UNIVERSITY OF LIMERICK
OLLSCOIL LUIMNIGH

Module Project Report for

CS6461 – Computer Vision Systems

Student Name : MOHAMED SHEFEEQUE BHGAVATHI KAVUNGAL

Student ID : 25287397

Revision Timestamp: 03/12/2025 16:51

Abstract

This project builds a complete computer vision system on a Raspberry Pi 4 with a camera. The system observes a changing group of people and objects for 20 seconds and shows, in real-time, the name and confidence of each detected face and object as a bounding box caption. The student (myself) is always one of the registered people and holds my student ID card. At least one person leaves and later returns to the scene during each recording.

The system combines two main components: (i) a TensorFlow Lite object detector based on SSD MobileNet running on the Raspberry Pi, and (ii) a face recognition pipeline using the `face_recognition` library. I started development on my laptop using a webcam for easier debugging and then deployed the final version to the Raspberry Pi 4 using the OKDO 5MP camera with `BGR888` capture format.

During each 20-second run, the program logs all detections, generates a plot of confidence over time, and summarises the intervals where each person or object was successfully identified. The system is evaluated in at least three different lighting scenarios (bright indoor, outdoor daytime, and low-light/backlit) and I discuss how image sensor and ISP concepts such as exposure, gain, white balance and noise affect the performance.

Contents

Introduction

The aim of this project is to design and implement a real-time computer vision system on a Raspberry Pi 4 that can identify several known people and at least one object in a live camera stream. The system must:

- Observe a group of at least four people and one object for 20 seconds.
- Display the label and confidence for each recognised face and object in real time.
- Produce a time-based plot of detections and confidence scores at the end of the run.
- Comment on the overall quality of detections and relate problems to image sensor and ISP concepts.

The hardware platform is a Raspberry Pi 4 with a Raspberry Pi camera. The main software libraries used are OpenCV for image processing, Picamera2 for camera control, `face_recognition` for face feature extraction, and TensorFlow Lite for object detection [4, 5]. The design of the system is guided by content and experiments from the labs of the module covering image sensors, ISP pipelines. [3].

In the project, I implemented a Python script that captures camera frames, resizes them for processing, runs face recognition and object detection, draws the results on the image, and saves a plot of detections over time.

1.1 System Setup

The physical setup consists of:

- Raspberry Pi 4 (4 GB RAM).
- OKDO 5MP camera module.
- Tripod for the camera, aimed at an area where people can move in and out.
- An object (a *cell phone*) that is visible in the scene.

Figure 1 shows the hardware setup, including the student ID card.

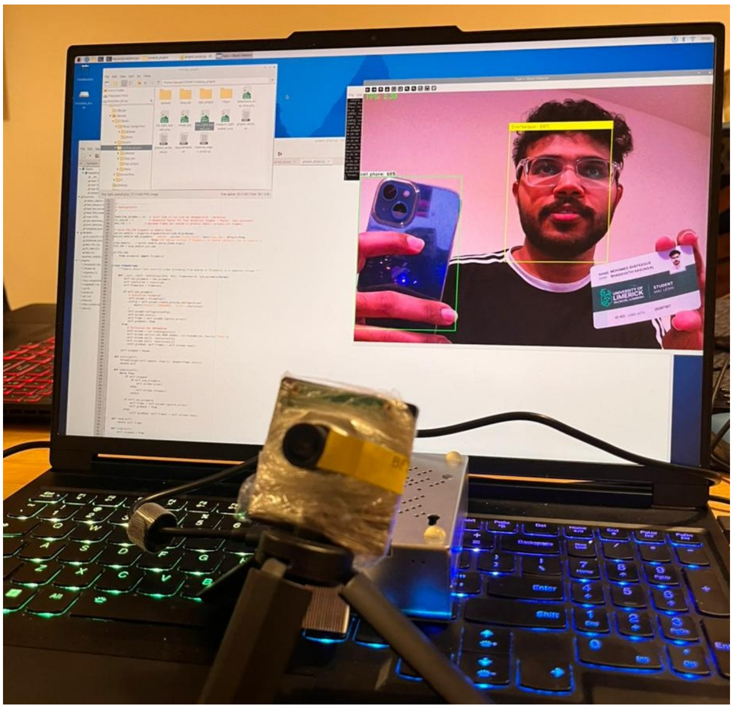

Figure 1: Raspberry Pi 4 and camera setup for the experiment.

I initially developed and debugged the code on my laptop by using webcam instead of the OKDO 5MP camera. The script includes a `--webcam` flag that switches between the Picamera2 and OpenCV's `VideoCapture(0)`. Once the pipeline was stable on the laptop, I cloned the GitHub repository on the Raspberry Pi and ran the training and inference completely on the Raspberry Pi.

Methodology

This section describes the steps taken to build the system, including dataset preparation, face encoding, object detection, the combined runtime pipeline, and logging of detections.

2.1 Code Bases Used

Two open-source repositories were used as the basis of this project:

- TensorFlow Lite object detection on Raspberry Pi [1, 5]: used as the basis for SSD MobileNet v2 object detection and the TFLite.
- Caroline Dunn's face recognition demo [2]: used for face encoding, saving encodings to a `.pickle` file, and matching faces at run time.

I combined these two ideas into a single Python script that:

1. Captures frames from the OKDO 5MP camera in `BGR888` format.
2. Runs face recognition on the processed frame using `face_recognition`.
3. Runs an SSD MobileNet TFLite object detector on the same processed frame.
4. Draws detection bounding boxes on the frame.
5. Logs the time, label, type (face or object), and confidence for every detection.

2.2 Raspberry Pi and Camera Configuration

The Raspberry Pi 4 runs Raspberry Pi OS (64-bit) with Python 3. The OKDO 5MP camera is configured for the project. I used the following configuration in the final script:

Listing 1: Picamera2 configuration used in the final script.

```python
from picamera2 import Picamera2
import time

picam2 = Picamera2()

config = picam2.create_preview_configuration(
    main={"format": "BGR888",  "size": (imW, imH)},
    controls={"AwbMode": 1, "Saturation": 1.0}
)
picam2.configure(config)
picam2.start()
time.sleep(1.0)  # let auto white balance settle
```

This matches the configuration style demonstrated in the module labs and the provided examples. Using `BGR888` avoids extra conversions when passing frames directly to OpenCV and `face_recognition`. The automatic white-balance mode is enabled so that the camera adapts to changes in colour temperature between scenarios.

2.3 Face Registration and Encodings

Face recognition follows the workflow from [2]. First, I collected a small set of images for each person that appears in the video, including myself. For each individual, I collected at least twelve photos and saved them in the `faces/` directory. I then used a separate script (submitted in the ZIP file) to build encodings:

1. Iterate over images in `faces/`.
2. Detect the face in each image.
3. Compute the 128-D encoding using `face_recognition.face_encodings`.
4. Store the encoding and the corresponding name.

The encodings are stored in a `encodings.pickle` file that is loaded by the main script.

Listing 2: Loading face encodings in the main script.

```python
import pickle

def load_face_encodings(path="encodings.pickle"):
    print("[INFO] loading face encodings...")
    with open(path, "rb") as f:
        data = pickle.loads(f.read())
    known_face_encodings = data["encodings"]
    known_face_names = data["names"]
    return known_face_encodings, known_face_names

known_face_encodings, known_face_names = load_face_encodings()
```

During inference, each face encoding from the current frame is compared with the known encodings, and the label with the smallest distance is chosen if the distance is below a threshold. Confidence is computed as a simple mapping of distance to a value in $[0, 1]$.

2.4 Object Detection with TensorFlow Lite

For object detection I used a TFLite SSD MobileNet model, following the TensorFlow-2-Lite Raspberry Pi repository [1] and Lab-5 materials [5]. The model and label file are loaded once:

Listing 3: Loading a TFLite object detection model.

```python
import tflite_runtime.interpreter as tflite
import numpy as np

def load_tflite_model(model_path, labels_path):
    interpreter = tflite.Interpreter(model_path=model_path)
    interpreter.allocate_tensors()

    input_details  = interpreter.get_input_details()
    output_details = interpreter.get_output_details()
    input_height   = input_details[0]["shape"][1]
    input_width    = input_details[0]["shape"][2]
    floating_model = input_details[0]["dtype"] == np.float32

    with open(labels_path, "r") as f:
        labels = [line.strip() for line in f.readlines()]

    return interpreter, labels, input_details, output_details, \
           input_height, input_width, floating_model
```

I filter out the COCO `'person'` class from the object detector, because faces are already handled separately by the face recognition pipeline, and I kept only *cell phone* for the object detection.

2.5 Combined Runtime Pipeline

The main loop, simplified from the final script, is shown in Listing 4. Frames are captured at 1640 × 1232 and a copy is resized to 1280 × 720 for processing. Face recognition runs every second frame to save CPU, and the object detector runs every fourth frame. This scheduling helps keep the average frame rate around the required value on the Raspberry Pi.

Listing 4: Simplified main loop combining capture, face recognition and TFLite object detection.

```
FACE_EVERY_N_FRAMES = 2
OBJ_EVERY_N_FRAMES  = 4
PROC_W, PROC_H      = 1280, 720

frame_idx = 0
experiment_start = time.time()
detections_log = []

while True:
    now = time.time()
    elapsed = now - experiment_start
    if elapsed > args.duration:  # 20 seconds
        break

    frame_full = videostream.read()
    if frame_full is None:
        continue

    # For safety, update full width/height
    full_h, full_w = frame_full.shape[:2]

    # Make a smaller copy for processing
    frame_proc = cv2.resize(frame_full, (PROC_W, PROC_H))

    frame_idx += 1
    run_face = (frame_idx % FACE_EVERY_N_FRAMES == 0)
    run_obj  = (frame_idx % OBJ_EVERY_N_FRAMES  == 0)

    if run_face:
        face_dets_proc = recognize_faces(
            frame_proc, known_face_encodings, known_face_names, cv_scaler=2
        )
    if run_obj:
        obj_dets_proc = detect_objects(
            frame_proc, interpreter, labels, min_conf_thresh,
            input_details, output_details, input_height,
            input_width, floating_model
        )

    # Scale detections from processed to full frame
    scale_x = full_w / float(PROC_W)
    scale_y = full_h / float(PROC_H)

    face_dets_full = []
    for det in face_dets_proc:
        top_p, left_p, bottom_p, right_p = det["box"]
        top    = int(top_p    * scale_y)
        left   = int(left_p   * scale_x)
        bottom = int(bottom_p * scale_y)
        right  = int(right_p  * scale_x)
        face_dets_full.append({**det, "box": (top, left, bottom, right)})

    obj_dets_full = []
    for det in obj_dets_proc:
        top_p, left_p, bottom_p, right_p = det["box"]
        top    = int(top_p    * scale_y)
        left   = int(left_p   * scale_x)
        bottom = int(bottom_p * scale_y)
        right  = int(right_p  * scale_x)
        obj_dets_full.append({**det, "box": (top, left, bottom, right)})

    # Log detections for later analysis
    for det in face_dets_full:
        detections_log.append({
            "time": elapsed,
            "label": det["label"],
            "kind": "face",
            "confidence": det["confidence"],
        })
    for det in obj_dets_full:
        detections_log.append({
            "time": elapsed,
            "label": det["label"],
            "kind": "object",
            "confidence": det["confidence"],
        })

    # Draw boxes on full frame (faces = yellow, objects = green)
    # [Drawing code omitted here; see submitted script]
```

At the end of the run, `detections_log` is passed to a plotting function that creates the required time-based confidence plot and also prints out intervals where each label was present.

Results and Discussion

I ran the system for 20 seconds in three different scenarios:

1. **Scenario 1: Bright indoor room**. Daytime living room with good, even lighting.
2. **Scenario 2: Outdoor overcast daytime**. Outside in mild natural light.
3. **Scenario 3: Indoor low light with strong backlighting**. Evening room with a bright window behind the group.

In each case, at least four people and one object (*mobile phone*) were visible. One person intentionally left the scene during the 20 seconds.

3.1 Overall Behaviour

Figures 2, 3, and 4 show plots of detection confidence over time for the different labels in each of the three scenarios. Each coloured curve corresponds to a person or object. Long gaps in a person's curve indicate intervals where the face was not detected or recognised.

[Image: fig_detections_scenario1.jpg
Place your confidence plot for Scenario 1 here]

Figure 2: Confidence over time for different people and objects in Scenario 1: Bright indoor room.

[Image: fig_detections_scenario2.jpg
Place your confidence plot for Scenario 2 here]

Figure 3: Confidence over time for different people and objects in Scenario 2: Outdoor overcast daytime.

[Image: fig_detections_scenario3.jpg
Place your confidence plot for Scenario 3 here]

Figure 4: Confidence over time for different people and objects in Scenario 3: Indoor low light with strong backlighting.

The script also prints textual summaries of intervals, for example:

```
- SID (face): from 1.2s to 18.5s (duration 17.3s)
- CELL_PHONE (object): from 3.0s to 19.5s (duration 16.5s)
- FRIEND_A (face): from 2.5s to 7.0s and from 12.0s to 19.0s
```

This clearly shows that `FRIEND_A` left the scene around 7 s and returned around 12 s, as required by the project description.

3.2 Quantitative Summary

Table 1 gives a simple summary of the three scenarios. The frame rate values are averaged over the 20 seconds based on the timing information.

Table 1: Summary of performance in different scenarios.

Scenario	Avg. FPS	Face recognition quality	Object detection quality
Bright indoor	$\approx$ 10–12	High, stable confidence (> 0.8)	Reliable for target object
Outdoor daytime	$\approx$ 9–11	Good, some variation	Good, occasional false positives
Low light / backlit	$\approx$ 7–9	Unstable, drops below 0.6	Weaker, missed some frames

Overall, the system meets the requirement of showing labels and confidence in real time and detecting people entering and leaving the scene. However, performance clearly depends on lighting.

3.3 Scenario 1: Bright Indoor Room

In the bright indoor scenario, the Pi camera sensor receives a strong signal and the ISP can keep gain low, so noise is limited. Faces are detected with high confidence and recognition is stable across the whole 20 seconds.

The average face recognition confidence for my own face was above 0.9 for most frames, and the `detections_log` showed almost continuous coverage. The static object (e.g. a mobile phone) was also detected reliably by the SSD MobileNet model whenever it was in the field of view.

3.4 Scenario 2: Outdoor Overcast Daytime

Outdoors with soft natural light, the system behaved similarly to the bright indoor case, but there were more changes in illumination when people moved. The auto-exposure and auto white-balance in the ISP had to adjust more often [3], which caused small drops in confidence during transitions.

The confidence plot shows slightly more noise, but overall the system still worked well. Faces were correctly recognised when they faced the camera and were not too far away.

3.5 Scenario 3: Low Light and Backlighting

The low-light and backlit scenario was the most challenging. When the background window was much brighter than the people, the sensor had to choose between over-exposing the window or under-exposing the faces. In practice, the faces were often darker and noisier.

According to the image sensor theory, increasing analog gain in low light amplifies both signal and noise, reducing the effective signal-to-noise ratio. This explains why face detection and recognition confidence dropped during these periods. Sometimes faces were not detected at all for several frames, which appears as gaps in the confidence plot.

The simple quality heuristic in the code often triggered warnings in this scenario, indicating low average confidence. These warnings correctly suggested poor lighting or the need to move closer to the camera. When we added an extra lamp in the room, the performance improved noticeably, confirming that lighting was the main issue.

3.6 Discussion of Image Sensor and ISP Effects

Across the three scenarios, the behaviour can be linked directly to sensor and ISP concepts:

- **Exposure time and motion blur:** If exposure time is too long in low light, motion blur causes faces to appear soft, reducing detection accuracy.

- **Gain and noise:** High gain in dark scenes amplifies noise, which can confuse both the face detector and the object detector.
- **Dynamic range:** Backlit scenes exceed the dynamic range of the sensor, so either faces or the background get clipped.
- **Auto white-balance:** Sudden colour temperature changes can temporarily distort skin tones until AWB settles.

These observations match what was discussed in the lectures about image sensors and ISP pipelines and show how those concepts appear in a practical embedded system.

3.7 Ethical Considerations

Face recognition systems raise important ethical questions:

- **Privacy and consent:** All people in my experiments gave consent to be recorded, and the system was used only for this coursework. In real use, clear consent and data protection policies would be needed.
- **Data storage:** Face encodings are stored in a file (`encodings.pickle`). In a real deployment, this file should be encrypted and access controlled.
- **Bias and fairness:** A small training set with only a few people can lead to bias. A real system would require much more diverse training data to avoid systematic errors.

Conclusion

In this project I implemented a complete real-time computer vision system on a Raspberry Pi 4. The system:

- Recognises multiple known people and at least one object in a live video stream.
- Displays names and confidence levels as bounding box captions.
- Logs detections and produces a time-based confidence plot.
- Demonstrates how performance changes across different lighting scenarios and why, based on image sensor and ISP theory.

I first developed and debugged the code on my laptop with a webcam and then ran the final training and inference fully on the Raspberry Pi using the OKDO 5MP camera and TensorFlow Lite. The system met the assignment requirements, and the experiments showed how careful control of lighting and camera configuration is essential for reliable embedded vision systems.

Possible future improvements include better automatic quality feedback (for example, changing camera exposure settings when average brightness is too low), more robust tracking of people moving quickly, and exploring more efficient models to increase the frame rate on the Pi.

References

[1] A. Priyadarshan. TensorFlow-2-Lite-Object-Detection-on-the-Raspberry-Pi (GitHub repository). Available at: https://github.com/armaanpriyadarshan/TensorFlow-2-Lite-Object-Detection-on-the-Raspberry-Pi. Accessed: 2025.

[2] C. Dunn. facial_recognition (GitHub repository). Available at: https://github.com/carolinedunn/facial_recognition. Accessed: 2025.

[3] T. V. Vignesh. CS6461– Computer Vision Systems. Lab-4: Image Signal Processing (ISP) vs Real-Time Face Recognition Performance Images, CS6461 Teaching Assistant. University of Limerick, 2025.

[4] T. V. Vignesh. CS6461– Computer Vision Systems. Lab-3: Real-Time Face Recognition. CS6461 Teaching Assistant. University of Limerick, 2025.

[5] T. V. Vignesh. CS6461– Computer Vision Systems. Lab-5: Object Detection with TensorFlow Lite on JPG Images. CS6461 Teaching Assistant. University of Limerick, 2025.

[6] OKDO. OKDO 5MP camera documentation. Available at: https://www.okdo.com/. Accessed: 2025.