

ĐẠI HỌC QUỐC GIA TP. HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN
KHOA KỸ THUẬT MÁY TÍNH

PHÙNG VĂN HẢO
PHAN HỮU ĐẠT

KHÓA LUẬN TỐT NGHIỆP
NGHIÊN CỨU, TÍCH HỢP ROS2 VÀ RTOS CHO
ROBOT TỰ HÀNH
RESEARCH ON INTEGRATING ROS2 AND RTOS INTO
AUTONOMOUS ROBOT

KỸ SƯ KỸ THUẬT MÁY TÍNH

TP. HỒ CHÍ MINH, 2022

ĐẠI HỌC QUỐC GIA TP. HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN
KHOA KỸ THUẬT MÁY TÍNH

PHÙNG VĂN HẢO – 17520451

PHAN HỮU ĐẠT – 17520339

KHÓA LUẬN TỐT NGHIỆP
NGHIÊN CỨU, TÍCH HỢP ROS2 VÀ RTOS CHO
ROBOT TỰ HÀNH
RESEARCH ON INTEGRATING ROS2 AND RTOS INTO
AUTONOMOUS ROBOT

KỸ SƯ KỸ THUẬT MÁY TÍNH

GIẢNG VIÊN HƯỚNG DẪN
PHẠM MINH QUÂN

TP. HỒ CHÍ MINH, 2022

THÔNG TIN HỘI ĐỒNG CHẤM KHÓA LUẬN TỐT NGHIỆP

Hội đồng chấm khóa luận tốt nghiệp, thành lập theo Quyết định số 66/QĐ-ĐHCNTT ngày 14 tháng 02 năm 2022 của Hiệu trưởng Trường Đại học Công nghệ Thông tin.

LỜI CẢM ƠN

Được sự phân công của quý thầy cô khoa Kỹ Thuật Máy Tính, Trường Đại Học Công Nghệ Thông Tin, sau hơn bốn năm chúng em cũng đã hoàn thành Khóa luận tốt nghiệp. Để hoàn thành khóa luận, ngoài sự nỗ lực học hỏi của bản thân còn có sự hướng dẫn tận tình của thầy cô, bạn bè. Chúng em chân thành cảm ơn giảng viên – ThS. Phạm Minh Quân, người đã hướng dẫn cho chúng em trong suốt thời gian làm khóa luận. Mặc dù thầy bận công tác nhưng không ngần ngại chỉ dẫn, định hướng, để chúng em hoàn thành tốt khóa luận này. Một lần nữa chúng em chân thành cảm ơn thầy và chúc thầy dồi dào sức khỏe.

Tuy nhiên vì kiến thức chuyên môn còn hạn chế và bản thân còn thiếu nhiều kinh nghiệm thực tiễn nên nội dung của báo cáo không tránh khỏi những thiếu sót, chúng em rất mong nhận sự góp ý, chỉ bảo thêm của quý thầy cô cùng toàn thể các bạn để báo cáo này được hoàn thiện hơn.

Một lần nữa xin gửi đến thầy cô, bạn bè lời cảm ơn chân thành và tốt đẹp nhất!

MỤC LỤC

Chương 1.	TỔNG QUAN	3
1.1.	Lý do chọn đề tài	3
1.2.	Tình hình hiện nay.....	4
1.2.1.	Tình hình nghiên cứu trong nước	4
1.2.2.	Tình hình nghiên cứu trên thế giới	5
1.3.	Mục tiêu.....	7
1.4.	Phương pháp thực hiện.....	7
1.4.1.	Tìm hiểu lý thuyết.....	8
1.4.2.	Dữ liệu đầu vào.....	8
1.4.3.	Xử lý dữ liệu	9
1.4.4.	Dữ liệu đầu ra	9
Chương 2.	CƠ SỞ LÝ THUYẾT	10
2.1.	Tổng quan về Real-Time Operating System - RTOS	10
2.1.1.	Khái niệm RTOS	10
2.1.2.	Một số khái niệm trong RTOS	11
2.1.3.	Một số chức năng của RTOS.....	13
2.2.	Tổng quan về hệ điều hành dành cho robot – ROS.....	19
2.2.1.	Khái niệm về ROS sự ra đời của ROS2	19
2.2.2.	Chức năng của ROS2	22
2.2.3.	DDS trên ROS2	22
2.2.4.	Một số khái niệm trong ROS2.....	22
2.2.5.	Một số tính năng ROS2 hỗ trợ.....	26
2.2.6.	Kết luận.....	26

2.3.	Laser Imaging, Detection and Ranging (LIDAR)	26
2.3.1.	Simultaneous Localization and Mapping (SLAM)	27
2.3.2.	Giải thuật Cartographer ROS	29
2.4.	Hệ thống Navigation2 (Nav2)	30
2.4.1.	Tổng quan về Nav2.....	31
2.4.2.	Module Costmap 2D.....	32
2.5.	Phần cứng	33
2.5.1.	Nvidia Jetson TX2 Developer Kit	33
2.5.2.	STM32F4 Discovery Kit	34
2.5.3.	Cảm biến RPLidar A1	35
2.5.4.	Khung robot	36
2.5.5.	Một số module khác	37
Chương 3.	PHÂN TÍCH VÀ THIẾT KẾ HỆ THỐNG.....	41
3.1.	Phân tích phần cứng	41
3.1.1.	Phân tích khung xe	41
3.1.2.	Liên kết các thành phần xe	42
3.2.	Phân tích phần mềm	43
3.2.1.	Phần mềm trên máy tính cá nhân	43
3.2.2.	Phần ROS2 trên Nvidia Jetson TX2 Developer Kit	43
3.2.3.	Phần RTOS trên STM32F429 Discovery Kit.....	45
3.3.	Tính toán Odometry, thuật toán Dijkstra và Dynamic Window Approach	47
3.3.1.	Tính toán Odometry	47
3.3.2.	Thuật toán Dynamic Window Approach.....	52

Chương 4.	THỰC NGHIỆM VÀ ĐÁNH GIÁ	55
4.1.	Phần khung robot.....	55
4.2.	ROS2 và các phần phụ thuộc	56
4.2.1.	ROS2 Foxy	56
4.2.2.	Nav2.....	57
4.2.3.	Micro-ROS	57
4.2.4.	RTOS	57
4.3.	Đánh giá bản đồ 2D.....	58
4.4.	Đánh giá Navigation Goal.....	59
4.4.1.	Thực nghiệm không có vật cản.....	62
4.4.2.	Thực nghiệm có vật cản.....	62
Chương 5.	KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN	64
5.1.	Kết quả đạt được.....	64
5.2.	Ưu điểm và nhược điểm của khóa luận.....	64
5.3.	Hướng phát triển.....	65

DANH MỤC HÌNH

Hình 1.1: Mô hình robot của nhóm sinh viên Trần Hoàng Phương và Võ Anh Tuấn [4]	5
Hình 1.2: Mô hình robot của nhóm sinh viên Phan Anh Kiệt và Lê Chí Bảo [5]..	5
Hình 1.3: Mô hình robot Turtlebot3 Waffle Pi và Burger của hãng ROBOTIS [6]	6
Hình 1.4: Mô hình robot R1mini của hãng OMOROBOT [7]	6
Hình 1.5: Sơ đồ khối cấu trúc của robot.....	8
Hình 2.1: Hệ điều hành RTOS [8]	10
Hình 2.2: Kernel trong RTOS [8]	11
Hình 2.3: Mô hình trạng thái của task trong RTOS [8]	12
Hình 2.4: Lập lịch theo Round-robin [8]	13
Hình 2.5: Lập lịch theo Priority base [8]	14
Hình 2.6: Lập lịch theo Priority-based pre-emptive [8]	14
Hình 2.7: Signal Events trong RTOS [8].....	16
Hình 2.8: Message Queue trong RTOS [8]	17
Hình 2.9: Mail Queue trong RTOS [8].....	18
Hình 2.10: Mutex trong RTOS [8]	19
Hình 2.11: Logo của ROS	19
Hình 2.12: Các phiên bản của ROS2 [9]	21
Hình 2.13: Package trong ROS2.....	23
Hình 2.14: ROS2 workspace	23
Hình 2.15: Node trong ROS2	24
Hình 2.16: Topic trong ROS2.....	24
Hình 2.17: Service trong ROS2	25
Hình 2.18: Action trong ROS2	25
Hình 2.19: Hình ảnh minh họa LiDAR	27
Hình 2.20: Hình ảnh minh họa dùng SLAM để vẽ bản đồ 2D [10]	29
Hình 2.21: Mô hình Cartographer trong ROS2	30

Hình 2.22: Mô hình Nav2 trong ROS [7].....	32
Hình 2.23: Nvidia Jetson TX2 carrier board	34
Hình 2.24: STM32F429 Discovery Kit	35
Hình 2.25: Hình ảnh mô tả các thông số của Rplidar A1	36
Hình 2.26: Khung xe vẽ trên phần mềm solidworks	37
Hình 2.27: Mạch nguồn giảm áp DC-DC.....	38
Hình 2.28: Servo driver MSD_E10A	39
Hình 2.29: Cảm biến gia tốc GY-521 6DOF IMU MPU6050	39
Hình 2.30: Động Cơ DC Servo JGB37 DC.....	40
Hình 2.31: Cảm biến hồng ngoại E3F-DS30Y2.....	40
Hình 3.1: Khung xe dựa theo mô hình Mars rovers	41
Hình 3.2: Khung xe hiện tại.....	42
Hình 3.3: Sơ đồ khối mô tả hệ thống trên STM32F429 Discovery Kit	45
Hình 3.4: Sơ đồ bố trí tọa độ của robot	48
Hình 3.5: Qua một khoảng thời gian rất nhỏ, chuyển động robot có thể gần bằng một cung tròn [4].....	49
Hình 3.6: Hai đường cung tròn tạo bởi hai bánh xe ảo [4].....	50
Hình 3.7: Vận tốc cho phép V_a trong DWA [11].....	53
Hình 3.8: Heading của robot trong DWA [11].....	54
Hình 4.1: Hình chụp thực tế của khung xe.....	55
Hình 4.2: ROS2 foxy	56
Hình 4.3: Kết quả quá trình vẽ bản đồ 2D của môi trường hoạt động	58
Hình 4.4: Đường đi được tạo ra từ thuật toán DWB	60
Hình 4.5: Sơ đồ mô tả giải thuật xử lý khi robot phát hiện vật cản ngoài bản đồ.....	61
Hình 4.6: Trạng thái Recovery	62

DANH MỤC BẢNG

Bảng 3.1: Các package để sử dụng robot	44
Bảng 3.2: Các package được kế thừa lại để thực hiện hệ thống.....	44
Bảng 3.3: Các tác vụ trong RTOS	46
Bảng 4.1: Bảng thông số thực nghiệm không có vật cản	62
Bảng 4.2: Bảng thông số thực nghiệm có vật cản	63

DANH MỤC TỪ VIẾT TẮT

AHB	Advanced High-Performance Bus
AMCL	Adaptive Monte Carlo Localization
APB	Advanced Peripheral Bus
CPU	Central Processing Unit
DC	Direct Current
DDS	Data Distribution Service
DDSI-RTPS	DDS-Interoperability Real Time Publish Subscribe
DSP	Digital Signal Processor
DWA	Dynamic Window Approach
FPU	Floating-Point Unit
GPS	The Global Positioning System
GPU	Graphics Processing Unit
GUI	Graphical User Interface
IMU	Inertial Measurement Unit
LIDAR	Light Detection and Ranging
PCL	Point Cloud Library
PID	Proportional–Integral–Derivative controller
RAM	Random Access Memory
ROS	Robot Operating System
RTOS	Real-Time Operating System
SLAM	Simultaneous Localization and Mapping
TF2	Transform Version 2

TÓM TẮT KHÓA LUẬN

Khóa luận “NGHIÊN CỨU VÀ TÍCH HỢP ROS2 VÀ RTOS CHO ROBOT TỰ HÀNH” ứng dụng LIDAR và Navigation Stack để thiết kế nên robot có khả năng vẽ được bản đồ khu vực hoạt động, tự hoạch định đường đi và đi đến nơi chỉ điểm trên bản đồ kết hợp tránh vật cản. Ứng dụng các thuật toán tìm đường đi để robot có thể di chuyển đến nơi chỉ định chỉ với các thao tác đơn giản nhất.

Với hệ thống LIDAR, robot sử dụng cảm biến Rplidar A1 từ thương hiệu SLAMTEC được phát triển để sử dụng vào các ứng dụng phát hiện vật cản, lập bản đồ bằng tia laser, hỗ trợ định vị xe hay robot tự hành,...

Bên cạnh đó STM32F429 Discovery Kit sử dụng Encoder để tính toán vận tốc và số vòng của bánh xe để có thể biết được trạng thái của robot. Từ đó áp dụng các thuật toán định vị và hoạch định đường đi để robot có thể tự động di chuyển đến nơi chỉ định trên bản đồ một cách chính xác và nhanh nhất có thể.

Để robot hoàn thiện và hoạt động tốt các hệ thống riêng lẻ phải được kết hợp và đồng bộ trên một hệ điều hành quy nhất. Khi đó, hệ điều hành dành cho robot (ROS) là một lựa chọn rất phù hợp với hệ thống. ROS là một hệ thống phần mềm giúp phát triển robot với các thư viện và công cụ được hỗ trợ để truyền dữ liệu giữa các chương trình, giữa robot và máy tính nhờ vào kiến trúc các gói trên hệ thống của robot. Ở đề tài lần này, nhóm sẽ tập trung chủ yếu vào nghiên cứu về ROS2 (hệ điều hành dành cho robot phiên bản thứ 2) với những tính năng mới ưu việt hơn so với ROS, các gói dành cho Rplidar và các gói phục vụ SLAM, định vị và hoạch định đường đi cho robot trên ROS2. Đi cùng với ROS2, nhóm sẽ nghiên cứu tích hợp hệ điều hành thời gian thực (RTOS) với chức năng giúp dễ dàng điều khiển và quản lý phần cứng.

Cuối cùng, để hoàn thiện hệ thống Navigation 2 Stack, robot sẽ được chia làm ba phần. Phần thứ nhất là Nvidia Jetson TX2 Developer Kit, nơi thực thi các thuật toán định vị và hoạch định đường đi. Phần thứ hai là máy tính cá nhân nơi quản lý các thông tin của robot. Phần cuối cũng là tầng nền với STM32F429

Discovery Kit nhận dữ liệu từ các cảm biến IMU và Encoder và tính toán trạng thái của robot, đồng thời điều khiển bốn động cơ DC để robot có thể di chuyển. Ba phần chính sẽ được liên kết chặt chẽ với nhau trên một hệ thống là Robot Operating System (ROS2).

Chương 1. TỔNG QUAN

1.1. Lý do chọn đề tài

Những năm gần đây tự động hóa và robotics phát triển nhanh và mạnh mẽ con người ngày càng cần đến các công cụ tự động nhằm nâng cao năng suất lao động cũng như tối ưu thời gian và chi phí. Từ đó đã phát triển rất nhiều loại robot tự động khác nhau đáp ứng các nhu cầu khác nhau như cánh tay robot trong công nghiệp, robot xử lý ảnh và phân loại sản phẩm... Trong đó, robot tự hoạch định đường đi là một dạng robot rất quan trọng được ứng dụng rộng rãi trong nhiều lĩnh vực đời sống và công nghiệp có thể kể đến là robot lau nhà, robot vận chuyển hàng hóa trong nhà máy... Hơn thế nữa, cộng đồng người dùng, nghiên cứu và phát triển robot trên Robot Operating System (ROS) hiện nay khá là lớn mạnh, giúp người dùng dễ dàng tiếp cận, xây dựng một robot phức tạp trở nên đơn giản hơn. Để robot hoàn thiện và hoạt động tốt các hệ thống riêng lẻ phải được kết hợp và đồng bộ trên một hệ điều hành duy nhất. Khi đó, ROS là một lựa chọn rất phù hợp. ROS là một hệ thống phần mềm giúp phát triển robot với các thư viện và công cụ được hỗ trợ để truyền dữ liệu giữa các chương trình, giữa robot và máy tính nhờ vào kiến trúc các gói trên hệ thống của robot.

Để có thể điều khiển robot di chuyển mượt mà và ổn định thì cần một hệ điều hành thời gian thực với khả năng xử lý dữ liệu siêu nhanh dành cho robot ở đây nhóm sẽ sử dụng Real-time Operating System (RTOS) – hệ điều hành thường được nhúng trong các loại vi điều khiển nơi mà các tài nguyên bên trong rất hữu hạn nên chỉ một sự chậm trễ cũng có thể làm hệ thống làm việc hoàn toàn sai lệch.

Trên thế giới các công trình nghiên cứu về robot tự động hoạch định đường đi (Navigation Robot) đã được thực hiện và cải tiến rất nhiều. Tuy nhiên tình hình phát triển trong nước vẫn còn hạn chế nên nhóm thực hiện tìm hiểu về ROS2 và RTOS dựa vào đó xây dựng robot xe mini tích hợp navigation. Bên cạnh đó, việc thực hiện đề tài sẽ giúp cho nhóm có thêm được kiến thức, kinh nghiệm cũng như cái nhìn

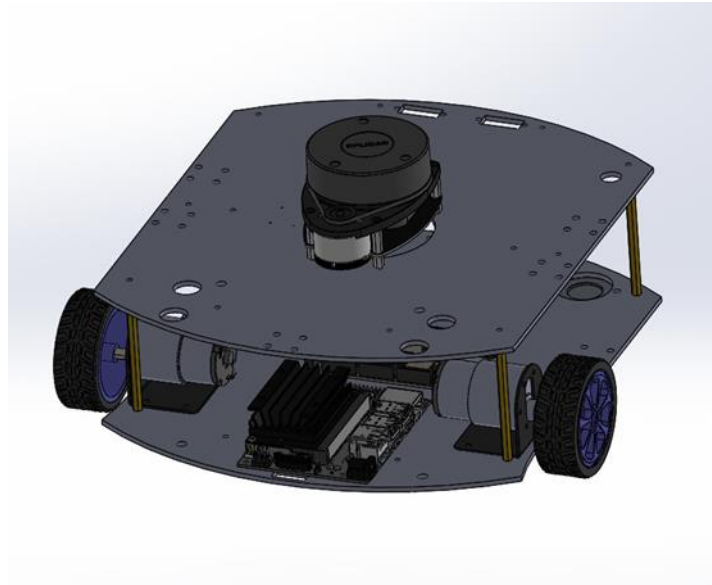
tổng quan trong quá trình thiết kế robot tự hành. Đây cũng là lý do nghiên cứu của nhóm trong quá trình thực hiện khóa luận tốt nghiệp lần này.

1.2. Tình hình hiện nay

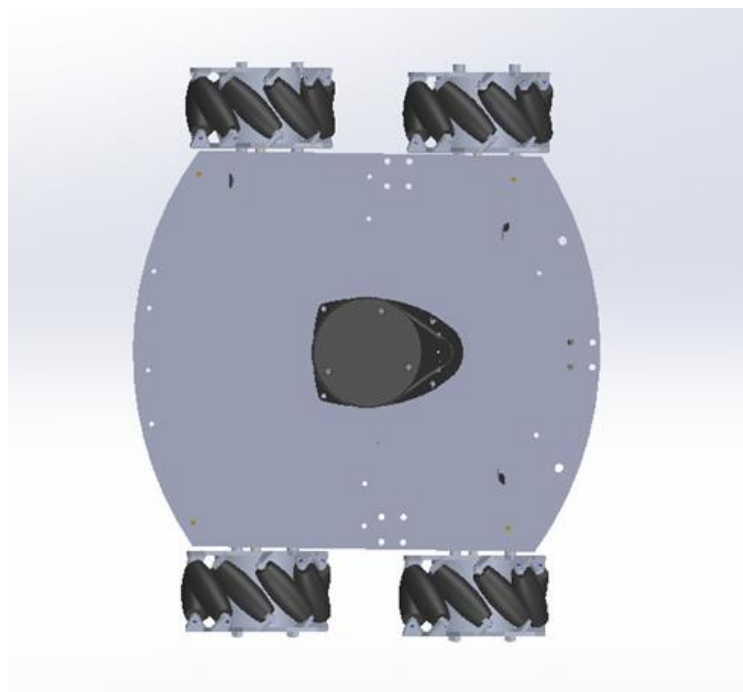
Do công nghệ xe tự hành mới được phổ biến tại Việt Nam trong những năm gần đây, nên những sản phẩm phục vụ cho đào tạo nghiên cứu còn hạn chế, nhỏ lẻ và không đồng nhất. Có những tổ chức nghiên cứu, đầu tư vào xe tự hành với mức kinh phí khổng lồ như Vin Group [1], FPT [2], Phenikaa [3],... với những cuộc thi về xe tự hành tổ chức cho sinh viên và các bạn học sinh trên toàn cả nước. Hiện tại ở trường Đại học Công nghệ Thông tin cũng đã có những bước đi đầu tiên khi đã tiến hành các cuộc thi về xe tự hành trong sinh viên do khoa Kỹ thuật Máy tính chủ trì. Việc xây dựng một hệ thống robot hoàn chỉnh tích hợp hệ thống ROS2 và RTOS sẽ giúp tạo ra hướng đi mới cho việc xây dựng và phát triển robot, giúp đồng bộ trên một hệ điều hành và dễ kế thừa phát triển, xây dựng mô hình xe ngày càng hoàn thiện cho các cuộc thi học thuật ở trường có thể mở rộng hơn nữa.

1.2.1. Tình hình nghiên cứu trong nước

Hiện tại, tại Khoa Kỹ thuật Máy tính thuộc trường Đại học Công nghệ Thông tin đã có đề tài “Robot tránh vật cản sử dụng công nghệ LIDAR” của nhóm sinh viên Trần Hoàng Phương và Võ Anh Tuấn dưới sự hướng dẫn của ThS. Phan Đình Duy (Hình 1.1) [4] và đề tài “Xây dựng robot tự hành quét bản đồ trong nhà và tránh vật cản” của nhóm sinh viên Phan Anh Kiệt và Lê Chí Bảo dưới sự hướng dẫn của ThS. Nguyễn Duy Xuân Bách (Hình 1.2) [5]. Cả hai đề tài này tuy là có mô hình robot khác nhau nhưng về cơ bản đều xây dựng một robot có khả năng vẽ bản đồ khu vực hoạt động bằng Light Detection and Ranging (LIDAR) và áp dụng giải thuật Simultaneous Localization and Mapping (SLAM), sau đó sẽ tiến hành tự hoạch định đường đi và di chuyển đến một vị trí xác định trên bản đồ. Điểm chung của hai đề tài này là đều sử dụng ROS trên máy tính nhúng Nvidia Jetson Nano để thực thi các giải thuật cũng như sử dụng Kit Arduino Mega để quản lý và điều khiển phần cứng.



Hình 1.1: Mô hình robot của nhóm sinh viên Trần Hoàng Phương và Võ Anh Tuấn [4]

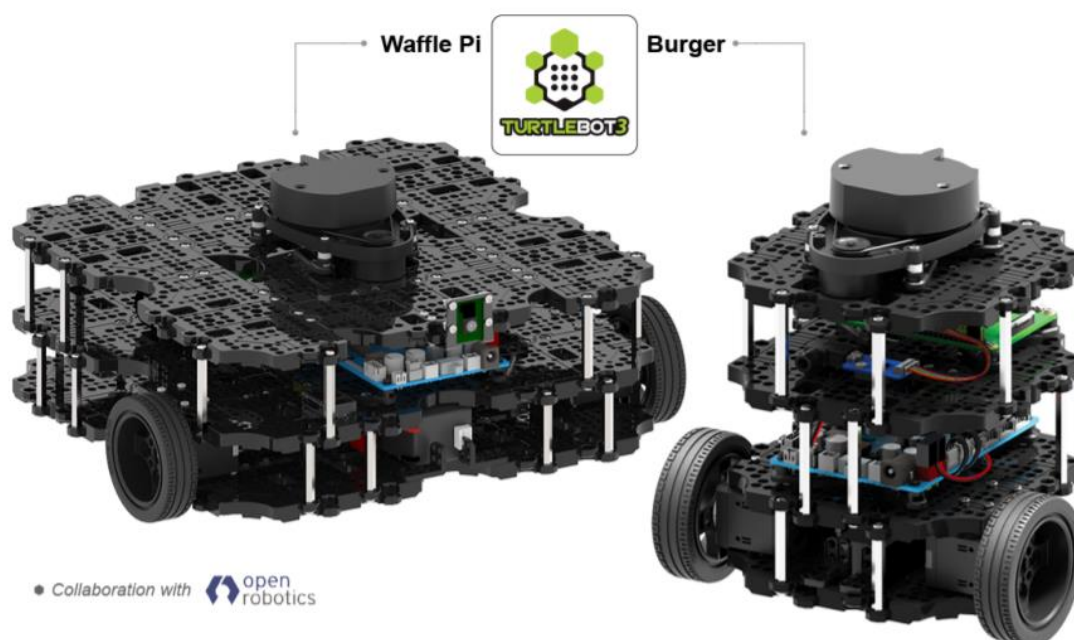


Hình 1.2: Mô hình robot của nhóm sinh viên Phan Anh Kiệt và Lê Chí Bảo [5]

1.2.2. Tình hình nghiên cứu trên thế giới

Trên thế giới cũng đã xuất hiện nhiều mô hình thương mại đến từ hãng ROBOTIS (Hình 1.3) và OMOROBOT (Hình 1.4) của Hàn Quốc. Với sản phẩm này người

dùng chỉ cần sử dụng linh kiện và lập trình theo hướng dẫn của nhà sản xuất thì đã có thể có một mô hình robot hoàn thiện có khả năng tự di chuyển trong không gian nhỏ.



Hình 1.3: Mô hình robot Turtlebot3 Waffle Pi và Burger của hãng ROBOTIS [6]



Hình 1.4: Mô hình robot R1mini của hãng OMOROBOT [7]

1.3. Mục tiêu

Mục tiêu của đề tài là nghiên cứu và tích hợp được ROS2 trên Nvidia Jetson TX2 Developer Kit và RTOS trên STM32F429 Discovery Kit với những ưu điểm hơn so với sử dụng ROS trên Nvidia Jetson Nano Developer Kit và vòng lặp while thông thường trên Kit Arduino Mega của hai đề tài đã được đề cập trước đó vào robot tự hành, để robot có khả năng hoạch định đường đi dựa trên bản đồ đã được vẽ sẵn bằng LIDAR áp dụng thuật toán SLAM. Cụ thể với các mục tiêu như sau:

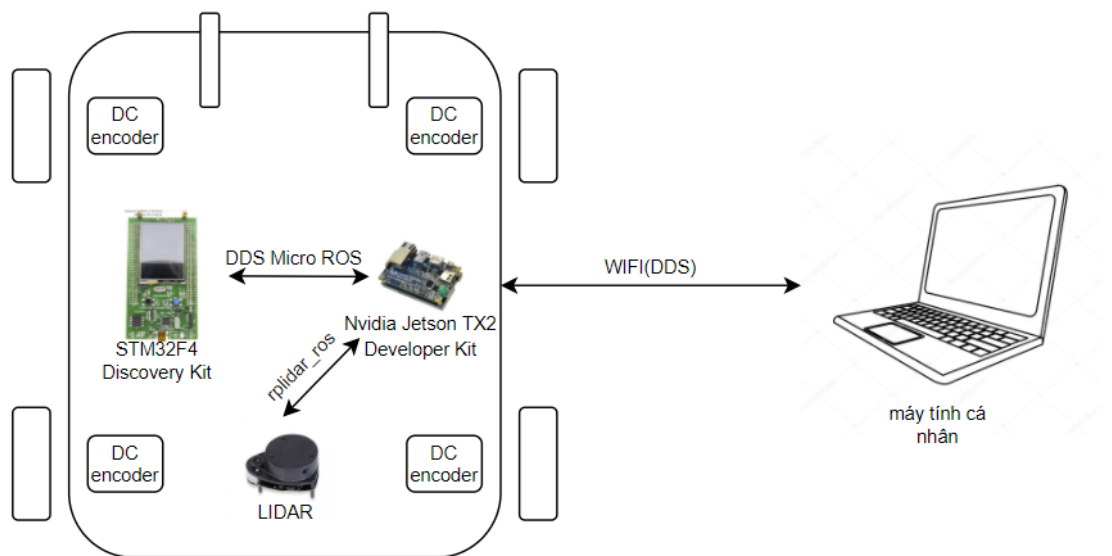
- Thiết kế và xây dựng một mô hình xe trên phần mềm thiết kế đồ họa Solidworks 2019, dựa trên đó tiến hành làm khung robot hoàn chỉnh đầy đủ các cảm biến trên xe như LIDAR, cảm biến hồng ngoại, cảm biến 6 bậc tự do IMU MPU6050.
- Tìm hiểu và thực hiện tích hợp ROS2 lên Nvidia Jetson TX2 Developer Kit.
- Tìm hiểu và hiện thực tích hợp RTOS và Micro-ROS lên STM32F429 Discovery Kit, quản lý phân chia tác vụ điều khiển xe.
- Hiểu và sử dụng thuật toán SLAM trong vẽ bản đồ 2D từ LIDAR, lưu trữ bản đồ khu vực.
- Hiểu và ứng dụng thuật toán của Navigation lên trên ROS2 hoạch định đường đi dựa theo bản đồ được vẽ trước đó.
- Hiểu và thực hiện kết nối các phần riêng lẻ của hệ thống như ROS2, RTOS, máy tính cá nhân thành một thể thống nhất, hoàn thiện robot.

1.4. Phương pháp thực hiện

Để xây dựng một robot dựa trên ROS2 và RTOS với chức năng navigation, nhóm nghiên cứu sẽ xây dựng một robot bằng khung nhôm định hình có bốn động cơ. Đây là mô hình xe để tính toán vận tốc cũng như góc quay hỗ trợ cho việc định vị và hoạch định đường đi chính xác. Cùng với việc sử dụng 4 bộ driver tích hợp bộ điều khiển PID riêng được tinh chỉnh theo động cơ giúp cho việc điều khiển xe di chuyển chính xác như ý, song song nguồn của mỗi bộ phận được tách ra thành nhiều phần khác nhau giúp đảm bảo an toàn từng khu vực và gia tăng thời gian vận hành của robot.

1.4.1. Tìm hiểu lý thuyết

Tìm hiểu các kiến thức về một hệ thống robot navigation. Bắt đầu với việc tìm hiểu về ROS2 trên nền tảng Linux và RTOS trên STM32F429 Discovery Kit sau đó tìm cách tích hợp SLAM để vẽ bản đồ và lưu bản đồ. Tìm hiểu về các thành phần cần thiết cho một hệ thống navigation như giải thuật tính toán Odometry, Joint-States, TF2 - thể hệ thứ hai của thư viện transform. Tiếp đến là các đầu vào cho thuật toán định vị AMCL - một hệ thống bản địa hóa xác suất cho một robot di chuyển ở chế độ 2D và hoạch định đường đi.



Hình 1.5: Sơ đồ khối cấu trúc của robot

1.4.2. Dữ liệu đầu vào

Để có thể thực hiện chức năng navigation thì bản đồ là đầu vào không thể thiếu do đó nhờ vào tín hiệu thu thập được từ cảm biến LIDAR gửi đến Nvidia Jetson TX2 Developer Kit ta sẽ thực hiện công việc vẽ bản đồ bằng các thuật toán SLAM sau đó lưu bản đồ lại. Để vẽ bản đồ ta cần một trình điều khiển robot trên máy tính cá nhân, giúp điều khiển robot có thể quét hết diện tích phòng lần đầu tiên. Tiếp đến, những đầu vào cần thiết cho một robot navigation sẽ bao gồm giải thuật Odometry (vị trí và vận tốc ước tính của robot trong không gian tự do), JointStates (bao gồm

các thông số về vị trí các thành chuyển động của robot), TF2 (sự thay đổi của robot trong các khoảng thời gian). Những dữ liệu này sẽ được tính toán nhờ vào Encoder và cảm biến góc nghiêng IMU (MPU6050) để phục vụ cho quá trình xử lý của các thuật toán khác.

1.4.3. Xử lý dữ liệu

Nvidia Jetson TX2 Developer Kit sẽ lấy dữ liệu từ LIDAR thông qua package `rplidar_ros` (một gói thực thi trên ROS dùng để giao tiếp với LIDAR) trên môi trường ROS2. Sử dụng thuật toán SLAM, ở đây thuật toán do Google phát triển có tên là Cartographer ROS được sử dụng, thông qua cảm biến LIDAR quét và tính toán các vật cản trong khu vực đưa ra bản đồ sau đó lưu lại.

Sau khi có bản đồ khu vực và xác định được vị trí của robot trên bản đồ, hệ thống navigation sẽ được hoạt động. Trên Nvidia Jetson TX2 Developer Kit thực hiện chạy thuật toán định vị AMCL và hoạch định đường đi (DWB) từ các giá trị của LIDAR, cảm biến 6 bậc tự do IMU MPU6050, encoder trả về để tính Odometry, TF2 sau đó truyền tín hiệu điều khiển động cơ cho hệ thống RTOS.

1.4.4. Dữ liệu đầu ra

Dữ liệu đầu ra là dữ liệu điều khiển của ROS2 trên Nvidia Jetson TX2 Developer Kit truyền đến Micro Ros trên RTOS của STM32F429 Discovery Kit để điều khiển robot di chuyển theo hướng đi đã được hoạch định trên ROS2 đồng thời truyền về Rviz2 (là trình hiển thị 3D cho ROS2) trên máy tính cá nhân để giám sát các trạng thái của robot.

Chương 2. CƠ SỞ LÝ THUYẾT

2.1. Tổng quan về Real-Time Operating System - RTOS

2.1.1. Khái niệm RTOS

RTOS (Hình 2.1) hay hệ điều hành thời gian thực thường được nhúng trong các dòng vi điều khiển dùng để điều khiển thiết bị một cách nhanh chóng và đa nhiệm.

Sự khác nhau giữa hệ điều hành thông thường và hệ điều hành RTOS:

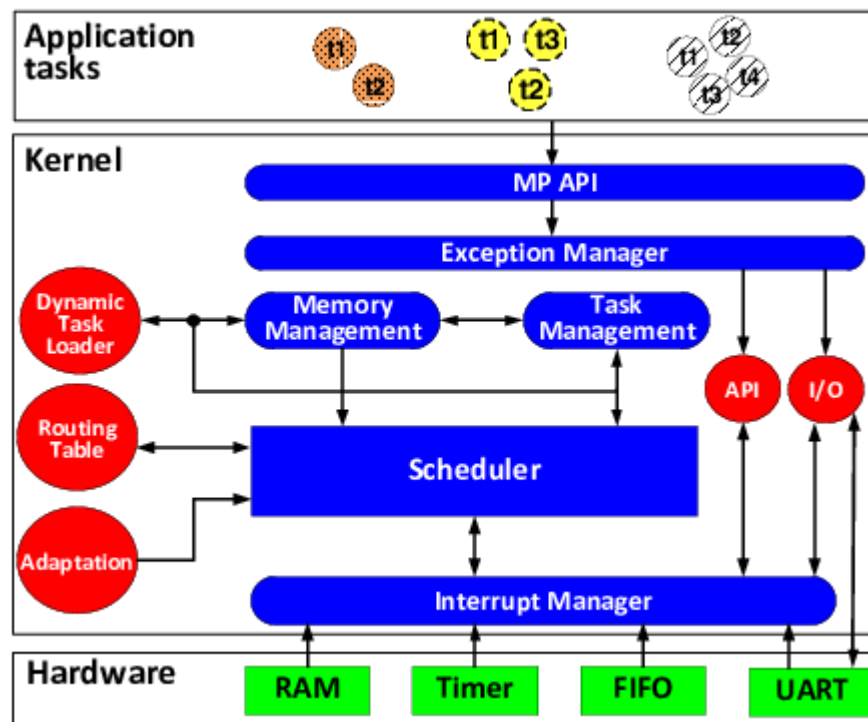
- Hệ điều hành thông thường (non-realtime): như Window, linux, android, ios... chính là thứ mà chúng ta sử dụng hằng ngày. Khi mở một phần mềm trên đó, có thể chúng ta phải chờ nó tải rất lâu, việc chờ đợi này cũng không ảnh hưởng gì cả. Bởi vì đa số phần mềm đó tương tác với con người chứ ít tương tác với các phần mềm hoặc thiết bị khác [8].
- Hệ điều hành thời gian thực (realtime): sinh ra cho các tác vụ cần sự phản hồi nhanh của hệ thống, thường được nhúng trong các loại vi điều khiển và không có giao diện (GUI) tương tác với người dùng. Chúng cần phản hồi nhanh bởi vì đa số các tác vụ tương tác với thiết bị, máy móc khác chứ không phải con người. Các tài nguyên bên trong rất hữu hạn nên chỉ một sự chậm trễ cũng có thể làm hệ thống làm việc hoàn toàn sai lệch [8].



Hình 2.1: Hệ điều hành RTOS [8]

2.1.2. Một số khái niệm trong RTOS

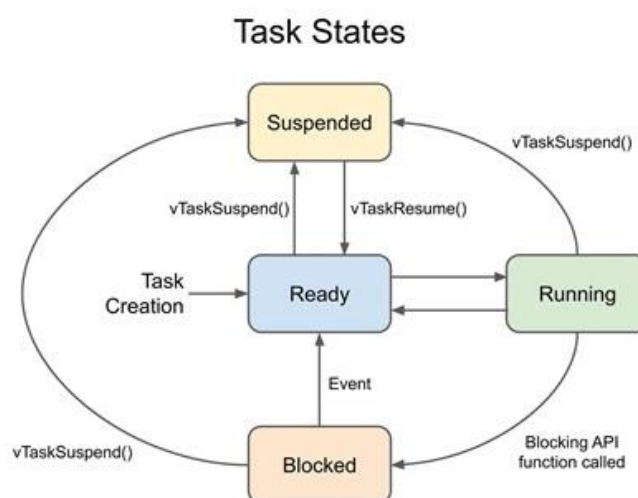
- Kernel (Hình 2.2):
 - o Có nhiệm vụ quản lý và điều phối các task. Mọi sự kiện (Event) như ngắt, Timer, data truyền tới... đều qua Kernel xử lý để quyết định xem nên làm gì tiếp theo.
 - o Thời gian xử lý của Kernel thường rất nhanh nên độ trễ rất thấp.



Hình 2.2: Kernel trong RTOS [8]

- Task – Tác vụ: là một đoạn chương trình thực thi một hoặc nhiều vấn đề gì đó, được Kernel quản lý. Kernel sẽ quản lý việc chuyển đổi giữa các task, nó sẽ lưu lại ngữ cảnh của task sắp bị hủy và khôi phục lại ngữ cảnh của task tiếp theo bằng cách:
 - o Kiểm tra thời gian thực thi đã được định nghĩa trước (time slice được tạo ra bởi ngắt systick).

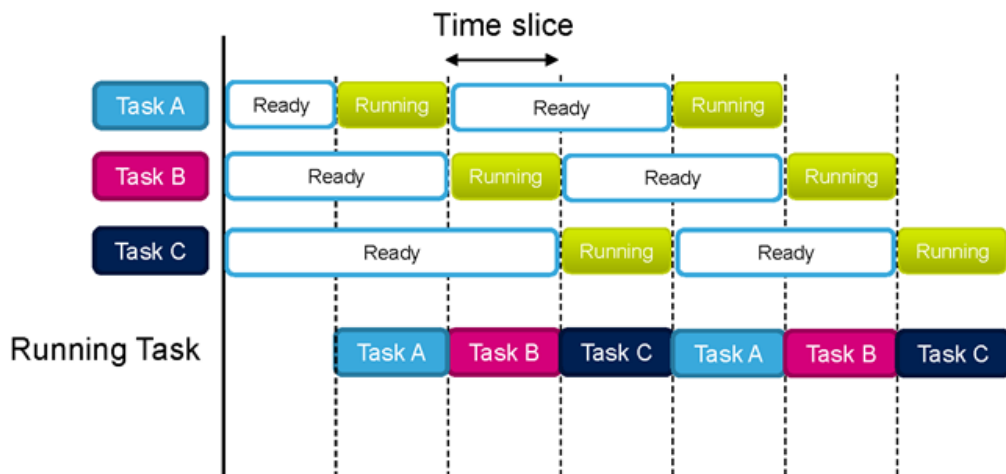
- Khi có các sự kiện unblocking một task có quyền cao hơn xảy ra (signal, queue, semaphore...).
 - Khi task gọi hàm Yield() để ép Kernel chuyển sang các task khác mà không phải chờ cho hết time slice.
 - Khi khởi động thì Kernel sẽ tạo ra một task mặc định gọi là Idle Task.
- Task States – Trạng thái Task: Một Task trong RTOS thường tồn tại ở bốn trạng thái (Hình 2.3).
- Ready: Task đã sẵn sàng để có thể thực thi nhưng chưa được thực thi do có các task khác với độ ưu tiên ngang bằng hoặc hơn đang chạy.
 - Running: Task đang thực thi.
 - Blocked: Task đang chờ 1 sự kiện nào đó xảy ra, sự kiện này có thể là khoảng thời gian hoặc 1 sự kiện nào đó từ task khác.
 - Suspended: Task ở trạng thái treo khi hàm vTaskSuspend() được gọi, về cơ bản thì trạng thái này cũng tương tự như Blocked nhưng điểm khác nhau là “cách” chuyển từ trạng thái hiện tại sang Ready State. Chỉ khi gọi hàm vTaskResume() thì task bị treo mới được chuyển sang trạng thái Ready để có thể thực thi.



Hình 2.3: Mô hình trạng thái của task trong RTOS [8]

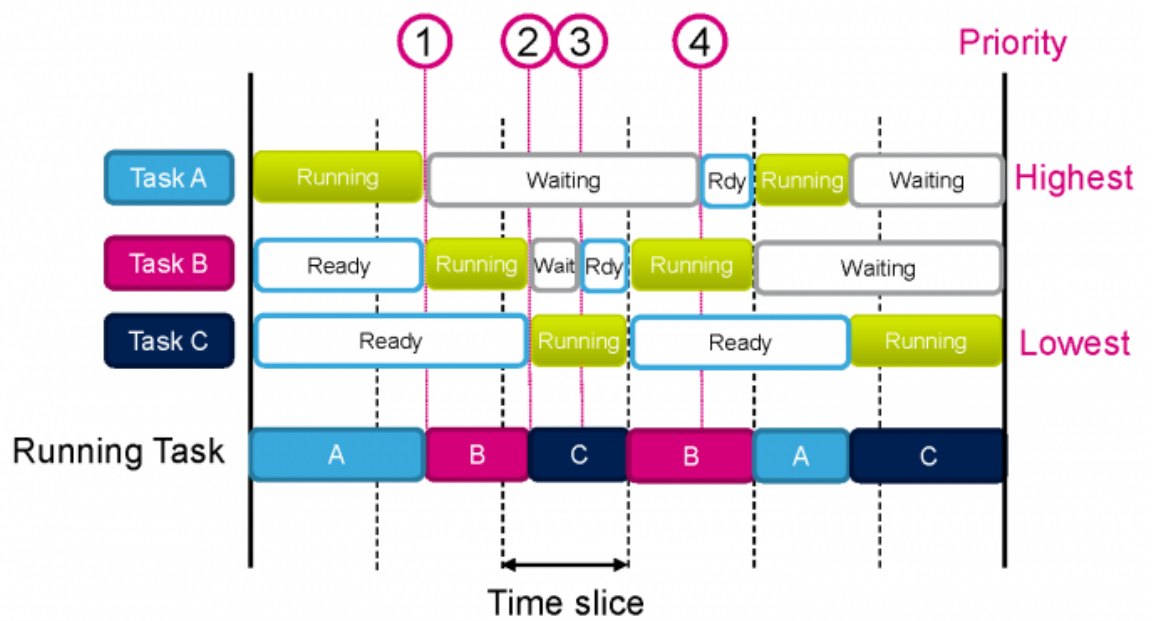
2.1.3. Một số chức năng của RTOS

- Scheduler – Lập lịch: Đây là một thành phần của Kernel quyết định task nào được thực thi và có một số luật cho scheduling như:
 - Cooperative: giống với lập trình thông thường, mỗi task chỉ có thể thực thi khi task đang chạy dừng lại, nhược điểm của nó là task này có thể dùng hết tất cả tài nguyên của CPU.
 - Round-robin: mỗi task được thực hiện trong thời gian định trước (time slice) và không có ưu tiên (Hình 2.4).



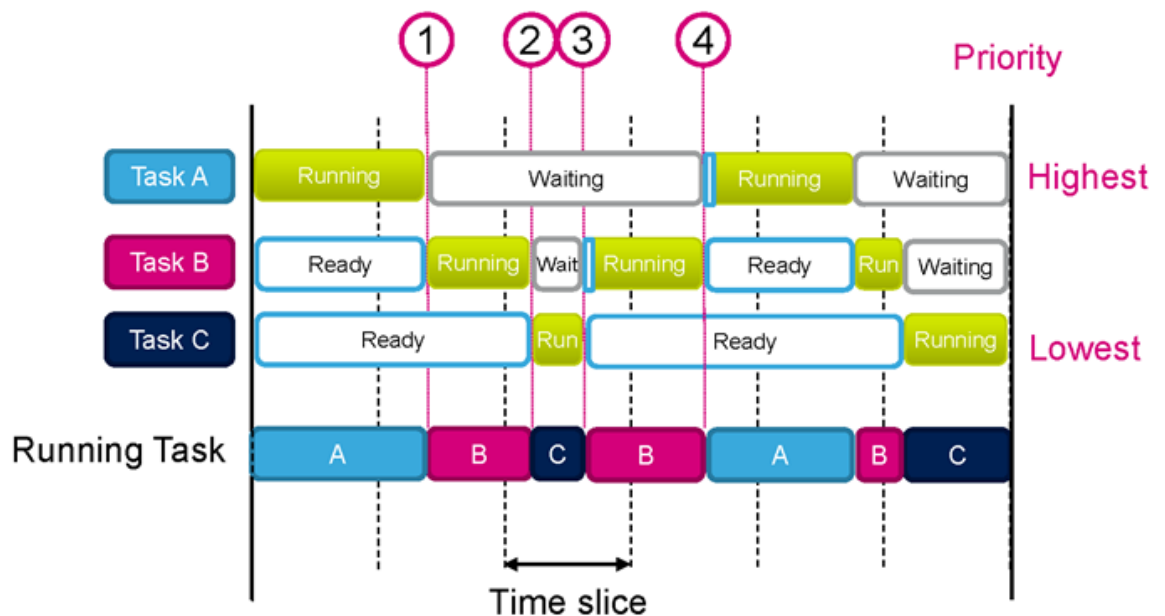
Hình 2.4: Lập lịch theo Round-robin [8]

- Priority-based: Task được phân quyền cao nhất sẽ được thực hiện trước, nếu các task có cùng quyền như nhau thì sẽ giống với round-robin, các task có mức ưu tiên thấp hơn sẽ được thực hiện cho đến cuối time slice (Hình 2.5).



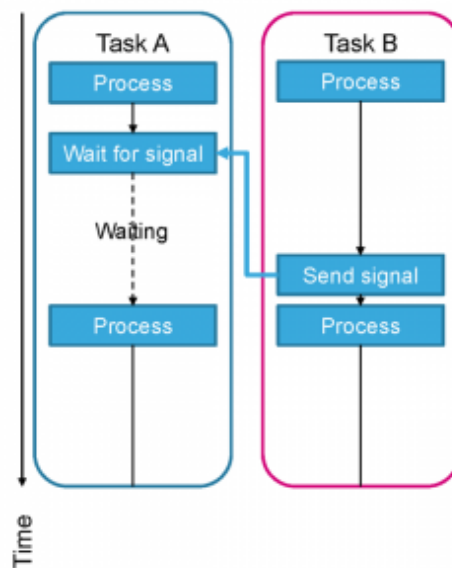
Hình 2.5: Lập lịch theo Priority base [8]

- Priority-based preemptive: Các task có mức ưu tiên cao nhất luôn nhường các task có mức ưu tiên thấp hơn thực thi trước (Hình 2.6).



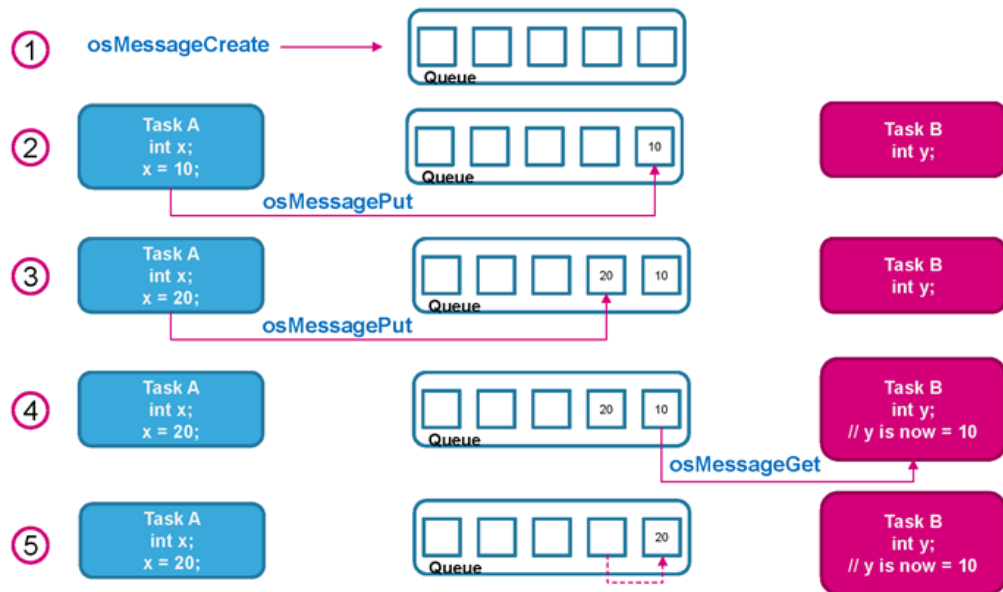
Hình 2.6: Lập lịch theo Priority-based pre-emptive [8]

- Kết nối Inter-task và chia sẻ tài nguyên: Để một chương trình có thể vận hành trơn tru thì các task cần phải kết nối và trao đổi dữ liệu với nhau để có thể chia sẻ tài nguyên, có một số khái niệm cần lưu ý:
 - Với Inter-task Communication:
 - Signal Events – Đồng bộ các task.
 - Message queue – Trao đổi tin nhắn giữa các task trong hoạt động giống như FIFO.
 - Mail queue – Trao đổi dữ liệu giữa các task sử dụng hàng đợi của khối bộ nhớ.
 - Với Resource Sharing:
 - Semaphores – Truy xuất tài nguyên liên tục từ các task khác nhau.
 - Mutex – Đồng bộ hóa truy cập tài nguyên sử dụng Mutual Exclusion.
- Signal Events: Một task sẽ đợi một tín hiệu trước khi thực thi, khi đó task này sẽ nằm ở trạng thái chờ (Waiting) cho đến khi tín hiệu (signal) được bật, lúc này task sẽ bắt đầu thực thi (Hình 2.7). Ta có thể thiết lập một hoặc nhiều tín hiệu trong bất kỳ các task nào khác. Ưu điểm của nó là thực hiện nhanh, sử dụng ít RAM hơn so với semaphore (một kỹ thuật để quản lý nhiều quá trình đồng thời) và message queue nhưng có nhược điểm lại chỉ được dùng khi một task nhận được tín hiệu.



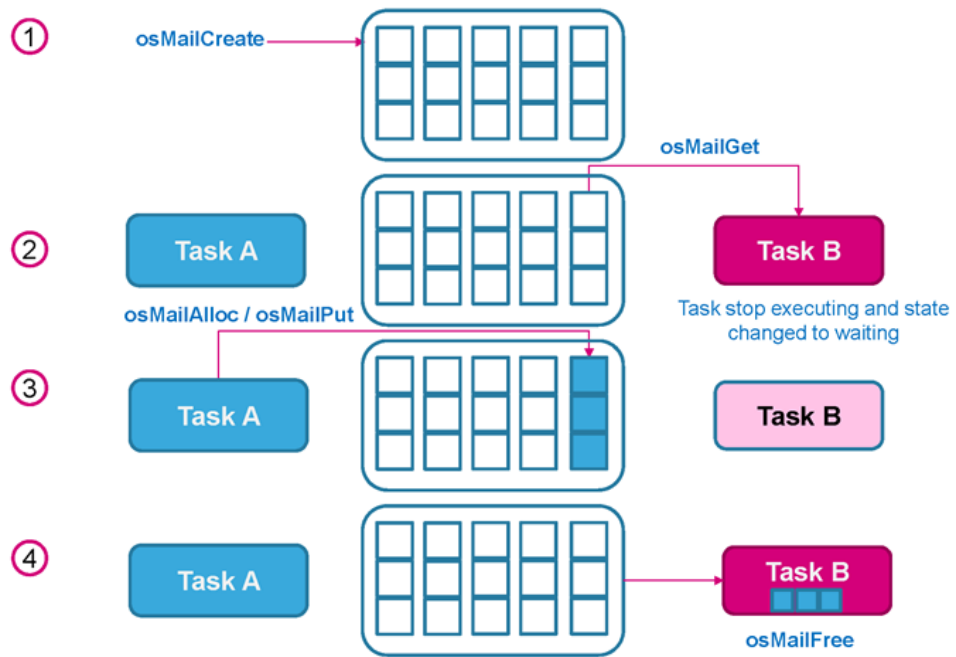
Hình 2.7: Signal Events trong RTOS [8]

- Message Queue: là cơ chế cho phép các task có thể kết nối với nhau, nó là một FIFO buffer được định nghĩa bởi độ dài (số phần tử mà buffer có thể lưu trữ) và kích thước dữ liệu (kích thước của các thành phần trong buffer) (Hình 2.8). Một ứng dụng tiêu biểu là buffer cho Serial I/O, buffer cho lệnh được gửi tới task.
 - Task có thể ghi vào hàng đợi (queue):
 - Task sẽ bị khóa (block) khi gửi dữ liệu tới một message queue đầy đủ.
 - Task sẽ hết bị khóa (unblock) khi bộ nhớ trong message queue trống.
 - Trường hợp nhiều task mà bị block thì task với mức ưu tiên cao nhất sẽ được unblock trước.
 - Task có thể đọc từ hàng đợi (queue):
 - Task sẽ bị block nếu message queue trống.
 - Task sẽ được unblock nếu có dữ liệu trong message queue.
 - Tương tự ghi thì task được unblock dựa trên mức độ ưu tiên.



Hình 2.8: Message Queue trong RTOS [8]

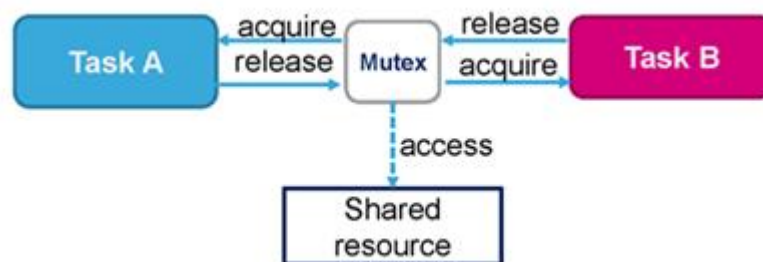
- Mail Queue: Giống như message queue nhưng dữ liệu sẽ được truyền dưới dạng khối (memory block) thay vì dạng đơn (Hình 2.9). Mỗi memory block thì cần phải cấp phát trước khi đưa dữ liệu vào và giải phóng sau khi đưa dữ liệu ra.
 - Gửi dữ liệu với mail queue:
 - Cấp phát bộ nhớ từ mail queue cho dữ liệu được đặt trong mail queue.
 - Lưu dữ liệu cần gửi vào bộ nhớ đã được cấp phát.
 - Đưa dữ liệu vào mail queue.
 - Nhận dữ liệu trong mail queue bởi task khác:
 - Lấy dữ liệu từ mail queue, sẽ có một hàm để trả lại cấu trúc/ đối tượng.
 - Lấy con trỏ chứa dữ liệu.
 - Giải phóng bộ nhớ sau khi sử dụng dữ liệu.



Hình 2.9: Mail Queue trong RTOS [8]

- Semaphore: Được sử dụng để đồng bộ task với các sự kiện khác trong hệ thống. Có 2 loại:
 - Binary semaphore: trường hợp đặc biệt của counting semaphore, có duy nhất 1 token và chỉ có một hoạt động đồng bộ.
 - Counting semaphore: có nhiều token và có nhiều hoạt động đồng bộ.
- Mutex: sử dụng cho việc loại trừ (mutual exclusion), hoạt động như là một token để bảo vệ tài nguyên được chia sẻ. Một task nếu muốn truy cập vào tài nguyên chia sẻ cần yêu cầu (đợi) mutex trước khi truy cập vào tài nguyên chia sẻ và phải đưa ra token khi kết thúc với tài nguyên. Tại mỗi một thời điểm thì chỉ có 1 task có được mutex. Những task khác muốn cùng mutex thì phải block cho đến khi task cũ thả mutex ra (Hình 2.10). Về cơ bản thì Mutex giống như binary semaphore nhưng được sử dụng cho việc loại trừ chứ không phải đồng bộ. Ngoài ra thì nó bao gồm cơ chế thừa kế mức độ ưu tiên (Priority inheritance mechanism) để giảm thiểu vấn đề đảo ngược ưu tiên, cơ chế này có thể hiểu đơn giản qua ví dụ sau:

- Task A (low priority) yêu cầu mutex.
- Task B (high priority) muốn yêu cầu cùng mutex trên.
- Mức độ ưu tiên của Task A sẽ được đưa tạm về Task B để cho phép Task A được thực thi.
- Task A sẽ thả mutex ra, mức độ ưu tiên sẽ được khôi phục lại và cho phép Task B tiếp tục thực thi.



Hình 2.10: Mutex trong RTOS [8]

2.2. Tổng quan về hệ điều hành dành cho robot – ROS

2.2.1. Khái niệm về ROS sự ra đời của ROS2



Hình 2.11: Logo của ROS








Trước đây khi nghiên cứu và thiết kế một robot, mọi người thường sẽ phải dành thời gian để thiết kế phần mềm nhúng trong robot cũng như phần cứng nên đòi hỏi nhiều về kiến thức chuyên môn về nhúng.

Robot Operating System là một hệ điều hành mã nguồn mở dành cho robot có logo như Hình 2.11. Nó cung cấp các dịch vụ tương tự như một hệ điều hành gồm trừu

tượng hóa phần cứng, kiểm soát thiết bị hạ tầng thông báo giữa các quy trình và quản lý package. ROS cung cấp các công cụ và thư viện để chạy trên nhiều máy tính nhúng khác nhau.

Mặc dù ROS tương tự như một hệ điều hành (Operating System) nhưng thực chất chỉ là tập hợp các công cụ, thư viện và quy ước nhằm mục đích đơn giản hóa nhiệm vụ tạo ra các hành vi phức tạp và mạnh mẽ của robot trên nhiều nền tảng máy tính nhúng khác nhau.

Robot rất cần tốc độ phản ứng cũng như độ trễ thấp nhưng bản thân ROS1 không phải hệ điều hành thời gian thực và ROS1 cũng sự kém đa dạng về các nền tảng giao tiếp. ROS2 là một bản sửa đổi lớn của API ROS, tận dụng thư viện và công nghệ hiện đại cho chức năng ROS cốt lõi, bổ sung hỗ trợ mã thời gian thực, phần cứng nhúng, tích hợp thêm một số tính năng cho ROS2 đa dạng và dễ dàng sử dụng trên mọi hệ thống hơn. Vì vậy bài viết này sẽ tập trung về các tính năng cách thức hoạt động của ROS2. Hiện tại, đã có nhiều phiên bản khác nhau của ROS2 được phát hành (Hình 2.12), tuy nhiên đề tài này sẽ tập trung chủ yếu vào nghiên cứu và tích hợp phiên bản ROS Foxy Fitzroy.

Distro	Release date	Logo	EOL date
Humble Hawksbill	May 23rd, 2022		
Galactic Geochelone	May 23rd, 2021		November 2022
Foxy Fitzroy	June 5th, 2020		May 2023
Eloquent Elusor	November 22nd, 2019		November 2020
Dashing Diademata	May 31st, 2019		May 2021
Crystal Clemmys	December 14th, 2018		December 2019
Bouncy Bolson	July 2nd, 2018		July 2019
Ardent Apalone	December 8th, 2017		December 2018
beta3	September 13th, 2017		December 2017
beta2	July 5th, 2017		September 2017
beta1	December 19th, 2016		Jul 2017
alpha1 - alpha8	August 31th, 2015		December 2016

Hình 2.12: Các phiên bản của ROS2 [9]

2.2.2. Chức năng của ROS2

ROS2 là một bộ phát triển mã nguồn mở cho các ứng dụng robot. Mục tiêu chính là cung cấp một nền tảng phần mềm tiêu chuẩn cho người dùng học hỏi tái sử dụng các thành quả trước cho lập trình và phát triển mới. ROS2 hỗ trợ để người dùng có thể sử dụng kết hợp các thư viện khác nhau để có thể làm cho hệ thống chạy tốt nhất như OpenCV, Gazebo, PCL, Moveit, ROS-industrial.

Một số ưu điểm của ROS2 mang lại:

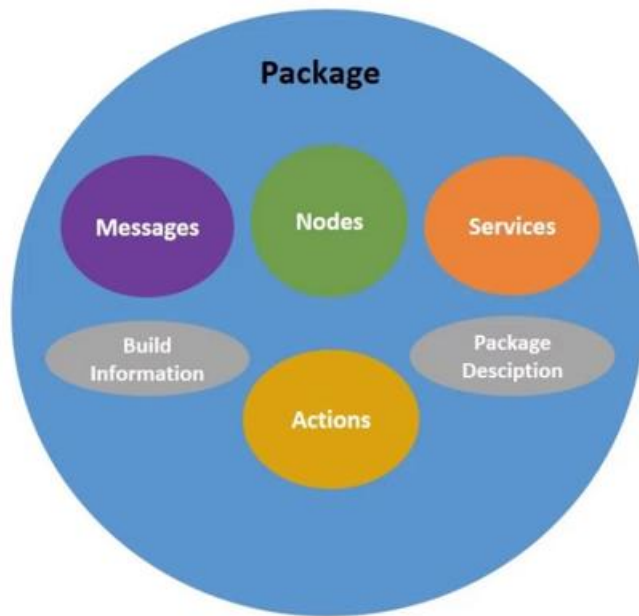
- Trừu tượng phần cứng: thông qua giao diện lập trình giao tiếp với các phần cứng khác nhau.
- Điều khiển thiết bị cấp thấp: thông qua các tính năng thực hiện giao tiếp với các phần cứng.
- Giao tiếp giữa các process: giúp làm giảm khối lượng công việc trên kernel.
- Sử dụng chuẩn giao tiếp DDSI-RTPS (DDS-Interoperability Real Time Publish Subscribe).
- Quản lý gói: dễ dàng quản lý các thư mục của ROS2 từ đó có thể chia sẻ và sử dụng trên các hệ thống khác.

2.2.3. DDS trên ROS2

DDS (Data Distribution Service) cho hệ thống thời gian thực là tiêu chuẩn machine to machine đôi khi được gọi là middleware hoặc connectivity framework. Tiêu chuẩn nhằm mục đích cho phép trao đổi dữ liệu đáng tin cậy, hiệu suất cao, có thể tương tác trong thời gian thực và có thể mở rộng bằng cách sử dụng mẫu đăng ký.

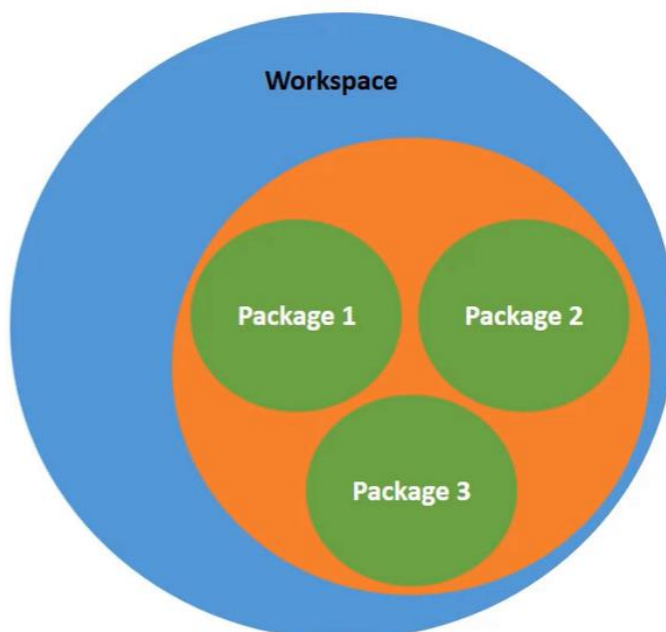
2.2.4. Một số khái niệm trong ROS2

- Package: là một vùng chứa code của ROS2 có thể xem như một container của ROS2, với package của ROS2 có thể thực hiện các thao tác xây dựng cũng như sử dụng một cách dễ dàng (Hình 2.13).



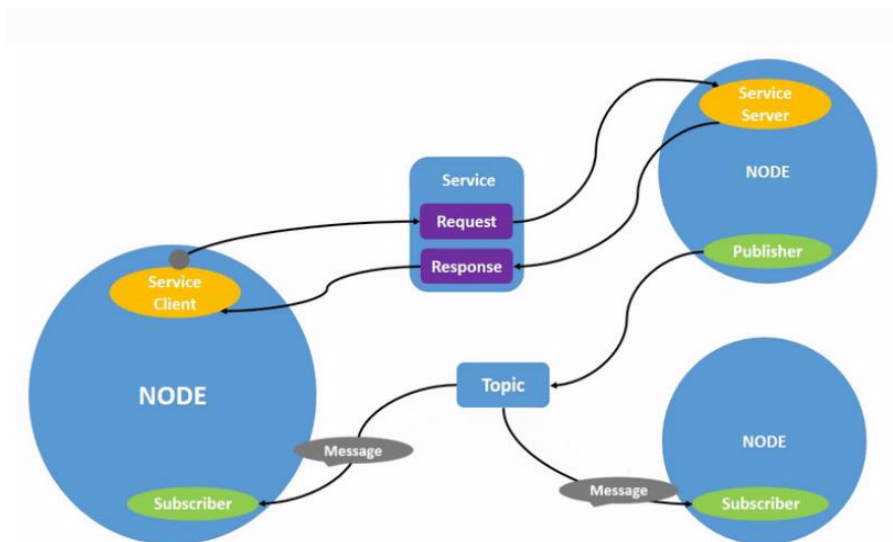
Hình 2.13: Package trong ROS2

- Workspace: là một thư mục chứa các package của ROS2, trước khi muốn sử dụng cần phải cài đặt ROS2 terminal hệ thống muốn sử dụng (Hình 2.14).



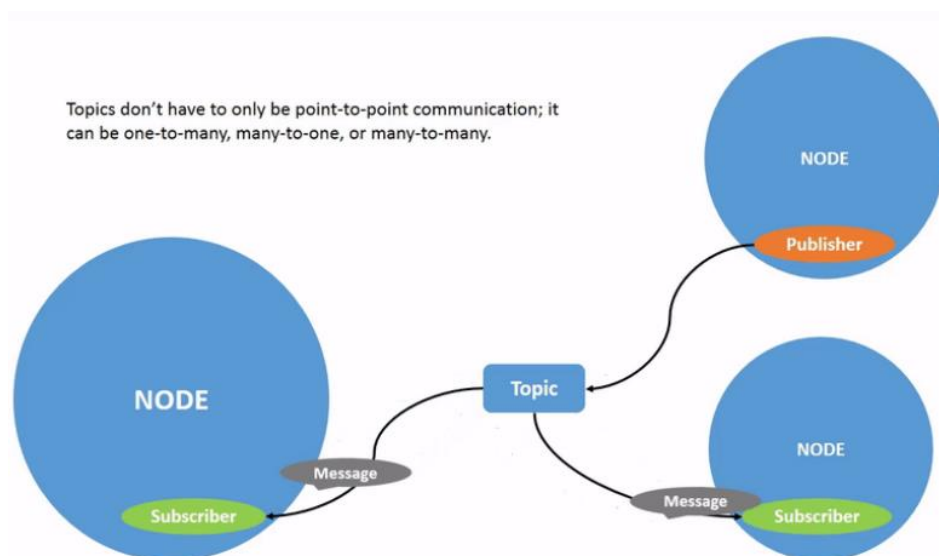
Hình 2.14: ROS2 workspace

- Nodes: mỗi node nằm trong ROS2 sẽ chịu trách nhiệm cho một module duy nhất trong hệ thống như node điều khiển động cơ, node điều khiển servo... và mỗi node đều có thể gửi hoặc nhận dữ liệu thông qua các topic, services, actions và parameters (Hình 2.15).



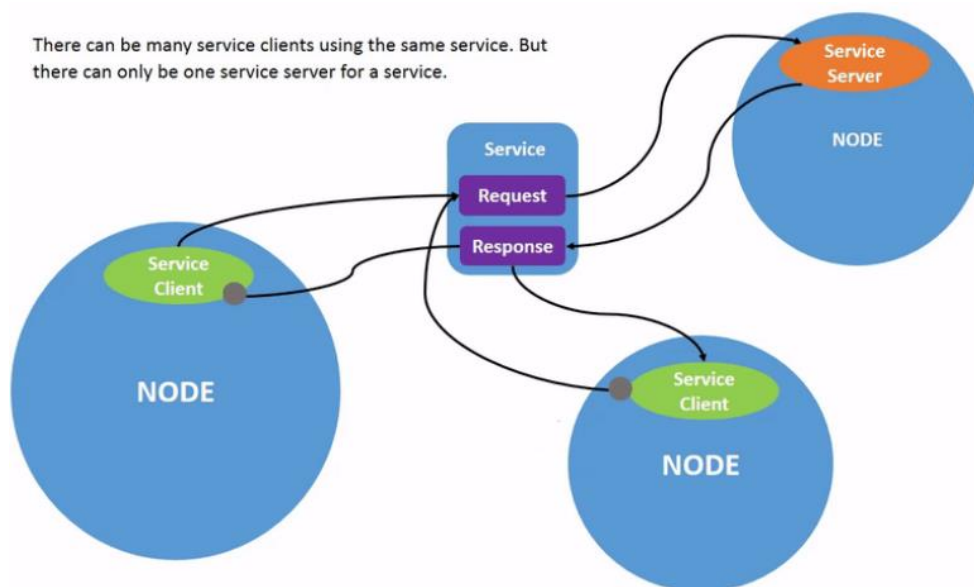
Hình 2.15: Node trong ROS2

- Topic: là một phần quan trọng để dữ liệu di chuyển giữa các node và các phần khác nhau trong hệ thống. Topic sẽ phân phối các message từ các publisher đến các subscriber (Hình 2.16).



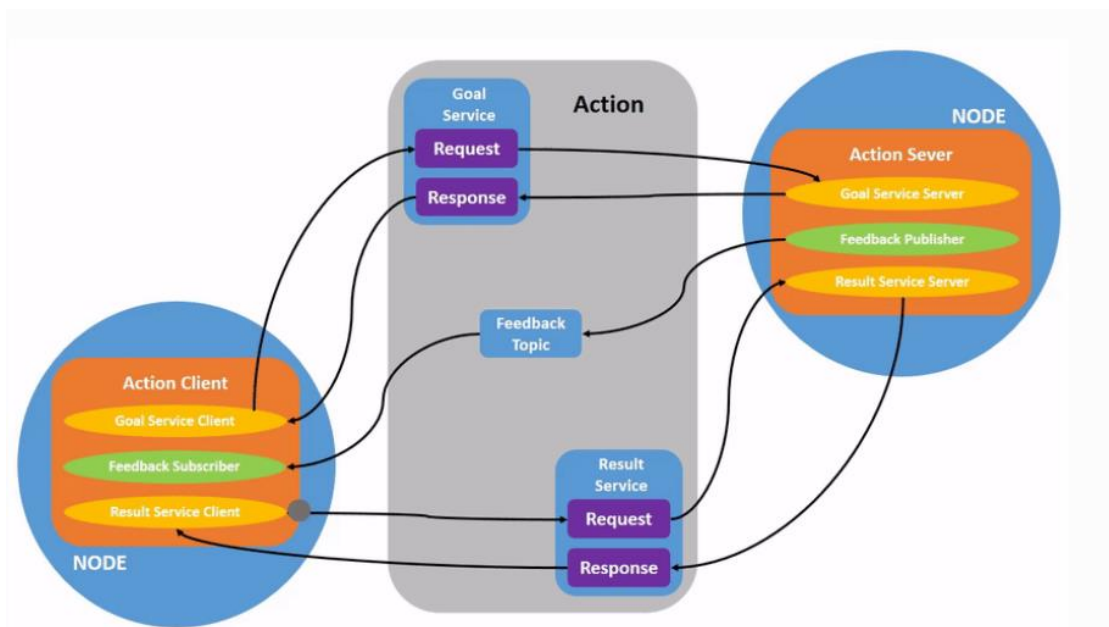
Hình 2.16: Topic trong ROS2

- Services: là một phương thức giao tiếp khác cho các node trên hệ thống ROS2. Các service dựa trên mô hình call-and-response. Khác với topic luôn cho phép các node khởi tạo và cập nhật liên tục thì service chỉ cung cấp dữ liệu khi được gọi từ một node cụ thể (Hình 2.17).



Hình 2.17: Service trong ROS2

- Actions: là một trong các phương thức giao tiếp trong ROS dành cho các dịch vụ chạy lâu dài gồm 3 phần: goal, result và feedback (Hình 2.18).



Hình 2.18: Action trong ROS2

2.2.5. Một số tính năng ROS2 hỗ trợ

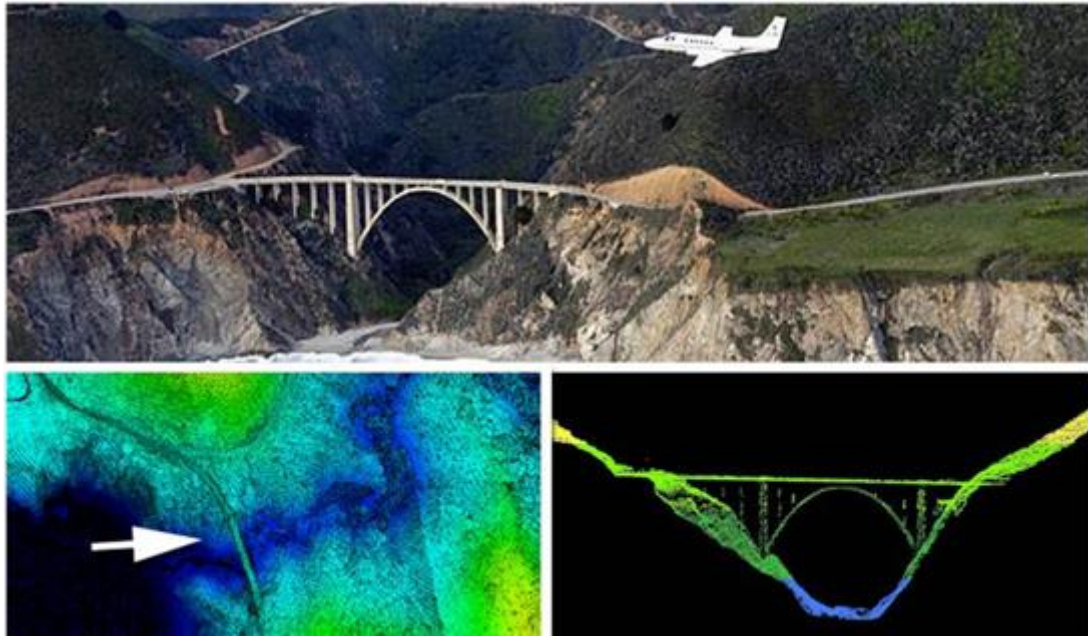
- ROS2 hỗ trợ trên nhiều hệ điều hành khác nhau như ubuntu, OS X El Capitan cũng như trên Windows 10.
- Sử dụng ngôn ngữ lập trình hầu hết các phiên bản của C++ như C++11, C++13, C++17 và từ python 3.5 trở lên.
- Giao tiếp với Micro Ros thông qua DDS.
- Giao tiếp với ROS2 trên máy tính thông qua cơ chế DDS của ROS2.

2.2.6. Kết luận

Trước đây khi chưa xuất hiện hệ điều hành cho robot, mọi nhà thiết kế và nghiên cứu robot sẽ phải dành một lượng lớn thời gian để thiết kế phần mềm nhúng robot, cũng như phần cứng, điều này yêu cầu người lập trình phải có đầy đủ các khả năng về cơ khí, điện tử, lập trình nhúng. Nhưng ngày nay khi công nghệ ngày càng phát triển nên việc kế thừa và phát huy dễ dàng là một điều cần thiết, ROS2 là một sự lựa chọn tốt vì có thể chạy trên đa nền tảng, ngôn ngữ lập trình phổ biến, khả năng đóng gói linh hoạt.

2.3. Laser Imaging, Detection and Ranging (LIDAR)

LIDAR là một phương pháp thu thập dữ liệu về khoảng cách của một vật thể với cảm biến nhờ vào việc chiếu sáng vật thể đó bằng tia Laser và tính toán dựa vào các tia phản xạ lại cảm biến. Trên thế giới LIDAR thường được ứng dụng vào những hệ thống lớn về địa tin học, lâm nghiệp, khảo cổ học, nông nghiệp để xây dựng bản đồ, vật thể ba chiều.



Hình 2.19: Hình ảnh minh họa LiDAR

Bên cạnh đó LIDAR còn được tích hợp vào robot mục đích cải thiện nhận quan cho robot nhờ vào sự chính xác về khoảng cách mà LIDAR mang lại.

LIDAR có thể phát ra tối đa 200.000 xung laser trong mỗi giây. Một bộ LIDAR cơ bản bao gồm một máy phát laser, một máy scan, một bộ thu nhận GPS được tùy biến. Khi một chùm laser được chiếu vào một điểm trên mặt đất, chùm sáng này sẽ bị phản xạ lại. Một cảm biến sẽ thu nhận thông tin của chùm phản xạ để đo khoảng cách dựa theo thời gian di chuyển của xung laser. Kết hợp với dữ liệu về vị trí và phương hướng từ hệ thống định vị GPS cũng như bộ đo quán tính, bộ quét góc, dữ liệu sẽ được đưa ra thành một tập hợp các điểm, gọi là “point cloud”. Mỗi point cloud sẽ có tọa độ xác định trong không gian ba chiều (bao gồm kinh độ, vĩ độ và cao độ) tương ứng với vị trí của nó trên bề mặt Trái Đất. Các điểm này sau đó được đem đi dựng thành mô hình.

2.3.1. Simultaneous Localization and Mapping (SLAM)

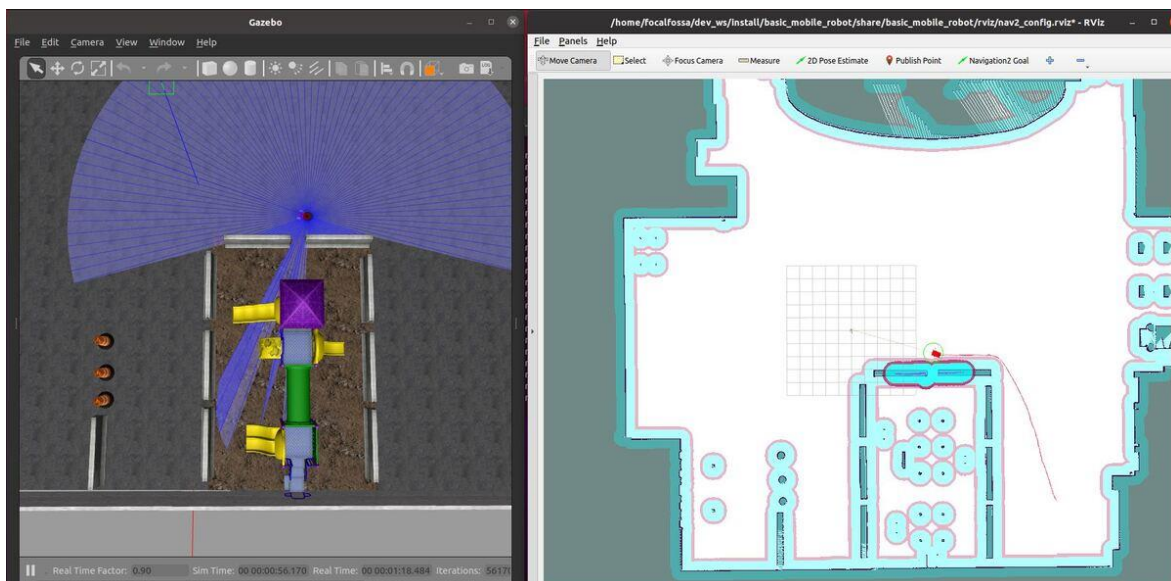
SLAM là một phương pháp để xây dựng và cập nhật bản đồ ở những nơi chưa biết trước đồng thời xác định được vị trí của chủ thể (Hình 2.20). Những thuật toán về

SLAM thường được sử dụng trong các lĩnh vực định vị, vẽ bản đồ, tính toán khoảng cách và định hướng, ngoài ra SLAM còn được áp dụng vào thực tế ảo (virtual reality) và thực tế tăng cường (augmented reality). Để có thể thực hiện SLAM chúng ta cần có những thiết bị ngoại vi như (Camera, Sona, Radar, LIDAR) để nhận thông tin từ bên ngoài.

Các kỹ thuật thống kê được sử dụng để tính gần đúng các phương trình SLAM bao gồm bộ lọc Kalman và bộ lọc Particle (hay còn gọi là phương pháp Monte Carlo). Chúng cung cấp ước tính của hàm xác suất cho tư thế của robot và cho các thông số của bản đồ. Các phương pháp tính gần đúng một cách bảo toàn mô hình trên bằng cách sử dụng giao điểm phương sai có thể tránh phụ thuộc vào các giả định thống kê độc lập để giảm độ phức tạp của thuật toán cho các ứng dụng quy mô lớn. Các phương pháp xấp xỉ khác đạt được hiệu quả tính toán được cải thiện bằng cách sử dụng các vùng giới hạn đơn giản.

Các kỹ thuật tập hợp thành phần chủ yếu dựa trên sự lan truyền khoảng giới hạn. Họ cung cấp một bộ bao gồm tư thế của robot và một tập hợp gần đúng của bản đồ. Điều chỉnh và ước tính sau thử nghiệm (MAP), là một kỹ thuật phổ biến khác cho SLAM sử dụng dữ liệu hình ảnh, cùng ước tính các tư thế và vị trí mốc, tăng độ chính xác của bản đồ và được sử dụng trong các hệ thống SLAM thương mại hóa như ARCore của Google, thay thế dự án thực tế tăng cường trước đó ‘Tango’. MAP tính toán về các tư thế của robot và bản đồ cung cấp dữ liệu cảm biến, thay vì cố gắng ước tính toàn bộ xác suất.

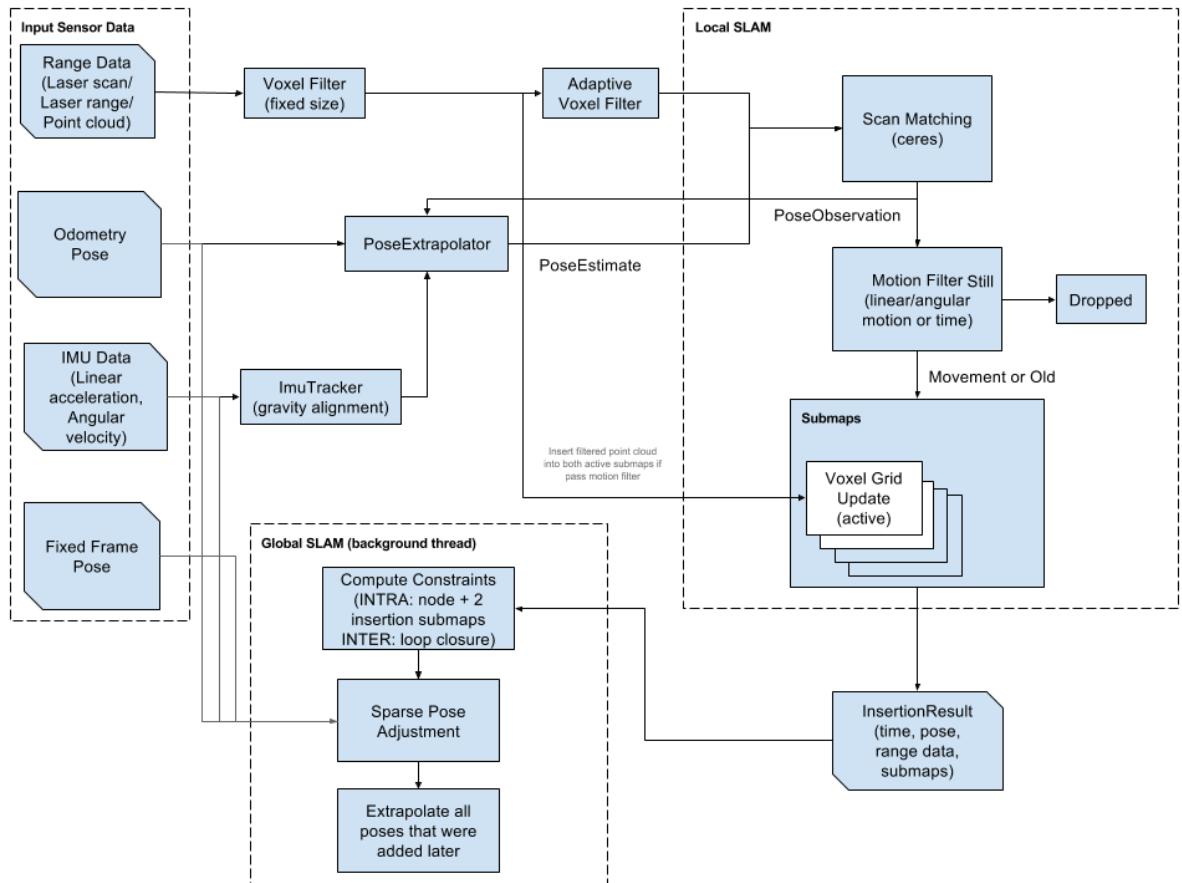
Các thuật toán SLAM vẫn là một lĩnh vực nghiên cứu mới, và thường được yêu cầu và tạo ra giả định khác nhau về các loại bản đồ, cảm biến và mô hình như chi tiết khác nhau.



Hình 2.20: Hình ảnh minh họa dùng SLAM để vẽ bản đồ 2D [10]

2.3.2. Giải thuật Cartographer ROS

Đây là một gói thực thi trên ROS2 do Google phát triển sử dụng dữ liệu từ LIDAR và vị trí của robot để tạo ra bản đồ 2D (gần giống với bản thiết kế mặt cắt của sàn nhà) hoặc 3D theo thời gian thực (Hình 2.21).



Hình 2.21: Mô hình Cartographer trong ROS2

2.4. Hệ thống Navigation2 (Nav2)

Nav2 là sự kế thừa và phát triển dựa trên ROS Navigation Stack. Hệ thống này nhằm mục đích tìm kiếm một đường đi an toàn để di chuyển từ điểm A đến điểm B. Nó cũng có thể được áp dụng trong các ứng dụng tương tự khác liên quan đến điều hướng robot như theo dõi mục tiêu di chuyển.

Nav2 sử dụng các thông tin, dữ liệu từ Odometry, cảm biến (sensor), điểm mục tiêu (goal pose) và xuất ra tín hiệu vận tốc gửi xuống robot. Việc sử dụng Nav2 trên một robot bất kỳ thì khá là phức tạp. Điều kiện tiên quyết để sử dụng Nav2 là robot phải được chạy trên ROS2, phải có một Transform Tree, và truyền dữ liệu của cảm biến theo đúng kiểu Message trong ROS2. Thứ hai, Nav2 cần được cấu hình về hình dạng và đặc tính của robot để có thể hoạt động ở một cấp độ cao hơn.

Mặc dù Nav2 được thiết kế sao cho dùng trong mục đích chung nhất có thể, nhưng vẫn có một số yêu cầu bắt buộc về phần cứng như sau:

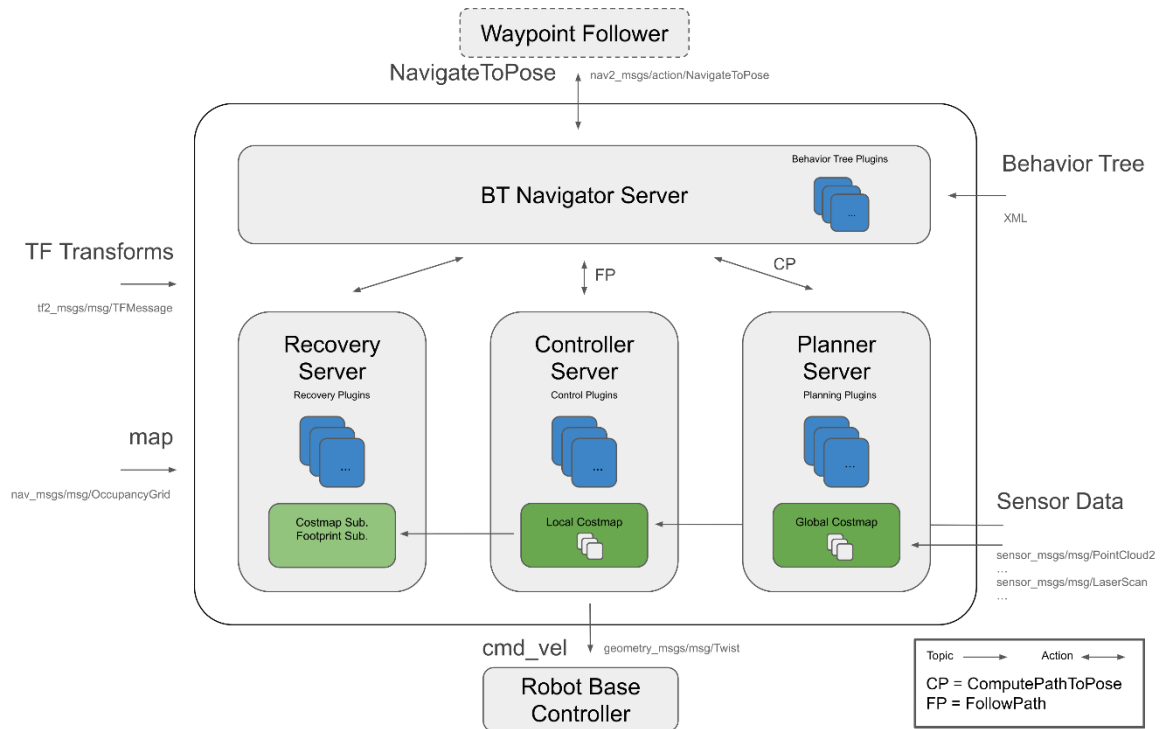
- Được thiết kế dành cho cả mô hình robot lái bằng hiệu tốc độ (differential drive) và mô hình robot có bánh xe đa hướng (holonomic wheeled robots). Chỉ việc gửi trực tiếp giá trị vận tốc xuống cho bộ phận điều khiển di chuyển của robot để đạt được các giá trị của robot mong muốn như vận tốc theo trục x, vận tốc theo trục y và vận tốc xoay.
- Nav2 cần phải sử dụng dữ liệu từ một cảm biến laser scan được gắn lên robot. Laser scan này dùng để xây dựng bản đồ (mapping), định vị (localization) và phát hiện vật cản.
- Nó được phát triển trên những robot có hình vuông vì thế để được hiệu năng tốt nhất thì hình dạng robot nên là hình vuông hoặc hình tròn. Nav2 cũng làm việc được trên robot có hình dạng và kích thước bất kỳ nhưng sẽ khó khăn với những robot có kích thước lớn khi làm việc trong những không gian hẹp.

2.4.1. Tổng quan về Nav2

Nav2 có nhiều công cụ khác nhau để thực hiện nhiều mục đích riêng của từng loại robot, dưới đây là những module thường được sử dụng trong hệ thống Nav2 (Hình 2.22):

- Map server: tải, phân phát và lưu trữ bản đồ.
- AMCL: bản địa hóa robot trên bản đồ.
- Nav2 Planner: lập kế hoạch để đi từ điểm A đến điểm B trên bản đồ và tránh các chướng ngại vật.
- Nav2 Controller: điều khiển robot đi theo đường dẫn đã được lập ra.
- Nav2 Costmap 2D: chuyển đổi giá trị trả về của robot và chuyển thành biểu đồ chi phí của bản đồ.
- Nav2 Behavior Trees and BT Navigator: xây dựng các hành vi phức tạp của robot bằng cách sử dụng cây hành vi.

- Nav2 Recoveries: trong trường hợp thất bại thì tính toán các hành vi và khôi phục nó.
- Nav2 Waypoint Follower: thực hiện theo các điểm tham chiếu tuần tự.
- Nav2 Lifecycle Manager: quản lý vòng đời và watchdog cho server.
- Nav2 Core: tính toán và thêm các thuật toán di chuyển của riêng mình.



Hình 2.22: Mô hình Nav2 trong ROS [7]

2.4.2. Module Costmap 2D

Module Costmap 2D là một gói để triển khai bản đồ chi phí (costmap) từ việc lấy dữ liệu của cảm biến từ môi trường, xây dựng một lưới các nơi bị chiếm chỗ và tăng vùng chi phí trong bản đồ chi phí 2D dựa trên lưới các nơi bị chiếm chỗ và bán kính tăng chi phí do người dùng định nghĩa.

Costmap 2D dùng để chứa thông tin các vật cản cũng như những nơi mà robot không thể tới được. Costmap sử dụng dữ liệu của các cảm biến từ bản đồ đã được xây dựng trước đó và cập nhật thông tin của vật cản vào costmap.

Các phương pháp cơ bản sử dụng để ghi dữ liệu lên costmap được cấu hình đầy đủ, mỗi bit chức năng nằm trên một lớp khác nhau. Ví dụ, bản đồ tĩnh (bản đồ đã được dựng trước đó) được nằm trên một lớp, và các vật cản được nằm trên một lớp khác. Lớp chứa vật cản lưu trữ thông tin các vật thể theo ba hướng nhằm cho vấn đề xử lý các vật cản trong bản đồ thông minh hơn.

2.5. Phần cứng

2.5.1. Nvidia Jetson TX2 Developer Kit

Nvidia Jetson TX2 Developer Kit (Hình 2.23) là một máy tính nhỏ nhưng rất mạnh mẽ cho phép chạy song song nhiều mạng neural sử dụng cho các ứng dụng như phân loại hình ảnh, phát hiện đối tượng, phân đoạn và xử lý giọng nói. Tất cả trong cùng một nền tảng dễ sử dụng cũng như tiêu tốn ít hơn 5 watts.

Nvidia Jetson TX2 Developer Kit cung cấp 1.33 TFLOPS để chạy các thuật toán AI hiện đại một cách nhanh chóng, với CPU Dual-Core NVIDIA Denver 2 64-Bit CPU và Quad-Core Arm® Cortex®-A57 MPCore processor, bên cạnh còn có GPU 256-core NVIDIA được tích hợp trên board mạch, cũng như bộ nhớ 4GB LPDDR4. Có thể chạy song song nhiều mạng neural và xử lý đồng thời một số cảm biến có độ phân giải cao.

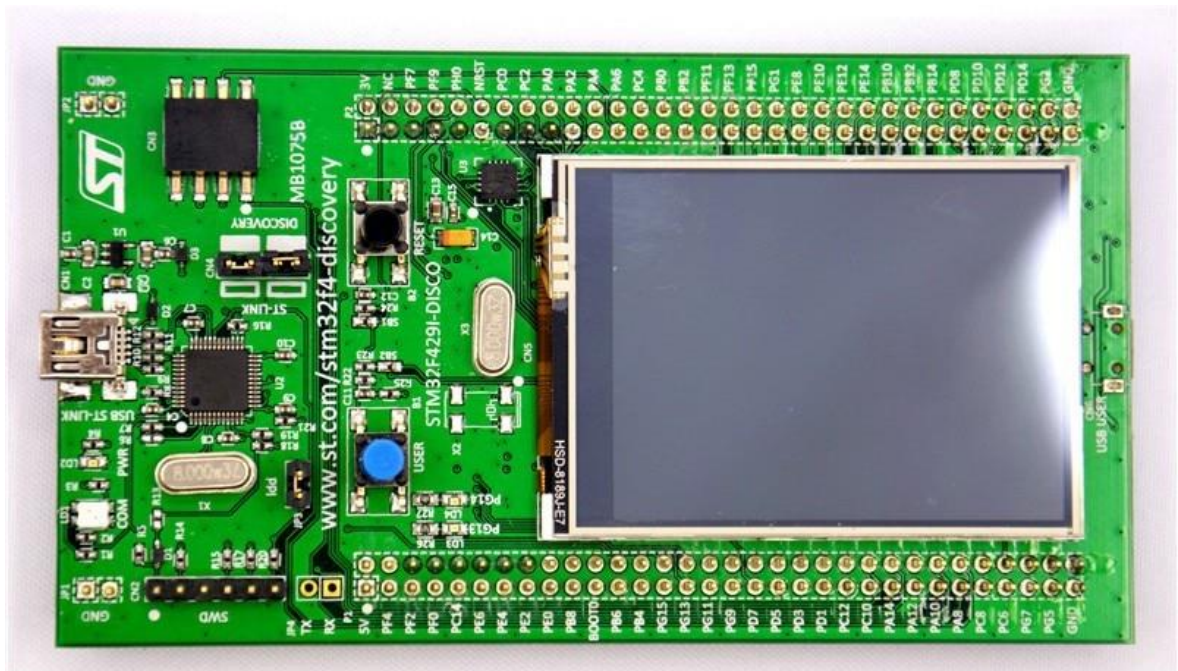
Nvidia Jetson TX2 Developer Kit cũng được hỗ trợ bởi NVIDIA JetPack. Bao gồm các gói hỗ trợ board (BSP), CUDA, cuDNN và thư viện phần mềm TensorRT cho deep learning, computer vision, GPU computing, multimedia processing và nhiều ứng dụng khác. SDK cũng bao gồm khả năng cài đặt frameworks Machine Learning (ML) mã nguồn mở như TensorFlow, PyTorch, Caffe / Caffe2, Keras và MXNet, cho phép các nhà phát triển tích hợp các model AI/ framework yêu thích của họ vào các sản phẩm một cách nhanh chóng và dễ dàng.



Hình 2.23: Nvidia Jetson TX2 carrier board

2.5.2. STM32F4 Discovery Kit

STM32F429 Discovery Kit (Hình 2.24) dựa trên lõi RISC 32-bit Arm® Cortex®-M4 32-bit hiệu suất cao hoạt động ở tần số lên đến 180 MHz, lõi Cortex-M4 có độ chính xác đơn đơn vị dấu chấm động (FPU) hỗ trợ tất cả các kiểu dữ liệu và hướng dẫn xử lý dữ liệu độ chính xác đơn của Arm®. Nó cũng thực hiện một tập hợp đầy đủ các hướng dẫn DSP và một đơn vị bảo vệ bộ nhớ (MPU) giúp tăng cường bảo mật cho ứng dụng. Các thiết bị STM32F427xx và STM32F429xx tích hợp bộ nhớ nhúng tốc độ cao (bộ nhớ Flash lên đến 2 Mbyte, lên đến 256 Kbyte SRAM), lên đến 4 Kbyte SRAM dự phòng và một loạt các I/Os nâng cao và thiết bị ngoại vi được kết nối với hai APB bus (Advanced Peripheral Bus), hai bus AHB (Advanced High-Performance bus) và ma trận bus AHB 32 bit.



Hình 2.24: STM32F429 Discovery Kit

2.5.3. Cảm biến RPLidar A1

Rplidar A1 là module quét laser 2D 360 độ với giá thành rẻ, được công ty SLAMTEC sản xuất. Module có thể quét laser 360 độ với khoảng cách lên đến 6 m, cung cấp dữ liệu điểm 2D có thể được sử dụng để làm mapping, localization (SLAM) và dựng mô hình của môi trường/vật thể.

Item	Unit	Min	Typical	Max	Comments
Distance Range	Meter(m)	TBD	0.15 - 6	TBD	White objects
Angular Range	Degree	n/a	0-360	n/a	
Distance Resolution	mm	n/a	<0.5 <1% of the distance	n/a	<1.5 meters All distance range*
Angular Resolution	Degree	n/a	≤1	n/a	5.5Hz scan rate
Sample Duration	Millisecond(ms)	n/a	0.5	n/a	
Sample Frequency	Hz	n/a	≥2000	2010	
Scan Rate	Hz	1	5.5	10	Typical value is measured when RPLIDAR A1 takes 360 samples per scan

Hình 2.25: Hình ảnh mô tả các thông số của Rplidar A1

Nguyên lý hoạt động:

- Bằng cách phát đi một chùm tia laser rồi thu nhận lại tín hiệu phản hồi. Tốc độ ánh sáng đã biết trước, độ trễ phản hồi được ghi nhận, từ đó tính được khoảng cách giữa bộ cảm biến và vật thể một cách tương đối chính xác. Sự chênh lệch về thời gian và bước sóng laser sau đó được sử dụng để mô hình số 2 chiều của đối tượng, Khoảng cách (d) tính bằng $\frac{1}{2}$ tích giá trị tốc độ ánh sáng (c) và thời gian (t):

$$d = \frac{1}{2} t \times c$$

Trong đó: d là khoảng cách từ cảm bộ cảm biến đến vật

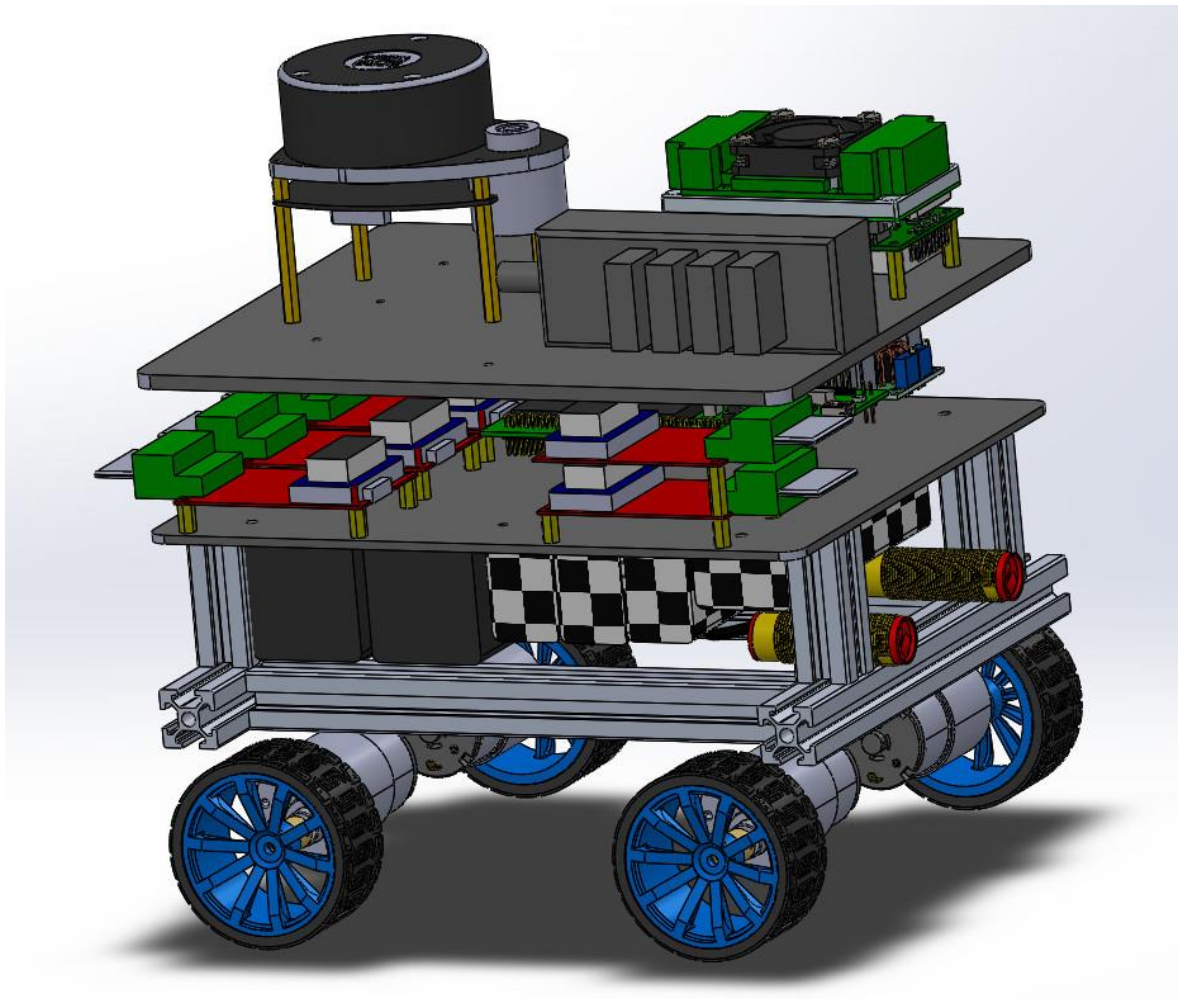
t là thời gian từ lúc phát tín hiệu đi đến khi nhận tín hiệu trả về

c là tốc độ ánh sáng (299.792.458 m/s)

2.5.4. Khung robot

Robot được thiết kế với 3 tầng khác nhau đặt trên bánh xe là khung nhôm định hình, tầng này chứa pin để xe hoạt động, tầng tiếp theo là mạch driver và STM32F429

Discovery Kit giúp điều khiển các trạng thái của robot và tầng cuối cùng là LIDAR cùng với Nvidia Jetson TX2 Developer Kit thực hiện quét bản đồ ra tín hiệu cho xe di chuyển. Khung robot có kích thước $240 \times 200 \times 245$ mm, sử dụng 4 động cơ DC Servo JGB37 DC Geared Motor được gắn với 4 bánh xe có đường kính 65 mm (Hình 2.26).

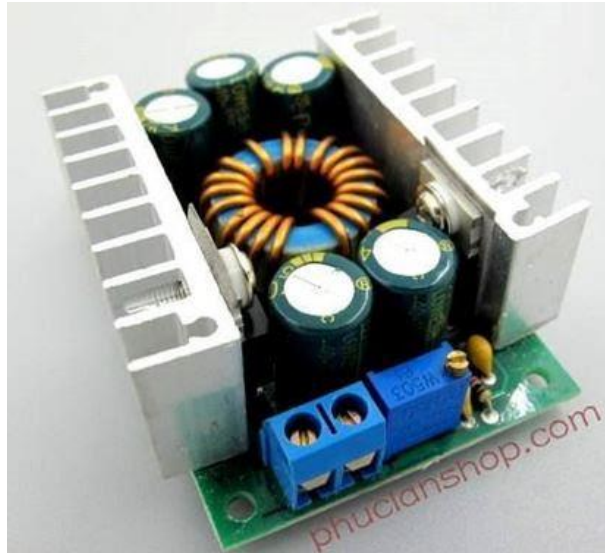


Hình 2.26: Khung xe vẽ trên phần mềm solidworks

2.5.5. Một số module khác

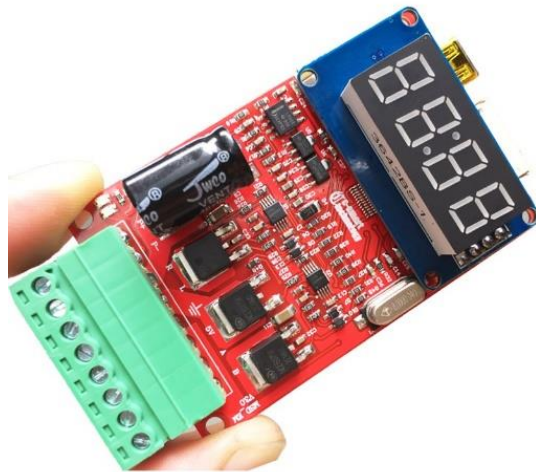
Mạch giảm áp DC (Hình 2.27) hiệu suất cao 95% sử dụng ở dải điện áp 4.5-30 V, dòng đầu ra lên tới 12 A, thích hợp cho các ứng dụng giảm áp cần công suất lớn.

Mạch được thiết kế dễ dàng sử dụng với một đầu cấp điện áp và một ngõ ra được điều chỉnh điện áp ra bằng biến trở tinh chỉnh được gắn trực tiếp trên mạch. Mạch được gắn thêm 2 miếng nhôm tản nhiệt cỡ lớn giúp quá trình tản nhiệt tốt hơn giúp công suất của mạch được cao hơn.



Hình 2.27: Mạch nguồn giảm áp DC-DC

Module Servo Driver MSD_E10A (Hình 2.28) là một mạch có nhiều tính năng như điều khiển bằng xung/chiều, giao tiếp qua UART hoặc COM ảo. Bảo vệ quá tải quá dòng quá nhiệt phát hiện encoder hư, động cơ hư. Có phần mềm hỗ trợ, tích hợp với nhiều phương pháp điều khiển trực quan khác nhau với các thông số PID có thể hiệu chỉnh trực tiếp trên phần mềm. Mạch này có công suất tối đa là 200 W, 10-28 VDC.



Hình 2.28: Servo driver MSD_E10A

Cảm biến gia tốc GY-521 6DOF IMU MPU6050 (Hình 2.29) được sử dụng để đo 6 thông số: 3 trục góc quay (Gyro), 3 trục gia tốc hướng (Accelerometer), là loại cảm biến gia tốc phổ biến nhất trên thị trường hiện nay.



Hình 2.29: Cảm biến gia tốc GY-521 6DOF IMU MPU6050

Động Cơ DC Servo JGB37 DC Geared Motor có cấu tạo bằng kim loại cho độ bền và độ ổn định cao, được sử dụng trong các mô hình robot, xe, thuyền,..., hộp giảm tốc của động cơ có nhiều tỉ số truyền giúp dễ dàng lựa chọn giữa lực kéo và tốc độ (lực kéo càng lớn thì tốc độ càng chậm và ngược lại), động cơ sử dụng nguyên liệu chất lượng cao (lõi dây đồng nguyên chất, lá thép 407, nam châm từ tính mạnh...) cho sức mạnh và độ bền vượt trội hơn các loại giá rẻ trên thị trường hiện nay (sử dụng lõi dây nhôm, nam châm từ tính yếu).



Hình 2.30: Động Cơ DC Servo JGB37 DC

Cảm biến hồng ngoại E3F-DS30Y2 dùng ánh sáng hồng ngoại để giám sát vật cần mong muốn: Khi vật cần giám sát trong vùng làm việc, cảm biến sẽ kích hoạt ngõ ra lên mức ON báo hiệu có sự xuất hiện của vật. Khi vật cần giám sát ở bên ngoài vùng làm việc, cảm biến cho ngõ ra ở mức OFF. Cảm biến cho độ phản hồi nhanh và rất ít nhiễu do sử dụng mắt nhận và phát tia hồng ngoại theo tần số riêng biệt. Khoảng cách làm việc của cảm biến là 30 cm, có thể chỉnh khoảng cách làm việc thông qua biến trở.



Hình 2.31: Cảm biến hồng ngoại E3F-DS30Y2

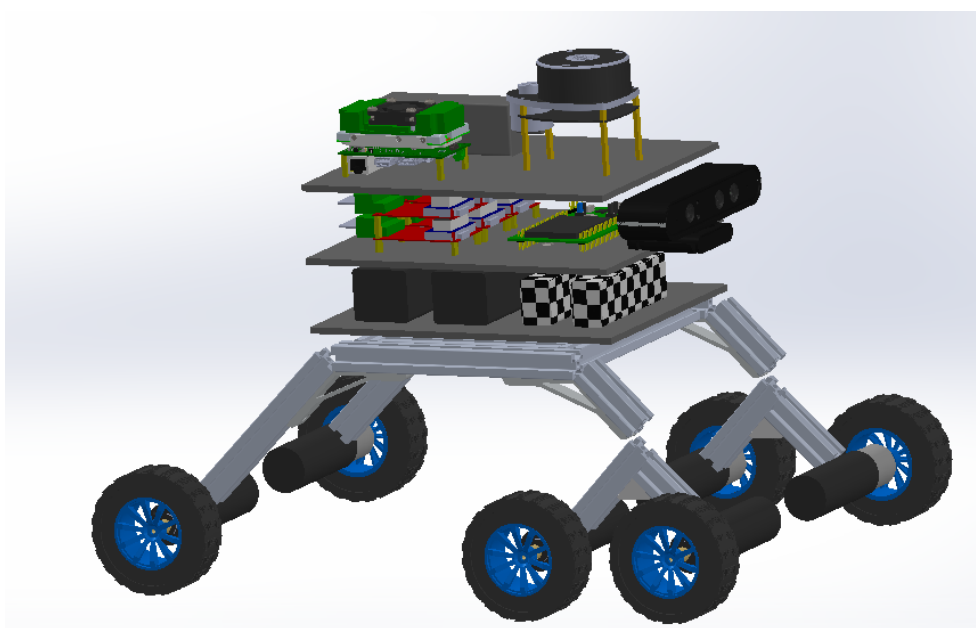
Chương 3. PHÂN TÍCH VÀ THIẾT KẾ HỆ THỐNG

3.1. Phân tích phần cứng

3.1.1. Phân tích khung xe

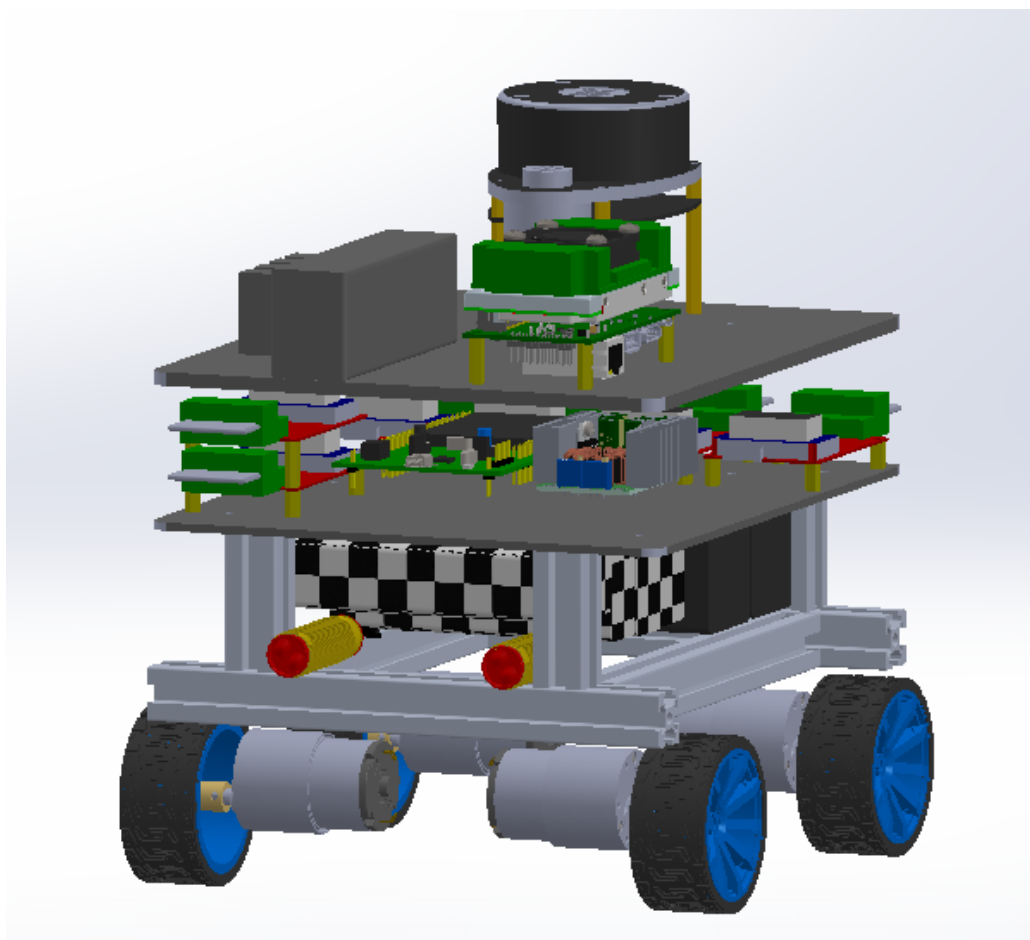
Để giúp xe hoạt động tốt và chắc chắn thì cần một khung xe chắc chắn, thiết kế gọn và vị trí đặt LIDAR phải nằm vị trí thuận lợi nhất để có thể quét được nhiều vật cản nhất có thể. Qua quá trình thử nghiệm và hiệu chỉnh, khung xe đã được thay đổi từ mô hình robot Mars Rover (Hình 3.1) thành mô hình như hiện nay (Hình 3.2).

Với khung xe được sử dụng dựa theo mô hình robot Mars Rover nhằm hướng tới khả năng di chuyển được trên nhiều địa hình địa hình phức tạp khác nhau nhưng gặp vấn đề về cơ khí, các khớp nối động còn lỏng lẻo, độ chính xác về cơ khí không được như ý muốn ban đầu nên đã ảnh hưởng đến khả năng di chuyển của robot, đặc biệt là khung xe quá cao nên LIDAR nằm quá cao so với mặt đất nên khi quét bản đồ không thể quét được các vật thấp hơn LIDAR.



Hình 3.1: Khung xe dựa theo mô hình Mars rovers

Để phát hiện được các vật cản nằm tầm thấp và giúp xe di chuyển linh hoạt và chắc chắn hơn khung xe được thiết kế lại theo như Hình 3.2.



Hình 3.2: Khung xe hiện tại

3.1.2. Liên kết các thành phần xe

Vì robot sử dụng cùng lúc 4 động cơ và 4 driver để hoạt động được tốt nhất nên robot được trang bị cho mỗi driver 1 cục pin 2200 mAh. Để cấp nguồn hoạt động của STM32F429 Discovery Kit và Nvidia Jetson TX2 Developer Kit nhằm cho xe thời gian sử dụng tốt nhất có thể thì cần 2 cục pin 7200 mAh. Nguồn cấp cho STM32F429 Discovery Kit được đưa qua mạch hạ áp DC-DC trước khi cấp vào mạch và tới các bộ phận khác như cảm biến hồng ngoại, MPU, Servo driver MSD_E10A.

Từ mạch STM32F429 Discovery Kit sẽ nối các dây tín hiệu điều khiển các board Servo driver MSD_E10A từ đó điều khiển động cơ theo ý muốn, bên cạnh đó

STM32F429 Discovery Kit còn nhận thông tin từ các cảm biến hồng ngoại để xử lý các vật cản ở gần đọc tín hiệu của MPU để tính toán vị trí thêm chính xác và gửi về cho Nvidia Jetson TX2 Developer Kit thực hiện thuật toán SLAM và Nav2 để vẽ bản đồ và tính toán tự hành trên bản đồ.

3.2. Phân tích phần mềm

Phần mềm trên hệ thống gồm ba thành phần: Phần mềm trên máy tính cá nhân nhằm khởi động kết nối với hệ thống bên dưới, xuất thông tin ra màn hình để giao tiếp với robot trực quan hơn. Phần mềm trên Nvidia Jetson TX2 Developer Kit tính toán và xử lý các thuật toán phức tạp. Phần RTOS trên STM32F429 Discovery Kit giao tiếp trực tiếp các thành phần còn lại của hệ thống tính toán tham số điều khiển phần cứng và giao tiếp với các cảm biến.

3.2.1. Phần mềm trên máy tính cá nhân

Phần mềm trên máy tính cá nhân sẽ là hệ thống ROS2 với các package được tạo lại để dễ dàng giao tiếp với ROS2 trên Nvidia Jetson TX2 Developer Kit và xuất ra màn hình cho người dùng có cái nhìn trực quan hơn. Công cụ được sử dụng chủ yếu ở phần này chính là Rviz2.

3.2.2. Phần ROS2 trên Nvidia Jetson TX2 Developer Kit

Phần này sẽ là hệ thống ROS2 server của hệ thống, lấy các thông tin dữ liệu từ các cảm biến trả về sử dụng cho các thuật toán SLAM để vẽ bản đồ, lưu bản đồ lại sau đó sử dụng Nav2 cùng với bản đồ đã được vẽ trước đó cùng với giá trị trả về từ LIDAR cũng như dữ liệu về vị trí xe để có thể di chuyển theo ý muốn bên cạnh đó còn đảm nhận phần Micro ROS để giao tiếp được với hệ thống của STM32F429 Discovery Kit. Bảng 3.1 thể hiện các package cần để cho robot hoạt động. Các package được kế thừa lại để thực hiện hệ thống được thể hiện như trong Bảng 3.2.

Bảng 3.1: Các package để sử dụng robot

STT	Package	Chức năng
1	My_robot_bringup	Khởi tạo những thiết đặt ban đầu cần thiết cho robot
2	My_robot_description	Mô tả các thành phần của robot cũng như vị trí cụ thể của những thành phần đó so với trọng tâm của robot
3	my_robot_slam	Sử dụng các dữ liệu từ LIDAR và thông tin về vị trí của robot để vẽ và lưu bản đồ của khu vực hoạt động
4	my_robot_navigation	Giúp robot di chuyển và tránh né vật cản xuất hiện trên bản đồ
5	my_robot_teleop	Package dùng để truyền các hiệu điều khiển xe di chuyển phục vụ cho quá trình vẽ bản đồ

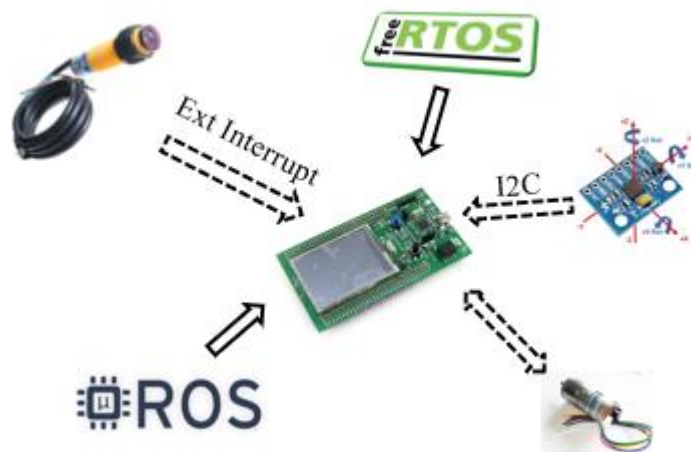
Bảng 3.2: Các package được kế thừa lại để thực hiện hệ thống

STT	Package	Chức năng
1	micro_ros_agent	Truyền dữ liệu từ Nvidia Jetson TX2 Developer Kit đến STM32F429 Discovery Kit bằng UART
2	rplidar_ros	Giao tiếp với Rplidar_A1 bằng UART trên ROS2
3	robot_state_publisher	Nhận dữ mô tả URDF để publish các trạng thái của robot để hiển thị lên Rviz2
4	cartographer_ros	Hỗ trợ các thuật toán vẽ và lưu bản đồ

5	navigation2	Gồm các package hỗ trợ các thuật toán cho quá trình navigation như DWB, ACML, costmap2D...
---	-------------	--------------------------------------------------------------------------------------------

3.2.3. Phần RTOS trên STM32F429 Discovery Kit

Phần này sẽ sử dụng hệ điều hành RTOS tích hợp Micro-ROS để dễ dàng giao tiếp với hệ thống ROS2 trên Nvidia Jetson TX2 Developer Kit thông qua DDS đồng thời chia thời gian quản lý các tác vụ điều khiển bốn driver cho xe di chuyển theo ý muốn. Song song đó RTOS cũng tính toán giá trị của encoder quãng đường đi được, cảm biến hồng ngoại nhận diện vật cản và xử lý dữ liệu thô của cảm biến IMU MPU6050. Sơ đồ khối mô tả hệ thống được thể hiện như Hình 3.3.



Hình 3.3: Sơ đồ khối mô tả hệ thống trên STM32F429 Discovery Kit

Chức năng hoạt động của robot sẽ được chia nhỏ ra thành nhiều tác vụ hoạt động song song với nhau nhằm đảm bảo đáp ứng thời gian hoạt động (Bảng 3.3). Tùy vào mức độ yêu cầu đáp ứng thời gian mà mỗi tác vụ sẽ có những độ ưu tiên và chu kỳ thực thi khác nhau.

Bảng 3.3: Các tác vụ trong RTOS

Tác vụ	Chức năng	Chu kì (ms)	Độ ưu tiên
MicroRosTask	Thực hiện giao tiếp với ROS2 trên Nvidia Jetson TX2 Developer Kit, truyền nhận những dữ liệu cần thiết về vị trí của robot cũng như tín hiệu điều khiển để robot di chuyển	1	osPriorityAboveNormal
MotorControlTask	Điều khiển động cơ	10	osPriorityBelowNormal
ObstacleTask	Đọc dữ liệu từ cảm biến phát hiện vật cản để đưa ra quyết định về trạng thái di chuyển của xe	10	osPriorityRealtime
IMUTask	Đọc dữ liệu về góc quay của robot phục vụ cho tính toán Odometry	5	osPriorityNormal

Ngoài những tác vụ trên, trong hệ thống còn có một phần rất quan trọng chính là chương trình ngắt ngoài trên STM32F429 Discovery Kit. Đây là phần xử lý có mức độ ưu tiên cao nhất với chức năng khi có vật cản xuất hiện, hàm ngắt được thực thi

và sẽ dừng động cơ ngay lập tức, tránh trường hợp robot di chuyển va chạm với vật cản. Sau đó thì tác vụ ObstacleTask sẽ tiếp tục thực thi để đưa ra những quyết định mới về trạng thái di chuyển của xe.

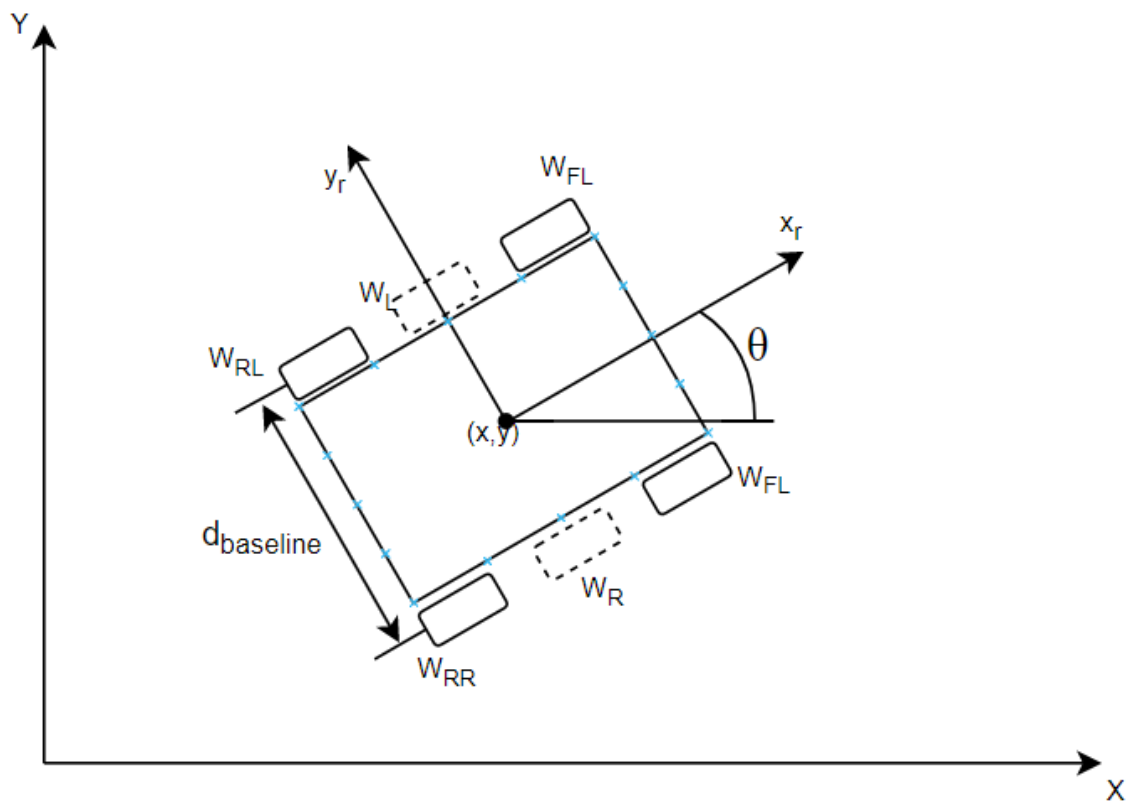
3.3. Tính toán Odometry, thuật toán Dijkstra và Dynamic Window Approach

3.3.1. Tính toán Odometry

Hầu hết các vấn đề về robot đều dẫn tới một câu hỏi: Robot đang ở đâu? Robot có thể đi đến nơi khác bằng cách nào khi mà không biết vị trí hiện tại của nó? Một số nghiên cứu về robot hi vọng rằng các mục tiêu, vị trí có thể được xác định bằng các dữ liệu của các cảm biến khi mục tiêu nằm trong phạm vi của các cảm biến đó, nhưng một cách tốt hơn để làm việc đó chính là navigation. Một phương pháp điều hướng cơ bản, được hầu hết tất cả robot hiện nay sử dụng là Odometry, sử dụng kiến thức liên quan đến chuyển động của bánh xe để ước tính chuyển động của xe.

Mục tiêu của phần này là để chỉ ra cách sử dụng dữ liệu để tính toán Odometry. Một tập dữ liệu thực sẽ được sử dụng rộng rãi để cho thấy phương pháp này có thể hoạt động ra sao. Giả sử robot bắt đầu từ điểm gốc. Trạng thái của nó là $(x, y, \theta) = (0, 0, 0)$. Nếu robot di chuyển thẳng trong ba giây với vận tốc 1 m/s, thì một dự đoán chính xác là trạng thái của robot sẽ là $(3, 0, 0)$.

Chúng ta sẽ giả định rằng chiếc xe được dẫn động bằng vi sai: nó có một động cơ ở bên trái và một động cơ khác ở phía bên phải. Nếu cả hai động cơ quay về phía trước, robot đi thẳng. Nếu động cơ bên phải quay nhanh hơn động cơ bên trái, robot sẽ di chuyển sang trái. Mục tiêu của chúng ta là đo tốc độ quay của động cơ bên trái và bên phải. Từ điều này, chúng ta có thể đo vận tốc và tốc độ rẽ của chúng, sau đó tích hợp các dữ liệu này để thu được vị trí. Vận tốc động cơ cho chúng ta hai đại lượng: độ dịch chuyển khi xoay và độ dịch chuyển khi đi thẳng., từ đó chúng ta có vị trí hiện tại của robot (x, y, θ) . Như vậy chúng ta có tốc độ góc (angular velocity) được lấy dữ liệu và tính toán từ cảm biến MPU6050 và tốc độ tuyến tính (linear velocity) được lấy dữ liệu từ encoder của động cơ.



Hình 3.4: Sơ đồ bố trí tọa độ của robot

x – trọng tâm robot trên trục X

y – Trọng tâm robot trên trục Y

x_r – trục x cục bộ robot xác định phía trước của robot

y_r – trục y cục bộ của robot

θ – vị trí góc của robot

W_{FL} – bánh trước bên trái

W_{FR} – bánh trước bên phải

W_{RL} – bánh sau bên trái

W_{RR} – bánh sau bên phải

W_L – bánh xe trái ảo

W_R – bánh xe phải ảo

$d_{baseline}$ – khoảng cách giữa bánh trái và phải robot

Vị trí của robot được xác định bởi một bộ (x, y, θ) . Nhiệm vụ động học phía trước là tìm vị trí robot mới $(x, y, \theta)'$ sau thời gian Δt đối với các thông số điều khiển đã cho.

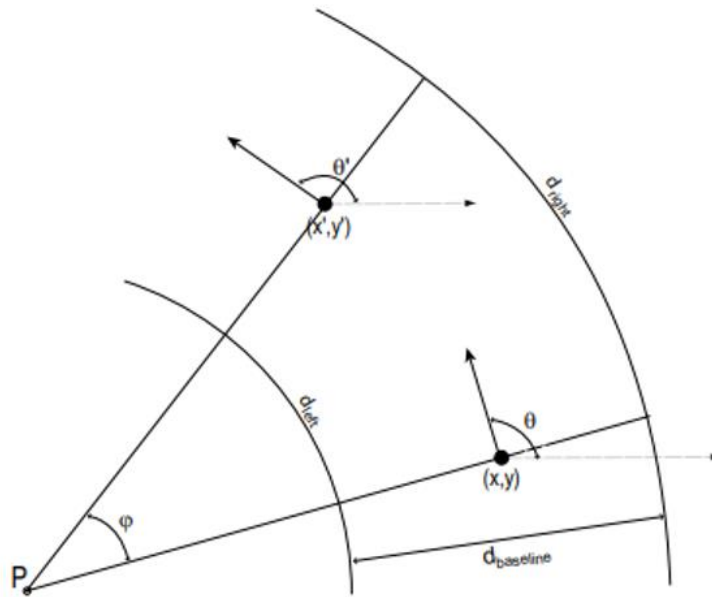
Trong trường hợp của chúng ta, vị trí và góc của mỗi bánh xe ảo sẽ là giá trị trung bình của các bánh xe thực của nó:

$$W_L = \frac{W_{FL} + W_{RL}}{2} \quad (1)$$

$$W_R = \frac{W_{FR} + W_{RR}}{2} \quad (2)$$

$$\theta W_L = \frac{\theta W_{FL} + \theta W_{RL}}{2} \quad (3)$$

$$\begin{aligned} \theta W_R \\ = \frac{\theta W_{FR} + \theta W_{RR}}{2} \end{aligned} \quad (4)$$



Hình 3.5: Qua một khoảng thời gian rất nhỏ, chuyển động robot có thể gần bằng một cung tròn [4].

Vị trí của robot (x, y, θ) là vị trí gốc của robot, với θ là hướng của robot. Sau khi cả 2 bánh xe đã đi được một khoảng d_{left} và d_{right} , chúng ta sẽ tính ra vị trí mới (x', y', θ') . Trung tâm của robot (điểm chính giữa đoạn thẳng nối hai bánh xe) cũng di

chuyển một vòng cung theo hai bánh xe. Độ dài cung tròn được dựa trên 2 đại lượng d_{left} và d_{right} .

$$D_{\text{center}} = \frac{d_{\text{left}} + d_{\text{right}}}{2} \quad (5)$$

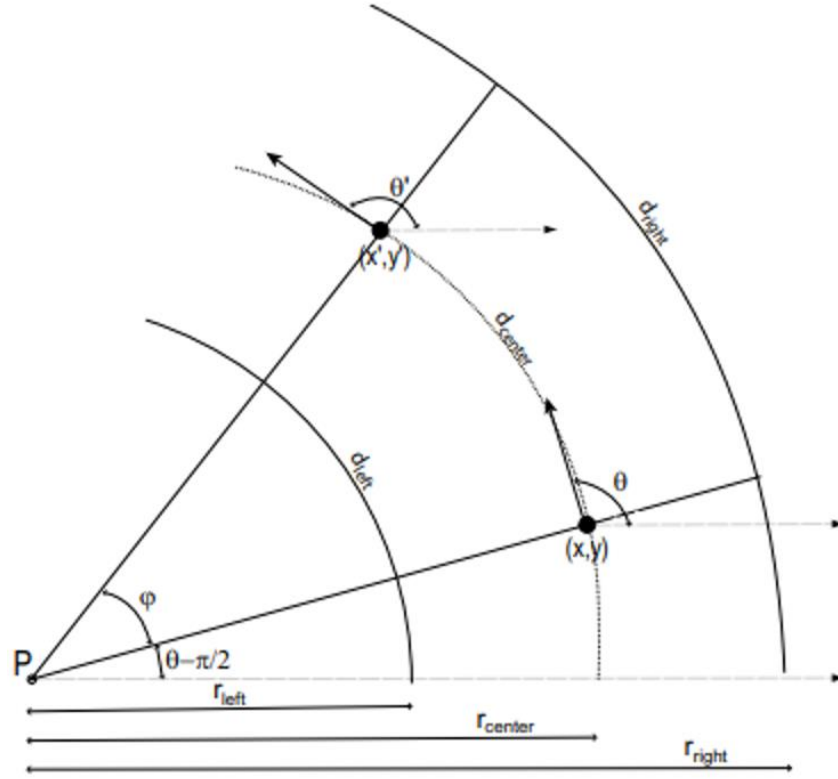
Từ đó ta có:

$$\varphi r_{\text{left}} = d_{\text{left}} \quad (6)$$

$$\varphi r_{\text{right}} = d_{\text{right}} \quad (7)$$

Nếu d_{baseline} là khoảng cách giữa hai bánh ảo ta lấy:

$$r_{\text{left}} + d_{\text{baseline}} = r_{\text{right}} \quad (8)$$



Hình 3.6: Hai đường cung tròn tạo bởi hai bánh xe ảo [4].

Lấy (7) trừ (6),

$$\varphi r_{\text{right}} - \varphi r_{\text{left}} = d_{\text{right}} - d_{\text{left}}$$

$$\varphi(r_{\text{right}} - r_{\text{left}}) = d_{\text{right}} - d_{\text{left}}$$

$$\varphi d_{\text{baseline}} = d_{\text{right}} - d_{\text{left}}$$

$$\varphi = \frac{d_{\text{right}} - d_{\text{left}}}{d_{\text{baseline}}} \quad (9)$$

Để phép tính vị trí trở nên đơn giản hơn, tất cả các cung tròn đều có chung một tâm tròn P. Góc xoay của robot dựa trên trục x với công thức $\theta - \pi/2$. Chúng ta có thể tính ra tọa độ của P:

$$\begin{aligned} P_x &= x - r_{\text{center}} \cos(\theta - \pi/2) \\ &= x - r_{\text{center}} \sin(\theta) \end{aligned} \quad (10)$$

$$\begin{aligned} P_y &= y - r_{\text{center}} \sin(\theta - \pi/2) \\ &= y + r_{\text{center}} \cos(\theta) \end{aligned} \quad (11)$$

Chúng ta tính ra được x' và y' :

$$\begin{aligned} x' &= P_x + r_{\text{center}} \cos(\varphi + \theta - \pi/2) \\ &= x - r_{\text{center}} \sin(\theta) + r_{\text{center}} * \sin(\varphi + \theta) \\ &= x + r_{\text{center}} [-\sin(\theta) + \sin(\varphi)\cos(\theta) + \sin(\theta)\cos(\varphi)] \end{aligned} \quad (12)$$

$$\begin{aligned} y' &= P_y + r_{\text{center}} \sin(\varphi + \theta - \pi/2) \\ &= y + r_{\text{center}} \cos(\theta) - r_{\text{center}} * \cos(\varphi + \theta) \\ &= y + r_{\text{center}} [\cos(\theta) - \cos(\varphi)\cos(\theta) + \sin(\theta)\sin(\varphi)] \end{aligned} \quad (13)$$

Nếu φ nhỏ (thông thường khoảng thời gian nhỏ), chúng ta ước lượng $\sin(\varphi) = \varphi$ và $\cos(\varphi) = 1$. Vậy ta có:

$$\begin{aligned} x' &= x + r_{\text{center}} [-\sin(\theta) + \varphi \cos(\theta) + \sin(\theta)] \\ &= x + r_{\text{center}} \varphi \cos(\theta) \\ &= x + d_{\text{center}} \cos(\theta) \end{aligned} \quad (14)$$

$$\begin{aligned} y' &= y + r_{\text{center}} [\cos(\theta) - \cos(\theta) + \varphi \sin(\theta)] \\ &= y + r_{\text{center}} [\varphi \sin(\theta)] \\ &= y + d_{\text{center}} \sin(\theta) \end{aligned} \quad (15)$$

Vậy tóm lại, phương trình tính Odometry (x' , y' , θ') được đơn giản thành:

$$d_{center} = \frac{d_{left} + d_{right}}{2}$$

$$\varphi = \frac{d_{right} - d_{left}}{d_{baseline}}$$

$$\theta' = \theta + \varphi \quad (16)$$

$$x' = x + d_{center} \cos(\theta) \quad (17)$$

$$y' = y + d_{center} \sin(\theta) \quad (18)$$

3.3.2. Thuật toán Dynamic Window Approach

Thuật toán DWA được dùng để tìm ra một tín hiệu điều khiển hợp lý gửi xuống robot nhằm mục đích điều khiển nó đến đích an toàn, nhanh chóng dựa trên global planner đã hoạch định từ trước. Thuật toán này gồm hai bước chính là cắt giảm không gian tìm kiếm (search space) của vận tốc và tìm được vận tốc tối ưu trong không gian tìm kiếm đó.

Thuật toán DWB planner là bản cải thiện của thuật toán DWA planner dùng trên ROS2.

3.3.2.1. Không gian tìm kiếm

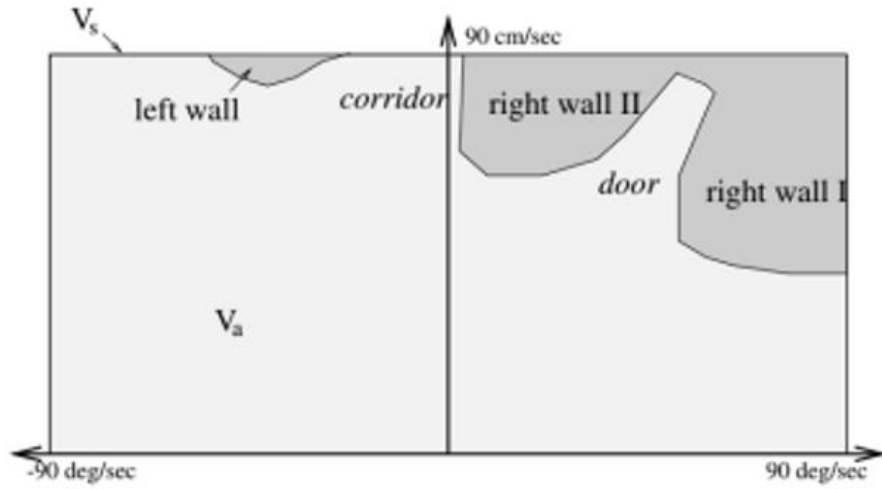
Các vận tốc có thể điều khiển được trong không gian tìm kiếm được cắt giảm theo ba bước sau:

- Quỹ đạo tròn: thuật toán DWA chỉ xét đến quỹ đạo là hình tròn (đường cong) được xác định duy nhất bởi một cặp vận tốc thẳng và vận tốc xoay (v, ω).
- Vận tốc cho phép: nhằm tạo ra một quỹ đạo an toàn cho robot để tránh vật cản. Một cặp vận tốc (v, ω) được cho phép là khi robot có thể dừng trước vật cản gần nhất mà không có sự va chạm trên đường cong tương ứng với vận tốc đó. Vận tốc cho phép được định nghĩa như sau:

$$V_a = \{(v, \omega) | v \leq \sqrt{2 \times dist(v, \omega) \times \dot{v}_b} \wedge \omega \leq \sqrt{2 \times dist(v, \omega) \times \dot{\omega}_b}\}$$

Trong đó:

- V_a là chuỗi các giá trị vận tốc (v, ω) cho phép robot dừng trước vật cản mà không có sự va chạm.
- $dist(v, \omega)$ là khoảng cách nhỏ nhất mà robot dừng trước vật cản để không có sự va chạm.
- $\dot{v}_b, \dot{\omega}_b$ là gia tốc của vận tốc thẳng và vận tốc xoay tối đa nếu robot di chuyển sẽ gây va chạm với vật cản.



Hình 3.7: Vận tốc cho phép V_a trong DWA [11]

Dynamic window: nhằm hạn chế vận tốc cho phép đối với những vận tốc có thể đạt được trong khoảng chu kỳ cho trước với gia tốc tối đa của robot. Để Δt là khoảng thời gian mà trong đó gia tốc $\dot{v}, \dot{\omega}$ sẽ được thực thi để (v_a, ω_a) là vận tốc thực được gửi xuống robot. Từ đó, vận tốc V_d sẽ được định nghĩa như sau:

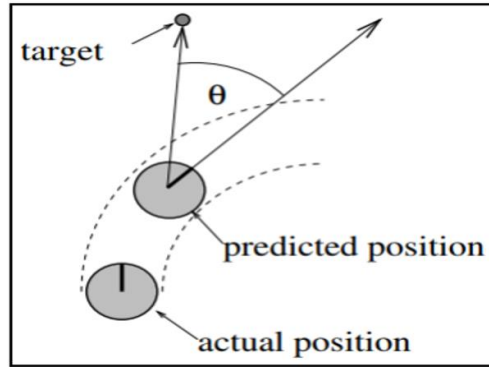
$$V_d = \{(v, \omega) | v \in [v_a - \dot{v} \cdot \Delta t, v_a + \dot{v} \cdot \Delta t] \wedge \omega \in [\omega_a - \dot{\omega} \cdot \Delta t, \omega_a + \dot{\omega} \cdot \Delta t]\}$$

3.3.2.2. Tối ưu

Ta có hàm mục tiêu được định nghĩa như sau:

$$G(v, \omega) = \alpha \cdot heading(v, \omega) + \beta \cdot dist(v, \omega) + \gamma \cdot vel(v, \omega)$$

Để có thể tối ưu hóa vận tốc ngõ ra thì hàm mục tiêu phải có giá trị tối đa. Để thực hiện được điều này, ta thực hiện các bước sau:



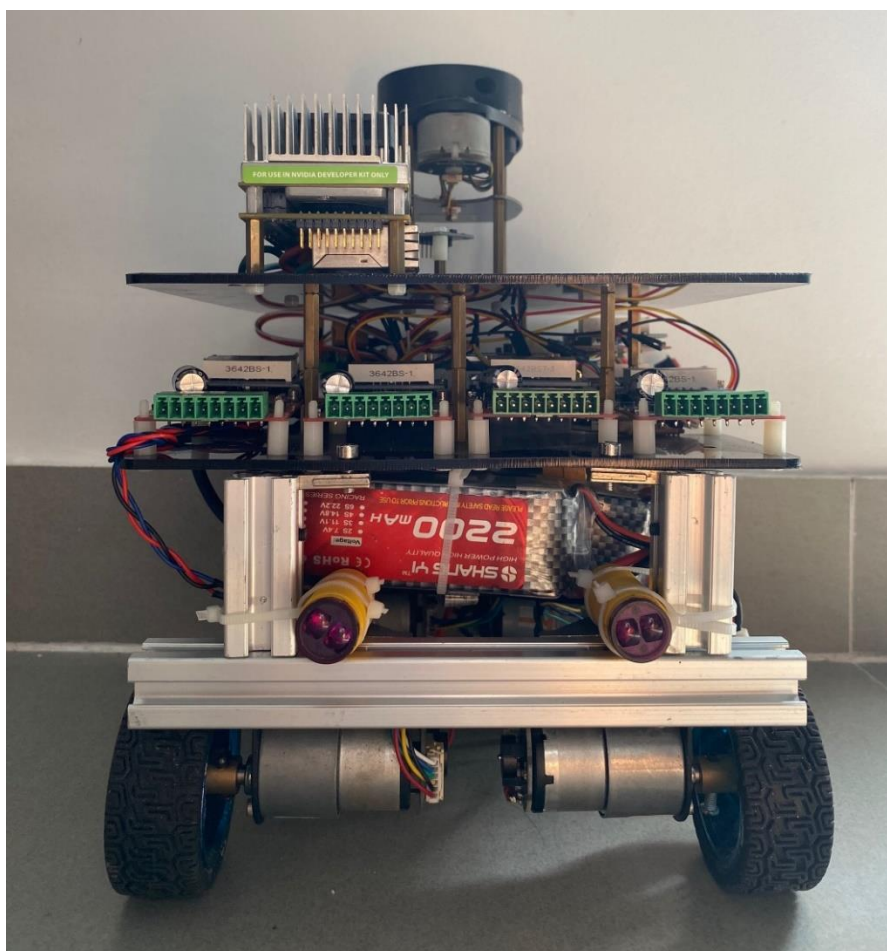
Hình 3.8: Heading của robot trong DWA [11]

- Target heading: heading là giá trị đo tiến độ hướng đến đích của robot. Giá trị sẽ mang giá trị tối đa khi robot di chuyển trực tiếp về phía đích. Giá trị của $\text{heading}(v, \omega)$ được tính bởi công thức $180 - \theta$, với θ là góc giữa hướng của robot và điểm đích.
- Không gian trống (clearance): hàm $\text{dist}(v, \omega)$ thể hiện khoảng cách tính từ robot đến vật cản gần nhất nằm trên quỹ đạo cong của nó. Giá trị này sẽ rất lớn nếu không có vật cản nằm trên quỹ đạo cong di chuyển của nó. Giá trị này càng nhỏ thì việc nó đối mặt với vật cản càng cao, khi đó nó sẽ di chuyển xung quanh vật cản ấy.
- Vận tốc: hàm $\text{vel}(v, \omega)$ là vận tốc di chuyển thẳng của robot và hỗ trợ di chuyển nhanh hơn.
- Các hệ số α, β, γ được chọn sao cho phù hợp với đặc tính của robot và môi trường hoạt động. Khi hàm mục tiêu có giá trị lớn nhất thì quỹ đạo tối ưu sẽ được chọn với vận tốc (v, ω) tốt nhất và là kết quả của thuật toán.

Chương 4. THỰC NGHIỆM VÀ ĐÁNH GIÁ

4.1. Phần khung robot

Khung robot được lắp ráp từ nhôm định hình có kích thước 240×200×245 mm, sử dụng 4 động cơ DC Servo JGB37 DC Geared Motor gắn với 4 bánh xe có đường kính 65 mm. Mặt trước của robot được gắn 2 cảm biến hồng ngoại giúp phát hiện vật cản, đảm bảo an toàn cho xe (Hình 4.1). Những cục pin dùng để cung cấp năng lượng cho xe hoạt động được đặt ở phần gầm của robot giúp hạ thấp trọng tâm của robot, hạn chế trường hợp robot bị lật trong trường hợp di chuyển với quán tính cao. Phía trên sẽ là nơi đặt các mạch điều khiển cũng như máy tính nhúng và ở vị trí cao nhất sẽ là cảm biến LIDAR hỗ trợ cho quá trình quét môi trường khi robot hoạt động.



Hình 4.1: Hình chụp thực tế của khung xe.

4.2. ROS2 và các phần phụ thuộc

4.2.1. ROS2 Foxy

Robot được cài đặt ROS theo ý tưởng ban đầu sử dụng ROS2 Foxy (có logo như Hình 4.2) chạy trên hệ điều hành ubuntu linux 20.04 thiết lập và chạy trên máy tính cá nhân và trên Nvidia Jetson TX2 Developer Kit để chạy các tác vụ chính để điều khiển robot.



Hình 4.2: ROS2 foxy

ROS2 có hỗ trợ nhiều chương trình phần mềm giúp người dùng giao tiếp với robot như Rviz2 ta có thể thấy được trạng thái di chuyển của robot cũng như quá trình vẽ bản đồ, né tránh vật cản như nào để có thể thực hiện kiểm tra và đánh giá. Bên cạnh đó, công cụ quản lý gói tin RQT hỗ trợ rất tốt trong quá trình kiểm tra và chỉnh sửa lỗi của hệ thống.

Để cài đặt Ros Foxy, ta thực hiện theo các hướng dẫn tại trang web: [Installing ROS 2 via Debian Packages — ROS 2 Documentation: Foxy documentation](#) [9].

4.2.2. Nav2

Nav2 là một phần vô cùng quan trọng trong đề tài này vì Nav2 là phần chính dùng để so sánh bản đồ đã được lưu với giá trị của LIDAR trả về từ đó cập nhật bản đồ, tính toán vị trí hiện tại, xác định địa điểm A và B nhận diện và thực hiện thiết lập đường đi động, tính toán và điều khiển vận tốc động cơ tránh chướng ngại vật trên đường đi.

Nav2 được cài đặt lên trên ROS2, sử dụng các giá trị đầu vào ban đầu như LIDAR, tốc độ động cơ vị trí góc để hoạt động. Từ đó Nav2 đã thực hiện được tốt các việc quét các vật cản xung quanh nhờ LIDAR với tốc độ tối đa là 50ms/lần. Nhờ việc quét và phát hiện các vật cản di chuyển liên tục thì robot có thể thực hiện né các vật cản cố định cũng như di chuyển trong khu vực.

Để cài đặt gói Nav2, ta thực hiện theo các hướng dẫn tại trang web: [Build and Install — Navigation 2 1.0.0 documentation \(ros.org\)](#) [12].

4.2.3. Micro-ROS

Micro-ROS là phần giao tiếp của ROS2 trên Nvidia Jetson TX2 Developer Kit và RTOS trên STM32F429 Discovery Kit. Micro-ROS được cài đặt trên 2 hệ điều hành ROS2 và RTOS, truyền nhận tín hiệu giữa các thành phần với nhau ổn định không bị mất kết nối hay mất tín hiệu đường truyền.

Để cài đặt gói Micro-ROS trên môi trường ROS2 phục vụ cho quá trình giao tiếp với Micro-ROS trên RTOS, ta thực hiện theo các hướng dẫn tại trang web: [First micro-ROS Application on FreeRTOS | micro-ROS](#).

Để cài đặt gói Micro-ROS trên RTOS, ta thực hiện theo các hướng dẫn tại trang web: [micro-ROS/micro_ros_stm32cubeux_utils: A set of utilities for integrating micro-ROS in a STM32CubeMX project \(github.com\)](#).

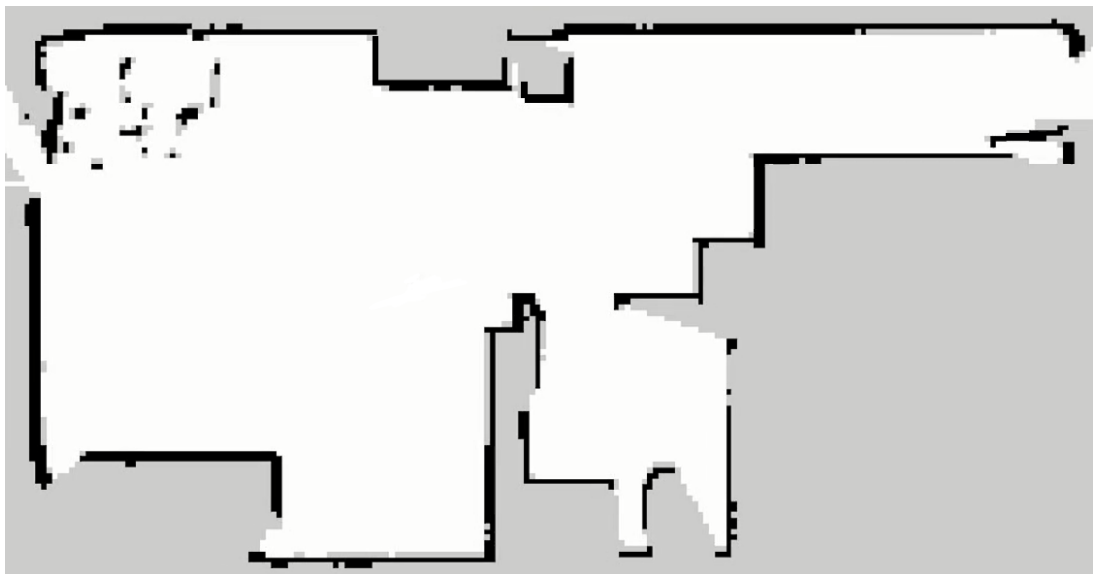
4.2.4. RTOS

Ở đây RTOS điều khiển chính các động cơ phân chia tác vụ con cho từng module khác nhau giúp điều khiển xe đúng theo đường dẫn đã được lập bởi Nav2. Kết quả

nhận được là hệ thống RTOS truyền dẫn tín hiệu điều khiển robot nhanh có độ trễ thấp, không bị ngắt quãng hoặc mất tín hiệu điều khiển. Các tác vụ hoạt động ổn định, không xuất hiện tình trạng deadlock (các tác vụ bị khóa do tranh chấp quyền hoạt động), đặc biệt là phần ngắt ngoài của cảm biến phát hiện vật cản hoạt động tốt, đảm bảo được an toàn cho xe trong những trường hợp xuất hiện vật cản không mong muốn ngoài bản đồ.

4.3. Đánh giá bản đồ 2D

Để đánh giá bản đồ 2D vật cản trong môi trường sẽ là một ảnh hưởng lớn. Với môi trường nhiều vật cản sẽ làm xuất hiện các sai lệch trong quá trình tạo bản đồ. Đối với môi trường ít vật cản, dọn trống môi trường xung quanh, chỉ để lại các vật thể cố định như cột, bàn, ghế... hạn chế những vật cản chuyển động trong quá trình vẽ thì bản đồ được vẽ chính xác hơn. Dưới đây là bản đồ được robot vẽ lại tại một phòng trống (Hình 4.3).



Hình 4.3: Kết quả quá trình vẽ bản đồ 2D của môi trường hoạt động

Bản đồ thể hiện được rõ các đường nét, góc cạnh của các vật thể trong phòng, thể hiện được ở độ chính xác cao. Diện tích bề mặt vật thể tối thiểu có thể nhận diện trong bán kính quét 4.5 m là $4 \times 4 \text{ cm}^2$, cụ thể ở góc trái của Hình 4.3 chính bộ bàn

ghế trong phòng có diện tích bề mặt chân ghế tại vị trí ngang tầm quét của LIDAR là $4.2 \times 4 \text{ cm}^2$.

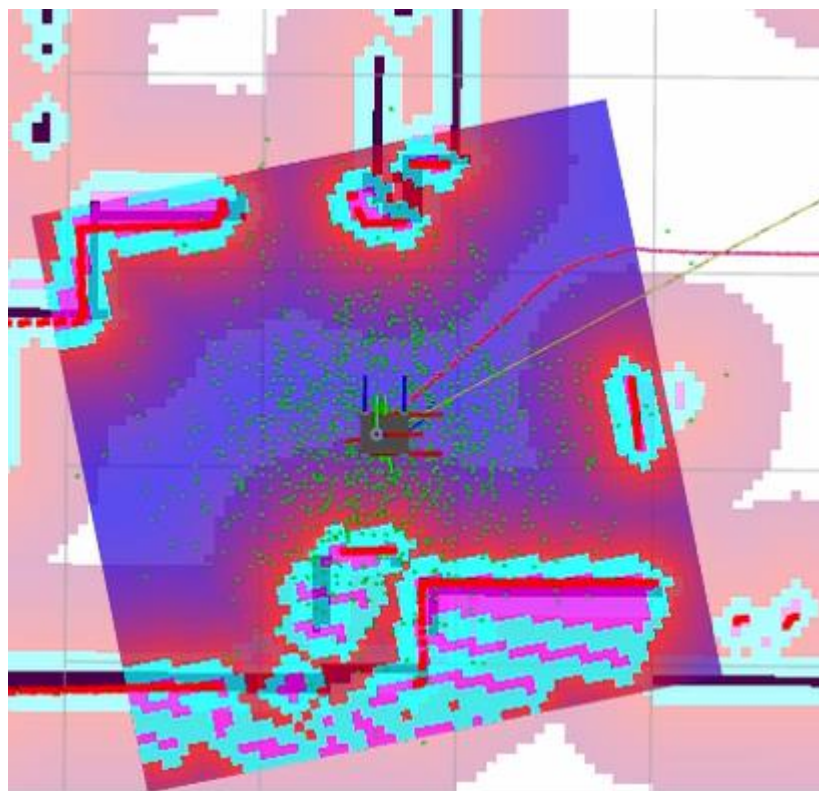
Những vật thể cố định hoặc có thời gian ổn định tại vị trí càng lâu sẽ được thể hiện bằng màu sắc càng đậm. Đối với vật thể di chuyển, giải thuật của Cartographer sẽ dựa vào độ ổn định của vật thể tại vị trí đó theo thời gian và sẽ cập nhật lại bản đồ với mức độ thể hiện đậm dần cho vật thể đó sau mỗi chu kỳ cập nhật, nếu ở chu kỳ cập nhật tiếp theo vật thể rời khỏi vị trí trước đó thì mức độ thể hiện nhạt dần và loại khỏi bản đồ. Chu kỳ cập nhật bản đồ đang được sử dụng là 1 giây, chúng ta hoàn toàn có thể thay đổi được chu kỳ cập nhật này bằng cách chỉnh sửa lại thông số của file thực thi, tuy nhiên nếu chu kỳ quá nhỏ thì độ chi tiết của vật thể sẽ bị giảm, khả năng xảy ra nhiễu cao hơn.

4.4. Đánh giá Navigation Goal

Quá trình đánh giá Navigation Goal sẽ được chia làm hai phần bao gồm thực nghiệm khi không có vật cản và thực nghiệm với có vật cản cố định hoặc di chuyển.

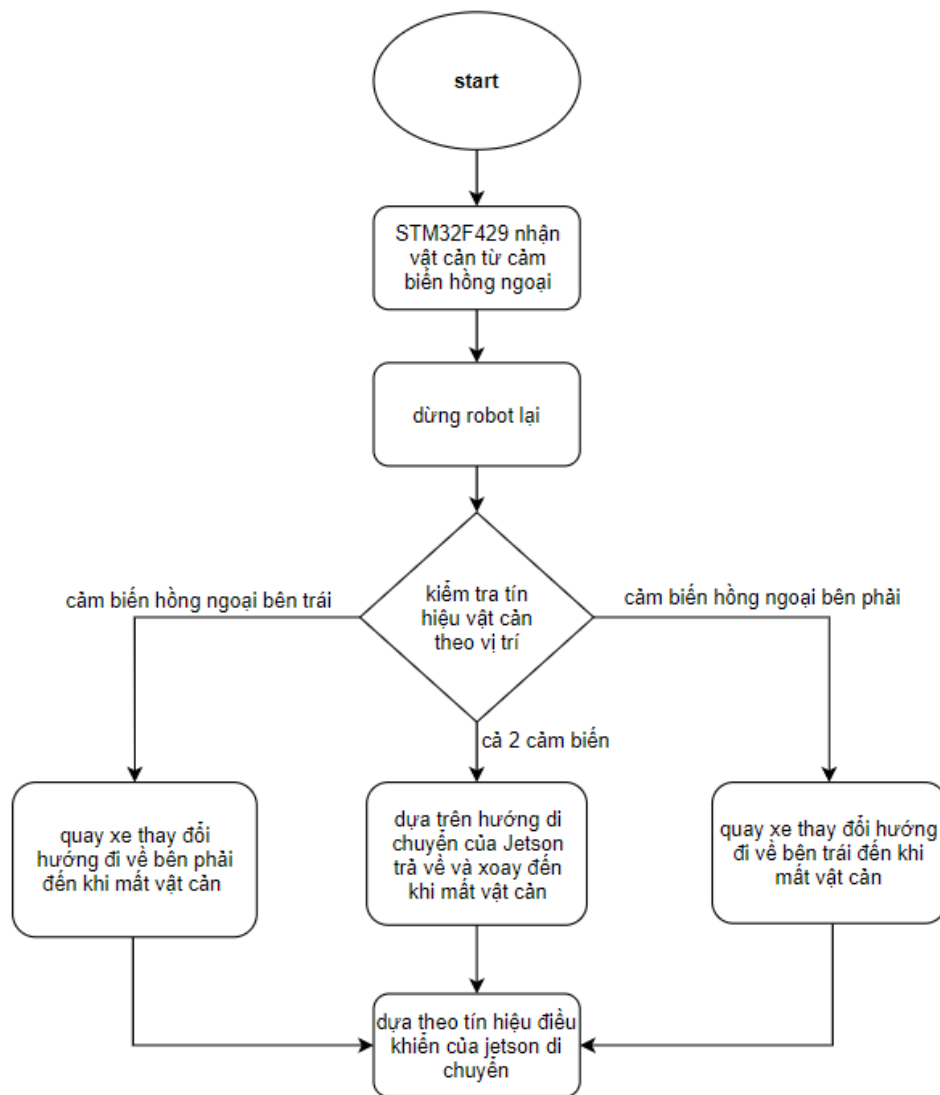
Sau khi có đầy đủ dữ liệu về bản đồ và định vị, thuật toán DWB có thể sẵn sàng tạo đường đi cho bất cứ yêu cầu di chuyển tới đích từ phía người dùng. Trên bản đồ cũng sẽ được tích hợp thêm Costmap 2D, đây là dạng bản đồ 2D của phần không gian bị ngăn trở bởi vật cản hoặc tường ngoài môi trường lan rộng nó ra trong bản đồ dựa trên bản đồ dạng lưới và độ rộng của vùng lan. Vùng được tô màu đen đậm tượng trưng cho vật cản hay tường của bản đồ costmap, phần đồ đậm tượng trưng cho sự lan ra của vật cản hay tường, độ rộng của nó phụ thuộc vào bán kính của robot. Hình 4.4 thể hiện đường đi màu đỏ chính là đường được tạo ra cho robot, thuật toán sẽ tạo ra đường đi bằng tổ hợp n điểm tọa độ sát nhau tạo thành một đường gần như nối liền. Các điểm tạo nên đường đi sẽ ưu tiên nằm ngoài vùng lan ra được tạo bởi costmap nhằm đảm bảo an toàn cho robot khi di chuyển. Vì vậy, nếu có nhiều vật cản xuất hiện trên bản đồ tạo nên nhiều vùng đỏ đậm chồng lên nhau và khoảng trống an toàn không được tạo ra giữa robot và vị trí đích, đường đi sẽ không được tạo và robot sẽ không tiếp tục di chuyển. Tốc độ tạo ra đường đi của

robot khoảng 10 Hz và đáp ứng đủ nhu cầu di chuyển của robot khi môi trường có nhiều vật cản di động.



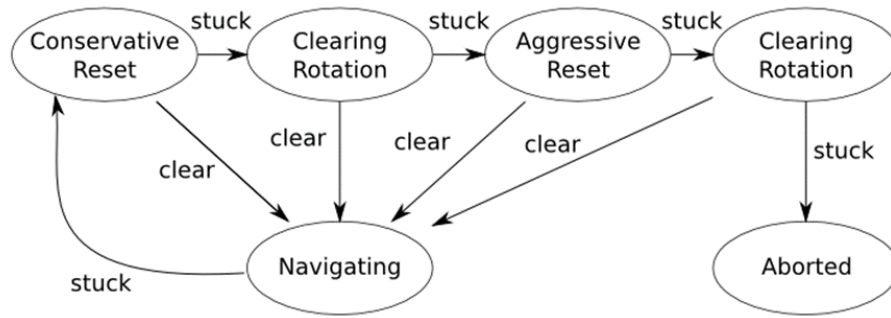
Hình 4.4: Đường đi được tạo ra từ thuật toán DWB

Việc di chuyển theo đường đi được tạo phụ thuộc vào dữ liệu đầu ra của thiết bị được tích hợp trên robot nên những yếu tố khách quan tạo nên những sai lệch cho robot là không thể tránh khỏi. Vì vậy hệ thống các cảm biến phát hiện kịp thời những vật cản nằm ngoài tầm phát hiện trên bản đồ là rất cần thiết. Trong trường hợp này, robot sẽ hoạt động độc lập mà không phụ thuộc vào các tín hiệu điều khiển từ trên Nvidia Jetson TX2 Developer Kit truyền xuống.



Hình 4.5: Sơ đồ mô tả giải thuật xử lý khi robot phát hiện vật cản ngoài bản đồ

Bên cạnh phân tự xử lý khi xuất hiện vật cản ngoài bản đồ, vẫn có những nguyên nhân khiến cho tính toán Odometry bị sai dẫn đến sai lệch giữa tín hiệu quét laser của LIDAR và bản đồ, khi đó robot sẽ tiến hành vào mode recovery. Trong chế độ recovery (Hình 4.6), robot tiến hành xoay vòng cho đến khi tín hiệu quét của LIDAR có thể trùng với bản đồ nhiều nhất.



Hình 4.6: Trạng thái Recovery

4.4.1. Thực nghiệm không có vật cản

Môi trường thực nghiệm trong phòng gồm có sàn nhà, bàn, ghế và tường sau khi thử nghiệm thu được kết quả như Bảng 4.1:

Bảng 4.1: Bảng thông số thực nghiệm không có vật cản

Số lần thử nghiệm	Số lần Goal Reach khi đi thẳng	Số lần Goal Reach khi rẽ hướng
20	19	18

Từ kết quả thu được thấy rằng đối với địa hình là nền nhà bằng phẳng, vật cản trong khu vực ít bị nhiễu nên khả năng di chuyển robot trong khu vực hoạt động tốt.

4.4.2. Thực nghiệm có vật cản

Đối với thực nghiệm có vật cản thì có nhiều yếu tố ảnh hưởng tới kết quả thực nghiệm như kích thước vật cản, tốc độ di chuyển của vật cản, khoảng cách vật cản xuất hiện trước robot. Vì vậy ở lần thử nghiệm này chỉ thực hiện với vật cản không di chuyển lớn trong môi trường nền nhà và với vật cản di chuyển là người đi trong khu vực quét và chắn trước đường đi của robot khoảng 30 cm.

Bảng 4.2: Bảng thông số thực nghiệm có vật cản

Số lần thử nghiệm	Số lần Goal reach vật cản cố định	Số lần Goal reach vật cản di chuyển
20	19	16

Trong quá trình thử nghiệm với vật cản di chuyển lại gần thì một số lần khi gặp vật cản quá gần (nhỏ hơn 30 cm) thì robot vẫn xử lý được khi hoạch định đường đi mới phù hợp nhưng tỷ lệ Goal Reach thấp kết quả thu được như Bảng 4.2.

Tốc độ nhận diện vật cản của và cập nhật lại bản đồ là 100 ms, bên cạnh đó cảm biến phát hiện vật cản hoạt động dường như ngay lập tức hỗ trợ đảm bảo an toàn cho robot khi xuất hiện một vật cản bất ngờ trong khi bản đồ vật cản mới chưa cập nhật kịp.

Chương 5. KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

5.1. Kết quả đạt được

Như vậy, qua khoảng thời gian tìm hiểu nghiên cứu và làm về khóa luận nhóm đã thiết kế được một robot vẽ bản đồ và hoạch định đường đi tránh được vật cản. Nhóm cũng đã học được nhiều kiến thức mới thông qua quá trình làm khóa luận như chuyển động cơ học, thiết kế hệ điều hành RTOS, ROS.

Dựa trên kết quả đạt được, robot đã có thể vẽ được map 2D của một khu vực.

5.2. Ưu điểm và nhược điểm của khóa luận

Qua quá trình tìm hiểu nghiên cứu, thiết kế và xây dựng robot thì nhóm nhận thấy robot có một số ưu điểm cũng như khuyết điểm sẽ được trình bày sau đây:

- Ưu điểm:
 - Ngôn ngữ lập trình hiện đang phổ biến và dễ tiếp cận là C và Python.
 - Hỗ trợ chạy trên nhiều nền tảng hệ điều hành khác nhau như Ubuntu, Window, Mac OS, RTOS thuận lợi trong làm việc.
 - Sử dụng chuẩn giao tiếp DDS hỗ trợ giao tiếp mạng nội bộ thuận lợi và dễ dàng.
- Nhược điểm:
 - Khả năng bị nhiễu do khu vực vẽ bản đồ không gọn gàng.
 - Tốc độ di chuyển thấp.
 - Chưa thể di chuyển qua các bề mặt có độ gồ ghề lớn.
 - Khó khăn trong cài đặt nhiều package.
 - ROS2 là một chương trình khá mới để có thể tìm tài liệu khắc phục vấn đề gặp khi dùng robot.

5.3. Hướng phát triển

- Tích hợp thêm camera, hỗ trợ vẽ bản đồ 3D và nhận diện vật cản tốt hơn, tối ưu cho quá trình tự hành và né tránh vật cản.
- Tối ưu phân cứng để xe có thể di chuyển trên nhiều địa hình khác nhau hơn.

TÀI LIỆU THAM KHẢO

- [1] T. Nhận, "vnexpress," Vingroup thử nghiệm xe điện tự hành cấp độ 4, 8 8 2021. [Online]. Available: <https://vnexpress.net/vingroup-thu-nghiem-xe-dien-tu-hanh-cap-do-4-4337563.html>.
- [2] N. Nam, "baodautu," FPT thử nghiệm thành công xe tự hành tại Ecopark, 1 11 2019. [Online]. Available: <https://baodautu.vn/fpt-thu-nghiem-thanh-cong-xe-tu-hanh-tai-ecopark-d110172.html>.
- [3] N. nguyên, "vietnamfinance," Cận ảnh chiếc xe 'không người lái' made in Việt Nam của Tập đoàn Phenikaa, 28 3 2021. [Online]. Available: <https://vietnamfinance.vn/can-anh-chiec-xe-khong-nguoi-lai-made-in-viet-nam-cua-tap-doan-phenikaa-20180504224251226.htm>.
- [4] V. A. T. Trần Hoàng Phương, "Robot tránh vật cản sử dụng công nghệ LIDAR," HCM, 2021.
- [5] L. C. B. Phan Anh Kệt, "Xây dựng Robot tự hành quét bản đồ trong nhà và tránh vật cản," HCM, 2021.
- [6] ROBOTIS, Turtlebot 3: Waffle Pi & Burger, Jekyll & Minimal Mistakes., 2021.
- [7] omorobot, "OMOROBOT," R1 mini An Open Mobile Platform for autonomous driving education and development, 2021. [Online]. Available: <https://omorobot.com/en/portfolio-item/r1-mini/>.
- [8] K. NGUYỄN, "Tổng quan về hệ điều hành thời gian thực RTOS," pp. <https://khuenguyencreator.com/tong-quan-ve-he-dieu-hanh-thoi-gian-thuc-rtos/>, 15 07 2021.
- [9] Robotics, "Robotics," ROS 2 DocumentationÁ, 2021. [Online]. Available: <https://docs.ros.org/en/foxy/index.html>.
- [10] A. Sears-Collins, "Automatic Addison," Navigation and SLAM Using

- the ROS 2 Navigation Stack, 10 9 2021. [Online]. Available: <https://automaticaddison.com/navigation-and-slam-using-the-ros-2-navigation-stack/>.
- [11] W. B. S. T. D. Fox, "semanticscholar," 1 3 1997. [Online]. Available: <https://www.semanticscholar.org/paper/The-dynamic-window-approach-to-collision-avoidance-Fox-Burgard/ccfb06935ea9c8832ec6086000f71cf584a79679#citing-papers>.
- [12] S. a. M. F. a. W. R. a. G. C. J. Macenski, "The Marathon 2: A Navigation System," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2020.