Titel

Team 1: Eggs & Beacon

*Tobias Gilgenreiner, Felix Dollinger, Marco Stoiber,*

*Maximilian Kritzenthaler, Eduard Schröder*

Untertitel

# Inhalt

[1. Backend 2](#_Toc48821215)

[1.1. Systemhardware 2](#_Toc48821216)

[1.2. Einrichtung des Access Point 2](#_Toc48821217)

[1.3. Einrichten weiterer Systeme 4](#_Toc48821218)

[1.4. Programmausführung des zentralen Backend 4](#_Toc48821219)

[1.5. Programmausführung weiterer Systeme 5](#_Toc48821220)

[1.6. Zeitlicher Ablauf der Ausführungen 6](#_Toc48821221)

# Backend

Im Folgenden wird das Einrichten der Systeme als Access Point/Client, der Aufbau der Verbindungen vom/zum Backend und die laufenden Konsolen-Anwendungen genauer beschrieben.

## Systemhardware

*Getestet auf: Raspberry Pi 4, Raspberry Pi 3b+, Banana Pi M2+*

Die Systeme sollen auf Linux basieren. Raspbian als Betriebssystem ist am geeignetsten, da sämtliche inkludierten Bibliotheken direkt unterstützt werden sollten (besonders wiringPi.h/wiringSerial.h für UARTs am GPIO).

Es können mehrere Systeme verwendet werden, da dies erlaubt ggf. weitere UARTs Ports auszunutzen. So unterstützt theoretisch der Raspberry Pi 4 bis zu 6 UARTs (durch doppelte Belegung), aber es kann auch ein Setup erfolgen wie: Raspberry Pi 3b+ (1 GPIO UART + 1 USB UART) & ein Banana Pi M2+ (2 GPIO UARTs). Dies geschieht durch die Aufteilung auf die zwei Programme, welche in *1.4**Programmausführung des zentralen Backend*für das Access Point System und in *1.5 Programmausführung weiterer Systeme* für weitere verbundene Systeme erklärt werden.

## Einrichtung des Access Point

*Getestet auf: Raspberry Pi 4, Raspberry Pi 3b+*

Die Einrichtung findet nach dem offiziell von Raspberry vorgestellten Ablauf statt (Raspberry Pi, 2020)[[1]](#footnote-1). Zu beachten ist, dass abhängig von dem Modell unterschiedliche Konnektivität-Stabilitäten zu erwarten sind (Raspberry Pi 4 >> Raspberry Pi 3b+). Über LAN wird ein weiteres Netzwerk überbrückt, d.h. das über den Router somit noch Zugang zum Internet für das System und seine verbundenen Geräte hergestellt werden kann.

Die Einrichtung findet über die Konsole statt. Code nach dem Befehl „nano“ wird entsprechend in die Datei eingefügt und abgespeichert.

1. ***hostapd* installieren und beim Startvorgang aktivieren:**

sudo apt install hostapd

sudo systemctl unmask hostapd

sudo systemctl enable hostapd

1. ***dnsmasq* installieren für Netzwerk Management Dienste (DNS, DHCP)**

sudo apt install dnsmasq

1. **Tools für Firewall Regeln (Speicherung/Wiederherstellung) installieren**

sudo DEBIAN\_FRONTEND=noninteractive apt install -y netfilter-persistent iptables-persistent

1. **Netzwerk konfigurieren: Feste IP mit 192.168.4.1 (darf nicht bei Nutzen eines Routers über LAN vergeben sein)**

sudo nano /etc/dhcpcd.conf

interface wlan0

static ip\_address=192.168.4.1/24

nohook wpa\_supplicant

1. **Routing und IP masquerading aktivieren (falls Internet erwünscht ist)**

sudo nano /etc/sysctl.d/routed-ap.conf

net.ipv4.ip\_forward=1

sudo iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE

sudo netfilter-persistent save

1. **Konfigurieren der DHCP- und DNS-Dienste: Dynamische Zuweisung der Geräte. Zusätzlich sicherstellen, dass WiFi nicht blockiert wird.**

sudo mv /etc/dnsmasq.conf /etc/dnsmasq.conf.orig

sudo nano /etc/dnsmasq.conf

interface=wlan0 # Listening interface

dhcp-range=192.168.4.2,192.168.4.20,255.255.255.0,24h

# Pool of IP addresses served via DHCP

domain=wlan # Local wireless DNS domain

address=/gw.wlan/192.168.4.1

# Alias for this router

sudo rfkill unblock wlan

1. **Konfigurieren des Access Points: Nun die erwünschten Parameter eingeben.**

**Netzwerk Name bei *ssid* angeben.**

Netzwerk Passwort bei *wpa\_passphrase* angeben. (Zwischen 8-64 Chars)

sudo nano /etc/hostapd/hostapd.conf

country\_code=DE

interface=wlan0

ssid=RPI4

hw\_mode=g

channel=7

macaddr\_acl=0

auth\_algs=1

ignore\_broadcast\_ssid=0

wpa=2

wpa\_passphrase=INSERTPASSWORDHERE

wpa\_key\_mgmt=WPA-PSK

wpa\_pairwise=TKIP

rsn\_pairwise=CCMP

**Anm.:** Der Channel wird nicht automatisch festgelegt, d.h. falls es mit einem umgebenen Router eine Kollision gibt wird der Raspberry mit aller Wahrscheinlichkeit verlieren und kein Netzwerk anbieten! Dies ggf. dann an die Situation anpassen.

1. **Neustarten: Access Point wird fortan mit dem Starten aktiviert.**

sudo systemctl reboot

## Einrichten weiterer Systeme

*Getestet auf: Banana Pi M2+*

Damit sich weitere Systeme mit den Hauptsystem, also dem Access Point verbinden, muss nur eine WLAN-Verbindung zwischen diesen Systemen aufgebaut werden.

Dafür sucht man sich das Netz, das in der *hostapd.conf* von dem eingerichteten Access Point angegeben ist. Also die *ssid* als Name des Netzes und *wpa\_passphrase* als Passwort.

In der *main.cpp* des zentralen Backend muss noch ein weiterer Thread angelegt werden, welcher diese Daten ausließt. Dies erfolgt ähnlich der Funktion *thread\_get\_rpi3bplus\_data()* in der *main.cpp.* Die angegebene Port muss mit der in der *DistributeData::send\_data()* des weiteren Systems übereinstimmen und weiterhin nicht anderweitig belegt sein.

**Anm.:** Allgemein ist zu sagen, dass das WLAN-Netzwerk von derartigen Systemen nicht stark genug ist um datenintensive Aufgaben, wie Remote-Desktop oder Remote-Debug einwandfrei und stabil auszuführen. Deshalb hat sich in der Praxis gezeigt, dass solche Aufgaben am besten in einem weiteren Netzwerk über LAN stattfinden sollten. Beispielsweise wenn der Banana Pi und das Notebook jeweils über LAN an einen Router verbunden wird, sodass nun über dieses Netzwerk Remote-Desktop ausgeführt wird.

## Programmausführung des zentralen Backend

*Getestet auf: Raspberry Pi 4, Raspberry Pi 3b+*

Das Visual Studio Linux-Projekt „rm\_backend\_rpi4“ soll am besten über Visual Studio am Remotecomputer ausgeführt werden, um mehr Debug Informationen/Möglichkeiten zu erhalten. Eine Vorgehensweise wird von Microsoft beschrieben (Microsoft, 2020)[[2]](#footnote-2)

Sonst müsste einfach die erstellte .out Datei (rm\_backend\_rpi4.out) in einer Konsole ausgeführt werden.

Es wird zunächst nach einer Benutzereingabe gefragt. Mögliche Parameter sind „live“ für die Ausführung mit den Echtzeitdaten der momentan angeschlossenen Sensoren, oder eine \*.txt Datei, welche eine Abfolge von voraufgezeichneten Sensordaten enthält. Diese können etwa während einer live Ausführung aufgenommen werden. Dazu muss nur währenddessen „R“ eingegeben werden, und in einer weiteren Eingabe einen Namen für die zu erstellende Textdatei. Mit „S“ wird die Aufnahme gestoppt und bei dem Verzeichnis „/home/pi/Documents/\*.txt“ gespeichert.

Wird eine live Demo erwünscht versucht das Programm alle lokalen UART Verbindungen auszulesen und ggf. TCP/IP Verbindungen mit weiteren Systemen aufzubauen. Diese sind in der *main.cpp* aufgeführt. Sollte nur eine Verbindung fehlerhaft sein, wird das Programm unterbrechen.

Anschließend werden die TCP/IP Verbindungen mit den Unity Clients und dem RoboNova WLAN-Modul aufgebaut. Ebenso gilt hier: Sollte nur eine Verbindung fehlerhaft sein wird das Programm nicht fortsetzten. Daten werden nun an die Clients gesendet.

## Programmausführung weiterer Systeme

*Getestet auf: Banana Pi M2+*

Das Visual Studio Linux-Projekt „rm\_backend\_rpi3bplus2“ soll am besten über Visual Studio am Remotecomputer ausgeführt werden, um mehr Debug Informationen/Möglichkeiten zu erhalten. Eine Vorgehensweise wird von Microsoft beschrieben (Microsoft, 2020)2

Sonst müsste einfach die erstellte .out Datei (rm\_backend\_rpi3blus2.out) in einer Konsole ausgeführt werden.

Bei Ausführung versucht der Client sich mit dem Access Point zu verbinden und diesem gleich die ungefilterten Daten der an den UARTs angeschlossenen Sensoren zu schicken.

## Zeitlicher Ablauf der Ausführungen

Allgemein ist zu sagen, dass auch Abweichungen bei dem Starten von Diensten erfolgen können, doch diese Abfolge hat sich als Erfolgreichste erwiesen:

1. Programmstart des zentralen Backend mit „live“ oder \*.txt Datei
2. Ausführung der Unity Simulation
3. Verbinden des RoboNova WLAN-Modul
4. Starten weiterer Systeme

## Software des WLAN-Modul

Die Software zum Flashen des ESP8266 WLAN-Modul beinhaltet die festen *ssid* und *wpa\_passphrase* des Access Point. Ebenso sind dort die IP-Adresse und Port des zentralen Backend fest vergeben. Das Modul versucht sich nach der Stromversorgung mit dem Access Point zu verbinden und empfängt daraufhin die Daten und schickt diese gleich weiter an die angeschlossene UART Quelle. Bei einem Timeout würde das Modul versuchen sich alle 5 Minuten wieder neu zu verbinden, jedoch war das Resetten des Moduls durch die Stromversorgung meistens zielführender.

## Mögliche nachträgliche Änderungen

Die aktuelle Vorgehensweise kann dem Code entnommen werden. Hier werden noch mögliche Änderungen erwähnt:

1. Filteränderung

Aktuell wird ein einfacher Komplementärfilter verwendet. Falls ein genauerer Filter erwünscht ist müsste die Funktion *filter\_values()* in der Klasse *DistributeData* verändert bzw. ausgetauscht werden.

1. Shutdown der TCP/IP Verbindungen

Die Umsetzung der TCP/IP Verbindungen in C++ führt teils zu Komplikationen bei den gebundenen Ports, sodass ein abruptes Abbrechen des Programmes noch zu existierenden Binds bei den Ports führt, welche erst von dem Betriebssystem nach einer gewissen Zeit beendet werden. Dafür wurden Benutzereingaben eingerichtet, welche eine Verbindung sicher beendet. Diese wurden aber nicht getestet und weiterhin entsteht das Problem bei sonstigen Beendigungen.

Stundentabelle:

|  |  |
| --- | --- |
| **Aktivität** | **Stunden** |
| Wöchentliche Meetings | 35 |
| Gemeinsame Arbeitssprints | 50 |
| Einarbeitung Sockets | 10 |
| Aufbau der Komponenten | 5 |
|  |  |

1. <https://github.com/raspberrypi/documentation/blob/master/configuration/wireless/access-point-routed.md> [↑](#footnote-ref-1)
2. <https://docs.microsoft.com/de-de/cpp/linux/connect-to-your-remote-linux-computer?view=vs-2019> [↑](#footnote-ref-2)