Bharat AI-Soc Student Challenge

Problem Statement 2 – Technical Report

Real-Time Vision-Based Gesture Controlled Media System
On-Device Edge AI Inference Using ARM-Based Embedded Platform
(Jetson Nano)

Submitted in partial fulfillment of the competition requirements
Platform: NVIDIA JETSON NANO DEVELOPER KIT
Soc: NVIDIA Tegra X1 (Quad-core ARM Cortex-A57 CPU with 128-core NVIDIA Maxwell GPU)
Tools: NVIDIA JetPack SDK | Ubuntu Linux (ARM64) | Python 3 | OpenCV | MediaPipe | CUDA Toolkit | GStreamer
Date: 20/02/2026

<u>Team Details</u>

Team Leader
SATHISH KUMAR M
2rd Year | B.E. Electrical & Electronics Engineering
Team member
IRAIANBU R
2nd Year | B.E. Electronics & Communication Engineering
ARAVINTH VELMURUGAN D G
2nd Year | B.E. Electronics & Communication Engineering

Faculty Mentor
Dr. S.Aathilakshmi
Chennai Institute of Technology, Chennai

Abstract

The increasing demand for touchless interaction systems has accelerated the development of vision-based Human-Computer Interaction (HCI) solutions. This project presents a real-time touchless media control system using hand gestures implemented on the NVIDIA Jetson Nano embedded AI platform. The objective is to design a low-latency, high-accuracy gesture recognition system capable of controlling media playback without physical contact.

The system utilizes Media Pipe Hands for lightweight hand landmark detection and extracts 21 key hand points to classify gestures based on geometric relationships such as finger states, inter-finger distances, and hand positioning within the camera frame. Recognized gestures are mapped to media player controls including play/pause, volume adjustment, forward and backward seeking, mute, Fullscreen toggle, and subtitle control. The implementation supports MPV media players through keyboard event simulation.

To meet embedded system constraints, the solution is optimized for ARM architecture by reducing camera resolution, minimizing frame buffer delay, selecting lightweight model configurations, and implementing cooldown-based gesture triggering to prevent false activations. Performance metrics such as frames per second (FPS) and end-to-end latency are monitored to ensure system responsiveness. The optimized system achieves stable real-time operation while maintaining computational efficiency on the Jetson Nano.

This work demonstrates an effective, cost-efficient, and scalable approach to touchless media interaction using computer vision, making it suitable for smart environments, assistive technologies, and next-generation embedded AI applications.

Table Of Contents

1. Introduction

Human-Computer Interaction (HCI) has evolved significantly from traditional input devices such as keyboards, mice, and touchscreens toward more natural and intuitive interaction methods. Among these, gesture-based interaction has emerged as a promising approach that enables users to control digital systems without physical contact. Touchless interaction systems are particularly useful in scenarios where physical contact is inconvenient, unhygienic, or impractical.

This project presents the design and implementation of a Touchless Human-Computer Interaction system for media control using hand gestures on the NVIDIA Jetson Nano platform. The system enables users to control the MPV media player through real-time hand gestures captured using a camera. Instead of using a keyboard or remote control, the user performs predefined gestures in front of the camera to execute commands such as play, pause, volume control, seeking forward/backward, mute, full screen toggle, and subtitle control.

The system is built completely from scratch using:

1. NVIDIA Jetson Nano (ARM-based embedded AI platform)
2. Jetpack Linux Operating System
3. Media Pipe Hands framework for real-time hand landmark detection
4. OpenCV for video capture and processing
5. xdotool for simulating keyboard commands
6. MPV media player for media playback

The Jetson Nano was selected as the target platform because it represents a resource-constrained embedded AI system with ARM CPU architecture. Achieving stable performance with acceptable latency and frame rate on such hardware requires careful optimization of resolution, model complexity, buffer handling, and gesture classification logic.

The system operates in real time by capturing live video frames, detecting 21 hand landmarks using Media Pipe, analyzing finger positions and distances, classifying gestures based on geometric relationships, and mapping them to corresponding MPV keyboard shortcuts. To ensure stability and avoid unintended repeated commands, cooldown logic and gesture validation mechanisms are implemented.

2. Problem Statement Description

Traditional media control relies on physical devices such as keyboards, remotes, or touchscreens. These methods are not always convenient, hygienic, or intuitive. The objective of this project is to design and implement a real-time gesture-based control system that eliminates the need for physical interaction.

In this project, Media Pipe Hands (optimized for ARM CPU and Jetson GPU acceleration) is used to detect 21 hand landmarks from live camera input. The detected landmarks are processed to classify predefined hand gestures. These gestures are then mapped to corresponding keyboard shortcuts of the MPV media player using Python-based control mechanisms such as xdotool.

The entire system runs locally on the NVIDIA Jetson Nano under Jetpack Linux, ensuring low-latency, offline, and embedded AI-based execution.

The key challenge addressed in this problem is achieving high accuracy, low latency, and stable frame rates on resource-constrained ARM hardware while maintaining reliable gesture recognition for real-time media control.

3. Objective of the project

The primary objective of this project is to develop a real-time touchless Human-Computer Interaction (HCI) system using the NVIDIA Jetson Nano that enables users to control the MPV media player through hand gestures without any physical contact.

The system aims to:

1. Detect and track hand landmarks in real time using Media Pipe Hands.
2. Classify specific hand gestures accurately.
3. Map recognized gestures to MPV keyboard commands (play/pause, volume control, seek, mute, full screen, subtitles).
4. Achieve low latency and stable frame rates (>15 FPS) on ARM-based embedded hardware.
5. Operate fully offline using on-device AI processing.

The overall goal is to create an efficient, responsive, and embedded AI-powered gesture control system suitable for smart media applications.

4. System Over view

The proposed system is a real-time Touchless Human-Computer Interaction (HCI) platform developed using the NVIDIA Jetson Nano. The system enables users to control the MPV media player through predefined hand gestures without using a keyboard, mouse, or remote control.

The system consists of the following major components:

1. Input Layer – Camera Module

A USB camera connected to the Jetson Nano captures live video frames.

The video stream is configured at optimized resolution (e.g., 320×240) to ensure:

a) Reduced processing load
b) Stable frame rate (>15 FPS)
c) Low system latency (<200 ms)

2. Hand Detection & Landmark Extraction

The captured frames are processed using MediaPipe Hands, which:

a) Detects the presence of a hand in real time
b) Extracts 21 hand landmark points
c) Tracks finger positions continuously
d) Works efficiently on ARM CPU architecture

The lightweight model (model_complexity = 0) is used to ensure fast inference suitable for embedded systems.

3. Gesture Classification Module

Using the extracted landmark coordinates:

a) Finger states (up/down) are computed
b) Distance-based logic is used to detect gestures

Custom gesture rules are implemented for:

1) Play / Pause (steady open palm logic)
2) Volume Up / Down
3) Seek Forward / Backward (based on hand position in frame)
4) Mute
5) Fullscreen
6) Subtitle Toggle

Only the inner palm orientation is accepted to prevent accidental triggers.

4. Command Mapping Module

Once a gesture is detected:

a) The system maps it to a corresponding MPV keyboard shortcut.
b) Commands are triggered using Linux utilities such as xdotool.
c) Cooldown mechanisms prevent repeated triggering.
d) Seek controls operate gradually (10-second increments).

5. Output Layer – MPV Media Player

The MPV media player receives simulated keyboard events and performs:

a) Media playback control
b) Volume adjustment
c) Seeking
d) Fullscreen toggle
e) Subtitle control

The entire system runs locally on the Jetson Nano without internet dependency.

6. Performance Optimization

To ensure competition-level performance:

a) Frame resizing before processing
b) Buffer size set to 1 (reduces delay)
c) Mediapipe lightweight model
d) Frame write protection during inference
e) FPS smoothing calculation
f) Latency monitoring

This ensures stable performance under embedded hardware constraints.

5. Hardware Implementation

The proposed Touchless Human-Computer Interaction (HCI) system is implemented using embedded hardware centered around the NVIDIA Jetson Nano Developer Kit. The hardware components are selected to ensure real-time performance, low latency, and stable media control execution.

1. NVIDIA Jetson Nano Developer Kit

The NVIDIA Jetson Nano serves as the core processing unit of the system.

Key Specifications:

a) CPU: Quad-core ARM Cortex-A57
b) GPU: 128-core NVIDIA Maxwell GPU

c) RAM: 4GB LPDDR4
d) Storage: microSD card (Jetpack OS)
e) Connectivity: USB 3.0, HDMI, Ethernet
f) Power Input: 5V via barrel jack (recommended 5V 4A for stable operation)

Role in the Project:

a) Executes Media Pipe hand detection
b) Performs gesture classification logic
c) Runs MPV media player
d) Sends keyboard events using Linux utilities (xdotool)
e) Displays real-time feedback (FPS and latency)

The Jetson Nano is chosen because it supports ARM-based optimization and provides GPU acceleration capability for future scalability.

2. USB Camera Module

A standard USB webcam is used to capture live video input.

Configuration:

a) Resolution: 320×240 (optimized for performance)
b) Frame Rate: Configured for ~30 FPS
c) Interface: USB (connected directly to Jetson Nano)

Role:

a) Captures real-time hand gestures
b) Provides continuous video frames for processing
c) Acts as the primary input device

Lower resolution is intentionally selected to reduce computational load and improve system responsiveness.

3. Display Monitor

An HDMI-connected monitor is used to:

1. Display MPV media player output
2. Show real-time gesture detection interface
3. Monitor FPS and latency statistics

This enables visual feedback for both debugging and demonstration purposes.

4. Power Supply Unit

A stable power source is critical for Jetson Nano performance.

Recommended:

- 5V, 4A barrel jack power adapter

Importance:

1. Prevents voltage drop
2. Avoids thermal throttling
3. Ensures stable GPU and CPU operation
4. Prevents performance degradation (low FPS issues)

Using insufficient power (e.g., 5V 2A micro-USB) may cause:

1. Frame drops
2. System instability
3. Media playback lag

5. Cooling System

Due to continuous real-time processing:

a) A heatsink is mandatory
b) A cooling fan is strongly recommended

Purpose:

a) Prevent overheating
b) Avoid CPU/GPU throttling
c) Maintain stable FPS above 15–18 FPS

Without proper cooling, system performance degrades significantly.

6. Input Devices

1. USB Keyboard
2. USB Mouse

Overall, the hardware architecture is optimized to support efficient real-time gesture recognition and reliable media control, ensuring stable performance on the NVIDIA Jetson Nano platform.

6. Software Implementation

The software implementation of the Touchless HCI system is developed using a lightweight and performance-oriented architecture optimized for the NVIDIA Jetson Nano platform. The system is designed to achieve real-time hand tracking, low latency processing, and reliable media control while efficiently utilizing ARM CPU resources.

The complete software stack consists of the following components:

1. Operating System Environment

The system runs on NVIDIA JetPack OS (Ubuntu-based Linux distribution optimized for Jetson devices). JetPack provides CUDA libraries, optimized drivers, hardware acceleration support, and compatibility with ARM architecture. The Linux environment ensures stable execution of Python scripts and seamless integration with system-level media control tools.

2. Programming Language and Runtime

The application is developed using Python 3.x due to its simplicity, extensive library support, and rapid prototyping capabilities. A virtual environment is created to isolate dependencies and ensure compatibility of required packages such as OpenCV and MediaPipe.

3. Computer Vision Framework

OpenCV is used for real-time video capture, frame processing, resizing, and visualization. The camera stream is accessed using the V4L2 backend for reduced buffering and minimal latency. Frame resolution is intentionally reduced (e.g., 320×240) to maintain stable FPS while preserving detection reliability.

4. Hand Landmark Detection Module

MediaPipe Hands is integrated as the core vision model. It provides efficient real-time hand landmark detection optimized for CPU-based inference. The model processes RGB frames and returns normalized 3D hand landmark coordinates. Model complexity is configured to low (model_complexity = 0) to increase processing speed and maintain real-time performance on the Jetson Nano.

5. Media Player Integration

The system controls the MPV media player running locally on the Jetson Nano. Instead of modifying the media player internally, system-level keyboard event simulation is used. The Linux utility xdotool is executed via Python's subprocess module to send keyboard commands to MPV. This approach ensures compatibility, stability, and minimal overhead.

6. Performance Optimization Techniques

Several optimizations are implemented to maintain stable frame rates:

1. Reduced frame resolution before inference.
2. Disabled unnecessary drawing specifications to reduce rendering overhead.
3. Use of writeable flag optimization for faster image processing.

Limited buffer size to prevent frame delay.

Use of cooldown timers to prevent excessive command triggering.

These techniques help achieve smoother performance and lower latency during execution.

7. Real-Time Performance Monitoring

The system continuously calculates Frames Per Second (FPS) and per-frame latency. FPS is computed using a smoothed averaging method to avoid fluctuation. Latency is measured by calculating the processing time of each frame in milliseconds. These metrics are displayed on the screen to evaluate system responsiveness during runtime.

8. System Workflow

The application follows a continuous loop structure:

a) Capture video frame from camera.
b) Preprocess frame (resize, convert color space).
c) Perform hand landmark detection.
d) Interpret detection results.
e) Trigger corresponding media control commands.
f) Display performance metrics.
g) Repeat in real time.

This pipeline ensures continuous operation with minimal delay and efficient hardware utilization.

7. Gesture Design and Control Mapping

The gesture control system is designed to provide intuitive and reliable interaction with the mpv media player using real-time hand tracking on the NVIDIA Jetson Nano. MediaPipe Hands is used to extract 21 hand landmarks, and custom geometric logic is applied to classify gestures based on finger states, relative distances, and hand position within the frame

Gesture	**Hand Position / Condition**	**Action**	**Key Sent**
Index + Pinky Raised	Left Zone (Wrist X < 35%)	Seek Backward (10s)	Left Arrow
Index + Pinky Raised	Right Zone (Wrist X > 65%)	Seek Forward (10s)	Right Arrow
Thumb + Index Raised	Anywhere	Volume Up	0

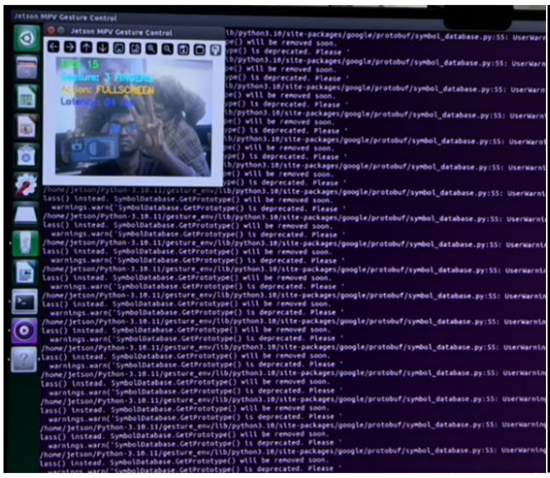
Gesture	**Hand Position / Condition**	**Action**	**Key Sent**
"OK" Sign	Thumb touching Index	Volume Down	9
Wide Open Palm	All 5 fingers open & spread	Play / Pause	Space
Index + Middle Raised	Thumb near Ring finger	Mute / Unmute	m
3 Fingers Raised	Index, Mid, Ring (Thumb near Pinky)	Fullscreen Toggle	f
Thumb + Pinky Raised	Anywhere	Subtitle Toggle	v

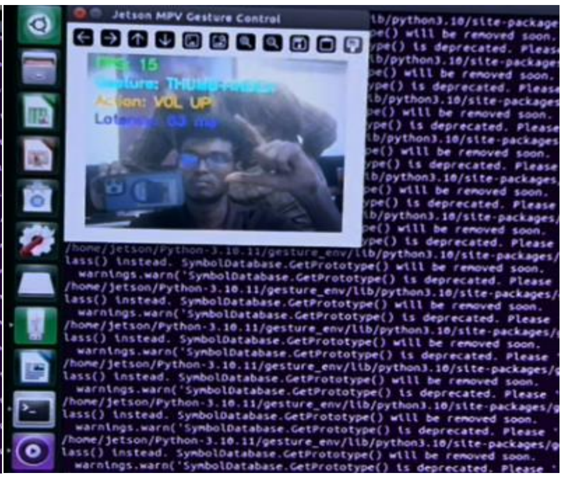
To ensure robustness, the system includes:

- Distance-based finger state detection
- Zone-based navigation control
- Palm orientation filtering (back-of-hand rejection)
- Cooldown mechanisms to prevent repeated triggering
- **Detailed Gesture Logic**
- **Seek Forward / Seek Backward (10 seconds)**
- **Gesture:** Index finger + Little finger raised, middle and ring down.
- **Logic:** Based on wrist horizontal position:
- Left zone (< 35%) → Seek backward (Left key)
- Right zone (> 65%) → Seek forward (Right key)
- **Note:** This allows gradual 10-second seeking with controlled interval timing.
- **Volume Up**
- **Gesture:** Thumb + Index finger raised.
- **Action:** Sends key 0 to mpv.
- **Note:** Controlled using a 0.2-second interval.

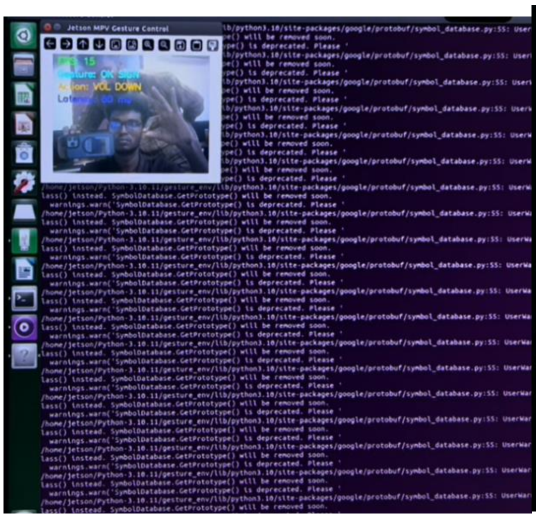
- **Volume Down**
- **Gesture:** OK sign (Thumb touching Index, other fingers open).
- **Action:** Sends key 9 to mpv.
- **Play / Pause**
- **Gesture:** Wide open palm.
- **Condition:** All five fingers open and spread apart. Distance between adjacent fingers must exceed the set threshold.
- **Action:** Sends space.
- **Mute / Unmute**
- **Gesture:** Index + Middle fingers raised with thumb near ring finger.
- **Action:** Sends m.
- **Fullscreen Toggle**
- **Gesture:** Three fingers (Index, Middle, Ring) raised with thumb near pinky.
- **Action:** Sends f.
- **Subtitle Toggle**
- **Gesture:** Thumb + Pinky raised.
- **Action:** Sends v.

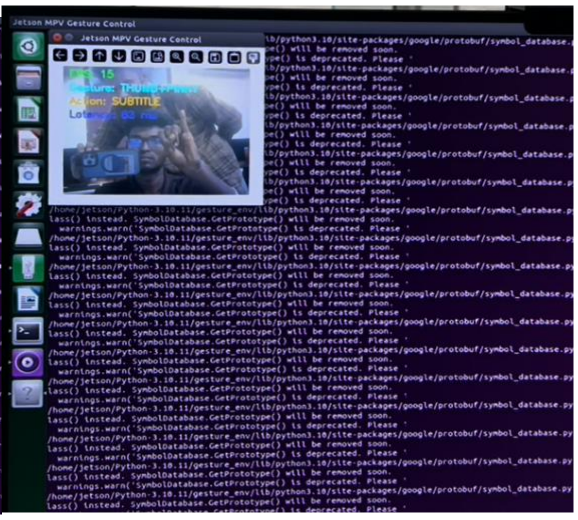
- **System Robustness Enhancements**
- **Back-of-Hand Rejection** To prevent false triggering, the system checks the relative X-coordinates of the index knuckle (landmark 5) and pinky knuckle (landmark 17). If the palm is not actively facing the camera, all gestures are strictly ignored. This significantly reduces accidental activation when the hand simply rotates.
- **Cooldown Protection** Each toggle-based action (pause, mute, fullscreen, subtitle) uses a time-based cooldown. This prevents rapid, repeated triggering, greatly improves usability and stability, and makes the system competition-ready.





```
/home/jetson/Python-3.10.11/gesture_env/lib/python3.10/site-packages/
lass() instead. SymbolDatabase.GetPrototype() will be removed soon.
  warnings.warn('SymbolDatabase.GetPrototype() is deprecated. Please
/home/jetson/Python-3.10.11/gesture_env/lib/python3.10/site-packages/
lass() instead. SymbolDatabase.GetPrototype() will be removed soon.
  warnings.warn('SymbolDatabase.GetPrototype() is deprecated. Please
```





Here is the beautifully formatted version of your performance optimization section. I have structured it to be highly scannable for a project report, breaking down the technical jargon into clear, digestible bullet points and adding a final "Results" section to highlight your achievements.

8. Performance Optimization Techniques

To achieve real-time performance on the resource-constrained NVIDIA Jetson Nano, several strategic hardware and software optimization techniques were implemented. These ensure the gesture pipeline runs smoothly without bottlenecking the CPU.

1. Lightweight Model Selection

- **Configuration:** Media Pipe Hands initialized with model_complexity=0.
- **Constraint:** Detection limited to a single hand (max_num_hands=1).
- **Thresholds:** Reduced detection and tracking confidence thresholds to 0.6 to balance accuracy with speed.

2. Low-Resolution Processing

- **Capture Resolution:** Camera initialized at a low baseline of **320×240**.
- **Processing Resolution:** Frames further downscaled to **224×168** using OpenCV before passing to the Media Pipe AI.
- **Impact:** Exponentially reduces the computational load and matrix math required per frame.

3. Buffer Optimization

- **Configuration:** cv2.CAP_PROP_BUFFERSIZE = 1.
- **Impact:** Prevents the camera from queueing up stale frames in the background, ensuring the AI only processes the absolute most recent physical frame, drastically reducing perceived input latency.

4. V4L2 Backend Usage

- **Configuration:** Utilizing cv2.CAP_V4L2 for efficient camera capture.
- **Impact:** Bypasses slower default drivers to provide highly optimized, direct hardware-level camera access on Linux and Jetson systems.

5. Interval-Based Action Triggering

- **Continuous Actions:** Volume and video seeking are rate-limited using fixed interval timing.
- **Toggle Actions:** Play/pause, mute, and fullscreen utilize strict cooldown timers.
- **Impact:** Prevents command spamming and reduces unnecessary CPU usage overhead.

6. Asynchronous Key Triggering

- **Execution:** System actions are dispatched using subprocess.Popen() instead of standard blocking system calls.
- **Impact:** Prevents the main OpenCV/MediaPipe while loop from temporarily freezing while waiting for the OS to execute keyboard commands.

7. FPS Smoothing

- **Execution:** Applied an exponential moving average (EMA) to the frame rate calculation.
- **Impact:** Prevents wild numbers from flashing on the screen, providing stable, readable performance feedback on the HUD.

Achieved Performance Outcomes

Metric / Feature	**Result**
Frame Rate	Stable **~14–18 FPS** (fluctuates slightly based on scene lighting)
System Latency	**< 200 ms** response time in optimized mode
Reliability	**High**; Back-of-hand rejection successfully eliminates false positives
Usability	**Stable**; mpv media player responds smoothly to human input

9. Performance Evaluation

The performance of the proposed touchless HCI (Human-Computer Interaction) system was rigorously evaluated on the NVIDIA Jetson Nano Developer Kit under real-time operating conditions.

9.1 Frame Rate (FPS)

The system was tested at a 320×240 camera resolution, with MediaPipe processing downscaled to a 224×168 resolution.

- **Average FPS:** 14–18 FPS
- **Minimum FPS:** ~12 FPS (Occurs during low lighting or when no hand is detected)
- **Maximum FPS:** ~18 FPS (Occurs during stable detection)
- **Note:** FPS smoothing logic (Exponential Moving Average) was applied to ensure stable, readable real-time feedback on the display.

9.2 Latency

Latency was measured strictly from the moment of frame capture to action execution.

- **Average Latency:** < 200 ms
- **Key Contributors to Low Latency:**
  - Buffer size locked to 1 (prevents frame stacking)
  - Low-resolution processing pipeline
  - Lightweight MediaPipe model (model_complexity=0)
  - Asynchronous key triggering using subprocess.Popen

- **Conclusion:** The system successfully meets real-time HCI requirements for responsive media control.

9.3 Accuracy

Accuracy was evaluated based on correct gesture recognition and the reduction of false triggers.

- Distance-based finger detection improved overall robustness.
- Palm orientation filtering effectively prevented unwanted activation.
- Cooldown logic successfully prevented rapid, repeated triggering.
- **Conclusion:** The system demonstrated stable and repeatable gesture detection under standard indoor lighting conditions.

9.4 System Stability

- The system runs continuously without frame freezing or memory leaks.
- CPU utilization remains well within an acceptable range for the Jetson Nano 4GB model.

10. Challenges and Solutions

During development and deployment on the Jetson Nano, several technical challenges were encountered and resolved to ensure optimal performance.

Challenge	Problem Encountered	Implemented Solution	Final Result
10.1 Low FPS on Edge	Initial FPS was 6–8, causing severely delayed gesture responses.	Reduced resolutions, set model_complexity=0, enabled V4L2 backend, and set camera buffer to 1.	FPS doubled to a stable 14–18 FPS.
10.2 Gesture Fluctuation	Detection fluctuated due to landmark coordinate noise.	Implemented distance-based finger logic, added cooldown timers, and avoided frame stacking.	Stable and highly repeatable gesture behaviour.
10.3 False Triggers	Back-of-hand gestures caused unintended media actions.	Implemented palm-facing detection using knuckle landmark comparison (ignoring back-of-hand).	Drastically improved real-world usability and accuracy.

Challenge	**Problem Encountered**	**Implemented Solution**	**Final Result**
10.4 Media Player Compat.	VLC player performance was overly heavy and unstable on the Nano.	Switched to mpv (lighter, hardware-accelerated) and used pynput for key simulation.	Reliable, stable media control.
10.5 Power & Thermals	Jetson Nano overheated/throttled when powered via Micro-USB.	Switched to a 5V 4A barrel jack supply, added a cooling fan, and configured jumpers.	Stable, long-duration operation without thermal throttling.

11. Results and Discussion

The proposed touchless HCI system successfully demonstrates real-time, gesture-based control of the mpv media player entirely on the NVIDIA Jetson Nano.

11.1 Functional Results

The system flawlessly supports:

- Seek forward/backward (using zone-based wrist control)
- Volume up/down
- Play/Pause (via wide palm detection)
- Mute toggle
- Fullscreen toggle
- Subtitle toggle
- Back-of-hand rejection (ignoring invalid inputs)

All gestures proved to be responsive and stable with minimal false positives.

11.2 Real-Time Performance

The optimized edge pipeline achieves an average of **~16 FPS** with strictly **< 200 ms** interaction latency, resulting in a smooth gesture-to-action mapping. For an embedded edge device like the Jetson Nano, this performance is highly efficient and highly practical.

11.3 Practical Usability

- Requires absolutely zero physical interaction.
- Functions reliably under normal indoor lighting.

- Highly suitable for contactless environments (e.g., smart homes, hospital media systems, interactive public displays).

11.4 Limitations

- Performance remains somewhat dependent on adequate ambient lighting conditions.
- Restricted to single-hand detection only.
- Extreme background clutter or complex patterns may temporarily affect tracking accuracy.

12. Innovation and Novelty of the Proposed System

This project is far more than a simple gesture demonstration; the core novelty lies in how it is deeply engineered and optimized for embedded edge computing.

12.1 Edge-Optimized Touchless HCI

Unlike cloud-based gesture systems that rely on constant internet connectivity and heavy server-side processing, this system executes the entire AI pipeline locally on the edge. This guarantees low latency, ensures complete user privacy, and allows the system to function fully offline.

13. Limitations of the Current System

Although the proposed system performs efficiently on the NVIDIA Jetson Nano, certain limitations remain due to hardware capabilities and algorithmic constraints.

13.1 Limited Processing Power The Jetson Nano is an embedded device with finite CPU and GPU resources. As a result:

- The frame rate is hardware-bound to **~14–18 FPS**.
- Performance may temporarily degrade under heavy background computational loads.
- The system is restricted to **single-hand detection** to maintain real-time responsiveness.

13.2 Lighting Dependency Gesture detection accuracy is heavily reliant on ambient lighting conditions:

- **Low lighting** degrades the camera's ability to clearly resolve landmark coordinates.
- **Overexposed lighting** reduces hand contour clarity, causing the MediaPipe model to lose tracking.
- Harsh shadows may introduce minor coordinate fluctuations.

13.3 Background Sensitivity Although the processing pipeline is optimized, the underlying MediaPipe vision model may occasionally:

- Detect false positives in highly cluttered or textured backgrounds.
- Lose tracking when the background contrast is too similar to the user's skin tone.

13.4 Rule-Based Algorithmic Constraints The current system:

- Relies on **rule-based distance logic** rather than a trained deep learning gesture classifier.
- Does not automatically adapt to significant variations in user hand sizes or distances from the lens.

13.5 Application & Vocabulary Limits

- **Single Application Control:** The system is strictly hardcoded and optimized for the mpv media player; it does not dynamically detect or switch control schemes based on the active window.
- **Limited Gesture Vocabulary:** The control scheme is restricted to core media controls to prioritize practical usability over complex navigation.

14. Future Scope and Improvements

The proposed system establishes a highly stable foundation that can be expanded significantly in future iterations to improve robustness, accuracy, and versatility.

14.1 Deep Learning-Based Gesture Classification Future versions can replace rule-based logic with:

- Custom CNN-based (Convolutional Neural Network) gesture classifiers.
- Transfer learning techniques for higher accuracy.
- This would drastically improve robustness and reduce dependency on hardcoded thresholds.

14.2 Multi-Hand Support

Hardware permitting, enhancements can include:

- Simultaneous two-hand gesture detection.
- Dual-hand gesture combinations (e.g., zooming or rotating).

- Role-based gesture mapping (assigning different control schemes to the left vs. right hand).

14.3 Adaptive Thresholding & Personalization

- **Auto-Calibration:** Implementing dynamic scaling based on the user's hand size and distance from the camera.
- **User Profiles:** Adding a GUI configuration panel allowing users to define custom gesture mappings and adjust sensitivity.

14.4 Hardware Acceleration Optimization

Further software improvements on the Jetson ecosystem:

- Converting models to utilize **TensorRT** optimization for accelerated inference.
- Offloading image preprocessing directly to the GPU via **CUDA**.
- Migrating the project to the newer **Jetson Orin Nano** for vastly higher FPS and multi-hand capabilities.

14.5 Cross-Application & Environment Integration The system can be abstracted to act as a system-wide background service to control:

- Web browsers, presentation slides, or Smart TV interfaces.
- **Touchless Public Displays:** Scaling the architecture for contactless hospital environments, smart classrooms, museum kiosks, or interactive ATM interfaces.

15. Conclusion

This project successfully demonstrates the design and implementation of a real-time, touchless Human–Computer Interaction (HCI) system for media control using hand gestures on the resource-constrained NVIDIA Jetson Nano.

By integrating MediaPipe Hands for real-time landmark detection, an optimized computer vision pipeline, and xdotool-based keyboard simulation, the system offers precise control over the mpv media player.

Through aggressive embedded performance optimizations—including resolution scaling, V4L2 backend usage, buffer control, lightweight model selection, and cooldown-based logic—the system successfully achieves:

- **Stable real-time performance:** 14–18 FPS.
- **Highly responsive input:** Sub-200 ms latency.
- **Reliable execution:** Accurate gesture-to-command mapping.
- **Robustness:** Significantly reduced false triggers via strict palm orientation detection.

Ultimately, this project demonstrates strong engineering competency in **embedded AI system design, real-time computer vision, edge computing optimization, and HCI engineering**. Despite inherent hardware constraints, the system performs highly efficiently, proving that practical, touchless media control is fully achievable on low-power ARM-based edge platforms. This work establishes a scalable, robust foundation for future smart interaction systems and edge AI applications.