Bharat AI-SoC

Hand Gesture Controlled Smart Media Interface

V Madhav | Nihal Prasad | Aryaman Swami
Second Year
Electronics and Communication Engineering (ECE)
Indian Institute of Space Science and Technology (IIST), Thiruvananthapuram

1. Introduction

With the rapid advancement of human–computer interaction (HCI), touchless control systems are becoming increasingly important in modern computing environments. Gesture-based interfaces allow users to interact with systems without physical contact, improving accessibility, hygiene, and convenience.

This project presents a **real-time hand gesture recognition system** capable of controlling media playback and system volume using computer vision and machine learning. The system uses a webcam to detect hand gestures and maps them to actions such as play/pause, volume control, and mute.

The solution is designed to be lightweight, cross-platform, and responsive, making it suitable for smart environments, accessibility tools, and future AR/VR interfaces.

2. Objectives

The main objectives of this project are:

- Develop a real-time hand gesture recognition system
- Use machine learning for gesture classification
- Implement cross-platform media and volume control
- Ensure low latency and smooth user interaction
- Build an optimized and user-friendly interface
- Demonstrate practical AI + embedded interaction use case

3. System Overview

The system consists of the following modules:

i. **Camera Input Module** – Captures real-time video
ii. **Preprocessing** – Frame resize, RGB conversion.
iii. **Hand Detection Module** – Detects and tracks hand landmarks
iv. **Feature Extraction Module** – Converts landmarks into ML features
v. **Gesture Classification Model** – Predicts gesture using trained model
vi. **Action Control Module** – Maps gestures to system actions
vii. **User Interface Module** – Displays gesture and system state

4. Technologies Used

Technology	Purpose
Python	Core programming language
OpenCV	Image processing and camera handling
MediaPipe	Hand landmark detection
Scikit-learn	Machine learning model training
Joblib	Model saving/loading
NumPy	Numerical operations
Pynput	Keyboard automation
Subprocess / ctypes	System volume control
Platform module	Cross-platform compatibility

5. Methodology

5.1. Hand Detection

The system uses **MediaPipe Hand Landmarker** to detect hand landmarks. Each frame from the webcam is processed to extract **21 key points** of the hand. These landmarks represent finger joints and palm positions in 3D space.

5.2. Feature Extraction

To make gesture recognition robust:

- Wrist is used as reference point
- All landmarks normalized relative to wrist
- Coordinates flattened into feature vector
- Noise reduction using prediction history buffer

This ensures invariance to hand position and distance.

5.3. Machine Learning Model

A *Random Forest* classifier is used due to its fast inference, robustness, and suitability for structured landmark data. The workflow includes:

i. Collect gesture landmark dataset
ii. Train classifier
iii. Save model using joblib
iv. Load model during runtime

5.4. Gesture Recognition Logic

To ensure stable recognition and avoid false triggers:

- Confidence threshold applied
- Prediction smoothing using deque history
- Gesture hold-time detection
- Cooldown between actions

This ensures stable and intentional gesture recognition.

5.5. Control Modes

The system operates in two modes:

Mode Switching

Gesture	Action
Like	**Switch between Playback and Volume modes**

Playback Mode

Gesture	Action
Fist	**Play / Pause**
OK	**Fast Forward**
Rock	**Rewind**

Volume Mode

Gesture	Action
Fist	**Volume Increase**
OK	**Volume Decrease**
Rock	**Mute / Unmute**

5.6. Cross-Platform System Control

System automatically detects OS:

- Windows → Uses keyboard media keys
- Linux → Uses amixer commands
- Keyboard control via pynput

This makes system platform independent.

5.7. Performance Optimization

Several optimizations were implemented to maintain real-time performance:

- Reduced frame resolution for faster processing
- Even-frame processing
- Separate Thread for camera
- Reduced camera buffer size to minimize delay
- Efficient landmark normalization and smoothing

These improvements result in stable detection, low latency, and smooth user interaction.

6. User Interface Design

The system interface provides real-time visual feedback during gesture operation. The overlay includes:

- Current operating mode (PLAYBACK / VOLUME) displayed prominently at the top
- FPS and latency values for performance monitoring
- Live webcam feed with detected gesture label
- Progress bar indicating gesture hold validation
- "MODE CHANGED" visual feedback during mode switching

The UI uses a dark header-style overlay for status information and a high-contrast gesture label for clarity. The layout keeps the webcam feed central while allowing simultaneous media playback visibility, ensuring both usability and real-time system transparency.

7. Results

The system successfully demonstrates:

- Real-time hand gesture detection
- Accurate gesture classification
- Smooth media and volume control
- Reliable mode switching
- Responsive user interface

Overall performance remains stable with minimal delay and high usability.

8. Applications

This system can be used in:

- Smart home control
- Touchless media systems

- Accessibility tools
- AR/VR interaction
- Automotive infotainment
- Public kiosks

9. Challenges Faced

Key challenges included lighting variation affecting detection, occasional false gesture triggers, maintaining real-time performance, and ensuring consistent operation across different operating systems. These were handled through preprocessing, prediction filtering, performance optimization, and platform-specific control handling.

10. Future Improvements

Future enhancements may include:

- Deep learning gesture model (CNN/LSTM)
- Custom gesture training interface
- Multi-hand detection
- Smart home IoT integration
- Mobile device control
- Voice + gesture hybrid control
- Edge AI deployment (Raspberry Pi/Jetson)

11. Conclusion

The project demonstrates an efficient and intelligent gesture-controlled media interface using computer vision and machine learning. It successfully combines real-time processing, cross-platform compatibility, and user-friendly interaction.

This system highlights the potential of AI-driven touchless interfaces and provides a strong foundation for future smart interaction technologies.

12. System Demonstration

Figure: Playback Mode – OK Gesture Detection and Media Control
The system operating in Playback Mode, detecting the *OK gesture* in real time.

Figure 2: Mode Switching Using “Like” Gesture
Mode switching between playback and volume modes using the “Like” gesture.

Figure: Playback Mode – Fist Gesture for Play/Pause

APPENDIX

```
import ctypes
try:
    ctypes.CDLL("libgomp.so.1", mode=ctypes.RTLD_GLOBAL)
except:
    pass

import os
import subprocess
os.environ["QT_X11_NO_MITSHM"] = "1"

import mediapipe as mp
import cv2
import time
import numpy as np
import joblib
from collections import deque
from threading import Thread, Lock
from pynput.keyboard import Controller, Key

model = joblib.load("hand_gesture.joblib")
keyboard = Controller()

HDMI_SINK = "alsa_output.platform-70030000.hda.hdmi-stereo"

def volume_up():
    subprocess.run(["amixer", "-c", "0", "sset", "IEC958", "5%+"],
stdout=subprocess.DEVNULL)

def volume_down():
    subprocess.run(["amixer", "-c", "0", "sset", "IEC958", "5%-"],
stdout=subprocess.DEVNULL)

def volume_mute():
    subprocess.run(["amixer", "-c", "0", "sset", "IEC958", "toggle"],
stdout=subprocess.DEVNULL)

def play_pause():
    keyboard.press(Key.space)
    keyboard.release(Key.space)

def fast_forward():
    keyboard.press(Key.right)
    keyboard.release(Key.right)

def rewind():
    keyboard.press(Key.left)
    keyboard.release(Key.left)

mp_hands = mp.solutions.hands
hands = mp_hands.Hands(
    static_image_mode=False,
    max_num_hands=1,
    min_detection_confidence=0.6,
    min_tracking_confidence=0.6
)

class CameraThread:
```

```
    def __init__(self, src=0):
        self.cap = cv2.VideoCapture(src)
        self.cap.set(cv2.CAP_PROP_FRAME_WIDTH, 640)
        self.cap.set(cv2.CAP_PROP_FRAME_HEIGHT, 480)
        self.frame = None
        self.lock = Lock()
        self.running = True
        Thread(target=self.update, daemon=True).start()

    def update(self):
        while self.running:
            ret, frame = self.cap.read()
            if ret:
                with self.lock:
                    self.frame = frame

    def read(self):
        with self.lock:
            return None if self.frame is None else self.frame.copy()

    def stop(self):
        self.running = False
        self.cap.release()

MODES = ["playback", "volume"]
current_mode_index = 0
mode_switch_time = 0

prediction_history = deque(maxlen=3)
gesture_hold_start = None
current_active_gesture = None

HOLD_TIME = 1.0
CONF_THRESHOLD = 0.90

camera = CameraThread(0)
frame_counter = 0

while True:
    frame_start = time.perf_counter()
    original_frame = camera.read()

    if original_frame is None:
        continue

    frame_counter += 1
    if frame_counter % 2 != 0:
        continue

    small = cv2.resize(original_frame, (256, 192))
    frame_rgb = cv2.cvtColor(small, cv2.COLOR_BGR2RGB)

    results = hands.process(frame_rgb)

    h, w = original_frame.shape[:2]
    detected_gesture = "none"
    current_time = time.perf_counter()

    if results.multi_hand_landmarks:
```

```
        landmarks = results.multi_hand_landmarks[0].landmark
        lm = np.zeros((21, 3), dtype=np.float32)
        for i in range(21):
            p = landmarks[i]
            lm[i] = [p.x, p.y, p.z]

        lm -= lm[0]
        X_live = lm.reshape(1, -1)

        probabilities = model.predict_proba(X_live)[0]
        max_idx = np.argmax(probabilities)

        if probabilities[max_idx] >= CONF_THRESHOLD:
            detected_gesture = model.classes_[max_idx]

    prediction_history.append(detected_gesture)
    stable_gesture = max(set(prediction_history), key=prediction_history.count)

    if stable_gesture != "none":
        if stable_gesture != current_active_gesture:
            gesture_hold_start = current_time
            current_active_gesture = stable_gesture

        elapsed = current_time - gesture_hold_start
        progress = min(elapsed / HOLD_TIME, 1.0)

        bar_w = int(w * 0.6)
        bar_x = int((w - bar_w) / 2)

        cv2.rectangle(original_frame, (bar_x, h-40), (bar_x + bar_w, h-25), (100,
100, 100), -1)
        cv2.rectangle(original_frame, (bar_x, h-40), (bar_x + int(bar_w *
progress), h-25), (0, 255, 0), -1)
        cv2.putText(original_frame, stable_gesture.upper(), (bar_x, h-50),
cv2.FONT_HERSHEY_SIMPLEX, 0.8, (0, 255, 0), 2)

        if progress >= 1.0:
            if stable_gesture == "like":
                current_mode_index = (current_mode_index + 1) % len(MODES)
                mode_switch_time = current_time
                prediction_history.clear()
            else:
                mode = MODES[current_mode_index]
                if mode == "volume":
                    if stable_gesture == "fist": volume_up()
                    elif stable_gesture == "ok": volume_down()
                    elif stable_gesture == "rock": volume_mute()
                elif mode == "playback":
                    if stable_gesture == "fist": play_pause()
                    elif stable_gesture == "ok": fast_forward()
                    elif stable_gesture == "rock": rewind()

                prediction_history.clear()
            gesture_hold_start = current_time
    else:
        gesture_hold_start = None
        current_active_gesture = None

    frame_end = time.perf_counter()
```

```
    latency_ms = (frame_end - frame_start) * 1000
    fps = 1.0 / (frame_end - frame_start)

    cv2.rectangle(original_frame, (0, 0), (w, 60), (30, 30, 30), -1)
    mode_text = "MODE: " + MODES[current_mode_index].upper()
    mode_color = (0, 200, 255) if MODES[current_mode_index] == "playback" else (0,
255, 0)

    cv2.putText(original_frame, mode_text, (20, 40), cv2.FONT_HERSHEY_SIMPLEX,
1.0, mode_color, 3)

    if current_time - mode_switch_time < 1.0:
        cv2.putText(original_frame, "MODE CHANGED", (w//2 - 180, h//2),
cv2.FONT_HERSHEY_SIMPLEX, 1.2, (0, 0, 255), 3)

    cv2.putText(original_frame, f"FPS: {fps:.1f}", (w - 200, 30),
cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 255, 255), 2)
    cv2.putText(original_frame, f"Latency: {latency_ms:.1f} ms", (w - 200, 55),
cv2.FONT_HERSHEY_SIMPLEX, 0.7, (255, 200, 0), 2)

    cv2.imshow("Gesture Control Optimized", original_frame)

    if cv2.waitKey(1) & 0xFF == ord("q"):
        break

camera.stop()
cv2.destroyAllWindows()
```