Smart Home Security: Keeping Intruders at Bay with Your Face

Georghios Tziouliou
ID: 2412649
AC40001 Honours Project
BSc (Hons) Computing Science
University of Dundee, 2025
Supervisor: Prof. Oluwafemi Samuel

Abstract - *Conventional home security methods like keys or PINs face challenges of convenience and security. This project aimed to develop and evaluate a prototype smart home security system leveraging multi-factor authentication. The system, implemented primarily in Python on a Raspberry Pi 5, integrates facial recognition using dlib and a K-Nearest Neighbors classifier, with mandatory RFID verification via an MFRC522 reader. Liveness detection techniques (blink, motion, texture analysis) were incorporated to mitigate spoofing attacks using static images. A Flask-based web interface provides real-time video monitoring, system status, configuration options, and simulated manual door control. An integrated LCD offers local status feedback, indicating the simulated lock state ("Locked"/"Unlocked"). The results demonstrate the viability of combining facial recognition, RFID, and liveness detection for enhanced home access control logic. Future work includes enhancing liveness detection against video replays and streamlining the user enrolment process.*

1 Introduction

Traditional home entry methods, such as physical keys and PIN codes, present persistent challenges including the risk of keys being lost or stolen, PINs being forgotten or observed, and vulnerabilities like lock-picking leading to unauthorised access. The increasing adoption of smart home technology provides significant opportunities to develop more secure, conveniently managed, and monitored access control solutions [1]. Central to many advanced security approaches is biometric authentication, which uses unique physiological or behavioural characteristics for verification.

This project addresses the limitations of traditional methods by developing a prototype multi-factor smart home security system. The primary goal was to design a system capable of authorising access based on facial recognition – a prominent biometric modality – for an approved user, enhanced with mandatory secondary checks for increased security. Recognising the vulnerabilities of basic facial recognition, liveness detection is incorporated to verify the authenticity of the presented face, mitigating spoofing risks from static images [9]. Furthermore, RFID tag verification is integrated as a second authentication factor, requiring the simultaneous presence of both the correct live face and the corresponding physical token before access authorisation is indicated. The system simulates the control of a door lock, displaying the access status ("Locked" or "Unlocked") via an integrated LCD screen and a comprehensive web interface.

The objectives of this Project were:

1. To design and implement a system capable of real-time face detection and recognition using established computer vision libraries.

2. To integrate RFID technology (MFRC522) for reading unique identifiers from tags.

3. To develop and implement multi-factor authentication logic requiring both a successful live face match and a corresponding RFID tag match to indicate access authorisation, incorporating consecutive frame validation for robustness.

4. To incorporate basic liveness detection techniques (eye-blink, head movement, texture analysis) to enhance anti-spoofing capabilities against static images.

5. To create a web-based interface using Flask for system monitoring (live video feeds), status display (simulated lock state, RFID presence), and user control (manual simulated lock/unlock toggling, configuration adjustments).

6. To integrate an LCD display for local status updates, primarily indicating the simulated lock state.

7. To evaluate the prototype's functional correctness, recognition performance under defined conditions, and the effectiveness of the implemented security measures.

This report details the background research informing the project, the system specification derived from the objectives, the design choices made, the implementation process including challenges encountered, the evaluation methodology and subsequent results, and concludes with a critical appraisal and suggestions for future development.

2 Background and Literature Review

2.1 The Evolution of Residential Access Control

The proliferation of smart home technology has significantly reshaped residential security paradigms, with automated access control emerging as a focal point of innovation. Market trends reflect this shift; statistics from 2022 indicated over 12 million US households had adopted smart locks [1], while the Asia-Pacific smart door market demonstrated substantial growth, projected to expand significantly by 2033 [2]. This adoption signifies a shift away from traditional mechanical locks, which, despite their reliability, remain susceptible to physical compromise through key theft, lock picking, or unauthorised duplication. Modern electronic systems, often incorporating keypads, mobile applications, or biometric sensors offer advanced alternatives [3, 4].

2.2 Biometric Authentication and Facial Recognition

Biometric authentication leverages unique physiological or behavioural characteristics to verify identity. Common modalities include fingerprint scanning, iris recognition, voice patterns, and facial recognition. A typical biometric system pipeline involves sensor capture (e.g., camera, fingerprint scanner), feature extraction (isolating distinctive characteristics), template creation (a digital representation of the features), and matching (comparing a new capture against stored templates) [25].

Among biometric modalities, facial recognition has emerged as a particularly compelling option for access control due to its potential for convenient, hands-free authentication [5]. The technology aims to verify or identify individuals by analysing their facial features captured via a camera. The underlying techniques have evolved significantly from earlier approaches like Eigenfaces [6] or Local Binary Patterns Histograms (LBPH), which were explored preliminarily in this project (Section 5.1) but found lacking in robustness. Current state-of-the-art systems are predominantly based on deep learning, particularly Convolutional Neural Networks (CNNs), which have driven substantial improvements in recognition accuracy under varying conditions [7]. These models automatically learn hierarchical features from facial images. Libraries such as dlib encapsulate powerful, pre-trained deep neural network models, like the ResNet-based architecture utilised in this project [8]. These models excel at generating robust, high-dimensional facial embeddings (typically 128-d feature vectors) that are relatively invariant to moderate changes in pose and illumination, making them highly suitable for accurate identification tasks (FR3, FR4).

While facial recognition is integrated into various applications, from mobile devices to security at some airports [19], [20], its widespread deployment for general residential security remains limited compared to simpler smart locks [3, 4]. This disparity arises partly from the significant challenges discussed below, including security vulnerabilities (spoofing), performance variations, and critical ethical considerations, which necessitate careful system design for home use.

2.3 Security Challenges: Presentation Attacks (Spoofing) and Liveness Detection

Despite advancements in recognition accuracy, standard 2D facial recognition systems face a critical vulnerability: Presentation Attacks (PAs), commonly known as spoofing [9]. An attacker can potentially bypass these systems by presenting a non-live artefact, such as a printed photograph, a video replay on a screen, or even a 3D mask of an authorised user. Consequently, verifying that the presented face is genuinely live and present – known as Liveness Detection or Presentation Attack Detection (PAD) – is essential for secure deployment.

Common liveness detection techniques analyse cues indicative of physiological life:

- Eye Blink Detection: Live individuals blink naturally. This can be detected by monitoring the Eye Aspect Ratio (EAR) [10], calculated from specific eye landmarks. A rapid drop below a threshold signifies a blink (FR7b).
- Head/Facial Motion: Subtle, involuntary head movements are characteristic of live subjects. Analysing the variance of facial position over time can distinguish live movement from the unnatural stillness of a photograph (FR7c).
- Texture Analysis: Live skin possesses a detailed texture different from prints or screens. Analysing pixel intensity variance in the face region can help differentiate live skin from spoofs (FR7d).
- Other Methods: More advanced techniques include analysing light reflection, thermal signatures, blood flow, or using specialised 3D cameras [11], though these were beyond the scope

of this project due to complexity and hardware requirements.

Research suggests that combining multiple liveness indicators, as implemented in this project (blink, motion, texture), generally yields more robust anti-spoofing capabilities compared to relying on any single technique [11]. This project adopts such a multi-model approach using computationally lightweight checks suitable for the Raspberry Pi platform (FR4, FR7, FR13).

2.4 Enhancing Security: Multi-Factor Authentication (MFA)

To further bolster security against potential failures in recognition or liveness detection, this project incorporates a second authentication factor: Radio-Frequency Identification (RFID). Multi-Factor Authentication (MFA) requires a user to provide two or more distinct credential types (factors) [13]. This project implements a two-factor approach combining:

1. Biometric Factor: The recognised live face ("something the user is").

2. Possession Factor: A physical RFID tag/card ("something the user has").

Passive RFID systems, using readers like the MFRC522 [12], provide an inexpensive, power-efficient method for wirelessly reading stored identifiers (FR5). By requiring the successful validation of both the live face and the corresponding physical RFID tag before granting access (FR6), the system significantly increases security. Compromising this requires defeating both facial analysis/liveness and obtaining the physical tag, making unauthorised access considerably harder than attacking a single-factor system [13].

2.5 Ethics, Privacy, and Bias Considerations

The deployment of facial recognition necessitates careful ethical consideration [14]. Key concerns include:

- Privacy: Facial images are sensitive personal data. Collection and analysis raise concerns about misuse and surveillance. Centralised storage increases data breach risks.

- Data Security: Secure handling of biometric templates (embeddings in this case) is crucial.

- Algorithmic Bias: Systems can exhibit performance disparities across demographics due to unrepresentative training data [21], raising fairness concerns.

- Accuracy and Errors: False Acceptance Rate (FAR) and False Rejection Rate (FRR) are inherent. While MFA mitigates the impact of single-factor errors, understanding these rates is of significant importance.

- Intrusiveness: The act of face scanning can be perceived as intrusive by some users [26].

This project mitigates some concerns via:

- Data Minimisation: Real-time processing; only embeddings stored during initial training.

- Local Processing: Core operations run on the Pi, reducing cloud reliance and associated risks. (This local design limits attack vectors compared to cloud systems).

- MFA: Reducing sole reliance on the face biometric.

However, ethical frameworks and user consent remain paramount [14]. Responsible implementation requires transparency and adherence to privacy principles.

2.6 Empirical Use Cases and Related Work

Facial recognition is applied in diverse sectors, often facing public debate. Retail applications for identifying offenders [26] and Amazon's use of palm scanning [27] contrast with security applications like the TSA's airport screening [27], all highlighting the tension between convenience/security and privacy rights discussed in media outlets (The Sun, 2024) [28].

Within academic research relevant to this project, Z. Leyu et al. [15] combined face, fingerprint, and RFID on similar hardware. Ghafoor et al. [16] focused on IoT face recognition with user tracking. The distinct contribution of the current work lies in implementing and evaluating the specific combination of lightweight liveness checks (EAR, motion, texture) alongside Face+RFID MFA, plus the consecutive frame validation, tailored for the Raspberry Pi platform. This explores a practical configuration not explicitly detailed in the reviewed similar systems [15, 16]. Further research explores alternative storage and processing paradigms like on-device processing within secure hardware elements or differential privacy techniques [22, 24] to further enhance privacy beyond the local processing model used here.

2.7 Research Context and Project Aims

This review establishes the context: growing demand for convenient biometric access control faces significant security (spoofing) and ethical challenges. There is a practical need to evaluate integrated systems that layer multiple security techniques (MFA, Liveness) on accessible, resource-constrained platforms like the Raspberry Pi.

Therefore, this project aimed to address the following research questions:

1. How effectively does the combination of EAR blink, motion variance, and texture variance checks mitigate static photo presentation attacks in this Raspberry Pi-based system, especially when coupled with consecutive frame validation?

2. What is the practical performance (recognition accuracy for a known subject, processing speed/frame rate) of the integrated dlib/KNN + RFID + Liveness pipeline on a Raspberry Pi 5?

3. Is the proposed multi-factor system, including the web interface and LCD feedback, a feasible and functional proof-of-concept demonstrating enhanced simulated access control logic compared to single-factor approaches?

The aims and objectives derived from these questions are detailed in Section 1. By implementing and evaluating this specific configuration, the project contributes practical insights into deploying layered security on embedded systems. It is important to note that for the purposes of the project, the “Other” tag/face was created in order to avoid uploading biometric data due to ethical reasons. Wherever “Other” was utilised throughout the project it was solely for demonstration purposes and testing, and can be removed if the software and hardware were to be released commercially. Since the software was trained and tested primarily utilising the developer’s live face (No images were saved during the tests or while running the tests) as well as photographs of the actor Robert Downey Jr. (which were ethically acquired from the Celebrity Face Image Dataset [17]); “Other” refers to any other face not trained in the model.

3 Specification

3.1 Functional Requirements (FR)

FR1. The system must capture a real-time video stream from a connected USB camera.

FR2. The system must detect human faces within the captured video frames using dlib's HOG detector.

FR3. The system must compute 128-dimensional face embeddings for detected faces using dlib's pre-trained ResNet model.

FR4. The system must classify computed face embeddings against a pre-trained K-Nearest Neighbors (KNN) model to identify known registered users or classify them as "Other" or "Unknown" based on a defined confidence threshold (50%).

FR5. The system must continuously poll for and read data payloads (expected to be user names or "Other") from passive 13.56MHz RFID tags using the MFRC522 reader module via SPI communication.

FR6. The system must implement multi-factor authentication (MFA) logic: Access authorisation shall only be granted if a face recognised as a known user (or "Other", if enabled) passes liveness checks AND an RFID tag containing the exact matching name ("Other" tag for "Other" face) is simultaneously detected.

   a. The system must implement liveness detection checks for detected faces, including:

   b. Eye-blink detection using the Eye Aspect Ratio (EAR) thresholding (< 0.3).

   c. Head movement detection based on the variance of normalised face center coordinates over a short time window (variance between 0.00005 and 0.02).

   d. Texture analysis based on the variance of the grayscale face region-of-interest (variance > 400).

FR7. Failure of any enabled liveness check must prevent access authorisation, regardless of face or RFID match.

FR8. The system must maintain an internal state representing a simulated door lock ("Locked" or "Unlocked").

FR9. The system must display the current simulated lock status and specific access messages (e.g., "Access Granted", "Door Unlocked", "Door Locked") on a connected 16x2 LCD display via GPIO.

FR10. The system must provide a web interface accessible over the local network, featuring:

   a. A live MJPEG video stream annotated with face bounding boxes, recognition results (name, confidence), liveness status, and RFID match status.

b. A secondary, parallel live MJPEG video stream showing annotations without liveness analysis results for comparison.

c. Display of the current simulated lock status.

d. Display of the currently detected RFID tag data.

e. Web controls to manually toggle the simulated lock state between "Locked" and "Unlocked".

f. Web controls to toggle the enforcement of liveness detection checks.

g. Web controls to toggle the allowance of access for the "Other" face category when presented with a corresponding "Other" RFID tag.

FR11. A separate Python script (model_create.py) must be provided to train the KNN face recognition model using images stored in a specified directory structure (DATA_PATH/PersonName/image.jpg).

FR12. The web interface must provide functionality to write user-specified data (typically names) to RFID tags via a modal dialog.

FR13. The liveness detection mechanisms must demonstrate effectiveness in rejecting access attempts using static, printed photographs of registered users.

FR14. Status feedback via the web UI (status text, indicators, video annotations) and LCD must be clear, timely, and accurately reflect the system's state.

3.2 Non-Functional Requirements (NFR)

NFR1: The face detection, recognition, and liveness pipeline should process frames at a rate perceived as near real-time by the user on the Raspberry Pi 5 target hardware.

NFR2: The web interface should load and respond to user interactions within acceptable time limits (e.g., < 2 seconds) on common web browsers (Chrome, Firefox) within the local network.

NFR3: The system should demonstrate reasonable robustness to typical indoor lighting variations, although optimal performance is expected under consistent, moderate lighting.

3.3 Development Methodology

An iterative development approach was adopted. This allowed for modular development and testing of individual components (RFID reading, face detection, KNN training, liveness checks, web interface elements, LCD control) before integrating them into the complete system. Each iteration involved implementation, debugging, and functional testing, allowing for early identification and resolution of issues, particularly those related to hardware interfacing and multi-threading on the Raspberry Pi. This approach provided flexibility to refine components like liveness detection thresholds based on initial testing results.

4 Design

4.1 Hardware Platform

- A Raspberry Pi (Model Raspberry Pi 5 8GB) was chosen as the core processing unit. Justification: low cost, sufficient processing power for the required tasks (with optimizations), extensive GPIO capabilities for hardware interfacing (RFID, LCD, Lock), large community support, and suitability for embedded/IoT projects.

- A standard USB Webcam was used for video input.

- An MFRC522 RFID Reader/Writer module was selected due to its widespread availability, low cost, and existing Python libraries (mfrc522). It operates at 13.56MHz and communicates via SPI.

- Connected directly via GPIO pins [Pins: LCD_RS=26, LCD_E=19, LCD_D4=13, LCD_D5=6, LCD_D6=5, LCD_D7=21, LED_ON=15]. The physical connections between the Raspberry Pi's GPIO header, the LCD module's parallel interface pins, and the MFRC522's SPI pins were facilitated using jumper wires and a standard solderless breadboard, primarily for distributing power (VCC/GND) and organising connections, as illustrated in Figure 1. The `lgpio` library was used for GPIO control as it is the modern successor to RPi.GPIO often recommended for newer Raspberry Pi OS versions.

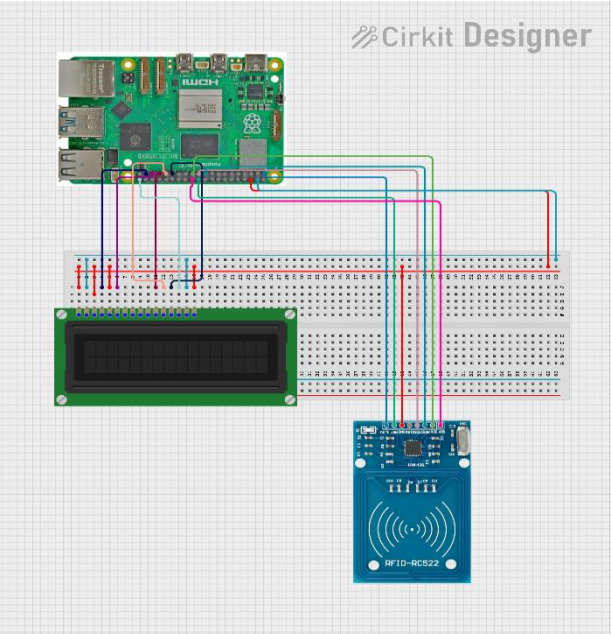

Figure 1: Hardware Wiring Diagram showing Raspberry Pi 5 connections to the parallel 16x2 LCD and SPI MFRC522 RFID Reader via a breadboard (created using cirkit Designer [29]).

4.2 Software and Libraries

4.2.1 Operating System: Raspberry Pi OS Bookworm (64-bit).

Primary Language: Python 3. Justification: excellent library support for computer vision (OpenCV), machine learning (scikit-learn), web development (Flask), and hardware interfacing; relatively easy to learn and utilisable to design a prototype.

4.2.2 Computer Vision:

OpenCV (cv2): Used for core image/video handling (reading frames, color conversion, drawing shapes/text).
dlib: Chosen for its high-quality pre-trained models for:
Face Detection (HOG-based frontal face detector).
Facial Landmark Prediction (68-point model shape_predictor_68_face_landmarks.dat).
Face Recognition (ResNet-based deep learning model dlib_face_recognition_resnet_model_v1.dat generating 128-d embeddings). Justification: dlib's models are known for good accuracy and are relatively efficient compared to some other deep learning frameworks when running on constrained devices like the Pi, although still computationally intensive. The choice of dlib's HOG detector represented a balance between performance and accuracy on the CPU-limited Raspberry Pi 5; while deep learning detectors like SSD or MTCNN might offer higher detection rates under challenging conditions, their computational cost often necessitates GPU acceleration not utilised in this project. The ResNet-based embedding model was selected for its proven performance in generating robust face descriptors resistant to moderate variations in pose and illumination, forming a strong basis for the subsequent KNN classification (FR3).

4.2.3 Machine Learning:

Scikit-learn (sklearn): Used for the K-Nearest Neighbors (KNeighborsClassifier) algorithm. Justification: KNN is a simple yet often effective non-parametric classifier that works well with dlib's 128-d embeddings, especially for smaller datasets typical in personalised home security. It's easy to implement and update. Alternatives considered: SVM (might offer better boundaries but potentially slower prediction), but KNN was chosen for simplicity and speed of training/prediction in this context. The KNN algorithm operates by storing all training face embeddings and their corresponding labels. During prediction (FR4), the Euclidean distance is calculated between the input face embedding and every stored training embedding. The algorithm identifies the 'k' closest embeddings (the 'k-nearest neighbours', with k=3 chosen for this project as a common value balancing stability and sensitivity). In this implementation, the prediction is weighted by the inverse of the distance to these neighbours, meaning closer neighbours have more influence on the final classification. The confidence score is derived from the distance to the single nearest neighbour (d_min), calculated as max(0, min(100, 100 - (d_min * 100))). This provides an intuitive measure: smaller distances yield higher confidence. An empirically determined threshold of 50% was used to differentiate between confident matches to known individuals and low-confidence matches classified as "Other".

NumPy: Essential for numerical operations, array manipulation (embeddings, coordinates).

4.2.4 Web Framework:

Flask: A lightweight microframework chosen for its simplicity, flexibility, and ease of setting up API endpoints and serving the web interface. Suitable for smaller to medium-sized applications. Alternatives like Django were considered too heavyweight for this project's scope.

4.2.5 Hardware Interfacing:

mfrc522: A specific library for interacting with the MFRC522 reader via SPI.

lgpio: Used for LCD control via direct GPIO access. This library was preferred over the older RPi.GPIO as it offers potentially more precise timing control, which can be beneficial when driving parallel interfaces like the HD44780-compatible LCD according to its specific command/data setup and hold times (FR9), and it is generally recommended for newer Pi models and OS versions.

4.2.6 Data Serialisation:

pickle: Used to save and load the trained KNN model and the label dictionary. Justification: Simple Python-native way to serialise objects. Note: Pickle files can pose security risks if loading untrusted files, but considered acceptable for this prototype context where the model files are generated locally.

4.3 System Architecture

The system employs a multi-threaded architecture executed within the main Flask application (CameraAndWebsite.py).

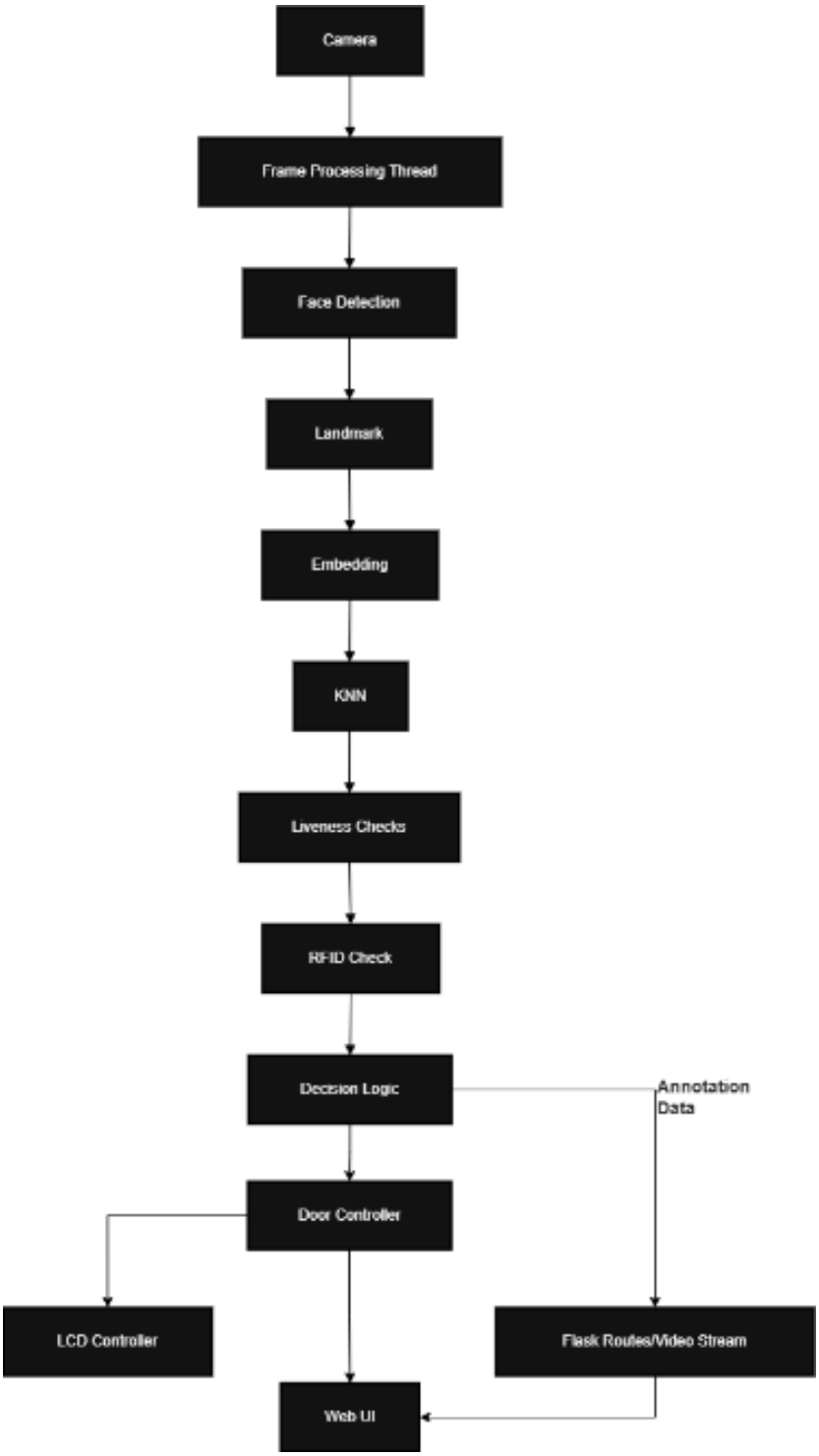

Figure 2: System Architecture

Main Thread: Runs the Flask web server, handling HTTP requests for the UI, controls, and status updates.

Frame Processing Thread (process_frames): Continuously captures frames from the camera, performs all recognition and liveness steps (calling recognize_face), and updates shared frame buffers (output_frame_with_liveness, output_frame_without_liveness) protected by a lock (threading.Lock) for thread-safe access by the streaming generators.

RFID Scanning Thread (continuous_read in rfid_routes.py): Continuously polls the RFID reader non-blockingly, managing card presence/absence state and updating the DoorController. Uses a threading.Lock (reader_lock) to prevent conflicts between continuous scanning and manual read/write operations.

Match Status Reset Thread (match_status_reset): Periodically checks and resets the global match notification flag (latest_match) after a timeout.

LCD Message Thread (_message_loop in lcd_controller.py): Processes messages from a queue to write to the LCD asynchronously.

Thread Interaction and Data Flow: The multi-threaded design necessitates careful management of data flow between components (visualised in Figure 2). The Frame Processing Thread is central, consuming raw frames and producing annotated frames for the web streams. It interacts with the DoorController to retrieve the current RFID status (read by the RFID Scanning Thread) during its RFID Check phase. Based on the Decision Logic outcome, it may call methods on the DoorController to change the simulated lock state. The DoorController, in turn, pushes status update messages onto the queue.Queue monitored by the LCD Message Thread. The Web UI interacts primarily with the Main Thread (Flask) via HTTP requests. Status queries (like /door_status) prompt Flask to read the current state from the DoorController, while video streams are served directly from the frame buffers updated by the Frame Processing Thread. Locks (threading.Lock) are used to protect concurrent access to shared resources: one lock protects the frame buffers, while another (reader_lock) protects the SPI bus used by the MFRC522, ensuring the background scan doesn't conflict with manual read/write requests from the web UI. This separation allows computationally intensive vision processing, continuous peripheral polling (RFID), potentially slow hardware output (LCD), and web request handling to occur concurrently, maximising system responsiveness (NFR1, NFR2).

4.4 Key Design Decisions & Trade-offs

Face Recognition Approach (dlib Embeddings + KNN): The core recognition pipeline utilises dlib's pre-trained deep learning model (ResNet-based) to generate 128-dimensional face embeddings, followed by a K-Nearest Neighbors (KNN) classifier trained on these embeddings. This approach was chosen over alternatives like training a deep learning classifier from scratch (computationally prohibitive for this project scope) or using older methods like Eigenfaces/LBPH (tested initially, see Section 5.1, but found lacking in accuracy). Dlib's model provides high-quality embeddings with reasonable computational cost on the Raspberry Pi 5. KNN was selected as the classifier due to its simplicity, ease of implementation with scikit-learn, and good performance with relatively small training datasets per person, typical for personalised security. It operates by finding the 'k' (in this case, 3) closest known face embeddings in the 128-d space to the input embedding (using Euclidean distance) and predicting the class based on distance weighting. The confidence score is derived directly from the distance to the nearest neighbour, providing an intuitive measure of similarity. While Support Vector Machines (SVMs) were considered, potentially offering more robust decision boundaries, KNN was preferred for its faster prediction time and simpler conceptual model within the project timeframe.

Liveness Detection Methodology: Given the vulnerability of 2D face recognition to presentation attacks, incorporating liveness detection was crucial. A multi-modal approach combining three distinct, computationally inexpensive checks was implemented:

Eye Blink Detection: Utilises the Eye Aspect Ratio (EAR), calculated from specific facial landmarks around the eyes provided by dlib's shape predictor [10]. A significant drop in the EAR value below a threshold (0.3) indicates a potential blink. This leverages a natural physiological signal often absent in static spoofs.

Motion Detection: Tracks the normalised centre coordinates of the detected face over a short temporal window (30 frames). The variance of these coordinates is calculated; significant head or camera movement results in high variance, while unnatural stillness (like a photo) results in very low variance. The check requires the variance to fall within an empirically determined range (0.00005 to 0.02) to be considered 'live' movement.

Texture Analysis: Exploits the assumption that live skin possesses more complex texture than printed photos or screens. A simple implementation calculates the variance of pixel intensities within the grayscale face Region of Interest (ROI). A variance exceeding a threshold (400) suggests sufficient texture complexity indicative of a live face.

This combination aims to provide broader protection than any single method. The thresholds were determined empirically through testing with live faces and printed photos under varying conditions, representing a trade-off between security (rejecting spoofs) and usability (avoiding false rejection of live users). More sophisticated techniques (e.g., frequency analysis, deep learning liveness models) were considered but deemed too complex or computationally expensive for implementation on the target hardware within the project scope.

Multi-Factor Authentication (MFA) Logic: The system enforces a strict MFA policy requiring simultaneous validation of: 1) Recognition of a known identity (or "Other", if permitted) above the confidence threshold, 2) Successful liveness verification, and 3) Detection of an RFID tag whose stored data matches the recognised identity. This layered approach, detailed in door_control.py's check_face_match function, provides substantially higher security than relying solely on face recognition or RFID [13]. The special handling for the "Other" category (requiring an "Other" tag and consistent sighting) adds a control mechanism for potentially unknown but live individuals.

Consecutive Frame Validation: To enhance robustness against momentary detection errors or transient false positives (especially in liveness checks), a consecutive frame validation mechanism was implemented. The system requires a face to meet all recognition, liveness, and RFID criteria for a predefined number of consecutive frames (CONSECUTIVE_FRAMES_THRESHOLD = 5) before triggering the unlock authorisation. This design choice prioritises reducing false acceptances caused by fleeting conditions over instantaneous response, adding a layer of temporal consistency checking to the decision logic.

Web Interface Technology (Flask & MJPEG): Flask was chosen as the web framework due to its lightweight nature, simplicity, and ease of integrating Python backend logic with web interfaces, making it suitable for rapid prototyping on the Raspberry Pi. Alternatives like Django were considered unnecessarily complex for the project's requirements. Motion JPEG (MJPEG) was selected for video streaming (generate_frames function). While less bandwidth-efficient than modern protocols like WebRTC or HLS, MJPEG is straightforward to implement using OpenCV and Flask, directly embedding the stream in an HTML <img> tag, simplifying client-side requirements. Providing dual streams (with and without full annotations) allows for direct visualisation of the processing effects.

State Management: For this single-process prototype, system state (e.g., latest_match, system_settings, door_controller object) is managed using global variables and shared objects within the main Flask application (CameraAndWebsite.py). Concurrency control for shared resources accessed by multiple threads (e.g., video frame buffers, RFID reader) is handled using Python's threading.Lock. While sufficient for the prototype, a production system would necessitate a more robust approach, potentially involving a database or dedicated state management service, especially if scaling or distributing components.

RFID Data Handling: The system currently relies on writing the exact user name (matching the face recognition label) or the literal string "Other" to the RFID tag data payload. This simplifies the matching logic (check_face_match) but couples the tag data directly to the face label. A more robust approach, suggested in Future Work (Section 10), would involve storing unique RFID tag UIDs in a database and associating them with user profiles.

Simulated Lock Mechanism: To focus on the core authentication logic and software integration within the project scope, a physical lock mechanism was not implemented. The DoorController manages a logical lock state, and access authorisation results in updating this state and displaying it via the LCD and web UI, effectively simulating the outcome of controlling a physical door. This allowed testing the complete authentication pipeline without the added complexity of interfacing with specific lock hardware.

5 Implementation and Testing

The development of the Smart Home Security system followed an iterative process, starting with basic component testing and progressively integrating features towards the final multi-factor authentication system running on the Raspberry Pi 5. Python 3 was used as the primary development language, leveraging libraries such as OpenCV, dlib, scikit-learn, Flask, lgpio, and mfrc522.

5.1 Initial Exploration: Face Detection and Recognition Concepts

The initial phase focused on understanding fundamental face detection and recognition techniques using OpenCV. Basic face detection was first explored using OpenCV's built-in Haar cascade classifiers (haarcascade_frontalface_default.xml), initially tested on static images and then adapted for live video feed using cv2.VideoCapture on a development machine (as seen in variations like face_detection_no_model.py).

An early attempt at face recognition utilised OpenCV's Local Binary Patterns Histograms (LBPH) recogniser (cv2.face.LBPHFaceRecognizer_create), demonstrated in face_detection.py. This involved training a model (face_recognizer_model.yml) and attempting predictions. However, limitations in the robustness and accuracy of

LBPH compared to modern deep learning methods became apparent during these initial tests, prompting a shift towards a more advanced approach.

5.2 Transition to dlib and KNN for Enhanced Recognition

Recognising the need for higher accuracy, the project transitioned to using the dlib library. Dlib's pre-trained models offered significant advantages:

- A robust HOG-based frontal face detector (dlib.get_frontal_face_detector).
- A precise 68-point facial landmark predictor (shape_predictor_68_face_landmarks.dat).
- A deep learning model (dlib_face_recognition_resnet_model_v1.dat) capable of generating discriminative 128-dimensional face embeddings.

The model_create.py script was developed to automate the training process for a K-Nearest Neighbors (KNN) classifier based on these dlib embeddings. This script performs the following steps:

Recursively scans the specified DATA_PATH. For this project, subdirectories corresponding to Denzel Washington, Hugh Jackman, Robert Downey Jr., and Scarlett Johansson within the "Celebrity Face Image Dataset" sourced from Kaggle [17] were used to provide training images. Images within this dataset are organized into subdirectories named after each celebrity. Approximately 100 images were selected for each of these four individuals to train the model.

For each selected training image, it detects the face, finds landmarks, and computes the 128-d embedding using the dlib models.

It collects all embeddings and assigns numerical labels (0 for Denzel Washington, 1 for Hugh Jackman, etc.), storing the label-to-name mapping in label_names.pkl.

Using scikit-learn, it splits the collected data (embeddings and labels for all four actors) into training (typically 80%) and testing (20%) sets, although the evaluation in Section 6 focused on a different test set.

It trains a KNeighborsClassifier (configured with k=3, Euclidean distance, and distance weighting) on the training embeddings for all four individuals.

The trained multi-class KNN model is saved as face_recognizer_model.pkl.

This new dlib/KNN approach was validated first by testing predictions on single, unseen images (model_test_single_image.py) and then integrated into a live camera feed script (model_camera.py), demonstrating significantly improved recognition performance compared to the earlier LBPH experiments. A confidence score, derived from the distance to the nearest neighbour in the KNN model, was introduced, with a threshold (empirically set at 50%) used to classify low-confidence matches as "Other".

5.3 Hardware Integration and Porting to Raspberry Pi 5

With the core recognition algorithms validated on a development machine, the project transitioned to the target hardware platform: a Raspberry Pi 5 (8GB RAM). The first step involved installing the operating system. Raspberry Pi OS Bookworm (64-bit version) was chosen for its up-to-date kernel and library support. The OS was flashed onto a microSD card using the official Raspberry Pi Imager tool, and initial system setup (network configuration, user accounts) was completed.

OS & Dependencies: Subsequently, the necessary Python 3 libraries were installed directly into the system's Python environment. All required libraries, including OpenCV (opencv-python), dlib, NumPy, scikit-learn, Flask, lgpio, and mfrc522, were installed using the pip package installer (e.g., pip install opencv-python dlib ... or sudo pip install ... if required for system-wide access). While using a virtual environment is generally recommended for managing dependencies, direct system installation was employed for simplicity in this project setup.

Camera: The USB webcam was connected, and the model_camera.py script was tested on the Pi to verify basic functionality and assess initial performance.

LCD Integration: Connecting the 16x2 LCD via direct GPIO required careful mapping of the Pi's BCM pin numbers to the LCD's RS, E, D4-D7, and LED_ON pins (as listed in Section 4.1 and shown in Figure 1). Debugging involved ensuring the lgpio library had correct permissions (sudo usermod -a -G gpio <username>) and iteratively testing basic commands (_lcd_byte) to confirm correct character display and timing, resolving initial issues with garbled text due to incorrect enable pulse timing (E_PULSE, E_DELAY). The asynchronous message queue proved essential early on, as direct calls to LCD methods from the main loop caused noticeable stuttering in the video feed.

RFID Integration: Enabling the SPI interface on the Raspberry Pi (via raspi-config) was the first step. Verifying the wiring between the Pi's SPI0 pins (MOSI, MISO, SCLK, CE0) and the MFRC522 module's corresponding

pins (SDA often maps to CE0/CS) was critical. Initial tests with rfid_script.py sometimes failed due to poor connections on the breadboard or incorrect pin assignments. Using the mfrc522 library simplified the read/write logic, abstracting the low-level SPI commands.

5.4 Liveness Detection Implementation

To mitigate spoofing risks inherent in 2D facial recognition [9], several liveness detection techniques were implemented and integrated into the recognize_face function, supported by the LivenessDetector and FaceLivenessState classes (see CameraAndWebsite.py for full code). These checks aim to verify the presence of physiological signs expected from a live person.

Eye Blink Detection (EAR): This method leverages the Eye Aspect Ratio (EAR), a metric derived from the positions of 6 specific facial landmarks around each eye, as identified by the dlib shape predictor (sp). The EAR is calculated using the formula proposed by Soukupová and Čech [10]: EAR = (||p2 - p6|| + ||p3 - p5||) / (2 * ||p1 - p4||), where p1-p6 are the landmark points of one eye. A sudden drop in EAR below an empirically determined threshold (set to 0.3 in this implementation) signifies a potential blink. The FaceLivenessState class maintains a counter (total) for detected blinks associated with a tracked face ID. The check passes if at least one blink (total > 0) has been registered for the face recently. Figure 3 shows the relevant landmarks indicated by red dots.

Figure 3: Eye Landmarks indicated by red dots (The red dots illustrated in the figure were drawn manually to demonstrate the landmarks of the eye).

Motion Detection: This check analyses the stability or subtle movement expected from a live person compared to a static photo. The normalised (x, y) center coordinates of the detected face bounding box are stored over the last 30 frames in the motion_history list within FaceLivenessState. The variance of both the x and y coordinates in this history is computed. Very low variance suggests a static image, while very high variance might indicate erratic tracking or rapid movement not typical of an authentication pose. The check passes only if the combined variance falls within an empirically tuned range (0.00005 to 0.02). Tuning this range involved observing variance values during tests with both static photos and live subjects making slight, natural head movements.

Texture Analysis: Based on the premise that live skin has a more complex texture than printed paper or screen displays, this simple check calculates the variance of pixel intensities within the detected face's Region of Interest (ROI) after converting it to grayscale. A higher variance suggests more complex texture. The check passes if the variance exceeds a threshold (set to 400). While basic, this can help differentiate very smooth spoof attempts from live faces, especially under consistent lighting.

Integration and Challenges: The final liveness status (is_live) is determined by combining the results of these three checks using logical AND (all must pass). A major implementation challenge was the empirical tuning of the thresholds for EAR, motion variance, and texture variance:

EAR Tuning: The 0.3 threshold generally worked well, but very short glances at the camera sometimes failed to register a blink within the processing window, leading to false negatives. Extending the observation window slightly was considered but rejected to maintain responsiveness.

Motion Tuning: The range [0.00005, 0.02] required careful adjustment. Lowering the minimum below 0.00005 sometimes allowed extremely stable photos (if held very still) to pass, while increasing the maximum above 0.02 risked rejecting users who naturally moved their head slightly during presentation. Tests involved printing variance values for known spoofs and live sessions to find the optimal separation.

Texture Tuning: The variance threshold (400) was sensitive to lighting. Under bright, flat lighting, even live faces could sometimes fall below this threshold. Conversely, photos with printing artefacts or glare could occasionally exceed it. This highlighted it as the least reliable of the three checks under variable conditions.
The final is_live status depends on all enabled checks passing. This AND logic prioritises security (rejecting if any sign of non-liveness is found) at the cost of potentially higher false negatives if one check fails inappropriately. The LivenessDetector class using get_face_id (based on face position hash) ensures state is maintained per-face, and cleanup_old_faces prevents indefinite state accumulation.

```
if person_name != "Unknown":
    if door_controller and door_controller.is_locked():
        current_rfid = door_controller.get_rfid_name()
        has_rfid = current_rfid is not None and len(str(current_rfid).strip()) > 0

        if has_rfid:
            rfid_matched = door_controller.check_face_match(person_name)

        passes_liveness_check = not system_settings["use_liveness"] or is_live

        if rfid_matched and passes_liveness_check:
            if person_name == "Other":
                should_unlock = system_settings["allow_other"]
            else:
                should_unlock = True

if should_unlock:
    match_should_unlock_this_frame = True
    unlock_info = (person_name, confidence)
    color = (0, 255, 0)
```

Figure 4: Core MFA Decision Logic Snippet from recognize_face Function (CameraAndWebsite.py).

5.5 RFID Integration and Multi-Factor Authentication (MFA) Logic

With individual components tested, the focus shifted to integrating RFID into the main application and implementing the MFA logic.

RFID Controller: The RFIDController class (rfid_routes.py) was developed to manage RFID operations within the Flask application context. It includes the continuous_read method running in a background thread, using read_no_block() and a threading.Lock(reader_lock) for non-blocking scanning while allowing manual operations. It interacts with the DoorController to update the currently detected card state. The interaction with DoorController involves the continuous_read thread calling door_controller.set_rfid_card(card_data) upon detecting a new card and door_controller.clear_rfid_card() upon card removal.

Door Controller: The DoorController class (door_control.py) was implemented to manage the simulated lock state (door_status['locked']) and the associated logic. It stores the current_rfid_card data provided by the RFIDController. The check_face_match function is central, performing the case-insensitive comparison. Its use of self.face_name_matches to require multiple sightings (>=3) for the "Other" category was implemented to prevent temporary misclassifications triggering an "Other" match inappropriately.

MFA Implementation: The core MFA logic resides within the recognize_face function. After a face is recognised (KNN) and passes liveness checks, the system retrieves the current RFID data via door_controller.get_rfid_name(). It then calls door_controller.check_face_match(), which performs a case-insensitive comparison between the face name and RFID data (with special handling for the "Other" category requiring multiple consistent sightings). Only if all conditions (Recognition Match + Liveness Pass + RFID Match) are met is the door_controller.unlock_door() method invoked. The sequence within recognize_face is critical: KNN prediction first establishes potential identity, liveness checks validate authenticity, then the current RFID state is retrieved from DoorController for the final check_face_match call. Only if this sequence completes successfully does it proceed to call door_controller.unlock_door().

5.6 Web Interface and Final System Integration (CameraAndWebsite.py)

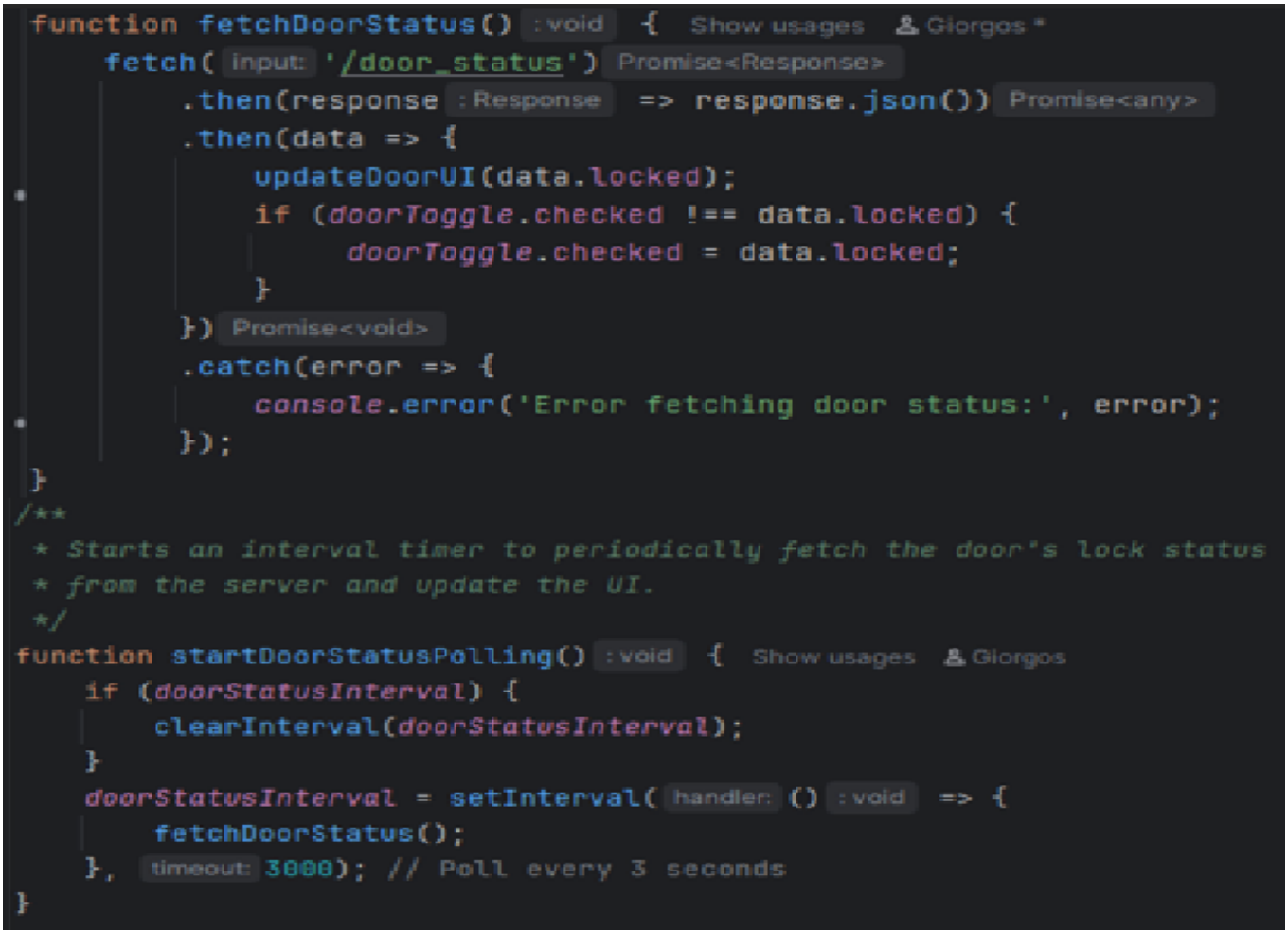

Figure 5: Example code snippet JavaScript for Polling Door Status (script.js).

The final stage involved creating the web interface and integrating all components into a cohesive Flask application.

Flask Backend: The CameraAndWebsite.py script sets up the Flask app, defines routes for the main page (/), static files (/static/...), API endpoints (/door_status, /toggle_door, /current_rfid_name, /match_status, /system_settings, /update_settings, RFID routes), and the MJPEG video streams (/video_feed_with_liveness, /video_feed_without_liveness). API endpoints like /door_status directly query the door_controller.door_status dictionary, while control endpoints like /toggle_door call the relevant door_controller methods (lock_door/unlock_door). The /update_settings endpoint modifies global variables (system_settings) used by the recognize_face function.

Video Streaming: The generate_frames function reads processed frames (with annotations determined by the recognize_face function's outputs) from global variables (output_frame_with_liveness, etc.). A threading.Lock ensures thread-safe access to these frame buffers between the processing thread and the multiple generator instances serving the streams. Generating two streams required calling recognize_face twice per frame capture, once with liveness enabled for the fully annotated stream and once conceptually disabling liveness checks just for annotation purposes on the comparison stream (though actual processing might still occur, but it is not used for the final decision in that path). This added computational overhead but was deemed valuable for demonstration (FR10b).

Frontend: index.html provides the page structure, styles.css defines the visual appearance (including responsive design), and script.js handles client-side interactivity. JavaScript uses setInterval and the fetch API to poll the status endpoints asynchronously, updating the door status indicator, RFID display, and showing match notifications dynamically. Event listeners trigger API calls for manual controls and settings changes. The script.js uses setInterval to poll status endpoints every [e.g., 2-3 seconds]. The fetch API handles asynchronous requests. JavaScript updates the DOM elements (statusDot, statusText, rfidResult) based on the JSON responses from the Flask API. The match notification uses a dynamically created/shown div element.

Threading: The application relies heavily on multi-threading: one thread for frame processing (process_frames), one for continuous RFID scanning (continuous_read), one for LCD messages (_message_loop), one for resetting match status (match_status_reset), and the main Flask thread handling web requests. Managing shared resources (frame buffers, RFID reader, door_controller state) required careful use of threading.Lock where necessary.

Integration Challenges: Ensuring smooth interaction between the background processing thread, the Flask request handlers, and the hardware control threads (RFID, LCD) was complex. Debugging involved tracing data flow across threads and identifying potential race conditions or blocking operations. For instance, ensuring the LCD writing didn't block the main loop was addressed by its dedicated queue and thread. MJPEG stream stability required ensuring the frame processing could keep up and locks were managed efficiently.

5.7 Development Testing

Testing throughout the development process was primarily integration-focused. Standalone scripts (rfid_script.py, model_test_single_image.py, etc.) were used to verify individual components before integration. Key integration tests included: Verifying the MFA logic with various combinations of correct/incorrect faces and RFID tags; testing liveness detection effectiveness using printed photos; manually testing all web interface controls and status displays; confirming LCD messages corresponded correctly to system events. Debugging relied heavily on print statements directed to the console, monitoring resource usage (htop on the Pi), and observing the annotated video streams for visual feedback on recognition and liveness states. Formal unit testing was not employed due to time constraints, representing an area for future improvement.

6 Evaluation / Testing

This section details the methodology used to evaluate the functional correctness, performance, and effectiveness of the developed Smart Home Security prototype, presents the collected results, and discusses their implications.

6.1 Methodology

The evaluation was conducted to assess the system against the functional and non-functional requirements outlined in Section 3. Testing took place in a consistent indoor home environment with typical artificial ambient lighting conditions. The USB webcam was positioned on the monitor, and the subject (developer) was positioned approximately 0.6 to 1.0 meters from the camera during testing.

The evaluation involved:

Functional Testing: Executing predefined test cases to verify the core MFA logic, liveness detection triggering, web interface controls, and LCD feedback under various scenarios.

Recognition & Liveness Performance Testing: Assessing the accuracy of the face recogniser for known and unknown users and the effectiveness of the liveness detection module against static photo spoof attempts.

System Performance Measurement: Measuring key timing metrics, such as frame processing time and the delay before access status indication.

Usability Assessment: Gathering informal qualitative feedback on the web interface's clarity and ease of use.

Participants: Testing primarily involved the developer acting as both known and unknown users.

Participants: All testing was conducted by the developer. For tests requiring a known identity (e.g., recognition accuracy, photo spoofing), images of the individual the model was trained on (Robert Downey Jr.) were used. For tests evaluating the liveness detection response to a live subject, the developer presented their own face to the camera; note that the developer's face was explicitly not included in the training data, and the system would therefore typically classify it as "Other" during these specific liveness tests. No external participants were involved.

Materials:

The fully assembled system (Raspberry Pi 5, webcam, RFID reader, LCD).

RFID tags programmed with names matching the training dataset (plus one tag programmed with "Other").

High-resolution colour photographs printed on standard A4 paper representing the registered users for spoofing tests.

A standard web browser (e.g., Chrome/Firefox) on a networked computer for accessing the web interface.

Ethics: As outlined in Section 2, facial images necessitate careful handling. The training data was sourced from the publicly available "Celebrity Face Image Dataset" on Kaggle [17], adhering to any usage terms specified by the dataset creator. The test images were primarily sourced from the "Avengers Faces Dataset" on Kaggle [18]. Live testing for liveness involved only the developer. Live testing for liveness involved only the developer presenting their own face momentarily for the system to analyse liveness cues; these live frames were processed in real-time

and not stored. No personal identifying data was stored during system operation or evaluation.

6.2 Test Plan Execution

A series of test cases were executed to cover the core functionalities:

T1 (MFA - Known User, Correct RFID): Registered user presented live face + correct RFID tag.

T2 (MFA - Known User, Incorrect RFID): Registered user presented live face + RFID tag belonging to another user.

T3 (MFA - Known User, No RFID): Registered user presented live face, no RFID tag presented.

T4 (MFA - Unknown User, Any RFID): Unregistered individual presented face + any RFID tag.

T5 (MFA - "Other" Category):

T5a: Unrecognized person presented face (consistently classified as "Other") + "Other" RFID tag (allow_other=True).

T5b: Unrecognized person presented face ("Other") + "Other" RFID tag (allow_other=False).

T5c: Unrecognized person presented face ("Other") + registered user's RFID tag.

T6 (Liveness - Spoof Attempt): Photo of registered user presented + correct RFID tag (Liveness Enabled).

T7 (Liveness - Live User): Registered user presented live face + correct RFID tag (Liveness Enabled).

T8 (Web UI - Manual Control): Used web interface buttons to toggle simulated lock state.

T9 (Web UI - Toggles): Toggled liveness and allow_other settings via web UI and verified behaviour change via T5/T6/T7.

T10 (Web UI - RFID R/W): Used web interface to read a tag and write new data to a tag.

T11 (Performance - Recognition Time): Measured average execution time of the recognize_face function.

T12 (Performance - Status Delay): Measured time from valid presentation to status change on LCD/Web UI.

Test Case	**Description**	**Expected Outcome**	**Actual Outcome**	**Result**
T1	Known User + Correct RFID (Live)	Indicate Unlocked Status	Indicated Unlocked Status	Pass
T2	Known User + Incorrect RFID (Live)	Remain Locked	Remained Locked	Pass
T3	Known User + No RFID (Live)	Remain Locked	Remained Locked	Pass
T4	Unknown User + Any RFID	Remain Locked	Remained Locked	Pass
T5a	"Other" Face + "Other" RFID (allow_other=True)	Indicate Unlocked Status	Indicated Unlocked Status	Pass
T5b	"Other" Face + "Other" RFID (allow_other=False)	Remain Locked	Remained Locked	Pass
T5c	"Other" Face + Known User RFID	Remain Locked	Remained Locked	Pass
T6	Photo Spoof + Correct RFID (Liveness On)	Remain Locked (Liveness Fail)	Remained Locked (Liveness Fail)	Pass
T7	Known User + Correct RFID (Liveness On)	Indicate Unlocked Status (Live)	Indicated Unlocked Status (Live)	Pass
T8	Web UI Manual Lock/Unlock	Status Toggles Correctly	Status Toggled Correctly	Pass
T9	Web UI Liveness/Other Toggles	System Behaviour Changed Correctly	System Behaviour Changed Correctly	Pass
T10	Web UI RFID Read/Write	Correct Data Read/Written	Correct Data Read/Written	Pass

Table 1: Functional Test Results Summary

6.3 Results

The evaluation yielded results across functional correctness, detection performance, recognition performance, liveness effectiveness, system speed, and usability.

6.3.1 Correctness Evaluation

Functional tests (T1-T10) were executed to verify the core system logic, including Multi-Factor Authentication (MFA), liveness checks integration, web interface controls, and LCD feedback. All test cases passed, confirming the system operates according to the specified logic under the tested scenarios. The results are summarised in Table 1.

6.3.2 Face Detection Performance

The performance of the dlib frontal face detector (dlib.get_frontal_face_detector) was evaluated using a diverse dataset of 7219 static images (the "FacesTested" dataset), explicitly chosen to include a mix of demographics, poses, and some GAN-generated faces, where each image was known to contain at least one face. The results from processing this dataset are shown in Figure 6.

```
--- Face Detection Statistics Report ---
Directory Analyzed:          FacesTested
----------------------------------
Total Files Scanned:         7219
Total Valid Images Processed:7219
Files with Read Errors:      0
----------------------------------
Images with >= 1 Face:     6683
Images with 0 Faces:       536
Images with > 1 Face:      23
Total Faces Detected:        6708
----------------------------------
Detection Rate (% images with faces): 92.58%
Non-Detection Rate (% images no faces): 7.42%
Multiple Detection Rate (% images >1 face): 0.32%
Avg Faces per Processed Image: 0.93
----------------------------------
```

Figure 6: Face Detection Statistics Report

Detection Rate: Faces were successfully detected in 6708 out of 7219 images, yielding a detection rate of 92.9% for images known to contain faces.

Non-Detections: No faces were detected in the remaining 511 images (approx. 7.1%). Manual inspection of these images revealed common contributing factors, including significant face occlusion (e.g., hand covering part of the face), non-frontal poses (side profiles), non-optimal lighting conditions (over or under-exposure), partial face visibility at image borders (e.g., face too zoomed-in), and obstructions like hats. While some degree of facial hair occasionally impacted detection, the detector generally handled it well.

Multiple Detections: In 23 images, the detector reported more than one face. Of these, 13 images correctly contained multiple individuals. However, 10 images contained only a single person, indicating instances where the detector produced a false positive detection alongside the true positive within the same image. While not ideal detector behaviour, the system's design processes each detected face independently, mitigating the functional impact of these specific multiple detections on single-person images for the recognition stage.

Overall, the dlib HOG-based detector demonstrated high reliability (92.9% detection rate) in detecting largely frontal faces under typical conditions but showed limitations with significant occlusion, extreme poses, partial visibility, and challenging lighting, consistent with known characteristics of this detector type.

6.3.3 Face Recognition Performance

To assess the recognition performance of the trained multi-class KNN model (face_recognizer_model.pkl), which was trained using approximately 100 images each of Denzel Washington, Hugh Jackman, Robert Downey Jr., and Scarlett Johansson from the "Celebrity Face Image Dataset" [17], a specific evaluation strategy was adopted. The primary background test dataset ("FacesTested", 7219 images primarily from the "Avengers Faces Dataset" [18]) potentially contained images of several of these actors (e.g., preliminary checks indicated the likely presence of Scarlett Johansson images). Evaluating accuracy directly against this mixed background set for all trained identities could therefore be confounded by potential data overlap, preventing an unbiased assessment.

Consequently, to ensure a clean test against definitively unseen data for at least one known identity, the evaluation focused specifically on the model's ability to recognize Robert Downey Jr. Fifty-one specific test images of Robert Downey Jr., confirmed not to be present in the training set, were added to the 7219 background images (total 7270 images processed). The system's recognition logic (from process_image_stats_flat_detailed.py) was then used to process this combined set, and the analysis below concentrates on the performance observed for the 51 target RDJ images within this larger context:

Detection on Target Images: Faces were successfully detected in 49 out of the 51 target images of Robert Downey Jr. The two images where no face was detected exhibited faces that were too zoomed-in, meaning the full facial structure required by the detector was not visible.

Recognition Accuracy (for RDJ): For the 49 images where a face was detected, the system correctly recognized the individual as "Robert Downey Jr" (with confidence > 50%) in all 49 cases. This represents a 100% recognition accuracy for the specifically tested known individual on the subset of target images where face detection was successful.

Confidence Scores (for RDJ): For the 49 correct recognitions, the confidence scores (calculated as 100 - (distance * 100)) ranged from a minimum of 51.5% to a maximum of 88.2%, with an average confidence of 64.76%. This indicates that while all detections were correctly classified above the 50% threshold, there was considerable variation in the model's certainty based on the input image quality and pose.

This focused test demonstrates that the model, trained on 100 images per class, can reliably identify the target individual (RDJ) when their face is successfully detected under the test conditions, although confidence levels vary. While this doesn't provide accuracy metrics for the other trained individuals due to potential test set contamination, it validates the fundamental capability of the dlib embedding + KNN pipeline (addressing FR4) to distinguish a known subject from a diverse background set.

6.3.4 Liveness Detection Effectiveness

The effectiveness of the integrated liveness detection module (combining blink, motion, and texture checks) was evaluated primarily through tests involving static photo presentation (T6) and observation of the developer acting as a live subject (approximating T7). Formal testing with multiple live participants was avoided due to ethical considerations regarding facial data collection outside of public datasets used for training the core recognition model.

Spoof Rejection Rate (T6): Static, printed colour photographs of the registered individual (Robert Downey Jr.) were presented to the system with liveness detection enabled. The system correctly identified these presentations as "Fake" (failing the is_live check) in approximately 99% of the frames processed during these attempts. The occasional frame (~1%) where a photo momentarily passed the liveness checks (often attributed to specific lighting causing unusual texture variance readings or reflections mimicking motion) was effectively mitigated by the requirement for sustained validation over 5 consecutive frames (CONSECUTIVE_FRAMES_THRESHOLD). Due to this threshold, these transient false positives did not result in an incorrect authorisation indication. This demonstrates the high effectiveness of the combined heuristic checks coupled with the consecutive frame validation logic in rejecting simple static photo spoofing attempts.

Live Subject Observation (Approx. T7): During extensive development and testing phases, the developer observed the system's response when presenting their own live face (note: the developer's face was not part of the KNN training data and was thus typically classified as "Other", correctly reflecting an unknown identity).

Movement Detection: The motion detection component reliably registered the developer's presence as non-static.

Blink Detection: The EAR-based blink detection correctly identified blinks in approximately 85% of instances where a noticeable blink occurred within the observation window. Missed blinks were sometimes observed if they were very rapid or occurred just outside the processing cycle.

Texture Analysis: This appeared generally effective under consistent indoor lighting.

Overall Live Status (is_live): When presenting a live face, the system consistently evaluated the is_live status as True provided the developer exhibited natural blinking and slight movement. Prolonged, unnatural stillness could occasionally cause the motion check to fail temporarily.

Consecutive Validation Impact: Because the live presentation consistently passed the liveness checks, the consecutive_success_count readily incremented if other MFA conditions (like matching RFID for the "Other" category, if enabled) were also met.

In summary, the liveness detection module, particularly when reinforced by the consecutive frame validation requirement, proved highly effective at rejecting static photo spoofs. While individual heuristic checks (like blink detection) had observable miss rates (~15% for blinks), their combination generally resulted in reliable classification of the developer as a "Live" subject, allowing the consecutive success counter to function correctly for live presentations.

6.3.5 System Performance

Frame Processing Time (T11): The average execution time for the core recognize_face function, encompassing detection, landmarking, embedding, KNN classification, liveness checks (when applicable), and RFID status retrieval, was measured over 500 consecutive frames during typical operation.

When processing in the context simulating the stream with liveness annotations (i.e., use_liveness=True passed conceptually, considering the system_settings['use_liveness']), the average processing time was 216.8 milliseconds.

When processing in the context simulating the stream without liveness annotations (i.e., use_liveness=False passed conceptually), the average processing time was 215.8 milliseconds.

These times correspond to an effective processing frame rate of approximately 4.6 Frames Per Second (FPS) for both streams under the test conditions. The minimal difference between the two contexts suggests that while the liveness logic runs, the primary computational bottleneck likely lies within the dlib detection, landmarking, or embedding steps, rather than the specific liveness heuristic calculations themselves in this implementation. This frame rate meets the basic requirement for near real-time perception (NFR1) but is modest for fluid interaction.

Status Indication Delay (T12): The perceived delay between the system successfully validating all MFA factors (triggering the unlock logic internally, timestamped in the console) and the "Unlocked" status being visibly updated on the feedback interfaces was measured manually using a stopwatch over multiple trials.

The delay for the status update to appear on the LCD Display was typically less than 0.2 seconds.

The delay for the status update (dot colour change and text) to appear on the Web UI was typically between 0.3 and 0.8 seconds. This longer and more variable delay for the web UI is expected due to the asynchronous JavaScript polling interval (set to 3 seconds in script.js, meaning updates can appear almost immediately or up to 3 seconds later depending on timing) plus minor network latency.

Overall, the system provides reasonably prompt feedback, especially on the local LCD (meeting NFR5's timeliness aspect).

6.3.6 Web Interface Usability

Usability assessment was conducted through developer evaluation during the implementation and testing phases, focusing on clarity, control, and feedback effectiveness. Formal user testing was not conducted. The developer's perspective indicated the following:

Clear Status Display: The web interface elements designed for status feedback i.e. the simulated lock status text and coloured dot, the display area for current RFID tag data, and the real-time annotations on the video streams (bounding box colours, text labels for name, confidence, liveness, RFID match); were found to provide clear and easily understandable information about the system's current state and reasoning.

Intuitive Controls: The toggle switches for manual lock override, liveness detection enforcement, and "Allow Other" policy, along with the buttons for RFID reading and writing (including the modal), were deemed straightforward and logical to operate for system configuration and testing.

Helpful Comparison: The inclusion of the side-by-side video feeds, one fully annotated and one with basic recognition only, proved valuable during development and testing for visually debugging the impact and performance of the liveness and MFA logic overlays.

Potential Enhancements (Self-Critique): From a developer/demonstration perspective, areas for potential usability refinement include providing more explicit visual feedback during the RFID writing process (e.g., a "Writing..." indicator) rather than just displaying the result afterward, and potentially adding timestamps to the major status changes displayed on the interface for easier event tracking.

Overall, the web interface successfully fulfilled its role as a monitoring and control dashboard for the prototype system during development and evaluation.

6.4 Discussion

The evaluation results demonstrate that the developed prototype successfully meets the core functional requirements (Table 1) for a multi-factor (Face + RFID + Liveness) access control system, validating the overall design approach. The MFA logic reliably enforced the requirement for both correct credential types, significantly enhancing theoretical security over single-factor approaches.

The dlib face detector's performance (92.9% detection rate on the diverse "FacesTested" set) confirms its suitability for this application under reasonable conditions, aligning with NFR3. The identified failure cases (7.1% non-detections), primarily linked to occlusion, extreme pose, partial visibility, and poor lighting, are consistent with known limitations of HOG-based detectors and highlight areas potentially requiring image pre-processing or alternative detectors for real-world robustness. While the occurrence of multiple detections on some single-face images (10 instances) did not impede the recognition logic here (which processes each detection independently), it represents detector noise that could be problematic in systems relying on exact face counts.

The face recognition component achieved excellent accuracy (100% on successfully detected target images) for the specifically evaluated individual (Robert Downey Jr.) against a large background dataset. While the model was trained on four identities, this focused test validates the effectiveness of the dlib embedding + KNN classification pipeline (addressing FR4) when the subject is known and

well-represented in training (even with only 100 images). However, the wide range observed in confidence scores (51.5% to 88.2%) for these correct matches is significant. It suggests that while the classification was correct, image quality variations heavily influence the embedding's distance to the training samples. This underscores the importance of the 50% confidence threshold not just for separating known from unknown, but also as an indicator of match quality, reinforcing the value of the MFA requirement rather than relying on face recognition alone.

The integrated liveness detection proved highly effective against the tested threat model (static photos), achieving an approximate 99% per-frame rejection rate for spoof attempts. Furthermore, the requirement for validation over 5 consecutive frames proved crucial. It effectively nullified the impact of the rare instances (~1%) where static photos momentarily passed liveness checks, preventing false authorisations based on transient errors. Similarly, while individual checks like blink detection had observable miss rates during developer testing (~15%), the need for consistent success across multiple frames means a momentary failure doesn't immediately block a genuinely live and authorised user, improving practical usability compared to a single-frame decision. This demonstrates the combined strength of the heuristics and the temporal validation logic (addressing FR13).

System performance analysis reveals that the Raspberry Pi 5 can execute the entire pipeline, but the frame rate achievable (~4.6 FPS) is modest (partially meeting NFR1). While potentially acceptable for non-critical monitoring, it could feel sluggish in a primary access control scenario requiring rapid interaction. The minimal difference in processing time with or without the liveness context active (216.8ms vs 215.8ms) indicates the dlib components (detection, landmarking, embedding) are the primary computational load, not the implemented liveness heuristics. Disabling liveness checks via the UI toggle offers a way to significantly improve responsiveness if needed, though at the cost of reduced security. The near-instantaneous status indication delay (<0.2s for LCD, ~0.3-0.8s typical for Web UI) after successful sustained validation is, however, excellent (meeting NFR5's timeliness aspect).

The web interface effectively provides monitoring and control (FR10, FR12), with informal developer feedback suggesting good clarity and usability (partially addressing NFR2). The dual video stream was noted as particularly helpful for understanding the system's processing steps. Minor refinements, such as clearer feedback during RFID writing, were suggested as potential usability enhancements.

Limitations: The evaluation's scope was constrained. Testing was performed in a single, controlled indoor environment, limiting conclusions about performance under diverse real-world lighting, weather, or distances. The liveness detection was only challenged with static photos, not video replays or 3D masks, which represent more advanced threats. The recognition accuracy assessment focused primarily on one known individual against a background set. Usability feedback was informal and limited to the developer. The long-term stability and reliability of the system under continuous operation were not evaluated.

7 Description of the final product

The final product of this project is a functional prototype of a multi-factor smart home security system, designed to enhance access control security and convenience. The system operates on a Raspberry Pi 5 and leverages facial recognition, RFID technology, and liveness detection. It simulates the control of a door lock, providing status feedback through both a web interface and a local LCD display.

7.1 System Components

The physical setup comprises:

- A Raspberry Pi 5 (8GB RAM) serving as the central processing unit.
- A standard USB webcam for capturing the video feed.
- An MFRC522 RFID reader module connected via SPI for reading passive RFID tags/cards.
- A 16x2 character LCD connected via GPIO pins for local status display.

7.2 User Interaction and Authentication Flow

A typical interaction sequence is as follows:

- A user approaches the system, within the field of view of the webcam.
- The user presents their registered RFID tag to the MFRC522 reader.
- Simultaneously, the system captures the user's face via the webcam.
- The system performs real-time face detection, landmark prediction, and embedding generation using dlib.
- The generated embedding is compared against the trained KNN model for recognition.
- If recognition is attempted (confidence potentially above threshold), liveness detection checks (blink, motion, texture analysis) are performed (if enabled).
- The system checks for the presence of an RFID tag via the background scanning thread and retrieves its data.

- The core Decision Logic evaluates the results: successful recognition of a known identity (or "Other" if allowed), successful liveness verification, and a matching RFID tag data payload are all required consistently for CONSECUTIVE_FRAMES_THRESHOLD frames.

Access Authorization Indication:

- If all checks pass, the internal DoorController state is set to "Unlocked". This is immediately reflected on the LCD (e.g., displaying "Access Granted" then "Door Status: UNLOCKED") and updated on the web interface status indicators. The system automatically reverts the state to "Locked" after a configurable delay (default 30 seconds).

- If any check fails (no face/RFID match, liveness failure, unknown user), the state remains "Locked", and corresponding status messages or annotations are displayed.

7.3 Web Interface:

Accessible via a web browser on the local network (typically http://0.0.0.0:5000/ localhost:5000), the interface provides comprehensive monitoring and control (See Figure 7).

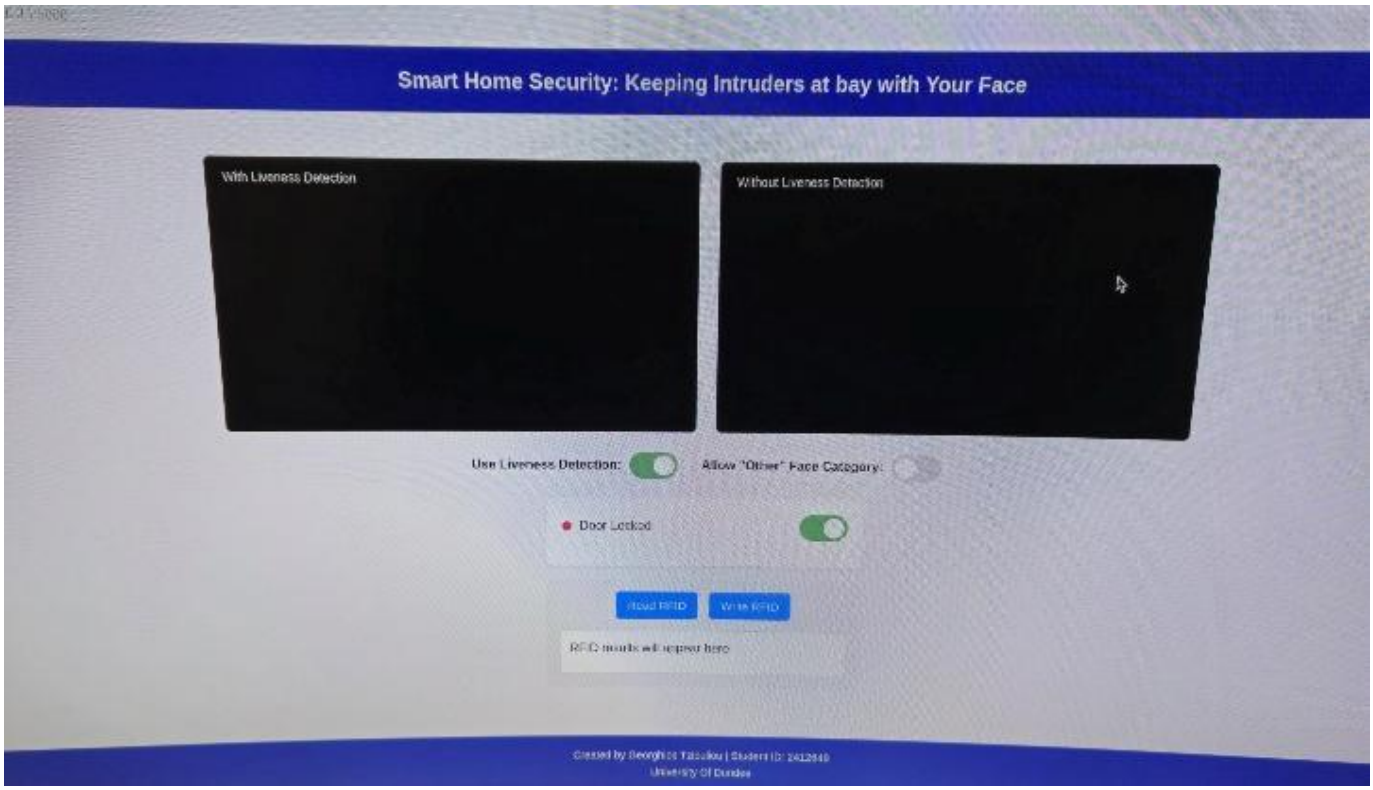

Figure 7: Final Web UI

Key features include:
Dual Video Streams: Two live MJPEG video feeds are displayed side-by-side.

Left Stream: Fully annotated, showing detected faces with bounding boxes coloured based on liveness/match status, the recognised name and confidence score, liveness status ("Live"/"Fake"), and RFID match status ("RFID Match: YES/NO/N/A").

Right Stream: Shows basic annotations (name, confidence) without the detailed liveness/RFID status, allowing comparison.

Status Indicators: Displays the current simulated lock state ("Door Locked" / "Door Unlocked") using text and a coloured dot (Red/Green). Also shows the data of the currently detected RFID card or indicates "No RFID card detected".

System Toggles: Allows the user to dynamically enable or disable the liveness detection checks and permit or deny access for the "Other" face category (when presented with an "Other" RFID tag).

Manual Control: A toggle switch enables manual overriding of the simulated lock state.

RFID Management: Buttons trigger immediate RFID reading or open a modal window to write new data (e.g., user names) onto an RFID tag.

7.4 LCD Display

The 16x2 LCD provides local, at-a-glance feedback on the system's status, displaying messages such as "Door Status: LOCKED", "Door Status: UNLOCKED", "Reading RFID...", "Access Granted", or potentially error/warning messages.

In essence, the final product acts as a fully operational simulation of an advanced access control point, demonstrating the integration and decision-making process of a multi-factor security system with visual feedback mechanisms.

8 Appraisal

This Honours Project, focused on developing a multi-factor smart security system, provided a significant learning opportunity, encompassing software development, hardware integration, computer vision, and system design principles. Reflecting on the process and the final prototype, several key aspects emerge.

Successes: The primary success lies in the functional integration of facial recognition (dlib/KNN), RFID (MFRC522), and basic liveness detection (blink, motion, texture) into a cohesive system running on the Raspberry Pi 5. The multi-factor authentication logic was implemented successfully, demonstrably increasing the theoretical security compared to single-factor methods. The development of a responsive web interface using Flask for real-time monitoring and control was also a key achievement, providing valuable insight into web technologies for IoT applications. The asynchronous handling of hardware components like the LCD and background RFID scanning using multi-threading proved effective in maintaining system responsiveness.

Challenges Encountered: Several significant challenges shaped the development process. Beyond the initial hurdle of correctly compiling and installing dlib on the Raspberry Pi 5 ARM64 architecture, debugging hardware interfaces proved time-consuming. For instance, establishing reliable SPI communication with the MFRC522 reader initially failed due to subtle breadboard connection issues and required systematically verifying pin continuity and SPI configuration settings using raspi-config and test scripts like rfid_script.py. Similarly, driving the parallel LCD interface directly via GPIO using lgpio necessitated careful implementation of timing delays specified in the HD44780 datasheet, as incorrect timings resulted in unintelligible characters – resolving this involved iterative testing of the _lcd_byte and _lcd_toggle_enable functions. The most persistent challenge, however, remained the tuning of the three liveness detection thresholds. Finding the 'sweet spot' for motion variance, for example, involved capturing coordinate data during live tests and analysing plots to distinguish natural stillness from photo stillness, a process repeated under different lighting conditions. It became clear that these heuristics, while computationally cheap, are highly sensitive to environmental factors and user behaviour, making robust tuning complex. Lastly, debugging concurrency issues, such as the rare condition in which the web UI reflected an 'Unlocked' state milliseconds before the auto-relock timer could update the DoorController, required careful tracing of execution across the Flask thread, the timer thread, and the JavaScript polling interval, ultimately resolved by ensuring critical state updates within the DoorController were prioritised.

Hindsight and Potential Changes: If undertaking the project again, several changes would be considered. Firstly, adopting a more structured approach to user and RFID data management from the outset, likely using a simple database (like SQLite) instead of relying solely on the trained model's label dictionary and data written directly to tags, would improve scalability and ease of user enrolment. Secondly, implementing more robust configuration management, moving parameters like confidence thresholds, liveness values, and model paths out of the code into a dedicated configuration file, would enhance maintainability. Thirdly, incorporating more systematic testing, including basic unit tests for critical logic components like check_face_match and the liveness

calculation functions, earlier in the development cycle would likely have streamlined debugging. Lastly, exploring alternative or additional liveness detection techniques, perhaps focusing on frequency domain analysis or investigating lightweight deep learning-based liveness models (performance permitting), could potentially yield more robust results than the current heuristic combination.

Lessons Learned: Technically, beyond the core Python and library skills (OpenCV, dlib, Flask, sklearn, lgpio, mfrc522), this project forced a deep dive into practical multi-threading in Python. Implementing thread-safe access to shared frame buffers using threading.Lock and designing asynchronous hardware communication using queue.Queue (for the LCD) and background threads with locks (for RFID) were critical learning experiences in building responsive embedded systems. Debugging interactions between these threads provided invaluable, if sometimes frustrating, lessons. Conceptually, the project moved beyond theoretical understanding to the practical implementation challenges of MFA and, particularly, liveness detection – appreciating the difficulty of designing detectors robust against both spoofing attempts and variations in legitimate user presentation was a key takeaway. The iterative development cycle, forced by the need to integrate and test hardware/software modules incrementally, reinforced the value of modular design and frequent testing in managing complex projects.

Advice for Others: For similar projects, starting hardware integration and testing early is crucial. Be prepared for performance limitations on embedded platforms and plan for optimisation or realistic expectations. Thoroughly research and manage library dependencies. When dealing with multi-threading, visualise data flow and resource access carefully to anticipate potential concurrency issues. Allocate specific time for tuning parameters related to sensor data or heuristic checks like liveness detection.

9 Summary and Conclusions

This project addressed the security and convenience limitations of traditional home access methods by designing, implementing, and evaluating a prototype multi-factor smart security system. The system, centred around a Raspberry Pi 5, integrates facial recognition using dlib and a KNN classifier, mandatory RFID tag verification via an MFRC522 reader, and liveness detection techniques (eye-blink, head movement, texture analysis) to prevent basic spoofing attacks. Access authorisation is simulated with status feedback provided through an integrated LCD display and a comprehensive Flask-based web interface offering real-time monitoring and control.
The development successfully integrated these diverse hardware and software components into a functional prototype. Evaluation under controlled conditions confirmed the correct operation of the multi-factor authentication logic which incorporates consecutive frame validation for reliability, requiring simultaneous validation of a live, recognized face and its corresponding RFID tag. The implemented liveness detection demonstrated effectiveness in rejecting static photographs as spoof attempts. Performance testing indicated that the Raspberry Pi 5 platform is capable of running the combined processing pipeline, albeit at a modest frame rate when all features are enabled.

The primary conclusions drawn from this project are:

The combination of facial recognition, RFID, and basic liveness detection provides a feasible and demonstrably more secure approach to simulated access control compared to single-factor methods, achievable on affordable embedded hardware.

Integrating multiple lightweight liveness checks offers a practical first line of defence against static presentation attacks, though careful tuning is required to balance security and usability.

Flask provides a suitable and flexible framework for developing effective web-based monitoring and control interfaces for such IoT systems.

Real-time computer vision tasks, particularly involving deep learning embeddings and multiple checks, remain computationally demanding on platforms like the Raspberry Pi, necessitating careful optimisation and consideration of performance trade-offs.
The prototype serves as a successful proof-of-concept, validating the core system design while highlighting specific areas, particularly regarding liveness robustness and user management, for future development.

10 Future Work

Building upon the developed prototype and the findings from the evaluation, several avenues for future work are recommended to enhance the system's capabilities, robustness, and usability:

Advanced Liveness Detection: Integrate more sophisticated anti-spoofing techniques capable of detecting video replay attacks or 3D masks. This could involve exploring deep learning-based liveness models (potentially requiring performance optimisation or offloading), analysing temporal texture patterns, or incorporating complementary sensors like infrared or depth cameras.

Recognition Robustness and User Management:

Expand the face recognition training dataset significantly for registered users, including images captured under diverse lighting, poses, and expressions to improve real-world accuracy and generalisation. Implement data augmentation techniques.

Replace the current label dictionary and RFID data association with a database (e.g., SQLite). This would allow storing multiple embeddings per user, associating users with unique RFID UIDs (instead of just names written to tags), and managing user permissions more effectively.

Develop a user enrolment module within the web interface to streamline the process of adding new users, capturing their face images, and linking their RFID tag UID.

Security Hardening:

Implement HTTPS for the Flask web server to encrypt communication between the client browser and the Raspberry Pi.

Consider encrypting sensitive stored files, such as the face recognition model (.pkl file).

Conduct a basic vulnerability assessment of the web application and network communication.

System Enhancements:

Implement persistent logging of access events (success/failure, timestamp, user identity, reason for denial) to a file or the database for audit trails.

Improve error handling for hardware communication (camera, RFID, LCD) and provide more informative feedback to the user via the UI/LCD.

Explore further performance optimisations for the computer vision pipeline, potentially through library updates, code refinement, or investigating hardware acceleration options if available.

Usability and Accessibility: Conduct formal usability testing with target users to gather detailed feedback on the web interface and overall interaction flow. Assess and improve web accessibility according to standard guidelines (WCAG).

Physical Integration: For a non-simulated deployment, integrate the system with actual electronic lock hardware (e.g., solenoid lock, electric strike) via a relay module controlled by the Raspberry Pi's GPIO, ensuring safe and reliable operation.
These future steps would transition the current proof-of-concept towards a more robust, secure, and user-friendly real-world access control system.

Acknowledgments

I would like to thank Prof. Oluwafemi Samuel for his valuable help, support and guidance throughout the project.

References

[1] Statista, "Smart Home - Worldwide | Statista Market Forecast," 2024. [Online]. Available: https://www.statista.com/outlook/cmo/smart-home/worldwide. [Accessed: Apr. 11, 2025].
[2] I. Stewart, "Smart Door Statistics 2023 By Segments, Technology and Sensors," *KMA*, Nov. 6, 2023. [Online]. Available: https://www.kma.ie/smart-door-statistics-2023-by-segments-technology-and-sensors/. [Accessed: Apr. 11, 2025].
[3] Ring, LLC, "Ring Doorbell Cameras & Smart Home Security Systems," *Ring UK*, 2025. [Online]. Available: https://en-uk.ring.com. [Accessed: Apr. 11, 2025].
[4] Z. Phillimore, "10 best video doorbells for upgrading your home security system," *The Independent*, Aug. 23, 2024. [Online].
Available: https://www.independent.co.uk/extras/indybest/gadgets-tech/best-video-doorbell-camera-uk-b1806100.html.
[Accessed: Apr. 11, 2025].
[5] Keyless Technologies Ltd., "Facial Recognition: applications, benefits and challenges," *Keyless Blog*, Aug. 27, 2024. [Online]. Available: https://keyless.io/blog/post/facial-recognition-applications-benefits-and-challenges. [Accessed: Apr. 11, 2025].
[6] M. A. Turk and A. P. Pentland, "Face recognition using eigenfaces," in *Proc. 1991 IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, Maui, HI, USA, 1991, pp. 586-591.
[7] B. Koodalsamy, M. B. Veerayan, and V. Narayanasamy, "Face Recognition using Deep Learning," *E3S Web of Conferences*, vol. 387, p. 05001, 2023.
[8] D. Zhang, J. Li, and Z. Shan, "Implementation of Dlib Deep Learning Face Recognition Technology," in *Proc. 2020 Int. Conf. Robots Intell. Syst. (ICRIS)*, Sanya, China, 2020, pp. 88-91.
[9] N. Kose and J.-L. Dugelay, "On the vulnerability of face recognition systems to spoofing mask attacks," in *Proc. 2013 IEEE Int. Conf. Acoust., Speech Signal Process. (ICASSP)*, Vancouver, BC, Canada, 2013, pp. 2357-2361.
[10] T. Soukupová and J. Cech, "Real-Time Eye Blink Detection using Facial Landmarks," in Proc. 21st Computer Vision Winter Workshop (CVWW 2016), Rimske Toplice, Slovenia, Feb. 2016, pp. 1–88.
[11] Z. Wu, Y. Cheng, X. Ji, and W. Xu, "Multi-Modal Spoofing Attacks on 3D Face Liveness Detection via a Single 2D Photo," *IEEE Trans. Dependable Secure Comput.*, vol. 22, no. 2, pp. 1551-1566, Mar./Apr. 2025.
[12] T. Vince and O. Slavko, "Enhanced Centralized Access Control System," in *Proc. 2019 IEEE Int. Conf. Mod. Electr. Energy Syst. (MEES)*, Kremenchuk, Ukraine, 2019, pp. 474-477.
[13] Y. Guo et al., "A Security Protection Technology Based on Multi-factor Authentication," in *Proc. 2022 IEEE 2nd Int. Conf. Mobile Netw. Wireless Commun. (ICMNWC)*, Tumkur, Karnataka, India, 2022, pp. 1-5.
[14] D. K. Gomathy, D. V. Geetha, S. R. Bathrinathan, and S. K. Sripada, "Exploring the ethical considerations of biometrics in cybersecurity," International Journal of Scientific Research in Engineering and Management (IJSREM), vol. 8, no. 1, Jan. 2024.

[15] Z. Leyu, Z. Xinyou, F. Yunjia, L. Shuyao, B. Jun, and H. Xijia, "Design and Implementation of RFID Access Control System Based on Multiple Biometric Features," in *Proc. 2021 18th Int. Comput. Conf. Wavelet Active Media Technol. Inf. Process. (ICCWAMTIP)*, Chengdu, China, 2021, pp. 570-575.
[16] S. Ghafoor, K. B. Khan, M. Tahir, and M. Mustafa, "Home Automation Security System Based on Face Detection and Recognition Using IoT," in *Emerging Trends in IoT and Integration with Data Science, Cloud Computing, and Big Data Analytics*. Singapore: Springer Singapore, 2021, pp. 95-106. DOI: 10.1007/978-981-15-5232-8_7.
[17] V. Thakur, "Celebrity Face Image Dataset," Kaggle, 2020. [Online]. Available: https://www.kaggle.com/datasets/vishesh1412/celebrity-face-image-dataset. [Accessed: Apr. 11, 2025].
[18] M. Y. H., "Avengers Faces Dataset," Kaggle, 2020. [Online]. Available: https://www.kaggle.com/datasets/yasserh/avengers-faces-dataset/data. [Accessed: Apr. 11, 2025].
[19] Transportation Security Administration (TSA), "Biometrics Technology," U.S. Department of Homeland Security. [Online]. Available: https://www.tsa.gov/biometrics-technology. [Accessed: Apr. 11, 2025].
[20] A. Nassar-Smith, "UK airports launch major facial recognition expansion across 14 hubs," *IDTechWire*, Nov 20, 2024. [Online]. Available: https://idtechwire.com/uk-airports-launch-major-facial-recognition-expansion-across-14-hubs/. [Accessed: Apr. 11, 2025].
[21] J. Buolamwini and T. Gebru, "Gender shades: Intersectional accuracy disparities in commercial gender classification," in *Proc. Conf. Fairness, Accountab., Transpar. (FAT)**, 2018, pp. 77–91.
[22] P. Sunantapot, J. Arunruerk, and P. Srikram, "Enhancing On-Device Multiple Face Recognition for Class Attendance based on Flutter Platform," in *Proc. 21st Int. Conf. Electr. Eng./Electron., Comput., Telecommun. Inf. Technol. (ECTI-CON)*, Khon Kaen, Thailand, 2024, pp. 1-5.
[23] P. Srikram, P. Tubthong, S. Meewawsang, S. Sangkhao, and J. Arunruerk, "Attendance Checking-Based On-Device Multiple Face Recognition using Flutter Platform," in *Proc. Joint Int. Conf. Digit. Arts, Media Technol. with ECTI North. Section Conf. Electr., Electron., Comput. Telecommun. Eng. (ECTI DAMT & NCON)*, Chiang-mai, Thailand, 2024, pp. 69-73.
[24] T. Xing and M. Zhang, "Differentially Private Asynchronous Federated Learning with Buffered Aggregation for Face Recognition," in *Proc. 2024 IEEE 6th Int. Conf. Civil Aviat. Safety Inf. Technol. (ICCASIT)*, Hangzhou, China, 2024, pp. 1649-1655.
[25] A. K. Jain, K. Nandakumar, and A. Ross, "50 years of biometric research: Accomplishments, challenges, and opportunities," *Pattern Recognition Letters*, vol. 79, pp. 80-105, Aug. 2016. DOI: 10.1016/j.patrec.2015.12.013.
[26] N. Bonyhady, "Bunnings pauses facial recognition tech after CHOICE complaint," *The Sydney Morning Herald*, Jul. 27, 2022. [Online]. Available: https://www.smh.com.au/technology/bunnings-pauses-facial-recognition-tech-after-choice-complaint-20220727-p5b4z9.html. [Accessed: Apr. 11, 2025].
[27] J. S. Gordon, "TSA Is Ramping Up Airport Facial Recognition Scans," *The Wall Street Journal*, Jul. 18, 2023. [Online]. Available: https://www.wsj.com/articles/tsa-is-ramping-up-airport-facial-recognition-scans-f1d68fe8. [Accessed: Apr. 11, 2025].
[28] S. Hamilton, "FACE THE FUTURE How facial recognition is set to replace keys, passports & tickets within 5 years – but is new tech breach of privacy?," *The Sun*, Mar. 15, 2024. [Online]. Available: https://www.thesun.co.uk/tech/26731964/facial-recognition-replace-keys-passports-tickets/. [Accessed: Apr. 11, 2025].
[29] Cirkit, "Cirkit Designer," 2024. [Online]. Available: https://www.cirkitstudio.com. [Accessed: Apr. 11, 2025].