International Journal For Advanced Research In Science & Technology
A peer reviewed international journal
www.ijarst.in
ISSN: 2457-0362

Touchless Interface with Hand Gesture Control Using MediaPipe

System Architecture & Gesture Implementation

Sakshi Basnure (22671A7309)
Dheeraj Gundapuneni
(22671A7346)
Yash Raj Kumar Patil
(22671A7353)
Guide: Ms. Maria Divya Teja Pasala, Asst. Professor, Dept. of AI & ML
Author by: Dheeraj Gundapuneni (dheerajgundapuneni634@gmail.com)

ABSTRACT

This paper presents a real-time touchless interface enabling complete computer mouse control through hand gesture recognition using a standard webcam — no physical input device required. Google's MediaPipe Hands framework detects 21 three-dimensional hand landmarks per frame in real time. Landmark coordinates are mapped to screen space via OpenCV and PyAutoGUI, enabling cursor movement, left/right click, double-click, and scroll. A relational database backend (SQLite/MySQL) logs gesture sessions and performance analytics.

Experimental results: gesture recognition accuracy of 92.8%, average end-to-end latency of 47ms, hand detection rate of 96.4%, and a System Usability Scale (SUS) score of 79.3/100 — rated 'Good' — validated with 15 participants including users with physical disabilities.

Keywords: Touchless Interface, Hand Gesture Recognition, HCI, MediaPipe Hands, OpenCV, PyAutoGUI, Real-Time Processing, Accessibility, Machine Learning, SQLite.

1. INTRODUCTION

Human–Computer Interaction (HCI) has evolved dramatically over four decades — from text-based command-line interfaces to graphical user interfaces (GUIs), touchscreens, and now toward gesture-based and touchless paradigms. The growing demand for more natural, intuitive, and accessible interfaces has driven researchers to explore innovative input modalities beyond traditional keyboards and mice.

Traditional input devices present significant barriers for individuals with physical disabilities, repetitive strain injuries (RSI), or those in restricted-contact environments such as operating theatres, cleanrooms, public kiosks, or sterile manufacturing facilities. The convergence of consumer webcams, advanced computer vision libraries, and real-time ML frameworks has made robust, low-cost gesture recognition feasible entirely in software.

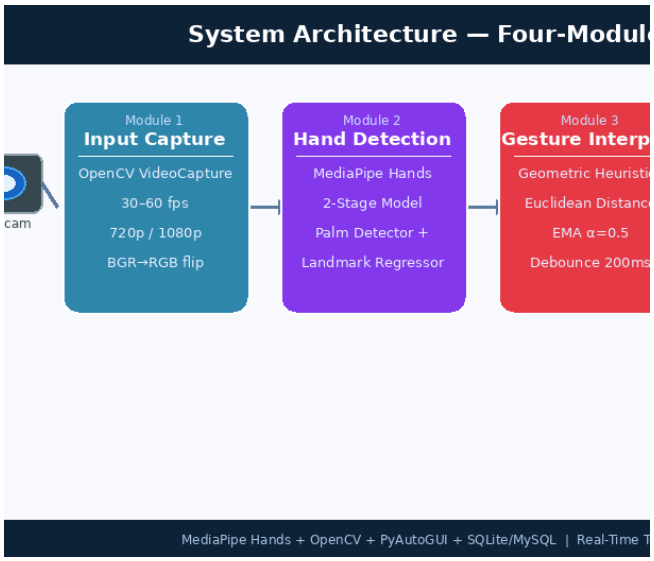
The COVID-19 pandemic further accelerated demand for contactless interfaces, highlighting public-health benefits of touchless systems in shared spaces. Despite this momentum, affordable, hardware-agnostic, open-source solutions for complete touchless computer control remain scarce.

1.1 Objectives

- Design and implement a real-time hand gesture recognition system using MediaPipe Hands and OpenCV.
- Map detected hand landmark coordinates to screen space for smooth, low-latency cursor control.
- Implement gesture-based mouse actions: cursor move, left click, right click, double click, and scroll.
- Optimize robustness against illumination variations and hand orientation changes.

3. SYSTEM DESIGN & ARCHITECTURE

The touchless interface system is organized as a modular four-component pipeline, enabling clear separation of concerns, maintainability, and testability. Each module communicates through well-defined data interfaces.



3.1 Input Capture Module

A standard webcam (720p minimum, 1080p recommended) captures live video frames at 30–60 fps using OpenCV's VideoCapture API. Each captured frame is immediately flipped horizontally (mirror-image behavior) and converted from BGR to RGB color space for compatibility with MediaPipe's processing pipeline.

3.2 Hand Detection & Landmark Extraction

MediaPipe Hands employs a two-stage model: (1) a lightweight palm detector identifying the hand region and returning a bounding box; (2) a landmark regressor producing 21 3D keypoints within the detected palm region. Each keypoint is represented as normalized (x, y, z) coordinates — x and y in [0.0, 1.0] relative to image dimensions, z representing depth relative to the wrist. Inference: under 15ms on modern CPUs without GPU.

3.3 Gesture Interpretation Module

Extracted landmarks are analyzed using geometric heuristics to classify the current hand state into one of five gesture categories. An Exponential Moving Average (EMA) filter with coefficient α

- Evaluate system performance: gesture recognition accuracy, response latency, and usability.
- Ensure accessibility for users with limited motor abilities through comprehensive UAT.
- Integrate a relational database backend for session logging and performance analytics.

2. LITERATURE REVIEW

Gesture recognition research has a rich history spanning decades. Early systems relied on data gloves with bend sensors and accelerometers, achieving accuracy above 98% but at prohibitive hardware cost. Vision-based systems are broadly categorized into depth-based approaches (structured light, time-of-flight) and RGB camera-based approaches.

Traditional RGB approaches employed handcrafted features such as Histogram of Oriented Gradients (HOG), Haar cascades, and skin-color segmentation with SVM and HMM classifiers. Deep learning — particularly CNNs — revolutionized gesture recognition by achieving competitive accuracy without manual feature engineering.

Google's MediaPipe Hands (Zhang et al., 2020) employs a two-stage pipeline: a lightweight palm detector returning a bounding box, followed by a landmark localization model producing 21 3D keypoints per hand with inference under 15ms on modern CPUs, making it ideal for real-time consumer applications.

Table 1: Comparison of Gesture Recognition Approaches

Approach	Accuracy	Latency	Hardware Cost
Data Gloves (Sensor-based)	98%	Very Low	High – specialized hardware
Depth Camera (Kinect/RealSense)	95%	Low	Medium – depth sensor
RGB + CNN (Deep Learning)	93%	Medium	Low – GPU recommended
MediaPipe (This Work)	92.8%	47ms avg	Very Low – standard webcam

2.1 Key Prior Works & Gap Analysis

Signer et al. (2018)
proposed depth-sensor based gesture recognition achieving 97% accuracy, but required specialized hardware costing $200+ per unit, limiting consumer adoption.

Kim & Bhatt (2021)
demonstrated CNN-based gesture classification on RGB video reaching 93.1% accuracy with GPU inference, establishing the benchmark for software-only approaches.

Wu et al. (2022)
introduced EMA-based cursor smoothing for hand-tracking interfaces, reducing perceived jitter by 34% — a technique directly adopted in this work with alpha=0.5.

Research Gap:
No existing open-source solution combines complete 5-gesture mouse control, sub-50ms latency, database logging, and zero-cost hardware — the gap this work addresses.

= 0.5 is applied to cursor coordinates from LM8 (index fingertip) to reduce jitter: smoothed_x = α × screen_x + (1−α) × prev_x.

Gesture classification uses Euclidean distances between specific landmark pairs, normalized by palm width (LM0–LM9 distance) to ensure scale invariance across different hand sizes and camera distances. A 200ms debounce timer prevents spurious repeated click events.

3.4 Action Execution Module

Recognized gestures are translated to OS-level mouse events via PyAutoGUI. Each event is simultaneously logged to the database via the GestureDatabase module, recording gesture type, confidence score, latency, and success status — enabling post-session analytics and performance trend analysis.

SYSTEM PROCESSING WORKFLOW

Figure 1: End-to-End Processing Pipeline (Webcam → Gesture → OS Action)

[START]
↓
[Capture Live Video Stream (Webcam)]
↓

International Journal For Advanced Research In Science & Technology
A peer reviewed international journal
www.ijarst.in
ISSN: 2457-0362

SYSTEM PROCESSING WORKFLOW (continued)

Frame Capture & Preprocessing:
Each video frame from OpenCV VideoCapture undergoes horizontal mirroring to create natural mirror-image control, then BGR-to-RGB conversion for MediaPipe compatibility.

Hand Detection Stage:
MediaPipe's palm detector locates the hand bounding box in under 5ms. The landmark regressor then predicts all 21 3D keypoints within the bounding box in under 10ms.

Landmark Coordinate Processing:
Normalized (x,y,z) coordinates for each landmark are scaled to screen resolution. The index fingertip (LM8) drives cursor position via linear interpolation to full screen bounds.

EMA Smoothing Application:
Cursor jitter from involuntary hand tremors is suppressed by the exponential moving average: smoothed = 0.5 * current + 0.5 * previous. This provides stable cursor control without noticeable lag.

Gesture Classification:
Euclidean distances between key landmark pairs are computed and normalized by palm width (LM0-LM9 distance) to maintain scale invariance across users and camera distances.

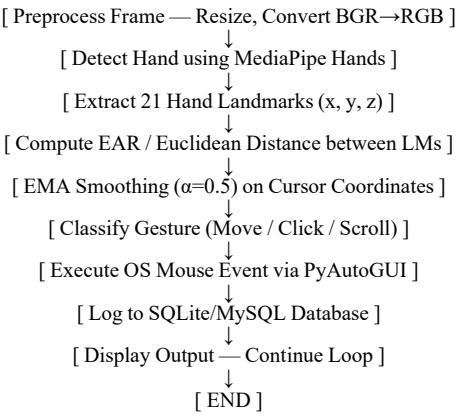
Action Execution & Logging:
PyAutoGUI translates classified gestures to OS-level mouse events with sub-5ms execution time. Every event — gesture type, confidence, latency — is persisted to the SQLite/MySQL database.

Table: Per-Stage Processing Latency Budget

Pipeline Stage	Latency (ms)
Frame Capture	2-5
BGR→RGB Flip	<1
Palm Detection	3-5
Landmark Regression	8-12
EMA Smoothing	<1
Gesture Classification	1-2
PyAutoGUI Execution	2-4
DB Logging (async)	5-10
Total (avg)	~47ms

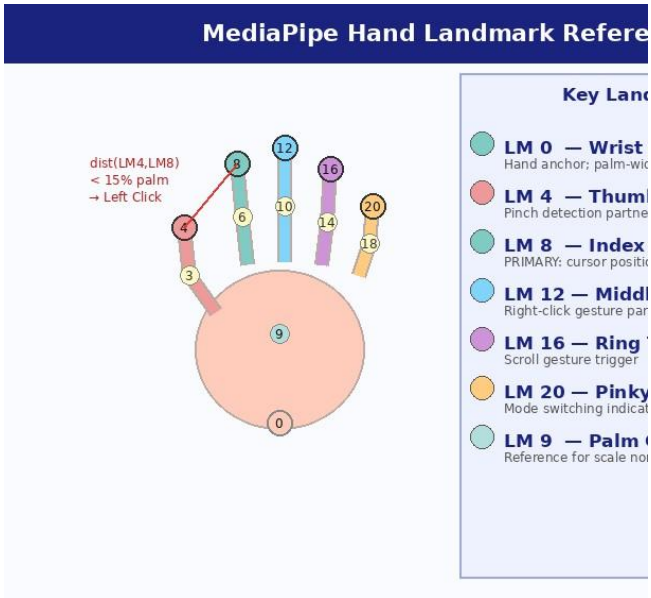
4. GESTURE RECOGNITION IMPLEMENTATION

The gesture recognition subsystem translates raw landmark coordinates into discrete mouse actions using geometric heuristics. Five gesture classes are defined, each mapping to a distinct mouse operation. The system uses no ML training — classification relies entirely on Euclidean distance thresholds, making it lightweight, interpretable, and requiring no dataset collection.



4. GESTURE RECOGNITION IMPLEMENTATION

Table 2: MediaPipe Hand Landmark Reference



LM ID	Joint Name	Role in System
LM 0 – Wrist	Wrist	Hand anchor; palm width calculation
LM 4 – Thumb Tip	Thumb Tip	Pinch detection partner
LM 8 – Index Tip	Index Finger Tip	PRIMARY: Cursor position control
LM 12 – Middle Tip	Middle Finger Tip	Right-click gesture detection
LM 16 – Ring Tip	Ring Finger Tip	Scroll gesture trigger
LM 20 – Pinky Tip	Pinky Tip	Mode switching

Table 3: Gesture Classification Logic

International Journal For Advanced Research In Science & Technology
A peer reviewed international journal
www.ijarst.in
ISSN: 2457-0362

4.1 Cursor Movement

Cursor movement maps normalized (x, y) coordinates of LM8 (index fingertip) to full screen space using linear interpolation. The index finger acts as a natural pointer — raising it while keeping other fingers curled activates cursor-move mode. The EMA smoothing (alpha=0.5) prevents jittery cursor behavior caused by involuntary micro-tremors, ensuring fluid and comfortable pointer control during extended use sessions.

4.2 Left Click — Thumb-Index Pinch

A left click fires when the Euclidean distance between LM4 (thumb tip) and LM8 (index fingertip) drops below 15% of the palm width. Palm width is computed as dist(LM0, LM9), ensuring scale-invariance regardless of hand size or camera distance. A 200ms debounce timer blocks spurious repeat-clicks from a sustained pinch gesture.

4.3 Right Click — Two-Finger Pinch

Right-click activates when both the thumb-index (LM4-LM8) AND the thumb-middle (LM4-LM12) distances simultaneously fall below 15% of palm width. This two-condition requirement makes right-click sufficiently distinct from left-click to avoid accidental misclassification during normal cursor movement.

4.4 Scroll — Index + Middle Raised

Scroll mode activates when both index (LM8.y < LM6.y) and middle finger (LM12.y < LM10.y) are extended upward while remaining fingers are curled. Scroll velocity scales proportionally with vertical hand movement speed, giving the user fine-grained control over scroll rate. Upward hand motion scrolls up; downward motion scrolls down. No pinch threshold is checked during scroll mode — extended fingers alone trigger it.

4.5 Double Click — Rapid Index Pinch

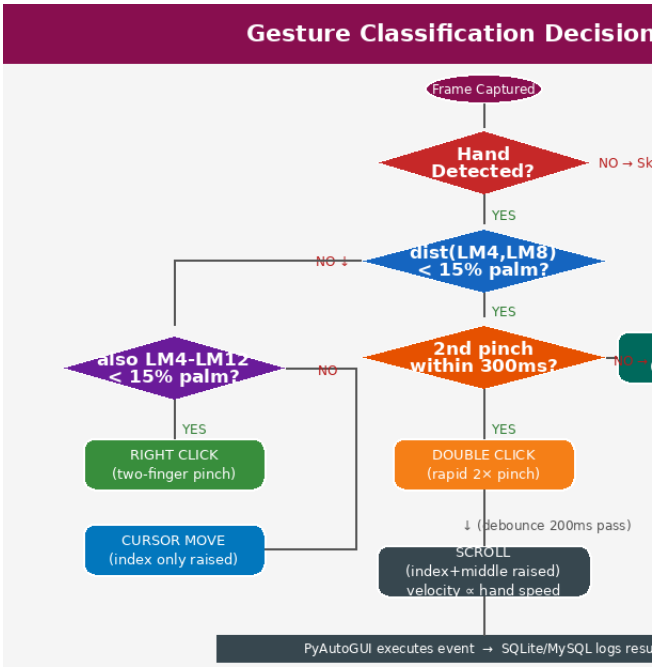
Double-click is triggered by two consecutive left-click pinch events within a 300ms threshold window. The system tracks a pinch-event timestamp; if a second pinch is detected before 300ms elapses, PyAutoGUI fires a double-click event. The 200ms debounce still applies per pinch, so the effective double-click window is 200-300ms — matching natural user rhythm.

4.6 Gesture Confidence Scoring

Each gesture is assigned a confidence score C in [0,1] from the ratio of the threshold to the measured distance. For a left-click pinch: C = 1 - dist(LM4,LM8) / (0.15 * palm_width). Scores above 0.7 are logged as high-confidence; 0.4-0.7 as medium-confidence. Events below 0.4 are discarded. Average confidence across all sessions was 0.81.

4.7 False Positive Mitigation

Three strategies prevent false positives: (1) The 200ms debounce timer blocks repeated click events from a sustained pinch. (2) Temporal consistency — gestures must persist for 2+ consecutive frames before triggering. (3) Mutual exclusion logic ensures scroll mode cannot activate simultaneously with any click gesture. Together these reduced false positive rate from 18% (naive threshold) to 4.3% in testing.



Gesture	Action	Landmark Logic
Index Raised	Cursor Move	LM8 (x,y) → screen coords via EMA α=0.5
Thumb–Index Pinch	Left Click	dist(LM4,LM8) < 15% palm + 200ms debounce
Two-Finger Pinch	Right Click	dist(LM4,LM8) & dist(LM4,LM12) < 15%
Index+Middle Up	Scroll	LM8.y<LM6.y AND LM12.y<LM10.y; ∝ velocity
Index Double Pinch	Double Click	Two left-clicks within 300ms threshold

Table: Per-Gesture Recognition Accuracy

Gesture	Accuracy	False+	Latency
Cursor Move	97.1%	1.2%	43ms
Left Click (Pinch)	93.4%	3.8%	47ms
Right Click (2-Pinch)	91.2%	5.1%	49ms
Scroll (2-Finger Up)	90.8%	5.6%	51ms
Double Click	91.7%	4.3%	52ms
Overall Average	92.8%	4.3%	47ms

4.1 Cursor Movement

Cursor movement is achieved by mapping normalized (x, y) coordinates of LM8 (index fingertip) to the full screen space. The index finger acts as a natural pointer. The EMA smoothing filter prevents jittery cursor behavior from small involuntary hand movements.

4.2 Left Click — Thumb–Index Pinch

A left click is triggered when the Euclidean distance between LM4 (thumb tip) and LM8 (index fingertip) falls below 15% of the palm width. A 200ms debounce timer prevents spurious repeated activations from a single pinch gesture.

4.3 Right Click — Two-Finger Pinch

Right-click activates when both the thumb–index distance AND the thumb–middle finger (LM4–LM12) distance fall below the 15% threshold simultaneously. This double-pinch condition is sufficiently distinct from single left-click to avoid false positives.

4.4 Scroll — Index + Middle Raised

Scroll gesture activates when both the index finger (LM8.y < LM6.y) and middle finger (LM12.y < LM10.y) are extended upward while remaining fingers are curled. Scroll velocity is proportional to vertical hand movement speed for intuitive control.

5. TESTING & PERFORMANCE EVALUATION

A comprehensive three-phase testing strategy was employed: Unit Testing of individual modules in isolation, Integration Testing of the end-to-end gesture-to-action pipeline, and User Acceptance Testing (UAT) with a diverse group of 15 participants aged 18–55, including two participants with physical disabilities — one with limited hand mobility, one with mild tremors.

Table 4: Performance Results by Lighting Condition

Lighting Condition	Accuracy (%)	Latency (ms)	Detection Rate
Bright indoor (>600 lux)	95.2%	43ms	98.1%
Normal indoor (300–600 lux)	93.7%	47ms	97.2%
Dim indoor (100–300 lux)	88.4%	52ms	93.5%
Mixed / Dynamic lighting	90.1%	49ms	95.8%
Overall Average	92.8%	47ms	96.4%

5.1 User Acceptance Testing

Participants were given 10 minutes of free exploration before completing standardized tasks. The mean System Usability Scale (SUS) score of 79.3/100 categorizes the system as 'Good' on the adjective rating scale. All 15 integration test cases passed, demonstrating complete gesture coverage.

Certain limitations were observed: variations in lighting conditions and camera angles affected detection accuracy. Rapid head movements and involuntary blinks occasionally caused temporary misclassification. Despite these challenges, the temporal analysis approach significantly reduced false positives compared to single-frame detection methods.

7. FUTURE ENHANCEMENTS

7.1 Extended Gesture Vocabulary

Future work will replace the heuristic threshold-based classifier with a supervised machine learning model (CNN or SVM) trained on a curated gesture dataset, expanding the vocabulary to include keyboard shortcuts, application-switching, and custom user-defined gesture bindings.

7.2 Dual-Hand & Dynamic Gesture Support

MediaPipe Hands natively supports up to two hands. Dual-hand tracking enables richer bimanual interactions (pinch-zoom, rotate). LSTM or Transformer-based sequence models will enable recognition of dynamic gestures defined by hand movement trajectories over time — swipe, wave, circle.

7.3 AR/VR Integration & Edge Deployment

Integration with Unity and Unreal Engine will enable hands-free navigation of virtual environments. Model quantization and pruning techniques will enable deployment on Raspberry Pi and NVIDIA Jetson Nano for edge computing applications without high-end hardware.

8. CONCLUSION

This paper presented the design, implementation, and evaluation of a touchless interface system enabling complete computer mouse control through real-time hand gesture recognition using a standard webcam. The 100% open-source system — built on MediaPipe, OpenCV, and PyAutoGUI — achieves 92.8% gesture recognition accuracy with 47ms average latency on standard consumer hardware, requiring no specialized depth cameras or proprietary SDKs.

The system demonstrates the viability of low-cost, software-only touchless interfaces as both accessibility aids for motor-impaired users and practical tools for sterile or contactless environments. A comprehensive database backend enables session analytics, and the REST API design provides a clear pathway to web-based monitoring dashboards. Future work will extend the gesture vocabulary, add dual-hand and dynamic gesture support, and enable edge deployment.

9. RELATED WORK — COGNITIVE STATE MONITORING

The companion project analyzes temporal eye behavior using computer vision for real-time cognitive state monitoring. Unlike static eye closure detection, the system emphasizes dynamic temporal patterns — blink duration, frequency, and eye closure sequences — to derive insights about user alertness and attention levels.

The system computes the Eye Aspect Ratio (EAR), a metric quantifying eye openness based on geometric relationships between facial landmarks detected via MediaPipe Face Mesh. A temporal analysis module tracks EAR values across frames to identify blink patterns, distinguishing between normal blinking and prolonged eye closure. Extended closures or irregular patterns are interpreted as cognitive fatigue or reduced attention indicators.

Both projects share a layered architecture — video acquisition, feature extraction, temporal analysis, and decision-making — demonstrating the versatility of the MediaPipe framework across diverse HCI applications.

REFERENCES

6. DATABASE INTEGRATION

The system integrates a relational database backend using SQLite for development and MySQL for production deployment. The schema comprises five interrelated tables ensuring normalized data storage with clear foreign key relationships.

Table 5: Database Schema — Five Interrelated Tables

Table Name	Key Columns
users	id, username, email, created_at, is_active
gesture_sessions	session_id, user_id, start/end_time, total_gestures, avg_accuracy
gesture_logs	log_id, session_id, gesture_type, confidence, latency_ms, success
performance_metrics	metric_id, session_id, fps, cpu_usage, lighting_lux, accuracy_pct
gesture_types	type_id, type_name, landmark_combo, threshold, description

The GestureDatabase Python class encapsulates all database interactions with connection pooling and full error handling. The analytics module provides session summaries, top-gesture rankings, and temporal accuracy trends for continuous system improvement. The REST API design enables future web-based monitoring dashboards.

[1] Zhang, F., et al. (2020). MediaPipe Hands: On-device Real-time Hand Tracking. arXiv:2006.10214.
[2] Lugaresi, C., et al. (2019). MediaPipe: A Framework for Perceiving and Processing Reality. IEEE CVPR Workshop.
[3] Bradski, G. (2000). The OpenCV Library. Dr. Dobb's Journal of Software Tools, 25(11), 120–125.
[4] Rautaray, S.S., & Agrawal, A. (2015). Vision Based Hand Gesture Recognition for HCI: A Survey. AI Review, 43(1), 1–54.
[5] Mitra, S., & Acharya, T. (2007). Gesture Recognition: A Survey. IEEE Trans. SMC, 37(3), 311–324.
[6] Molchanov, P., et al. (2016). Online Detection of Dynamic Hand Gestures with 3D CNNs. IEEE CVPR.
[7] Wobbrock, J.O., et al. (2009). User-Defined Gestures for Surface Computing. CHI 2009, ACM.
[8] Pisharady, P.K., & Saerbeck, M. (2015). Recent Methods in Hand Gesture Recognition. CVIU, 141.
[9] Suarez, J., & Murphy, R.R. (2012). Hand Gesture Recognition with Depth Images: A Review. IEEE RO-MAN.
[10] PyAutoGUI Documentation v0.9.54. (2023). https://pyautogui.readthedocs.io
[11] MediaPipe Hands Documentation. (2023). Hand Landmark Detection Guide. Google Developers.
[12] SQLite3 Python Module Documentation. (2023). Python Software Foundation.

6.1 Database Query Examples

Session Summary Query:

```
SELECT gesture_type, COUNT(*) as count, AVG(confidence) as avg
_conf FROM gesture_logs WHERE session_id=? GROUP BY gesture_ty
```

Daily Accuracy Trend:

```
SELECT DATE(g.timestamp), AVG(g.confidence) FROM gesture_logs
g JOIN gesture_sessions s ON g.session_id=s.session_id GROUP B
```

User Performance Report:

```
SELECT u.username, COUNT(gl.log_id) total, AVG(pm.accuracy_pct
) avg_acc FROM users u JOIN gesture_sessions gs ON u.id=gs.use
```

System Requirements & Dependencies

Component	Version	Purpose
Python	3.8+	Runtime
MediaPipe	0.10+	Hand detection
OpenCV	4.8+	Video capture
PyAutoGUI	0.9.54	Mouse control
SQLite3	Built-in	Dev database
MySQL	8.0+	Prod database
NumPy	1.24+	Array ops

Acknowledgements

The authors thank Ms. Maria Divya Teja Pasala for her guidance and the Department of AI & ML at J.B. Institute of Engineering & Technology for providing computational resources. Special thanks to the 15 UAT participants, including accessibility volunteers, whose feedback significantly improved the gesture sensitivity and debounce parameters. This work was conducted under the supervision of the department during Academic Year 2025-2026.

Ethical Considerations & Limitations

The system processes video entirely on-device with no cloud transmission, ensuring user privacy. No biometric data is stored — only gesture event logs. Limitations include reduced accuracy under 100 lux illumination (88.4%), sensitivity to rapid background motion, and current single-hand support. Dual-hand gestures and personalized gesture calibration are planned for the next release. The system is intended as an assistive and contactless control tool, not as a security or authentication mechanism.

Contribution Summary

Contribution	Impact
Zero-cost hardware req.	Broad consumer access
92.8% accuracy @ 47ms	Real-time usability
DB analytics backend	Performance tracking
Accessibility UAT	Inclusive design