ĐẠI HỌC BÁCH KHOA HÀ NỘI

ĐỒ ÁN TỐT NGHIỆP

Lập trình cho thiết bị nhúng giám sát vị trí NaviTracker

Hà Thế Hiển
hien.ht215575@sis.hust.edu.vn

Ngành Kỹ thuật máy tính

Giảng viên hướng dẫn: PGS. TS. Lã Thế Vinh _____

Chữ kí GVHD

Khoa: Kỹ thuật máy tính

Trường: Công nghệ Thông tin và Truyền thông

HÀ NỘI, 06/2025

LỜI CẢM ƠN

Đầu tiên, em xin gửi lời cảm ơn chân thành tới thầy giáo hướng dẫn, PGS.TS. Lã Thế Vinh, người đã tận tình chỉ bảo, hỗ trợ và động viên em trong suốt quá trình thực hiện đồ án tốt nghiệp này. Em cũng xin cảm ơn gia đình, những người luôn ở bên, ủng hộ và tạo mọi điều kiện thuận lợi để em hoàn thành tốt nhất công việc của mình. Cuối cùng, cảm ơn bạn bè đã chia sẻ và hỗ trợ em vượt qua những khó khăn trong thời gian qua. Em trân trọng và biết ơn những nỗ lực, sự kiên trì và quyết tâm của chính bản thân, đã không ngừng cố gắng để hoàn thiện đồ án này với kết quả tốt nhất.

TÓM TẮT NỘI DUNG ĐỒ ÁN

Việc giám sát và quản lý vị trí các đối tượng, phương tiện ngày càng trở nên quan trọng trong nhiều lĩnh vực như vận tải, logistic, và an ninh. Tuy nhiên, các giải pháp truyền thống thường chỉ cung cấp thông tin vị trí tức thời, thiếu khả năng phân tích lịch sử di chuyển, cảnh báo khi vượt ra khỏi khu vực cho phép (geofence), và khó tích hợp các thông số hệ thống quan trọng như điện áp, trạng thái kết nối vào trong một nền tảng chung. Một số hướng tiếp cận phổ biến hiện nay bao gồm sử dụng thiết bị định vị GPS đơn giản, hệ thống quản lý dựa trên mạng di động GSM, hoặc các nền tảng IoT để quản lý và phân tích dữ liệu vị trí. Các giải pháp này dù hiệu quả nhưng thường bị giới hạn trong khả năng mở rộng, tích hợp, và quản lý dữ liệu tập trung.

Trong đồ án này, tôi lựa chọn tiếp cận theo hướng phát triển phần mềm cho thiết bị nhúng NaviTracker với nền tảng IoT ThingsBoard. Tôi lựa chọn giải pháp này do khả năng linh hoạt cao, dễ dàng tích hợp nhiều loại dữ liệu và hỗ trợ các tính năng quản lý nâng cao như geofence, theo dõi lịch sử di chuyển, và giám sát các thông số hệ thống.

Giải pháp của tôi bao gồm phát triển một phần mềm nhúng trên NaviTracker có khả năng thu thập liên tục tọa độ GPS, tính toán quãng đường di chuyển, và gửi dữ liệu vị trí cùng các thông số hệ thống (như trạng thái pin, chất lượng tín hiệu GPS) tới server ThingsBoard thông qua giao thức MQTT. Trên ThingsBoard, tôi xây dựng dashboard trực quan hiển thị thông tin vị trí hiện tại, tổng hợp quãng đường đã đi, lịch sử hành trình chi tiết, quản lý geofence linh hoạt và cảnh báo khi thiết bị ra vào các khu vực được định nghĩa trước.

Đóng góp chính của đồ án là xây dựng một hệ thống hoàn chỉnh và tích hợp sâu giữa thiết bị nhúng NaviTracker và nền tảng ThingsBoard, cung cấp đầy đủ các tính năng giám sát và quản lý vị trí. Kết quả đạt được sau cùng là hệ thống hoạt động ổn định, trực quan và đáp ứng tốt các yêu cầu thực tế về quản lý và giám sát đối tượng di chuyển, đã được thử nghiệm thực tế và cho kết quả chính xác, đáng tin cậy.

Sinh viên thực hiện

(Ký và ghi rõ họ tên)

ABSTRACT

Mục này khuyến khích sinh viên viết lại mục “Tóm tắt” đồ án tốt nghiệp ở trang trước bằng tiếng Anh. Phần này phải có đầy đủ các nội dung như trong phần tóm tắt bằng tiếng Việt. Sinh viên không nhất thiết phải trình bày mục này.

Nhưng nếu lựa chọn trình bày, sinh viên cần đảm bảo câu từ và ngữ pháp chuẩn tắc, nếu không sẽ có tác dụng ngược, gây phản cảm.

MỤC LỤC

DANH MỤC HÌNH VẼ

DANH MỤC BẢNG BIỂU

DANH MỤC THUẬT NGỮ VÀ TỪ VIẾT TẮT

Thuật ngữ	**Ý nghĩa**
API	Giao diện lập trình ứng dụng (Application Programming Interface)
EUD	Phát triển ứng dụng người dùng cuối(End-User Development)
GWT	Công cụ lập trình Javascript bằng Java của Google (Google Web Toolkit)
HTML	Ngôn ngữ đánh dấu siêu văn bản (HyperText Markup Language)
IaaS	Dịch vụ hạ tầng

CHƯƠNG 1. GIỚI THIỆU ĐỀ TÀI

1.1 Đặt vấn đề

Trong những năm gần đây, công nghệ định vị toàn cầu đã phát triển mạnh mẽ và ngày càng được ứng dụng rộng rãi trong nhiều lĩnh vực khác nhau. Các thiết bị định vị được tích hợp trong ô tô, thiết bị vận tải, thiết bị giám sát cá nhân, và hệ thống quản lý tài sản ngày càng phổ biến. Tuy nhiên, việc giám sát vị trí một cách chính xác và liên tục vẫn đang gặp nhiều thách thức như độ ổn định của tín hiệu vệ tinh, khả năng quản lý dữ liệu thu được, và sự hạn chế của các thiết bị giám sát truyền thống trong việc tương tác với nền tảng quản lý từ xa.

Thực tế hiện nay, nhu cầu giám sát và quản lý phương tiện vận tải, tài sản hoặc con người trong thời gian thực ngày càng tăng cao, đặc biệt là trong các ngành logistics, an toàn giao thông, quản lý đội xe, và cả trong giám sát người già, trẻ nhỏ. Việc xác định được chính xác vị trí và tình trạng hoạt động của đối tượng theo dõi sẽ giúp tối ưu hóa các hoạt động quản lý, nâng cao hiệu quả vận hành, giảm thiểu rủi ro và tiết kiệm chi phí vận hành cho doanh nghiệp cũng như cá nhân.

Nếu giải quyết được vấn đề giám sát và quản lý vị trí một cách liên tục và chính xác, các doanh nghiệp và cá nhân không chỉ được hưởng lợi từ việc cải thiện đáng kể hiệu quả quản lý và an toàn, mà công nghệ này còn có thể mở rộng ứng dụng sang các lĩnh vực khác như quản lý đô thị thông minh, giám sát môi trường, và hỗ trợ khẩn cấp trong các tình huống thiên tai. Từ đó, việc nghiên cứu và phát triển một hệ thống giám sát vị trí với độ chính xác và tin cậy cao trở thành vấn đề cấp thiết và có ý nghĩa thực tiễn quan trọng hiện nay.

1.2 Mục tiêu và phạm vi đề tài

Hiện nay, với sự phát triển mạnh mẽ của công nghệ định vị toàn cầu (GNSS) và các thiết bị IoT, việc giám sát hành trình, quản lý phương tiện, và đảm bảo an toàn giao thông ngày càng được chú trọng. Nhiều sản phẩm đã được phát triển nhằm đáp ứng nhu cầu theo dõi và quản lý vị trí theo thời gian thực, trong đó có các thiết bị định vị GPS tích hợp trên phương tiện, ứng dụng giám sát từ xa qua internet, và các hệ thống quản lý dữ liệu tập trung. Các thiết bị này thường được thiết kế để ghi lại tọa độ, tốc độ, và lộ trình di chuyển, đồng thời gửi dữ liệu lên máy chủ để quản lý và theo dõi.

Tuy nhiên, các hệ thống hiện tại còn tồn tại một số hạn chế như độ chính xác thấp trong điều kiện tín hiệu yếu, dễ bị tấn công giả mạo tín hiệu GNSS, hoặc thiếu khả năng tùy chỉnh phù hợp với các yêu cầu cụ thể của người dùng. Bên cạnh đó,

một số giải pháp giám sát vị trí hiện tại chưa tích hợp được tính năng phát hiện và cảnh báo khi có tín hiệu giả mạo, dẫn đến nguy cơ mất an toàn và thiếu tin cậy trong các ứng dụng quan trọng.

Dựa trên các phân tích trên, đồ án này tập trung vào việc phát triển một thiết bị giám sát hành trình tích hợp, có khả năng giám sát vị trí và phát hiện tín hiệu GNSS giả mạo. Thiết bị sẽ được tối ưu về hiệu năng xử lý và mức tiêu thụ năng lượng để đảm bảo phù hợp với các điều kiện hoạt động khắc nghiệt của IoT. Thiết bị không chỉ thu thập tọa độ GNSS, mà còn thực hiện phân tích tín hiệu để phát hiện những bất thường hoặc nguy cơ tấn công giả mạo, từ đó cung cấp thông tin giám sát chính xác và đáng tin cậy cho người sử dụng.

Mục tiêu chính của đề tài là xây dựng một hệ thống giám sát GNSS tối ưu, có khả năng hoạt động ổn định trong điều kiện khắc nghiệt, đồng thời đảm bảo tính an toàn và chính xác của dữ liệu định vị. Hệ thống sẽ được tích hợp lên nền tảng ThingsBoard để hiển thị vị trí hiện tại, lộ trình di chuyển, và các cảnh báo về tín hiệu giả mạo trong thời gian thực. Kết quả của đề tài không chỉ nâng cao tính tin cậy trong giám sát vị trí, mà còn góp phần vào việc bảo vệ an toàn cho các ứng dụng yêu cầu tính chính xác cao trong quản lý phương tiện và tài sản.

1.3 Định hướng giải pháp

Dựa trên nhiệm vụ được xác định trong phần trước, đồ án này lựa chọn hướng tiếp cận sử dụng các công nghệ GNSS, xử lý tín hiệu số và nền tảng IoT ThingsBoard để giải quyết bài toán giám sát hành trình và phát hiện tín hiệu GNSS giả mạo. Công nghệ GNSS được sử dụng để xác định vị trí thời gian thực, trong khi xử lý tín hiệu số sẽ giúp phân tích và phát hiện các tín hiệu bất thường. Nền tảng IoT ThingsBoard sẽ đảm nhiệm vai trò hiển thị thông tin vị trí, quản lý dữ liệu, và phát cảnh báo trực quan. Việc lựa chọn các công nghệ này dựa trên khả năng đáp ứng yêu cầu về độ chính xác, tính ổn định và khả năng mở rộng của hệ thống.

Giải pháp của đồ án tập trung vào việc thiết kế một thiết bị giám sát hành trình GNSS tích hợp, với khả năng nhận diện tín hiệu giả mạo. Thiết bị sẽ thu thập dữ liệu GNSS từ các cảm biến, xử lý tín hiệu để phát hiện các bất thường, và gửi dữ liệu vị trí cùng cảnh báo về hệ thống giám sát ThingsBoard. Người dùng có thể theo dõi lộ trình, kiểm tra tính xác thực của tín hiệu định vị và nhận cảnh báo ngay lập tức khi có dấu hiệu đáng ngờ.

Định hướng cụ thể của giải pháp bao gồm: xây dựng phần mềm cho thiết bị IoT giám sát hành trình GNSS có khả năng chống giả mạo tín hiệu và tích hợp hệ thống vào nền tảng ThingsBoard để hiển thị và quản lý dữ liệu theo thời gian thực. Kết quả đạt được là một hệ thống giám sát hành trình an toàn, đáng tin cậy, đáp ứng

nhu cầu quản lý vị trí và bảo vệ dữ liệu định vị trong các ứng dụng thực tế.

1.4 Bố cục đồ án

Phần còn lại của báo cáo đồ án tốt nghiệp này được tổ chức như sau:

Chương 2 trình bày về khảo sát và phân tích yêu cầu của hệ thống giám sát GNSS tích hợp. Nội dung chương tập trung vào việc thu thập thông tin từ các tài liệu nghiên cứu, các giải pháp hiện có, cũng như khảo sát nhu cầu thực tiễn của người dùng đối với việc giám sát hành trình và bảo vệ tín hiệu GNSS. Chương này cũng đưa ra phân tích các yêu cầu chức năng và phi chức năng của hệ thống, từ đó xây dựng các tiêu chí để đánh giá hiệu quả của thiết bị giám sát GNSS tích hợp.

Chương 3 giới thiệu về các công nghệ và công cụ được sử dụng trong quá trình phát triển hệ thống. Các công nghệ chính bao gồm GNSS cho định vị, xử lý tín hiệu số để phát hiện giả mạo, và ThingsBoard để quản lý và hiển thị dữ liệu. Ngoài ra, chương này cũng mô tả ngắn gọn về các thư viện và nền tảng hỗ trợ triển khai giải pháp như MQTT, xử lý tín hiệu DSP và các giao thức truyền nhận dữ liệu từ thiết bị IoT lên server.

Chương 4 tập trung vào thiết kế phần mềm cho thiết bị giám sát GNSS tích hợp. Chương này trình bày kiến trúc hệ thống, mô hình phân lớp của phần mềm và các mô-đun chức năng chính. Các phương pháp tối ưu hiệu năng và tiết kiệm năng lượng cũng được thảo luận trong chương này nhằm đảm bảo thiết bị có thể hoạt động ổn định trong điều kiện khắc nghiệt.

Chương 5 mô tả việc triển khai server để thu thập, xử lý và hiển thị dữ liệu từ thiết bị giám sát. Hệ thống server được xây dựng dựa trên nền tảng ThingsBoard với khả năng hiển thị vị trí theo thời gian thực, lưu trữ lịch sử di chuyển, và quản lý các cảnh báo. Các vấn đề liên quan đến bảo mật và độ tin cậy của hệ thống cũng được phân tích trong chương này.

Chương 6 đưa ra kết luận về kết quả đạt được trong quá trình nghiên cứu và phát triển hệ thống giám sát GNSS tích hợp. Chương này tổng hợp lại các đóng góp chính của đề tài, đồng thời phân tích những điểm mạnh và hạn chế của hệ thống. Bên cạnh đó, chương này cũng đề xuất các hướng phát triển trong tương lai, bao gồm việc nâng cao độ chính xác của thuật toán phát hiện giả mạo và tích hợp thêm các chức năng thông minh vào hệ thống.

CHƯƠNG 2. KHẢO SÁT VÀ PHÂN TÍCH YÊU CẦU

Trong chương này, tôi sẽ tập trung vào việc khảo sát và phân tích yêu cầu của đề tài. Đầu tiên, tôi tiến hành khảo sát các công nghệ, thiết bị và giải pháp liên quan, nhằm thu thập các thông tin cần thiết phục vụ cho việc phát triển hệ thống. Tiếp theo, dựa trên kết quả khảo sát, tôi sẽ phân tích các hướng giải quyết khả thi, từ đó đề xuất phương án thực hiện phù hợp nhất cho đề tài.

2.1 Khảo sát hiện trạng

Phần này sẽ trình bày tổng quan khảo sát về một số hạn chế của các hệ thống theo dõi hiện nay.

2.1.1 Hạn chế của các hệ thống hiện tại

Hiện nay, các hệ thống giám sát GNSS (Global Navigation Satellite System) được ứng dụng rộng rãi trong nhiều lĩnh vực như giám sát phương tiện, quản lý đội xe, và định vị cá nhân. Tuy nhiên, các hệ thống này vẫn còn tồn tại nhiều hạn chế, ảnh hưởng đến độ chính xác và tính ổn định trong quá trình vận hành.

Một trong những hạn chế lớn nhất của các hệ thống GNSS hiện tại là độ chính xác bị ảnh hưởng bởi nhiều yếu tố ngoại cảnh. Tín hiệu vệ tinh có thể bị nhiễu do các tác động môi trường như thời tiết xấu, địa hình phức tạp (đồi núi, hẻm sâu), hoặc các vùng đô thị với mật độ cao. Điều này dẫn đến sai số định vị, ảnh hưởng trực tiếp đến khả năng giám sát liên tục và đáng tin cậy của hệ thống.

Một vấn đề đáng lo ngại khác là khả năng bị tấn công giả mạo tín hiệu GNSS. Các tín hiệu giả mạo có thể được tạo ra từ các thiết bị phát sóng trái phép (spoofing), làm cho hệ thống giám sát nhận diện sai vị trí thực tế. Điều này gây ra những hậu quả nghiêm trọng trong các ứng dụng yêu cầu độ chính xác cao như giám sát an ninh, quản lý giao thông, hoặc theo dõi phương tiện công cộng.

Các hệ thống GNSS hiện tại thường khó tích hợp với các nền tảng IoT hiện đại do khác biệt về giao thức truyền thông và cách thức xử lý dữ liệu. Việc truyền dữ liệu GNSS lên các nền tảng IoT để giám sát và phân tích thời gian thực thường yêu cầu xử lý trung gian phức tạp, dẫn đến độ trễ trong việc hiển thị vị trí và trạng thái di chuyển.

Trong các hệ thống giám sát di động, việc duy trì giám sát GNSS liên tục tiêu thụ một lượng năng lượng đáng kể. Điều này làm giảm thời gian hoạt động của thiết bị, đặc biệt trong các ứng dụng di động hoặc khi sử dụng các thiết bị IoT với nguồn pin hạn chế.

Nhìn chung, các hệ thống giám sát GNSS hiện tại còn nhiều hạn chế về độ chính xác, khả năng chống giả mạo, tích hợp với hệ thống IoT, và hiệu năng năng lượng. Để khắc phục các vấn đề này, cần có những giải pháp cải tiến cả về mặt công nghệ định vị lẫn tích hợp hệ thống, từ đó đảm bảo hệ thống giám sát GNSS đạt hiệu quả cao và đáp ứng được các yêu cầu thực tiễn.

2.1.2 Các yêu cầu chức năng của hệ thống

Hệ thống giám sát GNSS và geofence cần đảm bảo khả năng giám sát liên tục để đáp ứng các yêu cầu theo dõi vị trí theo thời gian thực. Điều này đòi hỏi hệ thống phải liên tục thu thập và xử lý dữ liệu định vị từ thiết bị, đồng thời truyền tải lên nền tảng ThingsBoard một cách ổn định. Các thông tin như tọa độ, tốc độ di chuyển và trạng thái kết nối phải được cập nhật tức thời, tránh tình trạng mất tín hiệu hoặc gián đoạn, đảm bảo hệ thống có thể theo dõi và giám sát liên tục các đối tượng được quản lý.

Bên cạnh đó, tính năng geofence là một trong những yêu cầu quan trọng của hệ thống. Hệ thống cần cung cấp khả năng thiết lập các vùng giám sát địa lý trên bản đồ, xác định ranh giới khu vực mà đối tượng được phép di chuyển. Khi đối tượng ra khỏi vùng geofence hoặc di chuyển vào khu vực không được phép, hệ thống cần ngay lập tức phát hiện và gửi cảnh báo tới người quản lý. Việc thiết lập, thay đổi hoặc xóa vùng giám sát phải được thực hiện một cách linh hoạt, dễ dàng từ giao diện quản lý.

Ngoài ra, hệ thống cần tích hợp tính năng cảnh báo tự động khi đối tượng di chuyển vượt quá ranh giới geofence. Điều này giúp người dùng kịp thời phát hiện các tình huống bất thường hoặc vi phạm khu vực giám sát. Các cảnh báo cần được gửi dưới dạng thông báo tức thời qua các kênh như tin nhắn, email hoặc qua nền tảng ThingsBoard, giúp người quản lý nhanh chóng đưa ra các biện pháp xử lý.

Hệ thống cũng cần đảm bảo khả năng lưu trữ và quản lý lịch sử di chuyển của đối tượng trong một khoảng thời gian nhất định. Dữ liệu lịch sử cần được tổ chức và truy xuất dễ dàng, hỗ trợ việc phân tích hành trình, đánh giá hiệu quả giám sát, cũng như phục vụ công tác quản lý và báo cáo. Điều này đòi hỏi hệ thống phải có cơ chế lưu trữ dữ liệu an toàn, với dung lượng đủ lớn để đáp ứng nhu cầu sử dụng lâu dài.

2.1.3 Các yêu cầu phi chức năng của hệ thống

Để đảm bảo hiệu quả trong quá trình vận hành, hệ thống giám sát GNSS và geofence cần đáp ứng các yêu cầu phi chức năng về hiệu năng xử lý và tiêu thụ năng lượng. Hiệu năng của hệ thống phải đảm bảo khả năng thu thập và xử lý dữ liệu định vị theo thời gian thực mà không gây ra độ trễ lớn. Điều này đặc biệt quan

trọng trong các ứng dụng giám sát liên tục như theo dõi phương tiện hoặc giám sát an ninh. Đồng thời, hệ thống cần tối ưu hóa tiêu thụ năng lượng, nhất là đối với các thiết bị di động hoặc IoT sử dụng nguồn pin. Việc giảm thiểu tiêu thụ năng lượng giúp kéo dài thời gian hoạt động của thiết bị, hạn chế việc gián đoạn do hết pin.

Một trong những yêu cầu phi chức năng quan trọng khác là bảo mật dữ liệu và chống giả mạo tín hiệu GNSS. Hệ thống cần đảm bảo rằng các dữ liệu định vị được thu thập và truyền tải đều được mã hóa, bảo vệ trước các tấn công mạng như giả mạo vị trí (spoofing) hoặc can thiệp tín hiệu (jamming). Các cơ chế phát hiện tín hiệu bất thường cần được tích hợp, đảm bảo việc giám sát không bị sai lệch do các cuộc tấn công có chủ đích. Việc bảo mật này giúp đảm bảo tính toàn vẹn và độ tin cậy của dữ liệu thu thập được.

Hệ thống cũng cần có tính tương thích cao với các nền tảng IoT hiện đại, đặc biệt là trong việc tích hợp với ThingsBoard để quản lý thiết bị và giám sát từ xa. Việc sử dụng các giao thức truyền thông phổ biến như MQTT giúp tăng cường khả năng tương tác giữa các thiết bị và nền tảng. Điều này tạo điều kiện thuận lợi cho việc triển khai trên nhiều môi trường khác nhau mà không gặp phải các vấn đề tương thích.

Cuối cùng, tính ổn định và độ tin cậy là yếu tố quan trọng trong mọi hệ thống giám sát. Hệ thống cần hoạt động liên tục, ổn định trong thời gian dài mà không gặp phải sự cố hoặc mất tín hiệu. Việc giám sát và tự động khôi phục trong trường hợp lỗi giúp giảm thiểu tác động tiêu cực khi hệ thống bị gián đoạn. Các phương thức kiểm tra, bảo trì và nâng cấp định kỳ cũng cần được thiết lập rõ ràng, giúp duy trì hiệu năng ổn định trong suốt vòng đời của hệ thống.

2.2 Phân tích hướng giải quyết

Sau khi đã khảo sát và tìm hiểu các hệ thống giám sát GNSS hiện tại cũng như các yêu cầu cần thiết, chương này sẽ tập trung vào việc phân tích các hướng giải quyết nhằm xây dựng một hệ thống giám sát GNSS và geofence hiệu quả, đáp ứng các yêu cầu kỹ thuật đã đề ra.

Phần này sẽ trình bày mô hình hệ thống với các thành phần chính, từ đó xác định vai trò và chức năng của từng thành phần trong quá trình giám sát. Tiếp theo, việc lựa chọn công nghệ và nền tảng sẽ được phân tích kỹ lưỡng để đảm bảo hệ thống đạt được hiệu năng cao, tối ưu hóa năng lượng, và có khả năng chống giả mạo tín hiệu GNSS.

Bên cạnh đó, chương cũng sẽ đánh giá tính khả thi của các giải pháp đề xuất, dựa trên các tiêu chí như độ chính xác, độ tin cậy, khả năng mở rộng và tích hợp

với các nền tảng IoT. Kết quả của quá trình phân tích này sẽ làm cơ sở để tiến hành thiết kế và triển khai hệ thống trong các bước tiếp theo.

2.2.1 Mô hình hệ thống

Để xây dựng một hệ thống giám sát hiệu quả, đáp ứng các yêu cầu về giám sát liên tục, và quản lý vị trí theo thời gian thực, mô hình hệ thống cần được thiết kế với cấu trúc rõ ràng, tích hợp các thành phần chính một cách hợp lý để đảm bảo khả năng thu thập, xử lý và truyền tải dữ liệu định vị.

Hệ thống giám sát GNSS và geofence bao gồm ba thành phần chính: thiết bị giám sát, hệ thống trung tâm xử lý dữ liệu và nền tảng hiển thị thông tin. Thiết bị giám sát được lắp đặt trên đối tượng cần theo dõi, có chức năng thu thập tín hiệu vệ tinh để xác định vị trí hiện tại. Thiết bị này cần tích hợp các module GNSS hiện đại, đồng thời có kết nối mạng để truyền dữ liệu về trung tâm.

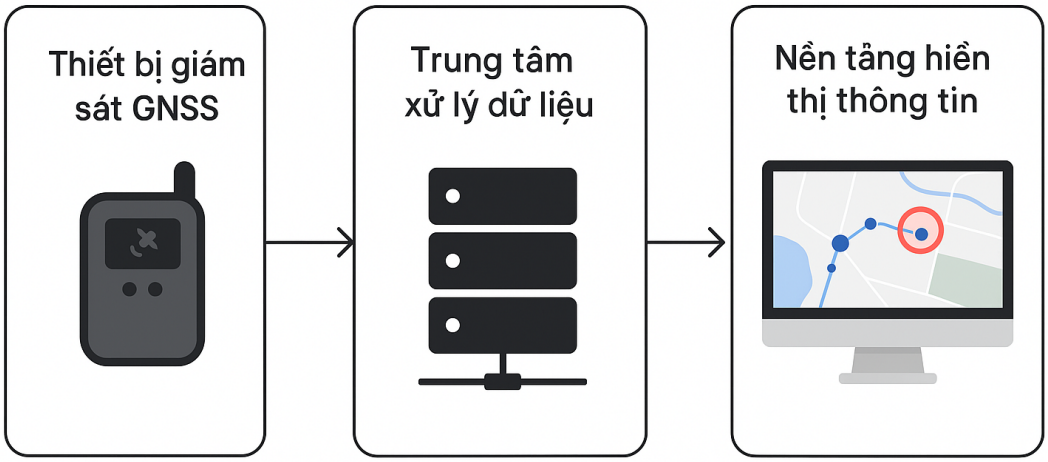
Hệ thống trung tâm xử lý dữ liệu đóng vai trò quan trọng trong việc thu thập, phân tích và lưu trữ dữ liệu định vị. Dữ liệu từ thiết bị GNSS được truyền về trung tâm thông qua các giao thức truyền thông như MQTT hoặc HTTP. Tại đây, dữ liệu được xử lý để phát hiện các sự kiện bất thường như mất tín hiệu hoặc di chuyển ra ngoài vùng geofence. Hệ thống xử lý cần có khả năng phân tích dữ liệu nhanh chóng, đảm bảo độ trễ tối thiểu khi xử lý tín hiệu định vị.

Nền tảng hiển thị thông tin đóng vai trò trực quan hóa dữ liệu, giúp người dùng dễ dàng theo dõi và giám sát đối tượng. Giao diện hiển thị cần tích hợp bản đồ số, thể hiện vị trí hiện tại của đối tượng theo thời gian thực. Ngoài ra, các chức năng cảnh báo khi vượt ranh giới geofence hoặc mất tín hiệu cũng được tích hợp để đảm bảo việc giám sát được diễn ra liên tục và ổn định.

Bên cạnh đó, mô hình hệ thống cũng cần thiết kế các cơ chế lưu trữ dữ liệu để đảm bảo khả năng truy vết và phân tích lịch sử di chuyển. Các dữ liệu định vị được lưu trữ dưới dạng cơ sở dữ liệu thời gian thực trên nền tảng ThingsBoard, hỗ trợ việc truy xuất và phân tích trong các trường hợp cần thiết.

Nhằm đảm bảo tính ổn định và độ tin cậy, mô hình hệ thống cũng cần thiết kế các cơ chế giám sát lỗi và tự động khôi phục khi gặp sự cố. Các thông số như chất lượng tín hiệu, độ trễ truyền tải và dung lượng lưu trữ cần được giám sát thường xuyên để kịp thời phát hiện và khắc phục các sự cố phát sinh.

Tóm lại, mô hình hệ thống giám sát GNSS và geofence được thiết kế với cấu trúc gồm ba thành phần chính: thiết bị giám sát, trung tâm xử lý dữ liệu và nền tảng hiển thị. Các thành phần này liên kết chặt chẽ với nhau, đảm bảo hệ thống hoạt động ổn định, đáp ứng các yêu cầu về giám sát liên tục và bảo mật thông tin.



Hình 2.1: Sơ đồ hệ thống

2.2.2 Lựa chọn công nghệ và nền tảng

Việc lựa chọn công nghệ và nền tảng phù hợp là yếu tố quan trọng để đảm bảo hệ thống giám sát và geofence hoạt động ổn định, hiệu quả và đáp ứng các yêu cầu kỹ thuật đã đề ra. Trong hệ thống này, các công nghệ được sử dụng sẽ bao gồm công nghệ định vị GNSS, nền tảng IoT để thu thập và quản lý dữ liệu, cũng như các giao thức truyền thông để kết nối các thành phần.

Công nghệ định vị GNSS đóng vai trò cốt lõi trong việc xác định vị trí của đối tượng. Các hệ thống định vị phổ biến hiện nay như GPS, GLONASS, Galileo và BeiDou đều có những ưu điểm riêng về độ chính xác, độ phủ sóng và khả năng hoạt động trong các điều kiện địa lý khác nhau. Dựa trên điều kiện triển khai và yêu cầu giám sát, hệ thống của tôi sử dụng công nghệ GPS vì độ phổ biến cao, khả năng định vị tốt và tương thích với nhiều thiết bị IoT hiện có.

Để đảm bảo khả năng giám sát và quản lý thiết bị từ xa, nền tảng IoT được lựa chọn là ThingsBoard. Đây là nền tảng mã nguồn mở, cung cấp nhiều tính năng mạnh mẽ như quản lý thiết bị, thu thập dữ liệu, giám sát trực quan và phát hiện sự kiện. ThingsBoard hỗ trợ giao thức truyền thông MQTT và HTTP, giúp tích hợp dễ dàng với các thiết bị IoT. Việc lựa chọn nền tảng này cũng giúp giảm chi phí phát triển và đảm bảo khả năng mở rộng trong tương lai.

Trong hệ thống giám sát, việc truyền tải dữ liệu vị trí từ thiết bị đến trung tâm

xử lý là một bước quan trọng. Để đảm bảo tính ổn định và độ trễ thấp, giao thức MQTT được sử dụng làm phương tiện truyền thông chính. MQTT là giao thức nhẹ, phù hợp cho các ứng dụng IoT có băng thông hạn chế và yêu cầu truyền tải dữ liệu liên tục. Nhờ cơ chế Publish/Subscribe, hệ thống có thể gửi và nhận thông tin một cách nhanh chóng, đảm bảo độ tin cậy trong giám sát thời gian thực.

Ngoài ra, để đảm bảo tính bảo mật trong việc truyền tải dữ liệu, hệ thống sử dụng mã hóa TLS (Transport Layer Security) để bảo vệ các gói tin trên nền tảng MQTT. Việc mã hóa này giúp đảm bảo tính toàn vẹn của dữ liệu, tránh nguy cơ bị giả mạo hoặc tấn công từ bên ngoài. Đồng thời, cơ chế xác thực hai lớp (Two-Factor Authentication) cũng được tích hợp để đảm bảo chỉ những thiết bị đã đăng ký mới có thể kết nối với hệ thống.

Đối với việc lưu trữ và xử lý dữ liệu, cơ sở dữ liệu thời gian thực TimescaleDB được lựa chọn do khả năng xử lý lượng lớn dữ liệu định vị với tốc độ cao. TimescaleDB tương thích với PostgreSQL, cung cấp khả năng phân tích dữ liệu lịch sử và giám sát hiệu năng. Điều này rất quan trọng trong việc lưu trữ lộ trình di chuyển và phát hiện các xu hướng bất thường.

Tóm lại, hệ thống giám sát GNSS và geofence được xây dựng dựa trên sự kết hợp của công nghệ GPS, nền tảng ThingsBoard và giao thức truyền thông MQTT. Bằng cách sử dụng các công nghệ hiện đại và nền tảng mạnh mẽ, hệ thống đảm bảo khả năng giám sát liên tục, bảo mật cao và dễ dàng mở rộng trong các ứng dụng thực tế.

2.2.3 Đánh giá tính khả thi

Việc đánh giá tính khả thi của hệ thống giám sát là một bước quan trọng nhằm đảm bảo hệ thống có thể đáp ứng các yêu cầu kỹ thuật cũng như phù hợp với thực tiễn triển khai. Trong quá trình đánh giá, các yếu tố như độ chính xác định vị, độ ổn định của hệ thống, khả năng mở rộng và tính bảo mật sẽ được xem xét một cách toàn diện.

Độ chính xác là yếu tố cốt lõi của mọi hệ thống giám sát GNSS. Với việc sử dụng công nghệ GPS, độ chính xác của hệ thống phụ thuộc vào nhiều yếu tố như điều kiện thời tiết, vị trí địa lý và môi trường xung quanh. Qua quá trình thử nghiệm thực tế, thiết bị GNSS cho thấy khả năng định vị với độ sai lệch từ 5 đến 10 mét trong điều kiện thông thường. Trong các môi trường đô thị dày đặc hoặc địa hình phức tạp, độ chính xác có thể giảm xuống đáng kể. Để khắc phục vấn đề này, hệ thống sử dụng các thuật toán lọc nhiễu và bù sai số như Kalman Filter, giúp cải thiện độ tin cậy của vị trí định vị.

Độ ổn định của hệ thống được đánh giá thông qua khả năng duy trì kết nối liên tục giữa thiết bị giám sát GNSS và nền tảng ThingsBoard. Các thử nghiệm cho thấy việc sử dụng giao thức MQTT giúp duy trì kết nối ổn định với độ trễ trung bình dưới 1 giây khi truyền tải dữ liệu định vị. Trong các trường hợp mất kết nối tạm thời, hệ thống tự động thực hiện cơ chế tái kết nối mà không làm gián đoạn quá trình giám sát. Điều này đảm bảo dữ liệu luôn được cập nhật kịp thời, đặc biệt quan trọng trong các tình huống yêu cầu giám sát liên tục.

Một trong những ưu điểm của hệ thống là khả năng mở rộng dễ dàng khi cần giám sát nhiều đối tượng cùng lúc. Với việc sử dụng nền tảng ThingsBoard, hệ thống có thể quản lý hàng trăm thiết bị mà không làm giảm hiệu năng. Cơ chế phân phối dữ liệu theo chủ đề (topic) của MQTT giúp việc xử lý dữ liệu từ nhiều nguồn diễn ra đồng thời mà không gây xung đột. Bên cạnh đó, cơ sở dữ liệu TimescaleDB hỗ trợ lưu trữ lượng lớn dữ liệu định vị trong thời gian dài, giúp việc phân tích lịch sử di chuyển trở nên hiệu quả.

Tính bảo mật của hệ thống được đảm bảo thông qua việc áp dụng các cơ chế mã hóa và xác thực chặt chẽ. Dữ liệu định vị từ thiết bị đến nền tảng ThingsBoard được mã hóa bằng TLS, ngăn chặn việc đánh cắp thông tin trong quá trình truyền tải. Ngoài ra, việc sử dụng xác thực hai lớp (2FA) giúp đảm bảo rằng chỉ những thiết bị được đăng ký trước mới có quyền truy cập vào hệ thống. Điều này giúp hạn chế nguy cơ bị xâm nhập hoặc giả mạo từ bên ngoài.

Thông qua các thử nghiệm ban đầu, hệ thống cho thấy khả năng giám sát liên tục với độ chính xác cao, đặc biệt là trong môi trường mở. Khả năng mở rộng và tính bảo mật được đảm bảo nhờ lựa chọn công nghệ và nền tảng phù hợp. Tuy nhiên, một số thách thức như độ trễ khi tín hiệu yếu hoặc mất kết nối vẫn cần được cải thiện.

Nhìn chung, hệ thống giám sát được thiết kế với sự chú trọng đến độ chính xác, độ ổn định, khả năng mở rộng và bảo mật. Các yếu tố này đã được thử nghiệm và cho kết quả khả quan trong điều kiện thực tế. Việc lựa chọn công nghệ hiện đại, nền tảng ThingsBoard và giao thức MQTT giúp hệ thống đạt được hiệu năng mong muốn, đồng thời đảm bảo tính bảo mật cao trong quá trình triển khai. Các kết quả đánh giá tính khả thi sẽ là nền tảng để tiến hành các bước thiết kế chi tiết và triển khai hệ thống trong thực tế.

CHƯƠNG 3. CÔNG NGHỆ SỬ DỤNG

Chương này trình bày các công nghệ, nền tảng và giao thức được sử dụng trong quá trình xây dựng hệ thống giám sát vị trí cho thiết bị nhúng NaviTracker. Việc lựa chọn công nghệ được thực hiện dựa trên yêu cầu chức năng và phi chức năng đã phân tích ở Chương 2, đảm bảo khả năng thu thập dữ liệu định vị GNSS, truyền thông ổn định tới nền tảng giám sát từ xa, và hiển thị thông tin một cách trực quan, linh hoạt.

Cụ thể, chương sẽ giới thiệu về nền tảng phần cứng của thiết bị NaviTracker, các giao thức truyền thông được tích hợp để đảm bảo trao đổi dữ liệu hiệu quả, và nền tảng ThingsBoard được sử dụng làm hệ thống hiển thị thông tin. Mỗi công nghệ được phân tích về nguyên lý hoạt động, lý do lựa chọn và vai trò của nó trong hệ thống tổng thể. Đồng thời, chương cũng sẽ so sánh một số lựa chọn thay thế có thể, qua đó làm rõ sự phù hợp của các công nghệ được sử dụng trong đồ án.

3.1 Thiết bị NaviTracker

Thiết bị NaviTracker là một nền tảng phần cứng nhúng được thiết kế chuyên biệt cho mục đích giám sát vị trí trong thời gian thực. Thiết bị tích hợp nhiều thành phần chính bao gồm: mô-đun vi điều khiển trung tâm SC20, mô-đun GNSS để thu thập dữ liệu định vị toàn cầu, mô-đun truyền thông không dây để truyền dữ liệu tới hệ thống giám sát ThingsBoard, cùng bộ nhớ trong để lưu tạm thời dữ liệu khi mất kết nối mạng. Nhờ sự tích hợp này, thiết bị có khả năng hoạt động độc lập, ghi nhận và truyền dữ liệu định vị một cách ổn định và liên tục.

Vi điều khiển SC20 đóng vai trò như một bộ xử lý trung tâm của hệ thống. Nó chịu trách nhiệm thu nhận dữ liệu GNSS từ mô-đun định vị thông qua giao tiếp UART, sau đó xử lý các bản tin theo định dạng chuẩn như NMEA hoặc UBX. Sau khi xử lý, SC20 trích xuất các thông tin cần thiết như tọa độ hiện tại, tốc độ di chuyển, hướng, và thời gian. Các dữ liệu này được SC20 định dạng lại và gửi lên máy chủ ThingsBoard bằng giao thức MQTT để hiển thị trên giao diện người dùng. Ngoài ra, SC20 cũng có khả năng thực hiện các tính toán cục bộ như xác định quãng đường di chuyển, phát hiện trạng thái đứng yên, kiểm tra vùng địa lý (geofence), và đánh giá tính hợp lệ của dữ liệu GNSS.

Một đặc điểm nổi bật trong thiết kế của NaviTracker là sự bổ sung của vi điều khiển phụ ATmega8 nhằm tối ưu hóa mức tiêu thụ năng lượng của toàn hệ thống. ATmega8 hoạt động như một bộ quản lý năng lượng thông minh, có nhiệm vụ điều khiển việc bật và tắt SC20 dựa trên trạng thái phát hiện chuyển động. Khi cảm

biến phát hiện có chuyển động ở đầu vào (kết nối tại chân PD3), ATmega8 sẽ kích hoạt SC20 bằng cách điều khiển chân PWRKEY trong một chu kỳ nhất định để bật nguồn SC20. Đồng thời, một đèn LED trạng thái cũng được bật để báo hiệu thiết bị đang hoạt động. Sau một khoảng thời gian nhất định nếu không còn chuyển động nào được phát hiện, ATmega8 sẽ đưa SC20 trở về trạng thái tắt nhằm tiết kiệm điện năng, đồng thời hệ thống được đưa vào chế độ ngủ sâu bằng chế độ sleep.

Kiến trúc này mang lại nhiều lợi ích thực tiễn. Thứ nhất, việc sử dụng ATmega8 giúp hạn chế đáng kể mức tiêu thụ năng lượng trong những thời điểm thiết bị không cần hoạt động liên tục, qua đó kéo dài thời gian sử dụng pin. Thứ hai, SC20 chỉ được kích hoạt khi có nhu cầu thu thập và truyền dữ liệu, nhờ đó duy trì được độ ổn định và tránh hao phí tài nguyên hệ thống. Cơ chế bật/tắt thông minh này phù hợp với các ứng dụng ngoài thực địa, nơi nguồn năng lượng thường bị giới hạn và thiết bị cần hoạt động bền bỉ trong thời gian dài.

So với các lựa chọn thay thế như sử dụng các bo mạch phát triển đa năng (Arduino, Raspberry Pi), giải pháp sử dụng NaviTracker thể hiện rõ ưu điểm về độ ổn định, tính gọn nhẹ, khả năng tích hợp truyền thông di động sẵn có, và đặc biệt là hiệu suất tiêu thụ năng lượng vượt trội. Điều này cho phép thiết bị dễ dàng triển khai trong các hệ thống IoT phân tán, nơi yêu cầu khắt khe về độ tin cậy, khả năng tự chủ và thời gian vận hành lâu dài.

Việc tích hợp hài hòa giữa SC20 và ATmega8 giúp đảm bảo tính liên tục trong thu thập và truyền tải dữ liệu định vị ngay cả khi thiết bị không được giám sát thường xuyên. Cấu trúc phần cứng này là nền tảng vững chắc cho các chức năng giám sát hành trình, xác định vùng địa lý, và phân tích di chuyển được trình bày trong các chương sau.

3.2 Ngôn ngữ và môi trường phát triển

Ứng dụng được phát triển cho thiết bị NaviTracker sử dụng hệ điều hành Android, với ngôn ngữ lập trình chính là Java. Đây là một lựa chọn phổ biến trong phát triển phần mềm di động và thiết bị nhúng chạy Android, nhờ vào hệ sinh thái phong phú, tính ổn định cao và khả năng truy cập trực tiếp các API phần cứng thông qua Android SDK.

Việc lựa chọn Android làm nền tảng phát triển xuất phát từ yêu cầu về khả năng xử lý linh hoạt, dễ dàng tương tác với mô-đun GNSS qua giao tiếp UART, và khả năng tích hợp các thư viện hỗ trợ giao thức MQTT để truyền dữ liệu lên hệ thống giám sát ThingsBoard. Môi trường phát triển sử dụng là Android Studio – một công cụ tích hợp hiện đại do Google phát triển, hỗ trợ đầy đủ cho việc lập trình, biên dịch, mô phỏng và gỡ lỗi ứng dụng Android.

So với các ngôn ngữ khác như C/C++ vốn thường dùng trong các hệ thống vi điều khiển truyền thống, Java trên Android mang lại lợi thế rõ rệt về tính mô-đun, khả năng mở rộng và dễ bảo trì mã nguồn. Đồng thời, hệ điều hành Android cho phép triển khai đồng thời nhiều tác vụ, như thu thập dữ liệu GNSS, lưu trữ cục bộ và truyền tải qua mạng, một cách hiệu quả nhờ vào cơ chế quản lý tiến trình và luồng xử lý riêng biệt.

Việc sử dụng Java và Android giúp rút ngắn thời gian phát triển, đồng thời đáp ứng linh hoạt các yêu cầu được phân tích trong Chương 2, đặc biệt là khả năng truyền dữ liệu định vị lên server và tương tác theo thời gian thực với các thành phần của hệ thống giám sát từ xa.

Song song với phần mềm Android, một chương trình nhúng chạy trên vi điều khiển ATmega8 cũng được phát triển nhằm quản lý nguồn điện cho thiết bị SC20. Chương trình này được viết bằng ngôn ngữ lập trình C, sử dụng bộ thư viện chuẩn AVR-GCC. Đây là trình biên dịch mã nguồn mở phổ biến cho dòng vi điều khiển AVR, hỗ trợ đầy đủ các chức năng truy cập thanh ghi, điều khiển ngắt ngoài, chế độ ngủ và thời gian trễ chính xác.

Việc kết hợp Java (trên Android) và C (trên ATmega8) tạo thành một hệ thống nhúng đa tầng, trong đó mỗi thành phần đảm nhận nhiệm vụ riêng biệt nhưng phối hợp chặt chẽ. Trong khi phần mềm Android xử lý các tác vụ cao cấp như phân tích GNSS, truyền dữ liệu MQTT và phản hồi theo thời gian thực, thì chương trình C trên ATmega8 đảm nhận việc tiết kiệm năng lượng bằng cách điều khiển chu kỳ hoạt động của thiết bị một cách thông minh.

Cách tiếp cận này mang lại sự cân bằng giữa tính linh hoạt của hệ điều hành Android và hiệu quả năng lượng của vi điều khiển nhúng, đảm bảo hệ thống hoạt động bền bỉ trong môi trường thực tế với yêu cầu cao về độ tin cậy và thời gian vận hành liên tục.

3.3 Giao thức truyền thông MQTT

MQTT (Message Queuing Telemetry Transport) là một giao thức truyền thông dạng publish/subscribe, được thiết kế đặc biệt cho các hệ thống nhúng và thiết bị IoT có tài nguyên giới hạn. Với đặc tính nhẹ, độ trễ thấp và cơ chế giữ kết nối ổn định, MQTT phù hợp để truyền dữ liệu trong thời gian thực, đặc biệt trong các môi trường mạng không ổn định hoặc có băng thông hạn chế.

Giao thức này hoạt động trên nền TCP/IP và cho phép các thiết bị gửi (publish) bản tin đến các chủ đề (topic) cụ thể, trong khi các bên nhận (subscriber) chỉ cần đăng ký nhận dữ liệu từ những chủ đề quan tâm. Cấu trúc này giúp hệ thống linh

hoạt, dễ mở rộng và giảm đáng kể độ phức tạp khi tích hợp nhiều thiết bị trong cùng một mạng lưới truyền thông.

Một trong những lý do chính để lựa chọn MQTT là khả năng duy trì kết nối liên tục thông qua cơ chế keep-alive, giúp đảm bảo truyền dữ liệu ổn định trong thời gian dài mà không cần thiết lập lại kết nối cho mỗi phiên làm việc. Trong môi trường mạng không ổn định, MQTT có thể tự động phát hiện mất kết nối và tiến hành kết nối lại, đồng thời hỗ trợ gửi lại bản tin nếu quá trình truyền bị gián đoạn. Những đặc điểm này giúp nâng cao độ tin cậy trong quá trình truyền dữ liệu thời gian thực giữa thiết bị và hệ thống trung tâm.

So với các giao thức khác thường được sử dụng trong hệ thống nhúng, MQTT thể hiện rõ ưu thế. Cụ thể: HTTP tuy phổ biến nhưng yêu cầu thiết lập kết nối lại cho mỗi lần truyền và hoạt động theo mô hình client-server nên gây tiêu tốn tài nguyên, không phù hợp với thiết bị chạy nền hoặc yêu cầu tiêu thụ năng lượng thấp.

CoAP là một lựa chọn nhẹ hơn khi sử dụng nền UDP, nhưng lại thiếu các cơ chế kiểm soát chất lượng dịch vụ và dễ gặp vấn đề mất gói trong mạng không ổn định.

WebSocket cung cấp giao tiếp hai chiều theo thời gian thực, tuy nhiên không định nghĩa rõ cấu trúc bản tin, thiếu cơ chế xử lý lỗi, kiểm soát luồng, và không hỗ trợ các tính năng đặc thù như mức QoS hay bản tin cuối cùng (last will) như MQTT.

Với khả năng vận hành hiệu quả trên cả nền tảng hệ điều hành đầy đủ như Android lẫn các hệ thống nhúng tối giản, MQTT phù hợp để triển khai trong các ứng dụng yêu cầu truyền dữ liệu định kỳ, giám sát liên tục và xử lý sự kiện trong thời gian thực. Thư viện mã nguồn mở như Eclipse Paho hoặc Mosquitto giúp việc tích hợp MQTT trở nên dễ dàng trong các môi trường phát triển hiện đại, cho phép ứng dụng nhanh chóng thiết lập kết nối, gửi và nhận bản tin với độ trễ thấp và độ tin cậy cao.

Tổng kết lại, MQTT là lựa chọn tối ưu cho hệ thống truyền thông của thiết bị nhúng giám sát vị trí, nhờ sự cân bằng giữa hiệu suất truyền tải, độ ổn định, và mức tiêu thụ tài nguyên thấp. Việc lựa chọn giao thức này là nền tảng quan trọng đảm bảo hệ thống hoạt động hiệu quả trong điều kiện thực tế, nơi thiết bị cần vận hành liên tục, ổn định và tiêu thụ năng lượng tiết kiệm.

3.4 Nền tảng hiển thị: ThingsBoard

ThingsBoard là một nền tảng mã nguồn mở được phát triển nhằm cung cấp giải pháp toàn diện cho việc quản lý thiết bị IoT, thu thập và xử lý dữ liệu từ xa, cũng

như hiển thị dữ liệu một cách trực quan và có hệ thống. Đây là một trong những nền tảng phổ biến nhất hiện nay trong lĩnh vực IoT, đặc biệt được ưa chuộng trong các ứng dụng yêu cầu giám sát thời gian thực, phản ứng tức thì với sự kiện, và quản trị thiết bị với quy mô lớn. Kiến trúc của ThingsBoard được thiết kế theo mô hình phân tán, hỗ trợ khả năng mở rộng theo chiều ngang, cho phép xử lý đồng thời hàng nghìn thiết bị và luồng dữ liệu mà vẫn đảm bảo hiệu năng ổn định.

Một trong những ưu điểm nổi bật của ThingsBoard là khả năng tương thích với nhiều giao thức truyền thông chuẩn trong lĩnh vực IoT, bao gồm MQTT, CoAP và HTTP. Điều này cho phép nền tảng dễ dàng kết nối với đa dạng thiết bị từ nhiều nhà sản xuất khác nhau, mà không yêu cầu phải thay đổi phần mềm hay phần cứng sẵn có của thiết bị. Giao thức MQTT – vốn được tối ưu cho các môi trường mạng không ổn định hoặc băng thông hạn chế - đặc biệt phù hợp để triển khai với ThingsBoard trong các ứng dụng giám sát từ xa hoặc trong môi trường công nghiệp.

Về mặt hiển thị dữ liệu, ThingsBoard cung cấp hệ thống dashboard động, cho phép người dùng thiết kế giao diện giám sát tùy chỉnh mà không cần lập trình. Các widget được hỗ trợ bao gồm bản đồ tương tác, biểu đồ dạng tuyến, thanh trạng thái, bảng dữ liệu, đồng hồ đo, biểu tượng cảnh báo, v.v. Nhờ khả năng kéo - thả và cấu hình trực quan, người vận hành hệ thống có thể dễ dàng tạo ra các giao diện phù hợp với mục đích giám sát, trình bày dữ liệu theo thời gian thực hoặc truy xuất dữ liệu lịch sử một cách có tổ chức.

Không chỉ dừng lại ở việc thu thập và hiển thị dữ liệu, ThingsBoard còn hỗ trợ xử lý logic sự kiện thông qua hệ thống rule engine - một thành phần cốt lõi cho phép xây dựng các chuỗi xử lý phức tạp theo nguyên tắc điều kiện - hành động. Ví dụ, hệ thống có thể được cấu hình để tự động phát hiện sự kiện bất thường như mất tín hiệu định vị, vượt ngưỡng tốc độ, thay đổi trạng thái hoạt động hoặc rời khỏi khu vực địa lý được phép, và ngay lập tức sinh ra cảnh báo hoặc gửi thông báo đến người quản trị. Việc xử lý logic này được thực hiện trực tiếp trên server, giúp thiết bị đầu cuối giảm bớt gánh nặng xử lý, đồng thời đảm bảo tính thống nhất và độ tin cậy của hệ thống phản hồi.

Bên cạnh đó, ThingsBoard cũng cung cấp khả năng tương tác hai chiều giữa thiết bị và nền tảng quản lý, cho phép hệ thống không chỉ tiếp nhận dữ liệu mà còn gửi cấu hình điều khiển ngược lại thiết bị từ phía máy chủ. Điều này đặc biệt quan trọng trong các ứng dụng cần thay đổi thông số vận hành động, ví dụ như cập nhật chu kỳ gửi dữ liệu, điều chỉnh vùng giám sát hoặc bật/tắt một tính năng cụ thể mà không cần can thiệp thủ công trực tiếp vào thiết bị. Cơ chế này giúp giảm thiểu thời gian vận hành, nâng cao tính linh hoạt và cho phép điều khiển tập trung toàn bộ hệ

thống từ một giao diện duy nhất.

So sánh với các nền tảng tương tự, ThingsBoard thể hiện ưu thế rõ ràng về mức độ tích hợp và khả năng mở rộng. Blynk là một giải pháp đơn giản hơn, dễ tiếp cận cho người mới bắt đầu nhưng lại thiếu tính năng xử lý logic, không hỗ trợ hiển thị dữ liệu lịch sử và khó mở rộng khi triển khai ở quy mô lớn. Node-RED cung cấp công cụ mạnh để xây dựng luồng xử lý dữ liệu bằng hình ảnh nhưng lại không có hệ thống dashboard phù hợp sẵn cho các tác vụ giám sát trực quan. Grafana là nền tảng mạnh về trực quan hóa dữ liệu chuỗi thời gian nhưng không được thiết kế dành riêng cho thiết bị IoT và thiếu khả năng tương tác hai chiều. Trong khi đó, ThingsBoard tích hợp tất cả các thành phần cần thiết – từ thu thập, xử lý, lưu trữ, đến hiển thị và điều khiển - trong một hệ thống duy nhất, mang lại trải nghiệm nhất quán và dễ triển khai hơn trong thực tiễn.

Với mã nguồn mở, tài liệu phong phú và cộng đồng phát triển rộng lớn, ThingsBoard không chỉ thích hợp cho các ứng dụng thương mại mà còn là lựa chọn lý tưởng trong môi trường học thuật, nghiên cứu và phát triển nguyên mẫu. Nền tảng này có thể triển khai trên nhiều loại hạ tầng khác nhau, từ máy chủ cục bộ đến môi trường điện toán đám mây, hỗ trợ đầy đủ các công nghệ hiện đại như Docker, Kubernetes và cơ sở dữ liệu thời gian thực như PostgreSQL hoặc Cassandra.

Việc lựa chọn ThingsBoard nhằm đáp ứng trực tiếp các yêu cầu đã phân tích trong Chương 2 như: hiển thị tọa độ hiện tại, ghi lại lịch sử di chuyển, tính toán quãng đường di chuyển, thiết lập vùng geofence và gửi cảnh báo. Nhờ khả năng cấu hình linh hoạt và khả năng mở rộng, nền tảng này phù hợp để triển khai cả trong môi trường thử nghiệm lẫn ứng dụng thực tế ngoài hiện trường.

3.5 Các thư viện và phần mềm hỗ trợ

Trong quá trình phát triển ứng dụng giám sát vị trí cho thiết bị NaviTracker, một số thư viện và phần mềm hỗ trợ đã được sử dụng nhằm đơn giản hóa việc lập trình, nâng cao độ ổn định của hệ thống, và rút ngắn thời gian triển khai. Các thành phần này chủ yếu phục vụ cho các chức năng thu thập dữ liệu GNSS, xử lý bản tin, truyền thông qua giao thức MQTT, và hiển thị thông tin tại nền tảng ThingsBoard.

Trên phía thiết bị, thư viện Eclipse Paho MQTT Client được sử dụng để hiện thực kết nối MQTT trong môi trường Android. Thư viện này hỗ trợ đầy đủ các chức năng publish/subscribe, quản lý kết nối, và xử lý sự kiện mạng, giúp quá trình truyền dữ liệu từ thiết bị lên server diễn ra ổn định và tin cậy. Ngoài ra, các thư viện truy cập UART như android-serialport-api hỗ trợ thiết bị đọc dữ liệu GNSS trực tiếp từ cổng nối tiếp, trích xuất thông tin định vị theo chuẩn NMEA.

Về phía ThingsBoard, các công cụ tích hợp sẵn như Rule Engine, Attribute Mapping, và Widget Library hỗ trợ xử lý dữ liệu đầu vào và trực quan hóa thông tin trên dashboard. Các widget được sử dụng để hiển thị tọa độ GPS, lịch sử di chuyển, geofence, cũng như trạng thái hoạt động của thiết bị theo thời gian thực.

So với việc tự xây dựng các thành phần này từ đầu, việc sử dụng thư viện và phần mềm có sẵn giúp đảm bảo tính ổn định và tương thích với hệ thống, đồng thời phù hợp với thực tiễn triển khai các ứng dụng IoT hiện nay. Tất cả các thư viện sử dụng đều là mã nguồn mở và được cộng đồng hỗ trợ rộng rãi, đảm bảo dễ dàng bảo trì và mở rộng trong tương lai.

3.6 Tổng kết chương

Trong chương này, tôi đã trình bày các công nghệ, nền tảng và công cụ chính được sử dụng trong quá trình phát triển hệ thống giám sát vị trí NaviTracker. Cụ thể, phần cứng thiết bị NaviTracker được lựa chọn nhờ khả năng tích hợp GNSS và truyền thông di động, đáp ứng yêu cầu giám sát vị trí liên tục. Về phần mềm, ứng dụng được phát triển trên nền Android với ngôn ngữ Java, cho phép triển khai linh hoạt các chức năng như thu thập dữ liệu, truyền thông qua MQTT, và xử lý cục bộ. Giao thức MQTT được sử dụng để đảm bảo việc truyền dữ liệu nhanh, nhẹ và ổn định trong môi trường IoT. Nền tảng ThingsBoard đóng vai trò trung tâm hiển thị, lưu trữ và tương tác với thiết bị thông qua các dashboard trực quan. Cuối cùng, các thư viện và phần mềm hỗ trợ giúp đơn giản hóa việc lập trình, nâng cao độ tin cậy và khả năng mở rộng của hệ thống.

Tất cả các công nghệ trên đều được lựa chọn dựa trên yêu cầu thực tiễn của hệ thống đã phân tích ở Chương 2, đồng thời được cân nhắc kỹ lưỡng so với các lựa chọn thay thế. Việc tích hợp các thành phần này một cách hợp lý đã tạo nền tảng kỹ thuật vững chắc cho việc hiện thực hóa hệ thống trong các chương tiếp theo.

CHƯƠNG 4. THIẾT KẾ PHẦN MỀM THIẾT BỊ GNSS

Chương này trình bày chi tiết về thiết kế phần mềm của hệ thống giám sát GNSS tích hợp. Mục tiêu của thiết kế là đảm bảo hệ thống hoạt động ổn định, hiệu quả và có khả năng thích ứng với môi trường hoạt động của các thiết bị IoT. Nội dung chương bao gồm kiến trúc tổng thể của phần mềm, mô hình phân lớp chức năng, mô tả chi tiết từng mô-đun chính và cơ chế tương tác giữa chúng. Ngoài ra, chương cũng tập trung vào các giải pháp tối ưu hiệu suất xử lý và tiết kiệm năng lượng, nhằm nâng cao độ tin cậy và kéo dài thời gian vận hành của thiết bị trong điều kiện thực tế. Các sơ đồ và bảng minh họa được sử dụng để hỗ trợ việc diễn giải kiến trúc và luồng dữ liệu trong hệ thống.

4.1 Kiến trúc tổng thể hệ thống

Hệ thống được xây dựng với mục tiêu giám sát vị trí thiết bị di động một cách tiết kiệm năng lượng và hiệu quả. Kiến trúc tổng thể bao gồm hai thành phần chính: một thiết bị vi điều khiển ATmega8 và một thiết bị Android tích hợp module GNSS. Hai thành phần này hoạt động phối hợp để đảm bảo việc thu thập và truyền dữ liệu chỉ diễn ra khi thực sự cần thiết.

Vi điều khiển ATmega8 có nhiệm vụ giám sát cảm biến rung. Khi phát hiện rung, ATmega8 sẽ kích hoạt mô-đun GNSS và thiết bị Android thông qua chân điều khiển nguồn (PWRKEY). Ngược lại, nếu không còn phát hiện chuyển động trong một khoảng thời gian xác định, vi điều khiển sẽ chủ động tắt nguồn thiết bị Android nhằm tiết kiệm năng lượng cho toàn hệ thống.

Thiết bị Android sau khi được bật sẽ thực hiện các chức năng thu thập dữ liệu GNSS, kiểm tra tọa độ hiện tại, xác định trạng thái geofence và gửi thông tin này về máy chủ ThingsBoard thông qua giao thức MQTT. Trong suốt thời gian hoạt động, Android cũng ghi nhận trạng thái pin, trạng thái mạng và các thông số vận hành hệ thống khác.

Sự phân chia chức năng giữa ATmega8 và Android đảm bảo rằng thiết bị Android không hoạt động liên tục mà chỉ được kích hoạt trong các tình huống thực sự cần thiết. Điều này giúp kéo dài thời lượng pin của hệ thống, đặc biệt trong các ứng dụng giám sát di động không liên tục. Kiến trúc này cũng cho phép mở rộng linh hoạt với các loại cảm biến khác như cảm biến PIR, cảm biến gia tốc, hoặc công tắc từ để phục vụ các nhu cầu giám sát khác nhau.

4.2 Thiết kế phần mềm trên thiết bị Android

4.2.1 Kiến trúc phân lớp phần mềm

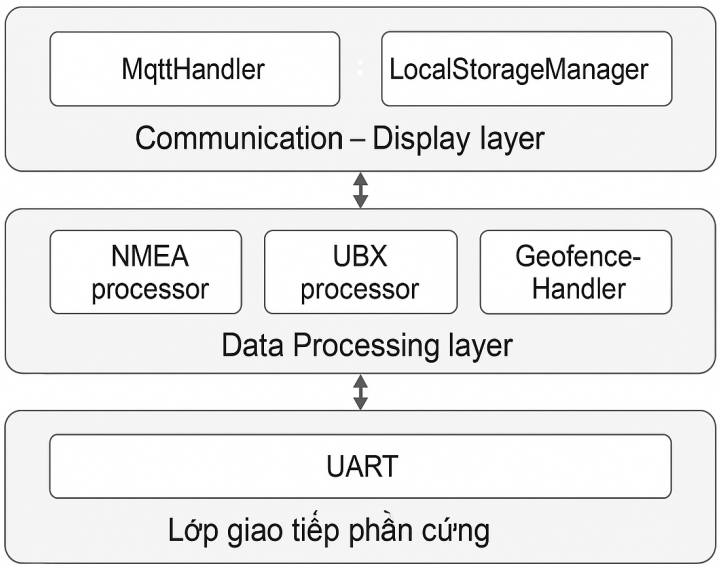
Phần mềm được thiết kế theo kiến trúc phân lớp nhằm đảm bảo tính mô-đun, dễ bảo trì và dễ mở rộng. Hệ thống được chia thành ba lớp chính: lớp giao tiếp phần cứng, lớp xử lý dữ liệu và lớp truyền thông - hiển thị.

Lớp giao tiếp phần cứng thực hiện việc thu thập dữ liệu từ thiết bị GNSS thông qua giao tiếp UART. Các byte dữ liệu được xử lý tuần tự để xây dựng các bản tin định vị hoàn chỉnh theo chuẩn NMEA và UBX.

Lớp xử lý dữ liệu chịu trách nhiệm phân tích, trích xuất và xác thực thông tin định vị từ các bản tin nhận được. Thông tin bao gồm toạ độ, tốc độ, trạng thái tín hiệu và các chỉ số liên quan đến chất lượng định vị. Lớp này cũng xử lý các thuật toán kiểm tra điểm nằm trong vùng địa lý (geofence), đồng thời lưu trữ dữ liệu vào bộ nhớ cục bộ trong trường hợp mất kết nối mạng.

Lớp truyền thông - hiển thị có chức năng gửi dữ liệu định vị và trạng thái thiết bị đến nền tảng ThingsBoard thông qua giao thức MQTT. Lớp này đồng thời tiếp nhận các cấu hình điều khiển từ máy chủ và cập nhật các thông số hoạt động tương ứng. Dữ liệu cũng được hiển thị trực tiếp trên giao diện ứng dụng người dùng để phục vụ giám sát thời gian thực.

Kiến trúc này cho phép phân tách rõ ràng giữa các chức năng, giảm thiểu sự phụ thuộc giữa các mô-đun, đồng thời hỗ trợ triển khai các cơ chế tối ưu tài nguyên và năng lượng phù hợp với đặc thù của thiết bị nhúng.



Hình 4.1: Mô hình phân lớp

4.2.2 Lớp giao tiếp phần cứng

Lớp giao tiếp phần cứng là lớp nền trong toàn bộ kiến trúc phần mềm, chịu trách nhiệm kết nối với thiết bị GNSS và thu nhận dữ liệu thông qua giao tiếp UART. Thành phần trung tâm trong lớp này là lớp UartReader, được xây dựng để thực hiện các thao tác mở cổng, đọc dữ liệu byte theo thời gian thực và gửi dữ liệu điều khiển xuống thiết bị.

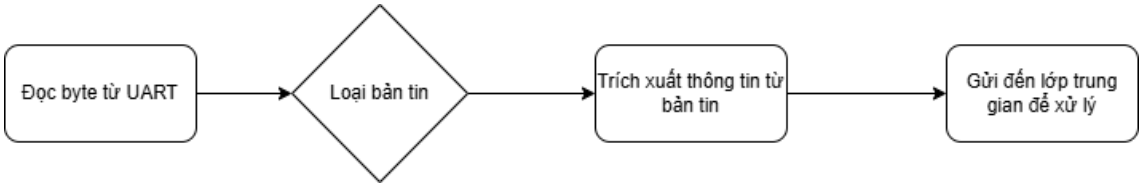
Ngay khi thiết bị khởi động, ứng dụng sử dụng UartReader để mở cổng UART tại đường dẫn cấu hình sẵn (/dev/ttyHSL0). Nếu mở cổng thành công, một luồng xử lý riêng sẽ được kích hoạt để liên tục đọc từng byte từ thiết bị GNSS. Mỗi byte nhận được sẽ được chuyển vào chuỗi xử lý kế tiếp thông qua các lớp phân tích dữ liệu.

Luồng xử lý dữ liệu sau khi nhận byte từ UART được chia thành hai nhánh chính, tương ứng với hai định dạng bản tin phổ biến của thiết bị GNSS là NMEA và UBX. Nếu dữ liệu là bản tin NMEA, tức có ký tự bắt đầu là dấu $, luồng byte sẽ được chuyển đến lớp NMEAProcessor để tái tạo thành một chuỗi hoàn chỉnh. Sau đó, chuỗi này được phân tích bởi lớp NMEAHandler nhằm trích xuất thông tin định vị như toạ độ, tốc độ di chuyển, hướng đi và tính hợp lệ của tín hiệu.

Ngược lại, nếu byte đầu là cặp 0xB5 0x62 thì dữ liệu thuộc định dạng UBX. Lúc này, nó sẽ được truyền đến các bộ xử lý như GNSSProcessor, NavStatusProcessor hoặc SFRBXProcessor để kiểm tra độ dài, xác thực checksum và phân tích phần nội dung payload. Các bộ xử lý này hỗ trợ việc trích xuất thông tin cấu hình GNSS hoặc trạng thái vệ tinh đang hoạt động.

Dữ liệu định vị hợp lệ sau khi được trích xuất sẽ được chuyển tiếp lên lớp xử lý trung gian để tính trung bình toạ độ, phát hiện ra các vi phạm geofence hoặc gửi thông tin đến nền tảng ThingsBoard thông qua giao thức MQTT. Bên cạnh chức năng đọc, lớp giao tiếp phần cứng còn cho phép gửi dữ liệu điều khiển xuống thiết bị GNSS. Chẳng hạn, lớp GNSSControl có thể tạo bản tin UBX-CFG-GNSS để bật hoặc tắt các hệ thống vệ tinh như GPS, GLONASS, Galileo hoặc BeiDou. Bản tin sau đó được gửi đi thông qua UART nhờ phương thức ghi của UartReader.

Luồng xử lý UART đảm bảo rằng dữ liệu GNSS được tiếp nhận liên tục và chính xác, bất kể trong điều kiện hoạt động bình thường hay gián đoạn mạng. Đây là bước đầu tiên và cũng là tiền đề quan trọng để hệ thống thực hiện các tác vụ giám sát, phân tích và truyền thông một cách ổn định, đáng tin cậy.



Hình 4.2: Luồng xử lý dữ liệu lớp phần cứng

4.2.3 Lớp xử lý dữ liệu

Lớp xử lý dữ liệu đóng vai trò trung gian trong hệ thống, đảm nhận nhiệm vụ tiếp nhận dữ liệu thô từ lớp giao tiếp phần cứng, sau đó phân tích, trích xuất và xử lý thông tin định vị cần thiết. Lớp này được cấu trúc thành nhiều mô-đun chức năng, mỗi mô-đun đảm nhận một nhiệm vụ chuyên biệt tương ứng với các loại bản tin định vị khác nhau.

a, Các bản tin từ module GNSS ublox NEO-M9

Module GNSS u-blox NEO-M9 hỗ trợ truyền dữ liệu định vị qua hai định dạng bản tin chính là NMEA và UBX. Bản tin NMEA được sử dụng phổ biến để cung cấp thông tin vị trí dưới dạng chuỗi ASCII, trong khi bản tin UBX là định dạng nhị phân độc quyền của u-blox, cho phép truyền thông tin cấu hình và trạng thái với độ chính xác và linh hoạt cao hơn. Hệ thống phần mềm sử dụng bốn bản tin chính: NMEA-RMC, UBX-CFG-GNSS, UBX-NAV-STATUS và UBX-RXM-SFRBX, với chức năng và cấu trúc như sau:

Bản tin RMC (Recommended Minimum Specific GNSS Data) cung cấp thông tin tối thiểu cần thiết về định vị như thời gian, vị trí, tốc độ và hướng. Chuỗi bắt đầu bằng $GxRMC và kết thúc bằng checksum. Các trường trong bản tin có định dạng văn bản, phân tách bằng dấu phẩy.

Field	Tên trường	Định dạng	Đơn vị	Ví dụ	Mô tả
0	xxRMC	Chuỗi	-	GPRMC	Mã tin nhắn RMC
1	Thời gian	hhmmss.sss	-	083559.00	Thời gian UTC
2	Trạng thái	Ký tự	-	A	Trạng thái tín hiệu (A = hợp lệ)
3	Vĩ độ	ddmm.mmmm	Độ	4717.11437	Vĩ độ
4	NS	Ký tự	-	N	Chỉ báo Bắc - Nam
5	Kinh độ	dddmm.mmmm	Độ	00833.91522	Kinh độ
6	EW	Ký tự	-	E	Chỉ báo Đông - Tây
7	Tốc độ	float	knots	22.4	Tốc độ mặt đất
8	Hướng đi	float	Độ	84.4	Hướng chuyển động

Field	Tên trường	Định dạng	Đơn vị	Ví dụ	Mô tả
9	Ngày	ddmmyy	-	230394	Ngày tháng năm
10	mv	Số	Độ	-	Biến thiên từ trường
11	mvEW	Ký tự	-	-	Biến thiên từ trường Đông - Tây
12	posMode	Ký tự	-	A	Chỉ báo chế độ
13	navStatus	Ký tự	-	V	Trạng thái định vị
14	cs	Thập lục phân	-	57	Checksum
15	CRLF	Ký tự	-	-	Ký tự xuống dòng

Bảng 4.1: Cấu trúc bản tin NMEA-RMC

Bản tin VTG (Course Over Ground and Ground Speed) cung cấp thông tin về hướng di chuyển và tốc độ của thiết bị so với mặt đất. Chuỗi bản tin này bắt đầu bằng ký hiệu $GxVTG và kết thúc bằng mã kiểm tra lỗi (checksum). Các trường dữ liệu trong bản tin có định dạng văn bản, được phân tách bằng dấu phẩy.

Field	Tên trường	Định dạng	Đơn vị	Ví dụ	Mô tả
0	xxVTG	Chuỗi	-	GPVTG	Mã bản tin VTG (xx là Talker ID)
1	cogt	Số thực	Độ	77.52	Hướng di chuyển mặt đất (đúng theo phương vị thực)
2	cogtUnit	Ký tự	-	T	Đơn vị hướng: T = True (cố định)
3	cogm	Số thực	Độ	-	Hướng di chuyển mặt đất (theo phương vị từ, nếu có)
4	cogmUnit	Ký tự	-	M	Đơn vị hướng: M = Magnetic (cố định)
5	sogn	Số thực	knots	0.004	Tốc độ di chuyển mặt đất theo đơn vị knots
6	sognUnit	Ký tự	-	N	Đơn vị tốc độ: N = knots (cố định)
7	sogk	Số thực	km/h	0.008	Tốc độ di chuyển mặt đất theo đơn vị km/h

Field	Tên trường	Định dạng	Đơn vị	Ví dụ	Mô tả
8	sogkUnit	Ký tự	-	K	Đơn vị tốc độ: K = kilometer/hour (cố định)
9	posMode	Ký tự	-	A	Chỉ báo chế độ định vị (xuất hiện từ phiên bản NMEA 2.3 trở lên)
10	cs	Thập lục phân	-	06	Checksum để kiểm tra lỗi
11	CRLF	Ký tự	-	-	Ký tự xuống dòng kết thúc bản tin

Bảng 4.2: Cấu trúc bản tin NMEA-VTG

Bản tin GNS (GNSS Fix Data) cung cấp thông tin về toạ độ định vị, trạng thái tín hiệu và số lượng vệ tinh được sử dụng từ các hệ thống GNSS. Chuỗi bắt đầu bằng $GxGNS và kết thúc bằng checksum. Các trường trong bản tin có định dạng văn bản, phân tách bằng dấu phẩy.

Field	Tên trường	Định dạng	Đơn vị	Ví dụ	Mô tả
0	xxGGA	Chuỗi	-	GPGGA	Mã bản tin GGA (xx là Talker ID)
1	time	hhmmss.ss	-	092725.00	Thời gian UTC
2	lat	ddmm.mmmmm	Độ	4717.11399	Vĩ độ (độ và phút)
3	NS	Ký tự	-	N	Chỉ báo Bắc/Nam
4	lon	dddmm.mmmmm	Độ	00833.91590	Kinh độ (độ và phút)
5	EW	Ký tự	-	E	Chỉ báo Đông/Tây
6	quality	Chữ số	-	1	Chất lượng tín hiệu định vị
7	numSV	Số	-	08	Số vệ tinh sử dụng (0–12)
8	HDOP	Số thực	-	1.01	Độ phân giải theo phương ngang
9	alt	Số thực	m	499.6	Độ cao so với mực nước biển trung bình
10	altUnit	Ký tự	-	M	Đơn vị độ cao: mét (cố định)

Field	Tên trường	Định dạng	Đơn vị	Ví dụ	Mô tả
11	sep	Số thực	m	48.0	Độ lệch địa hình giữa ellipsoid và mực nước biển
12	sepUnit	Ký tự	-	M	Đơn vị độ lệch địa hình: mét (cố định)
13	diffAge	Số	s	-	Độ tuổi hiệu chỉnh sai số (nếu có)
14	diffStation	Số	-	-	Mã trạm phát hiệu chỉnh (nếu có)
15	cs	Thập lục phân	-	5B	Checksum để kiểm tra lỗi
16	CRLF	Ký tự	-	-	Ký tự xuống dòng kết thúc bản tin

Bảng 4.3: Cấu trúc bản tin NMEA-GGA

Bản tin GNS (GNSS Fix Data) cung cấp thông tin về vị trí, trạng thái tín hiệu và số lượng vệ tinh được sử dụng từ các hệ thống định vị toàn cầu. Chuỗi bắt đầu bằng $GxGNS và kết thúc bằng checksum. Các trường trong bản tin có định dạng văn bản, phân tách bằng dấu phẩy.

Field	Tên trường	Định dạng	Đơn vị	Ví dụ	Mô tả
0	xxGNS	Chuỗi	-	GPGNS	Mã bản tin GNS (xx là Talker ID)
1	time	hhmmss.ss	-	091547.00	Thời gian UTC
2	lat	ddmm.mmmmm	Độ	5114.50897	Vĩ độ (độ và phút)
3	NS	Ký tự	-	N	Chỉ báo Bắc/Nam
4	lon	dddmm.mmmmm	Độ	00012.28663	Kinh độ (độ và phút)
5	EW	Ký tự	-	E	Chỉ báo Đông/Tây
6	posMode	Ký tự	-	AAAA	Chế độ định vị của từng hệ GNSS (GPS, GLONASS, Galileo, BeiDou)
7	numSV	Số	-	10	Số lượng vệ tinh được sử dụng (0–99)
8	HDOP	Số thực	-	0.83	Độ phân giải theo phương ngang

Field	Tên trường	Định dạng	Đơn vị	Ví dụ	Mô tả
9	alt	Số thực	m	111.1	Độ cao so với mực nước biển trung bình
10	sep	Số thực	m	45.6	Độ lệch địa hình giữa ellipsoid và mực nước biển
11	diffAge	Số	s	-	Độ tuổi hiệu chỉnh sai số (nếu có)
12	diffStation	Số	-	-	Mã trạm phát hiệu chỉnh (nếu có)
13	navStatus	Ký tự	-	V	Trạng thái điều hướng (có trong NMEA 4.10 trở lên)
14	cs	Thập lục phân	-	71	Checksum để kiểm tra lỗi
15	CRLF	Ký tự	-	-	Ký tự xuống dòng kết thúc bản tin

Bảng 4.4: Cấu trúc bản tin NMEA-GNS

Bản tin UBX-CFG-GNSS được sử dụng để cấu hình hệ thống GNSS đang hoạt động trên thiết bị. Bản tin chứa thông tin về số lượng kênh theo phần cứng và phần mềm, số hệ thống GNSS được cấu hình và trạng thái bật/tắt của từng hệ thống.

Offset	Tên trường	Định dạng	Đơn vị	Ví dụ	Mô tả
0	msgVer	uint8	-	0	Phiên bản bản tin
1	numTrkChHw	uint8	kênh	32	Số kênh phần cứng hỗ trợ
2	numTrkChUse	uint8	kênh	32	Số kênh GNSS được sử dụng
3	numConfigBlocks	uint8	-	6	Số hệ thống GNSS được cấu hình
4	repeated blocks	struct	-	...	Dữ liệu cấu hình cho từng hệ thống GNSS (ID, số kênh, cờ enable, flags, v.v.)

Bảng 4.5: Cấu trúc bản tin UBX-CFG-GNSS

Bản tin `UBX-NAV-STATUS` cung cấp thông tin về trạng thái định vị hiện tại của thiết bị, bao gồm kiểu fix, các cờ trạng thái, và thông tin xác thực tín hiệu.

Offset	Tên trường	Định dạng	Đơn vị	Ví dụ	Mô tả
0	iTOW	uint32	ms	501867000	Thời gian GPS (Time of Week)
4	gpsFix	uint8	-	3	Trạng thái định vị (3 = 3D fix)
5	flags	bitfield	-	0x0D	Các cờ phản ánh chất lượng và độ tin cậy tín hiệu
6	fixStat	bitfield	-	0x00	Trạng thái xác định vị trí
7	flags2	bitfield	-	0x00	Các cờ bổ sung

Bảng 4.6: Cấu trúc bản tin UBX-NAV-STATUS

Bản tin `UBX-RXM-SFRBX` chứa dữ liệu thô từ tín hiệu vệ tinh, thường được sử dụng cho mục đích phân tích tín hiệu ở lớp thấp. Bản tin cung cấp thông tin như ID vệ tinh, tần số và các từ dữ liệu 32 bit từ mỗi vệ tinh.

Offset	Tên trường	Định dạng	Đơn vị	Ví dụ	Mô tả
0	gnssId	uint8	-	0	ID hệ thống GNSS (0 = GPS)
1	svId	uint8	-	3	ID vệ tinh
2	reserved1	uint8	-	0	Trường dự phòng
3	freqId	uint8	-	0	ID tần số tín hiệu
4	numWords	uint8	-	10	Số lượng từ dữ liệu GNSS
8	dwrd	uint32[10]	-	...	Dữ liệu 32 bit từ tín hiệu vệ tinh

Bảng 4.7: Cấu trúc bản tin UBX-RXM-SFRBX

b, Luồng xử lý dữ liệu

Lớp xử lý dữ liệu đóng vai trò trung gian trong hệ thống, đảm nhận nhiệm vụ tiếp nhận dữ liệu thô từ lớp giao tiếp phần cứng, sau đó phân tích, trích xuất và xử lý thông tin định vị cần thiết. Lớp này được cấu trúc thành nhiều mô-đun chức năng, mỗi mô-đun đảm nhận một nhiệm vụ chuyên biệt tương ứng với các loại bản tin định vị khác nhau.

Đối với dữ liệu NMEA, sau khi được tái dựng thành chuỗi hoàn chỉnh, chuỗi này sẽ được truyền vào lớp NMEAHandler để trích xuất các thông tin như vĩ độ, kinh độ, tốc độ di chuyển và trạng thái tín hiệu. Việc xử lý các câu lệnh GGA, GNS, RMC, hoặc VTG được thực hiện thông qua các parser chuyên biệt, giúp đảm bảo khả năng trích xuất chính xác từ đa nguồn GNSS.

Trong trường hợp bản tin có định dạng UBX, hệ thống sử dụng các mô-đun chuyên biệt như GNSSProcessor, NavStatusProcessor và SFRBXProcessor. Các mô-đun này thực hiện việc nhận diện chuỗi bản tin, kiểm tra độ dài hợp lệ, xác thực checksum và trích xuất thông tin cấu hình hoặc trạng thái vệ tinh. Thông tin này phục vụ cho việc xác định số lượng hệ thống GNSS đang hoạt động, theo dõi

chất lượng tín hiệu hoặc tái cấu hình hoạt động GNSS thông qua các bản tin điều khiển.

Sau khi dữ liệu được phân tích và xác thực, lớp xử lý tiếp tục thực hiện các thao tác cao hơn như tính tốc độ di chuyển dựa trên thời gian thực và khoảng cách giữa các toạ độ, xác định thay đổi vị trí vượt quá ngưỡng cấu hình, hoặc phát hiện sự thay đổi trạng thái hoạt động của GNSS. Các dữ liệu hợp lệ được truyền đến lớp truyền thông để hiển thị hoặc đồng bộ với nền tảng giám sát.

Một chức năng quan trọng khác của lớp xử lý dữ liệu là cơ chế geofence, cho phép hệ thống phát hiện khi thiết bị di chuyển ra ngoài phạm vi khu vực đã được định nghĩa từ trước. Các khu vực này được biểu diễn dưới dạng tập hợp các đỉnh tạo thành một đa giác khép kín, thường tương ứng với ranh giới hành chính như tỉnh hoặc thành phố. Tọa độ hiện tại của thiết bị, sau khi được trích xuất và xác thực, sẽ được chuyển đến mô-đun GeofenceHandler để kiểm tra xem điểm đó có nằm trong vùng giới hạn hay không.

Thuật toán được sử dụng là thuật toán tia cắt (ray casting algorithm), hoạt động theo nguyên lý đếm số lần một tia xuất phát từ điểm cần kiểm tra cắt các cạnh của đa giác. Nếu số lần cắt là số lẻ, điểm được xác định là nằm trong vùng; ngược lại, nếu là số chẵn, điểm nằm ngoài. Trong quá trình xử lý, thuật toán cũng bao gồm cơ chế hiệu chỉnh để tránh các trường hợp biên gây ra sai số, ví dụ như khi điểm trùng với một đỉnh của đa giác hoặc nằm trên một cạnh ngang. Điều này giúp tăng độ tin cậy của quá trình xác định vị trí tương đối của thiết bị với khu vực cấu hình.

Kết quả kiểm tra geofence sẽ được sử dụng để tạo các bản tin cảnh báo, thông qua đó nền tảng ThingsBoard có thể hiển thị trạng thái thiết bị như "bên trong" hoặc "bên ngoài" vùng theo thời gian thực. Nếu thiết bị nằm ngoài vùng giới hạn, hệ thống sẽ gửi một thuộc tính isOutside = true lên server. Khi thiết bị quay lại vùng an toàn, giá trị này sẽ được cập nhật lại. Nhờ cơ chế này, hệ thống có thể hỗ trợ các ứng dụng yêu cầu giám sát chặt chẽ vị trí như theo dõi phương tiện, quản lý nhân lực hoặc cảnh báo vi phạm khu vực an toàn.

Ngoài ra, để đảm bảo hoạt động ổn định trong môi trường mạng không liên tục, các bản ghi dữ liệu định vị sẽ được lưu tạm thời bằng lớp LocalStorageManager. Khi mạng ổn định trở lại, dữ liệu này sẽ được đồng bộ qua lớp truyền thông MQTT. Cơ chế này giúp tránh mất dữ liệu trong quá trình thiết bị di chuyển qua các khu vực có chất lượng kết nối kém.

Toàn bộ lớp xử lý dữ liệu được thiết kế theo hướng mở rộng và tương thích với nhiều loại bản tin GNSS khác nhau, đảm bảo hiệu quả phân tích và tính linh hoạt trong các tình huống vận hành thực tế.

4.2.4 Lớp truyền thông

Lớp truyền thông có vai trò kết nối giữa thiết bị và nền tảng ThingsBoard thông qua giao thức MQTT. Thành phần trung tâm của lớp này là lớp MqttHandler, chịu trách nhiệm khởi tạo kết nối, gửi dữ liệu định vị, gửi các thông số trạng thái hệ thống và tiếp nhận các cấu hình điều khiển từ xa.

Ngay khi khởi động, MqttHandler tiến hành kết nối đến máy chủ MQTT bằng định danh thiết bị ThingsBoard. Sau khi kết nối thành công, thiết bị sẽ tự động gửi một bản tin init chứa các thuộc tính cấu hình hiện tại của thiết bị.

Chương trình sử dụng hai topic chính: telemetryTopic để gửi dữ liệu thời gian thực như toạ độ và tốc độ di chuyển và attributeTopic để gửi hoặc nhận các thuộc tính cấu hình hệ thống như trạng thái pin, geofence hoặc điều kiện hoạt động. Tùy theo chế độ định vị, hệ thống sử dụng hai chế độ gửi dữ liệu:

- DEVICE_LOCATION: gửi tọa độ từ dịch vụ định vị của Android.
- UBLOX_LOCATION: gửi tọa độ GNSS trung bình từ module u-blox.

Để tối ưu hiệu suất truyền thông và tiết kiệm năng lượng, lớp này chỉ gửi dữ liệu khi thiết bị có kết nối mạng hoặc khi tọa độ thay đổi vượt qua ngưỡng cấu hình. Nếu mất mạng, dữ liệu sẽ được lưu tạm thời trong bộ nhớ cục bộ thông qua lớp LocalStorageManager và tự động đồng bộ lại khi mạng ổn định trở lại.

Ngoài dữ liệu định vị, MqttHandler cũng xử lý các bản tin phản hồi từ ThingsBoard. Các bản tin thuộc chủ đề thuộc tính (`attribute`) được phân tích và cập nhật trực tiếp vào các thông số hoạt động như khoảng cách tối đa (maxDistance), thời gian tối đa (maxTimeout), tên tỉnh giám sát hoặc trạng thái bật/tắt vùng geofence. Nếu geofence được bật, ranh giới địa lý tương ứng với tỉnh được lấy từ file provinces.json và gửi ngược lên server để hiển thị trên dashboard.

Cuối cùng, lớp này còn hỗ trợ gửi các thuộc tính trạng thái như mức pin, trạng thái sạc, trạng thái GPS và cờ xác định thiết bị có nằm ngoài khu vực địa lý không (isOutside). Toàn bộ lớp được thiết kế bất đồng bộ, với cơ chế tự động kết nối lại khi xảy ra gián đoạn để đảm bảo tính liên tục trong quá trình truyền nhận dữ liệu.

4.3 Phần mềm nhúng trên vi điều khiển ATmega8

Phần mềm nhúng trên vi điều khiển ATmega8 đóng vai trò trung gian điều phối giữa các cảm biến vật lý và thiết bị Android, nhằm đảm bảo hệ thống chỉ được kích hoạt khi thực sự cần thiết. Cụ thể, ATmega8 giám sát tín hiệu từ cảm biến rung để phát hiện sự hiện diện hoặc hoạt động của người dùng. Khi có tín hiệu chuyển động, vi điều khiển thực hiện thao tác cấp nguồn và khởi động thiết bị Android. Ngược lại, nếu không còn chuyển động trong một khoảng thời gian định trước, hệ

thống sẽ chủ động ngắt nguồn và đưa thiết bị về trạng thái tiết kiệm năng lượng.

Chương này trình bày chi tiết thiết kế và triển khai phần mềm nhúng cho ATmega8, bao gồm cấu hình các chân I/O, thiết lập ngắt ngoài, lựa chọn và cài đặt chế độ tiết kiệm năng lượng phù hợp, cùng với mô hình điều khiển theo máy trạng thái. Thiết kế này cho thấy sự cân bằng giữa hiệu quả sử dụng năng lượng và khả năng đáp ứng nhanh với các sự kiện ngoài môi trường, góp phần nâng cao độ bền và tính ổn định cho toàn bộ hệ thống.

4.3.1 Cấu hình các chân GPIO trên vi điều khiển ATmega8

GPIO (General Purpose Input/Output) là các chân vào/ra mục đích chung cho phép vi điều khiển trao đổi tín hiệu số với các thiết bị ngoại vi. Các chân này có thể được thiết lập để hoạt động ở chế độ đầu vào (nhận tín hiệu) hoặc đầu ra (phát tín hiệu). Trong các hệ thống nhúng, GPIO đóng vai trò quan trọng trong việc điều khiển thiết bị, nhận tín hiệu điều kiện hoặc phản hồi trạng thái. Trên vi điều khiển ATmega8, toàn bộ các chân của các cổng I/O như PORTB, PORTC và PORTD đều có thể được sử dụng như các GPIO đa năng. Điều này giúp lập trình viên có khả năng cấu hình linh hoạt tùy theo yêu cầu phần cứng và chức năng ứng dụng.

Để cấu hình và vận hành các chân GPIO, ATmega8 cung cấp ba thanh ghi điều khiển chính cho mỗi cổng:

- **DDRx (Data Direction Register)**: đây là thanh ghi 8 bit dùng để xác định hướng dữ liệu của từng chân trong cổng. Nếu bit tại vị trí tương ứng được ghi là 1, chân đó sẽ được cấu hình là đầu ra. Ngược lại, nếu bit đó bằng 0, chân sẽ được cấu hình là đầu vào. Ví dụ, thiết lập DDRC |= (1 « PC0); sẽ cấu hình chân PC0 là đầu ra.
- **PORTx**: thanh ghi này có vai trò khác nhau tùy thuộc vào hướng của chân.
  - Nếu chân đang được cấu hình là đầu ra, ghi giá trị logic 1 vào bit tương ứng sẽ đặt chân ở mức cao (logic HIGH), còn ghi giá trị 0 sẽ đặt ở mức thấp (logic LOW). Ví dụ, PORTC |= (1 « PC0); sẽ kéo chân PC0 lên mức cao.
  - Nếu chân đang ở chế độ đầu vào, ghi giá trị 1 vào bit tương ứng sẽ bật điện trở kéo lên nội (pull-up), giúp đảm bảo tín hiệu ổn định khi không có tác động ngoại vi. Ví dụ, PORTD |= (1 « PD3); sẽ bật pull-up cho chân PD3 khi đang là đầu vào.
- **PINx**: đây là thanh ghi chỉ đọc, cho phép kiểm tra mức logic hiện tại tại các chân. Thường được dùng trong xử lý tín hiệu từ cảm biến hoặc nút nhấn. Ví dụ, if (PIND & (1 « PD3)) sẽ trả về đúng nếu chân PD3 đang ở mức cao.

Trong đồ án này, các chân GPIO của ATmega8 được sử dụng để điều khiển các chức năng quan trọng của hệ thống giám sát. Cụ thể, có bốn chân được sử dụng và được cấu hình như sau:

- **Chân PC0 (PWRKEY):** được cấu hình là **đầu ra (output)**. Chân này được kết nối với chân điều khiển PWRKEY của thiết bị Android để thực hiện thao tác bật hoặc tắt thiết bị. Trạng thái ban đầu của chân này được đặt ở mức logic thấp (PORTC &= ~(1 « PC0)), đảm bảo thiết bị không bị kích hoạt khi hệ thống mới khởi động.
- **Chân PC1 (LED):** được cấu hình là **đầu ra (output)**. Đây là chân điều khiển đèn LED dùng để hiển thị trạng thái hoạt động của hệ thống. Khi chân này ở mức cao, đèn LED sẽ sáng, biểu thị rằng hệ thống đang ở trạng thái bật. Trạng thái ban đầu được thiết lập ở mức logic thấp để LED tắt khi chưa có sự kiện kích hoạt.
- **Chân PB1 (UBLOX):** được cấu hình là **đầu ra (output)**. Chân này điều khiển đường cấp nguồn cho module định vị GNSS u-blox. Khi chân này được đặt ở mức cao, nguồn cho module GNSS sẽ được cấp. Trạng thái khởi tạo được đặt ở mức thấp, tức là module GNSS chưa được cấp nguồn khi hệ thống khởi động.
- **Chân PD3 (INPUT):** được cấu hình là **đầu vào (input)**. Đây là chân kết nối với cảm biến chuyển động (ví dụ cảm biến rung) nhằm phát hiện có hoạt động từ môi trường bên ngoài. Khi cảm biến phát hiện chuyển động, chân này sẽ chuyển sang mức cao. Để đảm bảo tín hiệu ổn định khi không có tác động, điện trở kéo lên nội được bật thông qua việc ghi giá trị 1 vào bit tương ứng trong thanh ghi PORTD.

Cách cấu hình này đảm bảo rằng mỗi chân đều có chức năng rõ ràng, định hướng hoạt động cụ thể và được khởi tạo với trạng thái an toàn. Điều này giúp hệ thống vận hành ổn định và phản hồi chính xác theo các tín hiệu đầu vào cũng như yêu cầu điều khiển đầu ra. Các thiết bị được kết nối sẽ chỉ hoạt động khi có điều kiện thích hợp, tránh tiêu tốn năng lượng không cần thiết hoặc phát sinh lỗi trong quá trình khởi động.

Việc cấu hình các chân này được thực hiện trong hàm setup_io() với các bước như sau:

- Các chân **PC0**, **PC1** và **PB1** được thiết lập làm đầu ra bằng cách ghi giá trị 1 vào các bit tương ứng trong các thanh ghi DDRC và DDRB.
- Trạng thái ban đầu của các chân đầu ra được đặt ở mức logic thấp bằng cách

ghi giá trị 0 vào các bit tương ứng trong PORTC và PORTB, giúp tránh kích hoạt ngoài ý muốn sau khi khởi động hệ thống.

- Chân **PD3** được cấu hình làm đầu vào bằng cách ghi giá trị 0 vào bit tương ứng trong DDRD, đồng thời bật điện trở kéo lên nội bằng cách ghi 1 vào bit tương ứng trong PORTD.

Việc bật điện trở kéo lên nội cho chân đầu vào là cần thiết để đảm bảo tín hiệu ổn định trong trạng thái nghỉ, tránh hiện tượng trôi logic. Cách cấu hình nêu trên đảm bảo rằng hệ thống có trạng thái xác định, đáng tin cậy và sẵn sàng hoạt động chính xác ngay từ thời điểm khởi động. Điều này góp phần quan trọng vào độ ổn định và tính đúng đắn trong điều khiển thiết bị Android theo điều kiện thực tế.

4.3.2 Cơ chế ngắt và cấu hình ngắt trên ATmega8

Cơ chế ngắt (interrupt) là một phần tử quan trọng trong thiết kế hệ thống nhúng, giúp vi điều khiển có khả năng phản ứng tức thời với các sự kiện bất đồng bộ từ môi trường. Ngắt cho phép hệ thống tạm dừng chương trình đang thực thi để xử lý một sự kiện khẩn cấp, sau đó tiếp tục lại đúng vị trí bị tạm dừng. Điều này làm giảm độ trễ phản hồi và tiết kiệm tài nguyên so với việc kiểm tra liên tục (polling) trạng thái các thiết bị.

Quá trình xử lý ngắt diễn ra theo các bước cơ bản sau:

- **Bước 1:** Một điều kiện ngắt được phát sinh, ví dụ: tín hiệu từ cảm biến, bộ định thời tràn, dữ liệu được nhận qua UART, v.v.
- **Bước 2:** Vi điều khiển kiểm tra xem ngắt đó có được cho phép không (bit enable của ngắt có được bật và ngắt toàn cục đã được cho phép bằng lệnh sei()).
- **Bước 3:** Nếu ngắt được chấp nhận, vi điều khiển tạm dừng chương trình chính (main), lưu vị trí hiện tại (Program Counter) và nhảy tới hàm xử lý ngắt tương ứng (ISR - Interrupt Service Routine).
- **Bước 4:** ISR được thực hiện. Sau khi hoàn tất, lệnh RETI (Return from Interrupt) được gọi để khôi phục trạng thái và tiếp tục chương trình chính từ vị trí bị gián đoạn.

Cơ chế này đảm bảo rằng hệ thống luôn sẵn sàng phản hồi nhanh chóng mà không cần tiêu tốn thời gian kiểm tra liên tục, đặc biệt hữu ích trong các ứng dụng tiết kiệm năng lượng hoặc thời gian thực.

ATmega8 hỗ trợ nhiều loại ngắt khác nhau, bao phủ toàn bộ các chức năng quan trọng của hệ thống:

- **Ngắt ngoài (External Interrupts):** gồm INT0 (PD2) và INT1 (PD3), có thể cấu hình để kích hoạt theo cạnh lên, cạnh xuống hoặc mức thấp. Dùng để phản ứng với các tín hiệu từ bên ngoài, ví dụ như cảm biến, công tắc, v.v.
- **Ngắt thay đổi mức logic (Pin Change Interrupts):** kích hoạt khi bất kỳ chân nào trong một nhóm cụ thể thay đổi trạng thái. Nhóm này không áp dụng được cho tất cả các chân, nhưng giúp mở rộng khả năng giám sát sự thay đổi tín hiệu đầu vào.
- **Ngắt bộ định thời (Timer/Counter Interrupts):**
  - Timer0 Overflow (tràn bộ đếm 8-bit)
  - Timer1 Overflow và Compare Match (cho bộ đếm 16-bit)
  - Timer2 Overflow

  Dùng để tạo các sự kiện định kỳ, đo thời gian, hoặc điều chế PWM.
- **Ngắt giao tiếp nối tiếp (USART Interrupts):** phát sinh khi có dữ liệu đến hoặc đi qua UART, ví dụ: ngắt nhận ký tự (RXC), ngắt truyền xong (TXC).
- **Ngắt chuyển đổi ADC (ADC Conversion Complete Interrupt):** được kích hoạt khi quá trình chuyển đổi tương tự – số (analog-to-digital) kết thúc.
- **Ngắt Watchdog Timer:** xảy ra khi bộ đếm watchdog hết thời gian mà không được đặt lại, giúp khởi động lại hệ thống nếu chương trình bị treo.

Trong phạm vi đồ án này, tôi chỉ sử dụng ngắt ngoài để kiểm soát chế độ của vi điều khiển. Để sử dụng các ngắt ngoài, cần cấu hình các thanh ghi sau:

- **GICR (General Interrupt Control Register):** thanh ghi này cho phép bật ngắt INT0 hoặc INT1. Cụ thể:
  - GICR |= (1 « INT1): bật ngắt ngoài INT1.
- **MCUCR (MCU Control Register):** xác định điều kiện kích hoạt ngắt cho INT0 và INT1 bằng hai bit ISCxx:
  - ISC11 = 1 và ISC10 = 0: ngắt xảy ra khi có cạnh xuống tại chân INT1 (PD3).
  - ISC11 = 1 và ISC10 = 1: ngắt xảy ra khi có cạnh lên.
  - ISC11 = 0 và ISC10 = 0: ngắt xảy ra khi mức thấp liên tục.
- **SREG (Status Register):** chứa bit I - bit cho phép ngắt toàn cục. Để cho phép hệ thống phản hồi ngắt, cần bật bit này bằng lệnh sei().

Trong đồ án, ngắt INT1 được cấu hình để kích hoạt khi có cạnh xuống tại chân

PD3. Điều này phù hợp với nguyên lý hoạt động của cảm biến rung, khi tín hiệu đầu ra chuyển từ mức cao sang thấp để báo hiệu có chuyển động. Việc cấu hình được thực hiện trong hàm setup_interrupts() như sau:

- Ghi 1 vào ISC11 và 0 vào ISC10 để chọn chế độ kích hoạt ngắt theo cạnh xuống: MCUCR |= (1 « ISC11); và MCUCR &= ~(1 « ISC10);
- Cho phép ngắt INT1 bằng cách ghi 1 vào bit INT1 của thanh ghi GICR: GICR |= (1 « INT1);
- Kích hoạt ngắt toàn cục bằng lệnh sei();

Trình xử lý ngắt (ISR) cho INT1 được định nghĩa bằng khối ISR(INT1_vect). Trong phiên bản hiện tại, hàm này không thực hiện xử lý cụ thể mà chỉ đóng vai trò duy trì trạng thái sẵn sàng cho việc mở rộng sau này. Tuy nhiên, việc khai báo hàm là cần thiết để tránh lỗi biên dịch và cho phép hệ thống phục hồi đúng trình tự sau khi ngắt kết thúc.

Việc sử dụng ngắt thay vì polling giúp hệ thống tiết kiệm năng lượng khi ở chế độ ngủ và chỉ phản hồi khi có sự kiện thực sự xảy ra, phù hợp với mục tiêu vận hành hiệu quả và ổn định trong môi trường có yêu cầu tiêu thụ điện năng thấp.

4.3.3 Cơ chế tiết kiệm năng lượng trên ATmega8

Trong các hệ thống nhúng hoạt động liên tục hoặc sử dụng nguồn năng lượng hạn chế như pin, việc tối ưu hóa tiêu thụ điện năng đóng vai trò rất quan trọng. Vi điều khiển ATmega8 hỗ trợ nhiều chế độ tiết kiệm năng lượng (power-saving modes) cho phép tạm thời dừng hoặc giới hạn hoạt động của các thành phần bên trong nhằm giảm tiêu thụ dòng. Các chế độ này đặc biệt hiệu quả khi hệ thống không cần xử lý liên tục mà chỉ phản ứng khi có sự kiện từ ngoại vi.

ATmega8 hỗ trợ 4 chế độ tiết kiệm năng lượng chính, được thiết kế để phù hợp với các mức độ yêu cầu khác nhau về hiệu năng và tiêu thụ:

- **Idle Mode:** CPU ngừng hoạt động trong khi các ngoại vi như bộ định thời (Timer), bộ truyền thông USART, bộ chuyển đổi ADC, và ngắt ngoài vẫn hoạt động bình thường. Đây là chế độ tiết kiệm năng lượng nhẹ nhất nhưng cho khả năng phản hồi nhanh nhất.
- **ADC Noise Reduction Mode:** chỉ tạm dừng CPU và hầu hết các ngoại vi, ngoại trừ bộ chuyển đổi ADC. Chế độ này tối ưu cho việc đo ADC chính xác khi không bị nhiễu do xung đồng hồ.
- **Power-down Mode:** toàn bộ hệ thống bị ngắt xung đồng hồ, bao gồm CPU và tất cả các ngoại vi, trừ bộ phát hiện ngắt ngoài và watchdog. Chế độ này tiêu

thụ dòng rất thấp nhưng thời gian khôi phục lâu.

- **Power-save Mode:** tương tự như Power-down, nhưng vẫn giữ hoạt động cho Timer2 (với bộ tạo xung riêng). Phù hợp cho các ứng dụng cần đo thời gian định kỳ với mức tiêu thụ năng lượng thấp.

Trong hệ thống được thiết kế, vi điều khiển không cần thực hiện các tác vụ liên tục mà chỉ hoạt động khi có chuyển động từ cảm biến. Tuy nhiên, để duy trì khả năng phản hồi tức thời thông qua ngắt INT1, vi điều khiển cần duy trì khả năng lắng nghe ngắt ngoài. Trong các chế độ sâu hơn như Power-down hay Power-save, các ngoại vi liên quan sẽ bị vô hiệu hóa, làm giảm khả năng phản hồi hoặc yêu cầu thời gian khôi phục lâu hơn. Do đó, chế độ Idle Mode là lựa chọn phù hợp nhất vì:

- Cho phép hệ thống ngừng hoạt động CPU để tiết kiệm năng lượng trong khi vẫn duy trì chức năng ngắt.
- Khả năng phản hồi nhanh ngay khi có tín hiệu kích hoạt từ cảm biến (qua ngắt INT1).
- Không yêu cầu phải cấu hình lại hệ thống sau khi thức dậy từ chế độ ngủ.

Việc cấu hình và kích hoạt chế độ ngủ trong ATmega8 liên quan đến hai thanh ghi chính:

- **MCUCR (MCU Control Register):** dùng để lựa chọn chế độ ngủ thông qua hai bit SM1 và SM0:
  - SM1 = 0, SM0 = 0: chọn chế độ Idle.
  - SM1 = 0, SM0 = 1: chọn chế độ ADC Noise Reduction.
  - SM1 = 1, SM0 = 0: chọn chế độ Power-down.
  - SM1 = 1, SM0 = 1: chọn chế độ Power-save.
- **SE (Sleep Enable bit):** là bit 7 trong thanh ghi MCUCR. Phải được đặt bằng 1 để cho phép vào chế độ ngủ khi thực hiện lệnh sleep_cpu() hoặc sleep_mode().

Trong đồ án, việc cấu hình chế độ Idle được thực hiện trong hàm sleep_mode_init() như sau:

- Ghi giá trị 0 vào hai bit SM1 và SM0 để chọn chế độ Idle:
  - MCUCR = (1 « SM1);
  - MCUCR = (1 « SM0);
- Bật bit Sleep Enable (SE) để cho phép vào chế độ ngủ: sleep_enable();
- Sau đó, mỗi khi cần chuyển vi điều khiển về chế độ ngủ, chỉ cần gọi lệnh:

```
sleep_mode();
```

Việc sử dụng chế độ Idle giúp hệ thống tiêu thụ ít năng lượng hơn trong trạng thái chờ mà không ảnh hưởng đến khả năng tiếp nhận và xử lý tín hiệu từ cảm biến thông qua cơ chế ngắt ngoài. Đây là giải pháp tối ưu về cả hiệu năng và năng lượng đối với hệ thống hoạt động theo sự kiện như đồ án đang triển khai.

4.3.4 Cơ chế hoạt động của chương trình

Chương trình điều khiển trên vi điều khiển ATmega8 được xây dựng dựa trên mô hình máy trạng thái hữu hạn (Finite State Machine - FSM). Đây là mô hình được sử dụng phổ biến trong các hệ thống nhúng để mô tả hành vi của một hệ thống theo từng trạng thái rời rạc. Ở mỗi thời điểm, hệ thống chỉ ở trong đúng một trạng thái, và trạng thái hiện tại sẽ chuyển sang trạng thái mới dựa trên điều kiện đầu vào. Mỗi trạng thái sẽ có những hành vi và tác vụ riêng biệt.

Trong đồ án, FSM được thiết kế với ba trạng thái chính:

- **STATE_SLEEP:** trạng thái chờ tiết kiệm năng lượng. Hệ thống tạm dừng hoạt động của CPU và đưa vi điều khiển vào chế độ ngủ Idle Mode. Trong trạng thái này, thiết bị Android và module GNSS đều tắt. Hệ thống chỉ có thể được đánh thức bởi tín hiệu từ cảm biến chuyển động (kích hoạt ngắt INT1 tại chân PD3).
- **STATE_ON:** trạng thái hoạt động bình thường. Hệ thống bật nguồn cho thiết bị Android, hiển thị đèn LED và cấp nguồn cho module định vị GNSS. Hệ thống duy trì ở trạng thái này miễn là vẫn phát hiện có chuyển động.
- **STATE_WAIT_OFF:** trạng thái chờ tắt. Khi hệ thống không còn phát hiện chuyển động (tín hiệu từ cảm biến không còn duy trì mức cao), chương trình chuyển sang trạng thái này để đếm thời gian ổn định trước khi tắt thiết bị. Nếu trong khoảng thời gian chờ vẫn không phát hiện lại chuyển động, hệ thống sẽ tắt thiết bị Android và quay trở về trạng thái STATE_SLEEP.

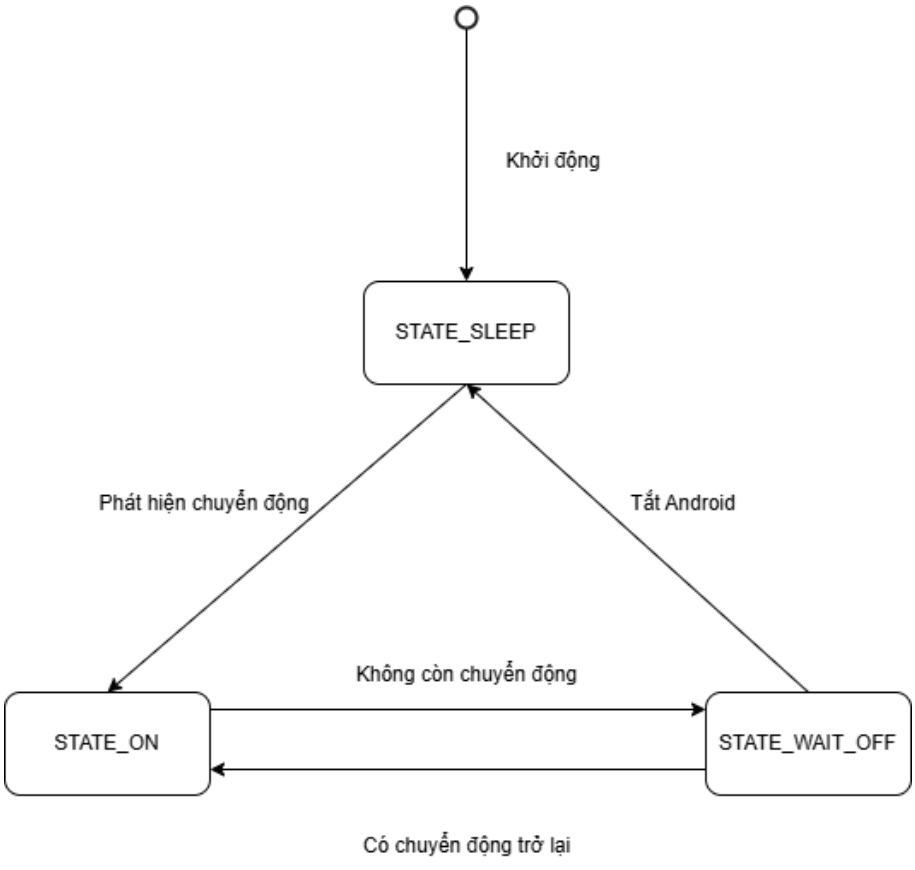
Khi hệ thống khởi động, hàm setup_io() cấu hình các chân GPIO, setup_interrupts() thiết lập ngắt ngoài INT1 và sleep_mode_init() chuẩn bị chế độ ngủ. Sau đó, vi điều khiển đi vào vòng lặp vô hạn và hoạt động theo FSM như sau:

- **Ban đầu**, hệ thống ở trạng thái STATE_SLEEP, CPU được đưa vào chế độ Idle để tiết kiệm năng lượng. Tại đây, vi điều khiển không thực thi mã lệnh nào cho đến khi có ngắt xảy ra.
- **Khi có chuyển động**, cảm biến phát tín hiệu mức cao đến chân PD3, kích hoạt ngắt INT1. Vi điều khiển được đánh thức và kiểm tra tín hiệu tại chân PD3. Nếu tín hiệu là mức cao, hệ thống chuyển sang trạng thái STATE_ON. Trong

trạng thái này, chân PB1 được đưa lên mức cao để cấp nguồn cho module GNSS, chân PC0 được kích để bật thiết bị Android (giữ mức cao trong 3 giây) và chân PC1 được đưa lên mức cao để bật LED trạng thái.

- **Khi không còn chuyển động**, tín hiệu tại PD3 trở lại mức thấp. Hệ thống phát hiện sự thay đổi và chuyển sang trạng thái STATE_WAIT_OFF. Tại đây, chương trình duy trì trạng thái chờ trong 30 giây, kiểm tra định kỳ xem tín hiệu tại PD3 có được kích hoạt trở lại hay không.
- **Nếu không có chuyển động trong thời gian chờ**, chương trình tiến hành tắt thiết bị Android (PC0 lên mức cao trong 10 giây), ngắt nguồn GNSS (PB1 ở mức thấp), tắt LED (PC1 ở mức thấp) và trở về trạng thái STATE_SLEEP.
- **Ngược lại**, nếu trong thời gian chờ có chuyển động xảy ra trở lại, hệ thống quay lại trạng thái STATE_ON và tiếp tục hoạt động.

Mô hình máy trạng thái trong chương trình giúp hệ thống hoạt động rõ ràng, tiết kiệm năng lượng và phản hồi chính xác theo điều kiện môi trường. Bằng cách tách biệt các hành vi tương ứng với từng trạng thái, việc kiểm soát logic và bảo trì chương trình trở nên đơn giản, đồng thời tăng độ tin cậy trong quá trình vận hành thực tế.



Hình 4.3: Sơ đồ mô hình máy trạng thái điều khiển hoạt động vi điều khiển ATmega8

CHƯƠNG 5. TRIỂN KHAI HỆ THỐNG SERVER

Trong hệ thống giám sát vị trí được xây dựng, server đóng vai trò là trung tâm tiếp nhận, xử lý và hiển thị dữ liệu định vị từ thiết bị. Thông tin thu thập từ thiết bị bao gồm tọa độ GPS, tốc độ di chuyển, trạng thái hệ thống và các thông số cấu hình liên quan. Những dữ liệu này được truyền về server theo thời gian thực thông qua giao thức MQTT và được lưu trữ để phục vụ việc giám sát, phân tích và truy xuất sau này.

Nền tảng ThingsBoard được lựa chọn làm giải pháp triển khai server do khả năng hỗ trợ trực quan hóa dữ liệu IoT một cách hiệu quả và dễ cấu hình. Server không chỉ có chức năng hiển thị vị trí hiện tại trên bản đồ, mà còn lưu trữ toàn bộ lịch sử di chuyển, hiển thị trạng thái hoạt động của thiết bị và phát hiện các tình huống vượt ra khỏi vùng giới hạn địa lý (geofence).

Ngoài ra, hệ thống server còn cho phép người dùng cấu hình các thông số từ xa như thời gian gửi dữ liệu, khoảng cách thay đổi tối thiểu để cập nhật vị trí, bật hoặc tắt tính năng geofence và chọn khu vực cần giám sát. Những điều chỉnh này được cập nhật động và gửi ngược trở lại thiết bị thông qua bản tin attributes.

Toàn bộ các chức năng trên được xây dựng nhằm đảm bảo hệ thống hoạt động liên tục, linh hoạt, đồng thời cung cấp giao diện thân thiện cho việc giám sát và quản lý thiết bị từ xa.

5.1 Cài đặt và cấu hình Thingsboard

ThingsBoard là một nền tảng mã nguồn mở chuyên dụng cho các ứng dụng IoT, hỗ trợ thu thập dữ liệu, xử lý, lưu trữ và trực quan hóa thông tin từ các thiết bị đầu cuối. Trong đồ án này, ThingsBoard được sử dụng làm server để tiếp nhận và hiển thị dữ liệu định vị từ thiết bị giám sát.

Server được cài đặt trên máy tính cá nhân chạy hệ điều hành Ubuntu, sử dụng bản Community Edition của ThingsBoard. Việc cài đặt được thực hiện thông qua tập tin .deb. Sau khi cài đặt thành công, ThingsBoard hoạt động như một dịch vụ chạy nền và có thể truy cập thông qua trình duyệt tại địa chỉ mặc định là http://localhost:8080.

Quá trình cấu hình ban đầu bao gồm việc khởi tạo tài khoản quản trị viên, tạo thiết bị mới (device), và gán Access Token cho thiết bị. Token này được sử dụng để xác thực mỗi khi thiết bị gửi dữ liệu lên server qua giao thức MQTT.

Việc cài đặt và cấu hình ThingsBoard đảm bảo hệ thống server có thể hoạt động ổn định, tiếp nhận dữ liệu theo thời gian thực, và phục vụ cho các chức năng giám

sát, hiển thị và điều khiển từ xa. Đây là bước nền tảng quan trọng để xây dựng toàn bộ kiến trúc hệ thống giám sát vị trí.

5.2 Cấu hình NAT để truy cập từ Internet

Mặc định, hệ thống ThingsBoard chỉ có thể truy cập được từ bên trong mạng nội bộ (LAN) nơi server được cài đặt. Tuy nhiên, để thiết bị giám sát có thể gửi dữ liệu từ bất kỳ vị trí nào trên Internet về server, cần thực hiện cấu hình NAT (Network Address Translation) và chuyển tiếp cổng (port forwarding) trên router.

Trước hết, địa chỉ IP nội bộ của máy chủ chạy ThingsBoard cần được gán cố định để tránh thay đổi mỗi khi khởi động lại hệ thống. Việc gán IP tĩnh đã được router tự động thực hiện dựa trên địa chỉ MAC của máy server.

Tiếp theo, cần truy cập vào giao diện cấu hình của router để thực hiện chuyển tiếp các cổng dịch vụ từ bên ngoài vào địa chỉ IP nội bộ của máy chủ. Cụ thể đã mở các cổng sau:

Cổng 8080/TCP: dùng cho giao diện quản trị ThingsBoard qua trình duyệt.

Cổng 1883/TCP: dùng cho giao tiếp MQTT giữa thiết bị và server.

Việc cấu hình NAT và mở cổng thành công cho phép hệ thống giám sát hoạt động liên tục và ổn định từ mọi vị trí có kết nối Internet. Đây là bước quan trọng để hệ thống có thể triển khai thực tế và phục vụ nhu cầu giám sát từ xa.

5.3 Giao tiếp thiết bị với server qua giao thức MQTT

Trong hệ thống được xây dựng, thiết bị giám sát sử dụng giao thức MQTT để truyền dữ liệu về server ThingsBoard. MQTT (Message Queuing Telemetry Transport) là một giao thức truyền thông nhẹ, tối ưu cho các thiết bị IoT có tài nguyên giới hạn và mạng kết nối không ổn định. Giao thức này hỗ trợ cơ chế publish/subscribe, cho phép thiết bị gửi dữ liệu lên server và nhận lại các thông số cấu hình từ xa.

Trên thiết bị, thư viện `MqttAndroidClient` được sử dụng để thiết lập kết nối đến ThingsBoard thông qua địa chỉ broker tại cổng 1883. Xác thực kết nối được thực hiện thông qua chuỗi `Access Token` được cấp khi tạo thiết bị trên ThingsBoard. Chuỗi token này được đặt tại trường `username` khi kết nối MQTT, trong khi `clientId` được sinh tự động theo UUID.

Sau khi kết nối thành công, thiết bị thực hiện hai chức năng chính:

- **Gửi dữ liệu telemetry:** Bao gồm các trường như `latitude`, `longitude`, `speed`, `gps_status`, `distance`, `battery_level` và `is_charging`. Các bản tin này được gửi định kỳ hoặc khi có thay đổi lớn về vị trí hoặc trạng

thái hệ thống.

- **Gửi và nhận thuộc tính (attributes):** Thiết bị gửi các thông tin như `isOutside`, `provinceBoundary`, đồng thời đăng ký nhận bản tin attributes từ ThingsBoard. Khi người dùng thay đổi thông số từ giao diện, chẳng hạn `max_timeout`, `max_distance`, `isGeofenceEnable`, hay `provinces`, ThingsBoard sẽ gửi các thuộc tính này về thiết bị. Thiết bị tiếp nhận và cập nhật lại cấu hình hoạt động tương ứng.

Các bản tin dữ liệu được đóng gói ở định dạng JSON. Ví dụ bản tin telemetry có dạng:

```
{
  "latitude": 21.028511,
  "longitude": 105.804817,
  "speed": 12.5
}
```

Trong khi đó, bản tin attributes có thể như sau:

```
{
  "battery_level": 85,
  "is_charging": false
}
```

Trước khi gửi bản tin, thiết bị luôn kiểm tra trạng thái kết nối. Nếu mất kết nối, hệ thống sẽ tự động thực hiện kết nối lại và gửi bản tin sau khi kết nối thành công. Nếu mạng không khả dụng, dữ liệu sẽ được lưu cục bộ và tự động đồng bộ lại khi có kết nối trở lại.

Việc sử dụng giao thức MQTT giúp hệ thống đạt được độ trễ thấp, độ tin cậy cao và khả năng mở rộng tốt. Đây là nền tảng quan trọng để đảm bảo kết nối liên tục giữa thiết bị và server trong các ứng dụng giám sát IoT thực tế.

5.4 Thiết kế dashboard hiển thị dữ liệu

Dashboard trong ThingsBoard là giao diện trực quan cho phép người dùng theo dõi dữ liệu thu thập từ thiết bị theo thời gian thực. Việc thiết kế dashboard đóng vai trò quan trọng trong việc trình bày thông tin một cách rõ ràng, dễ quan sát và thuận tiện cho việc giám sát từ xa.

Trong hệ thống được triển khai, dashboard được xây dựng để hiển thị các nhóm thông tin chính như sau:

- **Vị trí hiện tại của thiết bị:** Sử dụng widget dạng *Map* để hiển thị tọa độ GPS.

Vị trí của thiết bị được cập nhật liên tục thông qua dữ liệu telemetry gửi từ thiết bị. Marker trên bản đồ di chuyển theo vị trí thực tế, cho phép giám sát hành trình trực tiếp.

- **Lịch sử di chuyển:** Các dữ liệu định vị được lưu trữ tự động và có thể hiển thị lại theo khoảng thời gian tùy chọn. Người dùng có thể chọn khung thời gian cụ thể để truy xuất và quan sát tuyến đường đã đi của thiết bị.
- **Thông số vận hành của thiết bị:** Bao gồm tốc độ di chuyển (`speed`), trạng thái định vị (`gps_status`), mức pin (`battery_level`) và trạng thái sạc (`is_charging`). Các dữ liệu này được hiển thị thông qua các widget như *Gauge*, *Chart*, hoặc *Label*.
- **Trạng thái vùng giám sát (Geofence):** Khi tính năng geofence được bật, hệ thống sẽ kiểm tra xem thiết bị có nằm trong khu vực cho phép hay không. Widget cảnh báo sẽ hiển thị trạng thái `isOutside` bằng màu sắc hoặc thông báo, giúp người dùng nhận biết tức thời khi thiết bị vượt ra khỏi khu vực quy định.
- **Thông tin cấu hình:** Các tham số như `max_timeout`, `max_distance` và `provinces` được hiển thị trong bảng cấu hình. Người dùng có thể thay đổi các giá trị này trực tiếp trên dashboard. Các thay đổi sẽ được ThingsBoard gửi về thiết bị thông qua bản tin *attributes*.

Tất cả các widget được bố trí theo nhóm chức năng, sắp xếp hợp lý nhằm tối ưu hóa khả năng quan sát và thao tác. Dashboard hỗ trợ cập nhật dữ liệu thời gian thực, có khả năng tùy biến về giao diện và tương thích với nhiều kích thước màn hình, bao gồm cả máy tính và thiết bị di động.

Việc thiết kế dashboard đóng vai trò quan trọng trong việc chuyển hóa dữ liệu kỹ thuật sang dạng thông tin trực quan. Điều này giúp người vận hành hệ thống dễ dàng đánh giá tình trạng thiết bị, phát hiện bất thường và đưa ra phản ứng kịp thời. Đây là thành phần then chốt trong việc hiện thực hóa một hệ thống giám sát IoT hiệu quả và thân thiện với người dùng.

5.5 Kết luận chương

Chương này đã trình bày chi tiết quá trình triển khai hệ thống server sử dụng nền tảng ThingsBoard nhằm phục vụ mục tiêu giám sát thiết bị định vị từ xa. Từ việc cài đặt phần mềm trên máy chủ, cấu hình kết nối mạng qua NAT, thiết lập giao tiếp MQTT giữa thiết bị và server, cho đến xây dựng dashboard trực quan và xử lý logic geofence, toàn bộ các bước đều được thực hiện đồng bộ và có tính hệ thống.

Việc sử dụng ThingsBoard giúp đơn giản hóa quá trình tiếp nhận, lưu trữ và

hiển thị dữ liệu IoT. Hệ thống đã cho thấy khả năng hoạt động ổn định trong môi trường mạng thực tế, đồng thời đáp ứng tốt các yêu cầu về cập nhật thời gian thực, giám sát vị trí, phân tích hành vi di chuyển và cảnh báo ra khỏi vùng giám sát.

CHƯƠNG 6. KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

6.1 Kết luận

Hệ thống giám sát vị trí GNSS được phát triển trong khuôn khổ đồ án đã đáp ứng được các mục tiêu kỹ thuật đặt ra ban đầu. Cụ thể, hệ thống có khả năng thu thập dữ liệu GNSS thông qua giao tiếp UART, phân tích dữ liệu từ các bản tin NMEA và UBX, đồng thời truyền thông tin định vị và trạng thái hệ thống lên nền tảng ThingsBoard qua giao thức MQTT. Việc tích hợp thêm các chức năng như cảnh báo geofence, lưu trữ dữ liệu cục bộ, đồng bộ hóa khi có mạng, và cho phép điều khiển từ xa các tham số hoạt động đã góp phần hoàn thiện giải pháp giám sát thời gian thực.

So với các giải pháp giám sát vị trí hiện có, hệ thống trong đồ án có tính linh hoạt cao hơn về mặt cấu hình và triển khai, đồng thời tận dụng được lợi thế của nền tảng mã nguồn mở ThingsBoard để trực quan hóa dữ liệu. Việc sử dụng đồng thời cả dữ liệu GNSS từ thiết bị chuyên dụng và dịch vụ định vị của Android cho phép hệ thống hoạt động ổn định hơn trong các điều kiện khác nhau.

Trong quá trình thực hiện đồ án, sinh viên đã hoàn thành toàn bộ các chức năng cốt lõi của hệ thống như thiết kế mô-đun phần mềm, phân tích bản tin GNSS, xử lý và truyền dữ liệu, cũng như triển khai giao diện hiển thị và giám sát. Tuy nhiên, một số nội dung nâng cao chưa được tích hợp như chức năng phát hiện tín hiệu GNSS giả mạo, tối ưu thuật toán lọc dữ liệu và nâng cao tính bảo mật khi truyền dữ liệu.

Đóng góp nổi bật của đồ án là xây dựng được một hệ thống nhúng có khả năng giám sát hành trình với kiến trúc phần mềm rõ ràng, dễ mở rộng và có khả năng kết nối hiệu quả với nền tảng IoT. Đồng thời, việc xử lý linh hoạt theo từng tỉnh/thành phố và khả năng cập nhật tham số từ xa cũng là những điểm mới nổi bật.

Thông qua quá trình thực hiện, sinh viên đã rút ra nhiều bài học thực tiễn trong việc phát triển hệ thống nhúng tích hợp GNSS và IoT. Cụ thể, sinh viên tích lũy được kinh nghiệm trong việc xử lý dữ liệu thời gian thực, làm việc với giao tiếp phần cứng ở mức thấp, tổ chức phần mềm theo hướng mô-đun hóa và triển khai hệ thống giám sát phân tán qua mạng. Những kinh nghiệm này là nền tảng quan trọng để phát triển các dự án có tính hệ thống và tích hợp cao trong tương lai.

6.2 Hướng phát triển

Để hoàn thiện hơn nữa hệ thống giám sát GNSS đã xây dựng, cần thực hiện một số công việc bổ sung nhằm tăng độ ổn định và tính ứng dụng thực tế. Trước hết,

hệ thống cần được cải tiến về cơ chế lưu trữ và đồng bộ dữ liệu. Việc mở rộng khả năng lưu log định vị chi tiết hơn, kèm theo thời gian và trạng thái hoạt động của thiết bị, sẽ giúp nâng cao hiệu quả phân tích sau này. Ngoài ra, chức năng gửi lại dữ liệu khi mất kết nối nên được thiết kế theo hướng có xác nhận từ server để đảm bảo dữ liệu không bị mất mát.

Một vấn đề khác cần được xử lý là độ chính xác và độ tin cậy của dữ liệu GNSS. Việc tích hợp thêm thuật toán lọc số liệu, chẳng hạn như lọc Kalman, sẽ giúp loại bỏ các nhiễu loạn nhất thời và tăng cường độ ổn định của hệ thống. Song song đó, cần tối ưu hóa quá trình xử lý dữ liệu để giảm độ trễ khi gửi thông tin lên server, nhất là trong các điều kiện mạng yếu hoặc không ổn định.

Về các hướng phát triển mở rộng, một trong những tính năng quan trọng cần nghiên cứu là phát hiện tín hiệu GNSS giả mạo. Hệ thống hiện tại chưa có cơ chế phát hiện các nguồn tín hiệu không hợp lệ, trong khi đây là yếu tố quan trọng trong các ứng dụng liên quan đến an ninh và giám sát. Một hướng tiếp cận khả thi là kết hợp dữ liệu GNSS với các cảm biến quán tính (IMU) để phát hiện sai lệch bất thường, hoặc ứng dụng các phương pháp học máy để phân loại tín hiệu.

Bên cạnh đó, có thể phát triển thêm các chức năng giám sát nâng cao như phân tích hành trình, phát hiện dừng bất thường, hoặc cảnh báo tốc độ vượt ngưỡng. Việc mở rộng hệ thống sang các mô hình quản lý đội phương tiện hoặc logistics cũng là hướng đi tiềm năng, giúp nâng cao giá trị ứng dụng của sản phẩm.

Cuối cùng, về giao diện hiển thị, có thể tích hợp các biểu đồ thống kê, lịch sử di chuyển và bảng cảnh báo trực quan hơn trên ThingsBoard. Việc cung cấp giao diện tương tác cho người dùng cuối sẽ giúp hệ thống dễ sử dụng, dễ giám sát và phù hợp hơn với yêu cầu thực tế trong các bài toán triển khai trên diện rộng.

TÀI LIỆU THAM KHẢO

[1] u-blox AG. *u-blox M9 Receiver Description - Protocol Specification*. Document UBX-13003221, Revision 1.32. u-blox AG, Switzerland, 2021. https://content.u-blox.com/sites/default/files/u-blox-M9-SPG-4.04_InterfaceDescription_UBX-21022436.pdf