Hand Gesture Volume Control

Abdelrahman Tarek, Hatem Hussein, Rawan Yasser, Yasmine Moataz, Youssif Hamdi

November 21, 2025

1 Project Definition And Objective

This project aims to address the limitations of traditional input devices by developing a system that enables real-time hand detection and tracking using a standard webcam. The main objectives are to accurately track hand movements, visualize detected landmarks, and allow natural interaction between the user and the computer. The system targets users who require more intuitive and touch-free interaction, such as students, researchers, or individuals working on gesture-based interfaces, and ensures an effective HCI design that supports smooth and responsive interaction.

In terms of Human–Computer Interaction (HCI), the project applies the concept of natural interaction, where users communicate with the system using human actions instead of physical devices. By capturing hand gestures, the system reduces the cognitive and physical effort needed to interact with the computer. This aligns with HCI goals such as improving usability, enhancing accessibility, minimizing input complexity, and creating an interaction style that is more direct, intuitive, and user-centered. The project also reflects HCI principles like feedback (live landmark visualization), affordance (hands as input tools), and low-barrier interaction, allowing users to control the system in a way that feels natural and requires minimal learning.

2 System Design/Architecture

The system is built around three core modules: the Camera Input Module, the Hand Processing Module, and the Visualization Module. The data flow begins with the webcam, which continuously captures live video frames. These frames are forwarded to the Hand Processing Module, where MediaPipe analyzes the image, detects the hand region, and extracts detailed landmark coordinates for each finger joint and key hand position. After the hand data is processed, the output is sent to the Visualization Module, which overlays the detected landmarks and connections directly onto the video stream. This allows users to view their hand movements in real time, creating a smooth, responsive, and intuitive interaction experience. Altogether, the system operates as a real-time pipeline that captures video, understands hand gestures, and visually reflects the tracking results in a clear and interactive manner..

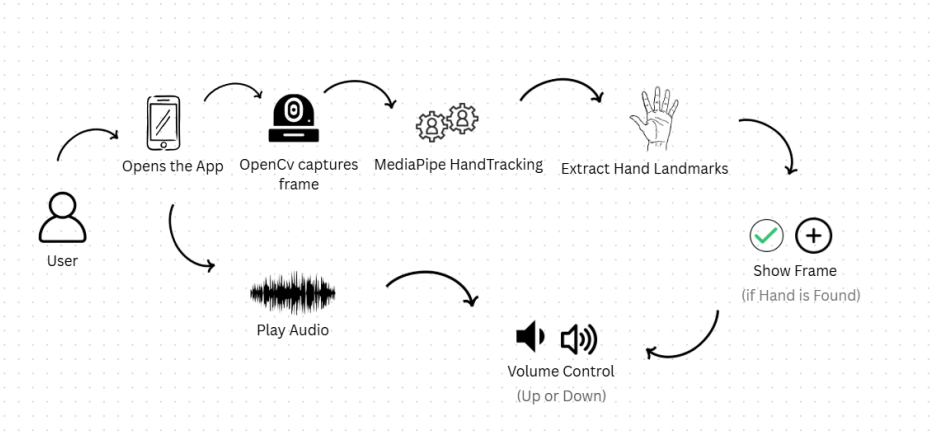

Figure 1: System Overview

3 User Interface (prototype)

The prototype includes main screens such as the live camera view, the hand-tracking display, and the volume-control interaction screen. The live camera view provides a clean and unobstructed display of the user's hand movements, allowing the system to capture frames in real time through the webcam. Once a frame is captured, the hand-tracking display processes it using the MediaPipe library, which detects the hand, extracts its landmarks, and visualizes key points directly on the video feed. In the code, these landmarks—particularly the index finger and thumb—are used to calculate the distance between the two points. This distance is then mapped to a predefined volume range, allowing users to control audio volume through simple hand gestures. When the thumb and index finger move closer or farther apart, the system interprets this gesture and adjusts the volume accordingly. The interface clearly highlights the detected landmarks and draws guiding lines between key points so the user can easily see how their hand movements influence the system. Overall, the prototype demonstrates an interactive, gesture-based control system where hand tracking, landmark detection, and real-time feedback work together to create an intuitive and functional user experience.

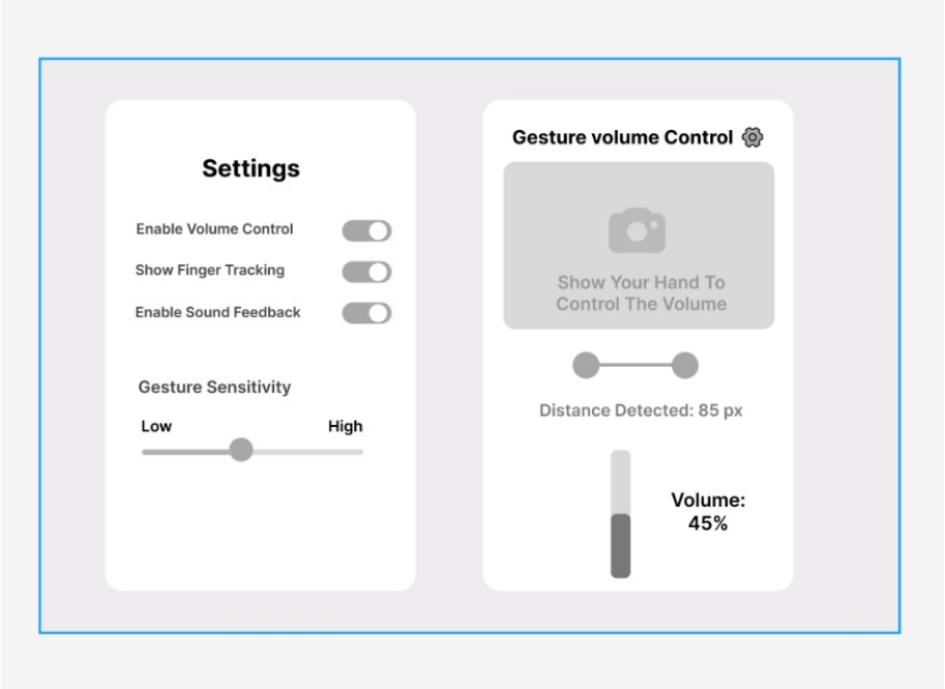

Figure 2: The User Interface

4 Tools and Technology Plan

Technologies Used

- **Python (Backend Language)**
- **OpenCV (Video Handling Library)**
- **Webcam &Cv2.VideoCapture(Real-Time Camera Input)**
- **MediaPipe Hand Tracking**
- **No Database / No Storage**

Python was selected as the backend language due to its simplicity, rapid development capabilities, and strong ecosystem of computer vision and machine learning libraries. OpenCV is used to manage all camera-related operations, including accessing the webcam, capturing frames, converting color formats, and preparing image data for processing. The webcam, accessed through OpenCV's `cv2.VideoCapture`, provides the real-time video stream required for live hand tracking. MediaPipe Hand Tracking was chosen for its high accuracy and efficiency, offering 21 hand landmarks per frame without requiring GPU acceleration, making it ideal for real-time gesture-based systems. Since each frame is processed instantly and discarded afterward, no database or storage component is necessary, which simplifies the system architecture while improving performance and user privacy.

5 Implementation of the concept code

The concept code represents the initial ten percent of the system, focusing on establishing the core functionality required for real-time hand detection and tracking. In this stage, the implementation covers two essential components: video capture and basic hand-tracking logic. Using OpenCV, the system continuously reads frames from the webcam, ensuring a stable and responsive video stream that serves as the foundation for all later processing. These frames are then passed into MediaPipe's Hand Detection module, where the model identifies the presence of a hand and extracts key landmark

positions. The output is visualized directly on the video feed, allowing us to verify that the system can accurately detect and follow hand movements in real time. This early prototype confirms that the core tracking mechanism works effectively, forming a solid base for future expansion into gesture interpretation, interaction design, and more advanced HCI functionalities.

```python
# camera_manager.py - Handles camera operations
import cv2

class CameraManager:
    def __init__(self, camera_index=0):
        self.camera_index = camera_index
        self.cap = None

    def initialize_camera(self):
        "Initialize the camera"
        self.cap = cv2.VideoCapture(self.camera_index)
        if not self.cap.isOpened():
            raise Exception("Error: Could not open webcam")
        print("✓ Webcam initialized successfully!")
        return True

    def read_frame(self):
        """Read a frame from the camera"""
        ret, frame = self.cap.read()
        if ret:
            frame = cv2.flip(frame, 1)  # Mirror effect
        return ret, frame

    def release_camera(self):
        "Release camera resources"
        if self.cap:
            self.cap.release()
        cv2.destroyAllWindows()

    def display_frame(self, frame, window_name='Hand Gesture Volume Control'):
        "Display the frame in a window"
        cv2.imshow(window_name, frame)

    def check_exit_key(self):
        "Check if 'q' key is pressed to exit"
        return cv2.waitKey(1) & 0xFF == ord('q')
```

```python
# hand_tracker.py - Handles hand detection and tracking
import cv2
import mediapipe as mp
import math

class HandTracker:
    def __init__(self):
        self.mp_hands = mp.solutions.hands
        self.mp_drawing = mp.solutions.drawing_utils
        self.hands = self.mp_hands.Hands(
            static_image_mode=False,
            max_num_hands=1,
            min_detection_confidence=0.5,
            min_tracking_confidence=0.5
        )
        print("✓ Hand tracking initialized!")

    def detect_hands(self, frame):
        "Detect hands in the frame and return landmarks"
        rgb_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
        results = self.hands.process(rgb_frame)
        return results

    def draw_hand_landmarks(self, frame, hand_landmarks):
        "Draw hand landmarks and connections on the frame"
        self.mp_drawing.draw_landmarks(
            frame,
            hand_landmarks,
            self.mp_hands.HAND_CONNECTIONS,
            self.mp_drawing.DrawingSpec(color=(0, 255, 0), thickness=2, circle_radius=2),
            self.mp_drawing.DrawingSpec(color=(255, 0, 0), thickness=2)
        )

    def get_finger_positions(self, hand_landmarks, frame_shape):
        "Get thumb and index finger positions in pixel coordinates"
        # Landmark indices: 4=thumb tip, 8=index finger tip
        thumb_tip = hand_landmarks.landmark[4]
```

```python
    index_tip = hand_landmarks.landmark[8]

    (variable) thumb_x: int
    thumb_x = int(thumb_tip.x * w)
    thumb_y = int(thumb_tip.y * h)
    index_x = int(index_tip.x * w)
    index_y = int(index_tip.y * h)

    return (thumb_x, thumb_y), (index_x, index_y)

def calculate_distance(self, point1, point2):
    "Calculate Euclidean distance between two points"
    return math.sqrt((point2[0] - point1[0])**2 + (point2[1] - point1[1])**2)

def draw_finger_visualization(self, frame, thumb_pos, index_pos, distance):
    "Draw visualization for fingers and distance"
    # Draw circles on finger tips
    cv2.circle(frame, thumb_pos, 10, (0, 0, 255), -1)  # Red for thumb
    cv2.circle(frame, index_pos, 10, (255, 0, 0), -1)  # Blue for index

    # Draw line between fingers
    cv2.line(frame, thumb_pos, index_pos, (0, 255, 255), 3)

    # Display distance
    cv2.putText(frame, f"Distance: {int(distance)}", (50, 120),
                cv2.FONT_HERSHEY_SIMPLEX, 0.7, (255, 255, 255), 2)
```

```python
# main.py - Main application that uses all modules
from camera_manager import CameraManager
from hand_tracker import HandTracker
import cv2

def main():
    # Initialize components
    camera = CameraManager()
    hand_tracker = HandTracker()

    try:
        # Start camera
        camera.initialize_camera()

        print("✓ Show your hand to the camera...")

        # Main loop
        while True:
            # Read frame from camera
            ret, frame = camera.read_frame()
            if not ret:
                break

            # Detect hands
            results = hand_tracker.detect_hands(frame)

            # Process hand detection
            if results.multi_hand_landmarks:
                for hand_landmarks in results.multi_hand_landmarks:
                    # Draw hand landmarks
                    hand_tracker.draw_hand_landmarks(frame, hand_landmarks)

                    # Get finger positions
                    thumb_pos, index_pos = hand_tracker.get_finger_positions(
                        hand_landmarks, frame.shape
                    )
```

```python
                    # Calculate distance
                    distance = hand_tracker.calculate_distance(thumb_pos, index_pos)

                    # Draw visualization
                    hand_tracker.draw_finger_visualization(
                        frame, thumb_pos, index_pos, distance
                    )

            # Display UI
            cv2.putText(frame, "Hand Gesture Volume Control", (50, 50),
                        cv2.FONT_HERSHEY_SIMPLEX, 1, (255, 255, 255), 2)
            cv2.putText(frame, "Press 'q' to quit", (50, 80),
                        cv2.FONT_HERSHEY_SIMPLEX, 0.7, (255, 255, 255), 2)

            # Show frame
            camera.display_frame(frame)

            # Check for exit
            if camera.check_exit_key():
                break

    except Exception as e:
        print(f"Error: {e}")

    finally:
        # Clean up
        camera.release_camera()
        print("✓ Application closed successfully!")

if __name__ == "__main__":
    main()
```

Figure 3: Implementation Of The Code