HUMAN HAND GESTURE CONTROLLED YOUTUBE VIDEO

Prepared by: Raymond V
Course: B. Tech CSE (AIML)

INDEX

1. Introduction:

In the age of increasing digital interaction, the ability to control devices hands-free through gestures offers not only convenience but also enhanced accessibility, especially in environments where physical contact with devices is not practical or desirable. This project, **Gesture-Based YouTube Video Controller**, aims to provide a seamless interaction experience by using hand gestures to control video playback on YouTube. By combining MediaPipe, OpenCV, and Selenium, this project brings together powerful computer vision and browser automation tools to achieve intuitive video control functionalities.

The project allows users to perform actions like play, pause, fast forward, rewind, and volume control by simply using hand gestures, providing a futuristic, touchless interface for video control.

2. Tools Used:

The project makes use of various libraries and tools to achieve real-time gesture recognition and browser automation:

a) OpenCV:

- ➢ Functionality: Open Source Computer Vision Library that helps in capturing video frames from the webcam and processing them.
- ➢ Application: Detects the user's hand in real-time from the webcam feed, flips the frame, and processes the input for gesture recognition.

b) MediaPipe:

- ➢ Functionality: A framework from Google for building machine learning pipelines for perceptual data, such as video and audio.
- ➢ Application: The MediaPipe Hands module detects hand landmarks and tracks the position of each finger in real time. These landmarks are used to identify the hand gestures, making gesture recognition possible.

Real-World Applications:

- ✓ Hand gesture control in VR/AR applications.
- ✓ Sign language recognition for accessibility.
- ✓ Interactive game controls without traditional hardware.

c) Selenium:

- ➢ Functionality: A browser automation tool used for controlling web browsers through programs and performing web-based tasks.

- ➢ Application: Automates the YouTube browser interface, enabling actions like play, pause, fast forward, rewind, and volume control based on the gestures recognized.

Real-World Applications:

- ✓ Automated testing of websites.
- ✓ Web scraping and data extraction.
- ✓ Controlling browsers for automation tasks like logging into accounts, form submissions, etc.

d) Python:

- ➢ Functionality: Python serves as the backbone of the project, integrating the tools mentioned above, handling the logic, and providing an easy-to-use environment for computer vision tasks.

Real-World Applications:

- ✓ Widely used in machine learning and data science.
- ✓ Suitable for automation scripts and applications in various fields.

3. Detailed Workflow:

a) Initialization:

- ➢ The system starts by importing the required libraries— OpenCV for video capture, MediaPipe for hand landmark detection, and Selenium for browser control. Selenium's Chrome WebDriver is initialized, and MediaPipe's Handsmodule is configured for detecting hand gestures.

b) Video Playback Automation:

- ➢ A YouTube video is opened using Selenium WebDriver, and the video is set to start playing automatically. The WebDriver allows sending JavaScript commands to control the YouTube video directly from Python code.
- ➢ The WebDriver is linked to Chrome, and through JavaScript, actions like play, pause, fast forward, rewind are executed in the browser.

c) Hand Gesture Detection:

- ➢ The OpenCV library captures video input from the webcam.
- ➢ The video frames are processed by MediaPipe Hands to detect the 21 landmarks (key points) on the user's hand.
- ➢ These landmarks include positions of the fingertips, palm, and joints. Based on these landmarks, specific hand gestures are recognized by analyzing the positions of fingers (open or closed).

d) Gesture Recognition Logic:

- ➢ A custom function, `recognizeHandGesture`, maps the landmarks to hand gestures. This function identifies different gestures by analyzing the relative positions of the fingers and palm.

Example gestures include:

- o All fingers open (gesture '5'): Pause the video.
- o Thumb closed, all other fingers open (gesture '4'): Play the video.
- o Fast Forward (gesture '8'): Moves the video forward by 5 seconds.
- o Rewind (gesture '9'): Moves the video backward by 5 seconds.
- o Mute/Unmute gestures: Control volume based on specific hand configurations.

e) Action Execution:

- ➢ Once a gesture is recognized, corresponding JavaScript commands are sent to the browser using Selenium to control video playback.
  The actions supported in this project include:

- o Play and Pause the video.
- o Speed control: Increase or reset playback speed.
- o Volume control: Mute/unmute the video.
- o Seek control: Fast forward or rewind the video by specific intervals.

f) Error Handling and Finalization:

- ➢ The system is designed to handle errors (e.g., insufficient landmarks detected) and can exit gracefully when the user presses the Esc key. Upon exiting, resources such as the webcam and browser are closed properly.

4. Applications of Gesture-Based Control:

The concept of gesture-based control has several real-world applications beyond this specific project:

a) Entertainment Systems:

Smart TVs, home theater systems, and media players can be controlled using hand gestures, eliminating the need for remote controls.

b) Accessibility:

Disabled users can benefit greatly from gesture-controlled systems, providing them with alternative ways to interact with media and applications.

c) **Gaming and Virtual Reality (VR):**

   Many gaming systems are now integrating gesture recognition as part of their control systems. This project could be extended into more interactive environments such as VR and AR.

d) **Automotive Controls:**

   Car infotainment systems can be designed to respond to hand gestures, providing distraction-free, touchless control while driving.

e) **Presentation and Educational Tools:**

   In environments like classrooms or meetings, presenters can use gestures to control slideshows, videos, or other multimedia, improving the flow of presentations without requiring a physical controller.

5. Real-World Impact of This Project:

This project has the potential to simplify how users interact with devices, especially in contexts where hands-free control is preferable or necessary. For example:

- ✓ In a household: Someone watching a video while cooking or doing another task could control the video playback without needing to touch their device.
- ✓ For people with disabilities: Hand gestures could replace the need for remote controls or other physical devices.
- ✓ Public spaces: Gesture-controlled screens or kiosks in airports, malls, or museums could enhance public engagement and improve hygiene by reducing the need for physical contact.
- ✓ In workspaces: Hands-free control systems can be used in environments like healthcare, where professionals need to control displays or media without direct contact.

This project represents the next step towards more natural and intuitive human-computer interaction. Gesture recognition systems have the potential to revolutionize how we interact with devices in an increasingly touchless world.

6. Program

```
import cv2
import mediapipe as mp
from func import recognizeHandGesture, getStructuredLandmarks
from selenium import webdriver
from selenium.webdriver.chrome.service import Service
from selenium.webdriver.chrome.options import Options
import time

mp_drawing = mp.solutions.drawing_utils
mp_hands = mp.solutions.hands

# Path to your ChromeDriver
chrome_driver_path = r"D:\Youtube Video controller\chromedriver-win32\chromedriver-win32\chromedriver.exe"
  # Ensure this path is correct

# Configure Selenium to use the correct driver
chrome_service = Service(chrome_driver_path)
chrome_options = Options()

def gest():
    driver = webdriver.Chrome(service=chrome_service, options=chrome_options)
    driver.get('https://www.youtube.com/watch?v=09R8_2nJtjg')
    driver.execute_script('document.getElementsByTagName("video")[0].play()')

    hands = mp_hands.Hands(min_detection_confidence=0.7, min_tracking_confidence=0.5)
    cap = cv2.VideoCapture(0)
    i = 0
    c = 0
    try:
        while cap.isOpened():
            l = []
            i += 1
            success, image = cap.read()
            if not success:
                break
            image = cv2.cvtColor(cv2.flip(image, 1), cv2.COLOR_BGR2RGB)
            image.flags.writeable = False
            results = hands.process(image)
            image.flags.writeable = True
            image = cv2.cvtColor(image, cv2.COLOR_RGB2BGR)
            if results.multi_hand_landmarks:
                for hand_landmarks in results.multi_hand_landmarks:
                    mp_drawing.draw_landmarks(image, hand_landmarks, mp_hands.HAND_CONNECTIONS)
                    c += 1
                    for lp in hand_landmarks.landmark:
                        l.append(lp.x)
                        l.append(lp.y)
            cv2.imshow('MediaPipe Hands', image)

            # Check if 'l' has the expected number of elements before processing
              if len(l) == 42:
                recognizedHandGesture = recognizeHandGesture(getStructuredLandmarks(l))
```

```python
                print(recognizedHandGesture)
                if recognizedHandGesture == 5:
                    driver.execute_script('document.getElementsByTagName("video")[0].pause()')
                elif recognizedHandGesture == 4:
                    driver.execute_script('document.getElementsByTagName("video")[0].play()')
                elif recognizedHandGesture == 3:
                    driver.execute_script('document.getElementsByTagName("video")[0].playbackRate = 2')
                elif recognizedHandGesture == 2:
                    driver.execute_script('document.getElementsByTagName("video")[0].playbackRate = 1')
                elif recognizedHandGesture == 6:
                    driver.execute_script('document.getElementsByTagName("video")[0].volume = 0')
                elif recognizedHandGesture == 7:
                    driver.execute_script('document.getElementsByTagName("video")[0].volume = 1')
                elif recognizedHandGesture == 8:
                    # Fast Forward (All fingers open except the ring finger)
                    driver.execute_script('document.getElementsByTagName("video")[0].currentTime += 5')
                elif recognizedHandGesture == 9:
                    # Rewind (All fingers open except the little finger)
                    driver.execute_script('document.getElementsByTagName("video")[0].currentTime -= 5')
            else:
                print("Insufficient landmarks detected")

            if cv2.waitKey(5) & 0xFF == 27:
                break
    except Exception as e:
        print('Not Recognized:', e)
    finally:
        hands.close()
        cap.release()
        driver.quit()
        cv2.destroyAllWindows()

gest()
```

```python
# func.py

def recognizeHandGesture(landmarks):
    thumbState = 'UNKNOWN'
    indexFingerState = 'UNKNOWN'
    middleFingerState = 'UNKNOWN'
    ringFingerState = 'UNKNOWN'
    littleFingerState = 'UNKNOWN'
    recognizedHandGesture = None

    pseudoFixKeyPoint = landmarks[2]['x']
    if (landmarks[3]['x'] < pseudoFixKeyPoint and landmarks[4]['x'] < landmarks[3]['x']):
        thumbState = 'CLOSE'
    elif (pseudoFixKeyPoint < landmarks[3]['x'] and landmarks[3]['x'] < landmarks[4]['x']):
        thumbState = 'OPEN'

    pseudoFixKeyPoint = landmarks[6]['y']
    if (landmarks[7]['y'] < pseudoFixKeyPoint and landmarks[8]['y'] < landmarks[7]['y']):
        indexFingerState = 'OPEN'
    elif (pseudoFixKeyPoint < landmarks[7]['y'] and landmarks[7]['y'] < landmarks[8]['y']):
        indexFingerState = 'CLOSE'

    pseudoFixKeyPoint = landmarks[10]['y']
    if (landmarks[11]['y'] < pseudoFixKeyPoint and landmarks[12]['y'] < landmarks[11]['y']):
        middleFingerState = 'OPEN'
    elif (pseudoFixKeyPoint < landmarks[11]['y'] and landmarks[11]['y'] < landmarks[12]['y']):
        middleFingerState = 'CLOSE'

    pseudoFixKeyPoint = landmarks[14]['y']
    if (landmarks[15]['y'] < pseudoFixKeyPoint and landmarks[16]['y'] < landmarks[15]['y']):
        ringFingerState = 'OPEN'
    elif (pseudoFixKeyPoint < landmarks[15]['y'] and landmarks[15]['y'] < landmarks[16]['y']):
        ringFingerState = 'CLOSE'

    pseudoFixKeyPoint = landmarks[18]['y']
    if (landmarks[19]['y'] < pseudoFixKeyPoint and landmarks[20]['y'] < landmarks[19]['y']):
        littleFingerState = 'OPEN'
    elif (pseudoFixKeyPoint < landmarks[19]['y'] and landmarks[19]['y'] < landmarks[20]['y']):
        littleFingerState = 'CLOSE'

    if (thumbState == 'OPEN' and indexFingerState == 'OPEN' and middleFingerState == 'OPEN' and
ringFingerState == 'OPEN' and littleFingerState == 'OPEN'):
        recognizedHandGesture = 5
    elif (thumbState == 'CLOSE' and indexFingerState == 'OPEN' and middleFingerState == 'OPEN' and
ringFingerState == 'OPEN' and littleFingerState == 'OPEN'):
        recognizedHandGesture = 4
    elif (thumbState == 'OPEN' and indexFingerState == 'OPEN' and middleFingerState == 'OPEN' and
ringFingerState == 'CLOSE' and littleFingerState == 'CLOSE'):
        recognizedHandGesture = 3
    elif (thumbState == 'OPEN' and indexFingerState == 'OPEN' and middleFingerState == 'CLOSE' and
ringFingerState == 'CLOSE' and littleFingerState == 'CLOSE'):
        recognizedHandGesture = 2
    elif (thumbState == 'OPEN' and indexFingerState == 'CLOSE' and middleFingerState == 'OPEN' and
ringFingerState == 'CLOSE' and littleFingerState == 'CLOSE'):
        recognizedHandGesture = 10
    elif (thumbState == 'OPEN' and indexFingerState == 'OPEN' and middleFingerState == 'CLOSE' and
ringFingerState == 'CLOSE' and littleFingerState == 'OPEN'):
        recognizedHandGesture = 7
    elif (thumbState == 'OPEN' and indexFingerState == 'CLOSE' and middleFingerState == 'CLOSE' and
ringFingerState == 'CLOSE' and littleFingerState == 'OPEN'):
        recognizedHandGesture = 6
    elif (thumbState == 'OPEN' and indexFingerState == 'OPEN' and middleFingerState == 'OPEN' and
ringFingerState == 'CLOSE' and littleFingerState == 'OPEN'):
        recognizedHandGesture = 8  # Fast forward
    elif (thumbState == 'OPEN' and indexFingerState == 'OPEN' and middleFingerState == 'OPEN' and
ringFingerState == 'OPEN' and littleFingerState == 'CLOSE'):
        recognizedHandGesture = 9  # Rewind
    else:
        recognizedHandGesture = 0

    return recognizedHandGesture

def getStructuredLandmarks(landmarks):
    """
    Convert the flat list of landmarks to a structured format.
    """
    if len(landmarks) != 42:
        raise ValueError("Landmarks list must contain exactly 42 elements")

    structured_landmarks = []
    for i in range(0, len(landmarks), 2):
        point = {'x': landmarks[i], 'y': landmarks[i + 1]}
        structured_landmarks.append(point)

    return structured_landmarks
```

7. Conclusion:

The **Gesture-Based YouTube Video Controller** demonstrates a significant application of modern computer vision and automation tools. By integrating OpenCV for capturing live video, MediaPipe for real-time gesture detection, and Selenium for controlling web content, the project bridges the gap between user gestures and media control. The ability to control video playback using hand gestures introduces a new level of convenience, accessibility, and innovation, proving that gesture-based systems are not only the future but also a valuable tool for present-day applications.

This project lays the groundwork for further exploration in the field of gesture recognition, which can be extended to broader domains like entertainment, healthcare, and public interfaces.