Abstract

Diese Arbeit wurde vom Studenten Martin Alexander Möller erstellt.

Der Titel der Arbeit lautet: „Entwicklung eines automatisierten Kfz-Fahrtenbuchs durch die Verwendung eines Raspberry Pis“.

Bei der Arbeit handelt es sich um eine Bachelorarbeit

Die vorliegende Arbeit beschreibt die Entwicklung eines automatisierten Fahrtenbuchs mit einem Raspberry Pi. Dieser wird mit einem GPS-Empfänger und einem LTE-Modem ausgestattet. Mit dem GPS-Empfänger wird die Position mitgeschrieben. Anschließend wird aus den Positionsdaten die gefahrene Route berechnet und über das Modem an den Firmenserver gesendet. Ein Theorieteil hat zum Ziel die technischen Grundlagen, die zum Nachvollziehen der Arbeit notwendig sind, zu vermitteln. Darauf folgt die Vorstellung und Erläuterung der entwickelten Schaltung. Zum Schluss der Arbeit wird ein Fazit gezogen und es werden weiterführende Arbeiten vorgeschlagen.

Die Arbeit wurde im Jahr 2016 fertiggestellt.

Bernburg
Dessau
Köthen

Hochschule Anhalt
Anhalt University of Applied Sciences

emw
Fachbereich
Elektrotechnik, Maschinenbau
und Wirtschaftsingenieurwesen

Bachelorarbeit

zur Erlangung des akademischen Grades
Bachelor of Engineering (B. Eng.)

Martin Alexander Möller

Vorname Nachname

Medientechnik, 2011, 4055168

Studiengang, Matrikel, Matrikelnummer

Thema:

Entwicklung eines automatisierten Kfz-Fahrtenbuchs durch die Verwendung eines Raspberry Pis.

Prof. Dr. Steffen Strauß

Vorsitzender der Bachelorprüfungskommission

Prof. Dr. Steffen Strauß

1. Prüfer

Prof. Dr. Michael Brutscheck

2. Prüfer

02.11.2016

Abgabe am

Inhaltsverzeichnis

Tabellenverzeichnis

Abbildungsverzeichnis

Formelverzeichnis

Abkürzungsverzeichnis

API	Application Programming Interface
BFH	Bundesfinanzhof
C/A	code coarse / acquisition code
CPU	Central Processing Unit
EStG	Einkommenssteuergesetz
GLONASS	Globalnaja nawigazionnaja sputnikowaja sistema
GPIO	General Purpose Input Output
GPS	Navstar Global Positioning System
GPSD	Global Positioning System Daemon
GPX	GPS Exchange Format
HDMI	High Definition Multimedia Interface
HTTP	Hypertext Transfer Protocol
I2C	Inter-Integrated Circuit
JSON	JavaScript Object Notation
LTE	Long Term Evolution
microSD	Micro Secure Digital Memory Card
NMEA	National Marine Electronics Association
PHP	PHP: Hypertext Preprocessor
PPPD	Point-to-Point Protocol daemon
RMC	Recommended Minimum Navigation Information
SGML	Standard Generalized Markup Language
SIM	subscriber identity module
TCP	Transmission Control Protocol
TTL	Transistor-Transistor-Logik

UART	Universal Asynchronous Receiver Transmitter
URI	Uniform Resource Identifier
USB	Universal Serial Bus
USV	Unterbrechungsfreihe Strom Versorgung
WGS84	World Geodetic System 1984
XML	Extensible Markup Language

1 Einleitung

Wer heute ein Dienstfahrzeug nutzt, das teilweise privat verwendet wird, muss dieses in Deutschland versteuern. Bei Autos, die zu mindestens 50% betrieblich genutzt werden, verwendet das Finanzamt die sogenannte 1%-Regel, um die Höhe der Steuerschuld zu berechnen. Definiert ist die Besteuerung im Einkommenssteuergesetz (EStG) Paragraph 6. Dieser besagt über die 1%-Regel in Absatz 1 Nr. 4 [1]:

„Die private Nutzung eines Kraftfahrzeugs, das zu mehr als 50 Prozent betrieblich genutzt wird, ist für jeden Kalendermonat mit 1 Prozent des inländischen Listenpreises im Zeitpunkt der Erstzulassung zuzüglich der Kosten für Sonderausstattung einschließlich Umsatzsteuer anzusetzen […].“

Auch bei Gebrauchtwagen bezieht sich diese Angabe auf den Listenpreis eines Neuwagens. Wird das Fahrzeug auch zur Fahrt in die Arbeit verwendet, erhöhen sich die Kosten anteilig zur gefahrenen Kilometerzahl. Paragraph 8 des EStG schreibt dazu in Absatz 2 [2]:

„Kann das Kraftfahrzeug auch für Fahrten zwischen Wohnung und erster Tätigkeitsstätte […] genutzt werden, erhöht sich der Wert in Satz 2 für jeden Kalendermonat um 0,03 Prozent des Listenpreises […] für jeden Kilometer der Entfernung zwischen Wohnung und erster Tätigkeitsstätte […].“

Die 1%-Regel entspricht einer Schätzung der privaten Nutzung des Dienstfahrzeugs.

Eine Alternative zur 1%-Regel stellt das Führen eines Fahrtenbuchs dar. Hierzu steht im Paragraph 6 des EStG Absatz 1 Nr. 4 [1]:

„Die private Nutzung kann abweichend von Satz 2 mit den auf die Privatfahrten entfallenden Aufwendungen angesetzt werden, wenn die für das Kraftfahrzeug insgesamt entstehenden Aufwendungen durch Belege und das Verhältnis der privaten zu den übrigen Fahrten durch ein ordnungsgemäßes Fahrtenbuch nachgewiesen werden […].“

Der Begriff Fahrtenbuch ist im Gesetzestext nicht näher definiert. Eine Klage vor dem Bundesfinanzhof (BFH) im Jahr 2006 hat sich mit dem Fall beschäftigt, woraufhin die Anforderungen an ein Fahrtenbuch im Urteil definiert worden sind [3]. Demnach sind im Fahrtenbuch jeweils das Datum, der Kilometerstand, das Ziel der Reise und der besuchte Geschäftspartner sowie gesondert private Fahrten festzuhalten. Aus diesen Angaben kann der Anteil an privater Nutzung des Fahrzeugs bestimmt und die Steuerschuld in Relation dazu berechnet werden. Oftmals ist es steuertechnisch deutlich günstiger, ein Fahrtenbuch zu führen.

Wird das Fahrtenbuch wegen Widersprüchlichkeiten bzw. falscher Führung nicht steuerlich anerkannt, wird die 1%-Regelung angewendet. Dies wurde in einem Urteil des Bundesfinanzhofs im Jahr 2013 bekräftigt [4]. Um das Fahrtenbuch möglichst fehlerlos zu führen, sollte nach jeder Fahrt sofort ein Eintrag vorgenommen werden.

Die Firma NC3, gegründet 1998, übernimmt vielfältige Aufgaben im Fachgebiet Streaming und macht Gebrauch von einem Fahrtenbuch für ihre Dienstfahrzeuge. Oft besteht die Arbeit darin, zu einem Kunden zu fahren, ein Event zu filmen und als Livestream im Internet zur Verfügung zu stellen. Für das Streamen wird ein eigenes weltweites Content-Delivery-Network (CDN) verwendet. Diese Bachelorarbeit beschäftigt sich damit, ein automatisches Fahrtenbuch zu entwickeln und damit eine Basis für das Nachtragen von vergessenen Fahrtenbucheinträgen zu schaffen. Dabei werden erst in einem Theorieteil die Hintergründe der verwendeten Technik erläutert und daraufhin die Umsetzung beschrieben.

2 Theoretische Grundlagen

Die Aufgabe der Bachelorarbeit ist es, ein automatisiertes Fahrtenbuch zu entwickeln. In diesem sollen die gefahrene Route des Autos sowie die gefahrene Kilometerzahl einsehbar sein. Des Weiteren sollen die gefahrenen Routen des Dienstfahrzeugs auf eine Art und Weise erfasst werden, in der der Fahrer selbst nicht tätig werden muss. Die entwickelte Schaltung soll in der Lage sein zu erkennen, wann eine Route beginnt und endet. Ermittelt werden sollen die Anfangs- und Endposition des Fahrzeugs sowie die gefahrene Distanz. Die Daten sollen an einem Ort zentral gespeichert werden und über einen Internetanschluss verfügbar sein. Dafür müssen die Daten von der Schaltung an den Firmenserver übertragen werden. Dies ist mit möglichst geringem Datenvolumen zu realisieren, um die Übertragungskosten so gering wie möglich zu halten. Eine sofortige Übertragung nach Ende einer Route ist wünschenswert aber nicht zwingend notwendig, da, abhängig von der Position des Wagens, eventuell keine Mobilfunkverbindung möglich ist. In diesem Fall ist die Route nachträglich an den Firmenserver zu übermitteln, wobei die ursprüngliche Reihenfolge beizubehalten ist. Die Schwerpunkte der Arbeit lassen sich damit in fünf Arbeitsschritte unterteilen:

- Aufnehmen der Position des Firmenwagens
- Ausarbeiten und Speichern der gefahrenen Routen aus den Positionsdaten
- Übertragung der Routen an den Firmenserver
- Eintragen der Route in die Datenbank
- Sicherstellen des automatischen Ablaufs

2.1 Stand der Technik

In allen modernen Smartphones sind heutzutage GPS-Empfänger verbaut. Sie dienen der Lokalisierung des eigenen Standorts in Kartendiensten wie beispielsweise Google Maps oder Open Street Map. Im Fall eines Diebstahls aktivieren gestohlene Smartphones den GPS-Empfänger und erlauben das Finden des Gerätes. Geldtransporter senden ständig ihre GPS-Position an die Zentrale und auch bei dem beliebten Smartphone-Spiel Pokemon Go spielt der GPS-Empfänger eine entscheidende Rolle. GPS-Empfänger sind heutzutage schon ab ca. 10€ erhältlich und bieten sich damit für Projekte an, in denen die Positionsbestimmung eine Rolle spielt.

Zur Interpretation der empfangenen GPS-Daten wird ein Computer benötigt. Navigationsgeräte im Auto haben diese bereits integriert. Da hierfür nicht viel Rechenleistung notwendig ist, genügen Single-Board-Computer. Ein grober Überblick der verfügbaren Single-Board-Computer ist in Tabelle *1* zusammengestellt. Den meisten Geräten ist gemeinsam, dass sie eine General Purpose Input Output (GPIO)-Leiste, Universal Serial Bus (USB)-Anschlüsse, einen High Definition Multimedia Interface (HDMI)-Anschluss, einen Micro Secure Digital Memory Card (microSD)-Slot haben und durch Micro-USB versorgt werden. Um eine Vergleichbarkeit zu ermöglichen, sind jeweils die Central Processing Unit (CPU) und der Arbeitsspeicher genannt. In der Arbeit kommt ein Raspberry Pi 2B zum Einsatz. Im weiteren Verlauf der Arbeit wird abkürzend der Begriff Raspberry Pi für das Modell Raspberry Pi 2B verwendet.

Tabelle 1: Übersicht der verfügbaren Single-Board-Computer		
Name	Preis Modell	Eigenschaften
Raspberry Pi	Ca. 40 € Raspberry Pi 2B	- Große Fangemeinde → gut dokumentiert - Bekanntester Single-Board-Computer - ARM Cortex-A7 Quad-Core 900MHz CPU - 1GB Arbeitsspeicher [5]
Banana Pi	Ca. 90 € Banana Pi-M3	- Sata-Anschluss für Festplatten - Ähnlicher Aufbau wie Raspberry Pi - ARM Cortex-A7 Octa-Core 1.8GHz CPU - 2GB Arbeitsspeicher [6]
Odroid	Ca. 50 € Odroid-C2	- Große Auswahl an Modellen - ARM Cortex A53 Quad-Core 1,5 GHz CPU - 2 GB Arbeitsspeicher [7]
Hummingboard	Ca. 100-170€ € HB Edge	- Hardware wählbar durch System-on-Module - i.MX6 Single- bis Quad-Core CPU - 512 MB bis 4GB Arbeitsspeicher [8]
Beagle Board	Ca. 65 € BeagleBone Black	- Freies Hardware-Design - AM335x ARM Cortex-A8 1GHz CPU - 512 MB Arbeitsspeicher [9] [10]
Arduino	Ca. 35 € Genuino Mega 2560 Rev3	- Ist ein Single-Board-Microcontroller - ATmega2560 MicroController - Weltweite, große Community → gut dokumentiert [11]

-

2.2 Hypertext Transfer Protocol (HTTP)

2.2.1 Einführung

HTTP wurde geschaffen, um die Kommunikation zwischen Servern und Clients zu ermöglichen. Zur Übertragung wird das Transportprotokoll Transmission Control Protocol (TCP) verwendet. HTTP gilt als „core protocol of the World Wide Web“ und wird von der Internet Engineering Task Force (IETF) HTTP Working Group gepflegt und weiterentwickelt [12]. HTTP

ist ein zustandsloses Protokoll, was bedeutet, dass Anfragen unabhängig voneinander betrachtet werden. Als HTTP-Client gilt dabei jedes Programm, das eine Verbindung zu einem Server aufbaut, um eine HTTP-Anfrage zu senden. Ein HTTP-Server ist ein Programm, das Verbindungen akzeptiert, um HTTP-Anfragen anzunehmen und durch Senden einer Antwort zu bearbeiten [13]. Obwohl im Mai 2015 die Version HTTP/2.0 veröffentlicht wurde, wird sie nur von ca. 10% aller Webseiten verwendet (Stand Oktober 2016) [14] [15]. Die meistverwendeten Anfrage-Methoden von HTTP sind POST und GET.

2.2.2 Aufbau der HTTP-Kommunikation

Sowohl Anfragen des Clients als auch Antworten des Servers folgen dabei folgendem Aufbau [13, S. 19 f.]:

```
Start-Zeile
Header
leere Zeile
Body
```

Die Start-Zeile definiert dabei, ob es sich um eine Anfrage oder um eine Antwort handelt. Die Start-Zeile einer Anfrage (Request Line) folgt folgendem Aufbau:

```
Methode Ziel Version CRLF
```

Methode	Die verwendete HTTP-Anfrage-Methode wird durch den Parameter Methode angegeben wie beispielsweise *POST* oder *GET*. Eine Liste der verfügbaren Methoden ist im offiziellen HTTP-Standard verfügbar [16, S. 24].
Ziel	Mit dem Parameter *Ziel* wird die Ressource auf dem Server angegeben, auf die die Anfrage angewendet werden soll.
Version	Die *Version* bezeichnet die verwendete HTTP-Version.
CRLF	*CRLF* steht für Carriage Return Line Feed und bezeichnet den Code für Zeilenvorschub, also ein Sprung in die nächste Zeile. Die Bezeichnung bezieht sich auf den Wagenrücklauf bei Schreibmaschinen, wo erst an den Anfang der Zeile und dann in die nächste Zeile gesprungen wurde.

Die Start-Zeile einer Antwort des Servers (Status Line) ist wie folgt aufgebaut:

```
Version Status-Code reason-phrase CRLF
```

Status-Code	Der Status-Code ist ein dreistelliger Code. Er beschreibt das Ergebnis der Interpretation der Anfrage des Servers. Eine Übersicht der existierenden Status-Codes ist im offiziellen HTTP-Standard verfügbar [16, S. 49].
reason-phrase	Die reason-phrase enthält eine Textbeschreibung des Status-Codes.

Der Header-Bereich kann keine bis mehrere Header enthalten. Header enthalten Meta-Informationen über den body und definieren die Parameter einer HTTP-Übertragung. So kann ein Header beispielsweise die akzeptierte Zeichencodierung (Accept-Charset) oder die Länge des Bodys in Byte (Content-Length) angeben. Eine Liste der definierten Header ist auf der Homepage der Internet Assigned Numbers Authority (IANA) verfügbar [17].

Der body einer Nachricht kann Nutzdaten enthalten, die es zu übertragen gilt, wie beispielsweise eine angeforderte Webseite. Er ist bei HTTP-Übertragungen optional [13, S. 28].

2.2.3 GET-Anfragen

Die GET-Anfrage eines Clients fordert eine bestimmte Ressource auf dem Server an. So wird beispielsweise das Aufrufen von Webseiten durch einen Browser mit HTTP realisiert. Der Client (Browser) schickt eine Anfrage (request) an den Server und erhält als Antwort die gewünschte Webseite [16, S. 24].

2.2.4 POST-Anfragen

Die POST-Anfrage eines Clients fordert, dass der body der Anfrage nach einer übergebenen Semantik bearbeitet wird. Dies wird zum Beispiel verwendet, um

- eingegebene Formulardaten einer Webseite an den Prozess auf dem Server zu übergeben, der sie interpretiert oder abspeichert.
- einen Beitrag an einen Blog oder ein Forum zu senden.

- eine neue Ressource auf dem Server zu erstellen.
- eine bereits vorhandene Ressource auf dem Server mit Daten zu erweitern.

Der Server beantwortet HTTP-Anfragen mit einem HTTP-Statuscode. Dieser gibt an, ob die Übertragung erfolgreich war.

2.3 PHP: Hypertext Preprocessor (PHP)

2.3.1 Einführung

PHP ist der Nachfolger der 1995 von Rasmus Lerdorf geschaffenen Programmiersprache PHP/FI. Sie bestand aus einem Set von Pearl-Skripten, mit denen die Zugriffe auf seine Homepage erfasst werden sollten. Daraus leitet sich der ursprüngliche Name Personal Homepage Tools/Forms Interpreter ab. Lerdorf hat den Sourcecode später veröffentlicht, so dass er von jedem benutzt und weiterentwickelt werden konnte. Mit der Version 3.0 wurde die Sprache zu PHP umbenannt, was eine rekursive Abkürzung für PHP: Hypertext Preprocessor darstellt [18].

Heute ist PHP eine weit verbreitete Open Source-Skriptsprache, die durch ihren breiten Funktionsumfang für den allgemeinen Gebrauch bestimmt ist [19]. Die drei Hauptgebiete für die Nutzung von PHP-Skripten sind serverseitige Programmierung, Kommandozeilenprogrammierung und das Schreiben von Desktop-Applikationen. Des Weiteren wird die Arbeit mit vielen verschiedenen Datenbanksystemen unterstützt. PHP kann auf allen gängigen Betriebssystemen verwendet werden und unterstützt die meisten gebräuchlichen Webserver [20]. Einen Überblick über die verfügbaren Funktionen bietet die offizielle Funktionsreferenz [21]. Die neueste Hauptversion ist 7.0 (Stand 18. Oktober 2016) und wurde am 3.12.2015 veröffentlicht [22]. Sowohl der Raspberry Pi als auch der Firmenserver arbeiten mit Version 5. In der vorliegenden Arbeit werden drei PHP-Skripte verwendet, von denen zwei der Kommandozeilenprogrammierung und eines der serverseitigen Programmierung zuzuordnen sind.

2.3.2 Kommandozeilenprogrammierung

PHP-Skripte der Kommandozeilenprogrammierung benötigen keinen Webserver und werden direkt auf dem Computer ausgeführt [23]. Damit das PHP-Skript von der Konsole ausgeführt

werden kann, benötigt es einen PHP-Parser. Ist dieser installiert, kann ein Skript mit folgendem Aufruf gestartet werden:

```
php [options] [-f] <file> [--] [args...]
```

Eine Übersicht über die möglichen Parameter ist im offiziellen PHP-Handbuch verfügbar [24].

2.3.3 Serverseitige Programmierung

Um ein serverseitiges Skript zu verwenden, wird ein Webserver mit installiertem PHP-Modul benötigt, auf den das Skript hochgeladen wird. Auf dem Webserver kann es beispielsweise von einem Browser aufgerufen werden [23].

2.3.4 POST-Anfragen mit PHP

Die Übertragung einer Variablen per POST-Anfrage durch PHP soll anhand folgenden Beispiels erläutert werden:

```
$options = array(
           'http' => array(
'header'   => "Content-type: application/x-www-form-urlencoded\r\n",
'method'   => 'POST',
'content'  => http_build_query($ubergabe)
           )
     );
$context  = stream_context_create($options);
```

Wird eine HTTP-Anfrage per PHP realisiert, geschieht dies durch Verwendung eines Streams.

Ein Stream in PHP ist, der einfachsten Definition nach, eine Ressource, die linear gelesen oder beschrieben werden kann. Er besteht aus einem Wrapper und einem Ziel und wird wie folgt referenziert: Streamwrapper://Ziel.

Streamwrapper sind Schnittstellen, die die Kommunikation zwischen Protokollen und Streams übernehmen. Der hier verwendete Streamwrapper http definiert den Stream als HTTP-Stream.

Das Ziel ist abhängig vom verwendeten Streamwrapper. Es besteht üblicherweise aus einem Hostnamen sowie angefügtem Pfad der Zieldatei [25].

Für den Stream wird ein Kontext definiert. Dafür wird die Variable $options geschaffen, in der in einem assoziativen Array Streamwrapper hinterlegt sind. In diesen sind ebenfalls assoziative Arrays, die die Kontext-Optionen als Schlüssel-Wert Paare enthalten [26]. Assoziative Arrays bezeichnen Arrays, deren Schlüssel eine Bezeichnung haben, wie in diesem Fall „http“ oder „header“. Eine Liste der Kontext-Optionen ist in der offiziellen PHP-Dokumentation verfügbar [27].

Der Header besagt, dass es sich beim Inhalt der Anfrage um den Internet media type application/x-www-form-urlencoded handelt. Dies ist der Standard-Typ für den Upload von Daten eines Formulars an einen Server [28].

Die Methode definiert den HTTP-Anfrage-Typ als POST.

Content enthält die zu übertragenden Daten. In diesem Fall wird das Array $ubergabe, in dem eine Route enthalten ist, mit der Funktion http_build_query zu einem URL-kodierten Query-String transformiert, damit es übertragen werden kann [29].

Anschließend wird aus den Optionen mit der Funktion stream_context_create ein Stream Kontext erstellt. Ein Stream Kontext enthält Parameter und wrapperspezifische Optionen, die das Verhalten eines Streams bestimmen [30].

```
$url = 'http://IP-des-Servers/receiver.php';

$result = (file_get_contents($url, false, $context));
```

Daraufhin wird ein Uniform Resource Identifier (URI) des Server-Skripts in der Variablen $url gespeichert und die Übertragung durchgeführt.

`file_get_contents` Mit dieser Funktion wird eine Datei in einen String gelesen. Anstatt einer Datei wird die URI des Skripts auf dem Firmenserver als Quelle verwendet. Mit *false* wird auf eine Suche nach der Quelle in einem mitzugebenden Pfad verzichtet. Das Einlesen folgt dabei den in $context festgelegten Optionen. Konkret wird also beim Einlesen der URI ein Array per POST-Anfrage übertragen. Der eingelesene String entspricht der Antwort des Server-Skripts und wird in der Variable $result gespeichert [31].

2.4 Globale satellitengestützte Navigationssysteme

2.4.1 Einführung

Es existieren momentan zwei globale satellitengestützte Navigationsysteme: Das amerikanische Navstar Global Positioning System (GPS) und das russische Globalnaja nawigazionnaja sputnikowaja sistema (GLONASS), die jeweils unter militärischer Kontrolle stehen. Ein europäisches, zivil kontrolliertes, satellitengestütztes Navigationssystem namens Galileo ist im Aufbau und soll Ende 2016 mit Grundfunktionen verfügbar sein [32]. In dieser Bachelorarbeit wird das amerikanische GPS verwendet.

Das GPS ist seit 1978 im Einsatz und teilt sich in die drei Segmente Space Segment, Control Segment und User Segment auf.

2.4.2 GPS Space Segment

Das GPS besteht aus 24 Satelliten, die die Erde täglich jeweils zweimal in einer Höhe von ca. 20200 km umrunden. Der Erdorbit wird dabei in sechs gleich große Zonen unterteilt, in denen jeweils vier Satelliten stationiert sind. Damit wird sichergestellt, dass von jedem Punkt der Erde aus mindestens vier Satelliten verfügbar sind. Im Juni 2011 wurde die Anzahl der Satelliten im Rahmen der „Expandable 24" Einstellung auf 27 erhöht. Tatsächlich sind sogar 31 Satelliten im Orbit, um die Verfügbarkeit auch in Wartungsfällen zu gewährleisten [33].

GPS bietet dabei zwei verschiedene Modi an:

- Der Precise Position Service (PPS) steht nur dem amerikanischen Militär und seinen Verbündeten zur Verfügung. Es unterliegt keinen Einschränkungen.
- Der Standard Position Service (SPS) steht auch der zivilen Bevölkerung zur Verfügung. Bis zur Nacht des 1. Mai 2000 wurde die Genauigkeit des SPS künstlich durch die Eigenschaft „Selective Availability (SA)" verschlechtert [34]. Für zivile Benutzer wird durchgehend der coarse/acquisition code (C/A code) auf der L1-Frequenz 1575,42 MHz ausgestrahlt. Die enthaltenen Informationen sind unter anderem der Startpunkt der Übertragung und die momentane Position des Satelliten [35, S. 8].

2.4.3 GPS User Segment

Empfänger berechnen aus dem Zeitunterschied zwischen Sendezeit und Empfangszeit des Signals die Entfernung zu den Satelliten, das sogenannte SPS Signal Ranging Measurement. Werden die Informationen von mindestens vier Satelliten empfangen, kann die eigene Position bestimmt werden. Die berechnete Position bezieht sich auf das World Geodetic System 1984 (WGS84), welches einen der Erdform angenäherten Referenzellipsoiden zur Verfügung stellt [35, S. 33]. Nähere Informationen zum WGS84 sind in der offiziellen Publikation des Standards verfügbar [36].

2.4.4 GPS Control Segment

Das Kontroll-Segment besteht aus einem globalen auf der Erde befindlichen Netzwerk von Einrichtungen, die die Position der Satelliten verfolgen, ihre Übertragungen beobachten, Analysen durchführen und Daten an die Satelliten senden. 15 Monitor-Stations beobachten die Satelliten und senden die aufgenommenen Daten weiter an die Master-Control-Station. An vier der Monitorstationen sind Ground Antennas angebracht, die eine Kommunikation mit den Satelliten ermöglichen. Die Master-Control-Station in Colorado nimmt die Daten der Monitor Stations entgegen, kontrolliert die Konstellation der Satelliten und überwacht ihren Zustand [37].

2.4.5 Distanzberechnung mit Koordinaten

Das WGS84 spannt ein kartesisches Koordinatensystem über der Erde auf, aus denen die Winkelangaben geographische Länge (longitude) und Breite (latitude) berechnet werden (siehe Abbildung 1). Dabei fungieren jeweils ein Breitenkreis und ein Längskreis als Bezugskreise. Als Bezugsbreitenkreis wurde der Äquator gewählt, dessen Ebene durch den Erdmittelpunkt verläuft und der senkrecht zur Rotationsachse der Erde steht. Als Bezugslängenkreis hat sich der durch die beiden Pole und die Sternwarte Greenwich bei London verlaufende Längenkreis durchgesetzt [38].

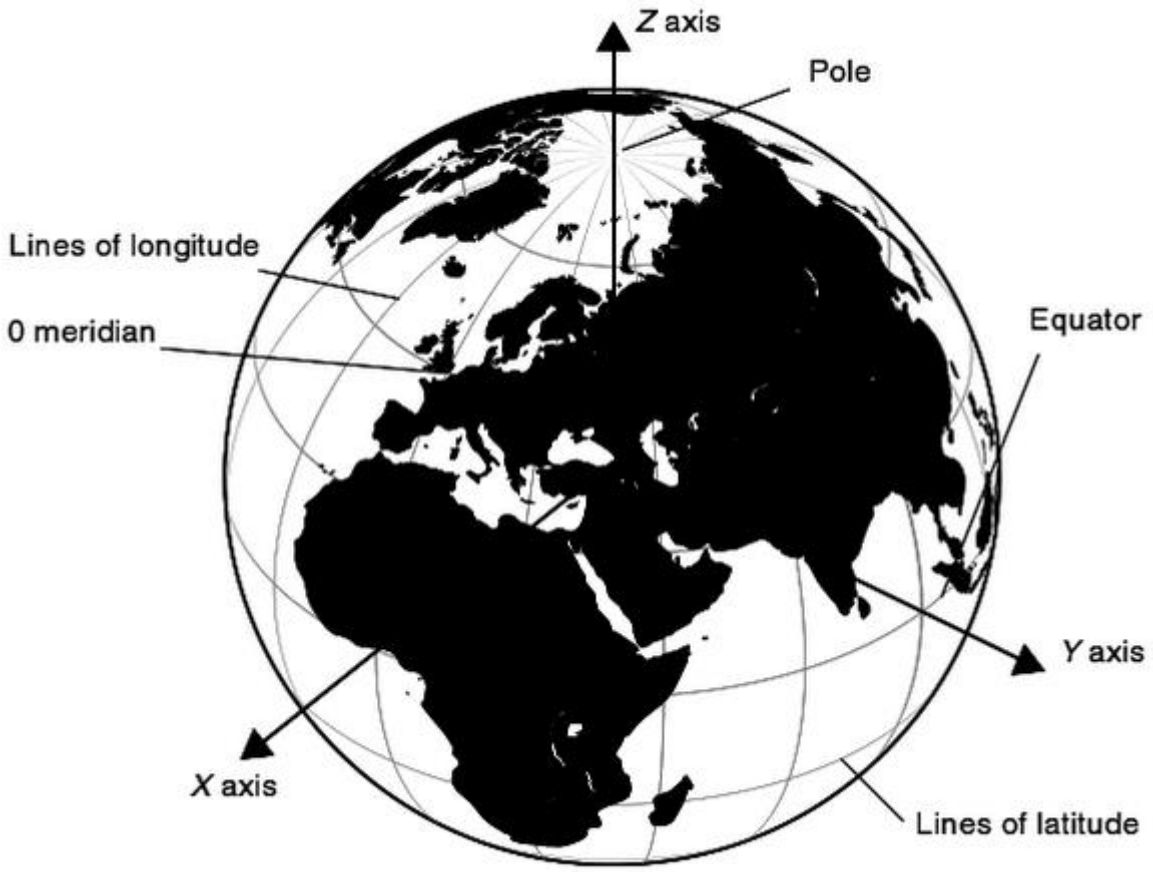

Abbildung 1: Koordinatensystem der Erde
(Quelle: Bartlett, D: The Cambridge Wireless Essentials Series: Essentials of Positioning and Location Technology, 2013, [39, S. 14].)

Da der in der Bachelorarbeit verwendete GPS-Empfänger die Koordinaten in Längen- und Breitengraden ausgibt, gilt es die Distanz zwischen zwei Punkten nach diesem System zu berechnen. Da geographische Länge und Breite im rechten Winkel zueinanderstehen, bilden sie ein rechtwinkliges Dreieck mit der gefahrenen Distanz als Hypotenuse. Die Distanz berechnet sich daher durch den Satz des Pythagoras mit Formel 1.

$$Distanz = \sqrt{gefahrene\ Länge^2 + gefahrene\ Breite^2}$$

Formel 1: Distanz nach Formel des Pythagoras

Breitenkreise sind parallel zum Äquator mit konstantem Abstand von 111,3 km pro Grad angeordnet. Damit berechnet sich die gefahrene geographische Breite durch Formel 2.

$$gefahrene\ Breite = 111{,}3\ km \cdot (Breite_{Start} - Breite_{Ziel})$$

Formel 2: Berechnung der gefahrenen geographischen Breite

Der Abstand zwischen den Längengraden ist nicht konstant. Am Äquator besteht ebenfalls ein Abstand von 111,3 km, dieser nimmt jedoch Richtung der Pole ab bis zu einem Abstand von 0. Diese Entwicklung wird durch den Kosinus beschrieben. Damit berechnet sich die gefahrene geographische Länge durch Formel 3, wobei zur Berechnung des Abstands die mittlere geographische Breite zwischen den zwei Punkten verwendet wird. Da in der Arbeit der Abstand von zwei Punkten, die mit einem Zeitunterschied von einer Sekunde aufgenommen werden, berechnet wird, ist der dadurch entstehende Fehler vernachlässigbar.

$$gefahrene\ Länge = 111{,}3\ km \cdot \cos\left(\frac{Breite_{Start} + Breite_{Ziel}}{2}\right) \cdot (Länge_{Start} - Länge_{Ziel})$$

Formel 3: Berechnung der gefahrenen geographischen Länge

2.4.6 GPS Exchange Format (GPX)

GPX ist ein Datenformat zum Austausch von GPS-Daten. Das Format leitet sich von der Extensible Markup Language (XML) ab, eine eingeschränkte Form der Standard Generalized Markup Language (SGML) [40]. Es wurde im Jahr 2002 von topografix veröffentlicht und ist inzwischen das Standard-Format zum Austausch von GPS-Daten geworden. Die aktuelle Version ist GPX 1.1 und wurde im August 2004 veröffentlicht [41]. In GPX-Dateien können verschiedene Typen von Daten gespeichert werden. So ist es möglich, Wegpunkte, Routen und Tracks abzuspeichern. Eine Route beschreibt hierbei eine Liste von Wegpunkten, die zu einem bestimmten Ziel führen und dient der Navigation. Das Aufzeichnen einer Strecke entspricht dem Typ Tracks und hat folgenden Aufbau [42]:

```
<gpx>
     <trk>
          <trkseg>
               <trkpt lat="51.322108" lon="12.378076">
                    [Informationen der einzelnen Trackpoints]
               </trkpt>
               [weitere Trackpoints]
          </trkseg>
     </trk>
</gpx>
```

Die Bedeutung der Elemente ist in Tabelle *2* erläutert.

Tabelle 2: Bedeutung der GPX-Elemente	
<gpx>	Das Root-Element der XML-Datei
<trk>	Kennzeichnet einen Track. Jeder Track besteht aus Segmenten.
<trkseg>	Kennzeichnet ein Segment. Jedes Segment besteht aus Trackpoints. Segmente stellen ununterbrochene Aufzeichnungen dar. Nach einer Unterbrechung werden die Daten in ein neues Segment gespeichert.
<trkpt>	Kennzeichnet einen Trackpoint. In Trackpoints sind die Koodinaten, Höhenangaben, Zeiten und Metadaten einzelner Punkte auf dem Track gespeichert.

(Quelle: topografix.com: GPX Schema, http://www.topografix.com/GPX/1/1/gpx.xsd [43])

2.5 Serielle Schnittstellen

2.5.1 Einführung

Serielle Schnittstellen übertragen Daten Bit für Bit nacheinander über eine Leitung. Das Gegenstück zur seriellen Schnittstelle ist die parallele Schnittstelle, die Bits gleichzeitig über mehrere Leitungen überträgt. Serielle Schnittstellen verwenden für das Senden und Empfangen jeweils eine Leitung. Diese werden als TX (transmit) und RX (receive) bezeichnet. Dabei wird die Sendeleitung der einen Schnittstelle mit der Empfangsleitung der anderen verbunden und umgekehrt. Dies erlaubt eine Kommunikation in beide Richtungen (vollduplex). Lange Zeit war der RS-232-Standard weit verbreitet und jeder Personal Computer damit ausgestattet. Heutzutage ist USB der meistverwendete Standard für serielle Übertragung. RS-232 wird nicht mehr in PCs verbaut, jedoch nach wie vor von Micro Controllern verwendet, da es ein erprobter und günstiger Standard ist. Des Weiteren gibt es viele günstige Komponenten, die mit RS-232 kommunizieren [44, S. 82 f.].

2.5.2 Universal Asynchronous Receiver Transmitter (UART)

UART bezeichnet eine Hardware, die der Realisierung serieller Schnittstellen dient. Oft wird sie mit dem Schnittstellen-Standard RS-232 verwendet. UART erlaubt es über eine Schnittstelle verschiedenste Peripheriegeräte wie Mäuse, Tastaturen oder Modems an einen PC oder Micro Controller anzuschließen. Die Übertragung erfolgt dabei asynchron, was bedeutet, dass es kein Taktsignal für Sender und Empfänger gibt. Stattdessen werden der Start und das Ende einer Übertragung durch START- und STOPP-Bits signalisiert [45, S. 130].

2.5.3 RS-232

RS-232 ist ein Übertragungs-Standard für seriellen Schnittstellen. Er definiert die Spannungspegel und den Ablauf der Übertragung. Dabei wird vor dem zu übertragenden Paket ein LOW-Level Start-Bit gesendet. Dies signalisiert dem Empfänger den Beginn der Übertragung. Daraufhin wird das Paket übertragen, wobei mit dem Least Significant Bit begonnen wird. Nach dem Paket kann ein optionales Paritäts-Bit gesendet werden. Daraufhin werden ein bis zwei HIGH-Level Stopp-Bits gesendet. Der Ablauf ist in Abbildung 2 skizziert. Jedes Bit wird gleich lange übertragen, wobei die Zeit von der Baudrate abhängt [45, S. 129 f.]. Zwischen den Übertragungen einzelner Pakete kommt es jeweils zu einer kurzen Pause. Diese signalisiert dem Empfänger, dass der nächste Übergang von HIGH (das Stopp-Bit der

letzten Übertragung) zu LOW (das Start-Bit) den Start der nächsten Übertragung darstellt. Damit werden Übergänge von HIGH zu LOW im Datenbereich nicht als Start-Bit erkannt, da die vorhergehende Pause fehlt [45, S. 130].

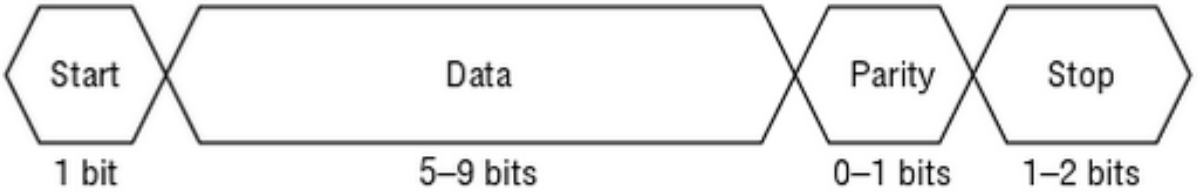

Abbildung 2: Ablauf einer RS-232-Übertragung
(Quelle: Langbridge, James A.: Arduino Sketches :Tools and Techniques for Programming Wizardry, Wiley, 2015, S. 85 [44, S. 85].)

RS-232 bietet verschiedene Einstellmöglichkeiten, welche die Parameter der Übertragung definieren. Es ist zwingend notwendig, dass zwei miteinander kommunizierende Schaltungen dieselben Einstellungen verwenden, da es sonst zu Fehlern bei der Übertragung kommt. Folgende Parameter können angepasst werden [44, S. 84 ff.]:

- Die Anzahl der Bits pro Paket entscheidet wie viele Bit pro Übertragung gesendet werden. Der normale Wert sind 8 Bit pro Übertragung, was einem Byte entspricht.
- Es können entweder ein oder zwei Stopp-Bits eingestellt werden.
- Das Paritäts-Bit kann aktiviert oder deaktiviert werden. Es ist von der Anzahl der logischen HIGH-Pegel der übertragenen Daten abhängig. Ist diese gerade, ist das Paritäts-Bit HIGH. Ist sie ungerade, ist das Paritäts-Bit LOW. Das Paritäts-Bit stellt eine Möglichkeit der Überprüfung der korrekten Übertragung dar.
- Die Baudrate ist die Geschwindigkeit der Übertragung. Eine Baudrate von 1 entspricht einer Übertragung von 1 Bit pro Sekunde. Übliche Baudraten sind 1200, 2400, 9600 und 115200 Baud.

Eine zu übertragende Nachricht wird der UART-Schaltung parallel zur Verfügung gestellt. Die UART-Schnittstelle teilt die zu übertragende Nachricht daraufhin in einzelne Bits auf und überträgt sie nacheinander (serielle Übertragung). Die empfangende UART-Schnittstelle setzt die empfangenen Bits wieder zu einem Byte zusammen und gibt dieses weiter [44, S. 84].

Der RS-232 Standard arbeitet mit Pegeln von -15 V bis +15 V. Dabei werden Pegel zwischen 3V und 15V als LOW, und Pegel zwischen -3 V und -15 V als HIGH interpretiert.

Der Raspberry Pi und der GPS-Empfänger arbeiten mit einer Betriebsspannung von 5V. In so einem Fall kann auch über Transistor-Transistor-Logik (TTL)-Pegel kommuniziert werden. Hierbei gilt eine Spannung von 0 V bis 0,8 V als LOW und 2.0 V bis 5 V als HIGH [46].

Dabei werden vier Leitungen verwendet, die eine Übertragung ermöglichen. Diese sind in Tabelle *3* dargestellt.

Tabelle 3: Signalleitungen des RS-232 Standards	
VCC	Über diese Leitung realisiert der Micro Controller die Stromversorgung des Peripheriegeräts.
GND	Gemeinsame Masse
TXD	Die Transmit Data Leitung sendet Daten.
RXD	Die Receive Data Leitung empfängt Daten.

Micro Controller und Peripheriegeräte werden nach der Nullmodem Schaltung verbunden. Diese besagt, dass TXD des einen UART mit RXD des anderen UART verbunden wird und umgekehrt [47, S. 206 f.].

3 Praktische Ausführung

3.1 Durchführung

In diesem Kapitel wird vorgestellt, wie die Arbeitsschritte umgesetzt werden:

Das Aufnehmen der Position des Firmenwagens wird durch einen Raspberry Pi realisiert, der die momentane Position mit Hilfe eines angeschlossenen GPS-Empfängers mitschreibt. Der Raspberry Pi wird über eine unterbrechungsfreie Stromversorgung (USV), die durch das Zündungsplus des Autos versorgt wird, mit Strom versorgt. Der Raspberry Pi wird parallel zur Zündung hoch- und runtergefahren, wobei die zwischengeschaltete USV ein sauberes Herunterfahren ermöglicht.

Auch zum Ausarbeiten und Speichern der gefahrenen Routen aus den Positionsdaten wird der Raspberry Pi verwendet. Dieser berechnet vor dem Herunterfahren aus den gesammelten Positionsdaten die gefahrene Route. Das Berechnen der Route wird von einem PHP-Skript durchgeführt.

Um die Übertragung der Routen an den Firmenserver zu ermöglichen, wird ein Long Term Evolution (LTE)-Modem per USB an den Raspberry Pi angeschlossen. Ein PHP-Skript stellt über diesen eine Internetverbindung über das Mobilfunknetz her und überträgt die Route.

Das Eintragen der Route in die Datenbank, sowie das Entgegennehmen der Übertragung werden durch ein weiteres PHP-Skript auf dem Server realisiert.

Das Sicherstellen des automatischen Ablaufs funktioniert über zwei feste Triggerpunkte.

- Nach dem Hochfahren des Raspberry Pi wird das Skript 1 auf dem Rasberry Pi über einen Eintrag im Autostart gestartet. Das Skript 1 beginnt das Aufnehmen der Position des Firmenwagens.
- Das Ausschalten der Zündung triggert das Umspringen der Versorgung auf den Sekundäreingang der USV sowie über ein Shell-Skript den Start von Skript 2 auf dem Raspberry Pi. Dieses berechnet die Route, überträgt die noch nicht gesendeten Routen und fährt anschließend den Raspberry Pi herunter.

Der komplette Durchgang einer Fahrt sieht damit folgendermaßen aus:

1. Der Fahrer startet das Auto und die Zündung wird aktiviert. Die PiUSV+ wird dadurch über den Primäreingang mit Strom versorgt und beginnt den Raspberry Pi mit Strom

zu versorgen. Dieser erhält dadurch eine steigende Flanke an der Versorgungsspannung und fährt hoch. Gleichzeitig beginnt die PiUSV+ den Akkumulator aufzuladen.

2. Nach erfolgreichem Hochfahren des Raspberry Pi wird per Autostart das PHP-Skript1 gestartet. Dieses startet den gpxlogger, welcher die Positionsdaten des GPS-Empfängers entgegennimmt und kontinuierlich in die Datei rohdaten.gpx schreibt. Dieser Zustand wird beibehalten, bis der Fahrer am Ziel ankommt und die Zündung ausschaltet.

3. Nach Ausschalten der Zündung wechselt die PiUSV+ zur Versorgung durch den Sekundäreingang. Durch den Wechsel wird über ein Shell-Skript das PHP-Skript2 gestartet. Dieses beendet den gpxlogger, berechnet aus den notierten Positionsdaten die gefahrene Route und fügt sie den zu übertragenden Routen in der Textdatei routen.txt hinzu. Daraufhin stellt es über das LTE-Modem eine Internetverbindung her und sendet alle noch nicht erfolgreich übertragenen Routen an den Server. Übertragene Routen werden in ein Archiv verschoben. Die Verbindung wird getrennt und der Raspberry Pi wird heruntergefahren.

4. Die USV erkennt, dass der Raspberry Pi heruntergefahren wurde und stellt die Spannungsversorgung durch den Akkumulator ein. Dadurch ist wieder eine steigende Flanke am Raspberry Pi möglich und eine neue Route kann begonnen werden.

Eine Übersicht über die Schaltung bietet die Skizze in Abbildung 3.

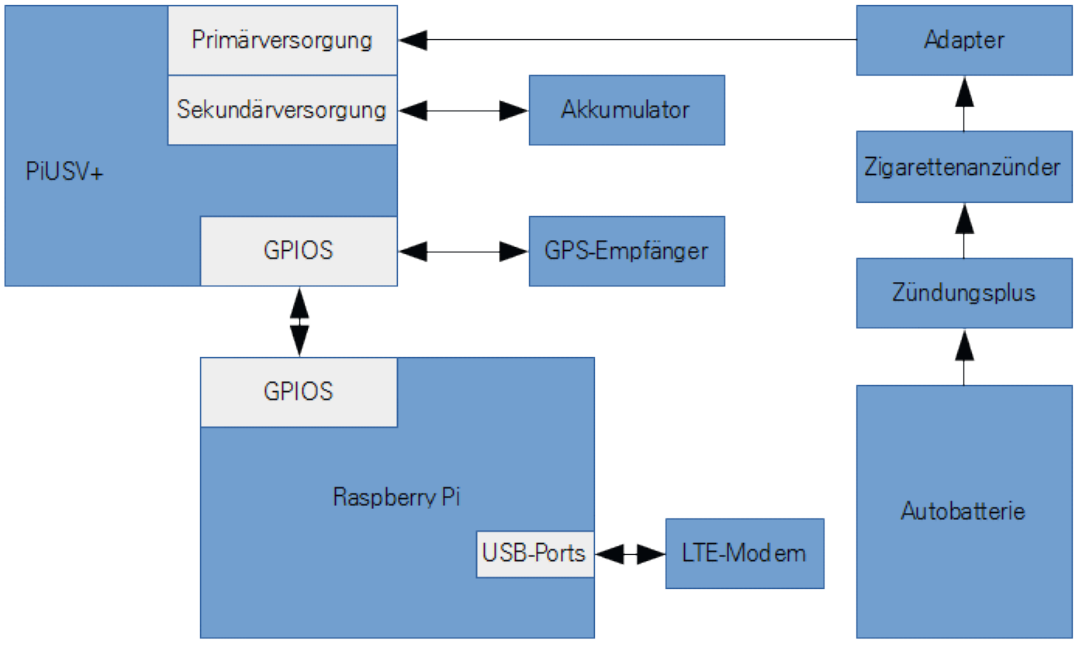

Abbildung 3: Skizze der Schaltung

3.2 Verwendete Komponenten

3.2.1 Raspberry Pi

Der verwendete Raspberry Pi ist ein Raspberry Pi 2B. Er ist ein Einplatinencomputer, alle Komponenten sind auf einer ca. kreditkartengroßen Platine zusammengefasst. Der Raspberry Pi ist das Herzstück der Schaltung an den alle Komponenten angeschlossen werden. Von den verfügbaren Anschlüssen werden ein USB-Anschluss und die GPIO-Leiste verwendet. Der USB-Anschluss wird verwendet, um ein LTE-Modem anzuschließen. Auf die GPIO-Leiste wird die USV gesteckt, die die Signale allerdings an den Raspberry Pi weitergibt. An die GPIOs der USV wird der GPS-Empfänger angeschlossen. Ein Bild des Raspberry Pi ist in Abbildung 4 vorhanden. Nähere Informationen und technische Daten des Raspberry Pi sind auf der Homepage verfügbar [48].

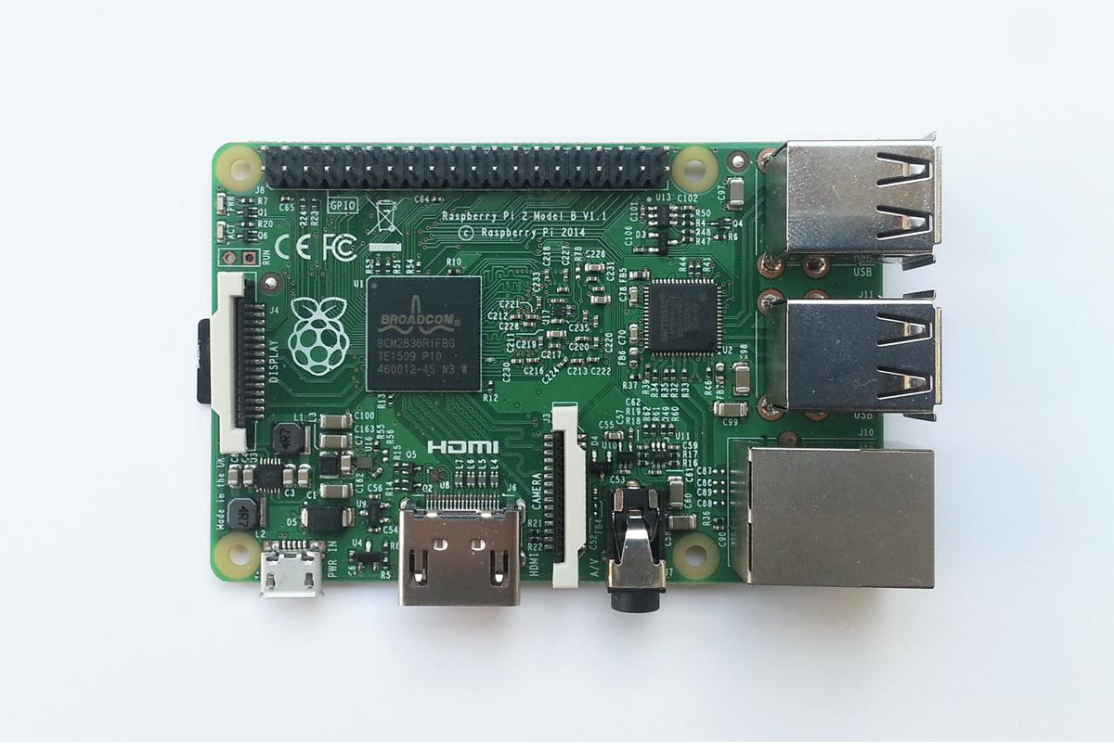

Abbildung 4: Raspberry Pi 2B

Autostart des Skript 1

Um den Start des Skript1 beim Systemstart zu realisieren, wird die Datei /etc/rc.local um folgenden Eintrag erweitert:

```
php /home/pi/Desktop/Sender/skript1.php
```

Die Datei rc.local führt in ihr hinterlegte Befehle zum Systemstart aus. Alle Befehle werden mit Root-Rechten ausgeführt.

3.2.2 Raspberry Pi USV PiUSV+

Die PiUSV+ ist eine Erweiterungsplatine für den Raspberry Pi und stellt eine speziell für den Raspberry Pi entwickelte USV dar. Sie wird auf die GPIOs des Raspberry Pi gesteckt und leitet die Signale an den GPIOs weiter an den Raspberry Pi. Die PiUSV+ besitzt zwei verschiedene Eingänge zur Spannungsversorgung. Der Primäreingang wird durch das Zündungsplus des Autos versorgt. An einem zweiten Eingang ist ein Akkumulator angeschlossen. Die PiUSV+ ist in Abbildung 5 dargestellt.

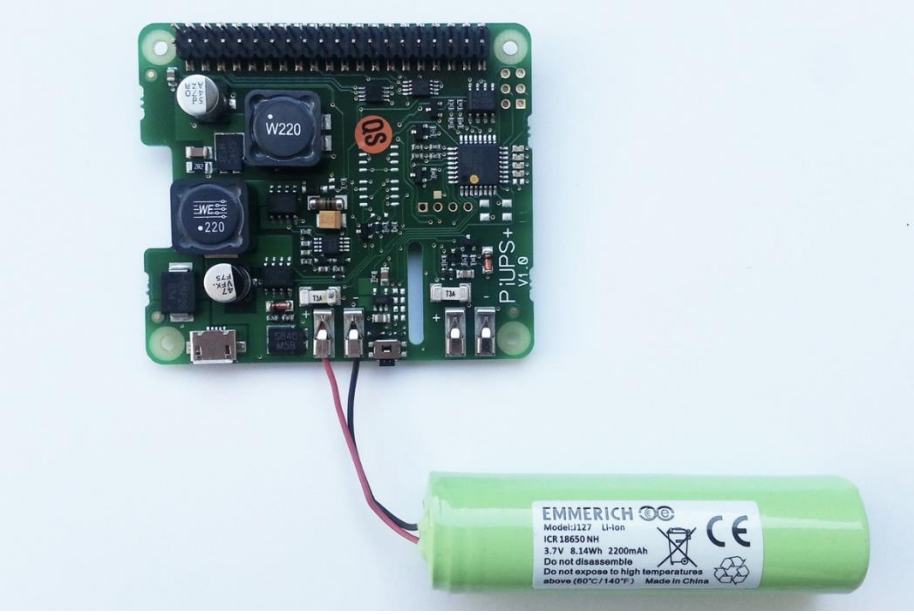

Abbildung 5: PiUSV+ mit angeschlossenem Akkumulator am Sekundäreingang

Solange die PiUSV+ durch den Primäreingang versorgt wird, wird der Akkumulator aufgeladen. Erfolgt durch den Primäreingang keine Spannungsversorgung mehr, passieren zwei Dinge:

1. Die Versorgung des Raspberry Pi schaltet fließend um auf die Sekundärversorgung (der Akkumulator).

2. Ein vorher bestimmtes Skript wird gestartet. Dieses Skript hat die Aufgabe, den Raspberry Pi herunterzufahren.

Auf dem Raspberry Pi ist *PiUSV-Monitor* installiert, ein spezielles Softwarepaket zur Konfiguration der PiUSV+. Dabei sind folgende Parameter einstellbar [49]:

- Der *ShutdownTimer* gibt an, nach wie vielen Sekunden der Raspberry Pi durch den *ShutdownCmd* heruntergefahren wird. Wird die Stromversorgung durch den Primäreingang wiederhergestellt bevor der *ShutdownTimer* abgelaufen ist, wird er gestoppt. Es sind fünf Sekunden eingestellt, womit eine kurze Unterbrechung der Versorgung ohne Konsequenzen bleibt.
- Der *PowerOffTimer* gibt an, wie lange die PiUSV+ eingeschaltet bleibt, nachdem der *ShutdownCmd* aktiviert wurde. Diese Option wurde auf das Maximum 255 Sekunden gesetzt, was dem Skript2 genug Zeit gibt, die Übertragung durchzuführen und herunterzufahren.
- Mit der Option *ShutdownCmd* kann ein Befehl hinterlegt werden, mit dem die PiUSV+ heruntergefahren werden soll, nachdem die Versorgung auf den Akkumulator gewechselt ist. Es kann hier auch ein auszuführendes Skript hinterlegt werden, welches das Herunterfahren des Raspberry Pi übernehmen soll. Da kein PHP-Skript hinterlegt werden kann, wird ein Bash-Skript gestartet, das das PHP-Skript2 startet.
- Das *LogLevel* gibt an, welche Einträge im Log gespeichert werden sollen.
- Mit *LogStatusDesc* wird eingestellt, ob im Log eine Beschreibung zu den Statusänderungen hinzugefügt wird.
- Die Option *StatusChangedCmd* kann ein weiterer Befehl ausgeführt werden, wenn die Versorgung auf den Akkumulator wechselt.

Der verwendete Akkumulator ist ein Li-Ion Akkumulator von Emmerich und wurde nach den offiziellen Vorgaben der PiUSV+ gewählt. Eine Auswahl der technischen Daten des Akkumulators sind in Tabelle *4* aufgelistet.

Tabelle 4: Technische Daten des Li-Ion-Akkus	
Kapazität	2200 mAh
Max. Ladespannung	4.2 V
Dauerentladestrom	4.3 A
Wiederaufladbar	Ja
Max. Ladestrom	2200 mA
Spannung	3.7 V

(Quelle: vgl. Emmerich: Datasheet ICR-18650NH [50].)

Berechnung der Versorgungszeit

Der Stromverbrauch des Raspberry Pi beträgt ca. 700 mA bei einer Spannung von 5 V. Dazu kommt der Verbrauch des GPS-Empfängers, der ca. 45 mA bei 5 V beträgt. Der Verbrauch des Modems ist nicht konstant und kann bis zu 500 mA bei 5 V betragen. Damit ergibt sich als maximaler Verbrauch eine Leistung von 6,225 W, wie nach Formel 4 berechnet.

$$P_{gesamt} = \left(I_{Raspberry\ Pi} + I_{GPS} + I_{Modem} \right) \cdot U_{5V} = 6{,}225\ W$$

Formel 4: Berechnung der am Raspberry Pi verbrauchten Leistung

Die Kapazität des Akkumulators wird mit Formel 5 in Wh umgerechnet. Sie beträgt 2200 mAh bei einer Spannung von 3,7 V.

$$E_{Akku} = Q_{Akku} \cdot U_{Akku} = 8{,}14\ Wh$$

Formel 5: Umrechnung der Kapazität des Akkumulators in Wh

Damit kann der Akkumulator den Raspberry Pi im vollständig geladenen Zustand etwa 78 Minuten versorgen, siehe Formel 6.

$$t = \frac{E_{Akku}}{P_{gesamt}} = 1{,}31h \sim 78\,minuten$$

Formel 6: Berechnung der Versorgungszeit bei vollständig geladenem Akkumulator

3.2.3 GPS-Modul

Das GPS-Modul ist ein UART GPS NEO-6M der Firma Waveshare, siehe Abbildung 6. Es empfängt Signale der L1-Frequenz, also 1575.42Mhz, und verwendet C/A-Kodierung. Die Positionsdaten werden gemäß dem NMEA-Protokoll ausgegeben. Das GPS-Modul ist über die GPIO-Leiste der PiUSV+ mit der GPIO-Leiste des Raspberry Pi verbunden. Die Kommunikation verläuft über einen UART nach dem RS-232 Standard [51].

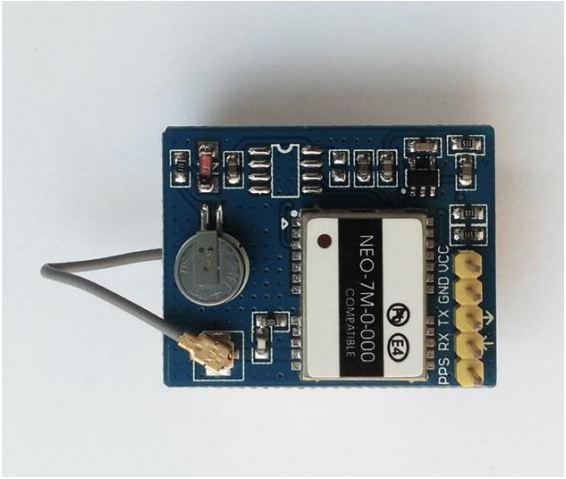

Abbildung 6: Der GPS-Empfänger NEO-6M

3.2.4 LTE-Modem

Das LTE-Modem ist ein K5005 von Huawei. Es ist per USB mit dem Raspberry Pi verbunden und ermöglicht die Einwahl in Mobilfunknetzwerke. Ein Bild des angeschlossenen Modems wird in Abbildung 7 dargestellt.

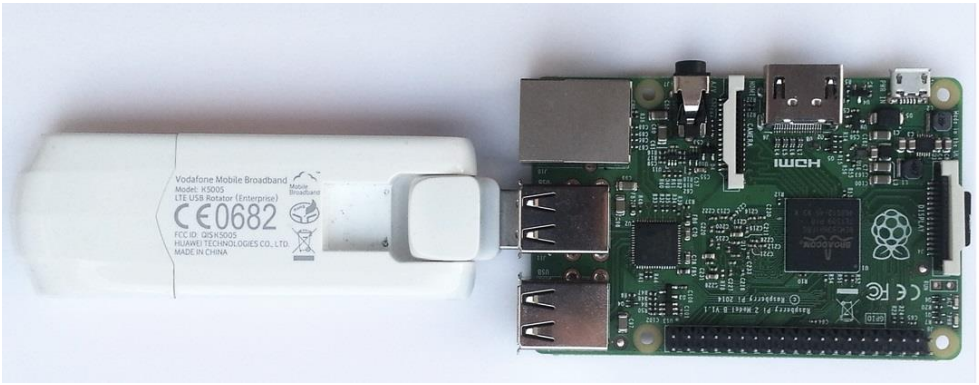

Abbildung 7: LTE-Modem K5005 mit Raspberry Pi

3.2.5 Adapter

Als Adapter wurde ein Modell der Firma CSL-Computer ausgewählt. Er wird in den Zigarettenanzünder gesteckt und liefert an zwei USB-Ausgängen jeweils bis zu 2,4 A. Ein Bild ist in Abbildung 8 eingefügt.

Abbildung 8: USB KFZ Adapter von CSL-Computer
(Quelle: amazon.de: CSL USB KFZ Adapter, https://images-na.ssl-images-amazon.com/images/I/71kth-dosulL._SL1500_.jpg)

3.3 Raspberry Pi und die Peripherie

3.3.1 Verbinden des Raspberry Pi mit dem GPS-Empfänger

Die serielle Schnittstelle des Raspberry Pi ist unter dem Pfad /dev/ttyAMA0 erreichbar. Standardmäßig ist sie als serielle Login-Konsole konfiguriert, die den Bootvorgang protokolliert. Um die Schnittstelle für die Kommunikation mit dem GPS-Empfänger zu verwenden, muss die Login-Konsole deaktiviert werden. Hierfür werden zwei Dateien bearbeitet [52]:

Mit der Datei /boot/cmdline.txt können Argumente an den Linux-Kernel übergeben werden. Hier werden die Referenzen der Login-Konsole mit ttyAMA0 gelöscht, die in den Befehlen „console=ttyAMA0,115200“ und „kgdboc=ttyAMA0,115200“ vorhanden sind.

Die Datei /etc/inittab ist die Konfigurationsdatei von init, das erste Programm das vom Betriebssystem gestartet wird. Da alle weiteren Programme von init gestartet werden, wird es als „Vater aller Prozesse“ bezeichnet [53]. Hier wird eine Login-Aufforderung des seriellen Ports, enthalten im Eintrag „T0:23:respawn:/sbin/getty -L ttyAMA0 115200 vt100“, entfernt oder auskommentiert.

Anschließend wird der GPS-Empfänger mit dem Raspberry Pi gemäß der Nullmodem-Schaltung verbunden, siehe 2.5.3, S.16. Die Pinbelegung des Raspberry Pi B Plus ist in Abbildung 9 erläutert. Sie ist identisch mit der Pinbelegung des Raspberry Pi 2 B.

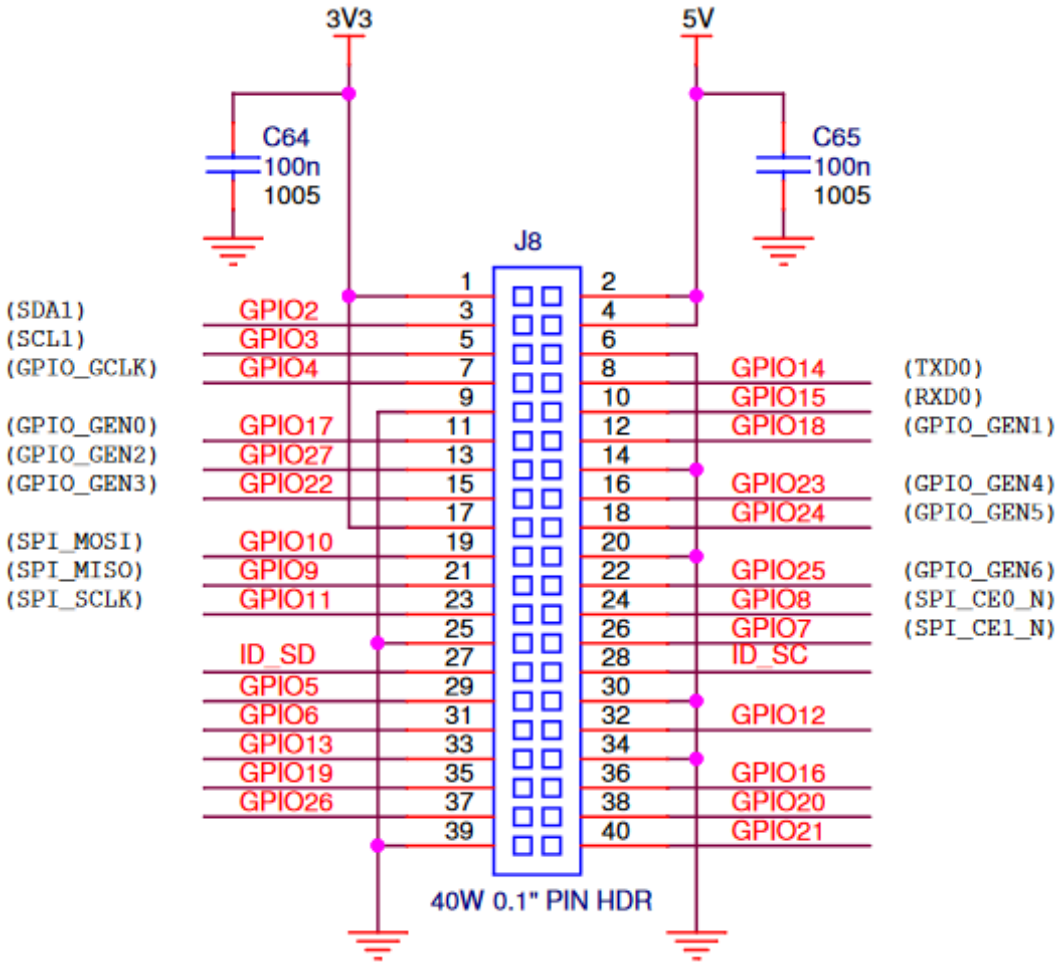

Abbildung 9: Belegung der GPIO-Pins des Raspberry Pi B Plus
(Quelle: vgl. Adams, J.: Raspberry Pi B+ (Reduced Schematics), https://www.raspberrypi.org/documentation/hardware/raspberrypi/schematics/Raspberry-Pi-B-Plus-V1.2-Schematics.pdf [54])

Daraufhin werden die Pins vier, sechs, acht und zehn mit dem GPS-Empfänger verbunden, wie in Abbildung 10 dargestellt. Zum Abrufen der Daten wird das Hintergrund-Programm Global Positioning System Daemon (GPSD) installiert. Es liest die Daten von einem oder mehreren GPS-Empfängern aus, die per serieller Schnittstelle angeschlossen sind. Ein client des GPSD ist das Programm gpxlogger, welches eine automatisierte Speicherung der Positionsdaten in eine vorgegebene Datei mit unterschiedlichen Bedingungen ermöglicht. Das Verhalten des gpxlogger wird durch die Parameter beim Starten bestimmt [55].

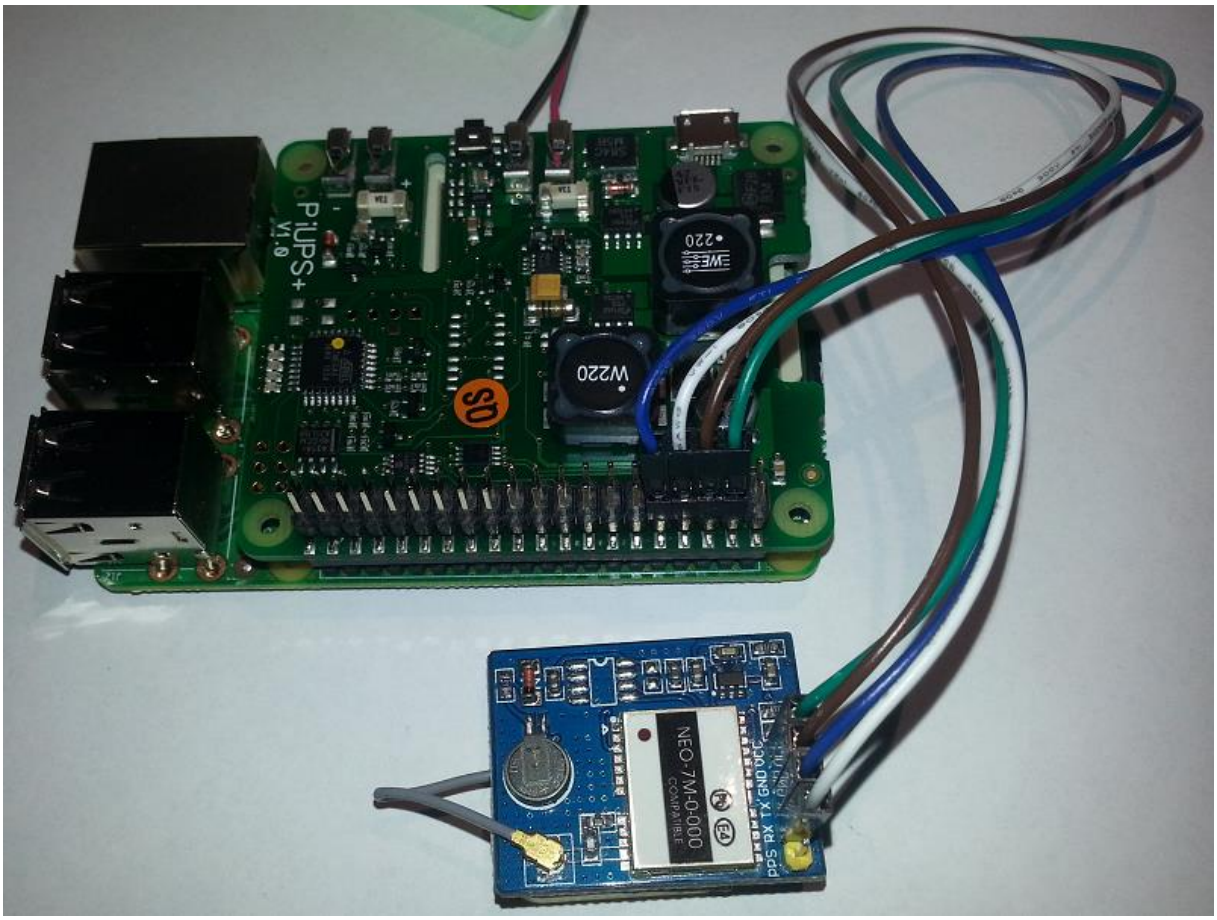

Abbildung 10: Nullmodemschaltung zwischen GPS-Empfänger und Rasberry Pi

3.3.2 Verbinden des Rasberrpy Pi mit der PiUSV+

Die PiUSV+ wird über die GPIO-Pins auf den Raspberry Pi aufgesteckt, siehe Abbildung 11. Die Verbindung zwischen der PiUSV+ und dem Raspberry Pi wird durch den Inter-Integrated Circuit (I2C)-BUS realisiert. Damit die Verbindung funktioniert, muss der I2C-Bus auf dem Raspberry Pi aktiviert und zu den bei Systemstart geladenen Kernel-Modulen hinzugefügt werden. Dies wird durch das Programm raspi-config ausgeführt, siehe Abbildung 12.

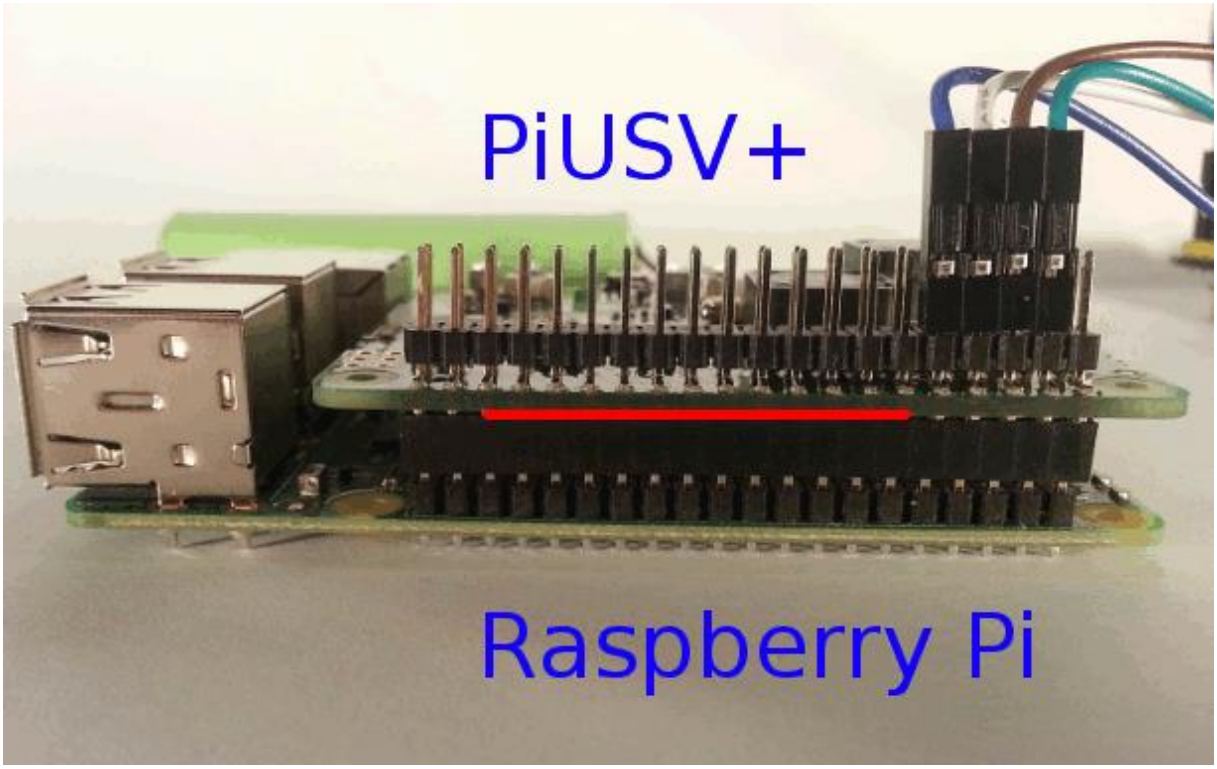

Abbildung 11: Verbindung der PiUSV+ mit dem Raspberry Pi über die GPIO-Leiste

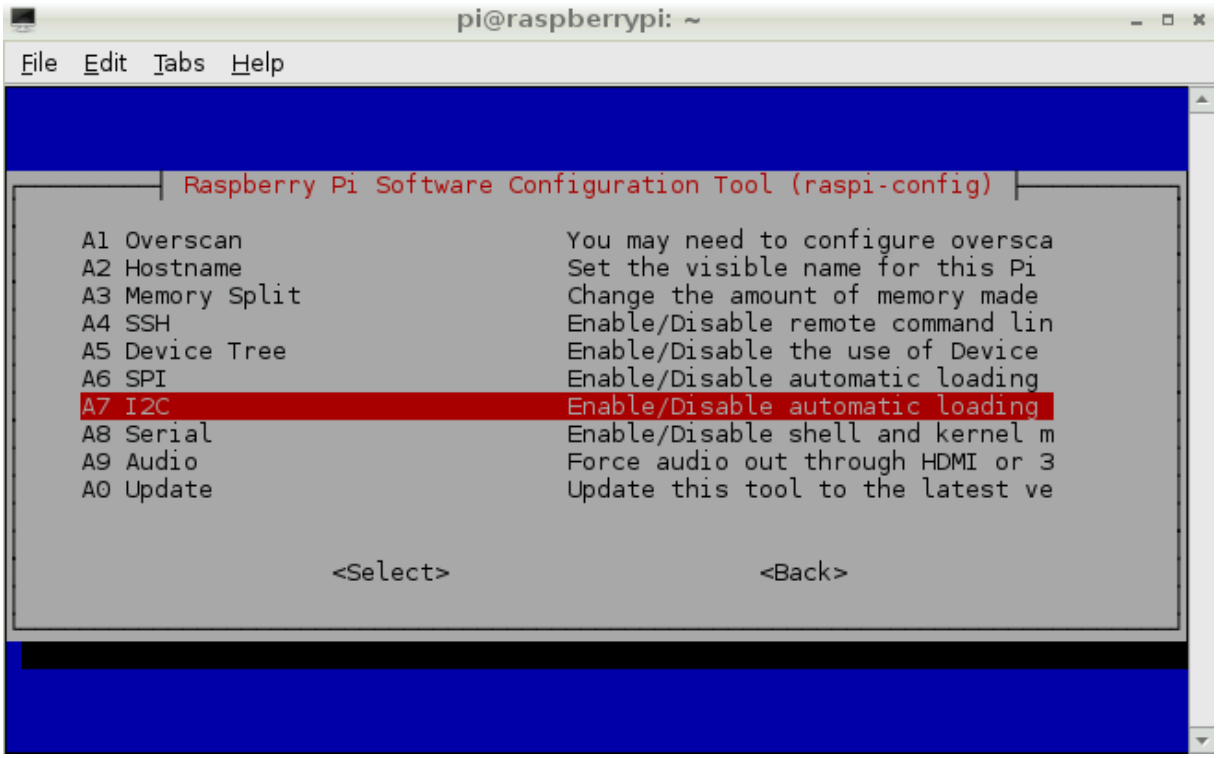

Abbildung 12: Aktivieren des I2C-BUS

3.4 Skripte auf dem Raspberry Pi und dem Server

3.4.1 Verwendete Dateien

Die Skripte auf dem Raspberry Pi arbeiten mit verschiedenen Text-Dateien. Die Dateien liegen im Ordner */home/pi/Desktop/Sender/*. Um den vorgestellten Code übersichtlicher zu gestalten, wurde die Pfadangabe auf den Dateinamen verkürzt. Die Skripte sind im Anhang ungekürzt einsehbar. Um das Verstehen der kommenden Kapitel zum Ablauf der Skripte zu erleichtern, soll hier ein kurzer Überblick über die verwendeten Dateien gegeben werden.

Die Datei rohdaten.gpx enthält die unbearbeiteten Einträge des *gpxloggers*. Sie folgen dem GPX-Format, siehe Kapitel 2.4.6, S.14.

Die Datei routen.txt enthält in jeder Zeile einen noch nicht übertragenen Routeneintrag. Ein Routeneintrag enthält nur noch die Informationen, die an den Server übertragen werden sollen. Wird eine Route erfolgreich übertragen, wird sie nach archiv.txt verschoben. Die enthaltenen Informationen sind:

- Startzeitpunkt der Route
- Geographische Länge des Startorts
- Geographische Breite des Startorts
- Endzeitpunkt der Route
- Geographische Länge des Zielorts
- Geographische Breite des Zielorts
- Gefahrene Distanz

Die Datei archiv.txt enthält die erfolgreich übertragenen Routen.

Die Dateien latalt.txt und lonalt.txt enthalten den Endpunkt der letzten Route. Sie werden als Anfangspunkt der nächsten Route verwendet. Das sorgt für einen lückenlosen Übergang der Routen und überbrückt die Zeit, die das GPS-Modul beim Hochfahren benötigt, um einen Fix herzustellen.

Die Datei skript1.error.log und skript2.error.log enthalten die Fehlermeldungen der PHP-Skripte.

3.4.2 Skript 1 auf dem Raspberry Pi

Beim Hochfahren des Raspberry Pi wird per Autostart das PHP-Skript1 ausgeführt. Es dient zum starten des Programms gpxlogger.

```
shell_exec('gpxlogger -d -i 3600 -f rohdaten.gpx');
```

`shell_exec` Die Funktion *shell_exec* führt das in Klammern übergebene Kommando auf der Shell aus und gibt den Output als String zurück [56].

Das Programm *gpxlogger* dient, wie in 3.3.1, S.27 erläutert, dem Auslesen und Speichern der Positionsdaten. Das Verhalten des *gpxlogger* wird durch die Parameter beim Starten bestimmt [55].

`-d` Der Client *gpxlogger* wird als Daemon, also als Hintergrundprogramm, gestartet. Damit wird er eine eigene Instanz im Betriebssystem und das Skript kann weiterlaufen.

`-f` Dieser Parameter lässt alle entgegengenommenen Werte in ein File schreiben. In diesem Fall das Textfile rohdaten.gpx.

`-i` 3600 Das -i gibt an, nach wie vielen Sekunden ohne Verbindung zu mindestens vier Satelliten ein neues Tracksegment angelegt wird. Der Wert 3600 wurde gewählt um sicherzustellen, dass alle Werte in das gleiche Tracksegment gespeichert werden. Der Einlesevorgang in Skript2 benötigt alle Werte in einem Tracksegment.

3.4.3 Shell-Skript zum Starten des PHP-Skript2

Das Shell-Skript wird von der PiUSV+ gestartet, nachdem die Zündung ausgeschaltet wurde. Es startet das PHP-Skript2 und beendet sich.

```
#!/bin/bash

php skript2.php

exit 0
```

3.4.4 Skript 2 auf dem Raspberry Pi

3.4.4.1 Beenden des gpxloggers

Nach dem Ende der Route wird der *gpxlogger* beendet. Hierbei wird sich erneut der Shell bedient, um erst die Prozess-ID des *gpxloggers* herauszufinden und diesen dann zu beenden.

`pgrep` Die Funktion *pgrep* durchsucht die laufenden Prozesse des Systems nach den übergebenen Parametern und übergibt die Prozess-ID.

`kill` Die Funktion *kill* beendet den Prozess mit der angegebenen Prozess-ID.

Pgrep und *kill* sind Teil des Software-Pakets *procps* [57].

```
$gpsid = shell_exec('pgrep gpxlogger');

shell_exec('kill $gpsid');
```

3.4.4.2 Berechnen der Distanz

Vor der Berechnung werden die Endkoordinaten der letzten Route als Startkoordinaten verwendet. Die Koordinaten werden aus den Textfiles „latalt.txt“ und „lonalt.txt“ ausgelesen, in den Variablentyp double umgewandelt und den Variablen $latalt und $lonalt als Wert gegeben.

`fopen` Mit der Funktion *fopen* wird eine Datei zum Bearbeiten geöffnet. Der hier verwendete Parameter "r" steht für read und öffnet eine Datei zum Lesen [58].

`fgets` Die Funktion *fgets* liest eine Zeile ab der Position des Dateizeigers ein [59].

`fclose` Über die Funktion *fclose* wird eine zum Bearbeiten geöffnete Datei geschlossen [60].

`floatval` Die Funktion *floatval* konvertiert einen Wert nach float [61].

```
$handle = fopen("latalt.txt", "r");
$latalt = floatval(fgets ($handle));
fclose ($handle);
$handle = fopen("lonalt.txt", "r");
$lonalt = floatval(fgets ($handle));
fclose ($handle);
```

Nun wird die Datei rohdaten.gpx zum Lesen geöffnet, in der die aufgenommenen Koordinaten des GPS-Empfängers vom Logger hinterlegt worden sind.

`simplexml_load_file` Mit dieser Funktion wird ein XML-File in ein Objekt umgewandelt [62]. Durch diese Umwandlung ist es möglich, gezielt Daten auszulesen.

```
$xmlobjekt = simplexml_load_file("rohdaten.gpx");
```

Zum Berechnen der gefahrenen Route werden die geographische Breite und Länge der Einträge benötigt. Um diese Informationen aus den Einträgen auszulesen, werden durch eine foreach-Schleife die Trackpoints nacheinander eingelesen. Dabei wird der momentane Trackpoint jeweils in der Variable $trkpt gespeichert. Die Attribute lat und lon des Trackpoints werden ausgelesen und in den Variablen $latneu und $lonneu gespeichert. Für den Aufbau der GPX-Datei siehe Kapitel 2.4.6, S. 14.

```
foreach( $xmlobjekt->trk->trkseg->{'trkpt'} as $trkpt ) {
            $latneu = floatval($trkpt->attributes()->lat);
            $lonneu = floatval($trkpt->attributes()->lon);
            [...Distanzberechnung...]
            $lonalt = $lonneu;
            $latalt = $latneu;
            $endzeit = $trkpt->time;
            }
```

Aus diesen Werten wird daraufhin mit der in Kapitel 2.4.5 hergeleiteten Formel die Distanz berechnet. Da PHP zur Berechnung des Cosinus mit dem Bogenmaß arbeitet, wird der Wert der mittleren geographischen Breite durch Formel 7 umgerechnet.

$$\frac{2 \cdot \pi}{360°} \cdot Winkel\ in\ Grad\ [°] = Winkel\ im\ Bogenmaß\ [rad]$$

Formel 7: Umrechnung von Grad in das Bogenmaß

```
$latbm = ($latneu + $latalt) / 2 * 0.01745;

$dlon = 111.3 * cos($latbm) * ($lonalt - $lonneu);

$dlat = 111.3 * ($latalt - $latneu);

$distance = $distance + round(sqrt(pow($dx,2) + pow($dy,2)),5);
```

3.4.4.3 Auslesen der Start- und Endwerte

Um die Startzeit zu bestimmen, wird der time-Wert des ersten Trackpoints ausgelesen und in der Variable $startzeit gespeichert.

Die Endzeit wird bereits in der foreach-Schleife gespeichert und ist in der Variable $endzeit verfügbar.

Für die Startkoordinaten werden die lat- und lon-Werte des ersten Trackpoints erneut ausgelesen und in den Variablen $startlat und $startlon gespeichert.

Die Endkoordinaten können ebenfalls der foreach-Schleife entnommen werden und werden zur Übersicht den Variablen $ziellat und $ziellon übergeben.

Anschließend werden die Endkoordinaten in den Dateien latalt.txt und lonalt.txt gespeichert, um für die nächste Route zur Verfügung zu stehen. Dabei werden die alten Werte überschrieben.

```
$startzeit = $xmlobjekt->trk->trkseg->trkpt->time;

$startlat = $xmlobjekt->trk->trkseg->trkpt->attributes()->lat;

$startlon = $xmlobjekt->trk->trkseg->trkpt->attributes()->lon;

$ziellat = $latneu;

$ziellon = $lonneu;

file_put_contents("latalt.txt",$latalt);

file_put_contents("lonalt.txt",$lonalt);
```

3.4.4.4 Eintragen des Routeneintrags in routen.txt

Es sind nun alle Daten verfügbar, die notwendig sind, um einen Routeneintrag zu verfassen.

Die Datei routen.txt wird mit fopen und dem Parameter „a“ geöffnet, was bedeutet, dass sie zum Anfügen geöffnet wird (append = anhängen). In der Variable $route werden daraufhin alle benötigten Werte für eine Route zwischengespeichert und als neuer Eintrag der Datei routen.txt angefügt. Nach jedem Eintrag wird mit dem Befehl *\n* in die nächste Zeile gesprungen.

```
$handle = fopen("routen.txt", "a");

$route =
"$startzeit,$endzeit,$startlat,$startlon,$ziellat,$ziellon,$distance
";

fwrite ($handle, "$route\n");

fclose ($handle);
```

3.4.4.5 Aufbau der Mobilfunkverbindung

Das per USB mit dem Raspberry Pi verbundene LTE-Modem ist unter der Adresse */dev/ttyUSB0* erreichbar. Um die Verbindung zu ermöglichen, werden die Pakete *wvdial* und *ppp* installiert.

Das Paket *ppp* enthält den Punkt-zu-Punkt Protokoll daemon (pppd). Dieser wird verwendet um eine Point-to-Point Verbindung zu managen. Das Point-to-Point Protokoll wird in der Regel verwendet, um eine Verbindung zwischen einem Modem und dem Internetanbieter herzustellen. Über diese Verbindung können daraufhin andere Protokolle übertragen werden. Es

wurde geschaffen, um das Einwählen in das Internet zu ermöglichen. Nähere Informationen sind im offiziellen Standard verfügbar [63].

Das Programm *wvdial* erkennt angeschlossene Modems und ermöglicht eine Internetverbindung durch den *pppd*. Da eine SIM-Karte von blau.de verwendet wird, wurde die Konfigurationsdatei, die unter */etc/wvdial/wvdial.conf* verfügbar ist, mit folgendem Inhalt erstellt:

```
[Dialer blau]
Modem Type = Analog Modem
Modem = /dev/ttyUSB0
Baud = 9600
Init1 = ATZ
Init2 = ATQ0 V1 E1 S0=0 &C1 &D2 +FCLASS=0
init3 = AT+CGDCONT=1,"IP","internet.eplus.de"
Phone = *99#
Username = blau
Password = blau
```

Das Herstellen einer Verbindung dauert ca. 35 Sekunden.

Anschließend wird die Verbindung mit folgendem Befehl aufgebaut:

```
shell_exec('sudo wvdial blau');
sleep (60);
```

sleep — Die Funktion sleep verzögert den weiteren Programmablauf um den angegebenen Wert in Sekunden. Der Wert 60 wurde gewählt um sicherzustellen, dass eine Verbindung hergestellt wurde, bevor eine Übertragung versucht wird [64].

3.4.4.6 Senden der Routen

Nach dem Aufbau der Mobilfunkverbindung wird versucht, alle bisher noch nicht übertragenen Routen zu senden. Eine Übersicht über den Ablauf der Übertragung ist in Abbildung 13 in einem Programmablaufplan vorhanden.

Da das Firmenfahrzeug an unterschiedlichen Orten unterwegs ist, ist nicht sichergestellt, dass nach jeder Route eine Verbindung hergestellt und die Route übertragen werden kann. Daher werden alle noch nicht übertragenen Routen in der Datei routen.txt gespeichert. Diese ist so aufgebaut, dass der älteste Eintrag in der ersten Zeile steht und neuere jeweils in einer neuen Zeile angefügt werden. Das Skript liest die Datei routen.txt Zeile für Zeile ein und versucht den jeweiligen Routeneintrag zu senden. Dadurch bleibt die ursprüngliche Reihenfolge der Routen erhalten.

Realisiert wird dies mit einer Do-While-Schleife. Sie sorgt dafür, dass mindestens ein Übertragungsversuch unternommen wird. In der Variable $result wird die Antwort des Skripts auf dem Server gespeichert. Diese ist bei erfolgreicher Übertragung „OK“.

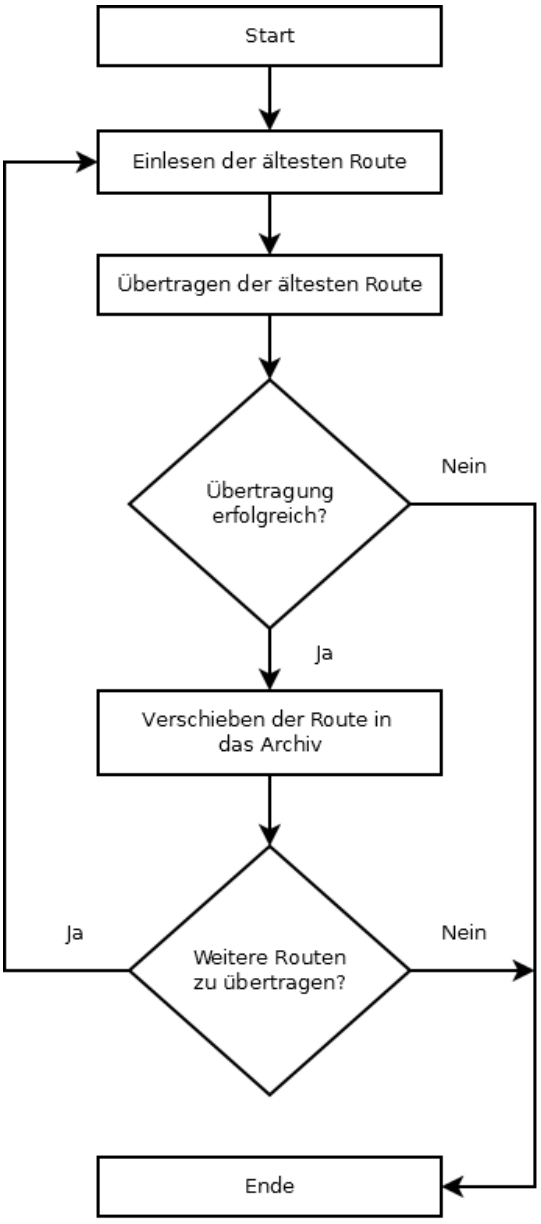

Abbildung 13: Programmablaufplan der Übertragung der Routen

```
do {

[Übertragung]

} while ($result == "OK");
```

Der Ablauf der Übertragung ist in Kapitel 2.3.4, S.9 erläutert.

Dabei werden bei jedem Durchgang die Daten der zu übertragenden Route in der Variablen $ubergabe als Array zusammengestellt. Die erste Zeile der Datei routen.txt wird eingelesen

und in der Variable $zeile gespeichert. Anschließend werden die Informationen durch explode aufgeteilt und in das Array $stringdaten gespeichert. Daraufhin wird das Array $ubergabe erstellt, wobei die Schlüssel benannt und mit den jeweiligen Werten ergänzt werden.

`explode` Die Funktion *explode* teilt einen String anhand eines Trennungszeichens auf. In diesem Fall wird ein Eintrag der Datei routen.txt nach jedem Komma in Teilstrings aufgetrennt. Explode gibt ein Array zurück, in dem die Teilstrings gespeichert sind [65].

```
$handle = fopen("routen.txt", "r");
$zeile = fgets ($handle);
$stringdaten = explode(",",$zeile);
fclose($handle);
$ubergabe = array(
     'startzeit'          =>   $stringdaten[0],
     'endzeit'            =>   $stringdaten[1],
     'startlat'           =>   $stringdaten[2],
     'startlon'           =>   $stringdaten[3],
     'ziellat'            =>   $stringdaten[4],
     'ziellon'            =>   $stringdaten[5],
     'distance'           =>   $stringdaten[6]
     );
```

3.4.4.7 Verschieben der übertragenen Route in das Archiv

Nach erfolgreicher Übertragung ist die jeweilige Route noch in der Variable $zeile gespeichert. Damit kann der Eintrag direkt der Archivdatei angefügt werden. Um den Eintrag aus routen.txt zu entfernen, wird die komplette Datei in das Array $routenarray eingelesen, wobei jede Zeile einem Element entspricht. Daraufhin wird das nullte Element des Arrays entfernt und das Array wieder in routen.txt geschrieben.

`file` Die Funktion *file* liest eine Datei in ein Array. Dabei entspricht jede Zeile der Datei einem Element des Arrays [66].

`unset` Mit *unset* wird eine Variable oder gezielt ein Element eines Arrays gelöscht [67].

`file_put_contents` Diese Funktion schreibt Daten in eine Datei. Sie kann, im Gegensatz zu *fwrite*, auch Arrays in eine Datei schreiben [68].

```
$handle = fopen("archiv.txt", "a");
          fwrite ($handle, "$zeile");
          fclose ($handle);
          $routenarray = file('routen.txt');
          unset($routenarray[0]);
          file_put_contents('routen.txt',$routenarray);
```

3.4.4.8 Überprüfung der Anzahl noch zu übertragender Routen

Ob noch zu übertragende Routen existieren, wird überprüft, indem die Zeilenzahl der Datei routen.txt gezählt wird. Daraus ergibt sich die Anzahl der noch zu übertragenden Routen, da jede Zeile eine Route enthält. Sind keine zu übertragenden Routen mehr vorhanden, wird die do-while-Schleife mit dem Befehl break verlassen.

`count` Die Funktion *count* zählt die Elemente eines Arrays [69].

`break` Mit *break* wird die Ausführung der aktuellen for-, foreach-, while-, do-while- oder switch-Struktur beendet [70].

```
$zeilenzahl = count(file('routen.txt'));
if ($zeilenzahl == '0'){
     break;
}
```

3.4.4.9 Beenden der Internetverbindung

Zum Beenden der Mobilfunkverbindung genügt es, den Prozess, der die Verbindung aufgebaut hat, zu beenden. Dies wird erneut realisiert, indem die Prozess-ID im Betriebssystem herausgefunden und der Prozess daraufhin beendet wird.

```
$dialid = shell_exec('pgrep sudo wvdial dialer');

shell_exec('kill $dialid');
```

3.4.4.10 Herunterfahren des Raspberry Pi

Der Befehl zum Herunterfahren wird durch die Shell ausgeführt

`shutdown` Mit dem Befehl shutdown wird das System heruntergefahren.

`-p` Der Parameter -p (powerdown) fährt das System herunter schaltet den Rechner aus.

`now` Mit dem Parameter now fährt das System sofort herunter.

```
shell_exec('sudo shutdown -p now');
```

3.4.5 Skript auf dem Server

Auf dem Server von NC3 wartet ein drittes Skript auf einen Aufruf durch das zweite Skript. Damit es aufgerufen werden kann, liegt es auf einem Webserver des Firmenservers.

3.4.5.1 Annehmen der Übertragung

Das per POST übertragene Array wird in die Variable $ubergabe gespeichert. Die Variable $_POST enthält die dem aktuellen Skript per POST übergebenen Daten, wenn application/x-www-form-urlencoded oder multipart/form-data als HTTP Content-Type verwendet worden ist [71]. Da der verwendete HTTP Content-Type ersterem entspricht, ist das Entgegennehmen der Route mit diesem Befehl möglich.

```
$ubergabe = $_POST;
```

Der entgegengenommene Inhalt entspricht folgendem Array:

```
array(
          'startzeit'                =>    Wert0,
          'endzeit'                  =>    Wert1,
          'startlat'                 =>    Wert2,
          'startlon'                 =>    Wert3,
          'ziellat'                  =>    Wert4,
          'ziellon'                  =>    Wert5,
          'distance'                 =>    Wert6
     );
```

3.4.5.2 Umgekehrtes Geocoding der Start- und Zieladresse

Umgekehrtes Geocoding bezeichnet die Umwandlung geographischer Koordinaten in menschenlesbare Adressen. Hierfür kann kostenlos die Google Maps Geocoding Application Programming Interface (API) verwendet werden, indem eine HTTP-Anfrage an den Dienst gesendet wird [72]. Der Aufruf der API wird mit den Werten der geographischen Breite und Länge aus $ubergabe angepasst. Anschließend wird die Anfrage durchgeführt und die Antwort in der Variable $google_start gespeichert. Da die Antwort im Datenformat JavaScript Object Notation (JSON) Format vorliegt, wird sie vor der Weiterverwendung in eine PHP-Variable umgewandelt.

`json_decode` Mit der Funktion *json_decode* wird eine JSON-kodierte Zeichenkette in eine PHP-Variable konvertiert [73].

Für den Zweck der Arbeit ist lediglich die den Koordinaten entsprechende Adresse interessant. Die Antwort der Google-API enthält jedoch eine Vielzahl weiterer Informationen. Daher wird die Adresse gezielt aus $google_start ausgelesen und als neuer Schlüssel „startadresse“ der Variable $ubergabe hinzugefügt. Mit dem gleichen Verfahren wird auch die Zieladresse ermittelt und als Schlüssel „zieladresse“ in $ubergabe gespeichert.

```
$url =
"https://maps.googleapis.com/maps/api/geocode/json?latlng=$ubergabe[
startlat],$ubergabe[startlon]";
$google_start = json_decode(file_get_contents($url));
$ubergabe['startadresse'] = $google_start->results[0]-
>formatted_address;
```

3.4.5.3 Eintrag ins Empfängerarchiv

Auf dem Server wird ein zweites Archiv geführt. Der Eintrag wird per fwrite vorgenommen, wobei das Array $ubergabe vorher in einen String umgewandelt wird. Damit sind die Archive einheitlich aufgebaut.

`implode` Die Funktion *implode* ist das Gegenstück zu *explode* und verbindet die Elemente eines Arrays zu einem String mit einstellbarem Trennungszeichen [74].

```
$handle = fopen("archiv_empf.txt", "a");
$archiv = implode(",", $ubergabe);
fwrite ($handle, "$archiv\n");
fclose ($handle);
```

3.4.5.4 Eintrag in die Datenbank

Bevor Daten in die Datenbank eingetragen werden können, muss diese noch erstellt werden. Dafür wurde das Programm „DB Browser for SQLite“ verwendet, das das Erstellen, Designen und Editieren von SQL-Datenbankfiles ermöglicht.

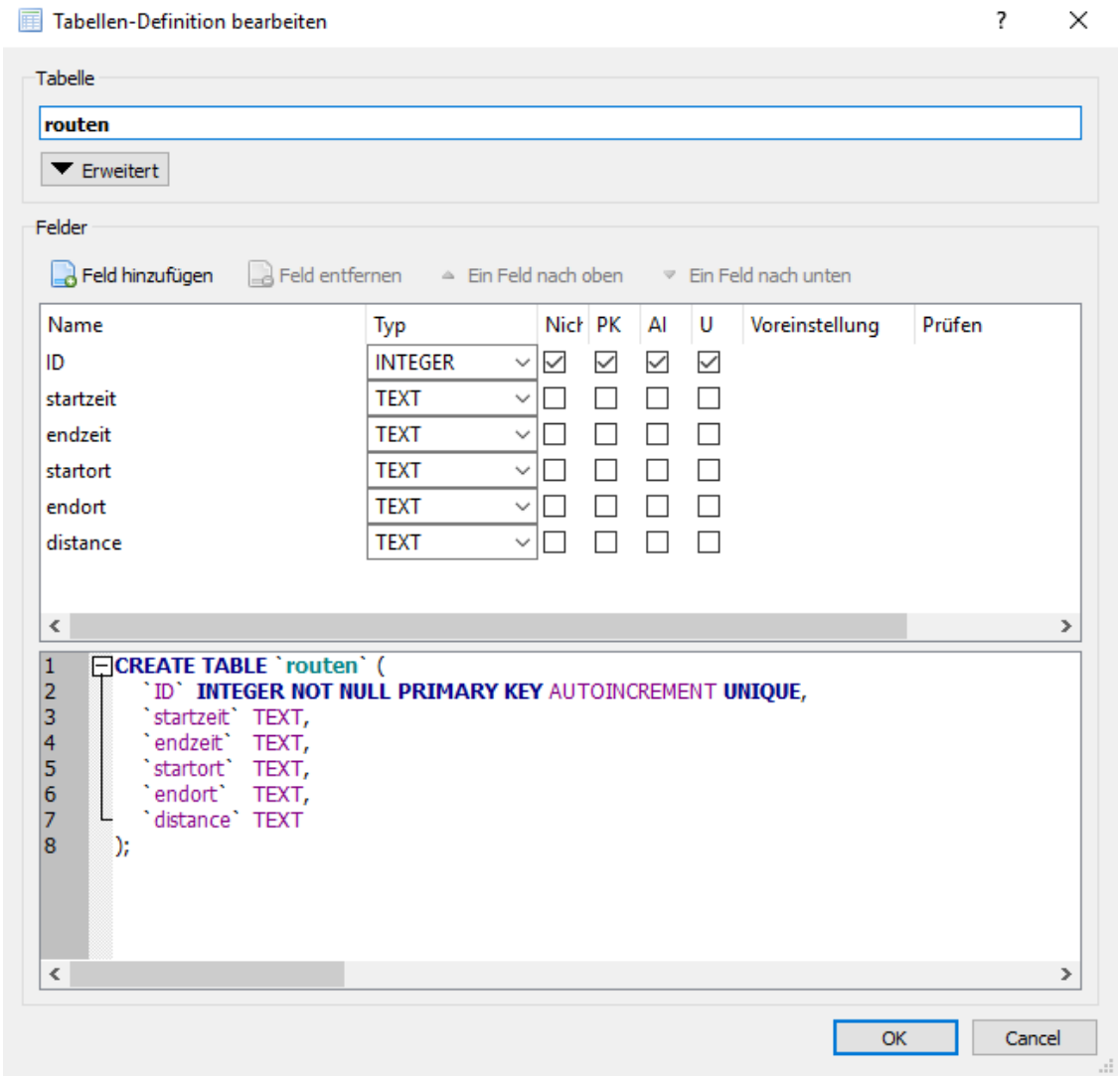

Abbildung 14: Erstellen der Datenbank mit „DB Browser for SQLite"

Die erstellte Datenbank enthält eine Tabelle mit den Werten der Route sowie eine ID-Spalte. Siehe dazu Abbildung 14.

Vor dem Eintragen muss eine Verbindung zwischen dem PHP-Skript und der Datenbank aufgebaut werden. Die Verbindung wird folgendermaßen durch eine PHP Data Objects (PDO)-Klasse definiert [75]:

```
$pdo = new PDO('sqlite:'.dirname(__FILE__).'/db.db');
```

`new PDO`	Der Befehl *new PDO* erstellt eine neue Instanz der PDO-Klasse.
`'sqlite:'`	Mit dem Argument *'sqlite:'* wird der verwendete PDO-Treiber spezifiziert [76].

`.dirname(__FILE__).'/db.db'`	Diese Angabe stellt den Pfad der Datenbankdatei dar. Es ist im gleichen Ordner wie das Skript und hat den Namen „db.db“.

Das Eintragen der Route in die Datenbank wird daraufhin durch folgenden Code vorgenommen:

```
$query = 'INSERT INTO routen (startzeit, endzeit, startort, endort,
distance)
                VALUES (   "'.$ubergabe['startzeit'].'",
                           "'.$ubergabe['endzeit'].'",
                           "'.$ubergabe['startadresse'].'",
                           "'.$ubergabe['zieladresse'].'",
                           "'.$ubergabe['distance'].'")';
     $stobj = $pdo->prepare($query);
     $stobj->execute();
     $pdo = null;
```

Zunächst wird in der Variablen $query die Anweisung zum Eintragen in die Datenbank als String erstellt. Dieser folgt folgendem Schema [77]:

```
INSERT INTO Tabellenname (Spaltenname1, Spaltenname2, …)  Values
(Wert1, Wert2, …);
```

Anschließend wird die Anweisung für die Ausführung vorbereitet und daraufhin durchgeführt. Danach wird die Verbindung zur Datenbank geschlossen, indem die Variable $pdo gleich null gesetzt wird.

$pdo->prepare($query)	Mit diesem Befehl wird die Anweisung $query auf die Durchführung auf der in $pdo definierten Datenbank vorbereitet. Dadurch wird ein SQLite3Stmt-Objekt erstellt. Objekte dieser Klasse sind vorbereitete Anweisungen für die SQLite-3-Erweiterung [78].
$stobj->execute()	Mit diesem Befehl wird die vorbereitete Anweisung durchgeführt [79].

3.4.6 Sonderfälle

3.4.6.1 Zündung wird ausgeschaltet bevor Koordinaten hinterlegt werden

In diesem Fall kommt es in Skript2 beim Auslesen des Objekts

```
$xmlobjekt->trk->trkseg->trkpt->time
```

zu einem Fatal Error, da die gpx-Datei in diesem Fall keine Objekte ab *trk* enthält. Damit bricht das Skript ab ohne den Raspberry Pi herunterzufahren. Um den Raspberry Pi in diesem Fall herunterzufahren wurde vor der Abfrage folgende if-Schleife hinzugefügt, die überprüft, ob `$xmlobjekt->trk->trkseg` ein Objekt ist:

```
$isobj = is_object($xmlobjekt->trk->trkseg);

if ($isobj = FALSE) {

shell_exec('sudo shutdown -h now');

}
```

`is_obj` Die Funktion *is_obj* überprüft ob eine Variable vom Typ object ist. Die Rückgabewerte sind TRUE oder FALSE [80].

3.4.6.2 Schwierigkeiten beim GPS-Empfang

Der GPS-Empfänger versucht kontinuierlich eine Verbindung herzustellen. Solange eine Verbindung besteht werden Werte aufgeschrieben. Besteht keine Verbindung werden auch keine Werte aufgeschrieben. Geht die Verbindung während einer Fahrt verloren, stoppt das Schreiben der Positionswerte für die Dauer des Verbindungsproblems. Sobald die Verbindung wiederhergestellt wird, beginnt das Schreiben der Positionswerte im direkten Anschluss.

3.4.6.3 Stromunterbrechung an der Versorgung der PiUSV+

Unterbrechungen unter fünf Sekunden werden ignoriert, siehe Kapitel 3.2.2, S. 22. Eine Unterbrechung, die länger als fünf Sekunden dauert, wird als Ende einer Route interpretiert.

Ein Wiederherstellen der Stromversorgung am Primäreingang nach 6 bis 255 Sekunden ist problematisch, da der Raspberry Pi bereits heruntergefahren ist, die PiUSV+ allerdings noch läuft. Dadurch wird dem Raspberry Pi keine steigende Flanke an der Versorgung gegeben und er fährt nicht hoch.

4 Zusammenfassung und Ausblick

Im Theorieteil der Arbeit wurden die theoretischen Grundlagen erläutert, die zum Nachvollziehen der Arbeit notwendig sind.

Das HTTP gilt als „core protocol of the World Wide Web". Es ermöglicht die Kommunikation zwischen Servern und Clients und wird täglich von vielen Menschen verwendet. Die häufigsten Anfrage-Methoden sind GET und POST. HTTP wird in der Arbeit verwendet, um Routen vom Raspberry Pi auf den Firmen-Server zu übertragen.

PHP ist eine weit verbreitete Open Source-Skriptsprache, die meistens für serverseitige Programmierung, Kommandozeilenprogrammierung und das Schreiben von Desktop-Applikationen verwendet wird. Die in der Arbeit verwendeten Skripte sind in PHP geschrieben.

Als Navigationssystem wird das amerikanische GPS verwendet. Mit einem Empfänger kann weltweit durch Empfangen der Satellitensignale die eigene Position bestimmt werden. Als Datenformat wird GPX verwendet, ein XML-basiertes Datenformat zum Austausch von GPS-Daten.

Serielle Schnittstellen übertragen Informationen nacheinander Bit für Bit. Die Kommunikation zwischen Raspberry Pi und dem GPS-Empfänger wird durch den seriellen Übertragungsstandard RS-232 definiert und durch einen UART realisiert.

Anschließend werden im Praxisteil der Aufbau und die Durchführung erläutert.

Der Raspberry Pi ist über eine USV im Auto an das Zündungsplus angeschlossen. Er wird parallel zur Zündung hoch- und runtergefahren. Durch einen angeschlossenen GPS-Empfänger schreibt er die momentane Position mit. Anschließend überträgt er die gefahrene Route durch ein verbundenes USB-LTE-Modem an den Firmenserver. Der Ablauf wird durch insgesamt drei PHP-Skripte durchgeführt.

Fazit

Vom Erfassen der Koordinaten bis zum fertigen Eintrag in der Datenbank konnten alle Arbeitsschritte erfolgreich umgesetzt werden. Es wurden mehrere erfolgreiche Testfahrten durchgeführt. Es ist zudem gelungen, das System so zu gestalten, dass der Fahrer selbst nicht tätig werden muss. Ein Problem stellen Pausen zwischen 5 und 255 Sekunden dar, da der Raspberry Pi in diesen Fällen herunterfährt, aber nicht wieder gestartet wird, da die Pi-USV+ noch läuft. Eine Lösung für dieses Problem zu finden kann Teil einer weiterführenden

Arbeit darstellen. Werden diese Fäll vermieden, wird es in Zukunft deutlich einfacher sein, vergessene Fahrtenbucheinträge nachzutragen.

Eine weiterführende Arbeit kann das Entwickeln einer Smartphone-App darstellen, auf der die Datenbankeinträge verfügbar sind. Des Weiteren kann ein Abgleich mit dem firmeninternen Google-Kalender vorgenommen und darauf aufbauend Vorschläge für einen Fahrtenbucheintrag erstellt werden.

5 Literaturverzeichnis

[1] Einkommensteuergesetz (EStG): § 6 Bewertung; https://www.gesetze-im-internet.de/estg/__6.html (10.10.2016, 11:24).

[2] Einkommensteuergesetz (EStG): § 8 Einnahmen; https://www.gesetze-im-internet.de/estg/__8.html (10.10.2016, 12:17).

[3] Bundesfinanzhof: Urteil vom 16.03.2006 - VI R 87/04; http://www.meyer-gwinner.de/downloads/fahrtenbuch-urteile/BFH_Urteil_16-3-2006.pdf (10.10.2016, 12:39), gespeichert als BFH_Urteil.pdf.

[4] Bundesfinanzhof: Urteil vom 6.08.2013 Az. VIII R33/11; https://openjur.de/u/669164.html (10.10.2016, 13:00), gespeichert als BFH_Urteil2.pdf.

[5] raspberrypi.org:Pi Foundation Review 2015, https://www.raspberrypi.org/files/about/RaspberryPiFoundationReview2015.pdf, gespeichert als: Raspi.pdf.

[6] in banana-pi.org: BPI-M3, http://www.banana-pi.org/m3.html (24.10.2016, 14:42).

[7] in Merz, A.: Odroid C2 und Lemaker Guitar im Test, http://www.golem.de/print.php?a=120001 (24.10.2016, 14:31).

[8] Solid-run.com: HummingBoard Edge, http://wiki.solid-run.com/lib/exe/fetch.php?media=imx6:hummingboard:docs:sr-imx6-hummingboard-edge-datasheet.pdf (29.10.2016, 11:35), gespeichert als: HB_Edge.pdf.

[9] in beagleboard.org: BeagleBone 101, https://beagleboard.org/Support/bone101/ (24.10.2016, 13:58).

[10] beagleboard.com: BeagleBone Black, https://beagleboard.org/black (29.10.2016, 11:51).

[11] arduino.cc: ArduinoMEGA 2560, https://www.arduino.cc/en/Main/ArduinoBoardMega2560 (24.10.2016, 14:11).

[12] httpwg.org: IETF HTTP Working Group, http://httpwg.org (07.10.2016, 13:06).

[13] Fielding, T.; Reschke, J.F.: Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing; https://tools.ietf.org/pdf/rfc7230.pdf (07.10.2016, 11:46), S. 7, gespeichert als: rfc7230.pdf.

[14] Belshe, M. et al.: Hypertext Transfer Protocol Version 2; https://tools.ietf.org/pdf/rfc7540.pdf (07.10.2016, 12:29).

[15] in w3techs.com: Usage of HTTP/2 for websites; https://w3techs.com/technologies/details/ce-http2/all/all (07.10.2016, 12:25), gespeichert als http_usage.mht.

[16] Fielding, T.; Reschke, J.F.: Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content; https://tools.ietf.org/html/rfc7231 (07.10.2016, 17:14).

[17] IANA.org: Message Headers; http://www.iana.org/assignments/message-headers/message-headers.xhtml (09.10.2016, 17:58).

[18] Achour, M. et al.; Cowburn, P. [Hrsg.]: Offizielle Sprach- und Befehlsreferenz; http://php.net/manual/de/history.php.php (18.10.2016, 18:37).

[19] Achour, M. et al.; Cowburn, P. [Hrsg.]: Offizielle Sprach- und Befehlsreferenz; http://php.net/manual/de/intro-whatis.php (18.10.2016, 18:09).

[20] Achour, M. et al.; Cowburn, P. [Hrsg.]: Offizielle Sprach- und Befehlsreferenz; http://php.net/manual/de/intro-whatcando.php (18.10.2016, 18:02).

[21] Achour, M. et al.; Cowburn, P. [Hrsg.]: Offizielle Sprach- und Befehlsreferenz; http://php.net/manual/de/funcref.php (18.10.2016, 18:16).

[22] php.net: Offizielle Sprach- und Befehlsreferenz; http://php.net/archive/2015.php#id2015-12-03-1 (18.10.2016, 18:52).

[23] Achour, M. et al.; Cowburn, P. [Hrsg.]: Offizielle Sprach- und Befehlsreferenz; http://php.net/manual/de/install.general.php (19.10.2016, 11:11).

[24] Achour, M. et al.; Cowburn, P. [Hrsg.]: Offizielle Sprach- und Befehlsreferenz; http://php.net/manual/de/features.commandline.options.php (19.10.2016, 11:27).

[25] Achour, M. et al.; Cowburn, P. [Hrsg.]: Offizielle Sprach- und Befehlsreferenz; http://php.net/manual/de/intro.stream.php (19.10.2016, 18:05).

[26] Achour, M. et al.; Cowburn, P. [Hrsg.]: Offizielle Sprach- und Befehlsreferenz; http://php.net/manual/de/function.stream-context-create.php (19.10.2016, 18:09).

[27] Achour, M. et al.; Cowburn, P. [Hrsg.]: Offizielle Sprach- und Befehlsreferenz; http://php.net/manual/de/context.http.php (19.10.2016, 17:12).

[28] w3.org: HTML 4.01 Specification, https://www.w3.org/TR/html401/interact/forms.html#h-17.13.4.1 (19.10.2016, 18:35).

[29] Achour, M. et al.; Cowburn, P. [Hrsg.]: Offizielle Sprach- und Befehlsreferenz; http://php.net/manual/de/function.http-build-query.php (19.10.2016, 18:42).

[30] Achour, M. et al.; Cowburn, P. [Hrsg.]: Offizielle Sprach- und Befehlsreferenz; http://php.net/manual/de/stream.contexts.php (20.10.2016, 18:56).

[31] Achour, M. et al.; Cowburn, P. [Hrsg.]: Offizielle Sprach- und Befehlsreferenz; http://php.net/manual/de/function.file-get-contents.php (20.10.2016, 18:00).

[32] esa.int: European Space Agency; http://www.esa.int/Our_Activities/Navigation/Galileo/What_is_Galileo (05.10.2016, 10:28).

[33] gps.gov: Space Segment des GPS; http://www.gps.gov/systems/gps/space/ (05.10.2016, 11:14).

[34] clinton.nara.gov: Offizielles Statement zur Abschaltung von SA; http://clinton3.nara.gov/WH/EOP/OSTP/html/0053_2.html (05.10.2016, 17:56).

[35] gps.gov: GPS Standard Positioning Service (SPS) Performance Standard; http://www.gps.gov/technical/ps/2008-SPS-performance-standard.pdf (04.10.2016, 18:10).

[36] NGA.mil: National Geospatial-Intelligence Agency; http://earth-info.nga.mil/GandG/publications/tr8350.2/wgs84fin.pdf (04.10.2016, 18:02).

[37] gps.gov: Control Segment des GPS; http://www.gps.gov/systems/gps/control/ (05.10.2016, 11:24).

[38] de Lange, Norbert: Geoinformatik: in Theorie und Praxis, Springer-Verlag Berlin Heidelberg GmbH, 2013, S. 167 f..

[39] (Quelle: Bartlett, D: The Cambridge Wireless Essentials Series : Essentials of Positioning and Location Technology, 2013, S. 14 [Bartlett].).

[40] in Bray, T. et al.: Extensible Markup Language (XML) 1.0 (Fifth Edition), https://www.w3.org/TR/xml/ (27.10.2016, 13:29).

[41] topografix.com: GPX: the GPS Exchange Format, http://www.topografix.com/gpx.asp (27.10.2016, 13:33).

[42] topografix.com: GPX 1.1 Schema Documentation, http://www.topografix.com/GPX/1/1/ (27.10.2016, 13:41) .

[43] topografix.com: GPX Schema, http://www.topografix.com/GPX/1/1/gpx.xsd (27.10.2016, 13:56).

[44] Langbridge, James A. :Arduino Sketches : Tools and Techniques for Programming Wizardry, Wiley Verlag, 2015, S. 82-86.

[45] Titus, Jonathan: The Hands-on XBEE Lab Manual : Experiments that Teach you XBEE Wirelesss Communica-tions, Newness Verlag, 2012.

[46] Cihat, K.: Grundlagen der Steuerungstechnik, Springer Fachmedien Wiesbaden, 2013, S. 12.

[47] Dembowski, K.: Raspberry Pi - Das technische Handbuch, Springer Fachmedien Wiesbaden, 2015.

[48] Raspberrypi.org: Raspberry Pi 2 Modell B, https://www.raspberrypi.org/products/raspberry-pi-2-model-b/ (16.10.2016, 15:42).

[49] piusv.de: Dokumentation CW2. PiUSV+, http://www.piusv.de/support/piusvdoku.pdf (17.10.2016, 12:41), gespeichert als: piusvdoku.pdf.

[50] Emmerich: Datasheet ICR-18650NH, http://www.produktinfo.conrad.com/datenblaetter/250000-274999/251007-da-01-en-EMMERICH_LiION_AKKU_ICR_18650NH_SP.pdf (22.10.2016, 11:44), gespeichert als: 18650NH.pdf.

[51] waveshare.com: UART GPS NEO 6M User Manual; http://www.waveshare.com/w/upload/9/90/UART-GPS-NEO-6M-UserManual.pdf (04.10.2016, 21:48), gespeichert als neo-6m.pdf.

[52] elinux.org: RPi Serial Connection, http://elinux.org/RPi_Serial_Connection (25.10.2016, 15:19).

[53] Ronneburg, F.: Das Debian GNU/Linux Anwenderhandbuch, http://debiananwenderhandbuch.de/startstop.html (25.10.2016, 14:59).

[54] in Adams, J.: Raspberry Pi B+ (Reduced Schematics), https://www.raspberrypi.org/documentation/hardware/raspberrypi/schematics/Raspberry-Pi-B-Plus-V1.2-Schematics.pdf (25.10.2016, 16:11), gespeichert als: GPIO_Raspberry.pdf.

[55] Treffcorn, R. et al.: offizielle GPSD Homepage; catb.org/gpsd/gps.html (26.09.2016, 16:21).

[56] Achour, M. et al.; Cowburn, P. [Hrsg.]: Offizielle Sprach- und Befehlsreferenz; http://php.net/manual/de/function.shell-exec.php (26.09.2016, 17:26).

[57] packages.debian.org: Informationen über Paket procps in Debian wheezy; https://packages.debian.org/de/wheezy/procps (26.09.2016, 17:09).

[58] Achour, M. et al.; Cowburn, P. [Hrsg.]: Offizielle Sprach- und Befehlsreferenz; http://php.net/manual/de/function.fopen.php (26.09.2016, 17:23).

[59] Achour, M. et al.; Cowburn, P. [Hrsg.]: Offizielle Sprach- und Befehlsreferenz; http://php.net/manual/de/function.fgets.php (26.09.2016, 17:28).

[60] Achour, M. et al.; Cowburn, P. [Hrsg.]: Offizielle Sprach- und Befehlsreferenz; http://php.net/manual/de/function.fclose.php (26.09.2016, 17:30).

[61] Achour, M. et al.; Cowburn, P. [Hrsg.]: Offizielle Sprach- und Befehlsreferenz; http://php.net/manual/de/function.floatval.php (27.10.2016, 12:08).

[62] Achour, M. et al.; Cowburn, P. [Hrsg.]: Offizielle Sprach- und Befehlsreferenz; http://php.net/manual/de/function.simplexml-load-file.php (27.10.2016, 12:14).

[63] in Simpson, W.: The Point-to-Point Protocol (PPP), https://tools.ietf.org/html/rfc1661 (27.10.2016, 20:22), gespeichert als: rfc1661.pdf.

[64] Achour, M. et al.; Cowburn, P. [Hrsg.]: Offizielle Sprach- und Befehlsreferenz; http://php.net/manual/de/function.sleep.php (27.10.2016, 21:13).

[65] Achour, M. et al.; Cowburn, P. [Hrsg.]: Offizielle Sprach- und Befehlsreferenz; http://php.net/manual/de/function.explode.php (04.10.2016, 22:07).

[66] Achour, M. et al.; Cowburn, P. [Hrsg.]: Offizielle Sprach- und Befehlsreferenz; http://php.net/manual/de/function.file.php (21.10.2016, 16:51).

[67] Achour, M. et al.; Cowburn, P. [Hrsg.]: Offizielle Sprach- und Befehlsreferenz; http://php.net/manual/de/function.unset.php (21.10.2016, 16:56).

[68] Achour, M. et al.; Cowburn, P. [Hrsg.]: Offizielle Sprach- und Befehlsreferenz; http://php.net/manual/de/function.file-put-contents.php (21.10.2016, 17:10).

[69] Achour, M. et al.; Cowburn, P. [Hrsg.]: Offizielle Sprach- und Befehlsreferenz; http://php.net/manual/de/function.count.php (21.10.2016, 17:23).

[70] Achour, M. et al.; Cowburn, P. [Hrsg.]: Offizielle Sprach- und Befehlsreferenz; http://php.net/manual/de/control-structures.break.php (21.10.2016, 17:27).

[71] Achour, M. et al.; Cowburn, P. [Hrsg.]: Offizielle Sprach- und Befehlsreferenz; http://php.net/manual/de/reserved.variables.post.php (22.10.2016, 12:08).

[72] google.com: Die Google Maps Geocoding API, https://developers.google.com/maps/documentation/geocoding/intro?hl=de (22.10.2016, 12:24).

[73] Achour, M. et al.; Cowburn, P. [Hrsg.]: Offizielle Sprach- und Befehlsreferenz; http://php.net/manual/de/function.json-decode.php (22.10.2016, 12:41).

[74] Achour, M. et al.; Cowburn, P. [Hrsg.]: Offizielle Sprach- und Befehlsreferenz; http://php.net/manual/de/function.implode.php (22.10.2016, 13:15).

[75] Achour, M. et al.; Cowburn, P. [Hrsg.]: Offizielle Sprach- und Befehlsreferenz; http://php.net/manual/de/pdo.connections.php (22.10.2016, 18:51).

[76] Achour, M. et al.; Cowburn, P. [Hrsg.]: Offizielle Sprach- und Befehlsreferenz; http://php.net/manual/de/ref.pdo-sqlite.php (22.10.2016, 19:20).

[77] w3resource.com: SQLite INSERT INTO, http://www.w3resource.com/sqlite/sqlite-insert-into.php (22.10.2016, 20:21).

[78] Achour, M. et al.; Cowburn, P. [Hrsg.]: Offizielle Sprach- und Befehlsreferenz; http://php.net/manual/de/sqlite3.prepare.php (22.10.2016, 20:13).

[79] Achour, M. et al.; Cowburn, P. [Hrsg.]: Offizielle Sprach- und Befehlsreferenz; http://php.net/manual/de/pdostatement.execute.php (22.10.2016, 20:17).

[80] Achour, M. et al.; Cowburn, P. [Hrsg.]: Offizielle Sprach- und Befehlsreferenz; http://php.net/manual/de/function.is-object.php (31.10.2016, 14:06).

[81] nmea.de: NMEA 0183 Datensätze; http://www.nmea.de/nmea0183datensaetze.html#rmc (28.09.2016, 12:20).

[82] Achour, M. et al.; Cowburn, P. [Hrsg.]: Offizielle Sprach- und Befehlsreferenz; http://php.net/manual/de/function.rewind.php (17.10.2016, 17:13).

[83] nmea.de: NMEA 0183, http://www.nmea.de/index.html (20.10.2016, 18:38).

[84] csl-computer.com: CSL Panther Tabs, https://shimg.csl-computer.com/images/product_images/popup_images/99521427700579_0.jpg (29.10.2016, 14:02), gespeichert als: adapter.jpg.

Anlagen

A. Skript1

```
<?php
error_reporting(E_ALL);
date_default_timezone_set('Europe/Berlin');
ini_set('display_errors',1);
ini_set('log_errors',1);
ini_set('error_log','skript1.error.log');


//Aktivieren des GPS-Loggers
shell_exec('gpxlogger -d -i 3600 -f
/home/pi/Desktop/Sender/rohdaten.gpx');
?>
```

B. Shell-Skript

```
#!/bin/bash
php /home/pi/Desktop/Sender/skript2.php
exit 0
```

C. Skript2

```
<?php
error_reporting(E_ALL);
date_default_timezone_set('Europe/Berlin');
ini_set('display_errors',1);
ini_set('log_errors',1);
ini_set('error_log','sender2.error.log');


$url = 'URI des Servers';


// Der GPS-Logger wird beendet.
$gpsid = shell_exec('pgrep gpxlogger');
shell_exec('sudo kill $gpsid');


// Distanzberechnung
//Einlesen der alt-werte
$handle = fopen("/home/pi/Desktop/Sender/latalt.txt", "r");
$latalt = floatval(fgets ($handle));
fclose ($handle);
$handle = fopen("/home/pi/Desktop/Sender/lonalt.txt", "r");
$lonalt = floatval(fgets ($handle));
fclose ($handle);


//initiieren von variablen
$distance =0;
$anzahl =0;
$dx = 0;
$dy = 0;
```

```
//Einlesen des gpx-files
$xmlobjekt =
simplexml_load_file("/home/pi/Desktop/Sender/rohdaten.gpx");


//Überprüfen ob Daten vorhanden sind
$isobj = is_object($xmlobjekt->trk->trkseg);
if ($isobj = FALSE) {
shell_exec('sudo shutdown -h now');
}


//Berechnung
foreach( $xmlobjekt->trk->trkseg->{'trkpt'} as $trkpt ) {
     $latneu = floatval($trkpt->attributes()->lat);
     $lonneu = floatval($trkpt->attributes()->lon);
     $latbm = ($latneu + $latalt) *0.00875;
     $dx = 111.3 * cos($latbm) * ($lonalt - $lonneu);
     $dy = 111.3 * ($latalt - $latneu);
     $distance = $distance + round(sqrt(pow($dx,2) + pow($dy,2)),5);
     $lonalt = $lonneu;
     $latalt = $latneu;
     $endzeit = $trkpt->time;
     }
//Startwerte auslesen
$startzeit = $xmlobjekt->trk->trkseg->trkpt->time;
$startlat = $xmlobjekt->trk->trkseg->trkpt->attributes()->lat;
$startlon = $xmlobjekt->trk->trkseg->trkpt->attributes()->lon;
$ziellat = $latneu;
$ziellon = $lonneu;
```

```
//Zielwerte speichern
file_put_contents("/home/pi/Desktop/Sender/latalt.txt",$latalt);
file_put_contents("/home/pi/Desktop/Sender/lonalt.txt",$lonalt);

//Ein Routeneintrag in routen.txt wird vorgenommen.
$handle = fopen("/home/pi/Desktop/Sender/routen.txt", "a");
$route =
"$startzeit,$endzeit,$startlat,$startlon,$ziellat,$ziellon,$distance
";
fwrite ($handle, "$route\n");
fclose ($handle);

// Aufbau der Mobilfunkverbindung
shell_exec('sudo wvdial blau');
sleep (60);

// Senden der Daten
do {
     $handle = fopen("/home/pi/Desktop/Sender/routen.txt", "r");
     $zeile = fgets ($handle);
     $stringdaten = explode(",",$zeile);
     fclose($handle);
     $ubergabe = array(
     'startzeit'          =>   $stringdaten[0],
     'endzeit'            =>   $stringdaten[1],
     'startlat'           =>   $stringdaten[2],
     'startlon'           =>   $stringdaten[3],
     'ziellat'            =>   $stringdaten[4],
```

```
     'ziellon'            =>    $stringdaten[5],
     'distance'           =>    $stringdaten[6]
     );

$options = array(
     'http' => array(
                'header'  => "Content-type: application/x-www-form-
urlencoded\r\n",
                'method'  => 'POST',
                'content'=>http_build_query($ubergabe)
          )
     );
$context  = stream_context_create($options);
$result = (file_get_contents($url, false, $context));
if ($result == "OK") {
     $handle = fopen("/home/pi/Desktop/Sender/archiv.txt", "a");
     fwrite ($handle, "$zeile");
     fclose ($handle);
     $routenarray = file('/home/pi/Desktop/Sender/routen.txt');
     unset($routenarray[0]);
     file_put_contents('/home/pi/Desktop/Sender/routen.txt',$routena
rray);
     }
     else {
          echo "Uebertragung nicht erfolgreich!\n\n";
     }
$zeilenzahl = count(file('/home/pi/Desktop/Sender/routen.txt'));
if ($zeilenzahl == '0'){
          echo "Es wurden alle Routen uebertragen.\n";
```

```
            break;
      }
      } while ($result == "OK");

//beenden der Verbindung
$dialid = shell_exec('pgrep sudo wvdial dialer');
shell_exec('kill $dialid');
echo "Internet-Verbindung bendet \n";

//herunterfahren
shell_exec('shutdown -h now');
?>
```

D. Skript auf dem Server

```
<?php
error_reporting(E_ALL);
date_default_timezone_set('Europe/Berlin');
ini_set('display_errors',0);
ini_set('log_errors',1);
ini_set('error_log','receiver.error.log');


$ubergabe = $_POST;
//Start-Adresse hinter Koordinaten herausfinden durch google API
$url =
"https://maps.googleapis.com/maps/api/geocode/json?latlng=$ubergabe[
startlat],$ubergabe[startlon]";
$google_start = json_decode(file_get_contents($url));
$ubergabe['startadresse'] = $google_start->results[0]-
>formatted_address;



//Ziel-Adresse hinter Koordinaten herausfinden durch google API
$url =
"https://maps.googleapis.com/maps/api/geocode/json?latlng=$ubergabe[
ziellat],$ubergabe[ziellon]";
$google_ziel = json_decode(file_get_contents($url));
$ubergabe['zieladresse'] = $google_ziel->results[0]-
>formatted_address;


// Daten ins Archiv schreiben
$handle = fopen("archiv_empf.txt", "a");
$archiv = implode(",", $ubergabe);
fwrite ($handle, "$archiv\n");
```

```
fclose ($handle);

// zur DB hinzufuegen

$pdo = new PDO('sqlite:'.dirname(__FILE__).'/db.db');

     $query = 'INSERT INTO routen (startzeit, endzeit, startort,
endort, distance)

                VALUES ("'.$ubergabe['startzeit'].'",

                          "'.$ubergabe['endzeit'].'",

                          "'.$ubergabe['startadresse'].'",

                          "'.$ubergabe['zieladresse'].'",

                          "'.$ubergabe['distance'].'")';

     $stobj = $pdo->prepare($query);

     $stobj->execute();

     $pdo = null;



echo "OK"; //Erfolgreiche Übertragung mitteilen

?>
```

Selbstständigkeitserklärung

Hiermit erkläre ich, dass die Arbeit selbstständig verfasst, in gleicher oder ähnlicher Fassung noch nicht in einem anderen Studiengang als Prüfungsleistung vorgelegt wurde und keine anderen als die angegebenen Hilfsmittel und Quellen, einschließlich der angegebenen oder beschriebenen Software, verwendet wurden.

Ort, Datum, Unterschrift