



ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH  
TRƯỜNG ĐẠI HỌC BÁCH KHOA

Khoa Điện – Điện tử

Bộ môn - Viễn Thông

# Hệ thống phát hiện và cảnh báo té ngã thời gian thực tích hợp cảm biến, xử lý ảnh và định vị

*Real-time fall detection and alert system integrating sensors,  
image processing, and positioning*

SV:

Trần Đức Hảo

GVHD:

PSG.TS Hà Hoàng Kha

Ngày 23 tháng 9 năm 2025

## Lời cảm ơn

Trước hết, tôi xin bày tỏ lòng biết ơn sâu sắc đến **PGS.TS. Hà Hoàng Kha**, người đã tận tâm hướng dẫn và đồng hành cùng tôi trong suốt quá trình thực hiện đồ án. Thầy đã chỉ bảo từng bước, nhắc nhở về thái độ học tập và khuyến khích tôi tiếp cận vấn đề một cách thực chất, từ đó tích lũy kiến thức và kỹ năng nghiên cứu chuyên nghiệp hơn.

Tôi cũng xin cảm ơn các thầy cô trong **Bộ môn Viễn Thông, Khoa Điện – Điện tử, Trường ĐH Bách khoa – ĐHQG TP.HCM** đã trang bị nền tảng kiến thức vững chắc và tạo điều kiện thuận lợi để tôi phát triển chuyên môn toàn diện.

Đặc biệt, tôi xin gửi lời tri ân đến **gia đình** – điểm tựa tinh thần vững chắc, nguồn động lực lớn để tôi kiên trì và hoàn thành luận văn.

Cuối cùng, tôi xin cảm ơn tất cả những người đã hỗ trợ tôi trong quá trình học tập và nghiên cứu.

# Mục lục

- 1 I-GIỚI THIỆU
- 2 II-CƠ SỞ LÝ THUYẾT
- 3 III-THIẾT KẾ VÀ TRIỂN KHAI
- 4 IV-KẾT QUẢ THỰC NGHIỆM
- 5 KẾT LUẬN CHUNG

# Mục lục chương: I-GIỚI THIỆU

## 1 I-GIỚI THIỆU

- Tình hình nghiên cứu
- Giới hạn thực hiện đề tài hệ thống cảnh báo té ngã

## 2 II-CƠ SỞ LÝ THUYẾT

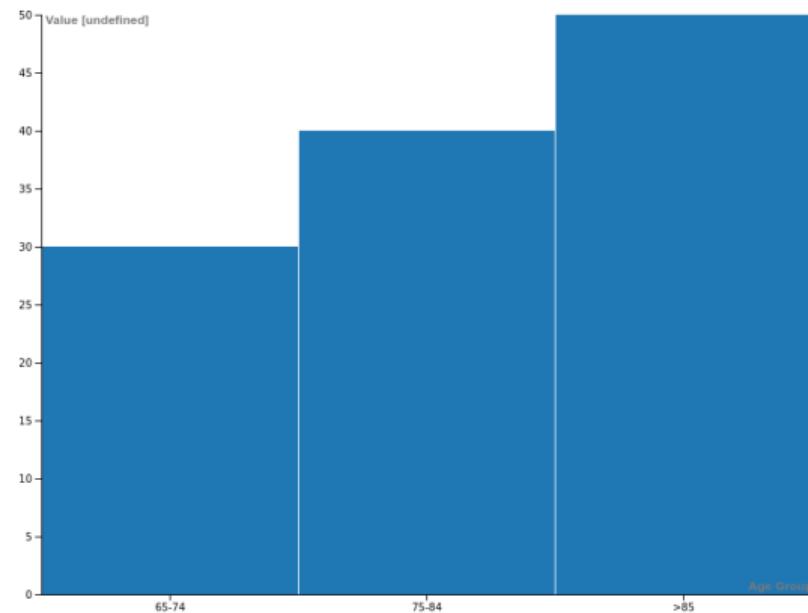
## 3 III-THIẾT KẾ VÀ TRIỂN KHAI

## 4 IV-KẾT QUẢ THỰC NGHIỆM

## 5 KẾT LUẬN CHUNG

# Té ngã: Vấn đề chung toàn cầu

- Nguyên nhân chính gây chấn thương và tử vong không cố ý.
- WHO:** ~ 646,000 ca tử vong/năm; > 80% ở các nước thu nhập trung bình/thấp.
- Người cao tuổi: **30%** té ngã/năm ở người > 65 tuổi, tăng lên **50%** ở người > 85 tuổi.



Hình: Tỷ lệ té ngã theo nhóm tuổi

# Các phương pháp phát hiện té ngã

- **Dựa trên thị giác (Vision-based):** Sử dụng camera và thuật toán nhận diện tư thế người (MediaPipe, OpenPose).
- **Dựa trên cảm biến đeo (Wearable-based):** Dùng cảm biến quán tính (IMU, MPU6050) trên thiết bị.
- **Kết hợp đa phương thức (Multi-modal):** Tích hợp dữ liệu từ nhiều nguồn để tăng độ chính xác.

# Nghiên cứu trong và ngoài nước

## Quốc tế

- **Xu hướng:** Sử dụng YOLO, Transformer, AI nhẹ, cảm biến mmWave.
- **Thành tựu:** Giảm false alarm, tối ưu cho thiết bị biên, Sensor Fusion.

## Trong nước

- **Thực trạng:** Chủ yếu mô hình thử nghiệm (PoC) với ESP32, Arduino.
- **Hạn chế:** Thiếu dữ liệu lớn, độ chính xác thấp (75-85%), thiếu tích hợp đa phương thức.

# Hiệu năng và Giới hạn

## Mục tiêu Hiệu năng

- Tổng độ trễ < 5 giây.
- Độ chính xác > 90%, False Alarm < 8%.
- Uptime dịch vụ MQTT > 99%.

## Giới hạn nghiên cứu

- Hoạt động trong nhà với điều kiện ánh sáng và mạng ổn định.
- Nguyên mẫu **ESP32** chưa tích hợp học sâu toàn phần.
- Không phát triển app di động/web phức tạp.

# Mục lục chương: II-CƠ SỞ LÝ THUYẾT

1 I-GIỚI THIỆU

2 II-CƠ SỞ LÝ THUYẾT

- Tổng quan lý thuyết
- Các phương thức truyền thông sử dụng
- Cơ sở lý thuyết về thị giác máy tính (CV)
- Nhận diện tư thế người
- Cơ sở lý thuyết xây dựng phần cứng

3 III-THIẾT KẾ VÀ TRIỂN KHAI

4 IV-KẾT QUẢ THỰC NGHIỆM

# Ưu nhược điểm các phương pháp phát hiện té ngã

Phương pháp	Cơ chế	Ưu điểm	Nhược điểm
Đeo được	IMU (gia tốc kế, con quay hồi chuyển); phát hiện gia tốc/tư thế bất thường	Phản hồi nhanh; chính xác; chi phí thấp	Cần đeo liên tục; dễ false positive; pin/hiệu chuẩn
Môi trường	Cảm biến cố định: sàn áp suất, PIR, âm thanh; AI phân tích	Không xâm phạm; giám sát nhiều người; tích hợp smart home	Chi phí cao; phạm vi hạn chế; nhầm vật thể
Thị giác	Camera RGB/RGB-D/IR; pose estimation (OpenPose/MediaPipe)	Thông tin trực quan; không cần đeo; tích hợp giám sát	Quyền riêng tư; phụ thuộc ánh sáng; cần phần cứng mạnh
Đa phương thức	Kết hợp IMU + camera + môi trường; data fusion (Kalman/Deep Learning)	Độ chính xác cao; giảm cảnh báo sai; mở rộng phạm vi; kinh tế	Phức tạp; tốn năng lượng; đồng bộ khó

- Kết hợp dữ liệu để xác nhận té ngã, giảm false positive.
- Chế độ linh hoạt: In-situ (cục bộ) + Mobile (edge device).
- Bảo mật: xử lý cục bộ, chỉ gửi dữ liệu tối thiểu, tùy chỉnh khu vực nhạy cảm.

# Các Giao Thức Truyền Thông trong Hệ Thống Cảnh Báo IoT

Hệ thống phát hiện ngã với ba giao thức chính.

- **SIP:** Truyền tải âm thanh/video cảnh báo thời gian thực
- **MQTT:** Vận chuyển dữ liệu cảm biến từ thiết bị IoT
- **JSON:** Định dạng cấu trúc dữ liệu trao đổi

## Mục tiêu

Xây dựng hệ thống cảnh báo không gián đoạn, độ trễ thấp từ cảm biến đến cuộc gọi VoIP

# Giao thức SIP - Khởi tạo Phiên

## Chức năng chính:

- Thiết lập cuộc gọi VoIP từ hệ thống cảnh báo
- Kết nối với Asterisk PBX để gọi điện thoại
- Truyền âm thanh cảnh báo qua RTP(real time protocol)

## Các bước hoạt động:

- ① REGISTER: Đăng ký thiết bị với server
- ② INVITE: Khởi tạo cuộc gọi cảnh báo
- ③ ACK: Xác nhận kết nối thành công
- ④ RTP: Truyền dữ liệu âm thanh
- ⑤ BYE: Kết thúc cuộc gọi

# Phân biệt Đường tín hiệu và Đường truyền phương tiện trong SIP

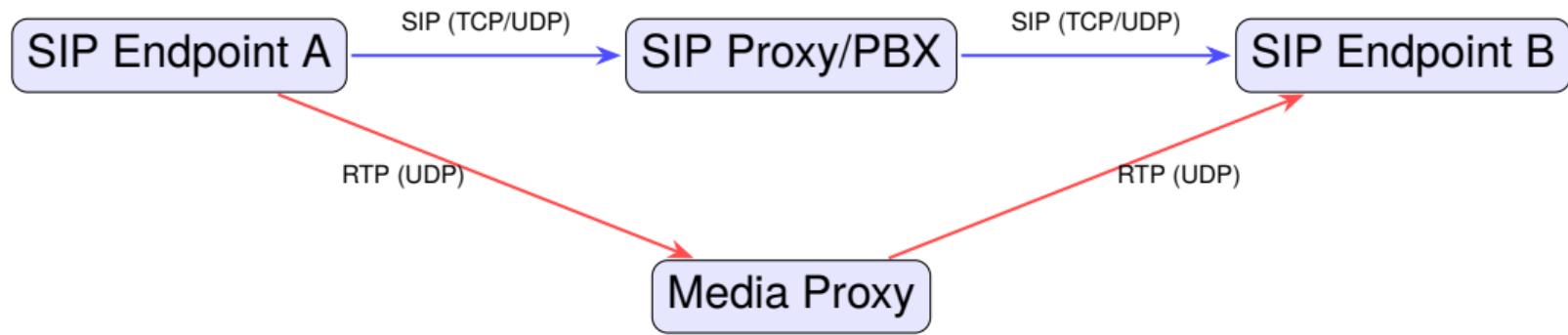
## Đường tín hiệu (Signaling Path)

- Mang các tin nhắn SIP (INVITE, BYE, 200 OK, v.v.)
- Thiết lập, quản lý và kết thúc cuộc gọi
- Sử dụng TCP hoặc UDP

## Đường truyền phương tiện (Media Path)

- Mang dữ liệu thoại/video thực tế
- Sử dụng RTP qua UDP
- Truyền trực tiếp giữa các điểm cuối

# Sơ đồ Đường tín hiệu và Đường truyền phương tiện



# Giao thức ICE (Interactive Connectivity Establishment)

## Vấn đề

Các thiết bị thường nằm sau NAT/firewall, ngăn cản truyền dữ liệu RTP trực tiếp

## Giải pháp ICE

- **Local IP:** Địa chỉ IP nội bộ của thiết bị
- **STUN:** Phát hiện địa chỉ IP công cộng và cổng NAT
- **TURN:** Máy chủ chuyển tiếp khi STUN thất bại

# SIP trong phần mềm mã nguồn mở Asterisk

## Lợi ích

- **Quản lý tập trung:** Đồng nhất cấu hình và quản lý thiết bị
- **Tương thích cao:** Hỗ trợ đa dạng nền tảng và thiết bị
- **Chuẩn mở:** Tích hợp dễ dàng với hạ tầng hiện có
- **Bảo mật:** Hỗ trợ TLS (SIP) và SRTP (RTP)

## Vai trò của Asterisk

Đóng vai trò như SIP server, xử lý đăng ký và định tuyến cuộc gọi

# Giao thức MQTT - Tổng quan

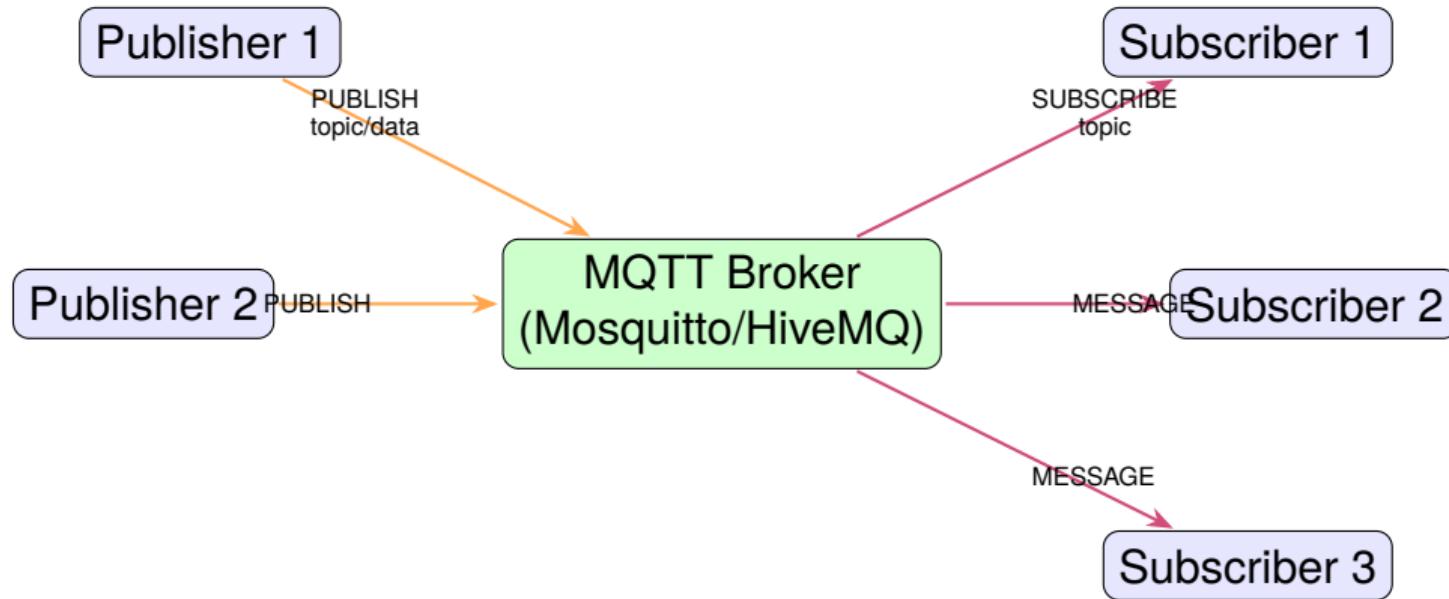
## Định nghĩa MQTT

- MQTT = Message Queuing Telemetry Transport
- Giao thức nhẹ, tối ưu cho IoT và M2M
- Hoạt động trên TCP/IP với cơ chế kết nối lâu dài

## Đặc điểm MQTT

- Thiết kế cho thiết bị có tài nguyên hạn chế
- Phù hợp với băng thông thấp, kết nối không ổn định
- Tiêu chuẩn OASIS cho IoT messaging

# Kiến trúc Publish/Subscribe của MQTT



# Lợi ích và Quality of Service (QoS)

## Lợi ích của mô hình

### Publish/Subscribe

- Không gian: Publisher và Subscriber không cần biết địa chỉ IP của nhau, giao tiếp qua broker
- Thời gian: Không yêu cầu kết nối đồng thời, hỗ trợ retained messages và clean session
- Đồng bộ: Truyền/nhận độc lập, giảm độ trễ và tăng hiệu suất

## Quality of Service (QoS)

- QoS 0: "Fire and forget", nhanh nhưng có thể mất message – Dữ liệu cảm biến thường xuyên
- QoS 1: Đảm bảo gửi ít nhất một lần, có thể trùng lặp – Cân bằng độ tin cậy và hiệu suất
- QoS 2: Đảm bảo gửi đúng một lần, tin cậy nhất nhưng chậm – Dữ liệu quan trọng

# Bảo mật trong MQTT

## Mã hóa Transport Layer

- Hỗ trợ TLS/SSL cho kết nối bảo mật
- MQTT over TLS (port 8883)
- Bảo vệ dữ liệu trong quá trình truyền

## Xác thực và Ủy quyền

- Username/Password authentication
- Client certificates cho xác thực mạnh
- Access Control Lists (ACL) kiểm soát quyền truy cập topic

# Các lệnh MQTT chính

## Connection Management

- CONNECT/CONNACK: Khởi tạo và xác nhận kết nối
- DISCONNECT: Ngắt kết nối graceful

## Messaging Operations

- PUBLISH: Gửi message tới topic
- PUBACK/PUBREC/PUBREL/PUBCOMP: QoS acknowledgments
- SUBSCRIBE/SUBACK: Đăng ký và xác nhận topic
- UNSUBSCRIBE: Hủy đăng ký topic

## Keep Alive

PINGREQ/PINGRESP: Duy trì kết nối

# MQTT trong Hệ thống IoT và Cảnh báo

## Ứng dụng trong IoT

- Thu thập dữ liệu sensors, điều khiển thiết bị
- Giám sát hệ thống, gửi thông báo

**Ví dụ topic:** sensor/room/temperature, alert/fall/detected

## Lợi ích cho hệ thống cảnh báo

- Kết nối đáng tin cậy, truyền real-time
- Hỗ trợ offline messaging, scale tốt
- Nhẹ, tiết kiệm băng thông
- Hỗ trợ 3 mức QoS đảm bảo độ tin cậy

# JSON - JavaScript Object Notation

## Định nghĩa và Đặc điểm

- Định dạng dữ liệu nhẹ, dễ đọc, độc lập ngôn ngữ
- Cấu trúc cặp key:value, dùng cho trao đổi dữ liệu
- Phổ biến trong IoT cho lưu trữ và truyền

# Cấu trúc JSON cơ bản

## Cấu trúc dữ liệu

- **Object:** {key: value}
- **Array:** [value1, value2]
- **Kiểu giá trị:** String, Number, Boolean, null, Object, Array

## Ví dụ JSON

```
1  {
2      "device_id": "ESP32_001",
3      "temperature": 25.5,
4      "sensors": ["temp", "light"]
5  }
```

# JSON trong Hệ thống IoT

## Ứng dụng:

- Lưu cấu hình thiết bị (Wi-Fi, MQTT), thông số cảm biến
- Trao đổi dữ liệu: payload cho MQTT, API, gửi cảnh báo

## Một số thư viện JSON cho nhúng:

- json-c: Phân tích cú pháp, tạo JSON cho C/C++
- FirebaseJson: Dễ dùng, hỗ trợ JSON phức tạp cho IoT

# Tối Ưu Hóa và kết hợp các phương thức

## Lợi ích kết hợp:

- MQTT: Thu thập dữ liệu hiệu quả
- JSON: Cấu trúc linh hoạt
- SIP: Cảnh báo âm thanh tức thì

## Tối ưu:

- Payload JSON nhỏ gọn
- QoS MQTT phù hợp
- Tự động reconnect
- Bảo mật TLS

# Định nghĩa và Mục tiêu

## Định nghĩa

**Thị giác Máy tính (CV):** Lĩnh vực AI cho phép máy tính xử lý, phân tích và diễn giải hình ảnh/video, mô phỏng thị giác con người.

## Mục tiêu

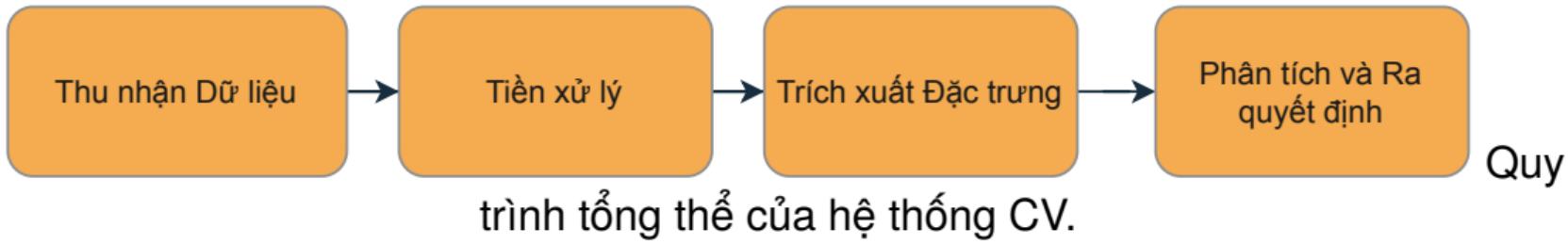
- Tái tạo khả năng nhận thức thị giác với tốc độ, độ chính xác và quy mô vượt trội.
- Ứng dụng trong Hệ thống phát hiện và cảnh báo té ngã thời gian thực tích hợp cảm biến, xử lý ảnh và định vị, đặc biệt là **Ước lượng Tư thế Người (HPE)**.

# Pipeline Cơ bản của Hệ thống CV

## Quy trình

- ① **Thu nhận dữ liệu:** Thu thập ảnh/video từ camera.
- ② **Tiền xử lý:** Chuẩn hóa kích thước, điều chỉnh sáng/tương phản, giảm nhiễu.
- ③ **Trích xuất đặc trưng:** Chuyển pixel thành đặc trưng trừu tượng (cạnh, góc, kết cấu).
- ④ **Phân tích và quyết định:** Phân loại, nhận dạng hoặc ước lượng tư thế.

## Minh họa



# Phân loại Bài toán CV

## Các bài toán cốt lõi

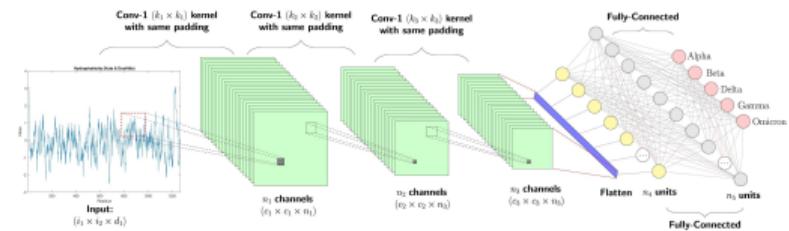
- **Phân loại Ảnh:** Gán nhãn cho toàn bộ ảnh (VD: "Người", "Xe").
- **Phát hiện Đối tượng:** Xác định vị trí và nhãn bằng hộp giới hạn.
- **Phân đoạn Ảnh:**
  - **Ngữ nghĩa:** Gán nhãn từng pixel (VD: Đường, Cây).
  - **Thể hiện:** Phân biệt các cá thể cùng lớp.
- **Ước lượng Tư thế Người (HPE):** Xác định tọa độ **khớp keypoint** để phân tích chuyển động.

# Mô hình Học sâu: CNN

- Mạng Nơ-ron Tích chập (CNN):** Kiến trúc chủ đạo cho xử lý ảnh.
- Phép tích chập:** Trích xuất đặc trưng cục bộ:

$$(I * K)(i, j) = \sum_m \sum_n I(i - m, j - n)K(m, n) \quad (1)$$

- Phép gộp:** Giảm kích thước, tăng tính bền vững (VD: Max Pooling).



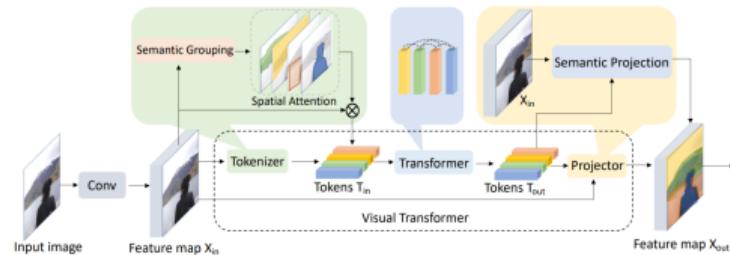
Hình: Phép tích chập và gộp.

# Mô hình Học sâu: Vision Transformer

- **Vision Transformer (ViT):** Chia ảnh thành **miếng vá**, xử lý như token.
- **Tự chú ý (Self-Attention):**

$$\text{Attention}(Q, K, V) = \text{softmax} \left( \frac{QK^T}{\sqrt{d_k}} \right) V \quad (2)$$

- Học quan hệ toàn cục, vượt giới hạn của CNN.



**Hình:** Kiến trúc ViT.

# Tập dữ liệu và Metrics Đánh giá

## Tập dữ liệu

- **ImageNet**: Phân loại ảnh (>14 triệu ảnh).
- **COCO**: Phát hiện, phân đoạn đối tượng.
- **MPII, COCO Keypoints**: Ước lượng tư thế người (HPE).

## Metrics đánh giá

- **IoU**: Đo độ trùng khớp hộp giới hạn.
- **mAP**: Trung bình độ chính xác cho phát hiện đối tượng.
- **F1-score**: Cân bằng Precision và Recall.
- **OKS**: Đo độ chính xác khớp trong HPE.

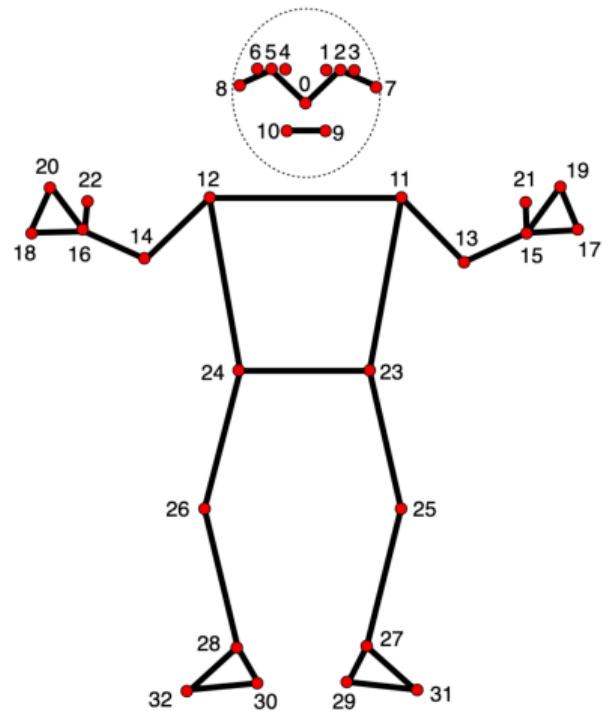
# Nhận diện Tư thế Người và Phát hiện Té ngã

## Tổng quan

Hệ thống tích hợp nhận diện tư thế (MediaPipe Pose) và phát hiện té ngã dựa trên đặc trưng động học/tư thế.

- Ứng dụng: Giám sát an toàn, phát hiện té ngã.
- Nền tảng: Thị giác máy tính thời gian thực.

# Nhận diện Tư thế Người



## Khái niệm

Ước lượng vị trí khớp từ hình ảnh/video:

$$\mathcal{K} = \{k_i = (x_i, y_i, z_i, c_i)\}$$

( $c_i$  là confidence score cho mỗi keypoint).

## Phương pháp

- Top-down:** Phát hiện người trước, sau đó keypoints (MediaPipe).
- Bottom-up:** Keypoints trước, nhóm thành người sau (OpenPose).

# MediaPipe Pose – Kiến trúc BlazePose

## Kiến trúc BlazePose

BlazePose tối ưu HPE 3D với:

- **Nodes:** Các module xử lý tín hiệu hình ảnh.
- **Edges:** Luồng dữ liệu đồng bộ giữa các module.

(*Nodes = các bước tính toán; Edges = kết nối dữ liệu giữa các bước*).

# MediaPipe Pose – Thành phần Hậu xử lý

## Thành phần chính

- **Detection:** ROI từ ảnh RGB, phát hiện người.
- **Landmark:** 33 keypoints 3D, Loss:  $\mathcal{L} = \sum \lambda_i \mathcal{L}_i$ .
- **Tracking:** Dự đoán vị trí ROI cho khung tiếp theo.  
*(Landmark 3D giúp đánh giá tư thế và động tác).*

## Hậu xử lý

- **One Euro Filter:** Làm mịn nhiễu trong dữ liệu keypoints.
- **Chuẩn hóa  $z$ :** Dựa trên hông, tăng độ chính xác 3D.

# Thuật toán Phát hiện Té ngã

## Đặc trưng Động học

- **Vận tốc COM:**  $\vec{v}_{COM} = \frac{\Delta \vec{p}}{\Delta t}$
- **Gia tốc:**  $a = \frac{\|\Delta \vec{v}\|}{\Delta t}$

## Đặc trưng Tư thế

- **AR (Aspect Ratio):** Tăng khi người nằm ngang
- $\theta_{body}$ : Góc vai-hông
- $\Delta h_{head}$ : Giảm chiều cao đầu  
( $AR, \theta, \Delta h$  giúp xác định tư thế bất thường).

## Ba Giai đoạn Phát hiện

- ➊ **Sớm:** Tốc độ/gia tốc COM cao
- ➋ **Xác nhận:** AR,  $\theta_{body}$  chỉ nằm ngang
- ➌ **Bất động:** Chuyển động  $< M_{th}$

# Tổng quan Kiến trúc Hệ thống Phát hiện Té ngã

## Phân loại hệ thống

- **Dựa trên Camera:** Xử lý hình ảnh cố định, yêu cầu máy chủ mạnh
- **Dựa trên Thiết bị đeo:** Cảm biến IMU, ESP32, truyền thông di động

## Ba thành phần cốt lõi

- 1 **Thiết bị Thu thập Dữ liệu:** IMU, Camera, GPS
- 2 **Máy chủ/Xử lý:** Phân tích dữ liệu, Học sâu
- 3 **Truyền thông:** Wi-Fi, 4G/LTE đảm bảo kết nối

# Môi trường Phát triển (ESP-IDF)

## Đặc trưng của ESP-IDF

- **Build system CMake + Kconfig** – cấu hình linh hoạt, dễ mở rộng component
- **FreeRTOS tích hợp sẵn** – quản lý đa nhiệm trên 2 lõi Xtensa
- **Driver cấp thấp** – I2C, SPI, UART, PWM, GPIO được tối ưu cho ESP32
- **Hỗ trợ mạng phong phú** – Wi-Fi, Bluetooth, TCP/IP stack, MQTT, HTTP(S)

## Lợi ích cho hệ thống Phát hiện Té ngã

- Quản lý **đa component** (IMU, SIM4G, LED, Fall Logic) độc lập
- Thực thi **song song**: lõi 1 xử lý cảm biến, lõi 2 lo truyền thông
- Hỗ trợ **OTA update** để nâng cấp firmware từ xa
- Debug chuyên nghiệp: idf.py monitor, gdbstub, logging

# Ví điều khiển ESP32

## Đặc điểm chính

- Lõi kép Xtensa LX6, FreeRTOS
- Wi-Fi + Bluetooth tích hợp
- Hỗ trợ MQTT, HTTP



# Cảm biến IMU và GPS

## IMU

- Gia tốc kế  $a = [a_x, a_y, a_z]$
- Con quay hồi chuyển  $\omega = [\omega_x, \omega_y, \omega_z]$
- Từ kế – xác định hướng
- Fusion: Kalman/Madgwick

## GPS

- Module NEO-6M / EC800K
- Định vị NMEA, tọa độ cứu hộ
- Kết hợp truyền thông SMS/4G

# Xử lý tại Biên và Máy chủ

## Edge (ESP32)

- Xử lý IMU thời gian thực
- Phát hiện té ngã sơ cấp
- Truyền dữ liệu JSON/MQTT

## Cloud/Server

- Xử lý ảnh từ Camera (ESP32-S3 + OV5640)
- TensorFlow/PyTorch, OpenCV

# Hệ thống Truyền thông và Logic Hoạt động

## Wi-Fi (chính)

- Truyền tải dung lượng lớn (ảnh/video)
- MQTT với máy chủ
- Độ trễ thấp

## 4G/LTE (dự phòng)

- SMS/cuộc gọi khẩn
- Định vị GPS
- Hoạt động khi Wi-Fi lỗi

## Logic Hoạt động Hệ thống:

- ① ESP32 thu thập dữ liệu IMU/Camera
- ② Phát hiện té ngã sơ cấp tại biên
- ③ Truyền dữ liệu lên máy chủ (ưu tiên Wi-Fi, dự phòng 4G)
- ④ Máy chủ xử lý tin và phát cảnh báo
- ⑤ Kích hoạt cảnh báo (SMS/cuộc gọi)

## 06<sub>b</sub>ackground<sub>s</sub>ummary

# Mục lục chương: III-THIẾT KẾ VÀ TRIỂN KHAI

1 I-GIỚI THIỆU

2 II-CƠ SỞ LÝ THUYẾT

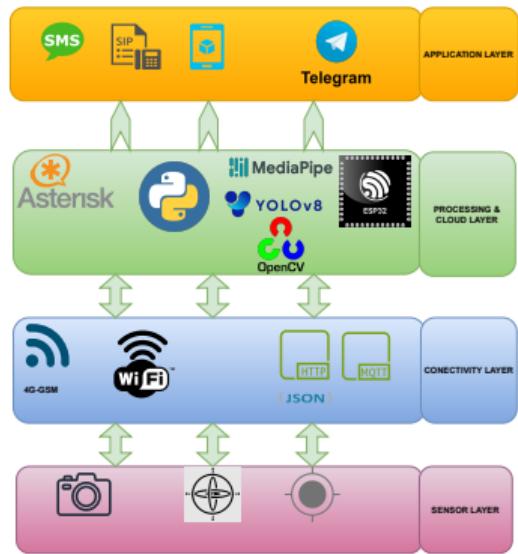
3 III-THIẾT KẾ VÀ TRIỂN KHAI

- Tổng quan kiến trúc
- Thực hiện phần cứng
- Triển khai thực hiện phần mềm

4 IV-KẾT QUẢ THỰC NGHIỆM

5 KẾT LUẬN CHUNG

# Kiến trúc Hệ thống FDAS



## Quy trình

- Minh họa kiến trúc hệ thống phát hiện té ngã và cảnh báo.

## Mô tả

- Lớp Thiết bị/Biên:** Thu thập dữ liệu.
- Lớp Kết nối:** Truyền tải dữ liệu.
- Lớp Xử lý/Đám mây:** Xử lý và ra quyết định.
- Lớp Ứng dụng/UI:** Giao diện người dùng.

# Lớp Xử lý/Dám mây

## Main Processing Server (Python)

- **MQTT Client:** Subscribe dữ liệu từ ESP32
- **AI & Image Processing:** YOLO model xử lý video
- **Business Logic:** Kết hợp 2 luồng dữ liệu → Quyết định cảnh báo

## Asterisk Server

- Tổng đài PBX
- Cuộc gọi khẩn cấp (SIP)

## Telegram Bot API

- Tin nhắn/hình ảnh
- Thông báo nhóm

# Lớp Ứng dụng/Giao diện

## Mobile/Web App

- Dashboard
- Lịch sử sự kiện
- Cấu hình

## SIP Client

- Linphone/Zoiper
- Nhận cuộc gọi báo động

## Telegram

- Thông báo trực tiếp
- Tương tác 2 chiều

**Cảnh báo đa kênh:** Gọi thoại + Nhắn tin

# Thực hiện Phần cứng FDAS

## Kiến trúc mô-đun

Hệ thống gồm 2 module hoạt động độc lập:

- **Module I:** Thiết bị đeo/Cảm biến - Thu thập chuyển động & định vị
- **Module II:** Camera giám sát - Xác nhận sự kiện qua hình ảnh

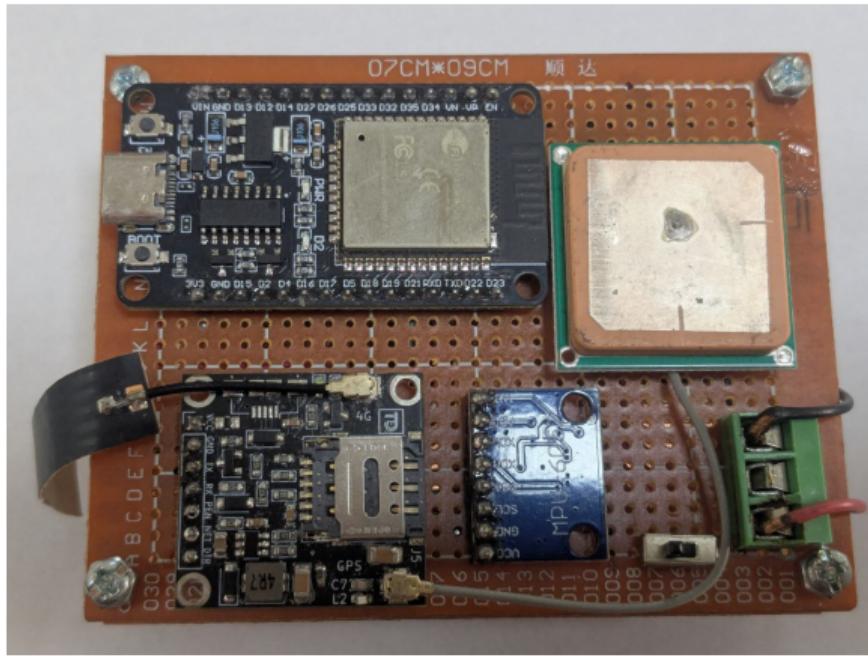
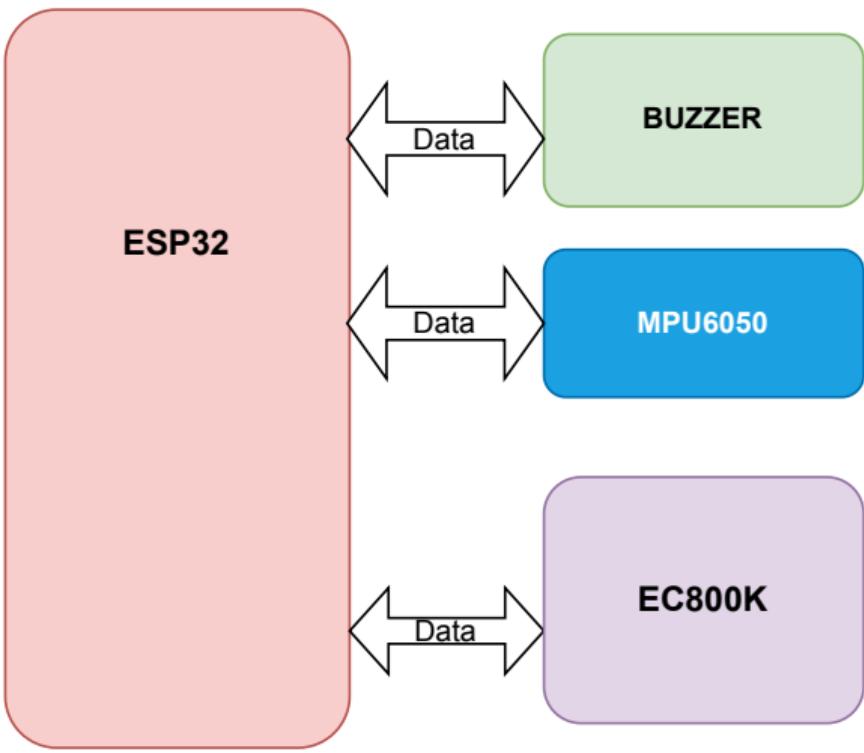
## Ưu điểm

- Linh hoạt triển khai
- Giám sát phạm vi rộng
- Duy trì hoạt động khi 1 module lỗi

## Nguyên lý

2 nguồn dữ liệu độc lập tăng độ tin cậy phát hiện

# Module I: Thiết bị đeo/Cảm biến



Hình: Module I thực tế

# Module II: Camera giám sát

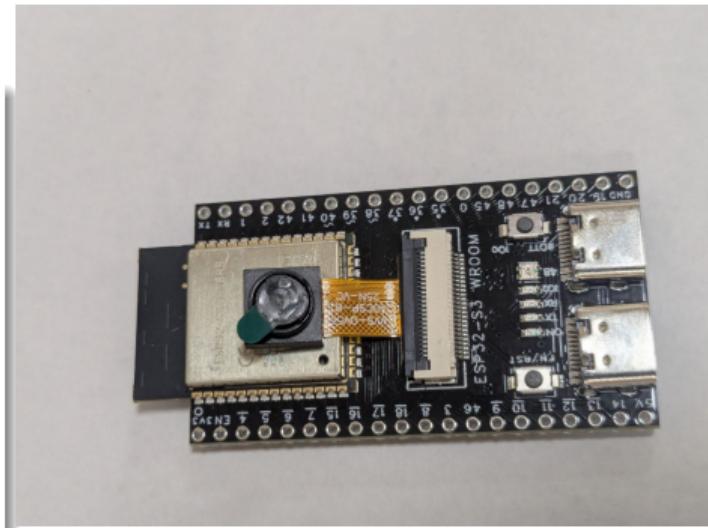
## Thành phần chính

- **ESP32-S3-N16R8:**

- Vi điều khiển mạnh mẽ
- Tích hợp PSRAM
- Giao diện camera chuyên dụng

- **Camera OV5640:**

- Cảm biến 5MP
- Đa kích thước (QQVGA-UXGA)
- Bus 8-bit, 20MHz



Hình: Module II thực tế

# Kết nối ESP32-S3 OV5640

Bảng: Sơ đồ kết nối chân

<b>Chức năng</b>	<b>Chân ESP32-S3</b>	<b>Mô tả</b>
XCLK	15	Xung nhịp camera
SIOD (SDA)	4	Dữ liệu I2C
SIOC (SCL)	5	Xung nhịp I2C
D0-D7	11,9,8,10,12,18,17,16	Bus dữ liệu 8-bit
VSYNC	6	Đồng bộ đọc
HREF	7	Tham chiếu ngang
PCLK	13	Xung nhịp điểm ảnh

## Giao tiếp

SCCB (tương tự I2C) cho cấu hình + Bus song song 8-bit cho dữ liệu

# Chi phí Phần cứng - Module I

Linh kiện	Chức năng	Giá (VNĐ)
ESP32-DevKitC-1	Vi điều khiển chính	110.000 - 125.000
MPU6050	Cảm biến IMU 6 trục	45.000 - 55.000
GPS antenna	Anten nhận GPS	35.000 - 60.000
GPS/4G EC800K	Định vị & gửi cảnh báo	240.000
Buzzer	Cảnh báo âm thanh	5.000 - 10.000
<b>Tổng Module I</b>		<b>435.000 - 490.000</b>

# Chi phí Phần cứng - Module II

Linh kiện	Chức năng	Giá (VNĐ)
ESP32-S3-N16R8	Vi điều khiển xử lý ảnh	275.000 - 300.000
Camera OV5640	Cảm biến hình ảnh 5MP	150.000 - 200.000
<b>Tổng Module II</b>		<b>425.000 - 500.000</b>
<b>TỔNG HỆ THỐNG</b>		<b>860.000 - 990.000</b>

# Tổng quan Triển khai Phần mềm toàn bộ Hệ thống

Thành phần	Nền tảng	Công nghệ chính	Chức năng
Module I	ESP32	ESP-IDF, C/C++	Cảm biến đeo, phát hiện ngã
Module II	ESP32	ESP-IDF, OpenCV	Camera giám sát
Server	Linux	Python, MQTT	Xử lý trung tâm, AI
Asterisk	Linux Mint 21	PJSIP, AMI	Hệ thống VoIP

# Asterisk: Liên lạc và Xử lý Cuộc gọi SIP

Thông tin	Chi tiết	Chức năng
Phiên bản	Asterisk 22.5.1 trên Linux Mint 21	Tổng đài VoIP
pjsip.conf	Quản lý endpoint	Kết nối qua Python (AMI/ARI)
extensions.conf	Dial Plan	Xử lý cuộc gọi SIP
manager.conf	Cấu hình AMI	Python điều khiển cuộc gọi

# Asterisk: Cấu hình PJSIP và Endpoint

```
1 [transport-udp]
2   type=transport
3   protocol=udp
4   bind=0.0.0.0
5
6 [6001]
7   type=endpoint
8   disallow=all
9   allow=ulaw
10  auth=auth6001
11  aors=6001
12  context=internal
13
14 [auth6001]
15   type=auth
16   auth_type=userpass
17   username=6001
18   password=1234
```

## Thành phần chính

### Endpoint [6001]:

- Thiết bị cuối nhận tin báo
- Context: internal
- Codec: ulaw

### Endpoint [server]:

- Điều phối SIP messages
- Kết nối với Python server

## Chức năng

# Asterisk: Cấu hình Dial Plan

```
1 [internal]
2 exten => 6001,1,Answer()
3 same => n,Dial(PJSIP/6001,20)
4 same => n,Hangup()
5
6 exten => 6000,1,Dial(PJSIP/6001&PJSIP/6002&PJSIP/6003,20)
7 same => n,Hangup()
8
9 [messages]
10 exten => _X.,1,NoOp(==> SIP MESSAGE from ${MESSAGE(from)})
11 same => n,MessageSend(pjsip:${EXTEN},pjsip:server)
12 same => n,NoOp(==> Send status: ${MESSAGE_SEND_STATUS})
13 same => n,Hangup()
```

## Context [internal]

- Xử lý cuộc gọi nội bộ
- Extension 6001: thiết bị đơn
- Extension 6000: gọi nhóm

## Context [messages]

- Xử lý tin nhắn SIP
- Chuyển tiếp đến server
- Kích hoạt xử lý Python

# Asterisk: Cấu hình Asterisk Manager Interface (AMI)

```
1 [general]
2 enabled = yes
3 port = 5038
4 bindaddr = 127.0.0.1
5
6 [hx]
7 secret = 123
8 read = all
9 write = all
```

## Bảo mật: Chỉ localhost

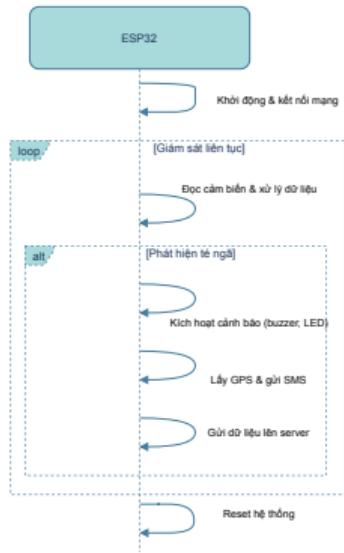
bindaddr = 127.0.0.1

Tham số	Chức năng
port 5038	Cổng kết nối AMI
[hx] account	Tài khoản Python server
read/write all	Quyền điều khiển đầy đủ

## Vai trò AMI

- Giao diện lập trình API
- Python điều khiển Asterisk
- Lấy trạng thái cuộc gọi
- Gửi lệnh quản lý

# Module cảm biến đeo: Luồng làm việc



## Quy trình

- Điều phối từ thu thập dữ liệu cảm biến đến xử lý sự kiện và kích hoạt cảnh báo.
- Mô hình hóa luồng làm việc:

## Mô tả

- Lưu đồ luồng làm việc của môđun phát hiện té ngã trên ESP32.

# Module cảm biến đeo: Môi trường phát triển

## Chi tiết

- Phần mềm xây dựng trên **ESP-IDF** (Espressif IoT Development Framework).
- Hệ thống build: **CMake**, quản lý qua `idf_component.yml`.
- API: UART, I2C, SPI, PWM, GPIO, Wi-Fi, MQTT, HTTP.
- Cấu hình qua **Kconfig**, lưu trong `sdkconfig`.
- Đảm bảo ổn định, mở rộng, tương thích driver.

# Module cảm biến đeo: Cấu trúc project

```
1 mainproject/
2   └── main/
3     ├── main.c
4     ├── app_main.c
5     └── app_main.h
6   └── components/
7     ├── buzzer/
8     ├── comm/
9     ├── data_manager/
10    ├── event_handler/
11    ├── fall_logic/
12    ├── json_wrapper/
13    ├── led_indicator/
14    ├── mpu6050/
15    ├── sim4g_gps/
16    ├── user_mqtt/
17    └── wifi_connect/
```

# Module cảm biến đeo: Mô tả các thành phần

## Thành phần chính

- **main.c**: Gọi app\_main().
- **app\_main.c/h**: Điều phối, khởi tạo.
- **fall\_logic**: Thuật toán phát hiện té ngã.
- **event\_handler**: Phản ứng sự kiện.
- **mpu6050**: Driver cảm biến.

## Thành phần khác

- **buzzer, led\_indicator**: Cảnh báo cục bộ.
- **sim4g\_gps, wifi\_connect**: Kết nối từ xa.
- **user\_mqtt**: Giao tiếp MQTT.
- **comm, data\_manager**: Quản lý giao tiếp.

# Module cảm biến đeo: Sơ đồ khối phần mềm



Hình: Sơ đồ khối phần mềm của mô-đun nhúng ESP32.

# Module cảm biến đeo: Thuật toán phát hiện té ngã

## Cơ chế

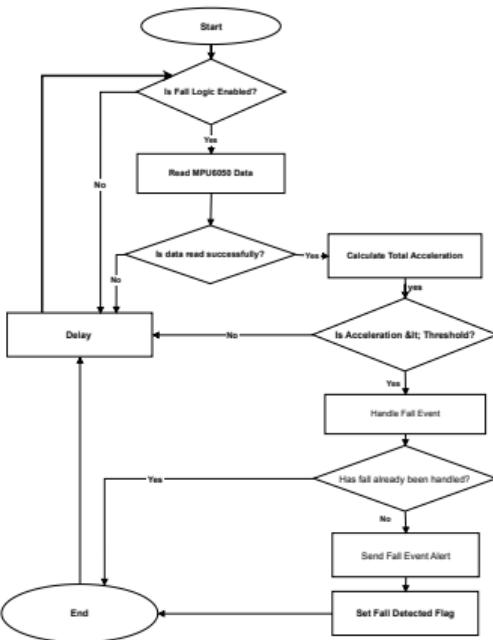
- Dựa trên **gia tốc tổng hợp**:

$$a_{total} = \sqrt{a_x^2 + a_y^2 + a_z^2}$$

- $a_{total} < \text{FALL\_THRESHOLD}$ : Đánh dấu sự kiện té ngã.
- Ngưỡng điều chỉnh qua `CONFIG_FALL_LOGIC_THRESHOLD_G`.
- Chạy trong **tác vụ FreeRTOS**, chu kỳ `CHECK_INTERVAL_MS`.

- ➊ Đọc dữ liệu từ mpu6050.
- ➋ Tính gia tốc tổng hợp, so sánh ngưỡng.
- ➌ Gửi `EVENT_FALL_DETECTED` tới `event_handler`.
- ➍ Đặt cờ `s_fall_detected`, reset qua `fall_logic_reset_fall_status()`.

# Module cảm biến đeo: Lưu đồ thuật toán component fall\_logic



Hình: Lưu đồ thuật toán phát hiện té ngã.

# Module camera: Tổng quan phần mềm

## Tổng quan

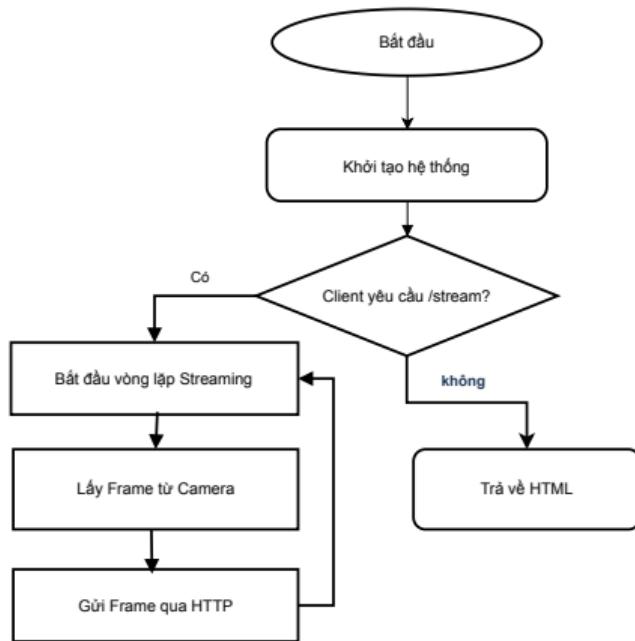
- Triển khai trên vi điều khiển **ESP32-S3**.
- Thu thập và truyền tải video từ camera **OV5640** theo thời gian thực.
- Cung cấp luồng video MJPEG qua HTTP cho các module xử lý thị giác máy tính.
- Dựa trên kiến trúc hướng sự kiện của **ESP-IDF**, quản lý đồng thời mạng, camera, và bộ đệm.

# Module camera: Luồng hoạt động chính

## Quy trình

- ① **Khởi tạo hệ thống:** Cấu hình Wi-Fi, camera, và máy chủ HTTP.
- ② **Chờ yêu cầu client:** Lắng nghe yêu cầu xem video.
- ③ **Lấy khung hình:** Sử dụng triple buffering trên PSRAM để giảm độ trễ.
- ④ **Gửi frame qua HTTP:** Frame JPEG đóng gói trong phản hồi HTTP multipart.
- ⑤ **Giám sát và xử lý lỗi:** Kiểm tra FPS, xử lý ngắt kết nối client.

# Module camera: Sơ đồ luồng hoạt động



Hình: Luồng hoạt động chính của phần mềm Module camera.

# Module camera: Cấu hình phần mềm với Kconfig

## Thông số chính

- **Wi-Fi:** SSID và Password để kết nối mạng.
- **Kích thước khung hình:** Từ QQVGA đến UXGA (chất lượng video, băng thông).
- **Chất lượng JPEG:** Giá trị 10–63 (độ nén hình ảnh).
- **Khoảng thời gian giữa các frame:** 0–200 ms (tốc độ khung hình, băng thông).

# Python server: Tích hợp và Phát hiện

## Tổng quan

- Hệ thống Python tích hợp dữ liệu từ:
  - ① **ESP32**: Trạng thái té ngã, GPS qua MQTT.
  - ② **Camera IP**: Phát hiện người, theo dõi, ước lượng tư thế bằng AI.
- Đồng bộ, tích hợp dữ liệu để phát hiện té ngã, gửi cảnh báo, lưu sự kiện.

# Python server: Kiến trúc

## Mô hình phân tầng

- **Lớp thu nhận:**
  - comm/: MQTT, Telegram Bot, AMI.
  - detection/: Xử lý video, phát hiện người, theo dõi.
- **Lớp xử lý:** fall/, processing/ tích hợp qua DetectionProcessor.
- **Lớp đầu ra:** database/ lưu fall\_events.db, comm/ gửi cảnh báo.

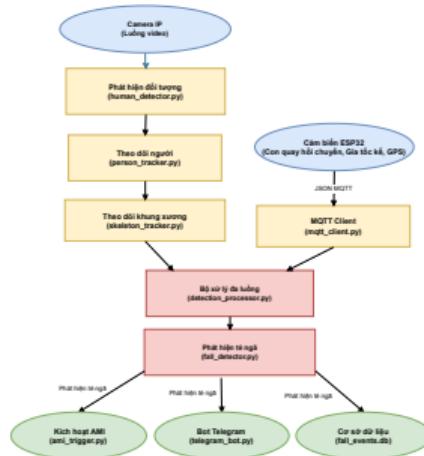
# Python server: Cấu trúc Thư mục

```
1 intergrate_fall/
2   ├── comm/ (MQTT, Telegram, AMI)
3   ├── config/ (câu hình)
4   ├── database/ (fall_events.db)
5   ├── detection/ (human, tracker)
6   ├── fall/ (fall_detector.py)
7   ├── processing/ (detection_processor.py)
8   ├── utils/ (draw_utils.py)
9   ├── models/ (yolov8n.pt)
10  └── tests/ (test_fall.py)
11    └── main.py
```

# Python server: Luồng Xử lý

## Quy trình

- 1 Thu nhận: ESP32 (MQTT), Camera IP (AI).
- 2 Tích hợp: DetectionProcessor đồng bộ dữ liệu.
- 3 Đầu ra: Lưu sự kiện, gửi cảnh báo qua AMI/Telegram.



# Python server: Tích hợp Dữ liệu

## Quy trình

- Xác thực: ESP32 (device\_id, fall\_detected, GPS).
- Xử lý camera: Phát hiện người, ước lượng tư thế.
- Đồng bộ: Kiểm tra timestamp, cooldown (5 phút).
- Cảnh báo: AMI, Telegram (thử lại nếu lỗi).

```
1  async def handle_camera_data(self, frame, person_id, box, landmarks):
2      entity_id = f"camera_person_{person_id}"
3      detector = self._get_or_create_detector(entity_id)
4      is_fall = detector.detect_fall(landmarks)
5      if is_fall and self._can_alert(entity_id):
6          fall_event = self._create_fall_event("camera", entity_id, 0, 0, False)
7          fall_id = await self._insert_fall_event_with_retry(fall_event)
8          alert_msg = f"Fall detected by camera for {entity_id}. ID: {fall_id}"
9          await self._send_alerts(alert_msg, frame)
```

# Python server: Thuật toán Phát hiện

## Quy trình trong FallDetector

- **Kiểm tra:** Landmarks hợp lệ (độ tin cậy  $\geq 0.5$ ).
- **Tính toán:**
  - Góc thân người: Dọc ( $> 60^\circ$ ), ngang ( $> 45^\circ$ ).
  - Vận tốc: Dịch chuyển thân người ( $> 0.5$  đơn vị).
- **Trạng thái:**
  - *Falling*: Góc và vận tốc vượt ngưỡng.
  - *Lying*: Góc vượt ngưỡng kéo dài.
- **Quyết định:** Bộ đếm  $\geq 5$  frame  $\rightarrow$  té ngã, đặt lại trạng thái.

## Python server: Lưu đồ Thuật toán



Hình: Luồng thuật toán phát hiện té ngã.

# Python server: Lưu trữ và Cảnh báo

## Cơ chế

- **Lưu trữ:** SQLite fall\_events.db (timestamp, source, entity\_id, GPS).
- **Cảnh báo:** AMI, Telegram khi phát hiện té ngã.
- **Vòng lặp:** Thu thập → Xử lý → Tích hợp → Cảnh báo.

```
1 CREATE TABLE fall_events (
2     id INTEGER PRIMARY KEY AUTOINCREMENT,
3     timestamp DATETIME DEFAULT CURRENT_TIMESTAMP,
4     source TEXT NOT NULL,
5     entity_id TEXT NOT NULL,
6     fall_detected BOOLEAN NOT NULL,
7     latitude REAL,
8     longitude REAL,
9     has_gps_fix BOOLEAN,
10    alert_status INTEGER DEFAULT 0
11 );
```

# Mục lục chương: IV-KẾT QUẢ THỰC NGHIỆM

1 I-GIỚI THIỆU

2 II-CƠ SỞ LÝ THUYẾT

3 III-THIẾT KẾ VÀ TRIỂN KHAI

4 IV-KẾT QUẢ THỰC NGHIỆM

- Thiết lập và kết quả thực nghiệm
- Kiểm thử từng thành phần

5 KẾT LUẬN CHUNG

# Thiết lập thực nghiệm

## Môi trường thử nghiệm

- Trong nhà, điều kiện lý tưởng
- Độ chính xác:** Phát hiện đúng té ngã
- Độ trễ:** Thời gian cảnh báo
- Độ ổn định:** Vận hành liên tục

## Cấu hình hệ thống

Thành phần	Thông số
Máy chủ	Intel i7-2630QM, GT 525M, Linux Mint
Phần mềm	Asterisk 22.4, Python 3.10.13
Cảm biến đeo	ESP32 + SIM, Wi-Fi/GSM
Camera	ESP32-S3 + webcam dự phòng
Di động	Smartphone + Linphone 6.0.17

# Kiểm thử Module 4G/GPS & Khởi tạo hệ thống

```
1 I (2329) SIM_4G: Received: Quectel EG800K
2 OK
3 I (5329) SIM_4G: Received: +CSQ: 31,99
4 OK
5 I (28339) SIM_4G: Received: +QIACT:
6   ↳ 1,1,1,"9.204.251.200"
7 OK
8 I (34349) SIM_4G: Received: +QGPSLOC:
9   ↳ 10.88862,106.77975
10 OK
11 I (9961) SIM4G_AT: Initializing SIM4G AT
   ↳ driver ...
I (10011) APP_MAIN: System initialization
   ↳ complete.
I (10021) APP_MAIN: Application started
   ↳ successfully
```

## Mục đích:

- Xác nhận giao tiếp AT command, kết nối 4G và thu nhận GPS.
- Kiểm tra các module SIM4G-GPS, Wi-Fi và các tác vụ chính.

## Quy trình:

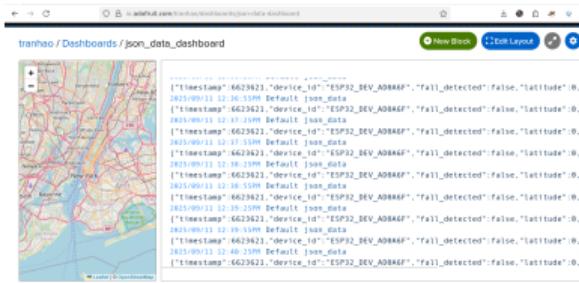
- ❶ Khởi tạo UART, kiểm tra modem và SIM.
- ❷ Đánh giá chất lượng sóng (+CSQ).
- ❸ Cấu hình APN, kích hoạt PDP context.
- ❹ Bật GPS và truy vấn tọa độ.

## Kiểm thử MQTT và truyền dữ liệu

- **Mục đích:** Kiểm tra kết nối broker MQTT và gửi bản tin định kỳ.
  - **Log tiêu biểu:**

```
I (19961) USER_MQTT:  
    ↳ MQTT_EVENT_CONNECTED  
I (39991) JSON_WRAPPER:  
    ↳ {"device_id": "ESP32_D
```

- **Kết luận:** Kết nối MQTT ổn định, dữ liệu được truyền tới hệ thống giám sát.



**Hình:** Dashboard hiển thị bản tin MQTT.

# Phát hiện Té ngã & Xử lý Cảnh báo

## ● Quy trình:

- ① Thuật toán ghi nhận chuỗi trạng thái  
LOW\_G → HIGH\_G.
- ② Kích hoạt cảnh báo: SMS, MQTT, buzzer,  
LED.

## ● Log thực tế:

```

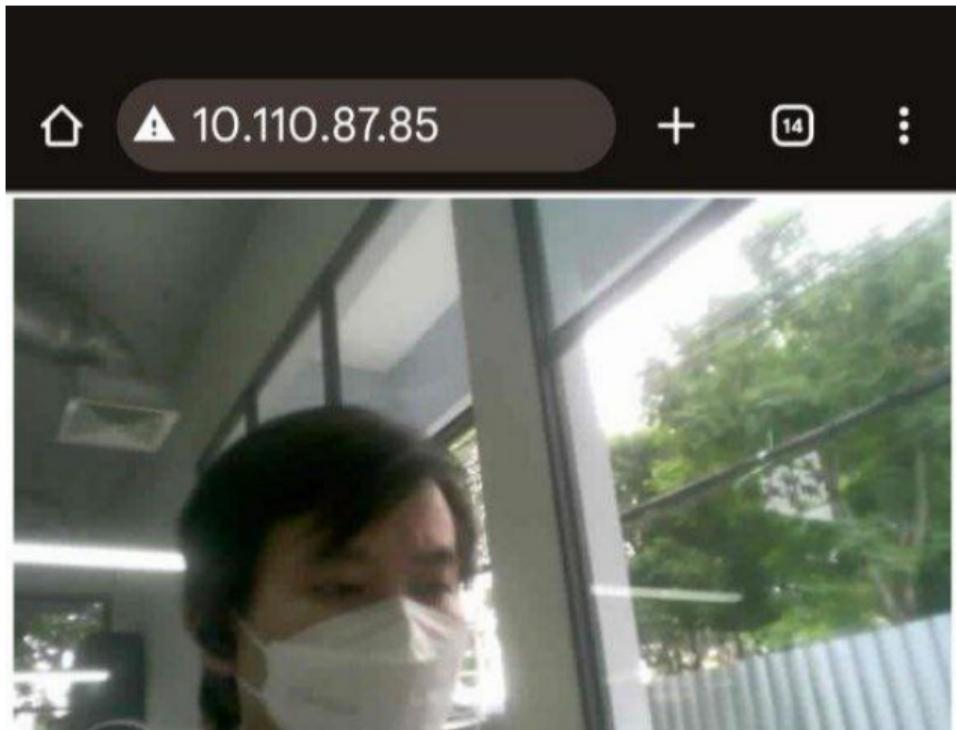
1 \alert{E (159131) FALL_LOGIC: FALL
2   ↳ DETECTED! Accel: 0.99 g}
3 I (159151) SIM4G_GPS: SMS request
4   ↳ queued successfully
5 I (159881) SIM4G_AT: SMS sent
6   ↳ successfully.

```



Hình: Log thực tế module cảm biến khi phát hiện té ngã.

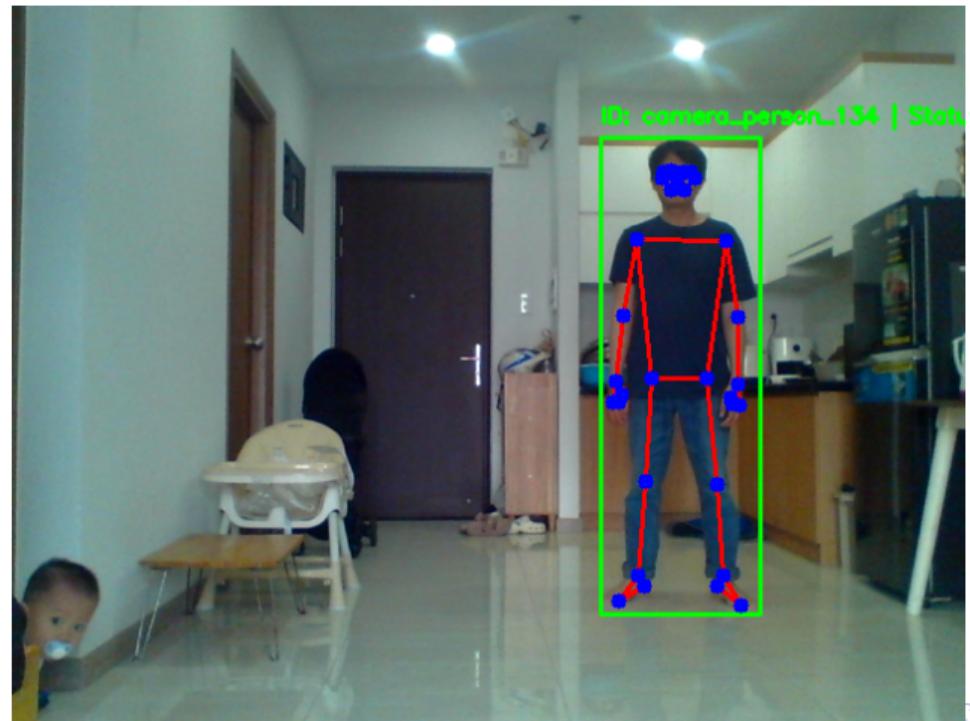
# Camera ESP32-CAM: Luồng hình ảnh và xử lý



- **Mục đích:** Kiểm tra kết nối Wi-Fi và phát luồng hình ảnh.
- **Xử lý Python:**
  - Nhận luồng video từ ESP32-CAM.
  - Tiền xử lý khung hình.
  - TensorFlow Lite phát hiện người, vẽ skeleton.
- **Kết quả:** Thời gian thực 3–5 FPS, hoạt động ổn định.
- **Ứng dụng:** Cảnh báo té ngã, đồng bộ MQTT, gửi thông báo Telegram.

# Xử lý nhận diện hình ảnh (Python)

- **Mục đích:** Phát hiện người và trích xuất skeleton từ luồng video.
- **Quy trình:**
  - Nhận luồng video từ ESP32-CAM.
  - Tiền xử lý khung hình.
  - TensorFlow Lite phát hiện người, vẽ skeleton.
- **Kết quả:** Thời gian thực 3–5 FPS, hoạt động ổn định.



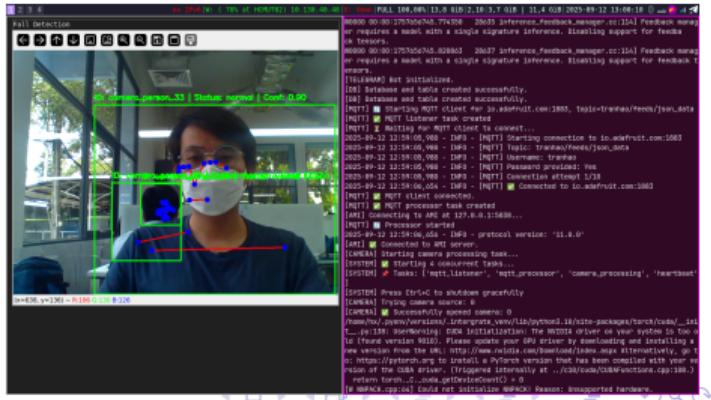
# Log Python

```
# Log Python xem lý luông camera và điều phôi
↪ cảnh báo
I (10521) PYTHON_APP: Frame received from
↪ ESP32-CAM
I (10535) PYTHON_APP: Preprocessing frame ...
I (10612) PYTHON_APP: Person detected, skeleton
↪ extracted
I (10635) MQTT_CLIENT: Fall alert sent
↪ successfully
```

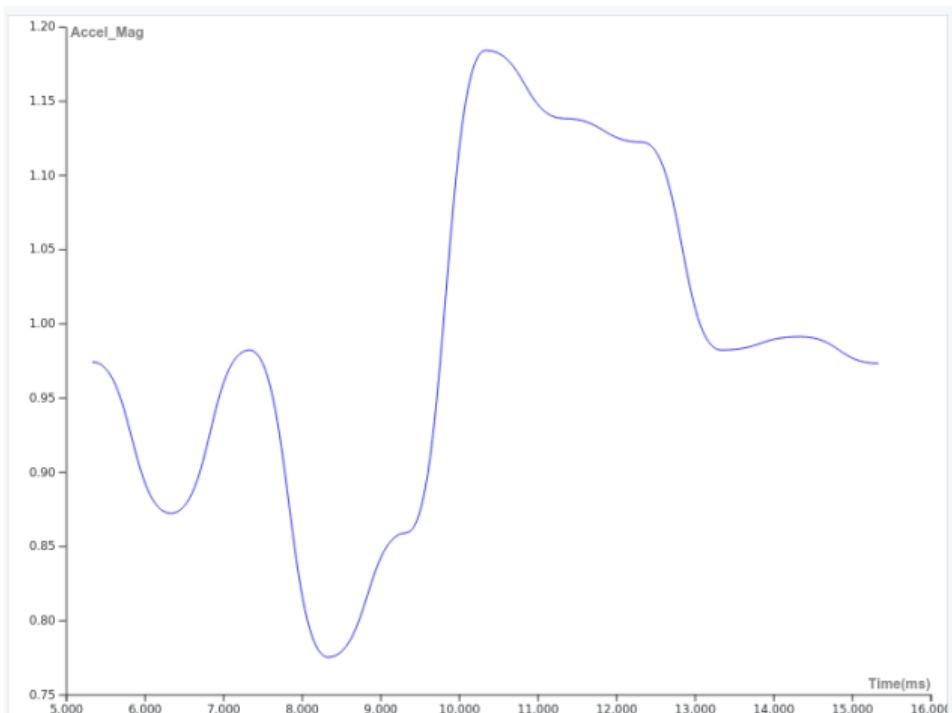
- Kết luận:

- Module Python hoạt động ổn định.
  - Đồng bộ MQTT và kích hoạt cảnh báo thành công.

- **Hình minh họa (tùy chọn):**



# Đánh giá dữ liệu cảm biến (MPU6050) – Accel\_Mag



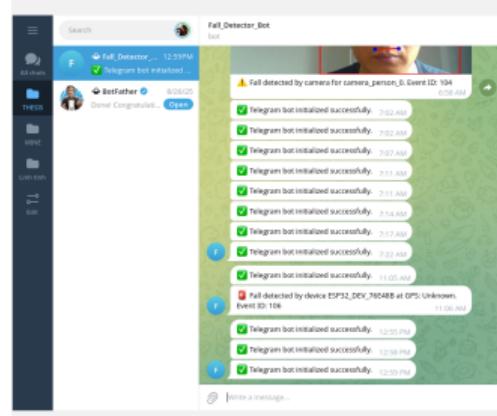
- **Quy trình:** Giả lập té ngã, so sánh dữ liệu bình thường và té ngã.
- **Kết quả:**
  - **Gyro (dps):** Bình thường  $\approx \pm 1.5$ ; té ngã tăng  $\approx \pm 250$ .
  - **Accel (g):** Bình thường  $\approx 0.93$ ; té ngã  $-2.0$  đến  $+1.0$ .
- **Phân tích:**
  - **Bình thường:** Gần 1 g, dao động nhỏ.
  - **Sự kiện té ngã:** Xuất hiện xung hoặc thay đổi đột ngột.
  - **Hậu té:** Trở về gần 1 g nhưng

# Kiểm thử cảnh báo qua Telegram

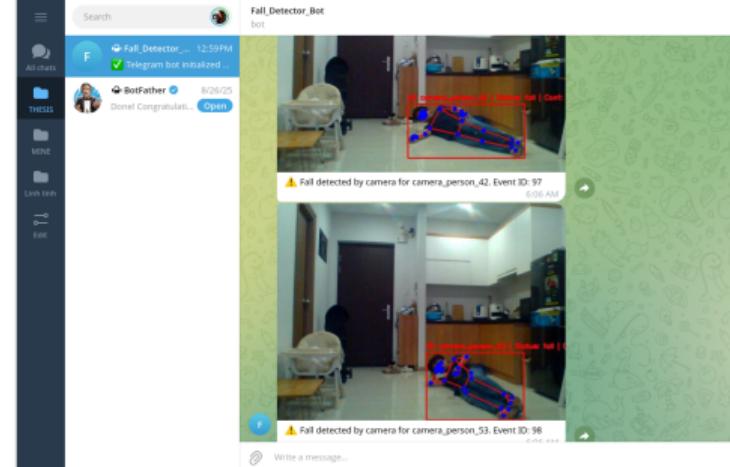
- **Cơ chế hoạt động:**

- **Phần cứng:** MQTT từ thiết bị -> trung tâm -> Telegram.
- **Python:** Camera -> Python gửi cảnh báo trực tiếp Telegram kèm hình.

# Thông báo Telegram



Hình: Thông báo từ phần cứng.



Hình: Thông báo từ Python.

# Đánh giá kết quả thực nghiệm

## Độ trễ hệ thống (Latency)

- **ESP32:** đọc cảm biến, xử lý và phát cảnh báo.
- **Mạng:** truyền gói MQTT tới broker.
- **Dịch vụ:** Python nhận dữ liệu, kích hoạt bot Telegram.
- **Tổng thể:** độ trễ < 5 giây, đáp ứng gần thời gian thực.

## Đánh giá độ ổn định & hiệu năng

- Hệ thống hoạt động liên tục, không tràn bộ nhớ hay reset đột ngột.
- Thuật toán phân biệt rõ trạng thái bình thường và té ngã.
- Dữ liệu phối hợp chính xác, phần mềm Python ổn định.
- Giật/lag do phần cứng, không ảnh hưởng tổng thể.

# Mục lục chương: KẾT LUẬN CHUNG

1 I-GIỚI THIỆU

2 II-CƠ SỞ LÝ THUYẾT

3 III-THIẾT KẾ VÀ TRIỂN KHAI

4 IV-KẾT QUẢ THỰC NGHIỆM

5 KẾT LUẬN CHUNG

# Kết quả chung & Hướng phát triển

## Kết quả chính

- Hệ thống IoT: ESP32, MPU6050, SIM4G–GPS, buzzer, LED, Python/FreeRTOS.
- Thuật toán phát hiện té ngã: sensor fusion dữ liệu cảm biến + ảnh, thời gian thực.
- Cơ chế cảnh báo đa dạng: buzzer, LED, SMS, MQTT–Telegram, SIP.
- Thực nghiệm: độ chính xác tương đối, độ trễ thấp, hoạt động ổn định.
- Tiềm năng ứng dụng: giám sát sức khỏe từ xa

## Hạn chế & hướng phát triển

- Quy mô thử nghiệm hạn chế; số lượng thiết bị kênh truyền còn nhỏ.
- Thuật toán chưa hoàn toàn ổn định; thiết bị còn cồng kềnh.
- Kết nối 4G/GPS chưa tối ưu và tin cậy
- **Hướng phát triển:** mở rộng thử nghiệm, cải tiến thuật toán, thiết bị nhỏ gọn, hoàn thiện kết nối 4G/GPS, tích hợp nền tảng IoT.

# Kết luận chung & Cảm ơn

- FDAS nguyên mẫu hoạt động thành công: phát hiện té ngã gửi cảnh báo thời gian thực.
- Độ trễ đầu-cuối < 5 giây; hệ thống ổn định, hiệu năng đáp ứng yêu cầu.
- Nghiên cứu mang lại kiến thức kinh nghiệm thực tiễn về IoT, nhúng, sensor fusion, hệ thống phân tán.
- Xác định các thách thức thực tế: tối ưu phần cứng, nâng cao độ chính xác thuật toán, ổn định truyền thông.

**Cảm ơn quý thầy/cô và các bạn đã chú ý theo dõi**