

TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI  
VIỆN CƠ KHÍ  
BỘ MÔN CƠ SỞ THIẾT KẾ MÁY & ROBOT

— \* —

LUẬN VĂN  
**TỐT NGHIỆP CAO  
HỌC**  
NGÀNH CƠ ĐIỆN TỬ

**Đề tài: Phát triển hệ thống tránh vật cản và  
di chuyển thông minh cho robot tự hành  
Dashgo D1**

Học viên thực hiện:

**Nguyễn Văn Huy**

Mã số học viên:

**CB180009**

Giáo viên hướng dẫn:

**Lớp CH2018B**

Giáo viên phản biện:

**TS. Nguyễn Xuân Hà**

HÀ NỘI 06/2020

# Mục lục

<b>MỤC LỤC</b>	<b>6</b>
<b>DANH SÁCH HÌNH VẼ</b>	<b>8</b>
<b>1 Tổng quan nghiên cứu</b>	<b>1</b>
1.1 Giới thiệu robot tự hành . . . . .	1
1.2 Ứng dụng của robot tự hành thông minh . . . . .	3
1.3 Các bài toán trên robot tự hành thông minh . . . . .	3
1.4 Các nghiên cứu tránh vật cản trong robot tự hành thông minh . .	4
1.4.1 Thuật toán Virtual Force Field (VFF) . . . . .	4
1.4.2 The Vector Field Histogram . . . . .	6
1.4.3 Phương pháp "Bóng bóng phản ứng" tránh vật cản . . . .	7
1.5 Nội dung nghiên cứu . . . . .	8
<b>2 Cơ sở lý thuyết</b>	<b>9</b>
2.1 Bài toán về nhiễu trong robot tự hành . . . . .	9
2.1.1 Sự không chắc chắn trong robot . . . . .	9
2.1.2 Xác suất trong robotics . . . . .	10
2.2 Hệ điều hành robot ROS và các ứng dụng . . . . .	14
2.2.1 ROS là gì? . . . . .	14
2.2.2 Tổng quan về HDH ROS . . . . .	15
2.2.3 Tại sao phải dùng ROS . . . . .	17
2.2.4 Một số thành phần cơ bản trong ROS . . . . .	19
2.3 Bài toán điều hướng robot . . . . .	23
2.3.1 Điều hướng robot di động . . . . .	23
2.4 Bài toán SLAM 2D . . . . .	26
2.4.1 Một số phương pháp định vị . . . . .	26
2.4.2 SLAM . . . . .	28
2.5 Lý thuyết điều hướng robot . . . . .	30
2.5.1 Costmap . . . . .	30
2.5.2 AMCL . . . . .	31

2.5.3 Dynamic Window Approach (DWA) . . . . .	32
<b>3 Thực nghiệm</b>	<b>34</b>
3.1 Đặt vấn đề . . . . .	34
3.2 Giới thiệu platform robot của nhóm . . . . .	34
3.2.1 Phần chân đế . . . . .	35
3.2.2 Phần cảm biến . . . . .	35
3.2.3 Hệ thống phần mềm . . . . .	36
3.3 Điều khiển Dashgo robot . . . . .	37
3.3.1 Quy trình thực hiện . . . . .	37
3.3.2 Đánh giá hoạt động của robot . . . . .	38
3.4 Phương pháp . . . . .	39
3.4.1 Phần cứng . . . . .	39
3.4.2 Xử lý dữ liệu cảm biến . . . . .	40
3.4.3 Trình bày giải thuật . . . . .	41
3.4.4 Phối hợp phân mức điều khiển . . . . .	43
3.4.5 Phương pháp phối hợp phân mức điều khiển . . . . .	43
3.5 Thực nghiệm và đánh giá kết quả . . . . .	44
<b>4 Kết luận và tầm nhìn</b>	<b>45</b>
4.1 Kết luận . . . . .	45
4.2 Tầm nhìn . . . . .	45

# Danh sách hình vẽ

1.1	Robot công nghiệp [Nguồn: Internet] . . . . .	1
1.3	Mô hình hệ thống robot tự hành . . . . .	2
1.4	Virtual Force Field [1] . . . . .	6
1.5	Vector Field Histogram [2] . . . . .	7
1.6	Bóng bóng phản ứng [2] . . . . .	7
1.7	Phương pháp bóng bóng phản ứng động . . . . .	8
2.1	Ý tưởng cơ bản của <i>Markov localization</i> : Robot di động đang định vị trong không gian toàn cục [3] . . . . .	11
2.2	Hình (a): robot điều hướng qua môi trường mở, thiếu các đặc trưng trong không gian để có thể theo dõi được nó đang ở đâu (định vị). Hình (b): Vấn đề này có thể được tránh bằng việc đặt gần các vật cản đã biết. Hai hình này là kết quả của thuật toán <i>coastal navigation</i> [3] . . . . .	13
2.3	Các bản phân phối gần đây của ROS. Theo <a href="http://wiki.ros.org/Distributions">http://wiki.ros.org/Distributions</a> . . . . .	16
2.4	Hệ sinh thái ROS [4] . . . . .	17
2.5	ROS File System . . . . .	19
2.6	Truyền thông giữa hai node . . . . .	21
2.7	Kiểu giao tiếp topic . . . . .	22
2.8	Kiểu giao tiếp service . . . . .	22
2.9	Kiểu giao tiếp Action . . . . .	23
2.10	Dead Reckoning . . . . .	24
2.11	Kalman Filter[4] . . . . .	27
2.12	Online SLAM . . . . .	29
2.13	Full SLAM . . . . .	30
2.14	Quan hệ giữa khoảng cách tới vật cản và giá trị costmap . . . . .	31
2.15	Quá trình AMCL cho ước tính trạng thái vị trí robot . . . . .	32
2.16	Không gian tìm kiếm vận tốc và cửa sổ động . . . . .	33
2.17	Vận tốc dài v và vận tốc góc $\omega$ . . . . .	33

3.1	Nền tảng robot di động Dashgo D1 . . . . .	34
3.2	Cấu tạo phần chân đế . . . . .	35
3.3	Kiến trúc phần mềm điều khiển robot trên Dashgo D1 . . . . .	36
3.4	Sơ đồ bố trí cảm biến . . . . .	39
3.5	Dead reckoning [4] . . . . .	39
3.6	Cảm biến khoảng cách hồng ngoại . . . . .	40
3.7	Work-flow xử lý dữ liệu cảm biến . . . . .	41
3.8	Vùng xác định vật cản . . . . .	42
3.9	Flow-chart với vùng khẩn cấp . . . . .	42
3.10	Các topic điều khiển robot . . . . .	43
3.11	Thiết kế phân quyền điều khiển . . . . .	44

# Chương 1

## Tổng quan nghiên cứu

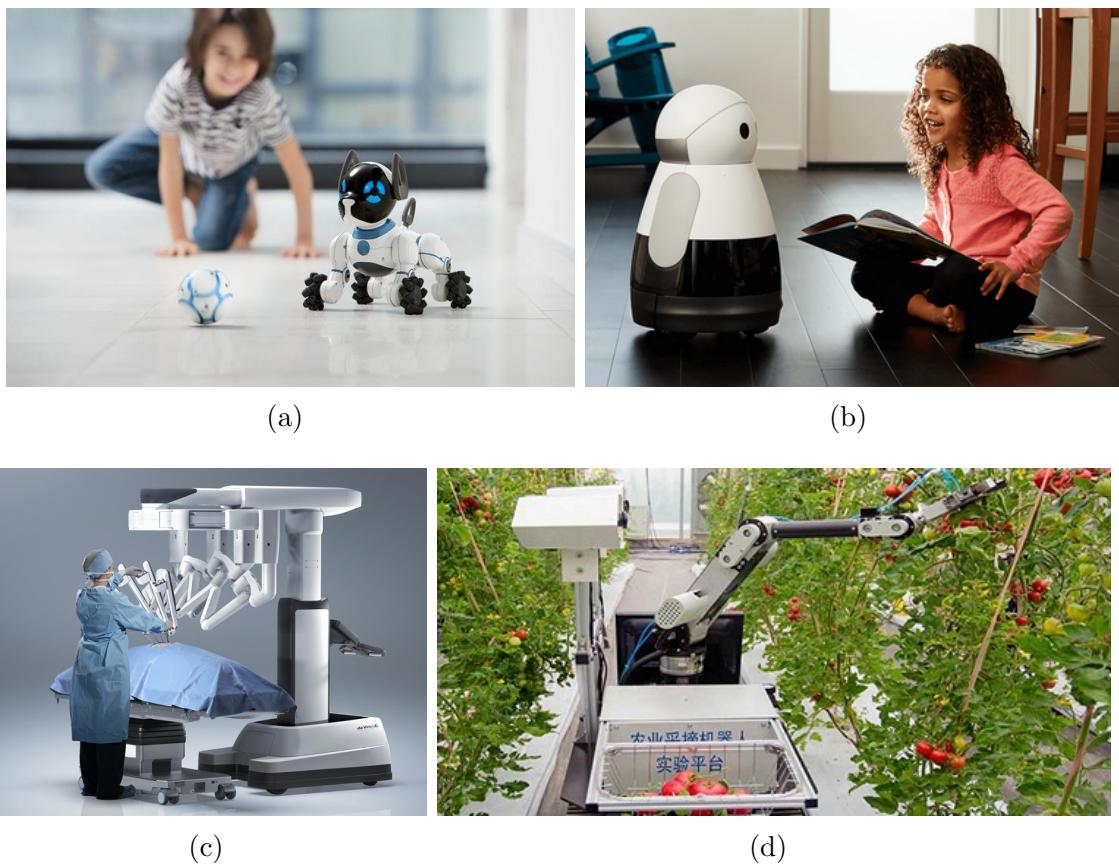
### 1.1 Giới thiệu robot tự hành

Người ta coi cánh tay robot là robot truyền thống. Bởi vậy khi nói đến thuật ngữ robot người ta thường nghĩ ngay đến cánh tay robot công nghiệp. Hình 1.1 là hình ảnh các cánh tay robot đang làm việc trong khâu sơn khung xe oto. Chúng ta không thấy bóng dáng con người ở trong bức hình này, bởi vì robot công nghiệp truyền thống phải làm việc trong không gian cách ly với con người, có hàng rào bảo vệ vì các lý do an toàn, con người không thể làm việc cùng không gian với robot.



Hình 1.1: Robot công nghiệp [Nguồn: Internet]

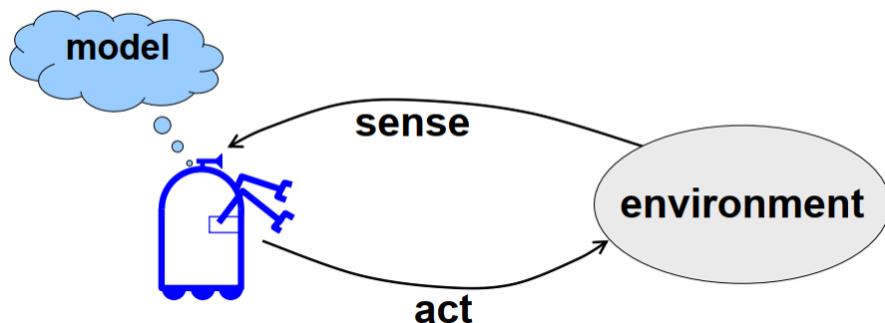
Bên cạnh đó, robot di động (mobile robot) truyền thống thực hiện các nhiệm vụ di chuyển trên quỹ đạo xác định trước. Robot di động cũng hoạt động dựa trên các khôi chương trình được lập trình sẵn. Các phương pháp điều khiển như điều khiển bằng tay thông qua bảng điều khiển, qua sóng RF, wifi... hay di chuyển bám đường chỉ dẫn gắn ở dưới sàn, đọc mã QR, bar. Robot dạng này bị hạn chế về không gian hoạt động. Việc thiết lập, cấu hình nhà máy, không



Hình 1.2: Một số loại robot mới

gian làm việc cho robot hoạt động hết sức tốn kém về chi phí và thời gian mà lại kém linh hoạt.

Ngày nay, robot có xu hướng đi ra khỏi không gian nhà máy, xuất hiện nhiều loại robot như: robot giải trí (Hình 1.2a), robot dịch vụ cá nhân (Hình 1.2b) (như máy tính cá nhân), robot trong y tế (Hình 1.2c); các loại robot tự động trong công nghiệp như robot hái quả (Hình 1.2d), phun thuốc trong nông nghiệp, robot thông minh tác hợp trong công nghiệp; robot đi tới các môi trường mà con người không tới được như trong lòng đất dưới nước, trên không, trong vũ trụ...



Hình 1.3: Mô hình hệ thống robot tự hành

Robot tự hành thông minh là loại robot di động, có thể cảm nhận, mô hình hóa môi trường xung quanh nó. Thực hiện các hành động di chuyển mà không cần sự giám sát cũng như điều khiển trực tiếp từ con người (Hình1.3).

## 1.2 Ứng dụng của robot tự hành thông minh

Với sự thông minh và cực kì linh hoạt của nó, robot tự hành thông minh có rất nhiều ứng dụng. Về cơ bản, nó là nền tảng di chuyển cho tất cả các loại robot di động thông minh ngày nay. Một số sản phẩm ứng dụng trong nhà như:

- Các ứng dụng trong nhà như robot hút bụi thông minh, robot dịch vụ, robot vận chuyển trong các kho hàng...
- Các ứng dụng ngoài trời như robot cắt cỏ, chăm sóc cây trồng...
- Ở các không gian mà con người không tới được như robot thám hiểm dưới nước, trong lòng đất, trên không trung trên các hành tinh khác...
- Tại các khu vực nguy hiểm như khu vực nhiễm chất phóng xạ, chữa cháy...
- Và đặc biệt, robot giao hàng tự động, xe tự lái đang phát triển rất nhanh trong những năm gần đây

## 1.3 Các bài toán trên robot tự hành thông minh

Một số bài toán chính trong robot tự hành như sau ([5]):

**Cảm biến:** Robot tự hành thông minh cần một số loại cảm biến để có thể hiểu được môi trường, định vị và di chuyển tránh vật cản.

Có rất nhiều loại cảm biến, để cảm nhận được đa dạng thông tin của môi trường. Có thể chia làm các nhóm như sau:

- Khoảng cách 1D: Cảm biến khoảng cách hồng ngoại, siêu âm
- Khoảng cách 2D: Lidar
- Cảm biến 3D: Có nhiều loại cảm biến 3D như Intel realsense, Microsoft Kinect, Asus Xtion...
- Ước tính trạng thái robot: GPS, IMU
- Cảm biến lực, momen, cảm biến chạm...
- Âm thanh, giọng nói như microphone, microphone array

- Các loại camera 2D

**Odometry:** là bài toán sử dụng thông tin nhận được từ các cảm biến nhận biết sự di chuyển của robot để ước tính sự thay đổi vị trí của robot qua thời gian. Odometry được sử dụng trong hầu hết các robot tự hành.

**Định vị:** Bài toán định vị giúp trả lời câu hỏi robot đang ở đâu, từ đó có cơ sở để thực hiện các tác vụ khác như tạo bản đồ, xác định hướng di chuyển.

**Xây dựng bản đồ:** Dự trên dữ liệu từ các loại cảm biến, từ odometry và định vị robot, robot sử dụng các thuật toán để xây dựng bản đồ của môi trường.

**Điều hướng robot:** Từ bài toán định vị và khi có được bản đồ và nhiệm vụ di chuyển từ vị trí hiện tại tới một vị trí đích. Robot sẽ tính toán quỹ đạo và điều khiển di chuyển tới đích.

**Tránh vật cản:** Trong quá trình di chuyển, robot phải phát hiện được các vật cản (bao gồm cả tĩnh và động) và tránh vật cản, sau đó thiết lập lại quỹ đạo di chuyển tới đích.

Vấn đề xuyên suốt trong các bài toán của robot tự hành thông minh đó là các dữ liệu đều không chắc chắn, các bài toán trên đều dựa vào các mô hình xác suất để mô hình hóa được trình bày trong tài liệu [?].

## 1.4 Các nghiên cứu tránh vật cản trong robot tự hành thông minh

Khả năng phát hiện và tránh vật cản theo thời gian thực là rất quan trọng trong robot tự hành. Do đó có rất nhiều nghiên cứu về giải pháp cho vấn đề này. Có nhiều loại cảm biến, nhiều giải thuật được sử dụng.

Các loại cảm biến sử dụng như cảm biến khoảng cách hồng ngoại, siêu âm với ứng dụng trên các thiết bị nhúng cấu hình thấp [6, 2]. Các phương pháp sử dụng cảm biến laser radar được trình bày trong [7, 8, 9, 10].

Các thuật toán phổ biến được dùng để phát hiện và tránh vật cản như Virtual Force Field (VFF) [11], Vector Field Histogram (VFH) [12], Dynamic Window Approach (DWA) [13]...

### 1.4.1 Thuật toán Virtual Force Field (VFF)

Được áp dụng cho điều khiển tránh vật cản trình bày trong [14, 11]. Ý tưởng của giải thuật này là tạo một ô lưới quanh robot. Khi có dữ liệu có vật cản từ cảm biến, ô tương ứng sẽ được đặt là bị chiếm dụng bởi vật cản với một tỉ số chiếm dụng (thể hiện cho sự không chắc chắn), nhiệm vụ của thuật toán là tính toán một lực để đưa robot xa khỏi ô bị chiếm dụng đó.

$$F(i, j) = \frac{F_{cr}C(i, j)}{d^2(i, j)} \left[ \frac{x_t - x_0}{d(i, j)} \dot{x} + \frac{y_t - y_0}{d(i, j)} \dot{y} \right] \quad (1.1)$$

Trong đó:

$F(i, j)$  Lực ảo chống lại vật cản tại ô (i,j)

$F_{cr}$  Hằng số lực chống lại vật cản

$d(i, j)$  Khoảng cách giữa ô (i,j) và robot

$C_{i,j}$  Độ chắc chắn tại ô (i,j)

$x_0, y_0$  Tọa độ robot

$x_i, y_j$  Tọa độ của ô (i,j)

Lực  $F_r$  đưa robot tránh khỏi các vật cản xuất hiện xung quanh robot là tổng của các lực tại mỗi ô lưới.

$$F_r = \sum_{i,j} F(i, j) \quad (1.2)$$

Trong khi đó, lực  $F_t$  kéo robot đi từ điểm hiện tại tới điểm đích như sau:

$$F_t = F_{ct} \left[ \frac{x_t - x_0}{d(t)} \dot{x} + \frac{y_t - y_0}{d(t)} \dot{y} \right] \quad (1.3)$$

Trong đó:

$F_{ct}$  Hằng số lực kéo robot tới đích

$d(t)$  Khoảng cách giữa robot và điểm đích

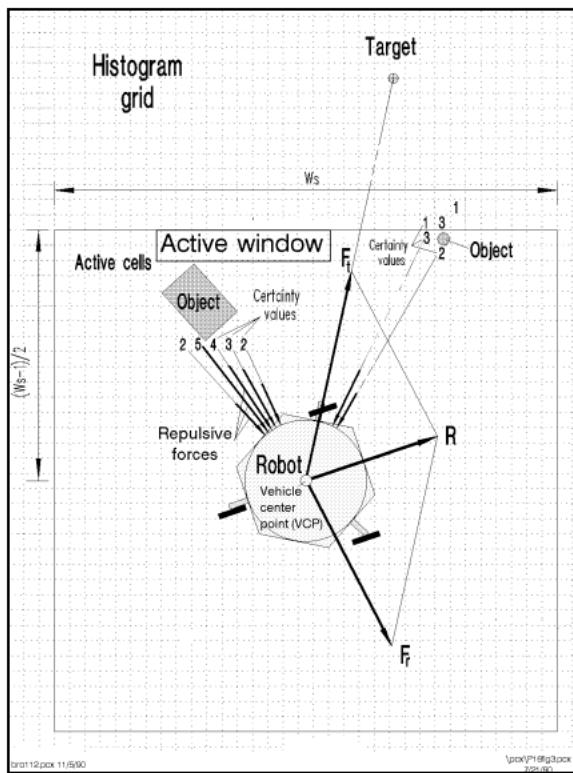
$x_t, y_t$  Tọa độ của điểm đích

Hợp lực của 2 loại lực này là lực chính kéo robot di chuyển.

$$R = F_t + F_r \quad (1.4)$$

Từ đó tính được hướng di chuyển của robot để điều khiển bánh xe di chuyển, đưa robot tới đích và tránh được vật cản. Theo [14], phương pháp này có một số ưu điểm như sau:

- Phương pháp này không xác định được đường viền cạnh của vật cản, nhưng có thể xác định được cụm các vị trí có vật cản.
- Phương pháp này không yêu cầu robot phải dừng lại để thực hiện lấy dữ liệu và tính toán. Trong điều kiện lý tưởng, phương pháp này giúp robot có thể tránh được tất cả vật cản trong khi vẫn di chuyển với vận tốc tối đa.
- Việc cập nhật bản đồ lưới và điều hướng sử dụng bản đồ lưới là hai nhiệm vụ hoàn toàn không phụ thuộc vào nhau, nhưng có thể đồng bộ với nhau để tối ưu tính toán.



Hình 1.4: Virtual Force Field [1]

- Phương pháp này có thể dễ dàng tích hợp nhiều loại cảm biến để bổ sung thông tin vào bản đồ ô lưới.

Tuy nhiên, theo [1], phương pháp này có một số điểm hạn chế như: Các trường hợp mắc bẫy do cực tiểu địa phương, không thể di chuyển qua giữa các vật cản gần nhau, lưỡng lự khi có vật cản, lưỡng lự trong lối đi hẹp

Do đó, phương pháp Vector Field Histogram cải thiện các hạn chế của phương pháp Vector Force Field

#### 1.4.2 The Vector Field Histogram

Phương pháp VFH [12] sử dụng kĩ thuật hai giai đoạn giảm dữ liệu và ba mức thể hiện dữ liệu:

- Mức biểu diễn dữ liệu cao nhất giữ mô tả chi tiết của môi trường robot, bản đồ lưới 2 chiều được cập nhật liên tục theo thời gian thực như trong phương pháp VFF.
- Mức ở giữa, một biểu đồ H một chiều được dựng quanh vị trí tức thời của robot. H bao gồm  $n$  góc với độ rộng  $\alpha$ , phép chuyển đổi từ C sang H.
- Mức biểu diễn dữ liệu thấp nhất là đầu ra của thuật toán VFH: các giá trị tham chiếu cho động cơ và bánh xe điều khiển robot.

Phương pháp này có thể phát hiện được lối đi đủ cho robot đi qua giữa các vật cản. Phương pháp VFF dễ bị ảnh hưởng bởi sai số của cảm biến, với phương pháp này, sử dụng làm mịn biểu đồ H đã giảm trọng số của các giá trị sai ngẫu nhiên của cảm biến, do đó phương pháp này mạnh hơn phương pháp VFF. Vẫn bị hiện tượng mắc bẫy khi vào các trường hợp "dead-end" (đường cùt) do sử dụng phương pháp cục bộ. Tốc độ di chuyển tối đa khi điều khiển robot bằng VFH bị giới hạn bởi tốc độ lấy mẫu của cảm biến.

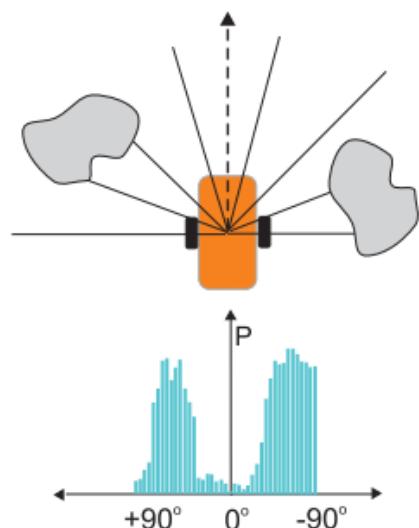
Dựa trên phương pháp này, [15] đã đề xuất phương pháp được gọi là VFH+ thực hiện 4 giai đoạn giảm dữ liệu từ bản đồ lưới chiếm dụng hai chiều xuống thành dữ liệu điều khiển hướng di chuyển của robot. Sử dụng một số cải tiến để giúp robot có thể di chuyển tốt hơn.

### 1.4.3 Phương pháp "Bong bóng phản ứng" tránh vật cản

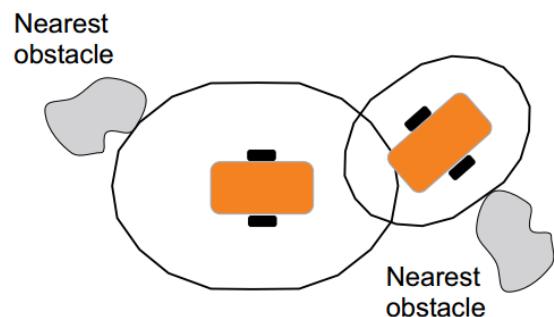
Phương pháp này được đề xuất trong [16], phương pháp này định nghĩa một "bong bóng" bao quanh robot thể hiện không gian lớn nhất có thể di chuyển mà không gặp vật cản. Hình dạng và kích thước của bong bóng được xác định đơn giản dựa trên hình dạng vật lý của robot và từ thông tin của cảm biến (như Hình 1.6).

Một phương pháp cải tiến được đề xuất trong [17]. Tác giả đã cải tiến hình dạng của bong bóng, không phải là hình dạng cố định như trong [16] mà hình dạng và kích thước được điều chỉnh liên tục dựa vào tốc độ di chuyển của robot. Khi các cảm biến phát hiện vật cản nằm trong giới hạn của bong bóng phản ứng, một thuật toán được dùng để xác định hướng của vật cản so với hướng di chuyển của robot Hình 1.7a. Một quá trình di chuyển của robot sử dụng phương pháp này được minh họa tại Hình 1.7b.

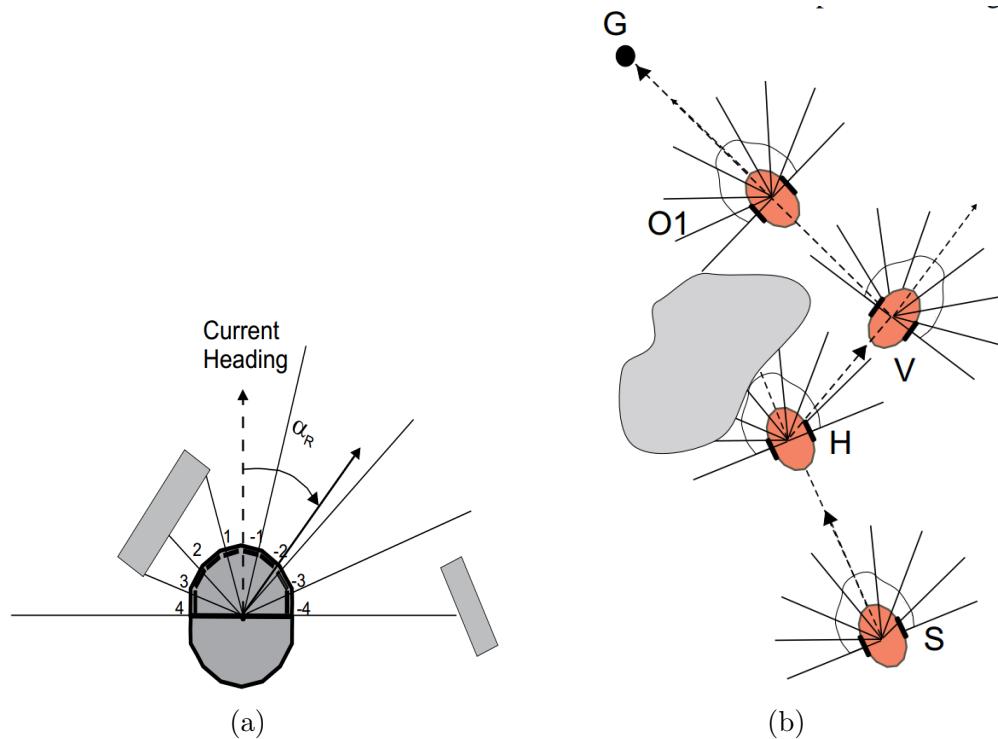
Cũng theo tác giả [17], phương pháp này có ưu điểm là có thể triển khai trên các thiết bị giá thành thấp, chi phí tính toán thấp nhưng có thể hoạt động theo



Hình 1.5: Vector Field Histogram [2]



Hình 1.6: Bong bóng phản ứng [2]



Hình 1.7: Phương pháp bóng phản ứng động

thời gian thực. Tuy nhiên, nó cũng có một số điểm hạn chế như quá trình di chuyển chưa mượt, robot khó xử lý khi gặp nhiều vật cản đồng thời ...

## 1.5 Nội dung nghiên cứu

Với mong muốn tiếp cận với công nghệ thế giới trong lĩnh vực robot tự hành, xe tự lái, luận văn này thực hiện 2 nhiệm vụ chính:

- Cấu hình robot ứng dụng SLAM trên nền tảng hệ điều hành ROS
- Tăng cường phát hiện và tránh vật cản cho robot bằng đa cảm biến.

# Chương 2

## Cơ sở lý thuyết

### 2.1 Bài toán về nhiễu trong robot tự hành

#### 2.1.1 Sự không chắc chắn trong robot

Robotics là một ngành khoa học về nhận thức và thực thi thế giới vật lý thông qua các thiết bị điều khiển bởi máy tính. Các hệ thống robotics được đặt trong thế giới vật lý, quan sát thông tin môi trường thông qua các cảm biến, và thao tác thông qua các lực vật lý. Đặc biệt trong công nghệ robot tự hành, robot phải có khả năng đáp ứng rất lớn sự không chắc chắn tồn tại trong thế giới vật lý.

Trước hết, môi trường của robot vốn dĩ đã không ổn định. Trong các môi trường có cấu trúc tốt, được thiết kế theo tiêu chuẩn như trong dây chuyền lắp ráp thì mức độ không ổn định thấp, trong khi đó các môi trường như trên đường, hay môi trường trong nhà của robot dịch vụ thì có mức độ không ổn định cao và nhiều chuyển động. Với các robot hoạt động gần con người thì độ không chắc chắn rất cao.

Robot sử dụng hệ thống cảm biến để cảm nhận, quan sát môi trường xung quanh. Tuy nhiên, cảm biến có các giới hạn quan sát của nó. Phạm vi và độ phân giải của một cảm biến phụ thuộc vào giới hạn vật lý. Ví dụ, các camera không thể nhìn xuyên tường, độ phân giải của camera cũng có giới hạn. Cảm biến rất khó tránh khỏi nhiễu, và các phép đo nhiễu không thể đoán trước được. Do đó nó giới hạn thông tin có thể trích xuất được từ cảm biến. Hoặc cảm biến có thể bị hỏng và việc phát hiện lỗi từ cảm biến vô cùng khó khăn.

Robot được dẫn động bằng động cơ, ở một mức độ nhất định nó cũng không dự đoán được sai số. Độ không chắc chắn sinh ra do ảnh hưởng từ nhiều điều khiển, mòn, hỏng và các lỗi cơ khí khác. Một vài cơ cấu chấp hành như cánh tay robot công nghiệp thường rất chính xác và độ tin cậy cao. Còn lại, như các robot di động giá thành thấp có thể rất dễ hỏng.

Nhiều nguyên nhân không chắc chắn có thể đến từ phần mềm của robot. Tất cả mô hình của thế giới vật lý đều gần đúng. Mô hình là trừu tượng hóa của thế giới thực, vì vậy, chúng ta chỉ có thể mô hình được một phần các quá trình vật lý cơ bản của robot và môi trường hoạt động của nó. Lỗi mô hình hóa dẫn đến sự không chắc chắn thường bị bỏ qua trong quá trình chế tạo robot, mặc dù thực tế là hầu hết các mô hình robot được sử dụng trong các hệ thống robot hiện đại khá sơ.

Sự không chắc chắn cũng có thể do các thuật toán gần đúng. Các hệ thống điều khiển robot có thể bị hạn chế lượng tính toán để đảm bảo robot có thể hoạt động theo thời gian thực. Đa số các thuật toán đều gần đúng, việc đảm bảo thời gian phản hồi có thể bị đánh đổi bởi độ chính xác.

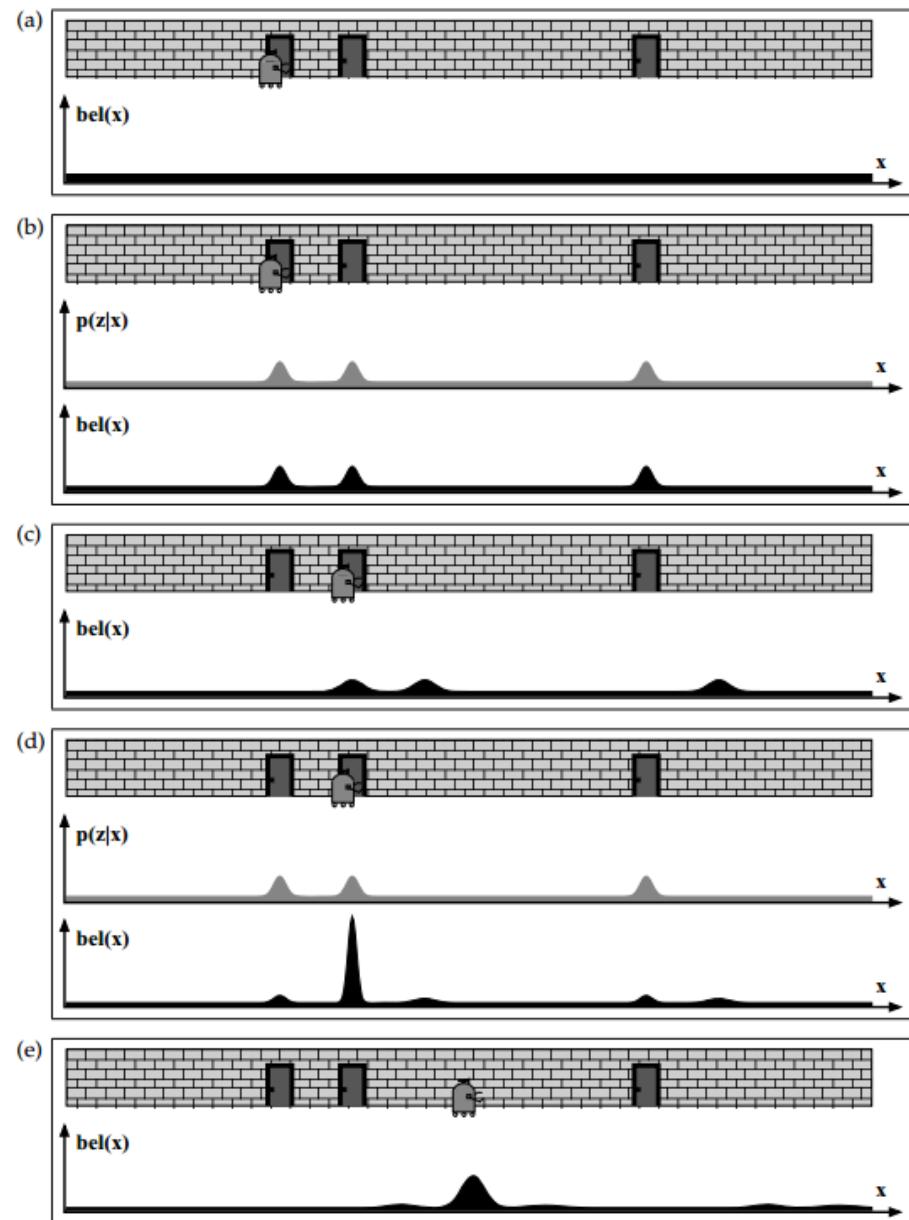
Mức độ không chắc chắn phụ thuộc vào lĩnh vực ứng dụng. Trong một vài lĩnh vực robotics, như dây chuyền lắp ráp độ không chắc chắn có thể chỉ là các yếu tố biên. Tuy nhiên, trong robot tự hành, mức độ không chắc chắn là rất lớn, nó đến từ việc hoạt động trong môi trường động, từ sai số của các cảm biến, từ sai số trong mô hình hóa chính robot ...

### 2.1.2 Xác suất trong robotics

Xác suất trong robot là tương đối mới với robot để giải quyết vấn đề trong quan sát và hành động của robot, đặc biệt là với robot tự hành. Ý tưởng chính trong bài toán xác suất trong robot là thể hiện sự không chắc chắn một cách rõ ràng bằng việc sử dụng lý thuyết tính toán xác suất. Nhưng thay vì dựa vào một "dự đoán tốt nhất" những gì có thể xảy ra, các thuật toán xác suất thể hiện thông tin bằng các phân bố xác suất qua toàn bộ không gian dự đoán. Như vậy, chúng có thể biểu diễn sự không rõ ràng và mức độ tin cậy trong toán học. Xác suất trong robot có thể chủ động lựa chọn để giảm sự không chắc chắn, do đó các thuật toán xác suất giảm độ phức tạp của sự không chắc chắn.

Ví dụ thứ nhất là bài toán định vị robot di động. Định vị robot là ước tính vị trí tương đối của robot với một không gian tham chiếu. Được cho một bản đồ của môi trường, nhưng việc tự nó định vị tới bản đồ này cần xem dữ liệu cảm biến của robot. Hình 2.1 là một ví dụ minh họa. Môi trường đã biết có ba cửa giống nhau, nhiệm vụ của robot là tìm xem chúng ở đâu, thông qua cảm biến và chuyển động.

Vấn đề định vị cụ thể này được gọi là *global localization* (định vị trên toàn cục). Trong global localization, robot được đặt ở đâu đó trong một môi trường đã biết, nó phải tự định vị được vị trí của nó đang ở đâu. Mô hình xác suất thể hiện độ tin cậy tức thời của robot bằng một hàm phân bố xác suất trên toàn bộ không gian như biểu đồ (a) trong Hình 2.1. Biểu đồ này thể hiện phân bố đồng



Hình 2.1: Ý tưởng cơ bản của *Markov localization*: Robot di động đang định vị trong không gian toàn cục [3]

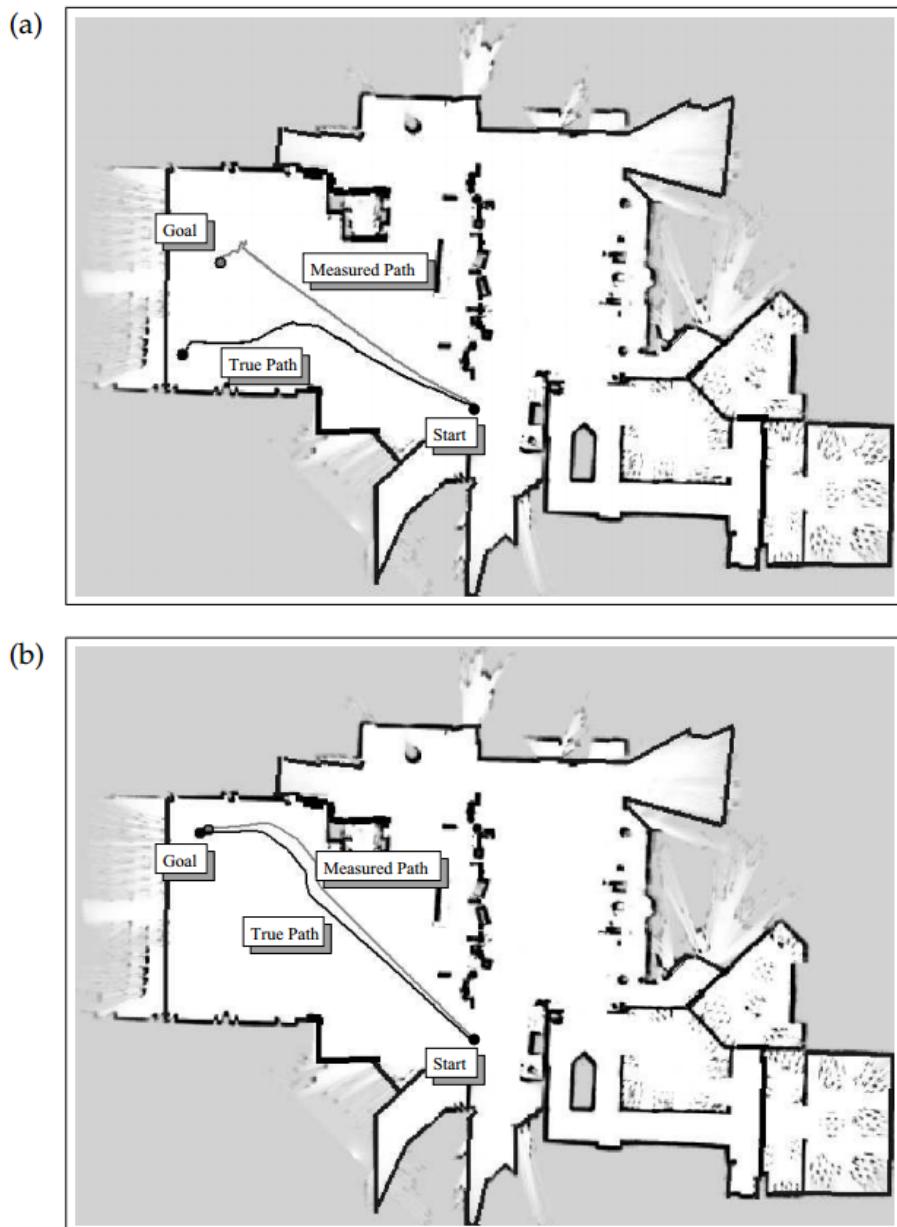
đều trên tất cả các vị trí. Bây giờ cho rằng robot nhận được dữ liệu đo đầu tiên từ cảm biến và theo dõi cánh cửa phía trước. Công cụ xác suất khai thác thông tin này để cập nhật vào độ tin cậy. **Độ tin cậy sau** thể hiện trong biểu đồ (b) trong Hình 2.1, tăng xác suất tại vị trí gần các cửa và giảm xác suất ở vị trí gần tường. Nhận thấy rằng phân bố này có ba chóp giống nhau, mỗi chóp tương ứng với một cửa. Do đó robot không biết được nó đang ở đâu.

Bây giờ cho robot di chuyển, biểu đồ (c) trong Hình 2.1 thể hiện ảnh hưởng đến độ tin cậy của robot. Độ tin cậy đã được dịch chuyển theo hướng chuyển động. Nó cũng trải ra một khoảng roojnghonw, thể hiện mức độ không chắc chắn thể hiện bởi sự dịch chuyển của robot. Hình (d) trong trong Hình 2.1 miêu tả độ tin cậy sau khi theo dõi một cửa khác. Sự theo dõi này dẫn đến thuật toán đặt phần lớn xác suất tại vị trí gần một cửa, và khi đó robot khá tự tin về việc nó đang ở đâu. Cuối cùng, biểu đồ (e) thể hiện độ tin cậy của robot sau khi di chuyển một đoạn xa trong hành lang.

Ví dụ này minh họa một vấn đề của mô hình xác suất. Nói theo xác suất, vấn đề quan sát của robot là một vấn đề ước tính trạng thái, ví dụ trên sử dụng một thuật toán có tên là *Bayes filter* cho hậu ước tính qua không gian của định vị robot. Sự thể hiện của thông tin alf một hàm phân bố xác suất. Việc cập nhật hàm này thể hiện thông tin giành được thông qua các phép đo của cảm biến, hoặc thông tin bị mất qua quá trình xử lý trong không gian làm tăng độ không chắc chắn của robot.

Ví dụ thứ hai là một ví dụ trong vấn đề lập kế hoạch và điều khiển robot. Như đã nói các thuật toán xác suất có thể tính độ không chắc chắn tức thời của robot. Nhưng chúng cũng có thể đoán trước được độ không chắc chắn trong tương lai, và chọn một độ không chắc chắn để xem xét sự lựa chọn điều khiển. Một trong các thuật toán vậy là *coastal navigation*, ví dụ như trong Hình 2.2. Hình này thể hiện một bản đồ 2-D của một tòa nhà. Hình phía trên so sánh đường đi ước tính và đường đi thực tế: Robot đi lệch là kết quả của sự không chắc chắn trong chuyển động của robot. Điểm thú vị ở đây là không phải toàn bộ quỹ đạo có mức độ không chắc chắn như nhau. Đường đi trong Hình 2.2a đi qua một không gian mở, thiết các đặc trưng để có thể giúp cho robot định vị. Hình 2.2b có quỹ đạo bám theo một góc rõ ràng, và sau đó ôm vào tường để giữ định vị. Không quá ngạc nhiên khi mức độ không chắc chắn sẽ giảm sau một quãng di chuyển, do đó điểm đến có độ chính xác cao hơn.

Ví dụ này minh họa một trong những cách để xem xét ảnh hưởng của sự không chắc chắn trong điều khiển robot. Rõ ràng robot sẽ ưu tiên lựa chọn cách đi thứ hai hơn, dù có sự không chắc chắn nhưng thuật toán xác suất giúp robot lựa chọn đường đi để nó có thể thu thập được nhiều thông tin, giúp giảm độ không chắc chắn và đạt được độ chính xác tốt hơn.



Hình 2.2: Hình (a): robot điều hướng qua môi trường mở, thiếu các đặc trưng trong không gian để có thể theo dõi được nó đang ở đâu (định vị). Hình (b): Vấn đề này có thể được tránh bằng việc đặt gần các vật cản đã biết. Hai hình này là kết quả của thuật toán *coastal navigation* [3]

Xác suất trong robotics kết hợp với dữ liệu cảm biến và khắc phục hạn chế của cả hai. Ý tưởng này không chỉ là vấn đề của điều khiển mức thấp, chúng có mặt tại mọi mức phần mềm robot từ thấp nhất tới cao nhất.

Trái ngược với các kĩ thuật lập trình truyền thống trong robot như các công cụ kế hoạch chuyển động dựa trên mô hình hay phản ứng dựa trên hành vi. Các cách tiếp cận theo xác suất có nhiều ưu điểm hơn với các giới hạn của cảm biến và mô hình hóa. Điều này giúp gần hơn với độ phức tạp của môi trường thế giới thực hơn là mô hình cũ. Trong thực tế, chắc chắn các thuật toán xác suất hiện nay chỉ mới biết các giải pháp với các vấn đề ước tính khó trong robotics như: vấn đề định vị, vấn đề xây dựng các bản đồ chính xác môi trường lớn.

So sánh với các công cụ robotics truyền thống dựa trên mô hình hóa, các thuật toán xác suất có yêu cầu thấp hơn về độ chính xác của các mô hình robot do đó giúp các nhà lập trình thoát khỏi gánh nặng không thể vượt qua để đưa ra các mô hình chính xác. Các thuật toán xác suất có yêu cầu thấp hơn về độ chính xác của các cảm biến hơn là các kĩ thuật dựa trên phản ứng, các kĩ thuật này phản ứng dựa trên dữ liệu cảm biến tức thời. Nhìn chung bài toán xác suất trong robot, việc học của robot là vấn đề ước lượng dài hạn.

Tuy nhiên, những ưu điểm này cũng đi kèm với cái giá của nó. Hai hạn chế thường xuyên được nhắc tới đó là độ phức tạp tính toán và việc phải tính xấp xỉ. Các thuật toán xấp xỉ vốn đã kém hiệu quả hơn so với các thuật toán không xác suất. Do thuật toán này xem xét trên toàn bộ phân bố chứ không phải tại một lần đoán. Hầu hết các loại robot đều hoạt động liên tục yêu cầu các thuật toán này phải tính toán được theo thời gian thực. Một số bài toán có thể giải quyết bằng một số mô hình đơn giản (như Gaussians) nhưng một số bài toán khác yêu cầu phải có các mô hình phức tạp hơn.

Việc phát triển của phần cứng máy tính ngày nay làm tăng số lượng phép tính trên một đơn vị giá. Điều này giúp cho lĩnh vực xác suất robot có hiệu quả hơn, thực hiện được các bài toán khó hơn. Tuy nhiên vẫn còn đó thách thức tính toán của lĩnh vực này. [3]

## 2.2 Hệ điều hành robot ROS và các ứng dụng

### 2.2.1 ROS là gì?

Hệ điều hành Robot – Robot Operating System (ROS) là một nền tảng mã nguồn mở phục vụ cho việc lập trình Robot, có thể cài đặt trên nhiều hệ điều hành khác nhau như Windows, Linux hay Mac OS. ROS mang đến một nền tảng phần mềm chung cho cộng đồng xây dựng và sử dụng Robot. Ở đó, mọi người có thể chia sẻ các ý tưởng và các trình điều khiển dễ dàng hơn. ROS đã

đạt được rất nhiều thành tựu. Kể từ khi ra đời, đã có hơn 2000 các gói phần mềm được viết và duy trì bởi gần 600 người. Gần 80 loại robot thương mại được hỗ trợ và hàng nghìn các bài báo đề cập đến ROS. Chính vì thế, chúng ta đã không còn phải làm tất cả mọi thứ từ đầu khi xây dựng Robot nhờ có sự trợ giúp của ROS. Chúng ta có thể dành nhiều thời gian hơn để nghiên cứu về Robotics thay vì tập trung quá nhiều vào xây dựng các trình điều khiển phần cứng. ROS bao gồm tập hợp đa dạng các trình điều khiển cho phép chúng ta đọc dữ liệu từ cảm biến, điều khiển các cơ cấu chấp hành; một lượng lớn các thuật toán cho phép xây dựng bản đồ, điều hướng Robot, thu thập dữ liệu, lập kế hoạch di chuyển... ROS cũng có một cộng đồng lớn nghiên cứu trong lĩnh vực Robot.

Nói cách khác, theo tác giả, ROS là một nền tảng cung cấp các phương thức để kết nối, trao đổi dữ liệu, quan sát dữ liệu giữa rất nhiều phần cứng với nhau như các cảm biến, các bộ phận chấp hành và các máy tính. Khi sử dụng ROS, người phát triển robot không cần quan tâm quá nhiều đến các thiết bị phần cứng (nếu thiết bị đó đã có thư viện driver ROS hỗ trợ), chỉ cần quan tâm đến việc tính toán, xử lý các dữ liệu để đạt được các mục đích trong các ứng dụng khác nhau. Nó làm giảm thời gian, giảm độ phức tạp khi phát triển robot đi rất nhiều lần.

### 2.2.2 Tổng quan về HĐH ROS

ROS là một nền tảng mã nguồn mở, một software framework phục vụ cho việc lập trình Robot. ROS cung cấp một hệ thống điều khiển phân tán cho các thiết bị phần cứng khác nhau, được dùng để xây dựng các ứng dụng cho Robot mà không cần phải quan tâm quá sâu về các phần cứng bên trong. ROS cung cấp các công cụ đa dạng để mô phỏng, mô hình hóa Robot và xử lý dữ liệu của Robot, sử dụng giao thức message-passing đồng bộ hoặc không đồng bộ để liên kết và trao đổi dữ liệu từ những thiết bị khác nhau. Phần mềm được tổ chức thành các gói, rất thuận tiện, với giao thức message-passing giữa các thiết bị phần cứng, nhà phát triển có thể tạo ra các mô hình robot rất đa dạng, ví dụ như tạo bản đồ và điều hướng với Mobile Robot. Các robot mới có thể trực tiếp sử dụng các gói này mà không cần phải chỉnh sửa source code. ROS cũng được sử dụng rộng rãi trong các trường Đại học, và có nhiều nhà phát triển đóng góp. Có thể nói rằng, ROS có một cộng đồng lớn, có nhiều dự án được hỗ trợ bởi cộng đồng phát triển toàn cầu. Có một hệ sinh thái thân thiện với các mô hình phát triển Robot khác. Tóm lại, ROS là sự kết hợp của mô hình truyền thông (Plumbing or communication), các công cụ (Tools), khả năng phần cứng (Capabilities) và hệ sinh thái (Ecosystem).

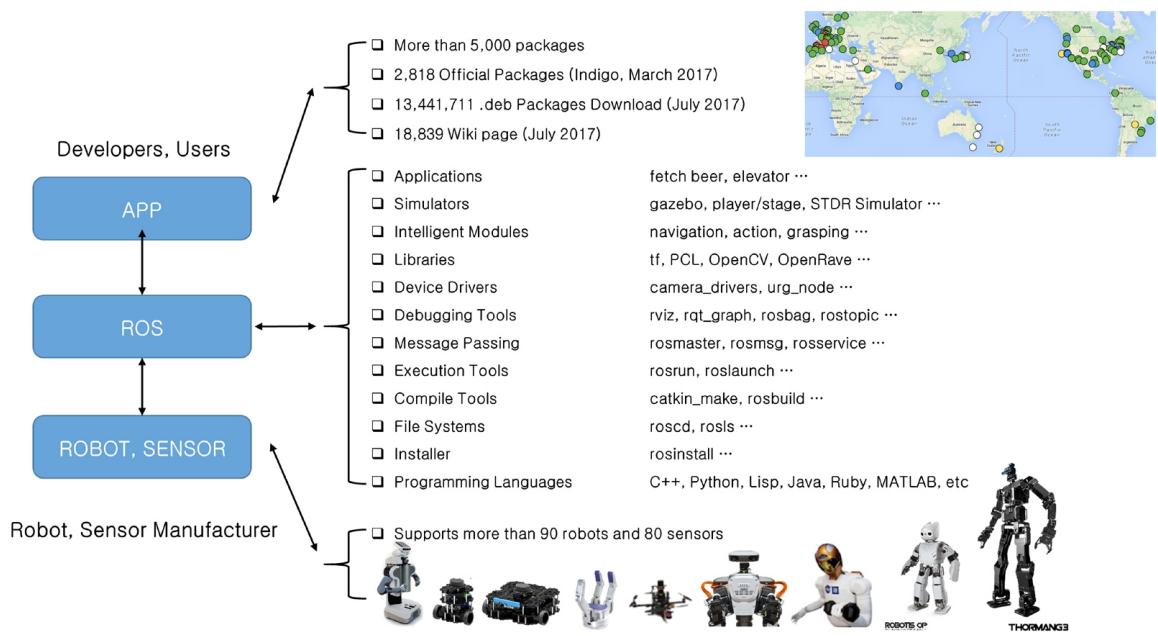
ROS có nhiều bản phân phối khác nhau tương ứng với các bản phân phối của

Distro	Release date	Poster	Tuturtle, turtle in tutorial	EOL date
ROS Noetic Ninjemy	May, 2020 (planned, see <a href="#">Upcoming Releases</a> )	TBA	TBA	May, 2025 (planned)
ROS Melodic Morenia <b>(Recommended)</b>	May 23rd, 2018			May, 2023 (Bionic EOL)
ROS Lunar Loggerhead	May 23rd, 2017			May, 2019
ROS Kinetic Kame	May 23rd, 2016			April, 2021 (Xenial EOL)
ROS Jade Turtle	May 23rd, 2015			May, 2017
ROS Indigo Igloo	July 22nd, 2014			April, 2019 (Trusty EOL)
ROS Hydro Medusa	September 4th, 2013			May, 2015

Hình 2.3: Các bản phân phối gần đây của ROS. Theo <http://wiki.ros.org/Distributions>

Linux và mỗi bản phân phối hỗ trợ cập nhật đến hết vòng đời. ROS tương thích hoàn toàn với hệ điều hành Ubuntu, phần lớn các bản phân phối của ROS được phát triển theo các phiên bản của Ubuntu. Hình 2.3

ROS như một hệ sinh thái, bao gồm từ các loại phần cứng thường dùng trong robot, các phần mềm, các nhà sản xuất và các nhà nghiên cứu<sup>1</sup>. Hình 2.4 thể hiện hệ sinh thái ROS.



Hình 2.4: Hệ sinh thái ROS [4]

### 2.2.3 Tại sao phải dùng ROS

Hình dung rằng chúng ta đang xây dựng một robot tự hành thông minh. Chúng ta lựa chọn ROS hơn là các nền tảng robotic khác như Player, YARP, Orocos, MRPT... bởi các lý do như sau:

- **Có độ sẵn sàng cao:** ROS có thể sử dụng được luôn, ví dụ như các gói thư viện **SLAM** (**Simultaneous Localization and Mapping** - tạm dịch là **định vị và tạo bản đồ đồng thời**) và **AMCL** (**Adaptive Monte Carlo Localization** - Là **thuật toán định vị** **thích nghi** **Monte Carlo**) trong ROS có thể được sử dụng cho mobile robot di chuyển tự động và gói **MoveIt** được sử dụng để lập kế hoạch chuyển động tay máy robot. Các gói thư viện này có thể dễ dàng sử dụng trực tiếp cho phần mềm robot của chúng ta. Việc sử dụng các gói thư viện này tốt hơn rất nhiều so với việc lập trình lại từ đầu để được thứ tương tự với một thứ đã có sẵn. Những gói chương

<sup>1</sup>Chi tiết các loại phần cứng, phần mềm ROS hỗ trợ tại <http://wiki.ros.org/>

trình này cũng rất dễ cấu hình lại, chúng ta có thể tinh chỉnh từng khả năng bằng cách sử dụng nhiều thông số khác nhau.

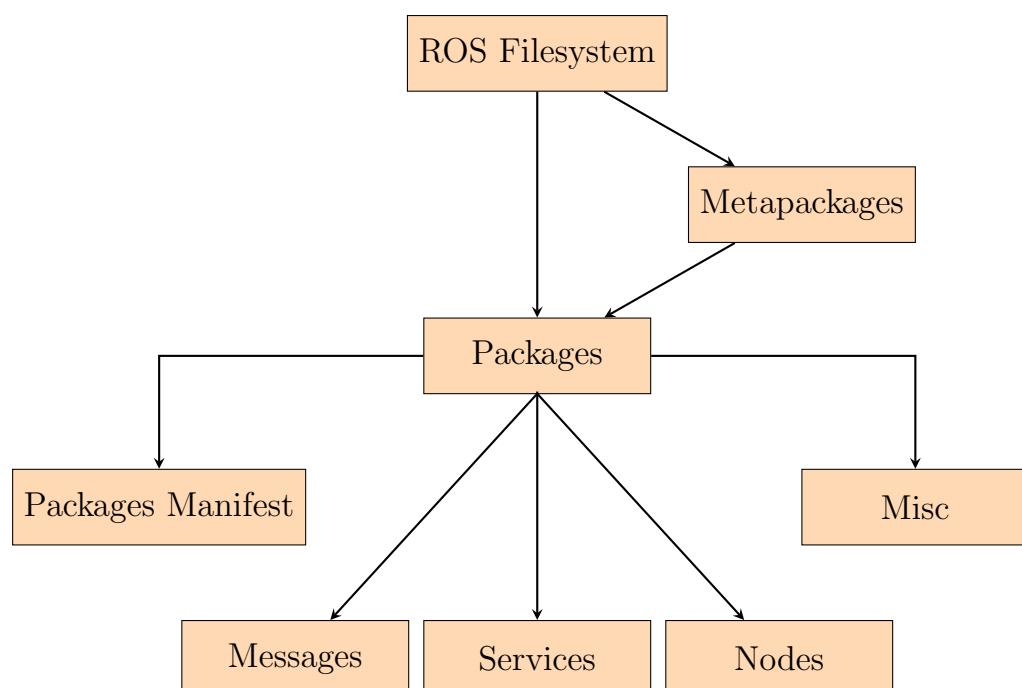
- **Rất nhiều công cụ:** ROS được đóng gói với rất nhiều công cụ để gõ rối chương trình, hiển thị và mô phỏng các quá trình. Các công cụ như là rqt\_gui, RViz, Gazebo là một số công cụ mã nguồn mở rất mạnh cho việc gõ rối, hiển thị và mô phỏng.
- **Hỗ trợ cao cho các cảm biến và cơ cấu chấp hành:** ROS được đóng gói với các gói driver và giao tiếp với các thiết bị của rất nhiều cảm biến và cơ cấu chấp hành trong robotics. Các cảm biến cao cấp bao gồm Lidar, Laser scanners, Kinect... và các cơ cấu chấp hành như động cơ servo Dynamixel. Chúng ta rất dễ để có thể giao tiếp với các phần cứng này với ROS.
- **Khả năng phân tích nền tảng bên trong:** Việc truyền thông điệp (message) ROS ở tầng giữa cho phép giao tiếp giữa các node (node) với nhau. Các node này có thể được lập trình bằng bất kì ngôn ngữ nào có hỗ trợ thư viện ROS (như rospy trong python, roscpp trong C++). Chúng ta có thể viết các node trong C++ hoặc C và các node khác bằng Python hoặc Java. Điều này rất linh hoạt mà không có trong bất kì nền tảng nào khác.
- **Module hóa:** Một trong những vấn đề có thể xảy ra trong phần lớn các ứng dụng robot đơn lẻ đó là, nếu bất kì luồng nào trong chương trình chính bị hỏng, toàn bộ ứng dụng robot đó có thể dừng lại. Trong ROS, trường hợp này sẽ khác, chúng ta viết các node khác nhau cho mỗi tiến trình và nếu một node bị hỏng, hệ thống vẫn có thể tiếp tục làm việc. ROS cũng cung cấp các phương pháp rất hữu ích để tiếp tục hoạt động một khi có bất kì cảm biến hoặc động cơ nào bị hỏng.
- **Điều khiển các nguồn tài nguyên đồng thời:** Việc sử dụng một nguồn tài nguyên phần cứng bởi nhiều hơn hai tiến trình luôn là một vấn đề đau đầu. Hãy hình dung, chúng ta muốn xử lý một hình ảnh từ một camera cho nhận dạng khuôn mặt và nhận dạng chuyển động, chúng ta có thể viết chương trình như một chương trình thực hiện toàn bộ cả hai vấn đề đó, hoặc chúng ta có thể viết riêng để thực hiện đồng thời. Nếu chúng ta muốn thêm nhiều hơn hai tính năng trong các luồng, hành vi ứng dụng sẽ trở nên phức tạp và rất khó để gõ rối. Nhưng trong ROS, chúng ta có thể truy cập các thiết bị bằng cách sử dụng các chủ đề ROS từ các bộ điều khiển ROS (driver). bao nhiêu node cũng có thể đăng ký tới thông điệp hình ảnh từ bộ điều khiển camera ROS và mỗi node có thể thực hiện các chức năng khác nhau. Nó có thể làm đơn giản tính toán và cũng như tăng khả năng gõ rối của toàn bộ hệ thống.

- Có một cộng đồng năng động:** Khi chúng ta lựa chọn một thư viện hoặc một nền tảng phần mềm, đặc biệt từ một cộng đồng mã nguồn mở, một trong những yếu tố chính là xem xét tới phần mềm hỗ trợ và cộng đồng các nhà phát triển sử dụng nó. Mã nguồn mở thì sẽ không đảm bảo có các công cụ hỗ trợ, một số công cụ có sự cung cấp hỗ trợ rất tốt nhưng ngược lại một số khác lại không. Trong ROS, cộng đồng hỗ trợ rất năng động. Có thể vào trang web sau để tham khảo sự hỗ trợ từ các người dùng khác: <http://answers.ros.org>. Đường như cộng đồng ROS có sự tăng trưởng đều đặn của các nhà phát triển trên toàn thế giới.

Có rất nhiều lý do để lựa chọn ROS cho việc phát triển robot.

#### 2.2.4 Một số thành phần cơ bản trong ROS

Một hệ thống ROS thông thường bao gồm những thành phần cơ bản như thể hiện trong Hình 2.5.



Hình 2.5: ROS File System

Một số thuật ngữ, thành phần cơ bản trong ROS:

**Node** là đơn vị xử lý nhỏ nhất đang chạy trong ROS. Có thể coi nó như một chương trình thực thi. ROS khuyên rằng nên tạo một node đơn cho mỗi mục đích và nên phát triển để có thể dễ dàng sử dụng lại. Ví dụ, trong robot di động, chương trình để vận hành robot được chia thành các hàm chức năng nhỏ. Mỗi node được dùng cho một hàm chức năng như driver cảm biến, biến đổi dữ liệu cảm biến, nhận dạng vật cản, driver động cơ, đầu vào encoder và định hướng.

Khi khởi động, một node ghi thông tin như tên, dạng thông điệp, địa chỉ URI và số cổng của node. node đã được ghi có thể hoạt động như một node xuất bản, node đăng ký, node chủ dịch vụ hoặc node khách dịch vụ dựa trên thông tin đã được ghi, và các node có thể trao đổi thông điệp bằng cách sử dụng chủ đề hoặc dịch vụ.

node sử dụng XMLRPC cho việc giao tiếp với node chủ và sử dụng XMLRPC hoặc TCPROS của giao thức TCP/IP khi giao tiếp giữa các node với nhau. Yêu cầu kết nối và phản hồi giữa các node sử dụng XMLRPC và truyền thông điệp sử dụng TCPROS bởi vì nó là giao tiếp trực tiếp giữa các node với nhau mà không phụ thuộc vào node chủ. Với địa chỉ URI và số cổng, một biến được gọi là `ROS_HOSTNAME` được lưu trên máy tính, nơi node đang chạy, được sử dụng địa chỉ URI và cổng được đặt bằng một giá trị duy nhất bất kì.

**Master** có là một node đặc biệt, nó hoạt động như một máy chủ cho các kết nối node tới node và truyền thông điệp. Lệnh `roscore` được dùng để chạy node chủ, và nếu chạy node chủ, nó sẽ ghi tên của mỗi node và lấy thông tin khi cần thiết. Các hoạt động của ROS không được thực hiện khi chưa chạy master.

Giao tiếp giữa node chủ và các node khách bằng XMLRPC (Viết tắt của XML-Remote Produce Call: nghĩa là gọi hàm từ xa XML), là giao thức dựa trên HTTP không duy trì kết nối. Nói cách khác, các node khách chỉ có thể truy cập khi chúng cần ghi thông tin của chúng hoặc yêu cầu thông tin của các node khác. Trạng thái kết nối không được kiểm tra thường xuyên. Vì đặc điểm này, ROS có thể được sử dụng trong các môi trường rất lớn và phức tạp. XMLRPC rất nhẹ và hỗ trợ rất nhiều ngôn ngữ lập trình khác nhau, rất phù hợp để sử dụng cho ROS vì ROS hỗ trợ rất nhiều phần cứng và nhiều ngôn ngữ lập trình. **Packages:** Gói chương trình là đơn vị chính trong tổ chức phần mềm của hệ điều hành ROS. Một package có thể chứa các lệnh thực thi của ROS (các node), thư viện, các tệp chứa thông số... Package chính là thành phần nguyên tử nhỏ nhất được xây dựng và đưa vào sử dụng trong ROS.

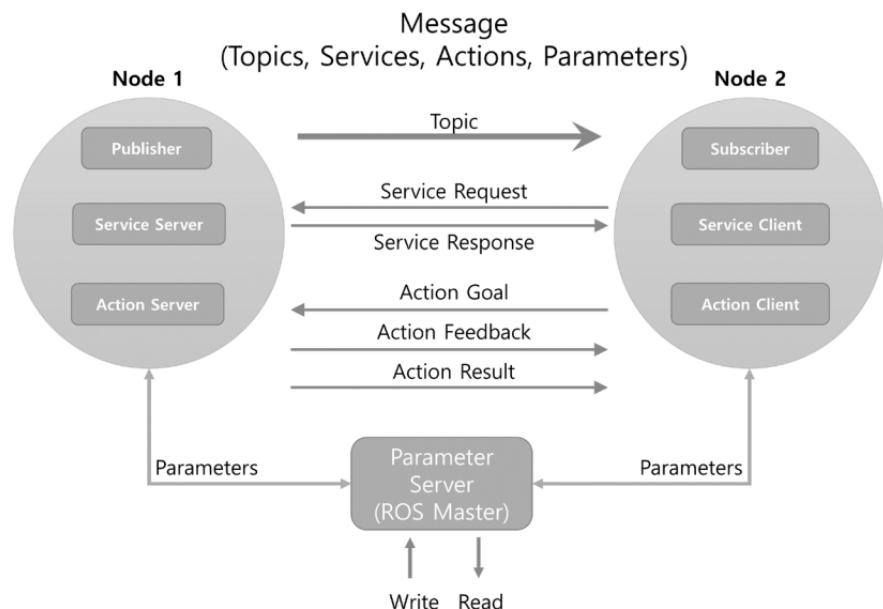
**Packages Manifest:** Bảng kê khai thông tin dữ liệu của package (`package.xml`), cung cấp siêu dữ liệu về package đó bao gồm tên gọi, phiên bản, thông tin bản quyền (`license`) và những yếu tố phụ thuộc của gói dữ liệu đó. Manifest còn chứa thông tin về đặc trưng của ngôn ngữ lập trình ví dụ như các cờ báo (`flags`) của trình biên dịch.

**Message** hay còn gọi là thông điệp, thông tin. Một node gửi hoặc nhận dữ liệu giữa các node thông qua một thông điệp. Các thông điệp là cá biến như số nguyên (`integer`), điểm số thực (`floating point`), hay logic (`boolean`). Có thể sử dụng thông điệp lồng vào các thông điệp hoặc một mảng các thông điệp khác.

Giao thức truyền thông TCPROS và UDPROS được sử dụng để truyền thông điệp. Chủ đề (`topic`) được sử dụng để truyền thông điệp đơn hướng trong khi

dịch vụ (service) được dùng để truyền thông điệp đa hướng bao gồm yêu cầu và phản hồi.

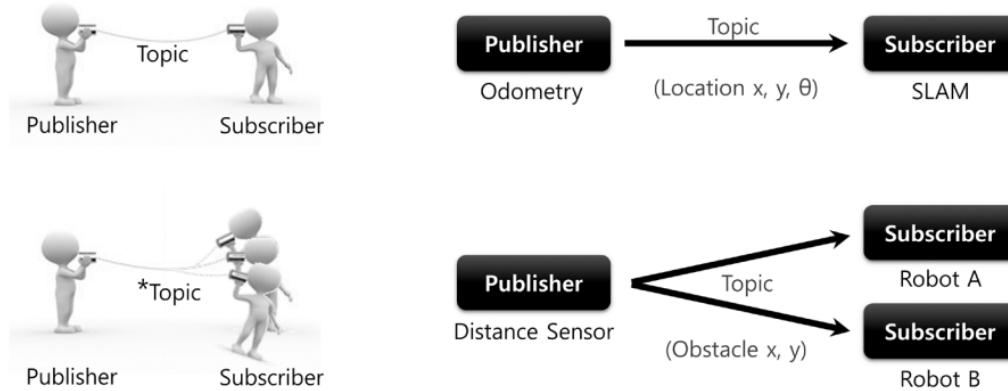
**Truyền thông tin:** ROS được phát triển dựa trên các đơn vị node, đơn vị nhỏ nhất để thực thi các chương trình đã chia nhỏ để có thể dễ dàng tái sử dụng theo các mục đích. Node trao đổi dữ liệu với nhau thông qua thông điệp message để tạo thành một chương trình lớn. Các node sẽ giao tiếp với nhau bằng cách trao đổi các thông điệp. Thông điệp có thể là các kiểu dữ liệu đơn giản như số nguyên, số thực, kí tự, biến logic... hoặc cũng có thể là tổ hợp của các thông điệp khác như: tọa độ của một điểm là tổ hợp của 3 số thực. Trong thực tế, ROS có rất nhiều loại thông điệp từ đơn giản đến phức tạp để phục vụ cho việc xử lý dữ liệu đa dạng, ví dụ như: thông điệp hình ảnh, thông điệp là dữ liệu từ một cảm biến, thông điệp là vị trí và vận tốc của robot... Có 3 phương pháp để trao đổi dữ liệu: **topic**, **service**, **action**. Ngoài ra, các thông số được sử dụng trong node có thể được thay đổi từ bên ngoài node. Điều này có thể được xem như một dạng của sự truyền thông điệp với nội dung lớn hơn. Sự truyền thông điệp được mô tả trong Hình 2.6



Hình 2.6: Truyền thông giữa hai node

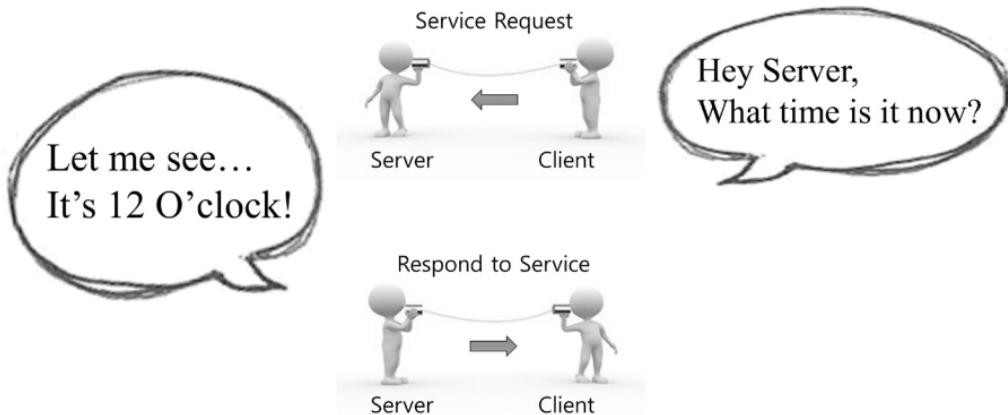
**Topic** hay còn gọi là chủ đề, giống như nghĩa đen là chủ đề trong một đoạn hội thoại. node xuất bản đầu tiên ghi thông tin chủ đề của nó với node chủ và sau đó bắt đầu xuất bản thông điệp trên một chủ đề. Các node đăng ký muốn nhận chủ đề yêu cầu thông tin của node đăng ký khớp với tên của chủ đề đã được ghi ở trên node chủ. Dựa vào thông tin này, node đăng ký kết nối trực tiếp tới node xuất bản để trao đổi thông điệp như một chủ đề. Hình 2.7

**Publish và Publisher** Hay còn gọi là xuất bản và node xuất bản. Xuất bản



Hình 2.7: Kiểu giao tiếp topic

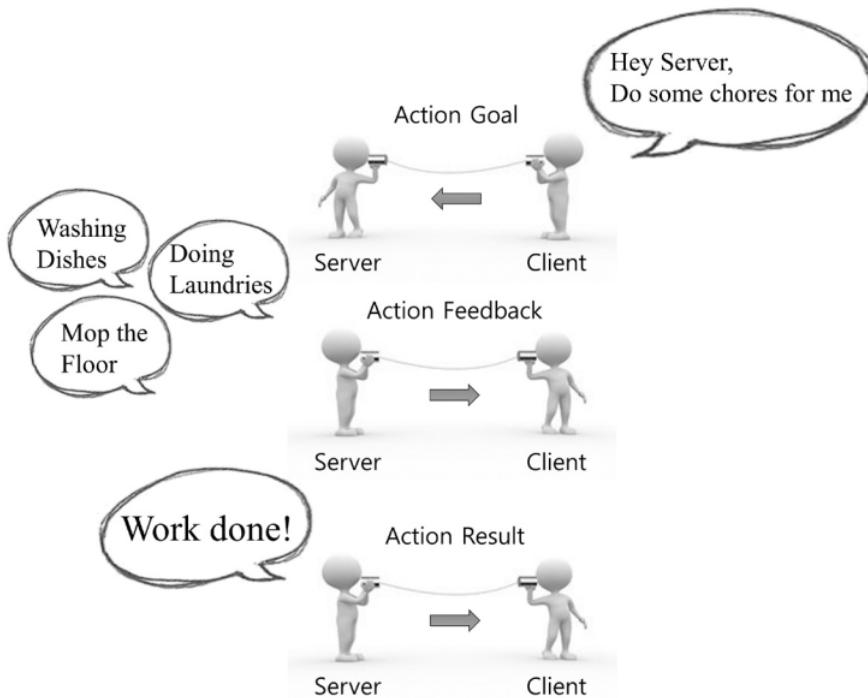
là thuật ngữ dành cho hành động chuyển các thông điệp liên quan tới đúng chủ đề. node xuất bản ghi thông tin của nó và chủ đề với node chủ, và gửi một thông điệp tới các node đăng ký, là các node quan tâm tới cùng chủ đề. **Subscribe** và **Subscriber**: Thuật ngữ 'Subscribe', còn gọi là Đăng ký, thể hiện hành động của việc nhận các thông điệp khớp với topic. node Subscriber đăng ký thông tin của nó và chủ đề với master, và nhận thông tin của các node xuất bản.



Hình 2.8: Kiểu giao tiếp service

**Service:** Giao tiếp đồng bộ giữa service client và service server. Client sẽ yêu cầu (request) 1 service và service server sẽ phản hồi lại yêu cầu đó. Khác với mô hình publish – subscribe, là một phương pháp không đồng bộ gấp khẩn trong các phương pháp truyền dữ liệu theo chu kỳ, mô hình request and response là 1 phương pháp đồng bộ, trong trường hợp này là Service Hình 2.8.

**Action:** Được sử dụng khi một nhiệm vụ thực hiện cần nhiều thời gian để hoàn thành cần 1 quá trình phản hồi (feedback). Trong action, **goals** và **results** sẽ tương tự như **request** (yêu cầu) và **response** (đáp ứng), ngoài ra còn có thêm



Hình 2.9: Kiểu giao tiếp Action

feedback là tín hiệu phản hồi đến client theo chu kì

## 2.3 Bài toán điều hướng robot

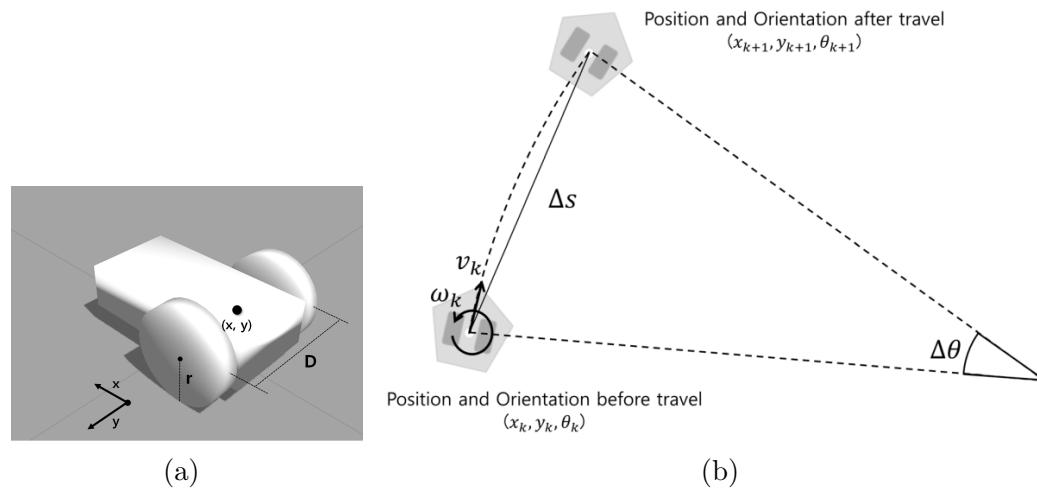
Giống như chúng ta sử dụng GPS để định vị và tìm đường trong quá trình di chuyển giữa thành phố đông đúc, rộng lớn, hoặc khi ta đi tới một địa điểm xa lạ. Khi đó chúng ta sử dụng các ứng dụng trên điện thoại di động như googlemap để định vị vị trí của mình, sau đó tìm đường tới địa điểm mong muốn. Tương tự như vậy, robot tự hành cũng cần định vị, và xác định đường đi tới điểm đích.

### 2.3.1 Điều hướng robot di động

Điều hướng không thể thiếu trong robotics. Điều hướng là sự di chuyển của robot tới một đích xác định, điều này không dễ dàng với robot. Robot phải biết được nó đang ở đâu và phải có một bản đồ của môi trường. Robot phải tìm được đường đi tối ưu trong rất nhiều đường đi khác, tránh vật cản như tường, đồ vật. Để có thể điều hướng robot, cần những yếu tố sau:

- Bản đồ
- Trạng thái của robot<sup>2</sup>

<sup>2</sup>Trong luận văn này, khi nhắc đến trạng thái robot có nghĩa là vị trí và hướng của robot



Hình 2.10: Dead Reckoning

- Cảm nhận thông tin từ cảm biến
- Tính toán đường đi và di chuyển

## Bản đồ

Tương tự như hệ thống điều hướng trên các phương tiện, hay ứng dụng google map chúng ta sử dụng hằng ngày đều có bản đồ rất chính xác và được cập nhật thường xuyên. Còn với robot, chúng cũng cần có bản đồ. Vì vậy chúng ta sẽ phải tạo một bản đồ và cung cấp cho nó, hoặc robot có thể tự tạo bản đồ.

SLAM (Simultaneous Localization And Mapping) được phát triển để robot có thể tự nó khám phá và tạo bản đồ tại một môi trường chưa biết.

## Trạng thái của robot

Robot cần phải đo và ước tính được trạng thái của nó. Hiện nay, phương pháp được sử dụng rộng rãi nhất để ước tính trạng thái trong nhà cho robot dịch vụ là **dead reckoning**<sup>3</sup> - dùng để ước tính trạng thái robot, nhưng nó đã được sử dụng từ rất lâu và phù hợp với các cảm biến giá rẻ và có thể đạt được một độ chính xác nhất định. Sự dịch chuyển của robot được đo bằng số vòng quay của bánh xe. Tuy nhiên, có sai số giữa khoảng cách tính được với số vòng quay của bánh xe và khoảng cách dịch chuyển thực tế. Tuy nhiên, thông tin quán tính từ IMU có thể được sử dụng để giảm thiểu sai số bằng việc bù sai số của vị trí và hướng giữa giá trị tính được và giá trị thực tế.

Hình 2.10a thể thông tin cần thiết cho dead reckoning bao gồm tọa độ tâm robot ( $x, y$ ), khoảng cách giữa hai bánh xe  $D$  và bán kính bánh  $r$ . Giả sử robot di

<sup>3</sup>[https://en.wikipedia.org/wiki/Dead\\_reckoning](https://en.wikipedia.org/wiki/Dead_reckoning)

chuyển một đoạn ngắn trong khoảng thời gian  $T_e$ , vận tốc góc  $(v_l, v_r)$  của bánh trái và bánh phải được tính như trong phương trình 2.1 và 2.2 với góc quay của động cơ bên trái và bên phải (giá trị encoder hiện tại là  $E_{lc}, E_{rc}$  và giá trị encoder trước đó là  $E_{lp}, E_{rp}$ )

$$v_l = \frac{(E_{lc} - E_{lp})}{T_e} \cdot \frac{\pi}{180} (\text{radian/sec}) \quad (2.1)$$

$$v_r = \frac{(E_{rc} - E_{rp})}{T_e} \cdot \frac{\pi}{180} (\text{radian/sec}) \quad (2.2)$$

Phương trình 2.3 và 2.4 tính vận tốc của bánh trái và bánh phải ( $V_l, V_r$ ). Từ vận tốc bánh trái và bánh phải, vận tốc dài ( $v_k$ ) và vận tốc góc ( $\omega_k$ ) của robot có thể tính được như phương trình 2.5 và 2.6

$$V_l = v_l \cdot r (\text{m/s}) \quad (2.3)$$

$$V_r = v_r \cdot r (\text{m/s}) \quad (2.4)$$

$$v_k = \frac{(V_r + V_l)}{2} (\text{m/s}) \quad (2.5)$$

$$\omega_k = \frac{(V_r - V_l)}{D} (\text{rad/s}) \quad (2.6)$$

Cuối cùng, sử dụng những giá trị này, chúng ta có thể tính được vị trí  $(x_{(k+1)}, y_{(k+1)})$  và hướng  $\theta_{k+1}$  của robot từ phương trình 2.7 tới phương trình 2.10

$$\Delta s = v_k T_e \quad \Delta\theta = \omega_k T_e \quad (2.7)$$

$$x_{k+1} = x_k + \Delta s \cdot \cos\left(\theta_k + \frac{\Delta\theta}{2}\right) \quad (2.8)$$

$$y_{k+1} = y_k + \Delta s \cdot \sin\left(\theta_k + \frac{\Delta\theta}{2}\right) \quad (2.9)$$

$$\Theta_{k+1} = \theta_k + \Delta\theta \quad (2.10)$$

## Cảm nhận môi trường

Thứ ba, việc phát hiện ra đâu là vật cản như tường, các đồ vật yêu cầu cần có các cảm biến. Có nhiều loại cảm biến như cảm biến khoảng cách và cảm biến hình ảnh. Cảm biến khoảng cách dạng laser như LDS, LRF, LiDAR, cảm biến siêu âm và cảm biến hồng ngoại. Cảm biến hình ảnh bao gồm stereo camera, monocular camera, omnidirectional camera,... và hiện nay có các loại camera như RealSense, Kinect, Xtion được sử dụng rộng rãi như camera chiều sâu được sử dụng để phát hiện các vật cản.

## Tính toán đường đi và di chuyển

Một đặc điểm cuối cùng cho điều hướng robot là tính toán và di chuyển theo đường đi tối ưu từ vị trí hiện tại tới đích xác định trong bản đồ. Được gọi là tìm kiếm đường đi và lên kế hoạch di chuyển. có nhiều thuật toán như thuật toán  $A^*$ , thuật toán *potential field*(Vùng tiềm năng), *particle filter*(lọc từng phần)...

Kế hoạch di chuyển bao gồm cả *global path planning* (kế hoạch đường đi toàn cục) trên toàn bộ bản đồ và *local path planning* (kế hoạch đường đi địa phương) cho một vùng nhỏ hơn xung quanh robot.

### Di chuyển và tránh vật cản

Nếu một lệnh gửi tới robot trên quỹ đạo di chuyển được tạo bằng *motion planning*, robot di chuyển tới đích theo đường đi đã được lên kế hoạch. Do đó việc cảm nhận môi trường, ước tính trạng thái vị trí robot và *motion planning* vẫn tiếp tục thực hiện, tính toán trong quá trình di chuyển., các vật cản và đối tượng di động xuất hiện đột ngột trong môi trường sẽ được tránh bởi giải thuật *Dynamic Window Approach* (DWA).

## 2.4 Bài toán SLAM 2D

### 2.4.1 Một số phương pháp định vị

Ước tính trạng thái vị trí robot là nghiên cứu rất quan trọng trong Robotics, và được nghiên cứu rất tích cực cho tới ngày nay. Nếu trạng thái vị trí của robot có thể được ước tính chính xác thì sẽ dễ dàng hơn trong việc xây dựng bản đồ dựa trên vị trí robot. Tuy nhiên, có nhiều vấn đề như không chắc chắn của phép đo cảm biến như đã nhắc đến trong 2.1, và cần phải hoạt động real-time<sup>4</sup> để hoạt động trong môi trường thực tế. Nhiều phương pháp để ước tính trạng thái vị trí robot đã được nghiên cứu để giải quyết vấn đề này. Trong khuôn khổ luận văn này, tác giả đề cập tới hai phương pháp là Kalman filter và Particle filter.

#### Kalman filter

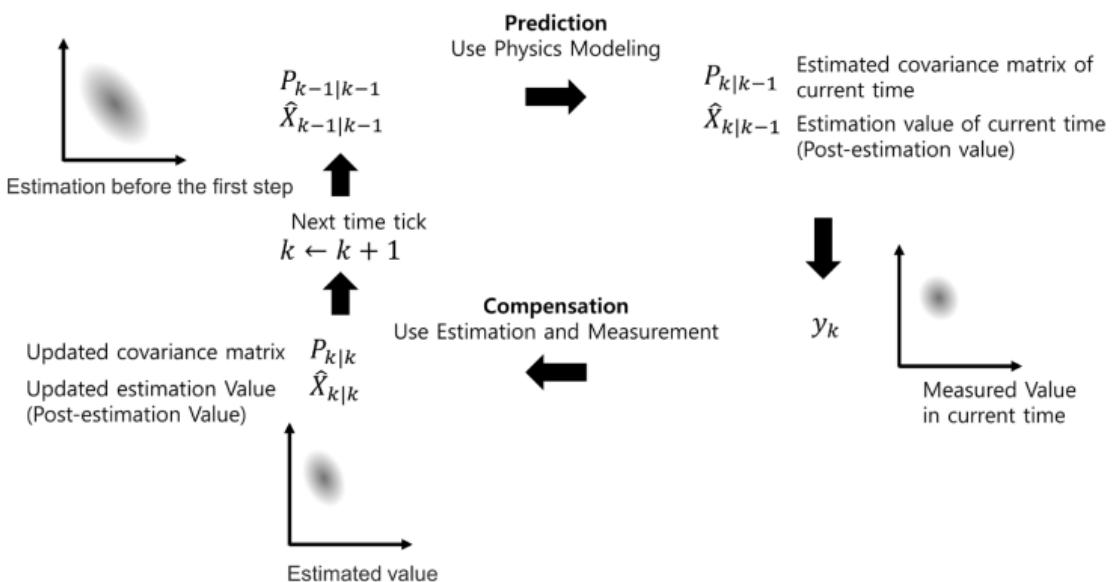
Kalman filter được sử dụng trong dự án Apollo của NASA, được phát triển bởi Tiến sĩ Rudolf E.Kalman. Phương pháp này theo dõi một vật thể trong hệ tuyến tính với nhiều. Bộ lọc dựa trên phương pháp xác suất Bayes bằng cách xây dựng một mô hình và sử dụng mô hình này để dự đoán trạng thái hiện tại từ trạng thái trước đó. Sau đó, sai số giữa giá trị dự đoán của bước ngay trước

---

<sup>4</sup>là hoạt động theo thời gian thực

và giá trị thực tế đo được được sử dụng để cập nhật vào giá trị ước tính để đạt được giá trị chính xác hơn. Bộ lọc lặp lại quá trình trên và tăng độ chính xác. Một quá trình cơ bản được minh họa trong Hình 2.11

Tuy nhiên, Kalman Filter chỉ có thể áp dụng với hệ tuyến tính. Phần lớn robot và cảm biến là hệ thống phi tuyến, và EKF (Extended Kalman Filter) có một số điều chỉnh từ Kalman Filter được sử dụng rộng rãi hơn.



Hình 2.11: Kalman Filter[4]

## Particle Filter

Particle filter là giải thuật phổ biến nhất trong theo dõi vật thể (object tracking). Monte Carlo localization sử dụng bộ lọc Particle. Bộ lọc Kalman filter chỉ chính xác với hệ tuyến tính và nhiễu theo chuẩn Gaussian. Phần lớn các vấn đề trong thế giới thực là các hệ phi tuyến.

Bởi vì các cảm biến và robot cũng là hệ tuyến tính, bộ lọc particle thường được sử dụng cho ước tính trạng thái vị trí robot. Nếu bộ lọc Kalman tìm kiếm các thông số bằng chuyển động tuyến tính thì bộ lọc Particle filter là một kỹ thuật dự đoán thông qua mô phỏng dựa trên phương pháp thử sai (try-and-error). Bộ lọc này ước tính giá trị bằng phân bố xác suất từng phần trong hệ thống.

Khi sử dụng SLAM, giá trị odometry của robot và các giá trị đo từ cảm biến được sử dụng để ước tính trạng thái hiện tại của robot. Trong phương pháp này, trạng thái vị trí không chắc chắn của robot được mô tả bằng một chùm điểm gọi là các mẫu. Chúng ta di chuyển các điểm tới một vị trí và hướng mới được

ước tính dựa trên mô hình chuyển động của robot và xác suất, phép đo trọng số của từng điểm dựa vào giá trị đo thực tế, và giảm nhiễu để ước tính trạng thái chính xác. Trong robot di động, mỗi điểm được biểu diễn bởi trạng thái vị trí  $(x, y, i)$ , trọng số.

Bộ lọc particle filter gồm 5 bước chính, ngoại trừ bước khởi tạo, các bước còn lại được lặp đi lặp lại để thực hiện ước tính vị trí robot. Nói cách khác, đây là phương pháp ước tính trạng thái vị trí của robot bằng cách cập nhật phân bố của các điểm thông qua xác suất của robot trong hệ tọa độ X, Y dựa trên các giá trị cảm biến đo được.

(1) **Khởi tạo:** Vì trạng thái khởi tạo của robot (vị trí và hướng) chưa biết, các điểm dự đoán được phân bố ngẫu nhiên trong phạm vi có thể, với  $N$  điểm. Mỗi điểm khởi tạo có trọng số  $1/N$ , và tổng các trọng số của các điểm dự đoán là 1.  $N$  được đặt theo kinh nghiệm, thông thường là vài trăm. Nếu vị trí khởi tạo đã biết, các điểm dự đoán được đặt gần với robot.

(2) **Dự đoán:** Dựa trên mô hình mô tả hệ thống chuyển động của robot, nó dịch chuyển mỗi điểm particle theo giá trị đo từ thông tin odometry và nhiễu.

(3) **Cập nhật:** Dựa trên thông tin đo được từ các cảm biến, xác suất của mỗi điểm dự đoán được tính toán và giá trị trọng số được cập nhật dựa trên xác suất tính được.

(4) **Ước tính trạng thái:** Vị trí, hướng và trọng số của tất cả các điểm dự đoán được dùng để tính toán trọng số trung bình, giá trị trung bình và giá trị trọng số lớn nhất cho việc dự đoán trạng thái của robot.

(5) **Lấy mẫu lại:** Bước này tạo ra các điểm dự đoán mới để loại bỏ các điểm có trọng số thấp và tạo các điểm mới kế thừa từ các điểm được giữ lại, tổng số điểm  $N$  phải được duy trì.

### 2.4.2 SLAM

SLAM - Simultaneous Localization And Mapping. Đây là một trong các vấn đề khó nhất của robotics. Vấn đề SLAM nảy sinh khi robot không có bản đồ của môi trường và cũng không biết trạng thái vị trí của chính nó. Thay vào đó là dữ liệu đo  $z_{1:t}$  và tín hiệu điều khiển  $u_{1:t}$ . Thuật ngữ SLAM mô tả kết quả của vấn đề: Trong SLAM, robot đạt được một bản đồ môi trường trong khi đồng thời định vị nó ở đâu trong bản đồ đó. Bài toán SLAM khó hơn bài toán định vị bởi vì bản đồ chưa biết trước và nó phải được ước tính vị trí theo một quãng đường dài. Nó cũng khó hơn bài toán tạo bản đồ vì trạng thái vị trí robot chưa biết và nó cũng phải được ước tính trong quãng đường dài.

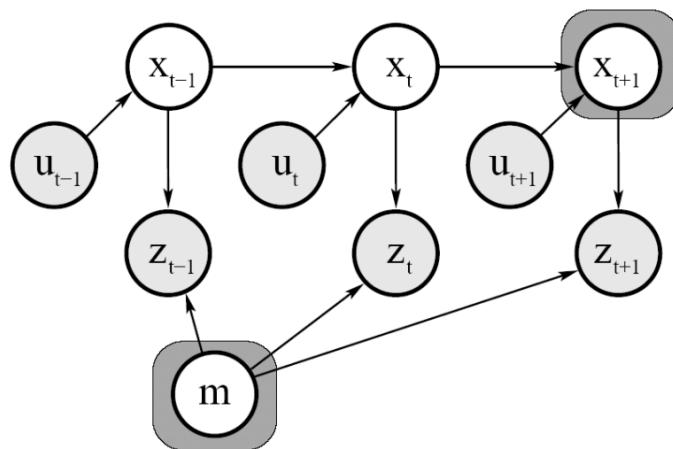
Như một bài toán con gà và quả trứng, bởi vì trong robot di động muốn định vị tốt thì cần phải có bản đồ chính xác và muốn tạo được bản đồ chính xác thì

lại phải định vị tốt.

Theo luật xác suất, có hai dạng chính của SLAM, cả hai dạng này đều quan trọng như nhau. Một dạng là SLAM online. Nó thực hiện các dự đoán qua các trạng thái hiện tại xuyên suốt bản đồ:

$$p(x_t, m | z_{1:t}, u_{1:t}) \quad (2.11)$$

Trong đó,  $x_t$  là trạng thái vị trí tại thời điểm  $t$ ,  $m$  là bản đồ, và  $z_{1:t}$  và  $u_{1:t}$  là các giá trị đo và điều khiển tương ứng. Vấn đề này gọi là SLAM online vì nó chỉ dự đoán các biến tại thời điểm  $t$ , bỏ qua các dữ liệu đo và điều khiển cũ một khi các dữ liệu đó đã được xử lý. Mô hình online SLAM được thể hiện như trong Hình 2.12



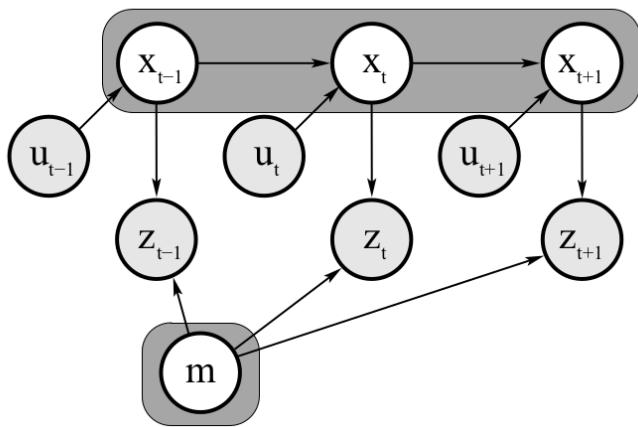
Hình 2.12: Online SLAM

Vấn đề SLAM thứ hai là được gọi là *full SLAM*. Trong full SLAM, chúng ta tính toán toàn bộ đường đi  $x_{1:t}$  xuyên suốt bản đồ, thay vì chỉ trạng thái hiện tại  $x_t$

$$p(x_{1:t}, m | z_{1:t}, u_{1:t}) \quad (2.12)$$

Sự khác nhau giữa online và full SLAM tạo nên sự phân nhánh trong các thuật toán tương ứng với mỗi loại. Cụ thể, online SLAM là kết quả của việc tích hợp trạng thái robot cũ từ full SLAM.

Một đặc điểm quan trọng nữa của SLAM đó là nó có hai thành phần liên tục và rời rạc. Vấn đề liên tục là ước tính vị trí của các vật thể, đối tượng trong bản đồ và các biến số đặt ra của robot. Các đối tượng có thể là các điểm mốc. Bản chất rời rạc liên quan đến sự tương ứng của các vật thể nó phát hiện được, nó phải xem xét vật thể đó đã được phát hiện ở bước trước đó hay chưa.



Hình 2.13: Full SLAM

## 2.5 Lý thuyết điều hướng robot

### 2.5.1 Costmap

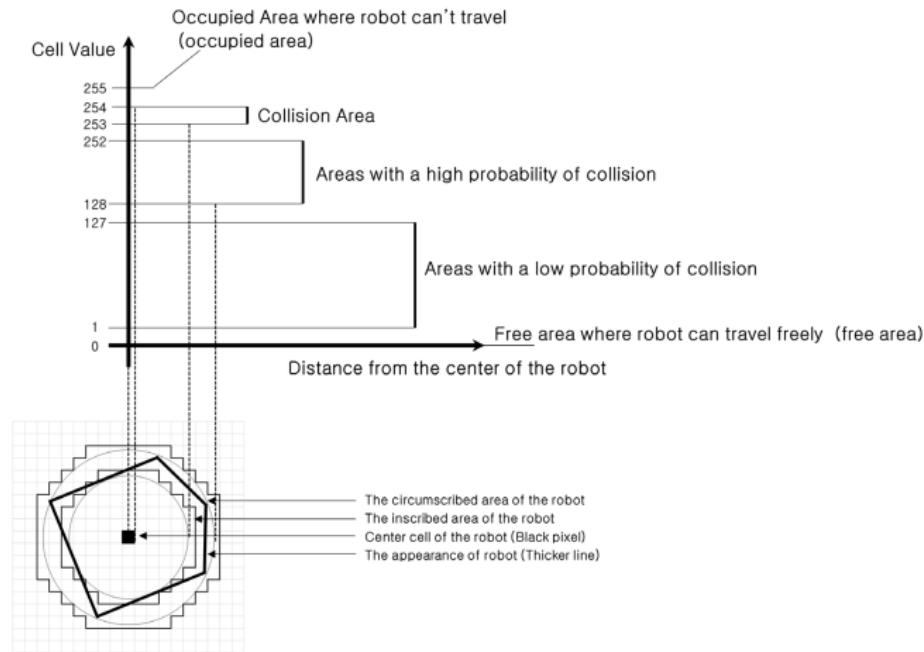
Trạng thái của robot được ước tính dựa trên odometry từ encoder và cảm biến gia tốc góc IMU. Và khoảng cách giữa robot và các vật cản được tính từ cảm biến khoảng cách gắn trên robot. Trạng thái robot và cảm biến, thông tin vật cản, một bản đồ lối chiêm dụng là kết quả của SLAM được sử dụng để tải bản đồ tĩnh và sử dụng các vùng bị chiêm dụng, các vùng trống và các vùng chưa biết cho điều hướng robot.

Trong điều hướng robot, costmap<sup>5</sup> tính vùng có vật cản, các vùng có khả năng va chạm và vùng robot có thể di chuyển. Phụ thuộc vào kiểu điều hướng, costmap có thể chia thành hai phần. Một là *global\_costmap*, thiết lập kế hoạch di chuyển cho điều hướng robot trong toàn bộ không gian của bản đồ cố định. Phần còn lại là *local\_costmap* được sử dụng cho kế hoạch di chuyển và tránh vật cản trong không gian giới hạn quanh robot. Mặc dù mục đích khác nhau nhưng cả hai loại bản đồ này được thể hiện theo một cách.

Costmap thể hiện giá trị từ 0 tới 255. Nghĩa của từng giá trị được thể hiện trong Hình 2.14, và tóm tắt như sau, được dùng để xác định nơi robot có thể di chuyển hoặc va chạm với vật cản.

- **000:** vùng trống robot có thể di chuyển thoải mái
- **001 - 127:** vùng xác suất thấp có vật cản
- **128 - 252:** vùng xác suất cao có vật cản
- **253 - 254:** vùng có vật cản

<sup>5</sup>Bản đồ giá trị



Hình 2.14: Quan hệ giữa khoảng cách tới vật cản và giá trị costmap

- 255: vùng bị chiếm dụng, robot không thể di chuyển.

### 2.5.2 AMCL

AMCL (Adaptive Monte Carlo Localization) có thể xem như phiên bản nâng cấp của phương pháp ước tính trạng thái vị trí Monte Carlo, cải thiện hiệu suất real-time bằng cách giảm thời gian tính toán với số mẫu nhỏ hơn giải thuật Monte Carlo. Mục đích của giải thuật ước tính trạng thái vị trí Monte Carlo (MCL) là xác định vị trí robot trong môi trường đã biết, đó là  $x$ ,  $y$  và  $\theta$  của robot trong bản đồ. MCL tính xác suất nơi robot có thể được đặt. Đầu tiên, vị trí và hướng ( $x, y, \theta$ ) của robot tại thời điểm  $t$  được biểu thị là  $x_t$ , và thông tin khoảng cách từ cảm biến tới thời điểm  $t$  được biểu thị là  $z_{0..t} = \{z_0, z_1, \dots, z_t\}$ , và thông tin di chuyển từ encoder tới thời điểm  $t$  là  $u_{0..t} = \{u_0, u_1, \dots, u_t\}$ . Chúng ta có thể tính độ tin cậy với phương trình:

$$bel(x_t = p(x_t | z_{0..t}, u_{0..t})) \quad (2.13)$$

Vì robot có thể có sai số phần cứng nên phải thiết lập mô hình cảm biến và mô hình di chuyển. Áp dụng quá trình dự đoán và cập nhật của bộ lọc Bayes.

Trong bước dự đoán, vị trí  $bel'(x_t)$  của robot tại thời gian tiếp theo được tính bằng cách sử dụng mô hình di chuyển  $p(x_t | z_{0..t}, u_{0..t})$  của robot và xác suất  $bel(x_t)$  tại vị trí trước đó, và thông tin chuyển động  $u$  nhận được trước đó.

$$bel'(x_t = \int p(x_t|z_{0..t}, u_{0..t}))bel(x_{t-1})dx_{t-1} \quad (2.14)$$

Tiếp theo là bước cập nhật. Thời điểm này, mô hình cảm biến  $p(z_t|x_t)$ , xác suất  $bel(x_t)$  và hằng số chuẩn ( $\eta_t$ ) được dùng để nâng độ chính xác xác suất  $bel'(x_t)$  dựa trên thông tin cảm biến.

$$bel(x_t) = \eta_t p(z_t|x_{0..t}) bel'(x_t) \quad (2.15)$$

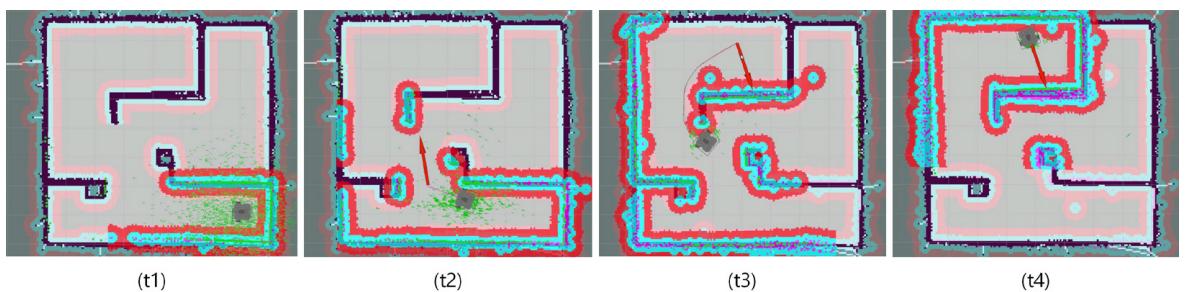
Phương trình sau để ước tính vị trí bằng việc tạo ra N điểm mẫu với particle filter sử dụng các xác suất đã tính  $bel(x_t)$  của vị trí hiện tại. Đầu tiên là quá trình lấy mẫu. Ở đây, có một tập mẫu  $x'_t$  được trích xuất bằng cách sử dụng mô hình di chuyển của robot  $p(x_t|x_{t-1}, u_{t-1})$  tại xác suất  $bel(x_{t-1})$  của vị trí trước. Mẫu thứ  $i$   $x_i^{(i)}$  giữa tập mẫu  $x'_t$ , thông tin khoảng cách  $z_t$  và hằng số chuẩn hóa  $\eta$  được sử dụng để tính trọng số  $\omega_t^{(i)}$ .

$$\omega_t^{(i)} = \eta p(z_t|x_t'^{(i)}) \quad (2.16)$$

Cuối cùng, trong quá trình lấy mẫu lại, chúng ta tạo N mẫu mới  $X_t$  sử dụng mẫu  $x_t'^{(i)}$  và trọng số  $\omega_t^{(i)}$ .

$$X_t = \left\{ x_t^{(j)} | j = 1..N \right\} \quad \left\{ x_t'^{(i)}, \omega_t^i \right\} \quad (2.17)$$

Bằng việc lặp lại quá trình trên trong khi di chuyển các điểm mẫu, vị trí được ước tính của robot có độ chính xác tăng lên. Ví dụ như trong hình Hình 2.15, quá trình hội tụ vị trí từ 't1' tới 't4'. Tất cả quá trình này đã được mô tả trong phần 2.1.2. [4]



Hình 2.15: Quá trình AMCL cho ước tính trạng thái vị trí robot

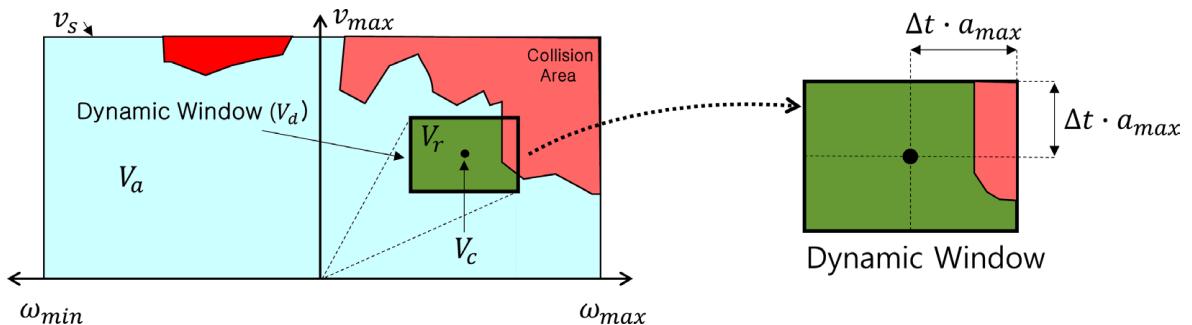
### 2.5.3 Dynamic Window Approach (DWA)

DWA là phương pháp phổ biến cho di chuyển và tránh vật cản. Đây là phương pháp lựa chọn tốc độ để có thể tới điểm đích nhanh nhất trong khi tránh vật

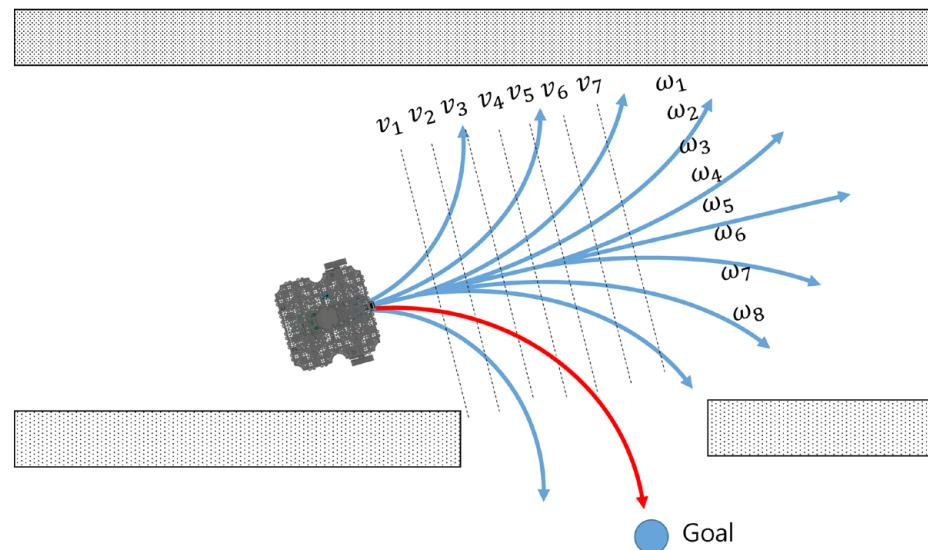
cản. Trong ROS, Trajectory planner được dùng để tạo kế hoạch di chuyển cục bộ, nhưng DWA được dùng để thay thế vì sự vượt trội của nó.

Đầu tiên, robot không nằm trong hệ tọa độ  $x, y$  mà trong không gian tìm kiếm với tốc độ di chuyển  $x$  và vận tốc góc  $\omega$ , như trong hình Hình2.16. Trong không gian này, robot có tốc độ tối đa cho phép do giới hạn phần cứng và được gọi là Cửa sổ động.

Trong cửa sổ động này, hàm mục tiêu  $G(v, \omega)$  được dùng để tính vận tốc dài  $x$  và vận tốc góc  $\omega$  tối đa từ hàm mục tiêu trên và xem xét hướng, vận tốc và vật cản của robot. Nếu vẽ đồ thị ra chúng ta có thể tìm được vận tốc tối ưu giữa nhiều giá trị  $v$  và  $\omega$  để tới đích như trong hình Hình2.17



Hình 2.16: Không gian tìm kiếm vận tốc và cửa sổ động



Hình 2.17: Vận tốc dài  $v$  và vận tốc góc  $\omega$

# Chương 3

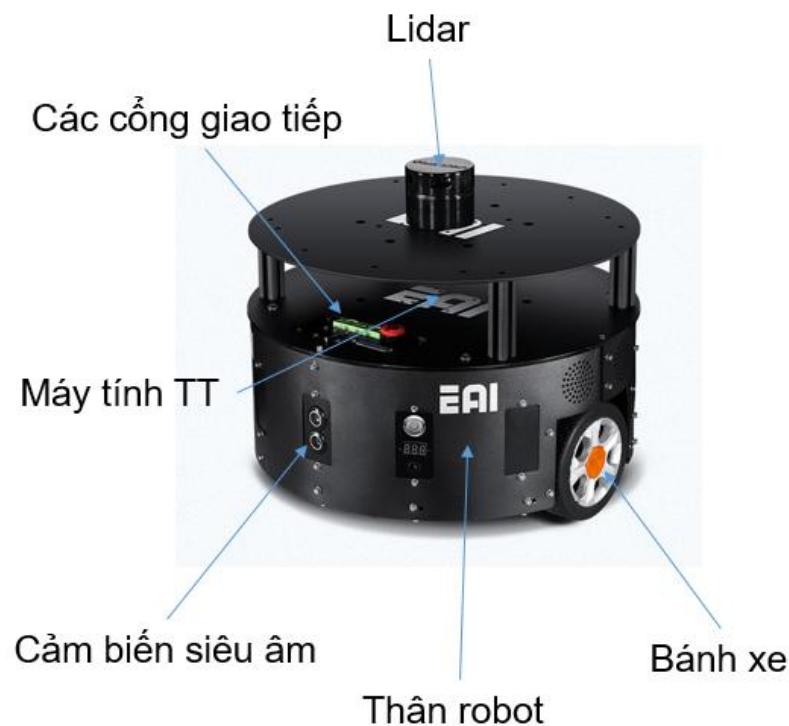
## Thực nghiệm

### 3.1 Đặt vấn đề

Phát triển robot tự hành ứng dụng SLAM trên nền tảng hệ điều hành ROS.  
Cải tiến tránh vật cản bằng

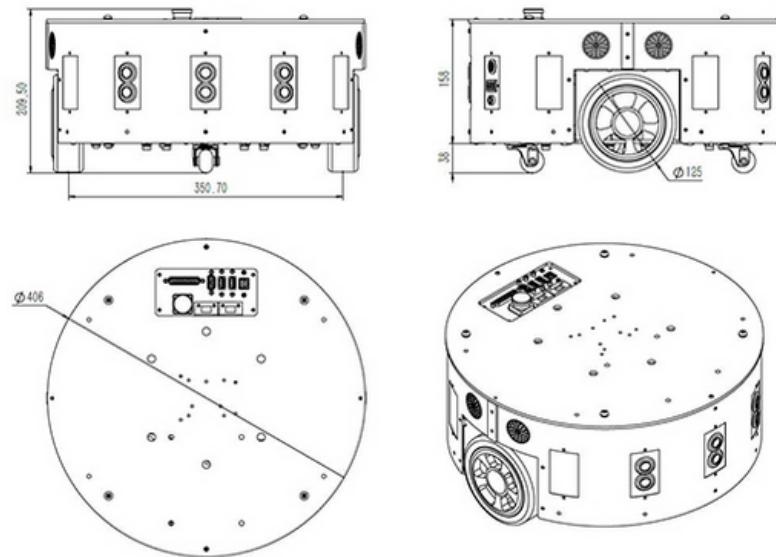
### 3.2 Giới thiệu platform robot của nhóm

Luận văn này được thực hiện dựa trên nền tảng robot di động EAI Dashgo D1 Hình3.1. Là một nền tảng robot di động thông minh được sản xuất phục vụ nghiên cứu về robot tự hành.



Hình 3.1: Nền tảng robot di động Dashgo D1

Robot có cấu tạo gồm 3 phần chính: Phần chân để thực hiện chức năng di chuyển, phần cảm biến và phần điều khiển trung tâm.



Hình 3.2: Cấu tạo phần chân để

### 3.2.1 Phần chân để

Phần chân để có cấu tạo gồm hai bánh chủ động và 2 bánh xe bị động (Hình 3.2). Động cơ có gắn encoder độ chính xác cao để phản hồi vị trí. Phần chân để được điều khiển bởi mạch điều khiển được mở rộng từ mạch arduino kết hợp với các module điều khiển bánh xe. Hoạt động nhờ nguồn điện 12V từ acquy với thời gian sạc đầy khoảng 4 tiếng.

### 3.2.2 Phần cảm biến

Robot được trang bị bốn cảm biến siêu âm và một cảm biến LIDAR. Bốn cảm biến siêu âm sử dụng cho mục đích tránh vật cản. Nguyên lý hoạt động của cảm biến siêu âm như sau:

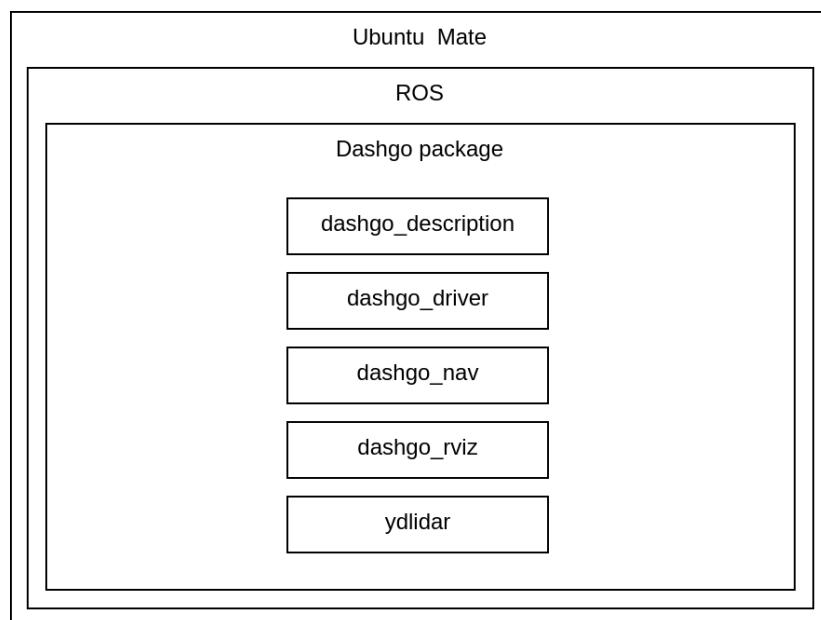
- Đầu phát của cảm biến phát ra sóng siêu âm
- Sóng siêu âm bị dội lại khi có vật cản
- Thực hiện đo khoảng cách từ lúc phát tới lúc nhận được sóng dội lại, tính toán dựa trên tốc độ di chuyển sóng âm trong không khí ta tính được khoảng cách từ cảm biến tới vật.

Cảm biến siêu âm có các ưu điểm như không bị ảnh hưởng bởi màu sắc của vật, hoạt động tốt trong môi trường tối, tiêu tốn ít năng lượng và dễ dàng kết nối với

các vi điều khiển. Tuy nhiên nó cũng có các điểm hạn chế như giới hạn khoảng cách đo được, độ phân giải và tần số đo thấp do đó nó không phù hợp với các ứng dụng có độ vật đích di chuyển nhanh. Cảm biến siêu âm không hoạt động được với các bề mặt không bằng phẳng, các bề mặt hấp thụ âm thanh.

Cảm biến LIDAR YDLIDAR G4 sử dụng tia laser sử dụng đạt tiêu chuẩn an toàn FDA Class 1<sup>1</sup>. Cảm biến tích hợp một động cơ để quay quét mắt laser  $360^{\circ}$  với tần số quét từ 5-12Hz. Do khoảng cách bằng laser với giải đo từ 0.10 - 16m với tần số lấy mẫu có thể đạt 9000Hz. Cảm biến này được sử dụng cho việc tạo bản đồ, điều hướng và tránh vật cản.

### 3.2.3 Hệ thống phần mềm



Hình 3.3: Kiến trúc phần mềm điều khiển robot trên Dashgo D1

Robot sử dụng mạch Raspberry Pi 3 làm bộ điều khiển trung tâm. Chạy hệ điều hành robot ROS Kinetic trên nền tảng Ubuntu MATE 16.04. Để thực hiện các chức năng cơ bản như Định vị, Tạo bản đồ, Di chuyển tới vị trí xác định trong bản đồ, Tránh vật cản trong quá trình di chuyển phải cài đặt gói chương trình dashgo được cung cấp bởi nhà sản xuất.

Hình 3.3 mô tả kiến trúc phần mềm trên Dashgo D1. Trong đó:

- `dashgo_description` là gói chứa các mô tả của robot, được dùng trong hiển thị Rviz và mô phỏng robot trong Gazebo.

---

<sup>1</sup>Tiêu chuẩn Laser FDA Class 1: Được xem là không gây nguy hiểm. Mức độ nguy hiểm tăng khi nhìn qua kính hội tụ quang học như kính lúp, kính hiển vi. Theo [www.fda.gov](http://www.fda.gov)

- `dashgo_driver` chứa các thông số cấu hình và chương trình driver để điều khiển robot.
- `dashgo_nav` chứa các chương trình điều khiển robot, bao gồm `gmapping` để tạo bản đồ môi trường và `navigation` để thực hiện di chuyển tự động trong bản đồ.
- `dashgo_rviz` chứa các file mẫu `rviz` của một số tác vụ cơ bản.
- `dashgo_tools` chứa các công cụ để làm việc với robot.
- `ydlidar` là gói quản lý, driver của YDLIDAR.

### 3.3 Điều khiển Dashgo robot

«1. Các bước thực hiện tạo bản đồ, lưu bản đồ bằng `gmapping` -> Navigation: ước tính vị trí, di chuyển tới các điểm xác định trong bản đồ  
2. Dánh giá hoạt động của robot»

#### 3.3.1 Quy trình thực hiện

**Tạo bản đồ:** Với một môi trường chưa biết trước, việc đầu tiên là phải tạo bản đồ. Chương trình tạo bản đồ được tạo sẵn trong file `gmapping.launch`

```
$ roslaunch dashgo_nav gmapping.launch
```

Chương chương trình này sẽ gọi tới một số chức năng khác như driver điều khiển chân để `driver.launch`, driver lidar `lidar.launch`, chạy thuật toán và khai báo các thông số tại `gmapping_base.launch`

Sau khi chạy file `gmapping.launch`, bật Rviz lên để xem vị trí của robot và bản đồ.

```
$ roslaunch dashgo_rviz view_navigation.launch
```

Sử dụng công cụ di chuyển robot bằng tay để thu thập thông tin tạo bản đồ của môi trường. Chúng ta có thể sử dụng bàn phím hoặc app để di chuyển robot. Cho robot di chuyển trong toàn bộ môi trường để thực hiện thu thập thông tin cho bản đồ. Theo dõi trực quan trên Rviz để quan sát bản đồ tạo được.

```
$ rosrun dashgo_tool teleop_keyboard.py
```

Sau đó lưu bản đồ lại bằng lệnh:

```
$ cd dashgo_nav/maps  
$ rosrun map_server map_saver -f <địa chỉ lưu đồ>
```

**Điều hướng:** Chương trình này sử dụng bản đồ đã được lưu, sử dụng thuật toán amcl để định vị và điều hướng trong bản đồ đã biết.

```
$ roslaunch dashgo_nav navigation.launch
```

Bật Rviz:

```
$ roslaunch dashgo_nav navigation.launch
```

Trên giao diện Rviz, chúng ta sử dụng công cụ 2D Pose Estimate để định vị lại vị trí của robot trong bản đồ bằng tay. Bởi vì khi mới khởi tạo, robot chưa có đủ thông tin để tự định vị nó đang ở đâu trong bản đồ. Bước định vị bằng tay này không cần quá chính xác bởi vì sau khi robot di chuyển nó sẽ tự mình định vị và điều chỉnh vị trí của robot trong bản đồ. Sau đó, để điều hướng robot đến một điểm bất kỳ, chúng ta sử dụng công cụ 2D Nav Goal. Ngoài ra, trên giao diện Rviz chúng ta còn có thể thấy các thông tin về: vị trí của robot, bản đồ, costmap...

### 3.3.2 Đánh giá hoạt động của robot

Với nền tảng robot hiện tại, hệ thống robot hiện tại có một số hạn chế như:

- Chỉ phát hiện và tránh vật cản được ở hai tầng cảm biến hiện tại (tương ứng với mặt phẳng đi qua cảm biến siêu âm và mặt phẳng quét của cảm biến LIDAR)
- Cảm biến siêu âm chỉ dùng để phát hiện và tránh vật cản, nhưng thường xuyên có sai số đo
- Cảm biến laser sử dụng cho nhiệm vụ định vị, điều hướng và cả tránh vật cản, vì vậy chi phí tính toán cao và thường phản ứng chậm với vật cản động.

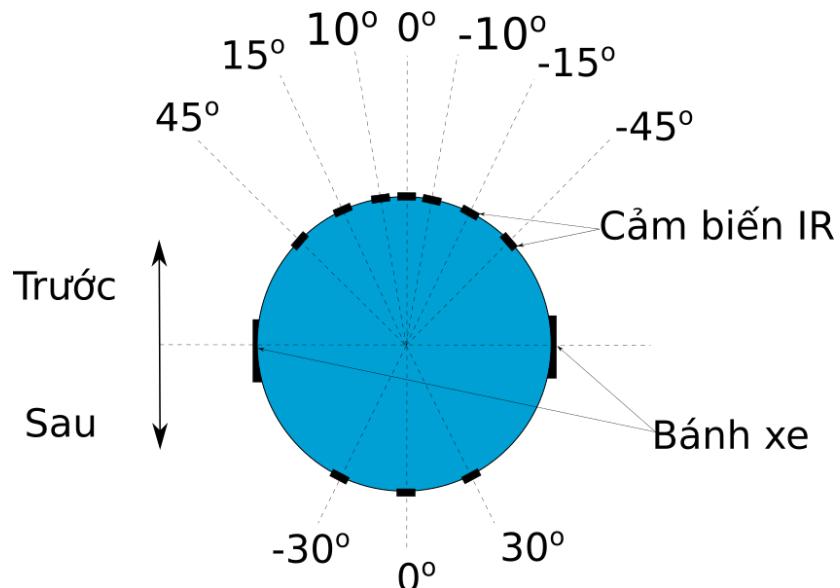
Với mục tiêu phát triển thành robot dịch vụ, hoạt động hiệu quả trong môi trường biến động, tác giả đề xuất phương án bổ sung thêm một tầng cảm biến để phát hiện và tránh vật cản. Hệ thống này sử dụng một số cảm biến hồng ngoại, chi phí thiết bị và chi tính toán thấp để dễ dàng tích hợp được vào robot,

phối hợp cùng với hai tầng cảm biến có sẵn nhưng vẫn đảm bảo robot hoạt động ổn định theo thời gian thực.

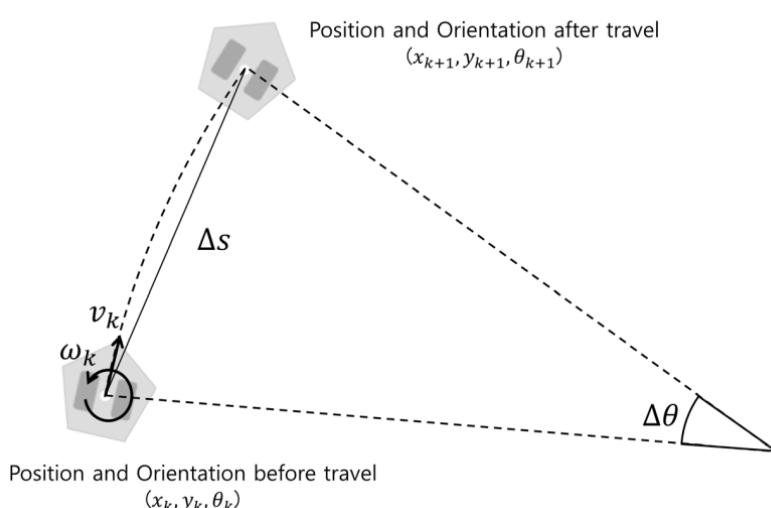
## 3.4 Phương pháp

### 3.4.1 Phần cứng

Phần cứng gồm một hệ thống cảm biến hồng ngoại IR Sharp GP2Y0A21YK0F và mạch Arduino Mega 2560 (Hình 3.7).



Hình 3.4: Sơ đồ bố trí cảm biến

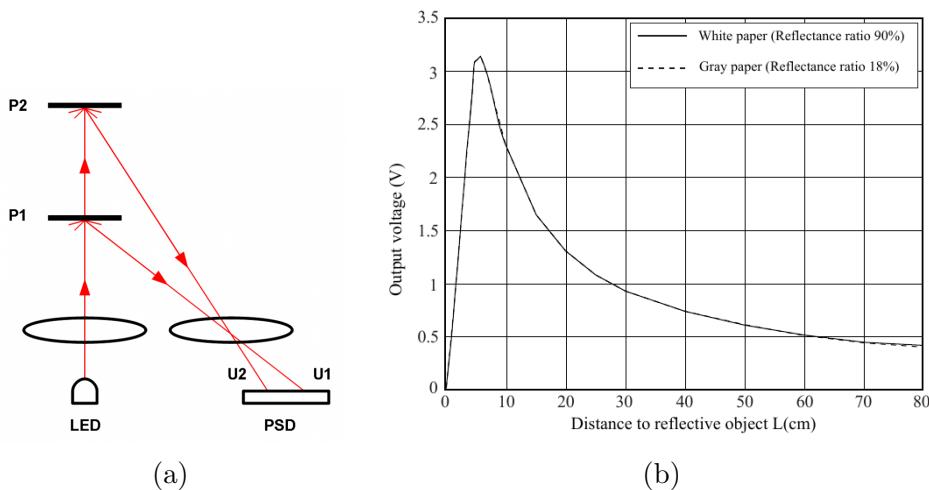


Hình 3.5: Dead reckoning [4]

Mười cảm biến khoảng cách hồng ngoại được bố trí như Hình 3.4. Bao gồm 7 cảm biến ở phía trước robot và 3 cảm biến ở phía sau robot. Trong quá trình

di chuyển, robot có cả tiến và lùi vì vậy cần phải đặt cảm biến ở cả trước và phía sau robot. Robot sử dụng hai bánh dẫn động chính và di chuyển trên mặt phẳng, do vậy chuyển động của robot được quy về hai thông số vận tốc dài  $v$  và vận tốc góc  $\omega$  theo phương pháp ước tính trạng thái robot dead-reckoning [4]. Do đó tác giả đề xuất phương án mới đặt cảm biến dày hơn ở khu vực chính giữa và thưa ra hai bên như Hình3.4.

### 3.4.2 Xử lý dữ liệu cảm biến



Hình 3.6: Cảm biến khoảng cách hồng ngoại

Cảm biến khoảng cách hồng ngoại là loại cảm biến sử dụng ánh sáng hồng ngoại với một đầu phát và một đầu thu. Cơ chế hoạt động của cảm biến khoảng cách hồng ngoại như Hình3.6a. Trong đó, đèn LED phát tia sáng hồng ngoại, khi gặp vật cản, tia sáng sẽ bị phản xạ lại, in lên tấm PSD tại các vị trí tương ứng với góc chiếu khác nhau, tạo điện áp khác nhau từ U1 đến U2 <sup>2</sup>.

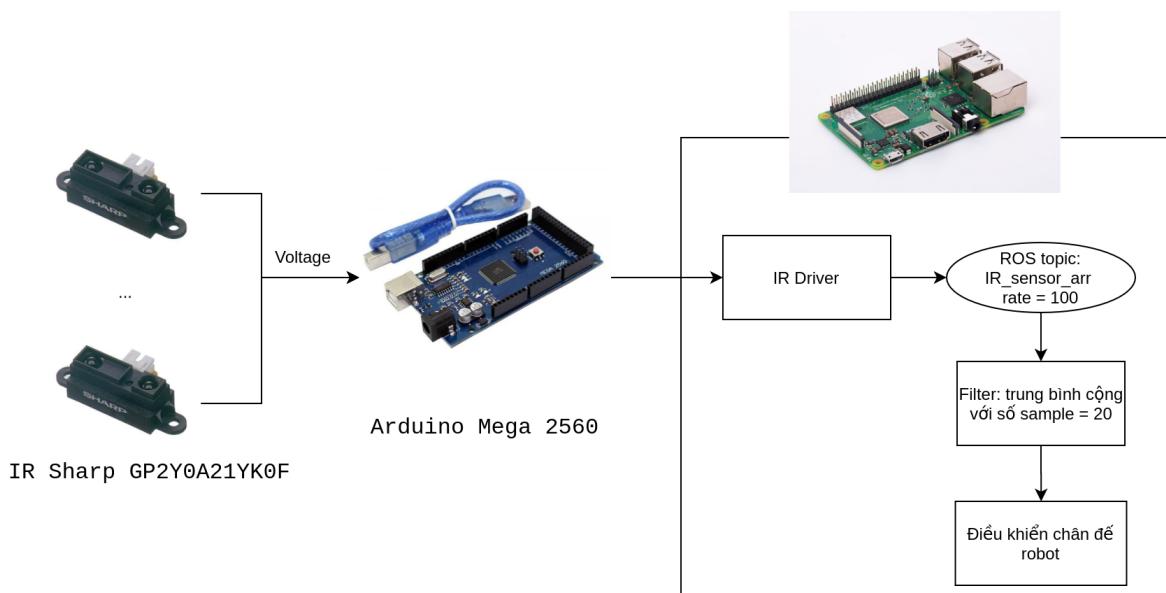
Mối quan hệ giữa điện áp ra và khoảng cách của module cảm biến khoảng cách hồng ngoại IR Sharp GP2Y0A21YK0F như Hình3.6b <sup>3</sup>

Mạch Arduino Mega sẽ đọc tín hiệu điện trả về từ các cảm biến IR mỗi khi có lệnh yêu cầu từ Raspberry Pi. Mỗi chân dữ liệu của cảm biến kết nối với chân Analog input trên mạch Arduino để đọc dữ liệu điện áp thông qua ADC 8 bit.

Trên Raspberry Pi, node `ir_driver` sẽ nhận các dữ liệu ADC của các cảm biến từ Arduino. Hình3.6b cho ta mối liên hệ giữa điện áp và khoảng cách của cảm biến SHARP GP2Y0A21YK0F. Tuy nhiên, đây là đồ thị một đường cong không được biểu diễn bởi một hàm toán học cụ thể. Phương pháp được sử dụng là dùng thí nghiệm đo với giá trị điện áp và khoảng cách tương ứng. Sau đó

<sup>2</sup>[https://home.roboticlab.eu/en/examples/sensor/ir\\_distance](https://home.roboticlab.eu/en/examples/sensor/ir_distance)

<sup>3</sup>[www.sparkfun.com/datasheets/Components/GP2Y0A21YK.pdf](http://www.sparkfun.com/datasheets/Components/GP2Y0A21YK.pdf)



Hình 3.7: Work-flow xử lý dữ liệu cảm biến

nội suy kết quả thí nghiệm thành một hàm toán học gần đúng. Trong luận văn này, tác giả tham khảo tại <https://github.com/guillaume-rico/SharpIR> và kiểm nghiệm thấy kết quả khá chính xác:

$$vol = \frac{ADC * 5.0}{1023.0} \quad (3.1)$$

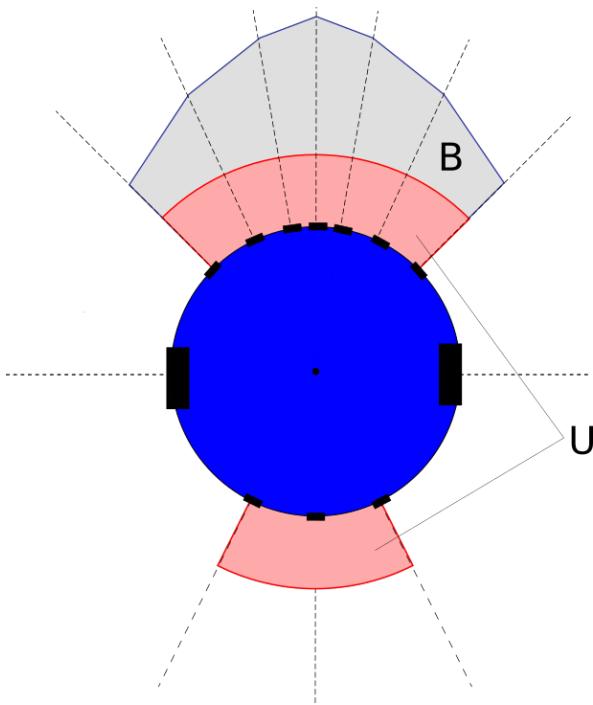
$$distance = 27.728 * vol^{-1.2045} \quad (3.2)$$

Sau khi tính được khoảng cách của các cảm biến sẽ publish ra topic `IR_sensor_arr` chứa thông tin đo được từ tất cả các cảm biến với tần số 20Hz. Sau đó dùng bộ lọc trung bình cộng để lấy trung bình cộng của 4 lần đo để loại bỏ một phần nhiễu.

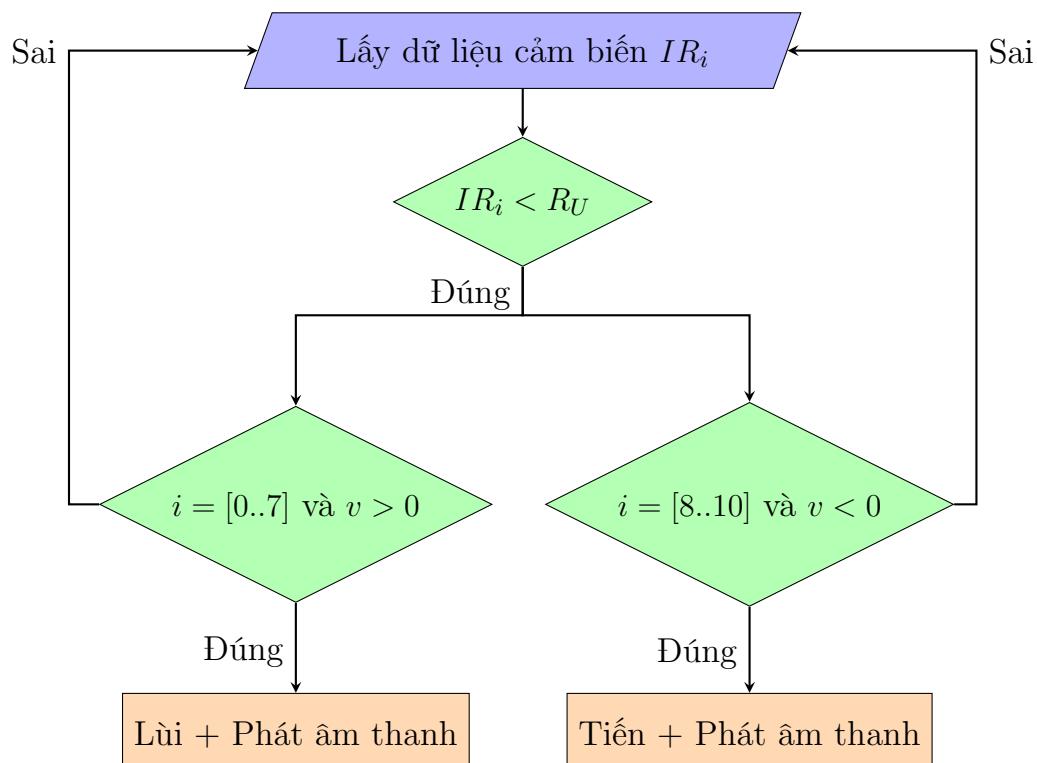
### 3.4.3 Trình bày giải thuật

Tác giả đề xuất sử dụng hai giải thuật để điều khiển robot tránh vật cản sử dụng tầng cảm biến hồng ngoại. Định nghĩa hai vùng phát hiện vật cản. Vùng B và vùng U như Hình 3.8. Chúng ta đặt mức độ ưu tiên khác nhau cho từng vùng. Đặt cờ C1, C2 lần lượt thể hiện thuật toán phản ứng ở vùng B và vùng U.

**Vùng khẩn cấp U** xác định một bằng một đường tròn bán kính  $R_U$  từ tâm của robot. Vùng này thể hiện vùng nguy hiểm, khi phát hiện có vật cản nằm trong vùng này nếu robot có lệnh di chuyển tiến hoặc lùi thì sẽ dừng lại, di chuyển robot lùi/tiến để tránh khỏi vật cản, đồng thời phát âm thanh để thông báo có vật cản cho đến khi không còn vật cản trong vùng này.



Hình 3.8: Vùng xác định vật cản

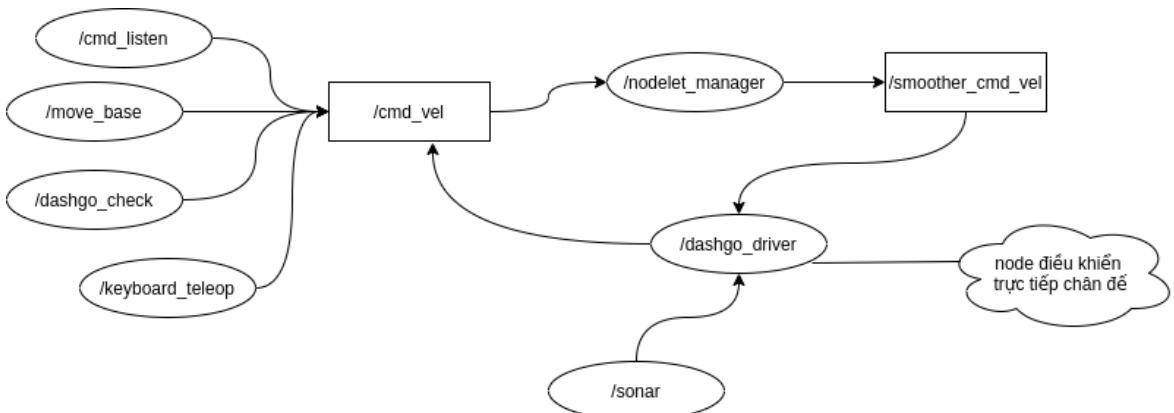


Hình 3.9: Flow-chart với vùng khẩn cấp

**Bong bóng phản ứng B** Trên cơ sở tham khảo [2] Thuật toán bong bóng phản ứng xác định một đường bao phía trước robot, đường bao này được làm mới sau mỗi chu kì  $\Delta t$ , kích thước của đường bao phụ thuộc vào trọng số tương ứng với mỗi vị trí cảm biến và phụ thuộc vào tốc độ di chuyển của robot.

### 3.4.4 Phối hợp phân mức điều khiển

Robot có nhiều chương trình khác nhau có thể tác động đến việc di chuyển của nó. Ví dụ trong phần lớn thời gian robot di chuyển dưới sự điều khiển của chương trình **navigation**, khi cần điều khiển bằng tay, robot lại chạy dưới sự điều khiển của chương trình **teleop**, khi gặp vật cản được phát hiện bởi cảm biến sonar, robot phản ứng lại. Như trường hợp đơn giản vừa rồi có tới ba tiến trình cùng đồng thời điều khiển robot.



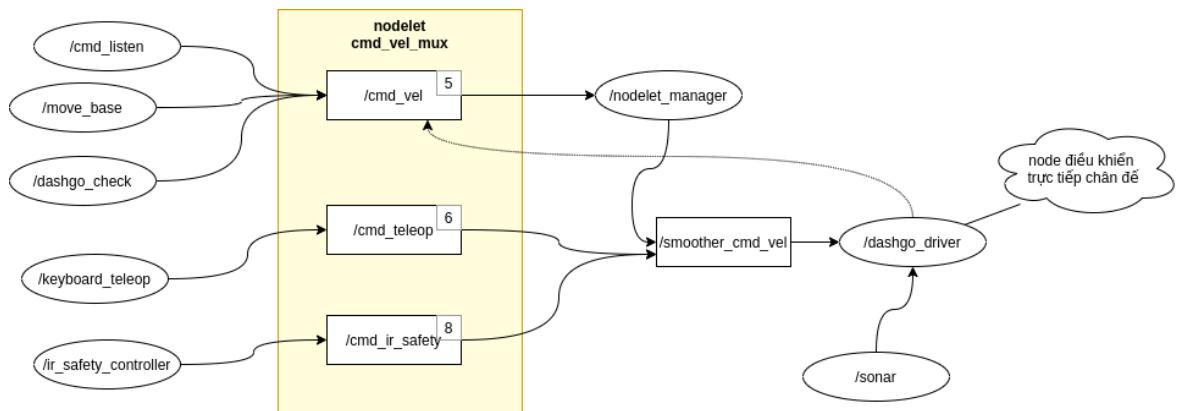
Hình 3.10: Các topic điều khiển robot

Hình 3.10 thể hiện các node và topic điều khiển robot Dashgo D1. Trong đó, topic **/cmd\_vel** nhận thông tin điều khiển di chuyển chân đế từ nhiều topic khác nhau. Sau đó thông qua trình quản lý **nodelet\_manager** để chạy chương trình làm mịn tốc độ, chống giật cho robot. **/smoother\_cmd\_vel** được topic **/dashgo\_driver** nhận và thực hiện các tính toán điều khiển tối vòng quay của động cơ để di chuyển. Ta thấy có **/sonar** được liên kết trực tiếp với **/dashgo\_driver**, ở robot này, dữ liệu từ các cảm biến siêu âm được driver điều khiển chân đế đọc trực tiếp, sau đó xử lý các tình huống và tính toán số vòng quay của động cơ để thực hiện điều khiển chân đế trong trường hợp khẩn cấp.

### 3.4.5 Phương pháp phối hợp phân mức điều khiển

Có một vài phương pháp để phối hợp phân quyền điều khiển cho robot. Khi chúng ta có nhiều thuật toán, nhiều chương trình cùng điều khiển tới một hoạt động nào đó.

Chúng ta sẽ sử dụng cmd\_vel\_mux của nodelet để thực hiện phân quyền điều khiển cho robot để dễ dàng tích hợp tín hiệu điều khiển từ cảm biến an toàn IR.



Hình 3.11: Thiết kế phân quyền điều khiển

### 3.5 Thực nghiệm và đánh giá kết quả

## **Chương 4**

# **Kết luận và tầm nhìn**

**4.1 Kết luận**

**4.2 Tầm nhìn**

# Tài liệu tham khảo

- [1] Y. Koren, J. Borenstein, *et al.*, “Potential field methods and their inherent limitations for mobile robot navigation.,” in *ICRA*, vol. 2, pp. 1398–1404, 1991.
- [2] I. Susnea, V. Mînză, and G. Vasiliu, “Simple, real-time obstacle avoidance algorithm for mobile robots,” in *CI 2009*, 2009.
- [3] S. Thrun, W. Burgard, D. Fox, *et al.*, *Probabilistic robotics, vol. 1*. MIT press Cambridge, 2005.
- [4] Y. Pyo, H. Cho, R. Jung, and T. Lim, *ROS Robot Programming*. 2017.
- [5] M. Wise and F. Robotics, “Understanding the Basics of AMR Technology: What You Need to Know,” *autonomous Mobile Robot Conference*.
- [6] Q. Dongyue, H. Yuanhang, and Z. Yuting, “The investigation of the obstacle avoidance for mobile robot based on the multi sensor information fusion technology,” *Int. J. Mat. Mech. Manuf*, vol. 1, pp. 366–370, 2013.
- [7] M. Gao, J. Tang, Y. Yang, Z. He, and Y. Zeng, “An obstacle detection and avoidance system for mobile robot with a laser radar,” in *2019 IEEE 16th International Conference on Networking, Sensing and Control (ICNSC)*, pp. 63–68, 2019.
- [8] P. Wu, S. Xie, H. Liu, J. Luo, and Q. Li, “A novel algorithm of autonomous obstacle-avoidance for mobile robot based on LIDAR data,” in *2015 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, IEEE, dec 2015.
- [9] Y. Peng, D. Qu, Y. Zhong, S. Xie, J. Luo, and J. Gu, “The obstacle detection and obstacle avoidance algorithm based on 2-D lidar,” in *2015 IEEE International Conference on Information and Automation, ICIA 2015 - In conjunction with 2015 IEEE International Conference on Automation and Logistics*, no. August, pp. 1648–1653, 2015.

- [10] N. Baras, G. Nantzios, D. Ziouzios, and M. Dasygenis, “Autonomous obstacle avoidance vehicle using lidar and an embedded system,” in *2019 8th International Conference on Modern Circuits and Systems Technologies (MOCAST)*, pp. 1–4, 2019.
- [11] J. Borenstein and Y. Koren, “Real-time obstacle avoidance for fast mobile robots,” *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 19, pp. 1179–1187, sep 1989.
- [12] J. Borenstein, Y. Koren, *et al.*, “The vector field histogram - fast obstacle avoidance for mobile robots,” vol. 7, no. 3, pp. 278–288, 1991.
- [13] D. Fox, W. Burgard, and S. Thrun, “The dynamic window approach to collision avoidance,” *IEEE Robotics & Automation Magazine*, vol. 4, pp. 23–33, mar 1997.
- [14] J. Borenstein and Y. Koren, “High-speed obstacle avoidance for mobile robots,” pp. 382 – 384, 09 1988.
- [15] I. Ulrich and J. Borenstein, “Vfh+: reliable obstacle avoidance for fast mobile robots,” in *Proceedings. 1998 IEEE International Conference on Robotics and Automation (Cat. No.98CH36146)*, vol. 2, pp. 1572–1577 vol.2, 1998.
- [16] S. Quinlan and O. Khatib, “Elastic bands: connecting path planning and control,” in *[1993] Proceedings IEEE International Conference on Robotics and Automation*, pp. 802–807 vol.2, 1993.
- [17] I. Susnea, A. Filipescu, G. Vasiliu, G. Coman, and A. Radaschin, “The bubble rebound obstacle avoidance algorithm for mobile robots,” in *IEEE ICRA 2010*, pp. 540–545, 2010.