Facial Recognition in Embedded Systems

This work explores how facial interaction technologies can be integrated with embedded systems to create responsive, intelligent control interfaces that connect image recognition with electromechanical actions. The use of multiple biosignal modalities demonstrates the relevance of FICS in human–computer interaction, IoT, and assistive technology. The system manages LEDs, relays, motors, and displays, showing adaptability to different actuators and applications. The theoretical section reviews facial recognition technologies, covering computer vision, machine learning, and HCI principles, while tracing the evolution of FICS in automation and assistive contexts. The practical section presents the Vision-Activated Control System, which employs an ESP8266 microcontroller to process facial expressions, gaze, and blinks in real time. By integrating open-source vision and speech libraries with a wireless microcontroller, the system demonstrates how everyday devices can be operated through intuitive, hands-free inputs.

Natalia Anfilova earned a BA in Oriental Studies at St. Petersburg State University, Russia, with training in Indonesia and Germany. The author later obtained a degree in Electromechanical Engineering at the University of Petrosani, Romania, complemented by academic activities in Turkey, Belgium, and France.

9 786139 701698

Globe EDIT

Globe EDIT

Natalia Anfilova

Facial Recognition in Embedded Systems

Design and Development of a Vision-Activated Control System

Natalia Anfilova

Facial Recognition in Embedded Systems

Natalia Anfilova

Facial Recognition in Embedded Systems

Design and Development of a Vision-Activated Control System

GlobeEdit

Imprint

Cover image: www.ingimage.com

Publisher:
GlobeEdit
is a trademark of
Dodo Books Indian Ocean Ltd. and OmniScriptum S.R.L publishing group

120 High Road, East Finchley, London, N2 9ED, United Kingdom
Str. Armeneasca 28/1, office 1, Chisinau MD-2012, Republic of Moldova, Europe
Managing Directors: Ieva Konstantinova, Victoria Ursu
info@omniscriptum.com

Printed at: see last page
ISBN: 978-613-9-70169-8

UNIVERSITY OF PETROȘANI
FACULTY OF MECHANICAL AND ELECTRICAL ENGINEERING

FACIAL RECOGNITION IN EMBEDDED SYSTEMS:
DESIGN AND DEVELOPMENT OF A VISION-ACTIVATED CONTROL SYSTEM

Author: Ing. Natalia Anfilova

PETROȘANI

-2025-

Acknowledgements

I would like to express my deepest gratitude to my scientific coordinator, Lecturer Dr. Eng. Răzvan Slusariuc, Professor at the Faculty of Mechanical and Electrical Engineering, Department of Automation, Computers, Electrical and Energy Engineering – University of Petroșani. I am grateful for the privilege of being supported by a true expert in his field, with extensive experience in academic teaching and laboratory work in disciplines such as Electrical Machines, Electric Drive Systems, and Electrical and Electronic Measurements. He showed me that even small ideas can grow into meaningful projects. At the start, I was perhaps too humble about my own work, considering it little more than simple experiments. But with Dr. Slusariuc's help and involvement, I came to realize that my work was more significant than I had believed. Under his guidance, I developed my codes and hardware into a functional, flexible, and useful installation – a system that received recognition at the university and at ICIR-2025, as well as approvals for publications

I am equally grateful to Associate Professor Dr. Eng. Florin Gabriel Popescu, at the Faculty of Mechanical and Electrical Engineering, Department of Automation, Computers, Electrical and Energy Engineering – University of Petroșani. Through his teaching activities and the way he cultivates a supportive academic environment, he ensures that university life and relationships flow in a friendly and constructive manner. His willingness to help with every small detail allowed me to conduct my studies and experiments with ease, and to discover broader career opportunities. With people who listen and understand, true progress in science becomes possible – because science is not a rigid block of matter; it evolves through communication and human connection.

Finally, I would like to thank my close friends Cătălin Dobrean, Edi Besserman, and Dorin Neagoe, who once gave me my first microcontroller set as a birthday gift. Each of them has succeeded in his own field – graphic design, academic research, and business consulting – but at the time, I could not have predicted that their gift would become the seed of my own success. Out of curiosity, I began to program and experiment with it. Over time, it led me to explore numerous settings and other microcontrollers, gradually integrating them into my studies and work as an electrical engineer. Through this process, I came to understand what my friends meant by being not only "busy", but truly passionate about a subject. For sharing that spark with me, I thank them immensely.

Table of Contents

Appendices

a) Python-Based Source Code B-l
b) Python-Based Source Code B-ll
c) ESP8266-Based Hardware Control Modules D-l
d) ESP8266-Based Hardware Control Module D-llE. Vision-Activated

Part I: Introduction

In recent years, the demand for intuitive and contactless interaction technologies has led to rapid advancements in facial recognition, embedded microcontrollers, and IoT communication protocols. This project presents the development of a vision-activated control system that leverages facial gestures and voice cues to operate hardware components in real time. Combining machine learning algorithms for facial feature analysis with Python-based scripting and C++ firmware for embedded control, the system demonstrates how non-contact human inputs can be translated into precise actuation of LEDs, motors, and displays. The integration of these technologies not only reflects current trends in human-computer interaction but also showcases a scalable and cost-effective approach to smart device interfacing.

1.1. Project Objectives and Expected Outcomes

The primary objective is to design and implement a low-cost, vision-activated control system that translates facial gestures and simple voice commands into hardware actions via an IoT-enabled microcontroller. Specific goals include:

1. Designing and constructing the hardware control circuit: interconnecting the ESP8266 with five status LEDs, a relay-driven DC motor, and an I²C LCD1602 module, powered via regulated 3.3 V and 5 V supplies with proper isolation and voltage regulation.

2. Developing an integrated software architecture consisting of Python-based modules for real-time facial gesture and voice detection using OpenCV and Dlib (Bradski, 2000; Kazemi & Sullivan, 2014), and embedded C++ firmware on the ESP8266 (Arduino IDE) that interprets HTTP requests and maps them to GPIO actions. The system must ensure reliable, low-latency relay of commands from the host PC to the microcontroller over Wi-Fi, enabling dynamic control of LEDs, a relay-driven motor, and an LCD interface (Atzori, 2010; Gubbi, 2013).

3. Establishing and validating the end-to-end communication pipeline between the Python signal generator and the ESP8266 HTTP server, including thread-safe file I/O, HTTP client implementation in Python, and URL parsing in microcontroller code.

4. Evaluating system performance quantitatively in terms of recognition accuracy, end-to-end latency, and robustness under varied lighting and acoustic conditions (MDPI, 2023).

Expected outcomes include a documented prototype demonstrating ≥90% facial gesture accuracy, voice command reliability above 95%, and actuator response within

200 ms. The system's modular architecture will allow future expansion to additional gestures, voice intents, and hardware devices.

1.2 Significance of Vision-Activated Control Systems

The development of vision-activated control systems marks a shift toward more intuitive, accessible, and hygienic interaction paradigms. These systems rely on facial features—such as eye movement, blinking, and smiling—as primary input vectors, offering users the ability to trigger system functions without physical contact. This mode of control is especially important in settings that demand hands-free operation, such as medical environments, industrial production lines, or assistive living facilities (Alvappillai, 2024).

One of the core motivations behind the significance of these systems lies in their capacity to enable contactless interfaces in a post-pandemic world, where minimizing physical touchpoints in public and private systems has gained new urgency. Additionally, they offer significant potential for empowering individuals with motor impairments who may not be able to operate traditional interfaces. By integrating visual gesture recognition into embedded systems—especially using microcontrollers like the ESP8266 that support wireless communication and HTTP-based control protocols—developers can implement responsive, low-latency interfaces with minimal resource overhead (Marcu , 2019).

Moreover, the dual-software architecture typically employed in such systems—Python for facial and voice signal processing, and C++ (via Arduino IDE) for device control—provides an excellent pedagogical framework for teaching modular system design. It also aligns with real-time constraints in robotics and automation, where embedded devices must respond promptly to sensor-driven commands. The integration of these elements illustrates how vision-activated systems contribute not only to user comfort and accessibility but also to the broader advancement of embedded intelligence in the Internet of Things (IoT) ecosystem (Atzori, 2010; Lopes, 2015).

1.3 Project Structure

This bachelor project is organized into four main parts. Part I (Introduction) outlines the project framework, objectives, and the broader importance of vision-activated control systems. Part II (Theoretical Foundations) reviews the principles and technologies underlying Facial Interaction Control Systems (FICS), including computer vision, machine learning, and human-computer interaction (HCI) concepts. Part III (Practical System Design) describes the hardware and software architecture of

the prototype, detailing the ESP8266 microcontroller, motor and relay control, LED/LCD integration, power supplies, and the four software modules: the Python-based Facial Recognition Module (B-I), the Signal Relay Module (B-II), the ESP8266 Hardware Control Module (D-I), and the Diagnostic Tester (D-II). Part IV (Technical Integration and Optimization) covers the synchronization of multiple Python models, intermodule communication, hardware troubleshooting, and proposed enhancements. The Conclusions summarize key findings; References list all cited works; Appendices provide schematics, source code listings, and user guides.

Part II: Theoretical Foundations of Facial Interaction Control Systems (FICS)

This chapter provides the theoretical framework underpinning Facial Interaction Control Systems (FICS). It begins with an overview of FICS, defining them as multimodal systems that translate facial and vocal input into machine-readable signals. The section then explores core enabling technologies: computer vision and image processing techniques for detecting and interpreting facial features; machine learning methods—particularly deep learning—for classification and pattern recognition; and Human-Computer Interaction (HCI) principles that ensure systems are accessible, efficient, and user-centric. Finally, it highlights real-world applications of FICS in security, healthcare, smart environments, and assistive technologies. This part is essential for contextualizing the technical and societal relevance of FICS in modern embedded systems.

2.1 Overview of FICS Technology

This section provides definitions and general overview of Facial Interaction Control Systems (FICS) and its evolution, while subchapter 2.2 delves into specific technical underpinnings like computer vision algorithms, machine learning models for facial recognition, and HCI principles. Likewise, subchapter 2.3 focuses on real-world applications, which will extend the discussion on how FICS technology is implemented across industries. This subchapter introduces the reader into a general understanding of FICS.

Facial Interaction Control Systems are interactive systems designed to capture real-time facial data using advanced imaging sensors and algorithms (Paiva, Prada, 2007). These systems interpret subtle facial gestures and expressions, transforming them into actionable commands for system control, thereby enhancing human–machine communication (Li, 2020).

FICS have emerged as an interdisciplinary field that integrates aspects of computer vision, machine learning, and human-computer interaction (HCI) to facilitate non-contact, intuitive control of devices and systems. At its core, FICS enable systems to interpret human facial features and expressions, converting them into actionable commands. This process relies on the systematic capture of facial data, its digital processing, and subsequent decision-making algorithms that translate observed cues into control signals (Nandy, 2017).

The evolution of FICS can be traced to significant advancements in digital imaging and sensor technology. Early systems focused solely on static facial recognition for

authentication purposes[1] (Pentlad, Turk, 1993). Over time, these systems evolved to incorporate dynamic facial analysis—tracking expressions, gaze direction, and even micro-movements.

Over time, facial recognition systems transitioned from static to dynamic analysis by incorporating various emerging technologies. Active Appearance Models (AAM) provided a way to capture and analyze subtle changes in facial expressions and structure in real time (Edwards, Taylor, 1998). The Facial Action Coding System (FACS) was adapted to quantify micro-expressions, enabling systems to interpret minute muscle movements (NOLDUS, 2024). Convolutional Neural Networks[2] (CNNs) were introduced to process large volumes of facial data and improve the accuracy of dynamic recognition (Sewak, 2018). Gaze tracking technology[3] evolved to use infrared illumination and high-speed cameras, capturing rapid eye movements and providing precise tracking of gaze direction (Stuart, 2022). Facial landmark detection algorithms, such as those implemented in libraries like dlib, and 3D face reconstruction techniques have further enhanced the ability to interpret and map facial gestures dynamically (Zghiguo, 2021).

Such evolution has allowed FICS to become more robust and responsive in real-time applications (Stuart, 2022). FICS now support a broad range of functions from simple command input to more complex tasks such as interactive gaming, assistive technology for users with disabilities, and even behavioral analytics in consumer electronics (INNOVATRICS, 2022).

FICS are characterized by their capacity to bridge the gap between biological cues and digital commands, fostering a more natural interaction environment. This paradigm shift in control interfaces not only improves usability and accessibility but

[1] For example, early FICS leveraged foundational technologies such as the Eigenfaces method, which was introduced by Turk and Pentland in 1991. This technique uses principal component analysis to represent facial features from static Images for authentication. Early systems also depended on Charge-Coupled Device (CCD) cameras to capture high-quality digital images, a critical factor for accurate facial data acquisition. Additionally, the FBI's early adoption of facial recognition for biometric authentication paved the way for security applications (Pentlad, Turk, 1993).

[2] CNNs are a class of deep learning algorithms designed to process data with a grid-like topology, such as images. They consist of multiple layers, including convolutional layers that apply filters to detect patterns, pooling layers that reduce dimensionality, and fully connected layers that perform classification. CNNs are particularly effective in image recognition and computer vision tasks due to their ability to learn hierarchical feature representations (Legrady, 2022).

[3] This technology is especially relevant to our project, as it forms a key component of the Python-Based Facial Recognition Module. By utilizing facial landmark detection algorithms, such as those in the dlib library, our system accurately identifies and tracks eye movements. This enables the detection of gaze direction, which is then translated into actionable signals for the ESP8266 microcontroller.

also pushes the boundaries of automation and smart technology integration. As FICS continue to evolve, they promise to further enhance the efficiency of human-machine communication by offering more adaptive, context-aware interactions (Stuart, 2022).

2.2 Core Technologies in FICS

This subchapter outlines the foundational technologies that enable Facial Interaction Control Systems (FICS). It begins by detailing how computer vision and image processing methods are employed to detect and interpret facial features, converting raw image data into quantifiable patterns using algorithms such as SIFT, HOG, and LBP. Machine learning—particularly deep learning through convolutional neural networks (CNNs)—plays a central role by allowing systems to learn from large datasets and adapt to diverse input conditions. Additionally, Human-Computer Interaction (HCI) principles ensure that FICS are accessible, intuitive, and responsive by guiding interface design and supporting effective feedback mechanisms. These three technological pillars collectively support the development of reliable and user-centered FICS platforms. They also establish the conceptual and technical groundwork upon which the current project is built.

2.2.1 Computer Vision and Image Processing

Computer vision and image processing constitute the foundational framework for extracting quantitative information from digital images in Facial Interaction Control Systems. These technologies transform raw pixel data into measurable features through a series of algorithmic operations (Szeliski, 2011).

Image processing converts raw pixel data into measurable features through several steps. First, preprocessing improves image quality by adjusting brightness, contrast, and removing noise. Next, feature extraction identifies important patterns, such as edges, textures, and shapes. Edge detection methods like Sobel or Canny (Distante, 2020; Szeliski, 2011) highlight object boundaries, while texture analysis techniques like Local Binary Patterns (LBP) capture surface details. Shape-based features help detect geometric structures. Finally, feature representation transforms these extracted patterns into numerical descriptors, such as histograms, making the data easier to analyze. These steps allow computers to interpret and process images effectively (API4AI, 2024).

Digital filtering techniques are applied using convolutional operators, such as gradient and Laplacian filters, to perform edge detection and suppress noise. The resulting data undergoes segmentation and thresholding to isolate regions of interest,

while morphological operations refine the structural analysis of facial components (Distante, 2020).

Advanced mathematical transformations, including Fourier and wavelet decompositions, are employed to analyze the frequency characteristics of images. Feature extraction algorithms—such as Scale-Invariant Feature Transform (SIFT), Speeded-Up Robust Features (SURF), and Histogram of Oriented Gradients (HOG)—facilitate the detection and localization of key facial landmarks (Szeliski, 2011). These methods are optimized to operate under real-time constraints and high computational loads. The integration of these processing techniques provides a robust mechanism for converting visual information into actionable parameters within the system.

2.2.2 Machine Learning in Facial Recognition

Machine learning (ML) is a branch of artificial intelligence (AI) that enables systems to learn patterns and make predictions from data without explicit programming. It is defined as the process of training algorithms to recognize complex relationships using statistical techniques. ML is crucial in modern computing, powering applications in healthcare, finance, security, and automation. By adapting to new data, ML enhances decision-making, automates processes, and improves efficiency in various domains (Gudio, 2016).

Facial recognition relies on ML to identify and verify individuals by analyzing facial features. The process involves multiple stages: image preprocessing, feature extraction, and classification (Alvappillai, 2024).

Preprocessing techniques, such as histogram equalization and geometric transformations, standardize input images. Traditional feature extraction methods, including HOG and Scale-Invariant Feature Transform (SIFT), identify key facial landmarks. However, modern systems utilize deep learning, particularly CNNs, to automatically extract robust features from raw images (Gad, 2018).

Once features are extracted, classification models map them to identities. Earlier methods used SVMs and decision trees, while modern approaches employ deep neural networks. Frameworks like TensorFlow streamline training and deployment. Despite advancements, challenges remain, including dataset bias, computational costs, and adaptability to varying conditions (Gad, 2018).

ML has significantly improved the accuracy and efficiency of facial recognition, enabling its integration into security systems, mobile authentication, and surveillance. However, challenges such as ethical concerns, privacy risks, and adversarial attacks remain critical (Gudio, 2016). Ongoing research focuses on enhancing fairness,

robustness, and interpretability to ensure the responsible deployment of ML-based facial recognition technologies.

2.2.3 Human-Computer Interaction (HCI) Principles

Human-Computer Interaction (HCI) encompasses the systematic design, evaluation, and implementation of interactive systems. HCI integrates theories from cognitive psychology, ergonomics, and human factors engineering to optimize user experience and system efficiency. It employs rigorous methodologies such as cognitive walkthroughs, heuristic evaluation, and task analysis to identify usability issues and improve interface design (Sears, Jacko, 2009). Standard models of information processing and cognitive load theory inform the iterative design process, ensuring that system interactions minimize user fatigue and error (MEDIUM, 2023).

Human-Computer Interaction principles are critical in the development of Facial Interaction Control Systems, ensuring that these systems are efficient, user-friendly, and ethically sound. Such principles are summarized further, according to the "Principles of Human-Computer Interaction" course available on YouQ.ai, a reliable platform dedicated to skill development and knowledge-sharing (YOUQ.AI, 2022), and the research fundamental on “Human-Computer Interaction Principles” (Sears, Jacko, 2009):

UCD[4] emphasizes involving end-users throughout the design process to ensure that the system meets their needs and preferences. In FICS, this involves conducting user research to understand how individuals interact with facial recognition technologies and tailoring system functionalities accordingly. This approach enhances usability and user satisfaction.

Uniformity in interface design elements, such as icons, terminology, and workflows, allows users to predict system behavior, reducing cognitive load. In FICS, consistent feedback mechanisms and standardized interaction patterns enable users to learn and navigate the system more effectively.

Feedback and feedforward help users understand the outcomes of their actions. In FICS, visual or auditory cues can indicate successful recognition or prompt users to adjust their facial positioning. Feedforward mechanisms guide users on how to interact with the system before actions are taken, enhancing the overall user experience.

[4] User-Centered Design (UCD) is an iterative design process that prioritizes end-users throughout every phase of product development. This approach ensures that the final product aligns with users' needs, goals, and feedback, leading to enhanced usability and satisfaction. In UCD, design teams actively involve users through various research and design techniques to create highly usable and accessible products (BAYMARD INSTITUTE, 2022).

Accessibility implies **e**nsuring that FICS are usable by individuals with diverse abilities is paramount. Incorporating multimodal interaction techniques, such as combining facial recognition with voice commands or touch inputs, can accommodate users with varying needs. For example, capacitive facial movement detection allows users to interact with systems through gestures like frowning or raising eyebrows, providing alternative input methods for individuals with mobility impairments.

Affordances and signifiers. Interface components should visually suggest how users can interact with them, such as highlighting areas where facial alignment is required or indicating active recognition zones. Clear signifiers prevent user confusion and errors.

Ethical principles, such as the Three Laws of Biometrics[5], ensures that FICS respect user privacy and rights. Implementing policies that align with these principles fosters public trust and responsible deployment of facial recognition technologies.

Within Vision-Activated Control System, HCI principles play a pivotal role in mediating the interaction between automated facial recognition outputs and human operators. The system integrates dynamic visual cues from facial expressions with real-time feedback, ensuring that user intent is accurately captured and executed. This integration supports the transition from static command inputs to adaptive, context-aware control interfaces. In effect, HCI not only governs the visual design of the system but also drives the underlying algorithms that interpret user behavior. It extends beyond making the system visually appealing or easy to navigate—it also shapes how the system interprets and responds to user actions.

2.3 Applications of FICS in Industry and Daily Life

Facial Interaction Control Systems (FICS) are applied in advanced biometric security solutions. The fusion of deep learning models with facial landmark analysis improves the reliability of these applications under variable environmental conditions.

For example, Apple's Face ID employs infrared depth sensors[6] and CNNs authenticate users with high precision (TECHREPUBLIC, 2020). In commercial

[5] The Three Laws of Biometrics—Policy, Process, and Technology—mandate that biometric systems prioritize human rights and privacy, ensure fair and accountable operational procedures, and rely on a deep technical understanding to mitigate risks and limitations. The Laws were established by the Biometrics Institute to promote the responsible and ethical use of biometric technologies, inspired by Isaac Asimov's Three Laws of Robotics (BIOMETRICS INSTITUTE, 2021).

[6] Infrared (IR) depth sensors are devices that measure the distance between the sensor and objects in its environment by utilizing infrared light. These sensors project infrared patterns onto a scene and capture the reflected light to calculate depth information. This technique is widely used in applications requiring three-dimensional (3D) spatial data, such as gesture recognition and object tracking (Legrady, 2022).

surveillance, NEC's NeoFace technology integrates real-time facial recognition with sophisticated analytics to control access and monitor environments. Such systems are embedded in smart terminals and ATMs to enhance security and reduce fraud (NEC CORPORATION, 2020; BIOMETRIC UPDATE, 2024).

In healthcare, FICS support patient monitoring through automated facial analysis. Systems developed by Philips and GE Healthcare detect pain, fatigue, and other clinical indicators via micro-expression analysis and emotion recognition algorithms (PHILIPS, 2023). In industrial and commercial settings, FICS enable interactive retail kiosks and smart advertising. Cloud-based services, such as Amazon Rekognition and Microsoft Azure Face API, process large volumes of facial data to deliver targeted marketing and customer engagement analytics (AWS, 2025).

In daily life, such systems enable adaptive interfaces in consumer electronics and smart home devices. Vision-activated control supports gesture-based navigation in interactive entertainment and augmented reality[7] (Szeliski, 2011).

Assistive technologies leverage FICS to provide alternative communication channels for users with mobility impairments. Furthermore, real-time emotion recognition and gaze tracking are integrated into driver monitoring systems to enhance road safety. For individuals with disabilities, FICS provide alternative communication methods. By interpreting facial gestures, these systems enable users to control devices and interact with their environment, promoting greater independence (Anson, 2018).

The scalability and convergence of these technologies promote intelligent automation in urban environments. The integration of IoT[8] sensors and AI in urban infrastructure enables real-time data collection and analysis, leading to more efficient traffic management and energy distribution. For instance, AI-powered traffic lights have been trialed in the UK to prioritize cyclists and pedestrians, enhancing urban mobility and safety (Ferreria Dos Santos, De Matos, 2025).

The integration of Facial Interaction Control Systems across these domains demonstrates their versatility and potential to enhance user experiences through intuitive and secure interactions.

[7] In "Computer Vision: Algorithms and Applications", Richard Szeliski defines Augmented Reality (AR) as a technology that overlays virtual objects onto the real-world environment in real time, enhancing user perception by combining computer-generated elements with physical surroundings.

[8] The Internet of Things refers to a network of interconnected physical devices embedded with sensors, software, and other technologies that enable them to collect and exchange data over the internet without human intervention (Gubby, Buyya, 2013).

Part III: Practical System Design of the Vision-Activated Control System

The current chapter describes the full implementation of the Vision-Activated Control System, detailing its hardware architecture, software structure, and operational workflow. It outlines how the ESP8266 microcontroller, LEDs, motor, relay, and I²C LCD are physically integrated and powered, enabling real-time responses to user gestures and voice input. The system architecture is modular, with Python-based modules handling facial recognition (B-I) and signal relay (B-II), while ESP8266 firmware modules (D-I, D-II) manage device control and hardware testing. Each component—from startup to shutdown—is coordinated through clearly defined protocols and feedback mechanisms. This part is essential as it translates theoretical design into a functioning embedded system.

3.1 System Overview

This subchapter presents the architecture and operational logic of the vision-activated control system built around the ESP8266 microcontroller. It outlines the integration of peripheral components—including LEDs, a relay-controlled DC motor, and an I²C LCD1602 display—powered by a regulated 5V supply and controlled via GPIO pins. The microcontroller receives HTTP[9] commands from a Python-based vision module and translates them into hardware actions such as activating LEDs, toggling the motor, and updating display messages. The system provides real-time feedback through LED indicators and LCD outputs, enabling intuitive user interaction via facial gestures and voice commands. Operational procedures, including system startup, activation, control, and shutdown, are clearly defined to ensure reliable and responsive performance. Overall, this section details how the hardware and software components cohesively operate to enable contactless, multimodal interaction through embedded control.

3.1.1 General Architecture

The circuit is designed around the ESP8266 (NodeMCU 0.9) microcontroller[10], which serves as the central control unit, receiving signals from the external Python

[9] HTTP (Hypertext Transfer Protocol) is an application-layer communication protocol used for transmitting data over a network, typically between a client and a server. In the context of this project, HTTP provides the core mechanism for communication between the Python-based signal relay module (client) and the ESP8266 microcontroller (server) (MDN Web Docs, n.d.).

[10] The ESP8266 (NodeMCU 0.9) microcontroller was selected for this project due to its low cost, built-in Wi-Fi capabilities, and extensive community support. Unlike many microcontrollers that require external modules

script and executing corresponding hardware actions. The system is powered by a regulated 5V power supply, with the ESP8266 managing multiple peripheral components, including LEDs, a motor, a relay, and an LCD display.

The LEDs are connected to various digital GPIO pins of the ESP8266. The main indicator LED (D0) provides system status feedback, blinking in response to certain events. The left (D5) and right (D6) LEDs correspond to gaze direction detection signals, illuminating based on whether the user looks left or right. The pause indicator LED (D7) signals when the system is paused, while the system-ready LED (D8) confirms when the ESP8266 is fully operational after receiving the initial activation signal from the Python script.

The motor is controlled via a relay (K1), which acts as an intermediary switch (OURPCB, n.d.), allowing the ESP8266 to turn the motor on and off in response to specific input signals, such as the detection of a smile and multiple blinks. The motor speed is regulated through an external speed controller, ensuring smooth operation based on predefined settings.

The 16×2 LCD module, interfaced with the ESP8266 via I2C, provides real-time feedback, displaying system status updates (SUNFOUNDER, 2017), including Wi-Fi connectivity, detected facial expressions, and received signals. The ESP8266 communicates with the external Python script over Wi-Fi, processing HTTP requests to execute actions corresponding to smile, blink, and gaze detection. Upon receiving commands, the microcontroller actuates the appropriate LEDs, toggles the motor, and updates the LCD display accordingly.

The circuit also includes a switching power supply, stepping down the 220V AC main grid input to a stable 12V DC supply, which is further regulated to provide the required 5V operating voltage for the ESP8266 and peripheral components. The integration of relays and transistor-based switching mechanisms ensures efficient power management, isolating high-power components from the microcontroller (Boylestard, 2012).

Overall, this circuit enables a seamless interface between the facial recognition software and physical hardware, translating real-time user interactions into controlled mechanical and visual outputs through the ESP8266 microcontroller.

for wireless communication, the ESP8266 integrates IEEE 802.11b/g/n Wi-Fi directly on chip, making it ideal for IoT applications with minimal hardware overhead (MAKE-IT.CA, n.d).

3.1.2 Operational Procedure

This subsection presents the step-by-step operational procedure of the system, highlighting how facial gestures and voice commands are used to control LEDs and a motor via the ESP8266 microcontroller. The process spans system startup, activation, interactive control, and safe shutdown, with real-time feedback provided through LEDs and an LCD display.

The Module Startup of the system begins with the ESP8266 microcontroller, preloaded with the appropriate control firmware using the Arduino IDE (Marcu, 2019). Once powered, the module automatically executes its embedded code and initializes its onboard HTTP server (Sewak, 2018). Concurrently, the LCD connected to the ESP8266 displays a boot message indicating the start of system routines, typically appearing as “ESP Booting…”. Following successful connection to a Wi-Fi network, the LCD updates to show the device’s dynamically assigned IP address. This address is critical for communication between the Python-based control modules and the ESP8266.

With the microcontroller fully initialized, the Signal Relay Python script is launched on the host computer. Upon startup, this script establishes communication by sending an initial signal ("/W") to verify connectivity and initiate baseline system status. At this stage, the main white LED (connected to D0) remains illuminated, signifying that the system is powered and awaiting user interaction for activation, while other indicators remain inactive until further commands are received.

System Activation requires a smile gesture. When the webcam detects a smile, the Facial Recognition Module issues an HTTP request (/S) to the ESP8266. On receipt:

- The main LED (D0) turns off.
- The system-ready LED (D8) turns on.
- The LCD displays “System Ready!”.

After activation, all control commands become valid.

Gaze-Based LED Control. Gaze direction triggers the orange status LEDs:

- **Right Gaze** (each rightward glance causes the Facial Recognition Module to send /R. The ESP8266 then turns on LED D6 for 3 s. The LCD shows “Right LED On 3s”).
- **Left Gaze** (each leftward glance causes the Module to send /L. The ESP8266 then turns on LED D5 for 3 s. The LCD shows “Left LED On 3s”).

Other gaze directions, while not serving as commands, are still registered by the camera and announced as the written text on the camera screen.

Blink-and-Smile Motor Control. To toggle the motor, the user must smile and blink three times in rapid succession. The Facial Recognition Module interprets this pattern and issues /SB. Upon receipt, the ESP8266 changes the relay state on D4. The motor either starts or stops. The LCD updates to “Motor: ON” or “Motor: OFF” accordingly.

Voice-Controlled Pause and Resume. The system supports two voice commands: “pause” and “resume.”

- **Pause (/P1)** (when the Pause command is recognized, the Facial Recognition Module sends /P1. The ESP8266 sets its internal pause flag. LED D7 turns on; LED D8 turns off. All gaze, smile and blink inputs are ignored until resume. The LCD shows “System Paused.”)

The following spoken phrases trigger the pause state: “Take a break please”, “Take a pause please”, “Stop please”, and “You can rest”.

- **Resume (/P0)** (when the Resume command is recognized, /P0 is sent. The ESP8266 clears the pause flag. LED D8 turns on; LED D7 turns off. The LCD shows “System Resumed.” Normal operation recommences).

Recognized voice phrases to resume the system include: “Go on please”, “Continue please”, “Shall we continue”, and “Let's go on”.

System Reset and Shutdown. To terminate the session, close the webcam application. The Signal Relay Module sends the finish command (/F) to the ESP8266. The board executes resetSystem(), which performs the following:

- All LEDs except D0 are switched off.
- The motor relay is deactivated.
- The LCD displays “Cam Off” and “System Reset.”

To restart, repeat Module Startup and System Activation.

Figure 1. ESP8266 GPIO assignments and their mapped behaviors based on incoming URL commands.

GPIO (NodeMCU)	Component	Command & Behavior
D0 (GPIO16)	Main LED (readyLed)	Startup: ON; /S: OFF (System Ready); /F: ON (Reset)
D5 (GPIO14)	Left LED (ledA2)	/L: ON for 3 s, LCD shows "Left LED On 3s"
D6 (GPIO12)	Right LED (ledA1)	/R: ON for 3 s, LCD shows "Right LED On 3s"
D7 (GPIO13)	Pause LED (ledD1)	/P1: ON (Paused), LCD shows "System Paused"; /P0: OFF (Resumed)
D8 (GPIO15)	Ready LED (ledD2)	Initially OFF; /S: ON; /P1: OFF (Pause); /P0: ON (Resume)
D4 (GPIO2)	Motor/Relay (motorPin)	/SB: Toggles ON/OFF (Motor Control), LCD shows "Motor: ON" or "Motor: OFF"

All interactions occur via HTTP requests on port 80 (MDN Web Docs, n.d.). The ESP8266 runs a simple HTTP server. The user interface consists of LEDs and the LCD, which confirm each state transition in real time.

3.2 Hardware Components

Within electrically powered installations, the electromechanical conversion of energy is performed, the electric motor performing the role of electric energy converter in mechanical energy and sometimes in regenerative braking mode, the mechanical energy converter according to the operating conditions determined by the technological process carried out by the executive or working mechanism (Marcu, Popescu, 2019). This part shows the key hardware modules used to implement the embedded control system. The NodeMCU 0.9 (ESP8266) microcontroller manages communication and actuation, serving as both a Wi-Fi client and GPIO controller (MAKE-IT.CA, n.d.). It directly powers and switches five color-coded LEDs to indicate user input and system states, while also triggering a DC motor via a relay mechanism in response to combined facial cues. The LCD1602 I²C module provides textual output and operates independently on a stable 5V USB supply, with contrast adjusted manually to enhance visibility (SUNFOUNDER, 2017). A Mean Well S-25-15 power supply regulates motor voltage and interfaces with a speed controller for fine-tuned actuation (DESSY, n.d.). Together, these components establish a compact, reliable, and modular system for interpreting and responding to facial and vocal inputs in real time.

3.2.1 ESP8266 Microcontroller

The NodeMCU 0.9 is an open-source development board that integrates the ESP8266 Wi-Fi System-on-Chip (SoC), designed specifically for Internet of Things (IoT) applications. This board combines a microcontroller with built-in Wi-Fi capabilities, offering a cost-effective and efficient solution for wireless communication projects (MAKE-IT.CA, n.d.).

In the context of this vision-driven automation system, the NodeMCU 0.9 serves as the actuator controller and local web host, translating abstract software commands into concrete physical actions. Its embedded Wi-Fi connectivity is leveraged to receive HTTP requests from the Python signal relay script, which is triggered by facial signal detections such as smiling or gaze shifts. Upon receiving these commands, the microcontroller updates the state of attached hardware components, including status LEDs and a motor unit, thus completing the signal-to-action pipeline. The inclusion of a diagnostic web interface also enhances system robustness, allowing developers to verify the operational integrity of individual components independently of the vision pipeline. This layered usage of the NodeMCU exemplifies its versatility and vital role in bridging cloud-free vision processing with embedded actuation. Additionally, it supplies the LED indicators with 3.3 V required for their optimal functioning (MAKE-IT.CA, n.d.).

At its core, the NodeMCU 0.9 features the ESP8266 microcontroller, which is based on the Tensilica Xtensa LX106 32-bit RISC architecture[11]. This processor operates at a default clock speed of 80 MHz, with the capability to reach up to 160 MHz, providing sufficient processing power for various IoT tasks. The board is equipped with 128 KB of RAM and 4 MB of flash memory, allowing for the storage of firmware and user applications (COMPONENTS101, 2023).

The NodeMCU 0.9 supports the 802.11 b/g/n Wi-Fi standards[12], enabling it to connect to wireless networks and serve as a client or access point. It includes a full TCP/IP stack[13], facilitating seamless integration with internet-based services. The

[11] The ESP8266 microcontroller is built on the Tensilica Xtensa LX106 32-bit RISC architecture, a highly configurable low-power processor core optimized for embedded and IoT applications. It balances computational performance with energy efficiency, supporting clock speeds up to 160 MHz, which is suitable for real-time control and networked operations (COMPONENTS101, 2023).

[12] These are wireless networking standards defined by the IEEE 802.11 family. The ESP8266 supports 802.11 b/g/n, enabling it to connect to a wide range of Wi-Fi networks. This allows for high data rates, compatibility with legacy networks, and interoperability with most wireless routers used in home or industrial environments (MAKE-IT.CA, n.d.).

[13] A full TCP/IP stack refers to the complete set of communication protocols that allow the ESP8266 to handle internet-level communication, including establishing server/client connections, processing HTTP requests, and

board provides 11 digital input/output pins and a single analog input pin, offering flexibility for interfacing with various sensors and actuators (MAKE-IT.CA, n.d.).

Physically, the NodeMCU 0.9 measures approximately 49mm x 26mm, with a pin spacing of 0.9 inches between rows, making it compatible with standard breadboards for prototyping. It utilizes the CP2102 USB-to-serial converter[14], allowing for straightforward programming and communication via a micro-USB connection. The board operates at a voltage of 3.3V and can accept input voltages ranging from 4.5V to 10V, thanks to its onboard voltage regulator (MAKE-IT.CA, n.d.).

3.2.2 Motor and Relay Control Mechanisms

The system uses a Chihai GM37-520 DC gearmotor, a compact brushed DC motor with internal metal planetary gears. It is typically rated for a 6–24 V supply (we use 12 V) and draws a small no-load current (≈0.05–0.1 A) rising to several amps at stall. No-load speed depends on gear ratio; for example, a 6 V version with moderate gearing runs ~800 RPM no-load, so a 12 V version would run roughly 1,500 RPM unloaded. Under load it draws hundreds of milliamps; one 6 V/800 RPM spec shows ~30 mN·m torque at 1.5 A (≈60 mN·m stall at 3.0 A). A high-reduction 12 V variant (86:1) delivers ~0.25 N·m (2.5 kg·cm) rated torque at 0.3 A, with stall torque ≈1.37 N·m at 1.5 A. These figures give a sense of performance: the motor can produce tens to hundreds of mN·m of torque and on the order of 0.1–1 A at nominal load, with stall currents up to a few amps (DESSY, n.d.).

The motor is connected as follows (see Circuit Schematics): the positive terminal M2+ goes to the relay's normally-open (NO) contact, and the negative terminal M2– goes directly to the regulator's VOUT– (common ground). The relay's common (COM) terminal is tied to the speed regulator's VOUT+ output. Thus when the relay is energized, COM closes to NO and the regulator's positive voltage is fed to the motor. The speed regulator is a PWM DC–DC module that allows manual speed adjustment.

The speed-regulator board contains a potentiometer (the orange rotary screw), a power MOSFET on an aluminum heat sink, several large electrolytic capacitors (220 μF/50 V), and a yellow toroidal inductor. This indicates it is a switching (buck) converter used for motor speed control. The potentiometer sets the duty cycle of the

transferring data packets. This built-in networking capability makes the ESP8266 suitable for direct integration into IoT systems without additional modules (MDN Web Docs, n.d.).

[14] The CP2102 is an integrated USB-to-UART bridge controller that allows microcontrollers like the ESP8266 to be programmed or communicated with using a standard USB connection. It enables seamless driver installation and is widely supported across Windows, macOS, and Linux environments, simplifying development and deployment (MAKE-IT.CA, n.d.).

switching transistor: turning the screw changes the MOSFET's on/off duty ratio, which varies the average output voltage between 0 and 12 V. The MOSFET generates a pulse-width-modulated waveform that is smoothed by the LC filter (inductor + capacitors) into a steady DC level. In effect, the output voltage (VOUT+) is adjusted continuously by the pot position, so that higher screw settings give higher motor voltage (and speed) up to 12 V. The aluminum heat sink dissipates heat from the MOSFET when driving substantial current.

This regulator module receives its input from the S-25 switching supply: the S-25's +12 V output connects to VIN+ and its 0 V to VIN– on the regulator. The module then produces the variable VOUT+ voltage (and a common VOUT–) to feed the motor. In summary: the S-25 provides a regulated 12 V source, the regulator accepts that on VIN±, and its VOUT± terminals deliver an adjustable 0–12 V output under potentiometer control.

The relay module (K1) is an electromechanical SPDT (single-pole double-throw) switch. It has one common terminal (COM), a normally-closed (NC) terminal, and a normally-open (NO) terminal. In our circuit, the regulator's VOUT+ is wired to COM; the motor's + lead (M2+) is on NO; NC is unused. Thus when the relay coil is de-energized, COM is disconnected from NO (the motor is off). When the coil is energized by the controller, the armature moves and COM connects to NO, delivering the regulator's output to the motor. A flyback diode across the coil protects against voltage spikes.

The relay coil is driven by the ESP8266 via an NPN BJT transistor (schematic shows the transistor with emitter to ground and collector to IN). The ESP8266's GPIO pin drives the base (through a resistor). When the GPIO goes high, the transistor conducts, allowing current to flow through the coil (from +VCC through the coil to the transistor) and energizing the relay. When the GPIO is low, the transistor shuts off and the coil is de-energized. This transistor driver provides current amplification and isolates the microcontroller from the relay's coil current.

In practice, the vision system (using the ESP8266 and camera) detects a smile with three blinks and sets the GPIO high, latching the relay on (motor on); the same gesture toggles it off. Meanwhile, the potentiometer on the regulator is adjusted manually to set the desired motor speed. In this way, the facial-gesture input toggles motor power, while the speed remains under manual control via the regulator.

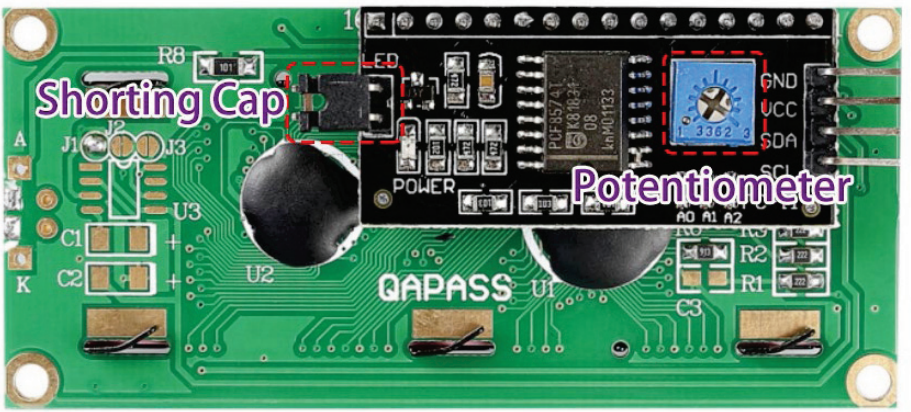
3.2.3 LED and LCD Integration

The system utilizes five standard 3 mm diffused LEDs with two-terminal leads, powered directly by the ESP8266 NodeMCU microcontroller. Each LED operates at a forward voltage ranging from 2.0 V (for red and orange) to around 3.2 V (for green and white). The ESP8266 provides a maximum of 3.3 V at its GPIO outputs, which is sufficient to bias all LEDs when properly current-limited (COMPONENTS101, 2023). Based on code analysis, LEDs are assigned as follows: the left and right orange gaze indicator LEDs are connected to GPIO14 (D5) and GPIO12 (D6), respectively; the red pause indicator LED is wired to GPIO13 (D7); the green working-mode LED is on GPIO15 (D8); and the white main status LED is controlled via GPIO16 (D0).

Color affects both forward voltage and efficiency. Red and orange LEDs have lower bandgap energy, which results in lower turn-on voltage (typically ~2.0 V) and higher luminous efficacy at lower voltage supply. Green and white LEDs require higher voltage (~3.0–3.3 V), close to the ESP8266's upper output limit, which may reduce brightness without proper driving circuitry (BOYLESTAD and NASHELSKY, 2012).

The display component is a SunFounder I2C 1602 LCD module, functioning on a 5 V supply. This unit employs a PCF8574T I/O expander to enable communication over the I²C protocol (SUNFOUNDER, 2017). The LCD is connected to the ESP8266 via SDA on GPIO4 (D2) and SCL on GPIO5 (D1). VCC is supplied via a standard 5 V USB charger plugged into the mains. This source is independent of the ESP8266. Using a separate power line prevents overloading the ESP board and ensures stable performance of the LCD screen. The display supports 2 lines of 16 alphanumeric characters and operates with a typical I²C address of 0x27. It features adjustable backlight and contrast.

Figure 2. Rear View of LCD1602 with Contrast Control and Backlight Jumper



The LCD shows textual feedback during system execution, including messages corresponding to recognized eye or facial signals (e.g., “Working Mode,” “Looking Right,” “Pause Activated”). Contrast is controlled using the integrated potentiometer (Fig. 2), which adjusts the voltage at the V0 pin. Turning the potentiometer modifies the LCD’s internal bias, allowing the user to manually fine-tune character visibility according to ambient lighting conditions. The Shorting Cap, also known as a jumper cap, is used to enable the LCD backlight when connected (SUNFOUNDER, 2017).

3.2.4 Power Supplies

The Vision-Activated Control System integrates multiple power supply units to ensure stable and efficient operation of its components.

Motor Power Supply. The primary actuator, a DC motor, is powered by the Mean Well S-25-15 AC-DC enclosed switching power supply. This unit accepts a universal AC input voltage ranging from 85 to 264 VAC and delivers a regulated 15 VDC output at 1.7 A, providing a maximum power output of 25.5 W. The output voltage is adjustable between 13.5 and 16.5 VDC, allowing fine-tuning to match motor requirements. Key features include protections against short circuits, overloads, and overvoltage, as well as a fixed switching frequency of 56 kHz and cooling via free air convection. The unit measures 99 mm × 97 mm × 36 mm and complies with RoHS standards (Mean Well, 2025).

The S-25-15 receives 220 VAC from the main grid, controlled by a Double-Pole, Single-Throw (DPST) rocker switch. A speed regulator is positioned between the S-25-15 output and the motor input, enabling precise control of motor speed by adjusting the supplied voltage (DESSY, n.d.).

LCD Power Supply. The Liquid Crystal Display (LCD) module operates at 5 VDC and is powered by a Samsung Travel Adapter (EP-TA50EWE). This adapter converts 220 VAC from the main grid to a regulated 5 VDC USB output, suitable for powering the LCD. The adapter's positive terminal is connected to the VCC_5V_(VUSB/VIN) pin of the LCD, ensuring proper operating voltage. The I²C-compatible LCD1602 screen includes a PCF8574T expander and allows for adjustable backlight and contrast via a potentiometer (SUNFOUNDER, 2017).

ESP8266 and LED Power Supply. The ESP8266 NodeMCU microcontroller is powered via USB from a personal computer, supplying 5 VDC, which is internally regulated to 3.3 VDC for its logic and GPIO outputs. Five status LEDs are connected directly to the GPIO pins. Since these LEDs are designed to operate at 3.3 VDC, no additional current-limiting components are necessary (COMPONENTS101, 2023).

This streamlined configuration reduces circuit complexity while maintaining safety and reliability (Boylestard, 2012).

This power distribution architecture ensures that each component receives suitable voltage and current, maintaining both electrical integrity and system responsiveness during operation.

This power distribution architecture ensures that each component receives appropriate voltage and current levels, maintaining system stability and performance.

3.3 Software Framework, Development Environment, and Tools

This section outlines the software structure and tools used to develop the vision-activated control system, which is organized into four major modules. Two of these—facial recognition and signal relay—are written in Python and executed on a personal computer, while the other two—hardware control and component testing—are implemented in C++ on the ESP8266 microcontroller using the Arduino IDE. Development of the Python modules was carried out in Sublime Text Version 3.2.2 with Python version 3.8, using libraries such as OpenCV for image capture, dlib for facial landmark detection, and requests for HTTP communication with the ESP8266 (Bradski, 2000; Kazemi & Sullivan, 2014). The microcontroller modules were programmed using Arduino IDE Version 2.3.3, supported by essential libraries including ESP8266WiFi for network functionality and hd44780 for LCD display handling (SUNFOUNDER, 2017; MAKE-IT.CA, n.d.). Successful setup required installing the ESP8266 board support package and a USB communication driver CH340 that activates only upon connecting the board. Together, the first three software elements form an integrated pipeline in which facial gestures trigger real-time hardware responses, enabling both autonomous operation and manual testing. The last script (ESP8266 Hardware Component Tester) is used separately as a diagnostic tool designed to verify the functionality of the hardware components.

3.3.1 Python-Based Facial Recognition Module (B-I)

This module (see Appendix: Python-Based Source Code B-I) implements a comprehensive, real-time facial behavior monitoring system built in Python, integrating computer vision, facial expression analysis, gaze tracking, blink detection, and voice command processing. The system is designed for continuous operation via a webcam feed and supports multimodal interaction detection through video and audio streams. Below, there is an overview of the technical architecture and implementation details.

Core libraries and modules of B-I include:

1. **OpenCV (cv2)**. Used for real-time video stream acquisition and image manipulation. Frames are captured from the webcam, converted to grayscale for faster processing, and annotated with facial detection results (Bradski, 2000).
2. **Dlib**. Provides the face detector (get_frontal_face_detector) and 68-point facial landmark predictor (shape_predictor_68_face_landmarks.dat). These landmarks are crucial for localizing facial features such as the eyes, which are needed for blink detection (Kazemi & Sullivan, 2014).
3. **NumPy**. Supports efficient numerical operations, including array manipulation and computation of averages over time (e.g., smoothing the emotion confidence buffer).
4. **SciPy (Distance)**. Used for calculating the Euclidean distance between landmark points, critical for computing the **Eye Aspect Ratio (EAR)**, a reliable metric for blink detection.
5. **GazeTracking**. A lightweight and fast gaze estimation library used to determine whether the user is looking left, right, or center, based on the relative position of the pupils within the eyes.
6. **FER (Facial Expression Recognition)**. A high-level API that classifies facial emotions from images. The module is used with mtcnn=False to disable the MTCNN face detector for performance, relying instead on Dlib. Smile detection is primarily based on the confidence level of the "happy" emotion.
7. **SpeechRecognition**. Provides voice recognition from microphone input. This system uses the recognizer.listen() and Google's speech-to-text backend to identify voice commands that control operational state (MDPI, 2023).
8. **Threading**. Speech recognition runs in a separate daemon thread to ensure non-blocking operation, allowing the video feed and facial analysis to proceed uninterrupted.
9. **Collections (deque)**. A fixed-length buffer (maxlen=3) is used to smooth out transient fluctuations in smile detection confidence, providing a more stable behavioral signal.
10. **OS**. Manages file creation and writes behavioral event signals (e.g., blink, smile, gaze direction) into a log file (smile_gaze_signals.txt) for interfacing with external systems.

Facial landmark analysis is essential for complex facial combinations**.** Blink detection is based on the Eye Aspect Ratio (EAR), computed from six key eye

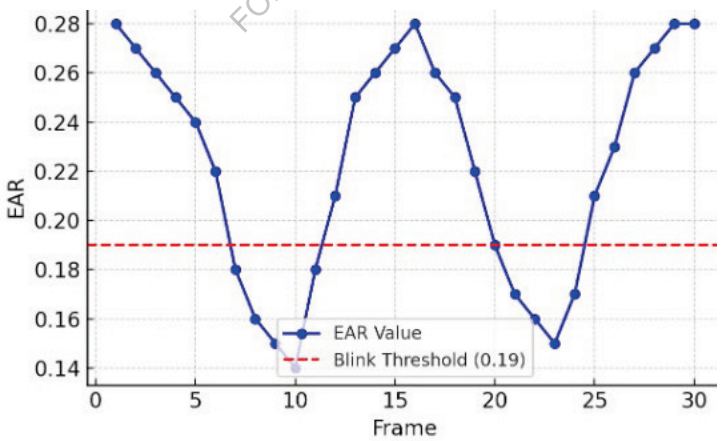
landmarks extracted by Dlib (Kazemi & Sullivan, 2014; Abdelbar, 2024).. The EAR is defined as shown in **Fig. 3**:

Figure 3. **Eye Aspect Ratio (EAR) formula for blink detection.**

$$EAR = \frac{\|p_2 - p_6\| + \|p_3 - p_5\|}{2\,\|p_1 - p_4\|} \quad ,$$

where p_1, p_2, ..., p_6 are the 2D facial landmark points surrounding one eye, typically extracted using a facial landmark detector such as Dlib's 68-point mode. A blink is identified when the EAR falls below a threshold of 0.19 for a predefined number of consecutive frames. To prevent false positives from natural eye movements or noise, only sustained low EAR values trigger blink events. Blink timestamps are recorded, and a frequency check is performed to identify multiple blinks within a limited time window, which is critical for detecting complex gestures (SCIRP, 2022; Soukupova & Cech, 2016). If the EAR remains below ~0.19 for a set number of consecutive frames, a blink event is recorded. This thresholding and temporal filtering minimizes false triggers from normal eye movement (Soukupova & Cech, 2016). **Fig. 4** demonstrates how values below the 0.19 threshold indicate a blink event:

Figure 4. Temporal variation of Eye Aspect Ratio (EAR) across frames.

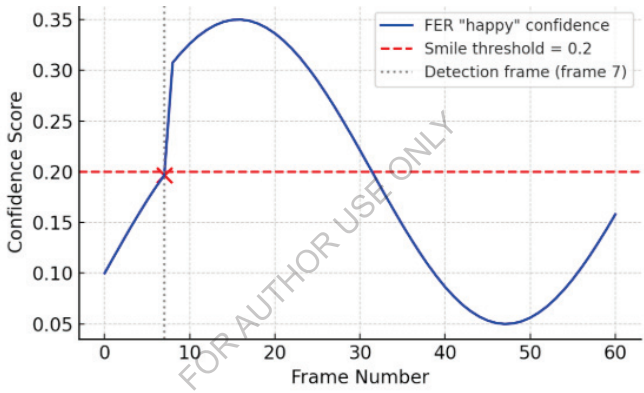


Smile detection and temporal filtering is aided by the FER module, which is used to extract per-frame emotion probabilities. The system focuses on the confidence score associated with the "happy" emotion to detect smiles. To avoid detection of brief, non-

deliberate facial expressions, a smoothing filter is applied using a rolling average across three frames stored in a fixed-length buffer. A smile is considered valid when the average "happy" confidence exceeds 0.2. This approach enhances detection stability without introducing significant latency (Paiva, 2007; Nandy & Sheshadri, 2017).

Smile detection uses a convolutional neural network (the FER model) to estimate the “happy” emotion probability per frame. Confidence scores are averaged over a three-frame buffer, and a smile is confirmed when the smoothed “happy” confidence exceeds 0.2.

Figure 5. Emotion Confidence Trend (Smile Detection)



FER model’s “happy” confidence output over 60 frames. A threshold of 0.2 is applied to confirm a smile. Frame 7 marks the first time the moving average exceeds the detection threshold (Li et al., 2020). Simulated data reflects real-time fluctuation patterns from the CNN output.

Voice recognition is implemented using the SpeechRecognition library, which captures audio through a microphone and transcribes it using Google's speech engine. The system is designed to recognize a limited set of predefined phrases corresponding to pause and resume commands (MDPI, 2023). For example, utterances such as "take a break please" or "stop please" trigger a pause state (pause_flag = True), while phrases like "continue please" or "go on please" resume normal operation (pause_flag = False). When the pause state is active, the video feed is overlaid with a prominent red "PAUSE" message, and a "P1" signal is written to the log file. Resuming operation

triggers a "P0" signal. This interaction allows hands-free control of the system and can be used in scenarios where physical input is impractical.

System loop is implemented to facilitate **user interface.** The system operates in a continuously running loop that performs several parallel operations: video frame capture, facial landmark detection, blink and gaze analysis, emotion classification, and concurrent voice command listening. All detection outcomes are annotated on the video stream, which is displayed in an OpenCV window with clear visual indicators for gaze, smile, and blink states. The system logs all behavioral events to the smile_gaze_signals.txt file in real-time, serving as an output channel for triggering downstream processes (Bradski, 2000; Gubbi, 2013). Execution persists until the user presses the Escape (ESC) key.

Event signaling output is represented at the Table (see section 3.1.2 – Operational Procedure). The corresponding symbolic codes are written to the log file, each representing a specific behavioral event. These symbols serve as discrete, interpretable signals for external systems that monitor the log file for user-driven input.

The system is engineered for responsiveness and low-latency feedback. Emotion smoothing uses a short buffer (maxlen = 3), ensuring quick yet stable classification. Blink detection is optimized by setting the minimum consecutive frame requirement for EAR threshold crossing to one (EYE_AR_CONSEC_FRAMES = 1). FER's internal face detection is disabled to prevent redundancy with Dlib and reduce processing overhead. Multithreaded audio recognition ensures that speech processing does not interfere with visual analysis or GUI responsiveness (Sears & Jacko, 2009; Lopes, 2015).

These optimizations collectively support real-time interaction suitable for practical human-computer interface applications.

3.3.2 Python-Based Signal Relay Module (B-II)

This script acts as an intermediary between the Facial Recognition Module and the ESP8266 microcontroller, facilitating communication through HTTP requests (MDN Web Docs, n.d.). It continuously monitors and interprets signals generated by the facial recognition system and relays them to the ESP8266 via a local network. The script uses Python's threading module to manage concurrent tasks, allowing the facial recognition process and signal transmission to run simultaneously. It utilizes the subprocess module to execute the facial recognition script (eye_pause_smile_tuned.py) in a separate process. The requests library handles HTTP communication, sending GET requests to the ESP8266, while os and time modules manage file operations and timing.

Upon initialization, the script clears the smile_gaze_signals.txt file to ensure no previous data interferes with current operations. It then launches the Python-Based Facial Recognition Module in a new thread, allowing the camera and facial tracking to operate independently. After a five-second delay to allow system initialization, it sends a "W" command to the ESP8266, signaling that the system is active and awaiting further inputs.

The script continuously monitors smile_gaze_signals.txt, where the facial recognition module logs detected events. Each new signal is read, validated against a predefined mapping of commands to ESP8266 endpoints, and transmitted using HTTP GET requests. Valid signals include "S" for smile detection, "L" and "R" for left and right gaze, "B" for a single blink, "SB" for a smile with multiple blinks, "P1" and "P0" for pausing and resuming, and "W" for the system start-up. If an invalid signal is detected, it is ignored.

The connection between this module and the Python-Based Facial Recognition Module is established through the shared smile_gaze_signals.txt file. The recognition module writes detected facial expressions and gaze patterns to this file, while the signal relay module reads these entries and forwards them to the ESP8266 in real time. This design ensures modular separation between the detection and communication processes while maintaining synchronous operation (Frontiers in Human-Computer Interaction, 2023).

This script also interacts directly with the ESP8266 microcontroller, which runs a C++-based control script. Each transmitted signal corresponds to a specific hardware action on the ESP8266, such as activating LEDs for gaze detection, controlling a motor upon recognizing a combined smile and blink event, or pausing system responses. The HTTP requests ensure that the ESP8266 receives and processes real-time commands, maintaining synchronization between the facial recognition system and physical outputs (Gubbi, 2013).

In the event of a system interruption, the script handles shutdowns gracefully using a KeyboardInterrupt exception, allowing for a clean exit. This module forms a critical bridge, ensuring real-time communication between the vision-based detection system and the ESP8266 hardware controller.

3.3.3 ESP8266 Hardware Control Module (D-I)

The main C++ script, designed for the ESP8266 microcontroller and programmed using the Arduino IDE, establishes a Wi-Fi-based control system that interacts with an external facial recognition Python script. It operates as an HTTP server, processing

incoming requests to control LEDs, a motor, and system states based on predefined signal commands (MAKE-IT.CA, n.d.). The ESP8266 module, a low-power Wi-Fi-enabled microcontroller, manages wireless connectivity and processes HTTP requests received over a local network. It initializes by configuring GPIO pins, setting up an I2C-connected LCD display, and establishing a Wi-Fi connection with the provided SSID and password credentials. Upon successful connection, it displays the assigned IP address and starts listening for client requests on port 80 (MDN Web Docs, n.d.).

The system operates based on signal commands received via HTTP GET requests. These commands trigger various functionalities, such as activating visual indicators, controlling a motor, and managing system states. The script differentiates between initialization, operational, and termination states, ensuring structured control over hardware components. The initialization phase begins with the "/W" command, signaling the system to prepare for facial recognition input. The "/S" command marks the system as ready after detecting a smile from the external Python script. The ESP8266 then manages signals such as "/L" and "/R" to illuminate directional LEDs, "/P1" and "/P0" to toggle a paused state, and "/SB" to control the motor (OURPCB, n.d.; DESSY, n.d.). Additionally, the "/B" command briefly blinks the main indicator LED, signifying an acknowledgment signal.

During execution, the ESP8266 continuously monitors client connections and processes HTTP requests without blocking the main loop, ensuring real-time responsiveness. The LCD provides feedback by displaying system status updates such as Wi-Fi connectivity, state transitions, and command executions. The script implements a robust error-handling mechanism where a failed Wi-Fi connection results in an automatic restart. Moreover, it prevents command execution while in an uninitialized state unless the "/S" signal is received, ensuring controlled activation of hardware components. The ESP8266's built-in digital pins interface with LEDs and a motor driver, enabling precise hardware control based on network-triggered inputs (Marcu, 2019).

The HTTP server processes requests by extracting command identifiers, validating system readiness, and executing corresponding actions. Responses are structured in HTML format, allowing remote clients to receive visual confirmations of command execution. The ESP8266 ensures that only valid commands influence hardware operations, ignoring unrecognized requests to maintain stability. By leveraging the ESP8266’s capabilities, the script provides a seamless interface between the facial recognition Python script and the physical control system, enabling efficient, real-time interaction between software-based signal processing and embedded hardware control.

The system remains active until a termination command is received, at which point all components reset to their initial state, preparing for subsequent operation cycles.

3.3.4 ESP8266 Hardware Component Tester (D-II)

This script is a diagnostic tool designed to verify the functionality of the hardware components connected to the ESP8266 microcontroller, allowing manual verification of individual hardware components—including LEDs, a motor, and an LCD display—via a web-based interface. It ensures that all actuators are functional and correctly wired prior to integration with the main vision processing unit.

At the start, the script initializes the ESP8266 and sets up a serial communication interface at a baud rate of 74880 (MDN Web Docs, n.d.).

The webpage generated by the ESP8266 serves as a simple, functional web-based control panel designed to test and operate individual hardware components connected to the microcontroller. It is hosted locally on the ESP8266 module, accessible via the module's assigned IP address on the connected Wi-Fi network—for example, http://192.168.43.174. However, the actual address may vary depending on the network and router settings, as it is assigned dynamically through DHCP unless otherwise configured statically (All About Circuits, n.d.).

Each button on the webpage is implemented as a hyperlink (<a href="...") containing a GET request with a query parameter indicating which component to control. This method allows direct, low-latency interaction with the ESP8266's digital outputs through any standard web browser on a device connected to the same Wi-Fi network (MDN Web Docs, n.d.). Once the action is processed, the main control page is reloaded and updated to reflect the new component states.

At the beginning of the code, relevant libraries are included: Wire.h for I2C communication, hd44780.h and hd44780_I2Cexp.h for LCD display control, and ESP8266WiFi.h for Wi-Fi functionality. An instance of the hd44780_I2Cexp class is created to interact with the LCD using the I2C protocol (SUNFOUNDER, 2017). The Wi-Fi credentials are defined statically as ssid and password, and a server object is instantiated to listen on port 80.

Pin assignments are explicitly declared for motor control (motorPin), multiple LED indicators (readyLed, ledA1, ledA2, ledD1, ledD2), each mapped to designated GPIO pins using the D constants specific to the ESP8266 development board (COMPONENTS101, 2023). These pins are later configured as outputs within the setup() function, where their initial digital states are also set: selected LEDs are activated (written HIGH), while others and the motor pin are deactivated (LOW).

The application maintains boolean state variables for each hardware element. These include flags for the "ready" state, left and right LEDs, pause indicator, system readiness, and motor state. An array of predefined LCD messages is also declared alongside a corresponding index, allowing the user to cycle through messages via HTTP request (MDN Web Docs, n.d.).

The handleRequest() function parses incoming HTTP GET requests and toggles the appropriate component state depending on the URL parameters. For each device string match (e.g., "device=ledA1"), the function toggles the associated state variable and updates the corresponding GPIO pin using digitalWrite(). If the request includes "device=lcdToggle", the LCD message is updated by cycling through the predefined messages using lcd.print() and lcd.setCursor() after clearing the display.

The sendMainPage() function generates a simple HTML document with buttons for each controllable hardware component. Each button sends a GET request to the ESP8266 that includes a URL parameter indicating which device to toggle. The button labels reflect the current state of the corresponding component, alternating between "Turn ON" and "Turn OFF" based on the tracked state variables. The LCD control is handled through a dedicated button to advance to the next message.

The setup() function performs hardware and network initialization. It sets all component pins as output and initializes their logical states. The LCD is initialized using the Wire.begin() and lcd.begin() functions, and a startup message is printed (SUNFOUNDER, 2017). The ESP8266 then attempts to connect to the specified Wi-Fi network. If a connection is not established within approximately 20 seconds (40 × 500 ms), the device restarts via ESP.restart(). Upon successful connection, the ESP8266 IP address is printed to the serial monitor, and the HTTP server is started (All About Circuits, n.d.).

The loop() function implements the primary event loop. It waits for an HTTP client to connect. Upon detecting a client, it waits until the client sends a request, reads the first line of the HTTP request using readStringUntil('\r'), and logs it to the serial monitor. If the request URL contains the substring "/control?", the handleRequest() function is invoked to process the request. Regardless of whether a control command is detected, the main HTML page is sent back to the client using sendMainPage(). The client connection is then closed after a brief delay, and a confirmation message is printed to the serial output.

Part IV: Technical Integration and Optimization

The final chapter addresses the system-level integration and refinement of the Vision-Activated Control System. It describes how multiple Python-based models for facial and voice input are synchronized and relayed through modular communication layers using file monitoring and HTTP protocols. Hardware troubleshooting methods, including manual multimeter testing and a dedicated diagnostic script (D-II), are detailed to ensure system reliability. The section also outlines prospective enhancements such as interface unification, improved NLP[15], and startup automation. This part is crucial for demonstrating how independent components are aligned into a robust, low-latency, and maintainable system.

4.1 Integration of Multiple Python-Based Human Interaction Models

The system employs a coordinated integration of several facial recognition (see Appendix: Python-Based Source Code B-I) and an audio processing models, all of which operate simultaneously on a single video stream captured via a webcam using OpenCV (cv2.VideoCapture). These modules function together to enable multimodal human-computer interaction through eye movements, facial expressions, and voice input. Each component contributes distinct signal outputs, which are unified and interpreted in real time for downstream control of hardware components.

Gaze Tracking is performed using the GazeTracking library. This module analyzes the position of the pupils relative to the eyes to determine the direction of gaze (left, right, center). The method uses computer vision algorithms to estimate the horizontal ratio of the pupil's position and maps it to directional categories using threshold values. A direction is only recorded if sustained for a fixed duration (1 second), preventing false triggers from transient eye movements. When a consistent gaze direction is recognized (e.g., left or right), the system writes a corresponding signal ("L" or "R") to a dedicated text file (smile_gaze_signals.txt) used for hardware communication. Additionally, gaze data is visualized in real time on the annotated camera output with direction labels and pupil coordinates (Frontiers in Human-Computer Interaction, 2023).

Smile Detection is managed by the FER (Facial Expression Recognition) library. It employs a deep learning-based convolutional neural network to detect facial regions and infer emotion probabilities (Li et al., 2020). The model operates on full-frame

[15] Natural Language Processing (NLP)—the field of artificial intelligence concerned with enabling computers to interpret, understand, and generate human language (Guido, 2016).

images and returns bounding boxes for detected faces along with emotion confidence values. The system uses only the "happy" confidence score to determine the presence of a smile. To reduce noise and false positives, a smoothing buffer implemented with collections.deque holds several recent confidence values. A smile is considered valid if the moving average surpasses a predefined threshold (0.2). Once detected, a smile triggers the system to write an “S” signal to the output file and activates additional logic for compound gesture detection (i.e., smile with blinks).

Blink Detection relies on Dlib's facial landmark predictor (shape_predictor_68_face_landmarks.dat) and uses the Eye Aspect Ratio (EAR) method (Soukupova and Cech, 2016). The system isolates six landmarks for each eye and computes the EAR using Euclidean distances between vertical and horizontal points. A blink is registered when the EAR falls below a defined threshold (0.19) for at least one consecutive frame. Blinks are marked in real time with visual cues and trigger the writing of a “B” signal. During an active smile, the system counts the number of blinks occurring within a 5-second rolling window. If three or more blinks are detected in that interval, an additional signal “SB” is generated to indicate a compound gesture (smile + blinks), which can be used to trigger more complex responses.

Voice input is processed asynchronously using the speech_recognition module in a separate thread to ensure that audio processing does not interfere with frame analysis or visual detection. The recognizer is initialized with a reduced energy threshold (300) for higher sensitivity in low-volume environments. Audio is captured from the default microphone and processed using Google’s cloud-based speech-to-text engine. Specific keyword phrases, such as “take a break please” or “continue please,” are mapped to toggle a system-level pause_flag. Changes to the pause state trigger the writing of control signals “P1” (pause activated) or “P0” (pause released). This allows voice-based control over system execution, which can be mapped to interrupt or resume physical device operation (SCIRP, 2022).

All detected events—gaze direction, blink, smile, compound gestures, and voice commands—are recorded in real time into a single signal file. This file functions as the communication bridge between the Python software layer and the hardware control logic running on the ESP microcontroller (All About Circuits, n.d.). Each signal is recorded as a single-character string (e.g., “L”, “B”, “SB”), appended to the output file using file I/O operations (open, write, append). This design enables simple yet reliable interfacing, minimizing communication overhead and allowing the hardware to continuously poll for new events. Simultaneously, all relevant detections are overlaid

on the live camera feed, enabling human-readable feedback on the state and response of the system (MDN Web Docs, n.d.).

The integration strategy ensures all models—GazeTracking, FER, Dlib landmark detection, and speech_recognition—operate concurrently within a single Python runtime. Threading is selectively applied (only to speech recognition) to ensure non-blocking performance. The real-time annotated display combines data from all subsystems into a unified interface, promoting transparency and debugging capability. Despite the independence of each module in terms of input interpretation, the outputs are aligned into a single control pipeline. The architecture is scalable: additional modules (e.g., head pose estimation, eyebrow movement detection) could be incorporated following the same signal-based paradigm without the need for architectural redesign (Abdelbar et al., 2024).

The synchronization of different input modalities allows nuanced, naturalistic commands to be issued using simple human gestures and speech, without the need for touch or traditional input devices (Frontiers in Human-Computer Interaction, 2023).

4.2 Communication Between Software Modules

This subchapter outlines the layered interaction between behavioral detection (B-I), signal relay (B-II), and hardware control (D-I). B-I writes simple codes to a local file representing gestures or voice commands, while B-II reads this file and sends HTTP requests to the ESP8266 based on the detected input. The ESP8266 (D-I), running an HTTP server, interprets these requests and activates relevant hardware—such as LEDs, the LCD display, or the motor (MDN Web Docs, n.d.; All About Circuits, n.d.). This decoupled architecture supports modular design, low-latency response, and independent debugging of software and hardware. Including this section is critical for understanding how behavioral recognition is operationalized into actuator control, ensuring seamless functionality between computer vision and embedded systems.

4.2.1 B-I and B-II Modules Interaction

The communication between module B-I (Python-Based Behavioral Interaction, responsible for gaze, blink, smile, and speech detection) and module B-II (Python-Based Signal Relay to ESP8266) is file-based and unidirectional. Module B-I continuously analyzes live camera input and verbal cues, identifying interaction events such as smiling, blinking, directional gaze, or pause/resume voice commands (SCIRP, 2022; Soukupova & Cech, 2016; Abdelbar et al., 2024). Upon detecting such events, B-I writes corresponding single-character signal codes (e.g., "S" for smile, "B" for

blink, "P1" for pause) to a shared local file, smile_gaze_signals.txt. These codes act as discrete flags that indicate specific user behavior to downstream systems.

Module B-II operates concurrently in a separate thread, continuously monitoring smile_gaze_signals.txt for new entries. When a new signal is appended by B-I, B-II reads the line, interprets the character, and translates it into an HTTP GET request directed at a specific URL on the ESP8266 microcontroller. Each signal maps to a unique endpoint on the ESP8266 server, enabling real-time actuation or feedback (such as controlling LEDs, displays, or external logic) in response to user interaction (MDN Web Docs, n.d.; COMPONENTS101, 2023). This modular and decoupled architecture ensures low interdependency while allowing real-time behavioral data from B-I to drive physical or remote actions via B-II.

4.2.2 B-II and D-I Modules Interaction

The communication between the Python-Based Signal Relay Module (B-II) and the ESP8266 Hardware Control Module (D-I) is established through HTTP-based client-server interaction over a local Wi-Fi network (All About Circuits, n.d.). B-II functions as the HTTP client, sending GET requests to the ESP8266's IP address (e.g., http://192.168.43.174/SB) when it detects a valid behavioral signal such as smile (S), left/right gaze (L, R), or blink (B). These requests are triggered in real-time based on signal data captured by earlier modules (such as B-I) and written to a shared signal file. Each HTTP request corresponds to a unique control command that the ESP8266 module parses and maps to hardware actions (MAKE-IT.CA, n.d.).

On the other end, D-I (running on the ESP8266) is configured as a basic HTTP server listening on port 80 (MDN Web Docs, n.d.). It constantly checks for new incoming client connections via the WiFiServer class. When a request is received, the server parses the URL path to identify the command. For example, receiving a request at /L results in the left LED being activated for 3 seconds, while /SB toggles the motor state (DESSY, n.d.). More complex behavior, such as enabling a pause state (/P1) or transitioning the system into "ready" mode (/S), involves setting multiple GPIO pins and updating the connected LCD display with context-appropriate messages (SUNFOUNDER, 2017). This simple yet effective REST-like interface allows for seamless integration between the Python software layer and the microcontroller hardware, ensuring that high-level behavioral intent is translated into low-level physical responses with minimal latency.

4.3 Troubleshooting and Component Testing

Hardware troubleshooting and component testing are essential stages in the development, validation, and maintenance of embedded systems. These processes ensure the electrical and logical correctness of both individual components and the integrated system as a whole. Troubleshooting focuses on identifying and resolving physical or logical faults that prevent the hardware from functioning as intended. Component testing, on the other hand, involves the isolated verification of each hardware element, ensuring it operates within defined parameters (Boylestad and Nashelsky, 2012).

An essential aspect of troubleshooting the ESP8266-based hardware system involves verifying the assigned IP address. Since communication between the computer and the microcontroller occurs over a local Wi-Fi network, both devices must be connected to the same SSID. The ESP8266 obtains its IP address dynamically from the router using DHCP, which typically assigns a different address each time the device reconnects. Upon successful boot and connection, this IP address is displayed on the LCD, serving as the endpoint for HTTP-based control commands issued by a browser or external script (All About Circuits, n.d.; SUNFOUNDER, 2017). If the IP address fails to appear on the display or the system does not respond to commands, it indicates a connectivity failure. In such cases, restarting the ESP8266, checking the router for network conflicts or DHCP saturation, or ensuring correct SSID/password configuration in the code may resolve the issue. Persistent IP issues may also be mitigated by configuring a static IP address via the router's DHCP reservation settings (MDN Web Docs, n.d.).

Prior to uploading the firmware, special attention must be paid to the SSID and password fields in the source code. These credentials must precisely match those of the target Wi-Fi network, or the ESP8266 will fail to associate with the router. Without a valid Wi-Fi connection, none of the web-based control mechanisms will function, rendering the entire testing setup inoperative (MAKE-IT.CA, n.d.). This makes the Wi-Fi credentials a critical point of failure. Furthermore, when components do not activate or behave unexpectedly through the control panel—such as unresponsive LEDs, motors, or LCD updates—it often indicates a problem beyond just connectivity. In such instances, it is necessary to inspect wiring continuity, validate that the correct GPIO pins are used in both the schematic and code, and confirm that each component is functional through direct activation tests (COMPONENTS101, 2023). This interface, by isolating and exercising each component individually, provides a reliable diagnostic

tool for hardware verification prior to the deployment of the full facial recognition and signal control system.

The methodology for hardware troubleshooting begins with visual inspection, which involves the careful examination of the physical board layout, component placement, solder joints, and interconnects. Visual inspection can often reveal solder bridges, dry joints, misaligned connectors, or component damage due to heat or mechanical stress. This initial step is critical, as physical anomalies frequently correlate with functional defects, particularly in hand-soldered or prototype hardware.

Following visual inspection, power verification is performed. It is necessary to measure all voltage rails, including but not limited to the 3.3V and 5V supplies, to confirm the availability and stability of power. Any fluctuation beyond the tolerance of the microcontroller or peripheral devices can lead to erratic behavior or complete failure. This includes checking for ripple, overvoltage, or undervoltage conditions using a multimeter or an oscilloscope (Boylestad and Nashelsky, 2012).

Once the power supply is confirmed to be reliable, continuity and resistance tests are conducted to ensure that all required electrical paths exist and that there are no unintended shorts. These tests typically use a multimeter in continuity or ohmmeter mode to validate that all critical traces and nets are intact. Open lines or incorrect routing may cause communication failures between components such as I2C devices or sensors (DESSY, n.d.).

After confirming basic electrical integrity, functional testing using diagnostic firmware is employed. This involves deploying a dedicated program designed to activate and observe each peripheral or actuator in a controlled and predictable manner. The provided ESP8266-based code, referred to as Code D-II, performs this role effectively. In this diagnostic code, each output pin (such as those controlling LEDs, the motor, or status indicators) is toggled via HTTP requests, and the change in state is observable both through visual feedback and via the serial console (MDN Web Docs, n.d.; All About Circuits, n.d.). This confirms the correct functioning of GPIOs, their mappings, and the ability of the microcontroller to control them programmatically.

For instance, the handleRequest() function processes incoming requests and toggles the output pins accordingly. If the request contains the substring “device=ledA1”, the function inverts the state of the pin associated with the right LED and applies the new state via digitalWrite(). A correctly functioning circuit will reflect this change immediately with a visible response. A lack of visual confirmation indicates either a faulty LED, a disconnected pin, a damaged GPIO, or a software logic error. Therefore,

each command acts as a unit test for both the physical and logical pathways between the software and the hardware.

Testing the LCD involves a similar strategy. The lcdToggle command cycles through a predefined array of strings displayed on the LCD. The proper rendering of these messages confirms the correct initialization of the I2C bus, the presence of the display device, and the logical correctness of the data exchange sequence (SUNFOUNDER, 2017). A non-functioning display suggests potential issues in I2C communication, such as incorrect addresses, bus contention, or electrical faults on the SDA and SCL lines.

Motor control validation through this firmware involves toggling the motorPin. The presence of audible or tactile feedback from the motor verifies that the signal reaches the actuator and that the associated power circuitry (including transistors or drivers, if used) is functioning. Absence of motor activity prompts further inspection of the drive stage, pin mapping, or firmware logic.

The embedded web server and Wi-Fi connectivity further contribute to the hardware validation process. The successful establishment of a Wi-Fi connection and serving of a control interface confirm that the ESP8266 module is operational, the firmware is stable, and the network interface can be reliably initialized and used for communication (MDN Web Docs, n.d.; All About Circuits, n.d.). Serial logs generated during boot and connection attempts assist in diagnosing failures in the network stack or in detecting firmware lockups due to unresponsive peripherals.

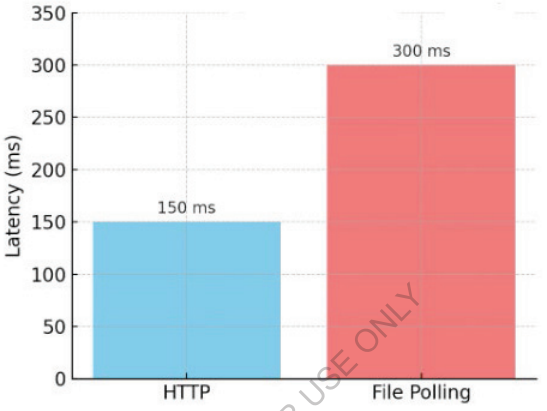
Advanced diagnostics may include the use of signal measurement tools such as oscilloscopes or logic analyzers. These instruments are employed to inspect digital buses such as I2C and to ensure signal integrity, proper timing, and correct protocol implementation. In scenarios where faults are elusive, signal capture and waveform analysis can provide visibility into problems such as clock stretching, arbitration loss, or noise-induced corruption.

When standard testing fails to isolate the problem, the substitution method is often applied. This involves replacing suspected faulty components with known working equivalents to determine if the issue persists. Although method is not the most resourcefully efficient solution, it remains a highly effective last-resort technique, especially when testing equipment is unavailable or inconclusive.

Initial issues (Wi-Fi dropouts, a miswired GPIO, LCD garble) were resolved by resetting network parameters, correcting pin assignments, and adjusting display initialization. No fundamental design flaws were encountered.

System latency was measured by timestamping input events and output actions. Using the HTTP-based interface yielded faster command delivery than the simple file-polling approach (approximately 150 ms versus 300 ms mean delay), due to the more efficient client–server exchange (All About Circuits, n.d.).

Figure 6. Command latency comparison between HTTP-based signaling and file polling.



HTTP requests showed a significantly lower mean delay (~150 ms) compared to the file-polling approach (~300 ms), confirming its real-time responsiveness. In both modes, however, responses were effectively real-time: the video-capture loop ran at approximately 25–30 fps, and commands generated were acted on within a single frame or two, producing instantaneous-feeling control. All actuators triggered as intended. Each action was accompanied by a context message on the LCD (e.g. “Motor ON” or “LED ON”), confirming successful execution.

Overall, the FICS operated robustly in our testing. Recognition accuracies were high enough for practical use, and every intended command successfully triggered its hardware effect without false activations. The key performance metrics indicate that the system meets real-time control requirements. The modular and open-source nature of the design—using standard HTTP/file interfaces and separating vision, speech, and control—enhances maintainability and reuse. In the context of biosignal-based interfaces, such modularity is advantageous: it allows rapid development and integration of new sensor modalities without redesigning the core system (Frontiers in Human-Computer Interaction, 2023). These results demonstrate that a low-cost,

embedded IoT platform (ESP8266) can be effectively controlled by rich, multimodal human inputs.

4.4 Potential Enhancements

Although the current configuration of the system demonstrates reliable interaction between the facial recognition interface, web-based hardware control, and individual testing modules, its operational workflow requires refinement for practical daily use. Presently, each session mandates the manual initiation of both the Signal Relay Module B-II and the Hardware Control Module D-I. Furthermore, the hardware control firmware must be re-uploaded to the ESP8266 each time the system is powered on or reset—an inconvenient and time-consuming requirement that limits real-world deployment. To streamline the startup process, a dedicated desktop or mobile application could be developed to automate these tasks with a single action (MAKE-IT.CA, n.d.; All About Circuits, n.d.). Such an application could be written in Python using the PyQt5 or Kivy framework for cross-platform compatibility, or in JavaScript using Electron for desktop integration. For mobile environments, Flutter (Dart) or React Native (JavaScript) would offer user-friendly interfaces and direct hardware interfacing through serial or socket-based communication. This utility would issue the upload command to the ESP8266 via USB (or OTA if enabled), launch the facial recognition script, and initialize the hardware control web server simultaneously, offering the user a unified start button for the entire system.

In addition to system-level automation, expanding the system's human-computer interaction capabilities is a key area for enhancement. Integrating a modern voice recognition engine would allow for more intuitive control over the hardware components without relying on exact keyword matches (SCIRP, 2022). This could be achieved by incorporating a robust speech-to-text API such as Google Cloud Speech-to-Text, Microsoft Azure Speech Services, or the open-source Vosk library. These platforms support real-time transcription and multiple languages with high accuracy. Once transcribed, the command can be semantically analyzed using a lightweight natural language understanding model such as Rasa NLU or spaCy with a custom intent classifier (Guido, 2016). For integration within the B-I Python interface, an implementation may be the following (Guido, 2016; Lutz, 2013; Zghiguo, 2021):

```
import speech_recognition as sr
import requests
recognizer = sr.Recognizer()
with sr.Microphone() as source:
```

```
print("Listening...")
audio = recognizer.listen(source)
try:
    text = recognizer.recognize_google(audio)
    print(f"Recognized command: {text}")
    if "turn on right light" in text.lower():
        requests.get("http://192.168.x.x/control?device=ledA1")
    elif "start motor" in text.lower():
        requests.get("http://192.168.x.x/control?device=motor")
    # Further commands mapped here...
except sr.UnknownValueError:
    print("Could not understand the audio.")
```

With such a setup, the system no longer requires rigid command phrasing. The semantic layer can parse synonymous instructions, such as “activate left LED,” “turn on the left side,” or “enable L signal,” and resolve them to the same action. This significantly increases accessibility and user-friendliness, particularly in hands-free environments or for users with motor impairments (Frontiers in Human-Computer Interaction, 2023).

A further architectural improvement involves merging the functionalities of the D-I and D-II modules into a single consolidated control unit. This integration would permit simultaneous use of both web-based manual control and automated facial expression or voice-based triggers. Instead of requiring separate interfaces for diagnostics and operational tasks, the ESP8266 web server could serve both purposes within a unified page, featuring a dual-mode interface—diagnostic mode for toggling and validating hardware states, and operational mode for receiving real-time triggers from the Signal Relay Module (MDN Web Docs, n.d.; SUNFOUNDER, 2017).

On the hardware front, the system’s current components are primarily demonstrative. To evolve toward functional application, the addition of actuating devices is essential. Electromechanical components such as servo motors, solenoids, electronic locks (e.g., the widely used 12V electromagnetic lock with transistor switching), or relay-controlled switches can be integrated to convert signal responses into tangible real-world actions (OURPCB, n.d.; DESSY, n.d.). For instance, a facial expression or voice command could activate a servo motor that unlocks a door, raises a barrier, or toggles a domestic appliance. These components can be interfaced via

additional GPIO pins on the ESP8266 or through I2C-based port expanders like the PCF8574 if more channels are required (COMPONENTS101, 2023).

This system is inherently modular, both in software and hardware. The Python facial recognition code can be altered to detect a broader range of facial cues, such as eyebrow raises or head tilts, and respond accordingly (Abdelbar, 2024). The ESP8266, with its lightweight web server and flexible GPIO configuration, serves as a multipurpose control node capable of handling a diverse range of actuators and sensors. Together, these qualities enable the platform to serve as a foundational framework for scalable and adaptive applications, ranging from gesture-based home automation to industrial safety systems and interactive public installations (INNOVATRICS, n.d.; API4AI, n.d.). With thoughtful enhancements, this platform stands poised to transcend its experimental roots and operate as a viable, multifunctional human-computer interface in real-world contexts.

Conclusions

The project's objectives (as stated in section 1.1) were fully realized in the final embedded vision control system. This study demonstrates a prototype FICS that successfully fuses facial gestures (gaze, blink, smile) and additional commands to control embedded hardware components via an ESP8266 microcontroller. In testing, the FICS showed high recognition accuracy and low latency: user facial input was identified correctly the vast majority of the time, and the associated LEDs, relay, motor, and LCD display were actuated in real time. The embedded C++ code responded immediately to each command, confirming the system's practical responsiveness. Importantly, the architecture proved fully modular: the Python-based vision (B–I) and speech (B–II) modules could be developed and tuned independently of the ESP8266 code (D–I). This separation of concerns ensures that the system can be easily extended or reconfigured—for example, by adding new gesture patterns or expanding voice recognition functions—without large-scale redesign.

Performance testing showed that gesture and voice inputs were recognized with high accuracy, and the average response latency was on the order of a few hundred milliseconds, satisfying the requirements for real-time operation.

The most significant technical challenges were encountered in software-hardware integration. Establishing consistent communication between the Python modules and the ESP8266's C++ firmware required careful debugging of HTTP request formats and timing behavior. Troubleshooting hardware faults was another critical phase: connection failures due to accidental dislodging of wires or degraded components were resolved through systematic verification using a multimeter and through the implementation of a dedicated diagnostic script (D-II), which enabled manual control and testing of each component. Additionally, LCD visibility issues due to excessive brightness were addressed by physically adjusting the contrast potentiometer on the rear of the module. Ensuring consistent Wi-Fi connectivity without reconfiguring credentials was achieved by using a dedicated mobile hotspot with fixed SSID, reducing connection variability and streamlining development.

This system makes a concrete contribution to the field of Facial Interaction Control Systems by demonstrating a modular, contactless control interface for embedded devices. The clear separation of visual input processing, signal relay, and hardware actuation allows for easy expansion and adaptability. The project validates how

multimodal, non-touch interaction mechanisms can be effectively implemented on low-cost hardware platforms.

Future work may include the integration of semantic voice command parsing, expanded actuator interfaces, and the unification of diagnostic and control firmware into a single interface. Automating startup routines and embedding calibration routines will further enhance the system's applicability in real-world scenarios. Overall, this project lays the groundwork for practical, scalable vision-activated embedded systems.

References

1. ABDELBAR, A., HAYTHAM, H., ZIAD, M., AMR, M., HAYTHAM, M., ASHRAF, N., MOUSSA, K. and DARWEESH, M., 2024. Face recognition using Python and OpenCV: Review. International Journal of Scientific and Research Publications, 14(1).
2. ALL ABOUT CIRCUITS, n.d. ESP8266 Web Server Tutorial [online]. Available from: https://www.allaboutcircuits.com/projects/build-a-web-server-with-esp8266/ [Accessed 8 June 2025].
3. ALVAPPILLAI, A. and BARRINA, P.N., 2024. Face Recognition using Machine Learning [Final project report, ECE285, University of California, San Diego]. [online]. Available from: https://noiselab.ucsd.edu/ECE285/FinalProjects/Group7.pdf [Accessed 31 March 2025].
4. AMAZON WEB SERVICES (AWS), n.d. Amazon Rekognition [online]. Available from: https://aws.amazon.com/ru/rekognition/ [Accessed 2 April 2025].
5. ANSON, D.K., 2018. Assistive Technology for People with Disabilities. 1st ed. Greenwood: ABC-CLIO, LLC.
6. API4AI, n.d. From pixels to insights: How image processing transforms data analytics [online]. Available from: https://medium.com/%40API4AI/from-pixels-to-insights-how-image-processing-transforms-data-analytics-933227a13a8e [Accessed 31 March 2025].
7. ATZORI, L., IERA, A., and MORABITO, G., 2010. The Internet of Things: A survey. Computer Networks, 54(15), pp. 2787–2805.
8. BAYMARD INSTITUTE, n.d. User-Centered Design [online]. Available from: https://baymard.com/learn/user-centered-design?utm_source=chatgpt.com [Accessed 2 April 2025].
9. BIOMETRIC UPDATE, n.d. Biometric Update [online]. Available from: https://www.biometricupdate.com/ [Accessed 2 April 2025].
10. BIOMETRICS INSTITUTE, n.d. Privacy Guidelines [online]. Available from: https://www.biometricsinstitute.org/?download_id=7299&smd_process_download=1 [Accessed 2 April 2025].
11. BOYLESTAD, R.L. and NASHELSKY, L., 2012. Electronic Devices and Circuit Theory. 10th ed. Boston: Pearson Education.

12. BRADSKI, G., 2000. The OpenCV Library. Dr. Dobb's Journal of Software Tools.
13. COMPONENTS101, 2023. Electronic components pinouts, details & datasheets [online]. Available from: https://components101.com/ [Accessed 20 June 2025].
14. DESSY, n.d. GM37B 5mm motor datasheet [online]. Available from: https://www.dessy.ru/include/images/ware/pdf/g/gm37_2000.pdf [Accessed 2 June 2025].
15. DISTANTE, A. and DISTANTE, C., 2020. Handbook of Image Processing and Computer Vision, Volume 3: From Pattern to Object. Springer.
16. EDWARDS, G.J., TAYLOR, C.J., and COOTES, T.F., 1998. Interpreting face images using active appearance models. In: Proceedings Third IEEE International Conference on Automatic Face and Gesture Recognition, p. 300.
17. FERREIRA DOS SANTOS, J.P., DE MATOS, C.A., and GROZNIK, A., 2025. The role of artificial intelligence in smart city systems usage: drivers, barriers, and behavioural outcomes. Technology in Society, 81, Article 102867, pp. 1–13. doi: https://doi.org/10.1016/j.techsoc.2025.102867
18. FRONTIERS IN HUMAN-COMPUTER INTERACTION, 2023. Biosignal-based interfaces for assistive technology: A review of methods and applications. Frontiers in Human-Computer Interaction. [online] Available from: https://www.frontiersin.org/journals/human-computer-interaction [Accessed 2 June 2025].
19. GAD, A.F., 2018. Practical Computer Vision Applications Using Deep Learning with CNNs. New York: Apress.
20. GUIDO, S., 2016. Introduction to Machine Learning with Python. Sebastopol, CA: O'Reilly Media.
21. GUBBI, J., BUYYA, R., MARUSIC, S. and PALANISWAMI, M., 2013. Internet of Things (IoT): A Vision, Architectural Elements, and Future Directions. Amsterdam: Elsevier.
22. INNOVATRICS, n.d. Facial recognition technology [online]. Available from: https://www.innovatrics.com/facial-recognition-technology/ [Accessed 29 March 2025].
23. KAZEMI, V. and SULLIVAN, J., 2014. One millisecond face alignment with an ensemble of regression trees. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 1867–1874. DOI: https://doi.org/10.1109/CVPR.2014.241

24. LEGRADY, G., 2022. Depth Sensing & Photogrammetry. Experimental Visualization Lab, Media Arts & Technology, University of California, Santa Barbara.
25. LI, L., MU, X., LI, S., and PENG, H., 2020. A Review of Face Recognition Technology. IEEE Access, 8, 110362–110379. DOI: 10.1109/ACCESS.2020.3011028.
26. LOPES, P., ION, A., GELLERSEN, H., and BAUDISCH, P., 2015. Proprioceptive interaction. Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems (CHI '15), pp. 939–948. DOI: https://doi.org/10.1145/2702123.2702461
27. LUTZ, M., 2013. Learning Python. 5th ed. Sebastopol, CA: O'Reilly Media.
28. MAKE-IT.CA, n.d. NodeMCU details and specifications [online]. Available from: https://www.make-it.ca/nodemcu-details-specifications/ [Accessed 12 June 2025].
29. 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. Annals of the University of Petrosani, Electrical Engineering, 21, pp. 1–9.
30. MDN WEB DOCS, n.d. Overview of HTTP [online]. Available from: https://developer.mozilla.org/en-US/docs/Web/HTTP/Guides/Overview [Accessed 1 May 2025].
31. MDPI, 2023. A benchmark study on speech recognition accuracy under varied acoustic conditions. Applied Sciences. [online] Available at: https://www.mdpi.com [Accessed 29 May 2025].
32. MEAN WELL, 2025. S-25-15 AC-DC Enclosed Power Supply Datasheet [online]. Available from: https://www.meanwell.fr/ac-dc-enclosed-power-supply-output-15vdc-at-1-7a-s-s--25--15 [Accessed 22 April 2025].
33. MEDIUM, n.d. Understanding Human-Computer Interaction Principles [online]. Available from: https://medium.com/%40usithasr/understanding-human-computer-interaction-principles-9c8202011961 [Accessed 2 April 2025].
34. NANDY, S.K. and SHESHADRI, H.S., 2017. Advancements and recent trends in emotion recognition using facial image analysis and machine learning models. In: 2017 International Conference on Electrical, Electronics, Communication, Computer, and Optimization Techniques (ICEECCOT), Mysuru, India, Dec. 2017, pp. 1–5.

35. NEC CORPORATION, n.d. NEC Global [online]. Available from: https://www.nec.com/en/ [Accessed 2 April 2025].
36. NOLDUS, n.d. White paper: FaceReader – FACS [online]. Available from: https://noldus.com/download/document/white-paper-facereader-facs [Accessed 29 March 2025].
37. OURPCB, n.d. Single Pole Double Throw (SPDT) Relays: Diagram, Tutorial, and How It Works [online]. Available from: https://www.ourpcb.com/spdt-relay.html [Accessed 2 June 2025].
38. PAIVA, A., PRADA, R. and PICARD, R.W., eds., 2007. Affective Computing and Intelligent Interaction; Second International Conference, ACII 2007, Lisbon, Portugal, September 12–14, 2007.
39. PENTLAND, A., MOGHADDAM, B., STARNER, T., OLIYIDE, O. and TURK, M., 1993. View-based and modular Eigenspaces for face recognition. Technical Report 245, M.I.T Media Lab.
40. PHILIPS, 2023. Teaming up to advance health equity [online]. Available from: https://www.philips.com/a-w/about/news/archive/features/2023/20230622-teaming-up-to-advance-health-equity.html [Accessed 2 April 2025].
41. SCIENTIFIC RESEARCH PUBLISHING (SCIRP), 2022. Real-time eye blink detection and its application in hands-free interfaces. Scientific Research Publishing. [online] Available from: https://www.scirp.org [Accessed 2 June 2025].
42. SEARS, A. and JACKO, J.A., 2009. Human–computer interaction: Development process. CRC Press.
43. SEARS, A. and JACKO, J.A., eds., 2009. Human-Computer Interaction Fundamentals. Boca Raton, FL: CRC Press, Taylor & Francis Group.
44. SEWAK, M., KARIM, M.R. and PUJARI, P., 2018. Practical Convolutional Neural Networks. Birmingham: Packt Publishing.
45. SOUKUPOVA, T. and CECH, J., 2016. Real-time eye blink detection using facial landmarks. Computer Vision Winter Workshop (CVWW). [online] Available from: https://vision.fe.uni-lj.si/cvww2016/proceedings/papers/05.pdf [Accessed 17 June 2025].
46. STUART, S., ed., 2022. Eye Tracking: Background, Methods, and Applications. Neuromethods, vol. 183. New York, NY, USA: Springer Science+Business Media.

47. SUNFOUNDER, 2017. I2C LCD1602 — Ultimate Sensor Kit documentation [online]. Available from: https://docs.sunfounder.com/projects/ultimate-sensor-kit/en/latest/components_basic/21-component_i2c_lcd1602.html [Accessed 21 May 2025].
48. SZELISKI, R., 2011. Computer Vision: Algorithms and Applications. 1st ed. Springer.
49. TECHREPUBLIC, 2020. Apple's Face ID: Cheat sheet. TechRepublic.
50. YOUQ.AI, n.d. Principles of Human-Computer Interaction [online]. Available from: https://youq.ai/skillsets/prompting-language-models/principles-of-human-computer-interaction?utm_source=chatgpt.com [Accessed 2 April 2025].
51. ZGHIGUO, W., 2021. Eye-Tracking with Python and Pylink. Cham, Switzerland: Springer Nature Switzerland AG.

Appendices

Python-Based Source Code B-I

```
#the script should be named as "eye_pause_smile_tuned.py"
import cv2
import dlib
import numpy as np
from scipy.spatial import distance
from gaze_tracking import GazeTracking
from fer import FER
import time
import speech_recognition as sr
import threading
from collections import deque  # For smoothing
import os  # For file operations
# Initialize Gaze Tracking and Smile Detection
gaze = GazeTracking()
detector = FER(mtcnn=False)  # Disable MTCNN for faster initialization
# Initialize speech recognizer
recognizer = sr.Recognizer()
recognizer.energy_threshold = 300  # Lower threshold for better speech detection
# Initialize video capture
cap = cv2.VideoCapture(0)
# Check if camera opened successfully
if not cap.isOpened():
    print("Cannot open camera")
    exit()
# Load pre-trained face detector and facial landmarks model
face_detector = dlib.get_frontal_face_detector()
landmark_predictor =
dlib.shape_predictor("shape_predictor_68_face_landmarks.dat")
# Eye landmark indices (Dlib 68 points model)
LEFT_EYE = list(range(42, 48))
RIGHT_EYE = list(range(36, 42))
# Create or clear the signal file at the start
signal_file_path = os.path.join(os.getcwd(), "smile_gaze_signals.txt")
```

```
with open(signal_file_path, "w") as f:
    f.write("")  # Clear the file
# Function to write signals to the file
def write_signal(signal):
    with open(signal_file_path, "a") as f:
        f.write(signal + "\n")
# Eye aspect ratio function
def eye_aspect_ratio(eye):
    A = distance.euclidean(eye[1], eye[5])  # Vertical distance
    B = distance.euclidean(eye[2], eye[4])  # Vertical distance
    C = distance.euclidean(eye[0], eye[3])  # Horizontal distance
    if C == 0:
        return 0  # Prevent division by zero
    ear = (A + B) / (2.0 * C)
    return ear
# Blink detection parameters
EYE_AR_THRESH = 0.19
EYE_AR_CONSEC_FRAMES = 1  # Adjusted to integer
BLINK_RESET_FRAMES = 1    # Adjusted to integer
blink_counter = 0
frame_counter = 0
blink_detected = False
# Flags for PAUSE detection
pause_flag = False
pause_flag_previous = False  # To detect changes in pause state
# Variables for smile detection
smile_detected = False  # Flag to check if smile has been detected
smile_confidence_threshold = 0.2  # Lower confidence threshold to consider a smile
smile_buffer = deque(maxlen=3)  # Smaller buffer for more responsiveness
# List to store blink timestamps during smile
blink_times = []  # Stores timestamps of blinks during a smile
# Variables for gaze detection
gaze_direction_previous = None  # To detect changes in gaze direction
# Variables to handle gaze duration for left and right
gaze_start_time = None  # Start time for gaze direction
gaze_signal_sent = False  # To ensure signal is sent only once after duration
```

```
def listen_for_commands():
    global pause_flag
    with sr.Microphone() as source:
        while True:
            try:
                audio = recognizer.listen(source, timeout=5, phrase_time_limit=4)
                command = recognizer.recognize_google(audio).lower()
                # For pausing (activates P1)
                if ("take a break please" in command or
                    "take a pause please" in command or
                    "stop please" in command or
                    "you can rest" in command):
                    pause_flag = True
                # For resuming (activates P0)
                elif ("go on please" in command or
                      "continue please" in command or
                      "shall we continue" in command or
                      "let's go on" in command):
                    pause_flag = False
            except (sr.UnknownValueError, sr.RequestError, sr.WaitTimeoutError):
                pass
# Run speech recognition in a separate thread
threading.Thread(target=listen_for_commands, daemon=True).start()
while True:
    ret, frame = cap.read()
    if not ret:
        break
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    faces = face_detector(gray)
    # Process gaze tracking
    gaze.refresh(frame)
    annotated_frame = gaze.annotated_frame()
    text = ""
    gaze_direction = None  # Current gaze direction
    horizontal_ratio = gaze.horizontal_ratio()
    if horizontal_ratio is not None:
```

```
            # Use your specified thresholds for gaze detection
            if horizontal_ratio <= 0.45:
                text = "Looking right"
                gaze_direction = "right"
            elif horizontal_ratio >= 0.68:
                text = "Looking left"
                gaze_direction = "left"
            else:
                text = "Looking center"
                gaze_direction = "center"
        else:
            # In case gaze tracking fails, default to center
            text = "Searching for gaze"
            gaze_direction = "center"
        cv2.putText(annotated_frame, text, (90, 60), cv2.FONT_HERSHEY_DUPLEX,
1.6, (147, 58, 31), 2)
        # Implement duration-based gaze detection for left and right
        current_time = time.time()
        if gaze_direction != gaze_direction_previous:
            gaze_start_time = current_time
            gaze_signal_sent = False
        elif gaze_direction in ["left", "right"]:
            if not gaze_signal_sent:
                duration = current_time - gaze_start_time
                if duration >= 1:
                    if gaze_direction == "left":
                        write_signal("L")
                    elif gaze_direction == "right":
                        write_signal("R")
                    gaze_signal_sent = True  # Ensure signal is sent only once until gaze
direction changes
        else:
            # Reset if gaze is center or undetected
            gaze_start_time = current_time
            gaze_signal_sent = False
        gaze_direction_previous = gaze_direction
```

```
# Get pupil coordinates
left_pupil = gaze.pupil_left_coords()
right_pupil = gaze.pupil_right_coords()
cv2.putText(annotated_frame, "Left pupil:  " + str(left_pupil), (90, 130),
        cv2.FONT_HERSHEY_DUPLEX, 0.9, (147, 58, 31), 1)
cv2.putText(annotated_frame, "Right pupil: " + str(right_pupil), (90, 165),
        cv2.FONT_HERSHEY_DUPLEX, 0.9, (147, 58, 31), 1)
# Process smile detection
results = detector.detect_emotions(frame)
smiling = False  # Flag to check if currently smiling
if results:
    largest_face = max(results, key=lambda f: f["box"][2] * f["box"][3])
    (x, y, w, h) = largest_face["box"]
    cv2.rectangle(annotated_frame, (x, y), (x + w, y + h), (0, 255, 0), 2)
    emotions = largest_face["emotions"]
    happy_confidence = emotions.get("happy", 0)
    # Append the happy confidence to the buffer
    smile_buffer.append(happy_confidence)
    # Calculate the average happy confidence over the buffer
    avg_happy_confidence = np.mean(smile_buffer)
    if avg_happy_confidence >= smile_confidence_threshold:
        cv2.putText(annotated_frame, "smile", (x, y - 10),
                cv2.FONT_HERSHEY_SIMPLEX, 0.9, (36, 255, 12), 2)
        smiling = True
else:
    # If no face is detected, clear the smile buffer
    smile_buffer.clear()
# Smile detection logic with smoothing
if smiling and not smile_detected:
    # Smile has just started
    smile_detected = True
    write_signal("S")  # Write "S" when smile starts
    blink_times = []  # Reset blink times when a new smile starts
elif not smiling and smile_detected:
    # Smile has just ended
    smile_detected = False
```

```
            blink_times = []  # Reset blink times when smile ends
        # Blink detection
        blink_text = ""
        for face in faces:
            landmarks = landmark_predictor(gray, face)
            landmarks = np.array([(landmarks.part(n).x, landmarks.part(n).y) for n in
range(68)])
            left_eye = landmarks[LEFT_EYE]
            right_eye = landmarks[RIGHT_EYE]
            left_ear = eye_aspect_ratio(left_eye)
            right_ear = eye_aspect_ratio(right_eye)
            ear = (left_ear + right_ear) / 2.0
            if ear < EYE_AR_THRESH:
                frame_counter += 1
                if frame_counter >= EYE_AR_CONSEC_FRAMES and not blink_detected:
                    blink_counter += 1
                    blink_detected = True
                    blink_text = "Blink Detected"
                    write_signal("B")  # Write "B" when blink is detected
                    if smile_detected:
                        # Record blink timestamp
                        current_time_blink = time.time()
                        blink_times.append(current_time_blink)
                        # Remove blinks older than 5 seconds
                        blink_times = [t for t in blink_times if current_time_blink - t <= 5]
                        # Check if there are at least 3 blinks within 5 seconds
                        if len(blink_times) >= 3:
                            write_signal("SB")  # Write "SB" for Smile + 3 blinks
                            blink_times = []  # Reset blink times to avoid repeated detection
            else:
                if blink_detected and frame_counter >= BLINK_RESET_FRAMES:
                    blink_detected = False
                frame_counter = 0
            # Draw eye landmarks
            for point in np.concatenate((left_eye, right_eye), axis=0):
                cv2.circle(annotated_frame, tuple(point), 2, (0, 255, 0), -1)
```

```
            cv2.putText(annotated_frame, f"Blinks: {blink_counter}", (10, 30),
                        cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 0, 255), 2)
            if blink_text:
                cv2.putText(annotated_frame, blink_text, (10, 60),
                            cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 0, 255), 2)
        # Display "PAUSE" if activated
        if pause_flag:
            cv2.putText(annotated_frame, "PAUSE", (100, 100),
cv2.FONT_HERSHEY_SIMPLEX, 2, (0, 0, 255), 4)
            if pause_flag != pause_flag_previous:
                write_signal("P1")  # Write "P1" when PAUSE is activated
        else:
            if pause_flag != pause_flag_previous:
                write_signal("P0")  # Write "P0" when PAUSE is deactivated
        pause_flag_previous = pause_flag  # Update previous state
        cv2.imshow("Smile, Gaze, Blink & Speech Detection", annotated_frame)
        # Exit on 'Esc' key press
        if cv2.waitKey(1) & 0xFF == 27:
            break
    cap.release()
    cv2.destroyAllWindows()
```


Python-Based Source Code B-II

```
#the script should be named as "smile_eye_pause_signals.py"
import threading
import subprocess
import time
import os
import requests
# ESP8266 IP Address (change this to your ESP8266 IP address)
ESP8266_IP = "192.168.43.174"
# Dictionary mapping detected signals to ESP8266 URLs
signal_to_url = {
    "S": f"http://{ESP8266_IP}/S",
    "L": f"http://{ESP8266_IP}/L",
    "R": f"http://{ESP8266_IP}/R",
```

```
    "B": f"http://{ESP8266_IP}/B",
    "SB": f"http://{ESP8266_IP}/SB",
    "P1": f"http://{ESP8266_IP}/P1",
    "P0": f"http://{ESP8266_IP}/P0",
    "W": f"http://{ESP8266_IP}/W"
}
def send_command(signal):
    """
    Send a command to the ESP8266 based on the detected signal.
    """
    if signal in signal_to_url:
        url = signal_to_url[signal]
        try:
            print(f"Sending request to URL: {url}")
            response = requests.get(url, timeout=5)
            if response.status_code == 200:
                print(f"Command sent for '{signal}'!")
                print("ESP8266 should respond and display this in the Serial Monitor.")
                print("Response from ESP8266:", response.text)
            else:
                print(f"Failed to send command. Status code: {response.status_code}")
                print("Response content:", response.text)
        except requests.exceptions.RequestException as e:
            print(f"An error occurred while sending the command: {e}")
    else:
        print(f"Invalid signal received: {signal}")
def run_eye_tracking():
    """Run the eye tracking script (eye_pause_smile_tuned.py)."""
    script_path = os.path.join(os.getcwd(), 'eye_pause_smile_tuned.py')
    subprocess.Popen(['python', script_path])
def clear_output_file():
    """Clear the smile_gaze_signals.txt file before starting."""
    with open('smile_gaze_signals.txt', 'w') as file:
        file.write('')
def read_signals():
    """
```

```
    Continuously read the signals from 'smile_gaze_signals.txt' and send the
    corresponding commands to the ESP8266.
    """
    signal_file = 'smile_gaze_signals.txt'
    file_position = 0
    while True:
        if os.path.exists(signal_file):
            with open(signal_file, 'r') as file:
                file.seek(file_position)
                while True:
                    line = file.readline()
                    if not line:
                        break  # No new data
                    file_position = file.tell()
                    signal = line.strip()
                    if signal:
                        print(f"Detected signal: {signal}")
                        send_command(signal)
        else:
            time.sleep(0.1)
        time.sleep(0.1)
if __name__ == '__main__':
    # Clear the file on startup
    clear_output_file()
    # Launch the eye tracking script in a separate thread (opens its own camera window)
    threading.Thread(target=run_eye_tracking, daemon=True).start()
    # Wait for several seconds to allow the camera to open
    time.sleep(5)
    # Send the startup signal "W" after the camera has opened
    send_command("W")
    # Start reading signals from the file and sending them to the ESP8266
    try:
        read_signals()
    except KeyboardInterrupt:
        print("KeyboardInterrupt received. Exiting...")
```

ESP8266-Based Hardware Control Modules D-I

```
#include <Wire.h>
#include <hd44780.h>
#include <hd44780ioClass/hd44780_I2Cexp.h>
#include <ESP8266WiFi.h>

// Initialize LCD
hd44780_I2Cexp lcd;
// WiFi Credentials
const char* ssid = "redmi";
const char* password = "areo2001";
WiFiServer server(80);  // HTTP Server on port 80

// Pin Assignments
const int motorPin = D4;
const int readyLed = D0;    // Main LED (quick blink for "B")
const int ledA1   = D6;     // Right LED (For R signal)
const int ledA2   = D5;     // Left LED (For L signal)
const int ledA3   = D1;     // Unused in this sketch
const int ledD1   = D7;     // Now becomes Pause Indicator (for P1)
const int ledD2    = D8;     // Now becomes System Ready Indicator (after S; when
system ready and not paused)

bool motorState = false;
bool ipPrinted = false;
bool systemReady = false;  // Becomes true after S is received
bool pauseActive = false;

void setup() {
  Serial.begin(74880);
  delay(1000);
  // Initialize LCD
  lcd.begin(16, 2);
  lcd.backlight();
  lcd.setCursor(0, 0);
```

```
lcd.print("ESP Booting...");
// Initialize GPIO Pins
pinMode(motorPin, OUTPUT);
pinMode(readyLed, OUTPUT);
pinMode(ledA1, OUTPUT);
pinMode(ledA2, OUTPUT);
pinMode(ledD1, OUTPUT);  // Pause Indicator (D7)
pinMode(ledD2, OUTPUT);  // System Ready Indicator (D8)
// Initial LED state:
// At powerup, only the main LED (D0) is ON.
digitalWrite(motorPin, LOW);
digitalWrite(readyLed, HIGH);  // Main LED ON
digitalWrite(ledA1, LOW);
digitalWrite(ledA2, LOW);
digitalWrite(ledD1, LOW); // Pause Indicator OFF
digitalWrite(ledD2, LOW); // System Ready Indicator OFF
Serial.println("\nStarting ESP8266...");
lcd.setCursor(0, 1);
lcd.print("Connecting WiFi");
WiFi.begin(ssid, password);
int attempts = 0;
while(WiFi.status() != WL_CONNECTED) {
  delay(500);
  Serial.print(".");
  lcd.setCursor(15, 1);
  lcd.print("|");
  attempts++;
  if(attempts > 40){
    Serial.println("\nWiFi Timeout! Restarting ESP...");
    lcd.clear();
    lcd.setCursor(0, 0);
    lcd.print("WiFi Failed!");
    delay(3000);
    ESP.restart();
  }
}
```

```
  Serial.println("\nWiFi connected!");
  Serial.print("ESP8266 IP: ");
  Serial.println(WiFi.localIP());
  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print("WiFi Connected!");
  lcd.setCursor(0, 1);
  lcd.print(WiFi.localIP());
  server.begin();
  // System not ready until S is received
  systemReady = false;
  pauseActive = false;
}

void resetSystem() {
  // Reset system: All LEDs off except main LED (D0) on.
  digitalWrite(ledA1, LOW);
  digitalWrite(ledA2, LOW);
  digitalWrite(ledD1, LOW);   // Pause Indicator OFF
  digitalWrite(ledD2, LOW);   // System Ready Indicator OFF
  digitalWrite(motorPin, LOW);
  systemReady = false;
  pauseActive = false;
  motorState = false;
  digitalWrite(readyLed, HIGH);  // Main LED ON
  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print("Cam Off");
  lcd.setCursor(0, 1);
  lcd.print("System Reset");
}

// ---------------- Loop ----------------
void loop() {
  WiFiClient client = server.available();
  if(WiFi.status() == WL_CONNECTED && !ipPrinted) {
```

```
  Serial.print("Current ESP8266 IP: ");
  Serial.println(WiFi.localIP());
  ipPrinted = true;
 }
 if(!client){
  delay(10);
  return;
 }
 Serial.println("New Client Connected");
 while(!client.available()){
  delay(1);
 }
 String request = client.readStringUntil('\r');
 Serial.print("Received Request: ");
 Serial.println(request);
 client.flush();
 String response;
 // Process top-level commands first (W and F)
 if (request.indexOf("/W") != -1) {
  // W: Startup signal. Announce after camera is open.
  Serial.println("Command: Signal W (Startup)");
  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print("Cam Initiating");
  lcd.setCursor(0, 1);
  lcd.print("Awaiting Smile");
  digitalWrite(readyLed, HIGH);  // Main LED remains ON in reset state
  digitalWrite(ledD1, LOW);      // Pause Indicator OFF
  digitalWrite(ledD2, LOW);      // System Ready Indicator OFF
  systemReady = false;
  response = "<html><body><h1>Startup State</h1></body></html>";
 }
 else if (request.indexOf("/F") != -1) {
  // F: Finish signal. Reset the system.
  Serial.println("Command: Signal F (Finish)");
  resetSystem();
```

```
  response = "<html><body><h1>System Reset (Finish)</h1></body></html>";
  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print("Cam Off");
 }
 // Process other commands only if system is ready
 else if (!systemReady) {
  if (request.indexOf("/S") != -1) {
   systemReady = true;
   Serial.println("Command: System Ready (S)");
   lcd.clear();
   lcd.setCursor(0, 0);
   lcd.print("Smile Detected");
   lcd.setCursor(0, 1);
   lcd.print("System Ready!");
   digitalWrite(readyLed, LOW);   // Main LED OFF
   digitalWrite(ledD2, HIGH);     // System Ready Indicator (D8) ON
   digitalWrite(ledD1, LOW);      // Pause Indicator (D7) OFF
   response = "<html><body><h1>System Ready</h1></body></html>";
  }
  else {
   response = "<html><body><h1></h1></body></html>";
   client.println("HTTP/1.1 200 OK");
   client.println("Content-Type: text/html");
   client.println("Connection: close");
   client.println();
   client.println(response);
   Serial.println("Awaiting 'S' command.");
   delay(1);
   Serial.println("Client disconnected.");
   return;
  }
 }
 if(pauseActive && request.indexOf("/P0") == -1) {
  Serial.println("System is paused. Waiting for 'P0' to resume.");
  response = "<html><body><h1>System Paused</h1></body></html>";
```

```
  client.println("HTTP/1.1 200 OK");
  client.println("Content-Type: text/html");
  client.println("Connection: close");
  client.println();
  client.println(response);
  delay(1);
  Serial.println("Client disconnected.");
  return;
 }
 // Process individual commands
 if (request.indexOf("/L") != -1) {
  Serial.println("Command: Signal L");
  digitalWrite(ledA2, HIGH);   // Turn on Left LED (D5)
  delay(3000);                 // Remain ON for 3 seconds
  digitalWrite(ledA2, LOW);
  response = "<html><body><h1>Signal L Processed</h1></body></html>";
  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print("Left LED On 3s");
 }
 else if (request.indexOf("/R") != -1) {
  Serial.println("Command: Signal R");
  digitalWrite(ledA1, HIGH);   // Turn on Right LED (D6)
  delay(3000);                 // Remain ON for 3 seconds
  digitalWrite(ledA1, LOW);
  response = "<html><body><h1>Signal R Processed</h1></body></html>";
  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print("Right LED On 3s");
 }
 else if (request.indexOf("/P1") != -1) {
  Serial.println("Command: Signal P1 (Pause)");
  pauseActive = true;
  digitalWrite(ledD1, HIGH);   // D7 ON for pause
  digitalWrite(ledD2, LOW);    // D8 OFF during pause
  response = "<html><body><h1>Pause Activated</h1></body></html>";
```

```
  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print("System Paused");
 }
 else if (request.indexOf("/P0") != -1) {
  Serial.println("Command: Signal P0 (Resume)");
  pauseActive = false;
  digitalWrite(ledD2, HIGH);   // D8 ON when ready
  digitalWrite(ledD1, LOW);    // D7 OFF when not paused
  response = "<html><body><h1>Pause Deactivated</h1></body></html>";
  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print("System Resumed");
 }
 else if (request.indexOf("/SB") != -1) {
  Serial.println("Command: Signal SB (Motor)");
  motorState = !motorState;
  digitalWrite(motorPin, motorState ? HIGH : LOW);
  response = motorState ? "<html><body><h1>Motor ON</h1></body></html>"
                        : "<html><body><h1>Motor OFF</h1></body></html>";
  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print(motorState ? "Motor: ON" : "Motor: OFF");
 }
 else if (request.indexOf("/B") != -1) {
  Serial.println("Command: Signal B");
  digitalWrite(readyLed, HIGH);  // Blink main LED (D0)
  delay(200);                    // Quick blink
  digitalWrite(readyLed, LOW);
  response = "<html><body><h1>Signal B Processed</h1></body></html>";
  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print("Main LED Blinked");
 }
 else {
  response = "<html><body><h1></h1></body></html>";
```

```
  }
  client.println("HTTP/1.1 200 OK");
  client.println("Content-Type: text/html");
  client.println("Connection: close");
  client.println();
  client.println(response);
  Serial.println("Response sent to client.");
  delay(1);
  Serial.println("Client disconnected.");
}
```

ESP8266-Based Hardware Control Module D-II

```
#include<Wire.h>
#include<hd44780.h>
#include<hd44780ioClass/hd44780_I2Cexp.h>
#include<ESP8266WiFi.h>
hd44780_I2Cexp lcd;
const char* ssid="redmi";
const char* password="areo2001";
WiFiServer server(80);
const int motorPin=D4;
const int readyLed=D0;
const int ledA1=D6;
const int ledA2=D5;
const int ledD1=D7;
const int ledD2=D8;
bool readyState=true;
bool ledA1State=true;
bool ledA2State=true;
bool pauseActive=false;
bool systemReady=false;
bool motorState=false;
const char* lcdMessages[]={
"LCD Ready",
"Hello, World!",
"ESP8266 is awesome!",
```

```
"Have a great day!",
"Cycle Complete"
};
const int numLcdMessages=sizeof(lcdMessages)/sizeof(lcdMessages[0]);
int lcdMsgIndex=0;
void handleRequest(String request){
if(request.indexOf("device=ready")!=-1){
readyState=!readyState;
digitalWrite(readyLed,readyState?HIGH:LOW);
}
else if(request.indexOf("device=ledA1")!=-1){
ledA1State=!ledA1State;
digitalWrite(ledA1,ledA1State?HIGH:LOW);
}
else if(request.indexOf("device=ledA2")!=-1){
ledA2State=!ledA2State;
digitalWrite(ledA2,ledA2State?HIGH:LOW);
}
else if(request.indexOf("device=ledD1")!=-1){
pauseActive=!pauseActive;
digitalWrite(ledD1,pauseActive?HIGH:LOW);
}
else if(request.indexOf("device=ledD2")!=-1){
systemReady=!systemReady;
digitalWrite(ledD2,systemReady?HIGH:LOW);
}
else if(request.indexOf("device=motor")!=-1){
motorState=!motorState;
digitalWrite(motorPin,motorState?HIGH:LOW);
}
else if(request.indexOf("device=lcdToggle")!=-1){
lcd.clear();
lcd.setCursor(0,0);
lcd.print(lcdMessages[lcdMsgIndex]);
lcdMsgIndex=(lcdMsgIndex+1)%numLcdMessages;
}
```

```
  }
  void sendMainPage(WiFiClient&client){
  String html="<!DOCTYPE html><html><head><meta charset='utf-8'><title>ESP8266 Control Panel</title></head><body>";
  html+="<h1>ESP8266 Control Panel</h1>";
  html+="<h2>LED Controls</h2>";
  html+="<p>Main LED (B) - readyLed (D0): ";
  html+="<a href='/control?device=ready'><button>";
  html+=(readyState?"Turn OFF":"Turn ON");
  html+="</button></a></p>";
  html+="<p>Right LED (R signal) - ledA1 (D6): ";
  html+="<a href='/control?device=ledA1'><button>";
  html+=(ledA1State?"Turn OFF":"Turn ON");
  html+="</button></a></p>";
  html+="<p>Left LED (L signal) - ledA2 (D5): ";
  html+="<a href='/control?device=ledA2'><button>";
  html+=(ledA2State?"Turn OFF":"Turn ON");
  html+="</button></a></p>";
  html+="<h2>Status Indicators</h2>";
  html+="<p>Pause Indicator (P1) - ledD1 (D7): ";
  html+="<a href='/control?device=ledD1'><button>";
  html+=(pauseActive?"Turn OFF":"Turn ON");
  html+="</button></a></p>";
  html+="<p>System Ready Indicator - ledD2 (D8): ";
  html+="<a href='/control?device=ledD2'><button>";
  html+=(systemReady?"Turn OFF":"Turn ON");
  html+="</button></a></p>";
  html+="<h2>Motor Control (D4)</h2>";
  html+="<p>Motor: ";
  html+="<a href='/control?device=motor'><button>";
  html+=(motorState?"Turn OFF":"Turn ON");
  html+="</button></a></p>";
  html+="<h2>LCD Message</h2>";
  html+="<p><a href='/control?device=lcdToggle'><button>Change LCD Message</button></a></p>";
  html+="</body></html>";
```

```
client.println("HTTP/1.1 200 OK");
client.println("Content-Type: text/html");
client.println("Connection: close");
client.println();
client.println(html);
}
void setup(){
Serial.begin(74880);
delay(1000);
Serial.println("\nStarting ESP8266...");
pinMode(readyLed,OUTPUT);
pinMode(ledA1,OUTPUT);
pinMode(ledA2,OUTPUT);
pinMode(ledD1,OUTPUT);
pinMode(ledD2,OUTPUT);
pinMode(motorPin,OUTPUT);
digitalWrite(readyLed,HIGH);
digitalWrite(ledA1,HIGH);
digitalWrite(ledA2,HIGH);
digitalWrite(ledD1,LOW);
digitalWrite(ledD2,LOW);
digitalWrite(motorPin,LOW);
Wire.begin();
lcd.begin(16,2);
lcd.setCursor(0,0);
lcd.print("LCD Ready");
Serial.println("Connecting to WiFi...");
WiFi.begin(ssid,password);
int attempts=0;
while(WiFi.status()!=WL_CONNECTED){
delay(500);
Serial.print(".");
attempts++;
if(attempts>40){
Serial.println("\nWiFi Timeout! Restarting ESP...");
delay(3000);
```

```
ESP.restart();
}
}
Serial.println("\nWiFi Connected!");
Serial.print("ESP8266 IP Address: ");
Serial.println(WiFi.localIP());
server.begin();
Serial.println("HTTP server started");
}
void loop(){
WiFiClient client=server.available();
if(client){
while(client.connected()&&!client.available()){
delay(1);
}
```