Hochschule Esslingen
University of Applied Sciences

Studienarbeit

Anbindung eines GPS-Moduls und Implementierung eines Treibers

Studiengang: Technische Informatik
Fakultät Informationstechnik

Max Pejs

Zeitraum:	1.10.2010 bis 19.6.2011
Erstprüfer:	Prof. Dr. Jörg Friedrich
Zweitprüfer:	Prof. Dr. Reinhard Malz
Betreuer:	B.Eng. Dionysios Satikidis

Inhaltsverzeichnis

1 Einführung

1.1 Vorstellung des Projekts "Quadrocopter"

Was ist eigentlich ein Quadrocopter? Ein Quadrocopter ist eine erweiterte Variante des Helikopters. Helikopter hat einen Zentralrotor und einen Heckrotor. Der Quadrocopter hingegen wird mit vier Rotoren angetrieben. Alle vier Rotoren sind horizontal angeordnet und über Kreuz an der Karoserie befestigt. Die Abbildung unten zeigt das Modell des Quadrocopters der Hochschule Esslingen.

Abbildung 1.1: Quadrocopter der HS Esslingen

Um die Summe der Drehmomente bei null zu halten werden zwei gegenüberliegende Rotoren im Uhrzeigersinn und die anderen zwei im Gegenuhrzeigersinn angetrieben. Um die Orientierung des Quadrocopters in der Luft zu erkennen unterscheidet sich ein Rotor farblich von den anderen drei.

Durch die Änderung der Drehzahl einzelner Rotoren wird eine Neigung des Quadrocopters erreicht. Dann wirkt die Hubkraft etwas seitlich auf den Boden und der Quadrocopter bewegt sich.

Ein Quadrocopter kann sich um alle drei Achsen drehen, um die Längsachse, die Querachse und die Hochachse. Im Vergleich zu einem Helikopter besitzt der Quadrocopter stabiles Flugverhalten.

Die Entwicklung des Quadrocopters an der Hochschule Esslingen ermöglicht den Studierenden kreative Ideen selbständig zu erforschen und umzusetzen.

1.2 Motivation

Die Steuerung des Quadrocopters läuft momentan manuell über eine Fernbedienung. Der Anstoß dieser Studienarbeit war dem Quadrocopter die "Sinne" zu geben, sodass er eigenständig fliegen kann. Einer der Sinne kann ein GPS-Empfänger repräsentieren, dass mithilfe des globalen Navigationssattelitensystem GPS eigene geografische Position bis auf einige Meter genau bestimmen kann. Er kann somit als Orientierungshilfe dienen. Einige GPS-Module haben sehr geringe Abmessungen, Gewicht und Stromverbrauch, sodass diese problemlos auch an einem Quadrocopter angebaut werden können.

So wäre es möglich eine Route durch sog. Chekpoints vorzugeben und der Quadrocopter wäre in der Lage dieser Route komplett selbständig zu folgen.

1.3 Aufgabenstellung

Gegenstand der hier vorgestellten Arbeit ist die Entwicklung eines Software-Treibers für die Quadrocopter-Software um eine Kommunikation zw. dem Quadrocopter und einem GPS-Device zu gewährleisten und die empfangenen GPS-Daten für Zwecke weiterer Verarbeitung vorzubereiten.

Die Entwicklung erfolgte nach dem international anerkannten V-Modell. In dieser Studienarbeit sind Schritte der Treiber-Entwicklung detailliert beschrieben.

Bei der Studienarbeit soll ein vorgegebener GPS-Empfänger angeschlossen werden und ein Hardware-Treiber entworfen und implementiert werden.

1.4 V-Modell

Die Entwicklung der Treiber-Software erfolgte bei dieser Studienarbeit nach dem V-Model. Das V-Modell beschreibt die Vorgehensweise für eine erfolgreiche Planung und Durchführung von Projekten. Das Model gilt international als Entwicklungsstandard für IT-Systeme. Das V-Modell legt fest, was zu tun ist und wie die Aufgaben durchgeführt werden sollen. Der Begriff "V-Model" wird von der V-förmigen Darstellung der Projektelemente abgeleitet. Als Projektelemente bezeichnet man Spezifikation, Analyse, Realisierung und Integration. [7]

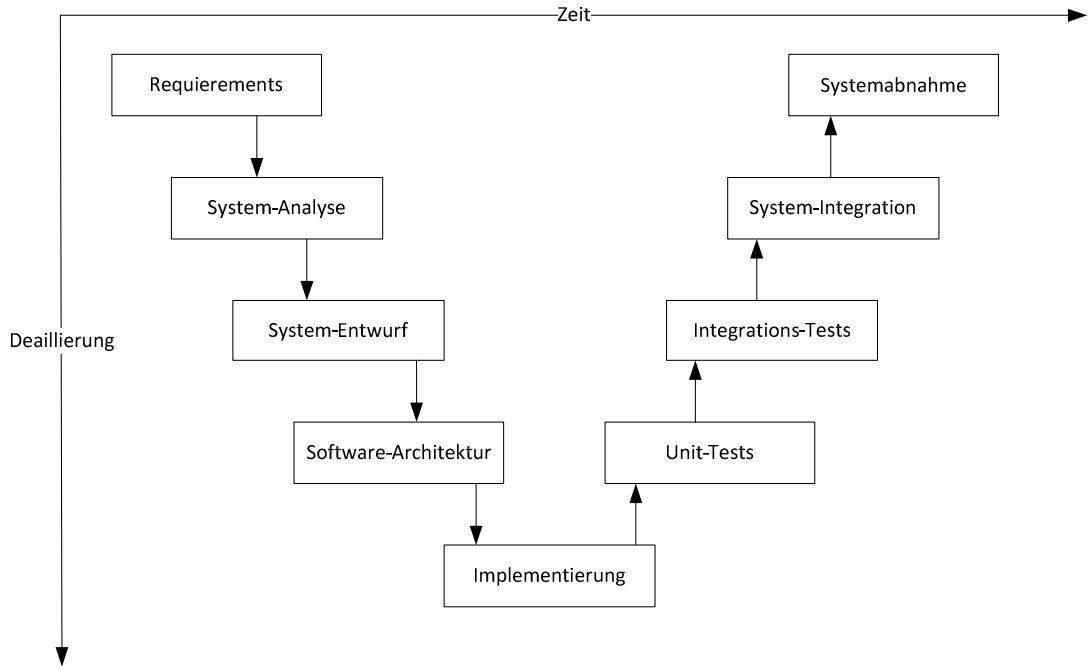

Abbildung 1.2: V-Modell

1.5 Entwicklungssoftware

Während der Entwicklung wurden zwei Entwicklungsumgebungen verwendet Microsoft Visual Studio 2010 und CodeWarrior. Die Implementierung erfolgte in der Programmiersprache C. Zum Zeichnen der Diagramme wurde Microsoft Office Visio 2010 verwendet.

1.6 Entwicklungshardware

Für die Entwicklung waren zwei Hardwarekomponenten vorgegeben:

- Ein Evaluierungsboard DEMO9S12XDT512 mit einem Freescale Prozessor

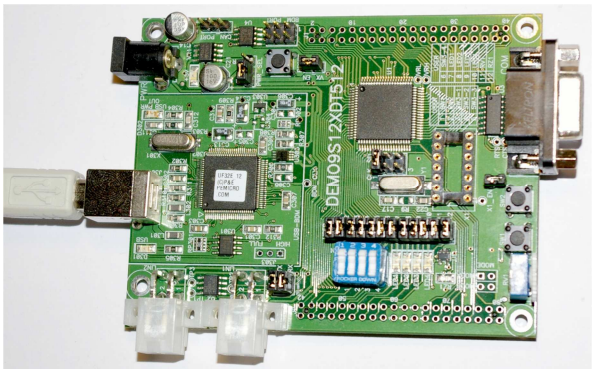

Abbildung 1.3: Evaluierungsboard

- Ein GPS-Empfänger ls20031 von LOCOSYS Technology

Abbildung 1.4: GPS-Empfänger

Nähere Informationen zur Hardware können den Datenblättern entnommen werden.

2 Grundlagen

2.1 GPS

Ein Globales Navigationssatellitensystem (engl. **g**lobal **p**ositioning **s**ystem), offiziell NAVSTAR GPS, ist ein System zur Positionsbestimmung und Zeitmessung. Es wurde seit den 1970er-Jahren vom US-Verteidigungsministerium entwickelt und steht seit 2000 auch für zivile Zwecke zur Verfügung. GPS hat sich als das weltweit wichtigste Ortungsverfahren etabliert und wird in Navigationssystemen weitverbreitet genutzt. [2]

GNSS[1]-Satelliten teilen über Funksignale ihre genaue Position und Uhrzeit mit. Zur genauen Positionsbestimmung werden die Signale von mindestens vier Satelliten benötigt. Im Empfangsgerät werden dann die Laufzeiten von den Signalen der vier Satelliten berechnet. Daraus werden dann die aktuelle Position (inklusive der Höhe) und die genaue Empfängeruhrzeit ermittelt. [2]

Bei einer Flughöhe von ca. 25.000 km müssen 24 bis 30 Satelliten um die Erde herum kreisen, damit jedes Empfangsgerät auf einem beliebigen Ort auf der Erde – auch bei nicht vollkommen freier Sicht zum Horizont – möglichst immer Signale von mindestens vier Satelliten gleichzeitig empfangen kann. [2]

Die Positionsgenauigkeit kann durch stationäre Empfangsstationen verbessert werden. Von den Landesvermessungsämtern wird das deutsche SAPOS-System betrieben. SAPOS stellt drei verschiedene Signaldienste zur Verfügung. Alle drei Dienste erreichen eine Genauigkeit von bis zu unter 1 cm. [2]

Die Laufzeiten müssen mit einer Genauigkeit von 10 Nanosekunden bestimmt werden, um eine Streckengenauigkeit von 3 Metern zu erreichen. Den Empfänger mit einer entsprechend hochgenauen Atomuhren auszustatten ist nicht nötig. Stattdessen wird der Fehler der Empfängeruhren ermittelt und bei der Positionsberechnung berücksichtigt. Zur Bestimmung der vier Unbekannten (drei Raumkoordinaten und Empfängeruhrenfehler)werden vier Satelliten benötigt. Dies führt zu einem Gleichungssystem aus vier Gleichungen mit vier Unbekannten. [2]

[1] Global Navigation Satellite System

Die ermittelten Koordinaten beziehen sich auf das Koordinatensystem des jeweiligen Navigationssystems; bei GPS beispielsweise auf WGS84. Auch die ermittelte Zeit ist durch das Navigationssystem definiert; so weicht z.B. die GPS-Zeit um einige Sekunden von der Universalzeit UTC ab, da Schaltsekunden bei der GPS-Systemzeit nicht berücksichtigt werden. Seit Anfang 2009 beträgt diese Abweichung 15 Sekunden. [2]

Aus den Raumkoordinaten können die geographische Länge, geographische Breite und die Höhe über dem definierten Referenzellipsoid berechnet werden. Zu beachten ist jedoch, dass die verwendeten Koordinatensysteme von anderen gängigen Koordinatensystemen abweichen können, so dass die ermittelte Position von der Position in vielen, insbesondere älteren Landkarten bis zu einigen hundert Metern abweichen kann. Auch die per GNSS ermittelte Höhe und die Höhe „über dem Meeresspiegel“ können vom tatsächlichen Wert (Geoid) um etliche Meter abweichen. [2]

Neben der amerikanischen Variante GPS existiert auch ein weiteres funktionsfähiges globales Satelliten-Positionierungssystem, das russische „GLONASS“. Andere Systeme, wie das chinesische „Compass“ und das europäische „Galileo“ befinden sich jedoch noch im Aufbaustadium.

2.2 Entfernungsberechnung

Die Berechnung zwischen zwei geografischen Koordinaten kann je nach Genauigkeitsansprüchen auf drei verschiedene Weisen berechnet werden.

2.2.1 Einfachste Entfernungsberechnung

Auf einem kleinen Kartenausschnitt stellen der Karte die Längen- und Breitenkreise gerade Linien dar und verlaufen parallel beziehungsweise rechtwinklig zueinander. Zur Entfernungsbestimmung für ein relativ kleines Gebiet kann man nach [4] die Erdoberfläche einfach als Ebene ansehen und zum Satz des Pythagoras greifen:

$$D = \sqrt{dx^2 + dy^2}$$

mit D: Entfernung in km

dx = 74.4 * (lon_1 – lon_2)

dy = 111.307 * (lat_1 – lat_2)

lat_1, lat_2, lon_1, lon_2: Breite, Länge in Grad [4]

Wenn die Länge und Breite in Grad angegeben werden, ergibt sich die Entfernung in Kilometern. Die Konstante 111,3 Km ist dabei der Abstand zwischen zwei Breitenkreisen und 74,4 Km der durchschnittliche Abstand zwischen zwei Längenkreisen in Stuttgarter Ortschaft.

2.2.2 Verbesserte Methode

Während der Abstand zwischen zwei Breitenkreisen immer konstant 111.3 km beträgt, ändert sich der Abstand zwischen zwei Längenkreisen in Abhängigkeit von der geografischen Breite: Am Äquator ist er 111.3 km, an den Polen hingegen 0. Genauer gesagt berechnet sich der Abstand nach der Formel:

$$111{,}307 \cdot \cos(lat)$$

Danach lässt sich wieder die Pythagoras-Formel anwenden. [4]

2.2.3 Exakte Entfernungsberechnung für die Kugeloberfläche

Für die Berechnung größerer Entfernungen sind die oben genannten Methoden nicht geeignet. Eine größere Fläche liegt nicht mehr in einer flachen Ebene, wie in zwei oben genannten Methoden. [4]

Wegen der kugelförmige Form der Erde, stellen die Längen- und Breitenkreise stellen kein rechtwinkliges Gitternetz mehr da. Während bei kleineren Flächen, wie einem Stadtplan, die Verzerrung kaum Auswirkung macht, lässt sich auf der Landkarte einer größeren Landschaft (z. B. Europa) die Verzerrung nicht mehr ignorieren. Hier muss zur Methoden der sphärischen Trigonometrie gegriffen werden. [4]

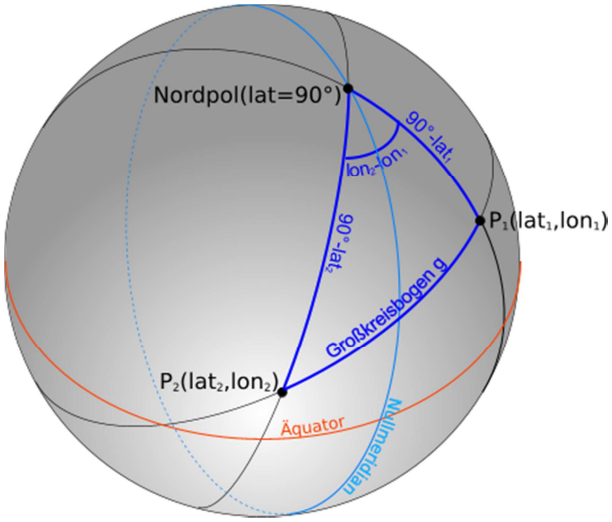

Abbildung 2.1: Die Abstandsberechnung auf der Kugeloberfläche [Quelle www.kompf.de]

Zur Anwendung kommt der Seitenkosinussatz:

$$\cos(g) = \cos(90° - lat_1) \cdot \cos(90 - lat_2) + \sin(90° - lat_1) \cdot \sin(90° - lat_2) \cdot \cos(lon_2 - lon_1)$$

mit g: Gesuchter Großkreisbogen und lat_1, lat_2, lon_1, lon_2: Breite, Länge in Grad.

Die Formel lässt sich vereinfachen:

$$\cos(g) = \sin(lat_1) \cdot \sin(lat_2) + \cos(lat_1) \cdot \cos(lat_2) \cdot \cos(lon_2 - lon_1)$$

Um die Entfernung in Kilometern zu bekommen, muss der Arkuskosinus gebildet und das Ergebnis mit dem Erdradius multipliziert werden:

$$D = 6378{,}388 \cdot \arccos(\sin(lat_1) \cdot \sin(lat_2) + \cos(lat_1) \cdot \cos(lat_2) \cdot \cos(lon_2 - lon_1)),$$

mit D: Entfernung in km. [4]

2.3 Serielle Schnittstelle

Im Jahr 1962 wurde von einem US-amerikanischen Standardisierungskomitee die RS232-Schnittstelle eingeführt. Die heutigen PCs werden noch mit dieser Schnittstelle ausgerüstet, während bei den Notebooks sie kaum noch eingebaut wird. Es besteht jedoch die Möglichkeit mithilfe eines Adapters die serielle COM-Schnittstelle über einen USB-Anschluss zu simulieren. [3]

Die Schnittstelle ist mittlerweile ca. 50 Jahre alt und hat bereits einige starke Konkurrenten wie USB oder FireWire. Dennoch ist die RS232-Schnittstelle unter Bastlern und beim industriellen Anwenden nach wie vor sehr beliebt. Die Beliebtheit ist den geringen Ansprüchen an Hardware und Software dankbar,

denn RS232 benutzt für die Datenübertragung ein einfaches asynchrones serielles Verfahren. [3]

Seriell bedeutet, dass die einzelnen Bits nacheinander über eine einzige Datenleitung transportiert werden. Es existiert auch kein gemeinsamer Takt und dem Datenempfänger ist die Ankunft des nächsten Bits auf die Leitung nicht bekannt. Daher wird die Schnittstelle als asynchron bezeichnet. Um bei diesem Verfahren eine sichere Datenübertragung zu gewährleisten, müssen der Sender und der Empfänger mit genau dem gleichen internen Takt betrieben werden und vor der Übertragung soll eine Synchronisation stattfinden. [3]

Die Datenleitungen werden mit negativer Logik betrieben. Die Spannungen zwischen -3V und -15V werden als logische "1" und die Spannungen zwischen 3V und 15V als logische "0" interpretiert. Spannungswerte zwischen -3V und 3V sind nicht zulässig. Heutzutage werden oft kleine Geräte verwendet, die mit TTL-Spanungspegel betrieben werden. Der zu hohe Standardspannungspegel von ±15V ist somit für solche Geräte ungeeignet. Bei den Geräten, die mit TTL-Spanungspegel betrieben werden entsprechen die Spannungswerte 2,4V-5V der logischen "1" und Spannungswerte, die kleiner als 2,4V sind, der logischen "0". [3]

Für eine Verbindungslose Übertragung werden in der Regel nur drei Leitungen benötigt. Eine Massenleitung und zwei Datenleitungen für eine bidirektionale Verbindung (eine Datenleitung für eine unidirektionale). Es ist ersichtlich, dass die serielle Übertragung recht geringe Ansprüche an die Hardware hat. [3]

Für eine Verbindungsorientierte Datenübertragung werden noch zusätzlich weitere 2 bis 4 Leitungen benötigt, um etwa die Sendebereitschaft oder den Sendewunsch mitzuteilen. [3]

Bei der Übertragung eines Bytes werden zusätzlich weitere Steuerbits mit übertragen. Jedes Byte auf der Leitung fängt mit einem Startbit an, danach folgen die Datenbits mit dem LSB voraus. Nach dem MSB kommt ein optionaler Paritätsbit (gerade oder ungerade). Die Übertragung wird mit einem oder zwei Stopbits abgeschlossen. [3]

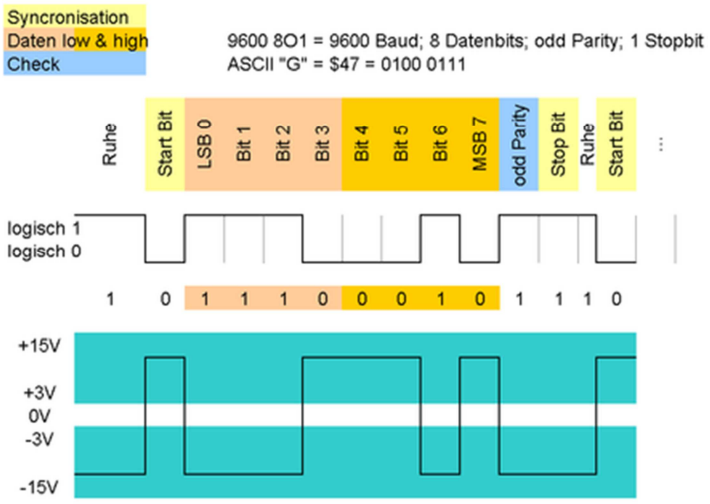

Abbildung 2.2: Bits auf der Leitung bei serieller Übertragung [Quelle:Wikipedia]

2.3.1 Übertragungsarten

Der µController des Quadrocopters unterstützt zwei Arten der Übertragungen:

Polling-Betrieb

Die einzelnen Bytes der zu übertragenden Nachricht werden in einer Schleife Zeichen für Zeichen in das Senderregister der seriellen Schnittstelle kopiert. Sobald das eine Zeichen versandt wird, kann das nächste in das Senderegister geschoben werden. Ein Nachteil dabei ist, dass bei etwas längeren Nachrichten der Programmablauf für kurze Zeit blockiert werden könnte. Es könnte dazu führen, dass der Quadrocopter etwas verzögert auf die Steuerung reagieren würde.

Interrupt-Betrieb

Das erste Byte der zu übertragenden Nachricht wird in das Senderregister der seriellen Schnittstelle kopiert und ein Sende-Interrupt-Flag wird aktiviert. Sobald der Byte versandt wird, löst ein Interrupt aus und eine Interrupt-Routine wird gestartet. In dieser Routine wird das nächste Byte ins Senderegister kopiert und der Interrupt-Flag erneut aktiviert.

Diese Methode hat einen Vorteil, dass die Übertragung quasi parallel zum Hauptprogramm verläuft. Soll vorwiegend bei Übertragung größerer Datenmengen eingesetzt werden, da zwischen den Übertragungen einzelner Bytes die Controllerressourcen nicht blockiert werden.

2.4 NMEA 0183 Protokoll

"NMEA 0183" ist ein Standard für die Kommunikation zwischen Navigationsgeräten der von der **N**ational **M**arine **E**lectronics **A**ssociation (NMEA) definiert wurde. Er besteht aus einer Definition von Datensätzen. [1]

Als Schnittstelle kann die RS-232-Schnittstelle verwendet werden. Für die Schnittstelle sind folgende Parameter definiert:

- Symbolrate: 9600 Baud(typischer Standardwert)
- Data Bits: 8
- keine Parität
- Stop Bits: 1 (oder mehr)
- kein Handshake

[1]

Die NMEA-Daten stellen Datensätze das, die aus ASCII-Zeichen bestehen. Jeder Datensatz kann maximal 80 ASCII-Zeichen umfassen. Der Satz fängt mit einem SOS-Zeichen (start of sequence) an, bei den meisten Datensätzen ist es ein „$“ ASCII-Zeichen, bei einigen Datensätzen kann auch ein „!“-Zeichen sein. Anschließend kommen die Geräte-ID (in der Regel zwei Zeichen) und die Datensatz-ID (meist drei Zeichen) als eine Zeichenkette. Darauf folgen die Datenfelder gemäß der Definition. Die Datenfelder werden durch ein Komma voneinander getrennt. Nach den Datenfeldern wird eine durch ein "*" abgetrennte hexadezimale CRC-Prüfsumme angehängt werden. Die Prüfsumme wird durch die XOR-Verknüpfung der ASCII-Werte aller Zeichen zwischen dem $ und dem * berechnet. Der Datensatz wird durch eine Kombination aus zwei ASCII-Zeichen 0x13 "<CR>" (Wagenrücklauf) und 0x10 "<LF>" (Zeilenvorschub) abgeschlossen. [1]

Beispiel eines Datensatzes:

```
$GPRMC,101212,A,4930.5900,N,07322.3900,E,10.0,10.0,130412,1.2,E,A*10
```

```
$GPRMC,HHMMSS,A,BBBB.BBBB,b,LLLLL.LLLL,l,GG.G,KK.K,DDMMYY,M.M,m,F*PP
```

RMC steht für **R**ecomended **M**inimum **S**entence **C**. Jedes GNSS[2]-Empfänger muss diesen Datensatz ausgeben können. Die nachfolgende Tabelle beschreibt die Felder des Datensatzes[1]

[2] **G**lobal **N**avigation **S**atellite **S**ystem

Symbol	Bedeutung
GP	Geräte-ID. **G**lobal **P**ositioning **S**ystem (GPS)
RMC	Datensatz-ID. **R**ecomended **M**inimum Sentence **C**
HHMMSS	Zeit (UTC)
A	Status (A für OK, V bei Warnungen)
BBBB.BBBB	Breitengrad
b	Ausrichtung (N für North, nördlich; S für South, südlich)
LLLLL.LLLL	Längengrad
l	Ausrichtung (E für East, östlich; W für West, westlich)
GG.G	Geschwindigkeit über Grund in Knoten
KK.K	Kurs über Grund in Grad bezogen auf geogr. Nord
DDMMYY	Datum (Tag Monat Jahr)
M.M	magnetische Abweichung (Ortsmissweisung)
m	Vorzeichen der Abweichung (E oder W)
F	Signalintegrität : A = Autonomous mode, D = Differential Mode, E = Estimated (dead-reckoning) mode M = Manual Input Mode S = Simulated Mode N = Data Not Valid
PP	hexadezimale Darstellung der Prüfsumme

Tabelle 1: Datenfelder eines RMC Datensatzes [Quelle: Wikipedia]

Im Folgenden werden einige weitere Datensätze kurz aufgelistet.

GPGLL (engl. Geographic Position-Latitude/Longitude) Ein kurzes Datenpaket, welches nur die geografische Position enthält.

GPVTG (engl. course over ground and ground speed). Das Datenpaket enthält nur aktuelle Kurs und Geschwindigkeit

GPGSA (engl. DOP and active satellites) Der Datensatz enthält Infos über alle Satelliten

GPGSV (engl. satellites in view) Der Datensatz enthält Infos über die Satelliten, die sich in der Sichtweite befinden

2.5 MTK NMEA packet Format

Um den GPS-Empfänger zu konfigurieren wird der von der Firma „MediaTec Inc.“ entwickelte MTK Paketformat benutzt. Der Datensatz wird über die serielle Schnittstelle an das GPS-Gerät übertragen. Der Aufbau des Datensatzes ist dem NMEA 0183 Format ähnlich. Einige Beispiele, die auch in der Arbeit verwendet werden sind im Folgenden ausführlich aufgeführt.

2.5.1 Einstellung der Baudrate

`$PMTK251,9600*17`

Packet-Typ = `PMTK`

Message-ID = `251`

Baudrate = `9600`

Prüfsumme = `17`

2.5.2 Einstellung des Ausgabeformats und -frequenz

`$PMTK314,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0*29`

Packet-Typ = `PMTK`

Message-ID = `314`

Insgesamt kommen anschließend 17 Felder im Datensatz, jedes Feld steht für ein bestimmtes NMEA-Ausgabeformat (werden später erläutert). Ein GPS-Empfänger kann seine Position mehrmals pro Sekunde bestimmen. Eine solche Positionsbestimmung wird üblicherweise als sog. Fix bezeichnet. Die Zahlen in diesen 17 Datenfeldern repräsentieren zu welchem Fix soll der Datensatz ausgegeben werden. Im Beispiel oben steht im zweiten Datenfeld nach dem Paket-Typ steht der Wert "1", das bedeutet der Datensatz `RMC` wird zu jedem Fix ausgegeben. Im nächsten Feld steht eine „5“, d.h. der Datensatz `VTG` wird alle 5 Fixes ausgegeben. Bei einem GPS-Empfänger mit 10Hz Updaterate würden die Datensätze mit den Intervallen 0,1s (`RMC`) und 0,5s (`VTG`) ausgegeben werden[3].

Prüfsumme = 29

[3] Nähere Informationen über den Aufbau weiterer Datenpakete können der MTK-Spezifikation entnommen werden

2.6 Threadsynchronisation

Bei dem Programmablauf im µController gibt es neben dem regulären Thread (das eigentliche µController-Programm) weitere Threads, die sogenannten Interrupt-Routinen. Diese Routinen werden durch das Auslösen eines bestimmten Ereignisses abgearbeitet. Sie haben höhere Priorität, d.h. tritt ein Ereignis ein, wird der reguläre Thread gestoppt, die Interrupt-Routine abgearbeitet und dann wird der Ablauf des regulären Threads fortgesetzt. Die Interrupt-Routine kann durch die anderen Interrupts nicht unterbrochen werden. Sie wird bis zum Ende ausgeführt, daher ist es besonders wichtig, dass die Interrupt-Routine nirgendwo stehen bleibt, sonst tritt sog. Deadlock ein.

Sollten der reguläre Thread und die Interrupt-Routine auf gemeinsame Daten zugreifen, kann es schnell zu inkonsistentem Datenzustand führen. Aus diesem Grund muss der Zugriff auf gemeinsame Daten synchronisiert und vor einem gleichzeitigen Gebrauch geschützt werden. Eine einfache Art einer Threadsynchronisation ist ein Mutex (Abk. für engl. ***mut**ual **ex**clusion*). Mutex bedeutet wechselseitiger Ausschluss. Die Funktionsweise besteht darin, dass der Thread, der den kritischen Bereich als erster betritt, seinen Aufenthalt in diesem Bereich markiert. Am einfachsten erfolgt dies mithilfe einer booleschen Variable. Vor dem Betreten des kritischen Bereichs wird der Wert dieser booleschen Variable geprüft. Ist er auf `true` gesetzt, heißt es im Bereich befinden sich keine andere Threads und der aktuelle Thread darf ihn betreten. Ist er auf `false` gesetzt, heißt es der Bereich wird gerade benutzt. Ob hier eine alternative Routine geben soll oder ein `return` aus der Funktion mit einem entsprechenden return-Wert, das ist dem Programmierer überlassen.

2.7 Präprozessor-Operator

Der ##-Operator heißt auch Verkettungsoperator. Damit können dynamisch Bezeichner verkettet werden. Das Ergebnis der Verkettung wird erneut geprüft ob es einer weiteren Ersetzung unterzogen werden kann. Das nachfolgende Programm zeigt die Funktionsweise eines Verkettungsoperators. [5]

```
#include <stdio.h>
#define VERBINDE(X,Y) X ## Y
int main( void )
{
   printf( "Verbindung zweier int-Zahlen: %d\n", VERBINDE (11, 22) );
   return 0;
}
```

Ausgabe:

```
Verbindung zweier int-Zahlen: 1122
```

Beim nachfolgenden Beispiel wird gewünscht, dass der Parameter WERT durch seinen Wert, also "1", ersetzt und mit dem Bezeichner ZAHL verbunden wird. Es wird ein Fehler gemeldet.

```
#include <stdio.h>
#define VERBINDE(X,Y) X ## Y
#define ZAHL1  5
#define WERT   1
#define VAR1   VERBINDE(ZAHL,WERT)
int main( void )
{
   printf( "Wert des dynmischen defines: %d\n", VAR1 );
   return 0;
}
```

Ausgabe:

```
error C2065: 'ZAHLWERT' : undeclared identifier
```

Das ist nur mit einer zweifach vertieften Struktur des ##-Operators möglich. Nun die richtige Variante:

```
#include <stdio.h>
#define VERBINDE2(X,Y) X ## Y
#define VERBINDE(X,Y) VERBINDE2(X,Y)
#define ZAHL1  5
#define WERT   1

#define VAR1   VERBINDE(ZAHL,WERT)
int main( void )
{
   printf( "Wert des dynamischen defines: %d\n", VAR1 );
   return 0;
}
```

Ausgabe:

```
Wert des dynamischen defines: 5
```

3 Anforderungsanalyse

3.1 Zweck des GPS-Moduls

Der GPS-Treiber soll eine Kommunikation zu dem GPS-Device sicherstellen, die NMEA-0183 Datensätze vom GPS-Device empfangen, verarbeiten und zum Zweck weiterer Verwendung der Hauptapplikation zur Verfügung stellen. Weiterhin soll eine Zusatzfunktion implementiert werden, die den Abstand zwischen zwei GPS-Positionen berechnet.

3.2 Requirements

3.2.1 Funktionale Requirements

Requirement 100:

Der GPS-Empfänger soll so konfiguriert werden, dass nur der RMC-Datensatz ausgegeben wird.

Requirement 110:

Der GPS-Empfänger soll auf eine Baudrate von 9600 Baud eingestellt werden.

Requirement 120:

Es soll möglich sein von der Hauptapplikation auf die empfangenen Daten, wie geografische Position, Uhrzeit und Datum zuzugreifen.

3.2.2 Nicht funktionale Requirements

Requirement 500:

Es sollen keine rechenintensive Operationen und Routinen implementiert werden, sonst besteht die Gefahr einer Blockierung.

Requirement 510:

Der GPS-Empfänger soll vor dem Flugstart des Quadrokopters initialisiert werden.

Requirement 520:

Die Baudrate für die Kommunikation mit dem GPS-Empfänger soll konfigurierbar sein.

Requirement 530:

Es soll ausschließlich die Festkommaarithmetik verwendet werden.

Requirement 540:

Werte, die üblicherweise in als Fließkommazahl gespeichert werden, sollen als eine Festkommazahl mit einem Exponenten abgespeichert werden, Z.B. die Zahl 10,46 soll als $1046 * 10^{-2}$ abgespeichert werden.

Requirement 560:

Für die Kommunikation mit dem GPS-Empfänger soll die serielle Schnittstelle SCI2 benutzt werden.

Requirement 570:

Während der Entwicklungs- und der Testphase auf dem Evaluierungsboard als Port für die Kommunikation mit dem GPS-Empfänger soll die serielle Schnittstelle SCI1 benutzt werden.

Requirement 590:

Empfang der Daten aus dem GPS-Empfänger soll mithilfe einer Interrupt-Routine umgesetzt werden.

Requirement 600:

Fehlerhafte Pakete sollen ignoriert werden.

Requirement 610:

Der Zugriff auf die zwischengespeicherten Daten soll synchronisiert werden, damit kein inkonsistenter Datenzustand entsteht.

4 Entwurf

In diesem Kapitel werden drei große Problembereiche samt der Lösungsansätze dargestellt. Die drei Bereich sind „Synchronisation der Baudraten“ „Daten empfangen“ und „Verarbeitung der Daten“.

4.1 Synchronisation der Baudraten

Für eine erfolgreiche Kommunikation mit dem GPS-Empfänger muss die Baudrate des Empfängers bekannt sein. Ein besteht dabei jedoch ein Problem. Laut dem Datenblatt beträgt die Default-Baudrate des GPS-Empfängers beträgt 9600 Baud. Es könnte jedoch passieren dass vom Werk ab der-Empfängers auf eine andere Baudrate eingestellt ist. In diesem Fall muss als erstes die eingestellte Baudrate bestimmt werden. Eine Möglichkeit ist die Bestimmung der Baudrate mittels eines Oszilloskops. Im Folgenden die Bestimmung der Baudrate des in der Entwicklung verwendeten GPS-Empfängers mithilfe eines Oszilloskops.

- Dauer eines Bits 104 µs.
- In einer Sekunde 9615,38Baud ≈ 9600 Baud

Sollte der GPS- Empfänger im Teilsystem ausgetauscht werden und nach dem Start-Up keine korrekten Daten liefern, so könnte als mögliche Ursache die falsch eingestellte Baudrate des Empfängers sein.

Da die Bestimmung der Baudrate jedes Mal mittels eines Oszilloskops recht aufwendig ist, sollte die Bestimmung automatisch durchgeführt werden. Der GPS-Empfänger unterstützt insgesamt fünf Baudraten: 4800, 9600, 14400, 34400 und 56700 Baud. Die gewünschte Baudrate wird in einer Nachricht als String angegeben (siehe Kapitel 2.5). Diese Nachricht soll auf allen unterstützten Baudraten an den GPS-Empfänger übertragen werden.

Das nachfolgende Aktivitätsdiagramm zeigt den Ablauf der Initialisierungs-Routine.

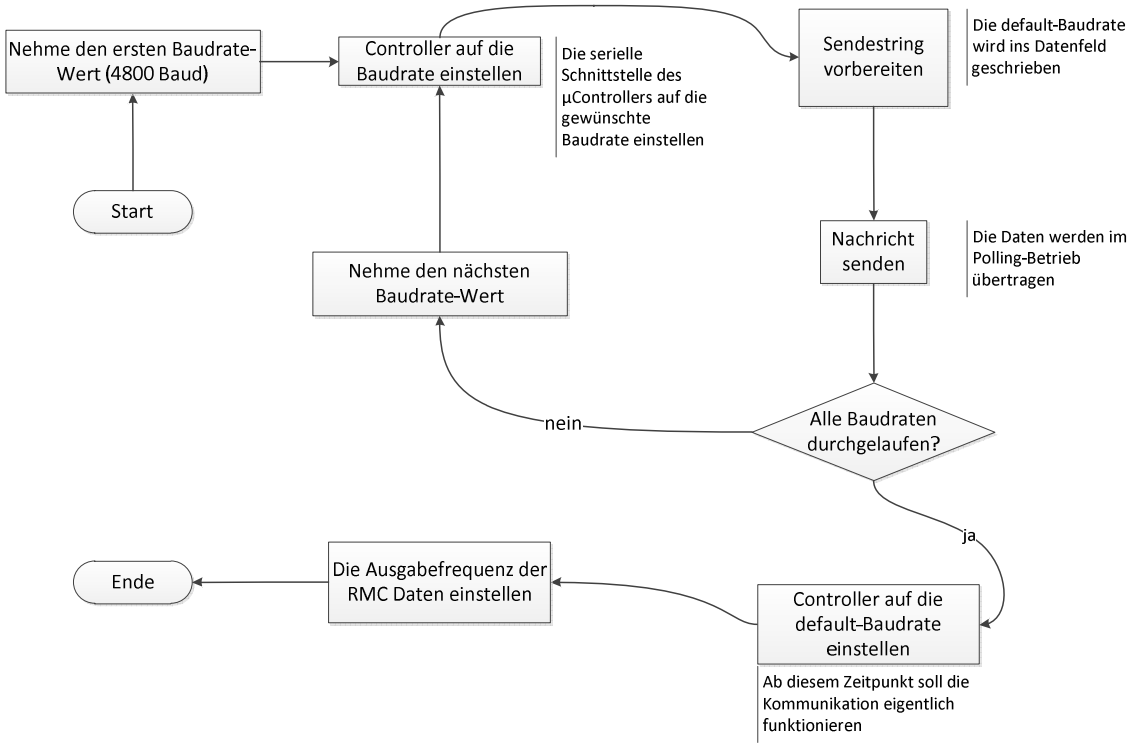

Abbildung 4.1: Aktivitätsdiagramm des Init-Vorgangs

Wie bereits im Requirement 580 gefordert, soll die Initialisierung des Treibers vor dem Flugstart ausgeführt werden, da diese blockierende Aufrufe enthält. In diesem Fall können die Echtzeitanforderungen der Quadrocopter-Steuerung unter Umständen nicht mehr garantiert werden.

4.2 Datenübertragung

4.2.1 GPS-Empfänger zu µController

Das Empfangen der NMEA-Nachricht erfolgt im nachfolgenden Bild dargestellten Zustandsautomaten.

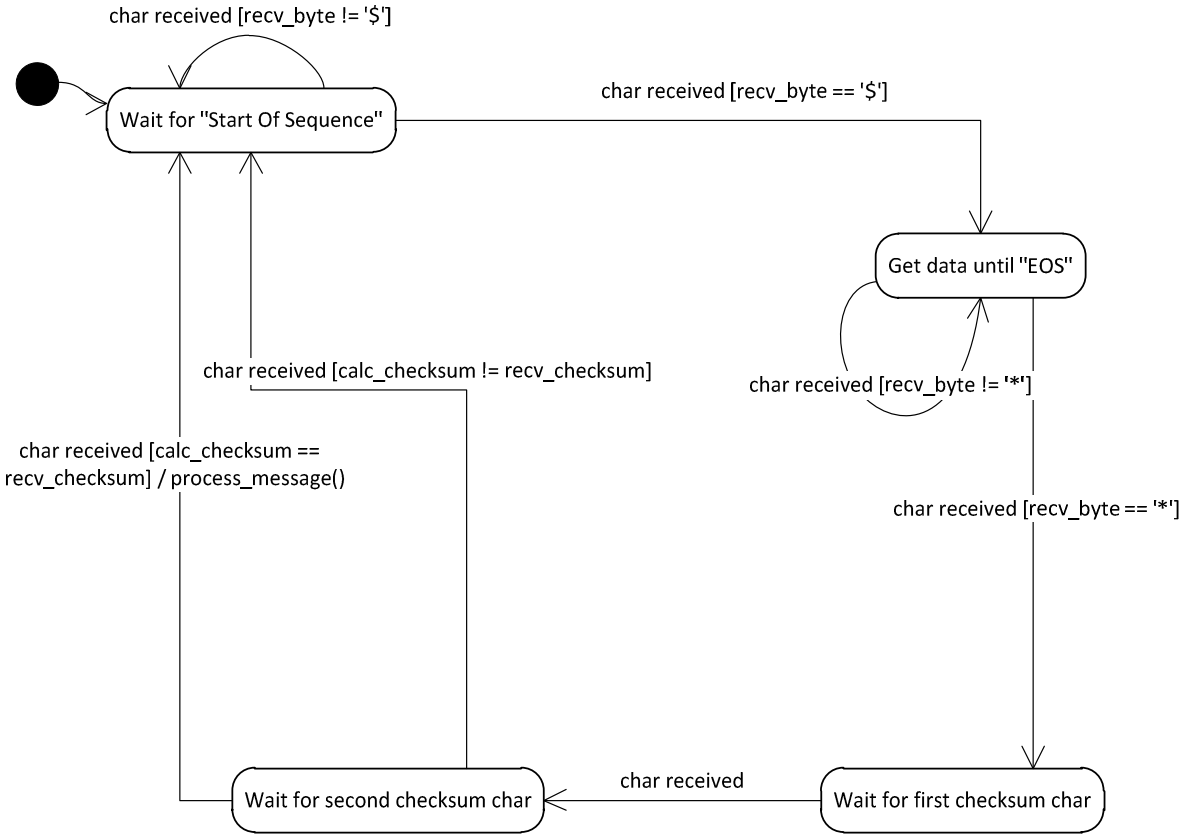

Abbildung 4.2: Zustandsautomat für den Datenempfang

Beschreibung des Zustandsautomaten

Init-Zustand `Wait for "Start Of Sequence"`

In diesem Zustand wird auf das SOS-Zeichen ‚$' gewartet. Entspricht der empfangene Byte dem SOS-Zeichen, ist der Anfang der NMEA-Nachricht gefunden. Der Zustand wird verlassen.

Zustand: `Get data until "EOS"`

In diesem Zustand werden alle nachkommenden Zeichen in einem Array abgespeichert solange kein Endzeichen "*" (**E**nd **O**f **S**equence) kommt. Es deutet auf ein Ende des Datenabschnitts und die nachkommende Prüfzeichen hin. Der Zustand wird verlassen. Sollte es in diesem Zustand zu einem Array-Überlauf kommen, wenn seit Beginn der Nachricht mehr als eine zulässige Anzahl der

Zeichen empfangen und gespeichert wurden, wird ein RESET ausgelöst, welcher den Zustandsautomaten in den Init-Zustand versetzt und den Zähler auf „0“ setzt.

Zustand: `Wait for first checksum char`

Dieser Zustand wird benötigt um das erste Prüfzeichen zu empfangen. Es repräsentiert den höheren Nibble einer 8-Bit Zahl. Das ASCII-Zeichen wird konvertiert und danach wird der Zustand verlassen.

Zustand: `Wait for second checksum char`

In diesem Zustand wird das zweite Prüfzechen empfangen. Es repräsentiert den unteren Nibble einer 8-Bit Zahl. Das Zeichen wird konvertiert daraus wird zusammen mit dem erster Prüfzeichen der CRC-Wert gebildet. Aus der empfangenen NMEA-Nachricht wird ebenfalls ein CRC-Wert gebildet und wird mit dem empfangenem verglichen. Stimmen die Werte überein, ist das Paket korrekt übertragen worden und kann weiter bearbeitet werden und im Anschluss wird der Zustand nach Init-Zustand gewechselt. Stimmen die CRC-Werte nicht überein wird das Paket ignoriert und es wird nach Init-Zustand gewechselt ohne das Paket zu bearbeiten.
Es wird kein „timeout“ im Zustandsautomaten verwendet, da der Empfänger, unabhängig davon ob ein Signal vorliegt oder nicht, ständig die Daten übermittelt.

4.2.2 µController zu GPS-Empfänger

Da die Datenübertragung an den GPS-Empfänger vor dem Flugstart stattfindet, wird dabei der Polling-Betrieb verwendet. Während des Flugs werden zurzeit keine Daten an den GPS-Empfänger übertragen.

4.3 Threadsynchronisation

ALs Synchronisationsmechanismus wird das mutex-Prinzip angewendet. Über eine bool‘sche Semaphore wird der Zugriff auf den kritischen Datenspeicherbereich synchronisiert.

4.4 Verarbeitung der Daten

Die NMEA Daten werden in einem Array abgespeichert. Als nächstes wird aus dem Array jedes Datenfeld einzeln gelesen und anschließend verarbeitet.

Dazu kommt ein Parser zum Einsatz, welcher ein bestimmtes Feld identifiziert und in ein temporäres Array kopiert.

Um die Effizienz zu steigern wurde bei jedem Suchvorgang die letzte Suchposition gespeichert. Bei der Anfrage des nächsten Feldes wurde der Suchvorgang ab dieser Position bis zum nächsten Separator fortgesetzt.

Eint Return-Wert „`true`“ kommt falls die suche nach dem nächsten Feld ab der Position erfolgreich war. Ist das Feld leer oder das Array mit den Daten ist zu Ende wird ein „`false`“ zurückgegeben. Ein leeres Feld (zwei nacheinander folgende Kommas) im Datenarray könnte so aussehen „456,,0,10.0,15.0,“

5 Implementierung

Die Implementierung der Funktionen erfolgte mithilfe der Entwicklungsumgebungen Visual Studio und CodeWarrior. Die Umgebung CodeWarrior diente ausschließlich zur Kompilierung und zum Hochladen der kompilierten Software auf den µController.

5.1 Modularer Aufbau

Die Abbildung 5.1 zeig den modularen Aufbau des GPS-Treibers. Es besteht hauptsächlich aus drei Teilen, aus der Applikation und aus zwei betriebssystemabhängigen Implementierungen, aus der Simulation unter Windows XP und dem Einsatz auf dem Target-Plattform.

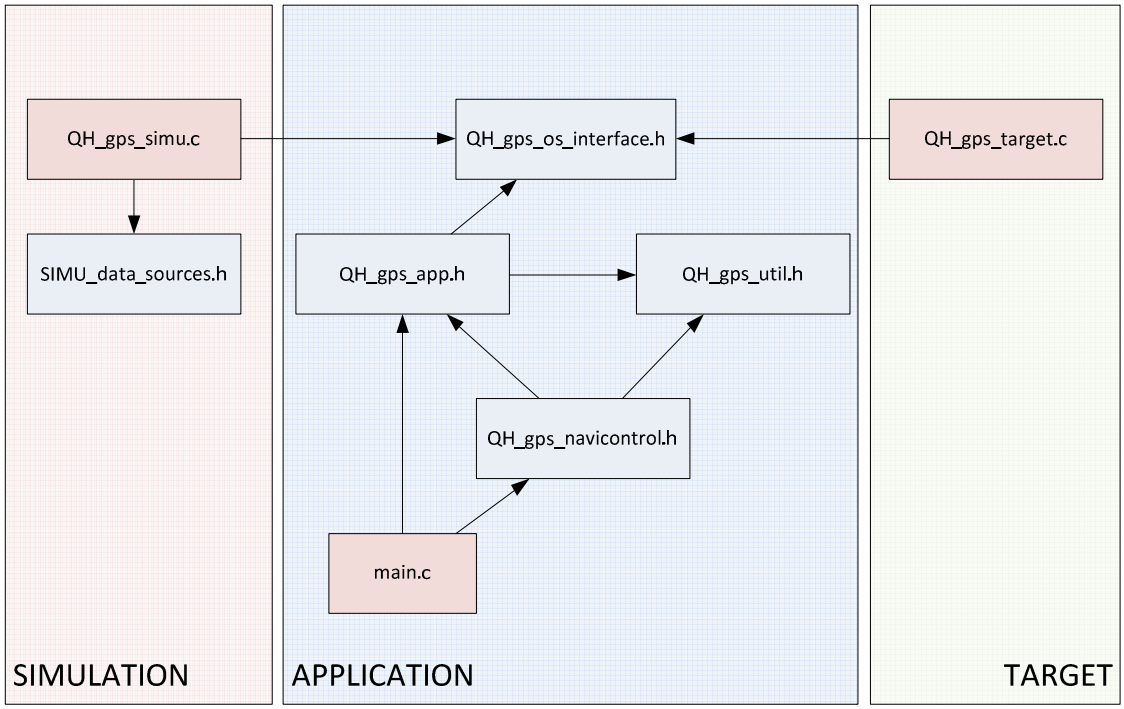

Abbildung 5.1: Modularer Aufbau der GPS-Software[4]

Beschreibung der Dateien

Die Datei `QH_gps_app.h` enthält die Schnittstelle zu der Applikation des GPS-Treibers, die die komplette Treiber-Logik enthält.

[4] Die Pfeile repräsentieren eine <<include>> Beziehung

In der Datei `QH_gps_navicontrol.h` sind Schnittstellen zu den Funktionen definiert, die für die Verwaltung der autonomen Navigation des Quadrocopters zuständig sind. Enthält momentan nur die Funktionen zum Berechnen des Abstands zwischen zwei GPS-Koordinaten.

`QH_gps_util.h` Schnittstelle zu den Funktionen die einfache Berechnungs- und Konvertierungsaufgaben ausführen.

`QH_gps_interface.h` Enthält Schnittstellen zu den betriebssystemabhängigen Funktionen.

`QH_gps_simu.c` Hier befinden sich dummy Funktionen, die die Funktionalitäten des µController simulieren. Wird eine dieser Funktionen ausgeführt, wird ein entsprechender Text auf der Konsole ausgegeben.

`QH_gps_target.c` Hier sind die betriebssystemabhängigen Funktionen des µController Implementiert

`SIMU_data_sources.h` Enthält die dummy-Daten für die Tests, wie NMEA-RMC-Datensätze

5.2 Dynamische Bezeichner

Das Evaluierungsboard, auf dem der Treiber entwickelt wurde, enthält nur zwei serielle Schnittstellen, SCI0 und SCI1. Auf dem Quadrocopter-µController wird der GPS-Empfänger über die serielle Schnittstelle SCI2 angeschlossen, welche auf Entwicklungs-µController nicht vorhanden ist. Jede serielle Schnittstelle hat ca. acht Bezeichner, wie z.B. `SCI1CR1` (Einstellungsregister der seriellen Schnittstelle SCI1) oder `SCI1DRL` (Senderegister der SCI1). Um die spätere Software-Integration zu vereinfachen werden die betriebssystemabhängigen Bezeichner anhand der Portnummer dynamisch angepasst. Unten ist ein Ausschnitt aus der Datei `QH_gps_app.h` aufgeführt. Hier wird der ##-Operator angewendet (siehe Kapitel 2.7). Über den Bezeichner `SCI_NUM` wird die gewünschte Portnummer festgelegt. Im Ausschnitt wird z.B. der Bezeichner `SCIxCR1` definiert, hinter dem eigentlich der Bezeichner `SCI0CR1` steht. Sollte ein anderer Port verwendet werden, muss schließlich nur der Wert des Bezeichners `SCI_NUM` geändert werden.

```
#define SCI_NUM 0    // serial port number
#define SETPORT2(a,b,c) a##b##c
#define SETPORT(a,b,c) SETPORT2(a,b,c)
#define SCIxCR1 SETPORT(SCI,SCI_NUM,CR1)
```

5.3 Abstandberechnung

Für die Abstandberechnung wurde die „verbesserte Methode“ (siehe Kapitel 2.2.2) verwendet, jedoch aufgrund der Festkommaarithmetik (siehe Requirement 530) wurde die Berechnung `cos(Breitengrad)` durch eine Lookup Tabelle ersetzt, die bereits berechnete Werte enthält. Die Berechnung der Wurzel einer Zahl wurde numerisch mithilfe des Heron-Verfahrens[5] ersetzt.

Die Funktion `calcDistance(…)` bekommt zwei GPS-Punkte `GeograficPosition` als Parameter, berechnet den Abstand und gibt ihn als Rückgabeparameter zurück.

5.4 Konfigurationsparameter

`RX_MESSAGE_BUFFER_SIZE` Größe des Empfangsbuffers. Ein NMEA-Datensatz enthält maximal 80 ASCII-Zeichen. Die Größe wurde auf 81 festgelegt und sollte somit für alle Arten der NMEA-Nachrichten ausreichen.

`TX_MESSAGE_BUFFER_SIZE` Größe des Sendebuffers. Die Größe wurde ebenfalls auf 81 festgelegt, was momentan ausreichend ist, sollte jedoch bei Bedarf erhöht werden.

`RMC_DATA_UPDATE_FREQ` Diese Zahl gibt an, zu welchem sog. „position fix“ soll der RMC Datensatz an den µController ausgegeben werden. Der GPS-Empfänger ls20031 bestimmt seine Position 10 Mal pro Sekunde, hat also 10 „fixes“. Der Wert 1 bedeutet, dass der Datensatz nach jedem „fix“ ausgegeben wird. Eine Zahl 10 bedeuten zu jedem zehnten „fix“, also einmal pro Sekunde.

`DEFAULT_BAUD_RATE` Eingestellte Baudrate für die Kommunikation zwischen dem µController und dem GPS-Empfänger. Standardwert 9600 Baud.

5.5 Schnittstellen zum Treiber

Die GPS-Treiber-Software besitzt eine Schnittstelle zum Hauptprogramm, die in der Datei `QH_gps_app.h` definiert sind. Sie besteht aus der Funktion `gpsInit()` zur Initialisierung und einem Satz der get-Funktionen um auf die empfangenen Daten wie GPS-Koordinaten, Datum, Uhrzeit oder auf den ganzen RMC-Datensatz zuzugreifen. Zwei weiteren Funktionen `processNMEAbyte(…)` und `syncBaudRate()`sollten auf **keinen Fall** explicit aus dem Hauptprogramm aufgerufen werden, da es sonst zur fehlerhaften Arbeit führen wird.

---

[5] Heron-Verfahren [7] für numerische Berechnung der Wurzel einer Zahl: $x_{n+1} = \frac{x_n + \frac{a}{x_n}}{2}$

6 Test

6.1 Aufbau

Die nachfolgende Abbildung zeigt den Gesamtaufbau des Prüfstandes.

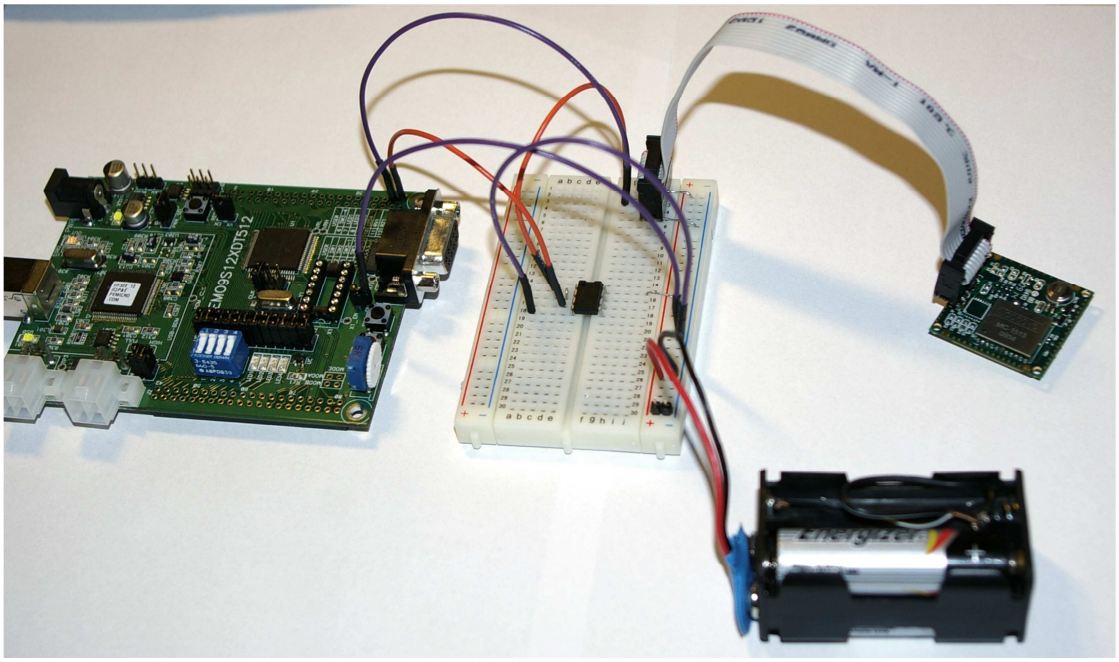

Abbildung 6.1: Aufbau des Prüfstandes

Der GPS-Empfänger wird von einem Batterieblock, bestehend aus drei AA-Mignon Batterien, mit einer Spannung ca. 3,6V versorgt und der µController wird über ein USB-Kabel mit Spannung versorgt.

Die Verkabelung der seriellen Verbindung erfolgte mithilfe einer Steckplatine. Der GPS-Empfänger wurde über einen 10-polligen Flachbandkabel mit Pfostenstecker (wird meist für USB-Verbindungen in PC-Gehäusen verwendet) an die Steckplatine angeschlossen, dazu wurde zuerst an dem GPS-Empfänger eine Stiftleiste mit fünf Pins angelötet.

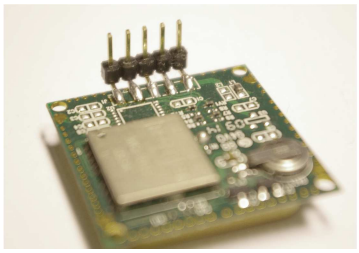

Abbildung 6.2: Anschlusspins auf dem GPS-Empfänger

Der Sende- und Empfangsanschluss der seriellen Schnittstelle des µControllers (im roten Rechteck im Bild unten) sowie der Masseanschluss des µControllers (im roten Kreis) wurden mithilfe drei Kabel an die Steckplatine angeschlossen.

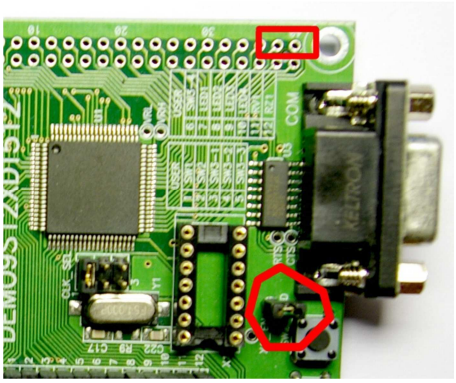

Abbildung 6.3: Anschlüsse auf dem µController

Laut dem Datenblatt des GPS-Empfängers 20031 darf der Spannungspegel auf der Empfangsleitung den Wert 3,6V nicht überschreiten. Der µController, der im Quadrocopter eingesetzt wird, wird mit einer Spannung von 3,3V betrieben. Das Evaluierungsboard wird aber von dem USB-Anschluss mit einer Spannung von 5V betrieben. Es soll daher eine Pegelanpassung auf der Empfangsleitung stattfinden. Die Umwandlung der Spannung erfolgte mithilfe einer OP-Schaltung auf der Steckplatine. Es wurde ein Impedanzwandler auf der Basis eines OP-Verstärkers LM358N verwendet. Die Versorgungsspannung des OPs betrug 3,6V und wurde von derselben Quelle wie des GPS-Empfängers angeschlossen.

6.2 Durchführung

Für jede Funktion wurden drei bis vier Tests implementiert, die sowohl die Funktionalität mit gültigen als auch mit ungültigen Parametern überprüfen. Um die Treiber-Funktionalität in einer mehr realen Umgebung zu Testen wurde die Quadrocopter-Software auf das Evaluierungsboard geladen und schließlich mit

dem angeschlossenen GPS-Empfänger mehreren dauerhaften Tests unterzogen.

7 Konfiguration und Integration

Dieser Kapitel ist für Anwender des GPS-Treibers und für Entwickler, die evtl. den Treiber erweitern bzw. verbessern möchten, gedacht. Dieser Studienarbeit liegt eine CD bei. Auf dieser CD befinden sich alle Quellcode-Dateien mit den Tests.

Im Ordner `Studienarbeit_GPS-Sensor\impl\SimulationProject` befindet sich der VisualStudio Projekt mit der Simulation des Treibers.

Im Ordner `Studienarbeit_GPS-Sensor\impl` befinden sich die Quellcode-Dateien des Treibers

Im Ordner `Studienarbeit_GPS-Sensor\impl\simulation` befinden sich simulationsspezifische Implementierungen sowie dummy-Daten.

Für eine erfolgreiche Integration des Treibers müssen folgende Schritte durchgeführt werden.

1. Einbindung der Dateien in das Quadrocopter-Projekt in der CodeWarrior-Entwicklungsumgebung. Folgende Dateien müssen eingebunden werden:
   `QH_gps_app.h` und `QH_gps_app.c`
   `QH_gps_navicontrol.h` und `QH_gps_navicontrol.c`
2. `QH_gps_target.c` und `QH_gps_os_interface.h`
   `QH_gps_util.h` und `QH_gps_util.c`
3. In der Datei `QH_gps_target.h` muss der Wert der variable `SCI_NUM` auf 2 geändert werden und alle vom SCI-Port abhängigen Bezeichner werden automatisch beim Kompilieren erzeugt (siehe Kapitel 5.2).

8 Ausblick

Nach dieser Studienarbeit ist der Quadrocopter nun in der Lage seine eigene Position zu bestimmen. Als nächster Schritt bzw. Erweiterung könnte die Programmierung einer Navigationslogik sein. Eine Route, bestehend aus einem Dutzend GPS-Koordinaten sog. Checkpoints, könnte z.B. drahtlos über das ZigBee-Modul übermittelt werden. So könnte der Quadrocopter durch diese Checkpoints mithilfe der Bestimmung eigener Position eigenständig abfliegen.

Der aktuelle Stand der Technik, die im Quadrocopter momentan eingesetzt wird, erlaubt einen Flugbetrieb von nur maximal ca. 20 Minuten. Sollte in Zukunft es möglich sein die Flugdauer zu erhöhen, so wäre der Einsatz des Quadrocopters als "kleiner" Botschafter oder z.B. für die Aufnahmen der Landschaften in der Kartografie denkbar.

Die nachfolgende Abbildung zeigt eine Skizze des Zustandsautomaten, der die Idee repräsentiert.

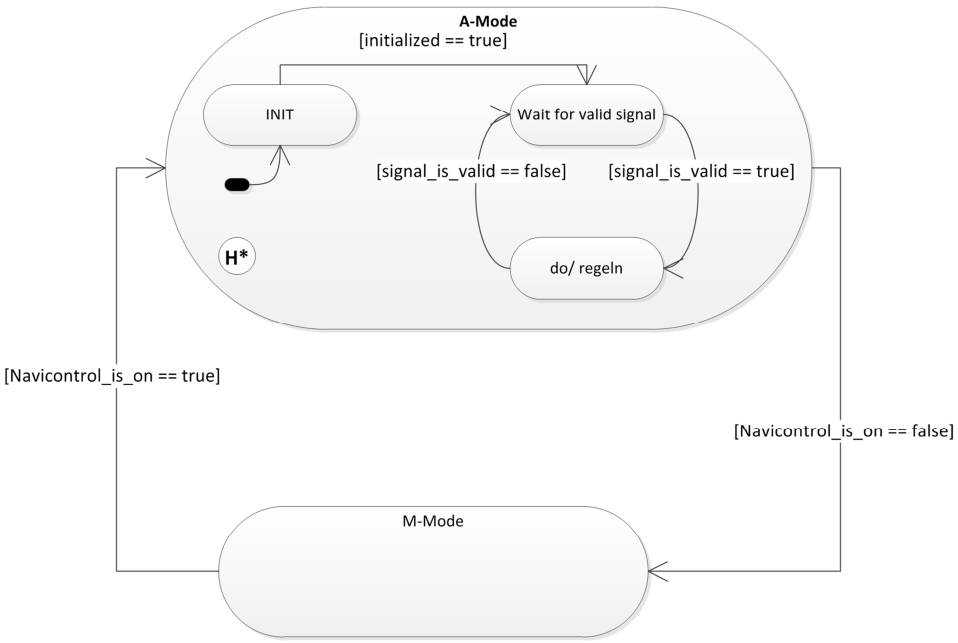

Abbildung 8.1: Zustandsautomat des Quadrocopters

Der Zustand `A-Mode` bedeutet autonomer Betrieb. In diesem Zustand fliegt der Quadrocopter nach vorgegebener Routine sobald das ein gültiger GPS-Signal vorliegt. Ist das Signal verschwunden, soll der Quadrocopter nichts unternehmen und wieder auf einen gültigen GPS-Signal warten. Jederzeit soll eine Möglichkeit bestehen den Quadrocopter auf den manuellen Betrieb umzustellen (im

Bild als Zustand M-Mode). Dann kann er mithilfe der Fernbedienung gesteuert werden. Der sog. Deep history connector (im Bild als H* gekennzeichnet) sorgt dafür, dass nach dem erneuten Umschalten auf dem autonomen Betrieb der innere Zustandsautomat des Zustands A-Mode sich seinen letzten Zustand nicht merkt.

Quellen

[Projektbeschreibung] Allgemeine Beschreibung des Quadrocopter-Projekts

[1] Wikipedia: "NMEA 0183".
Link: http://de.wikipedia.org/wiki/NMEA_0183

[2] Wikipedia: „Global Positioning System“
Link: http://de.wikipedia.org/wiki/Global_Positioning_System

[3] "RS232-Interface"
Link: http://www.sprut.de/electronic/interfaces/rs232/rs232.htm

[4] Entfernungsberechnung http://www.kompf.de/gps/distcalc.html

[5] “C als erste Programmiersprache” von Manfred Dausmann, Ulrich Bröckl, Joachim Goll, August 2005

[6] Wikipedia „Heron-Verfahren“
http://de.wikipedia.org/wiki/Heron-Methode

[7] Wikipedia „V-Modell“ http://de.wikipedia.org/wiki/V-Modell