Facial Recognition-Based Cabin Access Verification System

Internship Project at SoulVibe.Tech

Bhuvaneswari Gummalavarapu
B.Tech Computer Science Engineering
Intern, SoulVibe.Tech
bhuvanagummalass123@gmail.com

Contents

1 Introduction

In today's tech-driven world, having secure and convenient access control systems isn't just a luxury—it's a necessity. Traditional approaches like keycards, PINs, or plain old locks often fall short. Either they're too easy to bypass, or they become cumbersome in fast-paced or high-traffic settings. That's where biometrics—especially facial recognition—start to shine. It's fast, non-invasive, and frankly, feels like the future.

This project focuses on building a real-time facial recognition system specifically designed to control access to secure cabin spaces. The goal? Make sure only authorized individuals can get in—without relying on the cloud or external networks. Unlike many modern systems that send sensitive data over the internet (and sometimes compromise privacy), this one runs completely offline. All user data is stored locally, which means it's both faster and safer.

The system is built in Python, using powerful libraries like `face_recognition` and OpenCV to handle the heavy lifting—detecting faces, encoding features, and identifying users in real time. The architecture is modular, so the core recognition and logging functions are separated out for clarity and easier updates. There's even a feature to register new users right from the interface—handy for on-site management.

Each time someone tries to access the cabin, the system checks their face against the local database and logs the result along with a timestamp. This not only helps with security audits but can double up as an attendance tracker if needed.

This whole project came together during my internship at SoulVibe.Tech. It was a chance to bring textbook knowledge into the real world—bridging the gap between theory and actual deployment. Along the way, I got to dive deep into computer vision, work on software design and system integration, and think critically about user experience in a real, constrained environment.

2 System Architecture and Components

The application adopts a modular design approach. Python, due to its extensive library support and readability, is the primary programming language. Key libraries are used to implement face detection, recognition, image processing, and data management.

2.1 Core Components

- **face_recognition:** Provides simple yet powerful APIs for face detection and encoding comparison using deep learning models.
- **OpenCV (cv2):** Handles webcam input, frame capture, image conversion, and user interface rendering (boxes, text).
- **NumPy:** Facilitates efficient numerical operations, especially when working with face encodings as vectors.
- **Pandas:** Supports reading and writing user and log data in structured tabular formats.
- **CSV, OS:** Used for file system traversal, directory creation, and manipulation of CSV files.
- **Datetime, Time:** Responsible for timestamping events, measuring processing time, and managing delays in the UI.

2.2 Folder Structure

```
facial_access_system/
    train/                  # Stores authorized user face images
    access_log.csv          # Logs all access attempts
    authorized_users.csv    # Registered users (Name, Encoding, Timestamp)
    add_user.py             # Script to register new users
    access_control.py       # Real-time access control script
    report.pdf              # This report
    presentation.pptx       # Project slides
    README.md               # Project instructions
```

Each component is modular, making the system maintainable and extensible.

3 Key Features and Functionality

3.1 Face Detection & Recognition

- Real-time video frames are captured and resized to optimize performance.
- Frames are converted to RGB, and faces are detected using HOG + CNN models.
- Each face is encoded as a 128-dimensional vector using the `face recognition` library.
- Detected faces are compared against known encodings to check for matches.

3.2 Access Validation

- The `authorized users.csv` file stores names and corresponding face encodings.
- A matching face results in "Access Granted" with a green box overlay; otherwise, "Access Denied" with a red box.
- The match threshold is adjustable to balance false accept and reject rates.

3.3 Add New User Functionality

- Pressing 'A' during execution pauses recognition and starts the user registration routine.
- User is prompted for a name; a face snapshot is taken, encoded, and added to the database.
- System validates that only one face is present in the frame. It shows warnings if multiple or no faces are detected.
- Images are saved in the `train/` folder for audit purposes.

3.4 Access Logging

- Each access attempt—granted or denied—is logged into `access_log.csv`.
- Fields include: Name (or "Unknown"), Timestamp, and Access Status.
- This file can be used for auditing and generating attendance or security reports.

3.5 User Database Management

- At launch, the system reads the CSV and loads all user encodings into memory.
- When a new user is registered, both the CSV and in-memory list are updated in real time.
- Allows dynamic, live database updates without restarting the application.

4 System Flow (Conceptual)

4.1 Access Control Flow

1. System initializes and loads existing user encodings.
2. Webcam feed starts, capturing frames in real-time.
3. Frames are resized and converted to RGB format.
4. System detects faces and computes encodings.
5. Detected encodings are compared against stored encodings.
6. Access decision is made; bounding boxes and status labels are drawn.
7. Attempt is logged in CSV file.
8. System listens for:
    - ‘Q‘: Quit the application
    - ‘A‘: Start the user registration module
9. The process loops for continuous access control.

4.2 User Registration Flow

1. Admin presses ‘A‘ to register a new user.
2. Current webcam session is paused, and the user is prompted for their name.
3. A single face image is captured.
4. If the face detection finds exactly one face:
    - Encoding is created.
    - Name, encoding, and timestamp are saved to `authorized_users.csv`.
    - Image is saved to the `train/` directory.
5. Control is returned to the main access loop.

5 Challenges and Learnings

5.1 Challenges

- **Real-time Speed vs Accuracy:** Higher-resolution frames improve recognition accuracy but reduce speed. This was mitigated by resizing frames and limiting recognition frequency.
- **Encoding Storage in CSV:** Directly saving NumPy arrays in CSV is not straightforward. We used string conversion and evaluated them back to arrays during loading.
- **Webcam Conflicts:** Handling camera initialization and release during transitions between modules required careful resource management.
- **Multiple Faces:** Preventing registration errors when multiple faces were present required robust detection logic and user warnings.

5.2 Learnings

- Practical integration of facial recognition models with real-time video processing.
- Use of persistent data structures for maintaining biometric user databases.
- Importance of error handling and user feedback in live systems.
- Hands-on experience with modular design and scalable architecture for computer vision systems.

6 Conclusion

The Facial Recognition-Based Cabin Access System represents a meaningful leap toward smarter, more secure access control in the workplace. By combining cutting-edge computer vision tools with an entirely offline setup, the project offers a robust and scalable solution for modern access challenges. While it was originally designed with cabin security in mind, the system's modular design, real-time performance, and built-in support for user registration and activity logging make it versatile enough for broader environments—like smart offices, research labs, or restricted zones.

From the beginning, this wasn't just about getting the code to work—it was about building a complete, user-friendly experience. We paid close attention to how users interact with the system, how data is stored and retrieved, and how the application holds up in practical use cases. It was really satisfying to see how Python, along with libraries like OpenCV and `face_recognition`, could be stitched together into something that feels like a real-world product.

Working on this system gave me hands-on exposure to more than just coding. It involved a mix of system architecture, real-time video handling, and managing biometric data securely. The webcam integration, CSV-based logging, and the way users get feedback from the interface all pushed me to think across both software and hardware boundaries—which was both challenging and fun.

Looking ahead, there are definitely some areas I'd love to explore further. Adding multi-factor authentication (like facial recognition paired with RFID) could boost security. Liveness detection would help protect against spoofing attacks (like printed photos or videos). A feature to notify administrators about failed access attempts, maybe through mobile alerts, could be useful too. Also, a graphical dashboard would go a long way in making the system more intuitive for non-technical users.

All in all, this project has been a solid dive into building secure, real-time biometric systems. It's not just a good learning exercise in machine learning and computer vision, but also a lesson in how to balance performance, usability, and security in software engineering.