Inhaltsverzeichnis

Was ist das Schlapphut-Projekt?

Beim Schlapphut-Projekt handelt es sich um einen selbst gebauten GPS-Tracker mit WLAN Funktion für das Auto. Zielsetzung des Projekt ist dabei, alle Bewegungen eines Autos zu erfassen, diese zu Dokumentieren und auswerten zu können. Das Bewegungsprofil des Autonutzers wird quasi ausspioniert, wodurch sich der Name des Projekts ableiten lässt. Ein Detektiv oder Spion ist auf den Autonutzer angesetzt. Umgangssprachlich ein Schlapphut.

Das Projekt besteht dabei aus drei Komponenten: Die erste Komponente stellt dabei die Schlapphut-Hardware da, welche im Auto platziert wird. Komponente zwei ist eine Android-App, welche zum Datenaustausch mit der Hardware verwendet wird. Die letzte Komponente ist auf einem Windows-PC anzutreffen. Das selbst entwickelte Programm „KmlConverter“ konvertiert eine eingehende SLP-Datei in eine KML-Datei, welche anschließend mit der Software „Google-Earth“ geöffnet und angezeigt werden kann.

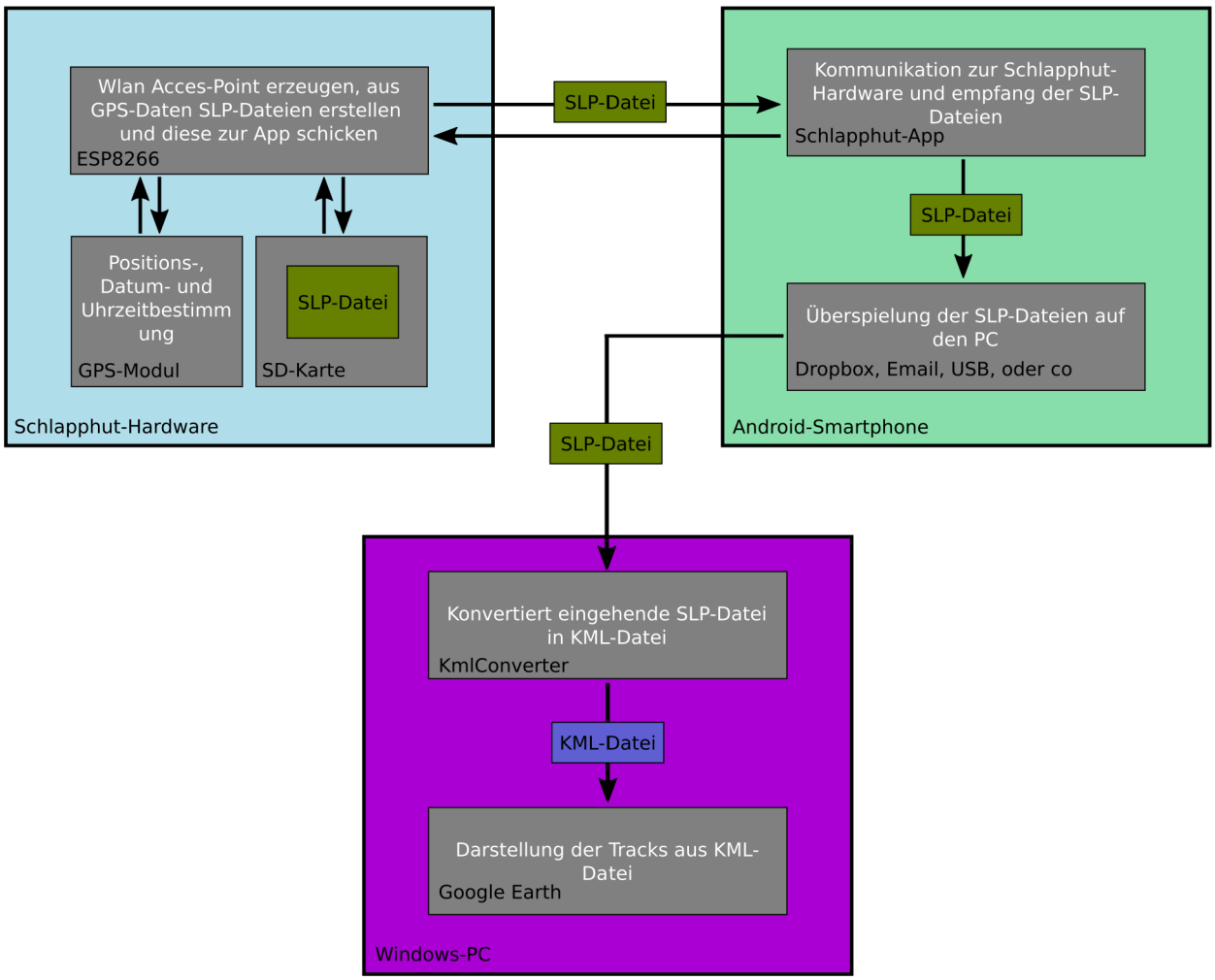


Abbildung 1: Zusammenspiel der Schlapphut -Komponenten

Vorstellung der Schlapphut-Komponenten

Schlapphut-Hardware

Die Hardware des Schlapphuts wird im zu überwachenden Auto versteckt. Als Stromversorgung wird die ungeschaltete Batteriespannung (Ausgangsspannung der Lichtmaschine) genutzt. Über eine Analyse der verschiedenen Spannungslevel der Lichtmaschine wird gleichzeitig detektiert, ob das Auto gerade in Betrieb ist (ob es fährt) oder ob es steht. Die Schlapphut-Hardware ist in einem Gehäuse für ODB-Analysegeräte verbaut. Ist die ODB-Schnittstelle des Autos etwas versteckt, zum Beispiel im Handschuhfach, kann die Hardware über diese Schnittstelle betrieben werden. Wem dies zu auffällig ist (wie mir), kann die Hardware auch im Sicherungskasten des Autos verstecken. Dabei wird über eine Sicherung mittels Stromdieb die Batteriespannung angezapft, die Metallfassung des Sicherungskasten, sprich die Fahrzeugkarosserie, dient als Masse.

Herzstück der Schlapphut-Hardware ist ein ESP8266 Mikrocontroller. Detektiert der Controller das losfahren des Autos, schaltet dieser ein angeschlossenes GPS-Modul ein. Das Modul liefert Positionsdaten mit Datum und Uhrzeit, welche vom Mikrocontroller eingelesen und verarbeitet werden. Anschließend werden die Daten in Form einer SLP-Datei, ein von mir erfundenes Dateiformat, auf eine angeschlossene SD-Karte gespeichert.

So speichert der Mikrocontroller bei jeder Autofahrt sekündlich die Positionsdaten ab und erstellt somit die Grundlage für ein Bewegungsprofil des Autofahrers. Um die Daten auf der SD-Karte zur weiteren Verarbeitung in die Außenwelt übertragen zu können, stellt der Mikrocontroller über einen Access-Point ein eigenes WLAN-Netz mit versteckter SSID bereit.

Schlapphut-Android-App

Die entwickelte Android-App hat die Aufgabe ausgewählte SLP-Dateien von der Schlapphut-Hardware herunterzuladen. Außerdem können einige Aufzeichnungsparameter eingestellt werden. Nach dem Starten der App verbindet diese das Smartphone mit dem vom ESP8266 Mikrocontroller erzeugten WLAN-Netz und lädt die Daten über das TCP-Protokoll auf den internen Handyspeicher. Fährt man also als Beifahrer im Auto mit, kann man ganz gemütlich die ausspionierten Daten runter laden. Anschließend werden die SLP-Dateien über Dropbox, E-Mail, USB oder ähnlichem auf einen Windows-PC zur weiteren Verarbeitung übertragen. Auf eine mögliche Verarbeitung direkt in der Android-App habe ich bewusst verzichtet, da ich eine Dateiverwaltung und eine gute Übersicht der Daten auf dem Handy zu unübersichtlich finde. Stichwort kleine Displaygröße und unübersichtlicher Dateiexplorer.

Der KmlConverter

Die Konsolenanwendung KmlConverter wandelt eine SLP-Datei in eine KML-Datei um, damit diese anschließend von Google-Earth geöffnet und zur Anzeige gebracht werden kann. Die KML-Datei ist das Wunschformat von Google-Earth. Theoretisch hätte das Konvertieren der SLP-Datei nach KML schon in der Android-App implementiert werden können. In Zukunft soll jedoch der KmlConverter zu einem größeren Auswertungsprogramm umgebaut werden. Die Anzeige in Google-Earth stellt bisher auch eher eine Übergangslösung dar.

Google Earth

Die mit dem KmlConverter erstellte KML-Datei wird mittels Google-Earth geöffnet. Ihr werden nun alle Fahrten des Autos in sogenannten Tracks mit Datumstempel dargestellt.

Hardware

Liste der verbauten Komponenten

Index	Name	Anzahl	Link
1	NodeMCU ESP8266 ESP-12E	1	Link
2	Teensy Micro SD Card Adaptor	1	Link
3	Ublox Neo-6m GPS Modul Flightcontroller GY-GPS6MV2 (nicht verwendet)	1	Link
4	GPS Receiver - GP-20U7 mit 56 Kanälen (verwendet)	1	Link
5	MP1584 3A XM1584 mini DC-DC Wandler step-down Modul 3A 0,8V - 20V LM2596	1	Link
6	OBD KFZ Diagnose Universal Gehäuse SET für OBD & RS232	1	Link
7	4er KFZ Mini Stromdieb Autosicherungen Steck Sicherung Verteiler Stromabgreifer	1	Link
8	Erico Caddy Schraubklemme Klemme EBC	1	Link
9	10 PCS SOP8 SO8 SOIC8 SMD to DIP8 Adapter PCB Board Convertor Double Sides	1	Link
	Komponenten von www.conrad.de	**Anzahl**	**Best.Nr.**
10	Trimmer linear 0.25 W 10 kΩ 270 ° Piher PT 15 NH 10K 1 St.	1	431893 - 62
11	Linear IC - Komparator Texas Instruments LM393AP Mehrzweck CMOS, MOS, Offener Kollektor	1	1010791 - 62
12	PMIC - Leistungsverteilungsschalter, Lasttreiber STMicroelectronics STMPS2141MTR High-Side SOIC-8	1	1185709 - 62
	Sonstiges	**Anzahl**	
13	Lochrasterplatine	1	
14	Widerstand 22 kOhm	1	
15	Widerstand 4,7 kOhm	2	
16	Widerstand 1,8 kOhm	1	
17	Widerstand 3,3 kOhm	1	
18	Jumper	1	
19	2er Stiftleiste für Jumper	1	
20	Kabel und Lötzinn		

D+ Simulator

Da der GPS-Tracker ca. 200 mA (200 mA war die erste Schätzung, tatsächlich sind es ca. 160 mA) im Betrieb benötigt, wäre eine Autobatterie mit einer Kapazität von 36 Ah innerhalb von 7,5 Tagen entladen. Aus diesem Grund soll der GPS-Tracker erst eingeschaltet werden, wenn das Auto sich in Betrieb befindet und die Lichtmaschine die Autobatterie lädt. Es muss also detektiert werden, wann die Lichtmaschine läuft.

In älteren Limas (Lichtmaschinen) wurde ein Zusatzverbraucher über den sogenannten „D+ Anschluss“ an die Spannung geschaltet , welcher ein Signal bereitstellt, sobald die Lichtmaschine in Betrieb ist. In neueren Limas ist dieser Anschluss jedoch nicht vorhanden oder schlecht erreichbar. Daher sind für ca. 50 Euro D+ Simulatoren zu erwerben, welche die Spannung der Autobatterie messen. Liegt diese Spannung über einen Schwellenwert (ca. 13,7 V), wird angenommen die Lichtmaschine lädt die Batterie. Nun wird der entsprechende Zusatzverbraucher mittels Relais eingeschaltet.

Da mir 50 Euro für einen entsprechenden Simulator zu teuer sind und dieser nur bedingt meine Anforderungen erfüllt, baue ich mir meinen eigenen Simulator.

Spannungen

Autobatterie: zwischen 12,4 und 12,7 Volt. Bei einer Spannung unter 12,4 Volt muss die Batterie aufgeladen werden

Lichtmaschine: Bei ca. 2000 U/min des Motors sollte die Lichtmaschine eine Spannung zwischen 13 und 14,5 Volt liefern. Je nach Lima kann diese Spannung auch schon im Standgas erreicht werden.

Mein D+ Simulator besteht aus einer Operationsverstärkerschaltung welche die Spannung an der Batterie überwacht. Steigt die Spannung auf 13,7 Volt, wird der Ausgang der Operationsverstärkers auf Low gezogen. Der ESP8266 wacht in einem eingestellten Zeitintervall aus dem Sleep-Modus auf und prüft die Spannung am Ausgang des OP‘s. Ist die Spannung 0V, fährt der ESP mit dem eigentlichen Programm fort. Ist die Spannung 3,3V legt der ESP sich wieder schlafen.

Bei Betrachtung des Schaltplans fällt auf, das die Operationsverstärkerschaltung theoretisch überflüssig ist. Der ESP verfügt über einen eigenen ADC. Dieser könnte im Zusammenspiel mit einem Spannungsteiler schon ausreichen, um die Spannung an der Autobatterie zu überwachen. Man könnte sogar feststellen, wenn sich die Ladung der Batterie einem kritischen Minimum nähern würde und entsprechend reagieren, um die Batterie nicht vollständig zu entladen. Dieser Punkt könnte ja eventuell ein einer Folgeversion implementiert werden.

Quellen Stand 08.06.2018

- https://de.wikihow.com/Autobatterien-pr%C3%BCfen
- https://de.wikihow.com/Eine-Lichtmaschine-kontrollieren

Stromverbrauch der Komponenten

- GPS-Modul: GY-GPS6MV2 (ist das alte Modul)
  - Ca. 65 mA
- GPS-Modul: GP-20U7 (das verbaute Modul)
  - nicht gemessen…
- ESP8266 im Soft Access Point Modus
  - Ca. 85 mA
- SD-Karten-Adapter mit eingesteckter SD-Karte
  - Ca. 1 mA
- DC-DC Converter, Komparatorschaltung, Controller im DeepSlepp-Modus
  - Ca. 5 mA
- Gesamtschaltung im Betrieb, SAP, SD-Karte und GPS (alt) an. Software Access Point keine Verbindung
  - Ca. 160 mA

Stromverbrauch der Gesamtschaltung wurde mit dem neuen GPS Modul (GP-20U7) nicht gemessen!

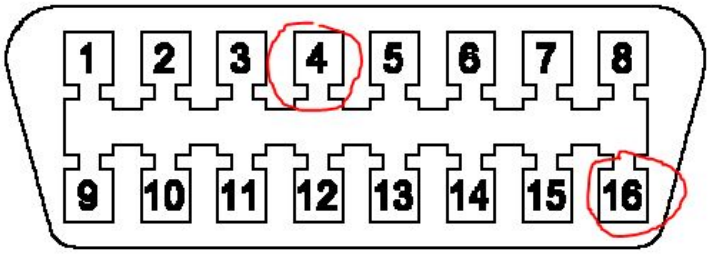
Platzierung der Hardware im Auto

ODB2-Schnittstelle

Die Schlapphut-Hardware kann über die OBD2-Schnittstelle mit Strom versorgt werden.

Pinbelegung der ODB2-Schnittstelle (Quelle: https://www.obd-2.de/stecker-belegungen.html) :

Diagnosebuchse (weiblich) im Fahrzeug:



Pin-Nr.	Beschreibung
1	Hersteller spezifisch
2	J1850 Bus+
3	Hersteller spezifisch
4	Fahrzeug Masse
5	Signal Masse
6	CAN High (J-2284)
7	ISO 9141-2 K Ausgang
8	Hersteller spezifisch
9	Hersteller spezifisch
10	J1850 Bus
11	Hersteller spezifisch
12	Hersteller spezifisch
13	Hersteller spezifisch
14	CAN Low (J-2284)
15	ISO 9141-2 L Ausgang
16	Batterie (+)-Spannung

Bei Fahrzeugen ab Baujahr 1996 kann anhand der Steckerbelegung bestimmt werden, welches Protokoll benutzt wird:

Stift (Signal)	Stift (Masse)	Stift (Signal)	Stift (Signal)	Stift (+12 V)	Protokoll
--	4 + 5	7	15 *)	16	ISO 9141-2
2	4 + 5	--	10	16	PWM J1850
2	4 + 5	--	--	16	VPW J1850
--	4 + 5	6	14	16	CAN Bus

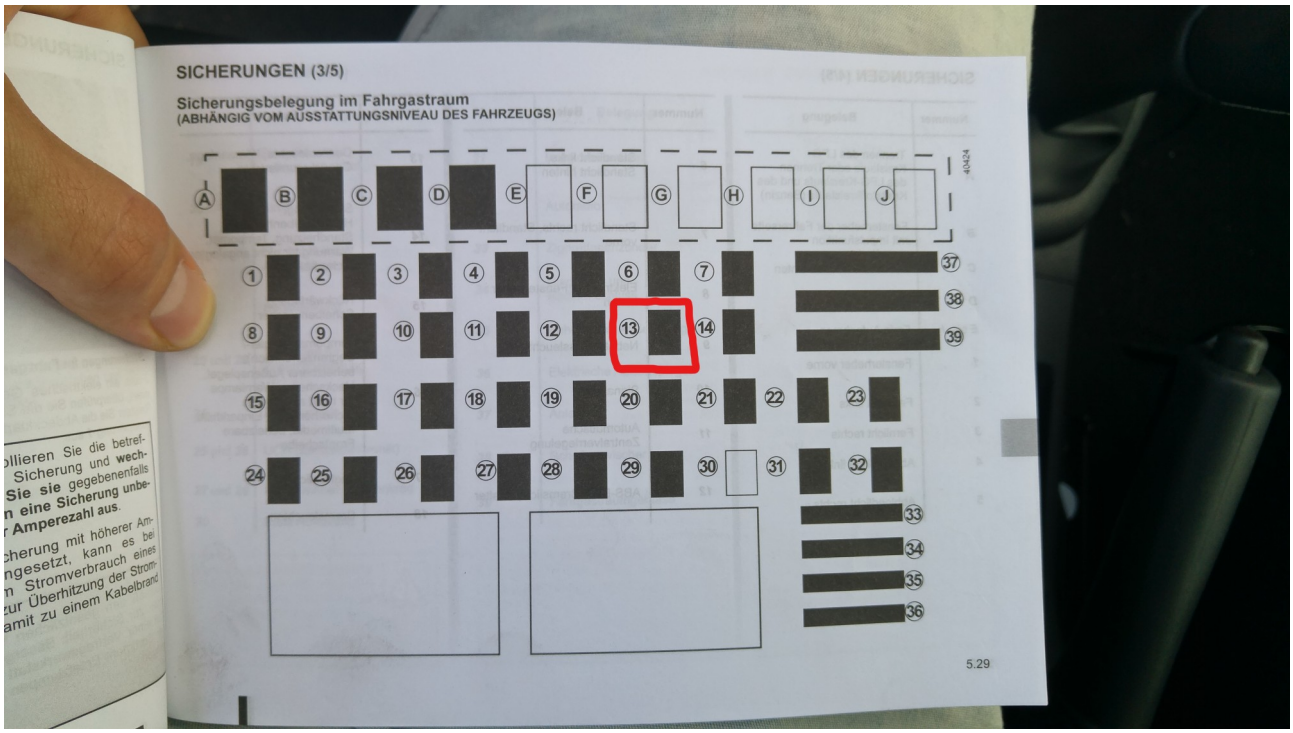
- *) Stift 15 kann, muß aber nicht für ISO 9141 vorhanden sein
- alle anderen Stifte sind für hersteller-spezifische Aufgaben reserviert und haben keine Relevanz für OBD-2

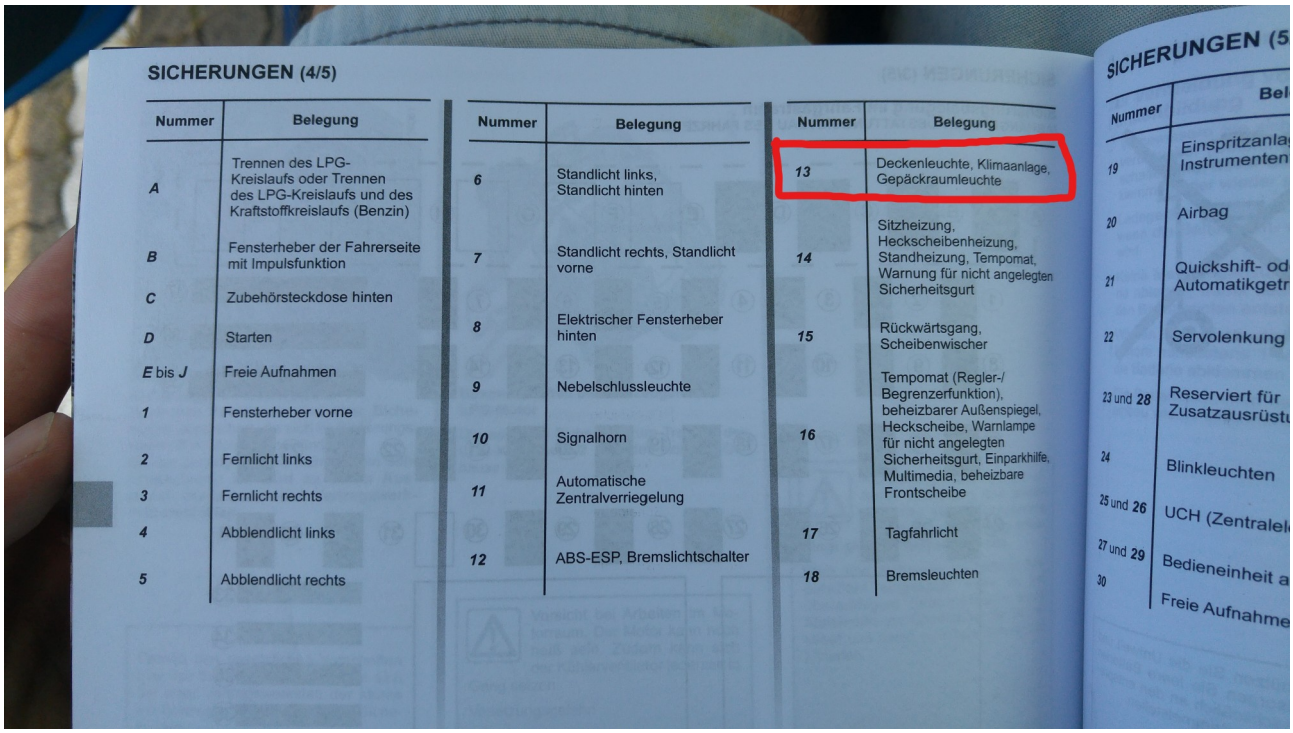
Sicherungskasten

Alternativ kann die Schlapphut-Hardware im Sicherungskasten versteckt werden. Der Strom kann mittels Stromdieb von einer Sicherung abgezweigt werden.

Wichtig: Es muss eine Sicherung sein, welche die ungeschaltete Spannung von der Lichtmaschine sichert. Einige Sicherungen sind erst nach einem Schalter geschaltet. Beispiel Abblendlicht: Durch die Sicherung fließt erst Strom, wenn das Abblendlicht vom Fahrer eingeschaltet wurde.

Ich habe die Sicherung für die Deckenleuchte, Klimaanlage und Gepäckraumleuchte verwendet.





SICHERUNGEN (4/5)

Nummer	Belegung
A	Trennen des LPG-Kreislaufs oder Trennen des LPG-Kreislaufs und des Kraftstoffkreislaufs (Benzin)
B	Fensterheber der Fahrerseite mit Impulsfunktion
C	Zubehörsteckdose hinten
D	Starten
E bis J	Freie Aufnahmen
1	Fensterheber vorne
2	Fernlicht links
3	Fernlicht rechts
4	Abblendlicht links
5	Abblendlicht rechts

Nummer	Belegung
6	Standlicht links, Standlicht hinten
7	Standlicht rechts, Standlicht vorne
8	Elektrischer Fensterheber hinten
9	Nebelschlussleuchte
10	Signalhorn
11	Automatische Zentralverriegelung
12	ABS-ESP, Bremslichtschalter

Nummer	Belegung
13	Deckenleuchte, Klimaanlage, Gepäckraumleuchte
14	Sitzheizung, Heckscheibenheizung, Standheizung, Tempomat, Warnung für nicht angelegten Sicherheitsgurt
15	Rückwärtsgang, Scheibenwischer
16	Tempomat (Regler-/Begrenzerfunktion), beheizbarer Außenspiegel, Heckscheibe, Warnlampe für nicht angelegten Sicherheitsgurt, Einparkhilfe, Multimedia, beheizbare Frontscheibe
17	Tagfahrlicht
18	Bremsleuchten

SICHERUNGEN (5

Nummer	Bel
19	Einspritzanla Instrumenten
20	Airbag
21	Quickshift- od Automatikgetr
22	Servolenkung
23 und 28	Reserviert für Zusatzausrüst
24	Blinkleuchten
25 und 26	UCH (Zentralel
27 und 29	Bedieneinheit a
30	Freie Aufnahme


30
30
10
5
5
5
52
5
52
5
10
10
10
10
52
52
52
30
30
15
15
15
15
52
1.5
1.5
5
5
52
02
15

Wieso sind zwei GPS-Module gelistet?

Bei Betrachtung der Hardwareliste fällt auf, dass dort zwei GPS-Module gelistet sind. Aber warum? Meine ersten Versuche habe ich mit dem GY-GPS6MV2 Modul unternommen. Ansich funktioniert das Modul gut, nur die Zeitspanne bis zum ersten GPS-Fix ist teils ziemlich hoch (bis zu 10 Minuten). Daher habe ich eine Alternative, das GP-20U7, verwendet. Dieses findet den ersten GPS-Fix nach subjektiver Wahrnehmung schneller.

Da beide Module den Ublox Neo-6m IC verwenden, und beide das NMEA-Protokoll abdecken, kann zwischen den Modulen gewechselt werden.

Edited: Nach einigen Recherchen wurde die Ursache der langen TTFF (Time To First Fix) gefunden. Ist das GPS-Modul über mehrere Stunden (länger als 4 Stunden) ausgeschaltet, sind die gespeicherten Satellitenbahndaten (Ephemeriden) veraltet und müssen erneut runter geladen werden. Dies wird auch als Cold-Start bezeichnet.

Abhilfe kann ein sogenanntes A-GPS System leisten. Hier werden die aktuellen Satellitenbahndaten von von einem „Assistenten“ bereitgestellt. Dies kann kein Server sein, der die Daten bereitstellt. Natürlich muss eine Internetverbindung zu dem Server bestehen. Nicht für den Anwendungsfall geeignet. Allerdings verfügt der Ublox Neo-6m IC über ein „AssistNow Autonomous“ Verfahren, welches die Satellitenbahndaten selber berechnen kann. Eventueller Punkt für eine Zweite Version des Schlapphuts?

Mehr dazu:

- http://www.kowoma.de/gps/agps.htm
- https://www.u-blox.com/sites/default/files/products/documents/u-blox6_ReceiverDescrProtSpec_%28GPS.G6-SW-10018%29_Public.pdf

Software – ESP8266

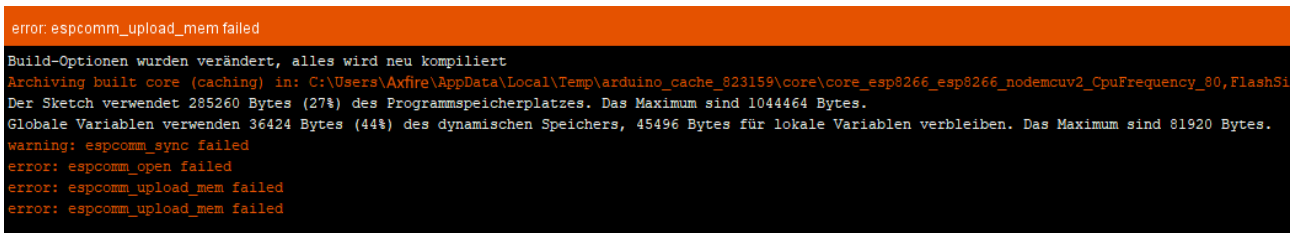
Liste der verwendeten Software-Libraries

- ESP8266 Arduino Core
  - kann in der Arduino IDE runtergeladen werden
  - https://arduino-esp8266.readthedocs.io/en/latest/index.html
- TinyGPS
  - GPS-Library
  - https://github.com/mikalhart/TinyGPS
- EspSoftwareSerial
  - angepasste Version von SoftwareSerial für den ESP
  - https://github.com/plerup/espsoftwareserial
- Arduino SD Library
  - https://www.arduino.cc/en/Reference/SD

Fehler beim Hochladen eines Arduino-Sketch

Manchmal kommt es beim Hochladen eines Arduino-Sketch zu einem Fehler und das Hochladen wird abgebrochen.

Fehlermeldung



Gesamte Meldung:

Arduino: 1.8.5 (Windows 10), TD: 1.41, Board: "NodeMCU 1.0 (ESP-12E Module), 80 MHz, 4M (1M SPIFFS), v2 Lower Memory, Disabled, None, Only Sketch, 115200"

Archiving built core (caching) in:
C:\Users\Axfire\AppData\Local\Temp\arduino_cache_593997\core\core_esp8266_esp8266_nodemcuv2_CpuFrequency_80,FlashSize_4M1M,LwIPVariant_v2mss536,Debug_Disabled,DebugLevel_None____,FlashErase_none,UploadSpeed_115200_372c9d9df1ff27add47b2318ef7f6992.a

Der Sketch verwendet 285260 Bytes (27%) des Programmspeicherplatzes. Das Maximum sind 1044464 Bytes.

Globale Variablen verwenden 36424 Bytes (44%) des dynamischen Speichers, 45496 Bytes für lokale Variablen verbleiben. Das Maximum sind 81920 Bytes.

warning: espcomm_sync failed

error: espcomm_open failed

error: espcomm_upload_mem failed

error: espcomm_upload_mem failed

Lösung

Noch keine wirkliche Lösung gefunden. Abhilfe schafft die Verbindung zwischen GPIO16/Wake und den RST-Pin für das Hochladen des Sketch zu unterbrechen. In der verlöteten Version jedoch nicht mehr möglich…

Edit: Habe nun doch einen Jumper zwischen Wake und RST gelötet, um die Verbindung zum Hochladen eines Sketch zu unterbrechen.

Betriebsarten des Schlapphuts

In der ESP-Software wird zwischen zwei Betriebsarten unterschieden. Zum Einen kann die Betriebsart „Track“ oder zum Zweiten die Betriebsart „Point“ angenommen werden.

In der Betriebsart „Track“ wird geprüft, ob die Lichtmaschine des Autos in Betrieb ist, ob das Auto also fährt. Wenn ja, wird das GPS-Modul eingeschaltet und bei gültigem GPS-Empfang in einem Zeitintervall von einer Sekunde die Positionsdaten des Fahrzeugs in eine Track SLP-Datei geschrieben. Ist die Lichtmaschine über einen definierten Zeitraum aus, steht das Fahrzeug also, wird die Aufzeichnung der Positionsdaten beendet und der ESP legt sich schlafen.

Der Betriebsmodus „Point“ soll einzelne Fahrzeugpositionen aufzeichnen. In einem großen Zeitintervall, z.B. einmal die Stunde, startet der ESP in den „Point-Modus“. Hier wird geprüft, ob das Fahrzeug steht, wenn ja wird das GPS-Modul eingeschaltet und genau eine Fahrzeugposition in eine Point SLP-Datei gespeichert. Danach legt sich der ESP wieder schlafen.

Die Implementierung des Betriebsmodus Point wurde jedoch in der Software des ESP etwas vernachlässigt und ist daher in der Version 1.0 nicht wirklich funktionsfähig. In der Android-App wurden allerdings schon einige Einstellungen für den „Point-Modus“ eingebaut.

SLP-Datei

Der ESP8266 speichert alle GPS-Punkte auf der SD-Karte in unterschiedlichen SLP-Dateien. Pro Tag wird eine neue SLP-Datei für jeden Betriebsmodus angelegt, in welcher die Positionsdaten mit Datumstempel gespeichert werden. Die jeweiligen SLP-Dateien sind nach dem Datum benannt. Also: YYYYMMDD.SLP

SLP-Datei? Nie gehört, was ist das?

Die SLP-Datei ist eine Eigenkreation von mir. Eigentlich handelt es sich um eine normale Textdatei (.txt), welche in eine SLP-Datei (.slp) umbenannt ist. Sie lässt sich also mittels Texteditor öffnen und bearbeiten.

Wenn die erzeugten SLP-Datei von der SD-Karte auf einen Computer kopiert werden, sind diese anschließend in eine KML-Datei zu konvertieren, um von Google-Earth geöffnet werden zu können. Für das Konvertieren habe ich das Programm "KML-Converter" geschrieben. Dieses liest die SLP-Datei ein, analysiert diese und erzeugt eine KML-Datei. Da Windows eine SLP-Datei nicht kennt, kann als Standartprogramm für diesen Dateityp der KML-Converter ausgewählt werden. Doppelklickt man nun auf eine SLP-Datei, startet sich das Konsolenprogramm "KML-Converter" und überführt diese in das KML-Format. Anschließend beendet sich das Konverter-Programm wieder.

Anmerkung: Der Name für meinen Dateityp (.slp) ist an den Projektnamen "Schlapphut" angelehnt ;)

Aufbau einer SLP-Datei

Es soll kurz der Aufbau einer SLP-Datei beschrieben werden. Eine SLP-Datei kann entweder Track Informationen oder Point Information speichern. Um was für Informationen es sich in der Datei handelt, wird im Header der Datei deklariert.

Im Bulk sind alle Nutzdaten der Datei aufgeführt.

Header

Erste Zeile der Datei. Gibt an, ob es sich um Track oder Point Informationen in der Datei handelt.

Track, Inhalt der ersten Zeile: TYPE=TRACK

Point, Inhalt der ersten Zeile: TYPE=POINT

Unterschied zwischen Track und Point Types: Points werden vom KmlConverter als Points interpertiert. Für jeden Eintrag in der SLP-Datei wird ein Point in der resultierenden KML-Datei eingetragen. Ist die SLP-Datei im Track-Type, wird vom KmlConverter ein Track (eine Linie, wo man lang gefahren ist) eingetragen.

Bulk

Wenn im Header TYPE=TRACK steht:

#Sxx

lat/lon/DateTimeString/hdop/age

lat/lon/DateTimeString/hdop/age

lat/lon/DateTimeString/hdop/age

lat/lon/DateTimeString/hdop/age

…

#Sxx ist das „Startzeichen“ eines neuen Tracks. „xx“ ist dabei die Zeit in Sekunden, die benötigt wurde um den ersten GPS-Fix seit Systemstart zu finden.

Danach werden nach jedem neuen GPS-Fix die Daten zeilenweise abgespeichert:

lat: Latitude

lon: Longitude

DateTimeString: Datum und Uhrzeit des GPS-Fix im yyyymmddhhmmss Format in UTC

hdop: Genauigkeit, aus Source-Datei: // horizontal dilution of precision in 100ths

age: Alter dex GPS-Fix in ms

Das Ende eines Tracks wird mit dem Beginn eines neuen Tracks oder mit dem Dateiende symbolisiert.

Wenn im Header TYPE=POINT steht:

Genau wie beim Track-Type, nur das es kein Startzeichen, also kein „#Sxx“ gibt.

LOG-Datei

Hier wird jeder erster GPS-Empfang einer neuen Autofahrt in Form des aktuellen Dateinamens protokolliert. Also jedes Mal, wenn der Schlapphut ein Losfahren detektiert, daraufhin das GPS-Modul einschaltet und dieses den ersten GPS-Fix gefunden hat.

Hintergrund: Manchmal kommt es vor, dass der GPS-Empfang so schlecht ist, dass in der eingestellten Zeit kein GPS-Fix gefunden werden kann und der Schlapphut wieder schlafen geht. In diesen fällen wird ebenfalls ein Eintrag in der LOG-Datei getätigt. Bei Sichtung der LOG-Datei kann erkannt werden, wie oft kein GPS-Fix gefunden werden konnte:

Beispiel Inhalt einer LOG-Datei:

20181013.slp
20181014.slp
20181014.slp
20181014.slp
kein GPS-Fix gefunden ... gehe schlafen
20181015.slp
20181015.slp
20181015.slp
kein GPS-Fix gefunden ... gehe schlafen
20181016.slp
20181016.slp
20181016.slp
20181016.slp

Befehle über TCP

Vom Server (ESP8266) an den Client (Android App) gesendet

Auf eine Auflistung der Befehle wurde an dieser Stelle verzichtet. Bei Interesse bitte in den Sourcecode der Android-App oder des ESP gucken. Dort sollte es ausreichend dokumentiert sein.

Vom Client (Android App) an Server (ESP8266) gesendet

Auf eine Auflistung der Befehle wurde an dieser Stelle verzichtet. Bei Interesse bitte in den Sourcecode der Android-App oder des ESP gucken. Dort sollte es ausreichend dokumentiert sein.

Software – Windows / C#

Liste der verwendeten Software-Libraries

- SharpKML
  - Library um KML-Dateien erzeugen zu können
  - https://github.com/samcragg/sharpkml

Der KmlConverter

Der KmlConvertiert ist ein in C-Sharp (C#) geschriebenes Konsolenprogramm und konvertiert eine SLP in eine KML-Datei. Die Nutzung des Programms kann auf zwei Arten erfolgen:

1. Den KmlConverter „normal“ starten. Nun per Drag and Drop eine SLP-Datei in das Konsolenfenster ziehen und mit Enter bestätigen. Der Konverter liest nun die Datei ein und erstellt eine KML-Datei.
2. Den Kml-Converter über „Öffnen mit...“ starten. Rechtsklick auf eine SLP-Datei und „Öffnen mit...“ klicken. In dem Dialog die Kml-Converter.exe auswählen und als Standardprogramm für diese Dateiendung verwenden. Nun weiß Windows, dass eine SLP-Datei mit dem KmlConverter geöffnet werden soll. Nun kann man auf eine beliebige SLP-Datei doppelklicken, der KmlConverter wird gestartet und konvertiert die Datei und schließt sich automatisch.

   Tipp: Mehrere SLP-Dateien im Windows-Explorer auswählen und mit Enter bestätigen. Nun werden alle ausgewählten Dateien auf einmal konvertiert.

Installation

Der Kml-Converter muss nicht installiert werden. Er kann einfach über die KmlConverter.exe gestartet werden. Es muss mindestens das .Net Framework 4.5 auf dem PC installiert sein.

Software - Android-App

Installation

Mindest Android-Version: Android 4.1 (JELLY_BEAN SDK-Version 16)

Die Android-App wird über ihre APK-Datei installiert (SchlapphutAndroidApp_v1_0.apk). Die APK-Datei muss dazu auf das Handy kopiert werden und dort ausgeführt werden (über den Dateiexplorer öffnen).

Tipp: Die APK-Datei über eine E-Mail auf das Handy schicken, dann kann diese einfach über den E-Mail-Anhang ausgeführt werden (mit der Gmail-App).

Wichtig: Da die App nicht signiert ist, muss vor der App-Installation eine Einstellungen im Handy gemacht werden (je nach Smartphonehersteller ist diese Option an anderen Einstellungsorten zu finden): Unter Einstellungen/Allgemein/Sicherheit muss der Punkt „Unbekannte Quellen – Installation von Apps aus anderen Quellen als Google Play Store erlauben“ aktiviert werden.

Mögliche Anpassungen/Erweiterungen für eine Neuauflage

- Zur Optimierung der Stromaufnahme und des Platzverbrauchs könnte kein Vollständiges ESP8266-Entwicklungsboard verbaut werden, sondern nur der Controller (ESP12-E oder ESP12-F).
- Die Operationsverstärkerschaltung für den D+ Simulator ist überflüssig, diese kann durch einen Spannungsteiler und den AD-Wandler des ESP ersetzt werden.
- Nutzung eines GSM GPRS Moduls, um die Positionsdaten direkt in Internet sendet zu können. So wäre ein Live-Modus möglich.
- Implementierung eines A-GPS Systems, um die TTFF-Zeit zu verkürzen. Eventuell würde auch ein häufiges (ca. einmal in der Stunde) anschalten des GPS-Moduls ausreichen (?)
- Versteck der Schlapphut-Hardware im Auto optimieren
- Möglichkeit für Over The Air Software-Updates der Schlapphut-Hardware

Linksammlung

- https://www.mikrocontroller-elektronik.de/nodemcu-esp8266-tutorial-wlan-board-arduino-ide/
- http://arduino-esp8266.readthedocs.io/en/latest/index.html
- https://www.esp8266.com/viewtopic.php?f=29&t=13715&p=62382&hilit=Search+this+forum%E2%80%A6tcp#p62382
- https://www.hackster.io/ruchir1674/how-to-interface-gps-module-neo-6m-with-arduino-8f90ad
- https://www.arduino.cc/en/Reference/SD
- https://www.myandroidsolutions.com/2012/07/20/android-tcp-connection-tutorial/#.Wz4u8bgyWUk
- https://www.electronics-tutorials.ws/de/operationsverstarker/opamp-komparator.html
- https://randomnerdtutorials.com/esp8266-deep-sleep-with-arduino-ide/