //Turn LED when gps is fixed.
        GPS_timer=DIYmillis(); //Restarting timer...
      }
      else{
        gpsFix=1; //invalid position
        digitalWrite(6,LOW);
      }

      if((solGpsFix >= 0x03)&&(solFlags&0x01)){
        gpsFix=0; //valid position
        digitalWrite(6,HIGH);  //Turn LED when gps is fixed.
        GPS_timer=DIYmillis(); //Restarting timer...
      }
      else{
        gpsFix=1; //invalid position
        digitalWrite(6,LOW);
      }

      if (ground_speed > SPEEDFILT && gpsFix==0) gc_offset = ground_course - ToDeg(yaw);
        recPakOne=0x00;
}
```

5.4 Paparazzi

In diesem Kapitel werden die Treiber der hinzugefügten Sensoren und die mit den Messwerten verbundenen Regelungen erläutert.

5.4.1 ArduImu

Das Modul *ArduImu.xml* beinhaltet neben der init-Methode zwei weitere Methoden. Diese werden periodisch mit einer einstellbaren Frequenz aufgerufen. Im Initialteil werden nur die notwendigen Konstanten zugewiesen und Anfangswerte gesetzt.

Die zwei periodischen Funktionen dienen dem Anfragen der Fluglage und dem Versenden der GPS-Daten über den I²C-Port. Wobei sich die verschiedenen Nachrichten-Pakete den gleichen Buffer teilen müssen. Deshalb muss beim Benutzten des Buffers kontrolliert werden, ob der Zugriff erfolgen darf, oder ob noch Daten im Buffer stehen, welche verarbeitet werden müssten. Diese Aussperrung erfolgt über das Setzten und Löschen von Flags, in welche der aktuellen Stand der Kommunikation gespeichert ist.

Die Funktion *ArduImu_periodic* wird mit 15 Hz aufgerufen. Sie dient dem Anfragen der aktuellen Lage, welche von der IMU gemessen wurde, und dem Aufrufen der Methode *IMU_Daten_verarbeiten*.

Die zweite periodische Funktion *ArduImu_periodicGPS* hat den Auftrag, der IMU die GPS-Informationen weiter zu leiten. Jedoch muss, bevor die GPS-Daten für den Versand bereit gestellt werden, geprüft werden, ob im Buffer noch Lage-Informationen gespeichert sind. Ist dies der Fall, so wird die Methode *IMU_Daten_verarbeiten* aufgerufen. Nach diesem Aufruf, oder wenn der Buffer leer ist, können die GPS-Daten in den Buffer geschrieben werden. Anschliessend wird mit einem *i2c0_transmitt* Aufruf der Buffer versendet. Ein Problem stellte die maximale Länge der Nachricht dar, welche über die I²C-Schnittstelle versendet werden kann. So musste die GPS-Nachricht in zwei Pakete aufgeteilt werden, welche abwechselnd versendet werden. Daraus resultiert die notwendige Frequenz von 8 Hz mit welcher die Funktion *ArduImu_periodicGPS* aufgerufen werden muss, um eine Updaterate von 4 Hz zu erreichen.

Die interne Funktion *IMU_Daten_verarbeiten* sorgt sich um das Zusammenfügen der empfangen Daten. Das bedeutet, dass aus jeweils zwei Bytes eine float Zahl mittels Schiftoperationen gebildet wird. Auch das Weiterleiten der Winkel in die estimator-Variablen gehört zum Funktionsumfang.

5.4.2 Amsys

Der Aufruf des Treibers läuft entsprechend dem Code der IMU über die Funktionen, welche ein Modul bereitstellt. Somit kann die gewünschte Frequenz der aufgerufenen Funktion nach Belieben eingestellt werden. Eine Frequenz von 20 Hz ist zurzeit eingestellt, und hat sich auch bewährt.

Der Treiber der Amsys Druck-Sensoren ist ebenfalls ähnlich, wie der oben beschriebene Treiber der IMU. Der grösste Unterschied liegt in dem zu verarbeitenden Signal. Denn die Druck-Sensoren haben je einen analogen Ausgang. Dieser wird in der periodischen Funktion *airspeed_AMSYS_periodic* eingelesen und in einem Buffer abgespeichert. Aus dem Buffer wird anschliessend der Mittelwert gerechnet, so dass das Signal geglättet wird. Danach werden nur noch die Variablen bereitgestellt, welche der Luftgeschwindigkeit und der Höhe über Meer entsprechen.

5.4.3 Airspeed-Regelung (fw_v_ctl.c)

Die Regelung, welche für die Höhe und die Geschwindigkeit zuständig ist, also die vertikale Regelung, musste für die Zwecke dieses Projektes ergänzt und angepasst werden. Die ganze Datei beruht eigentlich auf zwei Methoden, die vom *main_ap.c* aufgerufen werden. Dies ist die Funktion *v_ctl_climb_loop* und *v_ctl_throttle_slewed*. Wobei die Letzte nur ein PT-1 Verhalten für den Motor nachbildet. Das heisst, dass eine sprunghafte Änderung der Stellgrösse für den Motor nicht übernommen wird, sondern dass erst verzögert die gewünschte Stellgrösse erreicht wird.

In dem climb Regelkreis wird in einem Switch die gewünschte Regelung der Luftgeschwindigkeit ausgewählt. Der Nutzer hat somit die Möglichkeit zwischen den im Kapitel 0 dokumentierten Varianten umzuschalten. Dies kann während dem Flug im GCS eingestellt werden.

Um die Funktionsweise der Regelungen zu studieren, empfiehlt sich entweder das oben erwähnte Kapitel zu konsultieren oder im Anhang die Simulink Modelle [25] zu untersuchen.

5.4.4 Einrichten

Nach der Installation oder Update des Programms Paparazzi sind einige Modifikationen am Source Code durchzuführen, um die ArduImu, die Drucksensoren und die Airspeed-Regelung einzurichten. Im Folgenden werden die wichtigsten Schritte erklärt. Die detaillierten Anleitungen sind im Anhang zu finden [26].

Da die ArduImu und die Amsys-Sensoren als Module programmiert sind, müssen im verwendeten Airframe die entsprechenden Module geladen werden. Im Abschnitt der als Makefile deklariert ist, müssen zudem die Flags *USE_I2C0*, *USE_ADC5*, *USE_ADC6* und *USE_MODULES* definiert werden. Wenn es sich um einen neu gekaufte Hardware handelt, so ist darauf zu achten, dass der ADC6 Eingang modifiziert werden muss. Siehe dazu Kapitel 4.3.

Damit die Module aufgerufen werden können, müssen sie auch vorhanden sein. Darum müssen die xml-Dateien und die dazugehörigen Ordner an die entsprechenden Orte kopiert werden. Somit werden die Messwerte in die gewünschten esimator-Variabeln geschrieben. Um die Airspeed-Regelung noch zum Laufen zu bringen muss die Datei fw_v_ctl.c erstzt werden.

Der Aufruf der Infrarot-Sensoren aus dem *main_ap.c* muss im durch löschen oder auskommentieren verhindert werden, da sonst der *estimator_phi* und *estimator_theta* laufend überschrieben würden.

Somit kann der Code seine Aufgabe erfüllen. Um jedoch die Messwerte auf der Bodenstation überprüfen zu können, benötigt es zusätzlich eine periodische Nachricht. Siehe dazu im Anhang in der Trick Kiste [20] nach.

6 Ausblick und Fazit

6.1 Mögliche- und weiterführende Arbeiten

1) Kompass

Da die Kompensation des IMU-Drifts in der Z-Achse noch nicht vollständig gelöst ist, empfehlen wir eine möglichst schnelle Anpassung durch einen Magnetometer vorzunehmen. Dieser Sensor dürfte den Drift und die fehlerhaften Werte drastisch reduzieren, wenn nicht sogar vollständig eliminieren.

2) Optimiertes Reglereinstellverfahren

Um den Zeitaufwand bezüglich den Reglereinstellungen zu minimieren währe es vorteilhaft, ein Verfahren zu entwickeln mit dem die Regelung am Computer optimiert werden könnte.

3) Bessere IMU

Falls der Magnetometer den gewünschten Effekt nicht erzielt, muss geprüft werden, ob es nötig sein sollte einen genaueren IMU zu integrieren. Mögliche IMUs wären der VectorNav oder X-Sens. Es sollte aber darauf geachtet werden, dass GPS und Magnetometer vorhanden sind. Ausserdem sollte darauf geachtet werden, wie die Daten dem Autopiloten übermittelt werden können.

4) Strommessung

Bis anhin wurde die Energie, die der Motor verbraucht hat, immer mit einer linearen Funktion der Throttle-Stellung integriert. Das Problem ist, dass dieser Wert nur angenähert ist, da der Strom nicht linear zur Throttle-Stellung ist. Eine einfache und kostengünstige Variante um genaue Werte über die verbrauchte Energie zu bekommen wäre ein Strommesser, der den Stromfluss aus dem Hauptakku misst. Dies würde eine grosse Gewissheit über die noch vorhandene Energiemenge geben, die sich noch im Akku befindet und somit eine gewisse Sicherheit über die noch zur Verfügung stehende Energiemenge.

5) Telemetrie

Für die Verbindung zur Bodenstation wird ein X-Bee von Maxstream verwendet. Dies ist ein eher kleines und dementsprechend nicht sehr leistungsfähiges Modem. Um die Telemetriedaten sicher, auch über grosse Distanzen zu versenden würde sich ein entsprechend leistungsfähigeres Modem anbieten.

6) Einbau in die Drohne und Parametrierung

Der Einbau in die Drohne und erste Flugversuche ist sicher ein Punkt, der bald durchgeführt werden kann. Die Drohne wird sich mit Sicherheit ganz anders verhalten, als das Testflugzeug (MAJA). Erste Erfahrungen mit der Parametrierung der Drohne sollten darum bald gesammelt werden.

7) Flight-Kamera

Eine Flightcam könnte zusätzlich für gewisse Sicherheit sorgen. Während unserer Versuche ist es passiert, dass das Flugzeug vom Kurs abkam. Wenn das Flugzeug an der Grenze der Sichtweite ist, kann es für den Piloten sehr schwierig sein noch zu fliegen. Eine Kamera die sich auf dem Flugzeug befindet, kann es dem Piloten ermöglichen das Flugzeug zurückzufliegen. Ausserdem können die Bilder interessant für die Flugauswertung sein.

8) **Gehäuse für das Autopilotensystem**

Im Moment sind die Autopilotkomponenten mit Klettband auf eine Trägerplatte geklebt. Alle Verbindungen sind nach bestem Wissen und Gewissen befestigt und sollten gut halten. Für eine saubere Montage in der Drohne wäre es aber denkbar, eine Box zu entwickeln, in der das Energiemanagement und die Kabelführung weitestgehend gelöst ist, Anschlüsse für externe Komponenten vorhanden sind und eine grösstmögliche Sicherheit gegenüber Ausfällen bietet.

9) **Sicherheit**

Während unserer Flugversuche hatten wir nie Probleme wegen eines Ausfalles des Mikroprozessors oder Kabelprobleme. Für den Einsatz in der UMARS-Drohne wäre es aber sicher sinnvoll diese Probleme genau zu analysieren und allenfalls so zu erweitern, dass durch redundante Systeme eine grösstmögliche Sicherheit gewährleistet ist. Denkbar wären hier aus unserer Sicht eventuell zwei Autopilotensysteme die sich gegenseitig überprüfen können. Ein weiteres Sicherheitsrisiko ist bei einem Ausfall des Autopiloten, dass die Drohne nicht einmal mehr über die Handsteuerung (RC-Fernsteuerung) gesteuert werden kann. Dies könnte durch einen zusätzlichen Schaltkreis oder Mikrokontroller gelöst werden.

10) **Start und Landung**

Aus Zeitmangel war es uns nicht möglich, autonome Starts und Landungen durchzuführen. Grundsätzlich ist dies möglich und im Source Code vorhanden.

11) **Mehrere Fernsteuerungen die abwechselnd das Flugzeug übernehmen können.**

Eine wünschenswerte Funktion wäre, wenn die Drohne einer langen Strecke entlang geflogen werden soll, dass sich die Sicherheitspiloten am Boden der Strecke entlang aufstellen könnten und einer nach dem anderen das Flugzeug übernehmen könnte, wenn es in seine Sichtweite kommt und dem nächsten Piloten übergeben, wenn er es nicht mehr sieht. So könnte trotz der gesetzlichen Bestimmungen das Flugzeug über grössere Strecken geflogen werden.

6.2 Fazit

In dieser Bachelorarbeit und in der vorangegangenen Projektarbeit haben wir uns mit dem Autopilotensystem Paparazzi beschäftigt. Da es kein Handbuch oder Ähnliches dazu gibt, musste alles selbst erarbeitet und ausprobiert werden. Wir haben versucht, möglichst viele Erfahrungen mit in die Berichte zu schreiben um nachfolgenden Projektarbeiten eine Einstiegshilfe zu geben. Natürlich konnten wir nicht jedes Detail erwähnen und vieles konnte nicht vollständig erklärt werden, da dies den Umfang dieses Berichts gesprengt hätte.

Die Flüge in Lommis, aber auch die Implementierung neuer Funktionen war äusserst interessant und hat uns Einblicke in viele neue Gebiete gegeben. Das Programmieren des Mikrokontrollers auf dem Paparazzi und dem IMU, die Verwendung der I^2C Schnittstelle, serielle Datenübertragung sowie das Arbeiten mit der komplexen Elektronik war für uns absolutes Neuland, das uns beide einen tiefen Einblick in solche Systeme bot. Es war auch äusserst spannend die wie Flugregelung funktioniert, wie sie in einem C-Programm geschrieben werden und wie die ganze Theorie, die wir während des Studiums gehabt hatten, in der Luft an der Drohne angewandt werden konnte.

Der erste Schritt für die Einbindung des Autopilotensystems in das UMARS-Projekt ist gelungen. Es bedarf aber noch viel Arbeit um das Projekt mit allen Sicherheitsaspekten vollständig abschliessen zu können. Wir wünschen weiterhin viel Erfolg

7 Verzeichnisse

7.1 Abbildungsverzeichnis

7.2 Tabellenverzeichnis

7.3 Literatur- und Quellenverzeichnis

[1] http://paparazzi.enac.fr/wiki/Tuning, abgerufen am 11. August 2010

[2] http://de.wikipedia.org/wiki/Serielle_Schnittstelle, abgerufen am 11. August 2010

[3] http://de.wikipedia.org/wiki/Serial_Peripheral_Interfacem abgerufen am 11. August 2010

[4] http://dublin.zhaw.ch/~tham/TIn2/projekt/iic.pdf, abgerufen am 11. August 2010

[5] http://www.engr.usu.edu/wiki/index.php/OSAM, abgerufen am 11. August 2010

[6] http://code.google.com/p/ardu-imu/wiki/HomePage, abgerufen am 11. August 2010

[7] http://code.google.com/p/ardu-imu/wiki/Hardware, abgerufen am 11. August 2010

[8] Emilio Schmidhauser, René Chaney, PA: Flugregelung einer Kleindrohen für Forschungszwecke, 20.05.2010, ZHAW

[9] Digitaler Anhang/Sensoren/ArduIMU/Datenblatt Beschleunigungssensor adxl335.pdf

[10] Digitaler Anhang/Sensoren/ArduIMU/Datenblatt Roll-Pitchsensor lpr530al.pdf und Digitaler Anhang/Sensoren/ArduIMU/Datenblatt Yawsensor ly530alh.pdf

[11] http://www.umars.ch, abgerufen am 11. August 2010

[12] Digitaler Anhang/GPS/u-blox6.pdf, abgerufen am 11. August 2010

[13] http://paparazzi.enac.fr/wiki_images/Tiny_v2-1_Schematic.png, abgerufen am 11. August 2010

[14] http://www.mikrocontroller.net/articles/AVR-Tutorial:_ADC, abgerufen am 11. August 2010

[15] Martin Geiger, Thomas Matti, BA UMARS-Air Data Boom, 21.05.2010, ZHAW

[16] Digitaler Anhang/Sensoren/Drucksensoren/AMSYS.de.ams4711(d).pdf

[17] http://de.wikipedia.org/wiki/Differenzdrucksensor, abgerufen am 11. August 2010

[18] http://vrhome.net/vassilis/category/paparazzi, abgerufen am 11. August 2010

[19] http://www.eagletreesystems.com/standalone/standalone.htm, abgerufen am 11. August 2010

[20] Digitaler Anhang/Software/trickkiste.doc

[21] http://www.nxp.com/acrobat_download2/literature/9398/39340011.pdf

[22] http://www.arduino.cc/playground/Learning/I2C, abgerufen am 11. August 2010

[23] http://code.google.com/p/ardu-imu/wiki/HomePage, abgerufen am 11. August 2010

[24] Digitaler Anhang/Software/ArduIMU_Firmware/FTDI_Kabel.pdf

[25] Digitaler Anhang/Regelung Airspeed_matlabfiles/mit Sub System

[26] Digitaler Anhang/Software/Daten fuer Amsys Integration_Paparazzi und Digitaler Anhang/Software/Daten fuer ArduIMU Integration_Paparazzi

[27] Digitaler Anhang/Sensoren/Magnetometer/HMC5843.pdf

8 Anhang

8.1 Variablen

Simulink Namen	Beschreibung	c-Namen	c-Datei	Wert	Einheit	used in
estimator hspeed dir	Richtung des horizontalen Geschwindigkeits Vektors zum Boden (aus GPS-Daten)	estimator_hspeed_dir	estimator			h_ctl
course setpoint	Kurs Sollgrösse	h_ctl_course_setpoint	fw_h_ctl			h_ctl
estimator hspeed mod	Betrag des horizontalen Geschwindigkeits Vektors zum Boden (aus GPS-Daten)	estimator_hspeed_mod	estimator			h_ctl
nominal airspeed	eingestellte Konstante aus Airframe	NOMINAL_AIRSPEED	MAJA.xml	12	m/s	h_ctl
course pgain	Proportional Faktor für Kurs Regelung	h_ctl_course_pgain	fw_h_ctl	-1.1		h_ctl
roll max setpoint	begrenzt den maximalen Rollwinkel	ROLL_MAX_SETPOINT	MAJA.xml	0.5	rad	h_ctl
course pre bank		h_ctl_course_pre_bank	fw_h_ctl	0		h_ctl
course pre bank correction		h_ctl_course_pre_bank_correction	fw_h_ctl	1		h_ctl
course dgain	D-Faktor für Kurs Regelung	h_ctl_course_dgain	fw_h_ctl	0		h_ctl
roll pgain	Proportional Faktor für Roll Regelung	h_ctl_roll_attitude_gain	fw_h_ctl	-7500		h_ctl
estimator phi	Roll Winkel	estimator_phi	estimator			h_ctl
v ctl throttle setpoint	Motor Sollgrösse	v_ctl_throttle_setpoint	fw_v_ctl			h_ctl
aileron of throttle		h_ctl_aileron_of_throttle	fw_h_ctl	null		h_ctl
MAX PPRZ	maximaler paparazzi wert	MAX PPRZ	paparazzi.h	9600		h_ctl
pitch setpoint	Pitch Sollgrösse	h_ctl_pitch_setpoint	fw_h_ctl			h_ctl
pitch dgain	D Faktor für Pitch Regelung	h_ctl_pitch_dgain	fw_h_ctl	-10		h_ctl
elevator of roll		h_ctl_elevator_of_roll	fw_h_ctl	1200		h_ctl
estimator theta	Pitch Winkel	estimator_theta	estimator			h_ctl
pitch pgain	Proportional Faktor für Pitch Regelung	h_ctl_pitch_pgain	fw_h_ctl	-6000		h_ctl
pgain boost	Faktor der bei agressivem Climb die climb rate verstärkt	altitude_pgain_boost	fw_v_ctl	1		v_ctl
altitude pre climb		v_ctl_altitude_pre_climb	fw_v_ctl	0		v_ctl
altitude pgain	Proportional Faktor für Altitude Regelung	v_ctl_altitude_pgain	fw_v_ctl	-0.04		v_ctl
altitude max climb	maximale Steigrate	V_CTL_ALTITUDE_MAX_CLIMB	fw_v_ctl	2	m/s	v_ctl
altitude setpoint	Altitude Sollgrösse	v_ctl_altitude_setpoint	fw_v_ctl			v_ctl
estimator z	Höhe aus GPS-Daten	estimator_z	estimator			v_ctl

Simulink Namen	Beschreibung	c-Namen	c-Datei	Wert	Einheit	used in
auto_throttle_pitch_of_vz_pgain	P Vers:ärkung für pitch aus Steigrate (velocity z)	v_ctl_auto_throttle_pitch_of_vz_pgain	v_ctl	0.5		v_ctl
auto_throttle_pitch_of_vz_dgain	D Faktor für pitch aus Steigrate (velccity z)	v_ctl_auto_throttle_pitch_of_vz_dgain	v_ctl	0		v_ctl
auto_throttle_cruise_throttle	Einstel ung für Motorenleistung (% von max Leistung)	v_ctl_auto_throttle_cruise_throttle	v_ctl	0.4		v_ctl
auto_throttle_climb_throttle_increment		v_ctl_auto_throttle_climb_throttle_increment	v_ctl	0.1		v_ctl
auto_throttle_pgain	Proportional Faktor für auto throttle Regelung	v_ctl_auto_throttle_pgain	v_ctl	-0.03		v_ctl
auto_throttle_igain	I Fakto' für auto throttle Regelung	v_ctl_auto_throttle_igain	v_ctl	0.1		v_ctl
auto_throttle_dgain	D Faktor für auto throttle Regelung	v_ctl_auto_throttle_dgain	v_ctl	0		v_ctl
auto_throttle max sum err	limitierung des integrierten Fehlers	V_CTL_AUTO_THROTTLE_MAX_SUM_ERR	v_ctl	150		v_ctl
auto climb limit	Begrenzung	V_CTL_AUTO_CLIMB_LIMIT	v_ctl	0.125		vassilis
pitch max sum err	Begrenzung des integrierten Fehlers von pitch	V_CTL_AUTO_PITCH_MAX_SUM_ERR	v_ctl	100		vassilis
auto pitch pgain	P Faktor für auto pitch Regelung	V_CTL_AUTO_PITCH_PGAIN	v_ctl	-0.05		vassilis
auto pitch igain	I Fakto' für auto pitch Regelung	V_CTL_AUTO_PITCH_IGAIN	v_ctl	0.075		vassilis
auto_groundspeed_setpoint	Sollwert für Groundspeed	v_ctl_auto_groundspeed_setpoint	v_ctl	6	m/s	vassilis
groundspeed max sum err	Begrenzung des integrierten Fehlers von groundspeed	V_CTL_AUTO_MAX_GROUNDSPEED_ERR	v_ctl	100		vassilis
groundspeed pgain	P Faktor für groundspeed Regelung	v_ctl_auto_groundspeed_pgain	v_ctl	0.75		vassilis
groundspeed igain	I Fakto' für groundspeed Regelung	v_ctl_auto_groundspeed_igain	v_ctl	0.25		vassilis
auto airspeed setpoint	Sollwert für Airspeed	v_ctl_auto_airspeed_setpoint	v_ctl	14.5	m/s	vassilis
airspeed max sum err	Begrenzung des integrierten Fehlers von airspeed	V_CTL_AUTO_AIRSPEED_MAX_SUM_ERR	v_ctl	200		vassilis
airspeed igain	I Fakto' für airspeed Regelung	v_ctl_auto_airspeed_igain	v_ctl	0.05		vassilis
airspeed pgain	P Faktor für airspeed Regelung	v_ctl_auto_airspeed pgain	v_ctl	0.06		vassilis
estimator airspeec	Messwert von Pito-rohr Airspeed	estimator_airspeed	estimator			vassilis
Ml groundspeed setpoint	Sollwert für Groundspeed	Ml_goundspeed_setpoint	v_ctl	6	m/s	Ml
Ml groundspeed pgain	P Faktor für groundspeed Regelung	Ml_goundspeed_pgain	v_ctl	0.75		Ml
Ml Airspeed igain	I Fakto' für airspeed Regelung	Ml_airspeed_igain	v_ctl	0.05		Ml
Ml Airspeed pgain	P Faktor für airspeed Regelung	Ml_airspeed_pgain	v_ctl	0.06		Ml
maxAirspeed Erro'	Begrenzung des integrierten Fehlers von airspeed	MaxAirspeedError	v_ctl	200		Mi
Ml airspeed setpoint	Sollwert für Airspeed	Ml_airspeed_sdetpoint	v_ctl	14.5	m/s	Ml

8.2 Digitaler Anhang

- ARM7
  - ARM_Skript_Zhaw
  - Lpc2148_datasheet
  - Lpc2148UserManual
  - Lpc-ARM-book_srn
- Bilder
- Diverse Sicherungen
  - Bericht
  - Office Dokumente
- Encoder
  - Servo2ppm_manual_v4_2
- Flugdaten
  - Diverse Dokumente mit Auswertungen der Flugdaten
- GPS
  - U-blox6_ProtocolSpecification_Public(GPS-SW-09017)
- I2C
  - Philips Specs 39340011
- Regelung Airspeed_Matlabfiles
  - Regler
  - Nachbildung der Regelstrecke
  - Variablen
- Sensoren
  - ArduIMU
    - Datenblatt Beschleunigungsesor adx 335
    - Datenblatt Roll-Pitchsensor lpr530al
    - Datenblatt Yawsensor LY530ALH
  - Drucksensoren
    - Amsys.de.ams4711(d)
  - Magnetometer
    - HMC5843
- Software
  - Airframe
  - Arduimu_Firmware
  - Daten für AMSYS integration_Paparazzi
  - Daten für Arduimu integration Paparazzi
  - Paparazzi
  - Paparazzi Code Trickkiste
- X CTU
  - Firmware XBEE
  - XBEE Basics-Community Tutorials

    - Xbee setup
    - Xctu tutorial
- diplArbeit_Andreas Die
- einkaufsliste
- Paparazzi_Autopilot Bericht Projektarbeit SYAT_Chaneren_Schmiemi