

HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG
KHOA CÔNG NGHỆ THÔNG TIN 1



BÁO CÁO BÀI TẬP LỚN
ĐỀ TÀI: HỆ THỐNG ĐIỀU TIẾT GIAO THÔNG
THÔNG MINH SỬ DỤNG ESP32 VÀ CAMERA IMOU

Môn học: Xây dựng các hệ thống nhúng
Giảng viên: Đỗ Tiên Dũng
Nhóm sinh viên:
Trần Quý Đạt B21DCCN222
Nguyễn Văn Sơn B21DCCN653
Đào Hải Đăng B21DCCN197
Nguyễn Thị Thanh Lam B21DCCN474

HÀ NỘI, 04/2025

LỜI CẢM ƠN

Trước tiên, chúng em xin được bày tỏ lòng biết ơn sâu sắc đến Ban Giám hiệu cùng toàn thể quý thầy cô **Học viện Công nghệ Bưu chính Viễn thông** đã tận tâm giảng dạy, truyền đạt kiến thức, đồng hành và tạo điều kiện thuận lợi cho chúng em trong suốt quá trình học tập và rèn luyện tại trường. Môi trường học tập chuyên nghiệp, năng động cùng sự tận tụy của quý thầy cô chính là nền tảng quý giá giúp chúng em có cơ hội tiếp cận kiến thức chuyên môn cũng như phát triển tư duy nghiên cứu khoa học.

Đặc biệt, chúng em xin gửi lời cảm ơn chân thành và sâu sắc nhất tới **Thầy Đỗ Tiến Dũng** – giảng viên bộ môn “**Xây dựng các hệ thống nhúng**”, người đã trực tiếp hướng dẫn chúng em trong suốt quá trình thực hiện đề tài. Thầy không chỉ là người truyền đạt kiến thức mà còn là người định hướng, hỗ trợ và khơi dậy niềm đam mê nghiên cứu trong chúng em. Trong suốt quá trình làm đề tài, thầy đã dành nhiều thời gian quý báu để tận tình hướng dẫn, góp ý chỉnh sửa và tạo mọi điều kiện thuận lợi nhất để chúng em có thể hoàn thành tốt nhiệm vụ được giao. Những chỉ dẫn tận tâm và sự khích lệ kịp thời của thầy chính là động lực to lớn giúp chúng em vượt qua những khó khăn, thử thách trong quá trình triển khai đề tài.

Chúng em cũng xin gửi lời cảm ơn đến các anh/chị khóa trước, bạn bè và những cá nhân, tập thể đã hỗ trợ, đóng góp ý kiến, chia sẻ tài liệu, cũng như giúp đỡ chúng em trong quá trình khảo sát, tìm hiểu thực tế và thực hiện đề tài.

Trong quá trình nghiên cứu và thực hiện, mặc dù đã cố gắng nỗ lực hết mình, nhưng do hạn chế về kiến thức thực tiễn cũng như kinh nghiệm còn non trẻ, nên đề tài chắc chắn vẫn còn tồn tại những thiếu sót nhất định. Chúng em kính mong nhận được sự cảm thông và góp ý từ quý thầy cô để chúng em có thể hoàn thiện hơn trong các nghiên cứu và học tập sau này.

Một lần nữa, chúng em xin chân thành cảm ơn!

MỤC LỤC

CHƯƠNG 1: TỔNG QUAN ĐỀ TÀI.....	5
1.1 Đặt vấn đề.....	5
1.2 Mục tiêu.....	5
1.2 Đối tượng và phạm vi nghiên cứu.....	6
1.3 Phương pháp thực hiện.....	7
CHƯƠNG 2: CƠ SỞ LÝ THUYẾT.....	9
2.1 Thành phần điều khiển.....	9
2.1.1 Vi xử lý ESP32.....	9
2.1.2 Mạch 4 LED 7 đoạn TM1637.....	10
2.1.3 Đèn LED - Đèn LED.....	11
2.1.4 Thành phần khác.....	11
2.2 Thành phần thu thập và xử lý dữ liệu.....	12
2.2.1 Camera Imou.....	12
2.2.2 Mô hình YOLO v8.....	13
2.3 Thành phần quản lý dữ liệu.....	18
ThinkSpeak.....	18
2.4 Thành phần giám sát hệ thống.....	19
CHƯƠNG 3: THIẾT KẾ, CÀI ĐẶT.....	21
VÀ THỬ NGHIỆM HỆ THỐNG.....	21
3.1 Thiết kế tổng quan.....	21
3.1.1 Các khái niệm cơ bản của hệ thống.....	21
3.1.2 Một số logic xử lý chính.....	22
Cập nhật tín hiệu điều khiển.....	22
Cập nhật bộ đếm thời gian.....	23
3.2 Thiết kế chi tiết.....	30
3.2.1 Chi tiết module điều khiển.....	30
3.2.2 Chi tiết module xử lý dữ liệu.....	32
3.2.3. Chi tiết module giám sát.....	42
3.2.3.1. Kiến trúc tổng thể.....	42
3.2.3.2. Quản lý state và tối ưu hóa hiệu suất.....	42
3.2.3.3. Hệ thống tham chiếu DOM trực tiếp.....	43
3.2.4. Chi tiết module điều khiển.....	45
3.3 Cài đặt thử nghiệm và đánh giá.....	56
3.3.1. Phần cứng.....	56
3.3.2. Phần mềm.....	63

KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN.....	71
TÀI LIỆU THAM KHẢO.....	75

CHƯƠNG 1: TỔNG QUAN ĐỀ TÀI

1.1 Đặt vấn đề

Hiện nay, tình trạng ùn tắc giao thông đang là một trong những vấn đề nan giải tại các thành phố lớn, ảnh hưởng trực tiếp đến chất lượng sống, môi trường và hiệu quả kinh tế – xã hội. Mặc dù nhiều hệ thống điều tiết giao thông đã được triển khai, đa phần vẫn vận hành theo cơ chế cố định, chưa phản ánh linh hoạt mật độ phương tiện theo thời gian thực, dẫn đến việc phân phối thời gian đèn tín hiệu chưa hợp lý, gây lãng phí tài nguyên đường phố và làm tăng nguy cơ tắc nghẽn.

Trong bối cảnh đó, việc ứng dụng các công nghệ mới như Internet of Things (IoT), xử lý ảnh và trí tuệ nhân tạo (AI) vào giám sát và điều khiển giao thông đã mở ra những hướng tiếp cận tiềm năng và hiệu quả hơn. Các hệ thống sử dụng camera giám sát kết hợp với các mô hình học sâu như YOLO v8 có khả năng nhận diện phương tiện nhanh chóng và chính xác, từ đó phân tích được lưu lượng giao thông tại từng thời điểm. Dữ liệu này có thể được truyền đến bộ vi điều khiển như ESP32 WROOM để điều chỉnh thời gian pha đèn linh hoạt, giúp tối ưu hóa luồng di chuyển trên các tuyến đường.

Xuất phát từ nhu cầu thực tiễn và xu hướng phát triển của công nghệ, đề tài “Xây dựng hệ thống điều tiết giao thông sử dụng camera Imou, YOLO v8 và ESP32 WROOM” được thực hiện nhằm nghiên cứu, thiết kế và triển khai một hệ thống giao thông thông minh có khả năng tự động thu thập dữ liệu hình ảnh, phân tích mật độ lưu thông và điều khiển đèn giao thông theo thời gian thực. Việc triển khai đề tài không chỉ góp phần nâng cao năng lực chuyên môn cho nhóm thực hiện, mà còn mở ra các khả năng ứng dụng thực tế trong việc cải thiện hạ tầng giao thông tại Việt Nam.

1.2 Mục tiêu

Mục tiêu của đề tài đặt ra là:

- Xây dựng hệ thống thu thập dữ liệu giao thông thông qua camera giám sát (camera Imou), đảm bảo khả năng hoạt động ổn định và truyền hình ảnh về trung tâm xử lý.
- Ứng dụng mô hình AI (YOLO v8) để xử lý ảnh, nhận diện và phân loại phương tiện giao thông, từ đó đánh giá mật độ lưu thông theo thời gian thực.

- Thiết kế và lập trình bộ vi điều khiển (ESP32 WROOM) để tiếp nhận kết quả phân tích từ AI và điều khiển đèn tín hiệu giao thông tự động, linh hoạt theo lưu lượng xe.
- Tích hợp chức năng giám sát giao thông trực tuyến và cho phép can thiệp điều khiển thủ công từ người quản lý khi cần thiết, đặc biệt trong các tình huống bất thường hoặc khẩn cấp.
- Xây dựng hệ thống hoàn chỉnh, kết hợp phần cứng và phần mềm, có thể triển khai thực tế hoặc mô phỏng hoạt động của nút giao thông thông minh.
- Đánh giá hiệu quả hoạt động của hệ thống so với phương pháp điều khiển cố định truyền thống, từ đó đề xuất hướng cải tiến và mở rộng ứng dụng trong tương lai.

1.2 Đối tượng và phạm vi nghiên cứu

Đối tượng nghiên cứu của đề tài là hệ thống điều khiển giao thông thông minh tích hợp công nghệ IoT và trí tuệ nhân tạo, có khả năng thu thập, xử lý dữ liệu hình ảnh từ camera giao thông và điều khiển đèn tín hiệu một cách linh hoạt theo thời gian thực. Cụ thể:

- Camera giám sát giao thông (Imou): dùng để thu thập hình ảnh tại nút giao thông.
- Mô hình xử lý ảnh YOLO v8: thực hiện nhận diện và phân loại phương tiện giao thông, đánh giá mật độ lưu thông dựa trên hình ảnh thu được từ camera.
- Nền tảng Thingspeak: đóng vai trò trung gian truyền dữ liệu giữa mô hình AI và bộ điều khiển, giúp lưu trữ và chia sẻ tín hiệu phân tích lưu lượng giao thông một cách trực tuyến.
- Bộ vi điều khiển ESP32 WROOM: lấy dữ liệu từ Thingspeak để điều khiển đèn tín hiệu giao thông tự động, đồng thời cho phép giám sát và can thiệp điều khiển thủ công khi cần thiết.
- Hệ thống phần mềm tích hợp: bao gồm giao diện người dùng, chức năng giám sát, kết nối và điều khiển giữa các thành phần trong toàn hệ thống.

Về phạm vi nghiên cứu, đề tài tập trung nghiên cứu và triển khai hệ thống điều khiển đèn giao thông thông minh ở quy mô mô phỏng, thử nghiệm tại một nút giao thông đơn lẻ, với các giới hạn cụ thể như sau:

- Về không gian: chỉ triển khai tại một ngã tư giả lập có 4 làn đường thu nhỏ với 2 làn dọc và 2 làn ngang, không áp dụng cho toàn bộ hệ thống giao thông đô thị.

- Về thiết bị: sử dụng camera Imou để thu thập hình ảnh, vi điều khiển ESP32 WROOM để điều khiển tín hiệu đèn, nền tảng Thingspeak để truyền dữ liệu, không mở rộng sang các loại cảm biến hoặc thiết bị chuyên dụng khác.
- Về công nghệ xử lý: sử dụng mô hình YOLO v8 để nhận diện và phân loại phương tiện (xe máy, ô tô, xe tải), không xử lý các yếu tố phức tạp như nhận diện biển số, hành vi vi phạm, hay dự đoán luồng phương tiện trong tương lai.
- Về điều khiển: điều chỉnh đèn giao thông dựa trên mật độ lưu thông hiện tại, cho phép chế độ tự động và thủ công, chưa tích hợp cơ chế điều phối liên nút hoặc điều khiển ưu tiên cho xe ưu tiên (cứu thương, cứu hỏa...).
- Về thời gian và điều kiện: mô phỏng hoạt động trong điều kiện ánh sáng và thời tiết ổn định, không xét đến các tình huống đặc biệt như mưa lớn, ban đêm hoặc mất kết nối mạng.

1.3 Phương pháp thực hiện

Để triển khai hệ thống điều khiển giao thông thông minh sử dụng camera Imou, YOLO v8, ESP32 WROOM và nền tảng Thingspeak, đề tài áp dụng các phương pháp sau:

- Nghiên cứu lý thuyết
 - Tìm hiểu về các hệ thống điều khiển giao thông hiện có và các công nghệ liên quan như IoT, xử lý ảnh, trí tuệ nhân tạo.
 - Khảo sát nguyên lý hoạt động của các thiết bị phần cứng (camera IP, ESP32 WROOM) và phần mềm (YOLO v8, Thingspeak).
- Xây dựng mô hình xử lý dữ liệu
 - Huấn luyện hoặc tinh chỉnh mô hình YOLO v8 để nhận diện, phân loại phương tiện và ước lượng mật độ giao thông.
 - Cập nhật thông số lưu lượng giao thông lên Thingspeak để bộ điều khiển ESP32 WROOM có thể lấy và sử dụng dữ liệu đó để điều chỉnh thời gian đèn giao thông.
- Thiết kế hệ thống điều khiển
 - Lập trình ESP32 WROOM để lấy dữ liệu từ Thingspeak và điều khiển thời gian đèn giao thông tương ứng.
- Tích hợp và triển khai hệ thống
 - Kết nối các thành phần phần cứng và phần mềm thành một hệ thống hoàn chỉnh.
 - Điều chỉnh tham số để tối ưu hiệu quả điều tiết đèn theo mật độ lưu thông thực tế.

- **Dánh giá và báo cáo**

- So sánh kết quả điều khiển giữa hệ thống tự động và phương pháp điều khiển cố định.
- Phân tích ưu, nhược điểm của hệ thống và đề xuất hướng cải tiến trong tương lai.

CHƯƠNG 2: CƠ SỞ LÝ THUYẾT

2.1 Thành phần điều khiển

2.1.1 Vi xử lý ESP32



Hình 1. Vi điều khiển ESP32

ESP32 là một dòng **vi điều khiển** có thể ghép nối linh hoạt với cảm biến, màn hình, động cơ và các module không dây khác; được thiết kế với hiệu năng cao, tiêu thụ điện năng thấp. Vi điều khiển này hỗ trợ nhiều giao thức giao tiếp phổ biến như UART (truyền dữ liệu nối tiếp), SPI (truyền dữ liệu lớn, tốc độ cao), I2C,... và có thể lập trình bằng nhiều nền tảng như Arduino IDE, ESP-IDF hoặc MicroPython.

ESP32 có hỗ trợ Wi-Fi & Bluetooth với nguyên lí hoạt động là:

- Đối với Wifi chuẩn 802.11 b/g/n (2.4GHz)
 - ESP32 kết nối vào mạng Wi-Fi với vai trò trạm phát sóng (Station) hoặc phát mạng làm điểm truy cập (Access Point).
 - Giao tiếp dựa trên TCP/IP stack (được tích hợp sẵn).
 - Dữ liệu truyền/nhận qua socket hoặc HTTP/MQTT.
ESP32 dùng module MAC (Media Access Control) và PHY (Physical Layer) tích hợp để xử lý tín hiệu Wi-Fi.
- Đối với Bluetooth phiên bản v4.2 (Classic + BLE)
 - ESP32 có thể đóng vai trò là "thiết bị gửi dữ liệu" hoặc "thiết bị nhận dữ liệu" trong kết nối Bluetooth năng lượng thấp (BLE).
 - Dữ liệu trong BLE được chia nhỏ thành từng phần, gọi là "thuộc tính" (Characteristic), nằm trong các nhóm chức năng gọi là "dịch vụ"

(Service), và các dịch vụ này thuộc về một loại ứng dụng cụ thể gọi là "profile".

Tổng quan cấu trúc các chân theo chức năng trên ESP32-WROOM-32.

STT	Phân loại	Chân	Chức năng
1	Chân nguồn	3V3	Cấp điện cho module
		GND	Chân nối đất
2	Chân đa năng	Các chân GPIO thứ tự 0-5, 12-19, 21-23, 25-27, 32-36, 39	Nhận tín hiệu/dữ liệu vào/ra
3	Chân giao tiếp	Các chân GPIO thứ tự 1, 3, 9, 10, 16, 17	Giao tiếp UART
		Các chân GPIO thứ tự 5, 18, 19, 23	Giao tiếp SPI
		Các chân GPIO thứ tự 22, 23	Giao tiếp I2C
4	Chân chuyển đổi	ADC	Chuyển đổi tín hiệu tương tự sang tín hiệu số
		DAC	Chuyển đổi tín hiệu số sang tín hiệu tương tự
5	Chân đặc biệt	EN	Reset chip
		GPIO0	Chuyển sang chế độ BOOT

Bảng 1. Tổng quan các chân chức năng của ESP32

2.1.2 Mạch 4 LED 7 đoạn TM1637



Hình 2. 4 LED 7 đoạn TM1637

TM1637 là một **IC điều khiển LED 7 đoạn** phổ biến, thường được dùng để điều khiển các màn hình hiển thị số 4 chữ số trong các thiết bị nhúng như Arduino, ESP8266, ESP32.

Cấu trúc các chân chức năng của TM1637 như sau.

STT	Phân loại	Chân	Chức năng
1	Chân nguồn	VCC	Cấp điện áp cho module
		GND	Chân nối đất
2	Chân xung đồng hồ	CLK	Điều khiển, đồng bộ nhịp dữ liệu truyền nhận
3	Chân dữ liệu	DIO	Chân truyền nhận dữ liệu

Bảng 2. Tổng quan các chân chức năng của TM1637

2.1.3 Đèn LED

Đèn LED là một loại **diode bán dẫn** có khả năng phát ra ánh sáng có nhiều màu khi dòng điện chạy qua. Đây là một công nghệ chiếu sáng được sử dụng rộng rãi trong nhiều ứng dụng, từ đèn chiếu sáng cho đến hiển thị và các tín hiệu điện tử.



Mỗi LED có 2 chân, gồm:

Hình 3. Đèn LED

- Chân dương (chân dài hơn) được kết nối với cực dương của nguồn điện, dòng điện sẽ đi vào đây.
- Chân âm (chân ngắn hơn) được kết nối với cực âm của nguồn điện.

2.1.4 Thành phần khác



Hình 4. Breadboard



Hình 5. Dây nối

Dùng để ráp mạch điện mà không cần hàn.
Dùng để ghép nối các thành phần phân cứng.

2.2 Thành phần thu thập và xử lý dữ liệu

2.2.1 Camera Imou

Camera IMOU là dòng **camera an ninh** thông minh do hãng Dahua Technology phát triển, chuyên dùng cho giám sát gia đình, văn phòng hoặc cửa hàng.

Camera IMOU có thiết kế nhỏ gọn, dễ lắp đặt và hỗ trợ kết nối Wi-Fi/cáp Ethernet tiện lợi. Thiết bị tích hợp nhiều tính năng thông minh như quan sát ban đêm, phát hiện chuyển động và đàm thoại hai chiều. Ngoài ra, IMOU còn hỗ trợ lưu trữ linh hoạt qua thẻ nhớ, đám mây hoặc NVR.



Hình 6. Camera Imou

Để truyền thông tin video streaming, Imou sử dụng giao thức **RTSP (Real-Time Streaming Protocol)**, cụ thể:

- Cấu hình RTSP trên Camera thông qua App IMOU Life bằng cách bật RTSP và cài đặt username/password để xác thực.
- Lấy địa chỉ RTSP từ các thông tin username/password thiết lập trước đó, địa chỉ IP nội bộ của camera (trong App) và chỉ định kênh/luồng lấy dữ liệu.

```
rtsp://<username>:<password>@<IP_camera>:554/cam/realmonitor?channel
```

- Sử dụng phần mềm hỗ trợ RTSP như VLC Media Player/iSpy/Blue Iris/OBS mở luồng mạng với địa chỉ trên và phát để xem trực tuyến.
- Trong trường hợp muốn streaming video từ camera IMOU thông qua giao thức RTSP, cần lấy địa chỉ RTSP của camera và dùng phần mềm trung gian như FFmpeg hoặc media server để chuyển đổi luồng RTSP thành định dạng tương thích với trình duyệt (như HLS hoặc MJPEG). Sau đó, ứng dụng web có thể hiển thị luồng video thông qua thẻ <video> hoặc .

Để điều khiển chuyển động của camera, Imou hỗ trợ giao thức **ONVIF (Open Network Video Interface Forum)**, cụ thể:

- Cấu hình ONVIF trên Camera thông qua App IMOU Life bằng cách bật ONVIF và cài đặt username/password để xác thực.
- Sử dụng phần mềm hỗ trợ ONVIF như ONVIF Device Manager/iSpy/Blue Iris/Agent DVR, nhập thông tin username/password thiết lập trước đó và IP camera và truy cập luồng, điều khiển camera (xoay, quay), gửi lệnh điều khiển.
- Trong trường hợp muốn truy cập và điều khiển camera IMOU từ các server khác, cần cài đặt thư viện ONVIF tương ứng với ngôn ngữ lập trình được sử dụng, chẳng hạn như *onvif-zeep* cho Python, *onvif* SDK cho C++, hoặc *node-onvif* cho Node.js. Các thư viện này cho phép kết nối, gửi lệnh điều khiển PTZ, truy vấn cấu hình và luồng video từ xa thông qua giao thức ONVIF.

2.2.2 Mô hình YOLO v8

Giới thiệu và Lựa chọn Mô hình:

Trong bối cảnh xây dựng Hệ thống Điều tiết Giao thông Thông minh sử dụng ESP32 và Camera IMOU, khả năng nhận diện và định lượng chính xác các phương tiện đang lưu thông tại các nút giao thông là yếu tố then chốt. Bài toán này thuộc lĩnh vực phát hiện đối tượng (Object Detection) trong thị giác máy tính. Giữa vô số các kiến trúc mạng nơ-ron sâu, họ mô hình YOLO (You Only Look Once) đã khẳng định được vị thế nhờ sự cân bằng ánh tượng giữa tốc độ xử lý và độ chính xác, đặc biệt phù hợp với các ứng dụng yêu cầu thời gian thực. YOLO tiếp cận bài toán bằng cách nhìn vào toàn

bộ ảnh một lần duy nhất để đồng thời dự đoán vị trí (bounding box) và lớp (class) của các đối tượng.

Phiên bản YOLOv8, do Ultralytics phát triển, đại diện cho thế hệ mới nhất của họ YOLO (tại thời điểm nghiên cứu), kế thừa và cải tiến từ các phiên bản tiền nhiệm. YOLOv8 không chỉ là một mô hình mà là một framework linh hoạt, giới thiệu các cải tiến đáng kể như kiến trúc Anchor-Free, đầu mạng tách rời (Decoupled Head), và các hàm mất mát mới, giúp nâng cao hiệu suất tổng thể.

Trong số các biến thể của YOLOv8 (n, s, m, l, x), chúng tôi đã quyết định lựa chọn YOLOv8n (Nano). Lý do chính cho sự lựa chọn này là tối ưu hóa cho tốc độ và hiệu quả tài nguyên. Hệ thống điều tiết giao thông đòi hỏi phản ứng nhanh chóng với tình hình thực tế, do đó tốc độ xử lý (FPS - Frames Per Second) của mô hình phát hiện là cực kỳ quan trọng. YOLOv8n, là phiên bản nhẹ nhất, được thiết kế để đạt được FPS cao nhất, phù hợp với việc xử lý luồng video liên tục từ camera. Mặc dù điều này có thể đi kèm với một sự đánh đổi nhỏ về độ chính xác tối đa so với các phiên bản lớn hơn (như YOLOv8s hay YOLOv8m), nhưng đối với bài toán đếm các phương tiện giao thông phổ biến (ô tô, xe máy), độ chính xác của YOLOv8n được đánh giá là đủ dùng, đặc biệt khi sử dụng các mô hình đã được huấn luyện trước trên các bộ dữ liệu lớn. Hơn nữa, yêu cầu tài nguyên thấp hơn cũng giúp giảm chi phí phần cứng xử lý và tiêu thụ năng lượng.

Kiến trúc và Nguyên lý Hoạt động:

YOLOv8n, tương tự các thành viên khác trong gia đình YOLOv8, có cấu trúc gồm ba thành phần chính:

1. Backbone (Mạng xương sống): Đây là bộ trích xuất đặc trưng chính. Nó thường dựa trên kiến trúc CSPDarknet (Cross Stage Partial Darknet), một kiến trúc hiệu quả đã được chứng minh trong các phiên bản YOLO trước đó. Backbone của YOLOv8n được thiết kế với số lượng lớp và kênh ít hơn đáng kể so với các phiên bản lớn, nhằm giảm độ phức tạp tính toán và số lượng tham số, từ đó tăng

tốc độ suy luận. Nó xử lý ảnh đầu vào và tạo ra các bản đồ đặc trưng (feature maps) ở các độ phân giải khác nhau.

2. Neck (Cỗ mạng): Phần này đóng vai trò cầu nối, tổng hợp và tinh chỉnh các đặc trưng được trích xuất từ các tầng khác nhau của Backbone. YOLOv8 thường sử dụng các cơ chế như PANet (Path Aggregation Network) để kết hợp hiệu quả thông tin từ các đặc trưng cấp thấp (chi tiết, độ phân giải cao) và cấp cao (ngữ nghĩa, độ phân giải thấp). Việc kết hợp đa tỷ lệ này rất quan trọng để mô hình có thể phát hiện tốt các phương tiện ở các kích thước và khoảng cách khác nhau trong khung hình.
3. Head (Đầu mạng): Đây là phần thực hiện dự đoán cuối cùng. YOLOv8 mang đến hai cải tiến quan trọng ở phần Head:
 - Decoupled Head: Tách biệt nhiệm vụ dự đoán vị trí (bounding box regression) và dự đoán lớp (classification) thành các nhánh xử lý riêng biệt. Cách tiếp cận này thường giúp cải thiện độ chính xác so với việc gộp chung hai nhiệm vụ.
 - Anchor-Free Design: Thay vì dựa vào một tập hợp các hộp neo (anchor boxes) được định nghĩa trước với các tỷ lệ và kích thước cố định, mô hình Anchor-Free trực tiếp dự đoán tâm của đối tượng cùng với chiều rộng và chiều cao của nó. Điều này giúp giảm số lượng dự đoán, đơn giản hóa quá trình khớp nhãn trong huấn luyện và tăng khả năng thích ứng với các đối tượng có hình dạng đa dạng hoặc tỷ lệ khung hình không chuẩn.
4. Khi một khung hình từ camera IMOU được đưa vào YOLOv8n, nó sẽ đi qua Backbone để trích xuất đặc trưng, qua Neck để tổng hợp và làm giàu đặc trưng, và cuối cùng qua Head để tạo ra một loạt các dự đoán về vị trí bounding box và xác suất lớp cho các đối tượng tiềm năng. Sau đó, một bước hậu xử lý quan trọng là Non-Max Suppression (NMS) được áp dụng để lọc bỏ các bounding box trùng lặp hoặc có độ tin cậy thấp, chỉ giữ lại những dự đoán tốt nhất cho mỗi phương tiện được phát hiện.

Tích hợp vào Hệ thống Điều tiết Giao thông:

Trong kiến trúc hệ thống đề xuất, YOLOv8n đóng vai trò trung tâm trong việc thu thập dữ liệu mật độ giao thông:

1. Thu nhận Ảnh: Camera IMOU ghi lại hình ảnh/video tại các hướng của nút giao.
2. Truyền Dữ liệu: Dữ liệu hình ảnh/video được gửi đến một thiết bị xử lý chuyên dụng (ví dụ: máy tính cá nhân, máy tính nhúng như Raspberry Pi 4/Jetson Nano, hoặc một dịch vụ đám mây). Điều quan trọng cần nhấn mạnh là ESP32 không đủ khả năng tính toán và bộ nhớ để chạy trực tiếp mô hình YOLOv8n.
3. Suy luận với YOLOv8n: Trên thiết bị xử lý, các khung hình được tiền xử lý (resize, chuẩn hóa) và đưa vào mô hình YOLOv8n đã được tải (thường là mô hình tiền huấn luyện trên bộ dữ liệu COCO, vốn đã chứa các lớp 'car', 'motorcycle', 'bus', 'truck'). Mô hình thực hiện suy luận (inference) và trả về danh sách các phương tiện được phát hiện cùng tọa độ bounding box và lớp của chúng.
4. Đếm Phương tiện: Một logic đếm được áp dụng dựa trên kết quả của YOLOv8n. Phương pháp phổ biến là định nghĩa các Vùng Quan tâm (Region of Interest - ROI) tương ứng với từng làn đường hoặc hướng di chuyển. Hệ thống sẽ đếm số lượng bounding box có tâm nằm trong hoặc giao cắt đáng kể với các ROI này. Để tăng độ chính xác, có thể kết hợp với việc theo dõi đối tượng qua các khung hình để tránh đếm lặp hoặc bỏ sót.
5. Truyền Kết quả đến ESP32: Số lượng xe đếm được (hoặc một chỉ số đại diện cho mật độ giao thông) cho mỗi hướng được gửi từ thiết bị xử lý đến ESP32 thông qua kết nối mạng (ví dụ: Wi-Fi) bằng các giao thức như MQTT, HTTP request, hoặc WebSockets.
6. Ra Quyết định và Điều khiển: ESP32 nhận dữ liệu mật độ từ các hướng, áp dụng thuật toán điều khiển đã lập trình (ví dụ: ưu tiên hướng đông xe hơn, điều chỉnh chu kỳ đèn linh hoạt) để xác định thời gian xanh/đỏ tối ưu cho từng pha đèn và gửi tín hiệu điều khiển tương ứng đến các đèn tín hiệu.

Huấn luyện và Hiệu năng:

Thông thường, chúng tôi tận dụng mô hình YOLOv8n đã được huấn luyện trước (pre-trained) trên bộ dữ liệu lớn như COCO. Điều này tiết kiệm đáng kể thời gian và tài nguyên huấn luyện, đồng thời cung cấp khả năng nhận diện cơ bản tốt cho các loại phương tiện phổ biến. Tuy nhiên, để đạt hiệu suất tối ưu trong điều kiện cụ thể của giao thông Việt Nam hoặc tại một nút giao cụ thể (ánh sáng, góc nhìn, loại xe đặc thù), có thể thực hiện tinh chỉnh (fine-tuning) mô hình. Quá trình này bao gồm việc huấn luyện tiếp mô hình tiền huấn luyện trên một bộ dữ liệu nhỏ hơn, được thu thập và gán nhãn tại chính địa điểm triển khai. (Nếu bạn có thực hiện, mô tả thêm, nếu không, đề cập là hướng phát triển).

Hiệu năng của YOLOv8n được đánh giá qua:

- Độ chính xác: Sử dụng các chỉ số như Precision, Recall, và đặc biệt là mAP (mean Average Precision) để đo lường khả năng phát hiện đúng và đủ các đối tượng. mAP càng cao, mô hình càng chính xác.
- Tốc độ: Đo bằng FPS (Frames Per Second). YOLOv8n có thể đạt FPS rất cao trên phần cứng phù hợp (đặc biệt là GPU), đảm bảo khả năng xử lý thời gian thực.

Ưu điểm và Hạn chế:

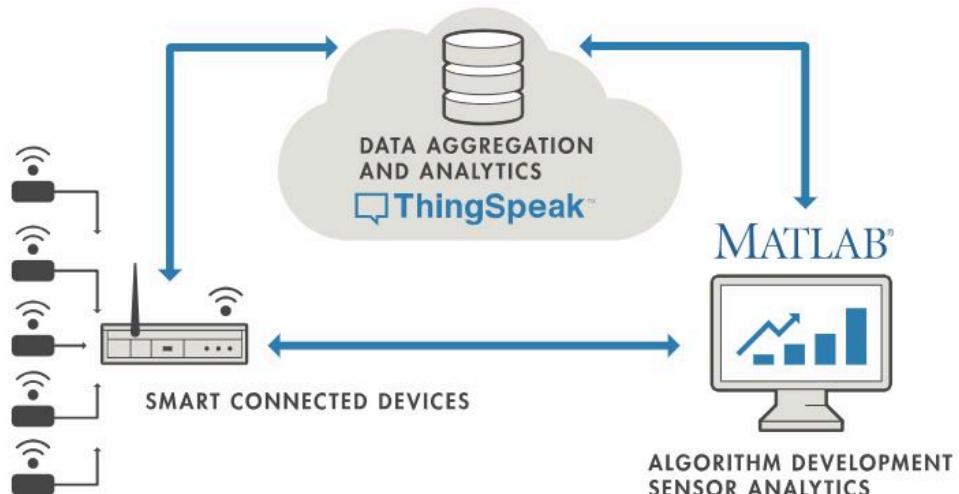
- **Ưu điểm:** Tốc độ cực nhanh, yêu cầu tài nguyên tính toán thấp, độ chính xác tốt cho các tác vụ phổ thông, dễ sử dụng và tích hợp thông qua framework Ultralytics, là một mô hình hiện đại (State-of-the-Art).
- **Hạn chế:** Độ chính xác có thể bị ảnh hưởng bởi đối tượng nhỏ, bị che khuất, điều kiện ánh sáng/thời tiết xấu. Đòi hỏi phần cứng xử lý riêng biệt (không phải ESP32). Việc đếm chính xác trong điều kiện phức tạp có thể cần kết hợp thêm thuật toán theo dõi. Độ trễ mạng giữa các thành phần (camera, bộ xử lý, ESP32) cũng cần được xem xét.

Kết luận về vai trò của YOLOv8n:

Tóm lại, mô hình YOLOv8n là một thành phần công nghệ không thể thiếu, mang lại "trí thông minh" cho hệ thống điều tiết giao thông đề xuất. Bằng cách cung cấp khả năng phát hiện và đếm phương tiện nhanh chóng, hiệu quả, YOLOv8n tạo ra dữ liệu đầu vào đáng tin cậy cho thuật toán điều khiển đèn tín hiệu của ESP32, cho phép hệ thống thích ứng linh hoạt với tình hình giao thông thực tế. Mặc dù có những thách thức nhất định, việc lựa chọn YOLOv8n thể hiện một giải pháp kỹ thuật cân bằng, hiệu quả và phù hợp với mục tiêu của đề tài, đồng thời mở ra tiềm năng ứng dụng rộng rãi của học sâu trong việc hiện đại hóa hệ thống giao thông đô thị.

2.3 Thành phần quản lý dữ liệu

ThinkSpeak



Hình 7. Nền tảng ThinkSpeak

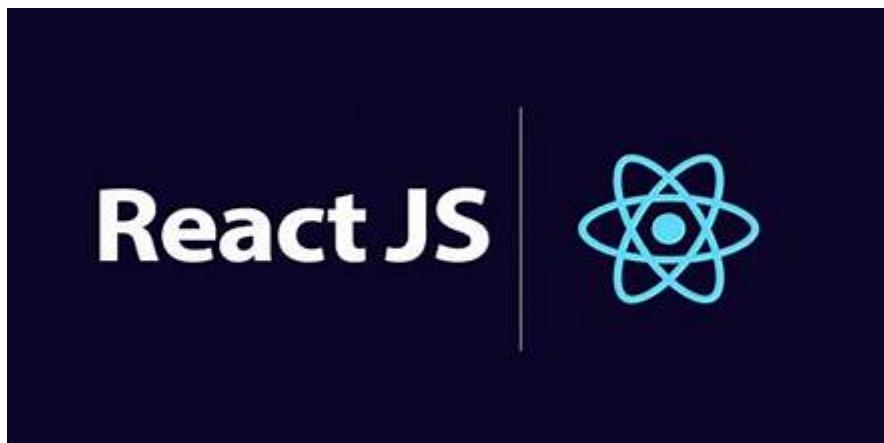
ThingSpeak là một nền tảng IoT (Internet of Things) mã nguồn mở cho phép thu thập, lưu trữ, phân tích và trực quan hóa dữ liệu cảm biến từ thiết bị kết nối Internet. Nó hỗ trợ giao tiếp qua giao thức HTTP, MQTT và cung cấp các biểu đồ thời gian thực, thuận tiện cho các ứng dụng giám sát từ xa như nhiệt độ, độ ẩm, môi trường, ... Cụ thể:

- Thu thập dữ liệu: Các thiết bị IoT (như Arduino, ESP8266, ESP32...) hoặc server API có thể gửi dữ liệu cảm biến, dữ liệu điều khiển lên ThingSpeak qua HTTP hoặc MQTT.
- Lưu trữ dữ liệu: ThingSpeak lưu trữ dữ liệu theo từng kênh (channel), mỗi kênh có thể chứa nhiều trường (fields) tương ứng với các loại cảm biến.

- Phân tích và xử lý: Có thể sử dụng tích hợp MATLAB để xử lý dữ liệu trực tiếp trên nền tảng (ví dụ: tính trung bình, phát hiện bất thường...).
- Hiển thị dữ liệu: ThingSpeak tự động tạo biểu đồ trực quan để theo dõi dữ liệu theo thời gian thực.
- Kích hoạt hành động: Có thể cấu hình triggers hoặc webhook để điều khiển thiết bị dựa trên dữ liệu thu thập được.

2.4 Thành phần giám sát hệ thống

ReactJS là một thư viện JavaScript mã nguồn mở được phát triển bởi Facebook (nay là Meta) vào năm 2013, tập trung vào việc xây dựng giao diện người dùng cho các ứng dụng web hiện đại. Khác với các framework toàn diện như Angular, React là một thư viện chuyên biệt cho việc xây dựng UI với kiến trúc dựa trên component, cho phép nhà phát triển tạo ra các thành phần giao diện có thể tái sử dụng và quản lý trạng thái một cách hiệu quả. Một trong những điểm mạnh nổi bật của ReactJS là cơ chế Virtual DOM - một bản sao nhẹ của DOM thật, được sử dụng để tối ưu hóa việc cập nhật giao diện. Thay vì cập nhật trực tiếp DOM thật (một quá trình tốn kém về mặt hiệu suất), React chỉ cập nhật những phần thực sự thay đổi, giúp tăng đáng kể hiệu suất ứng dụng.



Trong bối cảnh stream video qua WebSocket, ReactJS thể hiện nhiều ưu điểm vượt trội. Đầu tiên là khả năng cập nhật UI liên tục với tốc độ cao - yếu tố quyết định trong các ứng dụng streaming thời gian thực. Virtual DOM của React đặc biệt hiệu quả khi cần cập nhật nhanh chóng các frame video liên tiếp nhận được từ WebSocket. Thứ hai, React cung cấp nhiều cách tiếp cận linh hoạt để quản lý state, từ useState đơn giản đến các thư viện phức tạp như Redux, giúp việc quản lý buffer video, trạng thái kết nối và độ trễ trở nên đơn giản hơn. React cũng cung cấp hook useRef cho phép lưu trữ và cập nhật giá trị mà không gây ra re-render, một tính năng đặc biệt quan trọng trong ứng dụng streaming khi cần giảm thiểu các render không cần thiết để tối ưu hiệu suất.

WebSocket kết hợp với ReactJS tạo nên giải pháp lý tưởng cho ứng dụng streaming video. WebSocket thiết lập kênh giao tiếp hai chiều giữa client và server, cho phép truyền dữ liệu liên tục mà không cần thiết lập kết nối mới cho mỗi request như trong HTTP truyền thống. Việc này giảm đáng kể độ trễ - yếu tố cực kỳ quan trọng trong ứng dụng stream video. Trong các ứng dụng thực tế, dữ liệu video thường được mã hóa dưới dạng base64 hoặc blob và gửi qua WebSocket, sau đó được React hiển thị lên giao diện một cách hiệu quả. React cũng cung cấp các cơ chế tối ưu để xử lý các vấn đề phổ biến trong streaming như buffer overrun, network jitter hay các trường hợp mất kết nối tạm thời.

Trong bài này, giao diện người dùng của hệ thống giao thông thông minh được phát triển dựa trên thư viện ReactJS, một thư viện JavaScript hiện đại cho phép xây dựng các giao diện người dùng tương tác cao với hiệu suất tối ưu. Giao diện này được thiết kế nhằm đáp ứng các yêu cầu quan trọng sau:

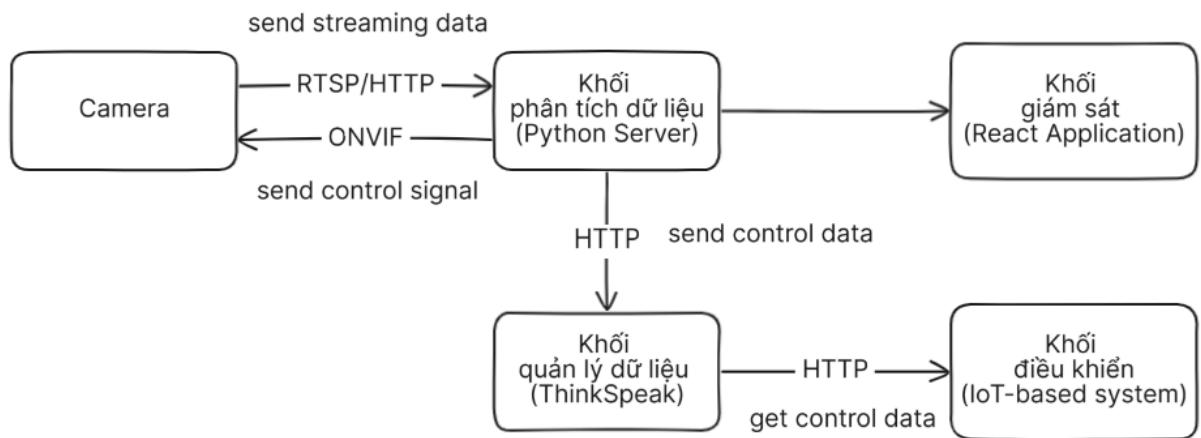
- Hiển thị hình ảnh trực tiếp từ camera giám sát với độ trễ thấp
- Theo dõi và hiển thị các thông số giao thông quan trọng theo thời gian thực
- Điều khiển hướng camera từ xa
- Giám sát trạng thái đèn giao thông và chu kỳ hoạt động
- Hiển thị kết quả đếm xe và dự đoán mật độ giao thông
- Cung cấp thông tin về thời gian trễ và trạng thái kết nối

CHƯƠNG 3: THIẾT KẾ, CÀI ĐẶT

VÀ THỬ NGHIỆM HỆ THỐNG

3.1 Thiết kế tổng quan

3.1.1 Các khối chức năng của hệ thống



Hình 8. Tổng quan các khối chức năng

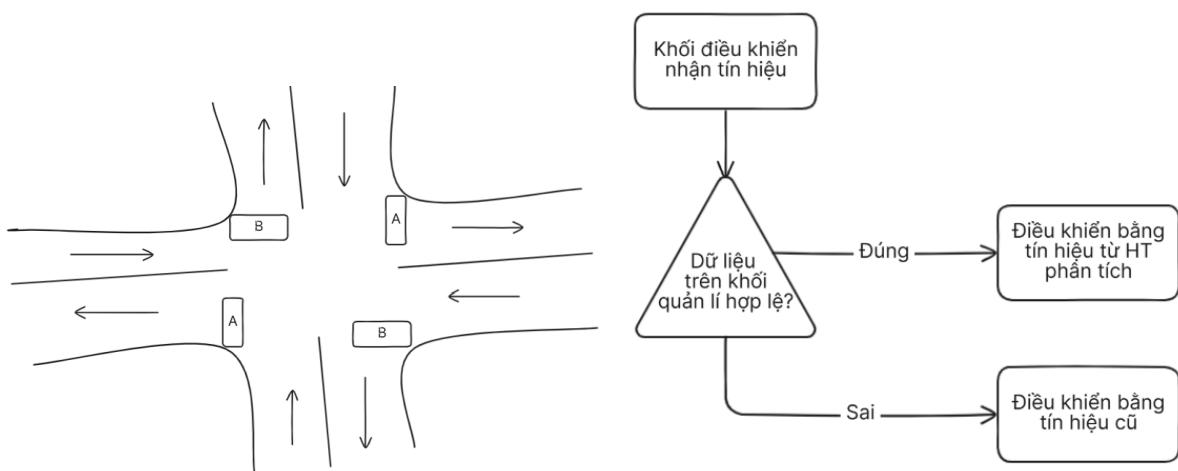
Tổng quan hệ thống điều tiết giao thông sẽ gồm 5 thành phần chính, bao gồm:

- Camera: Thu thập hình ảnh về giao thông trên 2 hướng di chuyển.
- Khối phân tích dữ liệu hay một Python server có chức năng:
 - Điều khiển Camera để lấy dữ liệu từ các hướng thông qua giao thức ONVIF.
 - Lấy thông tin mật độ giao thu thập từ Camera thông qua giao thức RTSP.
 - Sử dụng mô hình YOLO v8 để phân tích lưu lượng giao thông và quyết định thời gian đèn xanh/đỏ trong chu kỳ đèn tiếp theo.
 - Gửi thông tin điều khiển (thời gian đèn xanh/đỏ) lên ThinkSpeak.
- Khối quản lý dữ liệu là bên trung gian sẽ thực hiện gửi/nhận dữ liệu từ khối phân tích dữ liệu và khối điều khiển qua giao thức HTTP và lưu trữ.

- Khối điều khiển là tổng hợp ghép nối các thành phần cứng, lấy dữ liệu từ khói quản lý dữ liệu thông qua giao thức HTTP và thực hiện các điều khiển để hiển thị đèn và bộ đếm thời gian chính xác.
- Khối giám sát có các chức năng:
 - Hiển thị hình ảnh trực tiếp từ camera giám sát với độ trễ thấp
 - Theo dõi và hiển thị các thông số giao thông quan trọng theo thời gian thực
 - Điều khiển hướng camera từ xa
 - Giám sát trạng thái đèn giao thông và chu kỳ hoạt động
 - Hiển thị kết quả đếm xe và dự đoán mật độ giao thông

3.1.2 Một số logic xử lý chính

Cập nhật tín hiệu điều khiển



Hình 9. Mô phỏng các làn đường và logic cập nhật tín hiệu điều khiển

Trong khói điều khiển, để điều tiết hoạt động một tập hợp đèn tín hiệu và bộ đếm thời gian tại 4 làn đường ta cần một luồng tín hiệu điều khiển liên tục và một cơ chế điều khiển và cập nhật để không tạo ra gián đoạn hoặc bất thường.

Dựa trên hình ảnh mô phỏng bên trên (Hình 9), nhận thấy tại một thời điểm nhất định thì luôn có 2/4 làn đường được phép di chuyển (có thể là 2 làn dọc hoặc 2 làn ngang), như vậy cần điều khiển 2 tập hợp đèn và bộ đếm (một tập đại diện cho làn dọc và một tập đại diện cho làn ngang).

Cơ chế điều khiển 2 tập hợp đèn và bộ đếm được dựa trên nguyên lý của đèn giao thông và xác định như bảng sau:

Đèn/Tín hiệu	Xanh	Vàng	Đỏ
Làn dọc (A)	t1	t2	t3
Làn ngang (B)	t3 - t2	t2	t1 + t2

Bảng 3. Phân bổ thời gian sáng của 2 bộ đèn giao thông

Đối với cơ chế cập nhật tín hiệu từ ThinkSpeak, để đảm bảo không gây ra sự gián đoạn và bất thường, khói điều khiển sẽ tiến hành kiểm tra tín hiệu nhận được trước khi thực thi điều khiển:

- Nếu tín hiệu là hợp lệ (các tín hiệu thời gian khác 0), khói điều khiển sẽ điều khiển dựa trên tín hiệu này.
- Nếu tín hiệu là không hợp lệ, khói điều khiển sẽ sử dụng tín hiệu cũ (tín hiệu của chu kì cũ hoặc tín hiệu mặc định) để điều khiển.

Cập nhật bộ đếm thời gian

- Bối cảnh và Mục tiêu:

Hệ thống điều khiển đèn tín hiệu giao thông truyền thống thường hoạt động theo các chu kỳ thời gian cố định, được lập trình sẵn dựa trên các khảo sát lưu lượng định kỳ hoặc kinh nghiệm vận hành. Mặc dù đơn giản và dễ triển khai, phương pháp này bộc lộ nhiều hạn chế trong bối cảnh giao thông đô thị ngày càng phức tạp và biến động. Luồng phương tiện thay đổi liên tục theo thời gian trong ngày, ngày trong tuần, và các sự kiện đột xuất, khiến cho chu kỳ cố định thường xuyên trở nên không hiệu quả: gây lãng phí thời gian xanh cho các hướng vắng xe và kéo dài thời gian chờ đợi không cần thiết cho các hướng đông xe, dẫn đến ùn tắc, tăng tiêu thụ nhiên liệu và ô nhiễm môi trường.

Để khắc phục nhược điểm này, Hệ thống Điều tiết Giao thông Thông minh được đề xuất tích hợp khả năng **điều khiển thích ứng (adaptive control)**, tự động điều chỉnh thời gian pha đèn dựa trên dữ liệu mật độ giao thông được thu thập trong thời gian thực. Trái tim của khả năng thích ứng này nằm ở **thuật toán tối ưu hóa thời gian đèn tín hiệu động**. Mục tiêu chính của thuật toán này không chỉ đơn thuần là phân bổ thời gian xanh tỷ lệ thuận với số lượng xe, mà còn phải cân bằng nhiều yếu tố phức tạp:

- Hiệu quả (Efficiency): Tối đa hóa thông lượng xe qua nút giao, giảm thiểu thời gian chờ đợi trung bình.
- Công bằng (Fairness): Đảm bảo không có hướng nào bị "bỏ quên" hoặc phải chờ đợi quá lâu, ngay cả khi có lưu lượng thấp.
- Ôn định (Stability): Tránh các thay đổi thời gian đèn quá đột ngột và khó lường, duy trì sự ổn định và an toàn cho luồng giao thông.
- Tuân thủ ràng buộc (Constraint Compliance): Hoạt động trong giới hạn vật lý và quy định an toàn (thời gian xanh tối thiểu/tối đa, thời gian vàng cố định).

Mục này sẽ đi sâu phân tích cấu trúc, logic hoạt động, và các khía cạnh vận hành của thuật toán được thiết kế để đáp ứng các mục tiêu trên, sử dụng dữ liệu đầu vào từ mô hình phát hiện phương tiện YOLOv8n.

- **Mô tả Thuật toán và Dữ liệu Vận hành:**

Thuật toán hoạt động theo chu kỳ, thường đồng bộ với chu kỳ đèn giao thông tổng thể hoặc thực hiện tính toán định kỳ sau mỗi N chu kỳ. Nó nhận các dữ liệu đầu vào và sử dụng các tham số cấu hình để đưa ra quyết định về thời gian xanh cho chu kỳ tiếp theo.

+ **Dữ liệu Đầu vào (Input Data):**

- Dữ liệu Đếm Phương tiện Thời gian thực (Real-time Vehicle Counts): Đây là thông tin cốt lõi, được cung cấp bởi mô-đun xử lý ảnh sử dụng YOLOv8n. Để tăng độ tin cậy và giảm ảnh hưởng của nhiễu tức thời, thuật toán thường không chỉ dựa vào một lần đếm duy nhất mà sử dụng dữ liệu từ *nhiều khoảng thời gian đo lường gần nhất* (ví dụ: 2-3 chu kỳ đèn trước đó hoặc các khoảng thời gian cố định ngắn). Việc này giúp làm "tron" dữ liệu đầu vào.

- *Ví dụ cụ thể:* Hệ thống lưu trữ số xe đếm được cho mỗi hướng trong 2 chu kỳ gần nhất:

- counts_direction1 = [15, 17] (Lần đo 1: 15 xe, Lần đo 2: 17 xe)
 - counts_direction2 = [25, 27] (Lần đo 1: 25 xe, Lần đo 2: 27 xe)

- Trạng thái Hiện tại của Hệ thống (Current System State):

- current_green_times: Mảng lưu trữ thời gian đèn xanh đang được áp dụng cho mỗi hướng trong chu kỳ vận hành hiện tại. Thông tin này là tối quan trọng cho cơ chế làm mịn, đảm bảo tính liên tục và ổn định của hệ thống.

■ Ví dụ: `current_green_times = {direction1: 40, direction2: 75}`
 (Hướng 1: 40s, Hướng 2: 75s)

+ Tham số Cấu hình (Configuration Parameters):

Các tham số này được thiết lập trước (có thể hiệu chỉnh sau này) để định nghĩa giới hạn hoạt động và hành vi của thuật toán, đảm bảo an toàn và phù hợp với đặc thù của từng nút giao:

- MIN_GREEN_TIME: Thời gian xanh tối thiểu tuyệt đối (giây). Đảm bảo đủ thời gian cho pha khởi động của dòng xe hoặc cho người đi bộ (nếu có pha riêng). Ví dụ: *20 giây*.
- MAX_GREEN_TIME: Thời gian xanh tối đa tuyệt đối (giây). Ngăn chặn một hướng độc chiếm chu kỳ quá lâu, giới hạn thời gian chờ tối đa của các hướng khác. Ví dụ: *100 giây*.
- TOTAL_CYCLE_TIME: Tổng thời gian dự kiến cho một chu kỳ đèn hoàn chỉnh (giây). Tham số này có thể cố định hoặc tự điều chỉnh trong các thuật toán phức tạp hơn, nhưng trong trường hợp này, giả định là cố định. Ví dụ: *120 giây*.
- YELLOW_TIME: Thời gian đèn vàng cố định cho mỗi pha chuyển tiếp (giây). Đây là tham số an toàn, không thay đổi. Ví dụ: *3 giây*.
- NUMBER_OF_DIRECTIONS (hoặc NUMBER_OF_PHASES): Số lượng hướng (hoặc pha đèn xanh) độc lập trong một chu kỳ. Ví dụ: *2*.
- BASE_ALLOCATION_RATIO: Tỷ lệ phần trăm tối thiểu của tổng thời gian xanh có thể phân bổ mà *mỗi* hướng được đảm bảo nhận, bất kể lưu lượng. Giá trị này (từ 0 đến $< 1/\text{NUMBER_OF_DIRECTIONS}$) thể hiện mức độ "công bằng" cơ bản của hệ thống. Ví dụ: *0.3 (30%)*.

- SMOOTHING_FACTOR: Hệ số làm mịn (còn gọi là hệ số giảm chấn - dampening factor), giá trị từ 0 đến 1. Xác định mức độ thay đổi cho phép giữa chu kỳ trước và chu kỳ mới. Giá trị gần 0 làm thay đổi rất chậm (ổn định cao, phản ứng chậm), giá trị gần 1 làm thay đổi nhanh (phản ứng nhanh, dễ dao động). Nó hoạt động như một bộ lọc thông thấp (low-pass filter) cho tín hiệu điều khiển thời gian xanh. Ví dụ: 0.3.

- Quy trình Xử lý và Tính toán Chi tiết:

Thuật toán thực hiện một chuỗi các bước logic để đi từ dữ liệu đếm xe thô đến thời gian đèn xanh cuối cùng cho chu kỳ tiếp theo:

Bước 1: Tiền xử lý Dữ liệu - Tính Trung bình Số lượng Xe (Calculate Average Vehicle Count):

- Mục đích: Loại bỏ các biến động ngẫu nhiên, tạo ra một chỉ số ổn định hơn đại diện cho nhu cầu giao thông của mỗi hướng trong khoảng thời gian gần đây.
- Thực hiện: Tính giá trị trung bình cộng của các số liệu đếm xe đã lưu trữ cho mỗi hướng. Các phương pháp khác như trung bình trượt có trọng số (Weighted Moving Average - WMA), nơi các lần đếm gần hơn có trọng số cao hơn, cũng có thể được xem xét, nhưng trung bình cộng đơn giản thường đủ hiệu quả và dễ thực hiện.
- Công thức: $\text{direction_avg_count} = \text{sum(counts_direction)} / \text{len(counts_direction)}$
- Ví dụ:
 - $\text{direction1_avg_count} = (15 + 17) / 2 = 16$
 - $\text{direction2_avg_count} = (25 + 27) / 2 = 26$

Bước 2: Xác định Nguồn lực Thời gian - Tính Tổng Thời gian Xanh Có thể Phân bổ (Calculate Total Allocatable Green Time):

- Mục đích: Xác định "ngân sách" thời gian thực tế có thể linh hoạt chia sẻ giữa các hướng, sau khi đã trừ đi các khoảng thời gian cố định không thể thay đổi trong chu kỳ.
- Thực hiện: Lấy tổng thời gian chu kỳ trừ đi tổng thời gian dành cho tất cả các pha đèn vàng. Thời gian đèn đỏ của một hướng được ngầm định bằng tổng thời gian chu kỳ trừ đi thời gian xanh và vàng của chính hướng đó.
- Công thức: $\text{total_allocatable_green_time} = \text{TOTAL_CYCLE_TIME} - (\text{YELLOW_TIME} * \text{NUMBER_OF_DIRECTIONS})$
- Ví dụ: $\text{total_allocatable_green_time} = 120 - (3 * 2) = 114$ giây

Bước 3: Phân bổ Dựa trên Nhu cầu và Công bằng - Tính Tỷ lệ Phân bổ (Calculate Allocation Ratios):

- Mục đích: Đây là bước cốt lõi, xác định tỷ lệ phần trăm của `total_allocatable_green_time` mà mỗi hướng sẽ nhận được. Logic này phải cân bằng giữa việc ưu tiên hướng đông xe hơn và đảm bảo không hướng nào bị đối xử quá bất công.
- Thực hiện:
 - Tính tổng nhu cầu (tổng số xe trung bình): $\text{total_avg_count} = \text{sum}(\text{all_direction_avg_count})$.
 - Tính phần tỷ lệ thời gian có thể phân bổ linh hoạt dựa trên mật độ xe: $\text{variable_allocation_pool_ratio} = 1.0 - (\text{BASE_ALLOCATION_RATIO} * \text{NUMBER_OF_DIRECTIONS})$. Đây là phần trăm thời gian còn lại sau khi đã dành phần tối thiểu cho mỗi hướng.
 - Tính tỷ lệ cuối cùng cho mỗi hướng: Mỗi hướng nhận được phần cơ bản `BASE_ALLOCATION_RATIO`, cộng với một phần của `variable_allocation_pool_ratio` tỷ lệ thuận với số xe trung bình của hướng đó so với tổng số xe trung bình.
- Công thức:
 - $\text{total_avg_count} = 16 + 26 = 42$

- $\text{variable_allocation_pool_ratio} = 1.0 - (0.3 * 2) = 0.4$ (40% thời gian xanh phân bổ theo tỷ lệ xe)
- $\text{direction1_ratio} = \text{BASE_ALLOCATION_RATIO} + \text{variable_allocation_pool_ratio} * (\text{direction1_avg_count} / \text{total_avg_count})$
- $\text{direction1_ratio} = 0.3 + 0.4 * (16 / 42) \approx 0.3 + 0.15238 \approx 0.4524$
- $\text{direction2_ratio} = \text{BASE_ALLOCATION_RATIO} + \text{variable_allocation_pool_ratio} * (\text{direction2_avg_count} / \text{total_avg_count})$
- $\text{direction2_ratio} = 0.3 + 0.4 * (26 / 42) \approx 0.3 + 0.24762 \approx 0.5476$
- $(Kiểm tra tính nhất quán: 0.4524 + 0.5476 \approx 1.0)$

Bước 4: Chuyển đổi Tỷ lệ thành Thời gian - Tính Thời gian Xanh Thô (Calculate Raw Green Time):

- Mục đích: Áp dụng tỷ lệ phân bổ vừa tính được vào tổng thời gian xanh có thể phân bổ để có được thời gian xanh "lý tưởng" cho mỗi hướng nếu chỉ xét đến mật độ và yêu cầu tối thiểu.
- Thực hiện: Nhân `total_allocatable_green_time` với tỷ lệ phân bổ của từng hướng. Kết quả nên được làm tròn (ví dụ: làm tròn đến giây gần nhất) để phù hợp với đơn vị điều khiển đèn.
- Công thức: $\text{raw_green_time_direction} = \text{round}(\text{total_allocatable_green_time} * \text{direction_ratio})$
- Ví dụ:
 - $\text{raw_time1} = \text{round}(114 * 0.4524) = \text{round}(51.5736) = 52$ giây
 - $\text{raw_time2} = \text{round}(114 * 0.5476) = \text{round}(62.4264) = 62$ giây

Bước 5: Đảm bảo Tính Ôn định - Áp dụng Cơ chế Làm mịn (Apply Smoothing):

- Mục đích: Giảm thiểu sự thay đổi đột ngột của thời gian đèn xanh giữa các chu kỳ liên tiếp. Điều này quan trọng để tránh gây bất ngờ cho người lái, giảm nguy

cơ xung đột và ngăn chặn các dao động lớn (oscillations) trong hệ thống điều khiển, tạo ra trải nghiệm giao thông mượt mà hơn.

- Thực hiện: Tính toán thời gian xanh mới như một trung bình có trọng số giữa thời gian xanh của chu kỳ *trước* (current_green_time) và thời gian xanh thô *vừa tính* (raw_green_time). SMOOTHING_FACTOR xác định trọng số của giá trị mới.
 - Công thức: smoothed_green_time = round((current_green_time * (1 - SMOOTHING_FACTOR)) + (raw_green_time * SMOOTHING_FACTOR))
 - Ví dụ: Với SMOOTHING_FACTOR = 0.3.
 - smooth_time1 = round((40 * (1 - 0.3)) + (52 * 0.3)) = round((40 * 0.7) + (52 * 0.3)) = round(28 + 15.6) = round(43.6) = 44 giây
 - smooth_time2 = round((75 * (1 - 0.3)) + (62 * 0.3)) = round((75 * 0.7) + (62 * 0.3)) = round(52.5 + 18.6) = round(71.1) = 71 giây

Bước 6: Đảm bảo Tính Hợp lệ - Kiểm tra và Áp đặt Giới hạn (Apply Constraints / Clamping):

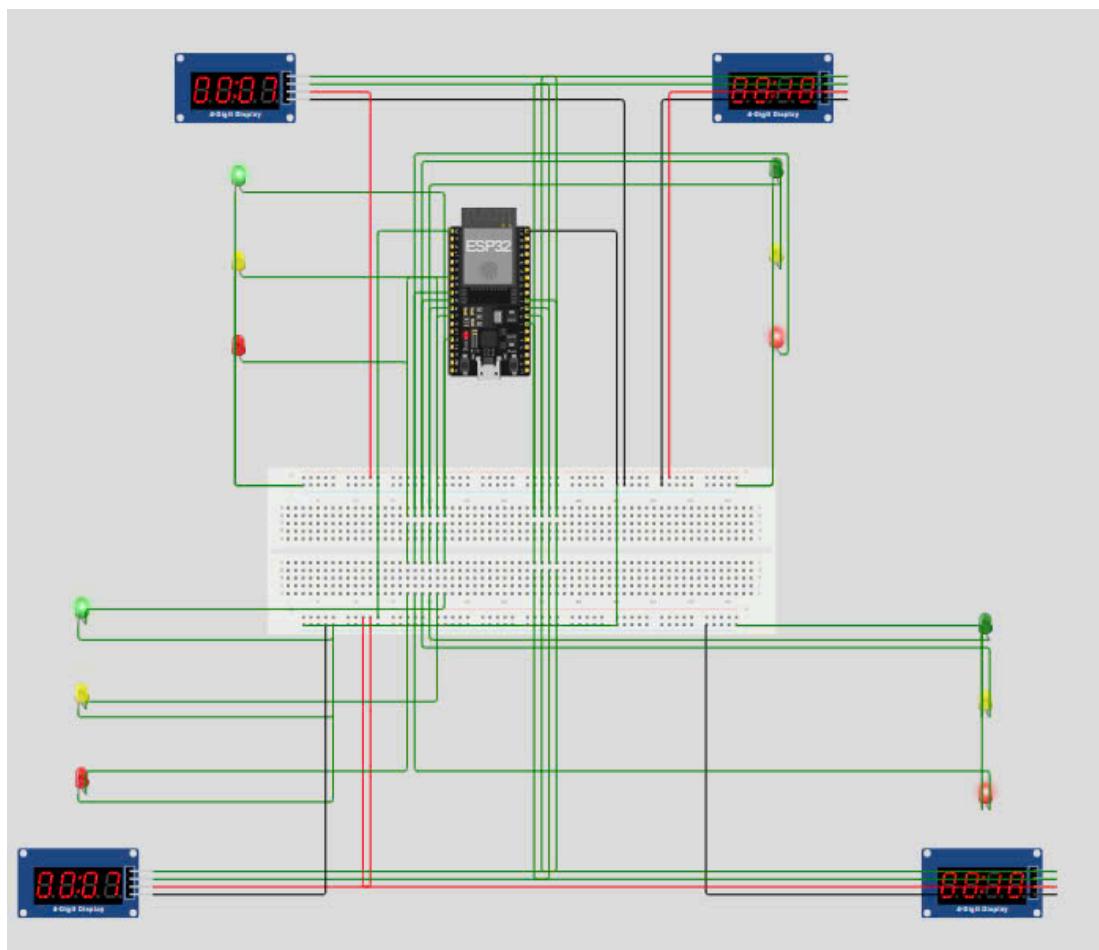
- Mục đích: Đảm bảo rằng thời gian xanh cuối cùng, sau khi đã làm mịn, vẫn nằm trong phạm vi hoạt động an toàn và hợp lý đã được định nghĩa bởi các tham số MIN_GREEN_TIME và MAX_GREEN_TIME.
- Thực hiện: Sử dụng hàm max() và min() để "kẹp" giá trị smoothed_green_time vào giữa hai giới hạn này.
- Công thức: final_green_time = max(MIN_GREEN_TIME, min(MAX_GREEN_TIME, smoothed_green_time))
- Ví dụ:
 - final_time1 = max(20, min(100, 44)) = max(20, 44) = 44 giây
 - final_time2 = max(20, min(100, 71)) = max(20, 71) = 71 giây

Bước 7: Hoàn tất Chu kỳ - Cập nhật và Lưu trữ Kết quả (Update and Store Results):

- Mục đích: Đưa thời gian xanh mới tính toán vào sử dụng cho chu kỳ đèn tiếp theo và lưu trữ chúng để sử dụng làm current_green_time cho lần tính toán kế tiếp.
- Thực hiện: Cập nhật cấu hình thời gian xanh của hệ thống điều khiển đèn (gửi lệnh đến ESP32) và cập nhật biến trạng thái nội bộ của thuật toán.
- Kết quả cuối cùng của ví dụ:
 - Thời gian xanh mới cho Hướng 1: 44 giây (tăng 4 giây so với 40 giây cũ)
 - Thời gian xanh mới cho Hướng 2: 71 giây (giảm 4 giây so với 75 giây cũ)

3.2 Thiết kế chi tiết

3.2.1 Chi tiết module điều khiển



Hình 10. Mô phỏng ghép nối phần cứng của hệ thống điều khiển

Thiết bị 1	Thiết bị 2	Chức năng	Chú thích
ESP32	TM1637 (1)	Điều khiển bộ đếm (1)	Bộ đếm (3) ứng với TM1637 (3) sẽ có các chân tín hiệu được nối tương ứng song song với bộ đếm (1)
3V3	VCC	Cấp nguồn cho TM1637 (1)	
GND	GND	Nối đất	
GPIO2	DIO	Truyền dữ liệu điều khiển từ ESP32	
GPIO4	CLK	Đồng bộ xung nhịp	
ESP32	TM1637 (2)	Điều khiển bộ đếm (2)	Bộ đếm (4) ứng với TM1637 (4) sẽ có các chân tín hiệu được nối tương ứng song song với bộ đếm (2)
3V3	VCC	Cấp nguồn cho TM1637 (2)	
GND	GND	Nối đất	
GPIO5	DIO	Truyền dữ liệu điều khiển từ ESP32	
GPIO13	CLK	Đồng bộ xung nhịp	
ESP32	LED xanh (1)	Điều khiển đèn xanh của bộ đèn (1)	Bộ đèn (3) gồm LED xanh/ vàng/ đỏ (3) sẽ có các chân tín hiệu được nối tương ứng song song với bộ đèn (1)
GNU	Chân âm	Nối đất	
GPIO14	Chân dương	Truyền dữ liệu điều khiển từ ESP32	
ESP32	LED vàng (1)	Điều khiển đèn vàng của bộ đèn (1)	
GND	Chân âm	Nối đất	
GPIO13	Chân dương	Truyền dữ liệu điều khiển từ ESP32	Bộ đèn (4) gồm LED xanh/ vàng/ đỏ (4) sẽ có các chân tín hiệu được nối tương ứng song song với bộ đèn (2)
ESP32	LED đỏ (1)	Điều khiển đèn đỏ của bộ đèn (1)	
GND	Chân âm	Nối đất	
GPIO32	Chân dương	Truyền dữ liệu điều khiển từ ESP32	
ESP32	LED xanh (2)	Điều khiển đèn xanh của bộ đèn (2)	
GND	Chân âm	Nối đất	Bộ đèn (4) gồm LED xanh/ vàng/ đỏ (4) sẽ có các chân tín hiệu được nối tương ứng song song với bộ đèn (2)
GPIO27	Chân dương	Truyền dữ liệu điều khiển từ ESP32	
ESP32	LED vàng (2)	Điều khiển đèn vàng của bộ đèn (2)	
GND	Chân âm	Nối đất	

GPIO26	Chân dương	Truyền dữ liệu điều khiển từ ESP32	
ESP32	LED đỏ (2)	Điều khiển đèn đỏ của bộ đèn (2)	
GND	Chân âm	Nối đất	
GPIO25	Chân dương	Truyền dữ liệu điều khiển từ ESP32	

Bảng 4. Bảng đấu nối các chân của các thành phần phân cứng

3.2.2 Chi tiết module xử lý dữ liệu

3.2.2.1. Giới thiệu:

Module xử lý dữ liệu đóng vai trò là trung tâm thần kinh của Hệ thống Điều tiết Giao thông Thông minh. Nó nhận dữ liệu đầu vào từ các cảm biến (qua module phát hiện đối tượng YOLOv8) và trạng thái hệ thống, sau đó thực hiện các phép tính toán, phân tích và logic cần thiết để đưa ra quyết định điều khiển tối ưu cho đèn tín hiệu. Chức năng cốt lõi của module này là chuyển đổi số liệu đếm xe thành thời gian biểu đèn xanh/đỏ linh hoạt, thích ứng với mật độ giao thông thực tế, nhằm mục tiêu giảm thiểu tắc và tối ưu hóa luồng phương tiện.

Các nhiệm vụ chính bao gồm:

- Thu thập và Tổng hợp Dữ liệu Đếm: Nhận diện và đếm chính xác phương tiện cho từng hướng trong các khoảng thời gian đo lường cụ thể, đảm bảo không đếm trùng.
- Tính toán Thống kê Giao thông: Xử lý dữ liệu đếm để tính toán các chỉ số như số lượng xe trung bình, mật độ tương đối.
- Áp dụng Thuật toán Điều khiển: Thực thi logic tính toán thời gian đèn xanh tối ưu, cân bằng giữa việc ưu tiên luồng đông và đảm bảo tính công bằng, ổn định.
- Quản lý Chu trình Hệ thống: Điều phối các pha hoạt động: đo lường tại các hướng, tính toán dựa trên kết quả đo, và trì hoãn trước khi áp dụng cấu hình mới.

- Giao tiếp và Lưu trữ: Gửi dữ liệu trạng thái, thống kê đến giao diện người dùng (qua WebSocket) và lưu trữ/gửi dữ liệu quan trọng đến các hệ thống bên ngoài (ThingSpeak).

3.2.2.2. Các Cấu trúc Dữ liệu Chính

Module này quản lý trạng thái và dữ liệu thông qua các biến và cấu trúc dữ liệu toàn cục quan trọng:

- vehicle_counting_data: Cấu trúc cốt lõi lưu trữ kết quả đếm xe. Nó chứa số lượng xe (cycle_counts) và tập hợp các ID xe đã đếm (cycle_ids) cho mỗi hướng (direction1, direction2) và cho từng lần đo trong một chu kỳ lớn (lần 1, lần 2). Ví dụ: {'cycle_counts': {'direction1': [15, 17], 'direction2': [25, 27]}, 'cycle_ids': {'direction1': [{id1, id2}, {id3, id4}], 'direction2': [...]}}. Dữ liệu này là đầu vào trực tiếp cho thuật toán tính toán thời gian đèn.
- cycle_control: Quản lý trạng thái chu trình tổng thể. Lưu pha hiện tại (measuring, calculating, delaying), trạng thái trì hoãn (in_delay_mode, delay_remaining), thời gian đèn xanh mặc định, thời gian chu kỳ tổng, và quan trọng nhất là next_session_config chứa thời gian đèn xanh đã được tính toán và sẵn sàng áp dụng cho phiên tiếp theo.
- traffic_stats: Lưu trữ dữ liệu thống kê để hiển thị. Bao gồm số xe nhìn thấy tức thời (vehicles), mật độ (density), lịch sử đo lường (history), và thống kê loại xe (vehicle_types) cho mỗi hướng.
- traffic_lights: Trạng thái đèn tín hiệu hiện tại. Lưu trạng thái (green, yellow, red) và thời gian còn lại (time_left) cho mỗi hướng.
- traffic_control_settings: Tham số cấu hình của thuật toán điều khiển. Chứa các giới hạn (min_green_time, max_green_time), thời gian vàng (yellow_time), và thời gian xanh hiện đang được áp dụng hoặc mặc định (direction1_green_time, direction2_green_time).
- camera_cycle: Quản lý hoạt động của camera PTZ. Bao gồm thời gian đứng yên tại mỗi vị trí (fixed_position_time), bước hiện tại trong chu trình quay

(current_cycle_step), tổng số bước (total_cycles), và trạng thái hoạt động/đo lường (active, measurement_active).

3.2.2.3. Quy trình Xử lý Dữ liệu và Ra Quyết định:

Hệ thống vận hành theo một chu trình logic gồm các pha nối tiếp nhau:

- **Pha 1: Đo Lường (Measuring Phase)**

- Camera di chuyển tuần tự đến các vị trí đã định (Hướng 1 -> Hướng 2 -> Hướng 1 -> Hướng 2).
- Tại mỗi vị trí, camera dừng lại trong fixed_position_time giây.
- Trong thời gian này, detector.detect() liên tục phân tích video và trả về danh sách các phương tiện (vehicle_detections).
- update_traffic_stats() cập nhật các chỉ số hiển thị tức thời.
- Quá trình này diễn ra cho đến khi hoàn thành đủ total_cycles (thường là 4 lần dừng).

- **Pha 2: Tính toán (Calculating Phase)**

- Nó thực hiện thuật toán phân bổ thời gian xanh (tính trung bình, áp dụng tỷ lệ cơ bản, phân bổ theo tỷ lệ xe, làm mịn, áp đặt giới hạn min/max).
- Kết quả thời gian xanh tối ưu cho chu kỳ tiếp theo được lưu vào cycle_control["next_session_config"].
- Dữ liệu kết quả (thời gian xanh, thời gian đỏ tương ứng) được gửi lên ThingSpeak.

- **Pha 3: Trì hoãn (Delaying Phase)**

- Hệ thống chuyển sang trạng thái in_delay_mode.
- Hoạt động đo lường và di chuyển camera tạm dừng.
- **Quan trọng:** Tất cả các cấu trúc dữ liệu liên quan đến đếm xe (vehicle_counting_data, cumulative_counts, bộ đếm nội bộ của detector) được reset để chuẩn bị cho một chu kỳ đo lường hoàn toàn mới.
- Hệ thống chờ hết khoảng thời gian delay_after_reset.

- **Pha 4: Áp dụng Cấu hình Mới và Bắt đầu lại Chu kỳ**

- Khi hết thời gian trì hoãn (kiểm tra bởi check_delay_status hoặc trong camera_control_thread_function), hệ thống thoát khỏi in_delay_mode.
- Thời gian đèn xanh mới từ cycle_control["next_session_config"] được **chuyển vào**. Từ thời điểm này, hàm update_traffic_lights sẽ sử dụng thời gian xanh mới này để điều khiển đèn.
- Hoạt động đo lường và di chuyển camera được kích hoạt trở lại, bắt đầu một chu kỳ mới từ Pha 1.

3.2.2.4. Phân tích Chi tiết các Hàm Xử lý Chính

```

def track_and_count_vehicles(direction, current_detections):
    """Theo dõi và đếm xe với cơ chế chống đếm trùng tốt hơn"""
    global cumulative_counts, vehicle_counting_data # Đảm bảo sử dụng biến toàn cục

    direction_key = f"direction{direction+1}"
    # Xác định đang ở lần đếm thứ mấy (0 hoặc 1) của hướng hiện tại
    # Dựa vào bước hiện tại của chu trình quay camera
    direction_index = 0 if direction == 0 else 1 # Xác định index của hướng (0 cho direction1, 1 cho direction2)
    cycle_index = 0 if camera_cycle["current_cycle_step"] < 2 else 1 # Xác định index lần đếm (0 cho lần 1, 1 cho lần 2)

    # Lấy set các ID đã đếm của lần đếm này
    # Khởi tạo cấu trúc nếu chưa có (an toàn khi chạy lần đầu)
    if direction_key not in vehicle_counting_data["cycle_ids"]:
        vehicle_counting_data["cycle_ids"][direction_key] = [set(), set()]
    # unique_ids là tập hợp các ID xe đã được đếm trong lần đo này (ví dụ: Hướng 1, Lần 2)
    unique_ids = vehicle_counting_data["cycle_ids"][direction_key][cycle_index]

    count_line_crossed = 0 # Đếm số xe mới vượt vạch trong frame này
    current_vehicles_in_frame = 0 # Đếm tổng số xe nhìn thấy trong frame

    # Duyệt qua tất cả các phương tiện được phát hiện trong frame hiện tại
    for detection in current_detections:
        vehicle_id = detection['id'] # Lấy ID duy nhất của xe từ tracker
        current_vehicles_in_frame += 1 # Tăng bộ đếm số xe trong frame

        # Điều kiện đếm: Xe phải vượt qua vạch (detection['crossed_line'] is True)
        # VÀ ID của xe chưa có trong tập hợp unique_ids của lần đo này
        if detection['crossed_line'] and vehicle_id not in unique_ids:
            # Nếu thỏa mãn điều kiện, thêm ID vào tập hợp để không đếm lại
            unique_ids.add(vehicle_id)

            # Cập nhật số lượng xe đếm được vào cấu trúc dữ liệu chính xác
            # Khởi tạo nếu cần
            if direction_key not in vehicle_counting_data["cycle_counts"]:
                vehicle_counting_data["cycle_counts"][direction_key] = [0, 0]
            # Tăng bộ đếm cho hướng và lần đo tương ứng
            vehicle_counting_data["cycle_counts"][direction_key][cycle_index] += 1
            # Tăng bộ đếm số xe mới vượt vạch trong frame này
            count_line_crossed += 1

            # Ghi log chi tiết về việc đếm xe
            logger.info(f"Đã đếm xe {vehicle_id} ở hướng {direction+1}, lần {cycle_index+1}. " +
                       f"Tổng: {vehicle_counting_data['cycle_counts'][direction_key][cycle_index]}")

    # Trả về số xe nhìn thấy trong frame và số xe mới được đếm khi vượt vạch
    # total_cumulative_count có thể không cần thiết nếu logic chỉ dựa vào vehicle_counting_data
    total_cumulative_count = cumulative_counts[direction_key]['total_vehicles']
    return current_vehicles_in_frame, total_cumulative_count, count_line_crossed

```

- **Giải thích:** Hàm này là cốt lõi của việc đếm xe chính xác cho từng lần đo. Nó nhận danh sách xe được phát hiện trong một frame. Quan trọng nhất, nó sử dụng vehicle_id (từ bộ theo dõi YOLO) và trạng thái crossed_line để xác định khi nào một xe *lần đầu tiên* vượt qua vạch đếm. Để tránh đếm trùng trong cùng một lần đo (ví dụ, xe dừng lại ngay sau vạch và vẫn được phát hiện), nó duy trì

một set các unique_ids cho từng lần đo của từng hướng. Chỉ khi một xe vượt vạch và ID của nó chưa có trong set này, bộ đếm vehicle_counting_data["cycle_counts"] mới được tăng lên và ID được thêm vào set. Điều này đảm bảo mỗi xe chỉ được đếm một lần duy nhất cho mỗi làn camera dừng tại một hướng.

```

● ● ●

def calculate_traffic_light_times():
    """Tính toán thời gian đèn giao thông cân bằng và ổn định hơn"""
    global cycle_control, traffic_control_settings, vehicle_counting_data

    # --- 1. Lấy và Tính Trung bình Dữ liệu Đếm ---
    try:
        # Truy cập trực tiếp vào dữ liệu đếm đã được tích lũy chính xác
        direction1_count1 = vehicle_counting_data["cycle_counts"]["direction1"][0]
        direction1_count2 = vehicle_counting_data["cycle_counts"]["direction1"][1]
        direction2_count1 = vehicle_counting_data["cycle_counts"]["direction2"][0]
        direction2_count2 = vehicle_counting_data["cycle_counts"]["direction2"][1]
        # Tính trung bình để làm mượt dữ liệu, giảm ảnh hưởng của biến động ngắn hạn
        direction1_avg_count = (direction1_count1 + direction1_count2) / 2
        direction2_avg_count = (direction2_count1 + direction2_count2) / 2
    except (KeyError, IndexError) as e:
        # Xử lý trường hợp dữ liệu chưa đủ (ví dụ: hệ thống mới khởi động)
        logger.error(f"Lỗi lấy dữ liệu đếm xe: {e}. Sử dụng giá trị mặc định.")
        direction1_avg_count = 1 # Gán giá trị nhỏ để tránh lỗi chia cho 0
        direction2_avg_count = 1
    # Ghi log số liệu đầu vào và kết quả trung bình
    logger.info(f"Đếm xe chỉ định: Hướng 1 = {direction1_count1}, {direction1_count2}, " +
               f"Hướng 2 = {direction2_count1}, {direction2_count2}")
    logger.info(f"Trung bình xe: Hướng 1 = {direction1_avg_count:.1f}, Hướng 2 = {direction2_avg_count:.1f}")

    # --- 2. Xác định Tham số và Ngân sách Thời gian ---
    min_time = traffic_control_settings["min_green_time"] # Thời gian xanh tối thiểu
    max_time = traffic_control_settings["max_green_time"] # Thời gian xanh tối đa
    # Tổng thời gian có thể phân bổ cho đèn xanh (trừ đi thời gian vàng cố định)
    total_green_time = cycle_control["total_cycle_time"] - (traffic_control_settings["yellow_time"] * 2)

    # --- 3. Tính Tỷ lệ Phân bổ (Cân bằng Hiệu quả và Công bằng) ---
    total_count = direction1_avg_count + direction2_avg_count
    if total_count < 1: # Trường hợp không có xe nào được đếm
        direction1_ratio = direction2_ratio = 0.5 # Chia đều 50:50
    else:
        # Áp dụng tỷ lệ cơ bản (Fairness): Đảm bảo mỗi hướng nhận ít nhất % thời gian nhất định
        base_ratio = 0.3 # Ví dụ: Mỗi hướng nhận ít nhất 30%
        # Phản thời gian còn lại sẽ phân bổ dựa trên tỷ lệ xe (Efficiency)
        variable_ratio = 1.0 - (base_ratio * 2)
        # Tính tỷ lệ cuối cùng: Phản cơ bản + Phản biến đổi tỷ lệ với số xe
        direction1_ratio = base_ratio + variable_ratio * (direction1_avg_count / total_count)
        direction2_ratio = base_ratio + variable_ratio * (direction2_avg_count / total_count)
        # Chuẩn hóa lại tỷ lệ để đảm bảo tổng là 1 (quan trọng nếu dùng số thực)
        total_ratio = direction1_ratio + direction2_ratio
        if total_ratio > 0:
            direction1_ratio /= total_ratio
            direction2_ratio /= total_ratio

    # --- 4. Tính Thời gian Xanh Thô (Dựa trên tỷ lệ) ---
    raw_time1 = int(total_green_time * direction1_ratio)
    # Tính raw_time2 từ phần còn lại để đảm bảo tổng chính xác
    raw_time2 = total_green_time - raw_time1

    # --- 5. Áp dụng Lọc mịn (Smoothing - Đảm bảo Ông định) ---
    # Lấy thời gian xanh của chu kỳ trước để làm cơ sở so sánh
    prev_time1 = traffic_control_settings.get("direction1_green_time",
                                              cycle_control["default_green_time"])
    prev_time2 = traffic_control_settings.get("direction2_green_time",
                                              cycle_control["default_green_time"])
    # Hệ số làm mịn: Kiểm soát mức độ thay đổi so với chu kỳ trước
    smooth_factor = 0.5 # Ví dụ: 50% dựa trên tính toán mới, 50% giữ lại từ giá trị cũ
    # Tính thời gian đã làm mịn
    smooth_time1 = int(prev_time1 * (1 - smooth_factor) + raw_time1 * smooth_factor)
    # Tính smooth_time2 sao cho tổng vẫn là total_green_time
    smooth_time2 = total_green_time - smooth_time1

    # --- 6. Áp đặt Giới hạn Min/Max (Đảm bảo An toàn và Hợp lý) ---
    # Kẹp giá trị smooth_time1 vào khoảng [min_time, max_time]
    final_time1 = max(min_time, min(max_time, smooth_time1))
    # Tính lại thời gian còn lại cho hướng 2 sau khi đã kẹp hướng 1
    remaining_time_for_dir2 = total_green_time - final_time1
    # Kẹp giá trị thời gian còn lại cho hướng 2 vào khoảng [min_time, max_time]
    final_time2 = max(min_time, min(max_time, remaining_time_for_dir2))
    # Kiểm tra lại: Nếu tổng vượt quá do cả 2 cùng bị kẹp ở min_time, cần điều chỉnh (ít xảy ra nếu min_time đủ nhỏ)
    # Hoặc đảm bảo rằng final_time1 cũng được điều chỉnh nếu final_time2 bị kẹp
    final_time1 = total_green_time - final_time2 # Điều chỉnh lại final_time1 để đảm bảo tổng chính xác

    # --- 7. Lưu Kết quả và Gửi Dữ liệu ---
    # Lưu thời gian xanh mới vào cấu hình chờ áp dụng
    cycle_control["next_session_config"]["direction1_green_time"] = final_time1
    cycle_control["next_session_config"]["direction2_green_time"] = final_time2
    cycle_control["next_session_config"]["ready"] = True # Đánh dấu đã sẵn sàng

    # Tính thời gian đỏ tương ứng (ví dụ: thời gian đỏ của hướng 1)
    # Băng thời gian xanh hướng 2 + vàng hướng 2 (hoặc cả 2 vàng nếu logic yêu cầu)
    red_time_direction1 = final_time1 + traffic_control_settings["yellow_time"]

    # Gửi dữ liệu lên ThingSpeak
    send_data_to_thingspeak(final_time1, red_time_direction1)

    # Ghi log kết quả cuối cùng
    logger.info(f"Thời gian đèn mới được tính toán: Hướng 1 = {final_time1}s, Hướng 2 = {final_time2}s")
    return final_time1, final_time2

```

- **Giải thích:** Đây là thuật toán cốt lõi để điều chỉnh thời gian đèn. Nó thực hiện một quy trình nhiều bước:

- Tổng hợp dữ liệu: Lấy kết quả đêm chính xác từ vehicle_counting_data và tính trung bình để ổn định số liệu.
- Xác định ngân sách: Tính tổng thời gian có thể dùng cho đèn xanh sau khi trừ đi thời gian vàng cố định.
- Phân bổ tỷ lệ: Quyết định mỗi hướng được bao nhiêu phần trăm thời gian xanh. Logic này kết hợp:
 1. base_ratio: Đảm bảo mỗi hướng có một phần tối thiểu (công bằng).
 2. Phần còn lại (variable_ratio): Chia tỷ lệ thuận với lượng xe trung bình (hiệu quả).
- Tính thời gian thô: Áp dụng tỷ lệ vào tổng thời gian xanh.
- Làm mịn: Trộn thời gian thô với thời gian của chu kỳ trước (prev_time) bằng smooth_factor. Điều này ngăn thời gian đèn thay đổi quá nhanh, gây bất ổn cho giao thông. Giá trị smooth_factor = 0.5 nghĩa là kết quả mới là trung bình của giá trị cũ và giá trị mới tính toán. Logic đảm bảo tổng thời gian xanh sau khi làm mịn vẫn bằng total_green_time.
- Áp đặt giới hạn: Đảm bảo thời gian xanh cuối cùng không nhỏ hơn min_time và không lớn hơn max_time. Logic điều chỉnh cả hai final_time để đảm bảo tổng của chúng không vượt quá total_green_time sau khi áp đặt giới hạn.
- Lưu trữ và Gửi: Lưu kết quả vào next_session_config để chờ áp dụng và gửi lên ThingSpeak.



```
def start_delay_phase():
    """Bắt đầu pha delay và reset dữ liệu đếm"""
    global cycle_control, camera_cycle, vehicle_counting_data, detector, cumulative_counts

    current_time = time.time()
    # Kích hoạt chế độ delay
    cycle_control["in_delay_mode"] = True
    # Đặt thời điểm kết thúc delay
    cycle_control["delay_end_time"] = current_time + cycle_control["delay_after_reset"]
    # Đặt lại pha hiện tại
    cycle_control["current_phase"] = "delaying"
    # Ghi lại thời điểm bắt đầu delay/reset
    cycle_control["last_reset_time"] = current_time

    # Tạm dừng hoạt động chính của camera: không di chuyển, không đo lường
    camera_cycle["active"] = False
    camera_cycle["measurement_active"] = False

    # --- RESET CÁC BỘ ĐÉM ---
    # Reset cấu trúc dữ liệu đếm chính cho chu kỳ mới
    vehicle_counting_data = {
        "cycle_counts": {"direction1": [0, 0], "direction2": [0, 0]}, # Số xe về 0
        "cycle_ids": {"direction1": [set(), set()], "direction2": [set(), set()]}, # Tập ID rỗng
        "current_cycle_index": 0 # Reset chỉ số (mặc dù không dùng nhiều)
    }

    # Reset các bộ đếm phụ khác (nếu còn sử dụng)
    for direction_key in ["direction1", "direction2"]:
        cumulative_counts[direction_key]["total_vehicles"] = 0
        cumulative_counts[direction_key]["counted_ids"] = set()

    # Gọi hàm reset của đối tượng detector (quan trọng để reset tracked_objects, counted_ids nội bộ)
    if detector:
        detector.reset_counter()

    # Ghi log xác nhận
    logger.info(f"Bắt đầu pha delay {cycle_control['delay_after_reset']} giây. Đã reset bộ đếm.")
    return True
```

- **Giải thích:** Hàm này được gọi sau khi calculate_traffic_light_times hoàn thành. Mục đích chính là tạo một khoảng nghỉ và quan trọng nhất là reset toàn bộ trạng thái đếm để chuẩn bị cho chu kỳ đo lường tiếp theo.
 - Kích hoạt Delay: Đặt in_delay_mode thành True, cập nhật pha thành delaying, và tính toán thời điểm kết thúc delay.
 - Dừng Camera: Tắt cờ active và measurement_active để ngăn camera di chuyển và ngăn hệ thống cập nhật thống kê đếm xe trong thời gian delay.
 - Reset Bộ đếm: Đây là bước cực kỳ quan trọng. Nó xóa sạch dữ liệu trong vehicle_counting_data (cả cycle_counts và cycle_ids), các bộ đếm phụ như cumulative_counts, và gọi phương thức reset_counter() của YOLOv8Detector để xóa trạng thái theo dõi và ID đã đếm bên trong đối

tượng detector. Việc reset triệt để này đảm bảo chu kỳ đo lường tiếp theo bắt đầu với số liệu hoàn toàn mới.

3.2.2.5. Các Hàm Hỗ trợ Khác

- update_traffic_stats(): Cập nhật các biến traffic_stats dùng cho hiển thị (số xe tức thời, mật độ, lịch sử). Lưu ý quan trọng là hàm này không thực hiện việc đếm chính cho thuật toán mà chỉ lấy kết quả từ track_and_count_vehicles và tính toán các chỉ số phụ. Nó cũng kiểm tra in_delay_mode để không cập nhật số liệu tích lũy trong thời gian trì hoãn.
- reset_counting_data(): Một hàm tiện ích (có thể được gọi bởi start_delay_phase) để reset vehicle_counting_data.
- check_delay_status(): Kiểm tra xem thời gian delay đã kết thúc chưa và nếu có thì thực hiện các hành động cần thiết để thoát khỏi chế độ delay và áp dụng cấu hình đèn mới.
- update_traffic_lights(): Mô phỏng hoặc cập nhật trạng thái đèn (time_left) dựa trên thời gian xanh/vàng/đỏ hiện tại được lưu trong traffic_control_settings.
- send_data_to_thingspeak(): Gửi dữ liệu thời gian đèn đã tính toán lên nền tảng IoT ThingSpeak.

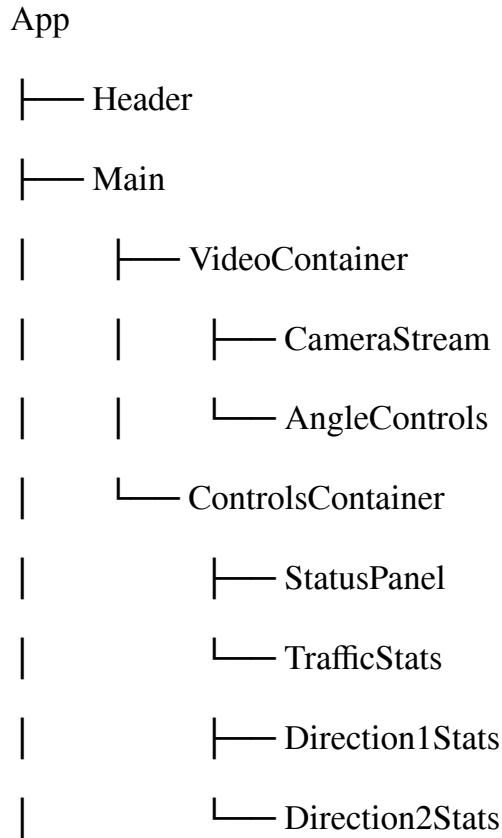
3.2.2.6. Kết luận

Module xử lý dữ liệu và ra quyết định điều khiển là bộ não của hệ thống giao thông thông minh. Nó tích hợp chặt chẽ việc thu thập dữ liệu đếm chính xác từ cảm biến thị giác, áp dụng một thuật toán điều khiển thích ứng có tính đến hiệu quả, công bằng và ổn định, đồng thời quản lý các trạng thái hoạt động của hệ thống một cách tuần tự và logic. Các cấu trúc dữ liệu được thiết kế cẩn thận (vehicle_counting_data) và các hàm xử lý chính (track_and_count_vehicles, calculate_traffic_light_times, start_delay_phase) phối hợp với nhau để tạo ra khả năng tự động điều chỉnh thời gian đèn tín hiệu, đáp ứng linh hoạt với sự biến động của luồng giao thông thực tế.

3.2.3. Chi tiết module giám sát

3.2.3.1. Kiến trúc tổng thể

Ứng dụng được xây dựng theo mô hình single-page application (SPA) với một thành phần chính là App.js. Giao diện được chia thành các phần chức năng riêng biệt để dễ dàng quản lý và bảo trì:



3.2.3.2. Quản lý state và tối ưu hóa hiệu suất

Một trong những điểm nổi bật của ứng dụng này là cách tiếp cận tối ưu trong việc quản lý state và rendering để đảm bảo hiệu suất cao:



```
// Giảm thiểu số lượng state để tránh re-render
const [connected, setConnected] = useState(false);
const [streamDelay, setStreamDelay] = useState(0);

const wsRef = useRef(null);
const videoRef = useRef(null);
const reconnectTimerRef = useRef(null);
const systemStateRef = useRef({
    // Dữ liệu hệ thống
});
```

Việc sử dụng useRef thay vì useState cho dữ liệu không trực tiếp ảnh hưởng đến UI giúp tránh các lần render không cần thiết, đồng thời vẫn đảm bảo các cập nhật dữ liệu được lưu trữ một cách nhất quán.

3.2.3.3. Hệ thống tham chiếu DOM trực tiếp

Để tối ưu hiệu suất, ứng dụng sử dụng cơ chế tham chiếu DOM trực tiếp thông qua useRef để cập nhật UI thay vì dựa vào cơ chế re-rendering của React:



```
// DOM references cho cập nhật trực tiếp
const timeDisplayRef = useRef(null);
const direction1VehiclesRef = useRef(null);
const direction1DensityRef = useRef(null);
```

3.2.3.4 Kết nối và truyền dữ liệu thời gian thực

Ứng dụng sử dụng WebSocket để kết nối với máy chủ backend, cho phép truyền dữ liệu theo thời gian thực với độ trễ thấp:

```
const connectWebSocket = () => {
    const protocol = window.location.protocol === 'https:' ? 'wss:' :
    'ws';
    const host = window.location.hostname || 'localhost';
    const port = 8000;

    console.log(`Connecting to ${protocol}//${host}:${port}/ws`);
    const ws = new WebSocket(`${protocol}//${host}:${port}/ws`);

    // Tối ưu hóa kết nối WebSocket để giảm độ trễ
    ws.binaryType = "arraybuffer";

    // Xử lý các sự kiện kết nối
};
```

3.2.3.5. Xử lý dữ liệu nhận được

Dữ liệu từ WebSocket được phân loại và xử lý theo hai loại chính:

- Frame data: Dữ liệu hình ảnh từ camera
- Stats data: Thống kê và trạng thái hệ thống

```
ws.onmessage = (event) => {
  try {
    const data = JSON.parse(event.data);

    if (data.type === 'frame') {
      // Xử lý frame mới
      if (frameQueueRef.current.length < MAX_QUEUE_LENGTH) {
        frameQueueRef.current.push(data);
      } else {
        frameQueueRef.current = [data];
      }
    } else if (data.type === 'stats') {
      // Cập nhật dữ liệu thống kê
      systemStateRef.current = {
        // Cập nhật state
      };

      // Cập nhật UI từ references
      updateUIFromRefs();
    }
  } catch (error) {
    console.error('Error parsing message:', error);
  }
};
```

3.2.4. Chi tiết module điều khiển

3.2.4.1. Giới thiệu

Module Điều khiển Cục bộ ESP32 đóng vai trò là bộ phận thực thi cấp thấp trong Hệ thống Điều tiết Giao thông Thông minh. Sau khi Khởi Xử lý Trung tâm (chạy script app.py) phân tích dữ liệu từ camera, tính toán thời gian đèn tối ưu và gửi kết quả lên nền tảng đám mây (ThingSpeak), module ESP32 này có nhiệm vụ:

- Kết nối vào mạng Wifi cục bộ.
- Định kỳ truy vấn dữ liệu thời gian đèn mới nhất từ API của ThingSpeak.
- Điều khiển trực tiếp trạng thái bật/tắt của các đèn LED mô phỏng đèn tín hiệu giao thông cho hai hướng chính theo đúng chu kỳ và thời gian đã nhận được.
- Hiển thị thời gian đêm ngược còn lại cho mỗi pha đèn trên các màn hình LED 7 đoạn tương ứng.

Module này hoạt động độc lập trong việc thực thi chu kỳ đèn dựa trên dữ liệu nhận được, giảm tải cho bộ xử lý trung tâm và đảm bảo hoạt động ổn định của mô hình vật lý.

3.2.4.2. Phần cứng Sử dụng

Module này được xây dựng dựa trên các thành phần phần cứng sau:

- Vi điều khiển: Board mạch ESP32 (chưa rõ model cụ thể, nhưng có khả năng kết nối Wi-Fi).
- Đèn LED Tín hiệu: Sáu (6) đèn LED đơn màu, được kết nối với các chân GPIO cụ thể của ESP32 (GPIO 13, 14, 32, 27, 26, 25). Các đèn này được nhóm thành hai bộ, mỗi bộ mô phỏng một cụm đèn Xanh-Vàng-Đỏ cho một hướng giao thông.
 - Hướng 1 (Giả định): LED_PIN_1 (Xanh), LED_PIN_2 (Vàng), LED_PIN_3 (Đỏ)
 - Hướng 2 (Giả định): LED_PIN_6 (Xanh), LED_PIN_5 (Vàng), LED_PIN_4 (Đỏ)
- Màn hình Hiển thị Thời gian: Hai (2) module màn hình LED 7 đoạn 4 chữ số dựa trên IC TM1637.
 - Display 1: Kết nối với ESP32 qua chân CLK1 (GPIO 5) và DIO1 (GPIO 17).
 - Display 2: Kết nối với ESP32 qua chân CLK2 (GPIO 16) và DIO2 (GPIO 4).

- Nguồn cấp: ESP32 và các module ngoại vi được cấp nguồn phù hợp (thường là 3.3V hoặc 5V qua cổng USB hoặc chân Vin).

3.2.4.3. Kiến trúc Phần mềm và Logic Hoạt động

Firmware chạy trên ESP32 được viết bằng Arduino Core cho ESP32, sử dụng các thư viện chuẩn và thư viện bên ngoài.

- **Khởi tạo**

- Khởi động giao tiếp Serial để debug.
- Kết nối ESP32 vào mạng Wi-Fi cục bộ sử dụng SSID và mật khẩu được cung cấp (Nha Bap 5G). Quá trình này lặp lại cho đến khi kết nối thành công.
- Cấu hình các chân GPIO điều khiển 6 đèn LED là OUTPUT.
- Khởi tạo hai đối tượng TM1637Display (display1, display2), đặt độ sáng và xóa màn hình ban đầu.
- Gọi controlTM1627s(0, 0) để hiển thị "00:00" ban đầu.



```
void setup() {
    Serial.begin(115200); // Khởi tạo Serial Monitor

    WiFi.begin(ssid, password); // Bắt đầu kết nối WiFi
    while (WiFi.status() != WL_CONNECTED) { // Chờ kết nối
        delay(1000);
        Serial.println("Đang kết nối...");
    }
    Serial.println("Kết nối WiFi thành công!");

    // Cấu hình chân LED là OUTPUT
    pinMode(LED_PIN_1, OUTPUT);
    pinMode(LED_PIN_2, OUTPUT);
    // ... (các pin khác)

    // Cấu hình và xóa màn hình TM1637
    display1.setBrightness(0x0f);
    display1.clear();
    display2.setBrightness(0x0f);
    display2.clear();

    controlTM1627s(0, 0); // Hiển thị 00:00 ban đầu
}
```

- **Giải thích:** Hàm setup thực hiện các bước cấu hình cơ bản và cần thiết một lần khi ESP32 khởi động, đảm bảo kết nối mạng và các thiết bị ngoại vi sẵn sàng hoạt động.
 - **Vòng lặp Chính:** Đây là chu trình hoạt động liên tục của ESP32.
- Bước 1: Lấy dữ liệu: Gọi hàm getDataFromAPI() để thực hiện yêu cầu HTTP GET đến ThingSpeak và lấy giá trị thời gian đèn xanh mới nhất cho hai hướng (lưu vào biến toàn cục field1 và field2).
- Bước 2: Chuyển đổi dữ liệu: Chuyển đổi các giá trị field1 và field2 (dạng String) sang kiểu số nguyên (long).

- Bước 3: Thực thi chu kỳ đèn: Gọi hàm controlLEDs() để bắt đầu điều khiển đèn và màn hình.
 - Nếu value1 và value2 hợp lệ (khác 0), thời gian đèn xanh (t1, t3) sẽ được tính bằng value1 * 1000 và value2 * 1000 (chuyển từ giây sang miligây). Thời gian đèn vàng (t2) được cố định là 3000ms.
 - Nếu dữ liệu từ API không hợp lệ (bằng 0), hệ thống sử dụng thời gian mặc định: Xanh1 = 20000ms, Vàng = 3000ms, Xanh2 = 15000ms.
- Bước 4: Reset màn hình (tùy chọn): Gọi controlTM1627s(0, 0) sau khi chu kỳ controlLEDs kết thúc. Bước này có thể nhằm mục đích xóa màn hình trước khi bắt đầu chu kỳ mới.



```

void loop() {
    getDataFromAPI(); // Lấy thời gian đèn mới từ ThingSpeak

    // Chuyển đổi dữ liệu String sang long
    long value1 = field1.toInt();
    long value2 = field2.toInt();

    // Kiểm tra dữ liệu hợp lệ và gọi hàm điều khiển đèn
    // Nếu dữ liệu hợp lệ, dùng value1, value2 (đã nhân 1000) làm thời gian xanh
    // Nếu không, dùng thời gian mặc định 20s và 15s
    if (!(value1 == 0 && value2 == 0))
        controlLEDs(value1 * 1000, 3000, value2 * 1000); // t1, t2 (vàng), t3
    else
        controlLEDs(20000, 3000, 15000); // Thời gian mặc định

    controlTM1627s(0, 0); // Reset màn hình sau chu kỳ (có thể không cần thiết)
}

```

- **Giải thích:** Hàm loop điều phối hoạt động chính: lấy dữ liệu mới nhất, sau đó thực thi một chu kỳ đèn hoàn chỉnh dựa trên dữ liệu đó (hoặc dữ liệu mặc định).
 - **Lấy Dữ liệu từ API (**
 - Sử dụng thư viện HTTPClient để tạo một yêu cầu HTTP GET đến apiUrl (ThingSpeak).
 - Kiểm tra mã trạng thái HTTP. Nếu thành công (ví dụ: 200 OK):

- Lấy nội dung phản hồi (payload) dạng String.
 - Sử dụng thư viện ArduinoJson để phân tích chuỗi JSON này.
 - Truy cập vào mảng feeds và lấy đối tượng cuối cùng (feeds[feeds.size() - 1]).
 - Trích xuất giá trị của "field1" và "field2" từ đối tượng cuối cùng và lưu vào các biến toàn cục tương ứng.
- Xử lý các trường hợp lỗi: không kết nối được HTTP, lỗi phân tích JSON, không có dữ liệu trong feeds.
 - Đóng kết nối HTTP (http.end()).

```

void getDataFromAPI() {
    Serial.println("Check"); // In ra Serial để biết đang lấy dữ liệu

    HTTPClient http;
    http.begin(apiUrl); // Khởi tạo yêu cầu HTTP
    int httpCode = http.GET(); // Gửi yêu cầu GET

    if (httpCode > 0) { // Kiểm tra yêu cầu thành công
        String payload = http.getString(); // Lấy nội dung phản hồi
        DynamicJsonDocument doc(1024); // Tạo bộ đệm JSON động
        DeserializationError error = deserializeJson(doc, payload); // Phân tích JSON

        if (error) {
            Serial.println("Không thể phân tích JSON");
            return; // Thoát nếu lỗi JSON
        }

        JSONArray feeds = doc["feeds"].as<JSONArray>(); // Lấy mảng 'feeds'

        if (feeds.size() > 0) { // Kiểm tra có dữ liệu không
            JsonObject lastFeed = feeds[feeds.size() - 1]; // Lấy bản ghi mới nhất
            // Lấy giá trị field1, field2 và lưu vào biến toàn cục
            field1 = lastFeed["field1"].as<String>();
            field2 = lastFeed["field2"].as<String>();
            Serial.println("Field1: " + field1);
            Serial.println("Field2: " + field2);
        } else {
            Serial.println("Không có dữ liệu trong feeds.");
        }
    } else {
        Serial.println("Yêu cầu thất bại, mã lỗi: " + String(httpCode));
    }

    http.end(); // Đóng kết nối HTTP
}

```

- **Giải thích:** Hàm này đóng vai trò cầu nối giữa ESP32 và dữ liệu điều khiển từ backend (qua ThingSpeak). Nó định kỳ kéo (pull) dữ liệu mới nhất để cập nhật thời gian đèn cho chu kỳ tiếp theo.
- **Điều khiển Đèn LED**
 - Đây là hàm cốt lõi thực thi chu kỳ đèn. Nó nhận 3 tham số thời gian (đơn vị: mili giây): t1 (Xanh Hướng 1), t2 (Vàng cả hai hướng), t3 (Xanh Hướng 2).
 - Sử dụng biến startMillis và millis() để theo dõi thời gian trôi qua trong chu kỳ.
 - Vòng lặp while(true) chạy cho đến khi tổng thời gian (t1 + t2 + t3) trôi qua.

- Logic bật/tắt LED: Dựa trên elapsedTime, các chân LED được điều khiển HIGH/LOW theo đúng trình tự pha:
 - Pha 1 ($0 \rightarrow t1$): LED1 (Xanh1) HIGH, LED4 (Đỏ2) HIGH. Các LED khác LOW.
 - Pha 2 ($t1 \rightarrow t1+t2$): LED2 (Vàng1) HIGH, LED4 (Đỏ2) HIGH. Các LED khác LOW.
 - Pha 3 ($t1+t2 \rightarrow t1+t2+t3$): LED3 (Đỏ1) HIGH, LED6 (Xanh2) HIGH. Các LED khác LOW.
 - Pha 4 ($t1+t2+t3 \rightarrow t1+t2+t3+t2$ - Logic ngầm định cho Vàng 2): Mặc dù không có pha t4 riêng, logic trong code hiện tại cho LED5 (Vàng2) chỉ bật khi $\text{elapsedTime} > t1 + t3 \ \&\& \text{elapsedTime} \leq t1 + t2 + t3$, điều này có vẻ không đúng với chu kỳ chuẩn. Logic chuẩn sẽ là: Pha Vàng 2 diễn ra sau Pha Xanh 2, tức là từ $t1+t2+t3$ đến $t1+t2+t3+t2$. Cân xem xét lại logic điều khiển LED 4, 5, 6 để đảm bảo đúng chu kỳ Xanh2->Vàng2->Đỏ2.
 - Tính toán Thời gian Còn lại: Trong mỗi vòng lặp, tính remainingTime1 và remaining Time 2 (thời gian còn lại của pha hiện tại cho mỗi hướng) dựa trên trạng thái của đèn Xanh hoặc Vàng đang sáng.

```

● ● ●

void controlLEDs(unsigned long t1, unsigned long t2, unsigned long t3) {
    unsigned long startMillis = millis(); // Ghi lại thời điểm bắt đầu chu kỳ

    while (true) { // Vòng lặp thực thi chu kỳ đèn
        unsigned long currentMillis = millis();
        unsigned long elapsedTime = currentMillis - startMillis; // Thời gian đã trôi qua

        // Thoát vòng lặp khi hết chu kỳ (tổng thời gian Xanh1+Vàng1+Xanh2+Vàng2 - logic vàng 2 cần xem lại)
        // Hiện tại đang thoát sau Xanh1+Vàng1+Xanh2
        if (elapsedTime > t1 + t2 + t3) { // Điều kiện thoát có thể cần +t2 nữa
            break;
        }

        // --- Điều khiển LED Hướng 1 ---
        // LED_PIN_1 (Xanh 1): Bật trong khoảng 0 -> t1
        digitalWrite(LED_PIN_1, elapsedTime <= t1 ? HIGH : LOW);
        // LED_PIN_2 (Vàng 1): Bật trong khoảng t1 -> t1+t2
        digitalWrite(LED_PIN_2, (elapsedTime > t1 && elapsedTime <= t1 + t2) ? HIGH : LOW);
        // LED_PIN_3 (Đỏ 1): Bật trong khoảng t1+t2 -> t1+t2+t3 (+t2 nữa?)
        digitalWrite(LED_PIN_3, (elapsedTime > t1 + t2 /*&& elapsedTime <= t1 + t2 + t3 + t2*/) ? HIGH :
        LOW); // Logic Đỏ 1 cần xem lại

        // --- Điều khiển LED Hướng 2 ---
        // LED_PIN_6 (Xanh 2): Bật trong khoảng t1+t2 -> t1+t2+t3
        digitalWrite(LED_PIN_6, (elapsedTime > t1 + t2 && elapsedTime <= t1 + t2 + t3) ? HIGH : LOW);
        // LED_PIN_5 (Vàng 2): Logic hiện tại bật khi Xanh 2 đang chạy? Cần sửa.
        // Logic đúng: Bật trong khoảng t1+t2+t3 -> t1+t2+t3+t2
        // digitalWrite(LED_PIN_5, (elapsedTime > t1 + t2 + t3 && elapsedTime <= t1 + t2 + t3 + t2) ? HIGH :
        LOW);
        digitalWrite(LED_PIN_5, (elapsedTime > t1 + t3 && elapsedTime <= t1 + t2 + t3) ? HIGH : LOW); // Logic hiện tại

        // LED_PIN_4 (Đỏ 2): Bật trong khoảng 0 -> t1+t2
        digitalWrite(LED_PIN_4, elapsedTime <= t1 + t2 ? HIGH : LOW); // Logic Đỏ 2

        // --- Tính toán thời gian còn lại cho hiển thị ---
        int remainingTime1_ms = 0;
        if (digitalRead(LED_PIN_1) == HIGH) { // Nếu Xanh 1 đang bật
            remainingTime1_ms = t1 - elapsedTime;
        } else if (digitalRead(LED_PIN_2) == HIGH) { // Nếu Vàng 1 đang bật
            remainingTime1_ms = t1 + t2 - elapsedTime;
        } else { // Ngược lại là Đỏ 1
            // Thời gian đỏ 1 = Xanh 2 + Vàng 2 = t3 + t2
            // Thời gian bắt đầu đỏ 1 là t1 + t2
            remainingTime1_ms = (t1 + t2 + t3 + t2) - elapsedTime; // Giả sử có pha vàng 2
        }

        int remainingTime2_ms = 0;
        if (digitalRead(LED_PIN_6) == HIGH) { // Nếu Xanh 2 đang bật
            remainingTime2_ms = (t1 + t2 + t3) - elapsedTime;
        } else if (digitalRead(LED_PIN_5) == HIGH) { // Nếu Vàng 2 đang bật (theo logic đúng)
            remainingTime2_ms = (t1 + t2 + t3 + t2) - elapsedTime;
        } else { // Ngược lại là Đỏ 2
            // Thời gian đỏ 2 = Xanh 1 + Vàng 1 = t1 + t2
            // Thời gian bắt đầu đỏ 2 là 0
            remainingTime2_ms = (t1 + t2) - elapsedTime;
        }

        // Cập nhật màn hình TM1637
        controlTM1627s(max(0, remainingTime1_ms), max(0, remainingTime2_ms));

        delay(50); // Delay nhỏ để tránh CPU load quá cao và cập nhật màn hình không quá nhanh
    }
}

```

- **Giải thích:** Hàm này mô phỏng hoạt động tuần tự của đèn giao thông. Nó chia chu kỳ thành các pha dựa trên thời gian và bật/tắt các LED tương ứng. Đồng thời, nó tính toán thời gian còn lại của pha hiện tại cho mỗi hướng và cập nhật lên màn hình 7 đoạn, tạo hiệu ứng đèn ngang. Cần lưu ý về tính chất blocking và kiểm tra lại logic điều khiển LED cho hướng 2 (đặc biệt là pha Vàng 2 và Đỏ 1) để đảm bảo tính chính xác của chu kỳ đèn 4 pha chuẩn.

- Điều khiển Màn hình 7 đoạn

- Nhận thời gian còn lại (mili giây) cho hai hướng.
- Chia cho 1000 để đổi sang giây.
- Tính toán số phút và số giây còn lại.
- Định dạng thành giá trị 4 chữ số (ví dụ: 1 phút 30 giây thành 130).
- Sử dụng hàm display.showNumberDecEx() của thư viện TM1637Display để hiển thị số lên màn hình, bật dấu hai chấm (0b01000000).



```

void controlTM1627s(unsigned long t1_ms, unsigned long t2_ms) {
    // Chuyển đổi mili giây sang giây
    int value1_sec = t1_ms / 1000;
    int value2_sec = t2_ms / 1000;

    // Tính phút và giây cho màn hình 1
    int minutes1 = value1_sec / 60;
    int seconds1 = value1_sec % 60;
    // Tạo giá trị hiển thị dạng M M S S (ví dụ: 0130 cho 1 phút 30 giây)
    int displayValue1 = minutes1 * 100 + seconds1;
    // Hiển thị số, bật dấu hai chấm (bit 6), giữ số 0 ở đầu nếu cần (true)
    display1.showNumberDecEx(displayValue1, 0b01000000, true);

    // Tính phút và giây cho màn hình 2
    int minutes2 = value2_sec / 60;
    int seconds2 = value2_sec % 60;
    int displayValue2 = minutes2 * 100 + seconds2;
    display2.showNumberDecEx(displayValue2, 0b01000000, true);
}

```

- **Giải thích:** Hàm này đảm nhận việc định dạng và hiển thị thời gian đếm ngược lên hai module LED 7 đoạn, giúp người quan sát dễ dàng theo dõi thời gian còn lại của các pha đèn.

3.2.4.4. Luồng Dữ liệu và Tích hợp

- Đầu vào: Dữ liệu thời gian đèn xanh (field1, field2) được kéo (pull) từ ThingSpeak API.
- Đầu ra: Tín hiệu điều khiển đầy (push) đến các chân GPIO để bật/tắt đèn LED và dữ liệu hiển thị được gửi đến các module TM1637.
- Tích hợp: Module này hoạt động như một *client* của dữ liệu do backend (app.py) tính toán và đẩy lên ThingSpeak. Nó không giao tiếp trực tiếp với backend qua Wi-Fi mà thông qua nền tảng đám mây trung gian.

3.2.4.5. Hạn chế và Điểm cần Lưu ý

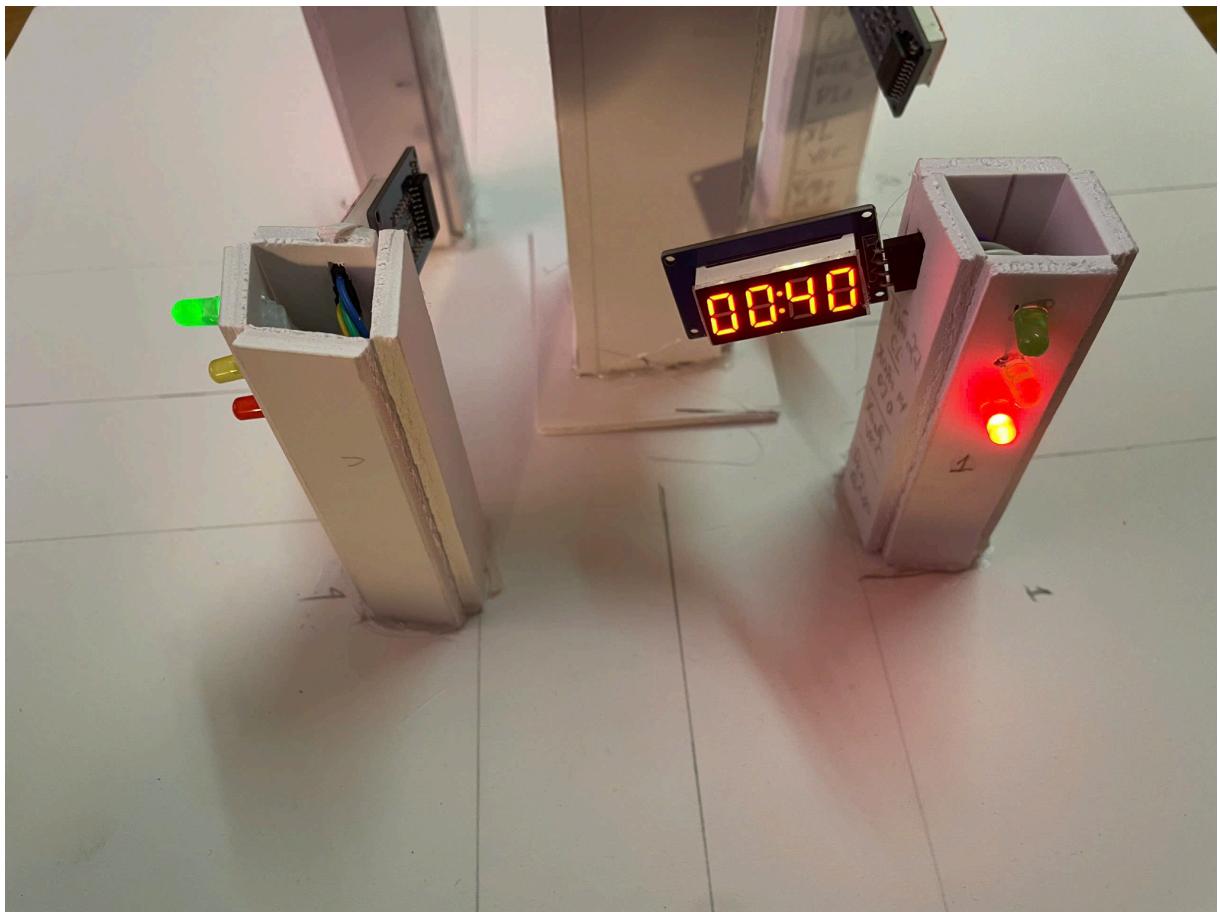
- Phụ thuộc ThingSpeak: Hoạt động của module hoàn toàn phụ thuộc vào khả năng truy cập và tính sẵn sàng của ThingSpeak API. Nếu ThingSpeak gặp sự cố hoặc kết nối mạng bị gián đoạn, ESP32 sẽ sử dụng thời gian mặc định.
- Logic Điều khiển LED Hướng 2: Cần kiểm tra và hiệu chỉnh lại logic bật/tắt cho LED_PIN_4, LED_PIN_5, LED_PIN_6 để đảm bảo chu kỳ Xanh2 -> Vàng2 -> Đỏ2 hoạt động chính xác và đồng bộ với Hướng 1. Đặc biệt là thời điểm bắt đầu/kết thúc của Đỏ 1 và Vàng 2.
- Tính chất Blocking: Hàm controlLEDs hiện đang chặn thực thi trong suốt chu kỳ đèn. Điều này có thể chấp nhận được nếu ESP32 chỉ làm nhiệm vụ này, nhưng sẽ là vấn đề nếu cần xử lý các sự kiện khác (ví dụ: nút nhấn, cảm biến cục bộ) trong khi đèn đang chạy. Có thể cải thiện bằng cách sử dụng các kỹ thuật lập trình không chặn (non-blocking) như máy trạng thái (state machine).
- Độ trễ Dữ liệu: Có độ trễ nhất định giữa thời điểm backend tính toán và đẩy dữ liệu lên ThingSpeak, và thời điểm ESP32 kéo dữ liệu về và áp dụng. Tuy nhiên, do chu kỳ đèn thường kéo dài hàng chục giây, độ trễ này có thể chấp nhận được.

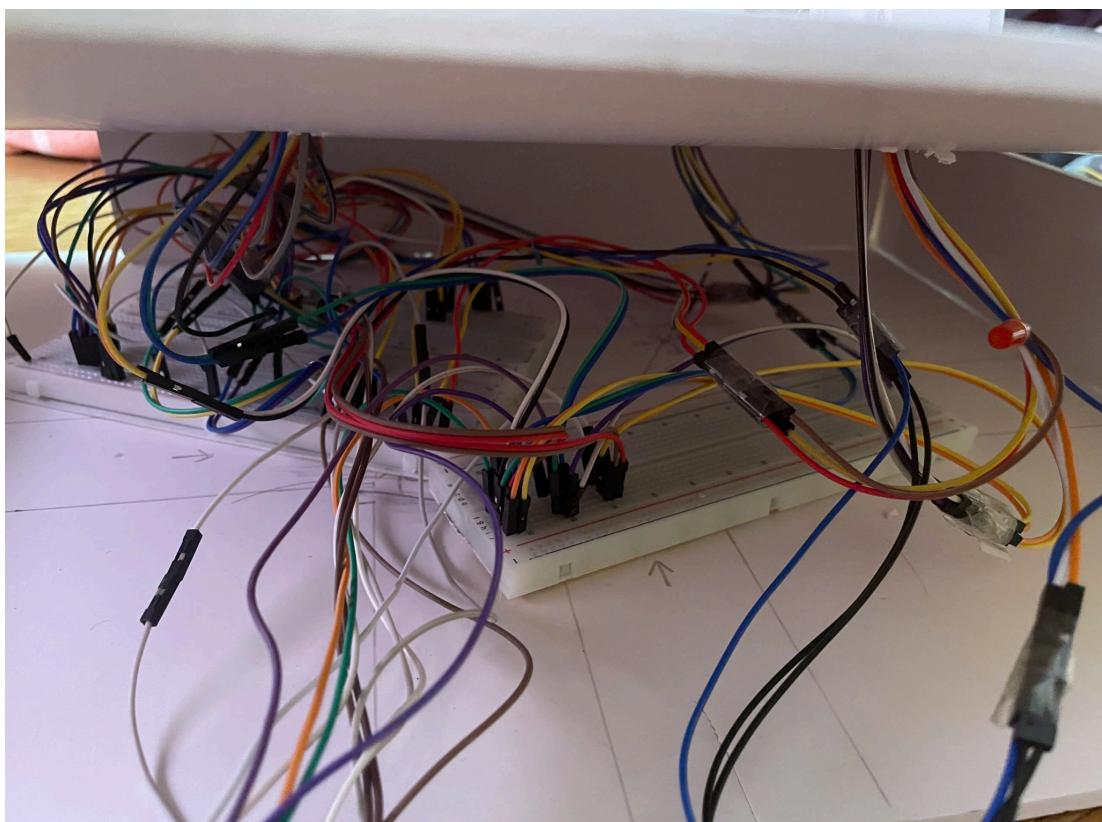
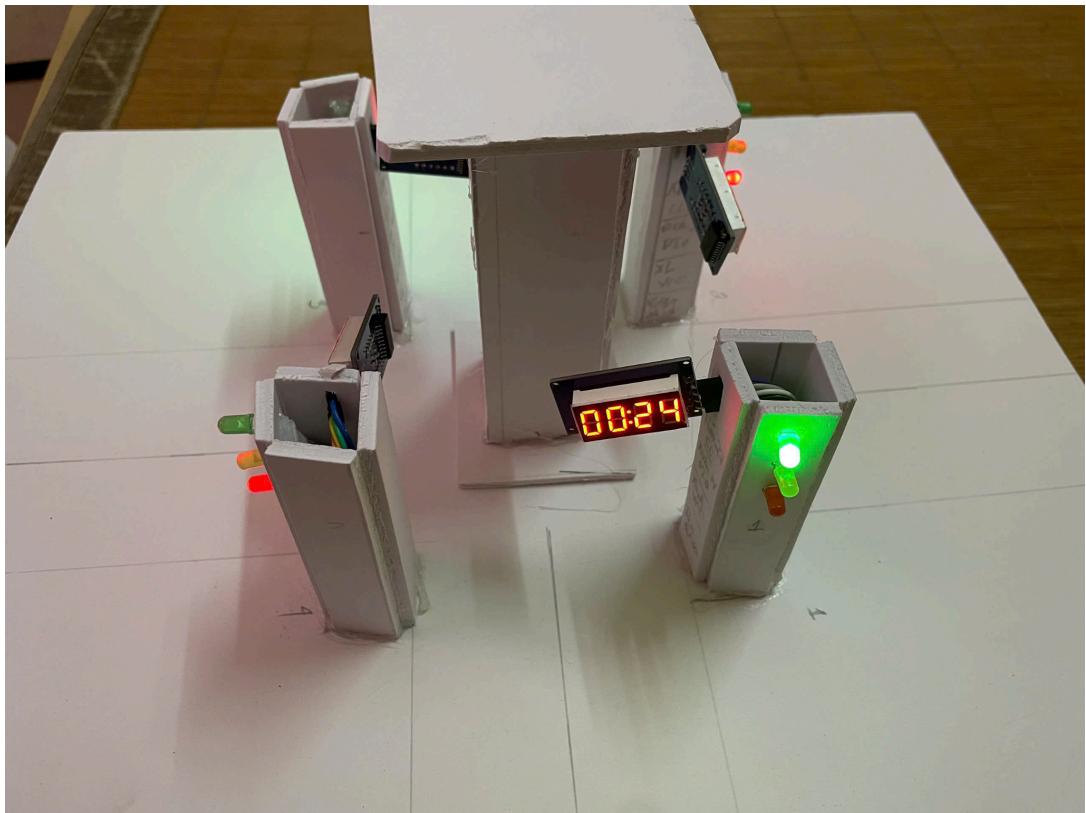
3.2.4.6. Kết luận

Module Điều khiển Cục bộ ESP32 là thành phần thực thi hiệu quả, đảm nhận việc chuyển đổi dữ liệu thời gian đèn nhận được từ đám mây thành các tín hiệu điều khiển vật lý cho đèn LED và màn hình hiển thị trên mô hình. Với khả năng kết nối Wi-Fi, xử lý JSON và điều khiển GPIO, ESP32 hoàn thành tốt vai trò của mình trong việc mô phỏng hoạt động của một bộ điều khiển đèn giao thông thực tế, phản ánh các quyết định được đưa ra bởi thuật toán thông minh ở backend. Mặc dù có một số điểm cần cải thiện trong logic và cấu trúc code (đặc biệt là tính blocking và logic LED hướng 2), module này đã chứng minh được khả năng hoạt động cơ bản theo yêu cầu của hệ thống.

3.3 Cài đặt thử nghiệm và đánh giá

3.3.1. Phần cứng





- **Giới thiệu:**

Để kiểm chứng và đánh giá hiệu quả của "Hệ thống Điều tiết Giao thông Thông minh sử dụng ESP32 và Camera IMOU", một mô hình thực nghiệm vật lý đã được xây dựng. Mô hình này mô phỏng một giao lộ bốn hướng điển hình, tích hợp các thành phần phần cứng cần thiết để tái tạo các chức năng cốt lõi của hệ thống, bao gồm thu thập dữ liệu hình ảnh, xử lý thông tin, ra quyết định điều khiển và thực thi thay đổi trạng thái đèn tín hiệu. Mục này trình bày chi tiết về các thành phần phần cứng được lựa chọn và cách chúng được tích hợp trong mô hình thực nghiệm.

- **Mô tả các Thành phần Phần cứng Chính:**

Hệ thống phần cứng bao gồm các khối chức năng chính sau:

- **Khối Cảm biến (Sensing Unit) - Camera IMOU:**

- Vai trò: Là "mắt thần" của hệ thống, chịu trách nhiệm thu thập hình ảnh hoặc luồng video thời gian thực của khu vực giao lộ. Dữ liệu hình ảnh này là đầu vào quan trọng cho thuật toán phát hiện và đếm phương tiện dựa trên YOLOv8.
- Đặc điểm: Sử dụng một camera IP Wifi thuộc dòng sản phẩm IMOU. Các camera này thường có độ phân giải đủ dùng (ví dụ: 1080p), khả năng kết nối không dây Wifi tiện lợi, và hỗ trợ giao thức RTSP (Real-Time Streaming Protocol) để truyền luồng video đến bộ xử lý trung tâm. Camera được cấp nguồn qua adapter riêng và cần được đặt ở vị trí có góc nhìn bao quát được các hướng quan trọng của giao lộ mô hình.

- **Khối Xử lý Trung tâm (Central Processing Unit - External):**

- Vai trò: Đây là bộ não tính toán chính của hệ thống, nơi thực thi các tác vụ nặng về xử lý. Do giới hạn về năng lực tính toán của vi điều khiển ESP32, các tác vụ như chạy mô hình YOLOv8 để phát hiện đối tượng, phân tích mật độ, thực thi thuật toán tính toán thời gian đèn

(calculate_traffic_light_times), và quản lý giao diện người dùng (through qua Fast API/WebSocket trong file app.py) được thực hiện trên một máy tính mạnh hơn.

- Thành phần: Thường là một Máy tính Cá nhân (PC), Laptop, hoặc một Máy tính Bảng đơn (SBC - Single Board Computer) đủ mạnh như Raspberry Pi 4 hoặc Jetson Nano (nếu cần tăng tốc GPU).
- Kết nối: Nhận luồng video từ Camera IMOU qua mạng Wi-Fi (sử dụng RTSP). Giao tiếp với Khối Điều khiển (ESP32) qua mạng Wi-Fi (sử dụng các giao thức như MQTT, HTTP requests, hoặc WebSockets) để gửi lệnh điều khiển đèn và nhận phản hồi (nếu có).

- **Khối Điều khiển Cục bộ (Local Control Unit) - Vi điều khiển ESP32:**

- Vai trò: Là trái tim điều khiển của mô hình vật lý. Nó nhận lệnh về thời gian đèn xanh/đỏ cho từng hướng từ Khối Xử lý Trung tâm qua Wi-Fi. Dựa trên lệnh này, ESP32 trực tiếp điều khiển trạng thái bật/tắt của các đèn LED tín hiệu và cập nhật thời gian đèn ngay trên màn hình hiển thị.
- Thành phần: Sử dụng một hoặc nhiều board mạch phát triển ESP32 (ví dụ: ESP32-DevKitC). ESP32 được chọn vì có tích hợp sẵn Wi-Fi, Bluetooth, đủ số chân GPIO để điều khiển nhiều đèn LED và giao tiếp với màn hình hiển thị, cùng với khả năng xử lý đủ để nhận và thực thi lệnh điều khiển. Trong mô hình (hình ảnh), các board mạch màu đen gắn trên cột đèn hoặc cấu trúc trung tâm rất có thể là các module ESP32 hoặc các module phụ trợ được ESP32 điều khiển.
- Kết nối: Kết nối vào mạng WiFi cục bộ để nhận lệnh từ Khối Xử lý Trung tâm. Các chân GPIO của ESP32 được nối trực tiếp (hoặc qua mạch đệm/driver nếu cần) đến các đèn LED và module màn hình hiển thị.

- **Khối Hiển thị Tín hiệu (Signal Display Unit) - Đèn Giao thông LED:**

- Vai trò: Mô phỏng trực quan trạng thái của đèn tín hiệu giao thông cho từng hướng.

- Thành phần: Mỗi cột đèn trong mô hình được trang bị 3 đèn LED với các màu tiêu chuẩn: Đỏ, Vàng (hoặc Cam), và Xanh lá. Các đèn LED này (thường là loại 5mm hoặc 3mm) được kết nối với các chân GPIO của ESP32 thông qua điện trở hạn dòng phù hợp.
 - Hoạt động: ESP32 bật/tắt các LED tương ứng theo đúng pha (Xanh, Vàng, Đỏ) và thời gian nhận được từ Khối Xử lý Trung tâm.
- **Khối Hiển thị Thời gian (Timer Display Unit) - Màn hình 7 đoạn:**
 - Vai trò: Hiển thị thời gian đêm ngược còn lại cho pha đèn hiện tại (thường là pha Xanh hoặc Đỏ), cung cấp thông tin trực quan cho người quan sát mô hình.
 - Thành phần: Sử dụng một module màn hình LED 7 đoạn 4 chữ số (như loại dựa trên IC TM1637 hoặc MAX7219). Module này giao tiếp với ESP32 thông qua một số ít chân (ví dụ: 2 chân cho TM1637 theo giao thức giống I2C, hoặc nhiều chân hơn cho MAX7219 theo SPI). Trong hình ảnh, màn hình hiển thị "00:24" rõ ràng là thành phần này.
 - Hoạt động: ESP32 tính toán thời gian còn lại của pha đèn hiện tại và gửi dữ liệu tương ứng đến module màn hình để hiển thị.
 - **Mô hình Giao lộ Vật lý:**
 - Vai trò: Tạo ra một không gian vật lý thu nhỏ để lắp đặt các thành phần và quan sát hoạt động của hệ thống.
 - Thành phần: Được làm từ các vật liệu mô hình phổ biến như tấm format (foam board), bìa cứng, hoặc mica. Mô hình bao gồm một mặt phản ánh để đại diện cho mặt đường, các đường kẻ phân làn (như trong hình), và các cột đèn được dựng lên tại các vị trí tương ứng của một giao lộ bốn hướng. Cấu trúc trung tâm có thể dùng để che giấu dây dẫn hoặc đặt ESP32 chính.
 - **Khối Nguồn (Power Supply Unit):**
 - Vai trò: Cung cấp năng lượng ổn định cho tất cả các thành phần điện tử.
 - Thành phần: Bao gồm các adapter nguồn cho Camera IMOU và Khối Xử lý Trung tâm (PC/Laptop/SBC). Các module ESP32, đèn LED, và màn

hình 7 đoạn thường được cấp nguồn qua cổng USB từ máy tính hoặc từ một nguồn điện DC 5V/3.3V riêng biệt, đảm bảo đủ dòng cho tất cả các thành phần hoạt động đồng thời.

- **Kiến trúc Kết nối và Luồng Dữ liệu:**

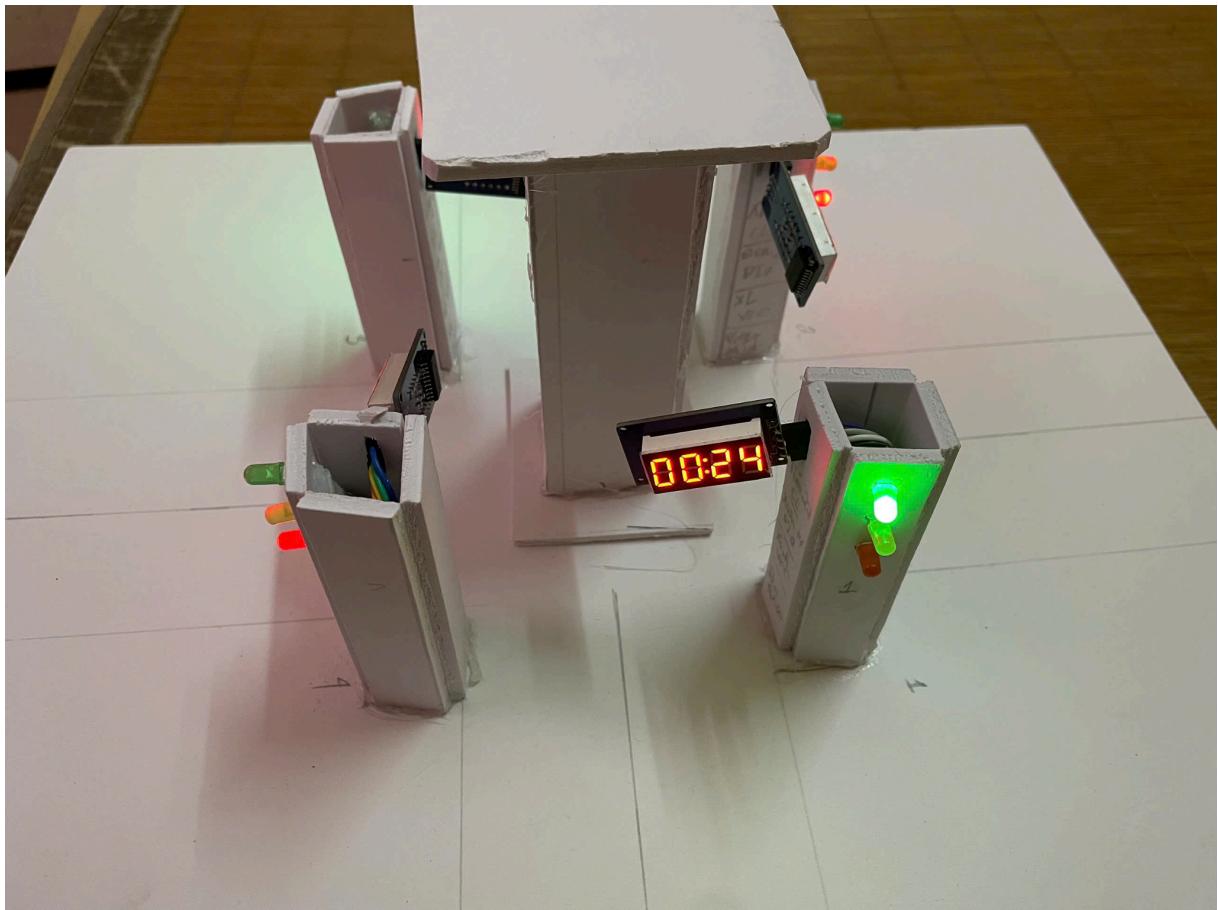
Luồng dữ liệu chính:

- Camera IMOU ghi hình và truyền luồng video RTSP qua Wi-Fi đến Khối Xử lý Trung tâm.
- Khối Xử lý Trung tâm chạy script app.py, nhận luồng video, xử lý bằng OpenCV và YOLOv8, đếm xe, và thực thi thuật toán calculate_traffic_light_times.
- Khối Xử lý Trung tâm gửi lệnh (chứa thông tin thời gian đèn xanh/đỏ mới) qua Wi-Fi đến ESP32.
- ESP32 nhận lệnh, lưu trữ thời gian, và điều khiển trực tiếp trạng thái các đèn LED và cập nhật màn hình 7 đoạn theo thời gian thực.

- **Thiết lập và Cấu hình:**

- Các đèn LED và màn hình 7 đoạn được hàn nối hoặc cắm vào board mạch ESP32 theo sơ đồ chân đã định nghĩa trong firmware của ESP32.
- Camera IMOU được đặt ở vị trí phù hợp để quan sát mô hình giao lộ.
- Tất cả các thiết bị (Camera, Khối xử lý, ESP32) được kết nối vào cùng một mạng Wifi.
- Địa chỉ IP của Camera và ESP32 cần được cấu hình trong script app.py để thiết lập kết nối RTSP và giao tiếp điều khiển.
- Các tham số vật lý như vị trí đường đếm ảo trên khung hình camera được cấu hình trong traffic_control_settings.

- **Hình ảnh Mô hình Thực nghiệm**



Hình ảnh cho thấy rõ cấu trúc mô hình giao lô với 4 cột đèn tín hiệu, mỗi cột gắn LED Xanh/Vàng/Đỏ. Một màn hình 7 đoạn hiển thị thời gian đêm ngược được gắn trên một cột. Các module điều khiển (có thể là ESP32) được nhìn thấy gắn trên các cột. Mô hình được làm từ vật liệu màu trắng, đặt trên nền trắng có kẻ vạch.

- **Kết luận:**

Việc xây dựng mô hình phản ứng thực nghiệm đóng vai trò quan trọng trong việc trực quan hóa và kiểm tra hoạt động của hệ thống điều khiển giao thông thông minh. Sự kết hợp giữa camera IP làm cảm biến, một bộ xử lý mạnh mẽ cho các tác vụ AI, và vi điều khiển ESP32 làm bộ điều khiển cục bộ cho các thiết bị ngoại vi (đèn, màn hình) tạo thành một kiến trúc phân tán linh hoạt và hiệu quả. Mô hình vật lý này cung cấp một nền tảng vững chắc để đánh giá thuật toán điều khiển và tinh chỉnh các tham số hệ thống trong môi trường gần với thực tế.

3.3.2. Phần mềm

Hệ Thống Giao Thông Thông Minh

Camera Stream



Tổng số xe đã đếm: 0
Xe trong frame: 0
Mật độ: 0.00

05-05-2025 09:30:12

Điều Khiển Camera

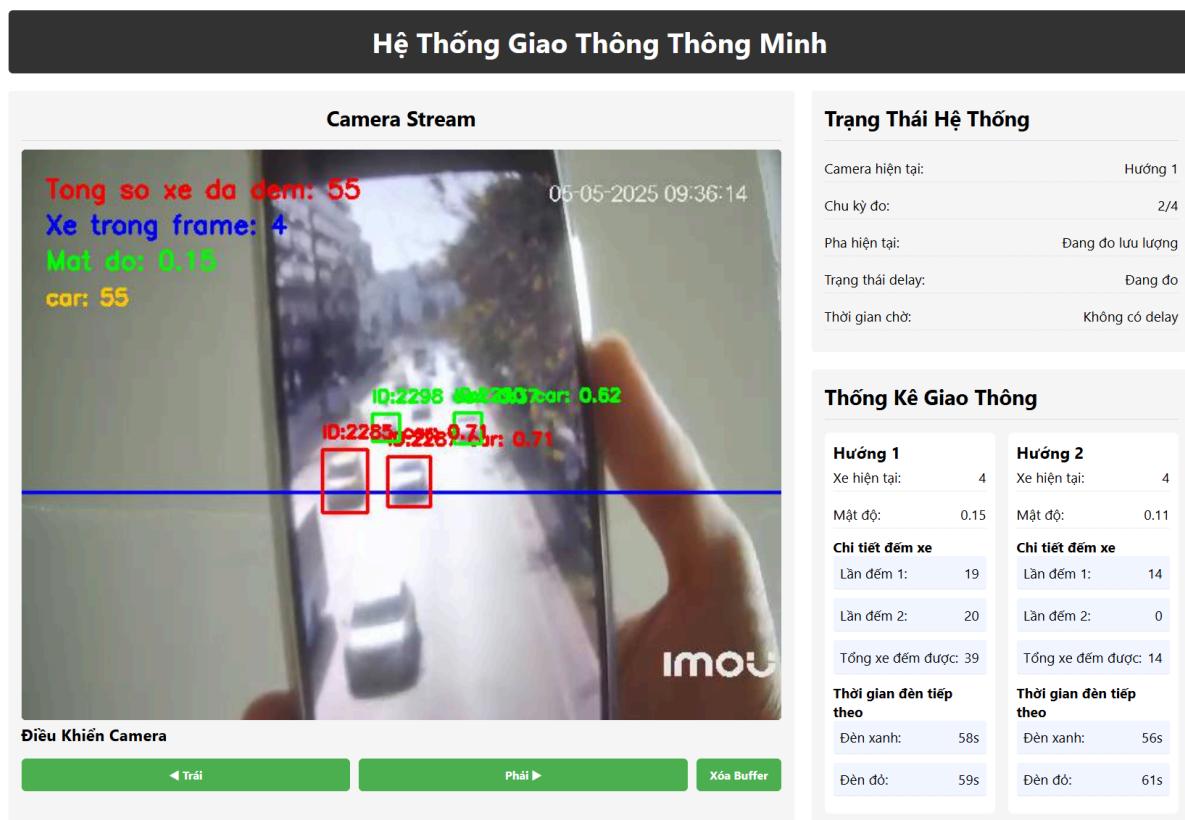
◀ Trái Phải ▶ Xóa Buffer

Trạng Thái Hệ Thống

Camera hiện tại:	Hướng 1
Chu kỳ đo:	2/4
Pha hiện tại:	Đang đo lưu lượng
Trạng thái delay:	Đang đo
Thời gian chờ:	Không có delay

Thông Kê Giao Thông

Hướng 1	Hướng 2
Xe hiện tại: 0	Xe hiện tại: 0
Mật độ: 0.00	Mật độ: 0.00
Chi tiết đếm xe	
Lần đếm 1: 0	Lần đếm 1: 0
Lần đếm 2: 0	Lần đếm 2: 0
Tổng xe đếm được: 0	
Thời gian đèn tiếp theo	
Đèn xanh: Chờ tính toán	Đèn xanh: Chờ tính toán
Đèn đỏ: Chờ tính toán	



- Giới thiệu:

Bên cạnh phần cứng mô phỏng và các module xử lý cấp thấp, phần mềm đóng vai trò then chốt trong việc tích hợp, điều khiển, giám sát và phân tích hoạt động của Hệ thống Điều tiết Giao thông Thông minh. Phần mềm bao gồm hai thành phần chính: ứng dụng backend (được mô tả chi tiết trong mục [Số mục tham chiếu đến mục Module xử lý dữ liệu]) chạy trên Khối Xử lý Trung tâm, và giao diện người dùng (frontend) dựa trên web để giám sát và tương tác. Mục này tập trung vào việc mô tả giao diện người dùng và làm rõ logic xử lý cốt lõi, đặc biệt là cơ chế đếm phương tiện.

- Giao diện Người dùng (Frontend Web Application):

Để cung cấp khả năng giám sát và điều khiển trực quan, một giao diện người dùng dựa trên web đã được phát triển. Giao diện này (như thể hiện trong Hình [Số hình]) được xây dựng bằng các công nghệ web tiêu chuẩn (HTML, CSS, JavaScript) và giao tiếp

với ứng dụng backend (FastAPI trong app.py) thông qua giao thức WebSocket để nhận dữ liệu cập nhật theo thời gian thực.

- **Thành phần Chính của Giao diện:**

- **Luồng Camera Trực tiếp (Camera Stream):**

- Hiển thị luồng video nhận được từ Camera IMOU sau khi đã được xử lý bởi backend.
 - **Quan trọng:** Luồng video này không phải là video gốc mà là video đã được *chồng lớp* thông tin xử lý bởi YOLOv8 và các thuật toán khác trên backend. Các thông tin chồng lớp bao gồm:
 - **Bounding Boxes:** Các hộp màu đỏ (hoặc màu khác) bao quanh các phương tiện được phát hiện.
 - **Thông tin Phát hiện:** ID theo dõi (ví dụ: ID:2285), lớp đối tượng (car), và độ tin cậy (cor: 0.71) hiển thị gần bounding box.
 - **Vạch Đếm Ảo:** Một đường kẻ ngang màu xanh dương (count_line_y trong code) được vẽ lên khung hình. Đây là ngưỡng quan trọng để kích hoạt bộ đếm.
 - **Thông tin Tổng hợp:** Các chỉ số như "Tổng số xe đã đếm", "Xe trong frame", "Mật độ" được hiển thị trực tiếp trên khu vực video.

- **Bảng Trạng Thái Hệ Thống (System Status Panel):**

- Cung cấp thông tin tổng quan về trạng thái hoạt động hiện tại của hệ thống:
 - Hướng camera đang quan sát ("Camera hiện tại").
 - Tiến trình của chu kỳ đo lường ("Chu kỳ đo", ví dụ: 2/4 nghĩa là đang ở bước thứ 2 trong tổng số 4 bước quay camera).
 - Pha hoạt động hiện tại ("Pha hiện tại": Đang đo lưu lượng, Đang tính toán, Đang delay).

- Trạng thái trì hoãn ("Trạng thái delay") và thời gian chờ còn lại (nếu có).
-
- **Bảng Thông Kê Giao Thông (Traffic Statistics Panel):**
 - Hiển thị chi tiết dữ liệu thống kê và kết quả tính toán cho từng hướng (Hướng 1, Hướng 2):
 - Số xe hiện tại trong khung hình ("Xe hiện tại").
 - Mật độ giao thông tức thời ("Mật độ").
 - **Chi tiết đếm xe:** Số lượng xe đếm được trong từng lần đo của chu kỳ (Lần đếm 1, Lần đếm 2). Đây là dữ liệu cốt lõi từ vehicle_counting_data.
 - Tổng số xe đếm được trong phiên đo hiện tại ("Tổng xe đếm được").
 - Thời gian đèn xanh/đỏ được tính toán cho phiên tiếp theo ("Thời gian đèn tiếp theo").
-
- **Bảng Điều Khiển Camera (Camera Control Panel):**
 - Cho phép người dùng tương tác cơ bản với camera (nếu backend hỗ trợ), ví dụ: nút "Trái", "Phải" để điều chỉnh thủ công góc quay. Nút "Xóa Buffer" có thể dùng để reset bộ đếm hoặc trạng thái nhất định.
- **Công nghệ và Luồng Dữ liệu:**
 - Frontend sử dụng JavaScript để thiết lập kết nối WebSocket đến backend (`ws://<địa chỉ backend>/ws`).
 - Backend (FastAPI) liên tục gửi các gói tin qua WebSocket:
 - Gói tin loại "frame": Chứa dữ liệu hình ảnh đã xử lý (đã vẽ bounding box, vạch đếm, thông tin) được mã hóa dưới dạng chuỗi base64. Frontend nhận, giải mã và hiển thị lên thẻ `` hoặc `<canvas>`.

- Gói tin loại "stats": Chứa toàn bộ dữ liệu trạng thái và thống kê (từ các biến traffic_stats, traffic_lights, camera_cycle, cycle_control, vehicle_counting_data). Frontend nhận dữ liệu JSON này và cập nhật nội dung các bảng thông tin tương ứng trên giao diện.
- Khi người dùng nhấn nút điều khiển trên frontend (ví dụ: nút "Trái"), JavaScript sẽ gửi một gói tin lệnh (ví dụ: {"type": "camera_control", "action": "left"}) qua WebSocket về cho backend để xử lý.

- Logic Xử lý Cốt lõi: Cơ chế Đếm Phương tiện Chính xác

Một trong những thách thức lớn nhất trong hệ thống đếm xe tự động là tránh việc đếm trùng (ví dụ: đếm cùng một xe nhiều lần khi nó di chuyển chậm qua vùng phát hiện) hoặc đếm sót. Để giải quyết vấn đề này, hệ thống áp dụng một logic đếm dựa trên việc vượt qua vạch đếm ảo kết hợp với theo dõi đối tượng (object tracking), được triển khai chủ yếu trong hàm track_and_count_vehicles của backend (app.py).

- Nguyên tắc Hoạt động:
 - Phát hiện và Theo dõi: Mô hình YOLOv8 (hoặc tracker tích hợp như ByteTrack) không chỉ phát hiện xe trong mỗi khung hình mà còn gán một ID theo dõi (tracking ID) duy nhất cho mỗi phương tiện và theo dõi nó qua các khung hình liên tiếp.
 - Xác định Vạch Đếm: Một đường kẻ ngang ảo (màu xanh dương trên giao diện) được định nghĩa tại một vị trí cố định trên khung hình (count_line_y, ví dụ: 60% chiều cao từ đỉnh).
 - Kiểm tra Vị trí: Đối với mỗi phương tiện được theo dõi (có ID), hệ thống liên tục kiểm tra vị trí của nó (thường là điểm dưới cùng của bounding box - bottom_y) so với vị trí vạch đếm (count_line_y).
 - Phát hiện Sự kiện Vượt Vạch: Hệ thống ghi nhận trạng thái "đã vượt vạch" (crossed_line) khi vị trí bottom_y của xe lớn hơn count_line_y.
 - Chống Đếm trùng bằng ID và Chu kỳ Đo:

- Quan trọng nhất: Một phương tiện chỉ được tính là "đã đếm" KHI VÀ CHỈ KHI nó chuyển từ trạng thái *chưa vượt vạch* sang trạng thái *đã vượt vạch* VÀ *ID theo dõi* của nó CHƯA TỪNG ĐƯỢC GHI NHẬN trong lần đo cụ thể đó của hướng đó.
 - Để thực hiện việc này, hệ thống sử dụng cấu trúc `vehicle_counting_data["cycle_ids"]`. Đây là một dictionary chứa các set (tập hợp). Ví dụ:
`vehicle_counting_data["cycle_ids"]["direction1"][0]` là tập hợp các ID xe đã được đếm cho Hướng 1 trong Lần đếm 1.
 - Khi một xe (với `vehicle_id`) thỏa mãn điều kiện vượt vạch, hệ thống sẽ kiểm tra `if vehicle_id not in unique_ids:` (với `unique_ids` là set tương ứng).
 - Nếu ID chưa có trong set, bộ đếm `vehicle_counting_data["cycle_counts"]` tương ứng sẽ được tăng lên 1, và `vehicle_id` sẽ được thêm vào set `unique_ids`.
 - Nếu ID đã có trong set (do đã được đếm trước đó trong cùng lần đo này), thì bộ đếm sẽ không tăng nữa, dù xe đó vẫn còn trong khung hình hoặc di chuyển qua lại vạch.
 - Reset Bộ đếm: Khi kết thúc một chu kỳ đo lường lớn và bước vào pha trì hoãn (`start_delay_phase`), toàn bộ dữ liệu trong `vehicle_counting_data` (cả `cycle_counts` và `cycle_ids`) đều được xóa sạch để đảm bảo chu kỳ đo lường tiếp theo bắt đầu với bộ đếm mới hoàn toàn.
- **Minh họa qua Code**

```

● ● ●

# ... (Lấy unique_ids = vehicle_counting_data["cycle_ids"][direction_key][cycle_index])

for detection in current_detections:
    vehicle_id = detection['id']
    crossed_line_flag = detection['crossed_line'] # Lấy trạng thái vượt vạch từ detector

    # Điều kiện cốt lõi: Đã vượt vạch VÀ ID chưa nằm trong set đã đếm của lần này
    if crossed_line_flag and vehicle_id not in unique_ids:
        unique_ids.add(vehicle_id) # Thêm ID vào set để đánh dấu đã đếm

    # Tăng bộ đếm chính xác cho lần đếm này
    vehicle_counting_data["cycle_counts"][direction_key][cycle_index] += 1
    # ... (Ghi log, cập nhật biến đếm phụ nếu cần)

```

- **Ưu điểm:**

- Độ chính xác cao hơn: Giảm đáng kể việc đếm trùng so với việc chỉ đếm số lượng bounding box xuất hiện.
- Khả năng thích ứng: Logic dựa trên ID theo dõi và trạng thái vượt vạch giúp xử lý tốt hơn các tình huống xe di chuyển chậm, dừng lại hoặc thay đổi tốc độ gần vạch đếm.

- **Tích hợp và Giao tiếp:**

- Backend Fast API đóng vai trò trung tâm, điều phối hoạt động giữa việc nhận dữ liệu từ camera, xử lý bằng YOLOv8, thực thi logic đếm và tính toán thời gian đèn, quản lý trạng thái hệ thống, và giao tiếp với frontend qua WebSocket.
- Frontend cung cấp giao diện trực quan để người dùng theo dõi trạng thái hoạt động, xem kết quả xử lý (video, số liệu thống kê), và thực hiện các thao tác điều khiển cơ bản.
- Giao thức WebSocket đảm bảo cập nhật dữ liệu gần như tức thời từ backend lên frontend, tạo trải nghiệm giám sát động.

- **Kết luận:**

Phần mềm hệ thống, bao gồm cả backend xử lý và frontend giám sát, là yếu tố then chốt tạo nên chức năng hoàn chỉnh. Giao diện web cung cấp khả năng hiển thị trực quan và tương tác cần thiết, trong khi logic xử lý cốt lõi, đặc biệt là cơ chế đếm xe dựa trên việc vượt qua vạch đếm ảo kết hợp ID theo dõi, đảm bảo thu thập dữ liệu lưu lượng một cách chính xác và đáng tin cậy. Sự tích hợp chặt chẽ giữa các thành phần phần mềm thông qua WebSocket cho phép hệ thống hoạt động như một thể thống nhất, cung cấp thông tin và khả năng điều khiển cần thiết cho việc vận hành và đánh giá Hệ thống Điều tiết Giao thông Thông minh.

KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

1. Kết luận Tổng quan

Đề tài "Hệ thống Điều tiết Giao thông Thông minh sử dụng ESP32 và Camera IMOU" đã được thực hiện thành công, đạt được mục tiêu chính là xây dựng và kiểm nghiệm một giải pháp điều khiển đèn tín hiệu giao thông có khả năng thích ứng với mật độ phương tiện theo thời gian thực. Dự án đã chứng minh tính khả thi của việc tích hợp các công nghệ tiên tiến bao gồm thị giác máy tính (Computer Vision) với mô hình học sâu YOLOv8n, Internet of Things (IoT) thông qua nền tảng đám mây ThingSpeak, và hệ thống nhúng với vi điều khiển ESP32 để giải quyết bài toán tối ưu hóa luồng giao thông tại các nút giao.

Hệ thống đã hoàn thành việc triển khai đầy đủ các khái niệm sau:

- + Khối Thu thập Dữ liệu: Sử dụng Camera IP IMOU kết hợp với mô hình YOLOv8n chạy trên một bộ xử lý trung tâm, hệ thống đã thành công trong việc phát hiện, theo dõi và áp dụng logic đếm phương tiện dựa trên việc vượt qua vạch ảo, cung cấp dữ liệu định lượng về lưu lượng cho từng hướng giao thông.
- + Khối Xử lý và Ra Quyết định: Ứng dụng backend (FastAPI) đã xử lý hiệu quả dữ liệu đếm xe thu thập được, thực thi thuật toán điều chỉnh thời gian đèn xanh động. Thuật toán này đã cân bằng thành công giữa việc ưu tiên hướng có lưu lượng cao hơn và đảm bảo tính công bằng (qua tỷ lệ cơ bản) cũng như tính ổn định (qua cơ chế làm mịn), tính toán ra thời gian biểu đèn tối ưu cho chu kỳ tiếp theo.
- + Khối Điều khiển Thực thi: Module ESP32 đã thể hiện khả năng kết nối mạng Wifi ổn định, truy vấn dữ liệu thời gian đèn từ ThingSpeak, và điều khiển chính xác các đèn LED tín hiệu cùng màn hình hiển thị thời gian trên mô hình vật lý, phản ánh trực quan các quyết định từ thuật toán.

- + Khối Giám sát: Giao diện người dùng dựa trên web cung cấp khả năng giám sát trực quan, hiển thị luồng camera đã xử lý, trạng thái hệ thống, và các thông số thống kê quan trọng theo thời gian thực thông qua WebSocket.

Qua quá trình xây dựng mô hình thực nghiệm và giao diện phần mềm, hệ thống đã chứng minh được khả năng hoạt động end-to-end, từ việc thu thập hình ảnh, phân tích lưu lượng, tính toán thời gian đèn, đến việc điều khiển trực tiếp các tín hiệu vật lý. Logic đếm xe khi vượt qua vạch ảo kết hợp với ID theo dõi đã cho thấy hiệu quả trong việc giảm thiểu đếm trùng, cung cấp đầu vào đáng tin cậy hơn cho thuật toán điều khiển. Thuật toán điều chỉnh thời gian đèn, mặc dù dựa trên các quy tắc và heuristic, đã thể hiện khả năng phân bổ thời gian xanh một cách linh hoạt, hứa hẹn tiềm năng cải thiện hiệu suất giao thông so với hệ thống đèn cố định truyền thống.

2. Đánh giá Ưu điểm và Hạn chế

• Ưu điểm:

- Tính Thích ứng: Hệ thống có khả năng tự động điều chỉnh thời gian đèn dựa trên mật độ giao thông thực tế, có tiềm năng giảm ùn tắc và thời gian chờ đợi.
- Tích hợp Công nghệ Hiện đại: Ứng dụng thành công Computer Vision (YOLOv8n), IoT (ThingSpeak, Wi-Fi) và Hệ thống nhúng (ESP32) vào giải quyết bài toán thực tế.
- Chi phí Hợp lý (cho mô hình): Sử dụng các thành phần phổ biến như ESP32, Camera IMOU giúp giảm chi phí triển khai mô hình thử nghiệm.
- Khả năng Giám sát Trực quan: Giao diện web cung cấp cái nhìn tổng thể và chi tiết về hoạt động của hệ thống.
- Logic Đếm Cải tiến: Cơ chế đếm dựa trên vượt vạch và ID theo dõi giúp tăng độ chính xác so với các phương pháp đếm đơn giản.

• Hạn chế:

- Yêu cầu Xử lý: Việc chạy YOLOv8n đòi hỏi một bộ xử lý mạnh hơn ESP32 (PC/Laptop/SBC), làm tăng độ phức tạp kiến trúc và tiềm ẩn độ trễ.
- Phụ thuộc Đám mây: Việc sử dụng ThingSpeak làm trung gian giao tiếp giữa backend và ESP32 tạo ra sự phụ thuộc vào kết nối internet và dịch vụ bên ngoài, có thể ảnh hưởng đến độ tin cậy và tốc độ phản hồi.
- Độ chính xác của YOLOv8n: Mặc dù hiệu quả, YOLOv8n có thể gặp khó khăn trong điều kiện ánh sáng yếu, thời tiết xấu, hoặc khi phương tiện bị che khuất nhiều, ảnh hưởng đến độ chính xác đếm.
- Thuật toán Điều khiển: Thuật toán hiện tại chưa xem xét các yếu tố phức tạp như chiều dài hàng đợi, loại phương tiện, hoặc phối hợp với các nút giao lân cận.
- Tính chất Blocking của ESP32: Firmware ESP32 hiện tại có thể bị chặn trong quá trình thực thi chu kỳ đèn, hạn chế khả năng xử lý đa nhiệm.

3. Hướng Phát triển Tương lai

Dựa trên những kết quả đạt được và các hạn chế đã xác định, một số hướng phát triển tiềm năng cho hệ thống bao gồm:

- + Tối ưu hóa Xử lý: Nghiên cứu triển khai các mô hình nhẹ hơn hoặc kỹ thuật tối ưu hóa (quantization, pruning) để có thể chạy trên các thiết bị biên (Edge Computing) mạnh hơn ESP32 (như Jetson Nano, Raspberry Pi hiệu năng cao) đặt gần nút giao, giảm độ trễ mạng và tăng khả năng xử lý cục bộ.
- + Cải thiện Giao tiếp: Thay thế ThingSpeak bằng các phương thức giao tiếp trực tiếp giữa backend và ESP32 như MQTT hoặc xây dựng API riêng trên backend để ESP32 truy vấn, tăng độ tin cậy và giảm độ trễ.
- + Nâng cấp Thuật toán Điều khiển:
 1. Tích hợp thông tin về chiều dài hàng đợi (có thể ước tính từ YOLO hoặc cảm biến khác).

2. Phân loại phương tiện và áp dụng trọng số khác nhau (ưu tiên xe buýt, xe cứu thương).
 3. Nghiên cứu các thuật toán điều khiển nâng cao hơn như Fuzzy Logic, Reinforcement Learning để tối ưu hóa đa mục tiêu.
 4. Xem xét yếu tố phối hợp giữa các nút giao lân cận.
- + Cải thiện Độ chính xác Phát hiện: Tinh chỉnh (fine-tuning) mô hình YOLOv8n trên bộ dữ liệu thu thập tại địa phương; sử dụng các kỹ thuật tăng cường ảnh để xử lý điều kiện khó khăn; hoặc thử nghiệm các phiên bản YOLOv8n lớn hơn nếu tài nguyên cho phép.

4. Lời kết

Đề tài "Hệ thống Điều tiết Giao thông Thông minh sử dụng ESP32 và Camera IMOU" đã thành công trong việc xây dựng một nguyên mẫu hoạt động, minh chứng cho tiềm năng to lớn của việc ứng dụng các công nghệ Trí tuệ Nhân tạo và IoT vào lĩnh vực giao thông vận tải. Hệ thống không chỉ giải quyết được bài toán phát hiện và đếm xe tự động mà còn hiện thực hóa được ý tưởng điều khiển đèn tín hiệu một cách linh hoạt dựa trên dữ liệu thực tế. Mặc dù còn những hạn chế cần khắc phục và nhiều hướng để phát triển sâu hơn, dự án đã đặt nền móng vững chắc, cung cấp những kinh nghiệm quý báu và mở ra triển vọng về các giải pháp giao thông thông minh hơn, hiệu quả hơn trong tương lai, góp phần giải quyết vấn nạn ùn tắc tại các đô thị hiện đại.

TÀI LIỆU THAM KHẢO

- [1] Võ Thành Nam, Phùng Thị Thu Hương. (2022). Ứng dụng trí tuệ nhân tạo trong điều khiển giao thông đô thị. Tạp chí Phát triển Khoa học và Công nghệ, 5(3), 865-872.
- [2] Nguyễn Đình Thắng, Trần Quốc Tuấn. (2023). Giải pháp tối ưu hóa chu kỳ đèn giao thông dựa trên dữ liệu thời gian thực. Tạp chí Giao thông Vận tải, 67(2), 123-135.
- [3] Lê Hồng Quân, Phạm Minh Đức. (2024). Nghiên cứu ứng dụng mô hình YOLO trong nhận diện và phân loại phương tiện giao thông tại Việt Nam. Tạp chí Khoa học Công nghệ Giao thông Vận tải, 75(1), 45-58.
- [4] Đỗ Văn Hùng. (2023). ESP32: Ứng dụng và triển khai trong các hệ thống IoT. NXB Khoa học và Kỹ thuật, Hà Nội.
- [5] Ultralytics. (2024). YOLOv8 Documentation. Truy cập ngày 15/03/2025, từ <https://docs.ultralytics.com/>
- [6] Espressif Systems. (2023). ESP32 Technical Reference Manual. Truy cập ngày 20/02/2025, từ https://www.espressif.com/sites/default/files/documentation/esp32_technical_reference_manual_en.pdf
- [7] MathWorks. (2024). ThingSpeak Documentation. Truy cập ngày 10/03/2025, từ <https://www.mathworks.com/help/thingspeak/>
- [8] Trần Văn Bình, Nguyễn Thành Long. (2023). Các thuật toán tối ưu hóa chu kỳ đèn giao thông thích ứng. Tạp chí Khoa học và Công nghệ, 59(3), 245-257.
- [9] Dahua Technology. (2024). IMOU Camera User Manual. Truy cập ngày 05/02/2025, từ <https://www.imoulife.com/support/download/>
- [10] Đinh Xuân Trường, Hoàng Văn Minh. (2023). Đánh giá hiệu quả của các hệ thống giao thông thông minh tại Việt Nam: Thách thức và giải pháp. Kỷ yếu Hội nghị Khoa học Quốc gia về Giao thông Thông minh, Hà Nội, 112-125.