Touchless Gesture Control Interface for Media Playback

Ali Zekai Deveci
Student Number: 60291

January 19, 2026

1 Introduction

This report documents the design and implementation of a **Touchless Gesture Control Interface** that allows users to control media playback using hand gestures captured by a webcam. The system replaces traditional input devices with computer vision-based interaction, focusing on accessibility, ergonomics, and intuitive mapping.

2 Updated Project Assumptions

The initial proposal has been realized with the following refinements:

- **Gestures Implemented:**
  - *Open Palm*: Play/Pause (Toggle).
  - *Closed Fist*: Pause (Toggle) / Neutral state.
  - *Volume Mode* (Thumb+Index Open): Move Up/Down to adjust volume.
  - *Mute* (Thumb+Index Pinch): Toggle system mute.
  - *Seek Mode* (Index+Middle Merged): Wave Left/Right to seek -10s/+10s.
- **Target Application:** The system was tested primarily on **YouTube** (browser), using keyboard shortcut simulation (Space, k, m, Arrows, j, l). However, it is designed to be generally applicable to any media player that supports standard hotkeys.
- **Robustness:** Added "cooldown" logic and merged finger thresholds to prevent accidental triggers (e.g., when exiting volume mode).

3 System Architecture / Processing Pipeline

The system follows a continuous processing pipeline:

1. **Input Capture:** Webcam captures frames at 30+ FPS.
2. **Preprocessing:** Frames are flipped (mirrored) for intuitive interaction and converted to RGB.
3. **Hand Tracking:** `MediaPipe Hands` processes the frame to extract 21 3D landmarks (x, y, z).
4. **Gesture Recognition:** The `GestureRecognizer` module analyzes landmark geometry (finger states, distances) to classify the current pose.
5. **State Machine & Logic:** The main loop tracks gesture history and "patience" counters to handle state transitions and debouncing.
6. **Action Execution:** If a valid gesture transition is detected, `MediaController` triggers the OS-level input via `pyautogui`.
7. **Feedback:** Visual overlays (text, skeletons, mode indicators) are drawn on the frame and displayed to the user.

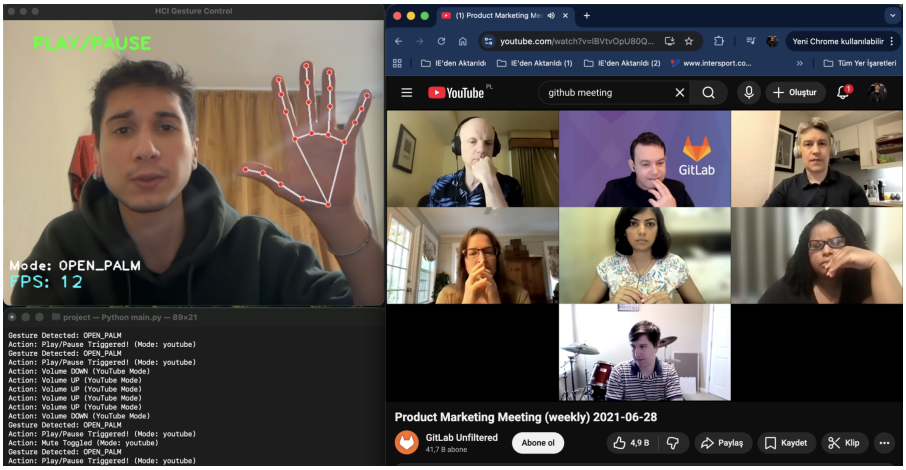

Figure 1: System overview showing the MediaPipe hand skeleton and UI overlays during an Open Palm gesture.

4 Main Algorithms and Technical Approach

4.1 Hand Tracking

The project utilizes Google's MediaPipe for robust hand tracking. It provides landmark coordinates normalized to [0,1].

```
img = tracker.find_hands(img)
lm_list, handedness = tracker.get_landmark_positions(img)
```

4.2 Gesture Classification

Gestures are defined by geometric heuristics:

- **Volume Mode:** Thumb and Index fingers are UP, others DOWN.
- **Seek Mode:** Index and Middle fingers are UP and MERGED (distance < 60px).
- **Mute:** In Volume Mode, if distance between Thumb tip and Index tip < 30px.

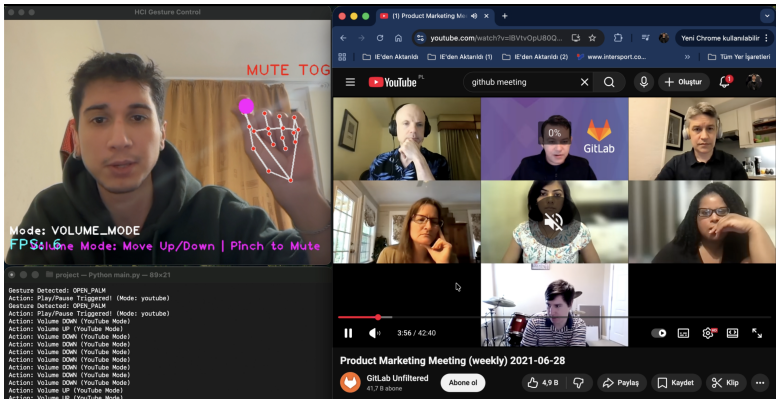

Figure 2: The "Pinch" gesture used to trigger a Mute toggle.

4.3 Smooth Interaction Logic

To prevent jitter and accidental triggers:

- **Debouncing:** Actions like Play/Pause only trigger on the "Rising Edge" (transition from Non-Palm to Palm).
- **Cooldown:** A cooldown timer (5 frames) blocks new gestures immediately after complex modes (Seek/Volume) to prevent "Open Palm" misfires during hand opening.
- **Anchoring:** For continuous controls (Volume/Seek), movement is calculated relative to an initial "anchor" position ($prev\_vol\_y$), which updates dynamically.

5 User Interface Design

The GUI is a real-time video feed with augmented reality elements:

- **Skeleton Overlay:** Shows tracking status.
- **Mode Text:** Displays current detected gesture (e.g., "Mode: SEEK_MODE").
- **Feedback Icons:** Visual circles follow the controlling fingers (Cyan for Seek, Magenta for Volume).
- **Action Logs:** Text hints like "VOL UP" or "SEEK +10s" appear near the hand when an action triggers.

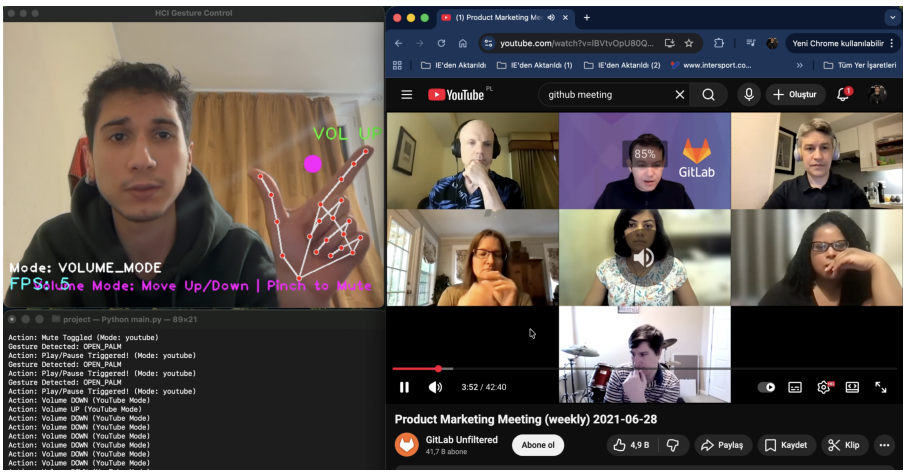

Figure 3: Volume Mode UI: Magenta indicators follow the hand for movement tracking during a Volume Up action.

6 Tools and Technology Stack

The project was implemented using the following technology stack:

- **Programming Language:** Python 3.11
- **Environment Management:** Python `venv` (Virtual Environment)
- **Core Libraries:**
  - **OpenCV (opencv-python):** For real-time computer vision and image processing.
  - **MediaPipe:** For efficient, on-device hand tracking and landmark extraction.
  - **PyAutoGUI:** For programmatic control of the mouse and keyboard to simulate user input.
  - **NumPy:** For efficient numerical operations and geometry calculations.
- **Hardware:** Standard RGB Webcam (built-in or USB).

7 Execution Instructions

7.1 Environment Setup

The project relies on a specific Python environment to ensure compatibility.

```
# 1. Create a virtual environment (Python 3.11 recommended)
python3.11 -m venv venv

# 2. Activate the virtual environment
# On macOS/Linux:
source venv/bin/activate
# On Windows:
# venv\Scripts\activate

# 3. Install dependencies
pip install -r requirements.txt
```

7.2 Running the System

Once the environment is active and dependencies are installed:

```
python3 main.py
```

- Ensure the camera has permission to access the video feed.
- For **YouTube control**, click on the video window once to ensure it has focus before using gestures.
- Press 'q' to exit the application.

8 Example Results Conclusion

The system successfully controls YouTube playback with low latency. The "Seek Mode" allows for precise video navigation, while the "Volume Mode" provides granular audio control. The addition of robustness features (cooldowns, patience) significantly improved the usability by reducing false positives during natural hand movements.

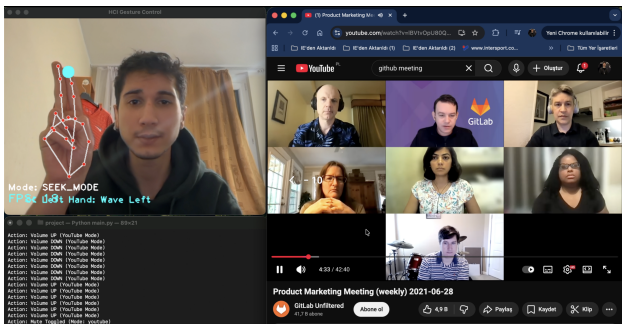

Figure 4: Seek Backward gesture.

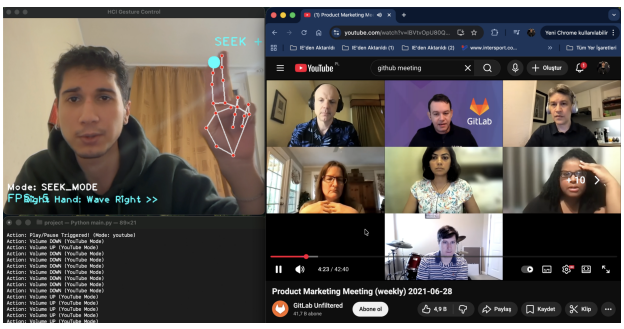

Figure 5: Seek Forward gesture.

9 Project Repository

The complete source code and documentation for this project are available on GitHub:

`https://github.com/alizekaid/Hand-Gesture-Video-Player-Controller`