

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



BÁO CÁO BÀI TẬP LỚN
IOT VÀ ỨNG DỤNG
ĐỀ TÀI: THIẾT KẾ HỆ THỐNG XE TỰ HÀNH TRÁNH VẬT CẢN VÀ TÌM
VỊ TRÍ CÓ CƯỜNG ĐỘ WIFI MẠNH NHẤT

Giảng viên: Kim Ngọc Bách

Nhóm học phần: 01

Nhóm bài tập lớn: 04

Thành viên nhóm:

B21DCCN150	Nguyễn Bá Anh
B21DCCN318	Mông Thanh Hải
B21DCCN145	Hà Trần Thế Anh
B21DCCN414	Nguyễn Sinh Hùng

Hà Nội, 2025

MỤC LỤC

MỤC LỤC	2
LỜI CẢM ƠN	4
TÓM TẮT	5
CHƯƠNG 1	7
GIỚI THIỆU ĐỀ TÀI	7
1.1 Lý do chọn đề tài	7
1.2 Mục tiêu, phạm vi đề tài	7
1.2.1 Mục tiêu đề tài	7
1.2.2 Phạm vi đề tài	8
1.3 Phương pháp tiếp cận	8
CHƯƠNG 2	10
TỔNG QUAN HỆ THỐNG	10
2.1 Tổng quan hệ thống xe tự hành	10
2.2 Công nghệ sử dụng	11
2.3 Phần cứng và phần mềm sử dụng	11
2.3.1 Linh kiện phần cứng	11
2.3.2 Công cụ phần mềm	12
2.4 Nguyên lý đo cường độ WiFi	12
CHƯƠNG 3	14
XÂY DỰNG HỆ THỐNG	14
3.1 Thiết kế sơ đồ khối hệ thống	14
3.2 Thiết kế sơ đồ mạch hệ thống	15
3.2.1 ESP32	15
3.2.2 Mạch cầu H L298N	16
3.2.3 Động cơ DC	16
3.2.4 Cảm biến siêu âm HC-SR04	17
3.2.5 Servo SG90	17
3.2.6 Màn hình OLED	17
3.2.7 Nguồn điện	18
3.3 Nguyên lý hoạt động	18
CHƯƠNG 4	20
TRIỂN KHAI HỆ THỐNG	20
4.1 Cài đặt phần mềm và lập trình	20
4.1.1 Cài đặt phần mềm trên ESP32	20
4.1.2 Cài đặt ứng dụng giám sát Android	21
4.1.3 Kết nối và đồng bộ dữ liệu	22
4.2 Trình bày thuật toán	22

4.2.1 Thuật toán đo khoảng cách (HC-SR04 + Servo).....	22
4.2.2 Thuật toán xây dựng bản đồ (Map Update).....	24
4.2.3 Thuật toán đo cường độ WiFi.....	26
4.2.4 Thuật toán điều hướng, tránh chướng ngại và tìm đường quay lại.....	27
4.2.5 Thuật toán điều khiển động cơ (Motor Control).....	32
4.2.6 Giao tiếp với ứng dụng Android (Server Communication).....	34
CHƯƠNG 5.....	37
THỬ NGHIỆM VÀ ĐÁNH GIÁ.....	37
5.1 Mục tiêu thử nghiệm.....	37
5.2 Thiết lập thực nghiệm.....	37
5.3 Dữ liệu thu thập và đánh giá kết quả.....	38
5.4 Ưu và nhược điểm.....	42
CHƯƠNG 6.....	44
KẾT LUẬN VÀ ĐỊNH HƯỚNG PHÁT TRIỂN.....	44
6.1 Đánh giá chung.....	44
6.2 Kết quả đạt được.....	44
6.3 Hạn chế.....	45
6.4 Định hướng phát triển.....	45
TÀI LIỆU THAM KHẢO.....	46

LỜI CẢM ƠN

Để hoàn thành tốt đề tài môn học này, ngoài sự nỗ lực của bản thân, chúng em còn nhận được rất nhiều sự giúp đỡ quý báu từ thầy hướng dẫn và các bạn trong nhóm.

Chúng em xin đặc biệt cảm ơn thầy Kim Ngọc Bách đã dành thời gian chỉ bảo, hỗ trợ và đưa ra những góp ý quý báu trong quá trình lựa chọn và thực hiện đề tài **“Thiết kế hệ thống xe tự hành tránh vật cản và tìm vị trí có cường độ WiFi mạnh nhất”**. Những lời hướng dẫn tận tình của thầy đã giúp nhóm có định hướng rõ ràng hơn, tháo gỡ những khó khăn kỹ thuật gặp phải và hoàn thiện bài báo cáo một cách tốt nhất.

Chúng em cũng xin chân thành cảm ơn các thành viên trong nhóm đã phối hợp nhịp nhàng, cùng nhau nỗ lực và chia sẻ trách nhiệm để hoàn thành đề tài đúng tiến độ.

Chúng em tin rằng những kiến thức, kỹ năng và trải nghiệm thu được từ quá trình thực hiện đề tài này sẽ là hành trang quý báu cho mỗi thành viên trên con đường học tập và làm việc sau này.

Chúng em xin trân trọng cảm ơn tất cả những sự giúp đỡ quý báu đó!

TÓM TẮT

Hệ thống xe tự hành được thiết kế để vừa tránh vật cản, vừa liên tục đo và đánh giá cường độ WiFi tại từng điểm di chuyển. Đề tài này tập trung vào việc phát triển thuật toán điều khiển chuyển động thông minh kết hợp với module đo đặc sóng WiFi, giúp tự động hóa công tác khảo sát vùng phủ sóng trong các tòa nhà, khuôn viên hoặc môi trường công nghiệp.

Khi tích hợp vào robot di động, hệ thống cho phép robot phát hiện và né tránh vật cản thông qua cảm biến siêu âm, đồng thời đo cường độ WiFi (RSSI) tại mỗi vị trí di chuyển. Dữ liệu RSSI được ghi nhận lên bản đồ lưới (grid map), từ đó cho thấy phân bố chất lượng kết nối mạng trong không gian khảo sát. Thuật toán điều khiển sẽ kết hợp thông tin về chướng ngại vật và bản đồ tín hiệu để lên lộ trình tự động, vừa đảm bảo an toàn, vừa thu thập dữ liệu WiFi một cách đồng đều và chính xác. Phần cứng sử dụng vi điều khiển ESP32 tích hợp module WiFi và cảm biến siêu âm, kết nối với động cơ, kết hợp thuật toán xử lý tín hiệu và định vị, giúp hệ thống hoạt động linh hoạt trong môi trường phức tạp.

Về cấu trúc, hệ thống tích hợp nhiều thành phần: cảm biến siêu âm HC-SR04 để đo khoảng cách, servo SG90 mở rộng góc quét, động cơ DC điều khiển qua mạch L298N để di chuyển, module WiFi ESP32 để quét tín hiệu và giao tiếp, màn hình OLED hiển thị thông tin, cùng một web server mini chạy trên ESP32 giúp giám sát, gửi lệnh từ xa. Tất cả được điều phối nhờ các thuật toán lập trình nhúng, đảm bảo hệ thống phản hồi chính xác, kịp thời, ổn định trong điều kiện thực tế.

Hệ thống sử dụng bản đồ lưới (grid map) kết hợp thuật toán heuristic và tìm kiếm theo chiều rộng (BFS) để điều hướng, tránh lặp lại đường đã đi và đảm bảo phủ kín toàn bộ khu vực khảo sát. Tại mỗi ô trên lưới, ESP32 tích hợp module WiFi sẽ đo chỉ số RSSI và ghi lại giá trị vào bản đồ tín hiệu, từ đó xây dựng ma trận phân bố độ mạnh–yếu của sóng WiFi. Quá trình thực thi gặp phải một số thách thức: bộ nhớ hạn chế và năng lực xử lý của ESP32 buộc phải tối giản thuật toán và cấu trúc dữ liệu; cảm biến siêu âm dễ bị nhiễu trong môi trường ồn hoặc khi gặp bề mặt hấp thụ sóng, đòi hỏi thêm các bước lọc và đo lặp; kết nối WiFi không ổn định khi robot di chuyển xa router, do đó cần cơ chế tự động ghép nối lại và lưu đệm dữ liệu; đồng thời, việc phối hợp điều khiển động cơ, cảm biến siêu âm, module WiFi và web server trên cùng một vi điều khiển yêu cầu quản lý chặt chẽ tài nguyên để tránh xung đột. Trong thử nghiệm thực tế, robot di chuyển theo lưới trong môi trường giả lập với chướng ngại vật bất ngờ, ghi nhận giá trị RSSI từng ô, phản hồi lệnh từ xa qua web server và xử lý các tình huống lỗi như mất tín hiệu WiFi hay cảm biến không phản hồi.

Kết quả cho thấy hệ thống vận hành ổn định, hoàn thành nhiệm vụ khám phá bản đồ và đo đạc RSSI với độ chính xác chấp nhận được. Đề tài không chỉ giúp nhóm rèn luyện kỹ năng lập trình nhúng và điều khiển robot mà còn tạo nền tảng cho các hướng phát triển tiếp theo, như ứng dụng trí tuệ nhân tạo để tối ưu lộ trình khảo sát, mở rộng thành hệ thống nhiều robot phối hợp hoặc tích hợp giám sát và phân tích dữ liệu qua đám mây.

CHƯƠNG 1

GIỚI THIỆU ĐỀ TÀI

1.1 Lý do chọn đề tài

Trong thời đại phát triển mạnh mẽ của công nghệ tự động hóa và Internet of Things (IoT), các hệ thống nhúng đóng vai trò then chốt trong việc điều khiển và kết nối các thiết bị thông minh. Một trong những ứng dụng tiêu biểu là xe tự hành – một chủ đề thực nghiệm rất tốt để chúng em có thể tiếp cận các kỹ thuật điều khiển, xử lý tín hiệu và tương tác với môi trường thông qua cảm biến

Đề tài "**Thiết kế hệ thống xe tự hành tránh vật cản kết hợp tìm vị trí có cường độ Wifi mạnh nhất**" được lựa chọn vì nó không chỉ giúp nhóm chúng em thực hành lập trình nhúng trên nền tảng vi điều khiển, mà còn tích hợp nhiều kiến thức thú vị khác như cảm biến siêu âm, điều khiển động cơ, thu thập dữ liệu mạng (RSSI), và thuật toán điều hướng

Bên cạnh đó, khả năng đo cường độ Wifi (RSSI) và tự động tìm vị trí có sóng mạnh nhất mang lại ứng dụng thực tiễn rõ rệt như hỗ trợ lắp đặt thiết bị mạng tại vị trí tối ưu, hay phục vụ các hệ thống robot cần kết nối ổn định với trạm chủ trong môi trường rộng

Qua đề tài này, nhóm chúng em mong muốn nâng cao kỹ năng làm việc với phần cứng, phần mềm nhúng, đồng thời phát triển tư duy tích hợp hệ thống và xử lý đa nhiệm trong các ứng dụng IoT thực tế.

1.2 Mục tiêu, phạm vi đề tài

1.2.1 Mục tiêu đề tài

Đề tài hướng tới việc xây dựng hệ thống xe tự hành có khả năng:

- Tự động tránh vật cản trong quá trình di chuyển dựa vào cảm biến siêu âm
- Lập bản đồ tốc độ wifi trong 1 khu vực xác định
- Thực hành thiết kế, lắp ráp hệ thống nhúng hoàn chỉnh, bao gồm cả phần cứng và phần mềm
- Tích hợp các kỹ thuật lập trình nhúng, xử lý tín hiệu cảm biến, điều khiển động cơ và xử lý logic trong thời gian thực

1.2.2 Phạm vi đề tài

Đề tài tập trung triển khai các chức năng trong phạm vi sau:

- Xe hoạt động trên môi trường mô phỏng nhỏ (phòng học, mô hình sàn phẳng) với các vật cản tĩnh đơn giản
- Dùng ESP32 làm bộ xử lý chính, điều khiển động cơ DC qua module L298N
- Sử dụng cảm biến siêu âm HC-SR04 để phát hiện vật cản phía trước
- Đo và ghi nhận giá trị RSSI từ mạng Wifi xung quanh
- Tìm vị trí có tín hiệu mạnh nhất bằng cách di chuyển và ghi nhận Mbps tại nhiều điểm

1.3 Phương pháp tiếp cận

Để thực hiện đề tài, nhóm đã lựa chọn phương pháp tiếp cận theo hướng sau:

1. Tìm hiểu và phân tích yêu cầu đề tài

Nghiên cứu các khái niệm liên quan như xe tự hành, cảm biến tránh vật cản, đo cường độ Wifi (RSSI), cách điều khiển động cơ, và các công nghệ nhúng phù hợp. Từ đó, xác định các chức năng chính của hệ thống và các thành phần cần thiết

2. Lựa chọn phần cứng và công cụ phù hợp

Dựa trên những nghiên cứu, nhóm chúng em đã chọn ESP32 làm bộ xử lý trung tâm do có tích hợp wifi, hỗ trợ đo tốc độ mạng và điều khiển nhiều thiết bị ngoại vi

3. Thiết kế sơ đồ kết nối: xây dựng sơ đồ khối và sơ đồ mạch hệ thống để kết nối ESP32 với các linh kiện khác

4. Xây dựng phần mềm điều khiển

Chương trình được viết trên Arduino IDE, chia thành các module chức năng khác nhau:

- Điều khiển động cơ (tiến, lùi, rẽ trái, phải, dừng)
- Phát hiện và xử lý vật cản
- Đo và ghi nhận giá trị RSSI tại các điểm

- So sánh và xác định vị trí có tín hiệu mạnh nhất
5. Tích hợp hệ thống và chạy thử
 6. Ghi nhận kết quả, đánh giá và đưa ra kết luận

Ghi nhận kết quả đo và hiển thị thông tin RSSI đo được lên màn hình OLED, bản đồ đường đi được hiển thị lên ứng dụng android

Tiến hành đánh giá khả năng hoạt động của xe qua các tiêu chí: khả năng tránh vật cản, độ chính xác khi đo RSSI, hiệu quả trong việc tìm vị trí sóng mạnh, và mức ổn định của hệ thống

CHƯƠNG 2

TỔNG QUAN HỆ THỐNG

2.1 Tổng quan hệ thống xe tự hành

Hệ thống xe tự hành (autonomous vehicle system) là một nhánh nghiên cứu nổi bật trong lĩnh vực robot di động, nơi các thiết bị có khả năng tự di chuyển, tự ra quyết định và tự điều chỉnh hành vi dựa trên dữ liệu cảm biến, hoàn toàn không cần sự can thiệp trực tiếp từ con người. Những hệ thống này mở ra ứng dụng đa dạng trong công nghiệp, nông nghiệp, kho bãi, thăm dò và các môi trường nguy hiểm, nhờ khả năng hoạt động độc lập và thích ứng linh hoạt với môi trường.

Một hệ thống xe tự hành điển hình bao gồm ba thành phần cốt lõi:

- **Cảm biến:** Giúp thu thập thông tin từ môi trường xung quanh như khoảng cách đến vật cản, tốc độ di chuyển, hướng đi, và trong một số ứng dụng nâng cao là tín hiệu mạng hoặc dữ liệu môi trường khác.
- **Bộ xử lý trung tâm:** Là trái tim của hệ thống, nơi thực hiện phân tích dữ liệu từ cảm biến, áp dụng các thuật toán lập trình để ra quyết định điều hướng, tránh vật cản, tối ưu lộ trình di chuyển, hoặc thực hiện các tác vụ đặc thù khác.
- **Hệ thống truyền động:** Bao gồm các cơ cấu cơ khí và động cơ, đảm nhiệm việc hiện thực hóa quyết định của bộ xử lý thông qua các hành động cụ thể như tiến, lùi, rẽ trái, rẽ phải hoặc dừng.

Đề tài “**Thiết kế hệ thống xe tự hành tránh vật cản và tìm vị trí có cường độ WiFi mạnh nhất**” là một phiên bản thu nhỏ tích hợp đầy đủ những thành phần nói trên. Điểm đặc biệt của hệ thống không chỉ dừng lại ở khả năng di chuyển và tránh vật cản bằng cảm biến siêu âm, mà còn mở rộng sang khả năng quét, ghi nhận và phân tích chất lượng tín hiệu WiFi tại từng vị trí. Việc này có ý nghĩa quan trọng trong bối cảnh IoT hiện đại, nơi các thiết bị phải tìm ra vị trí tối ưu để truyền hoặc nhận dữ liệu, đặc biệt ở các khu vực khó tiếp cận hoặc có yêu cầu cao về chất lượng kết nối mạng.

Để xây dựng hệ thống này, ta cần phối hợp kiến thức từ nhiều lĩnh vực: lập trình nhúng trên vi điều khiển (ESP32), điều khiển động cơ qua module cầu H (L298N), xử lý tín hiệu từ cảm biến siêu âm (HC-SR04), giao tiếp mạng WiFi, cũng như thiết kế và triển khai các thuật toán điều hướng, tránh vật cản, và quét tín hiệu mạng. Đây không chỉ là một bài toán kỹ thuật, mà còn là một bài học thực tế về cách tích hợp đa công nghệ

để tạo ra một sản phẩm hoàn chỉnh, phục vụ mục đích giám sát, thăm dò và thu thập dữ liệu tự động.

2.2 Công nghệ sử dụng

Hệ thống được xây dựng dựa trên nhiều công nghệ nổi bật:

- Công nghệ IoT (Internet of Things): Cho phép xe giao tiếp không dây qua mạng WiFi, truyền và nhận dữ liệu giữa thiết bị nhúng (ESP32) và ứng dụng người dùng (app Android).
- Kỹ thuật nhúng (Embedded Systems): Toàn bộ hệ thống được lập trình trên vi điều khiển ESP32 – một nền tảng có tài nguyên hạn chế, đòi hỏi lập trình tối ưu, quản lý bộ nhớ, và xử lý thời gian thực.
- Điều khiển robot di động: Áp dụng thuật toán tìm kiếm theo chiều rộng (BFS), và điều hướng dựa trên cảm biến để quyết định hướng đi an toàn, hiệu quả.
- Xử lý tín hiệu mạng: Đo tín hiệu RSSI, phân tích thời gian phản hồi HTTP, xây dựng bản đồ cường độ tín hiệu tại từng điểm.
- Giao diện người – máy (HMI): Hiển thị dữ liệu qua màn hình OLED, đồng thời cung cấp API HTTP trên app Android mini để người dùng dễ dàng giám sát, gửi lệnh điều khiển từ xa.

Việc kết hợp đồng thời các công nghệ trên giúp hệ thống không chỉ là một thiết bị robot cơ bản mà còn trở thành một nút mạng thông minh, có thể tự động điều chỉnh hành vi để phục vụ các mục tiêu mạng.

2.3 Phần cứng và phần mềm sử dụng

2.3.1 Linh kiện phần cứng

1. ESP32: Đây là vi điều khiển lõi kép tích hợp WiFi và Bluetooth, là trung tâm xử lý chính của hệ thống. ESP32 chịu trách nhiệm đọc dữ liệu cảm biến, điều khiển động cơ, giao tiếp mạng, chạy web server, hiển thị dữ liệu và thực thi thuật toán điều hướng.
2. Cảm biến siêu âm HC-SR04: Dùng để đo khoảng cách đến vật cản, giúp robot phát hiện nguy hiểm và tránh va chạm. Nguyên lý hoạt động dựa trên phát – nhận sóng siêu âm và đo thời gian phản hồi.
3. Servo SG90: Giúp xoay cảm biến siêu âm sang trái, phải, từ đó mở rộng vùng quét. Thay vì lắp nhiều cảm biến, chỉ một cảm biến kết hợp servo đã có thể dò được nhiều hướng.

4. Động cơ DC kết hợp mạch cầu H L298N: Điều khiển bánh xe, giúp xe tiến, lùi, rẽ trái, rẽ phải. Mạch cầu H cho phép thay đổi chiều dòng điện qua động cơ, nhờ đó kiểm soát hướng quay.
5. Màn hình OLED 0.96 inch: Hiển thị trực quan các thông tin như tốc độ mạng, khoảng cách vật cản, trạng thái hệ thống, giúp người dùng giám sát tại chỗ.
6. Pin, khung xe, bánh xe: Cung cấp năng lượng, phần cơ khí giúp hệ thống di chuyển ổn định.

2.3.2 Công cụ phần mềm

- Arduino IDE: Môi trường lập trình chính, biên dịch mã và nạp lên ESP32
- Thư viện phần mềm:
 - Adafruit_GFX và Adafruit_SSD1306: Điều khiển màn hình OLED.
 - ESP32 Servo: Điều khiển servo motor.
 - WiFi, HTTPClient: Kết nối mạng, gửi nhận dữ liệu HTTP.
 - ArduinoJson: Xử lý dữ liệu JSON để giao tiếp với web server.
 - ESPAsyncWebServer, AsyncTCP: Tạo web server trên ESP32.
- Các công cụ hỗ trợ: Serial Monitor (đọc log debug), phần mềm kiểm tra HTTP (Postman, trình duyệt), và các app Android thử nghiệm điều khiển từ xa.

2.4 Nguyên lý đo cường độ WiFi

Trong hệ thống xe tự hành, chất lượng kết nối mạng được đánh giá dựa trên cường độ tín hiệu WiFi (RSSI – Received Signal Strength Indicator). Đây là chỉ số phản ánh mức công suất tín hiệu mà robot nhận được từ bộ phát, đóng vai trò quan trọng trong việc duy trì kết nối ổn định, đặc biệt cần thiết cho các ứng dụng giám sát thời gian thực hoặc truyền tải dữ liệu.

ESP32 tiến hành đo cường độ tín hiệu WiFi thông qua phương pháp quét mạng:

- Robot thực hiện quét các mạng WiFi xung quanh, ghi nhận giá trị RSSI của từng mạng.
- RSSI thường là một giá trị âm (đơn vị dBm); giá trị càng gần 0, tín hiệu càng mạnh; giá trị càng nhỏ (âm sâu hơn), tín hiệu càng yếu. Ví dụ: -40 dBm là tín hiệu mạnh, trong khi -90 dBm là tín hiệu yếu.
- Dữ liệu RSSI tại mỗi vị trí được lưu lại để robot có thể ước tính chất lượng mạng ở từng vùng.

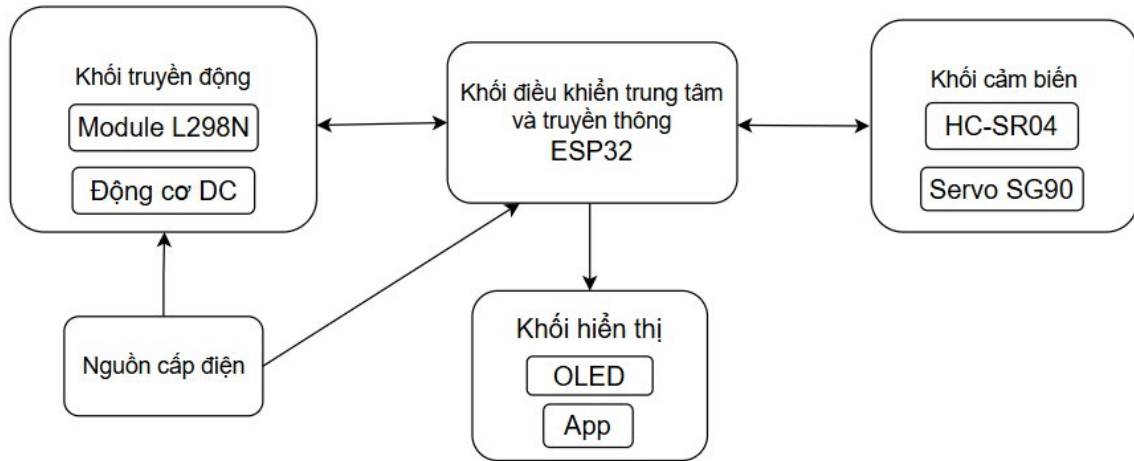
Tất cả các giá trị đo được được lưu trữ vào bản đồ lưới (grid map) gắn với từng ô mà robot đi qua. Bản đồ này giúp xe liên tục cập nhật, so sánh và ghi nhớ vị trí nào có

tín hiệu mạng mạnh nhất. Nếu cần tối ưu kết nối, sẽ ưu tiên di chuyển đến những vị trí đó để đảm bảo hiệu suất hoạt động tốt nhất.

CHƯƠNG 3

XÂY DỰNG HỆ THỐNG

3.1 Thiết kế sơ đồ khối hệ thống



Hệ thống được xây dựng dựa trên mô hình chia thành các khối chức năng chính, phối hợp chặt chẽ để hoàn thành nhiệm vụ di chuyển và thu thập dữ liệu. Mô hình tổng quát bao gồm các thành phần:

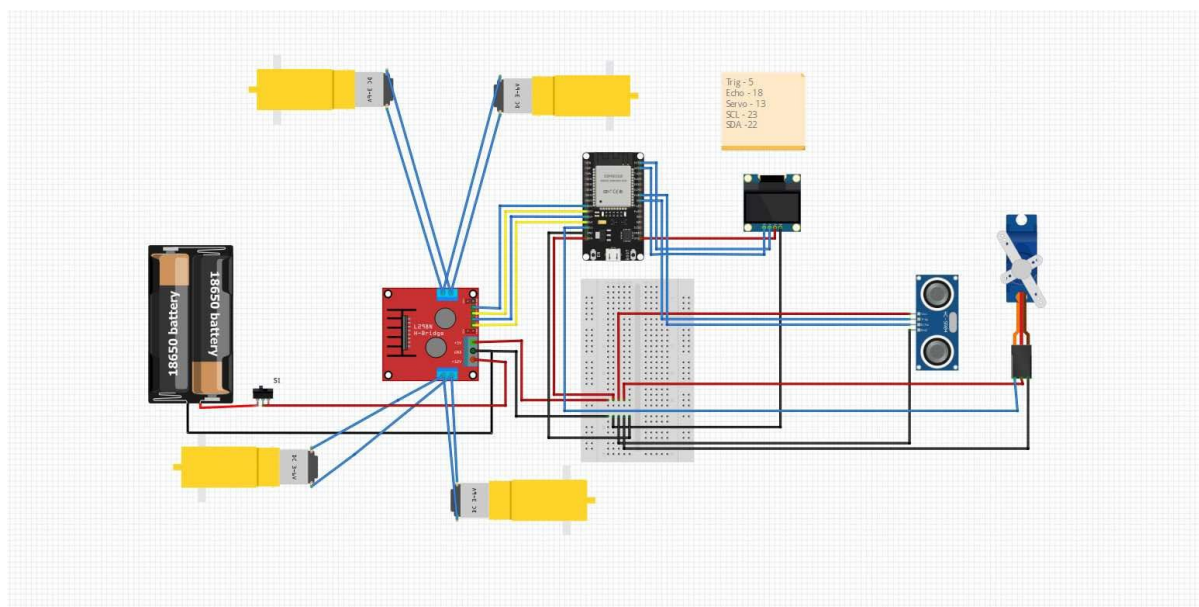
- **Khối cảm biến:** sử dụng cảm biến siêu âm HC-SR04 để đo khoảng cách phía trước, bên trái và bên phải; Servo SG90 quay cảm biến để quét các hướng; từ đó phát hiện chướng ngại vật
- **Khối điều khiển trung tâm và truyền thông:** sử dụng vi điều khiển ESP32 làm bộ não xử lý toàn bộ dữ liệu nhận được từ cảm biến, thực hiện tính toán, ra quyết định di chuyển, điều khiển motor, và giao tiếp WiFi
- **Khối truyền động:** sử dụng motor DC kết hợp module điều khiển cầu H (L298N), nhận tín hiệu điều khiển từ ESP32 để tiến, lùi, rẽ trái hoặc rẽ phải theo quyết định của thuật toán
- **Khối hiển thị:** màn hình OLED 0.96 inch hiển thị thông tin cường độ WiFi, trạng thái hoạt động, hoặc thông báo lỗi khi cần. App android sẽ hiển thị bản đồ đi của xe
- **Nguồn cấp điện (Pin 18650):** Cung cấp năng lượng cho toàn bộ hệ thống, bao gồm vi điều khiển, mạch L298N và động cơ. Pin được kết nối qua công tắc để dễ dàng bật/tắt hệ thống

Về mặt triển khai vật lý, toàn bộ các thành phần phần cứng được lắp ráp gọn gàng trên một khung xe bốn bánh:

- Cảm biến siêu âm được gắn lên giá xoay servo ở đầu xe
- ESP32 được đặt ở vị trí trung tâm xe, kết nối đến mạch cầu H, servo, cảm biến, và màn hình OLED qua các dây dẫn
- Module cầu H nối trực tiếp đến motor DC, nhận tín hiệu điều khiển từ ESP32 và cấp nguồn từ pin
- Hệ thống sử dụng nguồn pin di động hoặc pin 18650 để đảm bảo tính di động, không phụ thuộc vào nguồn điện ngoài

3.2 Thiết kế sơ đồ mạch hệ thống

Hình dưới đây mô tả cách các linh kiện trong hệ thống được kết nối với nhau



Sau đây là mô tả kết nối chi tiết của từng linh kiện

3.2.1 ESP32

Chân ESP32	Thiết bị kết nối	Chức năng
D5	Cảm biến siêu âm HC-SR04	Gửi tín hiệu Trig
D18	Cảm biến siêu âm HC-SR04	Nhận tín hiệu Echo
D13	Servo SG90	Tín hiệu điều khiển servo

D22	Màn hình OLED	Giao tiếp I2C (SDA)
D23	Màn hình OLED	Giao tiếp I2C (SCL)
D14, D25, D26, D27	Mạch cầu H L298N	Điều khiển chiều động cơ
GND	Nối chung với chân GND của các linh kiện	Mass hệ thống
5V / 3,3V	Màn hình OLED, Servo (qua breadboard)	Cấp nguồn

Bảng 1: Chi tiết kết nối của ESP32

3.2.2 Mạch cầu H L298N

Chân L298N	Thiết bị ghép nối	Chức năng
IN1	ESP32 - D25	Điều khiển chiều động cơ 1
IN2	ESP32 - D26	Điều khiển chiều động cơ 2
IN3	ESP32 - D27	Điều khiển chiều động cơ 3
IN4	ESP32 - D14	Điều khiển chiều động cơ 4
OUT1	Động cơ DC 1	Kết nối động cơ DC 1
OUT2	Động cơ DC 2	Kết nối động cơ DC 2
OUT3	Động cơ DC 3	Kết nối động cơ DC 3
OUT4	Động cơ DC 4	Kết nối động cơ DC 4
ENA, ENB	Nối VCC trực tiếp	Kích hoạt module điều khiển
VCC	Pin 18650	Nguồn cấp cho động cơ
GND	ESP32 + nguồn	Mass chung

Bảng 2: Chi tiết kết nối L298N

3.2.3 Động cơ DC

Động cơ	Chân kết nối	Mạch điều khiển
Động cơ 1	Gắn OUT1-OUT2	Mạch L298N

Động cơ 2	Gắn OUT3-OUT4	
Động cơ 3	Gắn OUT1-OUT2	
Động cơ 4	Gắn OUT3-OUT4	

Bảng 3: Chi tiết kết nối 4 động cơ DC

3.2.4 Cảm biến siêu âm HC-SR04

Chân cảm biến	Kết nối	Chức năng
VCC	5V (breadboard)	Cấp nguồn 5V
GND	ESP32 - GND	Mass
Trig	ESP32 - D5	Gửi tín hiệu đo
Echo	ESP32 - D18	Nhận tín hiệu phản xạ

Bảng 4: Chi tiết kết nối cảm biến siêu âm HC-SR04

3.2.5 Servo SG90

Dây Servo	Kết nối	Chức năng
Đỏ	5V	Cấp nguồn
Đen	GND	Mass
Cam	ESP32 - D13	Tín hiệu điều khiển

Bảng 5: Chi tiết kết nối Servo SG90

3.2.6 Màn hình OLED

Chân OLED	Kết nối	Chức năng
VCC	3,3V / 5V	Cấp nguồn
GND	GND	Mass
SDA	ESP32 - D22	Giao tiếp dữ liệu

SCL	ESP32 - D23	Giao tiếp xung nhịp
-----	-------------	---------------------

Bảng 6: Chi tiết kết nối màn hình OLED

3.2.7 Nguồn điện

Thiết bị	Chân kết nối	Chức năng
2 pin 18650	VCC, GND	Cấp nguồn cho mạch L298N và ESP32
Công tắc S1	Nối giữa pin và L298N	Ngắt/mở nguồn chính cho hệ thống

Bảng 7: Các kết nối nguồn điện cho hệ thống

3.3 Nguyên lý hoạt động

Hệ thống xe tự hành được thiết kế để thực hiện hai nhiệm vụ chính: tránh vật cản và tìm vị trí có tốc độ mạng WiFi mạnh nhất. Các chức năng này được thực hiện nhờ sự phối hợp giữa các cảm biến, mô-đun điều khiển và phần mềm nhúng chạy trên vi điều khiển ESP32. Nguyên lý hoạt động của hệ thống như sau:

1. Khởi động hệ thống

Khi cấp nguồn, ESP32 sẽ tiến hành khởi tạo các chân I/O, thiết lập kết nối WiFi và sẵn sàng nhận tín hiệu từ cảm biến và điều khiển động cơ

2. Di chuyển và tránh vật cản

Trong quá trình di chuyển, cảm biến siêu âm HC-SR04 liên tục phát sóng siêu âm và đo thời gian phản hồi để tính khoảng cách đến vật cản phía trước

Nếu phát hiện vật cản ở khoảng cách nhỏ hơn 20cm, xe sẽ dừng lại và chuyển hướng (quay trái, phải, hoặc lùi) cho đến khi không còn vật cản phía trước

3. Đo cường độ tín hiệu WiFi

Tại các vị trí khác nhau (sau mỗi bước di chuyển), ESP32 sẽ quét giá trị cường độ RSSI của một mạng WiFi nhất định và lưu lại giá trị RSSI. Mỗi lần quét RSSI sẽ được liên kết với vị trí tương đối của xe trong quá trình di chuyển

4. So sánh để xác định giá trị RSSI cao nhất

Hệ thống so sánh giá trị RSSI tại từng vị trí với giá trị lớn nhất ghi nhận được, sau đó lưu lại giá trị và vị trí lớn nhất

5. Thông báo kết quả ghi nhận

Khi hoàn tất quá trình đo và so sánh vị trí có giá trị RSSI mạnh nhất sẽ được hiển thị trên màn hình OLED

CHƯƠNG 4

TRIỂN KHAI HỆ THỐNG

4.1 Cài đặt phần mềm và lập trình

4.1.1 Cài đặt phần mềm trên ESP32

Hệ thống sử dụng nhiều thư viện phần mềm như:

1. WiFi.h: Kết nối mạng không dây.
2. ESPAsyncWebServer.h và AsyncTCP.h: Dựng web server bất đồng bộ, phản hồi các yêu cầu GET/POST từ bên ngoài
3. Adafruit_SSD1306.h và Adafruit_GFX.h: Điều khiển màn hình OLED hiển thị dữ liệu trực tiếp trên robot
4. ESP32Servo.h: Điều khiển động cơ servo quay quét cảm biến
5. ArduinoJson.h: Xử lý dữ liệu JSON để truyền tải về app

Các chức năng chính của phần mềm ESP32 gồm:

1. Kết nối mạng WiFi: ESP32 tự động kết nối vào mạng LAN nội bộ bằng tên mạng (SSID) và mật khẩu đã lập trình sẵn
2. Khởi tạo web server: Lắng nghe các yêu cầu HTTP từ bên ngoài trên cổng 80, đặc biệt là các endpoint /data để trả về dữ liệu trạng thái và /command để nhận lệnh điều khiển
3. Điều khiển phần cứng: Gửi tín hiệu tới motor, servo, cảm biến siêu âm HC-SR04 để robot thực hiện di chuyển, tránh vật cản, quét xung quanh
4. Quản lý bản đồ lưới 10x10: Cập nhật trạng thái từng ô (đã thăm, vật cản), đo tốc độ mạng tại mỗi vị trí và lưu lại vị trí có tín hiệu mạnh nhất
5. Đo tốc độ WiFi: Gửi các request thử (ping) tới địa chỉ test để ước lượng tốc độ phản hồi mạng
6. Hiển thị thông tin: Dữ liệu quan trọng như tốc độ mạng, khoảng cách đo được, trạng thái robot sẽ hiển thị trên màn hình OLED gắn trực tiếp lên xe

Quy trình cài đặt:

- Viết mã và biên dịch trên Arduino IDE
- Kết nối ESP32 với máy tính qua cáp USB, nạp chương trình bằng cổng COM
- Dùng Serial Monitor kiểm tra log khởi động, đặc biệt là địa chỉ IP được cấp
- Truy cập `http://<IP>:80/data` trên trình duyệt để test dữ liệu JSON xuất ra
- Nếu cần, có thể debug bằng cách ghi log chi tiết ra cổng Serial

4.1.2 Cài đặt ứng dụng giám sát Android

Ứng dụng giám sát được phát triển bằng ngôn ngữ Kotlin trên nền tảng Android Studio, sử dụng kiến trúc hiện đại với ViewModel, LiveData và Coroutine để đảm bảo luồng dữ liệu mượt mà, không gây đơ ứng dụng. Ngoài ra, thư viện Volley được dùng để gửi và nhận các request HTTP từ ESP32.

- Các chức năng chính của app Android gồm:
 - + Lấy dữ liệu trạng thái: Gửi yêu cầu GET tới ESP32 (theo IP được thiết lập trong mã) để nhận về dữ liệu JSON chứa thông tin vị trí, hướng, tốc độ WiFi, bản đồ trạng thái.
 - + Hiển thị trực quan:
 - Vẽ bản đồ lưới 10x10 bằng custom view (MapView).
 - Tô màu các ô (trắng: chưa thăm, xanh: đã thăm, đỏ: vật cản).
 - Đánh dấu vị trí robot bằng hình tròn và vẽ mũi tên hướng di chuyển.
 - Hiển thị bảng tốc độ mạng: RecyclerView hiển thị tốc độ mạng tại từng ô, tự động highlight ô có tốc độ nhanh nhất.
 - Bật/tắt chế độ cập nhật tự động (polling): Dùng SwitchMaterial để cho phép người dùng bật/tắt việc lấy dữ liệu liên tục từ ESP32.
 - Xử lý lỗi: Hiển thị thông báo trạng thái kết nối, lỗi mạng, hoặc lỗi phân tích dữ liệu JSON.
 - Tự động lưu bản đồ cuối cùng: Khi robot báo đã hoàn thành khám phá, app tự lưu bản đồ cuối cùng để hiển thị cố định, tránh bị thay đổi sau đó.
- Quy trình cài đặt:
 - Viết mã và build app trên Android Studio.
 - Cài file APK lên điện thoại (hoặc chạy trực tiếp qua cáp debug).

- Trong mã, chỉnh serverUrl đúng địa chỉ IP của ESP32.
- Đảm bảo cả điện thoại và ESP32 cùng kết nối mạng LAN (cùng router).
- Khởi chạy app, quan sát dữ liệu hiển thị trực tiếp trên giao diện.

4.1.3 Kết nối và đồng bộ dữ liệu

- Để đảm bảo hai phần mềm giao tiếp chính xác, cần lưu ý:
 - Địa chỉ IP: Luôn kiểm tra IP cấp phát cho ESP32, vì IP nội bộ có thể thay đổi theo router. Có thể set IP tĩnh để dễ truy cập.
 - Cổng mạng: Mặc định ESP32 mở server ở cổng 80, cần đảm bảo router không chặn.
 - Định dạng JSON: App Android dựa vào các trường cụ thể (robotX, robotY, wifiSpeed, grid...) nên cần chắc chắn ESP32 trả về đúng định dạng, tránh lỗi phân tích JSON.
 - Bảo mật: Nếu triển khai thực tế, có thể thêm lớp xác thực (token, API key) hoặc dùng giao thức HTTPS thay vì HTTP để tránh bị can thiệp dữ liệu.

4.2 Trình bày thuật toán

4.2.1 Thuật toán đo khoảng cách (HC-SR04 + Servo)

Thuật toán sử dụng cảm biến siêu âm HC-SR04 để đo khoảng cách đến các chướng ngại vật ở ba hướng (phía trước, bên trái, bên phải). Cảm biến hoạt động bằng cách gửi một xung siêu âm từ chân trigPin và đo thời gian phản hồi qua chân echoPin. Khoảng cách được tính dựa trên thời gian truyền sóng siêu âm (tốc độ âm thanh khoảng 340 m/s). Để đảm bảo độ chính xác, thuật toán thử đo tối đa MAX_DISTANCE_ATTEMPTS (5 lần). Nếu không nhận được tín hiệu hợp lệ, trả về giá trị mặc định 510 cm (tương đương khoảng cách tối đa ngoài tầm đo).

Các bước thực hiện:

1. Đặt chân trigPin ở mức LOW trong 2 micro giây để đảm bảo trạng thái ban đầu ổn định.
2. Gửi xung HIGH trong 10 micro giây để kích hoạt cảm biến.
3. Đặt lại trigPin về LOW và đo thời gian phản hồi bằng hàm pulseIn(echoPin, HIGH, 30000) (timeout 30ms).
4. Tính khoảng cách: $\text{distance} = \text{duration} * 0.034 / 2$ (cm).
5. Lặp lại tối đa 5 lần nếu kết quả không hợp lệ ($\text{duration} == 0$ hoặc $\text{distance} < 0$).

6. Nếu không đo được, trả về 510 cm và ghi log lỗi.

```
680 float measureDistance() {
681     long duration = 0;
682     float distance_cm = 0;
683     for (int i = 0; i < MAX_DISTANCE_ATTEMPTS; i++) {
684         digitalWrite(trigPin, LOW);
685         delayMicroseconds(2);
686         digitalWrite(trigPin, HIGH);
687         delayMicroseconds(10);
688         digitalWrite(trigPin, LOW);
689         duration = pulseIn(echoPin, HIGH, 30000);
690         distance_cm = duration * 0.034 / 2;
691         Serial.print("Attempt ");
692         Serial.print(i + 1);
693         Serial.print(": Duration = ");
694         Serial.print(duration);
695         Serial.print(" us, Distance = ");
696         Serial.print(distance_cm);
697         Serial.println(" cm");
698         if (duration != 0 && distance_cm >= 0) {
699             break;
700         }
701         delay(50);
702     }
703     if (duration == 0 || distance_cm < 0) {
704         Serial.println("HC-SR04: No valid echo received!");
705         return 510;
706     }
707     return distance_cm;
708 }
```

Trong loop(), chương trình thực hiện đo khoảng cách ở ba góc (0°, 90°, 180°) bằng cách điều khiển servo để xoay cảm biến:

```
550 int angles[] = {0, 90, 180};
551 for (int i = 0; i < 3; i++) {
552     int pos = angles[i];
553     if (servo.attached()) {
554         servo.write(pos);
555         delay(500);
556     }
557
558     float distance_cm = measureDistance();
559     if (pos == 0) distanceRight = distance_cm;
560     else if (pos == 90) distanceFront = distance_cm;
561     else distanceLeft = distance_cm;
562
563     Serial.print("Angle: ");
564     Serial.print(pos);
565     Serial.print(" Distance: ");
566     Serial.print(distance_cm);
567     Serial.println(" cm");
568     delay(300);
569 }
570 if (servo.attached()) {
571     servo.detach();
572 }
```

4.2.2 Thuật toán xây dựng bản đồ (Map Update)

Thuật toán xây dựng bản đồ sử dụng một lưới 10x10 (grid) để lưu trạng thái các ô:

- 0: Ô chưa thăm.
- 1: Ô đã thăm.
- 2: Ô có chướng ngại vật.
- 3: Ô hiện tại của robot.

Dựa trên dữ liệu từ cảm biến HC-SR04, các ô xung quanh robot được cập nhật trạng thái. Nếu khoảng cách đo được nhỏ hơn TARGET_DISTANCE (20 cm), ô tương ứng được đánh dấu là chướng ngại vật (2). Khi robot di chuyển, ô cũ được cập nhật thành đã thăm (1), và ô mới được đánh dấu là vị trí robot (3).

Các bước thực hiện:

1. Xác định tọa độ các ô phía trước (frontX, frontY), bên trái (leftX, leftY), và bên phải (rightX, rightY) dựa trên hướng hiện tại của robot (robotHeading: 0° - Bắc, 90° - Đông, 180° - Nam, 270° - Tây).
2. Kiểm tra khoảng cách đo được (distanceFront, distanceLeft, distanceRight). Nếu nhỏ hơn hoặc bằng TARGET_DISTANCE và ô nằm trong lưới, đánh dấu ô đó là chướng ngại vật (2) nếu trạng thái hiện tại là 0 (chưa thăm).
3. Khi robot di chuyển, cập nhật vị trí robot (robotX, robotY) và lưới: ô cũ thành 1, ô mới thành 3.

Các hàm xử lý trong code:

- Hàm cập nhật lưới

```

710 void updateGrid() {
711     int frontX = robotX, frontY = robotY;
712     int leftX = robotX, leftY = robotY;
713     int rightX = robotX, rightY = robotY;
714
715     if (robotHeading == 0) {
716         frontY--; rightX++; leftX--;
717     } else if (robotHeading == 90) {
718         frontX++; rightY++; leftY--;
719     } else if (robotHeading == 180) {
720         frontY++; rightX--; leftX++;
721     } else if (robotHeading == 270) {
722         frontX--; rightY--; leftY++;
723     }
724
725     if (distanceFront >= 0 && distanceFront <= TARGET_DISTANCE && frontX >= 0 && frontX < GRID_SIZE && frontY >= 0 &&
726         frontY < GRID_SIZE && grid[frontX][frontY] == 0) {
727         grid[frontX][frontY] = 2;
728     }
729     if (distanceRight >= 0 && distanceRight <= TARGET_DISTANCE && rightX >= 0 && rightX < GRID_SIZE && rightY >= 0 &&
730         rightY < GRID_SIZE && grid[rightX][rightY] == 0) {
731         grid[rightX][rightY] = 2;
732     }
733     if (distanceLeft >= 0 && distanceLeft <= TARGET_DISTANCE && leftX >= 0 && leftX < GRID_SIZE && leftY >= 0 && leftY <
734         GRID_SIZE && grid[leftX][leftY] == 0) {
735         grid[leftX][leftY] = 2;
736     }
737 }

```

- Hàm cập nhật vị trí robot

```

900 void updatePosition(String dir) {
901     int newX = robotX, newY = robotY;
902     int oldX = robotX, oldY = robotY; // Lưu vị trí cũ
903
904     if (dir == "F") {
905         if (robotHeading == 0) newY--; // Bắc
906         else if (robotHeading == 90) newX++; // Đông
907         else if (robotHeading == 180) newY++; // Nam
908         else if (robotHeading == 270) newX--; // Tây
909     } else if (dir == "L") {
910         robotHeading = (robotHeading + 270) % 360; // Quay trái
911     } else if (dir == "R") {
912         robotHeading = (robotHeading + 90) % 360; // Quay phải
913     }
914
915     if (dir == "F" && newX >= 0 && newX < GRID_SIZE && newY >= 0 && newY < GRID_SIZE && grid[newX][newY] != 2) {
916         robotX = newX;
917         robotY = newY;
918         // Cập nhật lưới: ô cũ thành 1, ô mới thành 3
919         if (oldX != newX || oldY != newY) {
920             if (grid[oldX][oldY] != 2) grid[oldX][oldY] = 1;
921             if (grid[robotX][robotY] != 2) grid[robotX][robotY] = 3;
922         }
923     } else if (dir == "F") {
924         Serial.println("Invalid move forward! Staying in place.");
925         // Không di chuyển, giữ nguyên vị trí
926     }
927 }

```

4.2.3 Thuật toán đo cường độ WiFi

Thuật toán đo cường độ tín hiệu WiFi (RSSI) để xây dựng bản đồ WiFi (wifiMap) trên lưới 10x10, lưu giá trị RSSI (dBm) tại mỗi ô. RSSI được đo bằng hàm WiFi.RSSI(). Giá trị RSSI cao nhất (maxWifiSpeed) và tọa độ tương ứng (maxWifiSpeedX, maxWifiSpeedY) được theo dõi để xác định vị trí có tín hiệu mạnh nhất. Thuật toán cũng xử lý lỗi kết nối WiFi bằng cách thử kết nối lại nếu mất tín hiệu.

Các bước thực hiện:

1. Kiểm tra trạng thái kết nối WiFi (wifiConnected và WiFi.status()).
2. Nếu mất kết nối, thử kết nối lại trong tối đa 10 giây (WiFi.reconnect()).
3. Đo RSSI bằng WiFi.RSSI() và lưu vào wifiMap[robotX][robotY].
4. Nếu RSSI hiện tại lớn hơn maxWifiSpeed, cập nhật giá trị tối đa và tọa độ tương ứng.
5. Hiển thị RSSI trên màn hình OLED (nếu khởi tạo thành công) mỗi wifiCheckInterval (5 giây).

Các hàm xử lý trong code:

- Hàm đo RSSI của Wifi

```
650 float measureWifiSpeed() {
651     if (!wifiConnected) {
652         Serial.println("WiFi not available, skipping RSSI measurement.");
653         return -100;
654     }
655
656     Serial.println("Checking WiFi status...");
657     if (WiFi.status() != WL_CONNECTED) {
658         Serial.println("WiFi not connected, reconnecting...");
659         WiFi.reconnect();
660         unsigned long startTime = millis();
661         while (WiFi.status() != WL_CONNECTED && millis() - startTime < 10000) {
662             delay(500);
663             Serial.print(".");
664         }
665         if (WiFi.status() != WL_CONNECTED) {
666             Serial.println("\nWiFi reconnection failed.");
667             wifiConnected = false;
668             return -100;
669         }
670         Serial.println("\nWiFi reconnected.");
671     }
672
673     long rssi = WiFi.RSSI();
674     return (float)rssi;
675 }
```

- Hàm cập nhật bản đồ Wifi và hiển thị

```

491 if (wifiConnected && currentTime - lastWifiCheckTime >= 10000) {
492     if (WiFi.status() != WL_CONNECTED) {
493         Serial.println("WiFi disconnected, attempting reconnect...");
494         wifiConnected = false;
495         WiFi.reconnect();
496         unsigned long startTime = millis();
497         while (WiFi.status() != WL_CONNECTED && millis() - startTime < 10000) {
498             delay(500);
499         }
500         if (WiFi.status() == WL_CONNECTED) {
501             wifiConnected = true;
502             Serial.println("WiFi reconnected. IP: " + WiFi.localIP().toString());
503         }
504     }
505     lastWifiCheckTime = currentTime;
506 }
507

```

```

508 if (currentTime - lastWifiDisplayTime >= wifiCheckInterval) {
509     wifiSpeed = measureWifiSpeed();
510     if (robotX >= 0 && robotX < GRID_SIZE && robotY >= 0 && robotY < GRID_SIZE) {
511         wifiMap[robotX][robotY] = wifiSpeed;
512         if (wifiSpeed > maxWifiSpeed) {
513             maxWifiSpeed = wifiSpeed;
514             maxWifiSpeedX = robotX;
515             maxWifiSpeedY = robotY;
516         }
517     }
518     if (isDisplayInitialized) {
519         display.clearDisplay();
520         display.setTextSize(1);
521         display.setTextColor(SSD1306_WHITE);
522         display.setCursor(0, 0);
523         display.println("WiFi RSSI:");
524         display.setCursor(0, 8);
525         display.print("RSSI: ");
526         display.print(wifiSpeed);
527         display.println(" dBm");
528         display.display();
529     }
530     Serial.println("Displaying WiFi RSSI on OLED");
531     lastWifiDisplayTime = currentTime;
532 }

```

4.2.4 Thuật toán điều hướng, tránh chướng ngại và tìm đường quay lại

Thuật toán điều hướng sử dụng chiến lược ưu tiên thăm các ô chưa thăm (0) trên lưới, sau đó là các ô đã thăm (1) có lân cận chưa thăm, và cuối cùng là các ô đã thăm bất kỳ. Robot tránh chướng ngại vật bằng cách kiểm tra khoảng cách từ cảm biến HC-SR04. Nếu không có đường đi hợp lệ, robot sử dụng thuật toán BFS (Breadth-First Search) để quay lại các vị trí đã lưu trong hàng đợi (posQueue). Thuật toán cũng giới hạn số lần quay liên tiếp (MAX_CONSECUTIVE_TURNS) để tránh vòng lặp vô hạn.

Các bước thực hiện:

1. Điều hướng và tránh chướng ngại vật:
 - Kiểm tra khoảng cách (distanceFront, distanceLeft, distanceRight) so với TARGET_DISTANCE (20 cm).
 - Xác định các ô phía trước, trái, phải dựa trên hướng robot (robotHeading).
 - Ưu tiên di chuyển đến ô chưa thăm (0) nếu khoảng cách lớn hơn TARGET_DISTANCE và ô nằm trong lưới.
 - Nếu không có ô chưa thăm, di chuyển đến ô đã thăm (1) có lân cận chưa thăm (kiểm tra bằng hasUnvisitedNeighbor).
 - Nếu không có lựa chọn trên, di chuyển đến ô đã thăm bất kỳ (1).
 - Lưu vị trí hiện tại vào posQueue trước khi di chuyển.
2. Tìm đường quay lại:
 - Nếu không có đường đi hợp lệ gần robot (hasNearbyValidCell trả về false) và posQueue không rỗng, lấy vị trí đầu tiên từ hàng đợi.
 - Tính toán hành động (F, L, R) để robot quay về vị trí trong hàng đợi bằng cách so sánh hướng hiện tại (robotHeading) với hướng cần thiết.
3. Kiểm tra hoàn thành:
 - Dừng nếu không còn ô chưa thăm (hasUnvisitedCells) hoặc không có đường đi hợp lệ (hasValidPath) và không tìm được ô đã thăm có lân cận chưa thăm.
 - Gọi exportData và resetMap khi hoàn thành.

Các hàm xử lý trong code:

- Quyết định hướng đi

```

757 String decideDirection() {
758     bool frontValid = (distanceFront > TARGET_DISTANCE);
759     bool rightValid = (distanceRight > TARGET_DISTANCE);
760     bool leftValid = (distanceLeft > TARGET_DISTANCE);
761
762     int frontX = robotX, frontY = robotY;
763     int rightX = robotX, rightY = robotY;
764     int leftX = robotX, leftY = robotY;
765
766     if (robotHeading == 0) {
767         frontY--; rightX++; leftX--;
768     } else if (robotHeading == 90) {
769         frontX++; rightY++; leftY--;
770     } else if (robotHeading == 180) {
771         frontY++; rightX--; leftX++;
772     } else if (robotHeading == 270) {
773         frontX--; rightY--; leftY++;
774     }
775
776     bool frontFreeUnvisited = frontValid && frontX >= 0 && frontX < GRID_SIZE && frontY >= 0 && frontY < GRID_SIZE && grid
[frontX][frontY] == 0;
777     bool rightFreeUnvisited = rightValid && rightX >= 0 && rightX < GRID_SIZE && rightY >= 0 && rightY < GRID_SIZE && grid
[rightX][rightY] == 0;
778     bool leftFreeUnvisited = leftValid && leftX >= 0 && leftX < GRID_SIZE && leftY >= 0 && leftY < GRID_SIZE && grid
[leftX][leftY] == 0;
779

```

```

780     if (frontFreeUnvisited) {
781         posQueue.push({robotX, robotY, robotHeading});
782         consecutiveTurns = 0; // Đặt lại khi di chuyển hợp lệ
783         return "F";
784     } else if (rightFreeUnvisited) {
785         posQueue.push({robotX, robotY, robotHeading});
786         return "R";
787     } else if (leftFreeUnvisited) {
788         posQueue.push({robotX, robotY, robotHeading});
789         return "L";
790     }
791
792     // Kiểm tra ô đã thăm có lân cận chưa thăm
793     bool frontFreeVisited = frontValid && frontX >= 0 && frontX < GRID_SIZE && frontY >= 0 && frontY < GRID_SIZE && grid
[frontX][frontY] == 1 && hasUnvisitedNeighbor(frontX, frontY);
794     bool rightFreeVisited = rightValid && rightX >= 0 && rightX < GRID_SIZE && rightY >= 0 && rightY < GRID_SIZE && grid
[rightX][rightY] == 1 && hasUnvisitedNeighbor(rightX, rightY);
795     bool leftFreeVisited = leftValid && leftX >= 0 && leftX < GRID_SIZE && leftY >= 0 && leftY < GRID_SIZE && grid[leftX]
[leftY] == 1 && hasUnvisitedNeighbor(leftX, leftY);

```

```

797     if (frontFreeVisited) {
798         posQueue.push({robotX, robotY, robotHeading});
799         consecutiveTurns = 0; // Đặt lại khi di chuyển hợp lệ
800         return "F";
801     } else if (rightFreeVisited) {
802         posQueue.push({robotX, robotY, robotHeading});
803         return "R";
804     } else if (leftFreeVisited) {
805         posQueue.push({robotX, robotY, robotHeading});
806         return "L";
807     }
808
809     // Kiểm tra ô đã thăm bất kỳ
810     bool frontFree = frontValid && frontX >= 0 && frontX < GRID_SIZE && frontY >= 0 && frontY < GRID_SIZE && grid[frontX]
[frontY] == 1;
811     bool rightFree = rightValid && rightX >= 0 && rightX < GRID_SIZE && rightY >= 0 && rightY < GRID_SIZE && grid[rightX]
[rightY] == 1;
812     bool leftFree = leftValid && leftX >= 0 && leftX < GRID_SIZE && leftY >= 0 && leftY < GRID_SIZE && grid[leftX][leftY]
== 1;

```



```

813
814     if (frontFree) {
815         posQueue.push({robotX, robotY, robotHeading});
816         consecutiveTurns = 0; // Đặt lại khi di chuyển hợp lệ
817         return "F";
818     } else if (rightFree) {
819         posQueue.push({robotX, robotY, robotHeading});
820         return "R";
821     } else if (leftFree) {
822         posQueue.push({robotX, robotY, robotHeading});
823         return "L";
824     }
825
826     // Quay lại nếu không có đường đi gần
827     if (!hasNearbyValidCell() && !posQueue.empty()) {
828         Position target = posQueue.front();
829         String action = calculateBacktrackAction(target);
830         if (action == "F") consecutiveTurns = 0; // Đặt lại khi di chuyển hợp lệ
831         return action;
832     }
833

```

```

834     // Tìm ô đã thăm gần nhất hoặc ô có lân cận chưa thăm
835     int targetX, targetY;
836     if (findNearestVisitedWithUnvisited(targetX, targetY)) {
837         if (robotHeading == 0 && targetY < robotY) return "F";
838         if (robotHeading == 90 && targetX > robotX) return "F";
839         if (robotHeading == 180 && targetY > robotY) return "F";
840         if (robotHeading == 270 && targetX < robotX) return "F";
841         if (robotHeading == 0 && targetX > robotX) return "R";
842         if (robotHeading == 90 && targetY > robotY) return "R";
843         if (robotHeading == 180 && targetX < robotX) return "R";
844         if (robotHeading == 270 && targetY < robotY) return "R";
845         if (robotHeading == 0 && targetX < robotX) return "L";
846         if (robotHeading == 90 && targetY < robotY) return "L";
847         if (robotHeading == 180 && targetX > robotX) return "L";
848         if (robotHeading == 270 && targetY > robotY) return "L";
849     }

```

```

850
851     if (findNearestVisited(targetX, targetY)) {
852         if (robotHeading == 0 && targetY < robotY) return "F";
853         if (robotHeading == 90 && targetX > robotX) return "F";
854         if (robotHeading == 180 && targetY > robotY) return "F";
855         if (robotHeading == 270 && targetX < robotX) return "F";
856         if (robotHeading == 0 && targetX > robotX) return "R";
857         if (robotHeading == 90 && targetY > robotY) return "R";
858         if (robotHeading == 180 && targetX < robotX) return "R";
859         if (robotHeading == 270 && targetY < robotY) return "R";
860         if (robotHeading == 0 && targetX < robotX) return "L";
861         if (robotHeading == 90 && targetY < robotY) return "L";
862         if (robotHeading == 180 && targetX > robotX) return "L";
863         if (robotHeading == 270 && targetY > robotY) return "L";
864     }
865
866     return "R"; // Quay phải để quét lại nếu không có lựa chọn nào khác
867 }

```

- Tìm ô đã thăm có lân cận chưa thăm

```

397 bool findNearestVisitedWithUnvisited(int &targetX, int &targetY) {
398     bool visited[GRID_SIZE][GRID_SIZE] = {false};
399     std::queue<std::pair<int, int>> q;
400     q.push({robotX, robotY});
401     visited[robotX][robotY] = true;
402
403     int directions[4][2] = {{0, -1}, {1, 0}, {0, 1}, {-1, 0}}; // Bắc, Đông, Nam, Tây

```

```

405 while (!q.empty()) {
406     int x = q.front().first;
407     int y = q.front().second;
408     q.pop();
409
410     if (grid[x][y] == 1 && hasUnvisitedNeighbor(x, y)) {
411         targetX = x;
412         targetY = y;
413         return true;
414     }
415
416     for (int i = 0; i < 4; i++) {
417         int nx = x + directions[i][0];
418         int ny = y + directions[i][1];
419         if (nx >= 0 && nx < GRID_SIZE && ny >= 0 && ny < GRID_SIZE && !visited[nx][ny] && (grid[nx][ny] == 1 || grid[nx][ny] == 0)) {
420             q.push({nx, ny});
421             visited[nx][ny] = true;
422         }
423     }
424 }
425 return false;
426 }

```

- Tìm ô đã thăm gần nhất

```

434 bool findNearestVisited(int &targetX, int &targetY) {
435     bool visited[GRID_SIZE][GRID_SIZE] = {false};
436     std::queue<std::pair<int, int>> q;
437     q.push({robotX, robotY});
438     visited[robotX][robotY] = true;
439
440     int directions[4][2] = {{0, -1}, {1, 0}, {0, 1}, {-1, 0}}; // Bắc, Đông, Nam, Tây

```

```

442 while (!q.empty()) {
443     int x = q.front().first;
444     int y = q.front().second;
445     q.pop();
446
447     if (grid[x][y] == 1 && (x != robotX || y != robotY)) {
448         targetX = x;
449         targetY = y;
450         Serial.print("Found nearest visited cell at (");
451         Serial.print(targetX);
452         Serial.print(", ");
453         Serial.print(targetY);
454         Serial.println(")");
455         return true;
456     }
457
458     for (int i = 0; i < 4; i++) {
459         int nx = x + directions[i][0];
460         int ny = y + directions[i][1];
461         if (nx >= 0 && nx < GRID_SIZE && ny >= 0 && ny < GRID_SIZE && !visited[nx][ny] && (grid[nx][ny] == 1 || grid[nx][ny] == 0)) {
462             q.push({nx, ny});
463             visited[nx][ny] = true;
464         }
465     }
466 }
467 return false;
468 }

```

- Quay lại vị trí trong hàng đợi

```

862 String calculateBacktrackAction(Position target) {
863     // Tính toán hướng cần quay để đối mặt với vị trí mục tiêu
864     int dx = target.x - robotX;
865     int dy = target.y - robotY;
866
867     // Xác định hướng mục tiêu dựa trên vị trí tương đối
868     int targetHeading = robotHeading;
869     if (dx == 1 && dy == 0) targetHeading = 90; // Đông
870     else if (dx == -1 && dy == 0) targetHeading = 270; // Tây
871     else if (dx == 0 && dy == 1) targetHeading = 180; // Nam
872     else if (dx == 0 && dy == -1) targetHeading = 0; // Bắc
873
874     // Tính số lần quay để đạt hướng mục tiêu
875     int turns = (targetHeading - robotHeading + 360) % 360 / 90;
876     if (turns == 0) {
877         // Đã đúng hướng, di chuyển tới và cập nhật posQueue
878         posQueue.pop(); // Xóa vị trí mục tiêu khỏi hàng đợi
879         return "F";
880     } else if (turns == 1) {
881         return "R"; // Quay phải
882     } else if (turns == 3) {
883         return "L"; // Quay trái
884     } else {
885         // Cần quay 180 độ, chọn quay phải hai lần (hoặc trái)
886         return "R";
887     }
888 }

```

- Kiểm tra hoàn thành

```

569 if (!hasUnvisitedCells() || (!isValidPath() && posQueue.empty() && !findNearestVisitedWithUnvisited(robotX,
robotY))) {
570     Serial.println("All cells explored or no path to unvisited cells. Exploration complete!");
571     if (isDisplayInitialized) {
572         display.clearDisplay();
573         display.setTextSize(1);
574         display.setTextColor(SSD1306_WHITE);
575         display.setCursor(0, 0);
576         display.println("Exploration Complete");
577         display.display();
578     }
579     isFinished = true;
580     exportData();
581     resetMap();
582     return;
583 }

```

4.2.5 Thuật toán điều khiển động cơ (Motor Control)

Thuật toán điều khiển hai động cơ DC thông qua module L298N để thực hiện các hành động: di chuyển tiến, quay trái, quay phải, và dừng. Các chân điều khiển (IN1, IN2, IN3, IN4) được đặt trạng thái HIGH hoặc LOW để điều khiển hướng quay của động cơ. Thời gian di chuyển (MOVE_DURATION: 500ms) và quay (TURN_DURATION: 525ms) được cố định để đảm bảo tính đồng nhất.

Các bước thực hiện:

1. Di chuyển tiến: Đặt IN1 và IN4 là HIGH, IN2 và IN3 là LOW để cả hai động cơ quay cùng chiều, đẩy robot tiến về phía trước trong MOVE_DURATION.

2. Quay trái: Đặt IN1 và IN3 là HIGH, IN2 và IN4 là LOW để động cơ trái quay thuận, động cơ phải quay ngược, khiến robot xoay trái trong TURN_DURATION.
3. Quay phải: Đặt IN2 và IN4 là HIGH, IN1 và IN3 là LOW để động cơ trái quay ngược, động cơ phải quay thuận, khiến robot xoay phải trong TURN_DURATION.
4. Dừng: Đặt tất cả các chân (IN1, IN2, IN3, IN4) là LOW để dừng động cơ.
5. Sau mỗi hành động, gọi stopMotors() để đảm bảo robot dừng hoàn toàn trước khi thực hiện hành động tiếp theo.

Các hàm xử lý trong code:

- Hàm tiến phía trước

```
919 void moveForward() {
920     Serial.println("Moving forward for " + String(MOVE_DURATION) + " ms...");
921     digitalWrite(IN1, HIGH);
922     digitalWrite(IN2, LOW);
923     digitalWrite(IN3, LOW);
924     digitalWrite(IN4, HIGH);
925     delay(MOVE_DURATION);
926     stopMotors();
927     Serial.println("Move forward complete.");
928 }
```

- Hàm rẽ trái

```
930 void turnLeft() {
931     Serial.println("Turning left for " + String(TURN_DURATION) + " ms...");
932     digitalWrite(IN1, HIGH);
933     digitalWrite(IN2, LOW);
934     digitalWrite(IN3, HIGH);
935     digitalWrite(IN4, LOW);
936     delay(TURN_DURATION);
937     stopMotors();
938     Serial.println("Turn left complete.");
939 }
```

- Hàm rẽ phải

```
941 void turnRight() {
942     Serial.println("Turning right for " + String(TURN_DURATION) + " ms...");
943     digitalWrite(IN1, LOW);
944     digitalWrite(IN2, HIGH);
945     digitalWrite(IN3, LOW);
946     digitalWrite(IN4, HIGH);
947     delay(TURN_DURATION);
948     stopMotors();
949     Serial.println("Turn right complete.");
950 }
```

- Hàm dừng xe

```

952 void stopMotors() {
953     Serial.println("stopMotors: All pins LOW");
954     digitalWrite(IN1, LOW);
955     digitalWrite(IN2, LOW);
956     digitalWrite(IN3, LOW);
957     digitalWrite(IN4, LOW);
958 }

```

4.2.6 Giao tiếp với ứng dụng Android (Server Communication)

Robot sử dụng thư viện AsyncWebServer để tạo máy chủ web trên cổng 80, cung cấp hai endpoint:

- /data: Gửi dữ liệu trạng thái robot (tọa độ, hướng, bản đồ, RSSI, khoảng cách, v.v.) dưới dạng JSON để ứng dụng Android hiển thị.
- /command: Nhận lệnh từ ứng dụng Android (stop, forward, left, right) qua yêu cầu POST và thực hiện hành động tương ứng.

Thuật toán đảm bảo robot có thể giao tiếp liên tục với ứng dụng Android thông qua WiFi, xử lý lỗi kết nối và trả về phản hồi HTTP.

Các bước thực hiện:

1. Khởi tạo server:

- Kết nối WiFi trong setup() với tối đa WIFI_MAX_ATTEMPTS (20 lần).
- Khởi tạo AsyncWebServer trên cổng 80 và thiết lập hai endpoint /data và /command.
- Gửi địa chỉ IP qua Serial để ứng dụng Android kết nối.

2. Endpoint /data:

- Nhận yêu cầu GET, tạo JSON chứa thông tin robot (robotX, robotY, robotHeading, isFinished, maxWifiSpeed, wifiSpeed, distanceLeft, distanceRight, distanceFront, direction, resetFlag, grid, wifiMap).
- Gửi JSON với mã trạng thái 200.
- Đặt lại resetFlag sau khi gửi dữ liệu lần đầu.

3. Endpoint /command:

- Nhận yêu cầu POST với tham số action (stop, forward, left, right).
- Thực hiện hành động tương ứng: dừng (stopMotors), tiến (moveForward), quay trái (turnLeft), quay phải (turnRight).
- Trả về phản hồi HTTP 200 nếu thành công, 400 nếu lệnh không hợp lệ hoặc robot đã dừng (isFinished).
- Cập nhật trạng thái robot và ghi log hành động.

Các hàm xử lý trong code:

```

147 Serial.println("Connecting to WiFi...");
148 WiFi.begin(ssid, password);
149 int wifiAttempts = 0;
150 while (WiFi.status() != WL_CONNECTED && wifiAttempts < WIFI_MAX_ATTEMPTS) {
151     delay(500);
152     Serial.print(".");
153     wifiAttempts++;
154 }
155 if (WiFi.status() != WL_CONNECTED) {
156     Serial.println("\nWiFi connection failed, continuing without WiFi.");
157     wifiConnected = false;
158     if (isDisplayInitialized) {
159         display.clearDisplay();
160         display.setTextSize(1);
161         display.setTextColor(SSD1306_WHITE);
162         display.setCursor(0, 0);
163         display.println("Connect Wifi FAILED");
164         display.display();
165     }
166 } else {
167     Serial.println("\nWiFi connected.");
168     Serial.print("IP Address: ");
169     Serial.println(WiFi.localIP());
170     wifiConnected = true;
171

```

```

213 // Thiết lập endpoint "/command" để nhận lệnh từ ứng dụng Android
214 server.on("/command", HTTP_POST, [](AsyncWebServerRequest *request){
215     if (request->hasParam("action", true)) {
216         String action = request->getParam("action", true)->value();
217         Serial.print("Received command: ");
218         Serial.println(action);
219         if (action == "stop") {
220             stopMotors();
221             isFinished = true;
222             Serial.println("Robot stopped via command. isFinished set to true.");
223             request->send(200, "text/plain", "Robot stopped");
224             if (isDisplayInitialized) {
225                 display.clearDisplay();
226                 display.setTextSize(1);
227                 display.setTextColor(SSD1306_WHITE);
228                 display.setCursor(0, 0);
229                 display.println("Stopped by User");
230                 display.display();
231             }
232         } else if (action == "forward") {
233             if (!isFinished) {
234                 moveForward();
235                 updatePosition("F");
236                 consecutiveTurns = 0;
237                 Serial.println("Robot moved forward.");
238                 request->send(200, "text/plain", "Robot moved forward");
239             } else {
240                 request->send(400, "text/plain", "Robot is stopped");
241             }
242         }
243     }
244 }

```

```

242     } else if (action == "left") {
243         if (!isFinished) {
244             turnLeft();
245             updatePosition("L");
246             consecutiveTurns++;
247             Serial.println("Robot turned left.");
248             request->send(200, "text/plain", "Robot turned left");
249         } else {
250             request->send(400, "text/plain", "Robot is stopped");
251         }
252     } else if (action == "right") {
253         if (!isFinished) {
254             turnRight();
255             updatePosition("R");
256             consecutiveTurns++;
257             Serial.println("Robot turned right.");
258             request->send(200, "text/plain", "Robot moved right");
259         } else {
260             request->send(400, "text/plain", "Robot is stopped");
261         }
262     } else {
263         Serial.println("Invalid action received.");
264         request->send(400, "text/plain", "Invalid action");
265     }
266 } else {
267     Serial.println("Missing action parameter in command request.");
268     request->send(400, "text/plain", "Missing action parameter");
269 }
270 });

```

```

272     server.begin();
273     Serial.println("Web server started at http://" + WiFi.localIP().toString() + "/data");
274 }
275
276 // Khởi tạo bản đồ và vị trí ban đầu của robot
277 grid[robotX][robotY] = 3; // Robot tại vị trí ban đầu
278 posQueue.push({robotX, robotY, robotHeading});
279 Serial.println("Setup complete. Robot starts at (0,0).");
280 }

```

CHƯƠNG 5

THỬ NGHIỆM VÀ ĐÁNH GIÁ

5.1 Mục tiêu thử nghiệm

Ứng dụng Robot Monitor được phát triển để giám sát và hiển thị thông tin từ một robot điều khiển từ xa (ESP32) trong quá trình di chuyển và đo lường tín hiệu WiFi (RSSI) trên một lưới 10x10. Các mục tiêu cụ thể bao gồm:

- Hiển thị thông tin thời gian thực của robot: vị trí, hướng di chuyển, khoảng cách đến vật cản, hướng di chuyển, và trạng thái (đang chạy hoặc hoàn thành).
- Đo lường và hiển thị RSSI WiFi tại các vị trí trên lưới, xác định vị trí có tín hiệu mạnh nhất.
- Vẽ bản đồ di chuyển của robot (Map 10x10) với các trạng thái ô (chưa thăm, đã thăm, vật cản).
- So sánh hiệu quả đo lường RSSI với phần mềm NetSpot để đánh giá độ chính xác và tính hữu ích.

5.2 Thiết lập thực nghiệm

Phần cứng: Robot sử dụng ESP32 với cảm biến HC-SR04, động cơ L298N, và servo để quét khoảng cách. Robot kết nối với mạng WiFi "Chuc" để đo RSSI.

Phần mềm:

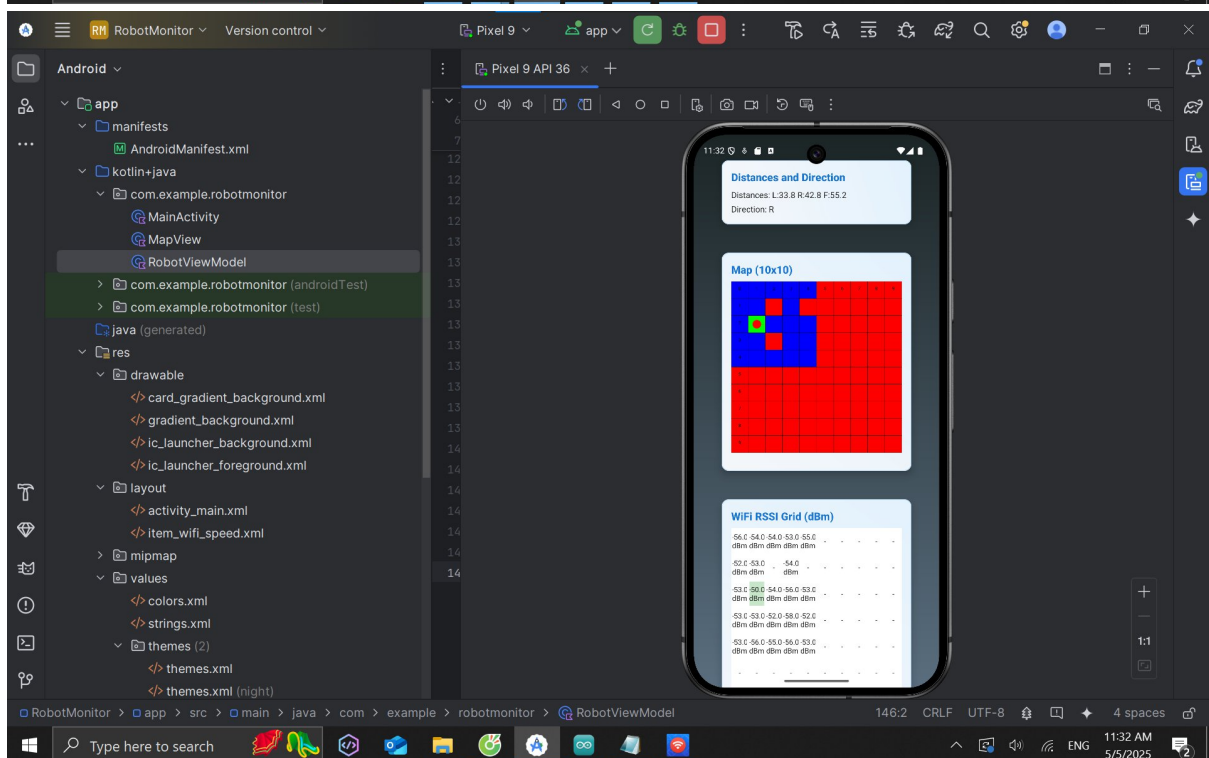
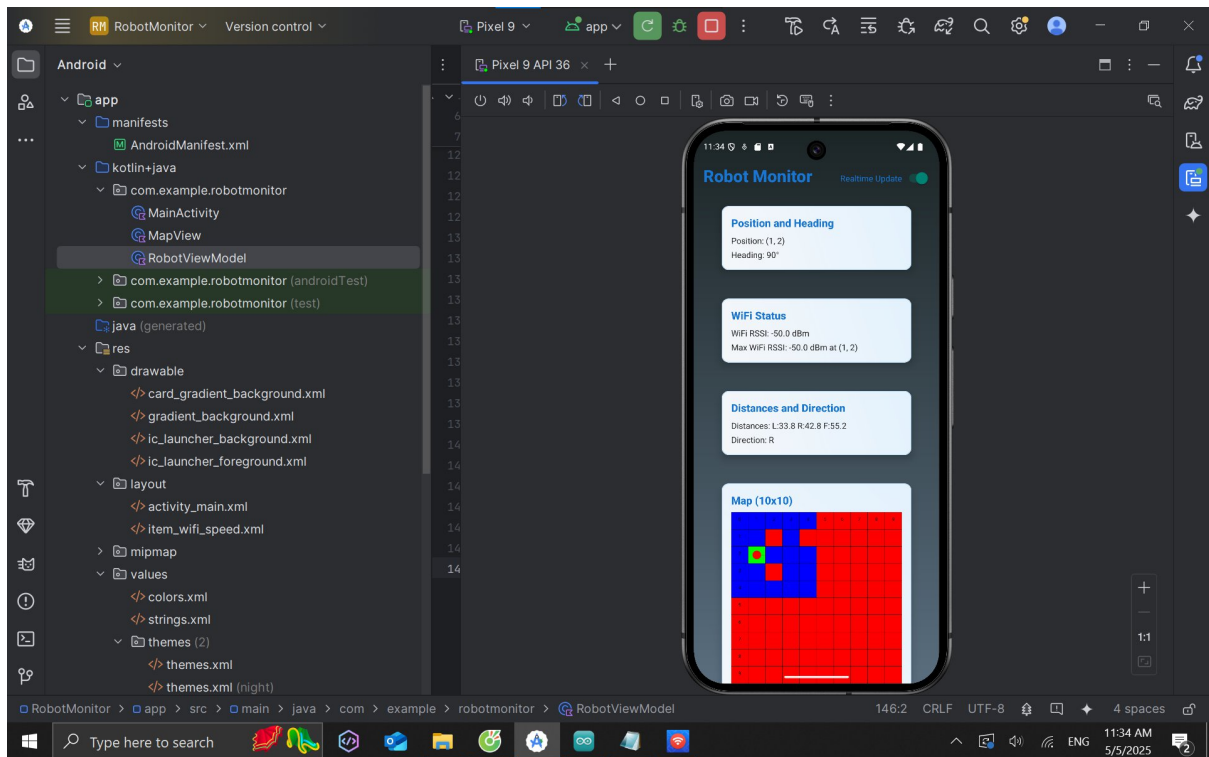
- Ứng dụng Robot Monitor chạy trên Android (Pixel 9, API 36) để hiển thị dữ liệu từ robot qua giao thức HTTP.
- Phần mềm NetSpot chạy trên laptop để đo RSSI của mạng "Chuc" tại cùng thời điểm robot hoạt động.

Môi trường:

- Robot di chuyển trên một lưới 10x10, mỗi ô tương ứng với một vị trí vật lý cố định.
- Mạng WiFi "Chuc" sử dụng băng tần 2.4 GHz, kênh 10, bảo mật WPA2-Personal.

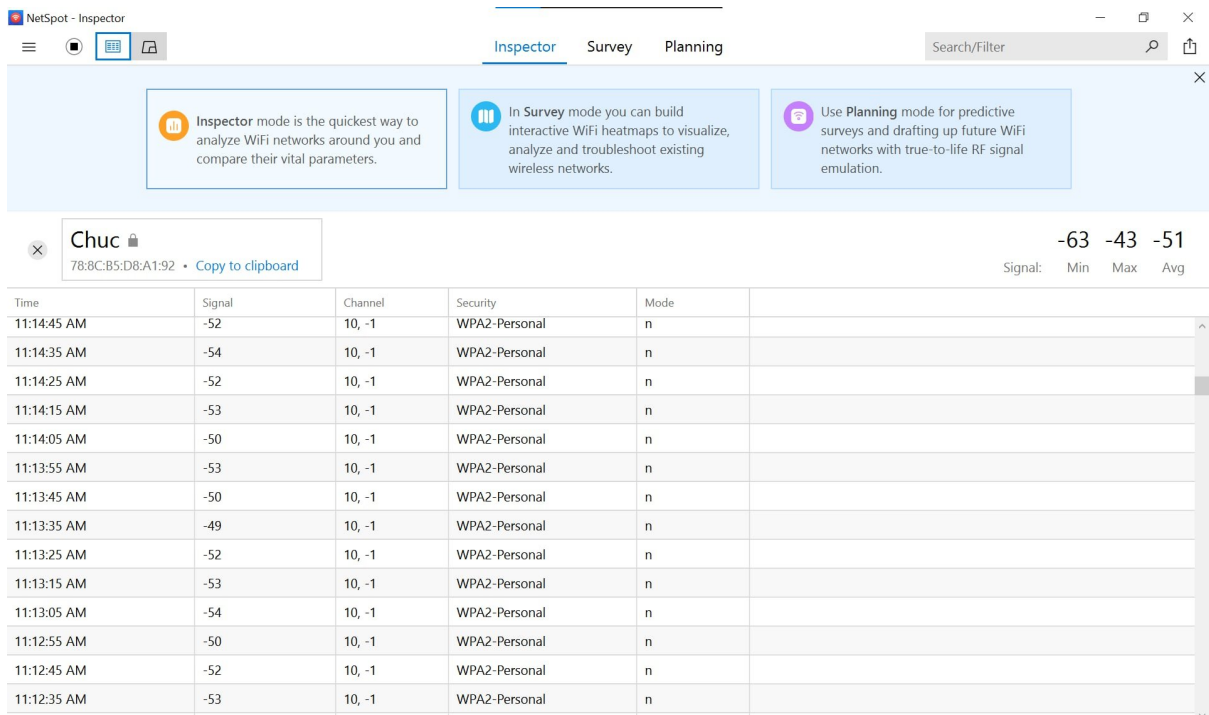
5.3 Dữ liệu thu thập và đánh giá kết quả

a. Phần mềm Robot Monitor:



- **Vị trí robot:** Robot có vị trí cuối cùng ở (2, 1) với hướng 90° (Đông).
- **Khoảng cách đến vật cản:** L: 33.8 cm, R: 48.4 cm, F: 85.2 cm.
- **Hướng di chuyển:** R (quay phải).
- **Bảng RSSI (WiFi RSSI Grid):**
 - RSSI tại (0, 0): -56.0 dBm.
 - RSSI cao nhất: -50.0 dBm tại (2, 1).
 - Các ô khác: dữ liệu hiển thị trên bảng Wifi RSSI Grid, các vị trí trống còn lại robot không di chuyển được sẽ gán là -100dBm
- **Bản đồ (Map):**
 - Robot đã thăm các ô (0, 0), (0, 1), (1, 0), (1, 1) (màu xanh lam)...
 - Ô (1, 2) có vật cản (màu đỏ).
 - Robot hiện tại ở (2, 1) (màu xanh lá).

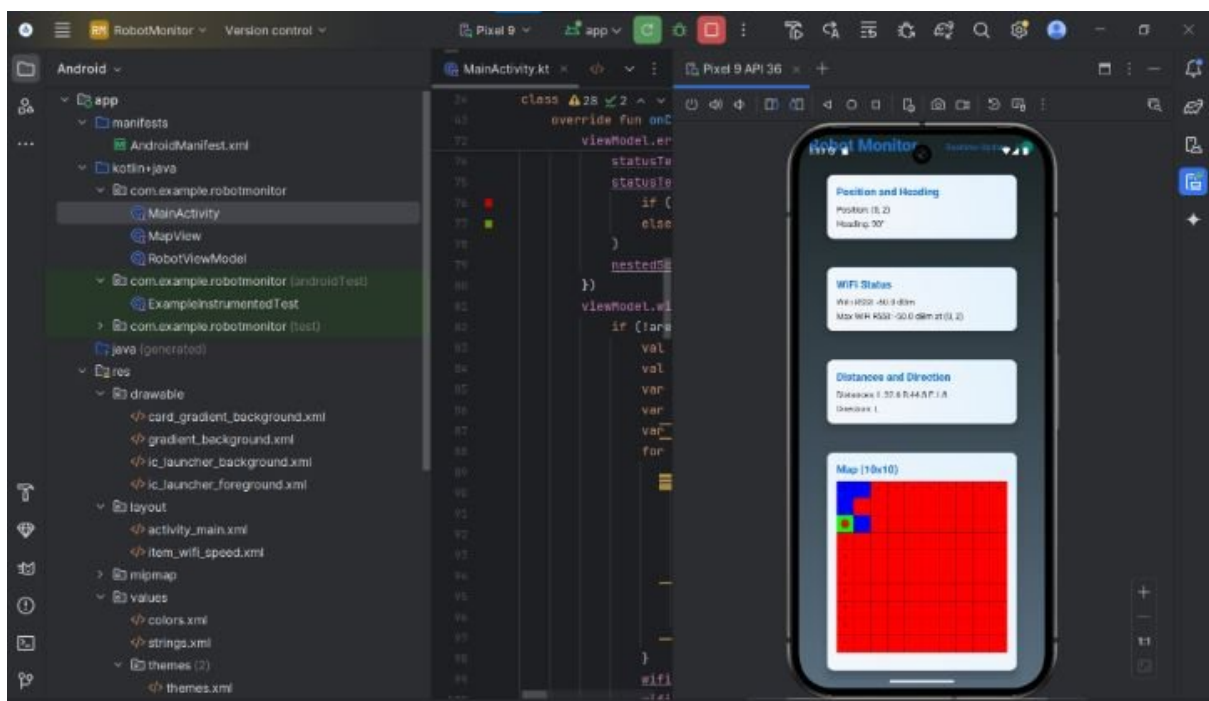
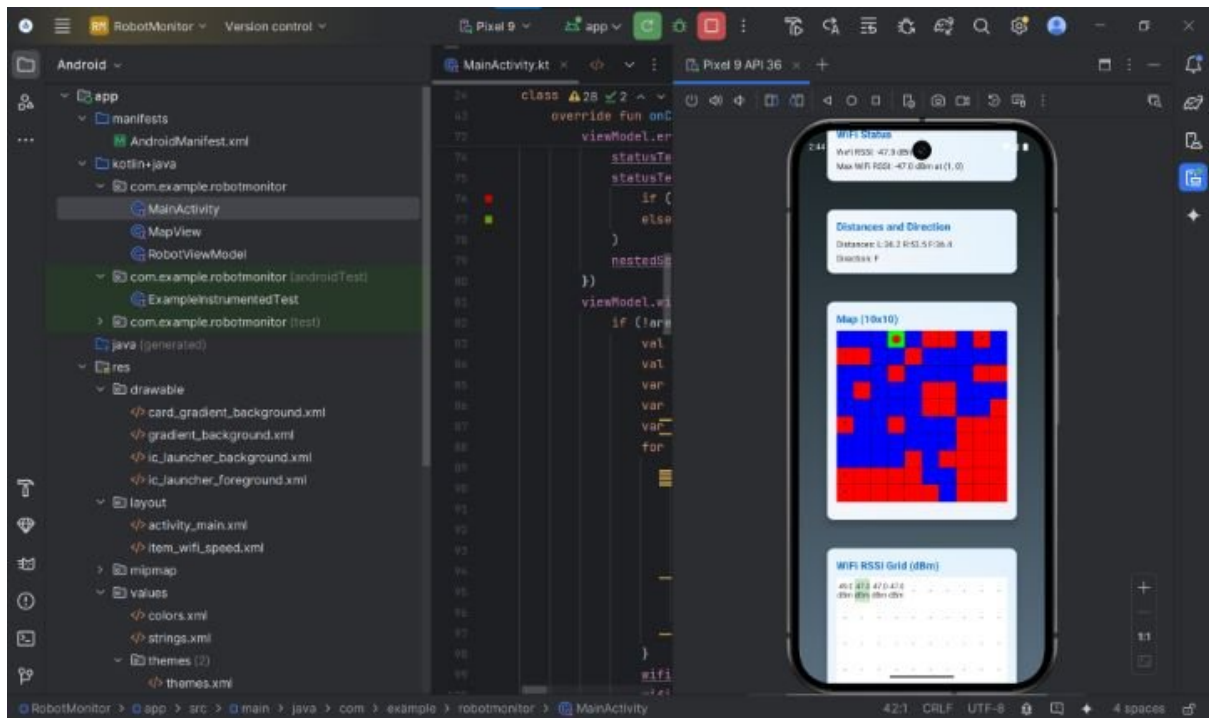
b. Phần mềm Netspot:

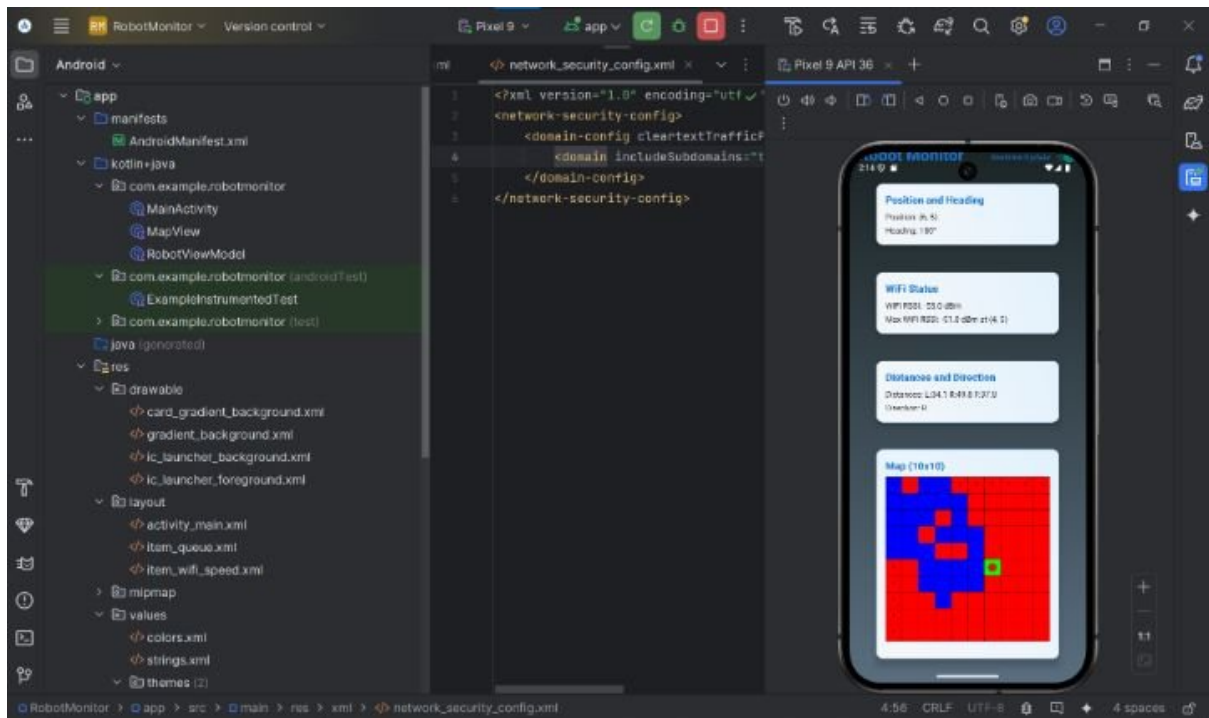


Time	Signal	Channel	Security	Mode
11:14:45 AM	-52	10, -1	WPA2-Personal	n
11:14:35 AM	-54	10, -1	WPA2-Personal	n
11:14:25 AM	-52	10, -1	WPA2-Personal	n
11:14:15 AM	-53	10, -1	WPA2-Personal	n
11:14:05 AM	-50	10, -1	WPA2-Personal	n
11:13:55 AM	-53	10, -1	WPA2-Personal	n
11:13:45 AM	-50	10, -1	WPA2-Personal	n
11:13:35 AM	-49	10, -1	WPA2-Personal	n
11:13:25 AM	-52	10, -1	WPA2-Personal	n
11:13:15 AM	-53	10, -1	WPA2-Personal	n
11:13:05 AM	-54	10, -1	WPA2-Personal	n
11:12:55 AM	-50	10, -1	WPA2-Personal	n
11:12:45 AM	-52	10, -1	WPA2-Personal	n
11:12:35 AM	-53	10, -1	WPA2-Personal	n

- RSSI của mạng "Chuc" dao động từ -49 dBm đến -54 dBm trong khoảng thời gian từ 11:32 AM đến 11:46 AM.
- Giá trị trung bình: -51 dBm.
- Giá trị tối đa: -49 dBm, tối thiểu: -54 dBm.

c. Các trường hợp bản đồ khác





d. Đánh giá kết quả

Hiện thị thông tin thời gian thực:

- Ứng dụng hiển thị chính xác vị trí, hướng, khoảng cách đến vật cản, và hướng di chuyển của robot.
- Bảng RSSI cập nhật đúng tại các ô robot đã thăm, với giá trị RSSI hợp lý.

Bản đồ di chuyển:

- Bản đồ 10x10 hiển thị rõ ràng các ô đã thăm, chưa thăm, và vật cản.
- Vị trí robot và hướng di chuyển được thể hiện trực quan (màu xanh lá với mũi tên hướng Đông).

Đo lường RSSI:

- Phần mềm Robot Monitor di chuyển liên tục còn phần mềm Netspot sử dụng trên thiết bị laptop cố định tại 1 điểm.
- RSSI tại mỗi ô trên map chênh lệch không lớn so với dữ liệu của bảng RSSI của Netspot.
- Giá trị tối đa trên phần mềm Robot Monitor là -50 dBm, trên phần mềm Netspot là -43 dBm, giá trị trung bình là -51 dBm, đảm bảo độ tin cậy do sai số từ thiết bị bắt wifi.

5.4 Ưu và nhược điểm

Ưu điểm:

- Robot Monitor không chỉ đo RSSI mà còn cung cấp thông tin về vị trí, hướng, và bản đồ di chuyển, phù hợp để phân tích WiFi trong không gian vật lý.
- Robot tự động di chuyển trên lưới 10x10, giảm sự can thiệp của người dùng.
- Ứng dụng Android hiển thị dữ liệu trực quan, dễ hiểu với bản đồ, bảng RSSI, và thông tin thời gian thực.
- Dễ dàng tích hợp thêm tính năng (ví dụ: lưu dữ liệu, xuất báo cáo,...).

Nhược điểm:

- RSSI đo được thấp hơn so với NetSpot (trung bình -54 dBm so với -51 dBm), có thể do vị trí đo của robot (gần mặt đất) hoặc nhiễu từ vật cản.
- Nếu robot gặp lỗi (hết pin, cảm biến hỏng), ứng dụng sẽ không thu thập được dữ liệu, trong khi NetSpot hoạt động độc lập.
- Do hạn chế về phần cứng, robot được thiết lập các hàm tiến, quay trái quay phải với độ chính xác tương đối theo các hằng số TURN_DURATION, MOVE_DURATION để điều khiển bánh xe hoạt động theo thời gian.
- Trong một số trường hợp địa hình phức tạp nhiều vật cản, robot có thể bị treo hoặc không di chuyển hết bản đồ do thuật toán di chuyển chưa tối ưu.
- Bản đồ mô phỏng hiện tại là bản đồ 2D với kích thước 10x10 ô, mỗi ô có độ dài khoảng 20cm nên phạm vi chưa được rộng lớn.
- Chưa xử lý được vật cản động di chuyển phức tạp, vật cản nhỏ theo thời gian thực, vật cản hiện tại được mô phỏng sao cho phù hợp với kích thước của ô trên map.

CHƯƠNG 6

KẾT LUẬN VÀ ĐỊNH HƯỚNG PHÁT TRIỂN

6.1 Đánh giá chung

Đề tài “**Xây dựng hệ thống xe tự hành tránh vật cản kết hợp tìm vị trí có cường độ WiFi mạnh nhất**” là một đề tài có tính ứng dụng cao, giúp nhóm tiếp cận thực tế các kỹ thuật lập trình hệ thống nhúng, cảm biến, điều khiển động cơ và xử lý tín hiệu không dây. Trong quá trình thực hiện, nhóm đã vận dụng kiến thức lý thuyết từ môn học để thiết kế, tích hợp phần cứng và phát triển phần mềm điều khiển trên nền tảng vi điều khiển ESP32

Qua quá trình xây dựng và thử nghiệm, hệ thống đã cho thấy khả năng hoạt động ổn định và hiệu quả trong môi trường thực tế mô phỏng. Các thành phần chính trong hệ thống, bao gồm phần cứng, phần mềm điều khiển trên ESP32, cảm biến HC-SR04 và ứng dụng Android, đều phối hợp tốt và hoàn thành chức năng đề ra

Trong quá trình thử nghiệm, xe có thể di chuyển chính xác theo các lệnh điều hướng như tiến, rẽ trái và rẽ phải, đồng thời phản hồi nhanh với các vật cản phía trước. Cảm biến siêu âm HC-SR04 phát hiện vật cản ổn định, giúp robot tránh được va chạm trong quá trình di chuyển. Một điểm đáng chú ý là hệ thống định vị và hiển thị bản đồ theo thời gian thực được tích hợp tốt giữa xe và ứng dụng Android. Trên app, người dùng có thể theo dõi tọa độ (x, y), hướng di chuyển, khoảng cách đến vật cản (trái, phải, trước) và trạng thái WiFi tại từng vị trí một cách rõ ràng

Giao diện ứng dụng được thiết kế trực quan, dễ sử dụng, có khả năng hiển thị bản đồ lưới 10x10, đánh dấu rõ ràng các ô đã đi qua, ô có vật cản và ô chưa thăm, cùng với vị trí hiện tại của xe. Việc đo và hiển thị cường độ WiFi ở từng ô giúp hệ thống xác định vùng có cường độ sóng mạnh nhất, hỗ trợ cho mục tiêu cuối cùng của đề tài

6.2 Kết quả đạt được

Dựa vào quá trình thử nghiệm và đánh giá về hệ thống, nhóm đã thu được một số kết quả như sau:

- Xe di chuyển chính xác theo các lệnh điều hướng đã lập trình
- Phát hiện vật cản hiệu quả trong môi trường thử nghiệm nhờ cảm biến HC-SR04

- Bản đồ lưới 10x10 hiển thị trực quan, thể hiện trạng thái từng ô (đã đi, vật cản, chưa đi)
- Ứng dụng ghi nhận thời gian phản hồi WiFi tại từng điểm, hỗ trợ xác định vị trí có cường độ RSSI mạnh nhất
- Giao tiếp không dây giữa xe và ứng dụng hoạt động ổn định, gần thời gian thực

6.3 Hạn chế

Vì thời gian còn hạn chế và kiến thức còn nhỏ nên hệ thống của nhóm khó tránh khỏi các hạn chế nhất định:

- Việc đo cường độ WiFi dễ bị ảnh hưởng nếu mạng không ổn định hoặc có nhiều thiết bị truy cập cùng lúc
- Cảm biến siêu âm có thể không phát hiện chính xác các vật mềm, nghiêng hoặc hấp thụ sóng siêu âm
- Hệ thống hiện tại chỉ hoạt động tốt trong môi trường tĩnh, đơn giản; chưa sẵn sàng cho môi trường động hoặc phức tạp hơn
- Xe chỉ di chuyển theo lưới cố định, chưa tích hợp thuật toán tìm đường tối ưu và chưa thể tìm trong môi trường rộng lớn

6.4 Định hướng phát triển

Mặc dù hệ thống đã hoạt động tốt dựa trên tiêu chí đặt ra, tuy nhiên để nâng cao hiệu quả và được ứng dụng rộng rãi hơn trong thực tế, cần phải phát triển mở rộng hơn nữa. Sau đây là một số đề xuất phát triển mà nhóm đưa ra:

- Áp dụng các thuật toán điều hướng thông minh (như A*) để xe chọn đường đi ngắn nhất và tránh vật cản hiệu quả hơn
- Tích hợp cảm biến nâng cao (Lidar, camera) để cải thiện khả năng nhận diện vật cản trong môi trường thực tế
- Kết nối hệ thống với nền tảng giám sát từ xa qua MQTT để theo dõi và điều khiển từ xa
- Mở rộng môi trường thử nghiệm sang các không gian lớn hơn, có nhiều điểm truy cập và vật cản động
- Tối ưu ứng dụng Android để bổ sung các chức năng khác như ghi lại nhật ký hành trình, thống kê tín hiệu...

TÀI LIỆU THAM KHẢO

- [1] R.Pertab and R.Murk, ESP32 Based Smart Surveillance System, Pakistan: IEEE, 2019
- [2] B.Jeremy, Exploring Arduino: Tools and Techniques for Engineering Wizardry, Wiley, 2013
- [3] “ESP32 Useful Wi-Fi Library Functions (Arduino IDE)”, [Online], Available: <https://randomnerdtutorials.com/esp32-useful-wi-fi-functions-arduino/>
- [4] S.Sebastian and S.Petros, RSSI-Based Indoor Localization With the Internet of Things, IEEE, 2018
- [5] Các nguồn tài liệu tham khảo khác trên Internet