Hand Gesture Detection by Mediapipe and OpenCV

A. Giới thiệu

1. **Các thành viên**
   - **Lê Quang Thật - SE183256**
   - **Văn Kiệt Khải - SE183848**
   - **Nguyễn Mai Phú Lộc - SE182042**
2. **Giới thiệu chung**

   Hand Gesture là một hình thức giao tiếp ngôn ngữ hình thể, sử dụng cử chỉ của từng ngón tay và tổ hợp của chúng để tạo ra 1 lời thoại mà người sử dụng muốn truyền đạt. Hand Gesture đóng vai trò quan trọng trong việc thể hiện cảm xúc, ý tưởng và thông tin đối với những người bị khiếm khuyết chức năng nói và nghe.

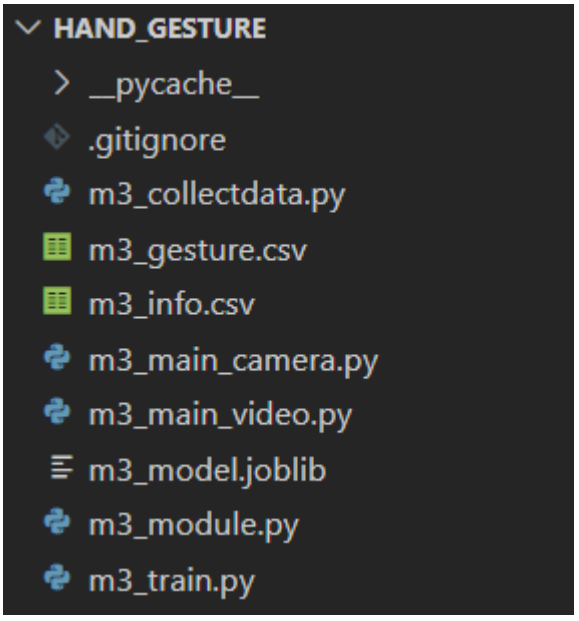
   Ngày nay, Hand Gesture cũng đã được tích hợp vào nhiều thiết bị và hệ thống hiện đại như cảm biến chuyển động và phân tích chúng, từ đó mở ra các ứng dụng mới trong việc điều khiển máy tính, trò chơi điện tử và tương tác thực tế ảo. Ngoài ra, trong văn hóa và nghệ thuật biểu diễn. Các điệu múa, biểu diễn và các hình thức nghệ thuật khác sử dụng Hand Gesture để kể chuyện và truyền tải cảm xúc một cách tinh tế.

B. Ý tưởng và Thuật toán

B.1 Ý tưởng

Ý tưởng ban đầu là chúng em sẽ tạo ra 1 thuật toán cho phép truy cập camera, từ đó chúng em chụp hình và tổng hợp thành 1 dataset chứa dữ liệu về hand gesture (cụ thể là khoảng cách tương đối và góc vector đã được chuẩn hóa) . Sau đó chúng em tiếp tục dựng thêm 1 mô hình có thể train bộ dữ liệu thô mà chúng em đưa vào từ đầu. Bước thứ 3, chúng em bắt đầu dựng landmarks cho bàn tay được hiển thị trên màn hình, từ những thông tin như hướng, góc, độ dài, tọa độ, chúng em có thể biết chính xác bàn tay input trên màn hình có đang giống với bộ dữ liệu đã được train hay không, nếu có nó sẽ trả ra tên của gesture đó. Sau khi hoàn thành bộ khung cho hệ thống này, cuối cùng chúng em viết function cho phép input video vào và đọc hiểu được nó.

B.2 Thuật toán



B.2.1 Các thành phần

m3_collecdata.py - Thu thập data tự động
m3_gesture.csv - Danh sách các cử chỉ
m3_info.csv - File data dùng để dựng mô hình
m3_main_camera.py - Chạy mô hình với camera
m3_main_video.py - Chạy mô hình với video
m3_model.joblib - Mô hình được lưu trữ sau khi dựng
m3_modulte.py - Nơi lưu trữ các lớp xử lý chính
m3_train.py - Dựng mô hình dựa

Bước đầu tiên là nhập tất cả Class từ module ‘m3_module’. Sau đó, lấy tên người dùng từ máy tính và thiết lập các tham số cơ bản của tệp CSV để lưu thông tin (m3_info.csv) và cử chỉ (m3_gesture.csv), màu sắc và kích thước phông chữ hiển thị, và thời gian đếm ngược.

```
username = getpass.getuser()
capture_person = username
csv_file = 'm3_info.csv'
gesture_file = 'm3_gesture.csv'
font = cv2.FONT_HERSHEY_SIMPLEX
font_scale = 1 # From 0 to 1
text_color_1 = (255, 0, 255)
count_down = 0.25 # Set the count down time - seconds
```

Các đối tượng cần thiết cho việc nhận dạng cử chỉ bao gồm 'MediaPipe' để phát hiện tay, 'Camera' để khởi tạo camera, 'DataProcessing' để xử lý dữ liệu và 'DataSaver' để lưu trữ dữ liệu. Bước đầu, chương trình sẽ khởi tạo camera và kiểm tra xem nó có hoạt động hay không. Nếu camera khởi động thành công, chương trình sẽ tiếp tục; nếu không, nó sẽ báo lỗi và thoát.

```
# Create objects
hand_detector = MediaPipe(gl_static_image_mode, 1) # Set 1 hand to create dataset
cam = Camera(gl_cam_index, gl_cam_width, gl_cam_height, gl_cam_fps)
data_processor = DataProcessing()
saving = DataSaver(csv_file)
```

Tiếp theo, chương trình hiển thị danh sách các cử chỉ được thêm sẵn từ tệp 'm3_gesture.csv' và yêu cầu người dùng nhập tên của cử chỉ mà họ muốn ghi lại. Trong vòng lặp chính, chương trình liên tục chụp các khung hình từ camera và xử lý chúng để phát hiện tay dựa trên landmarks. Khi một bàn tay được phát hiện, các đặc điểm của cử chỉ được tính toán và hiển thị trên khung hình.

```
while True:
    frame = cam.get_frame()
    if frame is None:
        print("Error: Cannot receive frame.")
        break

    # Process frame with MediaPipe
    hand_detector.process_image(frame) # Process the frame to detect hands

    # Process the frame and print features
    num_hands = hand_detector.num_detected_hands() # Get the number of detected hands
    if num_hands > 0:
        frame = hand_detector.draw_landmarks(frame) # Draw landmarks on the frame
        landmarks_list = hand_detector.get_hand_landmarks() # Get landmarks of detected hands
        features_list = data_processor.calculate(landmarks_list) # Calculate features

    if sec == 0:
        cv2.putText(frame, "Recording...", (10, 30), font, font_scale, text_color_1, 2, cv2.LINE_AA)
        # Save the features to a CSV file
        if save and num_hands > 0:
            saving.save_data(gesture, capture_person, features_list)
            count += 1
        sec = int(count_down * gl_cam_fps)

    else:
        cv2.putText(frame, f"Count down: {sec}", (10, 30), font, font_scale, text_color_1, 2, cv2.LINE_AA)
        sec -= 1
```

Đoạn code trên liên tục chụp các khung hình từ camera và xử lý chúng để phát hiện bàn tay bằng 'MediaPipe'. Nó vẽ các điểm landmarks trên bàn tay được phát hiện và tính toán các đặc điểm. Tùy thuộc vào đồng hồ đếm ngược, nó sẽ bắt đầu ghi các tính năng vào tệp CSV hoặc hiển thị đồng hồ đếm ngược trên khung. Vòng lặp chạy vô thời hạn cho đến khi xảy ra lỗi khi chụp khung

```
    # Put text on the frame
    cv2.putText(frame, f"Number of hands: {num_hands}", (10, 60), font, font_scale, text_color_1, 2, cv2.LINE_AA)
    cv2.putText(frame, f"Hold 'q' to quit.", (10, 90), font, font_scale, text_color_1, 2, cv2.LINE_AA)
    cv2.putText(frame, f"Gesture: {gesture}", (10, 120), font, font_scale, text_color_1, 2, cv2.LINE_AA)
    cv2.putText(frame, f"Capture person: {capture_person}", (10, 150), font, font_scale, text_color_1, 2, cv2.LINE_AA)
    cv2.putText(frame, f"FPS: {gl_cam_fps}", (10, 180), font, font_scale, text_color_1, 2, cv2.LINE_AA)
    cv2.putText(frame, f"Count: {count}", (10, 240), font, font_scale, text_color_1, 2, cv2.LINE_AA)
    if save:
        cv2.putText(frame, "Press 's' to stop saving.", (10, 210), font, font_scale, text_color_1, 2, cv2.LINE_AA)
    else:
        cv2.putText(frame, "Press 's' to start saving.", (10, 210), font, font_scale, text_color_1, 2, cv2.LINE_AA)

    # Display the frame
    cv2.imshow('Processing frame', frame)

    # Break the loop if 'q' is pressed
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

    # Save toggle button
    if cv2.waitKey(1) & 0xFF == ord('s'):
        save = not save
```

Nếu chế độ lưu được kích hoạt và một bàn tay được phát hiện, các đặc điểm của cử chỉ sẽ được lưu vào tệp CSV. Trong quá trình này, chương trình hiển thị thông tin như số lượng bàn tay được phát hiện, tên cử chỉ, tên người dùng, số lượng khung hình đã lưu và các chi tiết khác trên khung hình để người dùng theo dõi. Người dùng có thể nhấn phím 's' để bật/tắt chế độ lưu và phím 'q' để dừng chương trình.

B.2.2 Create Hand

```
class MediaPipe:
    def __init__(self, static_image_mode, max_num_hands):
        self.static_image_mode = static_image_mode
        self.max_num_hands = max_num_hands
        self.hands = mp.solutions.hands.Hands(static_image_mode, max_num_hands)
        self.draw = mp.solutions.drawing_utils
```

Định Nghĩa Mediapipe bằng cách khởi tạo với chế độ ảnh tĩnh và số lượng tay tối đa. Nó tạo các đối tượng cho việc phát hiện và vẽ bàn tay từ thư viện MediaPipe, giúp xử lý và hiển thị các bàn tay trong hình ảnh hoặc video một cách dễ dàng và hiệu quả. Điều này hỗ trợ trong các ứng dụng liên quan đến nhận diện cử chỉ tay và theo dõi chuyển động.

```
def process_image(self, image):
    # Convert image to RGB format expected by MediaPipe
    image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

    # Process image and get results
    self.results = self.hands.process(image_rgb)

    return self.results        Le Quang That, 4 hours ago • Res
```

Hàm Process_image: được dùng để chuyển đổi ảnh đầu vào sang định dạng RGB rồi sử dụng thư viện MediaPipe để xử lý ảnh, từ đó nhận diện các đặc trưng của bàn tay trong ảnh. Kết quả sau xử lý sẽ được lưu trong biến ‘self.results’ và trả về.

```python
def draw_landmarks(self, image):
    """
    Draw landmarks on the input image using the results from the last processed image.

    Parameters:
    - image: numpy array, the input image in BGR format

    Returns:
    - image: numpy array, the input image with landmarks drawn
    """
    # Check if there are any detected hand landmarks
    if self.results.multi_hand_landmarks:
        # Iterate through each detected hand
        for hand_landmarks in self.results.multi_hand_landmarks:
            # Draw landmarks on the image
            self.draw.draw_landmarks(image, hand_landmarks, mp.solutions.hands.HAND_CONNECTIONS,
                                     self.draw.DrawingSpec(color=(255, 0, 0), thickness=2, circle_radius=3),
                                     self.draw.DrawingSpec(color=(0, 255, 0), thickness=2))
    return image
```

Hàm draw_landmarks: vẽ các điểm mốc trên bàn tay trong ảnh đầu vào sử dụng các kết quả đã được xử lý. Nếu phát hiện bàn tay, code sẽ lặp qua từng bàn tay được nhận diện và vẽ các điểm mốc lên ảnh bằng cách sử dụng các thông số màu sắc và độ dày đã được định trước, sau đó trả về ảnh đã được vẽ.

```python
def get_hand_landmarks(self):
    """
    Retrieve the landmarks of the detected hands from the last processed image.

    Returns:
    - landmarks_list: list of numpy arrays, each array contains landmarks for a hand
                      Each landmark is represented by its normalized coordinates (x, y, z)
    """
    landmarks_list = []

    if self.results.multi_hand_landmarks:
        for hand_landmarks in self.results.multi_hand_landmarks:
            landmarks = []
            for landmark in hand_landmarks.landmark:
                # Store normalized coordinates of each landmark
                landmarks.append((landmark.x, landmark.y, landmark.z))
            landmarks_list.append(np.array(landmarks))

    return landmarks_list
```

Hàm get_hand_landmarks: lấy các tọa độ landmark của các bàn tay được phát hiện từ hình ảnh đã được xử lý. Hàm trả về danh sách các mảng numpy, mỗi mảng chứa các landmark của một bàn tay. Các landmark được đại diện bởi các tọa độ chuẩn hóa (x, y, z). Nếu có các landmark của tay trong self.results.multi_hand_landmarks, hàm sẽ lặp qua từng landmark, lưu trữ các tọa độ của từng landmark vào danh sách và thêm vào landmarks_list.

```python
def num_detected_hands(self):
    """
    Get the number of hands detected in the last processed image.

    Returns:
    - num_hands: int, the number of detected hands
    """
    return len(self.results.multi_hand_landmarks) if self.results.multi_hand_landmarks else 0
```

Hàm num_detected_hands: trả về số lượng tay được phát hiện trong hình ảnh đã xử lý cuối cùng. Hàm này kiểm tra thuộc tính ‘self.results.multi_hand_landmarks’ nếu tồn tại, trả về số lượng các đối tượng landmark của tay, ngược lại trả về 0. Điều này giúp xác định và đếm số tay trong hình ảnh một cách hiệu quả.

```python
def get_minmax_xy(self,landmarks):
    """
    Get the minimum and maximum x, y coordinates of the detected hand landmarks.

    Returns:
    - min_x, max_x, min_y, max_y: float, the minimum and maximum x, y coordinates
    """
    width = gl_cam_width
    height = gl_cam_height
    x_coords = landmarks[:, 0]
    y_coords = landmarks[:, 1]
    min_x, max_x = np.min(x_coords)*width, np.max(x_coords)*width
    min_y, max_y = np.min(y_coords)*height, np.max(y_coords)*height
    return [int(min_x), int(min_y), int(max_x), int(max_y)]
```

hàm get_minmax_xy: lấy các giá trị x và y nhỏ nhất và lớn nhất từ các landmark của tay được phát hiện. Hàm trả về các tọa độ x và y nhỏ nhất và lớn nhất dưới dạng số nguyên. Các tọa độ chuẩn hóa x và y từ landmarks được nhân với chiều rộng và chiều cao của camera (gl_cam_width và gl_cam_height) để chuyển đổi thành tọa độ thực tế. Cuối cùng, các giá trị min và max này được trả về dưới dạng danh sách bốn phần tử: [min_x, min_y, max_x, max_y].

```python
def draw_bounding_box(self, image, minmax_xy):
    """
    Draw a bounding box on the image using the min and max x, y values.

    Parameters:
    - image: numpy array, the input image in BGR format        Le Quang That,
    - minmax_xy: list, containing min_x, min_y, max_x, max_y values

    Returns:
    - image: numpy array, the input image with the bounding box drawn
    """
    min_x, min_y, max_x, max_y = minmax_xy
    cv2.rectangle(image, (min_x, min_y), (max_x, max_y), (0, 0, 255), 2)
    return image
```

Hàm draw_bounding_box: dùng để vẽ một hộp giới hạn trên hình ảnh bằng cách sử dụng các giá trị tọa độ min và max. Hàm nhận vào hình ảnh và danh sách tọa độ, trả về hình ảnh với hộp giới hạn được vẽ.

```python
class Camera:
    def __init__(self, cam_index, cam_width, cam_height, cam_fps):
        self.cam_index = cam_index
        self.cam_width = cam_width
        self.cam_height = cam_height
        self.cam_fps = cam_fps
        self.cap = None

    def initialize_camera(self):
        self.cap = cv2.VideoCapture(self.cam_index)
        if not self.cap.isOpened():
            print(f"Error: Could not open camera {self.cam_index}.")
            return False

        self.cap.set(cv2.CAP_PROP_FRAME_WIDTH, self.cam_width)
        self.cap.set(cv2.CAP_PROP_FRAME_HEIGHT, self.cam_height)
        self.cap.set(cv2.CAP_PROP_FPS, self.cam_fps)

        return True

    def close_camera(self):
        if self.cap is not None:
            self.cap.release()

    def test_camera(self):
        if self.cap is None or not self.cap.isOpened():
            print("Error: Camera is not initialized or could not open.")
            return

        while True:
            ret, frame = self.cap.read()
            if not ret:
                print("Error: Cannot receive frame.")
                break

            cv2.imshow('frame', frame)
            if cv2.waitKey(1) & 0xFF == ord('q'):
                break

        cv2.destroyAllWindows()

    def get_frame(self):
        if self.cap is None or not self.cap.isOpened():
            print("Error: Camera is not initialized or could not open.")
            return None

        ret, frame = self.cap.read()
        if not ret:
            print("Ignoring empty camera frame.")
            return None

        return frame
```

Class `Camera`: được quản lý thiết bị quay video, bao gồm khởi tạo, truy xuất khung và dọn dẹp. Nó định cấu hình các thuộc tính của camera, kiểm tra chức năng của camera và cung cấp các phương thức để bắt đầu và dừng camera.

```python
def calculate(self, landmarks_list):
    """
    Calculate distances and angles between landmarks for each hand.

    Parameters:
    - landmarks_list: list of numpy arrays, each array contains landmarks for a hand

    Returns:
    - features_list: list of numpy arrays, each array contains distances and angles between landmarks for a hand
    """
    thumb_flag = 0
    if landmarks_list[0][4][1] < landmarks_list[0][0][1]:
        thumb_flag = 1

    features_list = np.concatenate((self.calculate_distances(landmarks_list), self.calculate_angles(landmarks_list)), axis=1)
    features_list = features_list.flatten()
    features_list = np.append(features_list, thumb_flag)
    features_list = np.round(features_list, gl_round_decimal)
    return features_list
```

Hàm calculate: dùng để tính toán khoảng cách và góc giữa các điểm landmarks trong mỗi bàn tay, nối các đặc điểm này, làm phẳng mảng kết quả, thêm cờ biểu thị vị trí ngón tay cái và làm tròn các đặc điểm đến độ chính xác thập phân được chỉ định.

```python
def calculate_distances(self, landmarks_list):
    """
    Calculate distances between landmarks in each hand and normalize within a specified range.

    Parameters:
    - landmarks_list: list of numpy arrays, each array contains landmarks for a hand

    Returns:
    - normalized_distances: list of numpy arrays, each array contains normalized distances between landmarks for a hand
    """
    distances_list = []
    for landmarks in landmarks_list:
        distances = []
        # Calculate distances between consecutive landmarks
        for i in range(len(landmarks)):
            for j in range(i + 1, len(landmarks)):
                x1, y1 = landmarks[i][0:2]  # Take only x, y coordinates
                x2, y2 = landmarks[j][0:2]  # Take only x, y coordinates
                distance = np.sqrt((x2 - x1)**2 + (y2 - y1)**2)
                distances.append(distance)
        distances_list.append(np.array(distances))

    # Normalize distances within the range [0, 1]
    normalized_distances = []
    max_distance = max([np.max(distances) for distances in distances_list])
    min_distance = min([np.min(distances) for distances in distances_list])

    for distances in distances_list:
        normalized_distances.append((distances - min_distance) / (max_distance - min_distance))

    return normalized_distances
```

Hàm calculate_distances: dùng để tính toán và chuẩn hóa khoảng cách giữa tất cả các cặp landmarks trong mỗi bàn tay, trả về danh sách khoảng cách chuẩn hóa cho mỗi bàn tay.

```python
def calculate_angles(self, landmarks_list):
    """
    Calculate angles between consecutive landmarks for each hand.

    Parameters:
    - landmarks_list: list of numpy arrays, each array contains landmarks for a hand

    Returns:
    - angles_list: list of numpy arrays, each array contains angles between landmarks for a hand
    """
    angles_list = []

    # Calculate angles between landmarks for each hand
    for landmarks in landmarks_list:
        angles = []

        # Angles between fingertips
        angles.append(self.calculate_angle(landmarks[4], landmarks[0], landmarks[8]))
        angles.append(self.calculate_angle(landmarks[8], landmarks[0], landmarks[12]))
        angles.append(self.calculate_angle(landmarks[12], landmarks[0], landmarks[16]))
        angles.append(self.calculate_angle(landmarks[16], landmarks[0], landmarks[20]))

        # Angles between knuckles
        angles.append(self.calculate_angle(landmarks[1], landmarks[2], landmarks[3]))
        angles.append(self.calculate_angle(landmarks[5], landmarks[6], landmarks[7]))
        angles.append(self.calculate_angle(landmarks[9], landmarks[10], landmarks[11]))
        angles.append(self.calculate_angle(landmarks[13], landmarks[14], landmarks[15]))
        angles.append(self.calculate_angle(landmarks[17], landmarks[18], landmarks[19]))

        # Angles between fingers and wrist
        angles.append(self.calculate_angle(landmarks[0], landmarks[1], landmarks[2]))
        angles.append(self.calculate_angle(landmarks[0], landmarks[5], landmarks[6]))
        angles.append(self.calculate_angle(landmarks[0], landmarks[9], landmarks[10]))
        angles.append(self.calculate_angle(landmarks[0], landmarks[13], landmarks[14]))
        angles.append(self.calculate_angle(landmarks[0], landmarks[17], landmarks[18]))

        # Add angles to the numpy array
        angles_list.append(np.array(angles))

        # Normalize angles within the range [0, 1]
        normalized_angles = []
        max_angle = max([np.max(angles) for angles in angles_list])
        min_angle = min([np.min(angles) for angles in angles_list])

        for angles in angles_list:
            normalized_angles.append((angles - min_angle) / (max_angle - min_angle))

    return normalized_angles
```

Hàm calculate_angles: dùng để tính toán góc giữa các ngón tay, khớp ngón tay, và giữa các ngón tay với cổ tay đã được định nghĩa trước đó. Kết quả được chuẩn hóa về khoảng [0, 1] để thuận tiện cho các ứng dụng tiếp theo như nhận diện cử chỉ. Hàm này rất hữu ích trong các ứng dụng xử lý hình ảnh và thị giác máy tính, đặc biệt là trong nhận diện cử chỉ tay.

```python
def calculate_angle(self, point1, point2, point3):
    """
    Calculate angle between three points.

    Parameters:
    - point1: tuple, (x, y) coordinates of the first point
    - point2: tuple, (x, y) coordinates of the second point
    - point3: tuple, (x, y) coordinates of the third point

    Returns:
    - angle: float, the angle between the three points in degrees
    """
    v1 = np.array([point1[0] - point2[0], point1[1] - point2[1]])
    v2 = np.array([point3[0] - point2[0], point3[1] - point2[1]])
    dot_product = np.dot(v1, v2)
    v1_magnitude = np.linalg.norm(v1)       Le Quang That, 4 hours ago •
    v2_magnitude = np.linalg.norm(v2)
    angle_rad = np.arccos(dot_product / (v1_magnitude * v2_magnitude))
    angle_deg = np.degrees(angle_rad)

    return angle_deg
```

Hàm calculate_angle: dùng để tính toán góc giữa ba điểm trong mặt phẳng 2D (point1, point2, point3) dưới dạng tuple. Đầu tiên, hàm này tính toán hai vector từ điểm trung tâm (point2) đến hai điểm còn lại. Sau đó, nó sử dụng tích vô hướng (dot product) và độ dài của hai vector (norm) để tính toán góc giữa chúng. Kết quả trả về là giá trị góc giữa ba điểm, được biểu diễn bằng độ. Hàm này hữu ích trong các ứng dụng hình học và xử lý hình ảnh.

B.2.3 Save Data

```python
class DataSaver:
    def __init__(self, csv_file):
        self.csv_file = csv_file

        # Create CSV file if it does not exist on current directory
        os.chdir(os.path.dirname(os.path.abspath(__file__)))

        if not os.path.exists(self.csv_file):
            try:
                with open(self.csv_file, mode='w', newline='') as file:
                    writer = csv.writer(file)
                    header = ['person','gesture']
                    for i in range(1, 211):
                        header.append(f'f{i}')
                    for i in range(1, 16):
                        header.append(f'g{i}')
                    writer.writerow(header)

            except IOError as e:
                print(f"Error creating CSV file: {str(e)}")
                raise

        # Notify if creation was successful
        print(f"CSV file '{self.csv_file}' is ready.")

    def save_data(self, gesture, capture_person, features_list):
        label = gesture
        features = features_list
        data = [capture_person, label]
        data.extend(features)

        with open(self.csv_file, mode='a', newline='') as file:
            writer = csv.writer(file)
            writer.writerow(data)
```

Class `DataSaver`: được sử dụng để quản lý việc tạo và thêm dữ liệu vào tệp CSV. Phương thức `__init__` kiểm tra và tạo tệp CSV nếu nó không tồn tại và thiết lập tiêu đề cột. Phương thức `save_data` sẽ thêm một hàng dữ liệu mới vào tệp CSV, bao gồm thông tin của người biểu diễn, nhãn cử chỉ và danh sách các tính năng.

B.2.4 RandomForestTrainer (Model)

```
class RandomForestTrainer:
    def __init__(self, csv_file):
        self.csv_file = csv_file
        self.data = pd.read_csv(csv_file,header=0)
        self.model = None
        self.all_columns = list(self.data.columns)
```

Hàm __init__ của Class RandomForestTrainer để lấy đường dẫn của tệp CSV làm đầu vào, đọc dữ liệu vào DataFrame của gấu trúc, khởi tạo mô hình thành Không có và lưu tên cột của DataFrame.

```
def preprocess_data(self,corr_threshold=0.3):
    print(self.data.info())
    if self.data.isnull().sum().sum() > 0:
        self.data = self.data.dropna()
    print(self.data['gesture'].nunique())
    print(self.data['person'].nunique())
    self.data = self.data.drop('person', axis=1)
```

Hàm preprocess_data : Phương pháp này in thông tin về dữ liệu, kiểm tra các giá trị bị thiếu và loại bỏ các hàng có giá trị bị thiếu. Nó cũng in số lượng giá trị duy nhất trong cột 'cử chỉ' và 'người' và bỏ cột 'người'. Ngoài ra còn có mã nhận xét sẽ loại bỏ các cột có mối tương quan dưới ngưỡng với 'cử chỉ' của cột mục tiêu.

```
def split_data(self, test_size=0.2, random_state=42):
    X = self.data.drop('gesture', axis=1)
    y = self.data['gesture']
    self.X_train, self.X_test, self.y_train, self.y_test = train_test_split(X, y, test_size=test_size, random_state=random_state)
    print('Shape of X_train:', np.array(self.X_train).shape)
    print('Shape of X_test:', np.array(self.X_test).shape)
    print('Shape of y_train:', self.y_train.shape)
    print('Shape of y_test:', self.y_test.shape)
```

Hàm split_data: Phương pháp này chia dữ liệu thành các tập huấn luyện và kiểm tra. Các đặc điểm (X) là tất cả các cột ngoại trừ 'cử chỉ' và mục tiêu (y) là cột 'cử chỉ'. Việc phân chia được thực hiện bằng cách sử dụng kích thước thử nghiệm được chỉ định và trạng thái ngẫu nhiên để tái tạo. Nó cũng in hình dạng của các tập dữ liệu kết quả.

```
def create_model(self,ip_n_estimators,ip_max_depth,ip_min_samples_split,ip_min_samples_leaf,ip_max_features,ip_bootstrap,ip_random_state):
    self.model = RandomForestClassifier(
        n_estimators=ip_n_estimators,
        max_depth=ip_max_depth,
        min_samples_split=ip_min_samples_split,
        min_samples_leaf=ip_min_samples_leaf,
        max_features=ip_max_features,
        bootstrap=ip_bootstrap,
        random_state=ip_random_state
    )

def train_model(self,save_location='m3_rf_model.joblib'):
    self.model.fit(self.X_train, self.y_train)
    model_file = save_location
    dump(self.model, model_file)
```

Hàm create_model: Phương thức này khởi tạo mô hình Random Forest với các tham số được chỉ định, cho phép tùy chỉnh số lượng Tree ước lượng, độ sâu tối đa, số lượng mẫu tối thiểu cho mỗi lần tách, số lượng mẫu tối thiểu cho mỗi nút lá, số lượng đặc trưng tối đa, việc lấy mẫu bootstrap, và trạng thái ngẫu nhiên.

Hàm train_model: Phương thức này huấn luyện mô hình Random Forest bằng cách sử dụng dữ liệu huấn luyện và lưu mô hình đã huấn luyện vào một vị trí được chỉ định.

```python
def evaluate_model(self):
    y_pred = self.model.predict(self.X_test)
    print(self.X_test.info())
    print(classification_report(self.y_test, y_pred))
    print('Accuracy:', self.model.score(self.X_test, self.y_test))
```

Hàm evaluate_model: Phương thức này đánh giá mô hình bằng cách dự đoán các giá trị mục tiêu cho tập kiểm tra. Nó in thông tin về tập kiểm tra, báo cáo phân loại, và độ chính xác của mô hình trên tập kiểm tra.

```python
def plot_feature_importances(self):
    feature_importances = self.model.feature_importances_
    sorted_idx = np.argsort(feature_importances)
    plt.figure(figsize=(10, 6))
    plt.barh(range(self.X_train.shape[1]), feature_importances[sorted_idx], align='center')
    plt.yticks(range(self.X_train.shape[1]), [self.X_train.columns[i] for i in sorted_idx])
    plt.xlabel('Feature Importance')
    plt.title('Feature Importances')
    plt.show()
```

Hàm plot_feature_importances: Phương thức này sẽ vẽ biểu đồ Feature Importance của các đặc trưng của mô hình Random Forest đã được huấn luyện. Nó sắp xếp Feature Importance của các đặc trưng, tạo một biểu đồ thanh ngang và gán nhãn cho các thanh với tên của các đặc trưng tương ứng.

B.2.5 Train Model

```python
from m3_module import *

if __name__ == "__main__":
    trainer = RandomForestTrainer(csv_file='m3_info.csv')
    trainer.preprocess_data(0.3) # Corr < 0.3 will be removed
    trainer.split_data(test_size=0.2, random_state=42)
    trainer.create_model(
        ip_n_estimators=200,
        ip_max_depth=30,
        ip_min_samples_split=5,
        ip_min_samples_leaf=2,
        ip_max_features='sqrt',
        ip_bootstrap=True,
        ip_random_state=
    )
    trainer.train_model(save_location='m3_model.joblib')
    trainer.evaluate_model()
```

(parameter) save_location: str

Le Quang That, 4 hours ago • Re

1. **Tạo đối tượng RandomForestTrainer**:
   - Tạo một đối tượng của RandomForestTrainer với dữ liệu từ tệp CSV 'm3_info.csv'.
2. **Dữ liệu tiền xử lý**:
   - Gọi hàm preprocess_data với ngưỡng tương quan là 0,3 để xử lý dữ liệu.
3. **Tách dữ liệu**:
   - Chia dữ liệu thành tập huấn luyện và tập kiểm tra với tỷ lệ kiểm tra là 20% và random_state là 42 để đảm bảo tính tái lập
4. **Tạo mô hình**:
   - Tạo một mô hình Random Forest với các tham số cụ thể:

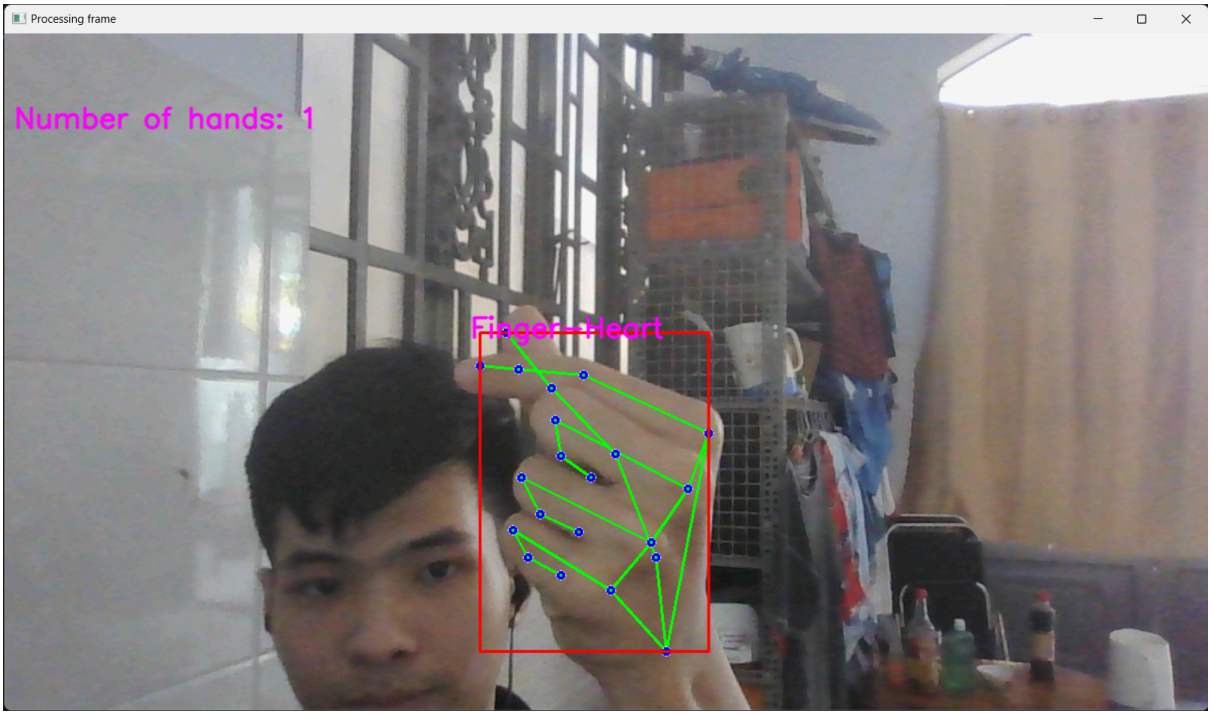
        - n_estimators=200: Số lượng tree trong Forest.
        - max_depth=30: Độ sâu tối đa của Tree.
        - min_samples_split=5: Số lượng mẫu tối thiểu cần thiết để tách một nút nội bộ.
        - min_samples_leaf=2: Số lượng mẫu tối thiểu cần thiết để có tại một nút lá.
        - max_features='sqrt': Số lượng đặc trưng cần xem xét khi tìm kiếm tách tốt nhất.
        - bootstrap=True: Sử dụng mẫu bootstrap khi xây dựng cây.
        - random_state=42: Đảm bảo tính tái lập.
5. **Huấn luyện mô hình**:
    - Huấn luyện mô hình với tập huấn luyện và lưu mô hình đã huấn luyện vào tệp 'm3_model.joblib'.
6. **Đánh giá mô hình**:
    - Đánh giá mô hình bằng cách sử dụng tập kiểm tra, in báo cáo phân loại và độ chính xác của mô hình.

C. Application

Công nghệ nhận dạng cử chỉ tay đã có những bước tiên tiến và tìm thấy ứng dụng trong nhiều lĩnh vực khác nhau, nâng cao khả năng tương tác, truy cập và chức năng. Trong tương tác người-máy tính, nó cho phép giao diện không chạm trong các hệ thống mà vẫn nhận diện được, những chiếc xe ô tô và thiết bị nhà thông minh có những ứng dụng về cử chỉ tay giúp cho các thao tác dễ dàng hơn. Các ứng dụng thực tế ảo và thực tế tăng cường hưởng lợi từ sự tương tác trực quan và chân thực, cho phép người dùng thao tác với các vật thể và điều hướng giao diện mà không cần bộ điều khiển vật lý.

Trong lĩnh vực truy cập, nhận dạng cử chỉ tay đóng vai trò quan trọng trong công nghệ hỗ trợ. Nó tạo ra điều kiện giao tiếp cho những người khiếm thính bằng cách dịch ngôn ngữ ký hiệu sang văn bản hoặc giọng nói. Ngoài ra, nó cung cấp phương tiện cho những người bị hạn chế khả năng vận động tương tác với các thiết bị kỹ thuật số. Công nghệ này cũng được ứng dụng trong trò chơi và giải trí, nâng cao trải nghiệm với thiết bị điều khiển bằng chuyển động và phương tiện tương tác, tạo ra các môi trường động và hấp dẫn.

D.Overall



Tổng quan lại, để xử lý được khung hình trong video nhằm nhận dạng các cử chỉ tay bằng mô hình được huấn luyện trước, xây dựng tốt và thể hiện sự hiểu biết sâu sắc về các thành phần cần thiết cho nhiệm vụ này. Thiết kế mô-đun, tận dụng MediaPipe để phát hiện tay, làm cho đoạn code trở nên hiệu quả. Việc sử dụng các mô hình và cấu hình từ các tệp bên ngoài để tăng thêm tính linh hoạt và khả năng thích ứng của hệ thống. Xử lý lỗi và tương tác người dùng hiệu quả, đảm bảo hoạt động trơn tru và dễ sử dụng.

Tuy nhiên, đoạn code có thể được cải thiện bằng cách thêm nhiều nhận xét và dữ liệu chi tiết hơn để giúp học và hiểu dễ dàng hơn. Xử lý các bug đặc biệt là đối với các tệp và quản lý tài nguyên, sẽ tăng độ tin cậy của đoạn code. Tối ưu hóa hiệu suất, đặc biệt là trong việc xử lý video độ phân giải cao, và trực quan hóa các đặc điểm và dự đoán sẽ có lợi cho việc gỡ lỗi và phát triển. Mở rộng chức năng để xử lý nhiều video sẽ tăng thêm tính linh hoạt cho ứng dụng.