

BỘ CÔNG THƯƠNG
TRƯỜNG ĐẠI HỌC CÔNG NGHIỆP TP.HCM

KHOA CÔNG NGHỆ ĐIỆN



ĐỒ ÁN TỐT NGHIỆP (CHUYÊN NGÀNH)

**XÂY DỰNG VÀ ĐIỀU KHIỂN ROBOT TỰ HÀNH TRONG
KHO XƯỞNG**

Giảng viên hướng dẫn: Hoàng Đình Khôi

Sinh viên thực hiện:

Họ và tên	MSSV	Lớp
Phan Quốc Bửu	19524931	DHDKT15ATT
Nguyễn Văn Trung	19523991	DHDKT15ATT
Nguyễn Văn Dũng	19511501	DHDKT15ATT
Phan Anh Hào	19492541	DHDKT15ATT

Tp Hồ Chí Minh, Ngày 30 Tháng 5 năm 2023

PHIẾU GIAO NHIỆM VỤ KHÓA LUẬN TỐT NGHIỆP

1. Họ và tên sinh viên/ nhóm sinh viên được giao đề tài:

STT	Họ và tên	MSSV	Lớp
1	Phan Quốc Bửu	19524931	DHDKTD15ATT
2	Nguyễn Văn Trung	19523991	DHDKTD15ATT
3	Nguyễn Văn Dũng	19511501	DHDKTD15ATT
4	Phan Anh Hào	19492541	DHDKTD15ATT

2. Tên đề tài:

ROBOT TỰ VẬN HÀNH VẬN CHUYỂN HÀNG HÓA TRONG KHO HÀNG

3. Nhiệm vụ (Nội dung và số liệu ban đầu):

- ✓ Tìm hiểu tổng quan về xe tự hành, hệ điều hành Ubuntu và Robot Operating System (ROS)
- ✓ Lựa chọn các thiết bị cho xe tự hành
- ✓ Thiết kế 3D vỏ robot cho xe bằng phần mềm SolidWork
- ✓ Lập trình vi xử lý Arduino Mega 2560
- ✓ Thiết kế giao diện điều khiển bằng Matlab
- ✓ Thiết lập điều khiển không dây từ xa qua Bluetooth
- ✓ Thiết lập định vị và điều khiển tự động điều hướng cho xe

4. Kết quả dự kiến (Tóm tắt kết quả dự kiến đạt được):

- ✓ Nắm được các kiến thức ROS
- ✓ Robot được làm ra đúng với thiết kế đã định
- ✓ Robot di chuyển tốt có thể điều khiển từ xa thông qua Bluetooth
- ✓ Ros có thể điều khiển tự động cho Robot
- ✓ Giao tiếp người – máy thông qua giao diện Matlab Gui

Tp. Hồ Chí Minh, ngày ... tháng ... năm 2023

Giảng viên hướng dẫn

Hoàng Đình Khôi

Trưởng bộ môn

Sinh viên

Phan Quốc Bửu

NHẬN XÉT CỦA GIẢNG VIÊN HƯỚNG DẪN

Điểm đánh giá: .../10

GV hướng dẫn

Hoàng Đinh Khôi

MỤC LỤC

PHIẾU GIAO NHIỆM VỤ KHÓA LUẬN TỐT NGHIỆP	I
NHẬN XÉT CỦA GIÁNG VIÊN HƯỚNG DẪN.....	II
MỤC LỤC	III
DANH MỤC CÁC HÌNH ẢNH.....	VI
DANH MỤC CÁC BẢNG.....	VIII
LỜI MỞ ĐẦU	IX
CHƯƠNG 1: TỔNG QUAN.....	1
1.1 Đặt vấn đề	1
1.2 Lịch sử phát triển của xe tự hành	2
1.3 Mục tiêu đề tài.....	4
1.4 Đối tượng và phạm vi nghiên cứu.....	4
CHƯƠNG 2: MÔ HÌNH ĐỘNG HỌC XE ROBOT	6
2.1 Thiết kế bản vẽ.....	6
2.2 Hệ thống di chuyển	7
2.3 Hệ thống cơ khí nâng hạ hành hóa.....	8
2.4 Phần khung Robot.....	10
2.5 Bộ nguồn Pin cung cấp	12
2.6 Hoàn thiện phần cứng	14
CHƯƠNG 3: XÂY DỰNG THUẬT TOÁN ĐIỀU KHIỂN.....	15
3.1 Một số khái niệm quan trọng	15
3.1.1 Hệ điều hành – ROS	15
3.1.2 Vi xử lý	16
3.1.3 Mạch Driver.....	17
3.1.4 Lidar (RPLIDAR S2)	18
3.1.5 Động cơ và encoder	18
3.2 Định vị và điều hướng (Navigation stack [6])	19
3.2.1 Chuyển đổi cấu hình	20
3.2.2 Xử lý thông tin cảm biến	21
3.2.3 Thông tin về vị trí và hướng của robot (Odometry information)	23
3.3 Các thuật toán được sử dụng.....	26
3.3.1 Bộ điều khiển PID	26
3.3.2 Tạo lập bản đồ và định vị Robot trong bản đồ (SLAM) [11].....	27

3.3.3 Bộ lọc hạt (Practice Filter)	28
3.3.4 Thuật toán AMCL [7].....	30
3.3.5 Thuật toán Gmapping [10]	32
3.3.6 Thuật toán costmap_2D [9]:.....	35
3.3.7 Chuyển động move_base.....	36
3.3.8 Thuật toán Teb Local Planner [14].....	38
CHƯƠNG 4: ROBOT XE NÂNG TỰ VẬN HÀNH TRONG KHO XUỐNG44	
4.1 Tổng quan về Robot.....	44
4.1.1 Bộ điều khiển trung tâm	44
4.1.2 Cảm biến.....	45
4.1.3 Vi xử lý và các thiết bị khác	46
4.2 Lập trình tích hợp hệ thống.....	47
4.2.1 Lập trình vi điều khiển – Arduino Mega 2560	47
4.2.2 Cài đặt các công cụ cần thiết trên mini PC.....	49
4.2.3 Thiết lập kết nối điều khiển	50
4.2.4 Xây dựng bản đồ (mapping).....	51
4.2.5 Điều hướng xe trên bản đồ	51
4.3 Thiết kế giao diện cho Robot	52
4.3.1 Ros Mobile	53
4.3.2 Matlab Gui.....	55
4.3.3 Giao diện và giao thức truyền nhận thông tin	58
CHƯƠNG 5: KẾT QUẢ THỰC NGHIỆM.....63	
5.1 Quá trình xây dựng bản đồ.....	63
5.2 Định vị robot trên bản đồ	64
5.3 Khả năng vận hành của bộ điều khiển PID:.....	64
5.4 Khả năng tự hành và né tránh vật thể di động	64
5.5 Khả năng giao nhận hàng.....	66
5.6 Giao diện điều khiển và giám sát	67
5.6.1 Điều khiển bằng Ros Mobile	67
5.6.2 Điều khiển bằng Bluetooth	68
5.6.3 Điều khiển bằng Matlab GUI	68
CHƯƠNG 6: TỔNG KẾT NHẬN XÉT VÀ HƯỚNG PHÁT TRIỂN.....71	
6.1 Tổng kết đề tài.....	71
6.2 Hướng phát triển	71

TÀI LIỆU THAM KHẢO.....	73
PHỤ LỤC	75
LỜI CẢM ƠN	95

DANH MỤC CÁC HÌNH ẢNH

Hình 1. 1: Hệ thống kho hàng thông minh	1
Hình 1. 2: Robot vận chuyển và phân loại hàng hóa Kiva của Amazon	2
Hình 1. 3: Những robot vận chuyển hàng hóa tự hành đời đầu	2
Hình 1. 4: Robot nghiên cứu và tìm kiếm sự sống trên sao Hỏa của NASA	3
Hình 2. 1: Logo Autocad	6
Hình 2. 2: Logo SolidWorks	7
Hình 2. 3: Bản vẽ kích thước mặt trước cơ cấu nâng 2D và 3D	8
Hình 2. 4: Bản vẽ kích thước 2 mặt bên cơ cấu nâng 2D và 3D	9
Hình 2. 5: Bản vẽ 2D và 3D kích thước 2 mặt bên khung xe	10
Hình 2. 6: Bản vẽ 2D và 3D kích thước mặt sau khung xe	11
Hình 2. 7: Bản vẽ 2D và 3D kích thước mặt trước khung xe	11
Hình 2. 8: Bản vẽ 2D và 3D kích thước mặt trên khung xe	11
Hình 2. 9: Hộp kỹ thuật	12
Hình 2. 10: Bộ nguồn pin 24V	13
Hình 2. 11: Bản vẽ 3D tổng thể robot	14
Hình 2. 12: Khung xe trước và sau khi lắp đặt	14
Hình 3. 1: Sơ đồ khái quát Input/ Output	15
Hình 3. 3: Hệ sinh thái đa ngôn ngữ của Ros	16
Hình 3. 4: Modul arduino Mega 2560	16
Hình 3. 5: Sơ đồ nguyên lý mạch driver điều khiển step motor	17
Hình 3. 6: Sơ đồ nguyên lý mạch driver điều khiển động cơ DC	18
Hình 3. 7: Sơ đồ kết nối động cơ DC và encoder với arduino	18
Hình 3. 8: Tổng quan về navigation stack	19
Hình 3. 9: ví dụ minh họa cho robot	20
Hình 3. 10: Cấu trúc cây biến đổi khung tọa độ của robot tự hành và sự biến đổi giữa các khung tọa độ	21
Hình 3. 12: Ví dụ về xác định vị trí robot trong tọa độ "odometry"	23
Hình 3. 13: Cấu trúc bộ mã hóa quãng đường đi được và đĩa (encoder)	24
Hình 3. 14: Hệ thống PID	26
Hình 3. 15: Bản đồ thu được từ gói gmapping	27
Hình 3. 16: Ví dụ minh họa	28
Hình 3. 17: Ví dụ về hoạt động tính toán bị trí của xe	29
Hình 3. 18: Hàm mật độ xác suất vị trí của xe (phân phối Gauss), xe có thể ở các vị trí trong phân phối xác suất	30
Hình 3. 19: Ví dụ về robot trong một bản đồ gồm 4 bức tường (bên phải là khởi tạo ngẫu nhiên vị trí tượng trưng)	31
Hình 3. 20: Khu vực co thể có vị trí robot sau khi lược bỏ các hạt có thông tin laser sai và khởi tạo lại các hạt trong các khu vực này	31
Hình 3. 21: Di chuyển robot đồng thời di chuyển tất cả các hạt	31
Hình 3. 22: Qua nhiều lần cập nhập thì mật độ hạt sẽ tập trung tại một khu vực	32
Hình 3. 23: Mô tả về quỹ đạo di chuyển của robot khi có bản đồ chi phí. Vì có bản đồ chi phí nên robot sẽ tránh tiếp xúc gần với các vật cản, giúp nó di chuyển an toàn và ta có thể dễ dàng tính toán	35
Hình 3. 24: Ví dụ về cost map trong ROS	36
Hình 3. 25: Ví dụ về đường di chuyển và quỹ đạo chuyển động của robot	37
Hình 3. 26 Đường dẫn toàn cục của robot (thuật toán Dijkstra, A*)	37
Hình 4. 1: Bộ điều khiển trung tâm (trước, sau)	44
Hình 4. 2 Cảm biến RPLIDAR S2	45
Hình 4. 3: Mặt trước mạch điều khiển động cơ DC	47
Hình 4. 4: Mặt sau mạch điều khiển động cơ DC	47

Hình 4. 5: Mạch điều khiển động cơ bước TB6600	48
Hình 4. 6: Công tắc hành trình	48
Hình 4. 7: Mạch bluetooth HC-05	48
Hình 4. 8: Giao diện điều khiển robot qua bluetooth	48
Hình 4. 9 Kết nối Roserial	49
Hình 4. 10 Logo Ubuntu 20.04 LTS.....	49
Hình 4. 11: Logo Arduino IDE.....	50
Hình 4. 12: Logo Rviz	50
Hình 4. 13: Sơ đồ khái các bước dựng bản đồ	51
Hình 4. 14: Quá trình mô phỏng và chọn tập mẫu để cho ra kết quả tốt nhất của robot	52
Hình 4. 15: Sự liên kết giữa con người và robot	52
Hình 4. 16: Ứng dụng Ros Mobile trên CH-Play	53
Hình 4. 17: Giao diện kết nối IP của Ros Mobile	54
Hình 4. 18: Tạo nút điều khiển trên Ros Mobile	54
Hình 4. 19: Cửa sổ câu lệnh khởi tạo Guide.....	55
Hình 4. 20: Tùy chọn tạo Gui	56
Hình 4. 21: Cửa sổ chính tạo GUI	56
Hình 4. 22: Cửa sổ cài chỉnh các đối tượng trong giao diện	57
Hình 4. 23: Gọi hàm Callback	57
Hình 4. 24: Bảng cài đặt thông số của các đối tượng trong giao diện.....	58
Hình 4. 25: Giao diện cài đặt cho robot.....	59
Hình 4. 26: Tạo file lưu trữ dữ liệu	59
Hình 4. 27: Giao diện điều khiển robot	60
Hình 4. 28: Lưu đồ và phương thức truyền nhận	62
Hình 5. 1 Bản đồ xây dựng được ở phòng X5.12	63
Hình 5. 2 Robot tự quy hoạch quãng đường	64
Hình 5. 3: Môi trường thực nghiệm.....	65
Hình 5. 4: Map trên Rviz quét được vật cản tĩnh và tự hoạch định đường di chuyển.....	65
Hình 5. 5: robot tự né vật cản tĩnh.....	65
Hình 5. 6: Trường hợp có vật cản di động.....	65
Hình 5. 7: Robot tự quy hoạch và di chuyển theo tuyến đường mới khi gặp vật cản di động	65
Hình 5. 8: Robot đến địa điểm nhân hàng và nâng ballet chứa hàng	66
Hình 5. 9: Robot đến địa điểm trả hàng và hạ thanh nâng	66
Hình 5. 10: Giao diện Ros Mobile điều khiển robot	67
Hình 5. 11: Giao diện điều khiển robot bằng bluetooth	68
Hình 5. 12: Giao diện giao tiếp với máy tính trung tâm và robot	69
Hình 5. 13: Màn hình điều khiển robot từ máy tính trung tâm.....	70

DANH MỤC CÁC BẢNG

Bảng 1: Thông số kích thước cơ bản của robot	7
Bảng 2: Các thành phần trong hệ thống di chuyển của robot	8
Bảng 3: Các thành phần trong hệ thống cơ khí nâng hạ	10
Bảng 4: Thông số kỹ thuật bộ nguồn.....	13
Bảng 5: Hệ thống xử lý và các ngõ vào và ra của robot.....	15
Bảng 6: Bảng thông số cấu trúc một messenger cho cảm biến	22
Bảng 7: Bảng thông số cấu trúc odometry của robot	25
Bảng 8: Một số thành phần khác của robot	46

LỜI MỞ ĐẦU

Trong ngành công nghiệp, robot xe nâng tự hành đang trở thành giải pháp hiệu quả để tăng cường năng suất và giảm chi phí trong quá trình vận chuyển hàng hóa trong kho xưởng. Với khả năng tự động di chuyển, phát hiện và tránh vật cản trong kho xưởng một cách chính xác và nhanh chóng, robot xe nâng tự hành có thể giải quyết được các thách thức trong quản lý và vận hành kho xưởng. Đặc biệt, khi kết hợp với các công nghệ tiên tiến như LIDAR và hệ điều hành ROS, robot xe nâng tự hành càng trở nên thông minh và linh hoạt hơn trong các môi trường khác nhau.

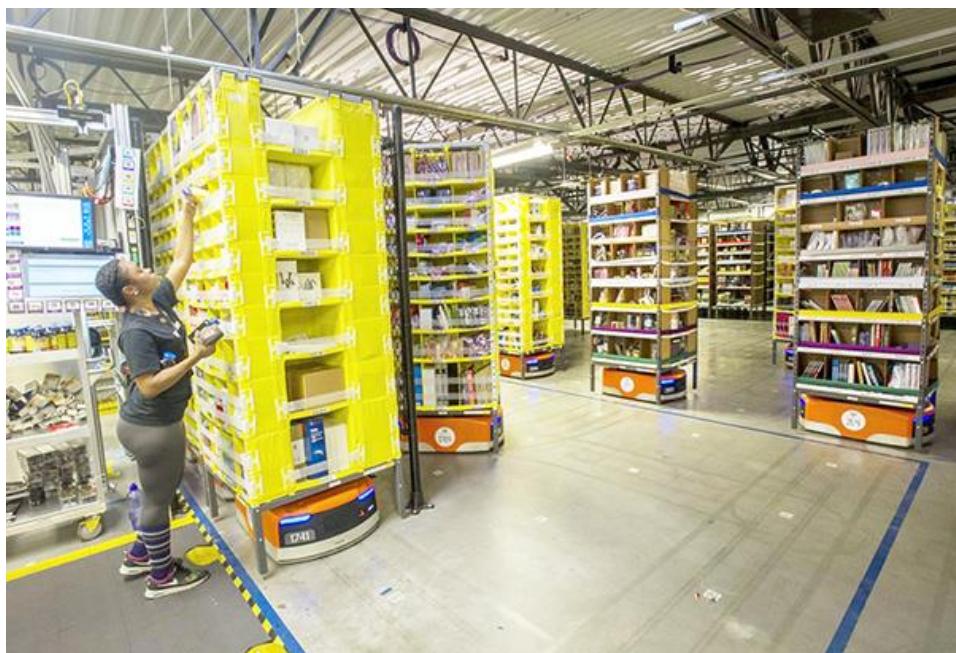
Hệ điều hành ROS cung cấp một khung công nghệ mạnh mẽ và linh hoạt để phát triển các ứng dụng robot. Điều này cho phép các nhà nghiên cứu và kỹ sư có thể xây dựng các hệ thống robot xe nâng tự hành linh hoạt và tùy chỉnh để phù hợp với các nhu cầu cụ thể trong kho xưởng. Đồng thời, sử dụng công nghệ LIDAR giúp robot xe nâng tự hành có thể phát hiện và định vị vật cản trong kho xưởng một cách chính xác và nhanh chóng, từ đó giúp robot tránh được các va chạm và giảm thiểu rủi ro trong quá trình vận hành.

Trong bài luận văn này, chúng em sẽ giới thiệu về Robot xe nâng tự hành trong kho xưởng sử dụng LIDAR và hệ điều hành ROS, và trình bày các ứng dụng và lợi ích của việc áp dụng công nghệ tự động hóa này trong quản lý và vận hành kho xưởng. Chúng em cũng sẽ trình bày chi tiết về cấu trúc, hoạt động và kết quả thực nghiệm của robot xe nâng tự hành trong kho xưởng sử dụng LIDAR và hệ điều hành ROS. Bài luận văn này mong muốn đóng góp ý nghĩa cho việc nghiên cứu và phát triển robot xe nâng tự hành trong kho vận và sản xuất trong tương lai, đồng thời cung cấp giải pháp hiệu quả để tối ưu hóa quá trình vận chuyển hàng hóa trong kho xưởng.

CHƯƠNG 1: TỔNG QUAN

1.1 Đặt vấn đề

Trong thời đại hiện nay, vận chuyển và lưu trữ hàng hóa là những vấn đề được quan tâm đặc biệt trong các kho hàng. Với sự phát triển của thương mại điện tử và nhu cầu vận chuyển hàng hóa ngày càng tăng, các kho hàng đang phải đổi mới với nhiều thách thức để đảm bảo việc quản lý hàng hóa một cách hiệu quả và đáp ứng nhu cầu của khách hàng. Các vấn đề như tăng cường an ninh, quản lý kho hàng thông minh, vận hành và xử lý đơn hàng nhanh chóng và chính xác đang được đặt lên hàng đầu. Vì vậy, việc sử dụng các công nghệ mới như IoT, robot và trí tuệ nhân tạo để giải quyết các vấn đề này đang trở thành xu hướng trong ngành lưu trữ và vận chuyển hàng hóa.



Hình 1. 1: Hệ thống kho hàng thông minh

Hướng ứng những điều trên việc sử dụng Robot và xe tự động để phục vụ cho việc vận chuyển và lưu trữ hàng hóa trong kho là một giải pháp đáp ứng nhu cầu của các kho hàng trong thời buổi hiện nay. Robot và xe tự động được sử dụng trong kho hàng được thiết kế để có thể di chuyển hàng hóa một cách linh hoạt và hiệu quả hơn so với các phương tiện vận chuyển truyền thống. Bằng cách sử dụng Robot và xe tự động, các kho hàng có thể đạt được tỷ lệ lỗi thấp hơn, tăng năng suất và giảm chi phí. Ngoài ra, việc sử dụng Robot và xe tự động còn giúp giảm thiểu nguy cơ tai nạn và chấn thương cho nhân viên trong quá trình vận chuyển hàng hóa.

Trên thế giới việc sử dụng Robot tự hành trong kho đã trở nên phổ biến trong các nhà kho lớn. Tiêu biểu như trong kho hành của Amazon công ty bán lẻ lớn nhất thế

giới, người ta dùng Robot tự động là Robot Kiva để quản lý, vận chuyển và vận hành kho hàng. Việc sử dụng lượng Robot trong kho lưu trữ hành hóa giúp tăng hiệu suất lưu thông, tiện ích trong việc quản lý, giảm sức lao động chân tay và giảm bớt tai nạn lao động.



Hình 1. 2: Robot vận chuyển và phân loại hàng hóa Kiva của Amazon

Ở Việt Nam hiện nay, Robot và xe tự hành đã phổ biến hơn, nhưng việc nghiên cứu và sử dụng Robot phục vụ cho công nghiệp và đời sống còn chưa được phổ biến vì nhân công còn rẻ và còn gặp nhiều hạn chế về kỹ thuật. Nhận thấy điều đó cùng với mong muốn học hỏi nhiều hơn về vấn đề tự động hóa nhóm chúng em đã quyết định sẽ lựa chọn nghiên cứu về Robot tự vận hành vận chuyển hàng hóa trong kho hàng.

1.2 Lịch sử phát triển của xe tự hành



Hình 1. 3: Những robot vận chuyển hàng hóa tự hành đời đầu

Xe tự hành hay Xe AGV (Automated Guided Vehicle) là một loại robot di động tự động hóa được sử dụng để vận chuyển và chuyển đổi hàng hoá trong các khu vực sản xuất, kho bãi, bệnh viện, sân bay, trung tâm thương mại, v.v. Xe AGV được lập trình để tự động di chuyển trên các đường ổn định và đến các vị trí được chỉ định mà không cần hướng dẫn từ con người.

Lịch sử phát triển của xe AGV bắt đầu vào những năm 1950 và 1960. Các công ty sản xuất xe nâng hàng như Yale & Towne và Raymond đã bắt đầu phát triển các hệ thống tự động hóa vận chuyển hàng hoá trong nhà máy của họ. Các hệ thống này được phát triển để giảm thiểu thời gian và chi phí lao động trong quá trình vận chuyển hàng hoá.

Đến những năm 1990 và 2000, với sự phát triển của các công nghệ thông tin và truyền thông, xe AGV đã được cải tiến và áp dụng rộng rãi hơn trong các lĩnh vực khác như bệnh viện, sân bay, trung tâm thương mại.

Một số tính năng của xe AGV bao gồm định vị và định hướng tự động, phát hiện và tránh vật cản, kết nối mạng và truyền thông, và tích hợp với các hệ thống quản lý kho, quản lý sản xuất và quản lý dịch vụ.

Từ những năm 2010, với sự phát triển của các công nghệ như AI, IoT và tự động hóa, xe AGV đang trở thành một phần quan trọng của sự phát triển của Công nghiệp 4.0. Các hãng sản xuất xe AGV đang nghiên cứu và phát triển các tính năng mới để tăng cường khả năng tự động hóa và tăng hiệu quả trong các ứng dụng khác nhau.



Hình 1. 4: Robot nghiên cứu và tìm kiếm sự sống trên sao Hỏa của NASA

1.3 Mục tiêu đề tài

Hệ thống xe tự hành trên thế giới đã có những thành tựu và cải tiến theo từng thời kỳ. Trong đồ án này nhóm chúng em muốn xây dựng một mô hình xe tự hành hoàn chỉnh có thể hoạt động thực tế trong kho xưởng.

Các mục tiêu cụ thể đề ra:

- Thiết lập được máy tính nhúng sử dụng ROS làm hệ điều hành.
- Tìm hiểu và ứng dụng các thuật toán hiệu quả vào trong Robot.
- Mô hình làm ra có tính thực dụng, chắc chắn có thể áp dụng trong thực tế.
- Giao diện điều khiển hiệu quả, đơn giản dễ sử dụng.
- Thiết bị điều khiển và giao thức truyền thông đa dạng.

1.4 Đối tượng và phạm vi nghiên cứu

Hệ điều hành:

- Đối tượng nghiên cứu: Ubuntu, ROS
- Phạm vi nghiên cứu:
 - Ubuntu: là một trong những hệ điều hành Linux phổ biến nhất trên thế giới. Tìm hiểu cách cài đặt, sử dụng và sửa lỗi cơ bản.
 - ROS: là một hệ thống phần mềm mã nguồn mở được sử dụng rộng rãi trong nghiên cứu và phát triển các ứng dụng robot. Tìm hiểu về quản lý điều khiển, cấu hình thiết bị, xử lý dữ liệu cảm biến và động cơ, và tích hợp các giải thuật điều khiển robot
 - Rviz: cho phép người dùng tương tác với robot và môi trường xung quanh robot thông qua các công cụ như điều khiển bằng chuột và bàn phím

Vì xử lý:

- Đối tượng nghiên cứu: Arduino Mega 2560
- Phạm vi nghiên cứu:
 - PID: Thuật toán PID sử dụng các thông số đầu vào như giá trị hiện tại, giá trị đặt trước và tốc độ thay đổi để tính toán ra một giá trị đầu ra, giúp điều khiển hệ thống đạt được giá trị đặt trước.
 - PWM: Kỹ thuật PWM có thể điều chỉnh độ rộng và tần số của tín hiệu xung để điều khiển mức độ hoạt động của một thiết bị.

- Stepper Motor: Thư viện Stepper Motor Arduino được cung cấp để giúp đơn giản hóa quá trình điều khiển động cơ bước trên Arduino.
- RosSerial: RosSerial là một công cụ cho phép Arduino và các thiết bị điện tử khác kết nối và giao tiếp với ROS (Robot Operating System).

Các thuật toán lập bản đồ, định vị và điều hướng:

- Đối tượng nghiên cứu: Lập bản đồ (gmapping), định vị và điều hướng (navigation stack).
- Phạm vi nghiên cứu:
 - Lập bản đồ: thu thập dữ liệu từ các cảm biến của robot và sau đó sử dụng các công cụ phần mềm như GMapping hoặc Hector SLAM để xây dựng bản đồ từ dữ liệu.
 - Định vị: xác định vị trí của robot trong môi trường.
 - Điều hướng: sử dụng các thuật toán điều khiển robot tự động để đưa ra các tín hiệu điều khiển để robot di chuyển.

Các phần mềm thiết kế và mô phỏng:

- Đối tượng nghiên cứu: autocad, Soliworks.
- Phạm vi nghiên cứu:
 - Thiết kế bản vẽ: thiết kế bản vẽ robot.
 - Mô phỏng 3D: Mô phỏng robot thành dạng 3D tăng tính thực tế.

CHƯƠNG 2: MÔ HÌNH ĐỘNG HỌC XE ROBOT

Với mỗi Robot mô hình động học là phần quan trọng quyết định đến kết quả sau này. Mô hình động học tốt và chính xác sẽ giúp các bước sau trở nên đơn giản và chính xác hơn.

Bước đầu tiên khi chúng ta xây dựng mô hình động học phải xác định các yêu cầu của cơ bản mà mô hình động học của Robot phải đáp ứng được:

- **Đáp ứng về tải trọng:** Vì là Robot chuyên hàng nên Robot phải đáp ứng được tải trọng lớn. Nhóm dự tính Robot sẽ có tải trọng khoảng 30kg và có thể nâng lượng hàng lên đến 35 kg.
- **Độ chắc chắn chống va đập:** Trong quá trình vận chuyển tải trọng lớn thi Robot phải có chất lượng tốt và chắc chắn để bảo vệ các linh kiện bên trong lúc bị va đập.
- **Độ an toàn:** Các góc cạnh sắc nhọn trên Robot được bọc lại và dán phản quang. Phần đi dây gọn gàng được bọc cách điện tốt tránh trường hợp chập điện gây cháy nổ.
- **Hợp lý về giá:** Để giảm thiểu giá thành và chủ động trong phần thiết kế, nhóm chúng em dùng vật liệu chính làm Robot là: Nhôm định hình và mica đen. Ưu điểm là sự chắc chắn, linh hoạt về kích thước về khối lượng riêng vừa phải.

2.1 Thiết kế bản vẽ

AutoCAD một phần mềm thiết kế đồ họa 2D được sử dụng rộng rãi trong nhiều lĩnh vực công nghiệp. Không chỉ giúp tăng độ chính xác và giảm thiểu sai sót, Autocad còn giúp tăng năng suất và tiết kiệm thời gian trong quá trình thiết kế và sản xuất.



Hình 2. 1: Logo Autocad

SolidWorks là một phần mềm thiết kế 3D được sử dụng rộng rãi trong ngành công nghiệp do hãng Dassault Systemes phát triển. SolidWorks cho phép người dùng tạo ra các mô hình 3D với độ chính xác cao và dễ dàng sử dụng.



Hình 2. 2: Logo SolidWorks

Kích thước khung xe robot:

Khung xe robot	Kích thước
Chiều dài	725mm
Chiều rộng	300mm
Chiều cao	632mm

Bảng 1: Thông số kích thước cơ bản của robot

2.2 Hệ thống di chuyển

Hệ thống di chuyển là một phần quan trọng trong thiết kế của robot, nó cho phép robot di chuyển từ vị trí này đến vị trí khác để hoàn thành các tác vụ được giao.

Thành phần	Hình ảnh	Kích thước (L x W x H)	Số lượng (SL) Tác dụng
Bánh xe phụ		L: 50mm W: 30mm H: 20 mm	SL: 2 cái Giữ thăng bằng và chịu lực cho robot
Bánh xe chính		L: 240mm W: 240mm H: 80 mm	SL: 2 cái Điều hướng và chịu lực chính cho robot
Hộp số động cơ điện WSJ 550		L: 230mm W: 92mm H: 43mm	SL: 2 cái Tỉ lệ chuyển đổi 1/24, tăng lực kéo của động cơ

Động cơ DC Servo JGB37-545 DC		L: 62mm W: 36mm H: 36mm	SL: 2 cái Động lực chính của xe, lực kéo tối đa 15kg, tốc độ tối đa 19500rpm
----------------------------------	---	-------------------------------	---

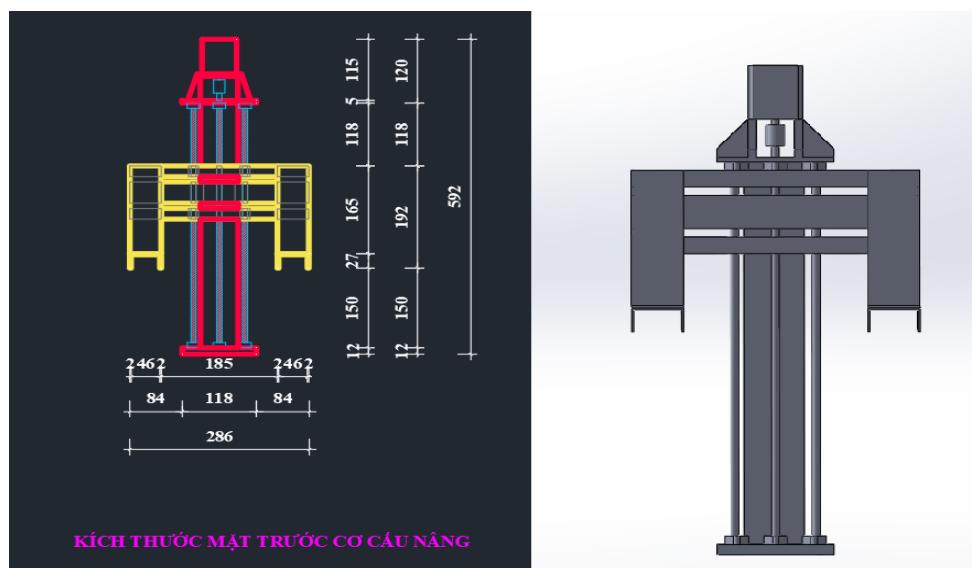
Bảng 2: Các thành phần trong hệ thống di chuyển của robot

Ghi chú:

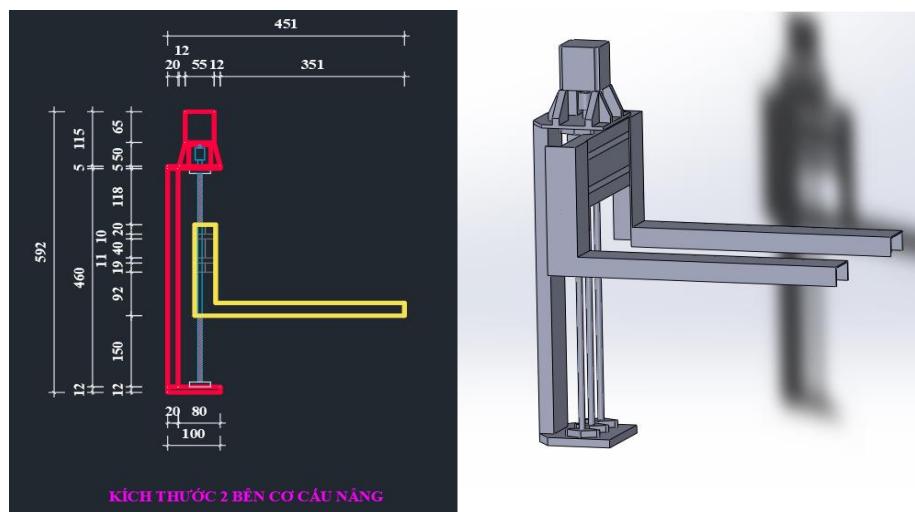
- L: Chiều dài
 - W: Chiều rộng
 - H: Chiều cao

2.3 Hệ thống cơ khí nâng hạ hành hóa

Hệ thống cơ khí nâng hạ hàng hóa là một phần quan trọng trong Robot, nó cho phép Robot đáp ứng các yêu cầu về nâng hạ hàng hóa, di chuyển và đặt hàng hóa một cách chính xác và an toàn.

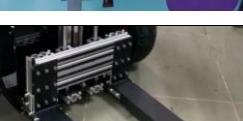


Hình 2. 3: Bản vẽ kích thước mặt trước cơ cấu nâng 2D và 3D



Hình 2. 4: Bản vẽ kích thước 2 mặt bên cơ cẩu nâng 2D và 3D

Thành phần	Hình ảnh	Kích thước (L x W x H)	Số lượng (SL) Tác dụng
Trục vít me (Trục vít xoắn)		L: 10mm W: 10mm H: 440mm	SL: 1 cái Chuyển động quay thành chuyển động tuyến tính
Trục tròn		L: 10mm W: 10mm H: 400mm	SL: 2 cái Chịu lực
Động cơ bước (Stepper Motor)		L:55mm W:55mm H65mm	SL: 1 cái Tạo lực xoay
Con trượt		L: 40mm W: 40mm H: 26mm	SL: 4 cái Liên kết với khung nâng. Giữ ổn định
Con trượt tròn		L: 30mm W: 40mm H: 40mm	SL: 1 cái Chuyển động trên trục vít me

Vòng bi gói đỡ		L: 80mm W: 18mm H: 46mm	SL: 2 cái Giữ cố định trực vít me
Kẹp trục ngang		L: 80mm W: 12mm H: 46mm	SL: 4 cái Giữ cố định trực tròn
Giá đỡ		L:118mm W:100mm H:530mm	SL: 1 Lắp đặt các thanh trực
Khung nâng		L:400mm W:283mm H:192mm	SL: 1 Dùng để nâng hàng hóa

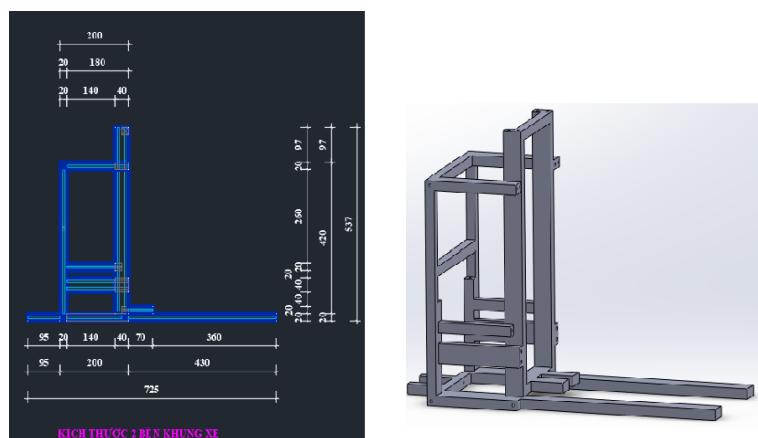
Bảng 3: Các thành phần trong hệ thống cơ khí nâng hạ

Ghi chú:

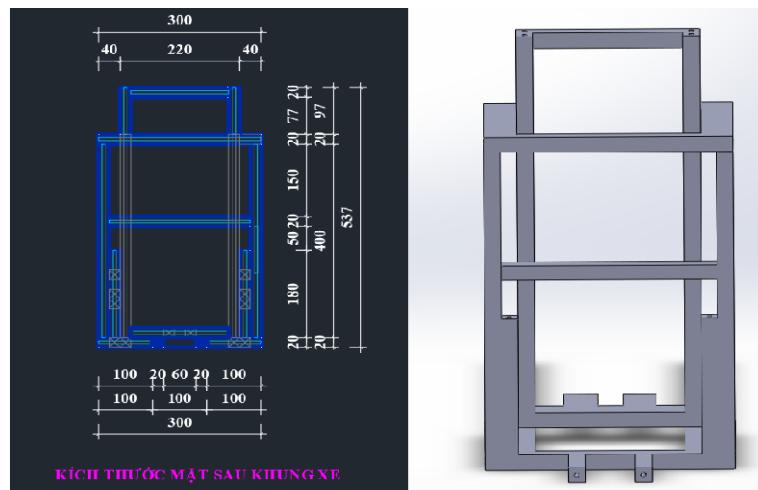
- L: Chiều dài
 - W: Chiều rộng
 - H: Chiều cao

2.4 Phân khung Robot

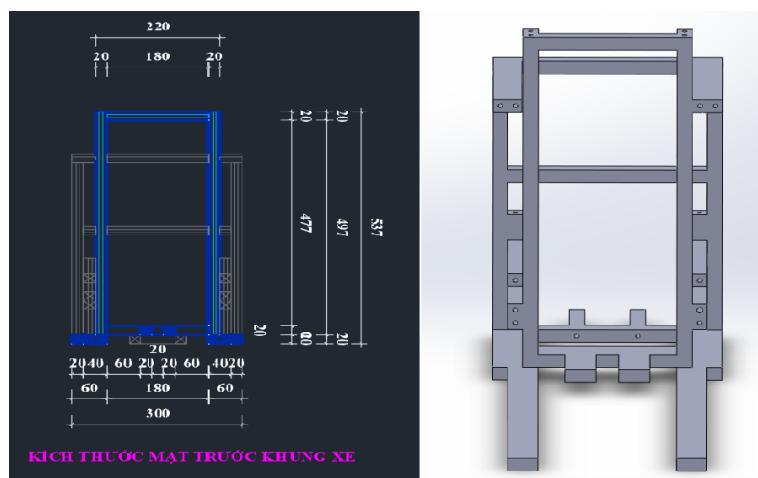
Phần khung Robot là một phần quan trọng trong thiết kế của Robot chuyên hàng. Với cấu trúc đa tầng vững chắc, khung robot của nhóm thiết kế có thể giúp cho Robot có thể chịu được tải trọng cung cấp nơi cố định các bộ phận khác của Robot và đảm bảo tính ổn định của Robot trong quá trình vận hành.



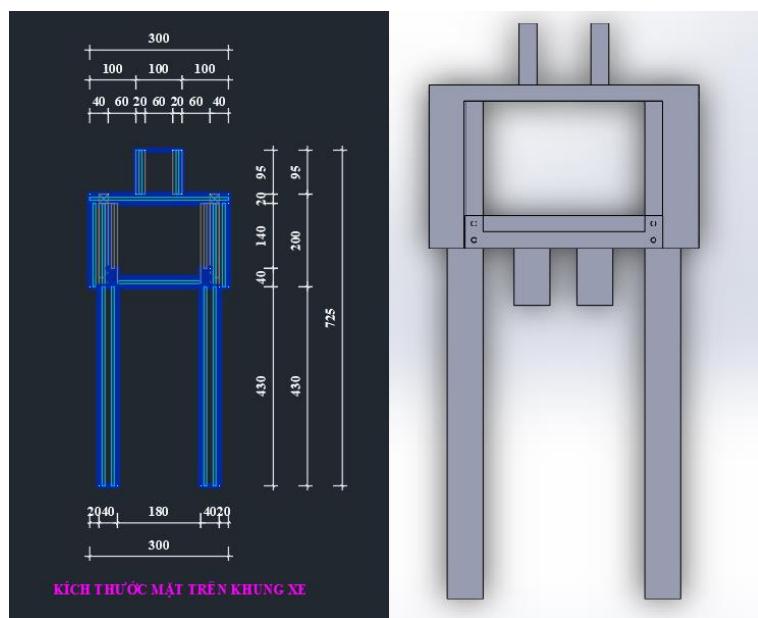
Hình 2. 5: Bản vẽ 2D và 3D kích thước 2 mặt bên khung xe



Hình 2. 6: Bản vẽ 2D và 3D kích thước mặt sau khung xe



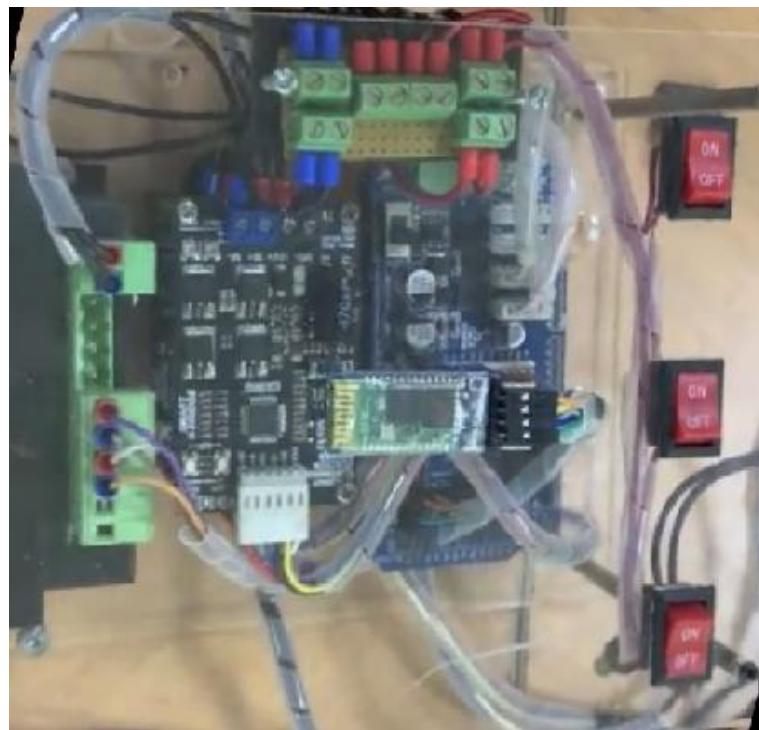
Hình 2. 7: Bản vẽ 2D và 3D kích thước mặt trước khung xe



Hình 2. 8: Bản vẽ 2D và 3D kích thước mặt trên khung xe

Các phần trên khung Robot:

- **Hộp kỹ thuật:** Là nơi lắp đặt các thiết bị mạch điện tử để tiện lợi cho việc lắp đặt và bảo trì , bên cạnh đó còn tránh được va đập và các sự cố rủi ro gây ra hư hỏng các thiết bị khi robot vận hành. Giúp cho việc nối dây giữa các thiết bị, động cơ và mạch điều khiển được kết nối gọn gàng, an toàn và tiết kiệm thời gian bảo trì sửa chữa.



Hình 2. 9: Hộp kỹ thuật

- **Phần khung đỡ phần cơ khí:** Đây là bộ phận quan trọng của khung xe , cần phải lắp đặt chính xác và kiên cố, có tính chắc chắn vì hệ thống duy chuyển sẽ ghép nối với khung xe và chịu rất nhiều tải trọng.

2.5 Bộ nguồn Pin cung cấp

Bộ nguồn Pin là nơi lưu trữ và cấp phát năng lượng cho Robot hoạt động. Bộ nguồn Pin cũng là nơi dễ hư hại trong quá trình Robot di chuyển và hoạt động. Nên bộ nguồn Pin cần có các yêu cầu cụ thể để đảm bảo độ an toàn và sự linh hoạt của Robot:

- **Dung lượng pin:** Pin được đóng thủ công từ 16 viên Litium, tổng dung lượng 6600mAh.
- **Cầu chì bảo vệ:** Cầu chì là một loại vật liệu dẫn điện quan trọng dùng để bảo vệ các linh kiện điện tử khỏi điện áp quá cao khi quá tải hoặc sự cố ngoài ý muốn. Sử dụng cầu chì Ô tô 10A.

- Mạch bảo vệ và mạch sạc: Sử dụng mạch sạc bảo vệ cell pin Li-ion 18650.
- Khối lượng và kích thước:

Các thông số hộp pin	Đơn vị
Khối lượng	1,5 kg
Chiều cao	6cm
Chiều dài	18cm
Chiều rộng	15cm

Bảng 4: Thông số kỹ thuật bộ nguồn

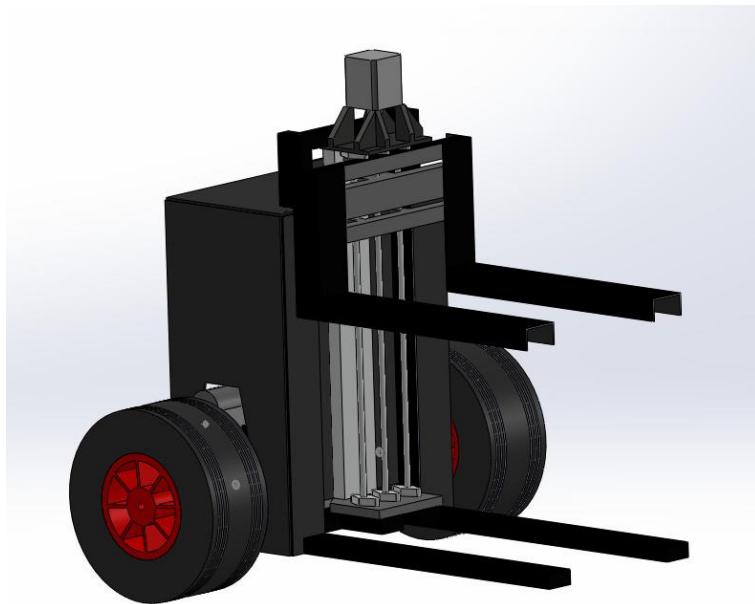
Thiết kế vỏ bọc cho nguồn pin: Phần hộp pin trong robot được thiết kế với mục đích cách ly nguồn điện với các thiết bị khác trong robot. Điều này giúp tránh các sự cố rò rỉ hoặc chập điện trong quá trình hoạt động, đảm bảo an toàn cho robot và người sử dụng. Ngoài ra, phần hộp pin còn có tác dụng bảo vệ pin khỏi các tác động bên ngoài như va đập, rung động hay nhiệt độ cao. Như vậy, việc thiết kế hộp pin cũng giúp kéo dài tuổi thọ của pin trong robot, giúp robot hoạt động ổn định và hiệu quả hơn.



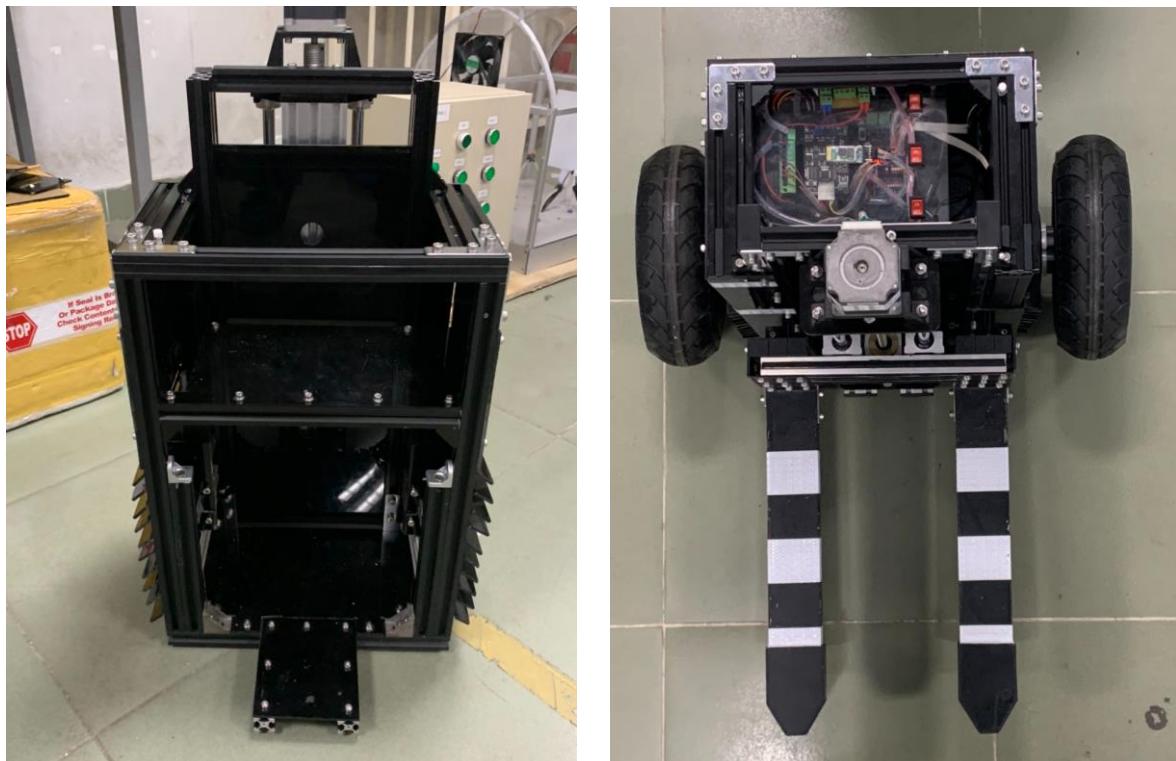
Hình 2. 10: Bộ nguồn pin 24V

2.6 Hoàn thiện phần cứng

Sau khi hoàn thiện các phần trên Robot đã đủ các bộ phận để hoạt động. Nhưng để tăng thêm phần thẩm mỹ thì phải tạo thêm phần vỏ bọc bên ngoài thân Robot.



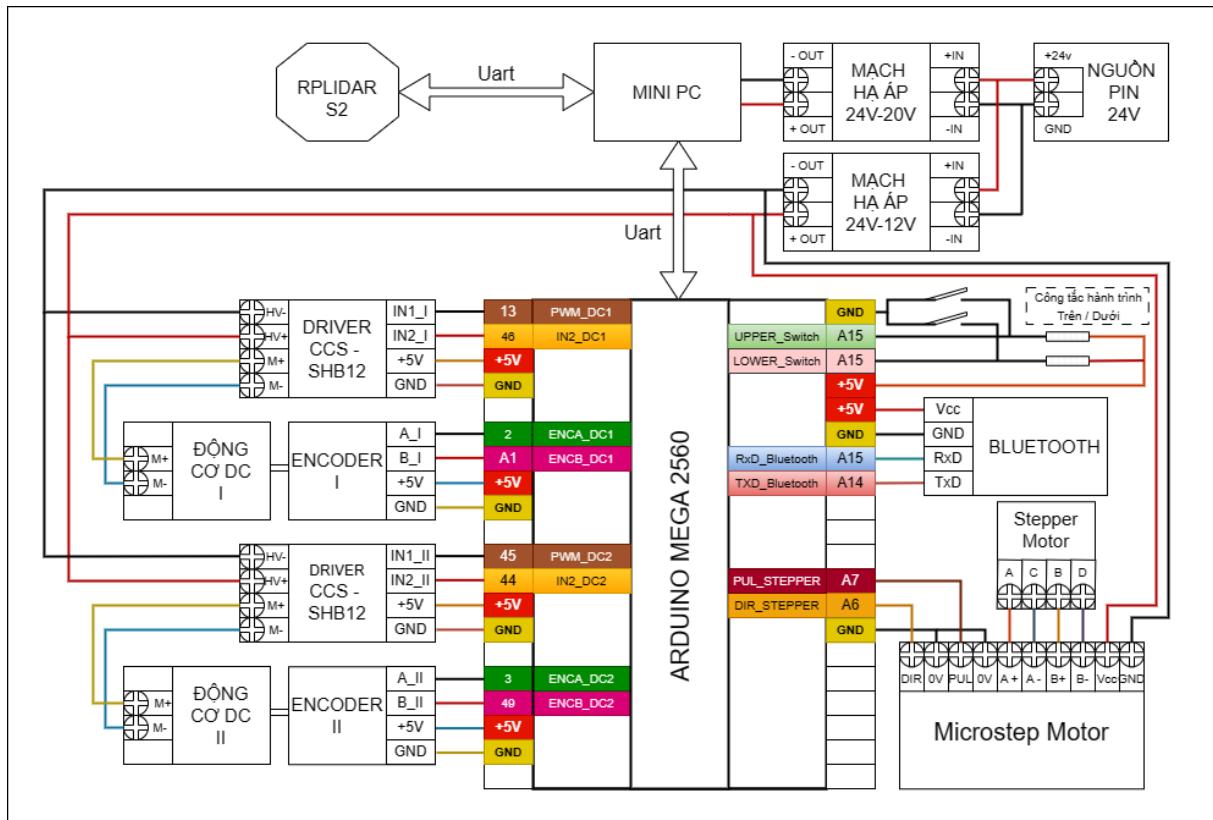
Hình 2. 11: Bản vẽ 3D tổng thể robot



Hình 2. 12: Khung xe trước và sau khi lắp đặt

CHƯƠNG 3: XÂY DỰNG THUẬT TOÁN ĐIỀU KHIỂN

3.1 Một số khái niệm quan trọng



Hình 3. 1: Sơ đồ khái quát Input/ Output

Để xây dựng thuật toán điều khiển của robot đầu tiên phải xác định được những tín hiệu INPUT, OUTPUT:

INPUT	XỬ LÝ	OUTPUT
<ul style="list-style-type: none"> Encoder Lidar Camera Công tắc hành trình 	<ul style="list-style-type: none"> Hệ điều hành (ROS) Vi xử lý (Arduino) Thuật toán Mạch Driver Microstep Motor 	<ul style="list-style-type: none"> Động cơ Stepper Motor

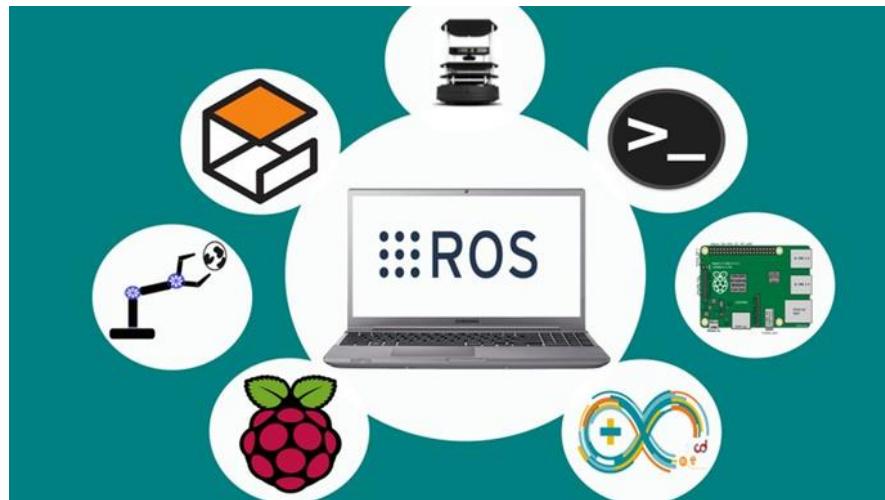
Bảng 5: Hệ thống xử lý và các ngõ vào và ra của robot

3.1.1 Hệ điều hành – ROS

Hệ điều hành (Operating System - OS) là một phần mềm quản lý và điều khiển tài nguyên phần cứng và phần mềm trong một máy tính hoặc một hệ thống máy tính. Hệ điều hành cung cấp một giao diện giữa người dùng và phần cứng, cho phép người dùng tương tác với máy tính và sử dụng các ứng dụng phần mềm một cách dễ dàng.

ROS là một hệ điều hành quan trọng trong lĩnh vực robot và tự động hóa. Nó cung cấp một nền tảng phát triển chuẩn, hỗ trợ việc phát triển các ứng dụng phức tạp, cung

cấp các tính năng phân tán, hỗ trợ tính tái sử dụng, hỗ trợ việc kiểm tra và mô phỏng và hỗ trợ cộng đồng phát triển rộng lớn. Sử dụng ROS cho phép các nhà phát triển xây dựng các ứng dụng robot và tự động hóa một cách nhanh chóng, dễ dàng và hiệu quả, đóng góp rất lớn cho sự tiến bộ của lĩnh vực này.



Hình 3. 2: Hệ sinh thái đa ngôn ngữ của Ros

3.1.2 Vi xử lý



Hình 3. 3: Modul arduino Mega 2560

Vi xử lý (microprocessor) là một thành phần quan trọng trong các thiết bị điện tử hiện đại. Vi xử lý có tác dụng xử lý và điều khiển các dữ liệu điện tử và các thiết bị ngoại vi khác. Một số tác dụng của vi xử lý:

Xử lý dữ liệu: Vi xử lý có khả năng xử lý các dữ liệu điện tử từ các thiết bị ngoại vi như cảm biến, bộ điều khiển, và các thiết bị khác. Vi xử lý có thể thực hiện các phép tính số học, các phép toán logic, và các phép toán khác để xử lý dữ liệu.

Điều khiển thiết bị: Vi xử lý có khả năng điều khiển các thiết bị ngoại vi khác như động cơ, đèn LED, màn hình LCD, và các thiết bị khác. Vi xử lý có thể điều khiển các thiết bị này bằng cách gửi các tín hiệu điều khiển từ chính nó hoặc từ các thiết bị khác.

3.1.3 Mạch Driver

Mạch Driver là một loại mạch điện được sử dụng để điều khiển các thiết bị điện như động cơ, bơm, đèn LED và các thiết bị khác. Mạch driver có nhiều tác dụng quan trọng, bao gồm:

Điều khiển tải: Mạch driver giúp điều khiển tải, tức là điều khiển động cơ, bơm, đèn LED và các thiết bị khác hoạt động theo ý muốn. Điều này giúp tăng tính linh hoạt và độ chính xác của các hệ thống điện.

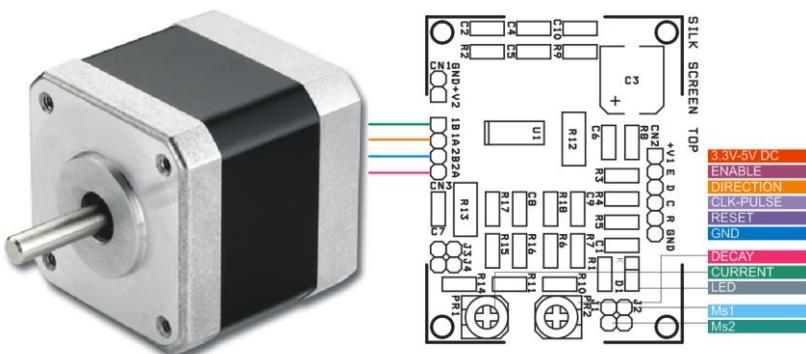
Tăng hiệu suất: Mạch driver giúp tăng hiệu suất của các thiết bị điện bằng cách giảm tổn thất điện năng trong quá trình điều khiển. Điều này giúp tăng hiệu quả và tiết kiệm năng lượng cho hệ thống điện.

Bảo vệ thiết bị: Mạch driver có thể được thiết kế để bảo vệ các thiết bị điện khỏi các tác động tiêu cực như quá dòng, quá áp, nhiễu điện từ và các vấn đề khác. Điều này giúp tăng tuổi thọ và độ tin cậy của các thiết bị điện.

Tăng độ bền: Mạch driver có thể giúp tăng độ bền của các thiết bị điện bằng cách giảm điện áp và dòng điện chạy qua chúng. Điều này giúp giảm hao mòn và tăng tuổi thọ của các thiết bị.

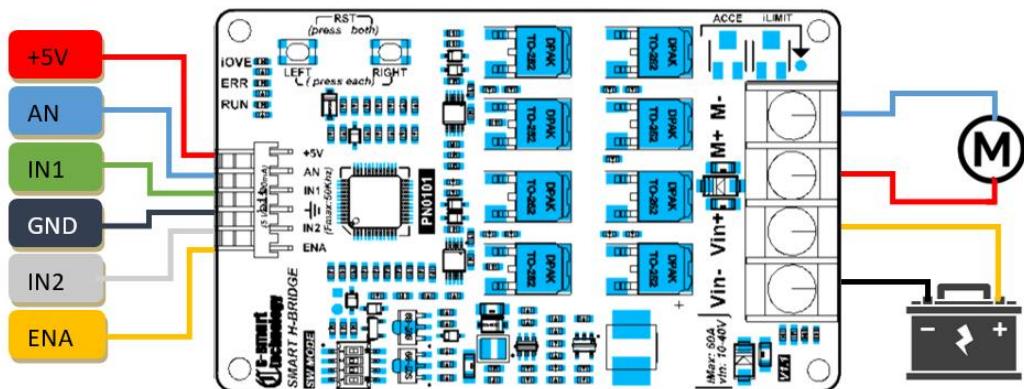
Hai loại mạch Driver được sử dụng trong Robot là:

- CCS – SHB12: Mạch điều khiển động cơ DC điều khiển chuyển động của Robot.



Hình 3.4: Sơ đồ nguyên lý mạch driver điều khiển step motor

- Microstep Driver TB6600: Mạch điều khiển động cơ bước phản ứng khí nén hàng.



Hình 3.5: Sơ đồ nguyên lý mạch driver điều khiển động cơ DC

3.1.4 Lidar (RPLIDAR S2)

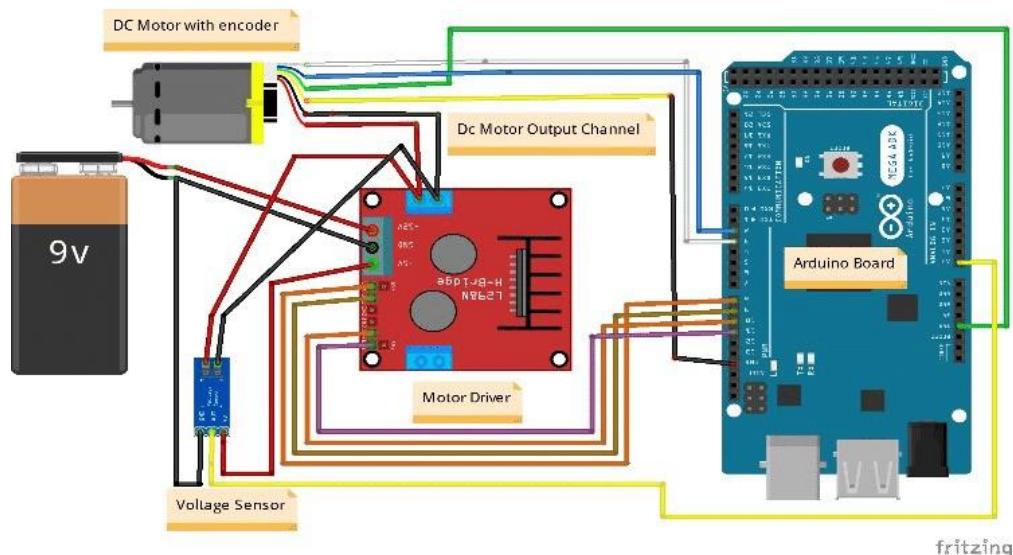
Lidar (Light Detection and Ranging) là một công nghệ đo khoảng cách bằng cách sử dụng ánh sáng laser. Lidar được sử dụng rộng rãi trong nhiều lĩnh vực và có nhiều tác dụng quan trọng, bao gồm:

Cung cấp thông tin về môi trường: Lidar có khả năng quét một khu vực rộng và thu thập thông tin về môi trường xung quanh.

Độ chính xác: Lidar cung cấp độ chính xác cao và có thể phát hiện các vật thể trong khoảng cách xa. Điều này làm tăng độ chính xác và độ tin cậy của các ứng dụng sử dụng Lidar.

Giảm thiểu tai nạn: Lidar có khả năng phát hiện các vật thể trong khoảng cách xa và cung cấp thông tin về chúng. Điều này giúp giảm thiểu nguy cơ tai nạn cho xe tự hành khi hoạt động trong môi trường có nhiều vật thể di động.

3.1.5 Động cơ và encoder



Hình 3.6: Sơ đồ kết nối động cơ DC và encoder với arduino

Động cơ và encoder là hai thành phần quan trọng trong các hệ thống điều khiển chuyển động của robot. Cả hai thành phần này có nhiều tác dụng quan trọng, bao gồm:

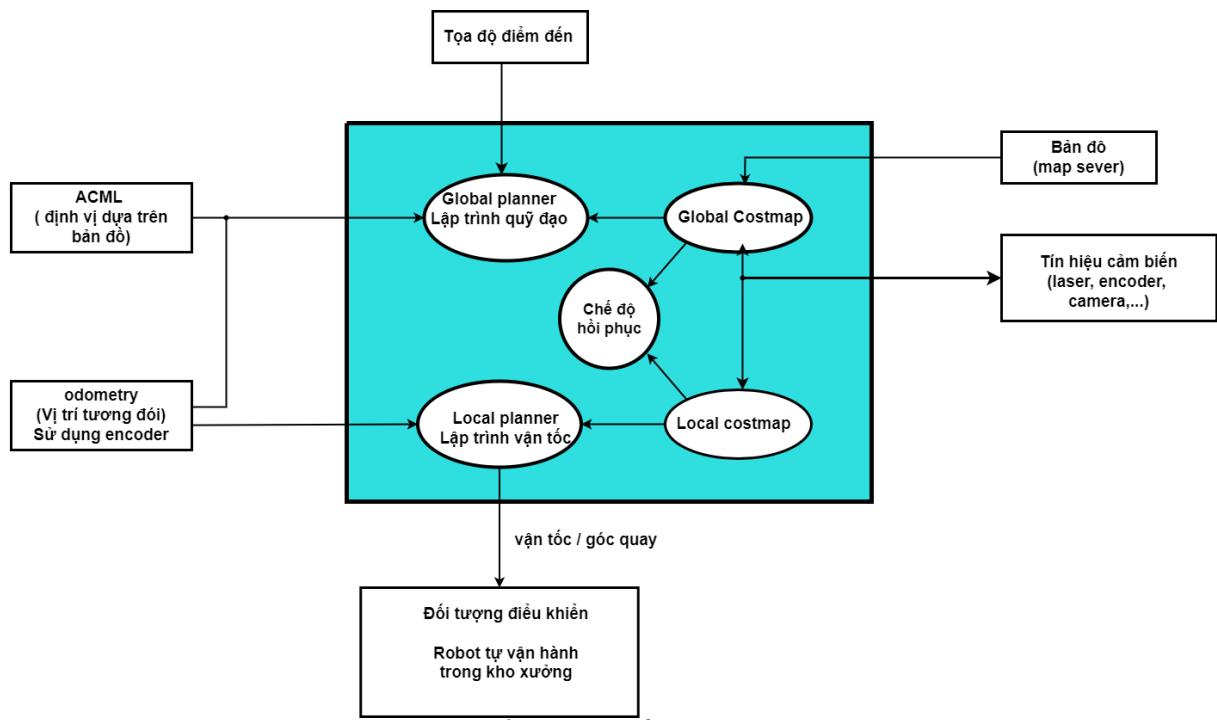
Điều khiển chuyển động: Động cơ là thành phần chính trong hệ thống điều khiển chuyển động. Nó chuyển đổi năng lượng điện thành năng lượng cơ học để tạo ra chuyển động. Encoder cung cấp thông tin về vị trí và tốc độ của động cơ, giúp điều khiển chuyển động một cách chính xác.

Giám sát tốc độ và vị trí: Encoder có khả năng giám sát tốc độ và vị trí của động cơ hoặc robot. Thông tin này có thể được sử dụng để giám sát tình trạng hoạt động của hệ thống và đưa ra các quyết định điều khiển phù hợp.

Tăng độ chính xác: Encoder cung cấp thông tin về vị trí và tốc độ chính xác hơn so với các phương pháp đo khác. Điều này giúp tăng độ chính xác và độ tin cậy của các hệ thống điều khiển động cơ và robot.

Điều khiển mô-men xoắn: Encoder có thể được sử dụng để điều khiển mô-men xoắn của động cơ. Thông tin về vị trí và tốc độ của động cơ được sử dụng để điều khiển mô-men xoắn phù hợp, giúp tăng hiệu suất và độ chính xác của các ứng dụng.

3.2 Định vị và điều hướng (Navigation stack [6])



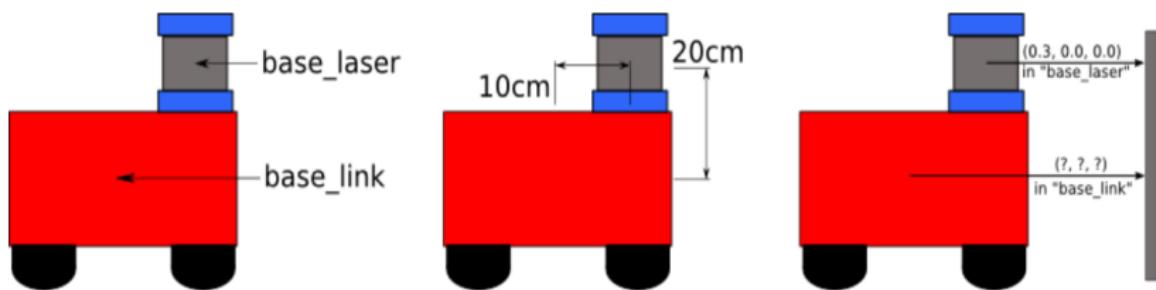
Hình 3.7: Tổng quan về navigation stack

Navigation stack có thể được xem như là tên gọi của toàn bộ quá trình mà một robot tự điều hướng di chuyển. Sơ đồ trên cho thấy tổng quan về cấu hình này thông qua các node có thể khởi tạo thông qua viết mã lệnh thông qua ROS. Các điều kiện tiên

quyết của ngăn xếp điều hướng, cùng với hướng dẫn về cách thực hiện từng yêu cầu, được cung cấp trong các phần bên dưới. Mỗi thành phần là mỗi node thực hiện mỗi chức năng và gửi các dữ liệu thông qua các topics.

3.2.1 Chuyển đổi cấu hình

Để làm rõ điều này, chúng ta sẽ xem xét một ví dụ về một robot đơn giản với một phần đế và một laser đặt trên đứng trước một bức tường. Chúng ta sẽ đặt cho mỗi bộ phận của robot một hệ tọa độ Oxyz có tên là “**base_link**” và “**base_laser**” và gốc tọa độ nằm ở điểm chính giữa của mỗi bộ phận.



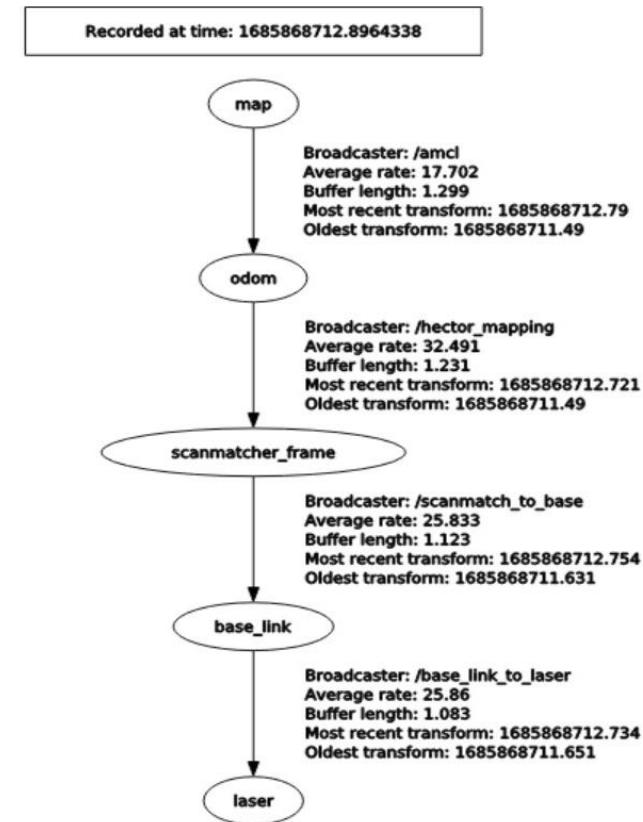
Hình 3. 8: ví dụ minh họa cho robot

Khi đó, giả sử ta có được dữ liệu khoảng cách thu được từ laser và được thể hiện thông qua tọa độ “**base_laser**” là (0.3, 0.0, 0.0) thể hiện cho khoảng cách từ laser đến bức tường là 0.3 met theo phương x. Giả sử ta muốn robot hoạt động trong môi trường thông qua phần thân robot, đầu tiên ta cần phải đặt hệ tọa độ gốc của là “**base_link**”. Sau đó chúng ta cần phải xác định mối quan hệ giữa hai hệ tọa độ “**base_link**” và “**base_laser**”. Ta có dữ liệu rằng tia laser gắn trên đế robot cao hơn phần tâm điểm của nó 0.2m theo phương z và lệch về phía trước theo phương x 0.1m. Ta có $d = (x: -0.1m, y: 0.0m, z: -0.2m)$ được xem như là độ lệch tịnh tiến của hai khung tọa độ này. Để biến đổi một điểm trong hệ tọa độ “**base_laser**” sang hệ tọa độ “**base_link**” chúng ta phải áp dụng một biến đổi tịnh tiến trên điểm đó $p_{base_link} = p_{base_laser} + d$ và có được kết quả (0.2, 0.0, -0.2).

Trong các ứng dụng robot lớn hơn, robot và hệ thống sẽ có nhiều khung tọa độ hơn, các biến đổi giữa các hệ tọa độ sẽ phức tạp hơn và thường là các ma trận biến đổi Affine bao gồm cả phép tịnh tiến và phép xoay ($x'y'z' = A_{matrix} * xyz$).

$$A_{matrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} & dx \\ a_{21} & a_{22} & a_{23} & dy \\ a_{31} & a_{32} & a_{33} & dz \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

ROS cung cấp cho ta công cụ tf là một công cụ để viết và quản lí các cấu hình chuyển đổi hệ tọa độ này. Trong tf để định nghĩa và cấu hình các chuyển đổi ta cần phải thêm chúng vào một “cây chuyển đổi”. Mỗi nút trong cây là một khung tọa độ và mỗi đường nối tương ứng phép biến đổi cần được áp dụng để chuyển đổi từ nút này sang nút khác. Ví dụ dưới đây là một cấu trúc cây của hệ thống xe tự hành.



Hình 3. 9: Cấu trúc cây biến đổi khung tọa độ của robot tự hành và sự biến đổi giữa các khung tọa độ.

Việc viết các chuyển đổi giữa các khung tọa độ được thực hiện thông qua công cụ tf (viết bằng source file). Ngoài ra, file URDF cũng tạo ra một khung tọa độ và duy trì các mối quan hệ giữa các bộ phận với nhau mà không cần phải định nghĩa lại thông qua tf.

Công cụ tf sẽ gửi các thông tin chuyển đổi này thông qua topics “/tf” và thông thường có thể hiển thị hoặc theo dõi thông qua Rviz.

3.2.2 Xử lý thông tin cảm biến

Mọi robot tự di chuyển cần phải biết mọi thứ về môi trường xung quanh nó. Để có thể sử dụng các gói tạo lập bản đồ, thông tin cảm biến là thứ đầu tiên phải được xác nhận.

Trong hệ thống mô phỏng của chúng ta chỉ sử dụng cảm biến Laser về khoảng cách giữa các vật cản và chính nó. Ta có thể mô phỏng cảm biến này trong Gazebo và laser sử dụng là Rplidar model S2.

Các dữ liệu được đưa truyền qua các node thông qua messages đã được định dạng trước là **sensor_msgs/LaserScan** hoặc **sensor_msgs/PointCloud**.

Vì ta chỉ sử dụng laser nên ta chỉ sử dụng **message sensor_msgs/LaserScan**. Những dữ liệu cảm biến này sẽ được thu thập và truyền tới các node khác để thực hiện

```
# Laser scans angles are measured counter clockwise, with 0 facing forward
Header header
float32 angle_min# start angle of the scan [rad]
float32 angle_max # end angle of the scan [rad]
float32 angle_increment # angular distance between measurements [rad]
float32 time_increment # time between measurements [seconds]
float32 scan_time # time between scans [seconds]
float32 range_min # minimum range value [m]
float32 range_max # maximum range value [m]
float32[] ranges # range data [m] (Note: values < range_min or > range_max
should be discarded)
float32[] intensities # intensity data [device-specific units]
```

Bảng 6: Bảng thông số cấu trúc một messenger cho cảm biến tính toán thông qua các sensor topics (thông thường tên topic là “scan”). Cấu trúc của một **message sensor_msgs/LaserScan**.

Header: đây là tiêu đề của tin nhắn, chứa thông tin như thời gian và nguồn gốc của dữ liệu.

angle_min: đây là góc bắt đầu của quét laser, được đo trong radian và đo theo chiều kim đồng hồ tính từ trục x của khung thiết bị.

angle_max: đây là góc kết thúc của quét laser, được đo trong radian và đo theo chiều kim đồng hồ tính từ trục x của khung thiết bị.

angle_increment: đây là khoảng cách góc giữa các lần đo của cảm biến laser, được đo trong radian.

time_increment: đây là khoảng thời gian giữa các lần đọc của cảm biến laser, được tính bằng giây.

scan_time: đây là khoảng thời gian giữa các lần quét của cảm biến laser, được tính bằng giây.

range_min: đây là giá trị khoảng cách tối thiểu mà cảm biến laser có thể đo được, được đo bằng mét.

range_max: đây là giá trị khoảng cách tối đa mà cảm biến laser có thể đo được, được đo bằng mét.

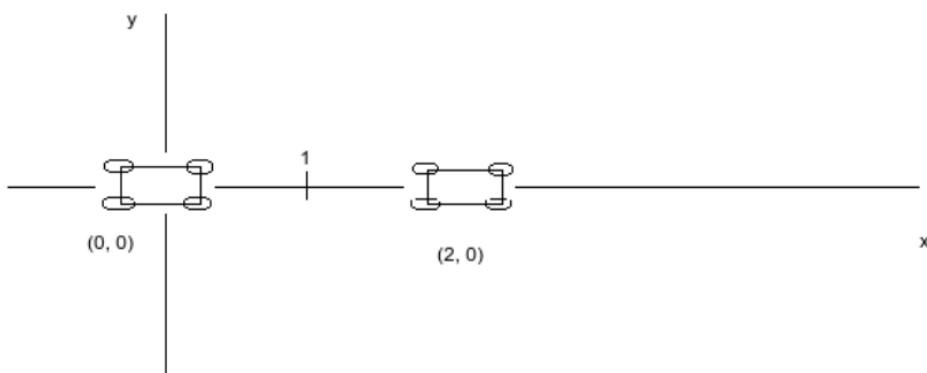
ranges: đây là dữ liệu khoảng cách đo được từ cảm biến laser, được đo bằng mét. Các giá trị nằm ngoài phạm vi từ range_min đến range_max nên được loại bỏ.

intensities: đây là dữ liệu độ mạnh của tín hiệu laser, được đo bằng đơn vị cụ thể của thiết bị.

3.2.3 Thông tin về vị trí và hướng của robot (Odometry information)

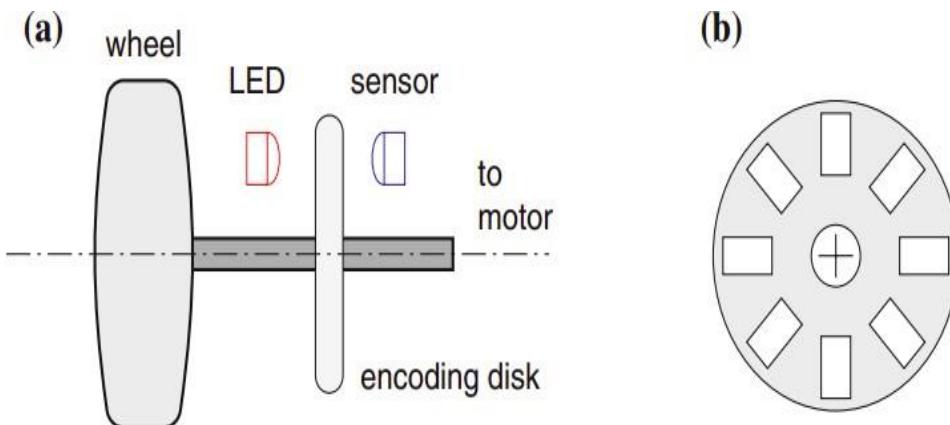
Trong robot học, thông tin về trạng thái của robot là thông tin về vị trí ước tính của robot (x_t , y_t , t) tại một thời điểm cụ thể sau khi di chuyển, liên quan đến vị trí bắt đầu của nó và được tính toán dựa trên các thông số như vận tốc và góc quay của robot. Thông tin Odometry được truyền qua các topic ROS có tên là "odom". Các node (các thành phần cơ bản của ROS) có thể đăng ký (subscribe) vào topic này để nhận thông tin Odometry hiện tại của robot. Thông tin Odometry có thể được cung cấp từ các cảm biến như bộ cảm biến IMU (Inertial Measurement Unit) hoặc các bộ mã hóa quay (encoder).

Ví dụ với robot xe bốn bánh tự hành, thông thường chúng ta sẽ đặt một khung tọa độ 2D Oxy tên là “odometry” tại nơi robot sẽ bắt đầu thực hiện di chuyển. Sau đó dựa vào thông tin vận tốc của robot để ước tính được vị trí của robot trong khung tọa độ “odometry” này. Xem hình 3.12, robot bắt đầu tại một vị trí cụ thể trong môi trường. Tại vị trí bắt đầu này, đặt hệ tọa độ “odometry” có gốc tại trung tâm của robot. Giả sử robot di chuyển theo phương thẳng x với vận tốc 0,2 m/s và di chuyển trong vòng 10s, khi đó robot sẽ kết thúc quãng di chuyển tại vị trí x bằng 2 (kết quả bằng vận tốc nhân thời gian di chuyển).



Hình 3. 10: Ví dụ về xác định vị trí robot trong tọa độ "odometry"

Thông thường trong robot thực tế, việc tính toán thời gian di chuyển của robot sẽ được tính bằng bộ đếm thời gian của bộ xử lý như máy tính nhúng hoặc bo mạch nhúng, việc tính toán quãng đường đã di chuyển của robot có thể dựa vào bộ mã hóa (encoder) được đặt ở bánh xe của chúng. Để triển khai bộ mã hóa này, một thiết kế phổ biến là một đèn LED đi-ôt phát sáng và cảm biến ánh sáng khi bánh xe di chuyển, và một đĩa mã hóa được gắn song song. Khi bánh xe di chuyển mỗi lần ánh sáng đi qua lỗ cảm biến sẽ tiếp nhận thông tin ánh sáng và gửi tín hiệu số về bộ xử lí. Chu vi của bánh xe là $2\pi r$, với r là bán kính bánh xe, với mỗi lần xoay hoàn toàn 1 chu kỳ hình tròn, ta có quãng đường của bánh xe là $n2\pi r$. Số lỗ trên đĩa mã hóa sẽ tương ứng với độ phân giải của bộ mã hóa. Ví dụ như hình 3.6.b với 8 lỗ trên đĩa mã hóa, quãng đường đi được của bánh xe robot là $n2\pi r/8$. Ngoài ra tốc độ còn có thể tính bằng các đặc tính của động cơ.



Hình 3.11: Cấu trúc bộ mã hóa quãng đường đi được và đĩa (encoder)

Đối với phần mềm mô phỏng ta có thể dùng cấu trúc động học của robot (**chương 2**) để tính thông tin về trạng thái của robot.

Việc xác định thông tin odometry thường bị nhầm lẫn với việc định vị robot. Trong các vấn đề định vị robot, thường một bản đồ của môi trường “map” bao gồm tất cả các vật cản và khu vực chiếm đóng sẽ được tính toán, yêu cầu và thông qua các thuật toán ta sẽ định vị robot trên bản đồ mà nó đã tính toán trước đó, vấn đề này thường được gọi là SLAM. Thông tin odometry thông thường được sử dụng trong việc định vị vị trí robot này, nó thường được kết hợp với thông tin laser để tính toán.

Ta có thể dùng Services để lấy các thông tin trạng thái robot từ phần mềm mô phỏng, và xuất bản thông tin này bằng dữ liệu dưới định dạng messages nav_msgs/Odometry.msg qua các topic /tf /odom. Cấu trúc của message:

```
[nav_msgs/Odometry]:
std_msgs/Header header
uint32 seq
time stamp
string frame_id
string child_frame_id
geometry_msgs/PoseWithCovariance pose
geometry_msgs/Pose pose
geometry_msgs/Point position
float64 x
float64 y
float64 z
geometry_msgs/Quaternion orientation
float64 x
float64 y
float64 z
float64[36] covariance
geometry_msgs/TwistWithCovariance twist
geometry_msgs/Twist twist
geometry_msgs/Vector3 linear float64 x float64 y float64 z
geometry_msgs/Vector3 angular float64 x float64 y float64 z
float64[36] covariance
```

Bảng 7: Bảng thông số cấu trúc odometry của robot

Header: đây là tiêu đề của tin nhắn, chứa thông tin như thời gian và nguồn gốc của dữ liệu.

seq: đây là số thứ tự của tin nhắn, giúp cho các node trong hệ thống ROS có thể xác định được thứ tự của các tin nhắn.

stamp: đây là thời gian ghi lại khi tin nhắn được gửi đi.

frame_id: đây là ID của khung dữ liệu của robot.

child_frame_id: đây là ID của khung dữ liệu của phần tử con của robot.

pose: đây là thông tin về vị trí và hướng của robot, bao gồm:

position: tọa độ vị trí của robot trong không gian, được đo bằng mét và được biểu diễn bởi Vector3 (bao gồm các trường x, y, z).

orientation: hướng của robot trong không gian, được biểu diễn bằng Quaternion (bao gồm các trường x, y, z).

covariance: ma trận hiệp phương sai của vị trí và hướng của robot, được biểu diễn bằng một mảng 36 phần tử.

twist: đây là thông tin về vận tốc và tốc độ góc của robot, bao gồm:

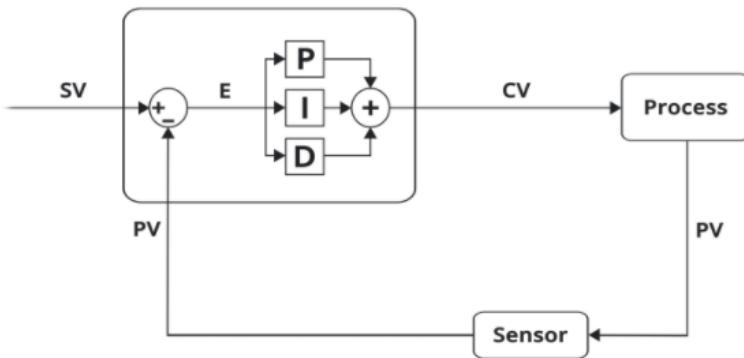
linear: tốc độ tuyến tính của robot, được đo bằng mét/giây và được biểu diễn bởi Vector3 (bao gồm các trường x, y, z).

angular: tốc độ góc của robot, được đo bằng rad/giây và được biểu diễn bởi Vector3 (bao gồm các trục x, y, z).

covariance: ma trận hiệp phương sai của vận tốc và tốc độ góc của robot, được biểu diễn bằng một mảng 36 phần tử.

3.3 Các thuật toán được sử dụng

3.3.1 Bộ điều khiển PID



Hình 3. 12: Hệ thống PID

Khái niệm:

Bộ điều khiển PID (Proportional-Integral-Derivative) là một bộ điều khiển phản hồi được sử dụng để điều khiển chuyển động của Robot. Bộ điều khiển PID sử dụng một thuật toán phản hồi để tính toán và điều chỉnh đầu ra điều khiển của robot dựa trên sự khác biệt giữa giá trị đầu vào (thiết lập mục tiêu) và giá trị đầu ra hiện tại của robot.

Bộ điều khiển PID sử dụng ba tham số để điều chỉnh chuyển động của robot: tỉ lệ (proportional), tích phân (integral) và vi phân (derivative). Thông qua các tham số này, bộ điều khiển PID sẽ điều chỉnh tốc độ và hướng di chuyển của robot để đạt được mục tiêu điều khiển.

Tác dụng:

Bộ điều khiển PID là một công cụ quan trọng trong việc điều khiển chuyển động của robot xe, giúp đảm bảo chính xác và đáng tin cậy trong việc điều khiển robot di chuyển theo mục tiêu đã định sẵn.

Lý thuyết tính toán:

Khâu tỉ lệ, tích phân, vi phân được cộng lại với nhau để tính toán đầu ra của bộ điều khiển PID. Định nghĩa rằng $u(t)$ là đầu ra của bộ điều khiển, biểu thức cuối cùng của giải thuật PID là:

$$u(t) = MV(t) = K_p e(t) + K_i \int_0^t e(\tau) d\tau + K_d \frac{d}{dt} e(t)$$

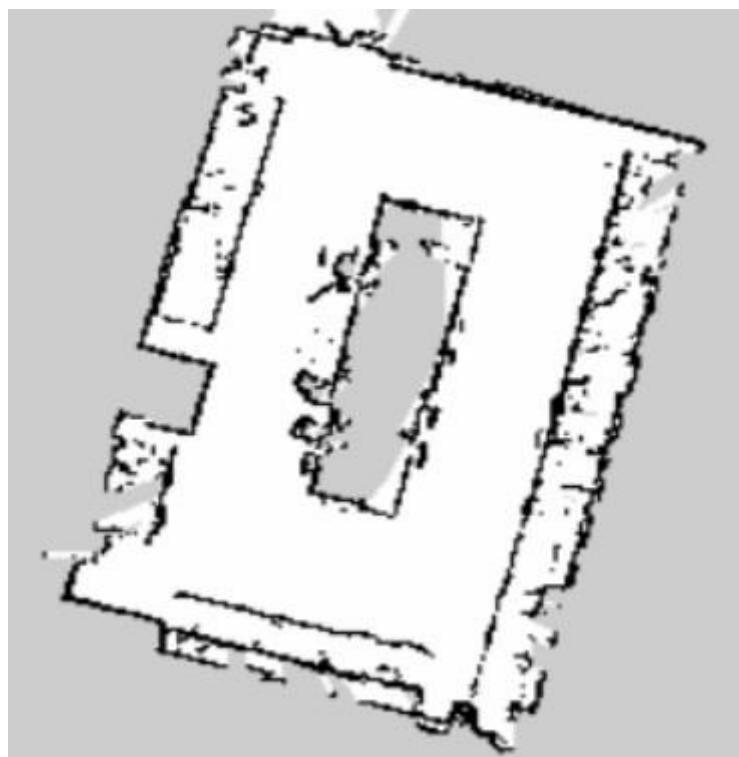
Trong đó các thông số điều chỉnh là:

K_p - Độ lợi tỉ lệ: giá trị càng lớn thì đáp ứng càng nhanh do đó sai số càng lớn, bù khâu tỉ lệ càng lớn. Một giá trị độ lợi tỉ lệ quá lớn sẽ dẫn đến quá trình mất ổn định và dao động.

K_i - Độ lợi tích phân: giá trị càng lớn kéo theo sai số ổn định bị khử càng nhanh. Đổi lại là độ vọt lố càng lớn: bất kỳ sai số âm nào được tích phân trong suốt đáp ứng quá độ phải được triệt tiêu tích phân bằng sai số dương trước khi tiến tới trạng thái ổn định.

K_d - Độ lợi vi phân: giá trị càng lớn càng giảm độ vọt lố, nhưng lại làm chậm đáp ứng quá độ và có thể dẫn đến mất ổn định do khuếch đại nhiều tín hiệu trong phép vi phân sai số.

3.3.2 Tạo lập bản đồ và định vị Robot trong bản đồ (SLAM) [11]



Hình 3. 13: Bản đồ thu được từ gói gmapping

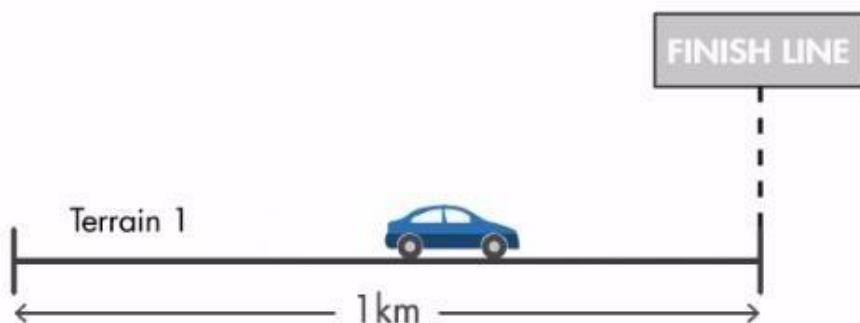
Dữ liệu cảm biến được thu thập được sẽ được sử dụng để tạo lập bản đồ hữu ích cho việc di chuyển của robot. Có nhiều thuật toán tạo lập bản đồ tuy nhiên các thuật toán hiện nay thường đi chung với định vị vị trí của robot trong chính bản đồ đã được tạo ra. Điều này thường được gọi là SLAM (tạo lập bản đồ đồng thời định vị vị trí robot). Để robot xây dựng được bản đồ, robot cần ước tính vị trí chính xác của nó và để

có được điều đó, robot cần phải có một bản đồ nhất quán. Nếu chúng ta có thông tin về bản đồ trước đó, ta có thể thực hiện định vị robot trên bản đồ này riêng lẻ qua các thuật toán khác. Trong luận văn này, chúng ta sẽ sử dụng thuật toán Gmapping để tạo lập bản đồ và AMCL để định vị vị trí cho robot.

3.3.3 Bộ lọc hạt (Practice Filter)

Bộ lọc hạt (Practice Filter) là một thuật toán ước lượng tối ưu. Nó được dùng trong việc ước lượng trạng thái của một hệ thống khi trạng thái này không thể đo lường, tính toán trực tiếp. Ví dụ với một chiếc xe di chuyển trên đường và ta cần xác định vị trí của nó trong một quãng đường cho trước, đặc biệt là kalman filter.

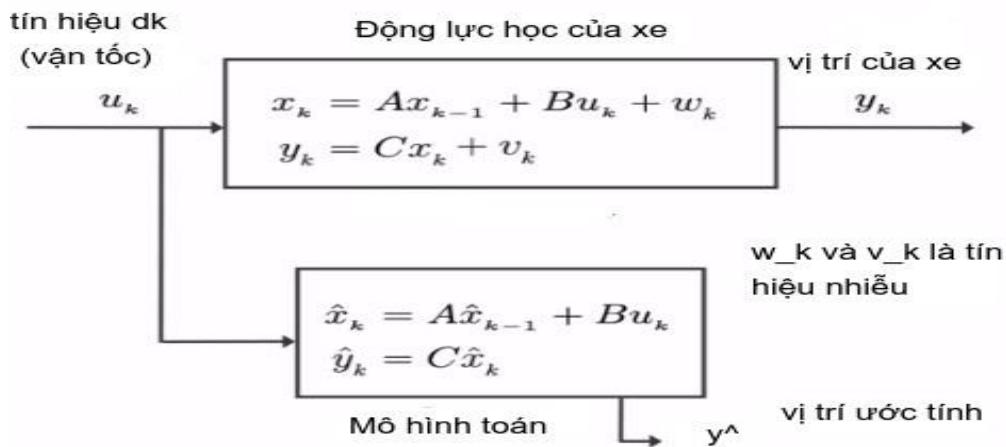
Bộ lọc Kalman (Kalman filter) được sử dụng trong dự án Apollo của NASA được phát triển bởi tiến sĩ Rudolf E. Kalman, người đã trở nên nổi tiếng với thuật toán này. Bộ lọc của ông là một bộ lọc đệ quy theo dõi trạng thái của một đối tượng trong một hệ thống tuyến tính với nhiễu. Bộ lọc này dựa trên xác suất Bayes giả định mô hình và sử dụng mô hình này để dự đoán trạng thái hiện tại từ trạng thái trước đó. Sau đó, sai số giữa giá trị dự đoán của bước trước và giá trị thực tế được đo bằng cảm biến được sử dụng để thực hiện bước cập nhật để ước tính giá trị trạng thái chính xác hơn. Bộ lọc lặp lại quá trình trên và tăng độ chính xác. Bộ lọc Kalman là một thuật toán mạnh mẽ được sử dụng trong các lĩnh vực khác nhau để ước tính trạng thái của một hệ thống với nhiễu. Nó có thể xử lý cả nhiều Gaussian và phi Gaussian và có khả năng ước tính trạng thái của một hệ thống trong thời gian thực.



Hình 3. 14: Ví dụ minh họa

Chúng ta có thể tính toán được vị trí của nó như cách tính thông tin odometry là tính dựa vào quãng đường mà nó di chuyển được thông qua vận tốc đo đạc được từ các thiết bị cảm biến bánh xe cùng với đó có thể kết hợp với việc tính toán thông qua GPS để xác định vị trí của nó trên mặt phẳng. Các việc tính toán này thường được xem là

động lực học của xe. Mặc khác, ta cũng có thể tính toán ước lượng vị trí của xe thông qua mô hình toán học động lực học của xe (như chương 2), cả hai cách tính toán này đều có sai số ảnh hưởng tới vị trí của robot. Việc sử dụng GPS hoặc tính toán thông tin odometry thường sẽ có nhiều và việc tính toán thông qua mô hình toán chỉ xem như là việc ước tính vị trí vì bỏ qua các tác động lực như lực quán tính, ma sát .v.v.



Hình 3. 15: Ví dụ về hoạt động tính toán bị trí của xe

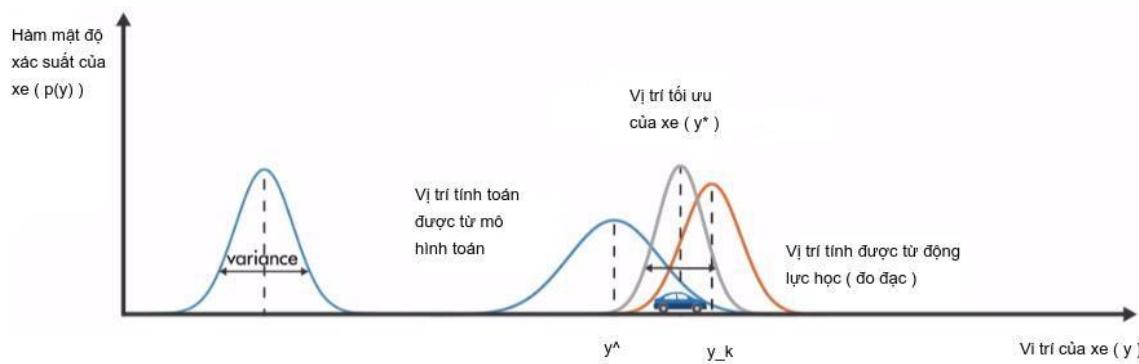
Thuật toán bộ lọc hạt cho phép chúng ta kết hợp hai mô hình này để ước tính vị trí gần như chính xác của xe trên quãng đường. Vì việc tính toán chính xác vị trí của xe là không có thật nên bài toán này đưa về bài toán xác suất, với xác suất vị trí của xe tại một điểm được dựa vào hàm mật độ $p(y)$. Vị trí của xe tại một thời điểm được phân bố theo một nhóm xác suất phân bố chuẩn Gauss (như hình), với mỗi đường cong mật độ xác suất hình chuông sẽ tương ứng với mỗi xác suất vị trí tính toán được của xe, đường cong xanh tương ứng với xác suất vị trí tính toán được thông qua mô hình toán $p(\hat{y})$, đường cong cam tương ứng với xác suất vị trí tính toán được thông qua đo đạc, động lực học $p(y_k)$ và màu xám tương ứng với xác suất vị trí tối ưu. Mỗi vị trí của xe trong khu vực mật độ xác suất được gọi là hạt (vị trí xe), thuật toán bộ lọc hạt sẽ lượt bỏ các hạt (vị trí xe) có xác suất thấp chính xác cập nhập chọn lọc cho tới khi đạt được hạt (vị trí xe) tối ưu.

Thuật toán bộ lọc Kalman được mô tả thông qua hai bước tính toán bao gồm: Uớc tính lần đầu và cập nhật việc ước tính dựa vào ước tính trước đó.

Uớc tính lần đầu (Piori estimate):

$$\hat{y}_k = Ay *_{k-1} + Bu_k$$

$$P^-_k = Ay *_{k-1} A^t + Q$$



Hình 3. 16: Hàm mật độ xác suất vị trí của xe (phân phối Gauss), xe có thể ở các vị trí trong phân phối xác suất

Đây xem như là việc tính toán không chắc chắn dựa trên mô hình toán học, với P_{-k} là phương sai của hàm mật độ xác suất. Việc tính toán này thông qua đầu vào của vị trí tối ưu trước đó y_{*-1} và phương sai trước đó. Và thông qua việc kết hợp giữa việc ước tính lần đầu và ước tính từ động lực học của xe. Ta sẽ cập nhập thông tin (Update) tính toán K_k và phương sai P_k của xác suất vị trí tối ưu y_k từ đó tính được vị trí tối ưu y_{*-1} dựa vào tính toán xác suất lớn nhất.

$$K_k = \frac{P_{-k} C^t}{C P_{-k} C^t + R}$$

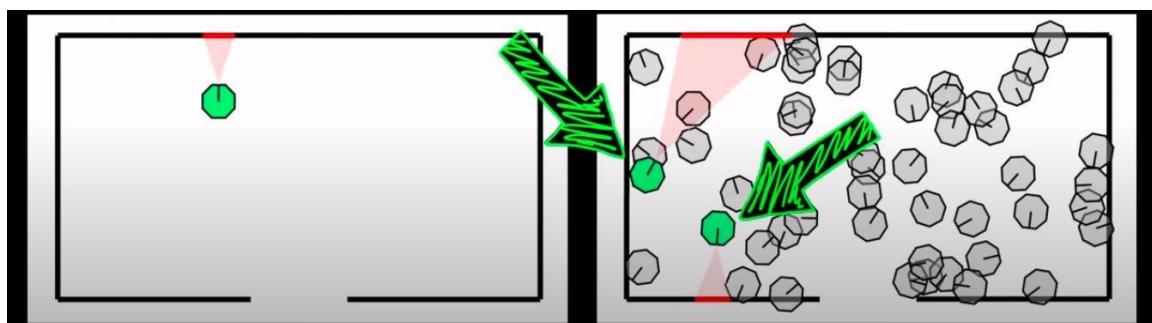
$$P_k = (I - K_k C) P_{-k}$$

Vị trí tối ưu của xe tiếp theo : $y_{*-1} = \hat{y}_k + K_k \times (y_k - C \times \hat{y}_k)$

Av cứ thế ta sẽ lại dùng vị trí và phương sai vừa tính toán y_{*-1} và P_k để tính toán tiếp cho việc Ước tính lần đầu tiếp theo) P_{-k+1} và \hat{y}_{k+1} .

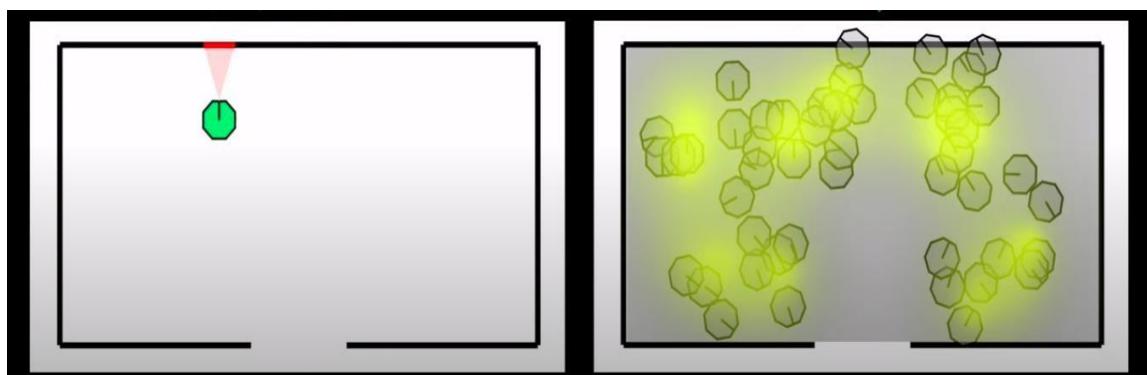
3.3.4 Thuật toán AMCL [7]

Quay trở lại bài toán định vị vị trí của robot trong một bản đồ đã có sẵn (có thể bằng phát họa tay), bộ lọc Kalman thường được dùng để kết hợp thông tin thu được từ cảm biến khoảng cách (Laser) để ước tính lần đầu và thông tin odometry thu được sẽ dùng để cập nhập bộ lọc. Đầu tiên chúng ta sẽ khởi tạo các vị trí ngẫu nhiên của robot trong bản đồ (giống như việc xe hơi có thể nằm đâu đó trong đoạn đường đã cho).



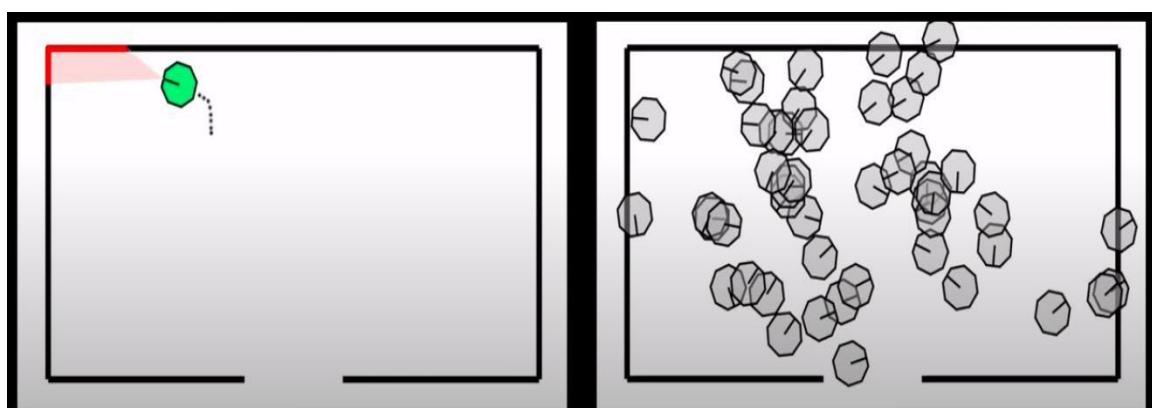
Hình 3.17: Ví dụ về robot trong một bản đồ gồm 4 bức tường (bên phải là khởi tạo ngẫu nhiên vị trí tượng trưng)

Dựa vào thông tin laser thu được (màu đỏ) ta có thể lọc đi các vị trí ngẫu nhiên đã tạo không có cùng thông tin mà nó thu được (như vị trí ngẫu nhiên 1 sẽ bị lược bỏ vì vị trí 2 thu được thông tin gần giống với thông tin robot) và xem xét các vị trí không phạm lỗi sẽ có xác suất chính xác hơn, các vị trí này sẽ được tổng hợp lại thành một mật độ xác suất và ta sẽ lại khởi tạo một số lượng các hạt ngẫu nhiên trên chính các vị trí này.



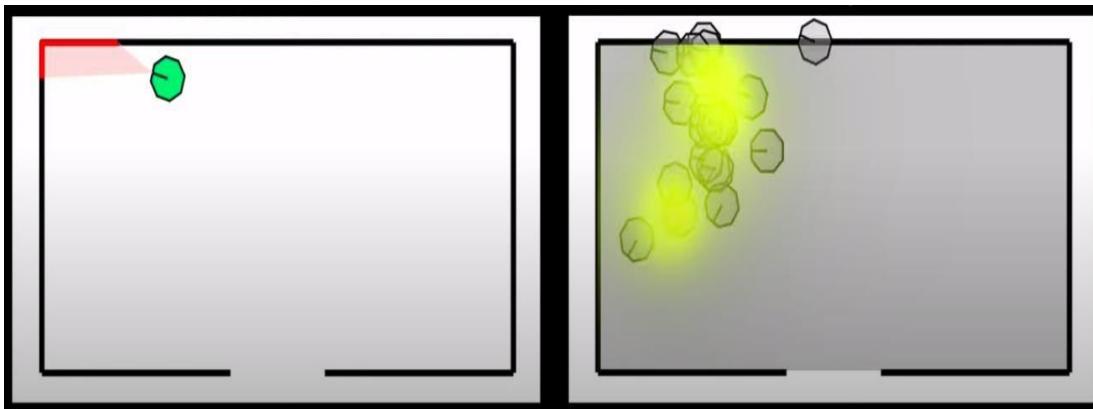
Hình 3.18: Khu vực co thể có vị trí robot sau khi lược bỏ các hạt có thông tin laser sai và khởi tạo lại các hạt trong các khu vực này

Tiếp đó, ta sẽ di chuyển robot đồng thời di chuyển tất cả các hạt vị trí ngẫu nhiên đã tạo trước đó để có thể thu thập được thông tin về odometry của robot, từ đây ta lại dung thông tin laser để cập nhật lại các vị trí thu được thông tin laser giống nhau như trên và lọc tiếp các hạt không trùng.



Hình 3.19: Di chuyển robot đồng thời di chuyển tất cả các hạt

Và cứ lặp đi lặp lại việc này cho tới khi mật độ xác suất tập trung lại một điểm đó chính là vị trí tối ưu nhất của robot sau khi qua bộ lọc kalman, đây được gọi là thuật toán Monte Carlo Localization (MCL).



Hình 3. 20: Qua nhiều lần cập nhật thì mật độ hạt sẽ tập trung tại một khu vực

3.3.5 Thuật toán Gmapping [10]

Tiếp theo ta sẽ nói về bài toán tạo lập bản đồ thông qua thuật toán Gmapping. Thuật toán này sẽ giúp chúng ta tạo nên một bản đồ lưới chiếm đóng từ thông tin cảm biến khoảng cách laser và là một thuật toán bộ lọc hạt Rao-Blackwellized hiệu quả. Thuật toán cung cấp hai cách để tăng hiệu năng của việc tạo lập bản đồ là: Đưa ra một phân phối để xuất xem xét độ chính xác của các cảm biến đo lường robot và cho phép vẽ các hạt với độ chính xác cao, theo sau là kỹ thuật lấy mẫu thích ứng, giúp duy trì số lượng lớn hạt và giảm nguy cơ cạn kiệt hạt.

Ý tưởng chính của thuật toán bộ lọc Rao-Blackwellized cho bài toán SLAM là ước tính một xác suất hậu kì $p(x_{1:t}, m|z_{1:t}, u_{0:t})$ về quỹ đạo tiềm năng $x_{1:t}$ dựa trên các thông số đo đạc theo miền thời gian kí hiệu là $z_{1:t}$ và tín hiệu thông tin odometry $u_{0:t}$ và dùng xác suất hậu kì này để tính toán một xác suất hậu kì cuối cùng về bản đồ và quỹ đạo.

$$p(x_{1:t}, m|z_{1:t}, u_{0:t}) = p(m|x_{1:t}, z_{1:t})p(x_{1:t}|z_{1:t}, u_{0:t})$$

Để ước tính xác suất hậu nghiệm $p(x_{1:t}|z_{1:t}, u_{0:t})$ trên các quỹ đạo tiềm năng, thuật toán tạo lập bản đồ Rao-Blackwellized sử dụng một bộ lọc hạt trong đó một bản đồ lẻ được liên kết với mọi mẫu. Mỗi map được xây dựng dựa trên tín hiệu cảm biến $z_{1:t}$ và quỹ đạo $x_{1:t}$ tương ứng cho mỗi hạt tương ứng. Một trong những thuật toán bộ lọc hạt phổ biến nhất là bộ lọc Sampling Importance Resampling (SIR). Thuật toán bộ lọc Rao-Blackwellized SIR cho tạo lập bản đồ từng bước xử lý các tín hiệu quan sát (cảm biến) và thông tin odometry nếu chúng có sẵn. Việc này có thể hoàn thành bởi việc

cập nhập một tập mẫu đại diện cho xác suất và quỹ đạo của robot. Thuật toán này bao gồm bốn bước:

Lấy mẫu: Những hạt được tạo kế tiếp $\{x^i\}$ đạt được từ những hạt được tạo hiện tại $\{x_{t-1:t-1}^i\}$ bằng cách lấy mẫu từ phân phối đề xuất $\pi(x_t|z_{1-t}, u_{0:t})$.

Trọng số quan trọng: Một trọng số cá nhân quan trọng được giao cho từng hạt, đạt được từ trọng số w^i giải thích rằng phân phối đề xuất π nói chung không bằng phân phối thực của các trạng thái kế tiếp.

$$w^i = \frac{p(x_t^i|z_{1-t}, u_{0:t})}{\pi(x_t|z_{1-t}, u_{0:t})}$$

Lấy mẫu lại: Các hạt có trọng số w^i thấp được thay thế bằng các hạt mẫu có trọng số cao hơn. Bước này là cần thiết vì chỉ có một số lượng hạt hữu hạn mới được sử dụng để ước tính xấp xỉ một phân phối liên tục. Hơn nữa, lấy mẫu lại cho phép áp dụng một bộ lọc hạt trong trường hợp phân phối thật khác với phân phối đề xuất.

Ước tính bản đồ: Với mỗi mẫu dâng x_t^i , một bản đồ ước tính tương ứng m_t^i được tính dựa trên quỹ đạo và sự quan sát (cảm biến) trong quá khứ dựa $p(m|x_{1:t}, z_{1:t})$. Phân phối đề xuất π được tính toán trong bước lấy mẫu phải xấp xỉ với phân phối thật $p(x_{1:t}|z_{1:t}, u_{0:t})$ và có thể được chọn tùy ý. Theo Doucet lựa chọn tối ưu của phân phối đề xuất π đối với phương sai của trọng số hạt và theo giả định Markov là

$$p(x_t|m_{t-1}^i, x_t^i, z_t, u_t) = \frac{p(z_t|m_{t-1}^i, x_t^i)p(x_t|x_{t-1}^i, u_t)}{\int p(z_t|m_{t-1}^i, x')p(x'|x_{t-1}^i, u_t)dx'}$$

Trong hệ thống thuật toán này, ta sẽ xấp xỉ $p(x_t|x_{t-1}^i, u_t)$ bởi k trong một khoảng L^i dựa trên

$$L^i = [x|p(z_t|m_{t-1}^i, x)] > \varepsilon$$

Với công thức này ta có

$$p(x_t|m_{t-1}^i, x_t^i, z_t, u_t) \approx \frac{p(z_t|m_{t-1}^i, x^t)}{\int x' \in L^i p(x'|x_{t-1}^i, u_t) dx'}$$

Sau đó, ước tính xấp xỉ cục bộ phân phối này xung quanh hàm khả năng lớn nhất bởi Gauss

$$p(x_t|m_{t-1}^i, x_t^i, z_t, u_t) \cong N\left(\mu^i, \sum_t^i\right)$$

Với sự xấp xỉ này, ta được một dạng đóng phù hợp để lấy mẫu. Với mỗi hạt hai tham số có thể được xác định bởi sự đánh giá bởi hàm khả năng cho một tập hợp các

điểm {} được lấy mẫu xung quanh sự tương ứng cục bộ lớn nhất được tìm bởi quá trình quét đối xứng

$$\mu_t^i = \frac{1}{n} \sum_j^k x_j p((z_t | m_{t-1}^i, x_j)$$

$$\sum_j^i = \frac{1}{n} \sum_{j=1}^k x_j p((z_t | m_{t-1}^i, x_j) p(x_j - \mu_t^i) (x_j - \mu_t^i)^T$$

Trong đó $\eta = \sum_{j=1}^k p((z_t | m_{t-1}^i, x_j)$ là một chuẩn hóa. Việc tính hai giá trị trên cũng như quá trình quét đối xứng được thực hiện với mỗi hạt. Giá trị $\{x_j\}$ được chọn để bao phủ một khu vực phụ thuộc vào độ không chắc chắn khi đọc giá trị odometry cuối cùng và mật độ phụ thuộc vào độ phân giải của bản đồ lưới. Trọng số quan trọng có thể tính gần đúng bằng công thức:

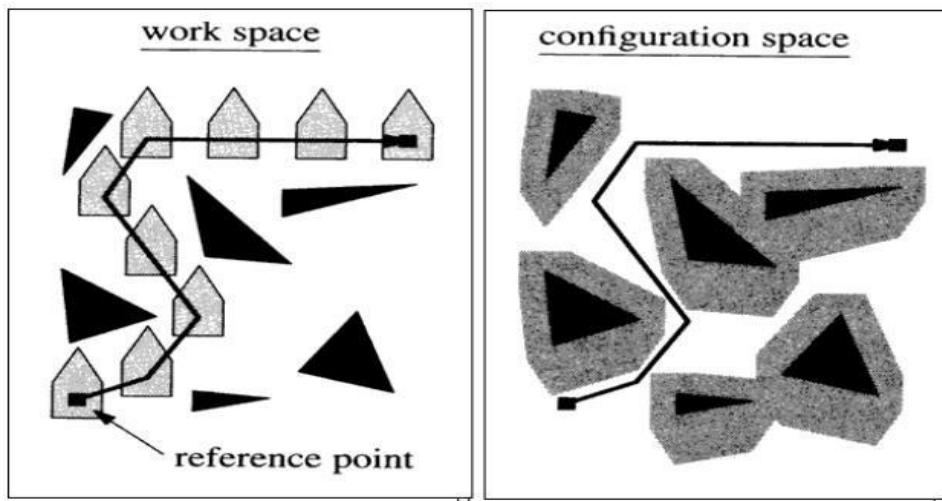
$$w_t^i = w_{t-1}^i p(z_t^i, m_{t-1}^i, u_t) = w_t^i k \eta$$

Một khía cạnh khác có ảnh hưởng lớn tới hiệu suất của bộ lọc hạt là bước lấy mẫu lại. Trong quá trình lấy mẫu. Trong quá trình lấy mẫu lại, các hạt với trọng số w_t^i nhỏ sẽ bị thay thế bởi các hạt có trọng số w_t^i lớn. Một mặt việc lấy mẫu là cần thiết vì chỉ có một số lượng hữu hạn hạt có thể sử dụng. Mặt khác, bước lấy mẫu lại có thể xóa đi các hạt mẫu tốt, gây sự suy giảm hạt. Một cách lấy hạt hiệu quả là

$$N_{eff} = \frac{1}{\sum_{i=1}^N (w^i)^2}$$

Giá trị này giúp đánh giá xem tập hợp hạt hiện tại đại diện cho xác suất hậu kì tốt như thế nào. Nếu các mẫu được lấy từ xác suất hậu kì thực sự, thì trọng số quan trọng w_t^i của các mẫu sẽ bằng nhau. Sự gần đúng càng xấu thì phương sai của các trọng số càng lớn. Vì N_{eff} có thể được coi là thước đo độ phân tán của các trọng số quan trọng, nên nó là thước đo hữu ích để đánh giá xem tập hợp hạt hậu kì thực sự như thế nào.

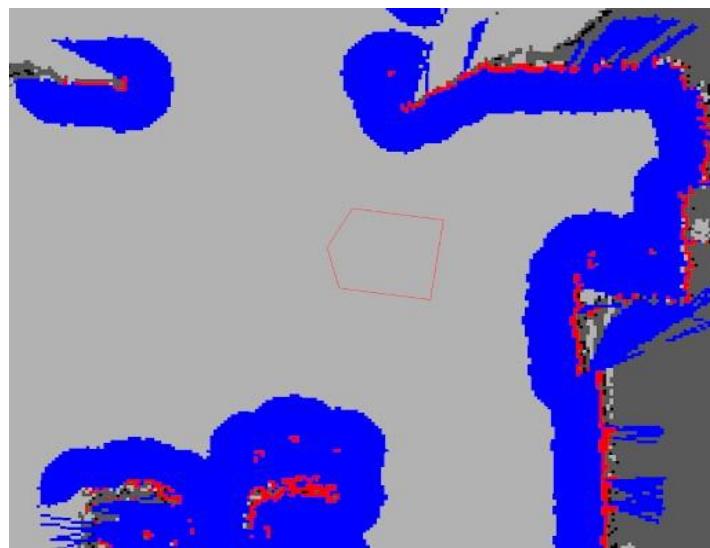
3.3.6 Thuật toán costmap_2D [9]:



Hình 3. 21: Mô tả về quỹ đạo di chuyển của robot khi có bản đồ chi phí. Vì có bản đồ chi phí nên robot sẽ tránh tiếp xúc gần với các vật cản, giúp nó di chuyển an toàn và ta có thể dễ dàng tính toán

Bản đồ chi phí thông thường giúp thổi phồng các giá trị cảm biến về vật cản giúp việc quản lý chuyển động của robot mượt mà hơn và là cần thiết trong việc bảo vệ robot an toàn, tránh các vật cản. Gói **costmap_2d** sử dụng sử dụng dữ liệu cảm biến và thông tin từ bản đồ tĩnh thu được từ trước để lưu trữ dữ liệu và cập nhập thông tin về môi trường và những vật cản xung quanh robot. Gói costmap_2d cung cấp dữ liệu thông tin dưới dạng 2D(x, y) về thông tin chiếm đóng dạng lưới của các vật cản. Nó đăng ký thông tin từ các chủ đề cảm biến để tự cập nhập các thông tin này trên bản đồ tĩnh (xóa, vẽ các vật cản trên bản đồ).

Việc thể hiện màu trên đồ lưới sẽ thêm hoặc xóa bỏ các ô màu trên bản đồ tương đương với tín hiệu cảm biến. Mỗi ô thông tin trên bản đồ có thể là không gian bị chiếm đóng, không gian trống và không gian chưa biết. Mỗi loại thường tương ứng với một giá trị màu cụ thể (0 - 255) để thể hiện rõ ràng trên bản đồ tĩnh (ta có thể tùy chỉnh để có màu khác nhau). Như hình dưới ta có khôi đa giác đó là vị trí robot, các dải ô đó là thông tin về vật cản thu được từ tín hiệu laser (không gian bị chiếm đóng), màu xám nhạt ở trong tượng trưng cho vị trí trống (không gian trống) và màu xám đậm là những thông tin chưa biết về môi trường (không gian chưa biết).

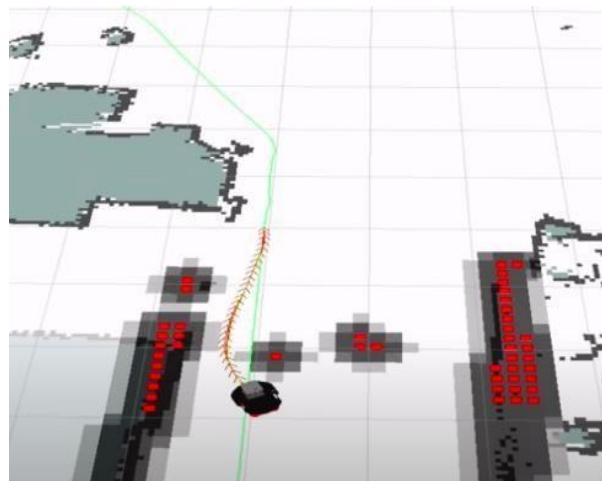


Hình 3. 22: Ví dụ về cost map trong ROS

Có hai loại bản đồ chi phí chính được khởi tạo thông qua ROS đó là: Bản đồ chi phí toàn cục (global cost map) và bản đồ chi phí cục bộ (local cost map). Bản đồ chi phí toàn cục được xây dựng trên bản đồ tĩnh lúc đầu. Trong khi đó, bản đồ chi phí cục bộ sẽ tập trung vào lúc di chuyển của robot, lấy thông tin về một bản đồ nhỏ xung quanh robot khi nó di chuyển, và tạo bản đồ chi phí trong khu vực bản đồ nhỏ này.

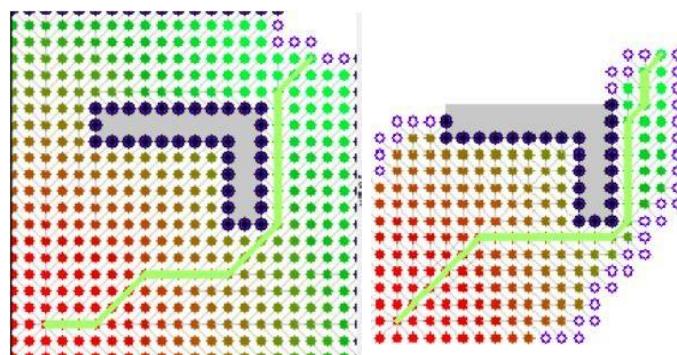
3.3.7 Chuyển động move_base

Việc robot chuyển động hoặc di chuyển cần phải thông qua các thuật toán lập kế hoạch chuyển động của robot. Thông thường đối với vấn đề điều hướng robot, sẽ có hai công đoạn để tính toán ra các tín hiệu điều khiển cho các thiết bị truyền động (motor) đó là: Lập kế hoạch đường đi (Path planning) thường là một vấn đề toàn cục, nó sẽ tạo ra một con đường cho phép robot di chuyển từ điểm này đến điểm mong muốn, tiếp sau đó là lập kế hoạch chuyển động (motion planning) thường là một vấn đề cục bộ, dùng để tính toán các lệnh hoạt động cần thiết để bán vào đường đi thu được từ lập kế hoạch đường đi. Việc lập kế hoạch chuyển động phụ thuộc vào mô hình chuyển động của robot (Chương 2) kết hợp với các thuật toán điều khiển tối ưu và các yếu tố tác động của môi trường, từ đó tính toán các đơn vị điều khiển robot như vận tốc, góc xoay bánh lái, hoặc gia tốc, ..v.v.. Để điều khiển hoạt động của robot.



Hình 3. 23: Ví dụ về đường di chuyển và quỹ đạo chuyển động của robot

Đầu tiên ta sẽ nói đến việc lập kế hoạch đường đi cho robot, thông thường là con đường ngắn nhất để dẫn đến vị trí mong muốn. Các thuật toán tìm đường ngắn nhất phổ biến như Djikstra sẽ giúp ta khám phá toàn bộ tất cả tuyến đường có thể có trong biểu đồ cho đến khi tìm thấy đường dẫn tới nút mục tiêu, sau đó tiếp tục có gắng tìm tuyến đường ngắn nhất. Một trong những thuật toán dựa trên **Djikstra**[8] và thường được sử dụng trong lập kế hoạch đường di chuyển là A*, việc khám phá bản đồ của thuật toán dựa trên hàm heuristic. Với heuristic, việc khám phá biểu đồ hiệu quả hơn vì thuật toán có thể mở rộng đầu tiên các nút có giá trị heuristic thấp hơn. Xem xét điều này, có nhiều khả năng tìm thấy con đường ngắn nhất đến mục tiêu mà không phải thực hiện tìm kiếm qua tất cả các nút trên bản đồ.



Hình 3. 24 Đường dẫn toàn cục của robot (thuật toán Djikstra, A*)

Node **move_base** tạo một đường dẫn toàn cầu thông qua gói **global planner**. Gói **global_planner** có thể sử dụng một số thuật toán lập kế hoạch đường dẫn khác nhau. Ta có thể lựa chọn đường dẫn toàn cục bằng cách set tham số.

Gói **base_local_planner** phát triển một quỹ đạo cục bộ thông qua các gói thuật toán như TEB, DWA, EBEND và cung cấp lại tín hiệu điều khiển dx, dy và dθ cho gói **move_base** để gửi tới làm chuyển động robot. **base_local_planner** là một phần quan

trọng của hệ thống nó thực hiện các quỹ đạo di chuyển ngắn tối ưu bám theo đường dẫn toàn cục (path planning) để di chuyển tới vị trí mong muốn.

3.3.8 Thuật toán Teb Local Planner [14]

TEB local planner là một thuật toán triển khai công cụ lập kế hoạch quỹ đạo cục bộ tối ưu trực tuyến để điều hướng và điều khiển robot di động như một plugin cho gói điều hướng ROS giúp suy ra các tính hiệu điều khiển . Quỹ đạo ban đầu được tạo bởi trình lập kế hoạch toàn cầu được tối ưu hóa trong thời gian chạy thực. giảm thiểu thời gian thực hiện quỹ đạo (mục tiêu tối ưu về thời gian), tách biệt khỏi các chướng ngại vật và tuân thủ các ràng buộc cơ học như thỏa mãn vận tốc và gia tốc tối đa.

Mục tiêu tổng thể là điều khiển robot từ một điểm s_s đến vị trí mong muốn s_f trong một thời gian ngắn nhất (điều khiển tối ưu). Vấn đề tối ưu hóa cơ bản được xác định trong các thuật ngữ của một vectơ tham số hữu hạn chiều bao gồm trình tự tùy ý của n cấu hình robot $(s_k)_k = 1,2,3 \dots$ Phương pháp TEB kết hợp thông tin tạm thời trực tiếp vào vấn đề tối ưu hóa và do đó giải quyết việc giảm thiểu thời gian chuyển đổi theo hạn chế cơ học. Đặt kí hiệu $(\Delta T_k)_k = 1,2, \dots, n-1$ biểu thị mỗi chuỗi thời gian dương nghiêm ngặc. Mỗi ΔT_k mô tả thời gian cần thiết để di chuyển từ trạng thái s_k tới s_{k+1} . Tập hợp các thông số để tối ưu hóa được xác định bởi :

$$\beta = (s_1, \Delta T_1, s_2, \Delta T_2, \dots, s_{n-1}, \Delta T_{n-1}, s_n)$$

Bài toán tối ưu hóa TEB được xây dựng dưới dạng phi chương trình tuyến tính:
Với các ràng buộc :

$$s_1 = s_c, s_n = s_f, 0 \leq \Delta T_k \leq \Delta T_{max}$$

$$h_k(s_{k+1}, s_k) = 0, \tilde{r}_k(s_{k+1}, s_k) \geq 0$$

$$o_k(s_k) \geq 0,$$

$$v_k(s_{k+1}, s_k, \Delta T_k) \geq 0, (k = 1,2, \dots, n-2)$$

$$\alpha_k(s_{k+2}, s_{k+1}, s_k, \Delta T_{k+1}, \Delta T_k) \geq 0, (k = 2,3, \dots, n-2)$$

$$\alpha_1(s_2, s_1 \Delta T_1) \geq 0, \alpha(s_n, s_{n-1}, s_k, \Delta T_{n-1}) \geq 0$$

Cấu hình ban đầu và cuối cùng s_k và s_{k+1} được gắn với trạng thái rô bốt hiện tại có được từ quá trình bản địa hóa rô bốt và trạng thái mục tiêu s_f . Khoảng thời gian dương đúng ΔT là giới hạn từ trên xuống ΔT_{max} để đạt được sự tùy ý của chuyển động thời gian liên tục về mặt mô hình rời rạc. Tối ưu $\sum^k \Delta T_k^2$ có xu hướng đạt được các khoảng thời gian đồng nhất (bằng chứng sau bằng phương pháp nhân Lagrange). Ràng

buộc s_k (.) thỏa mãn phương trình. Tính gần đúng của bán kính quay vòng nhỏ có giá trị trong trường hợp ΔT_{max} nhỏ. Chú ý, $\Delta \beta_k \rightarrow 0$ độ dài cung R_k có xu hướng tiến ra vô cùng. Trường hợp này là được xem xét riêng trong việc triển khai bộ giải cụ thể.

Giới hạn vận tốc và gia tốc: vận tốc tịnh tiến của cầu hình s_k tại bước thời gian rời rạc k được ký hiệu là v_k . Robot bao gồm khoảng cách R_k giữa s_k đến s_{k+1} được cho bởi p_k (chương 2) và $R_k = p_k \Delta \beta_k$. Do đó, vận tốc v_k của nó được xác định bởi:

$$v_k = \frac{p_k \beta_k}{\Delta T_k} \gamma(s_k, s_{k+1}) \approx \frac{\Delta \beta_k \ll 1}{\Delta T_k} \frac{\|d_k\|_2}{\gamma(s_k, s_{k+1})}$$

Hàm (.) thể hiện cho hướng của vận tốc của xe kể từ khi chuyển động của rô bốt bị hạn chế đối với tuyến tính và hình tròn tuần tự chuyển động, hình chiếu của vecto định hướng $q_k = [\cos \beta_k, \sin \beta_k, 0]$ vào khoảng cách d_k , kết quả là:

$$\gamma(s_k, s_{k+1}) = sign(\langle q_k, d_k \rangle) \approx \frac{k \langle q_k, d_k \rangle}{1 + |k \langle q_k, d_k \rangle|}$$

Phương trình trên được sử dụng như hầu hết các thuật toán tối ưu hóa thông thường không phù hợp với các chức năng không liên tục. Tham số $K \in R^+$ biểu thị tỷ lệ hệ số xác định độ dốc (ví dụ: $K = 10^2$).

Theo nghĩa bóng, tình huống này xảy ra nếu tư thế s_{k+1} là trực giao với s_k . Tuy nhiên, cầu hình như vậy không nằm trong tập hợp khả thi được đưa ra bởi ràng buộc h_k (chương 2) và đối với trường hợp đặc biệt, trong đó các bộ phận vị trí của cả hai các tư thế trùng nhau là $d_k = 0 \Rightarrow v_k = 0$. Cả hai thực tế và khoảng cách Euclide gần đúng của cung phân đoạn được áp dụng. Phần còn lại, chỉ có sau đó, mô tả gần đúng được điều tra. Tốc độ gốc $w_k = \frac{\Delta \beta_k}{\Delta T_k}$ của cầu hình s_k vốn dĩ ràng buộc với $|w_k| \leq w_{max}$ với $w_{max} = v_{max} p_{min}^{-1}$. Giới hạn $\pm v_{max}$ và $\pm w_{max}$ được thực thi bởi ràng buộc bất bình đẳng $v_k(s_{k+1}, s_k, \Delta T_k) = [v_{max} - |v_k|, w_{max} - |w_k|]^T$. Tương tự, gia tốc tịnh tiến a_k được giới hạn ở $\pm a_{max}$. Đặc biệt, a_k được xác định bởi sự khác biệt hữu hạn bởi:

$$a_k = \frac{2(v_{k+1} - v_k)}{\Delta T_k + \Delta T_{k+1}}$$

Để rõ ràng, s_{k+2}, s_{k+1} và s_k được thay thế bằng các vận tốc liên quan của chúng V_k . Kết quả của bất đẳng thức là :

$$a_k(s_{k+2}, s_{k+1}, s_k, \Delta T_{k+1}, \Delta T_k) = a_{max} - |a_k|$$

Các trường hợp đặc biệt xảy ra tại $k = 1$ và $k = n - 1$ với v_1 và v_{n-1} được thay thế bằng vận tốc đầu và vận tốc cuối mong muốn (V_s, ω_s) and (V_f, ω_f) tương ứng.

Giải phóng khỏi các chướng ngại vật : Quỹ đạo của robot được cho là sẽ đạt được mục tiêu mà không có bất kỳ va chạm nào với chướng ngại vật. Một chướng ngại vật được mô phỏng như một vùng được kết nối đơn giản trong không gian sinh hai và có kí hiệu là O . Khi có các chướng ngại vật $o_{R=1,2,3,\dots}$ với R là số lượng các vật cản. Khoảng cách giữa cấu hình sk và vị trí chướng ngại vật được định lượng trong không gian số liệu liên tục, số liệu Euclidean. Kí hiệu :

$$(S_k, O) : R^2 \times S^1 \times O \rightarrow R$$

Mô tả khoảng cách Euclid tối thiểu khoảng cách giữa chướng ngại vật O và tư thế sk. Khoảng cách tối thiểu δ_{min} giữa tất cả các chướng ngại vật và S_k cấu hình được xác định bởi ràng buộc bất đẳng thức:

$$o_k(s_k) = [\delta(s_k, o_1), \delta(s_k, o_2), \dots, \delta(s_k, o_R)]^T - [\delta_{min}, \delta_{min}, \dots, \delta_{min}]^T$$

Bộ chướng ngại vật O được cập nhật trực tuyến để thể hiện sự tác động giữa quỹ đạo và các môi trường. Ngoài ra, các mô hình dự đoán cho động chướng ngại vật có thể được bao gồm trong o_k .

Phương pháp tối ưu Approximative Least-Squares Optimization (Tối ưu hóa

Bình Phương Ít nhất Gần đúng) được dùng để tối ưu hóa trong bài toán này: Việc giải các chương trình phi tuyến với các ràng buộc khó tính toán rất tốn kém. Do đó, nâng cao hiệu quả của trình giải quyết trực tuyến nhanh đã là trọng tâm của nghiên cứu trong lĩnh vực phi tuyến tính tối ưu hóa trong thập kỷ qua. Phương pháp TEB nghỉ ngơi dựa trên các kỹ thuật tối ưu hóa không bị hạn chế vì chúng được nghiên cứu kỹ lưỡng và triển khai thành thực trong mã nguồn mở gói có sẵn rộng rãi. Chương trình phi tuyến chính xác (NLP) được chuyển đổi hành vấn đề tối ưu hóa bình phương phi tuyến tính gần đúng là được giải một cách hiệu quả khi bộ giải xấp xỉ Hessian bằng các dẫn xuất bậc nhất trong khi nó khai thác mô hình thưa thớt của vấn đề. Các ràng buộc được đưa vào mục tiêu.

Hoạt động như các điều khoản phạt bổ sung. Trong phần sau đối số của các ràng buộc được bỏ qua để dễ đọc hơn. Ràng buộc bình đẳng h được biểu diễn dưới dạng một bậc hai hình phạt với trọng số vô hướng và nhận dạng I bằng:

$$\emptyset(h_k, \sigma_h) = \sigma_h h_k^T I h_k = \sigma_h \|h_k\|_2^2$$

Các bất đẳng thức được ước lượng gần đúng bằng một phía có trọng số hình phạt bậc hai:

$$x(v_k, \sigma_v) = \sigma_v \|\min\{0, v_k\}\|_2^2$$

Toán tử min được áp dụng theo hàng. Các bất đẳng thức bổ sung α_k và σ_k được tính giàn đúng theo cách tương tự. Ban đầu và ràng buộc cuối cùng, ss và sf tương ứng, được loại bỏ, thay thế và do đó không phụ thuộc vào sự tối ưu hóa. Bài toán tối ưu hóa tổng thể không bị giới hạn với hàm mục tiêu $\tilde{V}(B)$, xấp xỉ (NLP) được đưa ra bởi:

$$\beta^* = \arg \min_{B \setminus \{s_1, s_n\}} \tilde{V}(B),$$

$$\begin{aligned} \tilde{V}(\beta) = & \sum_{k=1}^{n-1} [\Delta T_k^2 + \emptyset(h_k, \sigma_k) + x(\tilde{r}_k, \sigma_r) + x(v_k, \sigma_v) + x(o_k, \sigma_o) + x(a_k, \sigma_a) \\ & + x(a_n, \sigma_a)] \end{aligned}$$

β^* biểu thị vectơ giải pháp tối ưu, β^* chỉ trùng với bộ thu nhỏ thực tế của chương trình phi tuyến (NLP) trong trường hợp tất cả các trọng số đều có xu hướng vô cùng $\sigma \rightarrow \infty$. Không may, trọng số lớn đưa ra vấn đề không ổn định như vậy, rằng bộ giải cơ bản không hội tụ đúng do kích thước bước không đủ. Phương pháp TEB loại bỏ bộ giảm thiểu thực sự ủng hộ một giải pháp tối ưu nhưng hiệu quả hơn về mặt tính toán có thể đạt được với người dùng xác định trọng lượng. Đối với môi trường lọn xộn có kích thước vừa và nhỏ trong lĩnh vực nhận thức cục bộ robot, nghiên cứu cho biết rằng trọng số đơn vị là 1 cung cấp một điểm hợp lý khởi hành, ngoại trừ trọng lượng σh liên quan đến hạn chế bình đẳng của động học phi phân hình học mà lớn hơn một vài bậc của độ lớn (≈ 1000).

Phương pháp TEB sử dụng phương pháp LevenbergMarquardt (LM) do sự cân bằng thích hợp của nó giữa mạnh mẽ và hiệu quả. Khung tối ưu hóa đồ thị g2o triển khai một biến thể thưa thớt hiệu quả cao của LM. Vì các điều khoản của (NLP) chỉ phụ thuộc vào tập con nhỏ của các tham số, Hessian cơ bản là thưa thớt và dài.

Closed-loop Predictive Control (Kiểm soát dự đoán vòng kín): Phương pháp TEB xác định chiến lược kiểm soát dự đoán để giải quyết các xáo trộn, sự không chắc chắn của bản đồ và mô hình cũng như các môi trường động hướng dẫn robot khởi từ thời hiện tại S_c hướng tới từ thời mục tiêu S_f . Vấn đề (NLP) được giải quyết lặp đi lặp lại với tốc độ nhanh hơn tỷ lệ chu kỳ điều khiển robot. Tại mỗi trường hợp lấy mẫu, hành động điều khiển u_1 được suy ra từ quỹ đạo đã được tối ưu hóa. Thuật toán tổng thể được đưa ra bởi:

Procedure TIMEDELASTICBAND ($\beta, S_c, S_f, \mathcal{O}$)

Initialize or update trajectory

for all Iterations 1 to I_{teb} do

Update obstacle constraints from set \mathcal{O}

$\beta^* \leftarrow SloveNLP(\beta)$ $Slove(NLP)$

Check Feasibility

$u_1^* \leftarrow Map \beta^* to u_1^*$ obtain $[v_1, \emptyset_1]^T$ with (2)

return (sub-) optimal u_1^*

Giải pháp ban đầu thu được từ một tập hợp thưa thớt các phân đoạn đường dẫn tuyến tính không va chạm được tạo bởi một người lập kế hoạch thô. Ban đầu bao gồm các trạng thái robot sk nằm trên con đường thô cách đều không gian và thời gian. Rõ ràng, chiến lược khởi tạo ảnh hưởng đến hội tụ các giải pháp tối ưu cạnh tranh của địa phương và là thảo luận trong phần phân tích. Bộ được cho ăn trở lại trong khoảng thời gian lấy mẫu tiếp theo để khởi động ấm liên tiếp tối ưu hóa. Vì vấn đề tối ưu hóa được đề xuất giảm thiểu thời gian chuyển đổi hướng tới cấu hình cuối cùng cố định, chiến lược hoạt động với một chân trời thu hẹp. Một vòng tối ưu hóa bên ngoài với các lần lặp I_{teb} duy trì độ dài tương ứng với độ phân giải của quỹ đạo hiện tại. Số lượng cấu hình n được cập nhật tùy thuộc vào sự tùy ý theo thời gian .

if $\Delta T_k < \Delta T_{ref} - \Delta T_{hyst} > n_{min}$, remove S_k

if $\Delta T_k > \Delta T_{ref} + \Delta T_{hyst} > n_{max}$, insert S_k

được kí hiệu cho biểu thị độ phân giải thời gian mong muốn, giới thiệu một độ trễ để tránh dao động. Các số lượng mẫu được giới hạn trong $n_{min} \leq n \leq n_{max}$. Cấu hình tiêu thuyết được chèn bằng nội suy tuyến tính giữa người kế nhiệm và người tiền nhiệm.

Sự điều chỉnh độ dài của đường chân trời cho một vấn đề đường chân trời thu hẹp mang lại lợi thế chính của chiều dài đường dẫn tách và thời gian chuyển đổi tổng thể: Số tư thế cũng như số lần đảo ngược chuyển động được yêu cầu để đáp ứng là không xác định và thay đổi với chướng ngại vật động. Chèn và xóa các trạng thái trong thời gian chạy cho phép chiều dài đường dẫn tổng thể hợp đồng hoặc kéo dài, trong khi quỹ đạo bị biến dạng vẫn tuân theo thời gian áp đặt sự tùy tiện.

Theo một số sơ bộ, bộ điều khiển dự đoán với các ràng buộc bình đẳng, chẳng hạn như $S_n = S_f$, có thể được hiển thị cho thực thi sự ổn định miễn là các giải pháp tiếp theo của vấn đề tối ưu hóa cơ bản là khả thi. Tuy nhiên, một phân tích độ ổn định nằm ngoài phạm vi của bài báo này. Giả định tính khả thi của việc tối ưu hóa (gần đúng) vẫn

đề đủ cho các tình huống được điều tra. Mặt khác, có những tình huống trở ngại mà không tồn tại giải pháp thỏa mãn tất cả các ràng buộc. Vì thế tính khả thi được xác nhận bằng kiểm tra va chạm bên ngoài tại mỗi khoảng thời gian lấy mẫu trước khi áp dụng lệnh chuyển động.

CHƯƠNG 4: ROBOT XE NÂNG TỰ VẬN HÀNH TRONG KHO

XUỐNG

4.1 Tổng quan về Robot

Qua nhiều lần điều chỉnh và lắp đặt tích hợp hệ thống điều khiển vào phần khung Robot.

4.1.1 Bộ điều khiển trung tâm

Để có thể đáp ứng được khả năng cũng như tốc độ xử lý, chúng ta cần một board có khả năng xử lý đủ mạnh để chạy các thuật toán điều khiển có độ phức tạp cao. Bên cạnh đó, nhằm giảm kích thước và tăng tính linh động cho sản phẩm thì việc lựa chọn một bộ vi xử lý tích hợp có kích thước nhỏ là một sự lựa chọn hợp lý.

PC mini Lenovo ThinkCentre M75q:

- Bộ xử lý: AMD Ryzen 5
- Ram: 4GB
- Ổ đĩa cứng: 128GB SSD
- Kết nối mạng: Gigabit Ethernet, Wi-Fi và Bluetooth
- Cổng kết nối: USB, USB-C, DisplayPort, HDMI, RJ-45
- Hệ điều hành: Linux – Ubuntu 20.04.6



Hình 4. 1: Bộ điều khiển trung tâm (trước, sau)

4.1.2 Cảm biến

Cảm biến

RPLIDAR S2 là một loại cảm biến Lidar được thiết kế để phát hiện và đo khoảng cách đến các vật thể xung quanh và tạo ra một bản đồ không gian.

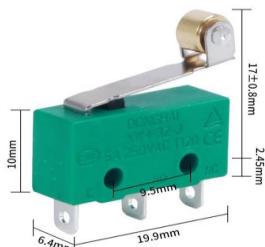
Lidar sẽ giúp xe xác định khoảng cách đến các vật thể xung quanh, tránh va chạm và xây dựng bản đồ 2D hoặc 3D của một không gian, giúp cho các hệ thống Robot hoặc tự động hóa có thể dễ dàng di chuyển

RPLIDAR S2 là một cảm biến Lidar hiệu quả và tiện lợi, giúp cho việc phát triển Robot xe nâng tự hành trở nên dễ dàng hơn và hiệu quả hơn



Hình 4. 2 Cảm biến RPLIDAR S2

4.1.3 Vi xử lý và các thiết bị khác

Tên gọi	Hình ảnh	Công dụng
Arduino Mega 2650		Điều khiển các thiết bị điện tử thông qua các cảm biến và các tín hiệu đầu vào khác
Mạch điều khiển động cơ smart H-brigde CC- Smart CC_SHB12		Điều khiển động cơ thông qua các tín hiệu điều khiển từ vi điều khiển
Mạch Hạ Áp 300W 20A Module Buck DC DC 6-40V Xuống 1.2-36V		Hạ áp từ 24V >20V Hạ áp từ 24V >12V
Motor servo		Động cơ có trang bị hệ thống encoder, giúp cho việc điều khiển vận hành robot được chính xác hơn
Limit switch		Giới hạn phạm vi chuyển động của cơ cấu nâng

Bảng 8: Một số thành phần khác của robot

4.2 Lập trình tích hợp hệ thống

4.2.1 Lập trình vi điều khiển – Arduino Mega 2560

Lập trình Arduino:

Sử dụng Arduino để điều khiển động cơ và các thiết bị điện tử là một trong những ứng dụng phổ biến nhất trong cộng đồng robot và tự động hóa.

Arduino là một công cụ đơn giản và mạnh mẽ, cho phép các nhà phát triển và học viên có thể tập trung vào việc phát triển các ứng dụng robot và tự động hóa một cách dễ dàng và nhanh chóng.

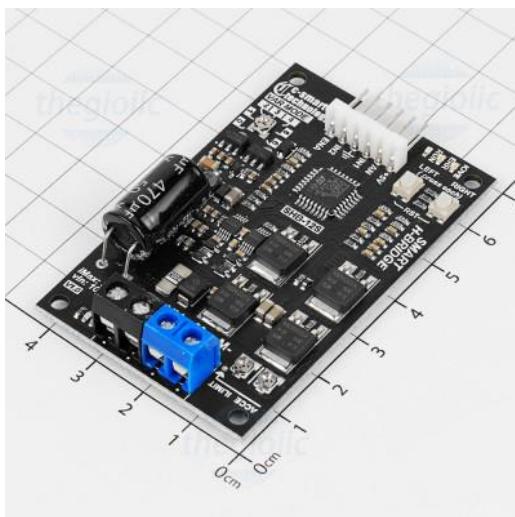
Chúng ta có thể sử dụng Arduino để điều khiển động cơ DC và động cơ bước. Đồng thời nhận tín hiệu từ các công tắc hành trình và encoder để giám sát và thu thập dữ liệu ngoại vi. Kết nối với các thiết bị điều khiển không dây thông qua mạch Bluetooth HC-05. Kết nối với ROS thông qua rosserial.

Điều khiển động cơ:

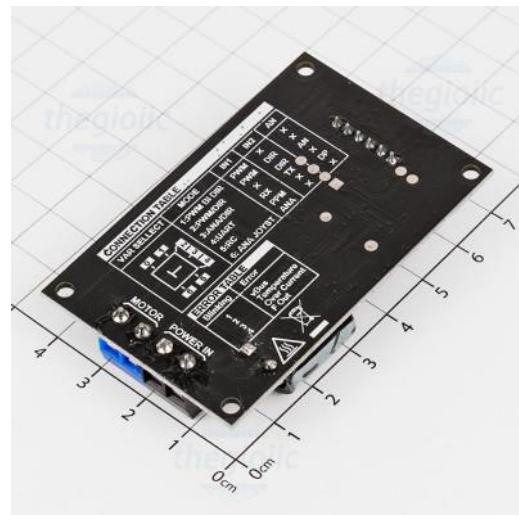
Mô tả: Arduino thu thập thông tin động cơ thông qua 2 xung A và B của Encoder gắn cùng trục với động cơ, sau khi xử lý Arduino phát ra:

- Xung PWM vào IN1: Điều khiển tốc độ động cơ.
- Tín hiệu Dir vào IN2: Điều khiển chiều quay của động cơ.

Điều khiển mạch điều khiển động cơ DC Smart H-Bridge CC-SMART CCS_SHB12



Hình 4. 3: Mặt trước mạch điều khiển động cơ DC



Hình 4. 4: Mặt sau mạch điều khiển động cơ DC

Điều khiển hệ thống cơ khí nâng hạ:

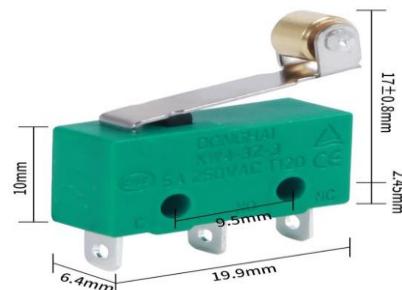
Mô tả: Arduino thu thập thông tin phần cơ khí nâng thông qua 2 công tắc hành trình ở đầu và cuối của hệ thống cơ khí, sau khi xử lý Arduino phát ra:

- Xung FUL vào FUL+: Cấp số bước hoạt động cho Stepper Motor.
- Tín hiệu Dir vào DIR+: Điều khiển chiều quay của động cơ.
- Tín hiệu EN vào ENA+: Dừng Stepper Motor.

Điều khiển mạch điều khiển Microstep Driver



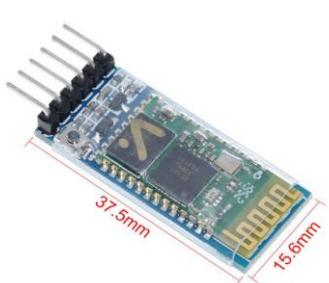
Hình 4. 5: Mạch điều khiển động cơ bước
TB6600



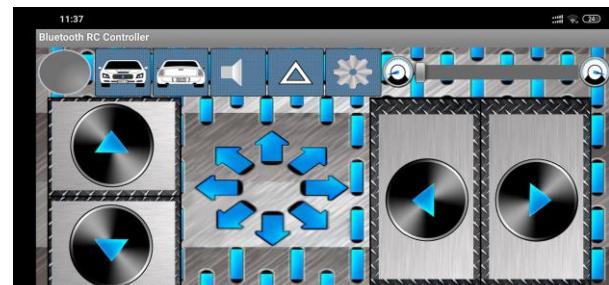
Hình 4. 6: Công tắc hành trình

(code arduino ở phần phụ lục)

Điều khiển từ xa thông qua Bluetooth:



Hình 4. 7: Mạch bluetooth HC-05



Hình 4. 8: Giao diện điều khiển robot qua bluetooth

Mô tả: Arduino kết nối với mạch Bluetooth HC – 05 thông qua kết nối USART.

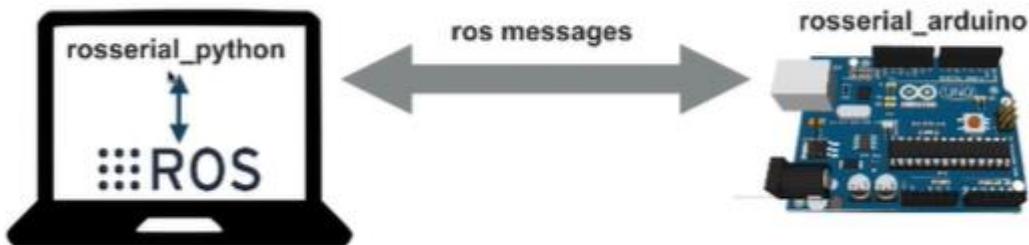
- Chân RxD (Receive Data) được sử dụng để nhận tín hiệu dữ liệu từ thiết bị gửi.
- Chân TxD (Transmit Data) được sử dụng để truyền tín hiệu dữ liệu từ thiết bị nhận.

Nhận tín hiệu điều khiển từ mạch HC-05 và được điều khiển bởi tín hiệu từ app điều khiển trên điện thoại.

Nhận tín hiệu điều khiển từ Ros qua cổng USB

Mô tả: Arduino là vi xử lý dùng để kết nối thu thập, xử lý và điều khiển các thiết bị ngoại vi. ROS là hệ điều hành để điều khiển Robot để ROS có thể kết nối với phần cứng để điều khiển robot thì Arduino là phần trung gian để ROS thu thập tín dữ liệu và điều khiển Robot.

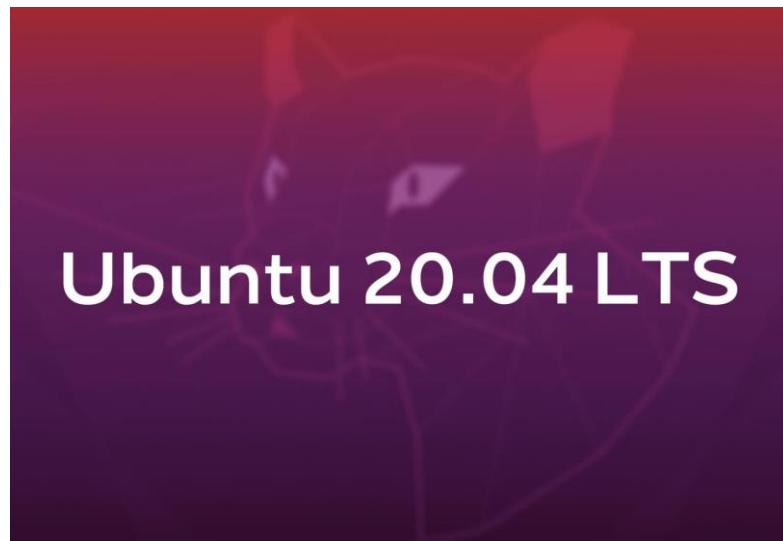
ROS có thể kết nối trực tiếp với Arduino qua Uart bằng giao thức kết nối RosSerial.



Hình 4. 9 Kết nối Rosserial

4.2.2 Cài đặt các công cụ cần thiết trên mini PC

- Cài đặt Ubuntu 20.04 LTS làm hệ điều hành Linux cho Ros Noetic:



Hình 4. 10 Logo Ubuntu 20.04 LTS

- Cài đặt Ros Noetic trên Ubuntu 20.04 LTS:

Code khởi tạo Ros Noetic trên ubuntu:

- sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu \$(lsb_release -sc) main" > /etc/apt/sources.list.d/ros-latest.list'
- sudo apt install curl # if you haven't already installed curl

- curl -s https://raw.githubusercontent.com/ros/rosdistro/master/ros.asc | sudo apt-key add -
- sudo apt update
- sudo apt install ros-noetic-desktop
- source /opt/ros/noetic/setup.bash
- roscore

→ Sử dụng lệnh roscore để chạy Ros Noetic. Sau đó truy cập vào tạo các gói package để chứa các file khởi chạy các thiết bị và các thuật toán cho robot.

- Cài đặt arduino IDE trên Ubuntu 20.04 LTS để nạp chương trình điều khiển trực tiếp lên mini PC:



Hình 4. 11: Logo Arduino IDE

- Cài đặt Rviz trên Ros Noetic để xuất bản đồ và điều khiển robot.



Hình 4. 12: Logo Rviz

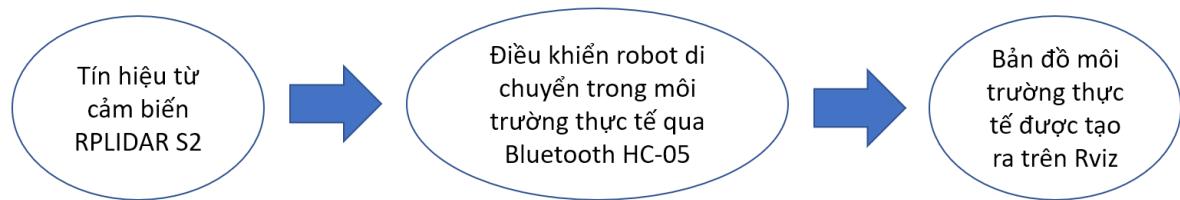
4.2.3 Thiết lập kết nối điều khiển

Để điều khiển robot đa phương tiện:

- Điều khiển robot bằng phần mềm Bluetooth RC controller trên điện thoại android để điều khiển robot thông qua modun bluetooth HC-05.
- Điều khiển robot qua wifi bằng phần mềm Ros Mobile trên điện thoại android và matlab.

4.2.4 Xây dựng bản đồ (mapping)

Để robot có thể vận hành tự động, trước tiên chúng ta cần xây dựng bản đồ 2D, cung cấp cho robot về dữ liệu môi trường, các vật thể động và tĩnh, ros đã cung cấp cho chúng ta một gói là gmapping. Sau đó robot sẽ sử dụng Navigation Stack và bản đồ đã dựng trước, dựa vào đó thuật toán Navigation Stack có thể đưa ra các thông số như vị trí tọa độ điểm đến, tọa độ các vật cản tĩnh hay một vật thể bất ngờ xuất hiện mà cảm biến lidar quét được và đưa ra phương án di chuyển phù hợp để tự động né tránh các vật cản. Đối với gmapping, gói yêu cầu dữ liệu từ odometry và một cảm biến laser range-finder. Vì thế chúng ta sẽ sử dụng cảm biến Rplidar S2 để thu thập dữ liệu và truyền về odometry kiểu dữ liệu laser range-finder. Sau đây là các bước xây dựng bản đồ bằng cảm biến Rplidar S2:



Hình 4. 13: Sơ đồ khái niệm các bước dựng bản đồ

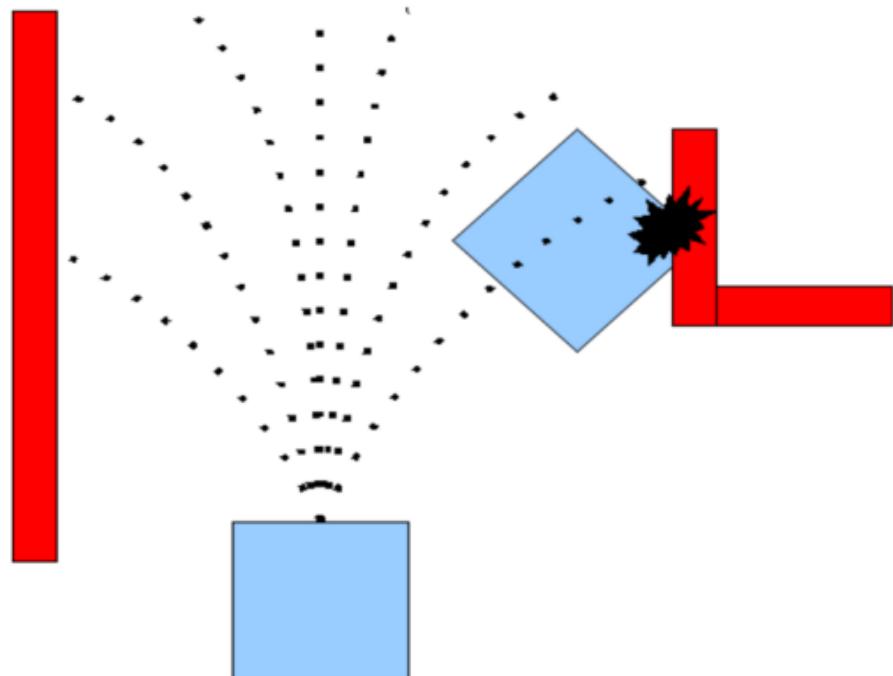
4.2.5 Điều hướng xe trên bản đồ

Để điều hướng cho xe có thể đến được đích an toàn (tránh vật cản) thì ta vẫn phải dùng cảm biến để xem xét môi trường. Nên các bước lấy dữ liệu từ laser cũng như các bước dựng bản đồ. Ngoài ra, để xác định vị trí của xe trong bản đồ thì vẫn cần đến AMCL ở mục 3.3.4 để thực hiện nhiệm vụ định vị robot.

Thuật toán AMCL cho chúng ta biết được robot đang ở vị trí nào trên bản đồ. Từ vị trí ban đầu của robot đã được xác định, ta tiến hành điều hướng robot di chuyển đến vị trí mong muốn trên bản đồ đã dựng sẵn bằng cách lập quỹ đạo di chuyển cho robot. Trước tiên cần xác định vị trí đến ở trên bản đồ có dạng tọa độ (x,y), từ đó chúng em dùng các tọa độ này để xác định nhiệm vụ điều hướng mà chúng em gửi đến ngăn xếp điều hướng của robot thực hiện nhiệm vụ đó.

Ở đây, để xây dựng được quỹ đạo di chuyển thì nhóm đã xây dựng một giải thuật Global planner[13] dựa vào tín hiệu đầu vào là cảm biến laser và đầu cuối là giá trị điểm cần đến trên bản đồ, từ đó giải thuật global planner sẽ vẽ ra một quỹ đạo tối ưu nhất robot có thể đến được vị trí cuối cùng. Quá trình thành lập quỹ đạo đã cho chúng ta một đường đi nhất định, từ quỹ đạo đã được lập, tín hiệu cần cảm biến, cũng như bản đồ đã

dựng ta tiến hành lập trình vận tốc thông qua bộ local planmer [12] để cho ra vận tốc, góc quay tức thời ứng với vị trí hiện tại của robot trên bản đồ. Bộ local planner sử dụng giải thuật Trajectory Rollout/Dynamic Window Approach để mô phỏng chuyển động của robot với các tập mẫu khác nhau, từ đó chọn tập mẫu cho kết quả tốt nhất, có thể tránh được các chướng ngại vật động.



Hình 4. 14: Quá trình mô phỏng và chọn tập mẫu để cho ra kết quả tốt nhất của robot

4.3 Thiết kế giao diện cho Robot



Hình 4. 15: Sự liên kết giữa con người và robot

Robot là công cụ thay thế con người trong nhiều công việc. Robot làm việc một cách quy tắc, nhanh chóng và chính xác. Để con người có thể làm việc với Robot thuận lợi và nâng cao hiệu suất thì giao tiếp giữa người và Robot là điều cần thiết. Giao diện giao tiếp giữa người-máy (human-computer interface) là phần hết sức quan trọng. Vì

thế nhóm sử dụng 2 phần mềm để tạo giao diện điều khiển robot là Ros Mobile chạy trên nền tảng android và Matlab GUIDE chạy trên nền tảng window (PC).

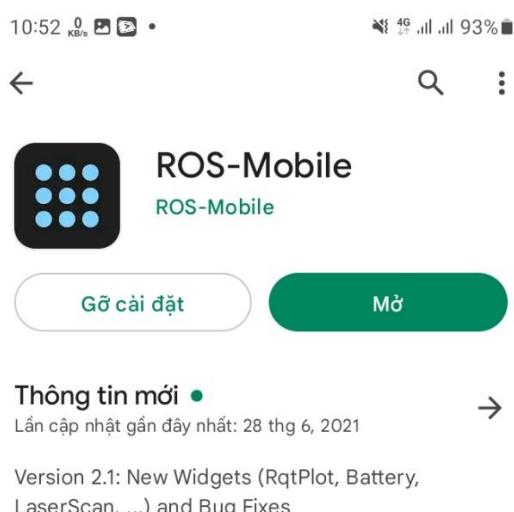
4.3.1 Ros Mobile

Giới thiệu về Ros Mobile:

"ROS Mobile" là một ứng dụng di động được phát triển bởi Open Robotics, được sử dụng để điều khiển robot bằng cách sử dụng ROS (Robot Operating System) trên các thiết bị di động như điện thoại thông minh hoặc máy tính bảng. ROS là một hệ thống phần mềm mã nguồn mở được sử dụng rộng rãi trong ngành robot để quản lý các tác vụ như điều khiển robot, xử lý dữ liệu cảm biến và lập trình ứng dụng robot. Với ROS Mobile, người dùng có thể tạo và điều khiển các robot thông qua một giao diện đơn giản trên thiết bị di động của mình. Hiện nay app chỉ tích hợp trên hệ điều hành Android.

Giao diện và sử dụng:

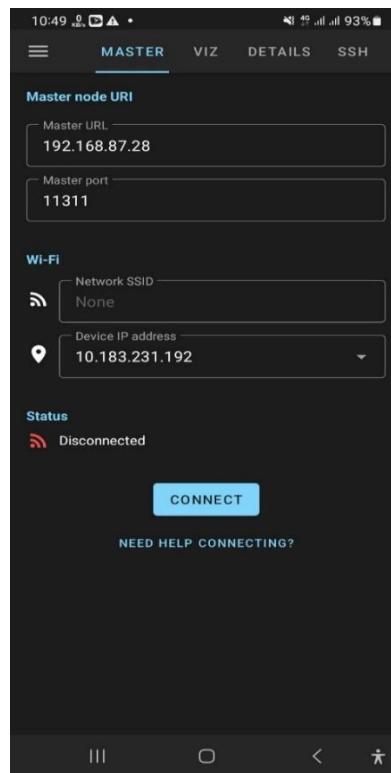
Bước 1: Tải app trên Google play.



Hình 4. 16: Ứng dụng Ros Mobile trên CH-Play

Bước 2: Kết nối Online với Ros Master.

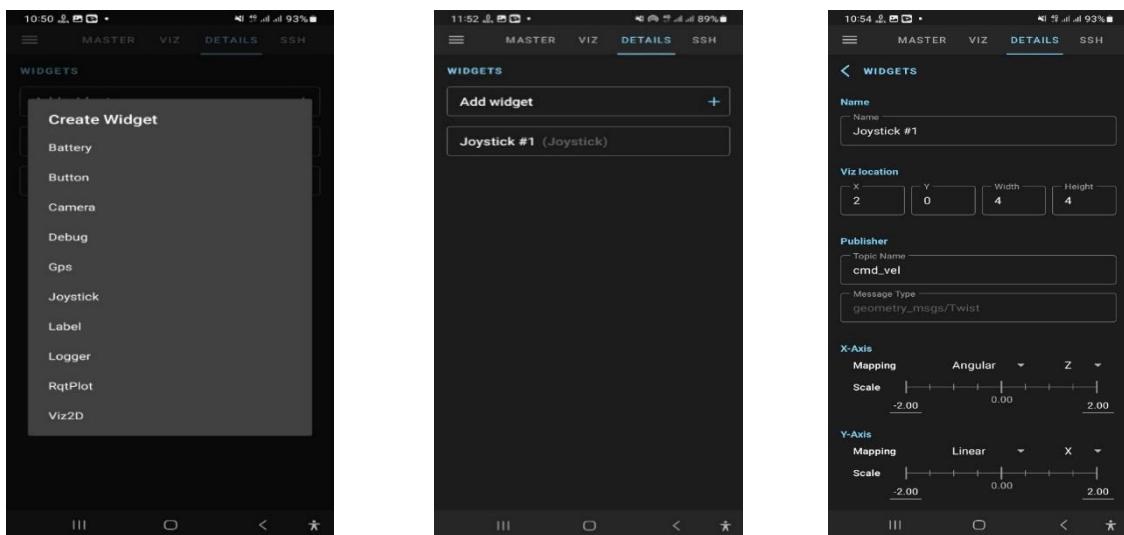
Nhấn vào phần MASTER để kết nối với Ros Master (nhập ID Master, port, địa chỉ IP và nhấn connect).



Hình 4. 17: Giao diện kết nối IP của Ros Mobile

Bước 3: Tạo giao diện điều khiển.

Nhấn vào Detail và dấu + và nhấn vào Joystick để tạo thêm nút điều khiển cho robot.



Hình 4. 18: Tạo nút điều khiển trên Ros Mobile

Vào Joystick #1 để thiết lập (Viz location, Publisher, X-Axis, Y-Axis); sau nhấn VIZ sẽ được giao diện điều khiển bằng nút trượt cảm ứng trên điện thoại Android.

Bước 4: Thiết lập Ros Topic xuất nhận tín hiệu.

Để gửi lệnh điều khiển và nhận giá trị hồi tiếp từ Mobile base, ta cần mô serial node để thực hiện nhiệm vụ này giao tiếp với hệ điều hành ROS. Cần chuyển các giá trị vận tốc được gửi qua cổng giao tiếp nói tiếp “cmd_vel” là mô message kiểu geometry_msgs/Twist.

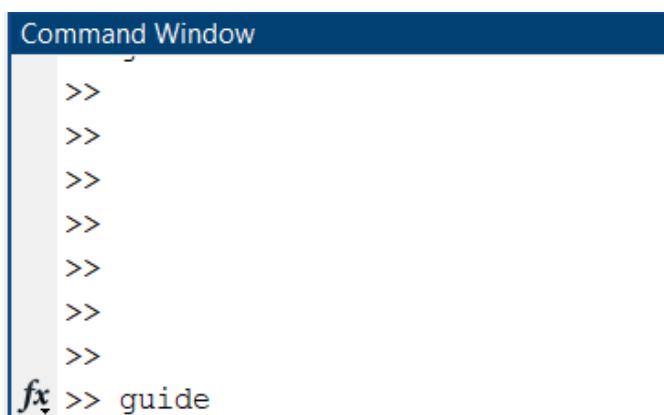
4.3.2 Matlab Gui

Giới thiệu về Matlab:

MATLAB là viết tắt của “Matrix Laboratory”, là một phần mềm cung cấp môi trường tính toán số và lập trình. Được Cleve Moler phát triển vào cuối thập niên 1970. Nguyên sơ được viết bằng ngôn ngữ Fortran và được sử dụng nội bộ trong trường Đại học Stanford. Sau này MATLAB được viết lại bằng ngôn ngữ C và được phổ biến trên thế giới dưới sự phát triển của công ty The MathWorks. Hiện nay, MATLAB đã được phát triển và cung cấp rất nhiều ứng dụng như: lập trình tính toán trên nền tảng các ma trận, thiết kế giao diện (Matlab GUI), xử lý ảnh,... Và Matlab cũng hỗ trợ các thuật toán để liên kết với ROS. Do đó MATLAB là lựa chọn thích hợp để nhóm sử dụng thiết kế giao diện giao tiếp với robot.

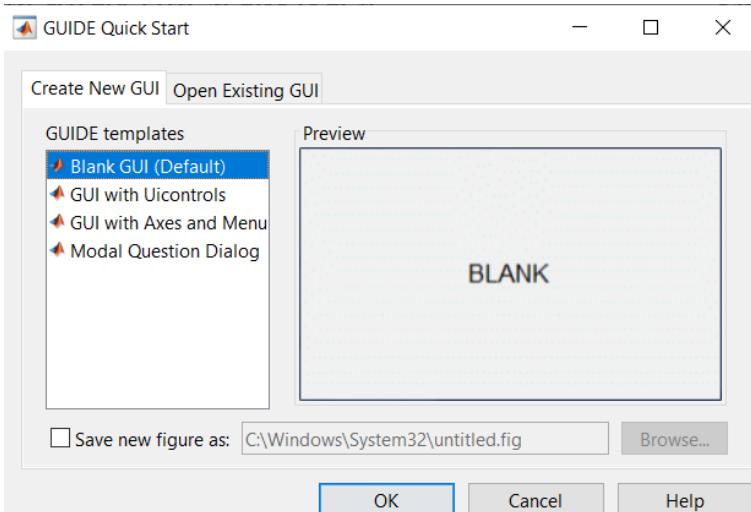
Matlab GUIDE (Graphical User Interface Development Environment) là một công cụ được tích hợp sẵn trong Matlab để xây dựng các giao diện người dùng (GUI) cho các ứng dụng Matlab. Nó cho phép người dùng thiết kế và tạo các giao diện người dùng một cách đơn giản và nhanh chóng bằng cách sử dụng các công cụ kéo và thả và các menu đơn giản. Người dùng sử dụng hàm thông dụng “Callback_function” để viết code chương trình chạy cho các thành phần trên giao diện GUI.

Để truy cập Matlab GUIDE: đầu tiên truy cập vào phần mềm Matlab, sau đó gõ vào cửa sổ Command Window. Viết vào lệnh Guide và nhấn enter.



Hình 4. 19: Cửa sổ câu lệnh khởi tạo Guide

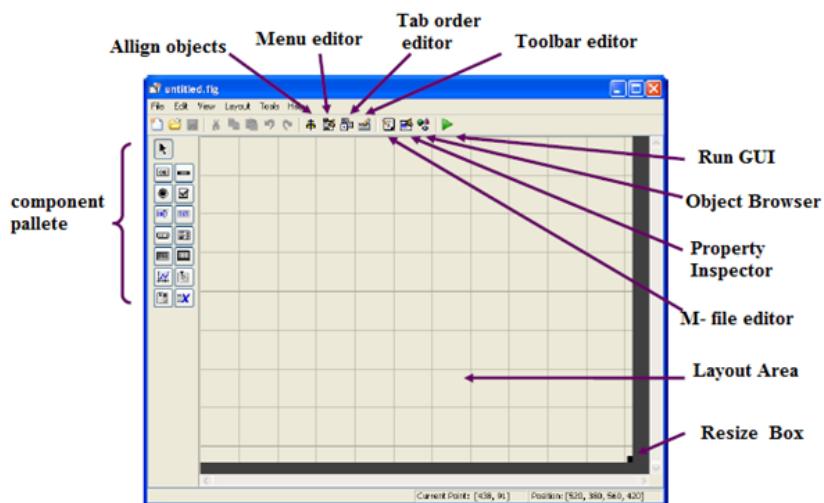
Cửa sổ tạo giao diện của Matlab Guide sẽ hiện ra, sau đó chọn vào Blank GUI (default) để tạo 1 bảng làm việc tạo giao diện GUI mới, nếu đã có sẵn chương trình, chọn Open Existing Gui.



Hình 4. 20: Tùy chọn tạo Gui

Sử dụng các tool bên trái màn hình (text, radio button, push button,...) trong phần Component palette để tạo giao diện GUI.

Để có thể vào phần viết chương trình cho các thành phần trên giao diện (GUI) ta chọn editor(M-file editor) trên thanh công cụ.



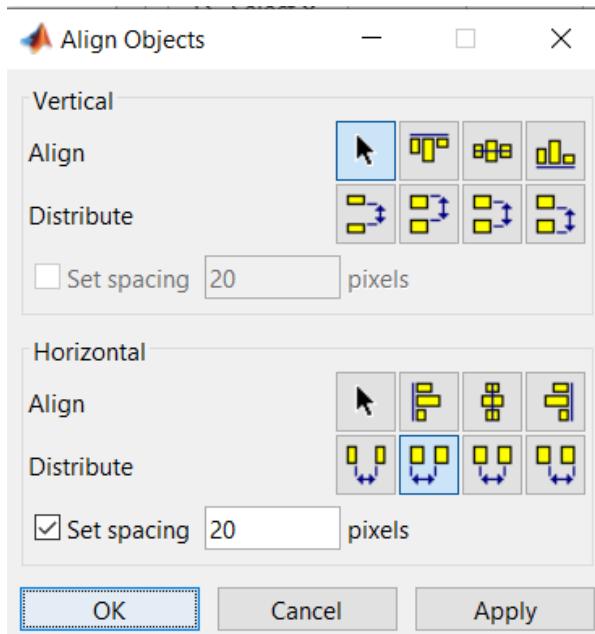
Hình 4. 21: Cửa sổ chính tạo GUI

Để chạy giao diện GUI ta chọn Run GUI trên thanh công cụ.

Để có thể thay đổi Size (Resize box) của giao diện, ta nhấn chuột và kéo ngay góc phải màn hình để thay đổi theo mong muốn.

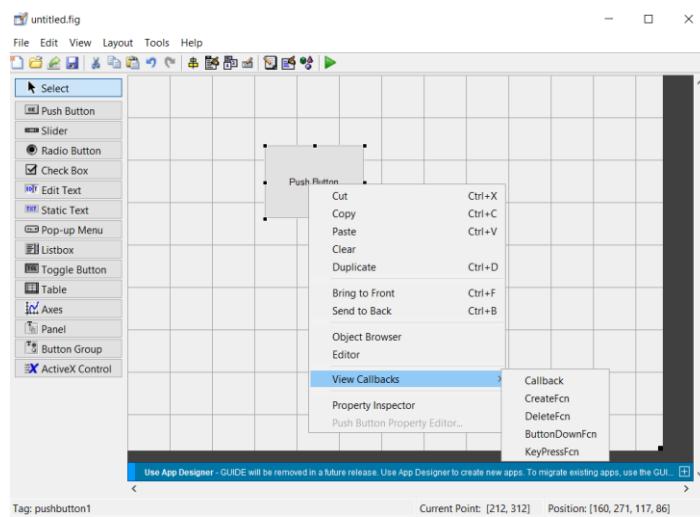
Để có thể căn chỉnh các thành phần như nút nhấn, text,.. trong giao diện cho thẳng hàng với nhau ta sử dụng chức năng Allign objects. Phần vertical dùng để căn chỉnh 2

hoặc nhiều đối tượng theo chiều dọc (chọn vào biểu tượng màu vàng của mục distribute để xem tỷ lệ căn chỉnh), tích vào ô set spacing và khoảng cách giữa các đối tượng. Phần Horizontal để căn chỉnh theo chiều ngang (chọn vào biểu tượng màu vàng của mục distribute để xem tỷ lệ căn chỉnh), mục set spacing tương tự như ở trên. Sau đó nhấn OK.



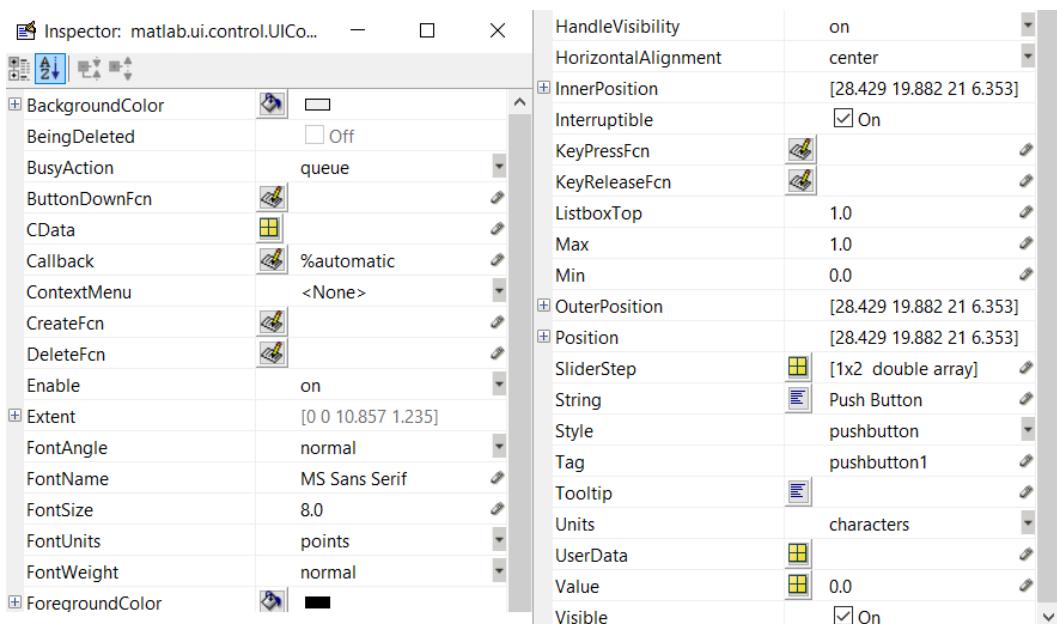
Hình 4. 22: Cửa sổ căn chỉnh các đối tượng trong giao diện

Để vào đúng đoạn chương trình để viết cho một đối tượng đã xác định, chúng ta nhập chuột phải vào đối tượng → viewCallback -> Callback.



Hình 4. 23: Gọi hàm Callback

Trong quá trình tạo các thành phần cho giao diện cho GUI như (button, text,..) ta cần phải chú ý đến thuộc tính của chúng. Chọn vào đối tượng cần set thuộc tính và chọn Property Inspector và cài đặt các nội dung quan trọng như (BackgroundColor, ForegroundColor, FontSize, FontWeight, String, Tag, Value)



Hình 4. 24: Bảng cài đặt thông số của các đối tượng trong giao diện

BackgroundColor: thay đổi màu của background đối tượng.

ForegroudColor: thay đổi màu chữ của đối tượng.

FontSize: thay đổi kích cỡ chữ của đối tượng.

FontWeight: thay đổi kiểu chữ (tô đậm chữ: Bold hoặc chữ thường: normal).

String: nội dung cần ghi vào để hiện thị lên đối tượng.

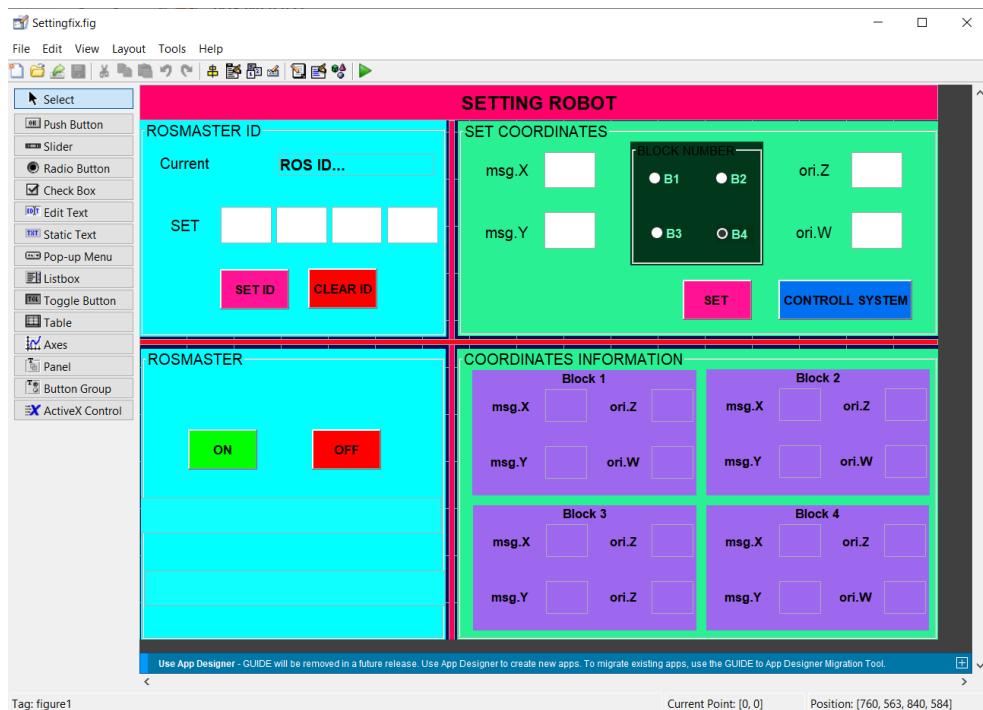
Tag: tên biến đặt cho đối tượng, biến này được sử dụng trong phần viết chương trình để gọi đối tượng.

Value: giá trị hiển thị 1 hoặc 0, thường sử dụng trong radio button, ta có thể set giá trị bằng 0 để hủy chọn hoặc bằng 1 để chọn, khi viết chương trình ta dùng hàm “a= set(handles.Biến của đối tượng, ‘value’)” để set cho một giá trị a nào đó. Sau đó xử lý giá trị a để chạy điều kiện hoặc chạy một chương trình nào đó nếu a= 1(chọn khi chạy GUI) hoặc a= 0 (hủy chọn khi chạy GUI).

4.3.3 Giao diện và giao thức truyền nhận thông tin

Giao diện lập trình tích hợp với hệ thống hiển thị giao diện matlab, sử dụng tool bên trái màn hình tạo các nút, giao diện:

Tạo giao diện cho phần Setting Robot.



Hình 4. 25: Giao diện cài đặt cho robot

Trước tiên ta tạo 1 file lưu trữ dữ liệu vị trí tọa độ của những nơi robot sẽ tự động đến khi ta chọn, ở giao diện này sẽ có 4 vị trí tọa độ được lưu trữ VT.csv.

Tiếp theo tạo 1 file lưu trữ dữ liệu của ID của RosMaster là ROSID.csv.

Để tạo 2 file trên ta phải vào thư mục lưu file GUI của matlab mà ta đã tạo trước và tạo 1 file script và lưu nó với tên là makeDatafile. Sau đó nhập code tạo bảng với x là biến để tạo file lưu trữ vị trí tọa độ và y là biến để tạo file lưu trữ ID của RosMaster:

```

Current Folder
Name
Robot_Control_system...
ROSID.csv
VT.csv
Robot_Control_system...
Setting.fig
Settingfix.fig
Robot_Control_system...
Setting.m
Settingfix.m
makeDatafile.m

Editor - C:\HOC TAP\Dung\Do an\GUI Robot\makeDatafile.m
makeDatafile.m x Robot_Control_system.m x Settingfix.m x +
1 - clear; close all;clc;
2 - y= rand(10,10);
3 - csvwrite('ROSID.csv',y);
4 - X= rand(10,10);
5 - csvwrite('VT.csv',X);

```

Hình 4. 26: Tạo file lưu trữ dữ liệu

Sử dụng static text và Button group để tạo form nền cho các nút nhấn, text,....

Tạo nút **SET** (đổi background thành màu như hình, chỉnh fontSize, fontWeight, foregroundColor, String, đặt biến tag cho nút **SET**) để cài đặt tọa độ nhập vào từ msg.X và msg.Y cho 4 vị trí tọa độ và lưu vào file VT.csv

Tạo nút **CONTROL SYSTEM** (đổi background thành màu như hình, chỉnh fontSize, fontWeight, foregroundColor, String, đặt biến tag cho nút **CONTROLL SYSTEM**) để chuyển đổi sang giao diện điều khiển khi đã cài đặt xong các thông số của robot.

Tạo nút **SET ID** (đổi background thành màu như hình, chỉnh fontSize, fontWeight, foregroundColor, String, đặt biến tag cho nút **SET ID**) để cài đặt ID cho ROSMASTER.

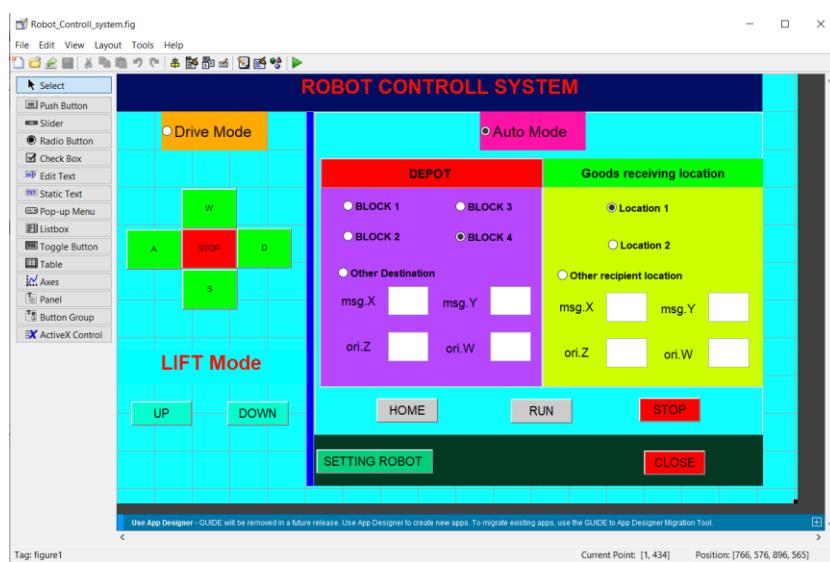
Tạo nút **CLEAR ID** (đổi background thành màu như hình, chỉnh fontSize, fontWeight, foregroundColor, String, đặt biến tag cho nút **CLEAR ID**) để xóa ID đã cài đặt ID cho ROSMASTER.

Tạo nút **ON** (đổi background thành màu như hình, chỉnh fontSize, fontWeight, foregroundColor, String, đặt biến tag cho nút **ON**) để kết nối với ROSMASTER thông qua ID đã nhập trong file ROSID.

Tạo nút **OFF** (đổi background thành màu như hình, chỉnh fontSize, fontWeight, foregroundColor, String, đặt biến tag cho nút **OFF**) để ngắt kết nối với ROSMASTER.

Tạo các nút radio button **B1 B2 B3 B4** (đổi background thành màu như hình, chỉnh fontSize, fontWeight, foregroundColor, String, đặt biến tag cho các nút chọn **B1,B2,B3,B4**). Để chọn vị trí số 4, trước khi nhấn SET vị trí thì ta chọn B4, nhập msg.X và msg.Y và nhấn SET thì hệ thống sẽ lưu tọa độ vị trí Block 4 vào file VT.csv . Để lưu các vị trí khác thì sẽ làm tương tự chọn vị trí cần lưu và nhập msg.X và msg.Y

Tạo giao diện cho phần Robot Controll System:



Hình 4. 27: Giao diện điều khiển robot

Tạo radio button (đổi background thành màu như hình, chỉnh fontSize, fontWeight, foregroundColor, String, đặt biến tag cho **Drive Mode**) để sử dụng chế độ điều khiển bằng tay.

Tạo nút (đổi background thành màu như hình, chỉnh fontSize, fontWeight, foregroundColor, String, đặt biến tag cho nút W) để điều khiển robot chạy tiến lên.

Tạo nút (đổi background thành màu như hình, chỉnh fontSize, fontWeight, foregroundColor, String, đặt biến tag cho nút S) để điều khiển robot chạy lùi.

Tạo nút (đổi background thành màu như hình, chỉnh fontSize, fontWeight, foregroundColor, String, đặt biến tag cho nút A) để điều khiển robot chạy sang trái.

Tạo nút (đổi background thành màu như hình, chỉnh fontSize, fontWeight, foregroundColor, String, đặt biến tag cho nút D) để điều khiển robot chạy sang phải.

Tạo nút (đổi background thành màu như hình, chỉnh fontSize, fontWeight, foregroundColor, String, đặt biến tag cho nút STOP) để điều khiển robot dừng lại; giao diện có 2 nút STOP, nút thứ nhất dành cho chế độ điều khiển bằng tay, nút thứ hai dành cho chế độ tự động điều khiển (**Auto Mode**).

Tạo nút (đổi background thành màu như hình, chỉnh fontSize, fontWeight, foregroundColor, String, đặt biến tag cho nút UP) để nâng thanh của robot lên, nâng các pallet chứa hàng hóa.

Tạo nút (đổi background thành màu như hình, chỉnh fontSize, fontWeight, foregroundColor, String, đặt biến tag cho nút DOWN) để hạ thanh của robot xuống.

Tạo radio button (đổi background thành màu như hình, chỉnh fontSize, fontWeight, foregroundColor, String, đặt biến tag cho **Auto Mode**) để sử dụng chế độ điều khiển tự động.

Tạo các nút radio button (đổi background thành màu như hình, chỉnh fontSize, fontWeight, foregroundColor, String,

đặt biến tag cho **BLOCK 1 , BLOCK 2, BLOCK 3, BLOCK 4**) để chỉ định vị trí đã cài đặt trước cho robot di chuyển đến vị trí đó tự động.

Tạo các nút radio button **Other Destination** (đổi background thành màu như hình, chỉnh fontSize, fontWeight, foregroundColor, String, đặt biến tag cho **Other Destination**) để cài đặt một tọa độ mới chưa có sẵn trong chương trình, nhấn nút **RUN** thì robot sẽ tự động chạy đến vị trí đó.

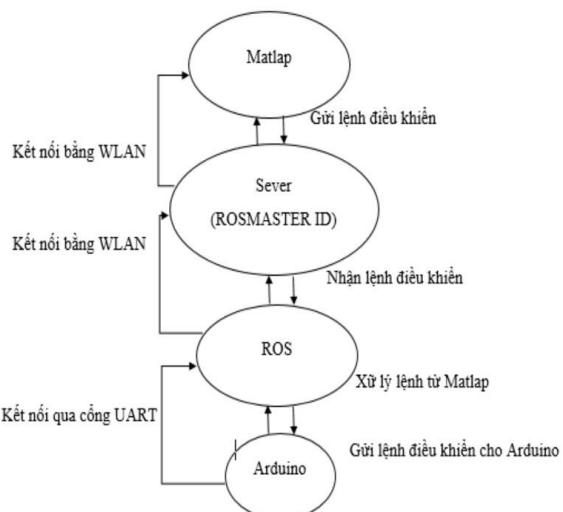
Tạo nút **HOME** (đổi background thành màu như hình, chỉnh fontSize, fontWeight, foregroundColor, String, đặt biến tag cho nút **HOME**) để robot quay về vị trí ban đầu (vị trí trước khi robot được cài đặt để đến 1 vị trí xác định khác).

Tạo nút **RUN** (đổi background thành màu như hình, chỉnh fontSize, fontWeight, foregroundColor, String, đặt biến tag cho nút **RUN**), nhấn nút **RUN** để robot tự động chạy đến 1 vị trí đã xác định (ví dụ robot chạy đến **BLOCK 1** sau khi chọn radio button của **BLOCK1**).

Tạo các nút radio button **SETTING ROBOT** (đổi background thành màu như hình, chỉnh fontSize, fontWeight, foregroundColor, String, đặt biến tag cho **SETTING ROBOT**) để chuyển đổi sang phần cài đặt cho robot.

Tạo nút **CLOSE** (đổi background thành màu như hình, chỉnh fontSize, fontWeight, foregroundColor, String, đặt biến tag cho nút **CLOSE**) để đóng giao diện điều khiển robot.

Phương thức truyền nhận:



Hình 4. 28: Lưu đồ và phương thức truyền nhận

CHƯƠNG 5: KẾT QUẢ THỰC NGHIỆM

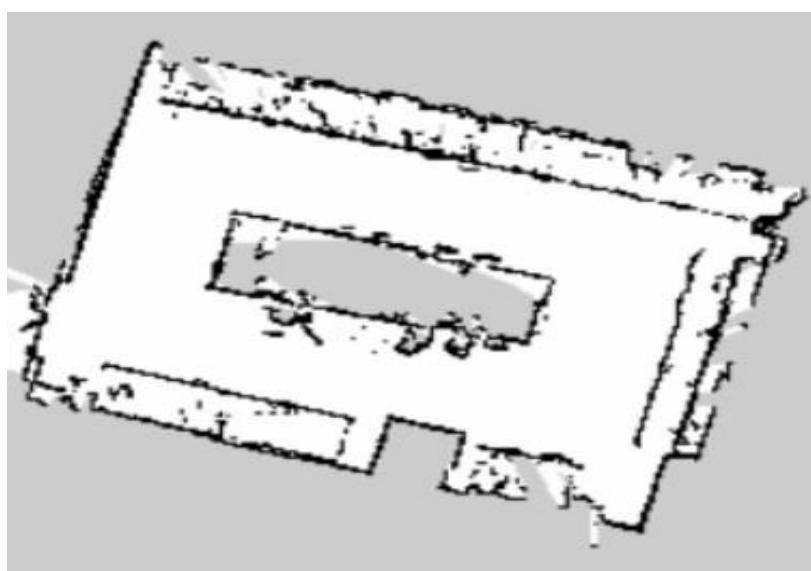
Qua quá trình xây dựng mô hình robot, thiết lập môi trường hệ điều hành Ros trên máy tính trung tâm, cấu hình và lập trình cho các node cần thiết trong quá trình vận hành, truyền nhận tín hiệu từ giao diện điều khiển matlab. Chúng ta sẽ vận hành thực nghiệm và kiểm chứng, để có thể cân chỉnh các thông số và tìm ra những ưu điểm, khuyết điểm của hệ thống.

5.1 Quá trình xây dựng bản đồ

Để có một bản đồ dựng sẵn cung cấp cho việc điều hướng thì chúng ta nên phải dựng bản đồ từ môi trường đó. Ta sử dụng những cảm biến laser sẽ là yếu tố trực tiếp để robot cảm nhận được môi trường như thế nào. Chúng ta cũng có thể tự vẽ lại một bản đồ theo đo đạc thực tế để cung cấp cho Navigation stack, nhưng có thể nó sẽ không hoạt động chính xác như ta mong đợi. Dưới đây là một số kết quả nhóm tác giả đạt được:

Kết quả đạt được khi tạo bản đồ 2D từ môi trường thực tế: Ở môi trường thông thoáng, ít vật nhiễu dữ liệu laser sẽ lạo bản đồ với độ chính xác cao. Tuy nhiên, ở môi trường có nhiều vật cản thì dữ liệu từ laser sẽ tương đối không chính xác, đôi khi khó phát hiện các góc cạnh nên sẽ tạo ra nhiều nhiễu trong quá trình lập bản đồ.

Một vấn đề nhòm thường gặp trong quá trình xây dựng bản đồ là “hiện tượng lệch map”, điều này xảy ra trong quá trình robot đang di chuyển quét map bản đồ, việc rẽ trái, hay rẽ phải với tốc độ lớn, trong khi tín hiệu lidar truyền về đôi lúc bị trễ làm cho map chồng lên map. Lỗi này bắt buộc chúng ta phải cài đặt lại thông số tốc độ khi robot rẽ trái hoặc phải và chạy lại rviz để quét map lại từ đầu.



Hình 5. 1 Bản đồ xây dựng được ở phòng X5.12

5.2 Định vị robot trên bản đồ

Chúng ta sẽ thử áp dụng gói AMCL và lý thuyết đã nêu ở trên để định vị robot trên bản đồ môi trường thử nghiệm đã thu thập được. Di chuyển robot tới một vị trí khác với lúc đầu, bản đồ sẽ cập nhật theo vị trí robot tuy nhiên sẽ khởi tạo với một bộ lọc hạt lớn tương ứng cho sự không chắc chắn vị trí của robot tại vị trí mới (vị trí robot trong môi trường gazebo và Rviz khác nhau). Ta sẽ di chuyển robot theo các hướng để bộ lọc hạt AMCL hoạt động và hội tụ (định vị robot).

5.3 Khả năng vận hành của bộ điều khiển PID:

Mục tiêu: Xây dựng PID cho robot vận hành linh hoạt, ổn định trong quá trình di chuyển của robot.

Nhận xét: Trong quá trình di chuyển của robot, nhóm nhận thấy robot di chuyển rất ổn định và chính xác, các thông số của bộ điều khiển PID đã được nhóm tính toán, thiết lập tốt để đảm bảo khả năng vận hành của robot 1 cách hiệu quả. Đáp ứng đầy đủ các yêu cầu tốc độ, cũng như khi vượt qua các tuyến đường rung lắc, hệ thống vẫn hoạt động ổn định, độ tin cậy cao.

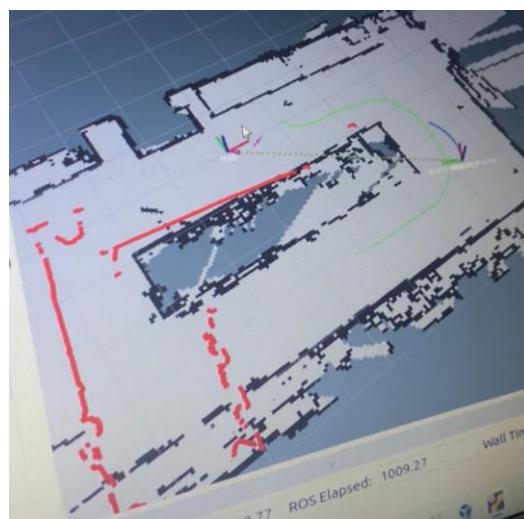
5.4 Khả năng tự hành và né tránh vật thể di động

Mục tiêu:

Khả năng điều hướng và né tránh vật cản của Robot Mục tiêu: Robot có khả năng tự quy hoạch và tính toán quãng đường ngắn nhất để đi đến vị trí đã định. Trên quãng đường, robot có khả năng tự quy hoạch lại quãng đường khi có vật cản.

Thực hiện:

- Khả năng tự quy hoạch quãng đường từ điểm A đến điểm B

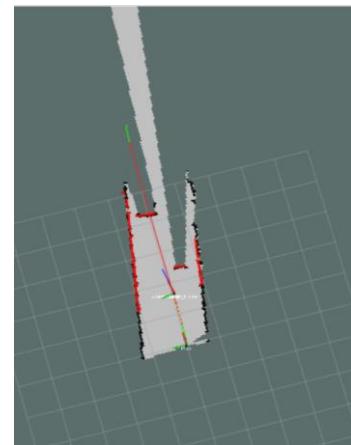


Hình 5. 2 Robot tự quy hoạch quãng đường

- Né tránh vật cản tĩnh và di động trên đường đi trên đường đi



Hình 5. 3: Môi trường thực nghiệm



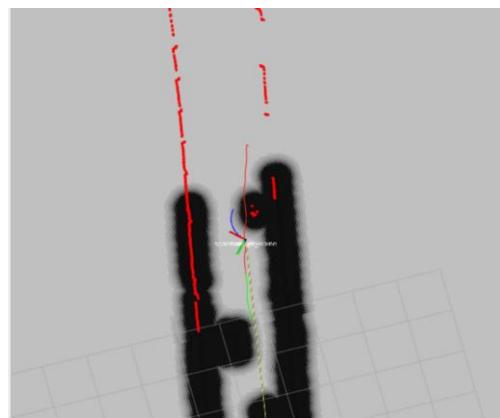
Hình 5. 4: Map trên Rviz quét được vật cản tĩnh và tự hoạch định đường di chuyển



Hình 5. 5: robot tự né vật cản tĩnh



Hình 5. 6: Trường hợp có vật cản di động



Hình 5. 7: Robot tự quy hoạch và di chuyển theo tuyến đường mới khi gặp vật cản di động

Nhận xét:

Trong quá trình vận hành thực nghiệm, khi chỉ định robot đến một vị trí nào đó trên bản đồ đã quét, robot đã có thể tự điều hướng và cho ra phương án di chuyển tối ưu

nhất, bám sát tuyến đường di chuyển mà nhóm đã đưa ra. Khi gặp vật cản trên đường di chuyển, robot đã có thể tự đưa ra phương án di chuyển mới một cách nhanh chóng, tuyến đường di chuyển mới có thể thay đổi tùy vào khoảng cách các vật cản tĩnh hoặc di động mà cảm biến lidar quét được gây cản trở việc di chuyển của robot.

5.5 Khả năng giao nhận hàng

Mục tiêu:

Robot có thể tự động lấy hàng và giao hàng đúng vị trí.

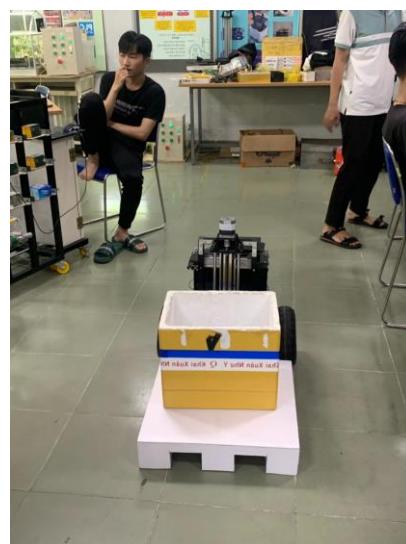
Thực hiện:

Quá trình nhận hàng robot: di chuyển đến điểm lấy hàng và nâng hàng lên. Sau đó di chuyển đến điểm trả hàng



Hình 5. 8: Robot đến địa điểm nhận hàng và nâng ballet chứa hàng

Quá trình giao hàng:



Hình 5. 9: Robot đến địa điểm trả hàng và hạ thanh nâng

Nhận xét:

Robot có thể đến đúng vị trí lấy hàng đã định trước và chuyển hàng đến vị trí theo yêu cầu người điều khiển.

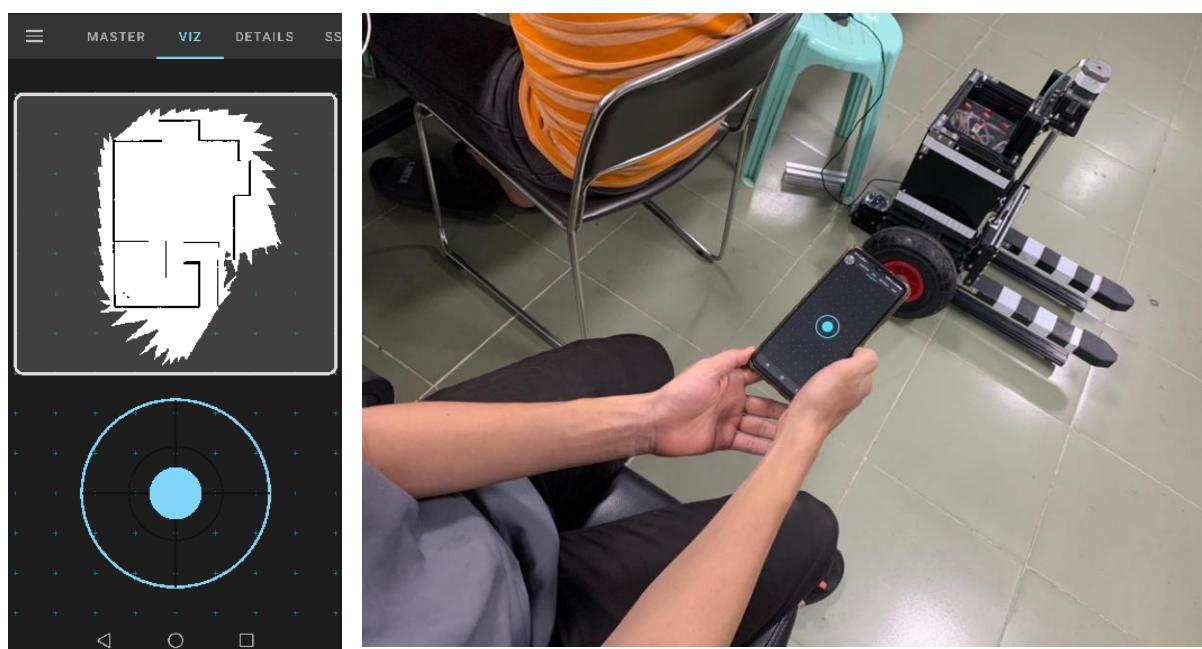
5.6 Giao diện điều khiển và giám sát

Nhóm sử dụng app Ros Mobile và app Bluetooth để điều khiển robot di chuyển, bên cạnh đó nhóm còn tạo giao diện Matlab GUI để có thể kết hợp với các thuật toán và chương trình điều khiển robot ngay từ máy tính trung tâm, giúp việc giám sát và quản lý robot trở nên dễ dàng hơn.

5.6.1 Điều khiển bằng Ros Mobile

Mục tiêu: Điều khiển robot thông qua mạng không dây nội bộ bằng app điều khiển robot có sẵn trên cửa hàng Ch-Play.

Nhận xét: Giao diện điều khiển của app Ros Mobile khá trực quan, dễ hiểu. Nhà phát hành đã tích hợp rất nhiều chức năng có thể hiện thị lên trên giao diện mà 1 Ros robot cần có. Tuy nhiên do sự tiếp cận của người dùng đối với app còn chưa nhiều, nên 1 số tính năng như xuất bản đồ và hình ảnh trên giao diện đã bị nhà phát hành hạn chế. Nhưng tính năng điều khiển thì vẫn hoạt động tốt, tín hiệu đường truyền rất ổn định. Nhóm hy vọng sẽ có thêm nhiều người dùng tiếp cận hơn nữa để nhà phát hành có thêm động lực phát triển thêm nhiều tính năng cho app.



Hình 5. 10: Giao diện Ros Mobile điều khiển robot

5.6.2 Điều khiển bằng Bluetooth

Mục tiêu: Điều khiển robot bằng bluetooth di chuyển né tránh vật cản và lấy hàng đúng vị trí.

Nhận xét: Điều khiển robot bằng app Bluetooth RC Controller thành công hơn mong đợi của nhóm. Tín hiệu bluetooth khi điều khiển rất ổn định, khi ngắt tín hiệu điều khiển, robot dừng ngay lập tức, độ trễ chỉ khoảng 0,05s, tầm hoạt động phù hợp là 10m. Nhóm cũng đã viết chương trình tích hợp khả năng nâng hàng cho robot, giúp cho con người có thể khai thác triệt để khả năng của 1 chiếc xe nâng thông thường.



Hình 5. 11: Giao diện điều khiển robot bằng bluetooth

5.6.3 Điều khiển bằng Matlab GUI

Mục tiêu: Điều khiển robot bằng giao diện do nhóm tự xây dựng trên Matlab GUI.

Nhận xét: Sau quá trình tìm hiểu về Matlab GUI, nhóm đã xây dựng thành công giao diện điều khiển với nhiều tính năng vượt trội. Giao diện hiển thị trực quan về hệ thống lưu trữ dữ liệu về mã IP RosMaster, đồng thời lưu trữ vị trí tọa độ của các nơi mà người dùng muốn robot di chuyển đến trong môi trường làm việc. Bên cạnh đó giao diện còn hiển thị đầy đủ các tính năng của hệ thống điều khiển như: chế độ điều khiển robot

bằng tay và điều khiển robot tự động, tính năng điều khiển cơ cấu nâng hạ, tính năng dừng khẩn cấp khi hệ thống gặp sự cố, tính năng quay trở về vị trí ban đầu.

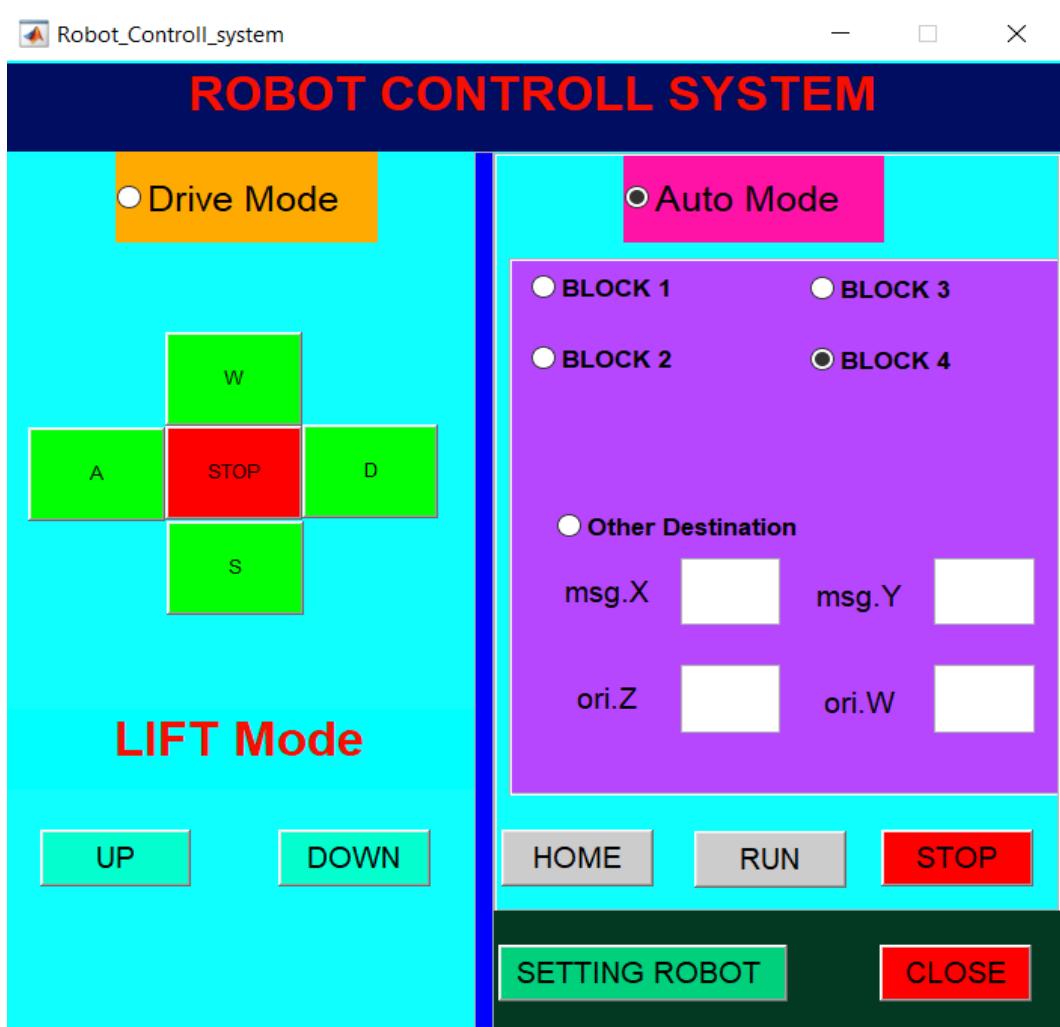
Trong quá trình hoạt động, khi sử dụng chế độ điều khiển bằng tay nhóm nhận thấy tín hiệu điều khiển từ Matlab GUI có độ trễ nhất định, điều này có thể là do tín hiệu mạng nội bộ yếu hoặc do sự đồng bộ giữa thời gian thực nhận tín hiệu của Matlab phiên bản 2020b trở xuống chưa được nhà phát hành cập nhật. Vì thế khi tạo giao diện phải chú ý sử dụng Matlab phiên bản 2020b trở đi, vì những phiên bản sau này đã được nhà phát hành tối ưu hóa quá trình truyền nhận giữa Ros và Matlab ổn định.

Quá trình hoạt động tự động của robot khá ổn định, khi nhập vị trí đã cài đặt sẵn, robot sẽ tự di chuyển, đồng thời sử dụng các thuật toán để xywr lý khi gặp vật cản khá tốt. Robot tự hoạch định 1 tuyến đường mới và đến vị trí đã định.

Một số hình ảnh về giao diện matlab điều khiển robot:



Hình 5. 12: Giao diện giao tiếp với máy tính trung tâm và robot



Hình 5. 13: Màn hình điều khiển robot từ máy tính trung tâm

CHƯƠNG 6: TỔNG KẾT NHẬN XÉT VÀ HƯỚNG PHÁT TRIỂN

6.1 Tổng kết đề tài

Qua quá trình tìm hiểu và nghiên cứu, nhóm đã được tiếp xúc, làm quen với nhiều công nghệ tiên tiến trên thế giới trong lĩnh vực chế tạo robot tự hành, cũng như các phương thức giao tiếp với robot.

Với ước muôn Việt Nam tự chủ nghiên cứu và phát triển đa dạng hóa robot phục vụ trong nghành vận chuyển hàng hóa, nhóm mong rằng lĩnh vực robot sẽ phổ biến ở Việt Nam nhiều hơn nữa. Tạo sân chơi sáng tạo cho thế hệ sau và những người đam mê robot có thể thỏa sức sáng tạo, tiếp cận với những công nghệ tiên tiến hơn để Việt Nam sớm làm chủ trong cuộc đua chế tạo robot trên thế giới hiện nay.

Dựa vào các kết quả đạt được trong đồ án lần này, nhóm đánh giá rằng việc nghiên cứu cơ bản cơ chế hoạt động dựa trên các thuật toán thành công, robot có thể tự đi từ điểm A tới điểm B và né tránh vật cản, kể cả khi có tình huống bất ngờ, robot sẽ đưa ra phương án di chuyển khác để có thể đến được đã yêu cầu.

Nhóm cũng chủ động nghiên cứu, chế tạo nguồn 24VDC, sau đó hạ áp xuống 20V và 12 V để tích hợp với các thiết bị điện tử trên xe nhằm tối ưu hóa nguồn năng lượng.

Bên cạnh đó, khả năng vận hành của robot rất vững chắc với khung nhóm kết cấu đa tầng, tuy nhiên khả năng di chuyển còn hạn chế do cơ cấu bánh xe phụ chỉ thích hợp trong các kho xưởng có nền phẳng, không gồ ghề.

Sự tiếp cận về việc tạo giao diện giữa người dùng và robot còn hạn chế, giao diện nhóm tạo ra chưa thể hiện đầy đủ các thông số của robot, cảm biến laser hoạt động ổn định cho hình ảnh map chất lượng.

Do thời gian nghiên cứu còn hạn chế nên một vài tính năng mới nhóm chưa triển khai lên robot như camera gắn trên robot để theo dõi quá trình robot hoạt động, khả năng quét bản đồ 3D đối với các đồ vật trong quá trình di chuyển của robot, tích hợp quét QR Code của hàng hóa để robot tự nhận diện và vận chuyển hàng đến vị trí đã được lưu trữ.

6.2 Hướng phát triển

Trong tương lai, nhóm sẽ nâng cấp cho robot với nhiều tính năng mới như phần tổng kết đề tài đã nói qua, nâng cao giao diện quản lý, giám sát thành một cấu trúc chỉnh

thể dễ giao tiếp và tiếp cận hơn. Nghiên cứu thêm thuật toán, chương trình để robot hoạt động mượt mà hơn, xử lý nhanh hơn với những sự cố xảy ra bất ngờ.

Trong những năm tới đây, do sự chuyển dịch cơ cấu dây chuyền sản xuất từ Trung Quốc sang các nước Đông Nam Á, đặc biệt là Việt Nam ta. Nhằm đáp ứng nhu cầu về vận chuyển hàng hóa trong kho xưởng ngày càng tăng, nhằm tối ưu hóa nguồn nhân công và năng suất lao động, sự tự chủ về công nghệ robot tự hành. Robot vận chuyển hàng hóa sẽ là công nghệ chủ chốt để thúc đẩy sự phát triển thịnh vượng của Việt Nam. Vì vậy, nhóm dự định sẽ thương mại hóa robot của nhóm sau khi đã hoàn thành những nâng cấp chủ chốt để robot có thể tự động hóa quá trình vận chuyển hàng hóa 100%.

Với kết quả đạt được từ đồ án này, đây sẽ là nền tảng cơ sở để nhóm phát triển và nâng cấp thêm cho robot với nhiều chức năng khác nhau để có thể đáp ứng được những tính năng mà một robot vận chuyển tự hành cần có.

TÀI LIỆU THAM KHẢO

- [1] Robot Operating System – Wikipedia.
<https://en.wikipedia.org/wiki/Robot_Operating_System>
- [2] About Ros Noetic – ROS.org.
<<http://wiki.ros.org/noetic>>
- [3] About packages of Ros – ROS.org.
<<http://wiki.ros.org/Packages>>
- [4] About nodes of Ros – ROS.org.
<<http://wiki.ros.org/ROS/Tutorials/UnderstandingNodes>>
- [5] Rosserial – ROS.org.
<<http://wiki.ros.org/rosserial>>
- [6] Ros navigation stack – ROS.org.
<<http://wiki.ros.org/navigation>>
- [7] Hiemstra, P., and A. Nederveen. "Monte carlo localization." *Ad Hoc Networks* 6.5: 718-733, (2007)
<<https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=9bd41c7874b7c07519d1af1b9a7a74a866cbcca9>>
- [8] Sun, Yinghui, Ming Fang, and Yixin Su. "AGV path planning based on improved Dijkstra algorithm." *Journal of Physics: Conference Series*. Vol. 1746. No. 1. IOP Publishing, 2021.
<[10.1088/1742-6596/1746/1/012052](https://doi.org/10.1088/1742-6596/1746/1/012052)>
- [9] How to costmap_2D – ROS.org.
<http://wiki.ros.org/costmap_2d>
- [10] Gmapping package – ROS.org.
<<http://wiki.ros.org/gmapping>>
- [11] AutonomuosSLAM – ROS.org.
<<http://wiki.ros.org/AutonomousSLAM>>
- [12] Tomović, Aleksandar. "Path Planning Algorithms For The Robot Operating System." p.5.
<https://www.micsymposium.org/mics2014/ProceedingsMICS_2014/mics2014_submission_2.pdf>

[13] Global planner – ROS.org.
[<http://wiki.ros.org/global_planner>](http://wiki.ros.org/global_planner)

[14] Teb local planner – ROS.org.
[<http://wiki.ros.org/treb_local_planner>](http://wiki.ros.org/treb_local_planner)

[15] Rottmann, Nils, et al. "Ros-mobile: An android application for the robot operating system." ,2020.

[<https://github.com/ROS-Mobile/ROS-Mobile-Android>](https://github.com/ROS-Mobile/ROS-Mobile-Android)

[15] Curio Res, "DC motor PID speed control" ,2021.

[<DC motor PID speed control - YouTube>](DC%20motor%20PID%20speed%20control%20-%20YouTube)

PHỤ LỤC**Arduino Code****1. Khai báo thư viện, các biến toàn cục và các chân GPIO.**

```
#include <ros.h>
#include <geometry_msgs/Twist.h>
#include <move_base_msgs/MoveBaseActionResult.h>

// StepperMotor
#include <StepperMotor.h>
#define pul      A7 // A7
#define dir_step A6 // A6
#define increase (0.03)
#define Upper_limit_switch 25
#define Lower_limit_switch 28
// Setup_StepperMotor
StepperMotor stepper(pul, dir_step);
// Setting_StepperMotor
float disPerRound = 1.0 ;// only round: 1.0 round, vitme T8-4mm: 4.0 mm, GT2
Pulley 16 Teeth: 16x2 = 32.0 mm
int microStep = 1; //1: full step, 2: haft step, ...
float angleStep = 0.05625; //a step angle of 1.8 degrees with 200 steps per
revolution
float stepsPerUnit = (1 / disPerRound) * 360.0 * microStep / angleStep;
//steps/round or steps/mm ...
float target = 0;
int count_steerer_motor=0;
float Min_target = -20;
int test_min_target = 0;

// DC_Motor_Encoder
#include <util/atomic.h>
// Declare DC1
#define ENCA_DC1 2
#define ENCB_DC1 A1
#define PWM_DC1 11
#define IN2_DC1 46
// Declare DC2
#define ENCA_DC2 3
#define ENCB_DC2 49
#define PWM_DC2 45
#define IN2_DC2 44

// Setting_PID_Motor_DC1
int pwr_DC1;
int dir_DC1;
long prevT_DC1 = 0;
int posPrev_DC1 = 0;
```

```

volatile int pos_i_DC1 = 0;
volatile float velocity_i_DC1 = 0;
volatile long prevT_i_DC1 = 0;
float eintegral_DC1 = 0;
float v1Filt = 0;
float v1Prev = 0;
// Setting_PID_Motor_DC2
int pwr_DC2;
int dir_DC2;
long prevT_DC2 = 0;
int posPrev_DC2 = 0;
volatile int pos_i_DC2 = 0;
volatile float velocity_i_DC2 = 0;
volatile long prevT_i_DC2 = 0;
float eintegral_DC2 = 0;
float v2Filt = 0;
float v2Prev = 0;
// Setting_boot_speed
float vt_DC1;
float vt_DC2;
int speed_pwm = 80;
float speed_pwm_right;
float speed_pwm_left;
int speed_ang_pwm = 40;
int rotation_speed = 30;

// Khai báo các biến toàn cục
double speed_ang;
double speed_lin_x;
double speed_lin_y;
float R_y, L_y;

```

2. Khai báo hàm Rosserial

```

void moveCallback(const geometry_msgs::Twist& msg) {
    if (0) {
        speed_lin_x = msg.linear.x*100; // limits the linear x value from -1 to
1
        speed_lin_y = msg.linear.y*100; // limits the linear x value from -1 to
1
        speed_ang = msg.angular.z*100; // limits the angular z value from -1 to 1

        if (speed_lin_y > 0) {
            R_y = 2;
            L_y = -0.7;
        }
        else {
            R_y = -0.7;
            L_y = 2;
        }
        speed_pwm_right = map(speed_lin_x, -100, 100, -speed_pwm, speed_pwm)
    }
}

```

```

        + map(speed_lin_y, -100, 100, -speed_ang_pwm,
speed_ang_pwm)*R_y
            + map(-speed_ang, -100, 100, -speed_ang_pwm,
speed_ang_pwm) ;
        speed_pwm_left = map(-speed_lin_x, -100, 100, -speed_pwm, speed_pwm)
            + map(speed_lin_y, -100, 100, -speed_ang_pwm,
speed_ang_pwm)*L_y
            + map(-speed_ang, -100, 100, -speed_ang_pwm,
speed_ang_pwm) ;
    if (speed_lin_x == 0 && speed_ang == 0 && speed_lin_y == 0) {
        vt_DC1 = 0;
        vt_DC2 = 0;
    }
    else {
        vt_DC1 = speed_pwm_right;
        vt_DC2 = speed_pwm_left;
    }
}
// Khai báo các biến ROS
ros::NodeHandle nh;
geometry_msgs::Twist action;
ros::Subscriber<geometry_msgs::Twist> sub("cmd_vel", &moveCallback);

int a = 0;
int k = 0;
// dieu khien nang hang
void moveBaseCallback(const move_base_msgs::MoveBaseActionResult& msg) {
    int status = msg.status.status; // Get the status value from the message
    if (status == 3) {
        a = a ^ 1;
    }
    if (a == k) {
    }
    else {
        k = a;
        test_min_target = k*2;
    }
}

// ROS subscriber declaration
ros::Subscriber<move_base_msgs::MoveBaseActionResult>
move_base_sub("/move_base/result", &moveBaseCallback);

// BLUETOOTH
#include <SoftwareSerial.h>
SoftwareSerial BLUETOOTH(A15, A14);
char Text_bluetooth; //for receiving bluetooth
int Mode_action_form = 0;
// Mode_action_form = 0 điều khiển chạy dài

```

```
// Mode_action_form = 1 điều khiển nâng hạ
int Mode_control_signal = 0;
```

3. Hàm setup các thông số và các hàm cơ sở

```
void setup() {
    // Tần số
    Serial.begin(115200); //230400 or 115200
    BLUETOOTH.begin(115200);

    // Khởi tạo kết nối ROS
    nh.initNode();
    nh.subscribe(sub);
    nh.subscribe(move_base_sub); Register the subscriber with the
    /move_base/result topic

    // Stepper Motor
    stepper.setStepsPerUnit(stepsPerUnit);
    stepper.setSpeed(1.0); //set 2 round/s
    stepper.setStartDirection(1);
    pinMode(Upper_limit_switch, INPUT);
    pinMode(Lower_limit_switch, INPUT);

    // DC1
    pinMode(ENCA_DC1, INPUT);
    pinMode(ENCB_DC1, INPUT);
    pinMode(PWM_DC1, OUTPUT);
    pinMode(IN2_DC1, OUTPUT);
    // DC2
    pinMode(ENCA_DC2, INPUT);
    pinMode(ENCB_DC2, INPUT);
    pinMode(PWM_DC2, OUTPUT);
    pinMode(IN2_DC2, OUTPUT);
    // Encoder
    attachInterrupt(digitalPinToInterrupt(ENCA_DC1), readEncoder_DC1, RISING);
    attachInterrupt(digitalPinToInterrupt(ENCA_DC2), readEncoder_DC2, RISING);
}

void loop() {
    if (1) {
        Bluetooth();
    }

    Stepper_Motor();

    PID();

    setMotor(dir_DC1, pwr_DC1, PWM_DC1, IN2_DC1);
    setMotor(dir_DC2, pwr_DC2, PWM_DC2, IN2_DC2);

    // Cập nhật kết nối ROS
}
```

```
nh.spinOnce();
}

void Bluetooth() {
    //Dk Bluetooth
    if (BLUETOOTH.available()) {
        Text_bluetooth = BLUETOOTH.read();
    }
    switch (Text_bluetooth) {
        case '1': {
            speed_pwm = 20;
        }
        break;

        case '2': {
            speed_pwm = 50;
        }
        break;

        case '3': {
            speed_pwm = 80;
        }
        break;

        case '4': {
            speed_pwm = 110;
        }
        break;

        case '5': {
            speed_pwm = 140;
        }
        break;

        case '6': {
            speed_pwm = 170;
        }
        break;

        case '8': {
            speed_pwm = 200;
        }
        break;

        case '9': {
            speed_pwm = 230;
        }
        break;

        case 'F': {
```

```
if (Mode_action_form == 0) {
    vt_DC1 = speed_pwm;
    vt_DC2 = -speed_pwm;
}
else {
    target = target + increase;
}
}

break;

case 'B': {
if (Mode_action_form == 0) {
    vt_DC1 = -speed_pwm;
    vt_DC2 = speed_pwm;
}
else {
    target = target - increase;
}
}

break;

case 'R': {
vt_DC1 = rotation_speed;
vt_DC2 = rotation_speed;
}
break;

case 'L': {
vt_DC1 = -rotation_speed;
vt_DC2 = -rotation_speed;
}
break;

case 'W': {
target = count_stepper_motor;
Mode_action_form = 1;
}
break;
case 'w': {
target = count_stepper_motor;
Mode_action_form = 0;
}
break;

case 'S':
vt_DC1 = 0;
vt_DC2 = 0;
break;

default:
```

```

    vt_DC1 = 0;
    vt_DC2 = 0;
    break;
}
}

void Stepper_Motor() {
    if (test_min_target == 1 && digitalRead(Lower_limit_switch) == 1) {
        target = target - 0.03;
    }
    if (test_min_target == 3) {
        target = 0;
    }
    if (digitalRead(Upper_limit_switch) == 0) {
        target = target - 0.03;
    }
    if (digitalRead(Lower_limit_switch) == 0) {
        test_min_target = 0;
        target = target + 0.03;
        Min_target = target;
    }
    if (test_min_target == 2 && target <= Min_target+10) {
        target = target + 0.03;
    }
    stepper.moveTo(target);
}

void PID() {
    int pos_DC1 = 0;
    float velocity1_DC1 = 0;
    int pos_DC2 = 0;
    float velocity2_DC2 = 0;
    ATOMIC_BLOCK(ATOMIC_RESTORESTATE) {
        pos_DC1 = pos_i_DC1;
        velocity1_DC1 = velocity_i_DC1;
        pos_DC2 = pos_i_DC2;
        velocity2_DC2 = velocity_i_DC2;
    }
    //Compute velocity with method 1
    long currT_DC1 = micros();
    float deltaT_DC1 = ((float)(currT_DC1 - prevT_DC1)) / 1.0e6;
    velocity1_DC1 = (pos_DC1 - posPrev_DC1) / deltaT_DC1;
    posPrev_DC1 = pos_DC1;
    prevT_DC1 = currT_DC1;

    //Compute velocity with method 1
    long currT_DC2 = micros();
    float deltaT_DC2 = ((float)(currT_DC2 - prevT_DC2)) / 1.0e6;
    velocity2_DC2 = (pos_DC2 - posPrev_DC2) / deltaT_DC2;
    posPrev_DC2 = pos_DC2;
    prevT_DC2 = currT_DC2;
}

```

```
//Convert count/s to RPM
float v1 = velocity1_DC1 / 600.0 * 60.0;
float v2 = velocity2_DC2 / 600.0 * 60.0;

//Low-pass filter (25 Hz cutoff)
v1Filt = 0.854 * v1Filt + 0.0728 * v1 + 0.0728 * v1Prev;
v1Prev = v1;
v2Filt = 0.854 * v2Filt + 0.0728 * v2 + 0.0728 * v2Prev;
v2Prev = v2;

// compute the control signal u
float kp_DC1 = 2.2;
float ki_DC1 = 4.2;
// compute the control signal u
float kp_DC2 = 1.9;
float ki_DC2 = 3.82;

float e_DC1 = vt_DC1 - v1Filt;
eintegral_DC1 = eintegral_DC1 + e_DC1 * deltaT_DC1;
float u_DC1 = kp_DC1 * e_DC1 + ki_DC1 * eintegral_DC1;

float e_DC2 = vt_DC2 - v2Filt;
eintegral_DC2 = eintegral_DC2 + e_DC2 * deltaT_DC2;
float u_DC2 = kp_DC2 * e_DC2 + ki_DC2 * eintegral_DC2;

//Set the motor speed and direction
dir_DC1 = -1;
if (u_DC1 < 0) {
    dir_DC1 = 1;
}
pwr_DC1 = (int)fabs(u_DC1);
if (pwr_DC1 > 255) {
    pwr_DC1 = 255;
}
//Set the motor speed and direction
dir_DC2 = 1;
if (u_DC2 < 0) {
    dir_DC2 = -1;
}
pwr_DC2 = (int)fabs(u_DC2);
if (pwr_DC2 > 255) {
    pwr_DC2 = 255;
}
}

void setMotor(int dir, int pwmVal, int pwm, int in2) {
analogWrite(pwm, pwmVal);
if (dir == 1) {
    digitalWrite(in2, LOW);
} else if (dir == -1) {
```

```

        digitalWrite(in2, HIGH);
    }
}

void readEncoder_DC1() {
    int b_DC1 = digitalRead(ENCB_DC1);
    int increment_DC1 = 0;
    if (b_DC1 > 0) {
        increment_DC1 = 1;
    } else {
        increment_DC1 = -1;
    }
    pos_i_DC1 = pos_i_DC1 + increment_DC1;

    //compute velocity with method 1
    long currT_DC1 = micros();
    float deltaT_DC1 = ((float)(currT_DC1 - prevT_i_DC1)) / 1.0e6;
    velocity_i_DC1 = increment_DC1 / deltaT_DC1;
    prevT_i_DC1 = currT_DC1;
}

void readEncoder_DC2() {
    int b_DC2 = digitalRead(ENCB_DC2);
    int increment_DC2 = 0;
    if (b_DC2 > 0) {
        increment_DC2 = 1;
    } else {
        increment_DC2 = -1;
    }
    pos_i_DC2 = pos_i_DC2 + increment_DC2;

    //compute velocity with method 2
    long currT_DC2 = micros();
    float deltaT_DC2 = ((float)(currT_DC2 - prevT_i_DC2)) / 1.0e6;
    velocity_i_DC2 = increment_DC2 / deltaT_DC2;
    prevT_i_DC2 = currT_DC2;
}

```

Matlab Code

❖ Code Setting Robot:

```

function Settingfix_OpeningFcn(hObject, eventdata, handles, varargin)
handles.output = hObject;
guidata(hObject, handles);
set(handles.SetX,'string','0');
set(handles.SetY,'string','0');
filein= 'VT.csv';
subsetA = csvread(filein);
tx1 = subsetA(1,1);
ty1 = subsetA(1,2);
tx2 = subsetA(2,1);

```

```

ty2 = subsetA(2,2);
tx3 = subsetA(3,1);
ty3 = subsetA(3,2);
tx4 = subsetA(4,1);
ty4 = subsetA(4,2);
set(handles.tx1,'string',tx1);
set(handles.tx2,'string',tx2);
set(handles.tx3,'string',tx3);
set(handles.tx4,'string',tx4);
set(handles.ty1,'string',ty1);
set(handles.ty2,'string',ty2);
set(handles.ty3,'string',ty3);
set(handles.ty4,'string',ty4);
%-----huong robot w, z auto
oriz1 = subsetA(5,1);
oriw1 = subsetA(5,2);
oriz2 = subsetA(6,1);
oriw2 = subsetA(6,2);
oriz3 = subsetA(7,1);
oriw3 = subsetA(7,2);
oriz4 = subsetA(8,1);
oriw4 = subsetA(8,2);
set(handles.oriz1,'string',oriz1);
set(handles.oriw1,'string',oriw1);
set(handles.oriz2,'string',oriz2);
set(handles.oriw2,'string',oriw2);
set(handles.oriz3,'string',oriz3);
set(handles.oriw3,'string',oriw3);
set(handles.oriz4,'string',oriz4);
set(handles.oriw4,'string',oriw4);

%-----code luu tru ROSID
filein ='ROSID.csv';
subsetC = csvread(filein);
FID1=num2str(subsetC(1,1));
FID2=num2str(subsetC(1,2));
FID3=num2str(subsetC(2,1));
FID4=num2str(subsetC(2,2));
ROSIDuse=strcat('http://',FID1,'.',FID2,'.',FID3,'.',FID4);
set(handles.TextROSID,'string',ROSIDuse);

function SetID_Callback(hObject, eventdata, handles)
fileid= 'ROSID.csv';
SID1 = str2num(get(handles.ID1,'string'));
SID2 = str2num(get(handles.ID2,'string'));
SID3 = str2num(get(handles.ID3,'string'));
SID4 = str2num(get(handles.ID4,'string'));
B = [SID1 SID2; SID3 SID4];
csvwrite(fileid,B);
subsetC = csvread(fileid);
FID1=num2str(subsetC(1,1));
FID2=num2str(subsetC(1,2));
FID3=num2str(subsetC(2,1));
FID4=num2str(subsetC(2,2));
ROSIDuse=strcat('http://',FID1,'.',FID2,'.',FID3,'.',FID4);
set(handles.TextROSID,'string',ROSIDuse);

function ClearID_Callback(hObject, eventdata, handles)
fileid= 'ROSID.csv';
B = [0 0; 0 0];

```

```

csvwrite(fileid,B);
subsetC = csvread(fileid);
FID1=num2str(subsetC(1,1));
FID2=num2str(subsetC(1,2));
FID3=num2str(subsetC(2,1));
FID4=num2str(subsetC(2,2));
ROSIDuse=strcat('http://',FID1,'.',FID2,'.',FID3,'.',FID4);
set(handles.TextROSID,'string',ROSIDuse);

function SETtXY_Callback(hObject, eventdata, handles)
filein= 'VT.csv';
subsetB = csvread(filein);
a=subsetB(1,1);
b=subsetB(1,2);
c=subsetB(2,1);
d=subsetB(2,2);
e=subsetB(3,1);
f=subsetB(3,2);
g=subsetB(4,1);
h=subsetB(4,2);
%-----huong robot auto
i=subsetB(5,1);
j=subsetB(5,2);
k=subsetB(6,1);
l=subsetB(6,2);
m=subsetB(7,1);
n=subsetB(7,2);
o=subsetB(8,1);
p=subsetB(8,2);
%-----
GT1 = get(handles.BL1, 'value');
GT2 = get(handles.BL2, 'value');
GT3 = get(handles.BL3, 'value');
GT4 = get(handles.BL4, 'value');

getValueX = get(handles.SetX, 'string');
getValueY = get(handles.SetY, 'string');
%-----gia tri huong robot auto
getValueZ = get(handles.SetOriZ, 'string');
getValueW = get(handles.SetOriW, 'string');

if (GT1==1) && (GT2==0) && (GT3==0) && (GT4==0)
    a = str2num(getValueX);
    b = str2num(getValueY);
    i = str2num(getValueZ);
    j = str2num(getValueW);
else if(GT1==0) && (GT2==1) && (GT3==0) && (GT4==0)
    c = str2num(getValueX);
    d = str2num(getValueY);
    k = str2num(getValueZ);
    l = str2num(getValueW);
else if(GT1==0) && (GT2==0) && (GT3==1) && (GT4==0)
    e = str2num(getValueX);
    f = str2num(getValueY);
    m = str2num(getValueZ);
    n = str2num(getValueW);
else (GT1==0) && (GT2==0) && (GT3==0) && (GT4==1);
    g = str2num(getValueX);
    h = str2num(getValueY);
    o = str2num(getValueZ);

```

```

    p = str2num(getValueW);
end
end
end
A = [a b; c d; e f; g h; i j; k l; m n; o p];
csvwrite(filein,A);
subsetB = csvread(filein);
tx1 = subsetB(1,1);
ty1 = subsetB(1,2);
tx2 = subsetB(2,1);
ty2 = subsetB(2,2);
tx3 = subsetB(3,1);
ty3 = subsetB(3,2);
tx4 = subsetB(4,1);
ty4 = subsetB(4,2);
%-----
set(handles.tx1,'string',tx1);
set(handles.tx2,'string',tx2);
set(handles.tx3,'string',tx3);
set(handles.tx4,'string',tx4);
set(handles.ty1,'string',ty1);
set(handles.ty2,'string',ty2);
set(handles.ty3,'string',ty3);
set(handles.ty4,'string',ty4);
%-----
oriz1 = subsetB(5,1);
oriw1 = subsetB(5,2);
oriz2 = subsetB(6,1);
oriw2 = subsetB(6,2);
oriz3 = subsetB(7,1);
oriw3 = subsetB(7,2);
oriz4 = subsetB(8,1);
oriw4 = subsetB(8,2);
set(handles.oriz1,'string',oriz1);
set(handles.oriw1,'string',oriw1);
set(handles.oriz2,'string',oriz2);
set(handles.oriw2,'string',oriw2);
set(handles.oriz3,'string',oriz3);
set(handles.oriw3,'string',oriw3);
set(handles.oriz4,'string',oriz4);
set(handles.oriw4,'string',oriw4);

function MAIN_Callback(hObject, eventdata, handles)
Robot_System = questdlg('Do you want to switch to the robot control
interface', ...
'Warning', ...
'Yes','No','No');
% Handle response
switch Robot_System
case 'Yes'
close
run Robot_Control_system.m
case 'No'
end

function CNRos_Callback(hObject, eventdata, handles)
set(handles.DCNROS, 'Value', 0);
set(handles.TSTROS1, 'BackgroundColor', 'black', 'ForegroundColor',
[0.06 1 1]);
fileid= 'ROSID.csv';

```

```

subsetA = csvread(fileid);
SID1=num2str(subsetA(1,1));
SID2=num2str(subsetA(1,2));
SID3=num2str(subsetA(2,1));
SID4=num2str(subsetA(2,2));
rosid = strcat('http://',SID1,'.',SID2,'.',SID3,'.',SID4,:11311');
setenv('ROS_MASTER_URI',rosid);
setenv('ROS_IP','192.168.45.111');
rosinit;

function CNR_Callback(hObject, eventdata, handles)
set(handles.TSTROS1, 'String', 'ROSMASTER has been conneted ^_^');
set(handles.TSTROS2, 'String','');
fileid= 'ROSID.csv';
subsetA = csvread(fileid);
SID1=num2str(subsetA(1,1));
SID2=num2str(subsetA(1,2));
SID3=num2str(subsetA(2,1));
SID4=num2str(subsetA(2,2));
rosid = strcat('http://',SID1,'.',SID2,'.',SID3,'.',SID4,:11311');
setenv('ROS_MASTER_URI',rosid);
rosinit;

function DCNR_Callback(hObject, eventdata, handles)
set(handles.TSTROS2, 'String', 'ROSMASTER has been disconnected!!!!');
set(handles.TSTROS1, 'String','');
rosshutdown;
fileid= 'ROSID.csv';
subsetA = csvread(fileid);
SID1=num2str(subsetA(1,1));
SID2=num2str(subsetA(1,2));
SID3=num2str(subsetA(2,1));
SID4=num2str(subsetA(2,2));
rosid = strcat('http://',SID1,'.',SID2,'.',SID3,'.',SID4,:11311');
setenv('ROS_MASTER_URI',rosid);

```

❖ Code Robot Controll System:

```

function DRIVE1_Callback(hObject, eventdata, handles)
set(handles.AUTO1, 'Value', 0);

function UP1_Callback(hObject, eventdata, handles)
% Create a publisher for the 'geometry_msgs/Twist' topic
twist_pub = rospublisher('/cmd_vel','geometry_msgs/Twist');
% Create a 'geometry_msgs/Twist' message object
twist_msg = rosmessage('geometry_msgs/Twist');
% Set the values of the linear x, y, z fields of the message
twist_msg.Linear.X = 1;
twist_msg.Linear.Y = 0;
% Set the values of the angular x, y, z fields of the message
twist_msg.Angular.Z = 0;
% Publish the message on the topic
send(twist_pub,twist_msg);
pause(0.01);
%-----
twist_pub = rospublisher('/cmd_vel','geometry_msgs/Twist');
% Create a 'geometry_msgs/Twist' message object
twist_msg = rosmessage('geometry_msgs/Twist');
% Set the values of the linear x, y, z fields of the message
twist_msg.Linear.X = 0;
twist_msg.Linear.Y = 0;

```

```
% Set the values of the angular x, y, z fields of the message
twist_msg.Angular.Z = 0;
% Publish the message on the topic
send(twist_pub,twist_msg);
pause(0.01);

function LEFT1_Callback(hObject, eventdata, handles)
twist_pub = rospublisher('/cmd_vel','geometry_msgs/Twist');
% Create a 'geometry_msgs/Twist' message object
twist_msg = rosmessage('geometry_msgs/Twist');
% Set the values of the linear x, y, z fields of the message
twist_msg.Linear.X = 0;
twist_msg.Linear.Y = 0;
% Set the values of the angular x, y, z fields of the message
twist_msg.Angular.Z = -1;
% Publish the message on the topic
send(twist_pub,twist_msg);
pause(0.01);
%-----
twist_pub = rospublisher('/cmd_vel','geometry_msgs/Twist');
% Create a 'geometry_msgs/Twist' message object
twist_msg = rosmessage('geometry_msgs/Twist');
% Set the values of the linear x, y, z fields of the message
twist_msg.Linear.X = 0;
twist_msg.Linear.Y = 0;
% Set the values of the angular x, y, z fields of the message
twist_msg.Angular.Z = 0;
% Publish the message on the topic
send(twist_pub,twist_msg);
pause(0.01);

function RIGHT1_Callback(hObject, eventdata, handles)
twist_pub = rospublisher('/cmd_vel','geometry_msgs/Twist');
% Create a 'geometry_msgs/Twist' message object
twist_msg = rosmessage('geometry_msgs/Twist');
% Set the values of the linear x, y, z fields of the message
twist_msg.Linear.X = 0;
twist_msg.Linear.Y = 0;
% Set the values of the angular x, y, z fields of the message
twist_msg.Angular.Z = 1;
% Publish the message on the topic
send(twist_pub,twist_msg);
pause(0.01);
%-----
twist_pub = rospublisher('/cmd_vel','geometry_msgs/Twist');
% Create a 'geometry_msgs/Twist' message object
twist_msg = rosmessage('geometry_msgs/Twist');
% Set the values of the linear x, y, z fields of the message
twist_msg.Linear.X = 0;
twist_msg.Linear.Y = 0;
% Set the values of the angular x, y, z fields of the message
twist_msg.Angular.Z = 0;
% Publish the message on the topic
send(twist_pub,twist_msg);
pause(0.01);

function DOWN1_Callback(hObject, eventdata, handles)
twist_pub = rospublisher('/cmd_vel','geometry_msgs/Twist');
% Create a 'geometry_msgs/Twist' message object
twist_msg = rosmessage('geometry_msgs/Twist');
```

```
% Set the values of the linear x, y, z fields of the message
twist_msg.Linear.X = -1;
twist_msg.Linear.Y = 0;
% Set the values of the angular x, y, z fields of the message
twist_msg.Angular.Z = 0;
% Publish the message on the topic
send(twist_pub,twist_msg);
pause(0.01);
%-----
twist_pub = rospublisher('/cmd_vel','geometry_msgs/Twist');
% Create a 'geometry_msgs/Twist' message object
twist_msg = rosmessage('geometry_msgs/Twist');
% Set the values of the linear x, y, z fields of the message
twist_msg.Linear.X = 0;
twist_msg.Linear.Y = 0;
% Set the values of the angular x, y, z fields of the message
twist_msg.Angular.Z = 0;
% Publish the message on the topic
send(twist_pub,twist_msg);
pause(0.01);

function STOP1_Callback(hObject, eventdata, handles)
twist_pub = rospublisher('/cmd_vel','geometry_msgs/Twist');
% Create a 'geometry_msgs/Twist' message object
twist_msg = rosmessage('geometry_msgs/Twist');
% Set the values of the linear x, y, z fields of the message
twist_msg.Linear.X = 0;
twist_msg.Linear.Y = 0;
% Set the values of the angular x, y, z fields of the message
twist_msg.Angular.Z = 0;
% Publish the message on the topic
send(twist_pub,twist_msg);
pause(0.01);

function AUTO1_Callback(hObject, eventdata, handles)
Run1_Callback();
set(handles.DRIVE1, 'Value', 0);
AutoRset = get(handles.AUTO1, 'Value');
if AutoRset == 1
%-----
pub = rospublisher('/move_base_simple/goal',
'geometry_msgs/PoseStamped');
goal_pose = rosmessage('geometry_msgs/PoseStamped');
goal_pose.Header.FrameId = 'map';
goal_pose.Pose.Position.X = 0;
goal_pose.Pose.Position.Y = 0;

goal_pose.Pose.Orientation.Z = 0;
goal_pose.Pose.Orientation.W = 0;
end

function Home1_Callback(hObject, eventdata, handles)
pub = rospublisher('/move_base_simple/goal',
'geometry_msgs/PoseStamped');
goal_pose = rosmessage('geometry_msgs/PoseStamped');
goal_pose.Header.FrameId = 'map';
goal_pose.Pose.Position.X = 0;
goal_pose.Pose.Position.Y = 0;

goal_pose.Pose.Orientation.Z = 0;
```

```

goal_pose.Pose.Orientation.W = 0;

function Run1_Callback(hObject, eventdata, handles)
filein= 'VT.csv';
subsetB = csvread(filein);
a=subsetB(1,1);
b=subsetB(1,2);
c=subsetB(2,1);
d=subsetB(2,2);
e=subsetB(3,1);
f=subsetB(3,2);
g=subsetB(4,1);
h=subsetB(4,2);
%-----
i=subsetB(5,1);
j=subsetB(5,2);
k=subsetB(6,1);
l=subsetB(6,2);
m=subsetB(7,1);
n=subsetB(7,2);
o=subsetB(8,1);
p=subsetB(8,2);
%-----

AT1 = get(handles.AUTO1, 'value');
%Lay vi tri can gui...
%-----
Blc1 = get(handles.BLC1, 'value');
Blc2 = get(handles.BLC2, 'value');
Blc3 = get(handles.BLC3, 'value');
Blc4 = get(handles.BLC4, 'value');
Oder5 = get(handles.OtherBlock, 'value');
%-----
if (Blc1==1) && (Blc2==0) && (Blc3==0) && (Blc4==0) && (Oder5==0) %Di chuyen den Blcck 1...
pub = rospublisher('/move_base_simple/goal',
'geometry_msgs/PoseStamped');
goal_pose = rosmessage('geometry_msgs/PoseStamped');
goal_pose.Header.FrameId = 'map';
goal_pose.Pose.Position.X = a;
goal_pose.Pose.Position.Y = b;

goal_pose.Pose.Orientation.Z = i;
goal_pose.Pose.Orientation.W = j;
% Xu?t b?n v? tri m?c ti?u
send(pub, goal_pose);
pause(0.05);
%-----
else if (Blc1==0) && (Blc2==1) && (Blc3==0) && (Blc4==0) && (Oder5==0)%Di chuyen den Block 2 ...
pub = rospublisher('/move_base_simple/goal',
'geometry_msgs/PoseStamped');
goal_pose = rosmessage('geometry_msgs/PoseStamped');
goal_pose.Header.FrameId = 'map';
goal_pose.Pose.Position.X = c;
goal_pose.Pose.Position.Y = d;

goal_pose.Pose.Orientation.Z = k;
goal_pose.Pose.Orientation.W = l;
% Xu?t b?n v? tri m?c ti?u

```

```

send(pub, goal_pose);
pause(0.05);
%-----
else if (Blc1==0) && (Blc2==0) && (Blc3==1) && (Blc4==0) &&
(Oder5==0)%Di chuyen den Block 3...
pub = rospublisher('/move_base_simple/goal',
'geometry_msgs/PoseStamped');
goal_pose = rosmessage('geometry_msgs/PoseStamped');
goal_pose.Header.FrameId = 'map';
goal_pose.Pose.Position.X = e;
goal_pose.Pose.Position.Y = f;

goal_pose.Pose.Orientation.Z = m;
goal_pose.Pose.Orientation.W = n;
% Xu?t b?n v? tri m?c ti?u
send(pub, goal_pose);
pause(0.05);
%-----
else if(Blc1==0) && (Blc2==0) && (Blc3==0) && (Blc4==1) &&
(Oder5==0)%Di chuyen den Block 4...
pub = rospublisher('/move_base_simple/goal',
'geometry_msgs/PoseStamped');
goal_pose = rosmessage('geometry_msgs/PoseStamped');
goal_pose.Header.FrameId = 'map';
goal_pose.Pose.Position.X = g;
goal_pose.Pose.Position.Y = h;

goal_pose.Pose.Orientation.Z = o;
goal_pose.Pose.Orientation.W = p;
% Xu?t b?n v? tri m?c ti?u
send(pub, goal_pose);
pause(0.05);
%-----
else (Blc1==0) && (Blc2==0) && (Blc3==0) && (Blc4==1) && (Oder5==1);%Di
chuyen den Vi tri khac...
pub = rospublisher('/move_base_simple/goal',
'geometry_msgs/PoseStamped');
goal_pose = rosmessage('geometry_msgs/PoseStamped');
goal_pose.Header.FrameId = 'map';
goal_pose.Pose.Position.X = str2num(get(handles.Xmsg,'string'));
goal_pose.Pose.Position.Y = str2num(get(handles.Ymsg,'string'));
%-----
goal_pose.Pose.Orientation.Z = str2num(get(handles.Zori,'string'));
goal_pose.Pose.Orientation.W = str2num(get(handles.Wori,'string'));

send(pub, goal_pose);
pause(0.05);
%-----
end
end
end
end

function LUP_Callback(hObject, eventdata, handles)
pub = rospublisher('/move_base/result',
'move_base_msgs/MoveBaseActionResult');
msg = rosmessage('move_base_msgs/MoveBaseActionResult');
msg.status.status = 3;
send(pub, msg);
pause(0.05);

```

```
function LDOWN_Callback(hObject, eventdata, handles)
pub = rospublisher('/move_base/result',
'move_base_msgs/MoveBaseActionResult');
msg = rosmessage('move_base_msgs/MoveBaseActionResult');
msg.status.status = 3;
send(pub, msg);
pause(0.05);

function SETTING_Callback(hObject, eventdata, handles)
Robot_System = questdlg('Do you want to switch to the robot settings
interface?', ...
'Warning', ...
'Yes','No','No');
% Handle response
switch Robot_System
case 'Yes'
close
run Settingfix.m
case 'No'
end

function CLOSE1_Callback(hObject, eventdata, handles)
Robot_System = questdlg('Do you want to exit program?', ...
'Warning', ...
'Yes','No','No');
% Handle response
switch Robot_System
case 'Yes'
close
case 'No'
end

function figure1_WindowKeyPressFcn(hObject, eventdata, handles)
key = eventdata.Key;
switch eventdata.Key
    case 'w'
        UP1_Callback();
    case 's'
        DOWN1_Callback();
    case 'a'
        LEFT1_Callback();
    case 'd'
        RIGHT1_Callback();
    case 'p'
        STOP1_Callback();
    case 'uparrow'
        LUP_Callback();
    case 'downarrow'
end
```

Code file luanch chạy Robot

```

<?xml version="1.0"?>
<launch>
    <master auto="start"/>
    <param name="/use_sim_time" value="false"/>
    <node name="rplidarNode"      pkg="rplidar_ros" type="rplidarNode" output="screen">
        <param name="serial_port"      type="string" value="/dev/ttyUSB0"/>
        <param name="serial_baudrate"   type="int"   value="1000000"/>
        <param name="frame_id"         type="string" value="laser"/>
        <param name="inverted"         type="bool"  value="false"/>
        <param name="angle_compensate"  type="bool"  value="true"/>
        <param name="scan_frequency"   type="double" value="10.0"/>
    </node>

    <!-- run serial connection -->
    <node      name="rosserial_python"      pkg="rosserial_python"      type="serial_node.py"
args="/dev/ttyACM0"/>
        <node pkg="laser_filters" type="scan_to_scan_filter_chain" name="laser_filter">
            <rosparam command="load" file="$(find hw_setup)/my_laser_config.yaml" />
            <remap from="scan" to="scan" />
        </node>

        <!--(Static) transformation for relation between laser frame and base_link frame -->
        <node pkg="tf" type="static_transform_publisher" name="base_link_to_laser" args="0.35 0.0 0.0
3.1415 0.0 0.0 /base_link /laser 40" />
        <node pkg="tf" type="static_transform_publisher" name="scanmatch_to_base" args="0.0 0.0 0.0 0.0
0.0 0.0 /scanmatcher_frame /base_link 40" />
        <node pkg="tf" type="static_transform_publisher" name="odom_to_scanmatch" args="0.0 0.0 0.0
0.0 0.0 0.0 /odom /scanmatcher_frame 40" />

    <!-- Start Gmapping node -->
    <!-- hector mapping -->
    <node pkg="hector_mapping" type="hector_mapping" name="hector_mapping" >
        <param name="scan_topic" value="scan_filtered" />
        <param name="base_frame" value="base_link" />
        <param name="odom_frame" value="odom"/>
        <param name="map_frame" value="map"/>
        <param name="output_timing" value="false"/>
        <param name="use_tf_scan_transformation" value="true"/>
        <param name="use_tf_pose_start_estimate" value="false"/>
    </node>

```

```

<param name="map_pub_period" value="1.0"/>
<param name="laser_max_dist" value = "30.0"/>
<param name="laser_min_dist" value = "0.05"/>
<param name="laser_z_min_value" value = "-0.3"/>
<param name="map_resolution" value="0.025"/>
<param name="map_size" value="1024"/>
<param name="map_start_x" value="0.5"/>
<param name="map_start_y" value="0.5"/>
<param name="map_multi_res_levels" value="3"/>
<param name="update_factor_free" value="0.4"/>
<param name="use_tf_pose_start_estimate" value="false"/>
<param name="pub_map_odom_transform" value="true"/>
<param name="pub_map_scanmatch_transform" value="true"/>
<remap from="map" to="map" />
<remap from="initialpose" to="hector_initial"/>
</node>-->

<!-- Original Move Base-->
<node pkg="move_base" type="move_base" respawn="false" name="move_base_node" output="screen">
    <rosparam file="$(find hw_setup)/launch/move_base/costmap_common_params.yaml" command="load" ns="global_costmap" />
    <rosparam file="$(find hw_setup)/launch/move_base/costmap_common_params.yaml" command="load" ns="local_costmap" />
    <rosparam file="$(find hw_setup)/launch/move_base/local_costmap_params.yaml" command="load" />
    <rosparam file="$(find hw_setup)/launch/move_base/global_costmap_params.yaml" command="load" />
    <rosparam file="$(find hw_setup)/launch/move_base/teb_local_planner_params.yaml" command="load" />
    <!-- <rosparam file="$(find hw_setup)/launch/move_base/base_test.yaml" command="load" />-->
</node>
</launch>

```

LỜI CẢM ƠN

Đầu tiên chúng em xin chân thành cảm ơn thầy Hoàng Đình Khôi đã tận tình hướng dẫn, giúp đỡ chúng em trong thời gian qua. Sự chỉ dẫn tận tình của thầy là nguồn động lực lớn để tiếp thêm ý chí giúp chúng em hoàn thành được quá trình làm đồ án và luận văn cho đề tài này.

Chúng em cũng xin chân thành cảm ơn đến trường Đại Học Công Nghiệp TPHCM và đặc biệt các quý thầy cô trong khoa điện, đã tạo điều kiện tốt nhất để chúng em tiếp thu nhiều kiến thức nền tảng, bổ ích trong khoảng thời gian học tập tại trường. Trang bị những kỹ năng cần thiết để chúng em có thể tự tin bắt tay hoàn thành đề tài này.

Do chưa có nhiều kinh nghiệm về làm đề tài, cũng như hạn chế về kiến thức, trong quá trình thuyết trình luận văn chắc chắn không tránh khỏi những thiếu sót. Chúng em rất mong nhận được sự nhận xét, ý kiến đóng góp, phê bình từ phía thầy cô để bài luận văn được hoàn thiện hơn.

Lời cuối cùng, chúng em xin chúc quý thầy cô, ban lãnh đạo các ban ngành của khoa điện và trường Đại Học Công Nghiệp TPHCM có thật nhiều sức khỏe, luôn giữ được ngọn lửa nhiệt huyết với nghề để có thể tiếp tục hành trình truyền đạt kiến thức, tạo nên những người kỹ sư tài giỏi cho đất nước của thế hệ sinh viên sau này.

Chúng em xin chân thành cảm ơn!

TP. Hồ Chí Minh, ngày 30, tháng 05, năm 2023

Sinh viên thực hiện

Phan Quốc Bửu

Phan Anh Hào

Nguyễn Văn Trung

Nguyễn Văn Dũng