Facial Recognition in Embedded Systems:
Design and Development of a Vision-Activated Control System

Natalia ANFILOVA, Marius Daniel MARCU, Florin Gabriel POPESCU, Razvan SLUSARIUC

Faculty of Mechanical and Electrical Engineering, University of Petrosani, Romania

What Can This Vision-Controlled System Do?

This is a **Facial-Input-Controlled System** — or **FICS** for short. It lets you **control real hardware** using only **your face and voice**. No hands needed.

You look, blink, smile, or speak — and the system responds real-time.

It's powered by **computer vision**, **voice recognition**, and a smart embedded board called **ESP8266**. The camera watches your face. A Python-based **Facial Recognition Module** detects your gestures. Another Python script — the **Signal Relay Module** — sends those signals to the ESP8266 microcontroller.

The ESP8266 runs **C++ code** written in the **Arduino IDE**. It connects to the same Wi-Fi as your computer. It receives commands, then turns on **LEDs**, **relays**, or an **electric motor**. The **LCD screen** shows what's happening in real time.

You can:

•Turn on lights with a gaze.

•Start a motor with a smile and blink combo.

•Pause and resume everything with simple voice commands.

Everything works over **HTTP requests** on your local network — fast, light, and direct.

FICSs like this one are not just cool — they are **useful**. They can be adapted for **home automation**, **robotics**, or **assistive technologies** for people with limited mobility.

This model is just the beginning.

Why FICSs Matter: Real-Life and Industry Use

Facial Interaction Control Systems (FICSs) represent a new generation of human-machine interfaces. They are fast, intuitive, and completely hands-free.

FICSs interpret micro-expressions, eye movements, and facial gestures to control machines — in real time. They rely on computer vision, convolutional neural networks (CNNs), and human-computer interaction (HCI) design. Our device, for example, uses Python-based gesture recognition to trigger commands over Wi-Fi using facial input alone.

- In **biometric security**, FICSs authenticate users through facial depth maps and landmark analysis. Used in Apple's Face ID or NEC's surveillance systems, these technologies detect spoofing attempts and adapt to changing light or expression.
- In **healthcare**, companies like Philips analyze facial cues to detect pain, fatigue, or confusion. These systems support passive monitoring in ICUs and elderly care.
- In **retail and marketing**, smart kiosks read emotional states to tailor advertising. Cloud-based systems like Amazon Rekognition process millions of faces to optimize engagement.
- In **assistive tech**, FICSs provide new communication channels for people with mobility challenges. Just like our FICS prototype, they allow users to toggle devices using only a smile or blink — a vital feature for independent living.
- In **automotive tech**, gaze tracking and facial fatigue detection now improve road safety. Driver-monitoring systems detect inattention and alert the user in real time.

FICSs are growing fast — from lab demos to commercial products. They are embedded in smart glasses, wearables, home automation, and even robotics. As image processing and deep learning models evolve, FICSs will become more adaptive, more ethical, and even more human.

This isn't the future — it's already here.
And your vision-driven controller is a small but powerful step in it.

Using the Device: A Quick Guide

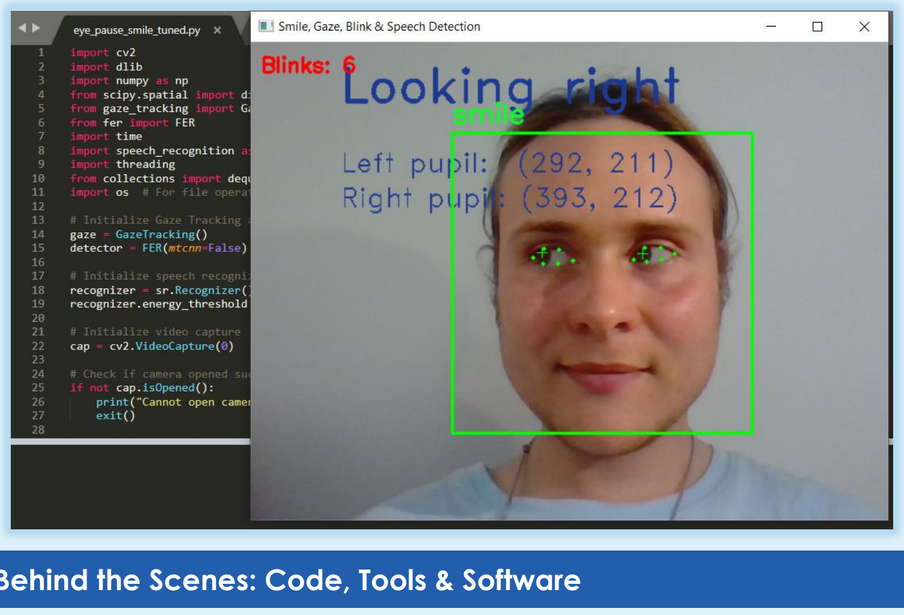
To use the **Vision-Activated Control System**, start by **uploading the main control sketch (written in C++)** to the **ESP8266 microcontroller** using the **Arduino IDE**. Once uploaded and powered, the ESP8266 begins running its code. The **LCD will first show "ESP Booting…"** and then the board's **IP address** when it **connects to Wi-Fi**.

Next, **run the Python-Based Signal Relay Module** on your computer. This script **communicates with the ESP8266** through **HTTP requests over Wi-Fi**. If all is working, the **main LED (D0) lights up**, showing the system is **ready but waiting to be activated**.

The system **activates when you smile at the webcam**. The **Python Facial Recognition Module sends a command (/S)** to the ESP8266, which then **turns off the main LED**, **turns on the "System Ready" LED (D8)**, and **updates the LCD** to say **"System Ready!"**.

Now, **left and right gazes control LEDs D5 and D6**. A **smile combined with three quick blinks** toggles the **motor using a relay connected to D4**. You can also **pause the system using voice commands** like **"Stop please,"** and **resume it** with phrases like **"Go on please."**

At the end of your session, just **close the webcam**. The **ESP8266 resets**, **switches off all LEDs and the motor**, and shows **"Cam Off"** on the LCD.



Running the Python-Based Facial Recognition Module.

A signal relay system continuously logs recognition events **in real-time**, mapping them to specific commands. When opened via **Python-Based Signal Relay Module**, these commands are then **transmitted via HTTP requests** to the **ESP8266 Hardware Control Module**, programmed in **Arduino IDE with C++**, allowing hardware components—LEDs, motors, and LCDs—to respond dynamically.

Behind the Scenes: Code, Tools & Software

The development process for this system was structured around **four core software modules**, each purpose-built to handle a specific part of the **user-to-hardware interaction pipeline**. We used a **split-stack architecture**: **Python** on the PC for *vision and logic*, and **C++** on the **ESP8266** for *hardware control*.

Python development was done using *Sublime Text 3.2.2* with **Python 3.8**, focusing heavily on **modularity** and **real-time performance**. The **Facial Recognition Module** was designed with **OpenCV**, **dlib**, **GazeTracking**, **FER**, and **SciPy** to extract and interpret *facial signals* such as **gaze direction**, **blinking frequency**, and **emotional expressions**. A key addition was **threaded voice command support** using **SpeechRecognition**, ensuring *non-blocking input handling*.

The **Signal Relay Module**, also in **Python**, handled the **communication layer**. It reads a **log file continuously updated** by the recognition script, maps the events to **pre-defined codes**, and sends **HTTP GET requests** to the ESP8266 controller using the **requests** library. **Multithreading** and **subprocess management** were used here to run *both modules in parallel*.

On the embedded side, the **ESP8266** was programmed using **Arduino IDE v2.3.3** with support libraries like **ESP8266WiFi.h** and **hd44780_I2Cexp.h**. The **Main Control Script** turns the microcontroller into a **web server**, receiving *HTTP commands* and executing *mapped actions* like **toggling LEDs** or **controlling a small motor**. The script was designed to be **non-blocking** and **state-aware**, ignoring inputs until **proper initialization** and *rejecting invalid commands*.

In addition, we implemented a **Component Testing Utility**, which runs independently on the **ESP8266**. It serves a **web-based control interface** accessible over **Wi-Fi**, allowing **direct toggling** of *individual devices* such as **LEDs** or the **LCD**. This utility was crucial during the **prototyping phase** to *validate connections* and *test behaviors* before integrating the full system.

Troubleshooting

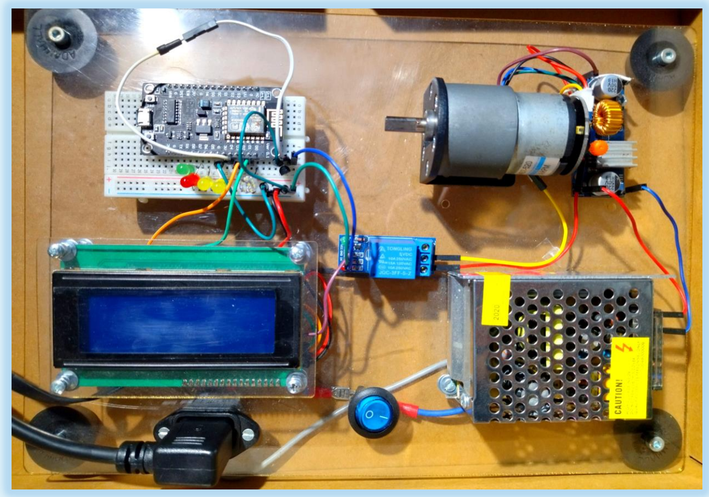
Make Sure You're Connected to the Right IP. For your computer to talk to the ESP8266 microcontroller, they both need to be on the same Wi-Fi network. The ESP8266 gets a new IP address from your router each time it connects — something like http://192.168.x.x.You'll see this address on the LCD screen after startup. This is the address the Python scripts use to send commands. If the IP doesn't appear or the device doesn't respond, restart the ESP8266 or check your router.

Update the Wi-Fi Info in the Code. Before uploading the C++ main script to the ESP8266 using Arduino IDE, don't forget to enter your Wi-Fi network name (SSID) and password in the code. Without this, the board won't connect to Wi-Fi, and nothing else will work.

Why does the IP change? It depends on your network setup and router. Unless set manually, the address is automatically assigned by your router (DHCP), so it may be different every time you connect.

If something doesn't turn on or react properly in this tester interface, it's a sign to double-check wiring, pin definitions, or the component itself. This tester is a great way to confirm everything works before running the main facial recognition and signal control system.

When Hardware Doesn't Respond. If the system runs but certain components don't respond—like an LED that won't blink, or the motor doesn't switch—it might be a hardware issue. To check this, you can use a special tool: the ESP8266 Hardware Component Tester. This is a separate Arduino sketch you upload to the ESP8266. Once it runs, it sets up a simple web-based control panel that lets you test each hardware component directly from your browser. Here's how it works: The ESP8266 shows its local IP (again, on the LCD).Type that address into your browser (e.g., http://192.168.43.174).You'll see buttons to test each part: LEDs, the LCD, the motor. Pressing a button sends a command to the ESP, and you'll instantly see if the part is working. The LCD can also show test messages that you can cycle through.



Basic Hardware Components of the Vision-Activated Control System

How It All Works Together: ESP8266, Python & Arduino

Everything begins in **Python**. The only script we run manually is the **Python signal relay script** — this acts as the *brainstem*, launching the **eye tracking and facial expression detection script** (**eye_pause_smile_tuned.py**) in a separate thread. That script uses a **camera** to analyze *gaze* and *facial features*, and then writes recognized actions (like a **smile** or *gaze direction*) as **signal codes** (**S, L, R, P1**, etc.) into a plain text file: **smile_gaze_signals.txt**.

The **relay script** continuously reads that file, *watching for new signals*. Once it spots one, it matches it to a **corresponding URL** like **http://192.168.43.174/S** using a *predefined dictionary*, and **sends an HTTP GET request** to the **ESP8266**.

The **ESP8266** receives this request through its *onboard web server* — it's programmed to parse the route (like **/S**) and execute the *right control command*. In the **Arduino code** (*which runs on the ESP8266*), each route triggers *specific GPIO pin changes*, like **turning motors, blinking lights, or activating relays**. This is where the **ESP8266 talks directly to the physical hardware** — via **digitalWrite()** or **analogWrite()**, depending on *what the actuator or sensor needs*.

There's **no classic serial communication** in this flow (e.g., **UART between PC and ESP**), because the **ESP8266 is connected over Wi-Fi** and uses **HTTP as the protocol** — *super simple, very IoT-style*.

The **LCD in this setup** is *connected directly* to the **ESP8266** and acts as a **real-time status display** for *received signals* and *system states*. Whenever the **ESP8266 receives an HTTP request** (like **/L** or **/SB**) from the **Python script**, it *processes that request* and **updates the LCD** with a **short label or code** that reflects the command — for example, showing **"LEFT"** when turning *left* or **"PAUSE"** when a *smile is detected*. This happens inside the **Arduino code** running on the **ESP8266**, where *each route handler* not only **controls GPIO pins** but also calls functions like **lcd.clear()** and **lcd.print()** to *update the screen*. The **LCD** is typically driven using the **LiquidCrystal_I2C** library, allowing **easy two-wire communication over I2C**, which *saves GPIO pins for other hardware*. It's mainly used here as a *debugging and feedback tool*, helping the *developer or user visually confirm* that a *command was received and processed correctly* without needing to check the *Serial Monitor* or *Python console*.

Adapting the System: One Design, Many Possibilities

This system is built to adapt. On the software side, the Python facial recognition script can easily be reconfigured to respond to different facial expressions or voice commands. Want to trigger an event when someone raises their eyebrows? Or issue a command when they say "open the door"? Just change the condition in the code or update the list of recognized phrases.

The ESP8266 is just as flexible. It's already controlling LEDs, a motor, and an LCD — but it can do much more. You can connect it to servo motors, smart locks, buzzers, light switches, irrigation valves, or even temperature sensors. Just add more GPIO outputs or I2C components.

This same structure could be used for:

•**Gesture-controlled light systems** for smart homes.

•**Smart wheelchair control**, triggered by facial gestures.

•**Factory machine toggling** with visual authentication for safety.

•**Interactive museum exhibits** that respond to gaze or emotion.

•**Contactless access systems** using personalized facial triggers.

•**Voice-triggered emergency** alert systems.

•**Hospital bedside tools** operated through facial recognition.

•**Educational kits** for teaching computer vision and IoT basics.

And more!

The code is modular. The hardware is expandable. Together, they form a base model ready to evolve into countless real-life applications.

Key References

1. **EDWARDS, G. J., TAYLOR, C. J., and COOTES, T. F., 1998** – *Interpreting Face Images Using Active Appearance Models* (IEEE International Conference)
2. **GAD, A.F., 2018** – *Practical Computer Vision Applications Using Deep Learning with CNNs* (Apress)
3. **GUIDO, S., 2016** – *Introduction to Machine Learning with Python* (O'Reilly Media)
4. **GUBBI, J., BUYYA, R., MARUSIC, S., and PALANISWAMI, M., 2013** – *Internet of Things (IoT): A Vision, Architectural Elements, and Future Directions* (Elsevier)
5. **LI, L., MU, X., LI, S., and PENG, H., 2020** – *A Review of Face Recognition Technology* (IEEE Access)
6. **MARCU, M., POPESCU, F.G., SLUSARIUC, R., and HANDRA, A.D., 2019** – *Wireless Control and Protection System for DC Motor Based on Microcontroller Embedded Algorithm* (University of Petrosani)
7. **NANDY, S. K., and SHESHADRI, H. S., 2017** – *Advancements and Recent Trends in Emotion Recognition Using Facial Image Analysis* (ICEECCOT Conference)
8. **PENTLAND, A., MOGHADDAM, B., STARNER, T., OLIYIDE, O., and TURK, M., 1993** – *View-Based and Modular Eigenspaces for Face Recognition* (M.I.T. Media Lab Technical Report)
9. **SZELISKI, R., 2011** – *Computer Vision: Algorithms and Applications* (Springer)
10. **STUART, S., ed., 2022** – *Eye Tracking: Background, Methods, and Applications* (Neuromethods, vol. 183, Springer)
11. **ZGHIGUO W., 2021** – *Eye-Tracking with Python and Pylink* (Springer Nature Switzerland AG)

The student developer: N. Anfilova
Email: nothing.plus@proton.me
+40729346937

The Faculty of Electrical and Mechanical Engineering
UPET, 2025