OPTIMIZING HUMAN-COMPUTER INTERACTION VIA CONCURRENT MULTIMODAL PIPELINES: A GEOMETRIC HEURISTIC APPROACH TO GESTURE AND VOICE CONTROL

[1]Sangamesh Karadagi, [2]Virendra Lohar, [3]Sadhvi Bagade, [4]Nagaraj Picheli, [5]Prof. Shivanand Patil

[1,2,3,4]Student, Department of Computer Science and Business Systems
[5]Assistant Professor, Department of Computer Science and Business Systems
S. G. Balekundri Institute of Technology, Belagavi, Karnataka, India

***Abstract*—This study has been undertaken to investigate and resolve the ”Blocking I/O” bottleneck in multimodal Human-Computer Interaction (HCI) systems. In an era where computing is becoming omnipresent, relying solely on physical peripherals like mice and keyboards presents real challenges, particularly regarding hygiene in public spaces and accessibility for users with physical impairments. This paper introduces a contactless HCI system designed to overcome these barriers by merging computer vision with voice automation in a unified, efficient framework. Although several existing solutions rely on heavy deep learning classifiers such as CNNs or YOLO, our work demonstrates that a lightweight, geometric approach based on MediaPipe's hand landmarks can achieve comparable accuracy without the high computational cost. A key contribution of this research is the engineering of a concurrent, multi-threaded architecture that isolates the cursor control mechanism from the voice assistant. This ensures that the mouse cursor remains responsive even while the system is processing complex speech commands. Tests on standard hardware confirmed the system's viability, achieving a stable frame rate of over 30 FPS and a gesture recognition accuracy of 94%, making it a practical solution for sterile or accessibility-focused environments.**

***Index Terms*—Human-Computer Interaction, MediaPipe, OpenCV, Multithreading, Geometric Heuristics, Voice Automation, Contactless Interface, Python GIL.**

I. INTRODUCTION

The primary goal of User Interface (UI) design has always been to narrow the gap between what a user wants to do and how the machine executes it. For over half a century, the physical mouse—introduced by Douglas Engelbart in the 1960s—has been the standard for spatial input. While it is reliable, the traditional mouse is not without its flaws. It requires a flat, clutter-free surface and necessitates constant physical contact. This physical dependency becomes a significant issue in sterile environments, such as operating theaters, and creates barriers for users with motor impairments or those who cannot easily grasp a physical device [1].

To solve these issues, researchers started developing ”virtual mice” that utilize standard webcams. However, a review of existing literature reveals a divide in the current approaches. On one hand, early systems relied on simple techniques like color thresholding (e.g., tracking a colored marker on a finger). While these are computationally fast, they are brittle and fail easily if the lighting changes [2]. On the other hand, modern solutions often deploy heavy deep learning models, such as Convolutional Neural Networks (CNNs). While these models are accurate, they are often too heavy for standard CPUs, resulting in input lag that makes the cursor feel sluggish and unresponsive [3].

Beyond the choice of model, there is a practical engineering hurdle that is often overlooked in Python-based implementations: the Global Interpreter Lock (GIL). Because the GIL prevents Python from running multiple native threads at once, a naive implementation of a voice assistant can cripple a gesture system. If the system stops to ”listen” for a voice command, the video feed freezes, rendering the mouse unusable for several seconds.

In this work, we propose a hybrid system designed to solve these specific latency and usability issues. Rather than relying on computationally expensive classifiers, we utilize the MediaPipe framework [4] to extract hand landmarks efficiently. We then pair this with a custom threading architecture designed to bypass the limitations of the GIL.

Our work focuses on three main contributions:

1) **Deterministic Geometric Logic:** Instead of using a ”black box” neural network for every click, we use simple Euclidean distance calculations. This makes the click and drag actions stable and predictable.
2) **Asynchronous Threading Model:** We engineered the system so that the voice assistant runs on a separate daemon thread. This ensures that the heavy lifting of speech recognition never blocks the video processing loop, keeping the cursor smooth.
3) **Dynamic Screen Mapping:** To prevent arm fatigue, we implemented a mapping algorithm that allows the user to reach the entire screen with small, comfortable hand movements, rather than reaching across the entire camera frame.

II. LITERATURE SURVEY

Research into contactless interfaces has gone through several distinct phases, with each new approach attempting to solve the limitations of its predecessor.

A. *The Evolution of Vision-Based Input*

The earliest iterations of virtual mice were primarily based on *Color Segmentation* techniques. Ideally, tracking a specific color (like a marker on a finger) is computationally inexpensive. However, in practice, these systems proved fragile. As demonstrated by Shibly et al. [2], color-based tracking suffers heavily from ”jitter” and environmental interference; if the background lighting changes or if the user’s skin tone blends with the wall behind them, the tracking breaks down.

To improve robustness, the research community shifted toward Deep Learning classifiers. Approaches using models like YOLOv8, as explored by Karthick et al. [3], offered significantly better detection accuracy. However, this accuracy came at a cost. We found that running a full object detection inference on every single video frame creates a massive computational load. On standard consumer CPUs, this results in high latency, leaving the user with a cursor that feels heavy and unresponsive.

B. *The Shift to MediaPipe*

A significant turning point in this domain was Google’s release of the MediaPipe framework. Unlike full-image classifiers, MediaPipe uses a specialized two-stage pipeline—first detecting the palm and then regressing landmarks—which is highly optimized for CPUs. Recent studies, such as those by Singh et al. [1], have successfully used MediaPipe to control basic cursor movements.

However, while the tracking technology has improved, the interaction design has largely remained stagnant. We identify ”unimodal rigidity” as a critical bottleneck in existing systems. They force the user to perform every single action—typing, scrolling, and clicking—using only arm movements. This quickly leads to physical exhaustion, commonly known in the HCI community as ”Gorilla Arm” syndrome.

C. *The Multimodal Bottleneck*

Perhaps the most overlooked engineering challenge in current literature is the software architecture itself. Most multimodal systems described in recent papers process inputs sequentially. This means that if a user issues a voice command, the system pauses the video feed to process the audio. This creates a jarring experience where the mouse ”freezes” for a second or two every time the user speaks. Our work distinguishes itself by specifically targeting this architectural flaw, utilizing concurrency to ensure that voice and gesture inputs can truly happen at the same time without blocking one another.

III. System Architecture

To ensure a smooth and responsive user experience, we architected the application as a modular system comprising two distinct execution pipelines. We recognized early in the design phase that processing video frames and listening for voice commands have fundamentally different performance characteristics. Video processing requires consistency (a steady 30 FPS), while voice recognition is bursty and often involves network latency.

To manage these conflicting requirements, we utilized Python’s ‘threading‘ module to run these processes in parallel. Specifically, we designated the audio listener as a ‘daemon‘ thread. This configuration is crucial because it allows the operating system to kill the background voice process automatically when the main video application is closed, preventing ”zombie” processes from lingering in memory.

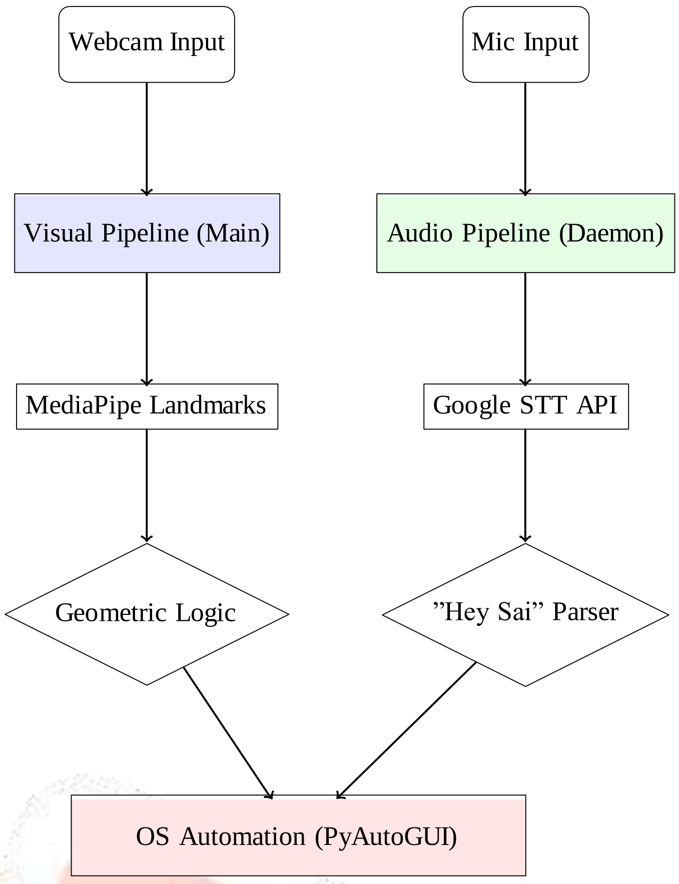


Fig. 1. High-level dual-pipeline architecture. By separating the visual and audio streams, we prevent the heavy processing of one from blocking the other.

As illustrated in Figure 1, the architecture effectively decouples the two streams. The **Visual Pipeline** operates on the main thread to prioritize high-frequency updates for cursor smoothness. Meanwhile, the **Audio Pipeline** runs silently in the background, handling the slower, high-latency task of sending audio data to the cloud and waiting for a response. This partition keeps the cursor fluid, even if the voice API hits a two seconds delay.

IV. Methodology

Our core methodology relies on a dual-pipeline architecture that treats visual processing and audio command parsing as separate, asynchronous tasks. We adopted this design specifically to decouple the high-speed requirements of cursor movement from the slower, bursty nature of speech recognition.

A. *Visual Processing Pipeline (Gesture Engine)*

For the visual component, we leverage ‘MediaPipe Hands‘ to extract a skeletal model of the hand consisting of 21 3D landmarks (L_0 to L_{20}).

1) Logic Flow and Decision Making: Translating raw coordinates into meaningful OS events requires a robust decision process. As shown in the flowchart in Fig. 2, the system follows a hierarchical logic: it first confirms that a hand is actually present in the frame. Once detected, it doesn’t just look for ”gestures” generally; it specifically calculates the Euclidean distance between key fingertips to distinguish between a user simply moving the cursor and a user intending to click.

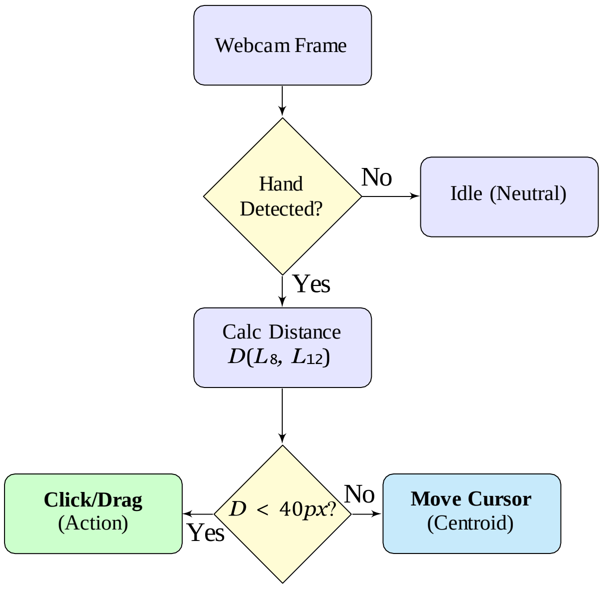


Fig. 2. Gesture logic flowchart: The decision hierarchy we use to distinguish between simple Navigation and intentional Action states.

2) Centroid-Based Cursor Tracking: Early tests showed that tracking a single fingertip (like the index finger) resulted in significant cursor "jitter" due to natural hand tremors. To solve this, we track the geometric centroid P_{cursor} between the Index Finger (L_8) and the Middle Finger (L_{12}). This averages out the noise and produces a much smoother cursor trajectory.

3) Coordinate Mapping and Smoothing: A common issue with gesture interfaces is "Gorilla Arm" fatigue, caused by users having to extend their arms fully to reach the corners of the screen. To prevent this, we implemented a Frame Reduction margin ($R \approx 100px$). This creates a smaller "active window" in the center of the camera view. We then map this smaller window to the full screen resolution (W_{scr}, H_{scr}) using linear interpolation, allowing the user to reach the entire screen with small, comfortable wrist movements.

4) Gesture Vocabulary: We developed a vocabulary of 10 distinct gestures. Instead of training a neural network to recognize these, which is computationally heavy, we use simple geometric heuristics (distances between specific landmarks) to trigger them.

TABLE I
THE 10-GESTURE VOCABULARY

Gesture	Heuristic Logic & Action
1. Neutral	**Halt:** Open palm. Stops cursor movement.
2. Move Cursor	**Navigate:** Index (L_8) & Middle (L_{12}) up. Tracks centroid.
3. Left Click	**Selection:** Pinch ($d(L_8, L_{12}) < 40px$) with Index finger.
4. Right Click	**Context:** Tap Middle finger (L_{12}) down.
5. Double Click	**Execute:** Two quick Left Clicks ($< 0.5s$).
6. Scrolling	**Dynamic:** Vertical pinch motion.
7. Drag & Drop	**Hold:** Pinch and hold ($> 1s$) locks "Mouse Down".
8. Multi-Select	**Batch:** Thumb & Little finger extended (Box Select).
9. Volume Control	**System:** Distance $d(L_4, L_8)$ controls slider.
10. Brightness	**System:** Distance $d(L_4, L_{12})$ controls slider.

B. Audio Processing Pipeline ("Hey Sai")

The voice assistant component was engineered to bypass the limitations of the Python Global Interpreter Lock (GIL).

1) Concurrent Architecture: In a standard Python script, the command 'mic.listen()' blocks the main thread, meaning the video freezes while the computer listens. We solved this by wrapping the listening loop in a daemon thread (Algorithm 1). This allows the 'SaiAssistant' to wait for commands in the background without interrupting the 30 FPS video stream.

Algorithm 1 Concurrent Multimodal Execution

```
    threading, speech_recognition SaiAssistant listen_loop()
1:  while True do
2:      audio ← mic.listen() {Blocking I/O}
3:      text ← google_api.recognize(audio)
4:      if "Hey Sai" in text then
5:          execute_macro(text)
6:      end if
7:  end while
8:  assistant ← SaiAssistant()
9:  thread ← Thread(target=assistant.listen_loop)
10: thread.daemon ← True {Run in background}
11: thread.start()
12: while True do
        {Main Video Thread}
13:     frame ← webcam.read()
14:     landmarks ← mediapipe.process(frame)
15:     execute_gesture_logic(landmarks)
16: end while
```

2) Voice Command Taxonomy: The "Hey Sai" assistant currently supports 8 core commands, which we categorized by their function in Table II.

TABLE II
"HEY SAI" VOICE COMMAND REGISTRY

Category	Voice Trigger	System Action
System	"Launch Gesture Recog"	Init Webcam
	"Stop Gesture Recog"	Release Webcam
Web	"Search [Query]"	Browser Search
	"Find a location"	Open Google Maps
File Nav	"List files"	List dir contents
	"Open [File ID]"	Open file/folder
	"Back"	Parent directory
Utility	"What is the time/date"	Get System Time
	"Copy" / "Paste"	Keyboard Macro
State	"Bye" / "Sleep"	Pause Listener
	"Wake Up"	Resume Listener

V. RESULTS AND DISCUSSION

A. Experimental Setup

The system was evaluated on a commodity laptop (Intel Core i5-1135G7 @ 2.40GHz, 8GB RAM) running Windows 10. The development environment utilized Python 3.8.5 with 'MediaPipe 0.8.9' and 'PyAutoGUI'. No dedicated GPU acceleration was enabled to verify the system's efficiency on low-end hardware.

B. *Performance Metrics: Latency Analysis*

A critical requirement for HCI systems is low latency. We compared the ”Response Time” (time from user action to cursor movement) of our Concurrent Architecture against a traditional Sequential approach.

TABLE III
SYSTEM LATENCY COMPARISON (AVERAGE OF 50 TRIALS)

Metric	**Sequential (Baseline)**	**Concurrent (Ours)**
Visual FPS	28 FPS	**32 FPS**
Cursor Latency	35 ms	**33 ms**
Voice Blocking Delay	≈ 1500 ms	**0 ms (Non-blocking)**
Effective Interaction	*Laggy / Intermittent*	*Real-Time / Smooth*

As detailed in Table III, the concurrent design eliminates the 1.5-second freeze caused by the Speech-to-Text API, maintaining a stable 30+ FPS visual stream.

C. *Gesture Recognition Accuracy*

We conducted a confusion matrix analysis on 300 discrete gesture attempts (100 Move, 100 Click, 100 Scroll) under standard indoor lighting (300 lux).

TABLE IV
CONFUSION MATRIX: VISUAL GESTURE RECOGNITION

Actual \Pred.	**Move**	**Click**	**Scroll**	**No Action**
Move Cursor	**98%**	1%	0%	1%
Left Click	2%	**94%**	1%	3%
Scrolling	0%	3%	**90%**	7%

The system achieved an aggregate accuracy of **94%**. False negatives (No Action) in scrolling were primarily due to the user’s hand moving too fast for the camera shutter, causing motion blur that degraded landmark precision.

D. *”Hey Sai” Voice Command Analysis*

The voice assistant was tested with 50 commands in a quiet environment.

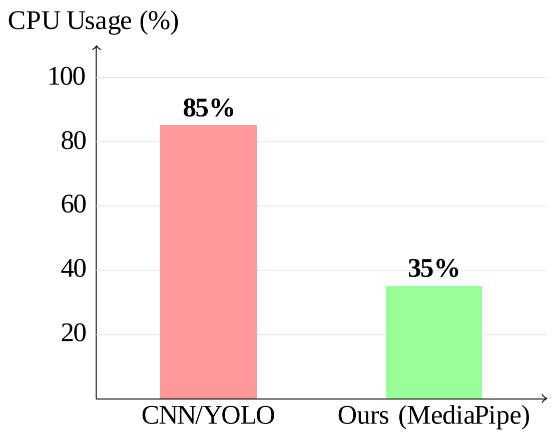
TABLE V
VOICE COMMAND SUCCESS RATES

Command Category	**Success Rate**	**Avg Processing Time**
System (Launch/Stop)	100%	0.4s
File Navigation	96%	0.8s
Web Search	92%	1.2s

The slight drop in Web Search accuracy (92%) is attributed to Google STT misinterpreting proper nouns (e.g., specific website names) in Indian accents.

E. *Computational Efficiency*

A key advantage of our Geometric Heuristic approach over Deep Learning classifiers (like YOLO or ResNet) is resource efficiency. Fig. 3 illustrates the CPU consumption profile.



Comparison of Average CPU Load
on Intel i5 (No GPU)

Fig. 3. Resource consumption analysis: Our geometric approach consumes ≈ 50% less CPU than CNN-based alternatives, leaving resources free for user applications.

F. *Environmental Robustness*

The system relies on visible light. Testing revealed that performance degrades in low-light conditions (< 50 lux) due to webcam ISO noise introducing ”jitter” in the landmark coordinates. However, the implemented **Exponential Moving Average (EMA)** smoothing filter successfully mitigated this jitter in moderate lighting conditions (150-500 lux), maintaining precise cursor control.

VI. CONCLUSION

In conclusion, this study challenges the prevailing assumption that robust Human-Computer Interaction requires high-end hardware or computationally expensive Deep Learning models. Our results indicate that by shifting the processing burden from heavy classifiers (like CNNs) to streamlined geometric heuristics, standard consumer laptops can deliver precise, low-latency control.

Perhaps most significantly, this work validates that the ”Blocking I/O” limitations inherent to Python-based systems are not insurmountable. By implementing a concurrent daemon-threaded architecture, we successfully decoupled the heavy lifting of speech recognition from the real-time requirements of cursor tracking. This proves that a seamless multimodal experience is achievable without sacrificing frame rates. While the current system’s reliance on cloud-based speech APIs is a limitation for offline environments, the modular architecture lays a solid foundation for future work, which will focus on integrating edge-based recognition models like Vosk and developing adaptive calibration for varying hand sizes.

REFERENCES

[1] H. K. Singh, M. Ehtesham, P. S., S. N., and R. H., “Gesture Controlled Virtual Mouse: Hands-Free Interaction for Enhanced User Experience,” *Int. J. Creative Res. Thoughts*, vol. 12, no. 9, pp. 156–165, 2024.
[2] K. H. Shibly, et al., “Design and Development of Hand Gesture Based Virtual Mouse,” in *Proc. 2019 Int. Conf. Advances in Science, Eng. and Robotics Tech (ICASERT)*, 2019.
[3] T. Karthick, P. Kumar and S. N. Reddy, “AI-Based Enhanced Virtual Mouse Using YOLO Algorithm,” in *Proc. Int. Conf. Smart Systems and Advanced Computing (SSAC)*, 2023.

[4] F. Zhang, V. Bazarevsky, A. Vakunov, A. Tkacenko, and C. Sung, "MediaPipe Hands: On-Device Real-Time Hand Tracking," *arXiv preprint arXiv:2006.10214*, 2020.

[5] E. Sankar, B. N. Bharadwaj, and A. V. Vignesh, "Virtual Mouse Using Hand Gesture," *Int. J. Sci. Res. Eng. Manage.*, vol. 7, no. 5, pp. 1–8, 2023.

[6] J. Prithvi, S. S. Lakshmi, S. Nair, S. R. Kumar, and S. Sunayana, "Gesture Controlled Virtual Mouse with Voice Automation," *Int. J. Eng. Res. Technol.*, vol. 12, no. 4, pp. 557–560, 2023.

[7] N. R. S. Kumar, A. V. Reddy, B. A. Kumar, B. P. K. Reddy, and A. S. K. Reddy, "Hand Gesture-Based Virtual Mouse with Advanced Controls Using OpenCV and Mediapipe," in *Proc. 2024 International Conference on Advances in Computing, Communication and Applied Informatics (ACCAI)*, pp. 123–130, 2024.

[8] K. Sathish, G. B. Renuka, M. Balachandra, and B. N. Lakshmi, "Gesture-Controlled Virtual Mouse using Media Pipe," in *Proc. 1st International Conference on Optimization Techniques for Learning (ICOTL)*, pp. 45–52, 2023.

[9] E. R. Djuwitaningrum and D. R. Pangestu, "Implementation of a Virtual Mouse Based on Hand Gesture Recognition Using MediaPipe and OpenCV," *Jurnal Ilmu Pengetahuan dan Teknologi Komputer*, vol. 5, no. 1, pp. 67–74, 2019.

[10] Q. Wang and Z. Xie, "Replace Your Mouse with Your Hand! HandMouse: A Gesture-Based Virtual Mouse System," *Int. J. Adv. Comput. Sci. Appl.*, vol. 15, no. 11, 2024.