TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN &

TRUYỀN THÔNG VIỆT HÀN

**Khoa Kỹ Thuật Máy Tính & Điện Tử**



**BÁO CÁO ĐỒ ÁN CHUYÊN NGÀNH 3**

**ĐỀ TÀI: ỨNG DỤNG UART TRONG KẾT NỐI ESP32-CAM VÀ ESP32 CHO HỆ THỐNG ĐÈN PHA MA TRẬN**

|  |  |  |
| --- | --- | --- |
| **Sinh viên thực hiện** | **:** | **Doãn Cao Danh – 21CE074**  **Vũ Gia Bảo – 21CE068** |
| **Lớp** | **:** | **21CE2** |
| **Giảng viên hướng dẫn** | **:** | **TS.NGUYỄN VŨ ANH QUANG** |

**Đà Nẵng, tháng 05 năm 2025**

TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN &

TRUYỀN THÔNG VIỆT HÀN

**Khoa Kỹ Thuật Máy Tính & Điện Tử**



**BÁO CÁO ĐỒ ÁN CHUYÊN NGÀNH 3**

**ĐỀ TÀI: ỨNG DỤNG UART TRONG KẾT NỐI ESP32-CAM VÀ ESP32 CHO HỆ THỐNG ĐÈN PHA MA TRẬN**

|  |  |  |
| --- | --- | --- |
| **Sinh viên thực hiện** | **:** | **Doãn Cao Danh – 21CE074**  **Vũ Gia Bảo – 21CE068** |
| **Lớp** | **:** | **21CE2** |
| **Giảng viên hướng dẫn** | **:** | **TS. NGUYỄN VŨ ANH QUANG** |

**Đà Nẵng, tháng 05 năm 2025**

**NHẬN XÉT CỦA GIẢNG VIÊN HƯỚNG DẪN**

……………………………………………………………………………

……………………………………………………………………………

……………………………………………………………………………

……………………………………………………………………………

……………………………………………………………………………

……………………………………………………………………………

……………………………………………………………………………

……………………………………………………………………………

……………………………………………………………………………

……………………………………………………………………………

……………………………………………………………………………

……………………………………………………………………………

*Đà Nẵng, tháng 05 năm 2025*

*Giáo viên hướng dẫn*

**LỜI CẢM ƠN**

Nhóm em xin gửi lời cảm ơn chân thành nhất tới Thầy/Cô và tất cả mọi người đã hỗ trợ và động viên trong quá trình thực hiện đề tài này.

Đề tài này không chỉ mang lại cho nhóm em cơ hội được tiếp xúc và áp dụng những kiến thức chuyên sâu trong lĩnh vực điện tử và công nghệ thông minh mà còn giúp nhóm em phát triển kỹ năng tư duy và năng lực sáng tạo của bản thân.

Nhóm em muốn bày tỏ lòng biết ơn sâu sắc đến sự hướng dẫn tận tình và những góp ý quý báu của Thầy/Cô. Đồng thời, nhóm em cũng muốn gửi lời tri ân đến giảng viên hướng dẫn Thầy Nguyễn Vũ Anh Quang đã chia sẻ, hỗ trợ và làm việc cùng nhóm em trong suốt thời gian nghiên cứu và thực hiện đề tài này.

Đề tài "Ứng dụng UART trong kết nối ESP32-CAM và ESP32 cho hệ thống đèn pha ma trận" không chỉ là một nỗ lực của nhóm mà còn là sự hợp tác, chia sẻ và học hỏi từ tất cả mọi người. Một lần nữa, nhóm em xin chân thành cảm ơn tất cả mọi người đã đồng hành và ủng hộ cho đồ án này.

*Đà Nẵng, tháng 05 năm 2025*

*Sinh viên thực hiện*

Doãn Cao Danh

Vũ Gia Bảo

**MỤC LỤC**

[1. Giới thiệu. 8](#_Toc198294821)

[2. Mục tiêu của đề tài. 8](#_Toc198294826)

[3. Nội dung và kế hoạch thực hiện 8](#_Toc198294827)

[a) Nội dung thực hiện 8](#_Toc198294828)

[b) Kế hoạch thực hiện 9](#_Toc198294829)

[CHƯƠNG 1: TỔNG QUAN VỀ ĐỀ TÀI 10](#_Toc198294830)

[1. Phân tích yêu cầu đề tài. 10](#_Toc198294831)

[Tên đề tài: 10](#_Toc198294832)

[Chức năng: 10](#_Toc198294833)

[2. Các ứng dụng đề tài. 10](#_Toc198294834)

[3. Công nghệ sử dụng. 11](#_Toc198294835)

[Proteus – Phần mềm mô phỏng mạch điện tử: 11](#_Toc198294836)

[CHƯƠNG 2. CƠ SỞ LÝ THUYẾT 18](#_Toc198294837)

[1. Tổng quan về ứng dụng UART trong kết nối ESP32-CAM và ESP32 cho hệ thống đèn pha ma trận. 18](#_Toc198294838)

[3. Huấn luyện mô hình AI bằng Edge Impulse: 30](#_Toc198294839)

[CHƯƠNG 3: THIẾT KẾ MẠCH NGUYÊN LÝ, THI CÔNG VÀ KẾT QUẢ 32](#_Toc198294840)

[1. Sơ đồ hoạt động của mạch ứng dụng UART trong kết nối ESP32-CAM và ESP32 cho hệ thống đèn pha ma trận . 32](#_Toc198294841)

[3. Lưu đồ thuật toán: 34](#_Toc198294842)

[4. Bố cục PCB và hiển thị 3D của mạch: 36](#_Toc198294843)

[5. Kết quả mạch. 39](#_Toc198294844)

[CHƯƠNG 4: KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN 48](#_Toc198294845)

[1. Kết luận. 48](#_Toc198294846)

[2. Hướng phát triển. 48](#_Toc198294847)

[TÀI LIỆU THAM KHẢO 49](#_Toc198294848)

**DANH MỤC HÌNH ẢNH**

Hình 1.1: Phần mềm Proteus

Hình 1.2: Phần mềm Arduino

Hình 1.3: Phần mềm Edge Impulse

Hình 1.4: Đèn LED ma trận xe Audi

Hình 2.1: Mô phỏng đèn LED ma trận

Hình 2.2: Giao tiếp UART và mạch chuyển USB UART TTL FT323RL

Hình 2.3: ESP32-CAM và sơ đồ chân

Hình 2.4: ESP32 và sơ đồ chân

Hình 2.5: Module Relay 1 kênh 5V H/L

Hình 2.6: Nguồn Adapter 5V

Hình 2.7: Mô phỏng huấn luyện mô hình AI bằng Edge Impulse

Hình 3.1: Sơ đồ nguyên lý của mạch

Hình 3.2: Sơ đồ khối

Hình 3.3: Lưu đồ thuật toán

Hình 3.4: Bố cục PCB ESP32

Hình 3.5: Hiển thị 3D mặt trước ESP32-CAM

Hình 3.6: Hiển thị 3D mặt sau ESP-CAM

Hình 3.7: Hiển thị 3D mặt trước ESP32

Hình 3.8: Hiển thị 3D mặt sau ESP32

Hình 3.9: Kết quả thực tế

Hình 3.10: Giao diện train Edge Impulse

Hình 3.11: Biểu đồ Percision/Recall/F1 Score theo Epoch

Hình 3.12: Biểu đồ Train Loss và Validation Loss theo Epoch

**MỞ ĐẦU**

## Giới thiệu.

## Trong bối cảnh công nghệ ngày càng phát triển, việc kết nối và giao tiếp giữa các vi điều khiển đóng vai trò then chốt trong việc xây dựng các hệ thống nhúng thông minh. Một trong những chuẩn giao tiếp nối tiếp phổ biến và hiệu quả hiện nay là UART (Universal Asynchronous Receiver/Transmitter), nhờ tính đơn giản, dễ triển khai và độ tin cậy cao trong truyền nhận dữ liệu.

## Đề tài “Ứng dụng UART trong kết nối ESP32-CAM và ESP32 cho hệ thống đèn pha ma trận” được thực hiện với mục tiêu tìm hiểu cách sử dụng UART để thiết lập đường truyền dữ liệu giữa hai vi điều khiển – một ESP32-CAM có tích hợp camera dùng để thu thập hoặc xử lý hình ảnh, và một ESP32 khác đảm nhiệm vai trò điều khiển hiển thị trên đèn pha ma trận.

## Trong hệ thống này, ESP32-CAM có thể thực hiện các tác vụ như nhận dạng khuôn mặt hoặc quét hình ảnh, sau đó gửi dữ liệu hoặc lệnh điều khiển đến ESP32 qua UART. ESP32 sẽ tiếp nhận thông tin này và điều khiển đèn pha ma trận hiển thị nội dung tương ứng. Giải pháp này giúp phân chia hợp lý khối lượng xử lý và tận dụng được thế mạnh riêng của từng module ESP32.

## Việc triển khai đề tài không chỉ giúp nhóm em củng cố kiến thức về UART, vi điều khiển ESP32 và LED ma trận, mà còn tạo nền tảng để phát triển các ứng dụng thực tiễn như hệ thống hiển thị thông minh, biển báo giao thông điện tử hoặc các mô hình giám sát tích hợp camera và hiển thị.

## Mục tiêu của đề tài.

* Tìm hiểu và ứng dụng giao tiếp UART giữa hai vi điều khiển ESP32.
* Xây dựng hệ thống giao tiếp ổn định giữa ESP32-CAM và ESP32 thông qua UART.
* Thiết kế và lập trình điều khiển đèn pha ma trận bằng ESP32 dựa trên dữ liệu nhận từ ESP32-CAM.
* Tích hợp và kiểm thử toàn bộ hệ thống nhằm đảm bảo khả năng hoạt động đồng bộ và ổn định.
* Nâng cao khả năng lập trình nhúng, tích hợp phần cứng và phát triển hệ thống ứng dụng thực tiễn.

## Nội dung và kế hoạch thực hiện

### a) Nội dung thực hiện

Các nội dung cần thực hiện trong đề tài :

* **Tìm hiểu và nghiên cứu công nghệ:**
* Nguyên lý hoạt động và các tính năng thông minh của đèn pha ma trận trên xe hơi.
* Cơ chế điều khiển ánh sáng để tránh gây chói mắt cho xe đối diện.
* Nguyên lý hoạt động của UART trong truyền dữ liệu giữa ESP32-CAM và ESP32.
* Các phương pháp tối ưu truyền thông nối tiếp giữa hai vi điều khiển.
* Cách ESP32-CAM nhận diện vật thể và xử lý hình ảnh.
* Áp dụng thuật toán nhận diện để thay đổi chế độ chiếu sáng của đèn pha.
* **Thiết kế và chế tạo hệ thống:**
  + Kết nối phần cứng giữa ESP32-CAM, ESP32 và đèn pha ma trận.
  + Viết chương trình điều khiển bằng Arduino IDE để xử lý nhận diện và điều khiển chiếu sáng.
* **Thử nghiệm và đánh giá:**
* Tiến hành thử nghiệm thực tế hệ thống nhận diện vật thể và điều khiển ánh sáng.
* Đánh giá hiệu suất hoạt động, độ ổn định của giao tiếp UART và chất lượng chiếu sáng.
* Đề xuất phương án cải tiến nếu cần thiết.
* **Tài liệu hóa và báo cáo:** Tạo tài liệu hóa đầy đủ về mạch điều khiển, bao gồm sơ đồ mạch, bảng mạch và hướng dẫn sử dụng. Viết báo cáo tổng kết về quá trình thực hiện đề tài, kết quả thử nghiệm và đánh giá hiệu suất của sản phẩm.

**Làm slide thuyết trình về dự án trang web**

* Công cụ thực hiện: PowerPoint.

**Làm báo cáo dự án**

* Công cụ thực hiện: MS Word

### b) Kế hoạch thực hiện

|  |  |
| --- | --- |
| **Thời gian** | **Nội dung thực hiện** |
| Từ 20/03 đến 01/04 | - Lên ý tưởng thiết kế mạch. |
| Từ 02/04 đến 12/04 | - Liệt kê linh kiện và thiết kế sơ đồ mạch. |
| Từ 15/04 đến 08/05 | - Xây dựng và thử nghiệm sản phẩm. |
| Từ 08/05 đến 09/05 | - Làm slide thuyết trình và báo cáo. |

# CHƯƠNG 1: TỔNG QUAN VỀ ĐỀ TÀI

## Phân tích yêu cầu đề tài.

### Tên đề tài:

**“**Ứng dụng UART trong kết nối ESP32-CAM và ESP32 cho hệ thống đèn pha ma trận”

### Chức năng:

Hệ thống có nhiều chức năng như sau:

* **Chụp ảnh và xử lý hình ảnh từ ESP32-CAM:**
* ESP32-CAM có nhiệm vụ ghi nhận hình ảnh phía trước xe (hoặc mô phỏng cảnh vật phía trước).
* Dữ liệu hình ảnh có thể được xử lý để xác định các tình huống như có vật cản phía trước ,....
* **Truyền dữ liệu qua UART giữa ESP32-CAM và ESP32:**
* Giao tiếp UART là cầu nối chính để truyền thông tin điều khiển từ ESP32-CAM (camera) sang ESP32 (điều khiển LED).
* Dữ liệu truyền có thể là tín hiệu nhận diện hoặc trạng thái môi trường, ví dụ: “có vật cản”, “oto”,...
* **Phản hồi theo tình huống thực tế mô phỏng:**
* Hệ thống phản ứng thông minh tùy theo dữ liệu từ ESP32-CAM:
* Có chướng ngại vật → tự động điều chỉnh đèn phù hợp với điều kiện được lập trình sẵn.
* Không có sự kiện đặc biệt → duy trì chế độ DRL (Daytime Running Light): Đèn sáng ở mức vừa phải.
* **Khả năng tùy chỉnh và mở rộng:**
* Có thể thêm nút điều khiển để test các chế độ thủ công trong quá trình thử nghiệm.

## Các ứng dụng đề tài.

Ứng dụng UART trong kết nối ESP32-CAM và ESP32 cho hệ thống đèn pha ma trận có thể được áp dụng trong nhiều ứng dụng khác nhau, bao gồm:

* **Mô phỏng hệ thống đèn pha chiếu sáng thông minh (Adaptive Driving Beam – ADB):**
* Hệ thống LED được chia theo vùng và điều khiển độ sáng từng vùng một cách linh hoạt dựa trên dữ liệu từ camera.
* Khi phát hiện xe hoặc vật cản ngược chiều, hệ thống sẽ tự động giảm hoặc tắt ánh sáng ở vùng đó để tránh gây chói mắt, giống như công nghệ trên xe Audi, Mercedes,...
* **Nền tảng nghiên cứu và đào tạo về công nghệ đèn pha ma trận trong ô tô:**
* Mô hình mô phỏng giúp sinh viên, kỹ sư hoặc nhà nghiên cứu hiểu rõ nguyên lý hoạt động của hệ thống ADB, từ xử lý hình ảnh → phân vùng ánh sáng → điều chỉnh độ sáng thực tế.
* Có thể dùng trong các môn học về điện – điện tử ô tô, hệ thống điều khiển chiếu sáng thông minh, hoặc IoT tích hợp camera.
* **Thử nghiệm thuật toán phát hiện phương tiện và điều chỉnh vùng chiếu sáng:**
* Sử dụng ESP32-CAM để nhận diện vật thể (xe, vật cản) và tính toán vị trí tương đối trên vùng chiếu sáng.
* Từ đó điều khiển LED sáng – tối theo vùng tương ứng trên mạch LED mô phỏng.
* Hệ thống có thể thử nghiệm với các thuật toán phân vùng ánh sáng, làm mờ mềm mại, hoặc tắt sáng cục bộ.
* **Tích hợp vào hệ thống chiếu sáng thông minh ngoài giao thông:**
* Ý tưởng điều khiển vùng sáng theo đối tượng không chỉ áp dụng cho ô tô, mà còn mở rộng ra chiếu sáng trong thành phố:
* Khi có vật cản hoặc xe đi ngang, hệ thống làm mờ ánh sáng hướng về phía họ để tạo cảm giác dễ chịu.
* Khi không có vật thể, đèn hoạt động ở chế độ chiếu sáng bình thường, tiết kiệm năng lượng.
* **Tiền đề phát triển hệ thống đèn pha ma trận trong xe mô hình hoặc xe thực tế:**
* Có thể mở rộng hệ thống này để gắn lên mô hình xe thu nhỏ, mô phỏng hành vi thực tế trên đường.
* Từ nguyên mẫu mô phỏng, có thể nâng cấp thành phiên bản ứng dụng được trong hệ thống đèn chiếu sáng thực sự của xe điện hoặc xe tự chế.

## Công nghệ sử dụng.

### Proteus – Phần mềm mô phỏng mạch điện tử:

Proteus là một phần mềm mô phỏng mạch điện tử rất phổ biến trong giảng dạy và nghiên cứu kỹ thuật. Nó cho phép người dùng thiết kế, kiểm tra và mô phỏng các mạch điện trước khi triển khai trên phần cứng thực tế.

* **Các tính năng nổi bật của Proteus:**
* Cho phép mô phỏng mạch tích hợp vi điều khiển với các linh kiện như LED, cảm biến, mô-đun truyền thông...
* Hỗ trợ lập trình mô phỏng trực tiếp với file HEX hoặc BIN từ Arduino.
* Giúp kiểm tra nhanh tín hiệu, dòng điện, điện áp trên các chân mạch mà không cần mạch vật lý.
* Giảm thiểu sai sót trước khi đi vào lắp ráp và thử nghiệm thực tế.
* **Lịch sử hình thành:**

Proteus Design Suite được phát triển bởi Labcenter Electronics Ltd. tại Yorkshire, Anh Quốc. Phần mềm này được sử dụng chủ yếu bởi các kỹ sư và kỹ thuật viên điện tử để tạo sơ đồ nguyên lý và bản in điện tử cho việc sản xuất mạch in (PCB).

* 1988: Phiên bản đầu tiên của Proteus, được gọi là PC-B, được viết bởi John Jameson, chủ tịch công ty, cho hệ điều hành DOS.
* 1990: Bổ sung tính năng Schematic Capture, cho phép người dùng vẽ sơ đồ nguyên lý.
* 1996: Tích hợp mô phỏng SPICE hỗn hợp, cho phép mô phỏng cả mạch tương tự và số.
* 1998: Thêm khả năng mô phỏng vi điều khiển, hỗ trợ nhiều dòng vi điều khiển như PIC, AVR, ARM, v.v.
* 2002: Bổ sung tính năng định tuyến tự động dựa trên hình dạng (Shape based autorouting).
* 2006: Cập nhật lớn với việc thêm tính năng hiển thị bảng mạch 3D (3D Board Visualisation).
* 2011: Ra mắt môi trường phát triển tích hợp (IDE) chuyên dụng cho mô phỏng.
* 2015: Thêm khả năng nhập/xuất
* **Đặc điểm của phần mềm proteus:**
* Tích hợp mô phỏng và thiết kế mạch:

Proteus cho phép người dùng thực hiện toàn bộ quy trình từ thiết kế sơ đồ nguyên lý, mô phỏng mạch điện đến thiết kế mạch in PCB ngay trên một nền tảng duy nhất.

* Mô phỏng vi điều khiển với mã chương trình thật:

Proteus hỗ trợ mô phỏng các dòng vi điều khiển phổ biến như PIC, AVR, 8051, ARM, ESP32,… với khả năng nạp trực tiếp mã chương trình (HEX, ELF, COF) vào chip. Người dùng có thể kiểm tra tương tác giữa phần cứng và phần mềm một cách trực quan.

* Thư viện linh kiện đa dạng, dễ mở rộng:

Phần mềm cung cấp thư viện linh kiện phong phú, từ các linh kiện cơ bản đến các module phức tạp. Ngoài ra, người dùng có thể tự tạo hoặc thêm linh kiện mới vào thư viện.

* Mô phỏng tín hiệu tương tự và số:

Proteus có khả năng mô phỏng cả tín hiệu analog và digital, giúp phân tích chi tiết hoạt động của mạch hỗn hợp.

* Thiết kế và mô phỏng mạch in (PCB):

Công cụ vẽ mạch in hỗ trợ nhiều lớp, định tuyến tự động hoặc thủ công, kiểm tra va chạm và tạo file Gerber để đưa vào sản xuất.

* Giao diện thân thiện, dễ sử dụng:

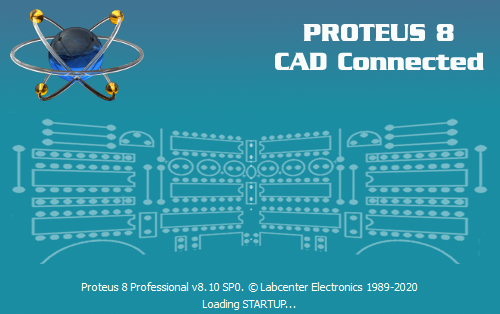
Proteus có giao diện trực quan, dễ làm quen đối với sinh viên và người mới bắt đầu, đồng thời vẫn đáp ứng tốt nhu cầu của người dùng chuyên nghiệp.

* Hỗ trợ mô phỏng thời gian thực:

Cho phép theo dõi và kiểm tra hoạt động của mạch như khi chạy trên phần cứng thực tế, giúp tiết kiệm thời gian thử nghiệm và phát hiện lỗi sớm.

* **Ứng dụng trong đề tài:**

Proteus được dùng để xây dựng sơ đồ nguyên lý mạch giữa ESP32, đèn LED và các phần điều khiển liên quan. Việc mô phỏng sơ bộ giúp người viết kiểm tra trước khả năng hoạt động, logic và dòng dữ liệu giữa các thành phần.



*Hình 1.1: Phần mềm Proteus*

**Arduino IDE – Môi trường lập trình cho vi điều khiển:**

Arduino IDE là một công cụ lập trình mã nguồn mở, rất phổ biến cho việc phát triển ứng dụng trên các dòng vi điều khiển như Arduino Uno, ESP32, ESP8266,... Giao diện đơn giản, thư viện phong phú giúp người mới học dễ tiếp cận.

* **Các tính năng nổi bật:**
* Hỗ trợ rất nhiều board vi điều khiển, đặc biệt là ESP32 và ESP32-CAM.
* Cung cấp thư viện lập trình dễ dùng cho UART, điều khiển LED, xử lý ảnh,...
* Có Serial Monitor và Plotter giúp kiểm tra, giám sát tín hiệu trong thời gian thực.
* **Lịch sử phát triển:**

Arduino được hình thành vào năm 2005 tại Viện Thiết kế Tương tác Ivrea (Interaction Design Institute Ivrea - IDII) ở Ý, với mục tiêu tạo ra một nền tảng phần cứng và phần mềm mã nguồn mở, chi phí thấp, dễ sử dụng cho sinh viên và những người không chuyên về kỹ thuật điện tử.

**Giai đoạn khởi đầu (2003-2005):**

* Năm 2003, Hernando Barragán, dưới sự hướng dẫn của Massimo Banzi và Casey Reas, phát triển nền tảng Wiring như một phần của luận án thạc sĩ tại IDII. Wiring bao gồm một bo mạch vi điều khiển ATmega128, môi trường phát triển dựa trên Processing và thư viện lập trình đơn giản hóa việc điều khiển phần cứng.
* Năm 2005, Massimo Banzi cùng với David Mellis và David Cuartielles mở rộng Wiring bằng cách hỗ trợ vi điều khiển ATmega8 có chi phí thấp hơn. Dự án mới này được tách ra và đặt tên là Arduino, lấy theo tên của một quán cà phê ở Ivrea nơi các nhà sáng lập thường gặp mặt.

**Phát triển và mở rộng (2005-2010):**

* Arduino nhanh chóng thu hút sự chú ý của cộng đồng nhờ vào tính đơn giản, chi phí thấp và khả năng mở rộng.
* Các phiên bản bo mạch như Arduino Uno, Mega và Nano được giới thiệu, đáp ứng nhu cầu đa dạng từ người mới bắt đầu đến các dự án phức tạp.
* Đến năm 2011, ước tính đã có hơn 300.000 bo mạch Arduino chính thức được sản xuất và phân phối trên toàn cầu.

**Tranh chấp thương hiệu và hợp nhất (2014-2017):**

* Năm 2014, xảy ra tranh chấp giữa các nhà sáng lập về quyền sở hữu thương hiệu Arduino, dẫn đến việc tồn tại hai thực thể: Arduino LLC (arduino.cc) và Arduino SRL (arduino.org).
* Năm 2015, Arduino LLC giới thiệu thương hiệu Genuino cho thị trường ngoài Hoa Kỳ nhằm phân biệt với Arduino SRL.
* Đến tháng 10 năm 2016, hai bên đạt được thỏa thuận hợp nhất, thành lập Arduino AG, với mục tiêu thống nhất và phát triển cộng đồng Arduino.

**Phát triển hiện đại (2017-nay):**

* Arduino tiếp tục mở rộng với việc giới thiệu các sản phẩm mới như Arduino MKR series, Arduino Nano 33 series và Arduino Portenta, hỗ trợ các ứng dụng IoT và công nghiệp.
* Năm 2019, Arduino ra mắt dịch vụ Arduino IoT Cloud, cho phép người dùng dễ dàng xây dựng và quản lý các thiết bị IoT
* Năm 2022, Arduino IDE 2.0 được phát hành, mang đến giao diện hiện đại, hỗ trợ gỡ lỗi và cải thiện hiệu suất lập trình.
* Đến năm 2020, cộng đồng Arduino đã đạt khoảng 30 triệu người dùng hoạt động, phản ánh sự phổ biến và ảnh hưởng rộng rãi của nền tảng này trong lĩnh vực điện tử và lập trình nhúng.

**Đặc điểm của phần mềm Arduino:**

Arduino sở hữu nhiều đặc điểm nổi bật giúp nó trở thành một trong những nền tảng phát triển phần cứng phổ biến nhất hiện nay. Cụ thể:

* Arduino có mã nguồn mở (Open-source):

Cả phần mềm (Arduino IDE) và phần cứng (sơ đồ mạch) đều được công bố công khai, cho phép người dùng tự do chỉnh sửa, chia sẻ hoặc phát triển các biến thể phù hợp với nhu cầu riêng.

* Arduino là một phần mềm dễ sử dụng:

Giao diện lập trình trực quan, cú pháp đơn giản, cùng với hệ sinh thái tài liệu, thư viện và ví dụ phong phú giúp người mới bắt đầu dễ dàng tiếp cận mà không cần nền tảng kỹ thuật sâu.

* Arduino có tính linh hoạt cao:

Arduino có thể kết nối với nhiều loại cảm biến, động cơ, màn hình, module giao tiếp (WiFi, Bluetooth, LoRa, v.v.) giúp triển khai đa dạng ứng dụng từ đơn giản đến phức tạp.

* Arduino tương thích nhiều phần cứng:

Hỗ trợ nhiều dòng vi điều khiển khác nhau như: Atmega328 (Uno), Atmega2560 (Mega), SAMD21 (MKR), ESP32/ESP8266 (IoT), v.v…, đáp ứng tốt cho cả các dự án học thuật và công nghiệp.

* Arduino có hệ sinh thái mạnh và cộng đồng rộng lớn:

Có hàng triệu người dùng trên toàn thế giới, giúp chia sẻ kiến thức, thư viện, ví dụ và giải pháp cho hầu hết các vấn đề thường gặp.

* Arduino có chi phí thấp:

Các bo mạch Arduino chính hãng cũng như bản tương thích có giá thành rẻ, phù hợp với sinh viên, người học hoặc các dự án nghiên cứu nhỏ.

* Arduino hỗ trợ đa nền tảng:

Arduino IDE có thể cài đặt và chạy trên nhiều hệ điều hành như Windows, macOS, Linux. Ngoài ra còn có phiên bản Arduino IDE 2.0 hiện đại hơn và Arduino Web Editor trên trình duyệt.

* **Ứng dụng trong đề tài:**

Arduino IDE được dùng để viết chương trình cho ESP32-CAM (phát hiện hình ảnh, gửi tín hiệu) và ESP32 (xử lý và điều khiển LED theo vùng). Đồng thời cũng dùng để thiết lập truyền UART giữa hai vi điều khiển này.



*Hình 1.2: Phần mềm Arduino*

**Edge Impulse – Nền tảng huấn luyện AI cho thiết bị nhúng:**

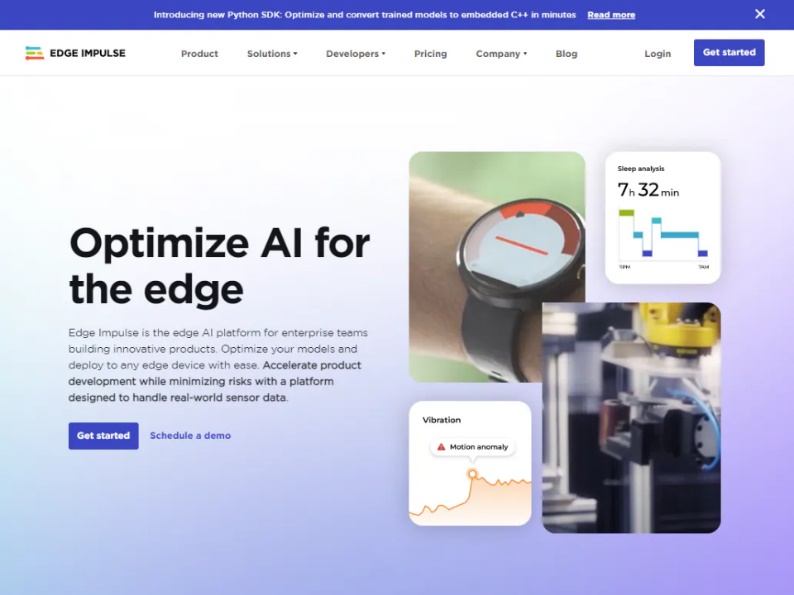
Edge Impulse là nền tảng chuyên dùng để phát triển các mô hình trí tuệ nhân tạo (AI) cho các thiết bị nhúng và IoT. Nền tảng này cho phép người dùng thu thập dữ liệu, huấn luyện mô hình và triển khai AI trực tiếp lên các thiết bị như ESP32-CAM.

* **Các tính năng nổi bật:**
* Có thể huấn luyện nhận diện hình ảnh, âm thanh, chuyển động,... trực tuyến.
* Hỗ trợ xuất mô hình dưới dạng thư viện tích hợp với Arduino, dễ dàng triển khai.
* Tương thích tốt với các dòng camera nhỏ như ESP32-CAM, giúp phát hiện vật thể,...
* **Lịch sử phát triển:**

Edge Impulse được thành lập vào năm 2019 bởi Zach Shelby và Jan Jongboom – hai chuyên gia có nhiều kinh nghiệm trong lĩnh vực IoT và phần mềm nhúng. Với mục tiêu "đưa AI đến biên (Edge AI)", nền tảng nhanh chóng phát triển và hiện được hỗ trợ bởi nhiều hãng phần cứng lớn như STMicroelectronics, Arduino, Renesas, Nordic Semiconductor…

* **Đặc điểm của phần mềm Edge Impulse:**
* Phát triển các ứng dụng AI trên thiết bị nhúng.
* Huấn luyện mô hình nhận dạng âm thanh, hình ảnh, chuyển động...
* Triển khai mô hình ML vào vi điều khiển (MCU), vi xử lý (MPU), hoặc thiết bị IoT.
* Phát triển nhanh nguyên mẫu (rapid prototyping) với khả năng thu thập và xử lý dữ liệu ngay trên nền tảng web.
* **Ứng dụng trong đề tài:**

Edge Impulse được sử dụng để huấn luyện mô hình nhận diện vật thể phía trước xe, như xe hoặc vật cản ngược chiều. Sau đó mô hình này được chạy trên ESP32-CAM để đưa ra tín hiệu điều chỉnh vùng sáng tương ứng trên đèn pha mô phỏng.



*Hình 1.3: Phần mềm Edge Impulse*

1. **Giới thiệu về hệ thống đèn pha ma trận trên xe Audi**

**Tổng quan công nghệ:**

Hệ thống đèn pha ma trận (Matrix LED Headlights) là một trong những công nghệ chiếu sáng tiên tiến được Audi phát triển và ứng dụng trên các dòng xe cao cấp như Audi A8, Q7, Q8, và e-tron. Ra mắt lần đầu vào năm 2013 trên mẫu Audi A8, công nghệ này đã đánh dấu bước đột phá trong lĩnh vực chiếu sáng ô tô, mang lại khả năng chiếu sáng thông minh và linh hoạt, đồng thời nâng cao tính an toàn và thẩm mỹ cho phương tiện.

**Nguyên lý hoạt động:**

Khác với các hệ thống đèn pha truyền thống sử dụng một nguồn sáng cố định, đèn pha ma trận của Audi bao gồm nhiều bóng LED nhỏ được sắp xếp thành từng vùng chiếu sáng riêng biệt. Mỗi bóng LED có thể được điều khiển độc lập thông qua phần mềm và cảm biến, cho phép hệ thống tự động điều chỉnh mẫu chiếu sáng theo thời gian thực dựa trên các yếu tố như:

* Tốc độ xe
* Hướng lái
* Sự hiện diện của các phương tiện khác
* Dữ liệu từ hệ thống định vị GPS

Ví dụ, khi phát hiện có xe ngược chiều hoặc xe phía trước, hệ thống sẽ tự động làm mờ hoặc tắt các vùng đèn LED tương ứng để tránh gây chói mắt cho người điều khiển phương tiện khác, trong khi vẫn duy trì ánh sáng tối ưu cho các khu vực còn lại trên đường. Ngoài ra, hệ thống còn có khả năng điều chỉnh ánh sáng khi vào cua, dựa trên dữ liệu từ hệ thống định vị, giúp chiếu sáng khu vực trước khi người lái thực hiện thao tác đánh lái.

**Các tính năng nổi bật:**

* Chiếu sáng thông minh: Hệ thống sử dụng camera và cảm biến để nhận diện các phương tiện khác và điều chỉnh ánh sáng phù hợp, đảm bảo tầm nhìn tối ưu cho người lái mà không gây chói mắt cho người tham gia giao thông khác.
* Đèn tín hiệu động (Dynamic Turn Signals): Đèn xi-nhan sử dụng hiệu ứng ánh sáng động, tạo ra chuỗi ánh sáng di chuyển theo hướng rẽ, tăng khả năng nhận diện và thẩm mỹ cho xe.
* Đèn chiếu sáng khi vào cua (Cornering Light): Hệ thống tự động điều chỉnh hướng chiếu sáng khi xe vào cua, giúp người lái quan sát rõ ràng hơn trong các khúc cua.
* Đèn chào mừng (Coming/Leaving Home Animation): Khi khóa hoặc mở khóa xe, hệ thống đèn pha thực hiện các hiệu ứng ánh sáng đặc trưng, tạo nên trải nghiệm tương tác độc đáo cho người sử dụng.

**Lợi ích và ứng dụng:**

Việc áp dụng công nghệ đèn pha ma trận mang lại nhiều lợi ích thiết thực:

* Tăng cường an toàn: Cải thiện tầm nhìn ban đêm và trong điều kiện thời tiết xấu, giảm nguy cơ tai nạn.
* Tiết kiệm năng lượng: Sử dụng công nghệ LED giúp giảm tiêu thụ năng lượng so với các hệ thống chiếu sáng truyền thống.
* Tăng tính thẩm mỹ và giá trị thương hiệu: Các hiệu ứng ánh sáng độc đáo góp phần tạo nên dấu ấn riêng cho thương hiệu Audi.



*Hình 1.4: Đèn LED ma trận xe Audi*

**Kết luận:**

Hệ thống đèn pha ma trận của Audi là minh chứng cho sự kết hợp giữa công nghệ tiên tiến và thiết kế tinh tế, mang lại trải nghiệm lái xe an toàn và đẳng cấp. Việc nghiên cứu và mô phỏng nguyên lý hoạt động của hệ thống này không chỉ giúp hiểu rõ hơn về công nghệ chiếu sáng hiện đại mà còn mở ra cơ hội ứng dụng trong các dự án phát triển hệ thống chiếu sáng thông minh cho các phương tiện giao thông trong tương lai.

# CHƯƠNG 2. CƠ SỞ LÝ THUYẾT

## Tổng quan về ứng dụng UART trong kết nối ESP32-CAM và ESP32 cho hệ thống đèn pha ma trận.

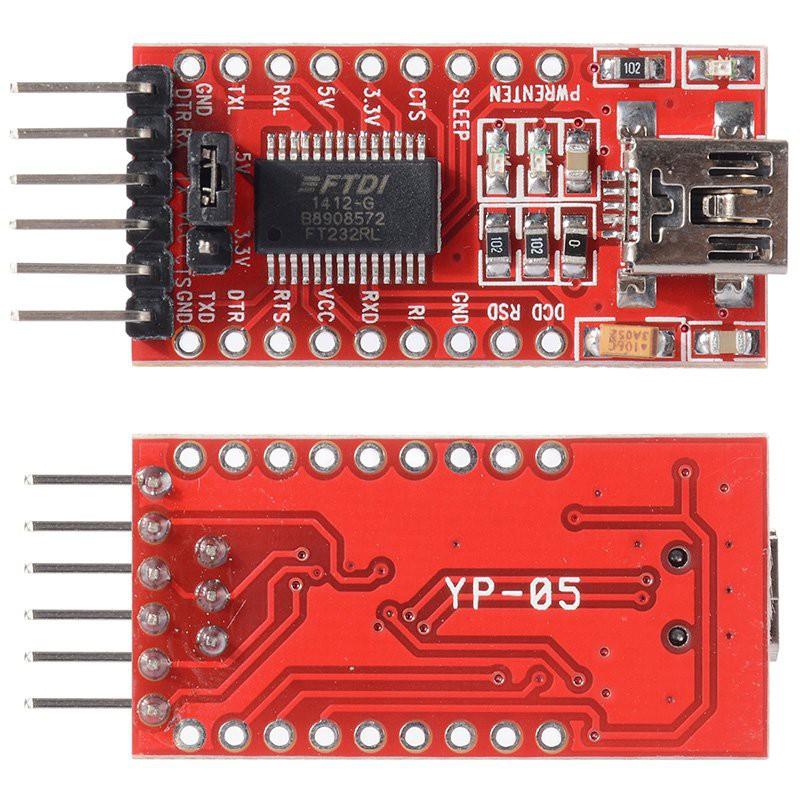
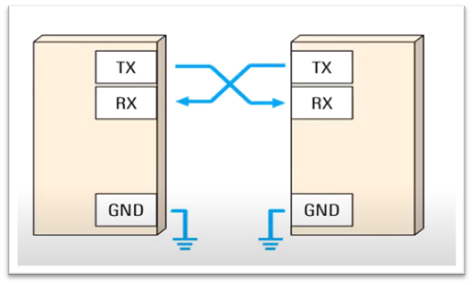
Trong đề tài này, việc xây dựng hệ thống mô phỏng đèn pha ma trận yêu cầu hiểu rõ một số nền tảng lý thuyết cơ bản liên quan đến truyền thông UART, vi điều khiển ESP32/ESP32-CAM, cũng như cách điều khiển LED ma trận theo vùng sáng.



*Hình 2.1: Mô phỏng đèn LED ma trận*

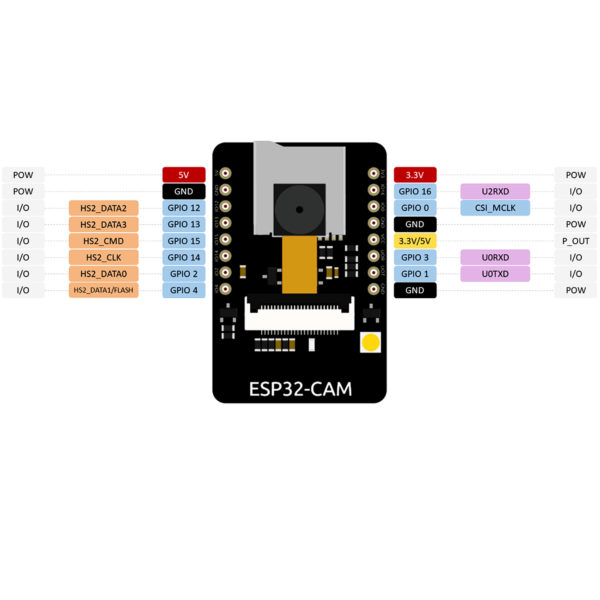
1. **Linh kiện và** **nguyên lý hoạt động của ứng dụng UART trong kết nối ESP32-CAM và ESP32 cho hệ thống đèn pha ma trận.**

* **Giao tiếp UART (Universal Asynchronous Receiver/Transmitter):**
* UART (Universal Asynchronous Receiver-Transmitter – Bộ truyền nhận dữ liệu không đồng bộ) là một giao thức truyền thông phần cứng dùng giao tiếp nối tiếp không đồng bộ và có thể cấu hình được tốc độ.
* Giao thức UART là một giao thức đơn giản và phổ biến, bao gồm hai đường truyền dữ liệu độc lập là TX (truyền) và RX (nhận). Dữ liệu được truyền và nhận qua các đường truyền này dưới dạng các khung dữ liệu (data frame) có cấu trúc chuẩn, với một bit bắt đầu (start bit), một số bit dữ liệu (data bits), một bit kiểm tra chẵn lẻ (parity bit) và một hoặc nhiều bit dừng (stop bit).
* Thông thường, tốc độ truyền của UART được đặt ở một số chuẩn, chẳng hạn như 9600, 19200, 38400, 57600, 115200 baud và các tốc độ khác. Tốc độ truyền này định nghĩa số lượng bit được truyền qua mỗi giây. Các tốc độ truyền khác nhau thường được sử dụng tùy thuộc vào ứng dụng và hệ thống sử dụng. Giao tiếp UART thường chỉ cần 2 dây:
* TX (Transmit) – Gửi dữ liệu
* RX (Receive) – Nhận dữ liệu
* Nguyên lý hoạt động:
* Uart truyền dữ liệu nối tiếp, theo 1 trong 3 chế độ:
* Simplex: Chỉ tiến hành giao tiếp một chiều
* Half duplex: Dữ liệu sẽ đi theo một hướng tại 1 thời điểm
* Full duplex: Thực hiện giao tiếp đồng thời đến và đi từ mỗi master và slave
* Chân Tx (truyền) của một chip sẽ kết nối trực tiếp với chân Rx (nhận) của chip khác và ngược lại. Quá trình truyền dữ liệu thường sẽ diễn ra ở 3.3V hoặc 5V. Uart là một giao thức giao tiếp giữa một master và một slave. Trong đó 1 thiết bị được thiết lập để tiến hành giao tiếp với chỉ duy nhất 1 thiết bị khác.
* Dữ liệu truyền đến và đi từ Uart song song với thiết bị điều khiển. Khi tín hiệu gửi trên chân Tx (truyền), bộ giao tiếp Uart đầu tiên sẽ dịch thông tin song song này thành dạng nối tiếp và sau đó truyền tới thiết bị nhận. Chân Rx (nhận) của Uart thứ 2 sẽ biến đổi nó trở lại thành dạng song song để giao tiếp với các thiết bị điều khiển.
* Dữ liệu truyền qua Uart sẽ đóng thành các gói (packet). Mỗi gói dữ liệu chứa 1 bit bắt đầu, 5 – 9 bit dữ liệu (tùy thuộc vào bộ Uart), 1 bit chẵn lẻ tùy chọn và 1 bit hoặc 2 bit dừng.
* Quá trình truyền dữ liệu Uart sẽ diễn ra dưới dạng các gói dữ liệu này, bắt đầu bằng 1 bit bắt đầu, đường mức cao được kéo dần xuống thấp. Sau bit bắt đầu là 5 – 9 bit dữ liệu truyền trong khung dữ liệu của gói, theo sau là bit chẵn lẻ tùy chọn để nhằm xác minh việc truyền dữ liệu thích hợp. Sau cùng, 1 hoặc nhiều bit dừng sẽ được truyền ở nơi đường đặt tại mức cao. Vậy là sẽ kết thúc việc truyền đi một gói dữ liệu.
* Ưu điểm và nhược điểm:
* Ưu điểm:
* Chỉ cần dùng 2 dây truyền dữ liệu
* Không cần đến tín hiệu clock
* Có 2 bit chẵn lẻ nên có thể kiểm tra lỗi dễ dàng
* Cấu trúc gói dữ liệu có thể thay đổi được miễn là cả 2 bên đều được thiết lập để giao tiếp với nhau
* Phương pháp giao tiếp Uart có nhiều tài liệu hướng dẫn và cũng là bộ truyền dữ liệu đang được sử dụng rộng rãi hiện nay
* Nhược điểm:
* Kích thước của khung dữ liệu giới hạn tối đa là 9 bit, khá nhỏ so với nhu cầu sử dụng
* Không được hỗ trợ nhiều hệ thống master và slave
* Tốc độ truyền của mỗi giao tiếp Uart phải nằm trong khoảng 10% của nhau
* Ứng dụng của giao tiếp UART:
* Kết nối thiết bị điện tử với máy tính: chẳng hạn như vi điều khiển, cảm biến, màn hình hiển thị, máy in và các thiết bị khác. Khi được kết nối với máy tính, các thiết bị này có thể gửi và nhận dữ liệu thông qua cổng giao tiếp UART.
* Giao tiếp giữa các vi điều khiển: Các vi điều khiển này có thể truyền và nhận dữ liệu để thực hiện các tác vụ như điều khiển động cơ, đo lường và kiểm soát các thông số, và điều khiển các thiết bị khác.
* Giao tiếp không dây: Giao thức UART có thể được sử dụng để giao tiếp không dây giữa các thiết bị, chẳng hạn như các thiết bị Bluetooth hoặc Wi-Fi. Trong trường hợp này, các dữ liệu được truyền qua sóng radio hoặc sóng vô tuyến.
* Điều khiển robot: Các thiết bị như bộ điều khiển và các mô-đun điều khiển robot có thể sử dụng giao thức UART để gửi và nhận dữ liệu từ nhau.
* Hệ thống đo lường: Giao thức UART có thể được sử dụng để kết nối các thiết bị đo lường với các thiết bị khác, chẳng hạn như máy tính hoặc các thiết bị nhúng. Các thiết bị đo lường có thể gửi dữ liệu về các thông số đo được thông qua cổng UART, và thiết bị nhận có thể hiển thị hoặc xử lý dữ liệu này.
* **Mạch chuyển USB UART TTL FT232RL:**
* Mạch chuyển USB UART TTL FT232RL sử dụng IC FT232RL từ chính hãng FTDI, mạch được thiết kể nhỏ gọn nhưng vẫn ra chân đầy đủ, rất dễ sử dụng với mọi hệ điều hành Windows, Mac, Linux
* Chip có sẵn ổn áp và dao động tích hợp bên trong, hoạt động rất ổn định so với các dòng chip USB to serial khác
* Mạch có thể hoạt động ở 2 chế độ 5v hoặc 3v3, bằng cách thiết lập trên jumper trên mạch
* Chân cắm ra gồm 2 loại theo chuẩn FTDI (phù hợp với Arduino) và chuẩn UART thường, được ký hiệu rõ ràng trên mạch. Đầu vào sử dụng loại USB B mini.
* Ngoài ra, trên mạch có sẵn 2 led cho tín hiệu TX và RX, giúp theo dõi trực tiếp trạng thái tín hiệu.
* USB to Serial – FTDI có kích thước nhỏ gọn dễ dàng tích hợp vào ứng dụng:
* Làm mạch nạp cho các bản Arduino không tích hợp mạch nạp onboard như: ChipiPRO-LITE, Arduino Pro. Lilypad…
* Làm trung gian giao tiếp bo mạch với máy tính. rất hữu ích khi cần truyền dữ liệu từ bo mạch lên máy tính để kiểm tra, phân tích.
* Làm mạch nạp cho một số dòng vi điều khiển ARM, AVR, 89, PIC,… có hỗ trợ nạp bằng UART.
* Thông số kỹ thuật:
* IC chính: FT232RL chính hãng FTDI
* Nguồn cấp: 5VDC từ cổng USB (cổng mini USB hoặc USB Type-C)
* Có ngõ ra nguồn có thể điều chỉnh 3V3 hoặc 5VDC
* Chuyển giao tiếp từ USB sang UART TTL
* Drive hỗ trợ Windows Mac, Linux
* Có cầu chì tự phục hồi: 500mA
* Tốc độ Baudrate: tùy chỉnh
* Kích thước PCB: 36 x 18.5mm
* Trọng lượng: 3g
* Sơ đồ chân ra của mạch chuyển USB UART TTL FT232RL
* DTR: Data terminal ready control output / Handshake signal. (có thể reset arduino khi nạp chương trình)
* RXD: Receive asynchronous data Input – nhận tín hiệu
* TXD: Transmit asynchronous data output – truyền tín hiệu
* VCC: Chân nguồn cấp, có thể chọn 5V hoặc 3.3VDC thông qua Jumper
* CTS: Clear to send control input / handshake signal (không sử dụng)
* Trong đề tài này, ESP32-CAM sử dụng giao tiếp UART để truyền thông tin nhận diện hình ảnh (ví dụ: có xe ngược chiều) đến ESP32, từ đó điều khiển từng vùng LED ma trận tương ứng để mô phỏng vùng bị giảm sáng.



*Hình 2.2: Giao tiếp UART và mạch chuyển USB UART TTL FT232RL*

* **ESP32-CAM – Vi điều khiển tích hợp camera:**
* ESP32-CAM là một module vi điều khiển tích hợp sẵn camera OV2640, có khả năng xử lý ảnh cơ bản và hỗ trợ truyền dữ liệu qua UART, WiFi hoặc Bluetooth. Dù có cấu hình hạn chế so với máy tính, ESP32-CAM vẫn có thể nhận diện hình ảnh ở mức đơn giản nhờ vào việc kết hợp với các nền tảng AI như Edge Impulse.
* Mạch thu phát Wifi BLE ESP32 này là mạch chính hãng AI – Thinker có chất lượng độ ổn định và độ bền rất cao, sử dụng camera OV2640 chất lượng cao hình ảnh sắc nét, không nhiễu sọc, không xảy ra tình trạng treo khi hoạt động do sử dụng ic cấp nguồn chất lượng cao. Mạch thu phát Wifi BLE ESP32 này là mạch chính hãng AI – Thinker có chất lượng độ ổn định và độ bền rất cao, sử dụng camera OV2640 chất lượng cao hình ảnh sắc nét, không nhiễu sọc, không xảy ra tình trạng treo khi hoạt động do sử dụng ic cấp nguồn chất lượng cao.
* Mạch thu phát Wifi BLE ESP32-CAM Ai-Thinker này có thể sử dụng Arduino IDE để biên dịch và viết code, được hỗ trợ mạnh mẽ từ cộng đồng.
* Trong đề tài, ESP32-CAM được sử dụng để nhận diện hình ảnh phía trước xe, phát hiện xe hoặc vật cản đối diện, và truyền tín hiệu điều khiển về cho ESP32 qua UART.
* Thông số kỹ thuật:
* Module: ESP32-CAM
* Kiểu: Dip-16
* Kích thước: 27 x 40.5 x 4.5 mm
* SPI Flash: Mặc định 32Mbit
* RAM: Internal520KB + PSRAM 4M bên ngoài
* Bluetooth: tiêu chuẩn Bluetooth4.2BR / EDR và BLE
* Wifi: 802.11 b/g/n/e/i
* Giao diện hỗ trợ: UART, SPI, I2C, PWM
* Hỗ trợ thẻ TF: Hỗ trợ tối đa 4G
* Cổng IO: 9
* Tốc độ cổng giao tiếp: 115200 bps mặc định
* Định dạng đầu ra hình ảnh: JPEG (chỉ được hỗ trợ bởi OV2640), BMP
* Phạm vi phổ: 2412 ~ 2484 MHz
* Dạng ăng ten: ăng-ten PCB trên bo mạch, đạt được 2dBi
* Truyền tín hiệu:
* 802.11b: 17 ± 2 dBm (@ 11Mbps)
* 802.11g: 14 ± 2 dBm (@ 54Mbps)
* 802.11n: 13 ± 2 dBm (@ MCS7)
* Độ nhạy:
* CCK, 1 Mb/giây: -90dBm
* CCK, 11 Mb/giây: -85dBm
* 6 Mb/giây (1/2 BPSK): -88dBm
* 54 Mbps (3/4 64-QAM): -70dBm
* MCS7 (65 Mb/giây, 72,2 Mb/giây): -67dBm
* Sự tiêu thụ năng lượng:
* Tắt đèn flash: 180mA @ 5V
* Bật đèn flash và điều chỉnh độ sáng tối đa: 310mA @ 5V
* Sleep Deep: Mức tiêu thụ điện năng thấp nhất có thể đạt 6mA @ 5V
* Modem-bed: lên tới 20mA @ 5V
* Sleep Light: lên tới 6,7mA@5V
* Bảo vệ: WPA / WPA2 / WPA2-Enterprise / WPS
* Phạm vi cung cấp điện: 5V
* Nhiệt độ hoạt động: -20 °C ~ 85 °C
* Môi trường lưu trữ: -40 °C ~ 90 °C, <90% rh



*Hình 2.3: ESP32-CAM và sơ đồ chân*

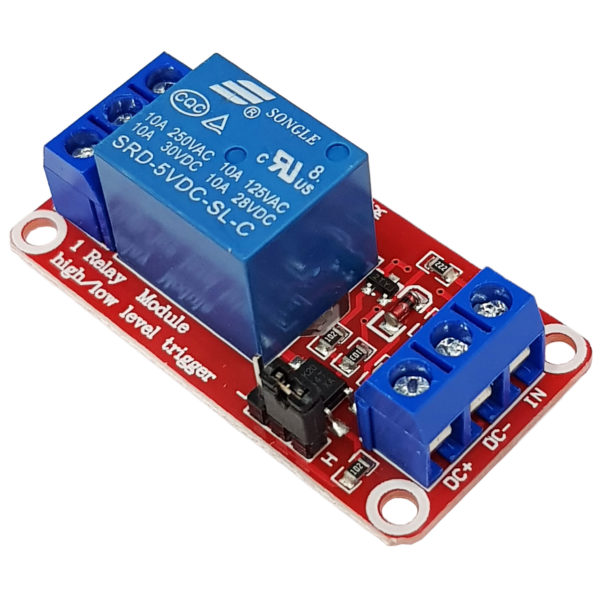
* **ESP32 – Vi điều khiển điều khiển LED theo vùng:**
* ESP32 là một vi điều khiển mạnh với lõi kép, tích hợp WiFi, Bluetooth và nhiều cổng giao tiếp. Trong hệ thống này, ESP32 đóng vai trò là bộ xử lý trung tâm, nhận dữ liệu từ ESP32-CAM, phân tích và điều khiển LED ma trận mô phỏng từng vùng đèn chiếu sáng.
* Điểm tốt về ESP32, giống như ESP8266 là các thành phần RF tích hợp của nó như bộ khuếch đại công suất, bộ khuếch đại nhận tiếng ồn thấp, công tắc ăng-ten, bộ lọc và Balun RF. Điều này làm cho việc thiết kế phần cứng xung quanh ESP32 rất dễ dàng vì bạn cần rất ít thành phần bên ngoài.
* ESP32 là nó được sản xuất bằng công nghệ 40 nm công suất cực thấp của TSMC. Vì vậy, việc thiết kế các ứng dụng hoạt động bằng pin như thiết bị đeo, thiết bị âm thanh, đồng hồ thông minh, ...
* Việc chia vùng LED (mỗi vùng đại diện cho một khu vực chiếu sáng nhất định) giúp mô phỏng được tính năng tự điều chỉnh vùng sáng như trên các hệ thống đèn pha ma trận thực tế của xe Audi.
* Thông số kỹ thuật:
* Loại: Wifi + Bluetooth Module
* Cổng nạp: Type C || Micro (tùy chọn trong phần phân loại)
* Mô hình: ESP32 38 chân
* Điện áp nguồn (USB): 5V DC
* Đầu vào/Đầu ra điện áp: 3.3V DC
* Công suất tiêu thụ: 5μA trong hệ thống treo chế độ
* Hiệu suất: Lên đến 600 DMIPS
* Tần số: lên đến 240MHz
* Wifi: 802.11 B/g/n/E/I (802.11N @ 2.4 GHz lên đến 150 Mbit/S)
* Bluetooth: 4.2 BR/EDR BLE 2 chế độ điều khiển
* Bộ nhớ: 448 Kbyte ROM, 520 Kbyte SRAM, 6 Kbyte SRAM trên RTC và QSPI Hỗ trợ đèn flash / SRAM chip
* Chip USB-Serial: CP2102
* Ăng ten: PCB
* GPIO kỹ thuật số: 24 chân (một số chân chỉ làm đầu vào)
* Kỹ thuật số Analog: 12bit SAR loại ADC, hỗ trợ các phép đo trên lên đến 18 kênh, một số chân hỗ trợ một bộ khuếch đại với lập trình tăng
* Bảo mật: IEEE 802.11, bao gồm cả WFA, WPA/WPA2 và WAPI
* Phần cứng tăng tốc mật mã học: AES, SHA-2, RSA, hình elip mật mã Đường Cong (ECC), số ngẫu nhiên Máy phát điện (RNG)
* Cân nặng: ~ 11g

A circuit board with many colors

AI-generated content may be incorrect.

*Hình 2.4: ESP32 và sơ đồ chân*

* **Module Relay:**
* Module 1 Relay với opto cách ly nhỏ gọn, có opto và transistor cách ly giúp cho việc sử dụng trở nên an toàn với board mạch chính, module 1 Relay với opto cách ly H/L 5v được sử dụng để đóng ngắt nguồn điện công suất cao AC hoặc DC, có thể chọn đóng khi kích mức cao hoặc mức thấp bằng Jumper.
* Tiếp điểm đóng ngắt gồm 3 tiếp điểm NC (thường đóng), NO(thường mở) và COM(chân chung) được cách ly hoàn toàn với board mạch chính, ở trạng thái bình thường chưa kích NC sẽ nối với COM, khi có trạng thái kích COM sẽ chuyển sang nối với NO và mất kết nối với NC.
* Relay là linh kiện điện tử đóng vai trò như một công tắc điều khiển bằng tín hiệu điện. Trong hệ thống này, module 4 relay được sử dụng để điều khiển việc bật/tắt các vùng LED mô phỏng đèn pha ma trận.
* Thông số kỹ thuật:
* Điện áp: 5VDC.
* Dòng tiêu thụ: ~200mA/1Relay
* Tín hiệu kích: High (5VDC) hoặc Low (0VDC) chọn bằng Jumper.
* Relay trên mạch:
* Nguồn nuôi: 5VDC.
* Tiếp điểm đóng ngắt max: 250VAC-10A hoặc 30VDC-10A
* Kích thước: 72 (L) \* 55 (W) \* 19 (H) mm.
* Nguyên lý hoạt động:
* Mỗi relay trong module có một chân IN điều khiển nối với các chân digital của ESP32.
* Khi ESP32 gửi tín hiệu mức thấp (LOW) xuống chân điều khiển, relay sẽ đóng mạch, cho phép dòng điện đi qua LED tương ứng và làm LED sáng.
* Ngược lại, khi không có tín hiệu hoặc ở mức cao (HIGH), relay sẽ ngắt mạch.
* Ưu điểm:
* Cách ly tốt giữa mạch điều khiển (ESP32) và tải công suất (LED).
* Dễ dàng mở rộng thêm các thiết bị có điện áp cao hơn, nếu cần thiết sau này.



*Hình 2.5: Module Relay 1 kênh 5V H/L*

* **Nguồn cấp cho hệ thống:**
* Trong hệ thống, có hai loại thiết bị tiêu thụ điện: vi điều khiển (ESP32, ESP32-CAM) và thiết bị tải (relay, LED). Để hệ thống hoạt động ổn định, cần có nguồn điện ngoài 5V DC cấp riêng cho relay và LED.
* Thông số kỹ thuật:
* Đầu ra:
* Điện áp đầu ra: +5.0V DC
* Dòng điện định danh (có tải): 4A
* Công suất tối đa: 20W
* Bảo vệ:
* Bảo vệ ngắn mạch
* Bảo vệ quá dòng
* Hiệu suất: 82%
* Cáp đầu ra: 1500±50mm UL2464 20AWG
* Kết nối đầu ra: DC Jack 2.1mm
* Đầu vào:
* Dải điện áp: 100-240VAC (rated)
* Tần số: 50/60Hz
* Dòng điện: 0.8A maximum
* Điện năng tiêu thụ (không tải): 0.1W maximum



*Hình 2.6: Nguồn Adapter 5V*

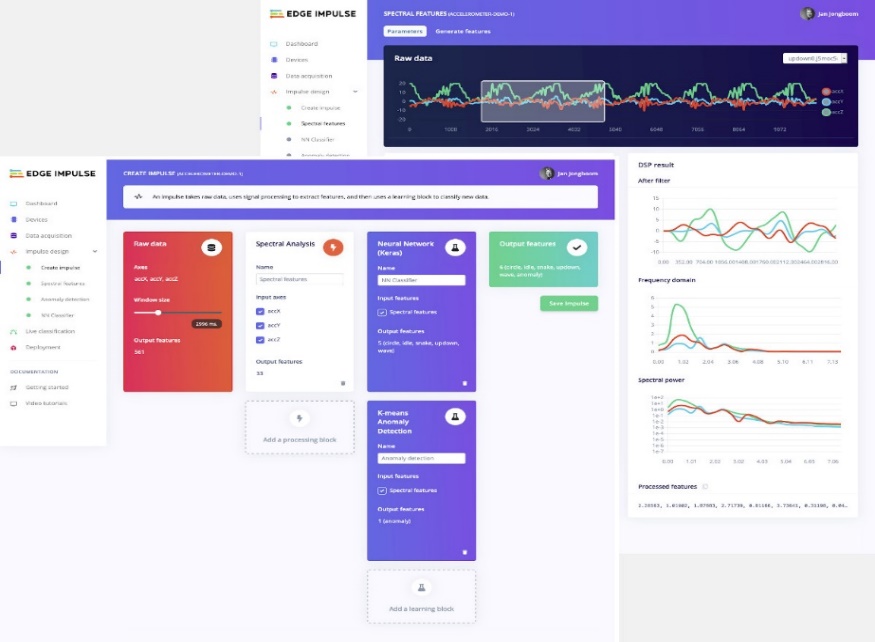
## Huấn luyện mô hình AI bằng Edge Impulse:

Để ESP32-CAM có thể tự động nhận diện xe đi đối diện hoặc vật cản phía trước, hệ thống cần tích hợp một mô hình AI được huấn luyện từ nền tảng Edge Impulse. Đây là bước quan trọng để giúp đèn pha ma trận có thể phản ứng thông minh theo từng tình huống thực tế.

* Quy trình huấn luyện hình ảnh:
* Thu thập dữ liệu:
* Sử dụng camera ESP32-CAM hoặc máy ảnh để chụp ảnh trong các tình huống có: xe ngược chiều hoặc không có vật cản.
* Mỗi ảnh được gán nhãn đúng theo tình huống.
* Tải ảnh lên Edge Impulse:
* Dữ liệu hình ảnh được tải lên qua trang web Edge Impulse hoặc thông qua dòng lệnh (CLI).
* Ảnh được chia thành tập training và testing.
* Xây dựng pipeline xử lý:
* Thêm khối Image Processing (resize, grayscale...) và khối Learning Block (ví dụ: Classification).
* Tiến hành huấn luyện mô hình bằng phương pháp học sâu (Deep Learning) phù hợp với vi điều khiển.
* Đánh giá độ chính xác mô hình:
* Hệ thống cung cấp biểu đồ confusion matrix và các chỉ số accuracy, loss.
* Nếu mô hình chưa tốt, tiếp tục bổ sung thêm dữ liệu hoặc chỉnh lại thông số.
* Triển khai mô hình lên ESP32-CAM:
* Tải mô hình xuống dưới dạng thư viện Arduino (.zip).
* Nhúng thư viện vào chương trình Arduino, ESP32-CAM có thể nhận ảnh trực tiếp và phân loại tình huống ngay trên bo mạch.
* Kết quả và ứng dụng trong hệ thống:

Sau khi được tích hợp mô hình AI, ESP32-CAM có thể:

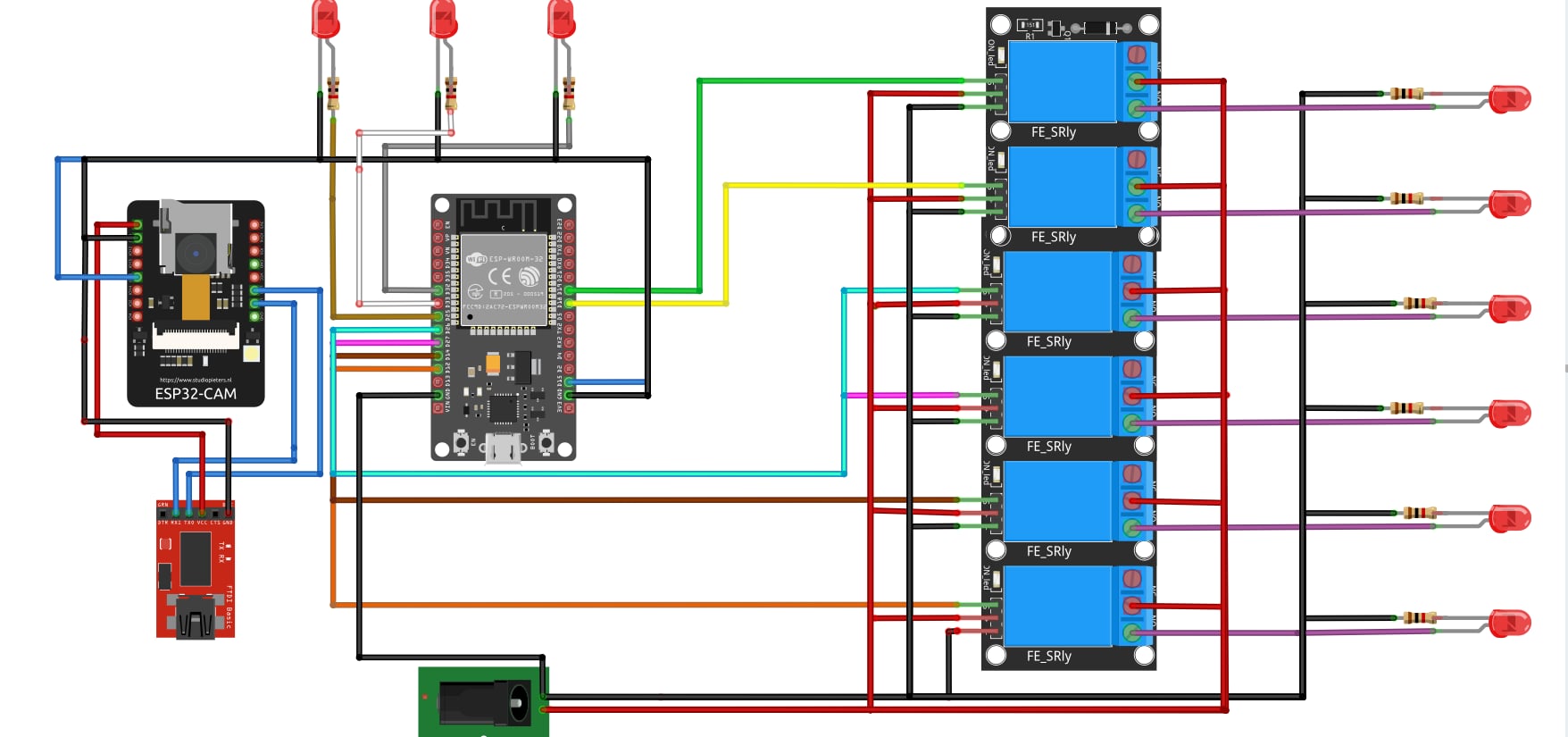
* Phân biệt được có hay không có xe đối diện.
* Xác định vị trí đối tượng (trái, phải, giữa).
* Gửi tín hiệu tương ứng sang ESP32 để xử lý vùng LED phù hợp.



*Hình 2.7: Mô phỏng huấn luyện mô hình AI bằng Edge Impulse*

# CHƯƠNG 3: THIẾT KẾ MẠCH NGUYÊN LÝ, THI CÔNG VÀ KẾT QUẢ

## Sơ đồ hoạt động của mạch ứng dụng UART trong kết nối ESP32-CAM và ESP32 cho hệ thống đèn pha ma trận .



*Hình 3.1: Sơ đồ nguyên lý của mạch*

* **Phân tích tổng thể:**

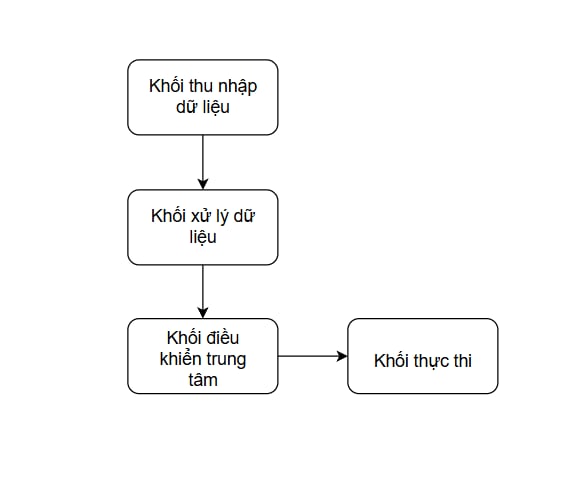
Hệ thống gồm các phần chính:

* ESP32-CAM: Đảm nhiệm việc thu nhận và xử lý hình ảnh từ camera.
* ESP32: Nhận dữ liệu từ ESP32-CAM qua UART, xử lý và điều khiển thiết bị ngoại vi.
* Module 4 Relay: Đóng vai trò điều khiển bật/tắt các vùng LED tương ứng.
* LED: Đại diện cho từng vùng đèn chiếu sáng (trái – giữa – phải – mở rộng...).
* Module chuyển USB to UART (FTDI): Dùng để nạp chương trình cho ESP32-CAM.
* Nguồn 5V ngoài: Cung cấp điện áp cho relay và các module.
* **Nguyên lý hoạt động của mạch:**

Hệ thống gồm hai vi điều khiển chính là ESP32-CAM và ESP32, phối hợp với nhau thông qua giảo tiếp UART, để điều khiển đèn LED ma trận thông minh mô phỏng theo đèn pha xe Audi.

* ESP32-CAM đảm nhiệm vai trò thu nhận hình ảnh phía trước bằng camera tích hợp, sau đó xử lý và phân loại hình ảnh bằng mô hình đã huấn luyện trên Edge Impulse. Dữ liệu kết quả được mã hóa thành ký tự đơn giản (như 'A', 'B',…) và gửi đến ESP32 qua giao tiếp UART.
* ESP32 nhận dữ liệu từ ESP32-CAM thông qua cổng UART (Serial2), sau đó phân tích nội dung và điều khiển các chân digital xuất ra module relay 4 kênh. Mỗi relay tương ứng với một vùng sáng trên LED ma trận.
* Các relay đóng vai trò bật/tắt từng vùng LED, giúp mô phỏng chức năng của đèn pha ma trận: khi có xe đối diện, hệ thống sẽ tắt một số vùng LED để không gây chói; khi đường trống, tất cả các vùng LED sẽ được bật sáng.
* Toàn bộ hệ thống relay và LED được cấp nguồn riêng 5V từ adapter ngoài, đảm bảo ổn định và tránh sụt áp cho vi điều khiển.

1. **Sơ đồ khối:**



*Hình 3.2: Sơ đồ khối*

* **Phân tích sơ đồ:**

**Khối thu thập dữ liệu:**

* Camera thu hình ảnh và thực hiện nhận diện vật thể qua camera
* Là nơi dữ liệu (ảnh/video) được thu thập ban đầu.

**Khối xử lý dữ liệu:**

* Chạy mô hình AI để phân loại xem ảnh chứa "car", "person" hay không.
* Xử lý bounding box và đưa ra quyết định gửi "oto", "vat\_can" hoặc "an\_toan".

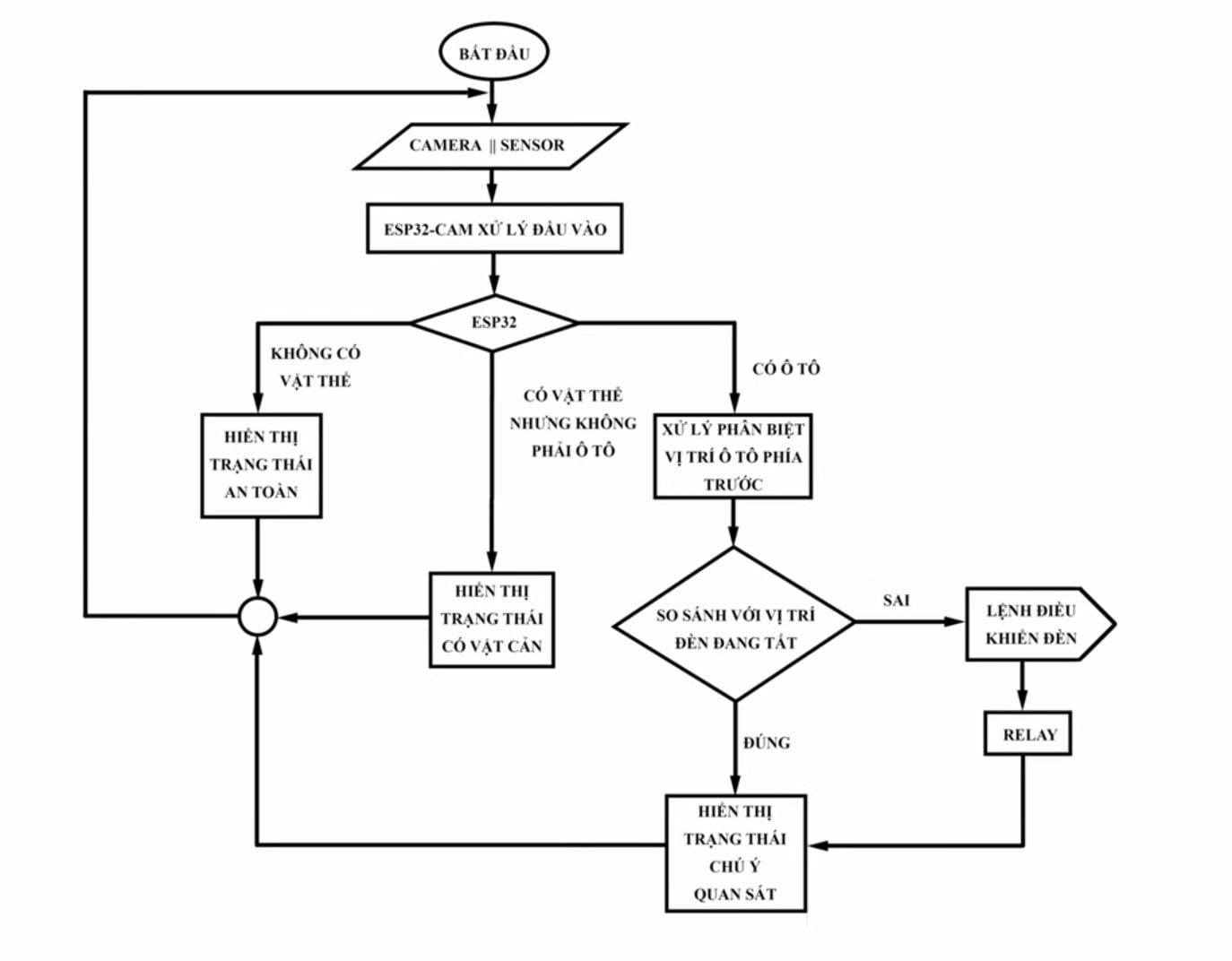
**Khối điều khiển trung tâm:**

* Nhận tín hiệu từ CAM qua UART.
* Xử lý logic: bật/tắt relay, LED cảnh báo...
* Là trung tâm điều phối hệ thống theo dữ liệu được xử lý.

**Khối thực thi:**

* Các thiết bị vật lý thực hiện hành động: mở/tắt relay, bật LED, v.v.

## Sơ đồ thuật toán:



*Hình 3.3: Sơ đồ thuật toán*

* **Phân tích sơ đồ**:
* Khởi động hệ thống:

Khi hệ thống cấp nguồn:

* Cả ESP32-CAM và ESP32-WROOM bắt đầu hoạt động.
* ESP32-CAM khởi động camera, chờ sẵn sàng để chụp ảnh.
* Đây là bước khởi tạo tất cả các module phần cứng: camera, UART, và các chân điều khiển relay/LED.
* Camera thu dữ liệu từ môi trường:

Camera trên ESP32-CAM liên tục thu nhận hình ảnh từ môi trường xung quanh (định kỳ 5–10 khung hình/giây).

* Mỗi ảnh thu được sẽ được đưa vào mô hình học sâu đã huấn luyện.
* Việc thu ảnh này đóng vai trò như mắt của hệ thống.
* Xử lý đầu vào với mô hình AI:

ESP32-CAM xử lý ảnh bằng mô hình FOMO (Faster Objects, More Objects) được huấn luyện bằng Edge Impulse:

* FOMO là một loại mạng nơ-ron tích chập (CNN), tối ưu cho thiết bị nhúng, có khả năng phát hiện nhiều vật thể cùng lúc.
* Mô hình được thiết lập để nhận diện 2 lớp: "car" và "person" hoặc "background".

Sau xử lý, kết quả trả về sẽ gồm:

* Tọa độ từng vật thể phát hiện
* Nhãn vật thể ("car", "person")
* Độ tin cậy (confidence)
* Gửi kết quả nhận diện qua UART đến ESP32-WROOM:

ESP32-CAM không tự điều khiển relay mà đóng vai trò thiết bị cảm biến thông minh. Kết quả xử lý được gửi qua UART:

* Nếu không phát hiện gì → gửi "an\_toan"
* Nếu phát hiện "person" hoặc vật thể khác → gửi "vat\_can"
* Nếu phát hiện "car" → gửi "OTO-x" (x là vùng A–F tương ứng vị trí xuất hiện), sau đó gửi "end" để xác nhận kết thúc một vòng phân tích
* ESP32-WROOM tiếp nhận và phân nhánh xử lý:

Sau khi nhận dữ liệu từ UART, ESP32-WROOM thực hiện các nhánh xử lý:

* Không có vật thể:
* Khi nhận "an\_toan", ESP32-WROOM:
* Tắt tất cả relay điều khiển đèn
* Bật LED xanh báo hiệu an toàn
* Trạng thái hệ thống là bình thường, không cần phản ứng thêm
* Có vật thể nhưng không phải ô tô:
* Khi nhận "vat\_can", hệ thống xác định là vật thể nhưng không đủ điều kiện để kích hoạt relay (không phải phương tiện giao thông).
* ESP32-WROOM bật LED cảnh báo màu vàng (hoặc đỏ) để người điều khiển biết có vật cản trong khu vực.
* Có ô tô:
* Khi nhận "OTO-x" (x = a–f), hệ thống hiểu rằng có một phương tiện đang xuất hiện tại vùng tương ứng.
* ESP32-WROOM sẽ lưu trữ thông tin các vùng có xe trong vòng đó.
* Khi nhận "end", hệ thống sẽ:
* So sánh trạng thái hiện tại của từng vùng (relay đã bật hay chưa)
* Bật relay tại vùng có xe (nếu chưa bật)
* Tắt relay tại vùng không còn xe (nếu trước đó đã bật)
* Bật LED cảnh báo màu đỏ hoặc trắng, tùy theo yêu cầu
* Cập nhật trạng thái cảnh báo:

Sau khi relay được bật hoặc tắt tương ứng, hệ thống sẽ:

* Hiển thị trạng thái “Chú ý quan sát” (nếu có ô tô đang hiện diện)
* Quay lại chu kỳ ban đầu và tiếp tục lặp lại

## Bố cục PCB và hiển thị 3D của mạch:

* **Bố cục PCB:**

**ESP32:**

**A blue circuit board with many small dots and lines

AI-generated content may be incorrect.**

*Hình 3.4: Bố cục PCB ESP32*

* **Hiển thị 3D:**

**ESP32-CAM:**

A green rectangular object with a black circle and a yellow button

AI-generated content may be incorrect.

*Hình 3.5: Hiển thị 3D mặt trước ESP32-CAM*

A green circuit board with black and white lines

AI-generated content may be incorrect.

*Hình 3.6: Hiển thị 3D mặt sau ESP32-CAM*

**ESP32:**

**A green circuit board with black and red components

AI-generated content may be incorrect.**

*Hình 3.7: Hiển thị 3D mặt trước ESP32*

A green circuit board with white dots and black arrows

AI-generated content may be incorrect.

*Hình 3.8: Hiển thị 3D mặt sau ESP32*

## Kết quả mạch.

* Mạch chạy ổn định.
* Vẫn còn nhiều hạn chế ở mạch.

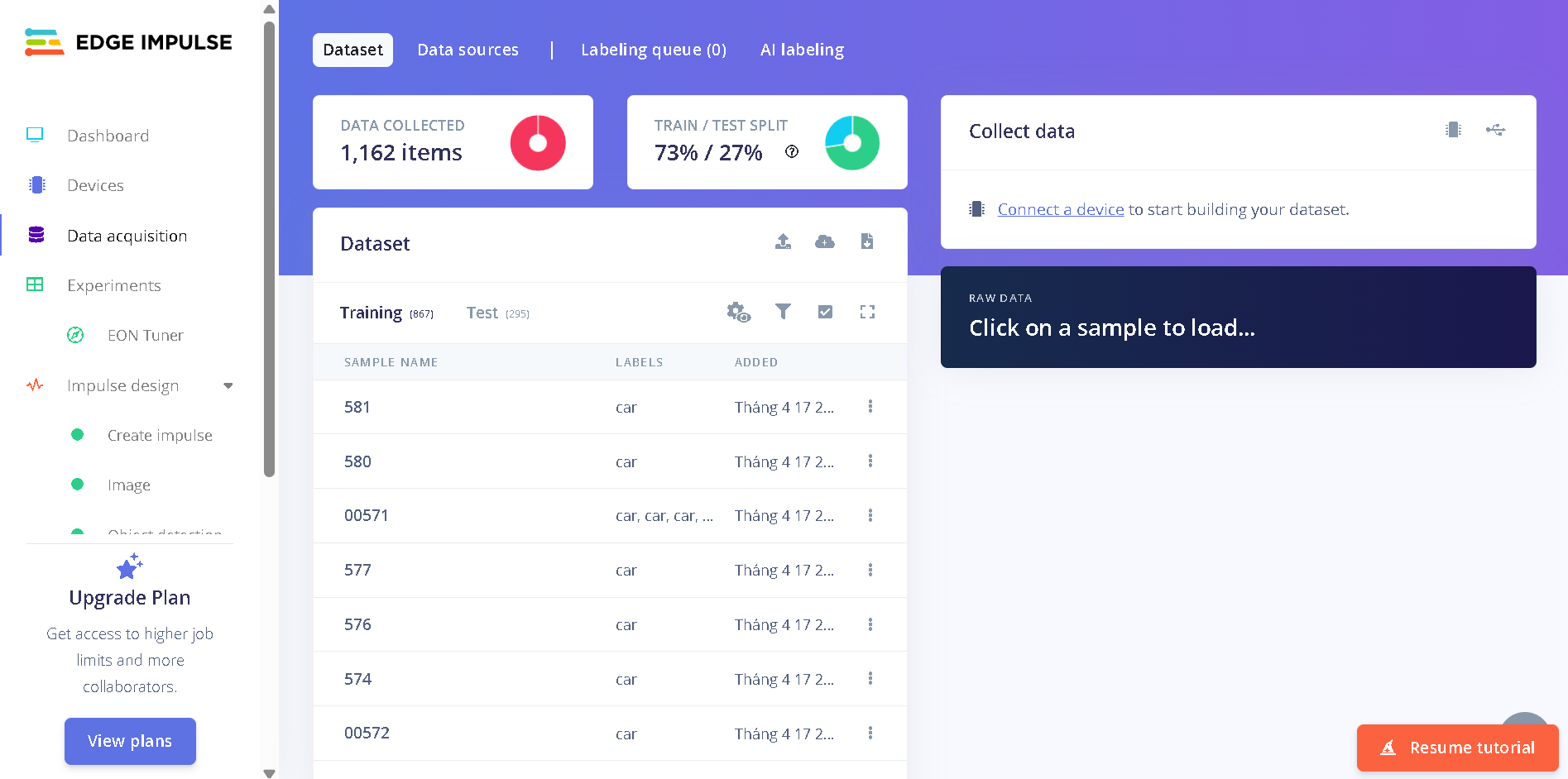
A group of wires connected to a circuit board

AI-generated content may be incorrect.

*Hình 3.9: Kết quả thực tế*

1. **Phần mềm của mạch:**

* **Phần mềm train Edge Impulse:**

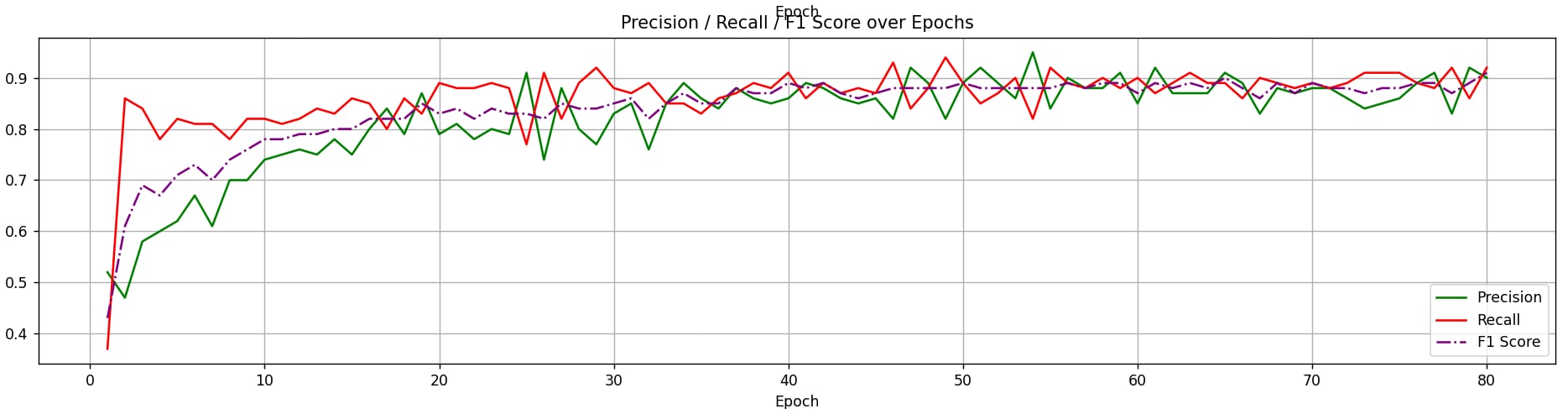
****

*Hình 3.10: Giao diện train Edge Impulse*

* **Nguyên lý hoạt động:**
* Thu thập dữ liệu ảnh: ESP32-CAM được kết nối với Edge Impulse Studio để thu ảnh từ môi trường thực tế, bao gồm hình ảnh có xe và không có xe ngược chiều. Dữ liệu này được gán nhãn (label) là *car*.
* Tiền xử lý (Preprocessing): Các ảnh được resize (thay đổi kích thước), chuyển sang định dạng grayscale (nếu cần), và chuyển đổi thành vector đầu vào phù hợp cho mô hình học máy.
* Huấn luyện mô hình học máy (Model training):

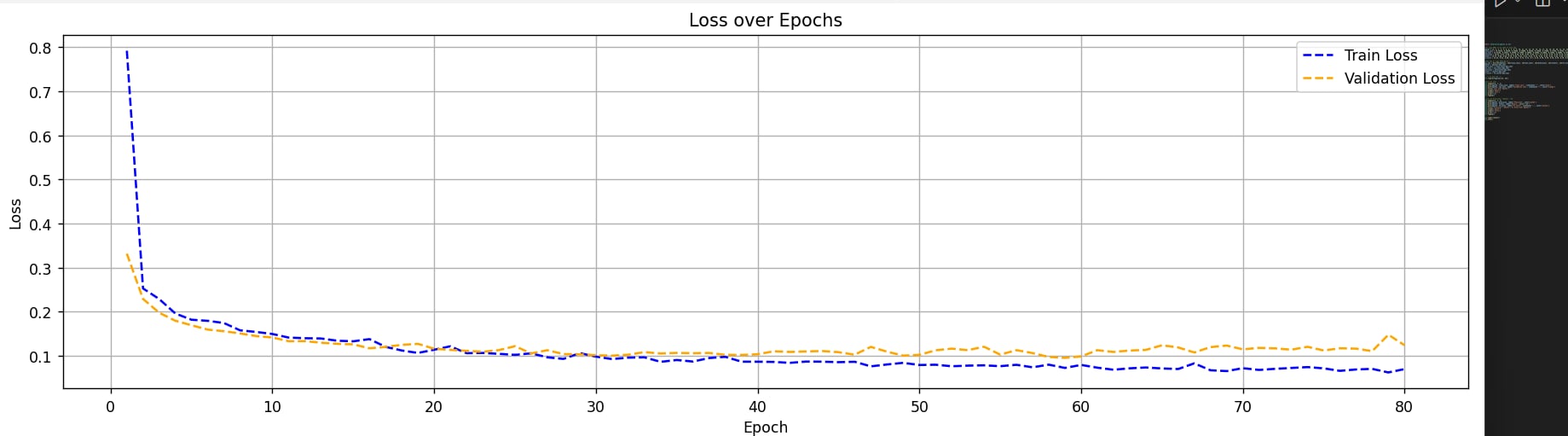
Dữ liệu được chia làm hai phần:

* Training (73%) để huấn luyện mô hình,
* Test (27%) để kiểm tra độ chính xác.
* Đánh giá mô hình (Model evaluation): Edge Impulse tự động kiểm tra độ chính xác, độ tin cậy và biểu đồ nhầm lẫn (confusion matrix) để xác nhận khả năng phân loại.
* Triển khai mô hình (Deployment): Sau khi huấn luyện thành công, Edge Impulse biên dịch mô hình thành mã C++ và tạo ra file *.h* chứa trọng số và cấu trúc mạng. File này được tích hợp vào ESP32-CAM.
* Triển khai thực tế: ESP32-CAM sử dụng mô hình đã huấn luyện để phân tích ảnh mới, xác định xem có xe đối diện không, và gửi kết quả tới ESP32 chính để điều khiển đèn pha LED.
* **Kết quả huấn luyện mô hình học máy:**



*Hình 3.11: Biểu đồ Percision/Recall/F1 Score theo Epoch*

* Phân tích biểu đồ:
* Percision (đường màu xanh): thể hiện tỉ lệ dự đoán đúng trên tổng số mẫu được mô hình dự đoán là dương tính (có xe ngược chiều). Giá trị Precision dao động quanh mức 0.75–0.85 sau khoảng 30 epoch, cho thấy mô hình có khả năng phân loại chính xác tương đối ổn định.
* Recall (đường màu đỏ): đo lường khả năng phát hiện đúng các trường hợp thực sự có xe ngược chiều. Ban đầu thấp, nhưng dần cải thiện và tiệm cận 0.8–0.9 về sau, cho thấy mô hình đã học dần dần được đặc trưng cần thiết để không bỏ sót các đối tượng cần nhận diện.
* F1 Score (đường nét đứt tím): là trung bình điều hòa của Precision và Recall, thể hiện mức cân bằng giữa hai yếu tố. Sau khoảng 15 epoch, F1 Score ổn định ở mức trên 0.8, cho thấy mô hình đạt hiệu suất tương đối tốt và ổn định.



*Hình 3.12: Biểu đồ Train Loss và Validation Loss theo Epoch*

* Phân tích biểu đồ:
* Train Loss (đường nét đứt xanh dương): giảm nhanh chóng trong 10 epoch đầu và tiếp tục giảm chậm sau đó, thể hiện mô hình học được rất nhiều đặc trưng hữu ích từ tập huấn luyện trong giai đoạn đầu.
* Validation Loss (đường nét đứt vàng): cũng giảm đều và song song với train loss, cho thấy mô hình không bị overfitting (quá khớp) và học được các đặc trưng tổng quát phù hợp với dữ liệu mới.
* Sau khoảng 30 epoch, cả hai đường loss gần như đi ngang ở mức thấp (khoảng 0.1), phản ánh sự hội tụ ổn định của mô hình.
* **Đánh giá tổng quan:**

Thông qua hai biểu đồ trên, có thể thấy mô hình học máy đã huấn luyện thành công và đạt độ chính xác tương đối cao trong việc nhận diện vật thể ngược chiều. Quá trình huấn luyện diễn ra ổn định, không có dấu hiệu overfitting nghiêm trọng và độ lệch giữa loss huấn luyện và kiểm thử là nhỏ

* **Phần code ESP32:**
* Khai báo chân (GPIO) và đối tượng UART:

|  |
| --- |
| #define RELAY\_A 18  #define RELAY\_B 19  #define RELAY\_C 14  #define RELAY\_D 12  #define RELAY\_E 26  #define RELAY\_F 27  #define LED\_SAFE 33  #define LED\_OBJECT 32  #define LED\_CAR 25  #define RX\_PIN 13  HardwareSerial mySerial(1); |

* Xác định chân GPIO điều khiển relay A–F
* Gán chân LED báo hiệu cho trạng thái an toàn, vật cản, xe xuất hiện
* RX\_PIN dùng để nhận UART từ ESP32-CAM (chỉ nhận, không truyền)
* mySerial là UART1 (UART mặc định đã dùng cho debug qua USB)
* Khởi tạo biến trạng thái:

|  |
| --- |
| bool current\_relay\_state**[**6**]** **=** **{false,** **false,** **false,** **false,** **false,** **false};**  bool newRelayState**[**6**]** **=** **{false,** **false,** **false,** **false,** **false,** **false};**  bool hasNewRelay **=** **false;**  String inputString **=** ""**;** |

* current\_relay\_state: lưu trạng thái bật/tắt hiện tại của các vùng A–F
* newRelayState: lưu trạng thái mới nhận được từ ESP32-CAM (qua "OTO-x")
* hasNewRelay: cờ kiểm tra có vùng nào vừa được phát hiện hay không
* inputString: lưu chuỗi UART nhận được từ CAM
* Setup() – Khởi tạo hệ thống:

|  |
| --- |
| void setup**()** **{**  Serial**.**begin**(**115200**);**  mySerial**.**begin**(**9600**,** SERIAL\_8N1**,** RX\_PIN**,** **-**1**);** |

* Serial.begin(115200): mở UART0 để in debug ra máy tính
* mySerial.begin(...): mở UART1 để nhận dữ liệu từ ESP32-CAM qua RX13 (không dùng TX)
* Loop() – nhận UART và xử lý:

|  |
| --- |
| **while** **(**mySerial**.**available**())** **{**  char c **=** mySerial**.**read**();** |

* Đọc từng ký tự từ UART, ghép thành chuỗi inputString
* Khi nhận được 1 chuỗi UART hoàn chỉnh (\n):

|  |
| --- |
| **if** **(**c **==** '\n'**)** **{**  inputString**.**trim**();**  inputString**.**toLowerCase**();** |

* Làm sạch và chuyển chuỗi về chữ thường để dễ so sánh
* Xử lý chuỗi nhận được:

Trường hợp “an\_toan”:

|  |
| --- |
| **if** **(**inputString **==** "an\_toan"**)** **{**  allRelaysOff**();**  allLEDsOff**();**  digitalWrite**(**LED\_SAFE**,** HIGH**);** |

* Hệ thống xác định an toàn → tắt toàn bộ relay, chỉ bật LED SAFE

Trường hợp “vat\_can”:

|  |
| --- |
| **else** **if** **(**inputString **==** "vat\_can"**)** **{**  digitalWrite**(**LED\_OBJECT**,** HIGH**);** |

* Hệ thống phát hiện vật thể không phải xe → bật LED cảnh báo vật cản

Trường hợp "oto-x"

|  |
| --- |
| **else** **if** **(**inputString**.**startsWith**(**"oto-"**))** **{**  **...**  newRelayState**[**idx**]** **=** **true;**  hasNewRelay **=** **true;** |

* Khi phát hiện "car" ở vùng nào (A–F):
* Xác định vị trí vùng (idx)
* Ghi nhớ vùng đó vào newRelayState[]
* Đánh dấu hasNewRelay = true để xử lý khi nhận "end"

Trường hợp "end":

|  |
| --- |
| **else** **if** **(**inputString **==** "end"**)** **{**  **if** **(**hasNewRelay**)** **{** |

* Sau khi "oto-x" được gửi xong, CAM gửi "end" để thông báo:

|  |
| --- |
| **for** **(**int i **=** 0**;** i **<** 6**;** i**++)** **{**  **if** **(**newRelayState**[**i**])** → bật relay vùng i nếu chưa bật  **else** → tắt relay vùng i nếu đang bật  **}** **else** **{**  allRelaysOff**();** |

* Lúc này WROOM sẽ so sánh vùng nào có xe, vùng nào không còn
* Nếu không có "OTO-x" nào → coi như an toàn, tắt tất cả
* Reset trạng thái sau xử lý:

|  |
| --- |
| memset**(**newRelayState**,** 0**,** **sizeof(**newRelayState**));**  hasNewRelay **=** **false;**  inputString **=** ""**;** |

* Dọn lại bộ nhớ trạng thái, chuẩn bị cho vòng UART tiếp theo
* Hàm tiện ích:

|  |
| --- |
| void allRelaysOff**()** **{**  **for** **(**int i **=** 0**;** i **<** 6**;** i**++)** digitalWrite**(**relayPin**(**i**),** HIGH**);**  **}** |

* Tắt tất cả relay

|  |
| --- |
| void allLEDsOff**()** **{**  **...**  **}** |

* Tắt tất cả LED

|  |
| --- |
| int relayPin**(**int idx**)** **{**  **switch** **(**idx**)** **{**  **case** 0**:** **return** RELAY\_A**;**  **...** |

* Hàm ánh xạ chỉ số vùng A–F thành chân relay
* **Phần code ESP32-CAM:**
* Khai báo chân và UART:

|  |
| --- |
| #define CAMERA\_MODEL\_AI\_THINKER  #define PWDN\_GPIO\_NUM 32  #define RESET\_GPIO\_NUM -1  #define XCLK\_GPIO\_NUM 0  #define SIOD\_GPIO\_NUM 26  #define SIOC\_GPIO\_NUM 27  #define Y9\_GPIO\_NUM 35  #define Y8\_GPIO\_NUM 34  #define Y7\_GPIO\_NUM 39  #define Y6\_GPIO\_NUM 36  #define Y5\_GPIO\_NUM 21  #define Y4\_GPIO\_NUM 19  #define Y3\_GPIO\_NUM 18  #define Y2\_GPIO\_NUM 5  #define VSYNC\_GPIO\_NUM 25  #define HREF\_GPIO\_NUM 23  #define PCLK\_GPIO\_NUM 22  #define EI\_CAMERA\_RAW\_FRAME\_BUFFER\_COLS 320  #define EI\_CAMERA\_RAW\_FRAME\_BUFFER\_ROWS 240  #define EI\_CAMERA\_FRAME\_BYTE\_SIZE 3  #define RX\_PIN 13  #define TX\_PIN 15  #define MIN\_CAR\_CONFIDENCE 0.7  #define MAX\_REGION 6  HardwareSerial mySerial**(**1**);** // UART1  static bool debug\_nn **=** **false;**  static bool is\_initialised **=** **false;**  uint8\_t **\***snapshot\_buf**;**  static camera\_config\_t camera\_config **=** **{**  **.**pin\_pwdn **=** PWDN\_GPIO\_NUM**,**  **.**pin\_reset **=** RESET\_GPIO\_NUM**,**  **.**pin\_xclk **=** XCLK\_GPIO\_NUM**,**  **.**pin\_sscb\_sda **=** SIOD\_GPIO\_NUM**,**  **.**pin\_sscb\_scl **=** SIOC\_GPIO\_NUM**,**  **.**pin\_d7 **=** Y9\_GPIO\_NUM**,**  **.**pin\_d6 **=** Y8\_GPIO\_NUM**,**  **.**pin\_d5 **=** Y7\_GPIO\_NUM**,**  **.**pin\_d4 **=** Y6\_GPIO\_NUM**,**  **.**pin\_d3 **=** Y5\_GPIO\_NUM**,**  **.**pin\_d2 **=** Y4\_GPIO\_NUM**,**  **.**pin\_d1 **=** Y3\_GPIO\_NUM**,**  **.**pin\_d0 **=** Y2\_GPIO\_NUM**,**  **.**pin\_vsync **=** VSYNC\_GPIO\_NUM**,**  **.**pin\_href **=** HREF\_GPIO\_NUM**,**  **.**pin\_pclk **=** PCLK\_GPIO\_NUM**,**  **.**xclk\_freq\_hz **=** 20000000**,**  **.**ledc\_timer **=** LEDC\_TIMER\_0**,**  **.**ledc\_channel **=** LEDC\_CHANNEL\_0**,**  **.**pixel\_format **=** PIXFORMAT\_JPEG**,**  **.**frame\_size **=** FRAMESIZE\_QVGA**,**  **.**jpeg\_quality **=** 12**,**  **.**fb\_count **=** 1**,**  **.**fb\_location **=** CAMERA\_FB\_IN\_PSRAM**,**  **.**grab\_mode **=** CAMERA\_GRAB\_WHEN\_EMPTY**,**  **};**  bool ei\_camera\_init**(**void**);**  void ei\_camera\_deinit**(**void**);**  bool ei\_camera\_capture**(**uint32\_t img\_width**,** uint32\_t img\_height**,** uint8\_t **\***out\_buf**);**  static int ei\_camera\_get\_data**(**size\_t offset**,** size\_t length**,** float **\***out\_ptr**);** |

* Xác định các chân (GPIO) kết nối với cảm biến camera OV2640.
* Cấu hình độ phân giải, chất lượng JPEG, tần số xung (XCLK), UART, v.v.
* Đặt chế độ đọc ảnh từ bộ nhớ PSRAM (ngoại RAM của ESP32-CAM).
* Setup – Khởi tạo hệ thống:

|  |
| --- |
| Serial**.**begin**(**115200**);**  mySerial**.**begin**(**9600**,** SERIAL\_8N1**,** RX\_PIN**,** TX\_PIN**);** |

* Serial.begin: giao tiếp UART0 để debug qua cổng USB.
* mySerial.begin: UART1 để gửi dữ liệu sang ESP32-WROOM.
* Loop() – Xử lý ảnh và chạy suy luận AI:

|  |
| --- |
| snapshot\_buf **=** **(**uint8\_t**\*)**malloc**(...);**  ei\_camera\_capture**(...)**  run\_classifier**(&**signal**,** **&**result**,** debug\_nn**);** |

* Cấp phát bộ nhớ để lưu ảnh raw RGB từ camera.
* Chụp ảnh từ camera, nạp vào bộ nhớ.
* Tạo tín hiệu đầu vào signal từ ảnh và chạy mô hình suy luận AI đã huấn luyện (FOMO từ Edge Impulse).
* Xử lý kết quả AI:

|  |
| --- |
| bool region\_has\_car**[**MAX\_REGION**]** **=** **{** **false** **};**  bool has\_car **=** **false;**  bool has\_object **=** **false;** |

* Các biến này dùng để lưu:
* Xe xuất hiện ở vùng nào
* Có vật thể không phải xe không
* Không phát hiện gì

|  |
| --- |
| **for** **(...)** **{**  **if** **(**bb**.**value **<** MIN\_CAR\_CONFIDENCE**)** **continue;**  **...**  **}** |

* Duyệt từng vật thể trong ảnh.
* Nếu label == "car" và độ tin cậy đủ cao:
* Tính vùng (chia ảnh theo chiều ngang thành 6 phần)
* Ghi nhận vùng đó có xe
* Nếu không phải "car" → đánh dấu là có vật thể khác.
* Gửi kết quả qua UART:

|  |
| --- |
| **if** **(**has\_car**)** **{**  **for** **(**int i **=** 0**;** i **<** MAX\_REGION**;** i**++)** **{**  **if** **(**region\_has\_car**[**i**])** **{**  sprintf**(**msg**,** "OTO-%c"**,** 'a' **+** i**);**  mySerial**.**println**(**msg**);**  **}**  **}**  mySerial**.**println**(**"end"**);**  **}** |

* Gửi các chuỗi "OTO-a", "OTO-b"... cho từng vùng có xe.
* Sau khi gửi xong hết các vùng, gửi "end" để báo hiệu kết thúc vòng suy luận.

|  |
| --- |
| **else** **if** **(**has\_object**)** **{**  mySerial**.**println**(**"vat\_can"**);**  **}**  **else** **{**  mySerial**.**println**(**"an\_toan"**);**  **}** |

* Nếu chỉ có vật thể khác → gửi "vat\_can"
* Nếu không có gì → gửi "an\_toan"
* Các hàm hỗ trợ:

Khởi tạo camera:

|  |
| --- |
| bool ei\_camera\_init**()** **{** **...** **}** |

* Gọi esp\_camera\_init()
* Cấu hình thêm như lật ảnh (vflip), độ sáng, độ bão hòa nếu dùng cảm biến OV3660.

Chụp ảnh từ camera:

|  |
| --- |
| bool ei\_camera\_capture**(...)** **{** **...** **}** |

* Lấy ảnh từ camera
* Chuyển định dạng JPEG sang RGB888 để mô hình có thể xử lý

Đọc dữ liệu ảnh:

|  |
| --- |
| static int ei\_camera\_get\_data**(...)** **{** **...** **}** |

* Chuyển từng pixel thành float RGB để phục vụ mô hình suy luận.

# CHƯƠNG 4: KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

## Kết luận.

Qua quá trình nghiên cứu và thực hiện đề tài, nhóm chúng em đã xây dựng được một hệ thống ứng dụng UART trong kết nối ESP32-CAM và ESP32 cho hệ thống đèn pha ma trận có khả năng nhận diện xe ngược chiều bằng hình ảnh. Sử dụng ESP32-CAM để thu thập dữ liệu và tích hợp mô hình học máy từ Edge Impulse, hệ thống có thể đưa ra quyết định bật/tắt đèn pha phù hợp, từ đó hạn chế tình trạng gây chói mắt cho phương tiện đối diện và góp phần tăng tính an toàn khi tham gia giao thông vào ban đêm.

Tuy chưa thật sự hoàn hảo, nhưng mô hình bước đầu đã chứng minh được tính khả thi trong việc ứng dụng AI trên phần cứng nhúng giá rẻ. Đề tài cũng giúp nhóm có cái nhìn thực tế hơn về việc xử lý hình ảnh trên thiết bị hạn chế tài nguyên và cách kết hợp giữa AI và điều khiển thiết bị vật lý.

## Hướng phát triển.

Nếu có điều kiện triển khai tiếp trong tương lai, nhóm mong muốn cải thiện hệ thống theo các hướng sau:

* Tăng độ chính xác mô hình AI: hiện tại mô hình vẫn còn gặp khó khăn với các ảnh thiếu sáng hoặc bị nhiễu. Việc bổ sung thêm dữ liệu, đặc biệt là trong điều kiện thực tế, sẽ giúp mô hình ổn định hơn.
* Kết hợp thêm cảm biến ánh sáng: để đèn pha có thể phản ứng không chỉ dựa vào phát hiện xe đối diện mà còn tùy vào điều kiện môi trường (ví dụ trời râm mát hay quá sáng).
* Nâng cấp phần cứng: camera có độ phân giải cao hơn hoặc sử dụng board mạnh hơn như Raspberry Pi sẽ mở rộng khả năng xử lý và giảm độ trễ.
* Tích hợp giao tiếp không dây: nếu có thể kết nối với điện thoại hoặc gửi dữ liệu về server, hệ thống sẽ tiện lợi hơn trong giám sát và cập nhật trạng thái.
* Thử nghiệm thực tế trên xe thật: cuối cùng, hệ thống cần được thử nghiệm trong điều kiện giao thông thực tế để đánh giá chính xác hiệu quả hoạt động, từ đó điều chỉnh thuật toán cho phù hợp.

# TÀI LIỆU THAM KHẢO

<https://edgeimpulse.com/>

<https://docs.ai-thinker.com/>

<https://docs.espressif.com/projects/esp-idf/en/stable/esp32/>

<https://youtu.be/HDRvZ_BYd08?si=2LeWjacaQtHcYQuv> (Simple ESP32-CAM Object Detection)

<https://dronebotworkshop.com/esp32-object-detect/>

<https://github.com/luisomoreau/ESP32-Cam-Edge-Impulse>

<https://www.edgeimpulse.com/blog/add-sight-to-your-esp32/>

<https://dientutuonglai.com/so-do-chan-esp32.html>

<https://lastminuteengineers.com/esp32-cam-pinout-reference/>

<https://circuitdigest.com/microcontroller-projects/object-recognition-using-esp32-cam-and-edge-impulse>