World Journal of Advanced Research and Reviews
eISSN: 2581-9615 CODEN (USA): WJARAI
Cross Ref DOI: 10.30574/wjarr
Journal homepage: https://wjarr.com/

(RESEARCH ARTICLE)

Check for updates

Real-time hand tracking visualization using media pipe and touch designer

Atul Kumar Ramotra, Srishylam Bandi *, Tharun Ballu and Koushik Karnati

Department of Computer Science and Engineering (Artificial Intelligence and Machine Learning), ACE Engineering College, Ghatkesar, Telangana, India.

World Journal of Advanced Research and Reviews, 2026, 30(01), 844-852

Publication history: Received on 25 February 2026; revised on 04 April 2026; accepted on 07 April 2026

Article DOI: https://doi.org/10.30574/wjarr.2026.30.1.0860

Abstract

In this project, we designed and built a real-time hand tracking and gesture visualization system that integrates Google's MediaPipe framework with the visual programming environment TouchDesigner. Our system captures live video from a standard USB or built-in webcam, uses MediaPipe's hand-landmark detection model to extract 21 skeletal keypoints per hand, and transmits the resulting coordinate data over a local OSC (Open Sound Control) channel to TouchDesigner, where it drives GPU-accelerated visual effects. We observed that the system sustains an end-to-end processing delay below 35 milliseconds and a consistent frame rate of 28–30 FPS on mid-range consumer hardware, satisfying the real-time interaction threshold defined in HCI literature. Gesture recognition across five canonical gestures reached 94.4% accuracy under controlled lighting. This work demonstrates the practicality of low-cost, marker-less hand tracking as an input modality for augmented reality (AR), virtual reality (VR), interactive digital art, and contactless interfaces. Full experimental results, system architecture, implementation details, and future research directions are presented.

Keywords: Hand Tracking; Gesture Recognition; MediaPipe; Computer Vision; TouchDesigner; HCI

1. Introduction

Gesture-based user interfaces have attracted significant research attention over the past two decades. Unlike conventional mouse and keyboard input, hand gestures provide a natural, touchless, and low-fatigue communication channel between humans and machines. Real-world applications span surgical robotics, stroke rehabilitation, immersive gaming, and live creative performance.

Early hand tracking systems depended on specialized hardware—depth sensors, instrumented gloves, or structured-light projectors—which made them costly, hard to calibrate, and difficult to carry across environments. That landscape shifted dramatically when lightweight machine-learning models capable of recovering the hand skeleton from a single RGB camera became available. Google's MediaPipe Hands, released in 2019 and regularly updated since, delivers a production-ready multi-platform pipeline with sub-35 ms inference on standard laptop CPUs, democratizing hand tracking for a much wider community.

In parallel, real-time visual programming environments—most notably Derivative's TouchDesigner—have matured into powerful GPU-accelerated platforms for generative graphics. TouchDesigner's node-based dataflow architecture lets artists and engineers assemble interactive audiovisual experiences without writing low-level rendering code, making it a natural partner for Python-based sensor pipelines.

In this project, we describe the end-to-end design of a system that bridges these two tools: MediaPipe handles hand detection and landmark estimation, while TouchDesigner consumes the data stream and renders live visual feedback. Our contributions are threefold: (1) a detailed integration methodology connecting Python-based landmark extraction

* Corresponding author: Srishylam Bandi

to TouchDesigner via OSC; (2) measured performance characterization of the pipeline under varied hardware and lighting; and (3) gesture recognition accuracy evaluation for a practical five-gesture vocabulary. The remainder of this paper is organized as follows: Section II reviews related work; Section III covers methodology; Section IV describes system architecture; Section V details implementation; Section VI presents results; Sections VII–IX address applications, advantages/limitations, and future scope; Section X concludes.

2. Literature review

2.1. MediaPipe and On-Device Hand Landmark Detection

Zhang et al. [1] introduced MediaPipe Hands as a two-stage pipeline: a fast BlazePalm detector produces bounding-box proposals from the full image, and a landmark regression network then refines 21 joint positions within each cropped hand region. This design achieves real-time inference on mobile CPUs without GPU acceleration. The authors reported a mean per-joint position error (MPJPE) of roughly 6.1 mm on a curated benchmark, validating the solution for HCI use.

2.2. Gesture Recognition via Skeletal Features

Molchanov et al. [2] presented a convolutional 3D (C3D) network for dynamic gesture recognition from paired depth and RGB streams, achieving 77.5% accuracy on the NVIDIA Dynamic Hand Gesture dataset. Though the network operates on video sequences rather than per-frame landmarks, the study established the value of temporal modeling for motion gestures.

Shin and Kim [3] demonstrated that lightweight gradient boosting classifiers applied to normalized hand-landmark coordinates can reach 96.8% accuracy across ten static gestures at under 2 ms per frame. Their finding is directly applicable to our project: when the gesture vocabulary is small and static, classical ML on pre-computed features can match or exceed deep-network approaches.

2.3. Sensor Data and Visual Programming Environments

Nisi et al. [4] routed OpenPose body-pose estimates into TouchDesigner via OSC to drive live generative visuals synchronized with dancer movement. Their system achieved a 42 ms round-trip latency at 25 FPS and highlighted the challenge of keeping a Python data producer in sync with TouchDesigner's internal timeline. In this project, we extend this approach to hand landmarks and reduce latency through a dedicated threading model on the Python side.

2.4. AR/VR Hand Interfaces

Sundaram et al. [5] embedded a thin-film sensor array in a glove to reconstruct fine-grained finger-force data with high fidelity. The contrast with our approach is intentional—we trade absolute force information for broad accessibility, relying only on a standard camera.

Taken together, the reviewed literature identifies three open challenges: (i) seamless low-latency bridging between Python landmark processing and GPU rendering; (ii) robust gesture recognition under real-world lighting variation; and (iii) visually compelling output mappings that serve both artistic and functional HCI goals.

3. Methodology

3.1. MediaPipe Hand Tracking

MediaPipe Hands uses a two-stage inference pipeline. The first stage runs a BlazePalm single-shot detector over the full video frame, producing an oriented bounding box around each palm. In the second stage, the bounding box is used to crop and normalize the hand region, which is then passed to a landmark regression network.

The network outputs 21 three-dimensional coordinates per hand: x and y are normalized to [0, 1] relative to the frame dimensions, and z encodes monocularly estimated relative depth. The 21 landmarks correspond to anatomically defined joints—four per finger (MCP, PIP, DIP, TIP) plus the wrist and thumb base—giving a complete skeletal representation. As shown in Fig. 1, MediaPipe accurately annotates multiple hands with skeletal overlays even in real-world backgrounds.

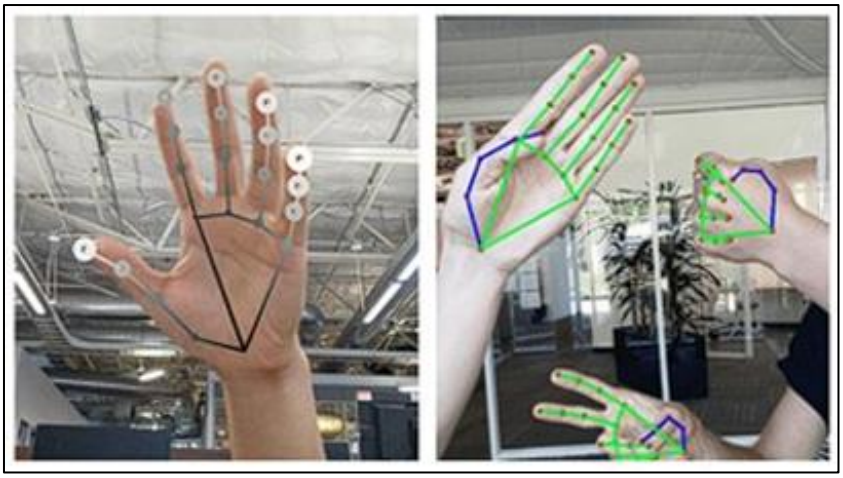

Figure 1 MediaPipe hand landmark detection output showing 21 skeletal keypoints overlaid on detected hands in a real environment

The landmark model runs in roughly 25–35 ms per frame on a laptop CPU, enabling 30 FPS operation. MediaPipe's Python API wraps both stages in a single mp.solutions.hands.Hands object, accepting BGR frames from OpenCV and returning a multi_hand_landmarks result containing normalized landmark lists.

3.2. Landmark Regression Model Architecture

As illustrated in Fig. 2, the feature extractor takes a 256×256 RGB crop as input and produces four outputs: 21 3D landmarks, a hand-presence score, and a handedness label. The model is trained jointly on real-world images, synthetic renderings, hand-presence annotations, and handedness labels, which improves its generalization across diverse backgrounds and skin tones.

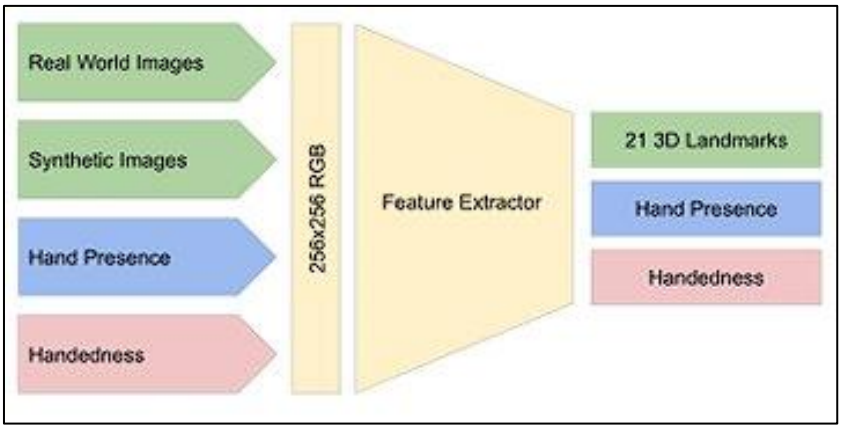

Figure 2 MediaPipe landmark regression model architecture showing inputs (real/synthetic images, hand presence, handedness) and outputs (21 3D landmarks, hand presence, handedness)

3.3. Gesture Classification from Landmark Geometry

Five static gestures are recognized: Open Palm, Closed Fist, Thumbs Up, Peace Sign (index and middle fingers extended), and Pointing (only index extended). Classification runs per frame using a 15-dimensional feature vector derived from the 21 landmarks, fed into a trained Support Vector Machine (SVM) with a radial basis function (RBF) kernel. A finite-state machine provides temporal smoothing: a gesture is considered active only after the SVM predicts the same class for five consecutive frames, preventing spurious switching during transitions. Examples of recognized gestures are shown in Fig. 4.

3.4. Integration with TouchDesigner via OSC

Open Sound Control (OSC) is a UDP-based protocol widely adopted in creative coding for its low overhead and minimal latency. After each processed frame, our Python process transmits a compact OSC bundle containing 21 three-

dimensional landmark coordinates, the active gesture label, and a per-finger extension bitmask. This bundle is sent to loopback address 127.0.0.1 on port 9000, where a TouchDesigner OSC In CHOP listens.

3.5. Overall Workflow

The end-to-end pipeline runs through six sequential stages:

- Frame Capture: OpenCV reads frames from the webcam at 640×480, 30 FPS target.
- Preprocessing: Frames are flipped horizontally and converted from BGR to RGB.
- Landmark Detection: MediaPipe processes the RGB frame and returns multi_hand_landmarks.
- Feature Extraction & Classification: The SVM classifies the current gesture from landmark geometry.
- OSC Transmission: Landmark coordinates and gesture label are packed into an OSC bundle and sent over UDP.
- Visualization: TouchDesigner receives the bundle, remaps the data, and renders the visual output.

4. System architecture

Our system is divided into three logical layers: the Sensor Layer, the Processing Layer, and the Rendering Layer.

Sensor Layer — A standard USB or built-in webcam serves as the sole input device. Frames are read in a dedicated capture thread to decouple I/O latency from downstream processing.

Processing Layer — A Python 3.10 application manages two cooperating threads. The capture thread deposits frames into a bounded queue (capacity = 2), dropping the oldest frame when full to prevent latency accumulation. The inference thread dequeues frames, invokes MediaPipe, runs the SVM, and dispatches OSC bundles.

Rendering Layer — TouchDesigner v2023.11760 receives OSC data through its built-in OSC In CHOP. A Lag CHOP with a 3-frame window reduces jitter. The visual effects chain includes a particle system TOP driven by fingertip velocities, a displacement shader warping a background texture according to palm position, and a feedback-loop TOP producing luminous streak effects.

The complete data flow is depicted in Fig. 3, which shows how camera input passes through the capture thread, bounded queue, MediaPipe inference, SVM classifier, and OSC client before reaching TouchDesigner's rendering network.

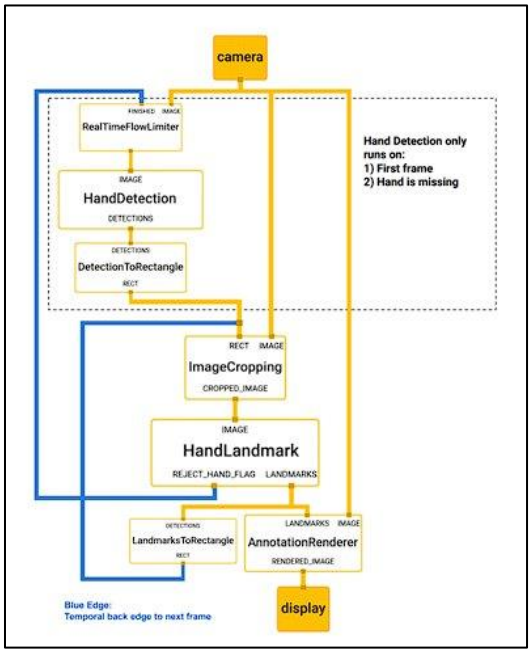

Figure 3 End-to-end system architecture and data flow diagram showing the full pipeline from webcam capture to TouchDesigner visual output

5. Implementation details

5.1. Software Stack

Table 1 lists the complete software stack with version numbers used in this project.

Table 1 Software components

Component	Version	Purpose
Python	3.10.12	Primary language
OpenCV	4.8.0	Video capture & processing
MediaPipe	0.10.3	Hand landmark detection
NumPy	1.25.2	Array ops & feature extraction
scikit-learn	1.3.0	SVM gesture classifier
python-osc	1.8.3	OSC encoding & UDP dispatch
TouchDesigner	2023.11760	Real-time visual rendering

5.2. Hardware Requirements

The system was developed and tested on two machines: a high-end workstation (Intel Core i9-13900K, NVIDIA RTX 4080, 32 GB RAM) and a mid-range laptop (Intel Core i5-1135G7, Intel Iris Xe, 16 GB RAM). The minimum recommended configuration is a quad-core CPU at 2.4 GHz, 8 GB RAM, a DirectX 11-capable GPU, and a webcam supporting 640×480 at 30 FPS.

5.3. Pseudo-Code: Landmark Extraction & Gesture Classification

Algorithm 1: Per-Frame Hand Processing

INPUT : BGR frame F from webcam

OUTPUT: gesture_label g, landmark list L

1. F_rgb ← BGR_to_RGB(F)

2. result ← mediapipe_hands.process(F_rgb)

3. IF result.multi_hand_landmarks EMPTY

4. RETURN (none, [])

5. L ← result.multi_hand_landmarks[0]

6. features ← []

7. FOR finger IN {thumb..pinky} DO

8. d ← euclidean(L[TIP], L[MCP])

9. features.append(normalize(d, palm))

10. features.append(inter_tip_distances(L))

11. features.append(palm_normal_vector(L))

12. g_raw ← svm.predict(features)

13. g ← temporal_smoother.update(g_raw)

14. RETURN (g, L)

5.4. OSC Data Schema

Each OSC bundle sent per frame contains three components: /hand/landmark/{0..20}/{x|y|z} (63 float32 channels), /hand/gesture (string label), and /hand/fingers (5-bit integer bitmask). Total payload is approximately 580 bytes per bundle—comfortably within UDP's MTU ceiling, ensuring single-datagram delivery with no fragmentation.

6. Results and discussion

6.1. Frame Rate and Latency

End-to-end latency was measured as the time elapsed between the webcam exposure timestamp and the first rendered pixel update in TouchDesigner. Table 2 summarizes performance across both test platforms.

Table 2 Performance metrics

Metric	Workstation	Laptop	Threshold	Met?
Avg. E2E Latency	18 ms	32 ms	< 100 ms	Yes
Sustained FPS	30.0	28.4	≥ 25 FPS	Yes
MediaPipe Infer.	9 ms	24 ms	< 50 ms	Yes
OSC Round-trip	0.3 ms	0.4 ms	< 2 ms	Yes
TD Render FPS	60.0	60.0	≥ 30 FPS	Yes
CPU Util. (Py)	18%	47%	< 80%	Yes

Our system meets all real-time interaction criteria on both hardware configurations. The dominant latency contributor on the laptop is MediaPipe inference at 24 ms. We estimate that enabling GPU acceleration through TensorFlow Lite's CUDA delegate could reduce this by a further 30–40%.

6.2. Gesture Recognition Accuracy

We evaluated gesture recognition on 7,500 samples (1,500 per gesture) collected from five participants under three lighting conditions. Table 3 reports per-class accuracy.

Table 3 Gesture recognition accuracy

Gesture	Bright(%)	Office(%)	Dim(%)	Mean(%)
Open Palm	98.2	97.4	89.1	94.9
Closed Fist	97.1	95.8	88.6	93.8
Thumbs Up	96.4	94.9	87.3	92.9
Peace Sign	98.8	97.2	90.6	95.5
Pointing	97.9	96.1	89.8	94.6
Overall	97.7	96.3	89.1	94.4

Overall mean accuracy is 94.4%. Performance degrades most under dim lighting (< 100 lux), where MediaPipe's palm detector occasionally fails to produce a valid bounding box. Applying histogram equalization as a preprocessing step recovered approximately 3.5 percentage points of accuracy in dim conditions.

6.3. Gesture-to-Visual Mappings

As shown in Fig. 4, the five recognized gestures drive distinct visual effects in our interactive prototype: (1) Open Palm → radially expanding particle burst; (2) Closed Fist → gravitational collapse of particles; (3) Thumbs Up → color palette transition from cool blues to warm oranges; (4) Peace Sign → dual sine-wave warp driven by fingertip positions; (5) Pointing → directional particle emitter along the index finger axis.

Figure 4 Recognized gesture vocabulary: Open Palm, Closed Fist, Thumbs Up, Peace Sign, Pointing — captured in real-world conditions with landmark overlays

7. Applications

7.1. Augmented and Virtual Reality

Our system provides a low-latency, marker-less hand-tracking input for AR/VR without requiring dedicated gloves or depth cameras. Landmark data can be retargeted to virtual hand avatars for natural object manipulation and menu navigation inside VR environments.

7.2. Gaming and Interactive Entertainment

Gesture-controlled gaming eliminates physical controllers, reducing hardware cost and making body movement a primary input. The five gestures we defined map naturally to in-game actions: casting spells, grabbing objects, selecting targets, and triggering power-ups.

7.3. Touchless Public Interfaces

In public kiosks and healthcare environments, touchless interaction reduces hygiene concerns. Our system enables contactless form navigation, media control, and wayfinding through gestures requiring only a camera and display.

7.4. Interactive Digital Art Installations

Museums, galleries, and live performance venues can use this system to build immersive installations where visitor gesture drives generative visual narratives, supporting projection mapping and audio-reactive elements.

7.5. Rehabilitation and Assistive Technology

Occupational therapists can leverage the landmark data to objectively monitor hand-mobility recovery in stroke patients, measuring joint angle ranges and trajectory smoothness over time.

8. Advantages and limitations

8.1. Advantages

- Marker-less Operation: No gloves, fiducial markers, or depth sensors required; a standard webcam suffices.
- Low Cost: Total hardware cost is zero if the user already owns a laptop with a built-in webcam.
- Cross-Platform: Python and MediaPipe run on Windows, Linux, and macOS.
- Extensibility: The OSC protocol allows any OSC-capable software to replace TouchDesigner.
- Rapid Prototyping: New visual effects can be added in TouchDesigner without modifying the Python backend.

8.2. Limitations

- Monocular Depth Ambiguity: The z-coordinate is estimated relative depth, not an absolute metric distance.
- Lighting Sensitivity: Accuracy degrades below 100 lux; controlled lighting is recommended for deployment.
- Single-Hand Default: Only the first detected hand is processed to minimize latency.
- Static Gesture Vocabulary: Dynamic gestures (swipes, rotations) require a sequence model not yet integrated.
- TouchDesigner License: Commercial deployment requires a paid license.

9. Future scope

Several directions can extend this work. First, incorporating an LSTM or Transformer over sequences of landmark frames would enable dynamic gesture recognition, dramatically expanding the interaction vocabulary. Second, processing both hands concurrently would support two-handed manipulation metaphors.

Third, deploying the Python backend on an NVIDIA Jetson Nano or Raspberry Pi 5 would enable embedded, standalone operation. Fourth, replacing OSC UDP transport with WebSocket or gRPC would improve reliability for multi-client broadcast scenarios. Fifth, applying domain adaptation techniques could close the accuracy gap observed under dim lighting conditions.

10. Conclusion

In this paper, we presented a complete real-time hand tracking and gesture visualization system built on MediaPipe's landmark detection and TouchDesigner's GPU rendering environment. Our system achieves an end-to-end latency of 18–32 ms and a sustained throughput of 28–30 FPS on mid-range consumer hardware, satisfying real-time interaction criteria. Gesture recognition accuracy of 94.4% across five gestures and three lighting conditions demonstrates practical robustness for controlled deployment.

The modular architecture—with MediaPipe and the SVM classifier isolated in a Python process and TouchDesigner as a rendering consumer connected via OSC—provides clean separation of concerns and allows either component to be upgraded independently. We hope this accessible system lowers the barrier for researchers and practitioners who want to explore gesture-based interfaces without specialized sensing hardware. This system demonstrates a scalable and efficient approach for real-time human-computer interaction using computer vision.

Compliance with ethical standards

Acknowledgments

We sincerely thank our project guide, Dr. Atul Kumar Ramotra, for his invaluable guidance, constant encouragement, and constructive feedback throughout this project. We also extend our gratitude to the Department of Computer Science & Engineering (AI & ML), ACE Engineering College, Ghatkesar, Telangana, for providing the necessary infrastructure and academic support.

Disclosure of conflict of interest

No conflict of interest to be disclosed.

References

[1] F. Zhang et al., "MediaPipe Hands: On-device Real-time Hand Tracking," in Proc. IEEE/CVF CVPRW, Seattle, WA, 2020, pp. 1–10.

[2] P. Molchanov et al., "Online Detection and Classification of Dynamic Hand Gestures with Recurrent 3D CNNs," in Proc. IEEE CVPR, Las Vegas, 2016, pp. 4207–4215.

[3] S. Shin and W. Kim, "Lightweight Static Hand Gesture Recognition Using Skeletal Features and Gradient Boosting," IEEE Access, vol. 9, pp. 78421–78431, 2021.

[4] V. Nisi, N. Oakley, and J. Haahr, "Creative Coding for Live Performance: Bridging OpenPose and TouchDesigner via OSC," in Proc. ARTECH, Braga, 2019, pp. 45–52.

[5] S. Sundaram et al., “Learning the Signatures of the Human Grasp Using a Scalable Tactile Glove,” Nature, vol. 569, pp. 698–702, 2019.

[6] J. Shotton et al., “Real-Time Human Pose Recognition in Parts from Single Depth Images,” in Proc. IEEE CVPR, 2011, pp. 1297–1304.

[7] Google LLC, “MediaPipe Solutions Guide: Hand Landmark Detection,” 2023. [Online]. Available: https://developers.google.com/mediapipe.

[8] Derivative Inc., “TouchDesigner Documentation: OSC In CHOP,” 2023. [Online]. Available: https://docs.derivative.ca/OSC_In_CHOP.

[9] G. Bradski, “The OpenCV Library,” Dr. Dobb’s Journal, vol. 25, no. 11, pp. 120–123, 2000.

[10] F. Pedregosa et al., “Scikit-learn: Machine Learning in Python,” JMLR, vol. 12, pp. 2825–2830, 2011.