**TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG VIỆT – HÀN**

**KHOA KỸ THUẬT MÁY TÍNH VÀ ĐIỆN TỬ**



**ĐỒ ÁN TỐT NGHIỆP**

**ĐỀ TÀI: THIẾT KẾ VÀ PHÁT TRIỂN CỤM ĐỒNG HỒ HIỂN THỊ THỜI GIAN THỰC TÍCH HỢP AI CHO HỆ THỐNG HỖ TRỢ LÁI XE NÂNG CAO**

|  |  |
| --- | --- |
| **Họ và tên sinh viên:** | Trần Văn Quốc Đạt – 21CE077 |
| **Lớp:** | 21CE2 |
| **Ngành:** | Công nghệ kỹ thuật máy tính |
| **Giảng viên hướng dẫn:** | ThS. Nguyễn Thị Huyền Trang |

Đà Nẵng, tháng 1, năm 2026

**TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG VIỆT – HÀN**

**KHOA KỸ THUẬT MÁY TÍNH VÀ ĐIỆN TỬ**



**ĐỒ ÁN TỐT NGHIỆP**

**ĐỀ TÀI: THIẾT KẾ VÀ PHÁT TRIỂN CỤM ĐỒNG HỒ HIỂN THỊ THỜI GIAN THỰC TÍCH HỢP AI CHO HỆ THỐNG HỖ TRỢ LÁI XE NÂNG CAO**

|  |  |
| --- | --- |
| **Họ và tên sinh viên:** | Trần Văn Quốc Đạt – 21CE077 |
| **Lớp:** | 21CE2 |
| **Ngành:** | Công nghệ kỹ thuật máy tính |
| **Giảng viên hướng dẫn:** | ThS. Nguyễn Thị Huyền Trang |

Đà Nẵng, tháng 1, năm 2026

**LỜI CẢM ƠN**

Lời đầu tiên, em xin bày tỏ lòng biết ơn sâu sắc đến Ban giám hiệu Trường Đại học Công nghệ Thông tin và Truyền thông Việt Hàn, cùng toàn thể quý thầy cô Khoa Kỹ thuật máy tính và điện tử đã tạo điều kiện thuận lợi và truyền đạt những kiến thức quý báu cho em trong suốt những năm học vừa qua.

Đặc biệt, em xin gửi lời cảm ơn chân thành và sâu sắc nhất tới cô Nguyễn Thị Huyền Trang. Trong suốt quá trình thực hiện đồ án, cô đã luôn dành thời gian quan tâm, định hướng và đưa ra những chỉ dẫn chuyên môn. Sự nhiệt tình và những góp ý khắt khe nhưng đầy tâm huyết của cô đã giúp em vượt qua nhiều thách thức kỹ thuật để hoàn thiện hệ thống Instrument Cluster một cách tốt nhất.

Em cũng xin gửi lời cảm ơn tới gia đình, những người đã luôn là điểm tựa tinh thần và tạo mọi điều kiện tốt nhất để em yên tâm học tập.

Cuối cùng, em xin cảm ơn những người bạn đã luôn sát cánh, chia sẻ kinh nghiệm và hỗ trợ em trong quá trình thiết kế mạch và huấn luyện mô hình AI.

Dù đã dành nhiều tâm huyết và nỗ lực để hoàn thành đồ án, nhưng do kiến thức và kinh nghiệm còn hạn chế, báo cáo chắc chắn không tránh khỏi những thiếu sót. Em rất mong nhận được sự chỉ bảo và đóng góp ý kiến của quý thầy cô trong hội đồng để đề tài được hoàn thiện hơn.

Em xin chân thành cảm ơn!

NHẬN XÉT

(Nhận xét của giảng viên hướng dẫn)

………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………….

………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………….………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………….………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………….

Đà Nẵng, ngày … tháng 1 năm 2026

Ký tên

MỤC LỤC

[MỞ ĐẦU 3](#_Toc217954271)

[CHƯƠNG I: TỔNG QUAN VÀ CƠ SỞ LÝ THUYẾT 1](#_Toc217954272)

[1.1 Tổng quan về hệ thống hỗ trợ lái xe nâng cao 1](#_Toc217954273)

[1.1.1 Định nghĩa, lịch sử phát triển 1](#_Toc217954274)

[1.1.2 Các chức năng chính của ADAS 2](#_Toc217954275)

[1.1.3 Vai trò của cụm đồng hồ trong hệ thống ADAS 3](#_Toc217954276)

[1.2 Trí tuệ nhân tạo và thị giác máy tính 3](#_Toc217954277)

[1.2.1 Các mô hình nhận diện đối tượng phổ biến 3](#_Toc217954278)

[1.2.2 Giới thiệu về mạng nơ-ron tích chập (CNN) 4](#_Toc217954279)

[1.2.3 TensorFlow Lite cho các ứng dụng nhúng 5](#_Toc217954280)

[1.3 Hệ thống nhúng và các chuẩn giao thức 6](#_Toc217954281)

[1.3.1 Vi điều khiển STM32F103C8T6 6](#_Toc217954282)

[1.3.2 Vi điều khiển ESP32 8](#_Toc217954283)

[1.3.3 Raspberry Pi 4 9](#_Toc217954284)

[1.3.4 Giao thức CAN bus 10](#_Toc217954285)

[1.3.5 Giao thức UART 14](#_Toc217954286)

[1.3.6 Giao thức UDP 16](#_Toc217954287)

[CHƯƠNG II: PHÂN TÍCH VÀ THIẾT KẾ HỆ THỐNG 17](#_Toc217954288)

[2.1 Phân tích yêu cầu hệ thống 17](#_Toc217954289)

[2.1.1 Yêu cầu chức năng 17](#_Toc217954290)

[2.1.2 Yêu cầu phi chức năng 17](#_Toc217954291)

[2.2 Kiến trúc hệ thống 18](#_Toc217954292)

[2.2.1 Sơ đồ khối hệ thống 18](#_Toc217954293)

[2.2.2 Lưu đồ thuật toán từng node 19](#_Toc217954294)

[2.3 Thiết kế phần cứng 24](#_Toc217954295)

[2.3.1 Lựa chọn linh kiện 24](#_Toc217954296)

[2.3.3 Thiết kế sơ đồ nguyên lý 26](#_Toc217954297)

[2.3.4 Thiết kế mạch in 29](#_Toc217954298)

[2.4 Thiết kế phần mềm 33](#_Toc217954299)

[2.4.1 Thiết kế chức năng xử lý hình ảnh 33](#_Toc217954300)

[2.4.2 Thiết kế module điều khiển 36](#_Toc217954301)

[2.4.3 Thiết kế giao diện người dùng 37](#_Toc217954302)

[2.5 Cấu hình giao thức giao tiếp 41](#_Toc217954303)

[2.5.1 Định dạng các gói tin CAN 41](#_Toc217954304)

[2.5.2 Cấu hình UART 42](#_Toc217954305)

[2.5.3 Cấu hình UDP 42](#_Toc217954306)

[CHƯƠNG III: TRIỂN KHAI VÀ ĐÁNH GIÁ HỆ THỐNG 44](#_Toc217954307)

[3.1 Quy trình triển khai hệ thống 44](#_Toc217954308)

[3.1.1 Công cụ và môi trường phát triển 44](#_Toc217954309)

[3.1.2 Quy trình tích hợp hệ thống 45](#_Toc217954310)

[3.2 Triển khai phần mềm 47](#_Toc217954311)

[3.2.1 Lập trình các node điều khiển 47](#_Toc217954312)

[3.2.2 Xây dựng Gateway 48](#_Toc217954313)

[3.2.3 Triển khai khối xử lý trung tâm 50](#_Toc217954314)

[3.2.4 Phát triển giao diện 51](#_Toc217954315)

[3.3 Triển khai phần cứng 52](#_Toc217954316)

[3.3.1 Quy trình thực hiện 52](#_Toc217954317)

[3.3.2 Mô hình thực tế 54](#_Toc217954318)

[3.4 Kiểm thử và đánh giá kết quả 54](#_Toc217954319)

[3.4.1 Kịch bản và môi trường kiểm thử 54](#_Toc217954320)

[3.4.2 Kết quả thực nghiệm 54](#_Toc217954321)

[3.4.3 Đánh giá tổng thể hệ thống 55](#_Toc217954322)

[KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN 57](#_Toc217954323)

[I. Kết luận 57](#_Toc217954324)

[II. Hướng phát triển 57](#_Toc217954325)

[TÀI LIỆU THAM KHẢO 59](#_Toc217954326)

**DANH MỤC HÌNH ẢNH**

[Hình 1.1 Sơ đồ chân STM32F103C8T6 7](#_Toc217956106)

[Hình 1.2 Sơ đồ chân ESP32 8](#_Toc217956107)

[Hình 1.3 Raspberry Pi 4 9](#_Toc217956108)

[Hình 1.4 Cấu trúc kết nối mạng CAN bus 10](#_Toc217956109)

[Hình 1.5 Minh họa tín hiệu vi sai 10](#_Toc217956110)

[Hình 1.6 Khả năng kháng nhiễu của CAN 11](#_Toc217956111)

[Hình 1.7 Sơ đồ một nút mạng CAN 12](#_Toc217956112)

[Hình 1.8 Cấu trúc khung dữ liệu 13](#_Toc217956113)

[Hình 1.9 Cấu trúc khung truyền UART 15](#_Toc217956114)

[Hình 1.10 Sơ đồ kết nối UART 15](#_Toc217956115)

[Hình 2.1 Sơ đồ tổng quan của hệ thống 19](#_Toc217956116)

[Hình 2.2 Lưu đồ thuật toán Body Control Node 20](#_Toc217956117)

[Hình 2.3 Lưu đồ thuật toán Motor Control Node 21](#_Toc217956118)

[Hình 2.4 Lưu đồ thuật toán Gateway Node 22](#_Toc217956119)

[Hình 2.5 Lưu đồ thuật toán Central Unit 23](#_Toc217956120)

[Hình 2.6 Schematic Diagram Body Control Node 26](#_Toc217956121)

[Hình 2.7 Sơ đồ nguyên lý Motor Control Node 27](#_Toc217956122)

[Hình 2.8 Sơ đồ nguyên lý Gateway Node 28](#_Toc217956123)

[Hình 2.9 Sơ đồ nguyên lý mạch nguồn 29](#_Toc217956124)

[Hình Layout mạch Body Control Node 30](#_Toc217956125)

[Hình Mô phỏng 3D mạch Body Control node 30](#_Toc217956126)

[Hình Layout mạch Motor Control Node 31](#_Toc217956127)

[Hình Mô phỏng 3D Motor Control Node 31](#_Toc217956128)

[Hình Layout mạch Gateway Node 32](#_Toc217956129)

[Hình Layout và mô phỏng 3D Gateway Node 32](#_Toc217956130)

[Hình Layout mạch nguồn 33](#_Toc217956131)

[Hình Mô phỏng 3D mạch nguồn 33](#_Toc217956132)

[Hình 2.10 Cây thư mục giao diện 38](#_Toc217956133)

[Hình 2.11 Data flow diagram 39](#_Toc217956134)

[Hình 2.12 Giao diện khởi động 40](#_Toc217956135)

[Hình 2.13 Giao diện Instrument Cluster 40](#_Toc217956136)

[Hình 3.1 Lưu đồ quy trình triển khai hệ thống 45](#_Toc217956137)

**DANH MỤC BẢNG BIỂU**

[Bảng 2.1 So sánh MCU cho các node điều khiển 24](#_Toc217954359)

[Bảng 2.2 So sánh MCU cho Gateway Node 24](#_Toc217954360)

[Bảng 2.3 So sánh máy tính nhúng cho Central Unit 25](#_Toc217954361)

[Bảng 2.4 So sánh thiết bị cho giao tiếp CAN 25](#_Toc217954362)

[Bảng 2.5 So sánh lựa chọn cảm biến 25](#_Toc217954363)

[Bảng 2.6 Dữ liệu Body Control Node 41](#_Toc217954364)

[Bảng 2.7 Dữ liệu Motor Control Node 42](#_Toc217954365)

[Bảng 2.8 Định dạng gói tin giao tiếp UDP nội bộ 43](#_Toc217954366)

**DANH MỤC TỪ VIẾT TẮT**

|  |  |
| --- | --- |
| ADAS | Advanced Driving Assistance System |
| AI | Artificial Intelligence |
| CNN | Convolutional Neural Network |
| ELU | Exponential Linear Unit |
| ReLU | Rectified Linear Unit |
| SSD | Single Shot Multilbox Detector |
| YOLO | You Only Look Once |
| TFLite | TensorFlow Lite |
| SoC | System on Chip |
| UART | Universal Asynchronous Receiver/Transmitter |
| ECM | Engine Control Module |
| BCN | Body Control Node |
| HAL | Hardware Abstraction Layer |
| PWM | Pulse Width Modulation |
| QML | Qt Modeling Language |
| UDP | User Datagram Protocol |

# MỞ ĐẦU

1. Lý do chọn đề tài

Sự cách mạng về ô tô là một cuộc chuyển đổi từ phương tiện xa xỉ thành phổ biến, bắt đầu với phát minh xe chạy xăng của Karl Benz (1886), rồi được Henry Ford dân chủ hóa với dây chuyền lắp ráp (Model T, 1908), và ngày nay đang bước sang kỷ nguyên mới với xe điện, xe hybrid, công nghệ tự lái, kết nối thông minh, làm thay đổi sâu sắc kinh tế, xã hội và môi trường, hướng tới sự bền vững và an toàn hơn.

Trong bối cảnh đó, bảng đồng hồ ô tô không còn đơn thuần được thiết kế cơ học mà dần chuyển sang màn hình kỹ thuật số, tích hợp công nghệ số hóa, hiển thị thông tin đa dạng, tùy biến giao diện và kết nối thông minh mang lại trải nghiện lái xe hiện đại, tùy chỉnh cao và thông tin trực quan hơn rất nhiều so với taplo cũ.

1. Mục tiêu nghiên cứu

Mục tiêu trọng tâm của đề tài là thiết kế và xây dựng thành công một mô hình mẫu hệ thống bảng đồng hồ kỹ thuật số có khả năng hoạt động ổn định trong môi trường thời gian thực. Cụ thể, hệ thống phải thực hiện được việc thu thập và giải mã chính xác dữ liệu từ các cảm biến thông qua giao thức truyền thông CAN, sau đó hiển thị lên màn hình với giao diện đồ họa trực quan và mượt mà. Bên cạnh đó, đề tài hướng tới việc tích hợp một model trí tuệ nhân tạo có khả năng phân tích hình ảnh từ camera để đưa ra các cảnh báo an toàn thông minh, từ đó chứng minh khả năng xử lý của các hệ thống AI trên thiết bị nhúng trong lĩnh vực điện tử ô tô.

1. Đối tượng và phạm vi nghiên cứu

Đối tượng nghiên cứu của đề tài tập trung vào kiến trúc hệ thống nhúng hiệu năng cao, các giao thức truyền thông tiêu chuẩn trong công nghiệp ô tô như CAN bus, và các thư viện đồ họa tối ưu cho màn hình hiển thị. Ngoài ra, nghiên cứu còn đi sâu vào các thuật toán thị giác máy tính và mô hình máy học (Machine Learning) được tối ưu hóa cho phần cứng có tài nguyên hạn chế. Về phạm vi nghiên cứu, đề tài giới hạn trong việc phát triển một hệ thống mẫu thực nghiệm, hoạt động trong môi trường phòng thí nghiệm với các dữ liệu mô phỏng từ ECU và camera hành trình, chưa thực hiện thử nghiệm trực tiếp trên xe thương mại trong các điều kiện giao thông thực tế khắc nghiệt.

1. Phương pháp nghiên cứu
2. Phạm vi về chức năng

Đề tài sẽ tập trung vào việc phát triển các chức năng cơ bản và nâng cao sau:

* Chức năng cơ bản: Thu thập và hiển thị các thông số lái xe (các đèn tín hiệu, tốc độ động cơ, biển báo) chính xác và tức thời.
* Chức năng AI tích hợp: Nhận diện biển báo giao thông: Sử dụng TFLite, camera để nhận diện các biển báo phổ biến.
* Chức năng kết nối: Tập trung vào việc giao tiếp với các ECU khác của xe thông qua giao thức CAN bus. Đề tài sẽ không đi sâu vào các giao thức phức tạp khác như LIN, FlexRay.

1. Phạm vi về công nghệ

* Nền tảng phần cứng: Sử dụng vi điều khiển (STM32, ESP32) và bo mạch phát triển nhúng (Raspberry Pi) để làm phần cứng trung tâm.
* Môi trường lập trình: Sử dụng các ngôn ngữ lập trình như C/C++ cho phần mềm nhúng và Python cho các model AI.
* Thư viện/Framework: Sử dụng thư viện đồ họa Qt để phát triển giao diện. Các thư viện AI như TensorFlow Lite và OpenCV sẽ được dùng để xử lý hình ảnh và dữ liệu.

1. Ý nghĩa khoa học và thực tiễn

Về mặt khoa học, đề tài góp phần nghiên cứu giải pháp tích hợp đa dịch vụ trên một nền tảng nhúng duy nhất, từ xử lý tín hiệu số, đồ họa thời gian thực đến tính toán trí tuệ nhân tạo. Kết quả nghiên cứu là tài liệu tham khảo có giá trị về việc tối ưu hóa mô hình AI trên các thiết bị nhúng công suất thấp. Về mặt thực tiễn, sản phẩm của đề tài mang lại một giải pháp hiển thị thông tin lái xe hiện đại, có tính tùy biến cao và tích hợp sẵn các tính năng an toàn thông minh. Đây là tiền đề để phát triển lĩnh vực ô tô nội địa, hỗ trợ quá trình chuyển đổi số trong ngành giao thông vận tải và mở ra hướng ứng dụng rộng rãi cho các phương tiện xe điện thông minh tại Việt Nam trong tương lai.

Xuất phát từ ứng dụng quan trọng trên, em đã chọn đề tài “Thiết kế và phát triển cụm đồng hồ hiển thị thời gian thực tích hợp AI cho hệ thống hỗ trợ lái xe nâng cao.”

# CHƯƠNG I: TỔNG QUAN VÀ CƠ SỞ LÝ THUYẾT

## 1.1 Tổng quan về hệ thống hỗ trợ lái xe nâng cao

### 1.1.1 Định nghĩa, lịch sử phát triển

a. Định nghĩa

Hệ thống hỗ trợ lái xe nâng cao (ADAS) là tập hợp các công nghệ an toàn chủ động được trang bị trên ô tô nhằm hỗ trợ người lái, giảm thiểu nguy cơ tai nạn, đồng thời mang lại trải nghiệm lái xe an toàn và tiện nghi hơn.

Các hệ thống ADAS hoạt động nhờ vào sự kết hợp giữa camera, radar, cảm biến siêu âm và phần mềm điều khiển thông minh, từ đó giúp xe phát hiện nguy cơ, đưa ra cảnh báo, thậm chí can thiệp tự động vào quá trình điều khiển (phanh, đánh lái, tăng tốc…) khi cần thiết.

b. Lịch sử phát triển

Lịch sử của ADAS là một hành trình tiến hóa từ các cơ chế an toàn cơ học đơn giản đến các hệ thống trí tuệ nhân tạo phức tạp ngày nay. Quá trình này có thể được chia thành các giai đoạn chính sau:

* Giai đoạn sơ khai (Thập niên 1950 - 1970): Những bước đi đầu tiên của ADAS bắt đầu với sự ra đời của hệ thống kiểm soát hành trình vào những năm 1950, giúp xe duy trì tốc độ ổn định trên đường cao tốc. Đến những năm 1970, hệ thống chống bó cứng phanh bắt đầu xuất hiện, đánh dấu sự can thiệp đầu tiên của điện tử vào quá trình điều khiển phanh để đảm bảo an toàn.
* Giai đoạn tích hợp cảm biến (Thập niên 1980 - 1990): Trong giai đoạn này, các cảm biến bắt đầu được sử dụng rộng rãi hơn. Hệ thống kiểm soát lực kéo và kiểm soát ổn định điện tử ra đời, giúp xe vận hành an toàn trong các điều kiện đường trơn trượt hoặc khi vào cua gấp.
* Giai đoạn phát triển cảm biến thông minh (Thập niên 2000 - 2010): Đây là thời điểm bùng nổ của các công nghệ hiện đại như Radar và Camera. Các tính năng như cảnh báo chệch làn đường, hỗ trợ giữ làn và phanh khẩn cấp tự động bắt đầu được trang bị trên các dòng xe cao cấp. Hệ thống bắt đầu có khả năng "nhìn" và "nhận biết" môi trường xung quanh thay vì chỉ phản ứng với các thông số vật lý của xe.
* Giai đoạn trí tuệ nhân tạo và kết nối (Thập niên 2020 đến nay): Hiện nay, ADAS đã bước sang một trang mới nhờ sự tích hợp của Trí tuệ nhân tạo và Machine Learning. Các hệ thống ADAS hiện đại không chỉ đưa ra cảnh báo mà còn có khả năng tự động xử lý các tình huống phức tạp như tự động đỗ xe, hỗ trợ lái xe tự hành cấp độ cao và giao tiếp với các phương tiện xung quanh. Việc tích hợp AI giúp hệ thống có khả năng tự học từ dữ liệu thực tế, từ đó nâng cao độ chính xác và tin cậy trong việc bảo vệ người lái.

### 1.1.2 Các chức năng chính của ADAS

Hệ thống ADAS thực hiện các chức năng từ cảnh báo thụ động đến can thiệp chủ động vào hệ thống điều khiển của xe. Các chức năng này có thể được chia thành các nhóm chính sau:

a. Nhóm chức năng cảnh báo

Đây là nhóm chức năng cơ bản sử dụng âm thanh, hình ảnh để thông báo cho tài xế về các mối nguy hiểm tiềm ẩn:

* Cảnh báo lệch làn đường: Nhận diện vạch kẻ đường và cảnh báo khi xe có dấu hiệu đi chệch khỏi làn đường mà không có tín hiệu xi-nhan.
* Cảnh báo va chạm phía trước: Sử dụng radar để đo khoảng cách với xe phía trước và cảnh báo nếu có nguy cơ xảy ra va chạm.
* Cảnh báo điểm mù: Phát hiện các phương tiện nằm trong vùng mù của gương chiếu hậu và đưa ra tín hiệu cảnh báo trên mặt gương.

b. Nhóm chức năng hỗ trợ kiểm soát

Các chức năng này can thiệp trực tiếp vào hệ thống phanh, ga hoặc vô lăng để hỗ trợ người điều khiển:

* Kiểm soát hành trình thích ứng: Không chỉ duy trì tốc độ cố định mà còn tự động điều chỉnh tốc độ để giữ khoảng cách an toàn với xe phía trước.
* Phanh khẩn cấp tự động: Nếu người lái không kịp phản ứng sau cảnh báo va chạm, hệ thống sẽ tự động kích hoạt lực phanh để giảm thiểu thiệt hại hoặc tránh tai nạn.
* Hỗ trợ giữ làn đường: Chủ động điều chỉnh vô lăng để đưa xe trở lại giữa làn đường nếu xe có xu hướng đi chệch làn.
* Hỗ trợ đỗ xe tự động: Tự động điều khiển vô lăng, phanh và ga để đưa xe vào vị trí đỗ (song song hoặc vuông góc) một cách chính xác.

c. Nhóm chức năng cải thiện tầm nhìn và nhận diện

Sử dụng camera và AI để xử lý thông tin môi trường:

* Nhận diện biển báo giao thông: Nhận diện các biển báo giới hạn tốc độ, cấm vượt... và hiển thị trực tiếp lên màn hình Instrument Cluster để nhắc nhở người lái.
* Đèn pha thích ứng: Tự động điều chỉnh luồng sáng để tối ưu tầm nhìn vào ban đêm mà không gây chói mắt cho phương tiện đi ngược chiều.
* Hệ thống quan sát ban đêm: Sử dụng camera nhiệt để phát hiện người đi bộ hoặc động vật trong điều kiện ánh sáng yếu vượt ngoài tầm đèn pha.

### 1.1.3 Vai trò của cụm đồng hồ trong hệ thống ADAS

Cụm đồng hồ kỹ thuật số đóng vai trò là trung tâm hiển thị thông tin trực quan cho người lái, biến dữ liệu phức tạp từ các cảm biến thành cảnh báo dưới dạng hình ảnh, âm thanh, rung động. Trong hệ sinh thái ADAS, thực hiện các nhiệm vụ chiến lược sau:

* Trực quan hóa dữ liệu cảm biến: Chuyển đổi các dữ liệu kỹ thuật phức tạp từ Radar, LiDar và Camera thành các tín hiệu hình ảnh dễ hiểu như mô phỏng làn đường, khoảng cách với xe phía trước hoặc các vật cản xung quanh xe.
* Truyền tải cảnh báo tức thời: Khi các chức năng ADAS phát hiện nguy hiểm, cụm đồng hồ là nơi đầu tiên phát ra các cảnh báo thị giác thông qua màu sắc và biểu tượng nhấp nháy, giúp người lái phản xạ nhanh chóng trước khi hệ thống can thiệp tự động.
* Quản lý trạng thái vận hành của hệ thống: Thông báo cho người lái biết các tính năng hỗ trợ nào đang hoạt động (ví dụ: Adaptive Cruise Control đang ở chế độ chờ hay đang giữ khoảng cách), giúp người lái tin tưởng và kiểm soát tốt hơn các tính năng tự động hóa.
* Tối ưu hóa sự tập trung của người lái: Bằng cách tích hợp trí tuệ nhân tạo để ưu tiên hiển thị các thông tin quan trọng nhất tùy theo ngữ cảnh lái xe, cụm đồng hồ giúp giảm bớt sự quá tải thông tin, đảm bảo người lái luôn nhận được thông báo cần thiết mà không bị xao nhãng.
* Cầu nối tương tác thông minh: Với sự tích hợp AI, cụm đồng hồ đóng vai trò hiển thị các phân tích hành vi tài xế (như cảnh báo buồn ngủ), tạo ra một vòng lặp phản hồi giúp điều chỉnh thói quen lái xe an toàn hơn.

## 1.2 Trí tuệ nhân tạo và thị giác máy tính

### 1.2.1 Các mô hình nhận diện đối tượng phổ biến

Trong lĩnh vực thị giác máy tính, bài toán nhận diện đối tượng không chỉ dừng lại ở việc phân loại hình ảnh mà còn phải xác định chính xác vị trí của đối tượng đó trong khung hình thông qua các hộp bao. Hiện nay, các mô hình nhận diện đối tượng dựa trên học sâu được chia làm hai kiến trúc chính dựa trên quy trình xử lý: Mô hình hai giai đoạn và Mô hình một giai đoạn.

1. Mô hình hai giai đoạn

Mô hình hai giai đoạn giải quyết vấn đề phức tạp bằng cách chia nhỏ thành hai giai đoạn để xử lý

* Giai đoạn 1: Mạng nơ-ron trích xuất các đặc trưng và đề xuất các vùng có khả năng chứa đối tượng.
* Giai đoạn 2: Các vùng đề xuất được đưa vào mạng con để phân loại đối tượng và tinh chỉnh tọa độ hộp bao.

Các mô hình tiêu biểu thuộc nhóm này như R-CNN, Fast R-CNN, và phổ biến nhất là Faster R-CNN.

Nhóm mô hình này đạt độ chính xác rất cao, đặc biệt với các đối tượng nhỏ hoặc bị che khuất. Tuy nhiên, do cấu trúc phức tạp và tính toán tuần tự qua hai bước nên tốc độ suy luận thường chậm (thường dưới 10 FPS trên phần cứng thông dụng), khó đáp ứng yêu cầu thời gian thực trên các thiết bị nhúng.

1. Mô hình một giai đoạn

Khác với mô hình hai giai đoạn, các mô hình một giai đoạn loại bỏ hoàn toàn bước xác định vị trí vật thể. Thay vào đó, chúng gộp hai bước vào chung một mạng nơ-ron và xử lý tất cả trong một lần lan truyền từ đầu vào đến đầu ra. Điều này có nghĩa mô hình này không cần sinh vùng đề xuất, không cần thêm giai đoạn để xử lý, ảnh được đưa vào sau đó mô hình trích xuất đặc trưng và trả về vùng bao và class ngay ở bước cuối. Đây là nhóm mô hình phù hợp nhất cho các bài toán nhúng và thời gian thực nhờ kiến trúc tinh gọn.

* YOLO: Là dòng mô hình nổi tiếng nhất trong nhóm này. YOLO chia ảnh đầu vào thành một lưới S x S. Nếu tâm của đối tượng rơi vào ô lưới nào, ô lưới đó sẽ chịu trách nhiệm phát hiện đối tượng. Các phiên bản cải tiến liên tục tối ưu hóa sự cân bằng giữa tốc độ và độ chính xác. Mô hình này có tốc độ xử lý cực nhanh và kích thước mô hình nhẹ.
* SSD: Mô hình giải quyết vấn đề phát hiện các đối tượng có kích thước khác nhau bằng cách sử dụng các feature map ở nhiều tỷ lệ khác nhau. SSD thường được kết hợp với các trích xuất đặc trưng nhẹ như MobileNet để chạy hiệu quả trên thiết bị có tài nguyên hạn chế.

### 1.2.2 Giới thiệu về mạng nơ-ron tích chập (CNN)

CNN là một trong những mô hình học sâu tiên tiến nhất, được thiết kế đặc biệt để xử lý các dữ liệu có cấu trúc lưới như hình ảnh kỹ thuật số. Trong các hệ thống hỗ trợ lái xe nâng cao, CNN đóng vai trò là bộ não thị giác, cho phép hệ thống phân tích và hiểu các dữ liệu hình ảnh thu được từ camera hành trình.

Cấu trúc cơ bản của một mạng CNN bao gồm các lớp thành phần chính sau:

1. Lớp tích chập (Convolutional Layer)

Đây là thành phần cốt lõi của mạng, thực hiện các phép tính tích chập giữa dữ liệu hình ảnh đầu vào và các bộ lọc. Quá trình này giúp mạng tự động trích xuất các đặc trưng từ thấp đến cao của hình ảnh, từ các đường nét, góc cạnh đơn giản đến các hình khối và thực thể phức tạp.

Trong quá trình thực hiện tích chập nếu quá trình di chuyển “nhảy cóc” một giá trị S nào đó thì ảnh đầu ra có kích thước giảm so với ảnh đầu vào gọi là Stride = S. Ở các lề của ảnh thiếu các pixel để tạo nên lân cận ta thêm P pixel ở lề, gọi Padding = P

Nếu gọi P là số pixel thêm vào lề, S là giá trị stride, m x n là kích thước ảnh đầu vào, k x k là kích thước mặt nạ, kích thước ảnh sau khi tích chập là

Đầu ra của lớp tích chập là các đặc trưng ảnh đầu vào gọi là feature map hay activation map. Sau khi có được K activation map thì chúng được xếp chồng lên nhau để tạo thành đầu vào của lớp tiếp theo.

1. Lớp hàm kích hoạt (Activation Function)

Áp dụng các hàm kích hoạt như ELU, Leaky ReLU, Tanh,… nhưng thường sử dụng hàm ReLU (Rectified Linear Unit) để đưa tính phi tuyến vào mô hình, giúp mạng có khả năng học và biểu diễn được các cấu trúc dữ liệu phức tạp hơn.

1. Lớp gộp (Pooling Layer)

Lớp gộp thường được dùng giữa các lớp tích chập có nhiệm vụ giảm kích thước không gian của các bản đồ đặc trưng sau khi tích chập. Quá trình này giúp giảm số lượng tham số tính toán, từ đó giảm bớt khối lượng tính toán cho hệ thống nhúng và hạn chế hiện tượng overfitting.

1. Lớp kết nối đầy đủ (Fully Connected Layer)

Các đặc trưng sau khi được trích xuất và giảm chiều sẽ được phẳng hóa (flatten) và đưa vào các lớp kết nối đầy đủ để thực hiện nhiệm vụ phân loại cuối cùng, chẳng hạn như xác định vật thể đó là biển báo giao thông, người đi bộ hay phương tiện khác.

### 1.2.3 TensorFlow Lite cho các ứng dụng nhúng

Trong bối cảnh triển khai các mô hình học sâu lên các thiết bị biên và hệ thống nhúng, thách thức lớn nhất nằm ở sự hạn chế về tài nguyên tính toán, dung lượng bộ nhớ và yêu cầu về năng lượng. TFLite được Google phát triển như một giải pháp chuyên biệt để giải quyết các vấn đề này, cho phép thực thi suy luận trực tiếp trên thiết bị với độ trễ thấp.

1. Kiến trúc và nguyên lý hoạt động

TFLite không dùng để huấn luyện mô hình mà tập trung hoàn toàn vào việc tối ưu hóa quá trình suy luận. Quy trình triển khai bao gồm hai thành phần chính:

* Bộ chuyển đổi: Chuyển đổi mô hình TensorFlow gốc (thường ở định dạng SavedModel hoặc Keras H5 sử dụng số thực 32-bit) sang định dạng .tflite. Trong quá trình này, bộ chuyển đổi sẽ loại bỏ các phép toán không cần thiết và tối ưu hóa biểu đồ tính toán.
* Bộ thông dịch: Là một thư viện gọn nhẹ chạy trên thiết bị nhúng (như Raspberry Pi, vi điều khiển STM32, ESP32). Bộ thông dịch này tải mô hình .tflite và thực thi tính toán dựa trên phần cứng sẵn có của thiết bị, hỗ trợ tăng tốc phần cứng.

1. Kỹ thuật lượng tử hóa

Một trong những tính năng quan trọng nhất của TFLite đối với hệ thống nhúng là kỹ thuật lượng tử hóa. Kỹ thuật này giảm độ chính xác của các tham số trọng số và kích hoạt từ số thực dấu phẩy động 32-bit xuống các định dạng nhỏ hơn như số nguyên 8-bit (Int8) hoặc số thực 16-bit (Float16).

Lợi ích của lượng tử hóa trong bài toán ADAS/Instrument Cluster:

* Giảm kích thước mô hình: Mô hình Int8 thường nhỏ hơn 4 lần so với mô hình Float32, giúp tiết kiệm bộ nhớ lưu trữ trên thẻ nhớ hoặc Flash.
* Tăng tốc độ suy luận: Các phép toán trên số nguyên 8-bit thực hiện nhanh hơn đáng kể so với số thực trên các kiến trúc CPU ARM (như Cortex-A trên Raspberry Pi), giúp tăng chỉ số FPS.
* Giảm độ trễ: Phù hợp với yêu cầu thời gian thực của hệ thống hỗ trợ lái xe, nơi độ trễ xử lý có thể ảnh hưởng đến an toàn.

## 1.3 Hệ thống nhúng và các chuẩn giao thức

Để xây dựng một hệ thống đồng hồ táp-lô kỹ thuật số tích hợp trí tuệ nhân tạo, việc lựa chọn phần cứng xử lý và phương thức giao tiếp đóng vai trò then chốt. Phần này trình bày tổng quan về các thiết bị phần cứng (STM32, ESP32, Raspberry Pi) và các chuẩn giao tiếp (CAN, UART) được sử dụng trong hệ thống.

### 1.3.1 Vi điều khiển STM32F103C8T6

1. Giới thiệu

STM32F103C8T6 là một vi điều khiển 32-bit thuộc dòng STM32 của STMicroelectronics, dựa trên kiến trúc ARM Cortex-M3. Đây là một vi điều khiển mạnh mẽ, được sử dụng rộng rãi trong các ứng dụng nhúng, từ thiết bị điện tử tiêu dùng đến tự động hóa công nghiệp, nhờ vào hiệu năng tốt, kích thước nhỏ gọn và khả năng tiêu thụ điện năng thấp. Trong đồ án này, nó đóng vai trò là khối xử lý thời gian thực, thực hiện nhiệm vụ thu thập các giá trị từ cảm biến và nút nhấn.

1. Thông số kỹ thuật

* Kiến trúc CPU: ARM Cortex-M3, tốc độ tối đa 72 MHz
* Flash: 64KB bộ nhớ Flash để lưu chương trình
* RAM: 20KB SRAM
* Nguồn cung cấp: 2.0V đến 3.6V
* GPIO: 37 chân GPIO, có thể cấu hình cho nhiều chức năng khác nhau
* Giao tiếp ngoại vi:
  + USART/UART: 3 kênh
  + SPI: 2 kênh
  + I2C: 2 kênh
  + CAN: 1 kênh (CAN 2.0B)
  + USB: 1 kênh (USB 2.0 Full-Speed)
  + ADC: 2 bộ ADC 12-bit với 16 kênh
  + PWM: Có hỗ trợ trên các chân Timer
* Timer: 4 bộ timer 16-bit (bao gồm cả PWM)
* Watchdog Timer: Để giám sát và khởi động lại hệ thống trong trường hợp gặp lỗi.

A circuit board with many different colored wires

Description automatically generated

Hình 1.1 Sơ đồ chân STM32F103C8T6

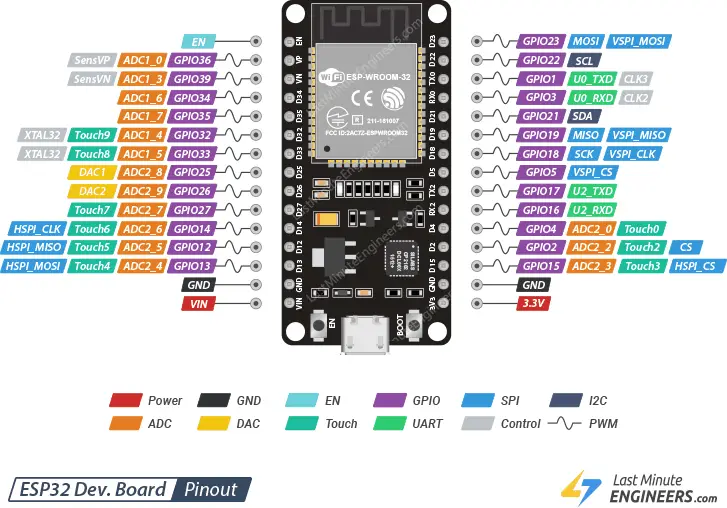
### 1.3.2 Vi điều khiển ESP32

1. Giới thiệu

Bên cạnh STM32 chịu trách nhiệm điều khiển cấp thấp, hệ thống tích hợp thêm module ESP32 từ hãng Espressif Systems. Đây là dòng vi điều khiển SoC mạnh mẽ, nổi bật với khả năng tích hợp sẵn kết nối không dây, đóng vai trò là cửa ngõ giao tiếp cho hệ thống. Trong đồ án này, ESP32 là điểm cuối của CAN bus, tổng hợp các thông tin thu thập được từ các node khác và gửi các thông tin đó đến Raspberry thông qua giao thức UART.

1. Thông số kỹ thuật

* Chip điều khiển: ESP32
* Số lượng chân: 30 pin
* Kết nối không dây: Wi-Fi và Bluetooth (v4.2)
* Điện áp hoạt động: 3.3V
* Bộ nhớ: RAM 520KB, Flash từ 4MB đến 16MB tùy phiên bản
* GPIO: 34 chân GPIO có thể lập trình
* ADC: 12-bit ADC với 18 kênh
* DAC: 2 kênh DAC 8-bit
* PWM: Hỗ trợ PWM cho điều khiển động cơ và ánh sáng
* Giao tiếp: SPI, I2C, UART, CAN



Hình 1.2 Sơ đồ chân ESP32

### 1.3.3 Raspberry Pi 4

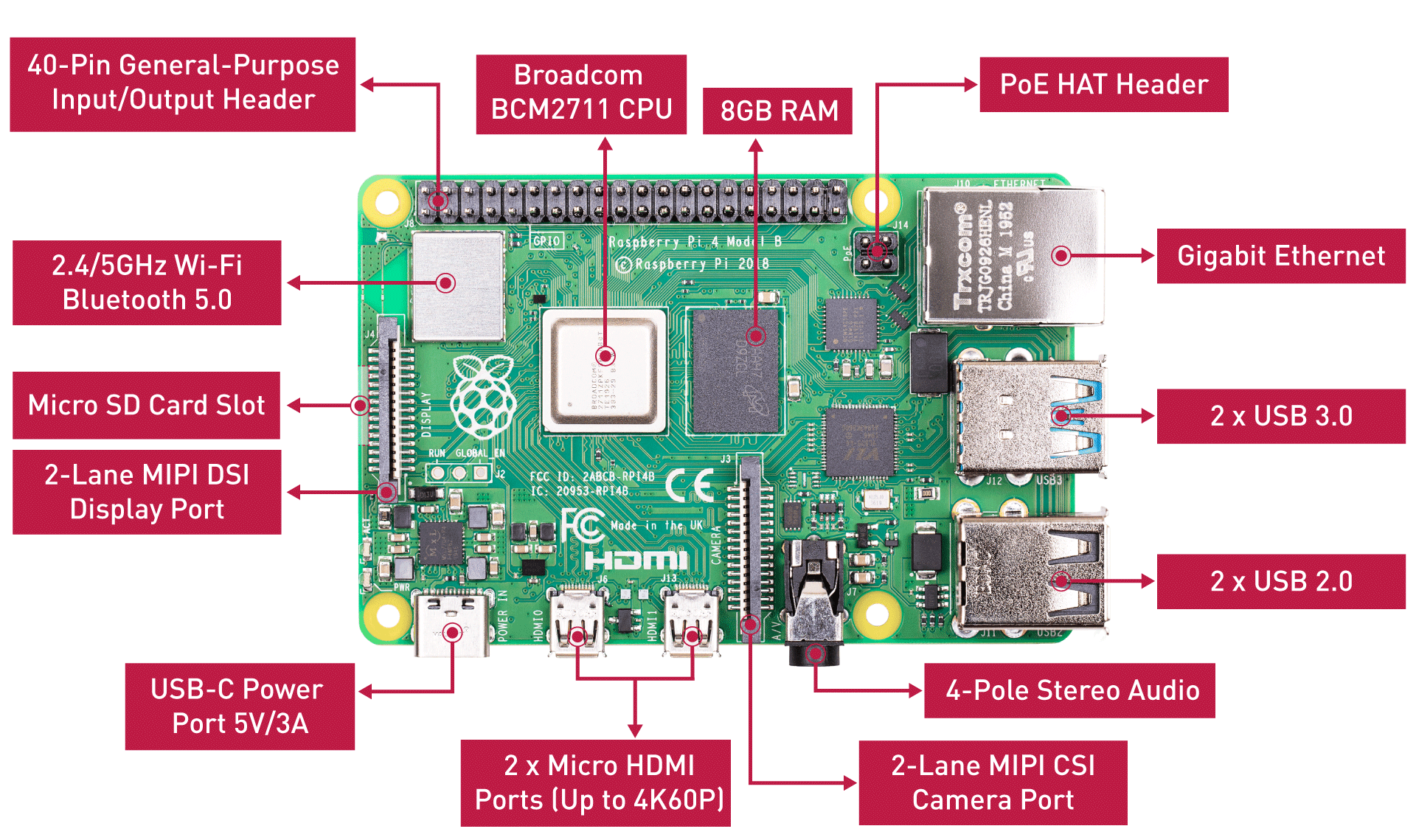
1. Giới thiệu

Raspberry Pi 4 là một máy tính mini nhỏ gọn, mạnh mẽ, lý tưởng cho các dự án DIY, học lập trình, AI, IoT, máy chủ web, trung tâm đa phương tiện nhờ hiệu năng cao và khả năng kết nối đa dạng.

Trong đồ án này Raspberry Pi 4 đóng vai trò là bộ xử lý trung tâm mạnh mẽ, chịu trách nhiệm chạy hệ điều hành, xử lý ảnh và hiển thị giao diện đồ họa.

1. Thông số kỹ thuật

* Broadcom BCM2711, Quad core Cortex-A72 (ARM v8) 64-bit SoC @ 1.5GHz
* 4GB LPDDR4-2400 SDRAM (depending on model)
* 2.4 GHz and 5.0 GHz IEEE 802.11ac wireless, Bluetooth 5.0, BLE
* Gigabit Ethernet
* 2 USB 3.0 ports / 2 USB 2.0 ports
* Raspberry Pi standard 40 pin GPIO header (fully backwards compatible with previous boards)
* 2 × micro-HDMI ports (up to 4kp60 supported)
* 2-lane MIPI DSI display port
* 2-lane MIPI CSI camera port
* 4-pole stereo audio and composite video port
* H.265 (4kp60 decode), H264 (1080p60 decode, 1080p30 encode)
* OpenGL ES 3.0 graphics
* Micro-SD card slot for loading operating system and data storage



Hình 1.3 Raspberry Pi 4

### 1.3.4 Giao thức CAN bus

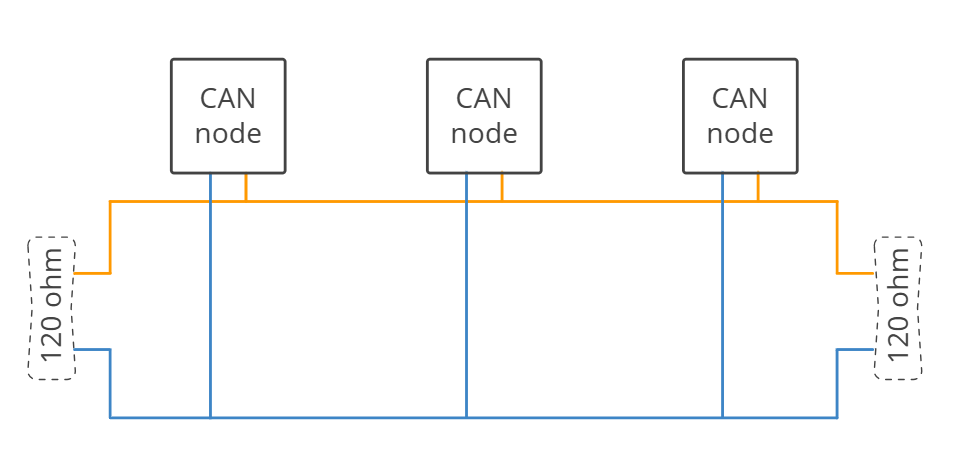
1. Tổng quan

Mạng giao tiếp CAN trên ô tô là một mạng chuẩn trong giao tiếp của ô tô cho phép các module khác nhau của ô tô giao tiếp với các ECU trên ô tô. Nó là một nhóm các đường dây điện dẫn tín hiệu với tốc độ rất cao cho phép truyền tải các dữ liệu và các lệnh mà xe hiệu chỉnh được nén qua lại từ các module khác nhau.

1. Cấu tạo mạng CAN bus

CAN sử dụng hai dây để giao tiếp. Các dây được gọi là CAN High và CAN Low. CAN kết nối với tất cả các thành phần trên mạng thông qua hai dây này. Trong CAN không phân biệt master và slave như một số giao thức khác. Đường dây bus kết thúc bằng điện trở 120 ohm (thấp nhất là 108 Ohm và tối đa là 132 ohm) ở mỗi đầu.

Mạng CAN được tạo thành bởi một nhóm các node. Mỗi node có thể giao tiếp với bất kỳ node khác trong mạng. Việc giao tiếp được thực hiện bằng việc truyền và nhận các message. Mỗi loại massage trong mạng CAN được gán cho một số ID (Identifier) tùy theo mức ưu tiên của message đó. Message nào có ID càng nhỏ thì có mức ưu tiên càng cao.



Hình 1.4 Cấu trúc kết nối mạng CAN bus

Không giống như một số giao thức khác trong việc sử dụng mức điện áp trên dây so với GND để quy đổi ra bit logic, CAN sử dụng mức chênh lệch điện áp giữa 2 dây để quy đổi. Những thay đổi về mức điện áp này sau đó được dịch sang mức logic.

A graph of a graph

Description automatically generated with medium confidence

Hình 1.5 Minh họa tín hiệu vi sai

Thông thường, giá trị điện áp cao nhất ở dây CAN High là 3.5V, điện áp thấp nhất là 2.5V. Ở dây CAN Low điện áp thấp nhất là 1.5V và cao nhất là 2.5V. Các giá trị điện áp được xác định bởi bộ chuyển đổi CAN và được thiết kế tuân theo chuẩn CAN.

Khi ở trạng thái Dominant (đại diện cho giá trị logic 0), điện áp trên CAN\_H tăng lên ~3.75V, Điện áp trên CAN\_L giảm xuống ~1.25V. Sự chênh lệch điện áp giữa CAN\_H và CAN\_L lúc này là khoảng 2.5V.

Khi ở trạng thái Recessive (đại diện cho giá trị logic 1), điện áp trên CAN\_H và CAN\_L gần như bằng nhau (~2.5V). Sự chênh lệch điện áp giữa CAN\_H và CAN\_L lúc này là không đáng kể (CAN\_H - CAN\_L = 0V).

A graph with red lines and white text

Description automatically generated

Hình 1.6 Khả năng kháng nhiễu của CAN

Vì sử dụng tín hiệu vi sai nên CAN có khả năng kháng nhiễu rất tốt. Khi có nhiễu tác động lên, CAN điều khiển mức điện áp ở 2 dây cùng thay đổi (cùng tăng hoặc cùng giảm) nên mức chênh lệch điện áp giữa 2 dây không thay đổi.

1. Cấu tạo một node ECU trên CAN

Một ECU trên CAN có 3 thành phần cơ bản: Vi điều khiển, CAN Controller và Can Transceiver với từng chức năng khác nhau

* Vi điều khiển: Xử lý dữ liệu nhận được từ bộ điều khiển CAN và xử lý dữ liệu để gửi đến một ECU khác, hai việc này có thể diễn ra độc lập hoặc cùng lúc tùy theo nhiệm vụ của ECU.
* CAN Controller: Truyền dữ liệu nối tiếp từ vi điều khiển vào bus khi bus rảnh. Khi nhận dữ liệu, nó lưu trữ các bit đã nhận từ bus cho đến khi đủ toàn bộ khung truyền và thông báo cho vi điều khiển.
* CAN transceiver: chuyển đổi mức điện áp trên CAN thành thông tin mà CAN Controller có thể hiểu được và ngược lại chuyển đổi luồng dữ liệu từ CAN Controller thành mức logic của CAN.

A diagram of a computer hardware system

Description automatically generated

Hình 1.7 Sơ đồ một nút mạng CAN

1. Các loại CAN frame

Mỗi ECU trên bus có một ID định danh phân biệt với các ECU khác. Tất cả các ECU trên bus đều được nối song song với đường dây, điều này dẫn đến tất cả các ECU đều nhận mọi dữ liệu chạy trên đường bus mọi lúc. Chúng ta có thể cấu hình để 1 ECU chỉ nhận gói tin đến từ một hay nhiều ECU nhất định bằng ID.

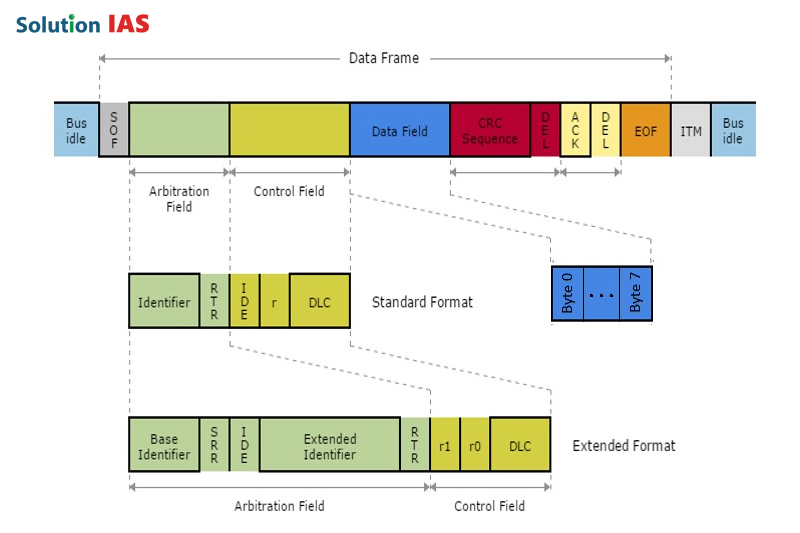
Dữ liệu CAN được truyền đi dưới dạng các Frame. Có 4 loại Frame khác nhau, đó là:

* Data Frame (khung dữ liệu): là khung mang dữ liệu từ một bộ truyền dữ liệu đến các bộ nhận dữ liệu. Khung này có vùng để mang các byte dữ liệu.
* Remote Frame (khung yêu cầu hay điều khiển): là khung được truyền từ một Node bất kỳ để yêu cầu dữ liệu từ Node khác. Khi Node khác đó nhận được yêu cầu sẽ truyền lại dữ liệu có ID trùng với ID được gửi trong Remote Frame.
* Error Frame (khung lỗi): là khung được truyền bởi bất kỳ Node nào khi Node đó phát hiện lỗi từ Bus.
* Overload Frame (khung báo tràn): là khung được gửi bởi một node nếu nó không kịp xử lý dữ liệu hoặc phát hiện điều kiện overload trên bus. Mục đích là tạo thêm thời gian trễ để hoàn thành xử lý. Overload Frame không trực tiếp ngăn các node khác gửi dữ liệu mà chỉ tạo một khoảng dừng để node xử lý kịp thời.

1. Cấu trúc gói tin khi truyền

Cấu trúc gói tin của ECU khi truyền bao gồm:

* Bus Idle: Bus ở trạng thái nghỉ.
* Bit bắt đầu khung: vị trí của bit đầu tiên trong khung dữ liệu.
* Arbitration Field (Trường xác định quyền ưu tiên):
  + Identifier: định danh của một ECU. Nó được sử dụng để chỉ rõ ECU nào đang gửi gói tin. ID cũng xác định mức độ ưu tiên: ID càng thấp, mức độ ưu tiên của gói tin càng cao. Gồm 11 bit ID cho gói tin Standard và 29 bit cho gói tin Extended.
  + Remote transmission request: Bit dùng để xác định khung là Data frame hay Remote frame.
  + Substitute remove request: Bit thay thế bit Remove transmission request ở khung chuẩn, giữ giá trị 1.
* Control Field:
* Identifier extension: Bit phân biệt khung chuẩn và khung mở rộng.
* r, r0, r1: Reserved bits, giữ giá trị 0.
* Data length code: Quy định số byte của trường dữ liệu Data field (0 đến 8 byte).
* Data Field: Chứa dữ liệu thực tế cần truyền đi, chiều dài tối đa là 8 byte.
* Cyclic Redundancy Check Field:
* Cyclic Redundancy Check Sequence: Chuỗi mã hóa để kiểm tra tính toàn vẹn của dữ liệu.
* Cyclic Redundancy Check Delimiter: Bit cố định 1, phân cách giữa CRC và ACK.
* Acknowledge Field:
* Acknowledge Slot: Node nhận dữ liệu ghi giá trị 0 vào đây để xác nhận nhận thành công.
* Acknowledge Delimiter: Bit cố định 1, phân cách ACK với phần sau.
* End Of Frame: 7 bit cố định 1, báo hiệu kết thúc khung dữ liệu.



Hình 1.8 Cấu trúc khung dữ liệu

Trong đề tài, giao thức CAN được sử dụng làm chuẩn truyền thông chính giữa các module trong hệ thống nhúng. CAN bus đảm nhiệm vai trò truyền dữ liệu điều khiển và trạng thái giữa các bộ xử lý, đảm bảo quá trình trao đổi thông tin diễn ra ổn định, tin cậy và phù hợp với yêu cầu của các hệ thống.

Cụ thể, giao thức CAN được sử dụng để truyền dữ liệu giữa các module xử lý trung tâm và module điều khiển trong hệ thống. Module xử lý có nhiệm vụ thu thập và xử lý dữ liệu, sau đó đóng gói dữ liệu thành các khung CAN và truyền lên mạng CAN. Module điều khiển nhận dữ liệu CAN, giải mã nội dung thông điệp và thực hiện các tác vụ điều khiển tương ứng.

Việc sử dụng giao thức CAN giúp hệ thống hoạt động ổn định trong môi trường có nhiễu, đảm bảo dữ liệu được truyền chính xác và kịp thời. Đồng thời, CAN cho phép mở rộng hệ thống bằng cách bổ sung thêm các nút mạng mà không cần thay đổi nhiều về cấu trúc phần cứng, góp phần nâng cao tính linh hoạt và khả năng ứng dụng thực tế của đề tài.

### 1.3.5 Giao thức UART

1. Tổng quan

UART là một giao thức truyền thông nối tiếp bất đồng bộ, được sử dụng phổ biến trong các hệ thống nhúng để trao đổi dữ liệu giữa vi điều khiển với các thiết bị ngoại vi hoặc giữa các vi điều khiển với nhau. Giao thức UART không yêu cầu xung clock chung giữa thiết bị phát và thiết bị thu, thay vào đó hai bên phải được cấu hình thống nhất các tham số truyền thông trước khi giao tiếp.

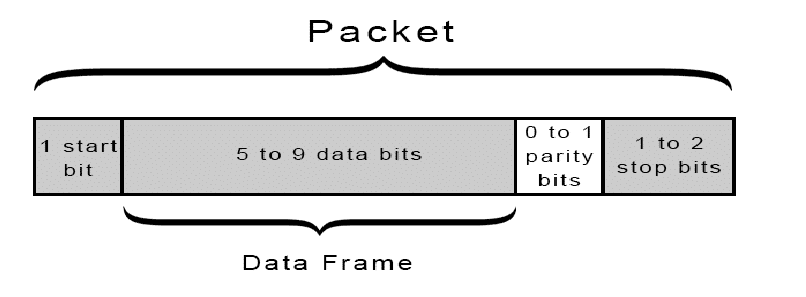
Trong thực tế, UART thường được sử dụng trong các ứng dụng như giao tiếp giữa vi điều khiển và máy tính, truyền dữ liệu debug, cấu hình thiết bị hoặc trao đổi dữ liệu điều khiển trong các hệ thống nhúng có yêu cầu đơn giản.

1. Nguyên lý truyền dữ liệu

Dữ liệu trong UART được truyền theo từng frame. Một khung truyền UART tiêu chuẩn bao gồm các thành phần sau:

* Start bit: Là bit bắt đầu truyền, có mức logic thấp, dùng để báo hiệu cho thiết bị thu rằng quá trình truyền dữ liệu sắp diễn ra.
* Data bits: Là các bit dữ liệu chính, thường có độ dài từ 5 đến 9 bit, trong đó phổ biến nhất là 8 bit.
* Parity bit (tùy chọn): Được sử dụng để kiểm tra lỗi đơn giản trong quá trình truyền dữ liệu.
* Stop bit: Là bit kết thúc khung truyền, có mức logic cao, có thể là 1 hoặc 2 bit.

Mỗi khung UART được truyền liên tục từ bit bắt đầu đến bit kết thúc. Thiết bị thu dựa vào tốc độ baud đã được cấu hình để lấy mẫu và giải mã dữ liệu nhận được.



Hình 1.9 Cấu trúc khung truyền UART

1. Các tham số cấu hình

Để đảm bảo quá trình truyền và nhận dữ liệu chính xác, các thiết bị giao tiếp UART cần được cấu hình thống nhất các tham số sau:

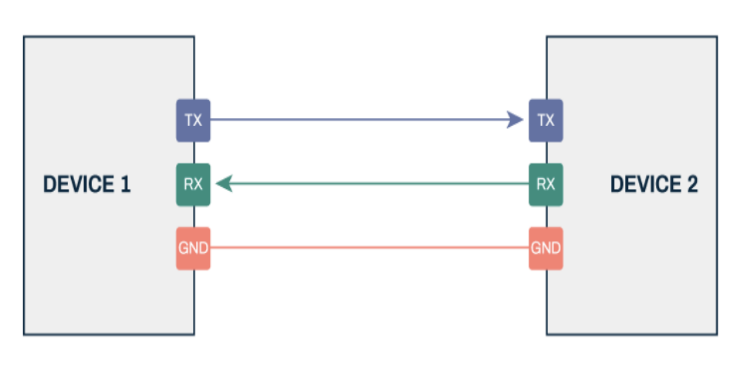
* Baud rate: Tốc độ truyền dữ liệu, đơn vị là bit/giây (bps), ví dụ 9600 bps, 115200 bps.
* Data bits: Số bit dữ liệu trong một khung truyền, thường là 8 bit.
* Parity: Chế độ kiểm tra lỗi, bao gồm None, Even hoặc Odd.
* Stop bits: Số bit kết thúc, thường là 1 bit.
* Flow control: Cơ chế điều khiển luồng dữ liệu, thường không sử dụng trong các ứng dụng đơn giản.

Trong các hệ thống nhúng hiện nay, cấu hình UART phổ biến nhất là 115200 baud, 8 data bits, không parity, 1 stop bit.

1. Sơ đồ kết nối

UART sử dụng hai đường tín hiệu chính là TX (Transmit) và RX (Receive). Khi kết nối hai thiết bị với nhau, chân TX của thiết bị này phải được nối với chân RX của thiết bị kia và ngược lại. Ngoài ra, các thiết bị cần sử dụng chung một điểm mass để đảm bảo mức điện áp tham chiếu giống nhau.

Việc kết nối UART có cấu trúc đơn giản, không yêu cầu phần cứng phức tạp, do đó rất thuận tiện cho việc triển khai và kiểm tra hệ thống trong giai đoạn phát triển.



Hình 1.10 Sơ đồ kết nối UART

Trong phạm vi đề tài, giao thức UART được sử dụng để truyền dữ liệu giữa ESP32 và Raspberry Pi, điều này khắc phục hạn chế về phần cứng, giảm độ phức tạp, đồng thời thuận tiện cho việc kiểm tra và hiệu chỉnh hệ thống trong quá trình phát triển.

### 1.3.6 Giao thức UDP

1. Tổng quan

UDP là một giao thức truyền thông thuộc tầng giao vận (Transport Layer) trong mô hình TCP/IP. UDP hoạt động theo cơ chế không kết nối, trong đó dữ liệu được gửi dưới dạng các gói tin độc lập mà không cần thiết lập kết nối trước giữa bên gửi và bên nhận.

Không giống như giao thức TCP, UDP không đảm bảo độ tin cậy tuyệt đối trong quá trình truyền dữ liệu, không thực hiện kiểm soát lỗi, kiểm soát luồng hay cơ chế truyền lại khi gói tin bị mất. Tuy nhiên, nhờ cấu trúc đơn giản và không yêu cầu xác nhận gói tin, UDP có độ trễ thấp và tốc độ truyền nhanh, phù hợp với các ứng dụng yêu cầu thời gian thực.

Gói tin UDP bao gồm phần tiêu đề có độ dài cố định 8 byte và phần dữ liệu. Phần tiêu đề chứa các thông tin cơ bản như cổng nguồn, cổng đích, độ dài gói tin và trường kiểm tra lỗi. Cấu trúc đơn giản này giúp giảm tải xử lý cho hệ thống khi truyền và nhận dữ liệu.

1. Đặc điểm

Các đặc điểm chính của giao thức UDP bao gồm:

* Không yêu cầu thiết lập kết nối trước khi truyền dữ liệu.
* Không đảm bảo thứ tự và độ toàn vẹn của gói tin.
* Độ trễ thấp và tốc độ truyền cao.
* Dễ triển khai trên các hệ thống nhúng và máy tính nhúng.

1. Ứng dụng trong hệ thống

Trong các hệ thống nhúng và hệ thống xử lý dữ liệu thời gian thực, UDP thường được sử dụng để truyền các dữ liệu trạng thái hoặc kết quả xử lý có tần suất cao. Trong đề tài này, giao thức UDP được sử dụng để truyền dữ liệu giữa các khối phần mềm chạy trên Raspberry Pi, cụ thể là truyền kết quả xử lý hình ảnh từ chức năng nhận diện biển báo giao thông đến giao diện Instrument Cluster.

Việc sử dụng UDP giúp đảm bảo dữ liệu được cập nhật nhanh chóng lên giao diện, đồng thời không làm gián đoạn quá trình xử lý chính của hệ thống, đáp ứng yêu cầu thời gian thực của đề tài.

# CHƯƠNG II: PHÂN TÍCH VÀ THIẾT KẾ HỆ THỐNG

## 2.1 Phân tích yêu cầu hệ thống

Mục này trình bày các yêu cầu của hệ thống nhằm xác định rõ các chức năng cần thực hiện cũng như các ràng buộc kỹ thuật trong quá trình thiết kế và triển khai. Việc phân tích yêu cầu hệ thống là cơ sở để xây dựng kiến trúc phần cứng, phần mềm và lựa chọn giao thức truyền thông phù hợp cho đề tài.

### 2.1.1 Yêu cầu chức năng

Hệ thống được xây dựng nhằm thực hiện các chức năng chính sau:

* Hệ thống có khả năng thu nhận hình ảnh từ camera và xử lý hình ảnh trên Raspberry Pi.
* Hệ thống thực hiện nhận diện biển báo giao thông dựa trên thuật toán xử lý ảnh và mô hình học sâu đã được huấn luyện.
* Hệ thống phân tích kết quả nhận diện để xác định thông tin điều khiển tương ứng.
* Hệ thống sử dụng các giao thức truyền thông (CAN, UART) để giao tiếp giữa các node
* Vi điều khiển (STM32, ESP32) tiếp nhận dữ liệu, giải mã thông tin và thực hiện điều khiển động cơ theo kết quả nhận diện.
* Hệ thống cho phép giám sát trạng thái hoạt động và hỗ trợ quá trình kiểm tra, thử nghiệm trong quá trình vận hành.
* Hệ thống hoạt động liên tục và đảm bảo đáp ứng yêu cầu điều khiển theo thời gian thực.

### 2.1.2 Yêu cầu phi chức năng

Bên cạnh các chức năng chính, hệ thống cần đáp ứng các yêu cầu phi chức năng sau:

* Yêu cầu về hiệu năng
* Thời gian xử lý nhận diện và truyền dữ liệu phải đủ nhanh để đáp ứng yêu cầu điều khiển.
* Độ trễ truyền thông giữa các node nằm trong giới hạn cho phép của hệ thống thời gian thực.
* Yêu cầu về độ tin cậy
* Hệ thống hoạt động ổn định trong thời gian dài.
* Dữ liệu truyền thông giữa các module đảm bảo tính chính xác và hạn chế lỗi truyền.
* Có khả năng phát hiện và xử lý lỗi trong quá trình giao tiếp.
* Yêu cầu về truyền thông
* Giao thức truyền thông phải phù hợp với môi trường có nhiễu và hệ thống nhiều node.
* Hỗ trợ mở rộng hệ thống bằng cách bổ sung thêm các nút điều khiển khi cần thiết.
* Đảm bảo tính tương thích giữa các thiết bị sử dụng trong hệ thống.
* Yêu cầu về phần cứng
* Sử dụng các phần cứng phổ biến như Raspberry Pi, STM32, ESP32 để dễ dàng triển khai và thay thế.
* Hệ thống phần cứng có cấu trúc rõ ràng, dễ lắp đặt và bảo trì.
* Phù hợp với điều kiện thí nghiệm và mô hình nghiên cứu.
* Yêu cầu về chi phí và tính khả thi
* Chi phí triển khai thấp, linh kiện dễ tìm trên thị trường.
* Hệ thống phù hợp để phát triển thành mô hình thực nghiệm phục vụ học tập và nghiên cứu.
* Có khả năng mở rộng và nâng cấp trong các nghiên cứu tiếp theo.

## 2.2 Kiến trúc hệ thống

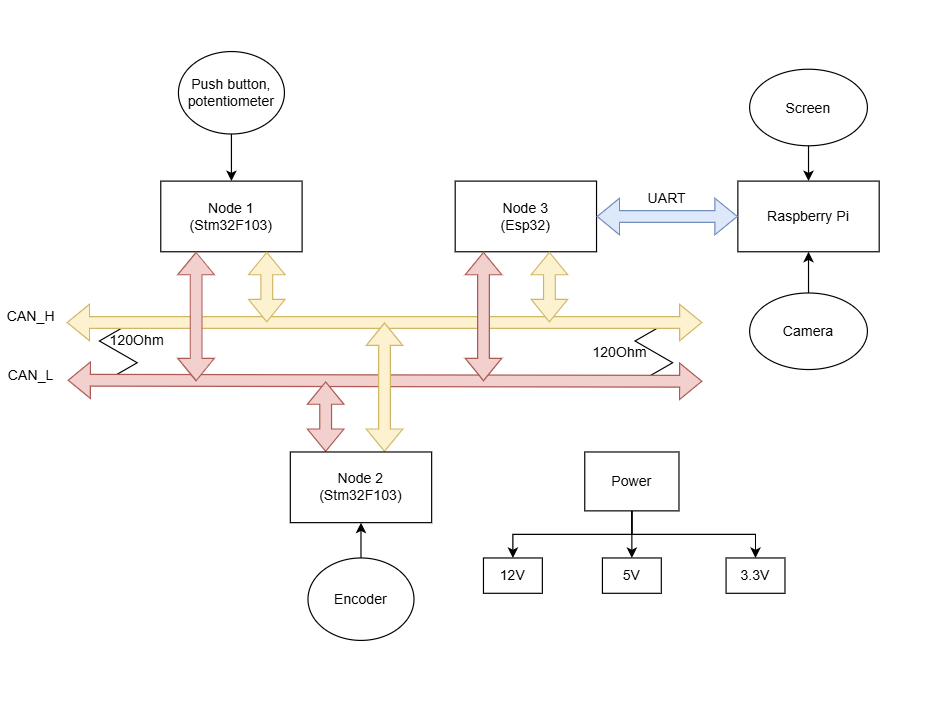
Mục này trình bày kiến trúc tổng thể của hệ thống, bao gồm sơ đồ khối, sơ đồ nguyên lý của từng node và nguyên lý hoạt động của hệ thống. Qua đó làm rõ mối quan hệ giữa các node và cách thức hệ thống phối hợp để thực hiện các chức năng đề ra.

### 2.2.1 Sơ đồ khối hệ thống

Sơ đồ khối hệ thống mô tả cấu trúc tổng thể của hệ thống và mối quan hệ giữa các khối chức năng chính. Hệ thống được thiết kế theo mô hình phân tán, trong đó mỗi khối đảm nhiệm một chức năng riêng biệt nhằm tăng tính linh hoạt và khả năng mở rộng. Hệ thống bao gồm 4 node chức năng chính kết nối qua hai đường truyền vật lý là CAN Bus và UART:

* Node 1 (Body Control Node): Sử dụng vi điều khiển STM32F103, chịu trách nhiệm thu thập tín hiệu điều khiển từ người lái (Nút nhấn, Joystick, HC-SR04).
* Node 2 (Motor Control Node): Sử dụng vi điều khiển STM32F103, chịu trách nhiệm điều khiển động cơ và đọc phản hồi từ Encoder.
* Node 3 (Gateway Node): Sử dụng vi điều khiển ESP32. Đây là node trung gian đóng vai trò cầu nối dữ liệu giữa mạng CAN Bus và máy tính nhúng.
* Central Unit (Khối xử lý trung tâm): Sử dụng Raspberry Pi, chịu trách nhiệm hiển thị giao diện người dùng, xử lý ảnh từ camera và giao tiếp với Node Gateway qua chuẩn UART.
* Khối nguồn: Đóng vai trò cung cấp năng lượng ổn định cho toàn bộ hệ thống. Khối sử dụng nguyên lý nguồn tuyến tính với biến áp hạ áp và chỉnh lưu cầu để chuyển đổi điện áp lưới 220V AC sang điện áp một chiều. Hệ thống nguồn cung cấp hai mức điện áp đầu ra riêng biệt là 12V và 5V.

Luồng dữ liệu trong hệ thống được truyền từ các node đến gateway node để tổng hợp dữ liệu rồi được truyền sang Raspberry Pi để hiển thị lên giao diện trên màn hình.



Hình 2.1 Sơ đồ tổng quan của hệ thống

### 2.2.2 Lưu đồ thuật toán từng node

Hệ thống phần cứng được chia thành 4 module riêng biệt, đảm bảo tính module hóa và dễ dàng bảo trì. Chi tiết cấu tạo và nguyên lý của từng node như sau:

1. Body Control Node

Body Control Node đóng vai trò là giao diện vật lý giữa người điều khiển và hệ thống xe, thực hiện chức năng thu thập các tín hiệu điều khiển từ người lái và chuyển đổi thành dữ liệu số để truyền lên mạng CAN Bus. Node này mô phỏng các chức năng cơ bản của vô lăng và cụm điều khiển thân xe trên ô tô.

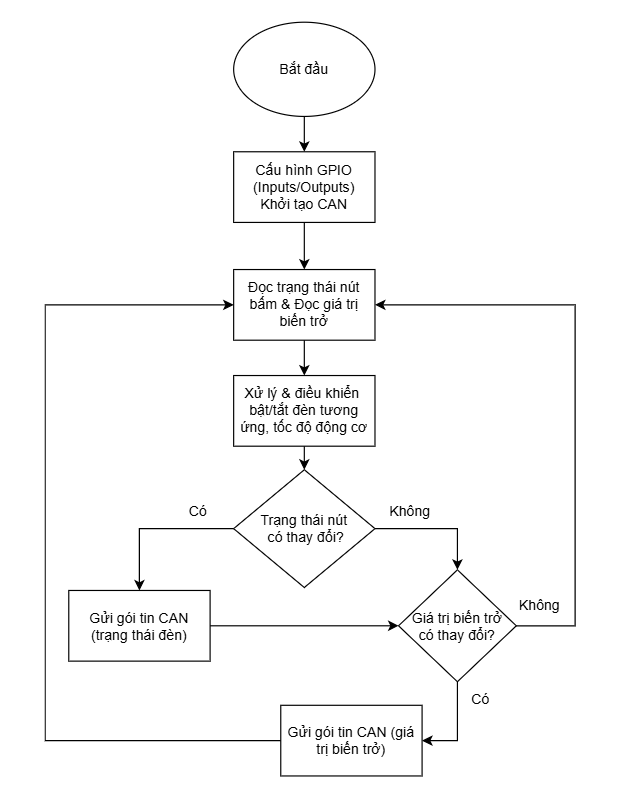
Vi điều khiển trung tâm sử dụng kit phát triển STM32F103C8T6 (Blue Pill), hoạt động ở tần số xung nhịp 72 MHz. Vi điều khiển có nhiệm vụ đọc tín hiệu từ các thiết bị đầu vào, xử lý và đóng gói dữ liệu thành các khung truyền CAN.

Hệ thống nút nhấn bao gồm các nút chức năng như: xi-nhan trái/phải, pha/cos, đèn cảnh báo nguy hiểm và các nút điều khiển cruise control (Set/Resume/Gap/Limit). Các nút được kết nối với các chân GPIO của vi điều khiển thông qua điện trở kéo lên nhằm đảm bảo trạng thái logic ổn định, hạn chế nhiễu và tránh hiện tượng trôi mức điện áp khi không có tác động nhấn.

Ngoài ra, node sử dụng joystick analog hai trục để điều khiển tốc độ và chiều quay của xe. Trong đó, trục Y (VRY) được kết nối với bộ chuyển đổi tương tự – số (ADC) của vi điều khiển, cho phép đọc giá trị điện áp biến thiên trong khoảng từ 0 V đến 3.3 V. Giá trị ADC thu được được ánh xạ thành mức tốc độ hoặc lệnh điều khiển phù hợp trước khi gửi lên mạng CAN.

Bên cạnh đó, cảm biến khoảng cách siêu âm HC-SR04 được tích hợp để đo khoảng cách đến vật cản phía trước xe. Module này giao tiếp với vi điều khiển thông qua hai chân GPIO (Trig và Echo). Dữ liệu đo được sử dụng để hỗ trợ các chức năng an toàn như cảnh báo hoặc giới hạn tốc độ khi phát hiện vật cản ở khoảng cách gần.

Về giao tiếp mạng, BCN sử dụng module MCP2515 kết hợp với transceiver TJA1050 để chuyển đổi giao tiếp SPI từ vi điều khiển sang tín hiệu vi sai của chuẩn CAN Bus, cho phép node truyền dữ liệu điều khiển và trạng thái hệ thống đến các node khác trong mạng.



Hình 2.2 Lưu đồ thuật toán Body Control Node

1. Motor Control Node

Motor Control Node đảm nhiệm chức năng điều khiển vận hành của xe, bao gồm điều khiển tốc độ, chiều quay động cơ và phản hồi thông tin vận tốc thực tế lên mạng CAN Bus. Node này mô phỏng chức năng của bộ điều khiển động cơ trong hệ thống ô tô.

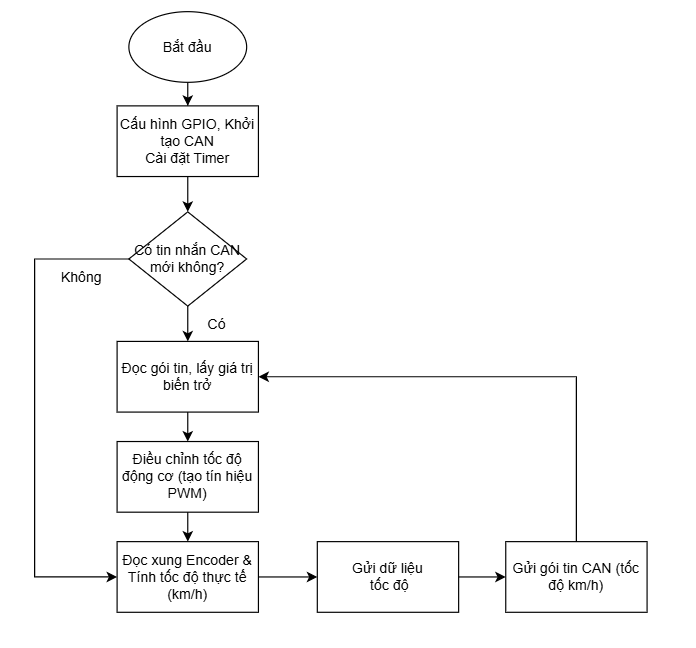
Vi điều khiển trung tâm sử dụng STM32F103C8T6, có nhiệm vụ tiếp nhận các khung dữ liệu CAN từ mạng, giải mã lệnh điều khiển do Body Control Node gửi đến và thực hiện điều khiển động cơ tương ứng. Đồng thời, vi điều khiển thu thập dữ liệu phản hồi từ encoder để tính toán vận tốc và truyền ngược trở lại mạng CAN.

Khối điều khiển động cơ sử dụng mạch cầu H, nhận tín hiệu điều xung PWM được tạo ra từ bộ Timer của vi điều khiển. Thông qua việc thay đổi độ rộng xung PWM, điện áp trung bình cấp cho động cơ được điều chỉnh, từ đó thay đổi tốc độ quay và chiều quay của động cơ.

Động cơ sử dụng trong hệ thống là động cơ DC giảm tốc, được cấp nguồn 12V, đảm bảo mô-men xoắn phù hợp cho quá trình di chuyển của mô hình xe.

Để đo và phản hồi vận tốc, hệ thống sử dụng encoder quang với đĩa encoder 20 xung gắn trực tiếp trên trục bánh xe, kết hợp với cảm biến quang chữ U. Tín hiệu xung đầu ra của encoder được đưa vào bộ Timer của STM32 ở chế độ External Clock. Vi điều khiển tiến hành đếm số xung trong một khoảng thời gian cố định (100 ms), từ đó tính toán vận tốc thực tế của xe.

Về giao tiếp mạng, Motor Control Node sử dụng module MCP2515 để kết nối vào mạng CAN Bus, cho phép node nhận lệnh điều khiển từ các node khác và gửi lại các thông tin phản hồi như vận tốc, trạng thái động cơ, phục vụ cho quá trình giám sát và điều khiển toàn hệ thống.



Hình 2.3 Lưu đồ thuật toán Motor Control Node

1. Gateway Node

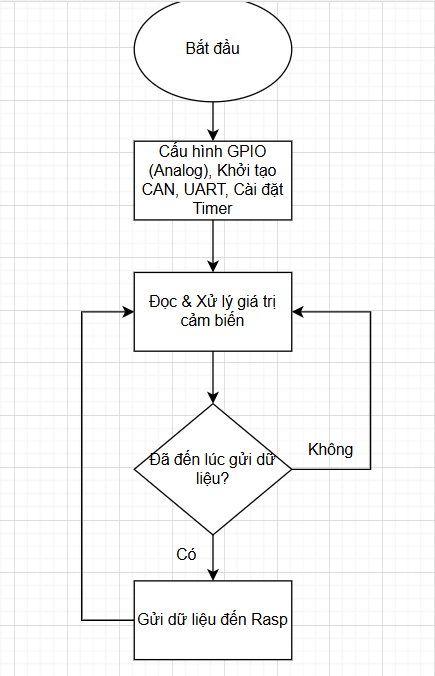
Gateway Node đóng vai trò là node trung gian trong hệ thống, thực hiện chức năng kết nối và chuyển đổi dữ liệu giữa hai môi trường truyền dẫn khác nhau gồm mạng điều khiển thời gian thực (CAN Bus) và mạng xử lý thông tin mức cao (UART). Node này giúp tách biệt khối điều khiển thời gian thực và khối xử lý dữ liệu, góp phần nâng cao tính ổn định và khả năng mở rộng của hệ thống.

Vi điều khiển trung tâm sử dụng module ESP32-WROOM-32, có khả năng xử lý mạnh và hỗ trợ nhiều giao tiếp ngoại vi, phù hợp cho vai trò gateway trong hệ thống đa giao thức.

Về phía mạng CAN, Gateway Node kết nối với CAN Bus thông qua module MCP2515, cho phép node lắng nghe và tiếp nhận các khung dữ liệu được truyền từ Body Control Node và Motor Control Node. ESP32 thực hiện việc đọc dữ liệu từ MCP2515, giải mã CAN ID và payload tương ứng để xác định loại thông tin.

Về phía mạng UART, Gateway Node sử dụng giao tiếp Serial (TX/RX) để thiết lập kênh truyền dữ liệu song công với máy tính nhúng Raspberry Pi. Dữ liệu sau khi được xử lý và chuyển đổi sẽ được gửi qua UART để phục vụ các tác vụ xử lý.

Chức năng chính của Gateway Node là thực hiện quá trình phân tích và chuyển đổi dữ liệu (Data Parsing). Cụ thể, ESP32 tách dữ liệu thô từ các khung CAN có ID quy ước (ví dụ: 0x01, 0x02, …), sau đó chuyển đổi dữ liệu này sang chuỗi ký tự hoặc cấu trúc byte theo định dạng thống nhất, đảm bảo Raspberry Pi có thể tiếp nhận và xử lý một cách chính xác và hiệu quả.



Hình 2.4 Lưu đồ thuật toán Gateway Node

1. Central Unit

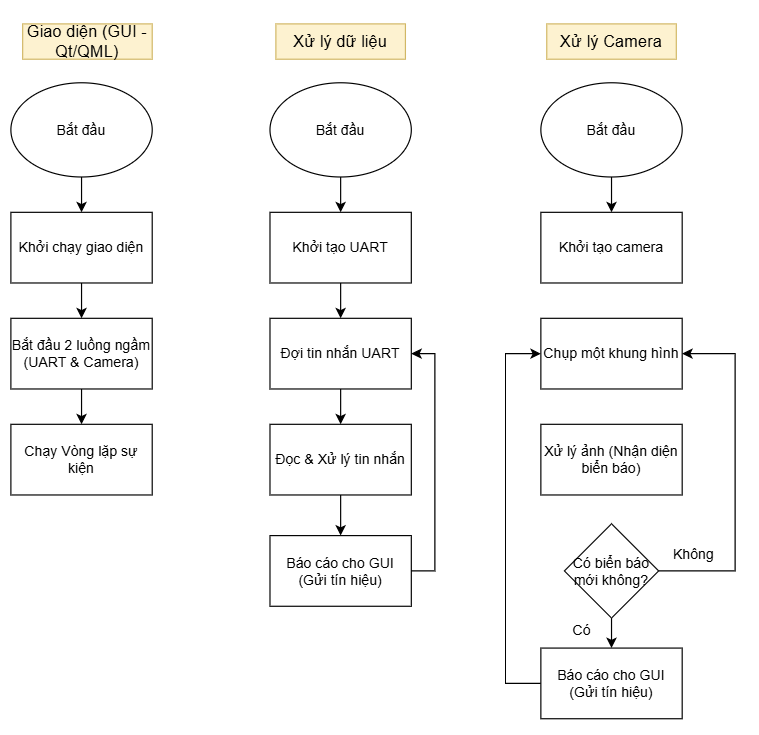
Central Unit đóng vai trò là khối xử lý trung tâm của toàn hệ thống, thực hiện các chức năng xử lý dữ liệu mức cao, bao gồm thu thập và phân tích dữ liệu hình ảnh từ môi trường, hiển thị thông tin vận hành và hỗ trợ ra quyết định điều khiển. Node này mô phỏng vai trò của bộ điều khiển trung tâm trong các hệ thống ô tô hiện đại.

Phần cứng sử dụng máy tính nhúng Raspberry Pi 4 Model B, có khả năng xử lý mạnh, phù hợp cho các tác vụ yêu cầu tài nguyên lớn như xử lý ảnh, giao diện người – máy (HMI) và quản lý dữ liệu hệ thống.

Về giao diện hiển thị, Central Unit kết nối với màn hình LCD thông qua cổng HDMI để hiển thị giao diện bảng đồng hồ. Giao diện này thể hiện các thông tin vận hành của xe như vận tốc, trạng thái động cơ, tín hiệu điều khiển và các cảnh báo, giúp người dùng theo dõi tình trạng hệ thống một cách trực quan.

Đối với chức năng cảm nhận môi trường, Central Unit sử dụng camera module được kết nối trực tiếp với Raspberry Pi để thu thập hình ảnh phía trước xe. Dữ liệu hình ảnh thu được được xử lý bởi các thuật toán nhận diện biển báo giao thông, phục vụ cho việc phân tích môi trường và hỗ trợ ra quyết định điều khiển phù hợp.

Về kết nối dữ liệu, Central Unit nhận các thông tin vận hành và trạng thái hệ thống từ Gateway Node thông qua giao tiếp UART (USB-to-TTL). Dữ liệu sau khi được tiếp nhận sẽ được phân tích và đồng bộ lên giao diện hiển thị, đồng thời có thể được sử dụng làm đầu vào cho các thuật toán xử lý và điều khiển ở mức cao.



Hình 2.5 Lưu đồ thuật toán Central Unit

## 2.3 Thiết kế phần cứng

Mục này trình bày quá trình thiết kế phần cứng cho hệ thống, bao gồm việc lựa chọn linh kiện phù hợp với yêu cầu đề tài, thiết kế sơ đồ nguyên lý và mạch in (PCB), đảm bảo hệ thống hoạt động ổn định, dễ mở rộng.

### 2.3.1 Lựa chọn linh kiện

Việc lựa chọn linh kiện được thực hiện dựa trên các tiêu chí chính gồm: khả năng đáp ứng chức năng, tính ổn định, khả năng mở rộng, mức độ phổ biến, và chi phí. Các linh kiện được so sánh với những phương án thay thế phổ biến trước khi đưa ra lựa chọn cuối cùng cho hệ thống.

1. Vi điều khiển cho các node điều khiển

|  |  |  |  |
| --- | --- | --- | --- |
| **Tiêu chí** | **STM32F103C8T6** | **ATmega328P** | **ESP8266** |
| Kiến trúc | ARM Cortex-M3 (32-bit) | AVR (8-bit) | Tensilica (32-bit) |
| Tần số hoạt động | 72 MHz | 16 MHz | 80 MHz |
| Ngoại vi Timer/ADC | Mạnh, đa dạng | Hạn chế | Trung bình |
| Phù hợp điều khiển thời gian thực | Rất tốt | Trung bình | Trung bình |
| Giá thành | Trung bình | Cao | Trung bình |

Bảng 2.1 So sánh MCU cho các node điều khiển

Lựa chọn: STM32F103C8T6

Lý do: STM32F103C8T6 có hiệu năng cao, nhiều bộ Timer và ADC, phù hợp cho các tác vụ điều khiển thời gian thực như tạo PWM, đọc encoder và xử lý dữ liệu CAN, đáp ứng tốt yêu cầu của đề tài.

1. Vi điều khiển Gateway Node

|  |  |  |  |
| --- | --- | --- | --- |
| **Tiêu chí** | **ESP32** | **STM32F4** | **Raspberry Pi Pico** |
| Số giao tiếp SPI/UART | Trung bình | Trung bình | Trung bình |
| Khả năng xử lý | Cao | Cao | Hạn chế |
| Hỗ trợ đa nhiệm | Tốt | Trung bình | Hạn chế |
| Giá thành | Trung bình | Cao | Cao |

Bảng 2.2 So sánh MCU cho Gateway Node

Lựa chọn: ESP32-WROOM-32

Lý do: ESP32 có khả năng xử lý mạnh, hỗ trợ nhiều giao tiếp đồng thời, phù hợp làm node trung gian chuyển đổi dữ liệu giữa CAN và UART.

1. Máy tính nhúng Central Unit

|  |  |  |  |
| --- | --- | --- | --- |
| **Tiêu chí** | **Raspberry Pi 4** | **Raspberry Pi 3** | **Jetson Nano** |
| Hiệu năng CPU | **Cao** | Trung bình | Cao |
| Hỗ trợ camera | Tốt | Tốt | Rất tốt |
| Hỗ trợ hiển thị | HDMI | HDMI | HDMI |
| Giá thành | Trung bình | Thấp | Cao |

Bảng 2.3 So sánh máy tính nhúng cho Central Unit

Lựa chọn: Raspberry Pi 4 Model B

Lý do: Raspberry Pi 4 đáp ứng tốt các tác vụ xử lý ảnh, hiển thị giao diện bảng đồng hồ và giao tiếp dữ liệu với các node khác trong hệ thống.

1. Giao tiếp CAN

|  |  |  |  |
| --- | --- | --- | --- |
| **Tiêu chí** | **MCP2515** | **MCP2551** | **CAN tích hợp** |
| Chuẩn truyền | CAN 2.0A / 2.0B | CAN 2.0A / 2.0B | CAN |
| Giao tiếp với MCU | SPI | SPI | Tích hợp nội |
| Chống nhiễu | **Tốt** | Tốt | Tốt |
| Tốc độ truyền |  |  |  |
| Khoảng cách truyền | Dài | Dài | Dài |
| Phù hợp hệ thống ô tô | **Rất tốt** | Rất tốt | Rất tốt |

Bảng 2.4 So sánh thiết bị cho giao tiếp CAN

Lựa chọn: MCP2515 module

Lý do: MCP2515 là giải pháp phổ biến, đóng vai trò là bộ thu phát CAN, chuyển đổi tín hiệu logic mức TTL sang tín hiệu vi sai CANH/CANL. Giải pháp này đảm bảo khả năng chống nhiễu tốt, phù hợp với mô hình mạng CAN trong hệ thống ô tô, đồng thời dễ triển khai và có tính ổn định cao cho đề tài.

1. Cảm biến

|  |  |  |  |
| --- | --- | --- | --- |
| **Thành phần** | **Phương án khác** | **Lựa chọn** | **Lý do** |
| Cảm biến khoảng cách | IR | **HC-SR04** | Giá rẻ, dễ dùng |
| Đo vận tốc | Hall sensor | **Encoder quang** | Độ chính xác cao |
| Driver động cơ | L298N | **TB6612** | Nhỏ gọn, điều khiển mượt mà |
| Nhiệt độ nước mát | NTC B3950 10K | DS18B20 | Độ chính xác cao, giao tiếp số ổn định |

Bảng 2.5 So sánh lựa chọn cảm biến

### Thiết kế sơ đồ nguyên lý

Dựa trên sơ đồ khối tổng quát và lựa chọn linh kiện, hệ thống phần cứng được thiết kế theo hướng module hóa, trong đó mỗi node đảm nhiệm một chức năng riêng biệt. Việc thiết kế sơ đồ nguyên lý và mạch in được thực hiện nhằm đảm bảo tính ổn định, dễ dàng lắp ráp, bảo trì và mở rộng hệ thống.

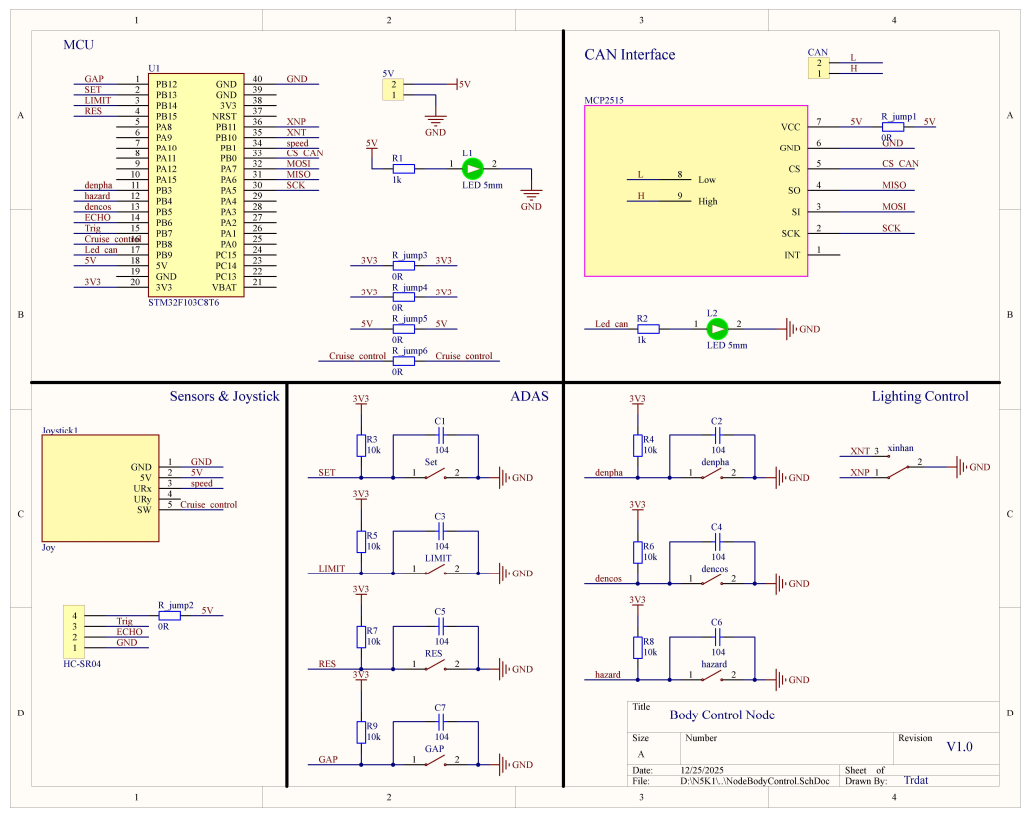
1. Body Control Node

Sơ đồ nguyên lý của BCN được thiết kế xoay quanh vi điều khiển STM32F103C8T6, đóng vai trò thu thập các tín hiệu điều khiển từ người lái. Các nút nhấn chức năng được kết nối trực tiếp đến các chân GPIO của vi điều khiển thông qua điện trở kéo lên, đảm bảo trạng thái logic ổn định và hạn chế nhiễu.

Joystick analog được kết nối đến kênh ADC1 của vi điều khiển để đọc giá trị điện áp biến thiên, phục vụ cho việc điều khiển tốc độ và hướng chuyển động. Cảm biến khoảng cách HC-SR04 được kết nối thông qua hai chân GPIO (PB7 cho Trig và PB6 Echo), cho phép đo khoảng cách vật cản phía trước.

Khối giao tiếp CAN sử dụng module MCP2515, giao tiếp với vi điều khiển thông qua chuẩn SPI. Các đường CAN\_H và CAN\_L được đưa ra đầu nối để kết nối vào mạng CAN Bus chung của hệ thống.

Mạch in của Body Control Node được thiết kế với kích thước nhỏ gọn, các khối chức năng được bố trí tách biệt rõ ràng giữa phần xử lý tín hiệu và phần giao tiếp, giúp giảm nhiễu và thuận tiện cho việc mở rộng.



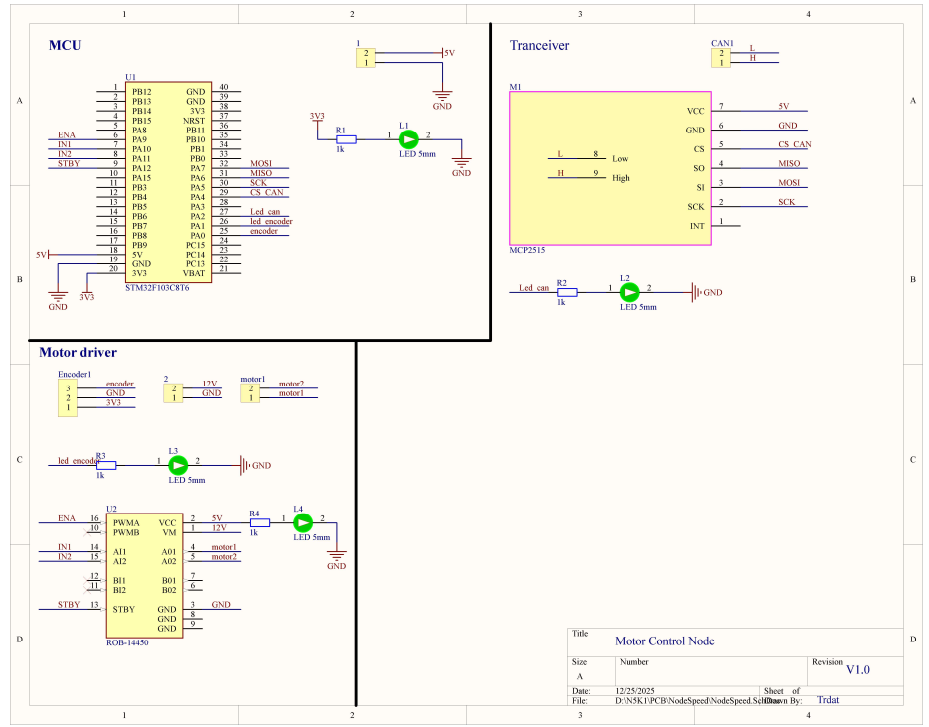
Hình 2.6 Schematic Diagram Body Control Node

1. Motor Control Node

MCN được thiết kế tập trung vào chức năng điều khiển động cơ và thu thập dữ liệu phản hồi vận tốc. Vi điều khiển STM32F103C8T6 nhận dữ liệu điều khiển từ mạng CAN và tạo tín hiệu PWM để điều khiển mạch cầu H.

Mạch cầu H (driver động cơ) được kết nối với các chân PWM của vi điều khiển, đảm nhiệm việc điều chỉnh điện áp cấp cho động cơ DC giảm tốc 12V. Để đo vận tốc, encoder quang được kết nối đến bộ Timer của vi điều khiển ở chế độ External Clock, cho phép đếm xung chính xác trong một khoảng thời gian xác định.

Tương tự Body Control Node, khối giao tiếp CAN sử dụng MCP2515, đảm bảo khả năng truyền nhận dữ liệu ổn định trên mạng CAN.



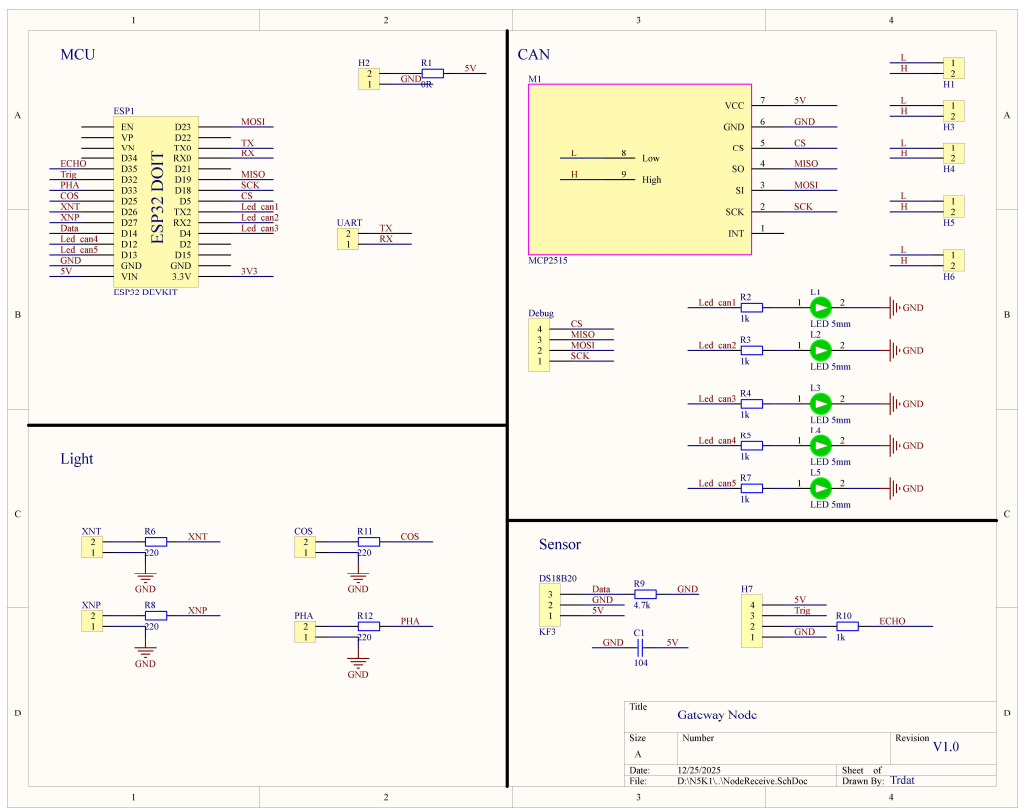
Hình 2.7 Sơ đồ nguyên lý Motor Control Node

1. Gateway Node

Sơ đồ nguyên lý Gateway Node sử dụng module ESP32-WROOM-32 làm vi điều khiển trung tâm. ESP32 giao tiếp với module MCP2515 thông qua chuẩn SPI để nhận và gửi dữ liệu trên mạng CAN Bus.

Bên cạnh đó, Gateway Node sử dụng cổng UART (TX/RX) để kết nối với Central Unit (Raspberry Pi). Các mức điện áp logic được thiết kế phù hợp nhằm đảm bảo tính tương thích và an toàn cho thiết bị.

Mạch in của Gateway Node được thiết kế nhỏ gọn, ưu tiên đường truyền SPI và UART ngắn gọn, đồng thời bố trí đầu nối CAN và UART rõ ràng để thuận tiện cho việc đấu nối và kiểm tra hệ thống.



Hình 2.8 Sơ đồ nguyên lý Gateway Node

1. Central Unit

Central Unit sử dụng máy tính nhúng Raspberry Pi 4 làm khối xử lý trung tâm. Do Raspberry Pi đã được tích hợp sẵn các cổng giao tiếp và bộ nguồn ổn định, nên trong đề tài này Central Unit không yêu cầu thiết kế mạch in riêng biệt.

Raspberry Pi được kết nối với màn hình LCD thông qua cổng HDMI để hiển thị giao diện bảng đồng hồ, đồng thời kết nối với camera để thu thập hình ảnh phía trước xe phục vụ thuật toán nhận diện biển báo giao thông.

Dữ liệu vận hành từ các node khác được truyền đến Central Unit thông qua Gateway Node bằng giao tiếp UART, cho phép Raspberry Pi cập nhật thông tin lên giao diện hiển thị và xử lý dữ liệu mức cao.

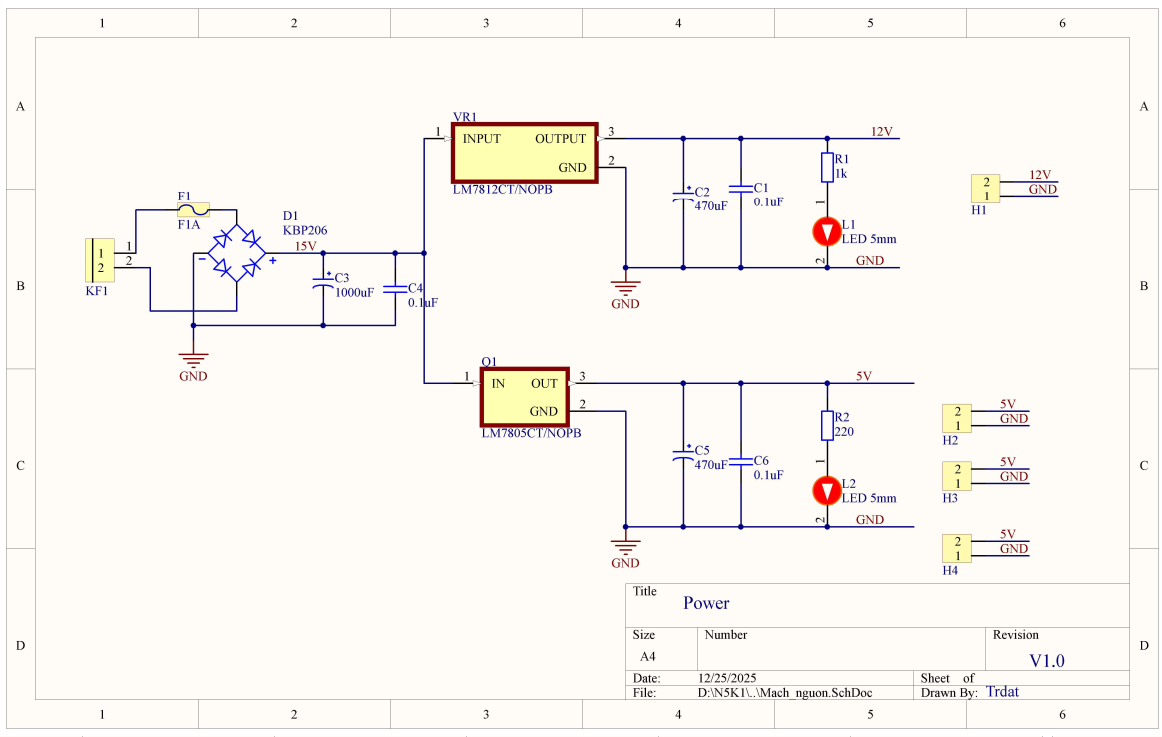
1. Mạch nguồn

Sơ đồ nguyên lý mạch nguồn được thiết kế nhằm cung cấp các mức điện áp 12V cho động cơ và 5V ổn định cho toàn bộ hệ thống. Mạch nguồn sử dụng cấu trúc nguồn tuyến tính với biến áp cách ly, đảm bảo an toàn điện và độ tin cậy trong quá trình vận hành.

Nguồn xoay chiều sau biến áp được chỉnh lưu toàn sóng thông qua cầu diode D1, sau đó được lọc bằng tụ điện dung lớn nhằm tạo ra điện áp một chiều tương đối ổn định. Các tụ bypass được bố trí song song để giảm nhiễu cao tần.

Điện áp một chiều sau chỉnh lưu được đưa vào IC ổn áp LM7812 để tạo ra nguồn 12V cấp cho driver động cơ. Đồng thời, IC LM7805 được sử dụng để tạo ra nguồn 5V cấp cho vi điều khiển, các module giao tiếp và cảm biến.

Mạch nguồn được tích hợp cầu chì bảo vệ ở ngõ vào, cùng với đèn LED báo trạng thái nguồn 12V và 5V, giúp dễ dàng giám sát tình trạng hoạt động của hệ thống.



Hình 2.9 Sơ đồ nguyên lý mạch nguồn

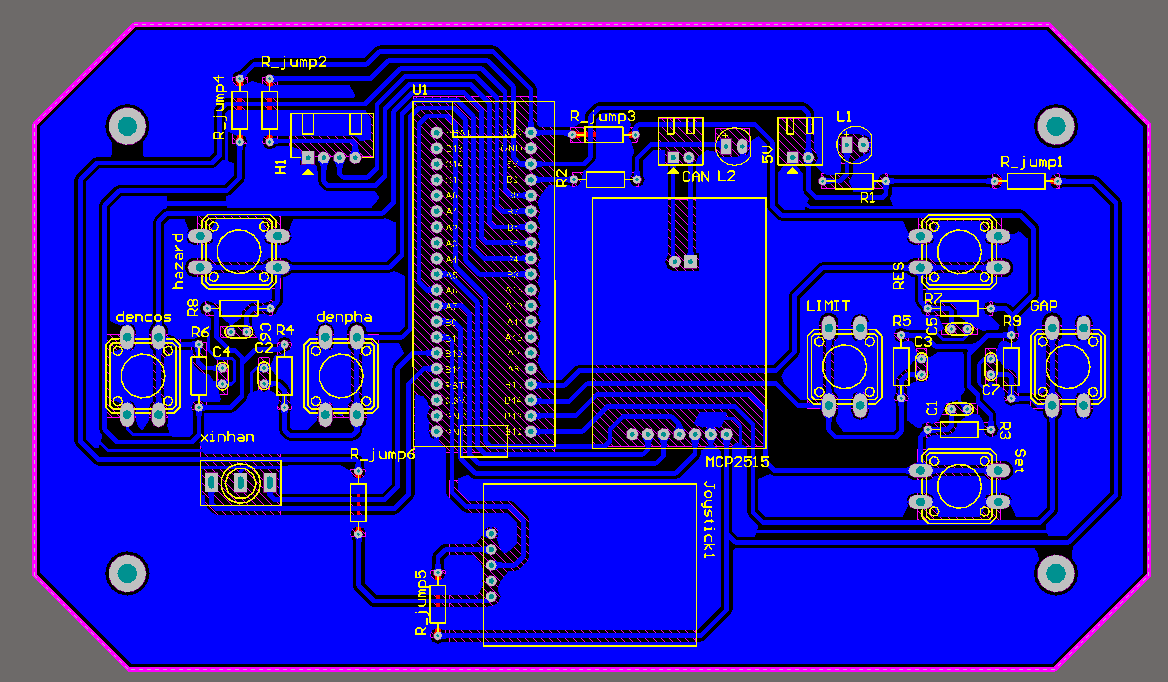
### 2.3.4 Thiết kế mạch in

Sau khi hoàn thiện sơ đồ nguyên lý, quá trình thiết kế mạch in được thực hiện với các tiêu chuẩn kỹ thuật nhằm đảm bảo khả năng gia công thủ công và độ ổn định khi vận hành.

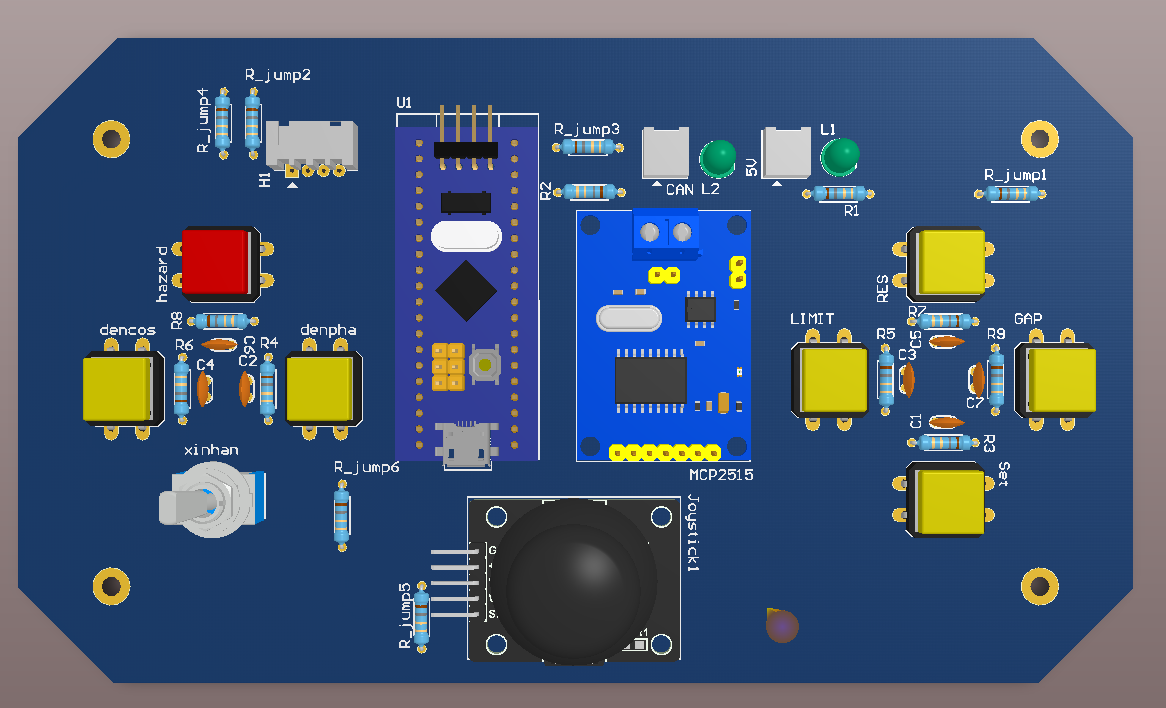
1. Body Control node

Đây là bo mạch điều khiển, tập trung nhiều nút nhấn và linh kiện đầu vào để giả lập các thao tác của người lái. Với các thành phần chính:

* Vi điều khiển: Module STM32F103C8T6 (Blue Pill).
* Giao tiếp: Module CAN MCP2515 nằm ngay cạnh vi điều khiển, đảm bảo đường truyền SPI ngắn nhất.
* Joystick: Cụm Joystick 2 trục dùng để điều động cơ.
* Nút nhấn chức năng: Các nút nhấn lớn màu vàng được quy hoạch rõ ràng: Đèn Pha, Đèn Cos, Xi nhan, và cụm nút điều khiển hành trình (SET, RES, GAP, LIMIT).
* Cảnh báo: Nút nhấn đỏ nổi bật dành cho chức năng đèn ưu tiên (Hazard).



Hình Layout mạch Body Control Node

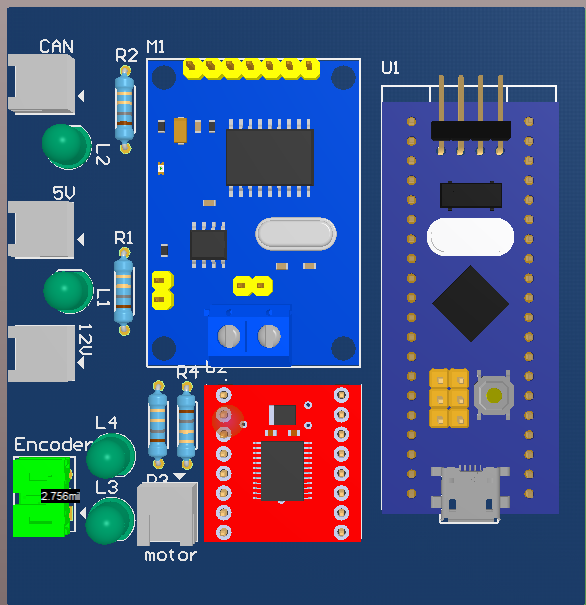


Hình Mô phỏng 3D mạch Body Control node

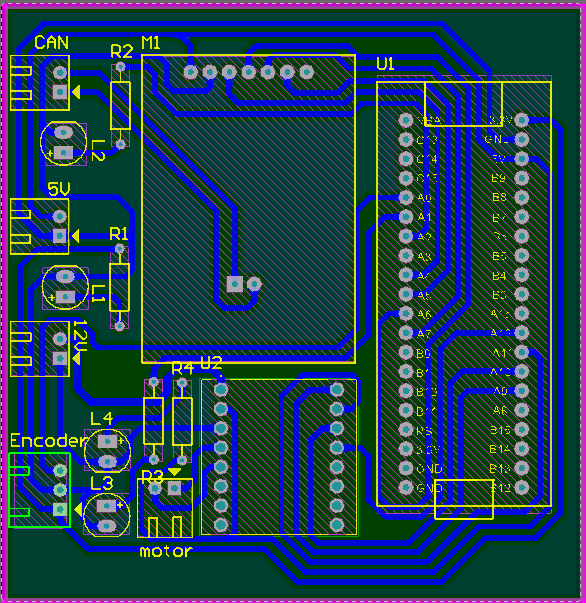
1. Motor Control Node

Mạch này thực hiện chức năng điều khiển tốc độ động cơ và đọc phản hồi từ Encoder. Các thành phần chính:

* Vi điều khiển: Module STM32F103C8T6 được cắm ở trung tâm.
* Driver động cơ: Module điều khiển động cơ TB6612, kết nối với STM32 qua các chân PWM.
* Cổng Encoder để đọc xung tốc độ.
* Cổng Motor cấp nguồn ra động cơ.
* Các cổng 12V, 5V, CAN được bố trí bên cạnh để cấp nguồn và giao tiếp.
* Chỉ thị: Tích hợp 4 đèn LED đơn (L1-L4) để báo trạng thái hoạt động.



Hình Layout mạch Motor Control Node

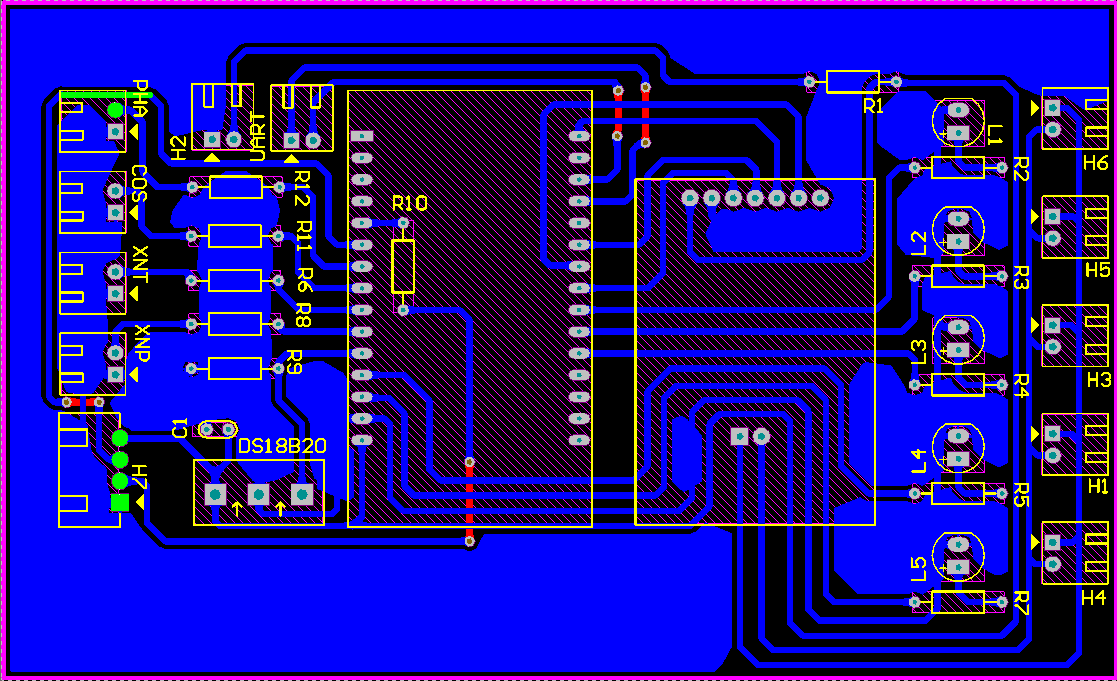


Hình Mô phỏng 3D Motor Control Node

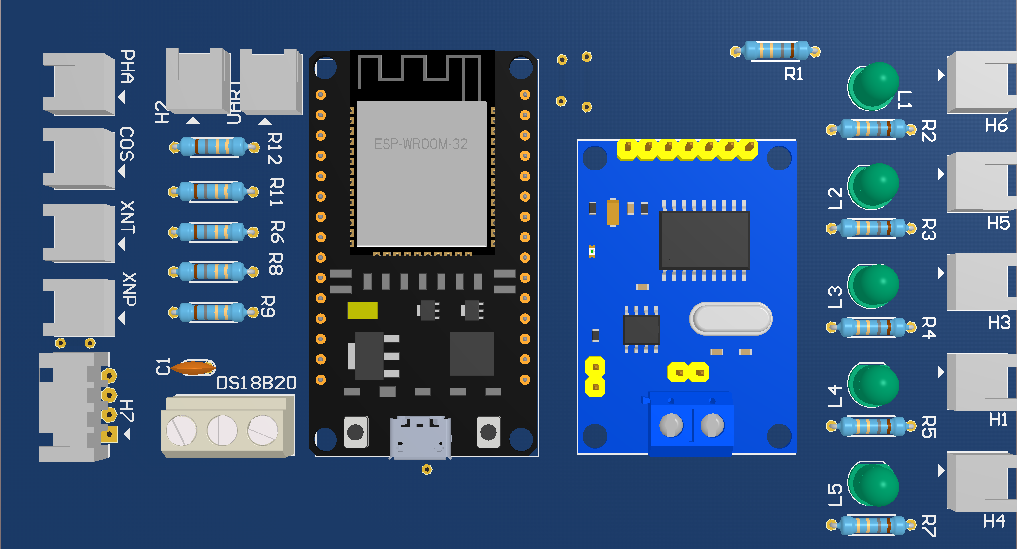
1. Gateway Node

Khác với thiết kế đơn thuần chỉ chuyển đổi dữ liệu, mạch Gateway trong mô hình này được tích hợp thêm chức năng điều khiển hệ thống đèn chiếu sáng và giám sát nhiệt độ nước mát.

* Vi xử lý trung tâm: Module ESP32-WROOM-32.
* Giao tiếp CAN: Module MCP2515 kết nối với mạng CAN của xe.
* Điều khiển đèn: Dãy cổng kết nối Header bên trái được dán nhãn cụ thể: PHA, COS, XNT (Xi nhan trái), XNP (Xi nhan phải). Điều này cho thấy ESP32 trực tiếp điều khiển các cụm đèn này dựa trên tín hiệu nhận được.
* Tích hợp cảm biến nhiệt độ DS18B20 và cầu đấu dây chờ sẵn, cho phép giám sát nhiệt độ nước mát.
* Chỉ thị: Dãy 5 đèn LED (L1-L5) báo trạng thái kết nối CAN.



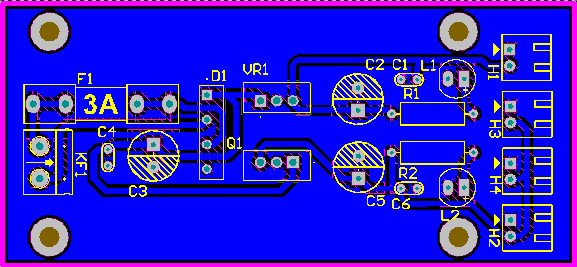
Hình Layout mạch Gateway Node



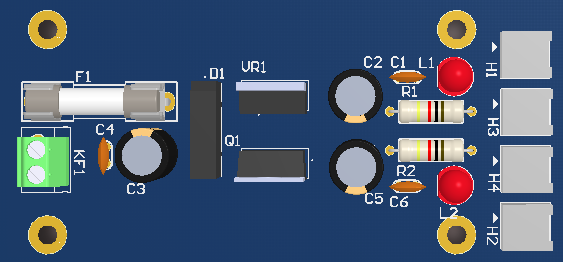
Hình Layout và mô phỏng 3D Gateway Node

1. Mạch nguồn

Để đảm bảo an toàn cho các module đắt tiền, một mạch nguồn riêng biệt được thiết kế. Mạch được tích hợp Cầu chì ống ngay tại đầu vào để ngắt mạch khi có sự cố quá dòng/ngắn mạch. Sử dụng các tụ hóa lọc nguồn dung lượng lớn (C2, C3, C5) kết hợp với tụ gốm để san phẳng điện áp. Các cổng ra (H1-H4) chia nguồn ổn định cho các Node chức năng (Motor, Body, Gateway), giúp đi dây gọn gàng từ một điểm tập trung.



Hình Layout mạch nguồn



Hình Mô phỏng 3D mạch nguồn

## 2.4 Thiết kế phần mềm

Phần mềm của hệ thống được thiết kế theo hướng module hóa, đảm bảo tính rõ ràng, dễ mở rộng và thuận tiện cho việc bảo trì. Các chức năng chính bao gồm xử lý hình ảnh, điều khiển hệ thống và hiển thị giao diện người dùng.

### 2.4.1 Thiết kế chức năng xử lý hình ảnh

Chức năng xử lý hình ảnh được triển khai trên Central Unit, sử dụng máy tính nhúng Raspberry Pi 4 Model B kết hợp với camera để thu thập dữ liệu hình ảnh từ môi trường phía trước xe. Mục tiêu của khối xử lý hình ảnh là nhận diện biển báo giao thông và cung cấp thông tin cho hệ thống điều khiển và hiển thị lên giao diện.

1. Xây dựng mô hình nhận diện

Mô hình nhận diện biển báo được xây dựng dựa trên kiến trúc mạng nơ-ron tích chập (CNN) MobileNetV2. Đây là kiến trúc được tối ưu hóa cho các thiết bị di động và nhúng nhờ sự cân bằng giữa độ chính xác và tốc độ xử lý.

Để tăng cường khả năng tổng quát hóa của mô hình và tránh hiện tượng Overfitting, kỹ thuật sinh ảnh tự động (Data Augmentation) được áp dụng. Từ tập dữ liệu gốc, các biến thể mới được tạo ra thông qua các phép biến đổi ngẫu nhiên:

* Xoay ảnh (Rotation): ± 15°.
* Dịch chuyển (Shift): Dịch ngang và dọc 10%.
* Biến dạng (Shear) và Phóng to/Thu nhỏ (Zoom): 10%.
* Thay đổi độ sáng (Brightness): Trong khoảng 0.9 - 1.1.

Quá trình huấn luyện sử dụng kỹ thuật Transfer Learning với trọng số từ ImageNet. Cấu trúc mạng được tùy chỉnh bằng cách loại bỏ các lớp Fully Connected gốc và thay thế bằng các lớp: GlobalAveragePooling2D, Dense (128 units, ReLU activation), và Dropout (0.3) để giảm thiểu Overfitting.

* Hàm mất mát: Categorical Crossentropy.
* Tối ưu hóa: Adam Optimizer với cơ chế giảm tốc độ học khi Loss không cải thiện.
* Chuyển đổi mô hình: Sau khi huấn luyện, mô hình được chuyển đổi sang định dạng TensorFlow Lite (.tflite) và áp dụng kỹ thuật lượng tử hóa (Quantization) để giảm kích thước và tăng tốc độ suy diễn trên Raspberry Pi.

1. Tiền xử lý ảnh

Trước khi đưa vào mô hình AI, hình ảnh thu được từ camera cần trải qua bước tiền xử lý để lọc nhiễu và xác định vị trí các biển báo tiềm năng. Quá trình này giúp giảm tải cho CPU bằng cách loại bỏ các vùng nền không cần thiết.

Sử dụng thư viện Picamera2 trên Raspberry Pi để thu hình ảnh định dạng RGB888 với độ phân giải 320x180 pixels, đảm bảo tốc độ khung hình cao. Sau đó chuyển đổi không gian màu từ BGR sang HSV. Hệ thống áp dụng bộ lọc màu để tách các đối tượng màu đỏ (đặc trưng của biển báo cấm và nguy hiểm). Dải màu đỏ được xác định trong hai khoảng HSV:

* Khoảng 1: H [0, 10], S [100, 255], V [100, 255].
* Khoảng 2: H [160, 180], S [100, 255], V [100, 255].

Sau khi áp dụng bộ lọc màu xác định được vùng có dải màu đỏ, sử dụng phép toán mở (Morphology Open) với kernel hình elip (5x5) để loại bỏ các điểm nhiễu nhỏ (noise) trên mặt nạ nhị phân. Các đường viền tìm được sẽ được kiểm tra các điều kiện hình học để xác định xem có phải là biển báo hay không:

Diện tích: Area ≥ 1500 pixel.

Tỷ lệ khung hình (Aspect Ratio): 0.7 ≤ ≤ 1.5.

Mật độ điểm ảnh (Pixel Ratio): Tỷ lệ điểm màu đỏ trong vùng ROI phải đạt ít nhất 30%.

Độ tròn (Circularity): Đối với biển báo Stop (bát giác/tròn), hệ thống kiểm tra thêm độ tròn dựa trên công thức:

Giá trị này phải lớn hơn 0.6 để được chấp nhận.

1. Nhận diện biển báo giao thông

Sau khi xác định được vùng quan tâm (ROI) chứa biển báo tiềm năng, quy trình nhận diện diễn ra như sau:

* Cắt và Chuẩn hóa: Vùng ROI được cắt ra từ khung hình gốc, thay đổi kích thước (Resize) về chuẩn đầu vào của mô hình (192x192 pixels) và chuẩn hóa giá trị điểm ảnh về khoảng [0, 1].
* Suy diễn (Inference): Sử dụng tflite\_runtime để thực hiện dự đoán lớp biển báo.
* Xác thực kết quả: Kết quả dự đoán chỉ được chấp nhận khi độ tin cậy lớn hơn ngưỡng thiết lập (0.85).
* Thuật toán ổn định nhận diện: Để tránh hiện tượng "nhấp nháy" (nhận diện sai trong tích tắc), hệ thống yêu cầu một biển báo phải được nhận diện giống nhau trong 4 khung hình liên tiếp trước khi đưa ra quyết định cuối cùng. Điều này tăng độ tin cậy của việc nhận diện biển báo, giúp mô hình không nhận diện nhầm trong trường hợp biển báo xuất hiện một phần trong khung hình

1. Xử lý kết quả và truyền dữ liệu

Kết quả nhận diện sau khi được xác thực sẽ được chuyển đổi thành các lệnh điều khiển và truyền đi qua giao thức UART. Các nhãn được ánh xạ sang các mã lệnh điều khiển tương ứng:

* speed\_limit\_40 → CMD\_LIMIT\_40
* speed\_limit\_50 → CMD\_LIMIT\_50
* stop → CMD\_STOP

Hệ thống gửi gói tin xuống Gateway node qua cổng /dev/ttyUSB0 với baudrate 115200. Định dạng gói tin: CMD\_NAME,Confidence. Hệ thống cài đặt thời gian chờ (SIGN\_HOLD\_TIME = 2s) để ngăn chặn việc gửi liên tục cùng một lệnh điều khiển, giúp hệ thống xe hoạt động ổn định hơn.

1. Đồng bộ với giao diện hiển thị

Để người lái hoặc người giám sát có thể theo dõi trạng thái, hệ thống xử lý ảnh đồng bộ dữ liệu với giao diện người dùng. Sử dụng Socket UDP để truyền dữ liệu nội bộ tới giao diện. Luồng dữ liệu được biểu diễn như sau:

* Gửi thông tin biển báo vừa nhận diện: AI:{Class\_ID}.
* Chức năng cầu nối: Hệ thống cũng đọc các dữ liệu cảm biến từ UART (do bộ điều khiển xe gửi lên) và chuyển tiếp qua UDP đến giao diện hiển thị, đảm bảo tính đồng bộ thời gian thực giữa phần cứng và phần mềm hiển thị.

### 2.4.2 Thiết kế module điều khiển

Module điều khiển nhúng được phát triển trên nền tảng STM32CubeIDE, sử dụng thư viện HAL để tương tác với phần cứng. Kiến trúc phần mềm được thiết kế theo mô hình vòng lặp vô hạn kết hợp các hàm chức năng, đảm bảo tính đơn giản nhưng vẫn đáp ứng được yêu cầu thời gian thực.

1. Thuật toán Body Control Node

Nhiệm vụ chính của Node này là chuyển đổi các tín hiệu vật lý từ người lái thành dữ liệu số để gửi lên mạng CAN.

Do đặc thù của biến trở Joystick thường bị nhiễu tín hiệu, hệ thống áp dụng thuật toán Lọc trung bình (Average Filter).Vi điều khiển thực hiện lấy mẫu ADC liên tiếp 5 lần, sau đó tính giá trị trung bình để loại bỏ các gai nhiễu ngẫu nhiên trước khi sử dụng.Thiết lập vùng chết (Deadzone) khoảng ±15 đơn vị quanh giá trị trung tâm để tránh hiện tượng xe tự trôi khi cần gạt ở vị trí nghỉ.

Các nút nhấn được quét liên tục (Polling) trong vòng lặp chính. Cảm biến siêu âm HC-SR04 được điều khiển bằng Timer để đo độ rộng xung phản hồi, từ đó tính toán khoảng cách vật cản theo công thức:

Toàn bộ trạng thái nút nhấn, giá trị Joystick và khoảng cách được đóng gói vào một khung truyền (Data Frame) với ID định danh là 0x01. Chu kỳ gửi tin là 50ms để đảm bảo độ mượt mà.

1. Thuật toán Motor Control Node

Đây là node có logic xử lý phức tạp nhất, bao gồm điều khiển động cơ và đo lường phản hồi.

Dữ liệu điều khiển (từ Joystick hoặc lệnh AI) được ánh xạ (Mapping) sang độ rộng xung PWM.Timer 1 được cấu hình để xuất xung PWM tần số 1kHz. Độ rộng xung thay đổi từ 0 đến 624 tương ứng với công suất động cơ từ 0% đến 100%.

Logic điều khiển chiều quay được lập trình để ưu tiên an toàn: Khi đảo chiều đột ngột, hệ thống sẽ đưa PWM về 0 trong một khoảng thời gian ngắn để bảo vệ hộp số và Driver.

Sử dụng Timer 2 ở chế độ External Clock Mode để đếm sườn xung từ Encoder. Do đặc điểm của động cơ chổi than gây nhiễu điện từ lớn, thuật toán Trung bình trượt (Moving Average) kích thước cửa sổ N=10 được áp dụng. Vận tốc tức thời là trung bình cộng của 10 lần đo gần nhất.

Thay vì sử dụng công thức vật lý phức tạp, hệ thống sử dụng phương pháp thực nghiệm. Vận tốc (V) được tính dựa trên số xung đếm được trong 100ms (Pfiltered) và số xung tối đa tương ứng với 100km/h (Pmax ≈ 14):

Giá trị này sau đó được gửi ngược lên mạng CAN (ID 0x02) để hiển thị trên giao diện.

1. Thuật toán Gateway Node

Node Gateway thực hiện giao tiếp giữa hai giao thức truyền thông không đồng bộ.

Luồng CAN to UART: Sử dụng ngắt (Interrupt) để bắt các gói tin CAN ngay khi chúng xuất hiện trên đường truyền. Dữ liệu Payload từ CAN được giải mã (Decode) và định dạng lại thành chuỗi ký tự theo cấu trúc JSON rồi gửi qua cổng Serial tới Raspberry Pi.

Luồng UART to CAN: Lắng nghe bộ đệm Serial. Khi nhận được chuỗi lệnh từ Raspberry Pi (ví dụ từ module xử lý ảnh: CMD\_STOP), ESP32 sẽ phân tích cú pháp (Parsing) để xác định loại lệnh.

Dựa trên loại lệnh, nó tạo ra một CAN Frame tương ứng (ví dụ ID 0x03 cho lệnh phanh khẩn cấp) và đẩy xuống CAN Bus để Node Động cơ thực thi.

1. Thuật toán Central Node

Central Node đóng vai trò là khối xử lý trung tâm của toàn hệ thống, chịu trách nhiệm tiếp nhận dữ liệu từ các node ngoại vi, xử lý thông tin mức cao và hiển thị trạng thái vận hành của hệ thống theo thời gian thực. Thuật toán tại Central Node được thiết kế theo mô hình đa luồng nhằm đảm bảo khả năng xử lý song song giữa nhận dữ liệu, xử lý hình ảnh và cập nhật giao diện.

Luồng UART từ Gateway Node: Central Node liên tục lắng nghe dữ liệu từ cổng UART để tiếp nhận các gói tin trạng thái và điều khiển do Gateway Node gửi lên. Khi có dữ liệu mới, chuỗi ký tự nhận được sẽ được giải mã và phân tích cú pháp nhằm trích xuất các thông tin cần thiết như vận tốc, trạng thái động cơ và tín hiệu cảnh báo. Các dữ liệu sau khi xử lý sẽ được đồng bộ và chuyển tới khối giao diện hiển thị.

Luồng xử lý hình ảnh: Camera được khởi tạo và hoạt động liên tục để thu thập hình ảnh môi trường phía trước xe. Mỗi khung hình thu được sẽ được đưa vào khối xử lý ảnh, nơi mô hình học sâu (TensorFlow Lite) thực hiện nhận diện biển báo giao thông. Khi phát hiện biển báo hợp lệ, Central Node xác định loại biển báo và sinh ra lệnh điều khiển tương ứng (ví dụ giảm tốc, dừng xe).

Luồng giao tiếp điều khiển: Các lệnh điều khiển được sinh ra từ khối xử lý hình ảnh hoặc từ logic hệ thống sẽ được đóng gói thành chuỗi dữ liệu và gửi ngược lại Gateway Node thông qua giao tiếp UART. Gateway Node sau đó chuyển đổi các lệnh này thành frame CAN phù hợp để truyền xuống các node điều khiển cấp thấp.

Luồng giao diện: Central Node cập nhật dữ liệu thời gian thực lên giao diện bảng đồng hồ thông qua cơ chế liên kết tín hiệu giữa Backend và Frontend. Khi có dữ liệu mới từ UART hoặc từ khối xử lý ảnh, giao diện sẽ tự động cập nhật các thành phần hiển thị như kim đồng hồ, biểu tượng cảnh báo và trạng thái hệ thống mà không làm gián đoạn các luồng xử lý khác.

Thuật toán Central Node được tổ chức theo kiến trúc phân luồng độc lập, giúp hệ thống hoạt động ổn định, giảm độ trễ và đảm bảo khả năng mở rộng trong các ứng dụng hỗ trợ lái xe thông minh.

### 2.4.3 Thiết kế giao diện người dùng

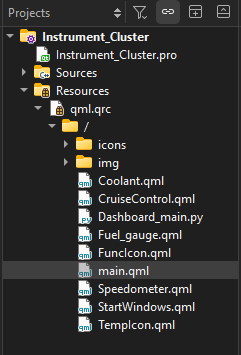
Giao diện người dùng của hệ thống được thiết kế và xây dựng bằng Qt Creator, sử dụng QML (Qt Modeling Language). Việc thiết kế giao diện theo hướng lập trình QML thuần cho phép linh hoạt trong bố cục, dễ dàng mở rộng và phù hợp với các hệ thống nhúng sử dụng màn hình hiển thị.

1. Kiến trúc giao diện

Giao diện được xây dựng theo mô hình module hóa, trong đó mỗi thành phần giao diện được đóng gói thành một file QML riêng biệt. File main.qml đóng vai trò là giao diện chính, chịu trách nhiệm khởi tạo cửa sổ, quản lý bố cục tổng thể và gọi các thành phần con. Một số module giao diện chính bao gồm:

* Speedometer.qml: Hiển thị đồng hồ tốc độ.
* Fuel\_gauge.qml: Hiển thị mức nhiên liệu.
* Coolant.qml: Hiển thị nhiệt độ nước làm mát.
* CruiseControl.qml: Hiển thị trạng thái hệ thống Cruise Control.
* Function.qml: Hiển thị các biểu tượng chức năng và cảnh báo.

Cách tổ chức này giúp giao diện rõ ràng, dễ chỉnh sửa và thuận tiện trong việc tái sử dụng các thành phần.



Hình 2.10 Cây thư mục giao diện

1. Thiết kế đồ họa và hiển thị

Các thành phần đồ họa của giao diện như nền bảng đồng hồ, kim chỉ, biểu tượng chức năng và các icon cảnh báo được xây dựng theo hướng module hóa, trong đó mỗi nhóm chức năng được thiết kế dưới dạng một module QML riêng biệt. Bên trong mỗi module, các thành phần đồ họa được tạo từ các hình ảnh tĩnh (PNG) và được nạp thủ công thông qua đối tượng Image trong QML.

Việc tổ chức các icon và thành phần hiển thị thành các module độc lập giúp giao diện dễ quản lý, thuận tiện cho việc chỉnh sửa giao diện từng phần mà không ảnh hưởng đến cấu trúc tổng thể. Đồng thời, các module này có thể được tái sử dụng hoặc mở rộng khi bổ sung thêm chức năng hiển thị mới.

Giao diện sử dụng các thuộc tính bố cục như anchors để căn chỉnh vị trí, scale để điều chỉnh tỷ lệ hiển thị và rotation để tạo chuyển động quay cho các thành phần động như kim chỉ đồng hồ. Các tham số hiển thị được tính toán dựa trên kích thước cửa sổ chính, cho phép giao diện thích nghi với các độ phân giải màn hình khác nhau mà vẫn đảm bảo tính thẩm mỹ và trực quan.

1. Liên kết dữ liệu và cập nhật trạng thái

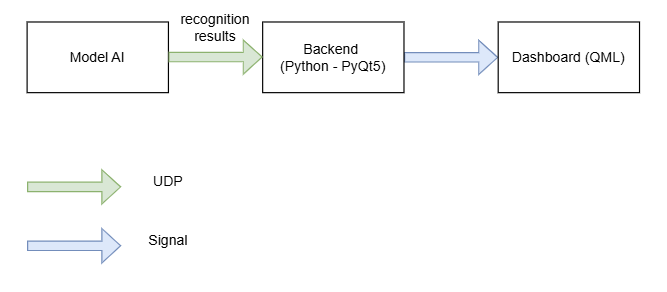
Để kết nối dữ liệu giữa khối xử lý trung tâm và giao diện hiển thị, hệ thống sử dụng một lớp trung gian được xây dựng bằng Python (PyQt5). Chương trình backend được triển khai trên Raspberry Pi, có nhiệm vụ khởi chạy giao diện QML, đồng thời tiếp nhận và phân phối dữ liệu từ các khối xử lý khác trong hệ thống.

Giao diện được hiển thị thông qua QQmlApplicationEngine, trong đó file main.qml đóng vai trò là giao diện chính. Một đối tượng backend được tạo trong Python và được nhúng vào môi trường QML thông qua cơ chế Context Property, cho phép giao diện QML truy cập trực tiếp đến các tín hiệu và dữ liệu do backend cung cấp.

Dữ liệu đầu vào của giao diện được truyền đến backend thông qua giao thức UDP, với một luồng riêng biệt thực hiện chức năng lắng nghe dữ liệu. Việc sử dụng giao tiếp UDP giúp giảm độ trễ và phù hợp với các dữ liệu trạng thái được cập nhật liên tục theo thời gian thực, như kết quả nhận diện biển báo giao thông hoặc trạng thái vận hành của hệ thống.

Khi backend nhận được dữ liệu từ cổng UDP, dữ liệu sẽ được xử lý sơ bộ và phát đi thông qua tín hiệu (signal). Các tín hiệu này được liên kết với các thành phần trong giao diện QML, cho phép cập nhật trực tiếp các thông số hiển thị mà không cần can thiệp vào luồng xử lý giao diện.

Cơ chế liên kết giữa Python và QML giúp tách biệt rõ ràng giữa phần xử lý dữ liệu và phần hiển thị, đảm bảo giao diện hoạt động mượt mà, đồng thời tăng tính linh hoạt khi mở rộng hoặc thay đổi thuật toán xử lý trong tương lai.



Hình 2.11 Data flow diagram

1. Hiệu ứng và tương tác

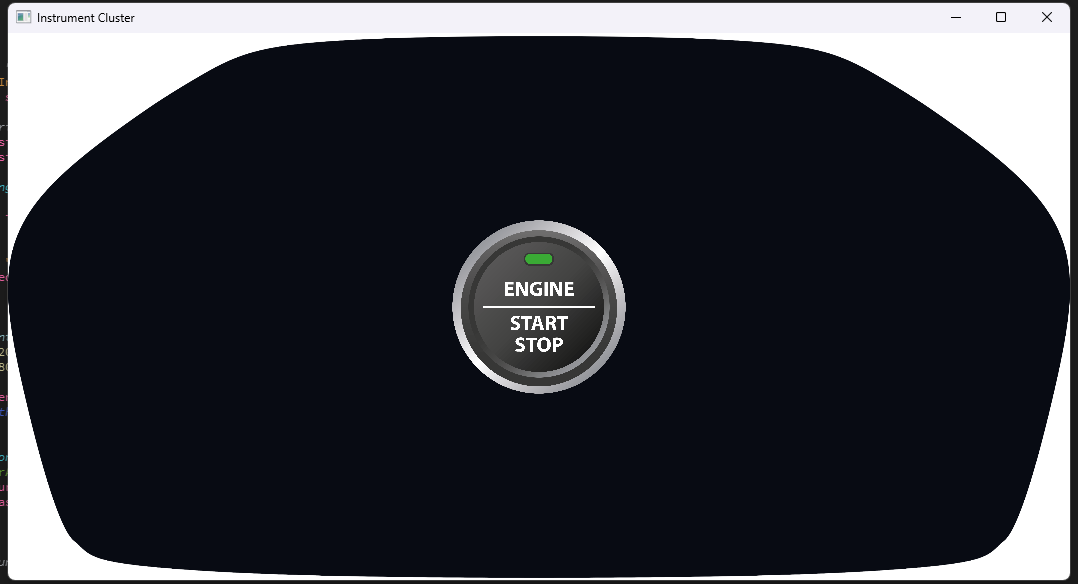
Để tăng tính trực quan, giao diện được tích hợp các hiệu ứng chuyển động và hoạt ảnh như:

* Hiệu ứng quay kim đồng hồ dựa trên giá trị vận tốc.
* Hiệu ứng chuyển cảnh và làm mờ khi thay đổi trạng thái hiển thị.
* Phản hồi trực quan khi kích hoạt các chế độ chức năng.

Các hiệu ứng được xây dựng bằng các thành phần chuẩn của QML như Behavior, NumberAnimation và Easing, giúp giao diện sinh động nhưng vẫn đảm bảo hiệu năng.

1. Khả năng mở rộng và tích hợp

Thiết kế giao diện theo hướng QML module hóa cho phép dễ dàng tích hợp với các khối xử lý khác trong hệ thống, đặc biệt là khối xử lý hình ảnh và khối điều khiển. Giao diện có thể mở rộng để hiển thị thêm các thông tin như trạng thái nhận diện biển báo giao thông, dữ liệu vận hành hoặc cảnh báo hệ thống mà không cần thay đổi cấu trúc tổng thể.



Hình 2.12 Giao diện khởi động



Hình 2.13 Giao diện Instrument Cluster

## 2.5 Cấu hình giao thức giao tiếp

Để đảm bảo việc trao đổi dữ liệu giữa các module trong hệ thống diễn ra chính xác và ổn định, các giao thức giao tiếp CAN, UART và UDP được cấu hình với định dạng dữ liệu và thông số truyền phù hợp. Nội dung mục này trình bày chi tiết cách định dạng gói tin và cấu hình truyền thông cho từng giao thức được sử dụng trong đề tài.

### 2.5.1 Định dạng các gói tin CAN

Giao thức CAN được sử dụng để truyền dữ liệu điều khiển và trạng thái giữa các node nhúng trong hệ thống, cấu hình với tốc độ 250kbps, định dạng khung chuẩn (Standard Frame 2.0A). Hệ thống định nghĩa 02 gói tin chính để trao đổi dữ liệu điều khiển và trạng thái.

1. Cấu trúc gói tin Body Control Node

Sender: Node Body Control (STM32F103)

Receiver: Node Speed (STM32F103), Node Gateway (ESP32)

Chu kỳ gửi: 50ms

|  |  |  |
| --- | --- | --- |
| Byte index | Data name | Detailed description |
| Byte 0 | Button Flags | Trạng thái các nút bấm ADAS |
|  |  | Bit 0: Cruise Activate |
|  |  | Bit 1: SET/- |
|  |  | Bit 2: RES/+ |
|  |  | Bit 3: GAP |
|  |  | Bit 4: LIMIT |
| Byte 1 | Light Flags | Trạng thái hệ thống đèn |
|  |  | Bit 0: High Beam |
|  |  | Bit 1: Low Beam |
|  |  | Bit 2: Left turn signal |
|  |  | Bit 3: Right turn signal |
|  |  | Bit 4: Hazard |
| Byte 2 | Joystick low | 8 bit thấp của giá trị Joystick (ADC) |
| Byte 3 | Joystick high | 8 bit cao của giá trị Joystick (ADC) |
| Byte 4 | Distance low | 8 bit thấp của khoảng cách |
| Byte 5 | Distance high | 8 bit cao của khoảng cách |
| Byte 6 | Gap level | Cấp độ khoảng cách an toàn ACC |
| Byte 7 | Target speed | Tốc độ mong muốn do ADAS tính toán |

Bảng 2.6 Dữ liệu Body Control Node

1. Cấu trúc gói tin Motor Control Node

Sender: Node Speed (STM32F103)

Receiver: Node Gateway (ESP32)

Chu kỳ gửi: 50ms

|  |  |  |
| --- | --- | --- |
| Byte index | Data name | Detailed description |
| Byte 0 - 3 | Encoder Freq | Tần số xung Encoder (32-bit integer), dùng để debug tín hiệu thô |
| Byte 4 | Speed Low | 8 bit cao của tốc độ thực tế (Real Speed). |
| Byte 5 | Speed High | 8 bit cao của tốc độ thực tế (Real Speed). |

Bảng 2.7 Dữ liệu Motor Control Node

### 2.5.2 Cấu hình UART

Trong thiết kế Node Gateway, giao thức UART đóng vai trò cổng kết nối chính để truyền tải dữ liệu thu thập được từ hệ thống mạng CAN lên Central Unit. Vi điều khiển ESP32 được cấu hình sử dụng bộ HardwareSerial (UART0) để đảm bảo độ ổn định cao nhất.

Các tham số truyền thông được thiết lập với tốc độ Baud là 115200 bps nhằm đáp ứng khả năng truyền tải nhanh chóng lượng dữ liệu lớn từ các cảm biến và trạng thái xe. Định dạng khung truyền được quy định: 8 bit dữ liệu, không kiểm tra chẵn lẻ (No Parity) và 1 Stop bit. Nhờ tính năng GPIO Matrix của ESP32, các chân truyền (TX) và nhận (RX) được ánh xạ linh hoạt tương ứng với thiết kế mạch in, đồng thời hệ thống cũng sử dụng bộ đệm (Buffer) kích thước lớn để tránh hiện tượng tràn dữ liệu (overflow) trong quá trình xử lý các tác vụ mạng không dây song song.

### 2.5.3 Cấu hình UDP

Để đảm bảo hiệu năng xử lý hình ảnh và độ mượt mà của giao diện hiển thị hoạt động độc lập, hệ thống sử dụng giao thức UDP như một kênh giao tiếp nội bộ (Local Inter-process Communication). Phương pháp này giúp tách biệt hoàn toàn luồng xử lý dữ liệu và luồng hiển thị, tránh hiện tượng treo giao diện khi hệ thống xử lý tác vụ nặng.

1. Mô hình kiến trúc gửi – nhận

Hệ thống được chia thành hai tiến trình độc lập chạy song song trên máy tính nhúng (Raspberry Pi):

* Tiến trình Xử lý trung tâm (Backend - Sender) được đảm nhiệm bởi module final.py. Có chức năng thực hiện đọc dữ liệu cảm biến từ vi điều khiển ESP32 qua cổng UART (tốc độ Baud 115200) và chạy thuật toán nhận diện biển báo giao thông. Dữ liệu sau khi xử lý được đóng gói và gửi đi qua socket UDP tới địa chỉ Loopback 127.0.0.1 tại cổng 5005.
* Tiến trình Hiển thị (Frontend - Receiver): Được xây dựng trên nền tảng Qt (PyQt5/QML) trong module Dashboard\_main.py. Có chức năng lắng nghe liên tục tại cổng 5005 trên luồng riêng biệt để cập nhật thông số lên màn hình Instrument Cluster mà không làm gián đoạn luồng vẽ đồ họa chính.

1. Cấu trúc gói tin

Do giao tiếp diễn ra trong nội bộ thiết bị với tốc độ cao, định dạng gói tin được thiết kế dưới dạng chuỗi ký tự (String-based) để thuận tiện cho việc gỡ lỗi và giải mã:

|  |  |  |  |
| --- | --- | --- | --- |
| Data type | Cú pháp | Nguồn dữ liệu | Description |
| Trạng thái xe | SPD:x, PHA:y, … | ESP32 (UART) | Dữ liệu vận hành nhận từ vi điều khiển, được chuyển tiếp nguyên trạng. |
| Cảnh báo AI | AI:[Class\_ID] | Raspberry Pi | Kết quả nhận diện biển báo (VD: AI:5 tương ứng với biển STOP). |

Bảng 2.8 Định dạng gói tin giao tiếp UDP nội bộ

# CHƯƠNG III: TRIỂN KHAI VÀ ĐÁNH GIÁ HỆ THỐNG

## 3.1 Quy trình triển khai hệ thống

### 3.1.1 Công cụ và môi trường phát triển

Để hiện thực hóa hệ thống từ bản vẽ thiết kế sang sản phẩm thực tế, quá trình triển khai sử dụng phối hợp các công cụ phần cứng và phần mềm chuyên dụng cho hệ thống nhúng và xử lý ảnh.

1. Phần mềm và môi trường lập trình

Altium Designer: Được sử dụng để thiết kế sơ đồ nguyên lý và vẽ mạch in cho các node điều khiển và Gateway. Công cụ hỗ trợ xuất file PDF để gia công mạch và kiểm tra quy tắc thiết kế trước khi thi công.

STM32CubeMX: Công cụ đồ họa được sử dụng để cấu hình STM32F103 một cách trực quan. Phần mềm hỗ trợ thiết lập sơ đồ xung nhịp, gán chức năng cho các chân GPIO, cấu hình tham số cho bộ truyền thông ADC, UART, Timer. Sau khi cấu hình, công cụ tự động sinh mã nguồn khởi tạo tương thích với chuẩn CMSIS để chuyển sang môi trường Keil C.

Keil µVision 5: Môi trường phát triển tích hợp chính cho dòng vi điều khiển STM32. Sử dụng trình biên dịch ARM C/C++ để viết firmware điều khiển mức thấp cho ngoại vi SPI, ADC, Timer và GPIO.

Arduino IDE: Sử dụng để phát triển firmware cho Node Gateway. Môi trường này cung cấp các thư viện hỗ trợ mạnh mẽ cho FreeRTOS và ngăn xếp giao thức mạng, giúp rút ngắn thời gian phát triển ứng dụng.

Visual Studio Code: Trình soạn thảo mã nguồn đa năng được sử dụng để phát triển phần mềm trên Raspberry Pi.

Môi trường Qt/QML: Hỗ trợ xây dựng giao diện người dùng thông qua thư viện PyQt5.

RealVNC Viewer: Để truy cập và điều khiển Raspberry Pi từ xa không cần màn hình rời.

Putty: Terminal trên máy tính dùng để giám sát dữ liệu debug qua cổng Serial/UART. Bên cạnh đó, phần mềm này còn dùng để thiết lập các chức năng ban đầu cho Raspberry Pi.

WinSCP: Phần mềm quản lý file nguồn mở sử dụng giao thức SFTP/FTP. Trong quá trình phát triển, WinSCP đóng vai trò quan trọng trong việc truyền tải dữ liệu, sao chép mã nguồn và cập nhật mô hình AI từ máy tính cá nhân sang hệ điều hành Linux của Raspberry Pi một cách nhanh chóng qua mạng.

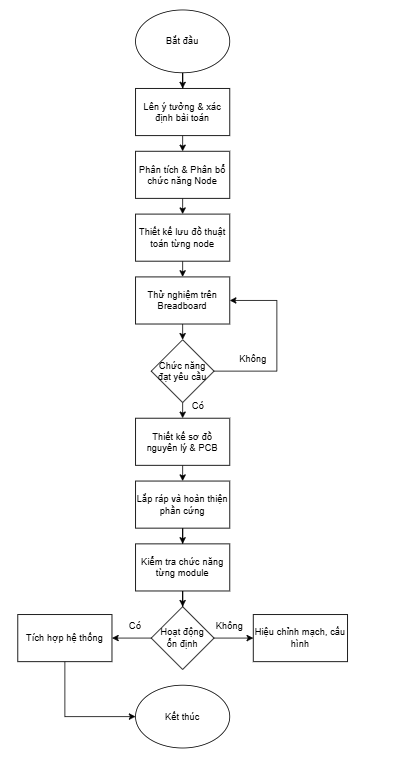
1. Công cụ phần cứng và nạp gỡ lỗi

Mạch nạp ST-Link V2: Sử dụng để nạp chương trình và gỡ lỗi thời gian thực cho vi điều khiển STM32F103.

Bộ phân tích logic: Bộ công cụ được sử dụng để kiểm tra và xác thực gói tin CAN. Thiết bị nối với CAN\_H, CAN\_L trên thiết bị truyền để kiểm tra xem dữ liệu gửi đi chứa những gì. Qua phần mềm phân tích trên máy tính, các tín hiệu mức thấp được giải mã thành các khung dữ liệu CAN và các cờ báo lỗi

### 3.1.2 Quy trình tích hợp hệ thống

Quá trình triển khai phần cứng của hệ thống được thực hiện theo một quy trình có hệ thống, bao gồm các bước từ hình thành ý tưởng, thiết kế, thử nghiệm cho đến lắp ráp và đánh giá, nhằm đảm bảo tính đúng đắn, ổn định và khả năng mở rộng của đề tài.



Hình 3.1 Lưu đồ quy trình triển khai hệ thống

Bước 1: Lên ý tưởng và xác định bài toán

Xuất phát từ yêu cầu mô phỏng hệ thống điều khiển và hiển thị thông tin trên ô tô, đề tài được định hướng xây dựng một hệ thống nhúng phân tán, trong đó các chức năng được chia thành nhiều node độc lập, giao tiếp với nhau thông qua mạng CAN Bus. Hệ thống phải đảm bảo khả năng điều khiển, giám sát và hiển thị dữ liệu theo thời gian thực.

Bước 2: Liệt kê và phân tích chức năng hệ thống

Dựa trên yêu cầu đề tài, các chức năng chính của hệ thống được xác định và phân bổ cho từng node, bao gồm:

* Thu thập tín hiệu điều khiển từ người dùng (Body Control Node)
* Điều khiển động cơ và phản hồi vận tốc (Motor Control Node)
* Trung gian giao tiếp và chuyển đổi dữ liệu (Gateway Node)
* Xử lý hình ảnh, nhận diện biển báo và hiển thị giao diện (Central Unit)

Việc phân chia chức năng giúp hệ thống có tính module hóa cao, dễ dàng mở rộng và bảo trì.

Bước 3: Thiết kế lưu đồ thuật toán cho từng node

Trên cơ sở các chức năng đã xác định, lưu đồ thuật toán cho từng node được xây dựng nhằm mô tả rõ trình tự xử lý dữ liệu, điều kiện hoạt động và luồng giao tiếp giữa các module. Các lưu đồ này đóng vai trò là cơ sở để triển khai phần mềm nhúng và kiểm tra logic hoạt động của hệ thống.

Bước 4: Thử nghiệm nguyên lý trên breadboard

Trước khi thiết kế mạch in, các module phần cứng được thử nghiệm nguyên lý trên breadboard. Các linh kiện như vi điều khiển, cảm biến, module CAN, driver động cơ và encoder được kết nối tạm thời để:

* Kiểm tra hoạt động của từng linh kiện
* Đánh giá khả năng giao tiếp giữa các module
* Phát hiện sớm các lỗi kết nối hoặc xung đột tín hiệu

Bước 5: Thiết kế sơ đồ nguyên lý và mạch in

Sau khi thử nghiệm thành công trên breadboard, sơ đồ nguyên lý chi tiết của từng node được thiết kế bằng phần mềm chuyên dụng. Dựa trên sơ đồ nguyên lý, mạch in được thiết kế nhằm đảm bảo tính gọn gàng, ổn định điện áp, giảm nhiễu và thuận tiện cho việc lắp ráp.

Bước 6: Lắp ráp linh kiện và hoàn thiện phần cứng

Các linh kiện được hàn và lắp ráp lên mạch in theo đúng sơ đồ đã thiết kế. Quá trình lắp ráp được thực hiện cẩn thận nhằm đảm bảo độ tin cậy cơ khí và điện của hệ thống.

Bước 7: Kiểm tra và đánh giá chức năng từng module

Sau khi hoàn thiện lắp ráp, từng module được kiểm tra độc lập để xác nhận:

* Hoạt động đúng chức năng thiết kế.
* Giao tiếp ổn định qua CAN Bus, UART và các giao thức liên quan.
* Không xảy ra lỗi nguồn, nhiễu hoặc mất dữ liệu.

## 3.2 Triển khai phần mềm

### 3.2.1 Lập trình các node điều khiển

Phần mềm điều khiển cho Node Body và Node Motor được phát triển trên nền tảng vi điều khiển STM32F103, sử dụng ngôn ngữ lập trình C trong môi trường phát triển tích hợp Keil µVision 5. Do kiến trúc phần cứng sử dụng module MCP2515 để giao tiếp trên CAN bus, phần mềm được xây dựng dựa trên giao thức giao tiếp SPI thay vì sử dụng bộ CAN tích hợp sẵn trên chip.

Quy trình xử lý và cấu trúc phần mềm được tổ chức thành các phân hệ chức năng như sau:

1. Cấu hình ngoại vi

Các tài nguyên phần cứng của vi điều khiển được thiết lập thông qua bộ thư viện HAL và STM32CubeMX để phục vụ các chức năng chuyên biệt:

Giao tiếp SPI: Do hệ thống sử dụng bộ điều khiển CAN rời, khối SPI1 được cấu hình ở chế độ Master với tốc độ truyền cao để giao tiếp với IC MCP2515. Các chân tín hiệu SCK, MISO, MOSI được thiết lập tương ứng, cùng với chân Chip Select được điều khiển thủ công bằng GPIO để đảm bảo tuân thủ giản đồ xung của module.

Timer/PWM: Sử dụng bộ Timer (TIM1/TIM2) cấu hình ở chế độ PWM Generation. Tần số và chu kỳ nhiệm vụ được tính toán để điều khiển Driver động cơ, cho phép thay đổi trơn tru tốc độ quay của động cơ DC thông qua việc điều chỉnh độ rộng xung.

GPIO Input: Cấu hình các chân đọc trạng thái nút nhấn (Pull-up/Pull-down) và chân nhận tín hiệu phản hồi sóng siêu âm từ cảm biến.

GPIO Output: Điều khiển các ngoại vi cơ bản như đèn báo trạng thái (LED), chân Trig từ cảm biến siêu âm, chân kích hoạt điều khiển động cơ.

1. Xây dựng lớp giao tiếp CAN

Để tăng tính linh hoạt, tái sử dụng mã nguồn và dễ dàng bảo trì, giao tiếp CAN không được viết trực tiếp trong chương trình chính mà được đóng gói thành một lớp riêng biệt. Lớp này thực hiện các chức năng cốt lõi:

* Trừu tượng hóa phần cứng: Cung cấp các hàm như CANSPI\_Initialize, CANSPI\_Transmit, CANSPI\_Receive. Lớp ứng dụng chỉ cần gọi các hàm này mà không cần quan tâm đến việc thao tác thanh ghi SPI hay quy trình bắt tay phức tạp với chip MCP2515.
* Đóng gói dữ liệu: Dữ liệu điều khiển (tốc độ, trạng thái đèn) được đóng gói vào cấu trúc dữ liệu uCAN\_MSG. Cấu trúc này bao gồm ID gói tin, độ dài dữ liệu và mảng dữ liệu, giúp chuẩn hóa định dạng bản tin giữa các Node.
* Cơ chế bộ lọc và mặt nạ: Tại hàm khởi tạo, các thanh ghi bộ lọc của MCP2515 được nạp giá trị để chỉ cho phép các gói tin có ID hợp lệ đi vào bộ đệm nhận, giúp giảm tải khối lượng xử lý ngắt cho CPU.
* Quản lý luồng dữ liệu: Xử lý việc kiểm tra trạng thái bộ đệm truyền/nhận, đảm bảo dữ liệu được gửi đi khi đường truyền rảnh và được giải mã chính xác khi có dữ liệu đến.

1. Logic điều khiển ứng dụng

Tại Node Motor: Vi điều khiển giải mã bản tin CAN nhận được để trích xuất giá trị tốc độ mong muốn, sau đó sử dụng bộ Timer để xuất xung PWM điều khiển mạch cầu H, đồng thời đọc phản hồi từ Encoder để giám sát vận tốc thực.

Tại Node Body: Phần mềm thực hiện quét liên tục trạng thái các tín hiệu đầu vào số (công tắc đèn, xi-nhan). Khi có sự thay đổi trạng thái, vi điều khiển đóng gói dữ liệu thành khung tin CAN và gửi đi, đảm bảo độ trễ tín hiệu ở mức thấp nhất.

### 3.2.2 Xây dựng Gateway

Gateway là nút trung tâm chịu trách nhiệm chuyển tiếp dữ liệu giữa mạng CAN Bus và khối xử lý trung tâm Raspberry Pi. Để đảm bảo tính ổn định và tránh hiện tượng nghẽn cổ chai dữ liệu, phần mềm cho ESP32 được thiết kế dựa trên hệ điều hành thời gian thực FreeRTOS.

1. Kiến trúc phần mềm đa nhiệm

Thay vì sử dụng vòng lặp vô hạn tuần tự truyền thống, chương trình được chia nhỏ thành các tác vụ độc lập, chạy song song giả lập nhờ bộ lập lịch của FreeRTOS. Điều này cho phép hệ thống ưu tiên xử lý các sự kiện quan trọng ngay lập tức mà không bị cản trở bởi các tác vụ chậm hơn (debug trên Serial).

1. Thiết kế các tác vụ

Task 1: Thu thập dữ liệu CAN (TaskCANReceive - Priority Cao nhất)

* Chức năng: Chịu trách nhiệm giao tiếp SPI với module MCP2515 để kiểm tra và đọc dữ liệu từ mạng CAN Bus.
* Đặc điểm: Được gán mức ưu tiên cao nhất để đảm bảo không bỏ sót bất kỳ gói tin nào từ hệ thống. Ngay khi có dữ liệu, Task này sẽ chiếm quyền CPU để xử lý, cập nhật trạng thái vào bộ nhớ chung và giải phóng CPU ngay sau đó.

Task 2: Xử lý logic hệ thống (TaskControl - Priority Trung bình)

* Chức năng: Thực hiện các logic điều khiển cục bộ tại Gateway như: tạo nhịp nháy cho đèn xi-nhan/Hazard, kiểm soát thời gian Timeout của các Node con để phát hiện mất kết nối.
* Đặc điểm: Chạy định kỳ với chu kỳ ngắn (10ms) để đảm bảo phản hồi thị giác mượt mà.

Task 3: Truyền thông UART (TaskSerial - Priority Thấp nhất)

* Chức năng: Định kỳ đọc dữ liệu tổng hợp từ bộ nhớ chung, đóng gói thành chuỗi ký tự và gửi lên Raspberry Pi qua cổng UART (Baudrate 115200).
* Đặc điểm: Do việc gửi dữ liệu qua UART tốn nhiều thời gian xử lý, tác vụ này được đặt mức ưu tiên thấp nhất để không làm gián đoạn luồng nhận tin CAN.

1. Cơ chế đồng bộ và bảo vệ dữ liệu

Do có nhiều tác vụ cùng truy cập vào một vùng dữ liệu chung, hệ thống sử dụng cơ chế khóa Mutex để đảm bảo toàn vẹn dữ liệu nhằm tránh tình trạng Race Condition, ví dụ: Task CAN đang ghi dở dữ liệu tốc độ thì Task Serial lại đọc dữ liệu đó để gửi đi, dẫn đến sai lệch thông tin. Để giải quyết vấn đề này, trước khi thực hiện ghi hoặc đọc, tác vụ phải lấy được "chìa khóa" (Semaphore Take). Nếu tài nguyên đang bận, tác vụ sẽ chuyển sang trạng thái chờ cho đến khi tài nguyên được giải phóng.

1. Quy trình xử lý dữ liệu

Thu thập: Task CAN nhận gói tin từ MCP2515, phân tích ID và cập nhật các trường tương ứng trong SystemState (ví dụ: stateLeft, stateRight, distance, speed).

Xử lý: Task Control đọc SystemState để điều khiển các chân GPIO.

Chuyển tiếp: Task Serial đọc SystemState, format thành chuỗi tin (ví dụ: "COS:ON| Gap:3") và gửi lên Raspberry Pi để hiển thị bảng đồng hồ.

### 3.2.3 Triển khai khối xử lý trung tâm

Phần mềm trên Raspberry Pi đóng vai trò là bộ não tính toán của hệ thống, chịu trách nhiệm xử lý các tác vụ nặng về thị giác máy tính và định tuyến dữ liệu. Chương trình được phát triển bằng ngôn ngữ Python, tận dụng các thư viện tối ưu hóa cho kiến trúc ARM như TensorFlow Lite Runtime và OpenCV.

1. Xây dựng giải thuật nhận diện biển báo

Để cân bằng giữa độ chính xác và tốc độ xử lý trên phần cứng nhúng, hệ thống không áp dụng mạng nơ-ron lên toàn bộ khung hình mà sử dụng phương pháp Hybrid Approach gồm hai giai đoạn:

Giai đoạn 1: Đề xuất vùng quan tâm:

* Sử dụng thư viện OpenCV để chuyển đổi không gian màu từ BGR sang HSV, giúp tách biệt màu sắc biển báo (đỏ/xanh) khỏi môi trường mà không bị ảnh hưởng nhiều bởi độ sáng.
* Áp dụng các bộ lọc hình thái học để loại bỏ nhiễu và tìm kiếm các đường viền.
* Lọc ứng viên dựa trên các đặc trưng hình học: Diện tích tối thiểu, tỷ lệ khung hình và độ tròn. Chỉ những vùng ảnh thỏa mãn điều kiện vật lý của biển báo mới được cắt và đưa sang giai đoạn sau.

Giai đoạn 2: Phân loại bằng AI:

* Các vùng ảnh tiềm năng được chuẩn hóa kích thước và đưa vào mô hình MobileNetV2 SSD đã được lượng tử hóa. Việc sử dụng mô hình lượng tử hóa giúp giảm kích thước mô hình và tăng tốc độ suy luận lên đáng kể so với mô hình gốc.
* Kết quả đầu ra chỉ được chấp nhận nếu độ tin cậy vượt ngưỡng 0.85.

1. Kỹ thuật ổn định tính hiệu

Để tránh hiện tượng kết quả nhận diện bị chập chờn do nhiễu camera hoặc điều kiện ánh sáng thay đổi, hệ thống áp dụng thuật toán xác nhận chuỗi. Một biển báo chỉ được xác nhận là hợp lệ khi và chỉ khi kết quả phân loại AI giữ nguyên trong 4 khung hình liên tiếp (CONSECUTIVE\_FRAMES\_REQ = 4). Điều này giúp loại bỏ các kết quả nhận diện sai thoáng qua và giúp giao diện hiển thị ổn định hơn.

1. Cơ chế cầu nối dữ liệu

Bên cạnh nhiệm vụ xử lý ảnh, chương trình Python còn hoạt động như một trung tâm định tuyến dữ liệu để liên kết các thành phần hệ thống. Khởi tạo cổng Serial (/dev/ttyUSB0) với tốc độ Baud 115200 bps để lắng nghe dữ liệu thô từ Node Gateway. Sử dụng cơ chế đọc không chặn hoặc kiểm tra bộ đệm để đảm bảo việc đọc dữ liệu xe không làm chậm luồng xử lý ảnh.

Toàn bộ dữ liệu sau xử lý (bao gồm ID biển báo từ AI và thông số xe từ UART) được đóng gói thành các bản tin chuỗi. Sử dụng UDP Socket gửi dữ liệu tới địa chỉ Localhost (127.0.0.1, Port 5005). Phương pháp này giúp tách biệt hoàn toàn luồng xử lý trung tâm khỏi luồng hiển thị đồ họa , tận dụng khả năng xử lý đa nhân của vi xử lý trên Raspberry Pi.

### 3.2.4 Phát triển giao diện

Dựa trên thiết kế giao diện người dùng đã đề xuất, quá trình hiện thực hóa phần mềm được triển khai trên nền tảng Qt Framework (PyQt5). Mã nguồn được tổ chức theo mô hình phân tách logic và hiển thị để tối ưu hóa hiệu năng trên thiết bị nhúng.

1. Tổ chức mã nguồn hiển thị

Để đảm bảo khả năng mở rộng và dễ dàng bảo trì, mã nguồn QML không viết tập trung mà được chia thành các Component riêng biệt, được quản lý bởi một file điều phối chính:

* File điều phối (main.qml): Đóng vai trò là khung xương sống, chịu trách nhiệm tải các tài nguyên hình ảnh và sắp xếp vị trí các module con.
* Các Component chức năng: Các thành phần giao diện phức tạp được tách thành các file riêng (SpeedGauge.qml cho đồng hồ tốc độ, WarningIcon.qml cho các cảnh báo, …). Cách tổ chức này giúp tái sử dụng mã nguồn và giảm tải bộ nhớ khi chỉ nạp những thành phần cần thiết.

1. Tích hợp Logic Backed

Kết nối giữa lớp xử lý dữ liệu và giao diện được thực hiện thông qua cơ chế Signals & Slots của Qt, đảm bảo cập nhật dữ liệu thời gian thực mà không gây xung đột luồng

* Khởi tạo Engine: Sử dụng QQmlApplicationEngine để nạp giao diện. Đối tượng Backend được nhúng trực tiếp vào ngữ cảnh QML thông qua thuộc tính định danh "myBackend".
* Cơ chế Binding dữ liệu: Tại phía QML, các thuộc tính hiển thị (như góc quay kim đồng hồ, trạng thái ẩn/hiện icon) được liên kết trực tiếp với tín hiệu dataReceived. Khi Python phát tín hiệu, giao diện tự động vẽ lại trạng thái mới mà không cần các hàm set/get thủ công.

1. Xử lý đa luồng

Một thách thức lớn khi triển khai trên Raspberry Pi là việc lắng nghe dữ liệu mạng có thể làm treo giao diện đồ họa. Giải pháp kỹ thuật được áp dụng như sau:

* Tách biệt luồng UDP: Một luồng riêng biệt được khởi tạo bằng thư viện threading để chạy tác vụ udp\_listener. Luồng này hoạt động song song và chịu trách nhiệm lắng nghe liên tục tại cổng 5005.
* Cơ chế cập nhật bất đồng bộ: Khi nhận được gói tin chuỗi từ hệ thống AI hoặc Gateway, luồng UDP sẽ giải mã và bắn tín hiệu sang luồng giao diện chính. Kiến trúc này giúp giao diện luôn duy trì độ mượt mà ngay cả khi lưu lượng dữ liệu đầu vào lớn.

## 3.3 Triển khai phần cứng

Sau khi hoàn tất thiết kế sơ đồ nguyên lý và lập trình phần mềm, quá trình thi công phần cứng được thực hiện nhằm chuyển đổi các bản vẽ thiết kế thành mô hình vật lý hoàn chỉnh. Quá trình này đòi hỏi sự chính xác trong việc gia công mạch in, hàn linh kiện và đấu nối hệ thống dây dẫn.

### 3.3.1 Quy trình thực hiện

Quá trình thi công phần cứng được tiến hành theo trình tự từ đơn vị nhỏ nhất đến tích hợp toàn hệ thống để giảm thiểu rủi ro hỏng hóc linh kiện.

Bước 1: Chuyển đổi bản vẽ và in ấn

Từ phần mềm Altium Designer, sơ đồ mạch in được xuất sang định dạng PDF với chế độ in đen trắng. Sử dụng máy in và giấy chuyên dụng để in bản. Mực in Laser có chứa bột than và nhựa, đóng vai trò là lớp bảo vệ trong quá trình ăn mòn.

Bước 2: Chuyển nhiệt và ăn mòn

Phíp đồng được cắt theo kích thước thiết kế, sau đó được đánh sạch bề mặt bằng giấy nhám mịn và dung dịch Aceton để loại bỏ oxy hóa, đảm bảo mực in bám dính tốt nhất.

Cố định giấy in lên bề mặt phíp đồng và sử dụng bàn là nhiệt để gia nhiệt. Dưới tác dụng của nhiệt độ cao và áp lực, lớp mực từ giấy sẽ nóng chảy và bám chặt vào bề mặt đồng.

Sau khi nguội, giấy in được bóc ra nhẹ nhàng, để lại các đường mạch đen sắc nét trên phíp đồng. Các vị trí đứt nét được tô lại thủ công bằng bút dạ dầu.

Phíp đồng được ngâm trong dung dịch muối sắt. Dung dịch sẽ ăn mòn lớp đồng trần không được mực che phủ, giữ lại các đường mạch điện. Quá trình được lắc đều để đẩy nhanh tốc độ phản ứng.

Bước 4: Gia công

Sử dụng máy khoan mạch cầm tay với các mũi khoan kích thước khác nhau tùy theo chân linh kiện:

Mũi 0.8mm: Cho các linh kiện thụ động (điện trở, tụ điện), header.

Mũi 1.0mm - 3.0mm: Cho các linh kiện công suất, cầu đấu dây và lỗ bắt vít cố định mạch. Đối với module MCP2515 và vi điều khiển, sử dụng các hàng rào cái để cắm linh kiện thay vì hàn chết, giúp dễ dàng thay thế khi có sự cố.

Sau khi khoan, mạch được chà nhẹ bằng giấy nhám mịn dưới vòi nước để lau sạch lớp mực in bảo vệ và làm phẳng các ba-via.

Sau khi làm sạch, mạch được kiểm tra thông mạch bằng đồng hồ vạn năng để đảm bảo không có đường mạch nào bị đứt hoặc chạm chập do quá trình ăn mòn.

Một lớp mỏng dung dịch nhựa thông lỏng được quét đều lên toàn bộ bề mặt mạch in. Lớp phủ này sau khi khô sẽ tạo thành màng bảo vệ ngăn chặn quá trình oxy hóa đồng, đồng thời đóng vai trò là chất trợ hàn giúp thiếc hàn loang đều, mối hàn bóng và đẹp hơn trong quá trình lắp ráp.

Bước 5: Hàn lắp linh kiện

Hệ thống sử dụng hoàn toàn linh kiện chân cắm để đảm bảo độ bền cơ học và dễ dàng thay thế. Quá trình hàn tuân thủ nguyên tắc "thấp trước - cao sau":

* Linh kiện thụ động thấp: Hàn các điện trở, tụ điện, nút nhấn.
* Đế cắm: Việc sử dụng đế cắm cho Vi điều khiển, module MCP2515 và ESP32 giúp tránh hỏng linh kiện do nhiệt độ mỏ hàn và thuận tiện cho việc tháo lắp, kiểm tra.
* Linh kiện cao/công suất: Tụ hóa, IC ổn áp và các trạm đấu dây.

Sau khi hàn hoàn tất, mạch được vệ sinh lại một lần nữa để loại bỏ các vết nhựa thông cháy, đảm bảo tính thẩm mỹ.

Bước 5: Kiểm tra nguội

Trước khi cấp nguồn, sử dụng đồng hồ vạn năng ở chế độ đo thông mạch để kiểm tra kỹ các đường nguồn (VCC - GND) nhằm đảm bảo không xảy ra ngắn mạch do thiếc hàn bị lem trong quá trình thi công.

### 3.3.2 Mô hình thực tế

Sau quá trình lắp ráp và đi dây, mô hình phần cứng được hoàn thiện với kết cấu vững chắc, gọn gàng và đáp ứng đầy đủ các yêu cầu thiết kế ban đầu. Hệ thống dây dẫn được bó gọn để đảm bảo tính thẩm mỹ và thuận tiện cho việc bảo trì, sửa chữa.

## 3.4 Kiểm thử và đánh giá kết quả

### 3.4.1 Kịch bản và môi trường kiểm thử

1. Môi trường kiểm thử

Điều kiện ánh sáng: Thực nghiệm được tiến hành trong điều kiện ánh sáng phòng thí nghiệm (ánh sáng đèn huỳnh quang ổn định) và điều kiện ánh sáng tự nhiên ngoài trời (ban ngày) để đánh giá khả năng thích nghi của thuật toán xử lý ảnh.

Sa hình: Sử dụng sa hình mô phỏng đường giao thông với các biển báo kích thước tiêu chuẩn thu nhỏ (tỷ lệ 1:10), bao gồm các biển: Dừng lại, giới hạn tốc độ.

1. Các kịch bản kiểm thử

Kịch bản 1 - Kiểm tra đường truyền CAN: Gửi liên tục gói tin từ Node Body đến Node Gateway và ngược lại để đo tỷ lệ mất gói và độ ổn định của mạng CAN Bus thủ công.

Kịch bản 2 - Kiểm tra độ trễ điều khiển: Đo thời gian từ khi người dùng thao tác trên bảng điều khiển (ví dụ: bật đèn) đến khi thiết bị chấp hành trên xe phản hồi.

Kịch bản 3 - Kiểm tra nhận diện và tự động lái: Xe di chuyển với tốc độ cố định, camera quét biển báo. Hệ thống phải nhận diện đúng loại biển và tự động điều chỉnh động cơ (ví dụ: gặp biển STOP phải dừng hẳn).

### 3.4.2 Kết quả thực nghiệm

1. Đánh giá hệ thống truyền thông

Kết quả đo đạc bằng bộ phân tích logic và quan sát log hệ thống cho thấy:

* Mạng CAN Bus: Hoạt động ổn định ở tốc độ Baud 250 kbps.
* Tỷ lệ lỗi đường truyền: Nhỏ hơn 1% (Rất thấp).
* Nhờ sử dụng dây xoắn đôi và điện trở đầu cuối 120Ω, tín hiệu tại các điểm nút rõ ràng, không bị nhiễu biên độ lớn dù mạch được gia công thủ công.
* Gateway đa nhiệm: Cơ chế Mutex và phân chia Priority hoạt động hiệu quả. Việc gửi dữ liệu lên UART (Task priority thấp) không làm gián đoạn việc nhận tin CAN (Task priority cao). Hệ thống không xảy ra hiện tượng treo khi chạy liên tục trong 1 giờ.

1. Đánh giá khả năng xử lý ảnh và AI

Tốc độ khung hình: Duy trì ổn định ở mức 25 đến 30 FPS. Đây là mức đủ mượt để điều khiển xe ở tốc độ thấp và trung bình.

Trong điều kiện ánh sáng phòng: Độ chính xác đạt trên 90% với ngưỡng Confidence 0.85.

Khoảng cách nhận diện hiệu quả: Từ 5cm đến 15cm trước đầu xe.

Cơ chế chống nhiễu: Việc áp dụng xác thực 4 khung hình liên tiếp giúp loại bỏ hoàn toàn hiện tượng nhận diện sai thoáng qua (False Positive), giúp xe vận hành trơn tru, không bị giật cục.

1. Đánh giá giao diện và độ trễ toàn hệ thống

Giao diện Instrument Cluster (Qt/QML): Hiển thị mượt mà, kim đồng hồ chuyển động tự nhiên nhờ cơ chế cập nhật bất đồng bộ (Signal & Slot). Không có hiện tượng đơ giao diện khi dữ liệu đổ về dồn dập.

Độ trễ tổng thể: Thời gian từ lúc Camera nhìn thấy biển báo → AI xử lý → Gửi lệnh qua UART/CAN → Động cơ phản hồi là khoảng 100 ms. Đây là mức độ trễ chấp nhận được đối với mô hình xe tự hành tốc độ thấp.

### 3.4.3 Đánh giá tổng thể hệ thống

Dựa trên các kết quả thực nghiệm, đề tài được đánh giá tổng quan thông qua các ưu điểm và hạn chế sau:

1. Ưu điểm

Kiến trúc phân tán hiện đại: Hệ thống áp dụng thành công mô hình mạng CAN Bus tiêu chuẩn công nghiệp ô tô, giúp giảm thiểu dây dẫn và dễ dàng mở rộng thêm các Node chức năng mà không cần thay đổi phần cứng trung tâm.

Ứng dụng công nghệ tiên tiến: Kết hợp hiệu quả giữa hệ điều hành thời gian thực (FreeRTOS) trên vi điều khiển và Trí tuệ nhân tạo trên máy tính nhúng, tạo nên một hệ thống lai mạnh mẽ.

Phần cứng tự chủ: Làm chủ hoàn toàn quy trình thiết kế và thi công mạch in thủ công, chứng minh khả năng hiện thực hóa sản phẩm từ lý thuyết. Thiết kế dạng Base-board giúp dễ dàng thay thế module khi hỏng hóc và giảm thiểu rủi ro lỗi hàn chip dán đối với quy trình làm mạch thủ công.

Giao diện chuyên nghiệp: Bảng đồng hồ được thiết kế tách biệt Backend và Frontend, có tính thẩm mỹ cao và khả năng phản hồi thời gian thực tốt.

1. Hạn chế

Ảnh hưởng bởi ánh sáng: Thuật toán xử lý ảnh còn chịu ảnh hưởng khi điều kiện ánh sáng thay đổi quá gắt (ngược sáng hoặc quá tối), do camera sử dụng là loại phổ thông, dải động chưa cao.

Công suất tính toán: Mặc dù đã tối ưu hóa model, nhưng Raspberry Pi vẫn bị nóng khi chạy tác vụ AI trong thời gian dài, có thể ảnh hưởng nhẹ đến hiệu năng nếu không có tản nhiệt tốt.

Nhiễu động cơ: Trong một số trường hợp động cơ khởi động nhanh, dòng khởi động lớn gây sụt áp nhẹ trên đường nguồn chung, dù đã có mạch ổn áp nhưng vẫn cần cải thiện khâu cách ly nguồn tốt hơn.

# KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

## I. Kết luận

Trong khuôn khổ đồ án tốt nghiệp, đề tài đã nghiên cứu, thiết kế và triển khai thành công một hệ thống nhúng tích hợp, mô phỏng các chức năng điều khiển và giám sát trên ô tô. Hệ thống được xây dựng theo kiến trúc phân tán, trong đó các node chức năng được kết nối với nhau thông qua giao thức CAN, đảm bảo khả năng trao đổi dữ liệu ổn định và có tính mở rộng cao.

Về mặt phần cứng, hệ thống đã được thiết kế và thi công hoàn chỉnh, bao gồm các node điều khiển sử dụng vi điều khiển STM32, node gateway sử dụng ESP32 và khối xử lý trung tâm Raspberry Pi. Các module phần cứng được bố trí hợp lý, đấu nối gọn gàng và hoạt động ổn định trong quá trình thử nghiệm.

Về mặt phần mềm, đề tài đã xây dựng được các chương trình điều khiển cho từng node, triển khai thành công cơ chế giao tiếp đa giao thức (CAN, UART, UDP) và phát triển giao diện giám sát trực quan bằng Qt/QML. Bên cạnh đó, khối xử lý trung tâm đã tích hợp thuật toán xử lý hình ảnh và nhận diện biển báo giao thông, cho phép hệ thống phản hồi và hiển thị thông tin theo thời gian thực.

Kết quả thực nghiệm cho thấy hệ thống hoạt động ổn định, các chức năng chính đáp ứng được mục tiêu đề ra ban đầu. Đề tài không chỉ giúp sinh viên củng cố kiến thức về hệ thống nhúng, mạng truyền thông công nghiệp và lập trình đa nền tảng, mà còn có ý nghĩa thực tiễn trong việc nghiên cứu các hệ thống hỗ trợ lái xe và mô phỏng điều khiển trên ô tô.

## II. Hướng phát triển

Mặc dù đã đạt được các mục tiêu đề ra, hệ thống vẫn còn một số hạn chế và có nhiều tiềm năng để tiếp tục phát triển trong tương lai. Một số hướng phát triển có thể xem xét bao gồm:

* Mở rộng chức năng nhận diện: Nâng cấp mô hình học sâu để nhận diện thêm các đối tượng giao thông như đèn tín hiệu, người đi bộ hoặc làn đường, nhằm tăng tính hoàn thiện cho hệ thống.
* Tối ưu hóa hiệu năng hệ thống: Cải thiện tốc độ xử lý hình ảnh và giảm độ trễ toàn hệ thống bằng cách tối ưu thuật toán, sử dụng phần cứng mạnh hơn hoặc tận dụng khả năng tăng tốc phần cứng.
* Hoàn thiện giao diện người dùng: Phát triển thêm các chức năng cấu hình, cảnh báo và ghi log dữ liệu, giúp giao diện bảng đồng hồ trở nên trực quan và thân thiện hơn.
* Nâng cấp kiến trúc truyền thông: Thử nghiệm các chuẩn giao tiếp mới như CAN FD hoặc Ethernet để tăng băng thông truyền dữ liệu và khả năng mở rộng hệ thống.
* Ứng dụng thực tế: Tích hợp hệ thống vào các mô hình xe tự hành quy mô nhỏ hoặc phục vụ cho mục đích nghiên cứu, giảng dạy và đào tạo về hệ thống nhúng trong lĩnh vực ô tô.

# TÀI LIỆU THAM KHẢO

**Tài liệu tham khảo tiếng Việt**

[1] Nguyễn Chí Thành, Trần Thế Sơn, Đặng Quang Hiển, Dương Hữu Ái (2023), “Thiết kế, thực nghiệm và đánh giá hoạt động của mạng điều khiển nội bộ CAN trong xe ô-tô”.

[2] Nguyễn Thanh Huyền (9/16/2021), “MobileNets – Mô hình gọn nhẹ cho mobile applications”, [MobileNets - Mô hình gọn nhẹ cho mobile applications](https://viblo.asia/p/cnn-architecture-series-1-mobilenets-mo-hinh-gon-nhe-cho-mobile-applications-1VgZvJV1ZAw), 27/11/2025.

[3] Toyota (23/5/2023), “Cruise Control là gì? Chức năng và cách sử dụng trên xe ô tô”, [Cruise Control là gì? Chức năng và cách sử dụng trên xe ô tô](https://www.toyota.com.vn/tin-tuc/thong-tin-bo-tro/cruise-control-la-gi-35728), 10/11/2025.

[4] Trần Quang Trung B (9/16/2019), “Tìm hiểu về SSD MultiBox Real-Time Object Detection”, [Tìm hiểu về SSD MultiBox Real-Time Object Detection](https://viblo.asia/p/tim-hieu-ve-ssd-multibox-real-time-object-detection-3P0lPEPG5ox), 28/11/2025.

[5] K2 (15/6/2019), “Giải thuật chống nhiễu cho nút nhấn”, [Giải thuật chống nhiễu cho nút nhấn](https://laubugs.wordpress.com/2019/06/15/giai-thuat-chong-nhieu-cho-nut-nhan/), 20/11/2025.

**Tài liệu tham khảo tiếng Anh**

[6] Andrew G.Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, Hartwig Adam (17/4/2017), “MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications”.

[7] Last Minute Engineers, “Create your own CAN network with MCP2515 Modules and Arduino”, [Create Your Own CAN Network With MCP2515 Modules and Arduino](https://lastminuteengineers.com/mcp2515-can-module-arduino-tutorial/), 2/11/2025.

[8] Lee Ji-Hoon (2018), “MCP2515 CAN Communication (SPI mode)”.

[9] Mamtaz Alam (2/2/2025), “ESP32 CAN Bus Tutorial | Interfacing MCP2515 CAN Module with ESP32”, [ESP32 CAN Bus Tutorial | Interfacing MCP2515 CAN Module with ESP32](https://how2electronics.com/esp32-can-bus-communication-with-mcp2515-module/), 2/11/2025.

[1] Mesidas, “CAN/CAN Bus là gì? Tổng quan về Control Area Network”, [CAN/CAN Bus là gì? Tổng quan về Control Area Network](https://mesidas.com/can-canbus/#top), 29/10/2025.