

ĐẠI HỌC QUỐC GIA TPHCM
TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA KHOA HỌC VÀ KỸ THUẬT MÁY TÍNH



BÁO CÁO
ĐỒ ÁN TỐT NGHIỆP KỸ THUẬT MÁY TÍNH

Hiện thực một mô hình
robot dịch vụ tự hành

Ngành: Kỹ thuật Máy tính

HỘI ĐỒNG: Kỹ Thuật Máy Tính
GVHD: TS. Phạm Hoàng Anh

SVTH 1: Lương Hữu Phú Lợi - 1911545
SVTH 2: Cao Thanh Lương - 1914076

Tp. Hồ Chí Minh, Tháng 05/2023

Phần ký duyệt

Phần này xác nhận rằng báo cáo Đồ án tốt nghiệp Kỹ thuật Máy Tính của Lương Hữu Phú Lợi và Cao Thanh Lương được Giảng viên hướng dẫn (GVHD) phê duyệt là theo yêu cầu của Khoa Khoa Học và Kỹ Thuật Máy Tính.

Chữ ký phê duyệt của Giảng viên hướng dẫn.

Ký tên

Ngày ký tên

Phạm Hoàng Anh
Giảng viên hướng dẫn

Lời cam đoan

Chúng tôi xin cam đoan rằng mọi điều được trình bày trong báo cáo, cũng như mã nguồn là do chúng tôi tự thực hiện. Ngoại trừ các kiến thức tham khảo có trích dẫn cũng như mã nguồn mẫu do chính nhà sản xuất, cộng đồng mã nguồn mở cung cấp, hoàn toàn không sao chép từ bất cứ nguồn có bản quyền nào khác. Nếu lời cam đoan trái với sự thật, nhóm xin chịu mọi trách nhiệm trước Ban Chủ Nhiệm Khoa và Ban Giám Hiệu Nhà Trường.

Lương Hữu Phú Lợi, Cao Thanh Lương

Lời cảm ơn

Chúng em xin được gửi lời cảm ơn chân thành và sâu sắc nhất đến tất cả mọi người đã luôn ủng hộ, giúp đỡ chúng em trong thời gian thực hiện đồ án vừa qua. Đồ án này sẽ không thể thực hiện được nếu không có sự hỗ trợ của mọi người. Đặc biệt là giáo viên hướng dẫn, tiến sĩ Phạm Hoàng Anh, Thầy đã tạo mọi điều kiện để chúng em có thể hiện thực hóa ý tưởng, thỏa chí sáng tạo, thể hiện hết khả năng của mình. Bên cạnh đó, Thầy còn có những góp ý đúng lúc, những lời động viên kịp thời khi chúng em gặp khó khăn trong suốt quá trình thực hiện đồ án.

Chúng em cũng xin cảm ơn toàn thể thầy cô trong khoa Khoa học và Kỹ thuật Máy tính nói riêng cùng thầy cô thuộc trường Đại học Bách khoa thành phố Hồ Chí Minh nói chung, đã tận tình chỉ dạy, cung cấp những kiến thức cốt lõi để từ đó chúng em có thể tiếp tục theo đuổi đam mê, chinh phục ước mơ của mình, trở thành một người kỹ sư giỏi.

Cuối cùng, chúng em xin bày tỏ lòng biết ơn đến cha mẹ của mình, những người luôn tạo mọi điều kiện tốt nhất cho sự phát triển của chúng em và luôn động viên khích lệ mỗi khi chúng em gặp khó khăn, thử thách trong cả học tập và cuộc sống. Ngoài ra, chúng mình cũng xin cảm ơn tất cả những người bạn, những người đã dùi dắt, giúp đỡ nhau trong suốt thời gian học tập tại trường Đại học Bách Khoa.

Lương Hữu Phú Lợi, Cao Thanh Lương

Tóm tắt

Đề tài tập trung khảo sát, nghiên cứu mô hình Robot dịch vụ **có sẵn**, những khía cạnh liên quan đến SLAM, các phương pháp giải SLAM cụ thể là phương pháp RTAB-Map. Bên cạnh đó, tìm hiểu về nền tảng phát triển ứng dụng robot, Robot Operating System (ROS), để từ đó xây dựng ứng dụng Robot tự hành, một nền tảng robot phụ vụ con người, dễ dàng mở rộng và triển khai trong các mô hình ứng dụng thực tế, có các chức năng như sau:

- Vẽ bản đồ 2D/3D.
- Định vị trong bản đồ đã xây dựng được.
- Tự hành đến vị trí cố định.
- Tránh vật cản trong quá trình vận hành.
- Tương tác với người dùng thông qua web, màn hình.

Mục lục

1 Giới thiệu	10
1.1 Lý do thực hiện đề tài	10
1.1.1 Đặt vấn đề	10
1.1.2 Động lực thúc đẩy	11
1.2 Mục tiêu của đề tài	12
1.3 Nội dung thực hiện	12
2 Kiến thức nền tảng liên quan	14
2.1 Phân tích đề tài	14
2.1.1 Robot dịch vụ	14
2.1.2 Một số loại robot dịch vụ trên thị trường	14
2.2 Các kiến thức nền tảng	16
2.2.1 SLAM (Simultaneous Localization and Mapping)	16
2.2.2 Những khái niệm căn bản trong SLAM	16
2.2.3 Xử lý thông tin không tin cậy trong SLAM	19
2.2.4 Kỹ thuật chung giải quyết bài toán SLAM	19
2.2.5 Các phương pháp giải SLAM phổ biến	21
2.2.6 RTAB-Map	22
2.2.7 Hệ điều hành cho Robot (Robot Operating System - ROS)	28
2.2.8 Gazebo - Công cụ mô phỏng Robot	32
3 Thiết kế và hiện thực hệ thống	33
3.1 Kiến trúc mô hình hệ thống/giải pháp	33
3.1.1 Kiến trúc chung	33
3.1.2 Sơ đồ các Node ROS	34
3.1.3 Các chức năng cho người dùng	35
3.1.3.1 Người dùng Phổ thông	35
3.1.3.2 Người dùng Quản lý	36
3.2 Kết quả hiện thực	36
3.2.1 Mô hình robot mô phỏng trên Gazebo	36
3.2.1.1 Xây dựng mô hình robot	36
3.2.1.2 Xây dựng khu vực mô phỏng	37
3.2.1.3 Xây dựng map với rtabmap trên Gazebo	39
3.2.2 Mô hình robot thực tế	41
3.2.2.1 Các thiết bị	41

3.2.2.2 Xây dựng map với Rtabmap	43
3.2.3 Thiết kế các message, services và topics	44
3.2.3.1 Message và Service	44
3.2.3.2 Truy xuất và gọi các message, service	45
3.2.3.3 Topics	46
3.2.4 Xử lý hàng chờ các Task	46
3.2.5 Màn hình giao tiếp với người dùng	48
3.2.5.1 Chức năng người dùng	48
3.2.5.2 Tạo giao diện người dùng	49
3.2.5.3 Tạo node đọc dữ liệu người dùng	51
3.2.5.4 Sơ đồ giao tiếp của hệ thống	53
3.2.6 Web Server	53
3.2.6.1 Xây dựng giao diện Web tương tác với Ros	53
3.2.6.2 Các chức năng của Web	54
3.3 Thử nghiệm và đánh giá	59
3.3.1 Quá trình xây dựng bản đồ	59
3.3.2 Khả năng tự hành trong thời gian dài	62
3.3.2.1 Chuẩn bị	63
3.3.2.2 Tiến hành thử nghiệm	65
4 Kết luận	78
4.1 Kết quả đạt được	78
4.1.1 Khả năng xây dựng bản đồ	78
4.1.2 Khả năng tự hành	78
4.1.3 Giao tiếp với người dùng	79
4.1.4 Điều khiển robot thông qua Web	79
4.2 Những hướng phát triển	80
Tài liệu tham khảo	81

Danh sách hình vẽ

1	Máy bay không người lái (Drone aircraft)	10
2	Robot phục vụ tại một cửa hàng cafe ở Nhật Bản	11
3	Dinerbot T5	15
4	Deebot	15
5	Aethon Hotel Robot	16
6	Ví dụ về kỹ thuật Mapping	18
7	Mã giả của SLAM	20
8	Mô hình quản lý bộ nhớ trong RTAB-Map [10]	23
9	Mô hình quản lý bộ nhớ trong RTAB-Map [10]	24
10	Front-end của RTAB-Map [10]	26
11	Back-end của RTAB-Map [10]	27
12	Thiết lập các thành phần triển khai RTAB-Map lên robot [3]	28
13	Minh họa hệ thống tập tin của ROS [12]	29
14	Packages trong ROS [12]	29
15	Computation Graph Level [12]	30
16	Kiến trúc chung của hệ thống	33
17	Sơ đồ các Node ROS trong hệ thống	34
18	Sơ đồ hoạt động của người dùng	35
19	Mô hình robot mô phỏng trên gazebo	37
20	Map mô phỏng trên gazebo (Map 1)	38
21	Map mô phỏng trên gazebo (Map 2)	38
22	Hình ảnh camera và lidar trên rviz (Map 1)	39
23	Hình ảnh camera và lidar trên rviz (Map 2)	39
24	Map xem trên rviz (Map 2)	41
25	Kobuki Robot	42
26	Kobuki Robot	42
27	Xây dựng map với Rtabmap	44

28	Các chức năng của Node Station Server	46
29	Cách Queue Node hoạt động	47
30	Các topic của Queue Node	48
31	Sơ đồ hoạt động của người dùng	49
32	Giao diện người dùng	50
33	Báo lỗi khi chọn lắp vị trí	51
34	Xác nhận khi chọn	51
35	Serial Node	52
36	Các thành phần giao tiếp với nhau	53
37	Web UI	53
38	Hình ảnh camera mô phỏng trên Gazebo	55
39	Hình ảnh camera thực tế	55
40	Vận tốc hiện tại của robot	56
41	Điều chỉnh Linear Velocity hoặc Angular Velocity	56
42	Bộ điều khiển thủ công robot	57
43	Mode Setgoal	57
44	Khi robot đến đích	58
45	Thêm trạm có tên Default	58
46	Xóa trạm có tên NoName	59
47	Mapping toàn bộ phòng 709	60
48	Mapping sảnh chính phòng 709	61
49	Robot "nhìn thấy" một khu vực hai lần	62
50	Khu vực thử nghiệm	63
51	Dán dấu vị trí Robot tại các điểm trên bản đồ	64
52	Các thông số tùy chỉnh	64
53	Vận tốc nhỏ và gia tốc nhỏ	65
54	Thí nghiệm 1: vị trí 1	66
55	Thí nghiệm 1: vị trí 2	66

56	Thí nghiệm 1: vị trí 3	67
57	Thí nghiệm 1: vị trí 1 lần 2	67
58	Vận tốc nhỏ và gia tốc lớn	68
59	Thí nghiệm 2: vị trí 1	68
60	Thí nghiệm 2: vị trí 2	69
61	Thí nghiệm 2: vị trí 3	69
62	Thí nghiệm 2: vị trí 1 lần 2	70
63	Vận tốc lớn và gia tốc lớn	70
64	Thí nghiệm 3: vị trí 1	71
65	Thí nghiệm 1: vị trí 2	71
66	Thí nghiệm 1: vị trí 3	72
67	Thí nghiệm 3: vị trí 1 lần 2	72
68	Thí nghiệm 1: vị trí 2 lần 2	73
69	Thí nghiệm 1: vị trí 3 lần 2	73
70	Vị trí khác trên bản đồ lần 1	74
71	Vị trí thực tế lần 1	75
72	Vị trí khác trên bản đồ lần 2	76
73	Vị trí thực tế lần 2	77
74	Hình ảnh 2D và 3D của bản đồ	78
75	Giao tiếp qua màn hình và loa	79
76	Giao diện Web	80

1 Giới thiệu

1.1 Lý do thực hiện đề tài

1.1.1 Đặt vấn đề

Hiện nay, mô hình robot dịch vụ tự hành đang ngày càng phổ biến trên thế giới, đặc biệt là trong giai đoạn số hóa với tốc độ phát triển chóng mặt như hiện nay. Các công nghệ mới phát triển như vũ bão để phục vụ nhu cầu của con người và robot tự hành cũng là một phần trong số đó. Nó được thiết kế để thực hiện các hoạt động mà hầu như không có sự tham gia của con người. Và từ đây, robot tự hành không còn là điều khoa học viễn tưởng, nó đã mang lại những sự đổi mới và tạo ra các giá trị to lớn cho con người, có thể kể đến như là:

- Cải thiện tốc độ và độ chính xác trong các quy trình ở các nhà máy công nghiệp.
- Tăng tính hiệu quả trong công việc khi robot có thể làm việc song song với con người.
- Giảm thiểu các rủi ro đối với con người trong các công việc nguy hiểm.



Hình 1: Máy bay không người lái (Drone aircraft)



Hình 2: Robot phục vụ tại một cửa hàng cafe ở Nhật Bản

Trên thế giới hiện tại có rất nhiều loại robot phục vụ cho nhiều mục đích khác nhau có thể kể đến như là những quá trình tự động trong công nghiệp đến những robot trên không, robot tự hành có khả năng thu thập dữ liệu và hình ảnh. Tùy thuộc vào nhu cầu và mục đích sử dụng chúng sẽ có sự khác nhau về kích thước, khả năng, tính di động, trí tuệ nhân tạo, chi phí. Trong tương lai, robot tự hành sẽ được tích hợp trí tuệ nhân tạo để chúng có thể thu thập, học hỏi từ môi trường xung quanh từ đó tự đưa ra quyết định.

1.1.2 Động lực thúc đẩy

Cuộc Cách mạng Công nghiệp lần thứ 4 đang dẫn đến một nền công nghiệp tự động hóa, trong đó robot tự động được sử dụng trong mọi khu vực của nhà máy, từ vận hành và giám sát trực tiếp đến quản lý và ra quyết định. Hơn nữa, toàn cầu đã trải qua một đợt dịch bệnh Covid-19 khó khăn chưa từng có, đây cũng là lúc thúc đẩy các nhóm nghiên cứu đang phát triển giải pháp robot tự động có thể vận chuyển thực phẩm đến từng cá nhân. Nhiệt độ cơ thể của bệnh nhân được tự động kiểm tra để hạn chế tiếp xúc giữa người với người và ngăn ngừa bệnh lây lan.

Ngoài thực hiện các tác vụ trong những môi trường xác định biết trước và tách biệt hoàn toàn với con người, việc áp dụng những công nghệ mới đem lại những hệ thống robot hiện đại có khả năng vận hành không chỉ trong những môi trường chưa biết trước mà còn có thể tương tác trực tiếp với con người hoặc có khả năng cộng tác với nhau. Theo đó, những năm gần đây nhiều công trình nghiên cứu được công bố với nỗ lực trong việc tìm kiếm những giải pháp mang tính cách mạng

trong công nghệ cảm biến và công nghệ năng lượng từ đó giúp cho robot tự hành có thể khám phá thế giới xung quanh một cách tốt hơn, đưa ra quyết định và vận hành hiệu quả hơn.

Cũng bởi khi được triển khai ở một môi trường chưa biết trước chẳng hạn như nhà ở hay văn phòng, thật khó để cung cấp sẵn cho robot một mô hình chính xác về môi trường nơi robot làm việc. Do đó, điều cần thiết trước hết là cung cấp cho robot những giải thuật cùng những cảm biến cần thiết giúp cho robot có thể tự khám phá môi trường xung quanh. Xây dựng bản đồ dựa vào những cảm biến trên robot là một thách thức không hề nhỏ, bởi lẽ robot cần biết vị trí của mình, kết hợp với thông tin từ cảm biến để cập nhật bản đồ, tuy nhiên để biết được vị trí hiện tại thì robot lại cần phải dựa vào thông tin từ bản đồ. Những tác vụ này do đó cần phải thực hiện một cách đồng thời, trong ngành robot học vấn đề này được gọi là SLAM (simultaneous localization and mapping) tức là đồng thời định vị và xây dựng bản đồ.

1.2 Mục tiêu của đề tài

Với mục đích tập trung khảo sát, nghiên cứu mô hình Robot dịch vụ **có sẵn**, những khía cạnh liên quan đến SLAM, các phương pháp giải SLAM cụ thể là phương pháp RTAB-Map. Bên cạnh đó, tìm hiểu về nền tảng phát triển ứng dụng robot, Robot Operating System (ROS), để từ đó xây dựng ứng dụng Robot tự hành, một nền tảng robot phụ vụ con người, dễ dàng mở rộng và triển khai trong các mô hình ứng dụng thực tế.

Mục tiêu cuối cùng của đề tài đó là xây dựng một nền tảng robot phục vụ có các chức năng như sau:

- Xây dựng bản đồ (Mapping) cả 2D lẫn 3D trong không gian robot hoạt động.
- Định vị (Localization) trong môi trường bản đồ đã được xây dựng.
- Tự hành (Navigation) đến vị trí cố định trên bản đồ.
- Tránh vật cản (Obstacle avoidance) ở các độ cao khác nhau.
- Tương tác với người dùng thông qua giao diện Web, màn hình cảm ứng trên robot.
- Lắp thêm loa cho robot để phát triển các tính năng âm thanh khác như giao tiếp với con người, thông báo trạng thái.

1.3 Nội dung thực hiện

Để đạt được mục tiêu đề ra, cần tìm hiểu những vấn đề sau:

- Khảo sát một số loại Robot dịch vụ trên thị trường.
- Nghiên cứu về SLAM, cả về SLAM dựa trên cảm biến lidar và SLAM dựa trên thị giác máy tính.

- Nghiên cứu một số giải pháp SLAM mã nguồn mở, đặc biệt là giải thuật SLAM bằng phương pháp RTAB-Map.
- Tìm hiểu về ROS - nền tảng phát triển ứng dụng robot.
- Tìm hiểu về Gazebo - công cụ mô phỏng robot.
- Thực hiện đề tài trên mô hình robot TurtleBot 2 - Kobuki.

2 Kiến thức nền tảng liên quan

2.1 Phân tích đề tài

2.1.1 Robot dịch vụ

Theo Wikipedia, Robot dịch vụ hay Robot phục vụ là robot hỗ trợ con người, thường bằng cách thực hiện công việc bẩn thỉu, đơn giản, xa xôi, nguy hiểm, hoặc lặp đi lặp lại, kể cả công việc nội trợ. Các robot này thường là tự trị và/hoặc được vận hành bởi một hệ thống điều khiển tích hợp, với các tùy chọn điều khiển bằng tay tác động đè (tác động tức thì).

Thuật ngữ "robot dịch vụ" không có định nghĩa kỹ thuật nghiêm ngặt. Liên đoàn Robot học Quốc tế (IFR) đã đề xuất định nghĩa dự kiến: "Robot dịch vụ là robot hoạt động bán hoặc hoàn toàn tự động để thực hiện các dịch vụ hữu ích cho nhu cầu của con người và thiết bị, ngoại trừ hoạt động sản xuất".

Robot có thể được ứng dụng để hỗ trợ công việc của con người trong nhiều lĩnh vực khác nhau. Dưới đây là một số loại robot phổ biến:

- **Robot công nghiệp:** Được sử dụng trong các môi trường công nghiệp để thực hiện các nhiệm vụ từ đơn giản đến phức tạp. Chúng có thể được sử dụng trong việc kiểm tra hàn, lắp ráp sản phẩm, hoặc thậm chí làm việc trong môi trường nguy hiểm như các nhà máy điện hạt nhân.
- **Robot gia dụng:** Thực hiện các nhiệm vụ hàng ngày trong môi trường không phải công nghiệp, như làm sạch nhà, cắt cỏ hoặc bảo trì hồ bơi. Robot gia đình có thể hỗ trợ người khuyết tật hoặc người lớn tuổi, giúp họ sống độc lập, và có thể đóng vai trò như một trợ lý hoặc quản gia.
- **Robot khoa học:** Sử dụng trong các nghiên cứu khoa học và có khả năng thực hiện nhiều chức năng. Chúng có thể thực hiện các nhiệm vụ lặp đi lặp lại, như thu thập mẫu gen và xác định trình tự. Các robot khoa học cũng có thể thực hiện các nhiệm vụ khó khăn hoặc không thể tiếp cận bởi con người, như khám phá đáy biển sâu hay thám hiểm không gian vũ trụ.

2.1.2 Một số loại robot dịch vụ trên thị trường

- **Dinerbot T5:** áp dụng công nghệ tự định vị và điều hướng tiên tiến. Ngoài ra còn có công nghệ tránh chướng ngại vật thông minh mới với cảm biến chướng ngại vật một chiều.



Hình 3: Dinerbot T5

- **Deebot:** robot dọn vệ sinh cao cấp cho những nơi khó tiếp cận. Với thiết kế ưu việt và công nghệ điều hướng hiện đại, robot dựng bản đồ trực quan của nhà bạn trong khi hút bụi và lau nhà, để lại sàn nhà sạch không tì vết.



Hình 4: Deebot

- **Aethon Hotel Robot:** với các chức năng tạo trải nghiệm an toàn, hiệu quả và đáng nhớ cho khách bằng cách tự động hóa việc giao đồ ăn, đồ dùng trong phòng và các yêu cầu của khách; hỗ trợ nhân sự khi cần. Robot có thể làm việc 24/7; cung cấp hỗ trợ hậu cần cho công việc hậu cần đồng thời cũng rất phù hợp trong các tình huống giãn cách xã hội.



Hình 5: Aethon Hotel Robot

2.2 Các kiến thức nền tảng

2.2.1 SLAM (Simultaneous Localization and Mapping)

SLAM nói đến vấn đề xây dựng bản đồ của một môi trường không xác định bằng robot di động (mobile robot) đồng thời điều hướng môi trường bằng cách sử dụng bản đồ đó.

Môi trường, trạng thái, cột mốc, cách robot tác động đến môi trường, các mô hình toán học, lập bản đồ, định vị và điều hướng đều là một phần của SLAM. Giới thiệu các mô hình xử lý thông tin không tin cậy phổ biến trong việc xây dựng giải thuật SLAM, cũng như giới thiệu phương pháp chung để giải quyết một bài toán SLAM.

2.2.2 Những khái niệm căn bản trong SLAM

- **Môi trường (Environment)**

Môi trường của robot là một hệ thống động với nhiều trạng thái bên trong. Các cảm biến cho phép robot thu thập thông tin về môi trường xung quanh. Tuy nhiên, các cảm biến thường có những hạn chế và có rất nhiều dữ liệu không thể thu thập trực tiếp từ môi trường. Kết quả là, robot cần duy trì niềm tin nội bộ liên quan đến trạng thái của môi trường xung quanh.

- **Trạng thái (State)**

Môi trường được đặc trưng bởi trạng thái. Trạng thái có thể thay đổi theo thời gian, ví dụ như vị trí của con người, hay có thể không thay đổi như vị trí của tường. Trạng thái có thể thay đổi được gọi là trạng thái động (dynamic), còn trạng thái không thay đổi được gọi là trạng thái tĩnh (static). Trạng thái cũng bao gồm những biến liên quan đến chính robot như tư thế robot (posture), vận tốc (velocity) hay hoạt động của cảm biến.

- **Cột mốc (Landmark)**

Cột mốc là các đặc điểm có thể dễ dàng phân biệt, quan sát lại và nhận ra từ môi trường xung quanh. Robot sử dụng các cột mốc này để tìm

ra vị trí hiện tại của nó trong không gian (định vị). Hãy tưởng tượng bạn bị bịt mắt để hiểu cách thức hoạt động của robot. Khi bạn bị bịt mắt và đi quanh ngôi nhà, bạn có thể với tay và chạm vào đồ vật hoặc bám vào tường để tránh bị lạc. Một số đặc điểm, chẳng hạn như cảm giác chạm vào khung cửa, có thể giúp bạn xác định vị trí của mình. Đối với robot, cảm giác chạm được cung cấp thông qua sonar và laser.

Các mốc cần được quan sát lại, chúng nên được phát hiện từ các vị trí và các góc độ khác nhau. Cột mốc phải đủ độc đáo để có thể dễ dàng xác định nhiều lần mà không gây nhầm lẫn giữa các mốc. Nói cách khác, nếu quan sát lại một cột mốc tại thời điểm sau đó, ta phải dễ dàng xác định điểm này là cột mốc nào đã xác định trước đó.

Cột mốc mà ta chọn để robot có thể nhận diện được không nên quá ít trong môi trường vì robot có thể phải dành một khoảng thời gian dài mà không tìm được cột mốc nào, điều này có thể khiến robot bị mất phương hướng trong không gian. Nên chọn cột mốc tĩnh vì nếu cột mốc không ở yên một chỗ, robot không thể xác định được vị trí dù xác định được cột mốc. Từ đó, ta có các yếu tố chính trong việc xác định cột mốc phù hợp:

- Cột mốc có thể dễ dàng quan sát lại.
- Cột mốc có thể dễ dàng phân biệt với các cột mốc khác.
- Cột mốc nên xuất hiện nhiều lần trong môi trường.
- Cột mốc phải cố định (tĩnh).

• Tương tác với môi trường

Có hai kiểu tương tác cơ bản giữa robot và môi trường: Robot có thể tác động đến trạng thái của môi trường thông qua các bộ truyền động. Và robot có thể thu thập thông tin về trạng thái của môi trường thông qua các cảm biến.

– Thu thập dữ liệu từ cảm biến (Sensor measurements): là quá trình robot sử dụng cảm biến để thu thập thông tin về trạng thái của môi trường như chụp ảnh hoặc quét laser. Kết quả của sự tương tác này gọi là các phép đo và các phép đo này thường có độ trễ về thời gian, do đó robot cập nhật trạng thái môi trường muộn hơn một khoảng thời gian không đáng kể so với thực tế. Đây có thể xem là nhận thức của robot về môi trường xung quanh.

– Hoạt động của robot (Control actions): Bằng cách tác động lên môi trường, robot có thể thay đổi trạng thái của môi trường. Chuyển động và tương tác với các đối tượng trong môi trường là các ví dụ về hoạt động của robot trong môi trường. Ngay cả khi robot không thực hiện bất kỳ hành động nào, trạng thái của nó vẫn có thể thay đổi. Do đó, để đảm bảo tính nhất quán, chúng tôi giả định rằng robot luôn thực hiện một hành động, ngay cả khi nó không di chuyển hay tương tác với môi trường xung

quanh. Trên thực tế, robot liên tục hoạt động và thu thập dữ liệu.

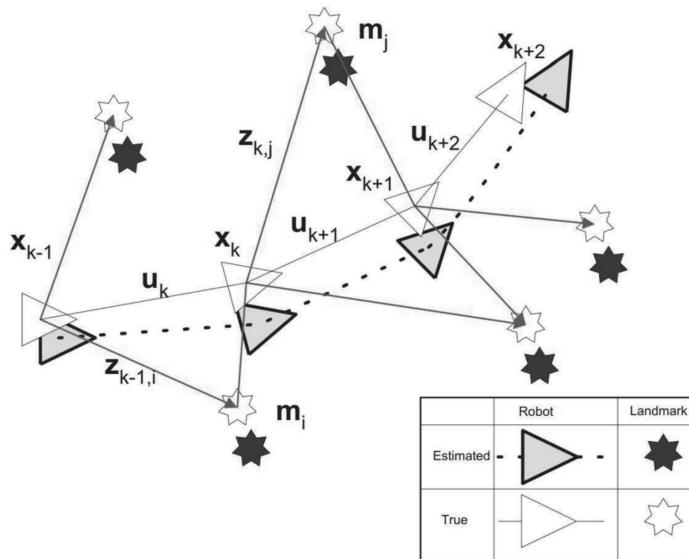
Có hai luồng dữ liệu riêng biệt tương ứng với hai loại tương tác được mô tả ở trên.

- **Dữ liệu đo lường (Measurement data)**

- **Dữ liệu hoạt động (Control data)**

- **Xây dựng bản đồ (Mapping)**

Đây là kỹ thuật khi robot đã biết vị trí của nó và sử dụng các cảm biến để ước tính vị trí của các cột mốc. Trong quá trình hoạt động, robot thực hiện các phép đo liên tục để thu thập thông tin về những cột mốc cũng như những vật cản xung quanh nó để từ đó kết hợp với vị trí hiện tại của mình, ước lượng vị trí của các cột mốc, vật cản. Tuy nhiên là việc ước lượng trạng thái của môi trường xung quanh thì không thể chính xác 100%, luôn tồn tại sai số, đặc biệt việc ước lượng vị trí này dựa vào các cảm biến, mà cảm biến thì luôn tồn tại nhiều ở mỗi lần đo. Ở hình minh họa bên dưới biểu diễn vị trí cột mốc được ước lượng trong một hình elipse thể hiện sự sai số trong phép ước lượng. Và phép ước lượng Gaussian là một giải pháp hiệu quả để biểu diễn việc ước lượng trạng thái xung quanh trong môi trường có nhiều ở cảm biến.



Hình 6: Ví dụ về kỹ thuật Mapping

- **Định vị (Localization)**

Một thành phần khác của SLAM là định vị, thường được sử dụng để ước lượng vị trí của robot bằng cách tính toán vị trí của các cột mốc xung quanh và sử dụng dữ liệu có được từ bước xây dựng bản đồ. Trong quá trình di chuyển theo quỹ đạo, robot liên tục phát hiện các mốc dựa trên dữ liệu nhận được từ cảm biến và robot có thể ước tính vị trí của nó dựa trên sự hiểu

biết về vị trí của các cột mốc. Một ví dụ về định vị trong SLAM, thường thì robot thực hiện việc quan sát có vẻ tốt ở lúc bắt đầu, tuy nhiên sau khi di chuyển về phía trước một thời gian, mô hình ước lượng trên robot phát hiện robot đã rẽ trái (do các yếu tố môi trường tác động lên robot làm lệch quỹ đạo của robot), và quỹ đạo đúng của robot là phải rẽ phải, do đó robot thực hiện quan sát xung quanh để tìm kiếm những cột mốc ứng với bản đồ nội tại và tiến hành điều chỉnh vị trí của robot, đưa nó về vị trí mong muốn. Khả năng điều chỉnh vị trí của robot dựa vào độ chính xác của cảm biến và hiệu năng của mô hình động học.

2.2.3 Xử lý thông tin không tin cậy trong SLAM

Một trong những thách thức lớn trong SLAM đó là kết quả thu thập từ việc đo đạc, hay nhận thức về thế giới xung quanh thông qua hệ thống cảm biến, thường không tin cậy. Những sai số nhỏ trong phép đo, nếu không có những phương pháp đánh giá và hiệu chỉnh phù hợp có thể dẫn đến những sai lệch lớn cho toàn hệ thống. Hơn nữa, SLAM là một quá trình đệ quy, những kết quả sai có thể bị cộng dồn. Do đó bất kỳ hệ thống SLAM nào cũng cần có những mô hình, kỹ thuật để loại bỏ những dữ liệu không đáng tin cậy, làm giảm độ chính xác trong quá trình thu thập dữ liệu và kể cả trong quá trình thực thi lệnh điều khiển. Kỹ thuật Extended Kalman Filter (EKF), dựa trên phân bố Gaussian, là kỹ thuật phổ biến đầu tiên để giải quyết vấn đề này. Phép lọc hạt (Particle Filter), dựa trên phân bố lưỡng kim (multimodal distributions), là cách thứ 2.

2.2.4 Kỹ thuật chung giải quyết bài toán SLAM

Một hệ thống SLAM hoàn chỉnh thông thường bao gồm nhiều tác vụ được thực hiện lặp đi lặp lại. Đầu tiên, tập hợp các điểm cột mốc (landmarks) cần được xác định từ môi trường thông qua các cảm biến, sau đó được xử lý trước khi thêm vào bản đồ. Khi robot di chuyển đến một vị trí mới, thông qua bước quan sát (Observation), dữ liệu quan sát được lúc này sẽ so trùng với tập dữ liệu đã quan sát trước đó, bao gồm các cột mốc giống nhau và thông tin từ các cột mốc mới, bước này gọi là liên kết dữ liệu (Data association). Nếu hệ thống phát hiện sai sót trong quá trình nhận diện cột mốc, hệ thống sẽ tiến hành định vị lại (relocalization) và khôi phục lại trạng thái. Sau đó, hệ thống sẽ thiết lập một mô hình ước lượng chuyển động (motion estimation). Dựa vào mô hình chuyển động để ước lượng mô hình quan sát, kết hợp với mô hình quan sát từ cảm biến, hệ thống sẽ loại bỏ những ước tính không chính xác, bước này gọi là tối ưu hóa (Optimization). Cuối cùng, nếu một phần nào đó của bản đồ được xây dựng bên trong hệ thống được quan sát lại, thuật toán đóng vòng lặp (Loop closure) được thực thi để tối ưu hóa hệ thống bao gồm những thông số đã thiết lập, và tối ưu bản đồ đã ghi nhận. Mã giả bên dưới mô tả chi tiết hơn về chuỗi các tác vụ sẽ được thực hiện trong một hệ thống SLAM

```

Repeat
    1. Observation
    2. Data association
    if tracking failure is detected then
        3. Relocalization
    4. Motion estimation
    5. Optimization
    if loop closure is detected then
        6. Loop closure correction

```

Hình 7: Mã giả của SLAM

- **Quan sát (Observation)**

Việc quan sát các cột xung quanh trong suốt quá trình nhận thức được thực hiện thông qua các biến cảm biến, phổ biến là Lidar và Camera RGB-D. Đối với bài toán sử dụng Lidar, dữ liệu quan sát được bao gồm tập hợp khoảng cách đến vật cản xung quanh trong một chu kỳ quét. Đối với bài toán sử dụng camera RGB-D, từ những khung hình thu được, hệ thống cần trích xuất đặc trưng và thông qua đó lựa chọn những điểm tiềm năng làm cột mốc.

- **Liên kết dữ liệu (Data association)**

Liên kết dữ liệu thiết lập mối quan hệ giữa dữ liệu quan sát được và cột mốc được thiết lập, giúp so khớp giữa những nhận thức không biết và những nhận thức đã ghi nhận. Có nhiều phương pháp sử dụng cho quá trình liên kết dữ liệu có thể kể đến là: RANSAC, JCBB, active search, active matching.

- **Tái định vị (Relocalization)**

Khi việc theo dõi các cột mốc bị lỗi, xảy ra khi robot di chuyển quá nhanh, hoặc những khung hình nhận được từ camera bị mờ, hệ thống robot cần thực hiện quá trình định vị lại. Theo đó, những tác vụ cần được thực hiện để phát hiện và khôi phục hệ thống khi gặp lỗi nhằm đảm bảo sự nguyên vẹn của bản đồ, không để những xử lí lỗi dẫn đến những lỗi cộng dồn.

- **Ước lượng chuyển động (Motion estimation)**

Dựa trên thông tin từ odometry, có thể dự đoán được sự thay đổi trong nhận thức về cùng một mốc giữa hai mốc thời gian (bao gồm cả encoder hoặc IMU). Thông tin từ odometry này có thể sử dụng để xây dựng một mô hình ước lượng tiên nghiệm của cột mốc và hiệp phương sai (covariance), là những thông tin cần thiết cho bước Optimization.

- **Tối ưu (Optimization)**

Các thuật toán SLAM truyền thống sử dụng phép tối ưu dựa trên đặc trưng của khung hình (Key-frame-based optimization) hoặc phép tối ưu dựa trên bộ lọc (Filtering-based optimization). Ví dụ gói Gmapping trong ROS sử dụng phép tối ưu dựa trên bộ lọc, trong khi gói RTAB-Map sử dụng phép tối ưu dựa trên đặc trưng khung hình.

Phương pháp tối ưu dựa trên đặc trưng của khung hình chia việc theo dõi vị trí robot với việc xây dựng bản đồ thành 2 tác vụ khác nhau. Khi số lượng cột mốc được phát hiện trong mỗi khung hình tăng lên, thì độ chính xác của việc ước lượng vị trí robot cũng tăng theo.

• Đóng vòng lặp (Loop closure)

Khi một vùng nào đó của bản đồ được quan sát lại, thuật toán SLAM cần phát hiện cột mốc tương ứng trong bản đồ đã ghi nhận. Tuy nhiên, trong một quỹ đạo di chuyển lớn ví dụ khi robot di chuyển xung quanh một nhà hàng hay khách sạn, trạng thái cột mốc trong 2 khu vực tương ứng trong bản đồ có thể không khớp, dẫn đến việc phát hiện trùng lặp sai do sai số trong những bước ước lượng các mô hình chuyển động. Phương pháp Image-to-image matching, sử dụng giải pháp so trùng hình ảnh và Scan-to-scan, sử dụng 2 tập hợp điểm thu được từ cảm biến lidar, sử dụng các phép dịch để kiểm tra đánh giá tập điểm hiện tại với tập điểm đã ghi nhận trước đó, là 2 giải pháp vấn đề này.

• Các giải pháp cảm biến được sử dụng trong SLAM

Định vị robot được xây dựng dựa trên các phép đo khoảng cách hay xác định vị trí của các điểm cột mốc trong môi trường làm việc của robot. Các cảm biến được sử dụng để đo khoảng cách thông dụng có thể kể tên là cảm biến laser, cảm biến siêu âm, cảm biến radar, cảm biến hồng ngoại hay có thể là camera có ảnh chiều sâu (RGB-D camera).

2.2.5 Các phương pháp giải SLAM phổ biến

Hiện nay, có rất nhiều thuật toán SLAM và cách tiếp cận SLAM, có thể kể đến như:

- Graph SLAM
- EFK SLAM
- Topological SLAM
- Visual SLAM
- 2D SLAM
- 3D SLAM

Và cũng có rất nhiều phương pháp hiện thực SLAM với mã nguồn mở được đóng gói thành các package trên ROS. Nhìn chung các phương pháp này có thể được chia thành 2 nhóm chính: SLAM dựa trên laser và SLAM dựa trên camera RGB-D.

- **LiDAR SLAM:** Hệ thống SLAM dựa trên LiDAR tạo bản đồ 3D về môi trường xung quanh bằng cảm biến laser. LiDAR (Light Detection and Ranging) xác định khoảng cách đến một vật (chẳng hạn như bức tường hoặc chân ghế) bằng cách chiếu sáng mục tiêu bằng một "xung" laser đang hoạt động.

LiDAR là một phương pháp nhanh chóng và chính xác có thể được sử dụng ở nhiều địa điểm và điều kiện khác nhau. Công nghệ này tạo ra đám mây điểm (point cloud) có độ chính xác cao, rất phù hợp để lập bản đồ trong xây dựng. Các phép đo khoảng cách có độ chính xác cao này cũng có thể được sử dụng cho nhiều mục đích khác nhau.

Một số giải thuật SLAM sử dụng laser (laser-based):

- Wildcat SLAM
- Hector SLAM
- Google Cartographer
- HDL graph SLAM
- LOAM

- **Visual SLAM:** còn được gọi là vSLAM, xác định vị trí và hướng của thiết bị liên quan đến môi trường xung quanh đồng thời lập bản đồ môi trường chỉ sử dụng duy nhất đầu vào trực quan từ camera. vSLAM dựa trên tính năng ghi lại các vị trí quan tâm trên các khung hình liên tiếp để lập tam giác vị trí 3D của máy ảnh, sau đó vị trí này được sử dụng để tạo bản đồ 3D.

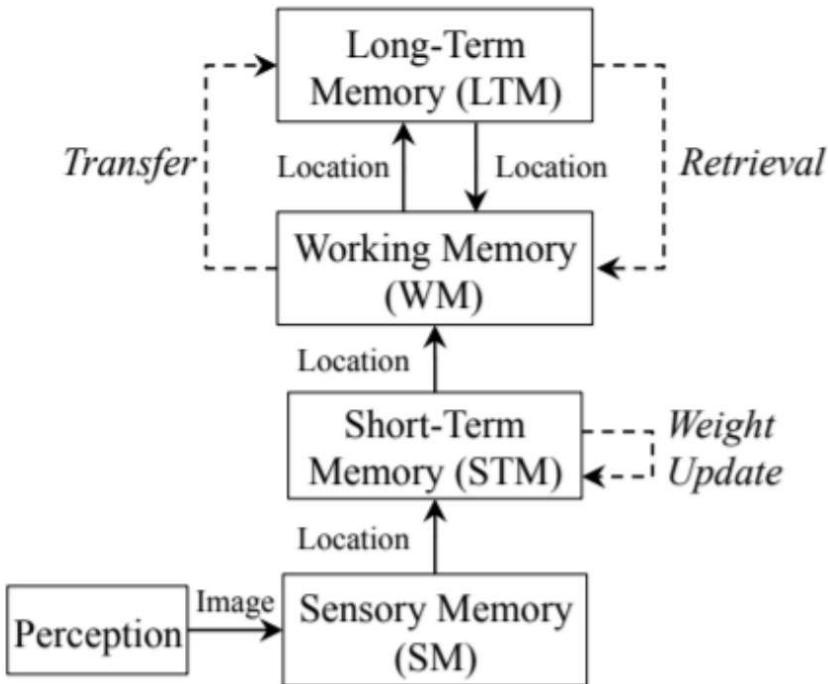
Một số giải thuật SLAM sử dụng visual (visual-based):

- ORB-SLAM
- DPPTAM
- ZEDfu
- RTAB-Map

2.2.6 RTAB-Map

- **Giới thiệu**

Phương pháp RTAB-Map, viết tắt của Real-Time Appearance-Based Mapping, tạm dịch là: Xây dựng bản đồ trong thời gian thực dựa vào phương pháp nhận diện hình ảnh, do tác giả Mathieu Labb đề xuất và xây dựng từ năm 2013. Với mục tiêu ban đầu là giải quyết bài toán vòng lặp đóng (Loop closure) tồn tại trong các giải pháp xử lý bài toán SLAM. Theo đó, tác giả sử dụng phương pháp so trùng ảnh (Image-to-image matching) để phát hiện Loop closure, đi kèm đó là cơ chế quản lý bộ nhớ hiệu quả nhằm đảm bảo thời gian thực hiện tính toán giải thuật luôn xấp xỉ thời gian cho phép khi robot được triển khai vận hành trong thời gian dài.

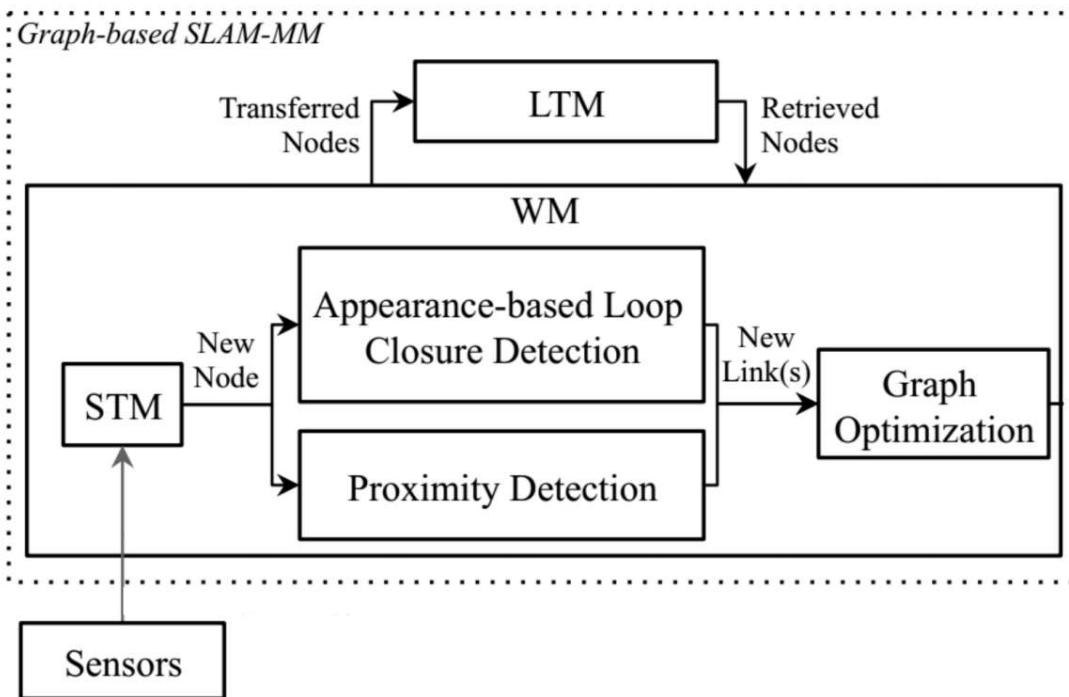


Hình 8: Mô hình quản lý bộ nhớ trong RTAB-Map [10]

Thư viện mã nguồn mở rtabmap_ros, cũng như kỹ thuật RTAB-Map, được sử dụng làm giải pháp SLAM chính trong đề tài này.

- **Quản lý bộ nhớ**

Một trong những vấn đề mà các thuật toán SLAM sử dụng thị giác máy tính gặp phải đó là quản lý bộ nhớ, cũng bởi theo thời gian dài hoạt động, chương trình phải liên tục ghi nhận hình ảnh từ môi trường và lưu giữ nó, sử dụng nó về sau để phát hiện việc ghé thăm lại một vị trí nào đó mỗi khi ghi nhận một vị trí mới trên bản đồ, để từ đó tiến hành hiệu chỉnh và tối ưu. Lý do này khiến cho nhiều giải thuật tỏ ra chậm chạp sau thời gian dài vận hành robot. Để giải quyết vấn đề này, phương pháp RTAB-Map đã đề xuất giải pháp quản lý bộ nhớ thông minh mà theo đó sẽ luôn đảm bảo thời gian tính toán ở mỗi vòng lặp tính toán.



Hình 9: Mô hình quản lý bộ nhớ trong RTAB-Map [10]

Giải thuật RTAB-Map dựa trên xử lý trên đồ thị, bao gồm các nút (node) và liên kết (link) giữa các node, trong đó mỗi node lưu các thông tin sau:

- **ID**: Mỗi node được gán một ID duy nhất.
- **Weight**: Trọng số xác định tầm quan trọng của node, được sử dụng trong việc quản lý bộ nhớ.
- **Bag-of-words (BOW)**: Các đặc trưng đã được số hóa, được sử dụng để đánh giá vòng lặp.
- **Sensor data**: ảnh màu (RGB image) được sử dụng để trích xuất đặc trưng; ảnh chiều sâu (Depth image) sử dụng trong việc xác định vị trí của ảnh trong không gian 3D; Vị trí (Pose) được tính toán từ odometry của hệ thống; Dữ liệu từ cảm biến lidar, được sử dụng để hiệu chỉnh odometry và phát hiện xấp xỉ (Proximity Detection).

Tiếp theo là liên kết của các node:

- **Liên kết kề nhau (Neighbor link)**: liên kết được tạo ra giữa node mới và node ngay trước nó.
- **Liên kết lặp kín (Loop closure link)**: liên kết được tạo ra khi phát hiện vòng lặp kín giữa node mới thêm vào và node đã có trên bản đồ.
- **Liên kết xấp xỉ (Proximity link)**: liên kết được tạo ra khi 2 node gần nhau có dữ liệu laser scan xấp xỉ nhau.

Chương trình xử lý sử dụng 2 vùng nhớ: vùng nhớ xử lý chính (working memory) và vùng nhớ dài hạn (long term memory), trong đó, mọi quá trình

xử lý, tính toán đều sử dụng các node ở vùng nhớ xử lý chính. Các node thường trực ở vùng nhớ xử lý chính sẽ cập nhật, thay đổi tùy vào tác vụ tính toán giải thuật, khi thời gian tính toán vượt ngưỡng cho phép, chương trình sẽ tiến hành chọn lọc những node có ít khả năng phát hiện sự trùng lặp trong bộ nhớ xử lý chính và chuyển sang bộ nhớ dài hạn. Ngược lại, trong quá trình xử lý nếu hệ thống phát hiện vòng lặp kín (loop closure) và xấp xỉ (proximity) giữa nút mới vừa tạo và nút có trong vùng nhớ xử lý chính, sau khi gắn liên kết giữa 2 nút này, chương trình sẽ duyệt những nút có bất kỳ liên kết nào với nút vừa được phát hiện trùng lặp vào vùng nhớ xử lý chính để sử dụng cho việc phát hiện loop closure ở các vòng lặp tiếp theo.

Có 2 tiêu chí để lựa chọn node lưu tại vùng nhớ xử lý chính:

- Dựa vào cơ chế ghi nhớ của con người, đó là duy trì những node thường xuyên được ghé thăm và những node được ghé thăm gần đây.
- Giữ lại những node nằm trên đường đi trong quá trình hoạch định đường đi đến điểm đích.

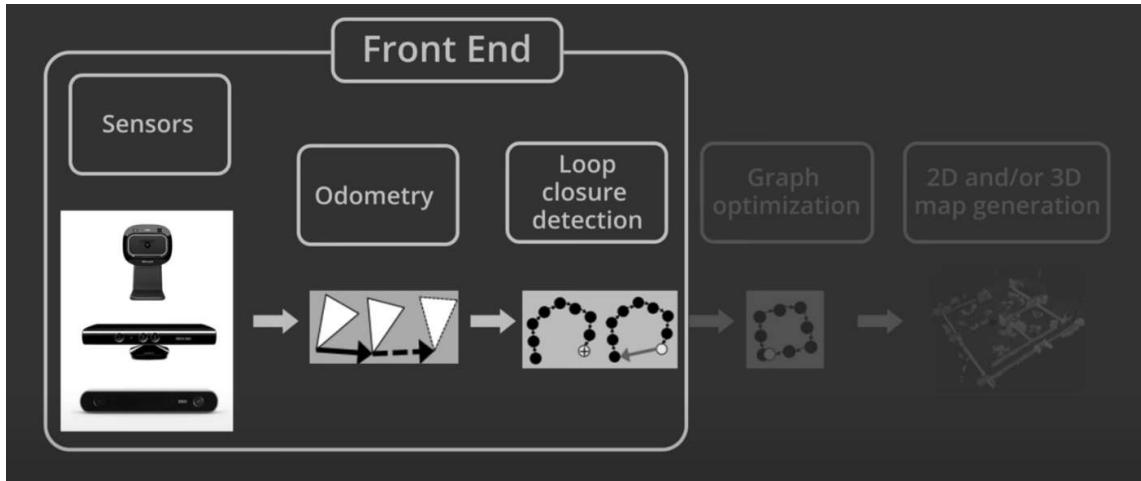
Lựa chọn các node nằm lại vùng nhớ xử lý chính, là những node có khả năng cao trong việc phát hiện vòng lặp kín và phát hiện node xấp xỉ. Cũng bởi càng phát hiện tốt sự ghé thăm lại những node đã từng đi qua, sẽ giúp cho hệ thống hiệu chỉnh tốt bản đồ mà robot đã xây dựng, và vị trí robot. Cứ mỗi khi phát hiện vòng lặp kín hoặc xấp xỉ, chương trình sẽ tiến hành thực hiện bước tối ưu (Optimization).

Sau mỗi chu kì hoạt động (mặc định là 1 giây) chương trình sẽ tạo 1 nút trên đồ thị, nút này được gán cho 1 ID duy nhất, trọng số khởi tạo là 0, cùng với đó là thông tin vị trí từ Odometry, hình ảnh từ RGB-D Camera, dữ liệu scan từ lidar, nếu node mới vừa tạo có hình ảnh thu về tương tự như hình ảnh của node kè trước (robot không di chuyển), nút mới sẽ bị xóa. Trong trường hợp ngược lại, nút mới sẽ tạo liên kết liền kề với nút kè trước đó và trọng số của nút kè trước được tăng lên 1 đơn vị. Sau đó, nút mới được đưa vào vùng nhớ ngắn hạn (short term memory). Khi số lượng nút ở vùng nhớ ngắn hạn đạt ngưỡng cho phép (ngưỡng này có thể thay đổi được, tùy vào khả năng tính toán và tốc độ di chuyển của robot), nút ở trong vùng nhớ ngắn hạn lâu nhất sẽ được đưa vào vùng nhớ xử lý chính, theo cơ chế FIFO. Sở dĩ phải sử dụng kỹ thuật buffering để lưu lại những nút trong vùng nhớ ngắn hạn như thế là để chương trình xử lý loop closure tránh sử dụng những nút vừa mới ghé thăm để phát hiện lặp kín, cũng bởi như thế là quá sớm và không có ý nghĩa trong việc tối ưu.

• Visual Odometry và Lidar Odometry

Dữ liệu cảm biến được sử dụng để xác định các ràng buộc trong các phương pháp tối ưu hóa là điểm nhấn phần Front-end của RTAB-Map. Trong RTAB-Map, không có giới hạn về số cột mốc. Chỉ có giới hạn về odometry và loop closure. Tất cả các bộ mã hóa bánh xe, IMU, LiDAR và phép đo đường

quang học đều có thể cung cấp các hạn chế về odometry. Các tính năng 2D như Speeded Up Rous Features (SURF) được sử dụng để thực hiện đo đường trực quan (visual odometry).



Hình 10: Front-end của RTAB-Map [10]

RTAB-Map dựa trên hình thức bên ngoài và không bao gồm thông tin về chỉ số khoảng cách. Để xác định việc đóng vòng lặp, RTAB-Map có thể sử dụng một máy ảnh một mắt (Monocular camera). Để tính toán ràng buộc hình học giữa các hình ảnh đóng vòng lặp cho các chỉ số GraphSLAM, RTAB-Map cần một máy ảnh RGB-D hoặc máy ảnh âm thanh nổi (Stereo camera). Ràng buộc hình học này cũng có thể được tính chỉnh bằng cách sử dụng laser. Quản lý đồ thị, bao gồm hình thành node và phát hiện loop closure bằng cách sử dụng bag-of-words, cũng là một phần của phần Front-end.

Lidar odometry, không giống như visual odometry, không thể khôi phục khi bị mất khi dự đoán chuyển động không có giá trị, dẫn đến độ chính xác đáng kể của phương pháp đo đường. Sau đó, phép đo odometry của lidar phải được đặt lại.

• Đóng vòng lặp (Loop closure)

Quá trình so khớp giữa các địa điểm hiện tại và các địa điểm đã ghé thăm trước đây trong SLAM được gọi là đóng vòng lặp (Loop closure). Local và global loop closure là hai hình thức phát hiện đóng vòng lặp.

Sự trùng hợp giữa một quan sát mới và một khu vực bản đồ được xác định trong các lần đóng vòng lặp cục bộ (local). Sự không chắc chắn liên quan đến vị trí của robot xác định kích thước và vị trí của khu vực bản đồ giới hạn này. Nếu vị trí ước tính không chính xác, phương pháp này không thành công.

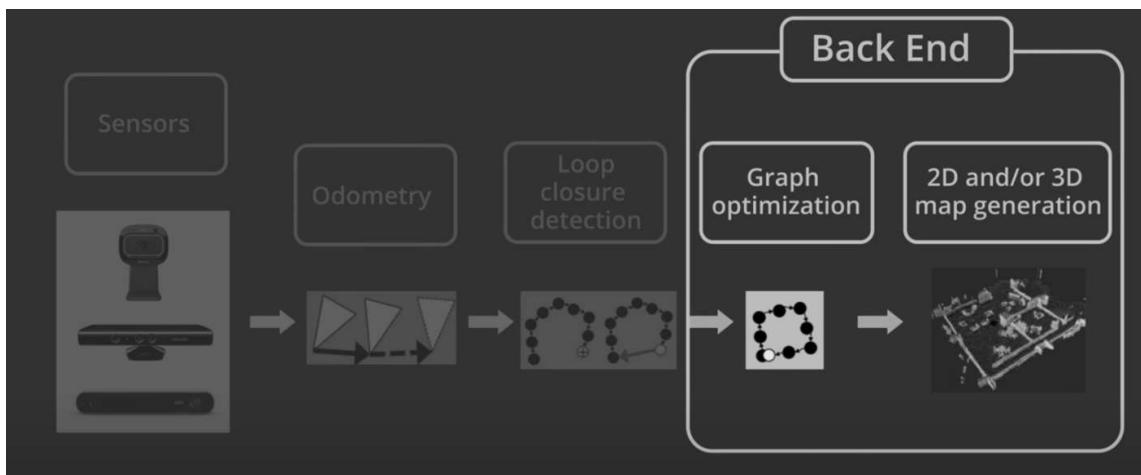
Một địa điểm mới được so sánh với các địa điểm được quan sát trước đó trong phương pháp đóng vòng lặp toàn cầu (global). Nếu không khớp, vị trí mới sẽ được lưu vào bộ nhớ. Khi robot di chuyển và bản đồ mở rộng, thời gian cần

thiết để so sánh các địa điểm mới với những địa điểm đã ghé thăm trước đó sẽ tăng tuyến tính. Bản đồ sẽ không thành công nếu thời gian tìm kiếm và so sánh hình ảnh mới với những hình ảnh đã được lưu trữ trong bộ nhớ vượt quá thời gian cho phép.

Để đảm bảo rằng quy trình đóng vòng lặp diễn ra trong thời gian thực (real time), RTAB-Map sử dụng quy trình đóng vòng lặp toàn cầu cũng như các quy trình khác.

- **Tối ưu hóa đồ thị, lắp ráp và xây dựng bản đồ**

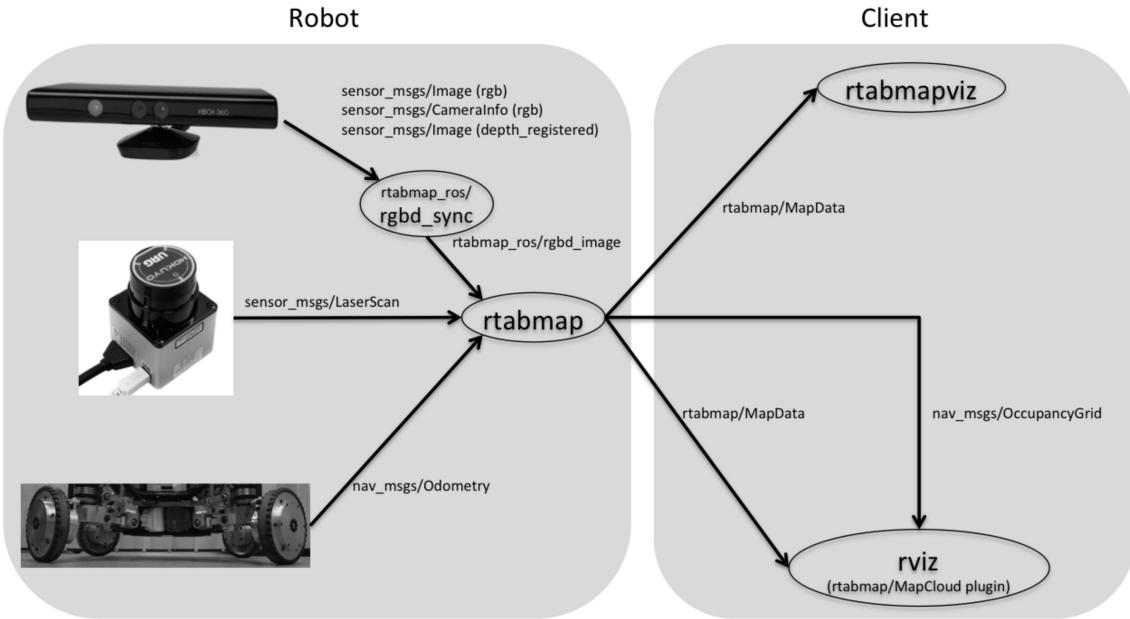
Back-end của RTAB-Map là một phần của tối ưu hóa đồ thị và xây dựng một occupancy grid từ dữ liệu của đồ thị.



Hình 11: Back-end của RTAB-Map [10]

- **Áp dụng RTAB-Map cho ứng dụng**

Với những ưu điểm đã trình bày, mô hình robot dịch vụ tự hành này đã chọn RTAB-Map làm phương pháp giải SLAM, trong đó hai cảm biến Orbbec Astra và Lidar được lựa chọn để bổ sung và hiệu chỉnh cho nhau trong quá trình hoạt động.



Hình 12: Thiết lập các thành phần triển khai RTAB-Map lên robot [3]

2.2.7 Hệ điều hành cho Robot (Robot Operating System - ROS)

- **Tổng quan về ROS**

Theo ROS Wiki, ROS (Hệ điều hành cho robot) cung cấp các thư viện và các công cụ để giúp các nhà phát triển phần mềm tạo ra các ứng dụng cho robot. Nó cung cấp phần cứng trừu tượng, trình điều khiển thiết bị, thư viện, hiển thị trực quan, tin nhắn giao nhận, quản lý gói, và nhiều hơn nữa. ROS được cấp phép theo giấy phép nguồn mở BSD.

Mặc dù ROS không phải là một hệ điều hành thực thụ nhưng nó cung cấp các dịch vụ quan trọng của một hệ điều hành. Bản thân ROS cũng không phải là một hệ điều hành thời gian thực nhưng ta vẫn có thể tích hợp nó với các chương trình thời gian thực khác hoặc kết nối với vi điều khiển một cách dễ dàng bằng giao thức Rosserial. Ngoài ra ROS 2.0 cũng đang được phát triển, hứa hẹn sẽ giải quyết vấn đề này một cách triệt để.

ROS cung cấp khả năng hoạt động tốt giữa nhiều nền tảng khác nhau: tin nhắn (messages) trong ROS giúp người dùng gửi dữ liệu giữa các nền tảng phần cứng khác nhau với các ngôn ngữ khác nhau; Từ những máy tính nhúng chạy ngôn ngữ C để tối ưu hiệu năng đến những máy chủ cấp cao với python, java. Hơn nữa ROS giúp người dùng dễ dàng hơn với cơ chế public và subscribe thông qua một topic. Nhờ những lợi ích được kể trên cộng với việc ROS có mã nguồn mở, cộng đồng sử dụng ROS đang ngày càng phát triển rất mạnh mẽ.

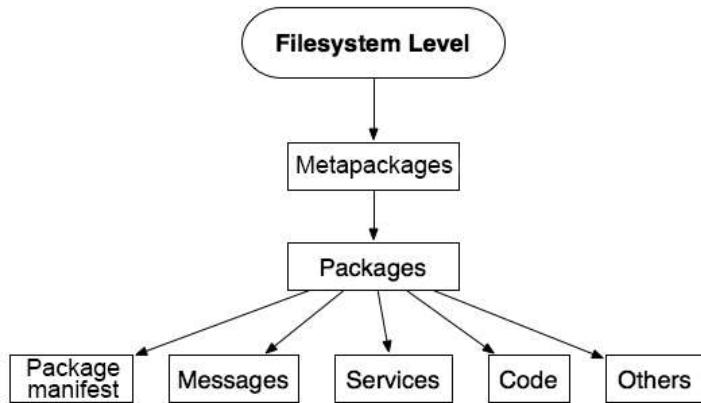
ROS hiện tại chủ yếu hỗ trợ trên hệ điều hành Ubuntu nhưng cũng đang được xây dựng để có thể hỗ trợ nhiều loại hệ điều hành khác. Trung bình mỗi năm,

ROS sẽ phát hành một bản phân phối ROS (ROS Distribution) để cập nhật phần lỗi, công cụ tiện ích và các thư viện.

• Cấu trúc ROS

ROS có 3 khái niệm chính:

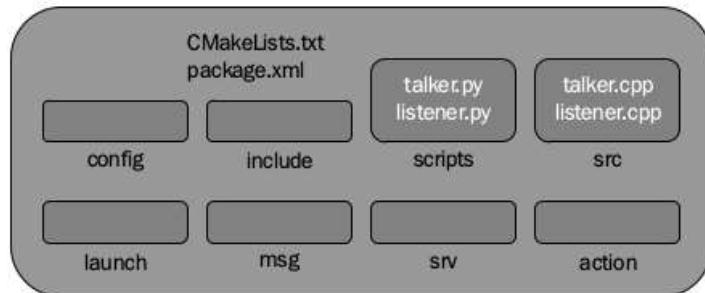
- **Cấp thứ nhất - Filesystem:** chứa cấu trúc thư mục và các tập tin tối thiểu để ROS hoạt động. Nó chủ yếu là các tài nguyên của ROS và được thực hiện trên đĩa cứng.



Hình 13: Minh họa hệ thống tập tin của ROS [12]

ROS quản lý tập tin bằng cách chia một chương trình thành nhiều thư mục, mỗi thư mục có một chức năng riêng. Ta có thể xét một vài thư mục chính sau đây:

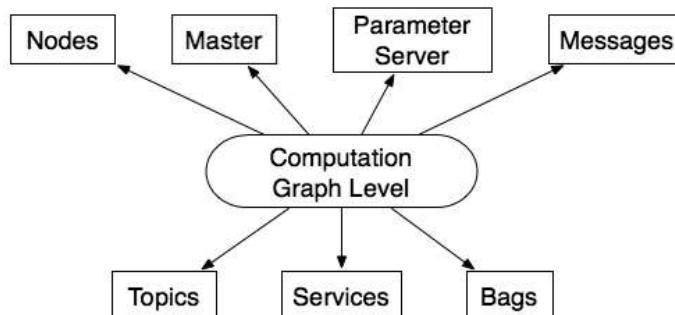
- * **Packages:** Cấp độ cơ bản nhất của ROS. Một package là một cấu trúc nhỏ nhất chứa một chương trình của ROS. Nó có thể chứa tiến trình ROS (node), tập tin cấu hình, ...



Hình 14: Packages trong ROS [12]

- * **Manifest:** Cung cấp thông tin về package, dependency, cờ biên dịch, tác giả, giấy phép, ... Tập tin Package.xml trong mỗi package là tập tin bảng kê khai của package đó.
- * **Stack:** Stack là tập hợp của nhiều package. Mỗi stack thường hỗ trợ một chức năng duy nhất: ví dụ Navigation Stack với chức năng điều hướng.

- * **Messages:** Thông tin mà một node gửi cho node khác. Người dùng có thể tự tạo một message trong thư mục **msg**. Phần mở rộng của message là “**.msg**”.
- * **Services:** Đây là một cách khác để giao tiếp giữa hai node. Trong khi message là phương pháp giao tiếp không đồng bộ, service cung cấp cách giao tiếp đồng bộ theo kiểu request/reply. Service thường được dùng để yêu cầu một node làm một việc cố định đã được định trước. Tương tự message, service do người dùng định nghĩa cũng được đặt trong một thư mục với tên **srv** và có phần mở rộng là “**.srv**”.
- **Cấp thứ hai - Computation Graph:** nơi giao tiếp giữa các process và hệ thống. Trong cấp này, ta phải thiết lập hệ thống, quản lý các tiến trình, giao tiếp giữa nhiều máy tính với nhau, ...



Hình 15: Computation Graph Level [12]

ROS tạo ra một mạng nơi tất cả những node được kết nối với nhau. Mỗi một node đều có thể dễ dàng truy cập vào mạng, giao tiếp với các node khác, nhìn thấy thông tin mà các node khác gửi và nhận trong mạng. Những khái niệm cơ bản nhất của ROS là node, master, parameter server, messages, services, topics và bags. Mỗi loại cung cấp dữ liệu cho mô hình theo một cách khác nhau:

- * **Node:** Là một tiến trình nơi những tác vụ tính toán, điều khiển được thực thi. Một Node khi được khởi tạo phải đăng ký một tên duy nhất trong hệ thống thông qua Master. Node được viết với các thư viện của ROS như roscpp (C++), rospy (Python). Sử dụng API của các bộ thư viện này, người dùng có thể dễ dàng giao tiếp giữa các node bằng nhiều cách khác nhau, phổ biến nhất là gửi Messages qua các Topic. Một chương trình cho robot sẽ có rất nhiều Node thực thi nhiều công việc khác nhau và mỗi Node thường sẽ đảm nhiệm một nhiệm vụ nhất định. Điều này sẽ làm cho các chương trình lớn trở nên đơn giản hơn cũng như sẽ khiến cho hệ thống dễ dàng mở rộng, bảo trì và sửa chữa.
- * **Master:** Cho phép các Node đăng ký tên và tra cứu tên của các Node khác. Nếu không có Master trong hệ thống, ta không thể thực hiện được các tác vụ của ROS như giao tiếp với các Node, gửi nhận Service, ... Tuy nhiên Master có thể ở một máy khác với máy chạy Node.

- * **Parameter Server:** Cho phép lưu Parameter trong một vị trí trung tâm. Tất cả các Node đều có thể truy cập và chỉnh sửa các Parameter này. Thông qua những Parameter này, người dùng có thể cấu hình, sửa đổi một Node ngay cả khi nó đang được thực thi mà không cần phải dừng nó lại. Parameter Server là một phần của Master.
 - * **Messages:** các Node giao tiếp với nhau thông qua Messages. Một Message là một cấu trúc dữ liệu, bao gồm các trường được định nghĩa như integer, floating point, boolean, ... Người dùng cũng có thể tự tạo riêng một Message của riêng mình từ các cấu trúc chuẩn này.
 - * **Services:** Mô hình publish/subscribe thì rất linh hoạt trong việc giao tiếp nhưng đặc điểm là chỉ truyền được đa đối tượng và một chiều. Nhưng đôi khi lại không thích hợp cho việc truyền theo dạng request/reply, như yêu cầu robot làm một việc nhất định và đợi kết quả. Do đó, việc truyền nhận theo dạng request/reply được dùng thông qua Services. Service được định nghĩa một cặp cấu trúc dữ liệu: một cho request và một cho reply. Một Node cung cấp một Service thông qua một thuộc tính Name, và một client sử dụng Service bằng việc gửi một Request message và đợi phản hồi.
 - * **Topics:** Khi một Node gửi hoặc nhận Messages, nó phải gửi hoặc nhận Messages đó thông qua một qua Topics và Master sẽ là người phân phát các Messages đã được gửi đó. Khi một Node gửi một Message thông qua một topic, ta gọi node đó đang Publish một Topic. Ngược lại, khi Node nhận một Message thông qua một topic, ta gọi Node đó đang Subscribe topic đó. Topic được tạo ra để tách rời bên gửi và bên nhận, do đó Node gửi và Node nhận có thể không biết được sự tồn tại của nhau. Mỗi topic phải có một tên duy nhất.
 - * **Bags:** Một định dạng để lưu trữ và phát lại dữ liệu Message trong ROS. Bags là một cơ chế quan trọng cho việc lưu trữ dữ liệu; ví dụ như giá trị của các cảm biến rất khó để thu thập cho sự nghiên cứu, phát triển và kiểm tra thuật toán. Vì thế việc dùng Bags là rất quan trọng trong việc phát triển robot, đặc biệt là những robot mang tính phức tạp cao.
- **Cấp thứ ba - Community:** Đây là nơi các nhà phát triển có thể chia sẻ kiến thức, hướng dẫn, thuật toán chương trình cho cộng đồng. Đây là cấp độ rất quan trọng vì nó ảnh hưởng đến sự phát triển của cộng đồng ROS. Có thể kể đến như sau:
- * **Distributions:** Tương tự như các bản phân phối của Ubuntu, các bản phân phối của ROS là một tập hợp những phiên bản của ROS mà người dùng có thể cài đặt. Những bản phân phối này vẫn liên tục được bảo trì, phát triển và cập nhật hàng năm.
 - * **Repositories:** Để tối đa hóa sự tham gia của cộng đồng, những nhà phát triển tạo khuyến khích lưu trữ mã nguồn vào một kho chung. Đây là nguồn tài nguyên lớn và hữu ích, và cũng là một trong những lý do khiến ROS lớn mạnh như vậy.
 - * **ROS Wiki:** Bách khoa toàn thư của ROS, gồm nhiều tài liệu về ROS.

Tại đây bạn có thể tìm bất cứ tài liệu nào mình cần và bất cứ ai cũng có thể chia sẻ tài liệu, cung cấp các bản cập nhật, viết các bài hướng dẫn.

2.2.8 Gazebo - Công cụ mô phỏng Robot

Gazebo là một công cụ mô phỏng động 3D với khả năng mô phỏng hiệu quả và chính xác các robot trong môi trường phức tạp cả trong nhà và ngoài trời. Mặc dù Gazebo tương tự như một trò chơi, nó cung cấp mô phỏng vật lý ở mức độ trung thực cao hơn nhiều, một bộ cảm biến và giao diện cho cả người dùng và chương trình.

Một số tính năng chính của Gazebo:

- Rất nhiều tương tác vật lý.
- Có một thư viện các mô hình vật thể, môi trường rất phong phú.
- Hỗ trợ rất nhiều loại cảm biến.
- Giao diện lập trình và đồ họa thuận tiện cho người dùng.

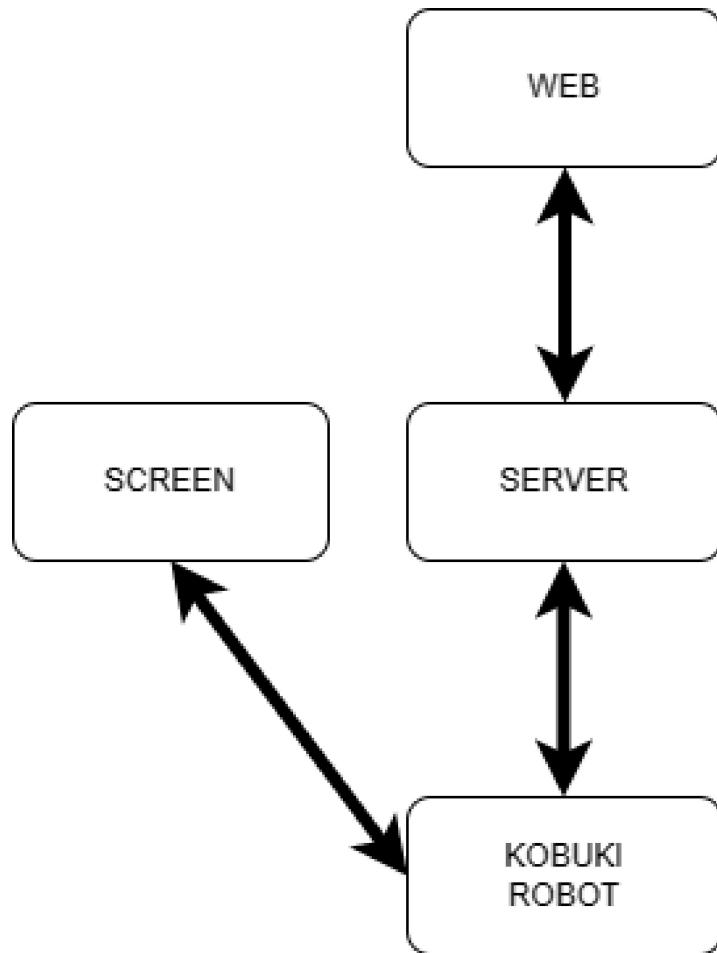
Các ứng dụng điển hình của Gazebo có thể kể đến như:

- Thử nghiệm các thuật toán về Robot.
- Thiết kế Robot.
- Thực hiện kiểm tra hoạt động của Robot với các kịch bản thực tế.

3 Thiết kế và hiện thực hệ thống

3.1 Kiến trúc mô hình hệ thống/giải pháp

3.1.1 Kiến trúc chung



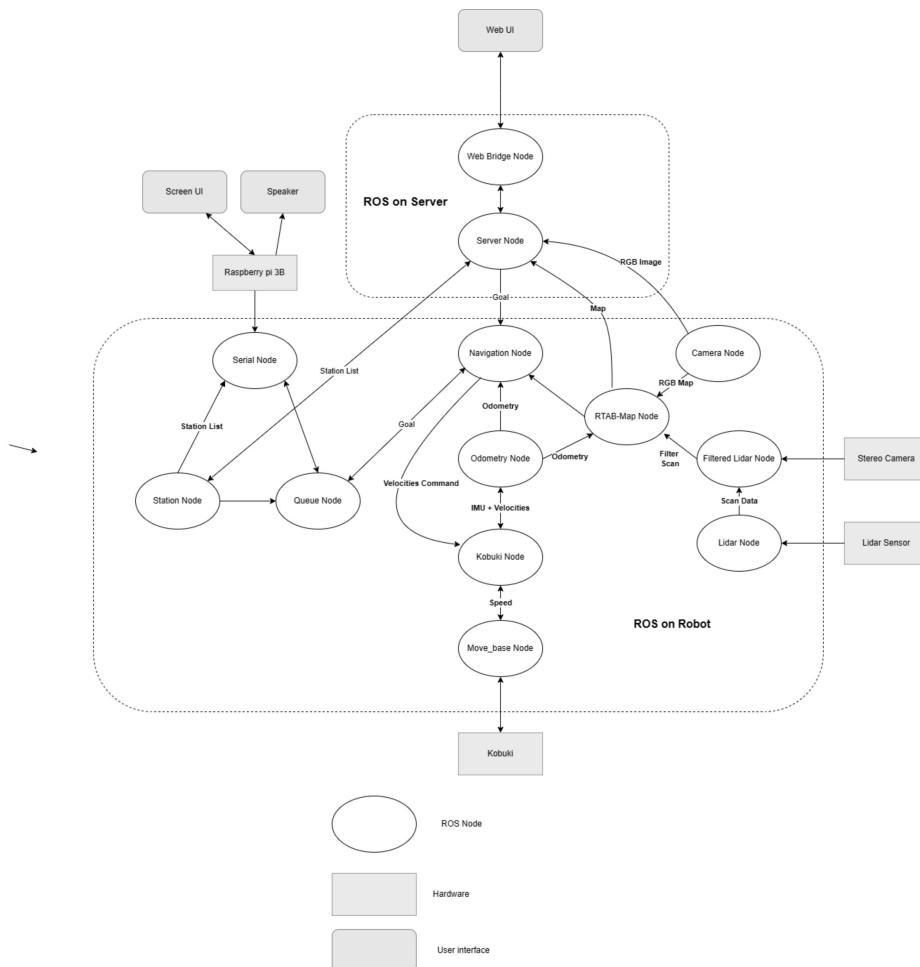
Hình 16: Kiến trúc chung của hệ thống

- **Khối Web:** Đối với người dùng quản lí thì đây là nơi cập nhật thông tin trạng thái của robot , bao gồm: vị trí robot, tốc độ hiện tại của robot, vị trí vật cản xung quanh, camera từ robot quan sát được, thông số CPU, RAM của board mạch chủ trên robot; giám sát quá trình robot xây dựng bản đồ; điều khiển robot theo các hướng và tốc độ tùy chỉnh; thiết lập các thông số của robot; Đặt tên cho một số vị trí cố định trên bản đồ đã xây dựng; chỉ định vị trí bất kỳ hoặc vị trí đã đặt tên trên bản đồ để robot tự hành đến vị trí này.
- **Khối Server:** Quản lý robot, là cầu nối để người dùng có thể tương tác với robot, lưu giữ các thiết lập của các robot và là nơi host web server để người dùng có thể truy cập vào.
- **Khối Kobuki:** Là thành phần quan trọng nhất của toàn hệ thống, đảm nhận

vai trò: xây dựng bản đồ; truyền hình ảnh trực tiếp từ camera cũng như vị trí hiện tại trên bản đồ về cho người dùng; tự hành đến đích do người dùng chỉ định, tránh vật cản trong quá trình vận hành.

- **Khối Sreen:** Đây là nơi robot và người dùng giao tiếp trực tiếp thông qua màn hình và loa được nối trực tiếp vào khói **Kobuki**. Người dùng sẽ dùng khói này để khởi động Robot, giao các Task và robot sẽ thông báo cho người dùng khi đến nơi bằng ăm thanh và chờ người dùng xác nhận để tiếp tục thực hiện nhiệm vụ.

3.1.2 Sơ đồ các Node ROS



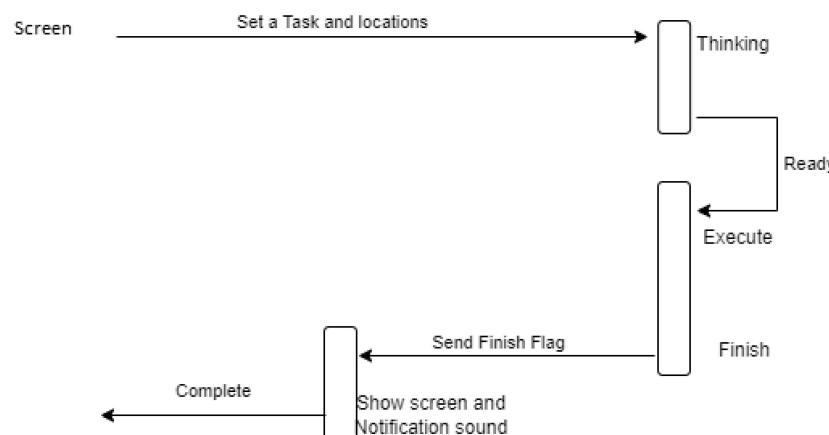
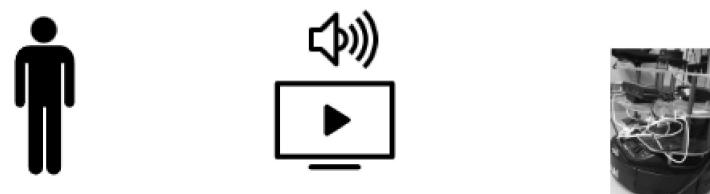
Hình 17: Sơ đồ các Node ROS trong hệ thống

Ứng dụng được xây dựng dựa trên nền tảng ROS, nên các thành phần chức năng của robot được xây dựng và chia nhỏ thành các nốt (node) chức năng hoạt động độc lập với nhau, nơi đây đảm nhiệm một vai trò cụ thể và giao tiếp với nhau qua việc đăng ký (subscribe) và phát đi (publish) các chủ đề (topic) với nội dung tin nhắn (message) có cấu trúc đã được thống nhất với nhau giữa 2 node.

3.1.3 Các chức năng cho người dùng

3.1.3.1 Người dùng Phổ thông

- **Yêu cầu robot tự hành đến điểm cố định:** Vị trí được yêu cầu có thể đã được đánh dấu từ trước. Sau khi nhận được tọa độ cần đến, robot sẽ tự hành để đến vị trí này, trong quá trình di chuyển robot vẫn sẽ cập nhật tọa độ của mình và kế hoạch di chuyển mà robot đã định ra lên bản đồ, khi đã đến được đích, robot sẽ phản hồi về người dùng thông qua WebApp hoặc màn hình được gắn trên Robot.
- **Giao một hoặc nhiều nhiệm vụ di chuyển cho robot:** Người dùng có thể lựa chọn thêm các Task mà mỗi Task là một điểm di chuyển mà robot cần tự hành đến.
- **Xem danh sách các Task:** Người dùng có thể xem danh sách các Task robot cần xử lý và trạng thái của các Task đó.



Hình 18: Sơ đồ hoạt động của người dùng

Hiện tại người dùng có thể ra lệnh trực tiếp với robot để yêu cầu robot di chuyển đến một vị trí cố định đã được đặt tên trên bản đồ thông qua màn hình trên robot. Ngoài ra, màn hình còn có thể hiện một số thông tin như danh sách các Task,... Hình trên thể hiện sơ đồ quá trình ra tạo Task bằng màn hình và nhận được thông báo trên màn hình trên robot khi robot đã di chuyển đến nơi.

3.1.3.2 Người dùng Quản lý

- Tất cả chức năng của người dùng phổ thông.
- **Chỉnh sửa các Task:** Trong trường hợp cần hủy Task hoặc trường hợp khẩn cấp (như robot bị hư mạch di chuyển) thì người quản lý có thể bắt buộc dừng di chuyển robot.
- **Điều khiển robot di chuyển:** Người dùng có thể điều khiển robot di chuyển theo các hướng và vận tốc một cách linh hoạt từ giao diện web.
- **Quản lý bản đồ được tạo và thêm hoặc xóa các trạm trên bản đồ:** Người dùng có thể quan sát bản đồ 2D mà robot đã xây dựng được và cập nhật các vị trí có trên bản đồ, các vị trí này có thể được đặt tên và sử dụng nó để yêu cầu robot di chuyển đến.
- **Quan sát hình ảnh từ Camera:** Người dùng có thể quan sát hình ảnh từ cả 2 camera.
- **Tùy chỉnh thông số di chuyển và quan sát trạng thái của robot:** Người quản lý có thể tùy chỉnh vận tốc di chuyển (đi thẳng hoặc xoay góc) của robot cũng như có thể theo dõi các trạng thái về CPU, RAM và Pin của Robot.

3.2 Kết quả hiện thực

3.2.1 Mô hình robot mô phỏng trên Gazebo

3.2.1.1 Xây dựng mô hình robot

Đầu tiên, để xây dựng mô hình Kobuki trên Gazebo ta có thể sử dụng package **kobuki_desktop** [13]. Đây là một package miễn phí có sẵn trên Github, nó sẽ khởi tạo mô hình robot Kobuki tương tự thực tế. Ta tải package và bỏ vào thư mục **catkin_ws/src** sau đó tiến hành build. Sau khi build thành công, ta khởi chạy nhanh Gazebo với robot Kobuki bằng câu lệnh sau:

```
1 $ roslaunch kobuki_gazebo kobuki_empty_world.launch
```

Tuy nhiên, package trên chỉ tạo ra mô hình Kobuki cơ bản trong khi thực tế robot của chúng ta còn các thiết bị khác như camera và lidar. Để robot Kobuki trên Gazebo giống với thực tế nhất, ta phải thêm 2 camera và 1 lidar. Ta sẽ chỉnh sửa file mô tả của Kobuki để thêm các thiết bị này vào.

Tại file **kobuki.urdf.xacro** trong thư mục **Kobuki_description/urdf** ta định nghĩa các thiết bị và liên kết (link) chúng với các phần khác của Kobuki với các thông số :

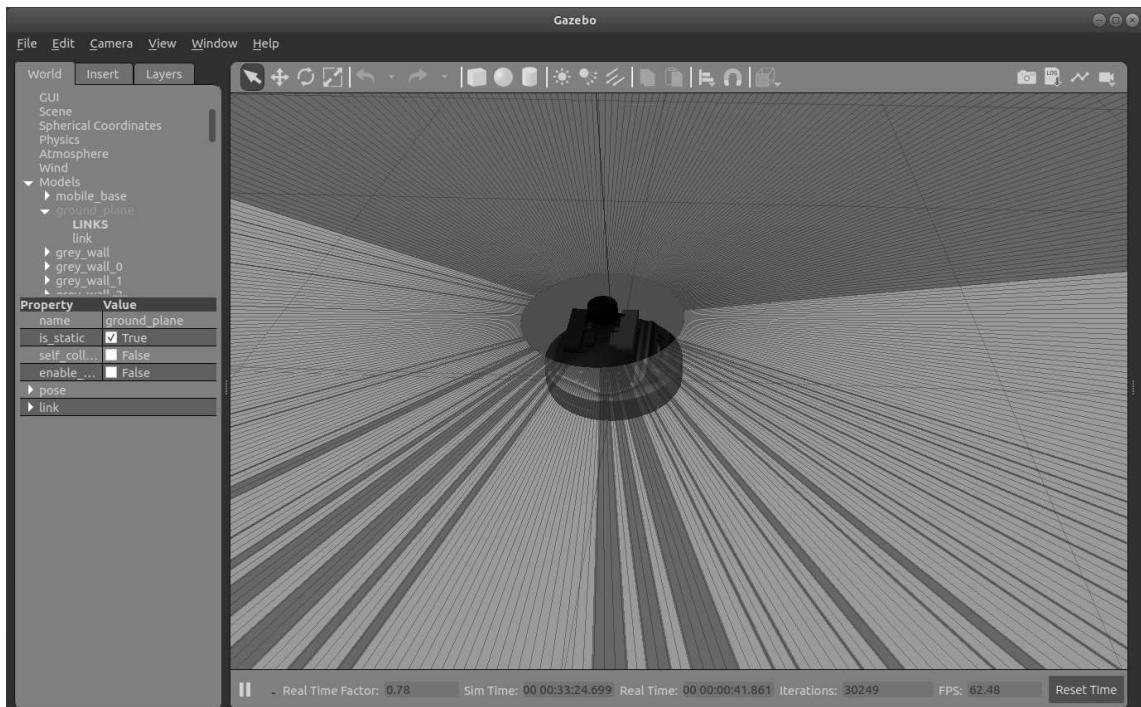
- **Astra Camera**
 - Tên: camera1_link
 - Vị trí
 - Loại hình ảnh thu được: camera_rgb_frame và camera_depth_frame

- RpLidar

- Tên: rplidar_joint
- Chức năng: Laser

Sau đó trong file **kobuki_gazebo.urdf.xacro**, ta định nghĩa tên các topic của camera và lidar để nó hoạt động giống thực tế.

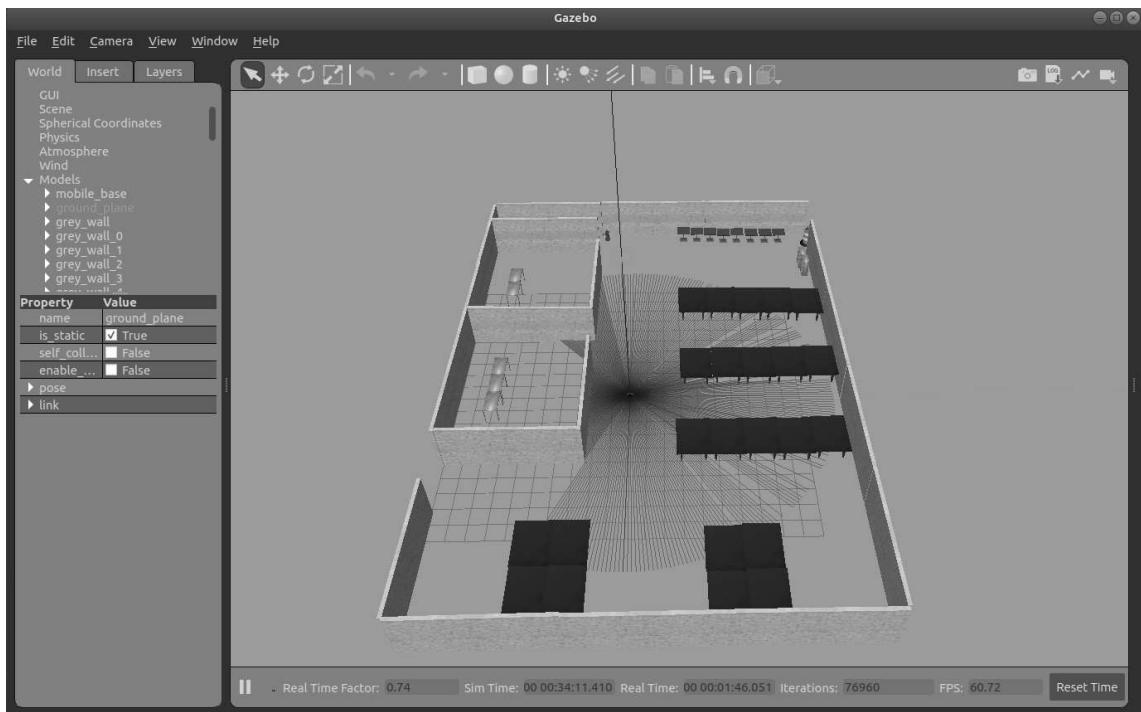
Cuối cùng, ta sẽ thu được một mô hình robot Kobuki trên Gazebo tương tự với thực tế, gồm 2 Camera Astra và 1 RPLidar.



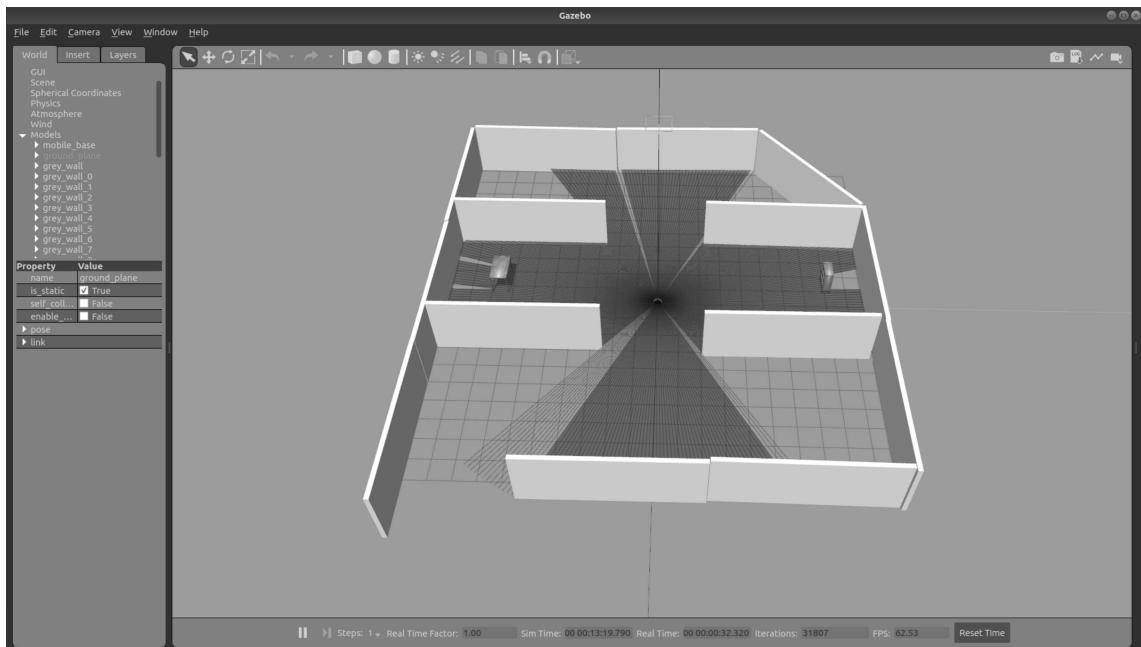
Hình 19: Mô hình robot mô phỏng trên gazebo

3.2.1.2 Xây dựng khu vực mô phỏng

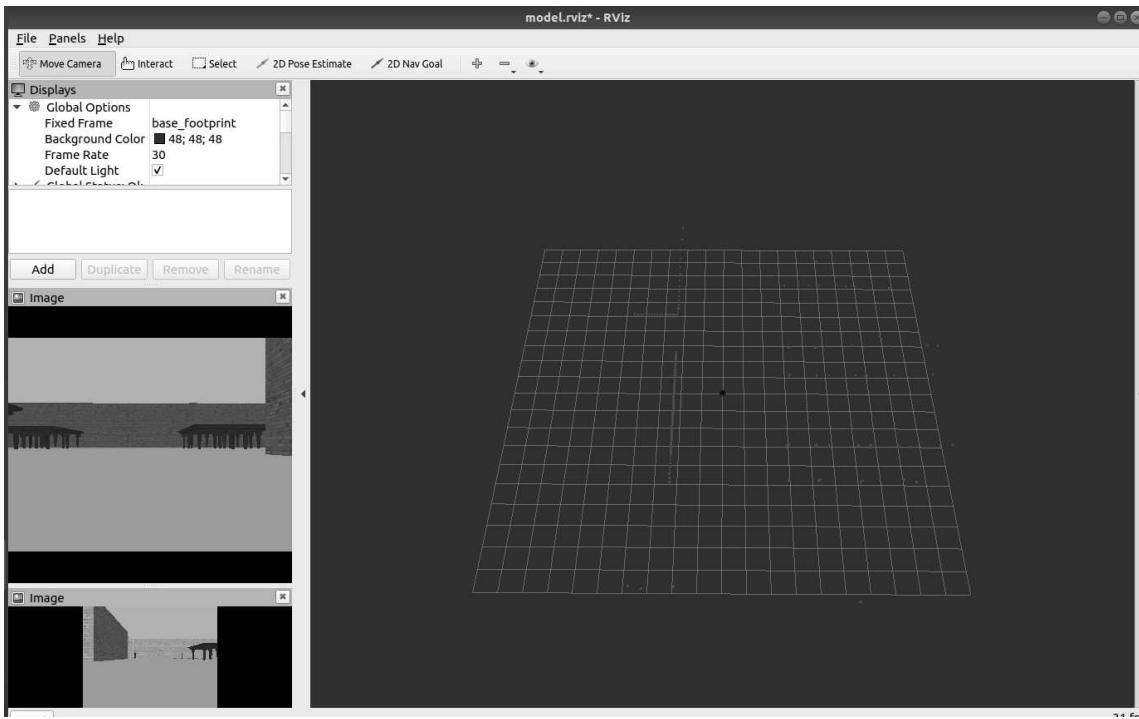
Sau khi đã có được mô hình robot, ta còn cần một môi trường để robot có thể chạy và thử nghiệm. Gazebo có hỗ trợ tạo một môi trường tùy chỉnh theo nhu cầu người sử dụng. Nhờ đó ta có thể dễ dàng tạo các môi trường tương tự với môi trường thực tế để chạy robot thử nghiệm trên Gazebo.



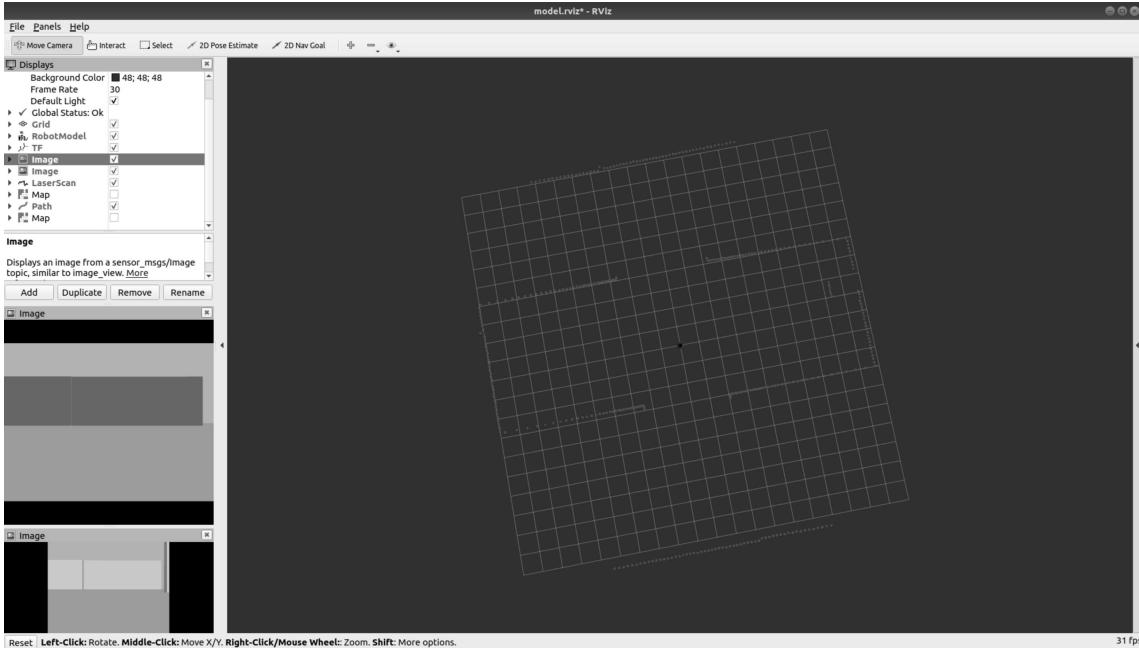
Hình 20: Map mô phỏng trên gazebo (Map 1)



Hình 21: Map mô phỏng trên gazebo (Map 2)



Hình 22: Hình ảnh camera và lidar trên rviz (Map 1)



Hình 23: Hình ảnh camera và lidar trên rviz (Map 2)

3.2.1.3 Xây dựng map với rtabmap trên Gazebo

Khi đã có được mô hình robot và môi trường mô phỏng, ta sẽ tiến hành xây dựng map với rtabmap. Để đơn giản, nhóm đã viết một file launch rtabmap sử dụng 1 camera với các thông số:

• Arguments

```

1 <arg name="database_path"           default="~/.ros/rtabmap.db"/>
2 <arg name="args"                  default=""/>
3 <arg name="wait_for_transform"    default="0.2"/>
4 <arg name="localization"        default="false"/>
```

• RTABMAP parameters

```

1 <param name="RGBD/ProximityBySpace"      type="bool" value="true"/>
2 <param name="RGBD/OptimizeFromGraphEnd"   type="bool" value="false"/>
3 <param name="Kp/MaxDepth"                 type="string" value="4.0"/>
4 <param name="Reg/Strategy"                type="string" value="0"/>
5 <param name="Icp/CorrespondenceRatio"     type="string" value="0.3"/>
6 <param name="Vis/MinInliers"              type="string" value="15"/>
7 <param name="Vis/InlierDistance"          type="string" value="0.1"/>
8 <param name="Vis/CorGuessWinSize"          type="string" value="0"/>
9 <param name="RGBD/AngularUpdate"          type="string" value="0.1"/>
10 <param name="RGBD/LinearUpdate"           type="string" value="0.1"/>
11 <param name="RGBD/ProximityPathMaxNeighbors" type="string" value="0"/>
12 <param name="Rtabmap/TimeThr"             type="string" value="0"/>
13 <param name="Mem/RehearsalSimilarity"     type="string" value="0.30"/>
14 <param name="Reg/Force3DoF"               type="string" value="true"/>
15 <param name="GridGlobal/MinSize"           type="string" value="3"/>
16 <param name="RGBD/SavedLocalizationIgnored" type="string" value="true"/>
```

• RTABMAP ROS parameters

```

1 <param name="frame_id"                 type="string" value="base_footprint"/>
2 <param name="queue_size"               type="int" value="10"/>
3 <param name="subscribe_depth"         type="bool" value="true"/>
4 <param name="subscribe_scan"          type="bool" value="true"/>
5 <param name="database_path"           type="string" value="$(arg database_path)"/>
6 <param name="wait_for_transform_duration" type="double" value="$(arg
    wait_for_transform)"/>
7 <param if="$(arg localization)" name="Mem/IncrementalMemory" type="string" value="
    false"/>
8 <param if="$(arg localization)" name="Mem/InitWMWithAllNodes" type="string" value="
    true"/>
```

• Input

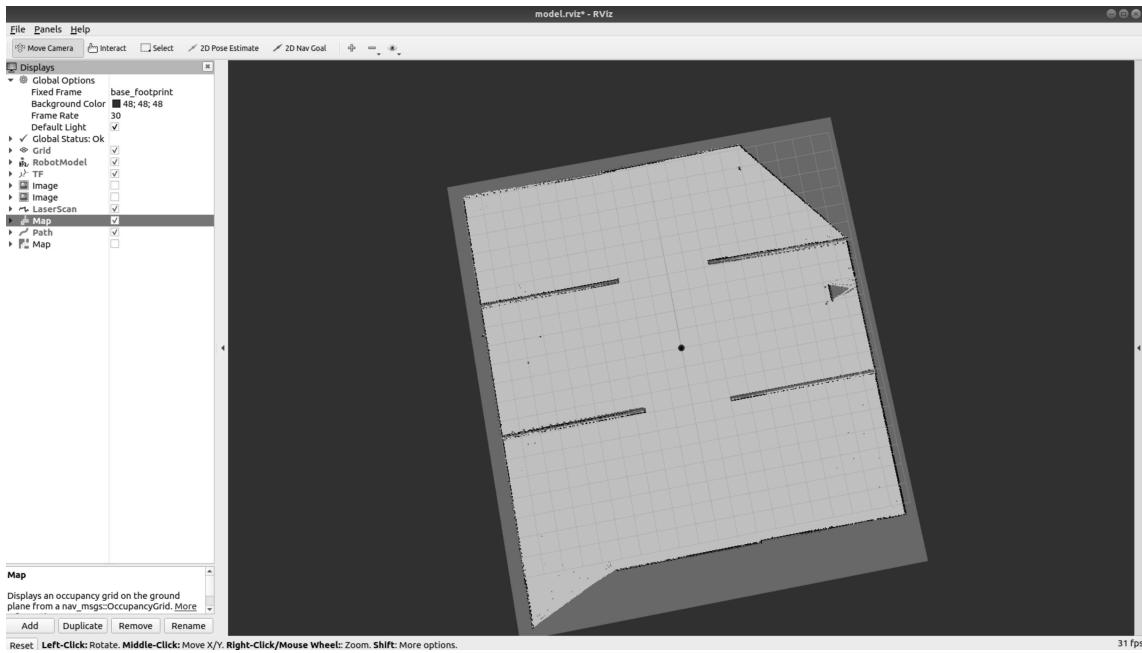
```

1 <remap from="odom" to="/odom"/>
2 <remap from="scan" to="/scan"/>
3 <remap from="rgb/image" to="/camera_01/rgb/image_raw"/>
4 <remap from="depth/image" to="/camera_01/depth/image_raw"/>
5 <remap from="rgb/camera_info" to="/camera_01/rgb/camera_info"/>
```

• Output

```
1 <remap from="grid_map" to="/map"/>
```

Sau đó tiến hành build map trên gazebo tương tự với việc build map với robot thực tế. Dưới đây là hình ảnh của map thu được sau khi build.



Hình 24: Map xem trên rviz (Map 2)

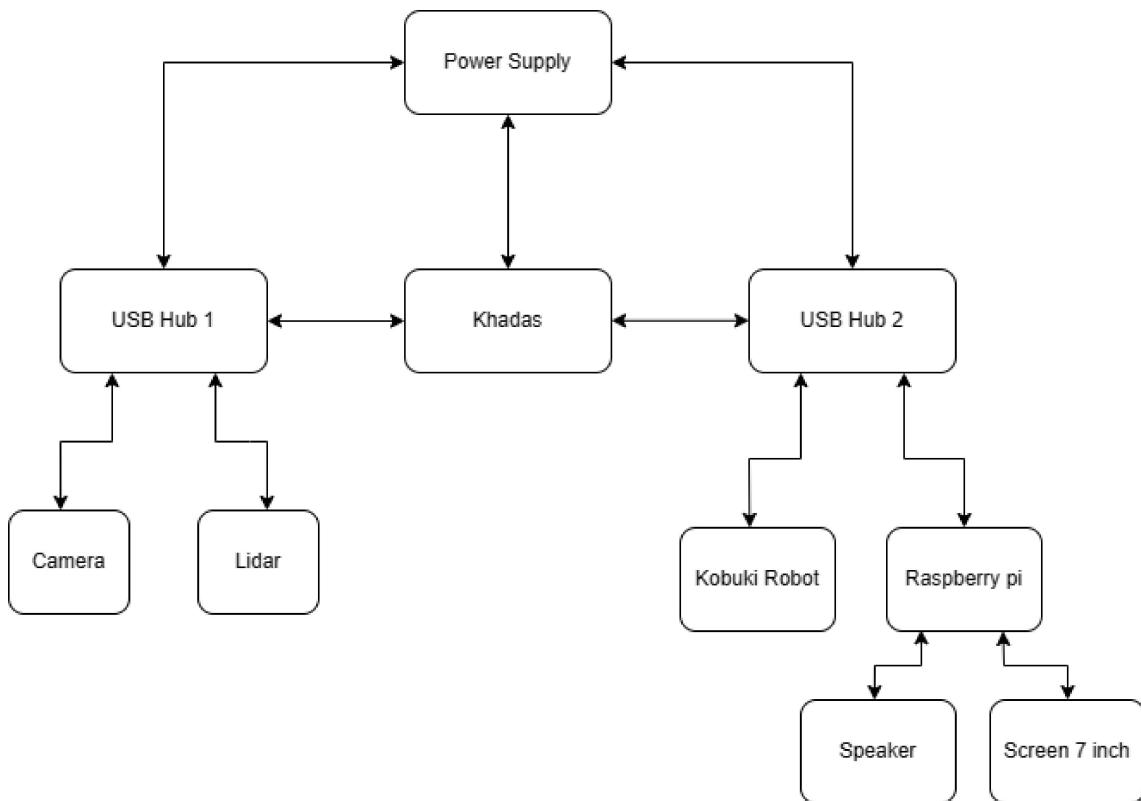
3.2.2 Mô hình robot thực tế

3.2.2.1 Các thiết bị

Cấu trúc của Kobuki Robot bao gồm các thiết bị sau:

- **Board nhúng Khadas Edge-V:** Bo mạch tích hợp (SBC) nhỏ gọn và mạnh mẽ cho các ứng dụng nhúng và học máy.
- **Astra camera:** Cảm biến camera 3D được sử dụng trong các ứng dụng thị giác máy tính và thực tế tăng cường.
- **RpLidar A1:** Cảm biến LiDAR nhỏ gọn được sử dụng để tạo ra hình ảnh 2D và 3D của môi trường xung quanh.
- **Raspberry pi 3B và màn hình:** Bo mạch tích hợp (SBC) được sử dụng trong các ứng dụng nhúng, IoT và một màn hình cảm ứng được thiết kế để sử dụng với Raspberry Pi.
- **Kobuki Robot:** Robot di động được sử dụng trong các ứng dụng dọn dẹp, giám sát, và vận chuyển.

Các thiết bị được kết nối với nhau như sơ đồ sau:

Hình 25: *Kobuki Robot*

Sau khi kết nối ta có mô hình thiết bị như sau;

Hình 26: *Kobuki Robot*

3.2.2.2 Xây dựng map với Rtabmap

Để robot có thể tự định vị bản thân nó và tự hành, việc đầu tiên cần làm là xây dựng bản đồ. Ta sử dụng các Ros package sau để lấy dữ liệu từ môi trường xung quanh:

- **RpLidar** Đầu tiên, cần thay đổi cổng nối USB cho máy quét LIDAR bằng cách chỉnh sửa dòng này trong tệp rplidar.launch:

```
1 <param name="serial_port" type="string" value="/dev/ttyUSB1"/>
```

Sau đó khởi động file launch:

```
1 rosrun rplidar_ros rplidar.launch
```

- **Astra camera** Tiếp theo, khởi động node camera để lấy thông tin hình ảnh gồm RGB Image và Depth Image từ Astra Camera bằng lệnh sau:

```
1 rosrun astra_camera multi_astra.launch
```

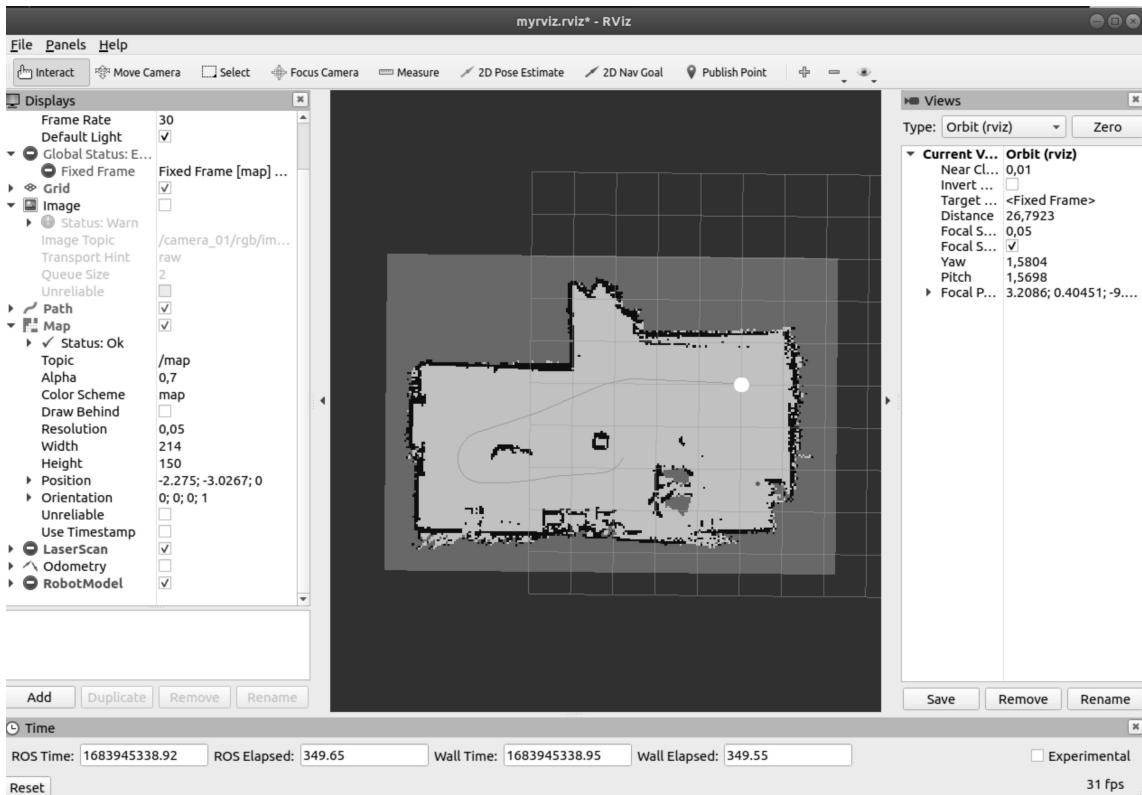
- **Kobuki** Để robot có thể di chuyển, ta cần chạy Kobuki_node để kết nối tới Kobuki Robot và bắt đầu điều khiển bằng tay

```
1 rosrun kobuki_node kobuki.launch
2 rosrun kobuki_keyop safe_keyop.launch
```

- **Rtabmap** Cuối cùng khởi chạy rtabmap để bắt đầu quá trình build map. Chú ý, để bắt đầu mapping ta cần chuyển giá trị *localization* thành false

```
1 rosrun rtabmap_ros rtabmap_1_cam.launch localization:=false
```

Sau quá trình build map, ta sẽ có một bản đồ để robot có thể tự di chuyển, mở bằng rviz để có hình ảnh sau:



Hình 27: Xây dựng map với Rtabmap

3.2.3 Thiết kế các message, services và topics

3.2.3.1 Message và Service

Các node có trong mạng ROS trao đổi thông tin với nhau thông qua các *message* được quy định với nhau từ trước. Các *message* có thể được tùy biến, thêm mới dựa trên những tập *message* cơ bản mà ROS hỗ trợ, có thể kể tên như std_msgs, geometry_msgs,... Bên cạnh đó, ROS còn sử dụng *service* để yêu cầu node nhận thực thi một nhiệm vụ nào đó, sau đó có thể có hoặc không gửi kết quả trả về Node gửi. Cơ chế này giống như khi gọi API (Application Programming Interface), ta có thể quy định service đó có tham số truyền vào/ tham số trả về hay không tùy theo nhu cầu sử dụng.

Ta sẽ thiết kế các service để yêu cầu Node Station thực hiện việc thêm, xóa hay lấy thông tin các trạm (station). Các service này sẽ là GetStation, AddStation, DeleteStation.

Dầu tiên, ta tạo message thông tin của một station sẽ có dạng như sau:

```

1 # Message for a station with name, pose, degrees, and id
2 string name
3 float32 positionX
4 float32 positionY
5 float32 positionZ
6 float32 orientationX
7 float32 orientationY
8 float32 orientationZ

```

```

9 float32 orientationW
10 string id

```

Service dùng để thêm trạm (station).

```

1 string name
2 float32 positionX
3 float32 positionY
4 float32 positionZ
5 float32 orientationX
6 float32 orientationY
7 float32 orientationZ
8 float32 orientationW
9 string id
10 ---
11 bool success

```

Service dùng để xóa trạm (station).

```

1 # Service to delete a station with id
2 string id
3 ---
4 bool success

```

Service dùng để lấy thông tin trạm (station).

```

1 # Service to get a list of stations from rosparam
2 ---
3 # List of stations
4 Station[] station_list
5 ---

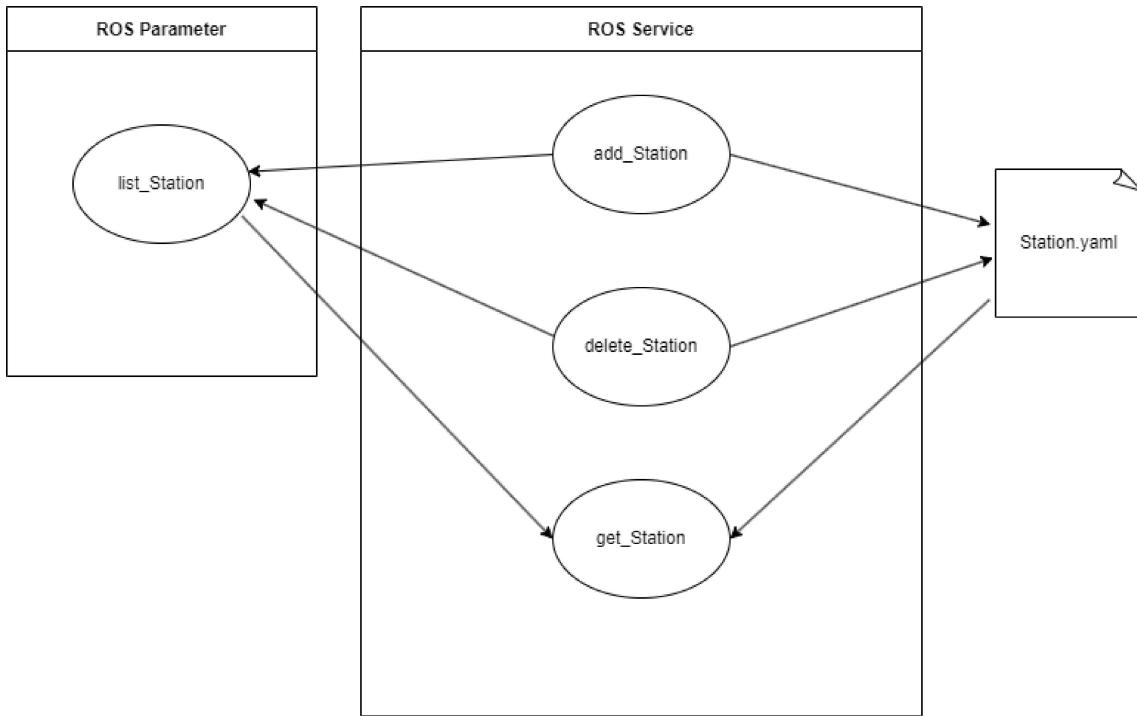
```

3.2.3.2 Truy xuất và gọi các message, service

Với mục tiêu cho người dùng thêm hoặc xóa các trạm (station) để có thể dễ dàng lưu lại các vị trí quan trọng, ta sẽ thiết kế **Node Station Server** trong Ros để tạo nên các *services* để người dùng gọi đến. **Node Station Server** có các chức năng sau sau:

- Add Station: Thêm các vị trí vào hệ thống
- Delete Station: Xóa các vị trí khỏi hệ thống
- Get Station: Lấy danh sách các Station mới nhất

Ta có sơ đồ dưới đây:



Hình 28: Các chức năng của Node Station Server

3.2.3.3 Topics

ROS sử dụng mô hình publish-subscribe để truyền thông tin giữa các thành phần của hệ thống, các thành phần giao tiếp thông qua các topic, là các kênh truyền thông mà dữ liệu được gửi và nhận trong hệ thống. Về cơ bản, một node trong ROS sẽ mở ra (*publish*) một kênh thông tin (*Topic*) để gửi các thông báo về sự kiện nào đó cho các node khác đăng ký nhận thông tin (*subscribe*) từ kênh thông báo đó để xử lý thông tin hoặc kích hoạt một sự kiện.

Trong hệ thống tự hành, nhóm sẽ tạo ra các topic con nằm trong nhóm `/goal` để giao tiếp giữa các node. Các topic con gồm:

- **move** thông tin danh sách các vị trí cần di chuyển đến
- **arrive** Thông báo robot đã di chuyển đến nơi
- **confirm** thông báo rằng người nhận đã xác nhận nhận hàng
- **emergency** tín hiệu khẩn cấp dừng mọi hoạt động

3.2.4 Xử lý hàng chờ các Task

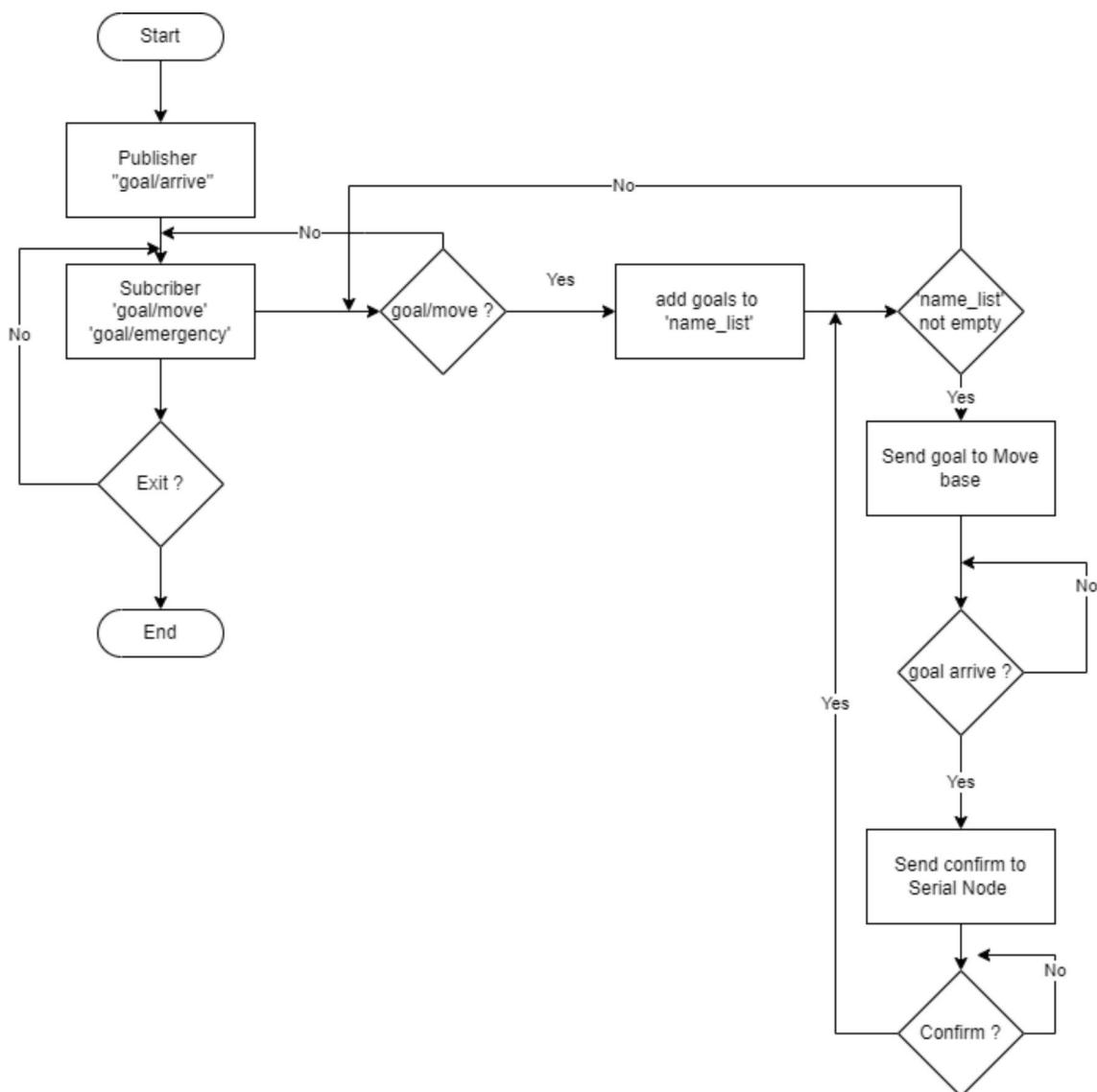
Trong quá trình giao hàng đến nhiều địa điểm khác nhau, robot sẽ gặp phải nhiều tình huống khác nhau như cần phải chờ người dùng xác nhận hoặc xử lý hàng hóa tại một địa điểm cụ thể trước khi di chuyển tiếp đến địa điểm khác. Để giải quyết vấn đề này, ta có thể sử dụng một hàng chờ để quản lý quá trình giao

hàng của robot.

Trước khi robot bắt đầu quá trình giao hàng, ta có thể tạo một danh sách các địa điểm cần giao hàng và thời gian dự kiến để hoàn thành việc giao hàng tại mỗi địa điểm. Robot sẽ theo dõi danh sách này và di chuyển đến từng địa điểm theo thứ tự.

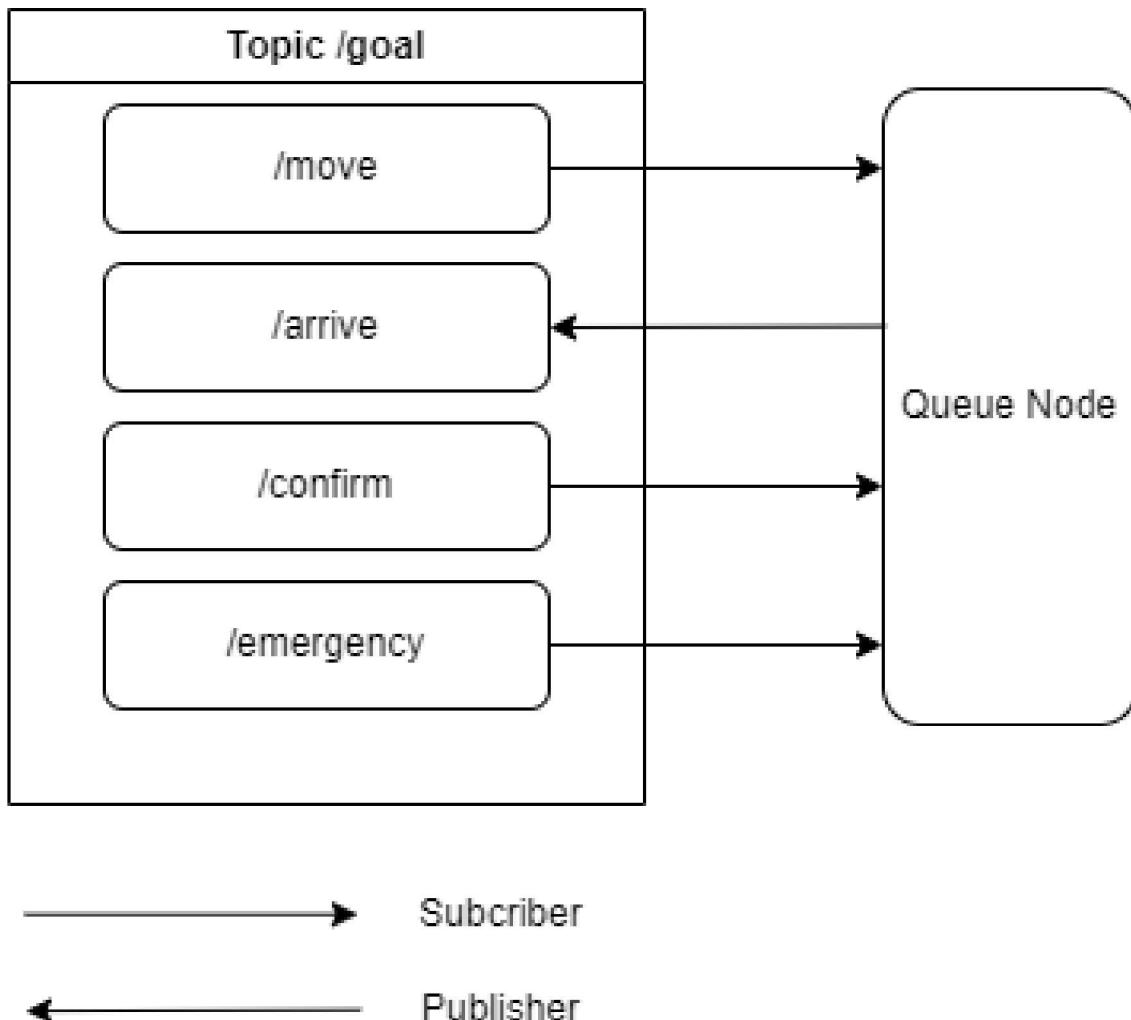
Khi robot đến một địa điểm để giao hàng, nó sẽ đưa ra thông báo cho người dùng biết rằng hàng hóa đã được đưa tới địa điểm cần giao. Robot sẽ đợi người dùng xác nhận rằng hàng hóa đã được nhận và xử lý tại địa điểm đó trước khi di chuyển đến địa điểm kế tiếp. Nếu người dùng không xác nhận, robot sẽ tiếp tục đợi trong hàng chờ để quản lý quá trình giao hàng.

Khi hàng hóa đã được xử lý tại một địa điểm, robot sẽ được di chuyển đến địa điểm tiếp theo trong danh sách và bắt đầu quá trình giao hàng tại địa điểm đó.



Hình 29: Cách Queue Node hoạt động

Trong quá trình hoạt động, *Queue Node* sẽ subscribe các topic để lấy thông tin và xử lý hàng chờ và publish các thông tin về trạng thái các task đang xử lý. Ta có các topic mà node publish và subscribe như sau:



Hình 30: Các topic của Queue Node

3.2.5 Màn hình giao tiếp với người dùng

3.2.5.1 Chức năng người dùng

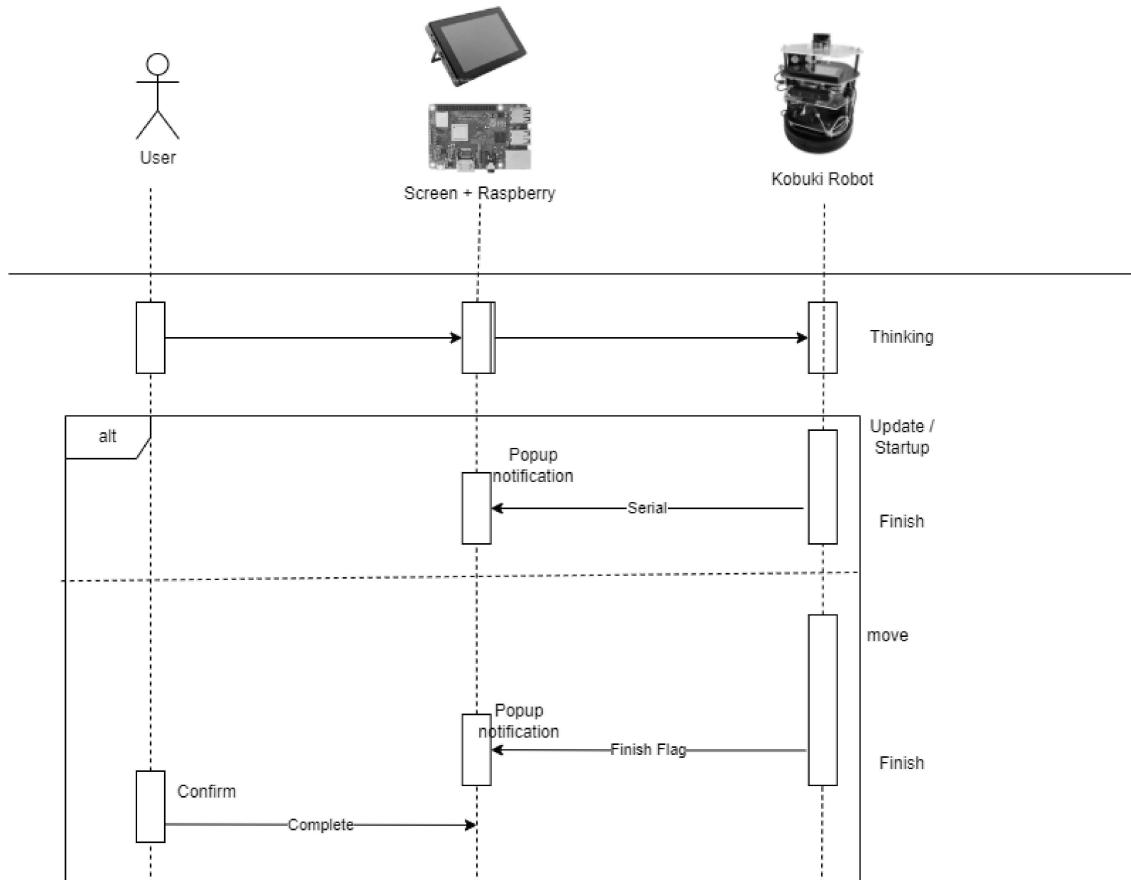
Người dùng sẽ tương tác trực tiếp với màn hình của Raspberry Pi. Trên màn hình này, người dùng có thể nhập các lệnh hoặc chọn các tùy chọn từ giao diện người dùng đồ họa (GUI) để điều khiển robot. Các lệnh này có thể bao gồm việc di chuyển robot, bắt đầu hoặc dừng các tác vụ, khởi động robot hoặc cấp nhật danh sách Station.

Khi Raspberry Pi nhận được lệnh từ người dùng, nó sẽ dịch lệnh này thành tín hiệu điều khiển tương ứng và gửi chúng đến robot qua kết nối serial. Robot sau

đó sẽ thực hiện các hành động theo tín hiệu điều khiển mà nó nhận được.

Sau khi robot thực hiện các tác vụ, nó sẽ gửi dữ liệu phản hồi về Raspberry Pi, bao gồm thông tin về các tác vụ. Raspberry Pi sau đó sẽ hiển thị thông tin này lên màn hình để người dùng có thể theo dõi.

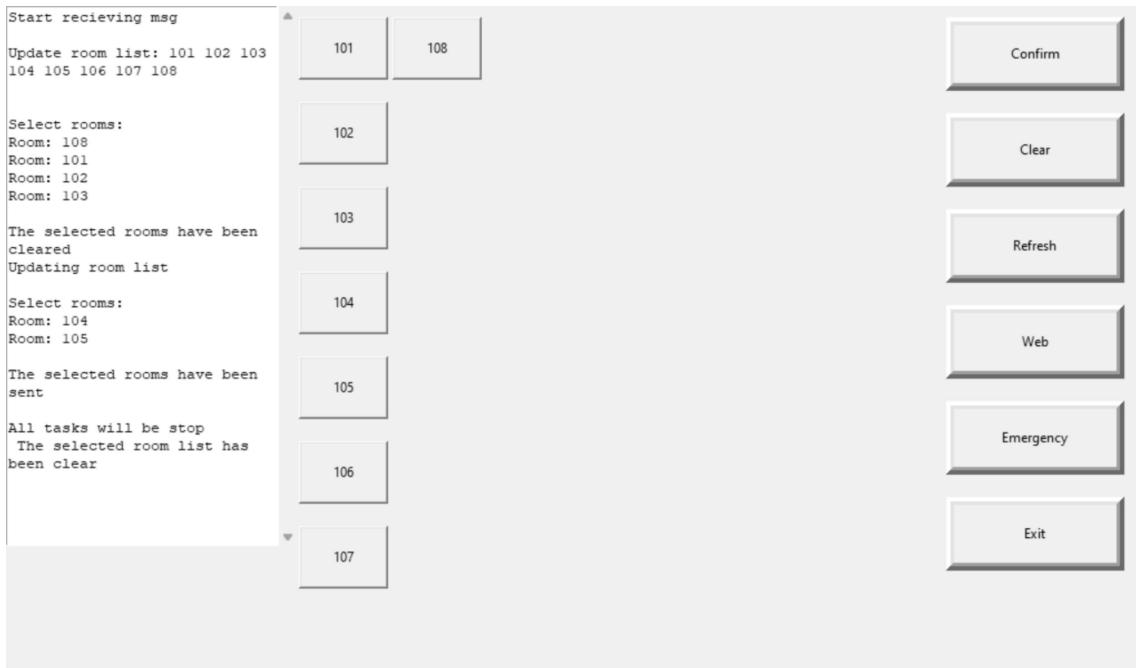
Tổng quát, sơ đồ dưới đây mô tả cách người dùng, Raspberry Pi, và robot tương tác với nhau để thực hiện các tác vụ cụ thể trong một hệ thống.



Hình 31: Sơ đồ hoạt động của người dùng

3.2.5.2 Tạo giao diện người dùng

Chương trình giao diện cho người dùng với **Tkinter** để điều khiển robot trên Raspberry Pi có thể được thiết kế để cho phép người dùng chọn 1 danh sách các địa điểm và gửi cho robot xử lý. Các phần tử chính của giao diện bao gồm các nút điều khiển, khung hiển thị và các popup để thông báo cho người dùng.

**Hình 32:** Giao diện người dùng

Chương trình sẽ có 4 chức năng chính là

- **Add/Clear list:** Người dùng có thể chọn 1 hoặc nhiều các vị trí khác nhau cũng như xóa bỏ danh sách đã chọn.
- **Refresh List:** Chức năng này để cập nhật danh sách các trạm (station) khi có sự thay đổi.
- **Start System:** Chức năng này để khởi động các thành phần của robot hoặc khởi động *web bridge ros* để lấy thông tin của robot cho Web UI.
- **Emergency:** Khi gặp phải vấn đề khẩn cấp, chọn chức năng này để dừng mọi hoạt động của Robot.

Tuy nhiên, khi sử dụng, người dùng cần phải xác nhận trước khi gửi lệnh thực hiện cho robot để tránh trường hợp gửi nhầm và khi chọn các station để di chuyển, mỗi station chỉ được chọn 1 lần.



Hình 33: Báo lỗi khi chọn lắp vị trí



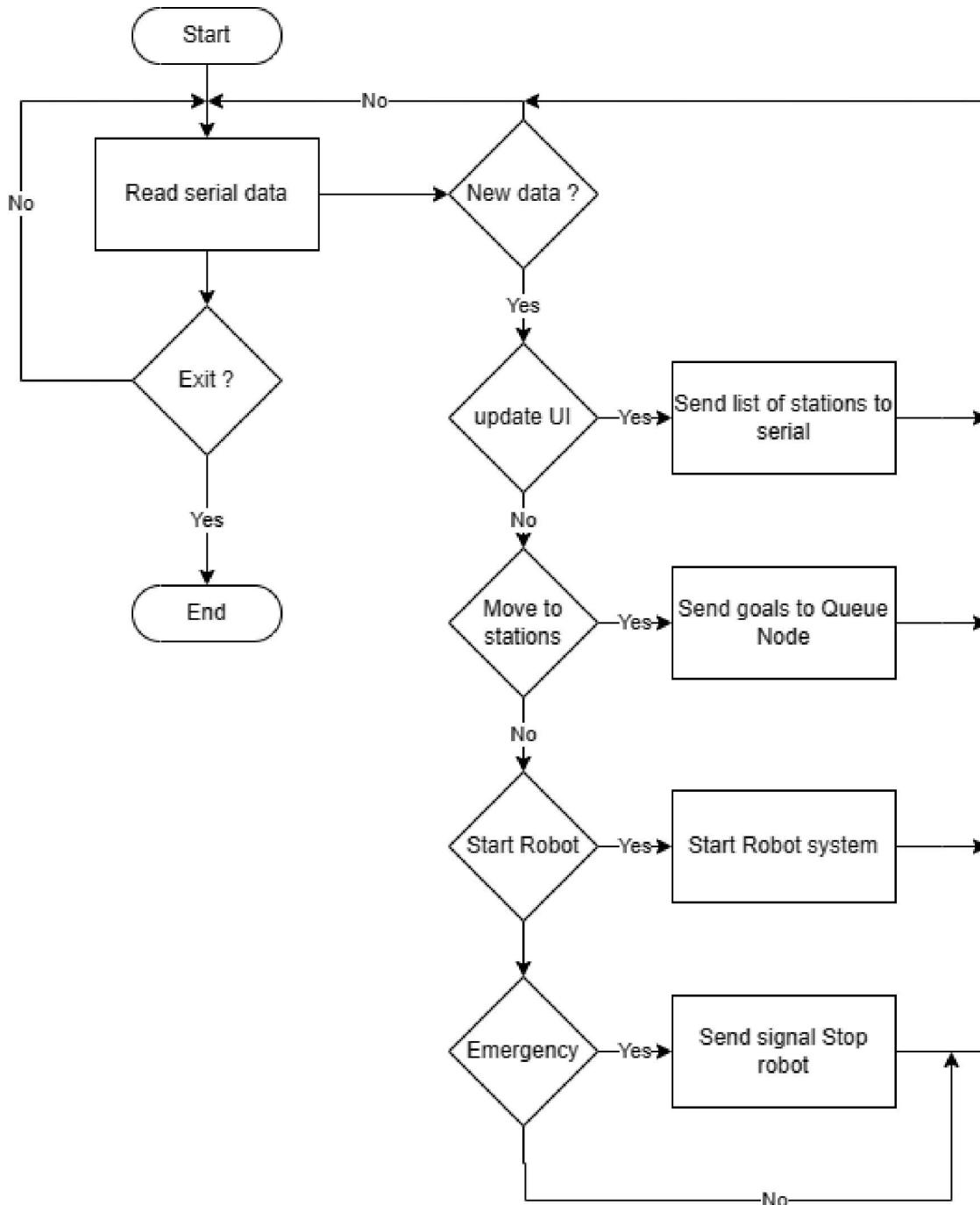
Hình 34: Xác nhận khi chọn

3.2.5.3 Tạo node đọc dữ liệu người dùng

Node xử lý dữ liệu serial là một node chịu trách nhiệm đọc và xử lý dữ liệu tín hiệu serial đến từ Raspberry Pi do người dùng chọn thông qua màn hình.

Node sử dụng thư viện Pyserial để kết nối với cổng serial và đọc dữ liệu tín hiệu đầu vào. Sau khi đọc dữ liệu, node sẽ xử lý các tác vụ tương ứng và đóng gói dữ liệu gửi đến các node khác trong mạng ROS.

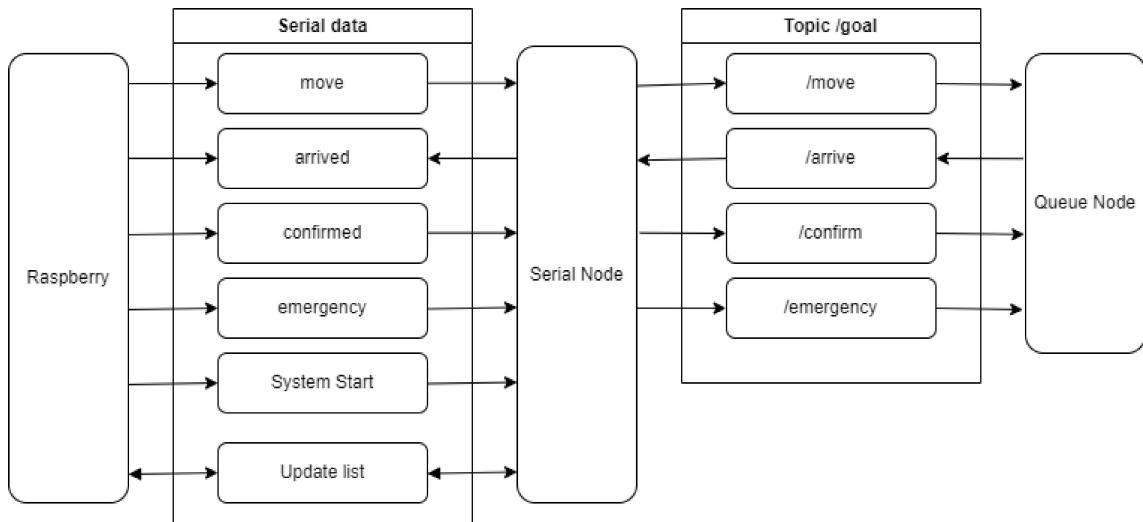
Node Serial sẽ có sơ đồ hoạt động như sau:



Hình 35: Serial Node

3.2.5.4 Sơ đồ giao tiếp của hệ thống

Sau khi tạo xong *Queue Node*, *Serial Node* và giao diện người dùng, ta liên kết chúng với nhau thông qua các message đã được định nghĩa khi từ giao diện người dùng gửi đến Robot và từ node này sang node kia trong ROS. Dưới đây là hình ảnh mô tả các thông tin khi truyền dữ liệu qua serial và thông qua các topic trong ROS:



Hình 36: Các thành phần giao tiếp với nhau

3.2.6 Web Server

3.2.6.1 Xây dựng giao diện Web tương tác với Ros



Hình 37: Web UI

Hình trên mô tả giao diện Web, người dùng có thể quan sát và điều khiển robot. Tại đây có thể xem hình ảnh camera mà robot truyền về, bản đồ mà robot đã dựng được cũng như vị trí robot trên bản đồ đó. Bên cạnh đó có thể điều khiển robot thông qua các nút điều hướng qua bàn phím, điều chỉnh vận tốc của robot, đặt tên cho 1 vị trí nào đó (thêm trạm) và ra lệnh cho robot di chuyển đến trạm đó.

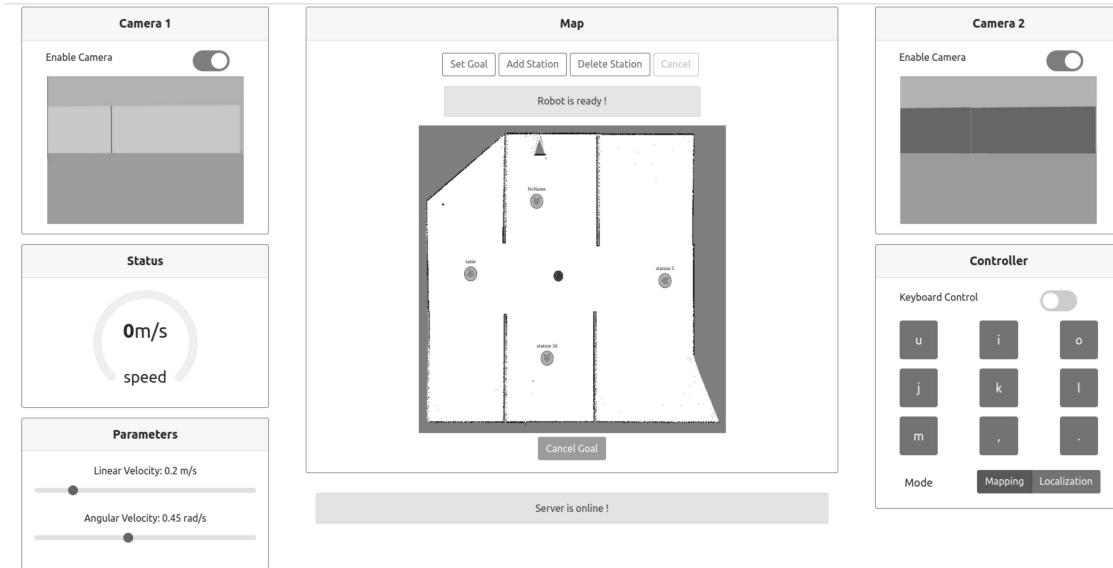
Ứng dụng giao diện người dùng được xây dựng dựa trên công nghệ ReactJS, với ngôn ngữ sử dụng là Javascript. Bao gồm các thành phần (component) nội dung chính:

- **Camera:** Hiển thị hình ảnh thu về từ camera của robot, sử dụng nút switch, để cho phép hoặc không cho phép chức năng.
- **Status:** Hiển thị trạng thái của robot bao gồm vận tốc hiện tại, phần trạm CPU và bộ nhớ của bộ xử lý trung tâm có trên robot.
- **Parameters:** Hiện tại chỉ cho phép người dùng tùy chỉnh vận tốc tối đa của robot.
- **Map:** Hiển thị bản đồ do robot xây dựng được, các nút chức năng để thêm/xóa trạm cố định trên bản đồ, và gửi tọa độ điểm đến xuống robot.
- **Controller:** Điều khiển robot các nút nhấn có trên giao diện hoặc cho phép điều khiển robot bằng bàn phím máy tính. Ngoài ra có thể lựa chọn chế độ hoạt động của robot là mapping hay localization.

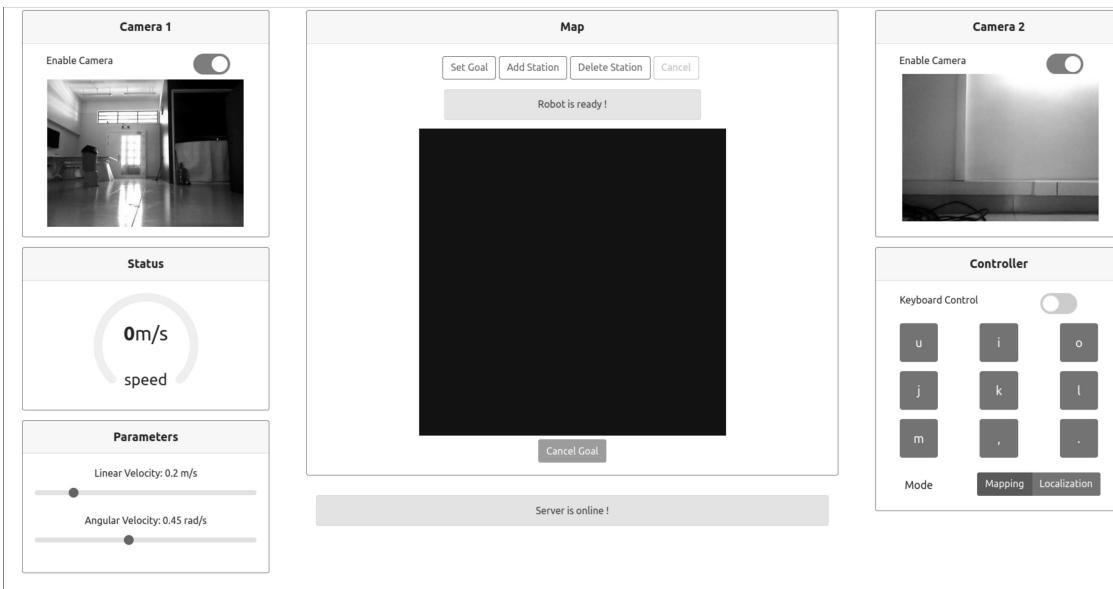
Các thành phần nội dung trên được tạo và quản lý từ khôi Main, khôi này còn tạo kết nối đến node thông qua websocket, và theo đó sẽ đăng ký lắng nghe (subscribe) các chủ đề (topic) cần thiết như: vận tốc robot, hình ảnh từ camera, bản đồ 2D, vị trí robot. Và thực hiện phát đi tin nhắn (message) qua các chủ đề về: lệnh điều khiển vận tốc, tọa độ mục tiêu để yêu cầu robot tự hành đến. Mỗi khi có tin nhắn mới, tùy vào chủ đề mà khôi Main sẽ truyền các state cho khôi quản lý nội dung tương ứng để hiển thị. Và mỗi khi người dùng tương tác với khôi nội dung kể trên, các khôi này sẽ có cơ chế gọi về (callback) khôi Main để khôi Main có thể thực hiện tác vụ và điều phối hoạt động.

3.2.6.2 Các chức năng của Web

- **Camera:** Hiển thị hình ảnh từ camera, có thể bật hoặc tắt camera.



Hình 38: Hình ảnh camera mô phỏng trên Gazebo



Hình 39: Hình ảnh camera thực tế

- **Status:** Hiển thị vận tốc hiện tại của robot.



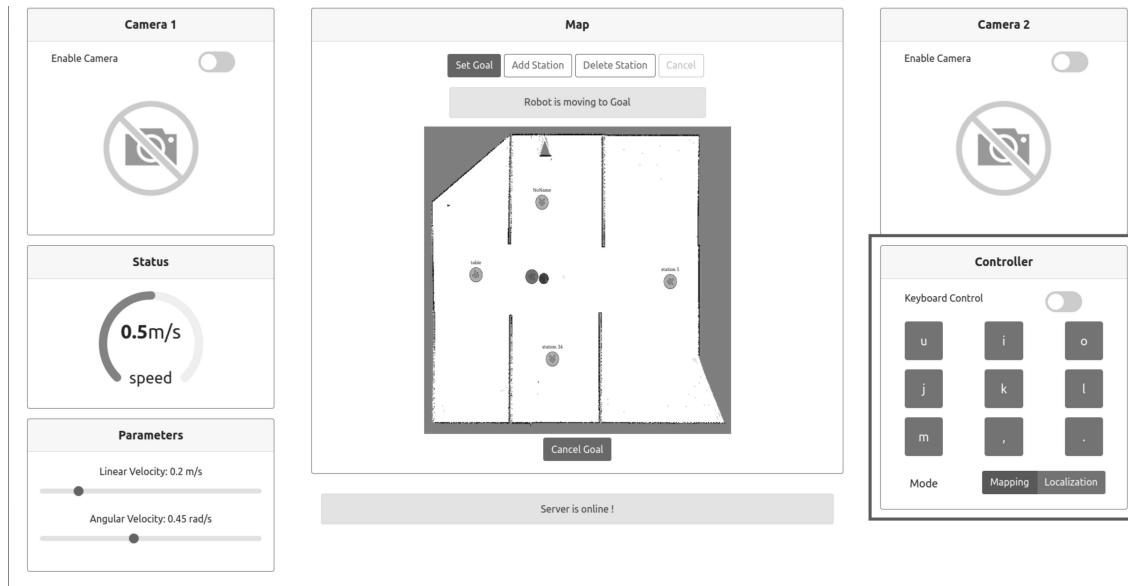
Hình 40: Vận tốc hiện tại của robot

- **Parameters:** Tùy chỉnh vận tốc (linear) và vận tốc gốc (angular).



Hình 41: Điều chỉnh Linear Velocity hoặc Angular Velocity

- **Controller:** Điều khiển robot bằng các nút nhấn trên giao diện hoặc bằng bàn phím máy tính (khi bật switch Keyboard Control). Ngoài ra có thể lựa chọn chế độ hoạt động là Mapping hay Localization.



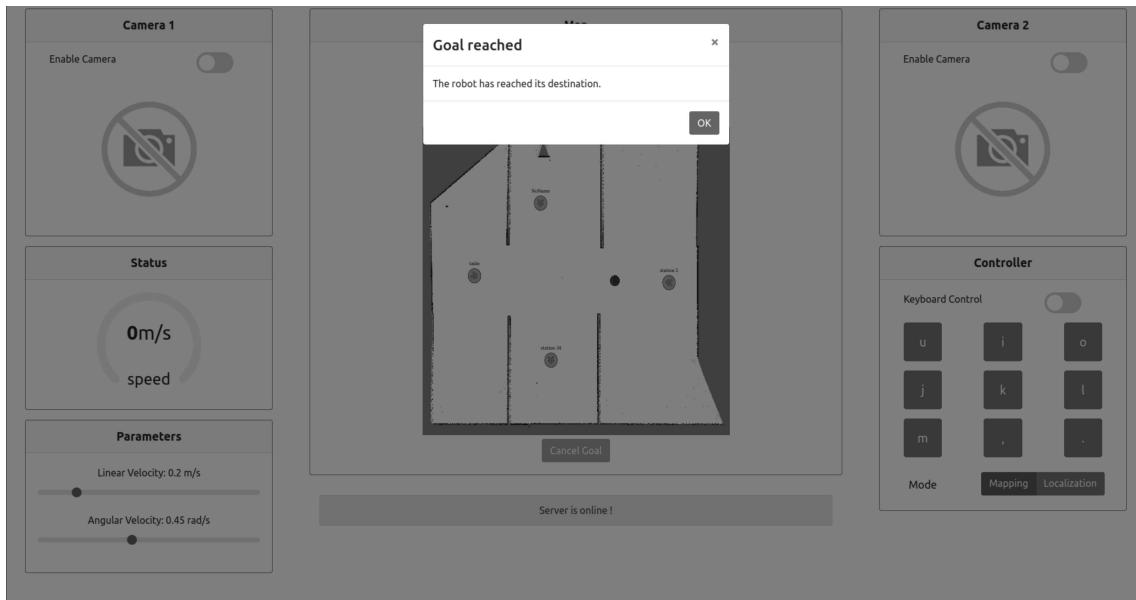
Hình 42: Bô điều khiển thử công robot

- **Map:** Hiển thị bản đồ do robot xây dựng được.
 - **Mode Set Goal:** gửi vị trí tọa độ xuống map, robot sẽ di chuyển đến vị trí này. Trong khi robot di chuyển ta có thể hủy goal bằng cách nhấn nút Cancel Goal.



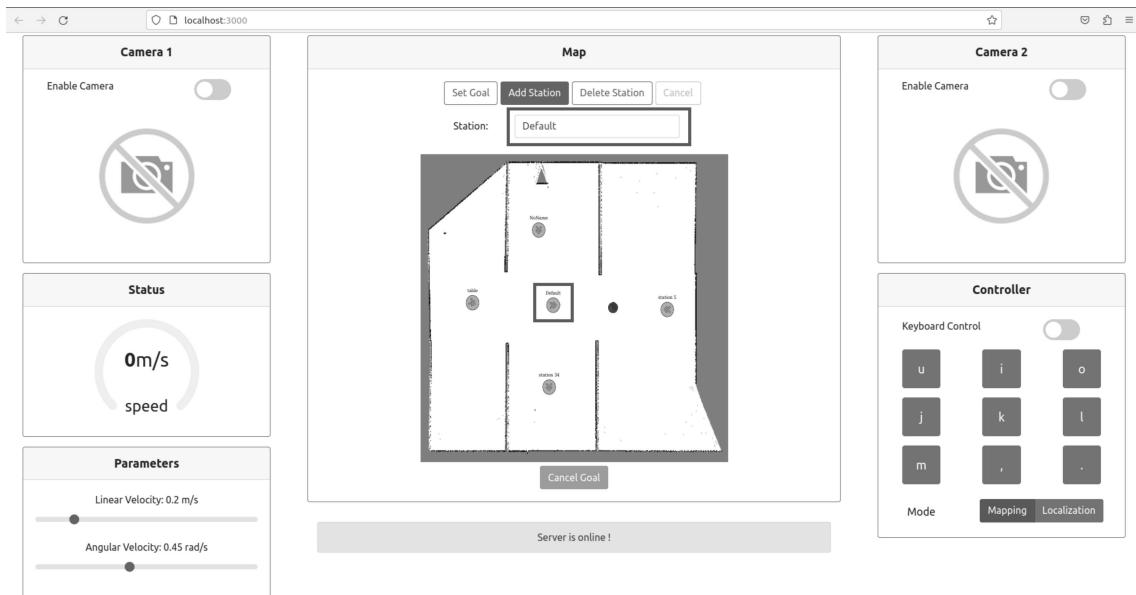
Hình 43: Mode Setgoal

Khi đến đích sẽ có message thông báo.



Hình 44: Khi robot đến đích

- **Mode Add Station:** điền tên của station sau đó chọn 1 vị trí tọa độ trên map để tạo một trạm mới.



Hình 45: Thêm trạm có tên Default

- **Mode Delete Station:** chọn vào một trạm (station) đang có trên map để xóa trạm này.



Hình 46: Xóa trạm có tên NoName

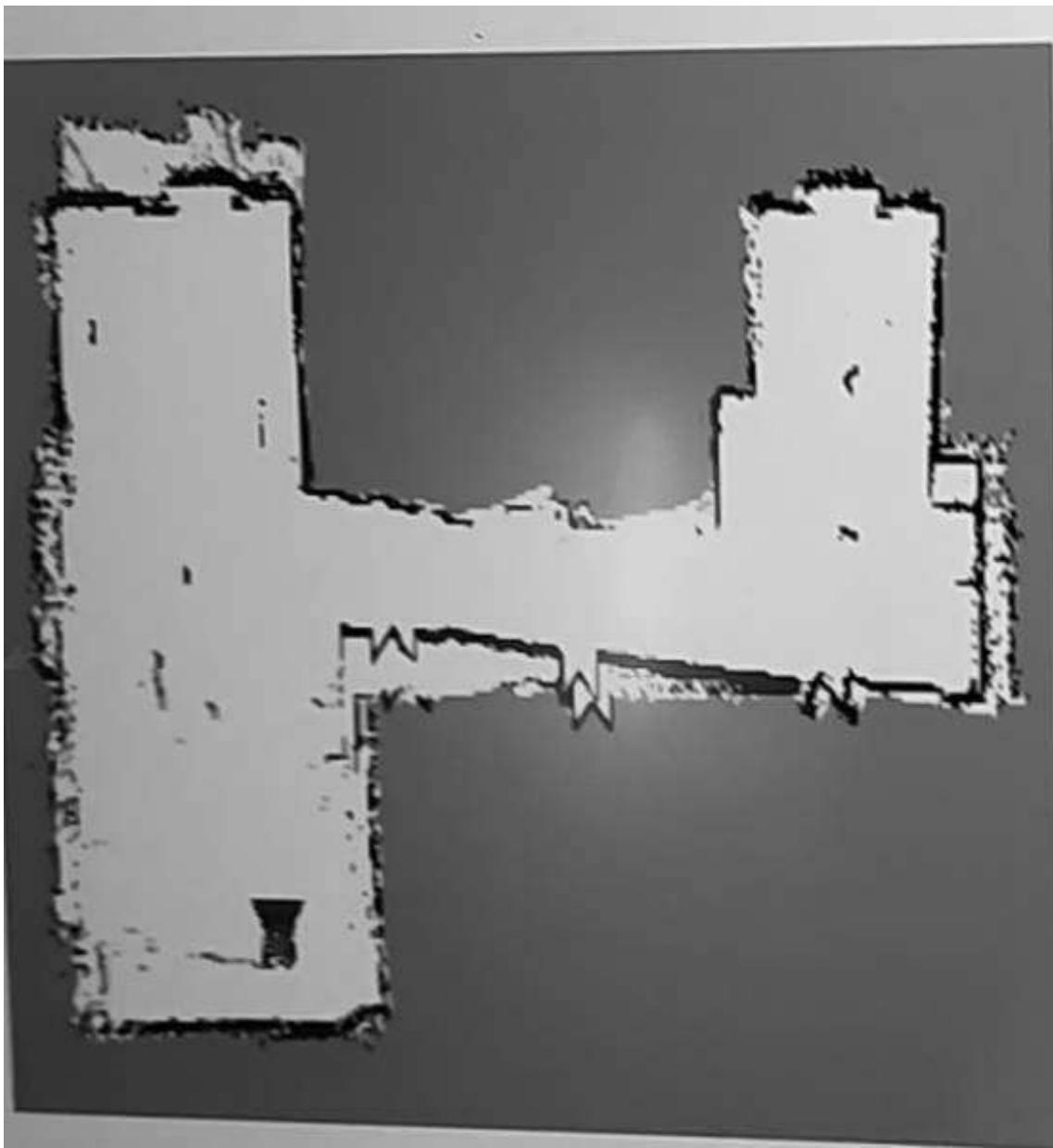
3.3 Thủ nghiệm và đánh giá

3.3.1 Quá trình xây dựng bản đồ

Trong phần này, nhóm đã tiến hành một số thử nghiệm để xem SLAM và RTAB Map hoạt động như thế nào. Nhóm sử dụng 1 camera và tốc độ di chuyển của robot là 0.1m/s.

Trong quá trình xây dựng bản đồ, robot sẽ di chuyển và thu thập dữ liệu cảm biến lidar và camera từ môi trường xung quanh để xác định vị trí của nó và tạo bản đồ chính xác hơn. Và nhóm sử dụng khu vực phòng 709 H6 tại cơ sở 2 để thử nghiệm.

Dầu tiên, nhóm bắt đầu mapping toàn bộ khu vực phòng và thu được kết quả:

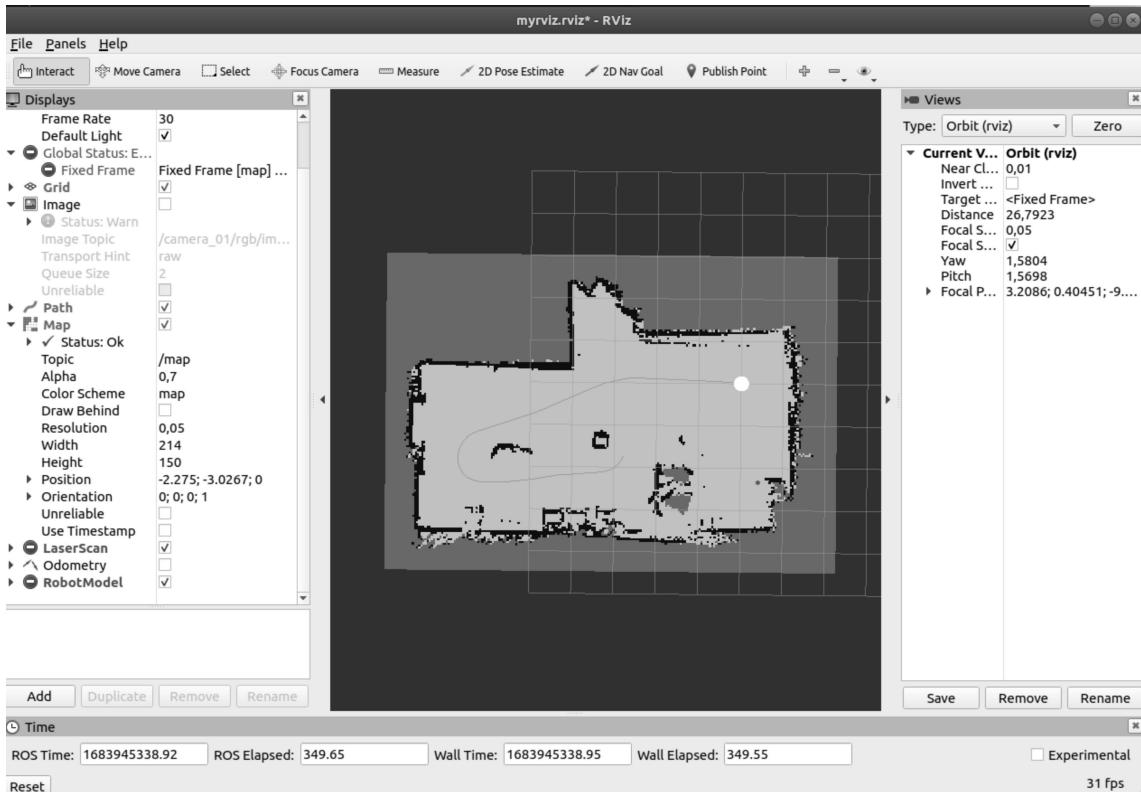


Hình 47: Mapping toàn bộ phòng 709

Tại khu vực phòng có không gian rộng lớn nên khi robot di chuyển để mapping 1 vòng quanh phòng thì kết quả cho ra đã bị lệch nhiều so với thực tế. Tại khu vực hành lang, khi robot di chuyển đi qua 1 lần thì tạo nên khu vực màu trắng (nên có thể di chuyển được) nhưng khi robot quay ngược lại 1 lần nữa thì phần tường đã lấn vào phía trong do sai số khi tự định vị trên bản đồ. Và tại khu vực sảnh chính, hàng ghế phía trong tường và thùng rác ở giữa phòng đã biến mất hoặc nhận diện chỉ còn 1 điểm nhỏ.

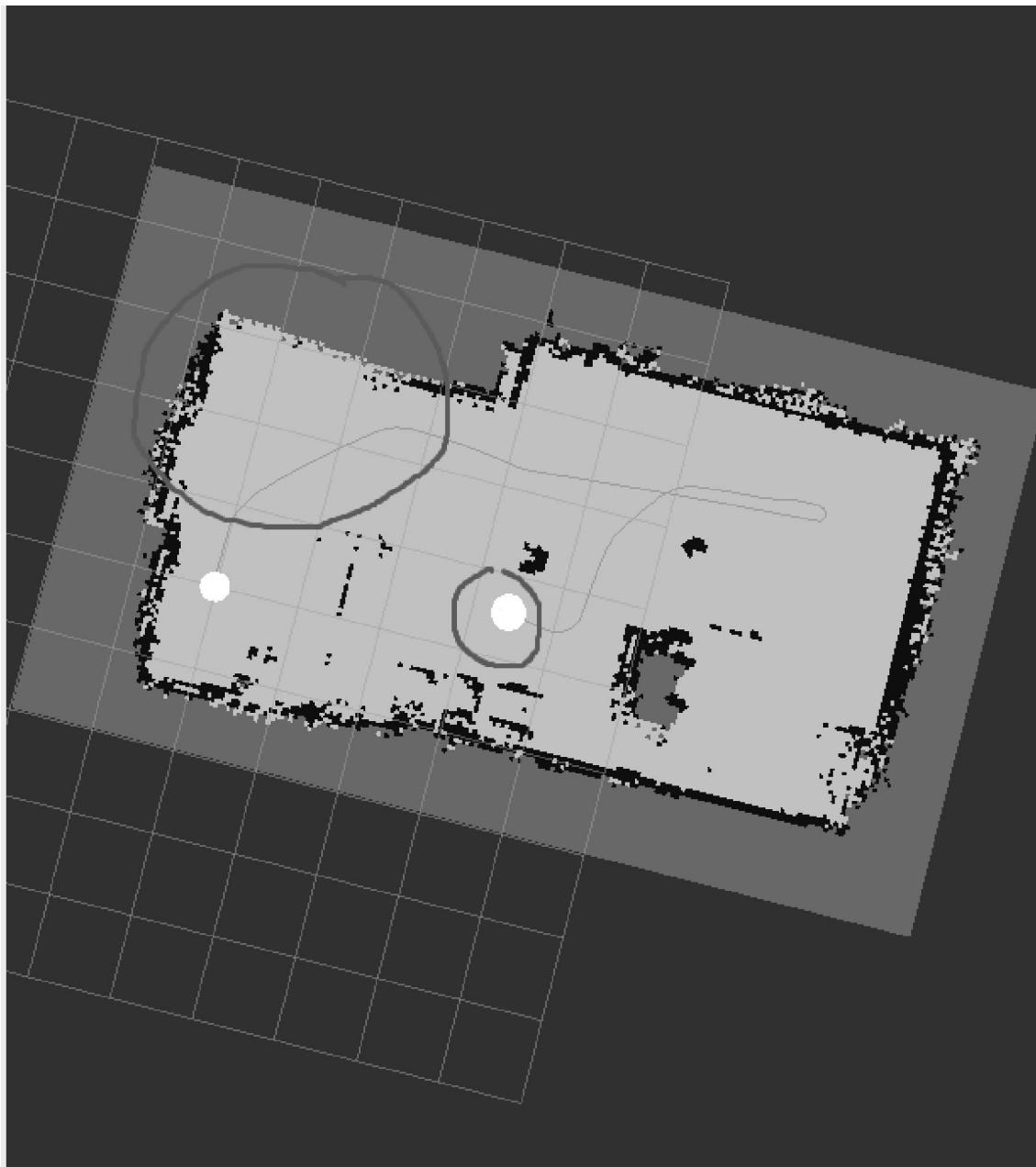
Tiếp theo, nhóm thu nhỏ khu vực di chuyển lại chỉ còn khu vực sảnh chính và chỉ di chuyển qua 1 lần thì kết quả cho ra chính xác hơn, hạn chế các điểm vỡ và lệch so với thực tế. Tại lần này, bản đồ vẫn còn hàng ghế phía trong tường và

thùng rác ở giữa sảnh rõ ràng hơn nhiều so với hình 54.



Hình 48: Mapping sảnh chính phòng 709

Tuy nhiên, trong quá mapping, nếu đường di chuyển của robot có xu hướng tạo thành một vòng lặp hoặc đi qua hoặc nhìn thấy một khu vực 2 lần thì khả năng cao bản đồ được xây dựng sẽ bị vỡ hoặc bị sai so với thực tế. Trong hình dưới, tại hình tròn màu trắng là vị trí ban đầu của robot và màu đỏ là vị trí hiện tại. Ngay tại khu vực được khoanh tròn màu đỏ là phần tường mà camera của robot quét qua được hai lần, lần đầu đi mới bắt đầu mapping và lần sau là ở vị trí hiện tại. Khu vực tường này có hiện tượng bị vỡ ảnh và biến mất phần giới hạn màu đen.



Hình 49: Robot "nhìn thấy" một khu vực hai lần

3.3.2 Khả năng tự hành trong thời gian dài

Tiếp theo, nhóm tiến hành thử nghiệm độ chính xác trong việc di chuyển tự hành của robot trong không gian với các tốc độ và gia tốc khác nhau để so sánh tính chính xác.

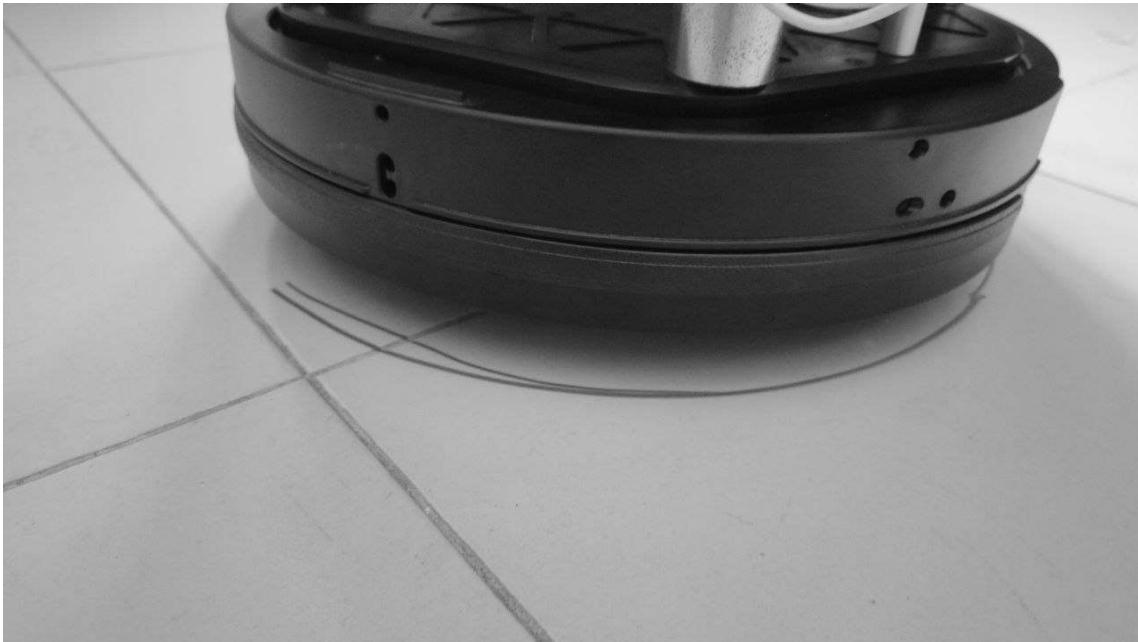
3.3.2.1 Chuẩn bị

Nhóm tiến hành thử nghiệm bằng cách cho robot di chuyển quanh khu vực sân phòng 709 với 3 đích đến ở 3 nơi khác nhau và có hướng khác nhau như hình dưới đây:



Hình 50: Khu vực thử nghiệm

Đầu tiên, robot sẽ di chuyển lần lượt đến 3 điểm để ghi lại vị trí thực tế bằng bút lông xanh nhưng hình dưới



Hình 51: Dán dấu vị trí Robot tại các điểm trên bản đồ

Tiếp theo, ta sẽ tùy chỉnh thông tin tốc độ và gia tốc của robot khi tự hành tại file *smoother.yaml* trong package *move_base*. Ta sử dụng các thông số sau trong file:

```
# Mandatory parameters
speed_lim_v: 0.8
speed_lim_w: 5.4

accel_lim_v: 1.0
accel_lim_w: 2.0
```

Hình 52: Các thông số tùy chỉnh

Trong đó, các thông số này có ý nghĩa:

- **speed_lim_v:** Đây là giới hạn tốc độ tuyến tính (linear velocity) của robot. Nó xác định tốc độ tối đa mà robot có thể di chuyển thẳng tới hoặc lùi lại theo trực trước và sau
- **speed_lim_w:** Đây là giới hạn tốc độ góc (angular velocity) của robot. Nó xác định tốc độ tối đa mà robot có thể xoay quanh trục z (trục quay).

- **accel_lim_v:** Đây là giới hạn gia tốc tuyến tính (linear acceleration) của robot. Nó xác định tốc độ tối đa mà robot có thể thay đổi tốc độ di chuyển tuyến tính trong một khoảng thời gian nhất định.
- **accel_lim_w:** Đây là giới hạn gia tốc tuyến tính (linear acceleration) của robot. Nó xác định tốc độ tối đa mà robot có thể thay đổi tốc độ di chuyển tuyến tính trong một khoảng thời gian nhất định.

3.3.2.2 Tiến hành thử nghiệm

Đầu tiên nhóm sẽ cho robot đi qua lần lượt 3 điểm được đánh dấu để ghi lại vị trí robot thực tế, bắt đầu từ điểm 1 đến điểm 3 và sau đó quay trở lại điểm 1 và đi lần lượt đến điểm 3 để ghi lại độ sai lệch so với vị trí thực tế. Sau mỗi lần thử nghiệm, nhóm sẽ hiệu chỉnh lại vận tốc và gia tốc tuyến tính, vận tốc và gia tốc góc để tìm ra tác động của chúng đến việc vận hành và xây dựng lại bản đồ mới cho mỗi lần để hạn chế sai lệch do di chuyển quá nhiều lần đến cùng 1 ví trí trên bản đồ.

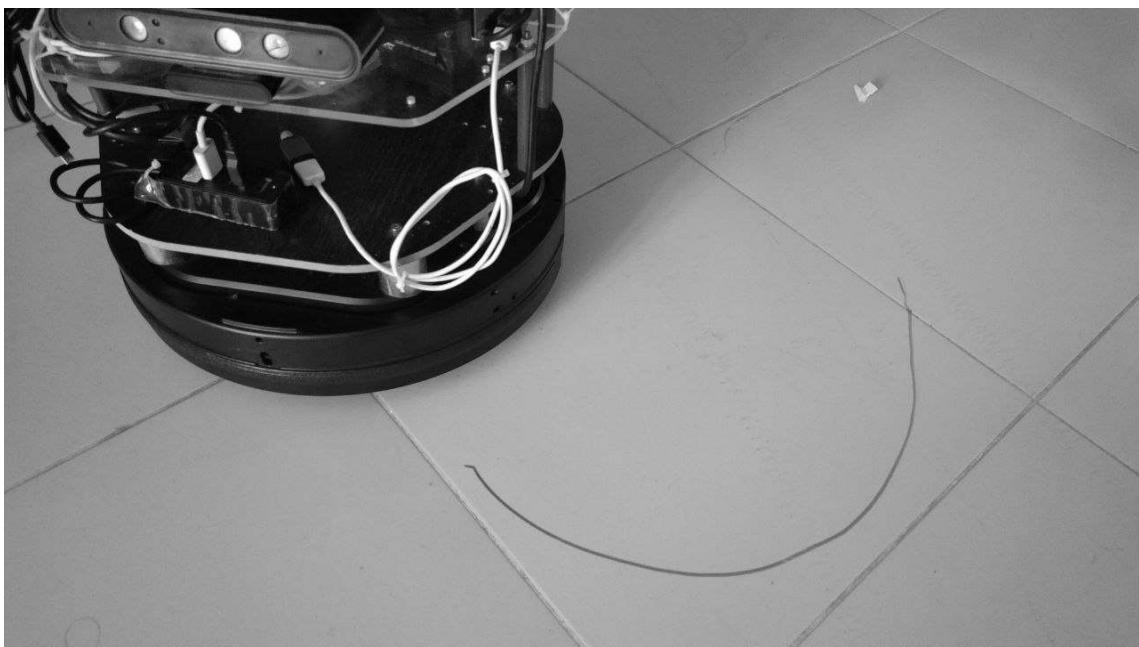
- Vận tốc nhỏ và gia tốc nhỏ

```
# Mandatory parameters
speed_lim_v: 0.3
speed_lim_w: 1.0

accel_lim_v: 0.1
accel_lim_w: 0.5|
```

Hình 53: Vận tốc nhỏ và gia tốc nhỏ

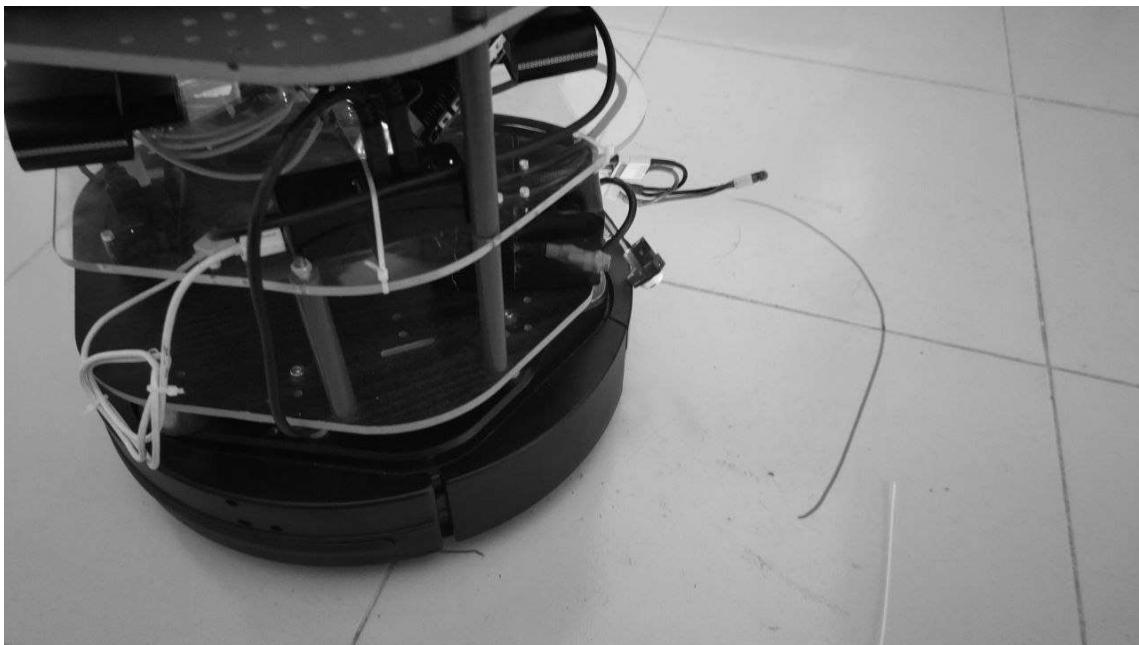
Tại thí nghiệm đầu tiên, nhóm sẽ cho robot di chuyển với vận tốc di chuyển tối đa là 0.3 m/s và gia tốc là 0.1, vận tốc góc tối đa là 1 radian/s và gia tốc là 0.5. Lúc này, khi bắt đầu di chuyển thì robot sẽ tăng tốc dần lên đến tối đa là 0.3 m/s, tương tự với việc xoay của robot. Nhóm thu được kết quả sau khi di chuyển qua 3 điểm lần lượt.



Hình 54: Thí nghiệm 1: vị trí 1



Hình 55: Thí nghiệm 1: vị trí 2



Hình 56: Thí nghiệm 1: vị trí 3

Trong thí nghiệm này, robot bị lệch khỏi vị trí ban đầu với khoảng cách khoảng 15-20 cm tính từ tâm tại mỗi vị trí. Khi cho robot di chuyển thêm từ vị trí số 3 đến lại vị trí số 1 thì ta sẽ thấy khoảng cách lệch của robot so với ban đầu càng lớn hơn khoảng gần 30 cm.



Hình 57: Thí nghiệm 1: vị trí 1 lần 2

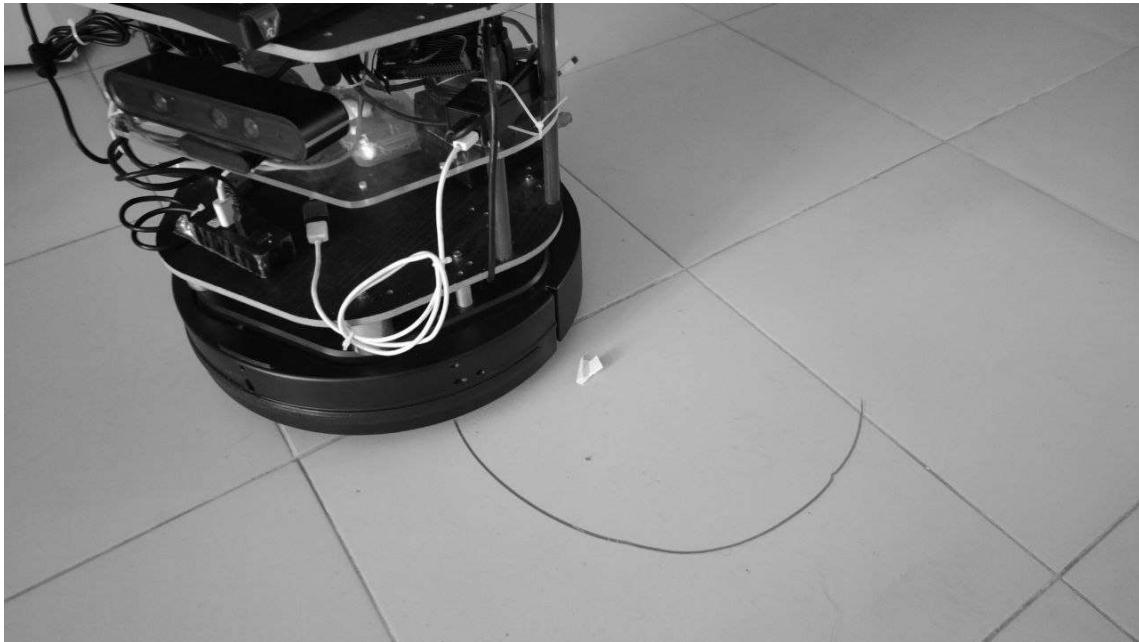
- Vận tốc nhỏ và gia tốc lớn

```
# Mandatory parameters
speed_lim_v: 0.3
speed_lim_w: 1.0

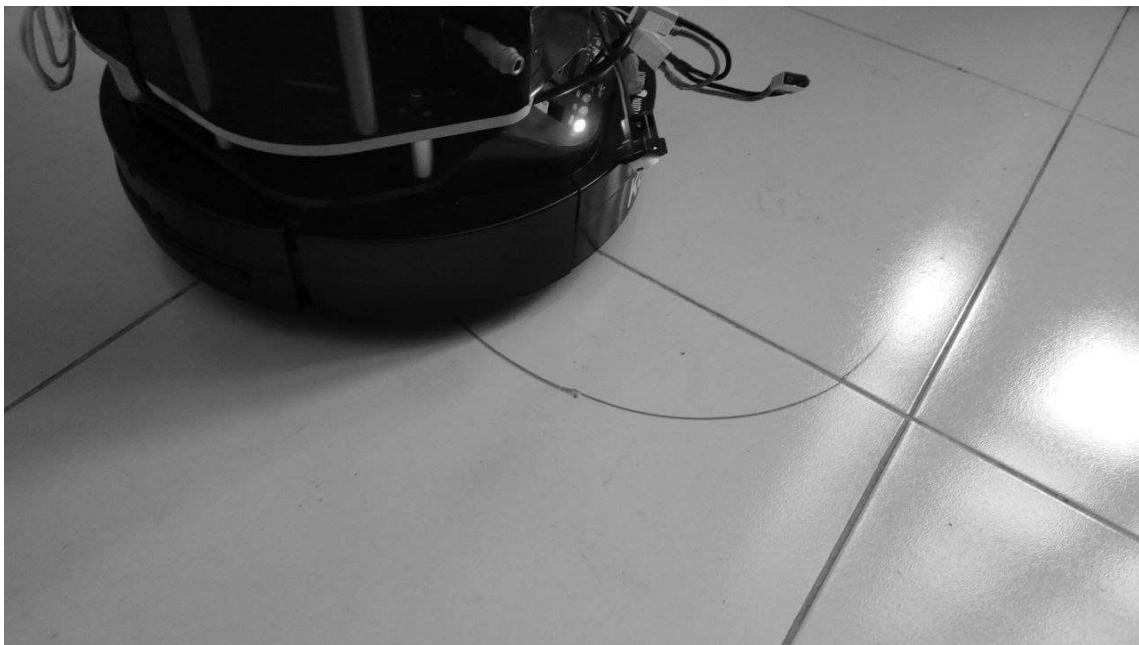
accel_lim_v: 0.5
accel_lim_w: 0.5
```

Hình 58: Vận tốc nhỏ và gia tốc lớn

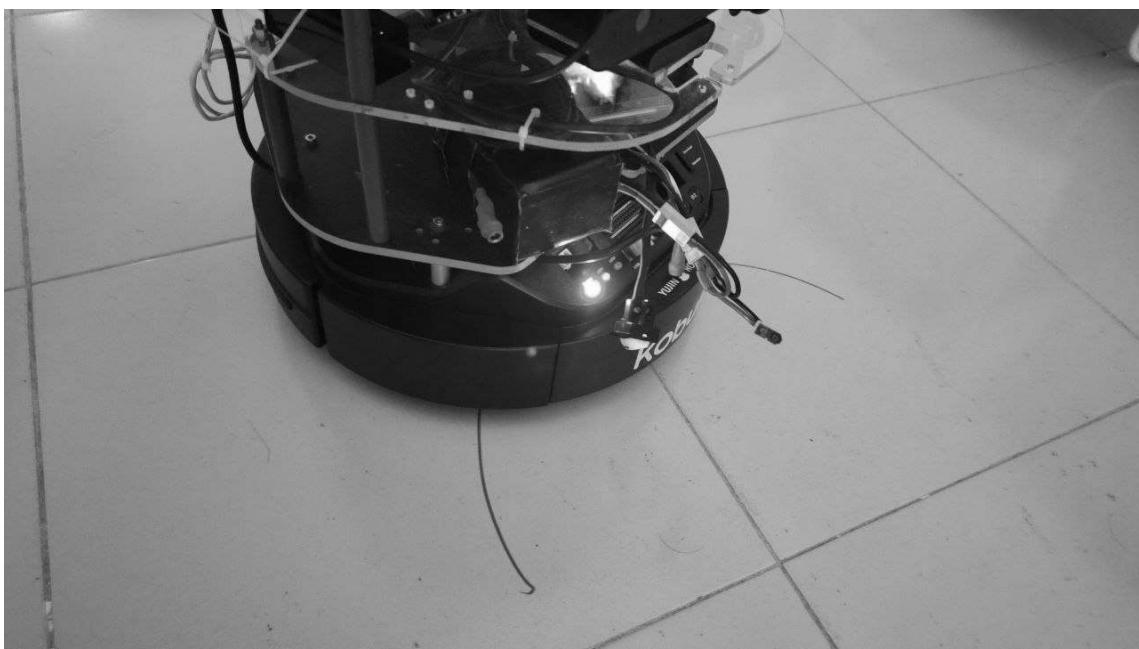
Trong thí nghiệm tiếp theo, nhóm chỉ tăng giá trị gia tốc tuyến tính cao hơn tốc độ tối đa, có nghĩa là tại thời điểm di chuyển, robot sẽ tăng tốc đến tối đa là 0.3 m/s ngay trong giây đầu tiên. Các kết quả thu được:



Hình 59: Thí nghiệm 2: vị trí 1

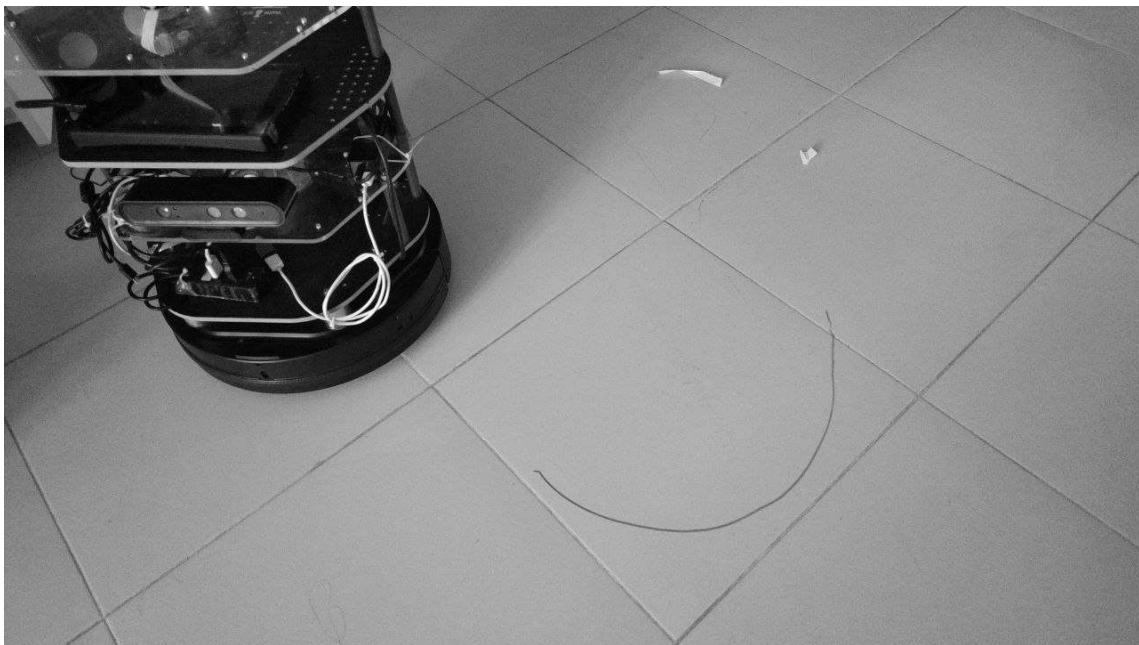


Hình 60: Thí nghiệm 2: vị trí 2



Hình 61: Thí nghiệm 2: vị trí 3

Kết quả thu được cho thấy độ lệch so với vị trí ban đầu đã giảm đi, chỉ còn khoảng 15-20 cm. Kể cả khi nhóm cho robot di chuyển từ vị trí 3 quay lại vị trí 1 khoảng cách lệch cũng đã giảm đi một phần so với thí nghiệm 1.



Hình 62: Thí nghiệm 2: vị trí 1 lần 2

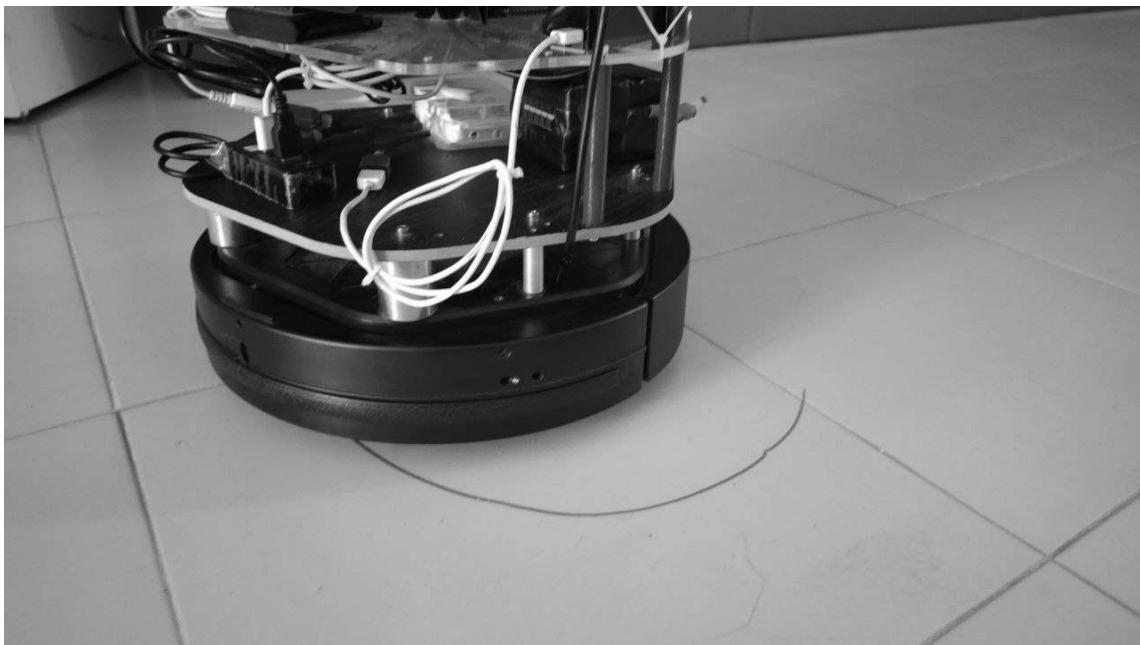
- Vận tốc lớn và gia tốc lớn

```
# Mandatory parameters
speed_lim_v: 0.8
speed_lim_w: 5.4

accel_lim_v: 1.0
accel_lim_w: 2.0
```

Hình 63: Vận tốc lớn và gia tốc lớn

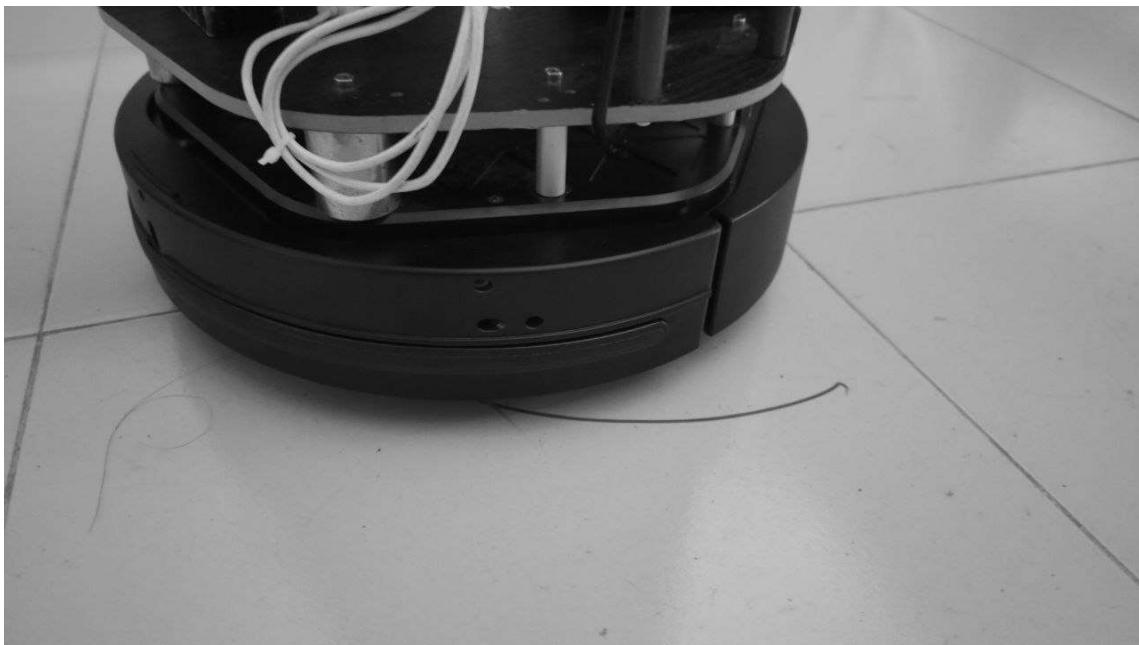
Qua 2 thí nghiệm trên, nhóm lựa chọn tăng tốc độ di chuyển và tốc độ xoay với gia tốc di chuyển lớn hơn vận tốc di chuyển để kiểm tra rằng liệu di chuyển nhanh hơn có làm tăng độ chính xác khi di chuyển không. Nhóm thi được kết quả



Hình 64: Thí nghiệm 3: vị trí 1



Hình 65: Thí nghiệm 1: vị trí 2



Hình 66: Thí nghiệm 1: vị trí 3

Kết quả thu được là độ lệch của robot đã giảm nhiều so với 2 thí nghiệm trước, rơi vào khoảng 4-7 cm. Lúc này, có thể thấy việc để robot di chuyển nhanh hơn để đến 1 mục tiêu thì độ sai lệch giữa bản đồ và thực tế được giảm đi.

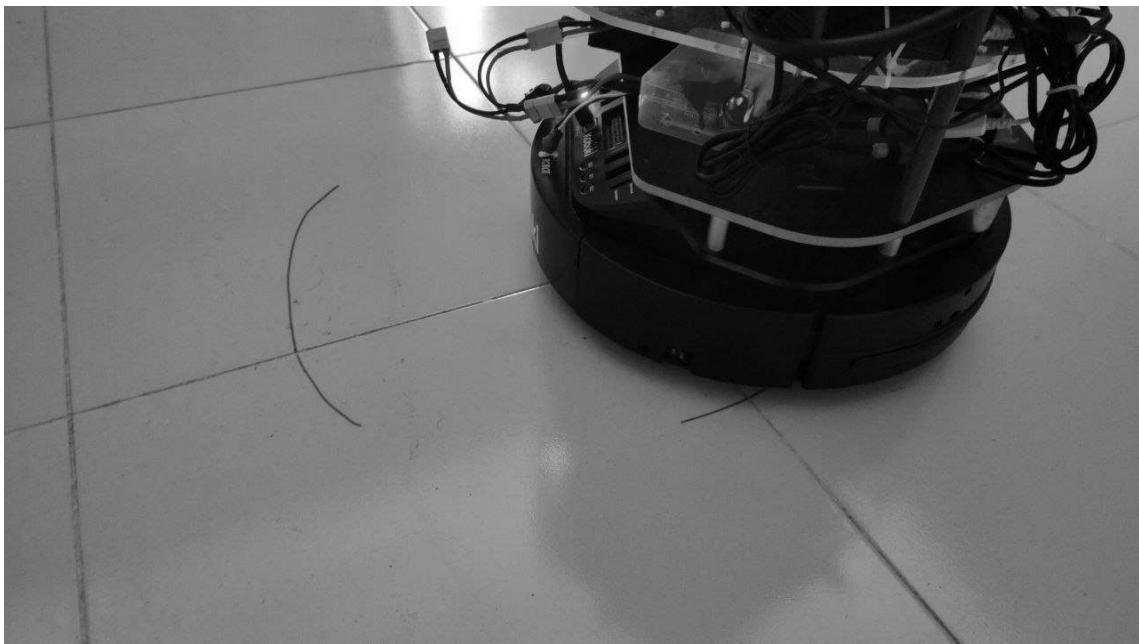
Nhóm thực hiện đo lần 2 với các thông số trong thí nghiệm này, robot sẽ đi lần lượt từ vị trí 1 đến 3 và thu được kết quả sau:



Hình 67: Thí nghiệm 3: vị trí 1 lần 2



Hình 68: Thí nghiệm 1: vị trí 2 lần 2



Hình 69: Thí nghiệm 1: vị trí 3 lần 2

Việc di chuyển lặp lại các vị trí sau mỗi vòng sẽ làm tăng độ lệch so với ban đầu. Tiếp theo nhóm cho robot di chuyển đến 1 vị trí khác với 3 điểm đã định trước trong map thì quan sát được vị trí trong map và thực tế khác nhau như thế nào. Trong 2 hình dưới, hình tròn màu đỏ là vị trí thực tế của robot, bị lệch đi so với hình robot trên bản đồ



Hình 70: Vị trí khác trên bản đồ lần 1

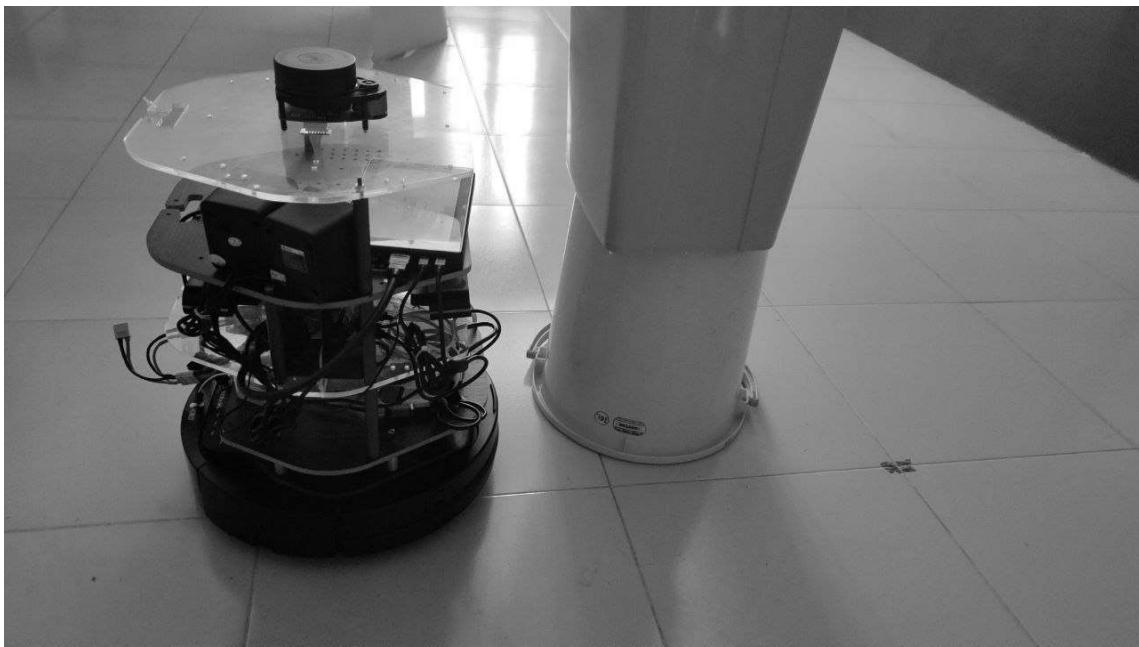


Hình 71: Vị trí thực tế lần 1

Khi cho robot di chuyển thêm 1 nữa để quay về vị trí số 1 thì robot di chuyển sai và gặp vấn đề về hoạch định đường đi. Trong khi trên bản đồ thì robot có thể di thẳng nhưng trên thực tế thì gặp phải vật cản nên khôi phục định đường đi gặp vấn đề tìm hướng đi khác cho robot, điều này làm cho robot dừng hẳn việc di chuyển và báo lỗi. Ở hình dưới, hình tròn màu đỏ là vị trí thật của robot



Hình 72: Vị trí khác trên bản đồ lần 2



Hình 73: Vị trí thực tế lần 2

4 Kết luận

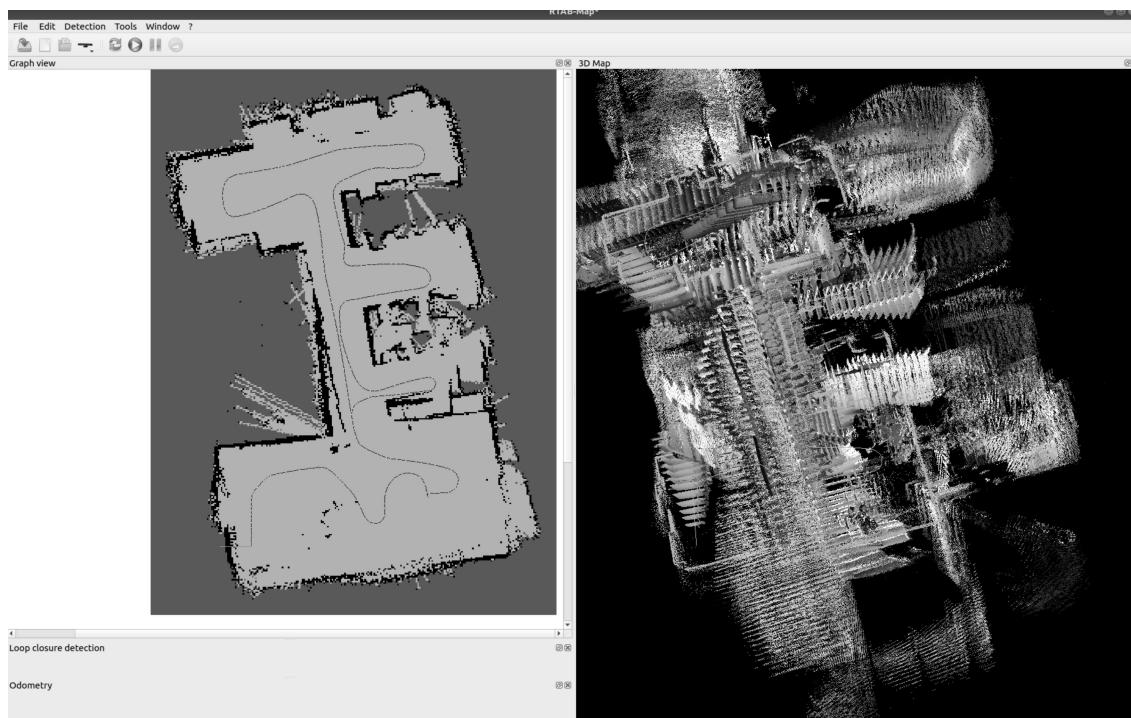
4.1 Kết quả đạt được

Sau khi nghiên cứu và thực hiện đề tài, nhóm đã cơ bản đã hoàn thành nền tảng robot dịch vụ gồm: mô hình robot có thể tự hành, ứng dụng giao tiếp với người dùng thông qua màn hình có trên robot và một ứng dụng có thể điều khiển robot thông qua Web với các chức năng:

- Tạo bản đồ 2D và 3D khu vực robot sẽ làm việc.
- Định vị robot trong môi trường
- Điều hướng đến vị trí được chỉ định trên bản đồ
- Tránh các rào chắn cố định có độ cao khác nhau.
- Giám sát và điều khiển robot qua giao diện web

4.1.1 Khả năng xây dựng bản đồ

Robot có khả năng xây dựng bản đồ trong một khu vực sau một khoảng thời gian điều khiển bằng tay và trích xuất hình ảnh 2D và 3D



Hình 74: Hình ảnh 2D và 3D của bản đồ

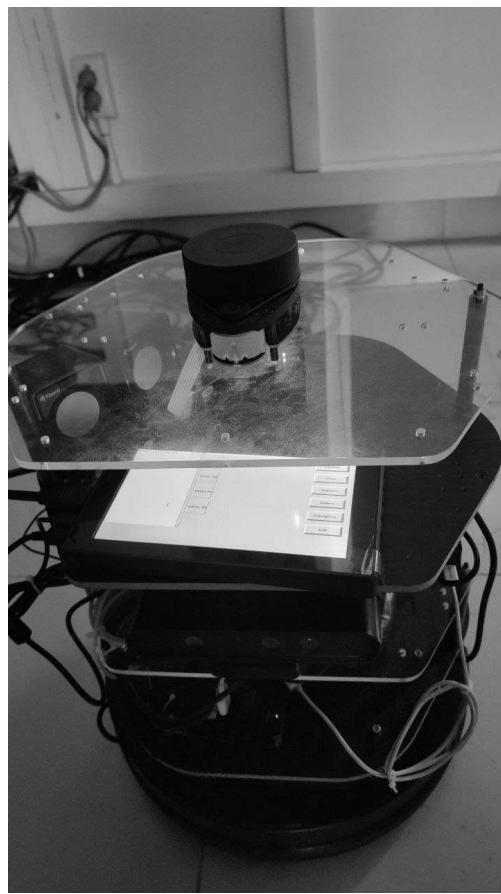
4.1.2 Khả năng tự hành

Trong quá trình tự hành, sự phối hợp của cảm biến Kinect và Lidar, khả năng định vị của robot được cải thiện hơn so với việc sử dụng chỉ một loại cảm biến.

Khi robot đến vị trí mà hình ảnh nghèo về đặc trưng (poor feature) và giá trị so sánh với tập ảnh trong bộ nhớ không gian làm việc thấp hơn ngưỡng, khả năng định vị dựa trên visual slam trở nên không hiệu quả. Nhờ vào sự hỗ trợ từ Lidar, robot sử dụng phương pháp ICP để tìm kiếm các node gần giống trên bản đồ để tái định vị. cho phép tạo ra một bức tranh về các vật cản đầy đủ hơn, giúp cho robot có khả năng hoạch định đường đi tốt hơn và tránh va phải vật cản.

4.1.3 Giao tiếp với người dùng

Robot có khả năng giao tiếp với người dùng thông qua mạch Raspberry Pi được trang bị màn hình và loa. Thiết bị này sử dụng giao thức serial để kết nối với ROS và cho phép người dùng tương tác với robot thông qua các lệnh được hiển thị trên màn hình. Việc sử dụng Raspberry Pi giúp cho quá trình giao tiếp giữa robot và người dùng trở nên dễ dàng hơn và đảm bảo tính linh hoạt trong việc thay đổi các chức năng và lệnh điều khiển của robot.



Hình 75: Giao tiếp qua màn hình và loa

4.1.4 Điều khiển robot thông qua Web

Sau nhiều nghiên cứu, nhóm đã tạo ra một ứng dụng web để tương tác với robot Kobuki. Ứng dụng này có nhiều ưu điểm như khả năng vị trí và mapping, hiển thị

rõ ràng hình ảnh camera của robot, cho phép người dùng điều khiển robot bằng bàn phím và điều chỉnh tốc độ phù hợp. Tuy nhiên, ứng dụng web vẫn còn một số điểm yếu như chất lượng hình ảnh của bản đồ 2D thấp hơn so với Rviz và chưa hoàn thiện được các chức năng như thêm các trạm để tương tác với robot bằng giọng nói như kế hoạch ban đầu.



Hình 76: Giao diện Web

4.2 Những hướng phát triển

Để phát triển robot trong tương lai, nhóm đề xuất một số điểm sau:

• Phía Robot

- Bổ sung thêm các thành phần để robot có thể hiệu chỉnh vị trí của mình và tăng khả năng phát hiện loop closure, có thể sử dụng giải pháp dựa trên sóng wifi
- Sử dụng các nền tảng máy tính nhúng khác để tăng khả năng xử lý và triển khai các mô hình trí tuệ nhân tạo giúp robot hoạt động tốt hơn.
- Thiết kế khung robot chịu trọng tải lớn hơn và cảm biến lidar có tầm quét rộng hơn

• Phía server

- Mở rộng để quản lý nhiều kết nối và nhiều robot trong hệ thống, xây dựng cơ sở dữ liệu lưu trữ thông tin, phát triển tính năng cập nhật chương trình từ xa (OTA) cho robot và thiết kế giải thuật chỉnh sửa map 2D và 3D thu được từ robot để hiển thị lên giao diện người dùng.
- Đối với giao diện web, nhóm đề xuất thiết kế các trang đăng nhập và bổ sung hiển thị hình ảnh 3D được xây dựng từ robot.

Tài liệu tham khảo

- [1] L. D. Binh, D. H. Long, and L. Nguyen, “Prototyping an autonomous service robot,” 2022.
- [2] “Gazebo tutorials,” [Online]. Available: <https://classic.gazebosim.org/tutorials>.
- [3] introlab, “Rtabmap ros,” [Online]. Available: <http://introlab.github.io/rtabmap/>.
- [4] “Kobuki tutorials,” [Online]. Available: <http://wiki.ros.org/kobuki/Tutorials>.
- [5] T. T. Loc, “Ứng dụng visual slam trên robot phục vụ,” 2020.
- [6] minhducse, “Superh-lab-kobuki-slam,” [Online]. Available: <https://github.com/minhducse/superh-lab-kobuki-slam>.
- [7] “Move_{base},” [Online]. Available: http://wiki.ros.org/move_base#Action_API.
- [8] “Robot,” 2022. [Online]. Available: <https://en.wikipedia.org/wiki/Robot>.
- [9] “Ros tutorials,” [Online]. Available: <http://wiki.ros.org/ROS/Tutorials>.
- [10] shivachandrachary, “Introduction to 3d slam with rtabmap,” [Online]. Available: <https://shivachandrachary.medium.com/introduction-to-3d-slam-with-rtab-map-8df39da2d293>.
- [11] G. Slam, “What is slam (simultaneous localization and mapping)?,” [Online]. Available: <https://geoslam.com/what-is-slam/>.
- [12] “Understanding the ros file system level,” [Online]. Available: <https://subscription.packtpub.com/book/iot-&-hardware/9781783551798/1/ch01lvl1sec11/understanding-the-ros-file-system-level>.
- [13] yujinrobot, “Kobuki desktop,” [Online]. Available: https://github.com/yujinrobot/kobuki_desktop.