

Bau eines Wardriving-Moduls

Christian Stahl
Nico Feld

16. Oktober 2018
Ausarbeitung
Hardwarenahe Systemprogrammierung
Prof. Dr. Sturm

Inhaltsverzeichnis

1	Motivation & Idee	2
2	Aufbau & Probleme	2
2.1	Teensy 3.6	3
2.2	GPS-Modul	3
2.3	WLAN-Modul	4
2.4	Bluetooth-Modul HC-05	4
2.5	GSM-Modul SIM800L	6
3	Fazit	6

1 Motivation & Idee

Diese Arbeit beschäftigt sich mit dem Erstellen eines Wardriving-Moduls. "Wardriving" bezeichnet dabei das systematische Erfassen von WLAN-Netzen an verschiedenen Positionen und deren anschließender Auswertung. Anschließend können dann die erfassten WLAN-Netze der Position zugeordnet werden und letztendlich kartographiert werden. Dafür wird ein extra Modul mit Positionsbestimmung (GPS) und WLAN-Komponente benötigt. Als Motivation kann einerseits das Finden von unsicheren Netzwerken und der anschließenden Meldung an den Besitzer oder aber auch einfach die Kartographie von offenen WLAN-Netzen für Touristen o.Ä. Im Rahmen des Moduls "Hardware-nahe Programmierung" haben wir sein solches Modul sowohl programmiert als auch zusammengebaut. Dabei haben wir das Modul um die Funktion erweitert Bluetooth-Geräte und ursprünglich auch die Netzbdeckung via GSM-Modul zu erfassen.

2 Aufbau & Probleme

Zur Erfassung der GPS-Daten, der WLANs, der Bluetooth-Geräte, sowie der GSM-Abdeckung, haben wir die verschiedenen Module sternförmig um einen Teensy 3.6 angeordnet, welcher die zentrale Koordination und Datenspeicherung übernimmt. Die verschiedenen Module werden per UART verbunden, die Spannungsversorgung liefert eine Powerbank per USB, teils wird sie über den Teensy weitergereicht. Dazu verwenden wir, wie in Abbildung 1 zu sehen, rote und schwarze Drähte, um sie von den Leitungen für logische Pegel zu unterscheiden.

Bei Start unseres Moduls initialisiert der Teensy die angeschlossenen Module. Anschließend wird in einer Schleife stets zunächst die GPS-Information abgefragt. Um Endlosschleifen zu vermeiden wurden alle Module um eine Abbruchbedingung (kill-count) erweitert, sodass kein Modul den Gesamtablauf blockieren kann. Sollte nach der gegebenen Anzahl Versuche keine verwertbare Information erfasst sein, so bricht die Erfassung des einzelnen Moduls ab. Wird kein GPS-Punkt gefunden, so haben wir die anderen Module nicht abgefragt, sondern eine Wartezeit von 20 Sekunden eingebaut, um zu verhindern, dass die Module heiß laufen, was bei ersten Implementierungen zu Problemen führte.

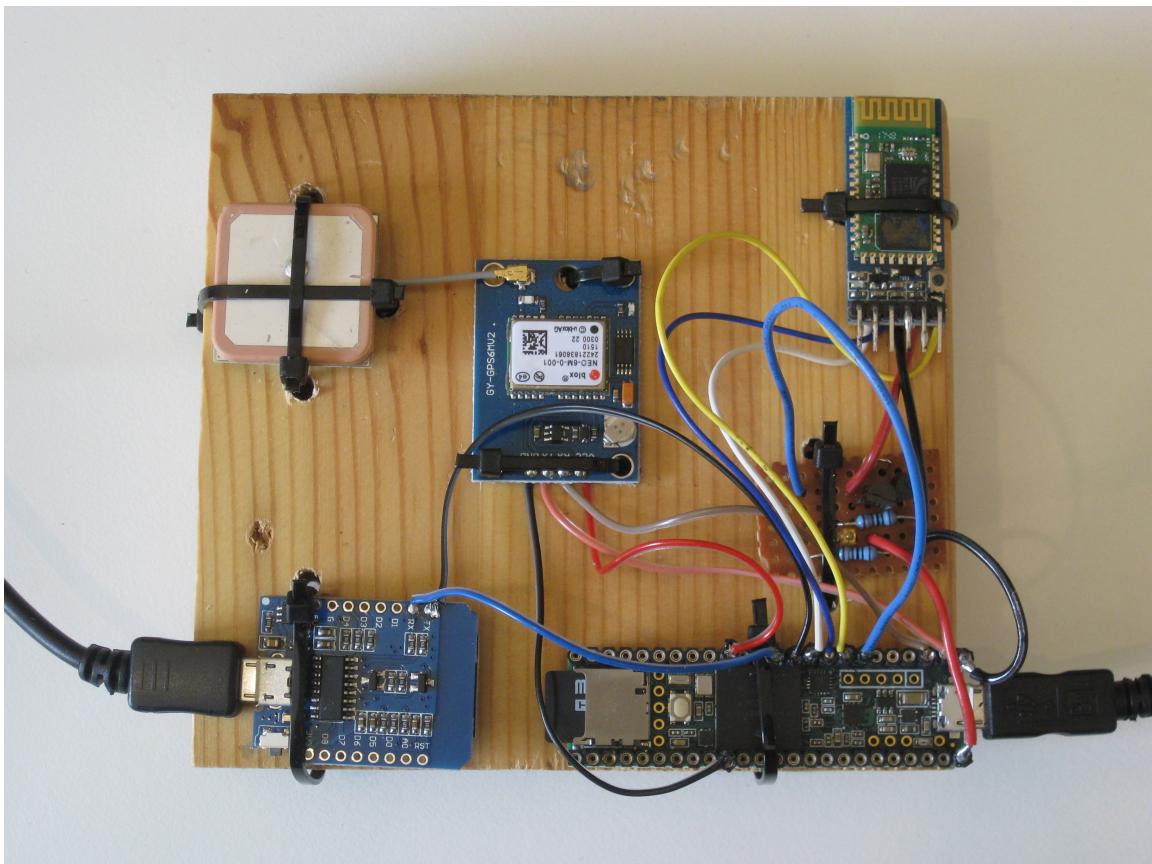


Abbildung 1: Aufbau unseres Wardriving-Moduls

2.1 Teensy 3.6

Als programmierbares “Herzstück“ des Moduls haben wir uns für den Teensy 3.6 entschieden, da er sowohl ausreichend Speicher besitzt um die ankommenden Daten zu verarbeiten als auch über einen eingebauten SD-Karten Slot verfügt, wodurch wir die erfassten Daten auf eine SD-Karte schreiben können. Des Weiteren konnte er sehr einfach mittels der “PlatformIO“-Erweiterung des Editors “Atom“ programmiert und gestartet werden. Die hohe Anzahl der Pins, genauer der UART-Pins, erlaubte uns außerdem die viele anderen Module ohne Schwierigkeiten zu anschließen.

2.2 GPS-Modul

Für das GPS-Modul verwenden wir einen Ublox NEO 6M. Das GPS-Modul stellt die wichtigste Komponente des Wardriving-Moduls dar, denn ohne die Positionsbestimmung kann keine Zuordnung der restlichen Daten erfolgen. Somit gibt dieses Modul auch den “Takt“ für die restlichen Module an: Erst wenn die Positionsbestimmung erfolgreich war, werden die Informationen der anderen Module (WLAN und Bluetooth) ausgewertet. Diese Positionsbestimmung erfolgt vom Modul jede Sekunden und wird per UART an den Teensy übertragen.

Die Informationen werden vom Modul im NMEA 0183-Standard¹ übermittelt. Dieser Standard beschreibt die Kodierung der erfassten Daten in ASCII-basierte Datensätze. Dabei werden vom Gerät mehrere dieser Datensätze pro Sekunde erfasst. In dieser Arbeit werden lediglich die beiden Datensätze *Recommended Minimum Sentence C (RMC)* und *Global Positioning System Fix Data*

¹https://de.wikipedia.org/wiki/NMEA_0183

(*GGA*) ausgewertet. Jeder dieser Datensätze beginnt immer mit einem Identifier. In diesem Fall wären das der Identifier “\$GPRMC,” bzw. “\$GPGGA,“. Anschließend folgen die Informationen des Datensatzes separiert mit Kommata. Eine mögliche Kodierung eines RMC-Datensatzes wäre also:

```
$GPRMC,162614,A,5230.5900,N,01322.3900,E,10.0,90.0,131006,1.2,E,A*13
```

Aus diesen Beiden Datensätzen werden in dieser Arbeit die Informationen *Status*, *Uhrzeit*, *Breitengrad*, *Längengrad*, *Anzahl der Satelliten* und *Höhe* ausgelesen. Falls der Status nicht “A“ beträgt, ist dies ein Zeichen dafür, dass die Positionsbestimmung nicht erfolgreich verlief. Falls dies der Fall ist wiederholt das GPS-Modul die eine Positionsbestimmung solange bis eine erfolgreiche Bestimmung erfolgte. Bei einer erfolgreichen Positionsbestimmung werden die restlichen Informationen in der “gps.csv“ auf der SD-Karte gespeichert. Anschließend werden die Informationen der anderen Module abgefragt, welche anhand der Uhrzeit nun eindeutig der bestimmten Position zugeordnet werden können.

2.3 WLAN-Modul

Für die WLAN-Modul verwenden wir das einen “WEMOS D1 MINI“. Dieses Modul benötigt eine eigene Firmware, welche in unserem Fall lediglich jede Sekunde alle WLAN-Netze in der Umgebung erkennt, anschließend jedes Netzwerk nach BSSID, Name (SSID), Typ der Verschlüsselung, Channel, Sichtbarkeit und RSSI (Verbindungsstärke) scannt und schließlich diese Daten per UART an den Teensy überträgt. Da die Abfrage im Teensy nach diesen Daten (`wifi->available()`) jederzeit eintreten kann, also auch während des Schreibprozesses des WLAN-Moduls, ist es wichtig nur die Daten auszuwerten wenn sie vollständig sind und erst zu beenden, wenn alle Daten übertragen wurden. Um dies zu gewährleisten fängt jede Zeile, die vom WLAN-Modul übertragen werden mit “42,“ an und das Ende der Übertragung wird mit der Zeile “end“ gekennzeichnet. Eine mögliche Ausgabe des Moduls wäre somit:

```
42,54:67:51:42: CF:E0,KabelBox-6700,WPA2/PSK,1,0,-90  
42,46:67:51:42: CF:E0,Vodafone Hotspot,open,11,0,-94  
42,A0:E4:CB:C4:86:B1,irie,WPA2/PSK,6,0,-92  
42,D0:6F:82:C7:CF:35,WLAN-QNN5B4,WPA2/PSK,0,0,-91  
end
```

Diese wird wie folgt vom Teensy interpretiert: Beginnt eine Zeile weder mit “42,“ noch mit “end“ wird diese Zeile verworfen, da es sich um Reste einer vorherigen Übertragung handelt. Andernfalls können durch die aktuelle Uhrzeit des GPS-Moduls die so übermittelten Daten der Position eindeutig zugeordnet werden. Um jedoch redundante Informationen, wie Name oder Typ der Verschlüsselung nicht jedes mal zu speichern, wenn ein Netzwerk entdeckt wird, werden zwei CSV-Dateien erzeugt. Wird ein Netzwerk mit bisher unbekannter BSSID erkannt so werden lediglich die Informationen *BSSID*, *Name*, *Verschlüsselung*, *Channel* und *Sichtbarkeit* in die “networks.csv“ geschrieben. Handelt es sich jedoch um ein bereits bekanntes Netzwerk so werden die Informationen *Uhrzeit*, *BSSID*, *RSSI* in die “wifi.csv“ geschrieben. So sind alle benötigten Informationen stets über die BSSID eindeutig zuordenbar ohne Informationen unnötig oft zu speichern.

2.4 Bluetooth-Modul HC-05

Zur Erfassung der verfügbaren Bluetooth-Geräte verwenden wir ein HC-05-Modul. Dieses verfügt über zwei Betriebsmodi. Der AT-Modus dient der Konfiguration des Moduls und ist aktiv, wenn der Enable-Pin des Moduls auf 3.3V (high) liegt und anschließend die Spannungsversorgung zugeschaltet wird. In diesem Modus blinkt die LED auf dem Bauteil langsam, etwa im

Halbsekundentakt. Der zweite Betriebsmodus, der Inquire-Mode, ist aktiv, wenn das Modul mit Betriebsspannung versorgt wird und am Enable-Pin logisch 0 (low) anliegt.

Die besondere Schwierigkeit bei der Implementierung der Bluetooth-Geräteerfassung mithilfe des Moduls lag darin, dass das Modul, unseres Wissens nach aufgrund eines Firmware-Fehlers, nicht wie dokumentiert arbeitet. Der von uns benötigte Inquire-Modus soll laut Dokumentation mit dem Befehlscode „AT+INQ“ gestartet werden und soll mit „AT+INQC“ gestoppt werden können. Beide Befehle erzeugten bei unseren Tests lediglich Fehlermeldungen, wobei der von „AT+INQ“ geworfene Fehlercode „1F“ nicht dokumentiert ist.

Da wir den Inquire-Mode bereits beim Start des Moduls aktivieren können, das Modul jedoch dann durchgehend Daten an den Teensy sendet, haben wir ein Workaround entwickelt, um die beiden nicht-funktionsfähigen Befehle zu umgehen.

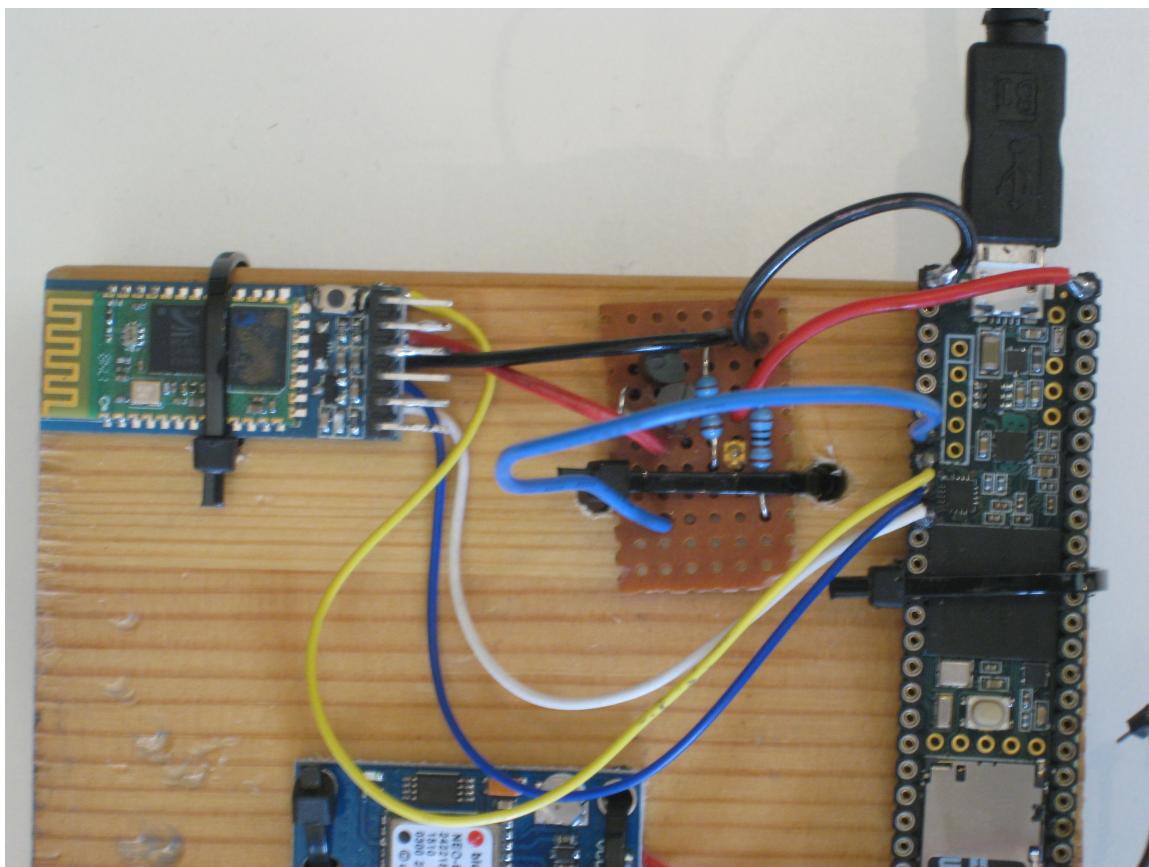


Abbildung 2: Bluetooth-Modul und Schaltung zur Spannungsversorgung

Zunächst wir das Modul im AT-Modus konfiguriert und anschließend abgeschaltet, indem es von der Stromversorgung getrennt wird. Diese Trennung führen wir durch einen logischen Pegel (hellblau) herbei, den wir an einen der Transistoren zwischen der Stromquelle (dem 5V-Pin des Teensy) und dem VCC-Pin des Moduls eingebaut haben. Zu Debug-Zwecken ist auf der Platine eine LED (zwischen den beiden Widerständen) verbaut, die leuchtet, solange das Modul mit Strom versorgt ist. Aufgrund der Hardwarelatenzen benötigen wir zum An- und Abschalten des Moduls jeweils circa 1,5 Sekunden, was die Zeit zur Erfassung aller Daten an einem GPS-Punkt deutlich verlängert. Um mit der Datenerfassung zu beginnen, aktivieren wir das Modul im Inquire-Mode und lesen die Ankommenden Daten ein. Nach 10 Sekunden schalten wir das Modul wieder ab und lesen die restlichen Daten, welche eventuell im Puffer des Teensy liegen ein. Wir

erhalten einen String, den wir an den Zeilenumbrüchen trennen, um die einzelnen Datentupel zu isolieren. Anschließend filtern wir mithilfe regulärer Ausdrücke die validen Tupel, um sie weiter zu verarbeiten. Dabei gehen wir ab diesem Punkt analog zum WLAN vor.

Durch die Latenzen und die relativ lange Erfassungszeit dauert eine Abfrage der Bluetooth-Geräte etwa 14,5 Sekunden und nimmt somit einen Großteil der gesamten Erfassungszeit ein. Diesen Kompromiss sind wir eingegangen, um die Erfassung der Bluetooth-Daten zu ermöglichen.

2.5 GSM-Modul SIM800L

Das SIM800L-Modul hat uns größere Schwierigkeiten bereitet, als alle anderen Module. Bis zuletzt war es uns nicht möglich es einzubinden. Eine Schwierigkeit bestand darin, dass verschiedene Bibliotheken für das Modul existieren, welche sich bei bereits bei der Initialisierung unterscheiden. Auch gibt es widersprüchliche Aussagen darüber, ob die verwendete Simkarte pingeschützt sein darf oder nicht.

Das größte Problem jedoch ist, dass das Modul, unabhängig von der verwendeten Bibliothek auf keine Eingaben reagiert. Es soll, wie alle anderen Module, per UART auf AT-Befehle reagieren, erzeugte bei unseren Tests jedoch nie eine Antwort. Für verschiedene Szenarien haben wir Schaltungen eingebaut und getestet. Ohne anliegende Spannung am Reset-Pin, mit Reset beim Start auf high und anschließend low, und konstant high oder low war es uns nicht möglich vom Modul eine Antwort zu erhalten. Somit mussten wir schließlich die Einbindung des GSM-Moudls aufgeben.

3 Fazit

*** Mit etwas mehr Geld wären funktionierende Komponenten drin gewesen, aber so mit den seltsamen Eigenschaften klar zu kommen war eine interessante Erfahrung -> wenn man sich die teile später nicht immer aussuchen darf. ***