Einführung

Das Projekt nennt sich “GPS-Tracker”. Das Hauptziel dieses Projekts ist, dass ein Gerät einer Zentrale die eigene Position mitteilt und diese Zentrale die Positionen speichern oder in einer HTML Karte umwandeln kann. Das Projekt besteht aus zwei Teile - das GPS-Tracker, der sogenannter Client(C Lang) und der Server(C++ Lang).

Der Client ist das Gerät, mit dem wir das Signal von den Satelliten abfangen können. Wenn das Signal einmal von dem Gerät abgefangen ist, wird es bearbeitet und weiter an dem Server geleitet.

Der Server kann nach Geräten suchen, Nachrichten von den schon gefunden Geräten lesen und die in einem Datei speichern, oder eine HTML Karte mit Hilfe von LeafletJS erstellen.

Client

1. Benötigtes Hardware

- NodeMCU
    - 32-Bit-Microcontroller
    - 128 RAM
    - 4MB Speicher
    - ESP8266
    - WLAN
- GPS Module (Neo-6M)
    - Externe Antene
    - Bekommt unbearbeitete Daten von Satelliten
- OLED Display Module (128x64px)

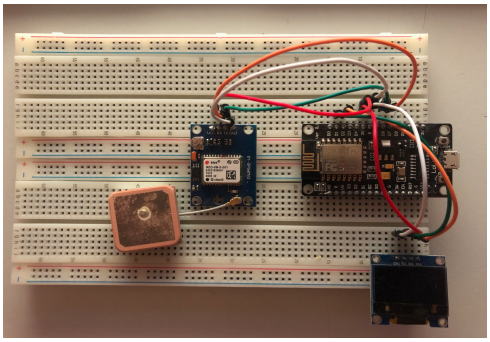
2. Aufbau des Clients

NodeMCU & GPS Module:

NodeMCU 3v3 mit GPS Module VCC
NodeMCU GND mit GPS Module GND
NodeMCU D6 (GPIO12) mit GPS Module TX
NodeMCU D7 (GPIO13) mit GPS Module RX

NodeMCU & OLED Display Module:

NodeMCU 3v3 mit OLED Display Module VCC
NodeMCU GND mit OLED Display Module GND
NodeMCU D1 (GPIO5) mit OLED Display Module SCL
NodeMCU D2 (GPIO4) mit OLED Display Module SDA

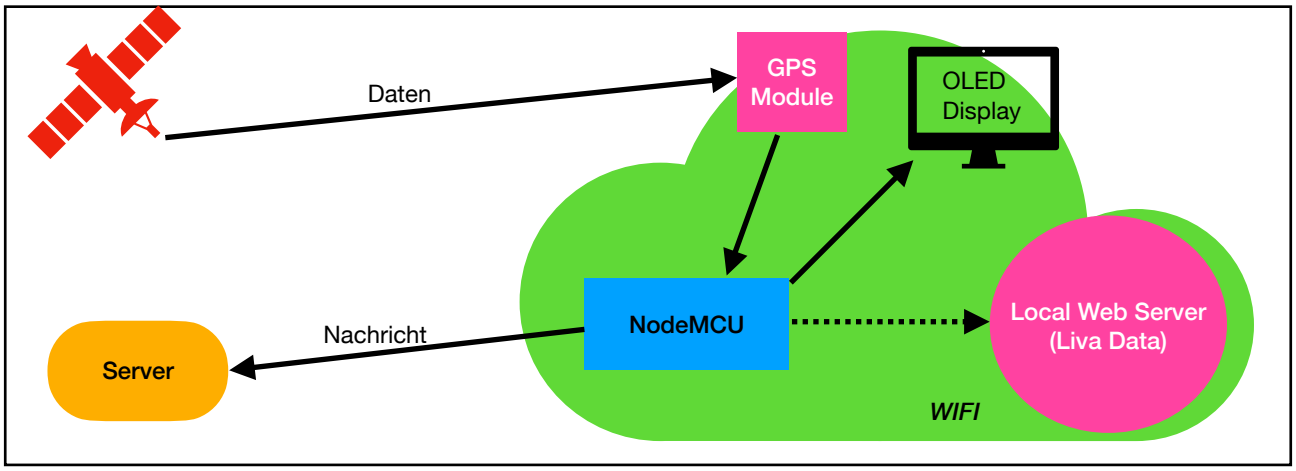


3. Wie wird das Programm auf dem Gerät installiert?

Vollständige Information zur Installierung in README.

4. Was sind die Hauptfunktionen des Geräts?

Das Gerät kann aktuelle Daten und Informationen zur Position von Satelliten darstellen, WiFi Verbindung erstellen und Informationen in einem eigenen Web Server darstellen. Die bearbeiteten Daten von den Satelliten sind: Latitude, Longitude, Date & Time. Der eigene Web Server kann nach einer erfolgreichen WiFi Verbindung unter dem gegebenen IP Adress im Browser erreicht werden. Da sind die Aktuellen Daten (die auf dem Display zu sehen sind) in einer tabellarischen Form dargestellt. Nachdem die aktuellen Daten vom GPS Module gelesen werden, werden die mit den vorherigen verglichen. Falls die sich unterscheiden, werden per UDP an dem Server gesendet.



5. Wie funktioniert der Client anhand des Codes?

Am Anfang werden die benötigten Bibliotheken hinzugefügt, Konstanten, Klassen und Geräte werden definiert bzw. initialisiert. Nachdem alles hinzugefügt, definiert und initialisiert ist, wird die erste Hauptfunktion aufgerufen ***setup()***. Diese Funktion ist eine Einführungsfunktion bzw. eine Startpunktfunktion des Programms. Die Setup Funktion wird nur einmal ausgeführt, und drinnen werden Teile des Programms definiert, die nicht als Konstanten zu sehen sind.

```
Serial.begin(115200);
ss.begin(9600);
```

Nachdem die Baud Rate definiert ist, müsste überprüft werden, ob eine Verbindung mit dem OLED Display erstellt werden kann. Falls eine Verbindung mit dem Display nicht möglich ist, wird eine Endlosschleife das Programm stoppen.

```
// SSD1306_SWITCHCAPVCC = generate display voltage from 3.3V internally
  if(!display.begin(SSD1306_SWITCHCAPVCC, SCREEN_ADDRESS)) {
    Serial.println(F("SSD1306 allocation failed"));
    for(;;); // Killing the program
  } else {
    Serial.println("SSD1306 allocation success");
  }
```

Nach diesem Schritt ist alles bereit zu starten. Eine Startnachricht wird gezeigt - ***startingDevice()***. Es wird versucht, mit den schon definierten *ssid* und *password* Konstanten eine Internetverbindung herzustellen und das wird so oft wiederholt, bis eine Verbindung hergestellt ist.

```
// ssid & password der WiFi Klasse übergeben.
WiFi.begin(ssid, password);
...
// Versuchen eine Internetverbindung herzustellen, bis wir eine haben
while (WiFi.status() != WL_CONNECTED) {
```

Ein Web Server wird erstellt, der dazu dient, die aktuellen Daten in dem lokalen Netzwerk zu zeigen.

Nach der ***setup()*** Funktion wird die zweite Hauptfunktion aufgerufen - ***loop()***. Diese Funktion dient dazu, das Gerät ständig am laufen zu halten und das Programm nie zu beenden. In dieser ***loop*** Funktion wird nur auf die Daten von den Satelliten gewartet und werden nur dann bearbeitet, wenn die Daten verfügbar sind und lesbar sind.

```
while (ss.available() > 0) {
  if (gps.encode(ss.read())) {
```

Die Antenne, die ich benutze, ist nicht stark, d.h Störungen können häufig auftretten (wegen schlechtem Wetter, wenn man sich in einem Tunnel befindet, usw.). Aus diesem Grund wird jedes Mal geprüft, ob die Daten, die das Gerät bekommt, gültig sind.

```
if (gps.location.isValid()) { }
...
if (gps.date.isValid()) { }
...
if (gps.time.isValid()) { }
```

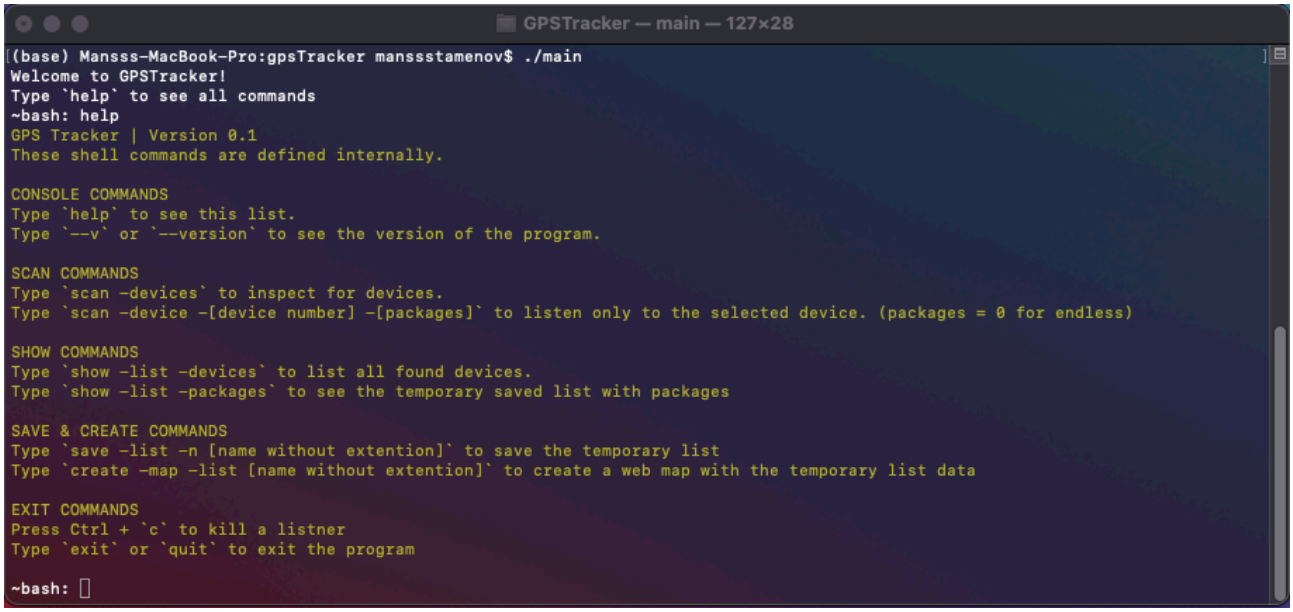
Falls valide Daten vorhanden sind, werden die Daten auf dem Display und im lokalen Web Server aktualisiert. Die Daten werden, aber nur dann an den Server gesendet, wenn die sich mit den vorherigen unterscheiden.

```
if( lat_str != prevPackage.lat || lng_str != prevPackage.lng
        || date_str != prevPackage.date || time_str != prevPackage.time) {
    sendUDP();
    prevPackage.lat = lat_str;
    prevPackage.lng = lng_str;
    prevPackage.time = time_str;
    prevPackage.date = date_str;
}
```

```
String udp_package = "";
udp_package += lat_str + ";";
udp_package += lng_str + ";";
udp_package += String(date) + String(".") + String(month)  + String(".") + String(year)   + String(";");
udp_package += String(hour) + String(":") + String(minute) + String(":") + String(second) + String(";");
....
Udp.beginPacket(IPAddress(192, 168, 0, 104), 1722);
Udp.write(udp_package.c_str());
Udp.endPacket();
```

Der Loop enthält am Ende des Blocks 2 Delays von insgesamt 2100ms.

Server



1. Wie wird das Programm installiert?

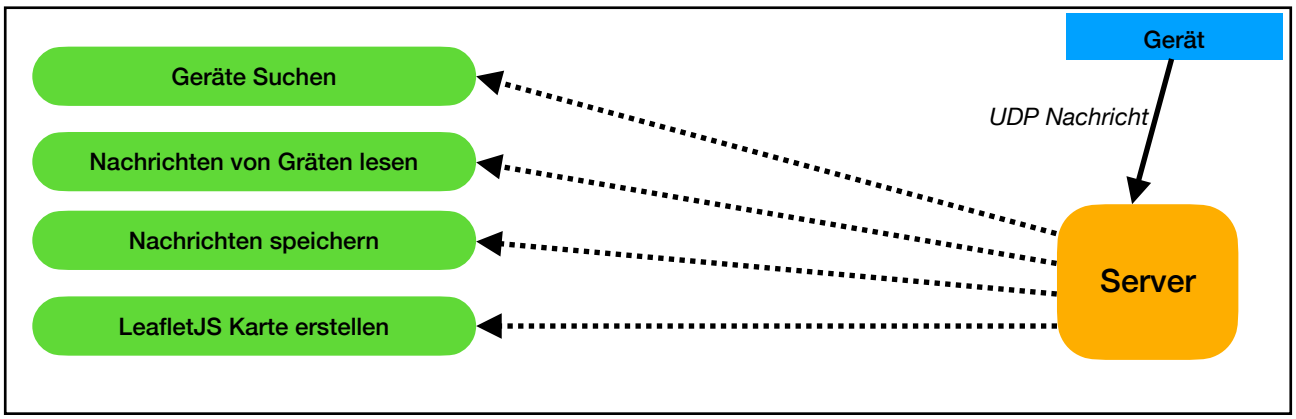
Vollständige Information zur Installierung in README.

2. Was sind die Hauptfunktionen des Programms?

Der Server kann nach verfügbaren Geräten suchen (***scan -devices***) und danach die gefundenen in einer Listenform darstellen (***show -list -devices***). Wenn man das gesuchte Gerät (GPS Tracker) in der Liste findet, kann man ***scan -devices -[die Nummer des Geräts in der Liste] -[Die Anzahl der gewünschte Nachrichten | 0 für nicht definierte Anzahl d.h bis ich es selber unterbreche (CTRL + C)]*** um Nachrichte von dem Gerät zu bekommen. Nachdem der Scan-Prozess beendet ist, kann man alle Nachrichte sehen: ***show -list -packages***.

Um die Daten zu speichern, muss man ***save -list -n [Name der Datei]*** eingeben und so werden die Nachrichten in einem *.gpstracker* Datei gespeichert.

Um eine Karte mit den gelesenen Nachrichten zu erstellen, muss man ***create -map -list [Name der Datei]*** eingeben. Nun wird eine HTML Datei mit Hilfe von LeafletJS erstellt, die mit einem beliebigen Browser (der Javascript unterstützt) geöffnet werden kann.



3. Wie funktioniert der Server anhand des Codes?

Der Server kann 10 UDP Nachrichten gleichzeitig bearbeiten.

```
const int MAX_THREADS = 10;
…
pthread_t threads [MAX_THREADS];
```

Wenn ein ***scan -devices*** ausgeführt wird, wird eine Funktion listen() aufgerufen, die am Anfang die benötigten Variablen vorbereitet.

```
// Define server & client
sockaddr_in serveraddr;
sockaddr_in clientaddr;

socklen_t addrlen = sizeof(clientaddr);
ssize_t receivedlen; // ssize_t = int
char buffer [PACKAGE_MAX_LENGTH]; // buffer sent in upd package
```

Es wird geprüft, ob ein Socket Listener existiert und falls nicht kann einer erstellt werden.

```
if ((fd = socket (AF_INET, SOCK_DGRAM, 0)) == -1) {
        std::cout << RED << "Socket creation failed.." << RESET << std::endl;
} else {
        std::cout << CYAN << "**Socket created.. " << RESET << std::endl;
```

Wenn eine Anbindung des Sockets mit dem Server IP erfolgreich ist, kann man weiter machen, ansonsten schlägt der Socket fehl.

```
// Bind the IP address and the port number to create the socket
if (bind (fd, (sockaddr*)&serveraddr, sizeof (serveraddr)) == -1) {
          std::cout << RED << "**Binding failed.." << RESET << std::endl;
} else {
          std::cout << CYAN << "**Binding succesful" << RESET << std::endl;
```

Der Port wird für den Socket belegt und jetzt können wir auf Nachrichten warten und Threads erstellen.

```
while (true) {
          // waiting to recieve the requests from client at port
          receivedlen = recvfrom (fd, buffer, PACKAGE_MAX_LENGTH, 0, (sockaddr*)
&clientaddr, &addrlen);
…
          pthread_create(&threads [threadno++], NULL, scanHandler, (void *)
req);
…
// Fill block of memory // clear memory
          memset(buffer, 0, sizeof(buffer));
```

Damit wir sicher sind, dass nach den gefundenen Geräten alles gut weiter läuft, schließen wir den Socket.

```
close(fd);
```

Auf dem gleichen Prinzip funktioniert auch ***scan -device -[das Gerät] -[Anzahl der Nachrichten]***. Aber der Unterschied liegt bei der Überprüfung, ob das Gerät existiert, bei der Gültigkeit der Anzahl der Nachrichten und ob die Anzahl der gelesenen Nachrichten erreicht ist.

Beim Speichern der Nachrichten ***save -list -n [Name der Datei]*** wird eine Datei erstellt und alle unbearbeiteten Nachrichten werden in der frisch erstellten Datei geschrieben.

Bei der Erstellung einer HTML Karte ***create -map -list [Name der Datei]*** wird die ***getMap*** Funktion aufgerufen, die ein Vector aus Strings erwartet (die Nachrichten) und liefert das ganze HTML Code als ein String zurück.

```
std::string html = getMap(PACKAGES);
```

Diese Funktion bearbeitet jede einzelne Nachricht und speichert die Lat, Long, Date & Time in String (JS Arrays). Die Markers zu den Positionen werden hinzugefügt und am Ende wird die Seite erstellt.

```
std::string html = "";
html += createHeader();
html += createBody(std::to_string(startLat), std::to_string(startLng), points,
markers, labels);
html += createFooter();
return html;
```

Nachdem die HTML String zurückgeliefert wird, wird eine Datei mit dem ausgewählten Namen und mit html Extension gespeichert und geschlossen.