CODE

```
import cv2
import mediapipe as mp
import pyautogui
import time

# Initialize MediaPipe hands module
mp_hands = mp.solutions.hands
mp_drawing = mp.solutions.drawing_utils

# Set up the webcam
cap = cv2.VideoCapture(0)

# Initialize the hands model
with mp_hands.Hands(max_num_hands=1, min_detection_confidence=0.5) as hands:
  while cap.isOpened():
    success, image = cap.read()
    if not success:
      print("Ignoring empty camera frame.")
      continue

    # Flip the image horizontally for a later selfie-view display
    image = cv2.flip(image, 1)

    # Convert the BGR image to RGB
    image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

    # Process the image and find hands
    results = hands.process(image_rgb)

    # Draw hand landmarks on the image
    if results.multi_hand_landmarks:
      for hand_landmarks in results.multi_hand_landmarks:
        mp_drawing.draw_landmarks(image, hand_landmarks,
mp_hands.HAND_CONNECTIONS)

        # Get landmark positions for gesture recognition
        thumb_tip_y =
hand_landmarks.landmark[mp_hands.HandLandmark.THUMB_TIP].y
        index_tip_y =
hand_landmarks.landmark[mp_hands.HandLandmark.INDEX_FINGER_TIP].y
```

```
            middle_tip_y =
hand_landmarks.landmark[mp_hands.HandLandmark.MIDDLE_FINGER_TIP].y

            # Determine the gesture based on landmark positions
            if thumb_tip_y < index_tip_y and middle_tip_y < index_tip_y:  # Thumb up gesture
                cv2.putText(image, "Gesture: Left Click", (10, 30),
cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 0), 2)
                pyautogui.click()  # Perform left click
                time.sleep(1)  # Prevent multiple clicks in quick succession

            elif index_tip_y < middle_tip_y:  # Peace sign gesture
                cv2.putText(image, "Gesture: Right Click", (10, 30),
cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 0), 2)
                pyautogui.click(button='right')  # Perform right click
                time.sleep(1)  # Prevent multiple clicks in quick succession

            elif thumb_tip_y > index_tip_y and middle_tip_y > index_tip_y:  # Fist gesture
                cv2.putText(image, "Gesture: Scroll", (10, 30), cv2.FONT_HERSHEY_SIMPLEX,
1, (0, 255, 0), 2)
                pyautogui.scroll(-10)  # Scroll down; use positive value for scrolling up
                time.sleep(1)  # Prevent multiple scrolls in quick succession

        # Display the resulting frame
        cv2.imshow('Hand Gesture Recognition', image)

        if cv2.waitKey(5) & 0xFF == 27:  # Press 'ESC' to exit
            break

# Release the webcam and destroy windows
cap.release()
cv2.destroyAllWindows()
```

IMPLEMENTATION

1. Importing Libraries

```
import cv2
import mediapipe as mp
import pyautogui
```

```
import time
```

- **OpenCV (cv2)**: Used for handling the webcam feed and displaying frames with annotations.
- **MediaPipe (mp)**: A framework that provides solutions for hand tracking and gesture recognition.
- **PyAutoGUI (pyautogui)**: Automates mouse actions like clicks and scrolling.
- **time**: Used to add delays between gestures to avoid multiple triggers in quick succession.

2. Initializing MediaPipe Hands Module and Drawing Utilities

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

- `mp_hands`: This initializes the MediaPipe hand tracking module, which can detect and track hands in real-time.
- `mp_drawing`: Provides drawing utilities to visualize the hand landmarks detected by MediaPipe.

3. Setting Up the Webcam

```
cap = cv2.VideoCapture(0)
```

- `cap`: Captures video feed from the default webcam (`0` indicates the default camera). This feed will be processed frame by frame.

4. Initializing and Running the Hand Tracking Model

```
with mp_hands.Hands(max_num_hands=1, min_detection_confidence=0.5) as
hands:
    while cap.isOpened():
```

- `mp_hands.Hands()`: This initializes the hand tracking model.
    - `max_num_hands=1`: Limits the tracking to a single hand.

- `min_detection_confidence=0.5`: The minimum confidence score to detect a hand.

The `while` loop processes each frame as long as the camera is open.

5. Reading and Flipping the Image

```
success, image = cap.read()
if not success:
    print("Ignoring empty camera frame.")
    continue

image = cv2.flip(image, 1)
```

- `cap.read()`: Captures a frame from the webcam.
- `cv2.flip(image, 1)`: Mirrors the image for a natural selfie view so that hand movements match real-time movements.

6. Converting to RGB and Processing with MediaPipe

```
image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
results = hands.process(image_rgb)
```

- `cv2.cvtColor`: Converts the color format from BGR (default in OpenCV) to RGB, required by MediaPipe.
- `hands.process(image_rgb)`: Processes the frame to detect hand landmarks. The `results` object contains the detection details.

7. Drawing Hand Landmarks and Identifying Gestures

```
if results.multi_hand_landmarks:
    for hand_landmarks in results.multi_hand_landmarks:
        mp_drawing.draw_landmarks(image, hand_landmarks,
mp_hands.HAND_CONNECTIONS)
```

- `results.multi_hand_landmarks`: If hands are detected, `multi_hand_landmarks` contains details about each hand's landmarks.
- `mp_drawing.draw_landmarks`: Draws landmarks on the hand and connects them to represent finger and palm positions.

8. Gesture Recognition Logic

Landmark positions are used to define three basic gestures and map them to mouse actions.

```
# Get landmark positions for gesture recognition
thumb_tip_y =
hand_landmarks.landmark[mp_hands.HandLandmark.THUMB_TIP].y
index_tip_y =
hand_landmarks.landmark[mp_hands.HandLandmark.INDEX_FINGER_TIP].y
middle_tip_y =
hand_landmarks.landmark[mp_hands.HandLandmark.MIDDLE_FINGER_TIP].y
```

- Retrieves y-coordinates of specific landmarks on the thumb, index, and middle fingers, used to detect gestures based on their relative positions.

8.1 Thumb-Up Gesture (Left Click)

```
if thumb_tip_y < index_tip_y and middle_tip_y < index_tip_y:
    cv2.putText(image, "Gesture: Left Click", (10, 30),
cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 0), 2)
    pyautogui.click()
    time.sleep(1)
```

- **Condition**: Both thumb and middle finger tips are above the index finger tip.
- **Action**: Displays "Left Click" on the frame and performs a left mouse click using `pyautogui.click()`.
- `time.sleep(1)`: Adds a 1-second delay to prevent multiple clicks.

8.2 Peace Sign Gesture (Right Click)

```
elif index_tip_y < middle_tip_y:
    cv2.putText(image, "Gesture: Right Click", (10, 30),
```

```
cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 0), 2)
    pyautogui.click(button='right')
    time.sleep(1)
```

- **Condition**: The index finger tip is above the middle finger tip, indicating a "peace sign."
- **Action**: Displays "Right Click" and performs a right mouse click using `pyautogui.click(button='right')`.

8.3 Fist Gesture (Scroll Down)

```
elif thumb_tip_y > index_tip_y and middle_tip_y > index_tip_y:
    cv2.putText(image, "Gesture: Scroll", (10, 30),
cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 0), 2)
    pyautogui.scroll(-10)
    time.sleep(1)
```

- **Condition**: Thumb and middle finger tips are below the index finger tip, forming a "fist."
- **Action**: Displays "Scroll" and scrolls down using `pyautogui.scroll(-10)`. Positive values would scroll up.

9. Displaying the Output and Exit Condition

```
cv2.imshow('Hand Gesture Recognition', image)
if cv2.waitKey(5) & 0xFF == 27:  # Press 'ESC' to exit
    break
```

- `cv2.imshow`: Shows the processed video feed with annotations.
- `cv2.waitKey(5) & 0xFF == 27`: Checks if the 'ESC' key is pressed to break the loop and end the program.

10. Releasing Resources

```
cap.release()
cv2.destroyAllWindows()
```

- **Releases the webcam** and **closes any OpenCV windows** to free up resources after the loop ends.