Einrichtungsroutine Smart Cam

Datum 26.03.21
Getestet auf Raspberry Pi 4 4GB / Google Cloud Platform

Inhalt

Lokales Setup

Overclock

nicht notwendig - empfehlenswert bei guter Kühlung -> reduziert Einrichtungszeit

```
sudo apt update
sudo apt dist-upgrade
sudo reboot
sudo nano /boot/config.txt

#uncomment to overclock the arm. 700 MHz is the default.
over_voltage=2
arm_freq=1750

sudo reboot
```

XRDP (Windows Remote Desktop)

nicht notwendig - nur benötigt falls Remote Desktop erwünscht

```
sudo apt-get update
sudo apt-get install xrdp -y
sudo systemctl status xrdp
ifconfig
```

Windows:

Remotedesktopverbindung öffnen und bei der IP Adresse aus ifconfig anmelden mit bei der Einrichtung gesetzten Nutzerdaten.

Samba

nicht notwendig - nur benötigt falls Directory Zugriff von Windows auf RPI gewünscht

```
sudo apt-get install samba samba-common-bin -y
sudo nano /etc/samba/smb.conf
```

Runterscrollen bis *Share Definitions*, Elemente wie folgt einfügen/entfernen:

#======================= Share Definitions =======================
[pihome]
comment=Pi Home
path=/home/pi
browseable= yes
read only = no
only guest=no
create mask=*0777*
directory mask=*0777*
public=no

```
sudo smbpasswd -a pi
```

Windows: Netzwerklaufwerk hinzufügen:

\\IPADDRESS_OF_PI\pihome

Kamera

In Raspberry Pi Config -> Kamera aktivieren und Raspberry neustarten

MariaDB

```
sudo apt-get update
sudo apt-get upgrade
sudo apt-get install mariadb-client mariadb-server -y
sudo mysql -u root -p
```

```
flush privileges;
exit;
```

```
sudo nano /etc/mysql/my.conf
```

port = 3306
bind-address = :: #Erlaubt den Zugriff auf die DB von jeder IP-Adresse aus

```
sudo mysql_secure_installation
```

```
sudo /etc/init.d/mysql restart
sudo systemctl enable mysql
```

Mosquitto

```
sudo apt-get install mosquitto mosquitto-clients -y
sudo systemctl enable mosquitto
```

OpenCV 4.51

Etwa mit 1.5h build zu rechnen

```
sudo apt-get update
sudo apt-get upgrade
sudo apt-get install cmake gfortran libjpeg-dev libtiff-
dev libgif-dev libavcodec-dev libavformat-dev libswscale-dev
libgtk2.0-dev libcanberra-gtk* libxvidcore-dev libx264-dev
libgtk-3-dev libtbb2 libtbb-dev libdc1394-22-dev libv4l-dev
libopenblas-dev libatlas-base-dev libblas-dev libjasper-
dev liblapack-dev libhdf5-dev protobuf-compiler -y
```

```
cd ~
wget -O opencv.zip
https://github.com/opencv/opencv/archive/4.5.1.zip
wget -O opencv_contrib.zip
https://github.com/opencv/opencv_contrib/archive/4.5.1.zip


unzip opencv.zip
unzip opencv_contrib.zip

mv opencv-4.5.1 opencv
mv opencv_contrib-4.5.1 opencv_contrib
pip3 install numpy

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

```
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 WITH_OPENMP=ON \ -D
BUILD_TIFF=ON \ -D WITH_FFMPEG=ON \ -D WITH_TBB=ON \ -D
BUILD_TBB=ON \ -D BUILD_TESTS=OFF \ -D WITH_EIGEN=OFF \ -D
WITH_V4L=ON \ -D WITH_LIBV4L=ON \ -D WITH_VTK=OFF \ -D
WITH_QT=OFF \ -D OPENCV_ENABLE_NONFREE=ON \ -D
INSTALL_C_EXAMPLES=OFF \ -D INSTALL_PYTHON_EXAMPLES=OFF \ -D
BUILD_NEW_PYTHON_SUPPORT=ON \ -D BUILD_opencv_python3=TRUE \
-D OPENCV_GENERATE_PKGCONFIG=ON \ -D BUILD_EXAMPLES=OFF ..
```

```
sudo nano /etc/dphys-swapfile
```

CONF_SWAPSIZE=2048

```
sudo /etc/init.d/dphys-swapfile stop
sudo /etc/init.d/dphys-swapfile start

make -j4

sudo make install
sudo ldconfig
make clean

sudo apt-get update

sudo nano /etc/dphys-swapfile
```

CONF_SWAPSIZE=100

```
cd ~
rm opencv.zip
rm opencv_contrib.zip
sudo reboot

sudo rm -rf ~/opencv
sudo rm -rf ~/opencv_contrib
```

Tensorflow (Build only)

Muss auf einem Docker fähigen Rechner erfolgen.

Ein Build ist **bereits** in dem Projekt enthalten sein für Raspberry Pi Python 3.7 mit Tensorflow 2.5: tensorflow-2.5.0-cp37-none-linux_armv7l.whl

Anbei nur die Anleitung, wie der Build erstellt wurde:

```
sudo apt update
sudo apt install --yes apt-transport-https ca-certificates
curl gnupg2 software-properties-common
curl -fsSL https://download.docker.com/linux/debian/gpg |
sudo apt-key add -
sudo add-apt-repository "deb [arch=amd64]
https://download.docker.com/linux/debian $(lsb_release -cs)
stable"
sudo apt update
sudo apt install --yes docker-ce

sudo usermod -aG docker $USER
logout


git clone https://github.com/tensorflow/tensorflow.git
cd tensorflow

tensorflow/tools/ci_build/ci_build.sh PI-PYTHON37 \
tensorflow/tools/ci_build/pi/build_raspberry_pi.sh
```

Tensorflow

im Directory der tensorflow wheel Datei tensorflow-2.5.0-cp37-none-linux_armv7l.whl

```
sudo pip3 install tensorflow-2.5.0-cp37-none-linux_armv7l.whl
```

Python Libraries

```
sudo pip3 install absl-py mariadb paho-mqtt PyJWT django
pymysql cryptography
```

Einrichtung Datenbank

Eine saubere Datenbank ist als clean_database.sql im Projekt enthalten.

name und *password* sind identisch mit der im Projekt enthaltenen /data/mariadb_config.json. Bitte diese ggf. auch anpassen!

```
sudo mysql
```

```
CREATE USER 'name'@'localhost' IDENTIFIED BY 'password';
GRANT ALL PRIVILEGES ON * . * TO 'name'@'localhost';
FLUSH PRIVILEGES;
exit;

mysql -u name -p

CREATE DATABASE smart_cam;
exit;
```

Im Directory der sql Datei:

```
mysql -u name -p smart_cam < clean_database.sql
```

Aufbau Datenbank:

Tabelle cameras

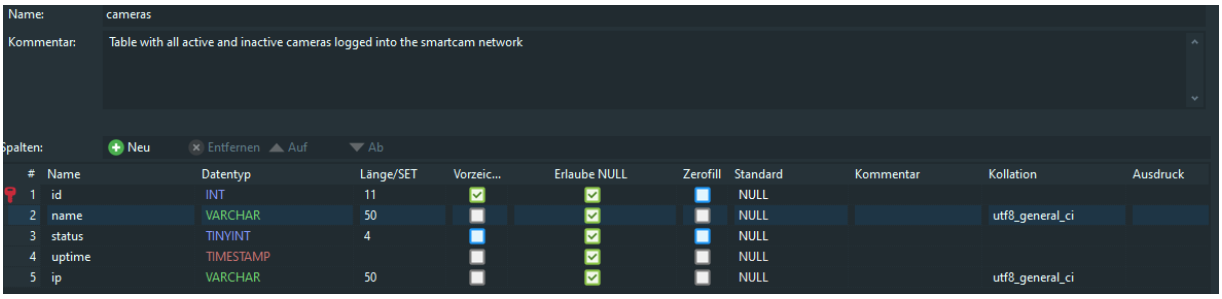
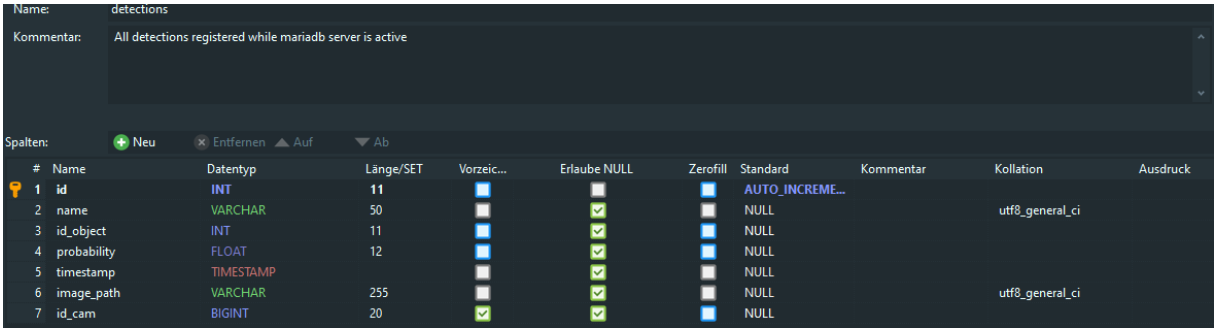


Tabelle detections



Django Webserver

In django/smartcam/smartcam/settings.py müssen die IP Adresse des Raspberry und die MariaDB Zugangsdaten angegeben werden:

```
ALLOWED_HOSTS = ['PIADRESSE', '127.0.0.1']

DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.mysql',
        'NAME': 'smart_cam',
        'USER': 'name',
        'PASSWORD': 'password',
        'HOST': '127.0.0.1',
       'PORT': '3306',
    }
}
```

In /data/django_config.json müssen username und password für den Superuser angegeben werden (wird anschließend erstellt). Ebenso die Email Adresse und Passwort von dem zu sendenden Benachrichtigungsdienst. Auch an welche Email versendet werden soll.

```
projektname/django/smartcam $ python3 manage.py
createsuperuser
```

Start des lokalen Setups

Nun kann das Setup lokal ausgeführt werden. Dazu werden 3 verschiedene Skripte gestartet:

Webserver:

```
projektname/django/smartcam $ python3 manage.py runserver 0.0.0.0:8000
```

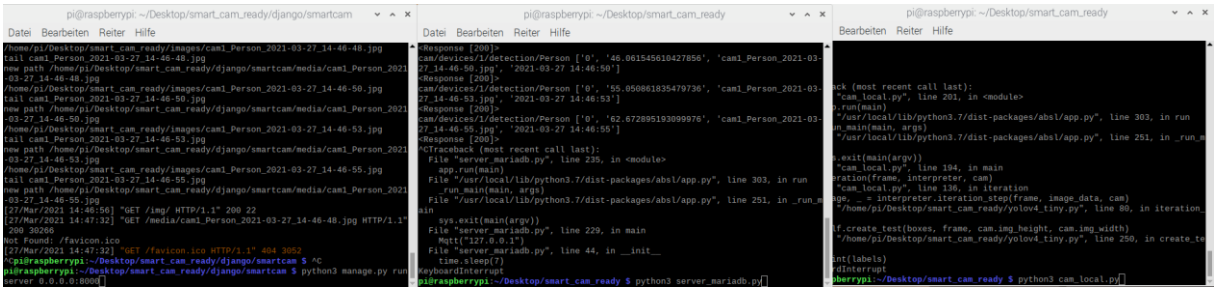
MariaDB Server:

```
projektname $ python3 server_mariadb.py
```

Kamera:

```
projektname $ python3 cam_local.py
```

Ausschnitt:



Für zusätzliche Einstellungen bitte die möglichen Parameter beachten! (In den Python Skripten nachverfolgbar)

Google Cloud Setup

Das lokale Setup ist trotz anderer Struktur auszuführen, da Gemeinsamkeiten gegeben sind. Es wurde zumindest nur so getestet; in der Theorie können die Kapitel MariaDB, Mosquitto, Tensorflow und Django Webserver übersprungen werden. Mit kompletter Durchführung sind aber alle Bedingungen für die Kamera gegeben und es muss nur die Google Cloud eingerichtet und Einstellungen konfiguriert werden.

Es wird davon ausgegangen, dass bereits ein Konto und Projekt erstellt wurde bei Google Cloud Platform. Zusätzlich sollte ein Rechnungskonto aktiviert sein.

SQL-Datenbank

Auf den Reiter SQL -> Instanz erstellen -> MySQL (ggf. API aktivieren):

Nun nach gewünschter Region/Konfiguration einstellen. Folgende Einstellungen wurden für dieses Projekt verwendet:

Instanz-ID: cam-mysql-db
Passwort: [generiert]
Datenbankversion: MySQL 8.0
Region: europe-west1 (Belgien)
Zone: Einzelne Zone

Auf den Reiter Datenbank -> Datenbank erstellen (Datenbankname: smart_cam, utf8)

Auf den Reiter Nutzer -> Nutzerkonto hinzufügen (name, password [wie in Django!], beliebiger Host zulassen)

Es muss noch eine Verbindung zwischen VM Instanz und SQL-Datenbank erstellt werden. Dafür muss aber erst eine VM Instanz hergestellt werden. Aber zunächst die MQTT Bridge.

MQTT-Bridge

Auf den Reiter IoT Core -> (ggf. API aktivieren) -> Registry erstellen:

Registry-ID: cam
Region: europe-west1
Cloud Pub/Sub Themen: Erstellen (ID des Themas: cam)

Geräte -> Gerät erstellen (Geräte-ID: cam-google) -> Kommunikation, Cloud Logging, Authentifizierung aufklappen -> Authentifizierung hochladen (Schlüsselformat RS256_X509, Wert: Aus Projekt rsa_public.pem)

Achtung: Das Schlüsselpaar wurde im Rahmen der Bachelorarbeit erstellt und sollte nicht weiterverwendet werden. Um ein eigenes Schlüsselpaar zu erstellen siehe https://cloud.google.com/iot/docs/how-tos/credentials/keys.

Wiederhole Geräteerstellung für Geräte-ID vm-instance

Auf den Reiter Pub/Sub -> Thema cam -> Abo hinzufügen

Abo-ID: cam_abo
Zustellungstyp: Pull
Expiration period: Nie

VM Instanz

Auf den Reiter Compute Engine -> VM-Instanz erstellen:

Nun nach gewünschter Region/Konfiguration einstellen. Folgende Einstellungen wurden für dieses Projekt verwendet:

Region: europe-west1 (Belgien)
Zone: europe-west1-b
Machinentyp: e2-standard-4 (4 vCPU, 16 GB Arbeitsspeicher)
Bootlaufwerk: Nichtflüchtiger Standartspeicher mit 100 GB
Deep Learning Image: TensorFlow 2.4 m66 CUDA 11.0
Firewall: ☑ HTTP-Traffic zulassen
☑ HTTPS-Traffic zulassen

Nun Zugriff über SSH -> Konsole öffnet sich:

```
sudo apt update
sudo apt install apache2 -y
sudo apt install libapache2-mod-wsgi apache2-dev -y
sudo apt-get install libapache2-mod-wsgi-py3 -y

pip install mod_wsgi
mod_wsgi-express module-config
```

Mögliche Einstellungen die zuvor noch am besten angepasst werden sollten:

In django/smartcam/smartcam/settings.py müssen die **IP Adresse der VM** angegeben werden und die MariaDB Zugangsdaten angegeben werden. **Achte** Host ist IP Adresse der SQL Datenbank. Diese kann als Öffentliche IP-Adresse aus der SQL-Instanzen Übersicht betrachtet werden. Port entfällt im Normalfall. Falls die VM durch eine andere IP-Adresse erreichbar werden sollte, dann müssen die stets in ALLOWED_HOSTS hinzugefügt werden. Für HTTPS Verbindungen sind nur Namenadressen zulässig (**benötigt**)

```
ALLOWED_HOSTS = ['VMADRESSE',  'NAMENADRESSEZUVM', '127.0.0.1']

DATABASES = {
   'default': {
      'ENGINE': 'django.db.backends.mysql',
      'NAME': 'smart_cam',
      'USER': 'name',
      'PASSWORD': 'password',
      'HOST': 'SQLDATENBANKADRESSE',
   }
}
```

Die Projekt Daten müssen auch im Skript geändert werden. Diese können auch als Parameter erfolgen. Empfohlen wird aber dies im Skript anzupassen (Damit später nichts vergessen wird). Dazu einfach vm_google_cloud.py überprüfen/bearbeiten:

```
FLAGS = flags.FLAGS
flags.DEFINE_string('project_id', cam-mysql-db, 'project id of google cloud
platform')
flags.DEFINE_string('cloud_region', 'europe-west1', 'cloud region of the
google IoT Core registry')
flags.DEFINE_string('registry_id', 'cam', 'registry id in the google IoT
Core')
flags.DEFINE_string('abo_id', 'cam_abo', 'abo id for the pub/sub theme')
flags.DEFINE_string('device_id', 'vm-instance', 'device id that is already
added in the corresponding registry id')

[…]

os.environ["GOOGLE_APPLICATION_CREDENTIALS"] = './data/objectreccam-
736626dabbe7.json'
```

Achtung: Es werden Google Application Credentials benötigt. Dies ist eine von Google generierte Authentifizierungsdatei im json Format. Siehe https://cloud.google.com/docs/authentication/production?hl=de für mehr Informationen. Diese ist entsprechend in data Directory zu hinterlegen und im Script anzupassen.

In der Übersicht der VM-Instanzen wird nun die Externe IP-Adresse der aktuellen Instanz kopiert. Im Reiter SQL geht man nun auf die vorhin erstellte SQL Instanz und auf Verbindungen -> NETWERK HINZUFÜGEN -> Netz: IP der VM nun einfügen -> speichern

Nun über das Einstellungsfenster aus der SSH Konsole das Projekt smart_cam_vm als Zip hochladen.

Webserver

```
unzip smart_cam_vm.zip
cd smart_cam_vm/django/smartcam

virtualenv env
. env/bin/activate

pip3 install django pymysql pillow


python3 manage.py makemigrations
python3 manage.py migrate
python3 manage.py collectstatic

python3 manage.py createsuperuser

deactivate

cd /etc/apache2/sites-available

sudo touch smartcam.conf
sudo nano smartcam.conf
```

Nun wird die in Django vorhin eingeführte Namenadresse mit Apache2 zugewiesen. Dazu wird ein Virtual Host mit folgender Einstellung in smartcam.conf editiert.

Achtung: Die Einstellungen müssen entsprechend dem in der VM angegeben User unter Home und der IP Domain angepasst werden! **Kein Copy Paste**! Die IP Domain ist in dem Beispiel smartcam.ddns.net und der User user_name. Empfohlen: In einem anderen übersichtlicheren Editor bearbeiten.

```
<VirtualHost *:80>
        ServerAdmin admin@smartcam.ddns.net
        ServerName smartcam.ddns.net
        ServerAlias smartcam.ddns.net
        DocumentRoot /home/user_name/smart_cam_vm/django/smartcam
        ErrorLog ${APACHE_LOG_DIR}/error.log
        CustomLog ${APACHE_LOG_DIR}/access.log combined

        Alias /static /home/user_name/smart_cam_vm/django/smartcam/static
        <Directory /home/user_name/smart_cam_vm/django/smartcam/static>
                Require all granted
        </Directory>

        Alias /static /home/user_name/smart_cam_vm/django/smartcam/media
        <Directory /home/user_name/smart_cam_vm/django/smartcam/media>
                Require all granted
        </Directory>

        <Directory /home/user_name/smart_cam_vm/django/smartcam>
                <Files wsgi.py>
                        Require all granted
                </Files>
        </Directory>

        WSGIDaemonProcess smartcam python-
path=/home/user_name/smart_cam_vm/django/smartcam python-
home=/home/user_name/smart_cam_vm/django/smartcam/env
        WSGIProcessGroup smartcam
        WSGIScriptAlias /
/home/user_name/smart_cam_vm/django/smartcam/smartcam/wsgi.py
</VirtualHost>
```

```
sudo a2ensite smartcam.conf
sudo nano /etc/hosts
```

Neuer Eintrag (Durch verwendete Namensdomain tauschen):

```
127.0.0.1     smartcam.ddns.net
```

Achtung: Nun wieder user_name durch den im home Verzeichnis angegeben user tauschen

```
sudo chmod 664
/home/user_name/smart_cam_vm/django/smartcam/db.sqlite3

sudo chown :www-data
/home/user_name/smart_cam_vm/django/smartcam/db.sqlite3

sudo chown :www-data
/home/user_name/smart_cam_vm/django/smartcam/

sudo apache2ctl configtest
```

```
cd home/user_name/smart_cam_vm/django/smartcam/

source env/bin/activate

python3 manage.py runserver 0.0.0.0:80

sudo service apache2 restart

deactivate
```

HTTPS Verbindung

```
cd /etc/apache2/sites-available

pip install mod_wsgi
sudo apt update
sudo apt install snapd -y
sudo snap install core; sudo snap refresh core
sudo snap install --classic certbot
sudo ln -s /snap/bin/certbot /usr/bin/certbot
sudo certbot --apache
sudo nano smartcam.conf
```

Kommentieren der Zeilen:

#WSGIScriptAlias ….
#WSGIDaemonProcess ….
#WSGIProcessGroup ….

```
sudo certbot --apache
```

Nun im der neu erstellten Config die Zeilen wieder auskommentieren:

```
sudo nano smartcam-le-ssl.conf
```

WSGIScriptAlias ….
WSGIDaemonProcess ….
WSGIProcessGroup ….

```
sudo service apache2 restart
```

Der HTTPS Server unter der Namendomain smartcam.ddns.net funktioniert nun und zeigt “Smart Cam” an.

Einrichtung Datenbank

Achtung: Nun wieder user_name auswechseln mit dem im Home Verzeichnis angegebenen. Bei [INSTANCE_IP] die IP Adresse der SQL Datenbank angeben!

```
cd /home/user_name/
sudo apt-get update
sudo apt-get upgrade
sudo apt-get install mariadb-server-10.3
mysql --host=[INSTANCE_IP] -u name -p smart_cam <
clean_database.sql
```

Python Libraries

```
sudo apt-get install python3-setuptools python-setuptools
sudo apt-get install gcc libpq-dev -y
sudo apt-get install python-dev  python-pip -y
sudo apt-get install python3-dev python3-pip python3-venv
sudo apt install libmariadb3 libmariadb-dev
python3-wheel -y
sudo apt install rustc
sudo apt install libgl1-mesa-glx
pip install wheel absl-py paho-mqtt mariadb PyJWT
cryptography opencv-contrib-python opencv-python
```

Start Google Cloud Setup

Bevor die Struktur genutzt werden kann sollten nochmal alle Config Files in data Directory beobachtet werden:

django_config.json:

username = superuser
password = Passwort vom superuser
ip = HTTPS Namensdomain (z.b. https://smartcam.ddns.net)
port = (nicht genutzt bei https, automatisch 443)

mariadb_config.json:

user = user aus Google SQL Datenbank
password = Passwort dementsprechend
host = IP Adresse der Google SQL Datenbank

Um nun die Google Infrastruktur zu nutzen muss in der VM das Script vm_cloud_google.py ausgeführt werden. Nun werden alle Nachrichten aus dem Pub Sub Thema gepulled.

In dieser Anleitung wurde ein Gerät registriert *cam-google.* Dafür muss auf den Raspberry das Skript cam_google.py ausgeführt werden. Aber auch hier nochmals überprüfen, ob die Konfiguration richtig ist:

```
FLAGS = flags.FLAGS
flags.DEFINE_string('project_id', 'smart-cam-ba', 'project id of google
cloud platform')
flags.DEFINE_string('cloud_region', 'europe-west1', 'cloud region of the
google IoT Core registry')
flags.DEFINE_string('registry_id', 'cam', 'registry id in the google IoT
Core')
flags.DEFINE_string('device_id', 'cam-google', 'device id that is already
added in the corresponding registry id')
```

Die VM empfängt nun Json Objekte:

Auf dem Raspberry wird das 416x416 große Bild gezeigt (ohne Erkennung, geschieht ja in der VM)

Über den Webserver sind die Daten dann auch über die Authentifizierung des Superusers erreichbar. Emails (über Gmail) wurden auch sofort verschickt.