User Guide

Version 1.1

18 Jan 2020

BY TENSORFACTORY

DISCLAIMER OF WARRANTIES

* **TENSORFACTORY** MEANS TENSORFACTORY ENTERPRISE AND ITS OWNER.

** **PFCS** MEANS PIFACECAM-CLIENT, PIFACECAM-IDSERVER AND PIFACECAM-APP

THIS DOCUMENT IS PROVIDED "AS IS" WITH NO WARRANTY OR WHATSOEVER, INCLUDING BUT NOT LIMITED TO ANY WARRANTY OF MERCHANTABILITY, NON-INFRINGEMENT, FITNESS FOR ANY PARTICULAR PURPOSE, OR ANY WARRANTY OTHERWISE ARISING OUT OF ANY PROPOSAL, SPECIFICATION OR SAMPLE.

YOU EXPRESSLY UNDERSTAND AND AGREE THAT YOUR USE OF THE PFCS** IS AT YOUR SOLE RISK AND THAT THE APPLICATION IS PROVIDED "AS IS" AND "AS AVAILABLE" WITHOUT WARRANTY OF ANY KIND FROM TENSORFACTORY*.

YOUR USE OF THE PFCS IS AT YOUR OWN DISCRETION AND RISK AND YOU ARE SOLELY RESPONSIBLE FOR ANY DAMAGE TO YOUR COMPUTER SYSTEM OR OTHER DEVICE OR LOSS OF DATA THAT RESULTS FROM SUCH USE.

TENSORFACTORY FURTHER EXPRESSLY DISCLAIMS ALL WARRANTIES AND CONDITIONS OF ANY KIND, WHETHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO THE IMPLIED WARRANTIES AND CONDITIONS OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NON-INFRINGEMENT.

YOU EXPRESSLY UNDERSTAND AND AGREE THAT TENSORFACTORY SHALL NOT BE LIABLE TO YOU UNDER ANY THEORY OF LIABILITY FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, CONSEQUENTIAL OR EXEMPLARY DAMAGES THAT MAY BE INCURRED BY YOU, INCLUDING ANY LOSS OF DATA, LOSS OF PROFITS, LOSS OF BUSINESS, INTERRUPTION OF BUSINESS, WHETHER OR NOT TENSORFACTORY HAVE BEEN ADVISED OF OR SHOULD HAVE BEEN AWARE OF THE POSSIBILITY OF ANY SUCH LOSSES ARISING.

SPECIFICATIONS AND INFORMATION CONTAINED IN THIS DOCUMENT ARE PROVIDED FOR INFORMATION USE ONLY, AND ARE SUBJECT TO CHANGE AT ANY TIME WITHOUT NOTICE, AND SHOULD NOT BE CONSTRUED AS COMMITMENT BY TENSORFACTORY. TENSORFACTORY ASSUMES NO RESPONSIBILITY OR LIABILITY FOR ANY ERRORS OR INACCURACIES THAT MAY APPEAR IN THIS DOCUMENT.

Contents:

1.0 Overview

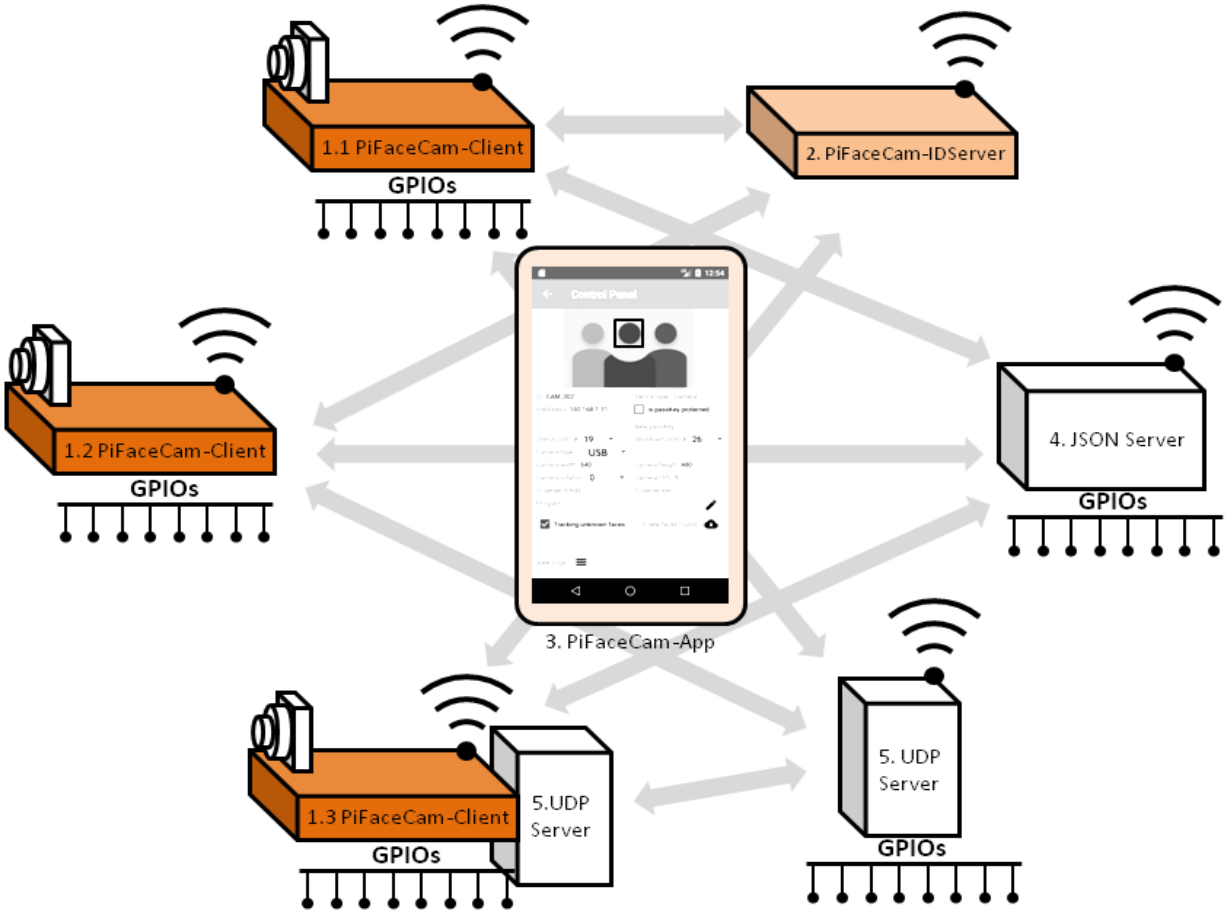


Figure 1.0

1.1 PiFaceCam system

PiFaceCam system uses deep neural network for accurate facial recognition. It is designed to run efficiently in Raspberry Pi*.

PiFaceCam currently consists of 3 components (which collectively known as PiFaceCam system).

1. **PiFaceCam-Client**: Runs in Raspberry Pi* with camera attached. It performs facial recognition, carries out instructions as per the uploaded program and also capable to conveys data to other components.
2. **PiFaceCam-IDServer** (Optional): Runs in Raspberry Pi*. It allows synchronization for face ids among multiple PiFaceCam-Clients.

* Tested on Pi3 Model B+ and Pi4 Model B 4GB.

3. **PiFaceCam-App**: Runs in Android devices (currently only support android devices). It is for managing and controls all PiFaceCam devices connected through local network.
4. JSON Server (Optional): A server that receive information and instruction from PiFaceCam-Clients in JSON formats (please see the examples of implementation section for better understanding).
5. UDP Server (Optional): In event of the built-in functions in PiFaceCam-Clients are not meeting your requirements, PiFaceCam-Client also exposes raw data via UDP packet. You can create a UDP server to receive the raw data and implement you own functions (please see an example in the examples of implementation section).

1.2 Features

- Facial recognition based on known face ids and carry instruction per scripts.
- Auto tracking and assigning temporary ids for unknown faces (which user can later review and assign proper ids).
- Comprehensive tools for collecting, editing and organizing face ids.
- 10 levels of detection sensitivity adjustment to accommodate all types of application conditions.
- Simple scripting concept for automation.
- User-friendly tools to create, edit and manage scripts/programs.
- Remote monitor, manage and control multiple devices.
- Built-in functions to control GPIOs, communicate with JSON-server and sending emails.
- Expose raw information (image, detected face-ids and face bounding boxes) via UDP socket for unrestricted capability extension.

2.0 PiFaceCam-Client setup

2.1 Hardware requirement

PiFaceCam-Client required Raspberry Pi 3 / 4 (Tested on Pi3 Model B+ and Pi4 Model B 4GB) to run. As facial-recognition is computationally heavy, proper heat management is required. Standard cooling fan with heat sink on CPU was tested to be sufficient in both Pi3 Model B+ and Pi4 Model B.

By default, GPIO 19 and 26 are reserved for status and shutdown purpose. GPIO 19 should be connected to a LED via a resistor. The LED will blinks during system loading and on continuously when system is ready. LED blinks indefinitely signify error has occurred. GPIO 26 will trigger system shutdown when connect to high.

2.2 Software

2.2.1 Getting system ready of PiFaceCam-Client

This PiFaceCam-Client was developed for Raspbian Buster with desktop (Kernel version: 4.19). You will also need to install the following supporting libraries.

1. Pyhton3 (develop in version 3.7.3)
2. Tensorflow 1.X (develop in version 1.15)
3. OpenCV for python (develop in version 4.2.0)
4. Scikit-learn (develop in version 0.22)
5. Pycryptodomex (develop in version 3.9.4)

2.2.2 Installing PiFaceCam-Client

After you have installed all the necessary libraries, copy the whole "pifacecam_client" folder from github (https://github.com/tensorfactory/PiFaceCam) into the home directory.

If you don't want to go through all the trouble of installing everything yourself, we are sharing out SD Card's image file at github as well. Please take note that we are using 16GB SD Card, therefore you will need a large card to flash the image.

After flashing your SD Card, remember to reclaim the missing disk space by running raspi-config -> Advanced Options -> Expand Filesystem Ensures that all of the SD card storage is available.

PiFaceCam_Client's files are placed in "home/pi" folder.

2.2.3 Naming your device

The default device id is "CAM001". The first thing you need to do is to give your device a unique name (Maximum 15 characters) by placing it in the "id.txt" file or setting it from PiFaceCam-App (Note: Value from PiFaceCam-App supersedes value in id.txt).

2.2.4 Main program

In the folder, you will see "PiFaceCam_Client_X_X_X.so". This is the PiFaceCam_Client program. To use it, first import "PiFaceCam_Client_X_X_X" to your project and call the "run_client(UDP_server_ipaddress, UDP_server_receiving_port)" method.

"run_client" takes 2 parameters. The ipaddress and port number of the UDP server used to receive raw data from "PiFaceCam_Client". In this example, we are not using this feature, we therefore put empty string for ipaddress and -1 for port number to disable it.

Once started, "run_client" will not return until user terminates it by setting the "shutdown" GPIO pin to high. We can make use of this blocking characteristic as shutdown mechanism for raspberry pi when "PiFaceCam_Client" exit, as in below example.

```
import PiFaceCam_Client_1_3_0
import os

if __name__ == "__main__":

    PiFaceCam_Client_1_3_0.run_client("", -1)
    os.system("sudo shutdown -h now")
```

(wrapper_without_UDP_server.py)

2.2.5 Using UDP server to retrieve raw information

```
import PiFaceCam_Client_1_3_0
import multiprocessing as mp
import socket
from struct import unpack
import pickle
import cv2
import numpy as np

def UDP_server(port):
    _server_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    _server_socket.bind(("", port))

    while True:
        # --------------Handling Receiving of packet----------------------
        bytes_AddressPort_Pair = _server_socket.recvfrom(65000)
        client_data = bytes_AddressPort_Pair[0]
        client_addressPort = bytes_AddressPort_Pair[1]

        data_length_bytes = client_data[:8]
        (data_length,) = unpack('>Q', data_length_bytes)
```

```
        data_bytes = client_data[8: 8 + data_length]
        (image_jpeg_np, bbox_minXminYmaxXmaxY_np, faceID_list) = pickle.loads (data_bytes)

        image_np = cv2.imdecode(image_jpeg_np, cv2.IMREAD_UNCHANGED)
        print("Packet length:", data_length)
        print("Shape of received image is ", np.shape(image_np))

        if (len(faceID_list) >0):

            for bbox_idx, face_id in enumerate(faceID_list):
                print("Bounding box coordinates for " +
                      face_id + ": minX={:.0f}, minY={:.0f}, maxX={:.0f}, maxY={:.0f}".
                      format(bbox_minXminYmaxXmaxY_np[bbox_idx, 0],
                             bbox_minXminYmaxXmaxY_np[bbox_idx, 1],
                             bbox_minXminYmaxXmaxY_np[bbox_idx, 2],
                             bbox_minXminYmaxXmaxY_np[bbox_idx, 3]))


if __name__ == "__main__":
    port = 60000
    UDP_server_process = mp.Process(target=UDP_server, args=(60000,))

    UDP_server_process.start()
    PiFaceCam_Client_1_3_0.run_client("localhost", port)
```

(wrapper_with_UDP_server.py)

In this example, a UDP server is running on a sub process on the same device which receives packets of raw information from "PiFaceCam_Client". After starting the UDP server process, the ipaddress and port number of this server were pass to "PiFaceCam_Client" as parameters of the "run_client" method.

The first 8 bytes (big-endian byte order, unsigned long long) represents the length of data-body, followed by the data-body. To extract the raw data, the data-body is unpickled into (image_jpeg_np, bbox_minXminYmaxXmaxY_np, faceID_list) tuple.

"image_jpeg_np" is jpeg-encoded and a numpy represents the image can be obtained using cv2.imdecode function (shown above). "bbox_minXminYmaxXmaxY_np" is a numpy where each row represents the [min_X, min_Y, max_X, max_Y] coordinates of a face's bounding box. "faceID_list" is the list of face IDs corresponding to the bounding boxes.

3.0 PiFaceCam-IDServer setup

3.1 Hardware requirement

PiFaceCam-IDServer required Raspberry Pi 3 / 4 (Tested on Pi3 Model B+ and Pi4 Model B 4GB) to run. By default, GPIO 19 and 26 are reserved for status and shutdown purpose. GPIO 19 should be connected to a LED via a resistor. The LED will blinks during system

loading and ON continuously when system is ready. LED blinks indefinitely signify error has occurred. GPIO 26 will trigger system shutdown when connect to high.

3.2 Software

3.2.1 Getting system ready of PiFaceCam-IDServer

This PiFaceCam- IDServer was developed for Raspbian Buster with desktop (Kernel version: 4.19). You will also need to install the following supporting libraries.

1. Pyhton3 (develop in version 3.7.3)
2. Scikit-learn (develop in version 0.22)
3. Pycryptodomex (develop in version 3.9.4)

3.2.2 Installing PiFaceCam- IDServer

After you have installed all the necessary libraries, copy the whole "pifacecam_idserver" folder from github (https://github.com/tensorfactory/PiFaceCam) into the home directory.

Similarly, all these files are included in the SD Card image file that we are sharing out at github. PiFaceCam_IDServer's files are placed in "home/pi" folder.

3.2.3 Naming your device

The default device id is "SERVER01". The first thing you should do is to give your device a unique name (Maximum 15 characters) by placing it in the "id.txt" file or setting it from PiFaceCam-App (Note: Value from PiFaceCam-App supersedes value in id.txt).

3.2.4 Main program

In the folder, you will see "PiFaceCam_IDServer_X_X_X.so" which is PiFaceCam_IDServer program. To use it, first import "PiFaceCam_IDServer_X_X_X" to your project and call the " run_server () method.

```
import PiFaceCam_IDServer_1_3_0
import os

if __name__ == "__main__":

    PiFaceCam_IDServer_1_3_0.run_server()
    os.system("sudo shutdown -h now")
```

(wrapper.py)

4.0 PiFaceCam-App

4.1 Requirements

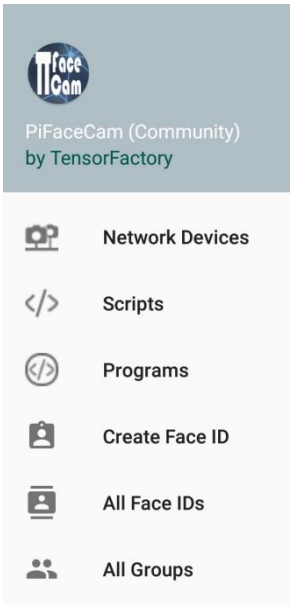
Requires Android OS Marshmallow and above. The device will also require network connection to communicate with PiFaceCam-Client or PiFaceCam-IDServer and camera if you want to create face ids directly from image captured.

4.2 Versions

There are 2 versions of PiFaceCam-App. PiFaceCam (Community) and PiFaceCam (Professional). The community version is free, has all the features but the number of face ids it can store or handle limited to 20. Both versions can be obtained from Google Play.

4.3 Overview

From the main page, you will be able to navigate to all the features.



1) Network Devices:
   a) Scan/access all connected PiFaceCam devices.
   b) Change settings of connected devices.
   c) Load face ids and program to PiFaceCam-Clients.

2) Scripts:
   a) List and manage of all scripts.
   b) Create and edit scripts.
   c) Export/import scripts to file.

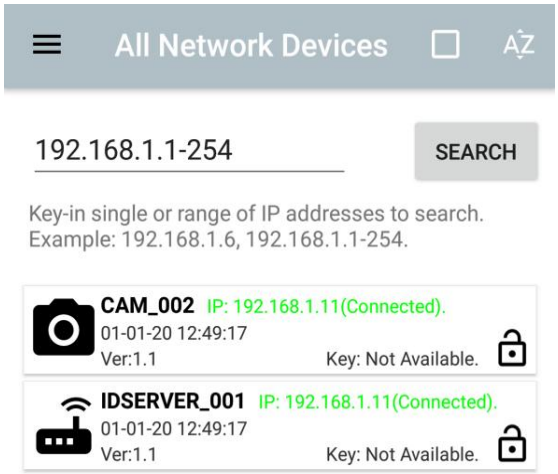
3) Programs:
   a) List and manage of all programs. (which are combinations of one or more scripts).
   b) Create and edit programs
   c) Export/import programs to file.

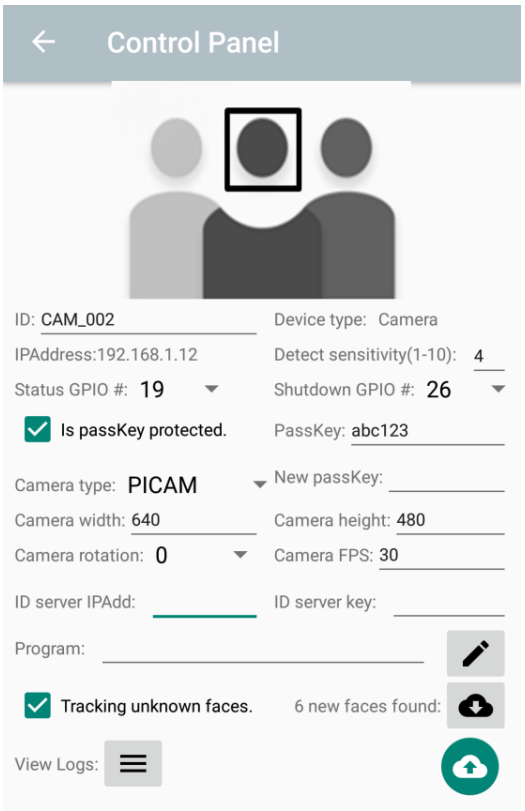
4) Create Face ID:
   a) Create face ids from camera or image gallery.
   b) Edit face ids.

5) All Face IDs:
   a) Shown all available face ids.
   b) Export / import face ids to file.

6) All Groups:
   a) List and manage of all groups. (which are combinations of one or more face ids).
   b) Export / import groups to file.

4.3.1 Network Devices





4.3.1.1 Search for devices

You can search all PiFaceCam devices that are connected to the local network. Or, if you already know the ipaddress of the device, you can key-in the exact address.

All connected PiFaceCam devices will be listed here. You can access the device by clicking on the list. The padlock symbol will indicate if the connected device is passkey locked which will require passkey to access.

Once clicked, you will reach the control panel. If the connected device is a camera, you will able to view live feeds from the camera.

4.3.1.2 Setting device id

From here, you can change the device ID. The maximum character for device ID is 15 (extra characters will be trimmed).

4.3.1.3 Face detection sensitivity level

You can adjust the detection sensitivity level. The default detection sensitivity level is 4. A higher number means a higher chance of a person is detected but also a higher chance of wrong identification. Choose a lower sensitivity level if low identification error is required and vice versa.

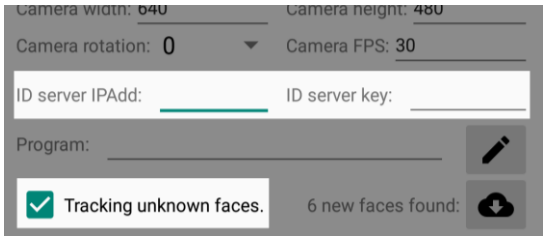
4.3.1.4 Status LED and device shutdown pin

You can set the GPIO pins number for device status and device shutdown. The device status pin will be connected to a LED via a resistor. Blinking indicates loading and continuous on indicates device is ready. The shutdown pin will trigger device to shutdown when connected to high.

4.3.1.5 Camera settings

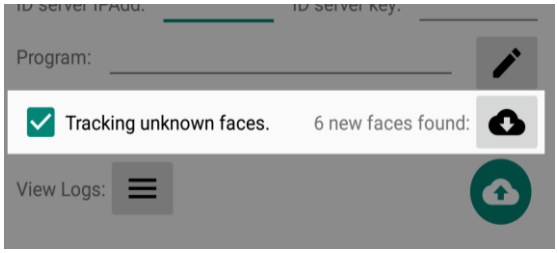
For camera types, you have options of USB or PICAM (Pi-Camera). You can change the camera setting here. The camera rotation (0˚, 90˚, 180˚, 270˚) and the FPS will only have effect if Pi-Camera is used. For Pi-Camera, we recommend using 30FPS, camera width 640 and height 480.

4.3.1.6 Tracking of unknown faces

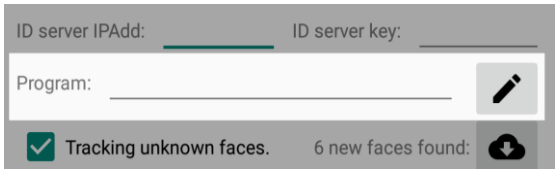


PiFaceCam has a feature to track unknown faces. If turned ON, it will start tracking all unknown faces and if there are sufficient good images, it will assigned a "local ID" to each person.

However, if you have multiple cameras running and each camera may assign a different id for the same person. To prevent this, you can setup a PiFaceCam-IDServer to synchronize the ids among these cameras. Once you have setup the PiFaceCam-IDServer, you need to put the ID-Server's Ipaddress and passkey here.



You can see the number of new unknown faces found by this connected device and download them to your face ID list where you can further edit them.



4.3.1.7 Program

Click on the "Pencil" icon to select a program you want to run on the connected device

4.3.1.8 View log

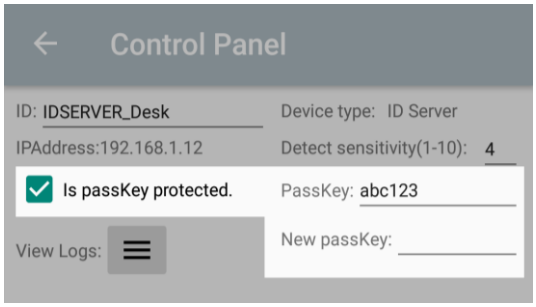
You can access the error/warning logging of remote device here.

4.3.1.9 Setting upload

Click this button to upload the new setting to the connected device.

Note: During setting upload, all face ids from your face ids list will also be uploaded to the connected device.

4.3.1.10 ID-Server



For connected ID-Server it is required to set it to passkey protected in order to function as an ID-Server. The detection sensitivity level should be set the same as all other PiFaceCam devices.

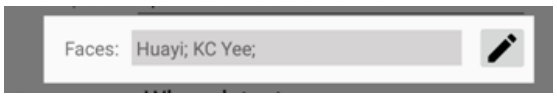
4.3.2 All Scripts

You will see all your created scripts (if available) listed here. Click on "+" button to add scripts. You have option to import from a ".srp" file or create from scratch.

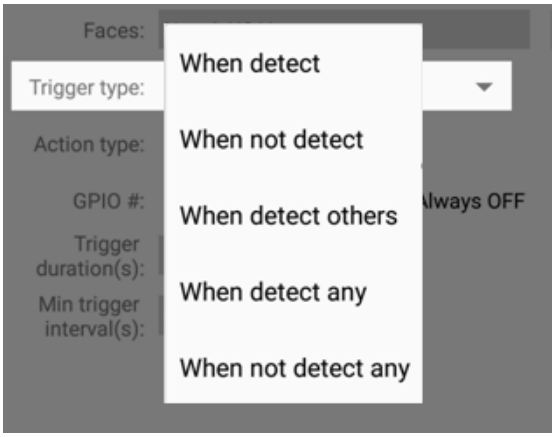
You can export any of the scripts to file by clicking on the share button at the top.

4.3.2.1 Creating a new script

To create a new script, first give you script a meaningful name and description.



Next Click on "pencil" icon to select face-ids that you will be using in this script.



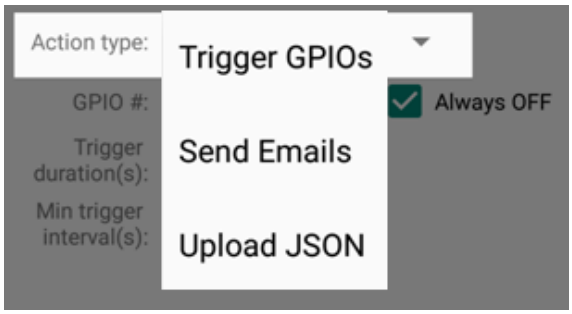
Select a trigger type.

1) Action is triggered when any of the selected faces are detected,
2) action is triggered when the none of selected faces are detected,

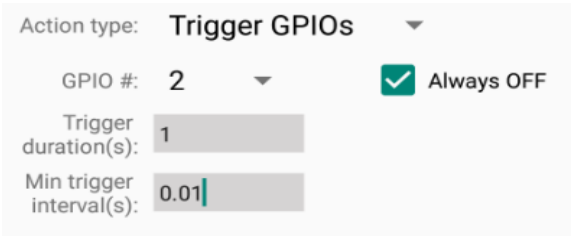
3) action is triggered when any faces other than those that were selected are detected,

4) action will triggered when detect any faces and

5) action will be triggered when no face is detected.

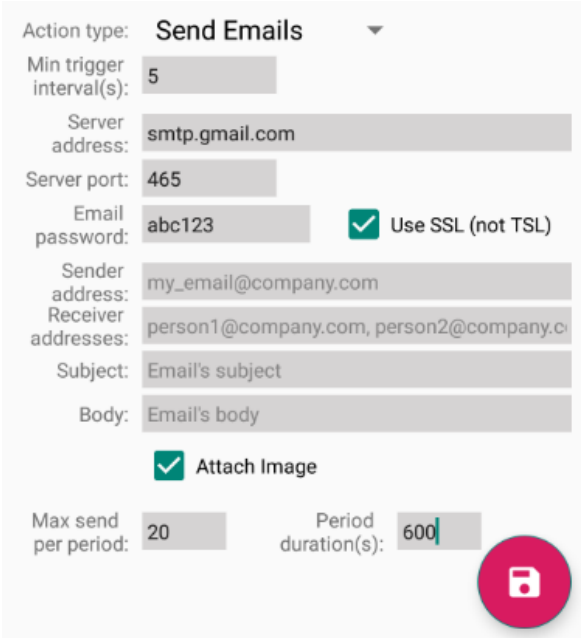


Next is to select the action type. You have options of triggering a GPIO, sending an email or upload face id list and a key value to a JSON server.



For GPIO, you can select any GPIO pin to trigger but it has to be different from the status and shutdown pin. You can also select always OFF or always ON mode. In always OFF mode, the pin will be always at low state until it is trigger and vice versa.

You can set the trigger duration and the minimum time interval between triggers "Min trigger interval(s)". However, as with other temporal settings, their accuracies are not very reliable and shall not be depended on for any application.

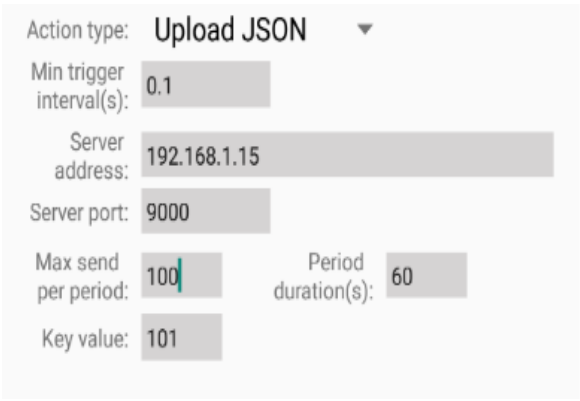


For action type of sending email through a smtp server. You have to provide the necessary smtp server information.

For security protocol, you can choose between SSL and TSL. Uncheck the "Use SSL" checkbox will select TSL protocol.

You have the option to provide a short email subject and body, and to include an attachment of the image that triggers this action.

In this example (left), to prevent multiple emails send when this script is triggered continuously, we set the min trigger interval to 5 seconds, meaning after the first email sent, it will wait for 5 seconds before the next triggering can happen. Also we set the max send per period to 20 and the period to 600 seconds (10 minutes) which means, in every 10 minutes, the script can only send a maximum of 20 emails. The counter will reset after each 10 minutes period.



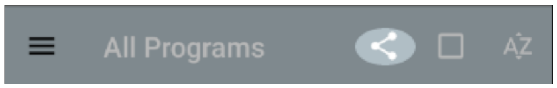
For action type, a JSON packet will be send to the JSON server with the following format.

```
{
        "device_id": device_id,
        "num_of_faces": num_of_faces,
        "faceID_list": faceID_list,
        "key_value": key_value,
        "time_stamp": time_stamp
                                                    }
```

"faceID_list" is list of face ids in the image that trigger this action. "key_value" is an arbitrary integer value to pass over to JSON server for identification. "time_stamp" is the number of seconds that have elapsed since January 1, 1970, 00:00:00 (UTC). Please see an example of implementation in our github folder.

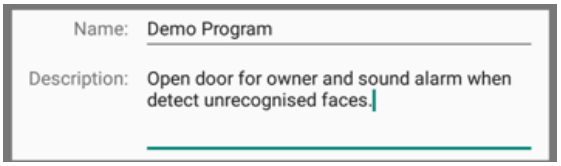
4.3.3 All Programs

You will see all your created programs (if available) listed here. Click on "+" button to add programs. You have option to import from a ".prg" file or create from scratch.



You can export any of the programs to file by clicking on the share button at the top.

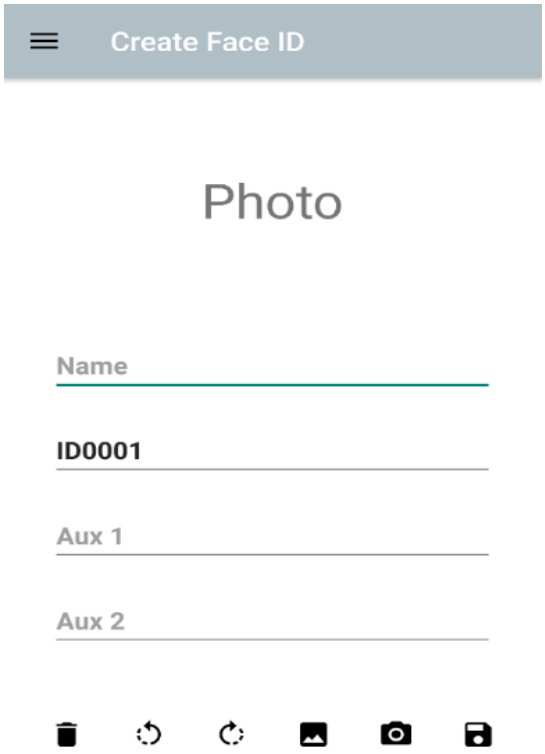
A program is a group of scripts. As each script is a unit of detection-action pair, using program will allow us to implement multiple scripts at once.



To create a new program, first give you program a meaningful name and description.

Next click on "pencil" icon to select scripts that you want to include in this program.

4.3.4 Create Face ID



Click on “Photo” or the gallery or camera button to start import face photos of this person.

Name is compulsory and ID is auto-generated (can be altered).

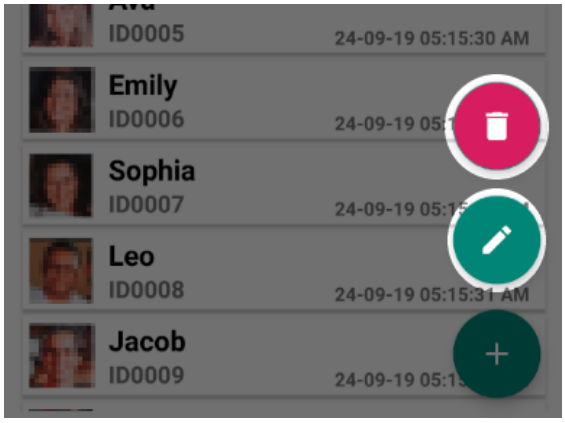
All imported face photos will be shown at the top and rejected photos will have a red cross.

You can delete a selected photo by clicking on the dustbin. (Long click will delete all photos).

You can also rotate the selected photo by clicking at the rotation button at the bottom.

Note: If you import more than 5 photos, PiFaceCam will choose the best 5 photos to used (saved).

4.3.5 All Face IDs



You will see all your created scripts (if available) listed here. Click on “+” button to add face ids. You have option to import from a “.fig” file or create from scratch.

You can export any of the face-ids to file by clicking on the share button at the top.

Long click on any face id to start selection.

Click on any item will open edit dialog.

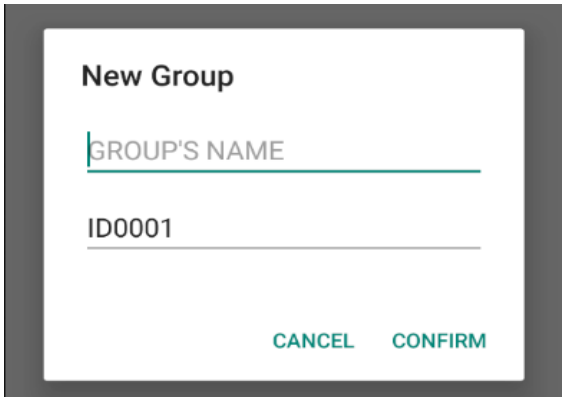
You can also delete or edit selected face id by clicking on the “dustbin” or “pencil” button.

4.3.6 All Groups

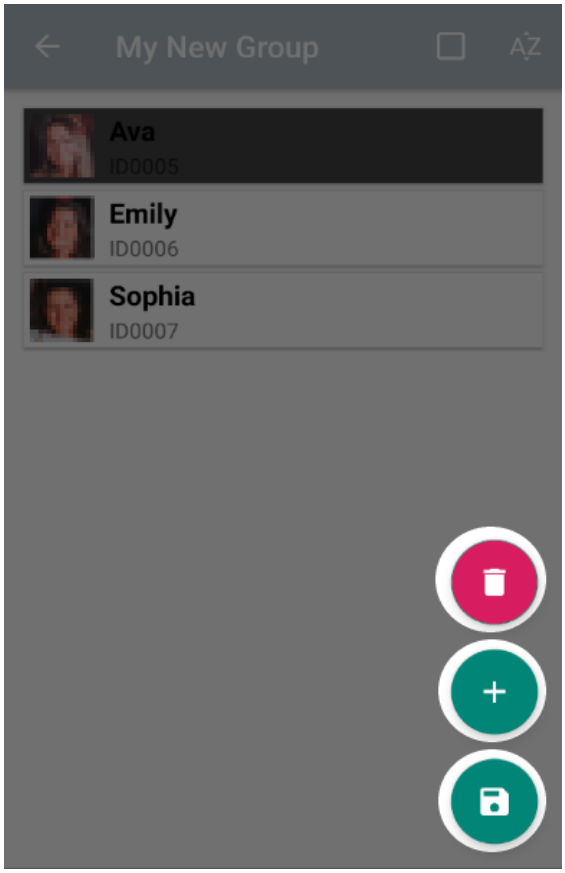
Group is a collection of face ids. This is to help us organize our face-ids.

You will see all your created groups (if available) listed here. Click on “+” button to create new group. You have option to import from a “.grp” file or create from scratch.

You can export any of the groups to file by clicking on the share button at the top.



If you choose to create new one, you will be prompted for a name.



Start adding face IDs as member of this group by clicking on the "+" button.

Long click on any face id to start selection.

You can delete selected IDs or continue to add more.

Remember to save changes before leaving.

5.0 Implementation Examples

5.1 Case 1: Simple auto door unlocking and alerting (buzzer and email).

In this example, we would like to automate our office's door locking so that only authorized personnels can enter and would like to receive an email if detect unauthorized person trying to enter. We will be using a Raspberry Pi (3B+ or 4B) with PiCamera attached.

Requirements:

- Keeping list of face ids of authorised personnels.
- When detect any of the authorised personnels, trigger door unlock for 3 seconds by setting GPIO 4 to high for 3 second.
- At the same time we would like to have a buzz of 1 second whenever door unlock is triggered. This is done by setting GPIO 5 to high for 1 second.
- When there an unauthorised person trying to enter, we would like our HR department to be alerted through email (hr_manager@company.com). The email should include an image of the person face.

Note: It is assumed PiFaceCam-Client is properly installed as per instruction in section (2.0) and the user is "pi". Also, the Raspberry Pi has and will auto connect to the local network.

Step 1) Setting up the PiFaceCam-Client to auto-start everytime we power ON the Raspberry Pi.

a) Create a script file “launcher.sh” with below lines

```
#!/bin/sh
sleep 10
cd /
cd home/pi/pifacecam_client
sudo python3 wrapper_without_UDP_server.py
```

b) Make the launcher script an executable, with command "chmod 755 launcher.sh"

c) Make a directory for the any errors in crontab to go

Navigate back to your home directory:
cd
Create a logs directory:
mkdir logs

d) Add to Your Crontab. To brings up a crontab window, run

sudo crontab -e

e) Now, enter the line:

@reboot sh /home/pi/pifacecam_client /launcher.sh >/home/pi/logs/cronlog 2>&1

Step 2) Setting up the face-ids of authorised personnels.

a) Following instructions from section 4.3.4 , create a face id for each of the authorised persons.

b) Create a group for these ids (name it something like "department A") following section 4.3.6.

Step 3) Creating scripts.

There are three detection-actions in this implementation. 1) Detect authorised personnels, unlock door, 2) detect authorised personnels sound buzz and 3) detect un-authorised persons, send email. We will create 3 scripts, one for each of the detecion-action as per section 4.3.2.

i) Script 1:

Name: Unlock door (Department A)
Description: ON GPIO 4 for 3 seconds when detect authorised personnels.
Faces: Select face ids from group "department A"
Trigger type: Select "When detect"
Action type: Select "Trigger GPIOs"
GPIO#: Select number 4
Check "Always OFF"
Trigger duration(s): 3
Min trigger intervals(s): 0.01

ii) Script 2:

Name: Sound buzzer (Department A)
Description: ON GPIO 5 for 1 second when detect authorised personnels.
Faces: Select face ids from group "department A"
Trigger type: Select "When detect"
Action type: Select "Trigger GPIOs"
GPIO#: Select number 5
Check "Always OFF"
Trigger duration(s): 1
Min trigger intervals(s): 0.01

iii) Script 3:

Name: Send email (Department A)
Description: Send email to HR manager when detect unauthorised persons.
Faces: Select face ids from group "department A"
Trigger type: Select "When detect others" (To detect unauthorised persons)
Action type: Select "Send Emails"
Min trigger intervals(s): 5 (Allow some gaps for smtp server to response)
Server address: smtp.gmail.com (Using google smtp service)
Server port: 587
Email password: abcd1234
Sender address: departmentA@gmail.com
Receiver addresses: hr_manager@company.com
Subject: Unauthorised person detection alert.
Body: Unauthorised person detection at department A. Please see attachment.
Check "Attach Image"

Max send per period: 20
Period duration(s): 600 (Every 10 minutes only allow maximum 20 emails.

Step 4) Creating program (following instructions from section 4.3.3)

Name: Door Security System (Department A)
Description: Manage door unlock and unauthorised persons detection.
Scripts: Select scripts
"Unlock door (Department A)",
" Sound buzzer (Department A)" and
" Send email (Department A)"

Step 5) Upload program to PiFaceCam-Client device.

a) Search the device in all network devices list, click on the device to open the control panel.
b) The default name is "CAM001" and we would like to rename it to "CAM_DEPT_A".
c) Since this is for door entrance and we would like to reduce the chances of wrong identification as much as possible and since the camera is mount in front of the door, the camera will have good angle of the person's face we can reduce the sensitivity level from 4 to 3.
d) By default, the PiFaceCam-Client device is not passkey protected, we will on the passkey protection and provide a "New passkey".
(Note: Once ON, the passkey will be automatically stored in this PiFaceCam-App. In event if the passkey was erased, you can key-in the password manually at the "PassKey" edit box.)
e) We are using Pi Camera, so we choose PICAM for camera type and use the recommended 640x480 camera size and 30 FPS.
f) Click on the pencil icon next to program and select "Door Security System (Department A)".
g) Click the green upload button at the bottom right to upload the setting and program to the connected PiFaceCam-Client. The PiFaceCam-Client will start executing the instructions in the program.

5.2 Case 2: Remote controlling and displaying of ids through JSON server.

Expanding from case 1, we would like to have our security office to be alerted whenever the camera detects a person. If it is an authorized person, green LED should light up and red LED if it is an unauthorized person. We also want the id of all the persons in the camera to be displayed.

ESP8266 JSON server

We will use a ESP8266 module (ESP07) as a JSON server which will receive information from PiFaceCam-Client. It will control 2 LEDs and display ids on a 20x4 hd44780 LCD (with I2C). The Arduino sketch for this can be found at https://github.com/tensorfactory/PiFaceCam/tree/master/ESP8266_JSON_server

This sketch uses

- "ArduinoJson" library by Benoit Blanchon for handling of JSON.
- "hd44780" library by Bill Perry to control LCD via I2C.
- "WiFiManager" library by Tzapu to manage network credentials (SSID and password).
- "TimeLib" library by Paul Stoffregen to convert Unix epoch time to standard format.

PiFaceCam-Client

As in case 1, we will create 2 scripts for this. One to send a key-value 1 to JSON server when detect authorized personnels and the other to send a key-value 2 when detect unauthorized personnels. ESP8266 server will receive a JSON packet with all these information and turn ON/OFF the corresponding LED as per the key_value. It will also display all (max 4) ids on the 20 x 4 matrix LCD.

```
{       "device_id": device_id,
        "num_of_faces": num_of_faces,
        "faceID_list": faceID_list,
        "key_value": key_value,
        "time_stamp": time_stamp    }
```

i) Script 1:

Name: JSON key-value 1 (Department A)
Description: Send key-value 1 to JSON server when detect authorised personnels.
Faces: Select face ids from group "department A"
Trigger type: Select "When detect"
Action type: Select "Upload JSON"
Min trigger interval(s): 0.1 (A small value as response of ESP8266 server is relatively fast).
Server address: IP address of the ESP8266 server.
Server port: Port number of the ESP8266 server.
Max send per period: 100 (A largevalue as response of ESP8266 server is relatively fast).
Period duration(s):1
Key value: 1

ii) Script 2:

Name: JSON key-value 2 (Department A)

Description: Send key-value 2 to JSON server when detect unauthorised personnels.
Faces: Select face ids from group "department A"
Trigger type: Select "When detect others" (To detect unauthorised persons)
Action type: Select "Upload JSON"
Min trigger interval(s): 0.1
Server address: IP address of the ESP8266 server.
Server port: Port number of the ESP8266 server.
Max send per period: 100
Period duration(s):1
Key value: 2

Append these 2 new scripts to the program in case 1, upload the program to PiFaceCam-client, it will start sending JSON packets to the ESP8266 server.

5.1 Case 3: Tracking and counting total number of visitors using multiple cameras.

In this case, the ids of visitors were not available prior to detection. We can turn on the "tracking of unknown faces" feature in PiFaceCam-client (section 4.3.1.6). Once turned on, each PiFaceCam-client will start tracking all unknown faces. If sufficient good angle images of a face are available, it will assign an id (local id) for this face and temporary (until power off) storing it. These local face ids can be later downloaded to PiFaceCam-App for analysis.

However, because of we have multiple cameras at the area; the same face may appear in more than one camera. To resolve this, we can set up a PiFaceCam-IDServer to coordinate issuing of face ids. Each PiFaceCam-Client will request a global id from this IDServer. This will ensure all PiFaceCam-Clients use the same id for the same face.

To do this, we need to set up PiFaceCam-IDServer as per section 4.3.1.10 and upload the IP address and passkey of this ID Server to all PiFaceCam-Client (section 4.3.1.6).