Biometrical Authentication System Using Face Recognition (Practical Project)

Adam Rogowski[1], Dmytro Kirshev[2]

[1] s232672@dtu.dk

[2] s232671@dtu.dk

Technical University of Denmark

Github repository

***Abstract*—This project was developed as part of the Network Embedded Systems course at DTU. The system comprises two interconnected modules communicating through a local server. The first module involves an ESP32-CAM, which captures a user's facial image. This image is then transmitted via HTTP to the local server. The server processes the image by comparing it with a database of 'known' faces. Upon analysis, the comparison results are published on an MQTT broker. Simultaneously, a second ESP-WROOM-32 module subscribes to the corresponding MQTT topic. It retrieves and displays the comparison outcome using external LEDs, providing a visual representation of the facial recognition results.**

I. Introduction

The conceptual framework of the system is illustrated in Fig. 1. Subsequent sections (II-IV) provide detailed descriptions of each element within the system. Section V presents the execution of a performance experiment along with the corresponding results. Paper is summarized with discussion about possible areas of development and improvements.

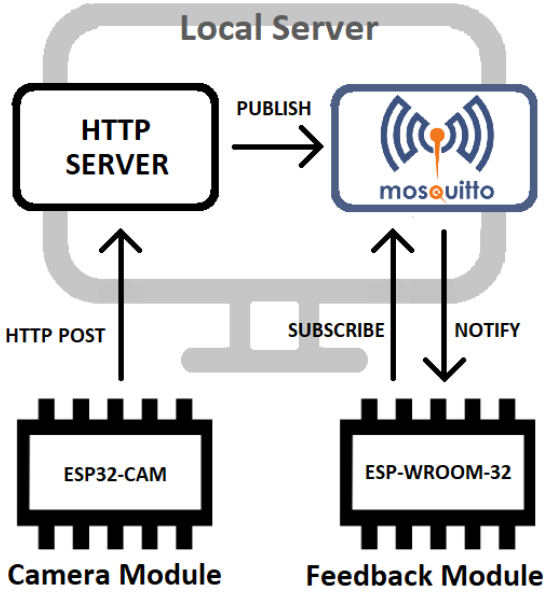

Fig. 1. Setup of deployed system conceptually consists of three elements: Camera Module, Local Server and Feedback Module

II. Camera Module

A. Physical Components

At the core of this element is the ESP32-CAM module, selected for its cost-effectiveness and versatility, making it well-suited for the requirements of this prototype/proof-of-concept project. The module includes an integrated camera and a built-in Wi-Fi module, facilitating a prompt and straightforward setup for a seamless solution. Furthermore, the ESP32 family benefits from an extensive and active community, offering many solutions and broad support.

In addition, the HC-SR04 Distance Ultrasonic Sensor was employed to initiate the capture of a facial image when the user is within a specific distance from the module. This sensor is widely recognized, accessible, and affordable, making it a fitting choice for integration into the project.

B. Technology Considerations

The Arduino IDE (Integrated Development Environment) was selected for program deployment primarily due to the authors' positive past experiences with the platform. While another consideration was using Visual Studio Code with ESP-IDF (Espressif IoT Development Framework), the authors are not familiar with this framework, which could potentially impede the deployment process.

Further considerations were given to the communication protocol between the Camera Module and the Local Server. Despite several valid options, HTTP was chosen as the most appropriate protocol for image transmission. While Websocket and Socket.IO were also evaluated, HTTP stood out due to its simplicity of implementation compared to these alternatives. Moreover, Websocket and Socket.IO introduced unnecessary overhead, especially considering the occasional nature of image transfers; features such as full-duplex communication were deemed excessive. The request-response model of HTTP aligns well with project requirements, given that such authentication system doesn't necessitate a persistent, long-live connection.

C. Program Overview

Upon initiation, the program configures peripherals by setting up the camera, pins for the built-in flash LED, and managing the distance sensor. Subsequently, it establishes a

connection to Wi-Fi and initiates a time counter. A deliberate project decision was made to enable the module to capture an image only at specified intervals. If the user is within the defined distance and the stipulated time has elapsed, a photo is taken, and the counter resets. Consequently, the user must wait until the next identity verification.

To signify image capture, the flash LED flashes, followed by the transmission of an HTTP POST request to the local server's address. Notably, the image is sent in raw format without processing on the ESP32-CAM module. While one might question the optimal approach in terms of message size, considering the absence of image compression, it was determined that processing the image with low-level Arduino tools proved to be complex compared to a straightforward conversion on the server side.

D. Final Result

The prototype was assembled efficiently using a compact breadboard, as shown in Fig. 2. The camera lens is strategically positioned just above the HC-SR04 sensor, ensuring that the user's facial image is directly captured when standing in front of the sensor.

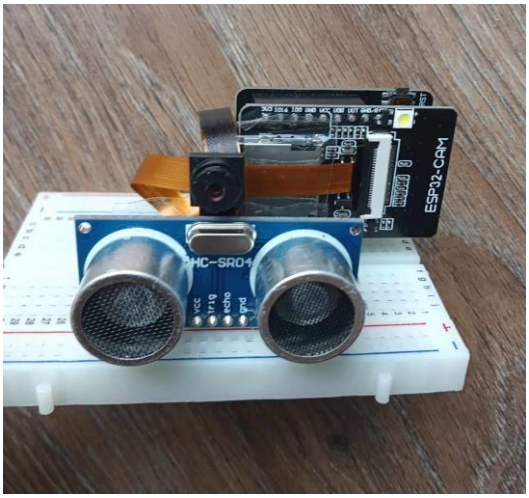

Fig. 2. Final prototype of the Camera Module

III. Local Server

A. Technology Considerations

Python, as a programming language, emerged as the evident choice for implementing the server, not only due to its suitability for deploying a basic, straightforward server but more importantly because of its compatibility with numerous powerful computer vision tools, which could allow analysis and comparison of the incoming facial images. While Node.js environment was initially considered, and even a Websocket server was deployed and connected with the Camera Module, the Python option ultimately prevailed. Both attempts were carried out in the Visual Studio Code IDE.

The project's goal was **not** to implement a machine-learning-based model for face recognition but to leverage an existing one. To fulfill this requirement, the PyPI face-recognition package was employed. This package offers a simple and intuitive interface, along with a powerful model. Notably, throughout the entire development and testing process, no false-positive face recognition errors were observed.

Additionally, the project utilized the following extensions/libraries to cover essential functionalities:

- Flask: Deployed for setting up the HTTP server.
- NumPy and OpenCV (cv2): Utilized to convert raw images to the jpg format.
- Paho MQTT: Employed for publishing MQTT messages to the broker.

The selection of these tools was primarily influenced by their popularity and the availability of usage examples, making their integration into the project straightforward.

B. Program Overview

Prior to initiating the HTTP server, the program incorporates all locally available facial images as references into the model, a process that is somewhat time-consuming. Subsequently, recognizing whether a new image contains previously included references becomes nearly instantaneous, eliminating the need for the user to wait during this evaluation.

Once this initialization is complete, the HTTP server is opened to incoming requests. Upon receiving a valid payload from the Camera Module, the received image is converted and examined for the presence of reference faces. The resulting analysis is published to the broker, rendering the server ready for subsequent requests.

C. Mosquitto Broker

The decision to employ MQTT for indirect communication between the Local Server and the Feedback Module lacks clear technological justification. While it effectively meets all project requirements in practice, the choice of this protocol was primarily driven by the authors' curiosity to explore the implementation of communication using a different protocol. The exchange of downlink messages from the Local Server could have been achieved equally well using HTTP or similar alternatives.

Inevitably, some overhead was introduced with the establishment of a Mosquitto Broker, responsible for listening to topic publications and subscriptions between the Local Server and the Feedback Module. However, aside from this, implementing MQTT client functionality is straightforward on both the high-level server side and the low-level esp-wroom-32 side.

IV. Feedback Module

The deployment of the third element of the project, the Feedback module, was relatively straightforward, focusing solely on receiving notifications from the MQTT broker regarding the outcome of face comparison. Subsequently, it displays the result using external LEDs. The ESP-WROOM-32, a familiar module from the ESP32 family, was employed for this purpose.

Upon connecting to Wi-Fi and subscribing to the relevant topic, the program enters the default state, indicated by the illumination of the blue LED. It remains in this state, awaiting results sent from the Local Server. Upon receiving a positive evaluation of face comparison, the green LED lights up for a specified time period, after which the program returns to the default state. Conversely, for a negative result, the red LED is illuminated. In the final prototype, all these components were integrated on a breadboard, as depicted in Fig. 3.

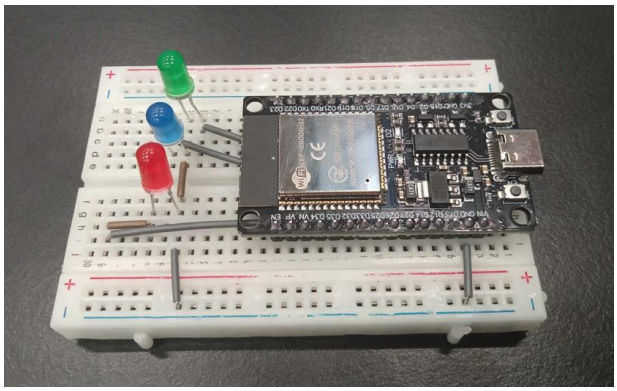

Fig. 3. Final prototype of the Feedback Module

V. Experiment

For the sake of meeting project requirements, an experiment was conducted with the objective of measuring the system's efficiency in authenticating the user based on the distance between the user and the Camera Module. The practical implications of the experiment lie in identifying the critical distance at which the system's inaccuracy remains within an acceptable level. However, from a theoretical standpoint, the experiment doesn't offer intriguing implications as it effectively measures the accuracy of PyPI's model when lower-quality images are provided due to increased distance from the camera lens.

The experiment took place indoors in a well-illuminated room, intentionally excluding the additional factor of daylight to simulate a worst-case scenario for the system in practical situations. The Camera Module was fixed in place, and tests were carried out by incrementally increasing the CAPTURE_DISTANCE constant in the code, leading to the triggering of photo capture. Results of the tests were included in the Table 1.

TABLE 1.
Performance Metrics for Varying Capture Distances

CAPTURE DISTANCE [cm]	30	60	90	120	150	180	210
Real Distance [cm]	29	63	99	138	189	240	>300
Number of Positive Trials	20	20	20	20	20	20	20
Number of Negative Trials	5	5	5	5	5	5	5
False Negatives	1	0	1	2	4	7	10
False Positives	0	0	0	0	0	0	0
Accuracy [%]	**96**	**100**	**96**	**92**	**84**	**72**	**60**

Noticeable discrepancies were observed between the calculated theoretical CAPTURE_DISTANCE and the actual trigger distance (measured manually). These variations indicate the HC-SR04 sensor's inaccuracy, particularly evident when measuring longer distances.

Ultimately, the experiment suggests that, with the current setup, achieving an accuracy exceeding 90% requires setting the capture distance at approximately 1 meter, with the actual distance not exceeding 1.5 meters. There is a possibility that utilizing a more effective camera lens could potentially extend the capture distance.

VI. Conclusion

In summary, this project, developed as part of the Network Embedded Systems course at DTU, successfully implements a biometric authentication, based on a facial recognition system. The interconnected modules, comprising an ESP32-CAM for image capture and an ESP-WROOM-32 for visual feedback, demonstrate effective communication and real-time processing. The system achieves its objectives by employing HTTP for image transmission, MQTT for result dissemination, and external LEDs for visual representation, showcasing a seamless integration of hardware components and image processing capabilities.

Potential areas for development include the implementation of a more robust evaluation system based on a larger dataset of face recognitions, possibly incorporating polling mechanisms. Furthermore, enhancing the model's capability to recognize users from various angles and in less favorable environments could be achieved by feeding the model with multiple photos for each individual. These improvements would contribute to the system's adaptability and overall accuracy.

Moreover, enhancing the security of the system is imperative. Implementing secure versions of communication protocols, such as HTTPS and authenticated MQTT, would significantly increase the confidentiality of this system. This additional layer of security is essential to safeguard sensitive authentication process and ensure the integrity of the overall system.

References

[1] Python Software Foundation, *face-recognition 1.3.0*, visited 3.11.2023, https://pypi.org/project/face-recognition/, 2020.

[2] masoudr, *face_recognition Windows Installation Guide*, visited 3.11.2023, https://github.com/ageitgey/face_recognition/issues/175#issue-257710508, 2017

[3] mosquito.org, An open source MQTT broker, visited 16.11.2023, https://mosquitto.org/, 2023