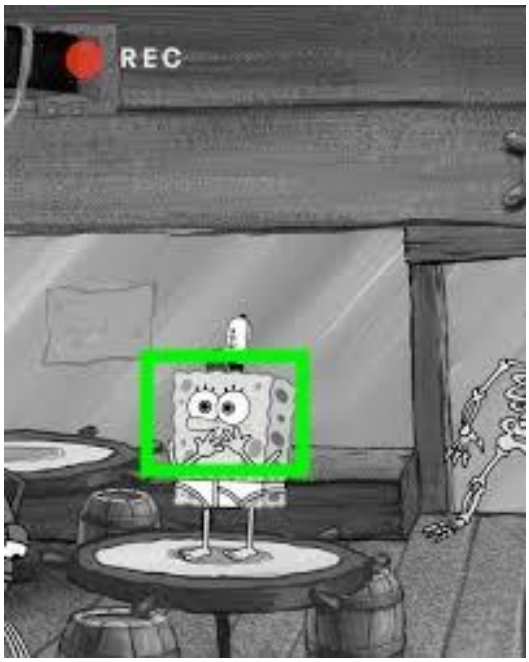
CMPT 433

How to Use OpenCV for Real-Time Face Detection with BeagleY-AI



Rajbir Singh Bains
Tej Singh Pooni
Sukhmanpreet Singh
Gurkeerat Singh Bouhgan

Prerequisites:

- HD Camera using USB connection (Recommended using no more than 5V supported devices) [1]

1. Setting Up Environment

1.1 Install tools on the host

Install cross-compiler for ARM:
sudo apt install g++-aarch64-linux-gnu

Install OpenCV Development Libraries for ARM
sudo apt-get install libopencv-dev:arm64

(**TROUBLESHOOT**) Update PKG_CONFIG_PATH: If CMake cannot find OpenCV configuration files, update the PKG_CONFIG_PATH
export PKG_CONFIG_PATH=/usr/lib/x86_64-linux-gnu/pkgconfig:$PKG_CONFIG_PATH

Download the Haar Cascade file [2], used for face detection, in the resources folder
wget https://github.com/opencv/opencv/raw/master/data/haarcascades/haarcascade_frontalface_default.xml
mv haarcascade_frontface_default.xml /path/to/project/resources/

1.2 Setup host environment (Python)

Create a new Conda environment named *faceenv* and install *dlib*
conda create -n faceenv python=3.10 -y
conda activate faceenv
conda install -c conda-forge dlib -y

Install necessary libraries
pip install face_recognition python-dotenv requests

(TROUBLESHOOT) May encounter an “illegal instruction” error. You need to reinstall *dlib* without precompiled binaries
pip uninstall dlib
pip install --no-binary :all: dlib

1.3 Install tools on target

Install OpenCV Development Libraries
sudo apt-get install libopencv-dev

Verify installation
python3 -c “import cv2; print(cv2._version_)”

2. Connecting and Configuring the Camera (All commands done of Target)

2.1 Connect camera to one of the USB ports on the BeagleY-AI

Verify that the camera is detected using:
*ls /dev/video**

**You should see a device like /dev/video0 or dev/video3.*
**Note: Might need to compare the dir to when USB is not connected vs when it is to know which /dev/video* directory corresponds to the camera.*

When Camera is not plugged in:

```
raj@rajbirb-beagle:~$ ls /dev/video*
/dev/video0  /dev/video1  /dev/video2
```

When Camera is plugged in:

```
raj@rajbirb-beagle:~$ ls /dev/video*
/dev/video0  /dev/video1  /dev/video2  /dev/video3  /dev/video4
```

Here we can see /dev/video3 and /dev/video4 correspond to our camera device

Troubleshoot:

If dev directory for new camera does not show, verify USB devices using : *lsusb*

```
raj@rajbirb-beagle:~$ lsusb
Bus 002 Device 002: ID 0451:8140 Texas Instruments, Inc. TUSB8041 4-Port Hub
Bus 002 Device 001: ID 1d6b:0003 Linux Foundation 3.0 root hub
Bus 001 Device 008: ID fefe:4321 Ruision UVC Camera
Bus 001 Device 002: ID 0451:8142 Texas Instruments, Inc. TUSB8041 4-Port Hub
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
```

Note: May need to run *sudo apt install usbutils* if command is not found

2.2 Camera is initialized by using OpenCV's VideoCapture class. The path obtained above (e.g. /dev/video3) is passed to the constructor. Set the resolution and format using OpenCV properties:

```
cv::VideoCapture cap("/dev/video0", cv::CAP_V4L2);
if (!cap.isOpened()) {
    std::cerr << "Error: Could not open the camera!" << std::endl;
    return -1;
}
cap.set(cv::CAP_PROP_FRAME_WIDTH, 1280);
cap.set(cv::CAP_PROP_FRAME_HEIGHT, 720);
```

**settings to capture high-quality frames for facial detection.*

Note: configure OpenCV to your preference in regards to directory structure. This guide is concerned with how we used OpenCV alongside our camera and BeagleBoard

3. Setting up a Python environment for Image Processing.
**refer to guide.txt*

3.1 This step is necessary to set up a Python environment for facial recognition. Ensuring all libraries such as face_recognition.dlib (for facial detection and recognition) and python-dotenv (for managing environment variables) are installed properly.

3.2 Download the appropriate Miniconda installer and run it
Apple Silicon (arm64)
Curl -0 https://repo.anaconda.com/miniconda/Miniconda3-latest-MacOSX-arm64.sh

Intel(x86_64)
Curl -0 https://repo.anaconda.com/miniconda/Miniconda3-latest-MacOSX-x86_64.sh

Linux
Curl -0 https://repo.anaconda.com/miniconda/Miniconda3-latest-Linux-x86_64.sh

```
bash Miniconda3-latest-MacOSX-<your_arch>.sh
```

**replace <your_arch> with arm64 or x86_64*

3.3 Install and verify Conda

source ~/.bashrc
conda --version

3.4 Create and activate a Conda environment. This ensures compatibility with the face_recognition library.

conda create -n faceenv python=3.10 -y && conda activate faceenv && conda install -c conda-forge dlib -y

3.5 Install the required Python libraries for facial recognition

ip install face_recognition python-dotenv requests

4. Processing the Captured Image

4.1 Load the Haar Cascade file for face detection. It is a pre-trained model that OpenCV uses to detect faces. It contains data to identify features such as eyes,nose, and the mouth. It is required by the detectMultiScale function to perform its task.

```
cv::CascadeClassifier face_cascade;
if (!face_cascade.load("resources/haarcascade_frontalface_default.xml")) {
    std::cerr << "Error: Could not load Haar cascade!" << std::endl;
    return -1;
}
```

4.2 Use OpenCV to capture and load the image frame:

```
void * ocv_detect(void* arg) {
    assert(isInitialized);

    // Mark the parameter as unused
    (void)arg;

    while (running) {
        // Capture a frame from the camera
        cap >> frame;

        if (frame.empty()) {
            std::cerr << "Error: Failed to capture frame." << std::endl;
            break;
        }

        // Detect faces in the frame using the Haar cascade
        face_cascade.detectMultiScale(frame, faces, 1.3, 5);

        if (!faces.empty()) {
            std::cout << "Face Found" << std::endl;
        }
    }

    pthread_exit(NULL);
}
```

*cap >> frame is OpenCV shorthand to capture a frame from video stream. Cap is an instance of the class **VideoCapture**, and is stored as a **Mat** object in the **frame** variable.*

***This operation reads the next frame from the camera and stores it in the frame object for further use.*

4.3 Save the captured image and write it to the appropriate directory.

```cpp
int ocv_captureImage() {
    if (frame.empty()) {
        std::cerr << "Error: Could not capture frame from camera!" << std::endl;
        return 0;
    }
    std::srand(std::time(nullptr));
    int random_number = std::rand();
    std::string file_name = "projectResources/pictures/screenshot_" + std::to_string(random_number) + ".jpg";

    if (cv::imwrite(file_name, frame)) {
        const char* imageSaved = "image_saved";
        UDPServer_send(imageSaved);
        std::cout << "Image saved successfully as " << file_name << std::endl;
        return 1;
    } else {
        std::cerr << "Error: Could not save the image!" << std::endl;
        return 0;
    }
}
```

**Random number is appended for uniqueness*
**We want our components to communicate effectively, a UDP message is sent to alert that the image is ready for processing.*

4.4 Process image for facial recognition. After the image is saved, a Python script (process_new_person()) picks up the image for facial recognition. The script below will load the saved image, extract facial encodings, and compare to known encodings in our authorized_personal directory.

```python
def process_new_person():
    known_encodings = load_known_encodings()

    for filename in os.listdir(NEW_PERSON_DIR):
        if filename.lower().endswith((".jpg", ".jpeg", ".png")):
            image_path = os.path.join(NEW_PERSON_DIR, filename)
            unknown_image = face_recognition.load_image_file(image_path)
            unknown_encodings = face_recognition.face_encodings(unknown_image)

            if not unknown_encodings:
                print(f"No face found in {filename}, skipping.")
                os.remove(image_path)
                continue

            unknown_encoding = unknown_encodings[0]
            results = face_recognition.compare_faces(known_encodings, unknown_encoding, tolerance=0.50)

            if any(results):
                print(f"{filename}: Authorized")
                udp_send_message("AUTHORIZED", BEAGLE_IP, BEAGLE_PORT)
            else:
                print(f"{filename}: Unauthorized! Sending to Discord...")
                udp_send_message("UNAUTHORIZED", BEAGLE_IP, BEAGLE_PORT)
                client.send_to_discord(
                    channel_id=CHANNEL_ID,
                    label="Unauthorized",
                    image_path=image_path,
                    additional_info={"Filename": filename, "Detected By": "SRT Model"}
                )
            os.remove(image_path)
```

a. *load_known_encodings()* function loads known personnel facial encodings
b. Script scans new_person directory for newly saved images
c. *face_recognition.face_encodings()* will extract the necessary facial features from image
d. *face_recognition.compare_faces()* compares with known personnel, and sends a UDP message.

References/Resources:

Testing using Logitech C270 HD Webcam:
https://www.amazon.ca/Logitech-Widescreen-Calling-Recording-960-000694/dp/B004FHO5Y6/?th=1

Front face detection XML:
https://github.com/opencv/opencv/blob/master/data/haarcascades/haarcascade_frontalface_default.xml