Master Thesis

im Rahmen des
Universitätslehrganges „Geographical Information Science & Systems“
(UNIGIS MSc) am Interfakultären Fachbereich für GeoInformatik (Z_GIS)
der Paris Lodron-Universität Salzburg

zum Thema

„Adaption GPS-basierter Datenerfassung zur Bewegungs- und Verhaltensmusteranalyse von Almrindern“

Untersuchung am Beispiel des subalpinen Almgebietes Vierkaser

vorgelegt von

Benedikt Bonath
u106712, UNIGIS MSc Jahrgang 2020

Betreuerin:

Assoz. Prof. Dr. Gudrun Wallentin

Zur Erlangung des Grades
„Master of Science – MSc”

Chieming, 14. August 2023

Danksagung

An dieser Stelle möchte ich meinen aufrichtigen Dank an all jene Personen aussprechen, die mich während meiner Masterarbeit unterstützt und begleitet haben. Ohne ihre Hilfe und Ermutigung wäre die Fertigstellung dieses Projekts nicht möglich gewesen.

Ein besonderer Dank gilt dem Weißbacherbauern, Sebastian Feldbacher und seinem engagierten Vorhaben der Wiederbewirtschaftung der Almflächen an der Vierkaseralm. Ohne sein großes Engagement für die Sache und der Bereitschaft diese Masterarbeit zu unterstützen, wäre die Datenerfassung nicht möglich gewesen.

Darüber hinaus möchte ich mich bei der Sektion Salzburg des österreichischen Alpenvereins bedanken, welche einerseits das Revitalisierungsprojekt tatkräftig unterstützt und zudem finanzielle Mittel zur Beschaffung der Komponenten der GPS-Tracker bereitgestellt hat.

Ein herzliches Dankeschön gilt auch der exzellenten Betreuung von Frau Prof. Dr. Gudrun Wallentin, welche maßgeblich Anteil daran hatte, dass ich mich für dieses Thema entschieden habe und für aufkommende Fragen stets ein offenes Ohr und gute Lösungen parat hatte.

Zu guter Letzt möchte ich mich bei meiner Frau bedanken, welche mir nicht nur immer den Rücken freigehalten und gestärkt hat, in den zeit- und arbeitsintensiven Phasen dieser Arbeit, sondern auch das Projekt des Almbauern beim herrichten der alten Almwege unterstützte.

Diese Masterthesis stellt letztendlich das Ergebnis des gemeinsamen Engagements dar. Vielen Dank!

Eigenständigkeitserklärung

Ich versichere, diese Masterthesis ohne fremde Hilfe und ohne Verwendung anderer als der angeführten Quellen angefertigt zu haben, und dass die Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen hat. Alle Ausführungen der Arbeit, die wörtlich oder sinngemäß übernommen wurden, sind entsprechend gekennzeichnet.

Thauernhausen, den 14. August 2023

.....
Benedikt Bonath

Zusammenfassung

Die Erfassung von Positions- und Bewegungsdaten bei Tieren mittels satellitengestützter Technologien ist heute in vielen Forschungsfeldern ein Standard bei der Erhebung von primären Geodaten. Jedoch bedarf es, je nach Anwendungsgebiet, Tierart, äußeren Einflüssen und späterer Datenanalyse spezielle, zweckgebundene technische Parameter bei der Wahl der Datenlogger.

Die Erfassung von Individuen einer Rinderherde in einer subalpinen Almlandschaft, zum Zwecke der Bewegungs- und Interaktionsmusteranalyse stellt dabei sehr spezielle Anforderungen an die Technik. Die Wahl der Messerintervalle, Akkulaufzeiten und Umwelteinflüsse müssen berücksichtigt und abgestimmt werden, um eine synchrone Datenerfassung der einzelnen Tiere über den Zeitraum eines Almsommers gewährleisten zu können. Diese spezifischen Anforderungen konnten in den eingehenden Recherchen durch kommerzielle GPS-Logger nicht vollständig abgedeckt werden, was zu dem Schluss führte, zweckmäßige Geräte in Eigenentwicklung umzusetzen. Zu den Almsommern 2021 und 2022 wurden zwei aufeinander aufbauende Generationen von GPS-Datenloggern entwickelt und im Untersuchungsgebiet Vierkaser-Alm eingesetzt.

Das Hauptaugenmerk richtete sich dabei auf die Evaluierung und Abstimmung von spezifischen Messparametern, sowie Soft- und Hardwarekomponenten, um eine durchgehende Datenerfassung in den Messzeiträumen für den späteren Einsatzzweck der Analyse von Interaktionsmustern zu gewährleisten. Damit stellt eine zielgerichtete Datenerfassung die Grundlage für weiterführende Analysen und Forschungsfragen dar.

Bereits im ersten Almsommer konnten einige Herausforderungen beim engmaschigen Tracking von Almweiderindern ermittelt werden. Primär die Abstimmung zwischen Messintervallen und Akkulaufzeiten, sowie die Synchronität der Logger untereinander stellten sich als schwierig heraus, zumal die Tiere bereits weit vor dem Almauftrieb besendert werden mussten und Wartungen während des Almsommers nicht möglich waren.

Diese Parameter galt es zum Almsommer 2022 näher zu betrachten und zu optimieren. Aufgrund von Komplikationen mit Hard- und Software konnten jedoch nur Teilerfolge - beispielsweise bzgl. der Synchronität verzeichnet werden.

Die erzielten Messergebnisse reichen für die Analyse von Interaktionsmustern der Herde nur bedingt aus, jedoch sind die Teilergebnisse hinsichtlich der Hard- und Software-Optimierungen vielversprechend. Insbesondere die Hardware zeigte sich als robust und geeignet für den subalpinen Einsatz an Kühen.

Für eine optimale, zielgerichtete Datenerfassung benötigt es weitere Forschungsarbeit und Adaptionen der Geräte.

Abstract

The collection of position and movement data in animals, using satellite-based technology, has become a standard practice in many research fields for gathering primary geospatial data. However, depending on the application area, animal species, external influences, and subsequent data analysis, specific technical parameters need to be considered when choosing data loggers.

The capture of individuals within a cattle herd in a subalpine landscape, for the purpose of analyzing movement and interaction patterns, presents very specific requirements for the technology. The selection of measurement intervals, battery life, and environmental influences must be taken into account and coordinated to ensure synchronous data capture of individual animals over the course of an alpine summer. These specific requirements could not be fully met by commercial GPS loggers, as revealed in the extensive research, leading to the decision to develop purpose-built devices. Two successive generations of GPS data loggers were developed and deployed in the Vierkaser-Alm study area during the summers of 2021 and 2022.

The main focus was on evaluating and coordinating specific measurement parameters, as well as software and hardware components, to ensure continuous data capture during the measurement periods for the subsequent analysis of interaction patterns. Targeted data capture thus forms the basis for further analysis and research questions.

In the first alpine summer, several challenges were identified in the close tracking of alpine grazing cattle. Primarily, the coordination between measurement intervals and battery life, as well as the synchronization of loggers among themselves, proved to be difficult, especially since the animals had to be tagged well before the start of the alpine grazing season, and maintenance during the summer was not possible.

These parameters needed to be examined and optimized for the alpine summer of 2022. However, due to complications with hardware and software, only partial successes were achieved, such as in terms of synchronization.

The obtained measurement results are only partially sufficient for analyzing herd interaction patterns, but the partial results regarding hardware and software optimizations are promising. In particular, the hardware has proven to be robust and suitable for subalpine use on cows.

Further research and adaptations of the devices are needed for optimal, targeted data capture.

Inhaltsverzeichnis

Anhang

Abkürzungsverzeichnis

API Programmierschnittstelle (Application Programming Interface)

DDL Data Definition Language

EEPROM Electrically Erasable Programmable Read-Only Memory

FAT File Allocation Table

FTDI Future Technology Devices International

GLONASS Globalnaja nawigazionnaja sputnikowaja sistema

GNSS Global Navigation Satellite System

GPS Global Positioning System

GSM Global System for Mobile Communications

HTTP Hypertext Transfer Protocol

I/O Input/Output

IDE Integrated Development Environment

JSON JavaScript Object Notation

MCU Microcontroller unit

NAVSTAR Navigational Satellite Timing and Ranging

PWM Pulsweitenmodulation

SD-Karte Secure Digital Memory Card

SDHC Secure Digital High Capacity

SPI Serial Peripheral Interface

SQL Structured Query Language

USB Universal Serial Bus

VENV Virtual Environment

VSCode Visual Studio Code

LiPo Lithium-Polymer-Akkumulator

NiMh Nickel-Metallhydrid

NiCd Nickel-Cadmium

LiFePO4 Lithium-Eisenphosphat

MOSFET Metal Oxide Semiconductor Field-Effect Transistor

UART Universal Asynchronous Receiver/Transmitter

TTFF Time to first Fix

DOP Dilution of Precision

HDOP Horizontal Dilution of Precision

UBX u-blox-Protocol

LED Leuchtdiode

NMEA National Marine Electronics Association

RINEX Receiver Independent Exchange Format

BLE Bluetooth Low Energy

LoRaWAN Long Range Wide Area Network

1. Einleitung

Die satellitengestützte Positionserfassung von Tieren findet in vielen wissenschaftlichen Forschungsfeldern Anwendung. Entscheidend für die spezifischen Anforderungen und Parameter der Datenerfassung ist dabei im Wesentlichen die Fragestellung, welche durch die erhobenen Daten beantwortet werden soll. So unterscheidet sich beispielsweise die Wahl der Erfassungstechniken bei Langzeitstudien über das Revierverhalten von Individuen oder Gruppen maßgeblich zu denen, die bei zeitlich begrenzten, kleinräumigen Verhaltensmusteranalysen im Vordergrund stehen. Technische Anforderungen und Hilfsmittel, Parametrisierung der Messvariablen, räumliche Begebenheiten und Umwelteinflüsse, sowie die Art der Tiere und deren Umfeld und Haltung sorgen für die Notwendigkeit hoch spezialisierter Anpassungen an die grundlegende Methodik der primären Datenerfassung via Global Navigation Satellite System (GNSS).

1.1. Ausgangslage und Motivation

Trotz der bereits vielfältigen Erkenntnisse im Bereich der Bewegungsmusteranalyse bei Tieren, stellt die spezifische Fragestellung nach dem Interaktionsmustern von Almrindern besondere Anforderungen an die primäre, GNSS-gestützte Datenerfassung. Im Rahmen eines Revitalisierungsprojektes einer lange brachliegenden Almfläche im Untersuchungsgebiet „Vierkaseralm“ an der deutsch-österreichischen Grenze, wie es Abbildung 1.1 zeigt, wurden mehrere Kühe mit GPS-Datenloggern ausgestattet, um das Bewegungsverhalten der Tiere tiefer analysieren zu können, um damit Almbauern bei künftigen Maßnahmen und Taktiken zur Wiederbeweidung, und der daraus resultierenden ökologischen Vorteile zu unterstützen. Das Hauptaugenmerk liegt auf der engmaschigen Sammlung von Positionsdaten über den Zeitraum eines Almsommers, zur weiteren Analyse des Bewegungs- und Interaktionsverhaltens der Herde, sowie einzelner Tiere untereinander. Insbesondere letzteres stellt hohe Anforderungen an die technische Beschaffenheit von GPS-Trackern, welche nicht ohne weiteres im kommerziellen Umfeld zu finden sind. Hinzu kommen weitere Herausforderungen, welche sich durch äußere Einflüsse des subalpinen Raumes ergeben. Die Qualität und Quantität der Daten müssen einen Vergleich der einzelnen Individuen der Rinderherde ermöglichen, zeitgleich sollen Hard- und Software der Datenlogger dem exponierten, schwer zugänglichen Almgebiet über mehrere Monate standhalten. Diese speziellen Grundvoraussetzungen bedürfen einer intensiven Evaluierung und Adaptierung einer Vielzahl an Parametern, hinsichtlich der Eignung von GPS-Trackern.

Vorgelagerte Tests von kommerziellen, auf Rinder abgestimmte Datenerfassungssysteme zeigten bereits eine unzureichende Verlässlichkeit, Akkulaufzeit und Genauigkeit der aufgenommenen Daten, wie es Kapitel 2.4 verdeutlicht. Die Schlussfolgerung der vorausgehenden Recherchen mündet in der Notwendigkeit, eigens entwickelter GPS-Datenlogger für diesen Einsatzzweck.

Abbildung 1.1.: Lage und Ausmaße des Untersuchungsgebietes „Vierkaseralm“

1.2. Stand der Forschung

Technisch unterstützte Methoden zur Erfassung von Bewegungen von Landtieren gibt es bereits seit mehreren Jahrzehnten - in den Anfängen wurde dabei mit Funkhalsbändern bei Wildtieren experimentiert, wie bereits Cochran und Lord (1963) beschreiben. Mit der vollen zivilen Nutzbarkeit von Global Positioning System (GPS) ab dem Jahre 2000 wurde auch die satellitengestützte Positionserfassung sowohl bei Wildtieren, als auch bei Nutztierherden tiefer erforscht, wie Maxa, Thurner und Wendl (2015) aufzeigen. Im Kontext der gezielten Optimierung von Weideflächen für Rinder haben Turner et al. (2000) GPS-Halsbänder verwendet, um Verhaltensmuster einer Herde zu analysieren. Dabei wurde bereits ein Fokus auf die Ermittlung geeigneter Messintervalle gesetzt, wobei generell kürzere Messintervalle (< 10 min) zu einer niedrigeren Fehlerrate, be-

zogen auf alle GPS-Geräte der Herde, führten. Diese Beobachtung konnten auch Cain et al. (2005) bestätigen und betrachteten zu dem den Einfluss der Topographie auf die Fehlerrate der räumlichen Messwerte, anhand der Analyse weitere Studien. Augustine und Derner (2013) beschreiben ein 5-Minuten Intervall bei der Überwachung von grasenden Herden als ausreichend hinsichtlich der GPS-Genauigkeit und Aussagekraft des Bewegungsverlauf über die ganze Herde hinweg. Forin-Wiart et al. (2015) veröffentlichten Untersuchungen zu Leistung und Genauigkeit bei GPS-Datenloggern, wobei ebenfalls eine niedrigere Fehlerrate bei kürzeren Messintervallen festgestellt wurde, jedoch bei signifikanter Reduzierung der Batterielaufzeit. Gerade aber die Batterielaufzeit wird in der Veröffentlichung von Maxa, Thurner und Wendl (2015) als Hauptkriterium bei der Datenaufnahme von Weiderindern in alpinen Gebieten beschrieben. Studien zur konkreten Datenaufnahme oder Bewegungsmusteranalyse von Einzeltieren innerhalb einer Rinderherde - insbesondere in subalpinen Gelände - sind jedoch rar.

Weitere satellitengestützte Telemetrie Systeme stellen ARGOS und die ICARUS-Initiative dar. Der Fokus bei diesen Studien und Systemen liegt meist auf der Langzeitbeobachtung von einzelnen Wildtieren oder Herdenbewegungen in zeitlich großen Abständen oder mit niedrigen Anforderungen an die räumliche Auflösung. Die Betrachtung aller Tiere innerhalb einer Herde in schlecht erreichbarem Gelände über einen längeren Zeitraum mittels engmaschiger, hochaufgelöster Positionsbestimmung ist jedoch kaum dokumentiert.

1.3. Zielsetzung

In der Arbeit werden grundlegende Parameter zur Optimierung von GNSS-gestützter Datenerfassung ermittelt, bewertet und auf die besonderen Begebenheiten von Almrindern in subalpinen Weideflächen adaptiert. Im Einzelnen bedeutet dies die

(1) Ermittlung geeigneter zeitlicher Intervalle für die Positionserfassung und Umwelteinflüssen im Kontext der erwarteten Bewegungs- und Interaktionsmustern der Rinder.

(2) Auswertung des Anforderungsprofil für den Vergleich, Ausschluss und Entwicklung von GNSS-Geräten.

(3) Datenerhebung und -bewertung im Zeitraum eines Almsommers

(4) Analyse und Bewertung aller gesammelten Daten zur Evaluierung der optimierten Parameter

2. Methodik

Die grundlegende Methodik zur Erfassung der Almrinder basiert auf satellitengestützte Positionserfassung mittels GNSS und der textbasierten Speicherung der ermittelten Daten. Hierfür wurde im Verlauf des Projektes primär mit eigens angefertigten GPS-Datenloggern gearbeitet. Die Entwicklungs-, Beschaffungs-, Test- und Optimierungsphasen der GPS-Tracker können als Teil des methodischen Ansatzes betrachtet werden, da dies notwendige Schritte zur eigentlichen Datenerfassung darstellten. Soft- und Hardware wurden in einem agilen Prozess realisiert, wodurch Teilergebnisse von Testphasen direkt für die weitere Optimierung der Geräte herangezogen wurden. Der iterative Vorgang der agilen Entwicklung wird in Abbildung 2.1 verdeutlicht.

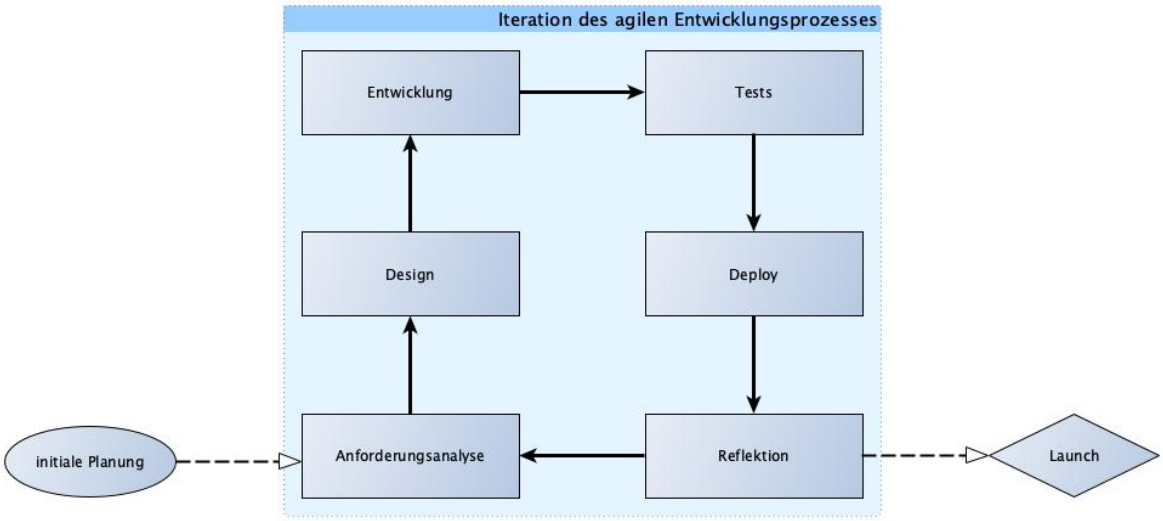


Abbildung 2.1.: Prozessbeschreibung des agilen Entwicklungsansatzes

2.1. GNSS

GNSS versteht sich als Oberbegriff für die Zusammenführung bereits etablierter, sowie künftigen Satellitensystemen im globalen Kontext. Durch die Kombination unterschiedlicher Systeme kann die räumliche Abweichung, des durch den Bodenempfänger ermittelten Standorts deutlich reduziert werden. Die Positionsgenauigkeit ist dabei einer der zentralen Parameter für die Ermittlung von individuellen Bewegungsmuster im Zu-

sammenspiel mit anderen Individuen einer Herde. Die primären Systeme, welche unter GNSS geführt werden, sind

- Navigational Satellite Timing and Ranging (NAVSTAR) – GPS
- Globalnaja nawigazionnaja sputnikowaja sistema (GLONASS)
- Beidou
- Galileo

Dabei gilt es jedoch noch technisch zu unterscheiden, zwischen GNSS der ersten Generation (GNSS-1), wozu GPS und GLONASS zählen und späteren Entwicklungen, wie Galileo, welche Systeme der zweiten Generation darstellen (GNSS-2). Verwendet wurden im Rahmen dieser Arbeit ausschließlich GNSS-1 Satelliten. Die Gründe hierfür werden im Kapitel 2.6.1 erläutert.

2.2. Arduino-Plattform

Zum Aufbau hoch spezialisierter Elektronik bietet sich als Umgebung die Arduino-Plattform an. Das 2005 gegründete OpenSource-Projekt beschäftigt sich mit der Bereitstellung von Hard- und Softwarekomponenten zur anwenderfreundlichen Programmierung von Einchipmikrorechnern, auch Microcontroller unit (MCU).

2.2.1. Hardware

Die Arduino Plattform baut im Wesentlichen auf Mikrocontroller bzw. -chips aus der megaAVR-Reihe auf. Je nach Modell kommen dabei unterschiedliche ATmega-Chips, Schwingquarze und Spannungsquellen zum Einsatz. Die gängigste Spannungsquelle sind die 5 V einer Universal Serial Bus (USB)-Versorgung, sowie die Taktung über einen 16 MHz Quarz.

Standardmäßig werden MCUs über externe Programmiergeräte mit dem entsprechenden Maschinencode versorgt. Die Arduino-Hardware bietet jedoch in der Regel einen vorinstallierten Bootloader, welche die Programmierung über serielle Schnittstellen ermöglicht. Gängig bei kleineren Boards ist dabei die Programmierung über den FT232RL-Adapter von Future Technology Devices International (FTDI), wie er auch in Abbildung 2.5 zu sehen ist, um USB-Signale in Serielle zu konvertieren.

Zur Steuerung und Kommunikation von externen Bausteinen, Sensoren und Signalen kommen bei Arduino-Boards digitale Input/Output (I/O)-Pins zum Einsatz. Diese können zwei Zustände annehmen:

HIGH Spannung liegt an

LOW Spannung liegt nicht an

Zudem können über die digitalen Pins auch Pulsweitenmodulation (PWM)-Signale übermittelt werden, um anhand der Impulsdauer einer Spannungsversorgung Steuersignale zu übermitteln. Weitere Pins dienen als Analogeingänge und werden im Rahmen dieser Arbeit nicht verwendet.

Microchip ATmega328P

Der Name Microchip ATmega328P beschreibt einen Einchipmikrocontroller aus dem Hause Microchip (vormals Atmel) der megaAVR Produktfamilie. In den Produktreihen des Herstellers stellen die ATmega-Boards einen Mittelweg zwischen Leistung und Baugröße dar.

Unter dem zentralen Aspekt einer möglichst geringen Baugröße, sowie eines niedrigen Energieverbrauchs wurde der Mikrocontroller ATmega328P verwendet, worauf auch der Arduino Pro Mini basiert. Dieser ist grundsätzlich in zwei Versionen erhältlich - als 3.3 V und 8 MHz Variante, sowie mit 5 V und 16 MHz. Technische Daten können dem Anhang G entnommen werden.

2.2.2. Software

Neben der Hardware bietet die Arduino-Plattform eine eigens mit Java entwickelte und damit multiplattformfähige (für Windows, Linux, Mac) Integrated Development Environment (IDE). Hauptbestandteile der IDE sind einerseits ein Code-Editor für die C bzw. C++ Programmierung, anderseits der freie C-Cross-Compiler `avr-gcc`, welcher auf dem gängigen `gcc`-Compiler aufbaut, erweitert durch die AVR-Library und weiteren Hilfsprogrammen. Diese Toolchain ermöglicht eine stark vereinfachte Programmierung von AVR-Mikrocontrollern in C bzw. C++ ähnlicher Syntax.

Für die Kompilierung eines lauffähigen Programms auf dem Mikrocontroller genügt es dabei zwei bestimmte Funktionen zu definieren. Die `setup()` Funktion dient der initialen Definition von Variablen und I/O-Werten der Pins. Der Quellcode innerhalb der `loop()` Funktion wird nach der Initialisierung in einer Endlosschleife durchlaufen und steuert in der Regel das Verhalten der MCU zur Laufzeit.

2.2.3. PlatformIO

Eine Schwäche der Arduino IDE stellt der Code-Editor dar, welcher dem Programmierer nur rudimentäre Unterstützung bietet. Inzwischen gängige Konzepte wie IntelliSense, automatische Codeformatierung oder Syntaxhighlighting unterstützt die IDE nicht.

Um diesen Nachteil auszugleichen kann in einem externen Editor oder einer IDE das plattformübergreifende Tool „PlatformIO“ verwendet werden. PlatformIO unterstützt dabei deutlich mehr Plattformen als nur die von Arduino verwendet Microchip AVR-Boards.

Im Rahmen dieser Arbeit wurde PlatformIO als Visual Studio Code (VSCode) - Erweiterung installiert und verwendet. Dadurch können sowohl die unterstützenden Features und Extensions von VSCode verwendet werden, zeitgleich wird der Code weiterhin über den modifizierten gcc-Compiler der Arduino-IDE kompiliert.

Darüber hinaus bietet PlatformIO eine integrierte Paketverwaltung für externe Softwarelibraries an und ermöglicht eine einfache Kompilierung des gleichen Codes für verschieden MCUs.

2.3. Ermittlung der Messintervalle

Die Ermittlung der Messintervalle, also den Abständen zwischen den einzelnen ausgewerteten GPS-Fixes, wurde aus der Literatur abgeleitet. In der Studie von Maxa, Thurner und Wendl (2015) wurden Bewegungen von Kühen in den italienischen Alpen hinsichtlich der zurückgelegten Wegstrecke mit unterschiedlichen Intervallen und GPS-Systemen untersucht. Dabei wurde bereits bei Erhöhung des Messintervalls vom niedrigsten Wert der Studie - 5 Minuten - auf 10 Minuten eine signifikante Reduzierung der Bewegungsinformationen von 38% festgestellt. Sie beschreiben zeitgleich, dass die Batterielaufzeit eines der Kernprobleme, bezogen auf die Intervalle bedeutet.

Nakano et al. (2020) verwendet Intervalle von 15 s bei der Analyse von individuellem Futterverhalten von Kühen, wobei die Messperiode der GPS-Daten nur zwei Tage betrug.

In Anbetracht der erwarteten Messperiode von > 50 Tagen innerhalb eines Almsommers und den Ergebnissen von Maxa, Thurner und Wendl (2015) hinsichtlich der Bewegungserfassung und Batterielaufzeiten, wird zunächst ein Messintervall von 300 s festgelegt, um eine ausgeglichene Berücksichtigung zwischen Auswertbarkeit der Interaktionsmuster und der Abdeckung des gesamten Almsommers zu erreichen.

2.4. Evaluierung zweckmäßiger kommerzieller Geräte

In Vorbereitung zur Datenerfassung wurde ein kommerzielles Ortungsgerät auf seine Tauglichkeit hin überprüft. Das Gerät „QTrack S2“ soll, in Kombination mit einer app- oder browserbasierten Ortungssoftware auf die Belange einer satellitengestützten Bewegungsüberwachung von Rindern zugeschnitten sein. Neben der GPS-Technik unterstützt der Tracker auch ein Global System for Mobile Communications (GSM)-Modul, um die erfassten räumlichen Daten direkt per mobilen Funknetz an die proprietäre Serversoftware zu übermitteln, welche dann in der Clientsoftware ausgewertet und visualisiert werden (LiveTracking). Am Server selbst werden die Daten über einen längeren Zeitraum vorgehalten, sodass auch eine Ableitung der Bewegungsmuster ermöglicht wird.

Im Standard befindet sich das Gerät in einem zeitlichen Intervall von 2h in dem die Daten erfasst werden. Gemäß der Gerätebeschreibung ist dabei eine Reduktion auf minimal 1h-Intervalle möglich, nach Herstellerangaben auch ein Echtzeit-Tracking in

einem experimentellen Modus einstellbar. Die Akkulaufzeit wird, in Abhängigkeit des Aufnahmeintervalls über die Dauer eines gesamten Almsommers angegeben.

Wie im Kapitel 2.3 beschrieben, sind Messintervalle von minimal 1h für eine Betrachtung der Interaktionsmuster von individuellen Rindern einer Herde nicht ausreichend, sodass für diese Fragestellung nur der experimentelle Echtzeit-Modus in Frage kommt. Erste Tests, ohne die Montage an Tieren, in diesem Modus zeigten bereits teilweise Lücken in der Übermittlung der Daten, sehr niedrige Akkulaufzeiten (< 2 Wochen) sowie komplette Ausfälle der Geräte, welche nur durch einen manuellen Neustart behoben werden konnten. Im Test waren diese Neustarts noch gut zu bewerkstelligen, was sich allerdings bei, an Rinderhälsen montierten Trackern, als deutlich schwieriger erweisen dürfte. Diese grundlegenden Problematiken warfen primär die Frage nach speziell entwickelten GPS-Trackern zur Beantwortung der Forschungsfrage auf.

In der Feldstudie wurden die QTrack-Geräte dennoch verwendet, um ggf. einen Vergleich zu den spezialisierten Trackern zu ermöglichen.

2.5. Wetterdaten

Neben den Bewegungsdaten werden auch Daten zum aktuellen Wettergeschehen im Versuchszeitraum ermittelt und gespeichert, um Abhängigkeiten zwischen bestimmten Wettergeschehnissen oder Temperaturveränderungen und den Bewegungsverläufen zu analysieren. Abgerufen werden die Daten über den zyklischen Aufruf einer offenen webbasierten Programmierschnittstelle (Application Programming Interface) (API) mittels Python-Skript. Die Speicherung erfolgt in einer Structured Query Language (SQL) Datenbank.

2.5.1. OpenWeatherMap API

Die Datengrundlage zur Ermittlung der Wetterdaten stellt dabei die webbasierte API von OpenWeatherMap. Zur Ermittlung der aktuellen Wetterdaten für ein bestimmtes Gebiet erfolgt ein parametrisierter Hypertext Transfer Protocol (HTTP)-Aufruf an einen bestimmten Endpoint per GET-Methode. Der gewählte `OneCall`-Endpoint liefert dabei folgende Daten im JavaScript Object Notation (JSON)-Format für ein definiertes Gebiet:

- Aktuelles Wetter
- Minütliche Vorhersage für den Zeitraum einer Stunde
- Stündliche Vorhersage für den Zeitraum von 48 Stunden
- Tägliche Vorhersage für den Zeitraum von acht Tagen
- Warnmeldungen des nationalen Wetterdienstes
- Historische Wetterdaten

Durch die Query-Parameter im Request können bestimmte Datenpakete ausgeschlossen werden. Der Fokus wurde hierbei auf die jeweils aktuellen Wettergeschehnisse gelegt, so wurden die Vorhersagedaten und historische Verläufe aus dem Antwort-JSON ausgeschlossen. Der Request baut sich dabei wie folgt auf:

lat Breitengrad des Untersuchungsgebietes

lon Längengrad des Untersuchungsgebietes

appid Persönlicher Schlüssel zur Nutzung der WebAPI

units Darstellungsform der Werte

exclude Auswahl der nicht benötigten Informationen

Eine detaillierte Beschreibung der API-Parameter stellt OpenWeatherMap per Online-Dokumentation (*One Call API 3.0 - OpenWeatherMap* (2022)) bereit. Der im Python-Skript verwendet GET-Aufruf wird in Quellcode 2.1 gezeigt - ein Beispiel der zugehörigen Antwort des OpenWeatherMap-Servers in Quellcode A.1.

Quellcode 2.1: Im Skript verwendeter WebAPI Aufruf

```
https://api.openweathermap.org/data/2.5/onecall?
      lat=47.71438&
      lon=12.95150&
      appid=<OwnAPIKey>&
      units=metric&
      exclude=minutely,hourly,daily
```

Die Metriken zur Datenermittlung, Datenqualität und Modellberechnung beschreibt OpenWeatherMap auf der eigenen Website (*Accuracy and quality of weather data - OpenWeatherMap* (2022)).

2.5.2. Datenbank-Schema

Zur Speicherung der Wetterdaten dient eine PostgresSQL-Datenbank mit lediglich einer Tabelle. Als Primärschlüssel dient der Timestamp der Wetterdaten im Epoch-Format. Dadurch wird auf einfache Weise verhindert, dass zeitlich doppelte Datensätze in der Datenbank hinterlegt werden. Die Daten selbst werden in einer Spalte vom Datentyp `JSONB` hinterlegt. Quellcode 2.2 zeigt das Data Definition Language (DDL) Skript für die Datenbank.

Quellcode 2.2: DDL Skript zur Erstellung des Datenbank-Schemas

```
1  CREATE DATABASE vierkaser ENCODING = 'UTF8' LOCALE = 'en_US.UTF-8';
2
3  CREATE USER vierkaser;
4  ALTER USER vierkaser with password '<newpassword>';
5
6  ALTER DATABASE vierkaser OWNER TO vierkaser;
7
8  CREATE TABLE IF NOT EXISTS public.weather
9  (
10     id bigint NOT NULL,
11     data jsonb,
12     CONSTRAINT weather_pkey PRIMARY KEY (id)
13 )
14
15 ALTER TABLE IF EXISTS public.weather
16     OWNER to vierkaser;
```

2.5.3. Zyklische Abfrage der Wetterdaten

Die regelmäßige Abfrage der Wetterdaten über die OpenWeatherMap-API erfolgt über das Python-Skript von Quellcode 2.3, welches als Dienst innerhalb einer Virtual Environment (VENV) ausgeführt wird. Das Ergebnis des Requests wird in die `weather`-Tabelle der Datenbank geschrieben, insofern es keine Constraint-Verletzung des Primärschlüssels zur Folge hätte, wodurch die Integrität der Tabelle gewahrt wird.

Quellcode 2.3: Python-Skript zur Abfrage der Wetterdaten

```
1  import request
2  import psycopg2
3  import time
4
5  # endless while loop for running as a service
6  while True:
7    # call OpenWeatherMap OneCall API
8    url = 'https://api.openweathermap.org/data/2.5/onecall'
9    params = {
10     'lat': '47.71438', # center of AOI
11     'lon': '12.95150',
12     'appid': '<OwnAPIKey>',
13     'units': 'metric',
14     'exclude': 'minutely,hourly,daily'
15   }
16   response = request.get(url, params)
17   result = response.json()
18
19   # write data to postgres db
20   conn = psycopg2.connect('dbname=<dbName> user=<dbUser>')
21   cur = conn.cursor()
22   # pkey is timestamp of weatherdata
23   # do not insert same timestamp twice to ensure unique weatherdata (
         constraint handling)
24   sql = cur.execute("INSERT INTO weather(id, data) VALUES ({0}, '{1}') ON
         CONFLICT ON CONSTRAINT weather_pkey DO NOTHING;"
25       .format(result.get('current').get('dt'), response.text))
26   conn.commit()
27   cur.close()
28
29   time.sleep(300) # wait 300 seconds until starting next loop
```

Ausgeführt wird das Skript über einen Debian Linux Systemd-Dienst, welcher sich im Fehlerfall automatisch neustarten würde. Definiert wird dieser Hintergrundprozess über das Skript aus 2.4.

Quellcode 2.4: Systemd Skript zur Initialisierung des Dienstes

```
1  [Unit]
2  Description=WeatherApi Service
3  After=multi-user.target
4  Conflicts=getty@tty1.service
5
6  [Service]
7  Restart=on-failure
8  RestartSec=5s
9  Type=simple
10 # path to python venv bin and python script
11 ExecStart=/root/vierkaser/bin/python /root/vierkaser/weather.python
12 StandardInput=tty-force
13
14 [Install]
15 WantedBy=multi-user.target
```

2.6. Entwicklung der GPS-Datenlogger

Bei der Entwicklung der GPS-Datenlogger, gehen Methodik, sowie Ermittlung und Auswertung von Teilergebnisse einher. Nur durch die vorzeitige Auswertung von bestimmten Mess- und Evaluierungsergebnissen war eine weitere Entwicklung und Optimierung der Geräte möglich, um noch vor den eigentlichen Datenaufnahmen im Almsommer ausgereifte Datenlogger zu erhalten. Dieses Vorgehen orientiert sich im Kern an der Projektstruktur einer agilen Softwareentwicklung. Der Prozess der Entwicklung gliedert sich im Grundsatz in drei wesentliche, rekursive wiederkehrende Teilbereiche:

- Methoden zur Evaluierung der benötigten Hardware-Komponenten
- Entwicklung der Soft- bzw. Firmware zur Datenaufnahme
- Auswertung von Teilergebnissen, Tests und Optimierung der Geräte

Abbildungen der fertigen GPS-Tracker sind in den Fotografien 2.9 und 2.14 zu sehen. Montiert am Tier, zeigt es Abbildung 2.10.

2.6.1. Komponenten

In einem ersten Schritt galt es die geeigneten elektronischen Bausteine und deren Kenndaten zu recherchieren und zu bewerten. Generell wurden diese unter den Hauptgesichtspunkten Bauteilgröße, Stromverbrauch, Verfügbarkeit, Handling und Marktpreis ausgewählt. Im Kern besteht der GPS-Datenlogger aus den vier Hauptkomponenten Mikrocontroller, GPSModul, einem Modul zur Speicherung der ermittelten Daten sowie einer unabhängigen Stromversorgung. Dazu kommen ergänzende Bauteile, wie Kondensatoren für die Schaltung, Leitungen und Widerstände, sowie weitere Hilfsmittel wie Lademodule der Stromversorgung oder Programmiergeräte für die Anpassung der MCUs.

MCU

Wie im Kapitel 2.2.1 bereits erwähnt, fiel die Wahl bei der Mikrocontroller-Einheit auf ein ATmega328p-basiertes Board mit dem Handelsnamen „Pro Mini“, beispielhaft in Abbildung 2.2 dargestellt.

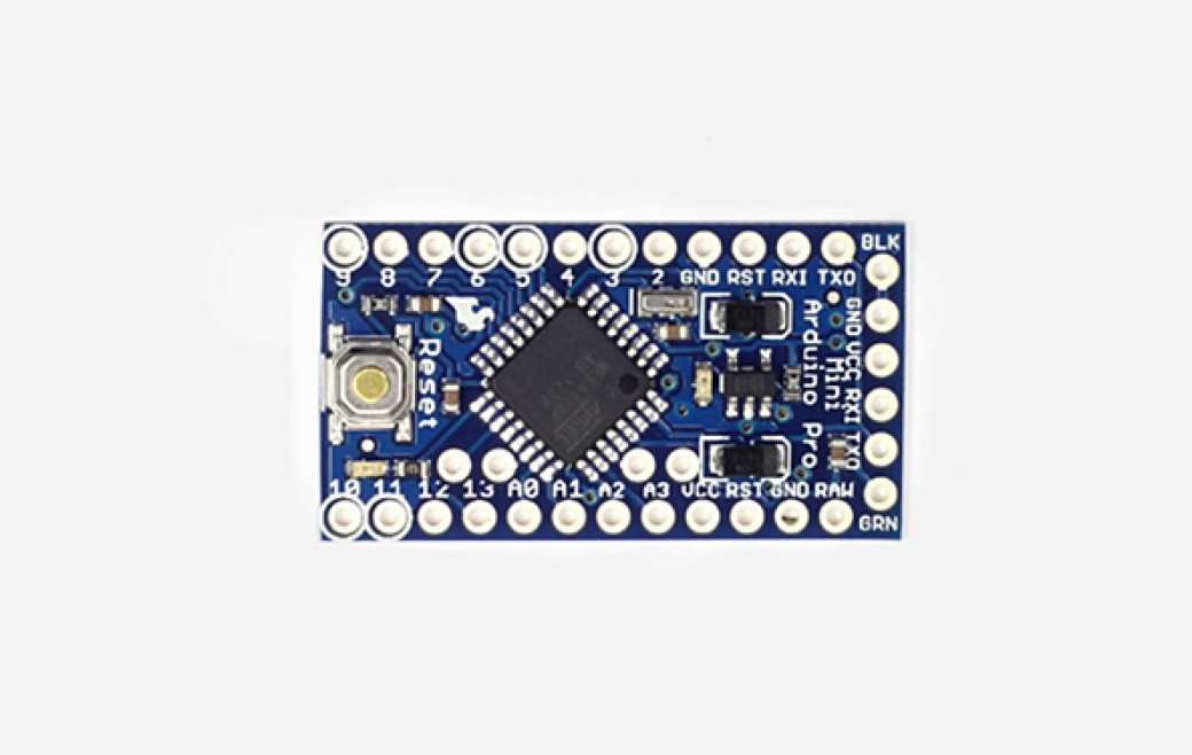


Abbildung 2.2.: Mikrocontroller: Arduino Pro Mini

Dieser Mikrocontroller stellt dabei einen Kompromiss der Hauptanforderungen dar. Trotz der geringen Größe des Boards, liefert die Platine ausreichend I/O Pins und Leistung bei einem vergleichsweise niedrigen Energieverbrauch. Ein Auszug der technischen Daten aus dem offiziellen Datenblatt (*ATmega328P | Microchip Technology* (2022)) ist in Tabelle 2.1 zu sehen.

Tabelle 2.1.: Technische Daten Pro Mini Board

MCU	Atmega328p – 8 BIT AVR controller
Operating Voltage	3.3 V
Raw Voltage Input	3.35 V - 12 V
Max. I/O Output	40 mA
Max. Output Total	200 mA
Flash Memory	32 KB
EEPROM	1 KB
Clock Frequency	8 MHz
Operating Temperature	-40 °C - 85 °C
Width (Board)	28 mm
Height (Board)	34 mm

GPS-Modul

Als GPS-Modul wurde das Modell BN-220 von Beitian verwendet, dessen Datenblatt in Anlage H hinterlegt ist. Die Vorteile dieses GPS-Modul liegen in der kompakten Bauform, der integrierten Antenne, den niedrigen Anschaffungskosten, dem vergleichbar geringen Stromfluss und der Verwendbarkeit in einem Spannungsbereich von 3.0 V bis 5.5 V. Dieser Baustein basiert auf einem u-blox[1] Chipsatz und ist damit komfortabel über die Software `u-center` konfigurierbar und auswertbar. Wie in den technischen Daten angegeben, können verschiedene Satellitensysteme und Kanäle verwendet werden. Zudem bietet der u-blox-Chipsatz einen Energiesparmodus (PowerSave-Mode). Zur Ermittlung der optimalen Kombination zwischen minimalen Stromverbrauch und minimaler Dilution of Precision (DOP), also der Streuung der Messpunkte, wurden mehrere Messungen mit verschiedenen Parametern hinsichtlich der verwendeten Systeme (GPS, GPS-GLONASS, GPS-GLONASS-GALILEO) und Energie-Modi (PowerSave-, Continuous-Mode) durchgeführt. Zudem wurden Time to first Fix (TTFF)-Werte, also die Zeiten bis zum ersten validen GPS-Signal, in den unterschiedlichen Startsequenzen (Cold-, Warm-, Hotstart) ermittelt. Die gemessenen Werte sind im Anhang D dargestellt.

Die Strommessungen erfolgten über die serielle Einbindung eines Multimeters in den Stromkreislauf. Es wurden jeweils die minimalen und maximalen Werte einer Probe über die Min/Max-Speicherfunktion des Multimeters ermittelt und notiert. Die Beobachtungen des Momentanverbrauchs am Display bestätigten in der Regel die errechneten Mittelwerte der Min-/Max-Messungen. In den technischen Daten des Messgerätes[2] wird eine Messgenauigkeit von $\pm 1.2\%$ angegeben. Die Kabelführung zum GPS-Modul wurde so weit verlängert, dass das Modul außerhalb des Gebäudes, am Fenster hängend, platziert werden konnte. Die Anpassungen und Messungen der weiteren Werte, geschah über eine Verbindung des GPS-Moduls via Universal Asynchronous Receiver/Transmitter (UART)-Schnittstelle mit der `u-center` Software mit Hilfe eines USB-TTL-Moduls, welches auch die 3.3 V-Versorgungsspannung liefert. Des Messaufbau wird in Abbildung 2.5 ersichtlich; die Verwendung der Software in den Abbildungen 2.4 und 2.5. Zur Ermittlung der TTFF- und DOP-Werte wurde das GPS-Modul teils mit dem proprietären u-blox-Protocol (UBX) abgefragt, wie Abbildung 2.4 zeigt.

Ein Vergleich der durchschnittlichen Streuungen (DOP), TTFF und Stromflüsse zeigte, dass die Kombination von GPS- und GLONASS-Satelliten, den besten Mittelweg der drei betrachteten System-Konstellationen darstellt. Unerwartet war dabei der höhere TTFF-Wert bei Nutzung der GPS-, GLONASS- und GALILEO-Kanäle, gegenüber der Nutzung von potenziell weniger verfügbaren Satelliten. Diagramm 2.3 spiegelt den Durchschnitt aller gemessenen Werte für die System-Konstellationen, unabhängig von der jeweiligen Startsequenz, wider.

[1] u-blox ist ein führendes Schweizer Unternehmen im Bereich von GNSS-Halbleiterbausteinen

[2] Kaiweets HT118A: technische Daten - `https://kaiweets.com/de-de/products/kaiweets-ht118a-digital-ac-dc-multimeter-trms-6000-counts?variant=42362992656561`

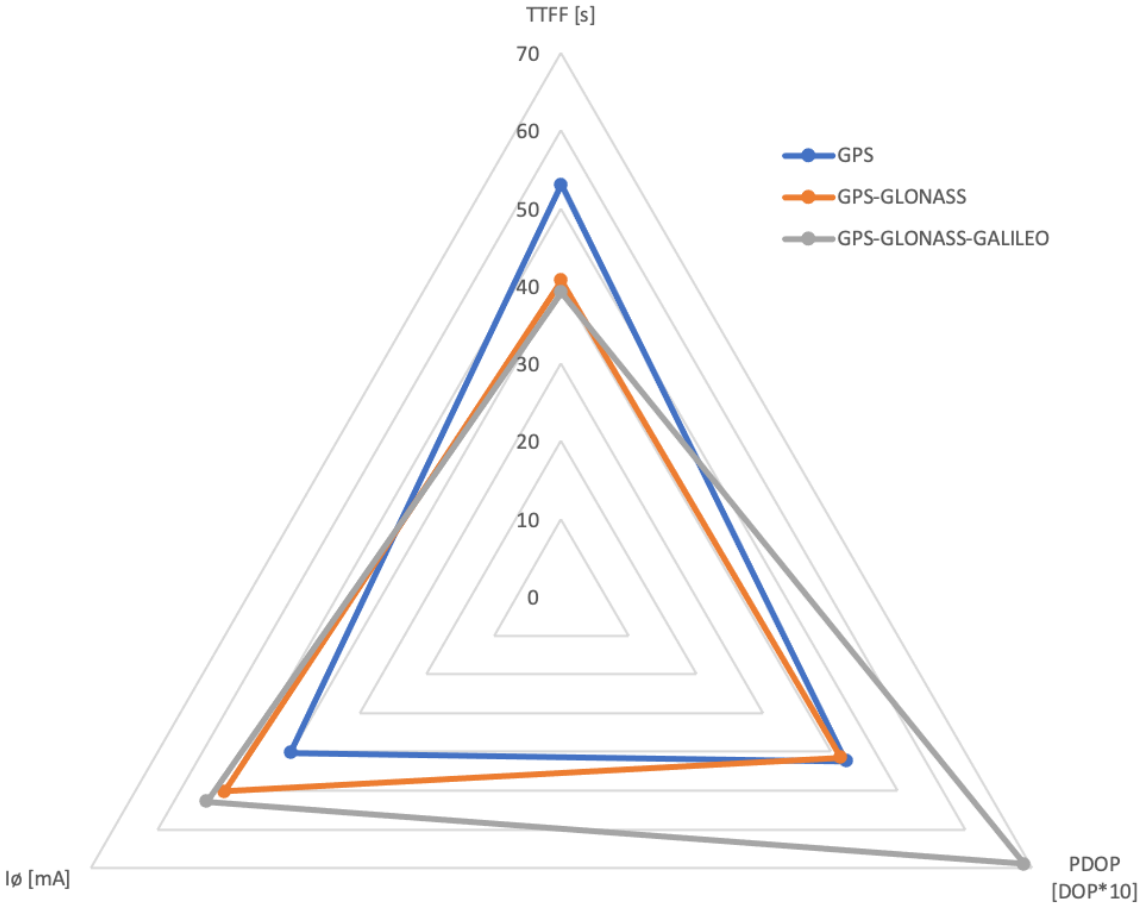


Abbildung 2.3.: Durchschnittswerte der gemessenen DOP-, TTFF-, I-Werte nach Satellitensystem - Konstellationen

Eine Berechnung der durchschnittlichen Stromaufnahme über die TTFF ergibt, dass ebenfalls die Kombination GPS-GLONASS am günstigsten hinsichtlich des Energieverbrauchs zu sein scheint, wie es Tabelle 2.2 zeigt.

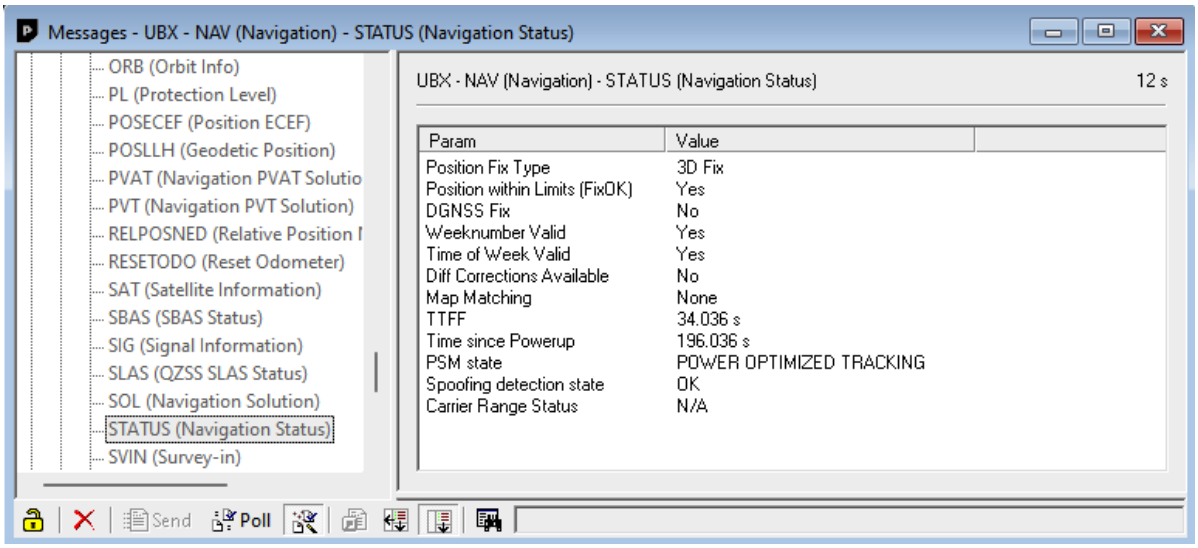
Tabelle 2.2.: ø-Stromfluss über TTFF

	TTFF [s]	$\bar{I}$ [mA]	$I \cdot TTFF$
GPS	53.07	42.33	2136.84
GPS-GLONASS	40.71	41.55	2041.04
GPS-GLONASS-GALILEO	39.16	68.83	2067.52

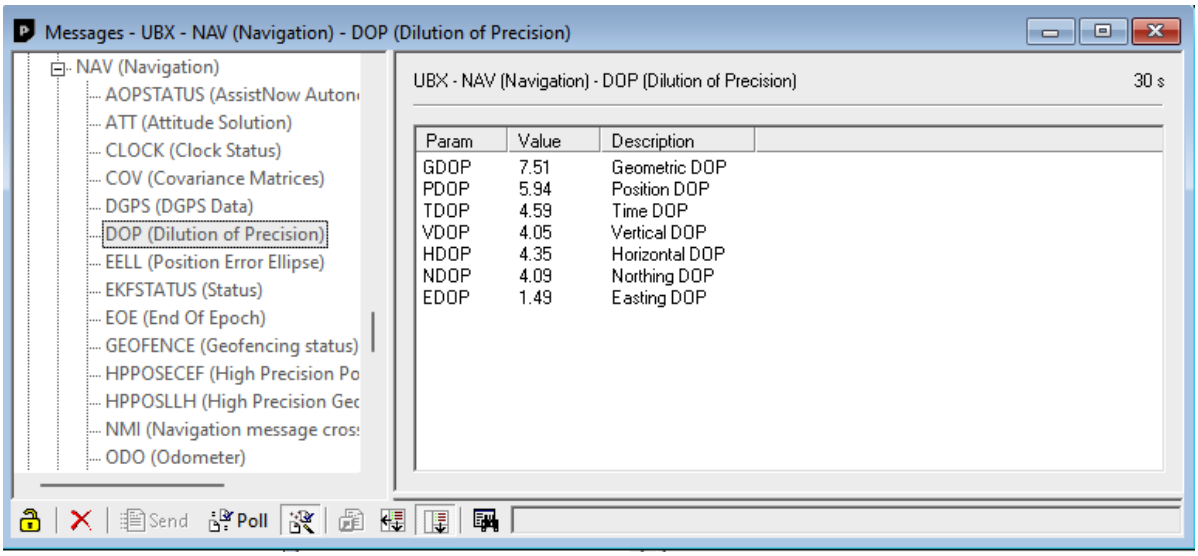
Eine Messreihe, bezogen auf die beiden möglichen Energie-Modi ergab, dass der Energiesparmodus zwar mit durchschnittlich 31,7 mA durchaus weniger Strom verbraucht, jedoch zu Lasten der DOP-Werte, womit potenziell eine niedrigere Genauigkeit erreicht

werden könnte. Das GPS-Modul wurde dabei im Warm-Start unter Nutzung der GPS- und GLONASS-Satelliten verglichen, mit dem Äquivalent im Continuous-Modus. Die TTFF- und die maximalen Strom-Mittelwerte waren allerdings sehr ähnlich, wie die Ergebnisse in Anlage E zeigen. Dieser Modus rechnet sich entsprechend eher in Anwendungsfällen, bei denen die räumliche Genauigkeit eine untergeordnete Rolle spielt oder das GPS-Modul im Dauerbetrieb verwendet wird.

Die Auswertungen dieser vorbereitenden Ergebnisse führten dazu, dass die Module für GNSS (GPS+GLONASS) im Continuous-Mode konfiguriert wurden.



(a) STATUS Informationen



(b) DOP Informationen

Abbildung 2.4.: UBX Einträge zur Ermittlung des Status und der DOPs

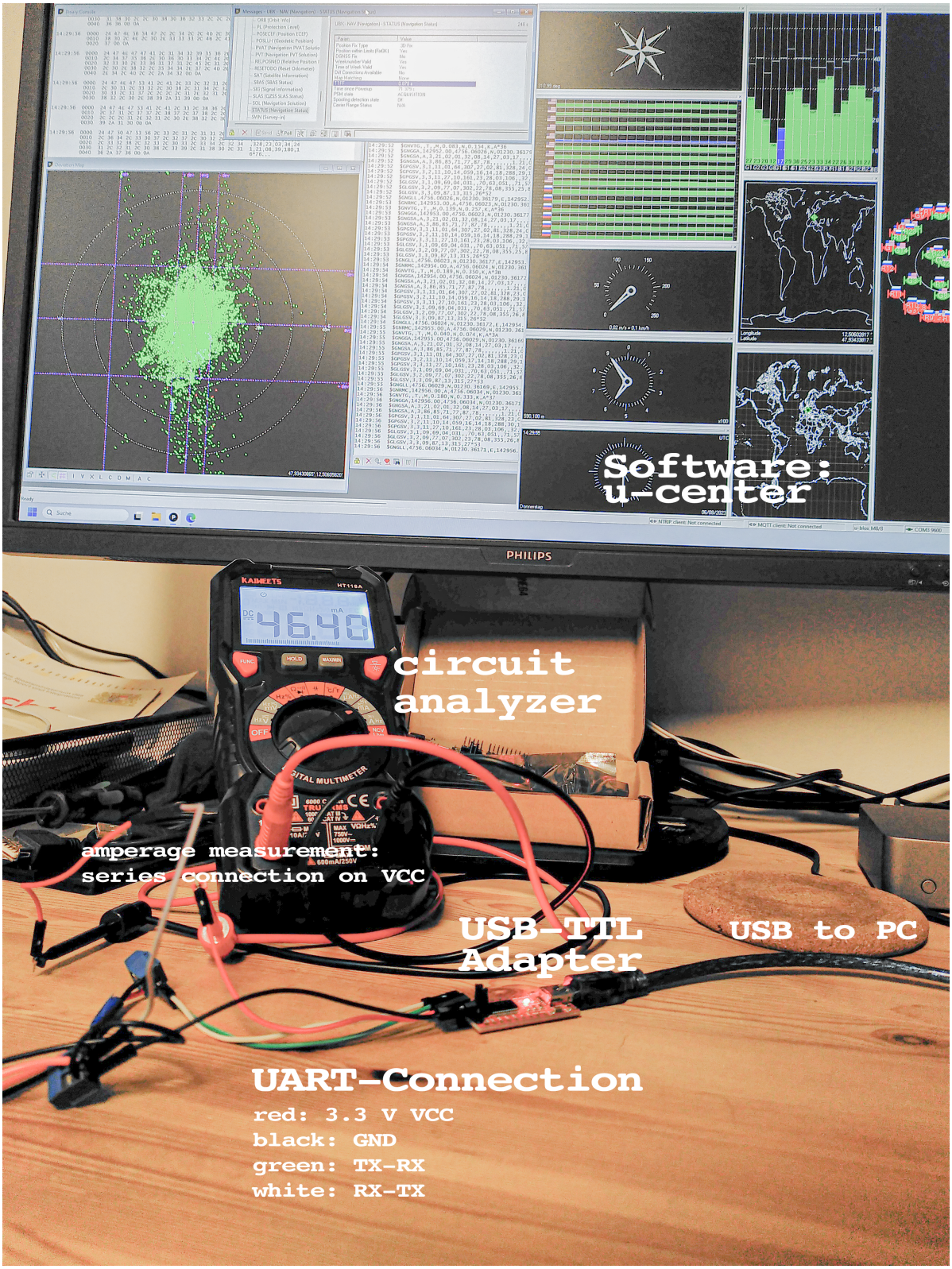


Abbildung 2.5.: Darstellung des grundlegenden Mess- und Programmieraufbaus zur Anpassung des GPS-Moduls

Speichermodul

Zur Speicherung der Positionsdaten wird ein nichtflüchtiger Datenspeicher in ausreichender Größe und Schreibgeschwindigkeit benötigt. Die Speicherbegrenzung des Onboard Electrically Erasable Programmable Read-Only Memory (EEPROM) des Mikrocontrollers von nur 1 KB bzw. des Flash-Speichers von 32 KB, welcher auch für die Speicherung des Quellcodes verwendet wird, macht es notwendig die ermittelten Daten auf ein externes Speichermedium zu schreiben. Hierfür wurde ein zusätzliches Modul mit dem Mikrocontroller verbunden, zur persistenten Speicherung auf einer Secure Digital Memory Card (SD-Karte), welches in Abbildung 2.6 zu sehen ist.

Das Hauptkriterium bei der Wahl des Kartenlese-Moduls ist eine niedrige Stromaufnahme. Der Baustein wird im Standard mit 3.3 V betrieben und benötigt dadurch keinen zusätzlichen Spannungsregler, welcher zu einem erhöhten Energiebedarf führen würde, wie es bei vergleichbaren Kartenlesern häufig der Fall ist. Zudem sollte das Modul eine möglichst kleine Bauform aufweisen, was durch den gewählten microSD-Standard unterstützt wird.

Die Kommunikation zwischen MCU und Kartenlese-Modul läuft über das standardisierte Serial Peripheral Interface (SPI), welches nach dem Master-Slave-Prinzip funktioniert, entwickelt von Hill, Jelemensky und Heene (1989). Im Kapitel 2.6.2 ist diese Verbindung schematisch dargestellt.

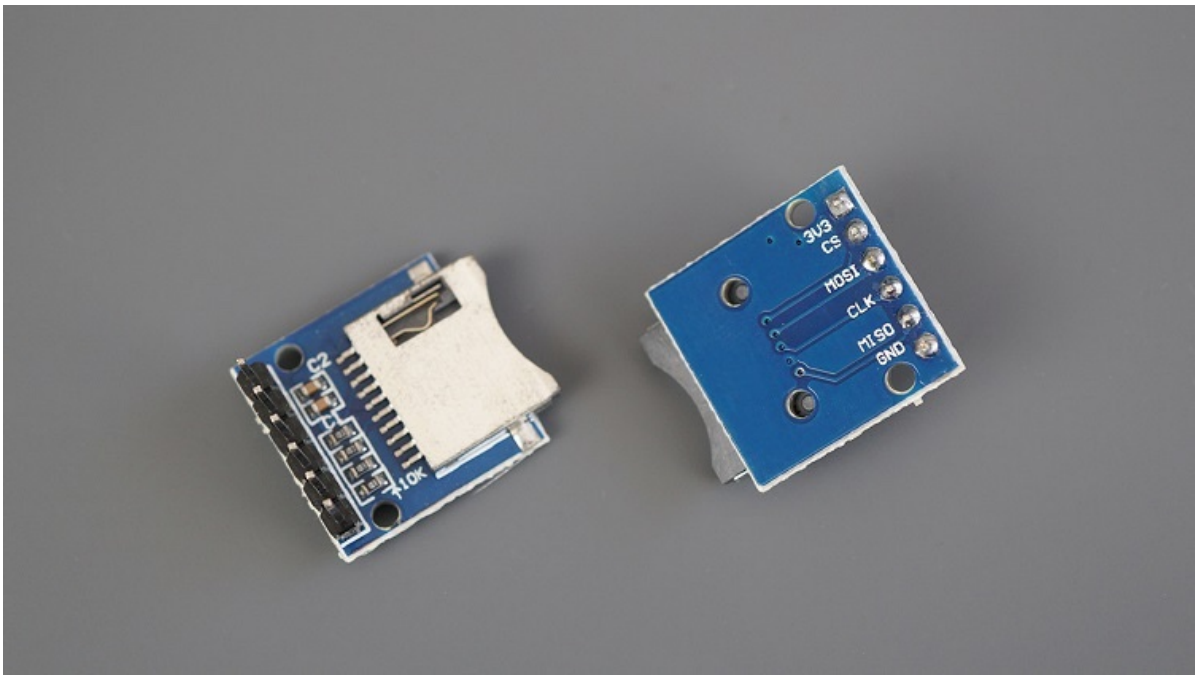


Abbildung 2.6.: SD-Kartenleser Modul

Da die zu erwartende Schreiblast und Datenmenge bei den GPS-Daten im genannten Intervall als sehr gering einzustufen ist, wurden bei der Wahl des Speichermediums keine expliziten Grundvoraussetzungen gestellt.

Bei den zu schreibenden ASCII-Daten wird pro Zeichen ein Speicherbedarf von 1 Byte benötigt. Die aufbereiteten Positionsdaten umfassen hochgerechnet pro Messwert max. 100 Zeichen und damit 100 Byte pro Schreibvorgang. Die Anzahl der Schreibvorgänge wird wiederum bestimmt durch das gewählte Messintervall. Bei einem angenommenen minimalen Messintervall von 60s, ergibt sich eine durchschnittlich benötigte Schreibgeschwindigkeit von 1,67 B/s und eine Speicherkapazität von maximal 17,57 MB, insofern das Messintervall konstant über den gesamten Almsommer (Juni bis September; 122 Tage) eingehalten werden könnte. Bedingt durch weitere Faktoren, wie Akkulaufzeiten, Adaption der Messintervalle, technische oder externe Einflüsse in der Positionsbestimmung innerhalb des Messintervalls oder des tatsächlichen Zeitraums des Almsommers sind dies nur hochgerechnete Näherungswerte zur Ermittlung der Mindestanforderung an das Speichermedium.

Verwendet wurden SD-Karten nach dem Secure Digital High Capacity (SDHC) - Standard mit der Geschwindigkeitsklasse C4, was einer sequenziellen Mindestschreibgeschwindigkeit von 4 MB/s entspricht und einem verfügbaren Speicherplatz von 4 GB, womit die angenommenen Mindestanforderungen um ein Vielfaches überschritten wurden.

Die Speichermedien wurden mit dem File Allocation Table (FAT)32-Dateisystem formatiert.

Stromversorgung

Die Stromversorgung stellt eines der Hauptkriterien und Hauptherausforderungen dar. Die nur mühsam erreichbaren Almweideflächen, starke Witterungseinflüsse im subalpinen Gelände, sowie das Artverhalten der Weidetieren selbst, stellen eine große Herausforderung an die Wahl der Energieversorgung, da ein Batteriewechsel während des Almsommers nicht umsetzbar ist.

Durch den Mikrocontroller vorgegeben wird lediglich die Betriebsspannung von mindestens 3.3 V und, bei Verwendung des integrierten Spannungsreglers, einer maximalen Spannung von 12 V. Wie in Kapitel 2.6.4 beschrieben, ist jedoch der Spannungsregler ein zusätzlicher Verbraucher, welcher durch die Wahl der passenden Stromquelle vermieden werden kann. In jedem Fall sind die Vor- und Nachteile unterschiedlicher mobiler Stromversorgungstechnologien abzuwägen. Die wesentlichen Vergleichsparameter waren hierbei:

- Nennspannung
- Kapazität
- Gewicht und spezifische Energiedichte

- Baugröße und -form
- Stabilität und Sicherheit
- Verluste durch Selbstentladung

Ein Vergleich verschiedener gängiger Akkutechnologien und Bauformen führte zur Wahl eines flachen Lithium-Polymer-Akkumulator (LiPo), begründet durch mehrere Vorteile gegenüber anderen bekannten Technologien, wie Nickel-Metallhydrid (NiMh), Nickel-Cadmium (NiCd), Lithium-Eisenphosphat (LiFePO4) und weiterer Lithium-Ionen-Derivate. Lithium-Polymer-Akkus sind in Bereichen, wie Selbstentladung, Spannung und Energiedichte vergleichbar mit anderen Lithium-Ionen-Batterien, wobei die Polymerbasis einen entscheidenden Vorteil bei der Bauform der Akkuzelle darstellt, und flache Zellformen zulässt, welche wiederum platzsparend verbaut werden können. Beim Vergleich zu Nickel-basierten Batterien ist vor allem die deutlich höhere spezifische Energiedichte und der daraus resultierende Gewichtsvorteil bei gleicher Kapazität, sowie eine geringere Selbstentladung ausschlaggebend. Mit einer Nennspannung von 3.7 V pro Zelle ist der LiPo geeignet, um die gewählte MCU auch ohne Spannungsregler zu betreiben und bei Bedarf weitere Spannungsgleiche Akkus parallel zu schalten. Nachteil der Lithium-Polymer-Zelle ist ihre Brennbarkeit bei mechanischer Beschädigung, sowie die instabile Bauform, welche leicht zu eben solchen Beschädigungen führen kann. Bei der Verwendung an Tieren, muss dieses Risiko durch geeignete Schutzmaßnahmen vor mechanischen Einflüssen kompensiert werden. Diese Schutzwirkung wird primär durch die Wahl eines passenden Gehäuses erzielt. Eine sicherere Alternative stellt ein LiFePO4-Akku dar, welcher allerdings wieder eine deutlich niedrigere Energiedichte und lediglich eine Nennspannung von 3.2 V aufweist, wodurch er für den Dauerbetrieb eines 3.3 V-Gerätes ungeeignet ist.

In Abwägung der Formfaktoren der einzelnen Bauteile, Platinen, des Gehäuses und Akkus und deren Zusammenspiel, beschreibt Tabelle 2.3 die grundlegenden Spezifikationen des Lithium-Polymer-Akkus, detaillierter das Datenblatt im Anhang J. Eine Abbildung des gewählten Akkus zeigt Bild 2.7

Tabelle 2.3.: Spezifikationen der Lithium-Polymer-Akkus

Höhe	6 mm
Breite	50 mm
Länge	80 mm
Spannung U_{Nenn}	3.7 V
Kapazität C	3000 mAh
Energie E	11.1 Wh
Stromstärke I_{perm}	0.5 C (= 1.5 A)
Betriebstemperatur	-10°C bis 60°C
Steckverbindung	JST-PH 2.0 mm

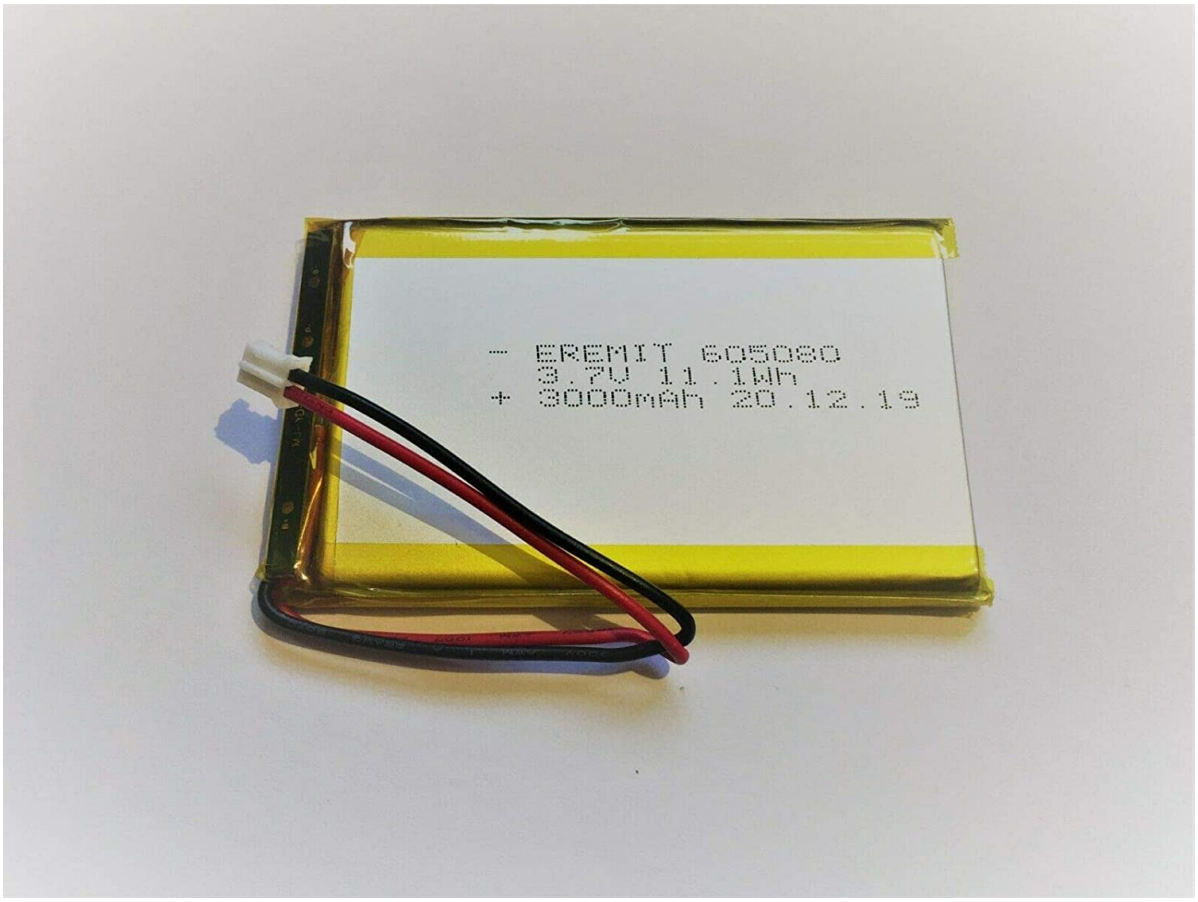


Abbildung 2.7.: Verwendete 3000 mAh Lithium-Polymer-Zelle

Vereinfachte Ermittlung der theoretischen Akkulaufzeit

Die Akkulaufzeit hängt im Wesentlichen vom Stromverbrauch im Mess- und Standby-Betrieb des Mikrocontrollers und GPS-Moduls ab. Darüber hinaus spielen äußere Einflüsse, wie die Umgebungstemperatur oder die Abschattung des GPS-Empfangs eine Rolle, sowie die Selbstentladung der Akkuzelle selbst. Die Selbstentladung bei Lithium-Batterien wird von Khan et al. (2018) vereinfacht mit ca. 5 % durchschnittlichen Kapazitätsverlust pro Monat bei einer Umgebungstemperatur von 20°C angegeben. Dies entspricht auch in etwa dem, im Datenblatt der verwendeten Lithium-Polymer Batterie (Anlage J), angegeben Wert. Bei der Berechnung der Laufzeit, wird entsprechend ein Wert von 5 % pro 30 Tage angenommen.

Der ermittelte Wert stellt jedoch nur eine stark pauschalisierte, erwartbare Laufzeit dar. Geomorphologische und wetterbedingte Abschattungen des GPS-Signals, welche die einzelnen Messzeiten verlängern können, haben einen wesentlichen Einfluss auf die effektive Laufzeit. Die Selbstentladung ist zudem nicht linear und hängt direkt proportional von Umgebungstemperatur ab - sie erhöht sich bei steigenden Temperaturen. Darüber

hinaus besitzen die Akkus eine Schutzschaltung vor Tiefenentladung, das Datenblatt gibt hier eine Abschaltung bei Erreichen von $V_{REL2} = 3.0\ V \pm 0.1\ V$ an. Zwar liegt die Mindestspannung der verbauten Komponenten bei 3.3 V, jedoch ist der genaue Zeitpunkt, bzw. die Spannung, bei der das GPS-Modul oder die MCU tatsächlich nicht mehr funktionieren in der Regel nicht exakt bei diesem Wert, sondern tendenziell niedriger. Dieses Verhalten ist unter anderem bedingt durch die Entfernung des Spannungsreglers. Spätestens bei Erreichen der Schwelle der Schutzschaltung ist eine weitere Spannungsversorgung allerdings ausgeschlossen. Diese Faktoren können im Vorfeld nicht bzw. nicht exakt ermittelt werden, und stellen sich erst im Ergebnis dar.

Die Parameter der Berechnung werden in Tabelle 2.4 dargestellt und stellen die gemessenen Werte der, in Kapitel 2.6.4 beschriebenen, bereits optimierten Hardware dar.

Tabelle 2.4.: Parameter zur vereinfachten Berechnung der theoretischen Akkulaufzeit (* ≙ angenommene Werte)

Kapazität LiPo	C	3000 mAh
Zeit pro Messung Ø *	$t_{measure}$	20 s
Messintervall	t_{int}	300 s
Sekunden pro Tag	t_{day}	86400 s
Messzeit pro Tag Ø *	t_{active}	$\frac{t_{day}}{t_{int}} \cdot t_{measure}$
Standbyzeit pro Tag Ø *	$t_{standby}$	$t_{day} - t_{active}$
Stromstärke Messung Ø	I_{active}	58.0 mA
Stromstärke Standby Ø	$I_{standby}$	0.6 mA

Die durchschnittliche Stromstärke ermittelt sich durch

$$I_{res} = \frac{I_{active} \cdot t_{active} + I_{standby} \cdot t_{standby}}{t_{day}} = 4.247\ mA$$

Womit sich über die Kapazität die maximale Akkulaufzeit in Tagen t_A ermitteln lässt

$$t_A = \frac{C}{I_{res}} \div 24\ h = 29,433\ d$$

Die durch die Selbstentladung reduzierte Anzahl an Tagen ($t_{A_{red}}$), beträgt bei einer Entladerate von 5 % pro 30 Tagen näherungsweise:

$$t_{A_{red}} = t_A * (1 - \frac{t_A}{30\ d} \cdot 0.05) = \underline{\underline{\mathbf{27,989\ d}}} \approx 28\ d$$

Weitere Bauteile

Gehäuse

Bei den Gehäusen handelt es sich um verschraubbare Standard-Gehäuse aus Thermoplast-Kunststoff mit 1.8 mm Wandstärke. Die Außenmaße betragen 72x50x28 mm, wie im Datenblatt in der Anlage K ersichtlich wird. Eine Verbauung der Akkus im Inneren ist bei dieser Gehäusegröße nicht möglich, jedoch decken die Flächen der Ober- und Unterseite die Ausmaße des Akkus weitestgehend ab und können als daher als Montagefläche verwendet werden.

Metal Oxide Semiconductor Field-Effect Transistor (MOSFET)

Zur Reduzierung der Leistungsaufnahme im Ruhemodus, wird das GPS-Modul über eine sogenannte MOSFET-Schaltung abgeschaltet. Diese dient als Alternative zu einem klassischen elektromechanischen Relais, zur Erhöhung der Schaltgeschwindigkeit, Minimierung der Schaltgeräusche und Vermeidung mechanischer Bauteile. Zum Einsatz kommt ein IRF3708 N-Kanal MOSFET, dessen technisches Datenblatt im Anhang I zu finden ist. Durch die angelegte Spannung zwischen Gate- und Source-Anschluss, wird der Stromfluss am Minuspol des am Drain-Pin angeschlossenen GPS-Moduls minimal und damit de facto deaktiviert. Diese Thematik wird in Kapitel 2.6.4 weiter behandelt.

Lochrasterplatine

Zur festen Verbindung der einzelnen Komponenten wird eine 5x7mm-Lochrasterplatine verwendet. Mit Bohrungen an den Ecken, kann diese in ein passendes Gehäuse geschraubt werden.

Hülle

Zur Montage der fertigen Geräte am Halsband der Kühe wurde ein Stück Schlauchmaterial, wie es auch bei Feuerwehren Verwendung findet, präpariert. Diese Lösung orientiert sich an den Hüllen der kommerziellen QTrack-Geräte.

Die Vorteile sind die hohe Stabilität der Schläuche, die einfache Verarbeitung und die Wasserdichtigkeit. Für die Befestigung am Halsband, sowie die Fixierung der Geräte innerhalb des Schlauches wurde ein 12mm langer Flachverbinder aus rostfreiem Edelstahl verwendet. Die Schläuche selbst folgen der „B-Druckschlauch“-Norm und haben damit einen Durchmesser von 75mm. Die Länge der Teilstücke richtet sich an den Längen der GPS-Tracker, plus einem doppelten Überstand in Breite der Flachverbinder, um die Enden jeweils einmal umschlagen zu können. Abbildung 2.8 zeigt die verpackten Geräte.

Abbildung 2.8.: Wasserdichte Hülle für die GPS-Tracker

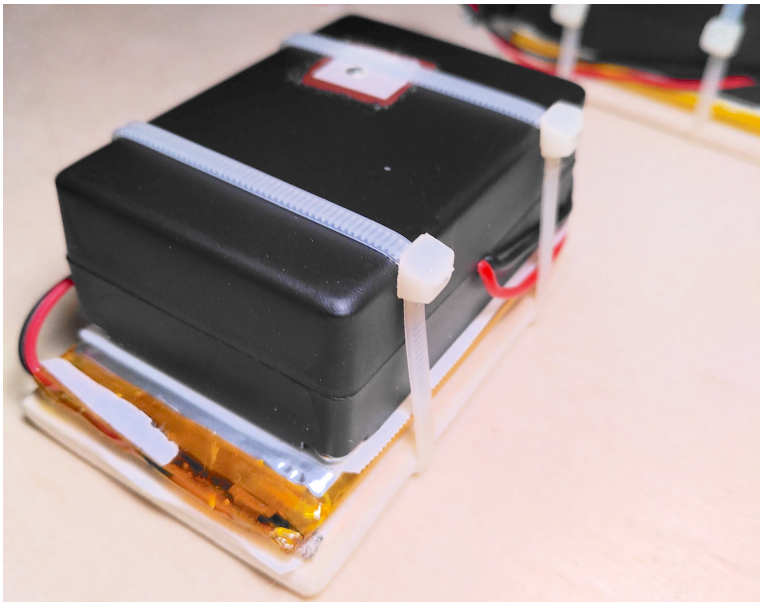


Abbildung 2.9.: Fertig montierter GPS-Tracker der ersten Generation ohne Hülle

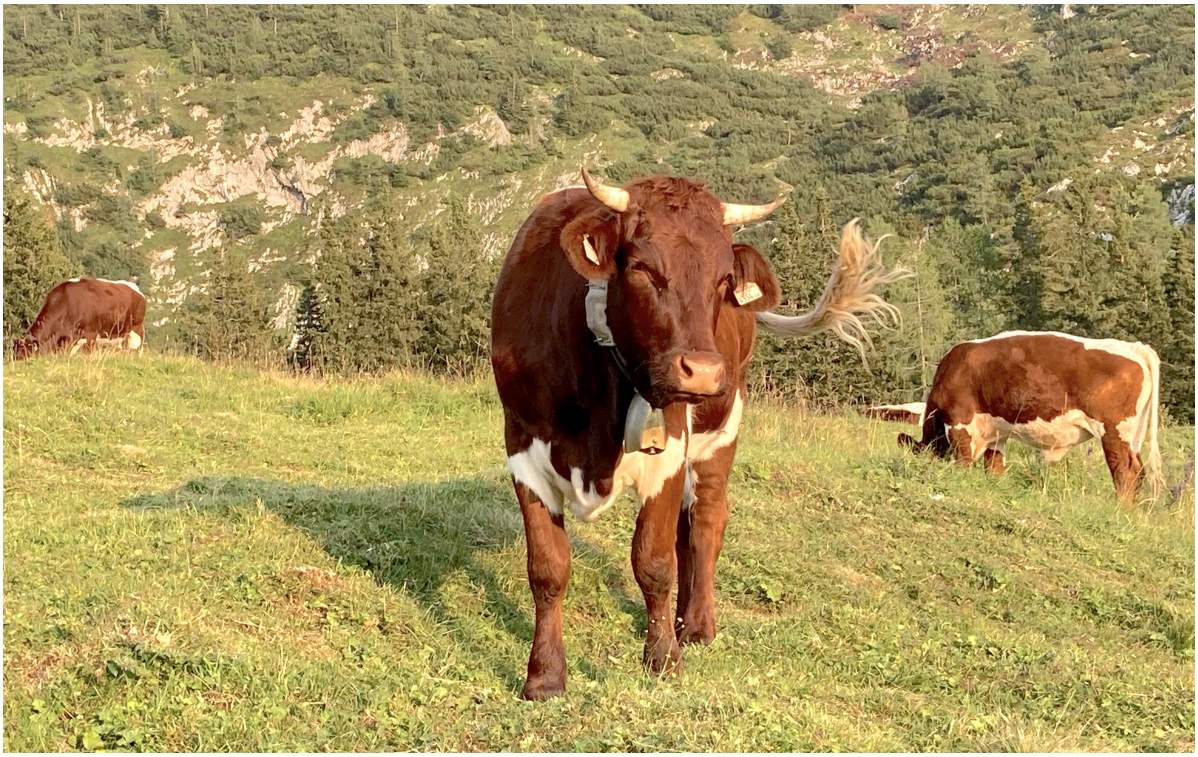


Abbildung 2.10.: Montiertes Gerät am Halsband der Kuh „Pia“ im Almgebiet

2.6.2. Schaltplan

Die Verdrahtung der einzelnen Bauteile des GPS-Datenloggers zeigt der Schaltplan in Abbildung 2.11 und Anhang F.

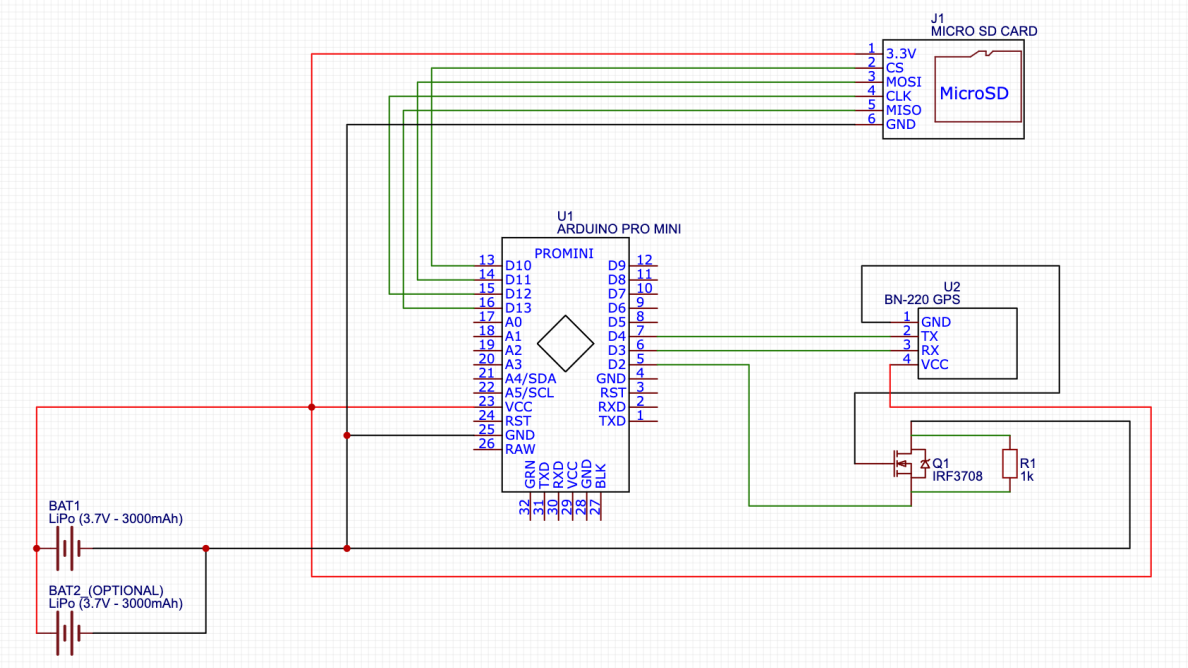


Abbildung 2.11.: Schematische Darstellung der Verdrahtung des GPS-Datenloggers

2.6.3. Programmierung

Die Programmierung fügt die Funktionalitäten der einzelnen Hardware-Komponenten zusammen und integriert dabei die bereits gewonnen Recherche-Ergebnisse. Umgesetzt wurde die Code-Entwicklung mit den in Kapitel 2.2.2 erwähnten Tools. Neben der Umsetzung der Kernaufgaben des GPS-Datenloggers, also Ermitteln und Speichern der Positionsdaten in einem bestimmten Intervall, stellt die Software-Entwicklung bei Mikrocontrollern weitere Herausforderung bereit. Die Begrenzung der verfügbaren Systemressourcen (CPU, RAM, Speicher), niedriger Stromverbrauch und ein, durch den Compiler bedingten eingeschränkten Syntax-Umfang fordern eine hocheffiziente, ressourcenschonende Programmierung, wobei die Methoden zum Debugging des Codes ebenfalls stark eingeschränkt sind, verglichen zur klassischen Web-, App- oder Desktopprogrammierung.

Der Quellcode ist mehrere Bereiche aufgeteilt, welche in den folgenden Sektionen dargestellt sind.

Nicht explizit in diesem Kapitel aufgeführt werden Helfer-Funktionen für Debug- oder Logging-Zwecke. Der gesamte Quellcode ist in Anhang B.1 hinterlegt.

Bibliotheken

Für die Ansteuerung der Bauteile und die Optimierung des Standby-Verhaltens wurden externen Bibliotheken verwendet, welche Teils von der Arduino-Community, teils von individuellen Entwicklern bereitgestellt werden. In vielen Fällen kommen für bestimmte Aufgaben verschiedene Bibliotheken in Frage - beispielsweise zum Parsen des National Marine Electronics Association (NMEA)-Protokolls. Die Kriterien zur Wahl einer bestimmten Library wurden definiert durch ihre Kompaktheit, einfache Bedienung und Erfahrungsberichte von Usern (z.B. Bewertung bei Github).

Es wurden, wie Quellcode 2.5 zeigt, folgende Libraries verwendet:

EEPROM Zur Ansprache des Nicht-Flüchtigen-Speichers

SoftwareSerial UART-Verbindung zu GPS-Modul und Debug-Konsole

TinyGPS++ Parsen der empfangenen NMEA-Sätze[3]

SPI SPI-Verbindung zum SD-Modul

SD Ansprache der SD-Karte via SPI

Sleep_n0m1 Energieoptimiertes Handling für Low-Power-Modi[4]

Quellcode 2.5: Einbindung der externen Bibliotheken

```
#include <EEPROM.h>

#include <SoftwareSerial.h>
#include <TinyGPS++.h>

#include <SPI.h>
#include <SD.h>

#include <Sleep_n0m1.h>
```

Globale Variablen und Konstanten

Die globalen Variablen und Konstanten am Anfang des Quellcodes bestimmen feste Parameter, wie PIN-Belegungen, global benötigte Objekte, welche in den beiden Systemfunktionen Verwendung finden, sowie Debug-Parameter zum schnellen Aktivieren erweiterter Ausgaben. Die Kommentare in Quellcode 2.6 beschreiben die jeweiligen Parameter.

[3] TinyGPS++ Dokumentation: `http://arduiniana.org/libraries/tinygpsplus/`
[4] Sleep_n0m1 Dokumentation: `https://github.com/n0m1/Sleep_n0m1`

Quellcode 2.6: Definition und Deklarierung von globalen Variablen

```
bool DEBUG = true; //use Debug-Build for writing console messages
int INTERVAL = 300; //interval for data collection in seconds

// LogTargets
const int LOG_SERIAL = 1; //enum for writing logs to console
const int LOG_SDCARD = 2; //enum for writing logs to file on sd card

// GPS
const int IO_GPS_RX = 4; //rx (receiver) pin number for UART connection
const int IO_GPS_TX = 3; //tx (transmitter) pin number for UART connection
const int IO_GPS_SWITCH = 2; //MOSFET switch pin number

SoftwareSerial gpsSerial(IO_GPS_RX, IO_GPS_TX); // register rx,tx for GPS
TinyGPSPlus gps; //init gps object

//UBX message for power save mode
uint8_t PSM[] = {0xB5, 0x62, 0x06, 0x11, 0x02, 0x00, 0x08, 0x01, 0x22, 0x92
    };

//UBX message for full off/on gps module - twice as documentation is not
    clear
uint8_t GPSFulloff[] = {0xB5, 0x62, 0x02, 0x41, 0x08, 0x00, 0x00, 0x00, 0x00
    , 0x00, 0x02, 0x00, 0x00, 0x00, 0x4D, 0x3B};
uint8_t GPSFullon[] = {0xB5, 0x62, 0x02, 0x41, 0x08, 0x00, 0x00, 0x00, 0x00,
     0x00, 0x01, 0x00, 0x00, 0x00, 0x4C, 0x37};
uint8_t GPSoff[] = {0xB5, 0x62, 0x06, 0x04, 0x04, 0x00, 0x00, 0x00, 0x08, 0
    x00, 0x16, 0x74};
uint8_t GPSon[] = {0xB5, 0x62, 0x06, 0x04, 0x04, 0x00, 0x00, 0x00, 0x09, 0
    x00, 0x17, 0x76};


//base sleep time - should be shorter in debug mode
const unsigned long sleepTimeBase = DEBUG ? 30000 : 300000;
Sleep sleep;

uint32_t GPSendtimemS; // time in milliseconds when GPS got fixed position
uint32_t GPSstarttimemS; // time in milliseconds GPS start to find fix

// initialize SD card
const int IO_SDCARD = 10; //IO pin for SD module
char FILENAME[] = "gpsLog.txt"; //filename to store gps data
bool withSDCard = true; //mode for using without SD (e.g. in debug)
```

Initialisierung

Die Funktion `setup` ist eine systemseitig benötigte Funktion. Sie wird nur einmalig beim Systemstart aufgerufen und dient für einmalige Operationen und Initialisierungen.

Beim Datenlogger wird die SD-Karte initialisiert und bei Bedarf die resultierenden Textdateien inklusive Kopfzeile angelegt. Im Anschluss startet die erste Signalsuche des GPS-Moduls, unabhängig vom Messintervall, solange bis die erste Position (Fix) erfolgreich ermittelt wurde.

Der Quellcode 2.7 beschreibt die einzelnen Schritte der Initialisierung.

Quellcode 2.7: Einmalige Initialisierung in der setup-Funktion

```
void setup()
{
  if (DEBUG)
  {
    Serial.begin(115200); //start console output with BAUD-Rate
    logger("DEBUG-MODE: Init", LOG_SERIAL); //logger function to define log
        output target
  }

  // init SDCard
  if (withSDCard && SD.begin(IO_SDCARD))
  {
    logger("Init SDCard: CardReader found", LOG_SERIAL);

    // write header if new file is needed
    if (!SD.exists(FILENAME))
    {
      logger("Sat;HDOP;Lat;Lon;Fix;Date;Time;Age;Alt;Course;Speed;Card",
          LOG_SDCARD);
    }
  }
  else
  {
    logger("Init SDCard: Failed - withSDCard -> " + String(withSDCard),
        LOG_SERIAL);
    withSDCard = false;
  }

  // init GPS
  logger("Init GPS: Started... with " + String(gps.libraryVersion()),
      LOG_SERIAL);
  digitalWrite(IO_GPS_SWITCH, HIGH); //set MOSFET switch to "on"
  gpsSerial.begin(9600); //listen to UART
```

```
  UBLOX_GPS_Wakeup(); //send ubx wakeup message

  //try to retrieve first valid fix
  //true = try as long as it takes on init
  getValidSignal(true);
}
```

Runtime

Die `loop`-Funktion ist ebenfalls eine systemseitige Funktion, welche aber dauerhaft während der Laufzeit iterativ durchlaufen wird. Damit werden innerhalb dieser Funktion die eigentlichen Aufgaben des GPS-Datenloggers definiert.

Wie in Quellcode 2.8 dargestellt, ist der Ablauf pro Iteration:

(1) Aktivieren des GPS-Moduls

(2) Empfang eines validen Signals

(3) Deaktivierung des GPS-Moduls

(4) Auswertung des letzten Signals

(5) Schreib-Operation der Daten auf SD-Karte

(6) Berechnung der Shutdown-/Standby Zeit

(7) MCU für eine bestimmte Zeit in Standby-Modus versetzen

(8) Automatische Reaktivierung nach Ablauf der Standby-Zeit

Quellcode 2.8: Iterative Funktion zum Datenlogging während der Laufzeit

```
void loop()
{
  //count time to first fix in milliseconds
  //will be used to reduce sleep-time, to ensure fix+sleep=interval
  unsigned long timeToFix = 0;

  logger("WakeUpGPS", LOG_SERIAL);
  digitalWrite(IO_GPS_SWITCH, HIGH); //switch on GPS
  UBLOX_GPS_Wakeup(); //ensure it is awaken

  logger("GetValidSignal", LOG_SERIAL);
  timeToFix = getValidSignal(false); //get valid signal, with an maximum of
      tries
```

```
UBLOX_GPS_Shutdown(); //set power safe mode to gps instant after valid fix

String dataLine = ""; //string builder for resulting dataset of last fix

// Satellites
dataLine = gps.satellites.isValid() ? String(gps.satellites.value()) : "";
dataLine += ";";
logInline(dataLine); //log instant to SD card - without new line

// HDOP
dataLine = gps.hdop.isValid() ? String(gps.hdop.hdop()) : "";
dataLine += ";";
logInline(dataLine);

// Lat/Lon
dataLine = gps.location.isValid() ? String(gps.location.lat(), 8) : "";
dataLine += ";";
logInline(dataLine);

dataLine = gps.location.isValid() ? String(gps.location.lng(), 8) : "";
dataLine += ";";
logInline(dataLine);

// Fix age
dataLine = gps.location.isValid() ? String(gps.location.age()) : "";
dataLine += ";";
logInline(dataLine);

// Date
TinyGPSDate d = gps.date;
dataLine = "";
if (d.isValid())
{
  dataLine += String(d.year()) + "-";

  dataLine += d.month() < 10 ? "0" : "";
  dataLine += String(d.month()) + "-";

  dataLine += d.day() < 10 ? "0" : "";
  dataLine += String(d.day());
}
dataLine += ";";
logInline(dataLine);
```

```
// Time
TinyGPSTime t = gps.time;
dataLine = "";
if (t.isValid())
{
  dataLine += t.hour() < 10 ? "0" : "";
  dataLine += String(t.hour()) + ":";

  dataLine += t.minute() < 10 ? "0" : "";
  dataLine += String(t.minute()) + ":";

  dataLine += t.second() < 10 ? "0" : "";
  dataLine += String(t.second());
}
dataLine += ";";
logInline(dataLine);

// Age
dataLine = d.isValid() ? String(d.age()) : "";
dataLine += ";";
logInline(dataLine);

// Altitude
dataLine = gps.altitude.isValid() ? String(gps.altitude.meters()) : "";
dataLine += ";";
logInline(dataLine);

// Course
dataLine = gps.course.isValid() ? String(gps.course.deg()) : "";
dataLine += ";";
logInline(dataLine);

// Speed
dataLine = gps.speed.isValid() ? String(gps.speed.kmph()) : "";
dataLine += ";";
logInline(dataLine);

// Card
dataLine = gps.course.isValid() ? String(TinyGPSPlus::cardinal(gps.course.
    deg())) : "";
logInline(dataLine);

logger("", LOG_SDCARD); //dataset finished - add new line to log file
```

```
  digitalWrite(IO_GPS_SWITCH, LOW); //turn of GPS completely
  logger("ShutDownMCU", LOG_SERIAL);
  sleep.pwrDownMode(); //set sleep mode

  //calculates time to sleep to ensure having right interval
  uint32_t currentTime = gps.time.minute() * 60 + gps.time.second();
  uint32_t secondsToStart = INTERVAL - (currentTime % INTERVAL);
  Serial.println(currentTime);
  Serial.println(secondsToStart);
  Serial.println(secondsToStart * 1000 - timeToFix);

  //activate sleep mode for given time
  sleep.sleepDelay(secondsToStart * 1000 - timeToFix);

  //MCU awakes automatically
  logger("WakeUpMCU", LOG_SERIAL);
}
```

Funktion zur Positionsbestimmung

Sowohl in der `setup`-, als auch in der `loop`-Funktion wird die `getValidSignal`-Funktion parametrisiert aufgerufen. Diese Funktion wurde selbst definiert und stellt die Hauptfunktion zur Ermittlung der Daten dar. Der übergebene Boolean-Parameter legt fest, ob der Versuch ein valides Signal zu ermitteln nach einer bestimmten Zeit abgebrochen werden soll, z.B. da aufgrund ungünstiger Witterungsverhältnisse, Abschattungen oder Satellitenpositionen der TTFF sehr lange dauert und damit die Akkukapazität stark belasten würde. Als `return`-Parameter wird der TTFF-Wert in Millisekunden als Integer zurückgegeben.

Die GPS-Datenermittlung wird innerhalb ein `while`-Schleife abgehandelt. Bei jeder Iteration wird die GPS-Library nach neuen verarbeiteten Daten befragt. Sofern ein neues valides Signal zur Verfügung steht, bleibt dies im `gps`-Objekt global erhalten. Eine Zählervariable wird um 1 erhöht - bei insgesamt 10 validen Signalen wird die Funktion verlassen, um das letzte Signal auf der SD-Karte zu loggen. Beobachtungen haben ergeben, dass mit zunehmender Anzahl an validen Signalen innerhalb einer Messreihe die ermittelte Streuung (DOP-Wert) niedriger wird. Dadurch wird mit der Nutzung späterer Fixes, die Positionsgenauigkeit potenziell erhöht. Der Abbruch der Messreihe nach 10 erfolgreichen Messungen stellt einen Mittelweg zwischen Genauigkeit und Stromverbrauch dar.

Sofern keine 10 validen Signale, aber mindestens eines innerhalb einer bestimmten Zeit ermittelt werden konnte, wird dieses geloggt. Diese Abbruch-Bedingung der Messreihe hat ebenfalls energietechnische Gründe. Hierbei wurde eine maximale Zeit zwischen TTFF und dem nächsten validen Signal von maximal 15 Sekunden festgelegt.

Wurde jedoch kein TTFF innerhalb einer adäquaten Zeit, also innerhalb des Messintervalls ermittelt, wird der Messversuch mit einem „Timeout“ abgebrochen. Einzige Ausnahme davon ist die Nutzung des booleschen Wertes `forever=true` während der Initialisierung, aufgerufen in der `setup`-Funktion, um dem GPS-Modul - insbesondere in Kaltstart-Situationen - genügend Zeit einzuräumen, ein valides Signal zu schreiben.

Quellcode 2.9: Funktion zur Ermittlung eines validen Signals

```
unsigned long getValidSignal(bool forever)
{
  GPSstarttimemS = millis(); //current mcu runtime in milliseconds
  unsigned int countValids = 0; //counter for measuring valid signals
  unsigned long firstValidTime = 0; //TTFF
  uint8_t GPSchar;
  while (1)
  {
    if (gpsSerial.available() > 0)
    {
      GPSchar = gpsSerial.read();
      gps.encode(GPSchar); //parse NMEA
    }
    if (gps.location.isUpdated() && gps.altitude.isUpdated() && gps.location.
        isValid())
    {
      if (firstValidTime == 0)
      {
        firstValidTime = millis();
      }
      countValids++;
      GPSendtimemS = millis();
      logger(String(gps.location.lat(), 6), LOG_SERIAL);
      logger(String(gps.location.lng(), 6), LOG_SERIAL);
      if (countValids >= 10)
      { // try to have ten signals before logging
        logger("10 valid fixes", LOG_SERIAL);
        break;
      }
    }
    // fallback if we dont have further valid signals within given seconds
    if (countValids >= 1 && millis() - firstValidTime >= 15000)
    {
      logger("return after 15s to TTFF", LOG_SERIAL);
      break;
    }
```

```
    //handle timeout
    if (millis() - GPSstarttimemS >= sleepTimeBase && !forever)
    {
      logger("Timeout", LOG_SERIAL);
      return 0;
    }
  }
  logger("FoundSignal", LOG_SERIAL);
  logger(String(GPSendtimemS - GPSstarttimemS) + "ms", LOG_SERIAL);
  return GPSendtimemS - GPSstarttimemS;
}
```

2.6.4. Hardware-Anpassungen

Nach Zusammenführung von Hard- und Softwarekomponenten konnten erste oberflächliche Tests hinsichtlich der Qualität der Datenaufnahme, Laufzeit der Akkus, Ausfallsicherheit und Layout durchgeführt werden. Die Teilergebnisse flossen wiederum in die Optimierung der Hard- und Software ein.

In erster Linie wurde das Hauptkriterium „unabhängige Stromversorgung“ näher untersucht. Bei Stromfluss-Messungen eines Prototyps fiel insbesondere ein vergleichsweise hoher Ruhestrom von >7 mA auf, welcher auf mehrere, folgend beschriebene Ursachen zurückzuführen ist. Als Ruhestrom wird die Stromstärke zwischen den eigentlichen Messintervallen bezeichnet, in denen das GPS-Modul und die MCU in den Standby-Modus versetzt werden.

MOSFET-Schaltung

Eine isolierte Strom-Messung, begrenzt auf das GPS-Modul, ergab einen Wert von 6.21 mA bei aktivierten Standby-Mode. Dieser Modus wird aktiviert durch die Übermittlung einer Nachricht über die UART-Schnittstelle im UBX-Protokoll. Die Hexadezimal-Darstellung dieser Nachricht lautet `B5 62 02 41 08 00 00 00 00 00 02 00 00 00 4D 3 B` und kann der u-blox (2023) Protokoll-Spezifikation entnommen werden.

Bei Anwendung der Strom-Berechnung aus Kapitel 2.6.1 auf die Messwerte des GPS-Bausteins (46 mA bei Messung, 6.21 mA im Standby), wäre der 3000 mAh - Akku bereits nach rund 12 Tagen aufgebraucht, ohne dabei die MCU zu berücksichtigen. Entsprechend ist bei größer werdenden Messintervallen und damit verbundenen, längeren Standby-Zeiten eine vollständige Abschaltung des GPS-Moduls sinnvoll. Hierzu wurde eine MOSFET-Schaltung gewählt, welche bereits in Kapitel 2.6.1 erwähnt wurde und im Schaltplan 2.11 technisch beschrieben ist. Durch die schaltbare Stromzufuhr zum Transistor wird das GPS-Modul ausreichend spannungsfrei, dass der Baustein deaktiviert wird.

Entfernung des Spannungsreglers

Trotz der signifikanten Verbesserung durch die Abschaltung des GPS, verbleibt ein optimierbarer Ruhestrom, welcher dem Mikrocontroller zuzuordnen ist. Recherchen ergaben, dass hierfür ein, bei niedrigen Strömen ineffizienter Spannungsregler eine Ursache sein kann. Empfohlen wird, u. a. im DIY-Projekt von Klements (2021), eine Umrüstung auf sogenannte „Low quiescent current - LDO“, also ein Spannungsregler mit sehr geringem Ruhestrom.

Durch Betrachtung der technischen Daten aller Einzelkomponenten (GPS-Modul, MCU, SD-Kartenleser, Akku) ist jedoch ein Spannungsregler in diesem Projekt nicht zwingend erforderlich. Die maximale Ausgangsspannung des LiPo wird angegeben mit 4.2 V und liegt dadurch unterhalb der maximalen Eingangsspannungen der Bauteile, aber oberhalb der minimalen Betriebsspannung. Entsprechend wurde der Spannungsregler des Mikrocontrollers ersatzlos entfernt, wie Abbildung 2.12 verdeutlicht.

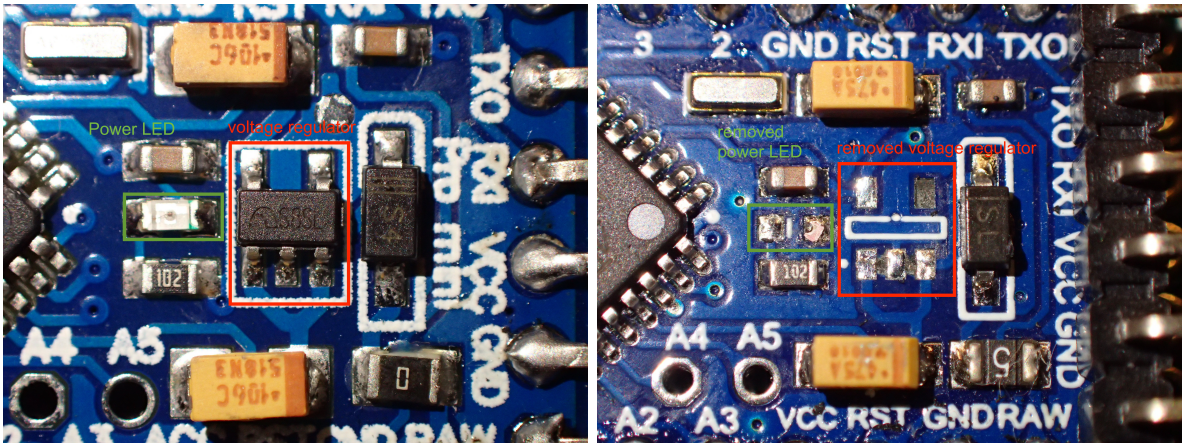


Abbildung 2.12.: Ausschnitt des Mikrocontrollers vor und nach den Hardware-Anpassungen

Entfernung der Power-LED

Eine weitere Empfehlung von Klements (2021) wird der Entfernung der Power - Leuchtdiode (LED) zuteil. Diese leuchtet bei Stromversorgung des Mikrocontrollers dauerhaft und zeigt lediglich dessen Betrieb an. Auf diese Funktion kann bei diesem Projekt verzichtet werden, da einerseits eine Auswertung des Betriebszustands, bei dem am Tier montierten Datenlogger nicht mehr möglich ist, zum anderen wird der erfolgreiche Betriebsstart durch weitere LEDs am GPS-Modul angezeigt. Die Leuchtdioden am GPS-Modul leuchten jedoch nicht dauerhaft, sondern blinken nur während des Signalempfangs. Diese kurzen Blinkphasen, werden zunächst aus energietechnischer Sicht nicht berücksichtigt, zumal die Blinksequenzen eine Semantik innehaben. Abbildung 2.12 zeigt den Zustand der Hauptplatine vor und nach Entfernung der LED.

2.6.5. GPS-Logger der zweiten Generation

Ableitend aus den ersten Ergebnissen des Almsommers 2021 aus Kapitel 3.2 und deren Auswertungen in Kapitel 4.1, wurden einige Optimierungen für eine zweite Version der GPS-Tracker, welche zusätzlich zur ersten Generation, im Almsommer 2022 Verwendung fanden, entwickelt und umgesetzt. Folgende Punkte beschreiben die überarbeiteten Komponenten.

Anpassungen der Hardware

Die Elektronik und deren Stromverbrauchs-Optimierungen blieb für die zweite Generation identisch; die Bestückung der Platine hat sich leicht verändert, um eine niedrigere Bauhöhe zu erreichen. Primär wurden hierfür Abstandshalter aus Kunststoff, sowie überstehende Pins entfernt und der Reset-Knopf der MCU entlötet. Dies war nötig, um mehr Sicherheit für die Akkus und Elektronik innerhalb neuer Gehäuse zu schaffen. Zudem wurde die Akku-Kapazität auf gesamt 6000 mAh verdoppelt, indem je zwei Zellen in Reihe geschaltet wurden. Schematisch ist diese Reihenschaltung als „Optional“ im Schaltplan 2.11 bereits gekennzeichnet.

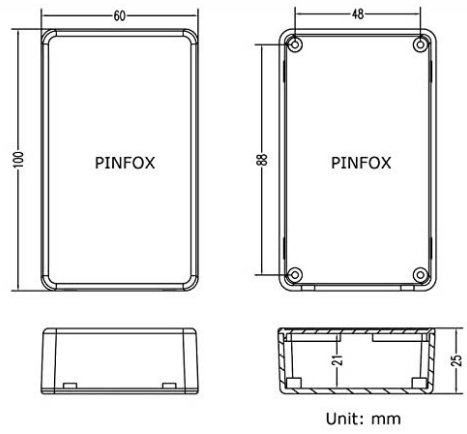


Abbildung 2.13.:
Technische Zeichnung der Gehäuse der zweiten Generation

Die neuen Gehäuse haben eine größere Grundfläche, womit die Akkus im Inneren platziert werden können. Es handelt sich hierbei um Standard-Gehäuse im Elektronikbereich aus ABS-Kunststoff, mit den Abmessungen aus Abbildungen 2.13. Der Reset-Button wurde einerseits zur Reduzierung der Bauhöhe selbst entfernt, zum anderen stellt er zeitgleich das höchste Bauteil auf der Oberfläche des Arduino Pro Mini dar. Um zu verhindern, dass dieser Button versehentlich während der Messperiode durch äußere Einflüsse betätigt wird, wurde dieser entfernt.

Aufgrund eines Lieferfehlers wurden statt 3.3 V mit 8 MHz, Mikrocontroller mit 5 V und 16 MHz geliefert. Aus Zeit- und Verfügbarkeitsgründen konnten keine korrekten Geräte mehr bezogen werden, zumal die MCUs optisch nahezu identisch sind und der Fehler zunächst unbemerkt blieb. Die höhere benötigte Betriebsspannung von 5 V wirkte sich in Tests nicht aus - die Geräte konnten bei entferntem Spannungsregler trotzdem mit den LiPo-Akkus betrieben werden. Jedoch schien die verdoppelte Taktfrequenz unvorhergesehene Auswirkungen auf die Ausführung des Programmcodes zu haben. Dies wirkte sich, trotz Kompilierung des Quellcodes im 16 MHz-Modus, in halbierten Standby- und Messzeiten, sowie kryptischen Log-Ausgaben aus. Gelöst wurde dieses Problem durch eine Umstellung des Bootloaders der MCU. Das führt dazu, dass nicht mehr der exter-

ne 16 MHz-Quarz für die Taktung verwendet wird, sondern ein interner, veränderbarer Taktgeber (internal clock) dies übernimmt. So wurde wiederum eine Taktung von 8 MHz erreicht, welche auch das Programm lauffähig machte. Tiefere Tests dieses Workarounds konnten aus Zeitmangel nicht realisiert werden.

Softwareanpassungen

Die Software wurde stärker angepasst, um die Schwächen der ersten GPS-Tracker auszugleichen. Die umgesetzten Änderungen kommen einem Rewrite der Firmware gleich, wobei die verwendeten Bibliotheken unverändert blieben. Hauptziel der Anpassungen waren eine bessere Lesbarkeit des Codes durch einen objektorientierten Ansatz, eine Optimierung der Auswahl der genausten Position innerhalb einer Messreihe, sowie die zeitliche Synchronisierung der Datenaufnahme der Geräte untereinander. Gleichzeitig sollte das Ziel sein, optional die rohen NMEA-Daten in eine weitere Datei zu speichern, um später bei Bedarf auf alle erfassten zugreifen zu können. Eine, auf den Zweck der Datenerfassung im Almsommer abgestimmte Prüfung, sollte erwirken, dass die Geräte vor Erreichen des Monats Juli lediglich einmal pro Tag aus dem Standby-Modus aufwachen, um aus den GPS-Daten das aktuelle Datum zu bestimmen. Dies sollte den Stromverbrauch reduzieren, zu Zeiten, bei denen die Geräte bereits an den Tieren montiert sind, aber diese nicht mit Sicherheit auf der Alm sind. Der vollständige resultierende Quellcode ist im Anhang C.1 hinterlegt.

Zur Erhöhung der Positionsgenauigkeit wurde die standardmäßig betrachtete Anzahl der validen Fixes verdoppelt auf 20. Fixes werden dabei nur als valide gewertet mit einem Horizontal Dilution of Precision (HDOP)-Wert < 10. Im Unterschied zur Generation 1, wird anschließend nicht das letzte GPS-Signal zurückgegeben, sondern jenes mit dem niedrigsten HDOP-Wert der Messreihe. Diese Änderungen wurden in den Codezeilen der `getValidSignal`-Funktion realisiert.

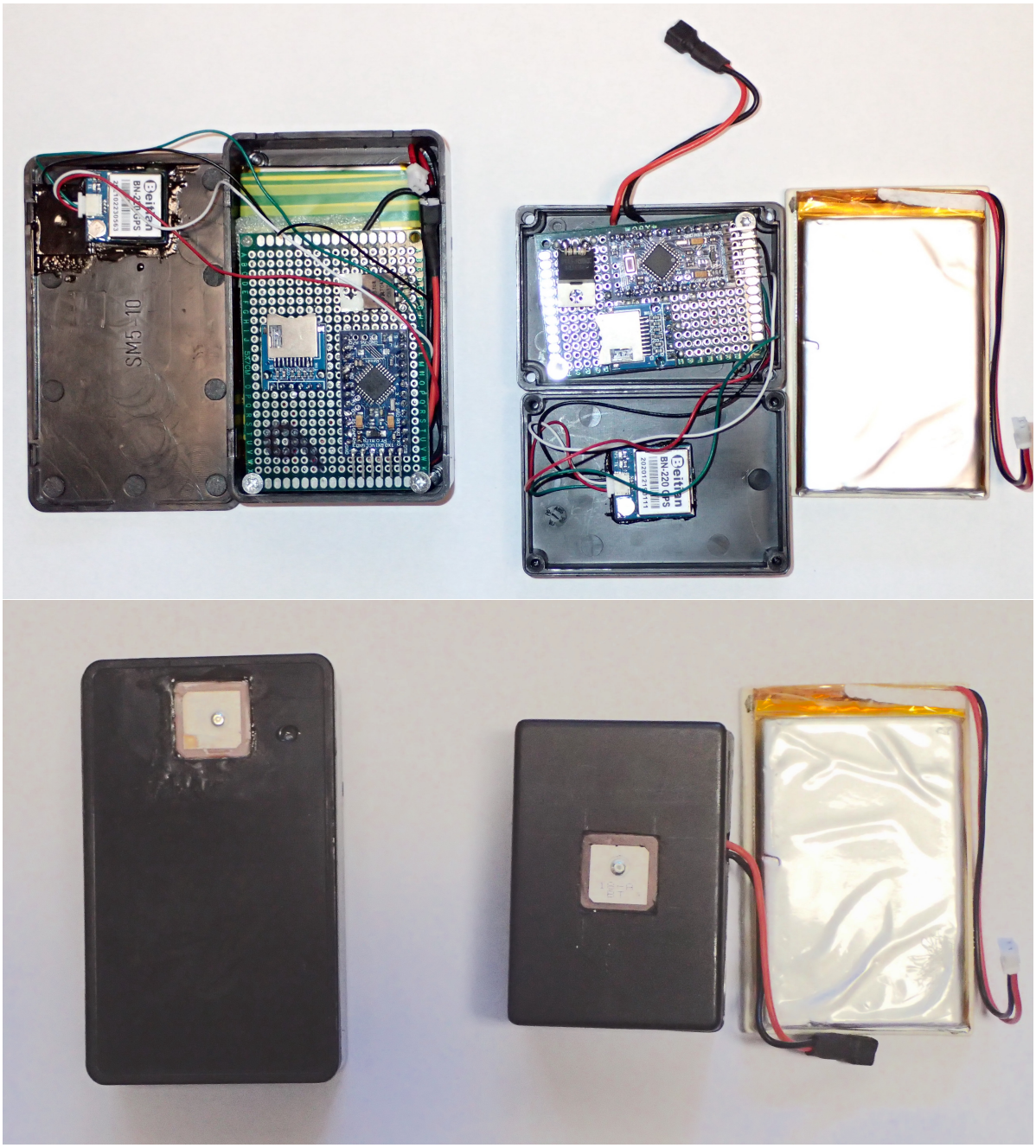


Abbildung 2.14.: Links: GPS-Tracker der 2. Generation
Rechts: GPS-Tracker der 1. Generation

3. Ergebnisse

Bei den nachfolgenden Ergebnissen werden neben den erfassten Wetter- und Positionsdaten der beiden Almsommer, auch die Zwischenauswertungen der Hard- und Softwareentwicklung aufgeführt.

3.1. Entwicklungsergebnisse

Bevor die Geräte an den Tieren montiert wurden, konnten bereits Teilergebnisse aus dem agilen Entwicklungsprozess erhoben werden. Diese beziehen sich primär auf die Messungen des Stromverbrauchs vor und nach den MCU-Optimierungen.

3.1.1. Energieverbrauch

Durch die Optimierungen der Hardware und Firmware, konnten messbare, signifikante Reduzierungen des Ruhestroms erreicht werden. Im Quellcode ist die Verwendung der `sleep_n0m1`-Bibliothek hervorzuheben, welche das klassische Handling von Ruhezeiten mittels der `delay`-Funktion, ersetzt. In Tabelle 3.1 werden die gemessenen Stromstärken der Modi, welche die Library zur Verfügung stellt, dargestellt. Der gewählte „PowerDown-Mode“ konnte dabei, bei optimierter Hardware, die Stromstärke um 3.07 mA reduzieren.

Ebenfalls in Tabelle 3.1 werden die Auswirkungen der Board-Anpassungen beschrieben. Durch die Entfernung des Spannungsreglers wird insbesondere der Stromfluss im Betriebsmodus `sleep_n0m1`-Library deutlich reduziert. Dies wirkt sich positiv auf die Messzeiten aus. Die Entfernung der Power-LED zeigt über alle Betriebsmodi hinweg eine Reduzierung des Stroms um ca. 1.3 mA.

Werden die Stromflüsse des GPS- und SD-Moduls hinzugerechnet, resultieren die in der Akkulaufzeit-Berechnung verwendet Werte von 58.0 mA im Mess- und 0.6 mA im Ruhemodus.

Tabelle 3.1.: Strommessungen der Hard- und Softwareoptimierungen

Betriebsmodus	Original [mA]	ohne U-Regler [mA]	ohne PwrLED [mA]
OnBoard-LED an	8.83	5.56	4.34
Ruhestrom	6.90	4.35	3.08
Idle-Mode	3.31	2.3	0.98
PowerSafe-Mode	1.96	1.67	0.31
Standby-Mode	1.59	1.53	0.18
PowerDown-Mode	1.28	1.39	0.01

3.2. GPS-Logger Generation 1: Almsommer 2021

Von den fünf GPS-Datenloggern der ersten Generation konnte im Almsommer 2021 unwetterbedingt nur eines vom Landwirt an den Rindern montiert werden. Die weiteren vier Geräte verblieben im KFZ des Landwirts und erfuhren daher identische äußere Einflüsse und das gleiche Bewegungsprofil. Dies bot den Vorteil, dass sich die Geräte `GPS02, GPS03, GPS04, GPS05` gut für einen direkten Qualitäts- und Quantitätsvergleich eigneten. Das montierte Gerät `GPS01`, lässt sich hinsichtlich der Datenaufnahme mit dem kommerziellen Gerät `Q2_24906` in Verbindung bringen, da beide an der Kuh „Pia“ befestigt waren.

Die Dauer des Almsommers betrug 77 Tage mit dem Almauftrieb am 26.06.2021 und dem Almabtrieb am 11.09.2021. Die Geräte wurden drei Tage zuvor, am 23.06.2021 in Betrieb genommen.

3.2.1. Positionsdaten

Die Sichtung der Rohdaten zeigte teilweise Daten vom 21.06.2021, welche jedoch nur zu Testzwecken aufgenommen wurden. Die Geräte wurden nach kurzer Zeit wieder deaktiviert. Die Auswertungen der Werte klammern diesen Tag aus und beginnen mit der offiziellen Inbetriebnahme vom 23.06.2021.

Tabelle 3.2 zeigt, dass die in Kapitel 2.6.1 berechnete Laufzeit von 28 Tagen bei `GPS01` knapp überschritten, bei den restlichen Geräten im Schnitt etwas unterschritten wurde.

Tabelle 3.2.: Anzahl der Messpunkte N_{21} und Messzeiten im Almsommer 2021 pro GPS-Gerät

Gerät	**Kuh**	N_{21}	**Startzeit** (MESZ)	**Endzeit** (MESZ)	**Messtage** Δt_{21}
GPS01	Pia	11292	23.06.21 15:33:33	02.08.21 07:03:23	29,65
GPS02	-	6632	23.06.21 15:33:41	16.07.21 21:12:44	23,24
GPS03	-	6981	23.06.21 15:37:39	18.07.21 02:38:50	24,46
GPS04	-	3533	23.06.21 15:36:34	06.07.21 05:23:08	12,57
GPS05	-	6992	23.06.21 15:31:40	18.07.21 03:23:59	24,54

Die aufgezeichneten HDOP-Werte lassen einen indirekten Rückschluss auf die Qualität bzw. Positionsgenauigkeit der Daten zu. Eine grafische Auswertung der Verteilung der Streuungswerte zeigen die Abbildungen 3.1, 3.2 und 3.3. Auffällig sind dabei die Werte mit 99.99, welche Mehrfach bei jedem GPS-Gerät aufgezeichnet wurden. Ansonsten scheint die Verteilung der Streuungswerte bei allen Geräten ähnliche gute Ergebnisse erzielen zu können. In Tabelle 3.3 sind grundlegende statistischen Daten der Verteilung dargestellt, sowie die Anzahl der zu hohen HDOP-Werte n_{99}. Der Median $\tilde{x}'$ wurden zusätzlich nochmals berechnet, unter Ausklammerung der Datensätze mit 99.99-Werte, um deren Einfluss zu prüfen.

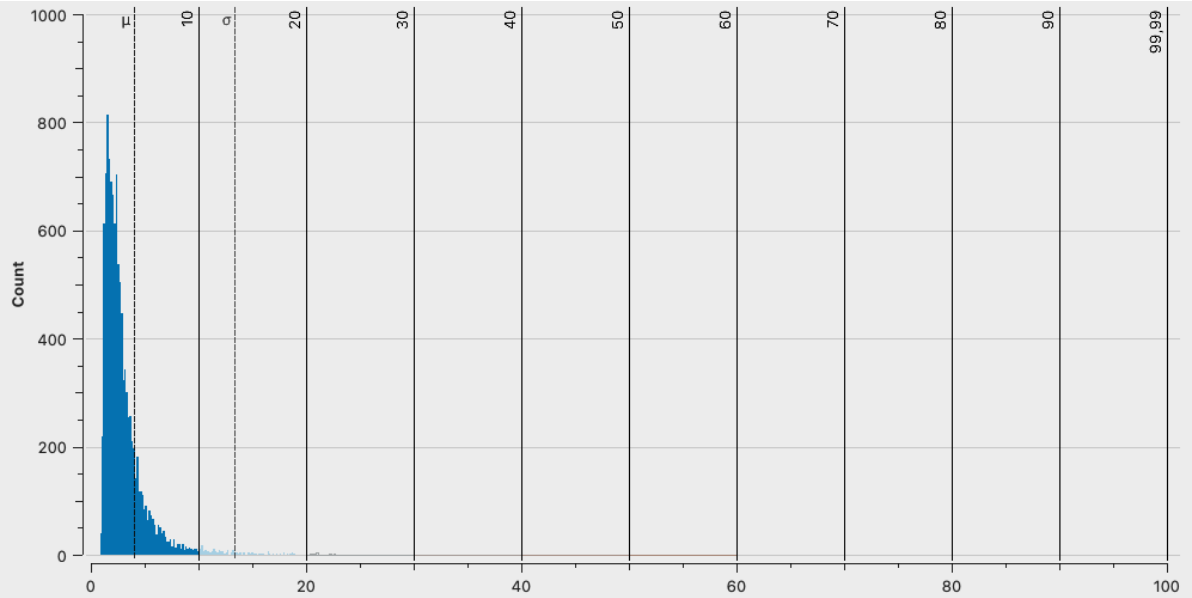
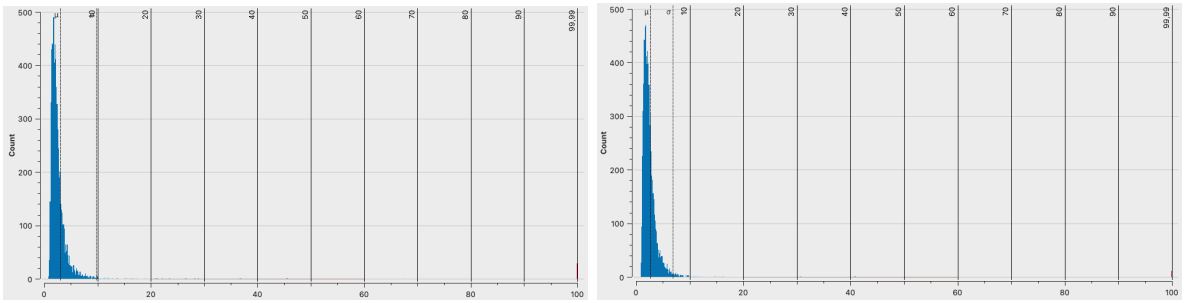
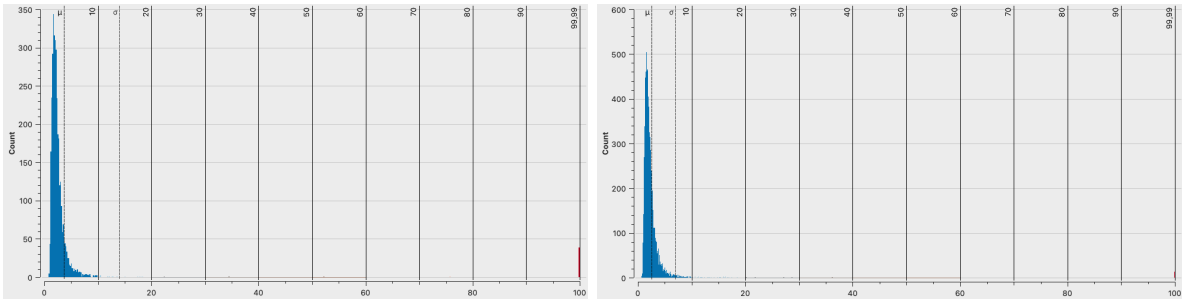


Abbildung 3.1.: Verteilung der HDOP-Werte für GPS01



(a) Histogram für GPS02 **(b)** Histogram für GPS03

Abbildung 3.2.: Verteilung der HDOP-Werte für GPS02 und GPS03



(a) Histogram für GPS04 **(b)** Histogram für GPS05

Abbildung 3.3.: Verteilung der HDOP-Werte für GPS04 und GPS05

Tabelle 3.3.: Statistik über die HDOP-Werte im Almsommer 2021 pro GPS-Gerät

Gerät	$\tilde{x}$	**Q1**	**Q3**	σ	$\bar{x}$	n_{99}	$\tilde{x}'$
GPS01	2.43	1.75	3.57	9.40	3.98	95	2.41
GPS02	2.15	1.65	2.91	6.73	3.00	29	2.15
GPS03	2.20	1.68	2.90	4.12	2.66	11	2.20
GPS04	2.14	1.69	2.81	10.4	3.60	39	2.13
GPS05	1.97	1.56	2.60	4.47	2.50	13	1.97

Ein direkter Vergleich der Datenqualität der GPS-Tracker untereinander ist nur bedingt möglich. Zwar konnte das 5-Minuten-Interval trotz Schwankungen in der TTFF pro Messung durch das Handling der berechnet Standby-Zeit meist kompensiert werden, jedoch zeichnen die einzelnen Geräte softwarebedingt nicht synchron zueinander auf. Daher kann es zu einer maximalen zeitlichen Verschiebung von 2,5 Minuten unter den Geräten kommen, in der nicht ausgeschlossen werden kann, dass sich die Position eines Gerätes verändert hat.

3.2.2. Hardware

Die Hardware-Komponenten der Geräte waren nach der Demontage weitestgehend unbeschädigt. Gehäuse und Verklebung der GPS-Module, sowie die elektronischen Bauteile und Verlötungen waren vollständig intakt. Es konnte kein Wassereintritt durch die äußere Hülle festgestellt werden, obwohl diese bei Gerät `GPS01` sichtlich den Umwelteinflüssen ausgesetzt war. Stromdurchgangsmessungen zwischen Lötpunkten und Bauteilen zeigten, dass zwischen allen Bauteilen noch ein Stromfluss möglich war. Durch das Entfernen der Verklebung der Akkus wurde nachträglich bei einer Zelle eine mechanische Verformung verursacht, welche eine potenzielle Brandgefahr mit sich bringt. Vergleichsdaten zwischen berechneter und tatsächlicher Betriebsdauer der Akkus wurden in Kapitel 3.2.1 behandelt. Die grundsätzliche Zweckmäßigkeit der Hardware ist damit gegeben.

3.2.3. Software

Die Programmierung der Mikrocontroller erfüllte ebenfalls ihren Zweck - die Messintervalle der KFZ-Geräte wurden eingehalten. Das montierte Gerät weist jedoch eine höhere Anzahl an Messpunkten pro Tag auf, wobei dies nicht nachweisbar auf Software-Probleme zurückzuführen ist.

3.2.4. Wetterdaten

Die ergänzenden meteorologischen Daten wurden im 5-Minuten-Intervall bezogen und in einer Datenbank im JSON-Format gespeichert. Der Zeitraum der Datenaufnahme überstreckte sich vom 29.06.21 - 13:15 Uhr, also erst drei Tage nach dem Almauftrieb bis zum Tag des Almabtriebs am 11.09.21 - 23:59 Uhr. In diesen 74.45 Tagen wurden 25073 Datensätze ermittelt, damit mehr als bei einem 5-Minuten-Intervall erwartet.

Der Datensatz 3.1 zeigt beispielhaft die ermittelten Daten.

Quellcode 3.1: Beispiel eines Wetterdatensatzes vom 10.09.21 - 03:51:52


```
{
    "lat": 47.7144,
    "lon": 12.9515,
    "current": {
        "dt": 1631237511,
        "uvi": 0,
        "temp": 7.51,
        "clouds": 47,
        "sunset": 1631295111,
        "sunrise": 1631248730,
        "weather": [
            {"id": 802,
            "icon": "03n",
            "main": "Clouds",
            "description": "scattered clouds"
            }],
        "humidity": 77,
        "pressure": 1017,
        "wind_deg": 150,
        "dew_point": 3.74,
        "feels_like": 4.88,
        "visibility": 10000,
        "wind_speed": 4.12
    },
    "timezone": "Europe/Vienna",
    "timezone_offset": 7200
}
```

3.3. GPS-Logger - Generation 2: Almsommer 2022

Der zweite Almsommer im Untersuchungszeitraum erstreckte sich, mit einer Dauer von 102 Tagen, vom 04.06.2022 bis 14.09.2022. Aufgrund der Softwareänderungen sollte die regelmäßige Datenerhebung jedoch erst mit dem 01.07.2022 starten. Entsprechend verkürzt sich die Messperiode auf 75 Tage und ist damit zwei Tage kürzer als die Vorjährige.

Die ersten Daten der GPS-Geräte wurden bereits Ende Mai zu Testzwecken ermittelt und geschrieben. Hierbei waren die Code-Teile, welche dafür sorgen, dass erst ab Juli mit den regelmäßigen Datenaufnahme begonnen werden soll, noch deaktiviert. Am 24.05.2022 wurden die Geräte aktiviert und an den Halsbändern montiert. Ab diesem Datum beendete sich der Standby-Modus einmal täglich zur Prüfung des aktuellen Datums anhand der GPS-Daten.

3.3.1. Positionsdaten

Beginnend mit dem 01.07.2022 wurden die Geräte in den 5-Minuten-Interval-Modus versetzt. Durch technische bedingte Ausfälle erreichten die Geräte `GPS01` und `GPS06` diese Phase nicht - beide konnten jedoch bis zum Ausfall die täglichen Intervalle einhalten. Für die weitere Betrachtung der Ergebnisse bleiben diese beiden Geräte stets unberücksichtigt.

Auffällig ist, dass bei den übrigen Geräten, mit Ausnahme von `GPS07`, des Öfteren größere Zeiträume ohne Messwerte verzeichnet sind. Der Tracker `GPS02` zeichnete zwischen 02.07. und 23.07.22, die Geräte `GPS05` und `GPS08` zwischen 04.07. und 13.07.22 und `GPS04` zwischen 07.07. und 10.07 keine Daten auf. Weitere Standby-Phasen über mehrere Tage können ebenso im August und September beobachtet werden. Durch die unterschiedlichen Standbyzeiten der einzelnen Geräte, konnten ausschließlich am 01.07.22 über alle Geräte hinweg vergleichbare Daten aufgenommen werden. Die Anzahl der Tage an denen gemessen wurde, beträgt bei `GPS02` 12 Tage, bei `GPS03` 14 Tage, bei `GPS04` 13 Tage, bei `GPS05` 15 Tage, bei `GPS07` 5 Tage und bei `GPS08` 7 Tage. Zu den Zeiten, in denen die Geräte aufzeichneten, wurden die 5-Minuten-Intervalle in der Regel eingehalten. Teils wurden in einem Messzyklus jedoch mehrere Datensätze geschrieben, obwohl nur der beste Fix einer Messung geloggt werden sollte.

Bei Betrachtung der Daten fiel auf, dass die hohen HDOP-Werte des Vorjahres nicht mehr auftraten. Jedoch konnten Datensätze beobachtet werden, mit Koordinaten-Werten von 0. In Tabelle 3.4 ist die Anzahl aller Datensätze n, sowie die reduzierte Anzahl ohne 0-Koordinaten n_0, sowie die Anzahl der ermittelten Koordinaten im Zeitraum des Almsommers ab Messbeginn am 01.07.2022 n_{red}, gelistet. Darüber hinaus werden die Start- und Endzeiten der GPS-Logs dargestellt, sowie die verwendete Hardware-Version V bzgl. Gehäuse und Akkukapazität. Zum Abgleich der Streuungswerte mit dem Vorjahr wurde je der Median $\tilde{x}_{red}$ der HDOP-Werte aus der Menge n_{red} ermittelt. In Abbildung 3.4 wird beispielhaft ersichtlich, dass die Streuung und damit die Messpunkte außerhalb

des Untersuchungsgebiet geringer ist als im Vorjahr.

Tabelle 3.4.: Anzahl der Messpunkte und Messzeiten, sowie Median der HDOP-Werte im Almsommer 2022 pro GPS-Gerät

Gerät	V	n	n_0	n_{red}	**Startzeit** (MESZ)	**Endzeit** (MESZ)	**HDOP** $\tilde{x}_{red}$
GPS01	1	355	338	0	18.05.22 10:39	12.06.22 11:02	-
GPS02	1	3135	3111	2846	20.05.22 18:20	07.10.22 04:54	1.09
GPS03	1	3404	3353	3024	20.05.22 19:33	15.09.22 14:36	1.37
GPS04	1	3316	3267	2782	17.05.22 18:48	05.10.22 21:04	1.39
GPS05	1	4512	4477	4447	20.05.22 18:33	11.08.22 20:49	1.21
GPS06	2	16	14	0	21.05.22 13:39	11.06.22 14:34	-
GPS07	2	1525	1504	1479	21.05.22 13:36	05.07.22 12:49	1.29
GPS08	2	1068	1057	1024	21.05.22 13:45	22.08.22 05:24	1.29

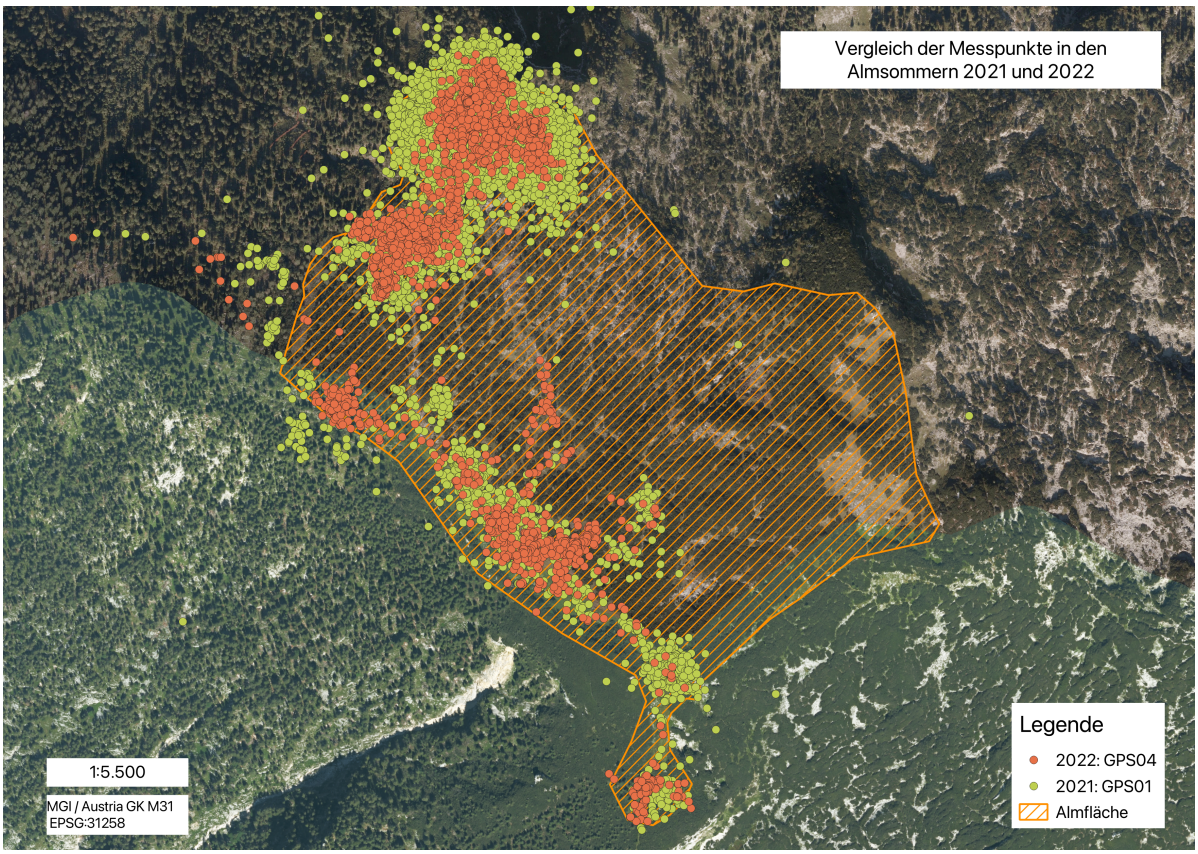


Abbildung 3.4.: Beispielhafter Vergleich der aufgezeichnete Messpunkte in den Almsommern 2021 und 2022

3.3.2. Hardware

Bei den wiederverwendeten Geräten der ersten Generation konnten nach Demontage teils Mängel festgestellt werden. Die Silikon-Verklebungen der GPS-Module in den Gehäusedeckeln waren teilweise, bei einem Gerät vollständig gelöst. Bei `GPS01` haben sich die Lötstellen zu den UART-Pins des GPS-Bausteins gelöst. Zwei der Akkumulatoren wiesen leichte mechanische, jedoch unkritische Beschädigungen auf.

Die Geräte der zweiten Generation waren gänzlich unbeschädigt. Einzig die Verklebung der GPS-Antennen mit Zwei-Komponenten-Kleber wurde spröde mit leichten Rissen, waren jedoch noch fest verbunden. Das Gehäuse der Generation 2 scheint Bauteile und LiPos besser zu schützen.

Abgeleitet aus den ermittelten Messtagen in Kapitel 3.3.1, ist trotz Verdopplung der Akkukapazität, die Anzahl der Tage an denen Werte ermittelt wurden, um Zweidrittel niedriger, im Vergleich zu den Werten der ersten Generation im zweiten Almsommer. Ein Vorjahres-Vergleich der Messtage mit den 3000 mAh-Geräten, zeigte eine Halbierung der Akkulaufzeit im Almsommer 2022.

3.3.3. Software

Die Codeanpassungen zur verzögerten Datenaufnahme haben gegriffen. Vor dem Monat Juli erwachten die Geräte lediglich einmal am Tag aus dem Standby-Modus - ab Juli wurde automatisch in den 5-Minuten-Intervall-Modus gewechselt. Eine zeitlich synchronisierte Erfassung - also jeweils zur Minute 5 - konnte ebenfalls erreicht werden, jedoch wurden, über den besten Messwert einer Reihe hinaus, noch weitere Messwerte auf die SD-Karte geschrieben. Dies erfordert eine ungewollte, manuelle Bereinigung der Daten.

Trotz scheinbar ausreichender Stromversorgung zeigen die Daten erhebliche zeitliche Lücken, durch falsch berechnete Standby-Zeiten. Dies bewirkt, dass trotz der synchronisierten Erfassung die Interaktionen der einzelnen Kühe nur sehr eingeschränkt abgeleitet werden können. Der maximale Zeitraum, indem durchgängig Daten eines Gerätes (`GPS05`) erfasst wurden, beträgt sechs Tage.

Die Erhöhung der Anzahl der nötigen Fixes in einer Messreihe haben die erhoffte Verbesserung der Streuungswerte erwirkt.

3.3.4. Wetterdaten

Im Zeitraum des Almsommers 2022 wurden aufgrund einer unbemerkten Umstellung der OpenWeatherMap-API keine Wetterdaten über das Skript aufgezeichnet.

4. Diskussion

Zur Interpretation der Ergebnisse werden zunächst die beiden Messperioden in den Jahren 2021 und 2022, mit den unterschiedlichen Hard- und Softwarevoraussetzungen einzeln betrachtet, um die jeweiligen Vor- und Nachteile herauszustellen. Aus der Zusammenführung dieser Teilergebnisse lässt sich eine abschließende Bewertung und Empfehlung für künftige, ähnlich gelagerte Projekte ableiten.

4.1. Fazit der Datenaufnahme: Almsommer 2021

Die Betrachtung der Anzahl der aufgenommenen Messpunkte N_{21}, sowie die maximale Anzahl an Tagen Δt_{21} an denen eine Datenaufnahme stattfand, wie in Tabelle 3.2 ersichtlich, zeigen durchaus Unterschiede zwischen den Geräten. Die KFZ-Geräte `GPS02`, `GPS03` und `GPS05` bewegen sich in einem ähnlichen Feld mit ca. 24 Messtagen und ca. 285 aufgenommenen Messpunkten pro Tag. Das ebenfalls im Auto verbliebene `GPS04` hat mit 281 knapp weniger Messungen pro Tag erreicht, jedoch mit 12.57 Tagen eine signifikant kürzere Laufzeit. Die Messung der Restspannung der LiPos nach dem Almsommer zeigte jedoch mit 0.0 V, also dem Erreichen des Tiefenentladungsschutzes, ein identisches Bild über alle Geräte hinweg. Entsprechend dürfte die Ursache nicht beim Mikrocontroller zu suchen sein, da bei einem Defekt vermutlich auch die Batterie nicht mehr weiter entladen worden wäre. Gründe könnten in einem fehlerhaften Tiefenentladungsschutz oder in einem Zelldefekt des Akkus liegen.

Das Gerät `GPS01` hat hingegen eine etwas höhere Laufzeit mit 29.65 Tagen. Eine Erklärung hierfür könnte eine konstantere und allgemein niedrigere Umgebungstemperatur direkt am Hals des Tieres in subalpiner Umgebung sein, welche zu einer geringeren Selbstentladung des Akkus führt. Zur Verifizierung wäre ein zusätzlicher Temperatursensor nötig. Mit durchschnittlich 381 Messungen pro Tag, ist jedoch die Anzahl der gemessenen Punkte höher, und verglichen mit den erwarteten 288 Messungen pro Tag bei einem 5-Minuten-Interval, deutlich zu hoch. Hier liegt die Vermutung nahe, dass der Fehler im Code oder am Mikrocontroller selbst zu suchen ist. Denkbar wäre es, dass sich die MCU, z.B. aufgrund von Fehlerströmen, häufiger neu gestartet und damit die Standby-Zeiten verkürzt hat.

In Anbetracht der inkonstanten äußeren Einflüsse auf die Batterien, können die vorhergesagten Berechnung aus Kapitel 2.6.1 als treffend bewertet werden.

Aufgrund der fehlenden Synchronität der Datensätze untereinander ist eine Analyse der Interaktion nur bedingt umsetzbar. Hinzu kommt, dass `GPS01` im realen Umfeld das

Messintervall nicht konstant aufzeichnen konnte.

Ein möglicher Einsatzzweck der Daten ist jedoch die Bewertung der Herdenbewegung, in Bezug auf die genutzten Areale der Almfläche. Dies wäre beispielsweise durch die Ableitung der Trajektorien oder der visuellen Darstellung einer Heatmap denkbar.

Die Betrachtung der verbauten Komponenten zeigte zwar kaum mechanische Beschädigungen oder Wassereintritte, jedoch ist ein gewisses Grundrisiko durch Akku-Schäden nicht auszuschließen. Insbesondere bei zusätzlicher Auswertung der resultierten Akkulaufzeiten, war eine Verdopplung der Akkukapazität sinnvoll, um den gesamten Almsommer abzudecken. Diese beiden Aspekte führten zum Schluss, dass eine Optimierung der Gehäuse-Bauform nötig ist, um die verbauten Akkuzellen besser zu schützen.

4.2. Fazit der Datenaufnahme: Almsommer 2022

Die Lehren des ersten Almsommers und die daraus resultierenden Optimierungen brachten in Teilbereichen Verbesserungen. Insbesondere die Gehäuse bieten einen deutlich besseren Schutz der zwei Akkuzellen und eine einfachere Handhabung ohne Verschraubungen oder Kabelbinder. Einzig die poröse Verklebung der GPS-Module im Gehäuse war nicht optimal und könnte durch die Wahl eines geeigneten, dauerelastischen Klebstoffes verbessert werden.

Trotz verdoppelter Akkukapazität fielen die Anzahl der Messtage über alle Geräte hinweg deutlich niedriger als im Vorjahr aus. Die Gründe sind einerseits in der Software zu suchen, andererseits könnten diese durch die Fehllieferung und Taktfrequenz-Anpassungen begründet sein. Eine genaue Ermittlung der Ursache gestaltet sich jedoch schwierig, da im Testbetrieb die Geräte wie erwartet aufzeichneten. Verbesserungen in den Debug-Möglichkeiten wären eine geeignete Grundlage zur Fehleranalyse.

Ein Vergleich der HDOP-Werte zum Vorjahr zeigt eine niedrige Streuung der räumlichen Messpunkte und damit eine potenziell höhere Datenqualität. Dies dürfte auf die Softwareanpassungen hinsichtlich der Wahl des besten Messpunktes einer Reihe zurückzuführen sein.

Bezüglich der Datensynchronität konnte eine potenzielle Verbesserung beobachtet werden, da die Geräte meist zu den gewünschten Zeitpunkten „zu Minute 5“ aktiv wurden. Reell wurden aber dennoch aufgrund der tageweisen Ausfälle kaum vergleichbare Daten generiert. Eine Erklärung bedarf ebenfalls Debug-Optionen.

4.3. Zusammenfassung

Die entwickelten GPS-Logger erfüllten in beiden Almsommern nur in Teilbereichen ihren Hauptzweck. Eine Bewertung des Interaktionsverhaltens der einzelnen Weidetiere untereinander ist, aufgrund von Datenlücken und fehlender Synchronität, mit den gesammelten Geodaten nur bedingt möglich. Dadurch ist auch die Wahl der Messintervalle für den gedachten Einsatzzweck nur eingeschränkt bewertbar. Ausgehend von den Angaben in der Literatur, kombiniert mit den vorliegenden Ergebnissen, wären jedoch tendenziell kürzere Messintervalle zielführender, insofern dies mit den Akkulaufzeiten vereinbar ist. Eben der Energiebedarf der Datenlogger stellt weiter eine der Hauptherausforderungen dar. Die schlechteren Ergebnisse des zweiten Almsommers, trotz verdoppelter Akkukapazität, waren jedoch unerwartet und können ohne weitere Debug-Möglichkeiten nur schwer begründet werden.

Die Wahl und Optimierung der Hardwarekomponenten war hingegen zweckmäßig und Bedarf nur in kleinen Teilen weiterer Verbesserungen.

4.4. Aussicht

Für den Zweck der Interaktionsanalyse zwischen den Individuen einer Rinderherde, bedarf es weiterer Adaptierungen einzelner Parameter der GPS-Datenlogger. Der Fokus sollte hierbei auf die Themen Ausfallsicherheit, Positionsgenauigkeit und Akkulaufzeiten gerichtet werden.

Ein möglicher Lösungsansatz zu den ersten beiden Punkten könnte eine Reduktion der Funktionen zur Daten-Aufbereitung auf dem Mikrocontroller selbst sein. Ein direktes Logging der Rohdaten, anstatt der geparsten NMEA-Sätze, könnte den Quellcode vereinfachen und damit die Last der MCU reduzieren. Optimal wäre hierbei eine Untersuchung geeigneter GPS-Empfänger, zur Ausgabe der Rohdaten als Receiver Independent Exchange Format (RINEX), welches im Postprocessing eine Optimierung der Datenqualität erlaubt. Bei Adaption der Tracker sind ausgiebige Feldtests und Zwischenauswertungen in leicht erreichbarem Umfeld zu empfehlen, da eine Anpassung während des Almsommers nicht möglich ist.

Zur Verbesserung der Akkulaufzeiten sollten neben der schlichten Erhöhung der Kapazität, welche mit einer Gewichtserhöhung einhergeht, weitere Optionen geprüft werden. Eine optimierte Formel zur Berechnung der theoretischen Akkulaufzeit, unter Einbeziehung der erhobenen Wetter-/Temperaturdaten könnten eine genauere Grundlage bieten. Denkbar wäre die Verwendung flexibler Photovoltaikmodule an der Außenseite der Hüllen oder am Halsband, unter Berücksichtigung von Sicherheitsaspekten im Umgang mit den Tieren. Zusätzliche Bluetooth Low Energy (BLE)-, Long Range Wide Area Network (LoRaWAN)- oder ähnliche Module würden den Energieverbrauch der Tracker erhöhen, jedoch dabei eine Fernsteuerung der Funktionen erlauben. Dadurch wäre beispielsweise eine bedarfsgerechte Ein-/Abschaltung der Geräte oder Anpassung der In-

tervalle möglich, womit die vorhandene Akkukapazität und Datenerhebung zielgerichtet genutzt werden könnte.

Ein weiteres Augenmerk sollte auf das Debugging gerichtet werden, um potenzielle Fehler besser zu erkennen. Spezielle Bootloader können helfen, die Fehlersuche zu vereinfachen und dabei gleichzeitig die Grundspeicherlast zu reduzieren, wodurch der eigentlichen Software mehr Speicher zur Verfügung stünde. Je nach experimentellem Stadium eines alternativen Bootloaders, ist jedoch die Ausfallsicherheit kritisch zu beurteilen.

Trotz der guten Ergebnisse hinsichtlich der Sicherheit vor mechanischen LiPo-Schäden im zweiten Almsommer, sollte die Verwendung weitere Akkutechnologien, wie LiFePO4, tiefer untersucht werden, um die ausgehende Brandgefahr auf ein Minimum reduzieren zu können.

Optimierungen im Handling der Geräte könnte die Nutzbarkeit weiter verbessern. Indem Ladepunkte oder die Slots der Speicherkarten nach außen geführt werden, wäre ein Öffnen der GPS-Logger zur Auswertung nicht mehr nötig. Platinen und Akkus könnten bei Bedarf mit einer Vergussmasse stabilisiert werden, was wiederum eine Reduktion der mechanischen Belastung bewirken sollte.

Zur Perfektionierung der Geräte bedarf es weiterer intensiver Tests, Untersuchungen und Entwicklungsarbeit im Hard- und Softwarebereich. Die gesammelten Ergebnisse liefern dabei eine Grundlage zur Evaluierung weiterer Ansatzpunkte.

Literaturverzeichnis

Accuracy and quality of weather data - OpenWeatherMap (2022). URL: https://openweathermap.org/accuracy-and-quality (besucht am 14. 11. 2022).

ATmega328P | Microchip Technology (2022). URL: https://www.microchip.com/en-us/product/ATmega328P#document-table (besucht am 14. 11. 2022).

Augustine, David J. und Justin D. Derner (2013). „Assessing Herbivore Foraging Behavior with GPS Collars in a Semiarid Grassland". In: *Sensors* 13.3, S. 3711–3723. ISSN: 1424-8220. DOI: 10.3390/s130303711.

Cain, James W. III et al. (2005). „Influence of topography and GPS fix interval on GPS collar performance". In: *Wildlife Society Bulletin* 33.3, S. 926–934. ISSN: 1938-5463. DOI: 10.2193/0091-7648(2005)33[926:IOTAGF]2.0.CO;2.

Cochran, William W. und Rexford D. Lord (1963). „A Radio-Tracking System for Wild Animals". In: *The Journal of Wildlife Management* 27.1, S. 9–24. ISSN: 0022-541X. DOI: 10.2307/3797775.

Forin-Wiart, Marie-Amélie et al. (2015). „Performance and Accuracy of Lightweight and Low-Cost GPS Data Loggers According to Antenna Positions, Fix Intervals, Habitats and Animal Movements". In: *PLOS ONE* 10.6, e0129271. ISSN: 1932-6203. DOI: 10.1371/journal.pone.0129271.

Hill, Susan C., Joseph Jelemensky und Mark R. Heene (1989). „Queued serial peripheral interface for use in a data processing system". US-Pat. 4816996A. Motorola Inc.

Khan, Kamrul et al. (2018). „PKL Electrochemical Cell and the Peukert's Law". In: *International Journal Of Advance Research And Innovative Ideas In Education* 4, S. 4219–4227.

Klements, Michael (2021). *Making An Ultra Low Power Arduino Pro*. The DIY Life. URL: https://www.the-diy-life.com/making-an-ultra-low-power-arduino-pro/ (besucht am 10. 06. 2023).

Maxa, J., S. Thurner und G. Wendl (2015). „Evaluation of different global navigation satellite tracking systems and analyses of movement patterns of cattle on alpine pastures". In: *Agricultural Engineering International: CIGR Journal*. ISSN: 1682-1130.

Nakano, M. et al. (2020). „Spatial Distribution of Forage Intake by Grazing Beef Cows on an Alpine Pasture in Japan". In: *International Grassland Congress Proceedings*.

One Call API 3.0 - OpenWeatherMap (2022). One Call API - Documentation. URL: https://openweathermap.org/api/one-call-3 (besucht am 14. 11. 2022).

Turner, L.W. et al. (2000). „Monitoring cattle behavior and pasture use with GPS and GIS". In: *Canadian Journal of Animal Science* 80.3, S. 405–413. ISSN: 0008-3984. DOI: 10.4141/A99-093.

u-blox (2023). *u-blox 8 / u-blox M8 Receiver description: Including protocol specification v15-20.30,22-23.01*.

Abbildungsverzeichnis

Tabellenverzeichnis

Quellcodeverzeichnis

Anhang

A. Beispiel einer Antwort des Requests 2.1

Quellcode A.1: Beispiel einer Antwort des Requests 2.1

```
{
  "lat":47.7144,
  "lon":12.9515,
  "timezone":"Europe/Vienna",
  "timezone_offset":3600,
  "current":{
     "dt":1668427090,
     "sunrise":1668406315,
     "sunset":1668440008,
     "temp":9.85,
     "feels_like":7.55,
     "pressure":1016,
     "humidity":39,
     "dew_point":-3.05,
     "uvi":1.05,
     "clouds":0,
     "visibility":10000,
     "wind_speed":4.63,
     "wind_deg":140,
     "weather":[
        {
           "id":800,
           "main":"Clear",
           "description":"clear sky",
           "icon":"01d"
        }
     ]
  },
  "alerts":[
     {
        "sender_name":"Deutscher Wetterdienst",
        "event":"STURMBOEEN",
        "start":1668365880,
        "end":1668434400,
        "description":"Es treten oberhalb 1500 m Sturmboeen mit
           Geschwindigkeiten bis 75 km/h (21 m/s, 41 kn, Bft 9) aus
           suedlicher Richtung auf.",
        "tags":[

        ]
```

```
        },
        {
           "sender_name":"Deutscher Wetterdienst",
           "event":"gale-force gusts",
           "start":1668365880,
           "end":1668434400,
           "description":"There is a risk of gale-force gusts (Level 2 of 4).
              Height range: 1500 m; Max. gusts: 75 km/h; Wind direction:
              south",
           "tags":[
              "Wind",
              "Wind"
           ]
        },
        {
           "sender_name":"Deutscher Wetterdienst",
           "event":"WINDBOEEN",
           "start":1668416400,
           "end":1668434400,
           "description":"In Foehntaelern treten Windboeen mit
              Geschwindigkeiten bis 50 km/h (14 m/s, 28 kn, Bft 7) aus
              suedlicher Richtung auf.",
           "tags":[

           ]
        },
        {
           "sender_name":"Deutscher Wetterdienst",
           "event":"wind gusts",
           "start":1668416400,
           "end":1668434400,
           "description":"There is a risk of wind gusts (level 1 of 4).
              Occurrence: in foehn valleys; Max. gusts: 50 km/h; Wind
              direction: south",
           "tags":[
              "Wind",
              "Wind"
           ]
        },
        {
           "sender_name":"Deutscher Wetterdienst",
           "event":"slight risk of icy surfaces",
           "start":1668466800,
           "end":1668499200,
```

```
            "description":"There is a slight risk of icy surfaces (level 1 of
               4). Occurrence: isolated",
            "tags":[

            ]
         },
         {
            "sender_name":"Deutscher Wetterdienst",
            "event":"frost",
            "start":1668466800,
            "end":1668499200,
            "description":"There is a risk of frost (Level 1 of 2). Minimum
               temperature: ~ 0 C; local minimum: in valleys and dips -2 C",
            "tags":[
               "Extreme temperature value"
            ]
         }
      ]
   }
```

B. Quellcode der GPS-Logger (Generation 1)

Quellcode B.1: Quellcode der GPS-Logger (Generation 1)

```
1   #include <EEPROM.h>
2
3   #include <SoftwareSerial.h>
4   #include <TinyGPS++.h>
5
6   #include <SPI.h>
7   #include <SD.h>
8
9   #include <Sleep_n0m1.h>
10
11  // config (can be overwritten in config.txt file)
12  bool DEBUG = true;
13  int INTERVAL = 300;
14
15  // LogTgt
16  const int LOG_SERIAL = 1;
17  const int LOG_SDCARD = 2;
18
19  String FILENAME = "gpsLog.txt";
20
21  uint8_t PSM[] = {0xB5, 0x62, 0x06, 0x11, 0x02, 0x00, 0x08, 0x01, 0x22, 0x92
        }; // Setup for Power Save Mode (Default Cyclic 1s)
22
23  uint8_t GPSFulloff[] = {0xB5, 0x62, 0x02, 0x41, 0x08, 0x00, 0x00, 0x00, 0
        x00, 0x00, 0x02, 0x00, 0x00, 0x00, 0x4D, 0x3B};
24  uint8_t GPSFullon[] = {0xB5, 0x62, 0x02, 0x41, 0x08, 0x00, 0x00, 0x00, 0x00
        , 0x00, 0x01, 0x00, 0x00, 0x00, 0x4C, 0x37};
25
26  uint8_t GPSoff[] = {0xB5, 0x62, 0x06, 0x04, 0x04, 0x00, 0x00, 0x00, 0x08, 0
        x00, 0x16, 0x74};
27  uint8_t GPSon[] = {0xB5, 0x62, 0x06, 0x04, 0x04, 0x00, 0x00, 0x00, 0x09, 0
        x00, 0x17, 0x76};
28
29  const int IO_SDCARD = 10;
30  uint32_t GPSendtimemS; // time in mS when GPS got a fix
31  uint32_t GPSstarttimemS; // time in mS when GPS is scanned fo fix
32
33  SoftwareSerial gpsSerial(4, 3); // rx,tx
34
35  const unsigned long sleepTimeBase = DEBUG ? 30000 : 300000;
```

```
36
37    Sleep sleep;
38    TinyGPSPlus gps;
39
40    bool withSDCard = false;
41
42    void setup()
43    {
44      if (DEBUG)
45      {
46        Serial.begin(115200);
47        logger("DEBUG-MODE: Init", LOG_SERIAL);
48      }
49
50      // init SDCard on
51      if (withSDCard && SD.begin(IO_SDCARD))
52      {
53        logger("Init SDCard: CardReader found", LOG_SERIAL);
54
55        // write header if new file is needed
56        if (!SD.exists(FILENAME))
57        {
58          logger("Sat;HDOP;Lat;Lon;Fix;Date;Time;Age;Alt;Course;Speed;Card",
              LOG_SDCARD);
59        }
60      }
61      else
62      {
63        logger("Init SDCard: Failed - withSDCard -> " + String(withSDCard),
            LOG_SERIAL);
64        withSDCard = false;
65      }
66
67      // init GPS
68      logger("Init GPS: Started... with " + String(gps.libraryVersion()),
          LOG_SERIAL);
69      digitalWrite(2, HIGH);
70      gpsSerial.begin(9600);
71
72      UBLOX_GPS_Wakeup();
73      getValidSignal(true);
74    }
75
76    void loop()
```

```
77  {
78    unsigned long timeToFix = 0;
79
80    logger("WakeUpGPS", LOG_SERIAL);
81    digitalWrite(2, HIGH);
82    UBLOX_GPS_Wakeup();
83
84    logger("GetValidSignal", LOG_SERIAL);
85    timeToFix = getValidSignal(false);
86
87    UBLOX_GPS_Shutdown();
88
89    // SD.begin(IO_SDCARD);
90    String dataLine = "";
91    // smartDelay(30000);
92
93    // Satelites
94    dataLine = gps.satellites.isValid() ? String(gps.satellites.value()) : ""
          ;
95    dataLine += ";";
96    logInline(dataLine);
97
98    // HDOP
99    dataLine = gps.hdop.isValid() ? String(gps.hdop.hdop()) : "";
100   dataLine += ";";
101   logInline(dataLine);
102
103   // Lat/Lon
104   dataLine = gps.location.isValid() ? String(gps.location.lat(), 8) : "";
105   dataLine += ";";
106   logInline(dataLine);
107
108   dataLine = gps.location.isValid() ? String(gps.location.lng(), 8) : "";
109   dataLine += ";";
110   logInline(dataLine);
111
112   // Fix age
113   dataLine = gps.location.isValid() ? String(gps.location.age()) : "";
114   dataLine += ";";
115   logInline(dataLine);
116
117   // Date
118   TinyGPSDate d = gps.date;
119   dataLine = "";
```

```
120     if (d.isValid())
121     {
122       dataLine += String(d.year()) + "-";
123
124       dataLine += d.month() < 10 ? "0" : "";
125       dataLine += String(d.month()) + "-";
126
127       dataLine += d.day() < 10 ? "0" : "";
128       dataLine += String(d.day());
129     }
130     dataLine += ";";
131     logInline(dataLine);
132
133     // Time
134     TinyGPSTime t = gps.time;
135     dataLine = "";
136     if (t.isValid())
137     {
138       dataLine += t.hour() < 10 ? "0" : "";
139       dataLine += String(t.hour()) + ":";
140
141       dataLine += t.minute() < 10 ? "0" : "";
142       dataLine += String(t.minute()) + ":";
143
144       dataLine += t.second() < 10 ? "0" : "";
145       dataLine += String(t.second());
146     }
147     dataLine += ";";
148     logInline(dataLine);
149
150     // Age
151     dataLine = d.isValid() ? String(d.age()) : "";
152     dataLine += ";";
153     logInline(dataLine);
154
155     // Altitude
156     dataLine = gps.altitude.isValid() ? String(gps.altitude.meters()) : "";
157     dataLine += ";";
158     logInline(dataLine);
159
160     // Course
161     dataLine = gps.course.isValid() ? String(gps.course.deg()) : "";
162     dataLine += ";";
163     logInline(dataLine);
```

```
164
165     // Speed
166     dataLine = gps.speed.isValid() ? String(gps.speed.kmph()) : "";
167     dataLine += ";";
168     logInline(dataLine);
169
170     // Card
171     dataLine = gps.course.isValid() ? String(TinyGPSPlus::cardinal(gps.course
            .deg())) : "";
172     logInline(dataLine);
173
174     logger("", LOG_SDCARD);
175
176     digitalWrite(2, LOW);
177     logger("ShutDownMCU", LOG_SERIAL);
178     sleep.pwrDownMode();
179
180     uint32_t currentTime = gps.time.minute() * 60 + gps.time.second();
181     uint32_t secondsToStart = INTERVAL - (currentTime % INTERVAL);
182     Serial.println(currentTime);
183     Serial.println(secondsToStart);
184     Serial.println(secondsToStart * 1000 - timeToFix);
185     sleep.sleepDelay(secondsToStart * 1000 - timeToFix);
186
187     logger("WakeUpMCU", LOG_SERIAL);
188   }
189
190   unsigned long getValidSignal(bool forever)
191   {
192     GPSstarttimemS = millis();
193     unsigned int countValids = 0;
194     unsigned long firstValidTime = 0;
195     uint8_t GPSchar;
196     while (1)
197     {
198       if (gpsSerial.available() > 0)
199       {
200         GPSchar = gpsSerial.read();
201         gps.encode(GPSchar);
202       }
203       if (gps.location.isUpdated() && gps.altitude.isUpdated() && gps.
              location.isValid())
204       {
205         if (firstValidTime == 0)
```

```
206         {
207           firstValidTime = millis();
208         }
209         countValids++;
210         GPSendtimemS = millis();
211         logger(String(gps.location.lat(), 6), LOG_SERIAL);
212         logger(String(gps.location.lng(), 6), LOG_SERIAL);
213         if (countValids >= 10)
214         { // try to have ten signals before logging
215           logger("10 valid fixes", LOG_SERIAL);
216           break;
217         }
218       }
219       // fallback if we dont have further valid signals within 5 seconds
220       if (countValids >= 1 && millis() - firstValidTime >= 15000)
221       {
222         logger("5sec after first fix", LOG_SERIAL);
223         break;
224       }
225
226       if (millis() - GPSstarttimemS >= sleepTimeBase && !forever)
227       {
228         logger("Timeout", LOG_SERIAL);
229         return 0;
230       }
231     }
232     logger("FoundSignal", LOG_SERIAL);
233     logger(String(GPSendtimemS - GPSstarttimemS) + "ms", LOG_SERIAL);
234     return GPSendtimemS - GPSstarttimemS;
235   }
236
237   void logger(String msg, int tgt)
238   {
239     switch (tgt)
240     {
241     case LOG_SERIAL:
242       if (!DEBUG)
243       {
244         return;
245       }
246       Serial.println(msg);
247     case LOG_SDCARD:
248       if (!withSDCard)
249       {
```

```
250         return;
251       }
252       if (!logToSD(msg))
253       {
254         msg = "SDCard failed with: " + msg;
255         logger(msg, LOG_SERIAL);
256       }
257     default:
258       logger("LogTgt missing: " + msg, LOG_SERIAL);
259       break;
260     }
261   }
262
263   bool logToSD(String msg)
264   {
265     bool success = false;
266     if (SD.begin(IO_SDCARD))
267     {
268       File file = SD.open(FILENAME, FILE_WRITE);
269       logger("SD Card connected", LOG_SERIAL);
270       if (file)
271       {
272         file.println(String(msg));
273         file.close();
274         logger("SD write succeeded", LOG_SERIAL);
275         success = true;
276       }
277       else
278       {
279         logger("SD write failed", LOG_SERIAL);
280       }
281     }
282     return success;
283   }
284
285   void logInline(String dataString)
286   {
287     Serial.print(dataString);
288     File file = SD.open(FILENAME, FILE_WRITE);
289     if (file)
290     {
291       file.print(String(dataString));
292       file.close();
293       logger("SD write success", LOG_SERIAL);
```

```
294       }
295       else
296       {
297         logger("SD write failed", LOG_SERIAL);
298       }
299     }
300
301     void UBLOX_GPS_Shutdown()
302     {
303       // sends command over serial interface to GPS to put it in PMREQ backup
              mode
304       uint8_t index;
305       uint8_t UBLOX_GPSStandby[] = {0xB5, 0x62, 0x02, 0x41, 0x08, 0x00, 0x00, 0
              x00, 0x00, 0x00, 0x02, 0x00, 0x00, 0x00, 0x4D, 0x3B};
306
307       for (index = 0; index < sizeof(UBLOX_GPSStandby); index++)
308       {
309         gpsSerial.write(UBLOX_GPSStandby[index]);
310         if (DEBUG)
311         {
312           Serial.print(UBLOX_GPSStandby[index], HEX);
313           Serial.print(" ");
314         }
315       }
316       logger("", LOG_SERIAL);
317     }
318
319     void UBLOX_GPS_Wakeup()
320     {
321       gpsSerial.println(); // send some characters to GPS to wake it up
322     }
```

C. Quellcode der GPS-Logger (Generation 2)

Quellcode C.1: Quellcode der GPS-Logger (Generation 2)

```
1   #include <Arduino.h>
2
3   #include <SoftwareSerial.h>
4   #include <TinyGPSPlus.h>
5
6   #include <SPI.h>
7   #include <SD.h>
8
9   #include <Sleep_n0m1.h>
10  #define DEBUG false
11  #define NMEA false
12  #define FILENAME "gps.txt"
13
14  class GpsFix
15  {
16  public:
17    uint32_t sat = 0;
18    float hdop = 0;
19    float lat = 0;
20    float lon = 0;
21    uint32_t fix = 0;
22    TinyGPSDate date;
23    TinyGPSTime time;
24    uint32_t age = 0;
25    float alt = 0;
26    float course = 0;
27    float speed = 0;
28    String card = "";
29    unsigned long timeToFix = 0;
30    uint8_t fixesCount = 0;
31    bool isValid = false;
32    GpsFix()
33    {
34      hdop = 0;
35      isValid = false;
36      fixesCount = 0;
37      timeToFix = 0;
38    };
39    String formatString()
40    {
```

```
41        String result = String(sat);
42        result.concat(";");
43        result.concat(String(hdop, 2));
44        result.concat(";");
45        result.concat(String(lat, 8));
46        result.concat(";");
47        result.concat(String(lon, 8));
48        result.concat(";");
49        result.concat(String(fix));
50        result.concat(";");
51        result.concat(getDateString());
52        result.concat(";");
53        result.concat(getTimeString());
54        result.concat(";");
55        result.concat(String(age));
56        result.concat(";");
57        result.concat(String(course, 2));
58        result.concat(";");
59        result.concat(String(speed, 4));
60        result.concat(";");
61        result.concat(card);
62        result.concat(";");
63        result.concat(String(fixesCount));
64        return result;
65      };
66
67    private:
68      String getDateString()
69      {
70        String result = "";
71        if (date.isValid())
72        {
73          result = String(date.year()) + "-";
74
75          result += date.month() < 10 ? "0" : "";
76          result += String(date.month()) + "-";
77
78          result += date.day() < 10 ? "0" : "";
79          result += String(date.day());
80        }
81        return result;
82      };
83      String getTimeString()
84      {
```

```
85        String result = "";
86        if (time.isValid())
87        {
88          result = time.hour() < 10 ? "0" : "";
89          result += String(time.hour()) + ":";
90
91          result += time.minute() < 10 ? "0" : "";
92          result += String(time.minute()) + ":";
93
94          result += time.second() < 10 ? "0" : "";
95          result += String(time.second());
96        }
97        return result;
98      };
99    };
100
101   // sleep interval in seconds
102   uint32_t INTERVAL_S = 300;
103
104   // GPS
105   const uint8_t IO_GPS_RX = 4;
106   const uint8_t IO_GPS_TX = 3;
107   const uint8_t IO_GPS_SWITCH = 2;
108
109   SoftwareSerial gpsSerial(IO_GPS_RX, IO_GPS_TX); // rx,tx
110   TinyGPSPlus gps;
111
112   const uint32_t TIMEOUT_MS = DEBUG ? 30000 : 300000;
113   Sleep sleeper;
114
115   bool WAITFORIT = false;
116
117   // SD
118   const uint8_t IO_SDCARD = 10;
119   bool withSDCard = true;
120
121   void UBLOX_GPS_Shutdown()
122   {
123     // sends command over serial interface to GPS to put it in PMREQ backup
          mode
124     uint8_t index;
125     uint8_t UBLOX_GPSStandby[] = {0xB5, 0x62, 0x02, 0x41, 0x08, 0x00, 0x00, 0
          x00, 0x00, 0x00, 0x02, 0x00, 0x00, 0x00, 0x4D, 0x3B};
126
```

```
127       for (index = 0; index < sizeof(UBLOX_GPSStandby); index++)
128       {
129         gpsSerial.write(UBLOX_GPSStandby[index]);
130
131     #if DEBUG
132         Serial.print(UBLOX_GPSStandby[index], HEX);
133         Serial.print(" ");
134     #endif
135       }
136     };
137
138     void UBLOX_GPS_Wakeup()
139     {
140       gpsSerial.println(); // send some characters to GPS to wake it up
141     };
142
143     void SDDateTime(uint16_t *date, uint16_t *time)
144     {
145       *date = FAT_DATE(gps.date.year() || 2009, gps.date.month() || 9, gps.date
              .day() || 9);
146       *time = FAT_TIME(gps.time.hour() || 9, gps.time.minute() || 9, gps.time.
              second() || 9);
147     };
148
149     GpsFix getValidSignal(bool forever)
150     {
151     #if DEBUG
152       Serial.println("GetValidSignal");
153     #endif
154       // activate GPS
155       if (digitalRead(IO_GPS_SWITCH) != HIGH)
156       {
157     #if DEBUG
158         Serial.println("WakeUpGPS");
159     #endif
160         digitalWrite(IO_GPS_SWITCH, HIGH);
161       }
162       UBLOX_GPS_Wakeup();
163       uint64_t GPSSTARTTIME_MS = millis(); // time in mS when GPS is scanned fo
              fix
164       uint8_t countValids = 0;
165       uint64_t firstValidTime = 0;
166       char GPSchar;
167       GpsFix bestFix;
```

```
168
169   #if NMEA
170     File nmeaFile = SD.open("nmea.txt", FILE_WRITE);
171   #endif
172
173     while (!bestFix.isValid)
174     {
175       while (gpsSerial.available() > 0)
176       {
177         GPSchar = gpsSerial.read();
178         gps.encode(GPSchar);
179
180   #if NMEA
181   #if DEBUG
182         Serial.print(GPSchar);
183   #endif
184         nmeaFile.print(GPSchar);
185   #endif
186       }
187
188       if (gps.location.isUpdated() && gps.location.isValid() && gps.hdop.
              isUpdated() && gps.hdop.isValid() && gps.hdop.hdop() < 10)
189       {
190   #if DEBUG
191         Serial.println(countValids);
192   #endif
193         countValids++;
194         if (firstValidTime == 0)
195         {
196           firstValidTime = millis();
197         }
198
199         if (!bestFix.hdop || gps.hdop.hdop() < bestFix.hdop)
200         {
201           bestFix.sat = gps.satellites.value();
202           bestFix.hdop = gps.hdop.hdop();
203           bestFix.lat = gps.location.lat();
204           bestFix.lon = gps.location.lng();
205           bestFix.fix = gps.location.age();
206
207           bestFix.age = gps.date.age();
208           bestFix.alt = gps.altitude.meters();
209           bestFix.course = gps.course.deg();
210           bestFix.speed = gps.speed.kmph();
```

```
211           bestFix.card = TinyGPSPlus::cardinal(gps.course.deg());
212
213           bestFix.date = gps.date;
214           bestFix.time = gps.time;
215     #if DEBUG
216           Serial.println("NewBestFix");
217           Serial.println(bestFix.hdop);
218     #endif
219         }
220       }
221
222       if (countValids >= 20)
223       { // try to have 20 signals before logging
224     #if DEBUG
225         Serial.println("20 valid fixes");
226         Serial.println(gps.location.lat(), 6);
227         Serial.println(bestFix.lat, 6);
228         Serial.println(gps.location.lng(), 6);
229         Serial.println(bestFix.lon, 6);
230     #endif
231         bestFix.isValid = true;
232       }
233       // fallback if we dont have further valid signals within 20 seconds
234       if (countValids >= 1 && millis() - firstValidTime >= 20000)
235       {
236         bestFix.isValid = true;
237     #if DEBUG
238         Serial.println("15sec after first fix");
239     #endif
240       }
241
242       if (millis() - GPSSTARTTIME_MS >= TIMEOUT_MS && !forever)
243       {
244         bestFix.isValid = true;
245     #if DEBUG
246         Serial.println("Timeout");
247     #endif
248       }
249     }
250     #if NMEA
251       nmeaFile.close();
252     #endif
253       bestFix.timeToFix = millis() - GPSSTARTTIME_MS;
254       bestFix.fixesCount = countValids;
```

```
255   #if DEBUG
256     Serial.println("We return now");
257   #endif
258     return bestFix;
259   }
260
261   void setup()
262   {
263   #if DEBUG
264     //Serial.begin(9600);
265     Serial.begin(115200);
266     // Serial.begin(57600);
267     Serial.println("DEBUG-MODE: Init");
268   #endif
269
270     // init SDCard on
271     if (withSDCard && SD.begin(IO_SDCARD))
272     {
273       SdFile::dateTimeCallback(SDDateTime);
274   #if DEBUG
275       Serial.println("Init SDCard: CardReader found");
276   #endif
277       // write header if new file is needed
278       if (!SD.exists(FILENAME))
279       {
280   #if DEBUG
281         Serial.println("Init SDCard: add new file");
282   #endif
283         File gpsFile = SD.open(FILENAME, FILE_WRITE);
284         gpsFile.println("Sat;HDOP;Lat;Lon;Fix;Date;Time;Age;Alt;Course;Speed;
              Card;FixesCount");
285         gpsFile.close();
286       }
287     }
288     else
289     {
290   #if DEBUG
291       Serial.println("Init SDCard: Failed - withSDCard -> " + String(
            withSDCard));
292   #endif
293       withSDCard = false;
294     }
295   #if DEBUG
296     Serial.println("Init SDCard: done");
```

```
297   #endif
298
299   // init GPS
300   #if DEBUG
301     Serial.println("Init GPS: Started... with " + String(gps.libraryVersion()
          ));
302   #endif
303     digitalWrite(IO_GPS_SWITCH, HIGH);
304     gpsSerial.begin(9600);
305     UBLOX_GPS_Wakeup();
306     getValidSignal(true);
307   }
308
309   void loop()
310   {
311   #if DEBUG
312     Serial.println("loopStart");
313   #endif
314
315     GpsFix fix = getValidSignal(false);
316   #if DEBUG
317     Serial.println("Write to SD");
318   #endif
319     File gpsFile = SD.open(FILENAME, FILE_WRITE);
320   #if DEBUG
321     if (gpsFile) {
322       Serial.println("File opened");
323     }
324   #endif
325
326   // ShutDownGPS
327   #if DEBUG
328     Serial.println("ShutDownGPS");
329   #endif
330
331     UBLOX_GPS_Shutdown();
332     digitalWrite(IO_GPS_SWITCH, LOW);
333
334     String result = fix.formatString();
335
336   #if DEBUG
337     Serial.println(result);
338   #endif
339
```

```
340       gpsFile.println(result);
341       gpsFile.close();
342
343
344
345       // SleepMode for Microcontroller, i.e. until every minute 5 (14:00,
              14:05, etc.)
346       if (WAITFORIT)
347       {
348         // Sleep 1 day if it's not Juli already
349     #if DEBUG
350       Serial.println("We need to Wait - it is Month" + gps.date.month());
351     #endif
352
353         if (gps.date.month() >= 7)
354         {
355           WAITFORIT = false;
356         }
357       }
358       INTERVAL_S = WAITFORIT ? 86400 : 300;
359
360     #if DEBUG
361       Serial.println(INTERVAL_S);
362       INTERVAL_S = 120;
363     #endif
364
365       uint32_t currentTime = gps.time.minute() * 60 + gps.time.second();
366       uint32_t secondsToInterval = INTERVAL_S - (currentTime % INTERVAL_S);
367       // remove the last timeToFix if its not too long; maybe it takes next
              time approxmatly the same
368       unsigned long msToWakeUp = secondsToInterval * 1000 - (fix.timeToFix <=
              60000 ? fix.timeToFix : 15000);
369
370     #if DEBUG
371       Serial.println("ShutDownMCU for ms: " + String(msToWakeUp));
372     #endif
373       delay(500); // delay to allow serial to fully print before sleep
374       sleeper.pwrDownMode();
375       sleeper.sleepDelay(msToWakeUp);
376     #if DEBUG
377       Serial.println("WakeUpMCU");
378     #endif
379     }
```

D. Messwerte: Vergleich der GPS-Module (Continuous-Mode)

Continous-Mode	TTFF-3D [s]	PDOP	HDOP	mA (max)	mA (min)	mA (ø)
GPS						
Hot						
G-H-1	2,464	2,3	1,3	38,99	34,53	36,76
G-H-2	3,263	2,3	1,3	43,51	35,66	39,585
G-H-3	2,289	2,3	1,3	42,88	35,01	38,945
G-H-4	1,642	2,3	1,3	49,11	34,67	41,89
G-H-5	1,921	5,7	2,8	46,02	36,01	41,015
G-H-6	1,497	5,2	3,4	42,19	34,98	38,585
G-H-7	1,193	3,6	2	43,87	35,47	39,67
G-H-8	1,785	2,3	1,3	40,79	33,99	37,39
G-H-9	1,238	3,3	2	41,65	37,01	39,33
G-H-10	1,87	2,3	1,3	39,87	35,11	37,49
Mittelwert (G-H)	**1,9162**	**3,16**	**1,8**	**42,888**	**35,244**	**39,066**
Warm						
G-W-1	41,351	4,3	2,4	45,54	32,75	39,145
G-W-2	55,851	3,6	2,8	45,89	33,74	39,815
G-W-3	98,683	4,5	3,5	45,48	33,66	39,57
G-W-4	29,019	3,9	2,5	45,46	33,54	39,5
G-W-5	68,335	4,2	2,9	45,72	33,81	39,765
G-W-6	134,31	4,6	2,8	45,22	33,87	39,545
G-W-7	99,431	5,5	2,9	40,34	34,39	37,365
G-W-8	140,884	8,4	7,9	41,04	34,45	37,745
G-W-9	56,005	4,8	2,4	40,17	34,41	37,29
G-W-10	62,1	2,3	1,3	41,38	34,45	37,915
Mittelwert (G-W)	**78,5969**	**4,61**	**3,14**	**43,624**	**33,907**	**38,7655**
Cold						
G-C-1	62,831	2,1	1,4	48,11	36,92	42,515
G-C-2	79,517	7,9	5,1	45,73	38,02	41,875
G-C-3	73,344	4,2	3	46,88	37,81	42,345
G-C-4	58,762	5,7	3,4	44,98	38,88	41,93
G-C-5	65,105	9,1	1,5	46,24	39,01	42,625
G-C-6	76,014	2,8	1,6	46,43	34,22	40,325
G-C-7	58,765	6	4	49,32	36,89	43,105
G-C-8	80,661	4,7	2,8	49,44	38,35	43,895
G-C-9	76,362	3,4	2,2	54,76	37,45	46,105
G-C-10	155,494	3,4	2,2	52,21	37,77	44,99
Mittelwert (G-C)	**78,6855**	**4,93**	**2,72**	**48,41**	**37,532**	**42,971**
Mittelwert (G-HWC)	**53,0662**	**4,23333333**	**2,55333333**	**44,974**	**35,561**	**40,2675**
GPS+GLONASS						
Hot						
GG-H-1	0,52	4	1,9	53,12	46,74	49,93
GG-H-2	0,631	1,6	1	50,51	45,98	48,245
GG-H-3	1,265	2	1,2	49,99	46,88	48,435
GG-H-4	0,438	1,6	1,2	53,71	46,32	50,015
GG-H-5	0,452	1,8	1	52,78	45,87	49,325
GG-H-6	0,478	1,8	1	50,65	46,01	48,33
GG-H-7	0,814	1,8	1	53,13	45,79	49,46

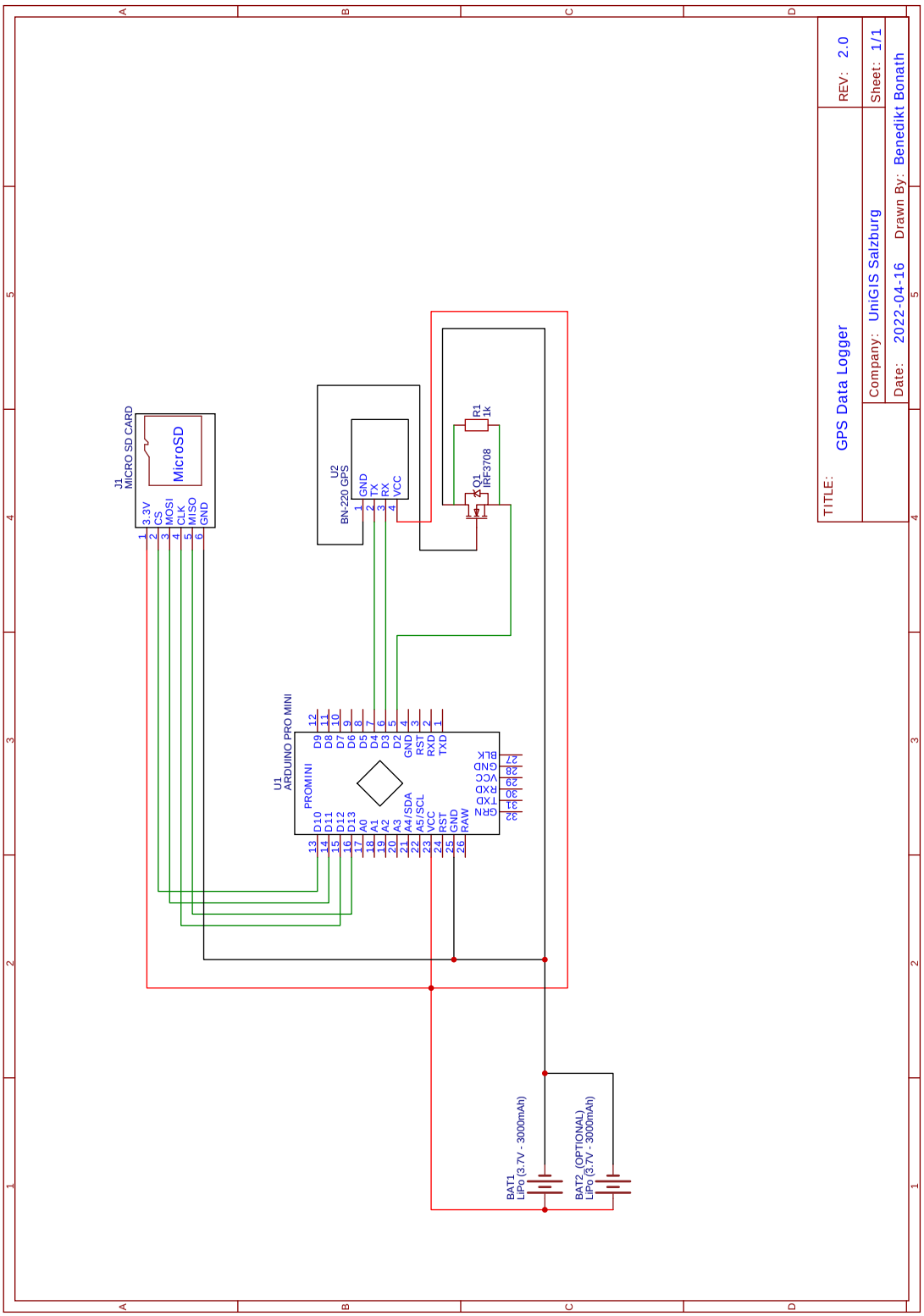
GG-H-8	0,829	1,7	1	51,88	45,55	48,715
GG-H-9	0,432	1,9	1,1	50,69	46,79	48,74
GG-H-10	0,552	1,8	1,1	54,11	47,01	50,56
Mittelwert (GG-H)	**0,6411**	**2**	**1,15**	**52,057**	**46,294**	**49,1755**
Warm						
GG-W-1	35,689	2	1	56,38	44,85	50,615
GG-W-2	36,95	8	3,1	54,88	47,12	51
GG-W-3	39,142	5	3	56,1	42,85	49,475
GG-W-4	89,659	4,9	2,5	56,86	44,03	50,445
GG-W-5	100,637	5,7	3,3	57,11	43,91	50,51
GG-W-6	123,703	5,7	2,8	58,05	43,61	50,83
GG-W-7	133,244	7,4	4,6	56,06	44,16	50,11
GG-W-8	74,673	2,4	1,4	54,32	44,92	49,62
GG-W-9	52,7	13	10,3	53,45	45,41	49,43
GG-W-10	60,945	9,2	4,7	53,03	44,99	49,01
Mittelwert (GG-W)	**74,7342**	**6,33**	**3,67**	**55,624**	**44,585**	**50,1045**
Cold						
GG-C-1	67,7	3,14	2	57,15	42,85	50
GG-C-2	62,95	12,1	10,7	56,78	43,05	49,915
GG-C-3	42,014	1,6	0,8	57,71	42,56	50,135
GG-C-4	46,918	2,1	1	55,81	44,35	50,08
GG-C-5	31,238	2,5	1,6	58,91	46,4	52,655
GG-C-6	62,877	2,5	1,4	57,69	43,44	50,565
GG-C-7	37,478	3,8	2,5	60,57	46,56	53,565
GG-C-8	38,381	4,4	2,7	59,95	44,35	52,15
GG-C-9	41,56	5	3,2	58,13	42,78	50,455
GG-C-10	36,376	4,2	2,1	57,66	46	51,83
Mittelwert (GG-C)	**46,7492**	**4,134**	**2,8**	**58,036**	**44,234**	**51,135**
Mittelwert (GG-HWC)	**40,7081667**	**4,15466667**	**2,54**	**55,239**	**45,0376667**	**50,1383333**
GPS+GLONASS+Galileo						
Hot						
GGG-H-1	0,422	1,7	0,8	56,16	46,11	51,135
GGG-H-2	0,661	2,1	1,4	54,97	45,42	50,195
GGG-H-3	0,842	2,8	1,6	60,99	49,07	55,03
GGG-H-4	0,813	2,2	1,5	55,55	46,93	51,24
GGG-H-5	0,646	2,2	1,5	56,12	49,25	52,685
GGG-H-6	1,004	1,7	0,8	57,6	47,22	52,41
GGG-H-7	1,041	2,2	0,9	56,12	48,02	52,07
GGG-H-8	0,906	1,7	1,4	56,71	49,93	53,32
GGG-H-9	0,838	2,7	1,6	56,63	46,75	51,69
GGG-H-10	0,842	2,1	1,5	59,81	47,63	53,72
Mittelwert (GGG-H)	**0,8015**	**2,14**	**1,3**	**57,066**	**47,633**	**52,3495**
Warm						
GGG-W-1	58,361	16,5	11,1	59,48	46,46	52,97
GGG-W-2	68,982	7,7	4	60,99	47,12	54,055
GGG-W-3	72,15	7,7	4,1	58,57	49,68	54,125
GGG-W-4	56,117	18,4	10,2	60,03	50,36	55,195
GGG-W-5	81,292	4,3	3,4	59,34	45,24	52,29

GGG-W-6	61,935	10,4	8,6	60,99	46,47	53,73
GGG-W-7	58,401	5,7	3,3	60,99	47,78	54,385
GGG-W-8	49,613	5,4	3,8	57,39	47,36	52,375
GGG-W-9	44,425	7,3	2,4	58	48,01	53,005
GGG-W-10	35,707	2		60,73	47,92	54,325
Mittelwert (GGG-W)	**58,6983**	**8,54**	**5,65555556**	**59,651**	**47,64**	**53,6455**
Cold						
GGG-C-1	62,149	22,4	11,4	60,99	45,81	53,4
GGG-C-2	42,211	4,5	3,6	60,99	45,9	53,445
GGG-C-3	52,541	11,3	4,4	60,99	43,29	52,14
GGG-C-4	62,514	5,2	2,9	60,86	44,48	52,67
GGG-C-5	51,112	8,6	6,6	55,16	43,84	49,5
GGG-C-6	67,289	2,9	1,5	60,99	43,44	52,215
GGG-C-7	79,154	20,4	11,3	59,97	43,32	51,645
GGG-C-8	59,335	9,8	9,2	60,99	46,09	53,54
GGG-C-9	46,675	3,2	1,6	60,99	42,59	51,79
GGG-C-10	56,773	11,4	6,5	60,99	46,36	53,675
Mittelwert (GGG-C)	**57,9753**	**9,97**	**5,9**	**60,292**	**44,512**	**52,402**
Mittelwert (GGG-HWC)	**39,1583667**	**6,88333333**	**4,28518519**	**59,003**	**46,595**	**52,799**

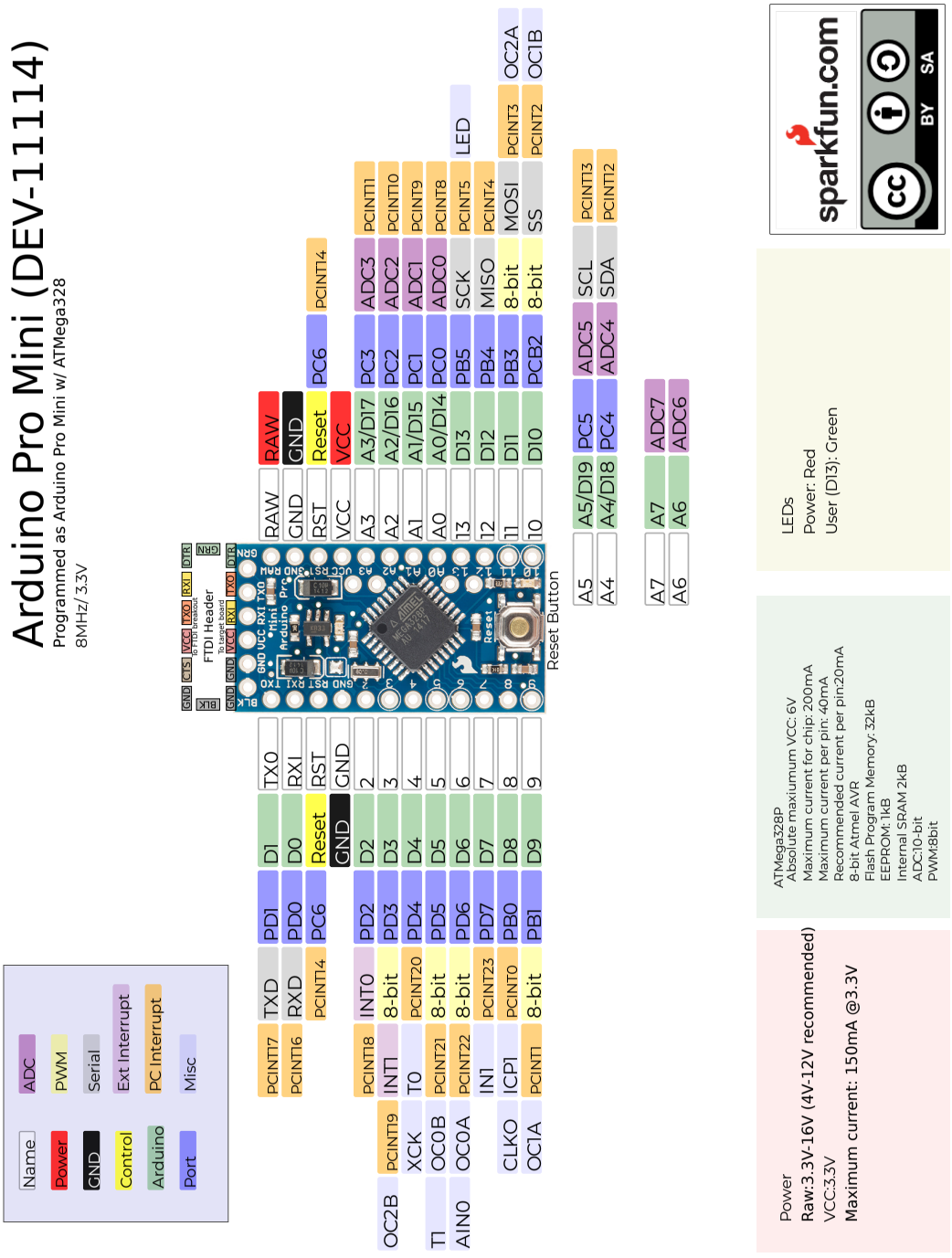
E. Messwerte: Vergleich der GPS-Module (PowerSave-Mode)

PowerSafe-Mode (GPS-GLONASS; Warm)	TTFF-3D [s]	PDOP	HDOP	mA (max)	mA (min)	mA (ø)
PS-GG-W-1	34,036	5,940	4,400	47,12	12,1	29,61
PS-GG-W-2	55,124	7,700	5,010	50,19	12,02	31,105
PS-GG-W-3	27,009	5,600	3,400	50,92	11,89	31,405
PS-GG-W-4	48,871	4,500	3,300	60,28	11,9	36,09
PS-GG-W-5	31,817	7,300	4,300	46,82	12,83	29,825
PS-GG-W-6	59,426	2,700	1,300	51,9	12,03	31,965
PS-GG-W-7	35,009	7,500	2,400	40,96	12,54	26,75
PS-GG-W-8	44,911	5,340	3,700	44,01	17,01	30,51
PS-GG-W-9	37,880	5,700	4,100	50,52	13,22	31,87
PS-GG-W-10	58,052	2,300	1,500	61,3	14,37	37,835
Mittelwert (PS-GG-W)	**43,214**	**5,458**	**3,341**	**50,402**	**12,991**	**31,697**

F. Schaltplan: GPS-Datenlogger



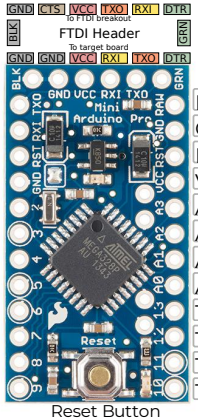
G. Datasheet: Arduino Pro Mini 8MHz / 16 MHz



Arduino Pro Mini (DEV-11113)

Programmed as Arduino Pro Mini w/ ATMega328
16MHz/ 5V

Name	ADC
Power	PWM
GND	Serial
Control	Ext Interrupt
Arduino	PC Interrupt
Port	Misc



Reset Button

		PCINT17	TXD	PD1	D1	TX0	RAW	RAW					
		PCINT16	RXD	PD0	D0	RXI	GND	GND					
			PCINT14	PC6	Reset	RST	RST	Reset	PC6	PCINT14			
					GND	GND	VCC	VCC					
		PCINT18	INT0	PD2	D2	2	A3	A3/D17	PC3	ADC3	PCINT11		
OC2B	PCINT19	INT1	8-bit	PD3	D3	3	A2	A2/D16	PC2	ADC2	PCINT10		
	XCK	T0	PCINT20	PD4	D4	4	A1	A1/D15	PC1	ADC1	PCINT9		
T1	OC0B	PCINT21	8-bit	PD5	D5	5	A0	A0/D14	PC0	ADC0	PCINT8		
AIN0	OC0A	PCINT22	8-bit	PD6	D6	6	13	D13	PB5	SCK	PCINT5	LED	
		IN1	PCINT23	PD7	D7	7	12	D12	PB4	MISO	PCINT4		
	CLKO	ICP1	PCINT0	PB0	D8	8	11	D11	PB3	8-bit	MOSI	PCINT3	OC2A
	OC1A	PCINT1	8-bit	PB1	D9	9	10	D10	PCB2	8-bit	SS	PCINT2	OC1B

A5	A5/D19	PC5	ADC5	SCL	PCINT13	
A4	A4/D18	PC4	ADC4	SDA	PCINT12	
A7	A7	ADC7				
A6	A6	ADC6				

Power
Raw:5V-16V (6V-12V recommended)
VCC:5V
Maximum current: 150mA @5V

ATMega328P
Absolute maxiumum VCC: 6V
Maximum current for chip: 200mA
Maximum current per pin: 40mA
Recommended current per pin:20mA
8-bit Atmel AVR
Flash Program Memory: 32kB
EEPROM: 1kB
Internal SRAM 2kB
ADC:10-bit
PWM:8bit

LEDs
Power: Red
User (D13): Green

sparkfun.com

Benedikt Bonath | u106712

H. Datasheet: BN220 GPS

BN-220 GPS Module + Antenna Datasheet

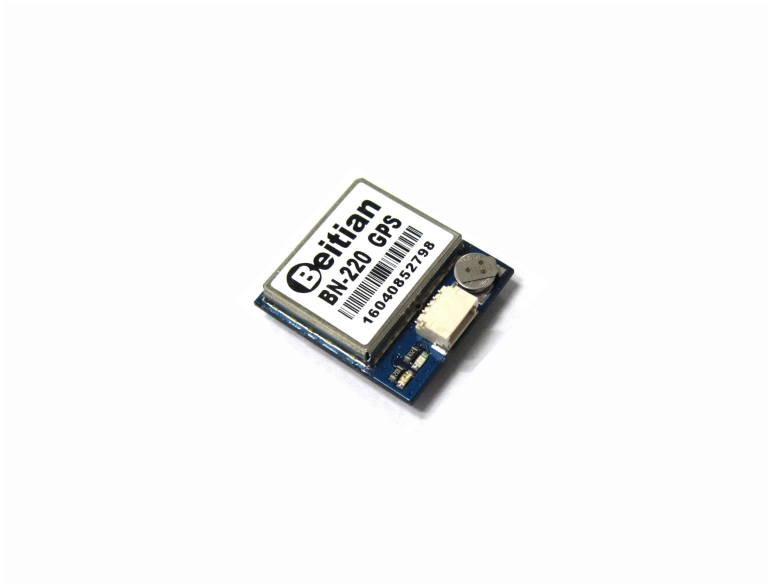
BN-220

GPS Module + Antenna

DataSheet

Revision: 5.0

Date:2015.10



BN-220 GPS Module + Antenna Datasheet

Features:

Parameter	Specification	
Electrical Characteristics	Chipset	u-blox M8030-KT
	Receiving Format	GPS,GLONASS,Galileo,BeiDou,QZSS and SBAS
	Frequency	GPS L1，GLONASS L1，BeiDou B1，SBAS L1，Galileo E1
	Channels	72 Searching Channel
Sensitivity	Tracking	-167dBm
	Reacquisition	-160dBm
	Cold start	-148dBm
	Hot start	-156dBm
Accuracy	Position Horizontal	2.0 m CEP 2D RMS SBAS Enable (Typical Open Sky)
	Velocity	0.1m/sec 95% (SA off)
	Timing	1us synchronized to GPS time
Acquisition Time	Cold Start	26s
	Warm start	25s
	Hot start	1s
Data and Update Rate	Support Rate	4800bps to 921600bps,Default 9600bps
	Data Level	TTL or RS-232,Default TTL level
	Data Protocol	NMEA-0183 or UBX, Default NMEA-0183
	Single GNSS	1Hz-18Hz
	Concurrent GNSS	1Hz-10Hz,Default 1Hz
Operational Limits	Altitude	50,000m Max
	Velocity	515m/s Max
	Acceleration	Less than 4g
Power consumption	VCC	DC Voltage 3.0V-5.5V,Typical: 5.0V
	Current	Capture 50mA@5.0V
Mechanical Specifications	Dimension	22mm*20mm*6mm
	Weight	5.3g
	Connector	1.00mm spacing between the 4pins patch seat
Environment	Operating temp	-40 °C ~ +85°C
	Storage Temp	-40°C ~ +105°C
LED	built-in LED	TX LED:blue.The data output, TX LED flashing
		PPS LED:red.PPS LED not bright when GPS not fixed,flashing when fixed

BN-220 GPS Module + Antenna Datasheet

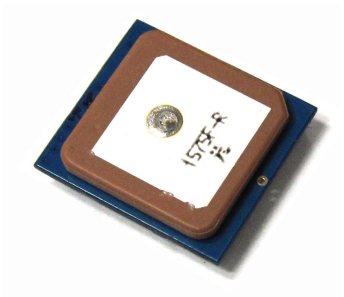
Pin Description：

PIN	PIN Name	I/O	Description
1	GND	G	Ground
2	TX	O	Serial Data Output.
3	RX	I	Serial Data Input.
4	VCC	I	DC 3.0V - 5.5V supply input,Typical: 5.0V

LED：

1.TX LED:blue.The data output, TX LED flashing
2.PPS LED:red.PPS LED not bright when GPS not fixed， flashing when fixed.

Rear view：



BN-220 GPS Module + Antenna Datasheet

Message Structure:

$xxGGA,time,lat,NS,long,EW,quality,numSV,HDOP,alt,M,sep,M,diffAge,diffStation*cs<CR><LF>

Example:

$GPGGA,092725.00,4717.11399,N,00833.91590,E,1,08,1.01,499.6,M,48.0,M,,*5B

Field No	Name	Unit	Format	Example	Description
0	xxGGA	-	string	$GPGGA	GGA Message ID (xx = current Talker ID)
1	time	-	hhmmss.ss	092725.00	UTC time
2	lat	-	ddmm.mmmmm	4717.11399	Latitude (degrees & minutes)
3	NS	-	character	N	North/South indicator
4	long	-	dddmm.mmmmm	00833.91590	Longitude (degrees & minutes)
5	EW	-	character	E	East/West indicator
6	quality	-	digit	1	0:No Fix / Invalid 1:Standard GPS (2D/3D) 2:Differential GPS 6:Estimated (DR) Fix
7	numSV	-	numeric	08	Number of satellites used
8	HDOP	-	numeric	1.01	Horizontal Dilution of Precision
9	alt	m	numeric	499.6	Altitude above mean sea level
10	uAlt	-	character	M	Altitude units: meters (fixed field)
11	sep	m	numeric	48.0	Geoid separation: difference between geoid and mean sea level
12	uSep	-	character	M	Separation units: meters (fixed field)
13	diffAge	s	numeric	-	Age of differential corrections (blank when DGPS is not used)
14	diffStation	-	numeric	-	ID of station providing differential corrections (blank when DGPS is not used)
15	cs	-	hexadecimal	*5B	Checksum
16	<CR><LF>	-	character	-	Carriage return and line feed

Message Structure:

$xxGLL,lat,NS,long,EW,time,status,posMode*cs<CR><LF>

Example:

$GPGLL,4717.11364,N,00833.91565,E,092321.00,A,A*6

Field No	Name	Unit	Format	Example	Description
0	xxGLL	-	string	$GPGLL	GLL Message ID (xx = current Talker ID)
1	lat	-	ddmm.mmmmm	4717.11364	Latitude (degrees & minutes)
2	NS	-	character	N	North/South indicator
3	long	-	dddmm.mmmmm	00833.91565	Longitude (degrees & minutes)

4	EW	-	character	E	East/West indicator
5	time	-	hhmmss.ss	092321.00	UTC time
6	status	-	character	A	V = Data invalid or receiver warning, A = Data valid
7	posMode	-	character	A	Positioning mode
8	cs	-	hexadecimal	*60	Checksum
9	<CR><LF>	-	character	-	Carriage return and line feed

Message Structure:

$xxGSA,opMode,navMode{,sv},PDOP,HDOP,VDOP,systemId*cs<CR><LF>

Example:

$GPGSA,A,3,23,29,07,08,09,18,26,28,,,,,1.94,1.18,1.54,1*0D

Field No	Name	Unit	Format	Example	Description
0	xxGSA	-	string	$GPGSA	GSA Message ID (xx = current Talker ID)
1	opMode	-	character	A	Operation mode M:Manually set to operate in 2D or 3D mode A:Automatically switching between 2D or 3D mode
2	navMode	-	digit	3	Navigation mode 1:Fix not available 2:2D Fix 3:3D Fix
Start of repeated block (12 times)					
3 + 1*N	sv	-	numeric	29	Satellite number
End of repeated block					
15	PDOP	-	numeric	1.94	Position dilution of precision
16	HDOP	-	numeric	1.18	Horizontal dilution of precision
17	VDOP	-	numeric	1.54	Vertical dilution of precision
18	systemId	-	numeric	1	NMEA defined GNSS System ID NMEA v4.1 and above only
19	cs	-	hexadecimal	*0D	Checksum
20	<CR><LF>	-	character	-	Carriage return and line feed

Message Structure:

$xxGSV,numMsg,msgNum,numSV,{,sv,elv,az,cno},signalId*cs<CR><LF>

Example:

$GPGSV,3,1,10,23,38,230,44,29,71,156,47,07,29,116,41,08,09,081,36,0*7F

$GPGSV,3,2,10,10,07,189,,05,05,220,,09,34,274,42,18,25,309,44,0*72

$GPGSV,3,3,10,26,82,187,47,28,43,056,46,0*7

BN-220 GPS Module + Antenna Datasheet

Field No	Name	Unit	Format	Example	Description
0	xxGSV	-	string	$GPGSV	GSV Message ID (xx = GSV Talker ID)
1	numMsg	-	digit	3	Number of messages, total number of GSV messages being output
2	msgNum	-	digit	1	Number of this message
3	numSV	-	numeric	10	Number of satellites in view
Start of repeated block (1..4 times)					
4 + 4*N	SV	-	numeric	23	Satellite ID
5 + 4*N	elv	deg	numeric	38	Elevation (range 0-90)
6 + 4*N	az	deg	numeric	230	Azimuth, (range 0-359)
7 + 4*N	cno	dBH	numeric	44	Signal strength (C/N0, range 0-99), blank when not tracking
End of repeated block					
5.. 16	signalId	-	numeric	0	NMEA defined GNSS Signal ID (0 = All signals) NMEA v4.1 and above only
6.. 16	cs	-	hexadecimal	*7F	Checksum
7.. 16	<CR><LF>	-	character	-	Carriage return and line feed

Message Structure:

$xxRMC,time,status,lat,NS,long,EW,spd,cog,date,mv,mvEW,posMode,navStatus*cs<CR><LF>

Example:

$GPRMC,083559.00,A,4717.11437,N,00833.91522,E,0.004,77.52,091202,,,A,V*57

Field No	Name	Unit	Format	Example	Description
0	xxRMC	-	string	$GPRMC	RMC Message ID (xx = current Talker ID)
1	time	-	hhmmss.ss	083559.00	UTC time, see note on UTC representation
2	status	-	character	A	Status V:Navigation receiver warning A :Data valid, see position fix flags description
3	lat	-	ddmm.mmmmm	4717.11437	Latitude (degrees & minutes), see format description
4	NS	-	character	N	North/South indicator
5	long	-	dddmm.mmmmm	00833.91522	Longitude (degrees & minutes), see format description
6	EW	-	character	E	East/West indicator

BN-220 GPS Module + Antenna Datasheet

7	spd	Knos	numeric	0.004	Speed over ground
8	cog	degr	numeric	77.52	Course over ground
9	date	-	ddmmyy	091202	Date in day, month, year format, see note on UTC representation
10	mv	degrees	numeric	-	Magnetic variation value (blank - not supported)
11	mvEW	-	character	-	Magnetic variation E/W indicator (blank - not supported)
12	posMode	-	character	-	Mode Indicator, see position fix flags
13	navStatus	-	character	V	Navigational status indicator (V = Equipment is not providing navigational status information)
14	cs	-	hexadecimal	*57	Checksum
15	<CR><LF>	-	character	-	Carriage return and line feed

Message Structure:

$xxVTG,cogt,T,cogm,M,knots,N,kph,K,posMode*cs<CR><LF>

Example:

$GPVTG,77.52,T,,M,0.004,N,0.008,K,A*06

Field No	Name	Unit	Format	Example	Description
0	xxVTG	-	string	$GPVTG	VTG Message ID (xx = current Talker ID)
1	cogt	degrees	numeric	77.52	Course over ground (true)
2	T	-	character	T	Fixed field: true
3	cogm	degrees	numeric	-	Course over ground (magnetic), not output
4	M	-	character	M	Fixed field: magnetic
5	knots	knots	numeric	0.004	Speed over ground
6	N	-	character	N	Fixed field: knots
7	kph	km/	numeric	0.008	Speed over ground
8	K	-	character	K	Fixed field: kilometers per hour
9	posMode	-	character	A	Mode Indicator, see position fix flags description
10	cs	-	hexadecimal	*06	Checksum
11	<CR><LF>	-	character	-	Carriage return and line feed

I. Datasheet: IRF3708 MOSFET

International
IOR Rectifier

SMPS MOSFET

PD - 93938B

IRF3708
IRF3708S
IRF3708L

HEXFET® Power MOSFET

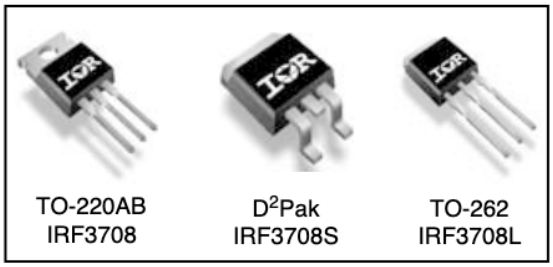
V_{DSS}	$R_{DS(on)}$ max	I_D
30V	12mΩ	62A

Applications

- High Frequency DC-DC Isolated Converters with Synchronous Rectification for Telecom and Industrial Use
- High Frequency Buck Converters for Computer Processor Power

Benefits

- Ultra-Low Gate Impedance
- Very Low $R_{DS(on)}$ at 4.5V V_{GS}
- Fully Characterized Avalanche Voltage and Current



TO-220AB IRF3708 | D^2Pak IRF3708S | TO-262 IRF3708L

Absolute Maximum Ratings

Symbol	Parameter	Max.	Units
V_{DS}	Drain-Source Voltage	30	V
V_{GS}	Gate-to-Source Voltage	±12	V
I_D @ T_C = 25°C	Continuous Drain Current, V_{GS} @ 10V	62	A
I_D @ T_C = 70°C	Continuous Drain Current, V_{GS} @ 10V	52	
I_{DM}	Pulsed Drain Current①	248	
P_D @T_C = 25°C	Maximum Power Dissipation③	87	W
P_D @T_C = 70°C	Maximum Power Dissipation③	61	W
	Linear Derating Factor	0.58	W/°C
T_J , T_{STG}	Junction and Storage Temperature Range	-55 to + 175	°C

Thermal Resistance

	Parameter	Typ.	Max.	Units
$R_{\theta JC}$	Junction-to-Case	—	1.73	°C/W
$R_{\theta CS}$	Case-to-Sink, Flat, Greased Surface ④	0.50	—	
$R_{\theta JA}$	Junction-to-Ambient④	—	62	
$R_{\theta JA}$	Junction-to-Ambient (PCB mount)*	—	40	

* When mounted on 1" square PCB (FR-4 or G-10 Material) .
For recommended footprint and soldering techniques refer to application note #AN-994

Notes ① through ④ are on page 10

IRF3708/3708S/3708L

International
IOR Rectifier

Static @ T_J = 25°C (unless otherwise specified)

	Parameter	Min.	Typ.	Max.	Units	Conditions
$V_{(BR)DSS}$	Drain-to-Source Breakdown Voltage	30	—	—	V	V_{GS} = 0V, I_D = 250μA
$\Delta V_{(BR)DSS}/\Delta T_J$	Breakdown Voltage Temp. Coefficient	—	0.028	—	V/°C	Reference to 25°C, I_D = 1mA
$R_{DS(on)}$	Static Drain-to-Source On-Resistance	—	8	12.0	mΩ	V_{GS} = 10V, I_D = 15A ③
		—	9.5	13.5		V_{GS} = 4.5V, I_D = 12A ③
		—	14.5	29		V_{GS} = 2.8V, I_D = 7.5A ③
$V_{GS(th)}$	Gate Threshold Voltage	0.6	—	2.0	V	V_{DS} = V_{GS}, I_D = 250μA
I_{DSS}	Drain-to-Source Leakage Current	—	—	20	μA	V_{DS} = 24V, V_{GS} = 0V
		—	—	100		V_{DS} = 24V, V_{GS} = 0V, T_J = 125°C
I_{GSS}	Gate-to-Source Forward Leakage	—	—	200	nA	V_{GS} = 12V
	Gate-to-Source Reverse Leakage	—	—	-200		V_{GS} = -12V

Dynamic @ T_J = 25°C (unless otherwise specified)

Symbol	Parameter	Min.	Typ.	Max.	Units	Conditions
g_{fs}	Forward Transconductance	49	—	—	S	V_{DS} = 15V, I_D = 50A
Q_g	Total Gate Charge	—	24	—	nC	I_D = 24.8A
Q_{gs}	Gate-to-Source Charge	—	6.7	—		V_{DS} = 15V
Q_{gd}	Gate-to-Drain ("Miller") Charge	—	5.8	—		V_{GS} = 4.5V ③
Q_{oss}	Output Gate Charge	—	14	21		V_{GS} = 0V, I_D = 24.8A, V_{DS} = 15V
$t_{d(on)}$	Turn-On Delay Time	—	7.2	—	ns	V_{DD} = 15V
t_r	Rise Time	—	50	—		I_D = 24.8A
$t_{d(off)}$	Turn-Off Delay Time	—	17.6	—		R_G = 0.6Ω
t_f	Fall Time	—	3.7	—		V_{GS} = 4.5V ③
C_{iss}	Input Capacitance	—	2417	—	pF	V_{GS} = 0V
C_{oss}	Output Capacitance	—	707	—		V_{DS} = 15V
C_{rss}	Reverse Transfer Capacitance	—	52	—		f = 1.0MHz

Avalanche Characteristics

Symbol	Parameter	Typ.	Max.	Units
E_{AS}	Single Pulse Avalanche Energy②	—	213	mJ
I_{AR}	Avalanche Current①	—	62	A

Diode Characteristics

Symbol	Parameter	Min.	Typ.	Max.	Units	Conditions
I_S	Continuous Source Current (Body Diode)	—	—	62	A	MOSFET symbol showing the integral reverse p-n junction diode.
I_{SM}	Pulsed Source Current (Body Diode) ①	—	—	248		
V_{SD}	Diode Forward Voltage	—	0.88	1.3	V	T_J = 25°C, I_S = 31A, V_{GS} = 0V ③
		—	0.80	—		T_J = 125°C, I_S = 31A, V_{GS} = 0V ③
t_{rr}	Reverse Recovery Time	—	41	62	ns	T_J = 25°C, I_F = 31A, V_R=20V
Q_{rr}	Reverse Recovery Charge	—	64	96	nC	di/dt = 100A/μs ③
t_{rr}	Reverse Recovery Time	—	43	65	ns	T_J = 125°C, I_F = 31A, V_R=20V
Q_{rr}	Reverse Recovery Charge	—	70	105	nC	di/dt = 100A/μs ③

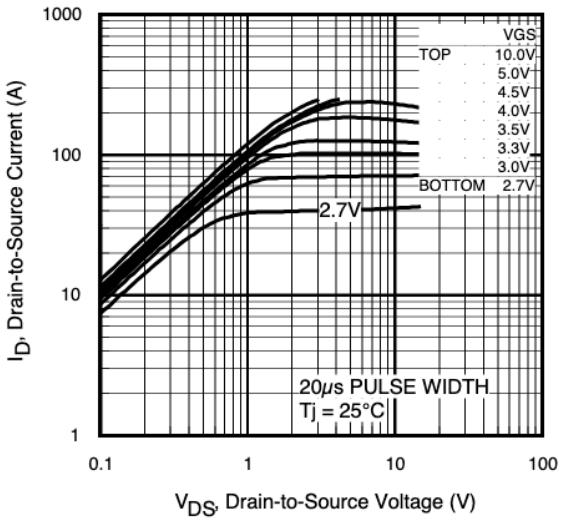


Fig 1. Typical Output Characteristics

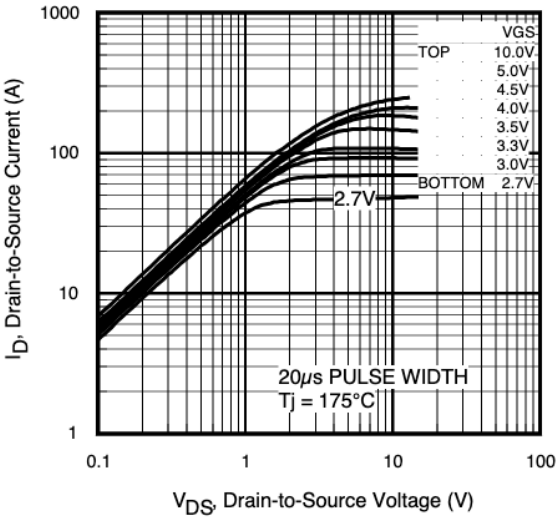


Fig 2. Typical Output Characteristics

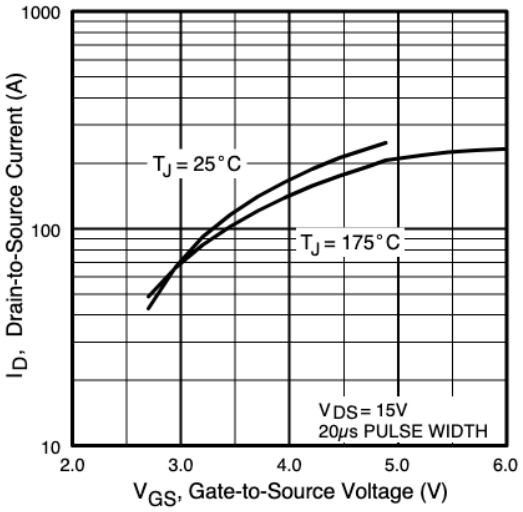


Fig 3. Typical Transfer Characteristics

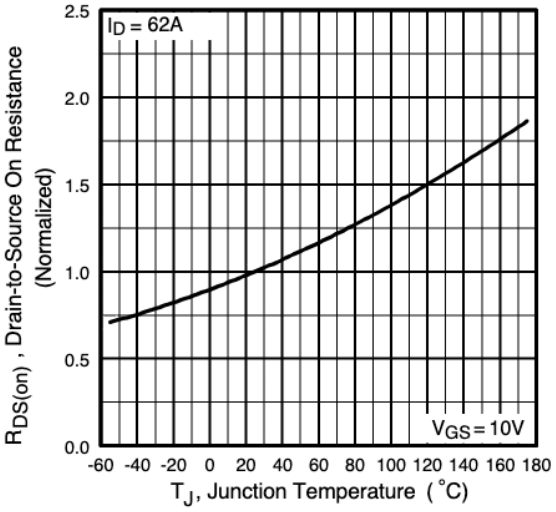


Fig 4. Normalized On-Resistance Vs. Temperature

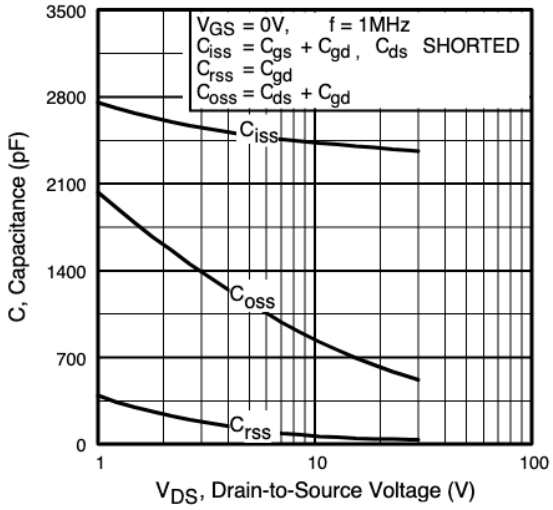


Fig 5. Typical Capacitance Vs. Drain-to-Source Voltage

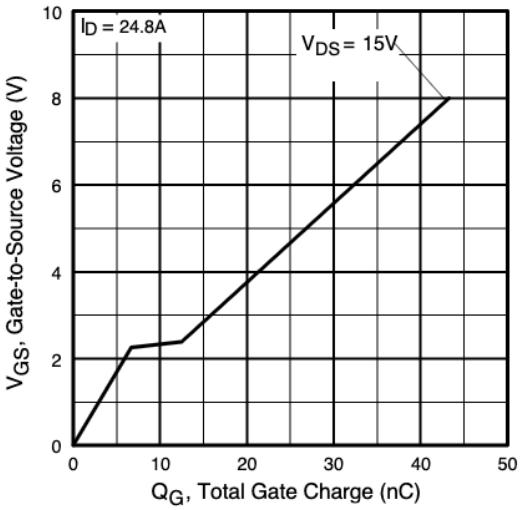


Fig 6. Typical Gate Charge Vs. Gate-to-Source Voltage

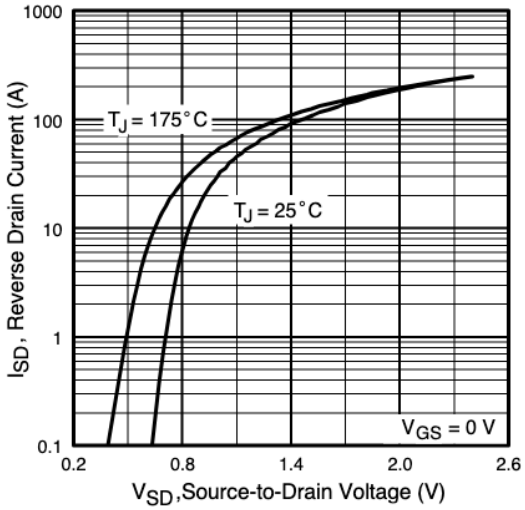


Fig 7. Typical Source-Drain Diode Forward Voltage

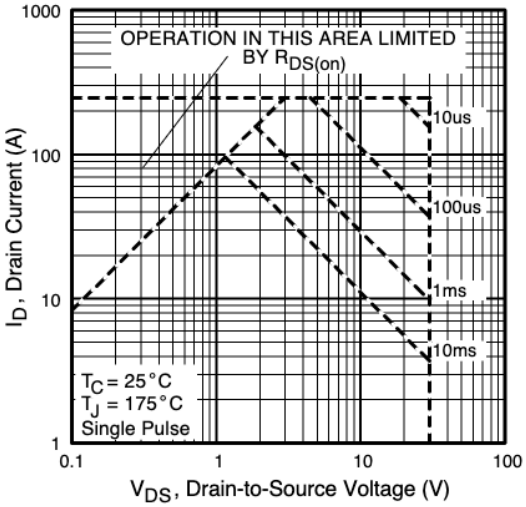


Fig 8. Maximum Safe Operating Area

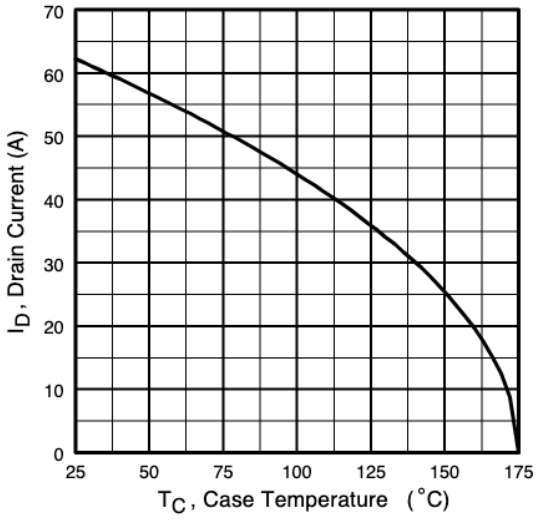


Fig 9. Maximum Drain Current Vs. Case Temperature

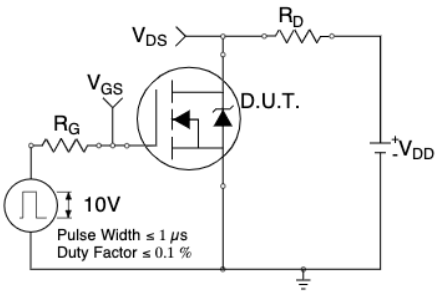


Fig 10a. Switching Time Test Circuit

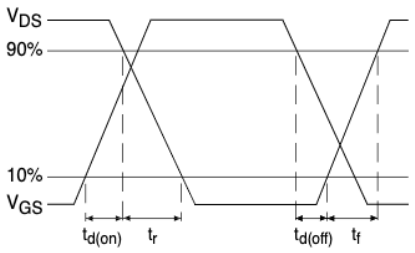


Fig 10b. Switching Time Waveforms

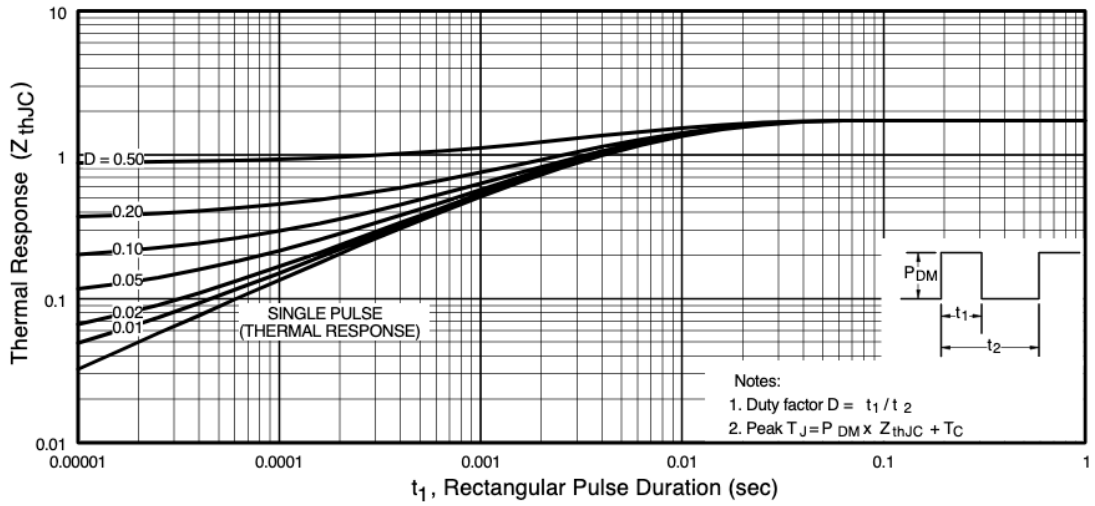


Fig 11. Maximum Effective Transient Thermal Impedance, Junction-to-Case

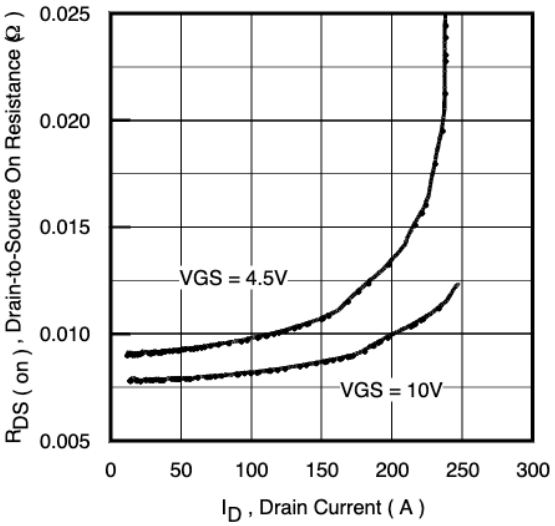


Fig 12. On-Resistance Vs. Drain Current

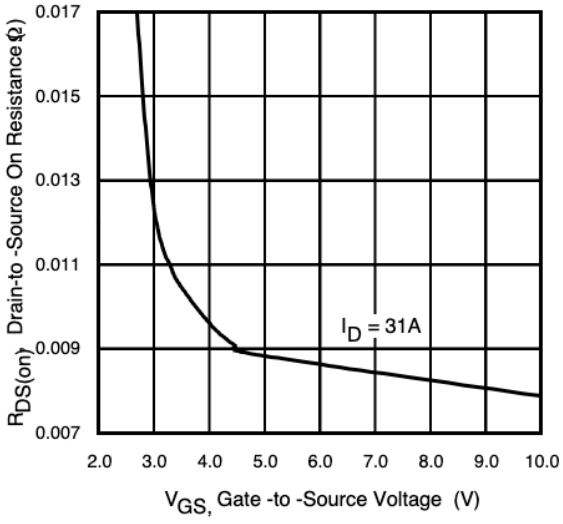


Fig 13. On-Resistance Vs. Gate Voltage

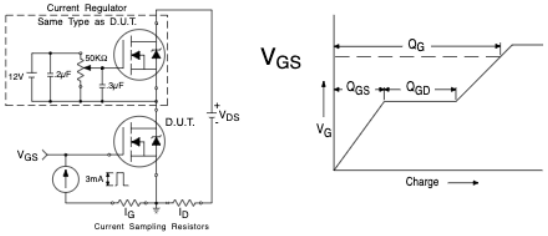


Fig 14a&b. Gate Charge Test Circuit and Waveform

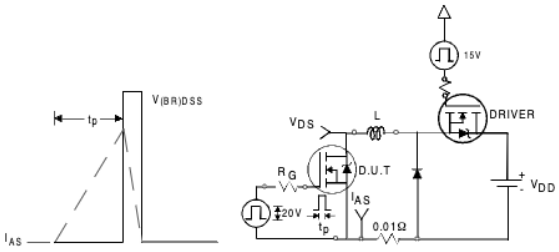


Fig 15a&b. Unclamped Inductive Test circuit and Waveforms

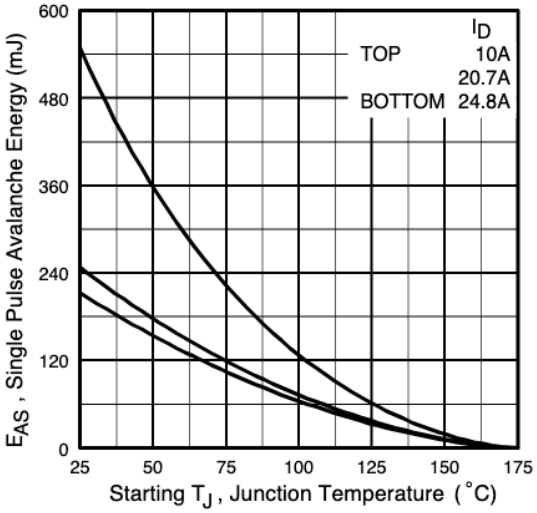


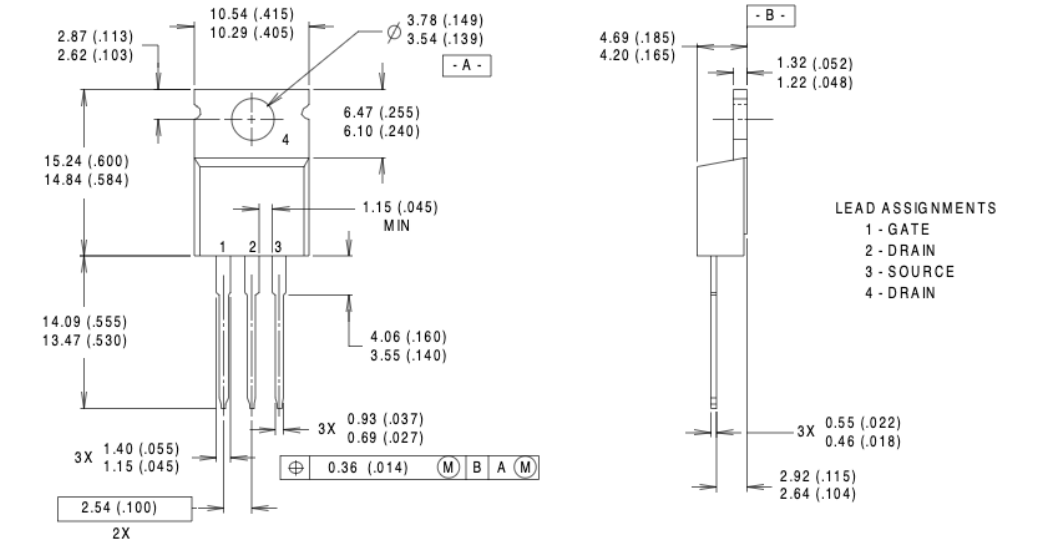
Fig 15c. Maximum Avalanche Energy Vs. Drain Current

International
IOR Rectifier

IRF3708/3708S/3708L

TO-220AB Package Outline

Dimensions are shown in millimeters (inches)

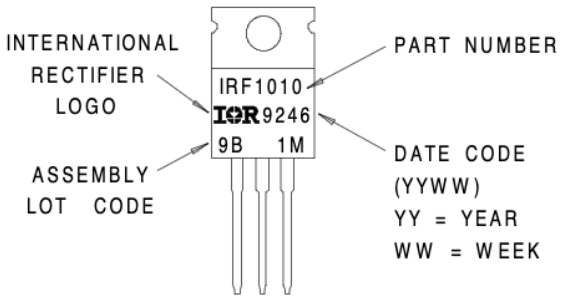


NOTES:

1 DIMENSIONING & TOLERANCING PER ANSI Y14.5M, 1982.
2 CONTROLLING DIMENSION : INCH
3 OUTLINE CONFORMS TO JEDEC OUTLINE TO-220AB.
4 HEATSINK & LEAD MEASUREMENTS DO NOT INCLUDE BURRS.

TO-220AB Part Marking Information

EXAMPLE : THIS IS AN IRF1010
WITH ASSEMBLY
LOT CODE 9B1M

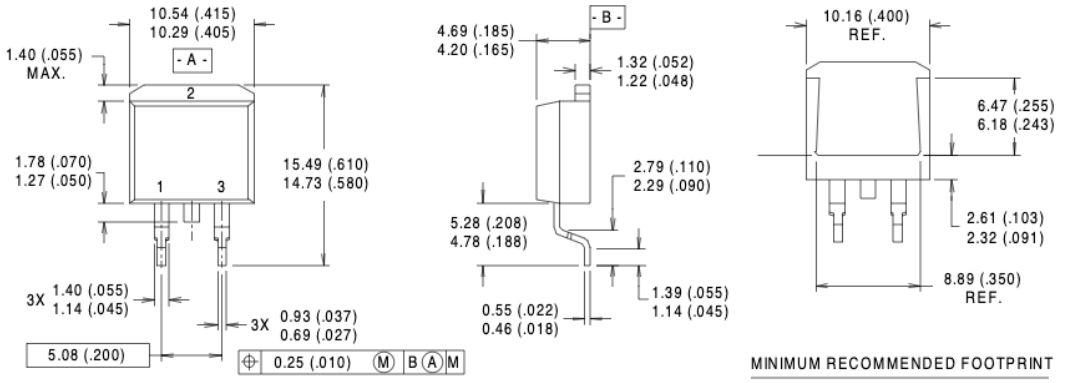


IRF3708/3708S/3708L

International
IOR Rectifier

D²Pak Package Outline

Dimensions are shown in millimeters (inches)



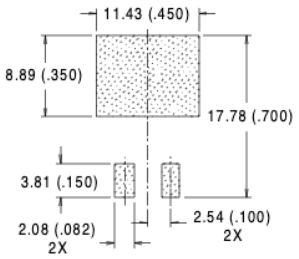
NOTES:

1 DIMENSIONS AFTER SOLDER DIP.
2 DIMENSIONING & TOLERANCING PER ANSI Y14.5M, 1982.
3 CONTROLLING DIMENSION : INCH.
4 HEATSINK & LEAD DIMENSIONS DO NOT INCLUDE BURRS.

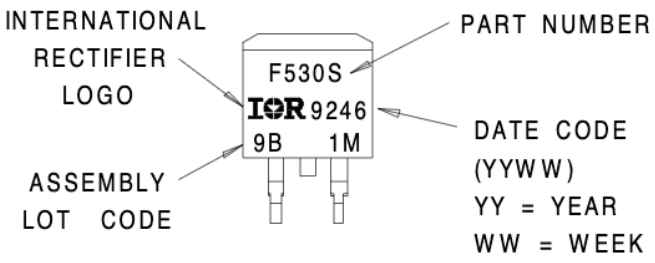
LEAD ASSIGNMENTS

1 - GATE
2 - DRAIN
3 - SOURCE

MINIMUM RECOMMENDED FOOTPRINT



D²Pak Part Marking Information

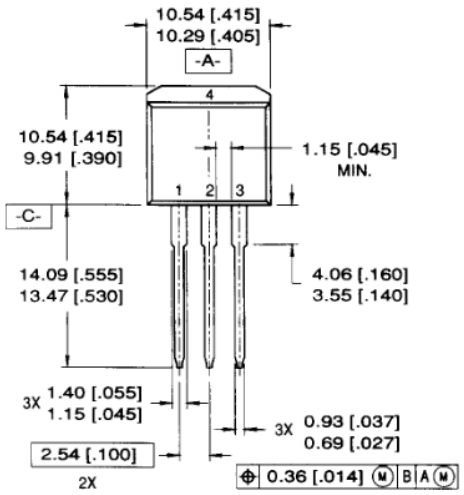


International
IOR Rectifier

IRF3708/3708S/3708L

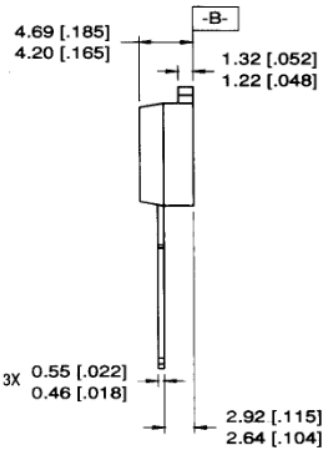
TO-262 Package Outline

Dimensions are shown in millimeters (inches)



LEAD ASSIGNMENTS

1 = GATE 3 = SOURCE
2 = DRAIN 4 = DRAIN

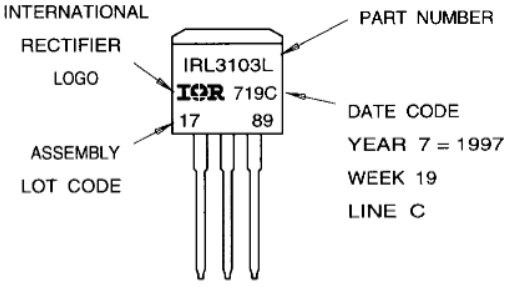


NOTES:

1. DIMENSIONING & TOLERANCING PER ANSI Y14.5M-1982
2. CONTROLLING DIMENSION: INCH.
3. DIMENSIONS ARE SHOWN IN MILLIMETERS [INCHES].
4. HEATSINK & LEAD DIMENSIONS DO NOT INCLUDE BURRS.

TO-262 Part Marking Information

EXAMPLE: THIS IS AN IRL3103L
LOT CODE 1789
ASSEMBLED ON WW 19, 1997
IN THE ASSEMBLY LINE "C"

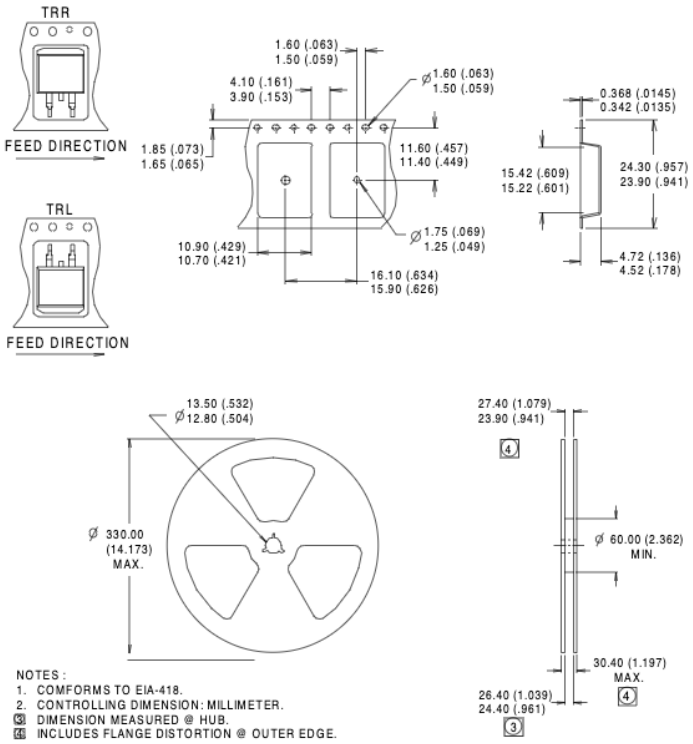


IRF3708/3708S/3708L

International
IOR Rectifier

D²Pak Tape & Reel Information

Dimensions are shown in millimeters (inches)



Notes:

① Repetitive rating; pulse width limited by max. junction temperature.

② Starting T_J = 25°C, L = 0.7 mH
R_G = 25Ω, I_{AS} = 24.8 A.

③ Pulse width ≤ 300μs; duty cycle ≤ 2%.

④ This is only applied to TO-220AB package

International
IOR Rectifier

IR WORLD HEADQUARTERS: 233 Kansas St., El Segundo, California 90245, USA Tel: (310) 252-7105
IR EUROPEAN REGIONAL CENTRE: 439/445 Godstone Rd, Whyteleafe, Surrey CR3 OBL, UK Tel: ++ 44 (0)20 8645 8000
IR CANADA: 15 Lincoln Court, Brampton, Ontario L6T3Z2, Tel: (905) 453 2200
IR GERMANY: Saalburgstrasse 157, 61350 Bad Homburg Tel: ++ 49 (0) 6172 96590
IR ITALY: Via Liguria 49, 10071 Borgaro, Torino Tel: ++ 39 011 451 0111
IR JAPAN: K&H Bldg., 2F, 30-4 Nishi-Ikebukuro 3-Chome, Toshima-Ku, Tokyo 171 Tel: 81 (0)3 3983 0086
IR SOUTHEAST ASIA: 1 Kim Seng Promenade, Great World City West Tower, 13-11, Singapore 237994 Tel: ++ 65 (0)838 4630
IR TAIWAN:16 Fl. Suite D. 207, Sec. 2, Tun Haw South Road, Taipei, 10673 Tel: 886-(0)2 2377 9936
Data and specifications subject to change without notice. 8/00

J. Datasheet: Lithium-Polymer Akkumulator

EREMIT®

Datasheet
Model 606090 4.000mAh

Daniel Beck
Königsteiner Straße 4
61449 Steinbach

Doc. Version 1
30.04.2019

For any questions: Please contact us.

info@theeremit.de
01602614589
www.theeremit.de

This specification is applies to describe the related battery product in this Specification and the battery/cell supplied by EREMIT

All batteries are originally produced by Bixell Technology limited.

Number	Description	Ratings	Remarks
1;	Nominal Capacity	4000mAh	At 0,2C CC discharge
2;	Minimal capacity	3950mAh	
3;	Nominal Voltage	3,7V	
4;	Delivery voltage	3,8x V	On delivery
5;	Charge voltage	4,2V	Max. 4.22V
6;	Standart Charging	0.2C Standart	6 hour nominal
		0.5C max.	2.5 hour rapid
7;	Standart discharging	0.2C CC to 2.5V	
		0.5C max. To 2.5V	
		1C Pulse	Pulse below 10 second 1C standart discharge may cause shorter lifetime

8;	Cell internal impedance	≤28mOhm	Measured at 1khz after 50% Charge
9;	Operating temperature	0-45°C	Maximum -10° - 60°C
		Recommended	10 - 34°C

10; Long time storage (-5°C – 30°C)

If the battery need be stored for a long time, the voltage should be 3.8~4.0V, and stored in the condition as storage proposal. It need at least one charge & discharge cycle every three months.

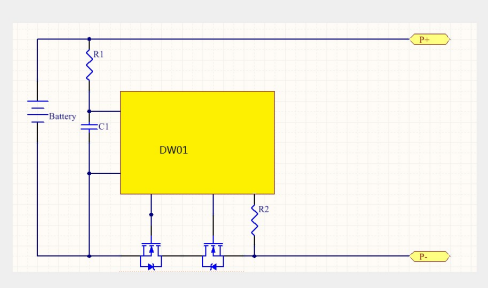
Maximum sizes: 6.2 x 60 x 92mm
Maximum weight: 74gramm

Battery Characteristics

Number	Description	Ratings	Remarks
1;	Standart Charge	Charging cell initially with constant current at 0.2C to 4.2V, then with constant voltage at 4,2V till charge current is below 0.02C	
2;	Rated capacity	Capacity means the discharge capacity of the cell, which is measured with discharge current of 0.2C with 3.0V cut-off voltage after standard charge.	>4.000mAh
3;	Cycle life	Test condition: Carge 0.2C to 4.2V -> discharge 0.2C to 3.0V 80% or more of 1st cycle capacity at 0.5C discharge of operation	>300
4;	Self discharge	After standart charging stored under storage condition descriped in page 2-10; then measured the capacity with 0.2C till 3.0V	Above 95% residual capacity

Protection circuit Data

Item	Symbol	Content	Criterion
Over charge Protection	V_{DET1}	Over charge detection voltage	4.30±0.05V
	tV_{DET1}	Over charge detection delay time	0.96～1.4s
	V_{REL1}	Over charge release voltage	4.10±0.05V
Over discharge protection	V_{DET2}	Over discharge detection voltage	2.40±0.1V
	tV_{DET2}	Over discharge detection delay time	95～173ms
	V_{REL2}	Over discharge release voltage	3.0±0.10V
Over current protection	V_{DET3}	Over current detection voltage	140±30mv
	I_{DP}	Over current detection current	5.0~9.5A
	tV_{DET3}	Detection delay time	7.2～11.0ms
		Release condition	Cut load
Short protection		Detection condition	Exterior short circuit
	T_{SHORT}	Detection delay time	50μs
		Release condition	Cut short circuit
Interior resistance	R_{DS}	Main loop electrify resistance	V_C=4.2V ; R_{DS}≤70mΩ
Current consumption	I_{DD}	Current consume in normal operation	2.4μA Type 6.0μA Max



Page 4

Over-Discharge

Short time over discharge does not affect the battery function, but long time over discharges can damage battery performance, and can't use any more. due to its own self-discharge characteristics also lead to over-discharge, to prevent over-discharge occurs, the battery should maintain the certain electric quantity， the cell shall be charged periodically to maintain between 3.9V and 4.1V.
Over-discharging may causes loss of cell performance, characteristics, or battery functions.
The electrical products shall be equipped with a device to prevent further discharging exceeding a cut-off
voyage specified in the Product Specification. Also the charger shall be equipped with a device to control the recharging procedures as follows:
The cell battery pack shall start with a low current (0.02C) for 30 - 45minutes, i.e. pre-charging, before rapid charging starts. The rapid charging shall be started after the （individual）cell voltage has been reached above 3V within 30 - 45 minutes that can be determined with the use of an appropriate timer for pre-charging. In case the (individual) cell voltage does not rise to 3V within the pre-charging time, then the charger shall have functions to stop further charging and display the cell/pack is at abnormal state.

Charging

Charging current：Do not surpass the largest charging current that specification stipulated。

Charging voltage：Do not surpass the highest limited voltage that specification stipulated。

Charging temperature：within temperature scope that specification stipulated。

Charge with constant current, then with the constant voltage, no reverse charge, which is dangerous

No.	Part Name	Description	Q'ty	Remark
1	Cell	606090-4000mAh	1	
2	Patterns Tape	Blace Insulated tape	2	
3	PCM	Fortune DW01-8205A	1	
4	Wire	AWG22	2	

12. Handling of Cells

Since cells are packed in soft material, for protecting it better performance, careful handling is very important.


a) Soft Aluminium Foil

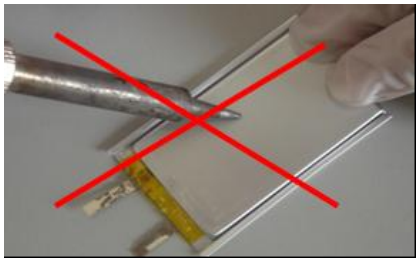
The soft aluminum packing foil may be damaged by sharp matter such as Ni-tabs, pins and needles or other jig and tool.

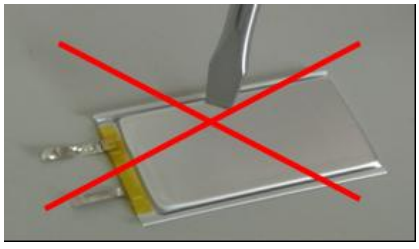
Not strike cell with any sharp matter.

Trim your nail or wear gloves before taking cell.

Clean worktable to sweep away dust.

Avoid component contacting with the edge of foil of cells. Otherwise, there will be corrosion, flatulence and other hidden dangers.



Period of Warranty

The period of warranty is one year from the date of shipment. Replacement is guaranteed within warranty if battery with defects proven due to manufacturing process instead of the customer's abuse and misuse.

Page 7

K. Datasheet: GEH KS 28 - Gehäuse