

DIPLOMARBEIT

Gesamtprojekt

CTrack

Entwicklung eines Ortungssystems für Weidetiere

Samuel Neureiter 5CHEL

Betreuer: Prof. Dipl.-Ing. Jürgen Triebelrig

Marvin Klabacher 5CHEL


Kooperationspartner: Alpenverein Salzburg

ausgeführt im Schuljahr 2022/23

Abgabevermerk:

Datum: 31.03.2023

übernommen von:

	HÖHERE TECHNISCHE BUNDESLEHR- UND VERSUCHSANSTALT Salzburg
	Elektronik und Technische Informatik

Eidesstattliche Erklärung


Ich erkläre an Eides statt, dass ich die vorliegende Diplomarbeit selbständig und ohne fremde Hilfe verfasst, keine anderen als die angegebenen Quellen und Hilfsmittel benutzt und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

Salzburg, am 31.03.2023

Verfasserinnen / Verfasser:

Samuel Neureiter

Marvin Klabacher

	HÖHERE TECHNISCHE BUNDESLEHR- UND VERSUCHSANSTALT Salzburg
	Elektronik und Technische Informatik

DIPLOMARBEIT DOKUMENTATION

Namen der Verfasserinnen / Verfasser	Samuel Neureiter Marvin Klabacher
Jahrgang Schuljahr	5CHEL 2022/23
Thema der Diplomarbeit	CTrack – Entwicklung eines Ortungssystems für Weidetiere
Kooperationspartner	Alpenverein Salzburg

Aufgabenstellung	Die reaktivierte Vierkaseralm am Untersberg ist nur über einen 1,5-stündigen Fußmarsch erreichbar. Der zuständige Nebenerwerbsbauer benötigt den Aufenthaltsort der Tiere. Unser System sollte dazu dienen, das Weidevieh zu orten und den Almbetreiber die GPS-Daten zur Verfügung zu stellen. Mit Hilfe des Projektes CTrack soll man das Weidevieh orten und den Almbetreiber die GPS-Daten zur Verfügung stellen können.
------------------	--

Realisierung	Mithilfe des LoRaWan Netzwerks werden die GPS-Daten, welche vom Sender aufgenommen werden, an den Empfänger übermittelt. Anschließend werden die Daten in einer Datenbank ausgewertet und auf einer Webseite ausgegeben.
--------------	--

Ergebnisse	Das Endergebnis zeigt eine Webseite und eine Datenbank mit den gespeicherten GPS-Daten. Zusätzlich werden die Daten am Sendermodul gespeichert, falls die Verbindung zum Empfänger abbricht.
------------	--

Typische Grafik, Foto etc.
(mit Erläuterung)



Teilnahme an Wettbewerben,
Auszeichnungen

Keine Teilnahmen


Möglichkeiten der
Einsichtnahme in die Arbeit

Schulbibliothek der HTBLuVA Salzburg

Approbation
(Datum / Unterschrift)

Prüferin / Prüfer

Direktorin / Direktor
Abteilungsvorständin / Abteilungsvorstand

	COLLEGE OF ENGINEERING Salzburg
	Electronics and Computer Engineering

DIPLOMA THESIS

Documentation

Author(s)	Samuel Neureiter Marvin Klabacher
Form Academic year	5CHEL 2022/23
Topic	CTrack - Development of a positioning system for grazing animals
Co-operation Partners	Alpenverein Salzburg

Assignment of Tasks	<p>The reactivated Vierkaseralm on the Untersberg can only be reached by a 1.5-hour hike. The responsible part-time farmer needs to know the location of the animals. Our system should be used to locate the grazing cattle and provide the GPS data to the alpine operator. The CTrack project aims to locate the grazing cattle and provide GPS data to the alpine operator.</p>
---------------------	---

Realisation	<p>Using the LoRaWAN network, GPS data collected by the transmitter is transmitted to the receiver. Subsequently, the data is analyzed in a database and displayed on a website.</p>
-------------	--

Results	<p>The end result displays a webpage and a database containing the stored GPS data. Additionally, the data is stored in the transmitter module in case the connection to the receiver is lost.</p>
---------	--

Illustrative Graph, Photo
(incl. explanation)



Participation in Competitions
Awards

No participations

Accessibility of
Diploma Thesis

Library of the Secondary Technical College in Salzburg

Approval
(Date / Sign)

Examiner

Head of College
Head of Department

Vorwort

Im Vorfeld der Diplomarbeit wurde versucht ein Projektthema zu finden, welches einen Nutzen hat.

Ziel war es ein System zu entwickeln, dass Landwirte eine Lokalisierung der eigenen Weidetiere ermöglicht.

Im Laufe der Entwicklung mussten sich alle Teammitglieder neuen Herausforderungen stellen. Zur Erfüllung der Wünsche des Kooperationspartners mussten viele neue Inhalte erlernt werden. Die Programmierung von beispielsweise Webapplikationen und Raspberry Pi war für alle neu. Dennoch wurden sie erlernt und ins Projekt integriert.

Die Aufgaben wurden gerecht und gleichmäßig aufgeteilt.

Danksagung

Wir bedanken uns bei jedem, der bei diesem Projekt in jeglicher Art mitgeholfen hat!

Vielen Dank an unseren Auftraggeber **Herrn Sebastian Feldbacher** des Alpenvereins Salzburg. Der Verein hat alle Kosten für dieses Projekt übernommen und uns die Möglichkeit geboten, viel Neues zu lernen. Insbesondere möchten wir uns bei unserer Ansprechpartnerin **Frau Dr. Assoc. Prof. Gudrun Wallentin** bedanken.

Unser Dank gilt auch unserem Projektbetreuer **Herrn Prof. Dipl. -Ing. Jürgen Triebel**.

Ein großes Dankeschön widmen wir unseren Familien und Freunden, die uns im Laufe des letzten Jahres immer unterstützt und motiviert haben.

Inhaltsverzeichnis

1. Einleitung.....	12
1.1 Idee.....	12
1.2 Zielsetzung.....	12
1.3 Individuelle Aufgabenstellung.....	12
1.3.1 GANTT-Diagramm Samuel Neureiter	13
1.3.2 GANTT-Diagramm Marvin Klabacher	14
1.4 Blockschaltbild.....	15
2. Systemspezifikationen.....	16
2.1 Zielbestimmungen.....	16
2.1.1 Musskriterien	16
2.1.2 Wunschkriterien	16
2.1.3 Abgrenzungskriterien	16
2.2 Produkteinsatz.....	17
2.2.1 Anwendungsbereiche.....	17
2.2.2 Zielgruppen.....	17
2.2.3 Betriebsbedingungen	17
2.3 Qualitätsbestimmungen.....	17
2.4 Entwicklungsumgebungen	17
2.4.1 Software	17
2.4.2 Hardware	17
2.4.3 Orgware.....	17
3. Hardware	18
3.1 Modul TTGO T-Beam	18
3.2 Modul TTGO V2.0	18
3.5 Raspberry Pi 3b.....	19
4. Flussdiagramm Klabacher.....	20
5. LoRaWan	20
5.1 Warum LoRaWan?.....	20
5.2 Definition des Begriffs	20
5.3 Topologie von LoRaWan.....	21
5.4 LoRa	21
6. Sender Programmierung	22
6.1 Entwicklungsumgebung	22
6.1.1 PlatformIO	23
6.1.2 Platformio.ini.....	24
6.1.3 JSON.....	25

6.2 Bibliotheken	25
6.2.1 SparkFun_Ublox_Arduino_Library	27
6.2.2 ArduinoJson.h Bibliothek.....	27
6.3 Sender Programmcode.....	28
6.3.1 Schlaf Funktion	30
6.3.2 Setup Funktion	31
6.3.3 Separieren der GPS-Daten.....	32
6.3.4 Loop Funktion.....	33
6.3.5 Übertragung der Daten an den Empfänger	35
7. Flussdiagramm Neureiter	37
8. Empfänger Programmierung.....	37
8.1 Bibliotheken	37
8.2 Empfänger Software Klasse	38
8.3 Empfänger Header-File.....	39
9. Programmierung WebSocket	40
9.1 Was ist ein WebSocket?	40
9.2 Funktionsweise WebSocket	40
9.3 Erklärung Bibliotheken	41
9.4 Software Klasse Socket.....	42
9.5 Socket Header-File.....	46
10. Webapplikation	47
10.1 Design Webapplikation	47
10.2 Software HTML File	47
10.2.1 HTML Teil.....	47
10.2.2 JavaScript Teil	48
11. Datenbank	50
11.1 Wieso Raspberry Pi?.....	50
11.2 Datenbanksystem.....	50
11.2.1 Was ist ein Datenbanksystem?	50
11.2.2 Programmiersprache SQL.....	51
11.2.3 SQLite	52
11.3 Programmierung	53
11.3.1 Qt Creator.....	53
11.3.2 Erklärung Bibliotheken	54
11.3.3 Software Klasse	55
12. Quellen- und Literaturverzeichnis.....	56
13. Verzeichnis der Abbildungen, Tabellen und Codeausschnitte	58

13.1 Abbildungsverzeichnis.....	58
13.2 Codeausschnitte	59
13.3 Abbildungsquellen.....	61
14. Begleitprotokoll gemäß § 9 Abs. 2 PrO	62
14.1 Begleitprotokoll Samuel Neureiter.....	62
14.2 Begleitprotokoll Marvin Klabacher.....	63

1. Einleitung

1.1 Idee

Die reaktivierte Vierkaseralm am Untersberg ist nur über einen 1,5-stündigen Fußmarsch erreichbar. Der zuständige Nebenerwerbsbauer benötigt zu bestimmten Zeitpunkten den Aufenthaltsort der Tiere. Mit Hilfe des Projektes CTrack soll man das Weidevieh orten und den Almbetreiber deren GPS-Daten zur Verfügung stellen können.

1.2 Zielsetzung

Mit diesem innovativen Projekt soll eine Senderplatine, welche die GPS-Daten auf die Empfangsplatine, mit einem darauf laufenden Raspberry PI Server, sendet, entwickelt werden. Auf diese Art und Weise ist es möglich, die Kühe zu lokalisieren.

1.3 Individuelle Aufgabenstellung

Zur Organisation und Verteilung der Aufgabengebiete wurde ein GANTT-Diagramm entwickelt. Folgende Termine wurden als Meilensteine festgelegt:

DA Antrag:	23.10.2022
1. Review:	21.10.2022
2. Review:	18.11.2022
1. Präsentation:	16.12.2022
2. Präsentation (Tag der offenen Tür)	13.01.2023
3. Review:	24.02.2023
4. Review:	17.03.2023
Diplomabgabe:	31.03.2023

Bei den jeweiligen Reviews wurde der aktuelle Projektstand mit dem Projektbetreuer besprochen und die weiteren Vorgehensweisen festgelegt.

1.3.1 GANTT-Diagramm Samuel Neureiter

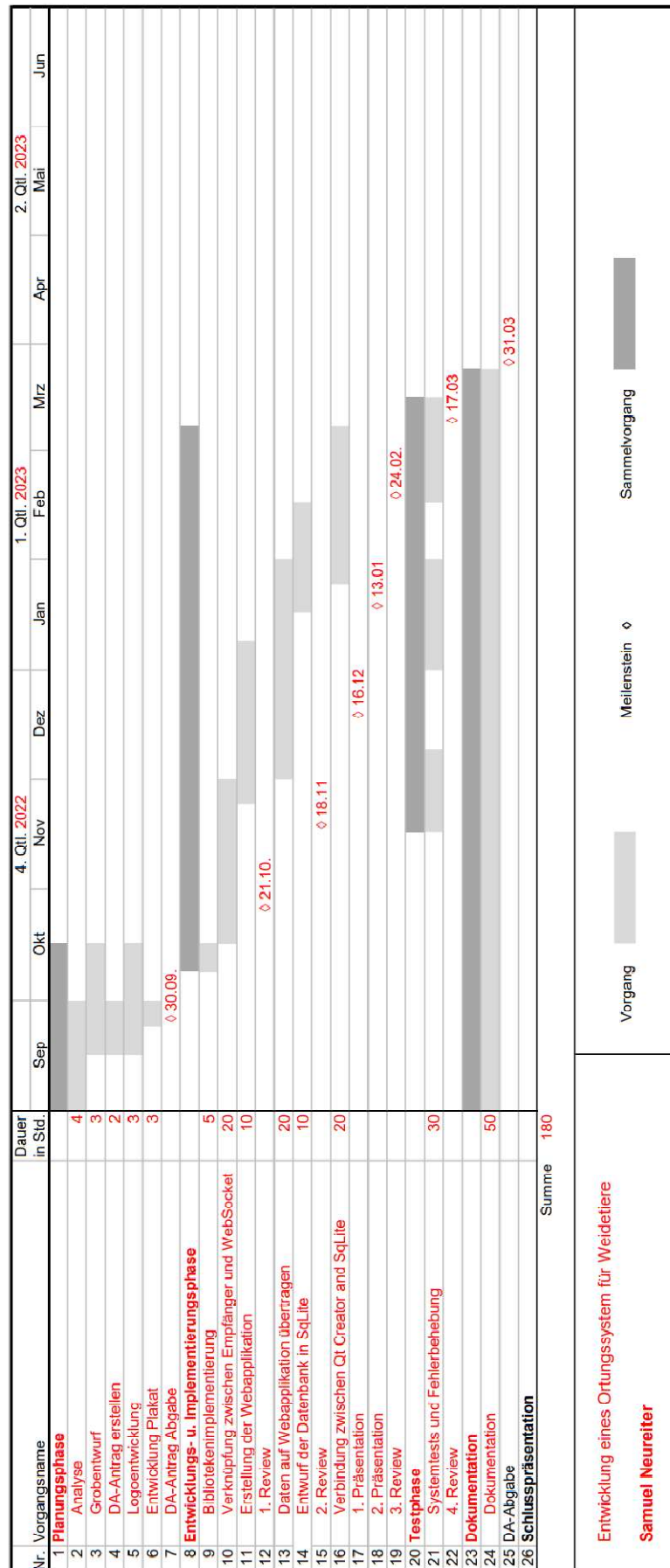


Abbildung 1: GANTT-Diagramm Samuel Neureiter

1.3.2 GANTT-Diagramm Marvin Klabacher

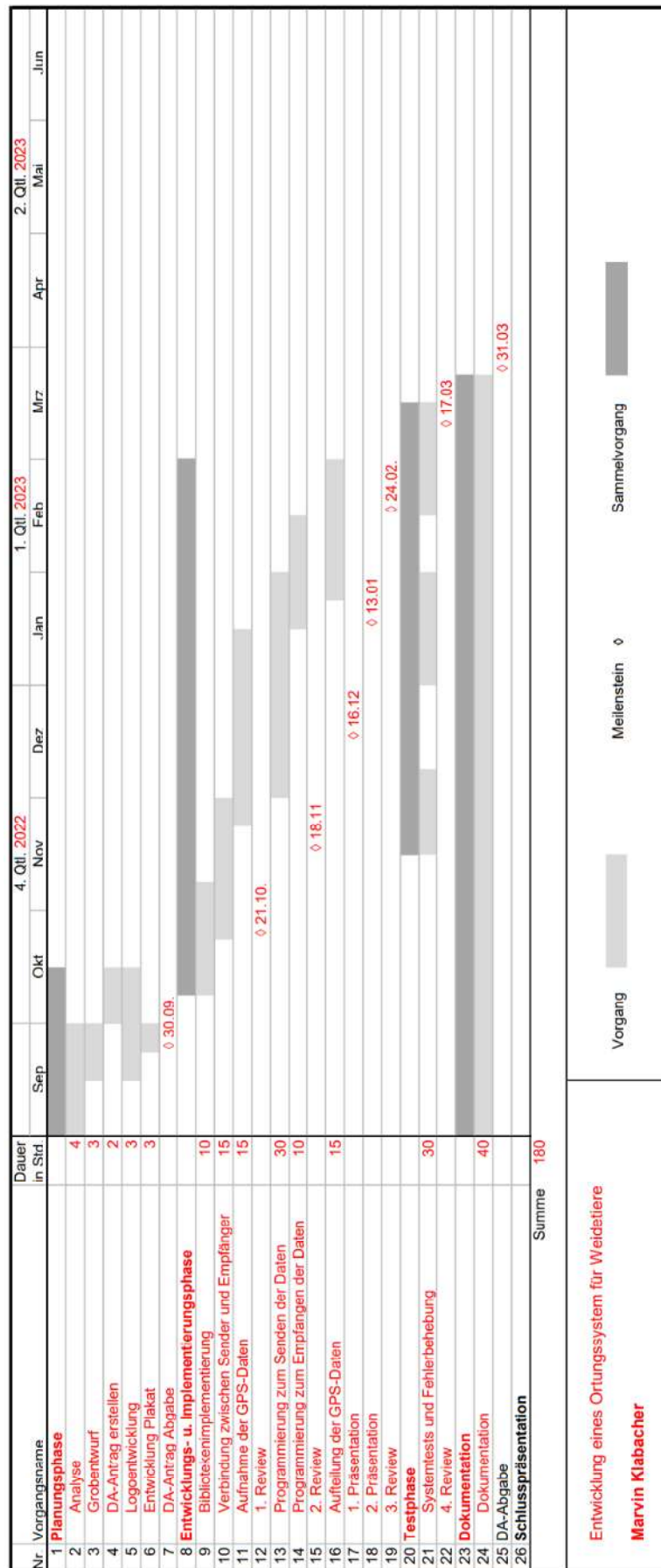


Abbildung 2: GANTT-Diagramm Marvin Klabacher

1.4 Blockschaftbild

Das Sendemodul nimmt die GPS-Daten auf und gibt diese an den Empfänger weiter. Der Empfänger gibt die Daten über den WebSocket an die Webseite und diese werden dort dauerhaft in einer Tabelle dargestellt.

Zur Speicherung der Daten wurde ein SQLite Datenbanksystem verwendet, welches am RaspberryPi läuft. Dort werden die Daten über den Qt-Creator von der Webseite gezogen und in die Datenbank geschrieben.

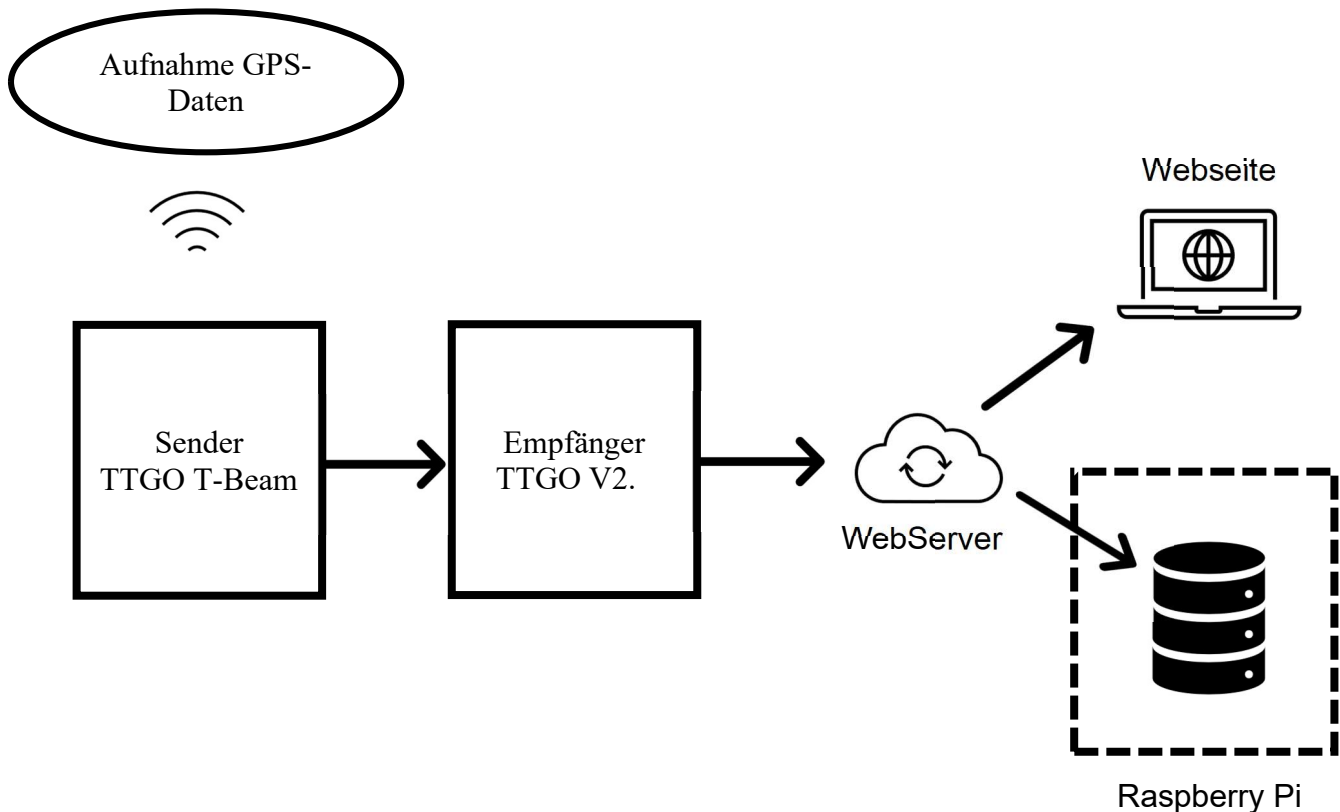


Abbildung 3: Funktionsweise CTrack

2. Systemspezifikationen

2.1 Zielbestimmungen

2.1.1 Musskriterien

- Sender/Empfänger
 - Aufnahme der GPS-Daten
 - Empfangsfunktion
 - Sendefunktion
 - Speicherung der Daten auf dem Sendermodul, wenn keine Verbindung besteht
- Web-Applikation
 - Abbildung der GPS-Daten auf der Webseite
- Datenbank
 - Speicherung der Daten

2.1.2 Wunschkriterien

- Web-Applikation/Datenbank
 - Aufteilung der GPS-Daten in einzelne Werte

2.1.3 Abgrenzungskriterien

- 3D Gehäuse

2.2 Produkteinsatz

2.2.1 Anwendungsbereiche

- Ortung von Weidetieren

2.2.2 Zielgruppen

- Landwirte mit Weidetieren

2.2.3 Betriebsbedingungen

- WLAN-Netzwerk
- Vorhandene Verbindung zum Empfänger

2.3 Qualitätsbestimmungen

	Sehr wichtig	Wichtig	Weniger wichtig	Unwichtig
Komfortabilität		X		
Robustheit	X			
Effizienz		X		
Sicherheit	X			
Zuverlässigkeit	X			

2.4 Entwicklungsumgebungen

2.4.1 Software

- Visual Studio Code
- Qt Creator
- SQLite

2.4.2 Hardware

- TTGO T-Beam
- TTGO V2.0
- Raspberry Pi 3b

2.4.3 Orgware

- Microsoft Office (inkl. OneDrive)
- GitHub

3. Hardware

3.1 Modul TTGO T-Beam

Für den Sendevorgang wird das Modul TTGO T-Beam von LILYGO verwendet. Auf der Platine befinden sich eine Antenne und ein LoRa- sowie ein GPS-Modul. Gesteuert wird das Modul mithilfe eines ESP32. Das GPS-Modul kommuniziert über eine serielle Schnittstelle mit dem ESP32. Es liefert Positionsdaten wie Längen- und Breitengrade und das Datum sowie die genaue Uhrzeit. Die Stromversorgung erfolgt durch einen 18650 Lithium-Ionen-Akku.



Abbildung 4: TTGO T-Beam

3.2 Modul TTGO V2.0

Als Empfangsmodul wird das LoRa32 TTGO V2.0 von LILYGO gewählt. Dabei handelt es sich um eine Platine, auf welchem ein ESP32 und eine Antenne verbaut sind. Gesteuert wird die Platine über dem ESP32-Modul. Die Stromversorgung erfolgt über ein Micro USB-Kabel.

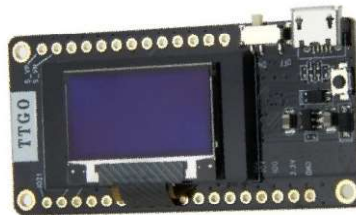


Abbildung 5: TTGO V2.0

3.5 Raspberry Pi 3b

Der Raspberry Pi 3b ist ein Minicomputer, der alle Prozesse gleich ausführt wie ein Computer in normaler Größe. Auf diesem Gerät läuft der Qt-Creator und die dazugehörige SQLite Datenbank.



Abbildung 6: Raspberry Pi 3b

4. Flussdiagramm Klabacher

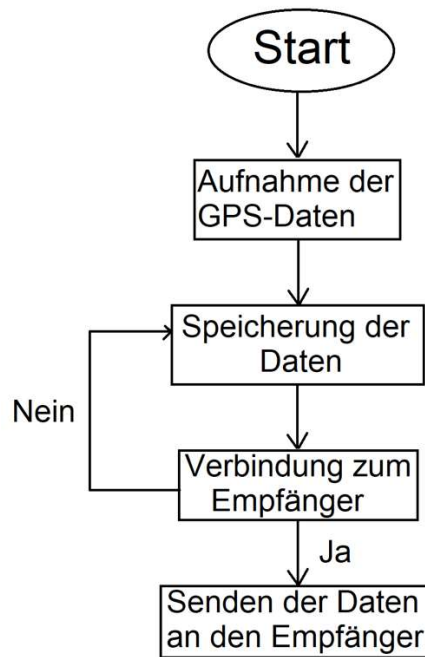


Abbildung 7: Flussdiagramm Klabacher

5. LoRaWan

5.1 Warum LoRaWan?

Für dieses Projekt wird LoRaWan¹ verwendet, da es im Vergleich zu anderen Netzwerken, wie Sigfox und NB-IoT einige Vorteile mitbringt. Entscheidende Vorteile sind, dass über eine große Distanz Daten gesendet und empfangen werden können und der Stromverbrauch sehr gering ist. Weiteres hat LoRaWan eine hohe Zuverlässigkeit und niedrige Kosten. Dies macht es zu einer kosteneffizienten Lösung für drahtlose Verbindungen.

5.2 Definition des Begriffs

LoRaWan ist eine spezifische Art von LPWAN (Low Power Wide Area Network), welche sich auf drahtlose batteriebetriebene Systeme bezieht. Diese Systeme befinden sich in einem nationalen, regionalen oder globalen Netzwerk, die aber an das Internet angeschlossen sein müssen. Dies wurde speziell für das Internet of things (IoT) und Industrial Internet of Things (IIoT) entwickelt. Das Ziel von LoRaWan ist eine abgesicherte Ende-zu-Ende Verbindung.

¹ (LineMetrics GmbH)

5.3 Topologie von LoRaWan

Bei der Architektur des Netzwerkes handelt es sich um eine Sterntopologie. Eine Sterntopologie gewährleistet, dass stets die Gateways verwendet werden, die in der individuellen Situation am geeignetsten sind. Die Gateways haben einen direkten Anschluss an das Internet

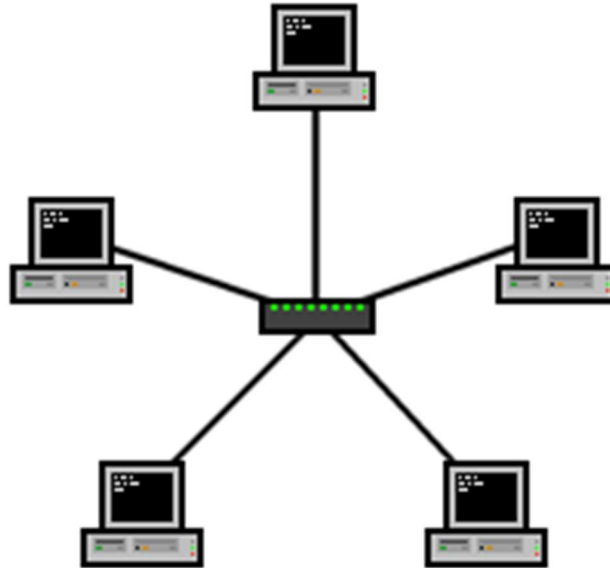


Abbildung 8: Topologie LoRaWan

5.4 LoRa

LoRa² ist die Funktechnik, auf die LoRaWan basiert. Entwickelt wurde diese Technik von der Firma Semtech. Es handelt sich dabei um eine niederfrequente und energieeffiziente Modulationstechnik die Daten bis zu einer Distanz von 40km senden kann. Damit die Übertragung störungsfrei funktioniert muss man sich an den vorgegebenen Frequenzbereich halten. Dieser liegt in Europa zwischen 863-870MHz.

² (LoRa Alliance)

6. Sender Programmierung

Die Aufgabe des Senders ist es, die vom GPS-Modul aufgenommenen Daten in das JSON-Format umzuwandeln und diese Daten mittels LoRaWan an den Empfänger zu senden. Wenn keine Verbindung mit dem Empfänger besteht, werden die empfangenen GPS-Daten in einem FIFO-Speicher am ESP32 gespeichert. Ist eine Verbindung vorhanden, dann werden die gesamten Daten, die im FIFO-Speicher gespeichert sind an den Empfänger gesendet. Das Senderprogramm ist in C++ geschrieben.

Das Programm ist in zwei Prozesse unterteilt:

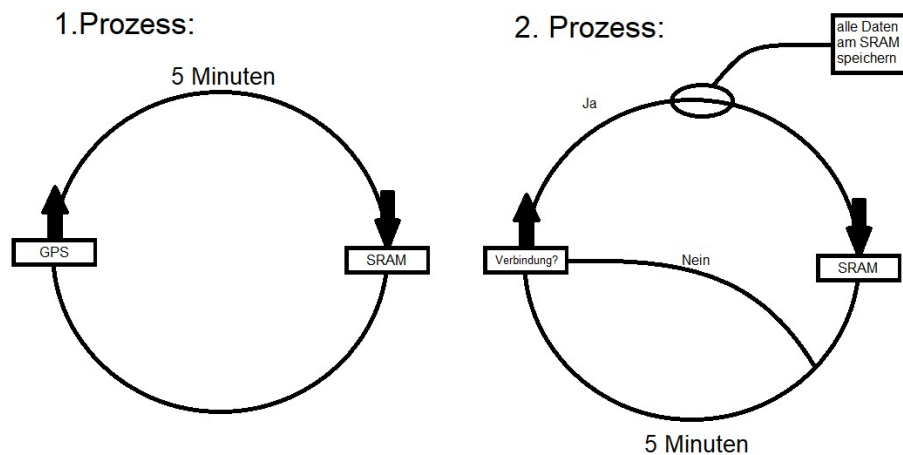


Abbildung 9: Prozesse

1. Prozess

Die Daten werden alle 5 Minuten im SRAM (FIFO-Speicher) gespeichert und an den Empfänger gesendet.

2. Prozess

Es wird alle 5 Minuten nach einer Verbindung gesucht. Wenn keine Verbindung vorhanden ist, werden die Daten im SRAM gespeichert. Besteht wieder eine Verbindung, dann werden alle Daten, die im FIFO-Speicher gespeichert sind, gesendet und der SRAM ist wieder leer.

6.1 Entwicklungsumgebung

Als Entwicklungsumgebung wird die Open-Source Software Visual Studio Code³ verwendet. Visual Studio Code wurde von Microsoft entwickelt und bietet eine breite Palette von Funktionen und Erweiterungen, welche die Entwicklung von Anwendungen erleichtern und beschleunigen können. Visual Studio Code wurde verwendet, da es einfach zu bedienen ist und man die meisten Bibliotheken direkt im Programm downloaden kann.

³ (Microsoft, 2023)

6.1.1 PlatformIO

PlatformIO⁴ ist eine Erweiterung für Visual Studio Code, die eine einfache Möglichkeit bietet IoT-Geräte zu programmieren. Es bietet eine einheitliche Entwicklungsumgebung für verschiedene Mikrocontroller-Plattformen, wie Arduino und ESP32. Somit ist PlatformIO für dieses Projekt sehr gut zu gebrauchen. Damit man ein Projekt mit PlatformIO erstellen kann muss man die Software zuerst installieren. Das funktioniert wie folgt:



Zunächst muss man unter „Extensions“ die Software PlatformIO IDE installieren.

Abbildung 10:
Extensions

Wenn die Installation abgeschlossen ist, kann ein neues Projekt angelegt werden.

Welcome to PlatformIO



Core 6.1.6 · Home 3.4.3

Quick Access

+ New Project

Import Arduino Project

Open Project

Project Examples

☒ Show at startup

Abbildung 11: Erstellung eines Projekts in Visual Studio Code

⁴ (PlatformIO, 2023)

Durch Doppelklick auf „New Projekt“ wird ein Fenster geöffnet, in welchem der Name, das Board und die Programmierumgebung ausgewählt werden kann.

Abbildung 12: Projekteigenschaften

6.1.2 Platformio.ini

In jedem Projekt, das mit PlatformIO erstellt wird, wird eine platformio.ini Datei automatisch mitgeneriert. Die Datei wird im Wurzelverzeichnis des Projekts platziert und enthält Informationen über das Zielsystem, das verwendete Board, die verwendeten Bibliotheken, Compiler-Optionen und andere projektspezifische Einstellungen wie die Geschwindigkeit des Seriellen Monitors, welche vor allem für das Testen wichtig ist.

```
; PlatformIO Project Configuration File
;
; Build options: build flags, source filter
; Upload options: custom upload port, speed and extra flags
; Library options: dependencies, extra library storages
; Advanced options: extra scripting
;
; Please visit documentation for the other options and examples
; https://docs.platformio.org/page/projectconf.html
```

```
[env:ttgo-t-beam]
platform = espressif32
board = ttgo-t-beam      //Verwendetes Board
framework = arduino      //Verwendete Programmierumgebung
monitor_speed = 115200   //Serieller Monitor Geschwindigkeit
lib_deps =               //Manuell installierte Bibliotheken
    sandeepmistry/LoRa@^0.8.0
    bblanchon/ArduinoJson@^6.21.2
    sparkfun/SparkFun u-blox Arduino Library@^1.8.11
```

Codeausschnitt 1: platformio.ini

6.1.3 JSON

JSON (JavaScript Object Notation)⁵ ist ein Datenformat, das verwendet wird, um Daten zwischen Anwendungen auszutauschen. Es ist eine textbasierte Syntax, die darauf ausgelegt ist, von Menschen leicht gelesen werden zu können und von Maschinen einfach verarbeitet werden zu können. In diesem Projekt wird das JSON-Format verwendet, um die Daten, welche vom GPS-Modul empfangen werden, zu trennen und um diese leichter leserlich zu machen.

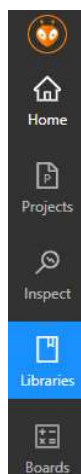
```
JSON string: {"id":1,"date":"160523","time":"201027.00","latitude":"4737.96240","longitude":"01309.09302"}
```

Abbildung 13: Beispiel JSON String

6.2 Bibliotheken

Ein Problem in Visual Studio Code ist, dass Bibliotheken wie, LoRa.h oder ArduinoJson.h nicht vorhanden sind und diese in PlatformIO manuell installiert werden müssen. Der IncludePath der Bibliotheken wird automatisch mitgeneriert und darf nicht verändert werden, da sonst die Librarys nicht mehr funktionieren und ein neues Projekt erstellt werden muss.

Dies funktioniert wie folgt:



Für den Download einer Bibliothek muss via Doppelklick das Verzeichnis „Libraries“ geöffnet werden.

Abbildung 14:
Bibliotheken

⁵ (Visual Studio Code, 2023)

Im Anschluss öffnet sich ein Fenster, indem man die gewünschte Bibliothek suchen kann. Für dieses Projekt muss LoRa.h, ArduinoJson.h und SparkFun_Ublox_Arduino_Library.h manuell installiert werden.

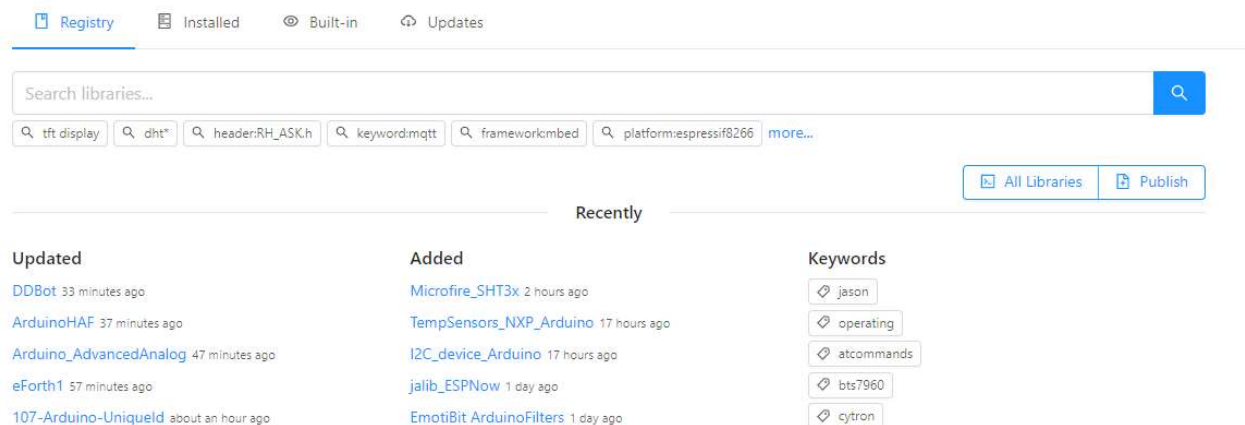


Abbildung 15: Bibliotheken Register

Hat man die benötigte Library gefunden, kann man diese ganz einfach zum Projekt hinzufügen.

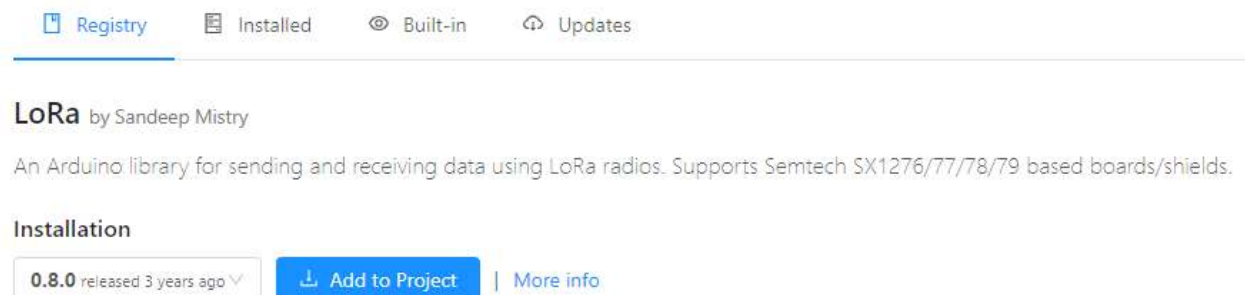


Abbildung 16: Installation der Bibliotheken

6.2.1 SparkFun_Ublox_Arduino_Library

Die Library wurde speziell für das GPS-Modul von u-blox entwickelt. Sie bietet die einfachste Möglichkeit, mit u-blox-Modulen zu kommunizieren und Daten zu lesen. Diese Bibliothek dient hauptsächlich zur Aufnahme und Erfassen der GPS-Daten.

6.2.2 ArduinoJson.h Bibliothek

ArduinoJson.h ist eine Header-Datei für die ArduinoJson-Bibliothek. ArduinoJson ist eine leistungsstarke JSON-Bibliothek für eingebettete Systeme, insbesondere für Arduino-Boards. Sie stellt Funktionen und Klassen zur Verfügung, um JSON-Daten zu analysieren und zu generieren. Durch das Einbinden dieser Header-Datei in den Code erhält man Zugriff auf die Funktionalität der Bibliothek, wie das Erstellen von JSON-Objekten.

6.3 Programmcode Sender

Bevor man mit dem Programm beginnen kann, müssen die Bibliotheken `ArduinoJson.h`, `SparkFun_Ublox_Arduino_Library.h` sowie die Arduino Bibliothek für den ESP32 installiert werden. Zu Beginn werden die benötigten Bibliotheken geladen.

Erklärung der verwendeten Bibliotheken:

```
#include <Arduino.h>
#include <string>
#include <SPI.h>
#include <LoRa.h>
#include <SparkFun_Ublox_Arduino_Library.h>
#include <ArduinoJson.h>
#include <Preferences.h>
#include <iostream>
#include <queue>
#include <esp_sleep.h>
```

Codeausschnitt 2:Includes

Arduino.h: Ist die Standardbibliothek von Arduino und stellt grundlegende Funktionen bereit.

string: Diese Library umfasst erweiterte Funktionen für Strings.

SPI.h: Ist eine Header-Datei, die Funktionen und Definitionen für die Kommunikation über den seriellen Peripheriebus bereitstellen.

LoRa.h: Ist eine Arduino Bibliothek zum Senden und Empfangen von Daten.

SparkFun_Ublox_Arduino_Library.h: Diese Library ermöglicht die serielle Kommunikation mit dem GPS-Modul.

ArduinoJson.h: Die Library bietet Funktionen und Klassen zum Umgang mit JSON-Daten

Preferences.h: Die Preferences.h Bibliothek ist eine Header-Datei, die Funktionen für die Verwendung des ESP32 bereitstellt.

iostream: Ist eine Standardbibliothek in C++ und bietet Funktionen und Klassen zum Ein- und Ausgeben von Datenströmen.

queue: Diese Bibliothek stellt eine Container-Klasse für das Speichern der Klassen zur Verfügung.

esp_sleep.h: Diese Bibliothek ist eine spezielle Library, welche für die Programmierung von Schlaf- und Energiesparmodi auf ESP32-Mikrocontrollern entwickelt wurde.

```
std::deque<String> gpsDataQueue;
```

Codeausschnitt 3: Queue zum Speichern anlegen

Diese Zeile deklariert eine Variable. Hierbei handelt es sich um eine Queue-Datenstruktur, welche dazu dient, eine Sammlung von Elementen vom Typ String zu speichern

```
#define SERIAL1_RX 34 // GPS_TX -> 34
```

```
#define SERIAL1_TX 12 // 12 -> GPS_RX
```

Codeausschnitt 4: Definition serielle Schnittstelle

Im nächsten Schritt wird der Ein- und Ausgang für die serielle Kommunikation mit dem GPS-Modul definiert.

```
#define BAND 866E6
```

Codeausschnitt 5: Frequenzbereich

Diese Anwendung definiert eine Konstante namens BAND, welche den Wert 866MHz hat, der den Frequenzbereich für Europa entspricht.

In den nächsten Zeilen werden die Pins deklariert, an denen das LoRa-Modul am ESP angeschlossen ist.

Definitionen:

```
#define SCK 5
```

```
#define MISO 19
```

```
#define MOSI 27
```

```
#define SS 18
```

```
#define RST 14
```

```
#define DIO0 26
```

Codeausschnitt 6: Pins Deklaration

SCK: Steht für „Serial Clock“ und bezieht sich auf den Takt-Pin des SPI Bus

MISO: Die Definition wird „Master In Slave Out“ genannt und hat die Aufgabe Daten an den Mikrocontroller zu senden.

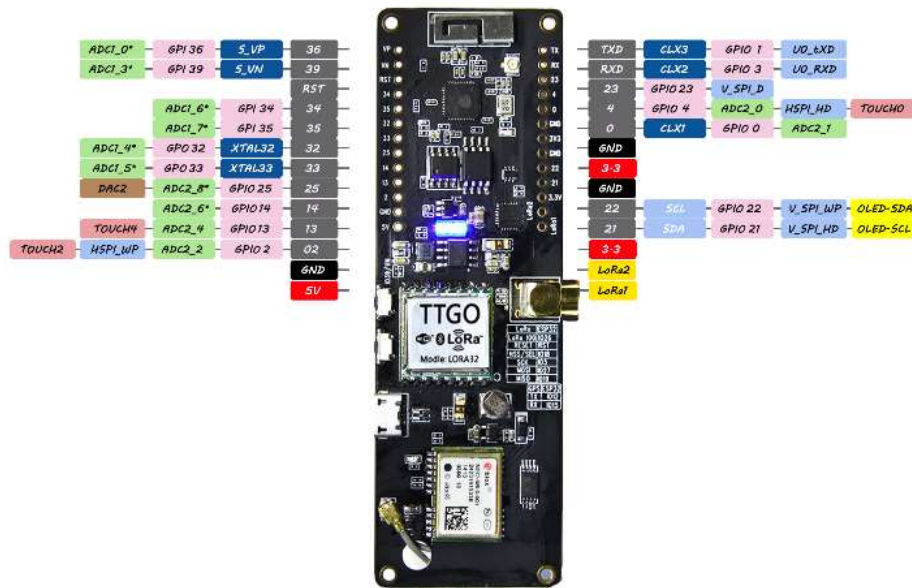
MOSI: „Master Out Slave In“ ist mit dem MISO-Pin verbunden und empfängt Daten.

SS: Bedeutet „Slave Select“ und wird verwendet, um ein Gerät im SPI-Netzwerk zu aktivieren.

RST: RST wird definiert, um das Gerät zu reseten oder neu zu starten.

DIO0: Wird „Digital Input/Output 0“ genannt und bezieht sich auf die digitalen Ein/Ausgangspins auf dem Mikrocontroller

Welcher Pin für was zuständig ist kann man aus der folgenden Abbildung erkennen. Für dieses Projekt werden die Pins RXD, TXD und RST verwendet.



Wifi+Buletooth Board
TTGO-LoRa T-Beam

Abbildung 17: Pinbelegung TTGO T-Beam

6.3.1 Schlaf Funktion

```
void sleepForFiveMinutes() {
    // Deep Sleep für 5 Minuten
    esp_sleep_enable_timer_wakeup(300000000); //300000000us
    esp_deep_sleep_start();
}
```

Codeausschnitt 7: Schlaf Funktion

Die Funktion `sleepForFiveMinutes()` bewirkt, dass das System in einen Schlafmodus versetzt wird und für fünf Minuten inaktiv bleibt. Der ESP32-Mikrocontroller wird während des Schlafmodus den Stromverbrauch reduzieren, um Energie zu sparen.

6.3.2 Setup Funktion

In der Funktion `setup()` werden folgende Konfigurationen und Initialisierungen vorgenommen, um auf das GPS-Modul zugreifen zu können.

```
void setup() {  
  
    SPI.begin(SCK, MISO, MOSI, SS);  
    LoRa.setPins(SS, RST, DIO0);  
  
    while (!Serial);  
    Serial1.begin(9600, SERIAL_8N1, SERIAL1_RX, SERIAL1_TX);  
    delay(300);  
}
```

Codeausschnitt 8: Setup Funktion

In den darauffolgenden Zeilen wird die SPI-Schnittstelle des Mikrocontrollers initialisiert, um mit dem GPS-Modul kommunizieren zu können. Als nächstes werden die Pins konfiguriert. Wenn Daten über die serielle Schnittstelle empfangen werden können, wird die Funktion `Serial1.begin()` aufgerufen und der Mikrocontroller empfängt die Daten. Für die serielle Kommunikation wird eine Baudrate von 9600 verwendet.

```
if (!LoRa.begin(BAND)) {  
    Serial.println("Connection failed");  
}  
do {  
    if (NEO6GPS.begin(Serial1)) {  
        Serial.println("Connection is OK");  
        NEO6GPS.setUART1Output(COM_TYPE_NMEA);  
        NEO6GPS.saveConfiguration();  
        NEO6GPS.disableNMEAMessage(UBX_NMEA_GLL, COM_PORT_UART1);  
        NEO6GPS.disableNMEAMessage(UBX_NMEA_GSA, COM_PORT_UART1);  
        NEO6GPS.disableNMEAMessage(UBX_NMEA_GSV, COM_PORT_UART1);  
        NEO6GPS.disableNMEAMessage(UBX_NMEA_VTG, COM_PORT_UART1);  
        NEO6GPS.enableNMEAMessage(UBX_NMEA_RMC, COM_PORT_UART1); // benötigtes  
Paket  
        NEO6GPS.disableNMEAMessage(UBX_NMEA_GGA, COM_PORT_UART1);  
        NEO6GPS.saveConfiguration();  
        break;  
    }  
    delay(1000);  
} while(1);  
}
```

Codeausschnitt 9: Verbindungsaufbauüberprüfung

In diesem Codeblock wird nach einer LoRa- und GPS-Verbindung gesucht. Wenn keine Verbindung besteht, wird „No connection“ im seriellen Monitor ausgegeben. In der `do-while`-Schleife wird überprüft, ob eine Verbindung zum GPS-Modul besteht. Wenn eine Verbindung zwischen den beiden Chips besteht, werden die Daten alle 5 Minuten an den ESP32 gesendet. Zu beachten ist, dass man die Pakete des NEO6GPS, die nicht benötigt werden deaktiviert, da diese die Rechenleistung des ESP32 beanspruchen und somit mehr Strom verbraucht wird. Bei diesem Projekt muss man nur das RMC-Paket aktivieren, da dieses Paket die gewünschte ID, Zeit, das Datum und Längen sowie Breitengrad sendet.

6.3.3 Separieren der GPS-Daten

```
String sentence_sep(String input, int index) {  
    int finder = 0;  
    int strIndex[] = { 0, -1 };  
    int maxIndex = input.length() - 1;
```

Codeausschnitt 10: Funktion sentence_sep

Die Funktion `sentence_sep()` enthält einen String und einen Integer-Wert als Eingabe und liefert einen String als Ausgabe zurück. Der Eingabe-String entspricht den empfangenen GPS-Daten und durchsucht diese, ob ein Komma enthalten ist. Ist dies der Fall, wird das Wort, welches nach dem Komma kommt, zurückgegeben.

```
    for (int i = 0; i <= maxIndex && finder <= index; i++) {  
        if (input.charAt(i) == ',' || i == maxIndex) { //',' = separator  
            finder++;  
            strIndex[0] = strIndex[1] + 1;  
            strIndex[1] = (i == maxIndex) ? i + 1 : i;  
        }  
    }  
  
    return finder > index ? input.substring(strIndex[0], strIndex[1]) : "";
```

Codeausschnitt 11: Extrahierung der der Daten

In den nächsten Zeilen des Codes wird das ganze Paket als Eingabe genommen und extrahiert den Indexwert. Als nächstes wird wie bei der Funktion `sentence_sep()` jeder Buchstabe des Strings auf ein Komma überprüft. Ist ein Komma gefunden, dann wird der Zähler erhöht. Ist ein Teilstring gefunden, wird der Teilstring zurückgegeben. Wenn kein Index gefunden wird, wird ein leerer String zurückgegeben.

6.3.4 Loop Funktion

Die Loop Funktion wird wiederholt ausgeführt, wenn der ESP32 eingeschaltet ist. Im Wesentlichen ist die Loop Funktion die Hauptfunktion des Programms.

```
void loop() {
```

Codeausschnitt 12: Loop Funktion

```
    read_sentence = Serial1.readStringUntil(13); //13 = return (ASCII)
    read_sentence.trim();
```

Codeausschnitt 13: Leerzeichen entfernen

Als nächstes wird eine Zeile von der seriellen Schnittstelle eingelesen und entfernt alle nachfolgenden Leerzeichen. Die Funktion `Serial1.readStringUntil(13)` liest eine Zeile von der seriellen Schnittstelle ein, die mit einem Zeilenumbruch endet. Diese Zeile wird als String zurückgegeben. `Read_sentence.trim()` entfernt alle nachfolgenden Leerzeichen.

```
    if (Serial1.available()) {
```

```
        String gps_ID    = sentence_sep(read_sentence, 0); //ID
        String gps_Time  = sentence_sep(read_sentence, 1); //Time
        String gps_lat   = sentence_sep(read_sentence, 3); //Latitude
        String gps_long  = sentence_sep(read_sentence, 5); //Longitude
        String gps_date  = sentence_sep(read_sentence, 9); //Date
```

Codeausschnitt 14: Aufteilung der GPS-Daten

Sind Daten auf dem Serial1-Port verfügbar, dann wird eine Zeichenkette mit den GPS-Daten von diesem Port gelesen und in der Variable „read_sentence“ die oben definiert wurde gespeichert. Mithilfe der Funktion `sentence_sep()` werden die GPS-Daten extrahiert und in eine separate String Variable gespeichert. Es werden die ID, die Zeit, das Datum sowie der Längen- und der Breitengrad in die genannten Variablen gespeichert.

6.3.3.1 Erstellen des JSON-Strings

```
    String gpsDataString = "{\"id\":1,\"date\":\"\" + gps_date +
    "\",\"time\":\"\" + gps_Time + "\",\"latitude\":\"\" + gps_lat + "\",\"lon-
    gitude\":\"\" + gps_long + "\"}";
```

Codeausschnitt 15: Erstellung des JSON-Strings

In diesem Codeabschnitt werden die aufgeteilten GPS-Daten in das JSON-Format umgewandelt. Dies basiert auf die ArduinoJson Bibliothek. Zu beachten ist, dass man die Variablen `gps_date`, `gps_Time`, `gps_lat` und `gps_long` nicht unter Anführungszeichen schreibt, da es nicht dem JSON-Format entspricht und somit einen Fehler zurückgibt.

```
    gpsDataQueue.push_back(gpsDataString);
```

Codeausschnitt 16: GPS-Daten in den FIFO-Speicher einfügen

In dieser Zeile werden die empfangenen GPS-Daten in den FIFO-Speicher für die Speicherung der Daten eingefügt.

```
Serial.print("Serialized JSON string: ");  
Serial.println(gpsDataString);
```

Codeausschnitt 17: JSON-String am Monitor ausgeben

Darauffolgend werden die empfangenen Daten am seriellen Monitor ausgegeben, um zu checken, ob Daten empfangen werden können.

```
if (Serial1.available() && !gpsDataQueue.empty())
```

Codeausschnitt 18: Verbindungsüberprüfung

In der if-Verzweigung wird gecheckt, ob eine Verbindung besteht und ob GPS-Daten im FIFO-Speicher vorhanden sind.

```
String serializedJsonString = gpsDataQueue.front();  
gpsDataQueue.pop_front();
```

Codeausschnitt 19: Datenpaket aus FIFO entnehmen

Mit diesen Codezeilen werden die gesamten GPS-Datenpakete aus dem FIFO-Speicher entnommen.

```
Serial.print("Sent JSON string: ");  
Serial.println(serializedJsonString);
```

Codeausschnitt 20: Finale Ausgabe am seriellen Monitor

In diesen beiden Zeilen wird der serialisierte JSON-String am seriellen Monitor ausgegeben.

```
Serialized JSON string: {"id":1,"date":"160523","time":"214234.00","latitude":"4737.96519","longitude":"01309.09227"}  
Sent JSON string: {"id":1,"date":"160523","time":"214234.00","latitude":"4737.96519","longitude":"01309.09227"}
```

Codeausschnitt 21: Beispiel vom seriellen Monitor

6.3.5 Übertagung der Daten an den Empfänger

```
LoRa.beginPacket();
```

Codeausschnitt 22: Sendevorgang startet

Die Funktion *LoRa.beginPacket()* ist eine Methode von der LoRa Bibliothek damit Daten über das LoRa Netzwerk gesendet werden können. Die Daten können nun alle fünf Minuten an den Empfänger übermittelt werden.

```
LoRa.print(serializedJsonString);
```

Codeausschnitt 23: Daten werden an Empfänger gesendet

Der Sender sendet nun über die Funktion *LoRa.print()* die GPS-Daten an den Empfänger. Es werden nun die ID als Integer-Wert gesendet, da das GPS-Modul keine ID liefert. Die restlichen Daten werden als String gesendet, die oben aufgeteilt wurden. Es kann die Möglichkeit bestehen, dass die Uhrzeit um ein paar Sekunden verzögert ist, da das GPS-Modul die Zeit immer aufnimmt und der Sender nicht mit dem Senden der Daten hinterherkommt

```
LoRa.endPacket();
```

Codeausschnitt 24: Sendevorgang wird beenden

Diese Funktion beendet den Sendevorgang.

```
sleepForFiveMinutes();
```

Codeausschnitt 25: Aufruf Schlaffunktion

Dieser Aufruf ist eine Funktion, die oben im Code definiert ist und den ESP32 in den tiefen Schlafmodus versetzt, um für fünf Minuten inaktiv zu bleiben.

```
gpsDataQueue.push_back("{\"id\":1,\"date\":\"2023-04-15\", \"time\":\"12:34:56\", \"latitude\":\"12.3456\", \"longitude\":\"-123.4567\"}");

Serial.print("GPS-Datenpaket in Warteschlange: ");
Serial.println(gpsDataQueue.front());

String serializedJsonString = gpsDataQueue.front();
gpsDataQueue.pop_front();

Serial.print("Entnommenes GPS-Datenpaket: ");
Serial.println(serializedJsonString);
```

In diesen Zeilen wird überprüft, ob der FIFO-Speicher funktioniert. Dies funktioniert wie folgt. Mit der Funktion *gpsDataQueue.push_back()* wird manuell ein JSON-Datenpaket zur Warteschlange hinzugefügt. Anschließend wird das erste Element in der Warteschlange abgerufen und über die serielle Schnittstelle ausgegeben. Danach wird die Zeichenkette des abgerufenen Elements in einer separaten Variablen *serializedJsonString* gespeichert und das Element aus der Warteschlange entfernt. Die Funktion *Serial.println()* gibt das entnommene GPS-Datenpaket am seriellen Monitor aus.

7. Flussdiagramm Neureiter

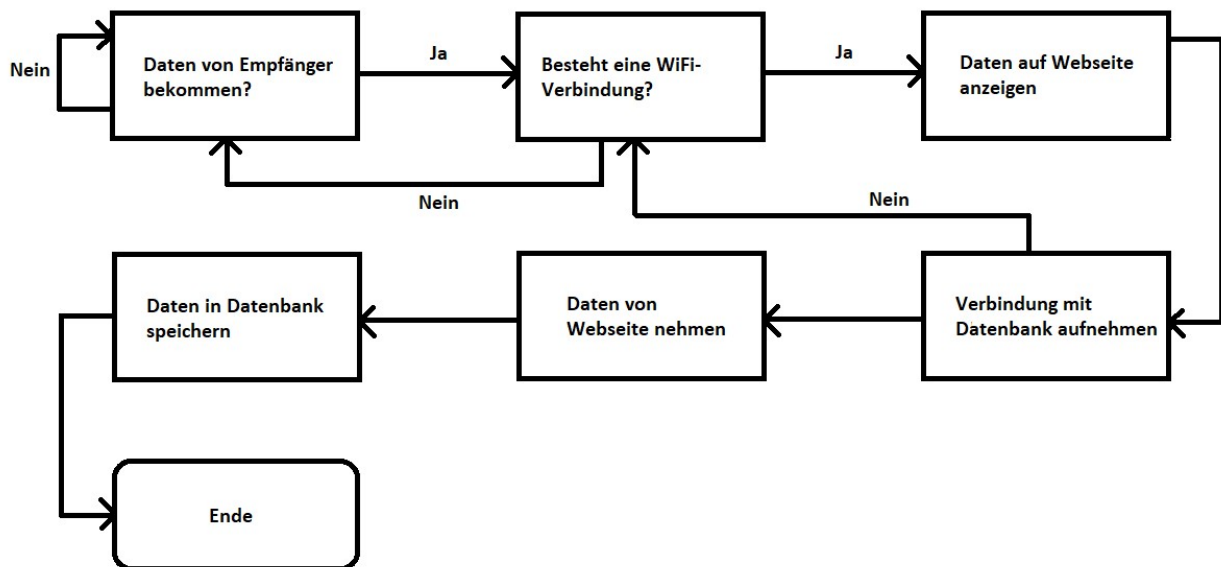


Abbildung 18: Flussdiagramm Neureiter

8. Empfänger Programmierung

8.1 Bibliotheken

```
#include <SPI.h>
#include <LoRa.h>
```

Codeausschnitt 26: Include-Files

SPI.h: Dies ist eine Standard-Arduino Bibliothek, die zur Kommunikation mit dem LoRa-Modul verwendet wird.

LoRa.h: Diese Bibliothek dient der Kommunikation und Steuerung von LoRa-Modulen.

8.2 Empfänger Software Klasse

```
String Receiver::LoRaData;
```

Codeausschnitt 27: Variablendeklaration

Diese Funktion wird mit *Receiver::* initialisiert, damit diese auch in anderen Klassen verwendbar sind.

In dieser Code-Zeile wird die Variable „LoRaData“ deklariert, welche die empfangenen Daten des LoRa-Modules beinhaltet und in den weiteren Klassen weitergibt.

```
void Receiver::setup_receiver() {  
  
    SPI.begin(SCK, MISO, MOSI, SS);  
    LoRa.setPins(SS, RST, DIO0);  
  
    if (!LoRa.begin(BAND)) {  
        Serial.println("LoRa konnte nicht gestartet werden!");  
        while (1);  
    }  
    Serial.println("LoRa OK! \n");  
}
```

Codeausschnitt 28: Funktion *setup_receiver()*

Die Funktion *setup_receiver()* initialisiert typische Anweisungen, die ausgeführt werden, bevor das Hauptprogramm beginnt. Die Pins des LoRa-Modules werden initialisiert und es wird geprüft, ob eine Verbindung zwischen Sender und Empfänger stattfindet.

```
void Receiver::loop_receiver() {  
    //Datenpaket parsen  
    int packetSize = LoRa.parsePacket();  
    if (packetSize) {  
        //Paket auslesen  
        while (LoRa.available()) {  
            LoRaData = LoRa.readString();  
        }  
    }  
}
```

Codeausschnitt 29: Funktion *loop_receiver()*

Der Aufruf der Funktion *loop_receiver()* liest das empfangene Datenpaket aus und speichert die Datei in der Variable **LoRaData**.

```
{"id":1,"date":"160523","time":"214234.00","latitude":"4737.96519","longitude":"01309.09227"}
```

Abbildung 19: Empfangenes Datenpaket

```
String Receiver::getLoRaData(){  
    return LoRaData;  
}
```

Codeausschnitt 30: Getter

Die Methode `getLoRaData()` wird benötigt, damit die erhaltenen LoRa-Daten in den weiteren Klassen verwendbar sind. Dazu wird eine Getter Funktion erstellt, welche den Zugriff auf den Wert einer privaten Instanzvariable in einer Klasse ermöglicht.

8.3 Empfänger Header-File

```
class Receiver  
{  
    public:  
        static void setup_receiver();  
        static void loop_receiver();  
        static String getLoRaData();  
  
    private:  
        static String LoRaData;  
};
```

Codeausschnitt 31: Header-Datei Receiver

Das Header-File des Empfängers ist eine Textdatei, die 3 öffentliche Funktionsprototypen und eine private Variablendeklaration enthält damit diese in anderen Klassen aufrufbar sind.

Öffentliche Funktionsprototypen:

- `setup_receiver()`: dient, um den Empfänger einzurichten
- `loop_receiver()`: dient, um den Empfänger zu überprüfen und Daten zu empfangen
- `getLoRaData()`: gibt die Daten zurück, die empfangen werden

Private Variablendeklaration:

- `LoRaData`: ist eine Variable, in der die empfangenen Daten gespeichert werden

9. Programmierung WebSocket

9.1 Was ist ein WebSocket?

Ein WebSocket⁶ basiert auf TCP (Transmission Control Protocol) und definiert, auf welche Weise Dateien zwischen Netzwerken ausgetauscht werden. TCP stellt eine Verbindung zwischen Kommunikationsendpunkten (Sockets) her, somit kann die Verbindung in zwei Richtungen erfolgen. Der WebSocket stellt die Verbindung zwischen einem WebSocket-Server und einer Webanwendung dar.

9.2 Funktionsweise WebSocket

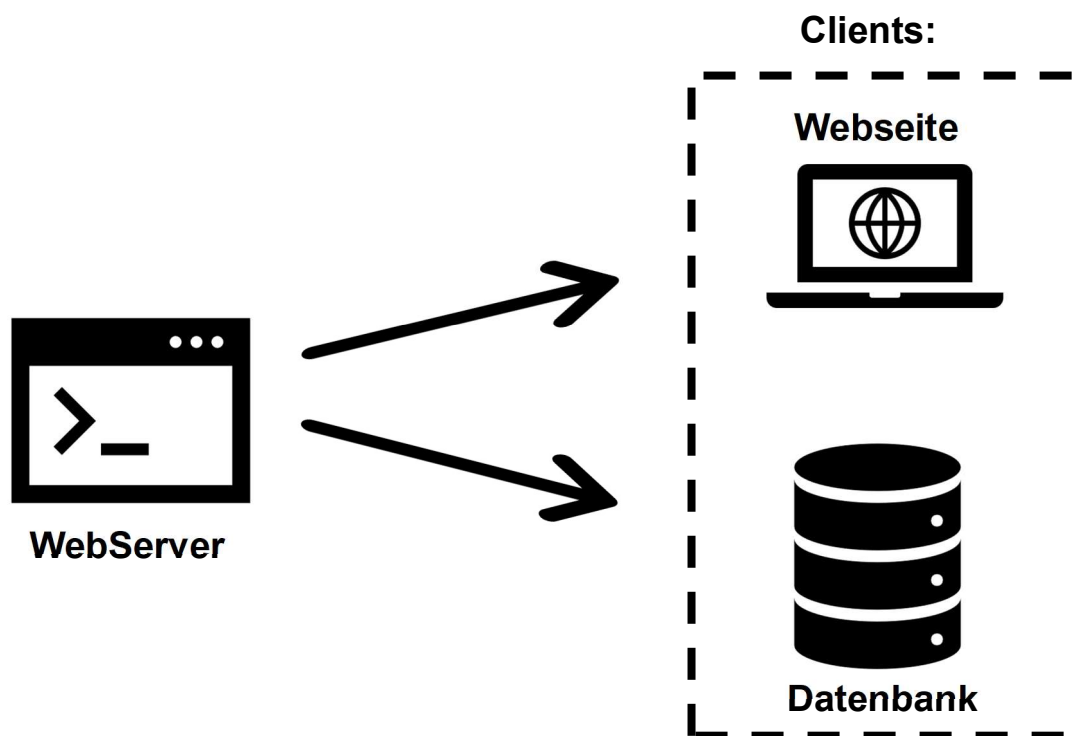


Abbildung 20: Funktionsweise WebSocket

Der WebSocket stellt eine Verbindung zwischen Client und Server her. Die Übertragung der Werte erfolgt von dem WebServer an die Webapplikation und wird dort ausgegeben. Die GPS-Daten werden von der Webseite geladen und mit dem Qt Creator Programm in die SQLite Datenbank gespeichert.

⁶ (IONOS)

9.3 Erklärung Bibliotheken

```
#include <Arduino.h>
#include <WiFi.h>
#include <string.h>
#include <iostream>
#include <string>
#include <cstring>
#include <ArduinoJson.h>
#include <WebSocketsServer.h>
#include <ESPAsyncWebServer.h>
```

Codeausschnitt 32: Bibliotheken Socket

Arduino.h: ist eine Kernbibliothek und stellt die grundlegenden Funktionen bereit.

WiFi.h: enthält Klassen und Funktionen für die Verbindung mit dem drahtlosen Netzwerk.

string.h: stellt Funktionen für Strings, wie das Kopieren, bereit.

iostream: beinhaltet Funktionen, die für die Konsolenausgabe verwendet werden.

string: umfasst erweiterte Funktionen für die Arbeit mit Strings.

cstring: enthält ähnliche Funktionen wie **string.h**

ArduinoJson.h: ermöglicht das Lesen, Schreiben und Bearbeiten von JSON-Daten.

WebSocketsServer.h: stellt eine Implementierung des WebSocket-Protokoll bereit, damit es dem Webbrowser gelingt mit dem ESP32-Board zu kommunizieren.

ESPAsyncWebServer.h: ermöglicht es, Webseiten auf dem Board zu erstellen, die von einem Webbrowser aus gehostet werden können.

9.4 Software Klasse Socket

```
const char *ssid = "FORTI_INTERNET_2G4";  
const char *password = "HTLInternet";
```

Codeausschnitt 33: Definition WLAN

Damit dies als WebSocket funktionieren kann, ist eine stabile WLAN-Verbindung erforderlich. Für eine erfolgreiche Verbindung wird die SSID des WLANs und das Passwort benötigt und muss, wie im Code definiert werden.

```
void notFound(AsyncWebServerRequest *request)  
{  
    request->send(404, "text/plain", "Not found");  
}
```

Codeausschnitt 34: Funktion notFound()

Die Funktion *notFound()* dient dazu, eine Antwort vom WebSocket zu bekommen, wenn keine Verbindung zum WebSocketServer möglich ist. Wenn dies der Fall ist, gibt die Funktion die Fehlermeldung **404 „Text“ Not found** zurück.

```
void onEvent(AsyncWebSocket *server,
             AsyncWebSocketClient *client,
             AwsEventType type,
             void *arg,
             uint8_t *data,
             size_t len)
{
    switch (type)
    {
        case WS_EVT_CONNECT:
            Serial.printf("WebSocket client #%u connected from %s\n", client-
>id(), client->remoteIP().toString().c_str());
            break;
        case WS_EVT_DISCONNECT:
            Serial.printf("WebSocket client #%u disconnected\n", client->id());
            break;
        case WS_EVT_DATA:
        case WS_EVT_PONG:
        case WS_EVT_ERROR:
            break;
    }
}
```

Codeausschnitt 35: Funktion onEvent()

Die Funktion *onEvent()* wird benötigt, um festzustellen, wann sich der WebSocket-Client auf den Server verbindet und trennt. Die Case-Anweisung unterscheidet folgende 5 Fälle:

- Web-Client verbunden: die Serielle Schnittstelle gibt die Client-ID und die IP-Adresse des Clients aus
- Web-Client getrennt: die Serielle Schnittstelle gibt die Client-ID des Clients aus
- Daten vom Client werden empfangen
- Pong-Paket wird empfangen: wird benötigt, um eine Verbindung aufrecht zu halten. Der Client antwortet mit diesem Pong-Paket, um zu bestätigen, dass die Verbindung weiterhin besteht.
- Fehler tritt auf: lässt es dir ermöglichen, auf Fehlerereignisse zu reagieren und eine Fehlerbehandlung durchzuführen.

```
void Socket::start_socket()
{

    Serial.begin(9600);
    WiFi.mode(WIFI_STA);
    WiFi.begin(ssid, password);
    if (WiFi.waitForConnectResult() != WL_CONNECTED)
    {
        Serial.printf("WiFi Failed!\n");
        return;
    }

    Serial.print("IP Address: ");
    Serial.println(WiFi.localIP());

    server.on("/", HTTP_GET, [](AsyncWebServerRequest *request)
        { request->send(200, "text/html", webpageCode); });

    server.onNotFound(notFound);
    server.begin();

    websocket.onEvent(onEvent);
    server.addHandler(&websocket);
}
```

Codeausschnitt 36: Funktion start_socket()

Zur Inbetriebnahme des Servers, wird die Funktion *start_socket()* benötigt. Damit eine Verbindung zum WLAN-Netzwerk herstellbar ist, wird der WLAN-Modus auf *WIFI_STA* gesetzt.

Ist die Verbindung fehlgeschlagen, gibt das Programm „WiFi Failed!“ aus.

Wenn die Verbindung erfolgreich stattfindet, wird die IP-Adresse des Servers über die Serielle Schnittstelle ausgegeben.

Dieser Code definiert Callback-Funktionen, die aufgerufen werden, wenn WebSocket Ereignisse auftreten. Die Funktion *server.begin()* startet den Server und *server.addHandler(&WebSocket)* fügt einen neuen Händler hinzu um auf WebSocket-Anfragen zu reagieren.

```
void update_webpage(String LoRaData)
{
    websocket.cleanupClients();

    JsonObject object = doc.to<JsonObject>();

    object["LoRaData"] = LoRaData;
```

Codeausschnitt 37: Funktion update_webpage()

Der Funktionsaufruf *update_webpage()* wandelt die LoRa-Daten in ein JsonObject um und übermittelt diese an die Webseite.

```
    serializeJson(object, jsonString);
    Serial.println(jsonString);

    websocket.textAll(jsonString);

    jsonString = "";
}
```

Codeausschnitt 38: Konvertierung in JsonString

Für die Ausgabe des Objektes, wird es in einen JSONString konvertiert und anschließend auf den WebSocket gesendet.

```
void Socket::send_data()
{
    LoRaData = Receiver::getLoRaData();
    update_webpage(LoRaData);
}
```

Codeausschnitt 39: Funktion send_data()

Die Funktion *send_data()* übermittelt die aufgenommenen GPS-Daten an die Funktion *update_webpage()*.

9.5 Socket Header-File

```
class Socket
{
private:
    int var;
    char* socket_message;
    const char* PARAM_MESSAGE = "message";

public:
    static String processor();
    static void send_data();
    static void start_socket();
};
```

Codeausschnitt 40: Header-File Socket

Die Header-Datei definiert die C++ Klasse Socket mit den folgenden privaten Member-Variablen:

- Integer-Variable **var**
- Zeiger auf einen Char-Array namens **socket_message**
- Konstanter Zeiger auf einen Char-Array namens **PARAM_MESSAGE** mit dem Wert „message“

Anschließend werden die folgenden drei öffentlichen Member-Funktionen, die als statisch deklariert sind, definiert:

- Funktion: *processor()*, die einen String zurückgibt
- Funktion: *send_data()*, ohne Rückgabewert
- Funktion: *start_socket()*, ohne Rückgabewert

10. Webapplikation

10.1 Design Webapplikation

CTRACK - Entwicklung eines Ortungssystems fuer Weidetiere

GPS-Daten
(1, 153810.00, 170323, 4749.35801, 01302.76526)

Abbildung 21: Webapplikation

Für das Design der Webapplikation wurden die Überschrift **CTRACK – Entwicklung eines Ortungssystems fuer Weidetiere** und die Tabelle mit den Daten in HTML erstellt. Diese wird mit den empfangenen GPS-Daten befüllt.

10.2 Software HTML File

10.2.1 HTML Teil

```
<p><b><br><center><font size="8em"> CTRACK - Entwicklung eines Ortungssystems fuer Weidetiere  
</font></center></br></b></p>
```

Codeausschnitt 41: HTML Code Überschrift

Die Überschrift der Webseite lautet **CTRACK – Entwicklung eines Ortungssystems fuer Weidetiere**. Diese wird mit dem HTML-Code mittig auf der Webseite angezeigt.

```
<div align="center">  
  <table border="5" width="70%">  
    <tr>  
      <th>GPS-Daten</th>  
    </tr>  
    <tr>  
      <td> <span id = "LoRaData"> </span></td>  
    </tr>  
  </table>
```

Codeausschnitt 42: Erstellung der Tabelle

Die Tabelle wird mit der Spalte **GPS-Daten** erstellt und mit den LoRa-Daten befüllt.

10.2.2 JavaScript Teil

```
var gateway = `ws://${window.location.hostname}/ws`;
```

Codeausschnitt 43: Variable gateway

Die Variable **gateway** erhält die IP-Adresse des WebServers.

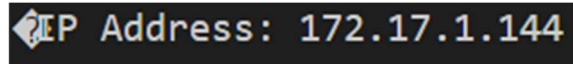


Abbildung 22: IP-Adresse WebServer

```
window.addEventListener('load', onLoad);
```

Codeausschnitt 44: EventListener

Der EventListener wartet, bis die Webseite geladen hat. Die Funktion *onLoad()* wird aufgerufen, wenn sie geladen hat.

```
function onLoad(event) {  
    init();  
}
```

Codeausschnitt 45: Funktion onLoad()

Die Funktion *onLoad()* ruft die *init()* Funktion auf, welche auf dem vorher definierten „gateway“ die WebSocket – Verbindung initialisiert.

```
function init() {  
    console.log('Trying to open a WebSocket connection...');  
    websocket = new WebSocket(gateway);  
    websocket.onmessage = onMessage;  
    websocket.onopen = onOpen;  
};  
  
function onOpen(event) {  
    console.log('Connection opened');  
}
```

Codeausschnitt 46: Funktion init()

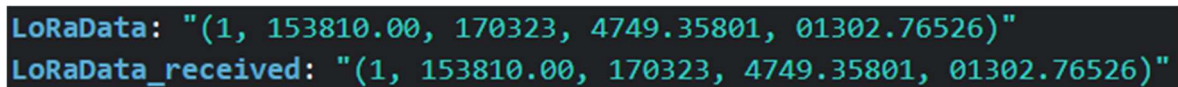
Besteht eine Verbindung oder wird eine Nachricht vom Server empfangen, werden zwei „callback“ Funktionen aufgerufen.


```
function onMessage(event) {  
  console.log(`Received a notification from ${event.origin}`);  
  console.log(event);  
  var obj = JSON.parse(event.data);  
  LoRaData = JSON.parse(event.data);  
  
  console.log(event.data);  
  console.log(LoRaData);  
}
```

Codeausschnitt 47: Funktion onMessage()

Wenn Daten empfangen werden, wird die Funktion *onMessage()* aufgerufen. Dabei handelt es sich um einen JSON-String, der von einem Server zum Client gesendet wird.

Damit der Wert in der Konsole des Browsers angezeigt wird, werden die beiden Befehle *console.log()* verwendet, wie im folgenden Bild.



```
LoRaData: "(1, 153810.00, 170323, 4749.35801, 01302.76526)"  
LoRaData_received: "(1, 153810.00, 170323, 4749.35801, 01302.76526)"
```

Abbildung 23: Konsole-Webseite

```
document.getElementById('LoRaData').innerHTML = obj.LoRaData_received;
```

Codeausschnitt 48: Zuweisung HTML-Element

Diese Codezeile weist dem HTML-Element „LoRaData“ den Wert des Objektes „LoRaData_received“ zu.

11. Datenbank

11.1 Wieso Raspberry Pi?

Damit eine Verbindung zur Datenbank herstellbar ist, wird ein Raspberry PI verwendet. Auf dem Minicomputer befindet sich eine SQLite Datenbank und wird mithilfe des Qt Creator Programmes mit den GPS-Daten befüllt.

11.2 Datenbanksystem

11.2.1 Was ist ein Datenbanksystem?

Eine Datenbank⁷ ist ein System, welches verwendet wird, um die Datenbankdateien erstellen, bearbeiten und verwalten zu können. Die Datenbanksoftware wird auch als „Datenbankverwaltungssystem“ (DBMS) bezeichnet. Ein Datenbanksystem besteht aus zwei wesentlichen Komponenten, auf der einen Seite die Datenbank und auf der anderen Seite dem Datenbankmanagementsystem.

⁷ (OCI, 2019)

11.2.2 Programmiersprache SQL

SQL ist eine Programmiersprache zum Verwalten von Datenbanken und steht für „Structured Query Language“.

Die Grundfunktionen von SQL sind:

- Datenbanken und Tabellen erstellen
- Strukturen verändern
- Daten auslesen
- Daten in eine Tabelle schreiben

Für diese Operationen sind verschiedene Befehle nötig. Die grundlegenden Kommandos sind in der *Abbildung 20* angeführt.

Grundlegende SQL - Befehle

show databases;	Datenbanken anzeigen
use test;	Auswählen der Datenbank "test"
create database test;	Datenbank "test" anlegen
create table test;	Tabelle "test" in einer Datenbank erstellen und fügt die Spalten "name" und
>(name varchar(20), strasse varchar(25));	"strasse" hinzu, und definiert gleichzeitig die Attribute der Spalten
show tables;	Tabellen der ausgewählten Datenbank anzeigen
describe test;	Überprüfung der Tabelle
alter table test rename test1;	Benennt die Tabelle "test" in "test1" um
drop table test;	Löscht Tabelle "test" unwiderruflich
alter table test add strasse varchar(20);	Fügt der Tabelle „test“ die spalte „strasse“ hinzu
alter table test drop strasse;	Löscht aus der Tabelle „test“ die Spalte „strasse“
alter table test change strasse str varchar(20);	Benennt in der Tabelle "test" die Spalte "strasse" in "str" um
alter table test change str str varchar(15);	Ändert Datentyp von "str"
load data local infile "c:\my\test.txt"	Lädt Daten von einer Datei "test.txt" in die Tabelle "test"
>into table test	
> fields terminated by ',' enclosed by '"'	
> lines terminated by '\r\n';	
Delete * From test	Löscht alle Datensätze in der Tabelle "test"
Insert Into test	Fügt Werte von der Kommandozeile aus in eine Tabelle ein. NULL = kein Wert
> Values	
> ('Wert1', 'Wert2', Null);	
Select * from test;	Zeigt den kompletten Inhalt einer Tabelle an
Select * From test Where Hausnr. = "24";	Zeigt alle Datensätze mit der Hausnr. "24" an
Select * From test	Zeigt alle Datensätze mit der Hausnr. "24" an,
> (Hausnr. = "24" AND Nachname = "Maier")	und mit Nachname "Maier", ODER
> OR	alle Datensätze mit der Hausnr. "12" und mit
> (Hausnr. = "12" AND Nachname = "Meier");	Nachname "Meier"

Stand: 22.03.2006

Abbildung 24: Grundlegende SQL-Befehle

11.2.3 SQLite

11.2.3.1 Was ist SQLite?

Die Software SQLite ist eine verbreitete Programmbibliothek und bietet eine vollständige Umgebung für eine SQL-basierte Datenbank und ist in allen gängigen Betriebssystemen implementiert. Ein Vorteil von SQLite ist, dass es gemeinfrei ist. Das heißt die Software ist frei von Urheberrechten.

11.2.3.2 Download SQLite

```
sudo apt install sqlite3
```

Codeausschnitt 49: Installation SQLite

Der Befehl wird ausgeführt, um SQLite installieren zu können.

11.2.3.3 Allgemeine Befehle für die Datenbank

```
sqlite3 CTrackDB.db
```

Codeausschnitt 50: Befehl Datenbankerstellung

Mittels des Befehls wird eine SQLite Datenbank erstellt.

```
.open CTrackDB.db
```

Codeausschnitt 51: Befehl für die richtige Datenbank

Der Aufruf wird ausgeführt, damit die richtige Datenbank verwendet wird.

```
create table ctrack (Daten TEXT);
```

Codeausschnitt 52: SQL-Befehl Tabellenerstellung

Der Befehl wird ausgeführt, damit man eine neue Tabelle in der ausgewählten Datenbank erstellen kann. Dabei gibt man den richtigen Datentyp **TEXT** mit.

```
select * from ctrack;
```

Codeausschnitt 53: SQL-Befehl auslesen

Dieser Aufruf gibt sie gespeicherten Daten aus.

11.3 Programmierung

11.3.1 Qt Creator

Qt Creator ist eine integrierte Entwicklungsumgebung für Linux, macOS und Windows. Die Software ist eine C++-Entwicklungsumgebung, welche besonders für die plattformunabhängigen C++-Programme mit der Qt-Bibliothek gedacht ist.

11.3.1.1 Download Qt Creator

```
sudo apt-get update
```

```
sudo apt-get upgrade
```

Codeausschnitt 54: Kommandos um Raspberry Pi upzugraden

Damit der Raspberry PI auf den neuesten Stand geladen wird, werden vor jeder neuen Installation diese zwei Kommandos ausgeführt.

```
sudo apt-get install qtcreator
```

Codeausschnitt 55: Kommando für die Qt Creator Installation

Mittels des Befehls wird der Qt Creator installiert.

11.3.2 Erklärung Bibliotheken

```
#include <QCoreApplication>
#include <QtSql>
#include <QtDebug>
#include <QSqlError>
#include <QSqlDatabase>
#include <QSqlQuery>
```

Codeausschnitt 56: Bibliotheken Qt Creator

QCoreApplication: wird benötigt, um eine Qt-Anwendung zu erstellen.

QtSql: vereinfacht den Zugriff auf relationale Datenbanken in Qt, da es Klassen und Funktionen enthält.

QtDebug: enthält Hilfsfunktionen, um Debugging-Informationen auszugeben. Diese sind notwendig, damit man Fehler in der Anwendung findet und löst.

QSqlError: ist eine Klasse, die Fehlermeldungen im Zusammenhang mit der Datenbank enthält.

QSqlDatabase: stellt Klassen und Funktionen bereit, um die Verbindung mit der Datenbank herstellen und auf diese zugreifen zu können.

QSqlQuery: enthält Klassen und Funktionen, die das Ausführen von SQL-Statements in Qt zu erleichtern.

11.3.3 Software Klasse

```
QSqlDatabase db = QSqlDatabase::addDatabase("QSQLITE");  
db.setDatabaseName("/home/user/ctrack2.db");
```

```
if (db.open())  
{  
    qDebug() << "Connected";  
}  
else  
{  
    qDebug() << "Failed";  
}
```

Codeausschnitt 57: Verbindung zu Datenbank

Dieser Code initialisiert eine QSqlDatabase-Instanz, damit eine Verbindung zu der SQLite-Datenbank herstellbar ist. Die Methode `db.setDatabaseName()` wird verwendet um den Pfad der SQLite-Datenbank anzugeben, die verwendet wird.

Im nächsten Schritt wird geprüft, ob eine Verbindung mit der Datenbank erfolgreich war oder nicht. Wenn die Verbindung erfolgreich hergestellt wurde, wird „Connected“ in der Qt Creator Konsole ausgegeben. Andernfalls wird „Failed“ ausgegeben.

```
QSqlQuery q;
```

```
q.exec("INSERT INTO ctrack (Daten) VALUES (LoRaDaten)");  
  
if (!q.exec())  
{  
    qDebug() << "Fehler beim Einlesen";  
}  
else  
{  
    qDebug() << "Einlesen erfolgreich";  
}
```

Codeausschnitt 58: Einlesen in Datenbank

Mithilfe des Befehles **INSERT INTO** wird das Einlesen in die Datenbank geschrieben. Es wird ein neuer Datensatz in die Tabelle **ctrack** eingelesen. Dazu werden die GPS-Daten in die Spalte **Daten** inkludiert. In der nächsten Zeile wird überprüft, ob das Einlesen erfolgreich war. Das Ergebnis wird in der Qt Creator Konsole ausgegeben.

12. Quellen- und Literaturverzeichnis

- LoRaWan: linemetrics.com/de/lorawand-und-lorawan-einfach-erklart, LineMetrics GmbH
Abfrage am 15.03.2023
industrie-wegweiser.de/lorawan-lora, Thomas W. Frick
Abfrage am 17.03.2023
elektronik-kompodium.de/sites/kom/2203171.htm, Elektronik-Kompodium
Abfrage am 21.03.2023
[Lora-alliance.org](https://lora-alliance.org), LoRa Alliance
Abfrage am 21.03.2023
- TTGO T-Beam: werner.rothschoepf.net/201905_ttgo_t_beam.htm, Werner Rothschoepf
Abfrage am 21.03.2023
- ESP32: elektor.de/das-offizielle-esp32-handbuch, Dogan Ibrahim
Abfrage am 21.03.2023
- JSON: lernprogrammieren.de/json, Arek
Abfrage am 22.03.2023
codebeautify.org/blog/how-to-create-json-file, CodeBeautify
Abfrage am 03.02.2023
code.visualstudio.com/docs/languages/json, Visual Studio Code
Abfrage am 03.02.2023
- VS Code: microsoft.com/de-de/techwiese, Microsoft
Abfrage am 28.03.2023
- PlatformIO: platformio.org, PlatformIO
Abfrage am 28.03.2023
- Datenaufnahme: aeq-web.com/lorawan-gps-tracker-the-things-stack-tts-application-server, Alex
Abfrage am 23.01.2023
- Extrahierung: randomnerdtutorials.com/guide-to-neo-6m-gps-module-with-arduino, Rui Santos
Abfrage am 24.02.2023
- LoRa: hutscape.com/tutorials/lorawand-duplex-a-esp32-t-beam, Sayanee Basu
Abfrage am 03.02.2023
- Senden: polluxlabs.net/arduino-tutorials/daten-per-lora-senden-und-empfangen-mit-dem-ttgo-lora32, Pollux Labs
Abfrage am 09.01.2023

SQL: [w3schools.com/sql](https://www.w3schools.com/sql/), w3schools
Abfrage am: 06.02.2023

Datenbank: oracle.com/de/database/what-is-database, OCI
Abfrage am: 06.02.2023

WebSocket: ionos.at/digitalguide/websites/web-entwicklung/was-ist-websocket,
IONOS
Abfrage am: 27.02.2023

Qt Creator: doc.qt.io/qt-6/sql-connecting.html, Qt Company
Abfrage am: 10.03.2023

SQLite: biteno.com/was-ist-sqlite, Daniel Faust
Abfrage am: 17.03.2023

13. Verzeichnis der Abbildungen, Tabellen und Codeausschnitte

13.1 Abbildungsverzeichnis

Abbildung 1: GANTT-Diagramm Samuel Neureiter.....	13
Abbildung 2: GANTT-Diagramm Marvin Klabacher	14
Abbildung 3: Funktionsweise CTrack.....	15
Abbildung 4: TTGO T-Beam.....	18
Abbildung 5: TTGO V2.0	18
Abbildung 6: Raspberry Pi 3b.....	19
Abbildung 7: Flussdiagramm Klabacher.....	20
Abbildung 8: Topologie LoRaWan.....	21
Abbildung 9: Prozesse.....	22
Abbildung 10: Extensions	23
Abbildung 11: Erstellung eines Projekts in Visual Studio Code	23
Abbildung 12: Projekteigenschaften	24
Abbildung 13: Beispiel JSON String	25
Abbildung 14: Bibliotheken	25
Abbildung 15: Bibliotheken Register.....	26
Abbildung 16: Installation der Bibliotheken.....	26
Abbildung 17: Pinbelegung TTGO T-Beam.....	30
Abbildung 18: Flussdiagramm Neureiter	37
Abbildung 19: Empfangenes Datenpaket.....	38
Abbildung 20: Funktionsweise WebSocket	40
Abbildung 21: Webapplikation	47
Abbildung 22: IP-Adresse WebServer	48
Abbildung 23: Konsole-Webseite	49
Abbildung 24: Grundlegende SQL-Befehle.....	51

13.2 Codeausschnitte

Codeausschnitt 1: platformio.ini	24
Codeausschnitt 2: Includes	28
Codeausschnitt 3: Queue zum Speichern anlegen	29
Codeausschnitt 4: Definition serielle Schnittstelle	29
Codeausschnitt 5: Frequenzbereich	29
Codeausschnitt 6: Pins Deklaration	29
Codeausschnitt 7: Schlaf Funktion	30
Codeausschnitt 8: Setup Funktion	31
Codeausschnitt 9: Verbindungsaufbauüberprüfung	31
Codeausschnitt 10: Funktion sentence_sep	32
Codeausschnitt 11: Extrahierung der der Daten	32
Codeausschnitt 12: Loop Funktion	33
Codeausschnitt 13: Leerzeichen entfernen	33
Codeausschnitt 14: Aufteilung der GPS-Daten	33
Codeausschnitt 15: Erstellung des JSON-Strings	33
Codeausschnitt 16: GPS-Daten in den FIFO-Speicher einfügen	33
Codeausschnitt 17: JSON-String am Monitor ausgeben	34
Codeausschnitt 18: Verbindungsüberprüfung	34
Codeausschnitt 19: Datenpaket aus FIFO entnehmen	34
Codeausschnitt 20: Finale Ausgabe am seriellen Monitor	34
Codeausschnitt 21: Beispiel vom seriellen Monitor	34
Codeausschnitt 22: Sendevorgang startet	35
Codeausschnitt 23: Daten werden an Empfänger gesendet	35
Codeausschnitt 24: Sendevorgang wird beenden	35
Codeausschnitt 25: Aufruf Schlaffunktion	35
Codeausschnitt 26: Include-Files	37
Codeausschnitt 27: Variablendeklaration	38
Codeausschnitt 28: Funktion setup_receiver()	38
Codeausschnitt 29: Funktion loop_receiver()	38
Codeausschnitt 30: Getter	39
Codeausschnitt 31: Header-Datei Receiver	39
Codeausschnitt 32: Bibliotheken Socket	41
Codeausschnitt 33: Definition WLAN	42
Codeausschnitt 34: Funktion notFound()	42
Codeausschnitt 35: Funktion onEvent()	43
Codeausschnitt 36: Funktion start_socket()	44
Codeausschnitt 37: Funktion update_webpage()	45
Codeausschnitt 38: Konvertierung in JsonString	45
Codeausschnitt 39: Funktion send_data()	45
Codeausschnitt 40: Header-File Socket	46
Codeausschnitt 41: HTML Code Überschrift	47
Codeausschnitt 42: Erstellung der Tabelle	47
Codeausschnitt 43: Variable gateway	48
Codeausschnitt 44: EventListener	48
Codeausschnitt 45: Funktion onLoad()	48
Codeausschnitt 46: Funktion init()	48
Codeausschnitt 47: Funktion onMessage()	49
Codeausschnitt 48: Zuweisung HTML-Element	49

Codeausschnitt 49: Installation SQLite.....	52
Codeausschnitt 50: Befehl Datenbankerstellung.....	52
Codeausschnitt 51: Befehl für die richtige Datenbank.....	52
Codeausschnitt 52: SQL-Befehl Tabellenerstellung.....	52
Codeausschnitt 53: SQL-Befehl auslesen	52
Codeausschnitt 54: Kommandos um Raspberry Pi upzugraden	53
Codeausschnitt 55: Kommando für die Qt Creator Installation.....	53
Codeausschnitt 56: Bibliotheken Qt Creator.....	54
Codeausschnitt 57: Verbindung zu Datenbank	55
Codeausschnitt 58: Einlesen in Datenbank	55

13.3 Abbildungsquellen

Abbildung 4: <https://german.alibaba.com/product-detail/TTGO-Meshtastic-T-Beam-V1-1-1600129636259.html>

Abbildung 5: : <https://pcbartists.com/design/embedded/esp32-c3-gpio-notes-strapping-pins/>

Abbildung 6: <https://store.sigma.com/neo-6m-0-001-module.html>

Abbildung 7:
http://www.lilygo.cn/prod_view.aspx?TypeId=50003&Id=1319&FId=t3:50003:3

Abbildung 8: <https://www.distrelec.at/de/raspberry-pi-modell-gb-ram-raspberry-pi-raspberry-pi-model/p/30085264>

Abbildung 9: <https://www.tinohempel.de/info/info/netze/topologie.htm>

Abbildung 18: <https://studylibde.com/doc/6828004/sql-befehle-1>

14. Begleitprotokoll gemäß § 9 Abs. 2 PrO

14.1 Begleitprotokoll Samuel Neureiter

Name: Hr. Samuel Neureiter

Diplomarbeitstitel: CTrack

KW	Beschreibung	Zeitaufwand
38	Diplomarbeitsantrag schreiben	10h
41	Projektplanung, Aufgabenverfeinerung	10h
43	Bibliotheken in Visual Studio Code einrichten	10h
44	LoRa kennenlernen	10h
45	LoRa kennenlernen	10h
46	Sender mit Empfänger verbinden	10h
49	Raspberry Pi und Datenbanksoftware kennenlernen	10h
50	Raspberry Pi einrichten	10h
51	Datenbank erstellt	10h
3	WebSocket und Qt Creator kennenlernen	10h
4	WebSocket aufsetzen	10h
5	WebSocket aufsetzen	10h
6	Webseite erstellen	10h
8	WebServer mit Webseite verknüpfen	10h
9	Qt Creator mit SQLite verknüpfen	10h
10	Qt Creator mit SQLite verknüpfen	10h
11	Diplomarbeit schreiben	10h
12	Diplomarbeit schreiben	10h

KW ... Kalenderwoche

---- ... Semesterabschnitt

14.2 Begleitprotokoll Marvin Klabacher

Name: Marvin Klabacher
Diplomarbeitstitel: CTrack

KW	Beschreibung	Zeitaufwand
38	Diplomarbeitsantrag schreiben	10h
41	Projektplanung, Aufgabenverfeinerung	10h
43	Bibliotheken in Visual Studio Code einrichten	10h
44	LoRa kennenlernen	10h
45	LoRa kennenlernen	10h
46	Sender mit Empfänger verbinden	10h
47	Sender mit Empfänger verbinden	10h
49	Aufnahme der GPS-Daten	10h
50	Programmierung zum Senden der Daten	10h
51	Programmierung zum Senden der Daten	10h
4	Aufteilung der GPS-daten	10h
5	Umwandlung in JSON-Format	10h
6	Umwandlung in JSON-Format	10h
8	Umwandlung in JSON-Format	10h
9	Speicherung der Daten	10h
10	Speicherung der Daten	10h
11	Diplomarbeit schreiben	10h
12	Diplomarbeit schreiben	10h

KW ...Kalenderwoche

---- ... Semesterabschnitt