ĐẠI HỌC ĐÀ NẴNG
TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA ĐIỆN

ĐỒ ÁN TỐT NGHIỆP
CAPSTONE PROJECT

NGÀNH: KỸ THUẬT ĐIỀU KHIỂN VÀ TỰ ĐỘNG HÓA

ĐỀ TÀI:

ỨNG DỤNG THUẬT TOÁN SLAM VÀ XỬ LÝ ĐIỆN TOÁN BIÊN CHO ĐỊNH VỊ, LẬP BẢN ĐỒ ROBOT TỰ HÀNH TRONG NHÀ KHO

Người hướng dẫn: **TS. NGÔ ĐÌNH THANH**
KS. NGUYỄN QUANG REN

Sinh viên thực hiện:

1. NGUYỄN DUY AN – MSSV: 105200353 – LỚP: 20THDLCL1
2. LÝ VĂN THƯƠNG – MSSV: 105200474 – LỚP: 20TDHCLC3
3. NÔNG NGỌC KIÊN – MSSV: 105200455 – LỚP: 20TDHCLC3

Đà Nẵng, 6/2025

TÓM TẮT NỘI DUNG

Tên đề tài: *Ứng dụng thuật toán slam và xử lý điện toán biên cho định vị, lập bản đồ rotbot tự hành trong nhà kho.*

Nhóm sinh viên thực hiện:

Nguyễn Duy An	MSSV: 105200353	Lớp: 20TDHCLC1
Nông Ngọc Kiên	MSSV: 105200455	Lớp: 20TDHCLC3
Lý Văn Thương	MSSV: 105200474	Lớp: 20TDHCLC3

Trong bối cảnh công nghiệp hóa – hiện đại hóa đang diễn ra mạnh mẽ, các hệ thống robot tự hành đóng vai trò quan trọng trong việc nâng cao hiệu quả và tự động hóa, đặc biệt trong các kho hàng thông minh. Đề tài này tập trung vào việc nghiên cứu, ứng dụng thuật toán SLAM (Simultaneous Localization and Mapping – định vị đồng thời và lập bản đồ) kết hợp với xử lý điện toán biên (Edge Computing) để nâng cao khả năng định vị chính xác, cập nhật bản đồ môi trường trong thời gian thực và tối ưu hiệu suất xử lý trên robot tự hành.

Mục tiêu chính của đề tài là xây dựng một hệ thống phần cứng và phần mềm giúp robot tự hành có thể di chuyển an toàn, chính xác trong môi trường kho hàng. Thay vì phụ thuộc vào server trung tâm, thuật toán SLAM sẽ được tích hợp và xử lý trực tiếp trên thiết bị điện toán biên Raspberry Pi 4, giúp giảm độ trễ và tăng độ tin cậy của hệ thống. Đồ án gồm 5 chương như sau:

Chương 1: Tổng quan về đề tài

Chương này trình bày tổng quan về đề tài, bắt đầu từ định hướng nghiên cứu đến các thách thức kỹ thuật mà đề tài đặt ra. Nội dung bao gồm bối cảnh ứng dụng robot tự hành trong môi trường kho bãi hiện đại, vai trò của công nghệ SLAM, điện toán biên và hệ điều hành ROS2 trong việc điều khiển và vận hành robot. Từ đó, xác định rõ nhu cầu, mục tiêu và phạm vi nghiên cứu của đồ án.

Chương 2: Phương án thiết kế và lựa chọn các thành phần trong hệ thống

Chương này tập trung xây dựng phương án thiết kế tổng thể cho hệ thống robot. Nội dung bao gồm lựa chọn thuật toán Graph-Based SLAM, thiết kế cấu trúc hệ thống sử dụng ROS2, giới thiệu kiến trúc điện toán biên và mô hình động học robot hai bánh. Ngoài ra, chương cũng trình bày chi tiết quá trình lựa chọn phần cứng như Raspberry Pi 4, ESP32, Lidar, IMU và phần mềm điều khiển, cùng với cấu hình truyền thông giữa các thành phần như ESP32 và máy tính điều hành qua giao thức micro-ROS.

Chương 3: Thiết kế và thi công mô hình

Chương này trình bày quá trình thiết kế cơ khí và thi công phần cứng của mô hình robot. Bao gồm thiết kế khung xe bằng phần mềm NX, các chi tiết như thân đỡ, trụ lục giác, giá gắn cảm biến Lidar và mô hình 3D tổng thể. Bên cạnh đó, chương cũng trình bày sơ đồ kết nối phần cứng giữa các thiết bị, cũng như quá trình thi công và hoàn thiện mô hình robot thực tế.

Chương 4: Xây dựng thuật toán và môi trường điều khiển – giám sát

Chương này mô tả chi tiết quá trình triển khai phần mềm và thuật toán điều khiển. Nội dung gồm cấu hình mô hình robot bằng file URDF, triển khai thuật toán Graph-Based SLAM và thiết lập Navigation2. Đồng thời, chương trình bày việc cấu hình truyền thông Micro-ROS giữa ESP32 và ROS2, xử lý dữ liệu từ Lidar và IMU, thiết kế bộ điều khiển cho động cơ, publish dữ liệu odometry, và hoàn thiện kết nối luồng dữ liệu của toàn hệ thống.

Chương 5: Kiểm nghiệm và đánh giá kết quả

Chương này thể hiện kết quả thu được qua quá trình mô phỏng và thực nghiệm. Trong môi trường mô phỏng Gazebo và Rviz, robot thực hiện các tác vụ lập bản đồ và định vị. Sau đó, hệ thống được thử nghiệm trong môi trường thực tế mô phỏng nhà kho, đánh giá các yếu tố như độ chính xác định vị, khả năng điều hướng, thời gian phản hồi, và tính ổn định của hệ thống. Kết quả thực nghiệm được phân tích nhằm đánh giá hiệu quả của hệ thống đã xây dựng.

ĐẠI HỌC ĐÀ NẴNG
TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA ĐIỆN

CỘNG HÒA XÃ HỘI CHỦ NGHĨA VIỆT NAM
Độc lập - Tự do - Hạnh phúc

NHIỆM VỤ ĐỒ ÁN TỐT NGHIỆP

TT	Họ tên sinh viên	Số thẻ SV	Lớp	Ngành
1	Nguyễn Duy An	105200353	20TDHCLC1	Kỹ thuật Điều khiển và Tự động hóa
2	Nông Ngọc Kiên	105200455	20TDHCLC3	Kỹ thuật Điều khiển và Tự động hóa
3	Lý Văn Thương	105200474	20TDHCLC3	Kỹ thuật Điều khiển và Tự động hóa

1. Tên đề tài đồ án: Ứng dụng thuật toán Slam và xử lý điện toán biên cho định vị, lập bản đồ rotbot tự hành trong nhà kho.

2. Đề tài thuộc diện: ☐ Có ký kết thỏa thuận sở hữu trí tuệ đối với kết quả thực hiện.

3. Các số liệu và dữ liệu ban đầu:

4. Nội đung các phần thuyết minh và tính toán:

a) Phần chung

TT	Họ và tên sinh viên	Nội dung
1	Nguyễn Duy An	- Viết báo cáo thuyết minh. - Tìm hiểu tổng quan về ROS2. - Nghiên cứu thuật toán SLAM. - Tính chọn thiết bị cho hệ thống. - Thi công lắp ráp mô hình cơ khí. - Thi công đấu nối điện cho toàn bộ hệ thống.
2	Nông Ngọc Kiên	
3	Lý Văn Thương	

b) Phần riêng

TT	Họ và tên sinh viên	Nội dung
1	Nguyễn Duy An	- Viết chương trình thuật toán SLAM và tích hợp hệ thống ROS2 trên edge device và máy tính operator - Nghiên cứu tích hợp cảm biến Lidar, IMU - Lập trình file mô tả Robot URDF và tính toán cấu hình NAV2 - Tính toán và lập trình chuyển đổi tọa độ cho robot

2	Nông Ngọc Kiên	- Viết chương trình điều khiển trên ESP32 - Thiết lập giao thức truyền thông Micro-ROS - Vẽ lưu đồ thuật toán
3	Lý Văn Thương	- Mô hình hóa và tính toán bộ điều khiển động cơ - Thiết lập giao thức truyền thông DDS - Nghiên cứu kiểm nghiệm động học differential-drive-robot . - Thiết kế khung thân robot trên phần mềm NX

5. Các bản vẽ, đồ thị (ghi rõ các loại và kích thước bản vẽ):

6. Họ và tên người hướng dẫn: TS. Ngô Đình Thanh
KS. Nguyễn Quang Ren

7. Ngày giao nhiệm vụ đồ án: *23/03/2025*

8. Ngày hoàn thành đồ án: *10/06/2025*

Đà nẵng, ngày…..tháng 6 năm 2025

Trưởng bộ môn Tự động hóa **Người hướng dẫn**

TS. Giáp Quang Huy TS. Ngô Đình Thanh

ĐẠI HỌC ĐÀ NẴNG
TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA ĐIỆN

CỘNG HÒA XÃ HỘI CHỦ NGHĨA VIỆT NAM
Độc lập - Tự do - Hạnh phúc

PHIẾU KIỂM SOÁT TIẾN ĐỘ LÀM ĐỒ ÁN TỐT NGHIỆP

Họ tên sinh viên: Nguyễn Duy An Số thẻ SV: 105200353
Nông Ngọc Kiên Số thẻ SV: 105200455
Lý Văn Thương Số thẻ SV: 105200474

Tên đề tài ĐATN: Ứng dụng thuật toán SLAM và xử lý điện toán biên cho định vị, lập bản đồ robot tự hành trong nhà kho.

Họ tên GVHD: Ts. Ngô Đình Thanh Đơn vị: Khoa Điện, Trường ĐHBK

Họ tên KS đồng HD: Ks. Nguyễn Quang Ren Đơn vị: Công ty ESTEC – Đà Nẵng

Tuần	**Ngày**	**Khối lượng**		**GVHD ký tên**
		đã thực hiện (%)	**tiếp tục thực hiện (%)**	
1	01/03/2025	5%	95%	
2	10/03/2025	10%	90%	
3	25/03/2025	20%	80%	
4	12/4/2025	Duyệt lần 1: Đánh giá khối lượng hoàn thành _____ % : Được tiếp tục làm ĐATN □ Không tiếp tục thực hiện ĐATN □		
5	19/4/2025	40%	60%	
6	26/04/2025	50%	50%	
7	03/05/2025	60%	40%	
8	10/05/2025	Duyệt lần 2: Đánh giá khối lượng hoàn thành _____ % : Được tiếp tục làm ĐATN □ Không tiếp tục thực hiện ĐATN □		
9	17/05/2025	80%	20%	
10	24/05/2025	90%	10%	
11	7/06/2025	95%	5%	

12	17/06/2025	Duyệt lần 3: Đánh giá khối lượng hoàn thành _____ % : Được tiếp tục làm ĐATN □ Không tiếp tục thực hiện ĐATN □		
13	17/06/2025	100%	0%	

LỜI NÓI ĐẦU VÀ CẢM ƠN

Trong bối cảnh cuộc cách mạng công nghiệp 4.0 đang phát triển mạnh mẽ, nhu cầu tự động hóa trong các lĩnh vực sản xuất, vận hành kho bãi ngày càng trở nên cấp thiết. Các hệ thống robot tự hành (Autonomous Mobile Robots – AMR) đóng vai trò quan trọng trong việc tối ưu hóa quy trình vận chuyển, giảm thiểu chi phí nhân công và nâng cao hiệu suất vận hành. Đặc biệt, việc định vị chính xác và xây dựng bản đồ môi trường hoạt động là yếu tố then chốt để robot có thể di chuyển và hoạt động một cách hiệu quả, an toàn.

Một trong những giải pháp được ứng dụng rộng rãi để giải quyết vấn đề này là thuật toán SLAM (Simultaneous Localization and Mapping – định vị và lập bản đồ đồng thời). Bên cạnh đó, sự phát triển của công nghệ điện toán biên (Edge Computing) cũng cho phép xử lý dữ liệu ngay tại thiết bị, giúp giảm độ trễ và tăng tính ổn định cho hệ thống robot. Sự kết hợp giữa SLAM và điện toán biên hứa hẹn sẽ tạo ra các giải pháp hiệu quả cho bài toán điều hướng và lập bản đồ trong môi trường thực tế, đặc biệt là trong các kho hàng có không gian phức tạp.

Qua quá trình tìm hiểu, nghiên cứu và thực nghiệm, dưới sự hướng dẫn tận tình của Thầy TS. Ngô Đình Thanh và sự hỗ trợ nhiệt tình từ KS. Nguyễn Quang Ren, nhóm chúng em đã hoàn thành đồ án tốt nghiệp với đề tài: *"Ứng dụng thuật toán SLAM và xử lý điện toán biên cho định vị, lập bản đồ robot tự hành trong nhà kho."*

Mặc dù đã rất nỗ lực trong quá trình thực hiện, song do hạn chế về thời gian và kinh nghiệm, đề tài không thể tránh khỏi những thiếu sót. Nhóm xin trân trọng cảm ơn và mong nhận được sự góp ý quý báu từ quý Thầy, Cô để hoàn thiện và phát triển đề tài theo hướng thực tiễn hơn trong tương lai.

Nhóm chúng em chân thành cảm ơn quý Thầy Cô Trường Đại học Bách khoa – ĐHĐN nói chung, các Thầy Cô khoa Điện nói riêng trong xuyên suốt 5 năm học vừa qua đã truyền đạt và trang bị cho nhóm chúng em kiến thức không những về chuyên môn, chuyên ngành mà còn những kinh nghiệm và kĩ năng rất cần thiết để nhóm chúng em có thể hoàn thành đồ án tốt nghiệp.

Đặc biệt nhóm chúng em xin gửi lời cảm ơn chân thành đến Thầy TS. Ngô Đình Thanh và Anh KS. Nguyễn Quang lời biết ơn sâu sắc nhất vì những chỉ dẫn, đánh giá, định hướng và những chia sẻ kinh nghiệm bổ ích từ đồ án tới luận văn, tạo điều kiện để chúng em có thể hoàn thành tốt đồ án tốt nghiệp này.

Chúng em xin gửi lời cảm ơn đến các Thầy Cô phản biện và các Thầy Cô trong

hội đồng bảo vệ đồ án đã dành thời gian để nhận xét và giúp đỡ chúng em trong quá trình bảo vệ.

Cuối lời, nhóm chúng em xin gửi đến các quý Thầy Cô, các bạn sinh viên, các cán bộ nhân viên Trường Đại học Bách khoa – Đại Học Đà Nẵng lời chúc sức khỏe, kính chúc toàn thể Nhà trường ngày càng phát triển và gặt hái được nhiều thành tựu to lớn.

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

LỜI CAM ĐOAN

Đồ án đánh dấu cho những thành quả, kiến thức chúng tôi đã thu nhận được trong quá trình rèn luyện, học tập tại trường. Chúng tôi xin cam đoan đồ án được hoàn thành bằng quá trình học tập và nghiên cứu của chúng tôi.

Trong đồ án này chúng tôi có tham khảo một số tài liệu và một số bài báo đều được đưa ra ở phần tài liệu tham khảo.

Chúng tôi xin cam đoan những lời trên là sự thật và chịu mọi trách nhiệm trước thầy cô và hội đồng bảo vệ đồ án.

Đà Nẵng, ngày…tháng…năm…

Nguyễn Duy An Lý Văn Thương Nông Ngọc Kiên

MỤC LỤC

MỤC LỤC HÌNH ẢNH

MỤC LỤC BẢNG

DANH SÁCH CÁC KÝ HIỆU, CHỮ VIẾT TẮT

ROS: Robot operating system
AMR: Autonomous Mobile Robot
DDS: Data distribution service
IMU: Inertial Measurement Unit
LIDAR: Light Detection and Ranging
EDGE: Edge Computing

MỞ ĐẦU

1. Mục đích thực hiện đề tài:

Trong kỷ nguyên của cuộc cách mạng công nghiệp 4.0, việc áp dụng các công nghệ tiên tiến vào quản lý kho bãi và vận chuyển hàng hóa đang trở thành xu hướng tất yếu. Một trong những giải pháp nổi bật là sử dụng robot tự hành (AMR) để thay thế con người trong các công việc lặp đi lặp lại, nguy hiểm hoặc yêu cầu độ chính xác cao. Trong đó, khả năng định vị chính xác và lập bản đồ môi trường là yếu tố then chốt để robot hoạt động hiệu quả trong các không gian phức tạp như nhà kho.

Hiện nay, thuật toán SLAM (Simultaneous Localization and Mapping) đang được sử dụng rộng rãi trong các hệ thống robot di động để giải quyết bài toán vừa định vị vừa xây dựng bản đồ môi trường xung quanh. Bên cạnh đó, công nghệ điện toán biên (Edge Computing) cho phép xử lý dữ liệu trực tiếp tại thiết bị robot, giúp giảm độ trễ và tăng độ tin cậy so với các hệ thống dựa trên điện toán đám mây.

Xuất phát từ thực tiễn trên, nhóm chúng em đã thực hiện đề tài "Ứng dụng thuật toán SLAM và xử lý điện toán biên cho định vị, lập bản đồ robot tự hành trong nhà kho" nhằm nghiên cứu, thiết kế và xây dựng một hệ thống robot có khả năng hoạt động độc lập trong môi trường thực tế, góp phần vào quá trình hiện đại hóa và tối ưu hóa hoạt động.

2. Mục tiêu của đề tài:

Nắm vững các nguyên lý cơ bản của các thuật toán điều hướng dành cho xe tự hành, dựa trên bản đồ được xây dựng từ môi trường thực tế.

Thiết kế và triển khai một robot di động có khả năng tiếp nhận tín hiệu điều khiển từ phần mềm điều hướng, đồng thời thực hiện chính xác các yêu cầu được đưa ra.

Tìm hiểu và thực hiện các bước cần thiết để phát triển một hệ thống xe tự hành hoàn chỉnh, bao gồm: xây dựng bản đồ môi trường, lập kế hoạch đường đi tối ưu, điều khiển xe di chuyển theo đường đi đã định, và tích hợp khả năng phát hiện cũng như tránh vật cản.

3. Đối tượng nghiên cứu:

Đề tài tập trung nghiên cứu hệ thống robot tự hành sử dụng cảm biến LiDAR, IMU để thực hiện định vị và lập bản đồ. Dữ liệu cảm biến được xử lý bằng máy tính nhúng Raspberry Pi4 đóng vai trò điện toán biên, nơi thuật toán SLAM được triển khai. Môi trường nghiên cứu là không gian nhà kho có các chướng ngại vật, lối đi.

4. Phạm vi nghiên cứu:

Phát triển một mẫu xe tự hành có khả năng mang theo máy tính nhúng và vận hành trong không gian trong nhà kho.

Tìm hiểu và áp dụng các thuật toán nhằm xác định lộ trình tối ưu và di chuyển theo quỹ đạo đã được tính toán trước.

Xây dựng ứng dụng bằng cách khai thác các gói phần mềm có sẵn trong cộng đồng ROS2, đồng thời phát triển các gói phần mềm tùy chỉnh để đáp ứng yêu cầu cụ thể của nghiên cứu.

5. Phương pháp nghiên cứu:

Thu thập tài liệu học thuật, giáo trình, nghiên cứu trước đó liên quan đến SLAM, robot tự hành và điện toán biên.

Sử dụng các công cụ mô phỏng như Gazebo, RViz để kiểm nghiệm thuật toán trong môi trường ảo.

Triển khai thực tế trên phần cứng mô hình để đánh giá khả năng ứng dụng thực tiễn.

6. Cấu trúc đề tài:

Đề tài tốt nghiệp được trình bày với các nội dung chính sau:

Chương 1: Tổng quan về đề tài

Chương 2: Phương án thiết kế và lựa chọn các thành phần trong hệ thống

Chương 3: Thiết kế và thi công mô hình

Chương 4: Xây dựng thuật toán và môi trường điều khiển giám sát

Chương 5: Kiểm nghiệm và đánh giá kết quả

CHƯƠNG 1. TỔNG QUAN VỀ ĐỀ TÀI

1.1 Định hướng cho đề tài

Trong kỷ nguyên của cách mạng công nghiệp lần thứ tư (Industry 4.0), sự phát triển mạnh mẽ của các công nghệ tự động hóa, trí tuệ nhân tạo và Internet vạn vật (IoT) đã tạo ra những thay đổi sâu sắc trong mọi lĩnh vực sản xuất và dịch vụ. Một trong những xu hướng nổi bật nhất hiện nay là việc ứng dụng robot tự hành (Autonomous Mobile Robot – AMR) vào các hệ thống sản xuất thông minh và chuỗi cung ứng hiện đại. Không chỉ đơn thuần là một thiết bị cơ khí di chuyển tự động, robot tự hành ngày nay được tích hợp các công nghệ cảm biến tiên tiến, các thuật toán xử lý thông minh và khả năng giao tiếp thời gian thực với hệ thống điều hành trung tâm, giúp nâng cao hiệu suất vận hành, tối ưu hóa quy trình và giảm thiểu sự phụ thuộc vào lao động thủ công.

Hình 1.1: Ứng dụng của robot tự hành trong công nghiệp

Robot tự hành có thể hoạt động độc lập mà không cần sự can thiệp trực tiếp từ con người. Chúng được sử dụng rộng rãi trong nhiều lĩnh vực như công nghiệp sản xuất, kho vận, y tế, nông nghiệp, và cả trong môi trường phức tạp như không gian hay dưới nước. Đặc biệt, trong bối cảnh thương mại điện tử bùng nổ và nhu cầu giao hàng nhanh ngày càng tăng, robot tự hành đang đóng vai trò then chốt trong việc xây dựng kho hàng thông minh – nơi mà mỗi đơn hàng có thể được xử lý nhanh chóng, chính xác và tiết kiệm chi phí.

Tuy nhiên, để robot có thể di chuyển hiệu quả trong không gian chưa được biết trước, một loạt các vấn đề kỹ thuật phức tạp cần được giải quyết, đặc biệt là các bài toán liên quan đến lập bản đồ, định vị, và lập kế hoạch di chuyển. Ba chức năng cốt lõi này tạo nên nền tảng cho khả năng "hiểu" và "phản ứng" với môi trường xung quanh của robot.

Theo nghiên cứu tổng quan của Cadena et al. (2016) trong bài báo "Past, Present, and Future of Simultaneous Localization and Mapping " [8], công nghệ SLAM đóng vai trò trung tâm trong việc giúp robot vừa xây dựng bản đồ môi trường, vừa định vị chính xác bản thân trong không gian đó. Đây là điều kiện tiên quyết để robot có thể hoạt động một cách tự chủ trong môi trường thực tế, đặc biệt là các nhà kho rộng lớn với cấu trúc phức tạp.

Trong những năm gần đây, công nghệ SLAM đã có nhiều bước tiến mạnh mẽ nhờ sự phát triển của các loại cảm biến và thuật toán xử lý dữ liệu thời gian thực. Đặc biệt, các phương pháp SLAM dựa trên cảm biến LiDAR đang thu hút sự quan tâm lớn bởi khả năng cung cấp dữ liệu khoảng cách chính xác và ổn định, không bị ảnh hưởng bởi điều kiện ánh sáng như các phương pháp dựa trên thị giác. Trong bài báo "LiDAR-based SLAM for robotic mapping: state of the art and new frontiers" [9], các tác giả đã thực hiện một khảo sát toàn diện về các thuật toán SLAM sử dụng LiDAR như GMapping, Hector SLAM, Cartographer và LOAM. Bài báo không chỉ phân tích điểm mạnh và hạn chế của từng thuật toán trong các kịch bản thực tế mà còn chỉ ra các hướng phát triển tiềm năng, như tích hợp dữ liệu từ nhiều cảm biến (sensor fusion), tối ưu hóa đám mây điểm theo thời gian thực và tăng khả năng thích ứng với môi trường phức tạp. Đây là cơ sở quan trọng cho việc xây dựng hệ thống robot tự hành trong môi trường nhà kho – nơi yêu cầu độ chính xác cao, vận hành ổn định và khả năng lập bản đồ linh hoạt để phục vụ các nhiệm vụ logistics tự động.

Song song đó, các nền tảng phần mềm mã nguồn mở như ROS2 (Robot Operating System 2) đang trở thành công cụ không thể thiếu trong cộng đồng nghiên cứu và phát triển robot. Với khả năng xử lý đa luồng, hỗ trợ thời gian thực và kiến trúc mô-đun linh hoạt, ROS2 cho phép lập trình viên tích hợp nhiều thuật toán phức tạp như điều hướng, tránh vật cản, giao tiếp IoT… vào hệ thống robot một cách dễ dàng. Theo nhóm phát triển Open Robotics, ROS2 đặc biệt phù hợp với các ứng dụng công nghiệp cần độ tin cậy cao và khả năng mở rộng linh hoạt, như trong hệ thống kho vận thông minh.

Từ những phân tích trên, có thể nhận thấy rằng việc ứng dụng robot tự hành trong môi trường kho hàng thông minh là một xu hướng tất yếu và khả thi, nhưng cần có giải pháp thiết kế phù hợp với điều kiện thực tiễn tại Việt Nam. Do đó, đồ án này định hướng phát triển một mô hình robot tự hành thu nhỏ, ứng dụng thuật toán SLAM kết hợp với nền tảng ROS2, hướng tới một hệ thống có thể hoạt động tự chủ, định vị chính xác và phản ứng linh hoạt trong không gian nhà kho.

Mục tiêu không chỉ dừng lại ở việc xây dựng một robot vận hành cơ bản, mà còn mở ra tiền đề cho các nghiên cứu tiếp theo trong lĩnh vực điều hành kho tự động, kết nối

dữ liệu thời gian thực, và tích hợp vào hệ thống điều hành kho tổng thể (Warehouse Management System – WMS).

1.2 Những thách thức của đề tài

Theo như bài báo nghiên cứu có tên "Mobile Robot Localization: Current Challenges and Future Prospective" [7], tác giả bài viết này phân tích các chiến lược định vị cho robot di động, nhấn mạnh:

- **Kết nối mạng hạn chế và không ổn định**: yêu cầu dữ liệu thời gian thực từ cảm biến và bản đồ môi trường. Tuy nhiên, trong môi trường IIoT, kết nối mạng có thể bị gián đoạn hoặc không ổn định, ảnh hưởng đến độ chính xác và độ tin cậy của định vị robot.
- **Vấn đề bảo mật và quyền riêng tư**: Việc truyền dữ liệu cảm biến và vị trí của robot qua mạng có thể dẫn đến rủi ro về bảo mật và quyền riêng tư, đặc biệt khi dữ liệu được xử lý trên các máy chủ từ xa.
- **Phụ thuộc vào máy chủ trung tâm**: thường dựa vào các máy chủ trung tâm để xử lý dữ liệu, điều này có thể gây ra độ trễ và giảm hiệu suất, đặc biệt khi mạng bị tắc nghẽn hoặc máy chủ gặp sự cố.

Ngoài ra bài bao nghiên cứu có tên "Review: Issues and Challenges of Simultaneous Localization and Mapping (SLAM) Technology in Autonomous Robot" [2] cũng chỉ ra một số thách thức và khó khăn về thuật toán.

- **Độ chính xác và độ tin cậy**: sai số tích lũy theo thời gian và thay đổi môi trường dẫn đến mất định vị hoặc bản đồ bị lệch.
- **Đòi hỏi phải có giải pháp**: Cần có một giải pháp đủ tốt để robot có thể hoạt động độc lập một cách hiệu quả.

1.3 Bối cảnh hệ thống

Trong bối cảnh phát triển công nghiệp và thương mại điện tử hiện nay, nhà kho đóng vai trò then chốt trong chuỗi cung ứng. Tuy nhiên, các kho hàng truyền thống thường phụ thuộc nhiều vào lao động thủ công, dẫn đến chi phí cao, năng suất không ổn định và tiềm ẩn rủi ro an toàn.

Các đặc điểm nhà kho để xe tự hành hoạt động hiệu quả:

- Mặt bằng nhà kho bằng phẳng, ít chướng ngoại vật bất ngờ
- Lối đi (hành lang) đủ rộng
- Bố trí kho hợp lý, có sơ đồ rõ ràng
- Hệ thống kết nối mạng ổn định

Hình 1.2: Bối cảnh nhà kho

Để giải quyết vấn đề trên, xu hướng tự động hóa kho hàng ngày càng phổ biến, trong đó nổi bật là việc ứng dụng robot tự hành. Robot tự hành có khả năng tự di chuyển, tránh vật cản, định vị chính xác và thực hiện các nhiệm vụ vận chuyển hàng hóa trong kho mà không cần sự can thiệp của con người.

So với môi trường gia dụng, môi trường kho có các đặc thù như diện tích lớn, nhiều chướng ngại vật, yêu cầu định vị chính xác cao và cần tích hợp với hệ thống điều hành trung tâm (WMS). Do đó, robot phải sử dụng các công nghệ cảm biến hiện đại như LIDAR, encoder, IMU, và thuật toán điều hướng SLAM để đảm bảo khả năng hoạt động ổn định và an toàn.

1.4 Kết luận

Chương 1 đã trình bày tổng quan về xu hướng phát triển robot tự hành trong bối cảnh công nghiệp 4.0, đặc biệt là trong lĩnh vực kho hàng thông minh. Việc ứng dụng các công nghệ như SLAM, LiDAR và ROS2 không chỉ giúp robot tự định vị và xây dựng bản đồ mà còn góp phần nâng cao hiệu quả vận hành, giảm phụ thuộc vào lao động thủ công.

Bên cạnh những tiềm năng rõ rệt, đề tài cũng đối mặt với nhiều thách thức như độ tin cậy của định vị, giới hạn về phần cứng xử lý, và yêu cầu cao về môi trường vận hành. Tuy nhiên, với cách tiếp cận phù hợp và lựa chọn công nghệ tối ưu, việc xây dựng một mô hình robot tự hành là hoàn toàn khả thi.

Những nội dung của chương này là nền tảng để bước sang chương tiếp theo đi sâu vào trình bày chi tiết về thuật toán và công nghệ được áp dụng trong đề tài.

CHƯƠNG 2. PHƯƠNG ÁN THIẾT KẾ VÀ LỰA CHỌN CÁC THÀNH PHẦN TRONG HỆ THỐNG

2.1 Yêu cầu đặt ra

Trong môi trường nhà kho, robot tự hành phải thực hiện được các chức năng:

- Tự định vị chính xác trong môi trường chưa biết trước.
- Lập bản đồ không gian xung quanh để phục vụ điều hướng.
- Tránh vật cản động/tĩnh, đảm bảo an toàn.
- Xử lý dữ liệu nhanh và ổn định trong thời gian thực.

Nhằm đạt được những yêu cầu trên, hệ thống cần triển khai:

- Một thuật toán SLAM phù hợp để vừa xây dựng bản đồ, vừa ước lượng vị trí robot.
- Một nền tảng điện toán biên (edge computing) để xử lý các tác vụ này ngay tại robot, giảm độ trễ và tránh phụ thuộc mạng internet.

2.1.1 Lựa chọn thuật toán SLAM

Trong lĩnh vực robot và xe tự hành, thuật toán SLAM (Simultaneous Localization and Mapping) đóng vai trò rất quan trọng. SLAM giúp robot hoặc xe tự hành vừa xác định được vị trí của mình trong không gian không cần tham chiếu hệ thống bên ngoài như GPS, vừa xây dựng bản đồ của môi trường xung quanh mà không cần thông tin định vị trước. Nói một cách đơn giản, SLAM giống như việc một người vừa đi vừa vẽ bản đồ khu vực mới, đồng thời ghi nhớ vị trí hiện tại của mình.

Trong quá trình phát triển hệ thống định vị và lập bản đồ đồng thời (SLAM) cho robot tự hành, việc lựa chọn thuật toán phù hợp đóng vai trò then chốt nhằm đảm bảo độ chính xác, hiệu suất tính toán và khả năng mở rộng của hệ thống. Trong phạm vi nghiên cứu này, ba nhánh thuật toán SLAM phổ biến được xem xét gồm: ekf-slam, fastslam, và graph-based slam.

Bảng 2.1: Bảng so sánh tiêu chí các thuật toán SLAM

Tiêu chí	Graph-Based SLAM	EKF-SLAM	FastSLAM
Môi trường mở rộng lớn	✓	X	✓
Xử lý thời gian thực	✓(phụ thuộc phần cứng)	✓	✓
Độ chính xác cao	✓	✓	✓
Tích hợp đa cảm biến	✓	✓	✓
Yêu cầu tài nguyên thấp	X	✓	X

Hỗ trợ tốt trên nền tảng ROS2	✓	✓	X

Với các yêu cầu cần xử lý bản đồ trong môi trường lớn và hỗ trợ tốt trên nền tảng ROS2 thì lựa chọn **Graph-Based SLAM** làm nền tảng triển khai chính cho hệ thống định vị và lập bản đồ của robot.

2.1.2 Tổng quan về thuật Graph-Based SLAM

2.1.2.1 Giới thiệu về thuật toán

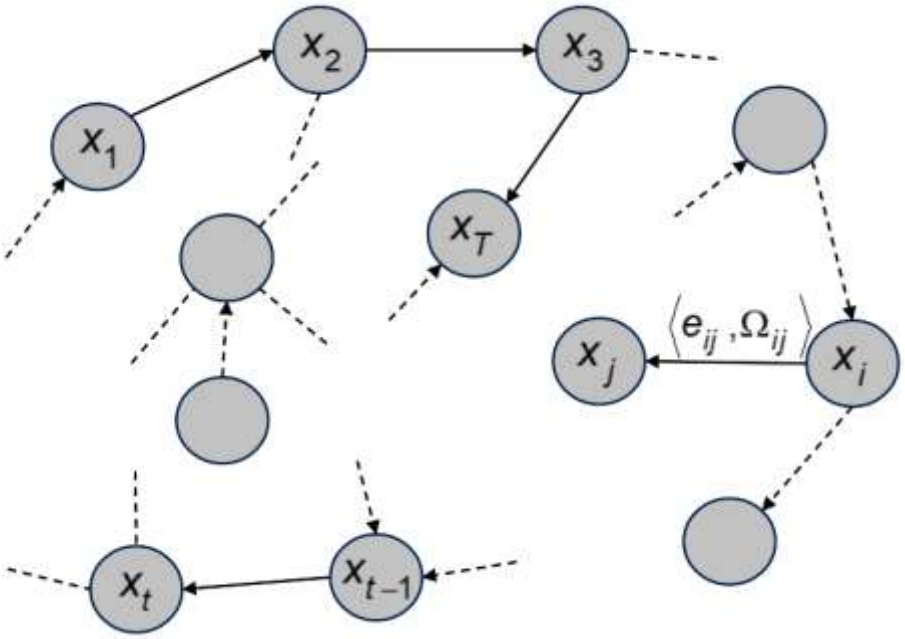
Thuật toán Graph-based SLAM (Simultaneous Localization and Mapping) là một phương pháp nổi bật trong lĩnh vực điều hướng và lập bản đồ cho robot tự động. Mục tiêu chính của Graph-based SLAM là đồng thời xác định vị trí của robot (localization) và xây dựng bản đồ môi trường xung quanh (mapping) thông qua các cảm biến và phép đo quan sát.

Trong Graph-based SLAM, bài toán được mô hình hóa dưới dạng một đồ thị, trong đó các nút (nodes) đại diện cho các trạng thái của robot và các vị trí landmark, còn các cạnh (edges) mô tả mối quan hệ giữa các trạng thái robot hoặc giữa robot và các landmark, được thiết lập dựa trên các phép đo quan sát. Mỗi cạnh của đồ thị tương ứng với một phép đo cảm biến, có thể là quan sát giữa các poses của robot hoặc giữa robot và các điểm landmark trong môi trường.

Quá trình tối ưu hóa trong Graph-based SLAM nhằm mục đích tìm ra các cấu hình của các poses sao cho hàm (cost function) giữa các quan sát thực tế và dự đoán được tối thiểu hóa. Phương pháp tối ưu phổ biến được sử dụng để giải quyết bài toán này là thuật toán Gauss-Newton. Các thuật toán này giúp tính toán và điều chỉnh các trạng thái robot và vị trí landmark sao cho bản đồ được xây dựng một cách chính xác nhất.

Graph-based SLAM là một giải pháp hiệu quả cho các hệ thống robot trong môi trường không gian lớn hoặc phức tạp, nơi mà việc duy trì một bản đồ chính xác trong suốt quá trình di chuyển là vô cùng quan trọng.

2.1.2.2 Thuật toán Graph-Based SLAM



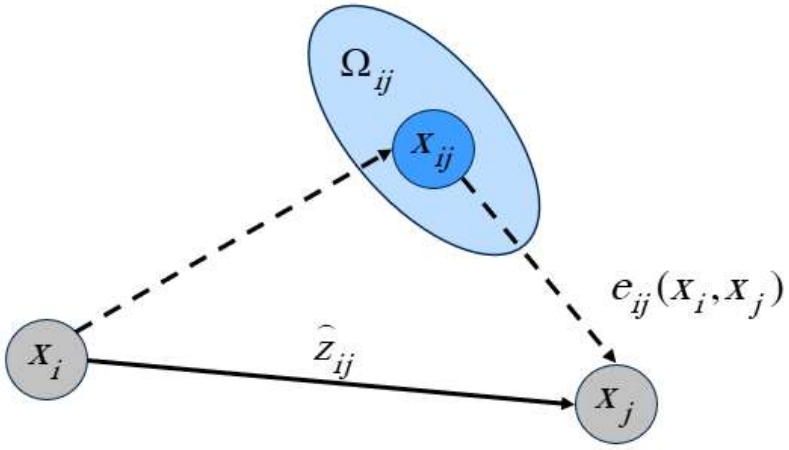
Hình 2.1: Biểu diễn các nút của quá trình SLAM

Gọi $x = (x_1 ... x_T)^T$ là một vector các tham số, trong đó x_i mô tả vị trí (pose) của nút (node) i. Gọi Z_{ij} và Ω_{ij} lần lượt là phép đo và ma trận thông tin giữa nút i và nút j. Phép đo này là một phép biến đổi sao cho các quan sát thu được từ i trùng khớp tối đa với các quan sát thu được từ j .Gọi $Z_{ij}(X_i, X_j)$ là hàm dự đoán của một phép đo ảo, được tính dựa trên cấu hình vị trí hiện tại của hai nút.

Thông thường, dự đoán $Z_{ij}(X_i, X_j)$ là phép biến đổi tương đối giữa hai nút. Do đó, *log-likelihood* l_{ij} của một phép đo Z_{ij} được tính theo công thức:

$$l_{ij} = \left[Z_{ij} - Z_{ij}(X_i, X_j) \right]^T \Omega_{ij} \left[Z_{ij} - Z_{ij}(X_i, X_j) \right]^T \quad (2.1)$$

Gọi $e(Z_{ij}, X_i, X_j)$ là một hàm tính toán độ sai lệch giữa hàm dự đoán Z_{ij} và quan sát thực tế Z_{ij} mà robot thu được.



Hình 2.2: Biểu diễn sơ đồ giữa 2 nút

Gọi C là tập hợp các cặp chỉ số mà tại đó tồn tại một ràng buộc (quan sát) Z . Mục tiêu của phương pháp tối đa hóa khả năng là tìm cấu hình của các nút X^* sao cho tối thiểu hóa *log-likelihood* $F(X)$ hay nói cách khác, làm sao cho sai số giữa các quan sát thực tế và dự đoán là nhỏ nhất.

$$F(X) = \sum_{ij \in C} e_{ij}^T \Omega_{ij} e_{ij} \tag{2.2}$$

Do đó, bài toán cần giải là:

$$X^* = agr \min_x F(x) \tag{2.3}$$

Do bài toán có bản chất phi tuyến, nên cần được tuyến tính hóa để có thể giải một cách hiệu quả. Phương pháp được áp dụng là tuyến tính hóa cục bộ lặp lại, dựa trên khai triển Taylor bậc nhất quanh một ước lượng ban đầu, sau đó giải bài toán tối ưu bằng thuật toán Gauss-Newton.

Nếu ta có một ước lượng ban đầu X vị trí của robot, ta tiến hành xấp xỉ hàm lỗi bằng khai triển Taylor bậc nhất quanh giá trị hiện tại X :

$$\begin{aligned} & e_{ij}(X_i + \Delta X_i, X_j + \Delta X_j) = e_{ij}(X + \Delta X) \\ & \approx e_{ij} + J_{ij} \Delta X \end{aligned} \tag{2.4}$$

Trong đó:

ΔX :là vector hiệu chỉnh, cần tìm để giảm sai số

J_{ij} :là ma trận Jacobian của $e(X_{ij})$

Thay $e_{ij} + J_{ij} \Delta X$ vào $F(X)$ (2.2) ta được:

$$F_{ij}(X + \Delta X) = e_{ij}(X + \Delta X)^T \Omega_{ij} e_{ij}(X + \Delta X)$$
$$\approx (e_{ij} + J_{ij}\Delta X)^T \Omega_{ij} (e_{ij} + J_{ij}\Delta X)$$
$$= e_{ij}^T \Omega_{ij} e_{ij} + 2e_{ij}^T \Omega_{ij} J_{ij}\Delta X + \Delta X^T J_{ij}^T \Omega_{ij} J_{ij}\Delta X$$

Đặt:

$$c = \sum_{ij \in C} e_{ij}^T \Omega_{ij} e_{ij}$$
$$b = \sum_{ij \in C} e_{ij}^T \Omega_{ij} J_{ij}$$
$$H = \sum_{ij \in C} J_{ij}^T \Omega_{ij} J_{ij}$$

Phương trình trên trở thành:

$$F(X + \Delta X) = c + 2b^T \Delta X + \Delta X^T H \Delta X \quad (2.5)$$

Với phương trình tối ưu bậc 2, điều kiện để tối thiểu hóa thì đạo hàm $F(X + \Delta X) = 0$

$$\frac{\partial F(X + \Delta X)}{\partial \Delta X} = 2b + 2H\Delta X = 0$$

Vậy phương trình cần giải là:

$$H\Delta X^* = -b \quad (2.6)$$

Với H là ma trận thông tin (information matrix), thưa và đối xứng, chứa thông tin về mối liên hệ giữa các node.

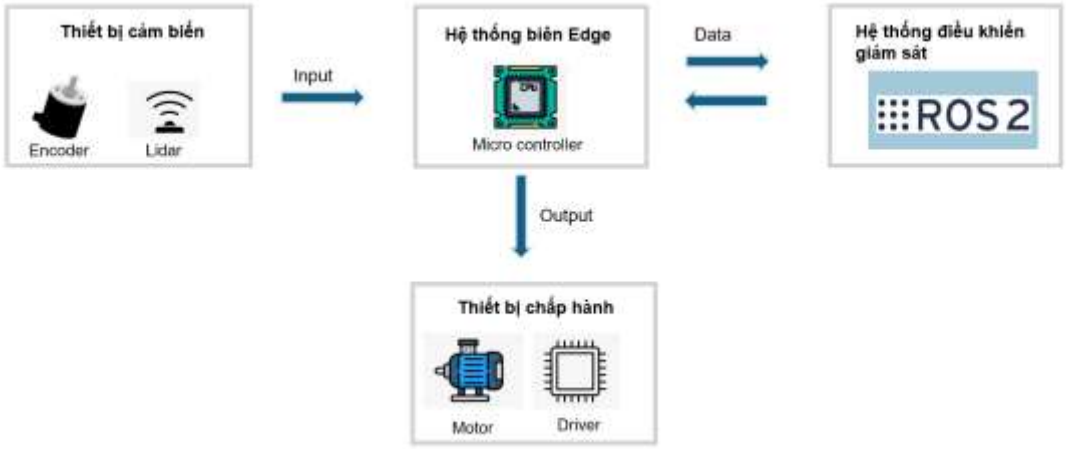
Sau khi giải phương trình thì nghiệm tối ưu là:

$$X^* = X + \Delta X^* \quad (2.7)$$

Thuật toán gauss-Newton sẽ lặp lại các bước tính toán để cập nhật các vị trí của robot một cách chính xác nhất.

2.2 Thiết kế tổng quan hệ thống

Hệ thống bao gồm các thành phần chính như hệ thống điều khiển giám sát, hệ thống biên, các thiết bị cảm biến và các thiết bị chấp hành. Sơ đồ tổng quan của hệ thống được biểu diễn trong hình ảnh dưới. Sơ đồ cung cấp một cái nhìn tổng quan về cấu trúc và sự tương tác giữa các thành phần chính trong hệ thống robot tự hành, hướng đến mục tiêu vận hành tự động trong môi trường nhà kho công nghiệp.



Hình 2.3: Sơ đồ tổng quan hệ thống

Mô tả rõ hơn chức năng của các thành phần trong hệ thống:

- **Các thiết bị cảm biến:** Các thiết bị cảm biến đóng vai trò quan trọng trong việc thu thập tín hiệu phản hồi từ robot trong quá trình di chuyển và thực hiện nhiệm vụ trong môi trường nhà kho. Các cảm biến như Lidar, Encoder, IMU, và cảm biến tiệm cận được sử dụng để theo dõi vị trí, tốc độ, hướng di chuyển, khoảng cách đến vật cản, cũng như trạng thái hoạt động của các bộ phận trên robot.
- **Hệ thống biên Edge:** Hệ thống biên đóng vai trò là trung tâm xử lý được tích hợp trực tiếp trên robot, giúp robot hoạt động tự chủ, phản hồi nhanh và ổn định trong môi trường thực tế.

Tại đây, dữ liệu thu thập từ các cảm biến như Lidar, IMU, Encoder sẽ được xử lý theo thời gian thực để thực hiện các tác vụ quan trọng như:

1. Tiền xử lý dữ liệu cảm biến: Lọc nhiễu, đồng bộ tín hiệu, chuẩn hóa dữ liệu đầu vào.
2. Xử lý thuật toán SLAM: Dựng bản đồ và định vị vị trí robot trong không gian.
3. Lập kế hoạch và điều hướng: Tính toán quỹ đạo, tránh vật cản, tìm đường đi tối ưu.
4. Điều khiển vận tốc: Tính toán và gửi lệnh tốc độ cho động cơ, đảm bảo robot di chuyển đúng hướng và an toàn.

- **Hệ thống điều khiển giám sát:** Đây là bộ điều phối trung tâm, đảm nhiệm tổ chức và quản lý toàn bộ luồng dữ liệu và xử lý:

1. ROS 2 đóng vai trò là hệ điều hành robot giúp kết nối các node xử lý: cảm biến, SLAM, navigation, điều khiển.
2. Hỗ trợ các thư viện, công cụ như: Nav2, RViz để trực quan hóa dữ liệu và trạng thái robot.
3. Cho phép giám sát và điều khiển từ xa, cập nhật bản đồ, quan sát đường đi và trạng thái hệ thống trong thời gian thực.

- **Cơ cấu chấp hành:** Là các thành phần thực hiện chuyển động vật lý theo lệnh điều khiển:

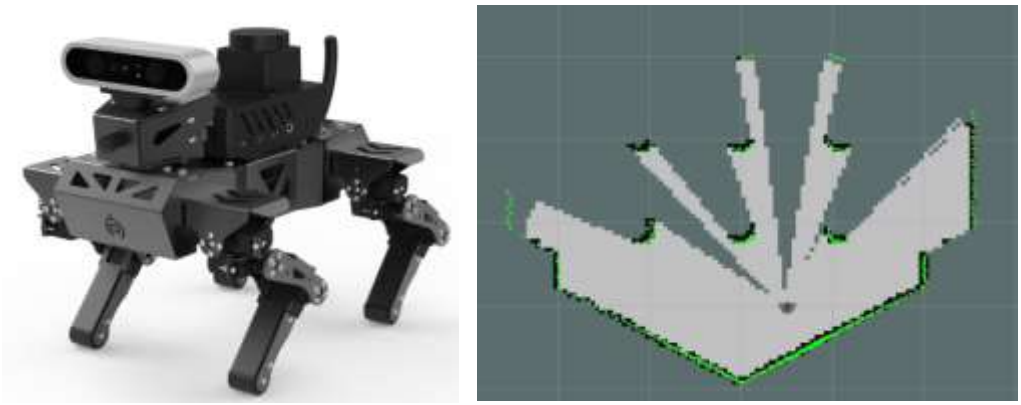
1. Động cơ DC có Encoder: Điều khiển bánh xe di chuyển chính xác.
2. Driver động cơ: Nhận tín hiệu điều khiển từ hệ vi điều khiển hoặc thiết bị biên, cấp nguồn và điều khiển chiều – tốc độ quay của động cơ.

2.2.1 Giới thiệu về hệ thống điều khiển robot ROS2

Hệ điều hành Robot Operating System (ROS2) là một nền tảng mã nguồn mở dành cho robot, được sử dụng rộng rãi trong lĩnh vực robotics nhờ vào nhiều ưu điểm nổi bật. Nó cung cấp một framework phần mềm có khả năng hoạt động trên nhiều loại robot khác nhau mà không đòi hỏi sự thay đổi lớn trong lập trình.

ROS2 ra đời với mục tiêu giúp việc chia sẻ và tái sử dụng phần mềm robot trở nên thuận tiện hơn, giảm bớt công sức xây dựng lại từ đầu khi triển khai trên phần cứng khác. Điều này giúp tiết kiệm thời gian và tài nguyên, đồng thời tạo điều kiện thuận lợi cho việc phát triển các thuật toán phức tạp dựa trên những nghiên cứu trước đó.

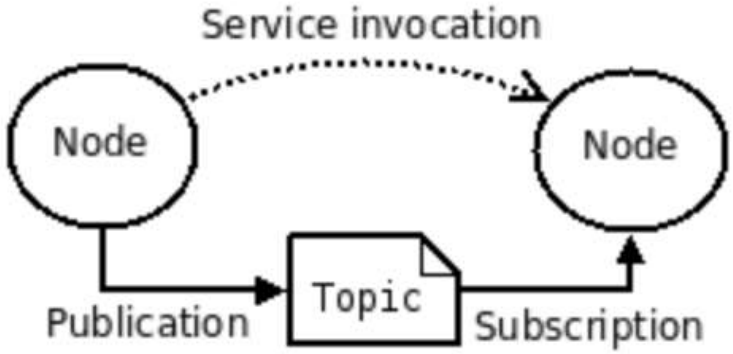
Nhờ những lợi ích mà ROS2 mang lại, nhiều tổ chức nghiên cứu trên thế giới đã và đang đóng góp vào hệ sinh thái này, cung cấp lượng lớn tài nguyên về cả phần cứng lẫn phần mềm. Hiện nay, ROS2 không chỉ được các viện nghiên cứu quan tâm mà còn được nhiều công ty và tổ chức ứng dụng để phát triển các sản phẩm robot có chất lượng cao hơn. Điều này giúp hệ sinh thái ROS2 ngày càng mở rộng với sự hỗ trợ của nhiều thiết bị trên toàn cầu.



Hình 2.4: Một số ví dụ về robot

2.2.2 Cấu trúc của hệ thống ROS2

Hệ thống điều khiển robot ROS2 là một bộ khung phần mềm mã nguồn mở được thiết kế để hỗ trợ phát triển các ứng dụng robot đa dạng và phức tạp. Thay vì hoạt động như một hệ điều hành truyền thống, ROS2 cung cấp một kiến trúc phân tán, linh hoạt, dựa trên các thành phần độc lập giao tiếp với nhau thông qua các giao thức chuẩn hóa. Cấu trúc của ROS2 được xây dựng xung quanh các khái niệm cốt lõi như *nodes*, *topics*, *messages*, *services*, và *parameters*, cùng với sự hỗ trợ của các công cụ và thư viện bổ trợ.



Hình 2.5: Sơ đồ giao tiếp trong ROS2

- **Node (nút)**

Nodes là các đơn vị thực thi nhỏ nhất trong ROS2, thường được viết bằng Python hoặc C++. Mỗi *node* thực hiện một chức năng cụ thể, chẳng hạn như đọc dữ liệu từ cảm biến, xử lý hình ảnh, hoặc điều khiển động cơ. Tính mô-đun của *nodes* cho phép các nhà phát triển chia nhỏ hệ thống robot thành các thành phần độc lập, dễ dàng bảo trì và tái sử dụng.

- **Topic (chủ đề)**

Topics là các kênh giao tiếp không đồng bộ trong ROS2, hoạt động theo mô hình xuất bản/đăng ký (*publish/subscribe*). Một *node* có thể xuất bản (*publish*) dữ liệu lên một *topic*, trong khi các *node* khác đăng ký (*subscribe*) để nhận dữ liệu từ *topic* đó. Cơ chế này đặc biệt hữu ích cho việc truyền dữ liệu liên tục, chẳng hạn như luồng dữ liệu từ cảm biến hoặc trạng thái của robot.

- **Messages (Thông điệp)**

Messages là định dạng dữ liệu được truyền qua *topics*. ROS2 cung cấp một hệ thống thông điệp linh hoạt, hỗ trợ các kiểu dữ liệu đơn giản (số nguyên, số thực, chuỗi) cũng như các cấu trúc dữ liệu phức tạp (mảng, đối tượng lồng nhau). Các thông điệp được định nghĩa trong các tệp .msg, cho phép tùy chỉnh theo từng ứng dụng cụ thể.

- **Services (Dịch vụ)**

Ngoài giao tiếp không đồng bộ qua *topics*, ROS2 còn hỗ trợ giao tiếp đồng bộ thông qua *services*. *Services* hoạt động theo mô hình yêu cầu/phản hồi (*request/response*), phù hợp cho các tác vụ cần kết quả tức thì, chẳng hạn như yêu cầu

robot di chuyển đến một vị trí cụ thể hoặc kiểm tra trạng thái hệ thống. Các *services* được định nghĩa trong các tệp .srv, bao gồm cả phần yêu cầu và phần phản hồi.

- **Parameters (Tham số)**

Parameters là các biến cấu hình toàn cục được lưu trữ trong *Parameter Server*, một phần của *ROS2 Master*. Các *node* có thể truy cập hoặc thay đổi *parameters* để điều chỉnh hành vi của hệ thống mà không cần sửa đổi mã nguồn.

2.3 Điện toán biên

2.3.1 Khái niệm

Trong các hệ thống IoT hiện đại, điện toán biên (Edge Computing) là một mô hình xử lý dữ liệu gần với nơi dữ liệu được sinh ra, thay vì gửi dữ liệu về trung tâm hoặc đám mây để xử lý. Mục tiêu chính của điện toán biên là giảm độ trễ, tiết kiệm băng thông, và nâng cao tính phản ứng thời gian thực cho các hệ thống yêu cầu tốc độ cao như robot tự hành, xe AGV, thiết bị y tế,…

Hình 2.6: Edge Computing

Đối với đề tài này, điện toán biên được triển khai trực tiếp trên nền tảng xử lý nhúng được tích hợp trên robot tự hành, có nhiệm vụ xử lý dữ liệu cảm biến, thực thi thuật toán SLAM và điều khiển robot mà không cần phụ thuộc hoàn toàn vào server trung tâm.

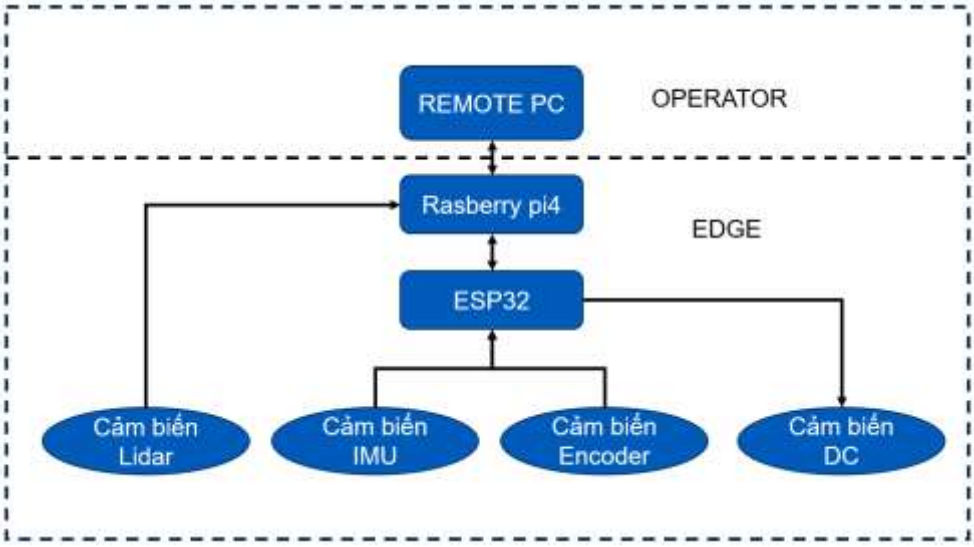
2.3.2 Vai trò của điện toán biên trong hệ thống robot tự hành

Hệ thống robot tự hành hoạt động trong môi trường nhà kho cần khả năng xử lý dữ liệu nhanh chóng và chính xác. Việc áp dụng điện toán biên mang lại các lợi ích sau:

- Xử lý thời gian thực: Các dữ liệu từ LiDAR, encoder, IMU được xử lý trực tiếp trên thiết bị biên, giúp xe phản hồi nhanh với môi trường.
- Tiết kiệm băng thông mạng: Không cần truyền toàn bộ dữ liệu cảm biến lên đám mây, chỉ cần chia sẻ bản đồ hoặc vị trí khi cần thiết.
- Tăng tính tự chủ: Robot tự hành vẫn hoạt động ổn định ngay cả khi mất kết nối mạng.

- Bảo mật dữ liệu: Giảm rủi ro rò rỉ dữ liệu vì dữ liệu không rời khỏi mạng nội bộ.

2.3.3 Kiến trúc xử lý điện toán biên



Hình 2.7: Kiến trúc xử lý điện toán biên

2.3.3.1 Tầng vận hành (operator)

Tầng này bao gồm một máy tính cá nhân (Remote PC) được sử dụng bởi người vận hành để giám sát trạng thái hoạt động của xe tự hành, theo dõi dữ liệu cảm biến và gửi các lệnh điều khiển. Remote PC giao tiếp hai chiều với bộ xử lý trung tâm tại tầng biên (Raspberry Pi 4) thông qua kết nối mạng nội bộ. Trên Remote PC có thể cài đặt các giao diện đồ họa (GUI) hoặc môi trường lập trình như ROS (Robot Operating System) để trực quan hóa quá trình điều hướng, lập bản đồ và các thuật toán điều khiển cao cấp.

2.3.3.2 Tầng xử lý biên (EDGE)

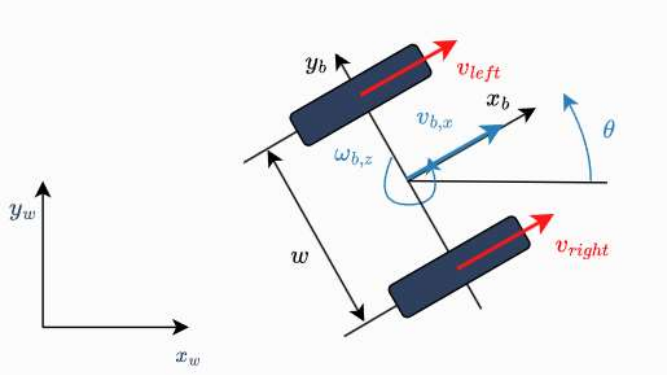
Tầng Edge đóng vai trò là lớp trung gian, thực hiện các chức năng thu thập dữ liệu cảm biến, xử lý tín hiệu, ra quyết định điều khiển và phản hồi về tầng Operator. Các thành phần chính trong tầng Edge bao gồm:

- Raspberry Pi 4: Là bộ xử lý chính tại tầng Edge, chịu trách nhiệm nhận dữ liệu từ cảm biến LiDAR để xử lý định vị, xây dựng bản đồ môi trường (SLAM) và điều hướng. Đồng thời, Raspberry Pi 4 kết nối với ESP32 để gửi các lệnh điều khiển tốc độ và vị trí của động cơ.
- ESP32: Là vi điều khiển có khả năng giao tiếp với các thiết bị ngoại vi và thực thi các tác vụ thời gian thực. ESP32 kết nối với các cảm biến IMU, Encoder và động cơ DC. Nó thực hiện đọc dữ liệu từ cảm biến, xử lý tại chỗ và điều khiển trực tiếp các động cơ dựa trên các thuật toán phản hồi.

2.4 Phương trình động học của robot

Robot tự hành thường được thiết kế với cấu trúc hai bánh hoặc bốn bánh, trong đó các bánh xe được điều khiển độc lập để thực hiện các chuyển động phức tạp như tiến, lùi, quay vòng, hoặc di chuyển theo đường cong. Trong đề tài này, mô hình động học của robot hai bánh được áp dụng để mô phỏng và điều khiển chuyển động của robot.

2.4.1 Các thông số chính



Hình 2.8: Động học robot 2 bánh

Ảnh trên cho thấy hai bánh cách nhau một khoảng cách ω, và mỗi bánh xe có tốc độ tuyến tính riêng: V_{left} (bánh trái) và V_{right} (bánh phải). Tọa độ gốc của robot được biểu thị trong hệ tọa độ (x_ω, y_ω), hệ tọa độ thân robot được định nghĩa bởi (x_b, y_b) với góc nghiêng θ (góc quay của robot so với trục x_ω).

Các thông số động học chính bao gồm:

- v_b,x : Thành phần tốc độ tuyến tính theo trục x_b của thân robot
- v_b,y : Thành phần tốc độ tuyến tính theo trục y_b của thân robot
- ω_b,z : Tốc độ góc quanh trục z (trục thẳng đứng), biểu thị tốc độ quay của robot.
- ω: Khoảng cách giữa hai bánh xe, đóng vai trò quan trọng trong việc tính toán tốc độ góc.

2.4.2 Phương trình động học

Dựa trên mô hình ***động học thuận*** của robot hai bánh, tốc độ tuyến tính của thân robot v_b và tốc độ góc $\omega_{b,z}$ có thể được tính từ tốc độ của hai bánh như sau:

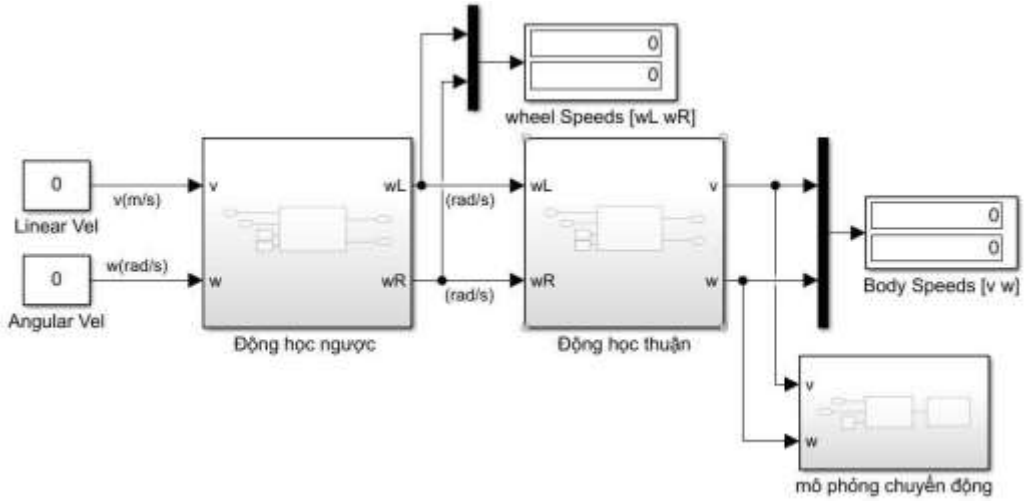
$$v_{b,x} = \frac{v_{right} + v_{left}}{2}$$
$$\omega_{b,z} = \frac{v_{right} - v_{left}}{\omega} \quad (2.8)$$

Để tính được tốc độ để điều khiển bánh xe di chuyển theo mong muốn thì dựa vào ***động học nghịch*** của mô hình trên để đạt được tốc độ bánh xe cần thiết để đạt được tốc độ mong muốn.

$$v_{left} = v_{b,x} - \omega_{b,z}\omega/2$$
$$v_{right} = v_{b,x} + \omega_{b,z}\omega/2$$

2.4.3 Kiểm nghiệm động học trên Matlab

Hình dưới đây trình bày sơ đồ mô phỏng quá trình chuyển đổi giữa vận tốc tuyến tính, góc quay và vận tốc bánh xe, từ đó kiểm nghiệm lại động học thuận và động học nghịch của robot.

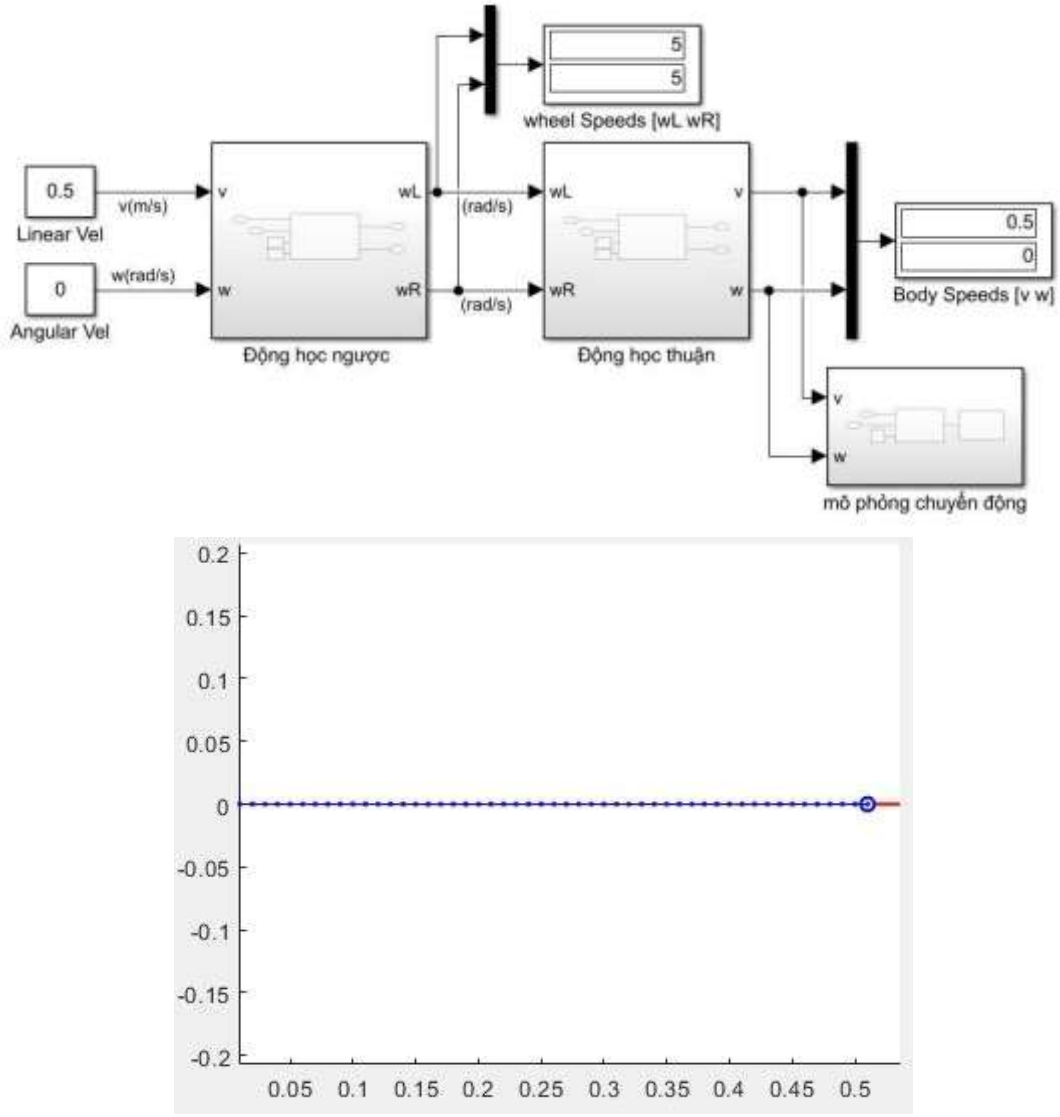


Hình 2.9: Sơ đồ khối động học robot 2 bánh

- **Linear Vel và Angular Vel**: Là đầu vào thể hiện vận tốc tuyến tính và vận tốc góc mong muốn của robot. Đây là thông số điều khiển mục tiêu để robot thực hiện chuyển động.
- **Khối Động học ngược & Động học thuận**:
  - ❖ **Động học ngược** chuyển đổi từ vận tốc thân robot sang vận tốc bánh xe (trái và phải), phù hợp để điều khiển từng động cơ.
  - ❖ **Động học thuận** thực hiện quá trình ngược lại, từ vận tốc bánh xe suy ra vận tốc thân robot, giúp kiểm tra độ chính xác của mô hình động học.
- **Khối Mô phỏng chuyển động:** Dựa trên vận tốc tuyến tính và góc quay, khối này tính toán và biểu diễn quỹ đạo di chuyển thực tế của robot trên mặt phẳng. Đây là phần mô phỏng quá trình robot tự hành trong không gian 2D.
- **Các khối Scope**: Dùng để hiển thị thời gian thực giá trị của các biến vận tốc bánh xe và vận tốc thân robot. Giúp người thiết kế dễ dàng quan sát, đánh giá và điều chỉnh hệ thống điều khiển.

2.4.3.1 Mô phỏng robot đi thẳng:

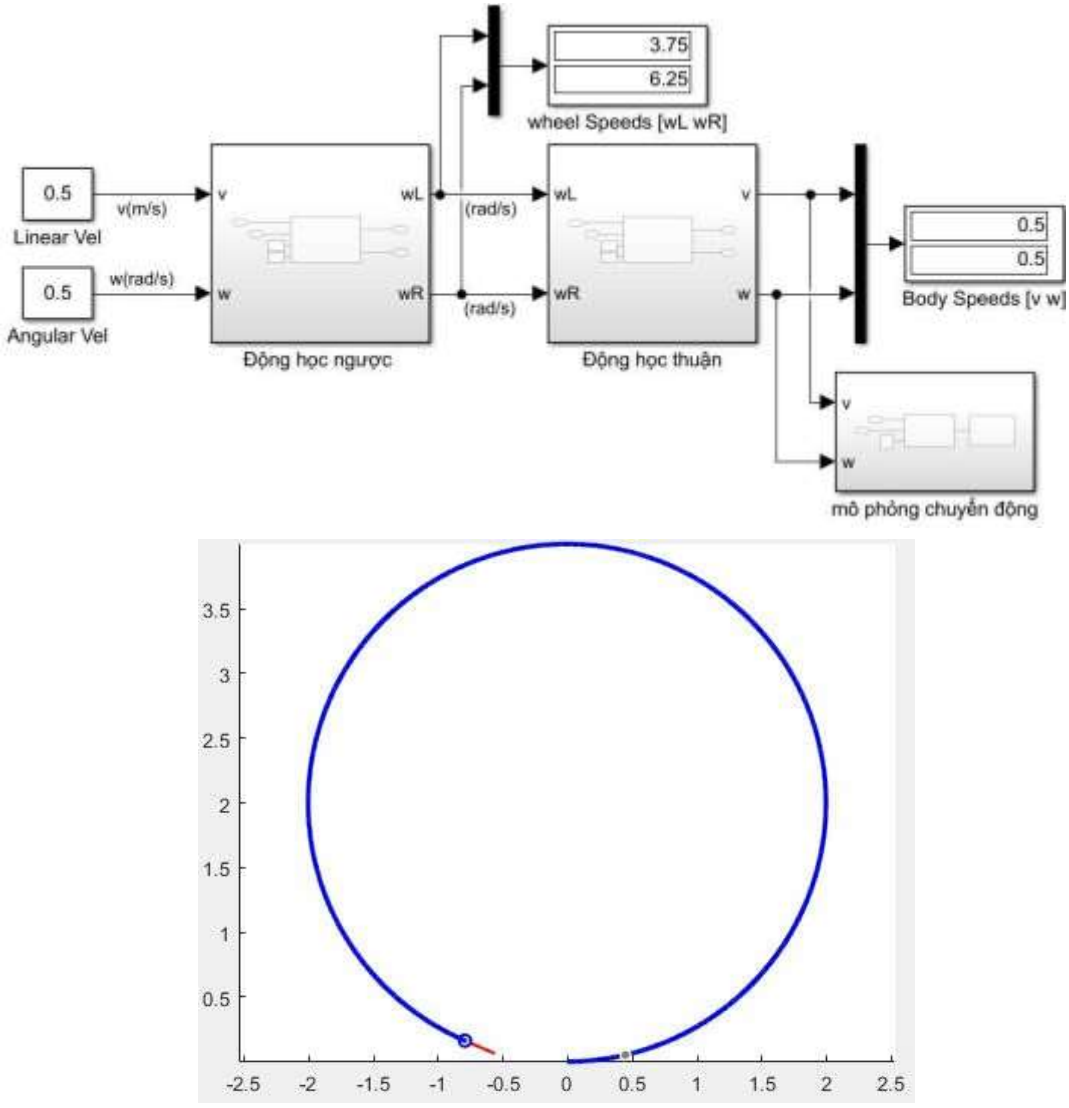
Với vận tốc đặt là 0.5 m/s và vận tốc góc là 0 rad/s



Hình 2.10: Kiểm nghiệm động học (đi thẳng)

2.4.3.2 Mô phỏng robot theo quỹ đạo tròn

Với vận tốc đặt là 0.5 m/s và vận tốc góc là 0.5 rad/s



Hình 2.11: Kiểm nghiệm động học (đi vòng tròn)

2.4.3.3 Đánh giá kết quả

Các giá trị vận tốc đầu ra từ khối động học thuận khớp với vận tốc đầu vào ban đầu (Linear Vel và Angular Vel), chứng tỏ mô hình động học nghịch và thuận đã được xây dựng chính xác và có thể áp dụng vào hệ thống điều khiển thực tế.

Khối mô phỏng chuyển động đã thể hiện chính xác quỹ đạo mà robot sẽ thực hiện dựa trên vận tốc đặt. Điều này giúp xác nhận rằng các phương trình động học của robot đã chính xác.

2.5 Tính chọn các thiết bị

2.5.1 Thiết bị phần cứng

2.5.1.1 Chọn hệ thống truyền động

❖ **Động cơ DC tích hợp encoder**

Hình 2.12: DC servo JGB37-530

Trong mô hình này, nhóm sử dụng 2 động cơ DC Servo JGB37-520, là loại động cơ giảm tốc có đặc điểm như sau:

Bảng 2.2: Bảng thông số động cơ DC

Điện áp hoạt động	Tốc độ không tải	Tốc độ tối đa	Moment định mức	Moment cực đại	Tích hợp Encoder
12V DC	333 (RPM) Vòng/phút	250 RPM	3.5 kg.cm	5 kg.cm	330 xung/vòng

❖ **Bánh xe**

Sử dụng bánh xe với thông số như sau:

Hình 2.13: Bánh xe chính (xanh) và phụ (trắng)

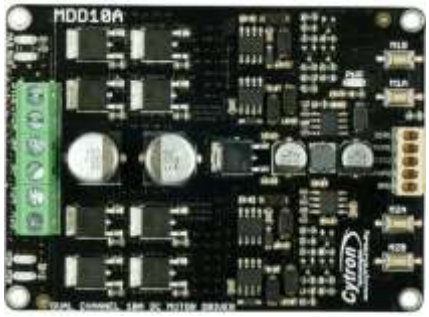
Bảng 2.3: Thông số bánh xe

Bánh xe chính (xanh)	Bánh xe phụ (trắng)
Đường kính 65mm, Độ dày 27mm Số lượng: 2	Đường kính 20mm, Độ dày 14mm Số lượng: 1

❖ **Mạch điều khiển động cơ**

Để điều khiển hai động cơ DC servo, hệ thống sử dụng mạch điều khiển MDD10A, có khả năng:

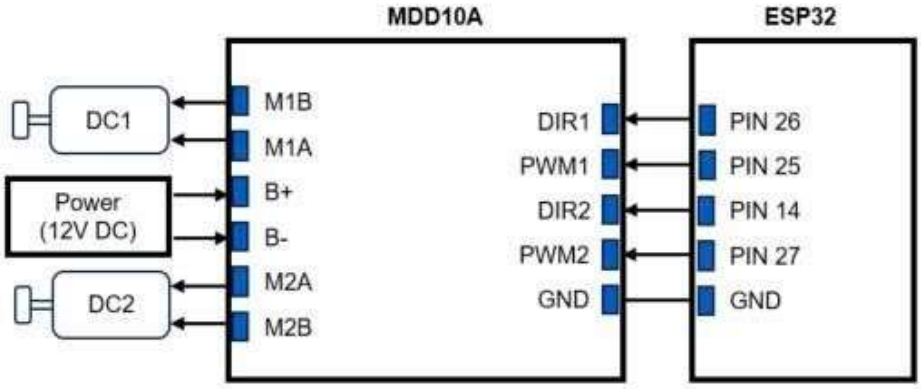
- Điều khiển 2 kênh động cơ độc lập
- Hỗ trợ dòng điện lớn (tối đa 30A)
- Hoạt động ổn định ở điện áp từ 5V đến 30V
- Điều khiển bằng xung PWM từ vi điều khiển (Esp32)



Hình 2.14: Mạch điều khiển động cơ MDD10A

Bảng 2.4: Driver MDD10A

Điện áp	Điện áp điều khiển tín hiệu	Output channel	PWM	Công suất tối đa	Nút nhấn điều khiển thủ công
7~24VDC	3.3V/5V	2 kênh	0~20 kHz	160W/1DC	2 kênh



Hình 2.15: Sơ đồ đấu nối mạch điều khiển động cơ

2.5.1.2 Chọn hệ thống điều khiển

Bộ xử lý trung tâm và các mạch điều khiển là thành phần quan trọng trong hệ thống xe tự hành, chịu trách nhiệm:

- Xử lý dữ liệu từ cảm biến (như Lidar, encoder),
- Tính toán thuật toán điều hướng và điều khiển (PID, SLAM),

- Phát lệnh điều khiển tới động cơ.

Hệ thống được chia làm hai cấp:

- Raspberry Pi 4 đảm nhiệm vai trò xử lý cao cấp, xử lý thuật toán SLAM và điều hướng.
- Esp32 đảm nhiệm xử lý thời gian thực như đọc encoder, phát PWM và điều khiển động cơ.

❖ **Raspberry Pi 4 – Bộ xử lý trung tâm**

Lựa chọn bộ xử lý dựa trên việc đáp ứng các tiêu chí:

- Cài đặt hệ điều hành Ubuntu với ROS2 (Robot Operating System)
- Giao tiếp với Lidar RPLIDAR A1M8 qua cổng USB để thu thập dữ liệu bản đồ
- Giao tiếp với Esp qua UART (hoặc USB) để gửi/nhận dữ liệu về Encoder, tốc độ động cơ
- Xử lý thuật toán điều hướng, SLAM và xuất lệnh điều khiển chuyển động

Hình 2.16: Raspberry pi 4

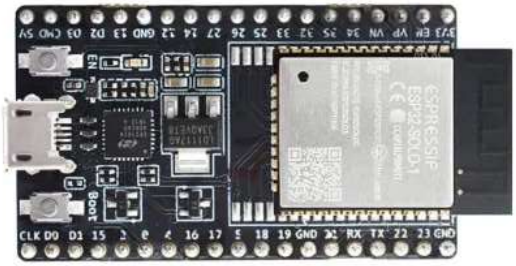
Bảng 2.5: Thông số Raspberry pi

Nguồn cấp	RAM	CPU	Kết nối
5V DC	8GB	Cortex-A72 (ARM v8) 64-bit	USB, GPIO, UART, I2C, SPI, WiFi, Bluetooth

❖ **ESP 32**

Chức năng được sử dụng trong mô hình:

- Đọc tín hiệu Encoder từ động cơ (kênh A/B)
- Tính toán tốc độ thực tế của từng bánh xe
- Phát xung PWM điều khiển tốc độ động cơ thông qua mạch MDD10A
- Nhận lệnh từ Raspberry Pi để điều chỉnh tốc độ, hướng đi của xe



Hình 2.17: Esp32

Bảng 2.6: Thông số Esp32

Điện áp hoạt động	**Chip**	**Tần số**	**Số chân GPIO**	**Giao tiếp**
5V DC	Tensilica Xtensa LX6	80 - 240 MHz	34	UART, I2C, SPI, PWM…

2.5.1.3 Chọn hệ thống cảm biến

Trong đề tài này, hệ thống cảm biến đóng vai trò rất quan trọng giúp xe tự hành có khả năng nhận thức môi trường xung quanh và xác định vị trí tương đối chính xác trong không gian. Hai cảm biến chính được sử dụng bao gồm:

❖ **Cảm biến rp_lidar**

Cảm biến Lidar trên xe tự hành có nhiệm vụ chính là:

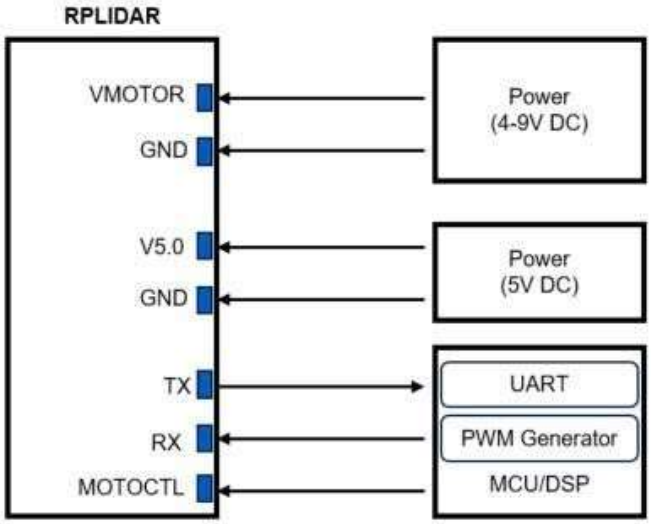
- Thu thập thông tin về môi trường xung quanh, bao gồm khoảng cách đến vật cản và hình dạng không gian
- Cung cấp dữ liệu cho thuật toán SLAM nhằm xây dựng bản đồ và định vị chính xác
- Hỗ trợ phát hiện và tránh vật cản trong quá trình di chuyển

Hình 2.18: Cảm biến RP-LIDAR A1M8

RPLIDAR A1M8 là một cảm biến Lidar 2D quay tròn 360 độ, sử dụng tia laser hồng ngoại để đo khoảng cách đến vật thể theo nhiều hướng xung quanh robot.

Bảng 2.7: Thông số cảm biến Lidar

Phạm vi đo	**Góc quét**	**Tốc độ quay**	**Tốc độ lấy mẫu**	**Độ chính xác**	**Giao tiếp**	**Điện áp**
0.15m~12m	360^0	5-10Hz	~8000 điểm/s	±2%	UART	5V DC



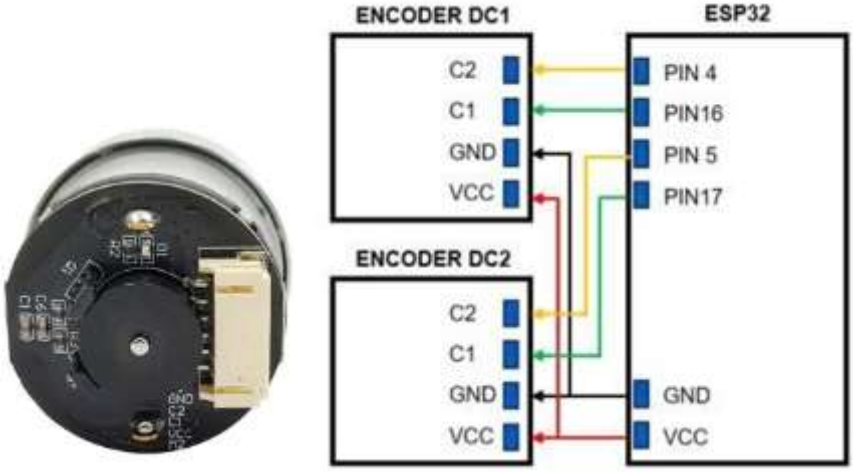
Hình 2.19: Sơ đồ đấu nối cảm biến Lidar

❖ **Cảm biến Encoder (tích hợp ở động cơ)**

Cảm biến Encoder tích hợp trong động cơ DC JGB520 (hay còn gọi là JGB37-520) là loại Encoder quang học được gắn phía sau trục động cơ. Cấu tạo của bộ Encoder này gồm một đĩa mã hóa có các rãnh được gắn trực tiếp lên trục động cơ, và một cặp cảm biến hồng ngoại (gồm LED phát và phototransistor thu) được đặt đối diện nhau. Khi động cơ quay, đĩa mã hóa cũng quay theo, các khe hở trên đĩa sẽ cho phép ánh sáng từ LED truyền qua đến cảm biến thu. Quá trình ánh sáng bị chặn và cho qua luân phiên này sẽ tạo ra các xung điện dạng vuông.

Bảng 2.8: Thông số encoder

Nguồn cấp	**Đo tốc độ và chiều quay**
3.3 ~ 5VDC	Kênh C1, C2



Hình 2.20: Sơ đồ đấu nối encoder

❖ Cảm biến IMU MPU6050

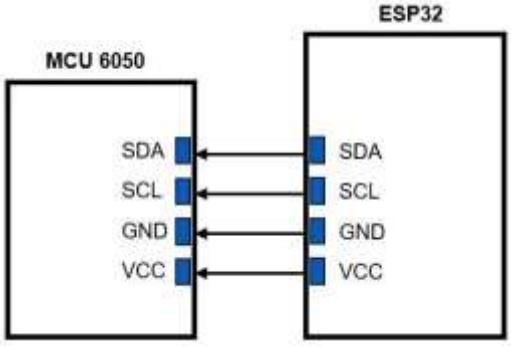
Cảm biến MPU 6050 trên xe tự hành có nhiệm vụ chính là:

- Đo vận tốc góc và gia tốc để xác định hướng chuyển động, góc quay của xe.
- Hỗ trợ thuật toán SLAM trong việc ước lượng chuyển động (odometry).
- Giúp ổn định định vị khi Lidar bị nhiễu hoặc tầm quét hạn chế.

Hình 2.21: Cảm biến MPU6050

Bảng 2.9: Thông số Cảm biến MPU 6050

Điện áp sử dụng	Giao tiếp	Cảm biến gia tốc	Cảm biến con quay
3-5V	I2C	±2g, ±4g, ±8g, ±16g	±250, ±500, ±1000, ±2000°/s



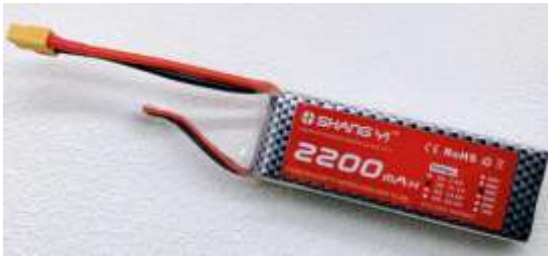
Hình 2.22: Sơ đồ đấu nối cảm biến gia tốc

2.5.1.4 Chọn hệ thống nguồn

Hệ thống nguồn giữ vai trò then chốt trong việc cung cấp năng lượng ổn định cho toàn bộ mô hình xe tự hành. Các thành phần chính như động cơ, vi điều khiển, máy tính nhúng (Raspberry Pi) và cảm biến đều yêu cầu mức điện áp và dòng điện khác nhau. Do đó, cần có một thiết kế hợp lý để vừa đảm bảo hiệu suất hoạt động, vừa tránh tình trạng sụt áp hoặc quá tải.

❖ **Pin Lipo – nguồn cung cấp cho động cơ**

Cấp nguồn cho DC motor driver điều khiển 2 động cơ DC Servo JGB37-520



Hình 2.23: Pin lipo 2200mAh

Bảng 2.10: Thông số pin Lipo

Điện áp	Dung lượng	Dòng xả	Kích thước	Kiểu cắm
11.1V / 3S (sạc đầy 12.6V)	2200 mAh.	45C	105mm x 34.5mm x 23mm	XT60

❖ **Pin cell lion – nguồn cho bộ xử lý**

Cấp nguồn cho Raspberry pi và Esp32, mục đích tách riêng nguồn cho bộ xử lý và động cơ để cách ly nguồn tránh nhiễu khi điều khiển.

Hình 2.24: Cell pin 18650 Samsung 35E

Bảng 2.11: Thông số pin 18650 Samsung 35E

Điện áp	Dung lượng	Dòng xả liên tục	Kích thước
3.6V/1 Cell	3500mAh	8A	18mm x 65mm

❖ **Mạch quản lý nguồn cho bộ xử lý**

Waveshare UPS HAT (B) là một bo mạch nguồn mở rộng cho Raspberry Pi và Esp32 , hoạt động như một UPS (Uninterruptible Power Supply – nguồn điện liên tục). Nó có khả năng chuyển đổi, bảo vệ, và giám sát nguồn điện cấp cho Raspberry Pi và Esp32. Việc lắp 2 cell pin lion đáp ứng đủ nguồn điện luôn ổn định và không bị gián đoạn cho 2 bộ xử lý.

Hình 2.25: Mạch Waveshahe UPS HAT (B)

Bảng 2.12: Thông số mạch quản lý nguồn

Điện áp vào	Điên áp ra	Dòng tối đa	Loại pin hỗ trợ	Tích hợp cổng sạc
6V~8V	5V DC	5A ổn định	2 cell lion 18650 (nối tiếp)	USB

2.5.2 Phần mềm

2.5.2.1 Phần mềm Visual Studio Code

Visual Studio Code (VS Code) là một môi trường lập trình mạnh mẽ, miễn phí và đa nền tảng, rất được ưa chuộng trong phát triển phần mềm robot sử dụng ROS 2. Với khả năng hỗ trợ lập trình Python vượt trội, tích hợp terminal và hệ thống tiện ích mở rộng phong phú, VS Code cho phép lập trình viên dễ dàng viết, quản lý và gỡ lỗi các node ROS 2 một cách trực quan và hiệu quả. Đặc biệt, các extension chuyên biệt cho ROS 2 giúp tự động nhận diện workspace, hỗ trợ launch file, topic, message và nhiều công cụ quan trọng khác trong hệ thống robot. Nhờ giao diện thân thiện, tính năng thông minh và khả năng mở rộng linh hoạt, VS Code trở thành lựa chọn lý tưởng cho việc phát triển và kiểm thử ứng dụng robot sử dụng ROS 2 bằng Python.

Hình 2.26: Phần mềm VS code

2.5.2.2 Phần mềm Arduino IDE

Arduino IDE là một phần mềm mã nguồn mở dùng để lập trình và nạp chương trình cho các vi điều khiển như Arduino Uno, Mega, Nano, và đặc biệt là ESP32 – dòng vi điều khiển 32-bit tích hợp Wi-Fi và Bluetooth rất phổ biến trong các ứng dụng IoT và robot. Arduino IDE có giao diện đơn giản, dễ tiếp cận, hỗ trợ lập trình bằng ngôn ngữ gần giống C/C++, rất thuận tiện cho cả người mới bắt đầu lẫn lập trình viên có kinh nghiệm.

Khi tích hợp board ESP32 thông qua Board Manager, Arduino IDE cho phép biên dịch và nạp chương trình trực tiếp qua cổng USB, đồng thời hỗ trợ đầy đủ các tính năng phần cứng của ESP32 như: giao tiếp UART, I2C, SPI, cảm biến ADC/DAC, PWM, Wi-Fi, Bluetooth và cả xử lý đa luồng (FreeRTOS). Ngoài ra, Arduino IDE còn cung cấp kho thư viện phong phú và cộng đồng người dùng rộng lớn, giúp người phát triển dễ dàng tìm kiếm, tái sử dụng và mở rộng các chức năng cần thiết cho dự án.

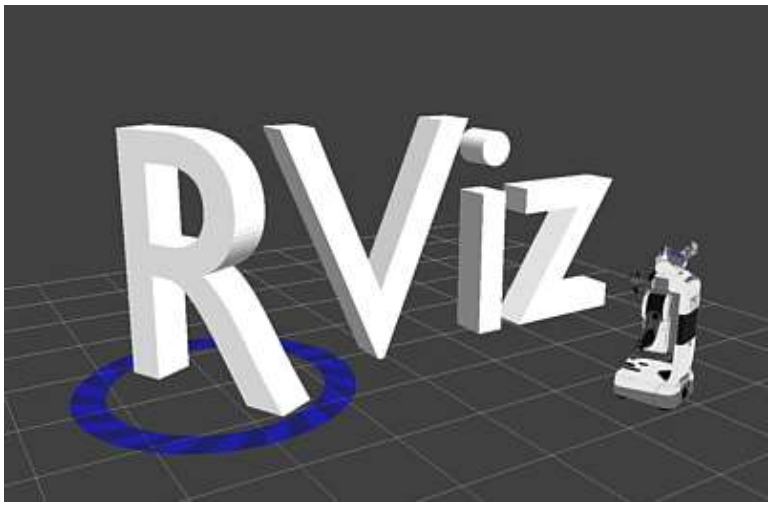
Hình 2.27: Phần mềm arduino IDE

2.5.2.3 Phần mềm Rviz

Rviz (viết tắt của *ROS Visualization*) là một công cụ trực quan hóa mạnh mẽ trong hệ sinh thái ROS 2, cho phép người dùng theo dõi và kiểm tra các dữ liệu cảm biến, trạng thái robot và thuật toán xử lý trong thời gian thực. Rviz hỗ trợ hiển thị nhiều loại dữ liệu khác nhau như bản đồ (Occupancy Grid), quỹ đạo, laser scan, hình ảnh từ camera, dữ liệu 3D từ Lidar, khung tọa độ (TF), và mô hình robot URDF. Đây là công cụ không thể thiếu khi phát triển và kiểm thử các hệ thống robot, giúp người dùng dễ

dàng phát hiện lỗi, đánh giá hoạt động và điều chỉnh các thuật toán điều hướng, nhận dạng hay SLAM.

Trong ROS 2, Rviz đã được nâng cấp với phiên bản mới có tên là Rviz2, tương thích với kiến trúc và chuẩn truyền thông DDS của ROS 2. Giao diện người dùng thân thiện, hỗ trợ kéo thả các thành phần trực quan và khả năng cấu hình cao giúp Rviz2 trở thành một phần mềm lý tưởng để giám sát trực quan hệ thống robot trong cả môi trường mô phỏng và thực tế. Nhờ Rviz, người phát triển có thể dễ dàng quan sát được robot đang di chuyển thế nào, cảm biến hoạt động ra sao, và các thuật toán xử lý dữ liệu có hiệu quả không – tất cả trong một không gian 3D sinh động và trực quan.



Hình 2.28: Phần mềm Rviz

2.5.2.4 Phần mềm Gazebo

Gazebo là một phần mềm mô phỏng robot mạnh mẽ, được thiết kế để tạo ra môi trường ảo ba chiều (3D) nhằm kiểm thử và phát triển các thuật toán điều khiển robot trước khi triển khai trên hệ thống thực tế. Trong hệ sinh thái ROS 2, Gazebo đóng vai trò như một "thế giới mô phỏng" nơi robot có thể di chuyển, cảm nhận môi trường và tương tác với vật thể thông qua các mô hình vật lý chính xác, cảm biến ảo (Lidar, Camera, IMU...) và mô hình robot URDF/SDF.

Gazebo cho phép người dùng kiểm tra các thuật toán định vị, SLAM, điều hướng, tránh vật và tương tác nhiều robot cùng lúc trong không gian ảo. Khi kết hợp với ROS 2, dữ liệu từ robot mô phỏng trong Gazebo có thể được truyền qua các topic, service và action để xử lý như một robot thật. Điều này giúp tiết kiệm thời gian, chi phí và giảm rủi ro khi phát triển hệ thống robot phức tạp.

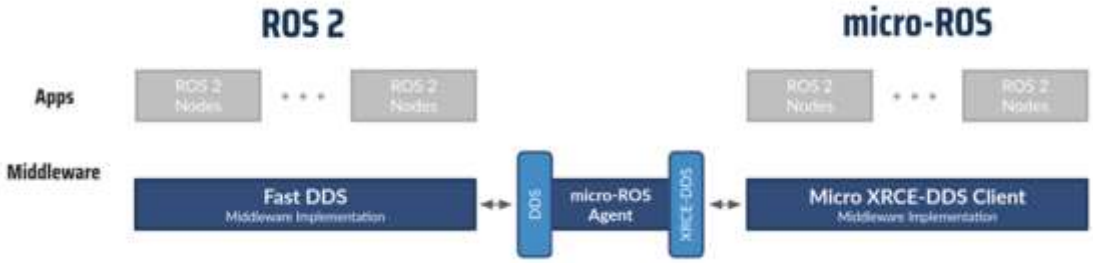
Nhờ khả năng tích hợp chặt chẽ với ROS 2, thư viện mô phỏng vật lý chân thực, giao diện trực quan và khả năng mô phỏng đa dạng robot cùng môi trường, Gazebo trở thành công cụ không thể thiếu trong nghiên cứu, giảng dạy và phát triển robot hiện đại.

Hình 2.29: Phần mềm Gazebo

2.6 Truyền thông giữa các thành phần trong hệ thống

2.6.1 Truyền thông giữ ESP32 với raspherry pi4 chạy ROS2

Lựa chọn giao thức micro-ROS: Giao thức micro-ROS (Micro Robot Operating System) là một phần mở rộng của ROS 2 dành cho các vi điều khiển (MCU) có tài nguyên hạn chế như ESP32, giúp chúng có thể giao tiếp với hệ sinh thái ROS 2 chạy trên máy chủ mạnh hơn như Raspberry Pi 4.



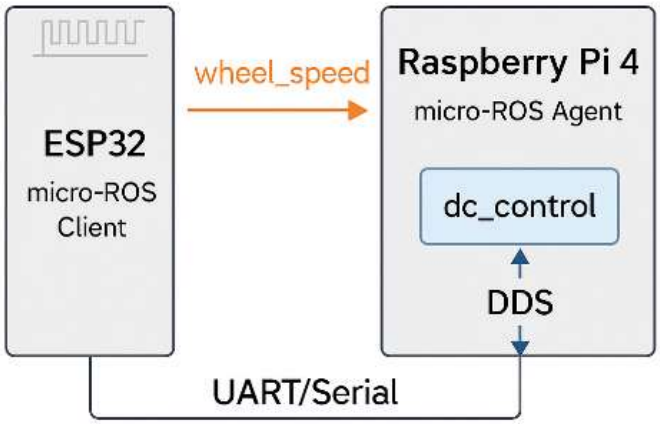
Hình 2.30: Sơ đồ truyền thông giữa ROS 2 và micro-ROS

Các tiêu chí lựa chọn micro-ROS:

Bảng 2.13: Tiêu chí lựa chọn micro-ROS

Tiêu chí	micro-ROS
Tương thích ROS 2	Có (chuẩn DDS)
Hiệu suất & Độ trễ	Thấp, phù hợp real-time
Dữ liệu có cấu trúc	Gửi Twist, Odometry, custom
Bảo trì & hỗ trợ	Được duy trì bởi cộng đồng ROS
Khả năng mở rộng	Có thể gửi/nhận nhiều topic, service

Kiến trúc truyền thông micro-ROS:



Hình 2.31: Kiến trúc truyền thông micro-ROS

Trong kiến trúc, ESP32 đóng vai trò là một micro-ROS Client, thu thập tốc độ bánh xe và góc quay từ cảm biến của robot, định kỳ gửi dữ liệu qua topic wheel_speed thông qua giao tiếp UART. Raspberry Pi 4 chạy micro-ROS Agent để nhận dữ liệu từ ESP32 và đưa vào mạng ROS 2 thông qua giao thức DDS. Node dc_control trên Raspberry Pi 4 sẽ subscribe topic wheel_speed để xử lý điều khiển động cơ, giúp robot hoạt động chính xác theo yêu cầu điều hướng và kiểm soát vận tốc.

2.6.2 Giao tiếp giữa raspherry pi4 chạy ROS2 với máy tính operator

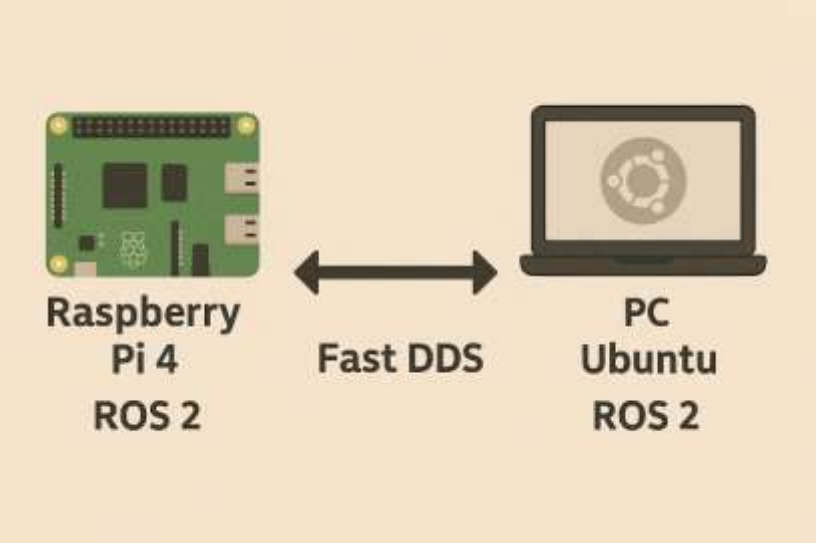
Lựa chọn giao thức DDS (Data Distribution Service) là một giao thức truyền thông theo mô hình publish-subscribe phân tán, được chuẩn hóa bởi OMG (Object Management Group). Nó cho phép các thiết bị hoặc ứng dụng phân tán trao đổi dữ liệu real-time, reliable, và scalable mà không cần biết nhau trực tiếp. DDS cũng là giao thức được hỗ trợ chính thức trong ROS2 giúp các hệ thống ROS2 giao tiếp với nhau một cách hiệu quả.

Bảng 2.14: So sánh các loại giao thức DDS

Tiêu chí	Fast DDS	Cyclone DDS	RTI Connext	Gurum DDS
Mã nguồn mở	✓	✓	✗	✗
Hiệu suất cao	✓	✓	✓	✓
Hỗ trợ QoS đầy đủ	✓	✓	✓	✓
Multicast hiệu quả	✓	✓	✓	✓

Hỗ trợ tốt cho ROS 2	✓	✓	✗	✗
Tài liệu & cộng đồng mạnh	✓	✓	✗	✗
Hỗ trợ real-time	✓	✓	✓	✓
Tương thích micro-ROS	✓	✗	✗	✗
Cài đặt dễ dàng	✓	✓	✗	✗

Từ bảng so sánh thì Fast DDS là lựa chọn tốt nhất cho ROS 2 vì nó là mã nguồn mở, hiệu suất cao, hỗ trợ đầy đủ QoS và multicast, được hỗ trợ chính thức trong ROS 2, tương thích tốt với micro-ROS và dễ dàng cài đặt.



Hình 2.32: Truyền thông Fast DDS

Kiến trúc giao tiếp giữa Raspberry Pi 4 và máy tính PC Ubuntu, cả hai đều chạy ROS 2 và sử dụng Fast DDS để truyền dữ liệu theo mô hình publish-subscribe. Với Domain_ID = 0, cả hai thiết bị cùng nằm trong một miền truyền thông chung, cho phép các node ROS 2 trên Raspberry Pi và PC tự động phát hiện và trao đổi thông tin mà không cần cấu hình địa chỉ thủ công. Fast DDS đảm nhận việc quản lý việc gửi nhận message một cách hiệu quả, đáng tin cậy và theo thời gian thực, giúp đồng bộ hóa dữ liệu cảm biến, điều khiển và các topic ROS 2 khác giữa hai hệ thống một cách liền mạch.

CHƯƠNG 3. THIẾT KẾ VÀ THI CÔNG MÔ HÌNH

3.1 Thiết kế khung xe và cơ cấu cơ khí

Khung xe được thiết kế theo dạng mô-đun nhiều tầng, tương tự với cấu trúc của dòng robot di động phổ biến TurtleBot3 Burger. Thiết kế này mang lại tính linh hoạt cao trong việc lắp ráp, tháo rời, và mở rộng các tầng chứa cảm biến, bo mạch điều khiển hoặc các thiết bị ngoại vi khác.

Cấu trúc tổng thể khung xe gồm ba tầng chính:

- Tầng dưới cùng: Gắn động cơ, bánh xe và các encoder.
- Tầng giữa: Gắn các bo mạch vi điều khiển, driver động cơ, và mạch nguồn và cảm biến IMU.
- Tầng trên cùng: Lắp cảm biến Lidar

Các tầng được kết nối với nhau bằng trụ lục giác, giúp tạo ra khoảng cách cố định, giữ khung chắc chắn, đồng thời tạo không gian cho dây dẫn và thiết bị điện tử bên trong.

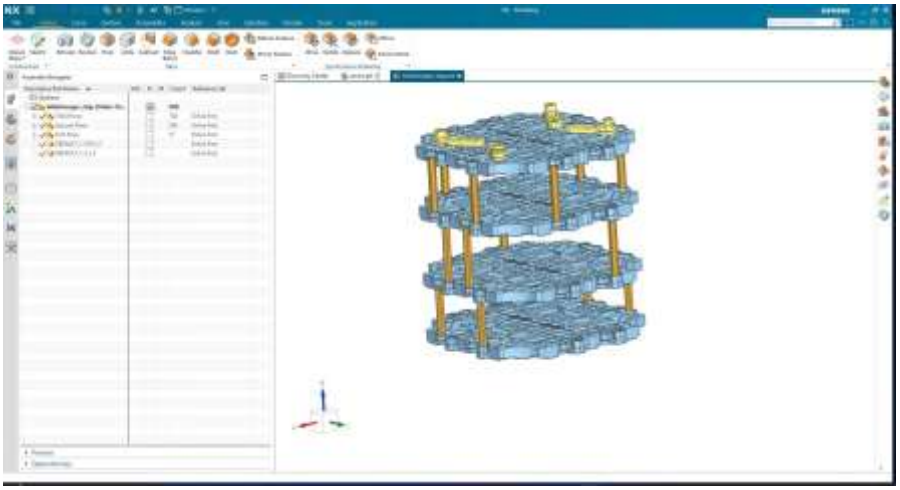
Chất liệu và phương pháp chế tạo: Khung được in bằng nhựa ABS thông qua công nghệ in 3D, đảm bảo độ cứng vững vừa đủ, trọng lượng nhẹ và khả năng gia công nhanh chóng.

Đặc điểm thiết kế:

- Các tấm đỡ đều được thiết kế sẵn các lỗ tròn và khe rãnh để bắt vít các linh kiện như động cơ, cảm biến, bo mạch,...
- Giá đỡ cảm biến LiDAR được thiết kế riêng biệt, có khả năng điều chỉnh vị trí nhằm đạt được góc quét phù hợp.
- Thiết kế dạng tầng tạo điều kiện cho việc mở rộng phần cứng và thay thế linh kiện dễ dàng.

3.1.1 Phần mềm thiết kế NX

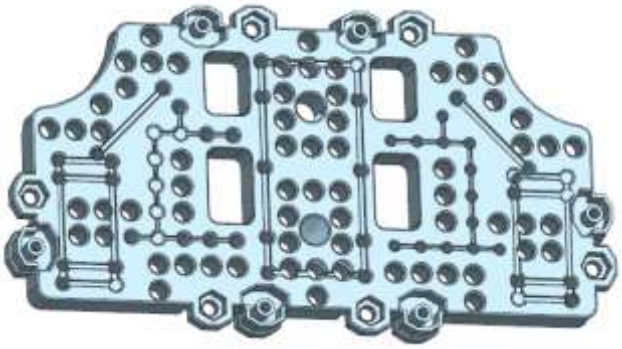
Phần mềm thiết kế 3D NX là một trong những công cụ mạnh mẽ và phổ biến trong lĩnh vực thiết kế cơ khí và robot. Được phát triển bởi Siemens PLM Software, NX cung cấp một loạt các tính năng và công cụ tiên tiến để tạo và quản lý mô hình 3D, từ việc tạo ra các bộ phận đơn lẻ cho đến thiết kế toàn bộ hệ thống phức tạp. Trong đồ án, phần mềm này được dùng để thiết kế khung xe, giá đỡ cảm biến và mô hình tổng thể trước khi tiến hành thi công thực tế.



Hình 3.1: Phần mềm thiết kế NX-Siemens

3.1.2 Thân đỡ chính

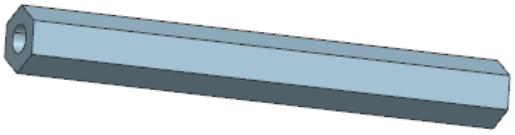
Phần thân đỡ này đóng vai trò khung chính, được thiết kế với nhiều lỗ tròn và hình chữ nhật để lắp ráp và cố định các bộ phận khác. Giúp kết nối và cố định các bộ phận khác của robot, đảm bảo độ chắc chắn và ổn định khi hoạt động. Chất liệu bằng nhựa abs in 3D.



Hình 3.2: Mảnh thân đỡ

3.1.3 Trụ lục giác chống đỡ

Ống lục giác này được sử dụng như một trụ đỡ, kết nối các tấm thân đỡ (như tấm trong hình trước) bằng cách bắt vít qua các lỗ tròn trên tấm. Nó giữ các tấm song song và cố định, đồng thời tạo không gian cho các linh kiện bên trong, như bo mạch hoặc dây cáp. Bộ phận này đóng vai trò quan trọng trong việc duy trì cấu trúc khung của robot, đảm bảo các tầng của robot được lắp ráp chắc chắn và chính xác.



Hình 3.3: Trụ lục giác

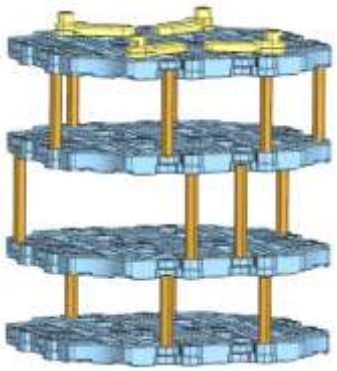
3.1.4 Giá đỡ cho chân cảm biến Lidar

Giá đỡ có dạng thanh phẳng, một đầu có lỗ tròn lớn với ren để bắt vít cố định cảm biến LiDAR, đầu còn lại có một khe hở hình chữ nhật dài để điều chỉnh vị trí và gắn vào khung thân robot.

Hình 3.4: Giá đỡ cảm biến

3.1.5 Mô hình khung 3D hoàn chỉnh

Sau khi lắp ráp các chi tiết sẽ cho ra khung hệ thống hoàn chỉnh trên không gian 3D



Hình 3.5: Khung 3D hoàn chỉnh

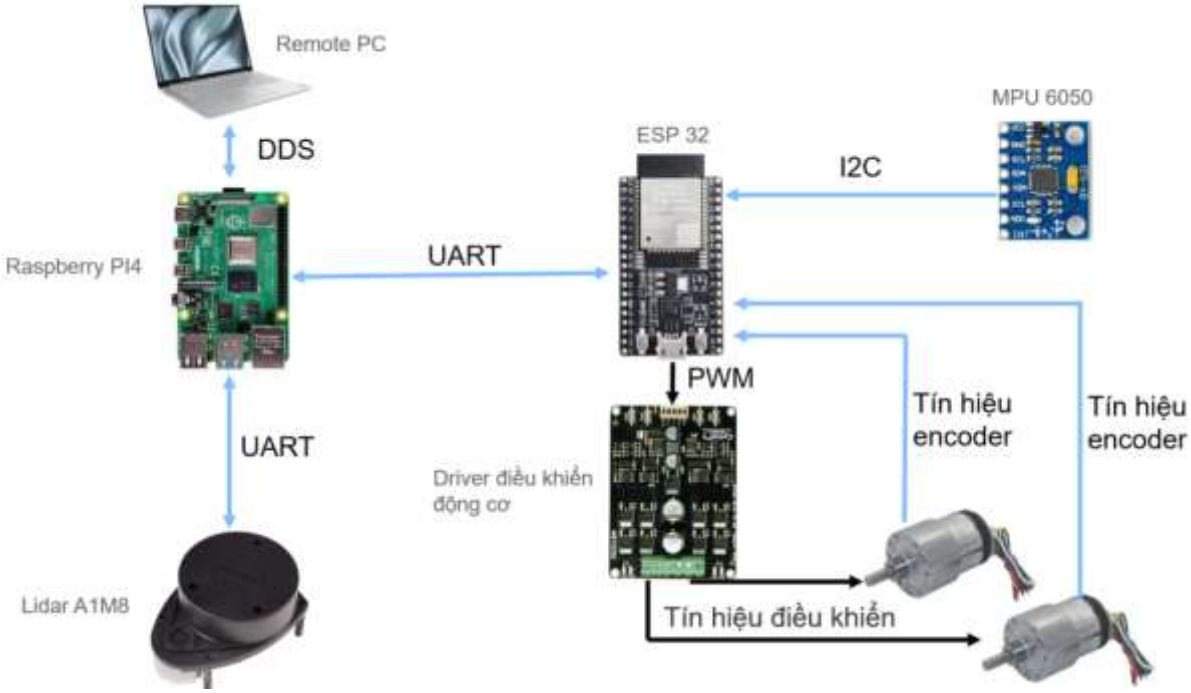
3.2 Sơ đồ kết nối hệ thống

Sau khi lựa chọn và xác định các thiết bị cần thiết, hệ thống được thiết kế với một sơ đồ kết nối tổng thể nhằm mô tả trực quan mối liên kết giữa các thành phần phần cứng chính. Sơ đồ này giúp minh họa rõ ràng cách thức các thiết bị giao tiếp, truyền tín hiệu và phối hợp hoạt động trong quá trình điều khiển và thu thập dữ liệu của robot.

Các thành phần chính trong sơ đồ bao gồm:

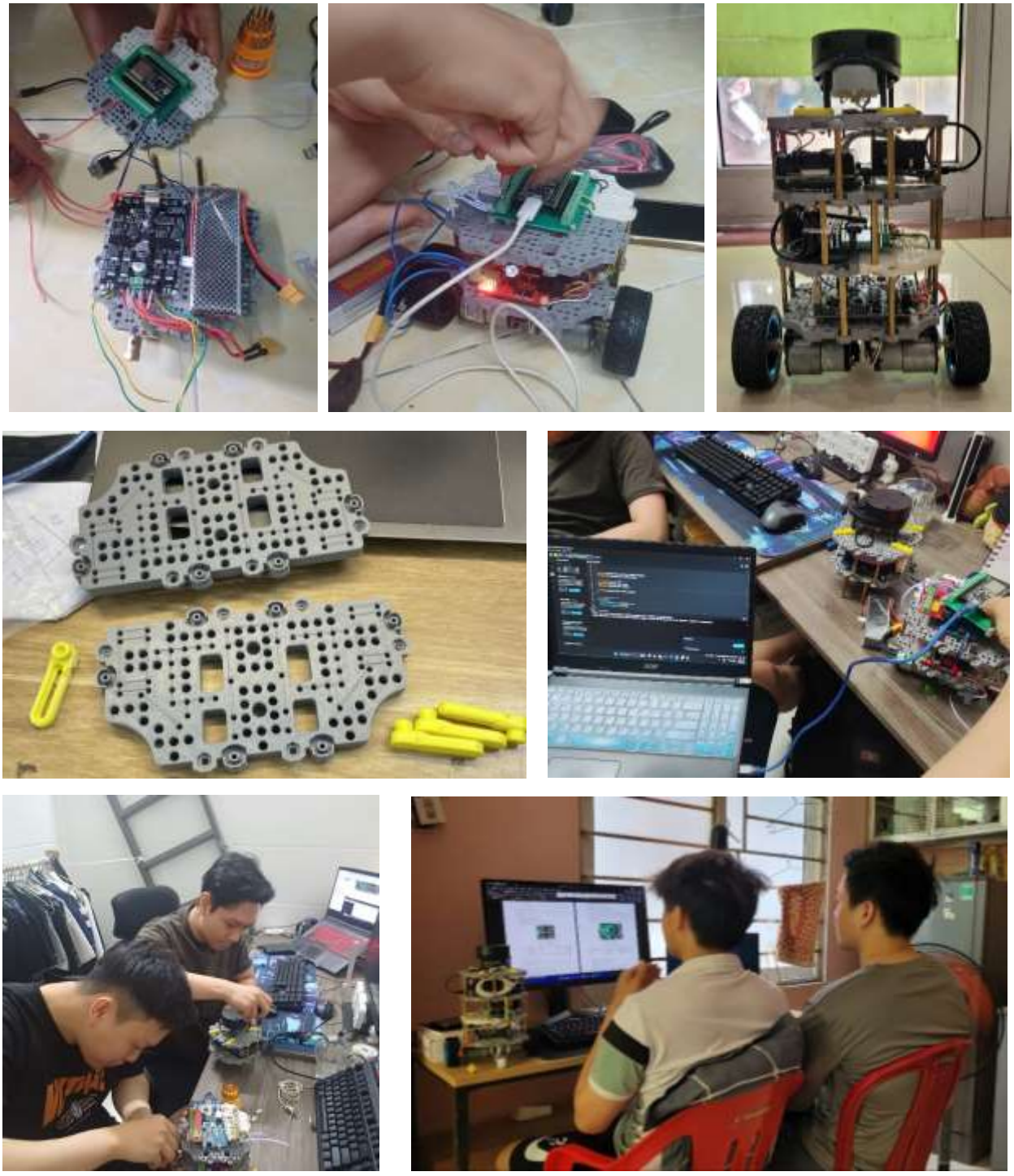
- **Raspberry Pi 4** đóng vai trò như bộ xử lý trung tâm chạy hệ thống ROS2, chịu trách nhiệm thu thập dữ liệu từ cảm biến LiDAR, điều khiển và giám sát vi điều khiển ESP32.
- **ESP32** giao tiếp Micro-ROS với ROS2, thực hiện các tác vụ điều khiển tốc độ động cơ, xử lý tín hiệu từ encoder, đọc dữ liệu từ cảm biến MPU6050 và truyền tốc độ thực tế về máy chủ ROS2 qua UART.

- **Driver điều khiển động cơ** nhận tín hiệu điều khiển dạng PWM từ ESP32 để điều khiển hai động cơ DC có encoder phản hồi.
- **Encoder tích hợp trong động cơ** gửi tín hiệu xung phản hồi về ESP32 để tính toán tốc độ thực tế.
- **Cảm biến MPU6050** kết nối với ESP32 thông qua giao thức I2C để cung cấp thông tin về góc quay và gia tốc.
- **Cảm biến LiDAR A1M8** truyền dữ liệu bản đồ và khoảng cách về Raspberry Pi 4 thông qua giao tiếp UART.
- **Remote PC** được sử dụng để điều khiển và giám sát từ xa, giao tiếp với Raspberry Pi thông qua giao thức DDS (Data Distribution Service) của ROS2.



Hình 3.6: Sơ đồ kết nối hệ thống

3.3 Thi công và hoàn thiện mô hình



Hình 3.7: Thi công và lắp đặt mô hình

CHƯƠNG 4. XÂY DỰNG THUẬT TOÁN VÀ MÔI TRƯỜNG ĐIỀU KHIỂN GIÁM SÁT

4.1 Khởi tạo toàn bộ hệ thống

Để triển khai phần mềm điều khiển robot, trước tiên cần tiến hành thiết lập môi trường làm việc cho cả máy tính điều khiển từ xa (Remote PC) và thiết bị điều khiển biên (Edge Device) gắn trên robot. Cả hai thiết bị đều sử dụng hệ điều hành Ubuntu 22.04 và được cấu hình với nền tảng ROS2 Humble. Toàn bộ quá trình cài đặt và cấu hình môi trường được trình bày chi tiết tại phần Phụ lục.

4.2 Triển khai thuật toán Graph-Based SLAM

Lưu đồ thuật toán giải nghiệm tối ưu thuật toán Graph-Based SLAM

Đầu vào: $X^{*} = X_{1:T}, C = \{e_{ij}, \Omega_{ij}\}$

Đầu ra: X^{*}, H^{*}

While ¬converged **do**

$b \leftarrow 0, H \leftarrow 0$

For all $\{e_{ij}, \Omega_{ij}\} \in C$ **do**

// Tính Jacobian của hàm sai lệch

$$A_{ij} \leftarrow \left.\frac{\partial e_{ij}(X)}{\partial X_i}\right|_{X=X}, B_{ij} \leftarrow \left.\frac{\partial e_{ij}(X)}{\partial X_j}\right|_{X=X}$$

// Tính toán ma trận thông tin

$$H_{[ii]} += A_{ij}^{T} \Omega_{ij} A_{ij}$$

$$H_{[ij]} += A_{ij}^{T} \Omega_{ij} B_{ij}$$

$$H_{[ji]} += B_{ij}^{T} \Omega_{ij} A_{ij}$$

$$H_{[jj]} += B_{ij}^{T} \Omega_{ij} B_{ij}$$

// Tính vector hệ số

$$b_{[i]} += A_{ij}^{T} \Omega_{ij} e_{ij}$$

$$b_{[j]} += B_{ij}^{T} \Omega_{ij} e_{ij}$$

End for

// Giữ cố định nút đầu tiên

$$H_{[11]} += I$$

// Giải nghiệm

$\Delta X \leftarrow solve(H\,\Delta X = -b)$

// Cập nhật thông số

$X \mathrel{+}= \Delta X$

End while

$X^{*} \leftarrow X$

$H^{*} = H$

// Bỏ cố định nút đầu tiên

$H^{*}_{[11]} -= I$

Return $\langle X^{*}, H^{*} \rangle$

Thuật toán trên tóm tắt quy trình lặp Gauss-Newton nhằm xác định giá trị trung bình và ma trận thông tin của phân phối hậu nghiệm (posterior) đối với các pose (vị trí và hướng) của robot.

Mỗi cạnh đóng góp vào:

- Các khối của ma trận Hessian: $H_{[ii]}, H_{[ij]}, H_{[ji]}, H_{[jj]}$
- Các phần tử tương ứng trong vector hệ số: $b_{[i]}, b_{[j]}$

Ngoài ra, vì ma trận Hessian là đối xứng, nên ta chỉ cần tính phần tam giác trên của ma trận để tiết kiệm tài nguyên tính toán.

Sai số của một ràng buộc e_{ij} chỉ phụ thuộc vào vị trí tương đối giữa hai vị trí X_i và X_j. Do đó, tổng sai số $F(X)$ của toàn bộ hệ thống là bất biến đối với phép biến đổi cứng (rigid transformation) áp dụng lên toàn bộ các vị trí.

Vì vậy, phương trình cần giải sẽ bị thiếu xác định (underdetermined). Để giải hệ tuyến tính một cách ổn định, ta thường ràng buộc một phần tử ΔX_k bằng 0, nhằm "cố định" một phần trong hệ thống.

Cách phổ biến là:

- Cộng ma trận đơn vị vào khối đường chéo $H[kk]$ để khóa ΔX_k.
- Trong thuật toán trên, nút đầu tiên X_1 được cố định mà không làm mất tính tổng quát.

4.3 Viết file urdf để mô tả robot trong hệ thống

File URDF (Unified Robot Description Format) được sử dụng để mô tả mô hình vật lý và hình học của robot trong các hệ thống ROS bao gồm: Mô tả hình dạng, mô tả va chạm, mô tả quán tính, mô tả khớp nối, xác định hệ tọa độ.

Bảng 4.1: Tham số URDF thân Robot

Tham số	Giá trị	Mô tả
Base collision box (Dài)	0.140 m	Kích thước chiều dài base link
Base collision box (Rộng)	0.140 m	Kích thước chiều rộng base link
Base collision box (Cao)	0.165 m	Kích thước chiều cao base link
Base link mass	0.82573504 kg	Khối lượng base link

Bảng 4.2: Tham số URDF bánh điều hướng chính

Tham số	Giá trị	Mô tả
Wheel collision (Đường kính)	0.065 m	Đường kính bánh xe
Wheel collision (Bán kính)	0.0325 m	Bán kính bánh xe
Wheel collision (Độ dày)	0.025 m	Độ dày bánh xe
Wheel mass	0.02849894 kg	Khối lượng bánh xe
Khoảng cách giữa 2 tâm bánh	0.18 m	Khoảng cách giữa tâm bánh trái và phải

Bảng 4.3: Tham số URDF bánh phụ

Tham số	Giá trị	Mô tả
Caster box (Dài)	0.030 m	Chiều dài caster
Caster box (Rộng)	0.009 m	Chiều rộng caster
Caster box (Cao)	0.020 m	Chiều cao caster
Caster mass	0.005 kg	Khối lượng caster

Bảng 4.4: Tham số URDF cảm biến Lidar

Tham số	Giá trị	Mô tả
LiDAR collision (Đường kính)	0.11 m	Đường kính LiDAR
LiDAR collision (Bán kính)	0.055 m	Bán kính LiDAR
LiDAR collision (Chiều cao)	0.0315 m	Chiều cao LiDAR

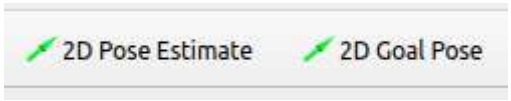
LiDAR mass	0.114 kg	Khối lượng LiDAR

4.4 Tham số cấu hình điều hướng (Navigation 2)

Navigation Stack trong ROS2 (thường được sử dụng trong các phiên bản như ROS1 hoặc ROS2) là một bộ công cụ phần mềm giúp robot tự động di chuyển từ điểm A đến điểm B trong môi trường không xác định, tránh chướng ngại vật và tối ưu hóa đường đi. Navigation Stack bao gồm các thành phần chính sau:

- **Localization (Định vị)**: Sử dụng thuật toán như AMCL (Adaptive Monte Carlo Localization) để xác định vị trí của robot trong bản đồ dựa trên dữ liệu từ cảm biến (ví dụ: LiDAR, encoder).
- **Mapping (Lập bản đồ)**: Sử dụng SLAM (Simultaneous Localization and Mapping) hoặc Gmapping để tạo bản đồ 2D/3D của môi trường, dựa trên dữ liệu từ cảm biến LiDAR.
- **Path Planning (Lập kế hoạch đường đi)**: Sử dụng thuật toán để tìm đường đi tối ưu từ vị trí hiện tại đến mục tiêu.
- **Motion Control (Điều khiển chuyển động)**: Tạo các lệnh tốc độ tuyến tính (v) và tốc độ góc (ω) để gửi đến hệ thống điều khiển robot, đảm bảo di chuyển chính xác.

Đầu ra của Navigation package sẽ là các lệnh vận tốc sẽ điều khiển robot đến vị trí đích đã cho.



Hình 4.1: Công cụ Nav2 trong Rviz

Navigation stack triển khai các thuật toán chuẩn, chẳng hạn như SLAM, EKF-SLAM, A *(star), Dijkstra, amcl, v.v., có thể được sử dụng trực tiếp trong ứng dụng.

Trong hệ thống ROS2, Navigation 2 (Nav2) là framework chính dùng để điều khiển robot di chuyển trong môi trường đã biết hoặc chưa biết. Để Nav2 hoạt động hiệu quả, ta cần cấu hình một số tham số quan trọng trong các node khác nhau. Dưới đây là phần trình bày các tham số cấu hình chính thường được sử dụng trong Nav2:

Bảng 4.5: Tham số cấu hình Nav2

Tham số	Giá trị	Mô tả
FollowPath.max_vel_x	0.26 (m/s)	Tốc độ tịnh tiến tối đa
FollowPath.min_vel_x	0.0 (m/s)	Tốc độ tịnh tiến tối thiểu
FollowPath.max_vel_y	0.0 (m/s)	Tốc độ y tối đa
FollowPath.min_vel_y	0.0 (m/s)	Tốc độ y tối thiểu
FollowPath.max_vel_theta	1.0 (rad/s)	Tốc độ quay tối đa

FollowPath.min_speed_theta	0.0 (rad/s)	Tốc độ quay tối thiểu
FollowPath.max_speed_xy	0.26 (m/s)	Tốc độ tịnh tiến tối đa (tuyệt đối)
FollowPath.min_speed_xy	0.0 (m/s)	Tốc độ tịnh tiến tối thiểu (tuyệt đối)
FollowPath.acc_lim_x	2.5 (m/s²)	Giới hạn gia tốc theo x
FollowPath.acc_lim_y	0.0 (m/s²)	Giới hạn gia tốc theo y
FollowPath.acc_lim_theta	3.2 (rad/s²)	Giới hạn gia tốc quay
FollowPath.decel_lim_x	-2.5 (m/s²)	Giới hạn giảm tốc theo x
FollowPath.decel_lim_y	0.0 (m/s²)	Giới hạn giảm tốc theo y
FollowPath.decel_lim_theta	-3.2 (rad/s²)	Giới hạn giảm tốc quay
FollowPath.xy_goal_tolerance	0.05 (m)	Khoảng dung sai vị trí x,y so với đích
FollowPath.yaw_goal_tolerance	0.25 (rad)	Khoảng dung sai góc quay so với đích
FollowPath.transform_tolerance	0.2 (s)	Khoảng trễ cho phép của tf

4.5 Cấu hình giao thức truyền thông Micro-Ros

4.5.1 Cấu hình firmware cho esp32

Trỏ vào môi trường Ros2 đã cài đặt

```
source /opt/ros/humble/setup.bash
```

Tạo không gian làm việc và tải xuống các công cụ micro-ROS

```
mkdir microros_ws
cd microros_ws
git clone -b humble https://github.com/micro-ROS/micro_ros_setup.git
src/micro_ros_setup
```

Cập nhật các phụ thuộc sử dụng rosdep

```
sudo apt update && rosdep update
rosdep install --from-paths src --ignore-src -y
```

Tải pip và biên dịch chương trình

```
sudo apt-get install python3-pip
colcon build
source install/local_setup.bash
```

Tạo firmware mới

```
ros2 run micro_ros_setup create_firmware_ws.sh host
```

Biên dịch firmware vừa tạo

```
ros2 run micro_ros_setup build_firmware.sh
source install/local_setup.bash
```

4.5.2 Cấu hình Agent trên Ros2

Tải các gói Micro-Ros-Agent

```
ros2 run micro_ros_setup create_agent_ws.sh
```

Sau khi cài xong thì tiến hành biên dịch

```
ros2 run micro_ros_setup build_agent.sh
source install/local_setup.bash
```

4.5.3 Kết nối esp32 và Ros2

Sau khi đã cài đặt thành công firmware và Agent thì có thể tiến hành kết nối thông qua cổng serial. Để cấp quyền truy cập micro-ROS vào không gian dữ liệu ROS 2 chạy lệnh:

```
ros2 run micro_ros_agent micro_ros_agent serial --dev /dev/ttyUSB1
```

Nếu thành công sẽ nhận được các thông báo như sau:

Thông báo trên Ros2	Hoạt động	Mô tả
create_client	Kết nối client	Vi điều khiển (MCU) kết nối thành công tới Agent
session established	Thiết lập phiên	Tạo phiên làm việc giữa MCU và Agent
create_participant	Tạo participant	MCU tham gia hệ thống DDS như 1 thực thể
create_topic	Tạo topic	Tạo chủ đề để truyền/nhận dữ liệu
create_publisher /create_datawriter	Thiết lập publish	MCU gửi dữ liệu lên topic
create_subscriber /create_datareader	Thiết lập subscribe	MCU nhận dữ liệu từ topic

4.6 Cấu hình truyền thông DDS

Để thiết lập truyền thông hiệu quả giữa Raspberry Pi 4 và Remote PC bằng DDS, cần cấu hình các tham số cơ bản bao gồm vai trò Publisher/Subscriber, Domain ID, và mạng WLAN. Bảng 4.6 trình bày chi tiết cấu hình này để đảm bảo kết nối ổn định và truyền dữ liệu chính xác trong hệ thống.

Bảng 4.6: Cấu hình truyền thông DDS

	Raspberry pi4	Remote PC
Node	Publisher/Subscriber	Subscriber/ Publisher
Domain ID	0	0
Network	WLAN	WLAN

4.7 Cấu hình tham số và publish dữ liệu lidar

Để tích hợp và sử dụng cảm biến LiDAR A1M8 trong hệ thống robot tự hành cần thiết lập các tham số cấu hình phù hợp nhằm đảm bảo thiết bị hoạt động ổn định và cung cấp dữ liệu chính xác cho hệ thống ROS. Các tham số này bao gồm cấu hình cổng kết nối, tốc độ truyền dữ liệu, chế độ quét, góc và khoảng cách quét, cũng như các thiết lập bổ trợ như bù góc và hướng quay dữ liệu. Bảng dưới đây trình bày chi tiết các tham số cấu hình cho LiDAR A1M8 trong môi trường ROS2.

Bảng 4.7: Cấu hình tham số cảm biến Lidar

Tham số	**Giá trị đề xuất**	**Kiểu dữ liệu**	**Ý nghĩa**
serial_port	/dev/ttyUSB0	string	Đường dẫn thiết bị của cổng serial nơi LiDAR được kết nối
serial_baudrate	115200	int	Tốc độ truyền dữ liệu qua UART
frame_id	laser hoặc base_scan	string	Tên khung TF dùng cho dữ liệu LaserScan
inverted	false	bool	Nếu true, đảo chiều quay dữ liệu quét
angle_compensate	true	bool	Bù góc nhằm làm đều mật độ tia quét
scan_mode	Standard	string	Chế độ quét: Standard là duy nhất hỗ trợ tốt trên A1M8
topic	/scan	string	Tên topic ROS để publish dữ liệu sensor_msgs/msg/LaserScan
range_min	0.15	float	Khoảng cách tối thiểu có thể đo được (m)
range_max	12.0	float	Khoảng cách tối đa có thể đo được (m)
angle_min	-3.14159	float (rad)	Góc bắt đầu quét
angle_max	3.14159	float (rad)	Góc kết thúc quét
scan_rate	10.0	float (Hz)	Tần suất quét

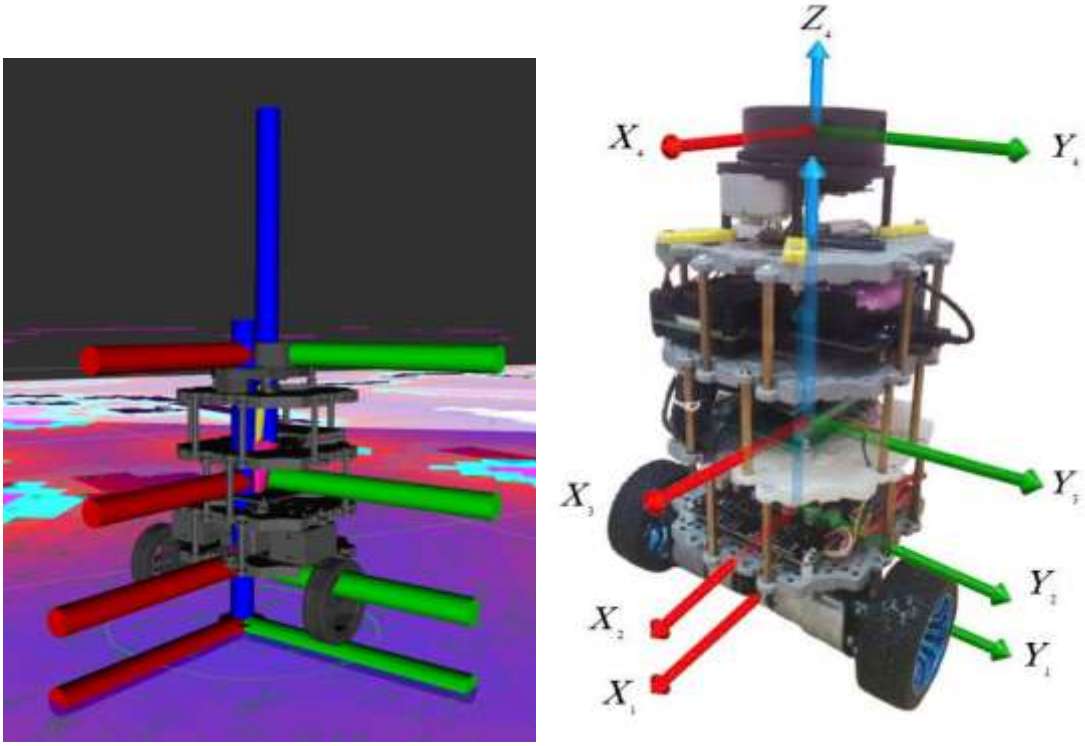
4.8 TF trong ROS2

Trong các hệ thống robot di động, việc xác định mối quan hệ không gian giữa các bộ phận như thân robot, cảm biến (LiDAR, camera), bánh xe, hoặc các điểm mốc là rất quan trọng để robot có thể cảm nhận, hiểu và tương tác chính xác với môi trường xung quanh.

TF (Transform) là một thành phần cốt lõi trong ROS 2, cho phép theo dõi hệ thống các hệ tọa độ (coordinate frames) và mối quan hệ hình học (biến đổi không gian) giữa chúng theo thời gian thực.

Mục đích chính của TF là quản lý các hệ tọa độ động, cho phép các node khác trong hệ thống truy xuất vị trí tương đối giữa các thành phần. Đồng thời hỗ trợ các bài toán điều hướng, định vị, cảm biến, thị giác máy tính và tích hợp cảm biến

4.8.1 Các hệ tọa độ chính trên robot



Hình 4.2: Hệ tọa độ robot

Hệ tọa độ *base_link* $(X_3 Y_3 Z_3)$*:* Là hệ tọa độ trung tâm, gắn trực tiếp với thân robot. Gốc tọa độ thường được đặt tại tâm của robot. Đây là hệ quy chiếu chính để biểu diễn vận tốc, hướng di chuyển, và vị trí tương đối của các bộ phận khác trên robot.

Hệ tọa độ *base_scan* $(X_4 Y_4 Z_4)$*:* Đây là hệ tọa độ gắn với cảm biến LiDAR hoặc các thiết bị quét laser 2D. Nó xác định chính xác vị trí và hướng của cảm biến so với

base_link, giúp chuyển đổi dữ liệu quét sang tọa độ robot để xử lý bản đồ và tránh vật cản.

Hệ tọa độ *IMU_link* $(X_2Y_2Z_2)$*:* Gắn với cảm biến IMU (Inertial Measurement Unit), dùng để đo góc quay, vận tốc góc và gia tốc tuyến tính. Hệ tọa độ này giúp xác định trạng thái chuyển động và định hướng của robot trong không gian.

Hệ tọa độ *map* $(X_1Y_1Z_1)$: Là hệ tọa độ tĩnh, cố định với môi trường xung quanh. Được sử dụng trong các thuật toán định vị và SLAM (Simultaneous Localization and Mapping), giúp robot biết vị trí của mình trong không gian toàn cục (global position).

Trong quá trình điều khiển robot di chuyển, tất cả các thông tin về vị trí, vận tốc, hướng đi và dữ liệu cảm biến đều cần được quy đổi về hệ tọa độ *base_link*. Điều này đảm bảo:

- Việc lập kế hoạch đường đi và tránh vật cản diễn ra đúng theo hướng di chuyển thực tế của robot.
- Các thuật toán điều khiển và phản hồi hoạt động chính xác trong không gian gắn với robot.
- Việc tích hợp dữ liệu từ nhiều cảm biến (LiDAR, IMU, camera...) diễn ra thống nhất và đồng bộ.

4.8.2 Công thức chuyển đổi giữa các hệ trên robot

Để chuyển một điểm P_A từ hệ tọa độ A sang hệ B, ta sử dụng ma trận biến đổi đồng nhất:

$$P_B = T_{A\to B}.P_A \tag{4.1}$$

Trong đó: P_A, P_B **:**Là các tọa độ điểm trong hệ A và B có dạng $\left[x, y, z, 1\right]^T$

$T_{A\to B}$ **:**Là ma trận biến đổi 4x4, bao gồm phép quay và tịnh tiến

Ma trận biến đổi đồng nhất có dạng:

$$T_{A\to B} = \begin{bmatrix} R & t \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.2}$$

- Chuyển đổi từ *map* về *base_link:*

$$P_{base\_link} = T_{map\to base\_link}.P_{map} \tag{4.3}$$

- Chuyển đổi từ *base_scan* về *base_link:*

$$P_{base\_link} = T_{base\_scan\to base\_link}.P_{base\_scan} \tag{4.4}$$

- Chuyển đổi từ *IMU_link* về *base_link:*

$$P_{base\_link} = T_{IMU\_link \to base\_link} . P_{IMU\_link} \quad (4.5)$$

4.9 Thiết kế bộ điều khiển cho động cơ

Động cơ DC là thành phần cốt lõi trong hệ thống truyền động của xe tự hành hai bánh, đảm nhiệm vai trò chuyển đổi năng lượng điện thành chuyển động cơ học để điều khiển tốc độ và hướng di chuyển của xe. Trong bối cảnh nhà kho, nơi yêu cầu xe di chuyển chính xác và ổn định trên các lối đi hẹp, việc mô hình hóa động cơ DC là bước quan trọng để hiểu rõ đặc tính động học và thiết kế bộ điều khiển hiệu quả.

4.9.1 Mô hình hóa động cơ

Do động cơ DC được sử dụng trong đồ án này không có thông số kỹ thuật từ nhà sản xuất, phương pháp nhận dạng hệ thống (System Identification) được áp dụng để xác định mô hình toán học dựa trên dữ liệu thực nghiệm.

Để xác định đặc tính của động cơ, hai động cơ DC được thử nghiệm bằng cách cấp điện áp 12V và đo tốc độ quay của chúng thông qua cảm biến tốc độ (encoder). Dữ liệu tốc độ được ghi lại với tần số lấy mẫu 10 Hz để đảm bảo độ chính xác trong việc mô tả đáp ứng động học. Quá trình đo được thực hiện trong điều kiện không tải để loại bỏ ảnh hưởng của mô-men tải bên ngoài.

```
From input "u1" to output "y1":
          1698
  ---------------------
  s^2 + 17.22 s + 59.59
Name: DC1
Continuous-time identified transfer function.

Parameterization:
   Number of poles: 2   Number of zeros: 0
   Number of free coefficients: 3
   Use "tfdata", "getpvec", "getcov" for parameters and their uncertainties.

Status:
Estimated using TFEST on time domain data "DC1".
Fit to estimation data: 96.13% (stability enforced)
```

Hình 4.3: Nhận dạng hàm truyền động cơ 1

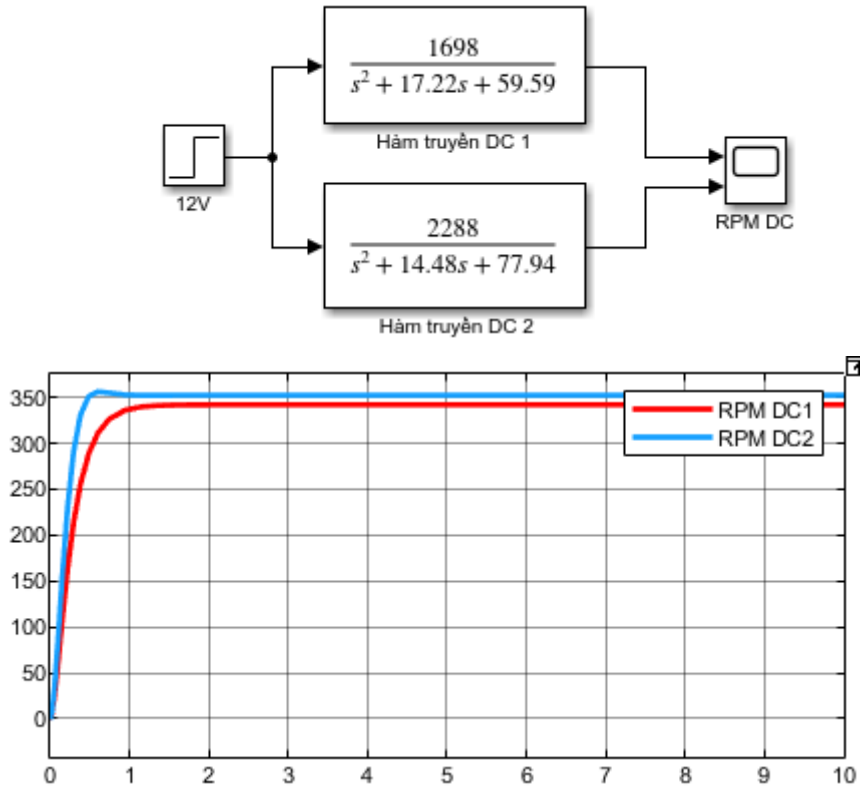
```
  From input "u1" to output "y1":
           2288
  ---------------------
  s^2 + 14.48 s + 77.94
Name: DC2
Continuous-time identified transfer function.

Parameterization:
   Number of poles: 2   Number of zeros: 0
   Number of free coefficients: 3
   Use "tfdata", "getpvec", "getcov" for parameters and their uncertainties.

Status:
Estimated using TFEST on time domain data "DC2".
Fit to estimation data: 98.37% (data prefiltered, stability enforced)
```

Hình 4.4: Nhận dạng hàm truyền động cơ 2

Kết quả nhận dạng hai động cơ có độ chính xác lớn hơn 96% , chứng tỏ mô hình bậc 2 phù hợp để mô tả đặc tính động học của động cơ trong điều kiện thử nghiệm.

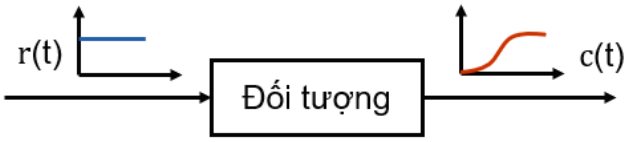


Hình 4.5: Mô phỏng tốc độ động cơ nhận dạng được

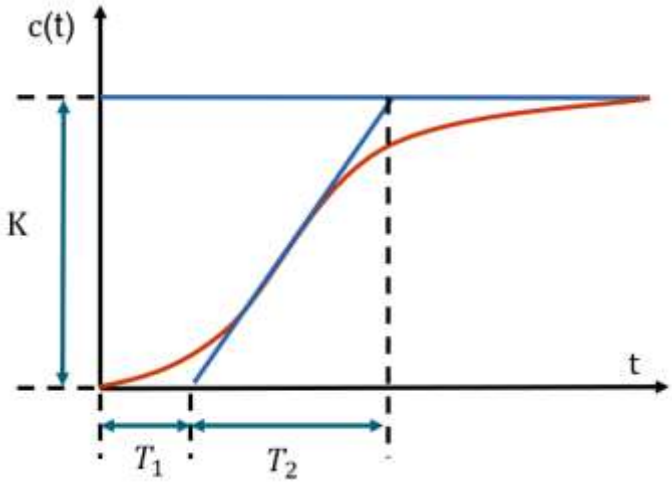
Hàm truyền thu được cung cấp cơ sở toán học để mô phỏng và thiết kế bộ điều khiển cho xe tự hành. Sự khác biệt nhỏ giữa hai động cơ được xem xét trong quá trình thiết kế để đảm bảo hiệu suất điều khiển đồng bộ giữa hai bánh xe.

4.9.2 Tính chọn bộ điều khiển cho động cơ

Thiết kế bộ điều khiển PID dựa vào đáp ứng quá độ của hệ hở: Phương pháp Zeigler-Nichols 1



Ở đây ta tiến hành tác động tín hiệu hàm bước ngõ vào và quan sát tín hiệu đầu ra.



Hình 4.6: Phương pháp kẻ tiếp tuyến

Dựa vào phương pháp kẻ tiếp tuyến để xác định mô hình xấp xỉ FOPDT (First Order Plus Dead Time) từ phản ứng bước của hệ thống.

Trong đó:

- Đường cong màu đỏ là phản ứng của hệ thống khi đưa vào tín hiệu bước
- K : độ lớn tín hiệu đặt (hàm bước)
- T_1 : Thời gian từ khi kích thích đầu vào đến khi tiếp tuyến cắt trục thời gian – gọi là thời gian trễ.
- T_2 : Thời gian từ lúc kích thích đến khi tiếp tuyến cắt mức giá trị cuối (K)

Bộ điều khiển PID có dạng:

$$G_c(s) = Kp\left(1 + \frac{1}{T_I s} + T_D s\right) \tag{4.6}$$

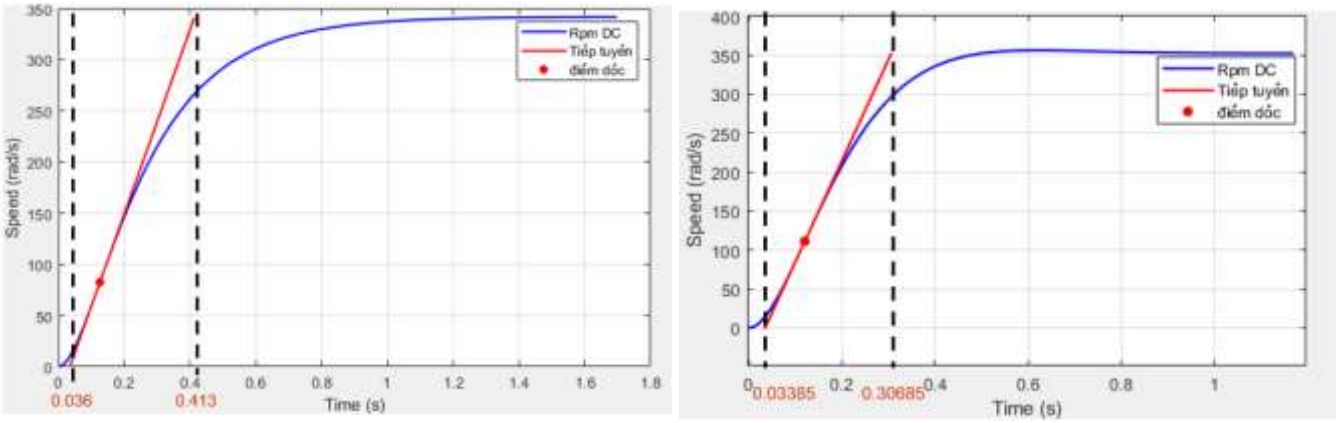
Xác định các thông số của bộ điều khiển dựa vào bảng thông số của Zeigler – Nichol

Bảng 4.8: Lựa chọn thông số cho bộ điều khiển

Controller	K_P	T_I	T_D
P	$T_2/(T_1K)$	∞	0
PI	$0.9T_2/(T_1K)$	$T_1/0.3$	0
PID	$1.2T_2/(T_1K)$	$2T_1$	$0.5T_1$

Dựa vào phương pháp Zeigler-Nichols 1 đã trình bày ở trên, tiến hành tính chọn bộ điều khiển PI cho động cơ:0

Sử dụng matlab viết chương trình xác định đường tiếp tuyến cho mỗi hàm truyền của động cơ, từ đó có cơ sở để xác định các thông số điều khiển hệ thống.



Hình 4.7: Xác định mô hình xấp xỉ FOPDT dựa vào phương pháp kẻ tiếp tuyến cho động cơ 1 và 2

❖ Động cơ 1

Sau khi thực hiện xác định mô hình xấp xỉ xong, ta xác định được các tham số chính:

$T_1 = 0.036(s)$

$T_2 = 0.413 - 0.036 = 0.377(s)$

$K = 341.8$

Dựa vào bảng thông số của Zeigler – Nichol, ta chọn bộ điều khiển PI cho động cơ 1:

$$K_p = 0.9\frac{T_2}{T_1 \times K} = 0.9\frac{0.377}{0.036 \times 341.8} = 0.02757$$

$$T_I = \frac{T_1}{0.3} = \frac{0.036}{0.3} = 0.12$$

$$K_I = \frac{K_p}{T_I} = \frac{0.02757}{0.12} = 0.22975$$

❖ Động cơ 2

Tương tự như ở động cơ 1, sau khi thực hiện xác định mô hình xấp xỉ xong, ta xác định được các tham số chính:

$T_1 = 0.03385(s)$

$T_2 = 0.30685 - 0.03385 = 0.273(s)$

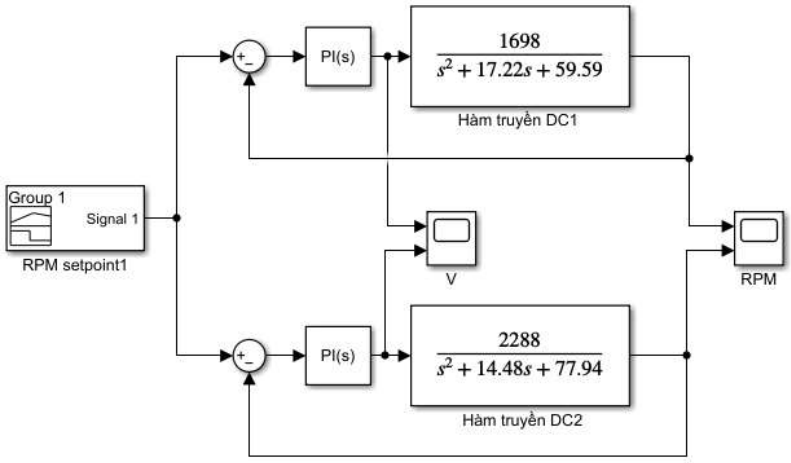
$K = 352.3$

Dựa vào bảng thông số của Zeigler – Nichol, ta chọn bộ điều khiển PI cho động cơ 2:

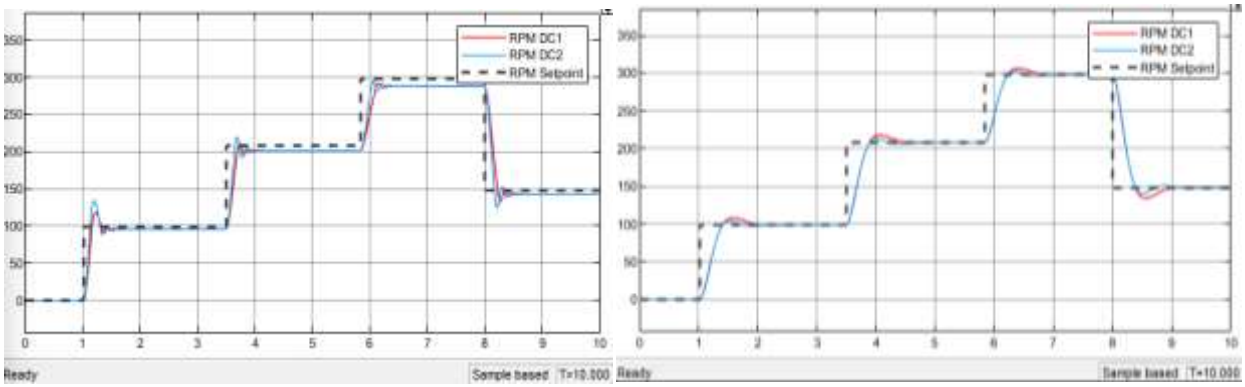
$$K_p = 0.9\frac{T_2}{T_1 \times K} = 0.9\frac{0.273}{0.03385 \times 352.3} = 0.0206$$

$$T_I = \frac{T_1}{0.3} = \frac{0.03385}{0.3} = 0.1128$$

$$K_I = \frac{K_p}{T_I} = \frac{0.0206}{0.1128} = 0.182575$$



Hình 4.8: Mô hình điều khiển tốc độ động cơ qua bđk PI



Hình 4.9: Đáp ứng của động cơ trước và sau khi có bộ điều khiển PI

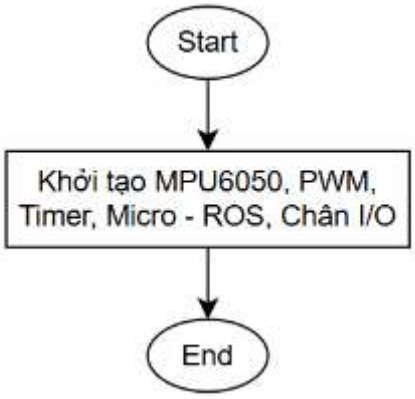
Ở chế độ chưa có bộ điều khiển (hệ hở), động cơ được tác động bằng tín hiệu bước. Quan sát phản ứng cho thấy hệ thống có dao động, đáp ứng chậm, và sai lệch tĩnh không bằng 0. Điều này chứng tỏ hệ thống chưa đáp ứng tốt yêu cầu điều khiển.

Sau khi tính toán được các thông số của bộ điều khiển PI theo phương pháp Ziegler-Nichols và đưa vào hệ thống, tiến hành mô phỏng lại. Kết quả cho thấy:

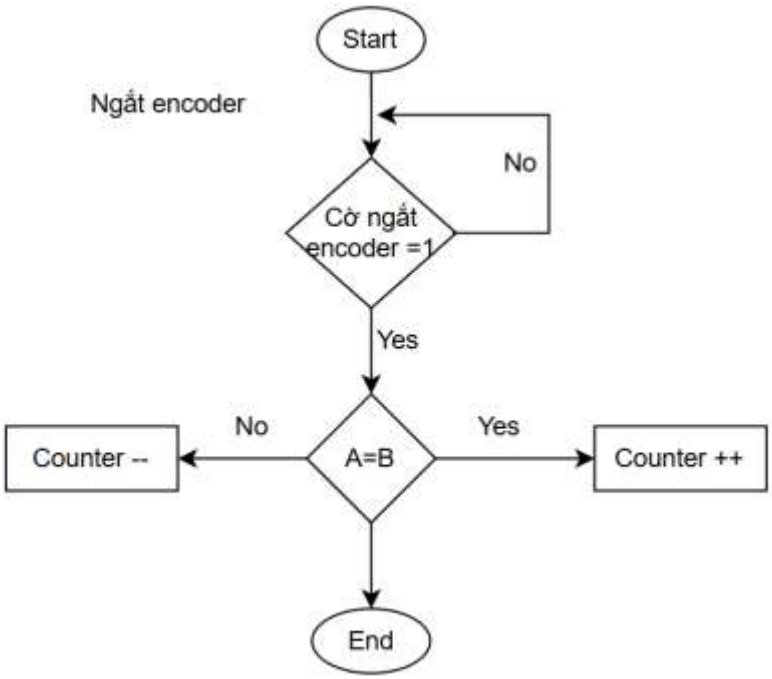
- yepQuá trình quá độ ít dao động
- Sai lệch tĩnh gần như bằng 0
- Hệ thống ổn định và đáp ứng tốt hơn khi chưa có bộ điều khiển

4.10 Cấu hình điều khiển trên esp32 và gửi dữ liệu phản hồi về ros2

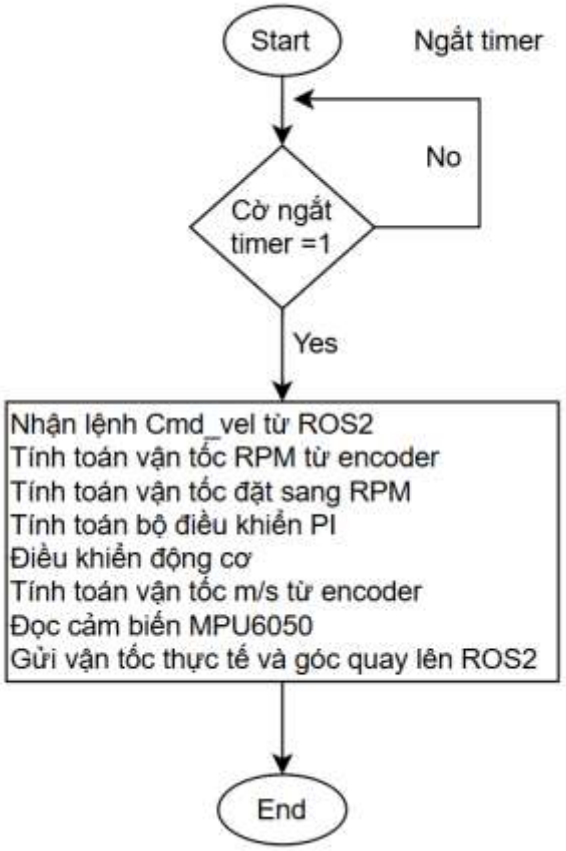
4.10.1 Lưu đồ thuật toán điều khiển và xử lý dữ liệu trên ESP32



Hình 4.10: Lưu đồ thuật toán chương trình chính



Hình 4.11: Lưu đồ thuật toán chương trình ngắt đọc encoder



Hình 4.12: Lưu đồ thuật toán chương trình ngắt timer xử lí tín hiệu

Quá trình xử lý tín hiệu:

- Nhận lệnh cmd_vel từ ROS2:

Esp32 sẽ nhận lệnh điều khiển vận tốc tuyến tính và góc (linear + angular) từ hệ thống ROS2 gửi xuống.

- Tính vận tốc RPM từ encoder:

Đọc encoder gắn trên động cơ để tính vận tốc thực tế hiện tại theo đơn vị RPM (vòng/phút).

- Tính vận tốc đặt (setpoint) sang RPM:

Chuyển đổi từ cmd_vel (m/s) sang đơn vị RPM, phù hợp để so sánh và điều khiển.

- Tính toán bộ điều khiển PI:

Sử dụng bộ điều khiển PI (Proportional–Integral) để tính toán sai số giữa vận tốc đặt và vận tốc thực tế, từ đó đưa ra giá trị điều khiển PWM cho động cơ.

- Điều khiển động cơ:

Gửi tín hiệu PWM đến driver động cơ dựa trên kết quả từ bộ điều khiển PI.

- Tính vận tốc thực tế (m/s):

Sau khi điều khiển, tính lại vận tốc thực tế theo m/s để gửi lên ROS2.

- Đọc cảm biến MPU6050:

Đọc góc quay (tốc độ góc quanh trục Z – omega_z) từ IMU để cung cấp thông tin định hướng.

- Gửi vận tốc và góc quay lên ROS2:

Gửi các dữ liệu thực tế vận tốc và góc quay lên ROS2 với topic /*wheel_speed.*

4.11 Cấu hình Node DC_Control xử lý publish dữ liệu odom

ESP32 có tài nguyên tính toán hạn chế (RAM và CPU thấp), nên để giảm tải xử lý, việc tính toán dữ liệu *odometry* sẽ được chuyển sang thực hiện trên Raspberry Pi 4.

ESP32 sẽ chỉ đảm nhiệm vai trò đo lường tốc độ bánh xe và truyền dữ liệu lên ROS 2 thông qua micro-ROS. Cụ thể, ESP32 sẽ publish dữ liệu lên topic */wheel_speed* dưới dạng một thông điệp *geometry_msgs/msg/Vector3*, trong đó gồm:

x : vận tốc bánh trái

y : vận tốc bánh phải

z : vận tốc góc của robot

Trên Raspberry Pi 4, một node có tên *dc_control* sẽ đảm nhận việc:

- Nhận dữ liệu từ topic */wheel_speed*
- Tính toán thông tin *odometry* theo công thức sau:

$$\begin{aligned} x &+= v.\Delta t.\cos(\theta) \\ y &+= v.\Delta t.\sin(\theta) \\ \theta &+= \omega.\Delta t \end{aligned} \tag{4.7}$$

Trong đó: $v = \frac{v_{left} + v_{right}}{2}$ là vận tốc tuyến tính của robot

ω: là vận tốc góc

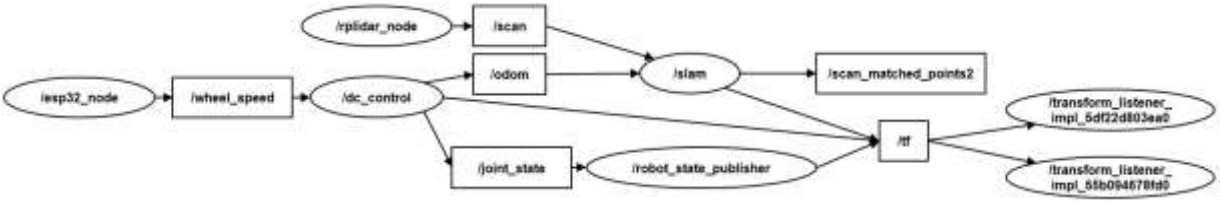
Δt : là khoảng thời gian giữa 2 lần cập nhật

θ : là góc quay của robot

Sau khi tính toán xong, node sẽ publish thông tin *odometry* lên topic /*odom* để các node khác như TF, localization, hoặc SLAM sử dụng.

4.12 Hoàn thiện kết nối luồng dữ liệu hệ thống

Sau khi kết nối hoàn thiện hệ thống, tiến hành chạy kiểm tra luồng dữ liệu:



Hình 4.13: Sơ đồ luồng dữ liệu giữa các node trong hệ thống ROS

Sơ đồ luồng dữ liệu được thể hiện trong Hình 4.13, mô tả mối liên hệ giữa các thành phần trong hệ thống. Tín hiệu tốc độ từ động cơ được thu thập bởi vi điều khiển ESP32 và gửi về node /esp32_node. Dữ liệu này được phát trên topic /wheel_speed, sau đó được node /dc_control xử lý để điều khiển động cơ, đồng thời tính toán vị trí tương đối của robot dựa trên phương pháp odometry. Node này xuất dữ liệu ra các topic /odom, /joint_state và /tf, tương ứng với thông tin vị trí, trạng thái khớp và các biến đổi tọa độ.

Song song với quá trình thu thập tốc độ, hệ thống cũng nhận dữ liệu từ cảm biến LiDAR thông qua node /rplidar_node, với dữ liệu quét môi trường được phát trên topic /scan. Cả hai dòng dữ liệu từ /scan và /odom được chuyển đến node /slam, nơi thực hiện thuật toán SLAM để xác định vị trí robot đồng thời với việc xây dựng bản đồ môi trường xung quanh. Kết quả SLAM bao gồm các điểm quét đã được hiệu chỉnh được gửi qua topic /scan_matched_points2, trong khi thông tin định vị được phát đến topic /tf nhằm cung cấp các biến đổi tọa độ phục vụ cho các thành phần khác trong hệ thống.

Để hiển thị mô hình robot và phục vụ các thuật toán điều hướng, node /robot_state_publisher sử dụng thông tin từ /joint_state để tính toán các biến đổi hình học dựa trên mô hình URDF của robot và phát tiếp lên topic /tf. Các node listener như /transform_listener_impl_* sẽ nhận các thông tin từ /tf để dựng lại vị trí và hướng của

robot trong không gian, từ đó phục vụ cho việc hiển thị, giám sát hoặc điều khiển robot. Tất cả các bước trên tạo thành một chu trình khép kín, đảm bảo robot có thể định vị chính xác, xây dựng bản đồ, và phản ứng kịp thời với môi trường xung quanh.

CHƯƠNG 5. KIỂM NGHIỆM VÀ ĐÁNH GIÁ KẾT QUẢ

5.1 Mô phỏng

Sau khi hoàn tất quá trình cấu hình hệ thống và lập trình các thuật toán trình bày trong Chương 4, nhóm tiến hành bước mô phỏng thuật toán SLAM nhằm kiểm chứng khả năng định vị và xây dựng bản đồ môi trường của hệ thống.

Việc mô phỏng được thực hiện trên nền tảng robot ảo TurtleBot3 Burger, sử dụng môi trường giả lập Gazebo.

Mục tiêu của quá trình mô phỏng là:

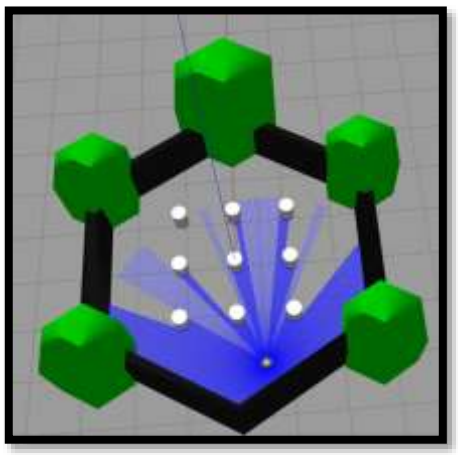
- Đánh giá hiệu quả hoạt động của thuật toán SLAM đã phát triển
- Kiểm tra độ chính xác định vị
- Kiểm chứng khả năng xây dựng bản đồ môi trường
- Phát hiện và điều chỉnh các vấn đề tiềm ẩn trước khi triển khai thực tế

5.1.1 Lập bản đồ

5.1.1.1 Khởi tạo môi trường mô phỏng Gazebo

Trước tiên, cần thiết lập môi trường giả lập Gazebo với mô hình robot TurtleBot3 trong bản đồ có sẵn (turtlebot3_world)

```
export TURTLEBOT3_MODEL=burger
ROS2 launch turtlebot3_gazebo turtlebot3_world.launch.py
```



Hình 5.1: Môi trường mô phỏng với robot sẵn sàng hoạt động.

5.1.1.2 Khởi chạy thuật toán SLAM

Khởi chạy thuật toán slam đã được xây dựng trước đó:

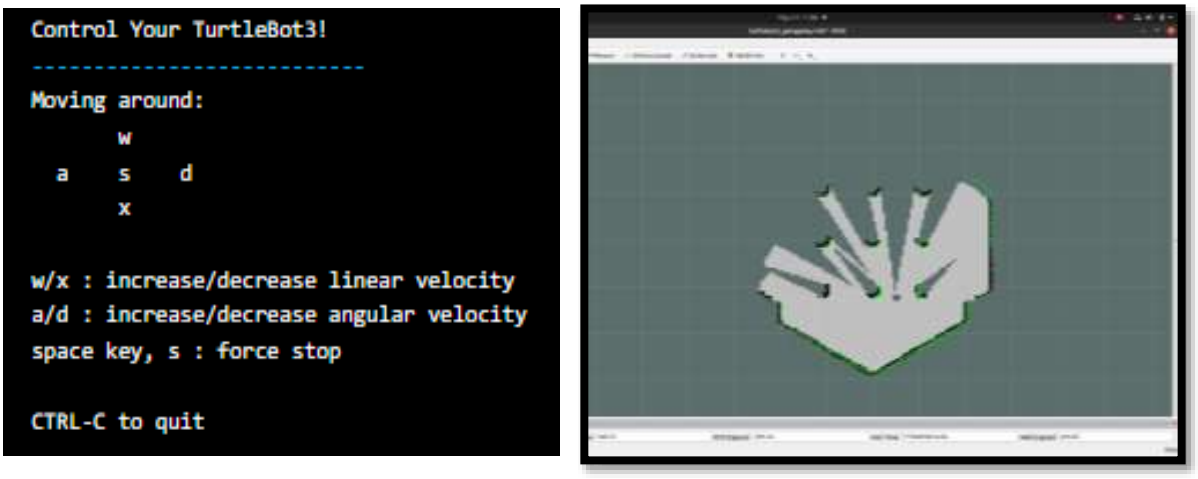
```
export TURTLEBOT3_MODEL=burger
ROS2 launch turtlebot3_slam slam.launch.py use_sim_time:=True
```

Câu lệnh trên sẽ khởi chạy node slam_gmapping, cho phép robot vừa di chuyển vừa xây dựng bản đồ môi trường xung quanh.

5.1.1.3 Di chuyển robot để thu thập dữ liệu bản đồ

Quá trình điều khiển robot được thực hiện thông qua giao diện điều khiển bàn phím giúp đảm bảo robot quét được nhiều khu vực khác nhau nhằm hoàn thiện bản đồ:

```
export TURTLEBOT3_MODEL=burger
ROS2 run turtlebot3_teleop teleop_keyboard
```



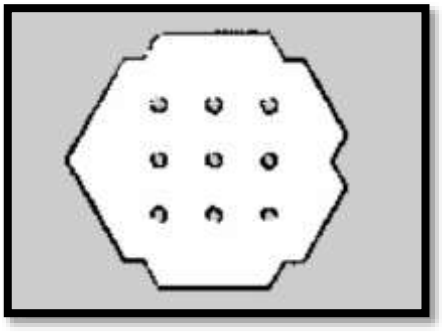
Hình 5.2: điều khiển robot di chuyển để quét map

5.1.1.4 Lưu bản đồ sau khi quét

Sau khi điều khiển robot quét xung quanh, bản đồ thu được thể hiện đầy đủ các cạnh tường, vật cản và hình dạng không gian. Điều này cho thấy thuật toán slam_gmapping xử lý dữ liệu scan một cách ổn định.

Khi bản đồ đã được xây dựng hoàn chỉnh, thực hiện lưu lại bản đồ này bằng công cụ map_server:

```
ROS2 run nav2_map_server map_saver_cli -f ~/map
```



Hình 5.3: Bản đồ mô phỏng được quét hoàn chỉnh

5.1.2 Định vị

Bản đồ thu được từ giai đoạn SLAM đóng vai trò làm cơ sở dữ liệu nền cho bước định vị tiếp theo. Tại đây, robot sử dụng thuật toán định vị để ước lượng vị trí hiện tại của mình trên bản đồ đã xây dựng, thông qua việc so sánh dữ liệu cảm biến thời gian thực với dữ liệu bản đồ.

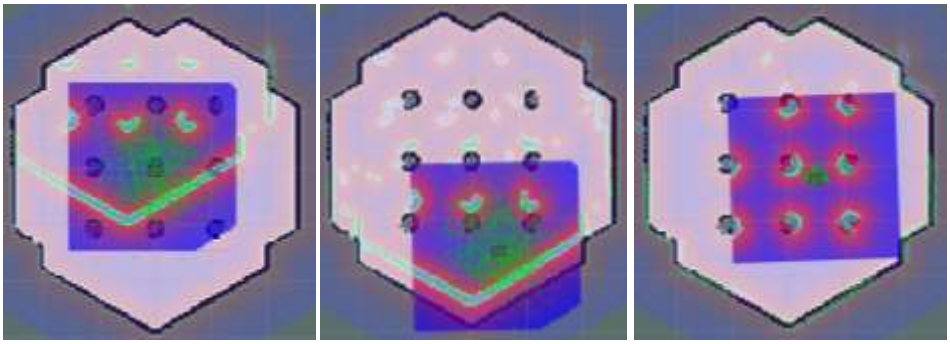
5.1.2.1 Khởi chạy thuật toán định vị

```
    export TURTLEBOT3_MODEL=burger
    ROS2 launch turtlebot3_navigation2 navigation2.launch.py
map:=$HOME/map.yaml
```

5.1.2.2 Quá trình robot định vị vị trí

Khi cả môi trường và thuật toán định vị đã sẵn sàng như hình 4.1a, bước kế tiếp là ước lượng vị trí ban đầu của robot trong bản đồ. Việc này được thực hiện trong công cụ trực quan hóa Rviz:

- Mở giao diện Rviz.
- Nhấn vào nút "2D Pose Estimate" trên thanh công cụ.
- Click chuột trái vào vị trí mong muốn trên bản đồ để xác định vị trí x, y.
- Kéo chuột để định hướng (yaw) cho robot.



(a). (b). (c).

Hình 5.4: Quá trình định vị theo hình a, b, c

Khi vừa ước lượng xong vị trí bán đầu như hình 4.1b ta có thể thấy được vị trí tương đối của robot trên bản đồ. Lúc này thuật toán slam bắt đầu hoạt động thể hiện bằng các vector dự đoán vị trí và hướng của robot trong môi trường. Để tăng độ chính xác trong việc định vị vị trí của robot, ta cần cho robot di chuyển. Sau khi di chuyển thuật toán slam đã tính toán và dự báo chính xác vị trí của robot như hình 4.1c, thể hiện bằng việc đường quét lidar đã trùng khớp với các cạnh của bản đồ và các vector dự đoán vị trí của robot đã hôi tụ lại với nhau.

5.1.3 Đánh giá kết quả mô phỏng

Bản đồ môi trường mô phỏng được xây dựng đầy đủ và rõ ràng, các đường biên, vật cản và không gian trống được thể hiện chính xác trong RViz.

Thuật toán SLAM hoạt động tốt, đảm bảo việc cập nhật liên tục bản đồ trong khi robot đang di chuyển, đồng thời giữ được định vị chính xác trong không gian mô phỏng.

Khả năng điều hướng chính xác**:** Robot có thể di chuyển từ vị trí xuất phát đến vị trí đích với quỹ đạo tối ưu, tránh vật cản và dừng đúng tại điểm mong muốn với sai số rất nhỏ.

5.2 Thực Nghiệm

Sau khi hoàn thành các bước thiết kế phần cứng, xây dựng phần mềm và tích hợp hệ thống, mô hình xe tự hành đã sẵn sàng cho việc kiểm thử trong điều kiện thực tế.

Kết quả thực nghiệm sẽ được trình bày chi tiết trong các mục tiếp theo, bao gồm hình ảnh minh họa quá trình robot hoạt động và bản đồ môi trường thu được từ phần mềm RViz.

5.2.1 Lập bản đồ

Để minh họa trực quan cho quá trình lập bản đồ, nhóm thực hiện đã ghi lại video toàn bộ quá trình xe tự hành thực hiện quét môi trường và xây dựng bản đồ trong thời gian thực. Dưới đây là một số hình ảnh được trích xuất từ video nhằm thể hiện từng giai đoạn của quá trình.

Đánh giá độ tin cậy của bản đồ được tạo ra bằng thuật toán SLAM, nhóm thực hiện đối chiếu hình ảnh thực tế từ camera (quay quá trình robot di chuyển) với bản đồ được hiển thị song song trên RViz. Việc so sánh này giúp xác nhận rằng các đặc điểm vật lý của môi trường thực tế đã được phản ánh chính xác trên bản đồ số.

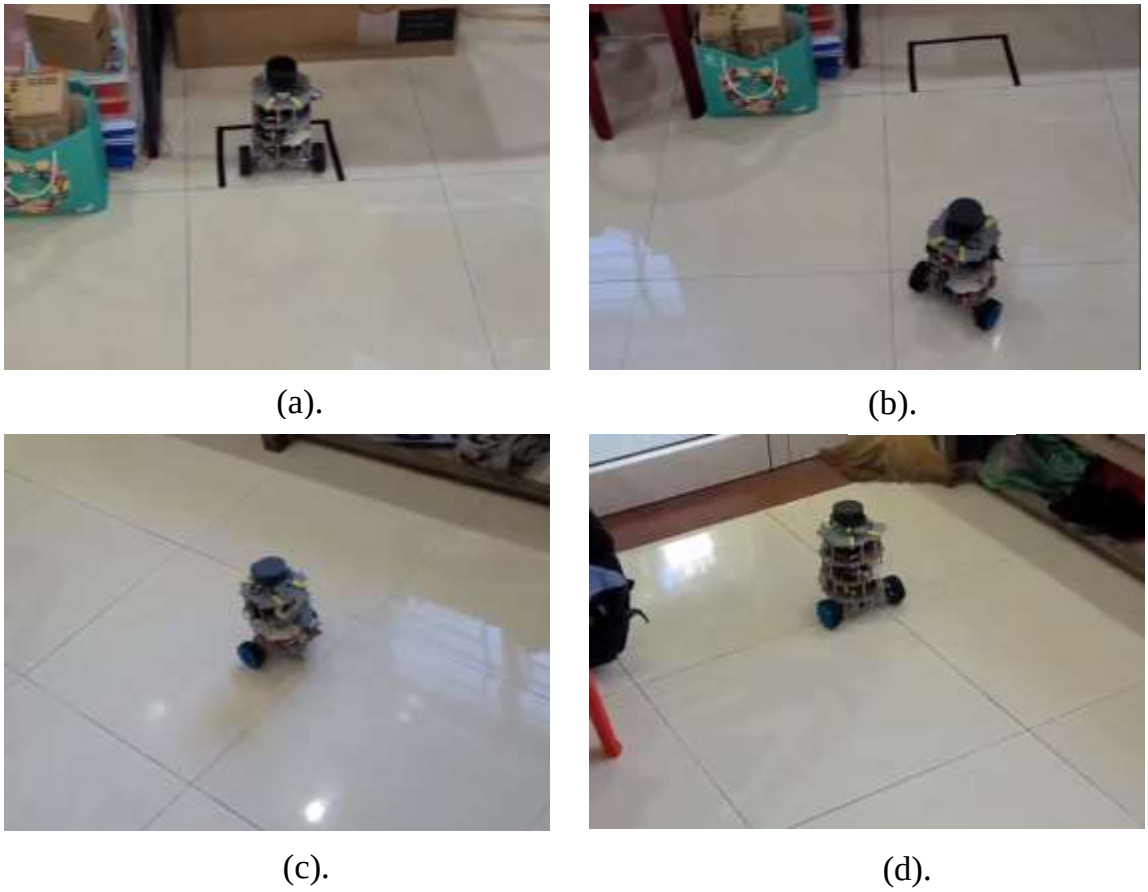
❖ Quá trình thực hiện:

Thiết lập môi trường và khởi chạy thuật toán slam.

```
export TURTLEBOT3_MODEL=burger
ros2 launch turtlebot3_slam slam.launch.py
```

Khởi chạy node dùng bàn phím điều khiển robot. Tại đây tiến hành điều khiển robot chạy để quét bản đồ.

```
export TURTLEBOT3_MODEL=burger
ros2 run turtlebot3_teleop teleop_keyboard
```



(a). (b).

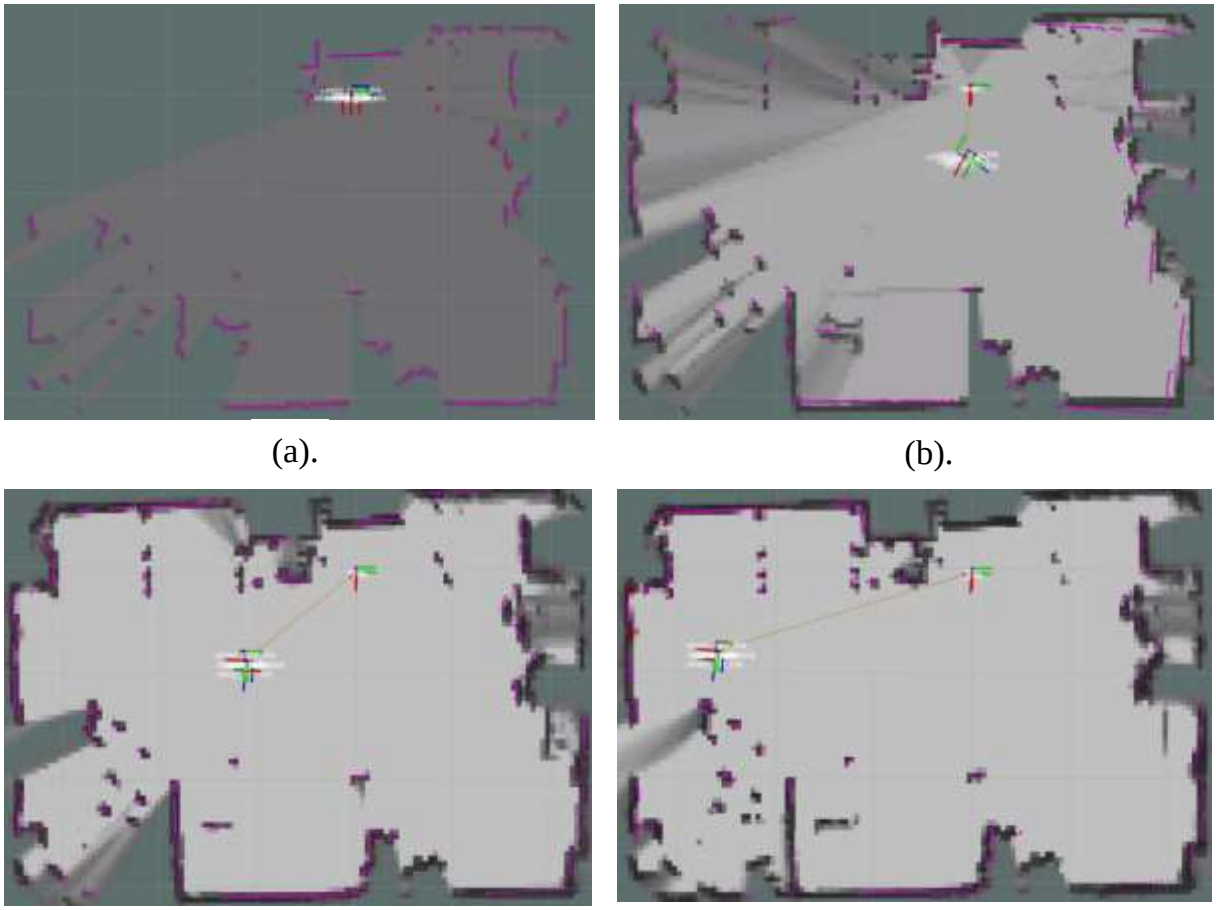
(c). (d).

Hình 5.5: Thực nghiệm lập bản đồ của robot theo trình tự các hình a, b, c, d

Hình 5.5 ghi lại quá trình robot di chuyển trong môi trường thực tế dưới sự điều khiển từ xa thông qua các lệnh điều hướng được gửi từ máy tính điều khiển (remote PC). Robot nhận lệnh di chuyển tiến, lùi, rẽ trái, rẽ phải thông qua ROS2 và thực hiện các chuyển động tương ứng trong không gian thực.

Song song với đó, Hình 5.6 thể hiện quá trình quét bản đồ trong thời gian thực trên giao diện RViz, tương ứng với từng tư thế và vị trí robot trong Hình 5.5.

Khi vừa khởi chạy thuật toán SLAM để bắt đầu quá trình xây dựng bản đồ, như thể hiện ở Hình 5.6.a, bản đồ còn ở trạng thái sơ khai với rất ít thông tin về môi trường xung quanh; các đặc trưng (landmark) chưa được xác định rõ ràng. Khi robot bắt đầu di chuyển, các cảm biến như LiDAR, encoder và IMU liên tục thu thập dữ liệu, và gửi về thuật toán SLAM xử lý và cập nhật dần thông tin môi trường. Bản đồ dần hiện rõ khi robot quét qua các khu vực mới, như Hình 5.6.b và 5.6.c. Cuối cùng, khi robot đã đi qua hầu hết không gian cần lập bản đồ, bản đồ được hoàn thiện một cách đầy đủ, thể hiện rõ các tường, vật cản và không gian trống, như thể hiện trong Hình 5.6.d.



(a). (b).

(c). (d).

Hình 5.6: Bản đồ được tạo theo trình tự các hình a, b, c, d

Đảm bảo rằng quá trình quét map đã chạy xong, bản đồ hiển thị trên RViz đã đầy đủ và chính xác. Sau đó, mở một terminal mới và chạy lệnh sau để lưu bản đồ vừa tạo:

```
ros2 run nav2_map_server map_saver_cli -f ~/map
```

5.2.2 Định vị và điều hướng

Sau khi hoàn tất quá trình xây dựng bản đồ môi trường bằng thuật toán SLAM, bước tiếp theo trong quá trình là triển khai chức năng định vị và điều hướng tự động dựa trên bản đồ đã lưu. Quá trình này cho phép robot xác định vị trí hiện tại của mình trong không gian và tự động di chuyển đến các vị trí đích được chỉ định, đồng thời đảm bảo an toàn tránh va chạm với các chướng ngại vật.

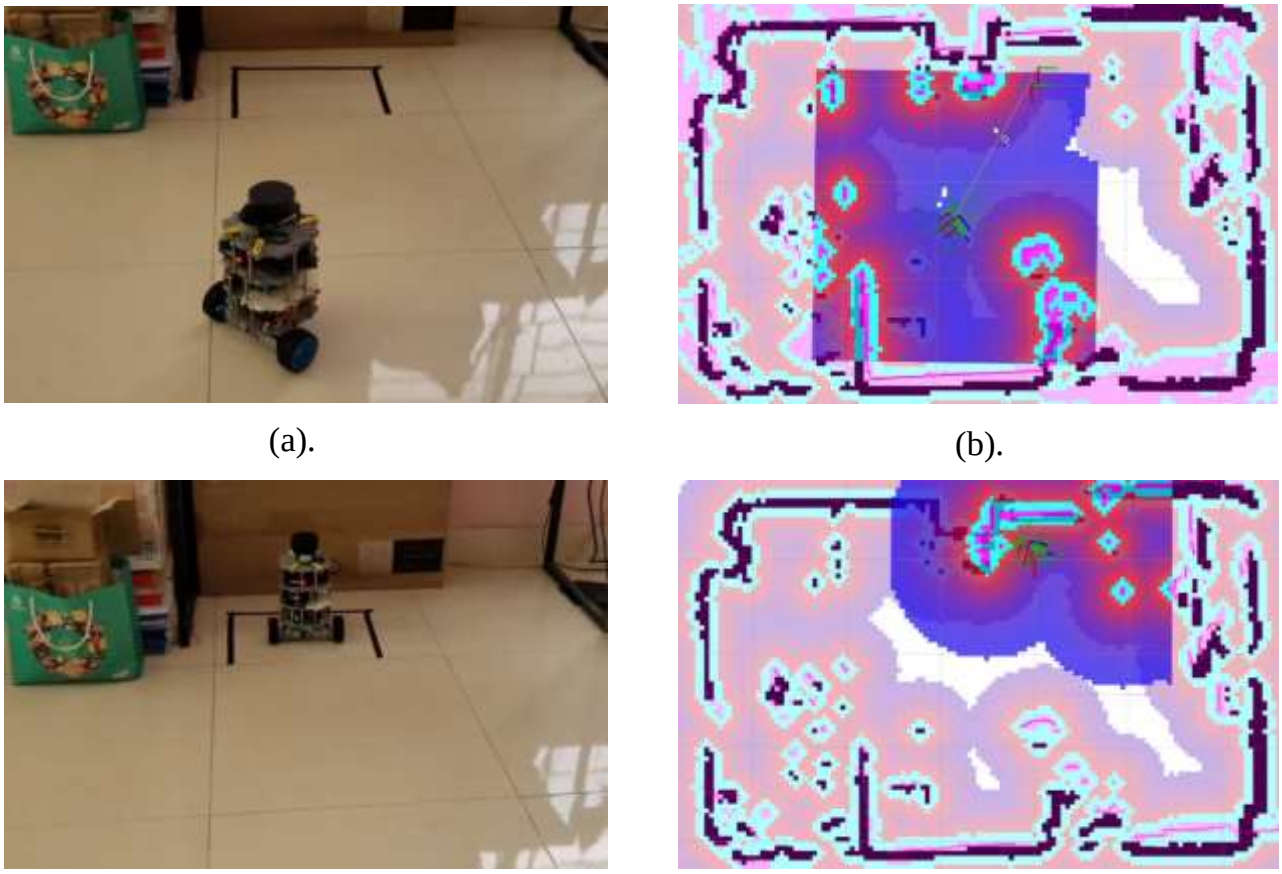
Hệ thống định vị và điều hướng được khởi động bằng launch file navigation2. launch.py nằm trong gói turtlebot3_navigation2. Launch file này tích hợp đầy đủ các

thành phần cần thiết cho quá trình tự hành. Để Launch file hoạt động tiến hành truyền đường dẫn tới file bản đồ .yaml đã lưu trước đó bằng lệnh:

```
export TURTLEBOT3_MODEL=burger
ros2 launch turtlebot3_navigation2 navigation2.launch.py
map:=$HOME/map.yaml
```

Song song với đó, khởi chạy giao diện trực quan RViz bằng lệnh:

```
rviz2 -d $(ros2 pkg prefix
nav2_bringup)/share/nav2_bringup/rviz/nav2_default_view.rviz
```



(a). (b).

(c). (d).

Hình 5.7 Quá trình định vị và điều hướng của robot

Trong quá trình kiểm thử chức năng định vị và điều hướng, robot được đặt tại một vị trí bất kỳ trong môi trường, trong khi vị trí đích được xác định là một khu vực có đánh dấu viền đen trên mặt sàn. Hình 5.7.a là vị trí khởi đầu thực tế của robot trong không gian thử nghiệm, trong khi Hình 5.7.b thể hiện vị trí tương ứng trên bản đồ ở máy tính giám sát thông qua phần mềm RViz. Sau khi gửi lệnh để nav2 điều hướng tự động đến

vị trí đích là khu vực được đánh dấu viền đen, hệ thống đã xử lý dữ liệu bản đồ và tính toán lộ trình di chuyển tối ưu, cho phép robot tự hành tiến đến khu vực mục tiêu một cách chính xác. Kết quả đạt được cho thấy robot đã di chuyển thành công đến vị trí được đánh dấu viền đen như minh họa ở Hình 5.7.c trong môi trường thực tế, đồng thời vị trí đích cũng được thể hiện rõ ràng trên bản đồ trong Hình 5.7.d. Điều này khẳng định độ chính xác và hiệu quả cao của hệ thống định vị và điều hướng dựa trên bản đồ SLAM.

5.3 Đánh giá kết quả thực nghiệm

❖ Về quá trình lập bản đồ (Mapping)

Tính chính xác cao: Bản đồ được tạo từ dữ liệu cảm biến LiDAR thông qua thuật toán SLAM phản ánh rõ ràng cấu trúc không gian thử nghiệm, bao gồm các bức tường, góc phòng và vật cản như bàn, ghế, tủ...

Tính liên tục: Trong quá trình robot di chuyển, bản đồ được cập nhật liên tục theo thời gian thực trên RViz mà không xảy ra hiện tượng mất dữ liệu hay nhảy vị trí.

Độ trễ thấp: Việc xử lý dữ liệu từ cảm biến và hiển thị trên bản đồ diễn ra gần như đồng thời, đáp ứng yêu cầu thời gian thực cho các ứng dụng điều hướng tự động.

Sai số nhỏ: Sai số định vị trung bình của robot trong quá trình lập bản đồ vào khoảng 5–10 cm, hoàn toàn chấp nhận được đối với môi trường trong nhà có không gian hạn chế.

❖ Về quá trình điều hướng (Navigation)

Khả năng di chuyển chính xác: Robot đã định hướng và di chuyển thành công từ vị trí bất kỳ đến vị trí đích được đánh dấu, với quỹ đạo hợp lý và tránh được các vật cản trên đường đi với độ chính xác khá cao.

Phản hồi tốt với bản đồ: Hệ thống sử dụng bản đồ đã xây dựng để lên kế hoạch đường đi, tính toán các bước rẽ, và điều chỉnh hướng di chuyển một cách linh hoạt.

Thành công trong môi trường thực tế: Thử nghiệm điều hướng cho thấy hệ thống có thể áp dụng trong các môi trường thực tế tương tự với điều kiện ánh sáng tự nhiên, nền bằng phẳng và có nhiều vật cản.

Độ ổn định tương đối cao: Robot ít bị mất phương hướng hoặc lệch khỏi quỹ đạo dự kiến trong suốt quá trình điều hướng.

KẾT LUẬN CHUNG

Kết luận

Đề tài "Ứng dụng thuật toán SLAM và xử lý điện toán biên cho định vị, lập bản đồ robot tự hành trong nhà kho" đã được thực hiện nhằm xây dựng một hệ thống robot có khả năng tự động nhận thức môi trường, tự định vị và lập bản đồ mà không cần dựa vào hạ tầng định vị sẵn có. Bằng việc tích hợp các thành phần phần cứng như cảm biến LiDAR, encoder và máy tính biên (Raspberry Pi 4), cùng với phần mềm ROS2 và thuật toán SLAM, nhóm thực hiện đã phát triển thành công một mô hình robot tự hành hoạt động ổn định trong môi trường.

Các nội dung chính đạt được trong đề tài gồm:

- Nghiên cứu và áp dụng thành công thuật toán SLAM (Simultaneous Localization and Mapping) để giúp robot đồng thời xác định vị trí của chính nó và xây dựng bản đồ môi trường xung quanh.
- Triển khai mô hình xử lý điện toán biên (Edge Computing) sử dụng vi máy tính nhúng nhằm giảm tải xử lý dữ liệu và đảm bảo hoạt động thời gian thực mà không phụ thuộc vào máy chủ trung tâm.
- Hoàn thiện mô hình mô phỏng và thực nghiệm, trong đó robot đã thực hiện tốt các chức năng: thu thập dữ liệu môi trường, lập bản đồ thời gian thực, tự động điều hướng và tránh vật cản.
- Kết quả thử nghiệm thực tế trong môi trường mô phỏng nhà kho (phòng kín) cho thấy robot có thể xây dựng bản đồ với độ chính xác cao (sai số định vị dưới 10 cm), đồng thời điều hướng thành công đến vị trí đích.

Hạn chế

Mặc dù đạt được nhiều kết quả tích cực, đề tài vẫn còn một số hạn chế như:

- Phạm vi thử nghiệm còn giới hạn trong môi trường nhỏ và tương đối ít vật cản.
- Chưa tích hợp thuật toán tối ưu đường đi nâng cao và điều hướng động trong môi trường thay đổi liên tục.

Ngoài ra còn một số hạn chế về mặt phần cứng:

- Cảm biến LiDAR A1M8: Đây là loại LiDAR giá rẻ, tầm quét chỉ khoảng 8–12 mét và góc quét 360° nhưng độ phân giải và tốc độ quét còn hạn chế. Trong các môi trường có ánh sáng phản xạ mạnh, nhiều vật thể nhỏ hoặc vật cản mỏng, A1M8 dễ bị nhiễu, ảnh hưởng đến độ chính xác của bản đồ.

- IMU tích hợp : IMU sử dụng trong đề tài không phải loại chất lượng cao nên tín hiệu dễ bị nhiễu, trôi dần theo thời gian (drift), làm giảm độ chính xác của định vị khi robot quay góc hoặc bị rung lắc.
- ESP32: Mặc dù có khả năng điều khiển động cơ và truyền dữ liệu, ESP32 có giới hạn về tốc độ xử lý và bộ nhớ. Việc truyền dữ liệu encoder hoặc cảm biến về Raspberry Pi có thể gặp độ trễ, ảnh hưởng đến phản hồi thời gian thực.
- Raspberry Pi 4: Đây là máy tính nhúng có hiệu năng khá tốt, nhưng vẫn chưa đủ mạnh để xử lý tối ưu thuật toán SLAM nếu môi trường phức tạp hoặc bản đồ lớn. Khi xử lý đồng thời nhiều luồng dữ liệu (LiDAR, encoder, …), hệ thống có thể bị giật hoặc trễ hình.
- Hệ thống sử dụng động cơ thường thay vì servo nên độ chính xác điều hướng chưa cao, dễ lệch hướng khi quay hoặc di chuyển chậm.

Hướng phát triển cho đề tài

Nâng cấp phần cứng: Sử dụng LiDAR độ phân giải cao, IMU chất lượng tốt, và thay Raspberry Pi bằng máy tính nhúng mạnh hơn để cải thiện tốc độ xử lý và độ chính xác định vị.

Tích hợp camera: Áp dụng thuật toán Visual SLAM (vSLAM) với camera RGB hoặc camera độ sâu (Realsense D435) để hỗ trợ nhận dạng và lập bản đồ 3D trong môi trường phức tạp hơn.

Sử dụng động cơ servo: Thay thế động cơ thường bằng động cơ servo để tăng độ chính xác khi di chuyển và điều hướng, giảm sai số khi quay và rẽ.

Cải tiến thuật toán điều hướng: Ứng dụng các thuật toán lập kế hoạch đường đi nâng cao để kết hợp tránh vật cản động và tối ưu hóa tuyến đường trong thời gian thực.

Mở rộng môi trường thử nghiệm: Áp dụng hệ thống trong môi trường nhà kho thực tế với nhiều vật thể di động, kiểm tra tính ổn định và khả năng thích nghi của robot trong điều kiện hoạt động thực tế.

Xây dựng giao diện điều khiển và giám sát từ xa: Phát triển dashboard trên web hoặc thiết bị di động để theo dõi trạng thái robot, bản đồ, và điều khiển từ xa qua mạng.

DANH MỤC THAM KHẢO

[1] Zheng Zhang, Juan Chen* and Qing Guo, Application of Automated Guided Vehicles in Smart Automated Warehouse Systems: A Survey, 2022

[2] Nazir Yusuf, Sabi'U Usman Suleiman, Review: Issues and Challenges of Simultaneous Localization and Mapping (SLAM) Technology in Autonomous Robot, 2023

[3] Saleh Abuali, Mickyas Tamiru Asfaw, Perla Sammour, Controlling a Mobile Robot Using ROS2 and Machine Learning, 2024

[4] Giorgio Grisetti, Rainer Kummerle, Cyrill Stachniss, Wolfram Burgard, A Tutorial on Graph-Based SLAM, 2014

[5] Cyrill Stachniss, EKF SLAM – Simultaneous-Localization and Mapping, 2023

[6] Krisztián Balázs Kis, János Csempesz, Balázs Csanád Csáji, A simultaneous localization and mapping algorithm for sensors with low sampling rate and its application to autonomous mobile robots, 2021

[7] Inam Ullah, Deepak Adhikari, Habib Khan, M. Shahid Anwar, Shabir Ahmad, Xiaoshan Bai, Mobile robot localization: Current challenges and future prospective, 2024

[8] Cesar Cadena, Past, Present, and Future of Simultaneous Localization and Mapping, 2016

[9] Xiangdi Yue, Yihuan Zhang, Jiawei Chen, Junxin Chen, Xuanyi Zhou, Miaolei He, LiDAR-based SLAM for robotic mapping: state of the art and new frontiers, 2023

[10] micro-ROS puts ROS 2 onto microcontrollers, micro-ROS | ROS 2 for microcontrollers [Ngày truy cập: 15/4/2025]

[11] Turtlebot3 – Robotis e-Manual, TurtleBot3, [Ngày truy cập: 10/2/2025]

[12] Wheeled Mobile Robot KINEMATICS, Wheeled Mobile Robot Kinematics — ROS2_Control: Rolling May 2025 documentation, [Ngày truy cập: 10/2/2025]

[13] ROS 2 Documentation, ROS 2 Documentation — ROS 2 Documentation: Jazzy documentation [Ngày truy cập: 10/3/2025]

[14] eProsima Fast DDS, https://docs.ros.org/en/humble/Installation/RMW-Implementations/DDS-Implementations/Working-with-eProsima-Fast-DDS.html [Ngày truy cập: 10/4/2025]

PHỤ LỤC

1. Cài đặt Ubuntu 22.04

Hệ điều hành Ubuntu 22.04 LTS được sử dụng để đảm bảo tương thích với ROS2 Humble.

2. Cài đặt ROS2 trên PC

2.1. Thiết lập nguồn phần mềm

Quá trình cài đặt ROS2 Humble cần có các gói phần mềm chính thức của ROS2 để đảm bảo khả năng tải và cài đặt các gói phần mềm cần thiết.

- Kích hoạt kho lưu trữ Universe

Kho lưu trữ "Universe" được kích hoạt nhằm cung cấp các gói phụ thuộc cần thiết cho ROS2.

```
    sudo apt install software-properties-common
    sudo add-apt-repository universe
```

- Thêm khóa GPG của ROS2

Thêm vào hệ thống để xác thực các gói phần mềm từ nguồn ROS2.

```
    sudo apt update && sudo apt install curl -y
    sudo curl -sSL
https://raw.githubusercontent.com/ROS2/ROS2distro/master/ROS2.key -o
/usr/share/keyrings/ROS2-archive-keyring.gpg
```

- Cấu hình nguồn ROS2 trong danh sách apt

Để hệ thống nhận diện và tải các gói từ máy chủ chính thức.

```
    echo "deb [arch=$(dpkg --print-architecture) signed
by=/usr/share/keyrings/ROS2-archive-keyring.gpg]
http://packages.ROS2.org/ROS2/ubuntu $(. /etc/os-release && echo
$UBUNTU_CODENAME) main" | sudo tee /etc/apt/sources.list.d/ROS2.list >
/dev/null
```

2.2. Cập nhật và cài đặt ROS2

- Cập nhật danh sách gói phần mềm

```
    sudo apt update
    sudo apt upgrade
```

- Cài đặt phiên bản Desktop

Đủ các công cụ như RViz và thư viện điều hướng.

```
sudo apt install ROS2-humble-desktop
```

- Cài đặt các công cụ: Trình biên dịch và các công cụ khác để xây dựng các gói ROS2

```
sudo apt install ROS2-dev-tools
```

2.3. Thiết lập môi trường

Môi trường hệ thống cần được cấu hình để nhận diện ROS2 ngay khi mở terminal làm việc.

```
source /opt/ROS2/humble/setup.bash
```

3. Cài đặt các công cụng trong gói ROS2

- Gazebo

Trình mô phỏng robot 3D. Cho phép kiểm tra robot, cảm biến và môi trường mà không cần phần cứng thật. Dùng kết hợp với ROS2 để mô phỏng chuyển động, cảm biến (lidar, camera…), vật lý…

```
sudo apt install ROS2-humble-gazebo-*
```

- Navigation2

Hệ thống điều hướng cho robot di chuyển từ điểm A đến B một cách tự động. Gồm các thành phần như: Localization (định vị), Mapping (bản đồ), Path planning (tính toán đường đi), Obstacle avoidance (tránh vật cản).

```
sudo apt install ROS2-humble-navigation2
sudo apt install ROS2-humble-nav2-bringup
```

4. Cài đặt gói Turtlebot3

Các gói TurtleBot3 được cài đặt để hỗ trợ vận hành và mô phỏng robot.

- Tạo một không gian làm việc

```
source /opt/ROS2/humble/setup.bash
mkdir -p ~/turtlebot3_ws/src
cd ~/turtlebot3_ws/src
```

- Tải mã nguồn TurtleBot3

```
git clone -b humble-devel https://github.com/ROBOTIS-GIT/DynamixelSDK.git
git clone -b humble-devel https://github.com/ROBOTIS-GIT/turtlebot3_msgs.git
git clone -b humble-devel https://github.com/ROBOTIS-GIT/turtlebot3.git
git clone -b humble https://github.com/ROBOTIS-GIT/turtlebot3_simulations.git
```

- Cài đặt công cụ mở rộng cho colcon – trình build chính trong ROS2

```
sudo apt install python3-colcon-common-extensions
```

- Biên dịch toàn bộ workspace, thêm lệnh thiết lập môi trường workspace vào .bashrc để tự động kích hoạt ROS2 workspace mỗi lần mở Terminal và áp dụng ngay thay đổi môi trường mà không cần khởi động lại terminal.

```
cd ~/turtlebot3_ws
colcon build --symlink-install
echo 'source ~/turtlebot3_ws/install/setup.bash' >> ~/.bashrc
source ~/.bashrc
```

5. Trỏ tới môi trường ROS2

Môi trường ROS2 được cấu hình để nhận diện các gói TurtleBot3 ngay khi mở terminal làm việc mà không cần khai báo lại.

```
echo 'source /usr/share/gazebo/setup.sh' >> ~/.bashrc
echo 'source /opt/ROS2/humble/setup.bash' >> ~/.bashrc
source ~/.bashrc
```