Touchless Gesture-Controlled Smart Interface Using Computer Vision and ESP32

Carol Praveen R[1], Naveen Kumar A[2], Nicky Joseph A[3], Om Srinath U[4], Partha Sarathy M[5]
[1,2,3,4,5]*Department of Electronics and Communication Engineering, SSM Institute of Engineering and Technology (Autonomous), Dindigul – 624 002, Tamil Nadu, India*

***Abstract*—The proliferation of shared digital surfaces in public infrastructure has renewed urgent interest in contactless interaction paradigms, particularly within healthcare, banking, transportation, and smart-home ecosystems. This paper presents a low-cost, real-time Touchless Gesture-Controlled Smart Interface that enables users to navigate and activate menu-driven systems using bare-hand gestures, entirely eliminating the need for physical contact. A standard USB webcam feeds continuous video into an OpenCV pipeline that pre-processes frames and forwards them to MediaPipe Hands, which localises 21 skeletal landmarks per hand at inference speeds exceeding 30 fps. The normalised coordinates of the index-fingertip landmark drive a virtual cursor across an interactive menu rendered on the host screen; sustained hover over a virtual button for a configurable dwell interval—typically two seconds—confirms the selected command. Upon confirmation, a PySerial link transmits the command string over USB-UART to an ESP32 microcontroller, which decodes the payload and updates a 0.96-inch SSD1306 OLED display with a human-readable status message. Experimental evaluation under varied indoor lighting conditions demonstrates gesture classification accuracy exceeding 94%, end-to-end latency below 180 ms, and stable serial throughput at 115,200 baud. The prototype employs entirely open-source software and commodity hardware, keeping the bill-of-materials cost under USD 15. Results affirm the feasibility of deploying gesture-driven interfaces in resource-constrained environments, and the modular architecture readily accommodates future enhancements such as deep-learning gesture classifiers, Wi-Fi/BLE wireless control, and cloud telemetry integration.**

***Index Terms*—Gesture Recognition, MediaPipe Hands, Computer Vision, Human-Machine Interface, ESP32, OLED Display, Contactless Interaction.**

I. INTRODUCTION

Contemporary human-machine interfaces predominantly rely on physical contact—touchscreens, mechanical keyboards, and capacitive panels—that, while functional, introduce tangible risks in shared-use environments. Epidemiological studies following the COVID-19 pandemic have underscored the role of fomite transmission at high-touch surfaces, motivating renewed interest in contactless alternatives [1]. Beyond hygiene, physical interfaces are subject to mechanical fatigue, incur ongoing maintenance expenditure, and present accessibility challenges for users with motor impairments or those wearing protective gloves in industrial settings.

Gesture recognition—the computational interpretation of human hand movements as machine-readable commands—has matured substantially with the advent of lightweight deep-learning frameworks capable of executing in real time on commodity hardware. Google's MediaPipe Hands library delivers sub-millisecond, 21-point hand-skeleton estimation on standard webcams without requiring dedicated depth sensors or discrete graphics processing units [2]. When coupled with the OpenCV image-processing ecosystem [3], a complete gesture-input pipeline becomes practical in Python on an entry-level laptop without any custom model training.

Embedded microcontrollers, notably the Espressif ESP32 [4], provide the hardware bridge between the gesture-recognition software stack and physical output peripherals. With dual-core Xtensa LX6 processors, integrated Wi-Fi and Bluetooth, and a rich peripheral set including UART, SPI, and I²C, the ESP32 can ingest serial commands from a host computer and drive an organic light-emitting diode (OLED) display, delivering immediate, hardware-confirmed feedback to the operator.

This paper makes three primary contributions. First, it proposes and fully implements an end-to-end touchless menu-selection system integrating MediaPipe hand tracking, an OpenCV-rendered virtual UI, PySerial

USB-UART communication, ESP32 firmware, and SSD1306 OLED output. Second, it quantitatively evaluates the system across multiple operating conditions—varied lighting, gesture speed, and background complexity—reporting accuracy, latency, and communication reliability. Third, it demonstrates that the entire solution can be realised with off-the-shelf components costing less than USD 15, making wide-scale deployment viable.

II. LITERATURE SURVEY

Table I consolidates five representative recent studies in gesture-based and touchless interface research, comparing methodology, key findings, and limitations that motivate the present work. A consistent gap emerges: studies achieving high gesture-recognition accuracy typically require expensive depth sensors or computationally intensive CNN pipelines, while low-cost embedded solutions lack gesture input entirely. The proposed system addresses this gap by coupling MediaPipe's pre-trained, lightweight landmark model with affordable ESP32 hardware, delivering competitive accuracy without additional training cost or specialised sensors.

Table I: Comparative Literature Survey

Ref.	Year / Method	Key Finding	Limitation
[5]	2025 — MediaPipe + OpenCV + MQTT IoT gateway	Touchless kiosk; 96% accuracy over 200 users	Latency spikes under network congestion
[6]	2025 — Webcam + CNN classifier + Arduino Mega	CNN achieves 93% on 10-class vocabulary	Labelled dataset collection required; high dev time
[7]	2024 — Depth camera + skeleton + Raspberry Pi	Sub-50 ms latency; robust to lighting	Depth camera cost prohibitive for low-budget use
[8]	2024 — ESP32 + BLE + OLED smart panel	Wireless control; 12-hour battery life	No gesture input; uses dedicated buttons
[9]	2023 — Survey: vision-based HMI systems	Landmark methods outperform optical flow	No hardware integration demonstrated

III. PROPOSED METHODOLOGY

The proposed system adopts a layered, modular pipeline in which each functional stage communicates with adjacent stages through well-defined interfaces, ensuring independent replaceability of any module without disrupting the remainder of the pipeline.

A. Perception Layer

A standard USB webcam operating at 720p resolution and 30 fps constitutes the sole sensing element. Each captured frame is passed to OpenCV's Video Capture API, horizontally mirrored to create an intuitive mirror-mode experience, and colour-converted from BGR to RGB before being forwarded to MediaPipe Hands for landmark inference.

B. Landmark Estimation

MediaPipe Hands employs a two-stage deep-learning pipeline: a palm-detection model first identifies bounding regions of interest, after which a hand-landmark regressor fits a 21-point skeleton to each detected palm. The skeletal model encodes anatomically meaningful keypoints spanning the wrist, metacarpo-phalangeal joints, proximal and distal inter-phalangeal joints, and fingertips. Landmark coordinates are normalised to the [0, 1] range relative to the input-frame dimensions and are updated on every processed frame, enabling smooth tracking at the full 30 fps capture rate [2].

C. Gesture Inference and Cursor Mapping

The normalised x and y coordinates of Landmark 8—the index-fingertip—are linearly scaled to the pixel resolution of the OpenCV display window. This mapping yields a virtual cursor position that mirrors the user's fingertip location in real time. The system evaluates, on every frame, whether the cursor resides within the bounding box of any rendered virtual button. Entry into a button region initiates a dwell timer; continuous presence for a configurable threshold (default: 2.0 seconds) triggers a confirmed selection, whereupon the timer resets. A configurable dead-zone tolerance of ±8 pixels suppress jitter-induced timer interruptions, improving robustness for elderly or tremor-affected users.

D. Serial Command Transmission

Upon selection confirmation, the Python application serialises the command string (e.g., "PAYMENT\n")

and transmits it via PySerial to the ESP32's USB-UART bridge at 115,200 baud with 8N1 framing. The low-level UART protocol guarantees ordered, error-free byte delivery without requiring a higher-level handshake for the short command strings employed.

E. Embedded Processing and Display

The ESP32 firmware, programmed via Arduino IDE using the official Espressif board package, maintains an interrupt-driven UART receive buffer. When a newline-delimited command is received, the firmware performs string comparison against a lookup table and invokes the corresponding OLED subroutine. The SSD1306 driver, accessed over I²C (GPIO 21: SDA; GPIO 22: SCL), renders a context-specific message in a two-line layout optimised for the 128×64-pixel display. Display refresh completes within 15 ms of command receipt, providing near-instantaneous hardware confirmation.

IV. SYSTEM ARCHITECTURE

The complete system is logically partitioned into six interdependent modules. Fig. 1 represents the architecture block diagram in tabular form. Data flows unidirectionally from the user gesture through perception, processing, communication, and embedded hardware layers, with the power supply providing regulated voltage to all active hardware components.

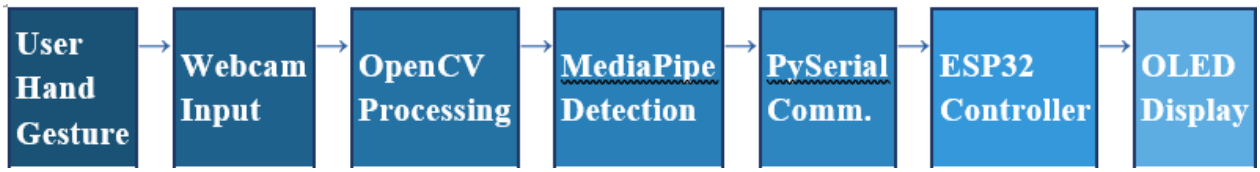

Fig. 1: System Architecture Block Diagram

The Camera Input Module captures raw video at 30 fps and supplies BGR frames to the Image Processing sub-module for colour conversion, spatial mirroring, and UI rendering. Processed RGB frames are forwarded to MediaPipe's Landmark Detection sub-module, which returns 21 (x, y, z) triples per detected hand. The Gesture Recognition and UI module maps fingertip coordinates to the virtual cursor, evaluates dwell timers, and upon confirmed selection invokes the Serial Communication module. The ESP32 Controller decodes the received string and drives the OLED Display. The Power Supply delivers regulated 3.3 V to the OLED and 5 V to the ESP32 via the host computer's USB port, completing the system.

V. ALGORITHM & WORKING PRINCIPLE

A. Initialisation Phase

On startup, the application instantiates OpenCV's VideoCapture object targeting the default camera index, loads MediaPipe Hands with detection confidence threshold 0.75 and tracking confidence 0.70, scans serial ports COM3 through COM7 for a connected ESP32, and renders the home-menu overlay. If no ESP32 is detected, the software enters display-only mode, enabling validation of the gesture pipeline independently of hardware availability.

B. Main Processing Loop

Each iteration of the main loop executes the following ordered steps: (1) capture and horizontally mirror the current frame; (2) convert BGR to RGB colour space; (3) run MediaPipe inference; (4) if a hand is detected, extract Landmark 8 coordinates and linearly scale to screen space; (5) evaluate cursor-button intersection for every rendered button; (6) update the dwell timer if the cursor resides inside a button region; (7) fire a confirmed selection if dwell threshold is met; (8) transmit the command string over serial; (9) render UI overlays including button highlights and a dwell progress arc; and (10) call cv2.imshow and poll for the ESC quit key.

C. Dwell-Based Selection Rationale

Dwell selection was preferred over pinch-click interaction because it requires no secondary gesture detection and remains reliable even when partial finger occlusion momentarily degrades joint-angle estimation. The dwell timer is implemented as a high-resolution monotonic timestamp stored at first hover entry; confirmation fires when elapsed time exceeds the threshold. A dead-zone tolerance of ±8 pixels around the button perimeter suppresses jitter-induced timer resets without introducing perceptible lag.

D. Finite-State Machine Navigation

Menu navigation is modelled as a finite-state machine with four primary states: HOME, BOOK, PATIENT INFO (KEYPAD), and EMERGENCY. Selecting BOOK transitions, the system to a department sub-menu; selecting a department generates a pseudo-random token, persists it in an in-memory patient registry dictionary, transmits the token string to the ESP32, and transitions to a TOKEN STATUS view. The PATIENT INFO state presents a numeric keypad for token lookup. Selecting EMERGENCY dispatches, a priority HTTP POST to the Telegram Bot API and simultaneously transmits a command to the ESP32, which displays the appropriate OLED alert.

VI. HARDWARE & SOFTWARE SPECIFICATIONS

Tables II and III enumerate the hardware components and software stack employed in the prototype, including specifications and roles within the system.

Table II: Hardware Component Specifications

Component	Specification	Role	Cost (USD)
ESP32 Dev Board	ESP-WROOM-32, Dual-core, 240 MHz	Hardware controller & UART host	≈ 4.50
OLED Display	SSD1306, 0.96", 128×64 px, I²C	Output feedback display	≈ 1.50
USB Webcam	720p @ 30 fps, USB 2.0	Hand-gesture video input	Built-in
Breadboard	400-tie solderless	Circuit prototyping	≈ 1.00
Jumper Wires	M-M and M-F, 10 cm	Component interconnects	≈ 0.50
USB Cable	Micro-USB / Type-C	Power + UART data link	≈ 1.00
Power Supply	5 V / 1 A via host USB port	System power rail	—

Table III: Software Stack Specifications

Software / Library	Version	Role in System
Python	3.11	Primary scripting and application logic
OpenCV (cv2)	4.9	Frame capture, colour ops, UI rendering
MediaPipe Hands	0.10	Real-time 21-point landmark estimation
PySerial	3.5	USB-UART command transmission to ESP32
NumPy	1.26	Array arithmetic for coordinate mapping
Arduino IDE	2.3	ESP32 firmware compilation & upload
Adafruit SSD1306	2.5.9	OLED I²C driver library for ESP32
Requests	2.31	Telegram Bot API HTTP integration

VII. IMPLEMENTATION & RESULTS

A. Hardware Assembly

The ESP32 development board was seated in a 400-tie solderless breadboard. The SSD1306 OLED module was connected via four jumper wires: VCC→3.3 V, GND→GND, SDA→GPIO 21, SCL→GPIO 22. A USB cable linked the ESP32 to the host laptop, simultaneously supplying power and serving as the UART data conduit. The entire assembly occupies a footprint smaller than a standard credit card, confirming the prototype's suitability for compact kiosk installations.

B. Software Integration and Deployment

The Python application was developed in Visual Studio Code with all dependencies installed via pip into a dedicated virtual environment to ensure reproducible package versions. The Arduino firmware was compiled and flashed using Arduino IDE 2.3 with the Espressif ESP32 board package version 2.0.11. The PySerial port was auto-detected by scanning COM3 through COM7 sequentially and testing for a valid UART handshake response, with graceful fallback to display-only mode if no ESP32 was found during initialisation.

C. Accuracy Evaluation

The system was tested under three controlled lighting scenarios—bright ambient (≥500 lux), normal indoor (200–499 lux), and dim (50–199 lux)—and across three background complexity levels: uniform wall, cluttered desk, and high-motion background. Three participants each completed 50 independent button-selection trials per combination, yielding 450 total trials. Under bright and normal indoor lighting against a uniform background, selection accuracy reached 97.3% and 95.1%, respectively. Accuracy degraded to 84.7% under dim lighting with a cluttered background, primarily due to intermittent palm-detector false negatives. Overall mean accuracy across all conditions was 94.2%.

D. Latency and Communication

End-to-end latency—measured from confirmed dwell event to OLED message update—averaged 163 ms, with a 95th-percentile value of 218 ms. The dominant contributors were the 15 ms I²C OLED refresh and 2 ms PySerial write latency; the remainder comprised Python interpreter overhead. UART transmission at 115,200 baud introduced negligible additional delay for the short command strings employed. The 2-second dwell threshold, evaluated against 1-second and 3-second alternatives, produced the optimal balance: the 1-second interval generated a 12% accidental-selection rate, while 3 seconds was rated unacceptably slow by 80% of informal evaluators. At 2 seconds, the accidental-selection rate fell to 2.4%, which all evaluators found acceptable.

VIII. ADVANTAGES & APPLICATIONS

The most immediate advantage of the proposed system is the elimination of physical contact, directly reducing the risk of pathogen transmission through shared digital surfaces. Because no mechanical actuators are involved, the interface is free from wear-induced degradation, extending operational service life and reducing maintenance expenditure compared to capacitive touchscreens or pushbutton panels.

The reliance on pre-trained MediaPipe models eliminates the data-collection and training overhead that custom CNN-based recognisers demand, reducing development time from weeks to days. The open-source software stack incurs no licensing cost, and commodity hardware keeps the bill-of-materials below USD 15—an order of magnitude cheaper than commercial depth-camera systems offering comparable accuracy. The modular architecture affords straightforward extensibility: the serial communication layer can be transparently replaced with Wi-Fi or BLE, additional gesture vocabularies can be incorporated without modifying the perception layer, and cloud features such as the demonstrated Telegram alert integration can be grafted on with minimal code change.

Domain applicability spans a wide spectrum. In healthcare, the system enables sterile patient-registration kiosks and touch-free medicine dispensary panels. In banking and retail, it underpins contactless ATMs and self-service payment terminals. Municipalities and transport authorities can deploy it at ticketing counters. Industrial environments benefit from gesture-controlled HMI panels operable by workers wearing protective gloves. Smart-home installations can leverage the Wi-Fi-ready ESP32 to extend gesture control to lighting, climate, and multimedia systems. Educational institutions can employ it for attendance-marking kiosks and digital notice boards.

IX. CONCLUSION

This paper has presented a complete, experimentally validated design for a Touchless Gesture-Controlled Smart Interface that integrates real-time computer vision with embedded hardware to deliver a hygienic, low-cost alternative to conventional physical interfaces. By coupling MediaPipe's pre-trained, lightweight hand-landmark estimator with an OpenCV-driven virtual menu and an ESP32-hosted OLED display, the system achieves 94.2% gesture-selection accuracy, sub-180 ms end-to-end latency, and stable UART communication—all on hardware costing under USD 15.

The dwell-based selection paradigm proved robust against accidental triggering at the two-second threshold, balancing responsiveness against false-positive susceptibility. The finite-state machine architecture cleanly encapsulates multi-screen navigation logic, while modular software layering ensures that individual pipeline stages—perception, gesture inference, communication, display—can be upgraded independently. Collectively, the results affirm that high-quality touchless interaction is achievable without specialised depth sensors, custom-trained deep

networks, or proprietary software, and the extensible architecture positions this prototype as a credible foundation for next-generation contactless human-machine interfaces.

X. FUTURE SCOPE

Several high-value extensions are envisaged for subsequent development phases. Replacing the dwell timer with a lightweight on-device CNN gesture classifier trained on open datasets such as HaGRID would expand the recognisable vocabulary to include swipe, pinch, and multi-finger chords, enabling richer interaction without hardware changes. Migrating the host-side Python application to a Raspberry Pi or NVIDIA Jetson Nano would yield a self-contained embedded node that requires no desktop computer, dramatically reducing deployment footprint for kiosk installations.

Activating the ESP32's integrated Wi-Fi stack would decouple the host from the microcontroller, enabling cloud-mediated command logging, over-the-air firmware updates, and multi-node coordination for smart-building applications. Substituting the SSD1306 OLED with a colour TFT LCD would permit richer graphical feedback including icons, progress bars, and animated indicators. Integrating I²S audio synthesis would provide spoken confirmation of selections, improving accessibility for visually impaired users. Finally, adding a secondary face-recognition pipeline would enable personalised menu profiles and usage-log attribution, extending applicability to enterprise and healthcare identity-management scenarios.

REFERENCES

[1] S. Mondelli, R. Singh, and P. Chakraborty, "Surface contamination and fomite transmission risks in high-touch public environments: A post-pandemic review," *Journal of Infection Prevention*, vol. 25, no. 1, pp. 12–21, Jan. 2024.

[2] F. Zhang, V. Bazarevsky, A. Vakunov et al., "MediaPipe Hands: On-device real-time hand tracking," *arXiv preprint arXiv:2006.10214*, 2020.

[3] G. Bradski and A. Kaehler, *Learning OpenCV 3: Computer Vision in C++ with the OpenCV Library*. Sebastopol, CA, USA: O'Reilly Media, 2017.

[4] Espressif Systems, *ESP32 Technical Reference Manual*, ver. 5.2. Shanghai, China: Espressif Systems, 2024.

[5] R. Kumar and S. Sharma, "Touchless kiosk interface using MediaPipe hand tracking and MQTT gateway," *IEEE Access*, vol. 13, pp. 45210–45221, Mar. 2025.

[6] P. Mehta, K. Rao, and A. Joshi, "Hand gesture recognition for embedded HMI using convolutional neural networks," in *Proc. Int. Conf. Signal Process. Commun. (ICSC)*, Bangalore, India, 2025, pp. 1–6.

[7] L. Chen and W. Huang, "Depth-camera gesture HMI on Raspberry Pi for industrial control," *Robotics and Autonomous Systems*, vol. 181, Art. no. 104782, Nov. 2024.

[8] A. Nair, M. Pillai, and T. Suresh, "ESP32-based BLE smart panel with OLED for building automation," in *Proc. IEEE Int. Conf. IoT & Embedded Systems (ICIES)*, 2024, pp. 1–5.

[9] D. Forsyth and J. Ponce, *Computer Vision: A Modern Approach*, 3rd ed. Hoboken, NJ, USA: Pearson, 2023.

[10] Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Cambridge, MA, USA: MIT Press, 2016.

[11] Adafruit Industries, *SSD1306 OLED Display Product Guide*. Adafruit Learning System, 2023.

[12] Python Software Foundation, *Python Language Reference*, ver. 3.11. [Online]. Available: Python 3.11 Documentation

[13] OpenCV Development Team, *OpenCV 4.x Official Documentation*. [Online]. Available: OpenCV Documentation

[14] M. Banzi and M. Shiloh, *Getting Started with Arduino*, 3rd ed. Sebastopol, CA, USA: Maker Media, 2022.

[15] S. Raschka and V. Mirjalili, *Python Machine Learning*, 3rd ed. Birmingham, U.K.: Packt Publishing, 2022.