Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Abschlussarbeit

Roman Kray

Entwicklung einer selbstüberwachten und robusten Systemarchitektur für einen Positionsdatenlogger

Fakultät Technik und Informatik

Roman Kray

Entwicklung einer selbstüberwachten und robusten Systemarchitektur für einen Positionsdatenlogger

Bachelorthesis eingereicht im Rahmen der Bachelorprüfung
im Studiengang Mechatronik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg
Betreuender Prüfer: Prof. Dr. rer. nat. Rasmus Rettig
Zweitgutachter: Prof. Dr. rer. nat. Thomas Lehmann
Eingereicht am: 12.07.2016

Thema der Abschlussarbeit

Entwicklung einer selbstüberwachten und robusten Systemarchitektur für einen Positionsdatenlogger.

Stichworte

Eingebettetes System, autonomes System, Positionsdatenlogger, Watchdog, fehlertolerante Programmierung, *C*, *Raspberry Pi*

Kurzzusammenfassung

Diese Arbeit umfasst die Entwicklung einer selbstüberwachten Systemarchitektur für einen Positionsdatenlogger, der in Nahverkehrsbussen eingesetzt wird. Es wird eine Systemarchitektur umgesetzt, die für die Erstellung von Routenverläufen und Geschwindigkeitsprofilen während der Fahrt zuständig ist und dabei eine hohe Fehlertoleranz aufweist.

Topic of the paper

Development of a self-monitoring and robust system architecture for a positiondatalogger.

Keywords

embedded system, stand-alone system, positiondatalogger, watchdog, fault-tolerant programming, *C*, *Raspberry Pi*

Abstract

This paper describes the development of a self-monitoring system architecture for a positiondatalogger which is used to create movement and velocity profiles in public busses. It is aiming for a high level of fault tolerance in hardware and software.

Inhaltsverzeichnis

1. Einleitung

Die Hamburger *HOCHBAHN* plant den jetzigen Busbetrieb auf einen emissionsfreien Busbetrieb umzubauen. Hierfür werden Informationen zur Umsetzbarkeit und Wirtschaftlichkeit von elektrisch angetriebenden Bussen benötigt.

Der Projektpartner Fraunhofer-Institut für Verkehrs- und Infrastruktursysteme (IVI) hat die Aufgabe mit Hilfe eines virtuellen Busmodells den elektrisch angetriebenden Bus zu simulieren und dadurch die geforderten Informationen zu generieren. Durch diese Simulation lässt sich außerdem eine Aussage über die theoretisch benötigte Ladeinfrastuktur machen.

Um eine möglichst realistische Simulation zu erhalten, werden Daten aus echten Bussen benötigt. Diese Aufgabe übernimmt das Projekt "Bewertung des Einsatzes von Elektrobussen mit Dezentraler Ladeinfrastruktur" (BEEDeL). Im Rahmen dieses der HAW Hamburg zugehörige Projekt wird ein Positionsdatenlogger entwickelt, welcher in Bussen eingebaut wird. Dieser Positionsdatenlogger zeichnet autonom während der Fahrt Routenverläufen und Geschwindigkeitsprofile auf und überträgt diese automatisch an das IVI.

Die grundlegende Systemarchitektur des Positionsdatenloggers wurde im Rahmen der Bachelorarbeit von Mario Wegner [1] entwickelt. Im Mai 2015 wurde der erste Prototyp in einen Bus eingebaut. Durch eine Vielzahl an Studien-, Bachelor- und Masterarbeiten, die im Rahmen des Projektes BEEDel entstanden sind, wurde der Positionsdatenlogger kontinuierlich weiterentwickelt und der Messbetrieb erweitert. Aktuell befinden sich 12 Positionsdatenlogger erfolgreich im Einsatz. [2]

1.1. Motivation

Da das System mit der Intention verbaut wird, als autonomes System zu fungieren, führen Systemausfälle zu einem Problem, welches nur durch Eingriff in den Betriebsablauf gelöst werden kann. Diesen Eingriff gilt es aus Kosten und Aufwandsgründen unbedingt zu vermeiden. Da es bei der zurzeit verwendeten Systemarchitektur, während des Betriebes, teilweise zu Abstürzen kam, soll im Rahmen dieser Arbeit eine neue Systemarchitektur entwickelt werden, die einen zuverlässigen Betrieb gewährleisten kann.

Dabei soll die zu entwickelnde Architektur im Vergleich zur bereits vorhandene Architektur ein verbessertes Fehlerverhalten besitzen. Die vorhandene Software muss also so überarbeitet werden, dass bei auftretenden Fehlern eine Selbstheilung ohne äußeren Eingriff möglich wird. Um dies zu erreichen soll eine Selbstüberwachung

realisiert werden. Zudem besteht die Aufgabe die Software hinsichtlich der Software-Qualität zu überarbeiten.

1.2. Aufbau der Arbeit

Die Arbeit beginnt zunächst mit einer Einführung wichtiger Grundbegriffe zum Thema Software-Sicherheit und der Vorstellung des aktuellen Positionsdatenloggers.
Danach wird, orientiert am V-Modell, zunächst die Anforderungsanalyse durchgeführt, mit deren Ergebnissen dann im Systementwurf die benötigten Architekturkonzepte identifiziert und dann spezifiziert werden.
In der Implementierung erfolgt die Auswahl der Architekturen und die Programmierung. Dabei wird die Realisierung der wichtigsten Kernfunktionen der neuen Systemarchitektur genauer dokumentiert.
Anschließend findet ein Test des Positionsdatenloggers statt. Bei diesem Test wird geprüft, ob die entwickelten Programme die Anfangs gestellten Anforderungen erfüllen können.
Zuletzt wird eine Zusammenfassung der Arbeit gegeben.

2. Grundlagen

In diesem Kapitel werden zunächst allgemeine Software-Konzepte, die zu einer Erhöhung der Systemrobustheit führen, erklärt. Hierbei muss zunächst geklärt werden, mit welchen Kriterien die Qualität von Software überhaupt gemessen wird. Dafür wird der Begriff Software-Qualität erläutert. Auf Grundlage dieses Begriffes wird dann die Strategie der fehlertoleranten Programmierung vorgestellt. Um ein allgemeines Verständnis von dem Positionsdatenlogger zu erhalten, wird außerdem auf die Hardware sowie die Software des aktuellen Systems eingegangen.

2.1. Software-Qualität

Der Begriff Software-Qualität wird in der ISO/IEC 25010:2011 (siehe Quelle [3]) wie folgt definiert:

"Software-Qualität ist die Gesamtheit der Merkmale und Merkmalswerte eines Software-Produkts, die sich auf dessen Eignung beziehen, festgelegte Erfordernisse zu erfüllen." [3]

Die Definition zeigt, dass es sich bei dem Begriff Software-Qualität um einen vielfältig einsetzbaren Begriff handelt. Software-Qualität lässt sich nicht über ein einziges zu erfüllendes Kriterium definieren. Der Begriff verbindet mehrere Qualitätskriterien, die bei der Entwicklung von qualitativ hochwertiger Software beachtet werden müssen. Im Folgenden werden diese Qualitätskriterien kurz aufgeführt und erläutert.

- **Funktionalität**: Die Grundmotivation Software zu entwickelt, entsteht durch den Nutzen der Software. Software besitzt immer einen zu erfüllenden Funktionsumfang. Dieses Qualitätsmerkmal beschreibt, inwieweit die Software ihre zugeschriebene Funktionalität erfüllt. Häufig wird dieses Kriterium mit dem Begriff Software-Qualität gleichgesetzt.
- **Laufzeit**: Dieses Kriterium beschreibt, in welchem Maß die Software ihre Laufzeitanforderungen erfüllt. Gerade bei zeitkritischen Systemen ist die Laufzeit ein ausschlaggebendes Qualitätsmerkmal.
- **Zuverlässigkeit**: Wie fehleranfällig ist das laufende System? Unter dieses Merkmal fällt auch die Fehlertoleranz. Gerade bei sicherheitskritischen Anwendungen ist es ein aussagekräftiges Merkmal.
- **Benutzbarkeit**: Bewertet die Qualität der Mensch-Maschine-Schnittstelle. Abhängig von der Benutzergruppe der Software.

- **Wartbarkeit**: Software, die nach der Inbetriebnahme ohne großen Aufwand korrigiert oder modifiziert werden kann, besitzt eine hohe Wartbarkeit.
- **Transparenz**: Wie strukturiert wurde die nach außen sichtbare Programmfunktionalität im Quellcode umgesetzt? Die Existenzberechtigung dieses Kriteriums wird vor allem bei Team-Projekten deutlich.
- **Übertragbarkeit**: Bewertet die Portierbarkeit (Plattformunabhängigkeit) von Software.
- **Testbarkeit**: Maß für die Qualität der Tests, die der Software unterzogen wurde.

Abschließend lässt sich sagen, dass der Begriff Software-Qualität eine Vielzahl an Kriterien umfasst. Es ist wichtig zu verstehen, dass sich die oben aufgeführten Qualitätskriterien gegenseitig beeinflussen. Der Versuch, ein Kriterium singulär zu erreichen, geht häufig mit der Optimierung einer ganzen Reihe von Kriterien einher. Außerdem ist es nicht möglich, alle Qualitätskriterien im gleichen Maße zu erfüllen, da einige Kriterien negativ und andere positiv miteinander korrelieren. Als Übersicht dieser Korrelationen dient die Tabelle 1. [4, Seite 7-9]

Tabelle 1: Korrelationsmatrix der verschiedenen Qualitätskriterien. Bild wurde modifiziert. Ursprüngliche Quelle siehe [4, Seite 11].

	Funktionalität	**Laufzeit**	**Zuverlässigkeit**	**Benutzbarkeit**	**Wartbarkeit**	**Transparenz**	**Übertragbarkeit**	**Testbarkeit**
Funktionalität		+	+	+	+	+	+	+
Laufzeit			+				-	+
Zuverlässigkeit					+	+		+
Benutzbarkeit						+		-
Wartbarkeit						+		
Transparenz							+	+
Übertragbarkeit								
Testbarkeit								

positive Korrelation = +
negative Korrelation = -
keine Korrelation = freies Feld

Die Abbildung zeigt die Zusammenhänge der oben aufgeführten Qualitätskriterien. Dabei steht ein Minuszeichen für eine negative und ein Pluszeichen für positive Korrelation. Bei einem leeren Feld besteht keine Korrelation. Das Kriterium Zuverlässigkeit beeinflusst die Kriterien Wartbarkeit und Testbarkeit in positiver Richtung. Das bedeutet, wenn die Software eine hohe Zuverlässigkeit besitzt, lässt sich die Softwäre häufig auch ohne großen Aufwand korrigieren. Haufig besitzt die Software eine hohe Zuverlässigkeit, weil über die Zeit immer wieder Verbesserungen implementiert wurden. Außerdem wurde eine derartige Software häufig vielfältigen Tests unterzogen, was ebenfalls die Zuverlässigkeit erhöht.

2.2. Fehlertolerante Programmierung

Um die fehlertolerante Programmierung zu erläutern, wird zuerst auf den Begriff Robustheit eingegangen. Dann werden einige Methoden aufgeführt, die in der fehlertoleranten Programmierung Anwendung finden.

2.2.1. Robustheit

Robustheit, auch als Fehlertoleranz bezeichnet, ist die Eigenschaft eines Systems, trotz unerwünschten Bedingungen noch zuverlässig zu funktionieren. Robustheit zählt zu dem Qualitätsmerkmal Zuverlässigkeit (siehe Kapitel 2.1), da sie einen direkten Einfluss auf die Zuverlässigkeit eines Systems hat.

Das Ziel der fehlertoleranten Programmierung ist es, die Robustheit eines Software-Systems zu erhöhen. Dabei kommen Methoden zum Einsatz, die zur Verbesserung des Fehlerverhaltens bei nichtvorgesehenen Systemzuständen führen. Nichtvorgesehene Systemzustände sind Zustände, die während des normalen Programmablaufs nicht auftreten. Einige dieser Methoden werden im Folgenden aufgeführt.

2.2.2. Error-Handling

Das Konzept Error-Handling (Ausnahmebehandlung) beschreibt die Möglichkeit, Fehler während der Programmausführung festzustellen und gegebenenfalls auf diese einzugehen.

Die 1967 von IBM entwickelte Sprache PL/I war die erste Sprache, bei der das Konzept des Error-Handlings, wie wir es heute kennen, Anwendung fand.

```
1 if(function() == EXIT_FAILURE)  //Erkennung des Fehlers
2 {
3     //Konzept zum Beheben des Fehlers.
4 }
```

Listing 1: Error-Handling-Beispiel

Wie aus dem Listing 1 zu entnehmen ist, führt das Auftreten eines Fehlers zwangsläufig zur Abänderung des Programmablaufs. Zuerst muss dafür gesorgt werden, dass der Fehler erkannt wird. Danach muss je nach Fehlerart ein Konzept ausgewählt werden, das in der Lage ist, den Fehler zu beheben. Da das Error-Handling in der Verantwortung des Programmierers liegt, werden auch nur die Fehler erkannt, die der Programmierer erwartet. Diese Fehler nennt man geplante Ausnahmen. Je nachdem wie gravierend die geplante Ausnahme ist, kann das Programm mit einer Fehlermeldung beendet werden oder es wird versucht, den Fehler zu beheben und mit dem Programmablauf fortzufahren. Beim Auftreten von ungeplanten Ausnahmen wird

das Error-Handling häufig vom Betriebssystem übernommen. Das Error-Handling des Betriebssystems hat meistens eine sofortige Beendigung des fehlerverursachenden Prozesses zur Folge. [4, Seite 105-106]

Error-Handling, welches zum Abbruch des Programms führt, hat eine Erhöhung der Transparenz (siehe Kapitel 2.1) zur Folge. Die Fehlertoleranz und Zuverlässigkeit (siehe Kapitel 2.1) eines Software-Systems wird erst erhöht, wenn das Error-Handling den Fehler beheben und der Programmablauf weiter fortgesetzt werden kann.

2.2.3. Watchdog

Ein Watchdog ist ein Konzept, welches häufig in sicherheitskritischen Systemen zum Einsatz kommt. Durch einen Watchdog wird die Selbstüberwachung des Systems erheblich erhöht. Je nach Anwendungsfall werden verschiedene Watchdog-Architekturen verwendet. Nun folgt ein Beispiel einer simplen Watchdog-Architektur.

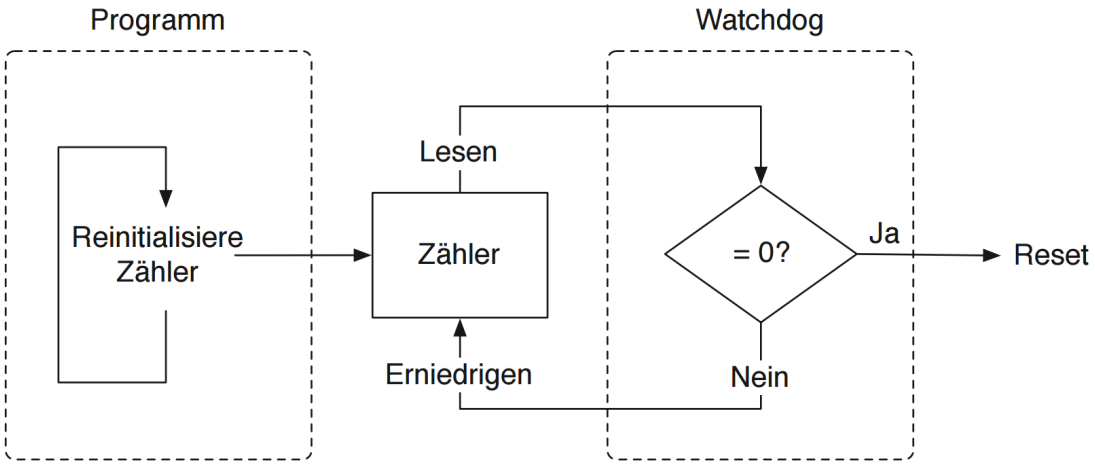


Abbildung 1: Beispiel Watchdog-Architektur [4, Seite 104]

Auf der rechten Seite der Abbildung sieht man das Programm, welches von dem Watchdog überwacht werden soll. Auf der linken Seite sieht man den Watchdog, welcher in Form einer Applikation auf derselben Hardware wie das zu überwachende Programm oder einer externen Hardware implementiert sein kann.

Bei der angewandten Logik dekrementiert der Watchdog periodisch einen Zähler. Im Normalbetrieb setzt die sicherheitskritische Software den Zähler ebenfalls periodisch auf einen Wert größer null. Wenn nun die Software abstürzt, erreicht der Zähler nach einer gewissen Zeit den Wert null und der Watchdog startet das Gesamtsystem neu. [4, Seite 104]

Durch diese einfache Logik wird das Qualitätsmerkmal Zuverlässigkeit erhöht. Gerade wenn viele Hardware-Komponenten in einem System untergebracht sind, ist die Verwendung eines Watchdogs sinnvoll, da dieser bei einem kritischen Fehler einen Hardware-Reset durchführen kann.

2.2.4. *MISRA-C*-Sprachkonvention

MISRA-C ist ein C-Programmierstandard, der 1998 von der *Motor Industry Software Reliability Association* unter dem Namen *Guidelines for the Use of the C Language in Vehicle Based Software* veröffentlicht wurde. *MISRA* ist ein Zusammenschluss aus verschiedenen Firmen aus der Automobilbranche, die sich zum Ziel gesetzt hat, die Qualität von Software in eingebetteten Systemen, vor allem im Automobilbereich, durch die *MISRA-C*-Sprachkonvention zu erhöhen. Dabei soll der Software-Entwickler durch, die von der Sprachkonvention aufgestellten Regeln vor *C*-typischen Programmierfehlern bewahrt werden. [4, Seite 74]

Dabei bestand die ursprünglich entwickelte Sprachkonvention aus 127 Regeln, die sich in required rules und advisory rules, also verpflichtende und empfohlene Regeln, unterteilen. Die *MISRA-C*-Sprachkonvention findet heutzutage nicht mehr nur im Automobilsektor Anwendung. Aus dem Grund beinhaltet die Ausgabe aus dem Jahr 2004 141 Richtlienen, die allgemein auf sicherheitskritische Systeme angewandt werden können. Durch die verstärkte Verwendung der Sprachkonvention gibt es mittlerweile statische code-analyse Werkzeuge, die eine automatische *MISRA-C*-Prüfung integriert haben. [4, Seite 75]

Zusammenfassend lässt sich sagen, dass durch eine *MISRA-C*-Konformität die Software-Architektur eine stärkere Fehlertoleranz erhält.

2.3. Hardware des Positionsdatenloggers

Um einen besseren Überblick über das System zu erhalten, wird in diesem Kapitel eine Systemanalyse der Hardware des aktuellen Positionsdatenloggers durchgeführt. Dabei wird auf jede verbaute Hardware-Komponente kurz eingegangen. Die Komponenten wurden in der Bachelorarbeit von Mario Wegner ausgewählt und implementiert (siehe Quelle [1]).

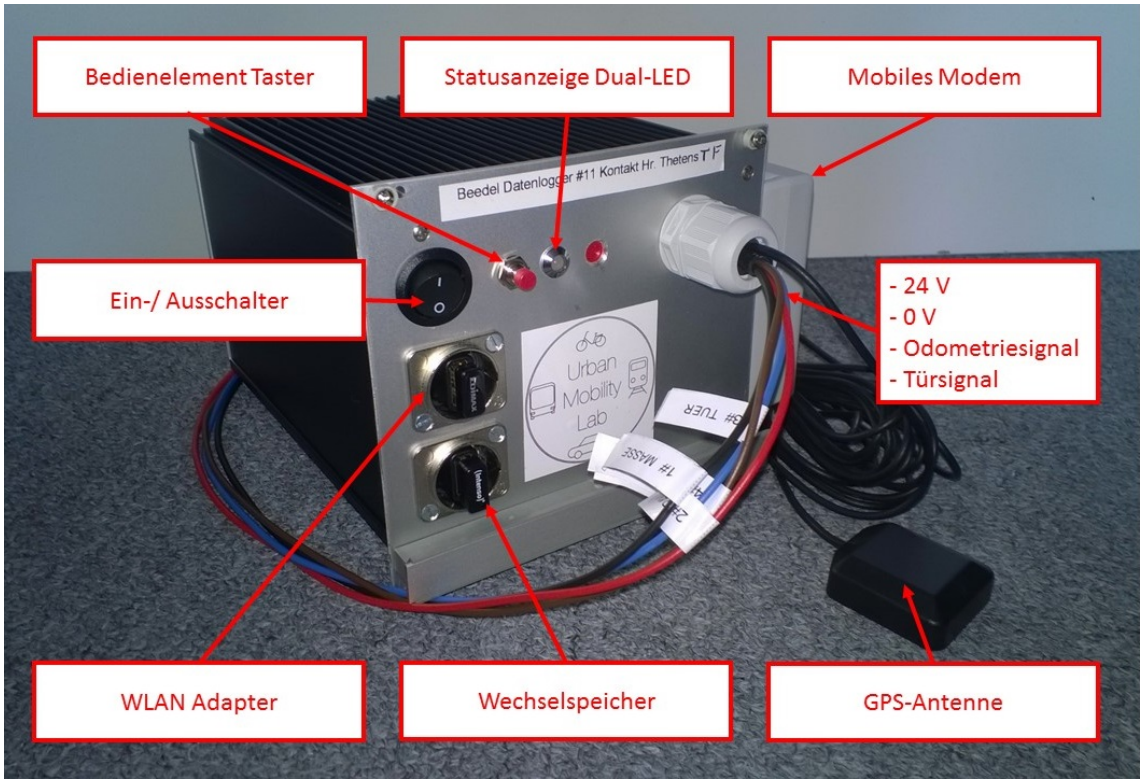


Abbildung 2: Positionsdatenlogger [1, Seite 73]

2.3.1. Übersicht über die Hardware-Architektur

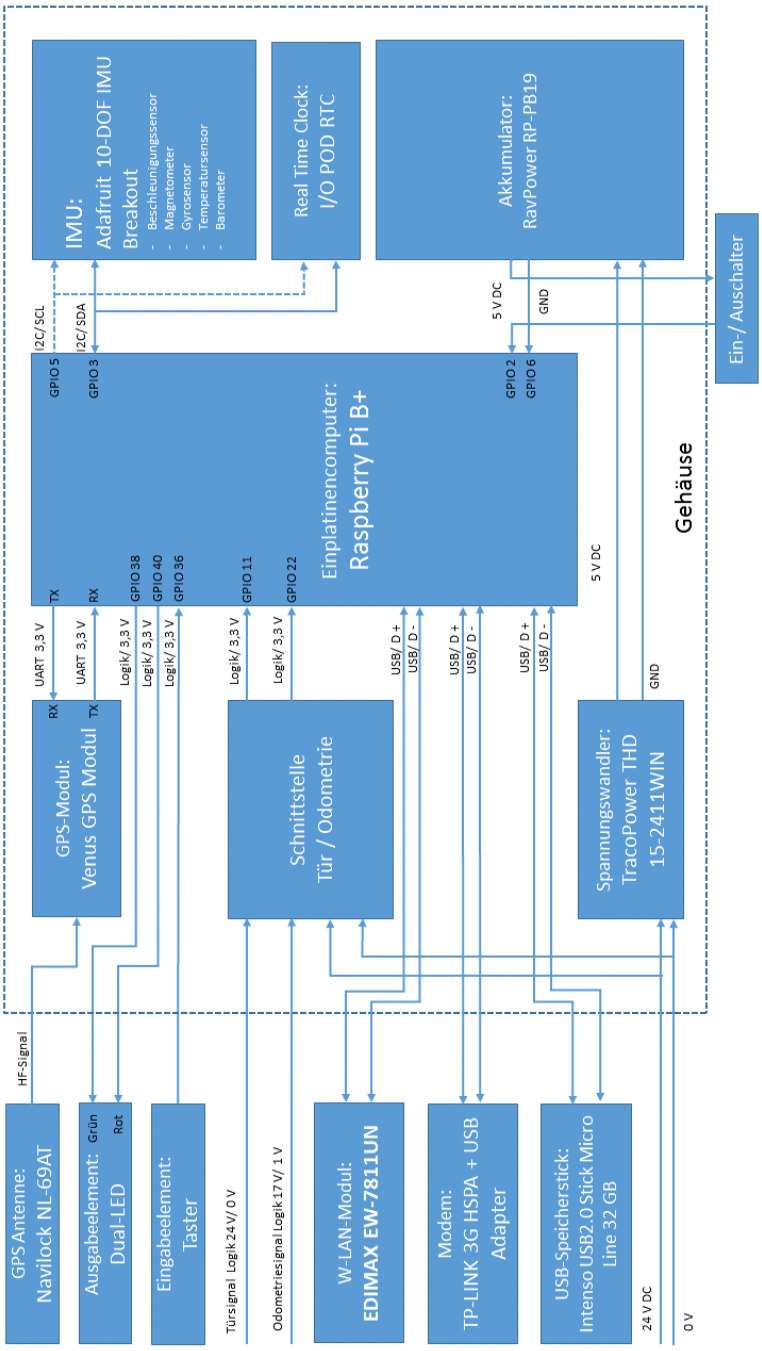


Abbildung 3: Hardware-Systemarchitektur [1, Seite 72]

2.3.2. Einplatinencomputer

Der GPS-Logger baut auf dem Einplatinencomputer *Raspberry Pi 1* auf. Dieser Einplatinencomputer wurde von der *Raspberry Pi Foundation* entwickelt. Diese gemeinnützige Organisation hat sich ursprünglich zur Aufgabe gemacht, eine kostengünstige programmierbare Hardware zu entwickeln, um Kindern die Möglichkeit zu geben, sich spielerisch mit einem System auseinanderzusetzen und dabei die Grundkonzepte der Programmierung zu lernen. [5]

Verbaut wurde das Model B+, welches die finale Version der ersten *Raspberry Pi* Generation ist. Der *Raspberry Pi* zeichnet sich durch seine kleine Baugröße von 85,60 mm x 56,5 mm aus, was ungefähr die Größe einer Kreditkarte entspricht. Der Preis der Platine liegt bei 42,99 €. [6] [7]

Alle Modelle aus der ersten Generation verfügen über einen "system on a chip" (SoC) von *Boardcom*. Das SoC beinhaltet unter anderem einen *ARM1176 Processor*. Dieser Prozessor ist mit 700 MHz getaktet. Der verfügbare Arbeitsspeicher (RAM) beträgt 512 MByte. Das Modell B+ hebt sich durch eine größere Anzahl an Schnittstellen von den älteren Generationen ab. Es stehen 40 GPIO-Pins, vier USB-Ports, ein Ethernet-Port und ein HDMI-Video-Ausgang zur Verfügung. Die Leistungsaufnahme beträgt 3.0 W bei 5.0 V und erfolgt entweder über eine MicroUSB-Schnittstelle oder über zwei GPIO-Pins. Einige der GPIO-Pins lassen sich für Kommunikations-Protokolle (UART, I2C, SPI) verwenden. Eine Übersicht der GPIO-Pins des Modells B+ ist dem Anhang beigefügt (siehe Anhang C.1). [6]

Da der *Raspberry Pi* so vielseitig einsetzbar ist, hat sich im Laufe der Zeit eine große Benutzer-Gemeinschaft gebildet. Dadurch lässt sich über das Internet auf eine umfangreiche Wissensansammlung zugreifen. Außerdem steht eine weitreichende Dokumentation der *Raspberry Pi* Hardware sowie eine Vielzahl an bereits implementierter Software zur Verfügung.

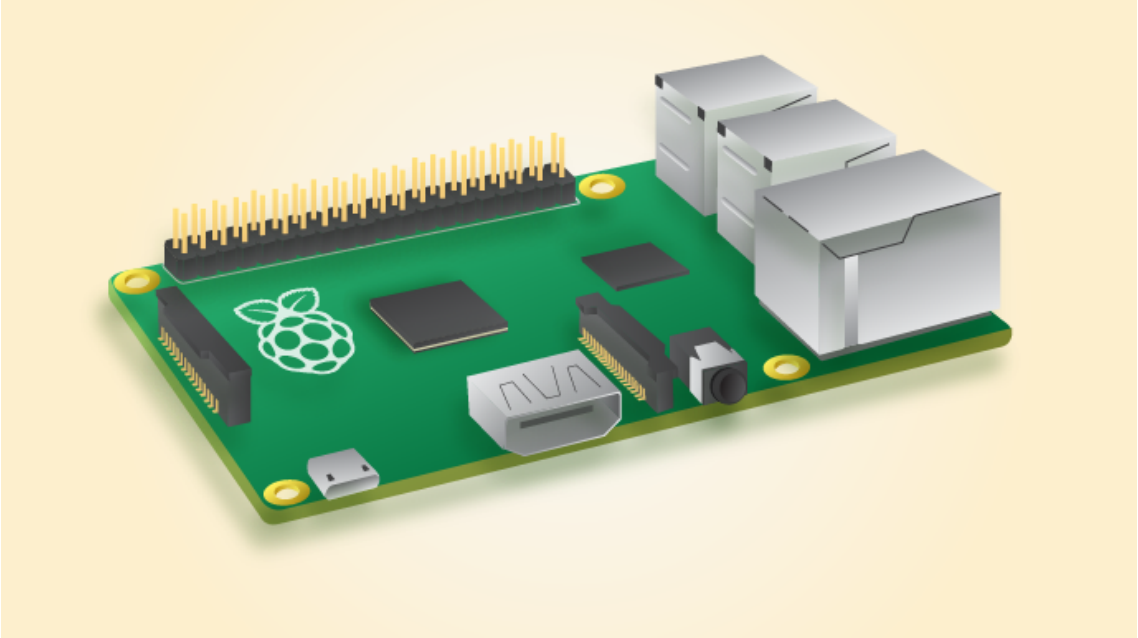


Abbildung 4: Raspberry Pi 1 Model B+ [6]

2.3.3. SD-Karte

Durch die eingelegte SD-Karte wird dem Einplatinencomputer das Betriebssystem und der verwendbare Speicher zur Verfügung gestellt. Es handelt sich um eine MicroSD-Karte (class10) von *INTENSO*. Auf der SD-Karte stehen 16 GB Speicher zur Verfügung, wobei ungefähr 2 GB von dem Betriebssystem, welches mit dem *Yocto-Pojekt* erstellt wurde (siehe Kapitel 2.4.4), belegt sind.

2.3.4. GPS-Empfänger

Um die GPS-Daten zu empfangen, wird das *SparkFun Venus GPS Modul* von *Skytraq* verwendet. Hierbei handelt es sich um ein GPS-Modul, welches eine Update-Rate von bis zu 20 Hz liefert. Außerdem wird der NMEA 0183 Standard unterstützt. Dieser Standard wurde ursprünglich für die Kommunikation zwischen Schiffen und Navigationsgeräten entwickelt. Heutzutage wird der Standard auch für die Kommunikation zwischen GPS-Empfängern und Computern verwendet, um fest formatierte ASCII-Zeichenketten zu definieren. Diese Zeichenketten beginnen immer mit "$GP" und darauf folgen je nach gewünschtem Datenformat unterschiedliche GPS-Informationen. [1, Seite 18, 46]

Der GPS-Empfänger des Positionsdatenloggers arbeitet mit einer Update-Rate von 10 Hz und ist so konfiguriert, dass bei jeder Messung eine GGA- und eine RMC-Zeichenkette erstellt wird. Ein GPS-Messpunkt besteht also aus zwei Zeichenketten, die beide unterschiedliche Informationen beinhalten. Um eine bessere Funk-Verbindung zu gewährleisten, wird eine externe GPS-Antenne mit einem SMA-Stecker angeschlossen. Die Datenverbindung zum Einplatinencomputer wird über UART hergestellt.

```
1 $GPGGA,194739.673,5314.4514,N,01024.7830,E,1,08,1.1,16.4,M,43.0,M,,0000*6F
```

Listing 2: Beispiel einer GGA-Zeichenkette

Informationen, die von dem Positionsdatenlogger gespeichert werden:

- 5314.4514 - Breitengrad in Grad und Dezimalminuten
- 01024.7830 - Längengrad in Grad und Dezimalminuten
- 08 - Anzahl verwendeter Satelliten
- 16.4 - Höhe über NN

```
1 $GPRMC,194739.673,A,5314.4514,N,01024.7830,E,027.2,196.9,280516,,,A*69
```

Listing 3: Beispiel einer RMC-Zeichenkette

Informationen, die von dem Positionsdatenlogger gespeichert werden:

- 027.2 - Geschwindigkeit in Knoten
- 196.9 - Richtung in Grad

[8, Seite 14, 18] Im weiteren Verlauf dieser Arbeit wird das GPS-Modul mit zu den Sensoren gezählt.

2.3.5. Inertial Measurement Unit

Um neben den GPS-Informationen noch weitere Sensorwerte aufzuzeichnen, befindet sich in dem Positionsdatenlogger die *Adafruit 10 DOF IMU Breakout*. Hierbei handelt es sich um einen Zusammenschluss aus Sensoren auf einem Chip. Die IMU liefert Werte für Beschleunigung, Magnetfeld, Drehrate und Druck/Temperatur. Die IMU wird über die I2C-Schnittstelle mit dem *Raspberry Pi* verbunden. [9]

2.3.6. Odometrie-Signal

Um eine genaue Information über die Busgeschwindigkeit zu erhalten, wird das Odometrie-Signal, welches im Bus für den Kilometerzähler verwendet wird, mit aufgezeichnet. Dieses ist mit vier Pulsen pro Meter aufgelöst. Es wird davon ausgegangen, dass der Bus eine Höchstgeschwindigkeit von 100 km/h nicht überschreitet. Dadurch lässt sich die maximal Frequenz des Signals auf 111 Hz festlegen. Das Odometrie-Signal wird mit Hilfe einer Interrupt Service Routine (ISR) ausgelesen (siehe Kapitel 2.4.2).

2.3.7. Tür-Signal

Bei dem Tür-Signal handelt es sich um ein weiteres, bereits im Bus vorliegendes Signal, welches den Zustand der Bustüren an den Positionsdatenlogger übermittelt. Ein HIGH-Pegel liegt an, wenn alle Bustüren geschlossen sind und ein LOW-Pegel, wenn eine der Türen geöffnet ist.

2.3.8. Real Time Clock

Um die korrekte Zeit beim Start auf dem Einplatinencomputer zu gewährleisten, wird das RTC-Modul *CISECO I/O POD RTC* verwendet. Dies wird über die I2C-Schnittstelle, die parallel auch von der IMU verwendet wird, ausgelesen. [1, Seite 51]

2.3.9. Mobiler Internetzugang

Die Sensordaten werden nach Beendigung des Logging-Prozesses und der darauf folgenden Datenauswertung auf einen Server hochgeladen. Um den Positionsdatenlogger mit dem Internet zu verbinden wird das mobile GSM-Modem *TP-Link 3G HSPA* verwendet. Es wird über einen Adapter mit einem USB-Port des Einplatinencomputers verbunden. [1, Seite 50]

2.3.10. Funkschnittstelle

Da das Gehäuse nicht dafür vorgesehen ist, Bildschirm, Maus und Tastatur anzuschließen, verfügt der Logger über eine drahtlose Schnittstelle, die bei Debug-Sessions den Zugang auf den Logger gewährleistet. Über den WLAN-Stick *EDIMAX EW-7811UN*, der über einen USB-Port mit dem *Raspberry Pi* verbunden ist, lässt sich mit Hilfe eines externen Computers ein Zugriff auf den Logger herstellen. [1, Seite 49]

2.3.11. Wechselspeicher

Um die SD-Karte zu entlasten werden die Rohdaten, die bei der Datenaufzeichnung entstehen, auf einen *Intenso MICRO LINE USB Stick* gespeichert. Dieser besitzt mit 32 GB ausreichend Speicher, um Rohdaten eines Monats zu speichern. Durch die Verlagerung der Rohdaten auf einen externen Speicher wird außerdem die Sicherheit der Datenspeicherung erhöht, da bei einem Ausfall der SD-Karte und damit auch des Betriebssystems die Rohdaten weiterhin auf dem USB-Stick verfügbar sind. [1, Seite 50]

2.3.12. Akkumulator

Die 5 V Spannungsversorgung des Positionsdatenloggers erfolgt über einen Akkumulator. Verwendet wird der *RavPower RP-PB19-USB-Akkumulator*. Dieser kann den Positionsdatenlogger bei maximaler Belastung über eine Dauer von 19,5 Stunden mit Energie versorgen. Um den Akkumulator mit dem *Raspberry Pi* zu verbinden, wird auf der Akkumulatorseite ein USB-Stecker und *Raspberry Pi*-seitig die dafür vorgesehenen GPIO-Pins verwendet. [1, Seite 51-52]

2.3.13. Spannungswandler

Um den Akkumulator zu laden, wird die businterne Betriebsspannung (24 V) verwendet. Diese ist nur verfügbar, wenn die Zündung des Busses betätigt wurde. Da der Akkumulator eine Ladespannung von 5 V benötigt, wird der DCDC-Wandler *TracoPower THD 15-2411WIN* verwendet. Dieser wandelt die Spannung von 24 V auf 5 V und trennt das Busnetz sowie das Netz des Positionsdatenloggers galvanisch. [1, Seite 52]

2.3.14. Gehäuse

Bei dem Gehäuse handelt es sich um einen 19-Zoll-Einschub, der in den im Bus vorhandenen 19-Zoll-Rahmen eingebaut wird. Da das Gehäuse aus Aluminium ist, wurde das GSM-Modem sowie der WLAN-Stick außerhalb des Gehäuses angebracht.

An der Vorderseite des Gehäuses befindet sich eine Dual-LED, die im Takt der Auslesung des GPS-Modules, das heißt mit 10 Hz, blinkt. Die LED leuchtet grün, wenn sich der Datenlogger im Wartezustand befindet. Wenn der Logging-Vorgang startet oder die Rohdaten ausgewertet werden, leuchtet die LED rot. Außerdem wurde ein Taster angebracht, um die Datenaufzeichnung zu starten oder zu stoppen. [10]

2.4. Software des Positionsdatenloggers

Die Software des Positionsdatenloggers lässt sich in mehrere Programme unterteilen. In diesem Kapitel wird auf das Logging-Programm, das Auswertungs-Programm und das Startup-Skript eingegangen. Entwickelt wurde die Software ebenfalls im Rahmen der Bachelorarbeit von Mario Wegner (siehe Quelle [1]).

2.4.1. Übersicht über die Software-Architektur

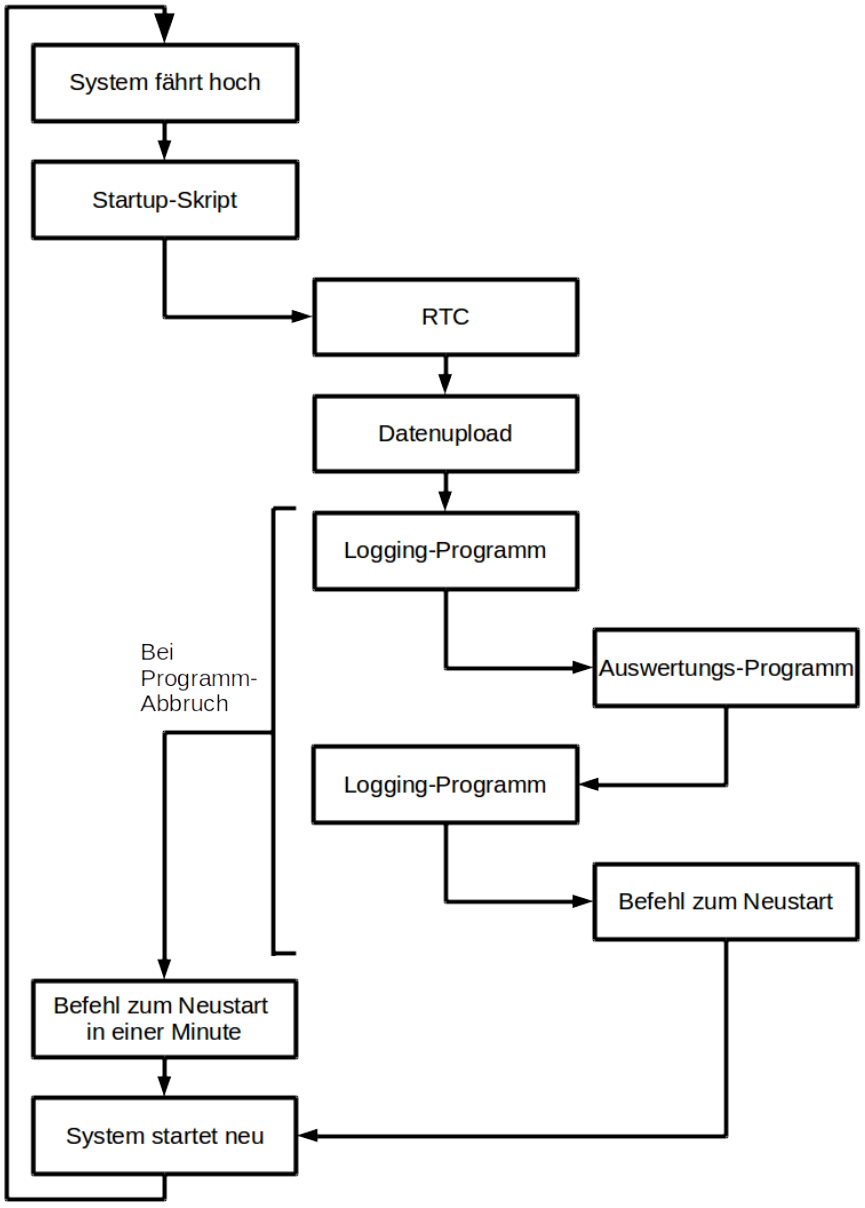


Abbildung 5: Übersicht über Systemablauf

Da es sich bei dem Positionsdatenlogger um ein autonomes System handelt, muss eine schleifenartige Architektur gewährleistet sein. Dieses wird unter anderem dadurch erreicht, dass der Programmablauf in jedem Fall mit einem Neustart des Systems abschließt. Dieses Verhalten wird nun mit der Verwendung eines Startup-Skriptes (siehe Kapitel 2.4.4) verknüpft, wodurch die schleifenartige Architektur entsteht.

2.4.2. Logging-Programm

In der Abbildung 6 ist das Programm aufgeführt, welches für die eigentliche Datenaufzeichnung zuständig ist. Das Programm ist in der Programmiersprache *C* geschrieben. Im Folgenden wird die Grundarchitektur kurz erklärt.

Zu Beginn werden die GPIO-Pins mit Hilfe der *wiringPi*-Bibliothek initialisiert. *WiringPi* ist eine in *C* geschriebene Bibliothek, die die Verwendung der GPIO-Pins des *Raspberry Pis* vereinfacht. Sie unterliegt der *free software license*, genauer der *GNU LGPLv3 license*, und darf somit in eigenen Projekten verwendet werden. [11]

Nach der Initialisierung der Sensoren und der Erstellung der Dateien, in die die Sensorwerte geschrieben werden, geht das Programm in eine while-Schleife. In dieser while-Schleife wird der eigentliche Aufzeichnungsprozess durchgeführt. Dabei ist die Geschwindigkeit der Auslesung aller Sensoren von der Schreibgeschwindigkeit des GPS-Modules abhängig.

Um die ankommenden Odometrie-Pulse (siehe Kapitel 2.3.6) auswerten zu können, wird eine ISR verwendet. Dabei wird der GPIO-Pin festgelegt, auf dem das Odometrie-Signal ankommt. Immer wenn nun eine steigende Flanke an dem Pin erkannt wird, wird der Programmablauf unterbrochen und die ISR wird abgearbeitet. Wenn die ISR ausgeführt wurde, wird der Programmablauf wieder aufgenommen. In diesem Fall ist die ISR eine Funktion, die die ankommenden Pulse zählt. Die Odometrie-Pulse werden also auch gezählt, wenn das Programm gerade die while-Schleife durchläuft.

Ein Durchlauf durch die while-Schleife sieht wie folgt aus. Das GPS-Modul schreibt die empfangenden NMEA-Zeichenketten in einen Buffer. Das Logging-Programm liest so lange den Buffer ein, bis der erste GPS-Messpunkt (GGA- und RMC-Zeichenkette) erkannt wird ("poll"-Methode). Daraufhin wird der Messpunkt in der dafür vorgesehenen Datei gespeichert und die anderen Sensorwerte werden ausgelesen ("pull"-Methode) sowie ebenfalls gespeichert.

Diese Architektur hat zur Folge, dass alle Sensoren mit 10 Hz ausgelesen werden. Wenn die Datenaufzeichnung beendet ist, werden alle geöffneten Ressourcen ent-

sprechend geschlossen und das Auswertungs-Programm wird gestartet.

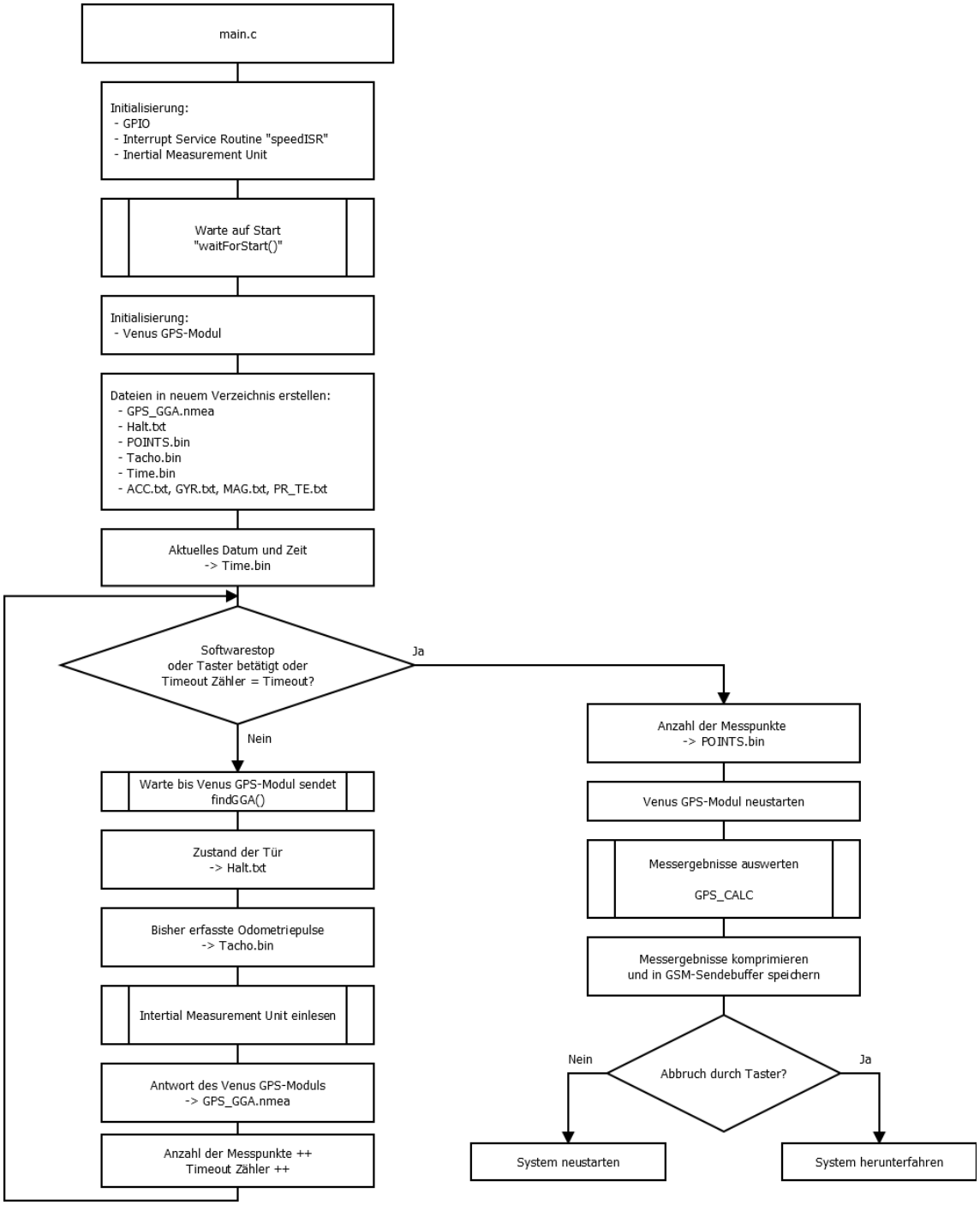


Abbildung 6: Flussdiagramm des Logging-Programms [1, Seite 63]

Im Laufe dieser Arbeit wird dieses Programm überarbeitet.

2.4.3. Auswertungs-Programm

In der Abbildung 7 ist das Programm aufgeführt, welches für die Datenauswertung und das darauffolgende Erstellen der mat-Datei zuständig ist. Durch die Verwendung einer mat-Datei lassen sich die Sensorwerte direkt in *MATLAB* auswerten. Die mat-Datei, die vom Auswertungs-Programm erzeugt wird, enthält nur die Sensorwerte des GPS-Modules, die Anzahl an Odometrie-Pulsen und die Information zum Türstatus. Das Programm ist in der Programmiersprache *C* geschrieben. Im Folgenden wird die Grundarchitektur kurz erklärt.

Zu Beginn werden die zuvor erstellten Dateien, welche die Sensorwerte beinhalten, geöffnet. Darauf werden die gemessenen Werte in einer for-Schleife ausgelesen und in Variablen gespeichert. Diese Variablen werden verwendet, um die mat-Datei zu erstellen. Dabei werden nicht alle Informationen des GPS-Modules verwendet.

Zur Erstellung der mat-Datei wird die Bibliothek *Matio* verwendet. *Matio* ist eine open-source *C* Bibliothek um mat-Dateien zu lesen und zu erstellen. Die Bibliothek ist unter der *BSD license* veröffentlicht und darf somit frei verwendet werden. [12]

Zum Schluss werden die Rohdaten zusammen mit der mat-Datei in ein zip-Archiv verpackt und in einen Ordner abgelegt. Dieser Ordner wird bei jedem Startprozess des Positionsdatenloggers auf den Server hochgeladen (siehe Kapitel 2.4.4).

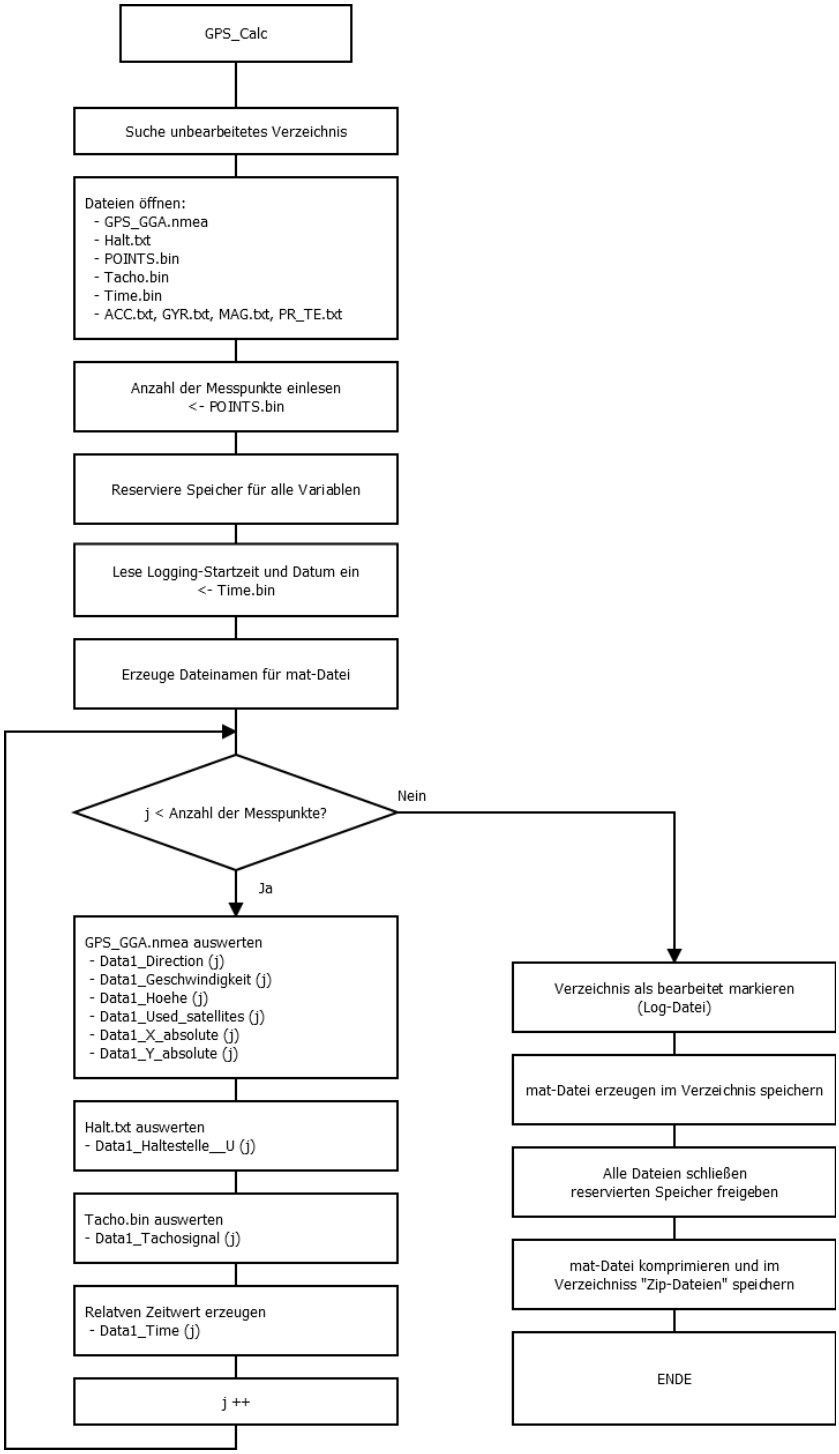


Abbildung 7: Flussdiagramm des Auswertungs-Programms [1, Seite 65]

Im Laufe dieser Arbeit wird dieses Programm überarbeitet.

2.4.4. Startup-Skript

Um das Logging- und Auswertungs-Programm in dem Betriebssystem zu integrieren, wird ein Startup-Skript verwendet. Dieses *Shell*-Skript wird nach dem Hochfahren des Betriebssystems ausgeführt.

Das zurzeit verwendete Betriebssystem wurde von Richard Günther mit Hilfe des *Yocto-Projektes* generiert. Das *Yocto-Projekt* ist ein Open-Source-System, dass Werkzeuge zur Erstellung von Linux-basierten Betriebssystemen zur Verfügung stellt. Dadurch kann ein Betriebssystem generiert werden, welches auf bestimmte Aufgaben speziallisiert ist. Das für den Positionsdatenlogger erstellte Betriebssystem besitzt zum Beispiel kein GUI, um den Prozessor des Einplatinencomputers nicht unnötig zu belasten. [13]

In Abbildung 8 sind die Befehle aufgeführt, die das Skript ausführt.

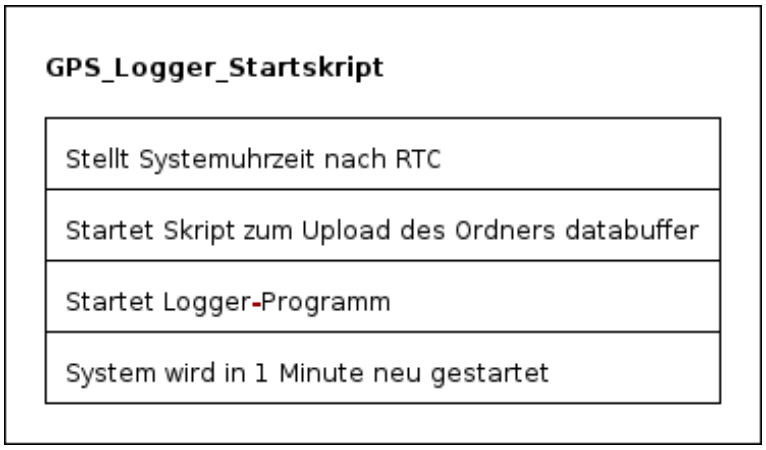


Abbildung 8: Struktogramm des GPS-Logger-Startskripts

Um eine korrekte Uhrzeit zu gewährleisten, wird am Anfang die Systemzeit mit Hilfe der RTC gestellt. Dann wird das Skript für den Datenupload gestartet. Durch dieses Shell-Skript werden alle Daten, die im Ordner "/home/pi/databuffer" liegen, auf den Server geladen. [14]

Anschließend startet das Logging-Programm und die Datenaufzeichnung beginnt. Wenn diese abgeschlossen ist, wird vom Logging-Programm aus der Prozess der Auswertung gestartet. Wenn dieser abgeschlossen ist, wird der *Raspberry Pi* durch das

Logging-Programm neu gestartet. Falls im Logging- oder Auswertungs-Programm ein Fehler auftritt, der dafür sorgt, dass das Programm abbricht, wird der Vorgang zum Neustarten von dem Startup-Skript nach einer Minute eingeleitet. Der Befehl zum Neustarten wurde also aus Sicherheitsgründen in dem Startup-Skript implementiert.

3. Anfordungsanalyse

Da die zu entwickelnde Systemarchitektur auf dem bereits im Einsatz befindlichen Programm aufbaut, sind die Anforderungen, welche die Aufnahme und Bereitstellung der Messdaten betreffen, bereits identifiziert. Um Anforderungen zu erkennen, die zu einer Erhöhung der Fehlertoleranz führen, werden möglichen Szenarien erstellt. Bei diesen Szenarien handelt es sich um Anwendungsfälle, die bei der Benutzung des Positionsdatenloggers auftreten können. Dadurch lassen sich Anforderungen analysieren, die das autonome Verhalten des Systems beeinflussen.

Dabei werden zuerst die möglichen Szenarien aufgestellt. Im nächsten Schritt wird genauer auf einzelne Szenarien und die Anforderungen, die diese mitbringen, eingegangen. Danach werden Anforderungen, die aus der Aufgabenstellung und dem Gesamtkontext hervorgehen, aufgezeichnet. Zum Schluss dieses Kapitels wird eine Übersicht über alle identifizierten Anforderungen aufgestellt.

3.1. Übersicht über Szenarien

In diesem Kapitel wird eine Übersicht über die möglichen Szenarien erstellt. Ein Szenario beschreibt eine bestimmte Zusammenstellung aus auftretenden Zuständen. Um die Szenarien besser zu trennen, erhält jedes Szenario einen Namen. Dieser Name beschreibt das aufgetretene Problem. Dabei ist hinzuzufügen, dass nicht jedes Problem gleich dazu führt, dass das System sein Ziel nicht erfüllt. Es werden unter anderem Zustände betrachtet, die in dem normalen Betrieb des Positionsdatenloggers vorkommen und nicht als Problem zu verstehen sind. Bei diesem Vorgehen werden nicht alle möglichen Szenarien betrachtet. Der Fokus liegt auf Szenarien, die den Programmablauf behindern.

Um alle möglichen Anwendungsfälle identifizieren zu können wird eine Tabelle angefertigt. Mit Hilfe dieser Tabelle lässt sich jede mögliche Zustandskombination schnell erzeugen. Bei diesem Vorgehen werden Anforderungen entstehen, die der jetzige Positionsdatenlogger bereits erfüllt und die in der Bachelorarbeit von Mario Wegner [1] identifiziert sowie umgesetzt wurden. Der Vollständigkeit halber werden diese Anforderungen trotzdem mit aufgeführt.

Tabelle 2: Szenarien mit jeweils einem nicht zutreffenden Zustand

Zustände / **Szenarien**	Akku geladen?	Bus-betriebs-spannung an?	Internet-verbindung verfügbar?	Fehler-freier *Raspberry Pi*-Betrieb?	Erfolg-reiche Sensor-auslesung?	Maximale Bus-geschwindig-keit eingehalten?
Normal Betrieb	✓	✓	✓	✓	✓	✓
Busstart bei ungeladenem Akku	×	✓	✓	✓	✓	✓
Akku wird nicht geladen	✓	×	✓	✓	✓	✓
Upload schlägt fehl	✓	✓	×	✓	✓	✓
System reagiert nicht mehr	✓	✓	✓	×	✓	✓
Auftreten eines Sensordefektes	✓	✓	✓	✓	×	✓
Odometrie-Pulse > 111 Hz	✓	✓	✓	✓	✓	×

ja = ✓, nein = ×

Es wurden sechs mögliche Szenarien identifiziert. Diese Szenarien werden nun im Einzelnen untersucht, um Anforderungen an ihnen abzuleiten.

3.1.1. Busstart bei ungeladenem Akku

Tabelle 3: Szenario "Busstart bei ungeladenem Akku"

Zustände / Szenarien	Akku geladen?	Bus-betriebs-spannung an?	Internet-verbindung verfügbar?	Fehler-freier *Raspberry Pi*-Betrieb?	Erfolg-reiche Sensor-auslesung?	Maximale Bus-geschwindig-keit eingehalten?
Busstart bei ungeladenem Akku	×	✓	✓	✓	✓	✓

ja = ✓, nein = ×

Die Systemarchitektur muss den Zustand des Busnetzes erkennen und bei Aktivierung des Netzes den Akkumulator laden. Außerdem muss der Positionsdatenlogger starten, sobald das Busnetz eingeschaltet und der Akkumulator geladen wird. Dieser Startvorgang muss auch bei einem nicht geladenen Akkumulator durchgeführt werden können.

Um kritische Zustände, die von der Spannungsversorgung ausgehen, wie zum Beispiel dieses Szenario, besser erkennen zu können, muss dem Positionsdatenlogger die Information über den Status des Busnetzes und dem Ladezustand des Akkumulators zur Verfügung gestellt werden. Diese Anforderung lässt sich außerdem mit dem Szenario "Akku wird nicht geladen" (siehe Kapitel 3.1.2) in Verbindung bringen. Auch hier ist die Information zum Ladestatus sinnvoll, um die Eigenüberwachung zu erhöhen.

Identifizierte Anforderungen:

- Durch Einschalten des Busnetzes muss es möglich sein, das System auch bei ungeladenem Akku zu starten.
- Aktueller Status des Busnetzes und des Akku-Ladezustands wird benötigt.

3.1.2. Akku wird nicht geladen

Tabelle 4: Szenario "Akku wird nicht geladen"

Zustände / Szenarien	Akku geladen?	Busbetriebsspannung an?	Internetverbindung verfügbar?	Fehlerfreier *Raspberry Pi*-Betrieb?	Erfolgreiche Sensorauslesung?	Maximale Busgeschwindigkeit eingehalten?
Akku wird nicht geladen	✓	×	✓	✓	✓	✓

ja = ✓, nein = ×

Der Positionsdatenlogger muss einen fehlerfreien Betrieb gewährleisten, auch wenn das Busnetz ausgeschaltet ist. Der Akkumulator muss also dazu fähig sein, das komplette System zu betreiben, ohne dabei geladen zu werden.

Diese Anforderung wurde bereits identifiziert und umgesetzt (siehe Quelle [1]).

3.1.3. Upload schlägt fehl

Tabelle 5: Szenario "Upload schlägt fehl"

Zustände / Szenarien	Akku geladen?	Busbetriebsspannung an?	Internetverbindung verfügbar?	Fehlerfreier *Raspberry Pi*-Betrieb?	Erfolgreiche Sensorauslesung?	Maximale Busgeschwindigkeit eingehalten?
Upload schlägt fehl	✓	✓	×	✓	✓	✓

ja = ✓, nein = ×

Beim Systemstart wird das Skript zum Hochladen der Daten ausgeführt. Falls zu diesem Zeitpunkt das Modem keine Internetverbindung aufbauen kann, müssen die

ausgelassenen Daten beim nächsten Systemstart mit hochgeladen werden. Als mögliche Ursache, kommt ein Hardwaredefekt des Modems oder ein Fehler bei der Ausführung des Skriptes in Frage.

Diese Anforderung wurde bereits identifiziert und umgesetzt (siehe Quelle [1]).

3.1.4. System reagiert nicht mehr

Tabelle 6: Szenario "System reagiert nicht mehr"

Zustände / Szenarien	Akku geladen?	Busbetriebsspannung an?	Internetverbindung verfügbar?	Fehlerfreier *Raspberry Pi*-Betrieb?	Erfolgreiche Sensorauslesung?	Maximale Busgeschwindigkeit eingehalten?
System reagiert nicht mehr	✓	✓	✓	×	✓	✓

ja = ✓, nein = ×

An diesem Szenario lassen sich mehrere Anforderungen ableiten. Zunächst muss erkannt werden, dass das System nicht mehr ausgeführt wird. Es wird also der aktuelle Status zur System-Ausführung benötigt.

Um den fehlerhaften Zustand zu verlassen, muss eine Maßnahme ergriffen werden, sodass der Positionsdatenlogger seine Aufgabe weiter ausführen kann. Da der unüberwindbare Zustand teilweise nur durch eine Trennung der Versorgungsspannung überwunden werden kann, muss die zu ergreifende Maßnahme einen solchen Zugriff beinhalten. Durch diese Maßnahme lassen sich Hardware- sowie Software-Fehler überwinden.

Identifizierte Anforderungen:

- Wiederherstellung des lauffähigen Zustands bei unüberwindbaren System-Fehlern.
- Aktueller Status der System-Ausführung wird benötigt.

3.1.5. Sensordaten nicht nach Spezifikation

Tabelle 7: Szenario "Sensordaten nicht nach Spezifikation"

Zustände / Szenarien	Akku geladen?	Bus-betriebs-spannung an?	Internet-verbindung verfügbar?	Fehler-freier *Raspberry Pi*-Betrieb?	Erfolg-reiche Sensor-auslesung?	Maximale Bus-geschwindig-keit eingehalten?
Falsches Format der Sensorwerte	✓	✓	✓	✓	×	✓

ja = ✓, nein = ×

Auch wenn bei der Auslesung der Sensoren das Format der aufzunehmenden Werte nicht dem erwarteten Format entspricht, darf die Datenaufzeichnung nicht abbrechen. Auch die darauf folgende Datenauswertung muss einen solchen Fehler erkennen und dem Benutzer mitteilen, was fehlgeschlagen ist. Außerdem muss gespeichert werden, welcher Sensor den fehlerhaften Wert gemessen hat.

Identifizierte Anforderungen:

- Fehlertoleranter Algorithmus zur Sensorauslesung und Datenauswertung.
- Erstellung eines Fehlerspeichers, um Informationen zu unerwarteten Systemzuständen zu speichern.

3.1.6. Odometrie-Pulse > 111 Hz

Tabelle 8: Szenario "Odometrie-Pulse > 111 Hz"

Zustände / Szenarien	Akku geladen?	Bus-betriebs-spannung an?	Internet-verbindung verfügbar?	Fehler-freier *Raspberry Pi*-Betrieb?	Erfolg-reiche Sensor-auslesung?	Maximale Bus-geschwindig-keit eingehalten?
Odometrie-Pulse > 111 Hz	✓	✓	✓	✓	✓	×

ja = ✓, nein = ×

Das Eintreten dieses Szenarios ist sehr unwahrscheinlich, da der Bus dafür schneller als 100 km/h fahren müsste (siehe Kapitel 2.3.6). Da aber auch die Möglichkeit besteht, dass ein fehlerhafter Odometrie-Sensor im Bus oder eine defekte Eingangsschaltung im Positionsdatenlogger Werte schneller als 111 Hz liefert, wird dieses Szenario trotzdem behandelt.

Die grundlegende Anforderung, die sich hieraus ableiten lässt, wurde bereits identifiziert und umgesetzt (siehe Quelle [1]). Diese Anforderung sieht einen fehlerfreien Betrieb bis zur maximal Geschwindigkeit des Busses vor. Durch dieses Szenario wird die bereits vorhandene Anforderung verschärft. Wenn der Bus über seiner maximalen Geschwindigkeit fährt, muss das Gesamtsystem weiterhin fehlerfrei arbeiten. Der Einplatinencomputer muss also eine Vielzahl an Interrupts bewältigen können, ohne dabei den Programmablauf der Datenaufzeichnung zu stören.

Identifizierte Anforderung:

- ISR darf Programmablauf zeitlich nicht behindern.

3.2. Allgemeine Anforderungen

Um eine erhöhte Flexibilität für den Einsatz des Positionsdatenlogger zu erhalten, sollen die einzelnen Sensormodule mit unterschiedlichen Frequenzen abgefragt werden können. Außerdem soll das Hinzufügen von weiteren Sensormodulen so umgesetzt werden, dass es ohne großen Aufwand durchgeführt werden kann.

Identifizierte Anforderungen:

- Unterschiedliche Sensorauslesefrequenzen einstellbar.
- Erhöhte Modularität beim Hinzufügen neuer Sensormodule.

3.3. Zusammenfassung der Anforderungen

Bereits identifizierte und umgesetzte Anforderungen werden nicht mit aufgeführt.

Tabelle 9: Zusammenfassung der Anforderungen

Teilsystem	**Anforderung abgeleitet aus Szenario**
Hardware	”Busstart bei ungeladenem Akku” 3.1.1: Durch Einschalten des Busnetzes muss es möglich sein, das System auch bei ungeladenem Akku zu starten.
	”System reagiert nicht mehr” 3.1.4: Wiederherstellung des lauffähigen Zustands bei nicht überwindbaren Hardware-Fehlern.
	”System reagiert nicht mehr” 3.1.4: Aktueller Status der Hardware-Ausführung wird benötigt.
Software	”Busstart bei ungeladenem Akku” 3.1.1: Aktueller Status des Busnetzes und des Akku-Ladezustands wird benötigt.
	”System reagiert nicht mehr” 3.1.4: Wiederherstellung des lauffähigen Zustands bei nicht überwindbaren Software-Fehlern.
	”System reagiert nicht mehr” 3.1.4: Aktueller Status der Software-Ausführung wird benötigt.
	”Sensordaten nicht nach Spezifikation” 3.1.5: Erstellung eines Fehlerspeichers, um Informationen zu unerwarteten Systemzuständen zu speichern.
	”Sensordaten nicht nach Spezifikation” 3.1.5: Robuster Algorithmus zur Sensorauslesung und Datenauswertung.
	”Odometrie-Pulse > 111 Hz” 3.1.6: ISR darf Programmablauf zeitlich nicht behindern.
	Allgemeine Anforderungen 3.2: Unterschiedliche Sensorauslesefrequenzen einstellbar.
	Allgemeine Anforderungen 3.2: Erhöhte Modularität beim Hinzufügen neuer Sensormodule.

4. Systementwurf

Die einzelnen Konzepte für die Entwicklung des Gesamtsystems werden im Folgenden erläutert. Dabei setzt sich das Gesamtsystem aus einzelnen Konzepten zusammen. Zuerst wird das Konzept für die Watchdog-Integration beschrieben. Anschließend folgt das Konzept für die Software-Architektur. Es werden zwei mögliche Architekturen vorgestellt.

4.1. Konzept für Watchdog-Integration

Um gegen das Szenario "System reagiert nicht mehr" (siehe Kapitel 3.1.4) zu bestehen, bietet sich die Verwendung eines externen Hardware-Watchdogs (siehe Kapitel 2.2.3) an. In diesem Kapitel wird ein Konzept zur Integration des Watchdogs in die Systemarchitektur erstellt.

Durch den Watchdog soll der Positionsdatenlogger in der Lage sein, eine Wiederherstellung des lauffähigen Zustands zu erreichen, auch wenn das Gesamtsystem nicht reagiert. Bei fehlender Reaktion des Gesamtsystems soll der Watchdog das System von der durch den Akkumulator zur Verfügung gestellten Versorgungsspannung trennen. Nach der Trennung wird gewährleisten, dass sich die verbaute Hardware sowie die Software in einem definierten Zustand befindet. Um nun einen lauffähigen Zustand zu erreichen, soll die Versorgungsspannung wieder aktiviert werden. Durch dieses Vorgehen lässt sich ein Aufhängen des Systems überwinden.

Um die fehlende Reaktion des Gesamtsystems zu erkennen, soll der Einplatinencomputer ein Signal an den Watchdog senden. Dieses Signal soll periodisch bei einer korrekten Systemausführung gesendet werden. Dadurch kann der Watchdog anhand des Signals auf den Status der Programmausführung des Positionsdatenloggers schließen. Wenn das Signal ausbleibt, befindet sich das System in einem nicht vorgesehenen Zustand und der Watchdog soll die Trennung durchführen.

Bevor die Trennung durchgeführt wird, soll der Watchdog dem *Raspberry Pi* ein Zeitfenster zur Verfügung stellen, in dem der *Raspberry Pi* die Möglichkeit hat das Gesamtsystem von sich aus neu zu starten. Um das Signal zum Herunterfahren an den *Raspberry Pi* zu senden, soll eine Kommunikation zwischen den beiden Systemen aufgebaut werden.

Auch bei dem Szenario "Busstart bei ungeladenem Akku" 3.1.1 soll der Watchdog eine zentrale Rolle spielen. Der Watchdog soll die Aktivierung des Akkumulators übernehmen, sobald das Busnetz eingeschaltet wurde. Dies soll auch funktionieren, wenn der Akkumulator leer ist und beim Einschalten des Busnetzes der Lade- sowie

Entladevorgang zeitgleich beginnt. Um die Informationen zur Spannungsversorgung zu übertragen, muss die Kommunikation zwischen den beiden Systemen erweitert werden. Hinzukommt die Übertragung des Status des Busnetzes und die aktuell am Positionsdatenlogger anliegende Spannung.

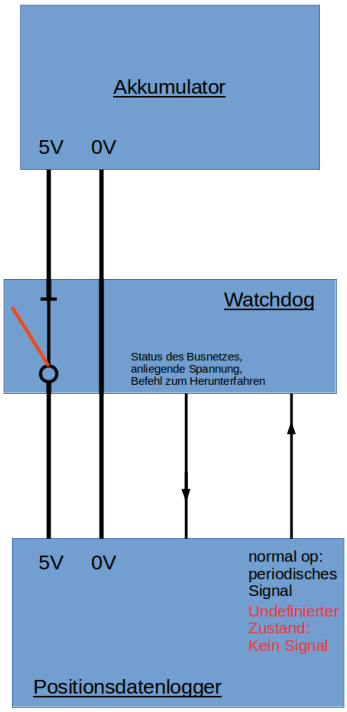


Abbildung 9: Watchdog-Konzept

4.2. Konzept für Software-Architektur

Da das aktuelle Programm des Postitionsdatenloggers die in Kapitel 3.3 genannten Anforderungen nicht erfüllen kann, wird eine neue Software-Architektur benötigt.

Im Folgenden werden zwei verschiedene Architekturen erläutert, die in der Lage sind, die Anforderungen zu erfüllen. Ein Großteil der Anforderungen wird von dem Logging-Programm, also dem Programm, welches für die Sensorauslesung zuständig ist, übernommen. Aus diesem Grund werden sich die Architektur-Entwürfe stark auf

das Logging-Programm beziehen. Das Auswertungs-Programm wird dann je nach gewählter Architektur entsprechend angepasst (siehe Kapitel 5.4).

Die erste Architektur sieht vor, die einzelnen Sensoren innerhalb einer while-Schleife unterschiedlich schnell abzufragen. Darauf folgt die zweite Architektur, die mit Hilfe von Multithreading für jeden Sensor eine eigene while-Schleife vorsieht. Am Ende des Kapitels werden die Vor- und Nachteile der beiden Architekturen gegenübergestellt und begründet entschieden, welche Architektur implementiert wird.

4.2.1. Architektur-Entwurf: Zeitliche Anpassung der Sensorauslese-Schleife

Dieser Architektur-Entwurf baut stark auf dem bereits im Einsatz befindlichen Logging-Programm auf. Der Kernunterschied ist die Abänderung der Sensorauslese-Schleife. Um den Anforderungen aus 3.3 gerecht zu werden, soll eine sich zeitlich an eine Frequenz anpassende while-Schleife verwendet werden. Innerhalb dieser Schleife werden die Sensorauslesungen je nach gewünschter Auslesefrequenz für eine gewisse Anzahl an Durchläufen übersprungen, bevor sie ausgelesen werden. Dadurch lässt sich jedem Sensor eine unterschiedliche Auslesefrequenz zuordnen.

```
1  while{Bedingung}
2  {
3      if(Passt der jetzige Schleifen-Durchlauf zur Sensorfrequenz1?)
4      {
5          Auslesung Sensor1;
6      }
7      if(Passt der jetzige Schleifen-Durchlauf zur Sensorfrequenz2?)
8      {
9          Auslesung Sensor2;
10     }
11     Sende Lebenszeichen an Watchdog;
12     Anpassung der Ausführungszeit;
13 }
```

Listing 4: Konzept: Zeitliche Anpassung der Sensorauslese-Schleife

Durch diesen Aufbau wird die while-Schleife mit einer konstanten Frequenz ausgeführt. Indem die Auslesung so lange übersprungen wird, bis die gewünschte Sensorfrequenz mit der Schleifenfrequenz übereinstimmt, können die Sensoren zu einstellbaren Frequenzen ausgelesen werden. Dabei wird die maximal Frequenz durch die Ausführungsfrequenz der while-Schleife vorgegeben. Um die so aufgenommenen Messwerte im Auswertungs-Programm zuordnen zu können, soll jeder Wert mit einem Zählindex markiert werden. Außerdem soll die while-Schleife das periodische Signal für den Watchdog erzeugen (siehe Kapitel 4.1).

4.2.2. Architektur-Entwurf: Multithreading

Es folgt eine kurze Erklärung des Prinzips Multithreading. Danach wird näher auf den Systementwurf eingegangen.

Ein Thread ist die kleinste Sequenz von programmierten Anweisungen, die von dem Betriebssystem gehandhabt werden kann. Der gerade ausgeführte Prozess beinhaltet immer mindestens einen Thread. Durch Multithreading erhält ein Prozess die Fähigkeit des Multitaskings. Im Folgenden wird aus Übersichtsgründen das Hardware-Multithreading, welches vom *Raspberry Pi* für gewisse Befehle Anwendung findet, ignoriert. Da der *Raspberry Pi* auf einem Single-Core-Prozessor aufbaut, wird das Prinzip der Nebenläufigkeit (engl.: concurrency) angewandt, um Multitasking zu erreichen. Wenn der Prozess mehrere Threads beinhaltet, werden diese von der CPU nebenläufig ausgeführt. Dies wird durch das Betriebssystem mit Hilfe von "context switching" realisiert. Dabei sorgt das Betriebssystem für ein ständiges Wechseln zwischen den Threads. Für den Benutzer fühlt es sich so an, als ob die Threads zeitgleich ausgeführt werden. Aus dem Grund spricht man auch von Software-Multithreading, welches eine illusionäre Nebenläufigkeit erzeugt. [15][16]

Jedem Prozess wird ein bestimmter Bereich an Ressourcen zur Verfügung gestellt. Wenn der Prozess nun mehrere Threads enthält, greifen die verschiedenen Threads auf den gleichen Ressourcenbereich zu. Um nun ungewünschte Interaktionen zwischen den Threads zu vermeiden, müssen in den Threads verwendete Funktionen die Eigenschaft "thread-safe" besitzen. Da bei der folgenden Architektur die Threads auf der Benutzerebene erzeugt werden, ist es noch wichtig zu erwähnen, dass das Umschalten zwischen den Threads von der entsprechenden Threadbibliothek und nicht von dem Betriebssystem übernommen wird. [16]

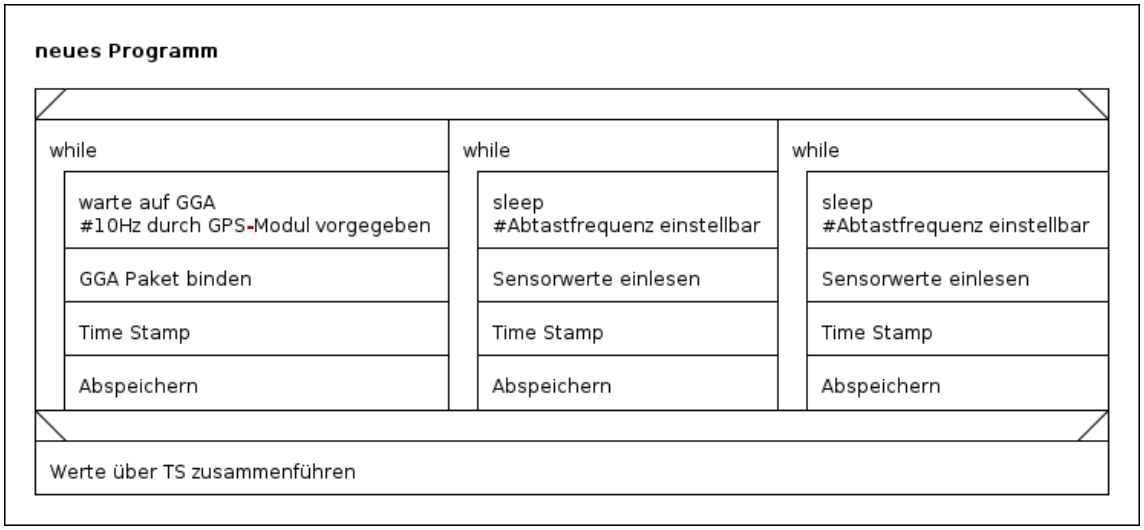


Abbildung 10: Multithreading Konzept

Wie in Abbildung 10 zu erkennen ist, basiert die Grundidee der Architektur auf dem Aufteilen der Sensorauslesungen auf einzelne Threads. Dabei soll jeder Thread eine eigene while-Schleife besitzen. Durch diesen Aufbau wird jeder Sensor in einer eigenen Schleife je nach gewünschter Frequenz ausgelesen. Die while-Schleife ist dabei ähnlich aufgebaut wie in Kapitel 4.2.1 mit dem Unterschied, dass die Sensorauslesung bei jeden Schleifen-Durchlauf durchgeführt wird. Um die so aufgenommenen Messwerte im Auswertungs-Programm zuordnen zu können, soll jeder Wert mit einem Zeitstempel markiert werden. Durch diese Architektur ist es möglich, neue Sensormodule durch das Hinzufügen weiterer Threads zu ergänzen.

4.2.3. Auswahl der Programm-Architektur

In diesem Kapitel wird die Auswahl der Programm-Architektur begründet. Um einen besseren Überblick über die beiden Architekturen zu erhalten, wird im Folgenden eine Tabelle aufgestellt, in der die Lösungskonzepte gegenübergestellt werden.

Tabelle 10: Vergleich zwischen möglichen Architekturen

Kriterium	**Singlethread**	**Multithread**
Modularität	Funktionen	Threads
unterschiedliche Auslesefrequenzen	Sensorauslesung abhängig vom Schleifendurchlauf	Sensorauslesung bei jedem Schleifendurchlauf
Zusammenführung der Messwerte	Zählindex	Zeitstempel
Überwachung durch Watchdog	Erzeugung des Lebenssignals innerhalb der Schleife	Überwachung der einzelnen Threads schwierig
Beeinflussung der Sensorauslesung	Sensorauslesung darf eine gewisse Zeit nicht überschreiten, sonst werden folgende Auslesungen zeitlich fehlerhaft beeinflusst	Sensorauslesung kann beliebig lange dauern, keine Beeinflussung vorhanden
CPU-Auslastung bei 100 Hz Odometrie-Pulsen	13 %	40 %
Erfahrung	wird seit 01.09.2015 im Feld eingesetzt	keine Erfahrung

Auf Grundlage der Tabelle 10 wurde die Singlethread-Architektur ausgewählt. Diese Entscheidung wird besonders durch die Kriterien ”Überwachung durch Watchdog” und ”CPU-Auslastung” getragen.

Da im Rahmen dieser Arbeit die Entwicklung einer selbstüberwachten Architektur durchgeführt werden soll, bietet sich eine Architektur an, bei der die Selbstüberwachung ohne großen Aufwand umgesetzt werden kann. Außerdem muss die maximale Hardware-Leistung beachtet werden. Bei der Multithread-Architektur führt ein Hinzufügen neuer Sensormodule und damit neuer Threads schneller zur kompletten Auslastung der von der Hardware zur Verfügung gestellten Leistung, als es bei der Singlethread-Architektur der Fall ist.

5. Implementierung

In diesem Kapitel geht es um die Implementierung der erstellten Konzepte. Zu Beginn wird kurz auf die verwendete Entwicklungsumgebung eingegangen. Anschließend folgt eine Ausführung zur Implementierung des Watchdogs. Im nächsten Schritt wird auf die Umsetzung des Logging- sowie Auswertungs-Programms näher eingegangen. Dann wird der *MISRA-C*-Test durchgeführt und eine Übersicht über die vom Gesamtprogramm erstellten Daten gegeben.

5.1. Entwicklungsumgebung

Um möglichst nahe an der Hardware zu arbeiten, wurde die Software direkt auf dem *Raspberry Pi* entwickelt. Da das verwendete Betriebssystem kein GUI besitzt, wurde das Betriebssystem *Raspbian* für die Entwicklung verwendet. *Raspbian* ist ein freies, auf Debian basierendes Betriebssystem, welches für die *Raspberry Pi*-Hardware optimiert wurde. Es besitzt ein GUI und eine Vielzahl von bereits vorinstallierter Software. [17]

Verwendet wurde die IDE *Code:Blocks*. Da der *Raspberry Pi 1* zu wenig Leistung besitzt, um *Code:Blocks* flüssig auszuführen, wurde der *Raspberry Pi 2* verwendet. Dieser besitzt vier Prozessoren, wodurch ein komfortables Arbeiten in *Code:Blocks* ermöglicht wird. Von den Schnittstellen ist der *Raspberry Pi 2* identisch zum *Raspberry Pi 1*, wodurch ein Wechsel ohne großen Aufwand durchgeführt werden kann.

Auch der *MISRA-C*-Test wurde unter der Verwendung der Software *Splint* (siehe Kapitel 5.5) auf dem *Raspberry Pi 2* durchgeführt.

5.2. Implementierung des Watchdogs

Die Entwicklung des externen Hardware-Watchdogs wurde im Rahmen der Studienarbeit "Weiterentwicklung des GPS-Datenloggers für Kraftfahrzeuge" von André Timmann (siehe Quelle [18]) durchgeführt. In der nachfolgenden Ausführung wird zuerst auf die in der Studienarbeit entwickelten Platine eingegangen. Des Weiteren wird die implementierte Kommunikation zwischen dem Positionsdatenlogger und dem Watchdog beschrieben. Zum Schluss wird auf die Integration der Watchdogfunktionalität in das Gesamtprogramm eingegangen.

5.2.1. *Micro-Power*

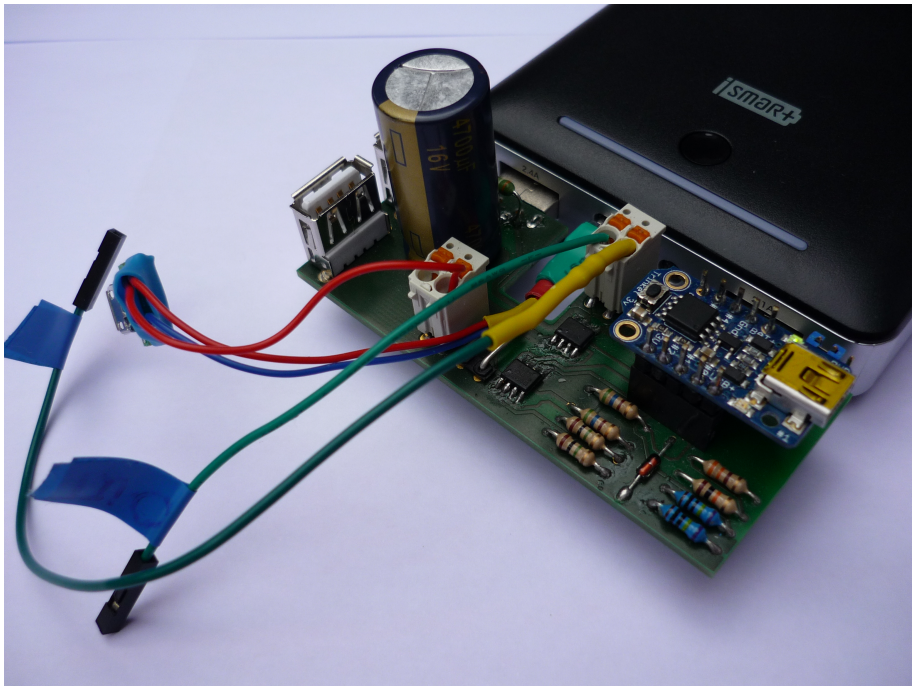


Abbildung 11: *Micro-Power*-Platine

Die *Micro-Power*-Platine beruht auf dem Board *Adafruit Trinket 3V*. Dieses beinhaltet einen mit 8 Mhz getakteten Micro-Controller der Firma *Atmel*. Der *Trinket* stellt fünf GPIO-Pins zur Verfügung. Um das auszuführende Programm auf den Trinket zu laden, wird ein USB-Anschluss verwendet. Dieser USB-Port unterstützt allerdings keine serielle Kommunikation, wodurch er als Kommunikationsschnittstelle zum *Raspberry Pi* nicht in Frage kommt. [18]

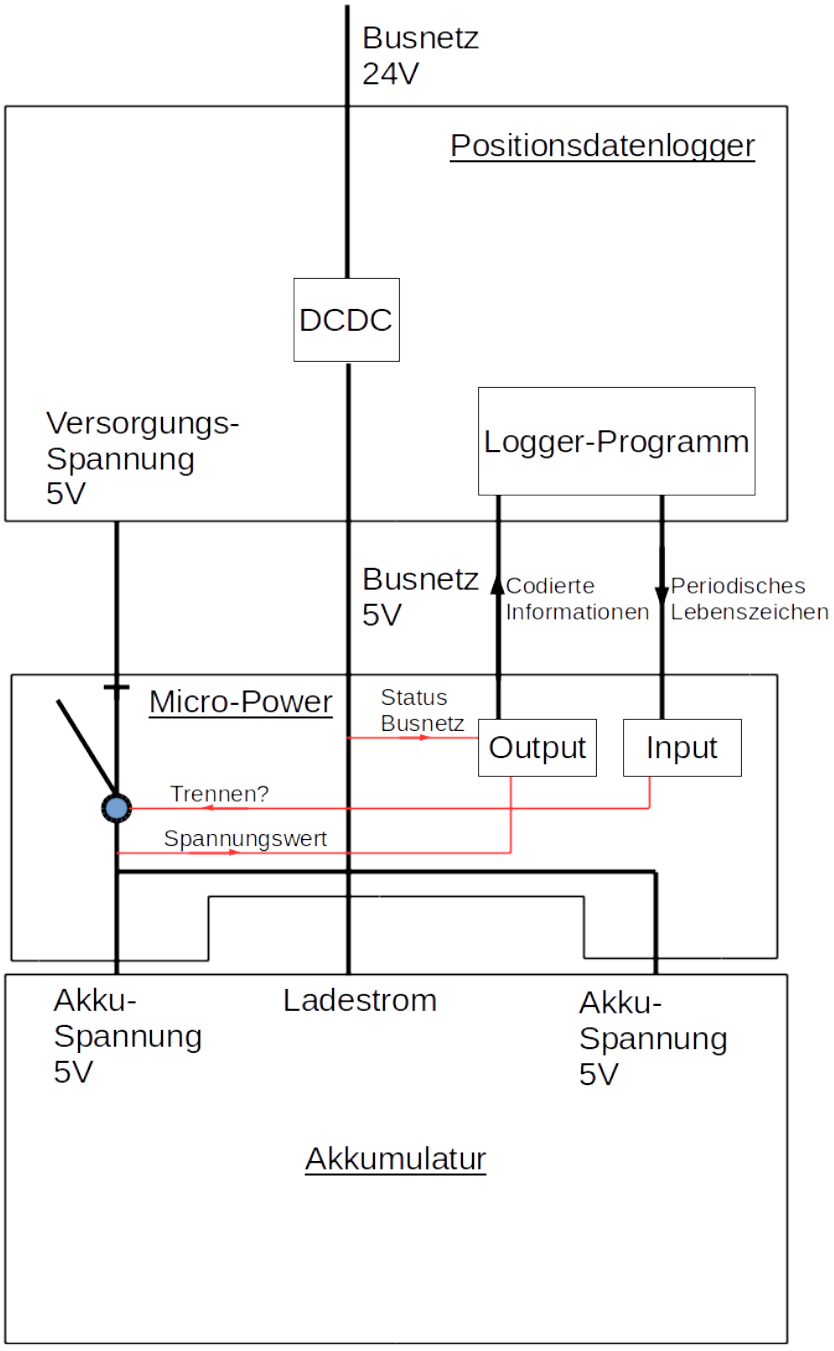


Abbildung 12: Blockdiagramm zur Integration der *Micro-Power*-Platine in das Gesamtsystem

Die Platine wurde so entwickelt, dass sie auf den Akkumulator des Positionsdatenloggers gesteckt werden kann. Als Ausgang stellt die Platine die am Akku anliegende Versorgungsspannung zur Verfügung. Der vom Busnetz kommende Akku-Ladestrom wird über einen Micro-USB-Port weiter an den Akku gegeben. Die *Micro-Power*-Platine ist in der Lage, den Ladestatus des Akkumulators, also den Zustand des Busnetzes, den Befehl zum Herunterfahren und die aktuell am Positionsdatenlogger anliegende Versorgungsspannung wiederzugeben (siehe Kapitel 5.2.2).

5.2.2. Kommunikation

Zuerst wird die Kommunikation vom *Raspberry Pi* zum *Micro-Power* erklärt. Danach wird näher auf das Ausgangssignal der *Micro-Power*-Platine eingegangen. Beide Signale wurden in Zusammenarbeit mit André Timmann festgelegt. Die Kommunikation wird über die GPIO-Pins realisiert, das heißt, es gibt ein Kabel für das periodische Lebenssignal und ein Kabel für die Informationen bezüglich der Spannungsversorgung beziehungsweise dem Befehl zum Herunterfahren. [18, Seite 23]

Kommunikation vom *Raspberry Pi* zum *Micro-Power*: Das in Kapitel 4.1 erstellte Konzept sieht vor, dass der Einplatinencomputer bei korrekter Funktion ein periodisches Signal an den Watchdog sendet.

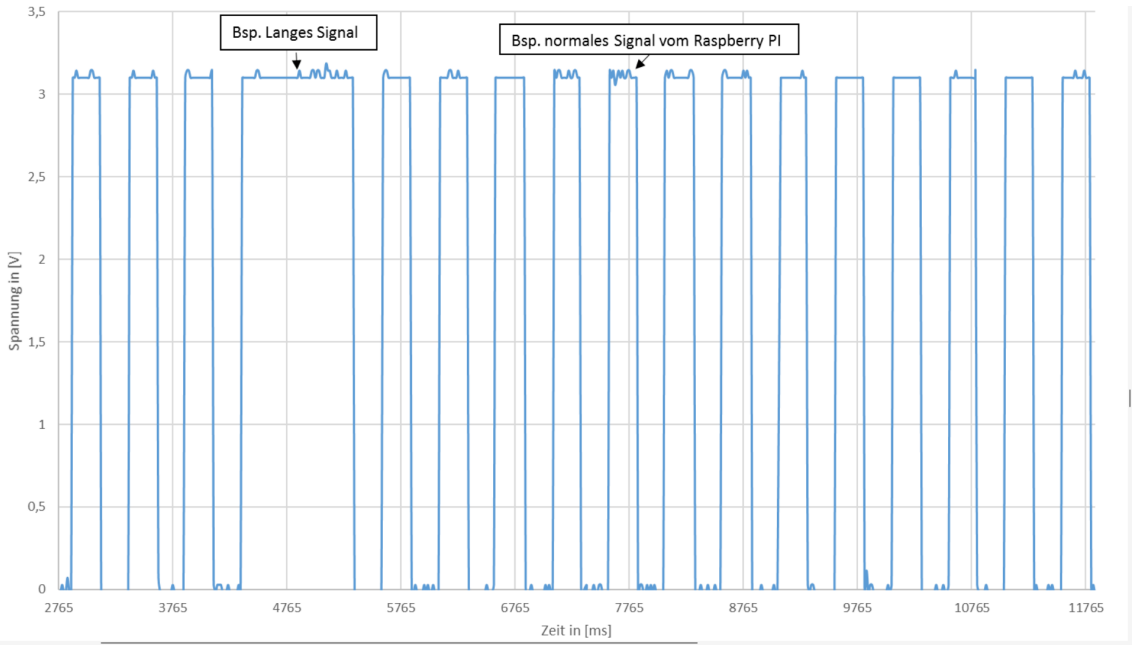


Abbildung 13: Ausgangssignal vom *Raspberry Pi*, Eingangssignal vom *Micro-Power* [18, Seite 34]

Wie in Abbildung 13 dargestellt, handelt es sich bei dem Ausgangssignal des *Raspberry Pis* um ein Rechtecksignal. Dieses Rechtecksignal besitzt eine Puls-/Pausenzeit von jeweils 250 ms. Wie dieses Signal vom Einplatinencomputer erzeugt wird, wird im Kapitel 5.2.3 näher beschrieben. Vom Micro-Power wird das ankommende Signal durch einen zeitlichen Referenzwertvergleich ausgewertet. Dieser Vergleich wird sowohl für den ”HIGH”- also auch für den ”LOW”-Zustand durchgeführt. Um die beiden Zustände mit unterschiedlichen Referenzwerten abzugleichen, lassen sich zwei verschiedene Werte im Watchdog-Programm festlegen. Wenn das Signal kürzer als der Referenzwert ist, wird die Software des Positionsdatenloggers wie geplant ausgeführt. Falls die Zeit des ”HIGH”- oder ”LOW”-Zustands größer als der entsprechende Referenzwert ist, also eine Wechsel des Signalpegels eine gewisse Zeit ausbleibt, trennt die Micro-Power-Platine die Versorgungsspannung des Datenloggers. Nach einer ebenfalls einstellbaren Zeit wird die Verbindung zum Akkumulator vom MicroPower wieder freigegeben. [18, Seite 34]

Zusammenfassend lässt sich sagen, dass durch diese Implementierung der Watchdog ein Ausbleiben des Pegelwechsels über eine gewisse Zeit erkennen kann und dadurch die fehlende Reaktion des Einplatinencomputers identifiziert sowie entsprechende Maßnahme einleitet.

Kommunikation vom *Micro-Power* zum *Raspberry Pi*: Um den Status des Busnetzes und den Wert der anliegenden Spannung an den *Raspberry Pi* zu senden, wird eine möglichst simple Codierung verwenden. Dabei wird die Information über die Pulsdauer, aber auch über die Pulsanzahl übertragen. Die folgenden Abbildungen zeigen die drei möglichen Zustände (Teil 1) und die Codierung der Akku-Spannung (Teil 2).

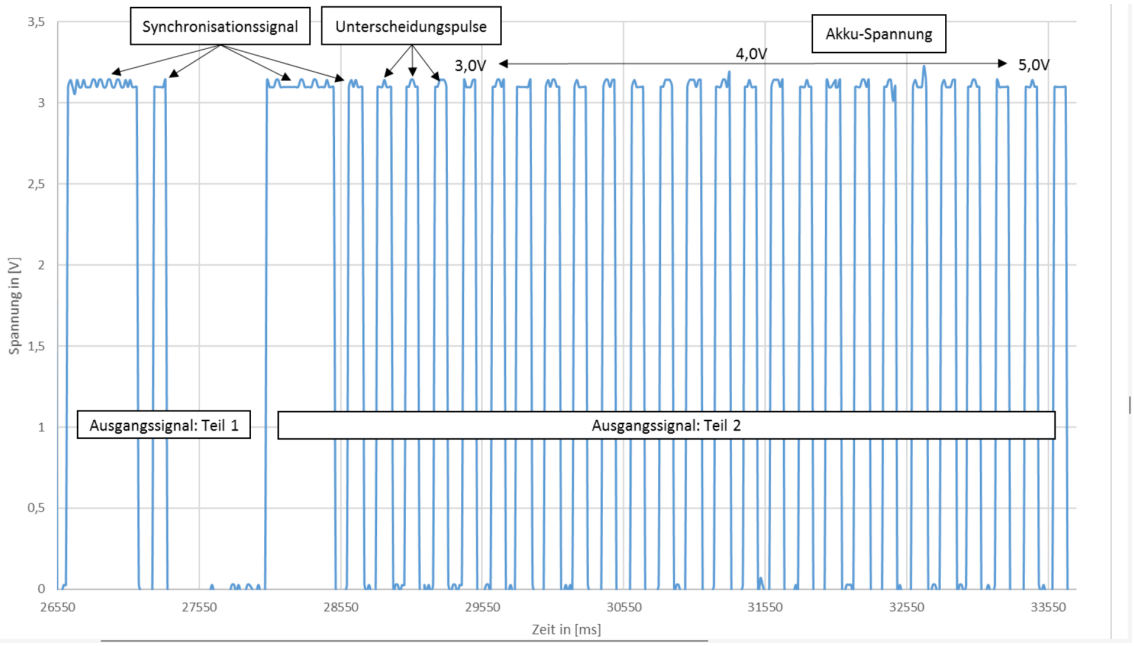


Abbildung 14: Ausgangssignal vom *Micro-Power* für den Status "Busnetz aus" [18, Seite 35]

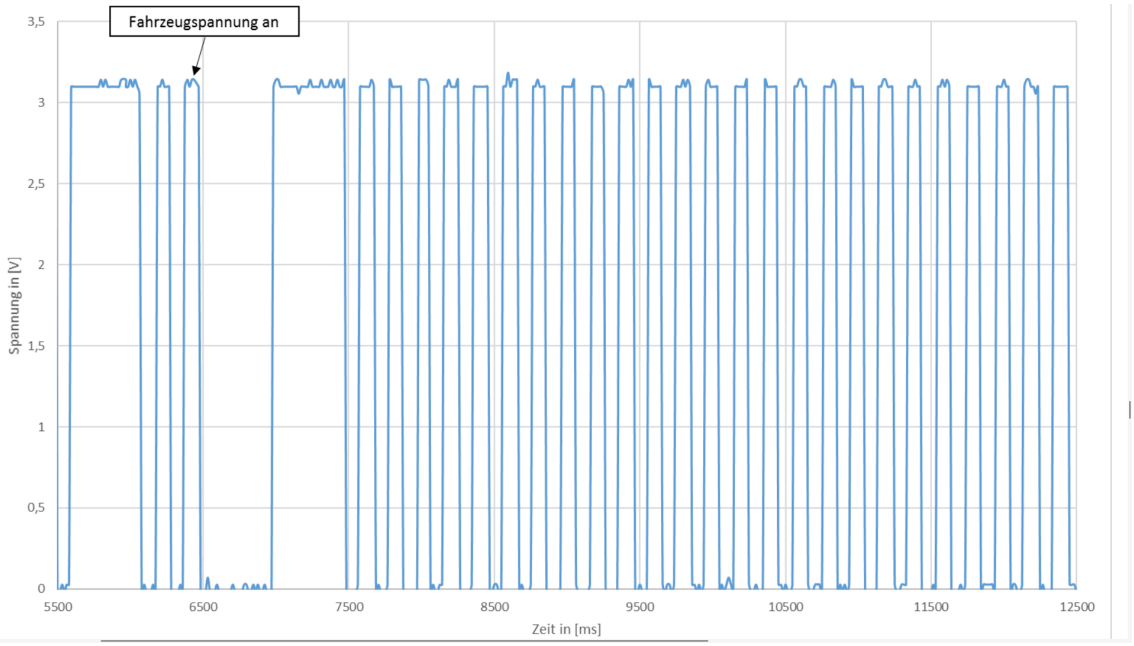


Abbildung 15: Ausgangssignal vom *Micro-Power* für den Status "Busnetz an"[18, Seite 35]

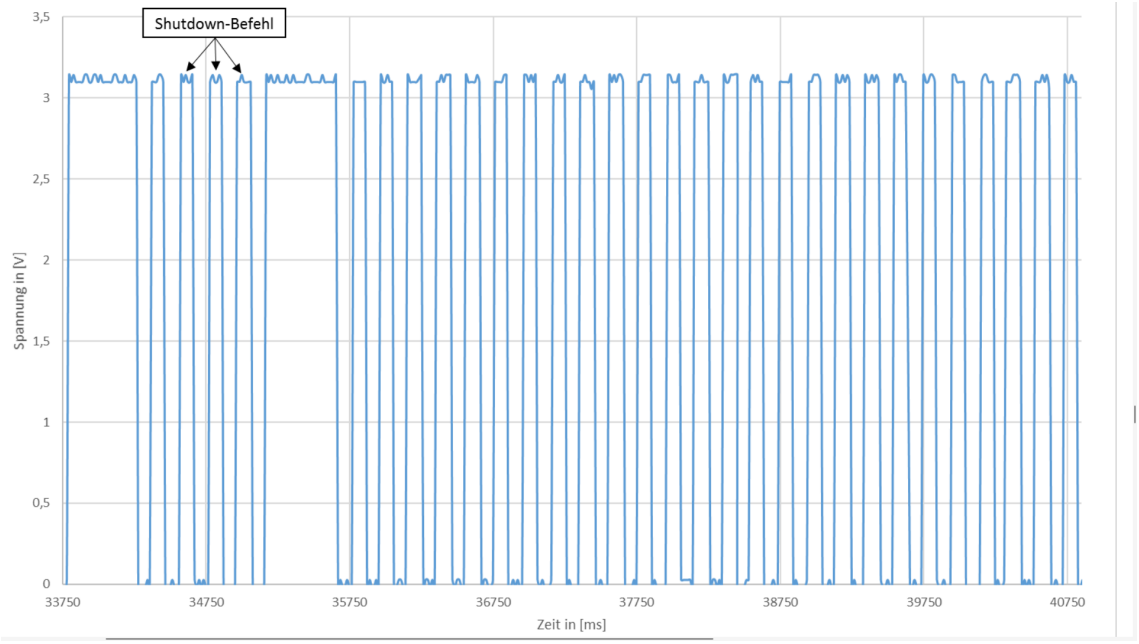


Abbildung 16: Ausgangssignal vom *Micro-Power* für den Status ”System herunterfahren” [18, Seite 36]

Das Gesamtsignal besteht aus zwei Teilen, die durch eine Synchronisationssequenz voneinander getrennt sind. Dabei handelt es sich bei dem ersten Teil um den Status des Busnetzes oder dem Befehl zum Herunterfahren und bei dem zweiten Teil um den aktuell anliegenden Spannungswert.

Die Synchronisationssequenz setzt sich aus zwei Pulsen zusammen, wobei der erste Puls um ein Vielfaches länger ist als der darauf folgende. Durch diesen Aufbau lässt sich das Synchronisationssignal durch die zeitliche Messung zwischen zwei Pulsen identifizieren. Die eigentliche Information wird durch die auftretende Anzahl an Pulsen übertragen.

Tabelle 11: Übersicht über Codierung

Anzahl an Pulsen	**zu übertragene Information**
0	Busnetz aus
1	Busnetz an
2	×
3	Befehl zum Herunterfahren
4	3,0 V
5	3,1 V
6	3,2 V
7 ... 23	3,3 V ... 4,9 V
24	5,0 V

Wenn sich die Pulsanzahl zwischen null und drei Pulsen befindet, handelt es sich um die Information zum Busnetz beziehungsweise den Befehl zum Herunterfahren. Sobald die Pulsanzahl größer als drei ist, handelt es sich bei der folgenden Information um die Akku-Spannung. Durch diese Codierung wurden die beiden Teilinformationen eindeutig voneinander getrennt.

5.2.3. Integration in das Gesamtprogramm

Zu Beginn wird auf das zeitlich periodische Signal eingegangen, welches vom *Raspberry Pi* zum *Micro-Power* übertragen wird. Danach wird erklärt, wie das Signal, welches die Informationen zum Versorgungsstatus und den Befehl zum Herunterfahren übertragen soll, vom *Raspberry Pi* ausgewertet wird.

Erzeugung des periodischen Signals: Zunächst muss geklärt werden, in welchen Teilen des Gesamtprogrammablaufs der periodische Pegelwechsel (Lebenssignal) benötigt wird. Nach kurzer Überlegung stellt sich heraus, dass der Pegelwechsel des Signals zum Watchdog immer dann periodisch durchgeführt werden muss, wenn ein Abschnitt des Programms durchlaufen wird, der zeitlich nicht konstant und damit

theoretisch eine hohe Ausführungszeit in Anspruch nehmen kann. Es wurden folgende Programmabschnitte festgestellt:

- while-Schleife zur Sensorabfrage im Logging-Programm.
- for-Schleife zur Auslesung der Messwerte im Auswertungs-Programm.
- zip-Vorgang zum Zusammenfügen der Messwerte im Auswertungs-Programm.

Um dem Watchdog mitzuteilen, dass die oben aufgelisteten Programmabschnitte erfolgreich ausgeführt werden, wird ein periodischer Pegelwechsel des GPIO-Pins durchgeführt. Der folgende Quellcode stellt eine Implementierung des Befehls "Sende Lebenszeichen an Watchdog;" aus Listing 4 dar. Die folgende Logik zur Durchführung des Pegelwechsels wurde in allen drei Programmabschnitten verwendet.

```
1 if(watchdog_out == 1)
2 {
3     watchdog_out = 0;
4 }
5 else if(watchdog_out == 0)
6 {
7     watchdog_out = 1;
8 }
9 digitalWrite(WD_OUTPUT, watchdog_out);
```

Listing 5: Pegelwechsel für Kommunikation zum Watchdog

Es wird eine Variable verwendet, um den aktuellen Zustand des Signals zu speichern. Diese Variable wird dazu gebraucht, um festzustellen, in welche Richtung der Pegelwechsel durchgeführt werden muss. Der eigentliche Prozess des Zustandswechsels wird mit Hilfe der von *wiringPi* (siehe Kapitel 2.4.2) zur Verfügung gestellten Funktion "digitalWrite()" durchgeführt. In der while- und for-Schleife wird der Pulswechsel so eingebunden, dass dieser nur alle 4 Hz (250 ms) durchgeführt wird.

Um während des im Auswertungs-Programm durchgeführten zip-Vorgangs weiterhin den Pegelwechsel durchzuführen, wird jede Rohdatei nacheinander zu dem zip-Archiv hinzugefügt. Dadurch lässt sich der Pegelwechsel zwischen den einzelnen Archivierungen durchführen. Durch diese Umsetzung darf ein Hinzufügevorgang einer Rohdatei die Zeit, die beim Ausbleiben des Pegelwechsels bis zur Trennung gewartet wird, nicht überschreiten.

Dekodierung des vom *Micro-Power* erzeugten Signals: Wie in Kapitel 5.2.2 beschrieben, beinhaltet das vom *Mircro-Power* zum *Raspberry Pi* übertragene Signal die Informationen in Form von Pulsdauer sowie der Pulsanzahl. Um die codierte

Information auszuwerten, wird eine Interrupt Service Routine innerhalb des Logging-Programms verwendet. Diese ISR wird mit Hilfe der Bibliothek *wiringPi* initialisiert. Ausgelöst wird die ISR bei einer steigenden Flanke.

```
1  static void watchdog_ISR(void)
2  {
3      struct timespec current_time = start_wd;
4      struct timespec old_time = start_wd;
5      double pulse_to_pulse_time = -1;
6
7      watchdog_pulse++;
8      clock_gettime(CLOCK_MONOTONIC, &current_time);
9      start_wd = current_time;
10     pulse_to_pulse_time = calc_time(old_time, current_time);
11     if(pulse_to_pulse_time >= ORCSynclength + ORCPulselength
12        - TRINKET_TIME_ERROR)
13     {
14         watchdog_pulse = watchdog_pulse - 2;        //sync pulse abziehen.
15         if(watchdog_pulse > 3)
16         {
17             global_akku_spannung = watchdog_pulse - 3;
18         }
19         else if((watchdog_pulse <= 3) && (watchdog_pulse >= 0))
20         {
21             global_netz_shutdown = watchdog_pulse;
22         }
23         else
24         {
25             save_error("Negative watchdog_pulse!\n");
26         }
27         watchdog_pulse = 0;
28     }
29 }
```

Listing 6: ISR für Decodierung der vom *Micro-Power* übertragenen Information

Um die Synchronisationssequenz zu erkennen, wird eine Zeitmessung zwischen den steigenden Flanken durchgeführt. Neben der Zeitmessung wird außerdem ein Pulszähler bei jeder ISR-Ausführung inkrementiert. Wenn die durchgeführte Zeitmessung auf die Synchronisationssequenz schließen lässt, wird der bis dahin inkrementierte Zähler je nach Größe in einer globalen Varibalen gespeichert und danach auf null gesetzt.

Zu Beginn der ISR wird die globale Zeitvariable im Stack der ISR gespeichert, um einen temporären Zugang zur Variablen zu erhalten (Zeile 3 - 4). Danach wird

der Pulszähler inkrementiert (Zeile 7) und die aktuelle Zeit wird sowohl temporär (Zeile 8), als auch in der globalen Zeitvariablen (Zeile 9) gespeichert. Dadurch wird der Zeitwert bis zur nächsten ISR-Ausführung gespeichert, was eine Berechnung der Zeitdifferenz zwischen zwei Pulsen ermöglicht. Mit Hilfe der temporären Zeiten wird im nächsten Schritt genau diese Zeitdifferenz berechnet (Zeile 10). Darauf folgt die Logik zur Erkennung der Synchronisationspulse. Dabei wird überprüft, ob die Zeitdifferenz größer als ein festgelegter Referenzwert ist (Zeile 11 - 12). Wenn dies zutrifft, handelt es sich um die Synchronisationssequenz. Nun werden die Synchronisationspulse von dem Pulszähler abgezogen (Zeile 14) und die Auswertung der Pulsanzahl zwischen zwei Synchronisationssequenzen beginnt. Dabei wird, wie in der Tabelle 11 dargestellt, vorgegangen. Wenn der Pulszähler zwischen null und drei liegt, wird der Wert des Pulszählers in der globalen Variablen "global_netz_shutdown" gespeichert (Zeile 19 - 22). Wenn der Pulszähler aber größer als drei ist, wird dieser um drei verringert und der globalen Variablen "global_akku_spannung" übergeben (Zeile 15 - 18). Durch die Verringerung um drei (Zeile 17) startet die Codierung der Akku-Spannung bei eins und nicht mehr wie in der Tabelle 11 beschrieben bei vier. Die Ursache für diese Umrechnung ist die dabei entstehende anschaulichere Dekodierung, auf die im Folgenden eingegangen wird.

Um die von der ISR zugewiesenen globalen Variablen auszuwerten, wird eine Funktion innerhalb der Sensorauslese-Schleife verwendet.

```
1  int decode_wd_input(char *watchdog_info, float *akku_v, int *shutdown_counter)
2  {
3      int akku_spannung = global_akku_spannung;
4      int netz_shutdown = global_netz_shutdown;
5
6      if(akku_spannung > 0)
7      {
8          *akku_v = ((float) akku_spannung * 0.1f) + 2.9f;
9      }
10     if(netz_shutdown == 0)
11     {
12         strcpy(watchdog_info, "Busnetz aus");
13     }
14     else if(netz_shutdown == 1)
15     {
16         strcpy(watchdog_info, "Busnetz an");
17     }
18     else if(netz_shutdown == 2)
19     {
20         save_error("ungültige Anzahl an WD_Info_Pulsen!\n");
21     }
```

```
22      else if(netz_shutdown == 3)
23      {
24          (*shutdown_counter)++;
25          if(*shutdown_counter > REPEAT_SHUTDOWN_SIGNAL)
26          {
27           strcpy(watchdog_info, "Befehl von Watchdog: shutdown durchführen");
28           //Programm sauber beeneden und Rpi runterfahren.
29          }
30      }
31
32      return EXIT_SUCCESS;
33  }
```

Listing 7: Decodierung der vom *Micro-Power* übertragenen Information

Auch hier werden zuerst die globalen Variablen in den Stack geladen (Zeile 3 - 4). Dann wird durch die Überprüfung der Variablen ”akku_spannung” festgestellt, ob die ISR bereits eine Sequenz mit Information zur Akku-Spannung ausgewertet hat (Zeile 6). Wenn dies zutrifft, wird die globale Variable dazu verwendet, die aktuell vom Akkumulator bereitgestellte Spannung zu berechnen (Zeile 8). Anschließend wird die Variable ”netz_shutdown” nach den in der Tabelle 11 aufgeführten Werten überprüft. Je nach Wert wird die festgelegte Information in einer Zeichenkette gespeichert.

Der Befehl zum Herunterfahren des Systems wird nur dann vom *Micro-Power* an den *Raspberry Pi* gesendet, wenn das periodische Signal ausbleibt, was sehr wahrscheinlich auf das Stoppen des Programmablaufs zurückzuführen ist. Aus diesem Grund wird der Befehl zum Herunterfahren in den meisten Fällen vom *Raspberry Pi* nicht ausgewertet, da die Software nicht korrekt ausgeführt wird. Falls aber der Zustand eintritt, bei dem das Logging-Programm und damit die Signalauswertung korrekt ausgeführt wird, aber das periodische Lebenssignal trotzdem ausbleibt, fährt der Einplatinencomputer runter und die Spannungstrennung erfolgt im ausgeschalteten Zustand. Um das Ausführen eines fehlerhaft erkannten Herunterfahr-Befehls auszuschließen, muss das Signal mehrfach hintereinander vom *Raspberry Pi* erkannt werden, bevor der Befehl ausgeführt wird (Zeile 25). Durch den Befehl zum Herunterfahren erhält der *Raspberry Pi* die Chance, beim Ausbleiben des Lebenssignals ein eigenständiges Herunterfahren einzuleiten, bevor der Watchdog einen Hardware-Reset durchführt.

5.2.4. Zeitlicher Ablauf der Überwachung

Um die Zeiten des Watchdogs einzustellen, werden am Anfang des *Trinket*-Programms, Macros definiert. Im Folgenden ist der Programmausschnitt (siehe Quelle [18]) mit

den verwendeten Zeiten dargestellt.

```
1  //Parameter;
2  #define DelayTime            120000 //[ms] = 2 [min]
3  //Delay time for the separation of the circuit (Time befor shutdown/turn on)
4  #define WatchdogTime         900000 //[ms] = 15 [min]
5  //Time before the Watchdog separates the circuit in case of no changing signal,
6  //must be longer than the full shutdown cycle
7  #define separationSignalTime 900000 //[ms] = 15 [min]
8  //must be longer than cycle Length, Length of the signal for separating
9  //the circuit
10 #define separationBlockDelay 600000 //[ms] = 10 [min]
11 //Blocking the next separation for a defenite delay.
12 //Calculate by hand Value=separationBlockDelay+(2*DelayTime)
```

Listing 8: Parameter des Watchdogs

Mögliche Gefahren sind es, eine Schwingung des Systems zwischen den An- und Aus-Zustand oder aber einen dauerhafte Trennung zu erzeugen. Um dem entgegenzuwirken, soll das Macro ”separationBlockDelay” nach einer Trennung ein bestimmtes Zeitfenster zur Verfügung stellen, in dem eine erneute Trennung ausgeschlossen ist. Die effektive Zeit des Macros ”separationBlockDelay” beginnt, sobald dieses größer ist als zweimal die DelayTime plus die Zeit, die der *Raspberry Pi* zum Hochfahren braucht. Das heißt, bei den jetzt verwendeten Parametern beträgt die effektive Blockierungszeit fünf Minuten.

Tabelle 12: Zeitlicher Ablauf der Trennung

Aktion	**Signal vom *Raspberry Pi* zum Watchdog**	**Signal vom Watchdog zum *Raspberry Pi***	**Ausführungszeit**	
Normalbetrieb	Pegelwechsel	Busnetzstatus, Akku-Spannung	×	
"System reagiert nicht mehr"	konstant 1 / 0	Busnetzstatus, Akku-Spannung	SeparationSignalTime (bei konst. 1) / WatchdogTime (bei konst. 0)	
"System reagiert nicht mehr"	konstant 1 / 0	Befehl zum Herunterfahren	Delay-	
Sytem fährt nicht runter	×	×	Time	Separation-
Trennung	×	×	×	Block-
System spannungsfrei	×	Busnetzstatus, Akku-Spannung	DelayTime	Delay
System fährt hoch	×	Busnetzstatus, Akku-Spannung	1 Minute	
Normalbetrieb Trennung nicht möglich	Pegelwechsel	Busnetzstatus, Akku-Spannung	Differenz	
Normalbetrieb	Pegelwechsel	Busnetzstatus, Akku-Spannung	×	

5.3. Implementierung des Logging-Programms

In diesem Kapitel wird die Implementierung des Logging-Programms beschrieben. Nachdem ein Überblick über das Logging-Programm gegeben wurde, wird erklärt, wie einzelne Hauptfunktionen umgesetzt wurden.

5.3.1. Überblick

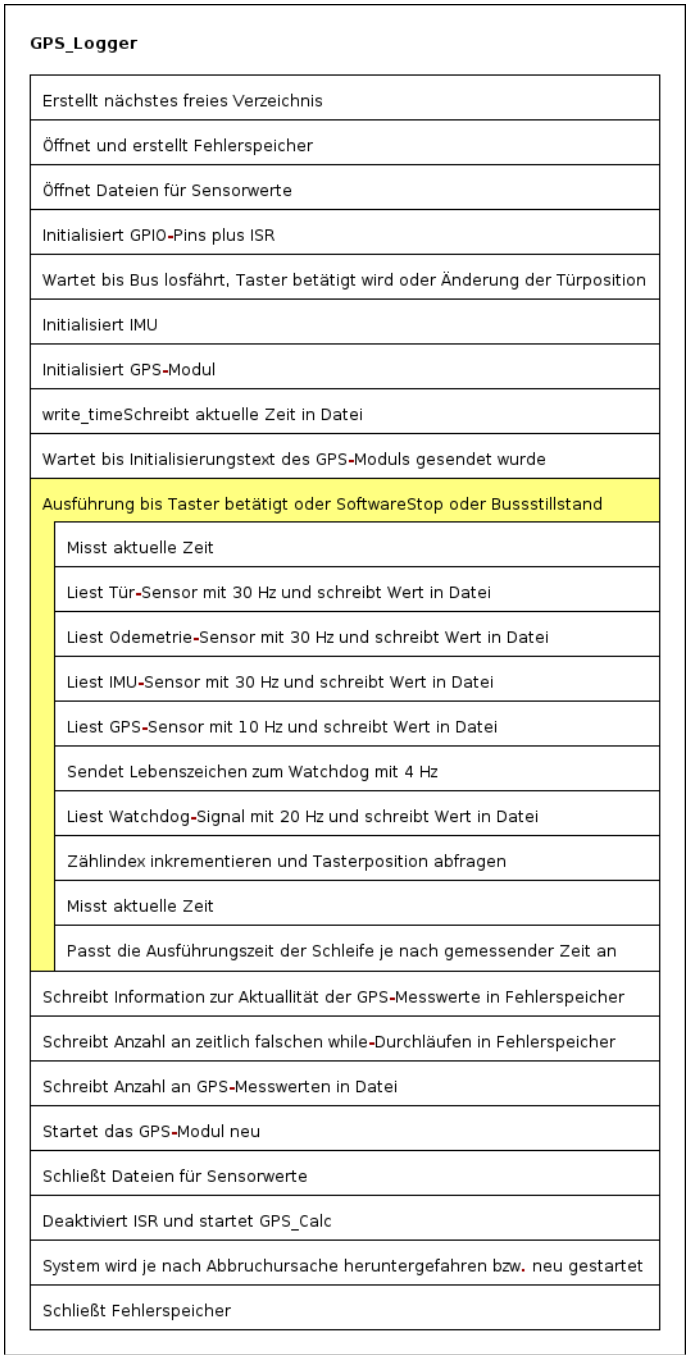


Abbildung 17: Struktogramm des Logging-Programms

Der Überblick zeigt, wie stark das Programm auf dem alten Logging-Programm (siehe Kapitel 2.4.2) aufbaut. Die Grundfunktion des Programms ist es, mit unterschiedlichen Frequenzen mehrere Sensoren abzufragen und die so erhaltenen Werte in Dateien zu speichern. Die Sensorauslese-Schleife ist gelb markiert und entspricht dem in Kapitel 4.2.1 beschriebenen Entwurf. Im Folgenden werden die wichtigsten neu implementierten Funktionen näher erklärt.

5.3.2. Modularität durch Sensorauslese-Funktion

Die Modularität wurde durch das Erstellen einer Sensorauslese-Funktion, in die die Auslese-Logik eingefügt wird, implementiert. Innerhalb dieser Funktion werden folgende Ausführungen durchgeführt:

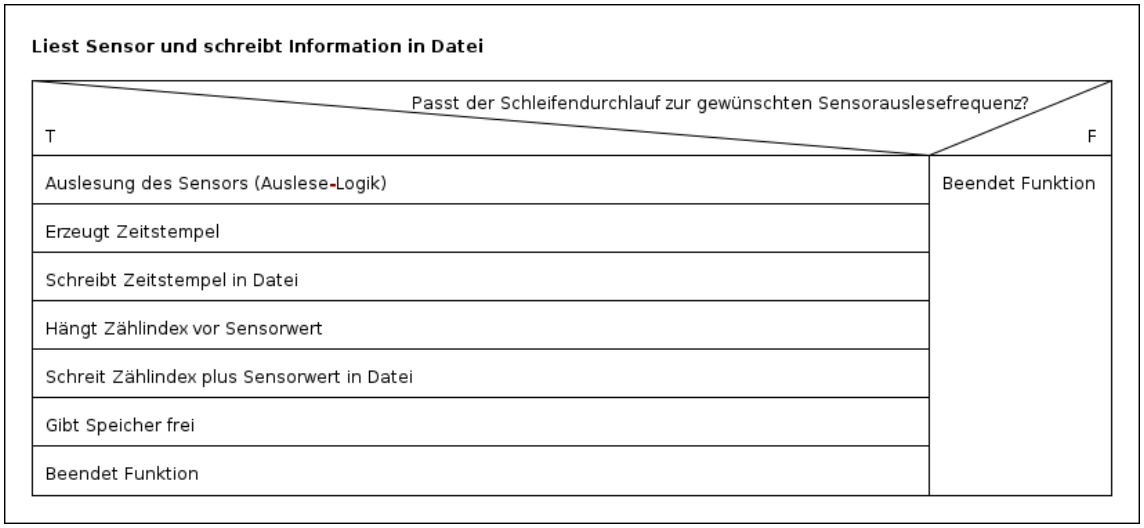


Abbildung 18: Struktogramm der Sensorauslese-Funktion

Wenn der aktuelle Schleifendurchlauf zur gewünschten Sensorauslesefrequenz passt, wird die jeweilige Auslese-Logik ausgeführt. Die Trennung der Sensorwerte wird beim GPS-Modul durch ein ”newline”- gefolgt von einem ’$’-Zeichen realisiert. Um ein einheitliches Format zu gewährleisten, wird am Ende der Auslese-Logik der Sensorwert um genau diese Zeichenfolge erweitert. Dadurch erhalten alle Sensorwerte jedes Sensormoduls die gleiche Zeichenfolge zur Wertetrennung.

Danach wird der aktuelle Zeitstempel erzeugt und in die Datei geschrieben. Der Zeitstempel wird in dem Auswertungs-Programm (siehe Kapitel 5.4) zur Überprüfung der Zeitdifferenz zwischen zwei Sensorwerten genutzt. Um die Werte der verschiedenen Sensoren zuordnen zu können, wird jeder aufgenommene Sensorwert mit

dem Zählindex des schnellsten Sensors versehen. Dadurch lässt sich im Auswertungs-Programm für jeden GPS-Messwert die zeitlich zugehörigen Werte der anderen Sensoren identifizieren. Zum Schluss wird der verwendete Heap-Speicher freigegeben und die Funktion wird beendet.

Entstandenes Format:
Zeitstempel:Zählindex_SensorwertTrennungszeichen
Beispiel:
2016.05.28.19.49.44.084:99_Sensorwert\n$

5.3.3. Fehlerspeicher

Wie in der Abbildung 17 zu erkennen ist, wird am Anfang des Programms eine Textdatei erstellt, die als Fehlerspeicher fungiert. Da diese Textdatei am Ende des Programms geschlossen wird, besteht während des gesamten Programmablaufs die Möglichkeit, Meldungen von auftretenden Fehlern zu speichern. Beim Auftreten eines Fehlers wird zunächst die Ursache im Fehlerspeicher eingetragen. Danach wird die Information eines Fehlerauftritts über den Rückgabewert der jeweiligen Funktion bis zur main-Funktion hochgereicht. Jede dabei durchlaufene Funktion schreibt ihren Namen in den Fehlerspeicher. Durch dieses Vorgehen lassen sich Fehler eindeutig lokalisieren.

```
1  No such file or directory->fread()->read_points()
```

Listing 9: Beispiel einer Fehlermeldung

Der Fehlerspeicher dient außerdem zur Speicherung von wichtigen Informationen, die während des Programmablaufs erzeugt werden. Hierzu zählen:

- Übrig gebliebener Venus-Buffer-Inhalt, um zu sehen, ob mit der richtigen Geschwindigkeit ausgelesen wurde.
- Anzahl an zeitlich falschen while-Durchläufen, um den zeitlichen Einfluss bestimmter Parameter zu testen.
- Wie wurde das Programm beendet?

5.3.4. Auslese-Logik für den GPS-Sensor

Da der GPS-Sensor den wichtigsten und am schwierigsten auszulesenden Sensor im Gesamtsystem darstellt, wird in diesem Kapitel nur auf die Auslese-Logik des GPS-Sensors eingegangen. Wie in Kapitel 2.3.4 beschrieben, wird die aktuelle GGA- und RMC-Zeichenkette alle 100 ms in einen Buffer geschrieben. Bei einer Auslesung des

GPS-Sensors muss zuerst festgestellt werden, ob aktuelle Werte im Buffer vorliegen. Um die Zeit zwischen dieser Anfrage ("poll"-Methode) und dem eigentlichen Auslesen des Sensors möglichst gering zu halten, wird die "poll"-Anfrage und der darauf folgende "pull"-Vorgang nur alle 100 ms (10 Hz) durchgeführt. Dadurch ist es möglich, die Zeitdifferenz zwischen der Anfrage und dem eigentlichem "pull"-Vorgang minimal zu halten.

Um die Auslesung robuster zu gestalten und auch fehlerhafte Zeichenketten mit aufzuzeichnen, wird eine Zeichenkette durch das wiederholte Auftreten des '$'-Zeichens erkannt. Da innerhalb einer Auslesung ein GPS-Messpunkt, also die GGA- sowie die RMC-Zeichenkette, aufgezeichnet werden soll, wird zweimal solange vom Buffer eingelesen, bis das '$'-Zeichen erkannt wird. Bei dieser Methode wird für die Erkennung der Zeichenkette das '$'-Zeichen der jeweils nächsten Zeichenkette verwendet.

```
1 GPGGA,194739.673,5314.4514,N,01024.7830,E,1,08,1.1,16.4,M,43.0,M,,0000*6F
2 $GPRMC,194739.673,A,5314.4514,N,01024.7830,E,027.2,196.9,280516,,,A*69
3 $nächster Messpunkt
```

Listing 10: Im Buffer vorliegende GPS-Zeichenketten

Nach der Aufzeichnung der beiden Zeichenketten werden diese separat voneinander behandelt. Die Zeichenketten erhalten einen gemeinsamen Zählindex und einen gemeinsamen Zeitstempel. Anhand des Zählindexes lässt sich die Zusammengehörigkeit der beiden Zeichenketten feststellen und überprüfen. Um eine Aussage über die zeitliche Qualität der Messwerte machen zu können, wird der Zeitstempel im Auswertungs-Programm verwendet.

```
1 $2016.05.28.19.49.44.084:99_GPGGA,194739.673,5314.4514,N,01024.7830,E, ...
2 $2016.05.28.19.49.44.084:99_GPRMC,194739.673,A,5314.4514,N,01024.7830,E,..
```

Listing 11: GPS-Zeichenketten aus der vom Logging-Programm erstellten Datei

5.4. Implementierung des Auswertungs-Programms

Im Folgenden wird die Implementierung des Auswertungs-Programms beschrieben. Zuerst wird ein Überblick über das Gesamtprogramm erstellt. Danach wird erklärt, wie einzelne Hauptfunktionen umgesetzt wurden.

5.4.1. Überblick

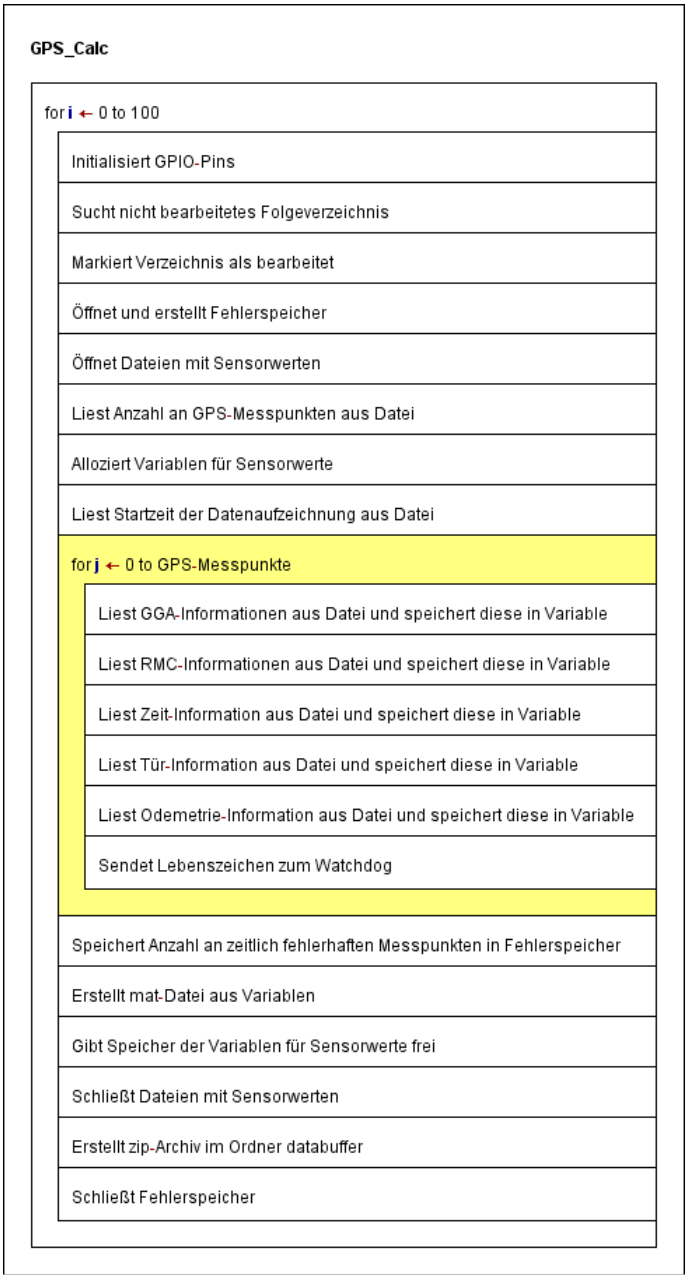


Abbildung 19: Struktogramm des GPS-Calc-Programms

Auch das Auswertungs-Programm baut auf der Kernarchitektur des alten Programms (siehe Kapitel 2.4.3) auf. Die Grundfunktion des Programms ist es, die Rohdaten, die während des Logging-Programms erstellt wurden, aufzubereiten und danach in einer mat-Datei zusammenzuführen. Das Programm muss also in der Lage sein, Werte aus Dateien zu lesen und diese in Variablen zu speichern. Die Variablen werden dann für die Erstellung der mat-Datei verwendet. Die for-Schleife, in der die einzelnen Messwerte ausgelesen werden, ist in gelb dargestellt. Im Folgenden werden die wichtigsten neu implementierten Funktionen näher erklärt.

5.4.2. Modularität durch Dateiauslese-Funktion

Da das Logging-Programm ein einfaches Erweitern neuer Sensormodule zulässt, muss auch das Auswertungs-Programm eine Methode bieten, die Werte des neuen Sensors der mat-Datei hinzuzufügen. Hierzu wurde wieder eine Funktion erstellt, die den Rahmen für variable Auslese-Logiken bereitstellt. Diese Dateiauslese-Funktion wird im weiteren Verlauf vorgestellt.

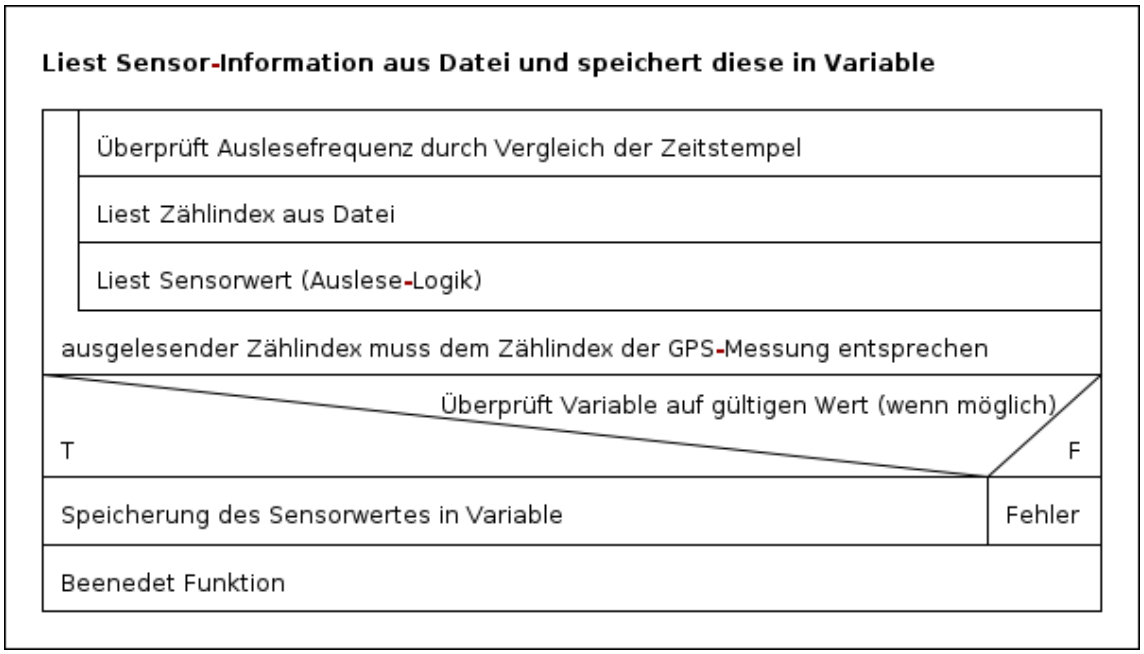


Abbildung 20: Struktogramm der Dateiauslese-Funktion

Anfangs wird der Zeitstempel eingelesen und mit dem vorherigen verglichen. Dadurch lässt sich eine Aussage über die Einhaltung der Auslesefrequenz machen (siehe Kapitel 6.1). Anschließend wird der dem Zeitstempel folgende Zählindex aus der

Datei eingelesen. Die darauf folgende Auslese-Logik wird je nach Sensor angepasst. Da im Logging-Programm ein einheitliches Format angewandt wurde, lässt sich das ”newline”- gefolgt von dem ’$’-Zeichen einheitlich zur Trennung der Sensorwerte verwenden. Die do-while-Schleife wird nur dann verlassen, wenn der ausgelesene Zählindex mit dem des GPS-Messwertes zusammenpasst. Durch dieses Vorgehen werden nur die Werte des Sensors in Variablen gespeichert, die in dem gleichen Schleifendurchlauf aufgezeichnet wurden wie der GPS-Messwert. Wenn ein Sensor also mit einer höheren Frequenz als 10 Hz ausgelesen wird, werden alle Sensorwerte ignoriert, die nicht zum GPS-Messwert passen. Dadurch lässt sich sicherstellen, dass alle in der mat-Datei enthaltenen Werte so gefiltert wurden, dass sie eine Frequenz von 10 Hz besitzen.

Tabelle 13: Beispiel einer Wertezusammenführung mit Hilfe des Zählindexes

Neuer Sensorwert mit Zählindex	**GPS-Sensorwert mit Zählindex**	**Aktion**
0_Wert		ignorieren
1_Wert		ignorieren
2_Wert		ignorieren
3_Wert	3_Wert	verwenden
4_Wert		ignorieren
5_Wert		ignorieren
6_Wert		ignorieren
7_Wert	7_Wert	verwenden

Auslesefrequenz neuer Sensor: 40 Hz
Auslesefrequenz GPS-Sensor: 10 Hz

Sobald der Vergleich der Zählindexe übereinstimmt, wird die do-while-Schleife verlassen und der Sensorwert wird in der dafür vorgesehenen Variable gespeichert. Vorher kann noch eine Überprüfung des Sensorwertes auf ungültige Zustände durchgeführt werden.

5.4.3. Fehlerspeicher

Der Fehlerspeicher ist genauso aufgebaut wie im Logging-Programm (siehe Kapitel 5.3.3). Um die auftretenden Fehler der beiden Programme voneinander zu trennen, wird eine zweite Textdatei erstellt. In diese Textdatei werden die Fehler eingetragen, die während der Ausführung des Auswertungs-Programms auftreten.

Als zusätzliche Information wird noch Folgendes mit aufgezeichnet:

- Anzahl an GPS-Messpunkten.
- Anzahl zeitlich falscher Messwerte, um die Auslesefrequenz der einzelnen Sensoren zu überprüfen (Zeitstempelvergleich).

5.4.4. Auslese-Logik für die GPS-Messwerte

Die GGA- und die RMC-Zeichenkette werden separat voneinander aus der gleichen Datei eingelesen. Da in beiden Fällen dieselbe Logik verwendet wird, wird in diesem Kapitel nur auf die Auslesung der GGA-Zeichenkette eingegangen.

Wie in der Abbildung 20 beschrieben, werden zunächst der Zeitstempel und der Zählindex eingelesen. Anschließend müssen die benötigten Informationen aus der GGA-Zeichenkette gefiltert werden. Um Zugriff auf jeden in der GGA-Zeichenkette befindlichen Wert zu erhalten, wird das ’,’-Zeichen zur Orientierung verwendet. Dabei wird die Zeichenkette solange eingelesen, bis ein Komma erkannt wird. Beim Auftreten eines Kommas wird über eine switch-Anweisung entschieden, ob der vorliegende Wert in einer Variable abgespeichert werden soll. Dadurch werden nur die gewünschten Werte gespeichert. Außerdem ist es durch das Hinzufügen weiterer switch-Fälle möglich, den Auswertevorgang um weitere in der Zeichenkette enthaltenden Werte zu erweitern.

Da die Auslese-Logik für den GPS-Sensor keine Überprüfung der Zeichenkette vornimmt, liegt diese Verantwortung bei dem Auswertungs-Programm. Hierbei wurde mit Hilfe von entsprechenden Tests (siehe Kapitel 6.4) ein umfangreiches Error-Handling implementiert.

Zum Beispiel wird, um bei vertauschter Reihenfolge der Zeichenketten die Auslesung trotzdem durchführen zu können, folgende Logik verwendet.

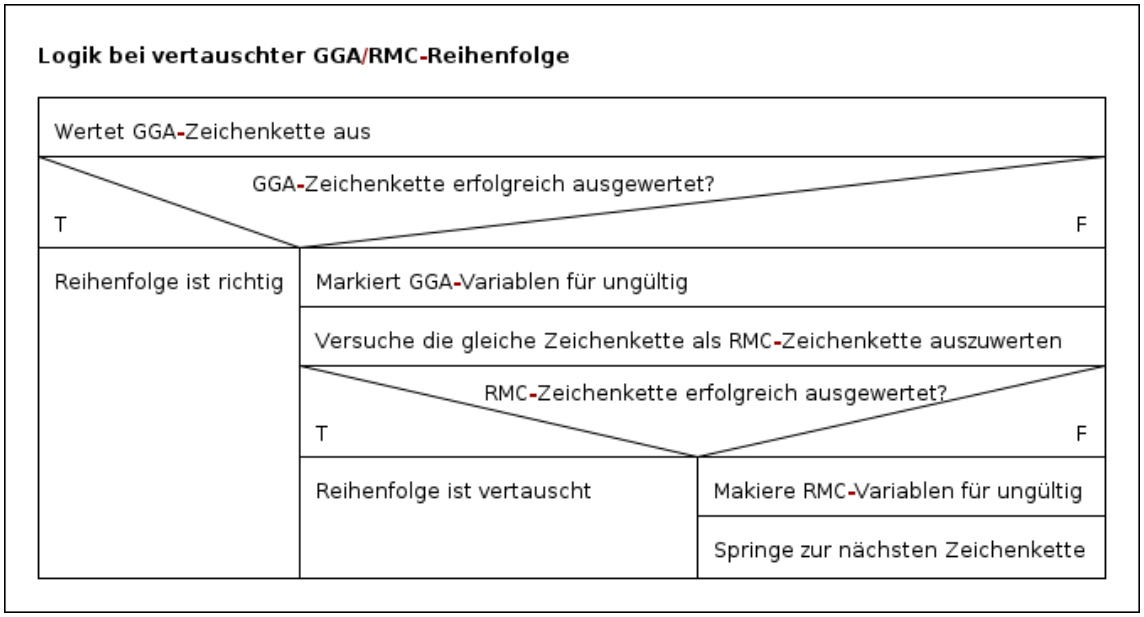


Abbildung 21: Struktogramm für Logik zum Überwinden einer vertauschten Reihenfolge

5.5. *MISRA-C*-Test

Das Dokument *MISRA C:2012 Guidelines for the use of the C language in critical systems*, in dem die *Misra-C*-Regeln enthalten sind, lässt sich für 45 £ in dem offiziellen *MISRA webstore* kaufen. [19]

Um die Regeln leichter überprüfen zu können, gibt es Compiler und statische code-analyse Werkzeuge, die eine Prüfung nach den *Misra-C*-Regeln durchführen. Ein solches statisches code-analyse Werkzeug ist die Software *PC-Lint*. *Lint*-Tools sind Programme, die eine statische Quellcode Analyse durchführen und dadurch den Entwickler auf mögliche Schwachstellen hinweisen. Mit *PC-Lint* kann unter anderem ein *MISRA-C*-Test durchgeführt werden. Dabei wird der Quellcode nach allen Regeln, die in den *MISRA C:2012*-Richtlinien stehen, geprüft. Eine Licence kostet 389 $. [20]

Für diese Arbeit wird die Software *Splint* verwendet. Hierbei handelt es sich um eine open-source Variante eines *Lint*-Werkzeugs. *Splint* wurde für die statische Quellcode Analyse von *C*-Programmen entwickelt. Über eine Konfigurationsdatei (.splintrc) lassen sich verschiedene Regeln aktivieren und deaktivieren. [21]

Splint beinhaltet keinen automatischen *MISRA-C*-Test. Um die *MISRA-C*-Regeln

trotzdem möglichst stark abzudecken, werden in der Konfigurationsdatei alle Regeln aktiviert, die in der *MISRA* Sprachkonvention vorgesehen sind. Dabei wird die von Ludolf Holzheid erstellte Konfigurationsdatei verwendet (siehe Quelle [22]). Durch dieses Vorgehen lässt sich keine vollständige *MISRA-C*-Konformität erreichen, da *Splint* nicht in der Lage ist, alle von *MISRA* vorgesehenen Tests durchzuführen.

Es wurden folgende Source-Dateien getestet:
GPS_Logger: main.c, DataLogger.c, LowFuncLogger.c, VenusGPS.c, ErrorStorage.c
GPS_Calc: main.c, DataCalc.c, LowFuncCalc.c, ErrorStorage.c

5.6. Datenaufkommen

Das Gesamtdatenaufkommen ist abhängig von der Frequenz mit der die Sensoren ausgelesen werden. Um einen Überblick über das Gesamtaufkommen zu erhalten, wird eine zehn Minuten lange Messung durchgeführt. Die dabei entstandenen Dateien sind in der folgenden Tabelle aufgeführt.

Tabelle 14: Überblick über Datenaufkommen aus 10 minütigen 30 Hz Logging-Prozess

Sensor	**Dateien**	**Frequenz, mit der die Werte vorliegen**	**Größe**	**Inhalt**
IMU	ACC.txt	30 Hz	1 MB	Beschleunigungswerte
	MAG.txt	30 Hz	1 MB	Magnetfeld
	GYR.txt	30 Hz	1 MB	Drehrate
	PR_TE.txt	30 Hz	873 kB	Druck/Temperatur
	HEIGHT.txt	30 Hz	144,3 kB	Höhe auf Grundlage von PR_TE.txt
Tür-Signal	DOOR.txt	30 Hz	584,3 kB	Zustand der Bustüren
Odometrie-Signal	TACHO.txt	30 Hz	653,2 kB	Anzahl der Odometrie-Pulse
GPS-Sensor	GPS_GGA.nmea	10 Hz	1,2 MB	GGA- und RMC-Zeichenkette
	POINTS.bin	einmalig	4 Bytes	Anzahl an GPS-Messpunkten
RTC	TIME.bin	einmalig	48 Bytes	Startzeit der Messung
Watchdog	WATCHDOG.txt	20 Hz	78,8 kB	Status des Busnetzes, Befehl zum Herunterfahren, Höhe der vom Akku zur Verfügung gestellten Spannung
/	Calc.txt	/	0 Bytes	Kein Inhalt, markiert Verzeichnis als ausgewertet
/	ERROR_LOGGER.txt	einmalig	297 Bytes	Fehlerspeicher des Logging-Programms
/	ERROR_CALC.txt	einmalig	125 Bytes	Fehlerspeicher des Auswertungs-Programms
/	HH_11_7301_2016 0614_1639_xxxx.txt	Zusammen-führung	43 kB	DOOR.txt, TACHO.txt, GPS_GGA.nmea (in mat-Datei mit 10 Hz zusammengeführt)

Gesmatgröße: 6,7 MB

6. Test

Um zu validieren, dass die entwickelte Systemarchitektur die Anforderungen erfüllt, werden in diesem Kapitel die durchgeführten Tests dokumentiert. Dabei wird zuerst auf die in der Software implementierten Validierungsfunktionen eingegangen. Anschließend werden die aus den Szenarien hervorgehenden Anforderungen behandelt. Die Validierung erfolgt dabei durch das Simulieren der Szenarien.

6.1. Vergleich der Zeitstempel

Der Vergleich der Zeitstempel wird in dem Ausführungs-Programm durchgeführt. Dabei werden die Zeitstempel von zwei aufeinander folgenden Werten subtrahiert und die Differenz wird mit einem Sollwert verglichen. Sobald die Abweichung der in ”DataCalc.h” festgelegten Prozentzahl überschreitet, gilt der ausgewertete Messpunkt als zeitlich falsch. Zum Schluss des Programms wird die Anzahl an Messpunkten, die durch diesen Vergleich als zeitlich fehlerhaft identifiziert wurden, im Fehlerspeicher gesichert. Durch diese Funktion lässt sich im Nachhinein eine Aussage über die Qualität der aufgezeichneten Messwerte machen. Da das Auswertungs-Programm nur die GPS-, Tür- und Odometrie-Rohdaten auswertet, kann auch nur für diese Sensoren der Zeitstempelvergleich durchgeführt werden. Die IMU wird trotzdem mit aufgeführt, da sie einen zeitlichen Einfluss auf den Logging-Programmablauf hat.

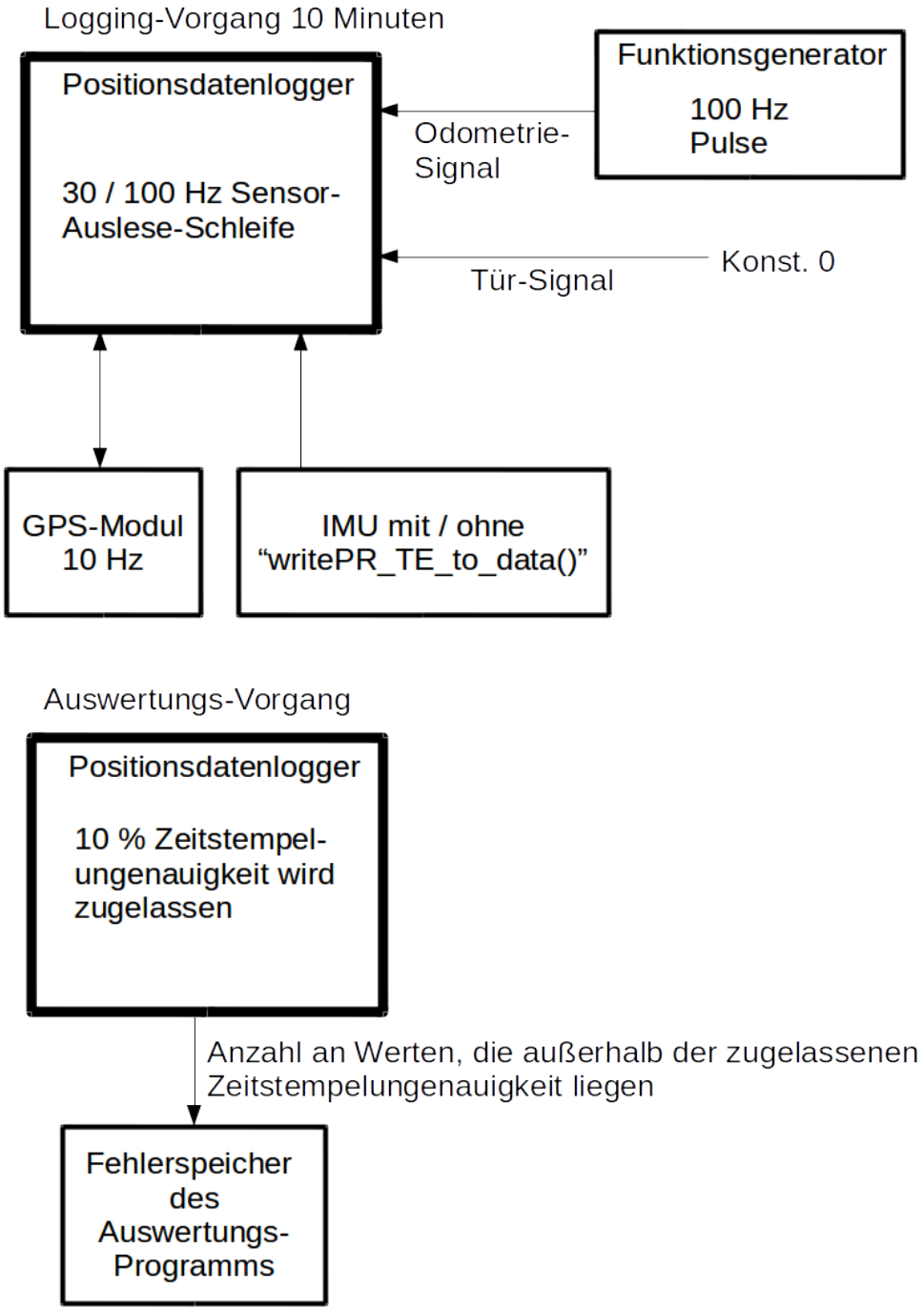


Abbildung 22: Versuchsaufbau: Zeitstempelvergleich

6.1.1. 30 Hz-Logging

Zuerst wird die Qualität der Messwerte im Normalbetrieb untersucht. Hiefür werden zehn aufeinanderfolgende Logging-Vorgänge mit eine while-Schleifen-Frequenz von 30 Hz und über eine Dauer von 10 Minuten aufgezeichnet. Außerdem werden konstant 100 Hz Odometrie-Pulse simuliert. Es wird eine Zeitstempelungenauigkeit von 10 % zugelassen. Zwischen den einzelnen Logging-Vorgängen wird jeweils ein Neustart des Einplatinencomputers durchgeführt.

Tabelle 15: 10 Minuten 30 Hz-Logging mit 100 Hz Odometrie-Pulsen, dabei wird eine Zeitstempelungenauigkeit von 10 % zugelassen (es wurden 10 Logging-Vorgänge durchgeführt)

Sensor	**Auslesefrequenz**	**Anzahl an gemessenen Werten**	**Durchschnittliche Anzahl an Werten außerhalb des zeitlichen Rahmens**
GPS-Sensor	10 Hz	6000	131 ($\sim 2\%$)
IMU mit "writePR_TE_ to_data()"	30 Hz	×	×
Tür-Signal	30 Hz	18000	687 ($\sim 4\%$)
Odometrie-Signal	30 Hz	18000	691 ($\sim 4\%$)

wird nicht ausgewertet = ×

Der Auslesevorgang des Odometrie- und Tür-Signals reagiert stärker auf nicht konstante Ausführungszeiten, da diese dreimal so oft ausgelesen werden, wie das GPS-Signal.

6.1.2. 100 Hz-Logging mit "writePR_TE_to_data()"

Nun wird eine while-Schleifen-Frequenz von 100 Hz getestet, wobei wieder 100 Hz Odometrie-Pulse simuliert werden. Die Dauer einer Messung beträgt wieder 10 Minuten.

Tabelle 16: 10 Minuten 100 Hz-Logging mit 100 Hz Odometrie-Pulsen, dabei wird eine Zeitstempelungenauigkeit von 10 % zugelassen (es wurde 1 Logging-Vorgänge durchgeführt)

Sensor	**Auslesefrequenz**	**Anzahl an gemessenen Werten**	**Anzahl an Werten außerhalb des zeitlichen Rahmens**
GPS-Sensor	10 Hz	1923	1922 ($\sim$ 100%)
IMU mit "writePR_TE_ to_data()"	100 Hz	×	×
Tür-Signal	100 Hz	19230	19229 ($\sim$ 100%)
Odometrie-Signal	100 Hz	19230	19229 ($\sim$ 100%)

wird nicht ausgewertet = ×

Wie in Tabelle 16 zu erkennen ist, lässt sich das Logging-Programm nicht mit einer Auslesefrequenz von 100 Hz betreiben. Dies liegt an der Ausführungszeit der durch die Bibliothek "adafruit_imu.h" bereitgestellten Funktion "writePR_TE_to_data()". Diese Funktion wird bei der IMU-Auslesung mit ausgeführt und besitzt eine Ausführungszeit zwischen 27 und 28 ms. Die bei 100 Hz zur Vergügung stehene Zeit für einen Schleifendurchlauf liegt bei 10 ms. Diese Laufzeitanforderung kann aufgrund der Funktion "writePR_TE_to_data()" nicht erfüllt werden. Deshalb darf die maximale Auslesefrequenz von 30 Hz nicht überschritten werden. Wenn die Ausführung dieser Funktion deaktiviert wird, lässt sich das Logging-Programm mit einer Frequenz von 100 Hz betreiben.

6.1.3. 100 Hz-Logging ohne "writePR_TE_to_data()"

Zum Schluss soll gezeigt werden, dass die Ausführung der while-Schleife mit 100 Hz möglich ist. Hiefür wird die Funktion "writePR_TE_to_data()" auskommentiert. Ansonsten wird der gleiche Logging-Vorgang wie in Kapitel 6.1.2 durchgeführt.

Tabelle 17: 10 Minuten 100 Hz-Logging mit 100 Hz Odometrie-Pulsen ohne ”writePR_TE_to_data()”, dabei wird eine Zeitstempelungenauigkeit von 10 % zugelassen (es wurden 10 Logging-Vorgänge durchgeführt)

Sensor	**Auslesefrequenz**	**Anzahl an gemessenen Werten**	**Durchschnittliche Anzahl an Werten außerhalb des zeitlichen Rahmens**
GPS-Sensor	10 Hz	6000	535 ($\sim$ 9%)
IMU ohne ”writePR_TE_ to_data()”	100 Hz	×	×
Tür-Signal	100 Hz	60000	3402 ($\sim$ 6%)
Odometrie-Signal	100 Hz	60000	3420 ($\sim$ 6%)

wird nicht ausgewertet = ×

Da bei 100 Hz der zeitliche Rahmen der für eine Messung zur Verfügung steht bei 10 ms liegt, erzeugt der GPS-Sensor einen höheren Anteil an zeitlich fehlerhaften Messwerten, da vor der Auslesung der ”poll”-Vorgang durchgeführt werden muss. Da der GPS-Sensor keine konstante Frequenz gewährleistet, mit der die Messpunkte in den Buffer geschrieben werden, kommt es teilweise zu Verzögerungen des Auslesevorgangs.

6.1.4. Fazit

Zusammenfassend lässt sich feststellen, dass das Logging-Programm eine maximale Auslesefrequenz von 30 Hz ermöglicht. Wenn eine höhere Frequenz gewählt wird, kann der volle Funktionsumfang nicht geboten werden. Dafür Verantwortlich ist die Funktion ”writePR_TE_to_data()”, welche durch eine hohe Ausführungszeit keine schnellere Frequenz zulässt.

6.2. Odometrie-Pulse > 111 Hz

Dieses Szenario wird ebenfalls mit Hilfe des Zeitstempelvergleichs (siehe Kapitel 6.1) getestet. Durch den Vergleich der Zeitstempel lässt sich eine Aussage über den zeitlichen Einfluss der Odometrie-ISR auf den Programmablauf machen. Dabei werden zehn aufeinanderfolgende 30 Hz-Logging-Vorgänge durchgeführt, bei denen

die Odometrie-Pulse mit 250 Hz anliegen. Dies entspricht einer Busgeschwindigkeit von 225 km/h. Diese Messreihe wird dann mit der Messung 15 verglichen.

Tabelle 18: 10 Minuten 30 Hz-Logging mit 250 Hz Odometrie-Pulsen, dabei wird eine Zeitstempelungenauigkeit von 10 % zugelassen (es wurden 10 Logging-Vorgänge durchgeführt)

Sensor	**Auslesefrequenz**	**Anzahl an gemessenen Werten**	**Durchschnittliche Anzahl an Werten außerhalb des zeitlichen Rahmens**
GPS-Sensor	10 Hz	6000	137 ($\sim$ 2%)
IMU mit "writePR_TE_ to_data()"	30 Hz	×	×
Tür-Signal	30 Hz	18000	692 ($\sim$ 4%)
Odometrie-Signal	30 Hz	18000	696 ($\sim$ 4%)

wird nicht ausgewertet = ×

Bei diesem Test wurde zusätzlich herausgefunden, dass die Odometriepulse nur dann aufgezeichnet werden, wenn das Busnetz eingeschaltet ist. Außerdem wurde festgestellt, dass das Ausschalten des Busnetzes einen Odometriepuls auslöst, auch wenn keine Odometriepulse anliegen. Der Grund hierfür ist der Aufbau der elektronischen Schaltung, welche die Aufnahme der Odometriepulse umsetzt.

Durch den Vergleich der beiden Messungen lässt sich sagen, dass die durchschnittliche Anzahl an zeitlich fehlerhaften Werten nur minimal voneinander abweichen. Bei 100 Hz Odometrie-Pulsen wurden 2 % der aufgezeichneten GPS-Messwerte zeitlich falsch aufgezeichnet. Im Vergleich dazu beträgt die durchschnittliche Anzahl an zeitlich fehlerhaften Werten bei 250 Hz Odometrie-Pulsen 137, was im Verhältnis zu den 6000 GPS-Messwerten, welche insgesmat aufgezeichnet wurden, 2 % entspricht. Abschließend lässt sich sagen, dass der Programmablauf durch eine Erhöhung der Odometrie-Pulse nicht beeinflusst wird.

6.3. Watchdog-Dekodierungstest

Die Kommunikation von der *Micro-Power*-Platine zum *Raspberry Pi* wird durch die Durchführung eines Dekodierungstests validiert. Durch diesen Test wird sowohl die Watchdog-ISR als auch die in der Auslese-Schleife durchgeführte Dekodierung getestet.

Dabei wird ein GPIO-Pin des Einplatinencomputers verwendet, um Pulssequenzen zu erzeugen. Diese Sequenzen sind so aufgebaut, dass jeder mögliche Zustand der Codierung durchlaufen wird. Um den Test durchzuführen, muss der Pin auf dem normalerweise das Signal des Watchdogs ankommt mit dem Pin, welcher die Sequenzen erzeugt, verbunden werden. Außerdem muss in der Header-Datei ”Data-Logger.h” das Macro ”WATCHDOG_TEST” definiert werden.

Bei dem Test werden durch eine Funktion alle Pulssequenzen erzeugt, die einer Information zugeordnet sind.

Tabelle 19: Ergebnis des Dekodierungstests

Signal (Teil 1 | Teil 2)	**Dekodierte Information aus WATCHDOG.txt (Info aus Teil 1 | Info aus Teil 2)**
0 Pulse, Sync | 4 Pulse, Sync	Busnetz aus | Akku-Spannung: 3.000V
1 Pulse, Sync | 5 Pulse, Sync	Busnetz an | Akku-Spannung: 3.100V
2 Pulse, Sync | 6 Pulse, Sync	2 Pulse (Undefinierter Zustand) | Akku-Spannung: 3.200V
3 Pulse, Sync | 7 Pulse, Sync	1ter Befehl zum shutdown | Akku-Spannung: 3.300V
3 Pulse, Sync | 8 Pulse, Sync	2ter Befehl zum shutdown | Akku-Spannung: 3.400V
3 Pulse, Sync | 9 Pulse, Sync	3ter Befehl zum shutdown | Akku-Spannung: 3.500V
3 Pulse, Sync | 10 Pulse, Sync	4ter Befehl zum shutdown | Akku-Spannung: 3.600V
3 Pulse, Sync | 11 Pulse, Sync	Befehl von Watchdog: shutdown durchführen | Akku-Spannung: 3.700V
3 Pulse, Sync | 12 Pulse, Sync	Befehl von Watchdog: shutdown durchführen | Akku-Spannung: 3.800V
...	...
3 Pulse, Sync | 24 Pulse, Sync	Befehl von Watchdog: shutdown durchführen | Akku-Spannung: 5.300V

Synchronisationssequenz = Sync

Der Dekodierungstest wurde erfolgreich durchgeführt, da alle Zustände erkannt wurden.

6.4. Sensordaten nicht nach Spezifikation

Um dieses Szenario zu simulieren, wurde eine Textdatei erstellt, die fehlerhafte GGA- und RMC-Zeichenketten enthält. Diese Textdatei simuliert bei dem Test den GPS-Buffer. Dadurch lassen sich alle Fehlerfälle simulieren.

Alle hier aufgeführten Fehlerfälle sind im Normalbetrieb in der dargestellten Form aufgetreten.

6.4.1. Fehlerfreier Buffer

Der erste hier aufgeführte Test wurde mit einem fehlerfreien Buffer durchgeführt. Um schneller überprüfen zu können, ob die Auslesung korrekt durchgeführt wurde, sind die Sensorwerte eines GPS-Paketes (GGA- plus RMC-Zeichenkette) so gewählt, dass die Werte logisch zusammenpassen (siehe Tabelle 20).

```
1  $GPGGA,115944.099,0106.0000,N,00106.0000,E,0,01,0.0,1.1,M,0.0,M,,0000*61
2  $GPRMC,115944.099,V,0000.0000,N,00000.0000,E,00.59,001.1,280606,,,N*70
3  $GPGGA,115944.200,0212.0000,N,00212.0000,E,0,02,0.0,2.2,M,0.0,M,,0000*63
4  $GPRMC,115944.200,V,0000.0000,N,00000.0000,E,1.187,002.2,280606,,,N*72
5  $GPGGA,115944.299,0318.0000,N,00183.0000,E,0,03,0.0,3.3,M,0.0,M,,0000*63
6  $GPRMC,115944.299,V,0000.0000,N,00000.0000,E,01.78,003.3,280606,,,N*72
7  $GPGGA,115944.400,0424.0000,N,00424.0000,E,0,04,0.0,4.4,M,0.0,M,,0000*65
8  $GPRMC,115944.400,V,0000.0000,N,00000.0000,E,02.37,004.4,280606,,,N*74
9  $GPGGA,115944.500,0530.0000,N,00530.0000,E,0,05,0.0,5.5,M,0.0,M,,0000*64
10 $GPRMC,115944.500,V,0000.0000,N,00000.0000,E,02.97,005.5,280606,,,N*75
```

Listing 12: GPS-Test-Buffer: Fehlerfrei

Tabelle 20: Ergebnis bei fehlerfreiem Buffer

Zeichenkette	**Breitengrad / Geschwindigkeit**	**Längengrad / Richtung**	**Satelliten**	**Höhe**
GGA	1.1	1.1	1.0	1.1
RMC	1.1	1.1	×	×
GGA	2.2	2.2	2.0	2.2
RMC	2.2	2.2	×	×
GGA	3.3	3.3	3.0	3.3
RMC	3.3	3.3	×	×
GGA	4.4	4.4	4.4	4.4
RMC	4.4	4.4	×	×
GGA	5.5	5.5	5.5	5.5
RMC	5.5	5.5	×	×

Information ist in der Zeichenkette nicht vorhanden = ×

Alle Informationen wurden erfolgreich erkannt.

6.4.2. Vertauschte Reihenfolge

Bei diesem Fehlerfall wird eine GGA-Zeichenkette nicht vollständig in den Buffer geschrieben. Danach folgt nicht wie erwartet die RMC-Zeichenkette, sondern die selbe GGA-Zeichenkette, die vorher nicht vollständig übertragen wurde. Das Problem dabei ist, dass als nächste Zeichenkette die RMC-Zeichenkette des nächsten Messpunktes in den Buffer geschrieben wird. Dies erzeugt eine Vertauschung der Reihenfolge. Um diesen Fehlerfall in beide Richtungen zu simulieren, werden zwei Vertauschungen durchgeführt.

```
1 $GPGGA,115944.100,...
2 $GPRMC,115944.100,...
3 $GPGGA,115944.200,...(nicht vollständig)$GPGGA,115944.200,...
```

```
4  $GPRMC,115944.300,...
5  $GPGGA,115944.300,...
6  $GPRMC,115944.400,...(nicht vollständig)$GPRMC,115944.400,...
7  $GPGGA,115944.500,...
8  $GPRMC,115944.500,...
```

Listing 13: GPS-Test-Buffer: Vertauschte Reihenfolge

Tabelle 21: Ergebnis bei vertauschter Reihenfolge

Zeichenkette	**Breitengrad / Geschwindigkeit**	**Längengrad / Richtung**	**Satelliten**	**Höhe**
GGA	1.1	1.1	1.0	1.1
RMC	1.1	1.1	×	×
GGA	2.2	2.2	2.0	2.2
RMC	-1.0	-1.0	×	×
GGA	3.3	3.3	3.0	3.3
RMC	3.3	3.3	×	×
GGA	-1.0	-1.0	-1.0	-1.0
RMC	4.4	4.4	×	×
GGA	5.5	5.5	5.5	5.5
RMC	5.5	5.5	×	×

Information ist in der Zeichenkette nicht vorhanden = ×

Das erste Zeichenkettenpaar (Messpunkt) wird richtig ausgewertet. Beim zweiten Paar wird die nicht vollständig gesendete GGA-Zeichenkette ignoriert und die vollständige Zeichenkette wird verarbeitet. Die Auslese-Logik erkennt das Fehlen der RMC-Information in dem Messpunkt und markiert deshalb die Werte als ungültig. Hierfür wird die Zahl -1.0 in den entsprechenden Wert geschrieben. Dadurch wird der Wert eindeutig als ungültig markiert, da die Zahl -1.0 im Normalbetrieb nicht

auftritt. Der dritte Messpunkt, welcher nun zuerst die RMC- und dann die GGA-Zeichenkette beinhaltet, wird trotz vertauschter Reihenfolge richtig ausgewertet. Der darauf folgende Messpunkt enthält zwei RMC-Zeichenketten, wobei die erste nicht vollständig übertragen wird. Auch hier wird erkannt, dass die GGA-Information in diesem Messpunkt fehlt und nur die RMC-Information wird verarbeitet. Der letzte Messpunkt, welcher die Zeichenketten in erwarteter Reihenfolge beinhaltet, wird fehlerfrei erkannt.

6.4.3. Falsch übertragene Zeichen

Eine weitere identifizierte Fehlerursache sind falsch übertragene Zeichen.

```
1  $GPGGA,115944.099,0106.A000,N,00106.0000,E,0,01,0.0,1.1,M,0.0,M,,0000*61
2  $GPRMC,115944.099,V,0000.0000,N,00000.0000,AE,00.59,001.1,280606,,,N*70
3  $CPGGA,115944.200,0212.0000,N,00212.0000,E,0,02,0.0,2.2,M,0.0,M,,0000*63
4  $CPRMC,115944.200,V,0000.0000,N,00000.0000,E,1.187,002.2,280606,,,N*72
5  $GPGGA,115944.299,0318.0000,N,00183.0000,E,0,03,0.0,12345678.9,M,0.0,M,,0000*63
6  $GPRMC,115944.299,V,0000.0000,N,00000.0000,E,01.78,123456789.0,280606,,,N*72
7  $GPGGA,115944.400,0424.000ON,00424.0000,E,0,04,0.0,4.4,M,0.0,M,,0000*65
8  $GPRMC,115944.400,V,0000.0000,N,00000.0000,E,02.37,004.4,280606,,,N*74
9  $GPGGA,115944.500,0530.0000,N,00530.0000,E,0,05,0.0,5.5,M,0.0,M,,0000*64
10 $GPRMC,115944.500,V,0000.0000,N,00000.0000,E,02.97005.5,280606,,,N*75
```

Listing 14: GPS-Test-Buffer: Falsch übertragene Zeichen

Tabelle 22: Ergebnis bei falsch übertragenen Zeichen

Zeichenkette	**Breitengrad / Geschwindigkeit**	**Längengrad / Richtung**	**Satelliten**	**Höhe**
GGA	-1.0	-1.0	-1.0	-1.0
RMC	1.1	1.1	×	×
GGA	2.2	2.2	2.0	2.2
RMC	2.2	2.2	×	×
GGA	3.3	3.3	3.0	12345678.9
RMC	-1.0	-1.0	×	×
GGA	-1.0	-1.0	-1.0	-1.0
RMC	4.4	4.4	×	×
GGA	5.5	5.5	5.5	5.5
RMC	-1.0	-1.0	×	×

Information ist in der Zeichenkette nicht vorhanden = ×

Ein falsch übertragenes oder zusätzliches Zeichen wird nur dann erkannt, wenn es in einem auszuwertenden Sensorwert auftritt. Genau das Verhalten lässt sich im ersten und zweiten Messpunkt beobachten.
Da im ersten Messpunkt das falsch übertragene Zeichen in der Breitengradinformation auftritt, wird die Zeichenkette als ungültig markiert. Die RMC-Zeichenkette liegt fehlerfrei vor und wird richtig ausgewertet.
Im zweiten Messpunkt werden die benötigten Werte richtig abgespeichert, da das fehlerhafte Zeichen nicht in einer auszuwertenden Sensorinformation vorliegt.
Beim dritten Messpunkt wird das Verhalten beim Auftritt einer zu langen Zeichenfolge innerhalb eines auszuwertenden Sensorwertes getestet. Sobald die Zeichenfolge eine bestimmte Länge (in diesem Fall 10 Zeichen) überschreitet, wird die Auslesung als fehlerhaft markiert.
Im vierten und fünften Messpunkt wurde ein Komma nicht übertragen. Durch das Fehlen des Kommas können die Sensorinformationen nicht voneinander getrennt

werden. Die betroffene Zeichenkette wird als ungültig markiert, unabhängig davon, an welcher Stelle das Komma fehlt.

6.4.4. Zusammenfassung

Um das Verhalten der Software zu kontrollieren, wurde in diesem Test das Szenario "Sensordaten nicht nach Spezifikation" für den GPS-Sensor simuliert. Als Ergebnis lässt sich festhalten, dass alle getesteten Sensordefekte erfolgreich erkannt und überwunden wurden.

Die anderen Sensormodule werden mit der gleichen fehlertoleranten Logik ausgewertet.

6.5. System reagiert nicht mehr

Wenn man die Kommunikation vom *Raspberry Pi* zum Watchdog betrachtet, lassen sich mehrere Varianten dieses Szenarios erkennen. Da ein periodischer Pegelwechsel durchgeführt wird, um dem Watchdog den aktuellen Software-Ausführungsstatus des Einplatinencomputers mitzuteilen, können bei einem unerwarteten Aufhängen der Software zwei verschiedene Zustände auftreten. Entweder kann der Programmablauf bei einem gerade anliegenden "HIGH"-Pegel stoppen oder das Programm hängt sich bei einem gerade anliegenden "LOW"-Pegel auf.

Um das implementierte Verhalten des Einplatinencomputers beim Erkennen des Befehls zum Herunterfahren zu testen, wird dieses Szenario ohne Beeinflussung des Programmablaufs getestet. Dabei wird das ‚durch den periodischen Pegelwechsel erzeugte, Lebenssignal durch ein konstantes "HIGH"- oder "LOW"-Signal ersetzt. Wenn der Befehl zum Herunterfahren vom *Raspberry Pi* erkannt wird, soll automatisch der Logging-Prozess beendet und das System heruntergefahren werden. Da nur ein gewisses Zeitfenster bis zur Trennung zur Verfügung steht, wird dabei das Auswertungs-Programm nicht gestartet. Dadurch wird verhindert, dass während der Ausführung des Auswertungs-Programms die Spannung getrennt wird.

Tabelle 23: Verhalten bei konstantem Pegel als Lebenssignal

Aktion	**Signal vom *Raspberry Pi* zum Watchdog**	**Signal vom Watchdog zum *Raspberry Pi***	**Ausführungszeit**	
Normalbetrieb	Pegelwechsel	Busnetz an | Akku-Spannung: 5.000V	×	
Logging-Programm wird ausgeführt	konstant 1	Busnetz an | Akku-Spannung: 5.000V	15 Minuten	
Logging-Programm wird ausgeführt	konstant 1	1ter...4ter Befehl zum shutdown | Akku-Spannung: 5.000V		
Logging-Programm wird beendet, Auswertungs-Programm wird nicht ausgeführt, System fährt runter	×	Befehl von Watchdog: shutdown durchführen | Akku-Spannung: 5.000V	2 Minuten	
System aus	×	Befehl von Watchdog: shutdown durchführen | Akku-Spannung: 5.000V		10 Minuten
Trennung	×	×	×	
System spannungsfrei	×	Busnetz an | Akku-Spannung: 5.000V	2 Minuten	
System fährt hoch	×	Busnetz an | Akku-Spannung: 5.000V	1 Minute	
Logging-Programm wird ausgeführt, Trennung nicht möglich	konstant 0	Busnetz an | Akku-Spannung: 5.000V	5 Minuten	
Logging-Programm wird ausgeführt	konstant 0	Busnetz an | Akku-Spannung: 5.000V	15 Minuten	
Logging-Programm wird ausgeführt	konstant 0	1ter...4ter Befehl zum shutdown | Akku-Spannung: 5.000V		
...	...	...	...	...

Bei dem Test hat sich herausgestellt, dass die im Trinket-Programm eingestellten Zeiten vom Trinket nicht genau eingehalten werden. Gerade bei längeren Zeiten konnte eine Differenz festgestellt werden. So musste das konstante Signal ungefähr 17 Minuten und nicht wie definiert 15 Minuten anliegen, um einen Watchdog-Eingriff auszulösen.

Abschließend lässt sich sagen, dass bei fehlender Reaktion die Wiederherstellung des lauffähigen Zustands erreicht wird.

6.6. Busstart bei ungeladenem Akku

Dieses Szenario wird getestet, indem der Positionsdatenlogger mit einem schwach geladenen Akku und einem ausgeschalteten Busnetz betrieben wird. Dadurch entlädt sich der Akku langsam, bis das System aufgrund des leeren Akkumulators ausfällt. Nun lässt sich durch das Einschalten des Busnetzes prüfen, ob der Positionsdatenlogger seine Funktion ausführt, wenn der Busstart bei einem ungeladenen Akku durchgeführt wird. Die folgende Tabelle dokumentiert die Ergebnisse dieses Versuches.

Tabelle 24: Verhalten bei ungeladenen Akku

Aktion	**Signal vom *Raspberry Pi* zum Watchdog**	**Signal vom Watchdog zum *Raspberry Pi***	**Ausführungszeit**
Normalbetrieb	Pegelwechsel	Busnetz an | Akku-Spannung: 5.000V	×
Busnetz aus	Pegelwechsel	Busnetz aus | Akku-Spannung: 5.000V	bis Akku entladen
Akku entladen, System spannungsfrei	×	×	×
Busnetz an	×	×	×
Akku wird geladen, System fährt hoch	×	Busnetz an | Akku-Spannung: 5.000V	1 Minute
Normalbetrieb	Pegelwechsel	Busnetz an | Akku-Spannung: 5.000V	×

Der Normalbetrieb wird trotz entladenen Akku erreicht.

Wenn ein komplett entladener Akku verwendet wird, was im Normalbetrieb nicht passiert, da hierfür der Akku mehrfach bis zum Ausfall entladen werden müsste, tritt beim Hochfahren des Systems ein Problem auf. Das Hochfahren benötigt eine sehr hohe Stromzufuhr. Die Folge dieser hohen Stromzufuhr ist, dass der Entladevorgang des Akkumulators größer ist als der Ladevorgang. Dadurch wird der Akku während des Hochfahrens des Systems komplett entladen und das Gesamtsystem fällt aus, obwohl der Akku durch das Busnetz geladen wird. Wenn der Zustand eines komplett entladenen Akkumulators eintritt, ist die Wiederherstellung des Normalbetriebs innerhalb des aktuellen Messvorgangs nicht möglich. Nachdem das Gesamtsystem ausgefallen ist, wird der Akku durch das Busnetz geladen und der Normalbetrieb kann im nächsten Messvorgang, also nach der nächsten Aus- und Einschaltung des Busnetzes, fortgeführt werden. Dabei wird ein Messvorgang und damit ein kompletter Messtag ausgelassen.

Eine bessere Lösung wäre es, beim Einschalten des Busnetzes das Einschalten des Positionsdatenloggers zeitlich zu verzögern. Dadurch hätte der Akku eine gewisse Zeit, indem nur der Ladevorgang stattfindet.

7. Zusammenfassung der Arbeit

Durch diese Arbeit konnte das autonome Verhalten des Positionsdatenlogger grundlegend verbessert werden. Dabei wurden Software- sowie Hardware-Konzepte umgesetzt und innerhalb einer Systemarchitektur verbunden.

Das dabei entstandene System ist in der Lage den Programmablauf beliebig lange aufrecht zu erhalten. Selbst bei unüberwindbaren Zuständen, wird durch den Watchdog die Spannung getrennt, wodurch der Positionsdatenlogger die Datenaufzeichnung wieder aufnehmen kann. Auch ein Großteil der zuerwartenden Fehler wurden in der Software berücksichtigt und soweit es möglich ist abgefangen.

Durch die Modularität der Systemarchitektur können nun dem Positionsdatenlogger ohne großen Aufwand neue Sensoren hinzugefügt werden. Jeder Sensor kann mit einer individuellen Frequenz ausgelesen werden. Außerdem kann entschieden werden, welche Sensorwerte ausgewertet und zur mat-Datei hinzugefügt werden sollen und welche nicht.

Zusammenfassend lässt sich sagen, dass durch diese Arbeit die allegemeine Qualität der Software erheblich verbessert wurde. Dazu zählt die Erhöhung der Wartbarkeit und Transperenz durch einen modularen Aufbau, die Verbesserung der Benutzbarkeit durch die Verwendung von Macros und der Sensorauslese-/Dateiauslese-Funktion, die Optimierung der Testbarkeit durch die in der Software implementierten Validierungsfunktionen und die Steigerung der Zuverlässigkeit durch die fehlertoleranten Algorithmen und die Verwendung eines Watchdogs.

In Zukunft soll die Architektur auf den 12 Postitionsdatenloggern eingesetzt werden. Hierfür muss eine Portierung auf das Betriebssystem, welches mit Hilfe des *Yocto-Projektes* erstellt wurde, durchgeführt werden. Diese Portierung sollte ohne Probleme durchgeführt werden können. Um eine Logging-Frequenz von 100 Hz bieten zu können ohne den Funktionsumfang einzuschränken, müsste die Funktion "writePR_TE_to_data()" hinsichtlich der Ausführungszeit optimiert werden.

Abbildungsverzeichnis

Tabellenverzeichnis

Quellcodeverzeichnis

Abkürzungsverzeichnis

Akku Akkumulator

BEEDeL Bewertung des Einsatzes von Elektrobussen mit Dezentraler Ladeinfrastruktur

CPU Central Processing Unit

GPIO General-Purpose Input/Output

GPS Global Positioning System

GSM Global System for Mobile Communications

GUI Graphical User Interface

I2C Inter-Integrated Circuit

IDE Integrated Development Environment

IMU Inertial Measurement Unit

ISR Interrupt Service Routine

IVI Fraunhofer-Institut für Verkehrs- und Infrastruktursysteme

LED Light-Emitting Diode

mat Microsoft Access Table

MISRA Motor Industry Software Reliability Association

NMEA National Marine Electronics Association

PL/I Programming Language One

RTC Real Time Clock

SD Secure Digital

SMA SubMiniature version A

SoC System on a Chip

SPI Serial Peripheral Interface

UART Universal Asynchronous Receiver/Transmitter

USB Universal Serial Bus

WLAN Wireless Local Area Network

Literatur

[1] Mario Wegner. *Entwicklung eines autonom arbeitenden GPS-Datenloggers mit hoher Updatefrequenz für die Anwendung in Nahverkehrsbussen.* Bachelorarbeit HAW-Hamburg, 2015.

[2] Rasmus Rettig. *Vortrag BEEDeL Steuerkreis (3. Sitzung).* 26.07.2016.

[3] ISO/IEC 25010:2011. *Systems and software engineering.* DIN-Norm, 2011.

[4] Dirk W. Hoffmann. *Software-Qualität.* 2. Auflage, Springer Vieweg, 2012.

[5] *About Us.* RASPBERRY PI FOUNDATION. https://www.raspberrypi.org/about/ (Zugriffsdatum: 02.06.2016).

[6] *RASPBERRY PI 1 MODEL B+.* RASPBERRY PI FOUNDATION. https://www.raspberrypi.org/products/model-b-plus/ (Zugriffsdatum: 03.06.2016).

[7] *Raspberry Pi Model B+ Mainboard (GPIO polig, MicroSD Speicherkartenslot, HDMI, 4x USB 2.0).* Amazon, 2016. https://www.amazon.de/Raspberry-Model-Mainboard-MicroSD-Speicherkartenslot/dp/B00LPESRUK/ref=sr_1_3?ie=UTF8&qid=1467665781&sr=8-3&keywords=raspberry+pi+b%2B (Zugriffsdatum: 04.07.2016).

[8] *Venus638FLPx GPS Receiver Data Sheet.* SkyTraq Technology, Inc. http://cdn.sparkfun.com/datasheets/Sensors/GPS/Venus/638/doc/Venus638FLPx_DS_v07.pdf (Zugriffsdatum: 03.06.2016).

[9] Stephan Vette. *Entwicklung eines eingebetteten Multi-Sensor-Systems.* Studienarbeit HAW-Hamburg, 2015.

[10] Jan Kastning. *Packaging eines GPS-Datenloggers zur Integration in einen Linienbus.* Studienarbeit HAW-Hamburg, 2015.

[11] *Wiring Pi - GPIO Interface library for the Raspberry Pi.* Gordon Henderson. http://wiringpi.com/ (Zugriffsdatum: 09.06.2016).

[12] *Matio - MAT File I/O Library.* Christopher Hulbert. https://sourceforge.net/p/matio/matio/ci/master/tree/ (Zugriffsdatum: 09.06.2016).

[13] *About.* https://www.yoctoproject.org/about (Zugriffsdatum: 04.07.2016).

[14] Roman Kray. *Aufbau einer VPN-Servers für die zentrale Zusammenführung von GPS-Informationen.* Studienarbeit HAW-Hamburg, 2015.

[15] Carsten Vogt. *Nebenläufige Programmierung: ein Arbeitsbuch mit UNIX/Linux und Java*. Hanser, 2012.

[16] Anthony Williams. *C++ Concurrency in Action*. Manning Publications, 2012.

[17] *What is Raspbian?* https://www.raspbian.org/ (Zugriffsdatum: 14.09.2016).

[18] Andre Timmann. *Weiterentwicklung des GPS-Datenloggers für Kraftfahrzeuge*. Studienarbeit HAW-Hamburg, 2016.

[19] *MISRA webstore*. 2016. http://www.misra.org.uk/shop/buy_now.php (Zugriffsdatum: 11.07.2016).

[20] Gimpel Software. *PC-lint for C/C++*. 2016. http://www.gimpel.com/html/pcl.htm (Zugriffsdatum: 11.07.2016).

[21] David Evans. *Splint*. Secure Programming Group at the University of Virginia Department of Computer Science. http://splint.org/ (Zugriffsdatum: 14.06.2016).

[22] Ludolf Holzheid. *[splint-discuss] MISRA for splint*. 2007. http://www.cs.virginia.edu/pipermail/splint-discuss/2007-March/000923.html (Zugriffsdatum: 14.06.2016).

[23] *Raspberry Pi2 GPIO Header*. 2014. https://www.element14.com/community/servlet/JiveServlet/previewBody/73950-102-4-309126/GPIO_Pi2.png?01AD=3nYlHfH_ZcxEpfuWPTWj4sRNOY_hh90khVPeRWMmvGWAGZuyy_KdG0Q&01RI=88F61A06991B950&01NA= (Zugriffsdatum: 19.06.2016).

[24] Maik Christoph Schöne. *Entwicklung eines integrierten Messsystems für einen GPS-Datenlogger mit verbesserter Höheninformation durch Sensorfusion*. Bachelorarbeit HAW-Hamburg, 2015.

A. Anhang: CD

- Abschlussarbeit
  - Abschlussarbeit im PDF-Format
- Quellcode
  - Vollständiger Quellcode des Logging-Programms ”GPS_Logger”
  - ”Code:Blocks”-Projekt des Logging-Programms ”GPS_Logger”
  - Vollständiger mega nicer Quellcode des Auswertungs-Programms ”GPS_Calc”
  - ”Code:Blocks”-Projekt des Auswertungs-Programms ”GPS_Calc”
  - Skript zur Datenübertragung (entnommen aus [14])
  - Startup-Skript (entnommen aus Quelle [1])
- Sonstiges
  - GPIO-Pin-Belegung des *Raspberry Pis* (entnommen aus Quelle [23])
  - Verwendeter GPS-Test-Buffer für die Simulierung eines Sensordefektes
  - Ergebnis Dekodierungstest
  - *Splint*-Konfigurationsdatei (entnommen aus Quelle [21])

B. Anhang: Quellcode

B.1. Logging-Programm

B.1.1. main.c

Die aus der Quelle [1] übernommenen Programmabschnitte wurden gekennzeichnet.

```
1
2  #include <stdio.h>
3  #include <stdlib.h>
4  #include <stdint.h>
5  #include <string.h>
6  #include <unistd.h>
7  #include <errno.h>
8  #include <termios.h>
9  #include <fcntl.h>
10 #include <unistd.h>
11 #include <sys/stat.h>
12 #include <time.h>
13 #include <sys/time.h>
14 #include <math.h>
15 #include <linux/i2c-dev.h>
16 #include <sys/ioctl.h>
17 #include <signal.h>
18
19 #include "global.h"
20
21 # ifndef S_SPLINT_S
22 #include "wiringPi.h"    // Zum Einbinden der Library: Build Options
23                          //-> Linker Settings -> Add usr/local/lib/libwiringPi.h
24 # endif
25
26 #include "VenusGPS.h"
27 #include "DataLogger.h"
28 # ifndef S_SPLINT_S
29 #include "adafruit_imu.h"
30 # endif
31 #include "HeightLib.h"
32 #include "ErrorStorage.h"
33 #include "LowFuncLogger.h"
34
35 volatile unsigned long int tacho_timeout_counter = TACHO_TIMEOUT;
36 //wie lange soll weiter geloggt werden, wenn Bus steht?
37 volatile unsigned long int tick = 0;     //zählt die Odemetrieflanken durch ISR.
38
```

```
39 volatile int watchdog_pulse = -1; //zählt Watchdog-Pulse durch ISR.
40 volatile int global_akku_spannung = -1; //speichert WD-Akku-Spannung.
41 volatile int global_netz_shutdown = -1; //speichert WD-Info.
42 volatile struct timespec start_wd = {999999, 9999999L};
43 //für Pulse-Zeit-Messung in WD-ISR.
44 int while_loop_wrong = 0;   //+1 falls while_loop zu langsam
45 /*@null@*/FILE *fp_ERROR; //Fehlerspeicher.
46
47 int main(void)
48 {
49     int fd = -1;            //Venus-File_Descriptor.
50     char c = (char) 0x00;   //Navigationsmodus für Venus GPS.
51                             //0x00 car, 0x01 pedestrain.
52     int rate = GPS_FREQ;    //GPS Sendefrequenz in Hz.
53
54     int fd_i2c = -1;        //IMU File-Descriptor.
55     double t_pressure = -1.0; //Der Druck wird von der HeightLib benötigt
56
57     char door[3 + 1] = "0\n$"; //wird von der HeightLib benötigt
58
59     struct timespec begin = {0};  //Zeitmessung while-Durchlauf.
60     struct timespec end = {0};
61     /*@unused@*/struct timespec my_tic = {0};  //Zeitmessung zum Debugging.
62     /*@unused@*/struct timespec my_toc =  {0};
63
64     int button = 1;         //HW Taster.
65
66     char fdir[] = "/media/INTENSO/Data000/"; //Start-Verzeichnis auf USB-Stick.
67
68     int watchdog_out = 0; //HIGH oder LOW. Lebenszeichen.
69     int shutdown_counter = 0; //shutdown-Info muss repetitiv auftreten.
70
71     char while_loop_wrong_string[60 + 1]= "";
72
73     int loop_counter_gps = 1; //steuert Auslesefreq der einzelnen Sensoren.
74     int loop_counter_door = 1;
75     int loop_counter_tacho = 1;
76     int loop_counter_acc = 1;
77     int loop_counter_mag = 1;
78     int loop_counter_gyr = 1;
79     int loop_counter_pr_te = 1;
80     int loop_counter_watchdog_output = 1;
81     int loop_counter_watchdog_input = 1;
82     int max_freq = WHILE_FREQ;  //while-Frequenz (max Auslesefreq).
83
```

```
84    unsigned long int points_gps = 0;      //Zählt GPS_Messungen.
85    unsigned long int index_fastest = 0;  //Zähl-Index der while-Schleife.
86
87    /*@null@*/FILE *all_files[MAX_FILES] = { NULL }; //alle fp in einem Array.
88  //index:                 0             1           2               3
89    char *file_names[] = {"POINTS.bin", "TIME.bin", "GPS_GGA.nmea", "DOOR.txt",
90  //4            5             6          7          8          9
91    "TACHO.txt", "HEIGHT.txt", "ACC.txt", "MAG.txt", "GYR.txt", "PR_TE.txt",
92  //10
93    "WATCHDOG.txt"};  //alle Dateinamen.
94  //index:                 0             1           2               3
95    char *file_mode[]  = {"wb"        , "wb"      , "w"           , "w"       ,
96  //4            5             6          7          8          9
97    "w"        , "w"         , "w"      , "w"      , "w"      , "w"        ,
98  //10
99    "w"           };  //alle Öffnungsmodi.
100
101   (void) printf("\n################################################\n"
102         "GPS_Logger start----------\n"
103         "################################################\n");
104
105   if(makedir(fdir) == EXIT_FAILURE)
106   {
107       (void) printf("->makedir()\n");
108       return EXIT_FAILURE;
109   }
110   if((fp_ERROR = openfile(fdir, "ERROR_LOGGER.txt", "w")) == NULL)
111   {
112       (void) printf("->openfile(fp_ERROR)");
113       return EXIT_FAILURE;
114   }
115   if(open_all_files(fdir, all_files, file_names, file_mode) == EXIT_FAILURE)
116   {
117       (void) printf("->open_all_files()\n");
118       return EXIT_FAILURE;
119   }
120
121   save_error("GPS_LOGGER\n");
122   save_error(fdir);
123   save_error(":\n");
124
125   if(initializes_gpio() == EXIT_FAILURE)
126   {
127       save_error("->initializes_gpio()\n");
128       return EXIT_FAILURE;
```

```
129     }
130     #ifdef WATCHDOG_TEST
131         test_watchdog_ISR(all_files[10]);
132 
133         if(close_all_files(all_files) == EXIT_FAILURE)
134         {
135             save_error("->close_all_files()\n");
136             return EXIT_FAILURE;
137         }
138         if(fclose(fp_ERROR) == EOF)
139         {
140             (void) printf("%s", strerror(errno));
141             (void) printf("->fclose(fp_ERROR)");
142             return EXIT_FAILURE;
143         }
144 
145         return EXIT_SUCCESS;
146     #endif
147     #ifndef TEST_BUFFER
148        waitForStart();
149     #endif
150     #ifndef S_SPLINT_S
151     digitalWrite(GREEN_LED,LED_ON);
152     #endif
153     (void) heightSetup(all_files[5]);       // / Hoehe ////////
154     #ifndef S_SPLINT_S
155     if((fd_i2c = initializes_imu()) == -1)
156     {
157         save_error("->initializes_imu()\n");
158         return EXIT_FAILURE;
159     }
160     #endif
161     #ifdef TEST_BUFFER
162         if((fd = open(TEST_BUFFER_LOCATION, O_RDWR | O_NOCTTY | O_NDELAY)) < 0)
163         {
164             return EXIT_FAILURE;
165         }
166     #endif
167     #ifndef TEST_BUFFER
168         if((fd = initializes_gps(c, rate)) == -1)
169         {
170             save_error("->initializes_gps()\n");
171             return EXIT_FAILURE;
172         }
173     #endif
```

```
174     #ifndef S_SPLINT_S
175     digitalWrite(GREEN_LED,LED_OFF);
176     #endif
177     if(write_time(all_files[1]) == EXIT_FAILURE)
178     {
179         save_error("->write_time()\n");
180         return EXIT_FAILURE;
181     }
182
183     if(read_venus_init(fd) == EXIT_FAILURE)
184     {
185         save_error("->read_venus_init()\n");
186         return EXIT_FAILURE;
187     }
188     (void) printf("\n-------- LOGGING STARTEN----------\n");
189     tacho_timeout_counter = 0;
190     tick = 0;
191     while((tacho_timeout_counter < TACHO_TIMEOUT) &&
192           (button != 0) && (shutdown_counter <= REPEAT_SHUTDOWN_SIGNAL)
193           #ifndef S_SPLINT_S
194           && (digitalRead(STOP_PIN) != 1)
195           #endif
196          )
197     {
198         clock_gettime(CLOCK_MONOTONIC, &begin);
199
200         #ifdef PRINT_TIME
201         clock_gettime(CLOCK_MONOTONIC, &my_tic);
202         #endif
203         if(write_door(all_files[3], door, max_freq, DOOR_FREQ,
204                       &loop_counter_door, index_fastest) == EXIT_FAILURE)
205         {
206             save_error("->write_door()\n");
207             break;
208         }
209         #ifdef PRINT_TIME
210         clock_gettime(CLOCK_MONOTONIC, &my_toc);
211         printf("DOOR: %lf ms\n", calc_time(my_tic, my_toc));
212
213         clock_gettime(CLOCK_MONOTONIC, &my_tic);
214         #endif
215         if(write_tacho(all_files[4], max_freq, TACHO_FREQ,
216                        &loop_counter_tacho, index_fastest) == EXIT_FAILURE)
217         {
218             save_error("->write_tacho()\n");
```

```
219             break;
220         }
221         #ifdef PRINT_TIME
222         clock_gettime(CLOCK_MONOTONIC, &my_toc);
223         printf("TACHO: %lf ms\n", calc_time(my_tic, my_toc));
224 
225         clock_gettime(CLOCK_MONOTONIC, &my_tic);
226         #endif
227         if(write_imu(fd_i2c, all_files[6], all_files[7], all_files[8],
228                      all_files[9], &t_pressure, max_freq,
229                      IMU_ACC_FREQ, IMU_MAG_FREQ, IMU_GYR_FREQ, IMU_PR_TE_FREQ,
230                      &loop_counter_acc, &loop_counter_mag, &loop_counter_gyr,
231                      &loop_counter_pr_te, index_fastest) == EXIT_FAILURE)
232         {
233             save_error("->write_imu()\n");
234             break;
235         }
236         #ifdef PRINT_TIME
237         clock_gettime(CLOCK_MONOTONIC, &my_toc);
238         printf("IMU_Gesamt: %lf ms\n", calc_time(my_tic, my_toc));
239 
240         clock_gettime(CLOCK_MONOTONIC, &my_tic);
241         #endif
242         heightSetData(t_pressure, door[0]);
243         #ifdef PRINT_TIME
244         clock_gettime(CLOCK_MONOTONIC, &my_toc);
245         printf("HEIGHT: %lf ms\n", calc_time(my_tic, my_toc));
246 
247         clock_gettime(CLOCK_MONOTONIC, &my_tic);
248         #endif
249         #ifndef S_SPLINT_S
250         digitalWrite(RED_LED, LED_ON);
251         #endif
252         if(write_gpspaket(fd, all_files[2], max_freq, GPS_FREQ,
253                           &loop_counter_gps, &points_gps,
254                           index_fastest) == EXIT_FAILURE)
255         {
256             save_error("->write_gpspaket()\n");
257             break;
258         }
259         #ifndef S_SPLINT_S
260         digitalWrite(RED_LED, LED_OFF);
261         #endif
262         #ifdef PRINT_TIME
263         clock_gettime(CLOCK_MONOTONIC, &my_toc);
```

```
264         printf("GPS_Gesamt: %lf ms\n", calc_time(my_tic, my_toc));
265
266         clock_gettime(CLOCK_MONOTONIC, &my_tic);
267         #endif
268         if(watchdog_output(&watchdog_out, max_freq, 4,
269                            &loop_counter_watchdog_output) == EXIT_FAILURE)
270         {
271             save_error("->watchdog_output()\n");
272             return EXIT_FAILURE;
273         }
274         #ifdef PRINT_TIME
275         clock_gettime(CLOCK_MONOTONIC, &my_toc);
276         printf("WD_OUT: %lf ms\n", calc_time(my_tic, my_toc));
277
278         clock_gettime(CLOCK_MONOTONIC, &my_tic);
279         #endif
280         if(watchdog_input(all_files[10], &shutdown_counter, max_freq, max_freq,
281                           &loop_counter_watchdog_input) == EXIT_FAILURE)
282         {
283             save_error("->watchdog_input()\n");
284             return EXIT_FAILURE;
285         }
286         #ifdef PRINT_TIME
287         clock_gettime(CLOCK_MONOTONIC, &my_toc);
288         printf("WD_IN: %lf ms\n", calc_time(my_tic, my_toc));
289         #endif
290
291         index_fastest++;
292         #ifndef S_SPLINT_S
293         button = digitalRead(BUTTON_PIN);
294         #endif
295
296         clock_gettime(CLOCK_MONOTONIC, &end);
297         if(match_while_freq(begin, end, max_freq) == EXIT_FAILURE)
298         {
299             save_error("->match_while_freq()\n");
300             break;
301         }
302         #ifdef LOGGING_DAUER
303         if(points_gps == LOGGING_DAUER)
304         {
305             save_error("Abbruch durch User (define LOGGING_DAUER)\n");
306             break;
307         }
308         #endif
```

```
309     }
310     read_full_buffer(fd);
311
312     if(snprintf(while_loop_wrong_string, 60 + 1, "Anzahl an zeitlich falschen"
313                 " while-Durchläufen: %d\n", while_loop_wrong) <= 0)
314     {
315         save_error(strerror(errno));
316         save_error("->snprintf(while_loop_wrong)");
317         return EXIT_FAILURE;
318     }
319     save_error(while_loop_wrong_string);
320
321     if(write_points(all_files[0], points_gps) == EXIT_FAILURE)
322     {
323         save_error("->write_points()\n");
324         return EXIT_FAILURE;
325     }
326     (void) printf("\n-------- LOGGING ABGESCHLOSSEN-----------\n");
327     (void) printf("Dateiverzeichnis: %s\n", fdir);  //aktuelles Verzeichnis.
328     #ifndef S_SPLINT_S
329     digitalWrite(RED_LED, LED_ON);
330     #endif
331     #ifndef TEST_BUFFER
332         if(GPS_reset(fd) == -1)
333         {
334             (void) printf("GPS Venus reset\t\t\t\t\t\t\t[  "err" ]\n");
335             save_error("Fehler");
336             save_error("->GPS_reset()\n");
337             return EXIT_FAILURE;
338         }
339         else {(void) printf("GPS Venus reset\t\t\t\t\t\t\t[  "ok"  ]\n");}
340     #endif
341     (void) heightClose();
342     if(close_all_files(all_files) == EXIT_FAILURE)
343     {
344         save_error("->close_all_files()\n");
345         return EXIT_FAILURE;
346     }
347     if(execute_system_calls(&shutdown_counter) == EXIT_FAILURE)
348     {
349         save_error("->execute_system_calls()\n");
350         return EXIT_FAILURE;
351     }
352     #ifndef TEST_BUFFER
353     if(execute_shutdown(button, &shutdown_counter) == EXIT_FAILURE)
```

```
354     {
355         save_error("->execute_shutdown()\n");
356         return EXIT_FAILURE;
357     }
358     #endif
359     if(fclose(fp_ERROR) == EOF)
360     {
361         (void) printf("%s", strerror(errno));
362         (void) printf("->fclose(fp_ERROR)");
363         return EXIT_FAILURE;
364     }
365     #ifndef S_SPLINT_S
366     digitalWrite(RED_LED, LED_OFF);
367     #endif
368     (void) printf("\n################################################\n"
369             "GPS_Logger beendet----------\n"
370             "################################################\n");
371 
372     //while(1 == 1);
373     return EXIT_SUCCESS;
374 }
```

B.1.2. Datalogger.c

Die aus der Quelle [1] übernommenen Programmabschnitte wurden gekennzeichnet.

```
1 
2 #include <stdio.h>
3 #include <stdlib.h>
4 #include <stdint.h>
5 #include <string.h>
6 #include <unistd.h>
7 #include <errno.h>
8 #include <termios.h>
9 #include <fcntl.h>
10 #include <unistd.h>
11 #include <sys/stat.h>
12 #include <time.h>
13 #include <sys/time.h>
14 #include <math.h>
15 #include <linux/i2c-dev.h>
16 #include <sys/ioctl.h>
17 #include <signal.h>
18 
19 #include "DataLogger.h"
```

```
20 #ifndef S_SPLINT_S
21 #include "wiringPi.h"
22 #endif
23 #include "global.h"
24 #include "ErrorStorage.h"
25 #ifndef S_SPLINT_S
26 #include "adafruit_imu.h"
27 #endif
28 #include "LowFuncLogger.h"
29
30 /*@unused@*/static void watchdog_ISR(void);
31 /*@unused@*/static void tacho_ISR(void);
32 /*------------------------------------------------------------------------------
33 Erstellt Folgeverzeichniss beginnend mit /Data000.
34 Kein save_error(). ERROR_STORAGE.txt noch nicht erstellt.
35 (Erstellt von Mario Wegner)
36
37 Parameter: *fdir    Start-Verzeichniss.
38
39 Return: EXIT_SUCCESS, wenn erfolgreich.
40         EXIT_FAILURE, wenn maxinale Verzeichnissanzahl erreicht.
41 ------------------------------------------------------------------------------*/
42 int makedir(char *fdir)
43 {
44     size_t length = strlen(fdir) - 2;
45
46     (void) printf("\nErstelle neues Verzeichnis:\n");
47     while(mkdir(fdir, 0777) == -1)
48     //wenn Verzeichniss(Data000) schon existiert, addiere 1.
49     {
50         if(fdir[length] == '9')
51         {
52             fdir[length] = '0';
53             if(fdir[length - 1] == '9')
54             {
55                 fdir[length - 1] = '0';
56                 if(fdir[length - 2] == '9')
57                 {
58                     (void) printf("Maximale Anzahl an Verzeichnissen erreicht");
59                     return EXIT_FAILURE;
60                 }
61                 else
62                 {
63                     fdir[length - 2]++;
64                 }
```

```
65              }
66              else
67              {
68                  fdir[length - 1]++;
69              }
70          }
71          else
72          {
73              fdir[length]++;
74          }
75      }
76      (void) printf("\t%s\n", fdir);
77
78      return EXIT_SUCCESS;
79  }
80
81  /*-------------------------------------------------------------------------
82  Öffnet und erstellt Datei.
83  Kein save_error(). ERROR_STORAGE.txt noch nicht erstellt.
84  (Erstellt von Mario Wegner)
85
86  Parameter: *fdir        Wo soll die Datei gespeichert werden?
87             *filename    Dateiname.
88             *mode        In welchem Modus soll die Datei geöffnet werden?
89                          "w" schreiben, "wb" schreiben binär.
90
91  Return: Pointer auf Datei, wenn erfolgreich.
92          NULL, wenn malloc-Fehler oder Datei konnte nicht geoeffnet werden.
93  -------------------------------------------------------------------------*/
94  FILE *openfile(char *fdir, char *filename, char *mode)
95  {
96      FILE *fp;
97      char *filepath;
98
99      filepath = concat(fdir, filename);
100     if(filepath == NULL)
101     {
102         (void) printf("->concat()");
103         return NULL;
104     }
105     fp = fopen(filepath, mode);
106     if(fp == NULL)
107     {
108         (void) printf("%s", strerror(errno));
109         (void) printf("->fopen()");
```

```
110         free(filepath);
111         filepath = NULL;
112         return NULL;
113     }
114     free(filepath);
115     filepath = NULL;
116 
117     return fp;
118 }
119 
120 /*---------------------------------------------------------------------------
121 Öffnet alle FILES.
122 Kein save_error(). ERROR_STORAGE.txt noch nicht erstellt.
123 
124 Parameter: *fdir               Wo sollen die Datei geöffnet werden?
125            **all_files   Array mit allen File-Pointern.
126            **file_names  Array mit allen File-Namen-Strings.
127            **file_mode   Array mit allen File-Modi-Strings.
128 
129 Return: EXIT_SUCCESS, wenn erfolgreich.
130         EXIT_FAILURE, wenn openfile() fehlschlägt.
131 ---------------------------------------------------------------------------*/
132 int open_all_files(char *fdir, FILE **all_files, char **file_names,
133                    char **file_mode)
134 {
135     int i = 0;
136 
137     for(i = 0; i < MAX_FILES; i++)
138     {
139         if((all_files[i] = openfile(fdir, file_names[i], file_mode[i])) == NULL)
140         {
141             (void) printf("->openfile()");
142             return EXIT_FAILURE;
143         }
144     }
145 
146     return EXIT_SUCCESS;
147 }
148 
149 /*---------------------------------------------------------------------------
150 Initialisierung der GPIO plus Interrupt Service Routinen (ISR).
151 (Erstellt von Mario Wegner)
152 
153 Parameter:  void
154 
```

```
155 Return: EXIT_SUCCESS, wenn erfolgreich.
156         EXIT_FAILURE, wenn wiringPISetup() oder wiringPiISR() fehlschlägt.
157 ------------------------------------------------------------------------------*/
158 int initializes_gpio(void)
159 {
160     (void) printf("\nInitialisiere GPIO:\n");
161 
162     #ifndef S_SPLINT_S
163     if (wiringPiSetup() < 0)
164     {
165         (void) printf("\twiringPiSetup()\t\t\t\t\t\t[  "err" ]\n");
166         save_error(strerror(errno));
167         save_error("->wiringPiSetup()");
168         return EXIT_FAILURE;
169     }
170     else {(void) printf("\twiringPiSetup()\t\t\t\t\t\t[  "ok"  ]\n");}
171     pinMode(STOP_PIN, INPUT);         // Softwarestop.
172     pinMode(25, OUTPUT);       //25 ist mit 5 verbunden über Kabel.
173     digitalWrite(25, LOW);
174 
175     pinMode(GREEN_LED, OUTPUT);      //LED.
176     pinMode(RED_LED, OUTPUT);
177     digitalWrite(GREEN_LED, LED_OFF);
178     digitalWrite(RED_LED, LED_OFF);
179 
180     pinMode(WD_OUTPUT, OUTPUT);      //Watchdog Pin.
181     pinMode(21, OUTPUT);
182     pinMode(22, OUTPUT);
183     digitalWrite(21, LOW);
184     digitalWrite(22, HIGH);
185 
186     digitalWrite(WD_OUTPUT, LOW);      //Lebenszeichen.
187     milli_sleep(250.0);
188     digitalWrite(WD_OUTPUT, HIGH);
189     milli_sleep(250.0);
190     digitalWrite(WD_OUTPUT, LOW);
191 
192     pinMode(DOOR_PIN, INPUT);         // Door.
193 
194     if (wiringPiISR(TACHO_PIN, INT_EDGE_RISING, &tacho_ISR) < 0)
195     {
196         (void) printf("\tInterrupt Service Routine für Tacho-Pulse einrichten"
197                "\t[  "err" ]\n");
198         save_error(strerror(errno));
199         save_error("->wiringPiISR(tacho)");
```

```
200         return EXIT_FAILURE;
201     }
202     else
203     {
204       (void) printf("\tInterrupt Service Routine für Tacho-Pulse einrichten"
205                   " \t[  "ok"  ]\n");
206     }
207     if (wiringPiISR(WD_INPUT, INT_EDGE_RISING, &watchdog_ISR) < 0)
208     {
209         (void)printf("\tInterrupt Service Routine für Wachtdog-Pulse einrichten"
210                "  [  "err" ]\n");
211         save_error(strerror(errno));
212         save_error("->watchdog_ISR(Watchdog)");
213         return EXIT_FAILURE;
214     }
215     else
216     {
217       (void)printf("\tInterrupt Service Routine für Watchdog-Pulse einrichten"
218                   " [  "ok"  ]\n");
219     }
220     pinMode(BUTTON_PIN, INPUT);
221     //HW Taster. Gedrückt gleich 0, nicht gedrückt gleich 1.
222     pullUpDnControl(BUTTON_PIN, PUD_UP);  // enable pull-up resistor.
223     #endif
224     (void) printf("Initialisiere GPIO abgeschlossen.\n");
225 
226     return EXIT_SUCCESS;
227 }
228 
229 /*------------------------------------------------------------------------------
230 Watchdog-Pulse einlesen. Durch Zeitmessung zw. Pulsen wird Sync erkannt.
231 Zwischen zwei Syncs werden Pulse gezählt und entsprechend
232 (>3 : Akku/<=3 : Info) in globalen Variablen abgespeichert.
233 
234 Parameter: void
235 
236 Return: void
237 ------------------------------------------------------------------------------*/
238 static void watchdog_ISR(void)
239 {
240     struct timespec current_time = start_wd;
241     struct timespec old_time = start_wd;
242     double pulse_to_pulse_time = -1;
243 
244     (void) usleep(50);
```

```
245     #ifndef S_SPLINT_S
246     if(digitalRead(WD_INPUT) != 1)
247     {
248         return;
249     }
250     #endif
251     watchdog_pulse++;
252     clock_gettime(CLOCK_MONOTONIC, &current_time);
253     start_wd = current_time;
254     pulse_to_pulse_time = calc_time(old_time, current_time);
255     if(pulse_to_pulse_time >= ORCSynclength + ORCPulselength
256        - TRINKET_TIME_ERROR)
257     {
258         watchdog_pulse = watchdog_pulse - 2;        //sync pulse abziehen.
259         if(watchdog_pulse > 3)
260         {
261             global_akku_spannung = watchdog_pulse - 3;
262         }
263         else if((watchdog_pulse <= 3) && (watchdog_pulse >= 0))
264         {
265             global_netz_shutdown = watchdog_pulse;
266         }
267         else
268         {
269             save_error("Negative watchdog_pulse!\n");
270         }
271         watchdog_pulse = 0;
272     }
273     #ifdef PRINT_WD_ISR
274         (void) printf("-------------------------------\n");
275         (void) printf("watchdog_pulse: %d, pulse_to_pulse_time: %f\n",
276                watchdog_pulse, pulse_to_pulse_time);
277         if(pulse_to_pulse_time >= ORCSynclength + ORCPulselength
278            - TRINKET_TIME_ERROR)
279         {
280             (void) printf("Sync erkannt\n");
281         }
282         else if(pulse_to_pulse_time >= (8 * ORCPulselength))
283         {
284             (void) printf("Info-Lücke (0 pulse) erkannt\n");
285         }
286         else if(pulse_to_pulse_time >= (6 * ORCPulselength))
287         {
288             (void) printf("Info-Lücke (1 pulse) erkannt\n");
289         }
```

```
290         else if(pulse_to_pulse_time >= (2 * ORCPulselength))
291         {
292            (void) printf("Pulse erkannt\n");
293         }
294         else
295         {
296             (void) printf("unbekannte Pulslänge!\n");
297         }
298         (void) printf("+++++++++++++++++++++++++++++++++++++++++++++\n");
299     #endif
300 }
301
302 /*------------------------------------------------------------------------------
303 Um die Interrupt Service Routine für den Watchdog zu testen,
304 muss der Software-Stop-Output (25) mit dem Watchdog-Input (24)
305 verbunden werden. Au ß erdem muss in DataLogger.h "#define WATCHDOG_TEST"
306 gesetzt werden.
307 Testfunktion geht jeden Zustand einmal durch.
308
309 Parameter: void
310
311 Return: void
312 ------------------------------------------------------------------------------*/
313 void test_watchdog_ISR(FILE *fp_WATCHDOG)
314 {
315     int lfv_pulse = 0;
316     int lfv_signal = 0;
317     /*@unused@*/double sync_time = ORCSynclength;
318     /*@unused@*/double pulse_time = ORCPulselength;
319
320     int shutdown_counter = 0;
321     int max_freq = 1;
322     int loop_counter_watchdog_input = 0;
323
324     # ifndef S_SPLINT_S
325     digitalWrite(25, 1);        //für Initialisierung.
326     milli_sleep(pulse_time);
327     digitalWrite(25, 0);
328     milli_sleep(pulse_time);
329
330     digitalWrite(25, 1);        //Sync.
331     milli_sleep(sync_time);
332     digitalWrite(25, 0);
333     milli_sleep(pulse_time);
334
```

```
335     digitalWrite(25, 1);        //Sync.
336     milli_sleep(pulse_time);
337     digitalWrite(25, 0);
338     milli_sleep(pulse_time);
339     #endif
340     (void) printf("\nWatchdog_ISR_Test beginnt!\n");
341     for(lfv_signal = 0; lfv_signal < 24; lfv_signal++)  //Sync Pulse(0-24) Sync.
342     {
343         if(lfv_signal < 4)
344         {
345             for(lfv_pulse = 0; lfv_pulse < lfv_signal ; lfv_pulse++)   //Pulse.
346             {
347                 #ifndef S_SPLINT_S
348                 digitalWrite(25, 1);
349                 milli_sleep(pulse_time);
350                 digitalWrite(25, 0);
351                 milli_sleep(pulse_time);
352                 #endif
353             }
354         }
355         else
356         {
357             for(lfv_pulse = 0; lfv_pulse < 3 ; lfv_pulse++)     //Pulse.
358             {
359                 #ifndef S_SPLINT_S
360                 digitalWrite(25, 1);
361                 milli_sleep(pulse_time);
362                 digitalWrite(25, 0);
363                 milli_sleep(pulse_time);
364                 #endif
365             }
366         }
367         #ifndef S_SPLINT_S
368         digitalWrite(25, 1);        //Sync.
369         milli_sleep(sync_time);
370         digitalWrite(25, 0);
371         milli_sleep(pulse_time);
372 
373         digitalWrite(25, 1);        //Sync.
374         milli_sleep(pulse_time);
375         digitalWrite(25, 0);
376         milli_sleep(pulse_time);
377         #endif
378         for(lfv_pulse = 0; lfv_pulse < lfv_signal + 4; lfv_pulse++)     //Pulse.
379         {
```

```
380             #ifndef S_SPLINT_S
381             digitalWrite(25, 1);
382             milli_sleep(pulse_time);
383             digitalWrite(25, 0);
384             milli_sleep(pulse_time);
385             #endif
386         }
387         #ifndef S_SPLINT_S
388         digitalWrite(25, 1);        //Sync.
389         milli_sleep(sync_time);
390         digitalWrite(25, 0);
391         milli_sleep(pulse_time);
392 
393         digitalWrite(25, 1);        //Sync.
394         milli_sleep(pulse_time);
395         digitalWrite(25, 0);
396         milli_sleep(pulse_time);
397         #endif
398         (void) watchdog_input(fp_WATCHDOG, &shutdown_counter, max_freq, 1,
399                               &loop_counter_watchdog_input);      //Auswertung.
400 
401     }
402 
403 }
404 
405 /*------------------------------------------------------------------------------
406 Zählt bei Oedemetrieflange globale Variable tick hoch
407 und setzt tacho_timeout_counter auf 0, da sich das Fahrzeug noch beweget.
408 (Erstellt von Mario Wegner)
409 
410 Parameter: void
411 
412 Return: void
413 ------------------------------------------------------------------------------*/
414 static void tacho_ISR(void)
415 {
416     (void) usleep(25);
417 
418     #ifndef S_SPLINT_S
419     if(digitalRead(TACHO_PIN) != 1)
420     {
421         return;
422     }
423     #endif
424     tick++;
```

```
425     tacho_timeout_counter = 0;
426
427 }
428
429 /*------------------------------------------------------------------------------
430 Wartet so lange bis Bus fährt, Tür auf/zu geht
431 oder HWTaster gedrückt wird. (Erstellt von Mario Wegner)
432
433 Parameter: void
434
435 Return: void
436 ------------------------------------------------------------------------------*/
437 void waitForStart(void)
438 {
439     int doorPosition, lastDoorPosition = -1;
440
441     (void) printf("\nWarte auf Start...");
442
443     #ifndef S_SPLINT_S
444     doorPosition = digitalRead(DOOR_PIN);
445     #endif
446     #ifdef S_SPLINT_S
447     doorPosition = -1;
448     #endif
449     lastDoorPosition = doorPosition;
450     while((tacho_timeout_counter >= TACHO_TIMEOUT) &&
451           (doorPosition == lastDoorPosition)
452           #ifndef S_SPLINT_S
453           && (digitalRead(BUTTON_PIN) == 1)
454           #endif
455          )
456     {
457         #ifndef S_SPLINT_S
458         digitalWrite(WD_OUTPUT, LOW);       //Lebenszeichen
459         milli_sleep(250.0);
460         digitalWrite(WD_OUTPUT, HIGH);
461         milli_sleep(250.0);
462         digitalWrite(WD_OUTPUT, LOW);
463         #endif
464         lastDoorPosition = doorPosition;
465         #ifndef S_SPLINT_S
466         doorPosition = digitalRead(DOOR_PIN);
467         #endif
468         //nach einer gewissen Zeit rausspringen?
469     }
```

```
470
471     (void) printf("\t\t\t\t\t\t[  "ok"  ]\n");
472 }
473
474 /*-------------------------------------------------------------------------
475 Speichert Zeit (Format: YYYYMMDDhhmmss) in binär Datei.
476 (Erstellt von Mario Wegner)
477
478 Parameter: *fp_TIME    In welcher binär Datei soll die Zeit gespeichert werden?
479
480 Return: EXIT_SUCCESS, wenn erfolgreich.
481         EXIT_FAILURE, wenn time() oder fwrite() fehlschlägt.
482 -------------------------------------------------------------------------*/
483 int write_time(FILE *fp_TIME)
484 {
485     time_t rawtime;
486     struct tm *timeinfo;
487     double temptime;
488
489     (void) printf("\nStartzeit in TIME.bin speichern:\n");
490     if(time(&rawtime) == -1)
491     {
492         save_error(strerror(errno));
493         save_error("->time()");
494         return EXIT_FAILURE;
495     }
496     timeinfo = localtime(&rawtime);
497     (void) printf("\t%s",asctime(timeinfo));
498
499     temptime = (double)(timeinfo->tm_year + 1900);
500     if(fwrite(&temptime, sizeof(double), 1, fp_TIME) != 1)
501     {
502         save_error(strerror(errno));
503         save_error("->fwrite(year)");
504         return EXIT_FAILURE;
505     }
506     temptime = (double)(timeinfo->tm_mon + 1);   // month [0 - 11].
507     if(fwrite(&temptime, sizeof(double), 1, fp_TIME) != 1)
508     {
509         save_error(strerror(errno));
510         save_error("->fwrite(year)");
511         return EXIT_FAILURE;
512     }
513     temptime = (double)timeinfo->tm_mday;
514     if(fwrite(&temptime, sizeof(double), 1, fp_TIME) != 1)
```

```
515     {
516         save_error(strerror(errno));
517         save_error("->fwrite(year)");
518         return EXIT_FAILURE;
519     }
520     temptime = (double)timeinfo->tm_hour;
521     if(fwrite(&temptime, sizeof(double), 1, fp_TIME) != 1)
522     {
523         save_error(strerror(errno));
524         save_error("->fwrite(year)");
525         return EXIT_FAILURE;
526     }
527     temptime = (double)timeinfo->tm_min;
528     if(fwrite(&temptime, sizeof(double), 1, fp_TIME) != 1)
529     {
530         save_error(strerror(errno));
531         save_error("->fwrite(year)");
532         return EXIT_FAILURE;
533     }
534     temptime = (double)timeinfo->tm_sec;
535     if(fwrite(&temptime, sizeof(double), 1, fp_TIME) != 1)
536     {
537         save_error(strerror(errno));
538         save_error("->fwrite(year)");
539         return EXIT_FAILURE;
540     }
541
542     return EXIT_SUCCESS;
543 }
544
545 /*------------------------------------------------------------------------------
546 Lie ß t solange vom Venus-Buffer bis Venus Initialisierung
547 und erstes $ Zeichen eingelesen wurde.
548
549 Parameter: fd     File_Descriptor, der auf den Venus-Buffer zeigt.
550
551 Return: EXIT_SUCCESS, wenn erfolgreich.
552         EXIT_FAILURE, wenn Fehler beim Auslesen des Venus-Buffers.
553 ------------------------------------------------------------------------------*/
554 int read_venus_init(int fd)
555 {
556     char init_void[300 + 1] = "";
557
558     #ifdef TEST_BUFFER
559         if(read_fd_till_string(fd, "710\n$",
```

```
560                                 init_void, 300 + 1) == EXIT_FAILURE)
561         {
562             save_error("->read_fd_till_string(init_void)");
563             return EXIT_FAILURE;
564         }
565     #endif
566     #ifndef TEST_BUFFER
567         if(read_fd_till_string(fd, "710\r\n$",
568                                 init_void, 300 + 1) == EXIT_FAILURE)
569         {
570             save_error("->read_fd_till_string(init_void)");
571             return EXIT_FAILURE;
572         }
573     #endif
574     return EXIT_SUCCESS;
575 }
576
577 /*-------------------------------------------------------------------------
578 Schreibt Timestamp + Zählindex + Tür-Signal in Datei.
579 Format: Closed 2016.05.19.13.17.31.311:3_0\n$
580         Opened 2016.05.19.13.17.31.311:3_1\n$
581         Error  2016.05.19.13.17.31.311:3_-\n$
582
583 Parameter: *fp_DOOR           Datei, in die das Tür-Signal geschrieben wird.
584            max_freq           Gewünschte Auslese-Frequenz des
585                               schnellsten Sensors.
586            sensor_hz          Frequenz, mit der in die Datei geschrieben wird.
587            *loop_counter_door Sensor überspringen oder auslesen? Abhängig
588                               von sensor_hz.
589            index_fastest      Zähl-Index der while-Schleife bzw.
590                               des schnellsten Sensors.
591
592 Return: EXIT_SUCCESS, wenn erfolgreich.
593         EXIT_FAILURE, wenn Fehler in Subfunc.
594 -------------------------------------------------------------------------*/
595 int write_door(FILE *fp_DOOR, char *door, int max_freq, int sensor_hz,
596                int *loop_counter_door, unsigned long int index_fastest)
597 {
598     pull_freq(max_freq, sensor_hz, loop_counter_door);
599     if(*loop_counter_door == 1)
600     {
601         char *index_plus_door;
602         int door_status = -1;
603         char timestamp[25] = "";
604
```

```
605         #ifndef S_SPLINT_S
606         door_status = digitalRead(DOOR_PIN);
607         #endif
608         if(door_status == DOOR_CLOSED)
609         {
610             door[0] = '0';
611         }
612         else if(door_status == DOOR_OPEN)
613         {
614             door[0] = '1';
615         }
616         else
617         {
618             door[0] = '-';
619             save_error("Fehler bei door Auslesung");
620             save_error("->write_door()\n");
621         }
622         if(write_timestamp(timestamp) == EXIT_FAILURE)
623         {
624             save_error("->write_timestamp()");
625             return EXIT_FAILURE;
626         }
627         if(fputs(timestamp, fp_DOOR) == EOF)
628         {
629             save_error(strerror(errno));
630             save_error("->fputs(TS)");
631             return EXIT_FAILURE;
632         }
633         if((index_plus_door = add_count_index(index_fastest, door)) == NULL)
634         {
635             save_error("->add_count_index()");
636             return EXIT_FAILURE;
637         }
638         #ifdef PRINT_TO_SCREEN
639             (void) printf("\nDoor: %s", index_plus_door);
640         #endif
641         if(fputs(index_plus_door, fp_DOOR) == EOF)
642         {
643             save_error(strerror(errno));
644             save_error("->fputs(DOOR)");
645             free(index_plus_door);
646             index_plus_door = NULL;
647             return EXIT_FAILURE;
648         }
649         free(index_plus_door);
```

```
650         index_plus_door = NULL;
651     }
652 
653     return EXIT_SUCCESS;
654 }
655 
656 /*------------------------------------------------------------------------------
657 Schreibt Timestamp + Zählindex + globale tick Variable in Datei.
658 Format: 2016.05.19.13.17.31.311:3_lu\n$
659 
660 Globale Variable enthält die bis dato gezählten Odemetrie-Flanken.
661 
662 Parameter: *fp_TACHO           Datei, in die die Odemetrie-Info geschrieben
663                                wird.
664            max_freq            Gewünschte Auslese-Frequenz des
665                                schnellsten Sensors.
666            sensor_hz           Frequenz mit der in die Datei geschrieben wird.
667            *loop_counter_tacho Sensor überspringen oder auslesen? Abhängig
668                                von sensor_hz.
669            index_fastest       Zähl-Index der while-Schlife bzw.
670                                des schnellsten Sensors.
671 
672 Return: EXIT_SUCCESS, wenn erfolgreich.
673         EXIT_FAILURE, wenn Fehler in Subfunc.
674 ------------------------------------------------------------------------------*/
675 int write_tacho(FILE *fp_TACHO, int max_freq, int sensor_hz,
676                 int *loop_counter_tacho, unsigned long int index_fastest)
677 {
678     pull_freq(max_freq, sensor_hz, loop_counter_tacho);
679     if(*loop_counter_tacho == 1)
680     {
681         unsigned long int temp1 = tick;
682         size_t tacho_length;
683         char *tacho;
684         char *index_plus_tacho;
685         char timestamp[25] = "";
686 
687         if((tacho_length = count_digits(temp1)) == 0)
688         {
689             save_error("->count_digits()");
690             return EXIT_FAILURE;
691         }
692         if((tacho = malloc(tacho_length + 1 + 1 + 1)) == NULL)//'\n', '$', '\0'.
693         {
694             save_error(strerror(errno));
```

```
695             save_error("->malloc(tacho)");
696             return EXIT_FAILURE;
697         }
698         if(snprintf(tacho, 10 + 3, "%lu\n$", temp1) <= 0)
699         {
700             save_error(strerror(errno));
701             save_error("->snprintf(tacho)");
702             free(tacho);
703             tacho = NULL;
704             return EXIT_FAILURE;
705         }
706         if(write_timestamp(timestamp) == EXIT_FAILURE)
707         {
708             save_error("->write_timestamp()");
709             free(tacho);
710             tacho = NULL;
711             return EXIT_FAILURE;
712         }
713         if(fputs(timestamp, fp_TACHO) == EOF)
714         {
715             save_error(strerror(errno));
716             save_error("->fputs(TACHO)");
717             free(tacho);
718             tacho = NULL;
719             return EXIT_FAILURE;
720         }
721         if((index_plus_tacho = add_count_index(index_fastest, tacho)) == NULL)
722         {
723             save_error("->add_count_index()");
724             free(tacho);
725             tacho = NULL;
726             return EXIT_FAILURE;
727         }
728         #ifdef PRINT_TO_SCREEN
729             (void) printf("\nTacho: %s", index_plus_tacho);
730         #endif
731         if(fputs(index_plus_tacho, fp_TACHO) == EOF)
732         {
733             save_error(strerror(errno));
734             save_error("->fputs(TACHO)");
735             free(tacho);
736             tacho = NULL;
737             free(index_plus_tacho);
738             index_plus_tacho = NULL;
739             return EXIT_FAILURE;
```

```
740         }
741         free(tacho);
742         tacho = NULL;
743         free(index_plus_tacho);
744         index_plus_tacho = NULL;
745     }
746 
747     return EXIT_SUCCESS;
748 }
749 
750 /*-------------------------------------------------------------------------
751 Schreibt Timestamp + Zählindex + IMU-Werte in Dateien.
752 Passiert in writeX_to_data (adafruit_imu.c).
753 Jeweils für ACC, MAG, GYR, PR_TE.
754 Format Bsp. ACC: 2016.05.19.15.26.55.289:3_   0.000;   0.000;   0.000\n$
755 
756 Parameter: *fd_i2c           File_Descriptor, der auf PR_TE-Sensor zeigt.
757            *fp_ACC           Datei, in die ACC geschrieben wird.
758            *fp_MAG           Datei, in die MAG geschrieben wird.
759            *fp_GYR           Datei, in die GYR geschrieben wird.
760            *fp_PR_TE         Datei, in die PR_TE geschrieben wird.
761            *t_pressure       Druckwert für HeightLib.
762            max_freq          Gewünschte Auslese-Frequenz des
763                              schnellsten Sensors.
764            sensor_hz         Frequenz mit der in die Dateien geschrieben wird.
765            *loop_counter_imu Sensor überspringen oder auslesen? Abhängig
766                              von sensor_hz.
767            index_fastest     Zähl-Index der while-Schlife bzw.
768                              des schnellsten Sensors.
769 
770 Return: EXIT_SUCCESS, wenn erfolgreich.
771         EXIT_FAILURE, wenn IMU-Wert nicht eingelesen werden kann.
772 -------------------------------------------------------------------------*/
773 int write_imu(int fd_i2c, FILE *fp_ACC, FILE *fp_MAG, FILE *fp_GYR,
774               FILE *fp_PR_TE, double *t_pressure, int max_freq,
775               int freq_acc, int freq_mag, int freq_gyr, int freq_pr_te,
776               int *loop_counter_acc, int *loop_counter_mag,
777               int *loop_counter_gyr, int *loop_counter_pr_te,
778               unsigned long int index_fastest)
779 {
780     #ifndef S_SPLINT_S
781     #ifdef PRINT_TIME
782     struct timespec my_tic = {0};
783     struct timespec my_toc = {0};
784     #endif
```

```
785
786     #ifdef PRINT_TIME
787     clock_gettime(CLOCK_MONOTONIC, &my_tic);
788     #endif
789     if(writeACC_to_data(fp_ACC, max_freq, freq_acc, loop_counter_acc,
790                         index_fastest) == EXIT_FAILURE)
791     {
792         save_error("->writeACC_to_data()");
793         return EXIT_FAILURE;
794     }
795     #ifdef PRINT_TIME
796     clock_gettime(CLOCK_MONOTONIC, &my_toc);
797     (void) printf("ACC: %lf ms, ", calc_time(my_tic, my_toc));
798
799     clock_gettime(CLOCK_MONOTONIC, &my_tic);
800     #endif
801     if(writeMAG_to_data(fp_MAG, max_freq, freq_mag, loop_counter_mag,
802                         index_fastest) == EXIT_FAILURE)
803     {
804         save_error("->writeMAG_to_data()");
805         return EXIT_FAILURE;
806     }
807     #ifdef PRINT_TIME
808     clock_gettime(CLOCK_MONOTONIC, &my_toc);
809     (void) printf("MAG: %lf ms, ", calc_time(my_tic, my_toc));
810
811     clock_gettime(CLOCK_MONOTONIC, &my_tic);
812     #endif
813     if(writeGYR_to_data(fp_GYR, max_freq, freq_gyr, loop_counter_gyr,
814                         index_fastest) == EXIT_FAILURE)
815     {
816         save_error("->writeGYR_to_data()");
817         return EXIT_FAILURE;
818     }
819     #ifdef PRINT_TIME
820     clock_gettime(CLOCK_MONOTONIC, &my_toc);
821     (void) printf("GYR: %lf ms, ", calc_time(my_tic, my_toc));
822
823     clock_gettime(CLOCK_MONOTONIC, &my_tic);
824     #endif
825     if(writePR_TE_to_data(fd_i2c, fp_PR_TE, t_pressure, max_freq, freq_pr_te,
826                           loop_counter_pr_te, index_fastest) == EXIT_FAILURE)
827     {
828         save_error("->writePR_TR_to_data()");
829         return EXIT_FAILURE;
```

```
830     }
831     #ifdef PRINT_TIME
832     clock_gettime(CLOCK_MONOTONIC, &my_toc);
833     (void) printf("PR_TE: %lf ms\n", calc_time(my_tic, my_toc));
834     #endif
835     #endif
836     //für Splint-Test
837     #ifdef S_SPLINT_S
838     char fake_string[4 + 1] = "fake";
839     fd_i2c = 0;
840     (void) fputs(fake_string, fp_ACC);
841     (void) fputs(fake_string, fp_MAG);
842     (void) fputs(fake_string, fp_GYR);
843     (void) fputs(fake_string, fp_PR_TE);
844     max_freq = 0;
845     loop_counter_acc = &freq_acc;
846     loop_counter_mag = &freq_mag;
847     loop_counter_gyr = &freq_gyr;
848     loop_counter_pr_te = &freq_pr_te;
849     index_fastest = 0;
850     #endif
851
852     return EXIT_SUCCESS;
853 }
854
855 /*---------------------------------------------------------------------------
856 Lie ß t GPS-Paket vom Venus-Buffer und schreibt es in Datei.
857 Es wird immer von $ bis $ eingelsen.
858 Format: GGA 2016.05.19.15.35.26.322:3_GPGGA,140732.201,5333.9807,N,01004.5735,
859             E,0,00,0.0,81.3,M,0.0,M,,0000*56\n$
860         RMC 2016.05.19.15.35.26.322:3_GPRMC,140732.201,V,5333.9807,N,01004.5735,
861             E,000.0,000.0,190516,,,N*7D\n$
862
863 Parameter: fd                File_Descriptor, der auf den Buffer zeigt.
864            *fp_GGA           Datei, in die das GPS_Paket geschrieben wird.
865            max_freq          Gewünschte Auslese-Frequenz des
866                              schnellsten Sensors.
867            sensor_hz         Frequenz mit der in die Datei geschrieben wird.
868            *loop_counter_gps Sensor überspringen oder auslesen? Abhängig
869                              von sensor_hz.
870            *points_gps       Anzahl an Sensorauslesung/GPS-Paketen.
871            index_fastest     Zähl-Index der while-Schlife bzw.
872                              des schnellsten Sensors.
873
874 Return: EXIT_SUCCESS, wenn erfolgreich.
```

```
875         EXIT_FAILURE, wenn Fehler beim Auslesen des Venus-Buffers
876                       oder Fehler beim Index anhängen.
877 ------------------------------------------------------------------------------*/
878 int write_gpspaket(int fd, FILE *fp_GGA, int max_freq, int sensor_hz,
879                    int *loop_counter_gps, unsigned long int *points_gps,
880                    unsigned long int index_fastest)
881 {
882     pull_freq(max_freq, sensor_hz, loop_counter_gps);
883     if(*loop_counter_gps == 1)
884     {
885         char gga_info[200 + 1] = "";    //max_length = 80, variable.
886         char rmc_info[200 + 1] = "";    //max_length = 72, const.
887         char *index_plus_gga;
888         char *index_plus_rmc;
889         char timestamp[25] = "";
890
891         #ifdef PRINT_TIME
892         struct timespec my_tic = {0};
893         struct timespec my_toc = {0};
894
895         clock_gettime(CLOCK_MONOTONIC, &my_tic);
896         #endif
897         if(read_fd_till_string(fd, "$", gga_info, 200 + 1) == EXIT_FAILURE)
898         {
899             save_error("->read_fd_till_string(gga_info)");
900             return EXIT_FAILURE;
901         }
902         #ifdef PRINT_TIME
903         clock_gettime(CLOCK_MONOTONIC, &my_toc);
904         printf("GGA: %lf ms, ", calc_time(my_tic, my_toc));
905
906         clock_gettime(CLOCK_MONOTONIC, &my_tic);
907         #endif
908         if(read_fd_till_string(fd, "$", rmc_info, 200 + 1) == EXIT_FAILURE)
909         {
910             save_error("->read_fd_till_string(rmc_info)");
911             return EXIT_FAILURE;
912         }
913         #ifdef PRINT_TIME
914         clock_gettime(CLOCK_MONOTONIC, &my_toc);
915         printf("RMC: %lf ms\n", calc_time(my_tic, my_toc));
916         #endif
917         if(write_timestamp(timestamp) == EXIT_FAILURE)
918         {
919             save_error("->write_timestamp()");
```

```
920             return EXIT_FAILURE;
921         }
922         if((index_plus_gga = add_count_index(index_fastest, gga_info)) == NULL)
923         {
924             save_error("->add_count_index(gga)");
925             return EXIT_FAILURE;
926         }
927         if((index_plus_rmc = add_count_index(index_fastest, rmc_info)) == NULL)
928         {
929             save_error("->add_count_index(rmc)");
930             free(index_plus_gga);
931             index_plus_gga = NULL;
932             return EXIT_FAILURE;
933         }
934         #ifdef PRINT_TO_SCREEN
935             (void) printf("\n%s", index_plus_gga);
936             (void) printf("%s", index_plus_rmc);
937         #endif
938         if(fputs(timestamp, fp_GGA) == EOF)
939         {
940             save_error(strerror(errno));
941             save_error("->fputs(TS)");
942             free(index_plus_gga);
943             index_plus_gga = NULL;
944             free(index_plus_rmc);
945             index_plus_rmc = NULL;
946             return EXIT_FAILURE;
947         }
948         if(fputs(index_plus_gga, fp_GGA) == EOF)
949         {
950             save_error(strerror(errno));
951             save_error("->fputs(GGA)");
952             free(index_plus_gga);
953             index_plus_gga = NULL;
954             free(index_plus_rmc);
955             index_plus_rmc = NULL;
956             return EXIT_FAILURE;
957         }
958         if(fputs(timestamp, fp_GGA) == EOF)
959         {
960             save_error(strerror(errno));
961             save_error("->fputs(TS)");
962             free(index_plus_gga);
963             index_plus_gga = NULL;
964             free(index_plus_rmc);
```

```
965             index_plus_rmc = NULL;
966             return EXIT_FAILURE;
967         }
968         if(fputs(index_plus_rmc, fp_GGA) == EOF)
969         {
970             save_error(strerror(errno));
971             save_error("->fputs(RMC)");
972             free(index_plus_gga);
973             index_plus_gga = NULL;
974             free(index_plus_rmc);
975             index_plus_rmc = NULL;
976             return EXIT_FAILURE;
977         }
978 
979         free(index_plus_gga);
980         index_plus_gga = NULL;
981         free(index_plus_rmc);
982         index_plus_rmc = NULL;
983 
984         tacho_timeout_counter++;
985         (*points_gps)++;
986     }
987 
988     return EXIT_SUCCESS;
989 }
990 
991 /*-----------------------------------------------------------------------------
992 Wechselt mit 4Hz den Pegel von der Kommunikation zum Watchdog.
993 (Lebenszeichen)
994 4Hz -> Puls/Pause = 250ms.
995 
996 Parameter: *watchdog_out       Bestimmt Output. Wechselt von 1 auf 0 / 0 auf 1.
997            max_freq            Gewünschte Auslese-Frequenz des
998                                schnellsten Sensors.
999            sensor_hz           Frequenz mit der der Pegel-Wechsel
1000                               durchgeführt wird.
1001           *loop_counter_imu   Sensor überspringen oder auslesen? Abhängig
1002                               von sensor_hz.
1003 
1004 Return: EXIT_SUCCESS, wenn erfolgreich.
1005         EXIT_FAILURE, wenn watchdog_out unzulässiger Wert.
1006 -----------------------------------------------------------------------------*/
1007 int watchdog_output(int *watchdog_out, int max_freq, int sensor_hz,
1008                     int *loop_counter_watchdog_output)
1009 {
```

```
1010     pull_freq(max_freq, sensor_hz, loop_counter_watchdog_output);
1011     if(*loop_counter_watchdog_output == 1)
1012     {
1013         if(*watchdog_out == 1)
1014         {
1015             *watchdog_out = 0;
1016         }
1017         else if(*watchdog_out == 0)
1018         {
1019             *watchdog_out = 1;
1020         }
1021         else
1022         {
1023             save_error("watchdog_out unzulässiger Wert");
1024             return EXIT_FAILURE;
1025         }
1026         #ifndef S_SPLINT_S
1027         digitalWrite(WD_OUTPUT, *watchdog_out);
1028         #endif
1029     }
1030 
1031     return EXIT_SUCCESS;
1032 }
1033 
1034 /*-----------------------------------------------------------------------------
1035 Decodiert Information aus globalen Variablen und schreibt diese in Datei.
1036 Format: 2016.05.19.15.35.28.246: Busnetz an | Akku-Spannung:5.100000\n$
1037 
1038 Parameter: *fp_WATCHDOG                Datei, in die die Watchdog Info +
1039                                        Akku-Spannung geschrieben wird.
1040            *shutdown_counter           Shutdown-Signal muss 3 Mal in Folge
1041                                        vorliegen.
1042            max_freq                    Gewünschte Auslese-Frequenz des
1043                                        schnellsten Sensors.
1044            sensor_hz                   Frequenz mit der in die Datei
1045                                        geschrieben wird.
1046            *loop_counter_watchdog_input Sensor überspringen oder auslesen?
1047                                        Abhängig von sensor_hz.
1048 
1049 Return: EXIT_SUCCESS, wenn erfolgreich.
1050         EXIT_FAILURE, wenn Fehler in Subfunc.
1051 -----------------------------------------------------------------------------*/
1052 int watchdog_input(FILE *fp_WATCHDOG, int *shutdown_counter, int max_freq,
1053                    int sensor_hz, int *loop_counter_watchdog_input)
1054 {
```

```
1055    pull_freq(max_freq, sensor_hz, loop_counter_watchdog_input);
1056    if(*loop_counter_watchdog_input == 1)
1057    {
1058        float akku_v = -1.0;
1059        char watchdog_info[42 + 1] = "-1.0";
1060        char info_plus_akku[68 + 1] = "";
1061        char timestamp[25] = "";
1062
1063        if(decode_wd_input(watchdog_info, &akku_v,
1064                           shutdown_counter) == EXIT_FAILURE)
1065        {
1066            save_error("->decode_wd_input()");
1067            return EXIT_FAILURE;
1068        }
1069
1070        if((akku_v >= 3) && !(strcmp(watchdog_info, "-1.0") == 0))
1071        {
1072
1073            global_akku_spannung = -1;
1074            global_netz_shutdown = -1;
1075
1076            if(snprintf(info_plus_akku, 1 + 42 + 18 + 5 + 4,
1077              " %s | Akku-Spannung: %.3fV\n$", watchdog_info, akku_v) <= 0)
1078            {
1079                save_error(strerror(errno));
1080                save_error("->snprintf(watchdog)");
1081                return EXIT_FAILURE;
1082            }
1083            if(write_timestamp(timestamp) == EXIT_FAILURE)
1084            {
1085                save_error("->write_timestamp()");
1086                return EXIT_FAILURE;
1087            }
1088            if(fputs(timestamp, fp_WATCHDOG) == EOF)
1089            {
1090                save_error(strerror(errno));
1091                save_error("->fputs(Watchdog)");
1092                return EXIT_FAILURE;
1093            }
1094            (void) printf("%s", info_plus_akku);
1095            if(fputs(info_plus_akku, fp_WATCHDOG) == EOF)
1096            {
1097                save_error(strerror(errno));
1098                save_error("->fputs(Watchdog)");
1099                return EXIT_FAILURE;
```

```
1100             }
1101         }
1102
1103     }
1104
1105     return EXIT_SUCCESS;
1106 }
1107
1108 /*-----------------------------------------------------------------------------
1109 Wartet, sodass while-Schleife eine bestimmte Anzahl an Durchläufen pro Sec hat.
1110 Falls der Durchlauf zulange gedauert hat, wird nicht gewartet.
1111
1112 Parameter: begin    Startzeit aufgenommen mit clock_gettime().
1113            end      Stopzeit aufgenommen mit clock_gettime().
1114            max_freq Gewünschte Auslese-Frequenz des schnellsten Sensors.
1115
1116 Return: EXIT_SUCCESS, wenn erfolgreich.
1117         EXIT_FAILURE, wenn Fehler in calc_time().
1118 -----------------------------------------------------------------------------*/
1119 int match_while_freq(struct timespec begin, struct timespec end, int max_freq)
1120 {
1121     double duration, max_duration;
1122
1123     if((duration = calc_time(begin, end)) < 0.0)
1124     {
1125         save_error("->calc_time()");
1126         return EXIT_FAILURE;
1127     }
1128     max_duration = (1.0 / max_freq) * 1000.0;
1129     #if defined(PRINT_TO_SCREEN) || defined(PRINT_TIME)
1130         (void) printf("Sleep %f ms\n", (max_duration - duration));
1131     #endif
1132     if((max_duration - duration) < 0)
1133     {
1134         #if defined(PRINT_TO_SCREEN) || defined(PRINT_TIME)
1135             (void) printf("Ausführung der while-Schleife dauert zu lange!\n");
1136         #endif
1137         while_loop_wrong++;
1138     }
1139     else
1140     {
1141         #if defined(PRINT_TO_SCREEN) || defined(PRINT_TIME)
1142             (void) printf("%f ms + %f ms = %f ms -> %i Hz\n",
1143                    duration, (max_duration - duration), max_duration, max_freq);
1144         #endif
```

```
1145            milli_sleep(max_duration - duration - 0.5);
1146        }
1147        #if defined(PRINT_TO_SCREEN) || defined(PRINT_TIME)
1148        (void) printf("-----------------------------------------------------"
1149                      "--------------------\n");
1150        #endif
1151        return EXIT_SUCCESS;
1152    }
1153
1154    /*-------------------------------------------------------------------------
1155    Speichert übrigen Buffer-Inhalt in ERROR_LOGGER.txt.
1156
1157    Parameter: fd   File_Descriptor, der auf den Venus-Buffer zeigt.
1158
1159    Return: void
1160    -------------------------------------------------------------------------*/
1161    void read_full_buffer(int fd)
1162    {
1163        char buffer [2] = "", full_string[5000] = "";
1164        int i = 0;
1165
1166        while(read(fd, buffer, 1) == 1)
1167        {
1168            full_string[i] = buffer[0];
1169            i++;
1170        }
1171        full_string[i] = '\0';
1172        save_error(full_string);
1173        save_error(" :übrig gebliebender Buffer-Inhalt\n");
1174    }
1175
1176    /*-------------------------------------------------------------------------
1177    Schreibt Anzahl an GPS-Paketen (Messpunkte) in Datei.
1178
1179    Parameter: *fp_POINTS   Datei, in die die Anzahl geschrieben wird.
1180               points_gps   Variable, in der die Anzahl an
1181                            Sensorauslesung/GPS-Paketen steht.
1182
1183    Return: EXIT_SUCCESS, wenn erfolgreich.
1184            EXIT_FAILURE, wenn Schreiben fehlschlägt.
1185    -------------------------------------------------------------------------*/
1186    int write_points(FILE *fp_POINTS, unsigned long int points_gps)
1187    {
1188        if(fwrite(&points_gps, sizeof(unsigned long int), 1, fp_POINTS) != 1)
1189        {
```

```
1190         (void) printf("\nAnzahl GPS-Messung (%lu) in POINTS.bin speichern"
1191                "  \t\t[  "err" ]\n", points_gps);
1192         save_error(strerror(errno));
1193         save_error("->fwrite()");
1194         return EXIT_FAILURE;
1195     }
1196     else {(void) printf("\nAnzahl GPS-Messung (%lu) in POINTS.bin speichern"
1197                  "  \t\t[  "ok"  ]\n", points_gps);}
1198 
1199     return EXIT_SUCCESS;
1200 }
1201 
1202  /*-------------------------------------------------------------------------
1203 Schließt alle Files.
1204 
1205 Parameter: *fp  alle Files.
1206 
1207 Return: EXIT_SUCCESS, wenn erfolgreich.
1208         EXIT_FAILURE, wenn fclose() fehlschlägt.
1209 -------------------------------------------------------------------------*/
1210 int close_all_files(FILE **all_files)
1211 {
1212     int i = 0;
1213 
1214     for(i = 0; i < MAX_FILES; i++)
1215     {
1216         if(fclose(all_files[i]) == EOF)       //Schließe alle Files.
1217         {
1218             save_error(strerror(errno));
1219             save_error("->fclose()");
1220             return EXIT_FAILURE;
1221         }
1222     }
1223 
1224     return EXIT_SUCCESS;
1225 }
1226 
1227 /*-------------------------------------------------------------------------
1228 Führt Systemaufrufe durch. GPS_Calc wird nicht ausgeführt,
1229 wenn der Shutdown-Befehl vom Watchdog kommt.
1230 Deaktiviert ISR.
1231 
1232 Parameter:*shutdown_counter Zeigt an, ob der shutdwon-Befehl vom WD kommt.
1233 
1234 Return: EXIT_SUCCESS, wenn erfolgreich.
```

```
1235         EXIT_FAILURE, wenn system() fehlschlägt.
1236 ------------------------------------------------------------------------------*/
1237 int execute_system_calls(int *shutdown_counter)
1238 {
1239     (void) printf("\nSystem-Calls:\n");
1240     if(system("/usr/local/bin/gpio edge 19 none") != 0)
1241     {
1242         save_error(strerror(errno));
1243         save_error("->system(wd_isr deaktivieren)");
1244         return EXIT_FAILURE;
1245     }
1246     if(system("/usr/local/bin/gpio edge 17 none") != 0)
1247     {
1248         save_error(strerror(errno));
1249         save_error("->system(tacho_isr deaktivieren)");
1250         return EXIT_FAILURE;
1251     }
1252 
1253     if((START_GPS_CALC == 1) && !(*shutdown_counter > REPEAT_SHUTDOWN_SIGNAL))
1254     {
1255         if(system("sudo /home/pi/worksC/GPS_Calc/bin/Debug/GPS_Calc") != 0)
1256         {
1257             save_error(strerror(errno));
1258             save_error("->system(GPS_Calc)");
1259             return EXIT_FAILURE;
1260         }
1261     }
1262     else if(*shutdown_counter > REPEAT_SHUTDOWN_SIGNAL)
1263     {
1264         save_error("GPS_Calc deaktiviert, da shutdown-Befehl vom WD kommt.\n");
1265     }
1266     else
1267     {
1268         save_error("Auswertung durch GPS_Calc ist deaktiviert.\n"
1269                    "Zum aktivieren DataLogger.h->#define START_GPS_CALC 1\n");
1270     }
1271     if(system("mkdir -p /home/pi/databuffer/") != 0)
1272     {
1273         save_error(strerror(errno));
1274         save_error("->system(databuffer)");
1275         return EXIT_FAILURE;
1276     }
1277     (void) printf("System-Calls abgeschlossen\n");
1278 
1279     return EXIT_SUCCESS;
```

```
1280 }
1281
1282  /*------------------------------------------------------------------------------
1283 Führt Shutdown aus.
1284
1285 Parameter: button    Variable für Hardware Taster.
1286
1287 Return: EXIT_SUCCESS, wenn erfolgreich.
1288         EXIT_FAILURE, wenn Fehler in system().
1289 ------------------------------------------------------------------------------*/
1290 int execute_shutdown(int button, int *shutdown_counter)
1291 {
1292     if(*shutdown_counter > REPEAT_SHUTDOWN_SIGNAL)
1293     {
1294         if(system("shutdown -h 0") != 0)
1295         {
1296             save_error(strerror(errno));
1297             save_error("->system()");
1298             return EXIT_FAILURE;
1299         }
1300         save_error("\nshutdown -h 0 durch WD\n");
1301     }
1302     else if (button == 1)
1303     {
1304         if(system("shutdown -r 0") != 0)
1305         {
1306             save_error(strerror(errno));
1307             save_error("->system(-r)");
1308             return EXIT_FAILURE;
1309         }
1310
1311         save_error("\nshutdown -r durch Fahrzeug-Stillstand/SW-Stop\n");
1312     }
1313     else if(button == 0)
1314     {
1315         if(system("shutdown -h 0") != 0)
1316         {
1317             save_error(strerror(errno));
1318             save_error("->system(-h)");
1319             return EXIT_FAILURE;
1320         }
1321
1322         save_error("\nshutdown -h durch Button\n");
1323     }
1324     else
```

```
1325     {
1326         save_error("Unzulässiger Wert der Variable button");
1327         return EXIT_FAILURE;
1328     }
1329 
1330     return EXIT_SUCCESS;
1331 }
```

B.1.3. Datalogger.h

Die aus der Quelle [1] übernommenen Programmabschnitte wurden gekennzeichnet.

```
1  #ifndef DATALOGGER_H_INCLUDED
2  #define DATALOGGER_H_INCLUDED
3  
4  #define ok    "\033[32mOK\033[m"
5  #define err   "\033[31mERR\033[m"
6  #define warn "\033[33mWARN\033[m"
7  
8  #define TACHO_PIN 0
9  #define STOP_PIN 5
10 #define DOOR_PIN 6
11 #define BUTTON_PIN 27
12 #define GREEN_LED 28
13 #define RED_LED 29
14 
15 #define WD_INPUT 24     //Watchdog
16 #define WD_OUTPUT 23
17 
18 #define DOOR_OPEN 0
19 #define DOOR_CLOSED 1
20 
21 #define LED_ON 1
22 #define LED_OFF 0
23 
24 #define NANO 1000000000L
25 
26 //Deaktiviert = als Kommentar markieren.
27 //#############################################################################
28 #define LOGGING_DAUER 6000  //wie viele GPS-Messpunkte sollen aufgezeichnet
29                             //werden? Auskommentieren bei Normalenbetrieb.
30 //#define PRINT_TO_SCREEN
31 //#define PRINT_TIME
32 #define START_GPS_CALC 1
33 #define TACHO_TIMEOUT 36000LU // Timeout by TACHO = 10 * 60 * min.
```

```
34 //#define TACHO_TIMEOUT 15LU  //für Buffer-Test
35 #define MAX_FILES 11
36
37 //in Hz
38 #define WHILE_FREQ 30 //muss gleich der Frequenz des schnellsten Sensors sein.
39 #define DOOR_FREQ 30
40 #define TACHO_FREQ 30
41 #define IMU_ACC_FREQ 30
42 #define IMU_MAG_FREQ 30
43 #define IMU_GYR_FREQ 30
44 #define IMU_PR_TE_FREQ 30
45 #define GPS_FREQ 10
46
47 //Watchdog
48 //#define PRINT_WD_ISR
49 #define ORCPulselength 5     //[ms] - Length of the ORC Pulse-Signal
50                               //(pulsepause and pulse).
51 #define ORCSynclength 50   //[ms] - Length of the ORC Sync-Signal.
52         //!!!! 8 * ORCPulselength < ORCPulselength + ORCSynclength !!!!
53 #define TRINKET_TIME_ERROR 0  //gleicht zeitliche Ungenauigkeit von Trinket aus.
54                               //lässt sich durch PRINT_WD_ISR herausfinden.
55 #define REPEAT_SHUTDOWN_SIGNAL 4    //Anzahl der shutdown-Einlesung bis
56                                     //tatsächlich shutdown durchgeführt wird.
57 //TESTING
58 //#define TEST_BUFFER             //TACHO_TIMEOUT auf 15 setzen.
59                                   //Keine Tacho-Pulse draufgeben!
60                                   //sonst werden mehr also 15 Werte eingelesen.
61                                   //TEST_BUFFER_LOCATION aktivieren.
62 //#define TEST_BUFFER_LOCATION "/home/pi/venus_test_buffer.txt"
63 //#define WATCHDOG_TEST //bevor der Test durchgeführt wird
64                         //25(SW_Stop_Output) mit 24(WD_IN vom Rpi ausgesehen)
65                         //verbinden.
66                         //Für mehr Info #define PRINT_WD_ISR aktivieren.
67 //#############################################################################
68
69
70 int makedir(char *fdir);
71 /*@null@*/FILE *openfile(char *fdir, char *filename, char* mode);
72 int open_all_files(char *fdir, /*@null@*/FILE **all_files,
73                    char **file_names, char **file_mode);
74 int initializes_gpio(void);
75 void test_watchdog_ISR(FILE *fp_WATCHDOG);
76 void waitForStart(void);
77 int write_time(FILE *fp_TIME);
78 int read_venus_init(int fd);
```

```
79
80 int write_door(FILE *fp_DOOR, char *door, int max_freq, int sensor_hz,
81                int *loop_counter_door, unsigned long int index_fastest);
82 int write_tacho(FILE *fp_TACHO, int max_freq, int sensor_hz,
83                 int *loop_counter_tacho, unsigned long int index_fastest);
84 int write_imu(int fd_i2c, FILE *fp_ACC, FILE *fp_MAG, FILE *fp_GYR,
85               FILE *fp_PR_TE, double *t_pressure, int max_freq,
86               int freq_acc, int freq_mag, int freq_gyr, int freq_pr_te,
87               int *loop_counter_acc, int *loop_counter_mag,
88               int *loop_counter_gyr, int *loop_counter_pr_te,
89               unsigned long int index_fastest);
90 int write_gpspaket(int fd, FILE *fp_GGA, int max_freq, int sensor_hz,
91                    int *loop_counter_gps, unsigned long int *points_gps,
92                    unsigned long int index_fastest);
93 int watchdog_output(int *watchdog_out, int max_freq, int sensor_hz,
94                     int *loop_counter_watchdog_output);
95 int watchdog_input(FILE *fp_WATCHDOG, int *shutdown_counter, int max_freq,
96                    int sensor_hz, int *loop_counter_watchdog_input);
97 int match_while_freq(struct timespec begin, struct timespec end, int max_freq);
98
99 void read_full_buffer(int fd);
100 int write_points(FILE *fp_POINTS, unsigned long int points_gps);
101 int close_all_files(FILE **all_files);
102 int execute_system_calls(int *shutdown_counter);
103 int execute_shutdown(int button, int *shutdown_counter);
104
105
106
107 #endif // DATALOGGER_H_INCLUDED
```

B.1.4. LowFuncLogger.c

Die aus der Quelle [1] übernommenen Programmabschnitte wurden gekennzeichnet.

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <stdint.h>
4 #include <string.h>
5 #include <unistd.h>
6 #include <errno.h>
7 #include <termios.h>
8 #include <fcntl.h>
9 #include <unistd.h>
10 #include <sys/stat.h>
11 #include <time.h>
```

```
12 #include <sys/time.h>
13 #include <math.h>
14 #include <linux/i2c-dev.h>
15 #include <sys/ioctl.h>
16 #include <signal.h>
17
18 #include "LowFuncLogger.h"
19 #include "global.h"
20 #include "ErrorStorage.h"
21 #ifndef S_SPLINT_S
22 #include "wiringPi.h"
23 #endif
24 #include "DataLogger.h"
25
26 /*---------------------------------------------------------------------------
27 Hängt den zweiten String an den ersten.  (Erstellt von Mario Wegner)
28 Kein save_error(). ERROR_STORAGE.txt noch nicht erstellt.
29
30 Parameter: s1   Erster String.
31            s2   Zweiter String.
32
33 Return: restult erster plus zweiter String.
34         NULL, wenn malloc() fehlschlägt.
35 ---------------------------------------------------------------------------*/
36 char *concat(char *s1, char *s2)
37 {
38     char *result = malloc(strlen(s1) + strlen(s2) + 1); //+1 for '\0'.
39
40     if(result == NULL)
41     {
42         (void) printf("%s", strerror(errno));
43         (void) printf("->malloc()");
44         return NULL;
45     }
46     strcpy(result, s1);
47     strcat(result, s2);
48
49     return result;
50 }
51
52 /*---------------------------------------------------------------------------
53 Sorgt dafür, dass Sensor entsprechend sensor_hz ausgelesen wird.
54
55 Parameter: max_freq         Gewünschte Auslese-Frequenz des schnellsten Sensors.
56            sensor_hz        Frequenz mit der der Sensor ausgelesen wird.
```

```
57            *loop_counter    Laufvariable, die zeigt, ob der Sensor übersprungen
58                             oder ausgelesen wird.
59
60 Return: void
61 ------------------------------------------------------------------------------*/
62 void pull_freq(int max_freq, int sensor_hz, int *loop_counter_hz)
63 {
64     if(sensor_hz == 0)
65     {
66         *loop_counter_hz = 0;
67     }
68     else if(*loop_counter_hz != (int) (max_freq / sensor_hz))
69     {
70         (*loop_counter_hz)++;
71     }
72     else
73     {
74         *loop_counter_hz = 1;
75     }
76 }
77
78 /*------------------------------------------------------------------------------
79 Hängt Zähl-Index vor String. (Format: index_string)
80
81 Parameter: *index_sensor Welcher Zähl-Index soll angeheftet werden?
82            *string       An welceh String soll der Index angeheftet werden?
83
84 Return: index_plus_string, wenn erfolgreich. free nicht vergessen.
85         NULL, wenn malloc() oder count_digits() fehlschlägt.
86 ------------------------------------------------------------------------------*/
87 char *add_count_index(unsigned long int index_sensor, char *add_string)
88 {
89     size_t index_length;
90     size_t length_string = strlen(add_string);
91     char *index_plus_string;
92
93     if((index_length = count_digits(index_sensor)) == 0)
94     {
95         save_error("->count_digits()");
96         return NULL;
97     }
98     index_plus_string = malloc(index_length + 1 + length_string + 1);
99     if(index_plus_string == NULL)
100    {
101        save_error(strerror(errno));
```

```
102         save_error("->malloc()");
103         return NULL;
104     }
105     if(snprintf(index_plus_string, index_length + 1 + length_string + 1,
106                 "%lu_%s", index_sensor, add_string) <= 0)
107     {
108         save_error(strerror(errno));
109         save_error("snprintf(index)");
110         free(index_plus_string);
111         index_plus_string = NULL;
112         return NULL;
113     }
114
115     return index_plus_string;
116 }
117
118 /*-------------------------------------------------------------------------
119 Schreibt die aktuelle Zeit in String. (Format: YYYY.MM.DD.hh.mm.ss.mmm:)
120
121 Parameter: *currentTime String, in dem die aktuelle Zeit gespeichert wird.
122
123 Return: EXIT_SUCCESS, wenn erfolgreich.
124         EXIT_FAILURE, wenn strftime() fehlschlägt.
125 -------------------------------------------------------------------------*/
126 int write_timestamp(char *currentTime)
127 {
128     struct timespec curTime = {0};
129     char TS_buffer[80] = "";
130     struct tm result = {0};
131     long int millisec;
132
133     clock_gettime(CLOCK_REALTIME, &curTime);
134     millisec = lrint((double) curTime.tv_nsec / (double) 1000000L) ;
135     if(millisec >= 1000)
136     {
137         millisec -= 1000;
138         curTime.tv_sec = curTime.tv_sec + (time_t) 1;
139     }
140     localtime_r(&curTime.tv_sec, &result);
141     if(strftime(TS_buffer, 80, "%Y.%m.%d.%H.%M.%S.", &result) == 0)
142     {
143         save_error(strerror(errno));
144         save_error("->strftime()");
145         return EXIT_FAILURE;
146     }
```

```
147     if(snprintf(currentTime, 20 + 3 + 1 + 1,"%s%03li:",
148                 TS_buffer, millisec) <= 0)  //YYYY.MM.DD.hh.mm.ss.+mmm+:+'\0'
149     {
150         save_error(strerror(errno));
151         save_error("->snprintf(TS)");
152         return EXIT_FAILURE;
153     }
154 
155     return EXIT_SUCCESS;
156 }
157 
158 /*-------------------------------------------------------------------------
159 Lie ß t solange vom FileDescriptor(fd) bis gesuchter String eingelesen wird.
160 
161 Parameter: fd               File_Descriptor, der auf den Venus-Buffer zeigt.
162            *search_string   Bis zu welchem String soll eingelesen werden?
163            *found_string    Alle Zeichen, die bis zum String eingelesen wurden.
164                             Einschlie ß lich search_string.
165            length           Wie viele Zeichen sollen maximal eingelesen werden?
166 
167 Return: EXIT_SUCCESS, wenn erfolgreich.
168         EXIT_FAILURE, wenn search_string nicht gefunden wurde
169                       oder zu lange ein leerer Buffer vorliegt.
170 -------------------------------------------------------------------------*/
171 int read_fd_till_string(int fd, char *search_string, char *found_string,
172                         int length)
173 {
174     char buffer[1 + 1];
175     int i = 0;
176     int empty_buffer = 0;
177     char found_string_temp[length];
178 
179     found_string_temp[0] = '\0';
180     do
181     {
182         if(i >= length - 1)
183         {
184             save_error(found_string_temp);
185             save_error(" :String nicht gefunden. Gesucht wurde:");
186             save_error(search_string);
187             return EXIT_FAILURE;
188         }
189         if(read(fd, buffer, 1) != 1)
190         {
191             empty_buffer++;
```

```
192             if(empty_buffer > 2000)
193             {
194                 save_error("2s leeren Buffer gelesen");
195                 return EXIT_FAILURE;
196             }
197             #ifdef PRINT_TO_SCREEN
198                 (void) printf("Buffer leer! %d", empty_buffer);
199             #endif
200             milli_sleep(0.5);
201             continue;
202         }
203         buffer[1] = '\0';
204         found_string_temp[i] = buffer[0];
205         i++;
206         found_string_temp[i] = '\0';
207     } while(strstr(found_string_temp, search_string) == NULL);
208     strcpy(found_string, found_string_temp);
209 
210     return EXIT_SUCCESS;
211 }
212 
213 /*---------------------------------------------------------------------------
214 Übersetzt Pulse-Anzahl in Text.
215 
216 Parameter: *watchdog_info     Speichert überstezte Info als Text.
217            *akku_v            Speichert übersetzte Akku-Spannung in Volt.
218            *shutdown_counter  Zählt hoch, wenn shutdown-Info eingelesen wird.
219                               Durch REPEAT_SHUTDOWN_SIGNAL lässt sich festlegen,
220                               wie oft shutdown-Info eingelesen werden muss,
221                               um einen shutdown durchzuführen.
222 
223 Return: EXIT_SUCCESS, wenn erfolgreich.
224         EXIT_FAILURE, wenn snprintf() fehlschlägt.
225 ---------------------------------------------------------------------------*/
226 int decode_wd_input(char *watchdog_info, float *akku_v, int *shutdown_counter)
227 {
228     int akku_spannung = global_akku_spannung;
229     int netz_shutdown = global_netz_shutdown;
230 
231     #ifdef PRINT_WD_ISR
232     (void) printf("\ninfo: %d | akku: %d | shutdown_counter: %d\n",
233                   netz_shutdown, akku_spannung, *shutdown_counter);
234     #endif
235     if(akku_spannung > 0)
236     {
```

```
237         *akku_v = ((float) akku_spannung * 0.1f) + 2.9f;
238     }
239     if(netz_shutdown == 0)
240     {
241         strcpy(watchdog_info, "Busnetz aus");
242     }
243     else if(netz_shutdown == 1)
244     {
245         strcpy(watchdog_info, "Busnetz an");
246     }
247     else if(netz_shutdown == 2)
248     {
249         strcpy(watchdog_info, "2 Pulse (Undefinierter Zustand)");
250         save_error("ungültige Anzahl an WD_Info_Pulsen! Pulse = 2.\n");
251     }
252     else if(netz_shutdown == 3)
253     {
254         (*shutdown_counter)++;
255         if(snprintf(watchdog_info, 10 + 23 + 1, "%dter Befehl zum shutdown",
256                     *shutdown_counter) <= 0)
257         {
258             save_error(strerror(errno));
259             save_error("->snprintf(watchdog_shutdown)");
260             return EXIT_FAILURE;
261         }
262         if(*shutdown_counter > REPEAT_SHUTDOWN_SIGNAL)
263         {
264             strcpy(watchdog_info, "Befehl von Watchdog: shutdown durchführen");
265             //Programm sauber beeneden und Rpi runterfahren.
266         }
267     }
268     else
269     {
270         //negativ
271     }
272
273     return EXIT_SUCCESS;
274 }
275
276 /*------------------------------------------------------------------------------
277 Berechnet Zeit-Differenz in double msec von begin zu end.
278 Ermittelt durch clock_gettime().
279
280 Parameter: begin    Startzeit aufgenommen mit clock_gettime().
281            end      Stopzeit aufgenommen mit clock_gettime().
```

```
282
283 Return: duration, wenn erfolgreich.
284         -1, wenn Falsche Zeit-Argumente.
285 ------------------------------------------------------------------------------*/
286 double calc_time(struct timespec begin, struct timespec end)
287 {
288     struct timespec temp = {0};
289     double duration;
290
291     if((end.tv_nsec - begin.tv_nsec) > 0)
292     {
293         temp.tv_sec = (end.tv_sec - begin.tv_sec);
294         temp.tv_nsec = (end.tv_nsec - begin.tv_nsec);
295     }
296     else if((end.tv_nsec - begin.tv_nsec) < 0)
297     {
298         temp.tv_sec = (end.tv_sec - begin.tv_sec) - 1;
299         temp.tv_nsec =  NANO + (end.tv_nsec - begin.tv_nsec);
300     }
301     else
302     {
303         save_error("Falsche Zeit-Argumente");
304         return -1.0;
305     }
306     duration = (double) temp.tv_sec + ((double) temp.tv_nsec / (double) NANO);
307     duration = duration * 1000.0;
308
309    // (void) printf("\nTime for Execution: %f ms\n", duration);
310
311     return duration;
312 }
313
314 /*------------------------------------------------------------------------------
315 Zählt Stellen einer unsigned long int Zahl.
316
317 Parameter: number   Zahl, dessen Stellen gezählt werden sollen.
318
319 Return: digits Anzahl der Stellen.
320         0, wenn falscher Parameter.
321 ------------------------------------------------------------------------------*/
322 size_t count_digits(unsigned long int number)
323 {
324     size_t digits = 0;
325
326     if(number == 0)
```

```
327     {
328         digits = 1;
329     }
330     else if(number > 0)
331     {
332         digits = (size_t) floor(log10((double) number)) + 1;
333     }
334     else
335     {
336         save_error("Falsches Argument number");
337         return 0;
338     }
339 
340     return digits;
341 }
342 
343 /*------------------------------------------------------------------------------
344 Wartet.
345 
346 Parameter: time_milli   Wie viele Millisekunden soll gewartet werden?
347                         (In double!)
348 
349 Return: void
350 ------------------------------------------------------------------------------*/
351 void milli_sleep(double time_milli)
352 {
353     struct timespec temp = {0};
354 
355     if(time_milli < 1000.0)
356     {
357         temp.tv_sec = 0;
358         temp.tv_nsec = (long int)(time_milli * 1000000.0);
359     }
360     else
361     {
362         temp.tv_sec = (time_t)(time_milli / 1000.0);
363         temp.tv_nsec = (long int)((time_milli - ((double) temp.tv_sec * 1000.0))
364                         * 1000000.0);
365     }
366     nanosleep(&temp, NULL);
367 }
368 
369 /*------------------------------------------------------------------------------
370 free
371 wird nicht verwendet. noch nicht.
```

```
372
373 Parameter: *ptr Pointer to free.
374
375 Return: void
376 ------------------------------------------------------------------------------*/
377 /*void clean_up(void *ptr)
378 {
379     if(ptr != NULL)    //verhindert double freeing
380     {
381         free(ptr);
382         ptr = NULL;
383     }
384
385 }
386
387 */
```

B.1.5. LowFuncLogger.h

Die aus der Quelle [1] übernommenen Programmabschnitte wurden gekennzeichnet.

```
1  #ifndef LOWFUNCLOGGER_H_INCLUDED
2  #define LOWFUNCLOGGER_H_INCLUDED
3
4  /*@null@*/char* concat(char *s1, char *s2);
5
6  void pull_freq(int max_freq, int sensor_hz, int *loop_counter_hz);
7  /*@null@*/char *add_count_index(unsigned long int index_sensor,
8                                  char *add_string);
9  int write_timestamp(char *currentTime);
10 int read_fd_till_string(int fd, char *search_string, char *found_string,
11                         int length);
12 int decode_wd_input(char *watchdog_info, float *akku_v, int *shutdown_counter);
13
14 double calc_time(struct timespec begin, struct timespec end);
15 size_t count_digits(unsigned long int number);
16 inline void milli_sleep(double time_milli);
17 //void clean_up(void *ptr);
18
19 #endif // LOWFUNCLOGGER_H_INCLUDED
```

B.1.6. ErrorStorage.c

```
1  #include <stdio.h>
```

```
2
3 #include "ErrorStorage.h"
4
5 void save_error(char *error_string)
6 {
7     if(fp_ERROR != NULL)
8     {
9         (void) fputs(error_string, fp_ERROR);
10    }
11 }
```

B.1.7. ErrorStorage.h

```
1 #ifndef ERRORSTORAGE_H_INCLUDED
2 #define ERRORSTORAGE_H_INCLUDED
3
4 /*@null@*/extern FILE *fp_ERROR;
5 void save_error(char *error_string);
6
7 #endif // ERRORSTORAGE_H_INCLUDED
```

B.1.8. global.h

Entnommen aus Quelle [1]. Modifizierter Inhalt wurde gekennzeichnet.

```
1 #ifndef GLOBAL_H_INCLUDED
2 #define GLOBAL_H_INCLUDED
3
4 extern volatile unsigned long int tick;
5 extern volatile unsigned long int tacho_timeout_counter;
6
7 /*-----------------------------------------------------------------------------
8 Start: Modifiziert von Roman Kray
9 -----------------------------------------------------------------------------*/
10 extern volatile int watchdog_pulse;
11 extern volatile int global_akku_spannung;
12 extern volatile int global_netz_shutdown;
13 extern volatile struct timespec start_wd;
14 extern int while_loop_wrong;
15 /*-----------------------------------------------------------------------------
16 Ende: Modifiziert von Roman Kray
17 -----------------------------------------------------------------------------*/
18
19 #endif // GLOBAL_H_INCLUDED
```

B.1.9. VenusGPS.c

Entnommen aus Quelle [1]. Modifizierter Inhalt wurde gekennzeichnet.

```
1  // WiringPi-Api einbinden
2  //#include <wiringPi.h>
3
4  // C-Standardbibliothek einbinden
5  #include <stdio.h>
6  #include <stdlib.h>
7  #include <string.h>
8  #include <unistd.h>
9  #include <errno.h>
10 #include <termios.h>
11 #include <fcntl.h>
12 #include <errno.h>
13
14 #include "VenusGPS.h"
15 #include "ErrorStorage.h"
16
17
18 static int open_serial(int Baudrate);
19 static int GPS_pos_rate(int fd, int rate);
20 static int GPS_serial_rate(int fd, int rate);
21 static int GPS_NMEA(int fd);
22 /*@unused@*/static int GPS_Message_Type(int fd, char type);
23 static int GPS_navmode(int fd, char x);
24 /*@unused@*/static void GPS_clearbuff(int fd);
25 static int GPS_waas(int fd);
26 /*@unused@*/static int GPS_power(int fd, char en);
27
28 // Aufbau einer Seriellen Schnittstelle (8N1) vom RaspberryPI-------------------
29
30 // Parameter:
31 // Baudrate = 9600, 19200, 38400, 57600 oder 115200
32 // - Setzt die Geschwindigkeit der Schnittstelle vom RaPi fest
33
34 // Return:
35 // fd = Schnittstellen Pointer
36 // -1 = Fehler beim Aufbau der Schnittstelle
37 static int open_serial(int  Baudrate)
38 {
39     int fd = -1;
40     struct termios options = {0};    // Optionen zum Schnittstellenaufbau
41
42     fd = open("/dev/ttyAMA0", O_RDWR | O_NOCTTY | O_NDELAY);//ttyAMA0: serielle
```

```
43                                                          //Schnittstelle an GPIO
44      if(fd>= 0)
45      {
46          switch(Baudrate)     // Zur Auswahl der Baudrate
47          {
48              case 9600:
49                  (void) cfsetispeed(&options, B9600);
50                  (void) cfsetospeed(&options, B9600);
51
52                  break;
53              case 19200:
54                  (void) cfsetispeed(&options, B19200);
55                  (void) cfsetospeed(&options, B19200);
56
57                  break;
58              case 38400:
59                  (void) cfsetispeed(&options, B38400);
60                  (void) cfsetospeed(&options, B38400);
61
62                  break;
63              case 57600:
64                  (void) cfsetispeed(&options, B57600);
65                  (void) cfsetospeed(&options, B57600);
66
67                  break;
68              case 115200:
69                  (void) cfsetispeed(&options, B115200);
70                  (void) cfsetospeed(&options, B115200);
71
72                  break;
73          }
74          options.c_cflag &= ~PARENB; // Kein Paritaetsbit
75          options.c_cflag &= ~CSTOPB; // Ein Stopbit
76          options.c_cflag &= ~CSIZE;
77          options.c_cflag |= CS8;     // 8 Datenbits
78          options.c_cflag |= (CLOCAL | CREAD);
79          options.c_lflag &= ~(ICANON | ECHO | ECHOE | ISIG);
80          options.c_iflag = (tcflag_t) IGNPAR;
81          options.c_oflag = (tcflag_t) ~OPOST;
82          options.c_cc[VMIN] = (cc_t) 255; // Mindeste Anzahl empfangender Woerter
83                                           //65 = GGA String
84          options.c_cc[VTIME] = (cc_t) 255;    // Timeout in x*0.1s
85          (void) tcflush(fd, TCIOFLUSH);      // Buffer löschen
86          if (tcsetattr(fd, TCSAFLUSH, &options) != 0)
87          {
```

```
88              return -1;
89          }
90      }
91
92      return (fd);
93  }
94
95
96
97  // Positionsbestimmungen pro Sekunde-----------------------------------------
98
99  // Parameter:
100 // fd = SchnittestellenPointer (erzeugt durch "open_serial"
101 // rate = 1, 2, 4, 5 oder 10
102 // - in Hz, ab 4Hz wird mindestens eine Baudrate von 38400 benötigt!!
103 static int GPS_pos_rate(int fd, int rate)
104 {
105     int i = 0, j = 0;
106     ssize_t anz = 0;
107     char c[2] = "";
108     char buffer[10 + 1] = "";
109     char char_rate = '0';
110
111     switch (rate)   // Zuweisung der Geschwindigkeit
112     {
113         case 1: char_rate = 0x01;
114         break;
115         case 2: char_rate = 0x02;
116         break;
117         case 4: char_rate = 0x04;
118         break;
119         case 5: char_rate = 0x05;
120         break;
121         case 10: char_rate = 0x0A;
122         break;
123     }
124     // Configurationsstring
125     buffer[0] = 0xA0;
126     buffer[1] = 0xA1;
127     buffer[2] = 0x00;
128     buffer[3] = 0x03;
129     buffer[4] = 0x0E;
130     buffer[5] = char_rate;
131     buffer[6] = 0x00;
132     buffer[7] = (char_rate^0x0E);
```

```
133     buffer[8] = 0x0D;
134     buffer[9] = 0x0A;
135     for(j = 0; j < 15; j++)
136     {
137         (void) write(fd, &buffer, sizeof(buffer));    // Senden!!
138         // ACK String abwarten
139         (void) tcflush(fd, TCIOFLUSH);      // Buffer löschen
140         for (i = 0; i < 200; i++)         // max 1000 Zeichen auf ACK warten
141         {
142             anz = read(fd, &c[0], 1);    // Zeichen einlesen, wenn kein Zeichen
143                                          //im Buffer anz = -1
144             while(anz == -1)  // warten Zeichen ein zeichen im Buffer ist
145             {
146                 anz = read(fd, &c[0], 1);
147             }
148             if ((c[1] == 0x83) && (c[0] == 0x0E))  // 83 = ACK, 0E = Message ID
149             {
150                 return 0;
151             }
152         //(void) printf("%c",c[0]);
153         //(void) printf("%d: %x, %x \n ",i,c[0],c[1]);
154         c[1] = c[0];    // schieben
155 
156         }
157         //(void) printf("GPS_pos_rate %d. Versuch: kein ACK von GPS Modul\n",
158                       //j+1);
159     }
160 
161     return -1;
162 }
163 
164 // Stellt die Schnittstellengeschwindigkeit vom VenusGPS ein
165 //(Standart ist 9600 Baud)-------------------------------------------------------
166 
167 //Parameter:
168 // fd = SchnittestellenPointer (erzeugt durch "open_serial"
169 // rate = 9600, 19200, 38400, 57600 oder 115200
170 // -Geschwindigkeit des VenusGPS Moduls. Danach muss die Schnittstelle mit
171 //der neuen Rate erneut aufgebaut werden!!
172 static int GPS_serial_rate(int fd, int rate)
173 {
174     int i = 0, j = 0;
175     char c[2] = "";
176     ssize_t anz = 0;
177     char buffer[11 + 1] = "";
```

```
178     char baud = '0';
179
180     switch (rate)   // Zuweisung der Geschwindigkeit
181     {
182         case 9600: baud = 0x01;
183         break;
184         case 19200: baud = 0x02;
185         break;
186         case 38400: baud = 0x03;
187         break;
188         case 57600: baud = 0x04;
189         break;
190         case 115200: baud = 0x05;
191         break;
192     }
193     // Configurationsstring
194     buffer[0] = 0xA0;
195     buffer[1] = 0xA1;
196     buffer[2] = 0x00;
197     buffer[3] = 0x04;
198     buffer[4] = 0x05;
199     buffer[5] = 0x00;
200     buffer[6] = baud;
201     buffer[7] = 0x00;
202     buffer[8] = baud^0x05;
203     buffer[9] = 0x0D;
204     buffer[10] = 0x0A;
205     (void) tcflush(fd, TCIOFLUSH);      // Buffer löschen
206     for(j = 0 ;j < 2; j++)
207     {
208         (void) write(fd, &buffer, sizeof(buffer));    // Senden!!
209         // ACK String abwarten
210         (void) tcflush(fd, TCIFLUSH);      // Buffer löschen
211         for (i = 0; i < 100; i++)         // max 1000 Zeichen auf ACK warten
212         {
213             anz = read(fd, &c[0], 1);    // Zeichen einlesen, wenn kein Zeichen
214                                          //im Buffer anz = -1
215             while(anz == -1)    // warten Zeichen ein zeichen im Buffer ist
216             {
217                 anz = read(fd, &c[0],1);
218             }
219             if ((c[1] == 0x83) && (c[0] == 0x05))  // 83 = ACK, 0E = Message ID
220             {
221                 //(void) printf("GPS Modul sendet mit %d BAUD\n",rate);
222                 return 0;
```

```
223             }
224             //(void) printf("%d: %x, %x \n ",i,c[0],c[1]);
225             c[1] = c[0];    // schieben
226         }
227         //(void) printf("GPS_serial_rate : kein ACK von GPS Modul\n");
228     }
229     //(void) printf("Das GPS Modul antwortet nicht");
230 
231     return -1;
232 }
233 
234 // Stellt ein, welche NMEA-Strings gesendet werden sollen, in diesem Fall
235 //werden nur die Strings GGA und RMC gesendet----------------------------------
236 // Parameter:
237 // fd = SchnittestellenPointer (erzeugt durch "open_serial"
238 static int GPS_NMEA(int fd)
239 {
240     int i = 0, j = 0;
241     char c[2] = "";
242     ssize_t anz = 0;
243     //  Configurationsstring
244     char buffer[] = {0xA0, 0xA1, 0x00, 0x09, 0x08, 0x01, 0x00, 0x00, 0x00,
245                      0x01, 0x00, 0x00, 0x00, 0x08, 0x0D, 0x0A};
246 
247     for(j = 0; j < 10; j++)
248     {
249         (void) write(fd, &buffer, sizeof(buffer));    // Senden
250         // ACK String abwarten
251         (void) tcflush(fd, TCIFLUSH);       // Buffer löschen
252         for (i = 0; i < 200; i++)          // max 1000 Zeichen auf ACK warten
253         {
254             anz = read(fd, &c[0], 1);    // Zeichen einlesen, wenn kein Zeichen
255                                          //im Buffer anz = -1
256             while(anz == -1)    // warten Zeichen ein zeichen im Buffer ist
257             {
258                 anz = read(fd, &c[0],1);
259             }
260             if ((c[1] == 0x83) && (c[0] == 0x08))  // 83 = ACK, 0E = Message ID
261             {
262                 //(void) printf("GPS Modul sendet den GGA String\n");
263                 return 0;
264             }
265             //(void) printf("%d: %x, %x \n ",i,c[0],c[1]);
266             c[1] = c[0];    // schieben
267         }
```

```
268         //(void) printf("GPS_NMEA %d. Versuch: kein ACK von GPS Modul\n",j+1);
269     }
270     //(void) printf("Das GPS Modul antwortet nicht");
271 
272     return -1;
273 }
274 
275 static int GPS_Message_Type(int fd, char type)
276 {
277     int i = 0, j = 0;
278     char c[2] = "";
279     ssize_t anz = 0;
280     char buffer[] = {0xA0, 0xA1, 0x00, 0x03, 0x09, type, 0x00, type^0x09,
281                      0x0D, 0x0A};
282 
283     for(j = 0; j < 10; j++)
284     {
285         (void) write(fd, &buffer, sizeof(buffer));
286         // ACK String abwarten
287         (void) tcflush(fd, TCIFLUSH);       // Buffer löschen
288         for (i = 0;i < 200; i++)          // max 1000 Zeichen auf ACK warten
289         {
290             anz = read(fd, &c[0], 1);     // Zeichen einlesen, wenn kein Zeichen
291                                           //im Buffer anz = -1
292             while(anz == -1)    // warten Zeichen ein zeichen im Buffer ist
293             {
294                 anz = read(fd, &c[0], 1);
295             }
296             if ((c[1] == 0x83) && (c[0] == 0x09))  // 83 = ACK, 0E = Message ID
297             {
298                 //(void) printf("GPS Modul sendet Biaerdaten\n");
299                 return 0;
300             }
301             //(void) printf("%d: %x, %x \n ",i,c[0],c[1]);
302             c[1] = c[0];    // schieben
303         }
304         //(void) printf("GPS_Message_Type %d. Versuch: kein ACK von GPS Modul\n"
305                       //,j+1);
306 }
307 //(void) printf("Das GPS Modul antwortet nicht\n");
308 return -1;
309 }
310 
311 int GPS_reset(int fd)
312 {
```

```
313     int i = 0, j = 0;
314     char c[2] = "";
315     ssize_t anz = 0;
316     char buffer[] = {0xA0, 0xA1, 0x00, 0x02, 0x04, 0x01, 0x05, 0x0D, 0x0A};
317 
318     for(j = 0; j < 10; j++)
319     {
320         (void) write(fd, &buffer, sizeof(buffer));
321         // ACK String abwarten
322         (void) tcflush(fd, TCIFLUSH);       // Buffer löschen
323         for (i = 0; i < 200; i++)          // max 1000 Zeichen auf ACK warten
324         {
325             anz = read(fd, &c[0], 1);     // Zeichen einlesen, wenn kein Zeichen
326                                           //im Buffer anz = -1
327             while(anz == -1)    // warten Zeichen ein zeichen im Buffer ist
328             {
329                 anz = read(fd, &c[0], 1);
330             }
331             if ((c[1] == 0x83) && (c[0] == 0x04))  // 83 = ACK, 0E = Message ID
332             {
333                 if(close(fd) == -1)
334                 {
335                     return -1;
336                 }
337                 //(void) printf("GPS Modul zurueckgesetzt\n");
338                 return 0;
339             }
340             //(void) printf("%d: %x, %x \n ",i,c[0],c[1]);
341             c[1] = c[0];     // schieben
342         }
343         (void) printf("GPS_reset %d. Versuch: kein ACK von GPS Modul\n", j + 1);
344     }
345 
346     return -1;
347     //(void) printf("Das GPS Modul antwortet nicht\n");
348 }
349 
350 
351 //
352 static int GPS_navmode(int fd,char x)
353 {
354     int i = 0, j = 0;
355     char c[2] = "";
356     ssize_t anz = 0;
357     char buffer[] = {0xA0, 0xA1, 0x00, 0x03, 0x3C, x, 0x00, (0x3C^x),
```

```
358                     0x0D, 0x0A};
359
360     for(j = 0; j < 10; j++)
361     {
362         (void) write(fd, &buffer, sizeof(buffer));
363         // ACK String abwarten
364         (void) tcflush(fd, TCIOFLUSH);      // Buffer löschen
365         for (i = 0; i < 200; i++)          // max 1000 Zeichen auf ACK warten
366         {
367             anz = read(fd, &c[0], 1);     // Zeichen einlesen, wenn kein Zeichen
368                                           //im Buffer anz = -1
369             while(anz == -1)    // warten Zeichen ein zeichen im Buffer ist
370             {
371                 anz = read(fd, &c[0], 1);
372             }
373             if ((c[1] == 0x83) && (c[0] == 0x3C))  // 83 = ACK, 3C = Message ID
374             {
375                 return 0;
376             }
377             //(void) printf("%c",c[0]);
378             //(void) printf("%d: %x, %x \n ",i,c[0],c[1]);
379             c[1] = c[0];    // schieben
380         }
381         //(void) printf("GPS_navmode : kein ACK von GPS Modul\n");
382     }
383     //printf("Das GPS Modul antwortet nicht\n");
384
385     return -1;
386 }
387
388 static void GPS_clearbuff(int fd)
389 {
390     char buffer[] = {0xA0, 0xA1, 0x00, 0x01, 0x19, 0x19, 0x0D, 0x0A};
391
392     (void) tcflush(fd, TCIOFLUSH);      // Buffer löschen
393     (void) write(fd, &buffer, sizeof(buffer));
394 }
395
396 static int GPS_waas(int fd)
397 {
398     int i = 0, j = 0;
399     char c[2] = "";
400     ssize_t anz = 0;
401     char buffer[] = {0xA0, 0xA1, 0x00, 0x03, 0x37, 0x01, 0x00, 0x36,
402                     0x0D, 0x0A};
```

```
403
404     for(j = 0; j < 10; j++)
405     {
406         (void) write(fd, &buffer, sizeof(buffer));
407         // ACK String abwarten
408         (void) tcflush(fd, TCIFLUSH);       // Buffer löschen
409         for (i = 0; i < 200; i++)          // max 1000 Zeichen auf ACK warten
410         {
411             anz = read(fd, &c[0], 1);     // Zeichen einlesen, wenn kein Zeichen
412                                           //im Buffer anz = -1
413             //(void) printf("%c",c[0]);
414             while(anz == -1)    // warten Zeichen ein zeichen im Buffer ist
415             {
416                 anz = read(fd, &c[0], 1);
417             }
418             if ((c[1] == 0x83) && (c[0] == 0x37))  // 83 = ACK, 0E = Message ID
419             {
420                 //(void) printf("GPS Modul WAAS aktiviert\n");
421                 return 0;
422             }
423             //(void) printf("%d: %x, %x \n ",i,c[0],c[1]);
424             c[1] = c[0];    // schieben
425         }
426         //(void) printf("GPS_waas %d. Versuch: kein ACK von GPS Modul\n",j+1);
427     }
428     //(void) printf("Das GPS Modul antwortet nicht\n");
429
430     return -1;
431 }
432
433 static int GPS_power(int fd, char en)
434 {
435     int i = 0, j = 0;
436     char c[2] = "";
437     ssize_t anz = 0;
438     char buffer[] = {0xA0, 0xA1, 0x00, 0x03, 0x0C, en, 0x00, (0x0C^en),
439                      0x0D, 0x0A};
440
441     for(j = 0; j < 5; j++)
442     {
443         (void) write(fd, &buffer, sizeof(buffer));
444         // ACK String abwarten
445         (void) tcflush(fd, TCIFLUSH);       // Buffer löschen
446         for (i = 0; i < 100; i++)          // max 1000 Zeichen auf ACK warten
447         {
```

```
448             anz = read(fd, &c[0], 1);     // Zeichen einlesen, wenn kein Zeichen
449                                           //im Buffer anz = -1
450             while(anz == -1)    // warten Zeichen ein zeichen im Buffer ist
451             {
452                 anz = read(fd, &c[0], 1);
453             }
454             if ((c[1] == 0x83) && (c[0] == 0x0C))// 83 = ACK, 0xXX = Message ID
455             {
456                 //(void) printf("GPS Modul WAAS aktiviert\n");
457                 return 0;
458             }
459             //(void) printf("%d: %x, %x \n ",i,c[0],c[1]);
460             c[1] = c[0];    // schieben
461         }
462         //(void) printf("GPS_waas %d. Versuch: kein ACK von GPS Modul\n",j+1);
463     }
464     //(void) printf("Das GPS Modul antwortet nicht\n");
465
466     return -1;
467 }
468
469 /*---------------------------------------------------------------------------
470 Start: Modifiziert von Roman Kray
471 ---------------------------------------------------------------------------*/
472 /*---------------------------------------------------------------------------
473 Initialisiert VENUS-GPS-Schnittstelle.
474
475 Parameter: void
476
477 Return: fd, wenn erfolgreich.
478         -1, wenn fehlschlägt.
479 ---------------------------------------------------------------------------*/
480 int initializes_gps(char c, int rate)
481 {
482     int fd = -1;      //Venus Schnittstelle
483
484     (void) printf("\nInitialisiere VenusGPS:\n");
485     fd = open_serial(9600);
486     if(fd == (-1))
487     {
488         (void) printf("\tSerielle Schnittstelle 9600 einrichten\t\t\t"
489                       "[ "err" ]\n");
490         save_error("Fehler");
491         save_error("->open_serial(9600)");
492         return -1;
```

```
493     }
494     else {(void) printf("\tSerielle Schnittstelle 9600 einrichten\t\t\t"
495                         "[  "ok"  ]\n");}
496
497     if( GPS_serial_rate(fd,115200) == -1)
498     {
499         (void) printf("\tGPS BAUD-Rate auf 115200 setzen\t\t\t\t[ "warn" ]\n");
500         save_error("Warnung");
501         save_error("->GPS_serial_rate()");
502         save_error("->initializes_gps()\n");
503         //return -1;    //warning
504     }
505     else {(void) printf("\tGPS BAUD-Rate auf 115200 setzen\t\t\t\t"
506                         "[  "ok"  ]\n");}
507
508     fd = open_serial(115200);
509     if(fd == (-1))
510     {
511         (void) printf("\tSerielle Schnittstelle 115200 einrichten\t\t"
512                       "[  "err" ]\n");
513         save_error("Fehler");
514         save_error("->open_serial(115200)");
515         return -1;
516     }
517     else {(void) printf("\tSerielle Schnittstelle 115200 einrichten\t\t"
518                         "[  "ok"  ]\n");}
519
520     if( GPS_waas(fd) == -1)
521     {
522         (void) printf("\tGPS WAAS Mode aktivieren\t\t\t\t[  "err" ]\n");
523         save_error("Fehler");
524         save_error("->GPS_waas()");
525         return -1;
526     }
527     else {(void) printf("\tGPS WAAS Mode aktivieren\t\t\t\t[  "ok"  ]\n");}
528
529     if( GPS_NMEA(fd) == -1)
530     {
531         (void) printf("\tGPS NMEA konfigurieren\t\t\t\t\t[  "err" ]\n");
532         save_error("Fehler");
533         save_error("->GPS_NMEA()");
534         return -1;
535     }
536     else {(void) printf("\tGPS NMEA konfigurieren\t\t\t\t\t[  "ok"  ]\n");}
537
```

```
538     if( GPS_navmode(fd,c) == -1)
539     {
540         (void) printf("\tGPS Navigationsmodus konfigurieren\t\t\t[  "err" ]\n");
541         save_error("Fehler");
542         save_error("->GPS_navmode()");
543         return -1;
544     }
545     else {(void) printf("\tGPS Navigationsmodus konfigurieren\t\t\t"
546                         "[  "ok"  ]\n");}
547 
548     if( GPS_pos_rate(fd,rate) == -1)
549     {
550         (void) printf("\tGPS Sendefrequenz %d Hz konfigurieren\t\t\t"
551                       "[  "err" ]\n", rate);
552         save_error("Fehler");
553         save_error("->GPS_pos_rate()");
554         return -1;
555     }
556     else {(void) printf("\tGPS Sendefrequenz %d Hz konfigurieren\t\t\t"
557                         "[  "ok"  ]\n", rate);}
558 
559     (void) printf("Initialisiere VenusGPS abgeschlossen.\n");
560     return fd;
561 
562 }
563 /*-------------------------------------------------------------------------
564 Ende: Modifiziert von Roman Kray
565 -------------------------------------------------------------------------*/
```

B.1.10. VenusGPS.h

Entnommen aus Quelle [1]. Modifizierter Inhalt wurde gekennzeichnet.

```
1  #ifndef VENUSGPS_H_INCLUDED
2  #define VENUSGPS_H_INCLUDED
3  
4  // WiringPi-Api einbinden
5  //#include <wiringPi.h>
6  
7  #define ok   "\033[32mOK\033[m"
8  #define err  "\033[31mERR\033[m"
9  #define warn "\033[33mWARN\033[m"
10 
11 int GPS_reset(int fd);
12 int initializes_gps(char c, int rate);
```

```
13 #endif // DATACALC_H_INCLUDED
```

B.1.11. adafruit_imu.c

Entnommen aus Quelle [9]. Modifizierter Inhalt wurde gekennzeichnet.

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <stdint.h>
4  #include <math.h>
5  #include <fcntl.h>
6  #include <sys/time.h>
7  #include <linux/i2c.h>
8  #include <linux/i2c-dev.h>
9  #include <signal.h>
10 #include <errno.h>
11 #include <sys/ioctl.h>
12 #include <unistd.h>
13 #include <string.h>
14
15 #include "adafruit_imu.h"
16 #include "ErrorStorage.h"
17 #include "DataLogger.h"
18 #include "LowFuncLogger.h"
19
20 static inline __s32 i2c_smbus_access(int file, char read_write, __u8 command,
21                                      int size, union i2c_smbus_data *data);
22 /*@unused@*/static inline __s32 i2c_smbus_write_quick(int file, __u8 value);
23 /*@unused@*/static inline __s32 i2c_smbus_read_byte(int file);
24 /*@unused@*/static inline __s32 i2c_smbus_write_byte(int file, __u8 value);
25 /*@unused@*/static inline __s32 i2c_smbus_read_byte_data(int file,
26                                                         __u8 command);
27 /*@unused@*/static inline __s32 i2c_smbus_write_byte_data(int file,
28                                                          __u8 command,
29                                                          __u8 value);
30 /*@unused@*/static inline __s32 i2c_smbus_read_word_data(int file,
31                                                         __u8 command);
32 /*@unused@*/static inline __s32 i2c_smbus_write_word_data(int file,
33                                                          __u8 command,
34                                                          __u16 value);
35 /*@unused@*/static inline __s32 i2c_smbus_process_call(int file, __u8 command,
36                                                       __u16 value);
37 /*@unused@*/static inline __s32 i2c_smbus_read_block_data(int file,
38                                                          __u8 command,
39                                                          __u8 *values);
```

```
40 /*@unused@*/static inline __s32 i2c_smbus_write_block_data(int file,
41                                                           __u8 command,
42                                                           __u8 length,
43                                                           const __u8 *values);
44 /*@unused@*/static inline __s32 i2c_smbus_read_i2c_block_data(int file,
45                                                             __u8 command,
46                                                             __u8 length,
47                                                             __u8 *values);
48 /*@unused@*/static inline __s32 i2c_smbus_write_i2c_block_data(int file,
49                                                              __u8 command,
50                                                              __u8 length,
51                                                            const __u8 *values);
52 /*@unused@*/static inline __s32 i2c_smbus_block_process_call(int file,
53                                                            __u8 command,
54                                                            __u8 length,
55                                                            __u8 *values);
56
57 static const unsigned char BMPx8x_OverSampling = 3;
58
59 // Calibration values - These are stored in the BMP/180
60 static short int            bmp_ac1;
61 static short int            bmp_ac2;
62 static short int            bmp_ac3;
63 static unsigned short int   bmp_ac4;
64 static unsigned short int   bmp_ac5;
65 static unsigned short int   bmp_ac6;
66 static short int            bmp_b1;
67 static short int            bmp_b2;
68 static int                  bmp_b5;
69 static short int            bmp_mb;
70 static short int            bmp_mc;
71 static short int            bmp_md;
72
73 static int      bmp_ReadInt          (int fd, uint8_t *devValues,
74                                       uint8_t startReg,uint8_t bytesToRead);
75 static int      bmp_WriteCmd         (int fd, uint8_t devAction);
76 static int      WaitForConversion    (int fd);
77 static int      bmp_GetPressure      (int fd, double *Pres);
78 static int      bmp_GetTemperature   (int fd, double *Temp);
79 //BMP180 END ------------------------------------------------------------------
80
81
82 //i2c-dev END -----------------------------------------------------------------
83
84 //FUNCTION LSM303 AND L3G START
```

```
85 int file;
86 static void     readBlock   (uint8_t command, uint8_t size, uint8_t *data);
87 static void     selectDevice  (int file, int addr);
88 static void     readACC     (int  *a);
89 static void     readMAG     (int  *m);
90 static void     readGYR     (int *g);
91 /*@unused@*/static void     writeAccReg   (uint8_t reg, uint8_t value);
92 /*@unused@*/static void     writeMagReg   (uint8_t reg, uint8_t value);
93 /*@unused@*/static void     writeGyrReg   (uint8_t reg, uint8_t value);
94 static int    enableIMU(void);
95 static void     INThandler    (int sig);
96 /*@unused@*/static int    mymillis(void);
97 /*@unused@*/static int timeval_subtract(struct timeval *result,
98                                         struct timeval *t2, struct timeval *t1);
99 static int          i2c_Open        (char *I2CBusName);
100 static int          bmp_Calibration (int fd);
101
102
103 //BMP180 START ------------------------------------------------------------------
104
105 // Returns a file id for the port/bus
106 static int i2c_Open(char *I2CBusName){
107   int fd;
108   //Open port for reading and writing
109   if ((fd = open(I2CBusName, O_RDWR)) < 0){
110     (void) printf ("\n Failed to open the i2c bus!\n");
111     return -1;      //Use this line if the function must terminate on failure
112     }
113     else{
114      return fd;
115     }
116 }
117
118 // BMP085 & BMP180 Specific code
119
120 static int bmp_ReadInt(int fd, uint8_t *devValues, uint8_t startReg,
121                        uint8_t bytesToRead)
122 {
123   int rc;
124   struct i2c_rdwr_ioctl_data messagebuffer;
125
126   //Build a register read command
127   //Requires a one complete message containing a command
128   //and anaother complete message for the reply
129   struct i2c_msg read_reg[2]={
```

```
130     {BMPx8x_I2CADDR,0,1,&startReg},
131     {BMPx8x_I2CADDR,I2C_M_RD,bytesToRead,devValues}
132   };
133
134   messagebuffer.nmsgs = 2;                  //Two message/action
135   messagebuffer.msgs = read_reg; //load the 'read__reg' message into the buffer
136   rc = ioctl(fd, I2C_RDWR, &messagebuffer); //Send the buffer to the bus
137                                             //and returns a send status
138   if (rc < 0 ){
139     (void) printf("\n");
140     (void) printf("Reg Read command failed!\n");
141     return rc;   //Use this line if it must return to the caller for processing
142   }
143   //note that the return data is contained
144   //in the array pointed to by devValues (passed by-ref)
145   return 0;
146 }
147
148 static int bmp_WriteCmd(int fd, uint8_t devAction)
149 {
150   int rc;
151   struct i2c_rdwr_ioctl_data messagebuffer;
152   uint8_t datatosend[2];
153   struct i2c_msg write_reg[1]={{BMPx8x_I2CADDR,0,2,datatosend}};
154   datatosend[0]=BMPx8x_CtrlMeas;
155   datatosend[1]=devAction;
156   //Build a register write command
157   //Requires one complete message containing a reg address and command
158
159
160   messagebuffer.nmsgs = 1;                  //One message/action
161   messagebuffer.msgs = write_reg;//load the 'write__reg' message into the buffer
162   rc = ioctl(fd, I2C_RDWR, &messagebuffer); //Send the buffer to the bus
163                                             //and returns a send status
164   if (rc < 0 ){
165     (void) printf("\n");
166     (void) printf("Write reg command failed!\n");
167     return rc;   //Use this line if it must return to the caller for processing
168   }
169   return 0;
170 }
171
172 static int bmp_Calibration(int fd)
173 {
174   uint8_t rValue[21];
```

```
175   if (bmp_ReadInt(fd,rValue,0xAA,22) == 0){
176      bmp_ac1=((rValue[0]<<8)|rValue[1]);
177      bmp_ac2=((rValue[2]<<8)|rValue[3]);
178    bmp_ac3=((rValue[4]<<8)|rValue[5]);
179    bmp_ac4=((rValue[6]<<8)|rValue[7]);
180    bmp_ac5=((rValue[8]<<8)|rValue[9]);
181    bmp_ac6=((rValue[10]<<8)|rValue[11]);
182    bmp_b1=((rValue[12]<<8)|rValue[13]);
183    bmp_b2=((rValue[14]<<8)|rValue[15]);
184    bmp_mb=((rValue[16]<<8)|rValue[17]);
185    bmp_mc=((rValue[18]<<8)|rValue[19]);
186    bmp_md=((rValue[20]<<8)|rValue[21]);
187
188     return 0;
189   }
190   return -1;
191 }
192
193 static int WaitForConversion(int fd){
194   uint8_t rValues[3];
195   int counter=0;
196   //Delay can now be reduced by checking that bit 5 of Ctrl_Meas(0xF4) == 0
197   do{
198     (void) usleep((BMPx8x_RetryDelay)*1000);
199     if (bmp_ReadInt(fd,rValues,BMPx8x_CtrlMeas,1) != 0 ) return -1;
200     counter++;
201
202   }while ( ((rValues[0] & 0x20) != 0)  &&  counter < 20 );
203   return 0;
204 }
205
206 // Calculate calibrated pressure
207 // Value returned will be in hPa
208 static int bmp_GetPressure(int fd, double *Pres)
209 {
210   unsigned int up;
211   uint8_t rValues[3];
212     int x1, x2, x3, b3, b6, p;
213
214   unsigned int b4, b7;
215 //Pressure conversion with oversampling 0x34+ BMPx8x_OverSampling 'bit shifted'
216   if (bmp_WriteCmd(fd, (BMPx8x_PresConversion0+(BMPx8x_OverSampling<<6)))!=0)
217   {
218       return -1;
219   }
```

```
220
221   //Delay gets longer the higher the oversampling must be at least 26 ms
222   //plus a bit for turbo clock error ie 26 * 1000/700 or 38 ms
223   //sleepms (BMPx8x_minDelay + (4<<BMPx8x_OverSampling));
224   //39ms at oversample = 3
225
226   //Code is now 'turbo' overclock independent
227   (void) usleep(BMPx8x_minDelay * 1000);
228   if (WaitForConversion(fd) !=0 ) return -1;
229
230   //printf ("\nDelay:%i\n",(BMPx8x_minDelay+(4<<BMPx8x_OverSampling)));
231   if (bmp_ReadInt(fd, rValues, BMPx8x_Results,3) !=0 ) return -1;
232   up = (((unsigned int) rValues[0] << 16) | ((unsigned int) rValues[1] << 8) |
233         (unsigned int) rValues[2]) >> (8-BMPx8x_OverSampling);
234
235
236
237   b6 = bmp_b5 - 4000;
238   x1 = (bmp_b2 * (b6 * b6)>>12)>>11;
239   x2 = (bmp_ac2 * b6)>>11;
240   x3 = x1 + x2;
241   b3 = (((((int)bmp_ac1)*4 + x3)<<BMPx8x_OverSampling) + 2)>>2;
242
243   x1 = (bmp_ac3 * b6)>>13;
244   x2 = (bmp_b1 * ((b6 * b6)>>12))>>16;
245   x3 = ((x1 + x2) + 2)>>2;
246   b4 = (bmp_ac4 * (unsigned int)(x3 + 32768))>>15;
247
248   b7 = ((unsigned int)(up - b3) * (50000>>BMPx8x_OverSampling));
249   if (b7 < 0x80000000)
250     p = (b7<<1)/b4;
251   else
252     p = (b7/b4)<<1;
253
254   x1 = (p>>8) * (p>>8);
255   x1 = (x1 * 3038)>>16;
256   x2 = (-7357 * p)>>16;
257   p += (x1 + x2 + 3791)>>4;
258   *Pres = ((double)p/100);
259   return 0;
260 }
261
262 // Calculate calibrated temperature
263 // Value returned will be in units of 0.1 deg C
264 static int bmp_GetTemperature(int fd, double *Temp)
```

```
265 {
266   unsigned int ut;
267   uint8_t rValues[2];
268 int x1, x2;
269  double result = ((bmp_b5 + 8)>>4);
270   if (bmp_WriteCmd(fd, BMPx8x_TempConversion)!=0) return -1;
271   //Code is now 'turbo' overclock independent
272   (void) usleep(BMPx8x_minDelay * 1000);
273   if (WaitForConversion(fd) !=0 ) return -1;
274 
275   if (bmp_ReadInt(fd, rValues, BMPx8x_Results,2) !=0 ) return -1;
276   ut=((rValues[0]<<8)|rValues[1]);
277 
278 
279   x1 = (((int)ut - (int)bmp_ac6)*(int)bmp_ac5) >> 15;
280   x2 = ((int)bmp_mc << 11)/(x1 + bmp_md);
281   bmp_b5 = x1 + x2;
282 
283 
284   *Temp = result/10;
285   return 0;
286 }
287 //BMP180 END ------------------------------------------------------------------
288 
289 // i2c-dev START
290 
291 /* Note: 10-bit addresses are NOT supported! */
292 
293 /* This is the structure as used in the I2C_SMBUS ioctl call */
294 
295 /*struct i2c_smbus_ioctl_data {
296   char read_write;
297   __u8 command;
298   int size;
299   union i2c_smbus_data *data;
300 };*/
301 
302 /* This is the structure as used in the I2C_RDWR ioctl call */
303 /*struct i2c_rdwr_ioctl_data {
304   struct i2c_msg *msgs;  pointers to i2c_msgs
305   int nmsgs;      number of i2c_msgs
306 };*/
307 
308 
309 
```

```
310 static inline __s32 i2c_smbus_access(int file, char read_write, __u8 command,
311                                        int size, union i2c_smbus_data *data)
312 {
313   struct i2c_smbus_ioctl_data args;
314
315   args.read_write = read_write;
316   args.command = command;
317   args.size = size;
318   args.data = data;
319   return ioctl(file,I2C_SMBUS,&args);
320 }
321
322
323 static inline __s32 i2c_smbus_write_quick(int file, __u8 value)
324 {
325   return i2c_smbus_access(file,value,0,I2C_SMBUS_QUICK,NULL);
326 }
327
328 static inline __s32 i2c_smbus_read_byte(int file)
329 {
330   union i2c_smbus_data data;
331   if (i2c_smbus_access(file,I2C_SMBUS_READ,0,I2C_SMBUS_BYTE,&data))
332     return -1;
333   else
334     return 0x0FF & data.byte;
335
336 }
337
338 static inline __s32 i2c_smbus_write_byte(int file, __u8 value)
339 {
340 return i2c_smbus_access(file,I2C_SMBUS_WRITE,value,
341 I2C_SMBUS_BYTE,NULL);
342 }
343
344 static inline __s32 i2c_smbus_read_byte_data(int file, __u8 command)
345 {
346 union i2c_smbus_data data;
347 if (i2c_smbus_access(file,I2C_SMBUS_READ,command,
348 I2C_SMBUS_BYTE_DATA,&data))
349 return -1;
350 else
351 return 0x0FF & data.byte;
352 }
353
354 static inline __s32 i2c_smbus_write_byte_data(int file, __u8 command,
```

```
355                                                 __u8 value)
356 {
357 union i2c_smbus_data data;
358 data.byte = value;
359 return i2c_smbus_access(file,I2C_SMBUS_WRITE,command,
360 I2C_SMBUS_BYTE_DATA, &data);
361 }
362
363 static inline __s32 i2c_smbus_read_word_data(int file, __u8 command)
364 {
365 union i2c_smbus_data data;
366 if (i2c_smbus_access(file,I2C_SMBUS_READ,command,
367 I2C_SMBUS_WORD_DATA,&data))
368 return -1;
369 else
370 return 0x0FFFF & data.word;
371 }
372
373 static inline __s32 i2c_smbus_write_word_data(int file, __u8 command,
374                                                 __u16 value)
375 {
376 union i2c_smbus_data data;
377 data.word = value;
378 return i2c_smbus_access(file,I2C_SMBUS_WRITE,command,
379 I2C_SMBUS_WORD_DATA, &data);
380 }
381
382 static inline __s32 i2c_smbus_process_call(int file, __u8 command, __u16 value)
383 {
384 union i2c_smbus_data data;
385 data.word = value;
386 if (i2c_smbus_access(file,I2C_SMBUS_WRITE,command,
387 I2C_SMBUS_PROC_CALL,&data))
388 return -1;
389 else
390 return 0x0FFFF & data.word;
391 }
392
393
394 /* Returns the number of read bytes */
395 static inline __s32 i2c_smbus_read_block_data(int file, __u8 command,
396                                                 __u8 *values)
397 {
398 union i2c_smbus_data data;
399 int i;
```

```
400 if (i2c_smbus_access(file,I2C_SMBUS_READ,command,
401 I2C_SMBUS_BLOCK_DATA,&data))
402 return -1;
403 else {
404 for (i = 1; i <= data.block[0]; i++)
405 values[i-1] = data.block[i];
406 return data.block[0];
407 }
408 }
409
410 static inline __s32 i2c_smbus_write_block_data(int file, __u8 command,
411                                                __u8 length, const __u8 *values)
412 {
413 union i2c_smbus_data data;
414 int i;
415 if (length > 32)
416 length = 32;
417 for (i = 1; i <= length; i++)
418 data.block[i] = values[i-1];
419 data.block[0] = length;
420 return i2c_smbus_access(file,I2C_SMBUS_WRITE,command,
421 I2C_SMBUS_BLOCK_DATA, &data);
422 }
423
424 /* Returns the number of read bytes */
425 /* Until kernel 2.6.22, the length is hardcoded to 32 bytes. If you
426 ask for less than 32 bytes, your code will only work with kernels
427 2.6.23 and later. */
428 static inline __s32 i2c_smbus_read_i2c_block_data(int file, __u8 command,
429                                                   __u8 length, __u8 *values)
430 {
431 union i2c_smbus_data data;
432 int i;
433
434 if (length > 32)
435 length = 32;
436 data.block[0] = length;
437 if (i2c_smbus_access(file,I2C_SMBUS_READ,command,
438 length == 32 ? I2C_SMBUS_I2C_BLOCK_BROKEN :
439 I2C_SMBUS_I2C_BLOCK_DATA,&data))
440 return -1;
441 else {
442 for (i = 1; i <= data.block[0]; i++)
443 values[i-1] = data.block[i];
444 return data.block[0];
```

```
445 }
446 }
447
448 static inline __s32 i2c_smbus_write_i2c_block_data(int file, __u8 command,
449                                                    __u8 length,
450                                                    const __u8 *values)
451 {
452 union i2c_smbus_data data;
453 int i;
454 if (length > 32)
455 length = 32;
456 for (i = 1; i <= length; i++)
457 data.block[i] = values[i-1];
458 data.block[0] = length;
459 return i2c_smbus_access(file,I2C_SMBUS_WRITE,command,
460 I2C_SMBUS_I2C_BLOCK_BROKEN, &data);
461 }
462
463 /* Returns the number of read bytes */
464 static inline __s32 i2c_smbus_block_process_call(int file, __u8 command,
465                                                  __u8 length, __u8 *values)
466 {
467 union i2c_smbus_data data;
468 int i;
469 if (length > 32)
470 length = 32;
471 for (i = 1; i <= length; i++)
472 data.block[i] = values[i-1];
473 data.block[0] = length;
474 if (i2c_smbus_access(file,I2C_SMBUS_WRITE,command,
475 I2C_SMBUS_BLOCK_PROC_CALL,&data))
476 return -1;
477 else {
478 for (i = 1; i <= data.block[0]; i++)
479 values[i-1] = data.block[i];
480 return data.block[0];
481 }
482 }
483
484
485 // i2c-dev END ------------------------------------------------------------------
486
487 //LSM303 AND L3G START
488
489
```

```
490 static void  readBlock(uint8_t command, uint8_t size, uint8_t *data)
491 {
492     int result = i2c_smbus_read_i2c_block_data(file, command, size, data);
493     if (result != size)
494     {
495         (void) printf("Failed to read block from I2C.");
496         exit(1);
497     }
498 }
499
500 static void selectDevice(int file, int addr)
501 {
502
503         if (ioctl(file, I2C_SLAVE, addr) < 0) {
504                 (void) printf("Error: Could not select device\n");
505         }
506 }
507
508 static void readACC(int  *a)
509 {
510   uint8_t block[6];
511         selectDevice(file,ACC_ADDRESS);
512     readBlock(0x80 | LSM303_OUT_X_L_A, sizeof(block), block);
513
514         *a = (int16_t)(block[0] | block[1] << 8) >> 4;
515         *(a+1) = (int16_t)(block[2] | block[3] << 8) >> 4;
516         *(a+2) = (int16_t)(block[4] | block[5] << 8) >> 4;
517 }
518
519 static void readMAG(int  *m)
520 {
521   uint8_t block[6];
522         selectDevice(file,MAG_ADDRESS);
523   // DLHC: register address order is X,Z,Y with high bytes first
524   readBlock(0x80 | LSM303_OUT_X_H_M, sizeof(block), block);
525
526         *m = (int16_t)(block[1] | block[0] << 8);
527         *(m+1) = (int16_t)(block[5] | block[4] << 8) ;
528         *(m+2) = (int16_t)(block[3] | block[2] << 8) ;
529 }
530 static void readGYR(int *g)
531 {
532   uint8_t block[6];
533
534         selectDevice(file,GYR_ADDRESS);
```

```
535
536   readBlock(0x80 | L3G_OUT_X_L, sizeof(block), block);
537
538         *g = (int16_t)(block[1] << 8 | block[0]);
539         *(g+1) = (int16_t)(block[3] << 8 | block[2]);
540         *(g+2) = (int16_t)(block[5] << 8 | block[4]);
541
542 }
543
544
545 static void writeAccReg(uint8_t reg, uint8_t value)
546 {
547     int result;
548     selectDevice(file,ACC_ADDRESS);
549   result = i2c_smbus_write_byte_data(file, reg, value);
550     if (result == -1)
551     {
552         printf ("Failed to write byte to I2C Acc.");
553         exit(1);
554     }
555 }
556
557 static void writeMagReg(uint8_t reg, uint8_t value)
558 {   int result;
559     selectDevice(file,MAG_ADDRESS);
560    result = i2c_smbus_write_byte_data(file, reg, value);
561     if (result == -1)
562     {
563         (void) printf("Failed to write byte to I2C Mag.");
564         exit(1);
565     }
566 }
567
568
569 static void writeGyrReg(uint8_t reg, uint8_t value)
570 {int result;
571     selectDevice(file,GYR_ADDRESS);
572    result = i2c_smbus_write_byte_data(file, reg, value);
573     if (result == -1)
574     {
575         (void) printf("Failed to write byte to I2C Gyr.");
576         exit(1);
577     }
578 }
579
```

```
580
581 static int enableIMU()
582 {
583
584   //__u16 block[I2C_SMBUS_BLOCK_MAX];
585
586         //int res, bus,  size;
587
588
589         char filename[20];
590         sprintf(filename, "/dev/i2c-%d", 1);
591         file = open(filename, O_RDWR);
592         if (file<0) {
593     (void) printf("Unable to open I2C bus!");
594         return -1;
595         }
596 #ifndef S_SPLINT_S
597  // Enable accelerometer.
598   writeAccReg(LSM303_CTRL_REG1_A, 0b01110111); //  z,y,x axis enabled ,
599                                                  //25Hz data rate
600   writeAccReg(LSM303_CTRL_REG4_A, 0b00011000); // +/- 4G full scale: FS = 01 on
601                                               //DLHC, high resolution output mode
602   writeAccReg(LSM303_FIFO_CTRL_REG_A, 0b00000000); //Bypass mode selectet
603
604
605  // Enable magnetometer
606     writeMagReg(LSM303_CRA_REG_M, 0b01110000);  //15 HZ
607     writeMagReg(LSM303_CRB_REG_M, 0b00100000); //+/- 1.3 Gauss
608         writeMagReg(LSM303_MR_REG_M, 0x00);  // enable magnometer
609                                               //continuous-conversation mode
610  // Enable Gyro
611         writeGyrReg(L3G_CTRL_REG1, 0b00001111); // 95Hz, cut-off 12,5 Normal
612                                                   //power mode, all axes enabled
613         writeGyrReg(L3G_CTRL_REG2, 0b00001001); //HP Normal Mode
614                                                   //+ cut-off frq. 0.009 Hz
615         writeGyrReg(L3G_CTRL_REG4, 0b00000000); // Continuos update,
616                                                   //250 dps full scale
617         writeGyrReg(L3G_CTRL_REG5, 0b01010000); //En Fifo, en HPF
618         writeGyrReg(L3G_FIFO_CTRL_REG, 0b00000000); //FIFO bypass mode
619 #endif
620     return 0;
621 }
622
623 static void  INThandler(int sig)
624 {
```

```
625         signal(sig, SIG_IGN);
626         exit(0);
627 }
628
629 static int mymillis()
630 {
631   struct timeval tv;
632   gettimeofday(&tv, NULL);
633   return (tv.tv_sec) * 1000 + (tv.tv_usec)/1000;
634 }
635
636 static int timeval_subtract(struct timeval *result, struct timeval *t2,
637                             struct timeval *t1)
638 {
639     long int diff = (t2->tv_usec + 1000000 * t2->tv_sec) -
640                     (t1->tv_usec + 1000000 * t1->tv_sec);
641     result->tv_sec = diff / 1000000;
642     result->tv_usec = diff % 1000000;
643     return (diff<0);
644 }
645
646 //LSM303 AND L3G END ----------------------------------------------------------
647
648
649 /*--------------------------------------------------------------------------------
650 Start: Modifiziert von Roman Kray
651 --------------------------------------------------------------------------------*/
652 int writeACC_to_data(FILE *fp_ACC, int max_freq, int freq_acc,
653                      int *loop_counter_acc, unsigned long int index_fastest)
654 {
655     pull_freq(max_freq, freq_acc, loop_counter_acc);
656     if(*loop_counter_acc == 1)
657     {
658         int  *Pacc_raw;
659         int  acc_raw[3];
660         char testc[36];
661         char* index_plus_imu;
662         char timestamp[25] = "";
663         float acc_x = 0.0;
664         float acc_y = 0.0;
665         float acc_z = 0.0;
666
667         Pacc_raw = acc_raw;
668         readACC(Pacc_raw);
669         if(acc_x_offset_faktor != 0 && acc_x_offset_summand != 0 &&
```

```
670             acc_y_offset_faktor != 0 && acc_y_offset_summand != 0 &&
671             acc_z_offset_faktor != 0 && acc_z_offset_summand != 0)
672         {
673             acc_x = (float) ((*acc_raw * ACC_GAIN) - acc_x_offset_summand)
674                     * acc_x_offset_faktor;
675             acc_y = (float) ((*(acc_raw + 1) * ACC_GAIN) - acc_y_offset_summand)
676                     * acc_y_offset_faktor;
677             acc_z = (float) ((*(acc_raw + 2)* ACC_GAIN) - acc_z_offset_summand)
678                     * acc_z_offset_faktor;
679         }
680         else
681         {
682             acc_x = (float) *acc_raw * ACC_GAIN;
683             acc_y = (float) *(acc_raw+1) * ACC_GAIN;
684             acc_z = (float) *(acc_raw+2) * ACC_GAIN;
685         }
686         #ifdef PRINT_TO_SCREEN
687         (void) printf("\nACC -> X: %7.3f m/s2 Y: %7.3f m/s2 Z: %7.3f m/s2\n",
688                acc_x, acc_y, acc_z);
689         #endif
690         sprintf(testc, "%8.3f;%8.3f;%8.3f\n$", acc_x, acc_y, acc_z);
691         if(write_timestamp(timestamp) == EXIT_FAILURE)
692         {
693             save_error("->write_timestamp()");
694             return EXIT_FAILURE;
695         }
696         if(fputs(timestamp, fp_ACC) == EOF)
697         {
698             save_error(strerror(errno));
699             save_error("->fputs(TS)");
700             return EXIT_FAILURE;
701         }
702         if((index_plus_imu = add_count_index(index_fastest, testc)) == NULL)
703         {
704             save_error("->add_count_index()");
705             return EXIT_FAILURE;
706         }
707         if(fputs(index_plus_imu, fp_ACC) == EOF)
708         {
709             save_error(strerror(errno));
710             save_error("->fputs(ACC)");
711             free(index_plus_imu);
712             index_plus_imu = NULL;
713             return EXIT_FAILURE;
714         }
```

```
715         free(index_plus_imu);
716         index_plus_imu = NULL;
717     }
718
719   return EXIT_SUCCESS;
720 }
721
722 int writeMAG_to_data(FILE *fp_MAG, int max_freq, int freq_mag,
723                      int *loop_counter_mag, unsigned long int index_fastest)
724 {
725     pull_freq(max_freq, freq_mag, loop_counter_mag);
726     if(*loop_counter_mag == 1)
727     {
728         int  *Pmag_raw;
729         int  mag_raw[3];
730         char testc[30];
731         char* index_plus_imu;
732         char timestamp[25] = "";
733         float mag_x = 0.0;
734         float mag_y = 0.0;
735         float mag_z = 0.0;
736
737         Pmag_raw = mag_raw;
738         readMAG(Pmag_raw);
739         mag_x = (float) *mag_raw / MAG_GAIN_XY;
740         mag_y = (float) *(mag_raw+1) / MAG_GAIN_XY;
741         mag_z = (float) *(mag_raw+2) / MAG_GAIN_Z;
742         #ifdef PRINT_TO_SCREEN
743         (void) printf("MAG -> X: %7.3f uT Y: %7.3f uT Z: %7.3f uT\n",
744                mag_x, mag_y, mag_z);
745         #endif
746         sprintf(testc, "%8.5f;%8.5f;%8.5f\n$", mag_x, mag_y, mag_z);
747         if(write_timestamp(timestamp) == EXIT_FAILURE)
748         {
749             save_error("->write_timestamp()");
750             return EXIT_FAILURE;
751         }
752         if(fputs(timestamp, fp_MAG) == EOF)
753         {
754             save_error(strerror(errno));
755             save_error("->fputs(TS)");
756             return EXIT_FAILURE;
757         }
758         if((index_plus_imu = add_count_index(index_fastest, testc)) == NULL)
759         {
```

```
760             save_error("->add_count_index()");
761             return EXIT_FAILURE;
762         }
763         if(fputs(index_plus_imu, fp_MAG) == EOF)
764         {
765             save_error(strerror(errno));
766             save_error("->fputs(MAG)");
767             free(index_plus_imu);
768             index_plus_imu = NULL;
769             return EXIT_FAILURE;
770         }
771         free(index_plus_imu);
772         index_plus_imu = NULL;
773     }
774
775   return EXIT_SUCCESS;
776 }
777
778 int writeGYR_to_data(FILE *fp_GYR, int max_freq, int freq_gyr,
779                      int *loop_counter_gyr, unsigned long int index_fastest)
780 {
781   pull_freq(max_freq, freq_gyr, loop_counter_gyr);
782   if(*loop_counter_gyr == 1)
783   {
784         int  *Pgyr_raw;
785         int  gyr_raw[3];
786         char testc[30];
787         char* index_plus_imu;
788         char timestamp[25] = "";
789         float gyr_x = 0.0;
790         float gyr_y = 0.0;
791         float gyr_z = 0.0;
792
793         Pgyr_raw = gyr_raw;
794         readGYR(Pgyr_raw);
795         gyr_x = (float) (*gyr_raw * G_GAIN);
796         gyr_y = (float) (*(gyr_raw+1) * G_GAIN);
797         gyr_z = (float) (*(gyr_raw+2) * G_GAIN);
798         #ifdef PRINT_TO_SCREEN
799         (void) printf("GYR -> X: %7.3f deg/s Y: %7.3f deg/s Z: %7.3f deg/s\n",
800                gyr_x, gyr_y, gyr_z);
801         #endif
802         sprintf(testc, "%8.3f;%8.3f;%8.3f\n$", gyr_x, gyr_y, gyr_z);
803         if(write_timestamp(timestamp) == EXIT_FAILURE)
804         {
```

```
805             save_error("->write_timestamp()");
806             return EXIT_FAILURE;
807         }
808         if(fputs(timestamp, fp_GYR) == EOF)
809         {
810             save_error(strerror(errno));
811             save_error("->fputs(TS)");
812             return EXIT_FAILURE;
813         }
814         if((index_plus_imu = add_count_index(index_fastest, testc)) == NULL)
815         {
816             save_error("->add_count_index()");
817             return EXIT_FAILURE;
818         }
819         if(fputs(index_plus_imu, fp_GYR) == EOF)
820         {
821             save_error(strerror(errno));
822             save_error("->fputs(GYR)");
823             free(index_plus_imu);
824             index_plus_imu = NULL;
825             return EXIT_FAILURE;
826         }
827         free(index_plus_imu);
828         index_plus_imu = NULL;
829     }
830 
831   return EXIT_SUCCESS;
832 }
833 
834 int writePR_TE_to_data(int fd, FILE *fp_PR_TE, double *t_pressure, int max_freq,
835                        int freq_pr_te, int *loop_counter_pr_te,
836                        unsigned long int index_fastest)
837 {
838     pull_freq(max_freq, freq_pr_te, loop_counter_pr_te);
839     if(*loop_counter_pr_te == 1)
840     {
841         double t_temperature;
842         char testc[30];
843         char* index_plus_imu;
844         char timestamp[25] = "";
845 
846         bmp_GetTemperature(fd, &t_temperature);
847         bmp_GetPressure(fd, t_pressure);
848         #ifdef PRINT_TO_SCREEN
849         (void) printf("PR_TR -> %f hPa %f degC\n", t_temperature, *t_pressure);
```

```
850         #endif
851         sprintf(testc, "%8.3f;%8.3f\n$", t_temperature, *t_pressure);
852         if(write_timestamp(timestamp) == EXIT_FAILURE)
853         {
854             save_error("->write_timestamp()");
855             return EXIT_FAILURE;
856         }
857         if(fputs(timestamp, fp_PR_TE) == EOF)
858         {
859             save_error(strerror(errno));
860             save_error("->fputs(TS)");
861             return EXIT_FAILURE;
862         }
863         if((index_plus_imu = add_count_index(index_fastest, testc)) == NULL)
864         {
865             save_error("->add_count_index()");
866             return EXIT_FAILURE;
867         }
868         if(fputs(index_plus_imu, fp_PR_TE) == EOF)
869         {
870             save_error(strerror(errno));
871             save_error("->fputs(PR_TE)");
872             free(index_plus_imu);
873             index_plus_imu = NULL;
874             return EXIT_FAILURE;
875         }
876         free(index_plus_imu);
877         index_plus_imu = NULL;
878     }
879
880   return EXIT_SUCCESS;
881 }
882
883 /*-----------------------------------------------------------------------------
884 Initialisiert IMU-Schnittstelle.
885
886 Parameter: void
887
888 Return: fd_i2c, wenn erfolgreich.
889         -1, wenn fehlschlägt.
890 -----------------------------------------------------------------------------*/
891 int initializes_imu(void)
892 {
893     int fd_i2c = -1;
894
```

```
895     (void) printf("\nInitialisiere IMU:\n");
896     fd_i2c = i2c_Open(I2CBus);
897     if(fd_i2c == -1)
898     {
899          (void) printf("\tI2C Schnittstelle einrichten\t\t\t\t[  "err" ]\n");
900          save_error("Fehler");
901          save_error("->i2c_Open()");
902          return -1;
903     }
904     else {(void) printf("\tI2C Schnittstelle einrichten\t\t\t\t[  "ok"  ]\n");}
905     if (bmp_Calibration(fd_i2c) == -1)
906     {
907         (void) printf("\tbmp_Calibration()\t\t\t\t\t[  "err" ]\n");
908         save_error("Fehler");
909         save_error("->bmp_Calibration()");
910         return -1;
911     }
912     else {(void) printf("\tbmp_Calibration()\t\t\t\t\t[  "ok"  ]\n");}
913     if(close(fd_i2c) == -1)
914     {
915         (void) printf("\tclose()\t\t\t\t\t\t\t[  "err" ]\n");
916         save_error(strerror(errno));
917         save_error("->close()");
918         return -1;
919     }
920     else {(void) printf("\tclose()\t\t\t\t\t\t\t[  "ok"  ]\n");}
921     if(signal(SIGINT, INThandler) == SIG_ERR)
922     {
923         (void) printf("\tsignal()\t\t\t\t\t\t[  "err" ]\n");
924         save_error(strerror(errno));
925         save_error("->signal()");
926         return -1;
927     }
928     else {(void) printf("\tsignal()\t\t\t\t\t\t[  "ok"  ]\n");}
929     if(enableIMU() == -1)
930     {
931         (void) printf("\tenableIMU()\t\t\t\t\t\t[  "err" ]\n");
932         save_error("Fehler");
933         save_error("->enableIMU()");
934         return -1;
935     }
936     else {(void) printf("\tenableIMU()\t\t\t\t\t\t[  "ok"  ]\n");}
937     (void) printf("Initialisiere IMU abgeschlossen.\n");
938 
939   return fd_i2c;
```

```
940 }
941 /*------------------------------------------------------------------------------
942 Ende: Modifiziert von Roman Kray
943 ------------------------------------------------------------------------------*/
944
945
946 //main fct END
```

B.1.12. adafruit_imu.h

Entnommen aus Quelle [9]. Modifizierter Inhalt wurde gekennzeichnet.

```
1  #ifndef ADAFRUIT_IMU_H_INCLUDED
2  #define ADAFRUIT_IMU_H_INCLUDED
3
4  #define ok   "\033[32mOK\033[m"
5  #define err  "\033[31mERR\033[m"
6  #define warn "\033[33mWARN\033[m"
7
8  #define acc_z_offset_summand 0.0000f
9  #define acc_z_offset_faktor 0.0000f
10
11 #define acc_y_offset_summand 0.0000f
12 #define acc_y_offset_faktor 0.0000f
13
14 #define acc_x_offset_summand 0.0000f
15 #define acc_x_offset_faktor 0.0000f
16 //BMP180 START ------------------------------------------------------------
17 #define I2CBus             "/dev/i2c-1"      //New Pi's
18
19 // BMP085 & BMP180 Specific code
20 #define BMPx8x_I2CADDR           0x77
21 #define BMPx8x_CtrlMeas          0xF4
22 #define BMPx8x_TempConversion    0x2E
23 #define BMPx8x_PresConversion0   0x34
24 #define BMPx8x_Results           0xF6
25 #define BMPx8x_minDelay          1//4    //require 4.5ms *1000/700
26                                          //'turbo mode fix'= 6.4-Retry =4.4
27 #define BMPx8x_RetryDelay        1//2     //min delay for temp 4+2=6ms,
28                                           //max 4+2*20=44ms for pressure
29                                   //Will stop waiting if conversion is complete
30
31 //L3G START
32 #define L3GD20_ADDRESS  0x6b
33 #define L3GD20_ADDRESS_SA0_HIGH   (0xD6 >> 1)
```

```
34 #define GYR_ADDRESS   (0xD6 >> 1)
35
36 #define L3G_WHO_AM_I      0x0F
37
38 #define L3G_CTRL_REG1     0xB0      //0x20
39 #define L3G_CTRL_REG2     0x21
40 #define L3G_CTRL_REG3     0x22
41 #define L3G_CTRL_REG4     0x23
42 #define L3G_CTRL_REG5     0x24
43 #define L3G_REFERENCE     0x25
44 #define L3G_OUT_TEMP      0x26
45 #define L3G_STATUS_REG    0x27
46
47 #define L3G_OUT_X_L       0x28
48 #define L3G_OUT_X_H       0x29
49 #define L3G_OUT_Y_L       0x2A
50 #define L3G_OUT_Y_H       0x2B
51 #define L3G_OUT_Z_L       0x2C
52 #define L3G_OUT_Z_H       0x2D
53
54 #define L3G_FIFO_CTRL_REG 0x2E
55 #define L3G_FIFO_SRC_REG  0x2F
56
57 #define L3G_INT1_CFG      0x30
58 #define L3G_INT1_SRC      0x31
59 #define L3G_INT1_THS_XH   0x32
60 #define L3G_INT1_THS_XL   0x33
61 #define L3G_INT1_THS_YH   0x34
62 #define L3G_INT1_THS_YL   0x35
63 #define L3G_INT1_THS_ZH   0x36
64 #define L3G_INT1_THS_ZL   0x37
65 #define L3G_INT1_DURATION 0x38
66 //L3G END ------------------------------------------------------------------
67
68 //LSM303 START
69
70 // register addresses
71 #define MAG_ADDRESS            (0x3C >> 1)
72 #define ACC_ADDRESS            (0x32 >> 1)
73 #define ACC_ADDRESS_SA0_A_LOW  (0x30 >> 1)
74 #define ACC_ADDRESS_SA0_A_HIGH (0x32 >> 1)
75
76 #define LSM303_CTRL_REG1_A       0x50   //0x30
77 #define LSM303_CTRL_REG2_A       0x21
78 #define LSM303_CTRL_REG3_A       0x22
```

```
79 #define LSM303_CTRL_REG4_A        0x23
80 #define LSM303_CTRL_REG5_A        0x24
81 #define LSM303_CTRL_REG6_A        0x25 // DLHC only
82 #define LSM303_HP_FILTER_RESET_A 0x25 // DLH, DLM only
83 #define LSM303_REFERENCE_A        0x26
84 #define LSM303_STATUS_REG_A       0x27
85
86 #define LSM303_OUT_X_L_A          0x28
87 #define LSM303_OUT_X_H_A          0x29
88 #define LSM303_OUT_Y_L_A          0x2A
89 #define LSM303_OUT_Y_H_A          0x2B
90 #define LSM303_OUT_Z_L_A          0x2C
91 #define LSM303_OUT_Z_H_A          0x2D
92
93 #define LSM303_FIFO_CTRL_REG_A    0x2E // DLHC only
94 #define LSM303_FIFO_SRC_REG_A     0x2F // DLHC only
95
96 #define LSM303_INT1_CFG_A         0x30
97 #define LSM303_INT1_SRC_A         0x31
98 #define LSM303_INT1_THS_A         0x32
99 #define LSM303_INT1_DURATION_A    0x33
100 #define LSM303_INT2_CFG_A         0x34
101 #define LSM303_INT2_SRC_A         0x35
102 #define LSM303_INT2_THS_A         0x36
103 #define LSM303_INT2_DURATION_A    0x37
104
105 #define LSM303_CLICK_CFG_A        0x38 // DLHC only
106 #define LSM303_CLICK_SRC_A        0x39 // DLHC only
107 #define LSM303_CLICK_THS_A        0x3A // DLHC only
108 #define LSM303_TIME_LIMIT_A       0x3B // DLHC only
109 #define LSM303_TIME_LATENCY_A     0x3C // DLHC only
110 #define LSM303_TIME_WINDOW_A      0x3D // DLHC only
111
112 #define LSM303_CRA_REG_M          0x1C       //0x00
113 #define LSM303_CRB_REG_M          0x01
114 #define LSM303_MR_REG_M           0x02
115
116 #define LSM303_OUT_X_H_M          0x03
117 #define LSM303_OUT_X_L_M          0x04
118 #define LSM303_OUT_Y_H_M          0x07
119                                        // The addresses of the Y and Z
120 #define LSM303_OUT_Y_L_M          0x08   // magnetometer output registers are
121 #define LSM303_OUT_Z_H_M          0x05   // reversed on the DLM and DLHC relative
122 #define LSM303_OUT_Z_L_M          0x06   //  to the DLH. These four defines have
123 //dummy values so the library candetermine the correct address based on
```

```
124 //the device type.
125
126 #define LSM303_SR_REG_M          0x09
127 #define LSM303_IRA_REG_M         0x0A
128 #define LSM303_IRB_REG_M         0x0B
129 #define LSM303_IRC_REG_M         0x0C
130
131 #define LSM303_WHO_AM_I_M        0x0F // DLM only
132
133 #define LSM303_TEMP_OUT_H_M      0x31 // DLHC only
134 #define LSM303_TEMP_OUT_L_M      0x32 // DLHC only
135 #define LSM303DLH_OUT_Y_H_M      0x05
136 #define LSM303DLH_OUT_Y_L_M      0x06
137 #define LSM303DLH_OUT_Z_H_M      0x07
138 #define LSM303DLH_OUT_Z_L_M      0x08
139
140 #define LSM303DLM_OUT_Z_H_M      0x05
141 #define LSM303DLM_OUT_Z_L_M      0x06
142 #define LSM303DLM_OUT_Y_H_M      0x07
143 #define LSM303DLM_OUT_Y_L_M      0x08
144
145 #define LSM303DLHC_OUT_Z_H_M     0x05
146 #define LSM303DLHC_OUT_Z_L_M     0x06
147 //LSM303 END ----------------------------------------------------------------
148
149 //i2c-dev START
150 /* To determine what functionality is present */
151
152 #define I2C_FUNC_I2C          0x00000001
153 #define I2C_FUNC_10BIT_ADDR       0x00000002
154 #define I2C_FUNC_PROTOCOL_MANGLING    0x00000004 // I2C_M_{REV_DIR_ADDR,
155                                                  // NOSTART,..}
156 #define I2C_FUNC_SMBUS_PEC        0x00000008
157 #define I2C_FUNC_SMBUS_BLOCK_PROC_CALL  0x00008000 /* SMBus 2.0 */
158 #define I2C_FUNC_SMBUS_QUICK      0x00010000
159 #define I2C_FUNC_SMBUS_READ_BYTE    0x00020000
160 #define I2C_FUNC_SMBUS_WRITE_BYTE   0x00040000
161 #define I2C_FUNC_SMBUS_READ_BYTE_DATA 0x00080000
162 #define I2C_FUNC_SMBUS_WRITE_BYTE_DATA  0x00100000
163 #define I2C_FUNC_SMBUS_READ_WORD_DATA 0x00200000
164 #define I2C_FUNC_SMBUS_WRITE_WORD_DATA  0x00400000
165 #define I2C_FUNC_SMBUS_PROC_CALL    0x00800000
166 #define I2C_FUNC_SMBUS_READ_BLOCK_DATA  0x01000000
167 #define I2C_FUNC_SMBUS_WRITE_BLOCK_DATA 0x02000000
168 #define I2C_FUNC_SMBUS_READ_I2C_BLOCK 0x04000000 /* I2C-like block xfer  */
```

```
169 #define I2C_FUNC_SMBUS_WRITE_I2C_BLOCK  0x08000000 /* w/ 1-byte reg. addr. */
170
171 #define I2C_FUNC_SMBUS_BYTE (I2C_FUNC_SMBUS_READ_BYTE | \
172                              I2C_FUNC_SMBUS_WRITE_BYTE)
173 #define I2C_FUNC_SMBUS_BYTE_DATA (I2C_FUNC_SMBUS_READ_BYTE_DATA | \
174                                   I2C_FUNC_SMBUS_WRITE_BYTE_DATA)
175 #define I2C_FUNC_SMBUS_WORD_DATA (I2C_FUNC_SMBUS_READ_WORD_DATA | \
176                                   I2C_FUNC_SMBUS_WRITE_WORD_DATA)
177 #define I2C_FUNC_SMBUS_BLOCK_DATA (I2C_FUNC_SMBUS_READ_BLOCK_DATA | \
178                                    I2C_FUNC_SMBUS_WRITE_BLOCK_DATA)
179 #define I2C_FUNC_SMBUS_I2C_BLOCK (I2C_FUNC_SMBUS_READ_I2C_BLOCK | \
180                                   I2C_FUNC_SMBUS_WRITE_I2C_BLOCK)
181
182 /* Old name, for compatibility */
183 #define I2C_FUNC_SMBUS_HWPEC_CALC I2C_FUNC_SMBUS_PEC
184
185 /*
186  * Data for SMBus Messages
187  */
188 #define I2C_SMBUS_BLOCK_MAX 32  /* As specified in SMBus standard */
189 #define I2C_SMBUS_I2C_BLOCK_MAX 32  /* Not specified but we use same structure*/
190 /*
191 union i2c_smbus_data {
192   __u8 byte;
193   __u16 word;
194   __u8 block[I2C_SMBUS_BLOCK_MAX + 2]; block[0] is used for length and
195                                                          //one more for PEC
196 }; */
197
198 /* smbus_access read or write markers */
199 #define I2C_SMBUS_READ  1
200 #define I2C_SMBUS_WRITE 0
201
202 /* SMBus transaction types (size parameter in the above functions)
203    Note: these no longer correspond to the (arbitrary) PIIX4 internal codes! */
204 #define I2C_SMBUS_QUICK         0
205 #define I2C_SMBUS_BYTE          1
206 #define I2C_SMBUS_BYTE_DATA       2
207 #define I2C_SMBUS_WORD_DATA       3
208 #define I2C_SMBUS_PROC_CALL       4
209 #define I2C_SMBUS_BLOCK_DATA      5
210 #define I2C_SMBUS_I2C_BLOCK_BROKEN  6
211 #define I2C_SMBUS_BLOCK_PROC_CALL   7   // SMBus 2.0
212 #define I2C_SMBUS_I2C_BLOCK_DATA    8
213
```

```
214
215 /* ----- commands for the ioctl like i2c_command call:
216  * note that additional calls are defined in the algorithm and hw
217  *  dependent layers - these can be listed here, or see the
218  *  corresponding header files.
219  */
220         /* -> bit-adapter specific ioctls */
221 #define I2C_RETRIES 0x0701  // number of times a device address
222         /* should be polled when not            */
223                                 // acknowledging
224 #define I2C_TIMEOUT 0x0702  // set timeout - call with int
225
226
227 /* this is for i2c-dev.c  */
228 #define I2C_SLAVE 0x0703  /* Change slave address     */
229         /* Attn.: Slave address is 7 or 10 bits */
230 #define I2C_SLAVE_FORCE 0x0706  /* Change slave address     */
231         /* Attn.: Slave address is 7 or 10 bits */
232         /* This changes the address, even if it */
233         /* is already taken!      */
234 #define I2C_TENBIT  0x0704  /* 0 for 7 bit addrs, != 0 for 10 bit */
235
236 #define I2C_FUNCS 0x0705  /* Get the adapter functionality */
237 #define I2C_RDWR  0x0707  /* Combined R/W transfer (one stop only)*/
238 #define I2C_PEC   0x0708  /* != 0 for SMBus PEC                   */
239
240 #define I2C_SMBUS 0x0720  /* SMBus-level access */
241
242 #define G_GAIN 0.00875        //[deg/s/LSB]
243 #define ACC_GAIN 0.002      //[g/LSB]
244 #define MAG_GAIN_XY 1100    //[LSB/gauss]
245 #define MAG_GAIN_Z 980      //[LSB/gauss]
246
247
248
249 /*---------------------------------------------------------------------------
250 Start: Modifiziert von Roman Kray
251 ---------------------------------------------------------------------------*/
252
253 int writeACC_to_data(FILE *fp_ACC, int max_freq, int freq_acc,
254                      int *loop_counter_acc, unsigned long int index_fastest);
255 int writeMAG_to_data(FILE *fp_MAG, int max_freq, int freq_mag,
256                      int *loop_counter_mag, unsigned long int index_fastest);
257 int writeGYR_to_data(FILE *fp_GYR, int max_freq, int freq_gyr,
258                      int *loop_counter_gyr, unsigned long int index_fastest);
```

```
259 int writePR_TE_to_data(int fd, FILE *fp_PR_TE, double *t_pressure, int max_freq,
260                       int freq_pr_te, int *loop_counter_pr_te,
261                       unsigned long int index_fastest);
262
263 int initializes_imu(void);
264
265 /*-------------------------------------------------------------------------------
266 Ende: Modifiziert von Roman Kray
267 -------------------------------------------------------------------------------*/
268
269 #endif // ADAFRUIT_IMU_H_INCLUDED
```

B.1.13. HeightLib.c

Entnommen aus Quelle [24].

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <stdint.h>
4  #include <math.h>
5
6  #include "HeightLib.h"
7
8  static FILE *height_fp;
9  static double height_halt_temp;
10 static int height_ramp_length;
11 static double height_pressure_average;
12 static double height_altitude_average;
13 static double height_pressure_temp;
14 static double height_altitude_temp;
15 static int height_door_open_time;
16
17 static double heightRampGetValue(int position);
18 static int heightWriteFileValue(double value);
19 static double heightCalculateHeight(double pressure, double refPressure);
20
21 // Diese Funktion dient zum initialisieren der Variablen sowie zum uebergeben
22 //des Filepointers.
23 // fp = Filepointer der Zieldatei, in die die Daten geschrieben werden sollen
24 // return = 1 wenn OK; -1 wenn Fehler (noch nicht implementiert)
25 int heightSetup(FILE *fp) {
26   height_fp = fp;
27   height_halt_temp = -1;
28   height_pressure_temp = -1;
29   height_altitude_temp = -1;
```

```
30    height_altitude_average = -1;
31    height_pressure_average = -1;
32    height_door_open_time = 0;
33
34    // in dieser Variable wird die Anzahl der Ticks gespeichert, während denen
35    // die Tuer geschlossen war. Dieser Wert wird dazu benutzt die Lineare Rampe
36    // zwischen zwei Haltestellen zu generieren.
37    height_ramp_length = 0;
38
39    return 1;
40  }
41
42  // Die heightSetData() Funktion muss jedes mal nach dem aufnehmen von Daten
43  //aufgerufen werden. Mittels dieser Funktion werden die Hoehendaten in
44  //abhaengigkeit des Tuersignals ausgewertet oder gespeichert.
45  // pressure = gemessener Druckwert in hPa
46  // halt = Haltestellensignal '1' oder '0'
47  void heightSetData(double pressure, char halt) {
48
49    int i;
50
51    // Tuer wurde geöffnet -> Haltestelle ----------------------------------------
52    if ((height_halt_temp == HEIGHT_TUER_GESCHLOSSEN) &&
53        (halt == HEIGHT_TUER_OFFEN)) {
54      // initialisieren des Drucks nachdem die Tuer das erste mal geoeffnet wurde
55      height_pressure_average = pressure;
56
57      // initialisieren der Tuer-Offen Zeit
58      height_door_open_time = 1;
59    }
60
61    // Tuer bleibt offen ---------------------------------------------------------
62    else if ((height_halt_temp == HEIGHT_TUER_OFFEN) &&
63             (halt == HEIGHT_TUER_OFFEN)) {
64      // Mittelwert des Drucks während der Haltestelle bilden
65      height_pressure_average = (height_pressure_average + pressure) / 2;
66
67      // erhoehen der Tuer offen Zeit
68      height_door_open_time++;
69    }
70
71    // Tuer wird wieder geschlossen ----------------------------------------------
72    else if ((height_halt_temp == HEIGHT_TUER_OFFEN) &&
73             (halt == HEIGHT_TUER_GESCHLOSSEN)) {
74
```

```
75      // pruefe ob es sich um den ersten Durchlauf handelt, wenn ja dann setzte
76      // Temp Druck gleich dem aktuellen Druck (erzeugt Ausgangshoehe = 0 Meter).
77      if (height_pressure_temp < 0) {
78        height_pressure_temp = height_pressure_average;
79      }
80
81      //berechne aus Druckwert die Hoehe. So geht es schneller als wenn jeder
82      //Rampenwert im Druck berechnet und anschliessend in die Hoehe umgerechnet
83      //werden muss.
84      height_altitude_average = heightCalculateHeight(height_pressure_average,
85                                                      height_pressure_temp);
86
87      // Fuehre die Rampenschleife NUR durch WENN bereits eine alte Hoehe gibt:
88      if (height_altitude_temp > 0) {
89        // Rampenschleife:
90        for (i = 0; i < height_ramp_length; i++) {
91      // return Wert ist bereits Hoehe in Meter, da umrechnung bereits erfolgt ist
92          heightWriteFileValue(heightRampGetValue(i));
93        }
94      }
95  //anschliessend schreibe den Wert des Durchschnitts der Haltestelle in die Datei
96      // Hierzu wird die Variable height_door_open_time benoetigt
97      for (i = 0; i < height_door_open_time; i++) {
98        heightWriteFileValue(height_pressure_average);
99      }
100
101     //Zwischenspeichern des Druckwertes als Referenzdruck fuer naechste Messung:
102     height_pressure_temp = height_pressure_average;
103
104     // setze height_ramp_length auf 1, da tuerkontakt aktuell bereits 0 ist
105     //(Tuer geschlossen)
106     height_ramp_length = 1;
107
108     // speichere aktuellen Druckwert als Temp-Wert fuer naechste Haltestelle
109     //(in Hiehe in Meter)
110     height_altitude_temp = height_pressure_average;
111   }
112
113   // Tuer ist geschlossen ------------------------------------------------------
114   else if ((height_halt_temp == HEIGHT_TUER_GESCHLOSSEN) &&
115            (halt == HEIGHT_TUER_GESCHLOSSEN)) {
116     height_ramp_length++;
117   }
118
119   // speichere alten tuerwert
```

```
120   height_halt_temp = halt;
121 }
122
123 // schliesst die Berechnung ab und schreibt alle noch fehlenden Werte
124 //in die Datei
125 // return = 1 wenn OK; -1 wenn Fehler (noch nicht implementiert)
126 int heightClose() {
127
128   int i;
129
130   // prüfe ob die Tuer offen oder geschlossen war beim letzten Wert
131   // Wenn JA dann berechne die Rampe und schreibe diese in die Datei,
132   // anschliessend schreibe für die Dauer, die die Tuer geoeffnet war den
133   //Durchschnittsdruck in die Datei.
134   if (height_halt_temp == HEIGHT_TUER_OFFEN) {
135
136     // berechne aus Druckwert die Hoehe. So geht es schneller als wenn jeder
137     // Rampenwert im Druck berechnet und anschliessend in die Hoehe umgerechnet
138     //werden muss.
139     height_pressure_average = heightCalculateHeight(height_pressure_average,
140                                                     height_pressure_temp);
141
142     // Wenn die Tuer offen war muss noch einmal die Rampe gebildet werden bis
143     // zu dem Moment, in dem die Tuer geoeffnet wurde:
144     for (i = 0; i < height_ramp_length; i++) {
145     // return Wert ist bereits Hoehe in Meter, da umrechnung bereits erfolgt ist
146       heightWriteFileValue(heightRampGetValue(i));
147     }
148
149     // anschliessend schreibe den Wert des Durchschnitts der Haltestelle in
150     // die Datei. Hierzu wird die Variable height_door_open_time benoetigt
151     for (i = 0; i < height_door_open_time; i++) {
152       heightWriteFileValue(height_pressure_average);
153     }
154
155   }
156     else {
157   // Ansonsten wenn die Tür geschlossen war (es ist davon auszugehen,
158   //dass der Bus stand).
159   // Deswegen kann der wert der letzten Haltestelle benutzt werden und über
160   // die Rampenlaenge geschrieben werden:
161   for (i = 0; i < height_ramp_length; i++) {
162     heightWriteFileValue(height_pressure_temp);
163   }
164 }
```

```
165
166   return 1;
167 }
168
169 // Diese Funktion berechnet eine lineare Rampe zwischen den als Globalen
170 // Variablen gespeicherten.
171 // Start-, Endwert und der länge der Rampe. Dabei werden folgende Globale
172 // Variablen benutzt:
173 // laenge der Rampe = height_ramp_length
174 // start = height_open_average
175 // ende = height_close_average
176 // Die Parameter der Funktion lauten:
177 // position = Aktuelle Position fuer die ein Wert berechnet werden soll
178 // return = Wert an der Stelle position
179 double heightRampGetValue(int position) {
180
181   double m; // steigung
182
183   m = (height_pressure_temp - height_pressure_average) / height_ramp_length;
184
185   return (m*position) + height_pressure_temp;
186 }
187
188 // schreibt den uebergebenen Wert in eine Binärdatei mit dem Filepointer,
189 // welcher durch die
190 // heightSetup() Funktion uebergeben wurde
191 // value = Wert, der geschrieben werden soll
192 // return = 1 wenn OK; -1 wenn Fehler (noch nicht implementiert)
193 int heightWriteFileValue(double value) {
194
195     char test[20];
196   //fwrite(&value, sizeof(value), 1, height_fp);
197   sprintf(test,"%.4lf\n",value);
198   fputs(test,height_fp);
199
200   return 1;
201 }
202
203 // Diese Funktion berechnet die Hoehe in Meter. Dazu muss der aktuelle Druck
204 // sowie der Referenzdruck uebergeben werden.
205 // Siehe BOSCH BMP180 Datenblatt Seite 16
206 // double pressure = aktueller Druck
207 // double refPressure = Referenz bzw. Bezugsdruck
208 // return = Hoehe in Meter
209 double heightCalculateHeight(double pressure, double refPressure) {
```

```
210
211   return 44330 * (1 - pow((pressure / refPressure), (1 / 5.255)));
212 }
```

B.1.14. HeightLib.h

Entnommen aus Quelle [24].

```
1  #ifndef HEIGHTLIB_H
2  #define HEIGHTLIB_H
3
4  #define HEIGHT_BUFFER_LEN 100
5  #define HEIGHT_TUER_OFFEN '1'
6  #define HEIGHT_TUER_GESCHLOSSEN '0'
7
8  int heightSetup(FILE *fp);
9  void heightSetData(double pressure, char halt);
10 int heightClose(void);
11
12
13 #endif
```

B.2. Auswärtungs-Programm

Die aus der Quelle [1] übernommenen Programmabschnitte wurden gekennzeichnet.

```
1
2  #include <stdio.h>
3  #include <stdlib.h>
4  #include <string.h>
5  #include <sys/stat.h>
6  #include <errno.h>
7  #include <time.h>
8
9  #ifndef S_SPLINT_S
10 #include "wiringPi.h"
11 #endif
12 #include "DataCalc.h"
13 #include "ErrorStorage.h"
14 #include "global.h"
15 #include "LowFuncCalc.h"
16
17 /*@null@*/FILE *fp_ERROR; //Fehlerspeicher.
18 int door_wrong;           //Laufvariablen für zeitlich falsche Werte.
19 int tacho_wrong;
20 int gga_wrong;
21 int rmc_wrong;
22
23 int main(void)
24 {
25     char wrong_sensor_values[100];  //Zusammenfassung der zeitlichen Fehler.
26     unsigned int old_ts_door;       //vorherige Timestemps.
27     unsigned int old_ts_tacho;
28     unsigned int old_ts_gga;
29     unsigned int old_ts_rmc;
30
31     unsigned int gga_period;  //Zeit zwischen Messwerten in ms.
32     unsigned int rmc_period;
33     unsigned int door_period;
34     unsigned int tacho_period;
35
36     unsigned long int index_gps;        //aktueller Index aus GPS File.
37     unsigned long int points_gps;       //gesmate Anzahl der GPS Messpunkte.
38     unsigned long int j;                //Lfv für Auslese-for-Schleife.
39
40     int watchdog_out;         //HIGH oder LOW. Lebenszeichen.
41     int loop_counter_watchdog_output;  //steuert Ausführungsfreq der WDFkt.
42
```

```
43    char fdir[23 + 1];  //Erstets Verzeichnis ohne Calc.log.
44    char mat_string[33 +1];   //Name mat-File.
45    char fdir_plus_mat_string[23 + 33 + 1];   //Location mat-File.
46    char void_string[300 + 1];
47    int second_rmc_success;
48
49    /*@null@*/double *sensor_arrays[MAX_ARRAYS]; //alle Sensor-Arrays.
50  //index:    0        1        2                      3
51  //        { Venus_Y, Venus_X, Venus_Used_satellites, Venus_Hoehe,
52  //4                      5               6           7            8
53  //Venus_Geschwindigkeit, Venus_Direction, Venus_Time, Haltestelle, tick }
54
55    /*@null@*/FILE *fp_LOG;  //markiert Verzeichniss als bearbeitet.
56    /*@null@*/FILE *all_files[MAX_FILES];  //alle fp in einem Array.
57  //index:                0             1           2
58    char *file_names[] = {"POINTS.bin", "TIME.bin", "GPS_GGA.nmea",
59  //3           4
60    "DOOR.txt", "TACHO.txt"}; //alle Dateinamen.
61  //index:                0             1           2
62    char *file_mode[]  = {"rb"        , "rb"      , "r"           ,
63  //3           4
64    "r"       , "r"        }; //alle Öffnungsmodi.
65
66    double timevector[6]; //speichert Ihnhalt von TIME.bin (Logging-Startzeit).
67
68    int i;  //lfv für komplettes Programm.
69    int m;  //Initialisierung timevector.
70
71    /*@unused@*/struct timespec my_tic = {0};  //Zeitmessung zum Debugging.
72    /*@unused@*/struct timespec my_toc = {0};
73
74    for(i = 0; i < 100; i++)  //for loop für das komplette Programm.
75    {
76        door_wrong = 0;
77        tacho_wrong = 0;
78        gga_wrong = 0;
79        rmc_wrong = 0;
80
81        strcpy(wrong_sensor_values, "");
82        old_ts_door = 1000; // erster TS Vergleich wird ausgelassen.
83        old_ts_tacho = 1000;
84        old_ts_gga = 1000;
85        old_ts_rmc = 1000;
86
87        gga_period = 0;
```

```
88          rmc_period = 0;
89          door_period = 0;
90          tacho_period = 0;
91
92          index_gps = 0;
93          points_gps = 0;
94          j = 0;
95
96          watchdog_out = 0;
97          loop_counter_watchdog_output = 0;
98
99          strcpy(fdir, "/media/INTENSO/Data000/");
100         mat_string[0] = '\0';
101         fdir_plus_mat_string[0] = '\0';
102         void_string[0] = '\0';
103         second_rmc_success = 0;
104
105         //initilase all_files and sensor_arrays
106         for(m = 0; m < 6; m++)
107         {
108             timevector[m] = 0.0;
109         }
110
111         (void) printf("\n##################################################\n"
112                 "GPS_Calc start----------\n"
113                 "##################################################\n");
114
115         if(initializes_gpio() == EXIT_FAILURE)
116         {
117             save_error("->initializes_gpio()\n");
118             continue;
119         }
120         if(getdir(fdir) == EXIT_FAILURE)
121         {
122             (void) printf("->getdir()\n");
123             continue;
124         }
125         if((fp_LOG = openfile(fdir, "CALC.log", "wb")) == NULL)
126         {
127             (void) printf("\nExistiert nicht."
128                           " Alle Rohdaten sind ausgewertet.\n");
129             (void) printf("\n##################################################\n"
130                     "GPS_Calc beendet----------\n"
131                     "##################################################\n");
132             return EXIT_SUCCESS; //riskant openfile könnte auch so fehlschlagen
```

```
133         }
134         if(fclose(fp_LOG) == EOF)
135         {
136             (void) printf("%s", strerror(errno));
137             (void) printf("->fclose(fp_LOG)");
138             continue;
139         }
140         if((fp_ERROR = openfile(fdir, "ERROR_CALC.txt", "w")) == NULL)
141         {
142             (void) printf("->openfile(fp_ERROR)");
143             continue;
144         }
145         if(open_all_files(fdir, all_files, file_names,
146                           file_mode) == EXIT_FAILURE)
147         {
148             (void) printf("->open_all_files()\n");
149             (void) close_all_files(all_files);
150             (void) fclose(fp_ERROR);
151             continue;
152         }
153 
154         save_error("GPS_CALC\n");
155         save_error(fdir);
156         save_error(":\n");
157 
158         if(read_points(all_files[0], &points_gps) == EXIT_FAILURE)
159         {
160             save_error("->read_points()\n");
161             (void) close_all_files(all_files);
162             (void) fclose(fp_ERROR);
163             continue;
164         }
165         if(allocate_all_arrays(points_gps, sensor_arrays) == EXIT_FAILURE)
166         {
167             save_error("->allocate_all_arrays()\n");
168             (void) free_all_arrays(sensor_arrays);
169             (void) close_all_files(all_files);
170             (void) fclose(fp_ERROR);
171             continue;
172         }
173 
174         if(save_time(all_files[1], mat_string, timevector) == EXIT_FAILURE)
175         {
176             save_error("->save_time\n");
177             (void) free_all_arrays(sensor_arrays);
```

```
178             (void) close_all_files(all_files);
179             (void) fclose(fp_ERROR);
180             continue;
181         }
182 
183         (void) printf("\n-------- CALC STARTEN-----------\n");
184         for(j = 0; j < points_gps; j++)
185         {
186             //clock_gettime(CLOCK_MONOTONIC, &my_tic);
187             //vertauschte GGA/RMC Reihenfolge wird nun erkannt.
188             if(read_gga(all_files[2], j, &old_ts_gga, &gga_period, &index_gps,
189                         sensor_arrays[0], sensor_arrays[1], sensor_arrays[2],
190                         sensor_arrays[3]) == EXIT_FAILURE)
191             {
192                 save_error("->read_gga(GGA/RMC haben Position getauscht)\n");
193                 skip_value(&sensor_arrays[0][j], &sensor_arrays[1][j],
194                            &sensor_arrays[2][j], &sensor_arrays[3][j]);
195                 if(read_rmc_info(all_files[2], &sensor_arrays[4][j],
196                                  &sensor_arrays[5][j]) == EXIT_FAILURE)
197                 {
198                     save_error("->read_rmc(fehlerhafter GGA-String)\n");
199                     skip_value(&sensor_arrays[4][j], &sensor_arrays[5][j],
200                                NULL, NULL);
201                     if(read_file_till_char(all_files[2], '$', void_string,
202                                            300 + 1) == EXIT_FAILURE)
203                     {
204                         save_error("->read_file_till_char($ Wert überspringen)");
205                         continue;
206                     }
207                 }
208                 else
209                 {
210                     second_rmc_success = 1;
211                 }
212             }
213 
214             if(second_rmc_success == 0)
215             {
216                 if(read_rmc(all_files[2], j, &old_ts_rmc, &rmc_period,
217                             &index_gps, sensor_arrays[4],
218                             sensor_arrays[5]) == EXIT_FAILURE)
219                 {
220                     save_error("->read_rmc(GGA/RMC haben Position getauscht)\n");
221                     skip_value(&sensor_arrays[4][j], &sensor_arrays[5][j],
222                                NULL, NULL);
```

```
223                     if(read_gga_info(all_files[2], &sensor_arrays[0][j],
224                                     &sensor_arrays[1][j], &sensor_arrays[2][j],
225                                     &sensor_arrays[3][j]) == EXIT_FAILURE)
226                     {
227                         save_error("->read_gga(fehlerhafter RMC-String)\n");
228                         if(read_file_till_char(all_files[2], '$', void_string,
229                                                 300 + 1) == EXIT_FAILURE)
230                         {
231                             save_error("->read_file_till_char($ Wert überspringen)");
232                             continue;
233                         }
234                     }
235                 }
236             }
237             else
238             {
239                 save_error("->EXTREME Verdrehung\n");
240                 second_rmc_success = 0;
241                 if(read_gga(all_files[2], j, &old_ts_gga, &gga_period, &index_gps,
242                             sensor_arrays[0], sensor_arrays[1], sensor_arrays[2],
243                             sensor_arrays[3]) == EXIT_FAILURE)
244                 {
245                     skip_value(&sensor_arrays[0][j], &sensor_arrays[1][j],
246                                &sensor_arrays[2][j], &sensor_arrays[3][j]);
247                     if(read_file_till_char(all_files[2], '$', void_string,
248                                             300 + 1) == EXIT_FAILURE)
249                     {
250                         save_error("->read_file_till_char($ Wert überspringen)");
251                         continue;
252                     }
253
254                 }
255             }
256
257             if(read_time(j, &gga_period, sensor_arrays[6]) == EXIT_FAILURE)
258             {
259                 save_error("->read_time()\n");
260                 skip_value(&sensor_arrays[6][j], NULL, NULL, NULL);
261             }
262             if(read_door(all_files[3], j, &old_ts_door, &door_period,
263                          &index_gps, sensor_arrays[7]) == EXIT_FAILURE)
264             {
265                 save_error("->read_door()\n");
266                 skip_value(&sensor_arrays[7][j], NULL, NULL, NULL);
267                 if(read_file_till_char(all_files[3], '$', void_string,
```

```
                                        300 + 1) == EXIT_FAILURE)
                {
                    save_error("->read_file_till_char($ Wert überspringen)");
                    continue;
                }
            }
            if(read_tacho(all_files[4], j, &old_ts_tacho, &tacho_period,
                        &index_gps, sensor_arrays[8]) == EXIT_FAILURE)
            {
                save_error("->read_tacho()\n");
                skip_value(&sensor_arrays[8][j], NULL, NULL, NULL);
                if(read_file_till_char(all_files[4], '$', void_string,
                                        300 + 1) == EXIT_FAILURE)
                {
                    save_error("->read_file_till_char($ Wert überspringen)");
                    continue;
                }
            }
            if(watchdog_output(&watchdog_out,
                               &loop_counter_watchdog_output) == EXIT_FAILURE)
            {
                save_error("->watchdog_out()\n");
                continue;
            }
            #ifdef PRINT_TO_SCREEN
            (void) printf("\nY: %f, X: %f, sat: %f, "
            "Z: %f\n", sensor_arrays[0][j],
            sensor_arrays[1][j], sensor_arrays[2][j], sensor_arrays[3][j]);
            (void) printf("v: %f, dir: %f\n", sensor_arrays[4][j],
            sensor_arrays[5][j]);
            (void) printf("t: %f\n", sensor_arrays[6][j]);
            (void) printf("stop: %f, tick: %f\n", sensor_arrays[7][j],
            sensor_arrays[8][j]);
            #endif

//            clock_gettime(CLOCK_MONOTONIC, &my_toc);
//            (void) printf("time for loop: %f\n", calc_time(my_tic, my_toc));
        }

        (void) printf("\n-------- CALC ABGESCHLOSSEN-----------\n");

        if(snprintf(wrong_sensor_values, 20 + 10 + 30 + 10 + 7 + 10 + 9 + 10 +
           8 + 10 + 2, "Anzahl GPS-Messung: %lu\nFehlerhafte Messpunkte->"
           "GGA: %d, RMC: %d, TACHO: %d, DOOR: %d\n",
           points_gps, gga_wrong, rmc_wrong, tacho_wrong, door_wrong) <= 0)
```

```
313         {
314             save_error(strerror(errno));
315             save_error("->snprintf(wrong_sensor_values)");
316             (void) free_all_arrays(sensor_arrays);
317             (void) close_all_files(all_files);
318             (void) fclose(fp_ERROR);
319             continue;
320         }
321         save_error(wrong_sensor_values);
322 
323         strcpy(fdir_plus_mat_string, fdir);
324         strcat(fdir_plus_mat_string, mat_string);
325         #ifndef S_SPLINT_S
326         if(create_mat_file(j, fdir_plus_mat_string, sensor_arrays,
327                            timevector) == EXIT_FAILURE)
328         {
329             save_error("->create_mat_file()\n");
330             (void) free_all_arrays(sensor_arrays);
331             (void) close_all_files(all_files);
332             (void) fclose(fp_ERROR);
333             continue;
334         }
335         #endif
336         if(free_all_arrays(sensor_arrays) == EXIT_FAILURE)
337         {
338             save_error("->free_all_arrays()\n");
339             (void) close_all_files(all_files);
340             (void) fclose(fp_ERROR);
341             continue;
342         }
343         if(close_all_files(all_files) == EXIT_FAILURE)
344         {
345             save_error("->close_all_files()\n");
346             (void) fclose(fp_ERROR);
347             continue;
348         }
349         if(zip_files(fdir) == EXIT_FAILURE)
350         {
351             save_error("->zip_files()\n");
352             (void) fclose(fp_ERROR);
353             continue;
354         }
355         if(fclose(fp_ERROR) == EOF)
356         {
357             (void) printf("%s", strerror(errno));
```

```
358             (void) printf("->fclose(fp_ERROR)");
359             continue;
360         }
361         (void) printf("\n################################################\n"
362                 "GPS_Calc beendet----------\n"
363                 "################################################\n");
364     }
365     (void) printf("\n100 Versuche fehlgeschlagen\n");
366     return EXIT_FAILURE;
367 }
```

B.2.1. DataCalc.c

Die aus der Quelle [1] übernommenen Programmabschnitte wurden gekennzeichnet.

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4  #include <math.h>
5  #include <errno.h>
6  #include <time.h>
7
8  #ifndef S_SPLINT_S
9  #include "matio.h"
10 #endif
11 #include "DataCalc.h"
12 #include "ErrorStorage.h"
13 #include "LowFuncCalc.h"
14 #ifndef S_SPLINT_S
15 #include "wiringPi.h"
16 #endif
17 /*---------------------------------------------------------------------------
18 Initialisierung des Watchdog-GPIO plus Interrupt Service Routine (ISR).
19 (Erstellt von Mario Wegner)
20
21 Parameter:  void
22
23 Return: EXIT_SUCCESS, wenn erfolgreich.
24         EXIT_FAILURE, wenn wiringPISetup() oder wiringPiISR() fehlschlägt.
25 ---------------------------------------------------------------------------*/
26 int initializes_gpio(void)
27 {
28     (void) printf("\nInitialisiere GPIO:\n");
29     #ifndef S_SPLINT_S
30     if (wiringPiSetup() < 0)
```

```
31      {
32          (void) printf("\twiringPiSetup()\t\t\t\t\t\t[  "err" ]\n");
33          save_error(strerror(errno));
34          save_error("->wiringPiSetup()");
35          return EXIT_FAILURE;
36      }
37      else {(void) printf("\twiringPiSetup()\t\t\t\t\t\t[  "ok"  ]\n");}
38
39      pinMode(WD_OUTPUT, OUTPUT);     //Watchdog.
40
41      digitalWrite(WD_OUTPUT, LOW);     //Lebenszeichen.
42      milli_sleep(250.0);
43      digitalWrite(WD_OUTPUT, HIGH);
44      milli_sleep(250.0);
45      digitalWrite(WD_OUTPUT, LOW);
46      #endif
47      (void) printf("Initialisiere GPIO abgeschlossen.\n");
48
49      return EXIT_SUCCESS;
50  }
51
52
53  /*------------------------------------------------------------------------
54  Sucht nicht bearbeitetes (ohne Calc.log) Folgeverzeichnis.
55  Kein save_error(). ERROR_STORAGE.txt noch nicht erstellt.
56  (Erstellt von Mario Wegner)
57
58  Parameter: *fdir    Start-Verzeichniss.
59
60  Return: EXIT_SUCCESS, wenn erfolgreich.
61          EXIT_FAILURE, wenn maxinale Verzeichnissanzahl erreicht.
62  ------------------------------------------------------------------------*/
63  int getdir(char *fdir)
64  {
65      size_t fdir_length = strlen(fdir);
66      size_t length = fdir_length - 2;
67      char temp1[fdir_length + 1];
68
69      (void) printf("\nSuche nächstes nicht bearbeitetes Verzeichniss:\n");
70      strcpy(temp1, fdir);
71      while(openfile(temp1, "CALC.log", "r") != NULL)
72      {     //(void) printf("%s\n",fdir);
73          if(temp1[length] == '9')
74          {
75              temp1[length] = '0';
```

```
76              if(temp1[length - 1] == '9')
77              {
78                  temp1[length - 1] = '0';
79                  if(temp1[length - 2] == '9')
80                  {
81                      (void) printf("Maximale Anzahl an Verzeichnissen erreicht");
82                      return EXIT_FAILURE;
83                  }
84                  else
85                  {
86                      temp1[length - 2]++;
87                  }
88              }
89              else
90              {
91                  temp1[length - 1]++;
92              }
93          }
94          else
95          {
96              temp1[length]++;
97          }
98
99      strncpy(fdir, temp1, 22);
100     }
101     (void) printf("\t%s\n", fdir);
102
103     return EXIT_SUCCESS;
104 }
105
106 /*-------------------------------------------------------------------------
107 Öffnet und erstellt Datei.
108 Kein save_error(). ERROR_STORAGE.txt noch nicht erstellt.
109 (Erstellt von Mario Wegner)
110
111 Parameter: *fdir        Wo soll die Datei gespeichert werden?
112            *filename    Dateiname.
113            *mode        In welchem Modus soll die Datei geöffnet werden?
114                         "w" schreiben, "wb" schreiben binär.
115
116 Return: Pointer auf Datei wenn erfolgreich.
117         NULL wenn malloc-Fehler oder Datei konnte nicht geoeffnet werden.
118 -------------------------------------------------------------------------*/
119 FILE *openfile(char *fdir, char *filename, char *mode)
120 {
```

```
121     FILE *fp;
122     char *filepath;
123 
124     filepath = concat(fdir, filename);
125     if(filepath == NULL)
126     {
127         (void) printf("->concat()");
128         return NULL;
129     }
130     fp = fopen(filepath, mode);
131     if(fp == NULL)
132     {
133         (void) printf("%s", strerror(errno));
134         (void) printf("->fopen()");
135         free(filepath);
136         filepath = NULL;
137         return NULL;
138     }
139 
140     free(filepath);
141     filepath = NULL;
142 
143     return fp;
144 }
145 
146 /*----------------------------------------------------------------------------
147 Öffnet alle FILES.
148 Kein save_error(). ERROR_STORAGE.txt noch nicht erstellt.
149 
150 Parameter: *fdir         Wo sollen die Datei geöffnet werden?
151            **all_files   Array mit allen File-Pointern.
152            **file_names  Array mit allen File-Namen-Strings.
153            **file_mode   Array mit allen File-Modi-Strings.
154 
155 Return: EXIT_SUCCESS, wenn erfolgreich.
156         EXIT_FAILURE, wenn openfile() fehlschlägt.
157 ----------------------------------------------------------------------------*/
158 int open_all_files(char *fdir, FILE **all_files, char **file_names,
159                    char **file_mode)
160 {
161     int i = 0;
162 
163     for(i = 0; i < MAX_FILES; i++)
164     {
165         if((all_files[i] = openfile(fdir, file_names[i], file_mode[i])) == NULL)
```

```
166         {
167             (void) printf("->openfile()");
168             return EXIT_FAILURE;
169         }
170     }
171 
172     return EXIT_SUCCESS;
173 }
174 
175 /*---------------------------------------------------------------------------
176 Lie ß t Anzahl der GPS-Pakete (Messpunkte) aus Datei.
177 
178 Parameter: *fp_POINTS    Datei, aus der die Anzahl gelesen wird.
179            *points_gps   Variable, in der die Anzahl gespeichert wird.
180 
181 Return: EXIT_SUCCESS, wenn erfolgreich.
182         EXIT_FAILURE, wenn fread() fehlschlägt.
183 ---------------------------------------------------------------------------*/
184 int read_points(FILE *fp_POINTS, unsigned long int *points_gps)
185 {
186     if(fread(points_gps, sizeof(unsigned long int), 1, fp_POINTS) != 1)
187     {
188         (void) printf("\nAnzahl GPS-Messung (%lu) aus POINTS.bin lesen"
189                "\t\t\t[  "err" ]\n", *points_gps);
190         save_error(strerror(errno));
191         save_error("->fread()");
192         return EXIT_FAILURE;
193     }
194     else {(void) printf("\nAnzahl GPS-Messung (%lu) aus POINTS.bin lesen"
195                  "\t\t\t[  "ok"  ]\n", *points_gps);}
196 
197     return EXIT_SUCCESS;
198 }
199 
200 /*---------------------------------------------------------------------------
201 Allozieret alle Arrays.
202 
203 Parameter: points_gps       Anzahl GPS Messungen.
204            **sensor_arrays  Array, der alle Sensor-Arrays enthält.
205 
206 Return: EXIT_SUCCESS, wenn erfolgreich.
207         EXIT_FAILURE, wenn malloc() fehlschlägt.
208 ---------------------------------------------------------------------------*/
209 int allocate_all_arrays(unsigned long int points_gps, double **sensor_arrays)
210 {
```

```
211     int i = 0;
212
213     for(i = 0; i < MAX_ARRAYS; i++)
214     {
215         if((sensor_arrays[i] = malloc((size_t) points_gps * sizeof(double)))
216                                 == NULL)
217         {
218             save_error(strerror(errno));
219             save_error("->malloc()");
220             return EXIT_FAILURE;
221         }
222     }
223
224     return EXIT_SUCCESS;
225 }
226
227 /*------------------------------------------------------------------------------
228 Lie ß t Zeit (Format: YYYYMMDDhhmmss) aus binär Datei
229 und erstellt daraus MAT-File-Name (Format: HH_11_7301_20160507_2036_xxxx.mat).
230 (Erstellt von Mario Wegner)
231
232 Parameter: *fp_TIME      Aus welcher binär Datei soll die Zeit gelesen werden?
233            *mat_string   String, in dem der MAT-File-Name gespeichert wird.
234            *timevector   String, in dem die Zeit aus TIME.bin gespeichert wird.
235                          Wird zur Erstellung des MAT-Files noch benötigt.
236            *lognummer    Logger-Number zweistellig.
237            *busnummer    Bus-Number vierstellig.
238
239 Return: EXIT_SUCCESS, wenn erfolgreich.
240         EXIT_FAILURE, wenn fread() fehlschlägt.
241 ------------------------------------------------------------------------------*/
242 int save_time(FILE *fp_TIME, char *mat_string, double *timevector)
243 {
244     char year[5], month[3], day[3], hour[3], min[3];
245
246     (void) printf("\nStartzeit aus Time.bin lesen:\n");
247     if(fread(timevector, sizeof(double), 6, fp_TIME) != 6)
248     {
249         save_error(strerror(errno));
250         save_error("->fread()");
251         return EXIT_FAILURE;
252     }
253     if(snprintf(day, 2 + 1, "%02d", (int)timevector[2]) <= 0)     //2 + '\0'.
254     {
255         save_error(strerror(errno));
```

```
256         save_error("->snprintf(day)");
257         return EXIT_FAILURE;
258     }
259     if(snprintf(month, 2 + 1, "%02d", (int)timevector[1]) <= 0)
260     {
261         save_error(strerror(errno));
262         save_error("->snprintf(month)");
263         return EXIT_FAILURE;
264     }
265     if(snprintf(year, 4 + 1, "%d", (int)timevector[0]) <= 0)
266     {
267         save_error(strerror(errno));
268         save_error("->snprintf(year)");
269         return EXIT_FAILURE;
270     }
271     if(snprintf(hour, 2 + 1, "%02d", (int)timevector[3]) <= 0)
272     {
273         save_error(strerror(errno));
274         save_error("->snprintf(hour)");
275         return EXIT_FAILURE;
276     }
277     if(snprintf(min, 2 + 1, "%02d", (int)timevector[4]) <= 0)
278     {
279         save_error(strerror(errno));
280         save_error("->snprintf(min)");
281         return EXIT_FAILURE;
282     }
283     strncat(mat_string, "HH_"        , 3);
284     strncat(mat_string, lognum       , 2);
285     strncat(mat_string, "_"          , 1);
286     strncat(mat_string, busnum       , 4);
287     strncat(mat_string, "_"          , 1);
288     strncat(mat_string, year         , 4);
289     strncat(mat_string, month        , 2);
290     strncat(mat_string, day          , 2);
291     strncat(mat_string, "_"          , 1);
292     strncat(mat_string, hour         , 2);
293     strncat(mat_string, min          , 2);
294     strncat(mat_string, "_"          , 1);
295     strncat(mat_string, "xxxx"       , 4);
296     strncat(mat_string, ".mat"       , 4);
297     (void) printf("\t%s\n", mat_string);
298
299     return EXIT_SUCCESS;
300 }
```

```
301
302 /*-------------------------------------------------------------------------
303 Lie ß t Timestamp (plus TS-Vergleich) + Zählindex + GGA-String aus Datei.
304 Au ß erdem werden Sensorwerte aus GGA-Sting in die jeweiligen Arrays geschrieben.
305
306 Parameter: *fp_GGA                      Datei, aus der das GPS_Paket gelesen wird.
307            j                            Array-Index.
308            *old_ts_gga                  vorheriger Timestamp.
309            *gga_period                  TS-Differenz.
310            *index_gps                   Zähl-Index von GPS Werten.
311            *Venus_Used_satellites       Sensor-Werte.
312            *Venus_Y                     x
313            *Venus_X                     x
314            *Venus_Hoehe                 x
315
316 Return: EXIT_SUCCESS, wenn erfolgreich.
317         EXIT_FAILURE, wenn Fehler in subfunc.
318 -------------------------------------------------------------------------*/
319 int read_gga(FILE *fp_GGA, unsigned long int j, unsigned int *old_ts_gga,
320              unsigned int *gga_period, unsigned long int *index_gps,
321              double *Venus_Y, double *Venus_X, double *Venus_Used_satellites,
322              double *Venus_Hoehe)
323 {
324     if(compare_timestamp(fp_GGA, old_ts_gga, gga_period, "GGA") == EXIT_FAILURE)
325     {
326         save_error("->compare_timestamp()");
327         return EXIT_FAILURE;
328     }
329     if(read_count_index(fp_GGA, index_gps) == EXIT_FAILURE)
330     {
331         save_error("->read_count_index()");
332         return EXIT_FAILURE;
333     }
334     if(read_gga_info(fp_GGA, &Venus_Y[j], &Venus_X[j],
335                      &Venus_Used_satellites[j], &Venus_Hoehe[j]) == EXIT_FAILURE)
336     {
337         save_error("->read_gga_info()");
338         return EXIT_FAILURE;
339     }
340     //$GPGGA,022501.200,5326.7383,N,00959.5293,E,0,00,0.0,134.4,M,0.0,M,,0000*6C
341
342     return EXIT_SUCCESS;
343 }
344
345 /*-------------------------------------------------------------------------
```

```
346 Lie ß t  Zählindex und RMC-String aus Datei.
347 Vergleicht Zählindex mit GPS-Zählindex
348 und schreibt Sensorwerte aus RMC-Sting in die jeweiligen Arrays.
349
350 Parameter: *fp_GGA                    Datei, aus der das GPS_Paket gelesen wird.
351            j                          Array-Index.
352            *index_gps                 Zähl-Index von GPS Werten.
353            *Venus_Geschwindigkeit     Sensor-Werte.
354            *Venus_Direction           x
355
356 Return: EXIT_SUCCESS, wenn erfolgreich.
357         EXIT_FAILURE, wenn Fehler in subfunc.
358 ------------------------------------------------------------------------------*/
359 int read_rmc(FILE *fp_GGA, unsigned long int j, unsigned int *old_ts_rmc,
360              unsigned int *rmc_period, unsigned long int *index_gps,
361              double *Venus_Geschwindigkeit, double *Venus_Direction)
362 {
363     unsigned long int index_rmc = 0;
364
365     if(compare_timestamp(fp_GGA, old_ts_rmc, rmc_period, "RMC") == EXIT_FAILURE)
366     {
367         save_error("->compare_timestamp()");
368         return EXIT_FAILURE;
369     }
370     if(read_count_index(fp_GGA, &index_rmc) == EXIT_FAILURE)
371     {
372         save_error("->read_count_index()");
373         return EXIT_FAILURE;
374     }
375     if(index_rmc != *index_gps)
376     {
377         save_error("gga und rmc asynchron");
378         return EXIT_FAILURE;
379     }
380     if(read_rmc_info(fp_GGA, &Venus_Geschwindigkeit[j], &Venus_Direction[j])
381                      == EXIT_FAILURE)
382     {
383         save_error("->read_rmc_info()");
384         return EXIT_FAILURE;
385     }
386     //$GPRMC,022501.300,V,5326.7383,N,00959.5293,E,000.0,000.0,211015,,,N*72
387
388     return EXIT_SUCCESS;
389 }
390
```

```
391 /*-----------------------------------------------------------------------------
392 Erstellt Time-Array auf Basis des TS-Vergleichs und
393 speichert diesen in Venus_Time.
394
395 Parameter: j            Array-Index.
396            *period      TS-Differenz aus read_gga.
397            *Venus_Time  Sensor-Wert.
398
399 Return: EXIT_SUCCESS, wenn erfolgreich.
400         EXIT_FAILURE, wenn Fehler in subfunc.
401 -----------------------------------------------------------------------------*/
402 int read_time(unsigned long int j, unsigned int *period, double *Venus_Time)
403 {
404     if (j == 0)
405     {
406         Venus_Time[0] = 0.0;
407     }
408     else
409     {
410         Venus_Time[j] = Venus_Time[j - 1] + ((double) *period / 1000.0);
411     }
412
413     return EXIT_SUCCESS;
414 }
415
416 /*-----------------------------------------------------------------------------
417 Lie ß t Timestamp (plus TS-Vergleich) + Zählindex + DOOR-String aus Datei,
418 solange bis Zählindex mit GPS-Zählindex übereinstimmt.
419 Dann wird Sensorwert aus DOOR-Sting in Array geschrieben.
420
421 Parameter: *fp_DOOR                  Datei, aus der das GPS_Paket gelesen wird.
422            j                         Array-Index.
423            *old_ts_door              vorheriger Timestamp.
424            *door_period              TS-Differenz.
425            *index_gps                Zähl-Index von GPS Werten.
426            *Haltestelle              Sensor-Werte.
427
428 Return: EXIT_SUCCESS, wenn erfolgreich.
429         EXIT_FAILURE, wenn Fehler in subfunc.
430 -----------------------------------------------------------------------------*/
431 int read_door(FILE *fp_DOOR, unsigned long int j, unsigned int *old_ts_door,
432               unsigned int *door_period, unsigned long int *index_gps,
433               double *Haltestelle)
434 {
435     unsigned long int index_door = 0;
```

```
436     char door_string[3 + 1] = "";
437     char void_string[1 + 1] = "";
438     int error_strto = EXIT_SUCCESS;
439
440     do
441     {
442         if(compare_timestamp(fp_DOOR, old_ts_door, door_period, "DOOR")
443                     == EXIT_FAILURE)
444         {
445             save_error("->compare_timestamp()");
446             return EXIT_FAILURE;
447         }
448         if(read_count_index(fp_DOOR, &index_door) == EXIT_FAILURE)
449         {
450             save_error("->read_count_index()");
451             return EXIT_FAILURE;
452         }
453         if(read_file_till_char(fp_DOOR, '\n', door_string, 2 + 1)
454                                 == EXIT_FAILURE)
455         {
456             save_error("->read_file_till_char(\n)");
457             return EXIT_FAILURE;
458         }
459         if(read_file_till_char(fp_DOOR, '$', void_string, 1 + 1)
460                                 == EXIT_FAILURE)
461         {
462             save_error("->read_file_till_char($)");
463             return EXIT_FAILURE;
464         }
465      //(void) printf("door_string: %s\tLänge: %i\n", door_string, strlen(door_string));
466     } while(index_door != *index_gps);
467     if(door_string[0] != '1' && door_string[0] != '0')
468     {
469         save_error("DOOR konnte nicht eingelesen werden.");
470         return EXIT_FAILURE;
471     }
472     else
473     {
474         Haltestelle[j] = my_str_to_double(door_string, &error_strto) * 9;
475         if(error_strto == EXIT_FAILURE)
476         {
477             save_error("->my_str_to_int(door_string)");
478             return EXIT_FAILURE;
479         }
480     }
```

```
481
482     return EXIT_SUCCESS;
483 }
484
485 /*-----------------------------------------------------------------------------
486 Lie ß t Timestamp (plus TS-Vergleich), Zählindex und TACHO-String aus Datei,
487 solange bis Zählindex mit GPS-Zählindex übereinstimmt.
488 Dann wird Sensorwert aus TACHO-Sting in Array geschrieben.
489
490 Parameter: *fp_TACHO                  Datei, aus der das GPS_Paket gelesen wird.
491            j                          Array-Index.
492            *old_ts_tacho              vorheriger Timestamp.
493            *tacho_period              TS-Differenz.
494            *index_gps                 Zähl-Index von GPS Werten.
495            *tick                      Sensor-Werte.
496
497 Return: EXIT_SUCCESS, wenn erfolgreich.
498         EXIT_FAILURE, wenn Fehler in subfunc.
499 -----------------------------------------------------------------------------*/
500  int read_tacho(FILE *fp_TACHO, unsigned long int j, unsigned int *old_ts_tacho,
501                 unsigned int *tacho_period, unsigned long int *index_gps,
502                 double *tick)
503  {
504     unsigned long int index_tacho = 0;
505     char tacho_string[10 + 1 + 1] = "";  //lu + '\n' + '\0'.
506     char void_string[1 + 1] = "";
507     int error_strto = EXIT_SUCCESS;
508
509     do
510     {
511         if(compare_timestamp(fp_TACHO, old_ts_tacho, tacho_period, "TACHO")
512                              == EXIT_FAILURE)
513         {
514             save_error("->compare_timestamp()");
515             return EXIT_FAILURE;
516         }
517         if(read_count_index(fp_TACHO, &index_tacho) == EXIT_FAILURE)
518         {
519             save_error("->read_count_index()");
520             return EXIT_FAILURE;
521         }
522         if(read_file_till_char(fp_TACHO, '\n', tacho_string, 10 + 1 + 1)
523                                == EXIT_FAILURE)
524         {
525             save_error("->read_file_till_char(\n)");
```

```
526             return EXIT_FAILURE;
527         }
528         if(read_file_till_char(fp_TACHO, '$', void_string, 1 + 1)
529                                 == EXIT_FAILURE)
530         {
531             save_error("->read_file_till_char($)");
532             return EXIT_FAILURE;
533         }
534 //(void) printf("\ntacho_string: %s length: %i\n", tacho_string,
535                                                 //strlen(tacho_string));
536     } while(index_tacho != *index_gps);
537     //keine Kontrolle möglich.
538     tick[j] = my_str_to_double(tacho_string, &error_strto);
539     if(error_strto == EXIT_FAILURE)
540     {
541         save_error("->my_str_to_lu(tacho_string)");
542         return EXIT_FAILURE;
543     }
544 
545     return EXIT_SUCCESS;
546  }
547 
548 
549 /*-----------------------------------------------------------------------------
550 Wechselt ungefähr alle 250ms den Pegel von der Kommunikation zum Watchdog.
551 (Lebenszeichen)
552 
553 Parameter: *watchdog_out                    Bestimmt Output.
554                                             Wechselt von 1 auf 0 / 0 auf 1.
555            *loop_counter_watchdog_output    for-Schleife ist zu schnell,
556                                             deshalb nur alle 2000 Durchläufe
557                                             einen Pegelwechsel.
558 
559 Return: EXIT_SUCCESS, wenn erfolgreich.
560         EXIT_FAILURE, wenn watchdog_out unzulässiger Wert.
561 -----------------------------------------------------------------------------*/
562 int watchdog_output(int *watchdog_out, int *loop_counter_watchdog_output)
563 {
564     (*loop_counter_watchdog_output)++;
565     if(*loop_counter_watchdog_output >= 2000)
566     {
567         *loop_counter_watchdog_output = 0;
568         if(*watchdog_out == 1)
569         {
570             *watchdog_out = 0;
```

```
571         }
572         else if(*watchdog_out == 0)
573         {
574             *watchdog_out = 1;
575         }
576         else
577         {
578             save_error("watchdog_out unzulässiger Wert");
579             return EXIT_FAILURE;
580         }
581         #ifndef S_SPLINT_S
582         digitalWrite(WD_OUTPUT, *watchdog_out);
583         #endif
584     }
585 
586     return EXIT_SUCCESS;
587 }
588 
589 /*------------------------------------------------------------------------------
590 Diese Funktion wird aufgerufen, wenn ein Fehler beim Einlesen auftritt.
591 Es wird dafür gesorgt, dass der aktuelle Sensorwert auf -1 gesetzt wird.
592 
593 Parameter: *fp_FILE         Aus welcher Datei soll eingelesen werden?
594            *value 1 bis 4   Welche Werte sollen verworfen (-1) werden?
595 
596 Return:
597 
598 ------------------------------------------------------------------------------*/
599 void skip_value(double *value1, double *value2, double *value3, double *value4)
600 {
601     if(value1 != NULL)
602     {
603         *value1 = -1.0;
604     }
605     if(value2 != NULL)
606     {
607         *value2 = -1.0;
608     }
609     if(value3 != NULL)
610     {
611         *value3 = -1.0;
612     }
613     if(value4 != NULL)
614     {
615         *value4 = -1.0;
```

```
616     }
617
618 }
619
620 /*------------------------------------------------------------------------------
621 Erstellt mat-File aus allen Sensor-Werten und der Startzeit.
622
623 Parameter: j                        Array-Index.
624            *fdir_plus_mat_string    Name des mat-Files.
625            **sensor_arrays          Alle Sensor-Werte.
626            *timevector              String mit Startzeit.
627
628 Return: EXIT_SUCCESS, wenn erfolgreich.
629         EXIT_FAILURE, wenn Fehler in subfunc.
630 ------------------------------------------------------------------------------*/
631 #ifndef S_SPLINT_S
632 int create_mat_file(unsigned long int j, char *fdir_plus_mat_string,
633                     double **sensor_arrays, double *timevector)
634 {
635     mat_t *matfp;
636     matvar_t *matvar;
637     size_t dims[2] = {(size_t) j, 1};
638     size_t timedims[2] = {1, 6}; // [JJJJ, MM, TT, hh, mm, ss].
639     char *mat_name[] = {"Data1_Y_absolute", "Data1_X_absolute",
640                         "Data1_Used_satellites", "Data1_Hoehe",
641                         "Data1_Geschwindigkeit",
642                         "Data1_Direction", "Data1_Time",
643                         "Data1_Haltestelle___U", "Data1_Tachosignal"};
644     int i = 0;
645
646     (void) printf(".mat-Datei erstellen");
647     matfp = Mat_CreateVer(fdir_plus_mat_string, NULL, MAT_FT_MAT5);    //Öffnen.
648     if(NULL == matfp)
649     {
650         save_error("Fehler .mat-Datei");
651         save_error("->Mat_CreateVer()");
652         return EXIT_FAILURE;
653     }
654     for(i = 0; i < MAX_ARRAYS; i++)
655     {
656         matvar = Mat_VarCreate(mat_name[i], MAT_C_DOUBLE, MAT_T_DOUBLE, 2, dims,
657                                sensor_arrays[i], 0);      //Alle sensor_arrays.
658         if(NULL == matvar)
659         {
660             save_error("Fehler beim Erstellen der matvar ");
```

```
661             save_error(mat_name[i]);
662             (void) Mat_Close(matfp);
663             return EXIT_FAILURE;
664         }
665         (void) Mat_VarWrite(matfp,matvar,MAT_COMPRESSION_ZLIB);
666         Mat_VarFree(matvar);
667     }
668     matvar = Mat_VarCreate("Start_time", MAT_C_DOUBLE, MAT_T_DOUBLE, 2,
669                            timedims, timevector, 0);       //timevector
670     if(NULL == matvar)
671     {
672         save_error("Fehler beim Erstellen der matvar Start_time");
673         (void) Mat_Close(matfp);
674         return EXIT_FAILURE;
675     }
676     (void) Mat_VarWrite(matfp, matvar, MAT_COMPRESSION_ZLIB);
677     Mat_VarFree(matvar);
678     (void) Mat_Close(matfp);
679     (void) printf("\t\t\t\t\t\t[  "ok"  ]\n");
680 
681     return EXIT_SUCCESS;
682 }
683 #endif
684 /*-----------------------------------------------------------------------------
685 free.
686 
687 Parameter: **sensor_arrays  alle Arrays.
688 
689 Return: EXIT_SUCCESS, wenn erfolgreich.
690         EXIT_FAILURE, wenn free(NULL).
691 -----------------------------------------------------------------------------*/
692 int free_all_arrays(double **sensor_arrays)
693 {
694     int i = 0;
695 
696     for(i = 0; i < MAX_ARRAYS; i++)
697     {
698         if(sensor_arrays[i] != NULL)
699         {
700             free(sensor_arrays[i]);
701             sensor_arrays[i] = NULL;
702         }
703         else
704         {
705             save_error("free(NULL)");
```

```
706             return EXIT_FAILURE;
707         }
708     }
709 
710     return EXIT_SUCCESS;
711 }
712 
713  /*-------------------------------------------------------------------------
714 Schließt alle Files.
715 
716 Parameter: *fp  alle Files.
717 
718 Return: EXIT_SUCCESS, wenn erfolgreich.
719         EXIT_FAILURE, wenn fclose() fehlschlägt oder fclose(NULL).
720 -------------------------------------------------------------------------*/
721 int close_all_files(FILE **all_files)
722 {
723     int i = 0;
724 
725     for(i = 0; i < MAX_FILES; i++)
726     {
727         if(all_files[i] != NULL)
728         {
729             if(fclose(all_files[i]) == EOF)
730             {
731                 save_error(strerror(errno));
732                 save_error("->fclose()");
733                 return EXIT_FAILURE;
734             }
735         }
736         else
737         {
738             save_error("fclose(NULL)");
739             return EXIT_FAILURE;
740         }
741     }
742 
743     return EXIT_SUCCESS;
744 }
745 
746 /*-------------------------------------------------------------------------
747 Führt alle Dateien zu einem Zip-Archiv zusammen und speichert es im Datenbuffer.
748 Beispiel:
749 zip -r -j /home/pi/databuffer/20160301_193320_yocto.zip /media/INTENSO/Data008/
750 Alle Rohdaten aus Data008 werden in gezipter Form im databuffer abgelegt.
```

```
751 Rohdaten bleiben auf usb-stick.
752 (Erstellt von Mario Wegner)
753
754 Parameter: *fdir    Verzeichniss in dem die zu archivierenden Dateien sind.
755
756 Return: EXIT_SUCCESS, wenn erfolgreich.
757         EXIT_FAILURE, wenn Fehler in time(), strftime() oder system().
758 ------------------------------------------------------------------------------*/
759 int zip_files(char *fdir)
760 {
761     time_t rawtime = {0};
762     struct tm *info;
763     char buffer[80] = "";
764     char zipstring[500] = "zip -j -u ";
765
766     (void) printf("\nZip-Archiv erstellen:\n");
767     if(time(&rawtime) == -1)    // getting the starttime.
768     {
769         save_error(strerror(errno));
770         save_error("->time()");
771         return EXIT_FAILURE;
772     }
773     info = localtime(&rawtime);
774     if(strftime(buffer, 80, "%Y%m%d_%H%M%S_yocto", info) == 0)
775     {
776         save_error(strerror(errno));
777         save_error("->strftime()");
778         return EXIT_FAILURE;
779     }
780     strcat(zipstring, "/home/pi/databuffer/");
781     strcat(zipstring, buffer);
782     strcat(zipstring, ".zip ");
783     strcat(zipstring, fdir);
784
785     if(zip_file(zipstring, "ACC.txt") == EXIT_FAILURE)
786     {
787         save_error("->zip_file");
788         return EXIT_FAILURE;
789     }
790     if(zip_file(zipstring, "MAG.txt") == EXIT_FAILURE)
791     {
792         save_error("->zip_file");
793         return EXIT_FAILURE;
794     }
795     if(zip_file(zipstring, "GYR.txt") == EXIT_FAILURE)
```

```
796     {
797         save_error("->zip_file");
798         return EXIT_FAILURE;
799     }
800     if(zip_file(zipstring, "PR_TE.txt") == EXIT_FAILURE)
801     {
802         save_error("->zip_file");
803         return EXIT_FAILURE;
804     }
805     if(zip_file(zipstring, "TACHO.txt") == EXIT_FAILURE)
806     {
807         save_error("->zip_file");
808         return EXIT_FAILURE;
809     }
810     if(zip_file(zipstring, "DOOR.txt") == EXIT_FAILURE)
811     {
812         save_error("->zip_file");
813         return EXIT_FAILURE;
814     }
815     if(zip_file(zipstring, "GPS_GGA.nmea") == EXIT_FAILURE)
816     {
817         save_error("->zip_file");
818         return EXIT_FAILURE;
819     }
820     if(zip_file(zipstring, "WATCHDOG.txt") == EXIT_FAILURE)
821     {
822         save_error("->zip_file");
823         return EXIT_FAILURE;
824     }
825     if(zip_file(zipstring, "*") == EXIT_FAILURE)
826     {
827         save_error("->zip_file");
828         return EXIT_FAILURE;
829     }
830 
831     (void) printf("Zip-Archiv erstellen abgeschlossen.\n");
832 
833     return EXIT_SUCCESS;
834 }
```

B.2.2. DataCalc.h

Die aus der Quelle [1] übernommenen Programmabschnitte wurden gekennzeichnet.

```
1 #ifndef DATACALC_H_INCLUDED
```

```
2  #define DATACALC_H_INCLUDED
3
4  #define ok   "\033[32mOK\033[m"
5  #define err  "\033[31mERR\033[m"
6  #define warn "\033[33mWARN\033[m"
7
8  #define lognum "11"
9  #define busnum "7301"
10
11 #define WD_OUTPUT 23
12
13 #define NANO 1000000000L
14
15 #define DOOR_PERIOD (1.0/DOOR_FREQ) * 1000.0
16 #define DOOR_GRENZE (1.0/DOOR_FREQ) * TS_UNGENAUIGKEIT * 10.0
17 #define TACHO_PERIOD (1.0/TACHO_FREQ) * 1000.0
18 #define TACHO_GRENZE (1.0/TACHO_FREQ) * TS_UNGENAUIGKEIT * 10.0
19 #define GPS_PERIOD (1.0/GPS_FREQ) * 1000.0
20 #define GPS_GRENZE (1.0/GPS_FREQ) * TS_UNGENAUIGKEIT * 10.0
21
22 //Deaktiviert = auskommentieren.
23 //#############################################################################
24 //#define PRINT_TO_SCREEN
25 #define MAX_FILES 5
26 #define MAX_ARRAYS 9
27
28 #define DOOR_FREQ 30       //in Hz
29 #define TACHO_FREQ 30
30 #define GPS_FREQ 10
31
32 #define TS_UNGENAUIGKEIT 10 //in %. Ab wann sollen die Werte als falsch gelten?
33 //#############################################################################
34
35 int initializes_gpio(void);
36 int getdir(char *fdir);
37 /*@null@*/FILE *openfile(char *fdir, char  *filename, char *mode);
38 int open_all_files(char *fdir, FILE **all_files,
39                    char **file_names, char **file_mode);
40 int read_points(FILE *fp_POINTS, unsigned long int *points_gps);
41 int allocate_all_arrays(unsigned long int points_gps,
42                         double **sensor_arrays);
43 int save_time(FILE *fp_TIME, char *mat_string, double *timevector);
44 int read_gga(FILE *fp_GGA, unsigned long int j, unsigned int *old_ts_gga,
45              unsigned int *gga_period, unsigned long int *index_gps,
46              double *Venus_Y, double *Venus_X, double *Venus_Used_satellites,
```

```
47              double *Venus_Hoehe);
48 int read_rmc(FILE *fp_GGA, unsigned long int j, unsigned int *old_ts_rmc,
49              unsigned int *rmc_period, unsigned long int *index_gps,
50              double *Venus_Geschwindigkeit, double *Venus_Direction);
51 int read_time(unsigned long int j, unsigned int *period, double *Venus_Time);
52 int read_door(FILE *fp_DOOR, unsigned long int j, unsigned int *old_ts_door,
53               unsigned int *door_period, unsigned long int *index_gps,
54               double *Haltestelle);
55 int read_tacho(FILE *fp_TACHO, unsigned long int j, unsigned int *old_ts_tacho,
56                unsigned int *tacho_period, unsigned long int *index_gps,
57                double *tick);
58 int watchdog_output(int *watchdog_out, int *loop_counter_watchdog_output);
59 void skip_value(/*@null@*/double *value1, /*@null@*/double *value2,
60                 /*@null@*/double *value3, /*@null@*/double *value4);
61 #ifndef S_SPLINT_S
62 int create_mat_file(unsigned long int j, char *fdir_plus_mat_string,
63                     double **sensor_arrays, double *timevector);
64 #endif
65 int free_all_arrays(double **sensor_arrays);
66 int close_all_files(FILE **all_files);
67 int zip_files(char *fdir);
68
69 #endif // DATACALC_H_INCLUDED
```

B.2.3. LowFuncCalc.c

Die aus der Quelle [1] übernommenen Programmabschnitte wurden gekennzeichnet.

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4 #include <math.h>
5 #include <errno.h>
6 #include <time.h>
7
8 #include "LowFuncCalc.h"
9 #include "ErrorStorage.h"
10 #include "global.h"
11 #include "DataCalc.h"
12 #ifndef S_SPLINT_S
13 #include "wiringPi.h"
14 #endif
15
16 static struct date_time read_timestamp(FILE *fp_FILE);
17 static int my_str_to_int(char *my_string_to_convert, int *error);
```

```
18 static unsigned long int my_str_to_lu(char *my_string_to_convert, int *error);
19
20 /*---------------------------------------------------------------------------
21 Hängt den zweiten String an den ersten.  (Erstellt von Mario Wegner)
22 Kein save_error(). ERROR_STORAGE.txt noch nicht erstellt.
23
24 Parameter: s1   Erster String.
25            s2   Zweiter String.
26
27 Return: restult, erster plus zweiter String.
28         NULL   , wenn malloc() fehlschlägt.
29 ---------------------------------------------------------------------------*/
30 char *concat(char *s1, char *s2)
31 {
32     char *result = malloc(strlen(s1) + strlen(s2) + 1); //+1 for the '\0'.
33
34     if(result == NULL)
35     {
36         (void) printf("%s", strerror(errno));
37         (void) printf("->malloc()");
38         return NULL;
39     }
40     strcpy(result, s1);
41     strcat(result, s2);
42
43     return result;
44 }
45
46 /*---------------------------------------------------------------------------
47 Lie ß t und vergleicht (alten/neuen) Timestamps. Falls im Vergleich
48 eine Ungenauigkeit von >8% festgestellt wird, wird globale Zählvaribale
49 inkrementiert.
50 globale Zählvariablen:
51 door_wrong;
52 tacho_wrong;
53 gps_wrong;
54
55 Parameter: *fp_FILE     Datei, aus der der TS gelesen wird.
56            *old_ts      vorheriger (alter) TS.
57            *period      Diff zwischen neuen und alten TS.
58            *sensor_name Bestimmt Ungenauigkeits-Fenster.
59
60 Return: EXIT_SUCCESS, wenn erfolgreich.
61         EXIT_FAILURE, wenn read_timestamp() fehlschlägt.
62 ---------------------------------------------------------------------------*/
```

```
63 int compare_timestamp(FILE *fp_FILE, unsigned int *old_ts, unsigned int *period,
64                       char *sensor_name)
65 {
66     struct date_time ts;
67 
68     ts = read_timestamp(fp_FILE);
69     if(ts.error == EXIT_FAILURE)
70     {
71         save_error("->read_timestamp()");
72         return EXIT_FAILURE;
73     }
74     if(*old_ts < ts.msec)
75     {
76         *period = ts.msec - *old_ts;
77     }
78     else if(*old_ts != 1000)
79     {
80         *period = 1000 + ts.msec - *old_ts;
81     }
82     else
83     {
84         #ifdef PRINT_TO_SCREEN
85         (void) printf("Erster Wert. Vergleich noch nicht möglich.\n");
86         #endif
87         *old_ts = ts.msec;
88         return EXIT_SUCCESS;
89     }
90 
91     #ifdef PRINT_TO_SCREEN
92     (void) printf("%s: Time between signals: %u ms\n", sensor_name, *period);
93     #endif
94     if((strcmp(sensor_name, "DOOR") == 0) &&
95        ((*period > (unsigned int)(DOOR_PERIOD + DOOR_GRENZE)) ||
96         (*period < (unsigned int)(DOOR_PERIOD - DOOR_GRENZE))))
97     {
98         door_wrong++;
99     }
100    else if((strcmp(sensor_name,"TACHO") == 0) &&
101            ((*period > (unsigned int)(TACHO_PERIOD + TACHO_GRENZE)) ||
102             (*period < (unsigned int)(TACHO_PERIOD - TACHO_GRENZE))))
103    {
104        tacho_wrong++;
105    }
106    else if((strcmp(sensor_name, "GGA") == 0) &&
107            ((*period > (unsigned int)(GPS_PERIOD + GPS_GRENZE)) ||
```

```
108                 (*period < (unsigned int)(GPS_PERIOD - GPS_GRENZE))))
109     {
110         gga_wrong++;
111     }
112     else if((strcmp(sensor_name, "RMC") == 0) &&
113             ((*period > (unsigned int)(GPS_PERIOD + GPS_GRENZE)) ||
114              (*period < (unsigned int)(GPS_PERIOD - GPS_GRENZE))))
115     {
116         rmc_wrong++;
117     }
118     else
119     {
120
121     }
122     *old_ts = ts.msec;
123
124     return EXIT_SUCCESS;
125 }
126
127 /*-----------------------------------------------------------------------------
128 Lie ß t Timestamp (Format: 2016.04.14.23.59.28.903:)
129 aus Datei und erstellt daraus struct.
130
131 Parameter: *fp_FILE     Datei, aus der der Timestamp gelesen werden soll.
132
133 Return: EXIT_SUCCESS, wenn erfolgreich.
134         EXIT_FAILURE, wenn der eingelesene TS falsches Format hat
135                       oder my_str_to_int() fehlschlägt.
136 -----------------------------------------------------------------------------*/
137 static struct date_time read_timestamp(FILE *fp_FILE)
138 {
139     struct date_time timestamp = {0};
140     char timestamp_string[5 + 1] = "";
141     int error_strto = EXIT_SUCCESS;
142
143     timestamp.error = EXIT_SUCCESS;
144
145     if(read_file_till_char(fp_FILE, '.', timestamp_string, 5 + 1)
146                            == EXIT_FAILURE)
147     {
148         save_error("->read_file_till_char(year)");
149         timestamp.error = EXIT_FAILURE;
150         return timestamp;
151     }
152     timestamp.year = (unsigned int)my_str_to_int(timestamp_string, &error_strto);
```

```
153     if(error_strto == EXIT_FAILURE)
154     {
155         save_error("->my_str_to_int(year)");
156         timestamp.error = EXIT_FAILURE;
157         return timestamp;
158     }
159 
160     if(read_file_till_char(fp_FILE, '.', timestamp_string, 3 + 1)
161                             == EXIT_FAILURE)
162     {
163         save_error("->read_file_till_char(month)");
164         timestamp.error = EXIT_FAILURE;
165         return timestamp;
166     }
167     timestamp.month = (unsigned int)my_str_to_int(timestamp_string, &error_strto);
168     if(error_strto == EXIT_FAILURE)
169     {
170         save_error("->my_str_to_int(month)");
171         timestamp.error = EXIT_FAILURE;
172         return timestamp;
173     }
174 
175     if(read_file_till_char(fp_FILE, '.', timestamp_string, 3 + 1)
176                             == EXIT_FAILURE)
177     {
178         save_error("->read_file_till_char(day)");
179         timestamp.error = EXIT_FAILURE;
180         return timestamp;
181     }
182     timestamp.day = (unsigned int)my_str_to_int(timestamp_string, &error_strto);
183     if(error_strto == EXIT_FAILURE)
184     {
185         save_error("->my_str_to_int(day)");
186         timestamp.error = EXIT_FAILURE;
187         return timestamp;
188     }
189 
190     if(read_file_till_char(fp_FILE, '.', timestamp_string, 3 + 1)
191                             == EXIT_FAILURE)
192     {
193         save_error("->read_file_till_char(hour)");
194         timestamp.error = EXIT_FAILURE;
195         return timestamp;
196     }
197     timestamp.hour = (unsigned int)my_str_to_int(timestamp_string, &error_strto);
```

```
198     if(error_strto == EXIT_FAILURE)
199     {
200         save_error("->my_str_to_int(hour)");
201         timestamp.error = EXIT_FAILURE;
202         return timestamp;
203     }
204 
205     if(read_file_till_char(fp_FILE, '.', timestamp_string, 3 + 1)
206                             == EXIT_FAILURE)
207     {
208         save_error("->read_file_till_char(minute)");
209         timestamp.error = EXIT_FAILURE;
210         return timestamp;
211     }
212     timestamp.minute = (unsigned int)my_str_to_int(timestamp_string, &error_strto);
213     if(error_strto == EXIT_FAILURE)
214     {
215         save_error("->my_str_to_int(minute)");
216         timestamp.error = EXIT_FAILURE;
217         return timestamp;
218     }
219 
220     if(read_file_till_char(fp_FILE, '.', timestamp_string, 3 + 1)
221                             == EXIT_FAILURE)
222     {
223         save_error("->read_file_till_char(sec)");
224         timestamp.error = EXIT_FAILURE;
225         return timestamp;
226     }
227     timestamp.sec = (unsigned int)my_str_to_int(timestamp_string, &error_strto);
228     if(error_strto == EXIT_FAILURE)
229     {
230         save_error("->my_str_to_int(sec)");
231         timestamp.error = EXIT_FAILURE;
232         return timestamp;
233     }
234 
235     if(read_file_till_char(fp_FILE, ':', timestamp_string, 4 + 1)
236                             == EXIT_FAILURE)
237     {
238         save_error("->read_file_till_char(msec)");
239         timestamp.error = EXIT_FAILURE;
240         return timestamp;
241     }
242     timestamp.msec = (unsigned int)my_str_to_int(timestamp_string, &error_strto);
```

```
243     if(error_strto == EXIT_FAILURE)
244     {
245         save_error("->my_str_to_int(msec)");
246         timestamp.error = EXIT_FAILURE;
247         return timestamp;
248     }
249 
250     return timestamp;
251 }
252 
253 /*---------------------------------------------------------------------------
254 Ließt Zähl-Index aus Datei und gibt ihn als Parameter wieder.
255 
256 Parameter: *fp_FILE          Datei, aus der der Index gelesen werden soll.
257            *index_sensor     String, in dem der gelesene Index gespeichert wird.
258 
259 Return: EXIT_SUCCESS, wenn erfolgreich.
260         EXIT_FAILURE, wenn der eingelesene Index falsches Format hat
261                       oder my_str_to_lu() fehlschlägt.
262 ---------------------------------------------------------------------------*/
263 int read_count_index(FILE *fp_FILE, unsigned long int *index_sensor)
264 {
265     char index_string[10 + 1] = "";
266     int error_strto = EXIT_SUCCESS;
267 
268     if(read_file_till_char(fp_FILE, '_', index_string, 10 + 1) == EXIT_FAILURE)
269     {
270         save_error("->read_file_till_char()");
271         return EXIT_FAILURE;
272     }
273     *index_sensor = my_str_to_lu(index_string, &error_strto);
274     if(error_strto == EXIT_FAILURE)
275     {
276         save_error("->my_str_to_lu(index_string)");
277         return EXIT_FAILURE;
278     }
279     //(void) printf("\nsensor_index: %i \n", *index_sensor);
280 
281     return EXIT_SUCCESS;
282 }
283 
284 /*---------------------------------------------------------------------------
285 Durchläuft GGA-String und filtert gewünschte Sensorwerte raus.
286 Orientierung anhand von ','. Ende des GGA-Strings wird durch '$' erkannt.
287 
```

```
288 Parameter: *fp_GGA                  Datei, aus der das GPS_Paket gelesen wird.
289            *Venus_Y                 gewünschte Sensorwerte.
290            *Venus_X                 x
291            *Venus_Used_satellites   x
292            *Venus_Hoehe             x
293
294 Return: EXIT_SUCCESS, wenn erfolgreich.
295         EXIT_FAILURE, wenn GGA-String nicht die erwartete Anzahl an ',' besitzt.
296                       fp wird durch fseek auf ursprüngliche Position gesetzt.
297                     , wenn der String nicht mit '$' abschlie ß t
298                       oder wenn my_str_to_int() fehlschlägt.
299 ------------------------------------------------------------------------------*/
300 int read_gga_info(FILE *fp_GGA, double *Venus_Y, double *Venus_X,
301                   double *Venus_Used_satellites, double *Venus_Hoehe)
302 {
303     char gga_string[11 + 1] = "";
304     size_t gga_string_length = 0;
305     char void_string[10 + 1] = "";
306     int i = 0;
307     int j = 0;
308     int error_strto = EXIT_SUCCESS;
309
310     for(i = 1; i <= 14; i++)
311     {
312         if(read_file_till_char(fp_GGA, ',', gga_string, 11 + 1) == EXIT_FAILURE)
313         {
314             save_error("->read_file_till_char(,)");
315             if(fseek(fp_GGA, -((long int)(gga_string_length + i)), SEEK_CUR) < 0)
316             {
317                 save_error(strerror(errno));
318                 save_error("->fseek()");
319             }
320             return EXIT_FAILURE;
321         }
322         gga_string_length = gga_string_length + strlen(gga_string);
323         switch(i)
324         {
325             case 3:
326                 for(j = 0; j < 9; j++)
327                 {
328                     if(gga_string[j] > 57)
329                     {
330                         save_error("->Buchstabe in Venus_Y");
331                         if(fseek(fp_GGA, -((long int)(gga_string_length + i)), SEEK_CUR) < 0)
332                         {
```

```
333                             save_error(strerror(errno));
334                             save_error("->fseek()");
335                         }
336                         return EXIT_FAILURE;
337                     }
338                 }
339                             // (Erstellt von Mario Wegner)
340                 *Venus_Y =   ((gga_string[0]-48)*10)+((gga_string[1]-48))+
341                             (((gga_string[2]-48)*10)+(gga_string[3]-48)+
342                              ((gga_string[5]-48)*0.1)+((gga_string[6]-48)*0.01)+
343                      ((gga_string[7]-48)*0.001)+((gga_string[8]-48)*0.0001))/60;
344 
345                 break;
346             case 5:
347                 for(j = 1; j < 10; j++)
348                 {
349                     if(gga_string[j] > 57)
350                     {
351                         save_error("->Buchstabe in Venus_X");
352                         if(fseek(fp_GGA, -((long int)(gga_string_length + i)), SEEK_CUR) < 0)
353                         {
354                             save_error(strerror(errno));
355                             save_error("->fseek()");
356                         }
357                         return EXIT_FAILURE;
358                     }
359                 }
360                                 // (Erstellt von Mario Wegner)
361                 *Venus_X =   ((gga_string[1]-48)*10)+(gga_string[2]-48)+
362                             (((gga_string[3]-48)*10)+(gga_string[4]-48)+
363                              ((gga_string[6]-48)*0.1)+((gga_string[7]-48)*0.01)+
364                      ((gga_string[8]-48)*0.001)+((gga_string[9]-48)*0.0001))/60;
365                 break;
366             case 8:
367                *Venus_Used_satellites = my_str_to_double(gga_string, &error_strto);
368                 if(error_strto == EXIT_FAILURE)
369                 {
370                     save_error("->my_str_to_int(gga_string)");
371                     if(fseek(fp_GGA, -((long int)(gga_string_length + i)), SEEK_CUR) < 0)
372                     {
373                         save_error(strerror(errno));
374                         save_error("->fseek()");
375                     }
376                     return EXIT_FAILURE;
377                 }
```

```
378                 break;
379             case 10:
380                 *Venus_Hoehe = my_str_to_double(gga_string, &error_strto);
381                 if(error_strto == EXIT_FAILURE)
382                 {
383                     save_error("->my_str_to_double(gga_string)");
384                     if(fseek(fp_GGA, -((long int)(gga_string_length + i)), SEEK_CUR) < 0)
385                     {
386                         save_error(strerror(errno));
387                         save_error("->fseek()");
388                     }
389                     return EXIT_FAILURE;
390                 }
391                 break;
392         }
393     }
394     if(read_file_till_char(fp_GGA, '$', void_string, 10 + 1) == EXIT_FAILURE)
395     {
396         save_error("->read_file_till_char($)");
397         if(fseek(fp_GGA, -((long int)(gga_string_length + i)), SEEK_CUR) < 0)
398         {
399             save_error(strerror(errno));
400             save_error("->fseek()");
401         }
402         return EXIT_FAILURE;
403 
404     }
405 
406     return EXIT_SUCCESS;
407 }
408 
409 /*-------------------------------------------------------------------------
410 Durchläuft RMC-String und filtert gewünschte Sensorwerte raus.
411 Orientierung anhand von ','.
412 Ende des RMC-Strings wird durch '$' erkannt.
413 
414 Parameter: *fp_GGA                  Datei, aus der das GPS_Paket gelesen wird.
415            *Venus_Direction         gewünschte Sensorwerte.
416            *Venus_Geschwindigkeit   x
417 
418 Return: EXIT_SUCCESS, wenn erfolgreich.
419         EXIT_FAILURE, wenn RMC-String nicht die erwartete Anzahl an ',' besitzt.
420                       fp wird durch fseek auf ursprüngliche Position gesetzt.
421                     , wenn der String nicht mit '$' abschlie ß t
422                       oder wenn my_str_to_double() fehlschlägt.
```

```
423 ------------------------------------------------------------------------------*/
424 int read_rmc_info(FILE *fp_GGA, double *Venus_Geschwindigkeit,
425                   double *Venus_Direction)
426 {
427     char rmc_string[11 + 1] = "";
428     size_t rmc_string_length = 0;
429     char void_string[7 + 1] = "";
430     int i = 0;
431     int error_strto = EXIT_SUCCESS;
432 
433     for(i = 1; i <= 12; i++)
434     {
435         if(read_file_till_char(fp_GGA, ',', rmc_string, 11 + 1) == EXIT_FAILURE)
436         {
437             save_error("->read_file_till_char(,)");
438             if(fseek(fp_GGA, -((long int)(rmc_string_length + i)), SEEK_CUR) < 0)
439             {
440                 save_error(strerror(errno));
441                 save_error("->fseek()");
442             }
443             return EXIT_FAILURE;
444         }
445         rmc_string_length = rmc_string_length + strlen(rmc_string);
446         switch(i)
447         {
448             case 8:
449                 *Venus_Geschwindigkeit = my_str_to_double(rmc_string, &error_strto)
450                                          * 1.852;        //knoten -> kmh
451                 if(error_strto == EXIT_FAILURE)
452                 {
453                     save_error("->my_str_to_double(rmc_string)");
454                     if(fseek(fp_GGA, -((long int)(rmc_string_length + i)), SEEK_CUR) < 0)
455                     {
456                         save_error(strerror(errno));
457                         save_error("->fseek()");
458                     }
459                     return EXIT_FAILURE;
460                 }
461                 break;
462             case 9:
463                 *Venus_Direction = my_str_to_double(rmc_string, &error_strto);
464                 if(error_strto == EXIT_FAILURE)
465                 {
466                     save_error("->my_str_to_double(gga_string)");
467                     if(fseek(fp_GGA, -((long int)(rmc_string_length + i)), SEEK_CUR) < 0)
```

```
468                     {
469                         save_error(strerror(errno));
470                         save_error("->fseek()");
471                     }
472                     return EXIT_FAILURE;
473                 }
474                 break;
475         }
476     }
477 
478     if(read_file_till_char(fp_GGA, '$', void_string, 7 + 1) == EXIT_FAILURE)
479     {
480         save_error("->read_file_till_char($)");
481         if(fseek(fp_GGA, -((long int)(rmc_string_length + i)), SEEK_CUR) < 0)
482         {
483             save_error(strerror(errno));
484             save_error("->fseek()");
485         }
486         return EXIT_FAILURE;
487     }
488 
489     return EXIT_SUCCESS;
490 }
491 
492 /*-----------------------------------------------------------------------------
493 Lie ß t solange vom aus Datei, bis gesuchter char eingelesen wird.
494 
495 Parameter: *fp_FILE         Aus welcher Datei soll eingelesen werden?
496            search_char      Bis zu welchem char soll eingelesen werden?
497            *found_string    Alle Zeichen, die bis zum char eingelesen wurden.
498                             Ohne search_char.
499            length           Wie viele Zeichen sollen maximal eingelesen werden?
500 
501 Return: EXIT_SUCCESS, wenn erfolgreich.
502         EXIT_FAILURE, wenn search_char nicht gefunden wurde.
503                       File_Pionter wird mit fseek() zurückgesetzt,
504                       damit '$' gefunden werden kann und somit der
505                       nächste Wert eingelesen werden kann.
506 -----------------------------------------------------------------------------*/
507 int read_file_till_char(FILE *fp_FILE, char search_char, char *found_string,
508                         int length)
509 {
510     int c;
511     int i = 0;
512 
```

```
513     found_string[0] = '\0';
514     do
515     {
516         if(i >= length - 1)
517         {
518             if(fseek(fp_FILE, -((long int)(length - 2)), SEEK_CUR) < 0)
519             {
520                 save_error(strerror(errno));
521                 save_error("->fseek()");
522             }
523             save_error(found_string);
524             save_error(" :Char nicht gefunden");
525             return EXIT_FAILURE;
526         }
527         c = fgetc(fp_FILE);
528         found_string[i] = c;
529         i++;
530         found_string[i] = '\0';     //sauberes Einlesen
531 
532     } while(c != search_char);
533     found_string[i - 1] = '\0';     //löscht search_char aus string
534 
535     return EXIT_SUCCESS;
536 }
537 
538 /*------------------------------------------------------------------------------
539 Berechnet Zeit-Differenz in double msec von begin zu end.
540 Ermittelt durch clock_gettime().
541 
542 Parameter: begin    Startzeit aufgenommen mit clock_gettime().
543            end      Stopzeit aufgenommen mit clock_gettime().
544 
545 Return: duration, wenn erfolgreich.
546         EXIT_FAILURE, wenn Falsche Zeit-Argumente.
547 ------------------------------------------------------------------------------*/
548 double calc_time(struct timespec begin, struct timespec end)
549 {
550     struct timespec temp = {0};
551     double duration;
552 
553     if((end.tv_nsec - begin.tv_nsec) > 0)
554     {
555         temp.tv_sec = (end.tv_sec - begin.tv_sec);
556         temp.tv_nsec = (end.tv_nsec - begin.tv_nsec);
557     }
```

```
558     else if((end.tv_nsec - begin.tv_nsec) < 0)
559     {
560         temp.tv_sec = (end.tv_sec - begin.tv_sec) - 1;
561         temp.tv_nsec =  NANO + (end.tv_nsec - begin.tv_nsec);
562     }
563     else
564     {
565         save_error("Falsche Zeit-Argumente");
566         return -1.0;
567     }
568     duration = (double) temp.tv_sec + ((double) temp.tv_nsec / (double) NANO);
569     duration = duration * 1000.0;
570    // (void) printf("\nTime for Execution: %f ms\n", duration);
571 
572     return duration;
573 }
574 
575 /*---------------------------------------------------------------------------
576 Wartet.
577 
578 Parameter: time_milli   Wie viele Millisekunden soll gewartet werden?
579                         (In double!)
580 
581 Return: void
582 ---------------------------------------------------------------------------*/
583 void milli_sleep(double time_milli)
584 {
585     struct timespec temp = {0};
586 
587     if(time_milli < 1000.0)
588     {
589         temp.tv_sec = 0;
590         temp.tv_nsec = (long int)(time_milli * 1000000.0);
591     }
592     else
593     {
594         temp.tv_sec = (time_t)(time_milli / 1000.0);
595         temp.tv_nsec = (long int)((time_milli - ((double) temp.tv_sec * 1000.0))
596                         * 1000000.0);
597     }
598     nanosleep(&temp, NULL);
599 }
600 
601 /*---------------------------------------------------------------------------
602 zip.
```

```
603
604 Parameter: *zipstring   Ordner, in dem sich die Datei befindet.
605            *file_to_zip Datei, die gezipt werden soll.
606
607 Return: void
608 ------------------------------------------------------------------------------*/
609 int zip_file(char *zipstring, char *file_to_zip)
610 {
611     char zipstring_plus_file[500] = "";
612
613     #ifndef S_SPLINT_S
614     digitalWrite(WD_OUTPUT, LOW);
615     milli_sleep(250.0);
616     digitalWrite(WD_OUTPUT, HIGH);
617     milli_sleep(250.0);
618     digitalWrite(WD_OUTPUT, LOW);
619     #endif
620     strcpy(zipstring_plus_file, zipstring);
621     strcat(zipstring_plus_file, file_to_zip);
622     (void) printf("%s\n", zipstring_plus_file);
623     if(system(zipstring_plus_file) != 0)
624     {
625         save_error(strerror(errno));
626         save_error("->system()");
627         return EXIT_FAILURE;
628     }
629
630     return EXIT_SUCCESS;
631 }
632 /*------------------------------------------------------------------------------
633 String to int, double, unsigned long int.
634
635 Parameter: *string  String, der in Zahl umgewandelt werden soll.
636            *error   Fehler-Variable.
637
638 Return: i, wenn erfolgreich
639         Exit_FAILURE, wenn strtoX() fehlschlägt.
640 ------------------------------------------------------------------------------*/
641 static int my_str_to_int(char *my_string_to_convert, int *error)
642 {
643     char *endptr = NULL;
644     long int i;
645
646     errno = 0;
647     i = strtol(my_string_to_convert, &endptr, 10);
```

```
648     if(errno > 0)
649     {
650         save_error(strerror(errno));
651         save_error("->strtol()");
652         *error = EXIT_FAILURE;
653         return EXIT_FAILURE;
654     }
655     if(endptr == NULL)
656     {
657         *error = EXIT_FAILURE;
658         return EXIT_FAILURE;
659     }
660     if(my_string_to_convert == endptr || *endptr != '\0')
661     {
662         save_error("String beinhaltet Zeichen");
663         *error = EXIT_FAILURE;
664         return EXIT_FAILURE;
665     }
666 
667     return (int) i;
668 }
669 
670 double my_str_to_double(char *my_string_to_convert, int *error)
671 {
672     char *endptr = NULL;
673     double i;
674 
675     errno = 0;
676     i = strtod(my_string_to_convert, &endptr);
677     if(errno > 0)
678     {
679         save_error(strerror(errno));
680         save_error("->strtol()");
681         *error = EXIT_FAILURE;
682         return 1.0;     //EXIT_FAILURE als double
683     }
684     if(endptr == NULL)
685     {
686         *error = EXIT_FAILURE;
687         return 1.0;
688     }
689     if(my_string_to_convert == endptr || *endptr != '\0')
690     {
691         save_error("String beinhaltet Zeichen");
692         *error = EXIT_FAILURE;
```

```
693         return 1.0;
694     }
695
696     return i;
697 }
698
699 static unsigned long int my_str_to_lu(char *my_string_to_convert, int *error)
700 {
701     char *endptr = NULL;
702     unsigned long int i;
703
704     errno = 0;
705     i = strtoul(my_string_to_convert, &endptr, 10);
706     if(errno > 0)
707     {
708         save_error(strerror(errno));
709         save_error("->strtol()");
710         *error = EXIT_FAILURE;
711         return 1;
712     }
713     if(endptr == NULL)
714     {
715         *error = EXIT_FAILURE;
716         return 1;
717     }
718     if(my_string_to_convert == endptr || *endptr != '\0')
719     {
720         save_error("String beinhaltet Zeichen");
721         *error = EXIT_FAILURE;
722         return 1;
723     }
724
725     return i;
726 }
```

B.2.4. LowFuncCalc.h

Die aus der Quelle [1] übernommenen Programmabschnitte wurden gekennzeichnet.

```
1 #ifndef LOWFUNCCALC_H_INCLUDED
2 #define LOWFUNCCALC_H_INCLUDED
3
4
5 struct date_time
6 {
```

```
7       unsigned int year;
8       unsigned int month;
9       unsigned int day;
10      unsigned int hour;
11      unsigned int minute;
12      unsigned int sec;
13      unsigned int msec;
14      int error;
15  };
16
17  /*@null@*/char *concat(char *s1, char *s2);
18
19  int compare_timestamp(FILE *fp_FILE, unsigned int *old_ts, unsigned int *period,
20                        char *sensor_name);
21  int read_count_index(FILE *fp_FILE, unsigned long int *index_sensor);
22  int read_gga_info(FILE *fp_GGA, double *Venus_Y, double *Venus_X,
23                    double *Venus_Used_satellites, double *Venus_Hoehe);
24  int read_rmc_info(FILE *fp_GGA, double *Venus_Geschwindigkeit,
25                    double *Venus_Direction);
26  int read_file_till_char(FILE *fp_FILE, char search_char, char *found_string,
27                          int length);
28
29  double calc_time(struct timespec begin, struct timespec end);
30  inline void milli_sleep(double time_milli);
31  int zip_file(char *zipstring, char *file_to_zip);
32
33  double my_str_to_double(char *my_string_to_convert, int *error);
34
35  #endif // LOWFUNCCALC_H_INCLUDED
```

B.2.5. ErrorStorage.c

```
1   #include <stdio.h>
2
3   #include "ErrorStorage.h"
4
5   void save_error(char *error_string)
6   {
7       if(fp_ERROR != NULL)
8       {
9           (void) fputs(error_string, fp_ERROR);
10      }
11  }
```

B.2.6. ErrorStorage.h

```
#ifndef ERRORSTORAGE_H_INCLUDED
#define ERRORSTORAGE_H_INCLUDED

/*@null@*/extern FILE *fp_ERROR;
void save_error(char *error_string);

#endif // ERRORSTORAGE_H_INCLUDED
```

B.2.7. global.h

```
#ifndef GLOBAL_H_INCLUDED
#define GLOBAL_H_INCLUDED

extern int door_wrong;
extern int tacho_wrong;
extern int gga_wrong;
extern int rmc_wrong;

#endif // GLOBAL_H_INCLUDED
```

B.3. datatransfer.sh

Entnommen aus Quelle [1].

```
#!/bin/sh
logger=logger14
echo "start datatransfer"
ifdown wwan0 > /dev/null
sleep 2s
ifup wwan0 > /dev/null
counter=0
while true
  do
  if [ $counter -lt 150 ]
    then
      sleep 1s
      counter=$(( $counter+1 ))
      echo "$counter"
      if [ "$(ifconfig | grep -o wwan0)" = 'wwan0' ]
        then
          echo "GSM on"
          break

      fi
  else
      echo "Kein Internet"
      ifdown wwan0 > /dev/null
      exit
  fi

  done

/etc/init.d/openvpn restart
while true; do ping -c1 10.8.0.1 > /dev/null && break; done
echo "VPN-Tunnel on"
scp -rp /home/root/databuffer $logger@10.8.0.1:/home/$logger
if [ $? -eq 0 ]
  then
    echo "Daten erfolgreich gesendet"
    rm -r /home/root/databuffer/*
  else
    echo "Fehler bei Datenübertragung"
fi
/etc/init.d/openvpn stop
echo "VPN-Tunnel off"
ifdown wwan0 > /dev/null
```

```
43 echo "GSM off"
```

B.4. GPS_Logger_Startskript.sh

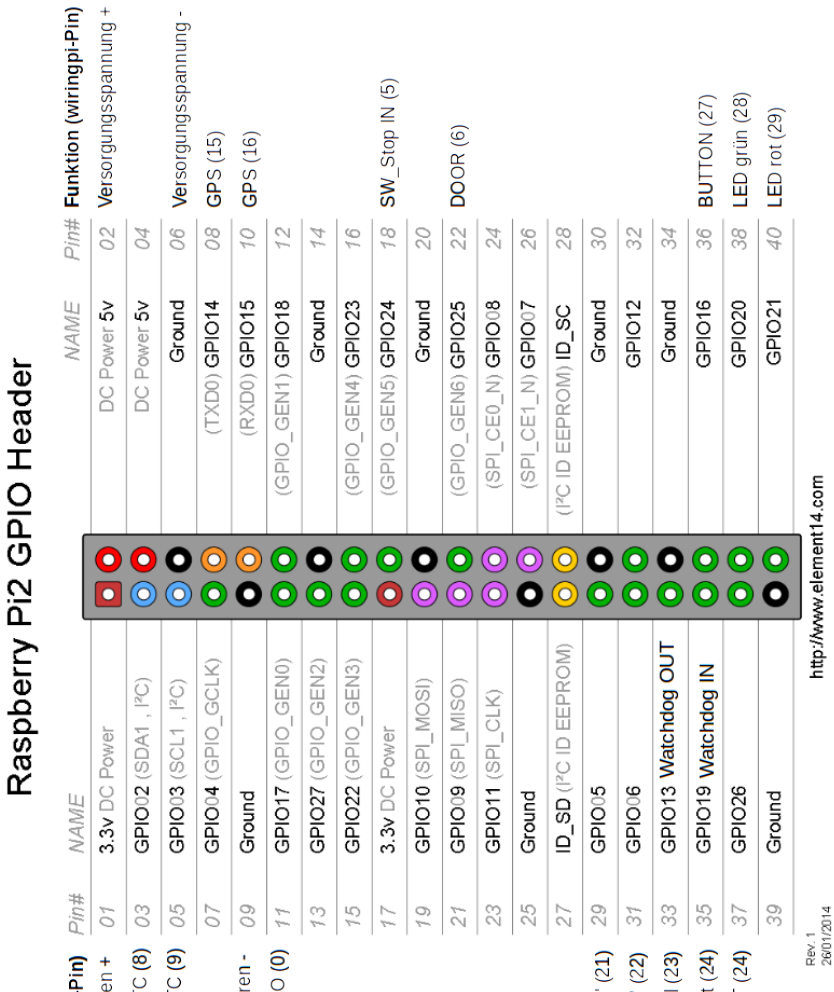
Entnommen aus Quelle [1].

```
1  # !/bin/sh
2
3  echo "set time from rtc"
4  sudo hwclock -s
5
6  sudo umount /media/*
7  sudo mkdir -p /media/INTENSO
8  sudo mount -t vfat /dev/sda1 /media/INTENSO -o uid=1000,gid=1000,dmask=0077
9
10 echo "Starte GPS Logger Skript"
11
12 sudo ifconfig wlan0 192.168.2.1
13
14 xterm -e "sudo ./datatransfer.sh"&
15 echo "datatransfer.sh gestartet"
16
17 echo "Starte GPS_Looger"
18 xterm -e "sudo /home/pi/worksC/GPS_Logger_Autostart/bin/Debug/GPS_Logger"
19 #Verzeichnis der GPS_Logger-Applikation
20 echo "GPS_Logger abgebrochen! Starte in 1 Minute neu!"
21
22 sudo shutdown -r 1
```

C. Anhang: Sonstiges

C.1. GPIO-Pin-Belegung: GPIO_PINOUT.png

Entnommen aus Quelle [23]. Bild wurde angepasst.



C.2. GPS-Test-Buffer: venus_test_buffer.txt

```
$SkyTraq,Venus6
$Kernel,v1.6.10,00030EDE,16304205,F,16.367667MHz
$ver,011023,rev,120710
$GPGGA,115944.099,0106.0000,N,00106.0000,E,0,01,0.0,1.1,M,0.0,M,,0000*61
$GPRMC,115944.099,V,0000.0000,N,00000.0000,E,00.59,001.1,280606,,,N*70
$GPGGA,115944.200,0212.0000,N,00212.0000,E,0,02,0.0,2.2,M,0.0,M,,0000*63
$GPRMC,115944.200,V,0000.0000,N,00000.0000,E,1.187,002.2,280606,,,N*72
$GPGGA,115944.299,0318.0000,N,00278.0000,E,0,03,0.0,3.3,M,0.0,M,,0000*63
$GPRMC,115944.299,V,0000.0000,N,00000.0000,E,01.78,003.3,280606,,,N*72
$GPGGA,115944.400,0424.0000,N,00424.0000,E,0,04,0.0,4.4,M,0.0,M,,0000*65
$GPRMC,115944.400,V,0000.0000,N,00000.0000,E,02.37,004.4,280606,,,N*74
$GPGGA,115944.500,0530.0000,N,00530.0000,E,0,05,0.0,5.5,M,0.0,M,,0000*64
$GPRMC,115944.500,V,0000.0000,N,00000.0000,E,02.97,005.5,280606,,,N*75

$GPGGA,115944.600,0106.0000,N,00106.0000,E,0,01,0.0,1.1,M,0.0,M,,0000*61
$GPRMC,115944.600,V,0000.0000,N,00000.0000,E,00.59,001.1,280606,,,N*70
$GPGGA,115944.700,0212.0000,N,00278.0000,E,$GPGGA,115944.200,0212.0000,N,
  00212.0000,E,0,02,0.0,2.2,M,0.0,M,,0000*63
$GPRMC,115944.299,V,0000.0000,N,00000.0000,E,01.78,003.3,280606,,,N*72
$GPGGA,115944.800,0318.0000,N,00278.0000,E,0,03,0.0,3.3,M,0.0,M,,0000*63
$GPRMC,115944.900,V,0000.0000,N,00000.0000,E,$GPRMC,115944.900,V,0000.00
  00,N,00000.0000,E,02.37,004.4,280606,,,N*74
$GPGGA,115945.000,0530.0000,N,00530.0000,E,0,05,0.0,5.5,M,0.0,M,,0000*64
$GPRMC,115945.000,V,0000.0000,N,00000.0000,E,02.97,005.5,280606,,,N*75

$GPGGA,115944.099,A106.0000,N,00106.0000,E,0,01,0.0,1.1,M,0.0,M,,0000*61
$GPRMC,115944.099,V,0000.0000,N,00000.0000,AE,00.59,001.1,280606,,,N*70
$CPGGA,115944.200,0212.0000,N,00212.0000,E,0,02,0.0,2.2,M,0.0,M,,0000*63
$CPRMC,115944.200,V,0000.0000,N,00000.0000,E,1.187,002.2,280606,,,N*72
$GPGGA,115944.299,0318.0000,N,00278.0000,E,0,03,0.0,12345678.9,M,0.0,M,,0000*63
$GPRMC,115944.299,V,0000.0000,N,00000.0000,E,01.78,123456789.0,280606,,,N*72
$GPGGA,115944.400,0424.0000N,00424.0000,E,0,04,0.0,4.4,M,0.0,M,,0000*65
$GPRMC,115944.400,V,0000.0000,N,00000.0000,E,02.37,004.4,280606,,,N*74
$GPGGA,115944.500,0530.0000,N,00530.0000,E,0,05,0.0,5.5,M,0.0,M,,0000*64
$GPRMC,115944.500,V,0000.0000,N,00000.0000,E,02.97005.5,280606,,,N*75
$
```

C.3. Ergebnis Dekodierungstest: WATCHDOG.txt

```
Busnetz aus | Akku-Spannung: 3.000V
Busnetz an | Akku-Spannung: 3.100V
2 Pulse (Undefinierter Zustand) | Akku-Spannung: 3.200V
```

```
1ter Befehl zum shutdown | Akku-Spannung: 3.300V
2ter Befehl zum shutdown | Akku-Spannung: 3.400V
3ter Befehl zum shutdown | Akku-Spannung: 3.500V
4ter Befehl zum shutdown | Akku-Spannung: 3.600V
Befehl von Watchdog: shutdown durchführen | Akku-Spannung: 3.700V
Befehl von Watchdog: shutdown durchführen | Akku-Spannung: 3.800V
Befehl von Watchdog: shutdown durchführen | Akku-Spannung: 3.900V
Befehl von Watchdog: shutdown durchführen | Akku-Spannung: 4.000V
Befehl von Watchdog: shutdown durchführen | Akku-Spannung: 4.100V
Befehl von Watchdog: shutdown durchführen | Akku-Spannung: 4.200V
Befehl von Watchdog: shutdown durchführen | Akku-Spannung: 4.300V
Befehl von Watchdog: shutdown durchführen | Akku-Spannung: 4.400V
Befehl von Watchdog: shutdown durchführen | Akku-Spannung: 4.500V
Befehl von Watchdog: shutdown durchführen | Akku-Spannung: 4.600V
Befehl von Watchdog: shutdown durchführen | Akku-Spannung: 4.700V
Befehl von Watchdog: shutdown durchführen | Akku-Spannung: 4.800V
Befehl von Watchdog: shutdown durchführen | Akku-Spannung: 4.900V
Befehl von Watchdog: shutdown durchführen | Akku-Spannung: 5.000V
Befehl von Watchdog: shutdown durchführen | Akku-Spannung: 5.100V
Befehl von Watchdog: shutdown durchführen | Akku-Spannung: 5.200V
Befehl von Watchdog: shutdown durchführen | Akku-Spannung: 5.300V
```

C.4. *Splint*-Kofigurationsdatei: .splintrc

Entnommen aus Quelle [21]. Modifizierter Inhalt wurde gekennzeichnet.

```
#------------------------------------------------------------------------------
#Start: Start: Modifiziert von Roman Kray
#------------------------------------------------------------------------------
+posixstrictlib
#-exportlocal
-fullinitblock
-initallelements
-globs #errno fehler
-I /usr/local/include/
-I /usr/include/linux/
-I /usr/include/arm-linux-gnueabihf/
-I /usr/include/x86_64-linux-gnu/
+D__signed__=signed
#------------------------------------------------------------------------------
#Ende: Modifiziert von Roman Kray
#------------------------------------------------------------------------------

#-hints
#-forcehints
```

```
 -linelen 120
# -parenfileformat
#+quiet

 -booltype  bool
 -booltrue  true
 -boolfalse false

 -standard

# the flags commented out below are switched on by default or the
# '-standard' shortcut.

 +ansi89limits
#+boolops
#+casebreak
+charint
 +cppnames
 +declundef
 +elseifcomplete
#+evalorder
 +exportheader
 +exportheadervar
#+exportlocal
 -externalnamelen 31
 +fcnmacros
#+firstcase
#-floatdouble
 +forblock
#+gnuextensions
 +ifblock
#-ignorequals
#-ignoresigns
#+imptype
#+incompletetype
#+incondefs
#+initsize
 -internalnamelen 31
 +isoreserved
 +isoreservedinternal
#-longintegral
#-longunsignedintegral
#+macroassign
#+macrofcndecl
#+macroparams
```

```
#+macroparens
#+macrostmt
 -maintype
#-matchanyintegral
#-modfilesystem
#+namechecks
#+nestcomment
#+noeffect
 +noparams
#+noret
 +oldstyle
#+predassign
#+predboolint
 +protoparammatch
#-protoparamname
 +ptrarith
#+realcompare
 +redecl
#+redef
 -relaxquals
#-relaxtypes
 +retval
#+shadow
#+shiftimplementation
#+shiftnegative
# -slashslashcomment
#+stackref
#+sysunrecog
#+type
#+unreachable
#+unrecog
#+unrecogdirective
#+usedef
#+usevarargs
 +whileblock
```

Versicherung über die Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Arbeit im Sinne der Prüfungsordnung nach §16 APSO-TI-BM ohne fremde Hilfe selbstständig verfasst und nur die angegebenen Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen habe ich unter Angabe der Quellen kenntlich gemacht.

Hamburg, 11. Juli 2016	
Ort, Datum	Unterschrift