

## MỤC LỤC

DANH MỤC HÌNH ẢNH.....	i
DANH MỤC BẢNG BIỂU.....	iii
DANH MỤC TỪ VIẾT TẮT.....	iv
MỞ ĐẦU .....	1
<b>CHƯƠNG 1: TỔNG QUAN VỀ HỆ ĐIỀU HÀNH ROBOT (ROS) .....</b>	<b>4</b>
<b>1.1 Một số phần mềm nghiên cứu phát triển robot di động điển hình .....</b>	<b>4</b>
1.1.1 Microsoft Robotics Developer Studio .....	4
1.1.2 Mobile Robot Programming Toolkit .....	5
<b>1.2 Tổng quan về hệ điều hành robot.....</b>	<b>6</b>
<b>1.3 Các gói ứng dụng cần thiết cho bài toán định vị, định hướng, lập kế hoạch cho robot di động.....</b>	<b>16</b>
1.3.1 Gói ứng dụng lập kế hoạch di chuyển với ROS.....	16
1.3.2 Gói ứng dụng định vị trên bản đồ có sẵn AMCL .....	23
1.3.3 Gói truyền thông giữa ROS và Arduino roserial_python .....	23
1.3.4 Gói ứng dụng định vị xây dựng bản đồ SLAM_GMAPPING .....	23
1.3.5 Gói ứng dụng cho điều khiển camera Kinect.....	25
<b>CHƯƠNG 2: CƠ SỞ LÝ THUYẾT ĐỊNH VỊ, ĐỊNH HƯỚNG, LẬP KẾ HOẠCH DI CHUYỂN CHO ROBOT DI ĐỘNG .....</b>	<b>27</b>
<b>2.1 Mô hình động học của robot di động.....</b>	<b>27</b>
<b>2.2 Thuật toán định vị.....</b>	<b>30</b>
2.2.1 Phương pháp định vị GPS .....	30
2.2.2 Phương pháp định vị Monte Carlo thích nghi.....	31
<b>2.3 Thuật toán lập kế hoạch di chuyển cho robot.....</b>	<b>42</b>
2.3.1 Thuật toán Dijkstra.....	42
2.3.2 Hệ thống dẫn đường cột mốc chủ động .....	43
2.3.3 Định hướng sử dụng bản đồ .....	44
<b>CHƯƠNG 3: THIẾT KẾ CƠ KHÍ VÀ TÍCH HỢP PHẦN ĐIỆN – ĐIỆN TỬ ROBOT DI ĐỘNG .....</b>	<b>46</b>

<b>3.1 Các yêu cầu về mô hình Robot.....</b>	<b>46</b>
3.1.1 Yêu cầu về cơ khí.....	46
3.1.2 Yêu cầu về phần điện-điện tử .....	46
3.1.3 Yêu cầu về hệ thống cảm biến.....	46
<b>3.2 Lựa chọn mô hình.....</b>	<b>47</b>
3.2.1 Lựa chọn mô hình cơ khí.....	47
3.2.2 Lựa chọn thiết bị cho mô hình robot.....	49
<b>3.3 Thiết kế Robot .....</b>	<b>60</b>
3.3.1 Thiết kế cơ khí cho mô hình Robot .....	60
3.3.2 Lắp ráp mô hình robot.....	65
<b>3.4 Sơ đồ hệ thống mô hình Robot.....</b>	<b>68</b>
3.4.1 Sơ đồ cấu trúc hệ thống mô hình Robot.....	68
3.4.2 Sơ đồ điều khiển.....	69
<b>CHƯƠNG 4: THỰC NGHIỆM VÀ ĐÁNH GIÁ .....</b>	<b>71</b>
<b>4.1 Cài đặt hệ điều hành Ubuntu.....</b>	<b>71</b>
<b>4.2 Xây dựng bản đồ tĩnh.....</b>	<b>71</b>
4.2.1 Công tác chuẩn bị.....	71
4.2.2 Tiến hành thực nghiệm.....	71
4.2.3 Đánh giá thực nghiệm .....	72
<b>4.3 Thực nghiệm khả năng định vị và điều hướng với gói ứng dụng     Navigation.....</b>	<b>73</b>
4.3.1 Công tác chuẩn bị.....	73
4.3.2 Tiến hành thực nghiệm.....	73
4.3.3 Đánh giá thực nghiệm .....	75
<b>KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN .....</b>	<b>76</b>
<b>TÀI LIỆU THAM KHẢO .....</b>	<b>78</b>
<b>Phụ lục 1: BẢN VẼ CHI TIẾT CƠ KHÍ .....</b>	<b>79</b>
<b>Phụ lục 2: CHƯƠNG TRÌNH ĐIỀU KHIỂN .....</b>	<b>80</b>

## DANH MỤC HÌNH ẢNH

Hình 1.1. Mô hình robot di động được mô phỏng trên MRDS.....	4
Hình 1.2. Cộng đồng các trung tâm tham gia phát triển cho ROS trên thế giới...	7
Hình 1.3. So sánh khối lượng công việc phải làm khi dùng và không dùng ROS	7
Hình 1.4. Mối quan hệ giữa Stack và các gói.....	9
Hình 1.5. Mối quan hệ giữa Ngăn xếp, Gói và các File mô tả theo dạng thư mục .....	10
Hình 1.6. Mô tả cơ chế quản lý thông số trên Master.....	11
Hình 1.7. Mô tả hoạt động của dịch vụ .....	12
Hình 1.8. Mô hình giao tiếp cơ bản trong ROS.....	13
Hình 1.9. ROS tổng thể và tài nguyên trong ROS.....	14
Hình 1.10. Các hệ tọa độ của robot và chuyển động trong không gian .....	15
Hình 1.11. Giao tiếp giữa action client và server .....	16
Hình 1.12. Biểu đồ tính toán của Navigation Stack .....	17
Hình 1.13. Costmap function.....	19
Hình 1.14. Trạng thái phục hồi của move_base .....	22
Hình 1.15. Sơ đồ sử dụng gói ứng dụng SLAM - Gmapping.....	24
Hình 1.16. Mô hình URDF của robot .....	26
Hình 2.1. Mô hình động học robot.....	27
Hình 2.2. Minh họa về quá trình định vị theo phương pháp xác suất, phân bố xác suất được biểu diễn ở dạng rời rạc.....	33
Hình 2.3. Minh họa mô hình xác suất của phương trình động học .....	34
Hình 2.4. Mô hình đo lường của robot trong khoảng thời gian $(t-1, t]$ . ....	35
Hình 2.5. Mô hình theo đo lường với các thiết lập thông số nhiều khác nhau. .	36
Hình 2.6. Phân bố xác suất của biến ngẫu nhiên $a$ , phương sai $b$ , kì vọng bằng 0 .....	37
Hình 2.7. Mô hình xác suất của phép đo từ cảm biến khoảng cách chùm tia ....	40
Hình 2.8. Minh họa trường hợp kidnapping - tại thời điểm thứ hai khi robot dự đoán những vị trí ở cả bốn góc của hành lang.....	41

Hình 2.9. Định vị sử dụng vật mốc .....	44
Hình 2.10. Định vị và dẫn đường cho Robot chuyển động sử dụng bản đồ.....	45
Hình 3.1. Bánh dẫn động của robot .....	50
Hình 3.2. Bánh bi cầu.....	51
Hình 3.3. Tổng hợp các lực tác dụng lên xe.....	52
Hình 3.4. Động cơ DC servo .....	54
Hình 3.5. Camera Kinect Xbox 360.....	55
Hình 3.6. Những thành phần chính của Kinect .....	55
Hình 3.7. Vùng nhìn thấy của camera khoảng cách Kinect.....	56
Hình 3.8. Mạch cầu H và sơ đồ kết nối với vi điều khiển .....	57
Hình 3.9. Bộ điều khiển Arduino mega 2560.....	58
Hình 3.10. Sơ đồ mắc pin vào mạch cân bằng sạc xả.....	59
Hình 3.11. Máy tính nhúng Raspberry.....	60
Hình 3.12. Tấm đế dưới.....	61
Hình 3.13. Tấm ốp mặt sau .....	62
Hình 3.14. Tấm ốp mặt trước .....	62
Hình 3.15. Tấm ốp mặt trên.....	63
Hình 3.16. Gá động cơ .....	64
Hình 3.17. Khung Robot .....	65
Hình 3.18. Mô hình 3D bố trí thiết bị điện – điện tử trong mô hình.....	66
Hình 3.19. Mô hình 3D phần gá camera và camera .....	66
Hình 3.20. Mô hình robot đã lắp ráp hoàn thiện trên Solidwords.....	67
Hình 3.21. Mô hình robot thực tế .....	67
Hình 3.22. Sơ đồ cấu trúc hệ thống điều khiển Robot.....	68
Hình 3.23. Sơ đồ điều khiển mô hình Robot.....	70
Hình 4.1. Bản đồ 2D thu được và hình ảnh địa hình thực tế .....	72
Hình 4.2. Kế hoạch di chuyển với vật cản cố định.....	74
Hình 4.3. Phát hiện vật cản chưa biết .....	75

## DANH MỤC BẢNG BIỂU

Bảng 1.1. So sánh một số phần mềm điều khiển robot.....	5
Bảng 2.1. Giải thuật motion_model_odometry .....	35
Bảng 2.2. Giải thuật prob_normal_distribution và giải thuật prob_triangular_distribution.....	37
Bảng 2.3. Giải thuật sample_normal_distribution và giải thuật sample_triangular_distribution.....	38
Bảng 2.4. Giải thuật sample_motion_model_đo lường.....	38
Bảng 2.5. Giải thuật beam_range_finder_model.....	40
Bảng 2.6. Giải thuật Augmented_MCL .....	41
Bảng 3.1. Dạng robot tự hành dạng 2 bánh xe, trích trang 18 [15].....	47
Bảng 3.2. Các dạng robot di động ba bánh xe, trích trang 19 [15] .....	47
Bảng 3.3. Một số dạng robot di động bốn bánh xe, trích trang 20 [15] .....	48
Bảng 3.4. Thông số kỹ thuật của động cơ DC servo Faulhaber-12V.....	54
Bảng 3.5. Thông số kỹ thuật của Raspberry Pi Model B+ .....	59

**DANH MỤC TỪ VIẾT TẮT**

MRDS	Microsoft Robotics Developer Studio
MRPT	Mobile Robot Programming toolkit
ROS	Robot Operation System
GPS	Global Positioning System
RPC	Remote Procedure Calls
MCU	Micro Controller Unit
EKF	Extended Kalman Filter
ER1	Evolution Robotics 1
PCL	Point Cloud Library
API	Application Programming Interface
PWM	Pulse Width Modulation
RGB	Red Green Black
VGA	Video Graphics Array
UART	Universal Asynchronous Receiver – Transmitter
IMU	Inertial Measurement Unit
UDP	User Datagram Protocol
GPIO	General Purpose Input Output

## MỞ ĐẦU

Việc ứng dụng các loại Robot khác nhau nhằm phục vụ con người trong sản xuất và sinh hoạt đang ngày càng thu hút sự quan tâm nghiên cứu của nhiều nước trên thế giới. Bên cạnh việc tăng năng suất lao động, tăng độ chính xác và ổn định chất lượng của sản xuất, robot còn đóng vai trò quan trọng trong việc thay thế con người làm việc nhà... Hơn nữa robot còn làm việc ở những môi trường mà con người không thể thực hiện hoặc có thể nguy hại đến sức khỏe của con người như: môi trường độc hại, cách li, ô nhiễm, phóng xạ hạt nhân... Trước tính hình đó, việc nghiên cứu chế tạo các mẫu robot có khả năng quan sát và có khả năng di chuyển linh hoạt trong nhà, leo cầu thang, có cơ cấu thao tác có thể giúp con người thực hiện một số công việc nào đó là vấn đề hết sức cấp thiết.

Robot tự hành là một robot có khả năng tự di chuyển, tự vận động (có thể lập trình lại) dưới sự điều khiển tự động để thực hiện tốt những công việc được giao. Môi trường hoạt động của robot có thể là mặt đất, nước, không khí, không gian hay là tổ hợp các môi trường đó. Các mô hình robot di động khác nhau được nghiên cứu theo môi trường mà robot hoạt động, ví dụ như xe tự hành trên mặt đất AGV (Autonomous Guided Vehicles), robot tự hành dưới nước AUV (Autonomous Underwater Vehicles), robot tự hành trên không UAV (Unmanned Aerial Vehicles), robot vũ trụ (Space Robot). Những khả năng cần có của một robot tự hành là dự trữ năng lượng đảm bảo cho robot tự hành trong thời gian yêu cầu, khả năng phát hiện và nhận biết vật cản, tránh va chạm với các vật thể và khả năng xử lý, tính toán các chương trình giải thuật phức tạp trong thời gian thực để đáp ứng kịp thời với những tác động của môi trường .v.v.

Trong lĩnh vực robot phục vụ con người, robot không chỉ di chuyển mà còn mang các vật dụng, tránh được vật cản (cố định, di chuyển được...), nhận biết được sự thay đổi của môi trường hoạt động, quyết định và vận hành theo các tác vụ đã được hoạch định trước.

Từ những yêu cầu cấp thiết đã nêu trên, dưới sự hướng dẫn của thầy giáo Hoàng Quang Chính, thầy giáo Hà Huy Hưng nhóm chúng tôi đã thực hiện đồ

án tốt nghiệp “ *Nghiên cứu ứng dụng hệ điều robot (ROS) trong điều khiển robot di động*”.

Mục đích của đề án là thiết kế một robot di động, sử dụng camera khoảng cách Kinect và các cảm biến khác cùng với thuật toán định hướng trên ROS để xây dựng bản đồ bằng phẳng, tự di chuyển trong bản đồ đó tới một vị trí cho trước, đồng thời xử lý tránh vật cản chưa biết. Mục đích phát triển sâu hơn là một Robot có thể thực hiện nhiều tác vụ nhằm giảm tải công việc cho con người như quét dọn, hút bụi, chuyển đồ... trong phạm vi nhà ở hoặc văn phòng, bệnh viện.

Với phạm vi nghiên cứu là triển khai thuật toán định hướng trên ROS và các ứng dụng ROS hỗ trợ việc định hướng và điều hướng cho robot di động trong nhà với sự hỗ trợ thu thập dữ liệu từ Kinect (thu thập dữ liệu và vẽ bản đồ cho robot định hướng) và cảm biến khác như encoder..., để Robot tự thực hiện được các chức năng theo yêu cầu.

Nội dung của đề án được bố cục như sau:

Phần mở đầu giới thiệu về đề án, đặt vấn đề và tính cấp thiết của đề án, mục đích nghiên cứu, phạm vi và bố cục của đề án.

- Chương 1: Tổng quan về hệ điều hành robot: tổng quan một số phần mềm điều khiển robot đi động điển hình; tổng quan về hệ điều hành robot (ROS); các gói ứng dụng cần thiết cho bài toán định vị, định hướng và lập kế hoạch cho robot tự hành.

- Chương 2: Cơ sở lý thuyết định vị, định hướng, lập kế hoạch di chuyển cho Robot di động: mô hình động học của Robot di động; các thuật toán định vị, định hướng, kế hoạch di chuyển cho Robot trên cơ sở các thông tin sẵn có phản hồi về từ Robot.

- Chương 3: Thiết kế cơ khí và tích hợp phần điện – điện tử cho Robot tự hành: các yêu cầu về mô hình cho Robot; lựa chọn mô hình và thiết bị; thiết kế mô hình cho Robot.



- Chương 4: Thử nghiệm và đánh giá kết quả: xây dựng sơ đồ hệ thống của mô hình; xây dựng bản đồ tĩnh, xây dựng bản đồ có vật cản chưa biết và hoạt động trong các bản đồ đó; thực nghiệm và đánh giá những kết quả đạt được.

## CHƯƠNG 1: TỔNG QUAN VỀ HỆ ĐIỀU HÀNH ROBOT (ROS)

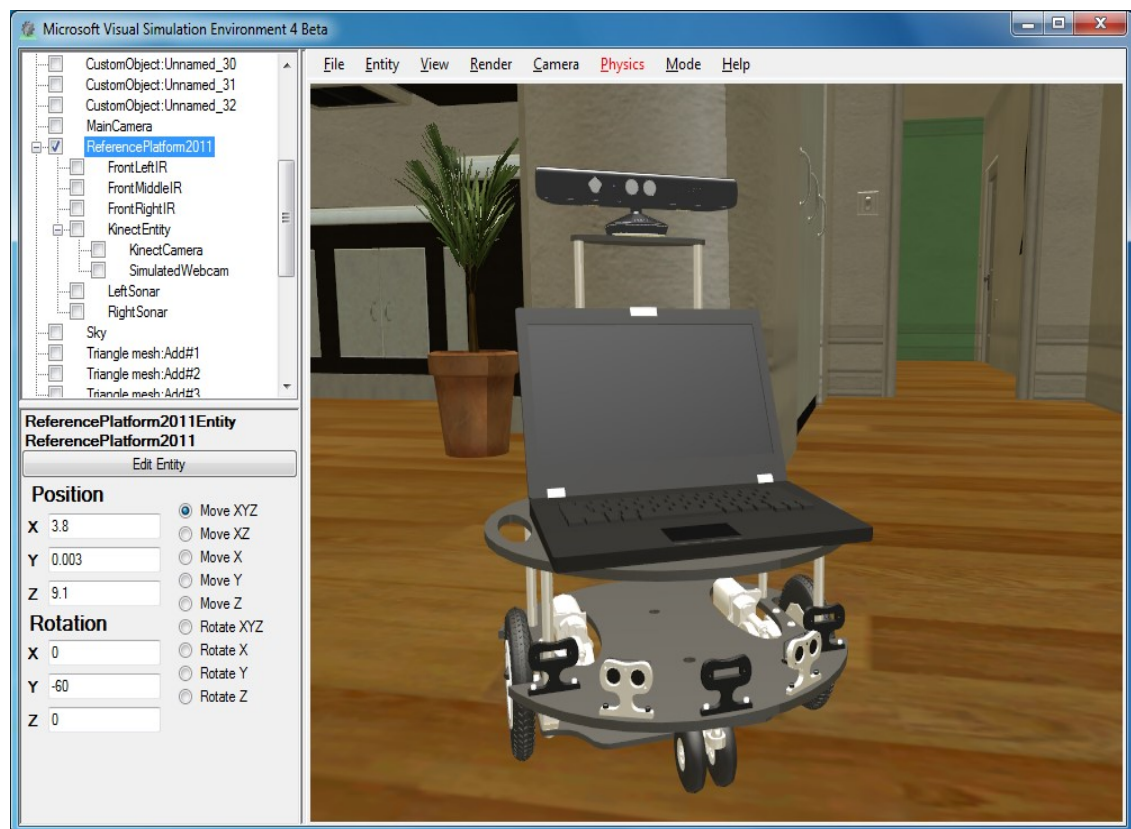
### 1.1 Một số phần mềm nghiên cứu phát triển robot di động điển hình

#### 1.1.1 Microsoft Robotics Developer Studio

*Microsoft Robotics Developer Studio* (Microsoft RDS, MRDS) là một phần mềm mã nguồn mở được thiết kế và phát triển bởi Microsoft. Đây là một môi trường dựa trên Windows để kiểm soát, điều khiển và mô phỏng robot.

Các tính năng bao gồm: một công cụ lập trình trực quan, Microsoft Visual Programming Language để tạo và gỡ lỗi các ứng dụng robot, các giao diện web và dựa trên cửa sổ, mô phỏng 3D (bao gồm tăng tốc phần cứng), dễ dàng truy cập vào cảm biến và bộ điều khiển robot. Ngôn ngữ lập trình chính là C#[1].

Ứng dụng thành công của MRDS:



Hình 1.1. Mô hình robot di động được mô phỏng trên MRDS

Nhưng đến năm 2014 trong cuộc tái cơ cấu của Microsoft MRDS không được phát triển tiếp do những mặt hạn chế và tính kinh tế không được cao và không cạnh tranh được với nhiều phần mềm thiết kế và phát triển robot khác.

Những hạn chế của *Microsoft Robotic Developer Studio*:

- *Microsoft robotic Developer Studio* là phần mềm mã nguồn mở nhưng tính cộng đồng rất hạn chế.
- Ngôn ngữ lập trình hạn chế, sử dụng duy nhất ngôn ngữ C#;
- Chỉ có một số ứng dụng với các phần cứng của Microsoft được xây dựng nên ứng dụng đối với MRDS rất hạn chế.

### 1.1.2 Mobile Robot Programming Toolkit

*Mobile Robot Programming Toolkit (MRPT)* là một cross-platform và mã nguồn mở C++ nhằm giúp các nhà nghiên cứu robot thiết kế và thực hiện các thuật toán liên quan đồng thời đến cả vị trí và lập bản đồ (SLAM), thị giác máy và kế hoạch di chuyển (tránh vật cản). Là một phần mềm để lập trình điều khiển được xây dựng riêng cho robot di động được phát triển khá đầy đủ các gói ứng dụng riêng cho robot di động như kế hoạch di chuyển, bản đồ, định hướng, định vị và hiển thị mô phỏng[2].

Những hạn chế của MRPT:

- Nguồn tài nguyên còn hạn chế;
- Số lượng người phát triển ít khi cần giải quyết các vấn đề khá khó khăn cho việc hỏi đáp trên diễn đàn;
- Ngôn ngữ lập trình không đa dạng, chỉ sử dụng duy nhất một ngôn ngữ C++.

*Bảng 1.1. So sánh một số phần mềm điều khiển robot*

	MRDS	MRPT	ROS
Nhà phát triển	Microsoft liên kết với cộng đồng	Nhóm MAPIR, Jose, Luis Blaco	Cộng đồng ROS
Phiên bản	4.0/ ngày 8 tháng 3 năm 2012	1.4/ ngày 22 tháng 4 năm 2016	ROS lunar là phiên bản thứ 7 phát hành tháng 5 năm 2017
Ngôn ngữ viết	C#	C++	C++, Python
Hệ điều hành	Windows 7, Windows XP...	Linux, Windows	Linux, Debian, OS X, OpenEmbedded

Hỗ trợ phần cứng	Hạn chế	Hạn chế	Khá đầy đủ
Xu hướng phát triển	Microsoft dùng phát triển	Ít phát triển	Đang phát triển bởi cộng đồng người dùng lớn
Trang truy cập	<a href="http://www.microsoft.com/robotics">www.microsoft.com/robotics</a>	<a href="http://www.mrpt.org">www.mrpt.org</a>	<a href="http://www.wiki.ros.org">www.wiki.ros.org</a>

## 1.2 Tổng quan về hệ điều hành robot

ROS[3] là một hệ điều hành mã nguồn mở, dùng cho các ứng dụng trên robot. Về cơ bản, ROS cũng có những khả năng cần thiết cho một hệ điều hành cơ bản như khả năng thực hiện các tác vụ (stack) song song, giao tiếp, trao đổi dữ liệu giữa các tác vụ bằng thông điệp và quản lý dữ liệu... Bên cạnh đó, để có thể ứng dụng trong lĩnh vực robotic, ROS đã phát triển các thư viện và công cụ chuyên biệt dành cho việc thu thập dữ liệu, xử lý, hiển thị và điều khiển... Ngoài ra, ROS còn có thể tương tác và kết hợp với nhiều ứng dụng khác như Player (một công cụ phần mềm cho robot và các ứng dụng về cảm biến, Orocos (điều khiển thông minh trong robot và tự động hóa, Carmen (Robot Navigation Toolkit), Orca (các đối tượng cho robot)...

ROS có nhiều khái niệm dựa trên đồ thị (graph), biểu diễn mối quan hệ giữa các thành phần trong hệ điều hành như ngăn xếp (stack), gói (package), chủ đề (topic), gói tin (message), dịch vụ (service)... cũng như các khái niệm hệ tọa độ và phép chuyển đổi hệ tọa độ.

Về khía cạnh trao đổi dữ liệu và giao tiếp, ROS có tích hợp sẵn một vài chuẩn giao tiếp đồng bộ theo chuẩn RPC (truyền thông tin giữa client và server) qua các dịch vụ. Truyền dữ liệu bất đồng bộ qua các chủ đề và lưu trữ dữ liệu trên tham số máy chủ (Parameter Server).

Hệ điều hành ROS cùng với các công cụ và thư viện hỗ trợ thường phát hành dưới dạng ROS phân phối (ROS Distribution), tương tự Linux Distribution, cung cấp những gói phần mềm ổn định để người dùng sử dụng và phát triển thêm. ROS là một hệ điều hành mã nguồn mở, cho nên thu hút được

sự quan tâm và đóng góp của cộng đồng trên khắp thế giới để xây dựng phát triển các dự án Robot, các công cụ và thư viện kèm theo. Có rất nhiều mô hình robot đã được xây dựng thành công với hệ điều hành ROS.

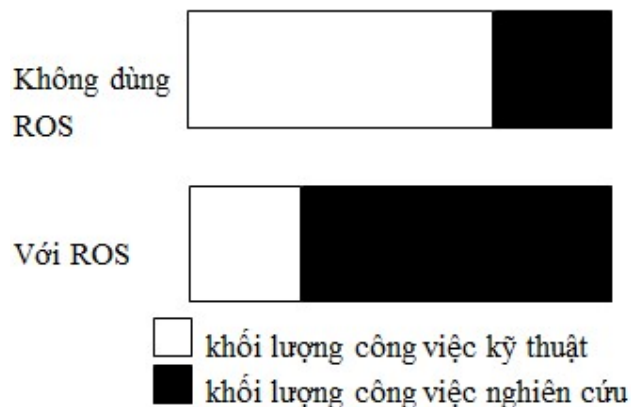
Hiện nay ROS chỉ chạy trên hệ điều hành Ubuntu và Mac OS X. Cộng đồng ROS đang thử nghiệm hỗ trợ cho các nền tảng như Windows, ARM Linux, Fedora và Gentoo.



Hình 1.2. Cộng đồng các trung tâm tham gia phát triển cho ROS trên thế giới

- Ưu điểm của ROS

Xây dựng ứng dụng robotics trên nền tảng ROS sẽ giảm đi một lượng đáng kể các công việc lập trình, thiết lập hệ thống, tận dụng nguồn tài nguyên mã nguồn mở vô cùng phong phú của cộng đồng. Ta có thể so sánh khối lượng công việc kỹ thuật cơ bản (required engineering) và khối lượng nghiên cứu khoa học (Core Research) như sau:



Hình 1.3. So sánh khối lượng công việc phải làm khi dùng và không dùng ROS

Từ đó ta thấy rằng, với sự hiệu quả từ ROS thời gian dành cho các công việc kỹ thuật cơ bản sẽ được giảm xuống rất đáng kể và do đó, tăng thời gian cho công việc nghiên cứu chuyên sâu, hàm lượng khoa học đạt được trong nghiên cứu sẽ lớn hơn nhiều lần.

Một số đặc điểm giúp cho ROS trở thành một hệ điều hành nên được sử dụng khi nghiên cứu phát triển một ứng dụng robot là:

- ROS là một hệ điều hành mã nguồn mở;
- Các tài liệu kỹ thuật, tài liệu hướng dẫn và các kênh hỗ trợ đầy đủ;
- Vấn đề cốt lõi nhất khiến ROS trở nên mạnh đó là tính cộng đồng rất lớn;
- Nguồn tài nguyên được cộng đồng đóng góp hầu như được xây dựng, phát triển từ những viện nghiên cứu và những trường đại học hàng đầu trên thế giới.

Những tài nguyên được cung cấp từ ROS thể hiện được sức mạnh trong lĩnh vực robotics như là:

- Đồ họa (Visualization);
- Nhận diện vật thể (Object recognition);
- Định hướng di động (Navigation);
- Thao tác/cầm nắm (Manipulation/Grasping).

ROS có 3 cấp khái niệm: Hệ thống tập tin (Filesystem), Biểu đồ tính toán (Computation Graph) và Tính cộng đồng (Community). Ngoài ra, ROS còn có một số khái niệm cấp cao đặc trưng cho các ứng dụng robot như hệ tọa độ, phép chuyển đổi thông tin mô tả...

### • Hệ thống tập tin trong ROS (Filesystem)

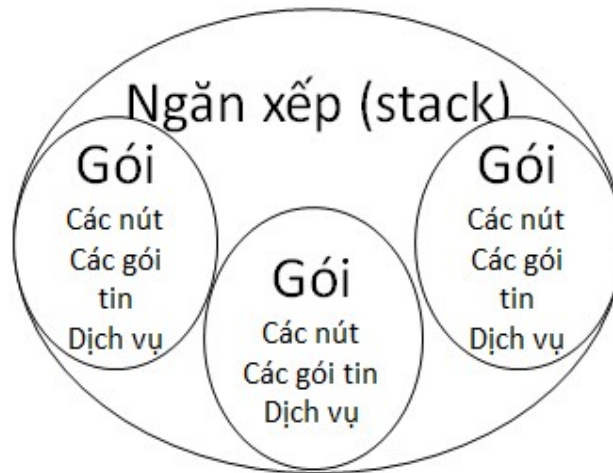
Hệ thống tập tin là mã nguồn tài nguyên ROS được lưu trữ trên bộ nhớ hệ thống, nó bao gồm các khái niệm:

Gói: gói ứng dụng là đơn vị cơ bản trong tổ chức phần mềm của ROS, một gói chứa source code cho một tác vụ thực thi một chức năng đặc thù, danh mục các mã nguồn kế thừa (dependency), là các hệ thống tập tin ngang cấp được

dựa trên đơn vị này, các tệp cấu hình (như tệp CMakeList.txt chứa các lệnh hướng dẫn biên dịch, yêu cầu tạo tệp thực thi .bin, hay chọn phiên bản cho các thư viện)...

Tệp kê khai (Manifest): là tệp kê khai thông tin mô tả gói (*manifest.xml*) cung cấp các cơ sở dữ liệu về gói đó, bao gồm điều kiện thực thi (license) và các sự phụ thuộc (dependency) của gói đó. Ngoài ra, tệp còn chứa những thông tin về đặc trưng của ngôn ngữ lập trình như cờ báo (flags) của trình biên dịch.

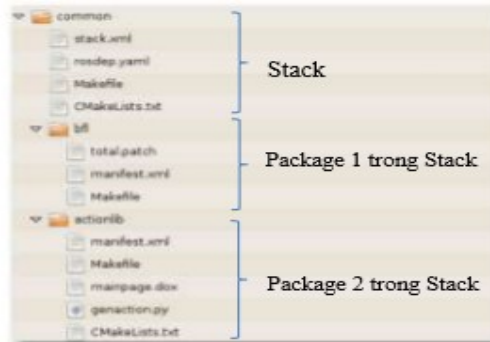
Ngăn xếp (Stacks): là tập hợp các gói phối hợp với nhau để thực hiện một chức năng cụ thể. Ngăn xếp còn mô tả cách thức biên dịch ROS và thông tin về phiên bản ROS tương thích (gọi là *distro*, ví dụ như các phiên bản ROS *hydro*, *groovy* hay *fuerte*). Ví dụ *pcl\_ros* là một ngăn xếp chứa các gói có chức năng cung cấp các bộ lọc cho điểm đám mây (point cloud) như bộ lọc PassThrough (giới hạn điểm đám mây theo các chiều x, y, z), VoxelGrid (lượng tử hóa point cloud thành các ô 3 chiều), Inlier/Outlier (tìm các điểm trong/ngoài một mặt theo các thông số của mặt đó trong điểm đám mây).



Hình 1.4. Mối quan hệ giữa Stack và các gói

Tệp kê khai ngăn xếp (Stack Manifest) (.xml): cung cấp mô tả cơ sở dữ liệu về một ngăn xếp, bao gồm điều kiện cho phép (license) và thông tin về các ngăn xếp phụ thuộc khác.

Phần phụ thuộc (Dependency): là mô tả trong tệp của một gói hay ngăn xếp về các hệ thống tệp tin ngang cấp (ngăn xếp hoặc gói khác) mà ngăn xếp hay gói kế thừa.



Hình 1.5. Mối quan hệ giữa Ngăn xếp, Gói và các File mô tả theo dạng thư mục

Gói tin (Message) (.msg): thông tin mô tả gói, được lưu trữ trong tệp có dạng `my_pack/msg/MyMessageType.msg`, định nghĩa các cấu trúc dữ liệu của gói tin được gửi trong ROS. Ví dụ có mô tả của gói tin `PointCloud2` được mô tả trong file `sensor_msgs/PointCloud2.msg`.

Dịch vụ (Service) (.srv): thông tin mô tả các dịch vụ được lưu trữ trong `my_package/srv/MyServiceType.srv`, định nghĩa cấu trúc dữ liệu cho các lệnh truy cập (request) và các lệnh phản hồi (response) của các dịch vụ trong ROS.

Thực thi (Launch): là các tệp `.launch` dùng để khởi tạo một tập hợp các nút cùng lúc, đồng thời cho các giá trị tham số và gán các chủ đề liên kết các nút liên kết các nút bằng lệnh `roslaunch` trong Command Terminal của Linux.

- **Biểu đồ tính toán trong ROS**

Computation Graph, tạm gọi là biểu đồ tính toán, là mạng ngang hàng (peer-to-peer) các tác vụ khi thực thi của ROS, trong đó các dữ liệu được trao đổi và xử lý giữa các nút. Các khái niệm cơ bản của biểu đồ tính toán của ROS là các nút, máy chủ, tham số máy chủ, dịch vụ, gói tin, các chủ đề và các túi.

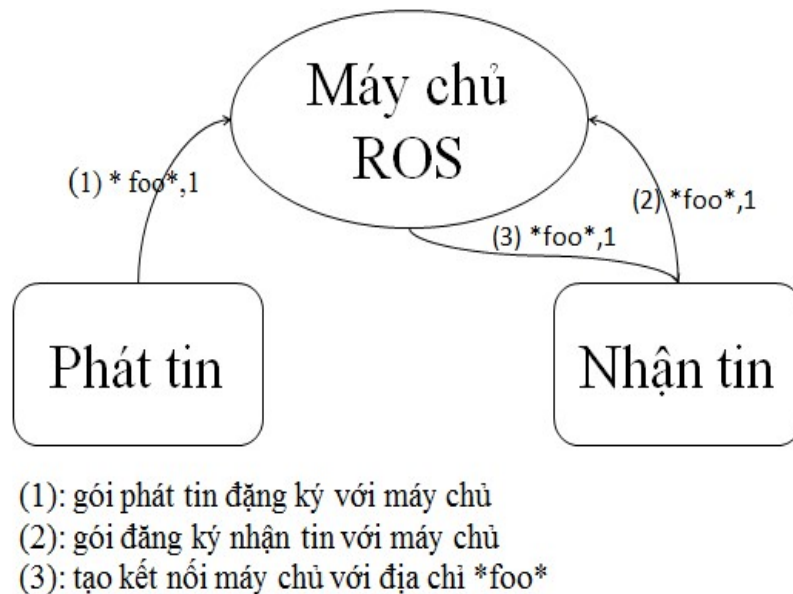
Nút: là đơn vị thực hiện một tác vụ tính toán, điều khiển. Một nút có thể được khởi tạo khi biên dịch thành công một gói và có thể được khởi tạo nhiều nút từ cùng một gói. Một hệ thống điều khiển thường bao gồm nhiều nút. Ví dụ



một nút điều khiển động cơ, một nút thực hiện tác vụ định vị, một nút vẽ quỹ đạo đường đi.

Máy chủ: ROS Master cung cấp tên đăng ký và tra cứu đến phần còn lại của biểu đồ tính toán. Nếu không có Master, các nút sẽ không tìm thấy nhau để trao đổi thông tin hay gọi các dịch vụ.

Tham số máy chủ: là một phần của Master, cho phép dữ liệu được lưu trữ trong một vị trí trung tâm và cho phép các nút truy cập tới.



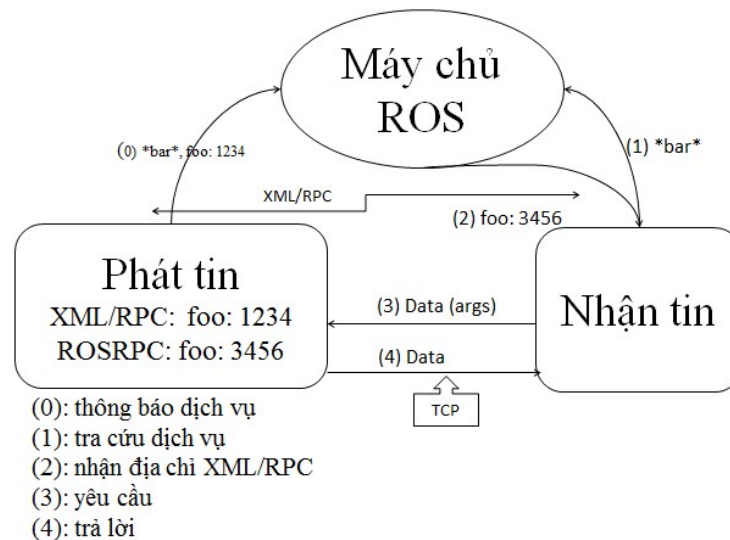
*Hình 1.6. Mô tả cơ chế quản lý thông số trên Master*

Gói tin: các nút giao tiếp với nhau thông qua các gói tin. Một gói tin trong biểu đồ tính toán là dữ liệu cụ thể có cấu trúc như trong khai báo của tệp .msg tương ứng. Các kiểu dữ liệu chuẩn như integer, floating, point, Boolean... và mảng với kiểu chuẩn đều được hỗ trợ. Bên cạnh đó, gói tin cũng có thể bao gồm các cấu trúc và các mảng lồng nhau như cấu trúc trong ngôn ngữ C. Các nút khi nhận gói tin cần phải xác định qua đệm để xử lý dữ liệu nhận được.

Chủ đề: gói tin được lọc thông qua một hệ thống vận chuyển, trong đó phân loại thành hai công việc chính: phát (publish) và nhận (subscribe). Một nút gửi đi một gói bằng việc đưa thông tin tới một chủ đề. Một chủ đề có tên và kiểu gói tin xác định. Một nút chỉ nhận đến đúng chủ đề có tên và kiểu dữ liệu như đã khai báo. Một chủ đề có thể có nhiều đối tượng đưa tin và cũng có thể có nhiều

đối tượng đăng ký nhận tin. Mỗi nút cũng có thể truyền tin trên nhiều chủ đề khác nhau, cũng như có thể nhận tin từ nhiều chủ đề khác nhau. Các nguồn truyền tin và các đối tượng nhận tin nhìn chung không cần phải biết về sự tồn tại của nhau. Ý tưởng xây dựng ROS ở đây là tách biệt nguồn tạo ra thông tin với bộ phận sử dụng thông tin đó. Chủ đề được xem như là một kênh truyền các thông điệp được định kiểu. Mỗi kênh truyền này có một tên riêng, và nút cũng có thể kết nối với kênh này để gửi/nhận thông tin, miễn là thông tin cùng kiểu với chủ đề đó.

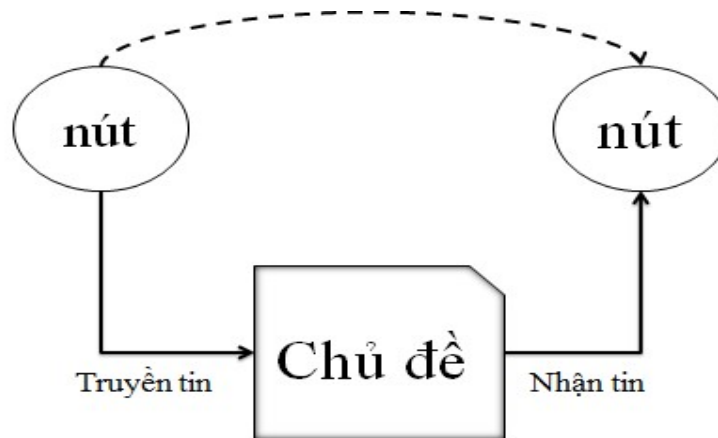
Dịch vụ: mô hình truyền thông theo mẫu publish/subscribe như trình bày ở trên là một mô hình rất linh hoạt, tuy vậy, đặc điểm của nó là thông tin được truyền đa đối tượng, thông tin một chiều (many-to-many, one-way) đôi khi lại không phù hợp với các trường hợp cần tương tác theo kiểu yêu cầu/trả lời (request/reply), kiểu tương tác này thường gặp trong các hệ thống phân phối. Do vậy, cần có thêm một thành phần nữa trong ROS Graph, đó là gói dịch vụ (service), nhằm thực hiện được các yêu cầu và một thông điệp dành cho đáp ứng. Một nút cung ứng một dịch vụ (service) với một thuộc tính tên (name), một khách hàng (client) sử dụng dịch vụ đó bằng cách gửi đi một thông tin yêu cầu rồi đợi phản hồi. Trong thư viện client của ROS, phương pháp tương tác này thường được cung cấp như một hàm được gọi từ xa.



Hình 1.7. Mô tả hoạt động của dịch vụ

Bags: là một định dạng để lưu trữ và phát lại dữ liệu từ ROS các gói tin. Bags là một cơ chế quan trọng để lưu trữ dữ liệu.

Cách xây dựng hệ thống trong ROS cho phép nguồn cung cấp tin và đối tượng nhận tin có thể tách rời nhau, và mỗi liên hệ được thực hiện thông qua thuộc tính tên. Tên là thuộc tính đóng vai trò rất quan trọng trong ROS: các nút, các chủ đề, các dịch vụ, và các tham số đều được đặt tên. Mỗi thư viện khách hàng ROS đều hỗ trợ dòng lệnh (command-line) để liên kết các tên này (remapping names), nhờ đó mà chương trình đã được biên dịch có thể cấu hình lại khi chạy ngay cả khi hoạt động trong cấu trúc biểu đồ tính toán khác.



Hình 1.8. Mô hình giao tiếp cơ bản trong ROS

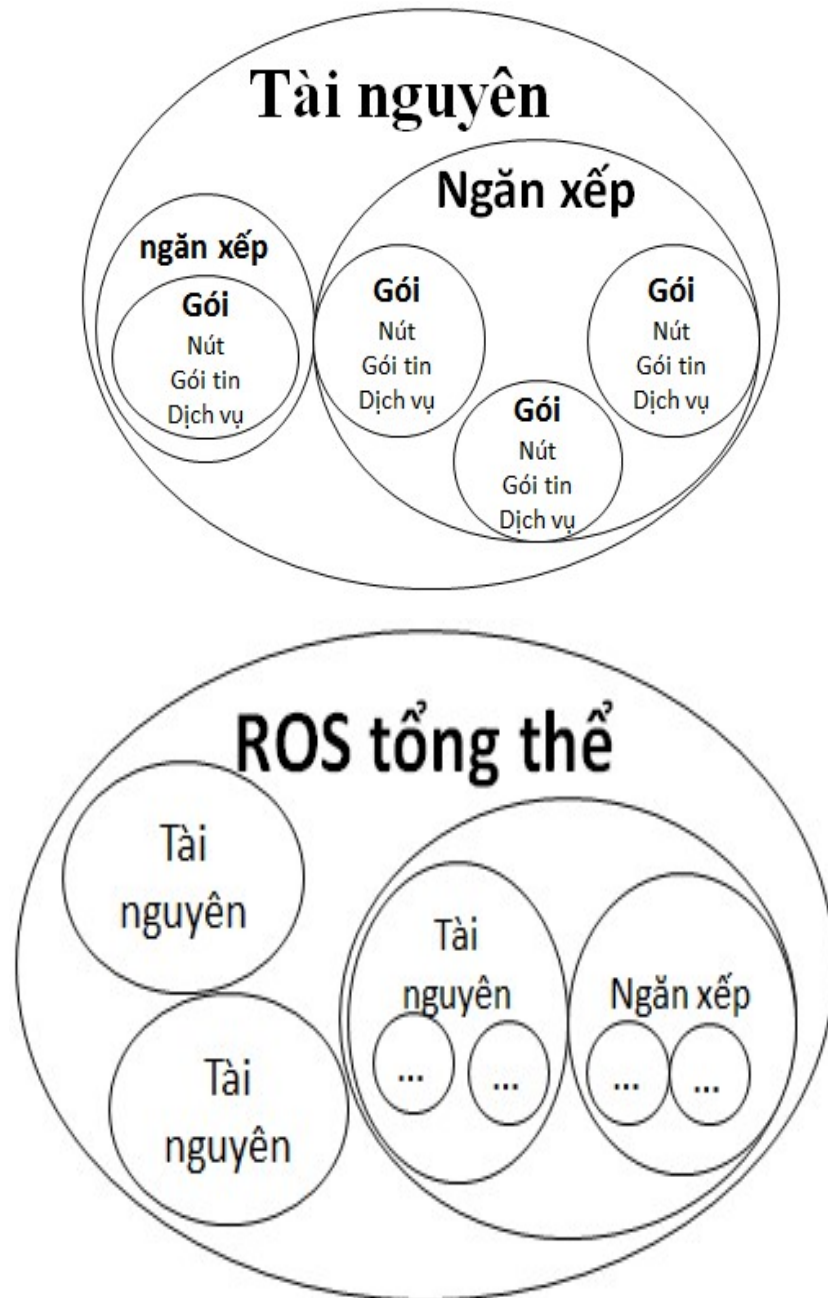
- **Cộng đồng ROS**

Cộng đồng ROS được định nghĩa là nguồn tài nguyên ROS mà các đơn vị nghiên cứu có thể trao đổi phần mềm và kiến thức. Các nguồn tài nguyên bao gồm:

**Phân phối ROS (ROS Distribution):** Sự phân phối ROS là bộ phiên bản các ngăn xếp tương thích mà người dùng có thể cài đặt. Phiên bản mới nhất hiện nay là phiên bản thứ tám lunar. Đề án sử dụng phiên bản là indigo nhằm đảm bảo tương thích với một số gói như slam\_gmapping, depthimage\_to\_laserscan định vị bằng thị giác và ước lượng tọa độ bằng encoder.

**Tài nguyên (Repository):** ROS là nguồn tài nguyên dựa trên cộng đồng mã nguồn mở, trong đó các thư viện nghiên cứu, trường đại học khác nhau có

thể phát triển đồng thời và công bố những mã nguồn trên mô hình robot của riêng họ.



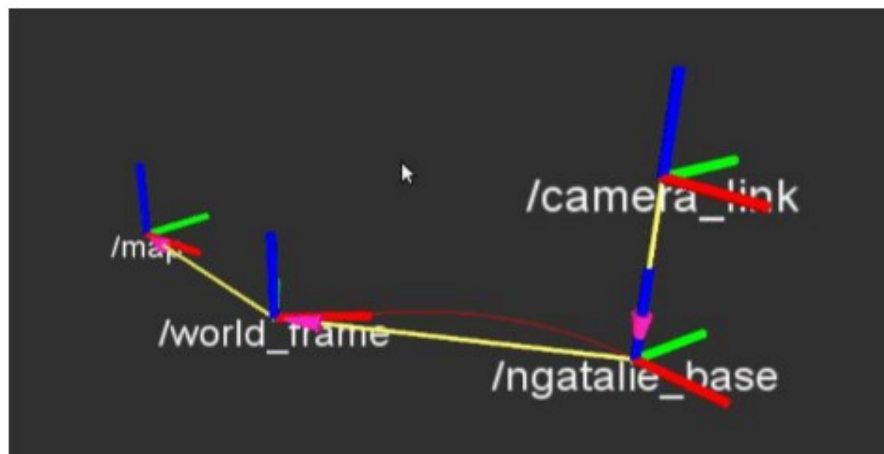
Hình 1.9. ROS tổng thể và tài nguyên trong ROS

ROS wiki: là bách khoa mở lưu trữ dữ liệu, tài liệu hướng dẫn về ROS. Bất kỳ ai cũng có thể đăng ký tài khoản để chia sẻ tài liệu, cập nhật hay sử dụng, viết bài hướng dẫn và đặt câu hỏi...Đây là một kênh tham khảo quan trọng bậc nhất khi làm quen với ROS.

- Các khái niệm cấp cao của ROS

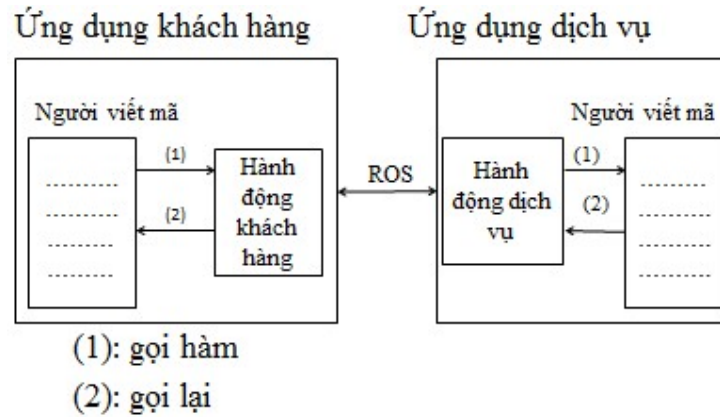
Ngoài các khái niệm cơ bản dựa trên cấu trúc cơ bản của một hệ điều hành. ROS còn quy chuẩn một số khái niệm liên quan đặc trưng tới các ứng dụng robot. Trong phạm vi đồ án, có một số khái niệm quan trọng cần quan tâm như ‘*hệ tọa độ*’, phép chuyển đổi giữa các hệ tọa độ ‘*tf*’, các kiểu gói tin đặc trưng cho robot và định dạng mô tả robot tổng quát.

Hệ trục tọa độ/phép chuyển đổi (Coordinate Frame/Transform): khái niệm hệ tọa độ và phép chuyển đổi hệ tọa độ (*tf*) liên quan đến các phần tử không gian của robot và mối quan hệ giữa các phần tử này, giúp cho việc hiểu đúng các dữ liệu cảm biến và giám sát, điều khiển robot trong không gian. Các cấu trúc trong ROS tuân theo cấu trúc cây, nghĩa là mỗi nhánh có tối đa một nhánh mẹ và có thể có nhánh con. Mỗi một nhánh trên cây này tồn tại một phép chuyển đổi ‘*tf*’ để chuyển đổi giữa nhánh mẹ và nhánh con. Gói ‘*tf*’ cung cấp các hàm dùng để chuyển hệ trục tọa độ, tính toán vị trí và quan hệ giữa các hệ trục tọa độ theo thời gian.



Hình 1.10. Các hệ tọa độ của robot và chuyển động trong không gian

Tác động/nhiệm vụ (Action/Tasks): các chương trình thực thi của ROS có thể được gọi là tác động hoặc nhiệm vụ. ActionLib cung cấp các hàm để chạy một chương trình thực thi trong ROS. Các tác động/nhiệm vụ trao đổi dữ liệu với nhau thông qua các chủ đề. Cơ chế xử lý, trao đổi dữ liệu có thể thông qua con trỏ hàm gọi lại (callback) hoặc gọi hàm (polling).



Hình 1.11. Giao tiếp giữa action client và server

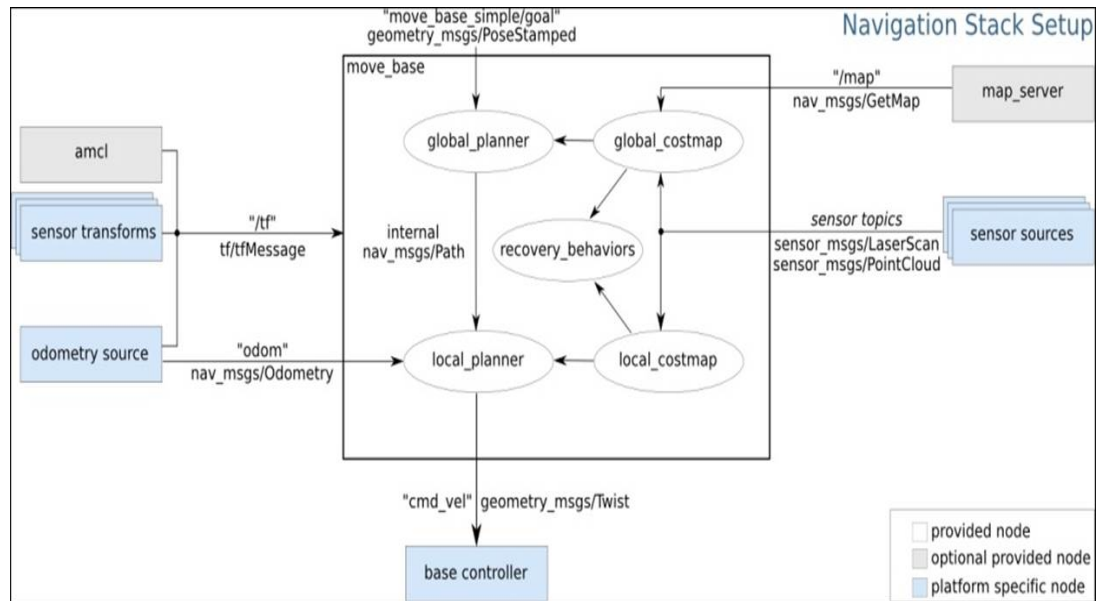
### 1.3 Các gói ứng dụng cần thiết cho bài toán định vị, định hướng, lập kế hoạch cho robot di động

#### 1.3.1 Gói ứng dụng lập kế hoạch di chuyển với ROS

Navigation Stack là một ứng dụng được phát triển đầy đủ nhất của ROS. Ý tưởng của Navigation Stack khá đơn giản, đó là một chương trình lấy thông tin từ đo lường (tọa độ ước lượng) từ các cảm biến, xử lý và xuất lệnh dưới dạng vận tốc xuống modul di động (mobile base). Sơ đồ bên dưới biểu diễn tổng quan về các phần tử của Navigation Stack. Navigation Stack khi triển khai sang cấp biểu đồ tính toán sẽ có dạng đầy đủ gồm một nút `move_base`, các nút `amcl` và `map_server` như trong sơ đồ Hình 1.12[4].

Khối màu trắng là các phần tử cần thiết đã được triển khai, các khối màu xám là các khối tùy chọn được hỗ trợ sẵn. Các khối màu xanh là các khối cần phải được xây dựng riêng cho từng mô hình robot để có thể ứng dụng được Navigation Stack trên mô hình.

Ta cần thiết kế các nút có thể cung cấp các phép chuyển đổi hệ tọa độ, nhận các thông tin từ cảm biến theo đúng kiểu gói tin (Message) và chuyển đổi lệnh điều khiển cho mobile base. Tiếp theo ta phải thiết lập Navigation Stack với các thông số liên quan tới cấu trúc và chuyển động của robot. Bên cạnh đó, có một số yêu cầu bắt buộc khi thiết kế robot ứng dụng Navigation Stack.



Hình 1.12. Biểu đồ tính toán của Navigation Stack

- Yêu cầu phần cứng

Navigation Stack được thiết kế để có thể áp dụng cho cả robot kiểu lái vi phân (differential drive robot) và robot có bánh lái (holonomic wheeled robot). Navigation Stack cho phép điều khiển mô đun di động (mobile base) bằng cách gửi vận tốc mong muốn theo dạng vận tốc trên các phương x, y và vận tốc góc của robot.

Navigation Stack yêu cầu lắp đặt một máy quét laser 2D đặt trên khung robot. Cảm biến được dùng để dựng bản đồ và định vị cho robot. Yêu cầu này có thể thỏa mãn với camera Kinect vì camera Kinect cho ta thông tin khoảng cách 3D nên có thể trích lấy giá trị 2D dễ dàng.

Navigation Stack được phát triển trên robot có dạng footprint đối xứng (hình chiếu của robot xuống bản đồ 2D), vì vậy navigation hoạt động tốt trên các robot có dạng đối xứng vuông hoặc tròn với tâm robot trùng với tâm quay tại chỗ. Đây là các mô hình tối ưu khi sử dụng Navigation Stack. Tuy nhiên, Navigation Stack vẫn có thể áp dụng cho robot có hình dạng và kích cỡ tùy ý, ta có thể khai báo một bán kính ngoại tiếp bao trùm hết cấu hình của robot để đảm bảo robot di chuyển và tránh được vật cản, tuy nhiên sẽ có hạn chế về độ linh hoạt của robot.

- Thiết đặt khung robot

Việc thiết đặt khung robot nhằm mô tả các thông số của robot và cho quá trình định hướng di động. Việc thiết đặt được hướng dẫn cụ thể trên wiki của ROS. Ở đây, trình bày có một số thiết đặt thực tế trong đồ án.

### **Costmap 2D**

Theo *Hình 1.12* trên, Navigation Stack dùng hai costmap để lưu trữ thông tin về các vận cản trong môi trường. Một costmap được dùng cho path-planning toàn cục, nghĩa là lộ trình tổng quát từ điểm hiện tại để đi đến điểm đích, và một costmap cho path-planning cục bộ và chuyển động tránh vật cản. Có một số thiết đặt chung cho cả hai costmap và một số dành riêng cho mỗi costmap. Ta tạo các thiết đặt này trong các tệp yaml và move\_base đọc vào khi khởi tạo bằng lệnh roslaunch. Dưới đây là các thiết đặt chung cho cả hai costmap:

```

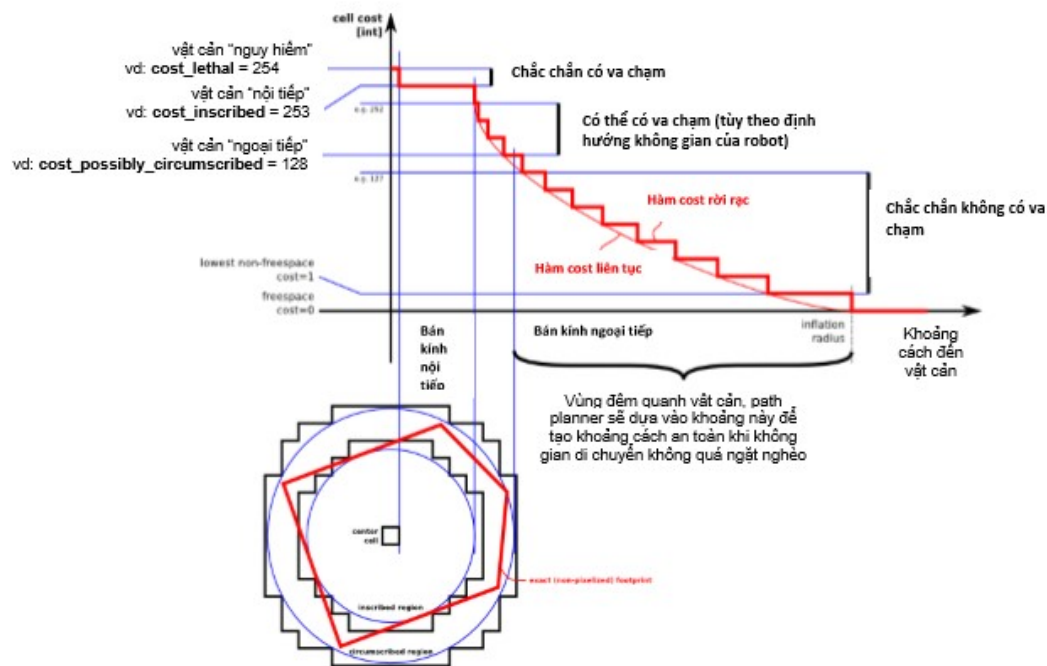
footprint: [[-0.15,-0.125], [-0.15,0.125], [0.15,0.125],[0.15, -0.125]]
#robot_radius: 0.33
obstacle_range: 1.5
raytrace_range: 3.0
observation_sources: scan
scan:
  data_type: LaserScan
  topic: scan
  marking: true
  clearing: true
  min_obstacle_height: 0.1
map_type: costmap

```

*obstacle\_range* là khoảng cách tối đa mà một vật cản được ghi nhận và có ảnh hưởng tới đường đi của robot. Raytrace\_range là khoảng cách tối thiểu đo được để ghi nhận không gian trống trước robot. Một cách dễ hiểu hơn, khi có một người xuất hiện trong khoảng obstacle\_range, người đó sẽ được ghi nhận là vật cản và robot sẽ tính toán đường đi sao cho không va chạm vật cản này. Khi người đó rời khỏi khoảng này, giả sử trong khoảng từ obstacle\_range đến raytrace\_range, robot phát hiện được một vật thể mới chẳng hạn như bức tường, vật cản cũ sẽ được giải phóng khỏi bản đồ.



Ta chọn một trong hai cách khai báo hình chiếu của robot trên mặt phẳng bằng thông số footprint hoặc robot\_radius. Cách đầu tiên nên dùng cho các robot có tâm quay tại chỗ trùng với tâm hình chiếu và cách còn lại dùng cho các robot có tâm quay không trùng tâm hình chiếu. Với cách khai báo footprint, tâm quay tại chỗ quy ước nằm tại điểm (0, 0), ta chỉ khai báo tọa độ các điểm của đa giác hình chiếu và trục x hướng về hướng nhìn của robot. Cả hai cách đều giúp xác định các bán kính nội tiếp và ngoại tiếp (inscribed\_radius và circumscribed\_radius) của footprint để kết hợp với inflation\_radius nhằm tính toán giá trị chiếm chỗ (cost) của các ô theo sơ đồ sau[5]:



Hình 1.13. Costmap function

Cost của các ô từ một ô (tùy chỉnh kích thước) được xác định có vật cản, phạm vi từ điểm đó mở rộng ra một khoảng bằng bán kính nội tiếp, nếu tâm robot nằm trong khoảng này ta chắc chắn có va chạm và có cost bão hòa. Trong khoảng từ bán kính nội tiếp đến bán kính ngoại tiếp, va chạm có thể xảy ra tùy theo định hướng không gian của robot. Các cell có cost thấp hơn ngưỡng cost\_possibly\_circumscribed sẽ được coi là vùng trống. Giá trị cost sẽ quyết định xu hướng mà các khối lập kế hoạch di chuyển sẽ ưu tiên chọn đường đi qua nó.

Ta sử dụng công thức sau để có được hàm cost mong muốn, với `costmap_2d::INSCRIBED_INFLATED_OBSTACLE` bằng 254:  $\exp(-\text{cost\_scaling\_factor} * (\text{distance\_from\_obstacle} - \text{inscribed\_radius})) * (\text{costmap\_2d::INSCRIBED_INFLATED_OBSTACLE} - 1)$ .

Các thiết đặt còn lại tương đối đơn giản. `Observation_sources` khai báo các loại cảm biến để phát hiện vật cản. Ta khai báo kiểu loại dữ liệu, chủ đề, cờ báo sử dụng dữ liệu để đánh dấu vật cản (marking), hay giải phóng vật cản (clearing).

Tiếp theo ta cần khai báo riêng cho `cosmap` toàn cục và `cosmap` địa phương. Với `cosmap` toàn cục ta có khai báo tiêu chuẩn như sau:

```
global_costmap:
  global_frame: /map
  robot_base_frame: /ngatalie_base
  update_frequency: 2.0
  static_map: true
```

Đầu tiên ta khai báo hệ tọa độ `global_frame` cho các đối tượng trong bản đồ toàn cục. Sau đó là hệ tọa độ `robot_base_frame` gắn với robot trong hệ tọa độ toàn cục. Tiếp theo là định nghĩa hệ tọa độ gắn với robot, tần số cập nhật các đối tượng trong bản đồ. Cuối cùng là cờ báo cho phép khởi tạo các vật cản cố định ban đầu từ một bản đồ thu được trước đó. Sau đó ta thiết đặt các thông số cho bộ `cosmap` địa phương với các thông số tiêu chuẩn như sau:

```
local_costmap:
  global_frame: /odom
  robot_base_frame: /base_link
  update_frequency: 5.0
  publish_frequency: 5.0
  static_map: false
  rolling_window: true
  meter_scoring: true
  width: 10.0
  height: 10.0
  origin_x: 992.0
```

```
origin_y: 992.0
resolution: 0.05
transform_tolerance: 0.5
```

Các thông số ban đầu cũng giống như costmap toàn cục. Ta quan tâm đến các thông số: `rolling_window` có giá trị `true` sẽ giữ tâm của costmap vào giữa robot. Chiều rộng và chiều dài là kích thước costmap 2D (`origin_x` và `origin_y`) và độ phân giải là kích thước của mỗi ô theo mét(`transform_tolerance`). Các thông số còn lại tương tự như các tham số của *global\_costmap*.

### **base local planner**

`base_local_planner` sẽ dựa trên giá trị costmap và các thông số động học của robot được khai báo để xuất lệnh `mobile_base`, các thiết đặt chính cho robot khi tạo tệp yaml như sau:

```
TrajectoryPlannerROS:
  max_vel_x: 0.15
  min_vel_x: -0.15
  max_vel_y: 0.0
  min_vel_y: 0.0
  max_vel_theta: 0.25
  min_vel_theta: -0.25
  min_in_place_vel_theta: 0.25
  escape_vel: -0.1

  acc_lim_theta: 0.2
  acc_lim_x: 1.0
  acc_lim_y: 0.0

  yaw_goal_tolerance: 0.2
  xy_goal_tolerance: 0.2

  sim_time: 3.0
  vx_samples: 20
  vtheta_samples: 30

  holonomic_robot: false
  meter_scoring: true
```

```

dwa: false
pdist_scale: 1.0
gdist_scale: 0.8
occdist_scale: 0.01
publish_cost_grid_pc: true

```

Tệp yaml trên khai báo các giới hạn vận tốc và gia tốc của mobile base, đồng thời khai báo holonomic\_robot cho biết mobile\_base có thuộc dạng lái holonomic(có bánh lái, lái kiểu trượt) hay không.

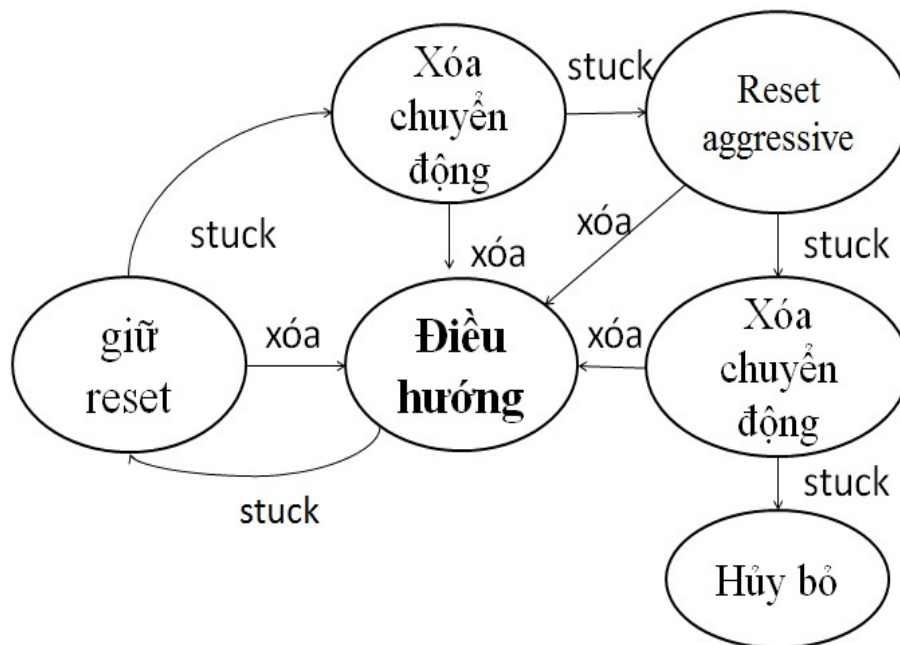
Ngoài ra, còn có các thông số sau cho phép ta thiết đặt dung sai (tolerance) cho sai số đến đích như sau:

*yaw\_goal\_tolerance* (double, default: 0.05): dung sai cho sai số góc heading của robot xung quanh giá trị đích.

*xy\_goal\_tolerance* (double, default: 0.10): dung sai cho sai số vị trí của robot xung quanh giá trị đích.

### **Recovery behavior của move\_base**

Trạng thái phục hồi là sơ đồ trạng thái mà Navigation Stack sẽ sử dụng khi rơi vào khu vực không nhận diện được. Robot sẽ cố gắng di chuyển và xác định lại vị trí hiện thời để tiếp tục đến đích hoặc bỏ cuộc.



Hình 1.14. Trạng thái phục hồi của move\_base

Đầu tiên, các vật cản bên ngoài một phạm vi do người dùng xác định sẽ được dọn khỏi bản đồ trên bộ nhớ (local). Sau đó robot sẽ thử thực hiện động tác xoay tại chỗ để xác định lại vị trí trong không gian. Nếu bước này thất bại, robot sẽ reset bằng cách bỏ qua các vật cản bên ngoài khu vực hình chữ nhật ngoại tiếp mà robot có thể xoay tại chỗ và thực hiện lại động tác này. Nếu tất cả đều thất bại robot sẽ xác định đích đến là bất khả thi và thông báo rằng nó đã bỏ cuộc.

### 1.3.2 Gói ứng dụng định vị trên bản đồ có sẵn AMCL

*Adaptive Monte Carlo Localization (AMCL)* là một gói ứng dụng của ROS được xây dựng dựa trên thuật toán của chuyển động xác suất và phương pháp định vị Monte Carlo thích nghi sẽ được giới thiệu ở phần sau. Yêu cầu sử dụng gói này là chúng ta có một bản đồ đã được xây dựng từ trước đó dưới định dạng ảnh (.png) và tệp đường dẫn (.yaml), một máy quét laser, một mô hình robot có trả về vận tốc mỗi bánh xe, và biến đổi hệ tọa độ từ map  $\rightarrow$  odom  $\rightarrow$  base\_link[6].

### 1.3.3 Gói truyền thông giữa ROS và Arduino rosserial\_python

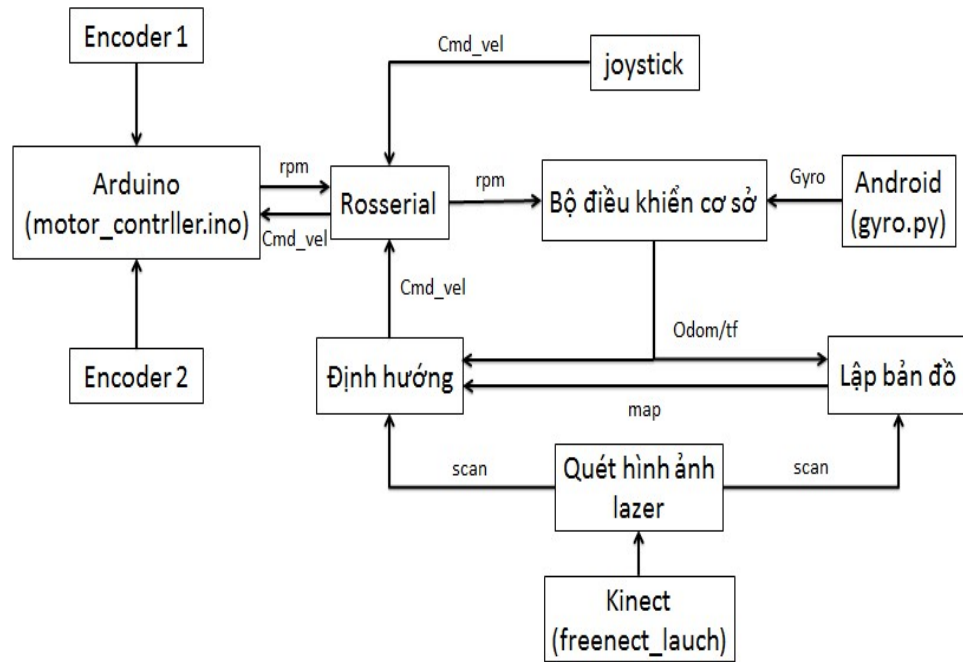
Như đã giới thiệu ở mục 1.3.1 để điều khiển *mobile\_base*, ta cần nút *rosserial\_python* thực hiện thao tác đăng ký tới chủ đề “cmd\_vel” được gửi từ *move\_base* và chuyển đổi các giá trị vận tốc dài và vận tốc góc trong chủ đề này sang giá trị vận tốc trên mỗi bánh xe.

*Rosserial\_python* là một nút trung gian có nhiệm vụ trao đổi dữ liệu hai chiều giữa *mobile\_base* và Navigation Stack. Để thực hiện chức năng này ta phải lập trình giao tiếp cổng nối tiếp trên máy tính. ROS có hỗ trợ giao tiếp qua cổng nối tiếp với gói *rosserial*, tuy nhiên *rosserial* có một giao thức riêng và chỉ mới được ứng dụng với dòng MCU Arduino.

### 1.3.4 Gói ứng dụng định vị xây dựng bản đồ SLAM\_GMAPPING

Bài toán định vị và lập bản đồ đồng thời (Simultaneous Localization and Mapping – SLAM) là một bài toán đặc biệt luôn đi kèm với bài toán định hướng. SLAM được định nghĩa là việc xây dựng hoặc cập nhật liên tục một bản

đồng, đồng thời định vị vị trí của robot trong bản đồ đó. Trong SLAM, hai tác vụ này không thể tách rời, khi biết chính xác vị trí của mình thì robot mới có thể đưa các cập nhật mới vào bản đồ một cách hợp lý. Đồng thời, khi bản đồ được cập nhật chính xác, robot mới có thể đánh giá được sự dịch chuyển của mình so với môi trường[7].



Hình 1.15. Sơ đồ sử dụng gói ứng dụng SLAM - Gmapping

#### Thông tin đo lường với SLAM\_GMAPPING

Một khía cạnh quan trọng của SLAM là thông tin đo lường. Mục tiêu của thông tin đo lường là cung cấp một vị trí gần đúng của robot được đo bằng sự di chuyển các bánh xe của robot, để phục vụ như là dự đoán ban đầu về nơi mà robot có thể nằm trong EKF. Việc lấy dữ liệu đo từ một robot ER1 khá dễ dàng bằng cách sử dụng máy chủ telnet cài sẵn. Có thể chỉ cần gửi một chuỗi văn bản đến máy chủ telnet trên một cổng cụ thể và máy chủ sẽ trả về câu trả lời.

Phần khó khăn về dữ liệu đo lường và dữ liệu laze là để có được thời gian đúng. Dữ liệu laser tại một thời điểm  $t$  sẽ trễ nếu dữ liệu đo lường được lấy ra sau đó. Để chắc chắn rằng dữ liệu trả về đồng thời, người ta có thể ngoại suy dữ liệu. Nếu người ta có quyền kiểm soát khi các phép đo được trả lại thì dễ dàng nhất để yêu cầu cả giá trị của máy quét laze và dữ liệu đo lường cùng một lúc.

### 1.3.5 Gói ứng dụng cho điều khiển camera Kinect

Ngoài việc nhận diện vật cản, đo khoảng cách camera Kinect còn được khai thác cho các ứng dụng khác như quét ảnh 3D, lập bản đồ và xác định địa hình di chuyển cho robot. Từ khi xuất hiện, cộng đồng ROS đã nhanh chóng phát triển các gói driver cho Kinect do những ưu thế của Kinect so với các cảm biến khoảng cách đang có như stereo camera, cảm biến laser scan và cảm biến siêu âm. Trong số các driver, phổ biến nhất là các gói dựa trên nền tảng của OpenNI từ PrimeSense là công ty phát triển kỹ thuật đo khoảng cách ứng dụng cho Kinect. Thư viện OpenNI được xem là thư viện mạnh nhất hỗ trợ Kinect, thư viện này hỗ trợ đa ngôn ngữ trên nhiều nền tảng khác nhau, giúp cho người lập trình có thể viết các ứng dụng trên Kinect một cách dễ dàng. Mục đích chính của OpenNI là xây dựng các hàm API chuẩn, cho phép thư viện có khả năng kết hợp với các middleware nhằm làm tăng sức mạnh cho Kinect. Một số gói ROS sử dụng OpenNI như `openni_camera` hỗ trợ truy cập dữ liệu từ các camera và `openni_launch` xử lý dữ liệu từ camera cho ra định dạng PointCloud trước khi ứng dụng cho các giải thuật trong ROS. Ngoài ra ta còn có gói độc lập `kinect_aux` có ứng dụng điều khiển góc nghiêng của Kinect[8].

Các driver của Kinect đều hướng đến việc xử lý dữ liệu từ camera để cho ra được định dạng cơ bản là point cloud. Với định dạng này ta có thể ứng dụng các hỗ trợ từ Point Cloud Library [9]. PCL là thư viện hỗ trợ cho việc xử lý ảnh trong không gian 3D. Thư viện này được xây dựng với nhiều giải thuật như bộ lọc, khôi phục bề mặt (surface reconstruction), phân vùng (segmentation), ước lượng đặc tính vật (feature estimation),... PCL có thể dùng trên nhiều nền tảng như Linux, MacOS, Windows and Android.

Có thể nói PCL là sự kết hợp của nhiều module nhỏ. Những module này thực chất cũng là các thư viện thực hiện chức năng riêng lẻ trước khi được PCL sử dụng. Các thư viện cơ bản là:

Eigen: một thư viện mở hỗ trợ cho các phép tính toán tuyến tính;

FLANN: (Fast Library for Approximate Nearest Neighbors) hỗ trợ trong việc tìm kiếm nhanh các điểm lân cận trong không gian 3D;

Boost: giúp cho việc chia sẻ con trỏ trên tất cả các module và thuật toán trong PCL để tránh việc sao chép trùng lặp dữ liệu đã được lấy về trong hệ thống;

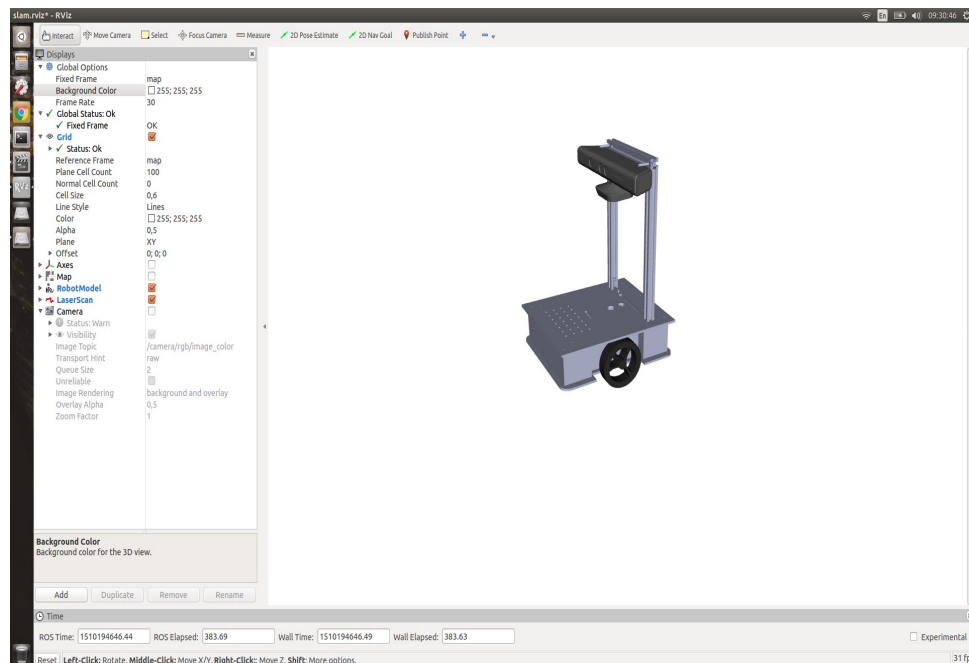
VTK: (Visualization Toolkit) hỗ trợ cho nhiều nền tảng cho việc thu về dữ liệu 3D, hỗ trợ việc hiển thị và ước lượng thể tích vật;

CMinPack: một thư viện mở giúp cho việc giải quyết các phép toán tuyến tính và phi tuyến.

### **Định dạng mô tả robot tổng quát – Universal Robot Description**

#### **Format:**

Đây là ngôn ngữ mô tả các phần tử, các kết nối giữa các phần tử trên robot. Các khớp có thể được khai báo là cố định hoặc chuyển động. URDF là cần thiết cho các mô hình phức tạp có các phần tử như cánh tay robot, robot omni... URDF giúp cho việc mô phỏng trong quá trình phát triển robot[3].



*Hình 1.16. Mô hình URDF của robot*



## CHƯƠNG 2: CƠ SỞ LÝ THUYẾT ĐỊNH VỊ, ĐỊNH HƯỚNG, LẬP KẾ HOẠCH DI CHUYỂN CHO ROBOT DI ĐỘNG

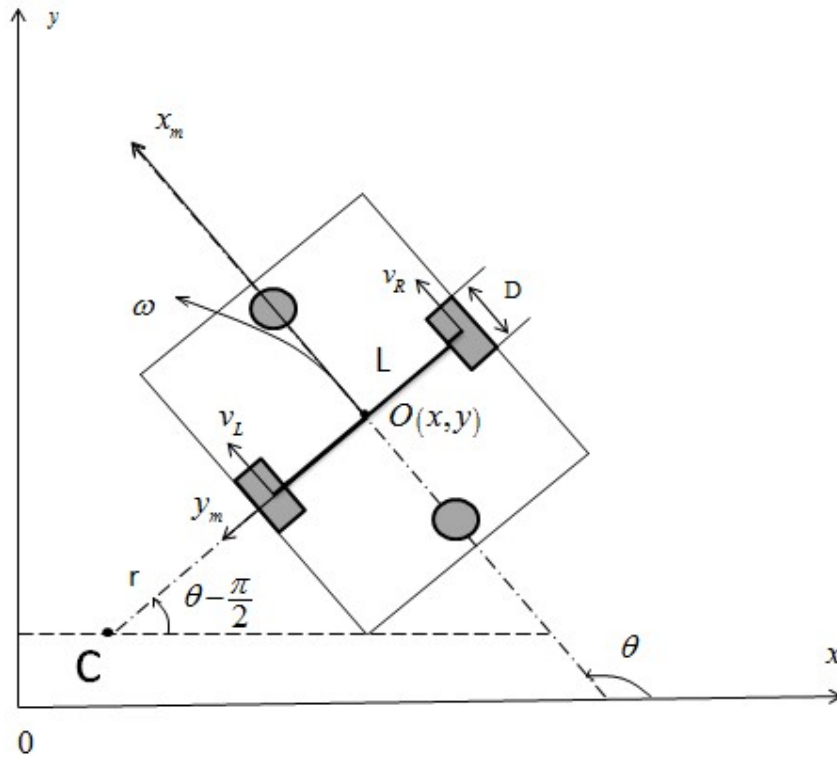
### 2.1 Mô hình động học của robot di động

Robot tự hành trong đồ án được thiết kế theo mô hình kiểu lái vi phân. Giả sử Robot chuyển động không trượt, ta sẽ ước lượng tọa độ của Robot dựa vào giá trị hồi tiếp từ encoder. Trong hệ tọa độ Oxy, Robot chuyển động với vận tốc tịnh tiến là  $v_R, v_L$  và vận tốc xoay  $\omega$ . Trong khoảng thời gian rất nhỏ  $dt$ ,  $v$  và  $\omega$  không đổi, chuyển động của Robot là chuyển động tròn quanh tâm quay tức thời C với bán kính  $r$ [21].

Tại khoảng thời gian đầu  $dt$ , tọa độ suy rộng  $O_m$  của Robot trong hệ tọa độ Oxy là  $[x \ y \ \theta]^T$ , tại điểm  $O_m$  ta có tọa độ của tâm C:

$$x_C = x - r \cos\left(\theta - \frac{\pi}{2}\right) = x - r \sin \theta \quad (2.1)$$

$$y_C = y - r \sin\left(\theta - \frac{\pi}{2}\right) = y - r \cos \theta \quad (2.2)$$



Hình 2.1. Mô hình động học robot

Sau khoảng thời gian  $dt$ , góc  $\theta$  thay đổi một lượng  $d\theta = \omega dt$ , tọa độ mới của  $O_m$  sau khoảng thời gian  $dt$  là  $\begin{bmatrix} x' & y' & \theta' \end{bmatrix}^T$ , sử dụng phương trình (2.1) và (2.2) để tìm quan hệ giữa tâm C và điểm này, ta có quan hệ giữa tọa độ mới và tọa độ cũ:

$$\begin{aligned} x' - r \sin(\theta + \omega dt) &= x - r \sin \theta \\ y' - r \cos(\theta + \omega dt) &= y - r \cos \theta \\ \Rightarrow \begin{bmatrix} x' \\ y' \\ \theta' \end{bmatrix} &= \begin{bmatrix} x \\ y \\ \theta \end{bmatrix} + \begin{bmatrix} r \sin(\theta + \omega dt) - r \sin \theta \\ r \cos(\theta + \omega dt) - r \cos \theta \\ \omega dt \end{bmatrix} \end{aligned} \quad (2.3)$$

Ta sẽ tìm quan hệ giữa các đại lượng  $v, \omega, r$  với giá trị thu được từ encoder và thay vào phương trình trên. Sử dụng quan hệ vận tốc dài và vận tốc góc của một vật chuyển động tròn, ta có các phương trình sau:

$$v_R = \omega \left( r + \frac{L}{2} \right) \quad (2.4)$$

$$v_L = \omega \left( r - \frac{L}{2} \right) \quad (2.5)$$

$$v = \omega r \quad (2.6)$$

Trong đó:

- $v_R$  vận tốc dài trục bánh phải
- $v_L$  vận tốc dài trục bánh trái
- $L$  khoảng cách giữa 2 tâm bánh xe
- $r$  bán kính quay tức thời
- $\omega$  vận tốc quay của robot

Lấy phương trình (2.4) trừ phương trình (2.5) về theo về ta được quan hệ giữa vận tốc góc với vận tốc hai bánh xe:

$$\omega = \frac{v_R - v_L}{L} \quad (2.7)$$

Lấy phương trình (2.4) cộng phương trình (2.5) về theo về, ta có:

$$\frac{1}{2}(v_R + v_L) = \omega r \quad (2.8)$$

Ta suy ra được bán kính quay tức thời  $r$  của Robot từ các phương trình (2.7) và (2.8)

$$r = \frac{v}{\omega} = \frac{L}{2} \cdot \frac{v_R + v_L}{v_R - v_L} \quad (2.9)$$

Thay  $v_R = \frac{ds_R}{dt}$  và  $v_L = \frac{ds_L}{dt}$  vào phương trình (2.9) với  $ds_R$  và  $ds_L$  là độ dài mỗi bánh dịch chuyển được trong khoảng thời gian  $dt$ . Thay lại  $r$  và  $\omega$  trong biểu thức này vào phương trình (2.3), ta có:

$$\begin{bmatrix} x' \\ y' \\ \theta' \end{bmatrix} = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix} + \begin{bmatrix} \frac{L}{2} \cdot \frac{ds_R + ds_L}{ds_R - ds_L} \cdot \left[ \sin \left( \theta + \frac{ds_R - ds_L}{L} \right) - \sin \theta \right] \\ \frac{L}{2} \cdot \frac{ds_R + ds_L}{ds_R - ds_L} \cdot \left[ \cos \left( \theta + \frac{ds_R - ds_L}{L} \right) - \cos \theta \right] \\ \frac{ds_R - ds_L}{L} \end{bmatrix} \quad (2.10)$$

Phương trình trên cho ta quan hệ chính xác giữa chuyển động của hai bánh xe với tọa độ của Robot. Ta có thể xây dựng giải thuật xấp xỉ phương trình trên để cập nhật tọa độ của Robot với đầu vào ở mỗi thời điểm cập nhật là  $n_R$  và  $n_L$ , là số đếm tăng thêm từ encoder của mỗi bánh và tọa độ  $[x \ y \ \theta]$  cuối chu kỳ trước. Khi đó ta gán giá trị  $n_R D\pi$  và  $n_L D\pi$  (Với  $D$  là đường kính bánh xe của Robot) vào vị trí của  $ds_R$  và  $ds_L$  trong phương trình (2.10).

Ngoài ra ta có thể xấp xỉ phương trình (2.10) và giảm đáng kể khối lượng tính toán nếu việc cập nhật tọa độ được thực hiện trên hệ thống nhúng. Ta có:

$$\begin{bmatrix} x' \\ y' \\ \theta' \end{bmatrix} = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix} + \begin{bmatrix} \frac{ds_R + ds_L}{2} \cdot \frac{\sin\left(\theta + \frac{ds_R - ds_L}{L}\right) - \sin\theta}{\frac{ds_R - ds_L}{L}} \\ \frac{ds_R + ds_L}{2} \cdot \frac{\cos\left(\theta + \frac{ds_R - ds_L}{L}\right) - \cos\theta}{\frac{ds_R - ds_L}{L}} \\ \frac{ds_R - ds_L}{L} \end{bmatrix}$$

Nếu như chuyển động quay của Robot rất nhỏ, ta có thể lấy xấp xỉ  $\lim \frac{ds_R - ds_L}{L} \rightarrow 0$  nên phương trình có dạng đơn giản hơn:

$$\begin{bmatrix} x' \\ y' \\ \theta' \end{bmatrix} = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix} + \begin{bmatrix} \frac{ds_R + ds_L}{2} \cdot \cos\theta \\ \frac{ds_R + ds_L}{2} \cdot \sin\theta \\ \frac{ds_R - ds_L}{L} \end{bmatrix}$$

Để điều khiển Robot, ta có quan hệ giữa  $v_R$ ,  $v_L$  với  $v$  và  $\omega$  như sau:

$$v_R = v + \frac{L}{2} \cdot \omega \quad (2.12)$$

$$v_L = v - \frac{L}{2} \cdot \omega \quad (2.11)$$

## 2.2 Thuật toán định vị

### 2.2.1 Phương pháp định vị GPS

Hệ thống định vị toàn cầu – GPS do bộ quốc phòng Mỹ phát triển và điều hành. Ban đầu để ứng dụng cho mục đích quân sự, ngày nay đã cho phép cả sử dụng cho dân sự[3]. Các máy thu GPS có độ chính xác trung bình trong vòng 15m. GPS dân sự dùng tần số L1 (sóng cực ngắn 1575.42 MHz) trong dải UHF.

Hệ thống GPS gồm có 3 phần:

- Phần vũ trụ (Space Segment): gọi là bộ phận không gian là một hệ thống gồm nhiều vệ tinh bay xung quanh trái đất theo các quỹ đạo khác nhau được điều khiển bởi bộ phận điều khiển. Gồm 24 vệ tinh xung quanh trái đất, độ cao của vệ tinh so với mặt đất là 20.183km, chu kỳ quay xung quanh trái đất là 11h57'58".
- Phần điều khiển (Control Segment): là một hệ thống các thiết bị đặt tại nhiều nơi khác nhau trên trái đất được sử dụng để giám sát và điều khiển các vệ tinh. Gồm một trạm điều khiển chính (đặt tại Mỹ), 5 trạm thu số liệu, 3 trạm truyền số liệu.
- Phần sử dụng (Use Segment): là các thiết bị GPS được lắp đặt trên các phương tiện hoặc đem theo người (điện thoại di động, đồng hồ, máy ảnh...). Chúng có chức năng thu tín hiệu phát ra từ các vệ tinh và tự tính toán vị trí của nó dựa trên các thông tin thu được. Như vậy, về bản chất tọa độ thiết bị GPS là kết quả tính toán từ thông tin về vị trí, khoảng cách giữa thiết bị GPS và các vệ tinh mà nó có thể kết nối được. Việc thu phát của hệ thống GPS được thực hiện theo tần suất 1s một lần.

Các vệ tinh GPS truyền tín hiệu từ không gian để truyền về máy thu GPS tại Mặt đất, sử dụng những tín hiệu này để tính toán vị trí trong không gian 3 chiều (kinh độ, vĩ độ và độ cao) tại thời điểm hiện tại.

GPS hoạt động trong mọi điều kiện thời tiết, nhiệt độ không gian khác nhau và mọi nơi trên trái đất. GPS được mọi người trên toàn cầu sử dụng miễn phí.

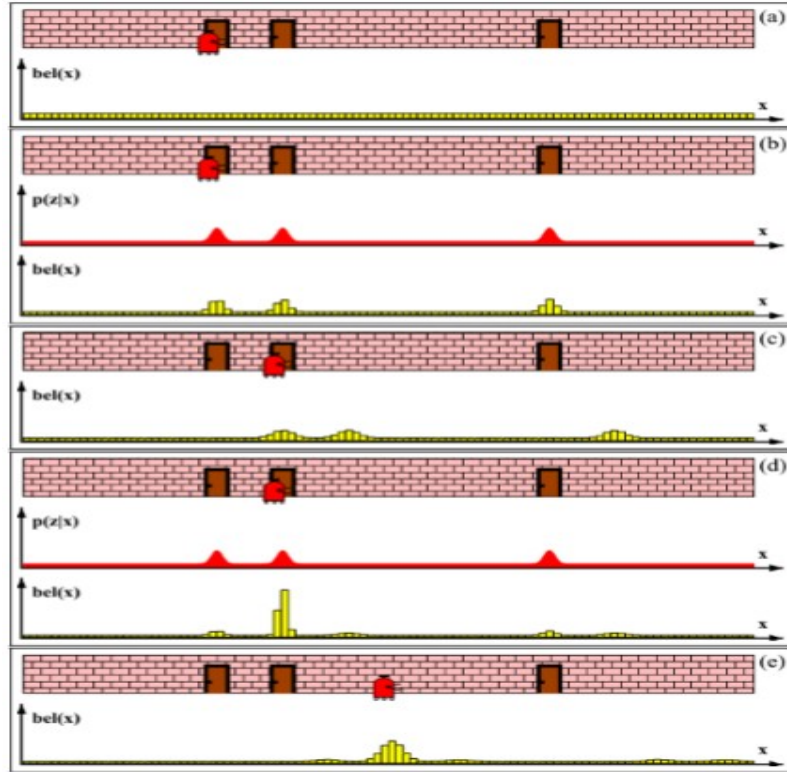
### *2.2.2 Phương pháp định vị Monte Carlo thích nghi*

Trong phần 2.1, mặc dù ta có được mô tả toán học chính xác chuyển động của Robot nhưng biểu thức (2.10) chứa các đại lượng vi phân chỉ có thể lấy xấp xỉ. Đồng thời giá trị trả về từ encoder lại cho ta giá trị rời rạc, cho nên tọa độ ước lượng vẫn chứa một hàm sai số qua mỗi bước lặp, sai số tích lũy dần dần và tọa độ ước lượng ngày càng có độ tin cậy thấp hơn.

Trong mọi trường hợp, đối với một robot cụ thể được thiết kế theo một mô hình toán học cho trước nào đó, ta có thể suy ra được quỹ đạo, vị trí của Robot từ mô hình của nó. Tuy nhiên, một vấn đề tất yếu là ta không bao giờ tìm được vị trí chính xác của Robot, mà chỉ có thể tìm được họ các vị trí có xác suất cao nhất mà Robot có thể rơi vào. Ngoài ra, khi Robot hoạt động với sự tác động của môi trường như thông số đo lường, hao mòn ... lại làm xuất hiện sai số và kết quả điều khiển không đúng như mô hình thực tế. Do đó cần phải quan tâm đến các yếu tố gây ra sai số này.

Trong mô hình động học xác suất, một trong những định lý quan trọng nhất của xác suất học đó là định lý Bayes[10]. Định lý Bayes cho phép tính xác suất của một biến ngẫu nhiên  $x$ , khi biết sự liên quan  $z$  và  $m$  đã xảy ra. Trong phương pháp định vị Monte Carlo thích nghi [11] với  $x$  là tọa độ của robot và  $z$ ,  $m$  là giá trị đo được từ cảm biến và thông tin bản đồ đã biết. Giả sử ta biết trước một hành lang có ba cửa phòng và robot đang ở một vị trí bất kỳ trong hành lang. Robot được trang bị một cảm biến và có thể cho ta một biến ngẫu nhiên  $z$  cho biết robot đang ở cạnh một cánh cửa với xác suất  $z$  phân bố theo hàm Gauss. Tại thời điểm đầu tiên, robot không biết được nó đang ở đâu trong đoạn hành lang. Hàm cảm tính  $bel(x)$  của robot có dạng phân bố đều trên toàn bộ chiều dài của hành lang như đồ thị đầu tiên trong Hình 2.2. Khi khởi động hành trình, cảm biến trên robot cho tín hiệu robot đang ở cạnh một cánh cửa. Phân bố xác suất  $\rho(z|x)$ , là xác suất để robot nhận được một tín hiệu cảm biến, như vậy trên toàn hành lang theo định lý Bayes, phân bố tập trung quanh vị trí các cánh cửa. Tại thời điểm đó,  $bel(x)$  của robot sẽ có dạng tương tự như  $\rho(z|x)$ . Nói cách khác, robot biết rằng nó đang ở gần một trong ba cánh cửa trong hành lang. Đồ thị thứ hai và thứ ba trong hình minh họa là hai phân bố xác suất  $\rho(z|x)$  và  $bel(x)$  tại thời điểm này. Khi robot di chuyển một đoạn so với vị trí hiện tại mà nó vừa cảm nhận được cánh cửa, dựa vào cảm biến và phản hồi robot sẽ dự đoán vị trí hiện thời, do đó  $bel(x)$  cũng dịch chuyển theo hướng di chuyển. Tuy nhiên do đo lường có sai số nên đỉnh tại các vùng xác suất cao cũng sẽ thấp dần như

đồ thị thứ tư trong *Hình 2.2*. Tiếp theo khi cảm biến cho ta phép đo mới của  $z$ . Cảm biến của robot sẽ có phân bố lớn xung quanh vị trí cánh cửa thứ hai và phân bố ở các khu vực khác đều giảm mạnh như trong đồ thị thứ sáu. Từ sự chắc chắn tại cánh cửa hiện tại, trong khoảng từ đó đến cánh cửa kế tiếp, cảm tính của robot dựa vào cảm biến, đo lường vẫn cho vị trí thực tế có giá trị cao mặc dù có giảm và sẽ giảm, và được bổ sung lại khi robot đến được cánh cửa thứ 3.



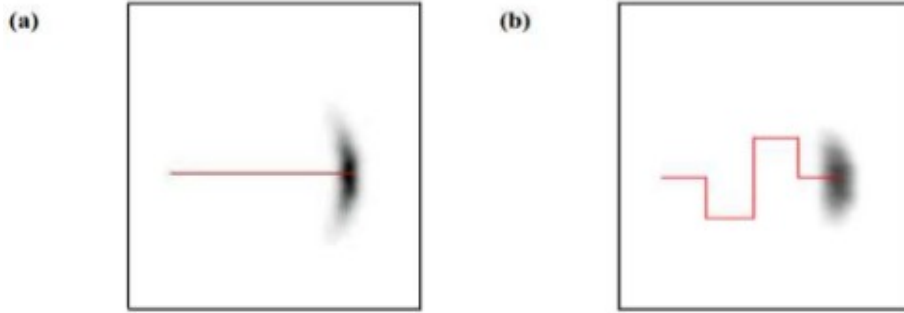
*Hình 2.2. Minh họa về quá trình định vị theo phương pháp xác suất, phân bố xác suất được biểu diễn ở dạng rời rạc*

Ta thử khảo sát trường hợp khi chưa có thông tin về bản đồ  $m$ , tọa độ ước lượng của robot có dạng một phân bố xác suất hậu nghiệm (*posterior probability*). Xác suất hậu nghiệm của một biến ngẫu nhiên là xác suất có điều kiện mà nó nhận được khi một vật có liên quan được xét đến. Phân bố xác suất hậu nghiệm của một biến ngẫu nhiên khi cho trước giá trị của một biến khác có thể tính theo Định lý Bayes. Ta biểu diễn phương trình động học xác suất của robot dưới dạng một hàm phân bố như sau:  $p(x_t | u_t, x_{t-1})$ .

Với  $x_t$  và  $x_{t-1}$  là trạng thái của robot tại thời điểm  $t$  và  $t-1$ ,  $u_t$  là giá trị điều khiển tại thời điểm  $t$ .

Phương trình này mô tả hàm phân bố xác suất hậu nghiệm, với điều kiện ràng buộc phụ thuộc vào trạng thái động học của robot, cần xét khả năng robot đạt trạng thái  $x_t$  với điều kiện giá trị điều khiển là  $u_t$  và biết robot đang ở trạng thái  $x_{t-1}$ .

Trong trường hợp cụ thể,  $u_t$  có thể được xác định nhờ đo lường, tuy nhiên khi phân tích lý thuyết ta sẽ xem như  $u_t$  là giá trị điều khiển.



Hình 2.3. Minh họa mô hình xác suất của phương trình động học

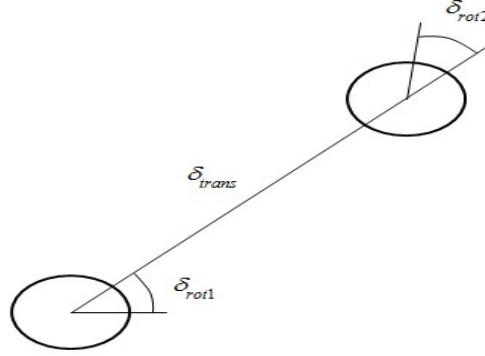
Ta giả sử quá trình di chuyển tịnh tiến trên một đoạn đường, sẽ xảy ra hiện tượng trượt hoặc có sự chênh lệch nhất thời vận tốc trên hai bánh, robot sẽ có một phân bố xác suất vị trí ở cuối đoạn đường, biểu thị độ chắc chắn trong một lân cận gần điểm cuối lý tưởng. Hình thứ hai minh họa độ chắc chắn trong một quãng đường phức tạp hơn. Ta thấy độ chắc chắn này phân bố trải đều hơn.

Với mô hình đo lường, ta sử dụng các số liệu chuyển động tương đối của robot. Trong khoảng thời gian  $(t-1, t]$  robot chuyển từ trạng thái  $x_{t-1}$  sang  $x_t$ .

Thông qua đo lường ta nhận được sự di chuyển của robot từ  $\bar{x}_{t-1} = (\bar{x} \quad \bar{y} \quad \bar{\theta})$  sang  $\bar{x}_t = (\bar{x}' \quad \bar{y}' \quad \bar{\theta}')$ . Trên Hình 2.4, đường thẳng màu đen là thể hiện cho kết quả đo lường trong hệ trục tọa độ gắn liền với robot (chưa biết tọa độ tương đối của hệ trục tọa độ gắn với robot so với hệ trục tọa độ cố định). Yếu tố quan trọng nhất trong bài toán ước lượng trạng thái là xác định sự sai biệt giữa



hai vị trí  $\bar{x}_{t-1}$  và  $\bar{x}_t$  (được gọi là ‘vi sai’ vị trí), giá trị sai biệt này dùng để ước lượng khoảng cách thực  $x_{t-1}$  và  $x_t$ .

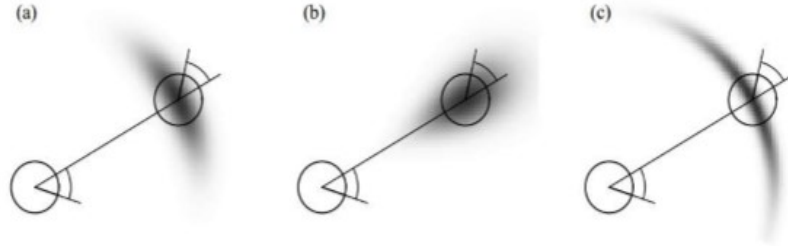


Hình 2.4. Mô hình đo lường của robot trong khoảng thời gian  $(t-1, t]$ .

Giải thuật trình bày trong Bảng 2.1 được dùng để tính giá trị của  $p(x_t | u_t, x_{t-1})$  với giá trị là trạng thái của robot tại thời điểm  $(t-1)$  là  $x_{t-1} = (x \ y \ \theta)^T$  và trạng thái tiếp theo  $x_t = (x' \ y' \ \theta')^T$ . Chuyển động của robot được diễn tả bởi  $u_t$  với  $u_t = (\bar{x}_{t-1} \ \bar{x}_t)^T$ . Chuyển động  $u_t$  được tách 3 bước liên tiếp: 1 chuyển động quay, 1 chuyển động thẳng, 1 chuyển động quay nữa. Hình 2.5 mô tả khai triển này: chuyển động đầu tiên gọi là chuyển động quay  $\delta_{rot1}$ , chuyển động tịnh tiến  $\delta_{trans}$  và chuyển động quay thứ hai  $\delta_{rot2}$ . Mỗi cặp vị trí  $(\bar{s} \ \bar{s}')$  (chuyển động từ  $\bar{s}$  tới  $\bar{s}'$ ) được diễn tả bằng vector với thông số thứ nhất  $(\delta_{rot1} \ \delta_{trans} \ \delta_{rot2})^T$  và những thông số này đủ để thiết lập quan hệ tương đối giữa  $\bar{s}$  và  $\bar{s}'$ . Mô hình chuyển động khảo sát ở đây xét trong trường hợp ba thông số trên sẽ bị sai do nhiều yếu tố.

Bảng 2.1. Giải thuật *motion\_model\_odometry*

- 1: Giải thuật **motion\_model\_odometry** ( $x_t, u_t, x_{t-1}$ )
- 2:  $\delta_{rot1} = a \tan 2(\bar{y}' - \bar{y}, \bar{x}' - \bar{x}) - \bar{\theta}$
- 3:  $\delta_{trans} = \sqrt{(\bar{x} - \bar{x}')^2 + (\bar{y} - \bar{y}')^2}$
- 4:  $\delta_{rot2} = \bar{\theta}' - \bar{\theta} - \delta_{rot1}$
- 5:  $\hat{\delta}_{rot1} = a \tan 2(y' - y, x' - x) - \theta$
- 6:  $\hat{\delta}_{trans} = \sqrt{(\bar{x} - \bar{x}')^2 + (\bar{y} - \bar{y}')^2}$
- 7:  $\delta_{rot2} = \theta' - \theta - \hat{\delta}_{rot1}$
- 8:  $p_1 = \mathbf{prob}(\delta_{rot1} - \hat{\delta}_{rot1}, \alpha_1 \hat{\delta}_{rot1} + \alpha_2 \hat{\delta}_{trans})$
- 9:  $p_2 = \mathbf{prob}(\delta_{trans} - \hat{\delta}_{trans}, \alpha_3 \hat{\delta}_{trans} + \alpha_4 (\hat{\delta}_{rot1} + \hat{\delta}_{rot2}))$
- 10:  $p_3 = \mathbf{prob}(\delta_{rot2} - \hat{\delta}_{rot2}, \alpha_1 \hat{\delta}_{rot2} + \alpha_2 \hat{\delta}_{trans})$
- 11: **return**( $p_1, p_2, p_3$ )



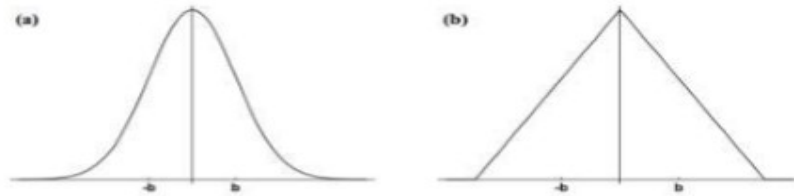
Hình 2.5. Mô hình theo đo lường với các thiết lập thông số nhiễu khác nhau.

Bước 2 đến bước 4 trong giải thuật cho phép tìm được thông số của chuyển động tương đối  $(\delta_{rot1} \quad \delta_{trans} \quad \delta_{rot2})^T$  từ các giá trị đo lường thu được, thông qua bài toán động học ngược. Bước 5 đến bước 7 tính các thông số chuyển động tương đối  $(\hat{\delta}_{rot1} \quad \hat{\delta}_{trans} \quad \hat{\delta}_{rot2})^T$ . Bước 8 đến bước 10 tính xác suất sai số cho các thông số chuyển động. Hàm phân bố xác suất  $prob(x, b)$  diễn tả sai số của chuyển động, hàm này tính giá trị xác suất xảy ra trạng thái  $x$  tuân theo quy luật phân phối với giá trị kỳ vọng (tham số vị trí, hay cũng chính là giá trị trung bình) bằng 0 (zero-centered) và phương sai  $b$ . Bảng 2.2 trình bày hai giải thuật có thể ứng dụng để tính hàm phân phối này, với biến sai số lần lượt

tính theo hàm phân phối chuẩn (phân phối Gaussian) và hàm phân phối tam giác. Lưu ý khi dùng phép tính này là tập xác định của giá trị vi sai góc là khoảng  $[-\pi \ \pi]$ . Do đó kết quả của  $\delta_{rot2} - \hat{\delta}_{rot2}$  phải được chọn lọc lại cẩn thận. Cuối cùng, bước 11 trả về xác suất sai số tổng hợp, cách nhân các xác suất sai số riêng lẻ  $p_1$ ,  $p_2$  và  $p_3$ . Hệ số từ  $\alpha_1$  đến  $\alpha_4$  là các thông số đặc trưng cho nhiều trong chuyển động của robot.

Bảng 2.2. Giải thuật *prob\_normal\_distribution* và giải thuật *prob\_triangular\_distribution*

1:	Giải thuật <b>prob_normal_distribution</b> (a, b):
	$\text{return } \frac{1}{\sqrt{2\pi b}} e^{-\frac{1a^2}{2b}}$
2:	Giải thuật <b>prob_triangular_distribution</b> (a, b):
	if $ a  > \sqrt{6b}$
	return 0
	else
	$\text{return } \frac{\sqrt{6b} -  a }{6b}$



Hình 2.6. Phân bố xác suất của biến ngẫu nhiên  $a$ , phương sai  $b$ , kì vọng bằng 0

### **Bộ lọc Particle**

Một khái niệm quan trọng cần giới thiệu trong mô hình động học xác suất đó là trong tính toán thực tiễn, ta sẽ sử dụng kỹ thuật của bộ lọc *particle* để mô phỏng các phân bố xác suất liên tục. Kỹ thuật particle còn được gọi là phương pháp Monte Carlo[12]. Ý tưởng chính của bộ lọc particle là ta sẽ đại diện một phân bố xác suất hậu nghiệm  $bel(x_t)$  bởi một bộ các mẫu ngẫu nhiên rút ra từ

phân bố này. Trong các bộ lọc *particle*, các mẫu của một phân bố xác suất hậu nghiệm được gọi là các particle và được biểu thị như sau:

$$\bar{\chi}_t := x_t^{[1]}, x_t^{[2]}, \dots, x_t^{[M]} \text{ với } M \text{ là số mẫu trong bộ } \bar{\chi}_t.$$

Xác suất hậu nghiệm  $bel(x_t) = p(x_t | u_t, x_{t-1})$  liên tục theo giải thuật bảng 2.3, ta có thể diễn đạt đơn giản là  $\bar{\chi}_t \approx bel(x_t)$ . Ta ứng dụng kỹ thuật particle để tìm giải thuật lấy mẫu ngẫu nhiên từ phân phối  $p(x_t | u_t, x_{t-1})$  theo mô hình đo lường như sau:

Giải thuật lấy mẫu ngẫu nhiên theo quy luật phân phối chuẩn (xấp xỉ) và tam giác với kỳ vọng 0 và phương sai  $b$ .

*Bảng 2.3. Giải thuật sample\_normal\_distribution và giải thuật sample\_triangular\_distribution*

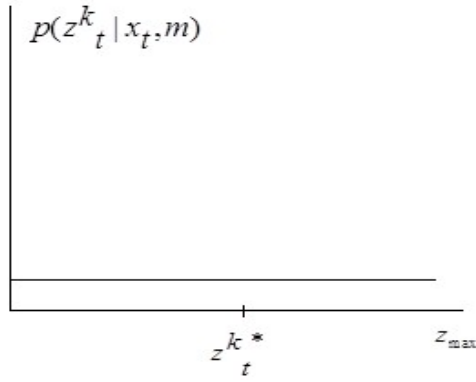
- 1: Giải thuật **sample\_normal\_distribution** (b):
- 2:      $return \frac{b}{6} \sum_{i=1}^{12} \mathbf{rand}(-1,1)$
- 3: Giải thuật **sample\_triangular\_distribution** (b)
- 4:      $return b.\mathbf{rand}(-1,1).\mathbf{rand}(-1,1)$

*Bảng 2.4. Giải thuật sample\_motion\_model\_đo lường*

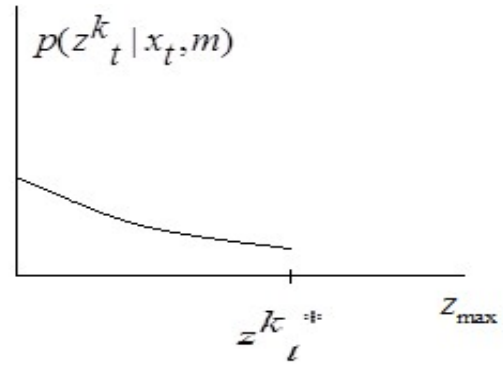
- 1: Giải thuật **sample\_motion\_model\_odometry** ( $u_t, x_{t-1}$ )
- 2:  $\delta_{rot1} = a \tan 2(\bar{y}' - \bar{y}, \bar{x}' - \bar{x}) - \bar{\theta}$
- 3:  $\delta_{trans} = \sqrt{(\bar{x} - \bar{x}')^2 + (\bar{y} - \bar{y}')^2}$
- 4:  $\delta_{rot2} = \bar{\theta}' - \theta - \delta_{rot1}$
- 5:  $\hat{\delta}_{rot1} = \delta_{rot1} - \mathbf{sample}(\alpha_1 \delta_{rot1} + \alpha_2 \delta_{rot2})$
- 6:  $\hat{\delta}_{tran} = \delta_{tran} - \mathbf{sample}(\alpha_3 \delta_{tran} + \alpha_4 (\delta_{rot1} + \delta_{rot2}))$
- 7:  $\delta_{rot2} = \delta_{rot2} - \mathbf{sample}(\alpha_1 \delta_{rot1} + \alpha_2 \delta_{trans})$
- 8:  $x' = x + \hat{\delta}_{trans} \cos(\theta + \hat{\delta}_{rot1})$
- 9:  $y' = y + \hat{\delta}_{trans} \sin(\theta + \hat{\delta}_{rot1})$
- 10:  $return x_t = (x', y', \theta')^T$

### Mô hình xác suất của phép đo

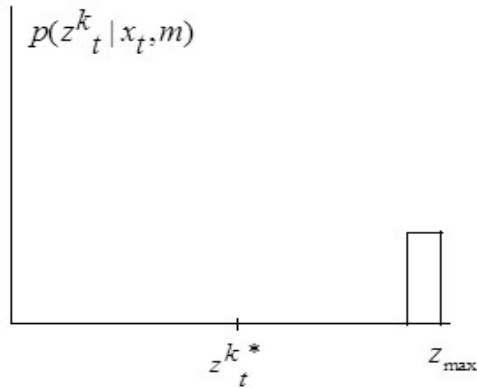
Mô hình xác suất của một phép đo, biểu thị bằng  $p(z_t | x_t, m)$  là xác suất có được một phép đo khi biết trước bản đồ và tọa độ của robot. Đối với đa số cảm biến được ứng dụng cho robot tự hành trong nhà, khoảng cách có thể được đo dọc theo một chùm tia vật lý (ví dụ như tia sáng hoặc tia siêu âm), ta có thể biểu thị  $z_t = \{z_t^1, z_t^2, \dots, z_t^K\}$  với  $z_t^k (1 \leq k \leq K)$  là giá trị đo thu được từ một chùm tia. Ngoài ra, do trong mô hình ta bắt đầu xét tới bản đồ nên ta cũng biểu thị  $m = \{m_1, m_2, \dots, m_N\}$  với  $m_n (1 \leq n \leq N)$  có một giá trị biểu thị mức độ chắc chắn một điểm có bị chiếm chỗ trong không gian. Ta sẽ quan tâm tới việc đi tìm mô hình xác suất của loại cảm biến khoảng cách bằng chùm tia [13].



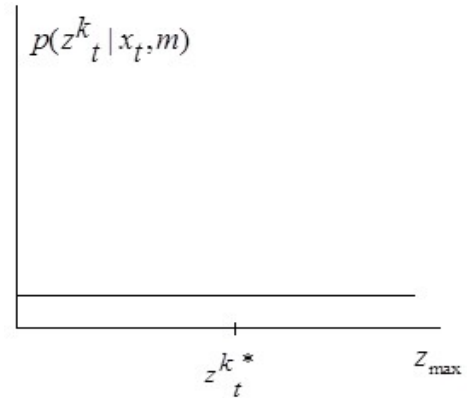
a) Gaussian distribution  $p_{hit}$



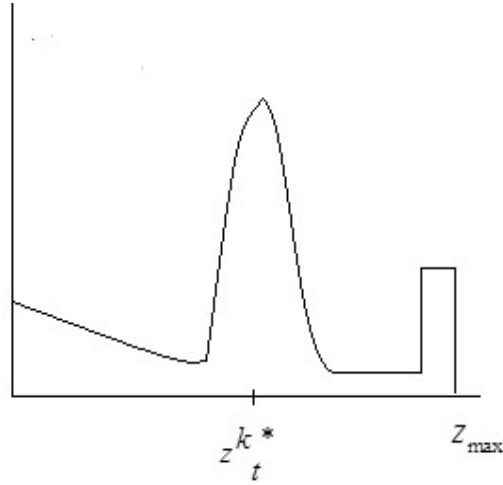
b) Gaussian distribution  $p_{short}$



c) Gaussian distribution  $p_{max}$



d) Gaussian distribution  $p_{rand}$



Hình 2.7. Mô hình xác suất của phép đo từ cảm biến khoảng cách chùm tia

Trong Hình 2.7, mô hình xác suất của một chùm tia tại tọa độ  $x_t$  trong  $m$  được mô tả với các thành phần  $p_{hit}, p_{short}, p_{max}, p_{rand}$ . Giả sử  $z_t^{k*}$  là khoảng cách thực từ cảm biến tới vật thể, ta có  $p_{hit}$  là phép đo chính xác với các nhiễu đo lường xung quanh  $z_t^{k*}$ ,  $p_{short}$  là nhiễu do các vật thể không biết trước,  $p_{max}$  là nhiễu cảm biến bị bão hòa và  $p_{rand}$  là nhiễu phân bố đều trên toàn dải đo. Mô hình xác suất tổng quát của phép đo là tích của cả bốn thành phần này. Giải thuật `beam_range_finder_model` sau cho ta khả năng của một phép đo  $z_t$ , khi biết tọa độ  $x_t$  và bản đồ  $m$ .

Bảng 2.5. Giải thuật `beam_range_finder_model`

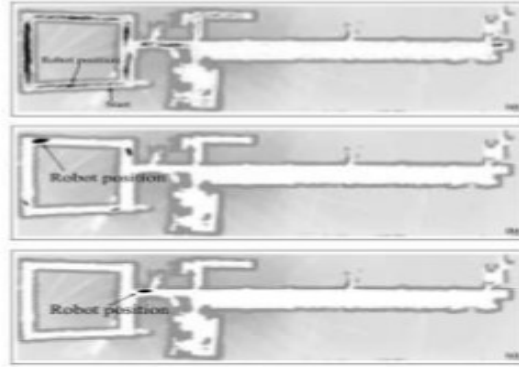
1:	Giải thuật <b>beam_range_finder_model</b> ( $z_t, x_t, m$ )
2:	$q = 1$
3:	for $q = 1$ to $K$ do
4:	tính $z_t^{k*}$ với phép đo sử dụng $z_t^k$ dùng tia dò
5:	$p = (z_{hit} \cdot p_{hit} + z_{short} \cdot p_{short} + z_{max} \cdot p_{max} + z_{rand} \cdot p_{rand}) (z_t^k   x_t, m)$
6:	$q = q \cdot p$
7:	return $q$

Vòng lặp ngoài cùng từ hàng 2 - 7 nhân tất cả khả năng của các chùm cảm biến  $z_t^k$  áp dụng xấp xỉ  $p(z_t | x_t, m) = \prod_{k=1}^K p(z_t^k | x_t, m)$ . Hàng 5 sử dụng mô

hình xác suất của từng chùm tia để tìm khả năng của từng phép đo. Các thông số  $p_{hit}, p_{short}, p_{max}, p_{rand}$  là đặc trưng của từng thành phần trong mô hình xác suất của  $p(z_t^k | x_t, m)$  với điều kiện  $z_{hit} + z_{short} + z_{max} + z_{rand} = 1$ .

*Phương pháp định vị Monte Carlo thích nghi – Adaptive Monte Carlo Localization[11]*

Phương pháp định vị Monte Carlo (MCL) là một giải thuật ước lượng tọa độ của robot từ các thông tin đo lường, cảm biến, bản đồ. Tính từ thích nghi trong phương pháp trên đề cập tới một biến thể của MCL. Bảng 2.6 trình bày đồng thời giải thuật MCL và biến thể của nó có màu đỏ. Phần biến thể của MCL được áp dụng trên dự đoán với cảm biến  $w_t^{[m]}$  để khắc phục trường hợp Robot rơi vào các vị trí có độ phân bố xác suất bằng nhau (kidnapping) được minh họa trên Hình 2.8.



Hình 2.8. Minh họa trường hợp kidnapping - tại thời điểm thứ hai khi robot dự đoán những vị trí ở cả bốn góc của hành lang.

Bảng 2.6. Giải thuật Augmented\_MCL

1: Giải thuật <b>Augmented_MCL</b> $x_{t-1}, u_t, z_t, m$ 2: static $\mathcal{G}_{slow}, \mathcal{G}_{fast}$ 3: $\bar{\chi}_t = \chi_{t-1} = \emptyset$ 4: for m=1 to M do 5: $x_t^{[m]} = \text{sample\_motion\_model}(u_t, x_{t-1}^{[m]})$ 6: $\mathcal{G}_t^{[m]} = \text{measurement\_model}(z_t, x_{t-1}^{[m]}, m)$ 7: $\bar{\chi}_t = \bar{\chi}_t + (x_t^{[m]}, \mathcal{G}_t^{[m]})$
--

```

8:    $\mathcal{G}_{avg} = \mathcal{G}_{avg} + \frac{1}{M} \mathcal{G}_t^{[m]}$ 
9: end for
10:  $\mathcal{G}_{slow} = \mathcal{G}_{slow} + \alpha_{slow} (\mathcal{G}_{avg} - \mathcal{G}_{slow})$ 
11:  $\mathcal{G}_{fast} = \mathcal{G}_{fast} + \alpha_{fast} (\mathcal{G}_{avg} - \mathcal{G}_{fast})$ 
12: for m = 1 to M do
13:   with probabitily  $\max (0.0, 1.0 - \mathcal{G}_{fast} / \mathcal{G}_{slow})$  do
14:     add random pose to  $\mathcal{G}_t$ 
15:   end
16:   draw  $i \in \{1, \dots, N\}$  with probabitily  $\alpha \mathcal{G}_t^{[i]}$ 
17:   add  $x_t^{[i]}$  to  $\chi_t$ 
18: endwith
19: endfor
20: return  $\chi_t$ 

```

Trong giải thuật MCL ở trên, *sample\_motion\_model* và *measurement model* có thể là các mô hình chuyển động xác suất từ thông tin đo lường và cảm biến như *Bảng 2.3* và *Bảng 2.5* đã khảo sát. Hàng số 4 trong giải thuật MCL sẽ lấy mẫu từ mô hình chuyển động sử dụng các mẫu để đại diện cảm tính của robot. Sau đó phép đo từ cảm biến sẽ được áp dụng vào dòng 5 để xác định trọng số của mẫu. Cảm biến ban đầu của robot  $bel(x_0)$  có thể được giả sử bằng một phân bố đều trên toàn không gian với trọng số ban đầu bằng nhau là  $M^{-1}$ . Các thông số  $\alpha_{slow}$ , và  $\alpha_{fast}$  được chọn với yêu cầu  $0 \leq \alpha_{slow} \leq \alpha_{fast}$ .

## 2.3 Thuật toán lập kế hoạch di chuyển cho robot

Một robot di động thông minh phải được điều khiển dẫn đường theo một chiến lược có hiệu quả. Có nhiều nghiên cứu trên thế giới với các giải thuật và các phương pháp khác nhau cho dẫn đường robot trong các môi trường trong nhà.

### 2.3.1 Thuật toán Dijkstra

Thuật toán của Dijkstra là một thuật toán để tìm các đường dẫn ngắn nhất giữa các nút trong một biểu đồ, ví dụ như các mạng lưới đường bộ.



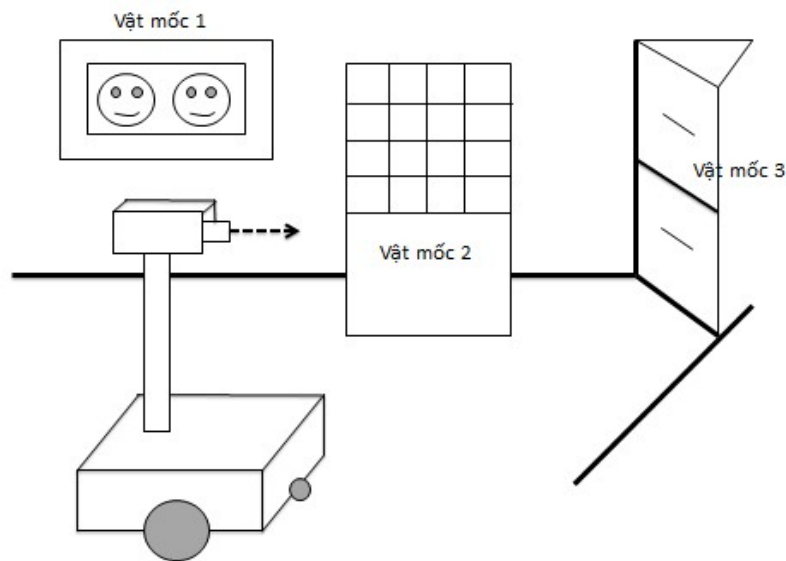
Nút mà chúng ta bắt đầu được gọi là nút ban đầu. Khoảng cách của nút Y là khoảng cách từ nút ban đầu đến Y. Thuật toán của Dijkstra sẽ chỉ định một số giá trị khoảng cách ban đầu và sẽ cố gắng cải tiến chúng từng bước.

1. Gán cho mỗi nút một giá trị khoảng cách: đặt về 0 cho nút ban đầu và đến vô cùng cho tất cả các nút khác.
2. Đặt nút ban đầu là hiện tại, tạo một tập hợp chứa tất cả các nút chưa được biết.
3. Đối với nút hiện tại, xem xét tất cả các nút lân cận và tính khoảng cách dự kiến của chúng. So sánh khoảng cách dự kiến mới được tính toán với giá trị được chỉ định hiện tại và sẽ lấy một giá trị nhỏ hơn. Ví dụ, nếu nút A hiện tại được đánh dấu bằng khoảng cách 6, và cạnh kết nối với một lân cận B có chiều dài 2, thì khoảng cách đến B (qua A) sẽ là  $6+2 = 8$ . Nếu B trước đó đánh dấu với khoảng cách lớn hơn 8 thì n sẽ lấy 8. Ngược lại, nó giữ giá trị hiện tại.
4. Khi chúng ta xem xét tất cả các lân cận của nút hiện tại, đánh dấu nó là đã truy cập và loại bỏ nó khỏi tập hợp chưa biết.
5. Nếu nút đích đã được đánh dấu truy cập (khi lập kế hoạch một đường giữa hai nút cụ thể) hoặc nếu khoảng cách dự kiến nhỏ nhất giữa các nút trong tập hợp biết là vô cùng (khi lập kế hoạch một đường đi hoàn chỉnh, xảy ra khi không có kết nối giữa nút ban đầu và các nút còn lại chưa thấy), sau đó dừng lại. Thuật toán đã kết thúc.
6. Nếu không, chọn nút chưa được đánh dấu bằng cách dự kiến khoảng cách nhỏ nhất, thiết lập nó như là nút hiện tại “mới” và quay trở lại bước 3.

### 2.3.2 Hệ thống dẫn đường cột mốc chủ động

Hệ thống dẫn đường cột mốc chủ động là hệ thống dẫn đường được sử dụng phổ biến nhất trên tàu biển và máy bay. Hệ thống này cung cấp thông tin vị trí rất chính xác với quá trình xử lý tối thiểu. Hệ thống cho phép tốc độ lấy mẫu và độ tin cậy cao nhưng đi kèm với nó là giá thành cao trong việc thiết lập và

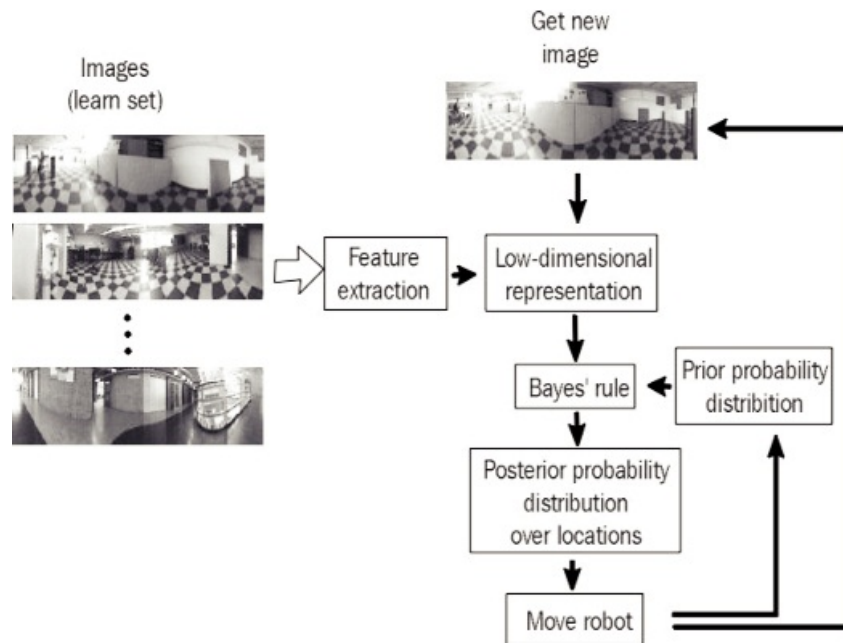
duy trì. Cột mốc được đặt tại các vị trí chính xác sẽ cho phép xác định tọa độ chính xác của vật thể. Có 2 phương pháp đo dùng trong hệ thống cột mốc chủ động, đó là phép đo 3 cạnh tam giác và phép đo 3 góc tam giác. Phép đo 3 cạnh tam giác xác định vị trí vật thể dựa trên khoảng cách đo được tới cột mốc biết trước. Trong hệ thống dẫn đường sử dụng phép đo này thông thường có ít nhất là 2 trạm phát đặt tại các vị trí biết trước ngoài môi trường và 1 trạm nhận đặt trên robot. Hoặc ngược lại có 1 trạm phát đặt trên robot và các trạm nhận đặt ngoài môi trường. Sử dụng thông tin về thời gian truyền của chùm tia, hệ thống sẽ tính toán khoảng cách giữa các trạm phát cố định và trạm nhận đặt trên robot. GPS - hệ thống định vị toàn cầu hoặc hệ thống cột mốc sử dụng cảm biến siêu âm là các ví dụ khi sử dụng phép đo 3 cạnh tam giác, phép đo 3 góc tam giác[14].



Hình 2.9. Định vị sử dụng vật mốc

### 2.3.3 Định hướng sử dụng bản đồ

Robot sử dụng các cảm biến được trang bị để tạo ra một bản đồ cục bộ môi trường xung quanh. Bản đồ này sau đó so sánh với bản đồ toàn cục lưu trữ sẵn trong bộ nhớ. Nếu tương ứng, robot sẽ tính toán vị trí và góc hướng thực tế của nó trong môi trường[14].



Hình 2.10. Định vị và dẫn đường cho Robot chuyển động sử dụng bản đồ

Để nâng cao độ chính xác của việc định vị thì một giải pháp tối ưu hiện nay là kết hợp hai hoặc nhiều phương pháp định vị nêu trên.

## CHƯƠNG 3: THIẾT KẾ CƠ KHÍ VÀ TÍCH HỢP PHẦN ĐIỆN – ĐIỆN TỬ ROBOT DI ĐỘNG

### 3.1 Các yêu cầu về mô hình Robot

#### 3.1.1 Yêu cầu về cơ khí

- Vận tốc di lớn nhất của Robot  $V_{\max} = 0.5 \text{ m/s}$ ;
- Gia tốc lớn nhất của Robot  $a_{\max} = 1 \text{ m/s}^2$ ;
- Khối lượng Robot dưới 4kg;
- Thiết kế một mô hình khung cơ khí Robot và khung giá camera;
- Kích thước Robot dài×rộng×cao : 300×280×380mm;
- Trên mô hình bố trí hệ thống điện-điện tử gồm các phần tử: mạch điều khiển trung tâm, động cơ, driver điều khiển động cơ, Bộ pin nguồn, mạch ổn áp và công tắc nguồn, máy quét laser. Yêu cầu bố trí hệ thống điện điện tử hợp lý, tránh chập cháy, không bị ảnh hưởng bởi rung xóc.

#### 3.1.2 Yêu cầu về phần điện-điện tử

Các phần tử điện-điện tử cần thiết trong đồ án như sau:

- Một máy chủ (đặt tại trạm) và một máy tính nhúng đặt trên mô hình Robot;
- Vi điều khiển để nhận và xử lý tín hiệu điều khiển động cơ;
- Các động cơ dẫn động cho Robot;
- Driver điều khiển động cơ;
- Nguồn cấp cho toàn bộ hệ thống Robot.

#### 3.1.3 Yêu cầu về hệ thống cảm biến

Thực hiện bài toán định vị, điều hướng và lập kế hoạch ta cần các cảm biến:

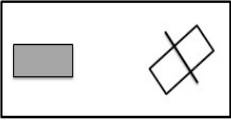
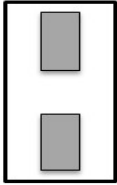
- Cảm biến khoảng cách để đo khoảng cách của Robot với vật cản;
- Encoder để đo tốc độ động cơ.

### 3.2 Lựa chọn mô hình

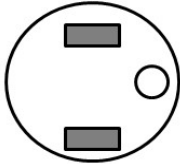
#### 3.2.1 Lựa chọn mô hình cơ khí

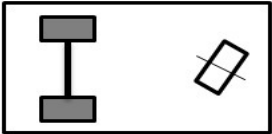
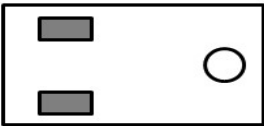
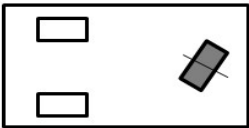
Robot tự hành có 2 dạng phổ biến là bánh xích và lốp. Bánh xích thường sử dụng với các robot ngoài trời và robot vượt địa hình khó khăn, nhiều trở ngại. Bánh lốp thường sử dụng cho các robot hoạt động trong nhà và các robot hoạt động với địa hình bằng phẳng. Robot trong đề án hoạt động ở môi trường trong phòng nên chúng ta sẽ sử dụng Robot dạng bánh lốp. Robot dạng bánh lốp có các dạng cơ bản như sau:

*Bảng 3.1. Dạng robot tự hành dạng 2 bánh xe, trích trang 18 [15]*

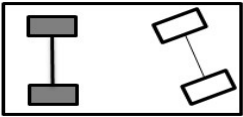
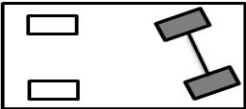
STT	Cấu tạo	Ưu nhược điểm và ứng dụng
1	 <p>Bánh dẫn động sau và bánh lái trước</p>	<p>+Ưu điểm: Lực ma sát với mặt phẳng chuyển động nhỏ, dễ thay đổi hướng chuyển động.</p> <p>+Nhược điểm: Giữ thẳng bằng khó, vận tốc chuyển động không đủ lớn để gây nghiêng đổ.</p> <p>+Ứng dụng: Sử dụng với xe đạp, xe máy hoặc robot cá nhân Cye.</p>
2	 <p>Hai bánh dẫn động hai bên</p>	

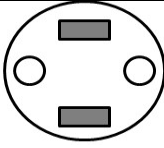
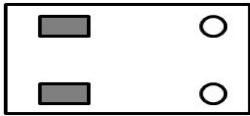
*Bảng 3.2. Các dạng robot di động ba bánh xe, trích trang 19 [15]*

STT	Cấu tạo	Ưu, nhược điểm và ứng dụng
1	 <p>hai bánh truyền động hai bên, bánh tự lùa phía trước</p>	<p>+ Ưu điểm: Khả năng giữ thẳng bằng tốt, điều khiển hướng linh hoạt.</p> <p>+ Nhược điểm: Chịu tải trọng kém. Khi chuyển động với tốc độ lớn thay đổi hướng dễ bị lật.</p>

2	 <p>Hai bánh chủ động đồng trục phía sau, bánh lái phía trước</p>	<p>+ Ứng dụng: Thích hợp với các loại xe chuyển động với tốc độ vừa và nhỏ, vượt địa hình không quá dốc như: Robot trong nhà, xe tải nhỏ. Cần linh hoạt thay đổi hướng di chuyển.</p>
3	 <p>Hai bánh chủ động độc lập phía sau, bánh lái phía trước</p>	
4	 <p>Bánh chủ động phía trước, bánh tự lựa phía sau</p>	

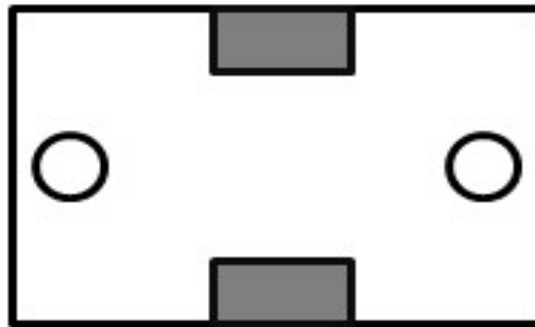
Bảng 3.3. Một số dạng robot di động bốn bánh xe, trích trang 20 [15]

STT	Cấu tạo	Ưu, nhược điểm và ứng dụng
1	 <p>Hai bánh chủ động đồng trục phía sau, hai bánh lái phía trước</p>	<p>+ Ưu điểm: Cấu hình 4 bánh là ổn định nhất. Dễ dàng di chuyển và thay đổi hướng với tốc độ lớn. Chịu tải trọng lớn và khả năng giữ thăng bằng tốt.</p> <p>+ Nhược điểm: 4 bánh xe sẽ khó cho việc các bánh cùng nằm trên một mặt phẳng.</p>
2	 <p>Hai bánh chủ động lái phía trước, hai bánh bị động độc lập phía sau</p>	

3	 <p>Hai bánh chủ động độc lập 2 bên, một bánh tự lựa phía trước, một tự lựa phía sau.</p>	+ Ứng dụng: Dạng mô hình 4 bánh được sử dụng rộng rãi cho các dạng xe tải cỡ vừa và lớn, các mô hình chuyển động ở địa hình dốc.
4	 <p>Hai bánh chủ động độc lập phía sau, hai bánh tự lựa độc lập phía trước</p>	

Từ những yêu cầu của hệ thống cơ khí của mô hình Robot đã được trình bày ở phần 3.1.1. Từ yêu cầu, nhiệm vụ của robot trong Đồ án, đồ án của chúng tôi lựa chọn:

- Mô hình cơ khí số 3 trong *Bảng 3.3* như sau:



Mô hình Robot sẽ được bố trí hai bánh dẫn động gắn với 2 động cơ riêng biệt hai bên, một bánh tự lựa phía trước và một bánh tự lựa phía sau.

### 3.2.2 Lựa chọn thiết bị cho mô hình robot

#### a. Lựa chọn bánh xe

- Bánh xe dẫn động

Vì mô hình xe hoạt động trên địa hình bằng phẳng, nên cứng nhắc nên cần lựa chọn bánh xe có lớp cao su để tăng độ ma sát giữa bánh xe và nền. Từ mô hình khung xe đã thiết kế thì chọn loại bánh xe có kích thước phù hợp.

Bánh xe dẫn động được bố trí 2 bên và đối xứng qua tâm xe. Hai bánh này được dẫn động bởi hai động cơ thông qua hộp giảm tốc được tích hợp sẵn trên động cơ.

Trong đồ án sẽ lựa chọn bánh xe có sẵn trên thị trường:



*Hình 3.1. Bánh dẫn động của robot*

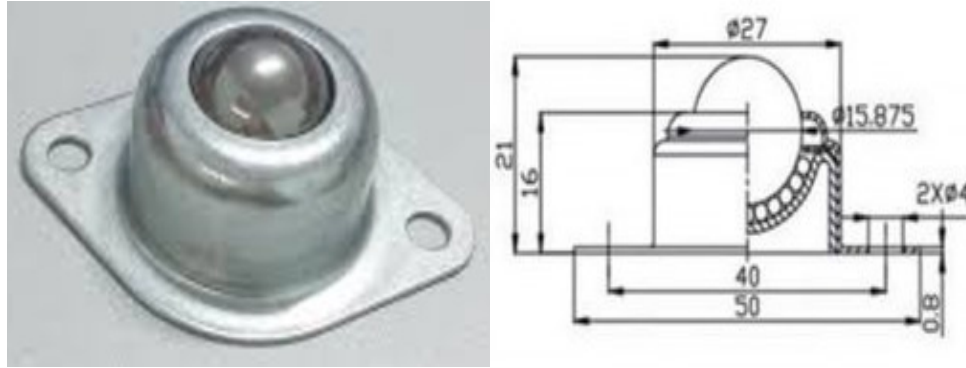
Đường kính ngoài của bánh dẫn động là 100mm, bề rộng 35mm, khối lượng 180g. Bánh xe có thể lắp trực tiếp với các loại động cơ giảm tốc có đường kính trục 6mm, tải trọng 50kg[19].

Cấu tạo của bánh gồm: vành bánh và lốp. Vành được làm bằng nhôm có độ bền cao. Lốp bánh bằng cao su bọc silicon mềm, bên trong không có săm, bên ngoài có nhiều rãnh tăng ma sát cho bánh. Hệ số ma sát lăn của lốp với bề mặt đường nhựa là  $\mu = 0,02$  [16]; hệ số ma sát trượt khoảng  $0,25 \div 0,8$  tùy chất lượng đường[16].

- Bánh tự lựa

Bánh tự lựa có hai loại bánh đa hướng và bánh bi cầu (bánh mắt trâu). Đối với môi trường nền cứng nhẵn và robot chuyển động với tốc độ không cao thì bánh bi cầu chuyển hướng dễ dàng hơn, linh hoạt hơn nên trong đồ án sẽ lựa chọn bánh bị động là bánh bi cầu. Hai bánh bi cầu bố trí một bánh phía trước và một bánh phía sau của Robot. Là loại bánh CY-15A, có kích thước dài 50mm, cao 21mm[19].





Hình 3.2. Bánh bi cầu

*b. Tính và lựa chọn động cơ*

Với yêu cầu đặt ra trên phần 3.1.1 và mô hình với bánh xe đã chọn ở phần 3.2.1 và 3.2.2a thì động cơ một chiều là lựa chọn tối ưu cho đề án.

Cần tính toán và lựa chọn động cơ phù hợp với loại bánh xe dẫn động đã chọn trong mục 3.2.1. Để tính toán lựa chọn các thông số động cơ thích hợp, cần tính toán được công suất tối thiểu mà động cơ cần. Do đó, cần biết các yêu cầu:

- Kết cấu động học của cơ cấu truyền động: loại truyền động (truyền động trực tiếp, đai truyền, truyền động bánh răng, xích...).
- Đặc điểm tải trọng: đặc điểm khối lượng, lực hoặc mô-men tải (không đổi, quy luật thay đổi, tải trọng lớn nhất), vị trí phân bố tải trọng.
- Yêu cầu động học: đặc điểm chuyển động theo thời gian (quỹ đạo, vận tốc, gia tốc).

Từ những yêu cầu như trên, thực hiện phân tích và tính toán công suất tối thiểu của động cơ. Trên cơ sở đó, tiến hành lựa chọn động cơ có công suất, kích thước và cách lắp đặt phù hợp. Tính chọn động cơ điện và hộp giảm tốc của bánh Robot.

Các bước thực hiện:

- Phân tích lực;
- Tính công suất động cơ;
- Chọn động cơ;
- Tính tỉ số truyền;

- Chọn hộp giảm tốc.

Thông số ban đầu của Robot:

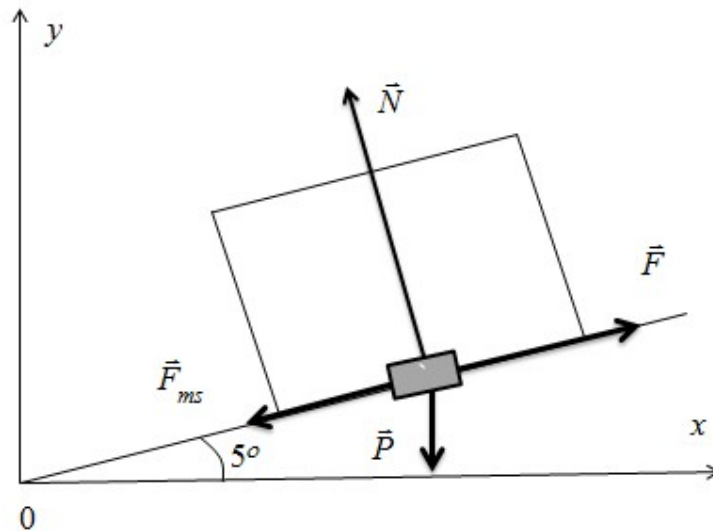
- R: bán kính bánh xe,  $R = 0,05\text{m}$ ;
- v: vận tốc di chuyển lớn nhất của Robot,  $v = 0,5\text{m/s}$ ;
- a: gia tốc lớn nhất của Robot,  $a = 1,0 \text{ m/s}^2$ ;
- m: khối lượng Robot, ước tính  $m = 4\text{kg}$ ;
- $F_{ms}$ : Lực ma sát giữa các bánh xe và mặt đường,  $F_{ms} = \mu N$ .

Với  $\mu$  là hệ số ma sát lăn phụ thuộc chất nền và chất liệu bánh robot, bánh robot bằng cao su và nền gạch, theo [16] ta có  $\mu = 0,02$ .

- Độ dốc tối đa của đường:  $\theta = 5^\circ$ .

#### ❖ Phân tích lực

- Coi toàn bộ khối lượng xe tập trung tại tâm đế xe;
- Chọn hệ quy chiếu Oxy như hình vẽ;
- Các lực tác dụng lên Robot: lực đẩy tạo bởi hai động cơ  $\vec{F}$ , trọng lực  $\vec{P}$ , tổng lực ma sát  $\vec{F}_{ms}$ , phản lực  $N$ .



Hình 3.3. Tổng hợp các lực tác dụng lên xe

#### ❖ Tính chọn công suất động cơ

Khi xe chuyển động lên dốc với gia tốc  $\vec{a}$ , theo định luật 2 Newton, ta được:

$$\vec{F} + \vec{P} + \vec{F}_{ms} + \vec{N} = m\vec{a} \quad (3.1)$$

Chiếu phương trình (3.1) lên mặt phẳng chuyển động, thu được:

$$F - P \sin \theta - F_{ms} = m.a \quad (3.2)$$

Trong đó,  $F_{ms} = \mu.N = \mu.m.g.\cos \theta$ . Với  $\mu = 0,02$ , là hệ số ma sát lăn. Từ phương trình (3.2), suy ra:

$$\begin{aligned} F &= P \sin \theta + F_{ms} + m.a \\ \Leftrightarrow F &= m(g.\sin \theta + \mu.g.\cos \theta + a) \end{aligned} \quad (3.3)$$

Từ phương trình (3.3), và do mỗi bánh xe phải chịu một nửa trọng lượng của toàn thân Robot suy ra mô men mà mỗi động cơ phải tạo ra trên trục bánh xe, để sinh ra lực đẩy F là:

$$M_t = \frac{1}{2}.F.R = \frac{1}{2}.m.R(g.\sin \theta + \mu.h.\cos \theta + a) \quad (3.4)$$

Thay giá trị các đại lượng vào trong phương trình (3.4) thu được:

$$M_t = \frac{1}{2}.F.R = \frac{1}{2}.3,5.0,05.(9,8.\sin 5^\circ + 0,02.9,8.\cos 5^\circ + 1) \approx 0,16(N.m)$$

Vậy tốc độ lớn nhất cần có ở trục bánh xe là:

$$\omega_t = \frac{v}{R} = \frac{0,5}{0,05} = 10(rad / s)$$

Công suất tối thiểu của động cơ (trích trang 44 [1]) là:

$$P = M_t.\omega_t$$

Trong thực tế có nhiều yếu tố như tải trọng động, ma sát trượt, ma sát, sai số lắp đặt. Để đảm bảo an toàn trong quá trình hoạt động, cần chọn động cơ có công suất lớn hơn với hệ số an toàn  $k = 1,5 \div 3$ . Do đó, động cơ được chọn phải có công suất tối thiểu là:

$$P = k.M_t.\omega_t \quad (3.5)$$

Chọn  $k = 3$  và thay vào các giá trị của các đại lượng vào biểu thức (3.5), ta có:

$$P = k.M_t.\omega_t = 3.0,16.10 = 4,8 \text{ (W)}$$

### ❖ Chọn động cơ

Với các thông số vừa tính toán, ta tiến hành lựa chọn động cơ từ các mẫu tiêu chuẩn có sẵn trên thị trường. Vì không yêu cầu tốc độ cao và momen lớn mà chỉ cần độ chính xác cao và đáp ứng tốc độ nhanh nên động cơ chọn cho đề án là động cơ một chiều Faulhaber-12VDC có hộp giảm tốc và có sẵn encoder[19].

Vì các động cơ điện 1 chiều có vận tốc quay lớn và mô men nhỏ nên không thể sử dụng để dẫn động trực tiếp cho bánh Robot, cần phải sử dụng hộp giảm tốc để tăng mô men và giảm tốc độ quay về mức phù hợp với đặc tính động học, tải trọng của robot.



Hình 3.4. Động cơ DC servo

Bảng 3.4. Thông số kỹ thuật của động cơ DC servo Faulhaber-12V

Điện áp (V)	12
Vận tốc góc (vòng/phút)	8100
Mô men xoắn (mN.m)	27
Dòng tải tối đa (mA)	1400
Dòng không tải (mA)	74
Chiều dài lớn nhất (mm)	120
Chiều dài động cơ (mm)	85
Đường kính ngoài (mm)	30

### c. Máy quét laser

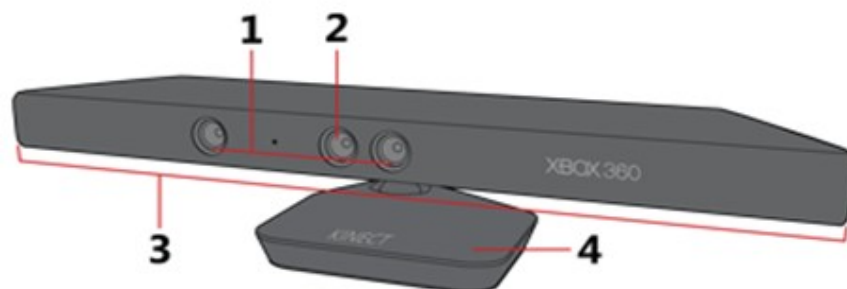
Để mobile robot xây dựng bản đồ, định vị và điều hướng cảm biến khoảng cách là một phần không thể thiếu. Máy quét laser là lựa chọn tối ưu cho cảm biến khoảng cách với robot tự hành hiện đại. Máy quét laser giúp robot nhận diện được vật cản xung quanh vị trí hoạt động. Nhưng do giá thành một

máy quét laser khá đắt và robot chỉ hoạt động ở trong phòng nên trong đồ án chúng ta sẽ sử dụng camera Kinect của Microsoft có thể đáp ứng được yêu cầu để làm một máy quét laser giả[20].



*Hình 3.5. Camera Kinect Xbox 360*

Kinect là sản phẩm của Microsoft dựa trên công nghệ camera xử lý chùm sáng có cấu trúc (structured-light imaging) được phát triển bởi PrimeSense, được tung ra lần đầu tiên vào tháng 11 năm 2010. Kinect được coi là một thiết bị ngoại vi cho Xbox 360, cho phép giao tiếp với con người thông qua cử chỉ. Khả năng giao tiếp giữa Kinect với con người dựa trên hai đặc tính: thông tin về độ sâu (depth map), khả năng phát hiện và bám theo đặc tính cơ thể người (body skeletontracking).



*Hình 3.6. Những thành phần chính của Kinect*

Những thành phần chính của Kinect theo Hình 3.5.

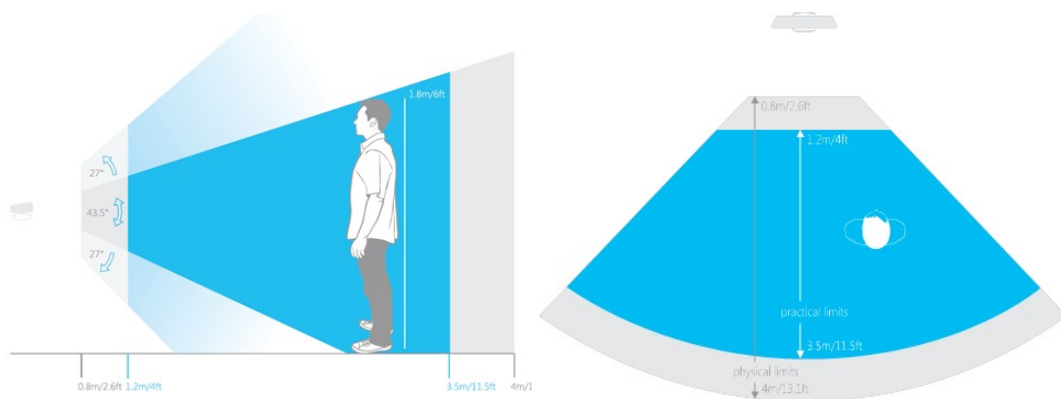
(1) Cảm biến độ sâu: độ sâu được thu về nhờ sự kết hợp của hai cảm biến: đèn chiếu hồng ngoại (IR Projector) và camera hồng ngoại (IR camera).

(2) RGB camera: là một camera có độ phân giải VGA (640x480), tốc độ tối đa 30 fps.

(3) Động cơ điều khiển góc ngẩng: là loại động cơ DC nhỏ, cho phép điều chỉnh thay đổi góc nhìn của camera, từ đó có thể tập trung vùng nhìn thấy của Kinect vào khu vực quan tâm.

(4) Dây microphone: gồm bốn microphone được bố trí dọc Kinect, dành cho các ứng dụng điều khiển bằng giọng nói.

Luồng video RGB mặc định của Kinect sử dụng độ phân giải VGA (640x480 pixel), tuy nhiên phần cứng thiết bị có thể cung cấp video có độ phân giải lên đến 1280x1024 ở tốc độ thấp hơn. Giá trị độ sâu của mỗi điểm ảnh có độ phân giải 11 bit. Kinect có vùng nhìn thấy thực tế trong khoảng 0.8 – 4m trong chế độ mặc định và 0.4 – 3m trong chế độ gần. Cảm biến có góc nhìn trong khoảng  $57^\circ$  theo chiều ngang và  $43.5^\circ$  theo chiều dọc cùng với khả năng của motor trên để có thể điều khiển góc ngẩng của camera trong khoảng  $27^\circ$ . Lấy vùng nhìn thấy thực tế khoảng 0.8m từ camera, ta có khung ảnh có kích thước thực tế là  $0.87 \times 0.63$ , với độ phân giải VGA ta có kích thước mỗi pixel biểu diễn một diện tích chỉ  $1.8 \text{ mm}^2$ , cho thấy độ chính xác hơn của Kinect. Trong ứng dụng robot tự hành trong nhà, robot có thể phát hiện được các vật thể cao hơn 30cm trong khoảng cách 0.6m.

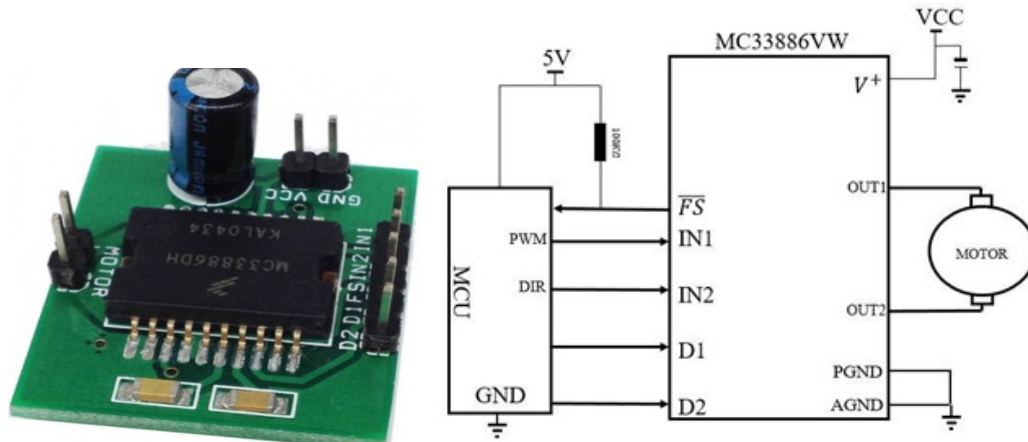


Hình 3.7. Vùng nhìn thấy của camera khoảng cách Kinect

Kinect có một hạn chế ở phần nguồn nuôi do không thể lấy trực tiếp nguồn từ cổng USB máy tính mà phải thông qua một adaptor AC sang DC. Để hoạt động trên robot, ta phải thiết kế một nguồn ổn áp DC 12V 1A. Đầu kết nối của Kinect có thiết kế duy nhất nên phải cắt lấy đầu này trên adaptor và nối với nguồn DC của robot.

#### d. Mạch driver

Để điều khiển động cơ DC Servo đã chọn trong phần 3.2.2b thì mạch cầu H là giải pháp tối ưu nhất. Động cơ được chọn có dòng cực đại là 1400mA nên mạch cầu H phải có dòng cực đại  $>1400\text{mA}$ . Ta sẽ chọn như sau[19]:

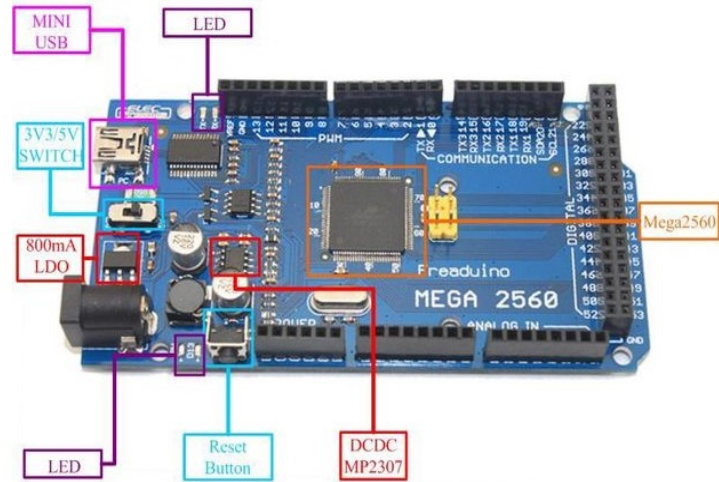


Hình 3.8. Mạch cầu H và sơ đồ kết nối với vi điều khiển

- IC điều khiển: MC33886VW
- Điện áp hoạt động: 5-40VDC
- Dòng điều tối đa: 5A
- Điện áp điều khiển: 5V chuẩn TTL gồm 1 tín hiệu điều khiển tốc độ và 1 tín hiệu điều khiển chiều quay.

#### e. Bộ điều khiển trung tâm Arduino mega 2560

Để điều khiển động cơ, đọc và gửi vận tốc về ROS trong đồ án cần bộ điều khiển trung tâm để làm nhiệm vụ đó. Bộ điều khiển được chọn trong đồ án là bộ điều khiển Arduino. Arduino là phần cứng mã nguồn mở với ngôn ngữ lập trình C và C++ khá phổ biến và dễ dàng cho người phát triển. Vì là mã nguồn mở nên Arduino được cộng đồng phát triển xây dựng rất nhiều bộ thư viện hỗ trợ giúp đơn giản cho việc lập trình. Trong đồ án ta sử dụng Arduino Mega 2560 với thông số sau[19]:



Hình 3.9. Bộ điều khiển Arduino mega 2560

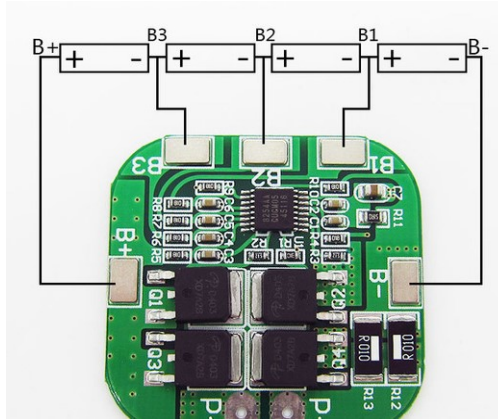
- Vi điều khiển chính: ATmega 2560
- IC nạp và giao tiếp UART: ATmega16U2
- Nguồn nuôi mạch: 5VDC từ cổng usb hoặc nguồn cắm ngoài từ 7-12VDC
- Số chân digital: 54 chân (15 chân PWM)
- Số chân analog: 16
- Giao tiếp UART: 4 bộ
- Giao tiếp SPI: 1 bộ (chân 50-53)
- Giao tiếp I2C: 1 bộ
- Ngắt ngoài: 6 chân
- Bộ nhớ Flash: 256kb, 8KB sử dụng cho Bootloader
- SRAM: 8KB
- EEPROM: 4KB
- Xung clock: 16MHz

*f. Nguồn nuôi cho mô hình robot*

Trong đồ án cần 1 nguồn cung cấp cho camera Kinect với dòng 1A và 2 động cơ với dòng tải lớn nhất mỗi động cơ là 1400mA. Vậy dòng xả lớn nhất của nguồn sẽ là 4A. Để có cung cấp nguồn cho Robot ta sử dụng nguồn từ cell pin có thể sạc lại thay acquy để giảm khối lượng và kích thước của xe. Trên



Robot ta sẽ dùng 4 pin cell với mạch cân bằng khi sạc và xả được mắc như sau[19]:



Hình 3.10. Sơ đồ mắc pin vào mạch cân bằng sạc xả

Điện áp ra của mạch cell nhỏ nhất sẽ là 14.4V và lớn nhất sẽ là 16.8V nhưng trong đồ án chỉ dùng nguồn 12VDC với dòng  $\sim 4A$  ta sẽ dùng mạch Buck DC-DC hạ áp xuống 12V và dòng qua tải max 5A để đáp ứng yêu cầu.

*g. Máy tính nhúng Raspberry*

Arduino giao tiếp với Raspberry qua cổng usb với giao thức UART. Raspberry đóng vai trò là một máy khách trên ROS và máy tính cá nhân là một máy chủ. Raspberry kết nối với wifi và giao tiếp với máy chủ server qua chuẩn giao tiếp chủ-khách của ROS với nhiều máy. Raspberry sẽ gửi dữ liệu từ camera và động cơ lên cho máy chủ qua mạng Wifi.

Raspberry Pi Model B+ là phiên bản nâng cấp đáng kể của Raspberry Pi Model B với phần GPIO nhiều hơn (40 chân), cổng USB nhiều hơn (4 cổng) và có thể cung cấp dòng lớn hơn bản cũ, cổng USB cung cấp dòng chuẩn 500mAh (ở bản cũ chỉ 200mAh). Ở phiên bản này Raspberry Pi B+ thay đổi lớn nhất và đáng kể nhất nằm ở phần nguồn với thiết kế nguồn tốt và ổn định hơn rất nhiều so với phiên bản cũ, thiết kế nguồn mới mang lại vô số ưu điểm so với bản cũ như nguồn nuôi ổn định hơn, bền hơn, cấp được dòng đủ cho cổng USB, cho cổng HDMI[17].

Bảng 3.5. Thông số kỹ thuật của Raspberry Pi Model B+

Vi điều khiển	ARM 32 bits ARM1176JZF-S
Điện áp hoạt động	5V
Điện áp vào giới hạn	6V
Chân I/O	40 chân khoảng cách 2.54mm
Dòng ổn định	2A
RAM	512Mhz
Tốc độ	700MHz từ Broadcom
Kết nối	1 cổng HDMI, 1 cổng Ethernet, jack Audio 3.5mm, Video RCA tích hợp và 4 cổng USB
Hệ điều hành	Linux với cộng đồng mã nguồn mở rất lớn.
Jack nguồn	5V cho Micro USB



*Hình 3.11. Máy tính nhúng Raspberry*

### 3.3 Thiết kế Robot

#### 3.3.1 Thiết kế cơ khí cho mô hình Robot

##### a) Phần khung xe

Phần khung xe là phần quan trọng của xe. Khung xe sẽ định hình toàn bộ hình dạng cho Robot và nâng đỡ toàn bộ các thành phần khác.

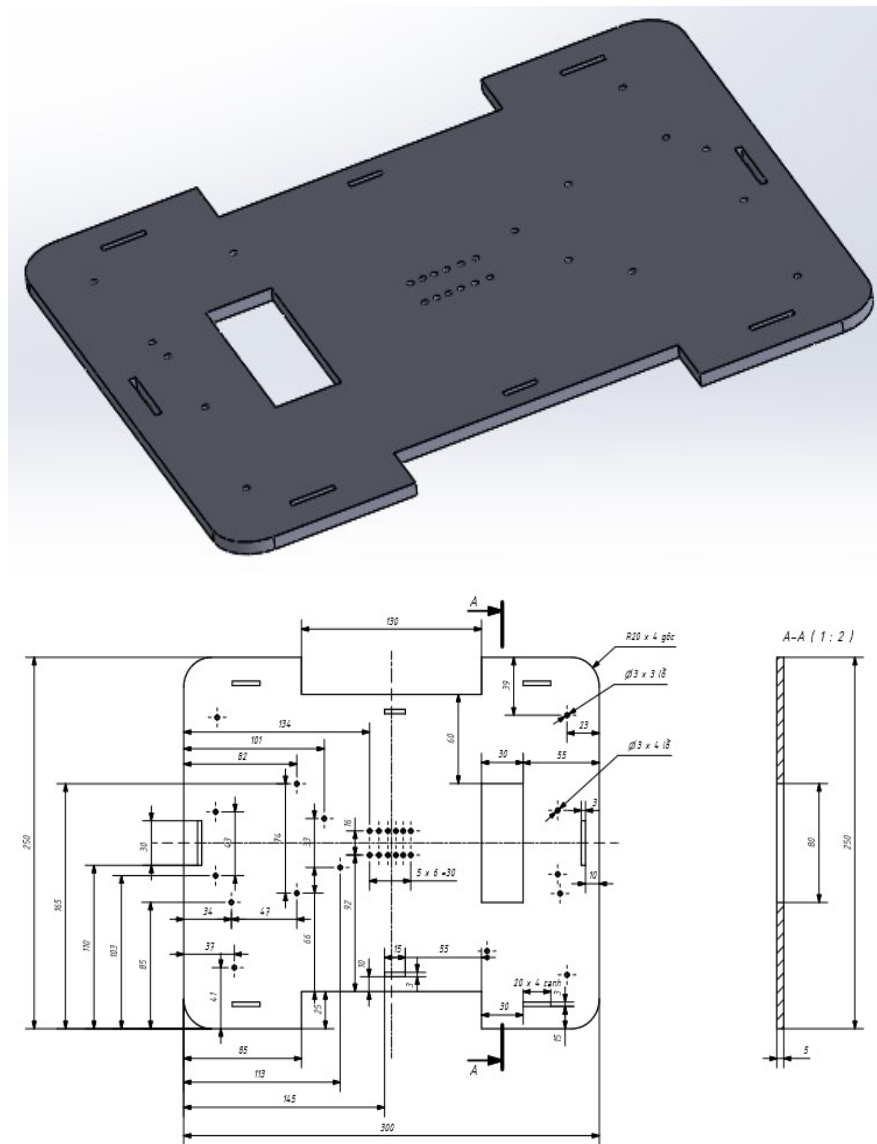
Khung xe có kích thước 300×250×75mm, bao gồm các bộ phận sau:

- Một tấm đế dưới dùng để làm giá;
- 2 tấm nhựa mặt trước và sau của modul di chuyển;
- 4 tấm nhựa 2 bên để bao robot thành một khung;
- Bánh dẫn động và bánh bi cầu;

- Động cơ và gá động cơ.

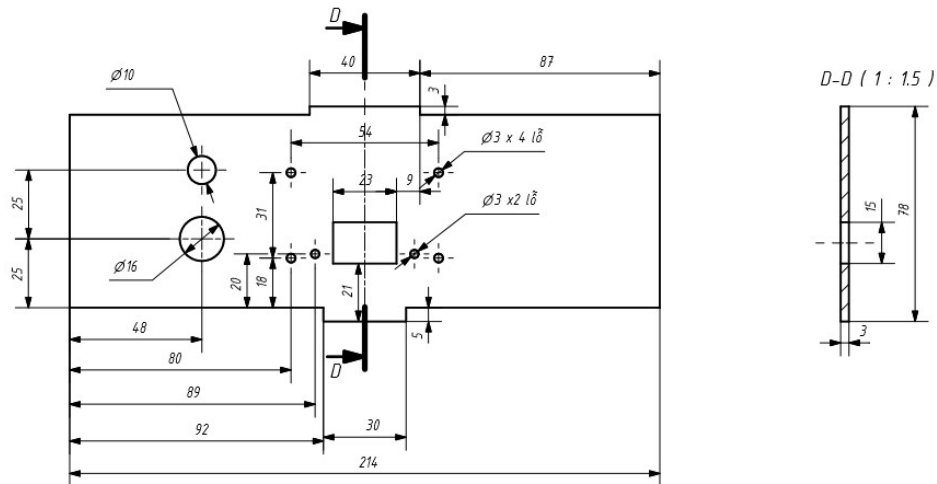
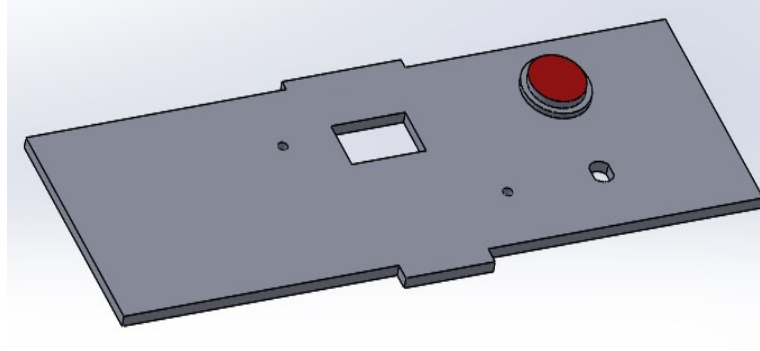
Bên trong khung chứa toàn bộ các bộ phận như: động cơ, mạch điều khiển (Arduino), driver điều khiển động cơ và bộ nguồn. Vật liệu sử dụng làm khung xe là nhựa mica đen.

Tấm mặt dưới có dạng hình chữ nhật, kích thước 250x300mm, độ dày là 5mm. Mặt tấm nhựa được khoan khá nhiều lỗ với nhiều kích thước để gá đặt linh kiện, dự trù khả năng có thể bố trí thêm linh kiện khác và đảm bảo khả năng tỏa nhiệt cho các thiết bị bên trong.

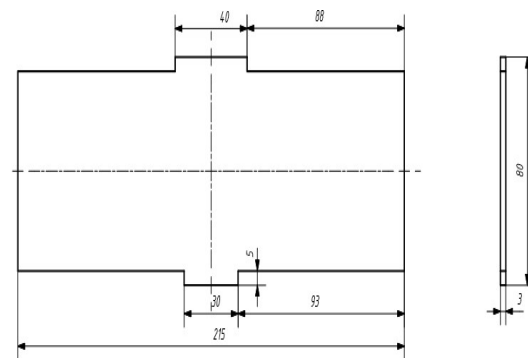
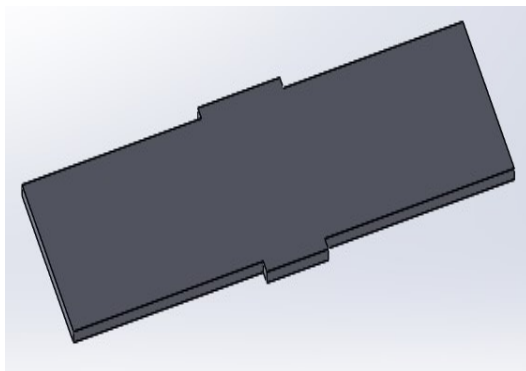


Hình 3.12. Tấm đế dưới

Tấm ốp trước và sau có dạng hình chữ nhật, kích thước 21,5x7mm, độ dày là 3mm. Trên mặt tấm ốp sau có khoan các lỗ để lắp đặt công tắc nguồn, rắc cắm sạc ac quy và ô hiển thị đèn led.



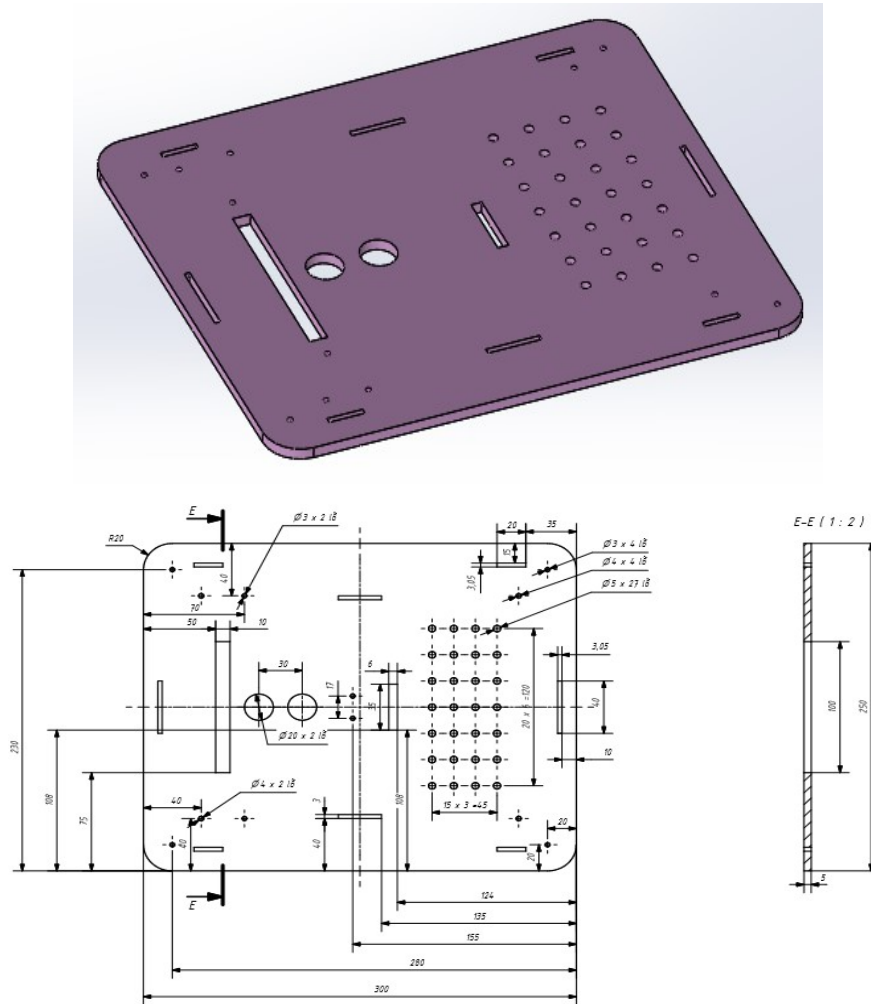
Hình 3.13. Tấm ốp mặt sau



Hình 3.14. Tấm ốp mặt trước

Tấm ốp mặt trên có kết cấu tương tự như mặt đáy, cũng có dạng hình chữ nhật, kích thước 250x300mm, độ dày là 5mm. Và cũng được khoan nhiều lỗ với

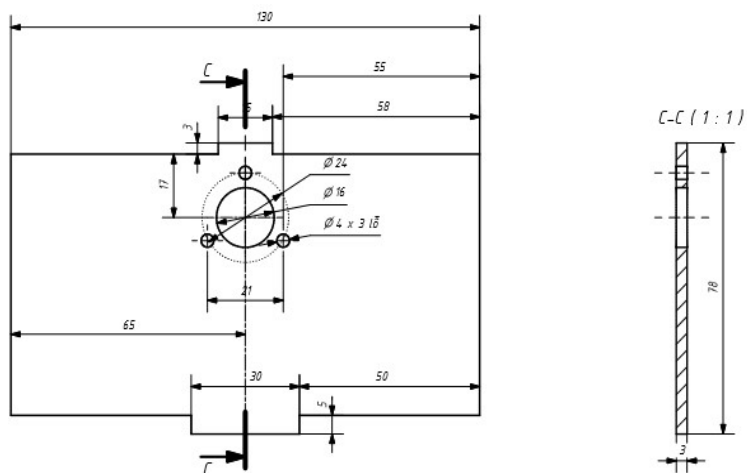
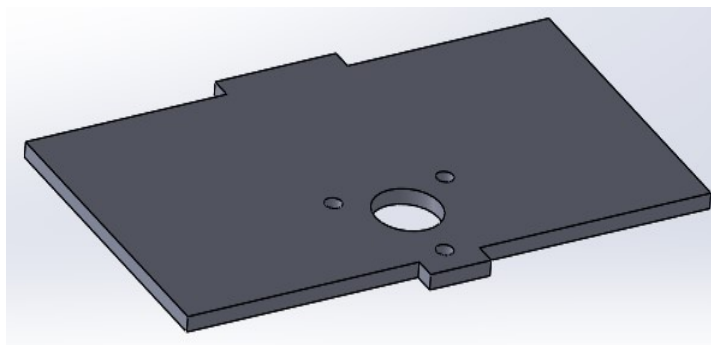
nhiều kích thước khác nhau để thuận tiện cho việc đi dây, tỏa nhiệt cho thiết bị và giảm khối lượng xe.



Hình 3.15. Tấm ốp mặt trên

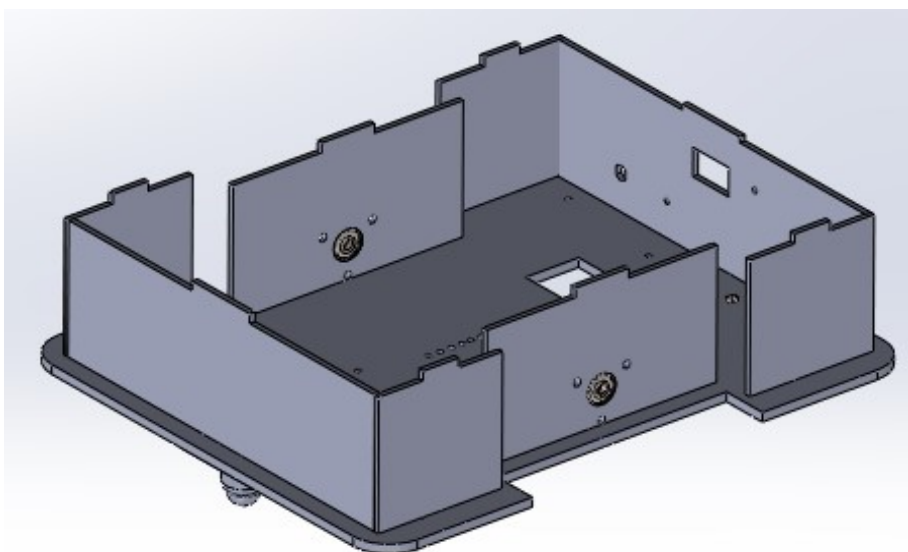
Thông số kỹ thuật của bánh dẫn động, bánh bi cầu và động cơ đã được tính toán và lựa chọn ở mục 3.2.2.

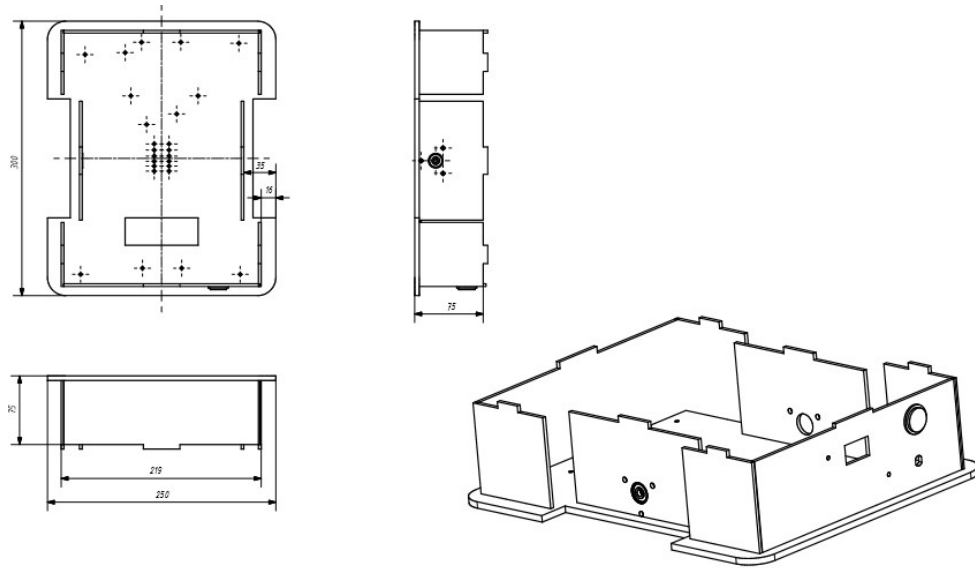
Để gá được động cơ cố định vào khung xe thì cần thiết kế một tấm gá có hình dạng và kích thước như sau:



Hình 3.16. Gá động cơ

Dưới đây là kết cấu của khung Robot sau khi lắp ráp hoàn thiện:





Hình 3.17. Khung Robot

Kích thước cụ thể các chi tiết thiết kế được thể hiện trong tập bản vẽ đính kèm ở phần phụ lục.

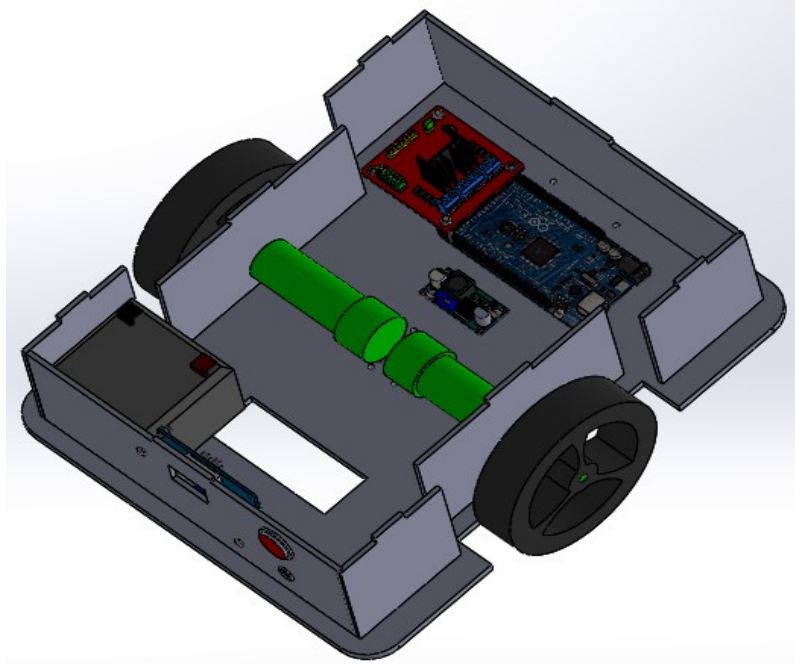
b) Các bộ phận khác đã được tính toán và lựa chọn trong mục 3.2.2.

### 3.3.2 Lắp ráp mô hình robot

Để đảm bảo lắp ráp dễ dàng và bố trí các phần tử hợp lý về cả phần cơ và phần điện – điện tử ta sẽ lắp ráp như sau:

- Trước tiên ta lắp 2 bánh dẫn động vào tấm đế dưới và lắp động cơ vào tấm gá động cơ.
- Tiếp theo lắp nguồn, các mạch nguồn, mạch điều khiển, các công tắc, nút nhấn và mạch cầu H vào tấm đế dưới.
- Dùng dây điện và jump cắm kết nối các phần tử điện-điện tử với nhau như sơ đồ mạch điện.
- Lắp camera vào khung và gắn khung vào đế trên.
- Lắp tấm ốp các mặt bên, tấm gá động cơ vào tấm đế dưới, lắp tấm đế trên lên và cố định các phần tử bằng bulong đai ốc ta được mô hình hoàn chỉnh.

a) Lắp ráp động cơ, bánh xe và các thiết bị điện – điện tử lên khung xe



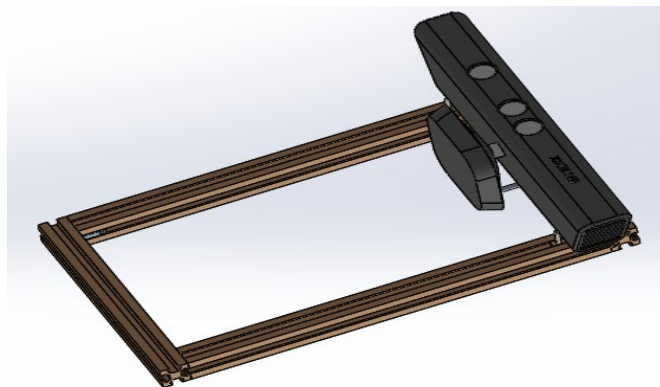
*Hình 3.18. Mô hình 3D bố trí thiết bị điện – điện tử trong mô hình*

b) Lắp ráp tấm mặt trên lên phần khung xe

Tấm mặt trên được cố định vào khung xe bằng các rãnh đã được khoét sẵn ở tấm mặt trên và các tấm trên phần khung xe.

c) Lắp khung camera và Camera Kinect

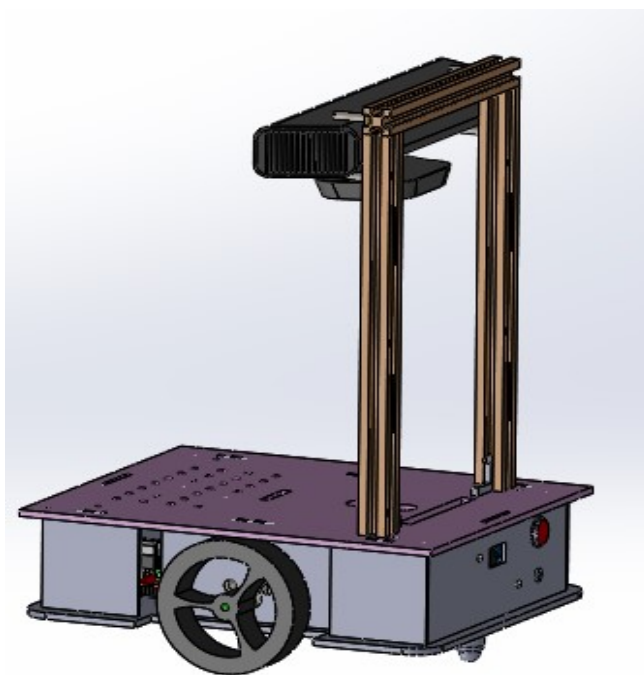
Camera được lắp cách mặt tấm trên từ 25 đến 35cm cho góc nhìn theo trục z của xe được rộng hơn và để đảm bảo xe không bị lật khi di chuyển với tốc độ lớn nhất của Robot. Vì vậy nhóm đã chọn nhôm định hình để đảm bảo các yêu cầu trên. Camera gắn với đế xe bởi một khung nhôm định hình vuông có kích thước 20x20mm.



*Hình 3.19. Mô hình 3D phần gá camera và camera*



## d) Lắp ráp Robot hoàn thiện



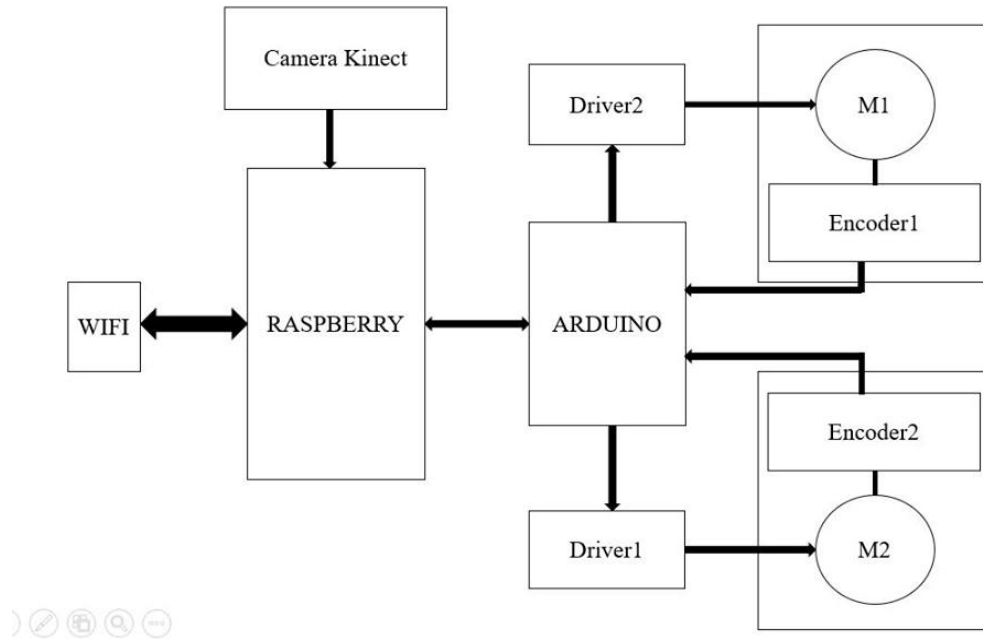
Hình 3.20. Mô hình robot đã lắp ráp hoàn thiện trên Solidwords



Hình 3.21. Mô hình robot thực tế

### 3.4 Sơ đồ hệ thống mô hình Robot

#### 3.4.1 Sơ đồ cấu trúc hệ thống mô hình Robot

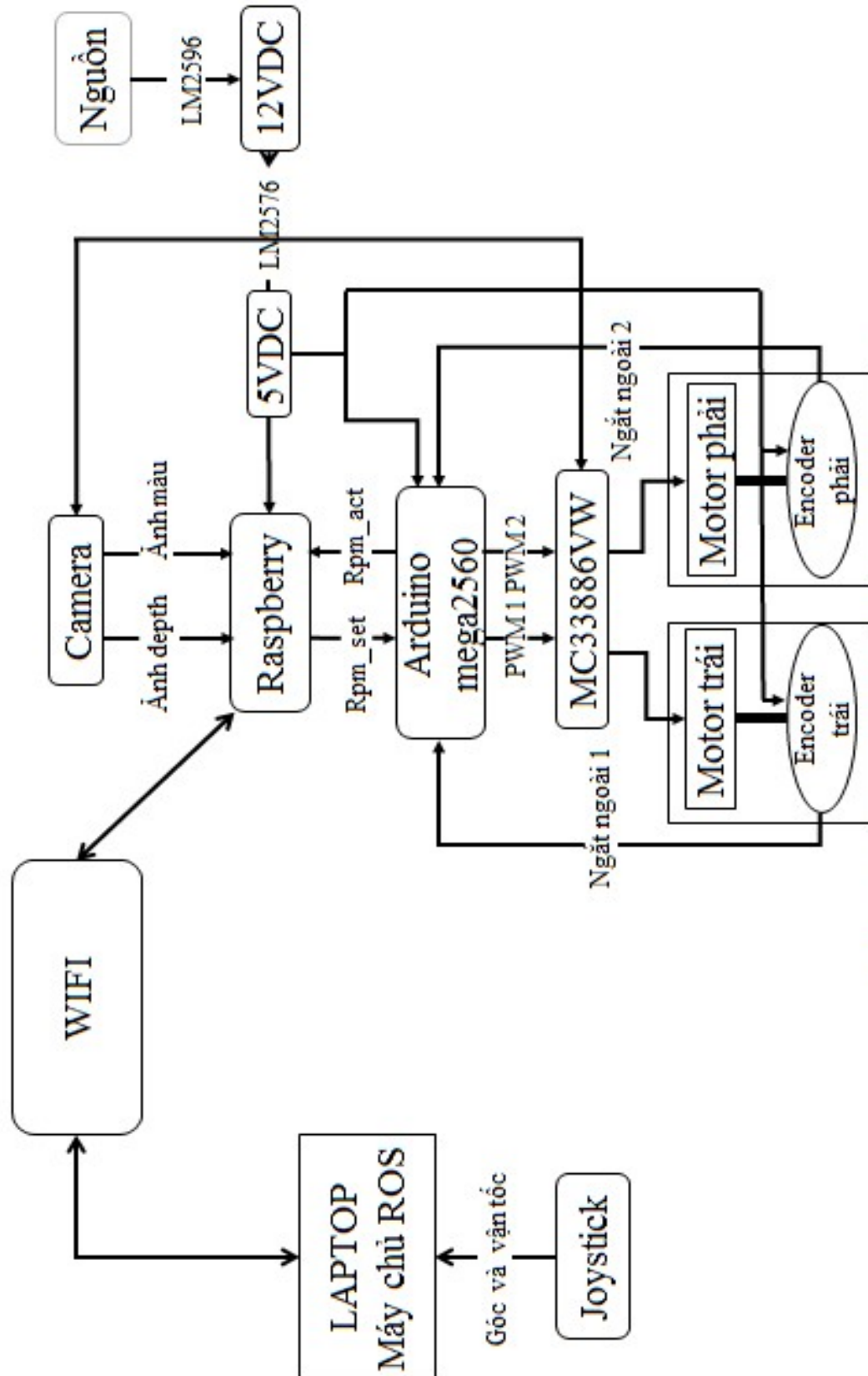


Hình 3.22. Sơ đồ cấu trúc hệ thống điều khiển Robot

Chức năng các khối trong sơ đồ:

- Raspberry đóng vai trò là một máy khách nhận dữ liệu ảnh depth từ camera Kinect xử lý và đưa ra khoảng cách đến vật cản, truyền/nhận tín hiệu vận tốc với Arduino và gửi dữ liệu đó về máy chủ qua Wifi.
- Arduino có chức năng tính toán, điều khiển vận tốc động cơ dẫn động M1 và M2 qua mạch công suất Driver1 và Driver2.
- Động cơ dẫn động M1, M2 có encoder phản hồi tốc độ về Arduino.

## 3.4.2 Sơ đồ điều khiển



Hình 3.23. Sơ đồ điều khiển mô hình Robot

*Hình 3.23. Sơ đồ điều khiển mô hình Robot*

## CHƯƠNG 4: THỰC NGHIỆM VÀ ĐÁNH GIÁ

### 4.1 Cài đặt hệ điều hành Ubuntu

- Trên Raspberry
  - Trên Raspberry cài đặt hệ điều hành Ubuntu MATE 16.04 (tương thích với phiên bản ROS Kinectic) và thiết đặt cấu hình cho Raspberry như một máy khách;
  - Cài đặt các gói: `roscpp_serial_python`, `freenect_camera`, `depthimage_to_laserscan`.
- Trên máy tính Laptop hoặc máy tính để bàn
  - Cài đặt hệ điều hành Ubuntu LTS 16.04 và hệ điều hành robot phiên bản kinetic và thiết đặt cấu hình trên ubuntu để laptop như 1 máy chủ;
  - Cài đặt các gói: `slam_gmapping`, `navigation`, `URDF`, `RVIZ`, `joy`.

### 4.2 Xây dựng bản đồ tĩnh

#### 4.2.1 Công tác chuẩn bị

- Tiến hành 2 thực nghiệm: Một thực nghiệm với môi trường đơn giản, có ít vật cản để sử dụng bản đồ cho khả năng tránh vật cản của Robot; Một thực nghiệm trong môi trường phức tạp hơn, có nhiều vật cản hơn.
  - + Môi trường đơn giản được chọn là khu vực thang máy tầng 2 nhà S1
  - + Môi trường phức tạp hơn ta sẽ bố trí các vật cản trong phòng học viên đến hành lang nhà S11.

#### 4.2.2 Tiến hành thực nghiệm

- Bước 1: Kết nối máy chủ và máy khách vào cùng một mạng wifi
- Bước 2: Chạy `roscpp_serial_python` trên máy chủ để khởi tạo ROS Master
- Bước 3: Chạy các gói ứng dụng cho camera Kinect và gói ứng dụng truyền/nhận dữ liệu với Arduino
- Bước 4: Chạy driver điều khiển joystick trên ROS và các gói ứng dụng cho SLAM\_GMAPPING
- Bước 5: Sử dụng joystick điều khiển cho Robot chạy quanh khu vực muốn quét bản đồ.

- Bước 6: Khi bản đồ được quét xong so sánh với thực tế và tiến hành lưu bản đồ vào máy chủ bằng gói ứng dụng map\_sever trên ROS

Sau quá trình thực nghiệm ta thu được 2 bản đồ ở 2 môi trường khác nhau.



*Hình 4.1. Bản đồ 2D thu được và hình ảnh địa hình thực tế*

Hình 4.1 là 2 bản đồ được xây dựng trong hai môi trường khác nhau. Hình với màu đen là các vật cản mà Robot nhận diện được. Màu trắng là khu vực không có vật cản. Mỗi ô vuông trên bản đồ tương ứng với hình vuông  $6 \times 6$  m ở ngoài thực tế (bằng 1 viên gạch nền của hành lang).

#### *4.2.3 Đánh giá thực nghiệm*

Robot đã tự xây dựng được bản đồ tương đối chính xác so với địa hình thực tế.

## ❖ Hạn chế:

- Do sai số thông tin đo lường nên các vòng quét xảy ra sự sai lệch, các vật cản trông khớp nhau hoàn toàn dẫn đến sai số trong quá trình định vị.
- Các vật cản cao hơn hoặc thấp hơn so với tầm nhìn của camera sẽ không lưu lại trong bản đồ.

### 4.3 Thực nghiệm khả năng định vị và điều hướng với gói ứng dụng Navigation

#### 4.3.1 Công tác chuẩn bị.

Tiến hành 2 thực nghiệm trên 2 bản đồ đã được xây dựng trong *Phần 4.1 thực nghiệm xây dựng bản đồ tĩnh*.

- Thực nghiệm thứ nhất cho robot hoạt động trong môi trường trong phòng (môi trường phức tạp) để kiểm tra khả năng lập kế hoạch di chuyển tránh những vật cản tĩnh được ghi lại trước đó
- Thực nghiệm thứ 2 cho Robot hoạt động trên môi trường đơn giản hơn (cửa thang máy tầng 2 nhà S1) nhưng trong khi Robot đang di chuyển thực hiện thay đổi vật cản trên quỹ đạo Robot đã lập ra để kiểm tra khả năng tránh vật cản chưa biết của Robot.

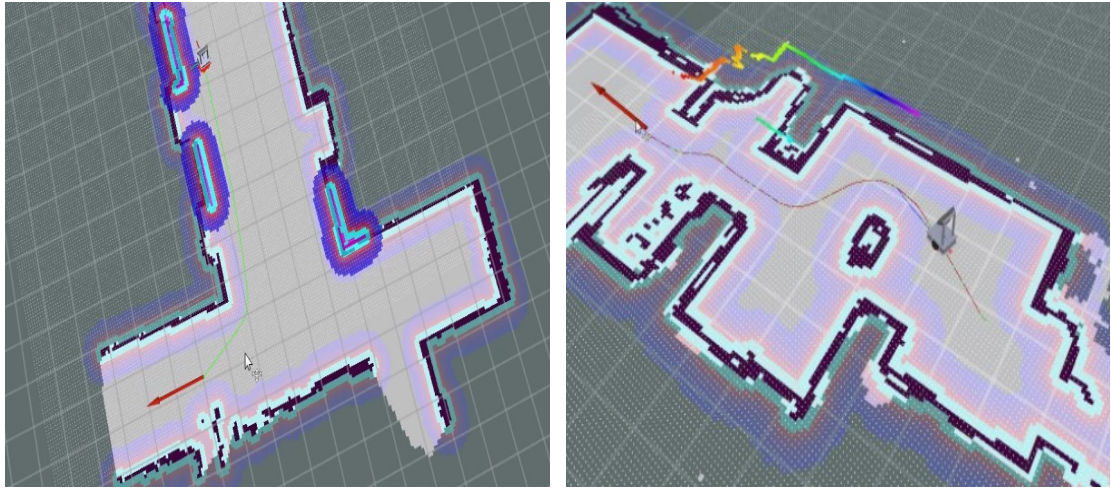
#### 4.3.2 Tiến hành thực nghiệm

##### a) Thực nghiệm 1: Kiểm tra khả năng lập kế hoạch di chuyển của Robot

## ❖ Các bước tiến hành:

- Bước 1: Kết nối máy chủ và máy khách tới cùng một wifi và thiết đặt cấu hình mạng cho mỗi máy sau đó chạy roscore trên máy chủ để khởi tạo ROS Master.
- Bước 2: Chạy các gói ứng dụng cho camera Kinect và gói ứng dụng truyền/nhận dữ liệu với Arduino trên máy khách.
- Bước 3: Chạy gói ứng dụng Navigation, driver joystick và nạp bản đồ đã xây dựng trên Phần 4.2 (Cửa thang máy tầng 2 nhà S1) vào máy chủ.

- Bước 4: Trên terminal chạy lệnh `rosservice call/global_localization` “{}” để phân tán xác suất vị trí của Robot vào mọi nơi trên bản đồ sau đó thực hiện di chuyển Robot bằng joystick để robot tự định vị bản thân cho đến khi các hạt phân bố xác suất tập trung tại chân của Robot trên RVIZ. Khi đó Robot đã định vị được vị trí của bản thân trên bản đồ.
- Bước 5: Trên thanh công cụ của RVIZ chọn *2D Nav Goal* để vẽ một vecto cho Robot tự lập kế hoạch đi đến cuối vecto đó.



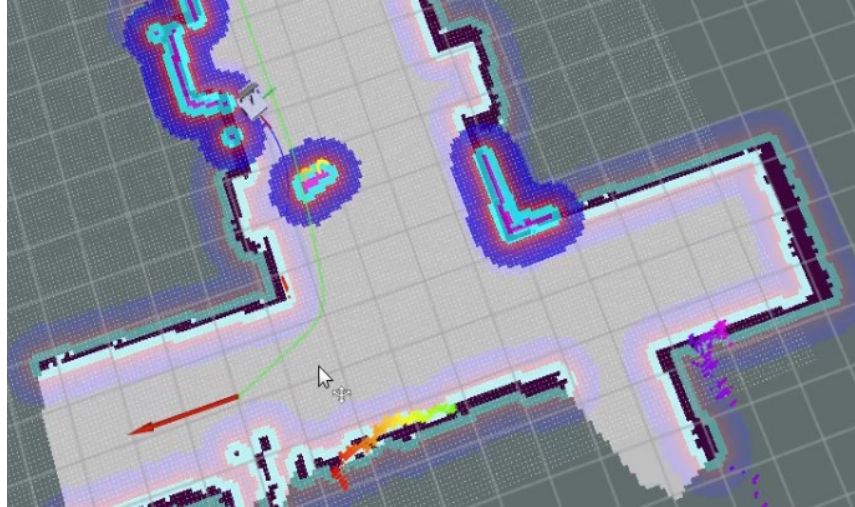
*Hình 4.2. Kế hoạch di chuyển với vật cản cố định*

- b) Thực nghiệm 2: Kiểm tra khả năng lập kế hoạch mới khi gặp vật cản trong kế hoạch cũ

❖ Các bước tiến hành thực nghiệm

- Các bước 1 đến bước 5 tiến hành giống như trong thực nghiệm 1, thay bản đồ phòng ở bằng bản đồ cửa thang máy tầng 2 nhà S1.
- Trên đường quỹ đạo Robot đã lập ra thực hiện thêm vật cản bằng cách cho một người đứng vào vị trí Robot sẽ đi qua và cách robot ít nhất 1 mét. Xem xét khả năng tránh vật cản của Robot.





*Hình 4.3. Phát hiện vật cản chưa biết*

#### 4.3.3 Đánh giá thực nghiệm

Thực nghiệm 1: Robot có khả năng định vị, lập kế hoạch và di chuyển theo quỹ đạo đã xây dựng khá ổn định.

Thực nghiệm 2: Robot có thể xây dựng được quỹ đạo mới khi gặp vật cản mới trên đường quỹ đạo cũ và thay đổi hướng di chuyển theo quỹ đạo mới.

Hạn chế: + Khi gặp vật cản mới trong khoảng cách gần Robot không kịp xử lý để tránh.

+ Khi bị trượt bánh Robot không có khả năng định vị lại vị trí của bản thân gây ra sai số thông tin đo lường lớn.

## KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

Trong thời gian thực hiện đồ án, cùng với sự giúp đỡ của thầy giáo Hoàng Quang Chính, thầy giáo Hà Huy Hưng và các thầy trong bộ môn, với đồ án “*Nghiên cứu ứng dụng hệ điều hành robot (ROS) trong điều khiển robot di động*” đã hoàn thành và giải quyết các vấn đề sau:

- Hiểu được cấu trúc, phương thức truyền nhận dữ liệu của hệ điều hành robot;

- Ứng dụng hệ điều hành robot để xây dựng bản đồ tĩnh 2D và robot có thể định hướng, định vị và lập kế hoạch di chuyển trên bản đồ đã xây dựng.

Bên cạnh những vấn đề đã được giải quyết, nhóm đồ án vẫn còn tồn tại một số vấn đề chưa được giải quyết triệt để như sau:

- Độ chính xác của cảm biến còn chưa cao nên bản đồ xây dựng có độ chính xác còn hạn chế và khả năng tự định vị còn có sai số.

Do kiến thức và thời gian có hạn nên quá trình nghiên cứu và phát triển robot tự hành của nhóm đồ án mới chỉ dừng lại ở bước xây dựng bản đồ và lập kế hoạch di chuyển với bản đồ 2D. Vậy nên nhóm đồ án xin đưa ra một số đề xuất nghiên cứu và phát triển thêm:

- Xây dựng bản đồ, định vị, định hướng và lập kế hoạch với bản đồ 3D;

- Để tăng chất lượng odometry, ta có thể tích hợp thêm cảm biến IMU chuyên dụng và GPS trên ROS. Với khả năng ước lượng tọa độ chính xác hơn, ta có thể tăng độ chính xác trong quá trình SLAM tạo bản đồ cho Robot. Thực tế trong đồ án cho thấy, bản đồ càng chính xác thì Navigation Stack càng hoạt động hiệu quả.

- Với quá trình tự hành, ta có thể sử dụng cảm biến laser chuyên dụng để tìm được thông tin khoảng cách có tầm xa lớn hơn giới hạn 5m của Kinect. Giúp tăng khả năng định vị của giải thuật *amcl*.

- Phát triển thêm cánh tay robot trên mô hình để nâng cao tính ứng dụng thực tế.

Tuy đã rất cố gắng những nhóm đồ án cũng không thể tránh khỏi sai sót trong quá trình làm đồ án, do đó kính mong các thầy thông cảm và góp ý cho nhóm đồ án của chúng em hoàn thiện hơn.

Một lần nữa, xin chân thành cảm ơn sự giúp đỡ nhiệt tình của thầy Hoàng Quang Chính, thầy Hà Huy Hưng và các thầy trong bộ môn Robot đặc biệt và Cơ điện tử đã giúp chúng em hoàn thành đồ án này.

## TÀI LIỆU THAM KHẢO

- [1]. [https://en.wikipedia.org/wiki/Microsoft\\_Robotics\\_Developer\\_Studio](https://en.wikipedia.org/wiki/Microsoft_Robotics_Developer_Studio)
- [2]. [https://en.wikipedia.org/wiki/Mobile\\_Robot\\_Programming\\_Toolkit](https://en.wikipedia.org/wiki/Mobile_Robot_Programming_Toolkit)
- [3]. <http://wiki.ros.org/Introduction>
- [4]. <http://library.isr.ist.utl.pt>
- [5]. [http://library.isr.ist.utl.pt/docs/roswiki/costmap\\_2d.html](http://library.isr.ist.utl.pt/docs/roswiki/costmap_2d.html)
- [6]. [https://en.wikipedia.org/wiki/Monte\\_Carlo\\_localization](https://en.wikipedia.org/wiki/Monte_Carlo_localization)
- [7]. [https://en.wikipedia.org/wiki/Simultaneous\\_localization\\_and\\_mapping](https://en.wikipedia.org/wiki/Simultaneous_localization_and_mapping)
- [8]. <http://developkinect.com/resource/sdk-middleware-driver/openni-primesense-package-installers>
- [9]. <http://pointclouds.org>
- [10]. [https://vi.wikipedia.org/wiki/Định\\_lý\\_Bayes](https://vi.wikipedia.org/wiki/Định_lý_Bayes)
- [11]. [https://vi.wikipedia.org/wiki/Phương\\_pháp\\_Monte\\_Carlo](https://vi.wikipedia.org/wiki/Phương_pháp_Monte_Carlo)
- [12]. <https://www.scribd.com/doc/phuong-phap-Monte-Carlo>
- [13]. [https://en.wikipedia.org/wiki/Gaussian\\_function](https://en.wikipedia.org/wiki/Gaussian_function)
- [14]. <http://automation.net.vn/Robot-Robotics/Mot-so-ky-thuat-dinh-vi-cho-Robot-di-dong.html>
- [15]. Hoàng Quang Chính, Trần Tuấn Trung, Hà Huy Hưng và cộng sự, *Thiết kế và lập trình minirobot*, Nhà Xuất Bản Quân Đội Nhân Dân, Hà Nội, 2016.
- [16]. <http://indoteck.vn/vi/2016/03/02/bang-tra-cac-he-so-ma-sat/>
- [17]. <http://hshop.vn/products/moy-tonh-raspberry-pi-model-b-made-in-uk>
- [18]. [S.K. Ghosh, 2007].
- [19]. <http://www.roboconshop.com>
- [20]. <http://haloshop.vn/kinect-for-xbox-360.html>
- [21]. <http://doantotnghiep.vn/duong-di-cua-robot-dung-kinect-tren-nen-tang-robot-operating-system.html>

**Phụ lục 1: BẢN VẼ CHI TIẾT CƠ KHÍ**

## Phụ lục 2: CHƯƠNG TRÌNH ĐIỀU KHIỂN

### ❖ Chương trình điều khiển modul di động trên ROS

Chương trình được viết bằng ngôn ngữ C++. Chương trình sẽ nhận dữ liệu vận tốc từ nút *joystick* tính toán và gửi dữ liệu vận tốc xuống modul di động. Nhận dữ liệu vận tốc tức thời từ modul di động gửi lên, xuất bản và chuyển đổi thông tin đo lường.

- *Tiền khai báo thư viện ROS*

```
#include <ros/ros.h>
#include <tf/transform_broadcaster.h>
#include <nav_msgs/Odometry.h>
#include <geometry_msgs/Vector3.h>
#include <stdio.h>
#include <cmath>
#include <algorithm>

#define pi                3.1415926
#define two_pi            6.2831853
#define gear_ratio        64
#define encoder_pulse     12
#define pi                3.1415926
#define wheel_diameter    0.1 //m
#define track_width       0.23 //m
#define max_RPM           120    //vong/p

double rpm_act1 = 0.0;
double rpm_act2 = 0.0;
double rpm_req1 = 0.0;
double rpm_req2 = 0.0;
double gyro_x = 0.0;
double gyro_y = 0.0;
double gyro_z = 0.0;
double rpm_dt = 0.0;
double x_pos = 0.0;
double y_pos = 0.0;
double theta = 0.0;
```

```

ros::Time current_time;
ros::Time rpm_time(0.0);
ros::Time last_time(0.0);

```

• *Nhận dữ liệu vận tốc tức thời từ ROS với tần số “rate”*

```

void handle_rpm( const geometry_msgs::Vector3Stamped& rpm) {
    rpm_act1 = rpm.vector.x;
    rpm_act2 = rpm.vector.y;
    rpm_dt = rpm.vector.z;
    rpm_time = rpm.header.stamp;
}

```

• *Nhận dữ liệu IMU*

```

void handle_gyro( const geometry_msgs::Vector3& gyro) {
    gyro_x = gyro.x;
    gyro_y = gyro.y;
    gyro_z = gyro.z;
}

```

```

int main(int argc, char** argv){
    ros::init(argc, argv, "base_controller");

    ros::NodeHandle n;
    ros::NodeHandle nh_private_("~");
    ros::Subscriber sub = n.subscribe("rpm", 50, handle_rpm);
    ros::Subscriber gyro_sub = n.subscribe("gyro", 50,
handle_gyro);
    ros::Publisher odom_pub =
n.advertise<nav_msgs::Odometry>("odom", 50);
    tf::TransformBroadcaster broadcaster;

    double rate = 10.0;
    double linear_scale_positive = 1.0;
    double linear_scale_negative = 1.0;
    double angular_scale_positive = 1.0;

```

```

double angular_scale_negative = 1.0;
double angular_scale_accel = 1.0;
double acc_theta = 0.0;
double acc_x = 0.0;
double acc_max_theta = 0.0;
double acc_max_x = 0.0;
double alpha = 0.0;
bool publish_tf = true;
bool use_imu = false;
double dt = 0.0;
double dx = 0.0;
double dy = 0.0;
double dth_odom = 0.0;
double dth_gyro = 0.0;
double dth = 0.0;
double dth_prev = 0.0;
double dth_curr = 0.0;
double dxy_prev = 0.0;
double dxy_ave = 0.0;
double vx = 0.0;
double vy = 0.0;
double vth = 0.0;
char base_link[] = "/base_link";
char odom[] = "/odom";
ros::Duration d(1.0);

```

• *Khai báo sử dụng các bộ tham số thiết đặt*

```

nh_private_.getParam("publish_rate", rate);
nh_private_.getParam("publish_tf", publish_tf);
nh_private_.getParam("linear_scale_positive",
linear_scale_positive);
nh_private_.getParam("linear_scale_negative",
linear_scale_negative);
nh_private_.getParam("angular_scale_positive",
angular_scale_positive);

```



```

    nh_private_.getParam("angular_scale_negative",
angular_scale_negative);
    nh_private_.getParam("angular_scale_accel",
angular_scale_accel);
    nh_private_.getParam("alpha", alpha);
    nh_private_.getParam("use_imu", use_imu);

    ros::Rate r(rate);
    while(n.ok()){
        ros::spinOnce();
        //
ros::topic::waitForMessage<geometry_msgs::Vector3Stamped>("rpm
", n, d);
        current_time = ros::Time::now();
        dt = rpm_dt;
        dxy_ave = (rpm_act1+rpm_act2)*dt*wheel_diameter*pi/(60*2);
        dth_odom = (rpm_act2-
rpm_act1)*dt*wheel_diameter*pi/(60*track_width);

        if (use_imu) dth_gyro = dt*gyro_z;
        dth = alpha*dth_odom + (1-alpha)*dth_gyro;

        if (dth > 0) dth *= angular_scale_positive;
        if (dth < 0) dth *= angular_scale_negative;
        if (dxy_ave > 0) dxy_ave *= linear_scale_positive;
        if (dxy_ave < 0) dxy_ave *= linear_scale_negative;

        dx = cos(dth) * dxy_ave;
        dy = -sin(dth) * dxy_ave;

        x_pos += (cos(theta) * dx - sin(theta) * dy);
        y_pos += (sin(theta) * dx + cos(theta) * dy);
        theta += dth;

        if(theta >= two_pi) theta -= two_pi;

```

```

if(theta <= -two_pi) theta += two_pi;

geometry_msgs::Quaternion odom_quat =
tf::createQuaternionMsgFromYaw(theta);

if(publish_tf) {
    geometry_msgs::TransformStamped t;
    t.header.frame_id = odom;
    t.child_frame_id = base_link;
    t.transform.translation.x = x_pos;
    t.transform.translation.y = y_pos;
    t.transform.translation.z = 0.0;
    t.transform.rotation = odom_quat;
    t.header.stamp = current_time;

    broadcaster.sendTransform(t);
}

```

• *Chuyển đổi vận tốc vào thông tin đo lường*

```

nav_msgs::Odometry odom_msg;
odom_msg.header.stamp = current_time;
odom_msg.header.frame_id = odom;
odom_msg.pose.pose.position.x = x_pos;
odom_msg.pose.pose.position.y = y_pos;
odom_msg.pose.pose.position.z = 0.0;
odom_msg.pose.pose.orientation = odom_quat;
if (rpm_act1 == 0 && rpm_act2 == 0){
    odom_msg.pose.covariance[0] = 1e-9;
    odom_msg.pose.covariance[7] = 1e-3;
    odom_msg.pose.covariance[8] = 1e-9;
    odom_msg.pose.covariance[14] = 1e6;
    odom_msg.pose.covariance[21] = 1e6;
    odom_msg.pose.covariance[28] = 1e6;
    odom_msg.pose.covariance[35] = 1e-9;
    odom_msg.twist.covariance[0] = 1e-9;
    odom_msg.twist.covariance[7] = 1e-3;
}

```

```

odom_msg.twist.covariance[8] = 1e-9;
odom_msg.twist.covariance[14] = 1e6;
odom_msg.twist.covariance[21] = 1e6;
odom_msg.twist.covariance[28] = 1e6;
odom_msg.twist.covariance[35] = 1e-9;
}
else{
odom_msg.pose.covariance[0] = 1e-3;
odom_msg.pose.covariance[7] = 1e-3;
odom_msg.pose.covariance[8] = 0.0;
odom_msg.pose.covariance[14] = 1e6;
odom_msg.pose.covariance[21] = 1e6;
odom_msg.pose.covariance[28] = 1e6;
odom_msg.pose.covariance[35] = 1e3;
odom_msg.twist.covariance[0] = 1e-3;
odom_msg.twist.covariance[7] = 1e-3;
odom_msg.twist.covariance[8] = 0.0;
odom_msg.twist.covariance[14] = 1e6;
odom_msg.twist.covariance[21] = 1e6;
odom_msg.twist.covariance[28] = 1e6;
odom_msg.twist.covariance[35] = 1e3;
}
vx = (dt == 0)? 0 : dxy_ave/dt;
vth = (dt == 0)? 0 : dth/dt;

```

• *Xuất bản thông tin đo lường*

```

odom_msg.child_frame_id = base_link;
odom_msg.twist.twist.linear.x = vx;
odom_msg.twist.twist.linear.y = 0.0;
odom_msg.twist.twist.angular.z = dth;

odom_pub.publish(odom_msg);
last_time = current_time;
r.sleep();
}

```

## ❖ Chương trình điều khiển trên Arduino

### • Khai báo thư viện ROS trên arduino

```
#include <ros.h>
#include <std_msgs/Float32.h>
#include <turtle_actionlib/Velocity.h>
#include <tf/transform_broadcaster.h>
#include <geometry_msgs/Vector3Stamped.h>
#include <geometry_msgs/Twist.h>
#include <tf/tf.h>
#include <ros/time.h>
#include <nav_msgs/Odometry.h>
#include "controlmotor.h"

#define LOOPTIME 100
#define SMOOTH 10
#define sign(x) (x > 0) - (x < 0)

unsigned long lastMilli = 0;          // loop timing
unsigned long lastMilliPub = 0;
double rpm_reqL = 0;
double rpm_reqR = 0;
volatile long count_L=0,count_R=0;
int pwmR,pwmL;
unsigned long ant_countR,ant_countL;
long pulse_R,pulse_L;
float Kp=1.46,Ki=0.05,Kd=0.02;
int errL,errR,pre_errL,pre_errR;
long                                     int
    pPartR=0,iPartR=0,dPartR=0,pPartL=0,iPartL=0,dPartL=0;
const int encoderLA=19;
const int encoderLB=25;
const int encoderRA=18;
const int encoderRB=23;
float x_pos = 0.0;
```

```
float y_pos = 0.0;
float theta = 0.0;
char base_link[] = "/base_link";
char odom[] = "/odom";
unsigned long time;
ros::NodeHandle nh;
```

- *Nhận dữ liệu vận tốc từ máy tính.*

```
void handle_cmd( const geometry_msgs::Twist& cmd_msg) {
    float x = cmd_msg.linear.x;
    float z = cmd_msg.angular.z;
    if (z == 0) {          // go straight
        // convert m/s to rpm
        rpm_reqL = x*60/(pi*wheel_diameter);
        rpm_reqR = rpm_reqL;
    }
    else if (x == 0) {
        // convert rad/s to rpm
        rpm_reqR = z*track_width*60/(wheel_diameter*pi*2);
        rpm_reqL = -rpm_reqR;
    }
    else {
        rpm_reqL = x*60/(pi*wheel_diameter) -
        z*track_width*60/(wheel_diameter*pi*2);
        rpm_reqR = x*60/(pi*wheel_diameter) +
        z*track_width*60/(wheel_diameter*pi*2);
    }
}
```

- *Khai báo truyền nhận dữ liệu với ROS.*

```
ros::Subscriber<geometry_msgs::Twist>sub("cmd_vel",
handle_cmd);
geometry_msgs::Vector3Stamped rpm_msg;
ros::Publisher rpm_pub("rpm", &rpm_msg);
```

```
ros::Time current_time;
ros::Time last_time;
```

- *Setup các giá trị và kiểu truyền nhận.*

```
void setup() {
nh.getHardware()->setBaud(57600);
nh.initNode();
nh.subscribe(sub);
nh.advertise(rpm_pub);
SetPinMotor();
pinMode(encoderLA, INPUT_PULLUP);
pinMode(encoderLB, INPUT_PULLUP);
pinMode(encoderRA, INPUT_PULLUP);
pinMode(encoderRB, INPUT_PULLUP);
attachInterrupt(4, Read_EncL, FALLING);
attachInterrupt(5, Read_EncR, FALLING);
}
```

- *Chương trình chính điều khiển hai bánh xe của modul di động*

```
void loop() {
    nh.spinOnce();
    unsigned long time = millis();
    if(time-lastMilli>= LOOPTIME) {           // enter tmed loop
        pwmL=MotorL_Speed(rpm_reqL);
        pwmR=MotorR_Speed(rpm_reqR);
        if (rpm_reqL>=0) {
            ForwardL(pwmL);
        }
        else if (rpm_reqL<0) {
            BackwardL(pwmL);
        }
        if (rpm_reqR>=0){
            ForwardR(pwmR);
        }
        else if (rpm_reqR<0){
```

```

        BackwardR(pwmR);
    }
    publishRPM(time-lastMilli);
    lastMilli = time;
}
if(time-lastMilliPub >= LOOPTIME) {
    lastMilliPub = time;
}
}

```

- *Truyền dữ liệu với ROS*

```

void publishRPM(unsigned long time) {
    rpm_msg.header.stamp = nh.now();
    rpm_msg.vector.x = rpm_actL;
    rpm_msg.vector.y = rpm_actR;
    rpm_msg.vector.z = double(time)/1000;
    rpm_pub.publish(&rpm_msg);
    nh.spinOnce();
}

```

- *Bộ điều khiển PID cho mỗi bánh xe của modul di động*

```

int MotorR_Speed(double set_SpeedR) {
    int PWM_valR;
    pulse_R=count_R-ant_countR;
    ant_countR=count_R;
    rpm_actL=60*CalVelocity(pulse_R, LOOPTIME);
    errR=abs(set_SpeedR)-abs(rpm_actL);
    pPartR=Kp*errR;
    dPartR=Kd*(errR-pre_errR);
    iPartR+=Ki*(errR+pre_errR);
    PWM_valR +=pPartR+dPartR+iPartR;
    PWM_valR =constrain(PWM_valR ,0,255);
    if(PWM_valR >255) PWM_valR=255;
    if(PWM_valR <0) PWM_valR =0;
    pre_errR=errR;
    return PWM_valR;
}

```

```

}
int MotorL_Speed(double set_SpeedL){
int PWM_valL;
pulse_L=count_L-ant_countL;
ant_countL=count_L;
rpm_actR=60*CalVelocity(pulse_L, LOOPTIME);
errL=abs(set_SpeedL)-abs(rpm_actR);
pPartL=Kp*errL;
dPartL=Kd*(errL-pre_errL);
iPartL+=Ki*(errL+pre_errL);
PWM_valL+=pPartL+dPartL+iPartL;
PWM_valL=constrain(PWM_valL,0,225);
if(PWM_valL>255) PWM_valL=255;
if(PWM_valL<0) PWM_valL=0;
pre_errL=errL;
return PWM_valL;
}
//doc encoder left
void Read_EncL(){
    if(digitalRead(encoderLB)==LOW)
        count_L++;
    else
        count_L--;
}
//doc encoder right
void Read_EncR(){
    if(digitalRead(encoderRB)==LOW) count_R++;
    else count_R--;
}

```