Developing a Facial Detection Program on a Raspberry Pi 4

Alex Rodriguez
CS370*: Operating Systems*
Colorado State University
Prof. Yashwant Malaiya
Fort Collins, Colorado
Alex.Rodriguez@colostate.edu

Zacharie Guida
CS370*: Operating Systems*
Colorado State University
Prof. Yashwant Malaiya
Fort Collins, Colorado
Zacharie.Guida@colostate.edu

Jocelyn Villegas
CS370*: Operating Systems*
Colorado State University
Prof. Yashwant Malaiya
Fort Collins, Colorado
Jocelyn.Villegas@colostate.edu

Abstract*— This paper introduces the methodologies and overview of developing and evaluating a facial detection program's features, limitations, and potential applications integrated with a 4GB Raspberry Pi 4 that communicates with a 4MP camera peripheral. In this development project, a program created using computer vision and machine learning for facial recognition from an open-source library for Python will be evaluated for power consumption, response times, and, thus, its potential for public or market use.

I. Introduction

Traffic lights, home security cameras, and dashcams- we are surrounded by algorithms that serve as intelligent decision-makers who interact between the cusp of our physical world and digital worlds: embedded systems. As a result of today's climate regarding emerging biometric technology, performing experiments on a scaled version of a facial detection program is essential for understanding how secure and effective technologies we use and see every day can be. This report aims to describe and document the process of designing and implementing the embedded system of a camera in an IoT device that serves the purpose of facial recognition. Receiving and processing data, making an intelligent decision based on an algorithm, and generating an output with a real-world impact is the epitome of an embedded system.

A. A Brief History

Historically, facial detection has been an elusive feat, most known by the public through its references in pop culture. Facial detection capabilities, however, have been well within computer scientists' reach since the 1960s due to Woody Bledsoe, Helen Chan Wolf, and Charles Bisson, whose work focused on teaching computers to recognize human faces. Facial recognition's beginning was dubbed a "man-machine," as humans would need to manually delegate coordinates of various facial features to be stored in a database- A task automated by machine learning algorithms today [1]. Thanks to continued research within human-centered computing, open-source libraries such as OpenCV now exist for academics, researchers, and curious individuals like us to create and explore facial recognition technology for ourselves.

II. Problem Characterization

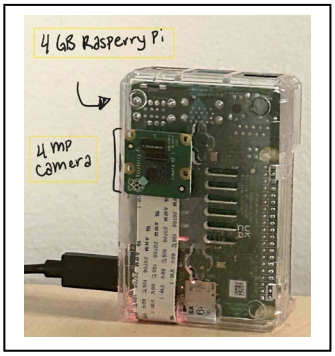
Before the development of a program begins, the integration of a camera peripheral requires that a system meets specific criteria to ensure it can handle the procedures that will be expected of it. Physically, using a single-board computer allows the convenience and portability seen in real-world applications of facial recognition products such as home security cameras or the more inconspicuous
examples of facial recognition systems in public spaces such as airports and subways. To ensure our program runs satisfactorily, a Raspberry Pi 4 [Figure 1] with the following specifications was selected [Table 1].

Table 1

Specification	Purpose
Quad-core Processing	Handle multiple tasks simultaneously
4 GB Memory	Handle real-time image processing and maintain a reference database
Cooling System	Ensure stable performance during long-term use

Of course, on a larger scale, such as for use in public spaces, these specifications would increase exponentially; however, for personal use and perhaps most similarly to a single-family home's security camera, these specifications suffice. Additionally, these specifications loosely follow the functionality requirements per the IEEE Standard for Technical Requirements for Face Recognition, a significant reference for characterizing a facial recognition device. [2]

FIGURE 1



Lastly, because this is an embedded system development project, our device is expected to perform only its specific facial recognition function rather than a range of tasks. Therefore, we have configured our Raspberry Pi 4 to communicate with three other external computers to utilize our system as a single-purpose machine. This was achieved through the following:

- Configured with a static IP address [3] for the necessary "reservation" of a non-changing IP address (for the ease of remote connections)

- Configured for use via other external systems:
  - Remote user interface: The GUI can be used on team members' laptops through the RealVNC Viewer application.
  - SSH connection via the command line on systems on the same network for memory/file management and updates/package installation.

III. Proposed Solutions and Implementation Strategy

A. Methodology

Our approach for developing a facial recognition program utilizes the Raspberry Pi operating system and begins by installing various necessary packages onto the system:

- Python3 (Primary Programming Language)
- Numpy (Computations)
- OpenCV: (Open Source Library)
  - Python face_recognition package
  - Imutils 0.5.4
  - cv2

Upon the completion of the program, live tests were performed via personal analysis for accuracy, ensuring sufficient documentation for ease of use and understanding, as well as the research and use of external tools for our primary data collection.

The following external tools were utilized, analyzed, and averaged with results from the built-in tools in Python and specifications from the system itself. [4] The following methods each have their tradeoffs, encouraging the derivation of an overall average. Limitations, if any, were also noted.

- *Python:* The psutil library measures CPU usage and time spent during the program.
- *Raspberry Pi System Statistics:* The Raspberry Pi OS provides terminal commands that return information about CPU load and temperature, which can be helpful when gauging power usage.

Program response times were analyzed via existing Python libraries:

- *time.perf_counter():* Used to get a precise elapsed time value.
- *timeit:* Used to time specific code sections to analyze our program's most time-consuming tasks.

B. Libraries

Open-Source Computer Vision [5] is powered by machine learning algorithms and artificial intelligence-making it a powerful tool for various image processing applications such as our own.

Because the library is an Apache 2 licensed product [6], OpenCV has few restrictions, allowing businesses and academics to utilize the code for various projects. It contains "more than 2500 optimized algorithms" [7] that give programmers the ability and potential to recognize faces, identify objects, find similar images from a given database, and more. Various reputable companies such as Google, Microsoft, Intel, IBM, Sony, and Toyota utilize the library, making it an easy choice due to its versatility, functionality, and extensive documentation on the official webpage and code base repository [8]. The library supports all major operating systems: Windows, Linux, Android, and Mac OS, and has C++, Python, Java, and MATLAB interfaces.

C. Design Integration

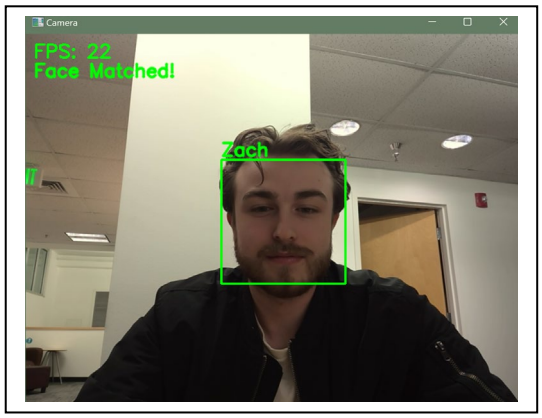
The program was implemented in Python utilizing libraries from OpenCV and developed, branched, tested, and merged via a GitHub repository [9]. Additionally, the program was primarily tested through the command line's SSH from three external systems, as mentioned earlier.

Our program can be simplified in that it performs three significant tasks:

- Run a live video capture feed that continuously takes data as image frames.
- Utilize OpenCV Library to detect human faces.
- Maintain and reference a serialized ("pickled") and identified faces database.

When these tasks are done correctly and efficiently, it is possible to produce an identification output [Figure 2].

FIGURE 2



Our program uses a pre-trained model called a Haar Cascade or, as instantiated in the code found in main.py, a "Cascade Classifier" named "face_detector." This object detection algorithm is what we use to detect human faces. [10] OpenCV provides various .xml files to be utilized as pre-trained models for specific human body regions and objects. We used the "frontal face" specific model. Utilizing "Haar" features is a machine learning algorithm developed by Paul Viola and Michael Jones in their work analyzing how a cascade of simple features serves as an effective technique for facial detection [11].

Additionally, the manipulation of images (frames taken from video stream) is achieved through the functions provided through the "cv2 import". This allows us to compare frames optimally by converting images to greyscale before invoking face detection functions and scaling the images and frames to be as similar in size as possible for an equal and fair comparison.

Pickler.py serves as the face enrollment database and interface for our program as it requests input and assigns name Strings using Python's Dictionaries to generate .pickle files created from serializing images taken of a single specific individual. "Pickling" [12], the images collected flatten the data into binary streams to then be stored in a designated directory: "pickled_images."

A comparison between two faces is performed by calculating the *mean sum error* [Figure 3].

FIGURE 3

$$\text{Mean Sum Error} = \sum (2(\text{Face1Size} - \text{Face2Size}))/\text{Face1Size}$$

The above computation uses the numpy library, a frame from the video stream, and an image from within the database of pickled image references. After the image sizes have been adjusted to be as similar as possible to ensure accurate and equal capturing of a frontal face- The "MSE" computation takes the summation of the difference between two images- doubled. This final summation is then divided by the frame's image size. The mean sum error determines facial match accuracy and can be adjusted. Our program deems any MSE value less than 90 to be a match.

Most importantly, Main.py serves as the image and video analysis subsystem and contains methods for the decision-making and comparison subsystems, as described above.

IV. QUANTITATIVE EVALUATIONS

To judge the effectiveness of our program, the following attributes were evaluated: analysis of power consumption on the hardware (Raspberry Pi 4), program response times, and, ultimately, its comparison to the IEEE Standards for Face Recognition [1] and guidelines for computational efficiency from ROE [13] an American biometric and computer vision provider.

A. Power Consumption

We evaluated various CPU metrics [Table 2], including the average CPU load, to compute our overall power consumption. When no programs were running, we ran a baseline measurement of our Raspberry Pi 4 system to act as a point of reference and comparison. Because our machine is quad core, the goal was to maintain a CPU load average below 4.0.

TABLE 2

System State	**Average 15 Minute CPU Load**	**Cores Running**	**CPU Temperature**	**Memory Used**
Baseline	0.36	1/4	40.4 °C	619 MB / 3.70 GB
Running Facial Detection Program	0.74	4/4	46.2 °C	694 MB / 3.70 GB

TABLE 3

CPU Load	**Meaning**
0 %	No CPU activity
0 -50 %	Low CPU activity
50 – 70%	Moderate CPU activity
70 – 100 %	High CPU activity

According to the Raspberry Pi documentation [13], reduced CPU performance can begin once the CPU reaches 85 °C, thus, to compensate for background processes running in addition to our program we strived for a temperature below 70 °C. Additionally, our program falls under the "High CPU Activity" at 74 % according to the Raspberry Pi hardware documentation [Table 3] [14]. It is also important to note that our program's method for facial recognition uses a significant amount of memory as we save recognized faces as .pickle files in varying amounts, therefore "a larger binary size will occupy a high percentage of available memory"[13]. This may also impact our program response times as discussed below, as detected individuals are assumed to already be enrolled in the database.

B. Response Time

We began with a baseline measure of OpenCV's facial detection algorithms before measuring the facial match response times. The following time measure values were collected each time the. detectMultiScale() method was called on the cascade classifier and averaged using the following calculation [Figure 4].

FIGURE 4

Average Facial Detection Time = ∑ (.detectMultiScale() call time)) / Total Detected Frames

This resulted in an average facial detection time of 0.0879 seconds, which meets the IEEE standard for face verification response time of no more than 2 seconds [2].

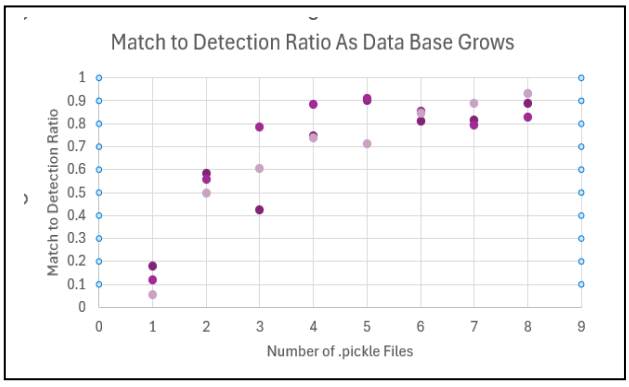
In addition to facial detection time, we calculated a facial match time by measuring the amount of time it takes to detect a match while iterating through the pickled_images database of face references. The average facial match time was derived using a similar method as the average facial detection time [Figure 5].

FIGURE 5

Average Facial Match Time = ∑ (Time it takes to find an MSE < 90) / Total Matched Faces

This resulted in an average facial match time of 0.0003 seconds. For face identification, the average response time of the face recognition system should not be greater than 2 seconds, according to the IEEE [2]. The correlation between response time and memory usage was also evaluated, as we measured the time taken to perform these tasks in relationship with the number of "pickled" images or the number of references in the face database for any single given individual [Figure 6].

FIGURE 6



Accuracy of our program was calculated with the following ratio [Figure 7]. This resulted in a consistent measure of how often our program can match and identify a human face. The accuracy of our program was also measured in relationship to the database size [Figure 8][Figure 9]. Throughout multiple tests with varying facial angles, and letting the program run for 10 seconds each time demonstrated a trend that the larger the database, the higher the accuracy.

FIGURE 7

Match to Detection Ratio = Total Number of Matches / Total Number of Detections

FIGURE 8

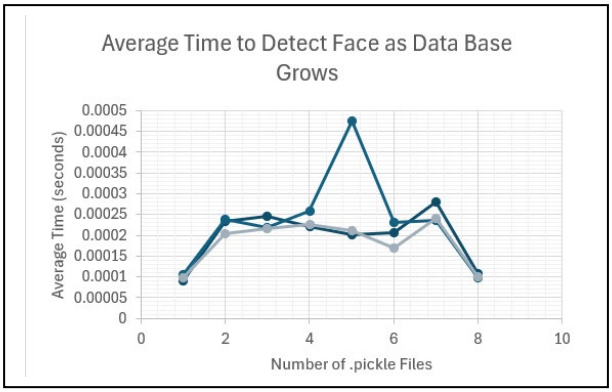
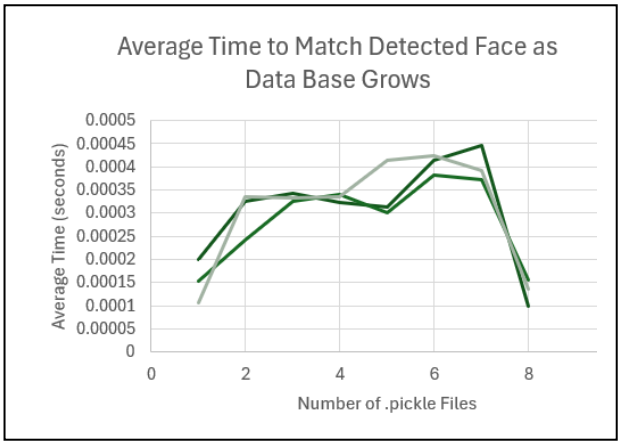


FIGURE 9



V. CONCLUSION

By leveraging OpenCV, and the libraries found within Python , we successfully developed a program capable of detecting and matching faces. Our efforts and evaluations represent the opportunity to further understand the viability of facial recognition solutions and have highlighted the balance between performance, resource consumption, and hardware limitations in facial detection systems. Providing an

exploratory understanding of potential low-cost and accessible face detection solutions while acknowledging the challenges of scaling for larger applications. We hope our work has provided a strong basis for future efforts that could apply additional optimization strategies to minimize response times and reduce power consumption. At this time the system has met and exceeded the goals we originally set out to accomplish while maintaining scope.

ACKNOWLEDGMENTS

This template was adapted from those provided by the IEEE on their website. [15]

REFERENCES

[1] Wikipedia contributors. (2024, November 15). *Facial recognitionsystem*.Wikipedia. https://en.wikipedia.org/wiki/Facial_recognition_system

[2] "IEEE Standard for Technical Requirements for Face Recognition," in IEEE Std 2945-2023, vol., no., pp.1-28, 27 March 2023, doi: 10.1109/IEEESTD.2023.10081261.

[3] "Set up a Static IP-Address." *The Raspberry Pi Guide*, raspberrypi-guide.github.io/networking/set-up-static-ip-address.

[4] *The Python profilers*. (n.d.). Python Documentation. https://docs.python.org/3/library/profile.html

[5] "OpenCV." *OpenCV*, opencv.org/#.

[6] Snyk. (2022, January 31). *Apache License 2.0 explained*. https://snyk.io/learn/apache-license/

[7] OpenCV. (2020, November 4). *About - OpenCV*. https://opencv.org/about/

[8] Opencv. (n.d.). GitHub - opencv/opencv: Open Source Computer Vision Library. GitHub. https://github.com/opencv/opencv

[9] al3jandroR.(n.d.-b).*GitHub-al3jandroR/Face-Detection_Project*. GitHub. https://github.com/al3jandroR/Face-Detection_Project

[10] *OpenCV:CascadeClassifier*.(n.d.).https://docs.opencv.org/3.4/db/d28/tutorial_cascade_classifier.html

[11] J. M. Al-Tuwaijari and S. A. Shaker, "Face Detection System Based Viola-Jones Algorithm," 2020 6th International Engineering Conference "Sustainable Technology and Development" (IEC), Erbil, Iraq, 2020, pp. 211-215, doi: 10.1109/IEC49899.2020.9122927. keywords: {Face detection;feature optimization;viola jones;Linear Discernment Analysis (LDA);Viola-Jones;BAT algorithms;Chicken algorithms},

[12] *pickle — Python object serialization*. (n.d.). Python Documentation. https://docs.python.org/3/library/pickle.html

[13] Klare, B. F. & Rank One Computing. (2020b). Efficiency considerations for face recognition algorithms. In *International Face Performance Conference (IFPC)*.

[14] RaspberryPi (n.d.). *Raspberry Pi Hardware*. RaspberryPi.com. Retrieved November 19, 2024, from https://www.raspberrypi.com/documentation/computers/raspberry-pi.html#measure-temperatures

[15] IEEE - The world's largest technical professional organization dedicated to advancing technology for the benefit of humanity. (n.d.). https://www.ieee.org/