Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorarbeit

Olaf Rempel

Entwicklung einer inertialen Lageregelung für einen Quadrocopter auf ARM7 Basis

Fakultät Technik und Informatik
Studiendepartment Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Olaf Rempel

Entwicklung einer inertialen Lageregelung für einen Quadrocopter auf ARM7 Basis

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung
im Studiengang Technische Informatik
am Studiendepartment Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer : Prof. Dr. rer. nat. Gunter Klemke
Zweitgutachter : Prof. Dr. Franz Korf

Abgegeben am 27. Mai 2008

Olaf Rempel

Thema der Bachelorarbeit

Entwicklung einer inertialen Lageregelung für einen Quadrocopter auf ARM7 Basis

Stichworte

ARM7TDMI, MEMS, Gyroskope, Beschleunigungssensor, bürstenlose Motoren

Kurzzusammenfassung

Im Rahmen dieser Arbeit wird ein Hard- und Softwaresystem zur Stabilisierung eines Quadrocopters entwickelt. Zur Erfassung der Fluglage werden Gyroskope und Beschleunigungssensoren verwendet, deren Messwerte von einem ARM7 Microcontroller ausgewertet werden. Das Regelungssystem ist als Softwarekomponente auf diesem Microcontroller implementiert. Zum Abschluss der Arbeit wird ein Quadrocopter mit dieser Lageregelung aufgebaut und das Flugverhalten getestet.

Olaf Rempel

Title of the paper

Development of an ARM7 based inertial attitude control system for quadrocopters

Keywords

ARM7TDMI, MEMS, Gyroscopes, Accelerometers, Brushless Motors

Abstract

This paper describes the development of an embedded system for stabiliziation of a quadrocopter. The flight attitude is sensed by gyroscopes and acceleration sensors, the measured values are processed by an ARM7 microcontroller. The control system is implemented as a software component on this microcontroller. This paper finishes with the construction of a quadrocopter which is equipped with this stabilization system.

Danksagung

Auf diesem Weg möchte ich mich bei einer Reihe von Personen
für ihre Unterstützung bedanken:

- Prof. Gunter Klemke für die gute Betreuung dieser Arbeit
- Boris Böttcher und Sebastian Gregor, für ihre moralische Unterstützung und dass sie mich, wenn nötig, auf den Boden der Tatsachen zurückgeholt haben
- Meiner Schwester Anika und ihrem Mann Christian, für Verbessungsvorschläge und das Korrekturlesen dieser Arbeit

Und natürlich meinen Eltern: ohne euch hätte ich nie studieren können.

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einführung

Im Rettungswesen können detailierte Aufnahmen der Unglücksstelle einen schnellen Überblick der Situation liefern und damit die Koordinierung von Einsatzkräften erleichtern. Für diese Art von Aufklärung können Hubschrauber oder Flugzeuge angefordert werden, wodurch bis zur Ankunft wertvolle Zeit verstreicht. Diese Aufnahmen können aber auch von kleinen UAVs (Unmanned Aerial Vehicle), also unbemannten Luftfahrzeugen, erstellt werden.

Als Technologie für diese aufklärenden UAVs gibt es verschiedene Ansätze. Kleine Flugzeuge können ein Kamerasystem mit Übertragungstechnik gut im Rumpf transportieren, benötigen aber eine Start- und Landefläche. Für die Beobachtung eines festen Punktes müssen sie prinzipbedingt über diesem kreisen, was eine laufende Veränderung des Blickwinkels zur Folge hat. Dadurch wird die Steuerung und auch die Auswertung der Aufnahmen erschwert. Modellhubschrauber können senkrecht starten und benötigen somit keine größeren Startflächen. Im Schwebeflug können sie auf einer Position gehalten werden und dabei längere Aufnahmen aus einem Blickwinkel durchführen. Aufgrund der hohen Geschwindigkeit der Hauptrotorspitzen birgt der Einsatz eines Hubschraubers ein hohes Verletzungsrisiko. Bei einem Quadrocopter wird der Auftrieb von vier einzelnen Rotoren erzeugt, die kleiner als bei einem Hubschrauber ausfallen. Die Propeller haben eine geringe Masse und drehen mit geringer Geschwindigkeit, wodurch das Verletzungsrisiko niedrig ist.

Ein Quadrocopter kann, wie ein Hubschrauber, Bewegungen in sechs Freiheitsgraden ausführen. Jede Bewegung lässt sich in drei geradlinige (translatorische) Beschleunigungen und drei Rotationsgeschwindigkeiten zerlegen. Zur Erfassung der momentanen Beschleunigungen und Rotationsgeschwindigkeiten werden inertiale Sensoren verwendet, die also auf dem Prinzip der Massenträgheit arbeiten. Die Lageregelung verarbeitet die erfassten Sensordaten und steuert die vier Rotoren des Quadrocopters an. Dadurch werden Störungen ausgeglichen und der Flug stabilisiert.

Im Rahmen dieser Bachelorarbeit wird eine erweiterbare Lageregelung für einen Quadrocopter als Embedded System entwickelt. Das System umfasst die benötigte Sensorik, sowie eine Lageregelung, die als Software eines Microcontrollers implementiert wird. Das System sorgt für eine Stabilisierung der Fluglage, ein vollständig autonomer Flug des Quadrocopters ist nicht vorgesehen.

1.1 Gliederung

Diese Arbeit lässt sich in drei große Abschnitte aufteilen: Kapitel 2 behandelt technische Grundlagen eines Quadrocopters die für das Verständnis der weiteren Kapitel wichtig sind. Im folgenden Kapitel 3 werden beispielhaft zwei verbreitete Quadrocoptersysteme vorgestellt und ihre technischen Daten aufgeführt. In Kapitel 4 wird anhand der Problemstellung ein Konzept für eine Quadrocopterplattform erstellt, welches in den Kapiteln 5 und 6 konkretisiert und implementiert wird. Die letzten beiden Kapitel 7 und 8 fassen die Ergebnisse dieser Arbeit zusammen und machen Vorschläge für eine Verbesserung des Systems.

2 Technische Grundlagen

In diesem Kapitel werden zunächst der Aufbau und das Antriebskonzept eines Quadrocopters dargestellt. Anschließend wird die Ansteuerung von bürstenlosen Gleichstrommotoren erklärt, welche die benötigte Leistung zum Antrieb eines Quadrocopters liefern. Anschließend wird im dritten Unterkapitel auf die Funktionsweise der Sensoren eingegangen, die für eine Lageregelung eines Quadrocopters erforderlich sind.

2.1 Prinzip eines Quadrocopters

Der Quadrocopter (auch Quadrotor) gehört zu der Familie der Hubschrauber, er kann also senkrecht starten, schweben und landen. Der zum Schwebeflug benötigte Auftrieb wird durch vier nach unten wirkende Propeller erzeugt. Durch Neigung des kompletten Chassis wird Vortrieb erzeugt.

Die vier Propeller werden von einzelnen Motoren angetrieben, somit kann die Schubleistung über Drehmoment/Drehzahl der Motoren verändert werden. Die bei Hubschraubern mit nur einem Hauptrotor verwendete kollektive Blattverstellung (Schubleistung wird über die Propellersteigung verändert) ist somit nicht erforderlich. Hierdurch ergibt sich im Vergleich zu Hubschraubern ein sehr einfacher mechanischer Aufbau.

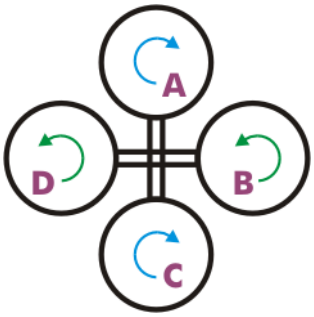


Abbildung 2.1: Rotordrehrichtungen eines Quadrocopters(Quelle: [16])

Je zwei Propeller haben die gleiche Drehrichtung, wodurch sich bei gleicher Drehzahl aller Propeller das Drehmoment des gesamten Systems aufhebt. Durch Verändern des Drehzahlverhältnisses zweier gegenüberliegenden Motoren wird das Schubgleichgewicht auf dieser Achse verändert und das gesamte System “kippt” in Richtung des langsameren Propellers (vgl. Abbildung 2.1):

- Rotor A $\Leftrightarrow$ Rotor C: Das System “nickt” nach vorne/hinten
- Rotor B $\Leftrightarrow$ Rotor D: Das System “rollt” nach links/rechts

Wenn die Schubleistung der rechtsdrehenden Rotoren (A + C) gegenüber den linksdrehenden (B + D) verändert wird, herrscht kein Drehmomentausgleich mehr und das System “giert” um seine z-Achse. Bei Hubschraubern mit einem Hauptrotor wird dieser Drehmomentausgleich über einen Heckrotor hergestellt.

Durch den Direktantrieb der Propeller wird ein hohes Drehmoment benötigt, daher kommen beim Aufbau von Quadrocoptern im Allgemeinen bürstenlose Gleichstrommotoren zum Einsatz. Eine weitere Beschreibung dieser Motoren und deren Ansteuerung folgt in Kapitel 2.2. Zur Stabilisierung eines Quadrocopters müssen die aktuellen Lagewinkel der Nick-, Roll- und Gierachse über Sensoren erfasst werden. In Kapitel 2.3 wird auf die benötigte Sensorik genauer eingegangen.

Zur Stromversorgung der Motoren werden Lithium-Polymer Zellen eingesetzt, da sie einen sehr hohen Strom liefern können und im Vergleich zu Nickel-Metall-Hydrid Zellen über eine höhere Energiedichte verfügen (LiPo: 140-180Wh/kg, NiMh: ca. 80 Wh/kg) [14]. Aufgrund des geringen Gewichtes der LiPo-Akkus ist es möglich eine hohe Nutzlast zu transportieren, die je nach Bauweise auch über dem Eigengewicht des Quadrocopters liegen kann.

Die Aufteilung der benötigten Schubleistung auf vier Rotoren ermöglicht im Vergleich zu Modellhubschraubern eine kleinere Dimensionierung der Propeller. Mit der kleineren Propellermasse und der geringeren Geschwindigkeit der Propellerspitzen besteht ein erheblich geringeres Verletzungsrisiko. Somit können Quadrocopter ohne Probleme im Innenbereich betrieben werden.

2.2 Ansteuerung von bürstenlosen Motoren

Im direkten Vergleich zu normalen Gleichstrommotoren haben bürstenlose Motoren ein höheres Drehmoment und somit höheren Wirkungsgrad. Außerdem kommt es durch die fehlenden mechanischen Kontakte zu einem geringeren Verschleiß. Die Ansteuerung von bürstenlosen Motoren ist aufwendiger, da das Weiterschalten der Rotorwicklungen (Kommutierung) von einer Elektronik nachgebildet werden muss.

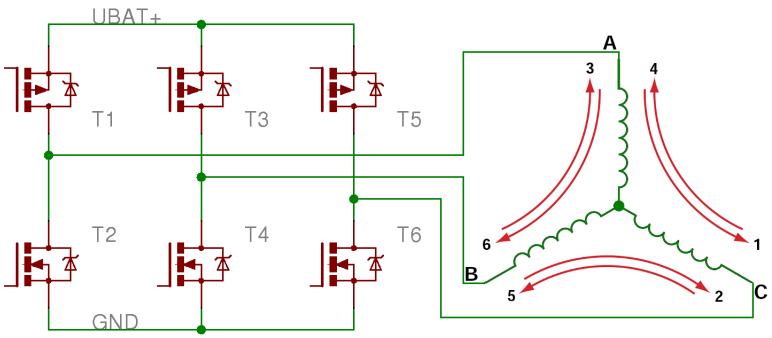


Abbildung 2.2: Ansteuerung eines bürstenlosen Gleichstrommotors

Der in Abbildung 2.2 dargestellte Motor enthält drei Statorwicklungen in Sternschaltung, die mit dem Leistungsteil einer Ansteuerung verbunden sind. Um den Motor in Rotation zu versetzen, wird eine Umdrehung in sechs einzelne Schritte unterteilt. Zu jedem Schritt sind jeweils zwei Wicklungen aktiv:

Schritt	Stromfluss
1	T1 $\rightarrow$ A $\rightarrow$ C $\rightarrow$ T6
2	T3 $\rightarrow$ B $\rightarrow$ C $\rightarrow$ T6
3	T3 $\rightarrow$ B $\rightarrow$ A $\rightarrow$ T2
4	T5 $\rightarrow$ C $\rightarrow$ A $\rightarrow$ T2
5	T5 $\rightarrow$ C $\rightarrow$ B $\rightarrow$ T4
6	T1 $\rightarrow$ A $\rightarrow$ B $\rightarrow$ T4

Mit jedem Schritt dreht sich das magnetische Feld des Stators um 60°. Da ein bürstenloser Gleichstrommotor in der Regel mehrere Permanentmagneten enthält, unterscheidet sich die “mechanische” Umdrehung des Rotors von der “elektrischen” Umdrehung des magnetischen Feldes. Ein Motor mit sieben magnetischen Polpaaren benötigt auch sieben “elektrische” Umdrehungen um eine vollständige 360° Drehung des Rotors zu erzeugen.

Der Ansteuerung muss die momentane Ausrichtung des Rotors bekannt sein, um die Weiterschaltung der Schritte dazu synchron alle 60° ausführen zu können. Eine Drehzahlsteuerung erfolgt nicht durch die Vorgabe einer Kommutierungsfrequenz, sondern wird über eine Pulsweitenmodulation (PWM) des Motorstroms erreicht. Das Drehmoment erhöht sich proportional zum Strom und lässt den Rotor die 60° Schritte schneller ausführen.

Bei sensorgeführten, bürstenlosen Motoren wird die Rotorposition durch mehrere Hallsensoren detektiert und der Ansteuerung zurückgemeldet. Im Flugmodellbau haben sich aufgrund der Kosten und der zusätzlichen Verdrahtung sensorlose Motoren durchgesetzt. Bei dieser

Motorenart wird die Position des Rotors anhand der induzierten Spannung in den Wicklungen erkannt.

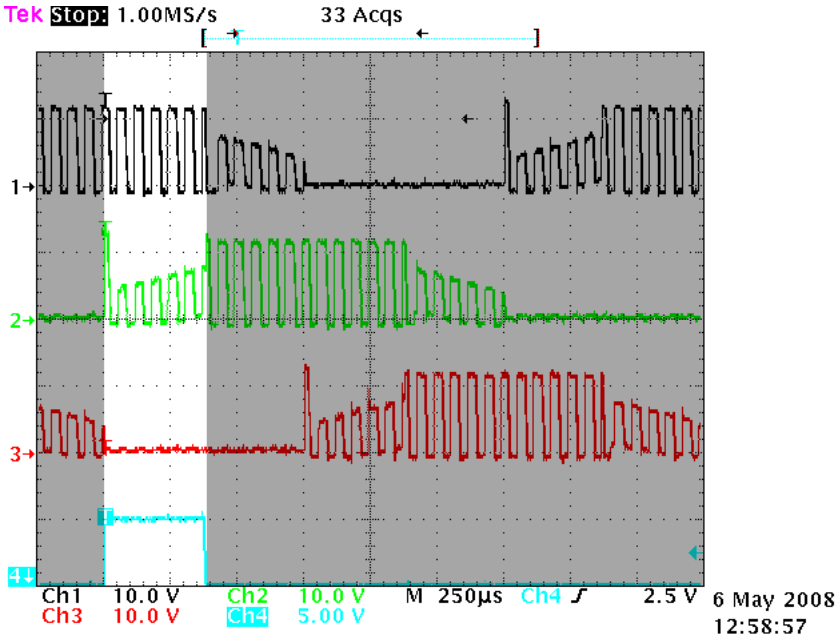


Abbildung 2.3: Spannungsverlauf der Motorphasen

In Abbildung 2.3 sind die Spannungen der drei Motorphasen A, B, C, sowie ein Synchronsignal während der sechs Schritte einer elektrischen Umdrehung zu erkennen. Die Motordrehzahl wird dabei über eine 16kHz PWM Ansteuerung gestellt. Im hervorgehobenen, ersten Schritt liegt der Motorstrom an den Anschlüssen A (Kanal 1, 12V PWM) und C (Kanal 3, 0V) an. Anhand der Nulldurchgänge induzierten Spannung am Anschluss B (Kanal 2) kann die Rotorposition erkannt werden und eine Kommutierung erfolgen.

Beim Starten des Motors und bei sehr niedriger Drehzahl ist die induzierte Spannung zu gering um ausgewertet werden zu können. Daher wird beim Anlaufen von sensorlosen Motoren eine Taktfolge vorgegeben, bis eine ausreichende Spannung induziert wird und sich die Ansteuerung auf den Motor synchronisieren kann.

Die für die Lageregelung eines Quadrocopters benötigten Motorregler müssen die Drehzahl der Motoren sehr schnell ändern können. Gängige Motorregler für bürstenlose Motoren aus dem Modellbaubereich erlauben allerdings nur alle 15-20ms eine neue Sollwertvorgabe und sind somit als Stellglieder für eine Lageregelung nur schlecht geeignet.

2.3 Benötigte Sensorik

Zur Erfassung der momentanen Lagewinkel eines Quadrocopters werden Gyroskope benötigt. Ein Gyroskop misst die Drehwinkelgeschwindigkeit einer Achse, das Integral dieses Signals entspricht dem Lagewinkel dieser Achse. Über drei senkrecht zueinander angeordnete Gyroskope können alle auftretenden Rotationen erfasst werden.

Durch die Verbindung von mechanischen Komponenten und elektronischen Schaltungen auf einem Chip (MEMS, Micro-Electro-Mechanical System) ist die Herstellung von sehr kompakten Gyroskopen möglich. Diese Sensoren arbeiten nicht mit einer sich schnell drehenden Masse, wie es bei mechanischen Gyroskopen (Kreiseln) üblich ist, sondern messen die Auslenkung einer vibrierenden Masse durch die Corioliskraft.

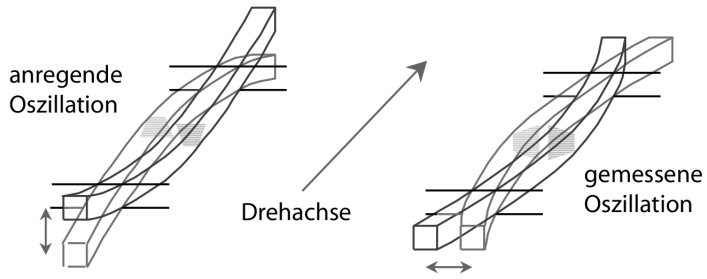


Abbildung 2.4: Aufbau eines MEMS Gyroskopes (Quelle: [24])

Das sensitive Element des Sensors ist ein Piezokristallstab, der durch eine elektrische Spannung zum Schwingen angeregt wird. Bei einer Drehung des Sensors wird durch die Corioliskraft eine zusätzliche Schwingung senkrecht zur Rotationsachse im Kristall erzeugt (Abbildung 2.4). Diese Schwingung kann elektrisch gemessen werden, wobei die Signalamplitude proportional zur Drehwinkelgeschwindigkeit ist.

Neben dem Aufbau mit vibrierenden Kristallstab gibt es noch weitere Bauformen für MEMS-Gyroskope. Allen gemein ist die Auslenkung einer vibrierenden Masse durch die Corioliskraft. Um die Frequenzanteile dieser internen Schwingungen zu unterdrücken, wird das Ausgangssignal mit einem Tiefpass gefiltert.

Über die Signale der Gyroskope lassen sich die momentanen Lagewinkel eines Quadrocopters durch Integration errechnen. Allerdings addieren sich mit der Zeit Fehler auf, womit die ermittelten Lagewinkel nicht mehr den realen Winkeln entsprechen. Mit Hilfe von Beschleunigungssensoren, die die Erdanziehung messen, kann ein ruhig schwebender Quadrocopter die realen Nick- und Rollwinkel ermitteln.

Beschleunigungssensoren werden mittlerweile als MEMS-Bausteine gefertigt, wobei fertige Bausteine mit bis zu drei internen Sensoren erhältlich sind. Als sensitives Element eines

Sensors wirkt dabei eine kleine Testmasse, die aufgrund der Massenträgheit bei Beschleunigungen gegenüber dem Sensorgehäuse ausgelenkt wird.

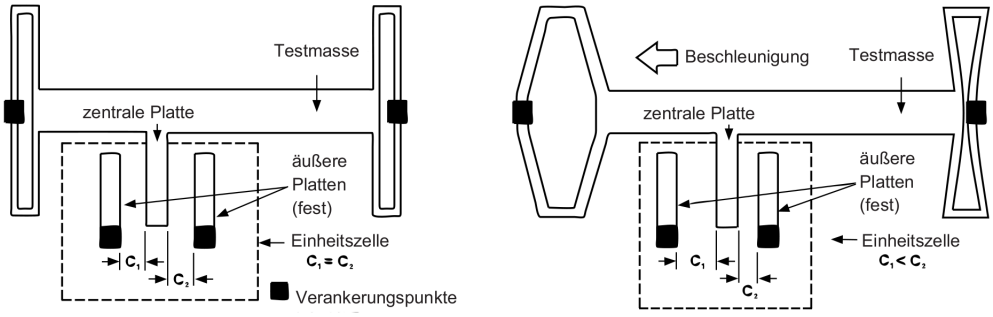


Abbildung 2.5: Aufbau eines MEMS Beschleunigungssensor (Quelle: [24])

Die Testmasse ist über zwei Federn mit dem Gehäuse des Sensors verbunden und kann sich nur in einer Achse frei bewegen. Die Wegveränderung der Testmasse verhält sich dabei proportional zur Stärke der einwirkenden Beschleunigung. Zusammen mit den Bezugspunkten bildet die Testmasse zwei Kondensatoren, deren Kapazität sich mit dem Weg verändert. Diese Kapazitätsänderungen können elektronisch erfasst und weiterverarbeitet werden (Abbildung 2.5).

3 Verwandte Systeme

Die Verbreitung von Quadrocopter Systemen ist in den letzten Jahren stark angestiegen. Neben Forschungsprojekten gibt es mittlerweile auch eine Vielzahl von kommerziellen Lösungen. Aber auch im Hobby- und Amateurbereich werden diese vielseitigen Systeme aufgebaut. Im folgenden Kapitel werden exemplarisch zwei Systeme vorgestellt.

3.1 Microdrones

Abbildung 3.1: MD4-200 (Quelle [1])

Die MD4-200 der Firma microdrones ist ein Quadrocopter mit 54cm Achsabstand und 37cm Propellerdurchmesser (Abbildung 3.1). Das Abfluggewicht mit Farbkamera und Funkübertragungssystem beträgt 680g. Dieses niedrige Gewicht wurde durch konsequenten Leichtbau aus Kohlenstofffaserverstärktem Kunststoff (CFK) erreicht. Zusätzlich kann eine maximale Nutzlast von 200g transportiert werden.

Zum Antrieb werden vier bürstenlose Gleichstrommotoren der Firma Plettenberg mit sensorgeführter Kommutierung eingesetzt. Die Motorregler werden direkt unter die Motoren montiert und bilden zusammen eine Einheit. Für die Kommunikation zwischen Motorreglern und Flugsteuerung wird ein CAN-Bus verwendet.

Neben den benötigten Gyroskopen und Beschleunigungssensoren kommen eine Reihe zusätzlicher Sensoren (barometrischer Höhensensor, Temperatursensor und elektronischer Kompass) zum Einsatz. Die Werte dieser Sensoren werden zusammen mit dem Signal eines GPS-Empfängers von einer 32bit CPU zu einer Position und Lage verrechnet. Eine zweite 32bit CPU ist für Navigation und Flugsteuerung zuständig. Mit Hilfe der GPS Navigation kann ein Position exakt gehalten werden, oder eine Reihe von Wegpunkten abgeflogen werden.

3.2 Mikrokopter

Das Mikrokopter Projekt [2] ist ein Selbstbau Quadrocopter auf Basis einer OpenSource Fluglageregelung. Neben dem Schwebeflug ermöglicht diese Regelung auch sportlichere Flugverhalten.

Abbildung 3.2: Mikrokopter mit Landegestell und Kamerahalterung (Quelle [3])

Der Aufbau, das Material und die Größe des Chassis (oder auch Rahmens) sind sehr unterschiedlich. Neben einfachen Chassis aus Aluminiumprofilen werden auch fertige Rahmensysteme aus GFK/CFGK angefertigt. Die Rahmengröße variert dabei zwischen 30-54cm. Abbildung 3.2 zeigt einen Mikrokopter aus Verbundmaterial mit 40cm Motorachsabstand. Aufgrund der unterschiedlichen Motorisierung mit bürstenlosen Gleichstrommotoren, sind

auch die möglichen Nutzlasten unterschiedlich. Abfluggewichte von 500-1000g sind mit der Standardmotorisierung erreichbar.

Allen Mikrokoptern ist die Fluglageregelung auf Basis eines 8bit AVR Controllers gemein. Auf dieser Platine sind Gyroskope und Beschleunigungssensoren montiert, zusätzlich kann ein barometrischer Drucksensor zu Höhenregelung bestückt werden. Für die Ansteuerung der Motoren werden speziell entwickelte Motorregler verwendet, die über einen I2C-Bus kommunizieren.

Über ein PC-basiertes System kann die Lageregelung an den verwendeten Rahmen und an das gewünschte Flugverhalten angepasst werden.

4 Analyse

Die vorgestellten Systeme werden von leistungsfähigen Fluglageregelungen stabilisiert. Eine Erweiterung der Systeme ist aber nicht möglich, da die einzelnen Komponenten nicht verändert werden können, oder aufgrund bereits ausgelasteter Resourcen eine Veränderung die Leistungsfähigkeit einschränken würde. Daher wird eine eigene Fluglageregelung entwickelt.

4.1 Konzeption

Die Hardware des im Laufe dieser Arbeit entstehenden Quadrocopters basiert auf dem Mikrokopter Projekt. Als Antrieb kommen vier bürstenlose Gleichstrommotoren zum Einsatz, auf deren Motorachsen Propeller mit 25cm Durchmesser direkt montiert werden. Für die Ansteuerung der Motoren wird ein Nachbau der beim Mikrokopter eingesetzten Motorregler ("blctrl-v1.0") verwendet. Zur Kommunikation mit den Motorreglern dient ein I2C-Bus. Der Nachbau der Motorregler und eine darauf laufende, eigene Firmware wurden nicht im Rahmen dieser Arbeit entwickelt. Zur Stromversorgung der Motorregler und der Elektronik dient ein dreizelliger Lithium-Polymer-Akku mit einer Nennspannung von 11.1V.

Für die Steuerung des Quadrocopters wird eine 35MHz Funkfernsteuerung aus dem Modellbaubereich verwendet. Die übertragenen Signale der Fernsteuerung können am Empfänger (RC-Empfänger) mit wenig Aufwand ausgewertet werden.

Eine Videokamera mit passendem Funkübertragungssystem ist nicht Teil dieser Arbeit. Die Stabilisierung des Quadrocopters durch die Lageregelung darf aber durch eine vergleichbare Nutzlast nicht beeinträchtigt werden.

Die Hardware der Lageregelung beinhaltet Sensoren für die Erfassung der Drehwinkelgeschwindigkeiten und Beschleunigungen der drei Achsen. Zusätzliche Sensoren für eine Höhenregelung und eine Stabilisierung der Gierachse sind auf der Platine der Lageregelung nicht vorgesehen. Der eingesetzte Controller muss aber eine Erweiterung der Hardware durch weitere Sensoren oder Aktoren ermöglichen. Als Schnittstelle zu den Motorreglern wird ein I2C-Bus benötigt. Für die Kommunikation mit einem Telemetriesystem werden serielle Schnittstellen nach RS232 und USB Standard verwendet. Eine Telemetrieverbindung

wird zunächst durch ein kabelgebundenes System hergestellt, der Einsatz einer Funkübertragung ist nicht vorgesehen.

Die Software der Lageregelung muss den Quadrocopter im Schwebeflug stabilisieren. Weiterführende Konzepte, die z. B. Kunstflug ermöglichen, sind nicht Teil dieser Arbeit. Dennoch muss der eingesetzte Controller leistungsfähig genug sein, um eine solche Erweiterung zu einem späteren Zeitpunkt zu ermöglichen. Neben der Hard- und Software der Fluglageregelung wird eine PC-basierte Telemetrieapplikation zur Überwachung und Einstellung von Parametern benötigt.

5 Design

Im Designkapitel werden zunächst die benötigten Bauteile für die Hardware der Fluglageregelung bestimmt und zu Baugruppen zusammengeführt. Aus diesen Baugruppen wird schließlich ein konkreter Schaltplan erstellt und in ein Platinenlayout umgesetzt. Aufbauend auf das Hardwaredesign werden im zweiten Unterkapitel die Softwarekomponenten der Fluglageregelungen aufgeteilt und ihre Funktionen erläutert. Im dritten Unterkapitel wird schließlich das Design eines erweiterbaren Telemetriesystems zur Überwachung und Parameterisierung der Fluglageregelung erstellt.

5.1 Hardware

Das Hardwaredesign (Abbildung 5.1) der Fluglageregelung umfasst die Sensorik, um Beschleunigungen und Drehwinkel der einzelnen Achsen erfassen zu können, sowie einen leistungsfähigen Mikrocontroller (MCU) zur Auswertung der gemessenen Signale. Außerdem sind Schnittstellen für einen RC-Empfänger, ein I2C-Bus für die Motorregler und eine Anbindung für das Telemetriesystem auf der Platine erforderlich.

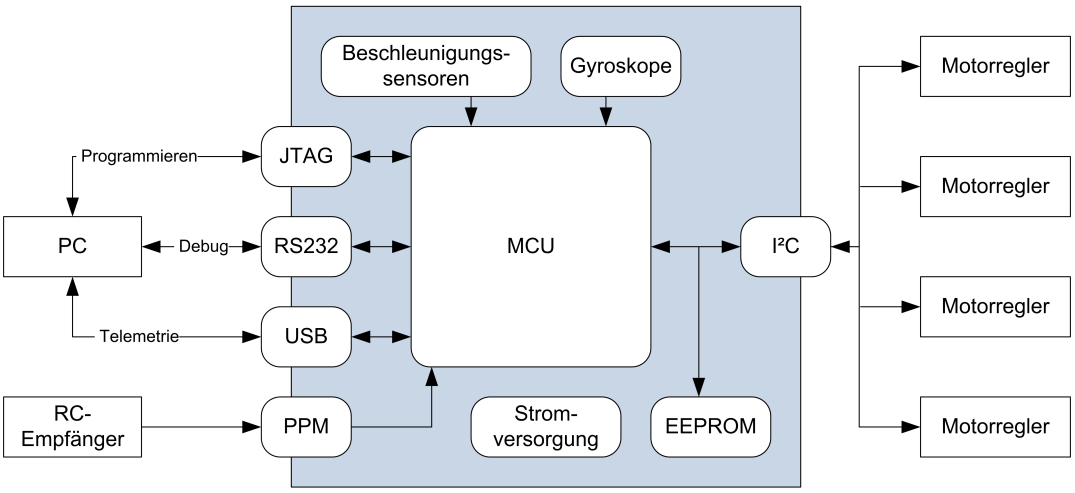


Abbildung 5.1: Übersicht Hardware Design

5.1.1 Controller

Wie in Kapitel 3 beschrieben, ist ein 8bit Atmel AVR Microcontroller für die reine Fluglageregelung ausreichend, jedoch ist durch die eingeschänkte Rechenleistung eine Erweiterbarkeit der Software nicht gegeben. Mit der 32bit ARM Architektur existiert eine leistungsfähige Alternative, für die eine große Anzahl von Open Source Programmiertools wie Compiler, Debugger und Programmieradaptern frei verfügbar ist. Daher fiel die Wahl auf einen 32bit ARM7TDMI [19] basierten Mikrocontroller von Atmel.

Der AT91SAM7S256 [22] stellt neben 256kByte Flash-Speicher und 64kByte SRAM auch die für Fluglageregelung benötigte Peripherie on-chip zur Verfügung. Darunter ein 8 Kanal Analog-Digital-Wandler (ADC) mit 10Bit Auflösung für die Sensorik, ein integrierter I2C-Controller zur Anbindung der Motorregler, sowie ein USB 2.0 Device Controller für eine schnelle Telemetrieverbindung (Abbildung 5.2).

Der Controller benötigt nur eine einzige 3.3V Versorgungsspannung, wobei der eigentliche CPU Kern von einem internen 1.8V Spannungsregler versorgt wird. Alle IO-Pins sind 5V-tolerant, d.h. ein extern zugeführter Pegel von 5V schadet oder zerstört den Controller nicht. Das erleichtert das weitere Design, da auf eine Pegelanpassung verzichtet werden kann.

Als Taktquelle für den Controller dient ein 18.432MHz Grundtonquarz, dessen Signal durch die on-Chip PLL auf 48MHz vervielfacht wird. Die maximale Arbeitsfrequenz des ARM Cores beträgt zwar 55MHz, für den Betrieb des USB Device Ports wird jedoch eine Frequenz von exakt 48MHz benötigt.

$$MCK = \frac{18.432MHz * 125}{24} * \frac{1}{2} = 48MHz$$

Über die integrierte JTAG (Joint Test Action Group) [15] Schnittstelle kann der Controller im verbauten Zustand sowohl programmiert als auch komfortabel debugged werden.

Die 64pol LQFP Bauweise kann von Hand gelötet werden und benötigt keine speziellen Lötverfahren bei der Bestückung.

Dem Controller fehlt ein interner, nicht-flüchtiger Speicher, wie er zum Ablegen von Parametern oder Kalibrierungsdaten benötigt wird. Der interne Flash-Speicher könnte diese Funktion übernehmen, da er in einzeln beschreibbare Bereiche unterteilt ist, allerdings kann dieser Speicher nicht byteweise gelöscht werden. Daher wird zusätzlich eine 32kByte I2C-EEPROM [20] zum Speichern von Parametern auf der Platine untergebracht.

AT91SAM7S512/256/128/64/321/161 Block Diagram

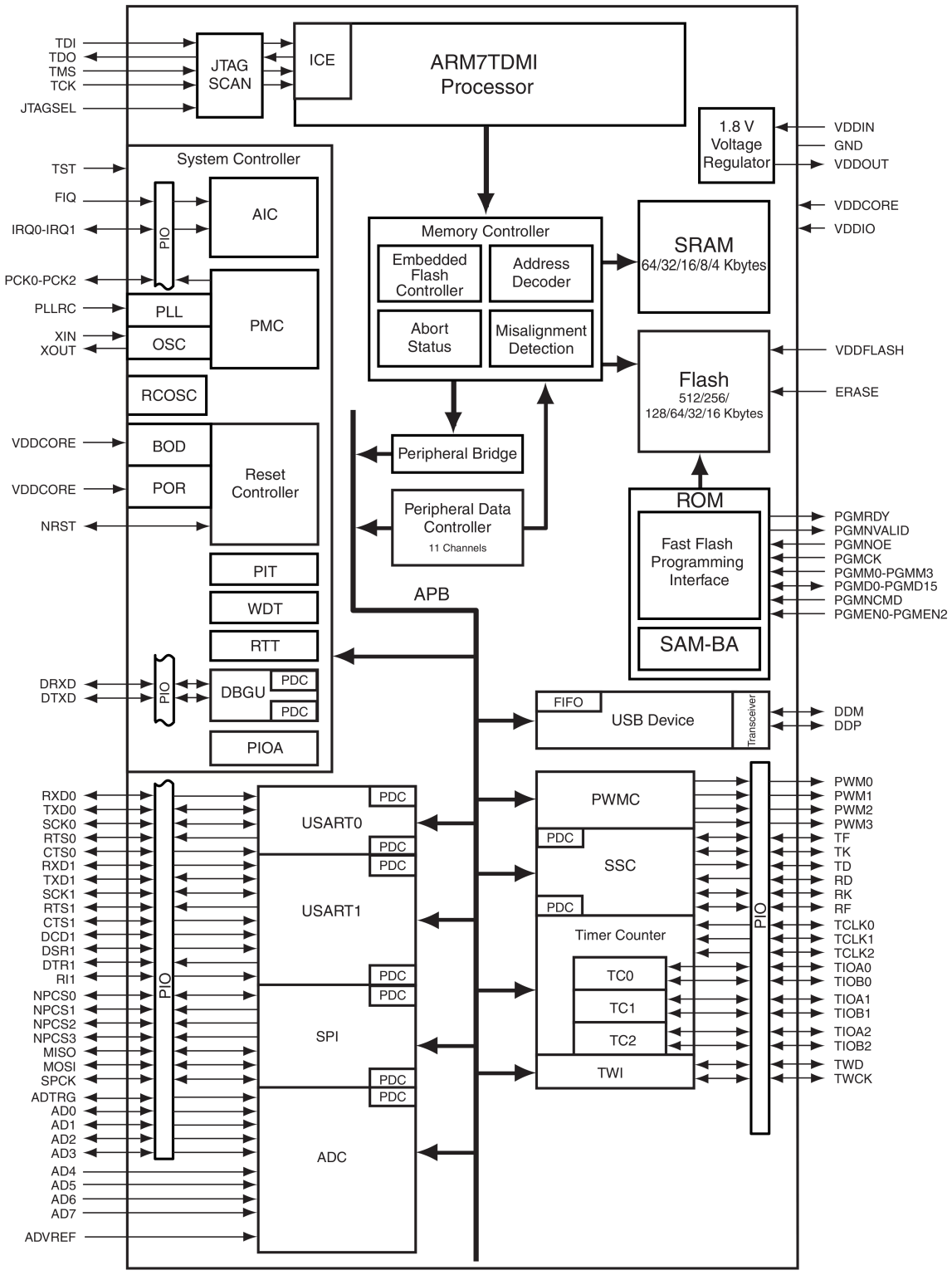


Abbildung 5.2: AT91SAM7S Blockschaltbild

5.1.2 Gyroskope

Zur Berechnung der Lagewinkel benötigt die Flugsteuerung drei Gyroskope, welche die Drehwinkelgeschwindigkeiten der Nick-, Roll- und Gierachsen messen.

Zur Auswahl standen die schon bei anderen Studienarbeiten [26] an der Hochschule für Angewandte Wissenschaften Hamburg eingesetzten Gyroskope vom Typ ADXRS300 [18] des Herstellers Analog Devices, sowie die im Mikrokopter-Projekt [2] eingesetzten ENC03J Gyroskope [27] des Herstellers Murata. Beide Sensoren haben einen Messbereich von $\pm 300°/s$ und geben ein, der Drehwinkelgeschwindigkeit proportionales, analoges Signal aus.

Aufgrund der 7x7mm 32pol CSBGA Gehäuseform des ADXRS300 wird für diese Sensoren jedoch eine Adapterplatine benötigt. Eine direkte Bestückung wäre nur mit speziellen Lötverfahren möglich. Die Wahl fiel daher auf die ENC03J Gyroskope, die aufgrund der bedrahteten Bauform einfacher zu bestücken sind. Zudem betragen die Kosten eines ENC03J nur ein Drittel der eines ADXRS300.

Durch den niedrigen Pegel von 0.67mV pro °/s kann das Ausgangssignal der Gyroskope nicht direkt vom AD-Wandler verarbeitet werden und muss zuvor verstärkt werden. Ein Operationsverstärker, der als aktiver Tiefpass 1. Ordnung beschaltet ist, passt das Sensorsignal an den Eingangsbereich des AD-Wandlers an (Abbildung 5.3). Gleichzeitig filtert er ungewünschte Frequenzanteile heraus, die sich aus dem internen Aufbau des Gyroskop ergeben.

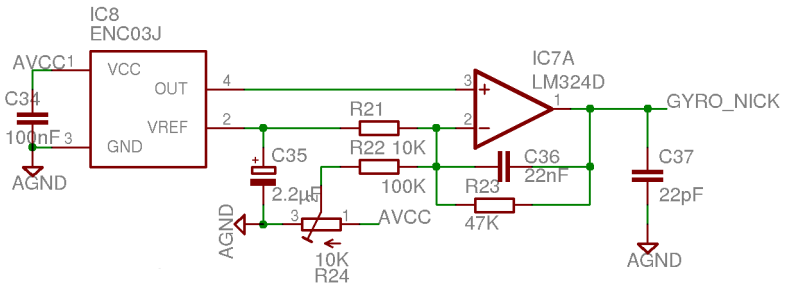


Abbildung 5.3: Teilschaltung Gyroverstärker

Die Grenzfrequenz f_g des Tiefpasses beträgt:

$$f_g = \frac{1}{2\pi * RC} = \frac{1}{2\pi * 47k\Omega * 22nF} = 154Hz$$

Die Gleichspannungsverstärkung A_0 beträgt:

$$A_0 = -\frac{R2}{R1} = -\frac{47k\Omega}{10k\Omega} = -4.7$$

Da der AD-Wandler des Controllers nur positive Werte erfassen kann, muss der Nullpunkt aller Ausgangssignale angehoben werden. Mit dem Potenziometer wird dieser Offset auf die halbe Referenzspannung des AD-Wandlers eingestellt. Der Ausgangspegel des Tiefpasses beträgt somit:

$$U_a = \frac{3.3V}{2} \pm \frac{300°/s * 0.67mV}{°/s} * 4.7 = 0.7V..2.6V$$

Aufgrund des internen Aufbaus der ausgewählten Operationsverstärker [28] ist der maximale Ausgangspegel immer kleiner als die Versorgungsspannung. Somit kann bei einer 3.3V Versorgungsspannug der benötigte Ausgangspegel von 2.6V nicht geliefert werden. Die Operationsverstärker werden daher mit 5V versorgt.

Beim Platinenlayout muss darauf geachtet werden, dass die Gyroskope parallel zu der zu messenden Achse positioniert werden. Die Sensoren für Nick- und Rollachse werden im 90° Winkel zueinander auf der Platine angeordnet, der Sensor für die Gierachse wird senkrecht dazu montiert.

5.1.3 Beschleunigungssensor

Zur Messung der statischen (Erd-)Beschleunigung kommt ein 3-achsiger LIS3L02AS4 [30] des Herstellers ST mit einem Messbereich von +/-2g zum Einsatz. Im Gegensatz zu den Gyroskopen muss das Ausgangssignal der Sensoren nicht durch einen Operationsverstärker angepasst werden. Der Ausgangspegel beträgt 660 mV/g und kann somit direkt vom AD-Wandler verarbeitet werden.

Die hochohmigen Ausgänge des Sensors bilden zusammen mit externen Kondensatoren Tiefpässe zur Filterung von Störungen durch Motorvibrationen (Abbildung 5.4). Diese Störungen treten in einem niedrigeren Frequenzbereich auf, daher werden auch die Grenzfrequenzen der Tiefpässe niedriger als bei den Verstärkern der Gyroskope gewählt.

Die Grenzfrequenz f_g des Tiefpasses beträgt:

$$f_g = \frac{1}{2\pi * RC} = \frac{1}{2\pi * 110k\Omega * 100nF} = 14.5Hz$$

Beim Platinendesign müssen die Achsen des Beschleunigungssensors parallel zu den Messachsen ausgerichtet sein. Weiterhin sollte eine Verschiebung der Drehachse vermieden werden, da ansonsten bei schnellen Drehungen die auftretenden Fliehkräfte als Beschleunigungen gemessen werden.

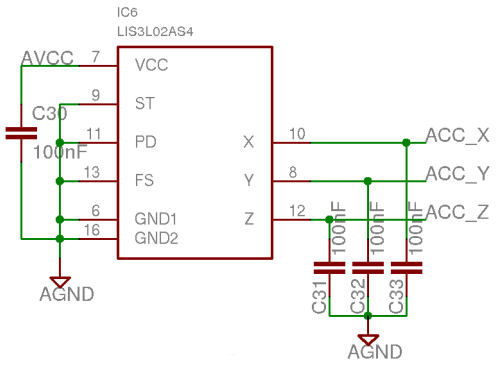


Abbildung 5.4: Teilschaltung Beschleunigungssensor

5.1.4 RC-Empfänger

Ein handelsüblicher RC-Empfänger aus dem Modellbaubereich gibt pro Kanal ein Rechtecksignal mit variabler Impulslänge aus. Über diese Signale werden Motorregler und Servos angesteuert. Über die Impulsbreite wird dabei die jeweilige Funktion der Fernsteuerung übertragen. Eine einzelne Auswertung der vier benötigten Steuersignale (Gas, Nick, Roll, Gier) wäre mit dem Controller möglich, würde aber vier Timerbaugruppen und IO-Pins benötigen. Über das interne Summensignal des Empfängers können dagegen alle Steuerfunktionen mit nur einer Timerbaugruppe ausgewertet werden.

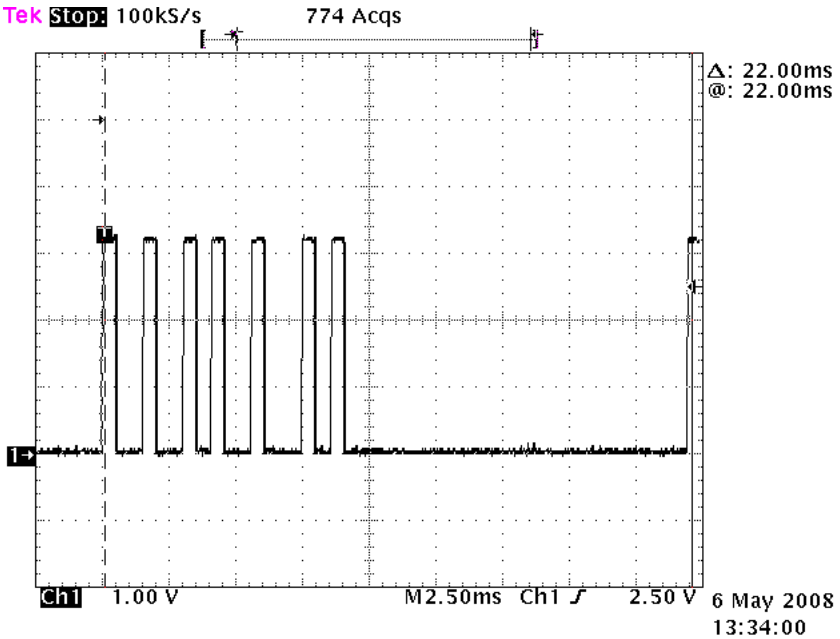


Abbildung 5.5: Summensignal eines RC-Empfängers

In dem Summensignal werden die einzelnen Steuerfunktionen der Fernsteuerung durch eine Puls Pausen Modulation (PPM) kodiert übertragen. Ein Impuls hat dabei immer die Länge von 500μs, die Impulspause ist je nach Stellung der Fernsteuerung 1-2ms lang (Abbildung 5.5). Der komplette Frame, bestehend aus 7 Impulsen, wiederholt sich alle 22ms.

Der Fernsteuerungsempfänger wird von der Platine mit 5V versorgt, das herausgeführte Summensignal wird der ersten Timerbaugruppe des Controllers zugeführt und ausgewertet. Da die IO-Pins des Controllers 5V tolerant sind und die vollständige Auswertung und Synchronisierung des Signals per Software erfolgen kann, ist eine weitere Anpassung nicht erforderlich.

5.1.5 Schnittstellen

Zur Kommunikation mit weiteren Baugruppen des Quadrocopters und externen Komponenten benötigt die Fluglageregelung eine Reihe von Hardwareschnittstellen, deren Funktionen im Folgenden abgegrenzt werden.

I2C-Bus

Der I2C-Bus [29] ist ein synchroner, serieller Datenbus für Geräte mit geringer Übertragungsgeschwindigkeit. Die TWI Baugruppe des Controllers ist kompatibel mit dem I2C-Bus und dient der Kommunikation mit den vier Motorreglern und dem EEPROM. Da die Motorregler einen I2C-Bus mit 5V Pegeln benötigen, werden sowohl die SDA/SCL Pullup-Widerstände, als auch das EEPROM mit 5V versorgt. Die Pullup-Widerstände sind darauf ausgelegt den Bus im Fast-I2C-Mode mit max. 400kHz betreiben zu können.

Serielle Ports

Zum einfachen Anschluss eines PCs oder von externen Sensoren werden serielle Schnittstellen nach RS232C Standard [17] verwendet. Der Controller enthält drei universelle, serielle Baugruppen (USART), die eine Kommunikation über verschiedene serielle Protokolle (RS232, RS485, SmartCards und IrDA) ermöglichen. Die TX/RX Signale von zwei dieser USARTs werden über einen MAX3232 [25] Pegelwandler geführt und an Lötpads auf der Platine zur Verfügung gestellt.

USB Device Port

Der Universal Serial Bus (USB) [31] ist eine schnelle serielle Schnittstelle zur Anbindung von Peripheriegeräten an einen PC. Über den USB Device Port des Controllers kann dieser als USB-Gerät arbeiten, ein Anschluss von anderen Geräten ist nicht möglich. Die komplette Signalaufbereitung und Verarbeitung ist bereits im Controller integriert. Als mechanischer USB-Anschluss wurde aufgrund der geringen Größe eine mini-B USB-Buchse gewählt.

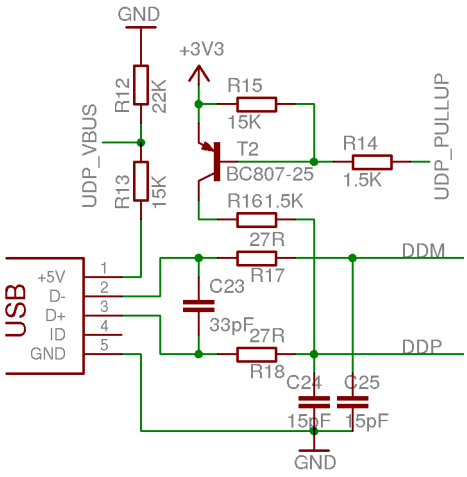


Abbildung 5.6: Teilschaltung USB Device Port

Zusätzlich zu den USB-Signalen DDM und DDP benötigt der Controller zwei weitere Signale. Mit Hilfe des UDP_VBUS Signals erkennt der Controller die Verbindung mit einem USB-Host. Über das UDP_PULLUP Signal kann der Controller dem Host seine eigene Kommunikationsbereitschaft mitteilen (Abbildung 5.6). Die Zuordnung dieser beiden Signale an die IO-Pins, sowie die Anpassung der USB-Pegel an den Controller ist im Hardware Design Guide [21] des Herstellers detailiert beschrieben.

JTAG

Der IEEE 1149.1 JTAG-Anschluss [15] zum Programmieren und Debuggen des Controllers ist über einen 8pol Pfostenstecker zugänglich. Über einen externen Adapter kann ein Standard 2x10pol ARM JTAG Adapter angeschlossen werden. (Siehe Anhang A) Auf dem externen Adapter befindet sich weiterhin ein Reset-Taster für den Controller.

5.1.6 Stromversorgung

Die Platine wird im laufenden Betrieb von einem dreizelligen Lithium-Polymer-Akku mit 11.1V Nennspannung gespeist. Über einen Kanal des AD-Wandlers kann der Controller die aktuelle Betriebsspannung laufend überwachen.

Zur Versorgung der Komponenten dienen zwei lineare Spannungsregler. Ein 3.3V Spannungsregler versorgt den Controller und die Sensoren. Das I2C-EEPROM, die Operationsverstärker, sowie der RC-Empfänger werden von einem 5V Spannungsregler versorgt. Eine Stromversorgung der Platine über den USB Port ist nicht vorgesehen, da hierfür eine aufwendige Umschaltlogik der 5V Versorgung nötig gewesen wäre. Aufgrund des niedrigen Stromverbrauchs ist ein Schaltregler mit kleinerer Verlustleistung nicht vorgesehen.

Um Störungen zu vermeiden, ist der analoge Schaltungsteil bestehend aus Sensoren und Operationsverstärkern auf der Platine vom restlichen digitalen Teil getrennt angeordnet. Die Versorgungsspannungen des analogen Teils werden zur Entstörung über Drosselspulen geführt und jeweils nur an einem Punkt übergeben, um Masseverschleifungen zu verhindern. Als Referenzspannung für den internen AD-Wandler des Controllers wird die 3.3V Versorgung des analogen Schaltungsteils verwendet.

5.1.7 Erweiterungen

Um auch während des Fluges einfache Rückmeldungen zu erhalten, befinden sich zwei schaltbare LEDs, sowie ein Anschluss für einen magnetischen Summer auf der Platine. Dieser Summer ist laut genug, um auch aus größerer Entfernung noch wahrgenommen zu werden.

Neben den 3.3V und 5V Versorgungsspannungen werden alle freien Signale des Controllers auf einen 20pol Pfostenstecker geführt und können für Erweiterungen genutzt werden.

5.1.8 Erstellung des Platinenlayouts

Der Schaltplan und das Layout der Platine werden mit dem Programm “Eagle” der Firma Cadsoft [5] erstellt. In der “Light” Version ist dieses Entwurfsprogramm für nicht-kommerzielle Projekte frei verwendbar, wobei nur die Abmessungen der Platinenlayouts und Umfang der Schaltpläne eingeschränkt sind.

Der erstellte Schaltplan ist in Anhang A.2 zu finden. Für das doppelseitige Platinenlayout (Abbildung 5.7) wurden mit Ausnahme der Gyroskope und der Steckverbinder nur oberflächenmontierte (SMD) Bauteile verwendet. Aufgrund der geringen Abmessungen (52x52mm)

der Platine konnte das automatische Routing von Eagle nicht überzeugend arbeiten. Daher wurde das Layout vollständig per Hand erstellt. Die ungenutzten Platinenflächen von Analog- und Digitalteil sind zur Vermeidung von Störungen mit dem jeweiligen Massepotential verbunden.

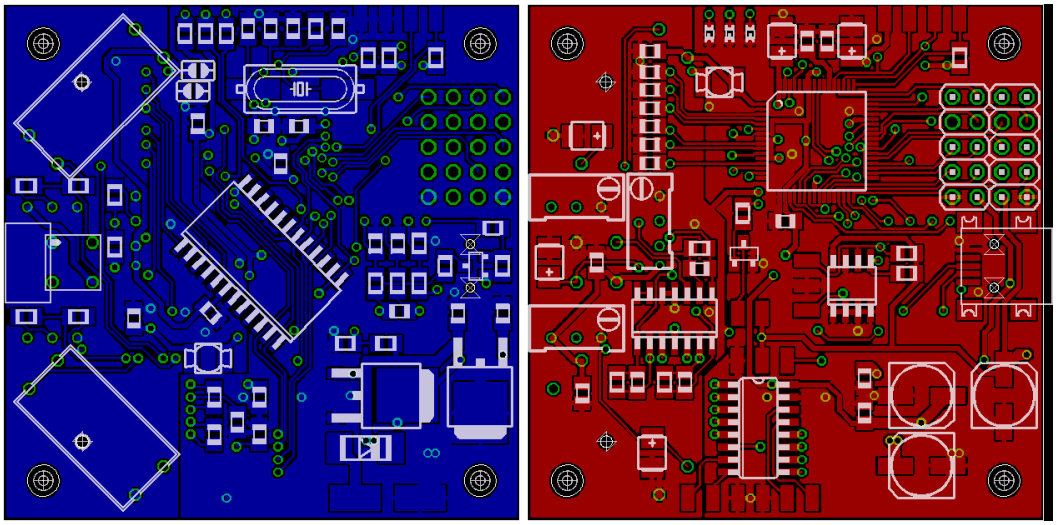


Abbildung 5.7: Unter- und Oberseite des Platinenlayouts

5.2 Software

Das Softwaredesign der Controller-Firmware läßt sich in Komponenten für den Hardwarezugriff und nebenläufige Prozesse aufteilen. Die hardwarenahen Funktionen umfassen das Einlesen der Sensordaten über den AD-Wandler, die Auswertung des RC-Summensignals mit Hilfe einer Timerbaugruppe und die Kommunikation über I2C-Bus und USB. Fluglageregelung und Telemetriesystem stellen nebenläufige Prozesse dar und nutzen die hardwarenahen Funktionen. Die Fluglageregelung verknüpft die Sensorwerte und Fernsteuerungssignale und errechnet daraus Sollwerte für die Motorregler. Das Telemetriesystem dient der Überwachung und Parameterisierung der Fluglageregelung.

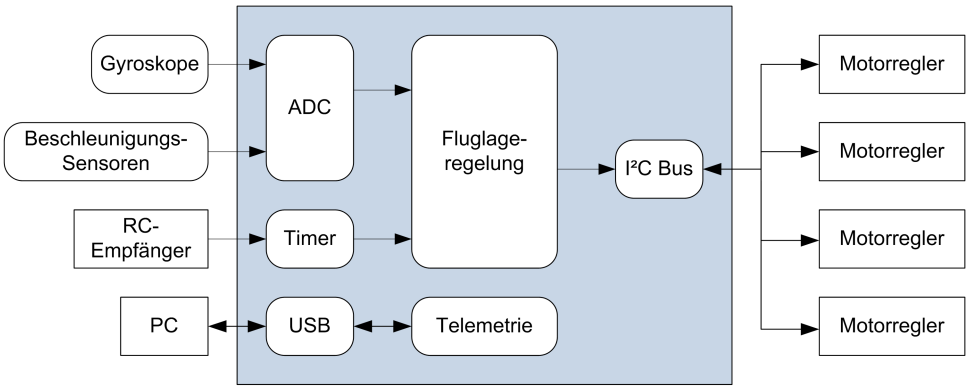


Abbildung 5.8: Software Übersicht

Die Firmware wird vollständig in der Programmiersprache C verfasst. Da die Fluglageregelung periodisch aufgerufen wird und sich die Verarbeitung der Telemetriedaten leicht unterteilen lässt, können beide von einer gemeinsamen Hauptschleife der Software abgearbeitet werden. Somit ist der Einsatz eines Open Source Betriebssystems wie FreeRTOS [7] oder nutOS [11] nicht erforderlich.

5.2.1 AD-Wandlung der Sensorsignale

Mit Hilfe des internen AD-Wandler des Controllers werden die Signale der drei Gyroverstärker, des 3-achsigen Beschleunigungssensors, sowie die Batteriespannung gemessen (Abbildung 5.9). Bei der maximalen Auflösung von 10bit beträgt die Wandelzeit nur knapp 3μs. Die Wandlung der Sensorsignale kann nacheinander erfolgen, da es bei dieser Anwendung keinen Bezug zwischen den einzelnen Werten gibt.

Nach der Wandlung müssen die Nullpunkt-Offsets der Sensorsignale abgezogen werden, wodurch sich vorzeichenbehaftete Werte der Drehwinkelgeschwindigkeiten bzw. der Beschleunigungen ergeben. Diese Offsets entsprechen den absoluten Signalwerten, wenn der

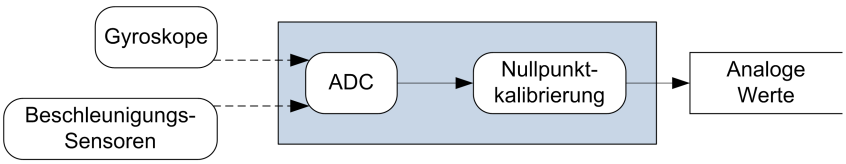


Abbildung 5.9: Wandlung der Sensorsignale

Quadrocopter auf einer ebenen Fläche ruht und müssen persistent abgespeichert werden, um sie nicht nach jedem Einschalten neu ermitteln zu müssen.

Die Sensorsignale werden ohne feste Skalierung verwendet, für die Batteriespannung wird das Ergebnis der AD-Wandlung in Volt umgerechnet. Für diese Umrechnung werden die Werte des vorgeschalteten Spannungsteilers benötigt (Abbildung A.2 im Anhang). Die Batteriespannung kann nach folgender Gleichung errechnet werden:

$$U_{bat} = \frac{ADC * U_{ref}}{1024} * \frac{R1 + R2}{R2} = \frac{ADC * 3.3V}{1024} * \frac{10k\Omega + 1k\Omega}{1k\Omega}$$

5.2.2 Auswertung des RC-Summensignals

Die Fernsteuerungsbefehle werden als Summensignal an die Timerbaugruppe 1 des Controllers geführt und dort ausgewertet. Der Timer läuft im Capture-Mode und ermittelt so laufend die Impulslängen des Signals.

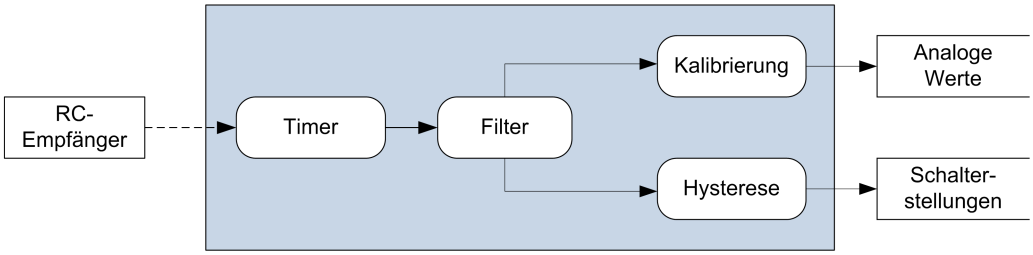


Abbildung 5.10: Auswertung der Fernsteuerungssignale

Die Länge einer Impulspause entspricht jeweils einer Steuerfunktion der Fernsteuerung, wobei das Summensignal alle 22ms erneut übertragen wird. Um eine möglichst feinfühlige Steuerung zu erhalten, werden die Impulslängen über mehrere PPM-Frames gemittelt. Eine anschließende Skalierung auf einen definierten Wertebereich vereinfacht die Berechnungen in der Fluglageregelung (Abbildung 5.10).

Die für die Skalierung benötigten Kalibrierungsdaten müssen nicht-flüchtig gespeichert und bei einem Wechsel der Fernsteuerung erneut erzeugt werden. Zur Kalibrierung müssen alle

Bedienelemente der Fernsteuerung zunächst einzeln in ihre min/max Positionen gebracht werden, danach wird ihre Ruhelage ermittelt.

Zusätzlich werden aus den analogen Werten durch fest definierte Schwellen Schaltfunktionen abgeleitet, die nicht von den Kalibrierungsdaten abhängig sind. Über diese Funktionen können später einfache Steuerungsaufgaben realisiert werden.

5.2.3 I2C-Kommunikation

Über den I2C-Bus kommuniziert der Controller mit den 4 Motorreglern, sowie mit dem EEPROM zur Speicherung von Parametern und Kalibrierungsdaten. Die Ansteuerung der Motorregler muss synchron zur Lageregelung erfolgen, wobei pro Motorregler 3 Bytes Daten übertragen werden (siehe Anhang B). Zusammen mit Start/Stopbedingungen und ACK Bits werden somit ca. 30 SCL Takte nur für das Setzen eines neuen Sollwerts benötigt. Das komplette I2C-Protokoll der Motorregler findet sich in Anhang B.

Weiterhin kann die Firmware auf den Motorreglern den I2C-Bus durch Clock Stretching kurzzeitig anhalten, z.B. wenn gleichzeitig höher priorisierte Interrupts auf den Motorreglern ausgeführt werden.

Ausgehend von der spezifizierten I2C-Frequenz von 100kHz werden also

$$t = \frac{4 * 30}{100kHz} = 1.2ms$$

zum Setzen von neuen Sollwerten auf allen Reglern benötigt. Da die Motorregler und das EEPROM bis mit zu 400kHz SCL Takt nach Fast-I2C-Spezifikation [29] betrieben werden können, kann die Frequenz im Laufe der Implementierung erhöht werden.

Grundsätzlich ist ein Zugriff auf das EEPROM auch während des Betriebs der Motorregler möglich. Aufgrund des kritischen Timings müsste dieser Zugriff aber in mehrere, kurze Abschnitte unterteilt werden, die die Kommunikation mit den Motorreglern nicht beeinflussen. Daher sollte in diesem Zustand das EEPROM nicht verwendet werden.

5.2.4 USB-Kommunikation

Über den USB-Anschluss kann der Controller mit einer Telemetrieanwendung kommunizieren, die auf einem PC läuft. Um auf der PC-Seite keine spezielle Software für die USB-Ansteuerung zu benötigen, bildet der Controller hierfür eine serielle Schnittstelle nach.

Der USB Device Port des AT91SAM7 Controllers enthält vier adressierbare Endpoints, die zum Senden oder Empfangen von Daten konfiguriert werden können (Abbildung 5.11).

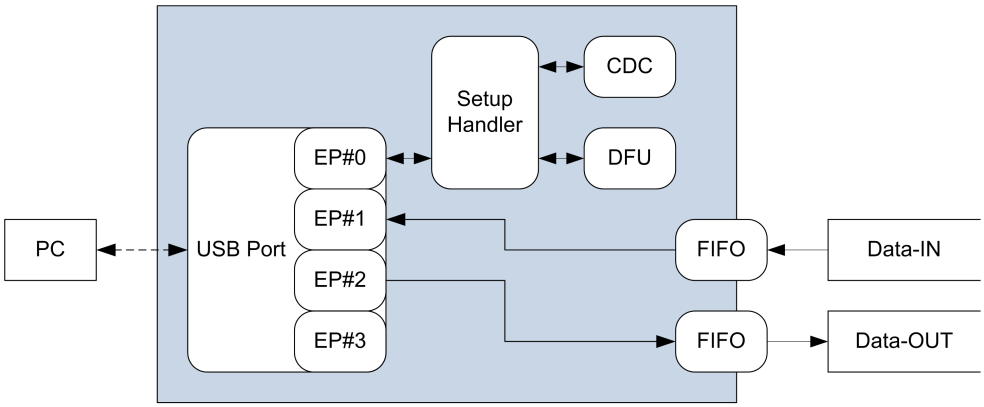


Abbildung 5.11: USB-Kommunikation

Nachdem der USB-Host (PC) den Controller als neues Gerät am Bus erkannt hat, beginnt die Enumeration des Controllers über den ersten Endpoint. Nach Vergabe einer eindeutigen Busadresse bietet das USB-Gerät eine Auswahl von Konfigurationen an, aus denen der USB-Host wählen kann. Ein Gerät kann mehrere Konfigurationen für unterschiedliche Aufgaben bereitstellen, wobei aber nur eine Konfiguration zur Zeit aktiv sein kann. Der Host kann nur zwischen den Konfigurationen wählen, sie aber nicht frei verändern.

Eine Konfiguration besteht meist aus einer Basisklasse, die durch weitere Attribute ergänzt wird. Die Implementierung einer seriellen Schnittstelle ist so Teil der Communication Device Class Spezifikation (CDC) [32] und wird darin über Attribute des Abstract Control Model (ACM) weiter definiert. Innerhalb der Konfiguration werden zwei weitere Endpoints für die Datenübertragung der seriellen Schnittstelle vorgesehen. Zusätzlich ist im ACM ein Endpoint als Interruptquelle für den USB-Host vorgesehen, der eingerichtet werden muss.

Über den ersten Endpoint werden außerdem Parameter und Statusmeldungen der implementierten Funktionen versendet. Im Fall einer seriellen Schnittstelle sind dies Handshakesignale, wie sie für Modems genutzt werden, und die Parameter der Baudrateneinstellung. Eine Auswertung der übertragenen Parameter muss in dieser Anwendung nicht erfolgen, allerdings müssen die Daten angenommen und bestätigt werden.

Zusätzlich zu der seriellen Schnittstelle wird die Device Firmware Upgrade (DFU) [33] Funktion implementiert. Über diese Funktion kann der USB-Host einen speziellen Bootloader auf dem Controller starten, mit dessen Hilfe ein Austausch der Firmware möglich ist.

Die Schnittstellenemulation wird auf einem Linuxsystem ohne weitere Treiber oder Anpassungen als “/dev/ttyACMx” Gerät bereitgestellt und kann wie eine “echte” serielle Schnittstelle von allen Programmen verwendet werden.

5.2.5 Lageregelung

Die Lageregelung wird vollständig als Software implementiert und vergleicht die drei tatsächlichen Lagewinkel des Quadrocopters mit den von der Fernsteuerung vorgegebenen Lagewinkeln. Die daraus errechneten Stellwerte für die einzelnen Achsen werden von einem Mixer in die Stellwerte für die vier Motoren umgerechnet (Abbildung 5.12).

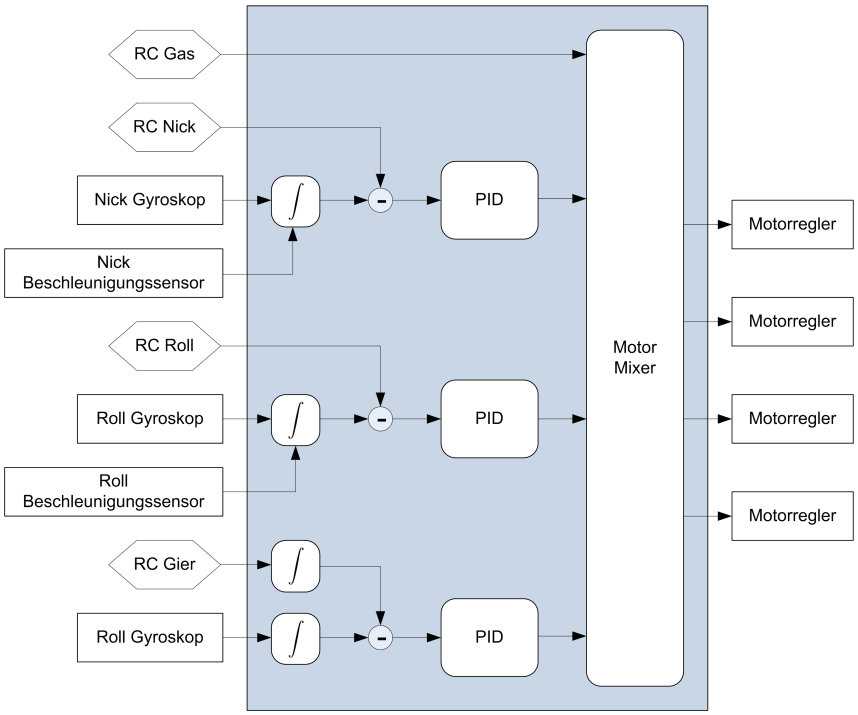


Abbildung 5.12: Lageregelung

Die momentanen Lagewinkel entsprechen den Integralen der Drehwinkelgeschwindigkeiten, die von den Gyroskopen direkt gemessen werden. Für die numerische Integration müssen die Messsignale periodisch erfasst werden, wobei eine hohe Abtastrate entscheidend für die Genauigkeit der Integrale ist.

Durch die Integration addieren sich auch Störungen der Sensorsignale mit der Zeit zu großen Abweichungen der Lagewinkel auf, wodurch der Quadrocopter schließlich nicht mehr “gerade” schwebt. Diese Fehler können mit Hilfe der Beschleunigungssensoren kompensiert werden, da sich über die Messung der statischen Erdbeschleunigung die absoluten Lagewinkel für Nick- und Rollachse ermitteln lassen. Ein Rückschluss auf den absoluten Lagewinkel der Gierachse ist über die Beobachtung der Erdbeschleunigung nicht möglich. Hierfür ist zusätzliche Sensorik erforderlich, z. B. in Form eines elektronischen Kompasses.

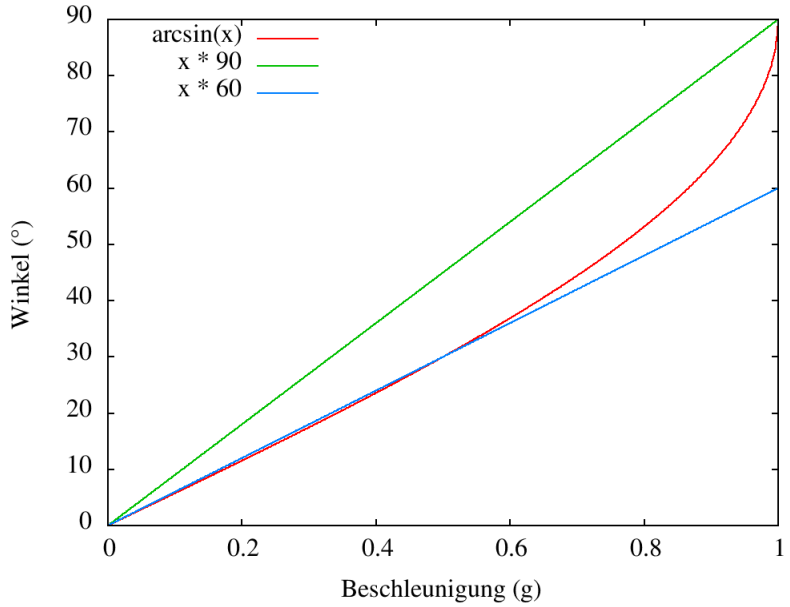


Abbildung 5.13: Umrechnung der Beschleunigungen

Die Beschleunigungswerte der Sensoren stellen die einwirkenden Kräfteverhältnise zwischen den senkrecht zueinander angeordneten Achsen dar. Diese Verhältnisse müssen vor der Verwendung als Korrekturwert der Integrale mit Hilfe des Arkussinus in einen Winkel umgerechnet werden. Eine Korrektur der Lageintegrale ist nur während des schwebenden Zustandes sinnvoll, da die bei Kurvenflügen auftretenden Fliehkräfte die Messungen der Beschleunigungssensoren verfälschen. Durch diese Einschränkung kann der Arkussinus durch einen Faktor ersetzt werden, da sich dieser bei den Anstellwinkeln des Schwebeflugs ($< 30°$) annähernd linear verhält (siehe Abbildung 5.13).

Der Einfluss der so errechneten Korrekturwerte des Beschleunigungssensors muss dabei groß genug sein, um die Störungen der Gyrosignale ausgleichen zu können. Da die Lageregelung nicht sicher erkennen kann, ob der Quadrocopter schwebt oder sich im Kurvenflug befindet, darf der ständige Einfluss der absoluten Lagewinkel auf die Integrale nicht zu groß sein.

Die aus den Signalen der Gyroskope und Beschleunigungssensoren berechneten Lagewinkel werden mit den Vorgaben der Fernsteuerung verglichen. Bei Nick- und Rollachse werden die Kanäle der Fernsteuerung als Sollwinkel der jeweiligen Achse interpretiert. Werden die Kanäle wieder in Ruhelage gebracht, steuert der Quadrocopter in den waagerechten Schwebeflug. Bei der Gierachse wird der Fernsteuerungskanal als Drehgeschwindigkeit interpretiert. Wird der Kanal ausgelenkt, dreht sich der Quadrocopter um seine Gierachse. Wieder in

Ruhelage, stoppt die Drehung. Somit ist eine vollständige Drehung, aber auch eine manuelle Nachführung dieser Achse möglich.

Aus den Sollwerten der Fernsteuerung und den Istwerten der Lagewinkel wird für jede Achse getrennt die Regeldifferenz eines PID-Reglers errechnet. Ein PID-Regler besteht aus drei Übertragungsgliedern, deren Ausgangsgrößen addiert werden.

- Das Proportionalglied verstärkt die Regelabweichung mit dem Proportionalbeiwert Kp und gibt das Ergebnis unverzögert aus. Die Stellgröße ist damit nur von der momentanen Regeldifferenz abhängig. Das P-Glied wird bei der Lageregelung für das schnelle Annähern an die Sollwinkel verwendet. Der Winkelfehler kann allerdings nicht vollständig ausgeregelt werden.
- Das Integralglied verstärkt das Integral der Regelabweichung (also die Summe über die Zeit) mit dem Integralbeiwert Ki. Die Stellgröße ist somit von der Dauer der Regeldifferenz abhängig. Das I-Glied arbeitet langsam, wird aber für eine vollständige Ausregelung der Lagefehler benötigt. Der maximale Einfluss muss begrenzt sein, da der Quadrocopter ansonsten ein starkes Schwingverhalten zeigt.
- Das Differentialglied verstärkt die Veränderung der Regelabweichung Zeit mit dem Differenzierbeiwert Kd. Bei einer konstanten Regeldifferenz bleibt die Stellgröße somit 0. Das D-Glied arbeitet sehr schnell und ist bei der Lageregelung des Quadrocopters für das Ausregeln von externen Störungen vorgesehen.

Für einen zeitdiskreten PID Regler gilt somit:

$$y_k = \underbrace{Kp * e_k}_{P-Anteil} + \underbrace{Ki * Ta * \sum_{i=0}^{k} e_i}_{I-Anteil} + \underbrace{\frac{Kd}{Ta}(e_k - e_{k-1})}_{D-Anteil}$$

e stellt die Regeldifferenz, y die Stellgröße des PID-Reglers dar. Ta beschreibt die Abtastrate des Reglers, also die Zeit zwischen zwei Rechenschritten. Somit können Integral- und Differenzierbeiwerte auch bei unterschiedlicher Lageregelfrequenz gleichbleiben.

Die Stellgrößen der drei PID-Regler, sowie der Gas-Kanal der Fernsteuerung werden von einem Motor-Mixer in die Sollwerte der Motorregler umgerechnet. Der Gas-Kanal wirkt dabei gleichmäßig auf alle Motoren, die Stellgröße des Gier-Reglers verändert das Verhältnis zwischen links- und rechtsdrehenden Motoren. Über die Stellgrößen der Nick- und Rollregler wird das Verhältnis der jeweiligen Achse verändert. (siehe auch Abbildung 2.1)

$$\begin{aligned} motor_A &= rc_{gas} + y_{gier} + y_{nick} \\ motor_C &= rc_{gas} + y_{gier} - y_{nick} \\ motor_B &= rc_{gas} - y_{gier} + y_{roll} \\ motor_D &= rc_{gas} - y_{gier} - y_{roll} \end{aligned}$$

5.3 Telemetriesystem

Für den Vergleich der verschiedenen Fluglageregelungen wird ein Telemetriesystem benötigt, welches Messwerte und Stellgrößen graphisch darstellen kann. Diese Darstellung muss in Echtzeit erfolgen, um eine unmittelbare Verbindung zwischen Flugverhalten und Graphen herstellen zu können. Neben der Aufzeichnung von Graphen muss das System die Möglichkeit bieten, Parameter der Fluglageregelung während der Laufzeit zu verändern.

Die Kommunikation des Telemetriesystems zwischen PC und Controller ist durch ein einfaches, binäres Protokoll beschrieben. Aufgrund des hohen Datendurchsatzes wird zur Übertragung die serielle USB-Schnittstellenemulation verwendet. Das System kann aber sehr einfach auf andere Hardware angepasst werden. Weiterhin ist die Nutzung der Telemetrieverbindung für zusätzliche Baugruppen vorgesehen. Der Controller dient dann als Relais für diese Baugruppen.

Die Übertragung einer Telemetrievariablen ist in Metadaten und den eigentlichen Wert aufgeteilt. Die Metadaten umfassen eine eindeutige ID, den Datentyp und den Namen oder eine Beschreibung der Variablen auf dem Controllersystem. Da sich diese Daten zur Laufzeit nicht ändern, werden sie nur einmalig zu Beginn einer Telemetrieverbindung übertragen. Der Controller beantwortet diese Anfrage für jede Variable getrennt (Abbildung 5.14).

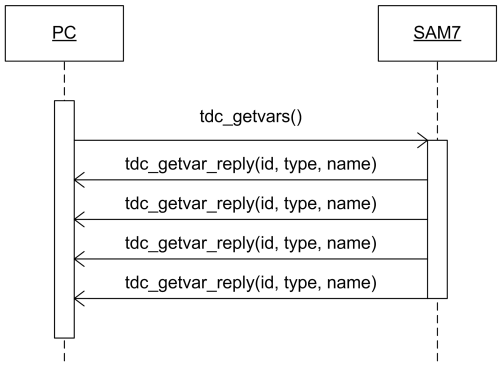


Abbildung 5.14: Telemetrie: Übertragung von Metadaten

Anhand der IDs in den Metadaten kann das Telemetriesystem die Werte jeder Variablen einzeln anfordern. Für das Antwortpaket liest der Controller den Wert aus und überträgt ihn in binärer Form an den PC. Da beiden Kommunikationspartnern der Datentyp bekannt ist, müssen außer der ID keine weiteren Daten übertragen werden. (Abbildung 5.15). Um den Wert einer Parametervariablen zu setzen, schickt der PC die ID und den binären Wert zum Controller. Variablen können bei der Einbindung in das Telemetriesystem als nur-lesend markiert werden. Dadurch lassen sich Statuswerte schützen, die nicht geschrieben werden dürfen.

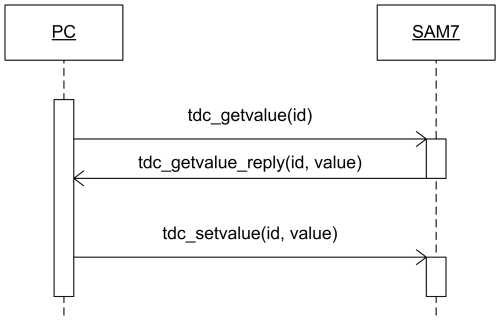


Abbildung 5.15: Telemetrie: Übertragung von Werten

Um beim Aufzeichnen von Graphen die Variablen nicht einzeln anfordern zu müssen, kann ein periodisches Update angefordert werden. Zu diesem Zweck sendet der PC eine Bitmaske mit einer Updaterate zum Controller. In der Bitmaske sind die IDs der angeforderten Variablen kodiert, die Updaterate beschreibt den Intervall in dem der Controller aktuelle Werte schicken soll (Abbildung 5.16). Das minimale Updateintervall beträgt 10ms. Reicht diese Zeit nicht zum Versand aller Variablen aus, werden diese aus dem periodischen Update ausgeschlossen.

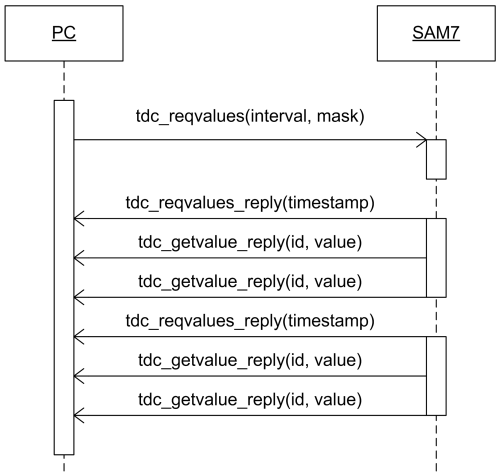


Abbildung 5.16: Telemetrie: Periodische Übertragung

6 Implementierung

In diesem Kapitel wird zunächst der Aufbau eines einfachen Quadrocopters auf Basis der in Kapitel 5.1 entwickelten Fluglageregelung beschrieben. Im Anschluss wird die Implementierung der in Kapitel 5.2 aufgeteilten Softwarekomponenten vorgestellt.

6.1 Aufbau des Quadrocopters

Das in Kapitel 5.1.8 beschriebene Platinenlayout wurde von einem externen Hersteller als Prototyp produziert und anschließend von Hand bestückt und gelötet (Abbildung 6.1). Das Gyroskop zur Messung der Drehwinkelgeschwindigkeit der Gierachse muss senkrecht zur Platine montiert werden. Um die mechanische Stabilität zu verbessern, wird das das Gyroskop mit einem kleinen Streifen Platinenmaterial an der Hauptplatine mit Heißkleber befestigt. Weiterhin werden die Potenziometer zur Einstellung der Gyroskop-Nullpunkte liegend bestückt.

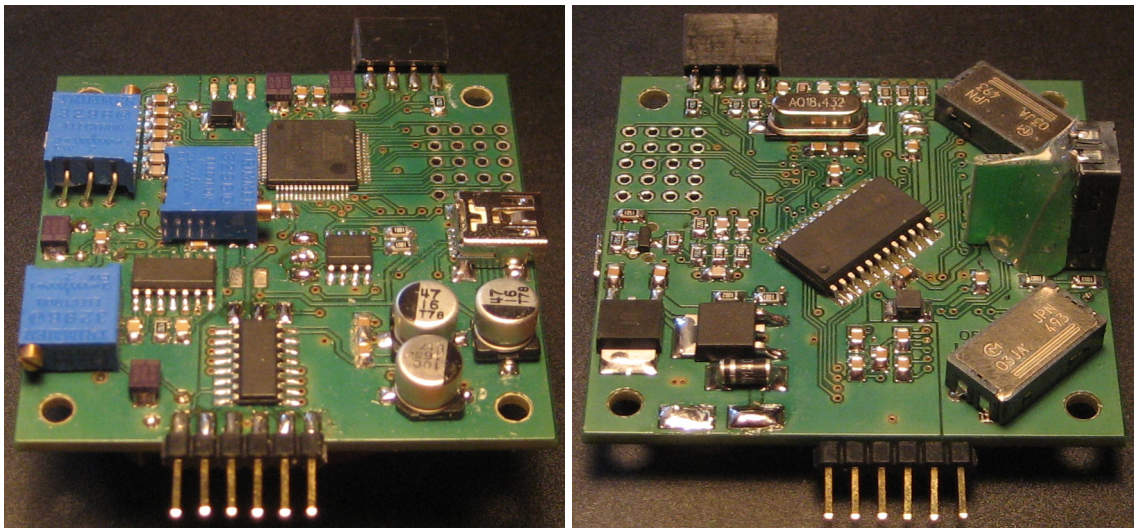


Abbildung 6.1: Ober- und Unterseite der bestückten Platine

Die Gesamtkosten für Herstellung einer Lageregelungsplatine betragen 133€. Diese Kosten lassen sich in drei Bereiche aufteilen:

- **Platinenherstellung:**
  Die Herstellungskosten der Platine werden auf Basis der Abmessungen berechnet. Aufgrund der Mindestbestellfläche von 1dm^2 ergibt sich ein Preis von 52€ für 6 einzelne Platinen.
- **Sensoren:**
  Die Sensorik der Lageregelung besteht aus Spezialbauteilen, die nur schwer erhältlich sind. Die Kosten für die Gyroskope und den Beschleunigungssensor betragen zusammen 95€ Eur.
- **restliche Bauteile:**
  Alle weiteren Bauteile inklusive des ARM7 Controllers sind Standardbauteile und bei einer Reihe von Distributoren erhältlich. Die Kosten betragen ca. 29€.

Für den Aufbau des Quadrocopters wird ein Rahmen aus 10x10mm Aluminiumprofilen gefertigt. Der Achsabstand zwischen zwei Motoren auf einer Achse beträgt 40cm.

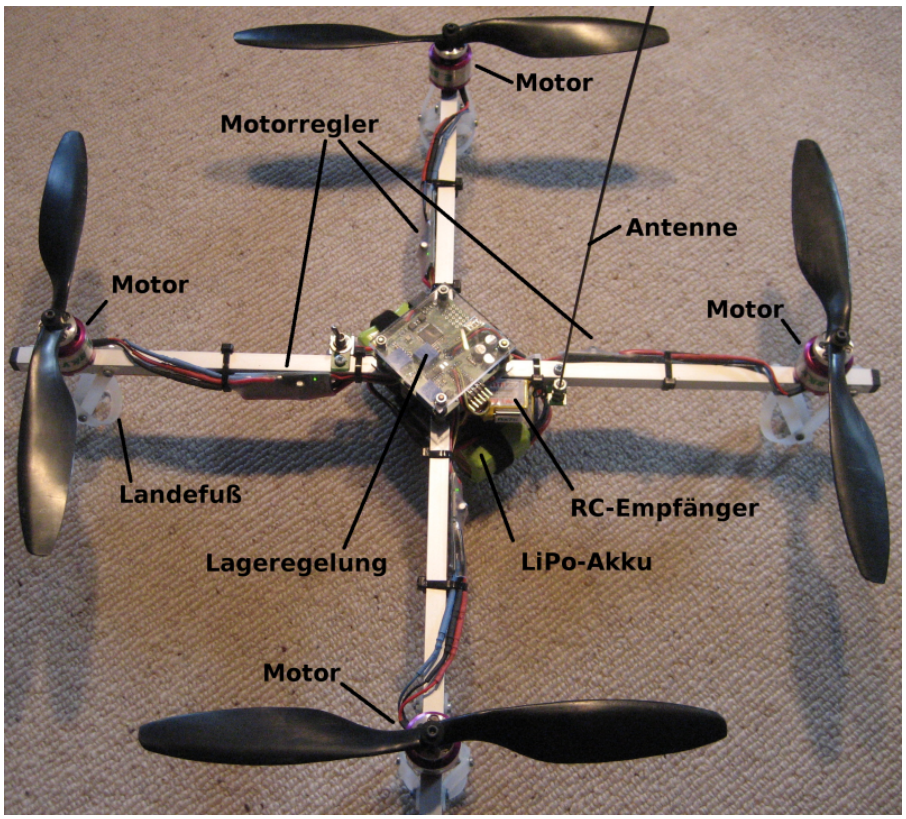


Abbildung 6.2: Quadrocopter Aufbau

Um die Verkabelung der Motoren kurz zu halten, werden die vier Motorregler einzeln an den Auslegern befestigt. Die Platine der Fluglageregelung ist zentral über dem Kreuz der Aluminiumprofile montiert und wird von einer Acrylglasscheibe geschützt. Der LiPo-Akku liegt unter dem Schwerpunkt des Quadrocopters und hat durch die Landefüsse eine Bodenfreiheit von ca. 2cm.

Das Gesamtgewicht des Quadrocopters beträgt flugfertig 716g.
Abbildung 6.3 zeigt die Verteilung des Gewichts auf die einzelnen Baugruppen.

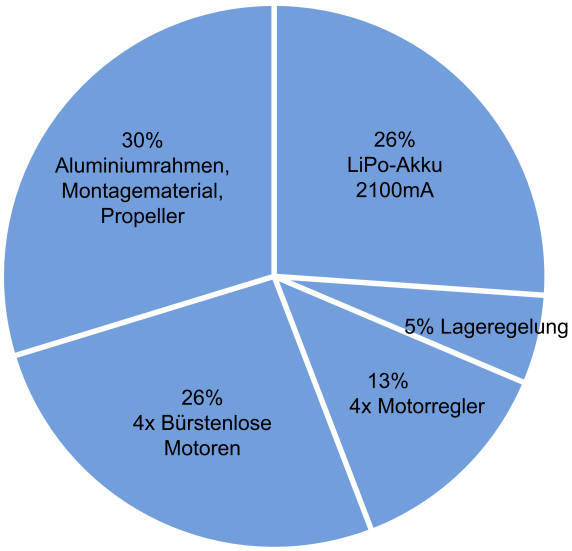


Abbildung 6.3: Gewichtsaufteilung des aufgebauten Quadrocopters

6.2 Software

Zur Implementierung der Controllerfirmware wird der frei verfügbare GNU-C-Compiler (gcc) [8] verwendet. Zusammen mit der Entwicklungsumgebung Eclipse [6, 4] und dem JTAG Tool openOCD [12] kommen nur Open Source Programme zum Einsatz.

Die Software für die Lageregelung wird vollständig in der Programmiersprache C implementiert, nur die Startup Routine und die zentralen Interrupthandler sind in Assembler verfasst. Die Startup Routine initialisiert die Stack- und Datenbereiche des C-Programms. Außerdem werden die kritischen Hardwarebaugruppen (Oszillator, Interruptcontroller) konfiguriert. Im zentralen Interrupthandler wird der Betriebsmodus des ARM-Cores gewechselt, wodurch ein Teil der CPU Register verändert wird. Diese Register (Stackpointer, Rücksprungadresse) müssen für eine Schachtelung von Interrupts gesichert werden und an die Interrupt Service Routinen (ISR) angepasst werden. Durch diese Anpassung können die eigentlichen ISRs in C implementiert werden.

Im folgenden wird auf die Implementierung der einzelnen Softwarekomponenten der Fluglageregelung eingegangen.

6.2.1 Abort Handler

Um die Fehlersuche bei Speicherzugriffsfehlern zu vereinfachen, wird eine Routine zur Ausgabe des momentanen CPU Zustandes auf einer seriellen Schnittstelle implementiert.

Der normale Programmablauf kann bei der ARM-Architektur neben Interrupts auch durch sogenannte Exceptions unterbrochen werden. Beim Auftreten einer solchen Exception wird ähnlich einem Interrupt ein Handler aufgerufen, der die Ursache der Exception verarbeitet. Der Memory Controller des SAM7 kann durch einen Speicherzugriffsfehler zwei verschiedene Exceptions auslösen:

- Ein “Prefetch Abort” wird durch das Laden einer Instruktion von einer nicht gültigen Adresse ausgelöst. Gültige Adressen umfassen beim SAM7 Controller nur die Bereiche des Flashspeichers und des SRAMs.
- Ein “Data Abort” wird durch den Zugriff auf eine ungültige Adresse während der Verarbeitung einer Instruktion (Load / Store) ausgelöst. Weiterhin müssen die Adressen für 32bit Zugriffe durch 4 teilbar und bei 16bit Zugriffen durch 2 teilbar sein.

Nach dem Auslösen eines Aborts werden die Interrupts deaktiviert und der Handler benutzt eigene UART Routinen, um die Werte aller Register, sowie einen Auszug des Stackframes

auszugeben. Somit ist der Handler nicht auf die fehlerfreie Funktion eines anderen Subsystems angewiesen. Da das Wiederherstellen eines gültigen Zustandes nicht möglich ist, wird nach Beendigung der Ausgabe die CPU angehalten.

```
ARM7TDMI 16bit Undefined Address Data Write Abort
(ASR:0x00020501 AASR:0x00a00246)
-- Registerdump (cpsr:0x60000013 - SVC):
r0:0x00a0023c r1:0x002001e4 r2:0x00000000 r3:0x0020064c
r4:0x00000400 r5:0x00100904 r6:0x00000000 r7:0x00000000
r8:0x00000001 r9:0x00000000 sl:0x00200234 fp:0x00000000
ip:0x001000f0 sp:0x0020fecc lr:0x001025fc pc:0x00102608
-- Stackdump:
0x0020fec0: 0x00100904 0x0000020c 0x00000025 0xfffff2ff
0x0020fed0: 0x001007c4 0x00200001 0x00100258 0x001000c4
0x0020fee0: 0xf1540d41 0xb06bf2ae 0x475d2469 0xa5a25ace
0x0020fef0: 0xad6e0350 0xf0a0e097 0x179615c0 0xc9b4bba1
```

Im Beispiel wurde ein schreibender 16Bit Zugriff vom Memory Controller abgefangen. Das Register “AASR” enthält die ungültige Adresse, auf die zugegriffen wurde. Der Programm Counter (pc) zeigt auf den verursachenden Befehl.

6.2.2 AD-Wandlung der Sensorsignale

Über den internen AD-Wandlers des SAM7 Controllers werden die Sensorsignale der Gyroskope und der Beschleunigungssensoren digitalisiert. Zusätzlich wird die Batteriespannung erfasst.

Die Implementierung nutzt einen der Peripherial DMA Controller (PDC) des SAM7. Mit Hilfe eines PDC können Daten zwischen dem Speicher und der on-chip Peripherie transferiert werden, ohne dabei die CPU durch Interrupts zu belasten. Dadurch kann die Übertragungsgeschwindigkeit drastisch erhöht werden, während die CPU für andere Aufgaben genutzt wird. Der AD-Wandler kann auf diese Weise alle benötigten Analogkanäle digitalisieren, die Ergebnisse in das interne SRAM schreiben und erst dann einen Interrupt zur gemeinsamen Verarbeitung der Daten auslösen.

Die Umwandlung der Sensorsignale wird von der Fluglageregelung per Software gestartet. Ein Start durch einen Hardwaretimer wäre möglich, erfordert aber eine zusätzliche Synchronisation zwischen der Fluglageregelung und dem PDC. Die AD-Wandlung wird dabei mit einer Genauigkeit von 10bit ausgeführt, der PDC überträgt diese Ergebnisse als 16bit Words.

Für die Digitalisierung aller Kanäle werden ca. 19μs benötigt, danach werden von den erfassten Werten die Nulloffsets der Sensoren abgezogen. Die so erhaltenen Messwerte sind nun vorzeichenbehaftet und erleichtern die Verarbeitung in der Fluglageregelung.

Die Nulloffsets ergeben sich aus den Ruhepegeln der Sensorensignale. Zur Kalibrierung werden die Mittelwerte aus 1000 Einzelmessungen verwendet. Während dieser Messungen muss der Quadrocopter ruhig auf einer waagerechten Fläche stehen. Die so ermittelten Offsets der Sensoren werden im EEPROM persistent abgespeichert.

6.2.3 Auswertung des RC-Summensignals

Zur Auswertung der Fernsteuerungskanäle wird das Summensignal des Empfängers von der ersten Timerbaugruppe des SAM7 Controllers ausgewertet. Dieser 16bit Timer wird mit einer Frequenz von 1.5MHz (48MHz /32) im Capturemode betrieben. Daraus ergibt sich eine maximal erfassbare Periode von ca. 44ms, in der Impulsverläufe mit einer Auflösung von 667ns ermittelt werden können.

Je nach Fernsteuerung besteht das Summensignal aus 3-9 Impulsen und wird alle 22ms erneut übertragen (siehe auch Abbildung 5.5). Die Timerbaugruppe ist so konfiguriert, dass bei einer negativen Flanke der erreichte Zählerstand in ein Captureregister gesichert wird. Danach wird der Zähler auf Null gesetzt und ein Interrupt ausgelöst. Der Registerwert enthält somit den Abstand zwischen zwei Impulsen, was einer einzelnen Steuerfunktion der Fernsteuerung entspricht (Abbildung 6.4).

In der Interrupt Service Routine werden die Impulse gezählt und ihre Impulsbreiten überprüft. Ein gültiges Signal ist durch eine Impulsbreite von 1-2ms definiert, wobei die genauen Werte abhängig von der Fernsteuerung sind. Werden über einen Zeitraum von 2.5ms keine Impulse erkannt, wird ebenfalls ein Interrupt ausgelöst. Dies dient der Erkennung der Synchronisationspause zwischen zwei Summensignal-Frames. In der zugehörigen Interrupt Service Routine wird der Kanalzähler zurückgesetzt und die Auswertung des Signals beginnt erneut mit Kanal 1. Ein Empfangsausfall wird erkannt, wenn über einen Zeitraum von 100ms kein gültiges Summensignal ausgewertet werden konnte.

Aus den erfassten Impulsbreiten werden zwei gleitende Mittelwerte mit unterschiedlicher Ordnung gebildet. Der erste Filter dient der feinfühligen Steuerung und bildet einen Mittelwert über die letzten 32 Summensignal-Frames. Der zweite Filter betrachtet nur die letzten 4 Frames und ermöglicht eine schnellere Reaktion auf die Steuerbefehle. Zwischen diesen beiden Filtern wird anhand der Geschwindigkeit der Steuerungsbewegungen umgeschaltet. Je nach Ordnung erwies sich ein einzelner gleitender Mittelwert als zu träge oder erlaubte keine feinfühlige Steuerung im Schwebezustand, daher wurden zwei getrennte Filter mit einer adaptiven Umschaltung implementiert.

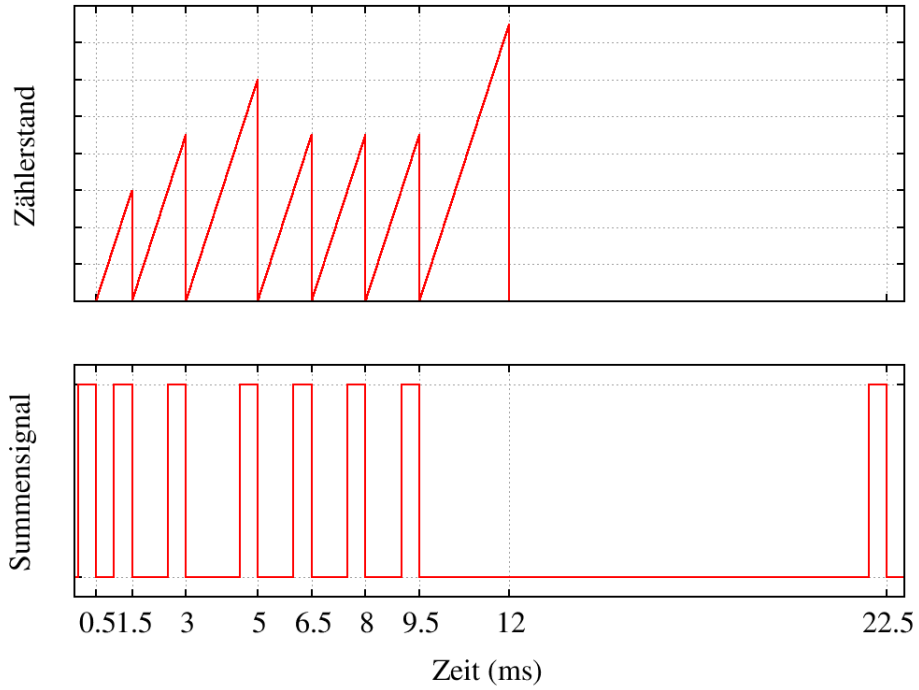


Abbildung 6.4: Summensignal und Zählerverlauf

Für die Erzeugung, der in der Lageregelung verwendeten Steuerungssignale, werden die gefilterten Impulsbreiten auf einen definierten Wertebereich skaliert. Zu diesem Zweck muss neben den min/max Werten der Impulsbreiten auch deren Ruhelage bekannt sein. Diese Kalibrierungsdaten werden im EEPROM gespeichert und müssen nur bei einem Wechsel der Fernsteuerung geändert werden.

Neben den analogen Werten für die Winkelvorgabe der Lageregelung werden aus den Impulsbreiten digitale Steuerfunktionen abgeleitet. Diese Funktionen werden auf Basis fester Schwellwerte erzeugt und können für einfache Steueraufgaben genutzt werden (Motoren an/aus).

6.2.4 USB-Kommunikation

Die Kommunikation über den USB-Anschluss erfolgt vollständig in einer Interrupt Service Routine. Der Controller bietet für den USB Device Port keine Unterstützung in Form eines PDC an, daher müssen alle Datenbytes einzeln von der CPU transferiert werden.

Für die Definition der USB-Datenstrukturen und Konstanten wird auf die Headerfiles des Linux-Kernels [10] zurückgegriffen. Diese Datenstrukturen sind ein Teil des USB-Protokolls,

daher ist ihr Aufbau auf allen Plattformen identisch und kann ohne Anpassung übernommen werden.

Die Emulation der seriellen Schnittstelle nach der CDC Spezifikation wurde nicht vollständig implementiert, so werden z. B. die nicht benötigten Funktionen für das Lesen und Schreiben von Handshakesignalen ignoriert. Um die ISR des USB Device Ports von der Telemetrie-Softwarekomponente zu entkoppeln, wird der Datentransfer über zwei FIFO Speicher gepuffert.

6.2.5 USB-Bootloader

Neben der Emulation einer seriellen Schnittstelle wurde auch ein Device Firmware Upgrade (DFU) Interface implementiert. Dieses Protokoll beschreibt einen Austausch von Firmware für Devices, ohne dabei auf spezielle Programmierschnittstellen angewiesen zu sein.

Der USB-Host kann über ein spezielles DFU-Kommando die Ausführung der eigentlichen Applikation auf dem Controller beenden und einen Bootloader starten lassen. Dieser Bootloader stellt dem USB Host daraufhin eine Reihe von Programmierschnittstellen bereit. So lassen sich mit dieser Implementierung der Flash-Speicher des Controllers, das I2C-EEPROM, sowie die Flash-Speicher der vier Motorregler einzeln auslesen und programmieren (Abbildung 6.5).

Für den Bootloader lassen sich große Teile der bereits entwickelten Softwarekomponenten wiederverwenden. Die DFU-Statemachine des Bootloaders basiert auf dem openPCD [13] Projekt.

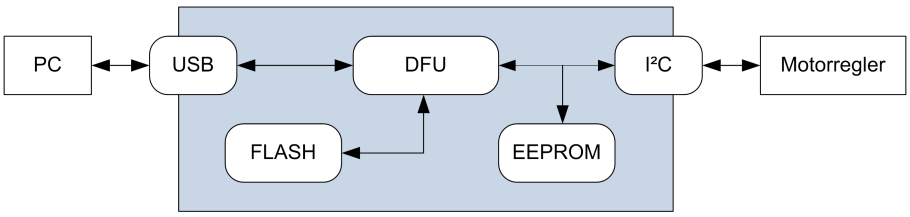


Abbildung 6.5: USB Bootloader

Um einen exklusiven Zugriff auf den internen Flash-Speicher zu ermöglichen, kopiert sich der Bootloader in das SRAM des Controllers und verlegt die Interruptvektoren ebenfalls dorthin. Durch die Verlegung der Vektoren ist sichergestellt, dass durch einen Interrupt nicht auf den deaktivierten Flash-Speicher zugegriffen wird. Für die Programmierung der Motorregler ist auf diesen ein eigener Bootloader implementiert, der einen Zugriff über den I2C-Bus ermöglicht (Protokoll im Anhang B)

6.2.6 Telemetrie

Da das Telemetriesystem auf dem Controller nebenläufig zur Fluglageregelung arbeitet, muss es jederzeit auf die Variablen der beobachteten Messwerte zugreifen können. Daraus folgt, dass sich der Speicherort der Messwerte zu keinem Zeitpunkt ändern darf. In der Programmiersprache C trifft diese Bedingung nur für statische oder globale Variablen zu. Lokale Variablen werden auf dem Stack allokiert und verlieren damit ihre Gültigkeit, wenn die Funktion, in der sie deklariert worden sind, verlassen wird.

Für die Speicherung der verfügbaren Messwerte, die über das Telemetriesystem abgefragt werden können, wurde im ersten Ansatz eine zur Laufzeit erzeugte Liste vorgesehen. Die Listenelemente enthalten Zeiger auf die Messwerte, sowie weitere Informationen zum Messwert.

```
struct tdc_var {
        struct tdc_var *list;   /* pointer to next tdc_var */
        uint32_t flags;         /* type information */
        void *ptr_to_var;       /* pointer to static value */
        const char *name;       /* name/description */
};
```

Jede Variable wird zur Laufzeit beim Telemetriesystem registriert und kann danach verwendet werden. Eine Sortierung der Liste ist nicht erforderlich, bei einem Zugriff auf eine Variable muss diese allerdings erst in der Liste gefunden werden. Da die Liste zur Laufzeit erstellt wird, muss das Listenelement im RAM gespeichert werden, obwohl sich die Nutzdaten niemals ändern. Daher wurde in einem zweiten Ansatz eine Lösung erarbeitet, die einen schnellen Zugriff auf die Elemente ermöglicht und keinen zusätzlichen RAM Speicher benötigt. Zu diesem Zweck wird vom C-Compiler für die Zeiger auf die Messwerte und den weiteren Informationen eine eigene Speichersektion erstellt. Über ein angepasstes Linkerscript werden diese Elemente der Reihe nach im Flash-Speicher abgelegt. Da die Position dieser Speichersektion und die Größe eines einzelnen Elementes zur Laufzeit bekannt sind, kann auf alle Variablen über einen Index zugegriffen werden:

```
struct tdc_value *value = &_tdc_value_table[id];
```

Um die Erstellung von Telemetrievariablen im Sourcecode zu vereinfachen, werden diese mit Hilfe eines Makros angelegt. Neben einer Typüberprüfung der Variablen wird auch das Element in der zusätzlichen Speichersektion ablegt.

```
static int32_t nick_integral;
TDC_INT32(nick_integral, 'Integral Nick Value');
```

Das Telemetriesystem wird laufend von der Hauptschleife der Firmware aufgerufen und kommuniziert über zwei FIFOs mit dem USB Device Port. Sobald vollständige Telemetrieanfragen im IN FIFO vorhanden sind, werden diese verarbeitet und beantwortet. Die Eingangsnachrichten werden jeweils als eine Transaktion verarbeitet. Die Anfrage verbleibt im IN FIFO, bis alle dadurch erzeugten Antworten in den OUT FIFO geschrieben sind (Abbildung 6.6).

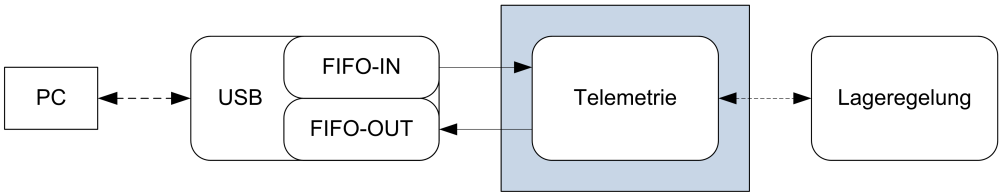


Abbildung 6.6: Telemetrie Kommunikation über USB

Kann der OUT FIFO keine weiteren Nachrichten aufnehmen, wird die Verarbeitung der Telemetriedaten unterbrochen. Wurden die Daten im OUT FIFO durch den USB Device Port an den Host ausgeliefert, ist wieder Speicher verfügbar. Somit kann die Verarbeitung mit der ersten Anfrage im IN FIFO wieder aufgenommen werden. Bei Anfragen, die mehrere Antwortpakete erzeugen, wird zusätzlich die genaue Position der Unterbrechung gespeichert. So werden bei der Wiederaufnahme der Verarbeitung keine doppelten Datenpakete versendet.

6.2.7 Lageregelung

Bei der Implementierung der Lageregelung wird auf effiziente Programmierung der Berechnungen geachtet. Der SAM7 Controller hat keinen Floatingpoint Coprozessor, daher müssen Fließkomma-Operationen durch eine aufwendige Softwareemulation nachgebildet werden. Bei der Lageregelung auf dem Controller wird vollständig auf Fließkomma-Arithmetik verzichtet, stattdessen wird mit 32bit Ganzzahlen gerechnet. Bei der Rechnung mit Ganzzahlen muss weiterhin beachtet werden, dass der ARM7TDMI Core keine Befehle für Divisionen unterstützt. Diese werden auch durch eine Softwareemulation durchgeführt. Bei der Lageregelung wird versucht die Divisionen durch Umstellen der Rechnungen und durch geschickte Wahl der Operanden zu vereinfachen. Diese Divisionen können vom Compiler in eine arithmetische Shiftoperation umgewandelt werden.

Im Folgenden wird auf die Implementierung der Lageregelung im Näheren eingegangen. Die aktuellen Sensorsignale werden jede Millisekunde vom AD-Wandler abgefragt. Aus den Drehwinkelgeschwindigkeiten der Gyroskope (ADC_GYRO_*) werden durch numerische Integration die Lagewinkel der drei Rotationsachsen errechnet. Zusätzlich wird aus dem Gier-Kanal der Fernsteuerung (STICK_GIER) das für die Gierachse benötigte Vergleichsintegral gebildet:

```
nick_integral += adc_result[ADC_GYRO_NICK];
roll_integral += adc_result[ADC_GYRO_ROLL];
gier_integral += adc_result[ADC_GYRO_GIER];
stick_gier_integral += rc.chan[STICK_GIER];
```

Die von den Beschleunigungssensoren ermittelten absoluten Lagewinkel (ADC_ACC_*) werden zur Korrektur der Lagewinkelintegrale von Nick- und Rollachse verwendet. Wie in Kapitel 5.2.5 beschrieben, wird der Einfluss der Sensoren als linear (ACC_ADJUST) angesehen und nicht über den Arkussinus der Beschleunigungen berechnet. Der Einfluss der Beschleunigungssensoren auf die Integrale beträgt hier ca. 0.1% (1/1024):

```
nick_integral = ((nick_integral * 1023) + (adc_result[ADC_ACC_NICK] * ACC_ADJUST * 1)) / 1024;
roll_integral = ((roll_integral * 1023) + (adc_result[ADC_ACC_ROLL] * ACC_ADJUST * 1)) / 1024;
```

Zur Lageregelung werden neben den Integralen die momentanen Drehwinkelgeschwindigkeiten verwendet. Diese dienen zur schnellen Ausregelung von äußeren Störeinflüssen. Anstatt die Drehwinkelgeschwindigkeiten durch Diffentialrechnung der Integralwerte im folgenden PID-Regler zu errechnen, werden die Signale der Gyroskope (ADC_GYRO_*) verwendet. Dies ersetzt somit den D-Anteil des PID-Reglers:

```
int32_t nick = (nick_integral / 256) + (adc_result[ADC_GYRO_NICK] / 4);
int32_t roll = (roll_integral / 256) + (adc_result[ADC_GYRO_ROLL] / 4);
int32_t gier = (gier_integral / 256) + (adc_result[ADC_GYRO_GIER] / 4);
```

Durch Vergleich mit den Fernsteuerungssignalen werden die Reglerabweichungen der Lageregelung berechnet und den PID-Reglern als Eingangsgröße übergeben. Die Parameter der PID-Regler sind in getrennten Datenstrukturen (pid_data_*) gespeichert. Die errechneten Stellwerte der drei Achsen werden zusammen mit dem Gas-Kanal der Fernsteuerung (STICK_GAS) von einem Motormixer verrechnet. Die sich daraus ergebenen Sollwerte werden über den I2C-Bus an die Motorregler gesendet.

```
int32_t mixer_nick = pid_ctrl(&pid_data_nick, rc.chan[STICK_NICK] - nick);
int32_t mixer_roll = pid_ctrl(&pid_data_roll, rc.chan[STICK_ROLL] - roll);
int32_t mixer_gier = pid_ctrl(&pid_data_gier, stick_gier_integral - gier);
motor_mixer(rc.chan[STICK_GAS], mixer_nick, mixer_roll, mixer_gier);
```

6.3 Telemetrie Anwendung

Da die komplette Entwicklung und Implementierung der Lageregelung unter Linux stattfand, wurde auch als Anzeige der Telemetriedaten eine einfache GTK2/Linux Applikation erstellt.

Über eine tabellarische Form werden die Werte und Beschreibungen aller übertragenen Telemetrievariablen dargestellt (Abbildung 6.7). Auf Basis der gtkdatabox [9] Bibliothek wurde eine graphische Ansicht erstellt, die es ermöglicht bis zu acht Messwerte bei einer Samplerate von bis zu 100Hz darzustellen (Abbildung 6.8).

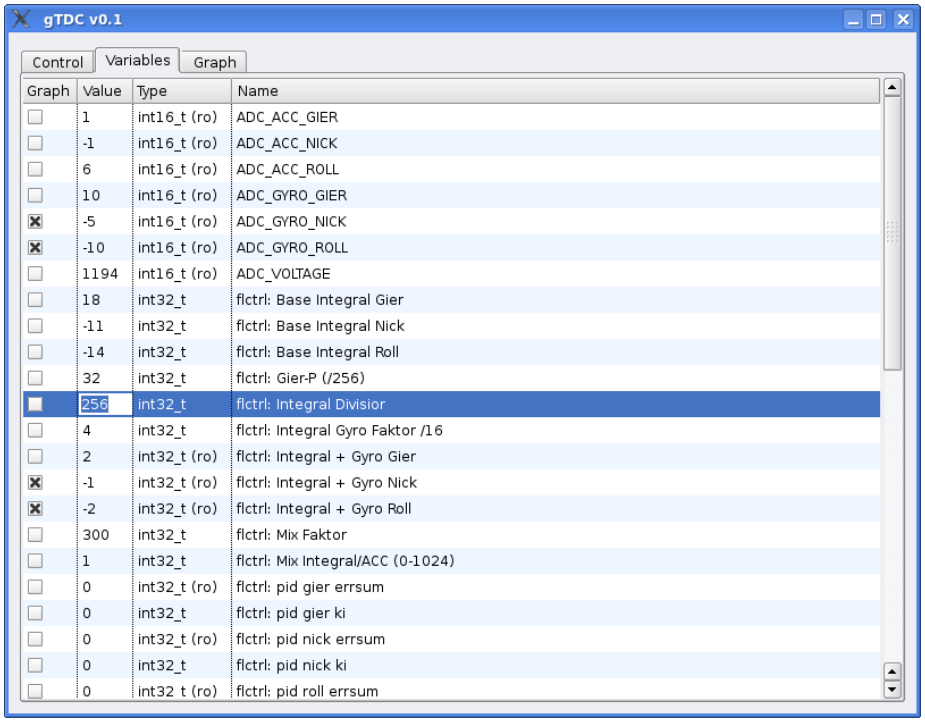


Abbildung 6.7: Übersicht der Variablen in der Telemetrieanwendung

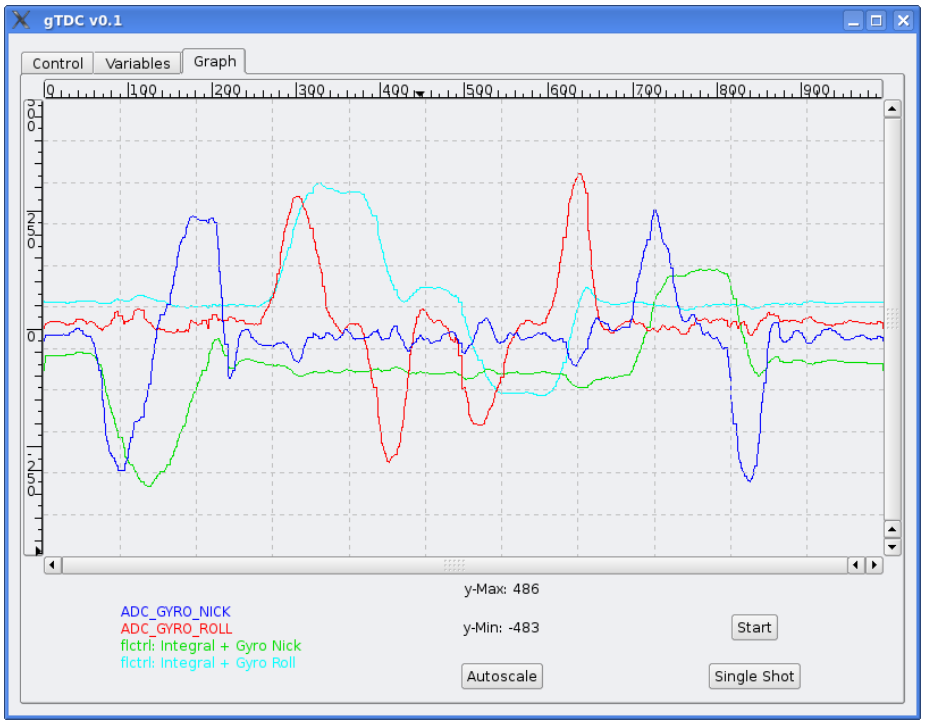


Abbildung 6.8: Graphische Darstellung in der Telemetrieanwendung

7 Testflug

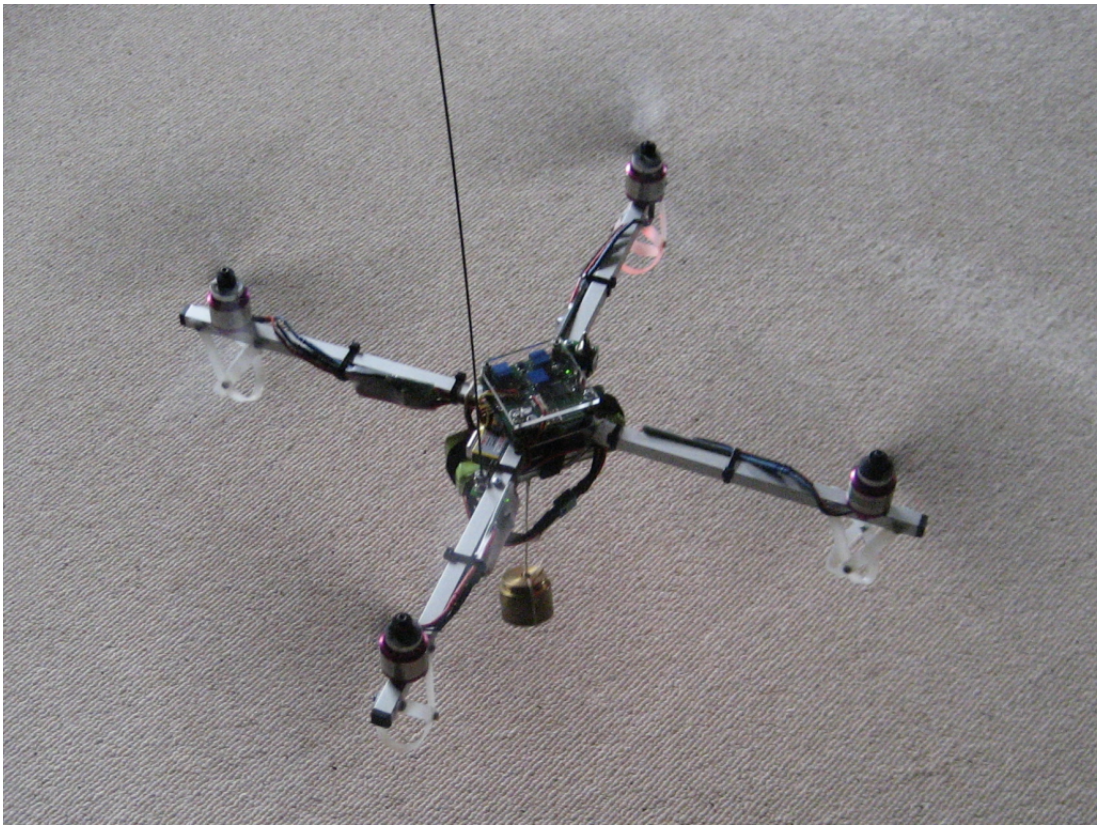


Abbildung 7.1: Quadrocopter Flug mit 180g Nutzlast

Für die Parameterfindung der Lageregelung kann der Quadrocopter mit der Hand festgehalten und die einzelnen Steuerbewegungen überprüft werden. In dieser Position lässt sich die Lageregelung gut erproben und der Einfluss der einzelnen Parameter verifizieren. Steuerbewegungen der Nick- und Rollachse lassen sich dabei einfach erkennen, da diese direkt vom Propellerschub erzeugt werden. Die Bewegungen der Gierachse sind schwächer, da sie nur durch das Ungleichgewicht der Motorendrehmomente erzeugt werden. Für einen längeren Testbetrieb ist die Verwendung eines ausreichend dimensionierten Labornetzteils (12V / 10A) erforderlich.

Vor einem Start des Quadrocopters müssen die internen Sensoren kalibriert werden, hierbei ist darauf zu achten, dass der Quadrocopter auf einer ebenen Fläche steht. Da die Ausgangspegel der Gyroskope in den ersten 1-2min nach dem Einschalten stark driften, muss die Kalibrierung dieser Sensoren ggf. wiederholt werden. Um eine einheitliche Steuerung zu verwenden, werden die in Tabelle 7.1 aufgelisteten Funktionen vom Mikrokopter Projekt übernommen.

Gas/Gier Kanal	**Funktion**
oben / links	Gyroskope kalibrieren (Nullpunkte ermitteln)
oben / rechts	Beschleunigungssensoren kalibrieren (Nullpunkte ermitteln)
unten / links	Motoren abschalten
unten / rechts	Motoren einschalten (Leerlauf)

Tabelle 7.1: Übersicht der Steuerfunktionen

Bei der Validierung der ermittelten Parameter im freien Flug stellte sich eine höhere Schwingungsneigung des Lagereglers heraus. Durch Verringern der P-Anteile der Achsenregler konnten diese Schwingungen im Schwebeflug eliminiert werden. Bei schnellem Flug, insbesondere bei einem schnellen Sinkflug, kommt es weiterhin zu einem starken Schwingen.

Die Abbildung 7.1 zeigt den freien Flug des Quadrocopters mit 180g Nutzlast (ca. 25% des Eigengewichts). Die Stabilität des Schwebefluges wird durch dieses zusätzliche Gewicht nicht verändert. Eine Nutzlast von 360g konnte zwar angehoben, aber nicht über die Grenze des Bodeneffekts gehoben werden. Der dynamische Flug mit einer solchen Nutzlast ist nur eingeschränkt möglich. Die Flugdauer ohne Nutzlast beträgt ca. 13min, mit Nutzlast reduziert sich die Flugdauer auf ca. 10min.

8 Zusammenfassung

Das Ziel dieser Arbeit ist es, eine inertiale Lageregelung für einen Quadrocopter zu entwickeln. Aus dieser Aufgabe heraus wird in Kapitel 4.1 ein Konzept einer solchen Lageregelung erstellt. Dieses Konzept dient als Ausgangspunkt für das in Kapitel 5.1 erarbeitete Hardwaredesign, das einen 32bit ARM7TDMI Controller als zentrale Komponente verwendet. Im Kapitel 5.1.8 entsteht aus dem Hardwaredesign ein Platinenlayout, das von einem externem Hersteller gefertigt wurde. Auf dieser Platine werden alle Sensoren montiert, die für eine Flugstabilisierung eines Quadrocopters benötigt werden. Auf Basis dieser Hardware wird in Kapitel 6.1 ein Quadrocopter aufgebaut.

In Kapitel 5.2 werden in Abstimmung mit der verwendeten Hardware die einzelnen Softwarekomponenten der Fluglageregelung identifiziert. Danach wird in Kapitel 6.2 die Implementierung der einzelnen Komponenten in der Programmiersprache C erläutert. Zusätzlich zu der Firmware auf dem Controller der Fluglageregelung wird in Kapitel 5.3 ein Telemetriesystem entwickelt und in Kapitel 6.3 implementiert.

In Kapitel 7 werden die Parameter für die Lageregelung ermittelt. Durch Testflüge wird schliesslich das Flugverhalten des entstandenen Quadrocopters erprobt. Im Schwebeflug kann die Lageregelung den Quadrocopter stabilisieren, bei schnellerem Flug oder bei schnelleren Kursänderungen arbeitet das System nicht zufriedenstellend. Die Lageregelung versetzt den Quadrocopter dabei in starke Schwingungen.

Die Tests haben gezeigt das die Stabilisierung eines Quadrocopters mit der entwickelten Lageregelung möglich ist. Augrund des Zeitmangels konnten aber nicht für alle Flugsituationen passende Parameter ermittelt werden.

8.1 Verbesserungsvorschläge

Durch eine Erweiterung der Hardware mit zusätzlicher Sensorik kann die Lageregelung weiter verbessert werden. So kann über die exakte Messung des Luftdrucks eine Regelung der Flughöhe erfolgen. Mit Hilfe eines elektronischen Kompasses ist eine driftfreie Stabilisierung der Gierachse möglich. Zusammen mit der Auswertung eines Navigationssystems wie GPS, lässt sich so eine vollständig autonome Lageregelung implementieren.

Literaturverzeichnis

[1] microdrones GmbH. URL http://www.microdrones.com/.

[2] MikroKopter Projekt. URL http://www.mikrokopter.de/.

[3] powerframe.de - Inovative Produkte für Quadrokopter. URL http://www.powerframe.de/.

[4] Eclipse C/C++ Development Tooling - CDT, . URL http://www.eclipse.org/cdt/.

[5] EAGLE Layout Editor, . URL http://www.cadsoft.de/.

[6] Eclipse - Open development platform, . URL http://www.eclipse.org/.

[7] FreeRTOS, . URL http://www.freertos.org/.

[8] GCC, the GNU Compiler Collection, . URL http://gcc.gnu.org/.

[9] GtkDatabox, . URL http://www.eudoxos.de/gtk/gtkdatabox/.

[10] Linux Kernel, . URL http://www.kernel.org/.

[11] Nut/OS, . URL http://www.ethernut.de/en/software/index.html.

[12] Open On-Chip Debugger (OpenOCD), . URL http://openfacts.berlios.de/index-en.phtml?title=Open_On-Chip_Debugger.

[13] OpenPCD, . URL http://www.openpcd.org/.

[14] Wikipedia: Energiedichte, . URL http://de.wikipedia.org/wiki/Energiedichte.

[15] Wikipedia: JTAG, . URL http://de.wikipedia.org/wiki/EIA-232.

[16] Wikipedia: Quadrocopter, . URL http://de.wikipedia.org/wiki/Quadrocopter.

[17] Wikipedia: RS232, . URL http://de.wikipedia.org/wiki/Joint_Test_Action_Group.

[18] Analog Devices, Inc. ADXRS300 Datasheet, 2004. URL http://www.analog.com/UploadedFiles/Data_Sheets/ADXRS300.pdf.

[19] ARM Ltd. ARM Architecture Reference Manual, 2005. URL http://www.arm.com/miscPDFs/14128.pdf.

[20] Atmel Co. AT24C256 Datasheet, 2007. URL http://www.atmel.com/dyn/resources/prod_documents/doc0670.pdf.

[21] Atmel Co. AT91SAM7S Schematic Check List, 2007. URL http://atmel.com/dyn/resources/prod_documents/doc6258.pdf.

[22] Atmel Co. AT91SAM7S256 Datasheet, 2007. URL http://www.atmel.com/dyn/resources/prod_documents/doc6175.pdf.

[23] Craig Peacock. BeyondLogic: USB in a Nutshell, 2002. URL http://www.beyondlogic.org/usbnutshell/usb-in-a-nutshell.pdf.

[24] Lasse Klingbeil. Entwicklung eines modularen und skalierbaren Sensorsystems zur Erfassung von Position und Orientierung bewegter Objekte, 2006.

[25] Maxim Integrated Products, Inc. MAX3232 Datasheet, 2007. URL http://datasheets.maxim-ic.com/en/ds/MAX3222-MAX3241.pdf.

[26] Sergej Meier. Erfassung und Verarbeitung von Sensormesswerten für die Stabilisierung in Längs- und Querrichtung eines holonomen Fahrzeugs bei Kurvenfahrt, 2006.

[27] Murata Manufactuaring Co., Ltd. ENC03J Datasheet, 2006. URL http://www.murata.com/catalog/s42e.pdf.

[28] National Semiconductor Co. LM324 Datasheet, 2004. URL http://cache.national.com/ds/LM/LM124.pdf.

[29] Philips Semiconductors. I2C Specification, 1995. URL http://www.nxp.com/acrobat_download/various/I2CBUS_SPEC.pdf.

[30] STMicroelectronics. LIS3L02AS4 Datasheet, 2005. URL http://www.st.com/stonline/books/pdf/docs/10221.pdf.

[31] USB Implementers Forum. Universal Serial Bus Revision 2.0 Specification, 2000. URL http://www.usb.org/developers/docs/usb_20_040908.zip.

[32] USB Implementers Forum. USB Class Definitions for Communication Devices v1.1, 1999. URL http://www.usb.org/developers/devclass_docs/usbcdc11.pdf.

[33] USB Implementers Forum. USB Class Specification for Device Firmware Upgrade v1.1, 2004. URL http://www.usb.org/developers/devclass_docs/DFU_1.1.pdf.

Glossar

ADC	Analog Digital Converter
ARM	Prozessorarchitektur, Markenname der Fa. ARM
AVR	Prozessorarchitektur, Markenname der Fa. Atmel
CAN	Controller Area Network (Bus)
CSBGA	Cost Savings Ball Grid Array (Bauteilgehäuseart)
EEPROM	Electrically Erasable Programmable Read Only Memory
FIFO	First-In, First-Out (Speicher)
I2C	Inter-IC (Bus), serieller 2-Draht Datenbus
IrDA	Infrared Data Association, Infrarot Übertragungsstandard
ISR	Interrupt Service Routine
JTAG	Joint Test Action Group
LiPo	Lithium-Polymer (Akkumulator)
LQFP	Low-profile Quad Flat Package (Bauteilgehäuseart)
MCU	Microcontroller Unit
MEMS	Micro-Electro-Mechanical System
NiMh	Nickel-Metall-Hydrid (Akkumulator)
PDC	Peripherial DMA Controller
PID	Proportional-Integral-Differential (Regler)
PLL	Phase-locked loop, phasengekoppelter Regelkreis zur Frequenzerzeugung
PPM	Pulsphasenmodulation

PWM Pulsweitenmodulation

SMD Surface-mounted Device, oberflächenmontierbares Bauteil

SRAM Static Random Access Memory

UART Universal Asynchronous Receiver Transmitter

USART Universal Synchronous/Asynchronous Receiver Transmitter

USB Universal Serial Bus

A Hardware Design

JTAG Adapter

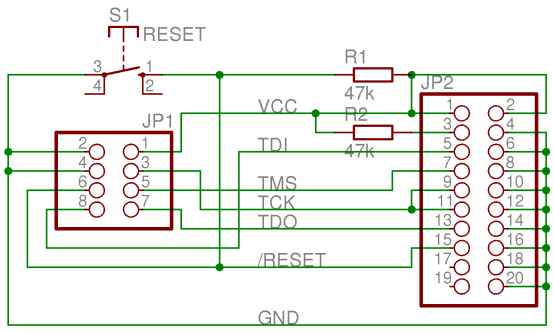


Abbildung A.1: Schaltplan JTAG-Adapter

Über diesen JTAG Adapter kann ein Standard 20pol ARM-JTAG Interface an die Fluglageregelung angeschlossen werden.

Schaltplan der Fluglageregelung

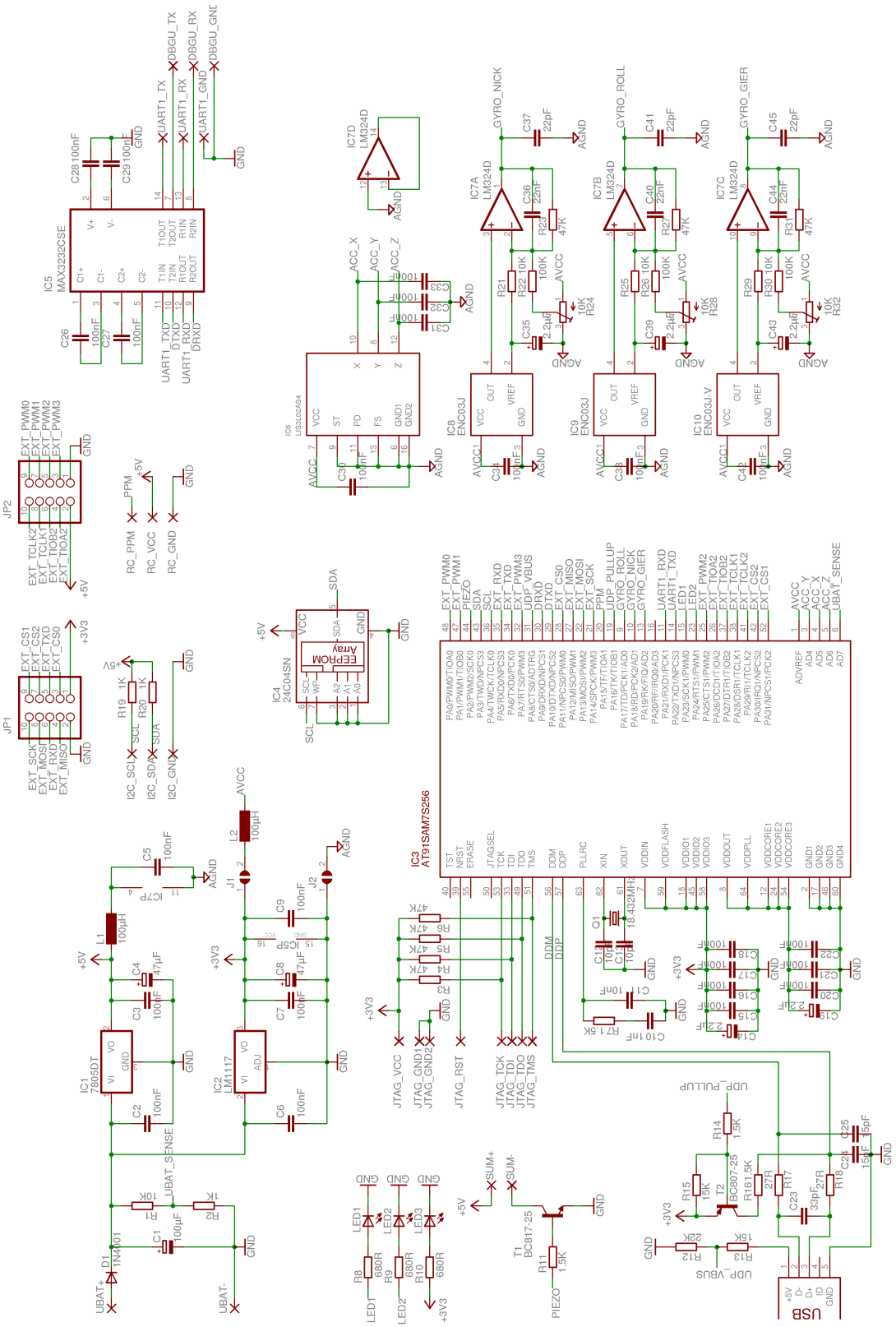


Abbildung A.2: Vollständiger Schaltplan

B Kommunikationsprotokolle

I2C-Protokoll für Motor Regler

Die Motorregler haben zwei Betriebsmodi: Nach einem Reset wird ein Bootloader gestartet, der auf Kommandos über den I2C-Bus wartet. Über den Bootloader können der Flash- und EEPROM-Speicher programmiert und gelesen werden. Dadurch ist es möglich, die Motorregler Firmware ohne zusätzlichen Programmieradapter auszutauschen.

Werden keine Kommandos über den I2C-Bus empfangen und es befindet sich eine gültige Firmware im Flashspeicher, wird diese nach einem Timeout ausgeführt.

Im Folgenden sind die implementierten Kommandos für die I2C-Kommunikation aufgeführt.

STA+W	I2C-Start, 7bit I2C-Adresse, Daten werden zum Slave geschrieben
STA+R	I2C-Start, 7bit I2C-Adresse, Daten werden vom Slave gelesen
STO	I2C-Stop

Get Info

STA+W	0x10	STA+R	{16 bytes}	STO

Das Kommando ist in Bootloader und Applikation implementiert.
Es wird ein 16 Byte Versionsstring zurückgelesen:

"TWIBOOT m8-v1.2"	Version des Bootloaders
"blctrl m8-v1.2"	Version der Applikation

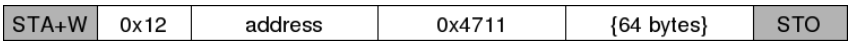
Get Signature

STA+W	0x11	STA+R	{4 bytes}	STO

Das Kommando ist nur im Bootloader implementiert.
Es werden 4 Signature Bytes zurückgelesen:

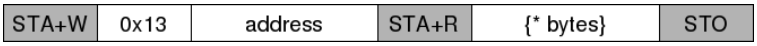
0x1E 0x93 0x07 0x00	AVR Signaturbytes des ATmega8

Write Flash Page



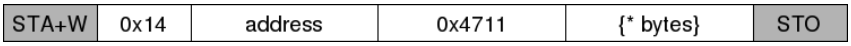
Das Kommando ist nur im Bootloader implementiert.
Es wird eine Page (64byte) an die Adresse <address> des Flash-Speichers geschrieben.
Ein Überschreiben des Bootloader-Bereiches ist nicht möglich.

Read Flash



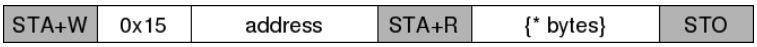
Das Kommando ist nur im Bootloader implementiert.
Es werden beliebig viele Bytes aus dem Flash-Speicher ab Adresse <address> gelesen.
Das Auslesen des Bootloader Bereiches ist möglich.

Write EEPROM



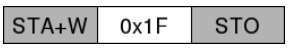
Das Kommando ist nur im Bootloader implementiert.
Es werden beliebig viele Bytes an die Adresse <address> des EEPROMs geschrieben.

Read EEPROM



Das Kommando ist nur im Bootloader implementiert.
Es werden beliebig viele Bytes aus dem EEPROM ab Adresse <address> gelesen.

Boot Application



Das Kommando ist nur im Bootloader implementiert.
Der Bootloader wird beendet und die Applikation gestartet.

Set PWM

STA+W	0x21	pwm	STO

Das Kommando ist nur in der Applikation Motorregler implementiert.
Setzt den Motorstellwert, bei 0x00 wird der Motor abgeschaltet.

Get Status

STA+W	0x22	STA+R	{8 bytes}	STO

Das Kommando ist nur in der Applikation Motorregler implementiert.

Byte 0x0	PWM Sollwert
Byte 0x1	PWM Istwert (durch Strombegrenzung evtl. niedriger)
Byte 0x2 - 0x3	momentane Stromaufnahme in mA
Byte 0x4 - 0x5	mementane Drehzahl in RPM
Byte 0x6 - 0x7	momentane Batteriespannung in mV

Set Parameters

STA+W	0x23	STA+R	{16 bytes}	STO

Das Kommando ist nur in der Applikation Motorregler implementiert.

Byte 0x0 - 0x1	interner Parameter (spinup_ticks)
Byte 0x2	interner Parameter (spinup_tick)
Byte 0x3	interner Parameter (spinup_step)
Byte 0x4	interner Parameter (spinup_wait)
Byte 0x5	interner Parameter (spinup_pwm)
Byte 0x6	minimaler PWM Sollwert (darunter wird Motor angehalten)
Byte 0x7	maximaler PWM Sollwert
Byte 0x8 - 0x9	Strombegrenzung in mA (PWM wird abgesenkt)
Byte 0xA - 0xB	Überstromabschaltung in mA (Motor wird abgeschatet)
Byte 0xC - 0xD	Unterspannungsabschaltung in mV (Motor wird abgeschaltet)

Dieser Parametersatz wird nichtflüchtig im EEPROM gespeichert und kann zum Anpassen an verschiedene Motoren verändert werden.

Get Parameters

STA+W	0x24	{16 bytes}	STO

Das Kommando ist nur in der Applikation Motorregler implementiert. Siehe “Set Parameters”.

Boot Loader

STA+W	0x2F	STO

Das Kommando ist nur in der Applikation Motorregler implementiert.
Die Applikation wird durch einen Reset beendet.

C Inhalt der CD

bachelorarbeit.pdf	Diese Arbeit als PDF Dokument
literatur/	Alle Dokumente die in der Arbeit erwähnt werden
source/sam7fc/	Quellcode der Lageregelung
source/sam7fc-bootloader/	Quellcode des Bootloaders auf der Lageregelung
source/gtdc/	Quellcode der Telemetrieanwendung
source/blmc/	Quellcode der Motorregler
source/twiboot/	Sourcecode der Bootloader auf den Motorreglern
eagle/sam7fc/	Schaltplan und Platinenlayout der Fluglageregelung
eagle/blmc/	Schaltplan und Platinenlayout der Motorregler
tools/	Bei der Entwicklung verwendete Tools (Compiler, JTAG, DFU)

Versicherung über Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Arbeit im Sinne der Prüfungsordnung nach §22(4) ohne fremde Hilfe selbstständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 27. Mai 2008	
Ort, Datum	Unterschrift