

**ĐẠI HỌC QUỐC GIA HÀ NỘI**

**TRƯỜNG ĐẠI HỌC CÔNG NGHỆ**

-----🙞🙜🕮🙞🙜-----

****

**BÁO CÁO THỰC HÀNH MÔN HỌC**

**TƯƠNG TÁC NGƯỜI – ROBOT**

**BÀI THỰC HÀNH SỐ 5**

**Sinh viên thực hiện**

**Nguyễn Huy Thắng - 22027545**

**Hà Nội, ngày 18 tháng 10 năm 2025**

**Tóm tắt nội dung**

Dự án triển khai một Pipeline Tương tác Phi Ngôn ngữ ba thành phần:

* Module Nhận dạng Cử chỉ: Sử dụng MediaPipe Hands và OpenCV để trích xuất các điểm khớp tay từ webcam, áp dụng phương pháp Rule-Based để nhận dạng 3 cử chỉ: "Dừng lại", "Giơ tay lên" và "Vẫy tay chào" .
* Module Điều khiển Robot: Thiết lập một Webots controller cho robot mô phỏng (ví dụ: Nao/Pepper) để thực hiện các hành vi phi ngôn ngữ như quay hướng, dừng, vẫy tay.
* Module Tích hợp: Sử dụng giao thức Socket TCP/HTTP để truyền mã lệnh điều khiển từ module nhận dạng cử chỉ sang module điều khiển robot, hoàn thành vòng lặp Cử chỉ 🡪 Lệnh 🡪 Hành vi Robot..

1. **Mục tiêu**

* Mục tiêu tổng thể của dự án là xây dựng và tích hợp thành công một pipeline hoàn chỉnh để ánh xạ cử chỉ tay của người dùng sang hành vi phi ngôn ngữ của robot mô phỏng.
* Xây dựng Module Nhận dạng Cử chỉ: Nhận diện ổn định 3 cử chỉ tay cơ bản từ webcam và ánh xạ chúng thành mã lệnh điều khiển.
* Xây dựng Module Điều khiển Robot: Lập trình robot mô phỏng trong Webots để thực hiện các hành vi phi ngôn ngữ (như quay đầu, dừng, vẫy tay) dựa trên lệnh nhận được qua mạng.
* Tích hợp Hệ thống: Thiết lập kênh giao tiếp (Socket/HTTP) để kết nối module nhận dạng và module điều khiển, đạt được khả năng tương tác theo thời gian thực.

1. **Thực hành**
   1. **Phần 1. Nhận dạng cử chỉ tay**
2. **Khởi tạo Mô-đun và Cấu hình Ánh xạ**

import cv2

import mediapipe as mp

# ...

hands = mp\_hands.Hands(

    max\_num\_hands=1,

    min\_detection\_confidence=0.6,

    min\_tracking\_confidence=0.6

)

# Ánh xạ gesture → hành vi

gesture\_map = {

    "raise\_hand": "STOP\_AT\_DISTANCE",

    "open\_hand": "LOOK\_AT\_HUMAN",

    "wave": "WAVE\_BACK"

}

**Giải thích:**

* import cv2, mediapipe as mp, json: Nạp các thư viện cần thiết: OpenCV - xử lý luồng video, MediaPipe - mô hình học sâu để phát hiện khớp tay, và JSON để đóng gói dữ liệu đầu ra.
* hands = mp\_hands.Hands(...): Khởi tạo mô hình MediaPipe Hands.
* max\_num\_hands=1: giúp tối ưu hóa tài nguyên xử lý và đảm bảo chỉ tập trung vào một bàn tay, giảm nhiễu khi áp dụng logic quy tắc.
* min\_detection\_confidence/min\_tracking\_confidence=0.6: Đặt ngưỡng tin cậy cho cả việc phát hiện ban đầu và theo dõi liên tục, cân bằng giữa độ nhạy và tính ổn định.
* gesture\_map = {...}: Thiết lập từ điển ánh xạ chuẩn hóa, đây là giao diện lệnh mà Module Điều khiển Robot sẽ nhận được và xử lý.

1. **Hàm Phân tích Trạng thái Ngón Tay**

def get\_finger\_state(landmarks):

    finger\_open = []

    # Ngón cái (so sánh x)

    if landmarks[4][0] > landmarks[3][0]:

        finger\_open.append(1)

    else:

        finger\_open.append(0)

    # Các ngón còn lại (so sánh y)

    tips = [8, 12, 16, 20]

    pip\_joints = [6, 10, 14, 18]

    for tip, pip in zip(tips, pip\_joints):

        if landmarks[tip][1] < landmarks[pip][1]:

            finger\_open.append(1)

        else:

            finger\_open.append(0)

    return finger\_open

**Giải thích:**

* Logic Trục X cho Ngón cái: Do ngón cái có chuyển động gập/duỗi chủ yếu trên mặt phẳng ngang (trục trong không gian ), việc so sánh tọa độ của đầu ngón (ID 4) với khớp gần lòng bàn tay (ID 3) hiệu quả hơn so với trục .
* Logic Trục Y cho 4 Ngón còn lại: Sử dụng trục để kiểm tra trạng thái duỗi/gập. Nếu tọa độ của đầu ngón (tips) nhỏ hơn tọa độ của khớp giữa (pip\_joints), ngón đó được coi là duỗi (do trục tăng khi đi xuống trong ảnh).

🡪 Chức năng: Chuyển đổi dữ liệu tọa độ khớp MediaPipe thành một mảng trạng thái Boolean (1 là mở, 0 là đóng), cung cấp dữ liệu đầu vào đơn giản cho logic nhận dạng cử chỉ.

1. **Khởi tạo động cơ bánh xe**

def detect\_gesture(landmarks):

    global previous\_x, movement\_counter, last\_wave\_time

    finger\_state = get\_finger\_state(landmarks)

    # Open hand: tất cả ngón mở

    if sum(finger\_state) == 5:

        return "open\_hand", 0.95

    # Raise hand: bàn tay ở nửa trên màn hình

    wrist\_y = landmarks[0][1]

    if wrist\_y < 0.4:  # tùy thuộc độ cao webcam

        return "raise\_hand", 0.90

    # Wave: phát hiện chuyển động qua trái/phải

    hand\_x = landmarks[0][0]

    if previous\_x is not None:

        if abs(hand\_x - previous\_x) > 0.05:

            movement\_counter += 1

            last\_wave\_time = time.time()

    previous\_x = hand\_x

    # Nếu tay vẫy qua lại nhiều lần trong thời gian ngắn

    if movement\_counter >= 4 and (time.time() - last\_wave\_time) < 1.5:

        movement\_counter = 0

        return "wave", 0.87

    return None, 0

**Giải thích:**

* Open Hand (Tĩnh): Áp dụng quy tắc hình dạng sum(finger\_state) == 5. Đây là cử chỉ tĩnh (pose-based) với độ tin cậy cao.
* Raise Hand (Ngữ cảnh/Vị trí): Sử dụng thông tin ngữ cảnh là vị trí của cổ tay (landmark ID 0). Việc sử dụng ngưỡng giúp phân biệt hành động "Giơ tay lên" với hành động "Mở tay" ở vị trí thấp hơn.
* Wave (Động): Đây là logic nhận dạng cử chỉ động (dynamic gesture).
* Sử dụng biến toàn cục để theo dõi sự tích lũy chuyển động qua nhiều khung hình.
* Ngưỡng : Xác định một dao động ngang đáng kể của cổ tay.
* Ngưỡng lần trong giây: Đảm bảo tính liên tục và tốc độ vẫy tay trước khi xác nhận cử chỉ, giúp loại bỏ nhiễu từ các chuyển động ngẫu nhiên.

1. **Vòng lặp chính và xuất dữ liệu**

  cap = cv2.VideoCapture(0)

print("Đang khởi động nhận dạng cử chỉ tay... (Nhấn 'q' để thoát)")

while True:

    ret, frame = cap.read()

    if not ret:

        break

    frame = cv2.flip(frame, 1)

    rgb = cv2.cvtColor(frame, cv2.COLOR\_BGR2RGB)

    results = hands.process(rgb)

    if results.multi\_hand\_landmarks:

        for hand\_landmarks in results.multi\_hand\_landmarks:

            h, w, \_ = frame.shape

            landmarks = []

            for lm in hand\_landmarks.landmark:

                landmarks.append([lm.x, lm.y])

            mp\_drawing.draw\_landmarks(frame, hand\_landmarks, mp\_hands.HAND\_CONNECTIONS)

            gesture, conf = detect\_gesture(landmarks)

            if gesture:

                command = gesture\_map[gesture]

                data = {"gesture": gesture, "confidence": round(conf, 2), "command": command}

                print(json.dumps(data, ensure\_ascii=False))

                cv2.putText(frame, f"{gesture} ({conf:.2f})", (10, 50),

                            cv2.FONT\_HERSHEY\_SIMPLEX, 1.2, (0, 255, 0), 3)

    cv2.imshow("Gesture Recognition", frame)

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

        break

cap.release()

cv2.destroyAllWindows()

**Giải thích:**

* command = gesture\_map[gesture]: Tra cứu lệnh hành động, Chuyển kết quả nhận dạng ("wave") thành mã lệnh chuẩn hóa ("WAVE\_BACK").
* data = {...}: Đóng gói dữ liệu cấu trúc, tạo một từ điển Python chứa lệnh, cử chỉ, và độ tin cậy.
* print(json.dumps(data, ...)): Chuyển từ điển thành chuỗi JSON (command packet) và in ra console.
* cv2.putText(...): Hiển thị kết quả nhận dạng lên màn hình webcam để người dùng theo dõi.

**Kết quả**

[Video kết quả](https://drive.google.com/file/d/1r5B87LODIltsqflYbtu1sNF65qt4fY6T/view?usp=sharing)

* Video thấy kết quả chạy đúng chương trình theo yêu cầu bài toán.
* Module đã hoàn thành mục tiêu. Logic Rule-Based kết hợp với việc phân tích vị trí và chuyển động đã tạo ra một hệ thống nhận dạng cử chỉ nhẹ, phù hợp cho pipeline tương tác thời gian thực.
* Chuẩn bị cho Phần 2 (Điều khiển) cần xây dựng môi trường và robot phản hồi với các cử chỉ nhận diện được ở mục trên.
  1. **Phần 2. Mô-đun điều khiển robot trong Webots**

Ở đây ta sẽ sử dụng robot Tiago bởi vì các đặc tính gần như humanoid robot và Tiago sử dụng bánh xe để di chuyển, ta có thể dễ dàng thực hiện các tác vụ di chuyển, quay trái, phải phù hợp với mục đích của bài thực hành này.



*Hình ảnh robot Tiago trong môi trường Webots*

1. **Khởi tạo thiết bị và cấu hình motor**

# ... (Phần Bánh xe)

wheel\_left = robot.getDevice("wheel\_left\_joint")

wheel\_right = robot.getDevice("wheel\_right\_joint")

# ... (Phần Đầu - Gaze)

possible\_yaw = ["head\_1\_joint", "head\_pan\_joint", "head\_yaw\_joint"]

# ... (logic tìm kiếm)

# ... (Phần Tay phải - Vẫy)

r\_shoulder\_pitch = robot.getDevice("arm\_5\_joint")

r\_elbow\_roll = robot.getDevice("arm\_4\_joint")

# ... (Thiết lập chung cho motor)

for m in motors:

    if m:

        m.setPosition(float('inf'))

        m.setVelocity(0.0)

**Giải thích:**

* robot.getDevice(...): Lấy thẻ định danh (tag) của các motor cần thiết để điều khiển.
* Motor Bánh xe: Tên chuẩn (wheel\_left\_joint, wheel\_right\_joint) được sử dụng cho các hành vi di chuyển và dừng.
* Khớp Đầu (Gaze): Sử dụng logic tìm kiếm linh hoạt (possible\_yaw, possible\_pitch) để tìm khớp đầu (như head\_1\_joint hay head\_2\_joint), đảm bảo code hoạt động trên các biến thể khác nhau của mô hình TIAGo.
* Khớp Tay: Xác định các khớp cụ thể (arm\_5\_joint - Vai, arm\_4\_joint - Khuỷu tay) để thực hiện hành vi biểu cảm.
* m.setPosition(float('inf')) đặt motor vào chế độ điều khiển vận tốc (velocity control) thay vì điều khiển vị trí (position control). Điều này bắt buộc đối với motor bánh xe và hữu ích cho motor tay để điều chỉnh vận tốc chuyển động.
* Chức năng: Thiết lập giao diện điều khiển phần cứng ảo, cho phép điều khiển vận tốc di chuyển và vị trí góc quay của các khớp..

1. **Hàm thực hiện hành động (perform\_action)**

Hàm này chứa logic cốt lõi cho việc ánh xạ lệnh nhận được sang chuỗi hành động của robot.

1. **Hành vi di chuyển (MOVE\_FORWARD, STOP, STOP\_AT\_DISTANCE)**

# ... (MOVE\_FORWARD)

wheel\_left.setVelocity(MAX\_VELOCITY)

wheel\_right.setVelocity(MAX\_VELOCITY)

# ... (STOP\_AT\_DISTANCE)

wheel\_left.setVelocity(MAX\_VELOCITY)

wheel\_right.setVelocity(MAX\_VELOCITY)

robot.step(150 \* TIME\_STEP)

wheel\_left.setVelocity(0.0)

wheel\_right.setVelocity(0.0)

**Giải thích**

* Điều khiển Vận tốc: Các lệnh di chuyển được thực hiện bằng cách đặt vận tốc dương (tiến), âm (lùi) hoặc khác dấu (quay) cho hai motor bánh xe.
* Dừng Giả lập (STOP\_AT\_DISTANCE): Hành vi này mô phỏng việc robot di chuyển tiến về phía người dùng và dừng lại ở một khoảng cách.
* Lệnh robot.step(150 \* TIME\_STEP) là cốt lõi: nó giữ robot ở trạng thái di chuyển trong bước thời gian mô phỏng, sau đó đặt vận tốc về để dừng lại.
* Chức năng: Chuyển đổi lệnh điều khiển thành vận tốc robot, mô phỏng hành vi di chuyển trong môi trường 3D.

1. **Hành vi biểu cảm (WAVE\_BACK)**

# ...

# 1. Đặt khớp vai (arm\_5\_joint) vào vị trí giơ cao cố định

r\_shoulder\_pitch.setPosition(-1.5)

robot.step(50 \* TIME\_STEP)

# 2. Tạo dao động vẫy bằng khớp khuỷu tay (arm\_4\_joint)

r\_elbow\_roll.setVelocity(2.0)

for \_ in range(5):

    r\_elbow\_roll.setPosition(wave\_pos1)

    robot.step(15 \* TIME\_STEP)

    r\_elbow\_roll.setPosition(wave\_pos2)

    robot.step(15 \* TIME\_STEP)

# ... (Về vị trí ban đầu)

**Giải thích**

* Chuyển động Phức hợp: Hành vi vẫy tay được tạo ra bằng cách kết hợp chuyển động của hai khớp để đạt được tính tự nhiên.
* Khớp Vai (arm\_5\_joint): Được đặt cố định ở góc rad để giơ cánh tay lên.
* Khớp Khuỷu tay (r\_elbow\_roll): Thực hiện chuyển động dao động nhanh giữa hai vị trí ( và rad). Tốc độ cao ) và bước thời gian ngắn () được sử dụng để mô phỏng chuyển động vẫy tay nhanh.
* Chức năng: Thực hiện cử chỉ biểu cảm bằng cách điều khiển tuần tự nhiều khớp trong thời gian ngắn, sau đó đưa robot về tư thế nghỉ.

1. **Hành vi phi ngôn ngữ (NOD, LOOK\_AT\_HUMAN)**

# ... (NOD)

head\_pitch.setPosition(0.25)    # cúi xuống

robot.step(20 \* TIME\_STEP)

head\_pitch.setPosition(-0.15)   # ngẩng lên

# ... (LOOK\_AT\_HUMAN)

head\_yaw.setPosition(0.5)       # quay sang trái

robot.step(40 \* TIME\_STEP)

head\_yaw.setPosition(-0.5)      # quay sang phải

**Giải thích:**

* NOD: Được thực hiện bằng cách điều khiển khớp head\_pitch di chuyển tuần tự giữa các vị trí góc khác nhau.
* LOOK\_AT\_HUMAN: Sử dụng khớp head\_yaw và head\_pitch để tạo chuyển động quét đầu sang hai bên, mô phỏng hành động "tìm kiếm" hoặc "hướng sự chú ý" về phía người dùng.
* Chức năng: Điều khiển động học đầu robot để thể hiện hành vi phi ngôn ngữ cơ bản.

1. **Socket Server và vòng lặp chính**

def socket\_server():

    # ... (Thiết lập Host/Port)

    server.listen(1)

    # ...

    while True:

        client, addr = server.accept()

        data = client.recv(1024).decode()

        if data:

            command = json.loads(data)

            perform\_action(command["command"])

        client.close()

# ...

threading.Thread(target=socket\_server, daemon=True).start()

# ...

while robot.step(TIME\_STEP) != -1:

    pass

**Giải thích:**

* Thiết lập Server Socket: Tạo một server socket (socket.AF\_INET, socket.SOCK\_STREAM) lắng nghe trên 127.0.0.1:5005.
* server.setsockopt(socket.SOL\_SOCKET, socket.SO\_REUSEADDR, 1): đảm bảo cổng có thể được sử dụng lại ngay lập tức.
* Luồng Riêng (Threading): Hàm socket\_server được chạy trong một luồng riêng biệt (threading.Thread). Lý do: Điều này cho phép server lắng nghe kết nối liên tục (blocking server.accept()) mà không làm tạm dừng vòng lặp mô phỏng Webots chính.
* Xử lý Lệnh: Khi nhận dữ liệu, nó được giải mã, chuyển đổi từ JSON sang từ điển Python (json.loads), sau đó gọi hàm perform\_action().
* Vòng lặp Chính Webots: Lệnh while robot.step(TIME\_STEP) != -1: là bắt buộc trong mọi controller Webots. Nó tiến hành mô phỏng qua từng bước thời gian (), cập nhật trạng thái motor và cảm biến.
* Chức năng: Thiết lập giao tiếp không đồng bộ giữa Webots và module bên ngoài, đảm bảo mô phỏng chạy mượt mà trong khi chờ lệnh điều khiển.

**Kết quả**

[Video kết quả](https://drive.google.com/file/d/160X6W_r_w0AV5nkkjSCzguu3Z-SIu1VM/view?usp=sharing)

* Tiago được kết nốt Socket TCP thành công và đã chờ được nhận lệnh.
* Khi được nhận lệnh Tiago đã in ra command nhận được và thực hiện mệnh lệnh với độ trễ thấp. Đầu ra robot đã thực hiện hành vi như mong muốn
* Kết quả phần này là sự chuẩn bị cho pipeline cho phần 3, nhận diện cử chỉ tay và truyền thông cho robot thực hiện.
  1. **Phần 3. Tích hợp pipeline**

Khi đã có nhận diện cử chỉ, và đã có thể truyền câu lệnh vào robot ta có thể tiến hành gộp hai phần này lại để xây dựng điều khiển hành vi phi ngôn ngữ của robot thông qua cử chỉ tay

1. **Khỏi tạo MediaPipe Hands**

# 1. Khởi tạo MediaPipe Hands

mp\_hands = mp.solutions.hands

mp\_drawing = mp.solutions.drawing\_utils

hands = mp\_hands.Hands(

   max\_num\_hands=1,

   min\_detection\_confidence=0.6,

   min\_tracking\_confidence=0.6

)

**Giải thích:**

* mp\_hands = mp.solutions.hands: Nhập module hands từ MediaPipe Solutions.
* mp\_drawing = mp.solutions.drawing\_utils: Nhập module giúp vẽ các điểm mốc (landmarks) và kết nối trên ảnh.
* mp\_hands.Hands(...): Khởi tạo đối tượng Hands.
* max\_num\_hands=1: Giới hạn chỉ nhận dạng một bàn tay duy nhất.
* min\_detection\_confidence=0.6: Ngưỡng tin cậy tối thiểu để phát hiện bàn tay (60%).
* min\_tracking\_confidence=0.6: Ngưỡng tin cậy tối thiểu để theo dõi bàn tay sau khi đã phát hiện (60%).

1. **Ánh xạ cử chỉ thành hành vi**

# 2. Ánh xạ gesture → hành vi

# =============================

gesture\_map = {

  "raise\_hand": "STOP\_AT\_DISTANCE",

 "open\_hand": "LOOK\_AT\_HUMAN",

"wave": "WAVE\_BACK"

}

**Giải thích:**

* gesture\_map = { ... }: Một từ điển (dictionary) ánh xạ tên cử chỉ (key) sang lệnh điều khiển robot (value).
* "raise\_hand": "STOP\_AT\_DISTANCE": Khi người dùng giơ tay cao, robot sẽ dừng lại ở một khoảng cách an toàn.
* "open\_hand": "LOOK\_AT\_HUMAN": Khi người dùng mở lòng bàn tay, robot sẽ nhìn về phía người (xoay đầu/camera).
* "wave": "WAVE\_BACK": Khi người dùng vẫy tay, robot sẽ vẫy tay lại (tương tác xã hội).

1. **Socket gửi lệnh đến robot\_controller**

# 3. Socket gửi lệnh đến robot\_controller

# =============================

HOST = "127.0.0.1"

PORT = 5005

def send\_to\_robot(command):

    try:

        s = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

        s.connect((HOST, PORT))

        msg = json.dumps({"command": command})

        s.send(msg.encode("utf-8"))

        s.close()

        print(f"[SOCKET] Sent command: {command}")

    except Exception as e:

        print(f"[SOCKET ERROR] {e}")

**Giải thích:**

* HOST = "127.0.0.1": Địa chỉ IP của máy chủ (robot controller), ở đây là localhost (máy tính hiện tại).
* PORT = 5005: Cổng mạng mà chương trình điều khiển robot (robot\_controller) đang lắng nghe.
* ef send\_to\_robot(command): Hàm chính thực hiện việc gửi lệnh:
* s = socket.socket(...): Tạo một socket mới.
* s.connect((HOST, PORT)): Thiết lập kết nối đến robot controller.
* msg = json.dumps({"command": command}): Đóng gói lệnh thành một chuỗi JSON để dễ dàng phân tích cú pháp ở phía nhận.
* s.send(msg.encode("utf-8")): Gửi dữ liệu qua kết nối.
* s.close(): Đóng kết nối sau khi gửi.
* try...except: Xử lý lỗi nếu kết nối thất bại (ví dụ: robot controller chưa chạy).

1. **Phát hiện trạng thái ngón tay và cử chỉ**
2. **Biến toàn cục và Cooldown**

previous\_x = None

movement\_counter = 0

last\_wave\_time = 0

last\_sent = None

cooldown = 2.0  # giãn cách 2 giây giữa các lệnh

**Giải thích:**

* + - revious\_x, movement\_counter: Dùng để theo dõi chuyển động ngang của bàn tay qua các khung hình để phát hiện cử chỉ vẫy tay (wave).
    - last\_wave\_time: Ghi lại thời điểm vẫy tay cuối cùng.
    - last\_sent: Ghi lại thời điểm lệnh cuối cùng được gửi.
    - cooldown = 2.0: Khoảng thời gian giãn cách tối thiểu (2 giây) giữa hai lần gửi lệnh liên tiếp, tránh gửi lệnh quá nhanh (spam).

1. **Hàm get\_finger\_state (landmarks)**

def get\_finger\_state(landmarks):

    finger\_open = []

    # Ngón cái

    finger\_open.append(1 if landmarks[4][0] > landmarks[3][0] else 0)

    # Các ngón còn lại

    tips = [8, 12, 16, 20]

    pip\_joints = [6, 10, 14, 18]

    for tip, pip in zip(tips, pip\_joints):

        finger\_open.append(1 if landmarks[tip][1] < landmarks[pip][1] else 0)

    return finger\_open

**Giải thích:**

* + - landmarks: Danh sách 21 điểm mốc của bàn tay (tọa độ x, y).
    - finger\_open = []: Danh sách chứa trạng thái mở (1) hoặc đóng (0) của 5 ngón tay.
    - Logic ngón cái (landmarks[4][0] > landmarks[3][0]): Kiểm tra vị trí x của đầu ngón cái (4) so với khớp giữa (3) để xác định ngón cái có mở ra không.
    - Logic các ngón còn lại (landmarks[tip][1] < landmarks[pip][1]): Kiểm tra vị trí y của đầu ngón (tip, ví dụ: 8) so với khớp gần kề (PIP, ví dụ: 6). Trong hệ tọa độ hình ảnh, Y nhỏ hơn có nghĩa là vị trí cao hơn, tức là ngón tay duỗi thẳng.
    - return finger\_open: Trả về một mảng binary (ví dụ: [1, 1, 1, 1, 1] cho Mở tay).

1. **Hàm detect\_gesture (landmarks)**

def detect\_gesture(landmarks):

    global previous\_x, movement\_counter, last\_wave\_time

    finger\_state = get\_finger\_state(landmarks)

    # Mở tay (open hand)

    if sum(finger\_state) == 5:

        return "open\_hand", 0.95

    # Giơ tay cao

    wrist\_y = landmarks[0][1]

    if wrist\_y < 0.4:

        return "raise\_hand", 0.9

    # Vẫy tay (wave)

    hand\_x = landmarks[0][0]

    if previous\_x is not None:

        if abs(hand\_x - previous\_x) > 0.05:

            movement\_counter += 1

            last\_wave\_time = time.time()

    previous\_x = hand\_x

    if movement\_counter >= 4 and (time.time() - last\_wave\_time) < 1.5:

        movement\_counter = 0

        return "wave", 0.87

    return None, 0

**Giải thích:**

* + - Mở tay (open\_hand): if sum(finger\_state) == 5: Nếu cả 5 ngón tay đều mở (tổng bằng 5).
    - Giơ tay cao (raise\_hand): if wrist\_y < 0.4: Kiểm tra vị trí Y của cổ tay (landmark 0). Nếu Y nhỏ hơn 0.4 - tương đối cao trên màn hình, xác định là giơ tay cao.
    - Vẫy tay (wave): if abs(hand\_x - previous\_x) > 0.05: Theo dõi chuyển động ngang đáng kể của bàn tay.
    - movement\_counter += 1: Đếm số lần chuyển động ngang trong một khoảng thời gian ngắn.
    - if movement\_counter >= 4 and (time.time() - last\_wave\_time) < 1.5: Nếu có ít nhất 4 lần chuyển động trong vòng 1.5 giây, xác nhận là vẫy tay.
    - return None, 0: Nếu không khớp với cử chỉ nào, trả về None.

1. **Chạy webcam và gửi tín hiệu**

def main(video\_index=0):

    global last\_sent

    cap = cv2.VideoCapture(video\_index)

    print(" Hệ thống nhận dạng cử chỉ tay đang khởi động... (Nhấn 'q' để thoát)")

    while True:

        ret, frame = cap.read()

        if not ret:

            break

        frame = cv2.flip(frame, 1)

        rgb = cv2.cvtColor(frame, cv2.COLOR\_BGR2RGB)

        results = hands.process(rgb)

        if results.multi\_hand\_landmarks:

            for hand\_landmarks in results.multi\_hand\_landmarks:

                h, w, \_ = frame.shape

                landmarks = [[lm.x, lm.y] for lm in hand\_landmarks.landmark]

                mp\_drawing.draw\_landmarks(frame, hand\_landmarks, mp\_hands.HAND\_CONNECTIONS)

                gesture, conf = detect\_gesture(landmarks)

                if gesture:

                    command = gesture\_map[gesture]

                    now = time.time()

                    # tránh gửi quá dày (spam socket)

                    if not last\_sent or now - last\_sent > cooldown:

                        data = {"gesture": gesture, "confidence": round(conf, 2), "command": command}

                        print(json.dumps(data, ensure\_ascii=False))

                        send\_to\_robot(command)

                        last\_sent = now

                    cv2.putText(frame, f"{gesture} ({conf:.2f})", (10, 50),

                                cv2.FONT\_HERSHEY\_SIMPLEX, 1.2, (0, 255, 0), 3)

        cv2.imshow("Gesture Recognition → Robot Control", frame)

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

            break

    cap.release()

    cv2.destroyAllWindows()

**Giải thích:**

* cap = cv2.VideoCapture(video\_index): Khởi tạo camera.
* while True: : Vòng lặp liên tục để đọc từng khung hình video.
* frame = cv2.flip(frame, 1): Lật khung hình theo chiều ngang để người dùng thấy hình ảnh phản chiếu như trong gương.
* results = hands.process(rgb): Xử lý khung hình bằng MediaPipe Hands để tìm ra bàn tay và các điểm mốc.
* if results.multi\_hand\_landmarks: : Nếu có bàn tay được phát hiện.
* mp\_drawing.draw\_landmarks(...): Vẽ các điểm mốc và đường kết nối lên khung hình.
* gesture, conf = detect\_gesture(landmarks): Gọi hàm nhận dạng cử chỉ.
* if not last\_sent or now - last\_sent > cooldown: Kiểm tra Cooldown: Chỉ gửi lệnh nếu đã đủ thời gian giãn cách.
* send\_to\_robot(command): Gọi hàm socket để gửi lệnh.
* cv2.putText(...): Hiển thị tên cử chỉ và độ tin cậy lên màn hình.
* cv2.imshow(...): Hiển thị khung hình đã xử lý.
* if cv2.waitKey(1) & 0xFF == ord('q'): break: Thoát khỏi chương trình khi nhấn phím 'q'.

1. **Entry point**

if \_\_name\_\_ == "\_\_main\_\_":

    parser = argparse.ArgumentParser()

    parser.add\_argument("--video", type=int, default=0, help="Chọn camera index (mặc định 0)")

    args = parser.parse\_args()

    main(args.video)

**Giải thích:**

* if \_\_name\_\_ == "\_\_main\_\_": Đảm bảo rằng mã bên trong chỉ chạy khi file được thực thi trực tiếp.
* parser = argparse.ArgumentParser(): Thiết lập bộ phân tích đối số dòng lệnh.
* parser.add\_argument(...): Cho phép người dùng chỉ định chỉ số camera (--video mặc định là 0).
* main(args.video): Bắt đầu vòng lặp camera chính với chỉ số camera đã chọn.

**Kết quả**

[Video kết quả](https://drive.google.com/file/d/1r-Ots3nKvKeBWjgu5746zkLFp3kCr-AS/view?usp=sharing)

* Robot đã thực hiện cử chỉ như mong muốn.
* Khi người dùng tay giơ quá màn hình, phần mềm sẽ nhận định hành động giơ tay, robot di chuyển sau đó dừng lại cách 1.5m.
* Khi người dùng mở tay, phần mềm nhận diện và truyền hành động robot quay mặt về phía người.
* Khi vẫy tay trong vòng 4 giây, phần mềm nhận diện vẫy tay và robot đã đáp lại bằng động tác vẫy tay.