Bachelorarbeit in
System und Automatisierungstechnik

Flugregelung einer Drohne für Forschungszwecke

Autoren: Leandro Chelini
chelilea@students.zhaw.ch
Dennis Lange
langede0@students.zhaw.ch

Dozent: Prof. Dr. Walter Siegl
siew@zhaw.ch

Auftraggeber: Oliver Ensslin
enso@zhaw.ch

Datum: 8. Juni 2011

1. Abstrakt (Deutsch)

An der ZHAW wurde eine Drohne namens UMARS, welche für atmosphärische Forschungsflüge ausgelegt ist, entwickelt. Ein Autopilot, basierend auf dem Open Source Projekt Paparazzi, wurde in früheren Projekten entwickelt, um die Nutzung der Drohne auch für Personal ohne Modellflugerfahrung möglich zu machen. In diesem Projekt wurde zunächst eine vorhandene Geschwindigkeitregelstrategie mit Hilfe einer Prandtlsonde und einem Differenzdrucksensor weiter entwickelt. Des Weiteren wurden neue, autonome Start- und Landeprozeduren erfolgreich implementiert. Um Störungen der Inertial Measurement Unit (IMU) durch die Beschleunigung beim Start zu umgehen, wurde ein Angel of Attack Sensor eingebaut, mit welchem die Fluglage bezüglich Luftströmung eindeutig bestimmt werden kann. Während dem Landeanflug ist eine genaue Messung der Höhe der Drohne notwendig. Für diese Aufgabe wurde ein Ultraschallsensor eingesetzt.

2. Abstract (Englisch)

A drone named UMARS was developed at the ZHAW for atmospheric research flights. An autopilot based on the open source project Paparazzi was developed in former projects in order to enable the use of the drone by personal with no model-flight experience. Thus, in this project an existing air-speed control strategy was further developed using now a Prandtl probe and a differential pressure sensor. New autonomous launch and landing procedures were also implemented sucessfully in this project. To avoid disturbances of the Inertial Measurement Unit (IMU) caused by the acceleration during the launch, an angle-of-attack sensor was mounted on the drone, which has improved clearly the flight attitude relativ to the air flow. An accurate measurement of the drone's height during the landing approach is necessary. An ultrasonic sensor was specified to accomplish this task.

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 der Paragraph 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.

4. Danksagung

An erster Stelle möchten wir hier Herrn Oliver Ensslin unseren Dank aussprechen. Während dieser Arbeit stand er uns immer mit Rat und Tat zur Seite. Er begleitete uns stets auf den Flugplatz, wo wir unsere Testflüge absolvieren konnten. Da wir beide, die an diesem Projekt gearbeitet haben, zu Beginn keine Erfahrungen im Modellflug hatten, benötigten wir jemanden, der unsere Übungsdrohne starten, landen und unter Umständen vor einem Absturz bewahren konnte. Dafür danken wir unserem Projektleiter, Herrn Oliver Ensslin

Des weiteren möchten wir uns bei Herrn Siegl bedanken, welcher uns mit seinen fundierten Kenntnissen in der Regelungstechnik zur Seite gestanden hat und uns durch wöchentliche Sitzungen auf dem richtigen Weg gehalten hat.

Ein weiterer Dank geht an die Paparazzi-Gemeinschaft, Herrn Bruno Neininger und an die Modellflug Gruppe Lauchetal.

Inhaltsverzeichnis

5. Einleitung

Ein grosses Projekt an der Zürcher Hochschule für Angewandte Wissenschaften (ZHAW) ist das Unmanned Modular Airborne Research System kurz UMARS. Der Kern dieses Projekts ist eine Drohne, welche an der ZHAW entwickelt wurde und welche dafür ausgelegt ist, meteorologische Forschungsflüge durchzuführen. Diese Drohne hat eine Spannweite von 5 Metern und kann eine maximale Zuladung von 10 kg bei einem Eigengewicht von 15 kg aufnehmen. In einer früheren Arbeit wurde ein Autopilot basierend auf dem Open Source Projekt Paparazzi gebaut und getestet. Dieser Autopilot sollte in dieser Arbeit verbessert und erweitert werden.

Zu Beginn der Arbeit wurden alle elektronischen Teile des Autopiloten in eine kompakte Box eingebaut, welche zwei Ziele hatte; zum einen sollte die Box einen Schutz gegen Erschütterungen und Beschädigungen bieten und zum anderen sollte so ein schnelles Ein- und Ausbauen ermöglicht werden. Danach sollte die vorhandene Regelung des True Airspeeds erweitert werden. Diese war bis vor dieser Bachelorarbeit so implementiert, dass die Höhe der Drohne über das Höhenruder und die Geschwindigkeit gegenüber der Luft über die Motorenleistung geregelt wurde. Diese Regelung hat den Nachteil, dass die Drohne bei einem Motorenausfall solange an Geschwindigkeit verliert, bis sie in den Strömungsabriss gerät und unkontrolliert an Höhe verliert. Dieses Problem kann umgangen werden, indem die Höhe über die Motorenleistung und die Geschwindigkeit über das Höhenruder geregelt wird. Sollte es nun zu einem Motorenausfall kommen, würde die Drohne immer noch an Höhe verlieren, da aber die Geschwindigkeit auf dem Sollwert gehalten werden kann, besteht keine Gefahr eines unkontrollierbaren Strömungsabrisses.

Um die Drohne nun auch für Forscher ohne Modellflugerfahrungen anwendbar zu machen, sollte in einem nächsten Schritt ein autonomer Start entwickelt werden. Bei diesem Start wird die Drohne an ein gespanntes Gummiseil gehängt, welches per Fusspedal ausgelöst wird. Danach wird die Drohne so stark beschleunigt, dass sie abheben und einige Meter fliegen kann. Erst wenn sichergestellt ist, dass sich das Gummiseil vom Haken an der Drohne gelöst hat, wird der Motor eingeschaltet und die Drohne steigt, bis sie ihre vorgegebene Endhöhe erreicht hat. Danach wird der nächste Punkt des Flugplans ausgeführt. Eine weitere Aufgabe war es, einen Algorithmus zu entwickeln, mit welchem die Drohne ohne Eingreifen der manuellen Steuerung gelandet werden kann. Dazu ist es aber nötig, die genaue Höhe der Drohne beim Landeanflug zu bestimmen, und es musste ein Messverfahren gefunden werden, mit welchem diese Aufgabe zuverlässig und

mit möglichst wenig Aufwand gelöst werden kann. Nachdem mehrere Verfahren untersucht wurden, fiel die Wahl auf einen Ultraschallsensor mit einer Reichweite von ca. 7.5 Metern. Die Landung wurde nun so umgesetzt, dass zu Beginn des Landeanflugs die Höhe nach GPS geregelt wird, bis der Ultraschallsensor eine zuverlässige Höhe liefert. Sobald dies der Fall ist, wird der Höhenregelung diese Höhe zur Weiterverarbeitung übergeben. Nun stellte sich noch die Frage nach einem geeigneten Regelsystem für die Landung. Die zu anfangs verfolgte Kombination aus Höhenregelung und Geschwindigkeitsregelung wurde im Verlauf der Arbeit verworfen und es wurde eine Kombination aus Höhen- und Anstellwinkelregelung implementiert. Mit diesem Regelsystem lässt sich die Drohne auch bei kleinen Geschwindigkeiten stabil fliegen. Ausserdem ist es möglich, den Anstellwinkel kurz vor dem Aufsetzen auf der Landebahn sukzessive zu erhöhen, wodurch die Geschwindigkeit nochmals verringert wird und die Drohne sanft auf dem Boden aufsetzt.

Während der ganzen Arbeiten wurden regelmässig Testflüge gemacht, an welchen Neues getestet und das Regelsystem parametriert wurde. Ein solcher Testflug wurde auf dem Flugplatz der Modellflug Gruppe Lauchetal durchgeführt und dieser Testflug wurde vorher mit Hilfe eines Flugplans (siehe Anhang B.1) geplant, um auf dem Flugfeld keine Zeit zu verlieren. Die Regelparameter, welche während eines solchen Fluges ermittelt wurden, wurden auf einem Flugprotokoll (siehe Anhang B.2) festgehalten.

5.1. Aufgabenstellung

MET
W. Siegl

System- & Automatisierungstechnik

Bachelorarbeit: Geregelter Flug einer kleinen Drohne für Forschungszwecken

Industriepartner: IMES/ZHAW
Hr. O. Ensslin

Studenten: Dennis Lange
Leandro Chelini

Beginn: 14. Februar 2011
Abgabe: 10. Juni 2011

Ausgangslage: In früheren Projekt- und Bachelorarbeiten an der ZHAW wurde für ein Modellflugzeug Typ "Maja" eine Flugzeugstabilisierung sowie eine Flughöhen- und Fluggeschwindigkeitsregelung praktisch implementiert und getestet. Dieses System erlaubt einen autonomen stabilen Flug mit geregelter Geschwindigkeit entlang einer vorgegebenen Trajektorie.

Ziel der Arbeit: Das definitive Konzept für die Fluggeschwindigkeitsregelung soll zunächst aufgrund objektiver Kriterien gewählt werden. Ein autonomes Start- und Landungssystem soll entwickelt und getestet werden. Bei diesen Aufgaben soll der notwendige C-Code von Paparazzi in einem Simulink-Programm veranschaulicht werden. Die erarbeitete Software soll für die Portierung auf den UMARS-Demonstrator vorbereitet werden. Dazu gehört eine Überarbeitung der Regelsysteme, da die aerodynamischen Eigenschaften der Drohnen unterschiedlich sind.

Aufgaben:

1. Implementierung und Test der gewählten Variante für die Fluggeschwindigkeit-Regelung. Falls hier noch verschiedene Varianten konkurrieren, dann sollen Ihre Vor- und Nachteile durch Bewertungskriterien klar dargestellt werden.

2. Konzeptentwicklung und praktische Verifikation für ein autonomes Start- und Landungssystem.

3. Simulink-Übersetzung der C-Routinen aus dem Paparazzi-Code welche für die Flugregelung notwendig sind. Das Ziel hier ist mindestens eine qualitative Information über die Wirkung der verwendeten Reglerparameter.

4. Für eine Messkampagne, welche im Sommer in Frankreich statt findet, sollen verschiedene Flugpattern entwickelt und implementiert werden.

5. Vorbereitung für die Portierung des Konzeptes auf die UMARS.

Betreuer: Prof. Dr. W. Siegl
Tel: 058 934 7534
walter.siegl@zhaw.ch

BA_FS11_Flugregelung 1 Siew-14.04.2011

5.2. Anforderungen an die Flugregelung

Die folgende Anforderungsliste dient als erster Entwurf für die späteren Anforderungen an die UMARS. Da jedoch in dieser Arbeit das Hauptgewicht auf den neuen Start- und Landealgorithmen lag, sind die einzelnen Parameter der Anforderungsliste noch nicht definitiv festgelegt.

5.2.1. Einleitung

Im diesem Unterkapitel werden die Regelanforderungen und Toleranzen der Regelsysteme einer Drohne definiert. Die Positionsregelung im dreidimensionalen Raum wird bei der Flugregelung in zwei Systeme unterteilt: Höhenregelung und Positionsregelung. Dies ist so, weil eine Änderung der Sollhöhe einen direkten Einfluss auf die Geschwindigkeit hat. Die Positionierung im Raum wird mit drei verschiedenen Systemen geregelt. Die drei Regelsysteme sind:

- Höhenregelung
- Positionsregelung auf einer konstanten Höhe
- Geschwindigkeitsregelung

Die Anforderungen werden hauptsächlich in vier Gruppen unterteilt:

1. Start
2. Anflug
3. Messung
4. Landung

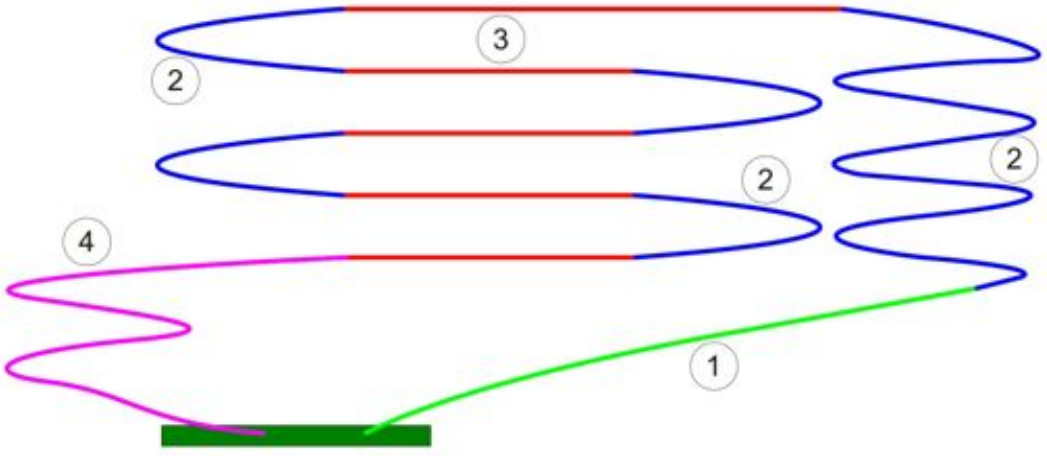


Abbildung 5.1.: Trajektorie eines kompletten Messfluges

5.2.2. Anforderungen beim Start (1)

Die Drohne wird mittels eines Gummiseils (Bungee) gestartet. Treten während des Startprozederes Fehler auf, soll, wenn möglich der Startvorgang sicher abgebrochen werden können, indem auf manuelle Steuerung umgeschaltet wird. Der Startvorgang soll schlussendlich ohne die Hilfe einer manuellen Steuerung durchführbar sein. Das Starten soll bei einer Windstärke von bis zu 10 m/s möglich sein. Jeder Start hat entgegen der Windrichtung zu erfolgen.

5.2.3. Regelanforderungen während des Anflugs (2)

Um die Drohne für eine gerade Messstrecke auszurichten, sind grössere Kursänderungen notwendig. Während diesen Kursänderungen wird jedoch keine Messung durchgeführt. Daher kann die Kursänderung auch ruckartig erfolgen, wichtig ist jedoch, dass möglichst wenig Energie für den Anflug aufgewendet wird.

Toleranzen

- Höhenregelung
  - Keine besonderen Anforderungen
- Positionsregelung
  - Keine besonderen Anforderungen
- Geschwindigkeitsregelung (True Airspeed)
  - keine extremen Ausschläge der Servomotoren
  - Bereichsüberwachung für die Istgeschwindigkeit

5.2.4. Regelanforderungen während der Messung (3)

Messdaten werden nur auf den geraden Strecken des vorgegebenen Kurses erfasst.
Bei der Einstellung der verschiedenen Regelsysteme wird darauf geachtet, dass die folgende Prioritätenliste eingehalten wird:

1. keine grossen Ausschläge der Servomotoren (maximalen Querlage der Drohne ca. 10°)
2. Höhenregelung
3. Regelung der horizontalen Position
4. True Airspeed-Regelung

Diese Prioritätenliste gilt natürlich nur so lange, wie alle Regelgrössen im zulässigen Bereich sind. Sollte beispielsweise die Geschwindigkeit zu stark abnehmen, muss diese wieder in den zulässigen

Bereich gebracht werden, auch wenn dies bedeutet eine Abweichung der Höhe in Kauf nehmen zu müssen.

Toleranzen

- Höhenregelung
  - ± 10 m
- Positionsregelung
  - ± 20 m
- Geschwindigkeitsregelung (True Airspeed)
  - ± 5 m/s

Sollte nun der Fall auftreten, dass eine Regelgrösse aufgrund einer Begrenzung einer Stellgrösse nicht im zulässigen Bereich gehalten werden kann, wird diese Begrenzung entsprechend angepasst. Zu Erklärung soll folgendes Beispiel dienen:
Beim Flug auf der Messstrecke treten aufgrund von Seitenwind Abweichungen in der Positionsregelung von mehr als 20 m auf. Durch die Begrenzung der maximalen Schräglage ist es jedoch nicht möglich den Kurs zu halten. In diesem Fall wird die maximale Schräglage auf z.B. 15° erweitert.

5.2.5. Anforderungen der Landung (4)

Eine erfolgreiche Landung ist, wenn die Drohne unbeschädigt bleibt und ohne Reparaturen wieder gestartet werden kann. Ein Landevorgang soll schlussendlich komplett über das GCS (Ground Control Station) am Notebook durchgeführt werden können. Bei fehlerhaften Landeanflügen aufgrund von Windböen oder Ähnlichem soll die Landung mittels automatischer Fehlerbehandlung abgebrochen werden und die Drohne soll wieder um einen Standbypunkt kreisen. Eine autonome Landung soll gegen den Wind bei einer Windstärke von maximal 15 m/s mit einem maximalen Winkel zur Landebahn von 30° möglich sein.

5.2.6. Überprüfung der Regelgenauigkeit

Um eine Aussage über die Güte der Regelung machen zu können, wurde ein Modul geschrieben, welches die Fehlerquadratsumme der jeweiligen Regelabweichung berechnet und ausgibt. Anhand des so gezeichneten Verlaufs und des Endwerts der Fehlerquadratsumme können objektive Beurteilungen über den Einfluss der Regelparameter gemacht werden.

6. Neue Kompaktbauweise des Autopiloten

Abbildung 6.1.: Kompaktbauweise des Autopiloten

Bei der Vorgängerdrohne des Typs `MAJA` von Borjet wurde eine offene Bauweise gewählt. Alle Elektronikelemente und Kabel lagen frei im Innern des Flugzeuges. Dies hatte zur Folge, dass nebst den Softwareproblemen auch häufig Defekte in der Hardware auftraten. So konnte es vorkommen, dass auf dem Flugfeld wegen eines Kabelbruchs die USB-Verbindung nicht hergestellt werden konnte und somit das Flashen unmöglich war. Aus diesen Gründen wurde die gesamte Elektronik inklusive der Drucksensoren in ein Kästchen gebaut. In der Abbildung 6.1 wird das Kästchen von aussen dargestellt. Die Abmasse sind 119 mm x 65 mm x 40 mm (l x b x h). Oben auf dem Deckel kann die GPS-Antenne erkannt werden.

6.1. Übersicht

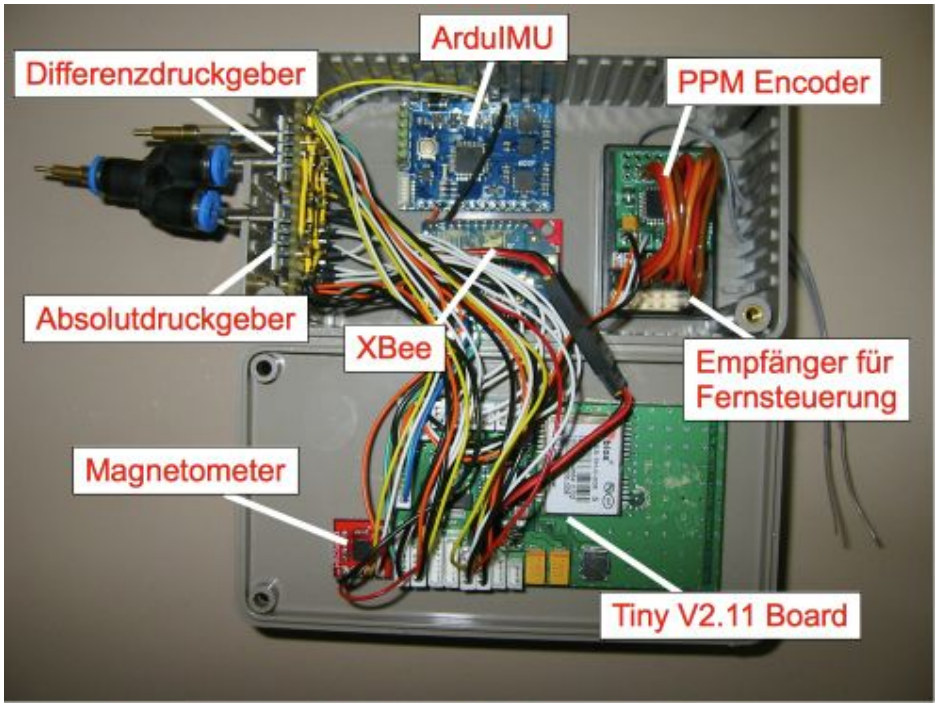


Abbildung 6.2.: Innenarchitektur des Autopiloten

Die Bauteile wurden mit doppelseitigem Klebeband befestigt, was einen sehr guten Halt gewährleistet, jedoch bei einem Umbau ersetzt werden muss. Auf der linken Seite der Abbildung 6.2 können die Drucksensoren erkannt werden, welche über eine Platine direkt an den I2C Kanal angeschlossen sind. Im Deckel der Kiste (unten) ist die Hauptplatine des Typs `Tiny` verbaut, welche den eigentlichen Autopiloten darstellt. Links neben dem `Tiny Board` wurde der Magnetometer (3D Kompass) platziert, welcher ebenfalls über den I2C Kanal mit dem System verbunden ist. Die Ausrichtung des Magnetometers ist definiert. Die Angaben zur Ausrichtung sind direkt auf dem Chip ersichtlich. Die `Inertial measurement Unit` namens `ArduIMU` wurde auf dem Boden der Kiste ganz am Rand platziert, da die Ausrichtung bei diesem Messinstrument auf die Funktionalität einen sehr grossen Einfluss hat. Die Angaben der Ausrichtung des `ArduIMUs` sind auf der Rückseite des Chips eingebrannt. Das Funksignal der Fernsteuerung wird vom Empfänger (rechts) empfangen und in PWM-Signale umgewandelt. Diese werden dann an denn PPM Encoder (Chip auf dem Empfänger) weitergeleitet, welcher die Signale zu einem PPM-Signal für das `Tiny Board` aufbereitet. Links neben dem Empfänger wurde das Funkmodem `XBee` befestigt. Auf dem Bild wird der grösste Teil des Chips verdeckt. Weitere Informationen über den XBee sind im Kapitel 17 enthalten.

6.2. Verdrahtung

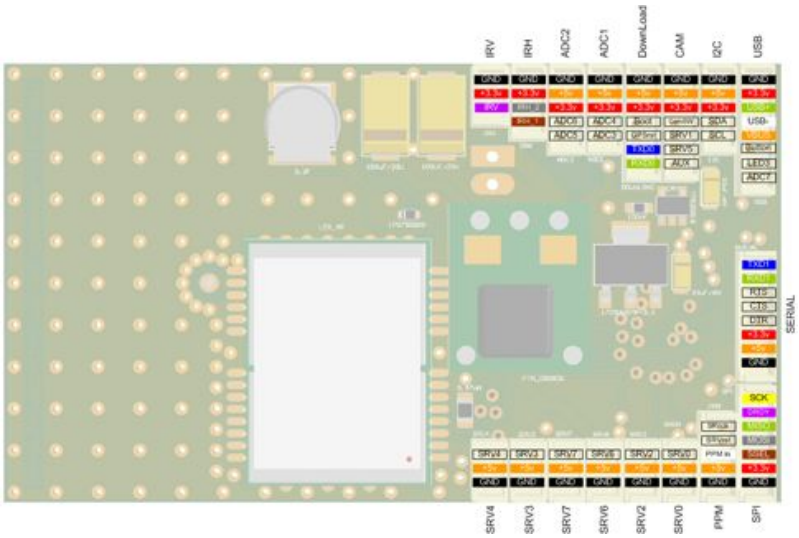


Abbildung 6.3.: Übersicht der Anschlüsse auf dem Tiny Board

In der Abbildung 6.3 ist die Pinbelegung des `Tiny Boards` ersichtlich. Alle nötigen Anschlüsse inklusive der Stromversorgung wurden mittels Stecker an das Board angeschlossen, was einen Ausbau des kompletten Boards ermöglicht, ohne zu löten. Die kleinen Stecker sind vom Typ `PicoBlade` (Abbildung 6.4). Ein grosser Nachteil dieser Stecker ist die Verbindung des Kontaktes (Abbildung 6.5) mit der Litze. Die einzelnen Kontakte mit einer Grösse von ca. 4 mm werden mit einer Spezialzange an die feine Litze gecrimpt, was nur eine sehr schwache Verbindung gewährleistet. Oft geschah es, dass bei Manipulationen im Kästchen, sich die eine oder andere Verbindung löste. Der einzige Trick beim Crimpen ist, darauf zu achten, dass die hintersten zwei Bügel an die Isolation der Litze gepresst werden.

Abbildung 6.4.: PicoBlade Stecker

Abbildung 6.5.: PicoBlade Kontakt

6.3. Pinbelegung

Abbildung 6.6.: Vorderseite der Kompaktbauweise

Unterhalb der Drucksensoren wurden diverse Anschlüsse aus der Box herausgeführt. In der Abbildung 6.6 ist ersichtlich, dass sogenannte Jumper verwendet wurden, um unterschiedliche Anschlussarten voneinander zu trennen. Falls zukünftig ein weiterer Anschluss benötigt wird, können die Jumpersteckplätze ebenfalls verwendet werden. Die Pinbelegung wurde so gewählt, dass bei einer Fehlplatzierung eines Steckers möglichst wenig beschädigt wird. Die negativen Pole sind alle in der unteren Pin-Reihe angeordnet, wobei die Speisungen ausser beim USB in der mittleren Reihe platziert wurden. Ganz links wird das USB-Verbindungskabel eingesteckt. Vom 6-Pin Stecker des USB-Kabels wurden nur die oberen und unteren zwei Pins belegt. Bei den Anschlüsse ADC3 und ADC4 wird eine 3.3V Speisung am mittleren Pin ausgegeben, wobei am oberen Pin das Eingangssignal erwartet wird. Die analogen Eingänge ADC5 und ADC6 stellen eine 5V Speisung zur Verfügung. Die Speisung der ADCs wurde so gewählt, dass sie zu der maximalen Eingangsspannung passt. Das Kästchen wird mit einer Spannung von ungefähr 12 V betrieben, welche am Power-Eingang angeschlossen werden. Die Servoausgänge SRV0, SRV2, SRV4, SRV6 und SRV7 werden über die Spannung von der Motorsteuerung SRV3 gespeist. Diese Speisung wird von der Motorenreglereinheit bereitgestellt. Die Ausgänge wurden so konfiguriert, dass SRV5 und SRV6 das Höhenruder und Seitenruder und SRV0 und SRV2 die Querruder ansteuern. Die Kabel wurden am Flugzeug nummeriert, um das Platzieren zu erleichtern.
Der Servoausgang SRV7 ist noch frei. Er wurde teilweise für das Ein- und Ausschalten der Flugzeugbeleuchtung verwendet.
Am linken Anschluss des Drucksensors, welcher in der Abbildung 6.6 ersichtlich ist, wird der Staudruck, gemessen über die Prandtlsonde, angeschlossen. An der Y-Druck-Verzweigung gleich daneben wird der Referenzdruck (Umgebungsdruck) angeschlossen. Nähere Inforamtionen über die Drucksensoren stehen im Kapitel 8.

7. Praktische Arbeiten

Diese Arbeit erforderte neben dem grossen Programmieraufwand auch viele praktische Arbeiten im Bereich des Modellbaus. In die eigens für diese Arbeit angeschaffte Übungsdrohne "Mentor" musste nicht nur der komplette Autopilot eingebaut werden, sondern auch ein Ultraschallsensor, eine Prandtlsonde, eine Angle of Attack Messvorrichtung und nicht zu Letzt eine Lichtanlage, welche sich aber eher als Spielerei herausstellte. Die nicht immer reibungslos verlaufenden Testflüge, zogen ausserdem einige Reparaturen mit sich.

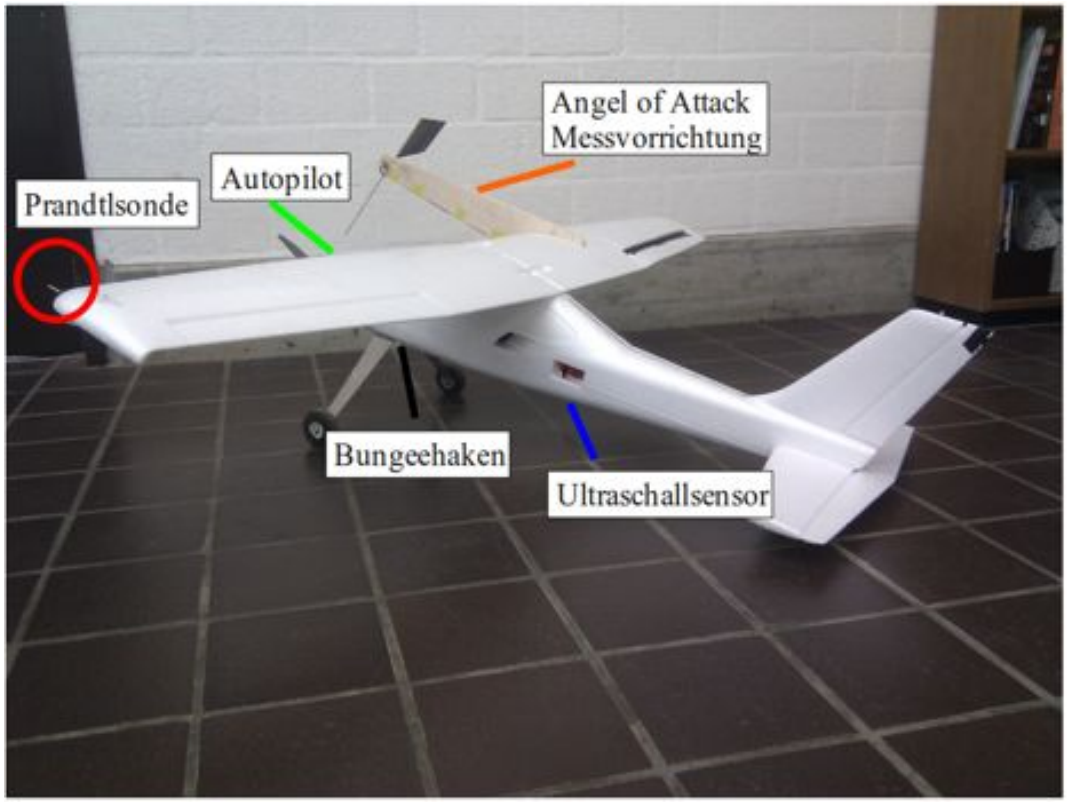


Abbildung 7.1.: Übungsdrohne Mentor Übersicht

Die Abbildung 7.1 zeigt eine Übersicht über die Positionen, an denen die entsprechenden Komponenten eingebaut wurden. Die Drohne selber und die einzelnen Komponenten sollen im Folgenden genauer beschrieben werden.

7.1. Beschreibung der Übungsdrohne

Bei der Übungsdrohne handelt es sich um einen Mentor der Firma MULTIPLEX, der zum grössten Teil aus Styropor gefertigt wurde. Diese Tatsache löste zu Beginn der Arbeit einige Unsicherheiten bezüglich Stabilität aus, doch es zeigte sich, dass die Konstruktion erstaunlich widerstandsfähig ist und dass sich kleine bis grössere Schäden problemlos mit Sekundenkleber beheben liessen. Bei dem Modell selber handelt es sich um einen Hochdecker, welcher eigentlich für den konventionellen Modellflug entwickelt wurde. Die Modifikationen, welche für ein autonomes Fliegen notwendig waren, sind unter Punkt 7.2 ff. beschrieben.

7.1.1. Technische Daten

Die Technischen Daten des Mentors sind:

Spannweite	1630 mm
Rumpfweite	1170 mm
Fluggewicht	ca. 2000 g
Flächeninhalt	ca. 45 dm^2
Flächenbelastung	ca. 44.5 g/dm^2
RC-Funktionen	Seiten-, Höhen-, Querruder, Motor

Tabelle 7.1.: Technische Daten des Mentors

Bei dem Motor handelte es sich um einen `Himax C 3528-1000` mit einem `CC PHOENIX 45`-Regler und einem `11x5,5"`-Propeller. Der Motor musste jedoch im Verlauf der Arbeit durch einen Motor, dessen Spezifikationen in derselben Grössenordnung liegen, welche jedoch nicht genauer bekannt sind, ersetzt werden. Er stammte von der früheren Übungsdrohne "Maja".

7.2. Notwendige Komponenten für den Autopiloten

Um den Mentor für den Autopiloten steuerbar zu machen, waren die folgenden Komponenten notwendig.

7.2.1. Grundlegende Elektronik

Damit die Drohne autonom gesteuert wird, ist die komplette Paparazzi-Hardware nötig. Diese wurde zu Beginn der Arbeit in eine kompakte Box verbaut (siehe Kapitel 6). Diese Box kann nun schnell und einfach eingebaut werden. Die Position, an welcher sie verbaut wurde, ist in Abbildung 7.1 ersichtlich.

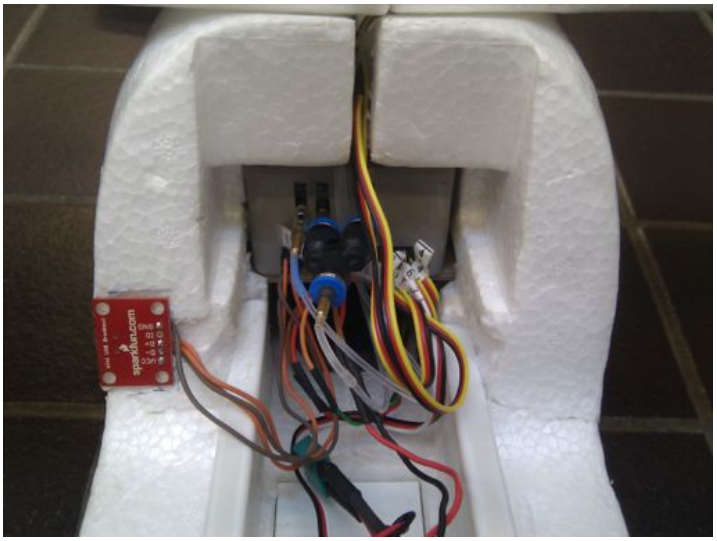


Abbildung 7.2.: Eingebaute Autopilot-Box

Abbildung 7.2 zeigt die eingebaute Box des Autopiloten im Mentor. Beim Einbau ist darauf zu achten, dass alle Servomotoren, der Ultraschallsensor, die Druckschläuche der Prandtlsonde und nicht zuletzt die Energieversorgung richtig angeschlossen werden, da die Elektronik sonst beschädigt werden könnte.

7.2.2. Prandtlsonde

Die Prandtlsonde wird zur Bestimmung des True Airspeeds benötigt. Damit die Sonde nicht von den Wirbeln des Motors beeinflusst wird, was die Messungen verfälscht, wurde sie ganz aussen an der linken Tragfläche befestigt.

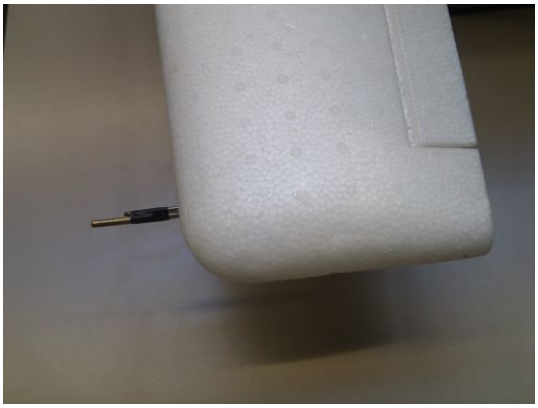


Abbildung 7.3.: Position der Prandtlsonde am Mentor

Abbildung 7.3 zeigt die Position der Prandtlsonde am linken Flügel des Mentors. Zur stabilen Befestigung war es nötig, einen Draht in den Flügel einzubringen, an welchem die Sonde mittels Klebeband befestigt wird. So ist es auch möglich, diese mit wenig Aufwand auszutauschen. Die Sonde wurde von unten her in den Flügel eingebracht dort, wo auch der Kanal für die Druckschläuche eingefräst wurde, siehe Abbildung 7.4. Bei den Druckschläuchen sollte kein weiches Material zum Einsatz kommen, da diese keinen Schutz gegen Druckstösse oder Vibrationen vom Motor bieten. Fehlt dieser Schutz, kann es zu Messfehlern kommen.

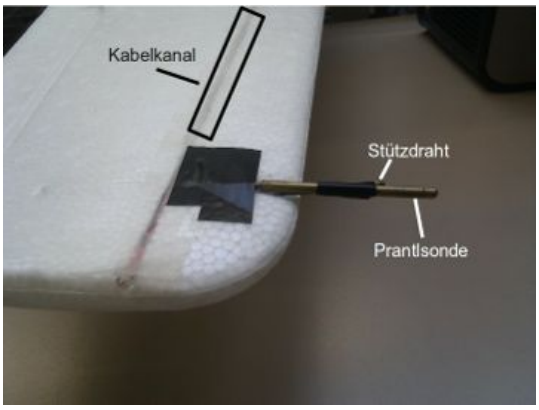


Abbildung 7.4.: Befestigung der Sonde, Kabelkanäle

7.2.3. Ultraschallsensor

Der Ultraschall, welcher für die genaue Höhenmessung vor der Landung verwendet wird, wurde so am Mentor platziert, dass er möglichst vor Beschädigungen geschützt ist . Ausserdem muss berücksichtigt werden, dass der Sensor einen gewissen Messkegel aufweist. In diesem Kegel dürfen sich also keine Teile des Flugzeuges befinden. Dies würde zu Messfehlern führen. Genauere Angaben zum Sensor sind unter Punkt I.2 zu finden.

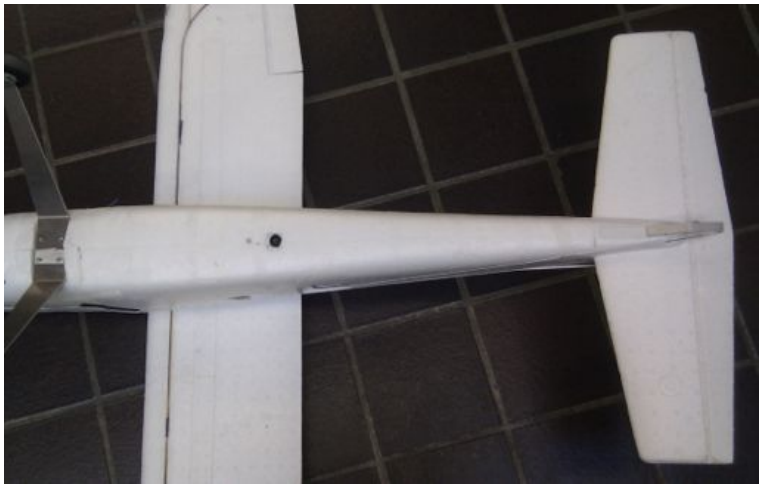


Abbildung 7.5.: Position des Ultraschallsensors (Ansicht von unten)

In Abbildung 7.5 ist die genaue Position des Ultraschallsensors zu sehen.

7.2.4. Bungeehaken

Der Bungeehaken dient der Befestigung des Gummiseils an der Drohne. Was die Position dieses Hakens angeht, ist es wichtig, dass er auf der Längsachse und in Flugrichtung vor dem Schwerpunkt der Drohne montiert wird. Ausserdem ist es wichtig, dass er geometrisch so geformt ist, dass sich das Seil problemlos lösen kann. Der Haken sollte ausserdem gut befestigt werden, da er zu Beginn des Starts eine hohe Kraft auf die Drohne übertragen können muss. Beim Mentor konnte der Haken in die Befestigung des vorderen Fahrwerks integriert werden (Abbildung 7.6).

Abbildung 7.6.: Postition des Bungeehakens (Ansicht von unten)

7.2.5. Angle of Attack Messvorrichtung

Die Angle of Attack Messvorrichtung setzt sich aus drei Komponenten zusammen: der Halterung, dem Drehgeber und der Messfahne (siehe Abbildung 7.7). Die Halterung ist aus leichtem, aber dennoch robustem Balserholz gefertigt. Sie wurde direkt an die Tragfläche geklebt. Bei dem Drehgeber handelt es sich um einen `Miniature Absolute Magnetic Shaft` Encoder der Firma US Digital, dessen Datenblatt im Anhang I.3 zu finden ist. Die Messfahne ist eine Eigenproduktion und besteht aus einem dünnen Blech und einer Stange. Das Blech wurde gefaltet und an die Stange geklebt. Die Stange sollte genau im Schwerpunkt an dem Drehgeber befestigt werden, damit die Messvorrichtung ausbalanciert ist und keine Messfehler entstehen.

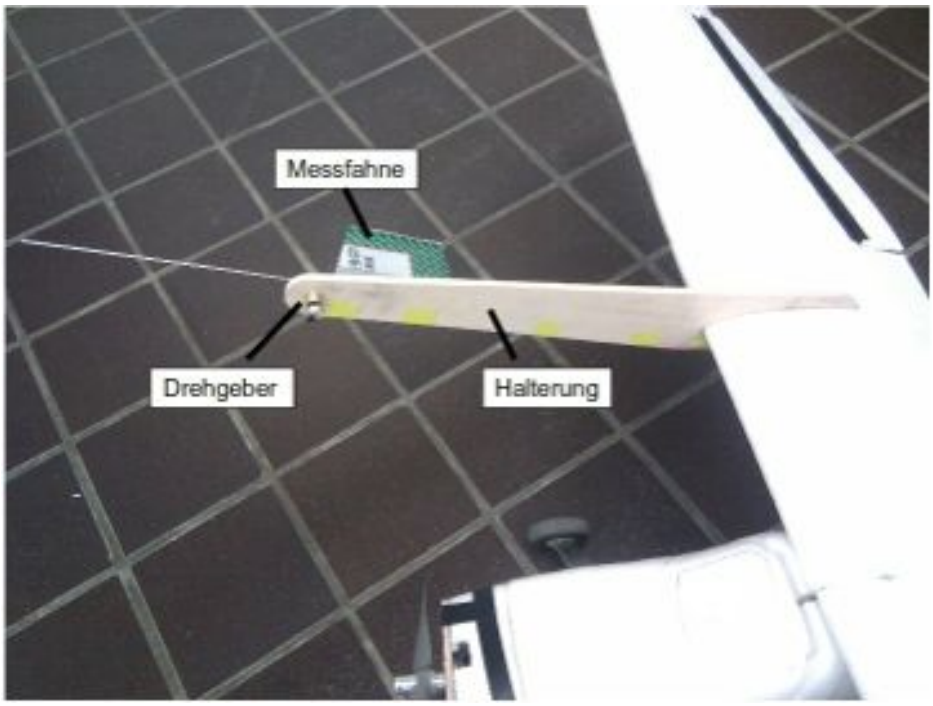


Abbildung 7.7.: Angle of Attack Messvorrichtung

7.2.6. Lichtanlage

Es wurde ebenfalls eine Lichtanlage am Mentor montiert, welche über die Fernbedienung ein- und ausschaltbar ist. Es handelt sich bei dieser Anlage um ein für den Modellflug bestimmtes LED-System mit fünf Leuchten in den Farben Grün, Rot und Weiss. Werden diese LEDs korrekt angebracht, ist es auch im Dunkeln möglich, die Flugrichtung und die Lage des Flugzeuges zu erkennen.

7.2.7. Reparaturen

Die Testflüge liefen nicht immer wie geplant und daher mussten einige Reparaturen am Mentor ausgeführt werden. Bei dem Test der Landemethode, welche im Paparazzi-Projekt bereits vorhanden war, ist der Motor beschädigt und fing daher bei hohen Drehzahlen an zu schwingen. In der Vorgängerdrohne Maja war glücklicherweise noch der Motor eingebaut. Dieser wurde ausgebaut und konnte im Anschluss in den Mentor eingebaut werden. Auch das Gelenk eines Querruders hatte bei einem missglückten Nachtflug Schaden genommen. Dieses wurde mit Hilfe von Scharnierstiften wieder in Gang gebracht.
Die grösste Reparatur musste jedoch nach einem stark missglückten Startversuch durchgeführt werden, bei dem der Mentor in zwei Hälften zerbrach und der Motorenflansch komplett zerstört wurde. Doch da sich Styropor extrem gut kleben lässt, konnte die Drohne noch auf dem Feld wieder zusammengeklebt werden und eine eilig herangeholte Sperrholzplatte, zu sehen in Abbildung 7.8, ersetzte den zerstören Flansch. Danach war der Mentor wieder flugtauglich und die Tests konnten fortgesetzt werden.

Abbildung 7.8.: Motorenflansch aus Holz

8. Druck Sensoren - AMSYS

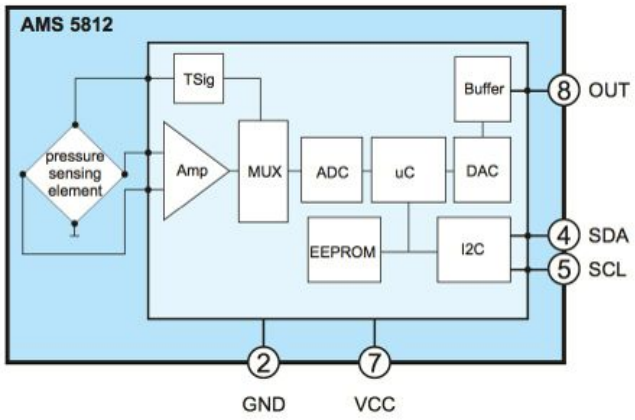


Abbildung 8.1.: Funktionsprinzip AMS5812

Für die Messung des barometrischen Drucks und des Differenzdrucks wurden Sensoren von AMSYS verwendet. Diese verwenden das serielle `I2C`-Protokoll, um die Daten zu übermitteln. Jeder Sensor verfügt über einen Temperatursensor, welcher ebenfalls über die `I2C`-Schnittstelle abgerufen werden kann.

Die Entscheidungsgründe, welche für die AMSYS-Sensoren sprachen, waren:

- hohe Auflösung
  - Differenzdrucksensor: 1267.6 Einheiten pro mbar
  - Absolutdrucksensor: 25.3 Einheiten pro mbar
- genaue Temperaturkompensation
- kostengünstig
- Analog- und I2C-Ausgang

8.1. Differenzdrucksensor

Abbildung 8.2.: AMSYS Sensor AMS5812

Für den Differenzdruck wurde der AMS 5812-0003-D gewählt.
Um den True Airspeed zu messen, wurde am linken Flügelende eine Prandtlsonde installiert. Diese Sonde ermöglicht das Messen des statischen Drucks und des Staudrucks. Aus dem Differenzdruck dieser beiden Drücke lässt sich die Geschwindigkeit gegenüber der Luft ermitteln. Um diesen Differenzdruck zu ermitteln, kam ein unidirektionaler Differenzdrucksensor der Firma AMSYS zum Einsatz. Die kompletten Messdaten und die Temperaturwerte werden in vier Byte-Pakete über die I2C Schnittstelle zur Verfügung gestellt. Um die Daten abzurufen und umzurechnen wurde das Modul `AMSYS_Airspeed`, welches im Abschnitt 15.4.1 beschrieben wird, geschrieben.

8.1.1. Windkanal

Um den Differenzdrucksensor und die Prandtlsonde zu kalibrieren und zu testen, musste eine Situation geschaffen werden, in welcher es möglich war, die gemessenen Daten mit einer Referenz zu vergleichen. Um einen reproduzierbaren Versuch durchzuführen, wurde die Prandtlsonde auf einen Aluminiumständer montiert und in einen Windkanal gestellt. Dazu wurde der Windkanal der ZHAW im TM-Gebäude verwendet.

- Δp inst. $\widehat{=}$ Δp der im Windkanal installierten Sonde
- Rohdaten $\widehat{=}$ Daten, welche direkt über die I2C Schnittstelle gelesen werden
- Δp Prandtl. $\widehat{=}$ zu testende Prandtlsonde
- Wind-geschw. $\widehat{=}$ Von Autopilot errechnete Windgeschwindigkeit

Die Tabelle 8.1 zeigt, dass der Differenzdruck bei höheren Windgeschwindigkeiten eine Affinität zu den gemessenen Werten der fest installierten Sonde des Windkanals aufweist. Bei höheren Windgeschwindigkeiten werden, wie in der Abbildung 8.3 ersichtlich, immer konstantere Werte gemessen. Bei tiefen Windgeschwindigkeiten wird der Druck somit viel schlechter gemessen als bei hohen. Da das Flugzeug jedoch eine Mindestgeschwindigkeit aufweist, welche deutlich über

Messnummer	Ventilatorleistung [%]	Δp inst. [Pa]	Rohdaten [-]	Δp Prandtl. [Pa]	Wind-geschw. [m/s]
1	30	4.83	3366	~6.8	~2.8
2	40	8.99	3415	~10.4	~3.5
3	50	14.7	3479	~16.7	~4.8
4	60	21.4	3560	~22	~5.7
5	70	29.7	3650	~30	~6.8
6	80	39.1	3760	~39	~7.8
7	90	50.2	3900	~49	~8.9
8	100	62.7	4050	~61.5	~10.0

Tabelle 8.1.: Vermessung der Prandtlsonde im Windkanal

den schlecht zu messenden Windgeschwindigkeiten liegt, sollte dies kein Problem darstellen. Jedoch muss beachtet werden, dass der Windkanalversuch mit idealen Bedingungen durchgeführt wurde. Das Ströhmungsbild der Luft, welches auf die Prandtlsonde wirkte, war laminar, was in der Realität sehr selten der Fall ist.

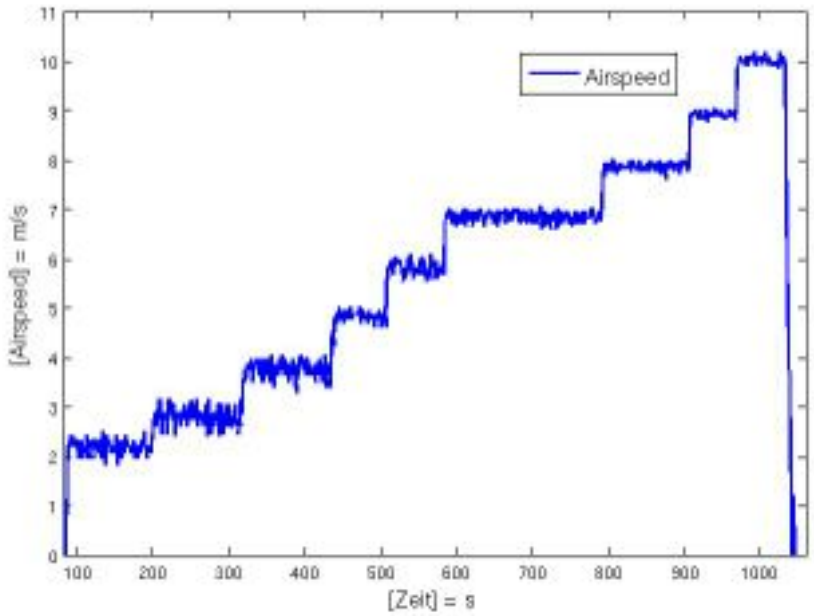


Abbildung 8.3.: Geschwindigkeitsmessung mit Prandtlsonde im Windkanal

Nach dem Versuch im Windkanal wurde ein weiterer mit Pressluft durchgeführt, um festzustellen, ob der Sensor durch die Motorenvibrationen oder den dabei hervorgerufenen magnetischen Feldern beeinflusst wird. Der Versuch ergab, dass der Differenzdrucksensor bei 100% Motorleistung einen vernachlässigbaren kleinen Unterschied macht. Bei dem Pressluftversuch resultierten ähnliche Graphen wie bei einem Testflug.

8.1.2. Filter

Nach dem Windkanalversuch und den ersten Flugtests stand fest, dass die Daten nicht ungefiltert verwendet werden können. Aus diesem Grund wurde nach einem passenden Filter gesucht. Ausgewählte Filter wurden dann auch implementiert und getestet.

Mittelwert Filter

Dieser Filter speichert alle Sensorwerte in ein `Array` und summiert sie. Danach wird die Summe durch die Anzahl der gespeicherten Werte geteilt. Die vorhandenen Module, welche benutzt wurden, um den True Airspeed mittels einem Analogsensor zu erfassen, arbeiteten mit einem sogenannten Mittelwert-Filter. Der Nachteil an diesem Filter ist, dass der benötigte Speicher proportional zur Anzahl der gespeicherten Werte steigt. Ein weiterer Nachteil ist, dass die maximale Filterrate fest kompiliert wird, was bei einer Änderung ein erneutes Flashen erforderlich macht.

PT1 Filter

Der PT1 Filter nimmt je nach Filterrate einen bestimmten Anteil der neuen Werte und addiert diesen zum einen Anteil der alten Werte.
Im folgenden Beispielcode werden 80% der alten Werte (`airspeed_old`) zu 20% des neuen Wertes (`airspeed_new`) addiert, wenn als Filterrate in der Variabel `airspeed_filter` der Wert 0.8 steht.

```
airspeed_amsys = airspeed_filter * airspeed_old + (1 - airspeed_filter)
* airspeed_new;
airspeed_old = airspeed_amsys;
```

Kalmanfilter

Am Anfang der Filtersuche war der Kalmanfilter einer der Favoriten. Jedoch wurde schnell erkannt, dass dieser Filter für die Anforderungen viel zu aufwändig ist. Da keine theoretischen Kenntnisse über diesen Filter vorhanden waren, wurde versucht, das fehlende Wissen mit verschiedenen Lektüren und Dokumentationen zu ergänzen. Schlussendlich wurde beschlossen, dass der Kalmanfilter nicht weiter beachtet wird und gegebenenfalls nur bei der barometrischen Höhenmessung Verwendung findet.

Sawitzky-Golay-Filter

Der Savitzky-Golay Filter ist eine Vorgehensweise für das Glätten von Zeitreihen. Dabei wird jedem Wert der Serie ein neuer Wert zugewiesen. Dieser neue Wert wird durch eine Polynomialanpassung mit 2n+1 Nachbarpunkten (inklusive dem zu glättenden Punkt) berechnet. Dabei ist n grösser oder gleich der Ordnung des Polynoms. Nach Savitzky und Golay sind die Koeffizienten des Polynoms konstant, was eine relativ einfache programmatische Umsetzung des Filters erlaubt.

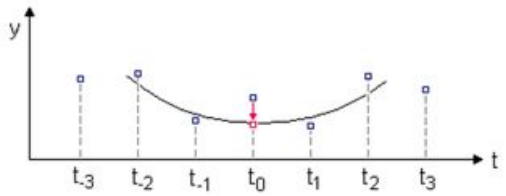


Abbildung 8.4.: Beispiel eines Savitzky Golay Filters

Es zeigte sich jedoch, dass der Filter die Daten der Prandtlsonde nicht ausreichend filterte und dass er daher das vorhandenen Problem nicht lösen konnte.

8.1.3. Gewählter Filter

Schlussendlich wurde der True Airspeed mit einem PT1-Filter gefiltert, weil dieser mit wenig Programmieraufwand implementiert werden kann und wenig Ressourcen verbraucht. Zusätzlich kann die Filterrate während des Flugs verändert werden. Jedoch darf die Filterrate nicht zu hoch ausfallen, da sonst neue Messdaten kaum mehr in den Endwert miteinbezogen werden und die Drohne bei einem True Airspeed geregelten Flug absturzgefährdet ist.
In der Abbildung 8.5 wird die Wirkung des verwendeten Filters verdeutlicht. Durch einen Pressluftversuch wurde eine schlagartige Änderung des True Airspeed simuliert, um die Verzögerung der Messdaten zu überprüfen. Mit einem Filterfaktor von 0.90 resultiert eine Verzögerung von knapp einer Sekunde. Dies ist erwünscht, da so das Regelsystem auf kurze Ausreisser der True Airspeed Messung wenig bis gar nicht reagiert.

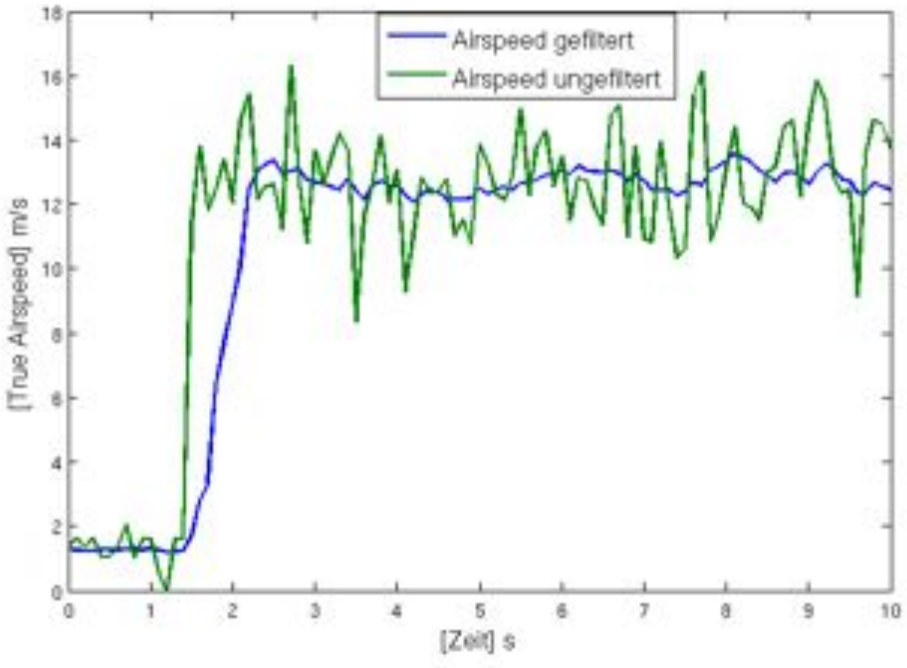


Abbildung 8.5.: Vergleich der True Airspeed - gefiltert / ungefiltert

8.2. Absolutdrucksensor

Für den Absolutdrucksensor wurde der AMS 5812-0150-A gewählt.

Um die Höhe abhängig vom barometrischem Druck ausrechnen zu können, wurde das Modul `AMSYS_Baro` geschrieben. Dieses Modul empfängt die Daten über `I2C` und berechnet die Höhendifferenz seit dem Einschalten des Autopiloten. Die Werte werden momentan noch ungefiltert im GCS angezeigt und haben keinen Einfluss auf die Flugregelungen. Falls es in Zukunft erwünscht ist, die GPS Höhe durch die barometrische Höhe zu ersetzen, muss dies im File `estimator.c` implementiert werden. Jedoch sollte dann mit einem Kalmanfilter gearbeitet werden, da die Berechnungsalgorithmen für die Filterung schon implementiert sind.

9. Geschwindigkeitsregelungen

Die Geschwindigkeit wird bezüglich dem True Airspeed geregelt. Das heisst, das Flugzeug soll bezüglich der Umgebungsluft eine relativ konstante Geschwindigkeit halten können. Da die Luft jedoch nicht homogen ist, sollen schlagartige Geschwindigkeitsänderungen, hervorgerufen durch Störgrössen wie Böen oder Luftverwirbelungen, keine sofortigen Auswirkungen auf die Stellgrösse haben.

9.1. Problemstellung

Die Drohne UMARS soll nach Anforderungsliste (siehe Abschnitt 5.2) $\pm$ 4 m/s genau fliegen können. Das Wichtigste dabei ist, dass die Drohne nicht auf jede kleine Geschwindigkeitsänderung reagiert, da sonst die Daten der Luftmessung, was die eigentliche Aufgabe der UMARS ist, verfälscht würden.
An der Testdrohne Mentor wurde eine Prandtlsonde montiert, welche zusammen mit dem Differenzdrucksensor unkonstante Werte liefert. Aus diesem Grund werden, wie im Abschnitt 8.1.3 beschrieben, die Messdaten gefiltert. Diese Filterung ermöglicht es, schlagartige Änderungen des True Airspeeds im Vorhinein zu unterdrücken.

9.2. Verschiedene Regelsysteme

Im Laufe der Bachelorarbeit wurden verschiedene Regelsysteme entwickelt, welche es uns ermöglichen sollten, den True Airspeed zu halten. Diese werden hier beschrieben. Die Ausgangsvariabeln `v_ctl_throttle_setpoint` und `h_ctl_pitch_setpoint` werden dem `Inner-Loop` übergeben, der dann das Flugzeug auf die gegebenen Sollwerte regelt. Der `v_ctl_throttle_setpoint` wird direkt an die Servoausgabe weitergeleitet. Die Variable `h_ctl_pitch_setpoint` definiert, welchen Winkel das Flugzeug zur Nulllage haben soll. Durch den `Pitch-Loop` im `Inner-Loop` wird das Höhenruder anhand der `h_ctl_pitch_setpoint` und den IMU-Daten geregelt.

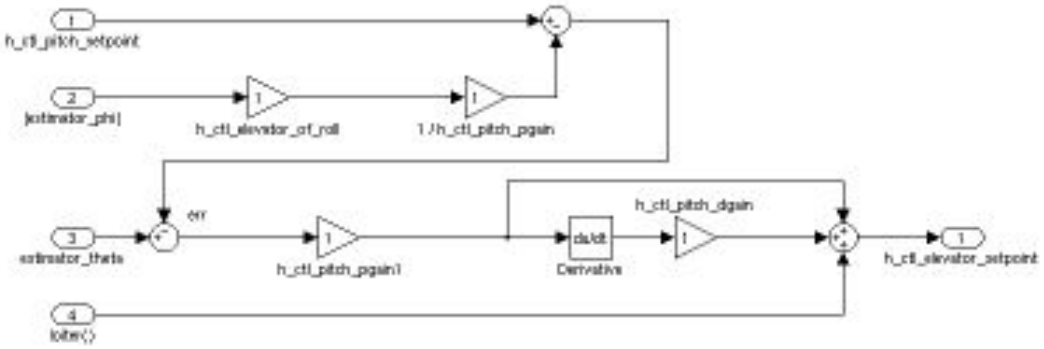


Abbildung 9.1.: Pitch-Loop von Inner-Loop

Im `Pitch-Loop` wird zuerst der zusätzliche Anstellwinkel für den Kurvenflug kompensiert, danach wird die Differenz zum Ist-Anstellwinkel (`estimator_theta`) berechnet und durch ein PD-Regler geleitet. Am Schluss wird zusätzlich noch eine `loiter` Variable dazu addiert, welche, um Fremdeinwirkungen zu vermeiden, bei diesem Projekt jedoch deaktiviert wurde.

9.2.1. Standardregelung

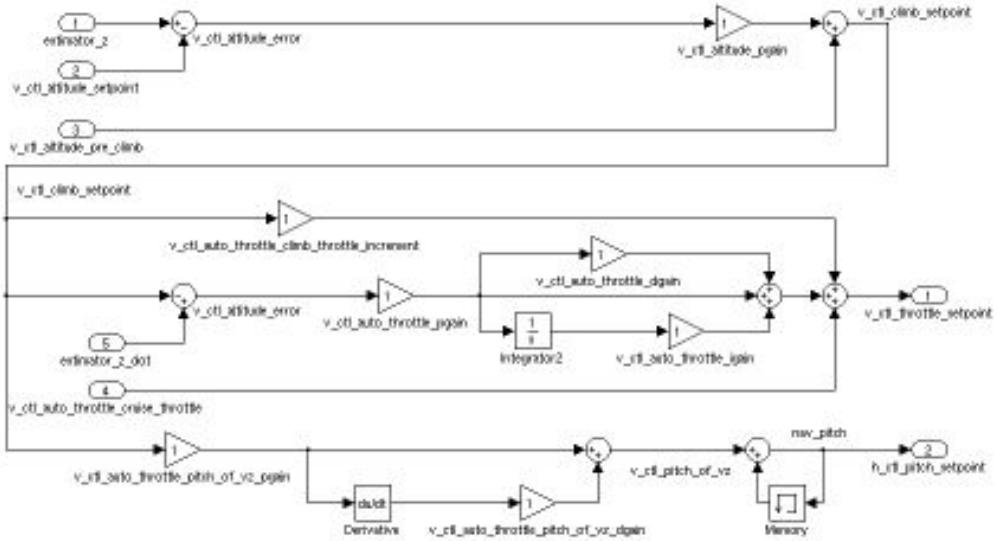


Abbildung 9.2.: Standardregelung

Die von uns benannte Standardregelung ist die Regelung, welche aktiv ist, wenn das Paparazzi-Projekt unverändert geladen wird.
Hier wird die Motorleistung anhand der gewünschten Steigrate errechnet, welche sich aus dem Flugplan ergibt. Die Höhe wird mit dem Anstellwinkel bezüglich der Erde verändert (Pitch), dieser Sollwert wird ebenfalls aus der Steigrate errechnet. Bei diesem Regelsystem wird der True Airspeed nicht geregelt.

9.2.2. Vassillis

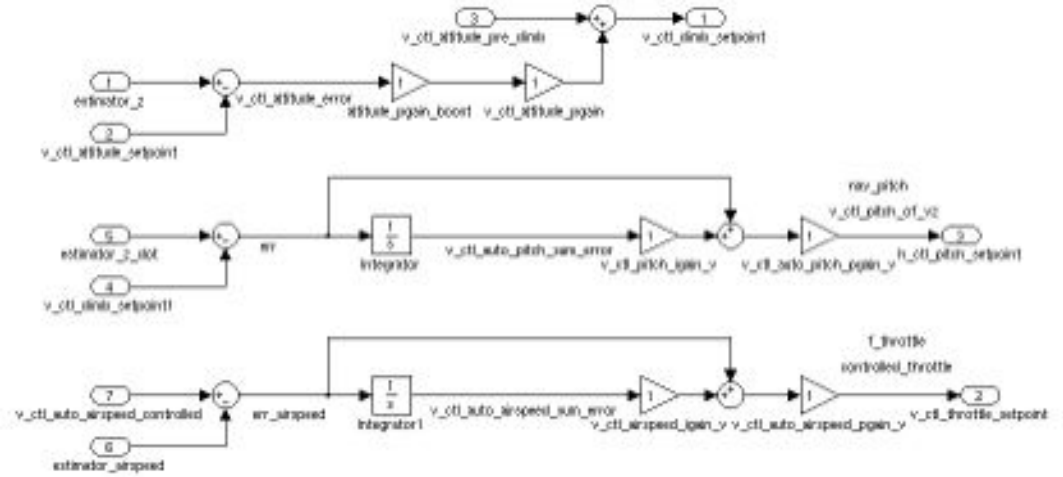


Abbildung 9.3.: Geschwindigkeitsregelung von Vassillis

Diese Regelung ist komplett in das Paparazzi-Projekt integriert. Es ist das momentan einzige Regelsystem im Hauptprojekt, welches den True Airspeed regelt.
Bei der Vassillis Regelung wird die Motorleistung durch die gewünschte Soll-Geschwindigkeit vorgegeben. Dazu wurde ein PI-Regler implementiert. Die Höhe wird, wie bei der Standardregelung, mit der Steigrate errechnet. Für die Flughöhe kommt ebenfalls ein PI-Regler zum Einsatz.

9.2.3. Airspeed Pitch Climbrate

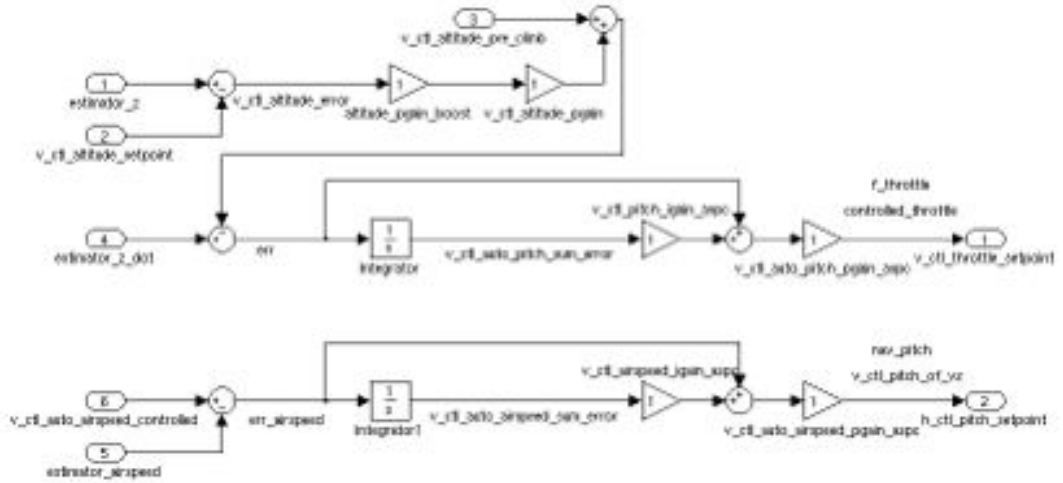


Abbildung 9.4.: Geschwindigkeitsregelung Airspeed Pitch Climbrate

Dieser Regler war der erste Versuch, im neu formatierten Paparazzi-Projekt eine Pitch-gesteuerte True Airspeed Regelung zu implementieren. Hier wird der True Airspeed mittels Pitch und die

Motorleistung abhängig von der Climbrate errechnet. Flugversuche haben gezeigt, dass diese Regelung sehr instabil und schwierig zu parametrieren ist.

9.2.4. Airspeed Pitch Simple

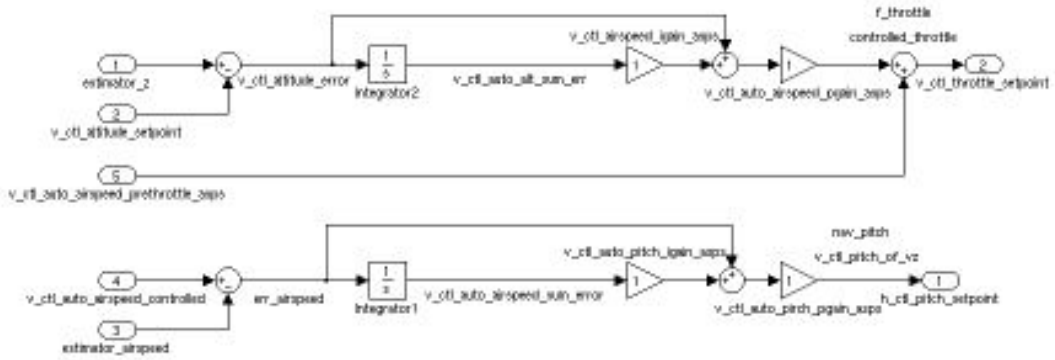


Abbildung 9.5.: Geschwindigkeitsregelung Airspeed Pitch Simple

Aus den Erkenntnissen von `Airspeed Pitch Climbrate` wurde die `Airspeed Pitch Simple` Regelung entwickelt. Der True Airspeed wird gleich wie bei der `Pitch Climbrate` Regelung mit einer Vorgabe des Pitch-Winkels geregelt. Die Flughöhe jedoch wird hier nur noch direkt durch den Unterschied zwischen Soll und Isthöhe errechnet. Für die Höhenregelung wird ein PI-Regler verwendet.

9.2.5. Airspeed Pitch Manual Power

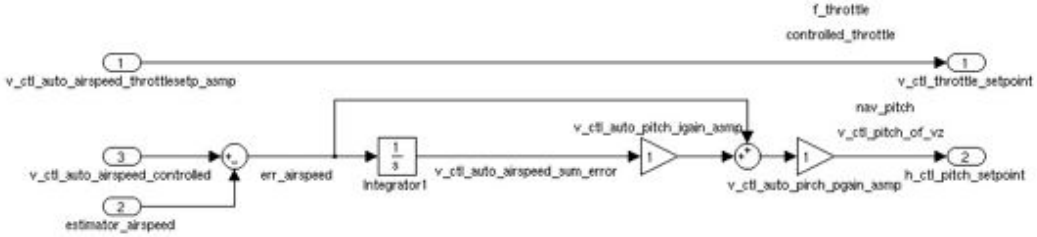


Abbildung 9.6.: Geschwindigkeitsregelung Airspeed Manual Power

Diese Variante der True Airspeed Regelung ist hauptsächlich für die Parameterfindung des `Airspeed Pitch Simple` Reglers nützlich, da die zusätzliche Airspeed Regelung unabhängig von der Motorleistung eingestellt werden kann. Die Motorleistung wird mittels eines Schiebereglers prozentual festgelegt, wobei beachtet werden muss, dass die Sollgeschwindigkeit mit der eingestellten Motorleistung im Einklang steht, da sonst das Flugzeug extrem steigt oder sinkt, weil die Höhe bei dieser Regelung nicht beachtet wird. Die gefundenen Regelparameter können

dann für die `Airspeed Pitch Simple` Regelung übernommen werden. Zu empfehlen ist, dass bei der Parameterfindung mehrere verschiedene Motorenleistungen gewählt werden, da sich das Verhalten des Flugobjekts je nach Motorleistung sehr stark verändert.

9.2.6. Airspeed Pitch Acceleration

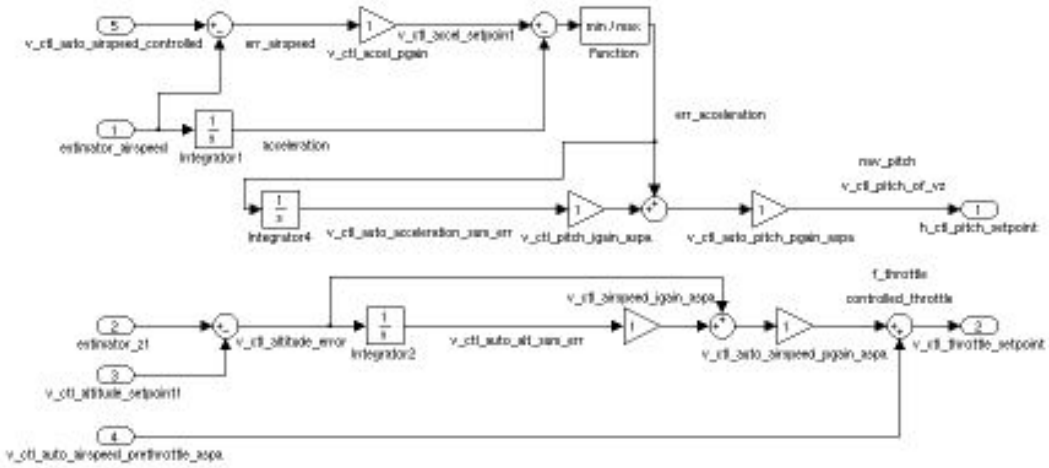


Abbildung 9.7.: Airspeed Pitch Acceleration

Dieser Regler entstand aus der Ableitung der Vassillis Regelung. Jedoch wird hier nicht die Steigrate, sondern die Geschwindigkeitsrate (Beschleunigung) verwendet.
Für die Regelung der Flughöhe dient derselbe Regler wie bei der `Airspeed Pitch Simple` Regelung. Der True Airspeed wird abgeleitet, um die Beschleunigung gegenüber der Umgebungsluft zu erhalten. Die Sollbeschleunigung erhält man durch das Multiplizieren eines P-Gains mit dem True Airspeed Fehler. Dies bedeutet, je grösser der Fehler zwischen Soll- und Istgeschwindigkeit wird, desto grösser soll die Beschleunigung sein. Falls der Fehler null beträgt, resultiert keine Beschleunigung, was zu einem Halten der Geschwindigkeit führt.
Dieser Regler ist wegen seiner Komplexität sehr schwer einzustellen. Die Regelsysteme können kaum entkoppelt werden, was die Parameterfindung erschwert. Dies hatte zur Folge, dass keine brauchbaren Regelparameter mit diesem Regler ermitteln werden konnten.

9.2.7. Airspeed Fixed Pitch

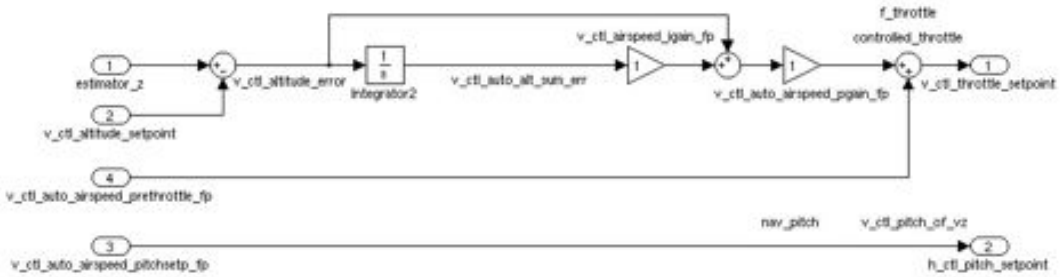


Abbildung 9.8.: Geschwindigkeitsregelung Airspeed Fixed Pitch

Dieses Regelsystem entstand bei der Entwicklung des Landeanfluges. Die Idee dahinter ist, einen festen Anstellwinkel (Pitch) zu halten und die Höhe mittels Motorleistung zu regeln. Da der optimale Anstellwinkel von jedem Flugzeug bekannt ist, fallen bei diesem Regler schon einige Parameter weg. Der Anstellwinkel wird lediglich im sogenannten `Inner-Loop` geregelt. Da zu Beginn keine Daten über den momentanen Anströmwinkel der Luft bekannt waren, da keine Mehrlochsonde oder Ähnliches auf der Testdrohne Mentor verbaut war, wurde für diesen Regler der Winkel zum Horizont aus der IMU verwendet. Im Verlauf dieser Arbeit wurde dann jedoch ein `Angle of Attack` Sensor gebaut, dessen Daten über den Anströmwinkel für diese Regelung schlussendlich verwendet wurden. Näheres zu diesem Thema wird in den Kapitel 7.2.5 und 15.4.5 beschrieben.

9.3. Wahl des optimalen Regelsystems

Beim Parametrieren der verschiedenen Regelparameter musste festgestellt werden, dass jedes Regelsystem sowohl positive als auch auch negative Aspekte mit sich bringt. Einige sind wegen ihrer Komplexität für eine True Airspeed Regelung unbrauchbar, während andere sich nur für eine bestimmte Aufgabe eignen, nicht jedoch für alle. Zum Beispiel eignet sich die `Airspeed Pitch Simple` Regelung sehr gut für den Messflug, jedoch nicht für die Landung. Für die Landung ist die `Airspeed Fixed Pitch` Regelung am besten, jedoch eignet sie sich nicht, um schnell Höhe zu gewinnen, da die Regelung dabei schnell überschwingen kann, da bei dieser Regelung direkt mit dem `Inner-Loop` geregelt wird. Somit wurde entschieden, einen sogenannten `Parameter-Changer` zu entwickeln, welcher es ermöglicht, zwischen verschiedenen Regelsystemen und Parametern während des Fluges zu wechseln. Die Methoden dieses `Changers` können aus einem Programmcode oder auch vom Flugplan her aufgerufen werden.

9.3.1. Airspeed Fixed Pitch Regelung in der Praxis

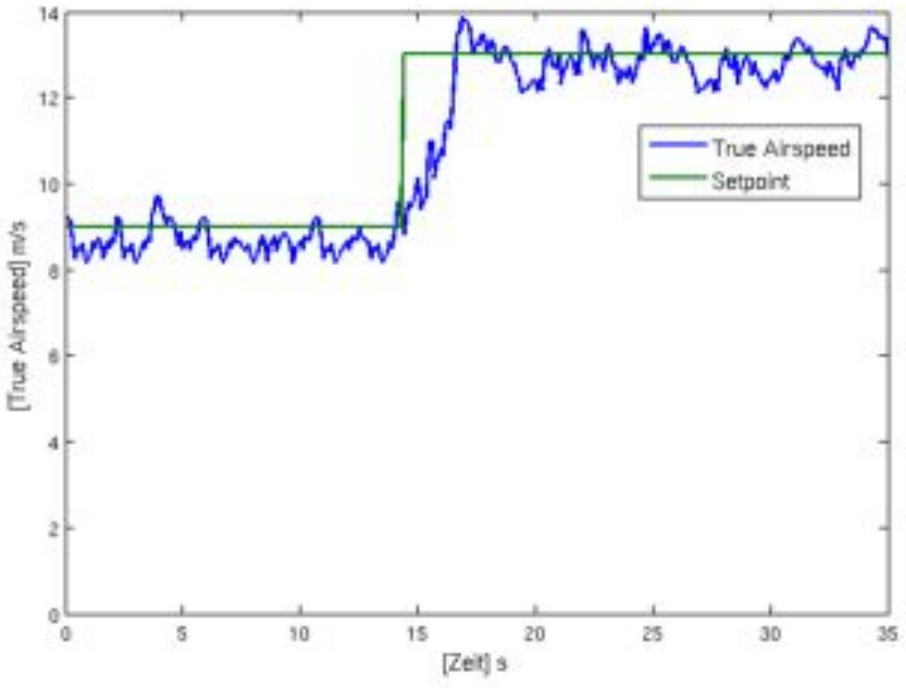


Abbildung 9.9.: Sprungantwort des True Airspeed von 9 m/s auf 13 m/s Simple Pitch geregelt

Wie im Abschnitt 9.3 beschrieben, wurde die `Airspeed Pitch Simple` Regelung und die `Airspeed Fixed Pitch` für den Messflug beziehungsweise für die Landung gewählt. Aus diesem Grund wurde die Fixed Pitch Regelung mittels dem Benchmarkmodul, welches im Abschnitt 15.4.3 beschrieben wird, so gut wie möglich parametriert. SSo gut wie möglich"heisst in diesem Fall, in möglichst wenigen Testfügen eine stabile und Störgrössen unabhängige Einstellung für verschiedene Sollwerte des True Airspeed zu finden. Die Abbildung 9.9 zeigt die Sprungantwort des True Airspeed von 9 m/s auf 13 m/s bei nahezu windstillen Verhältnissen nach der Parametrierung. Die Wetterverhältnisse spielen bei der Parametrierung der True Airspeed Regelsysteme eine sehr grosse Rolle, was bedeutet, dass die gefundenen Parameter zu einem späteren Versuch bei tieferen Temperaturen variieren können. Störgrössen wie Thermik, Wind, Temperatur und Luftfeuchtigkeit erschweren die Parameterfindung. Es muss versucht werden, die Parameter so zu wählen, dass ein möglichst breites Spektrum der Störgrössen abgedeckt wird. Erfahrungen haben gezeigt, dass der Einsatz eines Integralfaktors des PI-Reglers zu sehr stabilen, wenn auch langsam reagierenden, Regelungen führt. In einem weiteren Schritt könnte eine adaptive Regelung, welche beispielsweise mittels Temperatursensoren die Parameter anpasst, entwickelt werden.

10. Bungeestart

Die Drohne soll später möglichst einfach und auch unter schlechten Bedingungen, wie eine sehr kurze Startpiste, gestartet werden können. Daher sollte ein autonomer Startalgorithmus entwickelt werden, welcher es ermöglicht, die Drohne mit einem Gummiseil (Bungee) zu starten. Nachdem sich die Drohne vom Startseil gelöst hat, sollte der Autopilot den Motor starten und das Flugzeug bis zu einer gewissen Endhöhe navigieren. Danach sollte der Start abgeschlossen sein und der Autopilot in den nächsten Schritt des Flugplanes wechseln. In der Anfangsphase des Starts soll die Drohne so viel Energie aus dem Bungee erhalten, dass sie ohne die Hilfe des Motors abheben und einige Meter fliegen kann. So kann verhindert werden, dass sich das Startseil im Motor verfängt und die Drohne zum Absturz bringt.

Abbildung 10.1.: Übungsdrohne Mentor beim Gummiseilstart

10.1. Ablauf des Bungeestarts

Der grundsätzliche Ablauf des Starts ist denkbar einfach. In einem ersten Schritt spannt der Bediener der Drohne das Bungee-Seil. Dabei ist darauf zu achten, dass das Seil entsprechend dem Gewicht der Drohne zu spannen ist. Bei einer schweren Drohen ist also eine höhere Spannkraft des Gummiseils nötig als bei einer leichten. Danach macht der Benutzer die notwendigen Einstellungen an der `Ground Control Station` und lädt die Software gegebenenfalls neu auf den internen Speicher des Autopiloten. Zu diesen Einstellungen gehören etwa die minimale Geschwindigkeit für das Einschalten des Motors und die gewünschte Endhöhe des Startvorgangs. Sind alle Einstellungen gemacht, muss die Drohne für den Start vorbereitet werden und am Haken des Bungee-Seils befestigt werden. Nun sind alle Vorkehrungen für den Start getroffen. Der Bediener schaltet in den Modus `Auto 2` und betätigt kurz darauf das Fusspedal zur Auslösung des Seils. Daraufhin wird die Drohne stark beschleunigt, hebt ab und schaltet, sobald sie die `ThrottleLine` (siehe Abbildung 10.2) überquert hat den, Motor ein. Danach geht sie in den Steigflug über und fliegt geradeaus weiter, bis sie die vom Benutzer definierte Endhöhe erreicht hat. Zuletzt wechselt sie zum nächsten Punkt des Flugplanes und der Start ist abgeschlossen.

10.2. Vorhandener Startzyklus

In der aktuellen Paparazzi-Software ist bereits eine Bungee-Takeoff Routine vorhanden. Diese hat jedoch einige Schwachpunkte und sollte daher verbessert werden. Der Ablauf des vorhandenen Starts soll im Folgenden genauer betrachtet werden.

Der Bungee-Start oder auch Gummiseilstart benötigt im GCS nur einen Wegpunkt, den `WP_Bungee`. Dieser Punkt definiert die Position, an welcher der Haken des Gummiseils befestigt ist. Mit Hilfe von ihm und der aktuellen Position der Drohne wird die `LaunchLine` berechnet. Die `LaunchLine` ist eine gerade Linie, welcher gefolgt wird, bis die Drohne die `ThrottleLine` erreicht. Dabei wird die `LaunchLine`, welche den Kurs der Drohne darstellt solange neu berechnet, bis die Drohne das erste Mal den `MinSpeed` erreicht. Dies erlaubt dem Benutzer des Autopiloten, dass er seine Drohne nach dem Einschalten noch bewegen kann, ohne dass er vor dem Takeoff noch einen Neustart machen muss.
Die `ThrottleLine` verläuft rechtwinklig zum Sollkurs der Drohne und definiert, bei welcher Position der Motor eingeschaltet werden darf. Die Distanz vom `WP_Bungee` zur `ThrottleLine` kann über die `Takeoff_Distance` im `Airframe` definiert werden. Hat die Drohne die entspre-

chende Position und eine Geschwindigkeit, die grösser als die definierte Mindestgeschwindigkeit ist erreicht, wird der Motor eingeschaltet. Die Drohne fliegt nun weiter geradeaus, bis sie die vorgegebene Geschwindigkeit (`Takeoff_Speed`) und die gewünschte Endhöhe (`Takeoff_Height`) über der Höhe des Bungeehakens (`WP_Bungee`) erreicht hat. Danach ist der Start abgeschlossen und der nächste Punkt im Flugplan wird ausgeführt.

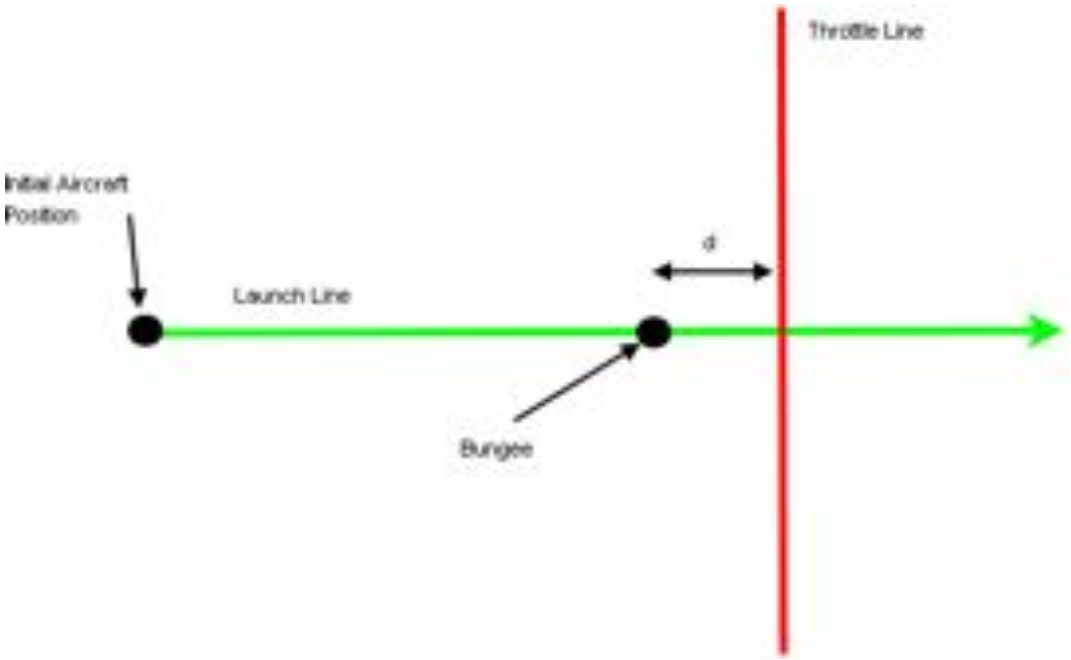


Abbildung 10.2.: Darstellung des vorhandenen Bunge-Takeoff

10.2.1. Probleme der vorhandenen Startroutine

Bei den Tests der vorhandenen Startroutine zeigte sich als Erstes, dass es nicht so einfach ist, den Punkt, an dem der Bungee-Haken befestigt ist, auf der Karte im Parazzi-GUI zu finden und zu setzen, da die Auflösung von dieser Karte nicht sehr hoch ist. Es ergibt sich also schnell eine Abweichung von einigen Metern zu der wirklichen Position des Hakens. Ausserdem fehlt eine visuelle Rückmeldung im `GCS`, welche anzeigt, an welcher Position der Motor gestartet werden soll. Eine solche Rückmeldung würde eine erste Kontrolle, ob die Einstellungen für den Start richtig gemacht wurden, ermöglichen.

Doch es zeigten sich noch zwei weitere Probleme, bevor das Gummiseil überhaupt ausgelöst wurde:

Das erste Problem trat aufgrund der ständigen Neuberechnung der `LaunchLine` auf. Die zwei

Punkte, durch welche diese Linie definiert ist, liegen sehr nahe zusammen, und da sich die Position der Drohne aufgrund von ungenauen GPS-Daten im Stillstand immer wieder etwas änderte, ergaben sich starke Winkeländerungen im Sollkurs.
Das zweite Problem trat aufgrund der schlechten Orientierung der Drohne im Stillstand auf. Wenn sich die Drohne nicht oder nur wenig bewegt, kann sie ihre Ausrichtung im Raum nicht bestimmen. Sie weiss also nicht, in welche Himmelsrichtung ihre Nase zeigt. Daher ist es oft vorgekommen, dass die Drohne sich in eine andere Richtung ausgerichtet glaubte, als sie in Wirklichkeit war. Dies hatte zur Folge, dass sie eine Kursänderung vornehmen wollte. Wurde nun also das Gummiseil ausgelöst, begann die Drohne sofort zu navigieren, wodurch sie von ihrem Kurs abkam. Um dieses Problem zu beheben, sollte ein Magnetometer eingebaut werden, mit welchem auch im Stillstand die Ausrichtung der Drohne bestimmt werden kann.
Einige Startversuche haben dann gezeigt, dass der Motor zu spät gestartet wird bzw. zu lange braucht, bis er seine maximale Drehzahl erreicht hat. Daher verlor die Drohne wieder einen grossen Teil der Höhe, welche sie mit Hilfe der Energie aus dem Startseil erreicht hatte. Da mit dieser Startroutine keine zufriedenstellenden oder wiederholbaren Starts durchgeführt werden konnten, wurde diese stark überarbeitet.

10.3. Neue Startzyklen

Da erst nach einem Test von einer neuen Startmethode eine definitive Aussage über dessen Güte gemacht werden kann und die Möglichkeit, einen solchen durchzuführen, nicht immer vorhanden war, wurden zwei neue Startalgorithmen entwickelt und eine Erweiterung, welche für beide Versionen anwendbar ist. Diese beiden neuen Methoden und die Erweiterung konnten dann an einem Flugtag getestet werden. Um einen Überblick über die neuen Startvorgänge zu erhalten, werden diese nun genauer beschrieben.

10.3.1. Änderungen in der Grundstruktur

Als Grundstruktur wird das bezeichnet, was für den normalen Benutzer beim Durchführen eines Starts sichtbar ist. Darunter fallen die Wegpunkte im `GCS` und die Randbedingungen, wie die Endhöhe des Starts, welche im `Airframe` definiert werden können.
Unter Abschnitt 10.2.1 wurde bereits angesprochen, dass sich der berechnete Kurs der Drohne, definiert durch zwei nahe zusammen liegende Wegpunkte, aufgrund von ungenauen GPS-Daten stark ändern kann. Dieses Problem wurde gelöst, indem ein neuer Wegpunkt für den Start erstellt

wurde, der `DirectionWP`. Mit diesem Punkt wird nun die Richtung des Starts definiert, da er aber viel weiter von der aktuellen Position der Drohne entfernt gesetzt werden kann (im `GCS`), haben GPS-bedingte Positionsschwankungen keinen grossen Einfluss mehr auf den Kurs der Drohne. Über die Höhe des `DirectionWP` kann ausserdem die Endhöhe des Starts definiert werden, was zur Folge hat, dass die Routinen des Autopiloten nach einer Änderung der Höhe nicht erneut auf den internen Speicher geladen werden müssen.
Ebenfalls unter Abschnitt 10.2.1 angesprochen, ist das Setzen der Position des Bungee-Hakens im `GCS` nicht einfach. Da nun die Richtung des Starts aber über die Position des `DirectionWP` definiert ist, wird dieser Punkt auch nicht mehr benötigt. Die Strecke, welche die Drohne zurücklegen muss, bis der Motor gestartet werden darf, kann nun direkt im `Airframe` definiert werden. Die letzte Änderung der Grundstruktur betrifft die Visualisierung der `ThrottleLine`. Um dem Benutzer einen Hinweis zu geben, bei welcher Position der Motor etwa starten wird, wurde ein neuer Wegpunkt erzeugt, welcher vom Autopiloten in den Schnittpunkt der `LaunschLine` und der `ThrottleLine` gesetzt wird. Dieser Punkt trägt den Namen `TP`.

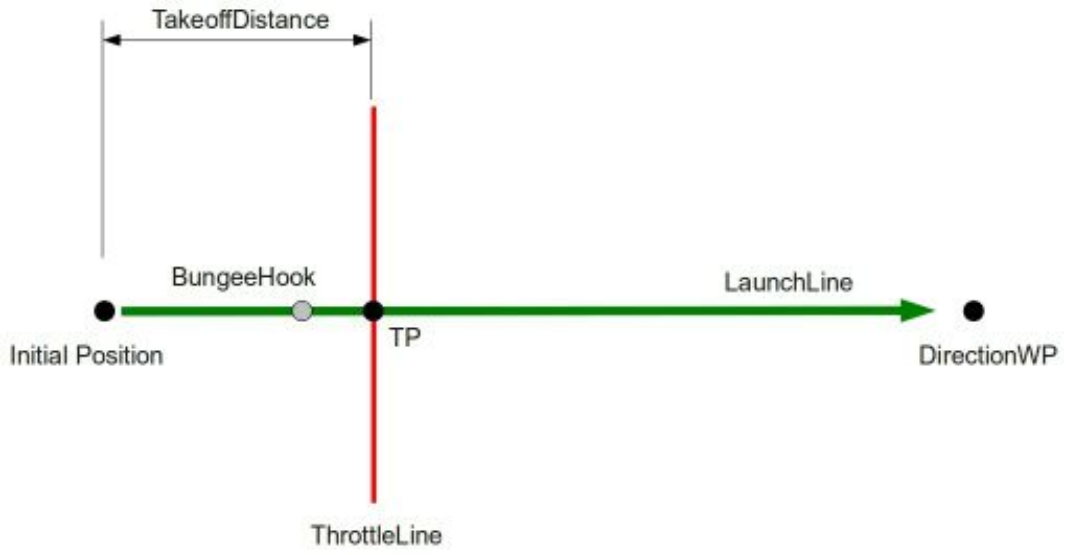


Abbildung 10.3.: Umstellung der Grundstruktur

Die oben stehende Abbildung 10.3 verdeutlicht die Umstellungen in der Grundstruktur der ZHAW-Startmethoden. Zu beachten ist, dass der Wegpunkt `BungeeHook` keine Bedeutung mehr hat und lediglich zur Visualisierung dargestellt wurde.

10.3.2. Allgemeine Änderungen im Startzyklus

Das unter Punkt 10.2.1 angesprochene Problem des zu frühen Navigierens der Drohne sollte mit einem Magnetometer behoben werden. Doch das vorhandene Magnetometer war beschädigt und konnte nicht gebraucht werden. Da kurzfristig kein neues erhältlich war, musste eine andere Lösung für das Problem gefunden werden. Die Lösung ist relativ naheliegend. Wenn die Drohne keinen Kurs hat, dem sie folgen soll, ist nur die Lageregelung und die Höhenregelung aktiv. Der Startalgorithmus wurde also so umgeschrieben, dass erst ein Kurs definiert wird, wenn der Motor eingeschaltet wird. Zu diesem Zeitpunkt wird die Drohne schon einige Meter zurückgelegt haben und ihre genaue Ausrichtung im Raum wird bekannt sein. Ausserdem wurde die Ursache des langsamen Hochdrehens des Motors gefunden und beseitigt. Es handelte sich um einen sogenannten `ThrottleSlewLimiter`, welcher die Steigerung der Motorenleistung begrenzt. Dies wird benötigt, wenn ein Verbrennungsmotor verwendet wird, welcher durch ein zu schnelles Gasgeben abgewürgt werden würde.

10.3.3. Ablauf des geführten Starts

Die erste Version des ZHAW-Takeoffs basiert auf dem bereits vorhandenen Algorithmus. Doch die Fehler, die bei diesem festgestellt wurden, wurden behoben und die Grundstruktur wurde wie unter Punkt 10.3.1 beschrieben umgestellt. Der genaue Ablauf des Starts soll anhand der unten stehenden Grafik 10.4 genauer beschrieben werden.

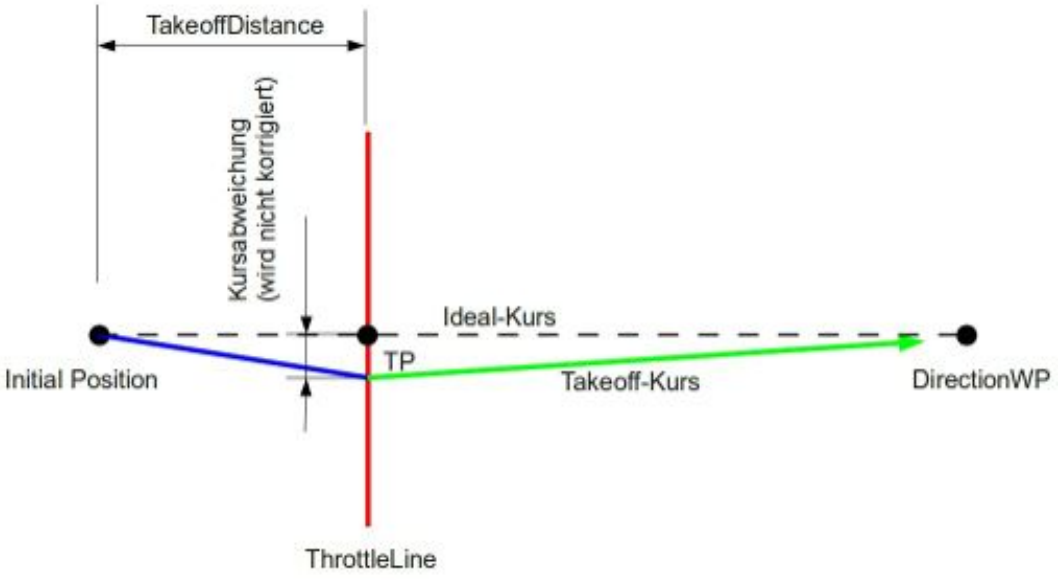


Abbildung 10.4.: Abbaluf des geführten ZHAW-Takeoffs (Draufsicht)

Zu beachten ist, dass es sich bei der Abbildung 10.4 um eine Ansicht von oben handelt, dargestellt ist also die x-y-Ebene.

Wenn alle Vorkehrungen für den Start getroffen sind, kann der Benutzer des Autopiloten in den Modus `Auto 2` schalten. Ab dann wird die Position der `ThrottleLine` nicht mehr neu berechnet. Kurze Zeit später kann er per Fusspedal den Start auslösen. Daraufhin wird die Drohne vom Gummiseil beschleunigt und hebt schliesslich ab. Während dieser Phase ist weder eine Kurs- noch eine Höhenregelung aktiv. Lediglich die Lageregelung der Drohne hält diese in einer waagrechten Position. Dadurch wird die Drohne automatisch in Richtung des Bungee-Seils beschleunigt und gerät nicht durch unbeabsichtigte Kursänderungen in eine Schieflage. Sie kann aber auf dieser Strecke (siehe blaue Linie Abbildung 10.4) eine gewisse Abweichung vom Idealkurs bekommen, welche nicht korrigiert wird.
Erst wenn die Drohne die Linie überquert, bei der der Motor eingeschaltet werden darf, werden Kurs- und Höhenregelung aktiv. Der Kurs, dem gefolgt werden soll, wird in diesem Moment mittels der aktuellen Position der Drohne und dem `DirectionWP` berechnet (Gerade Linie zwischen den beiden Punkten). So muss während des ganzen Starts keine starke Kursänderung vorgenommen werden. Die Höhenregelung wird ebenfalls dann eingeschaltet, wenn die `ThrottleLine` überquert wird. Sie gibt dem unbemannten Flugobjekt die Endhöhe des Starts als Sollwert vor. Wird diese Höhe erreicht, ist der Start beendet.

10.3.4. Start anschliessendem Gleitpfad

Der Start mit anschliessendem Gleitpfad stellt sicher, dass die Drohne nach dem Einschalten des Motors einer Linie im dreidimensionalen Raum folgt, also einem definierten Kurs folgt und dabei eine konstante Steigrate einhält.

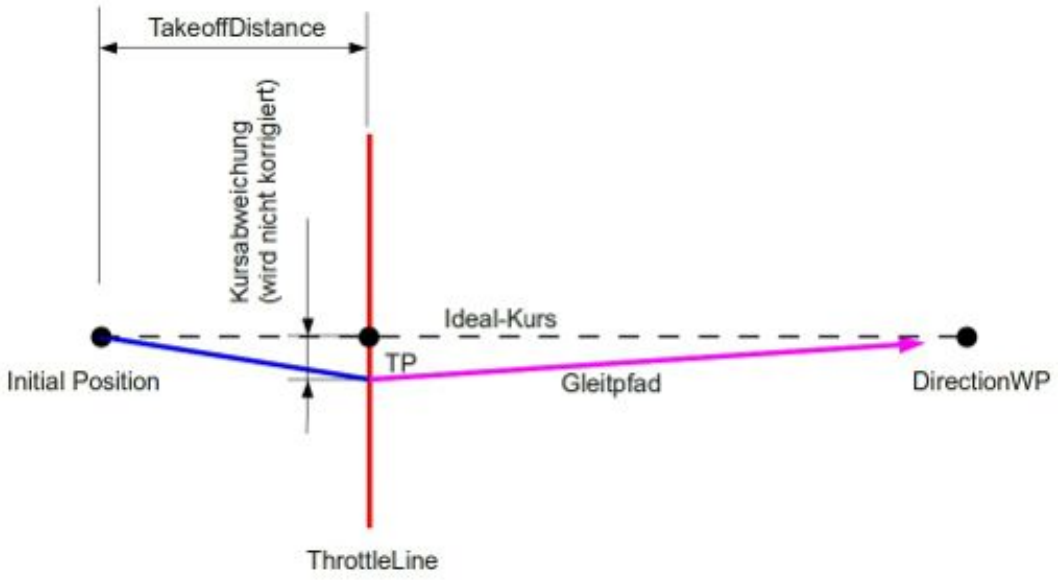


Abbildung 10.5.: Abbaluf des ZHAW-Takeoffs mit anschliessendem Gleitpfad (Draufsicht)

Die Abbildung 10.5 zeigt den Ablauf des Starts mit anschliessendem Gleitpfad. Der erste Teil bis zum Überqueren der `ThrottleLine`, ist exakt der gleiche wie bei der geführten Startprozedur unter Punkt 10.3.3 und daher wird dieser hier nicht näher beschrieben.
Auch hier werden die Kurs- und die Höhenregelung erst nach dem Einschalten des Motors aktiviert. Es werden jedoch nicht mehr separat ein Sollkurs und eine Sollhöhe definiert, sondern es wird ein Gleitpfad, eine Gerade mit definierter Richtung und Steigung, definiert, welchem die Drohne folgen soll. Diese Gerade wird durch die zwei Punkte `TP` und `DirectionWP` definiert. Hierbei ist darauf zu achten, dass der `DirectionWP` in genügender Entfernung gesetzt wird, da der Gleitpfad sonst zu steil wird. Ausserdem ist im Wegpunkt `TP` die entsprechende Bodenhöhe zu definieren. Wenn die Drohne dem Gleitpfad bis zum Ende gefolgt ist, ist der Start beendet.

10.3.5. Erweiterung der Startmethoden

Wie bereits angesprochen, wurde eine Erweiterung entwickelt, die für beide vorher genannten Startversionen angewendet werden kann. Bei dieser Erweiterung handelt es sich um einen Zwischenschritt, welcher in Kraft tritt, sobald die Drohne die `ThrottleLine` überquert hat. In diesem Zwischenschritt (Abbildung 10.6, oranger Abschnitt) wird zwar der Motor eingeschaltet und auch die Höhenregelung wird aktiviert, aber die Drohne wird noch nicht auf einen vorgegebenen Kurs gesteuert. Erst wenn sie die in Abbildung 10.6 dargestellte `NavigationLine` überschritten hat, wird die Kursregelung aktiviert.
Dies soll bewirken, dass die Drohne nach dem Aushängen des Gummiseils und dem Starten des Motors in eine stabile Fluglage gelangen kann, bevor sie mit dem Navigieren anfängt. Wenn sie die Position, ab welcher sie navigieren darf, überschritten hat, kann entweder einem festen Kurs gefolgt werden und dabei auf eine gewisse Höhe gestiegen werden oder es kann einem Gleitpfad, wie unter Abschnitt 10.3.4 beschrieben, gefolgt werden.
Ist die Erweiterung aktiviert, wird ein neuer Wegpunkt im `GCS` benötigt, welcher ähnlich wie der `TP` als Visualisierung der Position, ab welcher navigiert werden darf, dient.

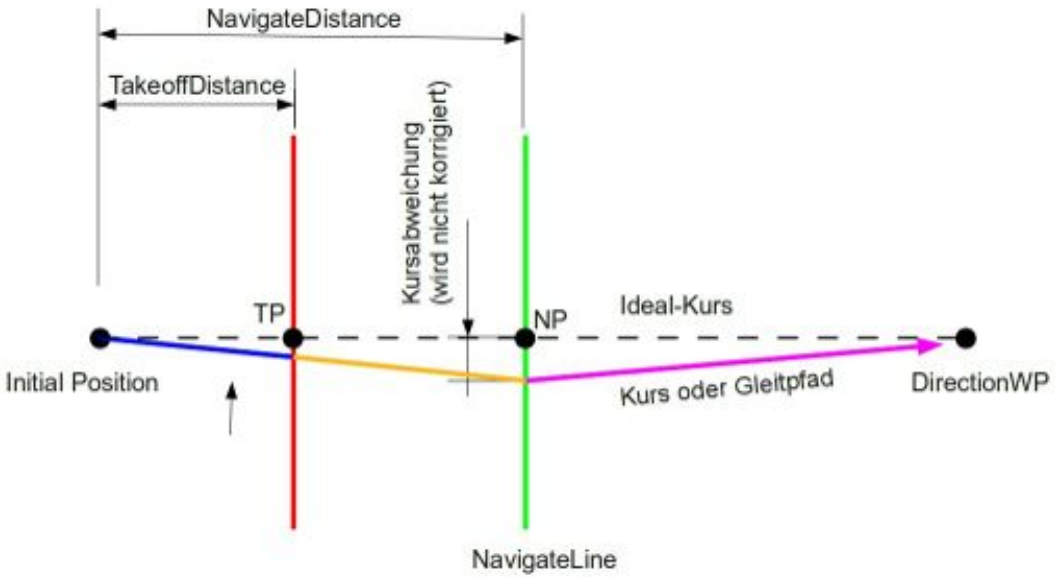


Abbildung 10.6.: Darstellung der Erweiterung zu den Takeoff-Methoden (Draufsicht)

10.4. Probleme beim Start

Bei den Tests des autonomen Starts zeigte sich immer wieder, dass die Drohne nach einer guten ersten Phase des Starts wieder zu sinken begann, obwohl sie laut Autopilot aber steigen sollte. Es stellte sich heraus, dass ein Problem mit der Inertial Measurement Unit (IMU) vorlag, welches durch die hohe Beschleunigung (bis zu 3.5 g) am Anfang des Starts zu erklären war.

Ergebnisse

Die beste Veranschaulichung der Resultate sind hier die Videos, welche auf der beiligenden CD unter `05_Videoaufnahmen/ZHAW_Start` zu finden sind. Es ist anzumerken, dass einige autonome Starts gelungen sind, jedoch trat fasst immer das bereits angesprochene Problem mit der IMU auf, welches zu einem Sinken der Drohne führt. Mögliche Lösungen für dieses Problem sind unter Abschnitt 18.2.5 beschrieben.

10.5. Checkliste für den Bungeestart

Die folgende Checkliste soll dazu dienen, dass bei einem Start keine Einstellung vergesse wird. Sie sollte vor jedem Start durchgegangen werden.

10.5.1. Vor dem Flashen

Die folgenden Angaben sind Airframe vor dem Flashen (Übertragen des Programms auf den Autopilot) zu setzen.

- `THROTTLE_SLEW_LIMITTER` = 0.5s
- `TAKEOFF_MINSPEED` setzen
- `TDISTANCE` bestimmen anhand der Auslenkung des Gummiseils
- `TDISTANCE` eintragen
- `TAKEOFF_SPEED` setzen

10.5.2. Angaben im GCS

Die letzten Angaben, die vor dem Start gemacht werden müssen, können direkt im GCS gesetzt werden, wenn die Drohne bereits startbereit am Gummiseil hängt.

- Bodenhöhe in `TP` eintragen
- Endhöhe für Start in `TOD` eintragen
- `TOD` setzen für gewünschte Startrichtung
- Autopilot launchen (Launch-Button)
- Start einleiten (Takeoff-Button)
- Kurz vor dem Auslösen des Seils in `Auto 2` schalten

11. Messung der Flughöhe vor der Landung

Bevor mit der detaillierten Planung der Landeroutine begonnen werden konnte, musste definiert sein, wie die genaue Höhe der Drohne unmittelbar vor der Landung bestimmt werden soll. Dazu wurden nicht nur die auf dem Markt erhältlichen Distanzmesssensoren, sondern auch andere Verfahren zur Distanzmessung betrachtet. Die verschiedenen Möglichkeiten seien im Folgenden kurz beschrieben.

11.1. Untersuchte Messarten

11.1.1. Optische Sensoren

Laufzeitmessung mittels Laser

Bei der Laufzeitmessung wird, wie in Abbildung 11.1 schematisch dargestellt, die Zeit gemessen, welche das Licht für das Zurücklegen des Weges zum Messobjekt und wieder zurück benötigt. Aus der Lichtgeschwindigkeit und der Brechkraft des entsprechenden Mediums, in welchem die Messung durchgeführt wird, lässt sich die Distanz berechnen.

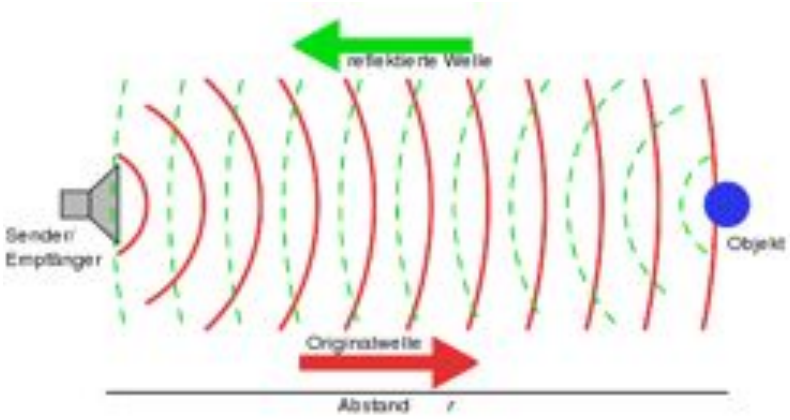


Abbildung 11.1.: Prinzip der Laufzeitmessung

Vorteile

- Der Vorteil dieser Messmethode ist die schnelle Reaktionszeit, welche es ermöglicht, die Messung auch bei hoher Geschwindigkeit durchzuführen.

Nachteile

- Aufgrund der hohen Geschwindigkeit des Lichtes sind die Zeiten, welche das Licht für das Zurücklegen der Distanz benötigt, sehr klein (Nano- bis Pikosekunden). Zur exakten Messung dieser Zeit müsste ein teurer Zeitgeber angeschafft werden.
- Ausserdem könnten durch einstrahlendes Sonnenlicht Messfehler entstehen.

Lasertriangulation

Bei der Lasertriangulation wird der Strahl des Lasers auf den Boden gerichtet und von einer neben dem Laser angeordneten Kamera beobachtet. Der Winkel, unter dem der Lichtpunkt beobachtet wird, ändert sich nun proportional zur Entfernung des zu messenden Objekts. Aus der Positionsänderung kann nun mit Hilfe der Trigonometrie die Entfernung berechnet werden (siehe Abbildung 11.2).

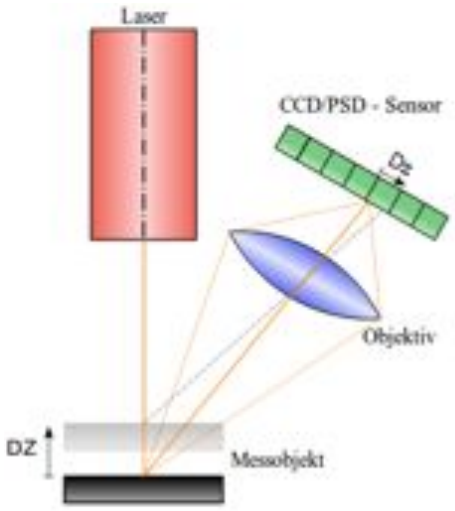


Abbildung 11.2.: Prinzip der Lasertriangulation

Vorteile

- Bei der Lasertriangulation werden nur trigonometrische Zusammenhänge für die Abstandsmessung benötigt. Daher kann die Messung kontinuierlich erfolgen und würde sich gut für die Höhenmessung aus der bewegten Drohne eignen.

Nachteile

- Es würde ein sehr starker Laser benötigt, damit dieses Verfahren auch unter den gegebenen Umständen (Messung auf Wiese / Acker) funktionieren würde. Um die Fremdlichtempfindlichkeit und den Einfluss inhomogen reflektierender Oberflächen zu senken, muss der Messpunkt möglichst klein und hell sein.

Stereokamera

Das Funktionsprinzip einer Stereokamera beruht auf Bilderkennung. An der Drohne werden zwei Kameras angebracht, welche einen definierten Abstand und Winkel zueinander haben und den gleichen Öffnungswinkel. Zur Abstandsmessung werden die Bilder, welche von den beiden Kameras geliefert werden, miteinander verglichen. Auf Grund der definierten Ausrichtung der Kameras kann aus den Unterschieden der beiden Kamerabilder die Distanz errechnet werden.

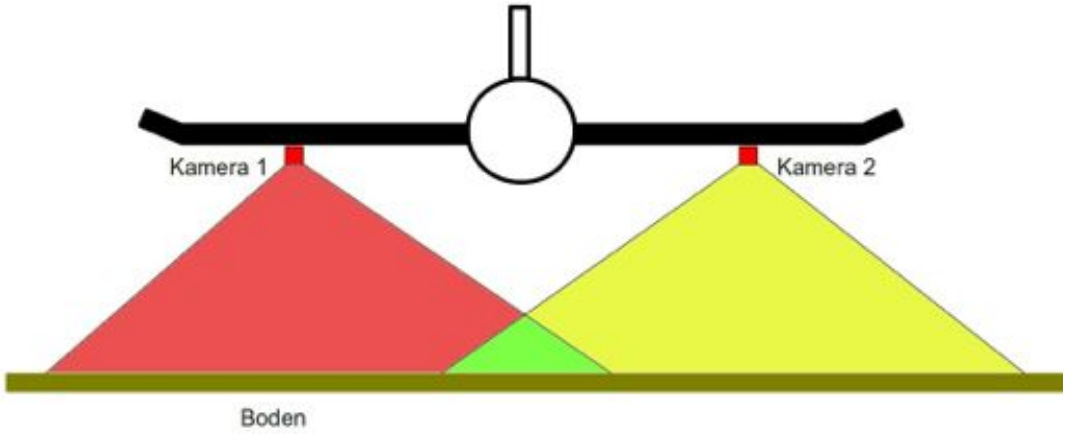


Abbildung 11.3.: Schematische Darstellung eines Stereokamerasystems

Vorteile

- Es entstehen keine Probleme mit unregelmässigen, schlecht reflektierenden Oberflächen

Nachteile

- Da die Bildverarbeitung sehr aufwändig ist, müsste für dieses Verfahren ein eigener Prozessor angeschafft werden. Zudem wäre die Umsetzung mit einem grossen Programmieraufwand verbunden.
- Bei Dunkelheit oder einer gleichmässigen Schneedecke würde das Verfahren versagen.
- Vibrationen in den Flügeln würden sich negativ auf die Genauigkeit auswirken.
- witterungsabhängig
- Gewicht (Kameras + Prozessor)

11.1.2. Ultraschall

Der Ultraschallsensor hat das gleiche Funktionsprinzip wie die Laufzeitmessung mit einem Lasersensor. Es wird ein Schallimpuls gesendet und die Zeit, bis das Echo zum Sensor zurück kehrt, wird gemessen. Nun kann mit der Schallgeschwindigkeit und der gemessenen Zeit die Entfernung berechnet werden.

Abbildung 11.4.: Verwendeter Ultraschallsensor

Vorteile

- Wenig Aufwand für den Hardwareaufbau nötig.
- Schnittstelle für den Anschluss an den Autopiloten bereits vorhanden.
- Messung kann kontinuierlich erfolgen.

Nachteile

- Messbereich bis nur etwa 7.5m.
- Mögliche Messungenauigkeiten über Wiese / Acker.

11.1.3. Barometrischer Drucksensor

Für diese Höhenmessung werden zwei Drucksensoren benötigt. Der eine wird auf Höhe der Landebahn angebracht, um einen Referenzdruck zu erhalten, während sich der andere in der Drohne befindet. Während des Flugs wird der Referenzdruck vom Boden über Funk an die Drohne gesendet, mit welchem diese dann ihre aktuelle Höhe berechnen kann. Es gäbe auch die Möglichkeit, nur einen Sensor in der Drohne zu verwenden, doch ein Gespräch mit einem Meteorologen ergab, dass sich der wetterabhängige Luftdruck in 30 Minuten. um einen Betrag ändern könne, welcher etwa 30 Höhenmetern entspricht. Diese Tatsache erhärtete die Notwendigkeit eines Referenzdrucks. Wie ein solcher Drucksensor aussehen könnte ist in Abbildung 11.5 zu sehen.

Abbildung 11.5.: Darstellung eines barometrischen Drucksensors

Vorteile

- Wenig Aufwand für den Hardwareaufbau nötig.
- Schnittstelle für den Anschluss an den Autopiloten bereits vorhanden.
- Messung kann kontinuierlich erfolgen.

Nachteile

- Eine Bodenstation für den Referenzdruck müsste eingerichtet werden.
- Eine weitere Erhöhung der zu sendenden Datenmenge.
- Messgenauigkeit nur etwa 1m.

11.1.4. Referenz GPS

Das vorhandene GPS-System liefert bereits eine ungefähre Höhenangabe, diese ist aber sehr ungenau und daher für die Landung nicht geeignet. Es gibt jedoch die Möglichkeit, ein Referenz GPS einzusetzen. Mit diesem lässt sich dann die Abweichung in der Höhe bestimmen und so kann eine genauere Angabe über die momentane Höhe der Drohne gemacht werden.

Vorteile

- Es ist kein weiterer Sensor nötig, dessen Daten eingelesen und verarbeitet werden müssen.

Nachteile

- Die Synchronisation der zwei GPS-Systeme ist mit erheblichem Programmieraufwand verbunden. Es muss sichergestellt werden, dass beide Systeme mit denselben Satelliten kommunizieren und die gleichen Datenpakete empfangen.

11.1.5. Leitstrahl

Dieses System wird an den Flughäfen eingesetzt, welche über ein Instrumentenanflugverfahren[1] verfügen. Das System basiert auf zwei gerichteten modulierten Funksignalen mit unterschiedlicher Frequenz. Die Funkkegel sind so ausgerichtet, dass bei einem Anflugwinkel von 3° das Strahlungsmaximum auftritt. Durch die Modulationen auf den zwei Trägerfrequenzen ist es möglich, die Abweichung zum optimalen Kurs zu detektieren und sich danach auszurichten. Der Sender, welcher die zwei Kegel sendet, wird Gleitwegsender genannt und befindet sich am Rand der Landebahn am Ort des Aufsetzpunktes.

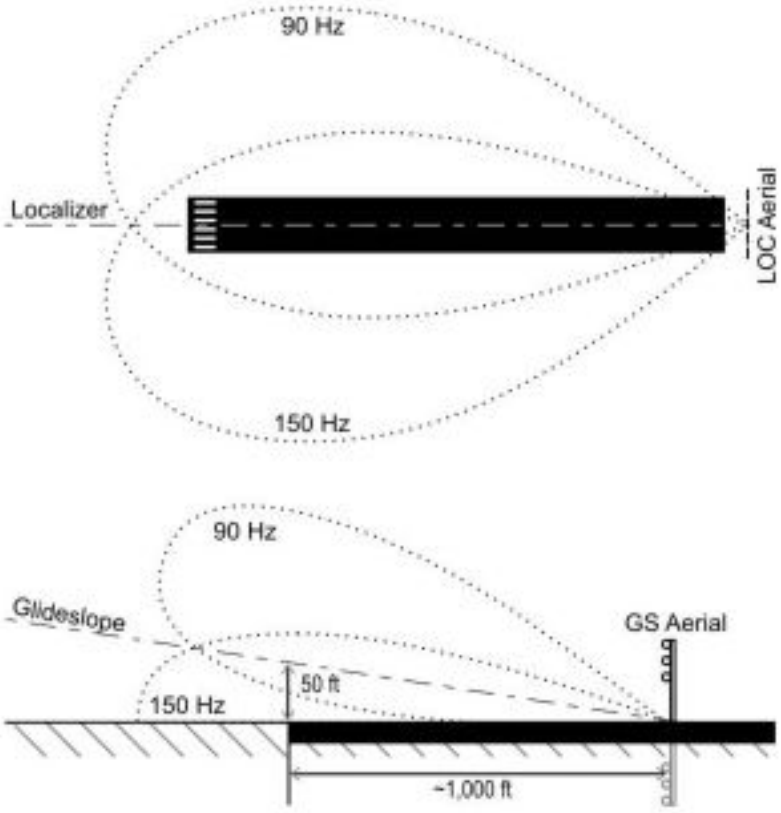


Abbildung 11.6.: Illustration des Leitstrahlsystems

Vorteile

- sehr genau
- witterungsunabhängig
- zuverlässig
- wird bei Passagierflugzeugen eingesetzt

[1] Quelle: http://de.wikipedia.org/wiki/Instrumentenlandesystem, Datum: 19.05.2011

Nachteile

- sehr teuer
- aufwändig
- benötigt ausgerichteten Sender am Boden
- grosses Wissen über Sender und Funkmodulationen nötig

11.2. Wahl des Sensors

Nach Abwägung der Vor- und Nachteile der oben aufgezeigten Distanzmessverfahren galt, es eine Entscheidung zu treffen. Schliesslich dauert es auch einige Tage, bis der Sensor geliefert würde.

Die Entscheidung fiel letztlich auf den Ultraschallsensor oder auch Sonar. Dieser Sensor war im Vergleich zu den anderen Sensoren leicht in die Drohne einzubauen und die Schnittstelle vom Analogausgang des Sensors zur Paparazzi-Hardware ist auch bereits definiert. Ausserdem ist er mit ca. sFr 100.- verhältnismässig billig in der Anschaffung.
Die laserbasierten Messverfahren fielen aus dem Rennen, da die Gefahr für Abweichungen durch reflektierendes Sonnenlicht und durch die inhomogen reflektierende Oberfläche zu gross ist. Um diese zu minimieren, müsste das gesendete Licht aufwändig frequenzmoduliert werden. Das Referenz-GPS- und das Richtstrahl-System schieden aus, da die benötigte Hardware zu teuer und die notwendige Software zu aufwändig waren.
Was zum Schluss noch blieb war, das Druckmesssystem. Bei einem Test zeigte sich aber, dass die Sensoren Druckschwankungen ausgaben, welche einem Höhenunterschied von bis zu 1.5 m entsprachen. Dies liess darauf schliessen, dass die Höhenmessung mittels Differenzdruck nicht genau genug gewesen wären.

11.3. Sensorspezifikationen

Nachdem die Sensorwahl getroffen war, musste der entsprechende Sensor angeschafft werden. Da bereits ein Sensor, der die geforderten Anforderungen erfüllte, vorhanden war, musste kein neuer bestellt werden. Der Sensor hat die Abmasse 22 mm x 20 mm x 25 mm (l x b x h) und eine maximale Reichweite von 7,65 m, dabei erreicht er eine Auflösung von 1 cm.
Die Abbildung 11.4 zeigt den verwendeten Sensor, das Datenblatt ist im Anhang I.2 zu finden.

12. Landung

Eine sehr wichtige Aufgabe dieser Arbeit war es, eine autonome Landung zu programmieren, welche es einem späteren Benutzer der Messplattform UMARS erlaubt, diese auch ohne Modellflugerfahrungen sicher landen zu können. Aus der konventionellen Fliegerei sind einige unterschiedliche Landeanflüge bekannt, welche im Rahmen dieser Arbeit genauer betrachtet wurden und auf ihre Tauglichkeit für eine programmatische Umsetzung geprüft wurden. Diese Landeabläufe sind immer an die jeweiligen Randbedingungen geknüpft. Eine Landung auf einem Flugzeugträger kann beispielsweise nicht gleich ablaufen wie eine auf einer grosszügigen Landebahn. Bei der Flugzeugträger-Landung ist der Aufsetzpunkt des Flugzeuges sehr wichtig, während bei der Landung auf einem Rollfeld mehr Gewicht auf ein sanftes Aufsetzen gelegt wird.

Abbildung 12.1.: UMARS im Landeanflug

12.1. Allgemeiner Ablauf einer Landung

Im Allgemeinen setzt sich eine programmierte Landung aus sieben Abschnitten zusammen, bei welchen jeweils Position, Geschwindigkeit und Höhe des Flugzeuges geregelt werden müssen. Da die Sollwerte für diese drei Regelgrössen jedoch in jedem Abschnitt unterschiedlich sind, resultiert daraus ein relativ komplexer Landeablauf. Das Grundgerüst für eine Landung soll im Folgenden mit Hilfe der Grafik 12.2 genauer beschrieben werden.

12.1.1. Grundgerüst einer Landung

1. **Reiseflug**
   - Das Flugzeug fliegt mit Reisegeschwindigkeit und Reisehöhe
   - Landung wurde noch nicht eingeleitet, Motor ist eingeschaltet

2. **Sinkflug**
   - Landezone wird angeflogen
   - Motor ist eingeschaltet

3. **Abfangen**
   - Wenn das Flugobjekt seine Sollhöhe erreicht, hat wird es abgefangen
   - Motor ist eingeschaltet

4. **Parallelflug gegenüber dem Boden**
   - Parallelflug gegenüber dem Boden, bis die Position für den Beginn des Flares erreicht ist
   - Motor ist eingeschaltet

5. **Flare**
   - Die Fluggeschwindigkeit (True Airspeed) wird auf die Landegeschwindigkeit abgebremst
   - Das Flugzeug geht in den Schwebeflug über
   - Motor wird ausgeschaltet

6. **Aufsetzen**
   - Das Flugzeug setzt auf dem Boden auf
   - Motor ist ausgeschaltet

7. **Ausrollen / Abbremsen**
   - Das Flugobjekt wird bis zum Stillstand mit allen vorhandenen Hilfsmitteln (Schubumkehr, Radbremsen, etc.) abgebremst

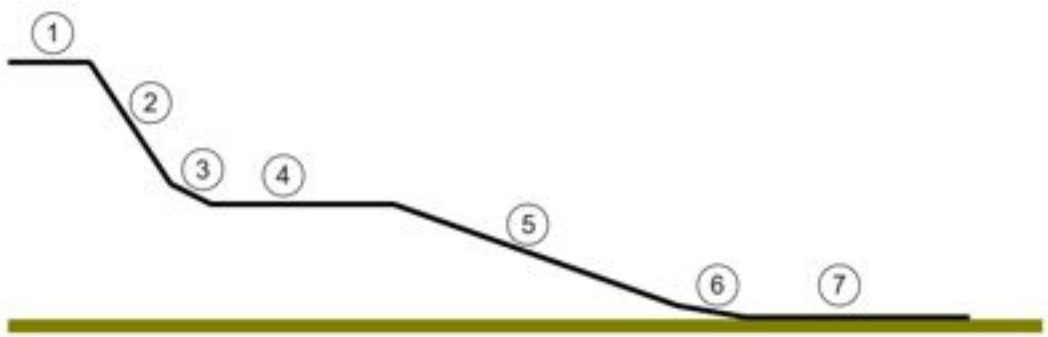


Abbildung 12.2.: Allgemeiner Landeablauf

12.2. Landemethoden

Nun gibt es verschiedene Landemethoden, welche aber alle auf dem oben beschriebenen Ablauf basieren. Im Folgenden sollen die drei Landemethoden

- Vorhandene Landung
- Landung mit konstanter Sinkrate
- Landung mit flaren

genauer betrachtet werden.

12.3. Vorhandener Landezyklus

In der zu Beginn dieser Arbeit heruntergeladenen Paparazzi-Software war bereits eine autonome Landung vorhanden, welche genau untersucht und auch ausprobiert wurde. Der Algorithmus kann über das GCS aufgerufen werden. Er basiert auf zwei GPS-Punkten, welche vorgängig auf der Karte im GSC platziert werden müssen. Bei den beiden Punkten handelt es sich um den `ApprachFix (AF)` und den `TouchTown (TD)`. Der `TD`-Punkt wird, wie der Name nahelegt, auf der Landebahn platziert, dort, wo die Drohne etwa landen soll. Beim AF-Punkt handelt es sich um den Punkt, welcher ausschlaggebend ist für die Position des Kreises, welchem die Drohne während des Sinkflugs (siehe Abbildung 12.2 Phase 2) folgt. Ausserdem muss dieser in der Verlängerung der Landebahn liegen, da er zusammen mit dem TD-Punkt den Landekurs definiert.

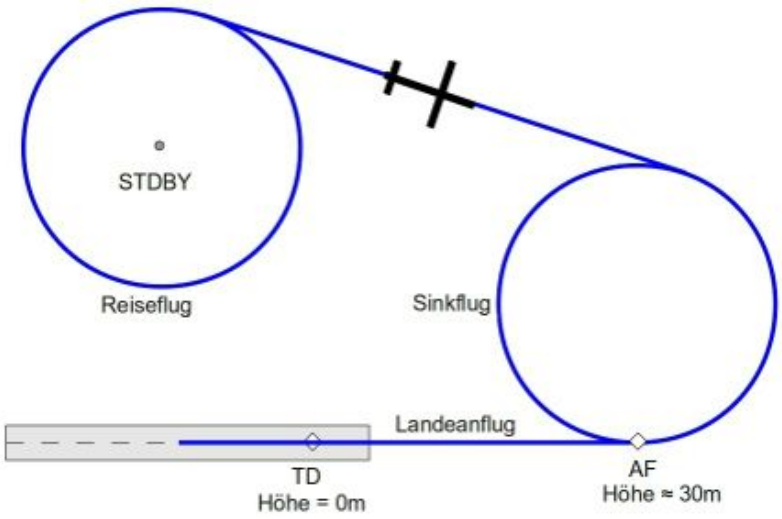


Abbildung 12.3.: Ablauf der vorhandenen Landung

Die Drohne fliegt nach dem Einleiten der Landesequenz als Erstes zu dem Kreis, auf welchem der Sinkflug stattfindet. Wenn sie diesen erreicht hat, sinkt sie so lange, bis die vorgegebene Höhe erreicht ist. Im Anschluss wird ein angefangener Kreis noch fertig geflogen, bis die Drohne auf Landekurs ist. Erst danach wird mit dem eigentlichen Landeanflug begonnen.
Nach dem Passieren des `AF`-Wegpunktes wird bereits der Motor abgeschaltet und die Drohne geht in den Gleitflug über. Ab diesem Zeitpunkt wird die vorgegebene Sollhöhe der Drohne immer nach Ablauf einer gewissen Zeit halbiert. So nähert sich die Drohne immer langsamer dem Boden und die Ungenauigkeit der Höhe über das GPS kann etwas ausgeglichen werden.

12.4. Probleme der vorhandenen Landung

Nachdem die vorhandene Landung einige Male getestet wurde, zeigten sich schnell einige Probleme. Zum einen erwies es sich als äusserst schwierig, die richtige Entfernung zwischen `AF` und `TD` zu finden und zum anderen war es auch nicht einfach, den `TD`-Punkt so zu setzen, dass die Drohne auf der Landebahn aufsetzt.

Ein weiterer Nachteil zeigte sich beim Aufsetzen. Da bei dieser Landung keine Erhöhung des Anstellwinkels (Break) kurz vor dem Aufsetzen gemacht werden kann, landet die Drohne mit der Nase nach unten, was für die UMARS bedeuten würde, mit dem Vorderrad aufzusetzen. Der Grund dafür, dass kein Break gemacht werden kann, ist, dass auf Grund der ungenauen GPS-Höhe nicht bekannt ist, wie hoch die Drohne genau ist.

Ausserdem werden bei der vorhandenen Landung weder der True Airspeed noch der Anstellwinkel geregelt, lediglich die Lageregelung der Drohne ist aktiv. Dadurch hat die Drohne beim Aufsetzten immer noch eine sehr hohe Geschwindigkeit und auch die Sinkgeschwindigkeit ist sehr hoch, wie Abbildung 12.4 zeigt.

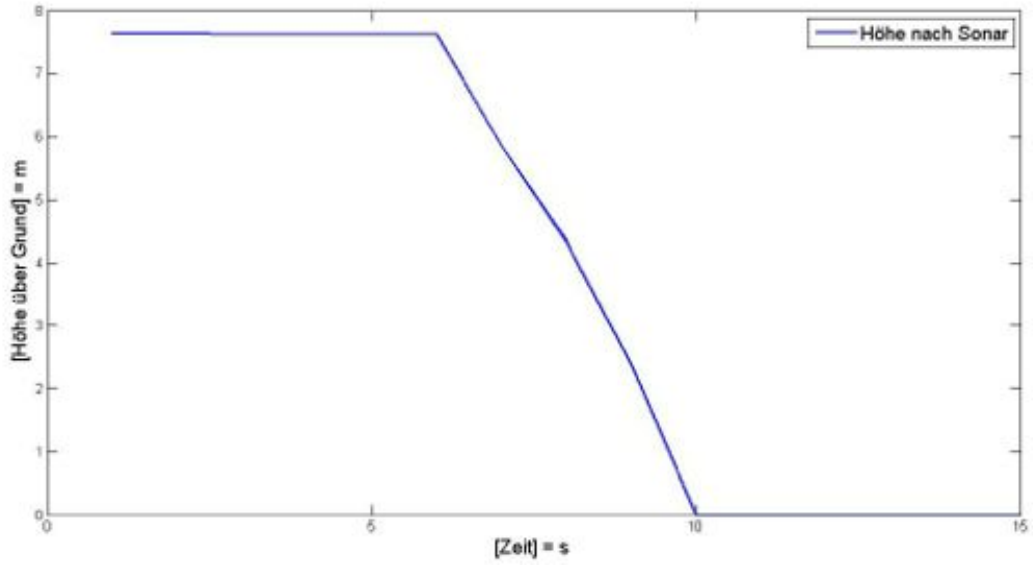


Abbildung 12.4.: Höhenplot der vorhandenen Landung

Da in Abbildung 12.4 die mit dem Sonar gemessene Höhe dargestellt ist, sind nur die letzten 7.5 Höhenmeter aufgezeichnet, da die Reichweite des Sensors begrenzt ist. Die Drohne befand sich aber bereits von Beginn des Plots an im Sinkflug.

12.4.1. Landung mit konstanter Sinkrate

Die Landung mit konstanter Sinkrate ist vor allem für das Landen auf einem Flugzeugträger wichtig, wo es, wie bereits erwähnt, sehr wichtig ist, an der richtigen Stelle auf dem Boden aufzusetzen. Wird diese Stelle verfehlt, kann nicht in das Fangseil, welches das Flugzeug auf einer sehr kurzen Strecke zum Stillstand bringt, eingehakt werden. Da die Sinkrate bis zum Aufsetzen beibehalten wird, entsteht eine erhöhte Belastung im Fahrwerk, auf welche dieses ausgelegt sein muss. Ausserdem entsteht kurz vor dem Aufsetzen ein sogenannter Bodeneffekt. Dieser Effekt ist auf eine Art Luftkissen zurückzuführen, welches unter den Tragflächen ab einer Höhe von ca. 1/3 der Spannweite entsteht. Dieses Luftkissen bewirkt eine Verkleinerung der Sinkrate und ist daher ein Vorteil für diese Landemethode.

1. **Reiseflug**
   - Das Flugzeug fliegt mit Reisegeschwindigkeit und Reisehöhe
   - Landung wurde noch nicht eingeleitet
2. **Sinkflug**
   - Landezone wird angeflogen
   - Motor ist eingeschaltet
3. **Abfangen**
   - Wenn das Flugobjekt seine Sollhöhe erreicht hat, wird es abgefangen
   - Die Sinkrate wird verringert
   - Motor ist eingeschaltet
4. **Flare**
   - Eine konstante Sinkrate wird bis zum Aufsetzen beibehalten
   - Motor wird ausgeschaltet
5. **Aufsetzen**
   - Das Flugzeug setzt auf dem Boden auf
   - Der Bodeneffekt verringert die Sinkrate kurz vor dem Aufsetzen
6. **Ausrollen / Abbremsen**
   - Das Flugobjekt wird bis zum Stillstand abgebremst

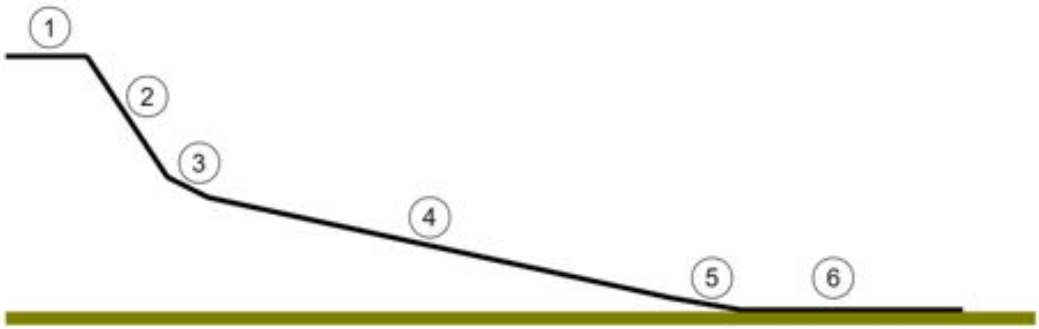


Abbildung 12.5.: Ablauf einer Flugzeugträgerlandung

12.4.2. Landung mit Flare

Die Flarelandung ist eine Landeart, bei der in geringer Höhe über der Landebahn absichtlich ein Strömungsabriss (Stall) erzeugt wird. Nach diesem Strömungsabriss fällt das Flugobjekt die letzten Meter oder Zentimeter bis zum Boden. Dies bedeutet eine erhöhte Belastung für das Fahrwerk. Diese erhöhte Belastung muss im Vorfeld abgeschätzt und das Fahrwerk darauf ausgelegt werden. Ausserdem ist es wichtig, dass eine genaue Höhenmessung verwendet wird, welche garantiert, dass sich das Flugobjekt beim Strömungsabriss nur in geringer Höhe befindet.

1. **Reiseflug**
   - Das Flugzeug fliegt mit Reisegeschwindigkeit und Reisehöhe
   - Landung wurde noch nicht eingeleitet

2. **Sinkflug**
   - Landezone wird angeflogen
   - Motor ist eingeschaltet

3. **Abfangen und Übergang in parallelen Flug zum Boden**
   - Wenn das Flugobjekt seine Sollhöhe erreicht hat, wird es abgefangen
   - Der Flug parallel zum Boden wird eingeleitet
   - Motor ist eingeschaltet

4. **Höhe konstant halten**
   - Eine geringe Höhe, aus welcher ein freier Fall keine Schäden zur Folge hat, wird gehalten
   - Motor wird ausgeschaltet
   - Die Geschwindigkeit des Flugzeuges wird durch Reibung abgebaut

5. **Aufsetzen**
   - Die Strömung über den Tragflächen reisst ab und das Flugzeug setzt auf

6. **Ausrollen / Abbremsen**
   - Das Flugobjekt wird bis zum Stillstand abgebremst

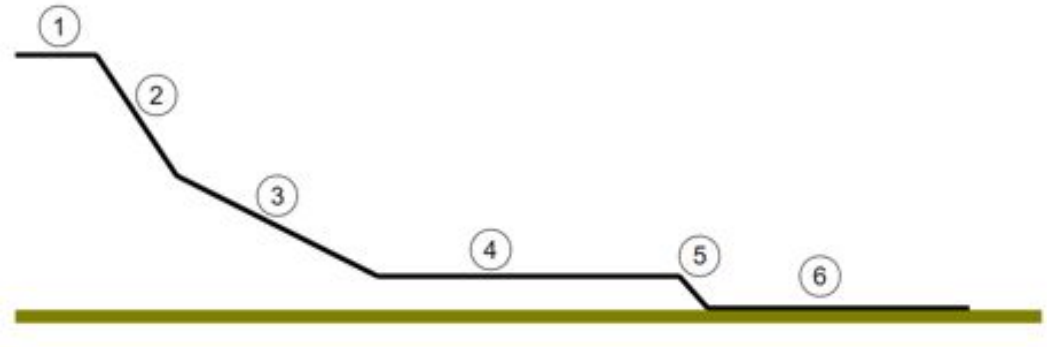


Abbildung 12.6.: Ablauf einer Stalllandung

12.4.3. Vergleich von Landung mit konstanter Sinkrate und Stalllandung

Der Vergleich der beiden Landemethoden zeigt, dass sich diese vor allem in der Aufsetzphase unterscheiden. Während das Flugzeug bei der Landung mit konstanter Sinkrate mit einer relativ hohen Geschwindigkeit aufsetzt, ist diese bei der Stalllandung deutlich kleiner. Die Stalllandung hat also den Vorteil, dass das Flugobjekt einen kürzeren Aussrollweg benötigt. Es ist jedoch schwierig, die Höhe in Schritt vier (siehe Abbildung 12.6) zu regeln. Die Regelung muss sehr schnell und genau sein, da sich das Flugzeug nur noch knapp über dem Boden befindet und eine Abweichung in der Höhe gegen unten, bevor die Drohne über der Landebahn ist, gravierende Auswirkungen hätte.

12.4.4. Schleppgaslandung

Die Schleppgaslandung ist eine im Modellflug verbreitete Landemethode, bei welcher während des Landeanflugs permanent Motorenleistung gegeben wird. So kann die Sinkrate des Flugzeugs verkleinert werden, wodurch der Landeanflug zwar länger wird, aber wenn sich der Pilot in der Höhe verschätzt hat, hat dies weniger gravierende Auswirkungen. Da in dieser Arbeit aber auf die genaue Höhe, die das Sonar liefert, zurückgegriffen werden kann, wird hier nicht näher auf diese Landung eingegangen.

12.5. Einfluss der aktiven Regelsysteme bei der Landung

Zur Wahl der Landemethode gehörte nicht nur der Ablauf, wie er unter Abschnitt 12.4.1, bzw. unter Abschnitt 12.4.2 beschrieben ist, sondern auch, mit welcher Geschwindigkeitsregelung geflogen werden soll. Während der Erarbeitung der autonomen Landung sind zwei unterschiedliche Regelparadigmen entstanden. Diese sollen im Folgenden kurz beschrieben werden.

12.5.1. Version 1, Regelung der Geschwindigkeit und der Höhe

Bei einem ersten Entwurf der Landung sollte die Geschwindigkeit und die Höhe der Drohne geregelt werden. Hierbei sollte darauf geachtet werden, dass solange wie möglich auf eine pitch-basierte Geschwindigkeitsregelung zurückgegriffen wird, da mit dieser Regelung, die Gefahr bei einem Motorausfall einen Strömungsabriss zu fliegen, nicht vorhanden ist. Die Drohne würde in einen kontrollierten Sinkflug übergehen. Eine pitch-basierte Geschwindigkeitsregelung regelt die Geschwindigkeit über das Höhenruder und die Höhe über die Motorleistung. Am Ende der Landung, also sobald die Drohne in Bodennähe kommt, sollte jedoch auf eine direkte Geschwin-

digkeitsregelung, also auf eine, bei der die Geschwindigkeit direkt über die Motorenleistung und die Höhe über das Höhenruder geregelt wird, umgeschaltet werden. Denn im Finale dieser Landeversion sollte absichtlich ein Strömungsabriss in geringer Höhe geflogen werden, nach welchem die Drohne auf dem Boden aufsetzt und die Landung abgeschlossen ist.

12.5.2. Version 2, Regelung des Anstellwinkels und der Höhe

Bei den erstens Tests der Landeversion 1 zeigte sich, dass diese noch einiges Verbesserungspotential hatte. Dadurch, dass nur die Geschwindigkeit und die Höhe geregelt wurden und nicht der Anstellwinkel, war die Landegeschwindigkeit noch immer sehr hoch bzw. es resultierte ein sehr langer Landeanflug. Ausserdem sind die verwendeten Regelungen vom Gewicht der Drohne abhängig. Es müsste also bei einem erhöhten Gewicht auch die Sollgeschwindigkeit angepasst werden, da die Landegeschwindigkeit mit zunehmendem Gewicht steigt. Da sich das Gewicht der UMARS häufig ändert, sollte eine gewichtsunabhängige Regelung entwickelt werden.
Eine solche gewichtsunabhängige Regelung ist möglich, wenn der Anstellwinkel (Pitch) und die Höhe der Drohne geregelt werden. Eine solche Regelung hat gleich mehrere Vorteile: Zum einen ist sie, wie schon erwähnt, gewichtsunabhängig und zum anderen besteht keine Gefahr für einen ungewollten Strömungsabriss. Ein Strömungsabriss tritt immer dann auf, wenn ein gewisser Anstellwinkel erreicht wird, und dieser ist ausschliesslich von der Geometrie des Flugzeuges anhängig. Die Anstellwinkelregelung wurde so umgesetzt, dass die Höhe über die Motorenleistung und der Pitch über das Höhenruder geregelt wird. Bei einer Erhöhung des Gewichts der Drohne spielt sich nun folgendes Szenario ab:
Durch das erhöhte Gewicht sinkt die Drohne bei gleichbleibender Geschwindigkeit nun schneller als vorher. Dadurch wird die Sollhöhe unterschritten, worauf die Regelung mit einer Erhöhung der Motorenleistung reagiert. Bei gleichbleibendem Soll-Anstellwinkel stellt sich also bei einer Erhöhung des Gewichts automatisch ein höherer True Airspeed ein. Dadurch wird die Regelung gewichtsunabhängig.
Die genaue Umsetzung der Landung mit Regelung des Anstellwinkels ist unter Punkt 12.7.2 beschrieben.

Zur Verdeutlichung, welcher Winkel genau unter Anstellwinkel oder auch Pitch zu verstehen ist, dient die folgende Abbildung 12.7.

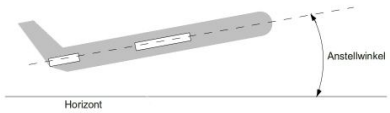


Abbildung 12.7.: Darstellung des Anstellwinkels, bzw. des Pitches

12.6. Entscheidung der Landemethode und der aktiven Regelsysteme

Bevor eine Entscheidung getroffen werden konnte, welche Landemethode nun wirklich besser ist, mussten diese ausprogrammiert und getestet werden, da es nur aufgrund der Theorie äusserst schwierig ist, zu beurteilen, bei welcher Vorgehensweise die Drohne das stabilere Flugverhalten zeigt. Ausserdem unterscheiden sich die beiden Versionen im zur Umsetzung notwendigen C-Code nur in der Schlussphase und in den parallel zum Lande-Code laufenden Geschwindigkeitsregelungen.

Nachdem dann aber beide Argorithmen programmiert und getestet waren, war klar, dass die Landung mit dem geregelten Anstellwinkel bessere Ergebnisse erzielt. Sie ergab nicht nur eine stabilere Fluglage des Flugzeugs, sondern es war auch möglich den Anstellwinkel kurz vor der Landung zu erhöhen, was zur Folge hat, dass das Aufsetzen sanfter und die Geschwindigkeit zu diesem Zeitpunkt kleiner ist. Dieser Unterschied zeigt sich auch, wenn man die Höhenplots der beiden Landungen miteinander vergleicht (Abbildung 12.8). Ausserdem ist die Erzeugung eines Strömungsabrisses immer mit einem erhöhten Risiko verbunden, da ein zu früh einsetzender Abriss fatale Folgen haben würde. Daher wird diese Landung in der konventionellen Fliegerei auch nicht eingesetzt. Aus diesen Gründen soll für die Portierung auf die UMARS die Landung mit Regelung des Anstellwinkels verwendet werden.

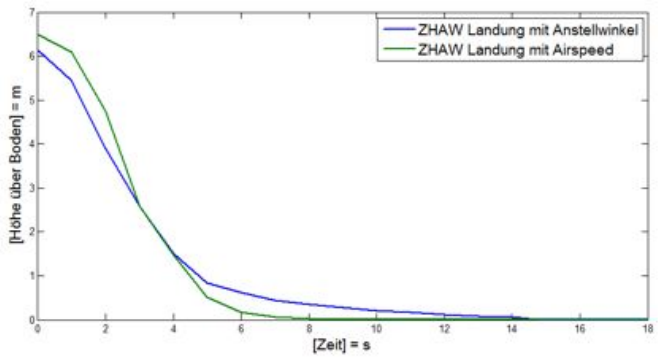


Abbildung 12.8.: Vergleich der Höhenplots der Anstellwinkel- bzw. Geschwindigkeitsgeregelten Landung kurz vor dem Aufsetzten

12.7. Umsetzung der Landung mit Ultraschall

Die Umsetzung der Landung mit Ultraschallsensor war mit einigem Programmieraufwand verbunden. Es konnte zwar grundsätzlich auf der vorhandenen Landung aufgebaut werden, aber es mussten einige neue Schritte eingefügt werden. Ausserdem war es für beide Versionen nötig, dass die Sonarhöhe nicht nur ausgelesen wird, sondern auch als Regelgrösse verwendet werden kann. Das in Paparazzi bereits vorhandene File `estimator.h`, welches die Höhe aus dem GPS ausliest und der Regelung zur Verarbeitung weitergibt, musste so modifiziert werden, dass auf die Höhe des Ultraschallsensors umgeschaltet werden konnte. Des weiteren mussten die Regelsysteme, die bei der Landung zum Einsatz kommen, im Vorfeld sehr genau ausparametriert werden, vor allem, um sicherzustellen, dass die Regelabweichung der Höhenregelung nicht zu gross wird, da sonst ein Absturz droht.

12.7.1. Version 1, mit Flare

Die Landung mit Geschwindigkeitsregelung, Höhenregelung und einem absichtlichen Stall kurz vor dem Aufsetzen wurde bei der programmatischen Umsetzung in sieben Schritte unterteilt. Diese sieben Schritte sind: `CircleDown`, `LandingWait`, `ApproachHeading`, `DeclineToSonar`, `Approach`, `Flare` und `Stall`.

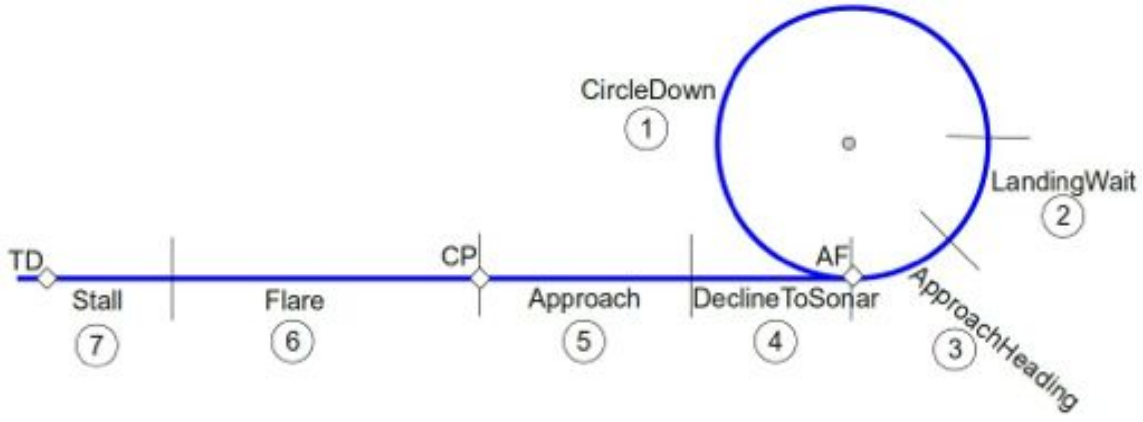


Abbildung 12.9.: Darstellung der Landeabschnitte (Draufsicht)

Die Abbildung 12.9 zeigt die sieben Schritte der Landung mit Strömungsabriss. Im Folgenden soll auf jeden Schritt etwas näher eingegangen werden.

Schritt 1, `CircleDown`

Das erste Manöver der Landung ist ein Kreis, welcher geflogen wird, bis die Drohne von ihrer ursprünglichen Höhe auf eine Höhe, die vom Benutzer des Autopiloten während des Flugs gesetzt werden kann, gesunken ist. Dabei ist es durchaus möglich, dass die Drohne mehrere Kreise fliegt, bis sie die vorgegebene Höhe erreicht hat und damit in den nächsten Schritt gehen kann. Als Regelung für den True Airspeed sollte hier die indirekte `Pitch_Simple`-Regelung zum Einsatz kommen.

Schritt 2, `LandingWait`

Wenn die vorgegebene Höhe erreicht wird, solange die Drohne noch nicht auf Landekurs ist, wird in den Schritt `LandingWait` gegangen, in welchem die Höhe gehalten wird und weiter dem vorgegebenen Kreis gefolgt wird. Nähert sich die Drohne dem Landekurs, wird in den Schritt drei, `ApproachHeading` übergegangen.

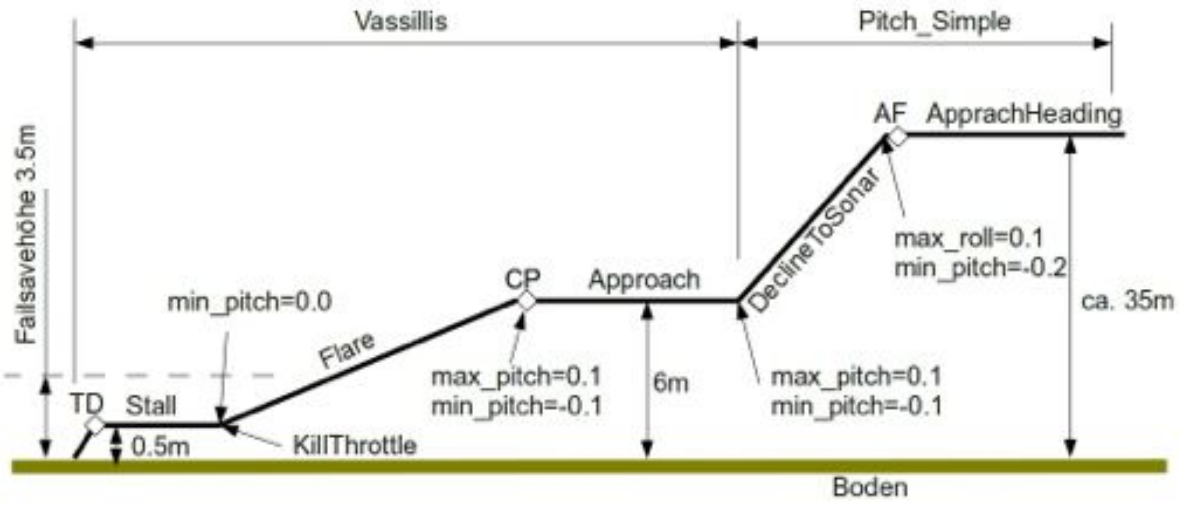


Abbildung 12.10.: Darstellung der Landeabschnitte der geschwindigkeitsgeregelten Landung (Seitenansicht)

Die Abbildung 12.10 zeigt den Höhenverlauf einer Landung mit Geschwindigkeitsregelung. Die Abbildung dient der genaueren Erläuterung der Schritte 3 bis 7. Zu beachten ist, dass die Schritte 1 und 2 nicht dargestellt sind.

Schritt 3, `ApprachHeading`

In diesem meist kurzen Zwischenschritt wird die Drohne nun genau auf den Landekurs ausgerichtet. Ausserdem werden am Schluss des Schritts, bevor in Schritt vier gewechselt wird, einige für die Landung notwendige Lagebegrenzungen gesetzt. Zum einen wird der maximale Rollwinkel der Drohne auf 0.1 rad begrenzt. Dies wird gemacht, damit die Drohne in Bodennähe nicht so stark navigieren kann, dass ein Flügel den Boden touchiert, was fatale Auswirkungen hätte. Zum anderen wird der Pitch-Winkel gegen unten, also der maximale Sinkwinkel auf 0.2 rad beschränkt, um zu verhindern, dass die Drohne im nächsten Schritt zu schnell sinkt.

Schritt 4, `DeclineToSonar`

In diesem Schritt sinkt die Drohne nun so lange, bis der Ultraschallsensor eine Höhe von weniger als 6.5 m ausgibt. Dieser Abstand von 6.5 m hängt mit der maximalen Reichweite des Sensors zusammen und sollte so gross wie möglich gewählt werden. Da der hier verwendete Sensor eine Reichweite von maximal 7.5 m hatte, wurde hier dieser spezifische Wert verwendet. Bevor die Höhe von weniger als 6.5 m erreicht ist, wird die vom GPS errechnete Höhe für die Höhenregelung verwendet. Sobald diese Höhe nun aber unterschritten wird und der Ultraschallsensor ein zuverlässiges Signal ausgibt, wird die Höhenregelung auf dieses Signal umgeschaltet. Bevor nun in den nächsten Schritt gewechselt wird, wird noch die neue maximale Pitch-Auslenkung gesetzt und die Geschwindigkeitsregelung wird auf `Vassillis`, eine direkte Regelung, umgeschaltet. Diese Regelung kann Fehler in der Höhe schneller korrigieren.

Schritt 5, `Approach`

Da die horizontale Strecke über Boden, die bei Schritt 4 zurückgelegt wird, von den gegebenen Windverhältnissen abhängt, musste hier, um die Landebahn sicher zu erreichen, ein Zwischenschritt eingefügt werden. In diesem Zwischenschritt wird die Höhe der Drohne über das Signal des Sonars geregelt und es wird auf einen fixen GPS-Punkt zugeflogen. Sobald dieser Punkt passiert wird, wird in den Schritt `Flare` gewechselt.

Schritt 6, `Flare`

In diesem Schritt folgt die Drohne einem Gleitpfad. Ziel ist es, mit einer konstanten Sinkrate bis auf eine Höhe von etwa 0.5 m zu sinken. Ist diese Höhe erreicht, wird der Motor abgeschaltet und der maximale Sinkwinkel auf 0.0 rad begrenzt. Dadurch geht der Bug der Drohne nach oben, was für ein Aufsetzen auf den Hinterrädern wichtig ist.

Schritt 7, `Stall`

Im letzten Schritt wird der Drohne eine Sollhöhe von 0.5 m gegeben, welche sie versucht zu halten. Dabei baut sich die kinetische Energie aufgrund des Luftwiderstands ab und die Drohne wird immer langsamer. Um die Höhe aber bei einer kritisch langsamen Geschwindigkeit dennoch halten zu können, muss sich der Bug immer mehr heben, was letztlich zu einem Strömungsabriss über den Tragflächen führt, worauf die Drohne auf dem Boden aufsetzt und die Landung abgeschlossen ist.

Failsave

Bei der Landung können zwei verschiedene Fehler auftreten, welche auch abgefangen werden. Ein möglicher Fehler ist, dass die GPS-Höhe, mit welcher die Schritte 1 bis 4 geflogen werden, sehr ungenau ist und sich die Drohne daher schon in einem dieser Schritte dem Boden auf weniger als die Sicherheitshöhe nähert. Sollte dieser Fall eintreten, würde dies von dem Ultraschallsensor detektiert und die Landung würde abgebrochen. Die Sicherheitshöhe kann vor dem Start der Drohne im `Ariframe` definiert werden.
Ein weiterer möglicher Fehler ist, dass die Drohne in Schritt 4 zu langsam sinkt und daher den GPS-Punkt `CP`, bereits während diesem Schritt überfliegt. Auch in diesem Fall würde die Landung vom Autopiloten abgebrochen und der Bediener der `Ground Control Station` hat die Möglichkeit, die Punkte `AF` und `TD` neu zu setzen, bevor eine weitere Landung eingeleitet wird.

12.7.2. Version 2, mit Anstellwinkelregelung

Bei der Umsetzung der autonomen Landung mit geregeltem Anstellwinkel konnte als Grundlage die erste Version mit dem Strömungsabriss verwendet werden. Diese wurde dann aber so angepasst, dass kurz vor dem Aufsetzen kein Strömungsabriss mehr entsteht, sondern mit einer immer kleiner werdenden Sinkrate gelandet wird. Dadurch konnte die Landung in sechs Schritten ausgeführt werden. Diese Schritte wurden bezeichnet mit: `CircleDown`, `LandingWait`, `ApproachHeading`, `DeclineToSonar`, `Approach` und `Flare`.

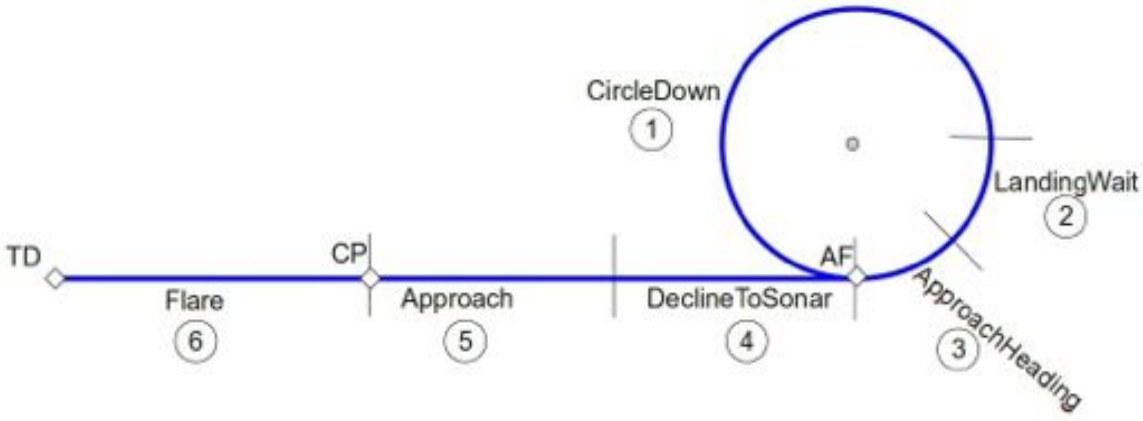


Abbildung 12.11.: Darstellung der Landeabschnitte (Draufsicht)

Die Abbildung 12.11 zeigt, dass sich am grundsätzlichen Ablauf der Landung nichts geändert hat. Lediglich der letzte Schritt konnte bei dieser Version der Landung weggelassen werden. Im Folgenden sollen nun die einzelnen Schritte der Landung genauer beschrieben werden.

Schritt 1, `CircleDown`

Nach dem Einleiten der Landesequenz wird auf die Geschwindigkeitsregelung `Airspeed_FixedPitch` umgeschaltet, welche zum einen den Anstellwinkel (Pitch) der Drohne über das Höhenruder regelt und zum anderen die Höhe über die Leistung des Motors. Die Drohne kreist nun um einen Punkt, bis sie die vorgegebene Höhe zur Einleitung des Landeanflugs erreicht hat.

Schritt 2, `LandingWait`

Wenn die Drohne die vorgegebene Höhe erreicht hat, folgt sie weiter dem Kreis, bis sie sich dem Landekurs nähert. Dieser Zwischenschritt könnte bei einer weiteren Überarbeitung der Landung weggelassen werden, wurde hier aber noch verwendet.

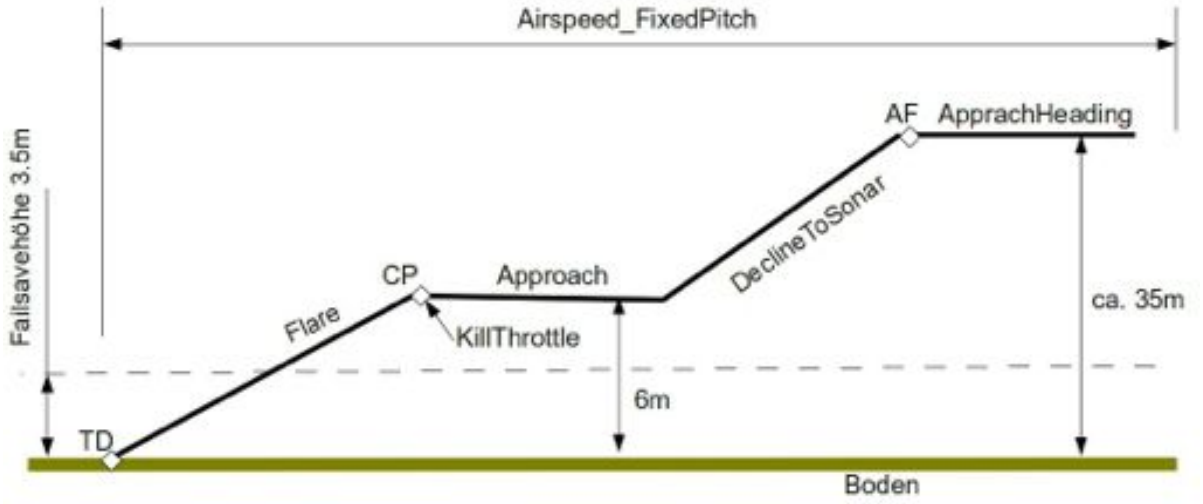


Abbildung 12.12.: Darstellung der Landeabschnitte der anstellwinkelgeregelten Landung (Seitenansicht)

Die Abbildung 12.12 zeigt den Höhenverlauf einer Landung mit Anstellwinkelregelung. Die Abbildung dient der genaueren Erläuterung der Schritte 3 bis 6. Zu beachten ist, dass die Schritte 1 und 2 nicht dargestellt sind.

Schritt 3, `ApproachHeading`

In diesem Schritt wird weiterhin die Höhe gehalten und die Drohne folgt so lange dem vorgegebenen Kreis, bis sie genau auf den Landeanflug ausgerichtet ist.

Schritt 4, `DeclineToSonar`

Nun beginnt der eigentliche Landeanflug auf die Piste. Die Drohne sinkt wie bei der Version 1 der Landung solange, bis das Sonar einen zuverlässigen Wert misst. Danach wird auf die Regelung der Sonar-Höhe umgeschaltet und in den nächsten Schritt gegangen.

Schritt 5, `Approach`

In diesem Schritt wird, genau wie unter Punkt 12.7.1, Abschnitt `Approach` beschrieben, die Höhe gehalten und auf einen fixen GPS-Punkt zugeflogen. Ist dieser Punkt passiert, wird der Motor abgeschaltet.

Schritt 6, `Flare`

Erst der letzte Teil dieser Landung zeigt signifikante Unterschiede zu der unter Punkt 12.7.1 beschriebenen Landung. Da hier mit einer Regelung des Anstellwinkels der Drohne geflogen wird, kann mit einem konstanten Sollwert für die Sinkrate bis zum Aufsetzen geflogen werden. Um jedoch ein noch etwas sanfteres Aufsetzen zu erhalten, wird der Sollwert des Anstellwinkels in Bodennähe, das heisst etwa ab einer Höhe von 0.5 m, stetig erhöht, was ein Abflachen der Flugbahn und ein sanftes Aufsetzen zur Folge hat. Dadurch entsteht das sogenannte Flaren des Fliegers kurz vor dem Aufsetzen und es wird sichergestellt, dass die hinteren Räder zuerst Bodenkontakt erhalten.

Failsave

Da der grundsätzliche Verlauf der Landung gleich geblieben ist, konnten auch die Methoden zur Fehlerabfangung von der Landung mit Strömungsabriss übernommen werden.

12.7.3. Vorteile der anstellwinkelgeregelten gegenüber der geschwindigkeitsgeregelten Landung

Schon beim Vergleich der Abbildungen 12.10 und 12.12 zeigt sich, dass für letztere deutlich weniger Bedingungen während des Programmablaufs gesetzt werden müssen. Ausserdem ist ein Umschalten zwischen zwei verschiedenen Geschwindigkeitsregelungen, welches immer eine kurze Höhen- und Geschwindigkeitsabweichung zur Folge hat, nicht nötig. Die gesamte Landung kann mit derselben Regelung durchflogen werden. Ausserdem zeigten die Tests, dass das Verhalten der Drohne mit einer Regelung des Anstellwinkels deutlich ruhiger und stabiler ist. Und die Anstellwinkelregelung ist gewichtunabhängig. Es stellt sich also automatisch bei einem erhöhten Gewicht der Drohne ein erhöhter True Airspeed ein.

12.8. Ergebnisse

Die beste Veranschaulichung der Ergebnisse sind die Videoaufnahmen der verschiedenen Landungen, welche auf der dieser Arbeit beiliegenden CD zu finden sind. Um aber noch genauere Aussagen über die Sinkgeschwindigkeit und den True Airspeed bei der Landung zu machen, dienen die folgenden Grafiken.

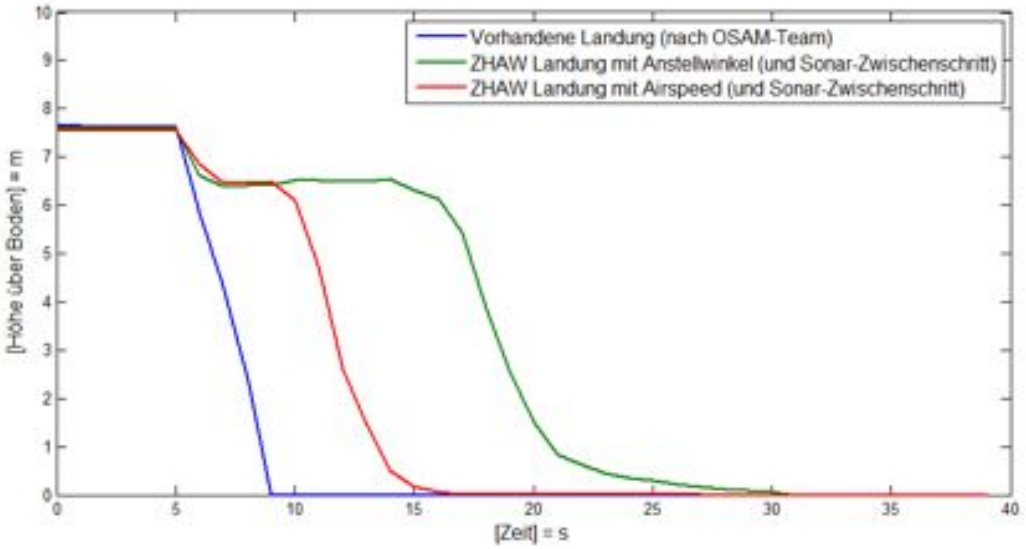


Abbildung 12.13.: Vergleich der drei getesteten Landeverfahren

In der oben stehenden Abbildung 12.13 sind die Höhenmesswerte des Ultraschallsensors bei allen drei getesteten Landeverfahren im Vergleich dargestellt. Dabei ist zu beachten, dass die Messwerte erst bei ca. 7.5 m beginnen, da dies die maximale Reichweite des verwendeten Sensors ist. Es ist deutlich zu sehen, dass bei der vorhandenen Landung eine konstante Sinkrate geflogen wird, welche sich am Ende nicht abflacht. Dies ist darauf zurückzuführen, dass diese Landung ohne Ultraschallsensor ausgeführt wird. Daher ist gar nicht bekannt, wann die Drohne kurz vor dem Boden ist. Beim Höhenverlauf der ZHAW Landung mit Airspeedregelung ist ein erstes Abflachen der Höhenkurve zu erkennen. Da bei dieser Landemethode jedoch der Anstellwinkel nicht geregelt wird, ist dieses Abflachen noch etwas zaghaft. Dies bedeutete für unsere Übungsdrohne, dass sie nach dem ersten Aufsetzen noch einige Male wieder in die Höhe sprang, bevor sie endgültig zum Stillstand kam. Dieses Hüpfen ist auf den Videos `ZHAW_Landung_1_2011-05-04.m2ts` und `ZHAW_Landung_2_2011-05-04.m2ts` auf der beiliegenden CD zu sehen.

Einzig bei der Landung mit Anstellwinkelregelung zeigt sich eine deutliche Verminderung der Sinkrate vor dem Aufsetzen. Dank dieser speziellen Regelung ist es hier möglich, den Sollwert des

Anstellwinkels in Abhängigkeit der Höhe der Drohne zu verändern. Also je tiefer die Drohne ist, desto grösser wird der Sollwert für den Anstellwinkel. So wurde ein sehr sanftes Aufsetzen der Drohne erreicht, wie auf den Videos `ZHAW_Landung_3_2011-05-06.m2ts` und `ZHAW_Landung_4_2011-05-06.m2ts` ersichtlich ist. Ein leichtes Hüpfen ist jedoch immer noch zu sehen, aber dieses ist auf das stark federnde Fahrwerk zurückzuführen. Eine etwas schwerere Drohne mit einem gedämpften Fahrwerk würde den Kontakt zum Boden nach erstmaliger Berührung nicht mehr verlieren.

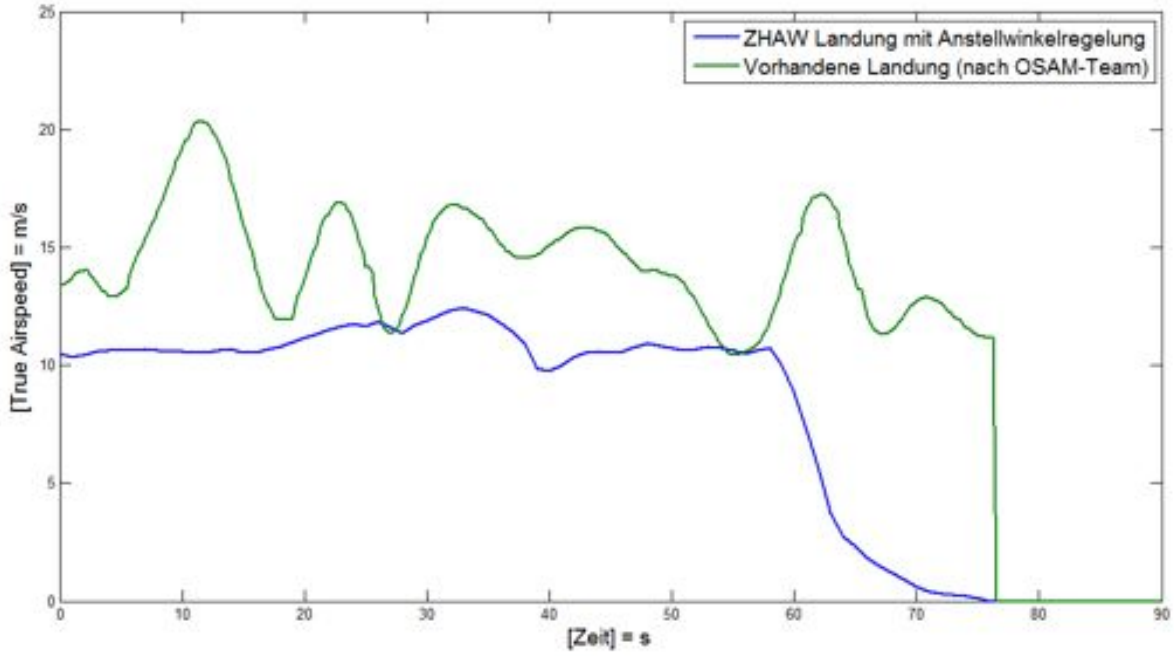


Abbildung 12.14.: Vergleich des True Airspeeds bei der Landung

Macht man nun noch einen Vergleich der Landegeschwindigkeit der bereits vorhandenen Landung und der neu entwickelten mit Anstellwinkelregelung so stellt man fest, dass bei der neuen ein deutliches "Bremsen"kurz vor dem Aufsetzen zu sehen ist (siehe Abbildung 12.14). Dieses Bremsen entsteht durch die oben angesprochene Erhöhung des Anstellwinkels und hat zur Folge, dass die Drohne nach dem Aufsetzen nur noch einen kurzen Ausrollweg benötigt. Bei der Landung, welche bereits im Paparazzi-Projekt eingebunden ist, ist kein solches Minimieren der Geschwindigkeit zu erkennen. Hier ist allerdings anzumerken, dass der abrupte Rückgang der Geschwindigkeit von ca. 12 m/s auf 0 m/s auf die Landung im hohen Grass zurückzuführen ist, welche durch das Verfehlen der Landebahn entstand. Daher fehlte hier das Ausrollen der Drohne auf dem Rollfeld. In den ersten 50 s des Plots ist ausserdem zu sehen, dass die Geschwindigkeit bei der Landung mit Anstellwinkelregelung um einiges konstanter ist als bei der Landung vom OSAM-Team[1]

[1] Entwicklungsteam der Utah State University

12.8.1. Position AF und Anflughöhe bestimmen

Eine nicht ganz einfache Aufgabe, die vor einer erfolgreichen Landung gelöst werden muss, ist die Bestimmung der Position und der Höhe des `TD`-Wegpunktes. Da die Drohne von diesem Punkt aus gerade auf die Landebahn zufliegt, ist an erster Stelle wichtig, dass der Wegpunkt in der Verlängerung der Landebahn ist. Es sollte so vorgegangen werden, dass die Entfernung einmal festgelegt wird und dann eine eher zu grosse Höhe eingeben wird. Dies hat den Effekt, dass die Drohne über der Landebahn eher zu hoch ist und die Landung automatisch abgebrochen wird. Dann kann in einem zweiten Anlauf die Höhe des `TD`-Wegpunktes herabgesetzt werden, bis die Landung durchgeführt werden kann. Abbildung 12.15 zeigt die Positionen der entsprechenden Punkte für die Übungsdrohne Mentor bei wenig bis keinem Gegenwind auf dem Flugplatz im Lauchetal.

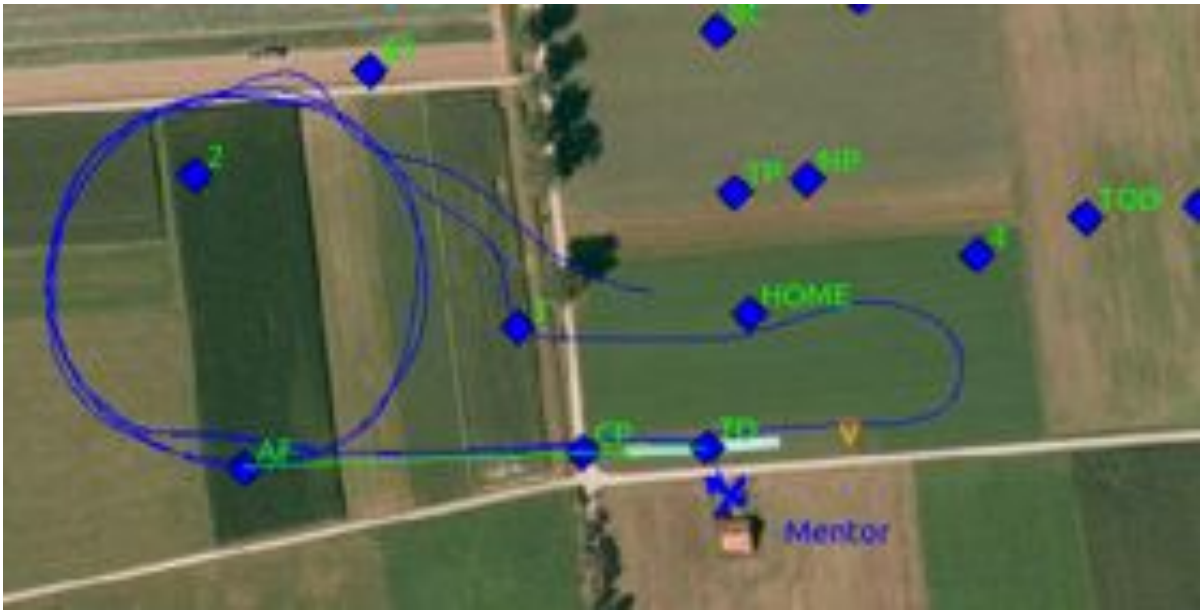


Abbildung 12.15.: Konfiguration für die Landung vom Mentor

Sind die Einstellungen einmal gefunden, können diese grösstenteils immer beibehalten werden. Die Strecke über dem Boden, welche die Drohne für den Sinkflug benötigt, hängt aber vom Wind ab. Bei starken Gegenwind wird diese immer kürzer und der `TD`-Wegpunkt sollte näher zur Landebahn geschoben werden. Um die Einstellungen immer problemlos auf die gegebenen Umstände anzupassen, benötigt es einige Erfahrung im Umgang mit der autonomen Landung.

12.9. Checkliste für die Landung

Die folgende Checkliste sollte vor jeder Landung abgearbeitet werden, um eine sichere Landung zu garantieren.

- Pitch-Sollwert für `Fixed_Pitch`-Regelung beim Landen setzen (im `Airframe`, vor dem Start)
- Bodenhöhe in `TD`-Wegpunkt eintragen
- `TD`-Wegpunkt auf der Landebahn platzieren
- Anflughöhe in `AF`-Wegpunkt eintragen
- `AF`-Wegpunkt positionieren (siehe Punkt 12.8.1)
- Gewünschte Drehrichtung für `CircleDown`-Kreis bestimmen
- Einleiten der Landung über Lande-Button

13. Testflüge

In diesem Kapitel soll beschrieben werden, wie bei den zahlreichen Test-und Prametrierflügen vorgegangen wurde und welche Hilfsmittel verwendet wurden.

13.1. Flugplatz

Während dieser Arbeit konnten Testflüge auf dem Modellflugplatz im Lauchetal nahe Lommis durchgeführt werden. Hier gibt es ein kleines Haus mit Stromversorgung, eine 100 m lange Start- und Landebahn und viel Platz zum Fliegen.

Abbildung 13.1.: Modellflugplatz Lauchetal

13.2. Vorbereitung

Im Vorfeld eines Flugtages wurden jeweils alle Vorkehrungen getroffen, um einen reibungslosen Ablauf zu garantieren. Die zu testenden Neuerungen wurden schriftlich auf einem Flugplan festgehalten (siehe Anhang C.1). Dann wurde das benötigte Material bereitgelegt und die Akkus für die Drohne und die Fernbedienung geladen. Was die Software anbelangt, wurde stets vor einem Flugtag ein Update des GIT-Projektes (siehe Kapitel 16) gemacht.

13.3. Parametrierflüge

Zu Beginn der Arbeit lag der Hauptschwerpunkt der Tests noch auf den neuen Geschwindigkeitsregelungen. Es galt, die insgesamt fünf neuen Regelparadigmen zu parametrieren und zu optimieren. Diese Aufgabe stellte sich schnell als langwierig und mühsam heraus, da die Parameter durch Ausprobieren gefunden werden mussten. So konnten aber mit einiger Übung die Grobeinstellungen für einen ruhigen Flug gefunden werden. Doch für das Feintuning war ein Hilfsmittel gefordert, mit welchem eine objektive Aussage über die Regelung gemacht werden konnte. Daher wurde das Modul `Benchmark`, näher beschrieben unter Punkt 15.4.3, programmiert. Jetzt konnte bestimmt werden, ob sich das Ändern eines Parameters positiv oder negativ auf das Verhalten auswirkte. Fortan wurde ein Oval geflogen, während welchem das `Benchmark`-Modul den Verlauf der Fehlerquadratsumme der Regelgrössen aufzeichnete. Die momentanen Parameter während einem solchen Oval-Flug wurden auf dem Flugprotokoll (siehe Anhang C.2) festgehalten. Die Abbildung 13.2 zeigt ein Beispiel einer solchen Ausgabe.

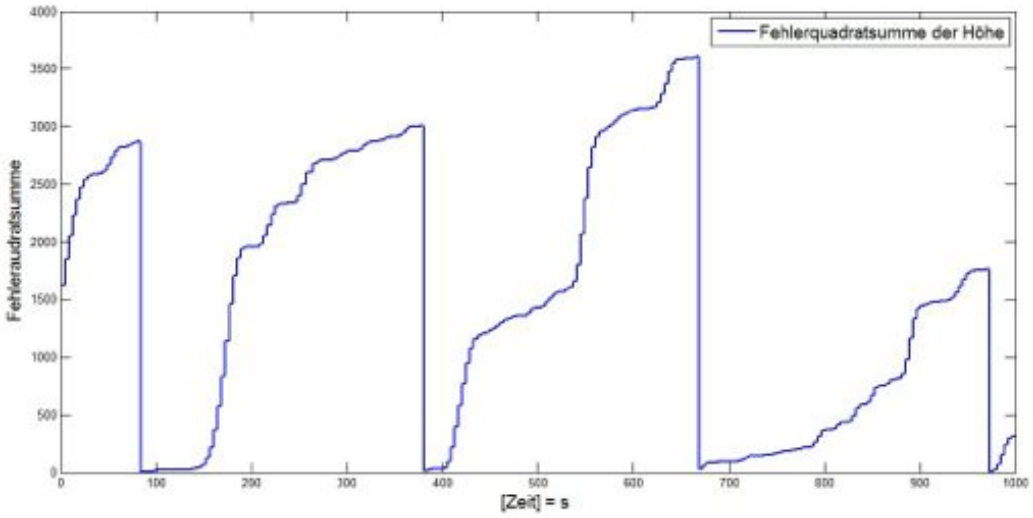


Abbildung 13.2.: Ausgabe des Benchmark-Moduls bei Oval-Flug

13.4. Test der Startroutinen

Im weiteren Verlauf der Arbeit wurde dann das autonome Starten der Drohne immer wichtiger. Um zu überprüfen, ob sich kein Programmierfehler in den Code eingeschlichen hatte, wurden "Trockenstarts"durchgeführt. Bei einer solchen Simulation des Startvorgangs wurde alles Vorbereitet wie bei einem richtigen Start, nur dass die Drohne nicht an das Bungeeseil gehängt wurde. Um die Drohne zu beschleunigen, wurde sie getragen. So konnte zumindest getestet werden, ob

der Motor unter den für den Start notwendigen Voraussetzungen zum richtigen Zeitpunkt starten würde. War dies gelungen, war der Test erfolgreich abgeschlossen.

13.5. Test der Landeroutine

Bei der Landung ist ein Test mit Hilfe von Tragen der Drohne wie beim Start nicht möglich oder zumindest nur die letzten Meter einer Landung. Und auch auf diesem Metern zeigt sich das Verhalten der Drohne, welches in dieser Phase am wichtigsten ist, nicht. Daher wurden bei der Landung nur grundlegende Tests am Boden durchgeführt und danach wurde die Landung von A bis Z mit der Übungsdrohne Mentor getestet, denn im Notfall konnte hier immer noch auf die manuelle Steuerung umgeschaltet werden.

14. Flugpattern

Ein Flugpattern dient dazu, einen kompletten Messflug durchzuführen, ohne dass während des Flugs Befehle von der Bodenstation an die Drohne gesendet werden müssen. Im Normalfall setzt sich ein solches Pattern wie folgt zusammen:

1. Start
2. Steigflug auf Höhe für Messbeginn
3. Drohne auf Kurs für Messstrecke bringen
4. Folgen der Messstrecke
5. Ende der Messstrecke
6. Wiederholen der Punkte 3 bis 5, bis der Messflug abgeschlossen ist
7. Landung

Die Messstrecke kann je nach Anforderungen der Messung variieren und muss vorgängig im Flugplan programmiert werden. Ein Beispiel für ein solches Flugpattern wurde während dieser Arbeit geschrieben und wird im Folgenden näher beschrieben.

14.1. Programmierung eines kompletten Flugplans mit Flugpattern

Nun folgt ein kompletter Flugplan, mit Start, Messflug in Form einer Acht und anschliessender Landung. Vom Bediener müssen vor dem Start lediglich die entsprechenden GPS-Punkte an ihre vorgegebene Position gesetzt werden. Der Bediener hat aber darauf zu achten, dass diese Punkte, sollte sich der Wind während des Fluges drehen, angepasst werden.

```
<block name="BungeeTakeoffGlide" strip_button="BTO_Glide (wp TOD)"
  strip_icon="bungee_launch.png">
   <call fun="set_start_params()"/>
   <call fun="InitializeZHAWBungeeTakeoff(WP_TOD, WP_TP)"/>
   <call fun="ZHAWBungeeTakeoff_glide()"/>
   <go from="TP" hmode="route" vmode="glide" wp="TOD"/>
   <call fun="set_measure_params()"/>
   <deroute block="Figure 8 8T to 8C"/>
</block>
```

In diesem Block wird der Bungeestart durchgeführt. Die Startparameter werden geladen und dem Startalgorithmus werden die zwei Punkte `WP_TOD` und `WP_TP` übermittelt. Nach dem Beschleunigen wird ein Gleitflug eingeleitet. Die Drohne folgt diesem Gleitpfad, bis die im Airframe vordefinierte Sollhöhe erreicht wird. Danach wird in den nächsten Block gewechselt.

```
<block key="F8" name="Figure 8 8T to 8C" strip_button="Figure 8 (wp 1-2)"
  strip_icon="eight.png" group="base_pattern">
    <circle radius="nav_radius" wp="CLIMB" alt="ground_alt+200"
      until="ground_alt+195 > estimator_z"/>
    <eight center="8C" radius="nav_radius" turn_around="8T" alt="ground_alt+200"
      until="nav_eight_count > 0"/>
    <eight center="8C" radius="nav_radius" turn_around="8T" alt="ground_alt+150"
      until="nav_eight_count > 0"/>
    <eight center="8C" radius="nav_radius" turn_around="8T" alt="ground_alt+100"
      until="nav_eight_count > 0"/>
    <deroute block="FixedPitch Land Right AF-TD"/>
</block>
```

Nach dem Start wird um den Punkt `CLIMB` gekreist und auf eine Höhe von 200 m über Boden geflogen. Danach wechselt die Drohne in ein Achterpattern, welches dreimal auf verschiedenen Höhen durchflogen wird. Das Achterpattern wird durch die zwei Punkte `8C` und `8T` beschrieben. Nach dem Durchfliegen des Patterns wird in die Landung gewechselt.

```
<block group="land" name="FixedPitch Land Right AF-TD"
  strip_button="FixedPitch Land right (wp AF-TD)" strip_icon="land-right.png">
  <call fun="set_approach_params()"/>
  <set value="DEFAULT_CIRCLE_RADIUS" var="nav_radius"/>
  <call fun="InitializeZHAWSkidLanding(WP_AF, WP_TD, WP_CP, nav_radius)"/>
  <call fun="ZHAWSkidLanding()"/>
  <go from="AF" hmode="route" vmode="glide" wp="FSP"/>
  <deroute block="Standby"/>
</block>
```

Der Landeanflug wird mit `Fixed Pitch` durchgeführt. Es muss unbedingt darauf geachtet werden, dass die Punkte des Landeanfluges unter Beachtung der Windrichtung korrekt gesetzt werden. Die Drohne wird dann in eine Abwärtsspirale navigieren und den Landeanflug einleiten.
Zur Veranschaulichung, wie das oben stehende Flugpattern dreidimensional aussehen würde, dient die folgende Abbildung 14.1.

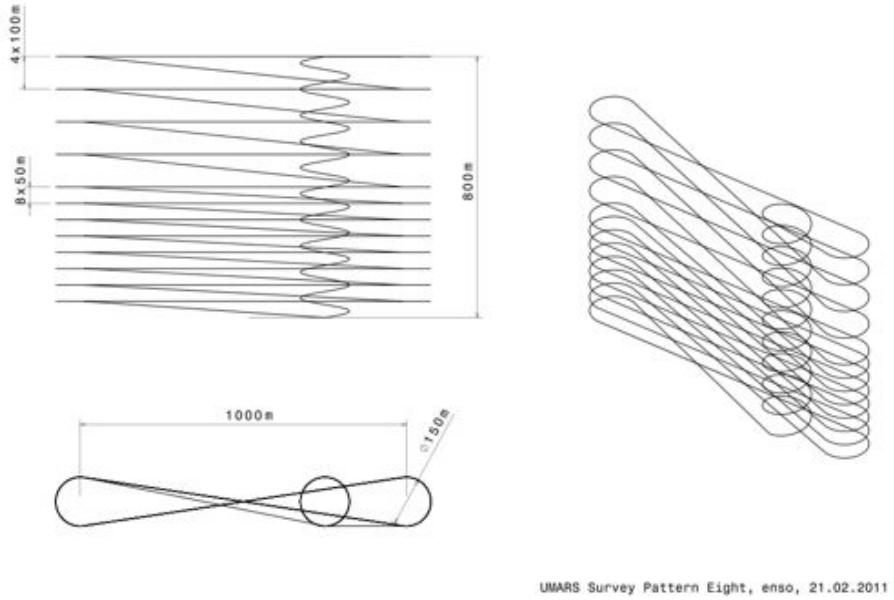


Abbildung 14.1.: Survey Pattern Eight

14.2. Beispiel eines Funnel-Patterns

Ein weiteres Beispiel eines Flugpatterns zeigt einen sogenannten Funnel. Der Start und die Landung wurden hier aber weggelassen.

```
<block name="Circle" strip_button="Circle" strip_icon="home.png">
  <circle radius="nav_radius" wp="CIRCLE" alt="ground_alt+300"
    until="estimator_z > ground_alt+295"/>
  <circle radius="nav_radius+200" wp="CIRCLE" alt="ground_alt+300"
    until="NavCircleCount() > 0"/>
  <circle radius="nav_radius" wp="CIRCLE" alt="ground_alt+250"
    until="ground_alt+255 > estimator_z"/>
  <circle radius="nav_radius+180" wp="CIRCLE" alt="ground_alt+250"
    until="NavCircleCount() > 0"/>
  <circle radius="nav_radius" wp="CIRCLE" alt="ground_alt+200"
    until="ground_alt+205 > estimator_z"/>
  <circle radius="nav_radius+160" wp="CIRCLE" alt="ground_alt+200"
     until="NavCircleCount() > 0"/>
  <circle radius="nav_radius" wp="CIRCLE" alt="ground_alt+150"
     until="ground_alt+155 > estimator_z"/>
  <circle radius="nav_radius+140" wp="CIRCLE" alt="ground_alt+150"
    until="NavCircleCount() > 0"/>
  <circle radius="nav_radius" wp="CIRCLE" alt="ground_alt+100"
    until="ground_alt+105 > estimator_z"/>
  <circle radius="nav_radius+120" wp="CIRCLE" alt="ground_alt+100"
    until="NavCircleCount() > 0"/>
  <deroute block="Standby"/>
</block>
```

Das im oben dargestellten Code beschriebene Flugpattern ist kreisförmig, wobei der Radius des Kreises sich mit abnehmender Höhe verkleinert. Die Drohne würde, wenn sie nach diesem Pattern fliegen würde, der in der folgenden Abbildung 14.2 dargestellten Trajektorie folgen.

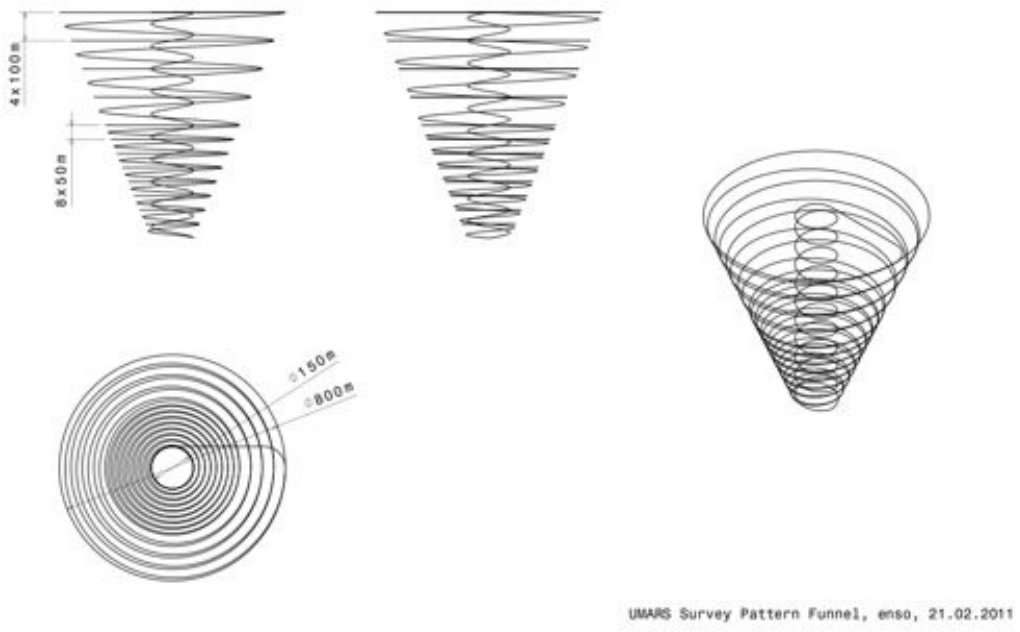


Abbildung 14.2.: Survey Pattern Funnel

Mit der Kombination der zwei hier gezeigten Beispiele sollte es möglich sein, beliebige Trajektorien zu planen. Alternativ zu `eight` kann auch `oval` verwendet werden. Jedoch müssen dann die dazu passenden Variablen für die Rundenzahl (`nav_oval_count`) angepasst werden.

15. Programmierung

In diesem Projekt wurde sehr viel neuer Sourcecode geschrieben und verändert. Dieses Kapitel befasst sich mit der Beschreibung aller geschriebenen Module und Algorithmen. Der aktuellste Stand des Projekts steht unter folgender Adresse zum Download bereit:

`https://github.com/Bruzzlee/paparazzi`

Im Kapitel 16 wird das Grundwissen über GIT vermittelt.

15.1. Strukturierung der Parameter im GCS

Im GCS ist es möglich, während des Flugs bestimmte Parameter zu verstellen. Diese Parameter werden beim Paparazzi-Projekt Settings genannt. Diese Settings werden beispielsweise in der Datei `Mentor_Se_2.xml` definiert. Neu ist es möglich, die Settings Modulweise zusammenzufügen. Dies ist sehr praktisch, da die Settings immer auf Variablen zugreifen, welche auch vorhanden sind beziehungsweise kompiliert wurden. Wird beispielsweise ein Modul ausgeschaltet, findet der Organisator der Settings die Variabel nicht mehr.

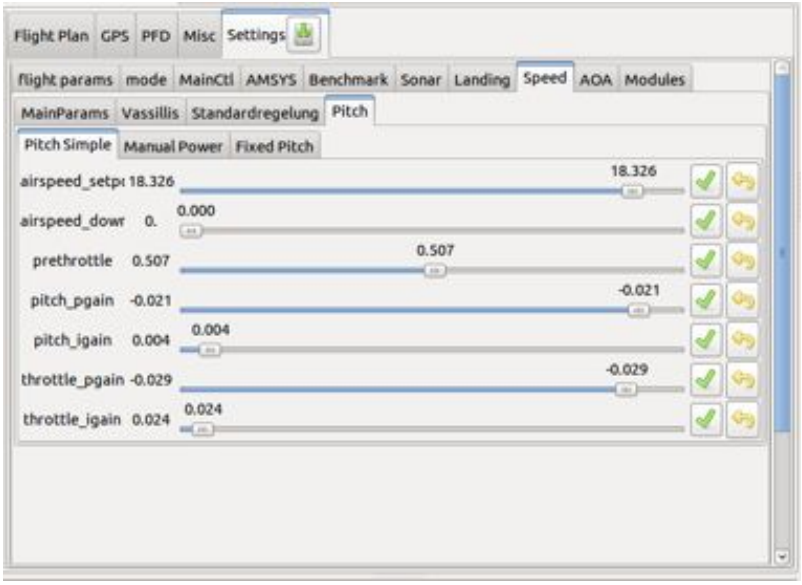


Abbildung 15.1.: Ansicht des Settingmenüs im GCS

15.1.1. Standardsettings - `Mentor_Se_2.xml`

- `flight params` - Standard Flugparameter wird für Standardregelung benötigt
  - `flight_altitude` - Flughöhe
  - `wind_east` - Windgeschwindigkeit von Osten
  - `wind_north` - Windgeschwindigkeit von Norden
- `mode` - Umschalten verschiedener Modi
  - `pprz_mode` - Manual / Auto 1 / Auto 2
  - `alt_kalman_enabled` - Kalmanfilter für die Höhenmessung aktivieren
  - `estimator_flight_time` - Flugzeit rücksetzen
  - `launch` - starten
  - `kill_throttle` - Motor hart ausschalten
  - `gps_reset` - GPS rücksetzen
- `MainCtl` - Haupteinstellungen
  - ins - IMU Einstellungen
    - `ins_roll_neutral` - IMU Roll Offset
    - `ins_pitch_neutral` - IMU Pitch Offset
  - `attitude` - Flugverhalten
    - `h_ctl_pitch_min_setpoint` - Minimaler Pitch (nach unten)
    - `h_ctl_pitch_max_setpoint` - Maximaler Pitch (nach oben)
    - `h_ctl_pitch_pgain` - Pitch Proportionalfaktor
    - `h_ctl_pitch_dgain` - Pitch Differentialfaktor
    - `h_ctl_elevator_of_roll` - Pitch-Anteil bei Kurvenflug
    - `h_ctl_roll_max_setpoint` - Maximaler Rollwinkel
    - `h_ctl_roll_attitude_gain` - Roll Proportionalfaktor
    - `h_ctl_roll_rate_gain` - Roll Differentialfaktor
    - `h_ctl_aileron_of_throttle` - Rotordrehmomentkompensation
  - `nav`
    - `h_ctl_course_pgain` - Kurs Proportionalfaktor
    - `h_ctl_course_igain` - Kurs Integralfaktor
    - `h_ctl_course_dgain` - Kurs Differentialfaktor
    - `h_ctl_course_pre_bank_correction` - Kurs Vorhaltekorrektur
    - `nav_glide_pitch_trim` - Pitch-trimmung für Gleitverhalten
    - `h_ctl_roll_slew` - Übergang in Schräglage beim Einfliegen in eine Kurve
    - `nav_radius` - Navigationsradius
    - `nav_course` - Benutzerdefinierte Flugrichtung in Grad, Norden = 0.0
    - `nav_mode` - Umschaltung auf windunabhängige Navigation
    - `nav_climb` - Benutzerdefinierte Steigrate festlegen m/s

        * `fp_pitch` - Pitch-Variabel für Flugplan in Grad
        * `nav_shift` - Kursabweichung festlegen
        * `nav_ground_speed_setpoint` - GPS Sollgeschwindigkeit
        * `nav_ground_speed_pgain` - Proportionalfaktor der GPS Geschwindigkeit
        * `nav_survey_shift` - Abstand der Linien im Überwachungsflug-Modus

15.1.2. Settings der Drucksensoren - `amsys.xml`

- `AMSYS`
  - `Airspeed`
    * `airspeed_scale` - Faktor, mit welchem die Rohdaten multipliziert werden
    * `airspeed_filter` - Filterrate
  - `Baro`
    * `baro_filter` - Filterrate

15.1.3. Settings des Benchmarkmoduls - `benchmark.xml`

- `Benchmark`
  - `benchm_reset` - Rücksetzen der Fehlerquadratsumme
  - `benchm_go` - Benchmark starten
  - `ToleranceAispeed` - Toleranzband für die Geschwindigkeit
  - `ToleranceAltitude` - Toleranzband für die Flughöhe
  - `TolerancePosition` - Toleranzband für die Position

15.1.4. Sonarsettings - `adc_sonar.xml`

- `Sonar`
  - `sonar_offset` - Distanzoffset
  - `sonar_filter` - Filterrate

15.1.5. Landesettings - `Landing.xml`

- `Landing`
  - `estimator_z_mode` - Zwischen Sonar und GPS Höhe wechseln

15.1.6. Settings der Geschwindigkeitsregelungen - `airspeedSwitch_PitchVas.xml`

- `Speed` - Originale Regelsysteme
  - `MainParams`
    * `v_ctl_airspeed_mode` - Geschwindigkeitsregelung auswählen
    * `v_ctl_auto_groundspeed_setpoint` - Minimale GPS Geschwindigkeit
    * `v_ctl_auto_groundspeed_pgain` - Proportionalfaktor der GPS Geschwindigkeit

    - `v_ctl_auto_groundspeed_igain` - Integralfaktor der GPS Geschwindigkeit
  - `Vassillis`
    - `v_ctl_auto_airspeed_setpoint` - Sollgeschwindigkeit
    - `v_ctl_auto_airspeed_deadband` - Totzone
    - `v_ctl_auto_airspeed_pitch_pgain_v` - Pitch Proportionalfaktor
    - `v_ctl_auto_airspeed_pitch_igain_v` - Pitch Integralfaktor
    - `v_ctl_auto_airspeed_throttle_pgain_v` - Motor Proportionalfaktor
    - `v_ctl_auto_airspeed_throttle_igain_v` - Motor Integralfaktor
  - `Standardregelung`
    - `v_ctl_altitude_pgain` - Proportionalfaktor der Höhenregelung
    - `v_ctl_auto_throttle_cruise_throttle` - Definition für normale Motorleistung
    - `v_ctl_auto_throttle_pgain` - Motor Proportionalfaktor
    - `v_ctl_auto_throttle_igain` - Motor Integralfaktor
    - `v_ctl_auto_throttle_climb_throttle_increment` - Motorleistungssteigerung beim Steigen (feed forward)
    - `v_ctl_auto_throttle_pitch_of_vz_pgain` - Pitch Proportionalfaktor
    - `v_ctl_auto_throttle_pitch_of_vz_dgain` - Pirch Differentialfaktor

- `Pitch` - Neue Pitchregelsysteme
  - `Pitch Simple`
    - `v_ctl_auto_airspeed_setpoint` - Sollgeschwindigkeit
    - `v_ctl_auto_airspeed_deadband` - Totzone
    - `v_ctl_auto_airspeed_prethrottle_asps` - Normale Motorleistung (feed forward)
    - `v_ctl_auto_airspeed_pitch_pgain_asps` - Pitch Proportionalfaktor
    - `v_ctl_auto_airspeed_pitch_igain_asps` - Pitch Integralfaktor
    - `v_ctl_auto_airspeed_throttle_pgain_asps` - Motor Proportionalfaktor
    - `v_ctl_auto_airspeed_throttle_igain_asps` - Motor Integralfaktor
  - `Manual Power`
    - `v_ctl_auto_airspeed_setpoint` - Sollgeschwindigkeit
    - `v_ctl_auto_airspeed_deadband` - Totzone
    - `v_ctl_auto_airspeed_throttlesetp_asmp` - Sollwert der Motorenleistung
    - `v_ctl_auto_airspeed_pitch_pgain_asmp` - Pitch Proportionalfaktor
    - `v_ctl_auto_airspeed_pitch_igain_asmp` - Pitch Integralfaktor
  - `Fixed Pitch`
    - `v_ctl_auto_airspeed_pitchsetp_fp` - Sollwert des Pitchs
    - `v_ctl_auto_airspeed_prethrottle_fp` - Normale Motorleistung (feed forward)
    - `v_ctl_auto_airspeed_throttle_pgain_fp` - Motor Proportionalfaktor
    - `v_ctl_auto_airspeed_throttle_igain_fp` - Motor Integralfaktor

15.2. Änderungen des Standardcodes

Für die Geschwindigkeitsregelung wurden verschiedene Änderungen in schon vorhandenen Dateien gemacht. Ebenso gilt dies für die Landung und den Bungeestart. Folgende Liste beschreibt die Änderungen und deren Ziel.

15.2.1. `main_ap.c`

- Ziel der Änderung
  - Flugmodus Auto 1 soll dieselben Begrenzungen von Pitch und Roll benutzen wie diejenigen, welche für Flugmodus Auto 2 definiert sind.
- Pfad
  - `sw/airborne/firmwares/fixedwing`

15.2.2. `guidance_v.c`

- Ziel der Änderung
  - Hier wurden verschiedene Änderungen durchgeführt, welche mit der Geschwindigkeitsregelung zusammenhängen, unter anderem die Regelungsumschaltung.
- Pfad
  - `sw/airborne/firmwares/fixedwing/guidance`
- Settings Datei
  - `airspeedSwitch_PitchVas.xml`

    Hierbei handelt es sich um eine optimierte Version, bei der unnötige Parameter herausgenommen wurden, um den Datenverkehr zu optimieren.

15.2.3. `stabilization_attitude.c`

- Ziel der Änderung
  - Bei dieser Datei wurde die Kursregelung von einem PD-Regler auf einen PID-Regler erweitert
  - Der `h_ctl_pitch_mode` - Umschalter wurde in dieser Datei implementiert. Dieser wird für das Regeln mit dem Angle of Attack Sensor benötigt.
- Pfad
  - `sw/airborne/firmwares/fixedwing/stabilization`

15.2.4. `estimator.c`

- Ziel der Änderung
  - Umschaltung der Höhenmessung von GPS zu Sonar
- Pfad
  - `sw/airborne`

15.2.5. `nav.c`

- Ziel der Änderung
  - Achterfigur-Methode für Flugpattern wurde so verändert, dass die Acht beliebig lang definiert werden kann. Ebenso wurde eine Zählervariable definiert, welche die Umrundungen der Achterfigur beinhaltet.
- Pfad
  - `sw/airborne/subsystems`

15.2.6. `gen_flight_plan.ml`

- Ziel der Änderung
  - Um bei der Achterfigur im Flightplan eine Bedingung mittels `until` zu realisieren, musste diese Datei angepasst werden.
- Pfad
  - `sw/tools`

15.3. Neue Start- und Ladealgorithmen - ZHAWNav

- Ort der Code- und Headerdatei: `sw/airborne/subsystems/navigation`
- Settings Datei: `Landing.xml`

In dem oben angegebenen Ordner befinden sich mehrere Dateien, die mit `ZHAWNav_` beginnen. In diese Dateien wurden die neuen Start und Landealgorithmen geschrieben, die bei dieser Arbeit entwickelt wurden. Da bei der Landung die beste Methode klar definiert werden konnte, existiert für die Landung nur noch ein File. Beim Start stehen noch zwei verschiedene Methoden zur Auswahl, daher existieren für diesen noch zwei Dateien. Momentan sind diese so in den Autopiloten eingebunden, dass beide Startmethoden und die Landemethode vom `GCS` her aufgerufen werden können.

15.4. Anleitung der neuen Module

Das überarbeitete Paparazzi-Projekt arbeitet mit Modulen. Das heisst, es ist mittels Airframe möglich, gewisse Codeteile modular ein- und auszuschalten. Diese Programme funktionieren meist geschlossen und sollten den übrigen Code nicht beeinflussen. Alle Module sind als separate XML-Dateien verfügbar, in welchen die Aufruffrequenz und die Informationen für den Compiler, wie zum Beispiel der Pfad zur c-Datei, stehen. Die XML-Dateien der Module befinden sich im Verzeichnis `conf/modules`.

15.4.1. AMSYS Airspeed

- ModulXML: `airspeed_amsys.xml`
- Ort der Code- und Headerdatei: `sw/airborne/modules/sensors`
- Settings Datei: `amsys.xml`

Dieses Modul liest über die I2C Schnittstelle die Differenzdruckwerte des AMSYS Drucksensors aus. Die Adresse des Sensors muss auf `0xF4` konfiguriert sein. Die Druckwerte werden, wie in Abschnitt 8.1.3 beschrieben, mittels einem PT1-Filter bearbeitet, bevor sie an den `estimator` weitergeleitet werden.
Ist der Parameter `USE_AIRSPEED` nicht definiert, werden keine True Airspeed Informationen an den `estimator` weitergeleitet.
Das Modul wurde so programmiert, dass durch das Definieren des Parameters `MEASURE_AMSYS_TEMPERATURE` im Airframe die Temperatur des Drucksensors zusätzlich übertragen wird. Ist dieser Parameter definiert, werden jedoch doppelt so viele Daten über die I2C Schnittstelle angefordert.
Ist eine grössere Frequenz der Geschwindigkeitsdaten, welche als Nachrichten dem GCS übermittelt werden, erwünscht, kann der Parameter `SENSOR_SYNC_SEND` im Airframe definiert werden.

15.4.2. AMSYS Baro

- ModulXML: `baro_amsys.xml`
- Ort der Code- und Headerdatei: `sw/airborne/modules/sensors`
- Settings Datei: `amsys.xml`

Mit diesem Modul werden die Absolutdruckdaten aus dem AMSYS Drucksensor gelesen und verarbeitet. Die Daten werden über die I2C Schnittstelle von der Adresse `0xF2` angefordert.
Die Druckdaten werden beim Einschalten in den ersten Sekunden gespeichert und gemittelt, was einen automatischen Offset ergibt. Dieser Offset wird dann von den folgenden gemessenen Druckdaten abgezogen. Aus der Differenz wird dann der Höhenunterschied errechnet. Um die Höhe über Meer zu berechnen, wird die Bodenhöhe aus dem Flugplan addiert. Alternativ dazu kann die GPS Höhe beim Einschalten dazu verwendet werden.
Später soll der Absolutdrucksensor für die Höhenregelung verwendet werden. Da die Sensoren sich jedoch noch in der Testphase befinden, werden lediglich die Messdaten aufgenommen, um später zu entscheiden, ob die UMARS danach geregelt werden soll. Aus diesem Grund werden die Daten nicht an den `estimator` weitergeleitet. Sollte in einem nächsten Schritt eine Regelung

nach Absolutdruck zu realisieren sein, muss das `estimator.c`-File angepasst und gegebenenfalls ein Kalman-Filter verwendet werden.
Ist der Parameter `SENSOR_SYNC_SEND` im Airframe gesetzt, werden die Daten in derselben Frequenz wie das Modul aufgerufen wird (10 Hz) an das GCS gesendet. Dies kann jedoch zur Überlastung des Funkmodems führen, was zur Folge hat, dass andere Daten nicht mehr übermittelt werden. Zu beachten ist ebenfalls, dass mit `SENSOR_SYNC_SEND` auch mehr Geschwindigkeitsdaten übermittelt werden, da das Modul `airspeed_amsys` auf denselben Parameter programmiert wurde. Der Parameter `SENSOR_SYNC_SEND` wird nur für die Fehlersuche oder für eine Bodenmessung benötigt.

15.4.3. Benchmark

- ModulXML: `benchmark.xml`
- Ort der Code- und Headerdatei: `sw/airborne/modules/benchmark`
- Settings Datei: `benchmark.xml`

Dieses Modul erlaubt eine quantitative Bewertung des Fluges. Es errechnet die Fehlerquadratsummen des zweidimensionalen Kurses (x/y), der Flughöhe und des True Airspeed. Die Fehlerquadratsummen des Kurses und der Flughöhe wurden getrennt, da diese getrennt geregelt werden und somit von verschiedenen Parameter abhängig sind. Das Modul wurde für die Optimierung der Regelparameter geschrieben und wurde auch schon erfolgreich dazu eingesetzt. Die Messung wird erst gestartet, wenn über die Settings die Variabel `benchm_go` auf 1 gesetzt wird.
Die Fehlerquadratsummen können durch Setzen der booleschen Variabel `benchm_reset` auf Null gesetzt werden. Dies lässt sich auch in einem Flugplan durchführen, was ein Flugplan mit automatischem Fehlerquadratsummen-Reset ermöglicht. Es ist also möglich, beispielsweise ein Oval zu fliegen und die Fehlerquadratsumme an immer derselben Stelle automatisch zurücksetzen zu lassen.
Dem Modul können im Airframe folgende Parameter definiert werden:

```
<define name="BENCHMARK_AIRSPEED"/>
<define name="BENCHMARK_ALTITUDE"/>
<define name="BENCHMARK_POSITION"/>
<define name="BENCHMARK_TOLERANCE_AIRSPEED" value="0."/>
<define name="BENCHMARK_TOLERANCE_ALTITUDE" value="0."/>
<define name="BENCHMARK_TOLERANCE_POSITION" value="0."/>
```

Die BENCHMARK_* Parameter aktivieren die entsprechende Berechnungen, wobei die BENCHMARK_TOLERANCE_* Parameter die Toleranz festlegen, in welcher die Fehlerquadratsumme nicht aufsummiert wird.
Im Kapitel 13.3 wird beschrieben, wie mit mit Hilfe des Benchmark-Modules die Parameter optimiert wurden.

15.4.4. ADC Sonar

- ModulXML: sonar_adc.xml
- Ort der Code- und Headerdatei: sw/airborne/modules/sonar
- Settings Datei: adc_sonar.xml

Das Sonar-Modul ermöglicht das ermitteln der Distanz eines Sonar-Sensors, welcher an einer analogen Eingangsschnittstelle anschlossen wurde. Die Parameter können wie folgt im Airframe definiert werden:

```
<configure name="ADC_SONAR" value="ADC_5"/>
<define name="SONAR_ADC_OFFSET" value="0"/>
<define name="SONAR_ADC_SCALE" value="0.0172"/>
<define name="SONAR_ADC_FILTER" value="0.9"/>
```

Mit ADC_SONAR wird beschrieben, welcher analoge Eingang abgerufen werden soll. Mit SONAR_ADC_OFFSET kann ein Distanzoffset definiert werden, was bis jetzt jedoch nicht benötigt wurde. Mit SONAR_ADC_SCALE wird der Multiplikator festgelegt, welcher das Volt / Distanz Verhältnis beschreibt. Die Distanz wird in Meter gemessen. Das Sonarmodul filtert die Werte mit demselben Prinzip des PT1-Filters wie das Airspeed-Modul. Mit SONAR_ADC_FILTER kann dessen Filterrate gesetzt werden.
Dieses Modul wird benötigt, wenn bei der Landung auf die Sonarhöhe umgeschaltet werden soll. Dazu muss jedoch der Parameter USE_SONAR im Airframe definiert sein.

15.4.5. AOA adc

- ModulXML: AOA_adc.xml
- Ort der Code- und Headerdatei: sw/airborne/modules/sensors
- Settings Datei: AOA_adc.xml

Dieses Modul misst die Positionswerte des Angle of Attack Sensors und rechnet diese in die Einheit rad um. Das Modul wird wie die IMU-Abfrage mit 60 Hz aufgerufen. Der errechnete Winkel

wird an das GCS gesendet und im estimator unter der Variabel estimator_AOA gespeichert. Der Winkel kann während des Flugs mittels eines Offsets so verändert werden, dass der Flieger bei einem schnellen, geraden Flug ungefähr Null anzeigt. Damit nicht jedes Zittern geregelt wird, wurde bei diesem Modul ebenfalls mit einem PT1-Filter gearbeitet. Diese Werte können im Airframe wie folgt vordefiniert werden.

```
<configure name="ADC_AOA" value="ADC_6"/>
<define name="AOA_OFFSET" value="0.0"/>
<define name="AOA_FILTER" value="0.0"/>
```

15.4.6. ArduIMU

Die ArduIMU ist eine IMU, welche mit drei Beschleunigungssensoren sowie drei Gyros die Lage des Flugzeuges errechnen kann. Der ArduIMU-Chip besitzt einen eigenen Prozessor inklusive Speicher. Ebenso beinhaltet der Chip einen EEPROM-Speicher, welcher Daten dauerhaft abspeichern kann. Auf der ArduIMU läuft ein separates Programm (Firmware), welches unabhängig zu Paparazzi funktioniert. Damit die IMU ihre Ausrichtung errechnen kann, ist es nötig, dass die GPS-Informationen vom Paparazzi-Chip an die ArduIMU gesendet werden. Diese Algorithmen sowie die ArduIMU-Firmware wurden im Laufe des Projekts von der Paparazzi-Community bereinigt und erweitert.
Mit der erneuerten Firmware wurde es möglich, die Referenzdaten, welche generiert werden, sobald die ArduIMU eingeschaltet wird, im EEPROM zu speichern, sodass beim nächsten Start die Offsets der Lage des Flugzeugs aus diesem Speicher gelesen werden und das Flugzeug beim Einschalten nicht horizantal ausgerichtet werden muss. Somit müssen die Offsets nur einmal bestimmt und gespeichert werden und beim Einschalten der Drohne auf dem Flugplatz muss nicht darauf geachtet werden, dass diese gerade steht. Um diese Funktion zu aktivieren, muss folgender Wert in der ArduIMU Firmware auf 1 gesetzt werden:

```
#define ENABLE_AIR_START 1
```

Die ArduIMU kann so eingerichtet werden, dass ein Hardwareschalter an den Chip angelötet wird, welcher es ermöglicht, das einmalige Referenzieren durch Betätigen dieses Schalters zu aktualisieren. Der Pin, an welchem der Schalter angebracht wird, wird wie folgt definiert:

```
#define GROUNDSTART_PIN 8
```

Die Drohne UMARS wird mittels Gummiseil (Bungee) gestartet. Dabei können Beschleunigungen bis zu 3 g auftreten. Da diese Beschleunigungen in die x-Richtung des Flugzeugs wirken, wird der

Theta-Winkel (Winkel, welcher die horizontale Lage entlang des Flugzeugs zum Boden beschreibt) verfälscht. Aus diesem Grund wurde in der ArduIMU-Firmware die GPS Beschleunigung von der x-Beschleunigung abgezogen. Jedoch belegten Tests, dass das GPS zu langsam ist, um solch grosse Beschleunigungen zu kompensieren.
Aus diesem Grund wurde ein Angle of Attack Sensor entwickelt, welcher die Theta-Messung der IMU am Start ersetzen soll.
Eine weitere Möglichkeit, dieses Problem in den Griff zu bekommen, ist eine Softwarelösung, welche die Beschleunigungssensoren beim Start deaktivieren. Die IMU berechnet dann den Winkel Theta nur noch mit den Gyros, welche jedoch einen Drift (mit der Zeit auftretender Fehler) aufweisen. Auf Grund dieses Drifts dürfen die Beschleunigungssensoren erst kurz vor dem Start ausgeschaltet und danach wieder eingeschaltet werden. Eine ähnliche Methode wird momentan von der Paparazzi-Community entwickelt und befindet sich noch in den ersten Testphasen.

Magnetometer

Damit die Ausrichtung des Flugzeugs auch im Stillstand erkannt werden kann und um Driftfehler um die z-Achse zu kompensieren, wurde die Drohne mit einem dreidimensionalen Magnetometer, auch 3D Kompass genannt, ausgerüstet. Nach dem Umbau der Elektronik in die kompakte Kiste wurde versucht, dieses Bauteil wieder in Betrieb zu nehmen.
Die Daten sollten vom Magnetometer gelesen und an die ArduIMU gesendet werden. Jedoch lieferte der Magnetometer stets konstannte Werte. Abklärungen haben folgende mögliche Ursachen ergeben:

- Magnetometer Chip defekt
- I2C Kanal überfordert
- I2C Kanal zu lang (Kabel zu lang)
- Stabilisierungswiderstände erforderlich

Aus Zeitgründen wurde bezüglich dieses Problems nicht weitergeforscht. Der Einsatz eines Magnetometers ist momentan nicht nötig, da der Startvorgang so programmiert wurde, dass eine falsche Ausrichtung des Flugzeugs im Stillstand keine Einfluss auf den Steigvorgang hat. Die Ausrichtung wird mittels GPS korrigiert, sobald sich die Drohne in eine Richtung bewegt.

16. GIT

Abbildung 16.1.: GIT Logo

Bei diesem Projekt wurde sehr viel neuer Source-Code programmiert und bereits vorhandener verändert. Damit der Überblick der Versionen nicht verloren ging, hat man sich für ein VCS (Version Control System) entschieden. Da das Open Source Projekt Paparazzi ausschliesslich über die VCS-Lösung GIT operiert, wurde für dieses Projekt dieselbe Lösung übernommen. Als GIT Server wurde `https://github.com/` gewählt. Das momentane Projekt ist unter `https://github.com/Bruzzlee/paparazzi` zu finden.

16.1. Beschreibung

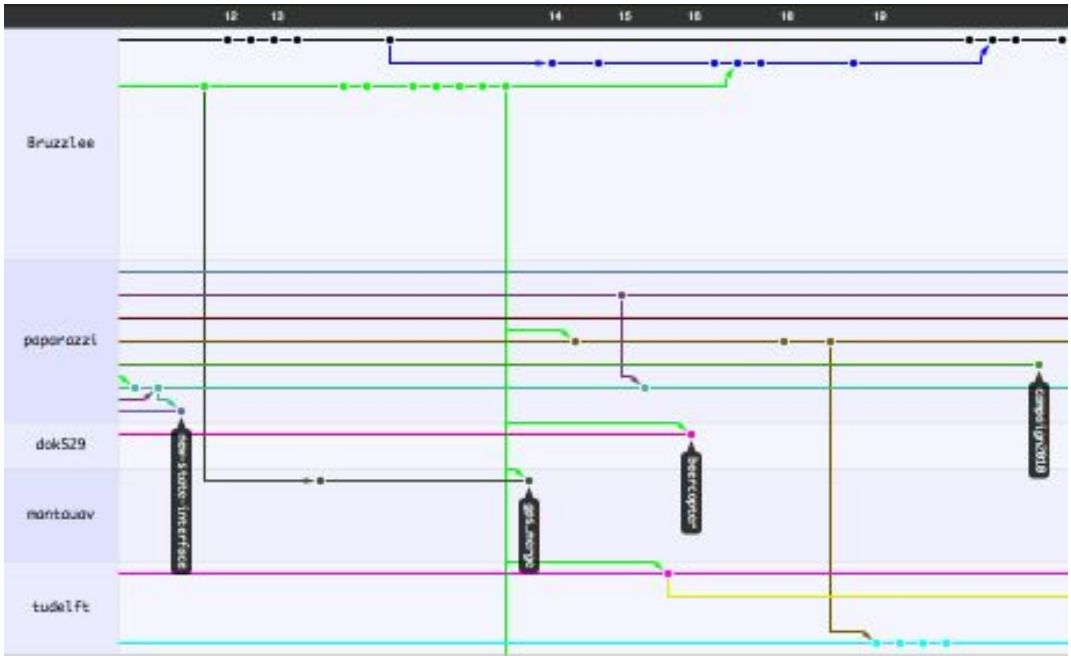


Abbildung 16.2.: Github Network Graph Viewer Ausschnitt

GIT ist eine VCS-Lösung, welche es ermöglicht, dass mehrere Personen am selben Projekt arbeiten können. Es erkennt Veränderungen in Dateien und ermöglicht bei zwei verschiedenen neuen Versionen das Zusammenführen. Der einzige Nachteil an dieser VCS-Lösung ist dessen Komplexität. Das Programm "GIT-Gui"hilft nur bei bestimmten Problemen. Da das Programm sehr umfangreich ist, ist es auch sehr schwierig für Anfänger, die geeigneten Befehle richtig zu interpretieren. Um den Einstieg zu erleichtern, wurde dieses Kapitel als Ergänzung zu der vorhandenen Beschreibung auf dem Paparazzi-Wiki geschrieben.

16.1.1. Wichtigste Vorteile

- Gleichzeitiges Arbeiten an einem Projekt möglich
- Projekt ist jederzeit online für alle verfügbar
- Änderungen sind klar und strukturiert als Zeitstrahl festgehalten
- Falls zwei Personen miteinander eine Datei verändert haben, wird dies gemeldet
- Bei Problemen ist ein Rollback möglich

16.2. Erste Schritte

Hier wird kurz auf die Schritte eingegangen, welche nötig sind, um an einem Projekt bei GitHup mitzuwirken.

16.2.1. Ohne Account

Das einfache Herunterladen des Codes kann bei GIT ohne Account durchgeführt werden. Dazu muss wie folgt vorgegangen werden:

```
cd /to/any/folder
git clone git://github.com/paparazzi/paparazzi.git
```

Mit diesem Befehl wird der gesamte Code heruntergeladen und in das Verzeichnis gespeichert, zu welchem man zuvor mit `cd` gewechselt ist.

16.2.2. Mit Account

Will man den eigenen Code veröffentlichen oder an einem Projekt mitarbeiten, wird folgendermassen vorgegangen:

- Account erstellen auf `http://github.com/`
- Ein SSH-Schlüssel muss generiert werden, welcher dann im GitHub Account eingetragen wird. Eine schrittweise Beschreibung kann auf
  `http://github.com/guides/providing-your-ssh-key` gefunden werden.
- In der Konsole: `git config --global user.name "Dein Name"`
- In der Konsole: `git config --global user.email you@yourdomain.example.com`
- Eine Kopie des Projekts auf GitHub erstellen, an welchem gearbeitet werden soll.
- Das Projekt herunterladen mit:
  `git clone -o <username> git@github.com:<your github username>/paparazzi.git`

16.3. Benutzung

Um den Einstig zu erleichtern, werden hier die wichtigsten Benutzervorgänge beschrieben.

16.3.1. Pull

Beim Pull-Befehl handelt es sich um das Herunterladen der aktuellen Projektversion vom Git-Server. Vom Befehl `git pull` wird abgeraten, da mit diesem Befehl das Programm versucht, das lokale Projekt mit dem Serverprojekt automatisch zusammenzuführen. Dies kann zu unglücklichen Überraschungen führen, da Dateiänderungen verloren gehen können. Um das lokale Projektabbild mit der Serverprojektversion zu aktualisieren, sollte wie folgt vorgegangen werden, wobei `orig` der Remote (`git remote`) zu dem betreffenden Projekt ist.

```
cd /to/your/paparazzi/project
git fetch
git merge orig
```

16.3.2. Push

Um einen neuen Programmcode zu veröffentlichen, muss das Projekt "gepusht"werden. Bevor jedoch ein `git push` ausgeführt wird, sollte die neuste Version vom Server geladen werden, wie es im Abschnitt 16.3.1 beschrieben ist. Danach kann mit folgenden Befehlen im Terminal der Push durchgeführt werden.

```
cd /to/your/paparazzi/project
git push orig
git status
git add --all (oder nur ausgewählte Dateien)
git status
git commit -m "Beschreibung der Änderungen"
git push orig
```

Zwischendurch kann mit `git status` kontrolliert werden, ob alles wie gewünscht ausgeführt wurde.

16.3.3. GIT Gui

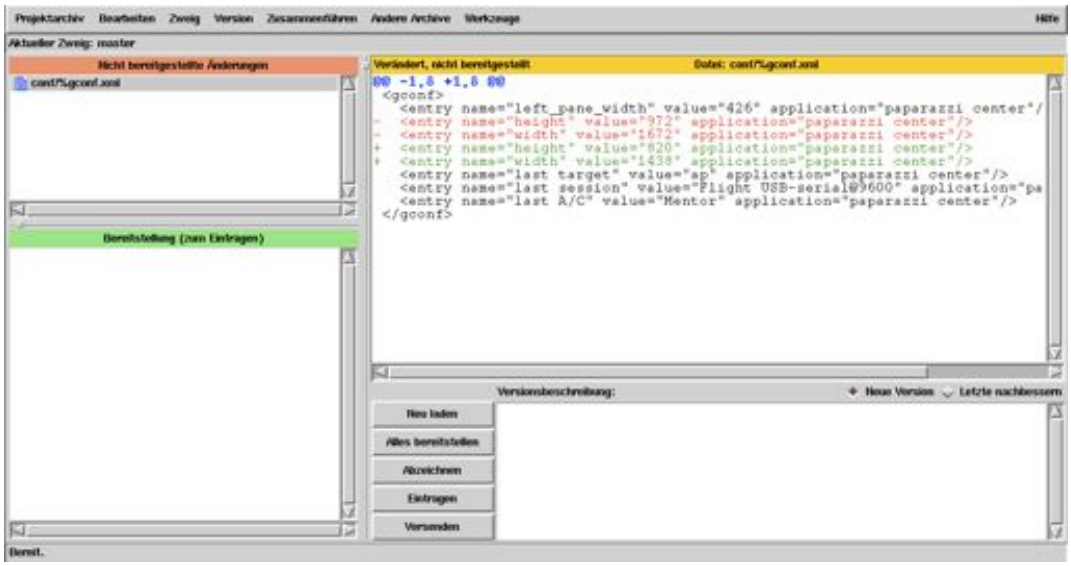


Abbildung 16.3.: GIT Gui Screenshot

Funktionen wie das „Pushen“ oder Zusammenführen von Projekten können über ein Programm namens `GIT GUI` getätigt werden. Es muss beachtet werden, dass ein Pull-Befehl mit diesem Programm ein automatisches Zusammenführen ausführt, was zu den im Abschnitt 16.3.1 beschriebenen Problemen führen kann. Für das „Pushen“ kann jedoch ohne Probleme mit dieser Oberfläche gearbeitet werden. Das Programm wird mittels folgendem Konsolenbefehl ausgeführt.

```
cd /to/your/paparazzi/project
git gui
```

Um veränderte Dateien, welche im Abschnitt `Nicht bereitgestellte Aenderungen` zu einem Commit (Transferauftrag) hinzuzufügen, muss die gewünschte Datei in dieser Liste angewählt

werden und über `Version - Zum Eintragen bereitstellen` oder über die Tastenkombination (`Ctrl-T`) zu der Liste `Bereitstellung` hinzugefügt werden. Falls beim Zusammenführen Konflikte auftreten, können diese im Fenster, in dem der Code angezeigt wird, mit der rechten Maustaste gelöst werden.

Sobald sich alle gewünschten Dateien in der `Bereitstellung`-Liste befinden, benötigt der Commit noch eine Versionsbeschreibung in der gleich benannten Eingabezeile. Danach ist nur noch ein Klick auf `Eintragen` und auf `Versenden` nötig, damit der Code auf den Server geladen wird.

17. XBee

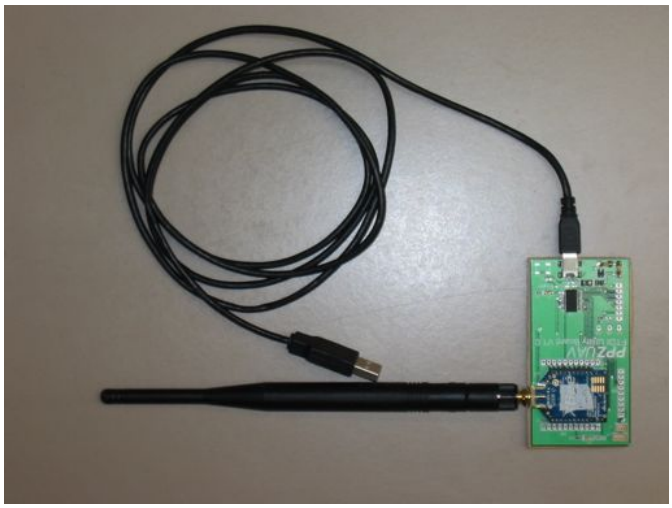


Abbildung 17.1.: XBee Bodenstation

Damit die GCS eine Verbindung zum Flugobjekt aufbauen kann, wird ein Funkmodem namens XBee der Firma Digi verwendet. Die Bauteile besitzen je einen eigenen Controller, welche die Datenverbindung aufbauen und halten. Jeder Controller muss separat konfiguriert werden. In dieser Arbeit wurde die Baudrate und das Pairing konfiguriert, da die negativen Konsequenzen jedoch zu gross waren, wurden die Änderungen wieder rückgängig gemacht. Es sollen nun die nötigen Konfigurationen erklärt werden.

17.1. X-CTU

Das Programm X-CTU dient zur Konfiguration der XBee-Elemente. X-CTU steht nur für Windows zu Verfügung, jedoch kann es problemlos mit `Wine` unter `Linux` betrieben werden. Auf die Installation des Programms wird hier nicht weiter eingegangen, da dies auf der Paparazzi-Seite schrittweise erklärt wird.
Als erster Schritt muss dafür gesorgt werden, dass X-CTU eine Verbindung zur XBee-Hardware aufbauen kann, welche mit dem PPZ-UAV FTDI Utility Board V1.0 (Abbildung 17.1) über USB verbunden ist. Wenn mit Wine gearbeitet wird, dient dafür folgender Konsolenbefehl:

```
sudo ln -s /dev/ttyUSB0 ~/.wine/dosdevices/com4
```

Nun ist das USB Gerät bei Wine an COM4 verfügbar. Dies muss in X-CTU nun, wie auf der Abbildung 17.2 ersichtlich, über das Eingabefeld `COM Port Number` eingegeben und mit `Add` bestätigt werden. Nachdem die richtige Baudrate ausgewählt wurde, kann durch den Knopf `Test / Query` die Verbindung getestet werden.

Abbildung 17.2.: Screenshot von X-CTU Verbindungskonfiguration

Nachdem die Schnittstelle definiert wurde, kann die Firmware geladen werden. Über den Button `Download new Versions` können Firmwareversionen importiert werden. Wie in der Abbildung 17.3 ersichtlich muss die Firmware als zip-Datei geladen werden. Die web-unterstütze Firmware-aktualisierung funktioniert nicht.

Bei diesem Projekt wurden die folgenden zwei Firmwareversionen verwendet:

- ZNET 2.5 COORDINATOR AT v.1041 - für die Bodenantenne
- ZNET 2.5 ROUTER/END DEVICE AT v.1241 - für den Sender im Flugzeug

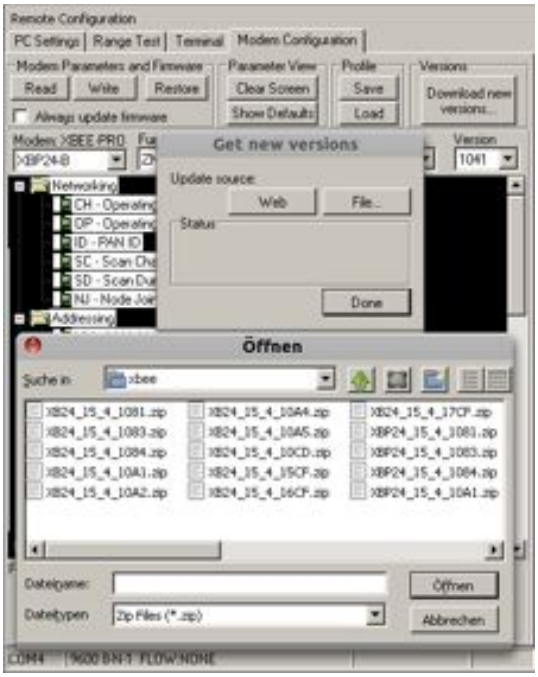


Abbildung 17.3.: Screenshot von X-CTU Laden der Firmware

Wurde die passende Firmware in X-CTU geladen, können die aktuellen Parameter aus der XBee-Hardware geladen werden. Nach dem Betätigen der Schaltfläche `Read` wird die richtige Firmwareversion ausgewählt und mit den Parametern versehen, die zur Zeit auf der Hardware konfiguriert sind. Diese können dann verändert werden und mit dem Knopf `Write` auf die Hardware zurückgeschrieben werden. Danach kann nochmals überprüft werden, ob die Parameter richtig geschrieben wurden, indem die Parameter nochmals ausgelesen werden.

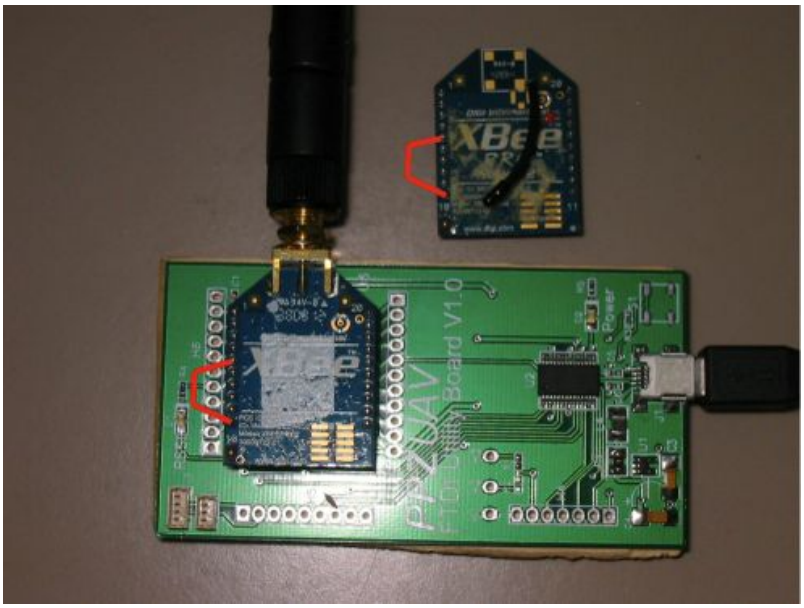


Abbildung 17.4.: Brücke um XBee rückzusetzen

Es kann vorkommen, dass beim Schreiben der Parameter ein Reset verlangt wird. In diesem Fall ist es nötig, den 5. und den 10. Pin (Siehe Abbildung 17.4) miteinander kurz zu verbinden. Danach sollte X-CTU mit dem Schreibvorgang fortfahren.

17.2. Upgrade

Laut Herstellerseite sei es möglich, die Firmwareversionen von einem XBee Pro v2.0 von ZNET auf ZIGBEE zu upgraden. Dies wurde bei diesem Projekt durchgeführt. Jedoch musste beim darauf folgenden Flugversuch festgestellt werden, dass die Funkverbindung während des Flugs oft unterbrochen wurde. Nach einigen Versuchen bei denen auch Standard Einstellungen getestet wurden, wurde die in diesem Projekt verwendeten Funkmodem wieder mit der alten Version (ZNET) beschrieben.

17.2.1. Versionen

- Bodenstation - COORDINATOR AT
  - ZNET 2.5 - Version 1041
  - ZIGBEE - Version 2041
- Funkeinheit im Flugzeug - ROUTER/END DEVICE AT
  - ZNET 2.5 - Version 1241
  - ZIGBEE - Version 2841

17.3. Baudrate

Da es im Verlauf der Projektarbeit immer häufiger nötig war, bestimmte Werte vom Flugzeug an die GCS zu senden, gab es Momente, bei welchen eine grössere Bandbreite nötig gewesen wäre. Um eine grössere Bandbreite zu erhalten, sollte die Baudrate erhöht werden. Sie wurde von 9600 auf 57600 erhöht. Nach dieser Änderung musste jedoch wie bei dem im Abschnitt 17.2 beschriebenen Upgrade mit Verbindungsausfällen gekämpft werden. Danach wurde die Baudrate wieder auf 9600 zurückgesetzt und das Programm so optimiert, dass möglichst wenig Daten versendet wurden. Nach diesem Versuch stand fest, dass eine hohe Baudrate zwar einen hohen Datendurchsatz erlaubt, jedoch die Stabilität der Datenverbindung enorm verringert. Das Testen mit einer Baudrate von 19200 wäre noch ein offener Punkt, welcher aus Zeitgründen jedoch nicht durchgeführt werden konnte. Ebenfalls könnte eine Änderung der aktuellen Kompaktbauweise des Autopiloten einen wesentlichen Beitrag zur Verbindungstabilität beitragen. Es sollte eine

externe Antenne angebracht werden, um die Verbindung nicht durch die restliche Elektronik des Autopiloten zu stören.

17.4. Pairing

Um mehrere Flugzeuge mit Autopilot gleichzeitig an einem Ort steuern zu können, muss gewährleistet werden, dass die Funksignale an die richtige GCS geschickt werden. Dazu kann das Pairing verwendet werden. Beim Pairing muss darauf geachtet werden, dass dem XBee Coordinator und dem End device dieselbe Pairing ID zugewiesen wird. Diese Pairing ID kann mittels X-CTU gesetzt werden. Das Pairing wurde im Verlauf dieses Projekt kurz eingeschaltet und wegen Verbindungsproblemen auf Grund von neuen Versionen wieder verworfen. Eine konstante Verbindung konnte nur ohne Pairing erzieht werden. Es konnte nicht 100-prozentig bestätigt werden, dass die Verbindungsausfälle auf Grund des Pairing zustande kamen. Es wäre auch möglich, dass Störungen durch andere Funksender eines Modellfliegers, welche ebenfalls mit einer Frequenz von 2.4 GHz senden, hervorgerufen werden. Bei Testversuchen in der Stadt Winterthur konnten keine Probleme festgestellt werden. Erst als die Drohne flog, ging die Verbindung verloren. Hier könnte möglicherweise ebenfalls eine externe Antenne am Flugzeug das Problem beheben.

18. Schlussfolgerung

Die wichtigsten Erkenntnisse dieser Arbeit werden in diesem Kapitel zusammengefasst.

18.1. Fazit

In der Zeit der Bachelorarbeit wurde viel Wissen über die Aviatik gesammelt, was für die Durchführung der autonomen Landung beziehungsweise des autonomen Startvorgangs nötig war.

Für die Weiterführung des Projekts ist es sinnvoll, wenn mindestens ein Teammitglied Modellflugerfahrung mitbringt. So ist es möglich, dass das Team die Testflüge im Verlauf der Arbeit selbständig durchführen kann, um hohe Personalkosten einzusparen, da dies der grösste Kostenpunkt dieser Bachelorarbeit war.

Bei den Parameterfindungen ist es notwendig ein Protokoll zu führen, da es möglich ist, dass die Speicherung der gefundenen Parameter vergessen geht und die Nachvollziehbarkeit so gewährleistet wird. Im Anhang B.2 wurde das in dieser Arbeit verwendete Protokoll eingefügt, in welchem die Parameter sowie die Fehlerquadratsummen festgehalten werden können.

18.1.1. Airspeed

In dieser Arbeit wurde viel Zeit in die Regelung des `True Airspeed` investiert, da für eine sichere Landung ein stabiler, windunabhängiger Flug nötig ist. Verschiedene Regelsysteme wurden entwickelt um schlussendlich die Besten unter ihnen auszuwählen.
Um die Regelparameter des Anstellwinkels (Pitch) zu finden, wurde während des Flugs zuerst der Proportionalfaktor so gewählt, dass keine der Regelgrössen schwingt, aber die Regelung dennoch schnell auf eine Sollwertänderung reagieren kann. Danach wurde der Integralanteil des Reglers erhöht, bis die bleibende Regelabweichung des Proportonalreglers aufgehoben werden konnte. Ein Differentialanteil war selten bis nie brauchbar, da dieser das System schon bei kleinen Werten destabilisierte.
Bei der Findung der Regelparameter für die Motorenleistung wurden gute Resultate mit erhöhten Integralfaktoren und reduzierten Proportionalfaktoren erzielt. Dies hängt damit zusammen,

dass die Energie des Motors nur sehr langsam in die Geschwindigkeit übergeht. Schlagartige Änderungen der Motorleistungen, herbeigeführt durch einen zu hohen Proportionalanteil, destabilisieren das System unnötig.
Zu Anfang dieser Arbeit war das Simulieren des Flugzeugs für die Parameterfindung noch einer der möglichen Punkte. Jedoch rückte diese Idee nach einer Besprechung in den Hintergrund, da herauskam, dass der Simulator, welcher in Matlab programmiert wurde, noch fehlerbehaftet war, die physikalischen Eigenschaften der Testdrohne nicht vorlagen und das Simulationsprogramm, welches die Aviatiker benutzen, nicht kompatibel mit der Autopiloten-Software ist. Ein anderes Team befasste sich mit der Findung der physikalischen Eigenschaften der UMARS. Mit diesen Werten sollte es möglich sein, ein Simulationsmodell zu erstellen, welche die Parameterfindung der UMARS simulieren und beschleunigen kann. Jedoch können diese Parameterwerte nur für die UMARS und nicht für die Testdrohne verwendet werden, da dessen physikalischen Eigenschaften dann stets unbekannt sind.

18.1.2. Start

Schnell musste festgestellt werden, dass beim Start mit einem Gummiseil (Bungee) die `ArduIMU` wegen den hohen Beschleunigungen noch bis ca. 5 bis 10 Sekunden nach dem Start falsche Pitchwerte liefert. Aus diesem Grund wurde ein `Angle of Attack` Sensor gebaut. Weitere Schritte zu diesem Thema sind in den Abschnitten 18.2.2 und 18.2.1 zu finden.

18.1.3. Landung

Für die Landung sind die Parameter der `Fixed Pitch Regelung` so zu wählen, dass die Höhenregelung einen minimalen Fehler nach unten aufweist. Dies wird erreicht, indem für den `Prethrottle`-Wert nur ein minimal unterhalb der benötigten Motorenleistung liegender Wert gewählt wird. Mittels hohem Integralfaktor und niedrigem Proportionalfaktor ist es dann möglich die Höhe bei konstanten Windstärken sehr genau auf dem Sollwert zu halten. Weitere Erkenntnisse der autonomen Landung sind unter Kapitel 12 zu finden.

18.1.4. Anforderungsliste

Beim erstellen Anforderungsliste für die UMARS musste festgestellt werden, dass es schwierig ist, die verschiedenen Anforderungen der beteiligten Personen in eine allgemein gültige Vorgabe mit Zahlenwerten zusammenzufassen. Die für dieses Projekt wichtigen Punkte, wie die Landung und der Startvorgang, wurden jedoch von Beginn an genau definiert, da sie, nicht wie die anderen Werte, ebenfalls für die Testdrohne `Mentor` gültig sind.

18.2. Weitere Schritte

In diesem Abschnitt werden die Schritte zusammengefasst, welche in weiterführenden Arbeiten behandelt werden können und sollten. Jegliche Massnahmen beziehen sich auf ein Problem, welches im Verlauf dieser Bachelorarbeit auftrat. Details zum jeweiligen Thema können im dazugehörigen Abschnitt nachgeschlagen werden.

18.2.1. ArduIMU

Wie im Abschnitt 15.4.6 erklärt wird, resultiert aus der hohen Beschleunigung des Bungeestarts ein verfälschter Theta-Winkel, welcher zu einem Absturz der Drohne führen kann. Aus diesem Grund ist es notwendig, dieses Problem zu beheben. Falls der Angle of Attack Sensor für diese Aufgabe unbefriedigende Resultate liefert, kann noch eine Softwarelösung in Betracht gezogen werden. Die Ursache des Messfehlers liegt an den Beschleunigungssensoren, welche die achsiale Beschleunigung in x-Richtung messen. Diese Beschleunigungssensoren könnten kurzzeitig deaktiviert werden, damit der Winkel nur noch mit den Gyros errechnet wird. Da die Gyros jedoch einen Drift aufweisen, sollten die Beschleunigungssensoren nur für kurze Zeit nicht in die Lageberechnung miteinbezogen werden. Ein Lösungsansatz wird momentan von der Paparazzi-Community entwickelt. Dieser Code ist jedoch noch in der Testphase.

18.2.2. Angle of Attack

Die Regelung des Angle of Attack hat bei dem durchgeführten Testflug keine befriedigenden Resultate geliefert. Zwar hat die Lageregelung im `Auto 1` sehr gut funktioniert, aber eine komplette Regelung im `Auto 2` war nicht möglich. Hier sollte eine gemischte Regelung aus Angle of Attack und Pitchwinkel entwickelt werden, bei welcher der AOA die eigentliche Regelgrösse ist. Der Pitchwinkel sollte hier zur Überwachung der Regelung zugeschaltet werden, um zu

verhindern, dass die Drohne zu stark steigt oder sinkt. Es ist natürlich wichtig, dass hier die absolute Lageregelung gegenüber der Erde die Priorität behält.

18.2.3. Airspeed

In dieser Bachelorarbeit wurde viel Zeit investiert, ein geeignetes Regelsystem für den Airspeed zu finden. Schlussendlich wurden die Regelungen gewählt, welche am besten zu parametrieren waren. Mit der `Airspeed Manual Power`-Regelung ist es möglich, ein entkoppeltes Regelsystem zu parametrieren. Bei dieser Regelung wird die Pitchregelung separat parametriert. Die gefunden Parameter können dann für die `Airspeed Pitch Simple` übernommen werden. Mit dieser Methode konnten brauchbare Parameterwerte gefunden werden.
Ein weiterer Schritt besteht darin, das System weiter zu entkoppeln oder eine bessere Methode für die Parameterfindung zu entwickeln. Ein geeignetes Simulationsmodell könnte hier eventuell in Betracht gezogen werden.

18.2.4. Start

Beim Start traten die grössten Schwierigkeiten während dieser Bachelorarbeit auf. Das Problem mit der Fehlfunktion der `IMU` durch die starke Beschleunigung beim Start konnte nicht restlos behoben werden. Als erster Ansatz wurde eine Angel of Attack Messvorrichtung gebaut, mit welcher zumindest die Lage des Flugzeuges relativ zur Luftströmung bestimmt werden kann. Diese konnte aber wegen schlechten Wetters und Zeitmangels nur noch in der Luft getestet werden, nicht aber beim Start mittels Gummiseil. In einem weiteren Schritt zur Verbesserung des autonomen Starts sollte genauer auf die `IMU` eingegangen werden, da die Probleme, die sich während dieser Arbeit beim Starten gezeigt haben auf diese zurückzuführen sind. Möglicherweise kann eine Softwarelösung gefunden werden, bei der für den Start nur die Gyros verwendet werden und die Beschleunigungssensoren abgeschaltet werden. Eine weitere Möglichkeit wäre es, andere Inertial Measurement Units zu testen. Ausserdem könnte zur Erhöhung der Sicherheit des autonomen Starts ein Sensor am Bungeehaken der Drohne befestigt werden, welcher detektiert, wann sich das Seil von diesem gelöst hat. Dadurch würde das Risiko, dass der Motor startet, bevor das Seil gelöst ist, minimiert und so verursachte Abstürze würden verhindert.

18.2.5. Landung

Da die Landung sehr gut funktioniert hat, sind hier keine grossen Änderungen am Code oder an der Hardware mehr nötig. Die Landung könnte jedoch durch gezielte Fehlersicherungen in den einzelnen Schritten sicherer gemacht werden. Ausserdem sollten die bereits vorhandenen Fehlersicherungen durch gezielte Tests überprüft und gegebenenfalls verbessert werden. Ausserdem sollten noch einige Tests bei unterschiedlichen Wetterbedingungen durchgeführt werden. Wenn die Landung bei allen Wetterlagen funktioniert, ist der Algorithmus bereit zur Portierung auf die UMARS.

19. Glossar

Begriff	**Erklärung**
Airframe	Das Airframe ist eine XML-Datei, in welchem Parameter definiert und Module zugeschaltet werden. Ebenso sind dort spezifische Einstellungen der Hardware definierbar.
AOA	Angle of Attack; ist der Winkel, in dem die Luft das Flugzeug anströmt. Zur Messung ist eine weitere Messeinrichtung nötig,welche in Kapitel 7.2.5 genauer beschrieben ist.
Array	Ein Datentyp in Form einer Tabelle welche beim Programmieren verwendet wird.
Auto 1 Mode	Befindet sich das Flugzeug im Auto 1 Mode, können mit der Fernsteuerung die Sollwerte von Pitch und Roll definiert werden. Der Autopilot regelt in diesem Modus mit diesen Sollwerten.
Auto 2 Mode	Befindet sich das Flugzeug im Auto 2 Mode, fliegt es völlig autonom. Die Sollwerte werden durch den vordefinierten Flugplan errechnet.
boolische Variable	Eine Art Variable, welche nur zwei Zustände annehmen kann. (True / False)
`BungeeHook`	Der BungeeHook ist der Wegpunkt, welcher in der Paparazzi-Startroutine für die Definition der Position des Bungeehakens benötigt wird.
Codedatei	Teil eines Programmcodes in dem das eigentliche Programm geschrieben wird. Dateiendung: .c
crimpen	Crimpen ist ein Fügeverfahren, welches durch plastisches Verformen der zusammengefügten Teile Kontakt herstellt.
`DirectionWP`	Der DirectionWP ist ein Wegpunkt im GCS, mit welchem die Startrichtung definiert wird.
Direkte Regelung	In dieser Arbeit wird als direkte Regelung eine Regelung bezeichnet, welche die Geschwindigkeit der Drohne über den Motor und die Höhe über das Höhenruder regelt (Gegenteil von indirekter Regelung).

estimator	Der Estimator ist ein Teil des Paparazzicodes. Von diesem Teil werden Variablen zu Verfügung gestellt, welche für die eindeutige Positionierung des Flugzeugs benötigt werden. Zum Beispiele: `estimator_z` oder `estimator_theta`.
Flare / flaren	Unter Flare versteht man den Zustand eines Flugzeuges, bei dem sich die Nase hebt. Durch diese Erhöhung des Pitchwinkels reduziert sich die Sinkrate.
Flashen	Mit Flashen wird das Beschreiben des Paparazzi-Speichers gemeint.
GCS	Ground Control Station; Bodenstation mit welcher dem Autopilot Befehle vom Boden gesendet werden können.
Geschwindigkeit	Mit Geschwindigkeit ist immer die Geschwindigkeit der Drohne gegenüber der sie umgebenden Luft gemeint (Synonym: True Airspeed).
GPS	Global Positioning System; System, mit welchem die Position eines GPS-Empfängers auf wenige Meter bestimmt werden kann.
Ground Speed	Ground Speed bezeichnet die Geschwindigkeit der Drohne gegenüber dem Boden
Headerdatei	Teil eines Programmcodes in dem unter anderem Methoden und öffentliche Variablen definiert werden. Dateiendung: .h
I2C	I2C ist ein serieller Datenbus, welcher aus zwei Kanälen besteht. SDA und SCL. Über diese Schnittstelle wurden die Drucksensoren sowie die IMU und der Magnetometer mit dem Tiny Board verbunden.
IMU	Die IMU (Inertial measurement unit) ist in diesem Fall ein Chip, welcher mit Hilfe von drei Beschleunigungs einem Gyroskop und den GPS Daten die Lage des Flugzeugs errechnet.
Indirekte Regelung	In dieser Arbeit wird als indirekte Regelung eine Regelung bezeichnet, welche die Höhe der Drohne über den Motor und die Geschwindigkeit über das Höhenruder regelt (Gegenteil von direkter Regelung).
`Inner-Loop`	Der Inner-Loop beinhaltet alle Regelkreise, welche schlussendlich die Aktoren ansteuern
kompilieren	Übersetzen und Vernetzen des Quellcodes in Maschinencode

`Launch Line`	Die Launch Line ist die Linie, welcher die Drohne beim Start folgt.
Linux	Meist frei erhältliches Betriebssystem basierend auf UNIX
loiter	Im Falle dieser Arbeit: gemütliches Fliegen (schlendern)
Magnetometer	Der Magnetometer ist ein 3D-Kompass. Er wurde als Chip in den Autopiloten eingebaut und über die I2C-Schnittstelle verbunden.
`max_pitch`	`max_pitch` bezeichnet den maximalen Pitch, welcher für die Drohne zulässig ist. Es dient als eine Begrenzung des angegebenen Winkels
`max_roll`	`max_roll` bezeichnet den maximalen Roll-Winkel welcher für die Drohne zulässig ist. Er dient als eine Begrenzung des angegebenen Winkels.
Pairing	Funkmodem: Paaren zweier oder mehreren XBees mittels PAN-ID. Ziel: Zwei verschiedene Funkverbindungen mit unterschiedlichen PAN-IDs nebeneinander gleichzeitig betreiben zu können.
Paparazzi-Projekt	Das Paparazzi-Projekt ist ein Open-Source Hardware und Software Projekt für die unbemannte Luftfahrzeuge.
parametrieren	Definition der Parameter (z.B. Proportionalfaktoren)
Parametrierflug	Ein Parametrierflug ist ein Testflug mit dem Ziel die Regelparameter für eine Regelung zu bestimmen.
PicoBlade	Steckverbindungstyp mit welchen die Litzen mit dem Tiny Board verbunden wurden.
Pitch	Winkel der Längsachse einer Flugzeuges gegenüber der Horizontalen. Dieser Winkel wird mit dem Höhenruder verändert.
Pitch-Loop	Dem Pitch-Loop wird ein Sollwinkel vorgegeben, welcher die horizontale Neigung des Flugzeugs definiert. Dieser Loop regelt diese Neigung zum Boden oder mit Angle of Attack Sensor zum Anströmwinkel.
Prandtlsonde	Eine Art Röhrchen, welches an der Spitze den Staudruck und an den Seiten den Umgebungsdruck abnimmt. Aus der Differenz dieser Drücke kann die Strömungsgeschwindigkeit des umströmenden Fluids berechnet werden.
Roll	Winkel der Querachse gegenüber der Horizontalen

Roll	Mit dem Rollwinkel wird die vertikale Lage des Flugzeugs beschrieben. Dieser Winkel wird mit den Seitenruder verändert.
Settings	In der Paparazzi-Software beschreiben Settings die Einstellungen, welche während des Flugs verändert werden können. Die Settings-XML-Dateien beinhalten alle Parameter, welche dann im GCS als Settings aufgelistet werden.
Throttle Line	Die Throttle Line ist eine unsichtbare Linie rechtwinklig zur Lauch Line. Sie definiert, ab welcher Position die Drohne beim Start den Motor starten darf.
`Throttle_Slew_Limiter`	Der `Throttle_Slew_Limiter` definiert eine Zeit, welcher der Motor mindestens in Anspruch nehmen soll, um eine grosse Drehzahländerung zu vollziehen. Wichtig bei Verbrennungsmotoren.
Tiny Board	Hauptplatine des Autopiloten. Auf dieser Platine ist der Speicher sowie der Prozessor und alle nötigen Schnittstellen des Autopiloten untergebracht.
Trajektorie	Flugpan, Raumkurve
True Airspeed	True Airspeed bezeichnet die Geschwindigkeit der Drohne gegenüber der sie umgebenden Luft (Synonym: Geschwindigkeit).
True Airspeed	Siehe Geschwindigkeit
Wegpunkt	Ein Wegpunkt ist ein Punkt im dreidimensionalen Raum, der benötigt wird, um der Drohne über das GCS zu steuern. Ein Wegpunkt hat drei Attribute: x-, y-Koordinate und Höhe über dem Meer.
Wine	Ein Programm unter Linux, welches ermöglicht, Programme für Windows zu verwenden.
XBee	Name des verwendeten Funkmodems
Yaw	Yaw beschreibt die Drehung um die Z-Achse des Flugzeugs. Dieser Winkel wird mit dem Querruder beeinflusst.
Zip-Datei	Ein komprimierender Container von Dateien und Ordner.

Tabelle 19.1.: Glossartabelle

A. Abbildungsverzeichnis

Flugplan Tag __

Status: *0 =nicht durchgeführt*
1 = abgeschlossen
2 = Nachbearbeitung nötig

Flug Nr.	Funktions -Nr	Zu testende Funktion / einzustellende Parameter	Status	Bemerkung
	1			
	2			
	3			
	4			
	5			
	6			

Flugprotokoll

Datum:_____ Flugtag-Nr. _____ Bearbeitete Regelung: _____

Startzeit: _____ Landezeit: _____ Flugdauer: _____

Notizen:

gndsp_setpoint:_____ gndsp_pgain:_____ gndsp_igain:_____

Nummer	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
as_setpoint										
as_deadband										
prethrottle										
pitch_pgain										
pitch_igain										
J-Speed										

Nummer	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
throttle_pgain										
throttle_igain										
J-High										

Nummer	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
course_pgain										
course_igain										
J-Course										

J = Fehlerquadratsumme

Flugplan Tag 1 Umurs

Status: 0 = nicht durchgeführt
1 = abgeschlossen
2 = Nachbearbeitung nötig

Flug Nr.	Funktions -Nr	Zu testende Funktion / einzustellende Parameter	Status	Bemerkung
1	1	Auto 1 kommen lassen/landen lassen	1	Es ist wichtig, dass die Umurs immer in der gleichen Position initialisiert sind (wegen IMU)
	2	Auto 2 kommt	2	
2+3	3	Auto 2 kommen	2	noch nicht alle Parameter aufeinander abgestimmt
	4			
	5			
	6			

Flugplan Tag 1

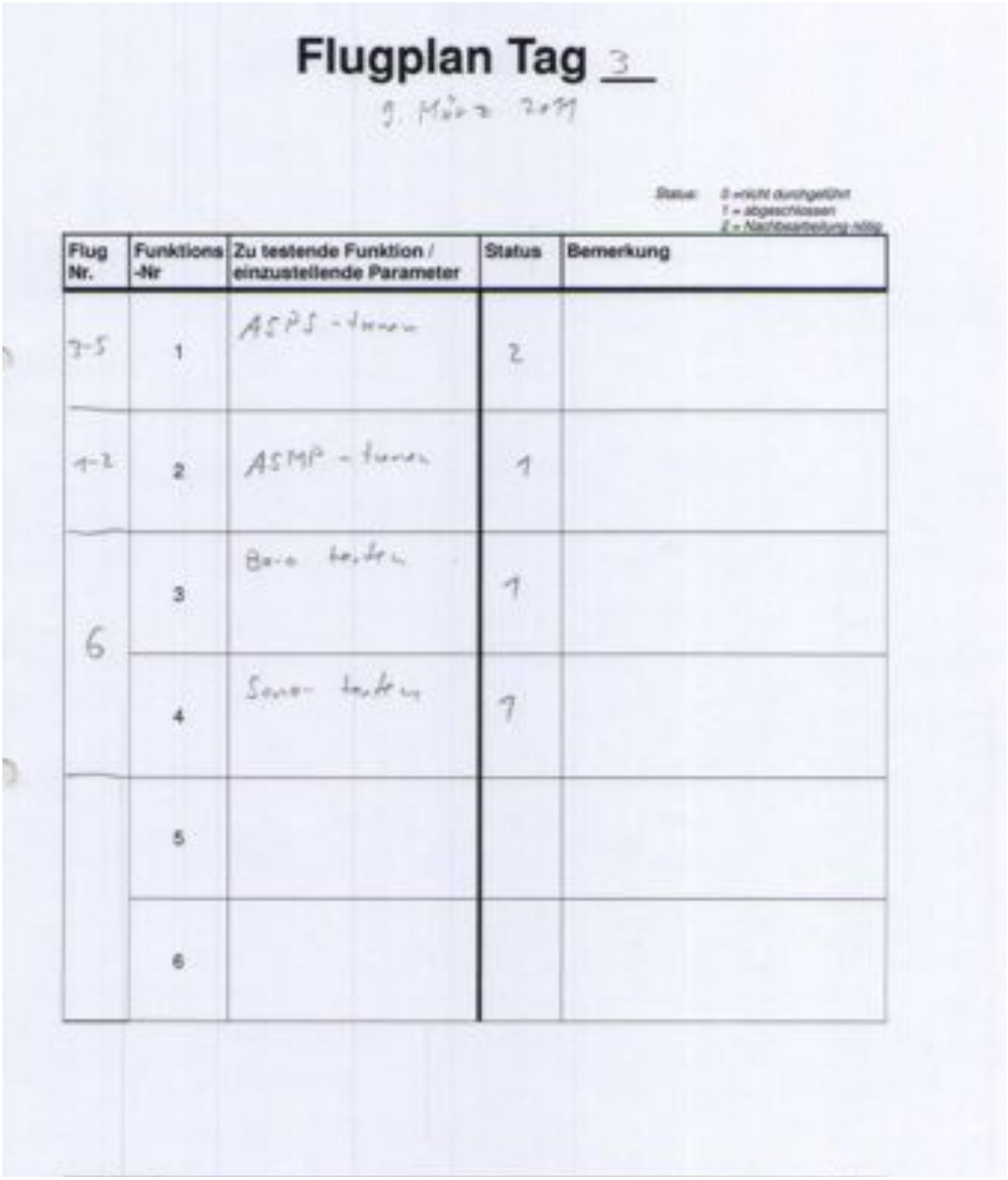
Status: 0 = nicht durchgeführt
1 = abgeschlossen
2 = Nachbearbeitung nötig

Flug Nr.	Funktions -Nr	Zu testende Funktion / einzustellende Parameter	Status	Bemerkung
1-2	1	I2C - Sensoren testen	1	Kalibrierung der Sensoren nötig
	2	Ventile - Parameter einstellen	2	Nur noch geringfügige Änderungen nötig
3-4	3	Autopilot Two Parameter einstellen	2	Nur noch geringfügige Änderungen nötig
	4	Bodenstation testen	0	
	5			
	6			

Flugplan Tag 2

Status: 0 = nicht durchgeführt
1 = abgeschlossen
2 = Nachbearbeitung nötig

Flug Nr.	Funktions -Nr	Zu testende Funktion / einzustellende Parameter	Status	Bemerkung
	1	Auto 1 tunen	1	
	2	Vario nachtunen	1	
	3	Airspeed 2 nachtunen	2	kleine Nachtunearbeiten nötig
	4	Gummiseilstart testen	1	Gefahr durch einhängen des Seils am hinteren Rad vermeiden
	5			
	6			



Flugplan Tag 3

9. März 2011

Status: 0 = nicht durchgeführt
1 = abgeschlossen
2 = Nachbearbeitung nötig

Flug Nr.	Funktions -Nr	Zu testende Funktion / einzustellende Parameter	Status	Bemerkung
3-5	1	ASPS - [illegible]	2	
1-2	2	ASMP - [illegible]	1	
6	3	Baro testen	1	
	4	Sonar testen	1	
	5			
	6			

Flugplan Tag 4

Status: 0 = nicht durchgeführt
1 = abgeschlossen
2 = Nachbearbeitung nötig

Flug Nr.	Funktions -Nr	Zu testende Funktion / einzustellende Parameter	Status	Bemerkung
4-5	1	ASPC - funct.	2	
3	2	ASPI - funct.	2	
1-2	3	ASMP - funct.	2	
	4	Landung testen	1	Landung erfolgt programmiert (Code überprüfen)
	5			
	6			

Flugplan Tag 5

7. April 2011

Status: 0 = nicht durchgeführt
1 = abgeschlossen
2 = Nachbearbeitung nötig

Flug Nr.	Funktions -Nr	Zu testende Funktion / einzustellende Parameter	Status	Bemerkung
1-7 + Lehnen 3	1	Boden-start Landen mit OSAMNer-Code	1	Probleme mit Throttle-Line; 4. Motor kommt zu spät; wenn der Motor kommt [illegible] [illegible] beim Start
8-10	2	ASPA - tunen	2	[illegible] nicht auf Änderungen des Soll-[illegible]
11-15	3	ASPS - tunen	1	[illegible] geflogen
Lehnen 3	4	[illegible] Landung mit OSAMNer-Code landen	1	1. Landung geglückt aber neben der Platte. ⊖ [illegible] ⊖ [illegible] zu Beginn; [illegible] ⇩ [illegible]
	5			
	6			

Flugplan Tag 6

20 April 2011

Status: 0 = nicht durchgeführt
1 = abgeschlossen
2 = Nachbearbeitung nötig

Flug Nr.	Funktions -Nr	Zu testende Funktion / einzustellende Parameter	Status	Bemerkung
	1	Auto T Parametrisieren		
	2	Bringer check laden		
	3	Bringer check mit glide laden		
	4	ASMP Parametrieren		
	5	ASPS Parametrieren		
	6			

Flugprotokoll

manual Pilot

Datum: 20. 4. 2011 Flugtag-Nr. 6 Bearbeitete Regelung: ASMP

Startzeit: 2011 Landezeit: _____ Flugdauer: _____

Notizen:

Letzte 2 Flüge Leos Laptop

gndsp_setpoint: 0 gndsp_pgain: 0 gndsp_igain: 0

Nummer	1	2	3	4	5	6	7	8	9	10
as_setpoint	12,3	11,1								
as_deadband	0	0								
~~pit~~throttle setpoint	0,5	0,5								
pitch_pgain	-0,02	-0,04								
pitch_igain	-0,02	0,01								
J-Speed										

Nummer	1	2	3	4	5	6	7	8	9	10
throttle_pgain										
throttle_igain										
J-High										

Nummer	1	2	3	4	5	6	7	8	9	10
course_pgain										
course_igain										
J-Course										

J = Fehlerquadratsumme

Flugprotokoll

Datum: 4. Mai Flugtag-Nr. 1 Bearbeitete Regelung: Variic

Startzeit: _____ Landezeit: 17:24 Flugdauer: _____

Notizen:

gndsp_setpoint: 0 gndsp_pgain: 0 gndsp_igain: 0,25

Nummer	1	2	3	4	5	6	7	8	9	10
as_setpoint	11									
as_deadband	—									
prethrottle	—									
pitch_pgain	-0,035									
pitch_igain	0									
J-Speed	70									

Nummer	1	2	3	4	5	6	7	8	9	10
throttle_pgain	0,015									
throttle_igain	0,149									
J-High	8009									

Nummer	1	2	3	4	5	6	7	8	9	10
course_pgain										
course_igain										
J-Course										

J = Fehlerquadratsumme

Flugprotokoll

Datum: 3 Mai Flugtag-Nr. 2 Bearbeitete Regelung: ASPS

Startzeit: 17:30 Landezeit: _____ Flugdauer: _____

Notizen:

gndsp_setpoint: 0 gndsp_pgain: 0 gndsp_igain: 0,25

Nummer	1	2	3	4	5	6	7	8	9	10
as_setpoint	12	12	13	13	13					
as_deadband	0	0	0	0	0					
prethrottle	0,5	0,5	0,5	0,5	0,5					
pitch_pgain	-0,01	-0,1	-0,1	-0,1	-0,02					
pitch_igain	0,03	0,03	0,03	0,03	0,003					
J-Speed	204	100	70	50	60					

Nummer	1	2	3	4	5	6	7	8	9	10
throttle_pgain	-0,01		-0,01	-0,02	-0,01					
throttle_igain	0,024		0,024	0,024	0,024					
J-High	2500		5000	7000	5000					

Nummer	1	2	3	4	5	6	7	8	9	10
course_pgain										
course_igain										
J-Course										

J = Fehlerquadratsumme

Flugprotokoll

Datum: 6.5.11 Flugtag-Nr. _____ Bearbeitete Regelung: ASMP

Startzeit: 16:15 Landezeit: 17:01 Flugdauer: _____

Notizen:

gndsp_setpoint: 0 gndsp_pgain: 0 gndsp_igain: 0

Nummer	1	2	3	4	5	6	7	8	9	10
as_setpoint	11									
as_deadband	0									
prethrottle	950									
pitch_pgain	-0,1									
pitch_igain	10,05									
J-Speed										

Nummer	1	2	3	4	5	6	7	8	9	10
throttle_pgain										
throttle_igain										
J-High										

Nummer	1	2	3	4	5	6	7	8	9	10
course_pgain										
course_igain										
J-Course										

J = Fehlerquadratsumme

Flugprotokoll

Datum: 6.5.11 Flugtag-Nr. _____ Bearbeitete Regelung: AS FIX PITCH

Startzeit: 17.14 Landezeit: _____ Flugdauer: _____

Notizen:

gndsp_setpoint: 0 gndsp_pgain: 6 gndsp_igain: 6

Nummer	1	2	3	4	5	6	7	8	9	10
~~as_setpoint~~	11									
as_deadband	0									
prethrottle	6									
pitch_pgain	-6048									
pitch_igain	0,024									
J-Speed										

Nummer	1	2	3	4	5	6	7	8	9	10
throttle_pgain										
throttle_igain										
J-High										

Nummer	1	2	3	4	5	6	7	8	9	10
course_pgain										
course_igain										
J-Course										

J = Fehlerquadratsumme

Flugtag 6: Pendenzenliste

Geplanter Ablauf des 6. Flugtages am 20. April 2011

1 IMU Initialisierung (vorgängig erledigt)

Es wurde der neuste IMU-Code auf die IMU geladen und dieser soll nun getestet werden. Bei der Initialisierung wir der künstliche Horizont so eingestellt, dass die Drohne, so wie sie bei der Initialisierung steht gerade ist. Dazu ist es wichtig, dass die Drohne bei der besagten Initialisierung so positioniert wird, wie sie beim geradeaus fliegen steht.

2 Auto 1 Parametrieren (45 Minuten)

Im Auto 1 Modus kann die Drohne zwar von Hand gesteuert werden, aber die Servos der Drohne werden nicht direkt angesteuert, sonder es werden lediglich Sollwerte an den Autopiloten gegeben. Der Autopilot steuert dann die Servos an. Im Auto 1 können die Regelparameter des Outer-Loops eingestellt werden, welche später für die Regelgeschwindigkeit im Auto 2-Modus wichtig sind.

2.1 Ziel

Die Drohne soll im Auto 1 stabil und flink geflogen werden können.

3 Bungee Start testen (30 Minuten)

Der neu programmierte Bungee-Start soll getestet werden und überprüft werden, ob die Probleme mit dem Geradeausflug und dem verzögerten Gasgeben nach wie vor bestehen.

3.1 Ziel

Der Bungee-Start soll ohne Eingreifen der manuellen Steuerung durchgeführt werden können. Falls dies nicht der Fall sein sollte, sollen Probleme oder mögliche Fehler genau dokumentiert werden.

4 Bungee Start mit Glide testen (30 Minuten)

Der Bungee Start mit Glide unterscheidet sich in der Endphase, in welcher die Drohne auf die Reisehöhe erreichen soll. Beim Bungeestart mit Glide wird unmittelbar nach dem Throttle Punkt ein Kurs vorgegeben, welcher einem Gleitpfad mit positiver Steigung entspricht. Um diesem Pfad zu folgen ist eine entsprechende Motorenleistung notwendig.

4.1 Ziel

Der Bungee-Start soll ohne Eingreifen der manuellen Steuerung durchgeführt werden können. Falls dies nicht der Fall sein sollte, sollen Probleme oder mögliche Fehler genau dokumentiert werden.

5 Vergleich der Startalgorithmen (15 Minuten)

Die beiden neuen Startalgorithmen sollen untereinander verglichen werden und die individuellen Vor- und Nachteile festgehalten werden, so dass am Schluss bestimmt werden kann, welcher Ablauf sich besser für einen sicheren Start eignet.

5.1 Ziel

Ziel ist es nach dem Test beider Startalgorithmen denjenigen zu bestimmen, welcher sicheren und besser wiederholbar ist.

6 Airspeed Manual Power Parametrieren (45 Minuten)

„Airspeed Manual Power“ ist ein regelung des Airspeeds, bei dem die Motorenleistung manuell bestimmt werden kann (über GCS). Die in dieser Regelung implementierte Airspeed-Regelung ist die gleiche, wie diejenige der Airspeed Pitch Simple Regelung. Daher können die hier gefundenen Parameter auf die Pitch Simple Regelung übertragen werden. Die Höhe wird hier nicht geregelt.

6.1 Ziel

Die Regelparameter sollen so weit verfeinert werden, dass ein stabiles, den Anforderungen entsprechendes, Flugverhalten einstellt.

7 Airspeed Manual Pitch Simple (45 Minuten)

Wie schon gesagt, ist die Airspeed-Regelung hier gleich aufgebaut, wie die bei der Manuel Power Regelung. Hier wird die Höhe der Drohne jedoch zusätzlich geregelt. Nun gilt es die Parameter für die Höhenregelung zu finden.

7.1 Ziel

Ziel ist es die Feinabstimmung der gesamten Regelung zu optimieren, dass die Anforderungen an die Regelung während der Messphase erfüllt werden.

8 (Programmierter Lowpass)

Als Option, wenn noch genügend Zeit übrig ist und der benötigte Algorithmus fertig gestellt

ist, ist die Durchführung eines Programmierten Lowpasses. Bei diesem wird der Landeanflug simuliert, aber nicht bis zum Ende durchgeführt. Die Drohne soll lediglich in eine Höhe von 5 bis 6 m die Landebahn passieren.

8.1 Ziel

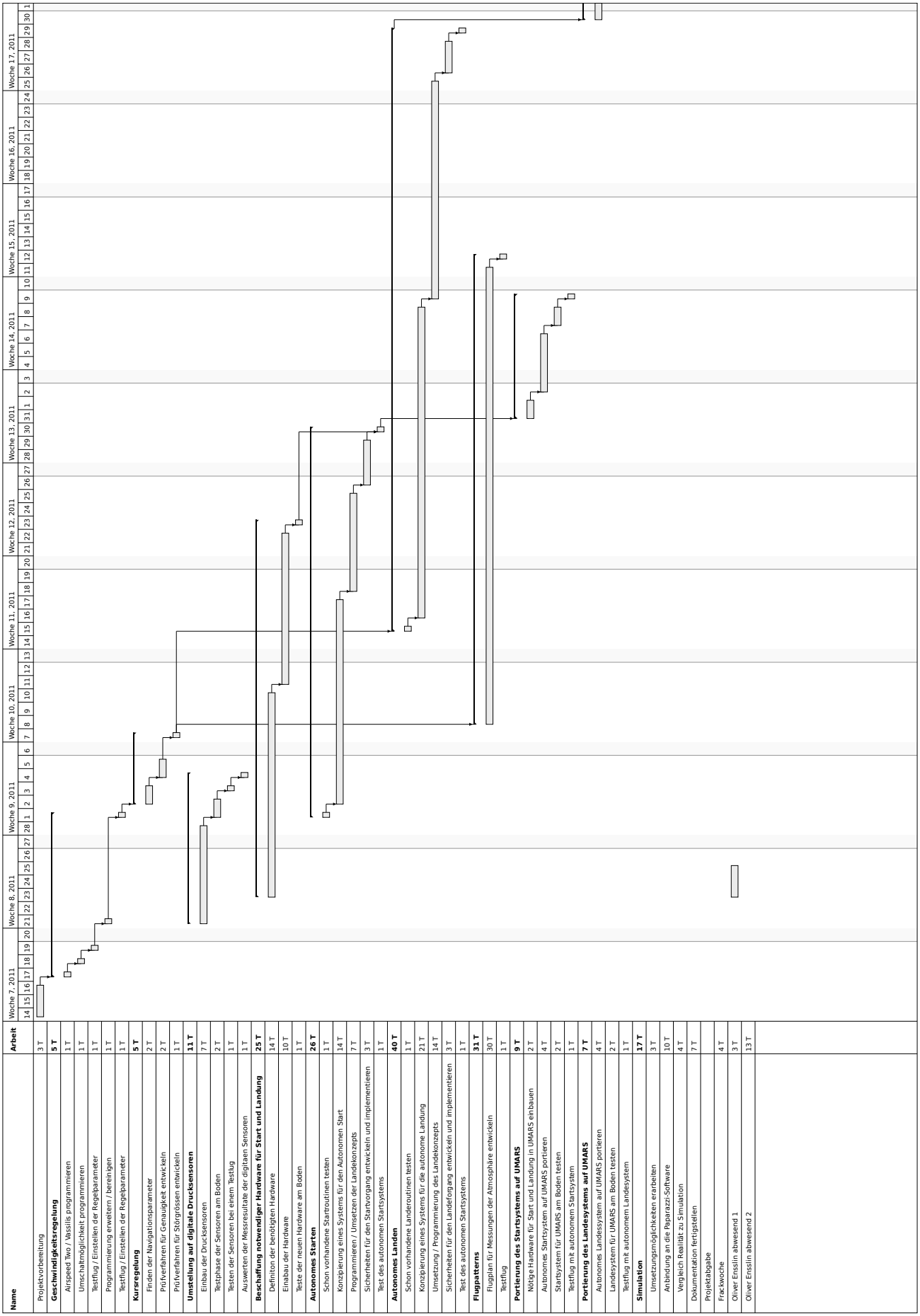
Ziel ist es heraus zu finden, wie sich die Drohne bei einem Späteren Landeanflug in geringer Höhe verhält. Ausserdem soll als Höhengeber der eingebaute Ultraschallsensor verwendet werden.

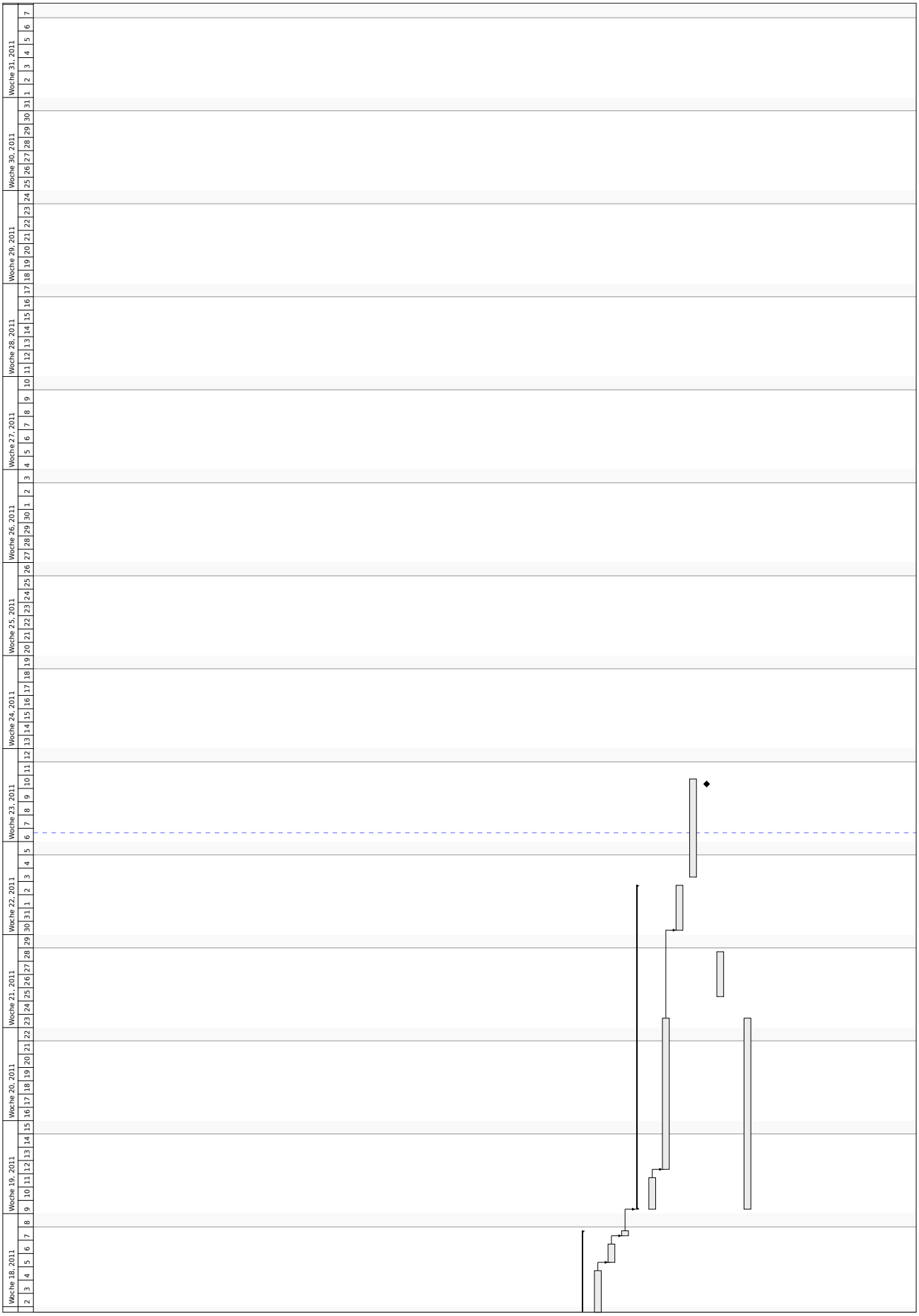
9 (Vasillis parametrieren)

Sollte noch genügend Zeit vorhanden sein, wäre es gut, wenn auch die Airspeed-Regelung nach Vassillis parametriert würde.

9.1 Ziel

Ziel ist auch hier die Regelung so weit zu verfeinern, dass die Drohne ein stabiles, den Anforderungen entsprechendes Flugverhalten zeigt.


Name
Arbeit
Woche 7, 2011
Woche 8, 2011
Woche 9, 2011
Woche 10, 2011
Woche 11, 2011
Woche 12, 2011
Woche 13, 2011
Woche 14, 2011
Woche 15, 2011
Woche 16, 2011
Woche 17, 2011
Projektvorbereitung 3 T
Geschwindigkeitsregelung 5 T
Airspeed Two / Vassils programmieren 1 T
Umschaltmöglichkeit programmieren 1 T
Testflug / Einstellen der Regelparameter 1 T
Programmierung erweitern / bereinigen 1 T
Testflug / Einstellen der Regelparameter 1 T
Kursregelung 5 T
Finden der Navigationsparameter 2 T
Prüfverfahren für Genauigkeit entwickeln 2 T
Prüfverfahren für Störgrössen entwickeln 1 T
Umstellung auf digitale Drucksensoren 11 T
Einbau der Drucksensoren 7 T
Testphase der Sensoren am Boden 2 T
Testen der Sensoren bei einem Testug 1 T
Auswerten der Messresultate der digitaen Sensoren 1 T
Beschaffung notwendiger Hardware für Start und Landung 25 T
Definiton der benötigten Hardware 14 T
Einabau der Hardware 10 T
Teste der neuen Hardware am Boden 1 T
Autonomes Starten 26 T
Schon vorhandene Startroutinen testen 1 T
Konzipierung eines Systems für den Autonomen Start 14 T
Programmieren / Umsetzen der Landekonzepts 7 T
Sicherheiten für den Startvorgang entwickeln und implementieren 3 T
Test des autonomen Startsystems 1 T
Autonomes Landen 40 T
Schon vorhandene Landeroutinen testen 1 T
Konzipierung eines Systems für die autonome Landung 21 T
Umsetzung / Programmierung des Landekonzepts 14 T
Sicherheiten für den Landefongang entwickeln und implementieren 3 T
Test des autonomen Startsystems 1 T
Flugpatterns 31 T
Flugplan für Messungen der Atmosphäre entwickeln 30 T
Testflug 1 T
Portierung des Startsystems auf UMARS 9 T
Nötige Hardware für Start und Landung in UMARS einbauen 2 T
Autonomes Startsystem auf UMARS portieren 4 T
Startsystem für UMARS am Boden testen 2 T
Testflug mit autonomem Startsystem 1 T
Portierung des Landesystems auf UMARS 7 T
Autonomes Landessystem auf UMARS portieren 4 T
Landesystem für UMARS am Boden testen 2 T
Testflug mit autonomem Landesystem 1 T
Simulation 17 T
Umsetzungsmöglichkeiten erarbeiten 3 T
Anbindung an die Paparazzi-Software 10 T
Vergleich Realität zu Simulation 4 T
Dokumentation fertigstellen 7 T
Projektabgabe
Frackwoche 4 T
Oliver Enssln abwesend 1 3 T
Oliver Enssln abwesend 2 13 T


Woche 18, 2011
Woche 19, 2011
Woche 20, 2011
Woche 21, 2011
Woche 22, 2011
Woche 23, 2011
Woche 24, 2011
Woche 25, 2011
Woche 26, 2011
Woche 27, 2011
Woche 28, 2011
Woche 29, 2011
Woche 30, 2011
Woche 31, 2011

Protokoll vom 22.02.2011

Ort, Zeit, Datum:	ZHAW TB 145, Winterthur, 14:30 Uhr, 22.02.2011
Teilnehmer:	Dr. W. Siegl; O.Ensslin; L. Chelini; D. Lange

Traktanden

1. Besprechung der Aufgabenstellung
2. Besprechung der Anforderungsliste

Beschlüsse

1. Die Genauigkeit der Verfolgung der Trajektorie soll sowohl optisch mittels Toleranzband, als auch durch eine Fehlerquadratsumme der X/Y-Position und der Höhe bewertet werden können. Die Windgeschwindigkeit und Richtung muss zu diesem Zweck ebenfalls aufgezeichnet werden.
2. Die Testtrajektorie wird eine genormte 8-Figur sein, welche sowohl von der Wind abgelegenen Seite, als auch von der Windseite angeflogen wird.
3. Die Sensoren, welche für die Höhenmessung benötigt werden, müssen von Herstellen oder anderen Quellen auf Ihre Funktionsweise auf Gras (Wiese) und Erde (Acker) bei ca. 60 km/h gewährleistet werden.
4. Die Flugpattern müssen, wie die Zeichnung von O. Ensslin beschreiben, programmiert werden. Diese werden bei der Vorführung in Frankreich vorgeführt.
5. Der Einsatz von JSBSim wird noch abgeklärt
6. Das maximale Toleranzfeld der Flugrute wird noch abgeklärt.

Weiteres Vorgehen

Folgende Aufgaben müssen als nächstes erledigt werden: siehe Pendenzen-liste.

Termine

Nächste Sitzung: 02.03.2011

Datum:	22.02.2011
Protokollführer:	L. Chelini

Pendenzenliste

Nr.	Pendenzen	Wer	Wann
01	Ausarbeiten der Airspeedregelung	chelilea, langede0	02.03.11
02	Spannungswandler von Platine auswechseln	chelilea	02.03.11
03	Flashen des in das Kästchen gebauten Chips testen	chelilea, langede0	02.03.11
04	Erste Version der Anforderungsliste	langede0	02.03.11
05	Zeitplan überarbeiten	chelilea, langede0	02.03.11
06	Abklärungen bezüglich Höhensensoren treffen	chelilea, langede0	02.03.11

Protokoll vom 02.03.2011

Ort, Zeit, Datum:	ZHAW TV 405, Winterthur, 15:00 Uhr, 02.03.2011
Teilnehmer:	Dr. W. Siegl; O.Ensslin; L. Chelini; D. Lange; L. Marchesini; T. Suter; M. Winet

Traktanden

1. Simulator

Beschlüsse

1. Falls ein Simulator erstellt wird, wird dieser mit D6 (6 degrees of freedom-Software) in die Paparazzi-Software eingebunden.

Weiteres Vorgehen

Aus dieser Besprechung fallen vorläufig keine weitere Pendenzen an

Termine

Nächste Sitzung: 08.03.2011

Datum:	02.03.2011
Protokollführer:	L. Chelini

Protokoll vom 08.03.2011

Ort, Zeit, Datum:	ZHAW TB 145, Winterthur, 14:30 Uhr, 08.03.2011
Teilnehmer:	Dr. W. Siegl; O.Ensslin; L. Chelini; D. Lange

Traktanden

1. Landezyklen
2. Startzyklen
3. Höhenmessung
4. Prantl-Sonde

Beschlüsse

1. Die vorhandenen Start- und Landezyklen werden vorerst einmal getestet
2. Das Barometer und Sonar Modul muss neu geschrieben werden.
3. „einfachere" - Airspeed Regelungen ergeben wahrscheinlich bessere Resultate (müssen programmiert werden)
4. Die Prantl-Sonde muss kalibriert und referenziert werden.

Weiteres Vorgehen

Folgende Aufgaben müssen als nächstes erledigt werden: siehe Pendenzen-liste.

Termine

Nächste Sitzung: 18.03.2011

Datum:	08.03.2011
Protokollführer:	L. Chelini

Pendenzenliste

Nr.	Pendenzen	Wer	Wann
01	Neue Airspeedregelungen	chelilea	18.03.11
02	Vorhandene Start und Landezyklen untersuchen	langede0	18.03.11
03	Module für Baro und Sonar schreiben	chelilea	18.03.11
04	Prantlsonde kalibrieren / referenzieren	chelilea, langede0	18.03.11

Protokoll vom 18.03.2011

Ort, Zeit, Datum:	ZHAW TB 145, Winterthur, 14:30 Uhr, 18.03.2011
Teilnehmer:	Dr. W. Siegl; L. Chelini; D. Lange

Traktanden

1. Sonar-Sensor
2. estimator
3. I2C
4. Landung
5. Airspeed Regelung

Beschlüsse

1. Das Sonarsensor-Modul soll durch einen Resetbutton im Flug (>10 m über Boden) geresettet werden können. Dabei wird der momentane Messwert den 7.65 m (Maximale Distanz des Sensors) zugeteilt. (Für die Berechnung)
2. Im extimator.c wird eine Methode benötigt, mit welcher das Umschalten der Sensordaten (GPS, Baro, Sonar) ermöglicht
3. Für die fehlerhaften Werte der I2C Übermittlung des Baros soll gegebenenfalls der Hersteller nähere Auskunft geben.
4. Die Landung sieht in einem ersten Schritt wie folgt aus:
   1. spiralförmiges Absinken auf 30 m
   2. In der Verlängerung der Landepiste (linear zur Landepiste) den Anflug beginnen (sinken)
   3. Sobald der Sonar einen Wort von z.B. 6 m angibt wird auf konstanter Höhe weitergeflogen.
   4. Bei erreichen eines vorraffinierten GPS Punktes wird ein konstanter Sinkflug angestrebt (Motor ausgeschaltet)
5. Die fünf neuen Airspeed Regelungen werden beim nächsten Testflug getestet.

Weiteres Vorgehen

Folgende Aufgaben müssen als nächstes erledigt werden: siehe Pendenzen-liste.

Termine

Nächste Sitzung: 23.03.2011

Datum:	18.03.2011
Protokollführer:	L. Chelini

Pendenzenliste

Nr.	Pendenzen	Wer	Wann
01	I2C - Problem beheben	chelilea	22.03.11
02	Sonar - Modul für Testflug vorbereiten	chelilea	22.03.11
03	Flugzeug Reparieren	langede0	22.03.11
04	Sonar einbauen	langede0	22.03.11
05	Start und Landezyklen ausprobieren	chelilea, langede0	22.03.11

Protokoll vom 22.02.2011

Ort, Zeit, Datum:	ZHAW TB 145, Winterthur, 14:30 Uhr, 22.02.2011
Teilnehmer:	Dr. W. Siegl; O.Ensslin; L. Chelini; D. Lange

Traktanden

1. Besprechung der Aufgabenstellung
2. Besprechung der Anforderungsliste

Beschlüsse

1. Die Genauigkeit der Verfolgung der Trajektorie soll sowohl optisch mittels Toleranzband, als auch durch eine Fehlerquadratsumme der X/Y-Position und der Höhe bewertet werden können. Die Windgeschwindigkeit und Richtung muss zu diesem Zweck ebenfalls aufgezeichnet werden.
2. Die Testtrajektorie wird eine genormte 8-Figur sein, welche sowohl von der Wind abgelegenen Seite, als auch von der Windseite angeflogen wird.
3. Die Sensoren, welche für die Höhenmessung benötigt werden, müssen von Herstellen oder anderen Quellen auf Ihre Funktionsweise auf Gras (Wiese) und Erde (Acker) bei ca. 60 km/h gewährleistet werden.
4. Die Flugpattern müssen, wie die Zeichnung von O. Ensslin beschreiben, programmiert werden. Diese werden bei der Vorführung in Frankreich vorgeführt.
5. Der Einsatz von JSBSim wird noch abgeklärt
6. Das maximale Toleranzfeld der Flugrute wird noch abgeklärt.

Weiteres Vorgehen

Folgende Aufgaben müssen als nächstes erledigt werden: siehe Pendenzen-liste.

Termine

Nächste Sitzung: 02.03.2011

Datum:	22.02.2011
Protokollführer:	L. Chelini

Pendenzenliste

Nr.	Pendenzen	Wer	Wann
01	Ausarbeiten der Airspeedregelung	chelilea, langede0	02.03.11
02	Spannungswandler von Platine auswechseln	chelilea	02.03.11
03	Flashen des in das Kästchen gebauten Chips testen	chelilea, langede0	02.03.11
04	Erste Version der Anforderungsliste	langede0	02.03.11
05	Zeitplan überarbeiten	chelilea, langede0	02.03.11
06	Abklärungen bezüglich Höhensensoren treffen	chelilea, langede0	02.03.11

Protokoll vom 24.03.2011

Ort, Zeit, Datum:	ZHAW TV 405, Winterthur, 17:00 Uhr, 24.03.2011
Teilnehmer:	B. Neininger; L. Chelini; D. Lange

Traktanden

1. Prantlsonde - Messdaten
2. Position der Prantlsonde
3. Wichtigkeit von Airspeed und Höhe
4. Flugpattern
5. Landung

Beschlüsse

1. Normale Turbulenzen können der True Airspeed maximal mit +/-1 m/s beeinflussen
2. Gegebenenfalls Teflonschläuche für die Druckleitung verwenden
3. Maximales Rauschen des Differenzdruck-Sensors soll 0.1 Hektopascal betragen
4. Vibrationen können Drucksensoren beeinflussen
5. Position der Prantlsonde war OK, wird jedoch noch weiter aussen am Flügel montiert
6. „Electomagnetic Interfenece" des Motors könnten Messdaten stören
7. Wichtigkeit der Telemetrie in absteigender Reihenfolge:
   1. Keine ruckartigen/nervöse Regelungen
   2. Höhe regeln +/- 5 m in 10 Sekunden
   3. Horizontale Position +/- 20 m in 3 Sekunden
   4. True Airspeed soll ein bestimmtes Minimum bzw. Maximum nicht über bzw. unterschreiten
8. Landezyklus kann fürs Erste wie geplant durchgeführt werden
9. Der True Airspeed darf 1.5 * minimal Airspeed (Strömungabriss) bei der Landung nicht unterschreiten

Weiteres Vorgehen

Folgende Aufgaben müssen als nächstes erledigt werden: siehe Pendenzen-liste.

Termine

Nächste Sitzung: 12.04.2011

Datum:	24.03.2011
Protokollführer:	L. Chelini

Pendenzenliste

Nr.	Pendenzen	Wer	Wann
01	Vorhandener Bungee-Start importieren	chelilea, langede0	30.03.11
02	Eigener Landezyklus ausarbeiten	chelilea, langede0	30.03.11
03	Filter gegebenenfalls anpassen	chelilea	30.03.11
04	Flugzeug reparieren	chelilea, langede0	30.03.11
05	Störquelle der Sensoren lokalisieren	chelilea, langede0	30.03.11

Protokoll vom 12.04.2011

Ort, Zeit, Datum:	ZHAW TB 145, Winterthur, 13:30 Uhr, 12.04.2011
Teilnehmer:	Dr. W. Siegl; O.Ensslin; L. Chelini; D. Lange

Traktanden

1. Bewertung des Flugverhaltens
2. Vorgaben an Flugregelung
3. Magnetometer
4. Flugplanung
5. Zusätzlicher I2C Bus
6. Parameterliste

Beschlüsse

1. Für die Bewertung des Flugverhaltens wird ein neues Modul geschrieben, welches die Fehlerquadratsumme der Airspeed- , Altitude- und Kursabweichung berechnet und an das GCS sendet.
2. Die Fehlerquadratsumme wird nach jedem abgeschlossenem Rundflug auf Null zurückgesetzt.
3. Die Toleranz der Fehler kann während des Fluges verändert werden.
4. Um die definitiven Vorgaben an die Flugregelung zu ermitteln, wird eine E-Mail an die beteiligten Personen gesendet.
5. Der Magnetometer muss für den Bungee in Betrieb genommen werden.
6. Vor den nächsten Testflügen wird ein grober Flugplan erstellt.
7. Ein separater I2C Kanal speziell für die IMU wird nicht implementiert. (evtl. später)
8. Um die Parameterfindung zu organisieren, wird eine Parameteränderungsliste während den Testflügen geführt.

Weiteres Vorgehen

Folgende Aufgaben müssen als nächstes erledigt werden: siehe Pendenzen-liste.

Termine

Nächste Sitzung: 27.04.2011

Datum:	12.04.2011
Protokollführer:	L. Chelini

Pendenzenliste

Nr.	Pendenzen	Wer	Wann
01	Email für Vorgabenbestimmung	chelilea, langede0	19.04.11
02	Programmierung des Bewertungsmoduls	chelilea	19.04.11
03	Magnetometer in Betrieb nehmen	chelilea	19.04.11
04	Flugplanungsformular	langede0	19.04.11
05	Parameteränderungsliste	langede0	19.04.11
06	Bungeestart überarbeiten	Langede0	19.04.11

Protokoll vom 27.04.2011

Ort, Zeit, Datum:	ZHAW TB 145, Winterthur, 13:00 Uhr, 27.04.2011
Teilnehmer:	Dr. W. Siegl; L. Chelini; D. Lange

Traktanden

1. Inhaltsverzeichnis der Dokumentation
2. Anforderungsliste

Beschlüsse

1. Inhaltsverzeichnis wurde angepasst und kann zum grossen Teil so übernommen werden
2. Anforderungsliste wird nochmals überarbeitet und korrigiert

Weiteres Vorgehen

Folgende Aufgaben müssen als nächstes erledigt werden: siehe Pendenzen-liste.

Termine

Nächste Sitzung: 04.05.2011

Datum:	27.04.2011
Protokollführer:	L. Chelini

Pendenzenliste

Nr.	Pendenzen	Wer	Wann
01	Inhaltsverzeichnis überarbeiten	langede0	04.05.11
02	Anforderungsliste überarbeiten	chelilea	04.05.11
03	Landeanflug verbessern	langede0	04.05.11
04	Startalgorithmus programmieren	chelilea, langede0	04.05.11
05	Benchamrk-Modul verbessern	chelilea	04.05.11

Protokoll vom 04.05.2011

Ort, Zeit, Datum:	ZHAW TB 145, Winterthur, 14:00 Uhr, 04.05.2011
Teilnehmer:	Dr. W. Siegl; L. Chelini; D. Lange

Traktanden

1. Anforderungsliste

Beschlüsse

1. Anforderungsliste wird nochmals überarbeitet und korrigiert

Weiteres Vorgehen

Folgende Aufgaben müssen als nächstes erledigt werden: siehe Pendenzen-liste.

Termine

Nächste Sitzung: 17.05.2011

Datum:	04.05.2011
Protokollführer:	L. Chelini

Pendenzenliste

Nr.	Pendenzen	Wer	Wann
01	Anforderungsliste überarbeiten	chelilea, langede0	17.05.11
02	Dokumentation	chelilea, langede0	17.05.11

Protokoll vom 17.05.2011

Ort, Zeit, Datum:	ZHAW TB 145, Winterthur, 14:30 Uhr, 17.05.2011
Teilnehmer:	Dr. W. Siegl; L. Chelini; D. Lange

Traktanden

1. Anforderungsliste
2. Angle of Attack Sensor

Beschlüsse

1. Die Anforderungsliste ist bereit für die Unterschriften der beteiligten Personen
2. Es wird ein Angle of Attack Sensor gebaut, welcher genaue Angaben über den Anströmwinkel liefert.

Weiteres Vorgehen

Folgende Aufgaben müssen als nächstes erledigt werden: siehe Pendenzen-liste.

Termine

Keine weiteren Sitzungen geplant.

Datum:	17.05.2011
Protokollführer:	L. Chelini

Pendenzenliste

Nr.	Pendenzen	Wer	Wann
01	Start- und Landealgorithmen überarbeiten	langede0	02.06.11
02	Drehgeber definieren, suchen und bestellen	chelilea	02.06.11
03	Angle of Attack Sensor implementieren	chelilea, langede0	02.06.11

Wochenjournal

Flugregelung einer Kleindrohne

Dieser Abschnitt dient der Übersicht über die erledigten Arbeiten, den jeweiligen Stand der Arbeit und was noch getan werden muss. Am Ende jeder Woche wurde ein Eintrag verfasst. So sollte der Überblick über das Projekt nicht verloren gehen, ausserdem war das Führen des Journals jeweils eine gute Gelegenheit für ein Revue über die letzte Woche und einen Abgleich mit unserem Terminplan.

Woche 1

Der Anfang der ersten Woche wurde für die Einarbeitung in das Projekt investiert. Unklarheiten wurden diskutiert und es wurde das individuelle Vorwissen untereinander ausgetauscht. Ausserdem wurde die, vom Projektleiter verteilte Aufgabenstellung näher betrachtet und Unklarheiten notiert. In der Mitte der ersten Woche wurde dann bereits mit der programmatischen Erweiterung des vorhandenen Autopiloten begonnen. Es wurden zwei verschiedene Regelungen für die Geschwindigkeit der Umgebungsluft (Airspeed) implementiert. Ausserdem wurde ein Umschalten (Switch) zwischen diesen zwei Regelungen und der Standartregelung während des Fluges realisiert. So konnte bereits am Ende der ersten Woche ein Testflug mit der Drohne Maja gemacht werden, bei welchem sich herausstellte, dass der Switch einwandfrei funktionierte.

Woche 2

In der zweiten Woche beschäftigten wir uns zuerst mit dem Simulator, mit welchem der Autopilot simuliert werden können sollte. Doch das anbinden der dazu benötigten Software (JSBSim und FlightGear) gestaltete sich schwieriger als gedacht. Schlussendlich konnte allerdings ein Flug mit einer Chessna simuliert werden.
Ein weiterer Punkt in dieser Woche war der Umstieg von analogen auf digitale Drucksensoren. Dazu mussten diese verlötet werden und es musste eine neue Schnittstelle namens I2C in Betrieb genommen werden.
Der letzte Arbeitspunkt in dieser Woche war der Einbau des kompletten Autopiloten in eine handliche Box, welche nun innert Minuten von Drohne zu Drohne verschoben werden kann.

Woche 3

Die dritte Woche sollte sehr interessant werden, denn es wurde angekündigt, dass wir am Dienstag die Möglichkeit haben werden, mit der UMARS-Drohne fliegen zu gehen. Der Auftrag war es, die Regelparameter der Drohne während des Fluges zu verfeinern. Mit dem sogenannten Trimmen sollten die Flugeigenschaften im Geradeausflug und in der Kurve verbessert werden.
Gegen Ende der Woche sollte noch der Erstflug mit der neuen Drohne, einem Mentor gemacht werden. Daher mussten noch zwei Tage in den Einbau des Autopiloten in diesen investiert werden. Ausserdem mussten noch Kanäle für die Kabel der Beleuchtung und für die Rohre der Prandtlsonde in die Flügel gefräst werden. Am Ende der Woche konnte dann zum Erstflug aufgebrochen werden, bei welchem sich zeigte, dass der Mentor bessere Flugeigenschaften als die Maja aufwies. So konnten bereits am ersten Flugtag die Grössenordnung aller Parameter für die zwei Airspeed Regelungen und für die Groundspeedregelung (Starndartregelung) bestimmt werden. Bei einem weiteren Flugtag mussten diese aber noch verfeinert werden.

Woche 4

In der vierten Woche konnten wir uns erstmals genauer mit dem autonomen Start und der autonomen Landung beschäftigen. Wir fällten die Entscheidung, einen Ultraschallsensor für die Höhenmessung kurz vor der Landung zu verwenden. Bei der Recherche über die Eigenschaften von verschiedenen Sensortypen hatte sich gezeigt, dass ein solcher Sensor unsere Anforderungen mit wenig Aufwand am besten erfüllen würde. Im Kontakt mit der Paparazzi-Online-Community zeigte sich ausserdem, dass sich bei unserer Pitch-basierten Airspeed-Regelung noch ein Fehler eingeschlichen hatte. Nachdem die Regelung umprogrammiert wurde, konnte dann ein weiterer Testflug durchgeführt werden. Bei dem Testflug und beim Tuning der verschiedenen Regelungen entstanden drei weitere Ideen für Airspeed-Regelungen. Die Umsetzung dieser Ideen war jedoch mit grossem Programmieraufwand verbunden und konnten daher nicht auf dem Feld programmiert werden. Gegen Ende des Flugtages testeten wir noch die Bodenstart-Einrichtung, bei welcher die Drohne durch ein Gummiseil beschleunigt und einige Meter in die Luft gebracht wird. Durch die in der Anfangsphase des Starts auftretende, starke Beschleunigung der Drohne befürchteten wir ein Fehlfunktion der IMU. Doch bei den Tests, bei welchem die Drohne von Hand gesteuert wurde, trat die befürchtete Fehlfunktion nicht auf.
Am Ende der Woche wurden dann noch die auf dem Feld entstandenen Airspeed-Regelungen ausprogrammiert, so dass sie beim nächsten Flug getestet werden konnten. Ausserdem wurde die bereits vorhandene Landeroutine untersucht.

Woche 5

Zu Beginn der fünften Woche sollte unsere Prandtlsonde genauer überprüft werden. Dazu wurde zuerst ein Versuch durchgeführt, bei dem die Sonde mit einem Ventilator angeströmt wurde und die Strömungsgeschwindigkeit mittels Paparazzi-Software abgelesen wurde. Dieser Wert wurde dann mit dem eines Anemometers verglichen. Bei diesem Versuch zeigte sich dann aber, dass das Strömungsbild des Ventilator zu unregelmässig war und so entschieden wir uns den Versuch in einem Windkanal zu wiederholen. Die Messungen im Windkanal der ZHAW zeigten dann, dass für die Berechnung des Differenzdrucks das falsche Datenblatt verwendet wurde. Dies war auf die schlechte Beschriftung des Sensors zurückzuführen. Mit den richtigen Werten ergab sich dann auch die richtigen Strömungsgeschwindigkeiten.
In der Mitte der Woche konnten wir uns dann erstmals mit dem Landeablauf und dessen Umsetzung auseinander setzen. Ausserdem wurden die bereits vorhandenen Start- und Landeroutinen betrachtet.
Des weiteren wurde in dieser Woche der Ultraschallsensor in Betrieb genommen um in der Übungsdrohne verbaut. Der Sensor war nun Bereit für einen ersten Testflug, welcher auf die nächste Woche angesetzt wurde. Gegen Ende der Woche sollte noch der Drucksensor für den barometrischen Druck in Betrieb genommen werden. Mit der Hilfe des so gemessenen Drucks sollte eine weitere Höhenabschätzung gemacht werden können. Doch es zeigten sich einige Probleme mit dem Übertragen der Informationen an den Autopiloten. Es erforderte einiges an Zeit und Geduld, bis der Fehler in der Software gefunden werden konnte. Am Ende der Woche war der Fehler behoben und auch der Baro-Sensor war bereit für den Testflug in der nächste Woche.

Woche 6

Beim Testflug in dieser Woche hatten wir sehr viele Neuheiten zu testen. Zum einen hatten wir die Software um vier neue Airspeed-Regelungen erweitert und zum anderen hatten wir zwei neue Sensoren, welche zum ersten mal im Einsatz waren.
Beim tunen der neuen Airspeed-Regelungen gelangten wir schnell zu Regelparametern, welche ein ruhiges aber dennoch exaktes Folgen der vorgegebenen Trajektorie erlaubten. Leider konnten wir aber nicht alle neuen Regelsysteme parametrieren, da das Finden der entsprechenden Parameter einige Zeit in Anspruch nimmt. Ausserdem stellten wir fest, dass unser Differenzdrucksensor für die Messung des True-Airspeeds ein sehr stark schwingendes Signal ausgab, was es unmöglich machte den Airspeed in einem Band von 1 m/s zu halten.
Am Ende des Flugtages testeten wir dann noch den Sensor für den barometrischen Druck und den Ultraschallsensor. Um eine Aussage über die Genauigkeit des barometrischen Drucks zu machen, aus welchem wir eine barometrische Höhe berechnen, verglichen wir die Baro-Höhe mit derjenigen, welche uns das GPS ausgab. Wir stellten fest, dass diese Werte gut korrelieren.
Für einen Test des Ultraschallsensors, welcher nur eine Reichweite von 7.65 m hat, musste Herr Oliver Ensslin das Flugzeug im Tiefflug über die Landebahn fliegen. Die aufgezeichneten Werte wurden später analysiert.
Am Ende der Woche baten wir Herrn Bruno Neininger noch um eine Sitzung, von welcher wir uns erhofften, die Ursache für die starken Schwingungen in der Airspeed-Messung finden zu können. Wir erhielten einige gute Inputs mit möglichen Ursachen für die starken Schwankungen von Herrn Neininger .

Woche 7

Im Gespräch mit Herrn Neininger am Ende der letzten Woche hatten sich die weichen Schläuche, welche als Verbindung zwischen Prandtlsonde und Drucksensor dienten, als mögliche Ursache für die Schwankungen herausgestellt. Diese weichen Schläuche sind empfindlich gegen Druckstösse oder Vibrationen von Aussen und müssen daher durch harte ersetzt werden. Nach dem Austausch der Schläuche führten wir einige Tests mit Druckluft durch, bei welchen wir die gesamte Elektronik des Autopiloten Schritt für Schritt einschalteten, um andere mögliche Störungsquellen detektieren zu können. Es stellte sich heraus, dass der Motor leichte Störungen erzeugt. Diese sind jedoch nicht in der gleichen Grössenordnung wie diejenigen, die wir bei den Messungen in der Luft hatten. Da wir nicht alle Störungen aus dem Signal beseitigen konnten, entschieden wir uns das Signal durch einen PT1-Filter filtern zu lassen.
Für den Flugtag in der nächsten Woche wurden noch die vorhandenen Routinen für Start und Landung in unser Paparazzi-Projekt integriert, was sich als schwieriger als erwartet herausstellte. Da das Wiki nicht auf dem neusten Stand war, mussten wir sehr viel Zeit investieren, um dem Start zu Implementieren. Da es sich aber um ein so kleines Detail handelte, welches uns fehlte musste zuletzt die Community zu Rate gezogen werden.
Weil wir nun beide immer parallel an unserem Paparazzi-Projekt arbeiteten, entschieden wir uns, um Änderungen einfacher übernehmen zu können, unser Projekt über Git online zu stellen. Doch auch hier gab es aufgrund fehlender Dokumentationen einige Probleme und es musste viel Zeit investiert werden. Doch nun ist unser Projekt auf dem Internet und damit auch für andere User von Paparazzi zugänglich, was schliesslich auch der Grundgedanke von Open-Source ist.

Woche 8

In der achten Woche konzentrierten wir uns ganz auf den nächsten Flugtag, welcher für Donnerstag geplant war. An diesem Tag standen wieder einige Tests von neuen Funktionen aus. Es sollten sowohl der bereits vorhandenen Bungee-Start, als auch die Landung des OSAM[1]-Teams getestet werden. Der Beginn der Woche wurde also für die nötigen Vorkehrungen für den Flug genutzt. Auch wurde die entsprechenden Code Seiten nochmals genau angeschaut. Auf dem Flugplatz testeten wir dann als erstes einige Male den Bungee-Start. Dabei stellte sich heraus, dass hier noch einige Probleme vorhanden sind. Zum einen ist es schwierig, den Bungee Waypoint an der richtigen stelle zu setzten und zum anderen folgt die Drohne nur schlecht dem vorgegebenen Pfad. Ausserdem schaltet sich der Motor erst sehr spät ein.
Auch der Ablauf der Landung ist für unsere Anforderungen nicht ausreichend. Auch hier bestehen Probleme mit dem Folgen der vorgegebenen Trajektorie und eine Abfrage der Ultraschallhöhe ist ebenfalls nicht implementiert. Es besteht also bei Start und Landung grosser Verbesserungsbedarf von unserer Seite her.
Die restliche Zeit des Flugtages nutzen wir noch um die beiden Airspeed-Regelungen „Air Speed Pitch Simple“ und „Air Speed Pitch Acceleration“ zu testen und die Richtigen Regelparameter einzustellen.

1 Entwicklungsteam der Utah State University

Woche 9

Bis jetzt musste beim Einschalten der Drohne immer darauf geachtet werden, dass diese ganz gerade ausgerichtet ist, da der verwendetet Code diese Position als Initial-Position annahm. Dies ist sehr mühsam und es ist auch schwierig zu gewährleisten, dass die Drohne auch auf dem Feld immer gleich ausgerichtet wird. Nun wurde aber eine neue Version des IMU-Codes veröffentlicht, bei welchem die Initial-Position einmal eingestellt werden und im Anschluss gespeichert werden kann.
Bei früheren Testflügen auf dem Feld haben wir ausserdem fest gestellt, dass unser Downlink, also die Verbindung zwischen Drohne und GCS mit dem erhöhten Datenvolumen, welches gesendet werden muss, langsam an seine Grenzen kommt. Daher entschieden wir uns, die Baud-Rate für diese Verbindung zu erhöhen.

Woche 10

In dieser Woche zeigte sich einmal mehr der „Vorführeffekt“. Als wir zum ersten Mal Herrn Siegl zu einer Präsentation der Ergebnisse auf dem Feld einluden funktionierte zu Beginn gar nichts mehr. Wir hatten Verbindungsprobleme, der Motor war beschädigt, beim Start navigierte die Drohne zu stark und die Parameter für die Standart-Regelung waren falsch. In der Vorangegangenen Woche hatten wir die Baudrate für das Xbee-Modem erhöht und ein Pairing zwischen Sender und Empfänger eingerichtet. Dadurch kamen die Verbindungsprobleme zu Stande, wieso genau dies nicht funktionierte konnten wir jedoch nicht eruieren. Nachdem alles wieder auf die ursprünglichen Einstellungen zurück gesetzt war, war die Verbindung wieder stabil. Die Probleme mit dem Motor kamen wohl von dem leichten Absturz, den wir in der Woche zuvor beim Test der vorhandenen Landung hatten. Die Motorwelle war leicht verbogen, wodurch bei einer hohen Drehzahl ein Schwingen entstand. Da der Motor im unteren Drehzahlbereich jedoch noch funktionierte, konnte trotzdem noch geflogen werden. Das Problem mit dem zu starken Navigieren beim Start war darauf zurück zu führen, dass die Drohne aufgrund des fehlenden Kompass im Stillstand nicht weiss, wie sie ausgerichtet ist. Es galt also den Kompass für die nächsten Starttests einzubinden.

Woche 11

Schon vor einiger Zeit hatten sich Forderungen nach einer Möglichkeit zur Überprüfung der Regelgüte während dem Flug laut gemacht. Daher entwickelten wir in dieser Woche ein Modul, welches die Fehlerquadratsumme der einzelnen Regelparameter berechnet und in Real-Time am GCS ausgegeben werden kann. Für die Höhenabweichung und die Airspeed-Abweichung war diese Rechnung relativ einfach und schnell gemacht. Aber bei der Kursabweichung zeigten sich einige Probleme, da der Autopilot so aufgebaut ist, dass die Drohne stets auf einen bestimmten Punkt zufliegt und nicht einer gegebenen Linie folgt. Ausserdem kann die Abweichung auf einer geraden Strecke nicht gleich berechnet werden, wie während einer Kurve. Die Lösung war, dass diese Strecke welcher die Drohne theoretisch folgen sollte durch das Modul berechnet wird und als Kursabweichung die kürzeste Entfernung zu dieser Strecke genommen wurde. Am Ende der Woche hatten wir noch die Möglichkeit das neue Modul, welches wir Benchmark tauften, zu testen. Nun kann beim Einstellen der Regelparameter ein Oval geflogen werden und nach jeder Umrundung wird die Fehlerquadratsumme automatisch zurück gesetzt. So kann eine

objektive Aussage über den Einfluss von den verschiedenen Regelparametern auf das Flugverhalten gemacht werden.
Ausserdem versuchten wir in dieser Woche den Kompass in die Software einzubinden, um die Navigationsprobleme beim Start zu beheben. Wir stellten jedoch fest, dass der Kompass beschädigt war und daher nicht funktionierte. Aus diesem Grund musste der Startalgorithmus so umprogrammiert werden, dass die Drohne zu beginn nicht navigiert.

Woche 12

Diese Woche war sehr stressig für uns und komplett ausgebucht mit verschiedenen Arbeiten. Da dies die letzte Woche war, in der Oliver Ensslin noch zu gegen war, hatten wir nur noch diese Woche Zeit um unsere Start- und Landealgorithmen zu testen. Zu unserem Glück hatten wir nun die Gelegenheit auf dem Flugplatz in Winterthur fliegen zu gehen, was uns einige Zeit für den Anfahrtsweg ersparte. Zu Beginn der Woche lag der Hauptfokus noch auf dem autonomen Start, bei welchem wir das Problem fest stellten, dass die Drohne nach dem Ausklinken aus dem Gummiseil immer etwas an Höhe verlor. Es machte sich der Verdacht breit, dass hier ein Problem mit der IMU vorliegen könnte, welches durch die starke Beschleunigung beim Start verursacht wird.
In der zweiten Hälfte der Woche konzentrierten wir uns dann voll und ganz auf die automatische Landung, welche wir am Mittwoch noch kurz testen konnten. Bei dem Test stellten wir fest, dass die Drohne noch immer mit einer sehr hohen Geschwindigkeit aufsetzte und das sie nicht in der Nähe des angegebenen Touch Down Punktes aufsetzte. Letzteres war auf einen Programmierfehler zurückzuführen, welcher bei der Überarbeitung des Codes am nächsten Tag gefunden und behoben wurde. Die zu hohe Geschwindigkeit vor dem Aufsetzten liess sich durch die Airspeed-Regelung erklären. Wir entschieden uns noch eine Regelung für den Anstellwinkel zu entwickeln. Beim letzten Testflug am Freitag konnten wir nun die Landung mit einem geregelten Anstellwinkel testen. Die Ergebnisse waren sehr zufrieden stellend. Die Drohne konnte mehrmals hintereinander, ohne Probleme und ohne manuelles Eingreifen landen.

Woche 13 und 14

Da in diesen zwei Wochen unser Pilot nicht anwesend war, konnten wir keine Testflüge mehr durchführen. Dies war allerdings schon lange bekannt und wir hatten geplant, in diesen Wochen einen grossen Teil der Dokumentation zu schreiben. Neben dem Dokumentieren suchten wir aber noch nach einer Lösung für das IMU-Problem nach dem starten. Wir kamen zu dem Ansatz, dass eine Regelung des Angle of Attack (AOA), also des Winkels mit welchem das Flugzeug von der Luft angeströmt wird, das Problem lösen könnte. Daher wurde ein Drehgeber bestellt, mit welchem eine Messeinrichtung für den Angle of Attack gebaut werden können wird.

Woche 15

In dieser Woche war die Frackwoche.

Woche 16

Zu Beginn der 16. Woche bauten wir eine Vorrichtung zur Messung des Angle of Attack und montierten diese an die rechte Tragfläche von unserer Drohne. Danach wurde der entsprechende Code für die Umrechnung des Signals des Drehgebers geschrieben und der Pitch-Loop wurde so umgeschrieben, dass es möglich ist den AOA zu regeln. Am nächsten Tag konnte ein weiterer Testflug angesetzt werden, um zu testen, ob der Start nun problemlos abläuft. Doch leider spielte das Wetter an unserem letzten Flugtag nicht ganz mit. Es war extrem windig und so konnte der Start nicht getestet werden. Es war jedoch möglich den AOA Sensor zu testen und auch mit der AOA-Regelung zu fliegen. Daher konnte trotz schlechtem Wetter ein Erfolg erzielt werden. Es sollte also in Zukunft mögliche sein den Start mit der AOA-Regelung durchzuführen.

Woche 17

In dieser Woche wurden hauptsächlich administrative Dinge erledigt. Die Dokumentation wurde fertig gestellt und zum Druck freigegeben. Es gab noch einige Dinge, wie das Poster, welcher ebenfalls noch bearbeitet werden mussten.
Am Ende der Woche wurde die Arbeit abgeschlossen und abgegeben. Es war ein sehr interessantes Projekt und wir sind mit dem Endresultat sehr zufrieden.

```
/*
 * $Id$
 *
 * Copyright (C) 2011  langede0
 *
 * This file is part of paparazzi.
 *
 * paparazzi is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2, or (at your option)
 * any later version.
 *
 * paparazzi is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with paparazzi; see the file COPYING.  If not, write to
 * the Free Software Foundation, 59 Temple Place - Suite 330,
 * Boston, MA 02111-1307, USA.
 *
 */
/** \file ZHAWNav_Takeoff.c
 *  \brief Bungee Takeoff / Bungee Takeoff Glide
 *
 */


#include "subsystems/navigation/ZHAWNav_Takeoff.h"
#include "firmwares/rotorcraft/autopilot.h"

#include "mcu_periph/uart.h"
#include "messages.h"
#include "downlink.h"

#ifndef DOWNLINK_DEVICE
#define DOWNLINK_DEVICE DOWNLINK_AP_DEVICE
#endif


/************** ZHAW Bungee Takeoff ********************************************/

/** Takeoff functions for bungee takeoff.
Run initialize function when the plane is on the bungee, the bungee is fully extended and you are
ready to
launch the plane. After initialized, the plane will follow a line drawn by the position of the plane
on initialization and the
position of the WP_TakeOffDirection (given in the arguments). Once the plane crosses the throttle
line, which is perpendicular to the line the plane is following,
and intersects the position of the ThrottlePoint (which has a fixed distance from the Initial Position
(TakeOff_Distance in airframe file) from the bungee just in case the bungee doesn't release directly
above the bungee) the prop will come on. The plane will then continue to follow the line until it has
reached a specific
height (defined in as Takeoff_Height in airframe file) above the bungee waypoint and speed (defined as
Takeoff_Speed in the airframe file).

<section name="Takeoff" prefix="Takeoff_">
  <define name="Speed" value="15" unit="m/s"/>
  <define name="Distance" value="10" unit="m"/>
  <define name="MinSpeed" value="5" unit="m/s"/>
</section>
 */

#ifndef TAKEOFF_DISTANCE
#define TAKEOFF_DISTANCE 10
#endif
#ifndef TAKEOFF_SPEED
#define TAKEOFF_SPEED 12
#endif
/*
#ifndef TAKEOFF_MINSPEED
#define TAKEOFF_MINSPEED 2
#endif
```

```
*/
enum TakeoffStatus { Launch, Throttle, Finished };
static enum TakeoffStatus CTakeoffStatus;
static float throttlePx;
static float throttlePy;
static float initialx;
static float initialy;
static float ThrottleSlope;
static bool_t AboveThrottleLine;
static float BungeeAlt;
static float TDistance;
static uint8_t TOD;
static uint8_t TP;
static float TakeOff_Height;
static float ThrottleB;
static float Takeoff_MinSpeed_local;
static float deltaTY;
static float deltaTX;

float ThrottleX;
float ThrottleY;


//Berechnet TrottlePoint, ThrottleLine und Seite auf der die Drohne steht (Seite ist der Rückgabewert)
bool_t calculateTakeOffConditions( void )  // TOD ist (0/0)
{

        //Set InitPos
        initialx = estimator_x;
        initialy = estimator_y;

        //Compute deltaTX and deltaTY with TOD=(0/0)
        deltaTX = initialx - (waypoints[TOD].x);
        deltaTY = initialy - (waypoints[TOD].y);

        //Find Launch line slope and Throttle line slope
        float MLaunch = deltaTY/deltaTX;


        //Compute Throttle Point
        if(deltaTX < 0)
                throttlePx = TDistance/sqrt(MLaunch*MLaunch+1);
        else
                throttlePx = - TDistance/sqrt(MLaunch*MLaunch+1);

        if(deltaTY < 0)
                throttlePy = sqrt((TDistance*TDistance)-(throttlePx*throttlePx));
        else
                throttlePy = - sqrt((TDistance*TDistance)-(throttlePx*throttlePx));


        //Find ThrottleLine
        ThrottleSlope = tan(atan2(deltaTY,deltaTX)+(3.14/2));                   //-1/MLaunch; //90°
Drehung der Kurve
        ThrottleB = (throttlePy - (ThrottleSlope*throttlePx));                  //y-Offset


        //Translate ThrottlePoint to absolut
        throttlePx= throttlePx+initialx;
        throttlePy= throttlePy+initialy;

        //Set TrottlePoint in GCS
        waypoints[TP].x= throttlePx;
        waypoints[TP].y= throttlePy;


        //Determine whether the UAV is below or above the throttle line
        if(deltaTY > ((ThrottleSlope*deltaTX)+ThrottleB))       //ist UAV über der ThrottleLine?
                return TRUE;                                    //UAV ist drüber
        else
                return FALSE;

}
```

```
bool_t InitializeZHAWBungeeTakeoff(uint8_t TODWP, uint8_t _TP)          //uint8_t _NP muss falls
navLine nicht verwendet wird noch entfernt werden!!!!
{
        TOD = TODWP;
        TP = _TP;
        TakeOff_Height = (waypoints[TOD].a);


        //** Für Airframe *********************************
        Takeoff_MinSpeed_local=3.0;   //!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!! für den Test =
TAKEOFF_MINSPEED
                                      //TAKEOFF_DISTANCE can only be positive
        TDistance = 9.0;              //fabs(TAKEOFF_DISTANCE);!!!!!!!!!!!!!!!!!!!!!!!!!!!! für
den Test
        //************************************************

        //Record bungee alt (which should be the ground alt at that point)
        BungeeAlt = (waypoints[_TP].a);


        AboveThrottleLine=calculateTakeOffConditions();                         //Auf welcher Seite
ist dir Drohne


        //Enable Launch Status and turn kill throttle on
        CTakeoffStatus = Launch;
        kill_throttle = 1; //MOTOR AUS

        return FALSE;
}


bool_t ZHAWBungeeTakeoff(void)
{
        //Translate the Current Position so that the THROTTLEPOINT is (0/0) (wie weit ist die Drohne
noch vom TP weg?)
        float Currentx = estimator_x - throttlePx;
        float Currenty = estimator_y - throttlePy;

        bool_t CurrentAboveThrottleLine;

        switch(CTakeoffStatus)
        {
        case Launch:
                //UAV on the Hook
                NavVerticalAutoThrottleMode(0.1);                               //Set the climb
control to auto-throttle with the specified pitch pre-command (navigation.h) -> No Pitch
                NavVerticalAltitudeMode(TakeOff_Height, 0.);                    //Vorgabe der Sollhöhe
                kill_throttle = 1;      //Motor ausgeschaltet


                //recalculate lines if the UAV is not in Auto2
                if(pprz_mode < 2)
                        AboveThrottleLine=calculateTakeOffConditions();


                //Find out if the UAV is currently above the line
                if(Currenty > (ThrottleSlope*Currentx) + 0)
                        CurrentAboveThrottleLine = TRUE;
                else
                        CurrentAboveThrottleLine = FALSE;


                RunOnceEvery(10, DOWNLINK_SEND_ZHAWTAKEOFF(DefaultChannel, &AboveThrottleLine,
&CurrentAboveThrottleLine));


                //Find out if UAV has crossed the line
                if(AboveThrottleLine != CurrentAboveThrottleLine && estimator_hspeed_mod >
Takeoff_MinSpeed_local)
                {
                        CTakeoffStatus = Throttle;
                        kill_throttle = 0;
```

```
                        nav_init_stage();
                        ThrottleX = estimator_x;
                        ThrottleY = estimator_y;

                        NavVerticalAutoThrottleMode(AGR_CLIMB_PITCH);
                        NavVerticalAltitudeMode(TakeOff_Height, 0.);
                        NavVerticalThrottleMode(9600*(1));
                }
                break;

        case Throttle:
                //Follow Launch Line
                NavVerticalThrottleMode(9600*(1));
                nav_route_xy(ThrottleX,ThrottleY,(waypoints[TOD].x),(waypoints[TOD].y));
                kill_throttle = 0;

                if(estimator_z > TakeOff_Height-10)
                {
                        CTakeoffStatus = Finished;
                        return FALSE;
                }
                else
                {
                        return TRUE;
                }
                break;
        default:
                break;
        }
        return TRUE;
}


//***************************** Bungee Takeoff
glide*****************************************************************


bool_t ZHAWBungeeTakeoff_glide(void)
{
        //Translate the Current Position so that the THROTTLEPOINT is (0/0) (wie weit ist die Drohne
noch vom TP weg?)
        float Currentx = estimator_x - throttlePx;
        float Currenty = estimator_y - throttlePy;

        bool_t CurrentAboveThrottleLine;

        switch(CTakeoffStatus)
        {
        case Launch:
                //UAV on the Hook
                NavVerticalAutoThrottleMode(0.1);
                NavVerticalAltitudeMode(TakeOff_Height, 0.);
                kill_throttle = 1;      //Motor ausgeschaltet

                //recalculate lines if the UAV is not in Auto2
                if(pprz_mode < 2)
                        AboveThrottleLine=calculateTakeOffConditions();


                //Find out if the UAV is currently above the line
                if(Currenty > (ThrottleSlope*Currentx) + 0)
                        CurrentAboveThrottleLine = TRUE;
                else
                        CurrentAboveThrottleLine = FALSE;


                RunOnceEvery(10, DOWNLINK_SEND_ZHAWTAKEOFF(DefaultChannel, &AboveThrottleLine,
&CurrentAboveThrottleLine));


                //Find out if UAV has crossed the line
                if(AboveThrottleLine != CurrentAboveThrottleLine && estimator_hspeed_mod >
Takeoff_MinSpeed_local)
```

```
                {
                        CTakeoffStatus = Finished;
                        kill_throttle = 0;
                        nav_init_stage();
                        return FALSE;
                }
                else
                {
                        return TRUE;
                }
                break;
        default:
                break;
        }
        return TRUE;
}
```

```
/*
 * $Id$
 *
 * Copyright (C) 2011  langede0
 *
 * This file is part of paparazzi.
 *
 * paparazzi is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2, or (at your option)
 * any later version.
 *
 * paparazzi is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with paparazzi; see the file COPYING.  If not, write to
 * the Free Software Foundation, 59 Temple Place - Suite 330,
 * Boston, MA 02111-1307, USA.
 *
 */
/** \file ZHAWNav_Takeoff2.c
 *  \brief Bungee Takeoff NavLine
 *
 */



#include "subsystems/navigation/ZHAWNav_Takeoff2.h"
#include "firmwares/rotorcraft/autopilot.h"

#include "mcu_periph/uart.h"
#include "messages.h"
#include "downlink.h"

#ifndef DOWNLINK_DEVICE
#define DOWNLINK_DEVICE DOWNLINK_AP_DEVICE
#endif


/************** ZHAW Bungee Takeoff ********************************************/

/** Takeoff functions for bungee takeoff.
Run initialize function when the plane is on the bungee, the bungee is fully extended and you are
ready to
launch the plane. After initialized, the plane will follow a line drawn by the position of the plane
on initialization and the
position of the WP_TakeOffDirection (given in the arguments). Once the plane crosses the throttle
line, which is perpendicular to the line the plane is following,
and intersects the position of the ThrottlePoint (which has a fixed distance from the Initial Position
(TakeOff_Distance in airframe file) from the bungee just in case the bungee doesn't release directly
above the bungee) the prop will come on. The plane will then continue to follow the line until it has
reached a specific
height (defined in as Takeoff_Height in airframe file) above the bungee waypoint and speed (defined as
Takeoff_Speed in the airframe file).

<section name="Takeoff" prefix="Takeoff_">
  <define name="Speed" value="15" unit="m/s"/>
  <define name="Distance" value="10" unit="m"/>
  <define name="MinSpeed" value="5" unit="m/s"/>
</section>
 */
/*
#ifndef Takeoff_Distance
#define Takeoff_Distance 10
#endif
#ifndef Takeoff_Speed
#define Takeoff_Speed 6
#endif
#ifndef Takeoff_MinSpeed
#define Takeoff_MinSpeed 2
#endif*/
```

```
enum TakeoffStatus { Launch, Stabilization, Climb, Finished };
static enum TakeoffStatus CTakeoffStatus;
static float throttlePx;
static float throttlePy;
static float navPx;
static float navPy;
static float initialx;
static float initialy;
static float ThrottleSlope;
static bool_t AboveLines;
static float BungeeAlt;
static float TDistance;
static float NDistance;
static uint8_t TOD;
static uint8_t TP;
static uint8_t NP;
static float TakeOff_Height;
static float ThrottleB;
static float NavB;
static float Takeoff_MinSpeed_local;
static float deltaTY;
static float deltaTX;

float StartThrottle_X;
float StartThrottle_Y;


//Berechnet TrottlePoint, ThrottleLine und Seite auf der die Drohne steht (Seite ist der Rückgabewert)
bool_t calculateTakeOffConditionsNavLine( void )  // TOD ist (0/0)
{

        //Set InitPos
        initialx = estimator_x;
        initialy = estimator_y;

        //Compute deltaTX and deltaTY with TOD=(0/0)
        deltaTX = initialx - (waypoints[TOD].x);
        deltaTY = initialy - (waypoints[TOD].y);


        //Find Launch line slope and Throttle line slope
        float MLaunch = deltaTY/deltaTX;


        //Compute Throttle Point and Nav Point
        if(deltaTX < 0)
        {
                throttlePx = TDistance/sqrt(MLaunch*MLaunch+1);
                navPx = NDistance/sqrt(MLaunch*MLaunch+1);
        }
        else
        {
                throttlePx = - TDistance/sqrt(MLaunch*MLaunch+1);
                navPx = - NDistance/sqrt(MLaunch*MLaunch+1);
        }

        if(deltaTY < 0)
        {
                throttlePy = sqrt((TDistance*TDistance)-(throttlePx*throttlePx));
                navPy = sqrt((NDistance*NDistance)-(navPx*navPx));
        }
        else
        {
                throttlePy = - sqrt((TDistance*TDistance)-(throttlePx*throttlePx));
                navPy = - sqrt((NDistance*NDistance)-(navPx*navPx));
        }


        //Find ThrottleLine and NavLine
        ThrottleSlope = tan(atan2(deltaTY,deltaTX)+(3.14/2));           //NavLineSlope the same like
ThrottleSlope
        ThrottleB = (throttlePy - (ThrottleSlope*throttlePx));          //ThrottleBase
        NavB = (navPy - (ThrottleSlope*navPx));                         //NavBase
```

```
        //Translate ThrottlePoint and NavPoint to absolut
        throttlePx= throttlePx+initialx;
        throttlePy= throttlePy+initialy;
        navPx= navPx+initialx;
        navPy= navPy+initialy;


        //Set TrottlePoint and NavPoint in GCS
        waypoints[TP].x= throttlePx;
        waypoints[TP].y= throttlePy;
        waypoints[NP].x= navPx;
        waypoints[NP].y= navPy;



        //Determine whether the UAV is below or above the throttle line and the NavLine
        if(deltaTY > ((ThrottleSlope*deltaTX)+ThrottleB))       //ist UAV über der ThrottleLine?
                return TRUE;                                    //UAV ist drüber
        else
                return FALSE;

}


bool_t InitializeZHAWBungeeTakeoffNavLine(uint8_t TODWP, uint8_t _TP, uint8_t _NP)
{
        TOD = TODWP;
        TP = _TP;
        NP = _NP;
        TakeOff_Height = (waypoints[TOD].a);

        Takeoff_MinSpeed_local=3.0;   //!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!! für den Test


        //Takeoff_Distance can only be positive
        TDistance = 9.0;               //fabs(Takeoff_Distance);!!!!!!!!!!!!!!!!!!!!!!!!!!!! für
den Test
        NDistance = 30.0;                                       //!!!!!!!!!!!!!!!!!!!!!!!!!!!! für
den Test

        //Record bungee alt (which should be the ground alt at that point)
        BungeeAlt = (waypoints[_TP].a);


        AboveLines=calculateTakeOffConditionsNavLine();                         //Auf welcher Seite
ist dir Drohne


        //Enable Launch Status and turn kill throttle on
        CTakeoffStatus = Launch;
        kill_throttle = 1; //MOTOR AUS

        return FALSE;
}


bool_t ZHAWBungeeTakeoffNavLine(void)
{
        //Translate the Current Position so that the THROTTLEPOINT is (0/0) (wie weit ist die Drohne
noch vom TP weg?)
        float CurrentThrottlex = estimator_x - throttlePx;
        float CurrentThrottley = estimator_y - throttlePy;

        //Translate the Current Position so that the NAVPOINT is (0/0) (wie weit ist die Drohne noch
vom NP weg?)
        float CurrentNavx = estimator_x - navPx;
        float CurrentNavy = estimator_y - navPy;

        bool_t CurrentAboveThrottleLine;
        bool_t CurrentAboveNavLine;

        switch(CTakeoffStatus)
```

```
	{
	case Launch:
		//Follow Launch Line
		NavVerticalAutoThrottleMode(0.1);                                //Set the climb
control to auto-throttle with the specified pitch pre-command (navigation.h) -> No Pitch
		NavVerticalAltitudeMode(estimator_z + 20.0, 0.);        //Vorgabe der Sollhöhe
		kill_throttle = 1;                                               //Motor ausgeschaltet


		//recalculate lines if the UAV is not in Auto2
		if(pprz_mode < 2)
			AboveLines=calculateTakeOffConditionsNavLine();



		//Find out if the UAV is currently above the Throttle line
		if(CurrentThrottley > (ThrottleSlope*CurrentThrottlex) + 0)
			CurrentAboveThrottleLine = TRUE;
		else
			CurrentAboveThrottleLine = FALSE;


		RunOnceEvery(10, DOWNLINK_SEND_ZHAWTAKEOFF(DefaultChannel, &AboveLines,
&CurrentAboveThrottleLine));


		//Find out if UAV has crossed the Throttle Line
		if(AboveLines != CurrentAboveThrottleLine && estimator_hspeed_mod >
Takeoff_MinSpeed_local)
		{
			CTakeoffStatus = Stabilization;
			kill_throttle = 0;
			nav_init_stage();
			NavVerticalAltitudeMode(estimator_z + 5.0, 0);          //Höhe halten in
Stabilisations-Zwischenschritt
			NavVerticalAutoThrottleMode(0.1);
		}
		break;

	case Stabilization:                                                      //Höhe halten, nicht
navigieren, Motor einschalten
		//NavVerticalAutoThrottleMode(0.1);
		NavVerticalThrottleMode(9600*(1));
		kill_throttle = 0;


		//Find out if the UAV is currently above the Nav line
		if(CurrentNavy > (ThrottleSlope*CurrentNavx) + 0)
			CurrentAboveNavLine = TRUE;
		else
			CurrentAboveNavLine = FALSE;



		if (AboveLines != CurrentAboveNavLine)
		{
			CTakeoffStatus = Climb;
			nav_init_stage();
			StartThrottle_X=estimator_x;
			StartThrottle_Y=estimator_y;

			NavVerticalAutoThrottleMode(AGR_CLIMB_PITCH);
			NavVerticalAltitudeMode(TakeOff_Height, 0.);
			NavVerticalThrottleMode(9600*(1));
		}

	case Climb:
		//NavVerticalAutoThrottleMode(AGR_CLIMB_PITCH);
		//NavVerticalAltitudeMode(TakeOff_Height, 0.);
		NavVerticalThrottleMode(9600*(1));
		nav_route_xy(StartThrottle_X,StartThrottle_Y,(waypoints[TOD].x),(waypoints[TOD].y));

		if(estimator_z > (TakeOff_Height-10))
		{
```

```
                        CTakeoffStatus = Finished;
                        return FALSE;
                }
                else
                {
                        return TRUE;
                }
                break;
        default:
                break;
        }
        return TRUE;
}
```

```
/*
 * $Id$
 *
 * Copyright (C) 2011  langede0
 *
 * This file is part of paparazzi.
 *
 * paparazzi is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2, or (at your option)
 * any later version.
 *
 * paparazzi is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with paparazzi; see the file COPYING.  If not, write to
 * the Free Software Foundation, 59 Temple Place - Suite 330,
 * Boston, MA 02111-1307, USA.
 *
 */
/** \file ZHAWNav_Landing.c
 *  \brief Advanced landing with ultrasonic sensor
 *
 */



#include "subsystems/navigation/ZHAWNav_Landing.h"
#include "firmwares/rotorcraft/autopilot.h"
#include "modules/sonar/sonar_adc.h"

#include "mcu_periph/uart.h"
#include "messages.h"
#include "downlink.h"
#include "subsystems/navigation/parameter_changer.h"
#include "estimator.h"

#include "mcu_periph/uart.h"
#include "messages.h"
#include "downlink.h"

#ifndef DOWNLINK_DEVICE
#define DOWNLINK_DEVICE DOWNLINK_AP_DEVICE
#endif


/************** ZHAW Landing fixed Pitch*********************************************/
/*
Landing Routine


  <section name="ZHAWLanding" prefix="Landing_">
    <define name="SaveHeight" value="3" unit="m"/>
    <define name="SonarHeight" value="6" unit="m"/>
    <define name="TDDistance" value="50" unit="s"/>
    <define name="FlareFactor" value="5"/>
  </section>

 */

#ifndef Landing_SaveHeight              //Height for Failsave
#define Landing_SaveHeight 3
#endif

#ifndef Landing_SonarHeight             //Height to Switch to Sonar
#define Landing_SonarHeight 6
#endif

#ifndef Landing_TDDistance              //Flare Disatnce
#define Landing_TDDistance 40
```

```
#endif

#ifndef Landing_FlareFactor            //Faktor mit dem die Sollhöhe beim Flare verkleinert wird
#define Landing_FlareFactor 0.5
#endif


enum LandingStatus { CircleDown, LandingWait, ApproachHeading, DeclineToSonar, Approach, Flare,
Stall };
static enum LandingStatus CLandingStatus;
static struct Point2D LandCircle;
static uint8_t AFWaypoint;
static uint8_t TDWaypoint;
static uint8_t CPWaypoint;
static uint8_t msgLandStatus;
static float LandRadius;
static float LandAppAlt;
static float LandCircleQDR;
static float ApproachQDR;
static float LandSlope;
static float DeltaFx;
static float DeltaFy;
static float CheckPx;
static float CheckPy;
static float TDDistance;
static float SonarHeight;
static bool_t AboveCheckPoint;
static bool_t CurrentAboveCheckPoint;

bool_t InitializeZHAWSkidLanding(uint8_t AFWP, uint8_t TDWP, uint8_t CPWP, float radius)
{
        AFWaypoint = AFWP;
        TDWaypoint = TDWP;
        CPWaypoint = CPWP;

        CLandingStatus = CircleDown;
        LandRadius = radius;
        LandAppAlt = estimator_z;
        TDDistance = fabs(Landing_TDDistance);          //TDDistance can only be positive
        SonarHeight = fabs(Landing_SonarHeight);        //SonarHeight can only be positive

        /*
        // für Airframe****************
        SaveHeight = 3;                 //Höhe für Failsave
        SonarHeight = 6;                //Höhe für Approach
        TDDistance = 50;                //Strecke, auf der geflaret wird
        FlareFactor = 0.5;              //Faktor mit dem die Sollhöhe beim Flare verkleinert wird
        // ********************************
        */

        set_approach_params();  // Parameter für Landung setzten (Airspeed, max_roll, ...)
        set_as_mode(3);         // Airspeed Pitch Simple

        //Translate distance from AF to TD so that AF is (0/0)
        float x_0 = waypoints[TDWaypoint].x - waypoints[AFWaypoint].x;
        float y_0 = waypoints[TDWaypoint].y - waypoints[AFWaypoint].y;

        // Unit vector from AF to TD
        float d = sqrt(x_0*x_0+y_0*y_0);        //d=Horizontale Strecke von AF zu TD
        float x_1 = x_0 / d;
        float y_1 = y_0 / d;

        //find the center of LandCircle
        LandCircle.x = waypoints[AFWaypoint].x + y_1 * LandRadius;
        LandCircle.y = waypoints[AFWaypoint].y - x_1 * LandRadius;


        //Compute the QDR-Angles
        LandCircleQDR = atan2(waypoints[AFWaypoint].x-LandCircle.x, waypoints[AFWaypoint].y-
LandCircle.y);

        if(LandRadius > 0)
        {
```

```
                ApproachQDR = LandCircleQDR-RadOfDeg(90);
                LandCircleQDR = LandCircleQDR-RadOfDeg(45);
        }
        else
        {
                ApproachQDR = LandCircleQDR+RadOfDeg(90);
                LandCircleQDR = LandCircleQDR+RadOfDeg(45);
        }


        return FALSE;
}

bool_t ZHAWSkidLanding(void)
{
        switch(CLandingStatus)
        {
        case CircleDown: // Kreisen bis die Höhe, die im AFWaypoint vorgegeben ist erreicht ist (um
den Wegpunkt der in InitializeSkidLanding berechnet wurde)
                if(NavCircleCount() < .1)
                {
                        NavVerticalAltitudeMode(LandAppAlt, 0);
                }
                else
                        NavVerticalAltitudeMode(waypoints[AFWaypoint].a, 0);


                nav_circle_XY(LandCircle.x, LandCircle.y, LandRadius);

                if(estimator_z < waypoints[AFWaypoint].a + 5) //Drohne hat die Höhe des AF_Waypoints
erreicht.
                {
                        CLandingStatus = LandingWait;
                        nav_init_stage();
                }
        msgLandStatus=1;
        break;

        case LandingWait: // Höhe halten und weiter um CircleCircle kreisen
                NavVerticalAltitudeMode(waypoints[AFWaypoint].a, 0);
                nav_circle_XY(LandCircle.x, LandCircle.y, LandRadius);

                if(NavCircleCount() > 0.5 && NavQdrCloseTo(DegOfRad(ApproachQDR)))  // Drohne nähert
sich dem Winkel (bzw. Kurs) auf dem Sie fliegen muss um zu landen (45° fehlen)
                {
                        CLandingStatus = ApproachHeading;
                        nav_init_stage();
                }
        msgLandStatus=2;
        break;


        case ApproachHeading:                           //Drohne fliegt den Kreis fertig, bis sie auf
Landekurs ist
                NavVerticalAltitudeMode(waypoints[AFWaypoint].a, 0);    //Sollhöhe geben
                nav_circle_XY(LandCircle.x, LandCircle.y, LandRadius);

                if(NavQdrCloseTo(DegOfRad(LandCircleQDR)))  //Drohne ist auf Landekurs und auf Höhe AF
                {
                        CLandingStatus = DeclineToSonar;
                        nav_init_stage();
                        set_land_params();
                }
        msgLandStatus=3;
        break;


        case DeclineToSonar: // Sinken, bis Sonar sich meldet
                NavVerticalAltitudeMode(waypoints[TDWaypoint].a+Landing_SonarHeight, 0);
                nav_route_xy(waypoints[AFWaypoint].x,waypoints[AFWaypoint].y,waypoints
[TDWaypoint].x,waypoints[TDWaypoint].y);

                if(sonar_dist < (Landing_SonarHeight + 0.5))
                {
```

```
                        CLandingStatus = Approach;
                        estimator_z_mode = SONAR_HEIGHT;
                        nav_init_stage();
                        AboveCheckPoint = CalculateCheckPoint();
                }
        msgLandStatus=4;
        break;

        case Approach: //Sonar Höhe (ca. 6m) halten, bis zum FlarePoint, wo die Landung begonnen
werden kann.
                NavVerticalAltitudeMode(Landing_SonarHeight, 0); //Sonarhöhe ÜBER BODEN!!!!
                nav_route_xy(waypoints[AFWaypoint].x,waypoints[AFWaypoint].y,waypoints
[TDWaypoint].x,waypoints[TDWaypoint].y);

                //find ouf if the UAV has crossed the CheckPoint first time
                if (UAVcrossedCheckPoint() == 1)
                {
                        CLandingStatus = Flare;
                        nav_init_stage();
//                      SonarHeight = SonarHeight * Landing_FlareFactor;
                        kill_throttle = 1;
                        set_fixed_pitch_pitch(0.2);
                }
        msgLandStatus=5;
        break;

        case Flare:
//              NavVerticalAltitudeMode(Landing_SonarHeight, 0);
                nav_route_xy(waypoints[AFWaypoint].x,waypoints[AFWaypoint].y,waypoints
[TDWaypoint].x,waypoints[TDWaypoint].y);


                if (estimator_z < 0.6)
                {
                        float flare_pitch = 1/estimator_z*Landing_FLARE_FACTOR;
                        set_fixed_pitch_pitch(flare_pitch);
                }

        msgLandStatus=6;
        break;

        case Stall:
                kill_throttle = 1;
                NavVerticalAltitudeMode(Landing_SonarHeight, 0);
                nav_route_xy(waypoints[AFWaypoint].x,waypoints[AFWaypoint].y,waypoints
[TDWaypoint].x,waypoints[TDWaypoint].y);

        msgLandStatus=7;
        break;

        default:

        break;
        }

        RunOnceEvery(5, DOWNLINK_SEND_ZHAWLAND(DefaultChannel, &msgLandStatus, &estimator_z_mode,
&AboveCheckPoint, &CurrentAboveCheckPoint, &SonarHeight, &estimator_z_sonar, &estimator_z));

        //Failsave Height
        if ((estimator_z < Landing_SaveHeight) && (CLandingStatus != Flare) && (CLandingStatus !=
Stall))
        {
                estimator_z_mode=GPS_HEIGHT;
                set_max_pitch(99);
                set_min_pitch(99);
                set_max_roll(99);
                return FALSE;
        }

        //Failsave CheckPoint
        if ((CLandingStatus == DeclineToSonar) && ( UAVcrossedCheckPoint() == 1 ))
        {
                estimator_z_mode=GPS_HEIGHT;
                set_max_pitch(99);
```

```
                set_min_pitch(99);
                set_max_roll(99);
                return FALSE;
        }

        return TRUE;
}

bool_t CalculateCheckPoint(void) //TD is (0/0)
{

        //Compute DeltaFx and DeltaFy between AF an TD with TD=(0/0)
        float deltaCX = (waypoints[AFWaypoint].x) - (waypoints[TDWaypoint].x);
        float deltaCY = (waypoints[AFWaypoint].y) - (waypoints[TDWaypoint].y);

        //Find Land line slope and Throttle line slope
        float MLaunch = deltaCY/deltaCX;

        //Compute Flare Point
        if(DeltaFx < 0)
                CheckPx = TDDistance/sqrt(MLaunch*MLaunch+1);
        else
                CheckPx = - TDDistance/sqrt(MLaunch*MLaunch+1);

        if(DeltaFy < 0)
                CheckPy = sqrt((TDDistance*TDDistance)-(CheckPx*CheckPx));
        else
                CheckPy = - sqrt((TDDistance*TDDistance)-(CheckPx*CheckPx));

        //Find TouchDownLine
        LandSlope = tan(atan2(DeltaFy,DeltaFx)+(3.14/2));
        float TouchDownB = (CheckPy - (LandSlope*CheckPx));

        //Translate CheckPoint to absolut
        CheckPx= CheckPx + (waypoints[TDWaypoint].x);
        CheckPy= CheckPy + (waypoints[TDWaypoint].y);

        //Set CheckPoint in GCS
        waypoints[CPWaypoint].x= CheckPx;
        waypoints[CPWaypoint].y= CheckPy;

        //Determine whether the UAV is below or above the CheckPoint
        if(DeltaFy > ((LandSlope*deltaCX)+TouchDownB))
                return TRUE;
        else
                return FALSE;

}

bool_t UAVcrossedCheckPoint (void)
{
        //Translate the Current Position so that the FLAREPOINT is (0/0) (wie weit ist die Drohne noch
vom FP weg?)
        float Currentx = estimator_x - CheckPx;
        float Currenty = estimator_y - CheckPy;


        //Find out if the UAV is currently above the line
        if(Currenty > (LandSlope*Currentx) + 0)
                CurrentAboveCheckPoint = TRUE;
        else
                CurrentAboveCheckPoint = FALSE;

        if(AboveCheckPoint != CurrentAboveCheckPoint)
        {
                return TRUE;
        }

        return FALSE;

}
```

```
/*
 * Copyright (C) 2011 Bruzzlee
 *
 * This file is part of paparazzi.
 * sonar
 * paparazzi is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2, or (at your option)
 * any later version.
 *
 * paparazzi is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with paparazzi; see the file COPYING.  If not, write to
 * the Free Software Foundation, 59 Temple Place - Suite 330,
 * Boston, MA 02111-1307, USA.
 *
 */

#include "modules/sonar/sonar_adc.h"
#include "mcu_periph/adc.h"
#include BOARD_CONFIG
#include "generated/airframe.h"
#include "estimator.h"
//Messages
#include "mcu_periph/uart.h"
#include "messages.h"
#include "downlink.h"

uint16_t adc_sonar_val;
float sonar_offset;
float sonar_dist;
float sonar_raw;
float sonar_scale;
float sonar_filter;
float sonar_old;

// Local variables
//uint16_t sonar_adc_offset;
//bool_t sonar_adc_offset_init;
//uint32_t sonar_adc_offset_tmp;
//uint16_t sonar_adc_cnt;


//Downlink
#ifndef DOWNLINK_DEVICE
#define DOWNLINK_DEVICE DOWNLINK_AP_DEVICE
#endif

#ifndef SITL // Use ADC if not in simulation

#ifndef ADC_CHANNEL_SONAR
#error "ADC_CHANNEL_SONAR needs to be defined to use sonar_adc module"
#endif

#ifndef ADC_CHANNEL_SONAR_NB_SAMPLES
#define ADC_CHANNEL_SONAR_NB_SAMPLES DEFAULT_AV_NB_SAMPLE
#endif

#ifndef SONAR_ADC_OFFSET
#define SONAR_ADC_OFFSET 0
#endif
#define SONAR_ADC_OFFSET_NBSAMPLES_INIT 40
#define SONAR_ADC_OFFSET_NBSAMPLES_AVRG 60
#define SONAR_ADC_NBSAMPLES_AVRG 10


struct adc_buf buf_sonar;

#endif
```

```
void sonar_adc_init( void ) {
#ifndef SITL
  adc_buf_channel(ADC_CHANNEL_SONAR, &buf_sonar, ADC_CHANNEL_SONAR_NB_SAMPLES);
#endif
        sonar_offset = SONAR_ADC_OFFSET;
        sonar_scale = SONAR_ADC_SCALE;
        sonar_filter = SONAR_ADC_FILTER;
}

void sonar_adc_update( void ) {
#ifndef SITL
        adc_sonar_val = buf_sonar.sum / buf_sonar.av_nb_sample;
        sonar_raw = sonar_scale * adc_sonar_val;
//      sonar_dist = sonar_raw - sonar_offset;

        //      Lowpass filter
        sonar_dist = sonar_filter * sonar_old + (1 - sonar_filter) * (sonar_raw - sonar_offset);
        sonar_old = sonar_dist;

//DOWNLINK_SEND_SONAR_ADC(DefaultChannel, &adc_sonar_val, &sonar_raw, &sonar_dist)

#ifdef USE_SONAR
        EstimatorSetAltSonar(sonar_dist);
#endif

#ifdef SONAR_SYNC_SEND
        DOWNLINK_SEND_SONAR_ADC(DefaultChannel, &adc_sonar_val, &sonar_raw, &sonar_dist);
#else
        RunOnceEvery(10, DOWNLINK_SEND_SONAR_ADC(DefaultChannel, &adc_sonar_val, &sonar_raw,
&sonar_dist));
#endif
#else // SITL


#endif //SITL

}
```

```
/*
 * Copyright (C) 2010 ZHAW
 *
 * Autor: Bruzzlee
 * Angle of Attack ADC Sensor
 * US DIGITAL MA3-A10-236-N
 *
 * This file is part of paparazzi.
 *
 * paparazzi is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2, or (at your option)
 * any later version.
 *
 * paparazzi is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with paparazzi; see the file COPYING.  If not, write to
 * the Free Software Foundation, 59 Temple Place - Suite 330,
 * Boston, MA 02111-1307, USA.
 *
 */

#include "modules/sensors/AOA_adc.h"
#include "mcu_periph/adc.h"
#include BOARD_CONFIG
#include "generated/airframe.h"
#include "estimator.h"
#include "std.h"
//Messages
#include "mcu_periph/uart.h"
#include "messages.h"
#include "downlink.h"

uint16_t adc_AOA_val;

//Downlink
#ifndef DOWNLINK_DEVICE
#define DOWNLINK_DEVICE DOWNLINK_AP_DEVICE
#endif

#ifndef SITL // Use ADC if not in simulation

#ifndef ADC_CHANNEL_AOA
#error "ADC_CHANNEL_AOA needs to be defined to use AOA_adc module"
#endif

#ifndef ADC_CHANNEL_AOA_NB_SAMPLES
#define ADC_CHANNEL_AOA_NB_SAMPLES DEFAULT_AV_NB_SAMPLE
#endif

struct adc_buf buf_AOA;
float AOA_offset, AOA_filter;
float AOA, AOA_old;
#endif


void AOA_adc_init( void ) {
        AOA_offset = AOA_OFFSET;
        AOA_filter = AOA_FILTER;
        AOA_old = 0;
#ifndef SITL
        adc_buf_channel(ADC_CHANNEL_AOA, &buf_AOA, ADC_CHANNEL_AOA_NB_SAMPLES);
#endif
}

void AOA_adc_update( void ) {
#ifndef SITL
        adc_AOA_val = buf_AOA.sum / buf_AOA.av_nb_sample;

//      PT1 filter and convert to rad
```

```
        AOA = AOA_filter * AOA_old + (1 - AOA_filter) * (adc_AOA_val*(2*M_PI)/1024-M_PI+AOA_offset);
        AOA_old = AOA;
#endif
        RunOnceEvery(30, DOWNLINK_SEND_AOA_adc(DefaultChannel, &adc_AOA_val, &AOA));
        EstimatorSetAOA(AOA);
}
```

```
/*
 * $Id$
 *
 * Copyright (C) 2011  Bruzzlee
 *
 * This file is part of paparazzi.
 *
 * paparazzi is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2, or (at your option)
 * any later version.
 *
 * paparazzi is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with paparazzi; see the file COPYING.  If not, write to
 * the Free Software Foundation, 59 Temple Place - Suite 330,
 * Boston, MA 02111-1307, USA.
 *
 */
/** \file paramter_changer.c
 *  \brief Change parameter min/max of pitch and roll and airspeed-mode
 *
 */

#include "subsystems/navigation/parameter_changer.h"
#include "firmwares/fixedwing/stabilization/stabilization_attitude.h"
#include "firmwares/fixedwing/guidance/guidance_v.h"
#include "estimator.h"

#include "mcu_periph/uart.h"
#include "messages.h"
#include "downlink.h"

#ifndef DOWNLINK_DEVICE
#define DOWNLINK_DEVICE DOWNLINK_AP_DEVICE
#endif

bool_t set_as_mode(uint8_t as_mode_set)
{
        //AS_MODE_STANDARD              0
        //AS_MODE_VASSILLIS             1
        //AS_MODE_ASP_CLIMBRATE         2
        //AS_MODE_ASP_SIMPLE            3
        //AS_MODE_ASP_MANUAL            4
        //AS_MODE_ASP_ACCEL             5
        //AS_MODE_FIX_PITCH             6
        v_ctl_airspeed_mode = as_mode_set;

        return FALSE;
}

bool_t set_max_roll(float max_roll)
{
        if(max_roll<10.0){
                h_ctl_roll_max_setpoint = max_roll;
        }
        else
                h_ctl_roll_max_setpoint = H_CTL_ROLL_MAX_SETPOINT;
        send_params();

        return FALSE;
}

bool_t set_max_pitch(float max_pitch)
{
        if(max_pitch<10.0){
                h_ctl_pitch_max_setpoint = max_pitch;
        }
        else
                h_ctl_pitch_max_setpoint = H_CTL_PITCH_MAX_SETPOINT;
```

```
        send_params();

        return FALSE;
}

bool_t set_min_pitch(float min_pitch)
{
        if(min_pitch<10.0){
                h_ctl_pitch_min_setpoint = min_pitch;
        }
        else
                h_ctl_pitch_min_setpoint = H_CTL_PITCH_MIN_SETPOINT;
        send_params();

        return FALSE;
}

bool_t set_approach_params()
{
        estimator_z_mode=GPS_HEIGHT;
        v_ctl_airspeed_mode = AS_MODE_ASP_SIMPLE;
        h_ctl_pitch_mode = H_CTL_PITCH_MODE_THETA;
//              set_max_roll(99.0);
//              set_max_pitch(99.0);
//              set_min_pitch(99.0);
        send_params();

        return FALSE;
}

bool_t set_measure_params()
{
        estimator_z_mode=GPS_HEIGHT;
        v_ctl_airspeed_mode = AS_MODE_ASP_SIMPLE;
        h_ctl_pitch_mode = H_CTL_PITCH_MODE_THETA;
//      set_max_roll(NAV_MEASURE_MAX_ROLL);
//      set_max_pitch(NAV_MEASURE_MAX_PITCH);
//      set_min_pitch(NAV_MEASURE_MIN_PITCH);
        send_params();

        return FALSE;
}

bool_t set_start_params()
{
        estimator_z_mode=GPS_HEIGHT;
        v_ctl_airspeed_mode = AS_MODE_FIX_PITCH;
        h_ctl_pitch_mode = H_CTL_PITCH_MODE_AOA;
//      set_max_roll(NAV_START_MAX_ROLL);
//      set_max_pitch(NAV_START_MAX_PITCH);
//      set_min_pitch(NAV_START_MIN_PITCH);
        send_params();

        return FALSE;
}

bool_t set_land_params()
{
        v_ctl_airspeed_mode = AS_MODE_FIX_PITCH;
//      h_ctl_pitch_mode = H_CTL_PITCH_MODE_AOA;
        h_ctl_pitch_mode = H_CTL_PITCH_MODE_THETA;
        set_max_roll(NAV_LAND_MAX_ROLL);
        //set_max_pitch(NAV_LAND_MAX_PITCH);
        //set_min_pitch(NAV_LAND_MIN_PITCH);
        send_params();

        return FALSE;
}

bool_t set_fixed_pitch_pitch(float fixedpitch)
{
        v_ctl_auto_airspeed_pitchsetp_fp=fixedpitch;

        return FALSE;
```

```
}

void send_params()
{
        DOWNLINK_SEND_ZHAWPARAMS(DefaultChannel, &h_ctl_roll_max_setpoint, &h_ctl_pitch_max_setpoint,
&h_ctl_pitch_min_setpoint, &v_ctl_airspeed_mode, &h_ctl_pitch_mode, &v_ctl_auto_airspeed_pitchsetp_fp);
}
```

```
/*
 * Driver for a Amsys Differential Presure Sensor I2C
 * AMS 5812-0003-D
 *
 * Copyright (C) 2010 The Paparazzi Team
 *
 * This file is part of paparazzi.
 *
 * paparazzi is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2, or (at your option)
 * any later version.
 *
 * paparazzi is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with paparazzi; see the file COPYING.  If not, write to
 * the Free Software Foundation, 59 Temple Place - Suite 330,
 * Boston, MA 02111-1307, USA.
 */

#include "sensors/airspeed_amsys.h"
#include "estimator.h"
#include "mcu_periph/i2c.h"
#include "mcu_periph/uart.h"
#include "messages.h"
#include "downlink.h"
#include <math.h>
//#include <stdlib.h>

#ifndef USE_AIRSPEED
// Just a Warning --> We do't use it.
//#ifndef SENSOR_SYNC_SEND
//#warning either set USE_AIRSPEED or SENSOR_SYNC_SEND to use amsys_airspeed
//#endif
#endif

#define AIRSPEED_AMSYS_ADDR 0xF4 // original F0
#ifndef AIRSPEED_AMSYS_SCALE
#define AIRSPEED_AMSYS_SCALE 1
#endif
#ifndef AIRSPEED_AMSYS_OFFSET
#define AIRSPEED_AMSYS_OFFSET 0
#endif
#define AIRSPEED_AMSYS_OFFSET_MAX 29491
#define AIRSPEED_AMSYS_OFFSET_MIN 3277
#define AIRSPEED_AMSYS_OFFSET_NBSAMPLES_INIT 40
#define AIRSPEED_AMSYS_OFFSET_NBSAMPLES_AVRG 60
#define AIRSPEED_AMSYS_NBSAMPLES_AVRG 10
#ifndef AIRSPEED_AMSYS_MAXPRESURE
#define AIRSPEED_AMSYS_MAXPRESURE 2068//2073 //Pascal
#endif
#ifndef AIRSPEED_AMSYS_I2C_DEV
#define AIRSPEED_AMSYS_I2C_DEV i2c0
#endif
#ifdef MEASURE_AMSYS_TEMPERATURE
#define TEMPERATURE_AMSYS_OFFSET_MAX 29491
#define TEMPERATURE_AMSYS_OFFSET_MIN 3277
#define TEMPERATURE_AMSYS_MAX 110
#define TEMPERATURE_AMSYS_MIN -25
#endif

#ifndef DOWNLINK_DEVICE
#define DOWNLINK_DEVICE DOWNLINK_AP_DEVICE
#endif

// Global variables
uint16_t airspeed_amsys_raw;
uint16_t tempAS_amsys_raw;
bool_t airspeed_amsys_valid;
float airspeed_tmp;
```

```
float pressure_amsys; //Pascal
float airspeed_amsys; //mps
float airspeed_scale;
float airspeed_filter;
struct i2c_transaction airspeed_amsys_i2c_trans;

// Local variables
volatile bool_t airspeed_amsys_i2c_done;
float airspeed_temperature = 0.0;
float airspeed_old = 0.0;


void airspeed_amsys_init( void ) {
        airspeed_amsys_raw = 0;
        airspeed_amsys = 0.0;
        pressure_amsys = 0.0;
        airspeed_amsys_i2c_done = TRUE;
        airspeed_amsys_valid = TRUE;
        airspeed_scale = AIRSPEED_SCALE;
        airspeed_filter = AIRSPEED_FILTER;
        airspeed_amsys_i2c_trans.status = I2CTransDone;
}

void airspeed_amsys_read_periodic( void ) {
#ifndef SITL
        if (airspeed_amsys_i2c_trans.status == I2CTransDone)
#ifndef MEASURE_AMSYS_TEMPERATURE
                I2CReceive(AIRSPEED_AMSYS_I2C_DEV, airspeed_amsys_i2c_trans, AIRSPEED_AMSYS_ADDR, 2);
#else
                I2CReceive(AIRSPEED_AMSYS_I2C_DEV, airspeed_amsys_i2c_trans, AIRSPEED_AMSYS_ADDR, 4);
#endif

#else // SITL
                extern float sim_air_speed;
                EstimatorSetAirspeed(sim_air_speed);
#endif //SITL
}

void airspeed_amsys_read_event( void ) {

        // Get raw airspeed from buffer
        airspeed_amsys_raw = 0;
        airspeed_amsys_raw = (airspeed_amsys_i2c_trans.buf[0]<<8) | airspeed_amsys_i2c_trans.buf[1];
#ifdef MEASURE_AMSYS_TEMPERATURE
        tempAS_amsys_raw = (airspeed_amsys_i2c_trans.buf[2]<<8) | airspeed_amsys_i2c_trans.buf[3];
        airspeed_temperature = (float)((float)(tempAS_amsys_raw-TEMPERATURE_AMSYS_OFFSET_MIN)/((float)
(TEMPERATURE_AMSYS_OFFSET_MAX-TEMPERATURE_AMSYS_OFFSET_MIN)/TEMPERATURE_AMSYS_MAX)
+TEMPERATURE_AMSYS_MIN);// Tmin=-25, Tmax=85
#endif

        // Check if this is valid airspeed
        if (airspeed_amsys_raw == 0)
                airspeed_amsys_valid = FALSE;
        else
                airspeed_amsys_valid = TRUE;

        // Continue only if a new airspeed value was received
        if (airspeed_amsys_valid) {

                // raw not under offest min
                if (airspeed_amsys_raw<AIRSPEED_AMSYS_OFFSET_MIN)
                        airspeed_amsys_raw = AIRSPEED_AMSYS_OFFSET_MIN;
                // raw not over offest max
                if (airspeed_amsys_raw>AIRSPEED_AMSYS_OFFSET_MAX)
                        airspeed_amsys_raw = AIRSPEED_AMSYS_OFFSET_MAX;

                // calculate raw to pressure
                pressure_amsys = (float)(airspeed_amsys_raw-AIRSPEED_AMSYS_OFFSET_MIN)
*AIRSPEED_AMSYS_MAXPRESURE/(float)(AIRSPEED_AMSYS_OFFSET_MAX-AIRSPEED_AMSYS_OFFSET_MIN);

                airspeed_tmp = sqrtf(2*(pressure_amsys)*airspeed_scale/1.2041); //without offset

        //      Lowpass filter
                airspeed_amsys = airspeed_filter * airspeed_old + (1 - airspeed_filter) * airspeed_tmp;
```

```
                airspeed_old = airspeed_amsys;

#ifdef USE_AIRSPEED
                EstimatorSetAirspeed(airspeed_amsys);
#endif
#ifdef SENSOR_SYNC_SEND
                DOWNLINK_SEND_AMSYS_AIRSPEED(DefaultChannel, &airspeed_amsys_raw, &pressure_amsys,
&airspeed_tmp, &airspeed_amsys, &airspeed_temperature);
#else
                RunOnceEvery(10, DOWNLINK_SEND_AMSYS_AIRSPEED(DefaultChannel, &airspeed_amsys_raw,
&pressure_amsys, &airspeed_tmp, &airspeed_amsys, &airspeed_temperature));
#endif
        }

        // Transaction has been read
        airspeed_amsys_i2c_trans.status = I2CTransDone;
}
```

```
/*
 * Driver for a Amsys Barometric Sensor I2C
 * AMS 5812-0150-A
 *
 * Copyright (C) 2010 The Paparazzi Team
 *
 * This file is part of paparazzi.
 *
 * paparazzi is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2, or (at your option)
 * any later version.
 *
 * paparazzi is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with paparazzi; see the file COPYING.  If not, write to
 * the Free Software Foundation, 59 Temple Place - Suite 330,
 * Boston, MA 02111-1307, USA.
 */

#include "sensors/baro_amsys.h"
#include "mcu_periph/i2c.h"
#include "estimator.h"
#include <math.h>
#include "generated/flight_plan.h" // for ground alt

//Messages
#include "mcu_periph/uart.h"
#include "messages.h"
#include "downlink.h"
//#include "gps.h"
#ifndef DOWNLINK_DEVICE
#define DOWNLINK_DEVICE DOWNLINK_AP_DEVICE
#endif

#ifdef SITL
#include "gps.h"
#endif

#define BARO_AMSYS_ADDR 0xF2
#define BARO_AMSYS_REG 0x07
#define BARO_AMSYS_SCALE 0.32
#define BARO_AMSYS_MAX_PRESSURE 103400 // Pascal
#define BARO_AMSYS_OFFSET_MAX 29491
#define BARO_AMSYS_OFFSET_MIN 3277
#define BARO_AMSYS_OFFSET_NBSAMPLES_INIT 40
#define BARO_AMSYS_OFFSET_NBSAMPLES_AVRG 60

#ifdef MEASURE_AMSYS_TEMPERATURE
#define TEMPERATURE_AMSYS_OFFSET_MAX 29491
#define TEMPERATURE_AMSYS_OFFSET_MIN 3277
#define TEMPERATURE_AMSYS_MAX 110
#define TEMPERATURE_AMSYS_MIN -25
#endif

//#define BARO_AMSYS_R 0.5
//#define BARO_AMSYS_SIGMA2 0.1
//#define BARO_ALT_CORRECTION 0.0
#ifndef BARO_AMSYS_I2C_DEV
#define BARO_AMSYS_I2C_DEV i2c0
#endif

// Global variables
uint16_t pBaroRaw;
uint16_t tBaroRaw;
float baro_amsys_offset;
bool_t baro_amsys_valid;
float baro_amsys_altitude;
float baro_amsys_temp;
float baro_amsys_p;
```

```
float baro_amsys_offset_altitude;
float baro_amsys_abs_altitude;
float ref_alt_init; //Altitude by initialising
float baro_filter;
float baro_old;

//float baro_amsys_r;
//float baro_amsys_sigma2;


struct i2c_transaction baro_amsys_i2c_trans;

// Local variables
bool_t baro_amsys_offset_init;
double baro_amsys_offset_tmp;
uint16_t baro_amsys_cnt;

void baro_amsys_init( void ) {
        baro_filter=BARO_FILTER;
        pBaroRaw = 0;
        tBaroRaw = 0;
        baro_amsys_altitude = 0.0;
        baro_amsys_p=0.0;
        baro_amsys_offset = 0;
        baro_amsys_offset_tmp = 0;
        baro_amsys_valid = TRUE;
        baro_amsys_offset_init = FALSE;
//      baro_amsys_enabled = TRUE;
        baro_amsys_cnt = BARO_AMSYS_OFFSET_NBSAMPLES_INIT + BARO_AMSYS_OFFSET_NBSAMPLES_AVRG;
//      baro_amsys_r = BARO_AMSYS_R;
//      baro_amsys_sigma2 = BARO_AMSYS_SIGMA2;
//      baro_head=0;
        ref_alt_init = 0;
        baro_amsys_i2c_trans.status = I2CTransDone;
}

void baro_amsys_read_periodic( void ) {
  // Initiate next read
#ifndef SITL
        if (baro_amsys_i2c_trans.status == I2CTransDone){
#ifndef MEASURE_AMSYS_TEMPERATURE
        I2CReceive(BARO_AMSYS_I2C_DEV, baro_amsys_i2c_trans, BARO_AMSYS_ADDR, 2);
#else
        I2CReceive(BARO_AMSYS_I2C_DEV, baro_amsys_i2c_trans, BARO_AMSYS_ADDR, 4);
#endif

                }
#else // SITL
        pBaroRaw = 0;
        baro_amsys_altitude = gps_alt / 100.0;
        baro_amsys_valid = TRUE;
        EstimatorSetAlt(baro_amsys_altitude);
#endif
}

void baro_amsys_read_event( void ) {
        pBaroRaw = 0;
        // Get raw altimeter from buffer
        pBaroRaw = (baro_amsys_i2c_trans.buf[0] << 8) | baro_amsys_i2c_trans.buf[1];
#ifdef MEASURE_AMSYS_TEMPERATURE
        tBaroRaw = (baro_amsys_i2c_trans.buf[2] << 8) | baro_amsys_i2c_trans.buf[3];
        baro_amsys_temp = (float)(tBaroRaw-TEMPERATURE_AMSYS_OFFSET_MIN)*TEMPERATURE_AMSYS_MAX/(float)
(TEMPERATURE_AMSYS_OFFSET_MAX-TEMPERATURE_AMSYS_OFFSET_MIN)+(float)TEMPERATURE_AMSYS_MIN;
#endif
        // Check if this is valid altimeter
        if (pBaroRaw == 0)
                baro_amsys_valid = FALSE;
        else
                baro_amsys_valid = TRUE;


        // Continue only if a new altimeter value was received
        //if (baro_amsys_valid && GpsFixValid()) {
        if (baro_amsys_valid) {
```

```
                //Cut RAW Min and Max
                if (pBaroRaw < BARO_AMSYS_OFFSET_MIN)
                        pBaroRaw = BARO_AMSYS_OFFSET_MIN;
                if (pBaroRaw > BARO_AMSYS_OFFSET_MAX)
                        pBaroRaw = BARO_AMSYS_OFFSET_MAX;

                //Convert to pressure
                baro_amsys_p = (float)(pBaroRaw-BARO_AMSYS_OFFSET_MIN)*BARO_AMSYS_MAX_PRESSURE/(float)
(BARO_AMSYS_OFFSET_MAX-BARO_AMSYS_OFFSET_MIN);
                if (!baro_amsys_offset_init) {
                        --baro_amsys_cnt;
                        // Check if averaging completed
                        if (baro_amsys_cnt == 0) {
                                // Calculate average
                                baro_amsys_offset = (float)(baro_amsys_offset_tmp /
BARO_AMSYS_OFFSET_NBSAMPLES_AVRG);
                                ref_alt_init = GROUND_ALT;
                                baro_amsys_offset_init = TRUE;

                                // hight over Sea level at init point
                                //baro_amsys_offset_altitude = 288.15 / 0.0065 * (1 - pow
((baro_amsys_p)/1013.25 , 1/5.255));
                        }
                        // Check if averaging needs to continue
                        else if (baro_amsys_cnt <= BARO_AMSYS_OFFSET_NBSAMPLES_AVRG)
                                baro_amsys_offset_tmp += baro_amsys_p;

                        baro_amsys_altitude = 0.0;

                }
                else {
                        // Lowpassfiltering and convert pressure to altitude
                        baro_amsys_altitude = baro_filter * baro_old + (1 - baro_filter) *
(baro_amsys_offset-baro_amsys_p)/(1.2041*9.81);
                        baro_old = baro_amsys_altitude;


                        //New value available
                        //EstimatorSetAlt(baro_amsys_abs_altitude);
                }
                baro_amsys_abs_altitude=baro_amsys_altitude+ref_alt_init;
        } /*else {
                baro_amsys_abs_altitude = 0.0;
        }*/


        // Transaction has been read
        baro_amsys_i2c_trans.status = I2CTransDone;
#ifdef SENSOR_SYNC_SEND
        DOWNLINK_SEND_AMSYS_BARO(DefaultChannel, &pBaroRaw, &baro_amsys_p, &baro_amsys_offset,
&ref_alt_init, &baro_amsys_abs_altitude, &baro_amsys_altitude, &baro_amsys_temp)
#else
        RunOnceEvery(10, DOWNLINK_SEND_AMSYS_BARO(DefaultChannel, &pBaroRaw, &baro_amsys_p,
&baro_amsys_offset, &ref_alt_init, &baro_amsys_abs_altitude, &baro_amsys_altitude, &baro_amsys_temp));
#endif

}
```

```
// #include BOARD_CONFIG
// #include <stdbool.h>
// #include "firmwares/fixedwing/main_fbw.h"
//#include "sys_time.h"
//#include "sys_time_hw.h"

//bruzzlee


#include "firmwares/fixedwing/guidance/guidance_v.h"
#include "estimator.h"
#include "messages.h"
#include "downlink.h"
#include "mcu_periph/uart.h"
#include "generated/airframe.h"
#include "subsystems/nav.h"
// #include "math/pprz_algebra_int.h"
// #include "math/pprz_algebra_float.h"

// For Downlink
#ifndef DOWNLINK_DEVICE
#define DOWNLINK_DEVICE DOWNLINK_AP_DEVICE
#endif



float SquareSumErr_airspeed;
float SquareSumErr_altitude;
float SquareSumErr_position;
float ToleranceAispeed;
float ToleranceAltitude;
float TolerancePosition;
bool_t benchm_reset;
bool_t benchm_go;


//uint8_t numOfCount;



void flight_benchmark_init( void ) {
        SquareSumErr_airspeed = 0;
        SquareSumErr_altitude = 0;
        SquareSumErr_position = 0;
        ToleranceAispeed = BENCHMARK_TOLERANCE_AIRSPEED;
        ToleranceAltitude = BENCHMARK_TOLERANCE_ALTITUDE;
        TolerancePosition = BENCHMARK_TOLERANCE_POSITION;
        benchm_reset = 0;
        benchm_go = 0;
}

void flight_benchmark_periodic( void ) {
        float Err_airspeed = 0;
        float Err_altitude = 0;
        float Err_position = 0;

        if (benchm_reset){
                flight_benchmark_reset();
                benchm_reset = 0;
        }

        if (benchm_go){
        #ifdef BENCHMARK_AIRSPEED
                Err_airspeed = fabs(estimator_airspeed - v_ctl_auto_airspeed_setpoint);
                if (Err_airspeed>ToleranceAispeed){
                        Err_airspeed = Err_airspeed-ToleranceAispeed;
                        SquareSumErr_airspeed += (Err_airspeed * Err_airspeed);
                }

        #endif
```

```
        #ifdef BENCHMARK_ALTITUDE
                Err_altitude = fabs(estimator_z - v_ctl_altitude_setpoint);
                if (Err_altitude>ToleranceAltitude){
                        Err_altitude = Err_altitude-ToleranceAltitude;
                        SquareSumErr_altitude += (Err_altitude * Err_altitude);
                }
        #endif

        #ifdef BENCHMARK_POSITION
        //      err_temp = waypoints[target].x - estimator_x;
                float deltaPlaneX = 0;
                float deltaPlaneY = 0;
                float Err_position_segment = 0;
                float Err_position_circle = 0;

//              if (nav_in_segment){
                        float deltaX = nav_segment_x_2 - nav_segment_x_1;
                        float deltaY = nav_segment_y_2 - nav_segment_y_1;
                        float anglePath = atan2(deltaX,deltaY);
                        deltaPlaneX = nav_segment_x_2 - estimator_x;
                        deltaPlaneY = nav_segment_y_2 - estimator_y;
                        float anglePlane = atan2(deltaPlaneX,deltaPlaneY);
                        float angleDiff = fabs(anglePlane - anglePath);
                        Err_position_segment = fabs(sin(angleDiff)*sqrt(deltaPlaneX*deltaPlaneX
+deltaPlaneY*deltaPlaneY));
//              }

//              if (nav_in_circle){
                        deltaPlaneX = nav_circle_x - estimator_x;
                        deltaPlaneY = nav_circle_y - estimator_y;
                        Err_position_circle = fabs(sqrt(deltaPlaneX*deltaPlaneX
+deltaPlaneY*deltaPlaneY)-nav_circle_radius);
//              }
                if (Err_position_circle < Err_position_segment){
                        Err_position = Err_position_circle;
                }
                else
                        Err_position = Err_position_segment;

                if (Err_position>TolerancePosition){
                        SquareSumErr_position += (Err_position * Err_position);
                }
        #endif
        }

        DOWNLINK_SEND_FLIGHT_BENCHMARK(DefaultChannel, &SquareSumErr_airspeed, &SquareSumErr_altitude,
&SquareSumErr_position, &Err_airspeed, &Err_altitude, &Err_position)

}

void flight_benchmark_reset( void ) {
        SquareSumErr_airspeed = 0;
        SquareSumErr_altitude = 0;
        SquareSumErr_position = 0;
}
```

AMS 5812
Drucksensor mit Analog- und Digital-Ausgang (I²C)

AMSYS
Analog - Digitale
Mikromechanische
Sensorsysteme

ALLGEMEINE BESCHREIBUNG

Die Drucksensoren der Serie AMS 5812 sind hochgenaue OEM-Drucksensoren mit einem analogen 0,5 – 4,5 V Spannungsausgang und einer digitalen I^2C-Schnittstelle.

Die AMS 5812 sind kalibriert und im industriellen Temperaturbereich von -25…85°C kompensiert. Sie werden in einem Dual-In-Line Package (DIP) zur Leiterplattenmontage geliefert und sind ohne weitere Komponenten betriebsbereit. Der elektrische Anschluss erfolgt über Lötpins in DIP-Konfiguration, der Druckanschluss über zwei vertikale metallische Stutzen.

Durch die Kombination von qualitativ hochwertigen piezoresistiven Drucksensorelementen mit einem modernen mixed-signal CMOS-ASIC mit volldigitaler Korrektur zur Signalauswertung auf einem Keramiksubstrat werden bei den Sensoren der Baureihe AMS 5812 höchste Messgenauigkeit sowie ausgezeichnete Drift- und Langzeitstabilität erreicht. Bemerkenswert ist das ausgezeichnete Überdruckverhalten der AMS 5812.

Die Sensoren der Serie AMS 5812 sind in verschiedenen Druckbereichen und Varianten verfügbar: von 0…0,075 PSI bis zu 0…100 PSI als differentielle (relative) Variante sowie im Bereich 0…15 PSI als () Absolutdruck- und barometrische Variante. Für den Druckbereich von -0,075/+0,075 PSI bis -15/+15 PSI liegen sie als bidirektionale differentielle Version vor für Unter- und Überdruck. Auf Anfrage können die Sensoren auch auf den Druckbereich Bar abgeglichen oder kundenspezifisch modifiziert werden.
Der analoge Spannungsausgang ist ratiometrisch zur Versorgungsspannung von 5V.

AMSYS GmbH & Co. KG
An der Fahrt 13
55124 Mainz

Tel.: +49 (0)6131-469875 - 0
Fax: +49 (0)6131-469875 - 66
Internet: www.amsys.de
E-Mail: info@amsys.de

EIGENSCHAFTEN

- **Kalibrierter und temperaturkompensierter Drucksensor**
- **Ratiometrischer 0.5-4.5V Spannungsausgang**
- **Digitaler kalibrierter Druck- und Temperaturwert über I²C-Schnittstelle**
- **Hohe Genauigkeit bei RT**
- **Kleiner Gesamtfehler im Temperaturbereich von -25 .. 85°C**
- **Differentielle/relative, absolute (barometrische) Varianten**
- **Kleines DIP-Gehäuse**
- **Ready-to-use**
- **RoHS konform**

Datenblatt, Rev. 1.3 August 2009

AMS 5812
Drucksensor mit Analog- und Digital-Ausgang (I²C)

ANWENDUNGEN

- Statische und dynamische Druckmessung
- Barometrische Messung
- Vakuummessung
- Füllstandsmessung
- Durchflussmessung
- Beatmungsgeräte (Medizintechnik)
- Heizung / Lüftung /Klima (HVAC)

DRUCKBEREICHE

Typ (Bezeichnung)	Druckart	Druckbereich in PSI	Berstdruck in PSI	Druckbereich in mbar	Berstdruck in bar
Niedrigstdrucksensoren					
AMS 5812-0000-D	differentiell / relativ	0 ….0,075	72	0 … 5,17	5
AMS 5812-0001-D	differentiell / relativ	0 … 0,15	72	0 … 10,34	5
AMS 5812-0000-D-B	bidirektional differentiell	-0,075 / +0,075	72	-5,17 / +5,17	5
AMS 5812-0001-D-B	bidirektional differentiell	-0,15 / +0,15	72	-10,34 / +10,34	5
Niederdrucksensoren					
AMS 5812-0003-D	differentiell / relativ	0 … 0,3	72	0 … 20,68	5
AMS 5812-0008-D	differentiell / relativ	0 … 0,8	72	0 … 55,16	5
AMS 5812-0015-D	differentiell / relativ	0 … 1,5	72	0 … 103,4	5
AMS 5812-0003-D-B	bidirektional differentiell	-0,3 / +0,3	72	-20,68 / +20,68	5
AMS 5812-0008-D-B	bidirektional differentiell	-0,8 / +0,8	72	-55,16 / +55,16	5
AMS 5812-0015-D-B	bidirektional differentiell	-1,5 / +1,5	72	-103,4 / +103,4	5
Standarddrucksensoren					
AMS 5812-0030-D	differentiell / relativ	0 … 3	72	0 … 206,8	5
AMS 5812-0050-D	differentiell / relativ	0 … 5	72	0 … 344,7	5
AMS 5812-0150-D	differentiell / relativ	0 … 15	72	0 … 1034	5
AMS 5812-0300-D	differentiell / relativ	0 … 30	225	0 … 2068	15,5
AMS 5812-0600-D	differentiell / relativ	0 … 60	225	0 … 4137	15,5
AMS 5812-1000-D	differentiell / relativ	0 … 100	225	0 … 6895	15,5
AMS 5812-0030-D-B	bidirektional differentiell	-3 / +3	72	-206,8 / +206,8	5
AMS 5812-0050-D-B	bidirektional differentiell	-5 / +5	72	-344,7 / +344,7	5
AMS 5812-0150-D-B	bidirektional differentiell	-15/ +15	72	-1034 / +1034	5
AMS 5812-0150-A	absolut	0 … 15	72	0 … 1034	5
AMS 5812-0150-B	barometrisch	11 … 17,5	72	758,4 … 1206	5

Tabelle 1: **Standard Druckbereiche der AMS 5812**

AMS 5812
Drucksensor mit Analog- und Digital-Ausgang (I²C)

RANDBEDINGUNGEN

Parameter	Minimum	Typisch	Maximum	Einheit
Maximale Versorgungsspannung : V_S (max)			6.5	V
Betriebstemperatur: T_{op}	-25		85	°C
Lagertemperatur: T_{amb}	-40		125	°C

SPEZIFIKATIONEN

Alle Parameter gelten für V_S =5.0V und T_{op} =25°C, falls nicht anders angegeben.

Parameter	Minimum	Typisch	Maximum	Einheit
Druckbereiche für verschiedene Typen [1]				
Niedrigstdrucksensoren (differentiell/relativ)	0 … 0,075		0 … 0,15	PSI
Niedrigstdrucksensoren (bidirektional differentiell)	-0,075 / +0,075		-0,15 / +0,15	PSI
Niederdrucksensoren (differentiell/relativ)	0,3		1,5	PSI
Niederdrucksensoren (bidirektional differentiell)	-0,3 / +0,3		-1,5 / +1,5	PSI
Standarddrucksensoren (differentiell/relativ)	0 … 3		0 … 100	
Standarddrucksensoren (bidirektional differentiell)	-3 / +3		-15 / +15	
Analoges Ausgangssignal (nur Druckmessung) [2]				
bei spezifiziertem Minimaldruck (gem. Druckbereich)		0.5		V
bei spezifiziertem Maximaldruck (gem. Druckbereich)		4.5		V
Spanne des Ausgangssignales (FSO) [3]		4		V
ohne Druckbeaufschlagung (bidirektional differentiell)		2,5		V
Digitales Ausgangssignal (Druckmessung) [4]				
bei spezifiziertem Minimaldruck (gem. Druckbereich)		3277		Counts
bei spezifiziertem Maximaldruck (gem. Druckbereich)		29491		Counts
Spanne des Ausgangssignales (FSO) [3]		26214		Counts
ohne Druckbeaufschlagung (bidirektional differentiell)		16384		Counts
Digitales Ausgangssignal (Temperaturmessung) [5]				
bei Minimaltemperatur -25°C		3277		Counts
bei Maximaltemperatur 85°C		29491		Counts
Genauigkeit [6] **(Druckmessung) bei T= 25°C**				
Niedrigstdrucksensoren (0,075, 0,15 PSI)			±1,5	%FSO
Niederdrucksensoren (0,3, 0,8, 1,5 PSI)			±1,0	%FSO
Standarddrucksensoren			±0.5	%FSO
Gesamtfehler [7] **(Druckmessung) bei T= –25...85°C**				
Niedrigstdrucksensoren (0,075, 0,15 PSI)			±2,0	%FSO
Niederdrucksensoren (0,3, 0,8, 1,5 PSI)			±1,5	%FSO
Standarddrucksensoren			±1,0	%FSO
Gesamtfehler Temperaturmessung				
alle Sensortypen T= –25...85°C			±3,0	%FSO
Auflösung			0,05	%FS

AMS 5812
Drucksensor mit Analog- und Digital-Ausgang (I²C)

Versorgungsspannung (V_S gegen Masse)	4,75	5	5,25	V
Eigenstromaufnahme			5	mA
Auflösung A/D-Wandler	14		14	Bit
Auflösung D/A Wandler	11		11	Bit
Reaktionszeit (10%...90% Anstiegszeit)		1	2	ms
Lastwiderstand am Ausgang R_L	2k			Ω
Kapazitive Last			50	nF
Ratiometriefehler (bei V_S = 4.75 …5.25V)			500	ppm
I2C Schnittstelle				
Input-High-Level	90		100	% V_S
Input-Low-Level	0		10	% V_S
Output-Low-Level	0		10	% V_S
Load capacitance @ SDA			400	pF
Clock frequency SCL			400	kHz
Pull-up Resistor	500			Ω
Druckwechsel	10^6			
Kompensierter Temperaturbereich	–25		85	°C
Gewicht		3		Gramm
Medienkompatibilität	vgl. Anmerkungen [8) 9)]			

Tabelle 2: **Spezifikationen**

ANMERKUNGEN

1) Vgl. *Tabelle 1*
2) Das analoge Ausgangssignal (nur Druckmessung) ist ratiometrisch zur Versorgungsspannung
3) Full Span Output (FSO) ist die algebraische Differenz zwischen dem Ausgangssignal bei spezifiziertem Maximaldruck und dem Ausgangssignal bei spezifiziertem Minimaldruck (gem. Druckbereich)
4) Das digitale Druck-Ausgangssignal ist nicht ratiometrisch zur Versorgungsspannung.
5) Das digitale Temperatur-Ausgangssignal ist nicht ratiometrisch zur Versorgungsspannung. Der ausgegebene Temperaturwert wird an der Druckmesszelle gemessen und ist die Sensortemperatur (incl. der Eigenerwärmung).
6) Genauigkeit ist definiert als max. Abweichung des Messwertes von der idealen Kennlinie bei RT in %FSO inkl. Einstellfehler (Nullpunkt und Spanne), Nichtlinearität, Druckhysterese, Wiederholgenauigkeit. Die Nichtlinearität ist die gemessene Abweichung von der ist Best Fit Straight Line (BFSL) über den Druckbereich. Die Druckhysterese ist die maximale Abweichung des Ausgangswertes an einem beliebigen Druckwert innerhalb des Druckbereichs bei einer zyklischen Änderung des Drucks innerhalb des Druckbereichs. Die Wiederholgenauigkeit ist maximale Abweichung des Ausgangswertes an einem beliebigen Druckwert innerhalb des Druckbereichs nach 10 Druckzyklen.
7) Gesamtfehler ist definiert als max. Abweichung des Messwertes von der idealen Kennlinie in %FSO im gesamten Temperaturbereich (-25 … 85°C)
8) Medienverträglichkeit Anschluss 1 (Bezeichnung port 1 siehe *Abbildung 5*): Saubere, trockene Gase, die nicht aggressiv sind gegen Silizium, RTV-Silicone, Gold (basische oder säurehaltige Flüssigkeiten können zur Zerstörung des Sensors führen)
9) Medienverträglichkeit Anschluss 2 (Bezeichnung port 2 siehe *Abbildung 5*): Flüssigkeiten und Gase, die nicht aggressiv sind gegen Silizium, Pyrex, RTV-Silicone.

AMS 5812
Drucksensor mit Analog- und Digital-Ausgang (I²C)

FUNKTIONSBESCHREIBUNG

Die Sensoren der Baureihe AMS 5812 kombinieren eine piezoresistive Silizium-Druckmesszelle und ein mixed-signal CMOS-ASIC mit volldigitaler Korrektur zur Signalauswertung auf einem Dickschicht-Keramiksubstrat.

Das Funktionsprinzip der Sensoren AMS 5812 wird anhand von *Abbildung 1* erläutert.

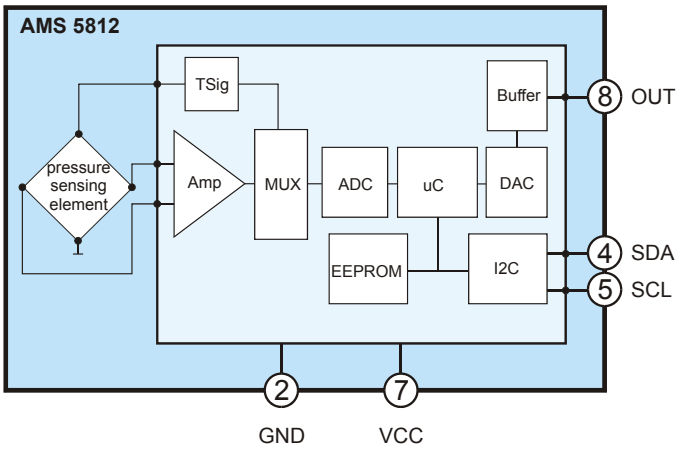


Abbildung 1: **Funktionsprinzip**

Die eigentliche Druckmessung findet an der piezoresistiven Druckmesszelle der AMS 5812 statt. Dort wird der zu messende Druck in ein differentielles, weitgehend druckproportionales Spannungssignal gewandelt. Dieses differentielle Spannungssignal wird dann im ASIC aufbereitet und korrigiert. Die Signalaufbereitung und Korrektur erfolgt in mehreren Schritten.

Zunächst wird das Spannungssignal von der Messzelle im ASIC verstärkt, über einen Multiplexer zum ADC weitergeleitet und dort in einen Digitalwert konvertiert. Zur Digitalisierung wird eine Wandlungstiefe des ADC von 14 Bit verwendet. Im nachfolgenden Micro-Controller Block des ASICs wird das digitalisierte Signal dann korrigiert und kalibriert.

Durch den werksseitigen Präzisionsabgleich der Sensoren werden für jeden einzelnen Sensor Korrekturkoeffizienten ermittelt und diese im EEPROM gespeichert. Auf diese Weise wird eine individuelle Kalibration und eine individuelle Korrektur (d.h. Temperaturkompensation und Linearisierung) des digitalisierten Drucksignales möglich. Das zur Temperaturkompensation notwendige Temperatursignal wird ebenfalls an der piezoresistiven Druckmesszelle erfasst und über den Multiplexer zum ADC weitergeleitet und dort in einen Digitalwert konvertiert. Im Micro-Controller Block des ASICs läuft ein zyklisches Programm, das laufend auf Basis der jeweils digitalisierten Druck- und Temperaturwerte mit Hilfe der Korrekturkoeffizienten das korrigierte und normierte digitale Drucksignal errechnet. Zusätzlich wird auch ein normiertes Temperatursignal berechnet. Die so errechneten, korrigierten 15-Bit Digitalwerte (Druck- und Temperaturwert) werden in das Ausgangsregister des ASICs geschrieben und laufend aktualisiert (typ. alle 0,5ms bei 14bit ADC-Auflösung).

Über die I2C Schnittstelle an PIN4 (SDA) und PIN5 (SCL) des Sensors kann das korrigierte digitale Drucksignal und das normierte digitale Temperatursignal ausgelesen werden. Die über die I2C Schnittstelle verfügbaren Digitalwerte (Druck und Temperatur) sind nicht ratiometrisch zur Versorgungsspannung.

AMS 5812
Drucksensor mit Analog- und Digital-Ausgang (I²C)

Zur Erzeugung des analogen Ausgangssignals wird das korrigierte digitale Drucksignal im ASIC mit einem 11bit DAC in eine analoge Spannung zurück gewandelt. Das normierte analoge Spannungsausgangssignal (0,5…4,5V) ist ratiometrisch zur Versorgungsspannung und steht an PIN 8 (OUT) des Sensors an.

INBETRIEBNAHME

Zum elektrischen Anschluss wird der Sensor auf eine Leiterplatte montiert. Die prinzipielle Beschaltung der Sensoren AMS 5812 bei Betrieb des Analogausgangs und des Digitalausganges/ I2C-Schnittstelle ist in *Abbildung 2* dargestellt.

Bei reinem Analogbetrieb genügt der Anschluss von PIN2 (GND), PIN7 (VCC) und PIN8 (OUT).

Bei reinem Digitalbetrieb muss neben PIN2 (GND) und PIN7 (VCC) noch der I2C-Bus an PIN4 (SDA) und PIN5 (SCL) angeschlossen werden.

Wichtig: Jede Busleitung muss über einen Pull-Up Widerstand (Empfehlung 4,7 kΩ) an VCC (oder +5V) angeschlossen sein, die zusätzlich eingezeichneten Serienwiderstände (Empfehlung 330 Ω) sind optional.

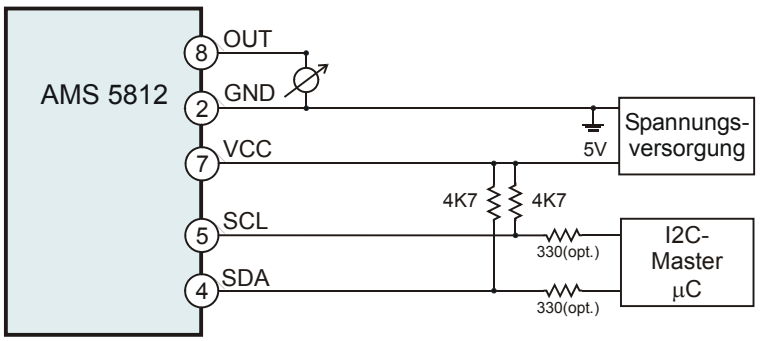


Abbildung 2: **Elektrische Beschaltung**

Der Druckanschluss erfolgt über die metallischen Druckanschlussstutzen des Sensors. Je nach Sensortyp und Druckart werden ein oder zwei Druckanschlussstutzen an das jeweilige Messmedium /-volumen angeschlossen. Für die Drücke an den Anschlussstutzen 1 und 2 (Bezeichnung siehe *Abbildung 5*) gelten die folgenden Bedingungen (mit der Definition p_1 = Druck am Anschluss 1 und p_2 = Druck am Anschluss 2):

für differentielle / relative Drucksensoren: $p_1 > p_2$

für bidirektional differentielle Sensoren: $p_1 > p_2$ sowie $p_1 < p_2$ möglich.

für Absolutdrucksensoren, barometrische Sensoren: p_1 = Messdruck.

Es sind die Vorschriften bezüglich der Medienkompatibiliät (Anmerkungen Punkt 8 und 9) zu beachten.

AMS 5812
Drucksensor mit Analog- und Digital-Ausgang (I²C)

Die I2C-Schnittstelle der AMS 5812

Die Drucksensoren der Serie AMS 5812 verfügen über einen digitalen Ausgang (I2C-Schnittselle). Über die I2C-Schnittstelle können die jeweils aktuellen, korrigierten digitalen Druck- und Temperaturwerte aus dem Ausgangsregister des Sensors gelesen werden.

Die Kommunikation über den I2C-Bus erfolgt nach dem Master-Slave Prinzip, d.h. der Datentransfer wird immer durch einen Master z.B. einen Mikroprozessor initiiert, der die Sensoren anspricht, und die Sensoren AMS 5812, die immer als Slave arbeiten, antworten.

Für die Kommunikation über die I2C Schnittstelle sind zwei Bus-Leitungen erforderlich: eine serielle Datenleitung SDA (serial data) und eine serielle Taktleitung SCL (serial clock). SDA und SCL sind bidirektionale Leitungen die über Pull-up Widerstände (Empfehlung R=4,7 kΩ) an die positive Versorgungsspannung angeschlossen werden (siehe *Abbildung 2*).

Die Kommunikation erfolgt nach dem üblichen I2C-Protokoll (siehe *Abbildung 3*).

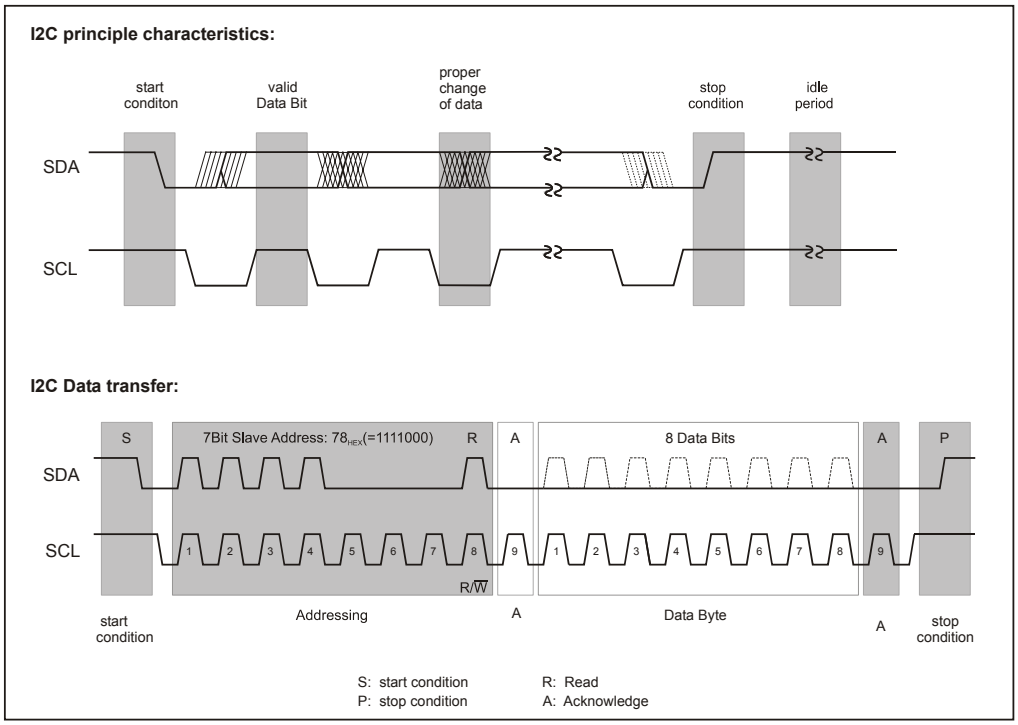


Abbildung 3: **Grundlagen I2C-Protokoll**

AMS 5812
Drucksensor mit Analog- und Digital-Ausgang (I²C)

Es werden folgende I2C-Kommunikationsphasen unterschieden:

Idle Period (Bus im Freilauf)

Im Freilauf werden beide I2C-Busleitungen (SDA und SCL) über die Pull-up Widerstände auf Versorgungsspannung gezogen (Pegel „High“)

Start S (Startbedingung)

Die Startbedingung wird immer durch den I2C-Master generiert. Sie ist erfüllt, wenn auf der SDA-Leitung ein Übergang vom Pegel „High“ auf „Low“ stattfindet, während der Pegel auf der SCL-Leitung „High“ ist. Das Auslesen von digitalen Werten aus dem Ausgangsregister beginnt immer mit einer Startbedingung.

Stop P(Stoppbedingung)

Die Stoppbedingung wird immer durch den I2C-Master generiert. Sie ist erfüllt, wenn auf der SDA-Leitung ein Übergang vom Pegel „Low“ auf „High“ stattfindet, während der Pegel auf der SCL-Leitung „High“ ist. Das Auslesen von digitalen Werten endet mit einer Stoppbedingung.

Valid Data (gültige Daten)

Die Datenübertragung erfolgt immer in Bytes (8 Bit) beginnend mit dem höchstwertigen Bit (MSB). Es wird jeweils ein Bit pro Clock-Impuls übertragen. Die übertragenen Bits sind nur gültig, wenn (nach einer Startbedingung) der Pegel auf der SDA-Leitung konstant bleibt während der Pegel auf der SCL-Leitung „High“ ist. Änderungen des SDA-Pegels müssen stattfinden, während der Pegel auf der SCL-Leitung „Low“ ist.

Acknowledge A (Bestätigung)

Nach der Datenübertragung eines Bytes muss vom jeweiligen Empfänger (Master bzw. Slave) eine Empfangsbestätigung (zusätzliches Acknowledge Bit) gesendet werden. Dazu erzeugt der Master einen zusätzlichen, dem Acknowledge Bit zugeordneten Clock-Impuls. Der Empfänger sendet das Acknowledge Bit, indem er den Pegel auf der SDA-Leitung während des zusätzlichen Clock-Impulses auf Low zieht.

Adressing/ Slave Adress (Adressierung / I2C Adresse AMS 5812)

Zur Adressierung /Auswahl eines Sensors sendet der Master das Adressierungs-Byte. Das Adressierungs-Byte enthält die individuelle 7 Bit Slave Adresse des jeweils angesprochenen Slave (AMS 5812) und ein sog. data direction Bit (R/$\overline{W}$). Eine „0“ steht für einen Datentransfer von Master zum Sensor/Slave (W: Schreiben), eine „1“ steht für eine Datenanforderung (R: Lesen).

Die AMS 5812 haben werksseitig die 7 Bit Slave Adresse 0x78Hex (1111000b)[1]

Auslesen der Digitalwerte über die I2C-Schnittstelle der AMS 5812

Das Auslesen der 15 Bit Digitalwerte für Druck und Temperatur aus dem Ausgangsregister des AMS 5812 erfolgt über die I2C-Schnittstelle der AMS 5812. Es wird anhand von *Abbildung 4* erläutert.

[1] Sollen mehrere AMS 5812 an einen I2C-Bus angeschlossen werden, so muss jedem Sensor eine individuelle Adresse einprogrammiert werden. Auf Anfrage kann jedem Sensor werksseitig eine zweite 7bit-Adresse programmiert werden. Alternativ kann der Kunde diese Programmierung auch mit dem Starter-Kit vornehmen. Die so programmierten AMS 5812 hören dann auf beide Adressen.

AMS 5812
Drucksensor mit Analog- und Digital-Ausgang (I²C)

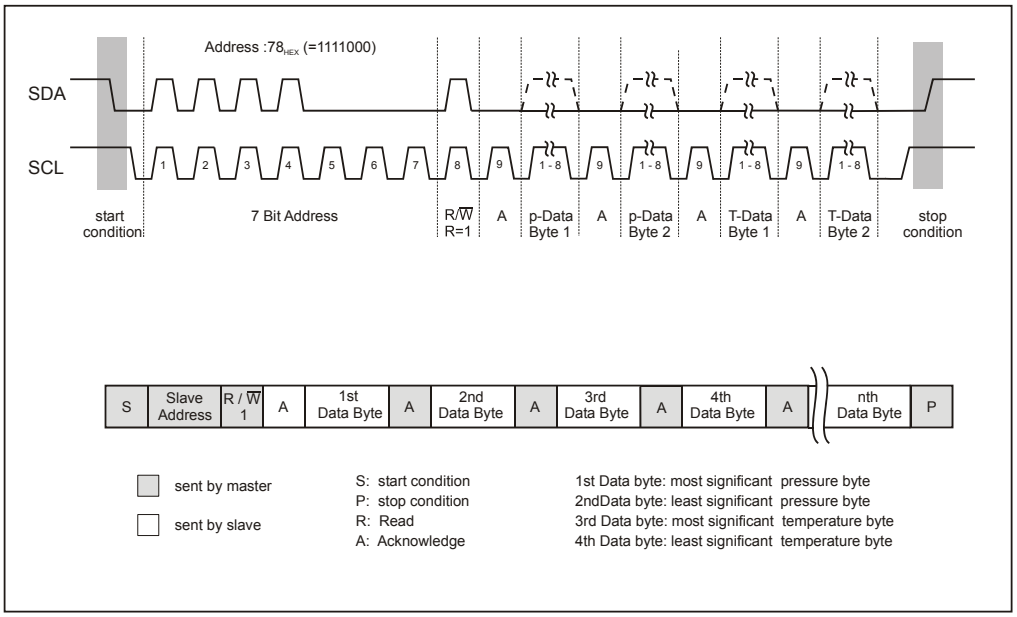


Abbildung 4: **Auslesen der digitalen Druck- und Temperaturwerte**

Der Datentransfer über den I2C-Bus beginnt immer mit einer Datenanforderung durch den I2C-Master. Der I2C-Master generiert dazu eine Startbedingung auf den Busleitungen. Danach sendet der I2C-Master das Adressierungs-Byte, das die 7 Bit Slave Adresse des angesprochenen Drucksensors (werksseitig haben die AMS 5812 die Slave Adresse 0x78Hex = 1111000b [1]) und das data direction Bit R/W=1 (für Read) enthält. Der angesprochene Drucksensor antwortet darauf zunächst mit einem Acknowledge-Bit. Danach startet der angesprochene Drucksensor mit der Datenübertragung aus dem Ausgangsregister. Für den aktuellen 15 Bit Druck- und den 15 Bit Temperaturwert werden insgesamt 4 Daten-Bytes vom Drucksensor and den I2C-Master übertragen. Zuerst werden 2 Bytes für den aktuellen Druckwert und danach 2 Bytes für den aktuellen Temperaturwert gesendet, wobei immer zuerst das höherwertige Byte gesendet wird. Nach jedem übertragenen Daten-Byte muss eine Bestätigung durch den I2C-Master in Form eines Acknowledge Bits erfolgen. Fehlt das Acknowledge Bit, so wird die Datenübertragung vom Drucksensor AMS 5812 unterbrochen. Die Datenübertragung wird durch eine Stoppbedingung vom I2C-Master beendet. Sendet der I2C-Master anstelle der Stopp-Bedingung ein weiteres Acknowledge-Bit nach dem letzten der 4 Daten-Bytes, so überträgt der Drucksensor erneut die jeweils aktuellen Druck- und Temperaturwerte aus dem Ausgangsregister.

Berechnung des aktuellen Druck- und Temperaturwertes aus den gelesenen 15-Bit Digitalwerten

Die digitalen Druck- und Temperaturwerte werden als einheitenlose 15 Bit Zahl übertragen, die in die physikalischen Einheiten Druck in PSI (oder bar) und Temperatur in °C umgerechnet werden müssen.

AMS 5812
Drucksensor mit Analog- und Digital-Ausgang (I²C)

Die Berechnung des anliegenden Drucks p in PSI (oder bar) erfolgt aus dem digitale Druckwert mit den folgenden Formeln

$$p = \frac{Digoutp(p) - Digoutp_{min}}{Sensp} + p_{min} \quad \text{mit} \quad Sensp = \frac{Digoutp_{max} - Digoutp_{min}}{p_{max} - p_{min}} \tag{1}$$

Darin bezeichnet ***p*** den aktuellen Druck in PSI (oder bar), $\boldsymbol{p_{min}}$ den Minamaldruck und $\boldsymbol{p_{max}}$ den Maximaldruck in PSI (oder bar) gemäß dem Druckbereich, ***Digoutp(p)*** den aktuellen digitalen Druckwert in Counts, $\boldsymbol{Digoutp_{min}}$ und $\boldsymbol{Digoutp_{max}}$ den digitalen Druckwert bei Minamaldruck bzw. Maximaldruck in Counts und ***Sensp*** die Sensitivität des Drucksensors in Counts/PSI (oder Counts/bar).

Die Berechnung der aktuellen Sensortemperatur in °C aus dem digitalen Temperaturwert erfolgt analog, d.h. mit den gleichen Formeln, wobei an allen Stellen p durch T zu ersetzen ist.

Beispiel

Für einen Drucksensor vom Typ AMS 5812-0015-D (0…1.5PSI differentiell) wird ein digitaler Druckwert von

Digoutp(p) = $550A_{Hex}$ counts = 21770_{Dec} counts

und ein digitaler Temperaturwert von

DigoutT(T) = $3A9A_{Hex}$ counts = 15002_{Dec} counts

gelesen.

Mit p_{min} = 0 PSI, p_{max}= 1.5 PSI und $Digoutp_{min}$= 3277, $Digoutp_{max}$ = 29491 berechnet sich nach Formel (1) der aktuelle Druck zu

$$p = \frac{(21770 - 3277)\ \text{counts}}{26214/1{,}5\ \text{counts/PSI}} + 0\ \text{PSI} = 1{,}058\ \text{PSI}$$

Mit T_{min} = -25 °C, T_{max}= 85 °C und $DigoutT_{min}$= 3277, $DigoutT_{max}$ = 29491 berechnet sich nach Formel (1) die aktuelle Sensortemperatur zu

$$T = \frac{(15002 - 3277)\ \text{counts}}{26214/110\ \text{counts/°C}} + (-25)\ \text{°C} = 24{,}2\ \text{°C}$$

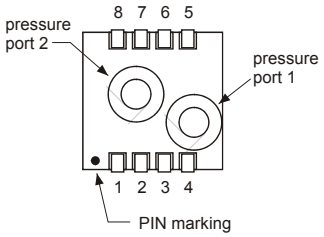
AMS 5812
Drucksensor mit Analog- und Digital-Ausgang (I²C)

ABMESSUNGEN UND PINBELEGUNG

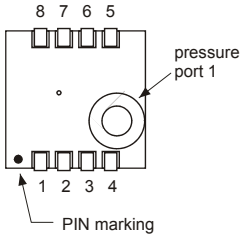
Die Sensoren AMS 5812 werden in einem Dual-In-Line Package (DIL) zur Leiterplattenmontage geliefert.
Die nachfolgende *Abbildung 5* zeigt das Pin-Out und die Abmessungen des DIL-Gehäuses.

Pin-Out and pressure connection:

differential types:



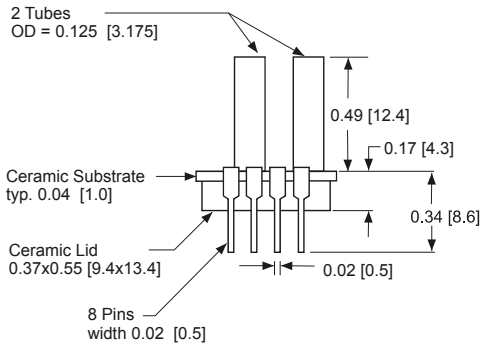
absolute, barometric types:



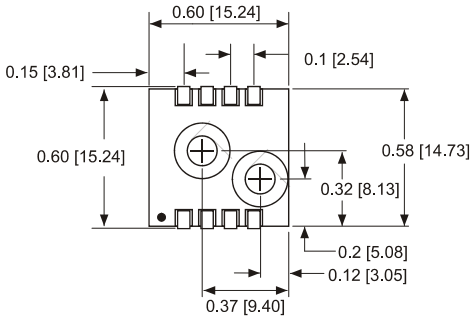
Pin	Description
1	N.C.
2	GND
3	N.C.
4	SDA
5	SCL
6	N.C.
7	VCC
8	OUT

Package Dimensions:

Side view :



Top view :



alle Dimensionen in inch [mm]

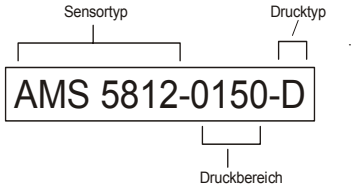
Abbildung 5: **Abmessungen**

Die Sensoren der Reihe AMS 5812 sind während ihrer Lebensdauer wartungsfrei.

AMS 5812
Drucksensor mit Analog- und Digital-Ausgang (I²C)

BESTELLINFORMATIONEN

Bestellcode:



Druckbereich:

Code Druckbereich	PSI	mbar	kPa
0000	0,075	5,17	0,517
0001	0,15	10,34	1,034
0003	0,3	20,68	2,068
0008	0,8	55,16	5,516
0015	1,5	103,4	10,34
0030	3,0	206,8	20,68
0050	5,0	344,7	34,47
0150	15	1034	103,4
0300	30	2068	206,8
0600	60	4137	413,7
1000	100	6895	689,5

Tabelle 3: **Druckbereiche**

Drucktyp:

Code Drucktyp		verfügbare Druckbereiche
D	differentiell / relativ (gage)	0…0,075 PSI bis 0…100 PSI
D-B	bidirektional differentiell	-0,075 / +0.075PSI bis -15 / +15 PSI
A	absolut	0…15 PSI
B	barometrisch (absolut)	11 … 17,5 PSI

Tabelle 4: **Drucktypen**

Zubehör

Zu den Sensoren AMS 5812 ist ein Starter-Kit mit Software erhältlich. Der Starter-Kit ermöglicht eine einfache Inbetriebnahme des Digitalausgangs (I2C-Bus) über die serielle RS232-Schnittstelle eines PCs. Darüber hinaus kann mit dem Starter-Kit dem jeweiligen Sensor neben der werksseitigen I2C-Adresse (0x78Hex) eine zweite individuelle I2C-Adresse einprogrammiert werden.

MB1200
MB1300

XL- MaxSonar®- EZ0™ (MB1200)
XL- MaxSonar®- AE0™ (MB1300)
Sonar Range Finder with High Power Output, Noise Rejection, Auto Calibration & Long-Range Wide Detection Zone (Hardware gain of 4000)


approximately actual size

The MB1200 and MB1300 have a new high power output along with real-time auto calibration for changing conditions (temperature, voltage or acoustic or electrical noise) that ensure you receive the most reliable (in air) ranging data for every reading taken. The MB1200 and MB1300 low power 3.3V to 5V operation provides very short to long-range detection and ranging, in a tiny and compact form factor. The MB1200 and MB1300 detect objects from 0-cm to 765-cm (25.1 feet) and provides sonar range information from 20-cm out to765-cm with 1-cm resolution. Objects from 0-cm* to 20-cm typically range as 20-cm. (*Objects from 0-mm to 1-mm may not be detected.) The interface output formats included are pulse width output (MB1200), real-time analog voltage envelope (MB1300), analog voltage output, and serial digital output.*

Features

- High acoustic power output
- Real-time auto calibration and noise rejection for every ranging cycle
- Calibrated beam angle
- Continuously variable gain
- Object detection as close as 1-mm from the sensor
- 3.3V to 5V supply with very low average current draw
- Readings can occur up to every 100mS, (10-Hz rate)
- Free run operation can continually measure and output range information
- Triggered operation provides the range reading as desired
- All interfaces are active simultaneously
- Serial, 0 to Vcc, 9600Baud, 81N
- Analog, (Vcc/1024) / cm
- Pulse Width (MB1200)
- Real-time analog envelope (MB1300)
- Sensor operates at 42KHz

Benefits

- Acoustic and electrical noise resistance
- Reliable and stable range data
- Sensor dead zone virtually gone
- Low cost
- Quality controlled beam characteristics
- Very low power range finder suited for multiple sensor or battery based systems
- Ranging can be triggered externally or internally
- Sensor reports the range reading directly, frees up user processor
- Fast measurement cycle
- User can choose any of the sensor outputs
- No calibration requirement is perfect for when objects may be directly in front of the sensor during power up
- Small size allows for easy mounting

Applications and Uses

- UAV blimps, micro planes, and some helicopters
- Bin level measurement
- Proximity zone detection
- People detection
- Robot ranging sensor
- Autonomous navigation
- Environments with acoustic and electrical noise
- Multi-sensor arrays
- Distance measuring
- Long range object detection
- Wide beam sensitivity
- Users who prefer to process the analog voltage envelope (MB1300)
- -40°C to +65°C operation (+85°C limited operation)

Email: info@maxbotix.com
Web: www.maxbotix.com
PD10335b

MB1200 & MB1300 Pin Out

Pin 1 - Leave open (or high) for serial output on the Pin 5 output. When Pin 1 is held low the Pin 5 output sends a pulse (instead of serial data), suitable for low noise chaining.

Pin 2 - MB1200 (PW) This pin outputs a pulse width representation of range. To calculate distance, use the scale factor of 58uS per cm.
MB1300 (AE) This pin outputs the analog voltage envelope of the acoustic wave form.

Pin 3 - (AN) This pin outputs analog voltage with a scaling factor of (Vcc/1024) per cm. A supply of 5V yields ~4.9mV/cm., and 3.3V yields ~3.2mV/cm. Hardware limits the maximum reported range on this output to ~700 cm at 5V and ~600 cm at 3.3V. The output is buffered and corresponds to the most recent range data.

Pin 4 - (RX) This pin is internally pulled high. The MB1200 &MB1300 will continually measure range and output if the pin is left unconnected or held high. If held low the MB1200 &MB1300 will stop ranging. Bring high 20uS or more for range reading.

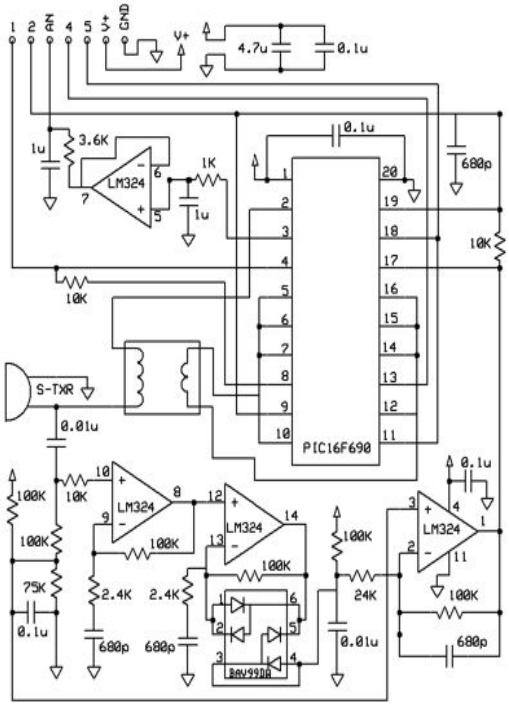
Pin 5 - (TX) When Pin 1 is open or held high, the Pin 5 output delivers asynchronous serial with an RS232 format, except voltages are 0-Vcc. The output is an ASCII capital "R", followed by three ASCII character digits representing the range in centimeters up to a maximum of 765, followed by a carriage return (ASCII 13). The baud rate is 9600, 8 bits, no parity, with one stop bit. Although the voltage of 0-Vcc is outside the RS232 standard, most RS232 devices have sufficient margin to read 0-Vcc serial data. If standard voltage level RS232 is desired, invert, and connect an RS232 converter such as a MAX232.
When Pin 1 is held low, the Pin 5 output sends a single pulse, suitable for low noise chaining (no serial data).

V+ Operates on 3.3V - 5V. The average (and peak) current draw for 3.3V operation is 2.1mA (50mA peak) and at 5V operation is 3.4mA (100mA peak) respectively. Peak current is used during sonar pulse transmit.

GND Return for the DC power supply. GND (& V+) must be ripple and noise free for best operation.

MB1200 & MB1300 Circuit

The sensor functions using active components consisting of an LM324 and PIC16F690, together with a variety of other components. The schematic is shown to provide the user with detailed connection information.



MB1200 & MB1300 Real-time Operation & Timing

175mS after power-up, the XL-MaxSonar® is ready to begin ranging. If Pin-4 is left open or held high (20uS or greater), the sensor will take a range reading. The XL-MaxSonar® checks Pin-4 at the end of every cycle. Range data can be acquired once every 99mS. Each 99mS period starts by Pin-4 being high or open, after which the XL-MaxSonar® calibrates and calculates for 20.5mS, and after which, thirteen 42KHz waves are sent.

Then for the MB1200, the pulse width (PW) Pin-2 is set high. When an object is detected the PW pin is set low. If no target is detected the PW pin will be held high for up to 44.4mS (i.e. 58uS * 765cm) (For the most accurate range data, use the PW output of the MB1200 product.)

For the MB1300 with analog envelop output, Pin-2 will show the real-time signal return information of the analog waveform.

For both parts, the remainder of the 99mS time (less 4.7mS) is spent adjusting the analog voltage to the correct level, (and allowing the high acoustic power to dissipate). During the last 4.7mS, the serial data is sent.

MB1200 & MB1300 Real-time Auto Calibration

Each time after the XL-MaxSonar® takes a range reading it calibrates itself. The sensor then uses this data to range objects. If the temperature, humidity, or applied voltage changes during sensor operation, the sensor will continue to function normally. The sensor does not apply compensation for the speed of sound change versus temperature to any range readings.

MB1200 & MB1300 Real-time Noise Rejection

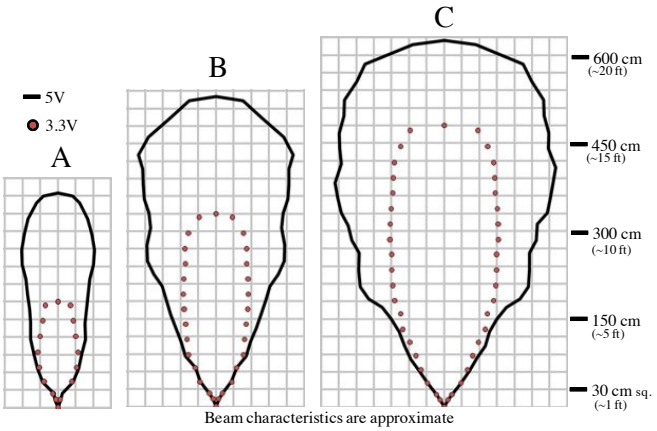
While the XL-MaxSonar® is designed to operate in the presence of noise, best operation is obtained when noise strength is low and desired signal strength is high. Hence, the user is encouraged to mount the sensor in such a way that minimizes outside acoustic noise pickup. In addition, keep the DC power to the sensor free of noise. This will let the sensor deal with noise issues outside of the users direct control (in general, the sensor will still function well even if these things are ignored). Users are encouraged to test the sensor in their application to verify usability.

For every ranging cycle, individual filtering for that specific cycle is applied. In general, noise from regularly occurring periodic noise sources such as motors, fans, vibration, etc., will not falsely be detected as an object. This holds true even if the periodic noise increases or decreases (such as might occur in engine throttling or an increase/decrease of wind movement over the sensor). Even so, it is possible for sharp non-periodic noise sources to cause false target detection. In addition, *(because of dynamic range and signal to noise physics,) as the noise level increases, at first only small targets might be missed, but if noise increases to very high levels, it is likely that even large targets will be missed.

*In high noise environments, if needed, use 5V power to keep acoustic signal power high. In addition, a high acoustic noise environment may use some of the dynamic range of the sensor, so consider a part with less gain such as the MB1210/MB1310, MB1220/MB1320 MB1230/MB1330 or MB1240/MB1340. For applications with large targets, consider a part with ultra clutter rejection like the MB7092.

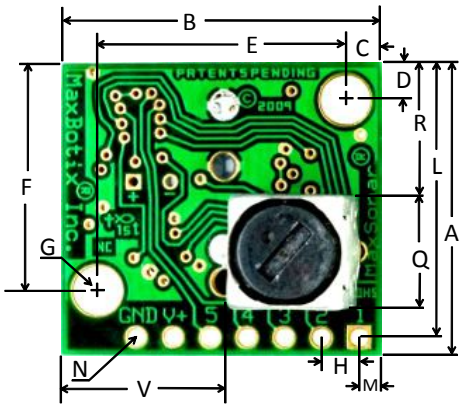
MB1200 & MB1300 Beam Characteristics

The MB1200 and MB1300 have a wide and long sensitive beam that offers excellent detection of objects and people. The MB1200 and MB1300 balances the detection of objects and people with minimal side-lobes. Sample results for measured beam patterns are shown to the right on a 30-cm grid. The detection pattern is shown for dowels of varying diameters that are place in front of the sensor;
(A) 6.1-mm (0.25-inch) diameter dowel,
(B) 2.54-cm (1-inch) diameter dowel,
(C) 8.89-cm (3.5-inch) diameter dowel,

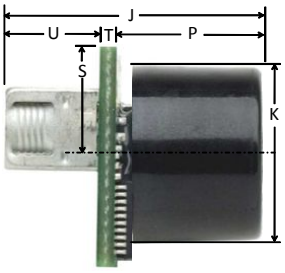


Beam characteristics are approximate

MB1200 & MB1300 Mechanical Dimensions

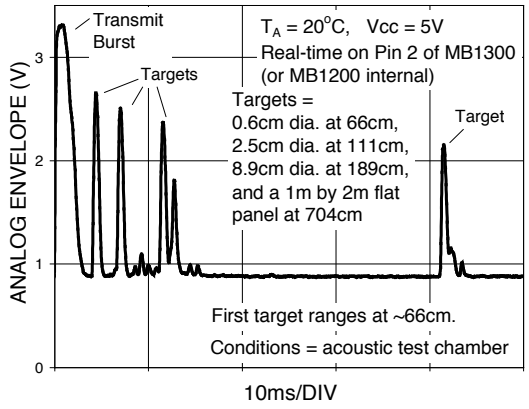


A	0.785"	19.9mm	L	0.735"	18.7mm
B	0.870"	22.1mm	M	0.065"	1.7mm
C	0.100"	2.54mm	N	0.038" dia.	1.0mm dia.
D	0.100"	2.54mm	P	0.537"	13.64mm
E	0.670"	17.0mm	Q	0.304"	7.72mm
F	0.610	15.5mm	R	0.351"	8.92mm
G	0.124" dia.	3.1mm dia.	S	0.413"	10.5mm
H	0.100"	2.54mm	T	0.063"	1.6mm
J	0.989"	25.11mm	U	0.368"	9.36mm
K	0.645"	16.4 mm	V	0.492"	12.5mm
values are nominal			Weight, 5.9 grams		

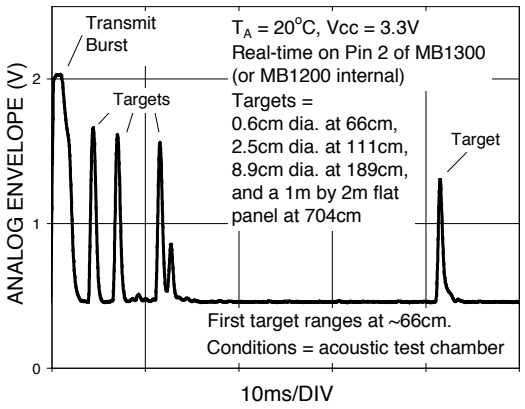


Typical Performance to Targets

Analog Envelope Output (Dowels, 5V)

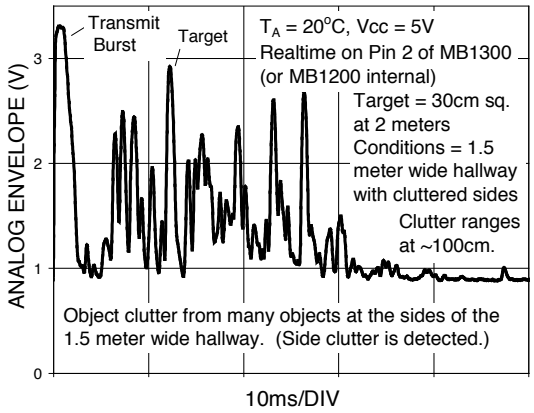


Analog Envelope Output (Dowels, 3.3V)

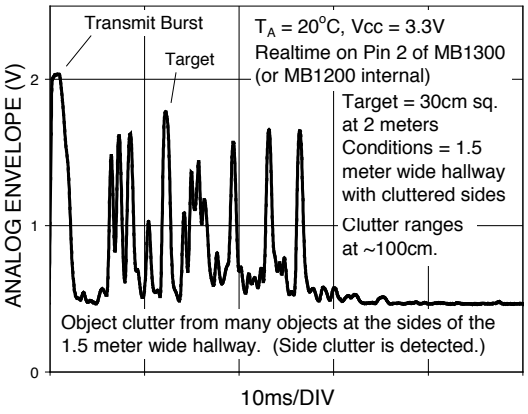


Typical Performance in Clutter

Analog Envelope Output (Clutter, 5V)



Analog Envelope Output (Clutter, 3.3V)



MA3 Miniature Absolute Magnetic Shaft Encoder

RoHS COMPLIANT

Description

The **MA3** is a miniature rotary absolute shaft encoder that reports the shaft position over 360 ° with no stops or gaps. The **MA3** is available with an analog or a pulse width modulated (PWM) digital output.

Analog output provides an analog voltage that is proportional to the absolute shaft position. Analog output is only available in 10-bit resolution.

PWM output provides a pulse width duty cycle that is proportional to the absolute shaft position. PWM output is available in 10-bit and 12-bit resolutions. While the accuracy is the same for both encoders, the 12-bit version provides higher resolution.

Three shaft torque versions are available. The standard torque version has a sleeve bushing lubricated with a viscous motion control gel to provide torque and feel that is ideal for front panel human interface applications.

The no torque added option has a sleeve bushing and a low viscosity lubricant (that does not intentionally add torque) for low RPM applications where a small amount of torque is acceptable.

The ball bearing version uses miniature precision ball bearings that are suitable for high speed and ultra low torque applications. The shaft diameter for ball bearing version option is 1/8" rather than 1/4".

Connecting to the **MA3** is simple. The 3-pin high retention snap-in 1.25mm pitch polarized connector provides for +5V, output, and ground.

Features

- Patent pending
- Miniature size (0.48" diameter)
- Non-contacting magnetic single chip sensing technology
- -40C to 125C. operating temperature range
- 10-bit Analog output - 2.6 kHz sampling rate
- 10-bit PWM output - 1024 positions per revolution, 1 kHz
- 12-bit PWM output - 4096 positions per revolution, 250 Hz

Related Products & Accessories

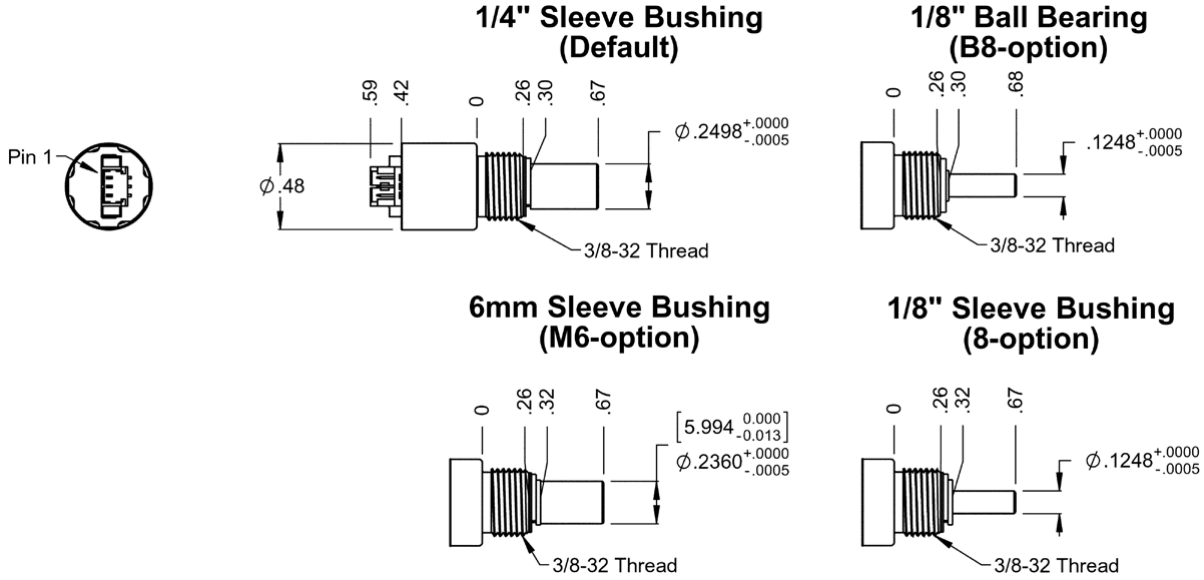
- CA-MIC3-SH-NC 3-Pin Micro / Unterminated Shielded Cable (Base price $7.30)
- CA-MIC3-W3-NC 3-Pin Micro / Unterminated 3-Wire Discrete Cable (Base price $6.80)
- CON-MIC3 3-Pin Micro Connector (Base price $3.15)

Mechanical Drawing

US DIGITAL
1400 NE 136th Avenue
Vancouver, Washington 98684, USA
info@usdigital.com
www.usdigital.com
Local: 360.260.2468
Toll-free: 800.736.0194

MA3 Miniature Absolute Magnetic Shaft Encoder

RoHS COMPLIANT



Mechanical

Specification	Sleeve Bushing	Ball Bearing
Moment of Inertia	4.1 x 10^-6 oz-in-s²	4.1 x 10^-6 oz-in-s²
Angular Accuracy	<0.5 deg. @ 25C	<0.5 deg. @ 25C
Angular Accuracy Over Temperature	<0.9 deg. @ -40C to 125C	<0.9 deg. @ -40C to 125C
Shaft Speed	100 RPM max. continuous	15,000 RPM max. continuous
Acceleration	10,000 rad/sec²	250,000 rad/sec²
Vibration	20G. 5Hz to 2kHz	20G. 5Hz to 2kHz
Shaft Torque	0.5 ± 0.2 in. oz. (**D** - torque option) 0.3 in. oz. max. (**N** - torque option)	0.05 in. oz. max.
Shaft Loading	2 lbs. max. dynamic* 20 lbs. max. static	1 lb. max.
Bearing Life	> 1,000,000 revolutions	$L10 = (18.3/Fr)^3$? Where $L10$ = bearing life in millions of revs, and Fr = radial shaft loading in pounds
Weight	0.46 oz.	0.37 oz.
Shaft Runout	0.0015 T.I.R. max.	0.0015 T.I.R. max.

* When a pulley, gear, or friction wheel drives the shaft, the Ball Bearing option is recommended instead of the Sleeve Bushing. The c hip that decodes position uses sampled data; note that there will be fewer readings per revolution as the speed increases. The formula f or number of readings per revolution is given by:

US DIGITAL
1400 NE 136th Avenue
Vancouver, Washington 98684, USA
info@usdigital.com
www.usdigital.com
Local: 360.260.2468
Toll-free: 800.736.0194

n = (60 / (rpm * 96 usec))

? only valid with negligible axial shaft loading

Environmental

Parameter	Value
Operating Temperature	-40C to +125C
Storage Temperature	-55C to +125C
ESD	± 2 kV max.
Humidity Non-condensing	5% to 85%

Mounting

Parameter	Value
Hole Diameter	0.375" +0.005 / -0.0
Panel Thickness	0.125" max.
Panel Nut Max. Torque	20 in.-lbs.

Materials

Parameter	Dimension
Shaft	Stainless
Bushing	Brass

Magnetic Field Crosstalk

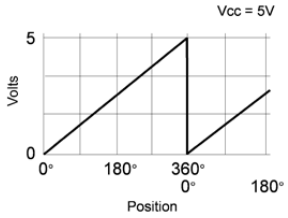
The **MA3** absolute encoder contains a small internal magnet, mounted on the end of the shaft that generates a weak magnetic field extending outside the housing of each encoder. If two **MA3** units are to be installed closer than 1 inch apart (measured between the center of both shafts), a magnetic shield, such as a small steel plate should be installed in between to prevent one encoder from causing small changes in reported position through magnetic field cross-talk.

Electrical

Parameter	Min.	Typ.	Max.	Units
Power Supply	4.5	5.0	5.5	Volts
Supply Current	-	16	20	mA
Power-up Time	-	-	50	mS

US DIGITAL
1400 NE 136th Avenue
Vancouver, Washington 98684, USA
info@usdigital.com
www.usdigital.com
Local: 360.260.2468
Toll-free: 800.736.0194

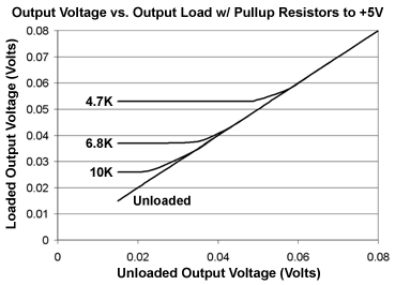
Analog Output Operation



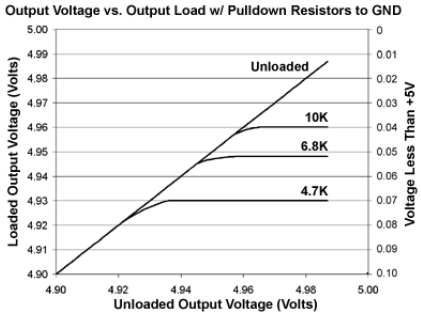
Analog output is only available in 10-bit resolution. The analog output voltage is ratiometric to the power supply voltage and will typically swing within 15 millivolts of the power supply rails with no output load. This non-linearity near the rails increases with increasing output loads. For this reason, the output load impedance should be $\geq$ 4.7kΩ and less than 100pF. The graphs below show the typical output levels for various output loads when powered by a 5V supply.

Parameter	Min.	Typ.	Max.	Units
Position Sampling Rate	2.35	2.61	2.87	kHz
Propagation Delay	-	-	384	?S
Analog Output Voltage Maximum	-	4.987	-	Volts*
Analog Output Voltage Minimum	-	0.015	-	Volts*
Output Short Circuit Sink Current	-	32	50	mA**
Output Short Circuit Source Current	-	36	66	mA**
Output Noise	160	220	490	µ Vrms**
Output Transition Noise	-	0.06	-	Degrees RMS***

* With no output load. See graphs below.
** Continuous short to +5V or ground will not damage the **MA3**.
*** Transition noise is defined as the jitter in the transition between two adjacent position steps.



US DIGITAL
1400 NE 136th Avenue
Vancouver, Washington 98684, USA
info@usdigital.com
www.usdigital.com
Local: 360.260.2468
Toll-free: 800.736.0194



PWM Output Operation

The magnetic sensor chip in the **MA3** has an on-chip RC oscillator which is factory trimmed to 5% accuracy at room temperature (10% over full temperature range). This tolerance influences the sampling rate and the pulse width of the PWM output. If only the PWM pulse width t on is used to measure the angle, the resulting value also has this timing tolerance. However, this tolerance can be cancelled by measuring both ton and toff and calculating the angle from the duty cycle. Angular accuracy including non-linearity is within 0.5 deg. at 25C, but may increase to 0.9 deg. at high temperatures.

Parameter	Min.	Typ.	Max.	Units
PWM Frequency (-40C to 125C)				
10-bit	0.877	0.975	1.072	kHz
12-bit	220	244	268	Hz
Minimum Pulse Width				
10-bit	0.95	1.00	1.05	?S
12-bit	0.95	1.00	1.05	?S
Maximum Pulse Width				
10-bit	974	1025	1076	?S
12-bit	3892	4097	4302	?S
Internal Sampling Rate				
10-bit	9.38	10.42	11.46	kHz
12-bit	2.35	2.61	2.87	kHz
Propagation				
10-bit	-	-	48	?S
12-bit	-	-	384	?S
High Level Output Voltage (V OH: @4mA Source)	Vcc -0.5	-	-	V*
Low Level Output Voltage (V OL: @4mA Sink)	-	-	0.4	V*

* Continuous short to +5V or ground will not damage the **MA3**.

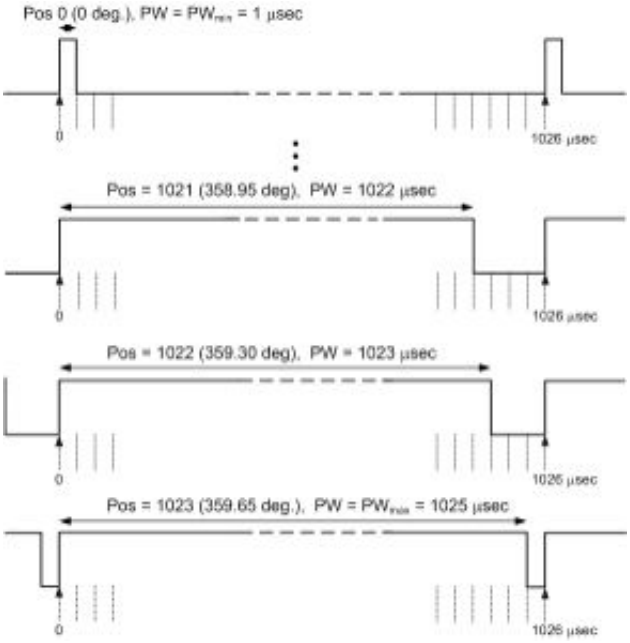
10-bit PWM:

x = ((ton * 1026) / (t on+ t off)) -1

US DIGITAL 1400 NE 136th Avenue Vancouver, Washington 98684, USA info@usdigital.com www.usdigital.com Local: 360.260.2468 Toll-free: 800.736.0194

If x <= 1022, then Position = x

If x = 1024, then Position = 1023



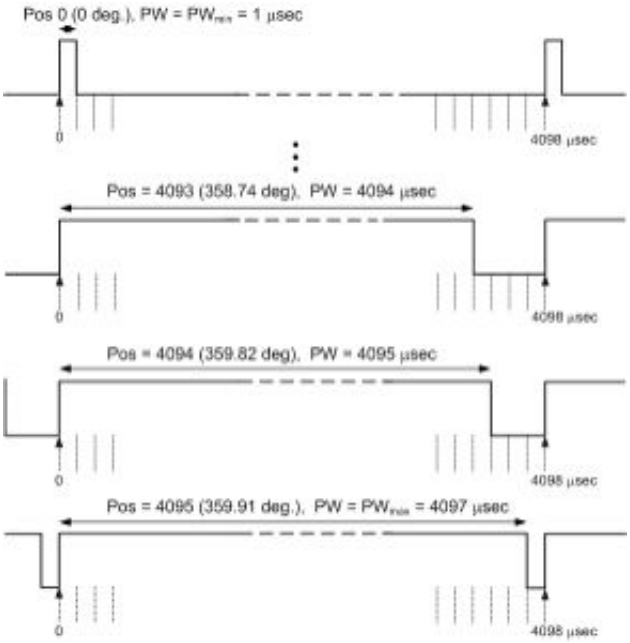
12-bit PWM:

x = ((ton * 4098) / (t on+ t off)) -1

If x <= 4094, then Position = x

If x = 4096, then Position = 4095

US DIGITAL 1400 NE 136th Avenue
Vancouver, Washington 98684, USA
info@usdigital.com
www.usdigital.com
Local: 360.260.2468
Toll-free: 800.736.0194



Pin-outs

Analog Output (MA3-A):

Pin	Name	Description
1	5	+5VDC power
2	A	Analog output
3	G	Ground

PWM Output (MA3-P10, MA3-P12):

Pin	Name	Description
1	5	+5VDC power
2	A	PWM output
3	G	Ground

US DIGITAL 1400 NE 136th Avenue Vancouver, Washington 98684, USA
info@usdigital.com www.usdigital.com
Local: 360.260.2468 Toll-free: 800.736.0194

MA3 Miniature Absolute Magnetic Shaft Encoder

RoHS COMPLIANT

Ordering Information

MA3 - [ ] - [ ] - [ ]

Interface	Shaft Diameter	Torque
A10 =*10-Bit Analog*	125 =*1/8"*	D =*Sleeve Bushing, Most Drag*
P10 =*10-Bit PWM*	236 =*6mm*	N =*Sleeve Bushing, Somewhat Lighter Drag*
P12 =*12-Bit PWM*	250 =*1/4"*	B =*Ball Bearing, Free Spinning (Least Drag)*

Rules

- Torque must be something other than B when Shaft Diameter is 125

Notes

- Cables and connectors are not included and must be ordered separately.
- US Digital warrants its products against defects in materials and workmanship for two years. See complete warranty for details.

Base Pricing

Quantity	Price
1	$39.60
10	$34.69
50	$30.56
100	$26.28

- Add 17% per unit for **Interface** of 12-Bit PWM
- Add $1.00 per unit for **Shaft Diameter** of 6mm
- Add $5.80 per unit for **Torque** of Ball Bearing, Free Spinning (Least Drag)

US DIGITAL
1400 NE 136th Avenue
Vancouver, Washington 98684, USA
info@usdigital.com
www.usdigital.com
Local: 360.260.2468
Toll-free: 800.736.0194