

Projekt

Computer Vision mit Python, OpenCV,
MQTT, Raspberry PI
und ESP8266

von Steven Bruck

Januar 2021

https://github.com/faraway030/EDU_OpenCV-Project

1 Inhalt

1	PROJEKT	4
1.1	VORWORT	4
1.2	IDEE	4
1.3	KONZEPT	5
1.4	PROJEKTSOFTWARE	7
1.4.1	<i>install-server.sh</i>	7
1.4.2	<i>install-client.sh</i>	7
1.4.3	<i>server.py</i>	7
1.4.4	<i>client.py</i>	8
1.4.5	<i>mqtt.py</i>	8
1.4.6	<i>encode_images.py</i>	8
1.4.7	<i>esp_proxy.py</i>	8
1.5	VORAUSSETZUNGEN	10
1.5.1	<i>Arbeitssystem</i>	10
1.5.2	<i>Hardware</i>	10
1.6	ALLGEMEINE HINWEISE	10
1.6.1	<i>Verwendung von Kameras</i>	10
1.6.2	<i>Arbeiten unter Linux</i>	10
2	ARBEITSUMGEBUNG EINRICHTEN	11
2.1	KLONEN DES GIT-REPOSITORY	11
2.2	PYTHON-UMGEBUNG EINRICHTEN	11
2.3	PLATFORMIO INSTALLIEREN	12
3	VORBEREITEN DER HARDWARE	13
3.1	RASPBERRY PI SERVER	13
3.2	RASPBERRY PI ZERO MIT KAMERA	14
3.2.1	<i>Betriebssystem installieren</i>	14
3.2.2	<i>SSH und WLAN konfigurieren</i>	14
3.2.3	<i>Kamera anschließen</i>	15
4	INSTALLATION PI 4 ALS SERVER	15
4.1	VERBINDUNG HERSTELLEN	15
4.2	BASISKONFIGURATION	16
4.2.1	<i>Neuen Benutzer anlegen</i>	16
4.2.2	<i>Hostnamen setzen</i>	16
4.2.3	<i>Zeitzone setzen</i>	16
4.2.4	<i>System aktualisieren</i>	16
4.3	EINRICHTUNG ALS WIFI-AP	17
4.4	VORBEREITUNG FÜR DIE INSTALLATION DER LIBRARIES	18
4.4.1	<i>Swapfile erstellen</i>	18
4.4.2	<i>Memorysplit</i>	19
4.5	INSTALLATION DER LIBRARIES	19
4.6	INSTALLATION VON OPENCV	20
4.6.1	<i>Installation via Pip</i>	20
4.6.2	<i>Installation via apt</i>	20
4.6.3	<i>Selbst kompilieren (empfohlen)</i>	21
4.6.4	<i>Test</i>	22
4.7	SWAPFILE ENTFERNEN	22
4.8	INSTALLATION DES MQTT SERVERS	23

4.9	INSTALLATION DER PROJEKTSOFTWARE.....	24
4.10	TEST	24
5	INSTALLATION PI ZERO ALS CLIENT	25
5.1	BASISKONFIGURATION.....	25
5.2	KAMERA AKTIVIEREN.....	26
5.3	VORBEREITUNG FÜR DIE INSTALLATION DER LIBRARIES	27
5.4	INSTALLATION VON PYTHON3	29
5.5	INSTALLATION VON OPENCV	29
5.5.1	<i>Installation via Pip</i>	<i>29</i>
5.5.2	<i>Kompilieren auf dem Pi Zero</i>	<i>30</i>
5.5.3	<i>Cross-Compiling auf dem Arbeitssystem (empfohlen)</i>	<i>30</i>
5.5.4	<i>Test</i>	<i>30</i>
5.6	INSTALLATION DER RESTLICHEN PYTHON-PAKETE	31
5.7	MODIFIKATIONEN RÜCKGÄNGIG MACHEN	31
5.8	INSTALLATION DER PROJEKTSOFTWARE	31
6	ESP32-KAMERA	32
6.1	MODIFIKATION DER FIRMWARE	32
6.2	FLASHEN DER FIRMWARE	33
7	INSTALLATION DES „FACEREACT“-MODULS.....	34
7.1	MODIFIKATION DER FIRMWARE	34
7.2	FLASHEN DER FIRMWARE.....	34
7.3	INSTALLATION.....	34
8	DATASET ERSTELLEN	35
9	INBETRIEBNAHME	37
9.1	ERSTE TESTS IM VISUELLEN MODUS	37
9.2	BETRIEBSMODUS.....	44
10	FAZIT	44

1 Projekt

1.1 Vorwort

Zuerst einmal möchte ich mich herzlich für die Möglichkeit zu diesem Projekt und die zusätzliche Zeit bedanken. Da ich mich privat sehr viel mit dem Thema „Smart Home“ beschäftige, Raspberry Pi's als Server für dieses betreibe und auch eigene kleine Komponenten wie Bewegungsmelder, Sensoren etc. konstruiere und entwickle, habe ich mich sehr gefreut ein Projekt in diesem Bereich verwirklichen zu können.

Eines der Dinge, die mich bisher sehr gestört haben, ist die fehlende Intelligenz in solchen Systemen. Simples „Wenn A, dann B“ ist zwar in einem Smart Home ganz praktisch, verliert aber zumindest für mich schnell den Reiz. Daher hatte ich schon länger den Wunsch, mich in den Bereich KI, Deep-Learning und Computer Vision einzuarbeiten. Ich finde es faszinierend, wenn ein System in der Lage ist, intelligent Probleme zu lösen, autark zu agieren und mehr kann, als nur rein nach fest definierten Bedingungen zu handeln.

Da ich in keinem dieser Bereiche Vorwissen hatte, die Hardware-Ressourcen sehr begrenzt waren und es den zeitlichen Rahmen bei Weitem überstiegen hätte, mich ausreichend in Neuronale Netze und Deep-Learning einzuarbeiten, habe ich mich für das Thema „Computer Vision“ entschieden.

1.2 Idee

Realisiert wird eine Identifizierung von Personen anhand des Gesichts mittels dlib bzw. dem Python-Modul face_recogniton. Dazu wird zu Beginn aus einigen Fotos der zu identifizierenden Personen ein Dataset erstellt.

Anschließend senden eine oder mehrere Kameras ihr Bild per Videostream an einen Server. Dieser verarbeitet die eingehenden Frames dann mittels OpenCV, dlib und Viola-Jones-Algorithmus.

Auf besonderen Wunsch habe ich auch einen Microcontroller eingebunden, um das Zusammenspiel mehrerer Geräte zu verdeutlichen. Somit wird das Ergebnis nicht nur auf dem Bildschirm ausgegeben, sondern auch ein Steuersignal an einen ESP8266 geschickt, der bei einem bekannten Gesicht eine grüne LED und bei einem unbekannten Gesicht eine rote LED aufleuchten lässt. Um eine weitere Technologie einzubeziehen, habe ich mich für die Kommunikation mit dem Microcontroller für MQTT entschieden, da dieses in Smart Home Umgebungen ein stark etabliertes Protokoll ist und auch privat von mir genutzt wird.

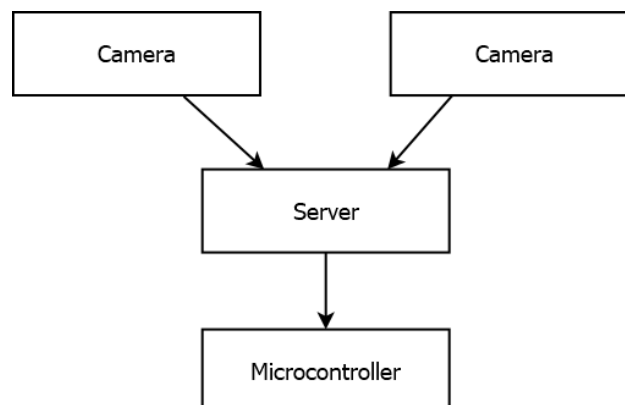


Abbildung 1.2.1

1.3 Konzept

Die Umsetzung wird bis auf die Firmware für ESP-Kamera und ESP8266 in Python erfolgen.

Den Kern des Projekts bildet der Raspberry Pi 4, welcher als WiFi-AP dient. Auch wenn ich sämtliche Sicherheitsfaktoren bei diesem Projekt außer Acht gelassen habe, war mir wichtig, dass das System Stand-Alone und ohne externe Abhängigkeiten funktioniert. Dieser betreibt außerdem den von mir programmierten Server, der die Videostreams empfängt, zusammen mit dem erstellten Dataset verarbeitet und entsprechende Ausgaben liefert. Dazu wird mosquitto als MQTT-Broker betrieben, der die Nachrichtenübermittlung zwischen Server und dem Microcontroller übernimmt.

Eine beliebige Anzahl an Kameras liefern also ihren Videostream an den Pi 4, dieser verarbeitet das Ganze und teilt dem Microcontroller via MQTT sein Ergebnis mit, welches von diesem dann mittels LED's sichtbar gemacht wird. Alle Clients verbinden sich direkt mit dem Pi 4, so dass von außen kein Zugriff auf die einzelnen Komponenten möglich ist.

Die folgenden Diagramme sollen den Aufbau verdeutlichen:

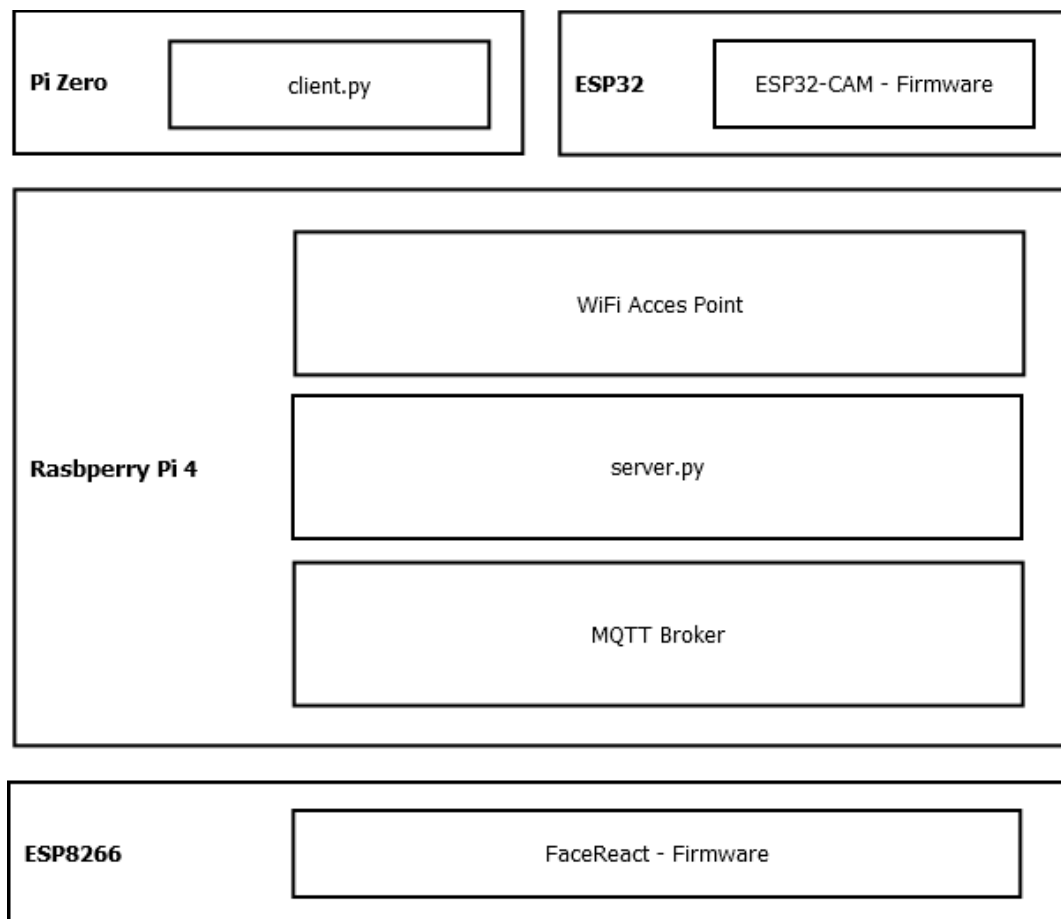


Abbildung 1.3.1

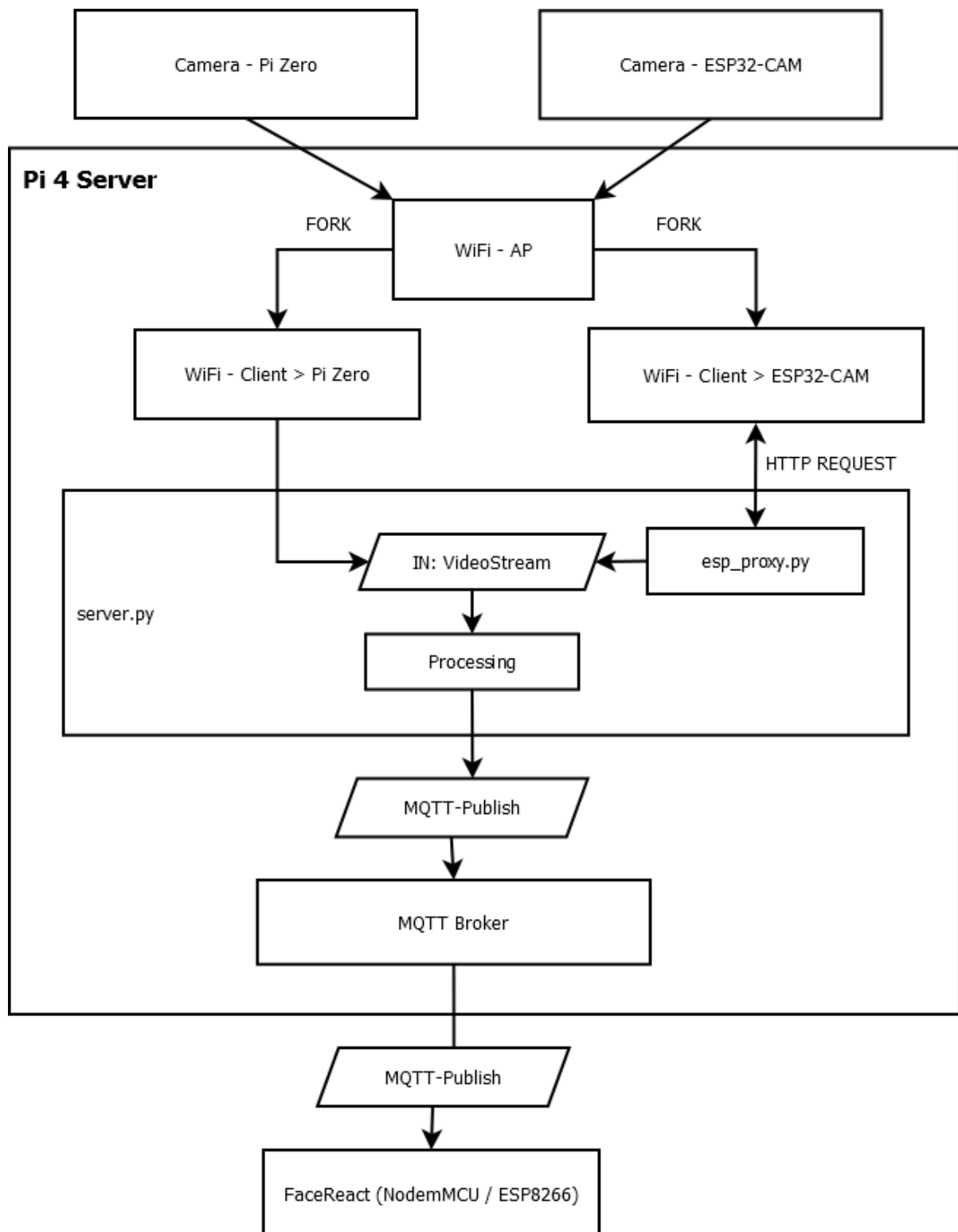


Abbildung 1.3.2

1.4 Projektsoftware

1.4.1 install-server.sh

- Bash-Installationsscript für die Server-Projektsoftware
- Lädt die entsprechenden Dateien von GitHub herunter und kopiert sie ins entsprechende Verzeichnis

1.4.2 install-client.sh

- Bash-Installationsscript für die Client-Projektsoftware
- Lädt die entsprechenden Dateien von GitHub herunter und kopiert sie ins entsprechende Verzeichnis

1.4.3 server.py

- Kern der Projektsoftware
- Empfängt Videostreams und verarbeitet diese
- Erkennt Personen anhand des Gesichts mittels zuvor erstelltem Dataset
- Sendet Steuerbefehle via MQTT an Microcontroller

```
usage: server.py [-h] [-i INTERFACE] [-e [NOESP]] [-v [VISUAL]] [-c
[COLUMNS]] [-t [TEST]]

optional arguments:
  -h, --help            show this help message and exit
  -i INTERFACE, --interface INTERFACE
                        Define WiFi-Interface the AP is running at
  -e [NOESP], --noesp [NOESP]
                        Deactivates EspProxy for supporting ESP32-Cams
  -v [VISUAL], --visual [VISUAL]
                        Enabled visual output
  -c [COLUMNS], --columns [COLUMNS]
                        Set max number of frames in a row (visual-mode
                        only)
  -t [TEST], --test [TEST]
                        Testrun of the server without any function
```

1.4.4 client.py

- Kern des Pi Zero
- Sendet Kamerastream zum Server

```
usage: client.py [-h] [-s SERVER]
```

optional arguments:

- h, --help show this help message and exit
- s SERVER, --server SERVER
IP-Address of the server to connect to.

1.4.5 mqtt.py

- Auslagerung der MQTT-Funktionen

1.4.6 encode_images.py

- Erstellt aus Fotos in einer definierten Ordnerstruktur eine Dataset für die spätere Gesichtserkennung

1.4.7 esp_proxy.py

Da ich dieses Projekt mit ESP32-CAM-Modulen begonnen habe und der Pi Zero mit Kameramodul erst später dazu kam, war ich vor folgendes Problem gestellt.

Es gab einiges an Beispiel-Firmware an denen ich mich orientieren konnte, jedoch haben diese ausschließlich einen Web-Server gestartet und das Bild lokal dorthin gestreamt. Eine Socket-Lösung konnte ich nicht finden und mich in Socket-Programmierung in C++ einzuarbeiten, habe ich zwar begonnen, hätte aber den zeitlichen Rahmen des Projekts überstiegen.

Deshalb habe ich diesen Proxy geschrieben, der im Server als Library eingebunden wird. Er startet ein UDP-Socket, das von der Firmware des Kameramoduls angesprochen wird. Anschließend greift der Proxy auf den lokalen Stream der Kamera zu und leitet diesen an der Server weiter.

Die Funktionsweise soll durch folgendes Diagramm erläutert werden:

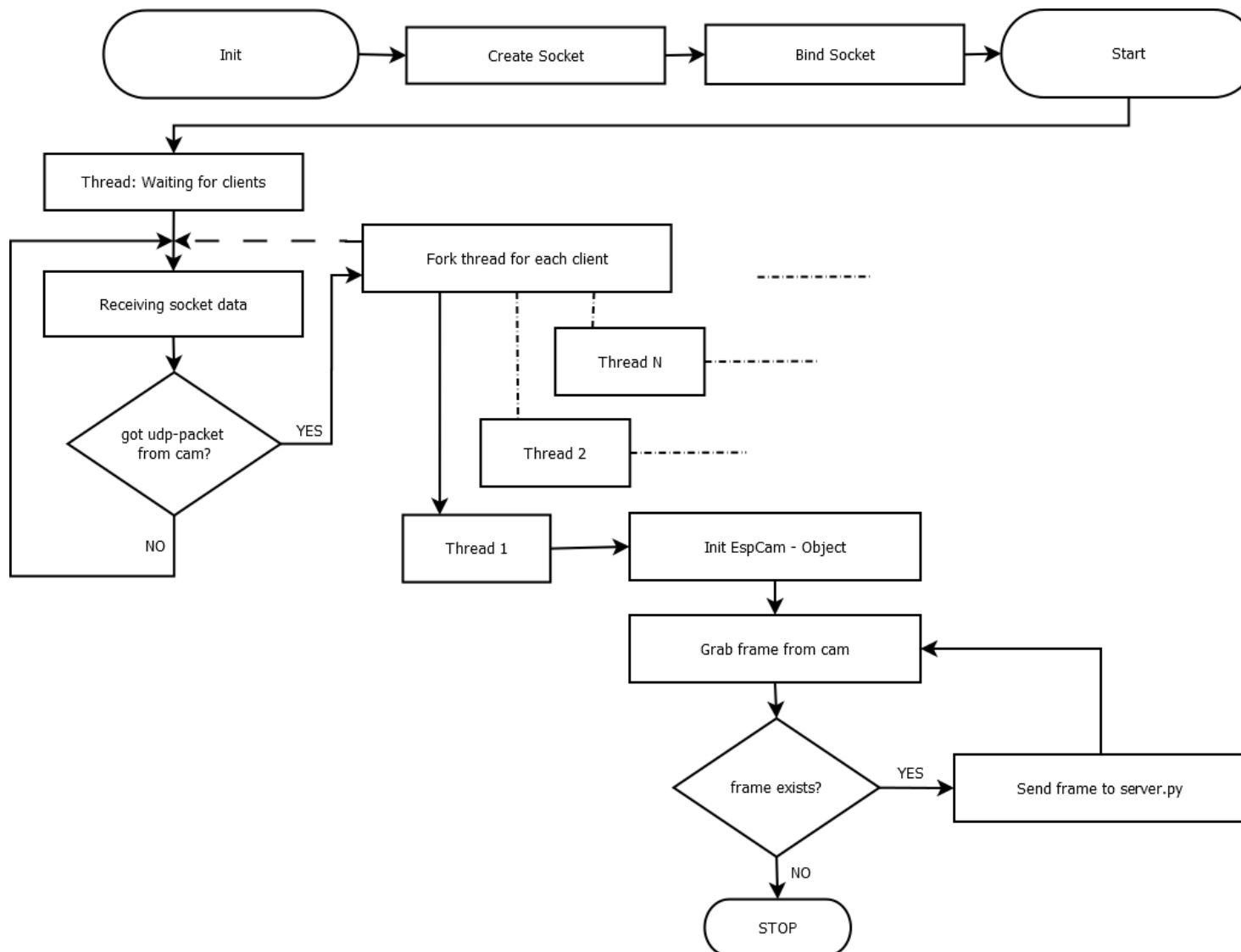


Abbildung 1.4.1

1.5 Voraussetzungen

1.5.1 Arbeitssystem

Mein Arbeitssystem:

Hardware:

- OS: Manjaro Linux
- CPU: Intel i7 10875H
- RAM: 16 GB DDR4
- GPU: nVidia RTX 2060

An Software wird vorausgesetzt:

- VS-Code mit PlatformIO
- Python3

1.5.2 Hardware

Als Hardware wird Folgendes verwendet:

1. Raspberry PI 4 mit 1 GB RAM
2. Raspberry PI Zero mit Kameramodul (OV5647 mit 170° Weitwinkel-Objektiv)
3. 1x ESP32-CAM Development Board inkl. OV2640 Kameramodul
4. 1x NodeMCU Development Board (ESP8266)
5. 3x 220 Ohm Widerstand
6. 3 LED's (grün und rot und blau)
7. LAN-Kabel
8. FTDI
9. 4x M-M Jumper-Kabel
10. 4x F-F Jumper-Kabel
11. Jumper Steckbrücke

1.6 Allgemeine Hinweise

1.6.1 Verwendung von Kameras

Die Software ist so ausgelegt, dass sie mit beliebig vielen Kameras arbeiten kann. Die Anzahl ist lediglich durch die technischen Ressourcen der Hardware limitiert. Weiterhin kann man auch nur den Pi Zero oder nur die ESP32-Kamera verwenden. Verbindet man seinen Computer mit dem AP oder den Pi 4 mit seinem LAN, kann man, Webcam und entsprechende Libraries vorausgesetzt, auch seinen Computer als Client nutzen und auf die Kameras verzichten.

1.6.2 Arbeiten unter Linux

1. Alle Befehle die im Folgenden unter Linux verwendet werden, werden als root ausgeführt. Ausnahmen werde ich anmerken.
2. Ich verzichte bewusst auf weitere Konfigurationen, die ich eigentlich machen würde, wie z.B. SSH per Publickey, UFW etc. und beschränke mich auf die Funktionalität des Projekts.

2 Arbeitsumgebung einrichten

2.1 Klonen des Git-Repository

Da VSCode bereits git integriert hat, können wir es uns hier einfach machen. Wir öffnen VSCode:

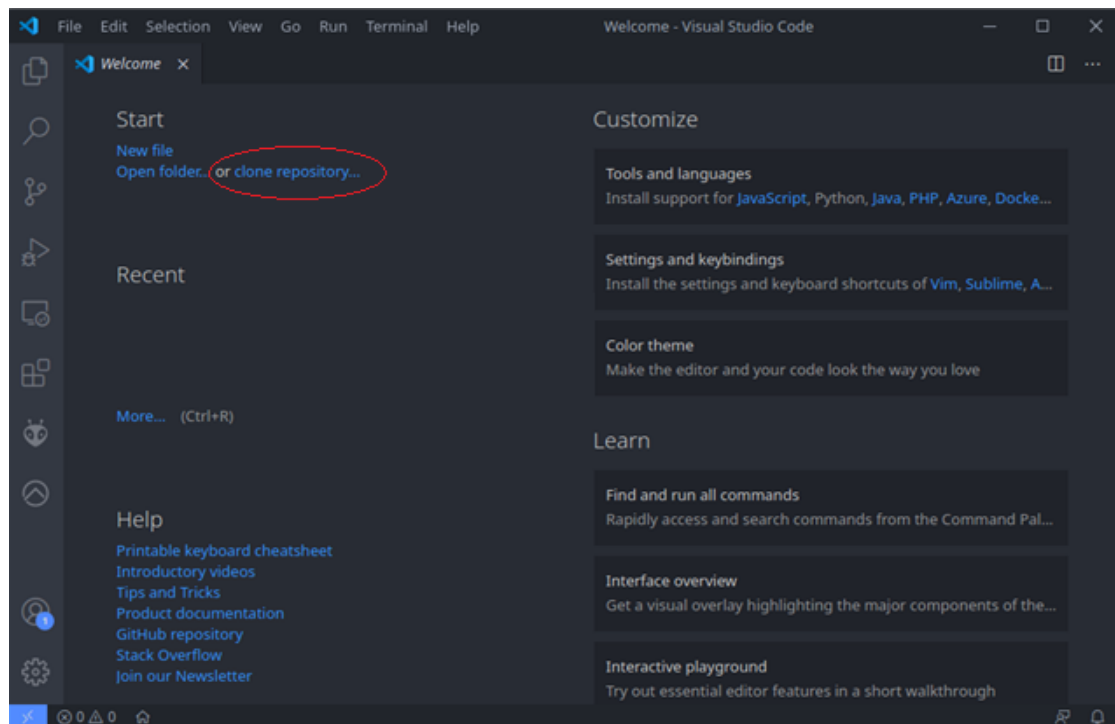


Abbildung 2.1.1

und wählen „clone repository...“ aus. In die sich öffnende Adresszeile geben wir https://github.com/faraway030/EDU_OpenCV-Project.git ein, wählen anschließend den Speicherort und bestätigen. Nun haben wir den Projektordner lokal gespeichert und können mit diesem arbeiten.

2.2 Python-Umgebung einrichten

Um die lokale Umgebung nicht zu verändern, richten wir für unser Python-Projekt eine virtuelle Umgebung ein. Dazu wechseln wir zunächst in unseren Projektordner und installieren das Paket „virtualenv“ für Python.

```
cd ~/Schreibtisch/EDU_OpenCV-Project
pip3 install virtualenv
```

Nun erstellen wir die virtuelle Umgebung und aktivieren diese:

```
python3 -m virtualenv env
source env/bin/activate
```

Was dann in etwa so aussehen sollte. (Natürlich sieht das mit einer Bash-Shell anders aus)



Abbildung 2.2.1

Alles was wir nun in Python installieren oder verändern, bleibt in dieser Umgebung. In dieser installieren wir nun alle Abhängigkeiten, die unser Projekt benötigt.

```
pip3 install -r requirements.txt
```

Um die Umgebung später wieder zu verlassen, geben wir folgenden Befehl ein.

```
deactivate
```

2.3 PlatformIO installieren

Zum Schluss installieren wir noch PlatformIO über die Extensions.

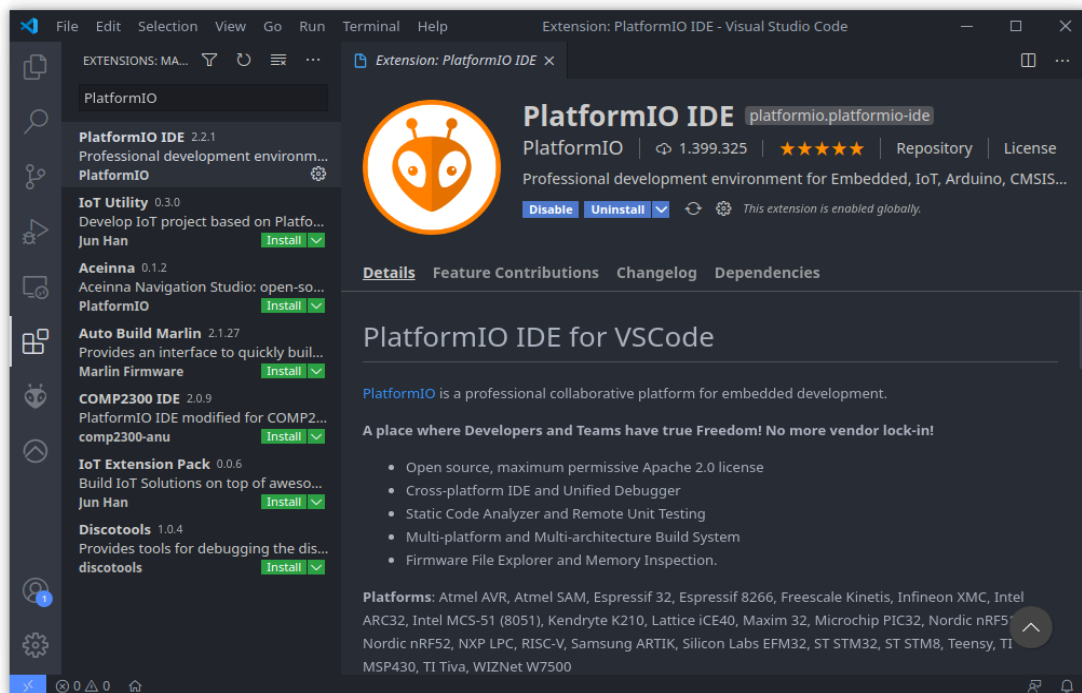


Abbildung 2.3.1

3 Vorbereiten der Hardware

3.1 Raspberry Pi Server

Den Server werden wir mit Ubuntu Server 20.10 32-Bit betreiben. Der Raspberry Pi 4 arbeitet zwar durchaus auch mit 64 Bit, da wir allerdings nur 1 GB RAM zur Verfügung haben und ansonsten später relevante Libraries selbst für eine 64-Bit-Architektur kompilieren müssten, reicht uns hier die 32 Bit Variante.0

Die einfachste Variante Raspberries mit einem Betriebssystem zu bestücken ist der Raspberry Pi Imager, den man hier (<https://www.raspberrypi.org/software/>) herunterladen kann. Dieser schreibt ein vorinstalliertes Abbild des Betriebssystems auf die Speicherkarte, so dass wir keinen Installationsprozess durchlaufen müssen und uns direkt an die Einrichtung machen können.

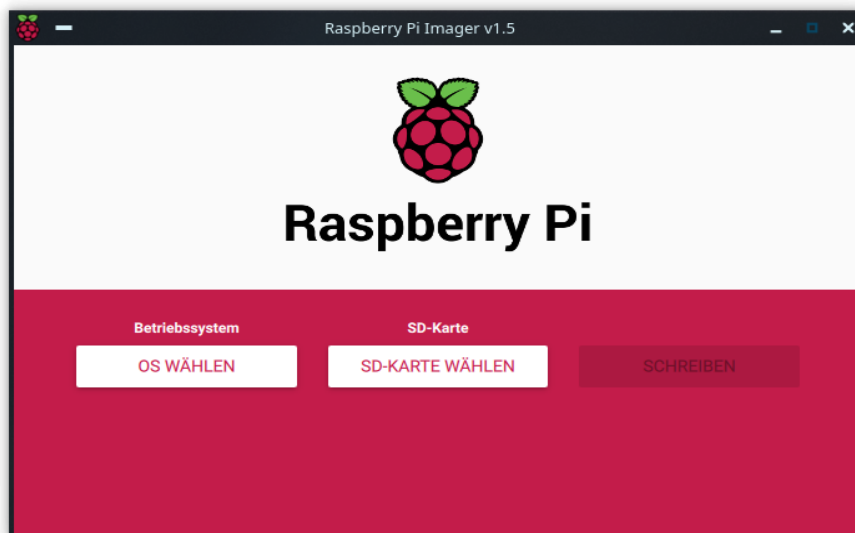


Abbildung 3.1.1

Wir öffnen den RPi Imager und wählen Ubuntu Server 20.10 32 Bit als Betriebssystem aus.



Abbildung 3.1.2

Anschließend wählen wir die Speicherkarte aus, die sich inzwischen im Kartenleser befinden sollte und klicken auf „Schreiben“. Der RPI Imager beginnt nun, das Abbild auf die Speicherkarte zu schreiben, verifiziert das Ganze im Anschluss nochmal und meldet dann, dass die Speicherkarte entnommen werden kann.

Da Ubuntu Server standardmäßig einen SSH-Server startet, müssen keine weiteren Modifikationen vorgenommen werden. Wir stecken die Speicherkarte nur noch in den Raspberry und das wars.

3.2 Raspberry PI Zero mit Kamera

3.2.1 Betriebssystem installieren

Hier gehen wir im Prinzip genau so vor, wie im Schritt zuvor. Jedoch wählen wir aufgrund der späteren Nutzung mit Kameramodul als Betriebssystem das „Raspberry PI OS Lite 32 Bit“ und verfahren genauso wie im Schritt zuvor.

3.2.2 SSH und WLAN konfigurieren

Bevor wir jedoch die Speicherkarte in den PI Zero stecken können, müssen wir noch einige Modifikationen vornehmen.

Mit der Speicherkarte im Kartenleser erstellen wir im Verzeichnis /mnt das Verzeichnis sdboot.

```
mkdir /mnt/sdboot
```

Wir vergewissern uns, welche Bezeichner die beiden Partitionen der SD-Karte haben:

```
fdisk -l
```

In meinem Fall ist die Boot-Partition /dev/sda1 und die System-Partition /dev/sda2. Die Boot-Partition mounten wir nun in den eben erstellten Ordner.

```
mount /dev/sda1 /mnt/sdboot
```

Das Raspberry PI OS berücksichtigt beim Systemstart die Existenz entsprechender Dateien in der Boot-Partition, womit sich z.B. SSH aktivieren oder die WLAN-Verbindung konfigurieren lässt.

Zur Aktivierung von SSH führen wir folgenden Befehl aus:

```
touch /mnt/sdboot/ssh
```

Zur Konfiguration der WLAN-Verbindung, müssen wir die Datei wpa_supplicant.conf erstellen:

```
vim /mnt/boot/wpa_supplicant.conf
```

und geben ihr folgenden Inhalt:

```
country=DE
ctrl_interface=DIR=/var/run/wpa_supplicant GROUP=netdev
update_config=1
network={
    ssid="FPA-OpenCV"
    psk="BoeckBoeck"
    key_mgmt=WPA-PSK
}
```

Zu den Zugangsdaten später mehr.

Wir lösen den mount wieder auf

```
umount /dev/sdboot
```

und können die Speicherkarte nun entnehmen und ebenfalls in den PI stecken.

3.2.3 Kamera anschließen

Anschließend installieren wir noch das Flachbandkabel der Kamera. Dazu ziehen wir am entsprechenden Anschluss des PI's die Befestigung vorsichtig zurück (von der Platine weg), schieben das Kabel mit den PIN's in Richtung Platine hinein und drücken anschließend die Befestigung wieder fest. Dasselbe machen wir an der Kamera und haben nun auch die Kamera installiert.

4 Installation Pi 4 als Server

4.1 Verbindung herstellen

Wir verbinden den PI 4 per LAN-Kabel mit unserem Netzwerk und schließen das erste Mal das Netzteil an. Sobald der Bootvorgang abgeschlossen ist, sollte der PI in unserem Router/DHCP-Server als Client auftauchen. Dort geben wir ihm im Optimalfall noch eine feste IP, damit wir ihn in Zukunft leicht wiederfinden.

Anschließend stellen wir von unserem Arbeitssystem eine SSH-Verbindung her.

```
ssh ubuntu@IP-VOM-PI
```

Das Standardpasswort von Ubuntu lautet ebenfalls „ubuntu“. Nach dem Login werden wir aufgefordert, unser Passwort zu ändern und sind dann erfolgreich mit per SSH verbunden.

4.2 Basiskonfiguration

4.2.1 Neuen Benutzer anlegen

Als ersten Schritt erstellen wir einen neuen Benutzer.

```
adduser steven
```

Wir vergeben unser Passwort, lassen alle weiteren Angaben leer und bestätigen alles mit Enter, bis unser Benutzer erstellt ist. Anschließend fügen wir unseren Benutzer noch der Gruppe „sudo“ hinzu, um den gleichnamigen Befehl nutzen und zum root-User wechseln zu können.

```
usermod -a -G sudo steven
```

Wenn alles bis hierhin funktioniert hat, verlassen wir die SSH-Session und versuchen uns mit unserem neuen Benutzer anzumelden.

Wenn auch dies funktioniert hat, löschen wir den Standardbenutzer ubuntu.

```
userdel -r ubuntu
```

4.2.2 Hostnamen setzen

Wir setzen den Hostnamen für den PI. Ich habe mich für FPA-OpenCV entschieden.

```
hostnamectl set-hostname ,FPA-OpenCV'
```

Nicht wundern, der neue Hostname wird erst beim nächsten Login in der Shell angezeigt.

4.2.3 Zeitzone setzen

Damit die Systemzeit auch mit unserer Zeit übereinstimmt, legen wir noch die Zeitzone fest.

```
timedatectl set-timezone ,Europe/Berlin'
```

4.2.4 System aktualisieren

Sollte bekannt sein:

```
apt update && apt dist-upgrade -y && reboot
```

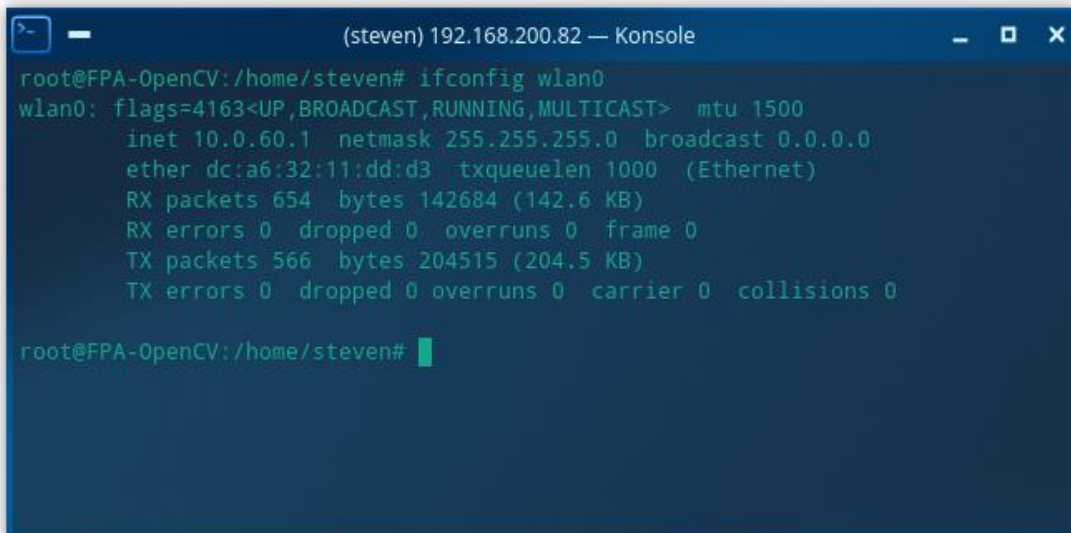

4.3 Einrichtung als WiFi-AP

Eine einfache, schnelle und für unseren Zweck ausreichende Art den Raspberry als WiFi Access Point einzurichten, ist „wifi-ap“ aus dem Snapstore.

```
apt install net-tools
snap install wifi-ap
wifi-ap.config set disabled=false
wifi-ap.config set wifi.security=wpa2
wifi-ap.config set wifi.security-passphrase="BoeckBoeck"
wifi-ap.config set wifi.ssid="FPA-OpenCV"
wifi-ap.config set share.disabled=false
snap restart wifi-ap
```

Überprüfen:

```
ifconfig wlan0
```



```
(steven) 192.168.200.82 — Konsole
root@FPA-OpenCV:/home/steven# ifconfig wlan0
wlan0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
    inet 10.0.60.1  netmask 255.255.255.0  broadcast 0.0.0.0
    ether dc:a6:32:11:dd:d3  txqueuelen 1000  (Ethernet)
    RX packets 654  bytes 142684 (142.6 KB)
    RX errors 0  dropped 0  overruns 0  frame 0
    TX packets 566  bytes 204515 (204.5 KB)
    TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

root@FPA-OpenCV:/home/steven#
```

Abbildung 4.3.1

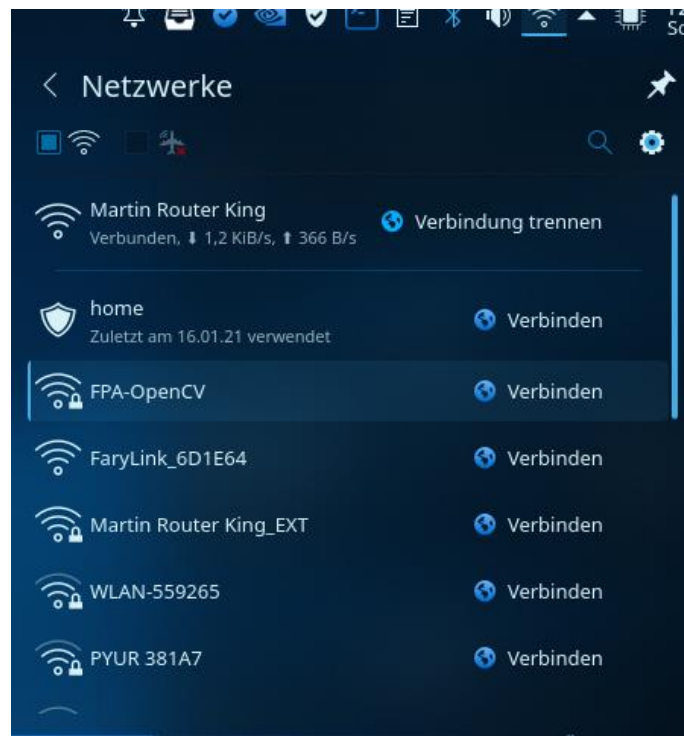


Abbildung 4.3.2

Und wir haben unseren Raspberry als Access Point eingerichtet und können uns per WLAN mit ihm verbinden. Die IP notieren wir uns für später. In diesem Fall ist es 10.0.60.1.

4.4 Vorbereitung für die Installation der Libraries

Aufgrund der geringen Ressourcen des Raspberries und dem Umfang der benötigten Software die wir installieren werden, müssen wir ein paar Veränderungen vornehmen, da es ansonsten zu Speicherproblemen kommt.

4.4.1 Swapfile erstellen

Mit folgenden Befehlen erstellen einen 2GB großen Swapfile und erstellen einen Eintrag in der fstab, um diesen beim Systemstart zu mounten.

```
fallocate -l 2G /swapfile
chmod 600 /swapfile
mkswap /swapfile
swapon /swapfile
echo „/swapfile swap swap defaults 0 0“ >> /etc/fstab
```

Nun überprüfen wir das Ganze noch:

```
swapon -show
free -h
```

Und sollten dann dieses Ergebnis haben:



```

(steven) 192.168.200.82 — Konsole
root@FPA-OpenCV:/home/steven# swapon --show
NAME      TYPE  SIZE USED PRIO
/swapfile file   2G   0B  -2
root@FPA-OpenCV:/home/steven# free -h
               total        used        free      shared  buff/cache   availabl
Mem:           852Mi        158Mi        80Mi        4.0Mi        613Mi        669M
Swap:          2.0Gi          0B        2.0Gi
root@FPA-OpenCV:/home/steven#

```

Abbildung 4.4.1

4.4.2 Memorysplit

Der Memorysplit gibt an, wieviel RAM der GPU zur Verfügung gestellt werden. Da wir den Server ohne GUI betreiben und den RAM anderweitig brauchen, reduzieren wir den Speicher für die GPU auf ein Minimum.

```
echo „gpu_mem=16“ >> /boot/firmware/config.txt
```

Anschließend starten wir den Raspberry neu.

4.5 Installation der Libraries

Zuerst installieren wir alle für die Installation benötigten Pakete:

```
apt install python3-pip build-essential cmake pkg-config
```

Nun laden wir aus dem Git-Repository die requirements.txt herunter, die alle Python-Abhängigkeiten beinhaltet.

```
curl -sSL https://raw.githubusercontent.com/faraway030/EDU_OpenCV-Project/master/requirements.txt > requirements.txt
```

Bevor wir fortfahren, müssen wir jedoch eine wichtige Änderung vornehmen. Da die Installation von OpenCV mittels Pip auf dem Raspberry sehr problematisch ist, entfernen wir aus der requirements.txt die gesamte Zeile von opencv. Anschließend starten wir die Installation mit folgendem Befehl:

```
python3 -m pip install --upgrade pip
python3 -m pip install -r requirements.txt
```

Die Installation wird ca. eine Stunde dauern, also ist dies ein guter Zeitpunkt für eine Pause.

4.6 Installation von OpenCV

Es gibt 3 Möglichkeiten OpenCV zu installieren, deshalb möchte ich zur Vollständigkeit kurz auf alle 3 Varianten eingehen.

4.6.1 Installation via Pip

Die Installation via Pip ist die Schnellste, jedoch auch die Problematischste, da es hier häufig zu diversen Fehlern kommt.

```
apt install libaom0 libatk-bridge2.0-0 libatk1.0-0 libatlas3-base
libatspi2.0-0 libavcodec58 libavformat58 libavutil56 libbluray2 libcairo-
gobject2 libcairo2 libchromaprint1 libcodec2-0.8.1 libcroc03 libdatrie1
libdrm2 libepoxy0 libfontconfig1 libgdk-pixbuf2.0-0 libgfortran5 libgme0
libgraphite2-3 libgsm1 libgtk-3-0 libharfbuzz0b libilmbase23 libjbig0
libmp3lame0 libmpeg123-0 libogg0 libopenexr23 libopenjp2-7 libopenmpt0
libopus0 libpango-1.0-0 libpangocairo-1.0-0 libpangoft2-1.0-0 libpixman-1-
0 librsvg2-2 libshine3 libsnappy1v5 libsoxr0 libspeex1 libssh-gcrypt-4
libswresample3 libswscale5 libthai0 libtheora0 libtiff5 libtwolame0 libva-
drm2 libva-x11-2 libva2 libvdpau1 libvorbis0a libvorbisenc2 libvorbisfile3
libvpx5 libwavpack1 libwayland-client0 libwayland-cursor0 libwayland-egl1
libwebp6 libwebpmux3 libx264-155 libx265-165 libxcb-render0 libxcb-shm0
libxcomposite1 libxcursor1 libxdamage1 libxfixes3 libxi6 libxinerama1
libxkbcommon0 libxrandr2 libxrender1 libxvidcore4 libzvb10
```

```
python3 -m pip install opencv-python
```

4.6.2 Installation via apt

```
apt install python3-opencv
```

apt sollte die Abhängigkeiten selbst erfassen und mitinstallieren.

4.6.3 Selbst kompilieren (empfohlen)

Der langsamste, aber beste Installationsweg ist es, OpenCV auf seinem System selbst zu kompilieren. Dies bietet den Vorteil, dass man je nach Bedarf Module einbinden oder weglassen kann und eine performante Installation erhält, da hier z.B. auch Optimierungsoptionen für den ARM-Prozessor wie NEON oder VFPV3 berücksichtigt werden.

Zuerst installieren wir die Abhängigkeiten:

```
apt install libjpeg-dev libtiff5-dev libjasper-dev libpng-dev install
libavcodec-dev libavformat-dev libswscale-dev libv4l-dev libxvidcore-dev
libx264-dev libfontconfig1-dev libcairo2-dev libgdk-pixbuf2.0-dev
libpango1.0-dev libgtk2.0-dev libgtk-3-dev libatlas-base-dev gfortran
libhdf5-dev libhdf5-serial-dev libhdf5-103 libqtgui4 libqtwebkit4 libqt4-
test python3-pyqt5 python3-dev

python3 -m pip install numpy
```

#Nun wechseln wir in unser Home-Verzeichnis und führen folgende Befehle aus:

```
git clone https://github.com/opencv/opencv.git
git clone https://github.com/opencv/opencv_contrib.git
```

Damit haben wir nun den aktuellen Sourcecode von OpenCV aus dem offiziellen Git-Repo geklont und können diesen kompilieren. Dazu erstellen wir im gerade heruntergeladenem Order „opencv“ den Ordner „build“ und wechseln in diesen.

```
mkdir ~/opencv/build && cd ~/opencv/build
```

Jetzt erstellen wir mit CMake noch die Konfiguration für den Kompiliervorgang:

```
cmake -D CMAKE_BUILD_TYPE=RELEASE \
-D CMAKE_INSTALL_PREFIX=/usr/local \
-D OPENCV_EXTRA_MODULES_PATH=~/opencv_contrib/modules \
-D ENABLE_NEON=ON \
-D ENABLE_VFPV3=ON \
-D BUILD_TESTS=OFF \
-D INSTALL_PYTHON_EXAMPLES=OFF \
-D OPENCV_ENABLE_NONFREE=ON \
-D CMAKE_SHARED_LINKER_FLAGS=-latomic \
-D BUILD_EXAMPLES=OFF ..
```

Wir warten einen Moment,0 starten dann mit folgendem Befehl den Kompiliervorgang:

```
make -j4
```

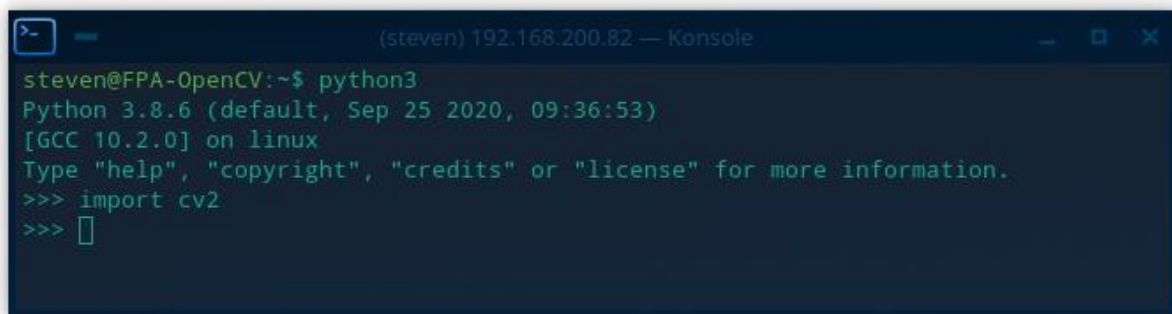
Die „4“ gibt an, wie viele CPU-Kerne make für den Kompiliervorgang verwendet. Somit nutzen wir alle 4 Kerne die der Pi zur Verfügung stellt. Das Ganze wird jetzt nochmal ca. eine Stunde dauern.

Im Anschluss installieren wir das Ganze noch:

```
make install  
ldconfig
```

4.6.4 Test

Wir überprüfen alles, in dem wir die Python3-Konsole aufrufen und OpenCV importieren.



```
(steven) 192.168.200.82 — Konsole  
steven@FPA-OpenCV:~$ python3  
Python 3.8.6 (default, Sep 25 2020, 09:36:53)  
[GCC 10.2.0] on linux  
Type "help", "copyright", "credits" or "license" for more information.  
>>> import cv2  
>>> 
```

Abbildung 4.6.1

Fertig! OpenCV ist nun erfolgreich installiert!

4.7 Swapfile entfernen

Da der große Swap zwar sehr schön ist, Speicherkarten aber nicht für so viele Schreibvorgänge ausgelegt sind und ihre Lebenserwartung dadurch deutlich reduziert wird, löschen wir unseren Swapfile wieder.

```
swapoff /swapfile  
rm /swapfile
```

und entfernen dann noch unseren Eintrag aus der Datei /etc/fstab.

4.8 Installation des MQTT Servers

Um mittels MQTT mit dem Microcontroller kommunizieren zu können, müssen wir noch einen MQTT-Broker installieren.

```
apt install mosquitto
```

Anschließend erstellen wir eine Konfigurationsdatei

```
vim /etc/mosquitto/conf.d/mosquitto.conf
```

und geben ihr folgenden Inhalt:

```
listener 1883
allow_anonymous true
```

Anmerkung: Dies ist eine absolute Minimalkonfiguration, die **nicht** in einer Produktivumgebung verwendet werden sollte, jedoch erfüllt sie für dieses Projekt ihren Zweck.

Wir überprüfen das Ganze wieder:

```
systemctl restart mosquitto.service
systemctl status mosquitto.service
```

Und sollten dann einen funktionierenden MQTT-Broker haben.

```
(steven) 192.168.200.82 — Konsole
● mosquitto.service - Mosquitto MQTT Broker
   Loaded: loaded (/lib/systemd/system/mosquitto.service; enabled; vendor preset: enabled)
   Active: active (running) since Sat 2021-01-30 12:33:42 CET; 1min 5s ago
     Docs: man:mosquitto.conf(5)
           man:mosquitto(8)
   Process: 2088 ExecStartPre=/bin/mkdir -m 740 -p /var/log/mosquitto (code=exited, status=0/SUCCESS)
   Process: 2089 ExecStartPre=/bin/chown mosquitto: /var/log/mosquitto (code=exited, status=0/SUCCESS)
   Main PID: 2090 (mosquitto)
      Tasks: 1 (limit: 1824)
   CGroup: /system.slice/mosquitto.service
           └─2090 /usr/sbin/mosquitto -c /etc/mosquitto/mosquitto.conf

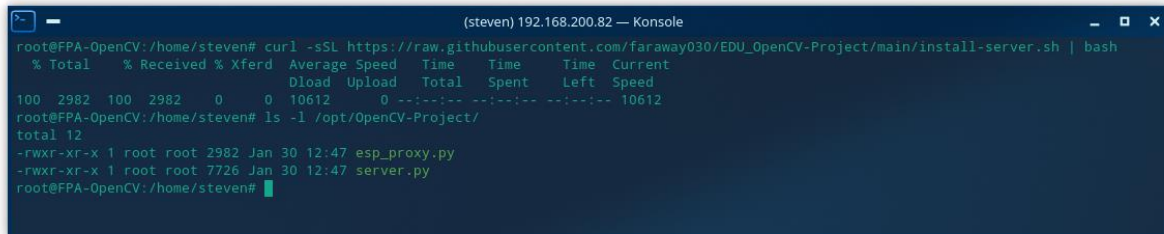
Jan 30 12:33:42 FPA-OpenCV systemd[1]: Starting Mosquitto MQTT Broker...
Jan 30 12:33:42 FPA-OpenCV mosquitto[2090]: 1612006422: Loading config file /etc/mosquitto/conf.d/mosquitto.conf
Jan 30 12:33:42 FPA-OpenCV systemd[1]: Started Mosquitto MQTT Broker.
~
lines 1-15/15 (END)
```

Abbildung 4.8.1

4.9 Installation der Projektsoftware

Als letzten Schritt installieren wir noch die Projektsoftware für den Server:

```
curl -sSL https://raw.githubusercontent.com/faraway030/EDU_OpenCV-Project/master/install-server.sh | bash
```



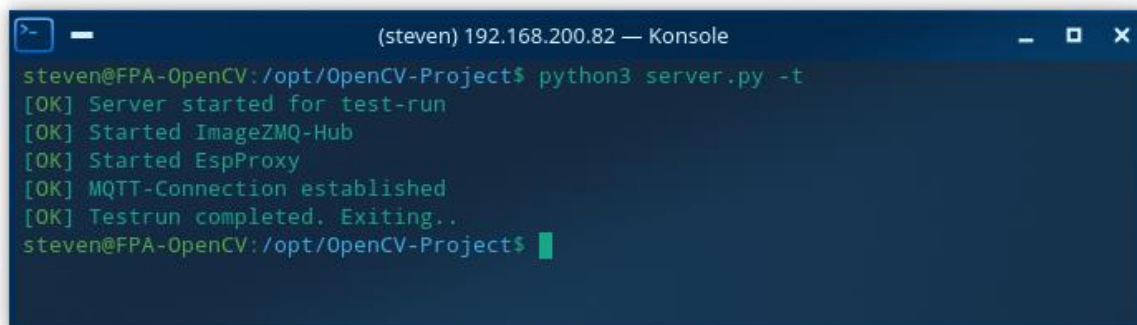
```
(steven) 192.168.200.82 — Konsole
root@FPA-OpenCV:/home/steven# curl -sSL https://raw.githubusercontent.com/faraway030/EDU_OpenCV-Project/main/install-server.sh | bash
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
           % Dload  % Upload   Total   Spent    Left    Speed
100 2982    100 2982    0     0  10612      0 --:--:-- --:--:-- --:--:-- 10612
root@FPA-OpenCV:/home/steven# ls -l /opt/OpenCV-Project/
total 12
-rwxr-xr-x 1 root root 2982 Jan 30 12:47 esp_proxy.py
-rwxr-xr-x 1 root root 7726 Jan 30 12:47 server.py
root@FPA-OpenCV:/home/steven#
```

Abbildung 4.9.1

4.10 Test

Zum Schluss testen wir das Ganze in dem wir in den Ordner /opt/OpenCV-Project wechseln und den Server im Testmodus starten. Wenn dies ohne Fehlermeldungen durchläuft, sind wir mit dem Server fertig.

```
python3 server.py -t
```



```
(steven) 192.168.200.82 — Konsole
steven@FPA-OpenCV:/opt/OpenCV-Project$ python3 server.py -t
[OK] Server started for test-run
[OK] Started ImageZMQ-Hub
[OK] Started EspProxy
[OK] MQTT-Connection established
[OK] Testrun completed. Exiting..
steven@FPA-OpenCV:/opt/OpenCV-Project$
```

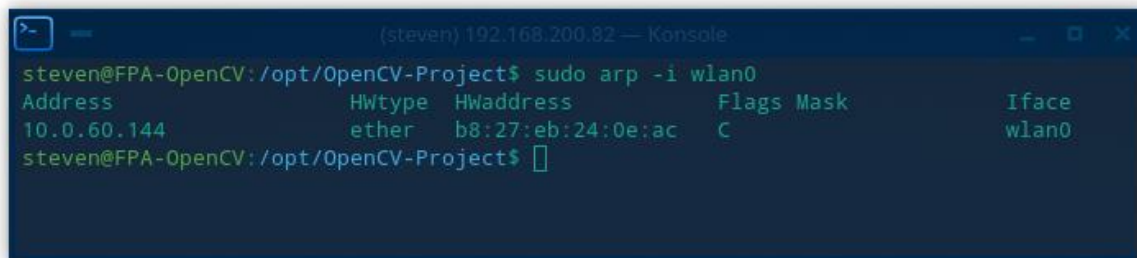
Abbildung 4.10.1

5 Installation Pi Zero als Client

5.1 Basiskonfiguration

Da der Pi Zero keinen LAN-Port hat und nur mit unserem Access Point verbunden ist, müssen wir etwas tricksen. Wir bleiben in der SSH-Session des Pi 4 und verbinden uns von hier per SSH weiter zum Pi Zero.

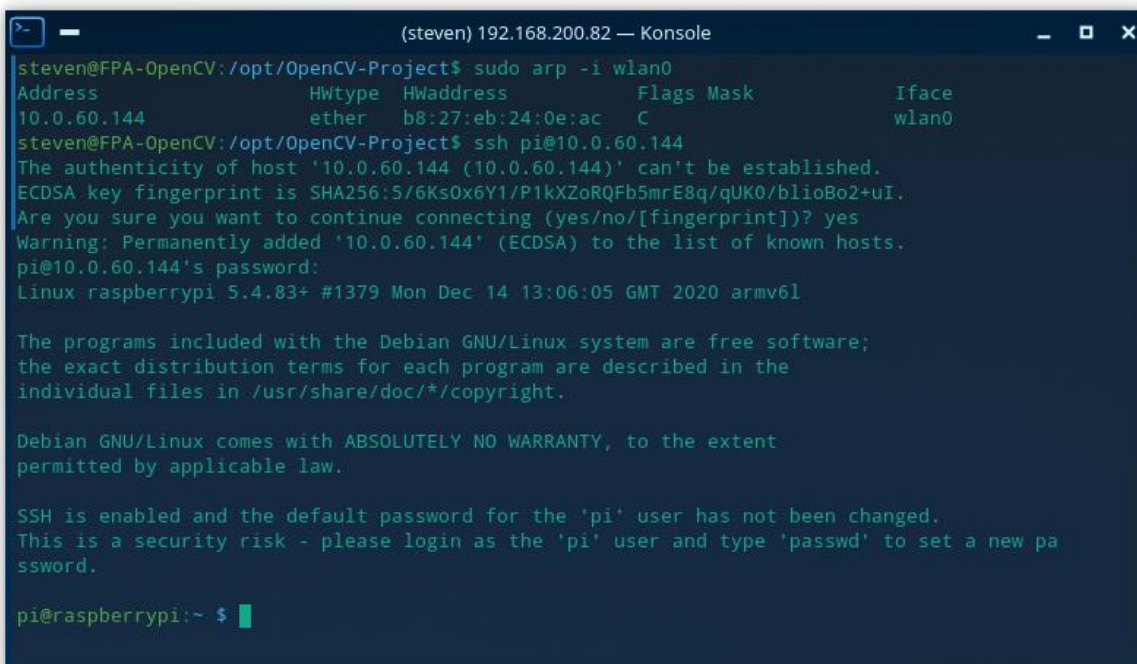
Da wir hierzu die IP benötigen, lassen wir uns die Clients des AP anzeigen



```
(steven) 192.168.200.82 — Konsole
steven@FPA-OpenCV:/opt/OpenCV-Project$ sudo arp -i wlan0
Address            HWtype  HWaddress      Flags Mask    Iface
10.0.60.144        ether   b8:27:eb:24:0e:ac  C           wlan0
steven@FPA-OpenCV:/opt/OpenCV-Project$
```

Abbildung 5.1.1

und können uns dann verbinden.



```
(steven) 192.168.200.82 — Konsole
steven@FPA-OpenCV:/opt/OpenCV-Project$ sudo arp -i wlan0
Address            HWtype  HWaddress      Flags Mask    Iface
10.0.60.144        ether   b8:27:eb:24:0e:ac  C           wlan0
steven@FPA-OpenCV:/opt/OpenCV-Project$ ssh pi@10.0.60.144
The authenticity of host '10.0.60.144 (10.0.60.144)' can't be established.
ECDSA key fingerprint is SHA256:5/6Ks0x6Y1/P1kXZoRQFb5mrE8q/qUK0/blioBo2+uI.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '10.0.60.144' (ECDSA) to the list of known hosts.
pi@10.0.60.144's password:
Linux raspberrypi 5.4.83+ #1379 Mon Dec 14 13:06:05 GMT 2020 armv6l

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.

SSH is enabled and the default password for the 'pi' user has not been changed.
This is a security risk - please login as the 'pi' user and type 'passwd' to set a new password.

pi@raspberrypi:~$
```

Abbildung 5.1.2

Das Standardpasswort lautet in dem Fall „raspberrry“.

Nun wiederholen wir die Schritte 4.2.1 – 4.2.4. Außer dem Hostnamen sind diese identisch. Ich habe den Pi Zero „PiCAM“ genannt.

5.2 Kamera aktivieren

Wir öffnen das Configuration Tool

```
raspi-config
```

und gehen dann folgende Schritte:

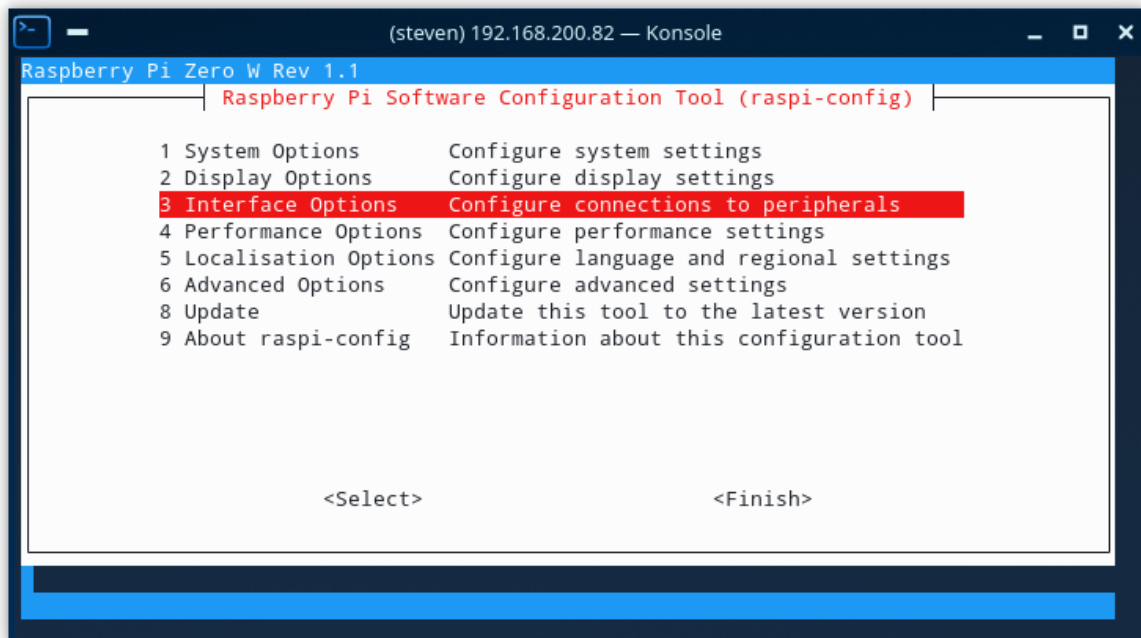


Abbildung 5.2.1

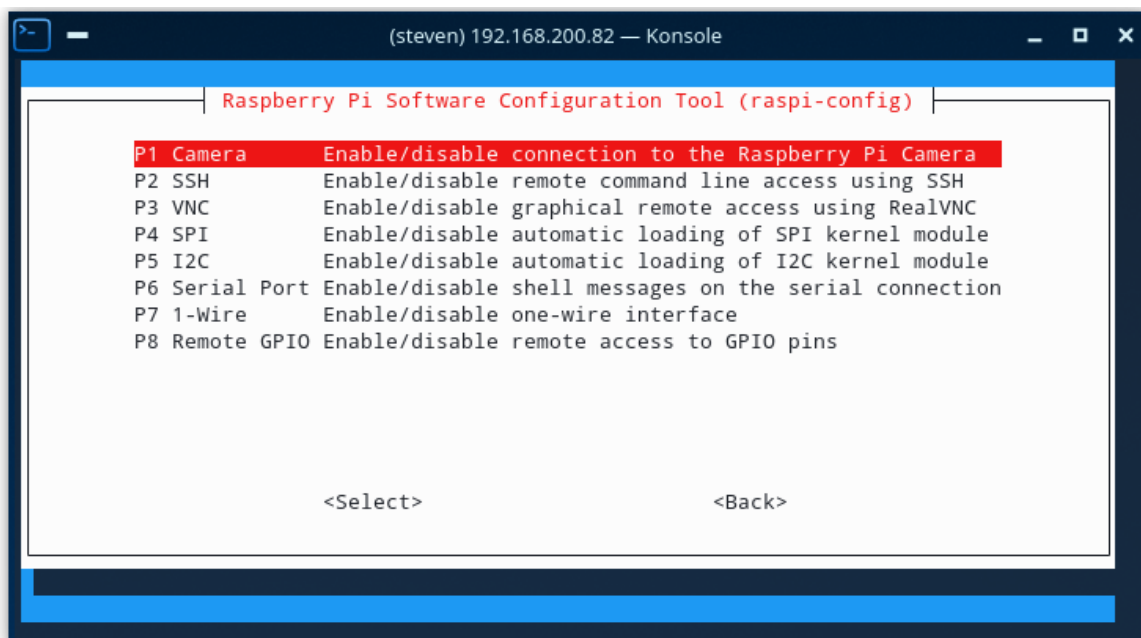


Abbildung 5.2.2

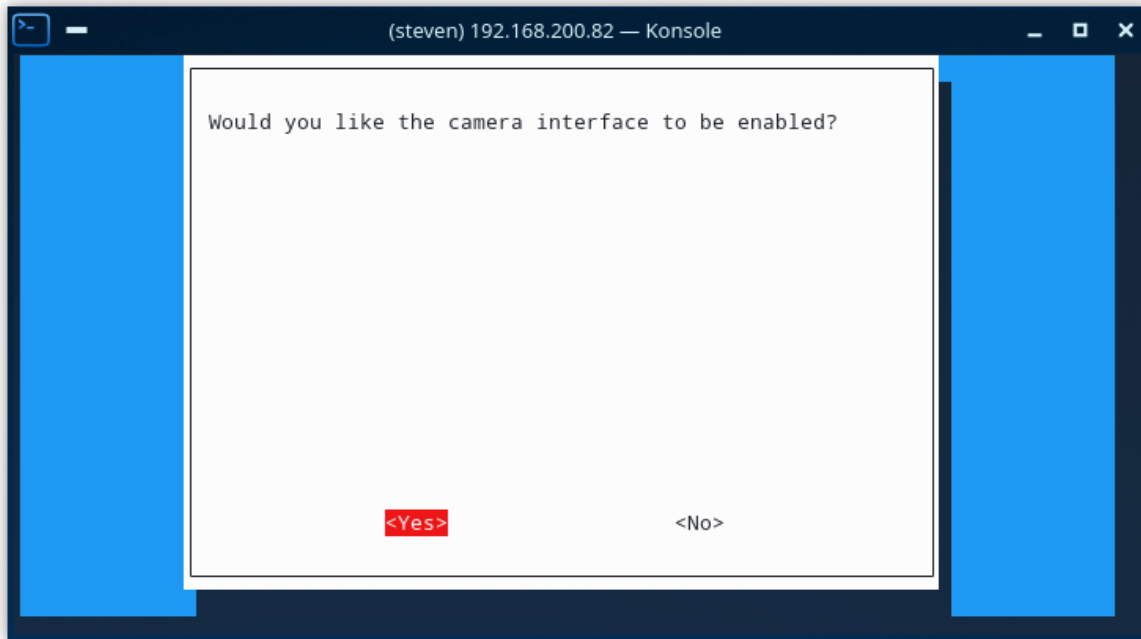


Abbildung 5.2.3

5.3 Vorbereitung für die Installation der Libraries

Da wir schon im Configuration Tool sind, können wir es uns hier einfach machen und mit folgenden Schritten den Memorysplit verändern:

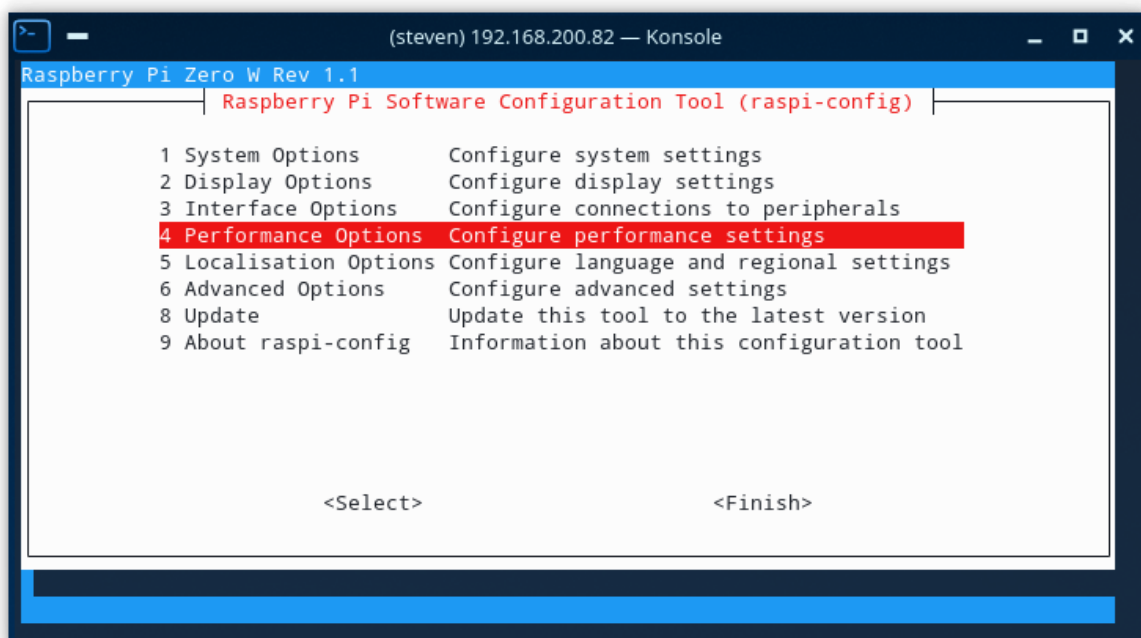


Abbildung 5.3.1

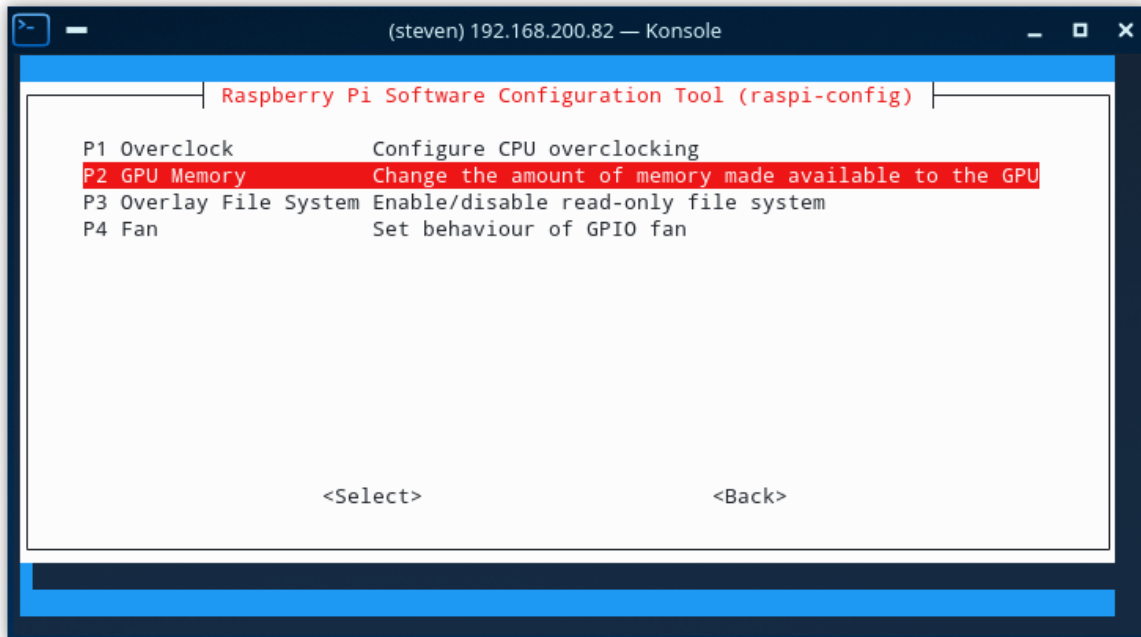


Abbildung 5.3.2

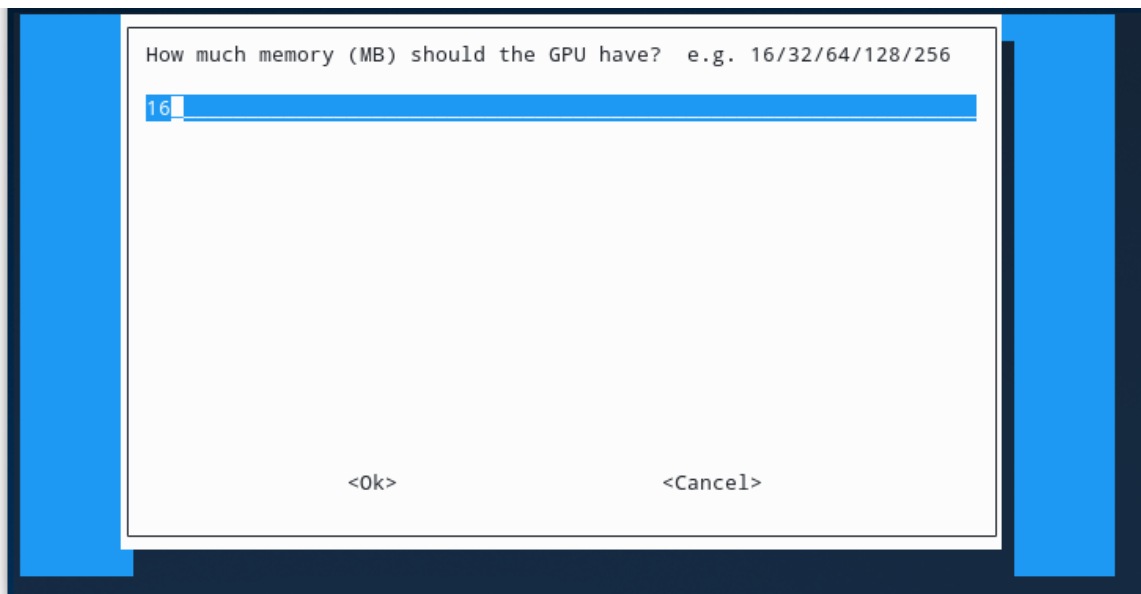


Abbildung 5.3.3

Wir bestätigen alles und verlassen über <Finish> das Configuration Tool. Die Frage ob wir neu starten wollen, verneinen wir aber vorerst.

ACHTUNG: Wenn zur Installation von OpenCV Punkt 4.5.3 verwendet, kann die Änderung des Swap übersprungen werden.

Nun ändern wir wieder die Größe des Swapfiles. Auch das ist unter Raspberry OS etwas einfacher. Wir öffnen die Datei `/etc/dphys-swapfile` und ändern den Wert von `CONF_SWAPSIZE` folgendermaßen:

```
CONF_SWAPSIZE=2048
```

Speichern das Ganze und nun machen wir einen reboot.

5.4 Installation von Python3

Wir installieren folgende Pakete:

```
apt install python3 python3-pip
```

Anschließend machen wir ein Update von Pip:

```
python3 -m pip install --upgrade pip
```

Und installieren gleich noch numpy mit, da wir dieses für OpenCV benötigen.

```
python3 -m pip install numpy
```

5.5 Installation von OpenCV

Auch hier gibt es wieder 3 Varianten. Um Ressourcen zu sparen und weil OpenCV hier nur zur Übertragung zum Server genutzt wird, installieren wir die headless-Variante.

5.5.1 Installation via Pip

Wenn sie funktioniert, ist sie wieder die schnellste Art der Installation.

```
apt install libaom0 libatlas3-base libavcodec58 libavformat58 libavutil56
libbluray2 libcairo2 libchromaprint1 libcodec2-0.8.1 libcroco3 libdatrie1
libdrm2 libfontconfig1 libgdk-pixbuf2.0-0 libgfortran5 libgme0
libgraphite2-3 libgsm1 libharfbuzz0b libilmbase23 libjbig0 libmp3lame0
libmpeg123-0 libogg0 libopenexr23 libopenjp2-7 libopenmpt0 libopus0
libpango-1.0-0 libpangocairo-1.0-0 libpangoft2-1.0-0 libpixman-1-0
librsvg2-2 libshine3 libsnappy1v5 libsoxr0 libspeex1 libssh-gcrypt-4
libswresample3 libswscale5 libthai0 libtheora0 libtiff5 libtwolame0 libva-
drm2 libva-x11-2 libva2 libvdpau1 libvorbis0a libvorbisenc2 libvorbisfile3
libvp8x5 libwavpack1 libwebp6 libwebpmux3 libx264-155 libx265-165 libxcb-
render0 libxcb-shm0 libxfixes3 libxrender1 libxvidcore4 libzvb10

python3 -m pip install opencv-python-headless
```

5.5.2 Kompilieren auf dem Pi Zero

OpenCV lässt sich auf dem Pi Zero genau so kompilieren, wie in Kapitel 4.6.3 beschrieben.

Hier sind nur folgende Änderungen im CMake-Befehl zu machen:

```
cmake -D CMAKE_BUILD_TYPE=RELEASE \
-D CMAKE_INSTALL_PREFIX=/usr/local \
-D OPENCV_EXTRA_MODULES_PATH=~/.opencv_contrib/modules \
-D ENABLE_NEON=OFF \
-D ENABLE_VFPV3=OFF \
-D BUILD_TESTS=OFF \
-D INSTALL_PYTHON_EXAMPLES=OFF \
-D OPENCV_ENABLE_NONFREE=ON \
-D CMAKE_SHARED_LINKER_FLAGS=-latomic \
-D BUILD_EXAMPLES=OFF \
-D WITH_GTK=OFF ..
```

Diese Variante ist aber nur zu empfehlen, wenn man es nicht eilig hat. **Sie dauert 9 Stunden!**

5.5.3 Cross-Compiling auf dem Arbeitssystem (empfohlen)

Diese 9 Stunden, lassen sich drastisch reduzieren, wenn man den Kompiliervorgang auf seinem Arbeitssystem durchführt. Dies benötigt nur einen Bruchteil der Zeit, ist aber auch recht aufwendig.

Daher habe ich die von mir kompilierten Binaries in Version 4.5.1 ebenfalls ins GitHub-Repo gestellt, so dass wir diese von dort runterladen und installieren können.

Mit folgenden Befehlen ist die Installation erledigt:

```
curl -sSL https://github.com/faraway030/EDU_OpenCV-
Project/raw/master/PiZero_CV2/opencv_pizero-4.5.1.tar.bz2 | tar -xz -C
/opt/opencv

ln -s /opt/opencv/lib/python3.7/dist-packages/cv2 /usr/lib/python3/dist-
packages/cv2
```

Wir sollten nun das Verzeichnis /opt/opencv haben und ein Symlink im Pythonverzeichnis, der auf unser OpenCV verweist.

Wer sich dennoch an das Cross-Compiling wagen will, findet hier die notwendigen Schritte:

<https://solarianprogrammer.com/2019/08/07/cross-compile-opencv-raspberry-pi-zero-raspbian/>

5.5.4 Test

Wir führen denselben Test durch, wie in Kapitel 4.6.4.

5.6 Installation der restlichen Python-Pakete

Wir installieren die Pakete imutils und imagezmq. Diese installieren wir erst jetzt, weil imutils OpenCV als Abhängigkeit hat und dies sonst zu Problemen geführt hätte.

```
python3 -m pip install imutils imagezmq
```

5.7 Modifikationen rückgängig machen

Wir machen die Änderungen am Swap aus Kapitel 5.3 wieder rückgängig und starten den Pi neu. Den Memorysplit können wir so lassen, da wir die GPU nicht benötigen.

5.8 Installation der Projektsoftware

Nun laden wir wie bereits beim Pi 4 mit dem entsprechenden Installations-Script die Projektsoftware.

```
curl https://raw.githubusercontent.com/faraway030/EDU_OpenCV-Project/master/install-client.sh | bash
```

Wechseln in das Projektverzeichnis

```
cd /opt/OpenCV-Project
```

und starten einmal die client.py.

```
python3 client.py -s 10.0.60.1
```

Wenn diese ohne Fehlermeldungen startet, legen wir noch einen Cronjob an, der die client.py bei jedem Systemstart automatisch startet und sind dann fertig.

Dazu wechseln wir wieder zu unserem Standardbenutzer und tun Folgendes:

```
crontab -e
```



Abbildung 5.8.1

Wir speichern und nun sollte sich die client.py automatisch bei jedem Systemstart starten.

6 ESP32-Kamera

6.1 Modifikation der Firmware

Zuerst öffnen wir die Workspace-Datei aus dem Ordner Firmware/ESP32-CAM in VSCode.

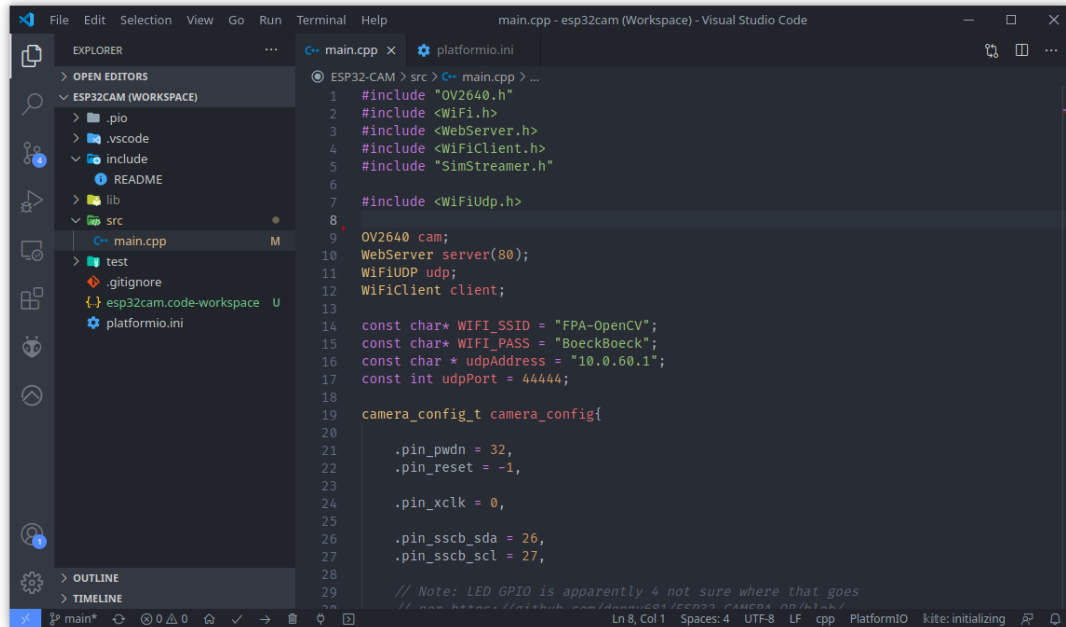


Abbildung 6.1.1

Dort öffnen wir, wenn sie nicht schon offen sein sollte die Datei src/main.cpp:
Wir gleichen die WiFi-Zugangsdaten und die IP des AP ab und ändern diese gegebenenfalls.

Anschließend klicken wir links auf den kleinen Alien von PlatformIO:
Bei Änderungen klicken wir einmal oben links auf „Build“ um die Firmware neu zu kompilieren.

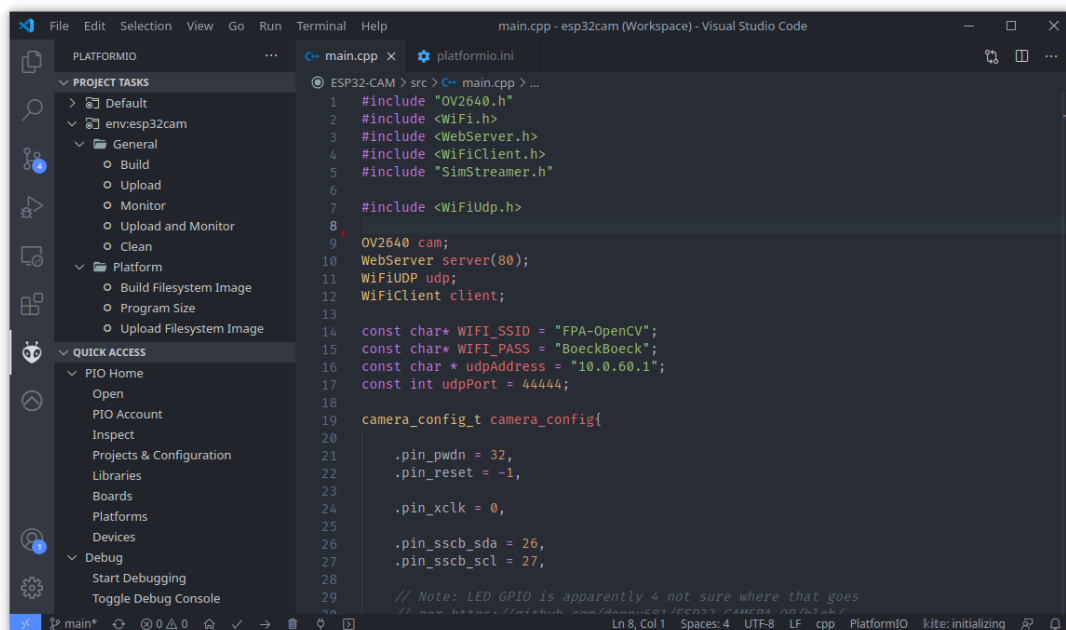


Abbildung 6.1.2

6.2 Flashen der Firmware

Da das Modul keinen USB-Anschluss hat, benötigen wir zum Flashen einen FTDI, der im Prinzip nur ein Adapter von USB zu serieller Schnittstelle ist.

Beide werden mittels Jumper-Kabel wie folgt verbunden.

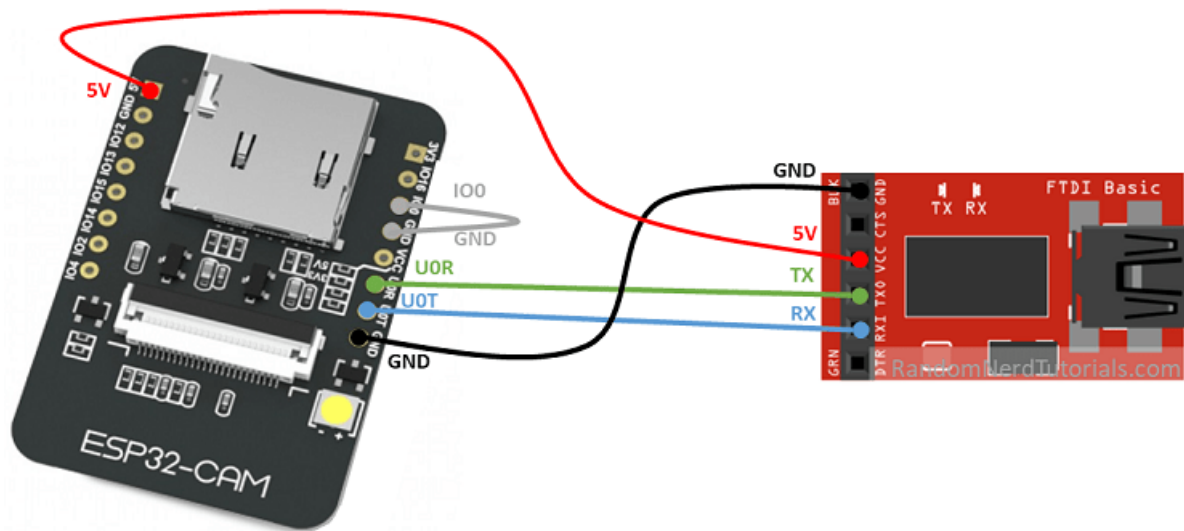


Abbildung 6.2.1

FTDI's haben meist einen 3V und einen 5V Ausgang. Wichtig ist, dass nur 3V auf 3V und 5V auf 5V verbunden wird.

Nun schließen das Ganze mit einem MicroUSB-Kabel an unser Arbeitssystem an und können dann mit „Upload“ oben links die Firmware auf das Modul schreiben.

Hat dies erfolgreich funktioniert, können wir alle Verbindungen wieder lösen und brauchen in Zukunft nur noch ein MicroUSB-Kabel als Spannungsquelle.

Es muss unbedingt darauf geachtet werden, GPIO0 mit GND (grau dargestellt), für den Flashvorgang zu brücken. Ansonsten lässt sich die Firmware nicht auf das Modul schreiben.

Ist der Vorgang erfolgreich, sieht das in etwa so aus:

```
Writing at 0x0001c000... (61 %)
Writing at 0x00020000... (69 %)
Writing at 0x00024000... (76 %)
Writing at 0x00028000... (84 %)
Writing at 0x0002c000... (92 %)
Writing at 0x00030000... (100 %)
Wrote 279888 bytes (205773 compressed) at 0x00000000 in 18.1 seconds (effective 123.4 kbit/s)...
Hash of data verified.

Leaving...
Hard resetting via RTS pin...
===== [SUCCESS] Took 20.61 seconds =====

Terminal will be reused by tasks, press any key to close it.
```

Abbildung 6.2.2

7 Installation des „FaceReact“-Moduls

7.1 Modifikation der Firmware

Analog zu 7.1

Ordner: Firmware/FaceReact

7.2 Flashen der Firmware

Das NodeMCU-Modul hat den Vorteil einen USB-Anschluss zu haben, weshalb der FTDI wegfällt. Wir können es also einfach per USB-Kabel anschließen und analog zu 7.2 flashen.

7.3 Installation

Anschließend erstellen wir mit 3 220 Ohm Widerständen folgenden Aufbau:

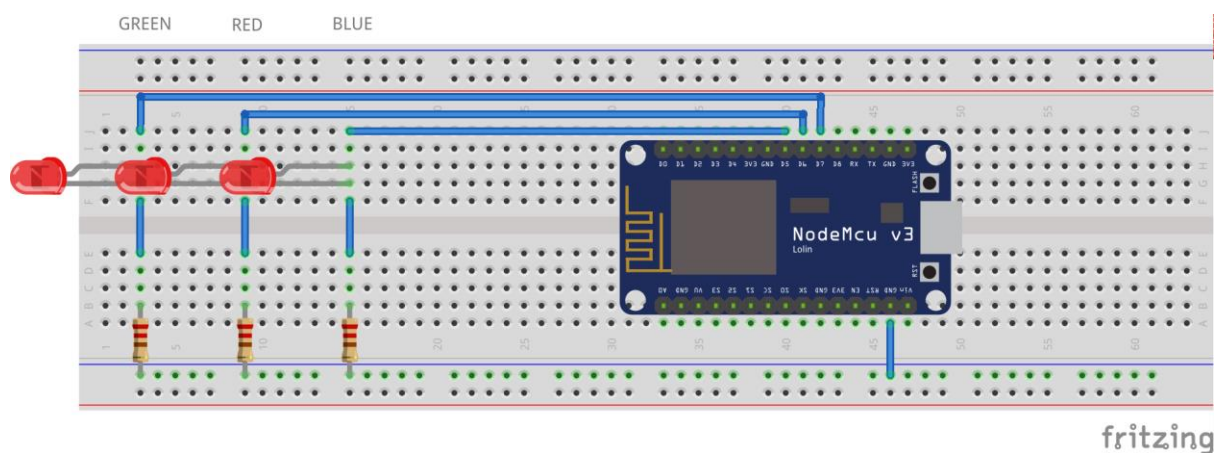


Abbildung 7.3.1

Die LED's sind wie folgt verbunden:

- BLUE -> D7
- RED -> D6
- GREEN -> D5

Leider gab es in der Software, mit der ich die Grafik erstellt habe, nur rote LED's, deshalb die Farbbezeichnungen am oberen Rand.

Wenn unser Pi 4 noch aktiv und der AP erreichbar ist, sollte die blaue LED nach erfolgreichem Flash- und Bootvorgang des Controllers blau aufleuchten. Dies zeigt an, dass eine WiFi- sowie MQTT-Verbindung zum Server besteht.

Funktionen der LED's:

Blau:	dauerhaftes Leuchten	->	bestehende Verbindung
	Blinken	->	Verbindungsversuch

Grün: bekanntes Gesicht erkannt

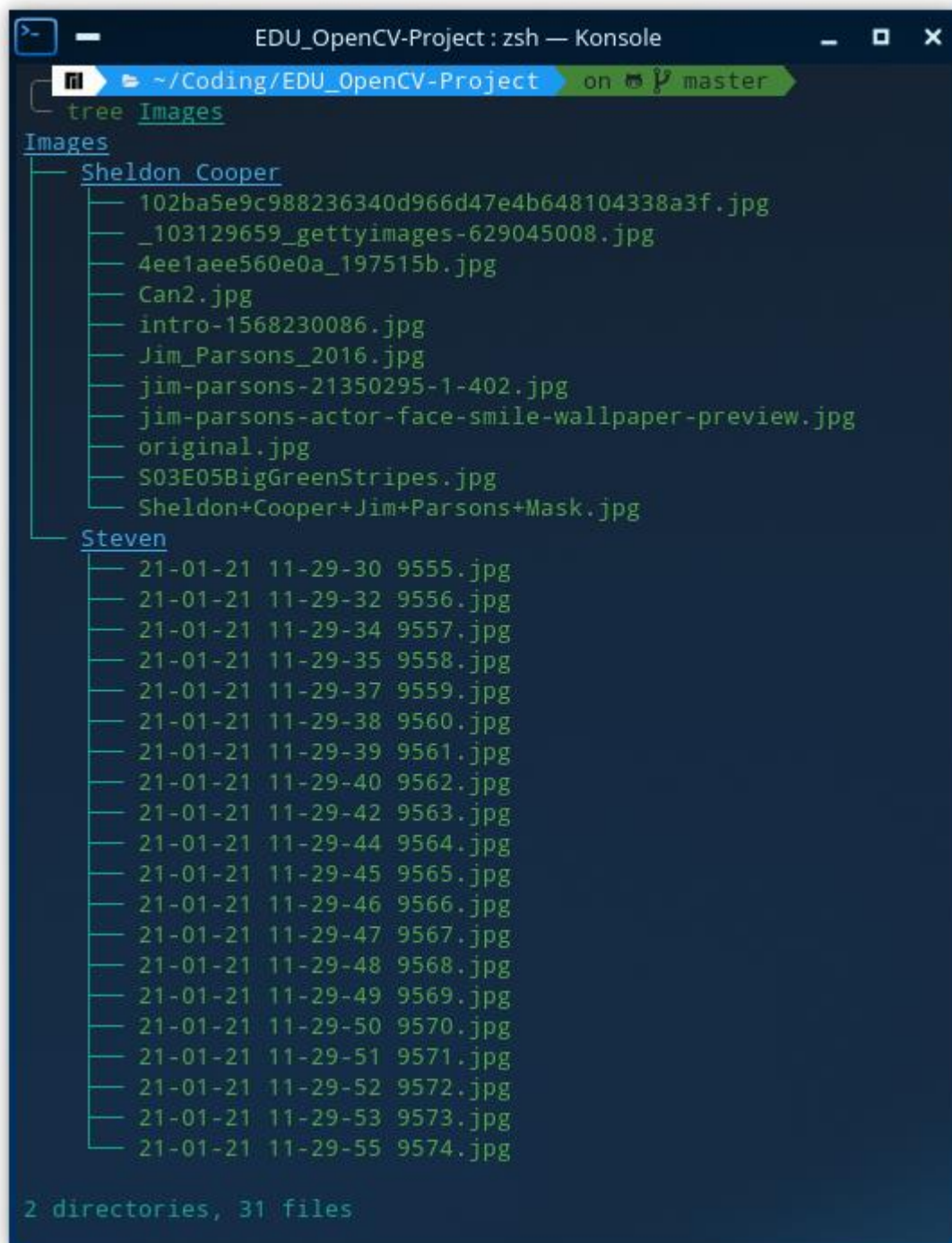
Rot: unbekanntes Gesicht erkannt

8 Dataset erstellen

Damit unser System auch Personen identifizieren kann, muss es diese zuerst einmal kennenlernen.

Dazu legen wir in unserem lokalen Projektordner einen Ordner „Images“ an und erstellen in ihm für jede zu erkennende Person einen Unterordner mit den jeweiligen Namen. Diese füllen wir dann mit Bildern der Person. Es müssen keine 20 sein, aber hier gilt: je mehr Daten, desto besser.

Das Ergebnis sollte dann in etwa so aussehen:



```

EDU_OpenCV-Project : zsh — Konsole
~/Coding/EDU_OpenCV-Project on master
tree Images
Images
├── Sheldon Cooper
│   ├── 102ba5e9c988236340d966d47e4b648104338a3f.jpg
│   ├── _103129659_gettyimages-629045008.jpg
│   ├── 4ee1aee560e0a_197515b.jpg
│   ├── Can2.jpg
│   ├── intro-1568230086.jpg
│   ├── Jim_Parsons_2016.jpg
│   ├── jim-parsons-21350295-1-402.jpg
│   ├── jim-parsons-actor-face-smile-wallpaper-preview.jpg
│   ├── original.jpg
│   ├── S03E05BigGreenStripes.jpg
│   └── Sheldon+Cooper+Jim+Parsons+Mask.jpg
└── Steven
    ├── 21-01-21 11-29-30 9555.jpg
    ├── 21-01-21 11-29-32 9556.jpg
    ├── 21-01-21 11-29-34 9557.jpg
    ├── 21-01-21 11-29-35 9558.jpg
    ├── 21-01-21 11-29-37 9559.jpg
    ├── 21-01-21 11-29-38 9560.jpg
    ├── 21-01-21 11-29-39 9561.jpg
    ├── 21-01-21 11-29-40 9562.jpg
    ├── 21-01-21 11-29-42 9563.jpg
    ├── 21-01-21 11-29-44 9564.jpg
    ├── 21-01-21 11-29-45 9565.jpg
    ├── 21-01-21 11-29-46 9566.jpg
    ├── 21-01-21 11-29-47 9567.jpg
    ├── 21-01-21 11-29-48 9568.jpg
    ├── 21-01-21 11-29-49 9569.jpg
    ├── 21-01-21 11-29-50 9570.jpg
    ├── 21-01-21 11-29-51 9571.jpg
    ├── 21-01-21 11-29-52 9572.jpg
    ├── 21-01-21 11-29-53 9573.jpg
    └── 21-01-21 11-29-55 9574.jpg

2 directories, 31 files
  
```

Abbildung 7.3.1

Nun erstellen wir mit folgendem Befehl unser Dataset:

```
python3 encode_images.py -i Images -e face_enc
```



Abbildung 7.3.2

Wir sollten nun die Datei „face_enc“ in unserem Ordner haben, welche wir noch auf den Pi 4 kopieren müssen. Dies machen wir mit dem Befehl:

```
scp face_enc user@pi4:/home/user
```

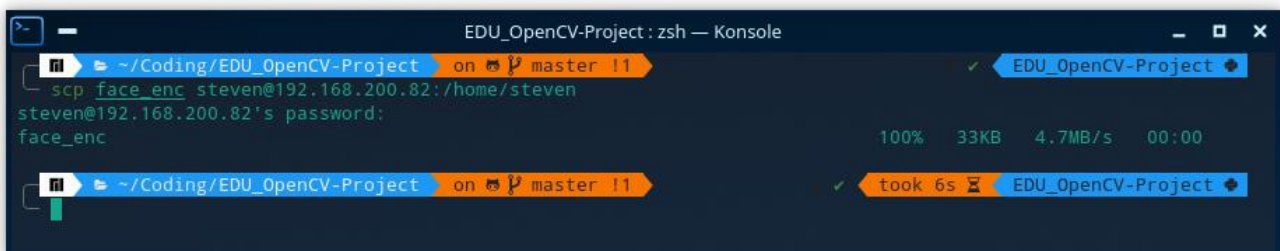


Abbildung 7.3.3

Anschließend loggen wir uns per SSH auf dem Pi 4 ein und verschieben die Datei aus dem Home-Verzeichnis nach „/opt/OpenCV-Project/data“.

9 Inbetriebnahme

9.1 Erste Tests im visuellen Modus

Wenn alle Schritte bis hierhin erfolgreich abgearbeitet sind, können wir den Aufbau nun testen. Dabei ist wichtig, dass alle Geräte bis auf den Pi 4 vom Strom genommen werden. (Die Kameras verbinden bei Systemstart und haben noch keine Reconnect-Funktionalität)

Dazu habe ich mir zuerst folgende Versuchsanordnung aufgebaut:



Abbildung 9.1.1

Anschließend öffnen wir auf unserem Arbeitssystem 2 Terminal-Fenster und starten in jedem eine SSH-Session zum Pi 4. Dabei ist wichtig, dass wir dem SSH-Befehl in dem Fenster, in dem wir den Server starten, ein Argument übergeben, was das X11-Forwarding aktiviert. Somit werden uns Fenster vom Pi 4 lokal auf unserem Arbeitssystem angezeigt.

```
ssh -X user@Pi4
```

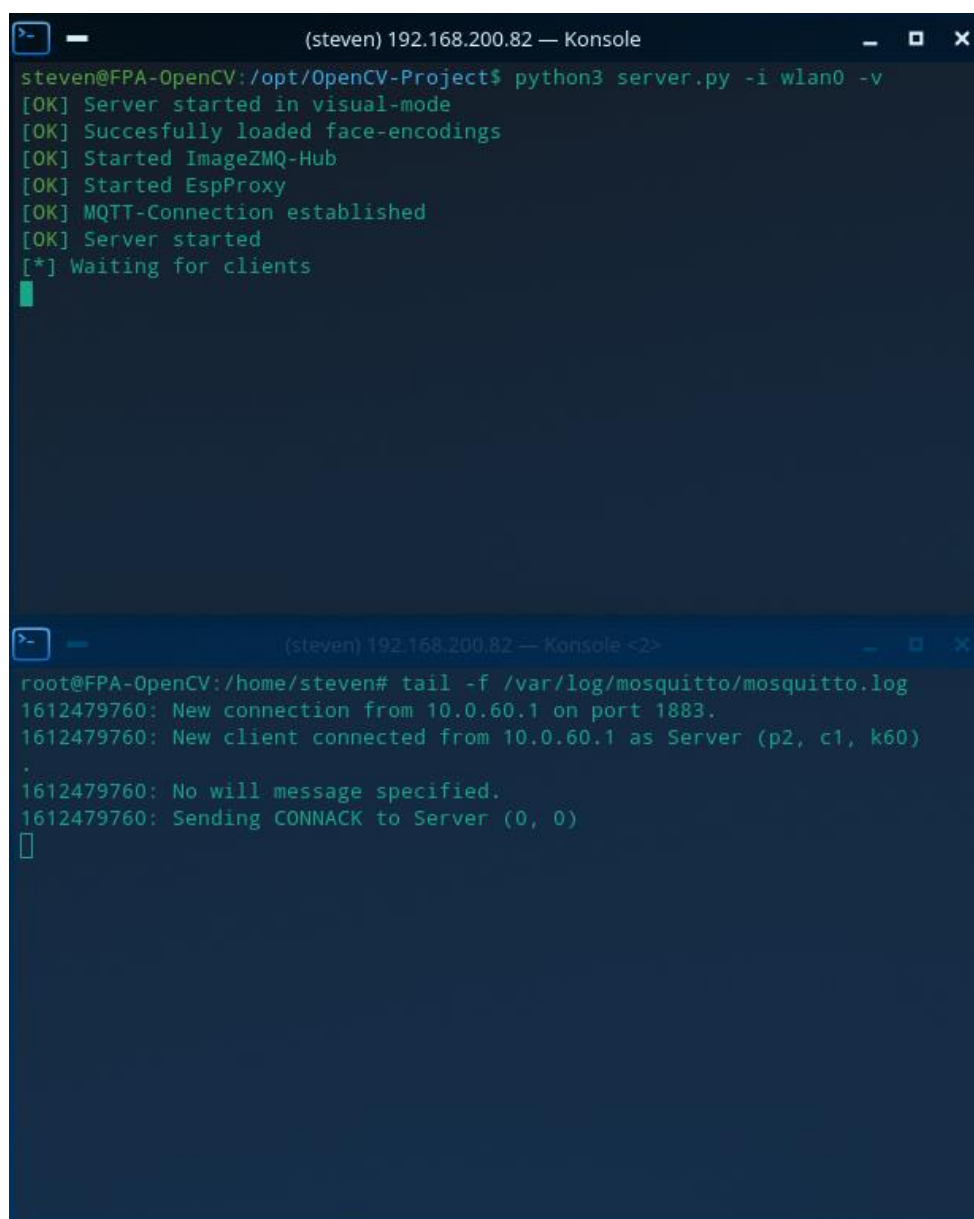
In dem zweiten Fenster lassen wir uns die Log-Datei von mosquitto ausgeben.

```
tail -f /var/log/mosquitto/mosquitto.log
```

In unserem Fenster mit X11-Forwarding starten wir nun die server.py.

```
cd /opt/OpenCV-Project  
python3 server.py -i wlan0 -v
```

Achtung: Dies muss als Standardbenutzer erfolgen, da die Datei die die Rechte für das X11-Forwarding hält, beim SSH-Login generiert wird. Da wir uns nicht als root verbinden, hat der root-User keine Berechtigung für das Forwarding.



```
(steven) 192.168.200.82 — Konsole  
steven@FPA-OpenCV:/opt/OpenCV-Project$ python3 server.py -i wlan0 -v  
[OK] Server started in visual-mode  
[OK] Successfully loaded face-encodings  
[OK] Started ImageZMQ-Hub  
[OK] Started EspProxy  
[OK] MQTT-Connection established  
[OK] Server started  
[*] Waiting for clients  
█  
  
(steven) 192.168.200.82 — Konsole <2>  
root@FPA-OpenCV:/home/steven# tail -f /var/log/mosquitto/mosquitto.log  
1612479760: New connection from 10.0.60.1 on port 1883.  
1612479760: New client connected from 10.0.60.1 as Server (p2, c1, k60)  
.  
1612479760: No will message specified.  
1612479760: Sending CONNACK to Server (0, 0)  
█
```

Abbildung 9.1.2

Anschließend schließen wir unser FaceReact-Modul an den Strom an. Die blaue LED sollte kurz blinken und dann dauerhaft leuchten.

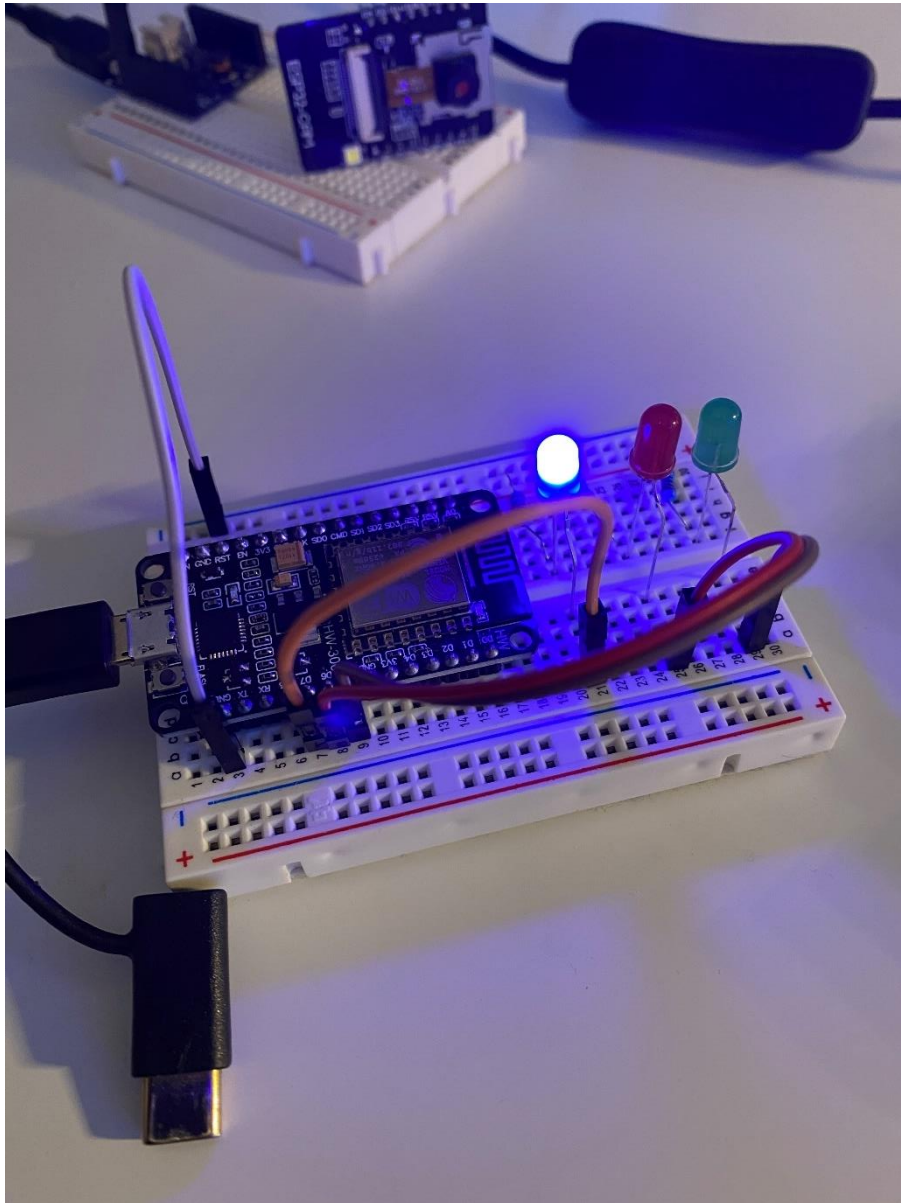
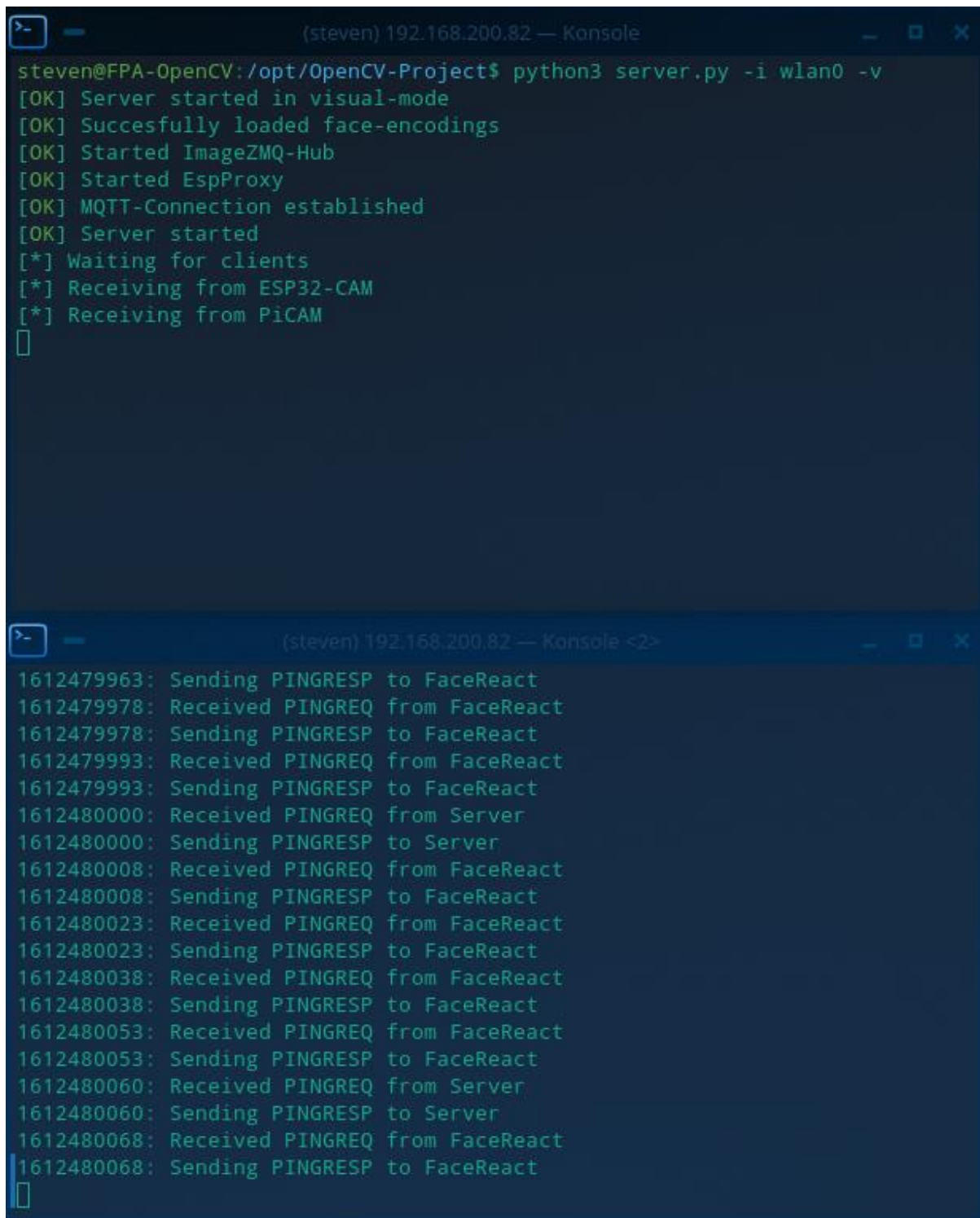


Abbildung 9.1.3

Ebenfalls sollten wir in der Mosquitto-Log sehen, dass sich FaceReact als Client angemeldet hat.

Nun schließen wir den PiZero und den ESP32 an den Strom an. Der PiZero wird hier wegen des längeren Bootvorgangs etwas auf sich warten lassen.

Wir sollten dann in unserem Serverfenster folgendes sehen:



The image displays two terminal windows. The top window shows the execution of a Python script that starts a server in visual mode, loads face encodings, starts an ImageZMQ-Hub, an EspProxy, and establishes an MQTT connection. It then reports receiving data from ESP32-CAM and PiCAM. The bottom window shows a series of ping requests and responses between a client (IP 1612479963) and a server (IP 1612480000), as well as between the client and FaceReact (IP 1612479978).

```
(steven) 192.168.200.82 — Konsole
steven@FPA-OpenCV:/opt/OpenCV-Project$ python3 server.py -i wlan0 -v
[OK] Server started in visual-mode
[OK] Succesfully loaded face-encodings
[OK] Started ImageZMQ-Hub
[OK] Started EspProxy
[OK] MQTT-Connection established
[OK] Server started
[*] Waiting for clients
[*] Receiving from ESP32-CAM
[*] Receiving from PiCAM
[]

(steven) 192.168.200.82 — Konsole <2>
1612479963: Sending PINGRESP to FaceReact
1612479978: Received PINGREQ from FaceReact
1612479978: Sending PINGRESP to FaceReact
1612479993: Received PINGREQ from FaceReact
1612479993: Sending PINGRESP to FaceReact
1612480000: Received PINGREQ from Server
1612480000: Sending PINGRESP to Server
1612480008: Received PINGREQ from FaceReact
1612480008: Sending PINGRESP to FaceReact
1612480023: Received PINGREQ from FaceReact
1612480023: Sending PINGRESP to FaceReact
1612480038: Received PINGREQ from FaceReact
1612480038: Sending PINGRESP to FaceReact
1612480053: Received PINGREQ from FaceReact
1612480053: Sending PINGRESP to FaceReact
1612480060: Received PINGREQ from Server
1612480060: Sending PINGRESP to Server
1612480068: Received PINGREQ from FaceReact
1612480068: Sending PINGRESP to FaceReact
[]
```

Abbildung 9.1.4

Da wir die `server.py` mit dem Argument „-v“ für den visuellen Modus gestartet haben, sollte sich nun ein Fenster öffnen, das uns das Kamerabild anzeigt. Leider ist aufgrund der suboptimalen Übertragungsprozesse des ESP32-Moduls die Übertragung etwas verzögert und nicht synchron mit der, des PiZero. Leider konnte ich das in dem zeitlichen Rahmen des Projekts nicht anders lösen.

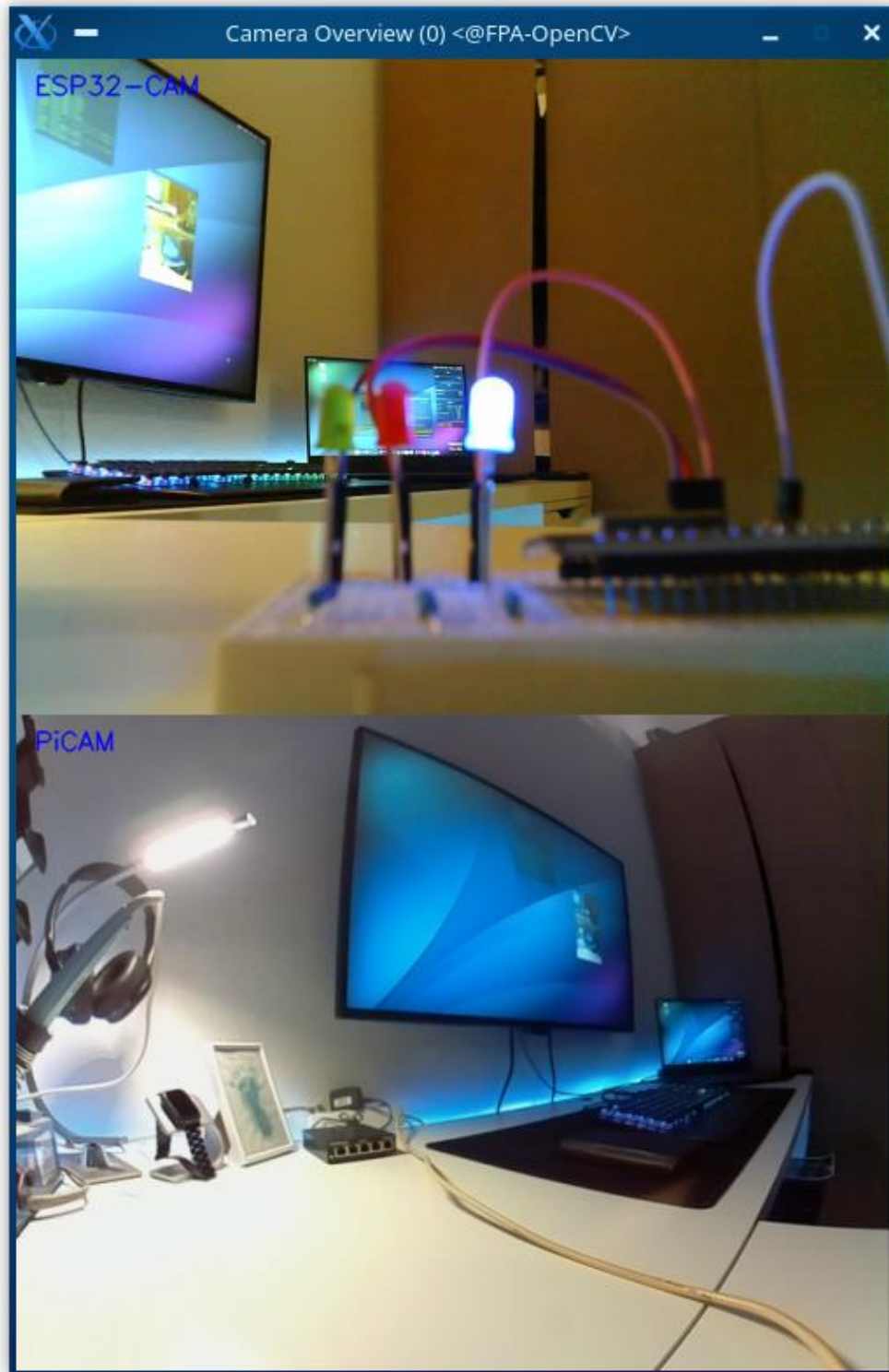


Abbildung 9.1.5

Da die Infrastruktur nun steht und alles funktioniert, testen wir noch die eigentliche Funktionalität. Die Kameras sollten Gesichter, die wir unserem Dataset hinzugefügt haben erkennen und benennen und alle anderen als unbekannt markieren. Weiterhin sollte die entsprechende LED des FaceReact-Moduls für 10 Sekunden (Firmware-Einstellung) aufleuchten.

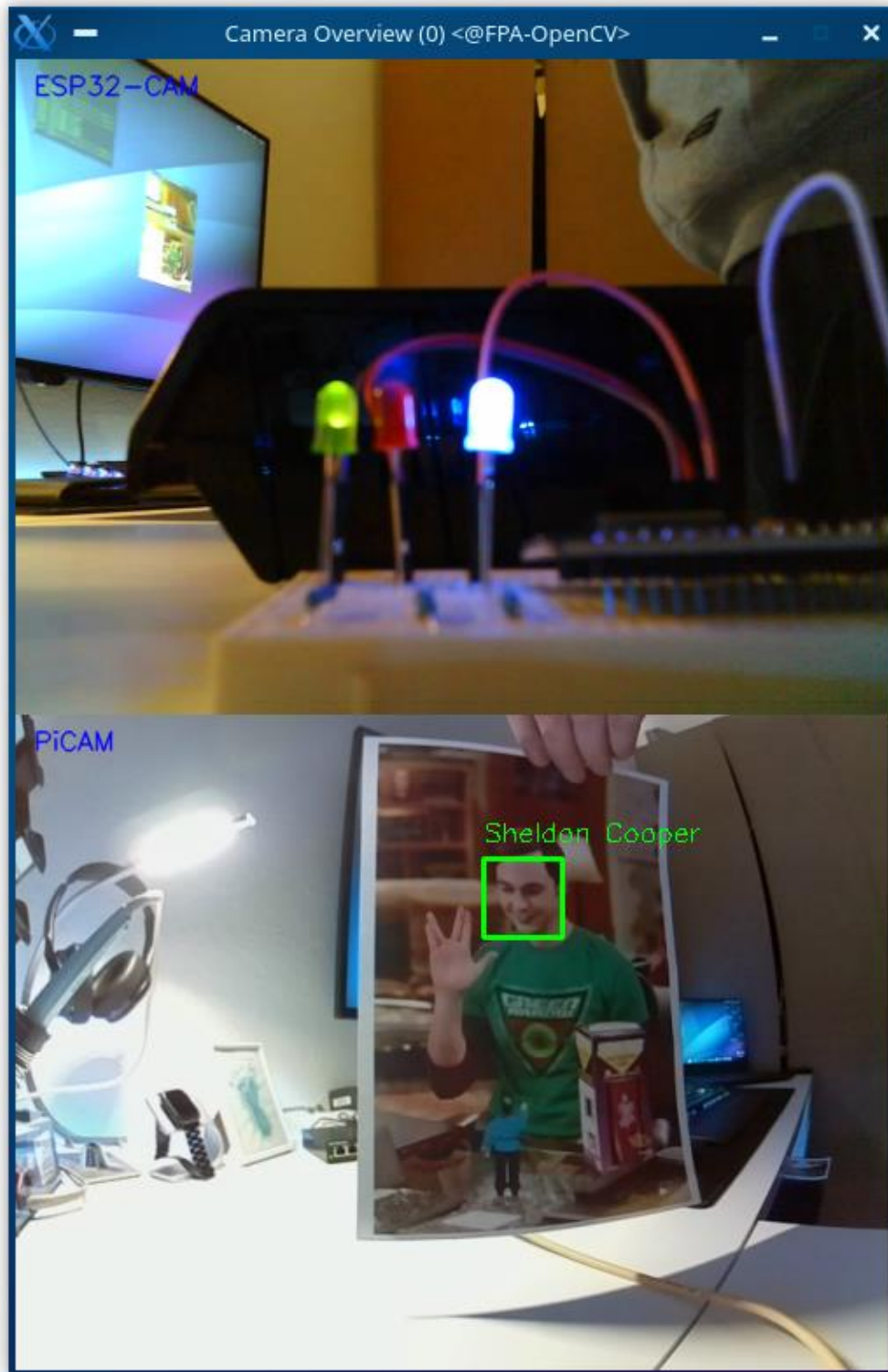


Abbildung 9.1.6

Und noch ein unbekanntes Gesicht:

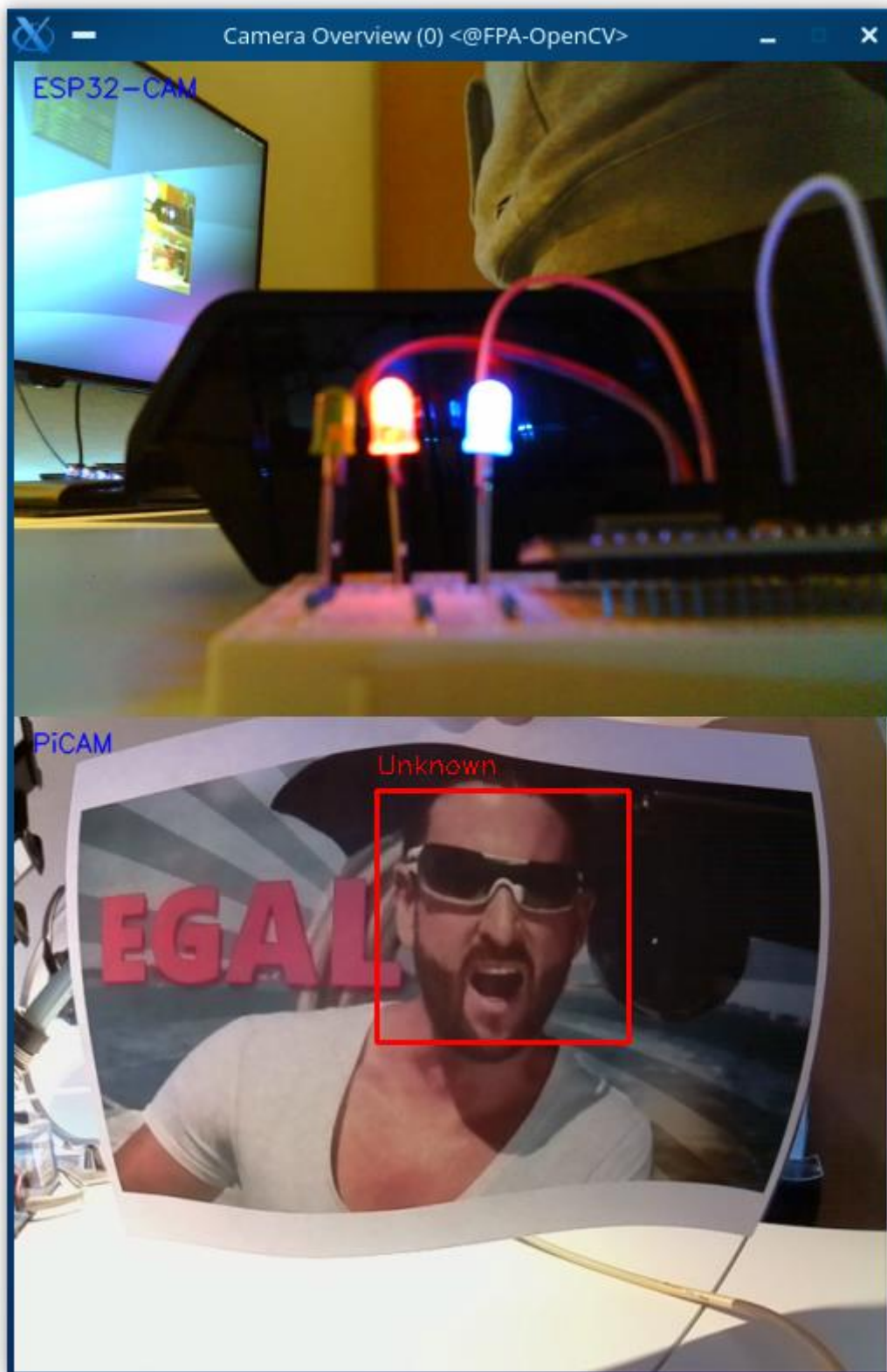


Abbildung 9.1.7

9.2 Betriebsmodus

Für den Betriebsmodus richten wir den Start der `server.py` ohne visuellen Modus wie in Kapitel 5.8 als Cronjob ein und können das System dann autark mit dem FaceReact-Modul als Ausgabeeinheit betreiben.

Dies soll symbolisch mögliche Anwendungen für ein solches System darstellen, wie z.B. das Laden personalisierter Konfigurationen für verschiedene Nutzer, Erkennung und Meldung unbefugter Personen an einem Arbeitsplatz, individuelle Reaktionen etc. Für Zugriffskontrollen ist dies jedoch aufgrund der Manipulationsmöglichkeiten nicht geeignet.

10 Fazit

Obwohl, bzw. vielleicht gerade weil ich bei diesem Projekt mit zahlreichen Problemen und Technologien konfrontiert war, hat mir dieses Projekt sehr viel Spaß gemacht. Ich bin leider etwas holprig gestartet, habe die ersten Tage andere Ideen aufgefasst und wieder verworfen, bis schließlich dieses Projekt entstanden ist. Da ich jedoch die Hardware erst bestellen musste, ging mir in Summe ca. eine Woche an produktiver Arbeitszeit verloren.

Angedacht war eigentlich ein System, was einerseits Personen anhand ihres Gesichts identifiziert (Face Recognition) und andererseits menschliche Körper erkennt (Object Detection) und somit wie ein optischer Bewegungsmelder die Anwesenheit erfasst. Beide Mechanismen sollten über einen eigenen Microcontroller die Auswertung sichtbar machen. Leider habe ich den Aufwand des Ganzen inkl. Dokumentation unterschätzt und musste die Object Detection aus dem Projekt wieder entfernen. Umsetzen werde ich sie dann privat.

Die Software zu diesem Projekt würde ich eher als beta bezeichnen, da es noch einige Dinge zu verbessern gibt, die Gesichtserkennung noch deutlich optimierbar ist und insbesondere die Datenübertragung von der ESP32-Kamera sehr suboptimal umgesetzt ist.

Mein Ziel, zu demonstrieren, wie so etwas mit einfachen Mitteln und relativ wenig Code umsetzbar ist, habe ich jedoch erreicht und hoffe, dass es unterhaltsam war.

Vielen Dank für die Aufmerksamkeit
Steven Bruck