

**ĐẠI HỌC QUỐC GIA HÀ NỘI
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ**



**Nguyễn Mạnh Cường Nguyễn Hữu Huy
MSV: 22027501 MSV: 22027534**

**THIẾT KẾ ROBOT 4 BÁNH OMNI
SỬ DỤNG GMAPPING VÀ ROS NAVIGATION**

BÁO CÁO BÀI TẬP LỚN

Môn học: Lập trình Robot với ROS

**Giảng viên hướng dẫn: TS. Lê Xuân Lực
KS. Dương Văn Tân**

HÀ NỘI - 2025

TÓM TẮT

Tóm tắt: Đề tài: **Thiết kế Robot 4 bánh Omni và Điều khiển Di chuyển, Quét bản đồ GMapping và Navigation** tập trung vào việc phát triển một hệ thống robot tự hành sử dụng thiết kế xe 4 bánh Omni. Robot này được trang bị các cảm biến, bao gồm LiDAR và camera, IMU, để hỗ trợ các chức năng quét bản đồ và điều hướng trong môi trường công nghiệp hoặc logistics. Đề tài sử dụng ROS1 để triển khai các mô-đun SLAM (GMapping) nhằm xây dựng bản đồ môi trường và thuật toán Navigation để robot có thể di chuyển tự động đến các vị trí yêu cầu. Mục tiêu là tạo ra một hệ thống di chuyển hiệu quả, chính xác, và tự động, đáp ứng nhu cầu của các ứng dụng thực tế trong các khu công nghiệp.

Từ khóa:

LỜI CẢM ƠN

Lời đầu tiên, em xin gửi lời cảm ơn đến TS. Lê Xuân Lực và KS. Dương Văn Tân, người đã cho bọn em những đóng góp, lời khuyên trong quá trình nghiên cứu cũng như hoàn thành đề tài này.

Em xin chân thành cảm ơn!

Nguyễn Mạnh Cường

Mục lục

TÓM TẮT

i

LỜI CẢM ƠN

ii

1 GIỚI THIỆU

1

1.1	Bối cảnh	1
1.2	Bánh Omni	2
1.2.1	Tổng quan về bánh Omni	2
1.2.2	Tại sao chọn bánh Omni để thiết kế?	3
1.3	Yêu cầu thiết kế và mục tiêu của dự án	4
1.3.1	Yêu cầu thiết kế	4
1.3.2	Mục tiêu của dự án	4
1.4	Bố cục tiểu luận	5

2 CƠ SỞ LÝ THUYẾT

6

2.1	Động học của bánh Omni	6
2.1.1	Động học ngược	6
2.1.2	Động học thuận	8
2.2	Tính toán Odometry	10
2.2.1	Dựa vào Gazbo	10
2.2.2	Dựa vào động học Robot	12

3 THIẾT KẾ ROBOT 4 BÁNH OMNI

15

3.1	Thiết kế Solidwork	15
3.1.1	Thân xe và cách bố trí bánh	15
3.1.2	Hình ảnh các bản thiết kế	17
3.2	Đặt trực và xuất file URDF	21
3.2.1	Cách đặt trực và xuất file URDF	21
4	THIẾT LẬP SLAM GMAPPING CHO ROBOT OMNI	26
4.1	Giới Thiệu Thuật Toán Gmapping	26
4.1.1	Nguyên lý hoạt động cơ bản của Gmapping	26
4.1.2	Ưu điểm và nhược điểm của Gmapping	27
4.2	Triển Khai SLAM Gmapping Trên Robot Omni 4 Bánh	28
4.2.1	Cấu Hình Gói Gmapping	28
4.2.2	Tích Hợp Dữ Liệu Cảm Biến (lidar và IMU)	28
4.2.3	Chạy Mô Phỏng SLAM Gmapping (Khởi chạy môi trường mô phỏng và các node liên quan)	29
4.2.4	Lưu Trữ Bản Đồ Đã Xây Dựng	31
4.3	Đánh Giá Và Phân Tích Kết Quả Mô Phỏng	31
5	NAVIGATION CHO ROBOT	33
5.1	Mục tiêu	33
5.2	Tổng quan về ROS Navigation Stack	33
5.3	Quy trình triển khai hệ thống điều hướng	34
5.3.1	Bước 1: Cấu hình Costmap	34
5.3.2	Bước 2: Cấu hình Move Base	36
5.3.3	Bước 3: Cấu hình Local Planner	36
5.3.4	Bước 4: Định vị Robot với AMCL	40
5.3.5	Bước 5: Gửi và Điều khiển Mục tiêu	40
5.3.6	Bước 6: Kiểm tra và Đánh giá	43

5.4 Phát triển trong tương lai	43
6 KẾT QUẢ VÀ ĐÁNH GIÁ	45
7 CÂU HỎI VÀ TRẢ LỜI CÂU HỎI	46

Danh sách hình vẽ

1.1	Bánh xe Omni	2
2.1	Phân tích động học	6
3.1	Bố trí bánh xe omni vuông góc với trục	15
3.2	Bố trí bánh xe omni theo hình trục X lệch góc	16
3.3	Thân dưới và động cơ	17
3.4	Thân trên	18
3.5	Kiểu bánh Omni	19
3.6	Sensor	19
3.7	Gun	20
3.8	Omni car	20
3.9	Thêm Add-in Export as URDF vào SolidWorks	21
3.10	Thiết lập cấu trúc URDF	21
3.11	Tạo point	22
3.12	Đặt trục	23
3.13	Chỉnh các tham số Joint	23
3.14	Chỉnh các tham số về khối lượng và quán tính	24
3.15	Trình xem URDF trực tuyến	24
3.16	Mô hình urdf	25
4.1	spawn robot	30
4.2	Robot di chuyển quét map trong rviz	30

4.3	Map được tạo ra khi robot quét hoàn thành	30
4.4	Map tạo ra khi sử dụng odometry test	31
4.5	Map được quét khi sử dụng odometry automatic	32
5.1	Robot đang xác định vị trí	40
5.2	rqt graph cho navigation	42

CHƯƠNG 1

GIỚI THIỆU

Chương này giới thiệu về robot di động 4 bánh đa hướng (omni-directional) và bài toán thiết kế tập trung vào khả năng quét bản đồ (mapping) và tự định vị đồng thời (SLAM - Simultaneous Localization and Mapping) để điều hướng tự động. Từ đó, sẽ phân tích các thách thức chính trong quá trình nghiên cứu và thiết kế hệ thống, đồng thời xác định các mục tiêu cụ thể mà dự án hướng đến trong việc phát triển robot omni 4 bánh có khả năng SLAM navigation.

1.1 Bối cảnh

Trong lĩnh vực robot di động, khả năng di chuyển linh hoạt và tự chủ điều hướng là yếu tố then chốt để ứng dụng robot vào nhiều môi trường và tác vụ khác nhau. Robot di động 4 bánh đa hướng nổi lên như một nền tảng hứa hẹn nhờ khả năng di chuyển độc lập theo mọi hướng mà không cần xoay thân, mang lại sự linh hoạt vượt trội trong không gian hẹp và các ứng dụng đòi hỏi độ chính xác cao.

Kết hợp khả năng di chuyển omni với công nghệ SLAM navigation mở ra tiềm năng to lớn cho robot. SLAM cho phép robot đồng thời xây dựng bản đồ môi trường xung quanh và xác định vị trí của chính nó trong bản đồ đó, loại bỏ sự phụ thuộc vào các hệ thống định vị bên ngoài. Việc phát triển một robot omni 4 bánh có khả năng SLAM navigation không chỉ là một bài toán kỹ thuật phức tạp mà còn hứa hẹn mang lại những giải pháp tự động hóa thông minh cho các lĩnh vực như kho vận, dịch vụ, và thám hiểm.

1.2 Bánh Omni

1.2.1 Tổng quan về bánh Omni

Bánh Omni, hay còn gọi là bánh đa hướng, là một loại bánh xe đặc biệt được thiết kế để di chuyển theo mọi hướng một cách linh hoạt mà không cần xoay trực bánh xe. Cấu tạo độc đáo của bánh Omni bao gồm một bánh xe chính được bao quanh bởi các con lăn nhỏ hơn, đặt vuông góc với chu vi bánh xe. Các con lăn này cho phép bánh xe trượt ngang một cách dễ dàng trong khi vẫn có thể lăn về phía trước hoặc phía sau như một bánh xe thông thường.

Cấu tạo cơ bản của bánh Omni bao gồm:

- **Vành bánh xe chính:** Cung cấp lực đẩy chính để di chuyển tiến và lùi.
- **Các con lăn phụ:** Các bánh xe nhỏ hơn bao quanh vành bánh xe chính, cho phép di chuyển ngang hoặc trượt. Các con lăn này thường được làm từ vật liệu có độ ma sát thấp để giảm thiểu lực cản khi trượt.
- **Trục bánh xe:** Kết nối bánh xe với hệ thống truyền động của robot.



Hình 1.1: Bánh xe Omni

1.2.2 Tại sao chọn bánh Omni để thiết kế?

Robot di động sử dụng bánh Omni được ưu tiên trong các ứng dụng đòi hỏi sự linh hoạt và khả năng di chuyển trong không gian hẹp vì những ưu điểm sau:

- **Khả năng di chuyển đa hướng:** Robot có thể di chuyển tiến, lùi, sang trái, sang phải, và xoay tại chỗ một cách mượt mà mà không cần thực hiện các thao tác lái phức tạp.
- **Tính cơ động cao trong không gian hẹp:** Khả năng di chuyển ngang và xoay tại chỗ giúp robot dễ dàng di chuyển và thao tác trong các môi trường chật hẹp.
- **Điều khiển đơn giản:** Mặc dù khả năng di chuyển phức tạp, việc điều khiển robot omni thường đơn giản hơn so với robot bánh lái truyền thống, đặc biệt khi kết hợp với các thuật toán điều khiển vector.
- **Phù hợp cho các ứng dụng SLAM:** Khả năng di chuyển chính xác và linh hoạt là rất quan trọng cho các thuật toán SLAM, giúp robot thu thập dữ liệu cảm biến tốt hơn và xây dựng bản đồ môi trường hiệu quả hơn.

Thông thường, robot omni sử dụng cấu hình 3 hoặc 4 bánh xe để đảm bảo sự ổn định và khả năng chịu tải. Với cấu hình 4 bánh, mỗi bánh xe thường được điều khiển độc lập về tốc độ và hướng quay để tạo ra các chuyển động phức tạp của robot.

1.3 Yêu cầu thiết kế và mục tiêu của dự án

1.3.1 Yêu cầu thiết kế

Đề tài này nghiên cứu, thiết kế Robot di động 4 bánh đa hướng có khả năng quét bản đồ và tự định vị SLAM navigation. Các mục tiêu cụ thể của đề tài như sau:

- Xây dựng một robot di động với hệ thống dẫn động 4 bánh omni.
- Tích hợp các cảm biến cần thiết (ví dụ: lidar, camera, IMU) để thu thập dữ liệu môi trường.

1.3.2 Mục tiêu của dự án

Dự án này nhằm thiết kế, xây dựng và thử nghiệm một robot di động 4 bánh đa hướng có khả năng SLAM navigation, với các mục tiêu chính:

- Tạo ra một nền tảng robot di động linh hoạt, có khả năng di chuyển tự do trong môi trường trong nhà và ngoài trời.
- Nghiên cứu và triển khai các thuật toán SLAM hiện đại để đạt được khả năng lập bản đồ chính xác và định vị tin cậy.
- Phát triển các phương pháp điều khiển và lập kế hoạch đường đi hiệu quả dựa trên bản đồ SLAM.
- Đánh giá hiệu suất của hệ thống trong các tình huống và môi trường thử nghiệm khác nhau.
- Góp phần vào nghiên cứu và phát triển các ứng dụng robot tự động trong các lĩnh vực như vận chuyển, giám sát, và dịch vụ.

Thiết kế này kỳ vọng sẽ cung cấp một giải pháp robot di động tự chủ mạnh mẽ, có tiềm năng ứng dụng rộng rãi trong nhiều lĩnh vực của đời sống và công nghiệp.

1.4 Bố cục tiểu luận

Bố cục của tiểu luận được trình bày thành 3 chương. Mỗi chương sẽ trình bày và thảo luận các vấn đề khác nhau liên quan đến tiểu luận và được tóm tắt như sau:

Chương 1: Giới thiệu

Tổng quan về đề tài, robot Omni.

Chương 2: Cơ sở lý thuyết

Trình bày về động học hệ thống robot 4 bánh Omni và cách tính odometry.

Chương 3: Thiết kế robot 4 bánh Omni

Trình bày về bản thiết trong Solidwork và cách xuất file urdf.

Chương 4: Thiết lập slam gmapping cho robot omni

Trình bày về thuật toán slam gmapping và cách triển khai trong Ros.

Chương 5: Navigation cho robot

Trình bày về navigation và cách triển khai hệ thống.

Chương 6: Kết quả và đánh giá

Tổng kết và đánh giá các công việc đã thực hiện và các kết quả đạt được. Bên cạnh đó, đưa ra một số hướng phát triển của dự án trong tương lai.

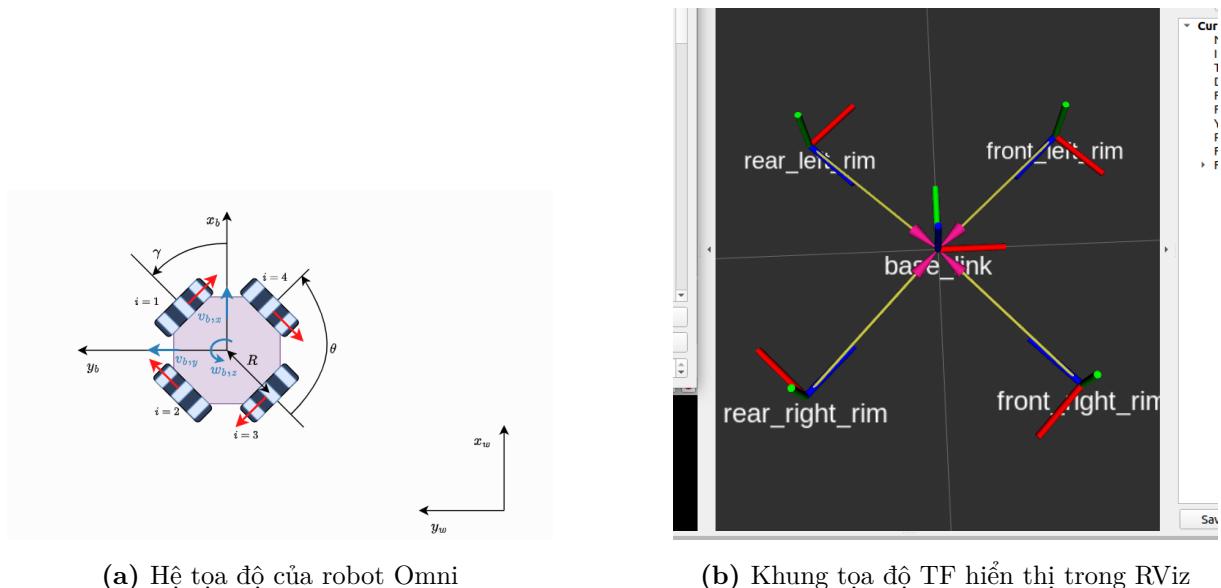
CHƯƠNG 2

CƠ SỞ LÝ THUYẾT

2.1 Động học của bánh Omni

2.1.1 Động học ngược

Động học ngược, hay còn gọi là động học nghịch, là quá trình xác định vận tốc góc cần thiết của từng bánh xe để robot đạt được một vận tốc tuyến tính và vận tốc góc mong muốn.



Hình 2.1: Phân tích động học

Các tham số được sử dụng trong phân tích động học:

- x_b, y_b là hệ tọa độ gắn với thân robot (body-frame), thường được đặt tại điểm tiếp xúc của một bánh xe với mặt đất hoặc tâm của robot.
- x_w, y_w là hệ tọa độ thế giới (world coordinate system), đóng vai trò là hệ quy

chiều tinh.

- $v_{b,x}$ là vận tốc tuyến tính của robot theo trục x của hệ tọa độ thân.
- $v_{b,y}$ là vận tốc tuyến tính của robot theo trục y của hệ tọa độ thân.
- $\omega_{b,z}$ là vận tốc góc của robot quanh trục z của hệ tọa độ thân.
- R là bán kính của robot, thường được hiểu là khoảng cách từ tâm robot đến các bánh xe (đối với cấu hình bánh xe đối xứng).
- Mũi tên màu đỏ trên bánh xe thứ i biểu thị chiều dương của vận tốc góc ω_i của bánh xe đó.
- γ là góc lệch của bánh xe thứ nhất so với trục x_b của hệ tọa độ thân.
- θ là góc giữa mỗi bánh xe, có thể được tính bằng công thức

$$\theta = \frac{2\pi}{n}$$

trong đó n là số lượng bánh xe.

Dựa vào phân tích hệ tọa độ và mối quan hệ giữa vận tốc của robot và vận tốc của các bánh xe, chúng ta có thể thiết lập ma trận Jacobian A để tính toán vận tốc góc cần thiết của từng bánh xe (ω_i) để đạt được vận tốc tuyến tính ($v_{b,x}, v_{b,y}$) và vận tốc góc ($\omega_{b,z}$) mong muốn của robot. Ma trận này phụ thuộc vào số lượng bánh xe (n), góc giữa các bánh xe (θ), và góc lệch của bánh xe đầu tiên so với trục x_b (γ), cũng như bán kính của robot (R).

Phương trình động học ngược có thể được biểu diễn dưới dạng ma trận như sau:

$$\begin{bmatrix} \omega_1 \\ \omega_2 \\ \omega_3 \\ \omega_4 \\ \vdots \\ \omega_n \end{bmatrix} = \frac{1}{r} \begin{bmatrix} \sin(\gamma) & -\cos(\gamma) & -R \\ \sin(\theta + \gamma) & -\cos(\theta + \gamma) & -R \\ \sin(2\theta + \gamma) & -\cos(2\theta + \gamma) & -R \\ \sin(3\theta + \gamma) & -\cos(3\theta + \gamma) & -R \\ \vdots & \vdots & \vdots \\ \sin((n-1)\theta + \gamma) & -\cos((n-1)\theta + \gamma) & -R \end{bmatrix} \begin{bmatrix} v_{b,x} \\ v_{b,y} \\ \omega_{b,z} \end{bmatrix}$$

Trong đó:

- $\omega_1, \omega_2, \dots, \omega_n$ là vận tốc góc của các bánh xe.
- r là bán kính của các bánh xe.
- $v_{b,x}, v_{b,y}, \omega_{b,z}$ là vận tốc tuyến tính theo trục x, vận tốc tuyến tính theo trục y, và vận tốc góc quanh trục z của robot trong hệ tọa độ thân.

Phương trình trên cho thấy mối liên hệ trực tiếp giữa "ý muốn" chuyển động của robot ($v_{b,x}, v_{b,y}, \omega_{b,z}$) và "hành động" cần thiết của các bánh xe ($\omega_1, \omega_2, \dots, \omega_n$).

Ta cũng có thể biểu diễn vận tốc góc của bánh xe thứ i một cách tường minh như sau:

$$\omega_i = \frac{\sin((i-1)\theta + \gamma)v_{b,x} - \cos((i-1)\theta + \gamma)v_{b,y} - R\omega_{b,z}}{r}$$

Công thức này cho phép chúng ta tính toán vận tốc góc cần thiết cho từng bánh xe một cách độc lập, dựa trên vận tốc mong muốn của robot và các thông số cấu hình của hệ thống bánh xe omni. Việc xác định chính xác ma trận Jacobian và các phương trình động học ngược là bước quan trọng để xây dựng hệ thống điều khiển chuyển động chính xác cho robot omni 4 bánh.

2.1.2 Động học thuận

Động học thuận là quá trình tính toán vận tốc của thân robot ($v_{b,x}, v_{b,y}, \omega_{b,z}$) dựa trên vận tốc góc của các bánh xe (ω_i). Trong trường hợp robot sử dụng bánh xe omni, mỗi quan hệ này có thể được thiết lập thông qua ma trận Jacobian A đã được giới thiệu ở phần động học ngược. Tuy nhiên, để tìm ra vận tốc thân robot từ vận tốc bánh xe, chúng ta cần sử dụng nghịch đảo (hoặc giả nghịch đảo - pseudoinverse) của ma trận A , ký hiệu là A^\dagger .

Phương trình động học thuận có thể được biểu diễn như sau:

$$\begin{bmatrix} v_{b,x} \\ v_{b,y} \\ \omega_{b,z} \end{bmatrix} = r A^\dagger \begin{bmatrix} \omega_1 \\ \omega_2 \\ \omega_3 \\ \omega_4 \\ \vdots \\ \omega_n \end{bmatrix}$$

Trong đó:

- $v_{b,x}, v_{b,y}, \omega_{b,z}$ là vận tốc tuyến tính theo trục x, vận tốc tuyến tính theo trục y, và vận tốc góc quanh trục z của robot trong hệ tọa độ thân.
- r là bán kính của các bánh xe.
- A^\dagger là ma trận giả nghịch đảo của ma trận Jacobian A .
- $\omega_1, \omega_2, \dots, \omega_n$ là vận tốc góc của các bánh xe.

Vì việc sử dụng giả nghịch đảo A^\dagger là cần thiết vì ma trận A thường không phải là ma trận vuông (đặc biệt khi số lượng bánh xe $n \neq 3$) và do đó không có nghịch đảo thông thường. Giả nghịch đảo cho phép chúng ta tìm ra nghiệm tối ưu (ví dụ: nghiệm có norm nhỏ nhất) cho hệ phương trình tuyến tính xác định vận tốc thân robot dựa trên vận tốc bánh xe.

Để tính toán A^\dagger , chúng ta có thể sử dụng công thức Moore-Penrose pseudoinverse:

$$A^\dagger = (A^T A)^{-1} A^T$$

nếu $A^T A$ khả nghịch, hoặc một công thức tổng quát hơn sử dụng phân tích giá trị suy dí (Singular Value Decomposition - SVD) trong trường hợp $A^T A$ không khả nghịch.

Một khi đã xác định được ma trận giả nghịch đảo A^\dagger , chúng ta có thể dễ dàng tính toán vận tốc của robot dựa trên vận tốc góc của các bánh xe. Điều này rất quan trọng trong việc ước tính trạng thái của robot và thực hiện các tác vụ điều khiển dựa trên thông tin phản hồi từ các encoder bánh xe.

2.2 Tính toán Odometry

Odometry ước tính vị trí và hướng robot dựa trên dữ liệu cảm biến chuyển động (ví dụ: encoder bánh xe).

Quan trọng trong Gmapping:

- Ước tính chuyển động ban đầu để ghép các lần quét lidar.
- Giảm độ trễ và tăng tính nhất quán của bản đồ.
- Cung cấp ràng buộc chuyển động.

Quan trọng trong Navigation:

- Ước tính vị trí hiện tại cho việc lập kế hoạch và theo dõi đường đi.
- Phản hồi cho hệ thống điều khiển chuyển động.
- Thường được kết hợp với các cảm biến khác để tăng độ chính xác.

Tóm lại, odometry là nền tảng cho cả việc xây dựng bản đồ và điều hướng tự động của robot.

2.2.1 Dựa vào Gazbo

A Dựa vào Plugin

Việc tính toán odometry thường được thực hiện thông qua các plugin điều khiển bánh xe (ví dụ: `diff_drive_controller`, `planar_move_controller`), dựa trên dữ liệu encoder (thông qua các topic như `/wheel_encoders`) và mô hình động học tích hợp trong plugin. Các plugin này publish thông tin odometry (topic `/odom`, message `nav_msgs/Odometry`).

Tuy nhiên, đối với các mô hình robot có cấu hình bánh xe đặc biệt mà không có plugin điều khiển chuyên dụng, việc sử dụng một plugin không phù hợp (ví dụ: dùng `diff_drive_controller` cho robot omni) đồng nghĩa với việc bỏ qua các đặc tính di chuyển độc đáo của bánh xe. Mặc dù vậy, trong một số trường hợp, việc "ép" sử dụng một plugin tiêu chuẩn có thể mang lại độ chính xác cao hơn trong việc ước tính odometry và do đó, giúp quá trình quét bản đồ (ví dụ: với Gmapping) diễn ra chuẩn xác hơn so với việc triển khai một phương pháp odometry tùy chỉnh.

nhưng chưa được tối ưu hóa. Điều này đặc biệt đúng nếu plugin được cấu hình và hiệu chuẩn cẩn thận để phần nào mô phỏng được chuyển động thực tế của robot.

B Dựa vào Model Gazebo

Một phương pháp khác để thu thập thông tin odometry, đặc biệt trong môi trường mô phỏng như Gazebo, là trực tiếp truy vấn trạng thái của model robot trong trình mô phỏng. Thay vì tính toán odometry dựa trên dữ liệu encoder và mô hình động học, chúng ta có thể gọi các dịch vụ (services) của Gazebo để lấy thông tin vị trí (x, y, z) và hướng (quaternion) hiện tại của robot.

Trong trường hợp này, việc điều khiển bánh xe vẫn có thể được thực hiện thông qua các phương pháp động học, gửi lệnh vận tốc tới các khớp bánh xe trong Gazebo. Tuy nhiên, việc tính toán và publish odometry lại được thực hiện bằng cách truy cập trực tiếp trạng thái của model robot trong Gazebo.

Ví dụ, đoạn code Python (sử dụng thư viện rospy của ROS) minh họa cách thực hiện điều này:

- Node ROS `odom_pub` được khởi tạo để publish topic `/odom`.
- Một service proxy được tạo để gọi service `/gazebo/get_model_state` của Gazebo.
- Một broadcaster `tf` được sử dụng để publish transform giữa frame `odom` và `base_footprint`.
- Trong vòng lặp chính:
 - Service `/gazebo/get_model_state` được gọi để lấy trạng thái của model có tên `omni_car`.
 - Thông tin vị trí (x, y) và hướng (z, w quaternion) được trích xuất từ kết quả.
 - Transform (vị trí và hướng) được publish qua `tf`.
 - Một message `nav_msgs/Odometry` được tạo và populated với thông tin vị trí, hướng, vận tốc tuyến tính và vận tốc góc (cũng lấy từ trạng thái model Gazebo).
 - Message `Odometry` được publish lên topic `/odom`.

Phương pháp này có ưu điểm là cung cấp thông tin odometry rất chính xác trong môi trường mô phỏng, vì nó trực tiếp lấy dữ liệu từ "thế giới thực" ảo của Gazebo, loại bỏ các sai số có thể phát sinh từ việc ước tính dựa trên encoder và mô hình động học. Điều này đặc biệt hữu ích cho việc phát triển và kiểm thử các thuật toán SLAM và navigation trong môi trường mô phỏng với độ tin cậy cao về dữ liệu odometry. Tuy nhiên, cần lưu ý rằng phương pháp này chỉ khả dụng trong môi trường mô phỏng có tích hợp Gazebo và không thể áp dụng trực tiếp cho robot thật.

2.2.2 Dựa vào động học Robot

Đối với các hệ thống robot mà không sử dụng các plugin điều khiển bánh xe tiêu chuẩn hoặc khi cần một phương pháp ước tính odometry tùy chỉnh dựa trên đặc tính động học cụ thể của robot, chúng ta có thể tự triển khai việc tính toán odometry dựa trên dữ liệu từ các cảm biến chuyển động và mô hình động học của robot.

Quá trình tính toán odometry dựa trên động học robot thường bao gồm các bước chính sau:

1. Khởi tạo:

- Khởi tạo node ROS.
- Đọc các tham số cấu hình của robot (ví dụ: bán kính bánh xe r , khoảng cách giữa các bánh xe L , hệ số hiệu chỉnh).
- Khởi tạo ma trận động học \mathbf{H} liên hệ giữa vận tốc bánh xe ω và vận tốc thân robot $\mathbf{v}_b = [v_x, v_y, \omega_z]^T$, dựa trên cấu hình bánh xe ($\mathbf{v}_b = \mathbf{H}\omega$). Đối với robot omni 4 bánh, ma trận này thường liên quan đến góc bố trí của từng bánh xe.
- Khởi tạo các biến lưu trữ trạng thái ước tính của robot (vị trí x, y , hướng θ) và thời gian cập nhật trước đó t_{last} .
- Thiết lập các bộ đệm (nếu cần) để làm mịn dữ liệu vận tốc.
- Tạo các publisher để gửi thông tin odometry (topic `/odom`, message `nav_msgs/Odometry` và transform TF (giữa frame `odom` và `base_footprint`)).
- Tạo các subscriber để nhận dữ liệu từ các cảm biến chuyển động (ví dụ: topic chứa vận tốc bánh xe từ encoder, topic IMU).

2. Xử lý dữ liệu vận tốc bánh xe (Callback):

- Khi nhận được dữ liệu vận tốc từ các bánh xe ω :
- Sắp xếp vận tốc của từng bánh xe theo đúng thứ tự tương ứng với cấu hình động học.
- Sử dụng ma trận động học \mathbf{H} để tính toán vận tốc tuyến tính theo trục x và y (v_x, v_y), cũng như vận tốc góc quanh trục z (ω_z) của thân robot (trong hệ tọa độ robot): $\mathbf{v}_b = \mathbf{H}\omega$.
- Áp dụng các hệ số hiệu chỉnh (ví dụ: bù trượt k_{slip}) nếu cần: $v'_x = k_{slip}v_x$, $v'_y = k_{slip}v_y$.
- Tính toán khoảng thời gian $\Delta t = t_{current} - t_{last}$ giữa các lần nhận dữ liệu.
- Cập nhật vị trí x, y và hướng θ của robot trong hệ tọa độ thế giới bằng cách tích phân vận tốc theo thời gian, có xét đến hướng hiện tại của robot:

$$\begin{aligned}x_{new} &= x_{old} + (v'_x \cos(\theta_{old}) - v'_y \sin(\theta_{old}))\Delta t \\y_{new} &= y_{old} + (v'_x \sin(\theta_{old}) + v'_y \cos(\theta_{old}))\Delta t \\\theta_{new} &= \theta_{old} + \omega_z \Delta t\end{aligned}$$

- Thực hiện làm mịn vận tốc (nếu sử dụng bộ đệm).

3. Xử lý dữ liệu IMU (Callback - tùy chọn):

- Khi nhận được dữ liệu từ IMU:
- Lọc và làm mịn dữ liệu vận tốc góc và/hoặc hướng.
- Sử dụng thông tin hướng từ IMU (thường chính xác hơn trong ngắn hạn so với odometry bánh xe) để hiệu chỉnh hoặc thay thế ước tính hướng θ từ odometry bánh xe.

4. Publish thông tin Odometry:

- Tạo message `nav_msgs/Odometry`.
- Diền các thông tin về header (timestamp, frame IDs).
- Diền thông tin về vị trí ($x, y, 0$) và hướng (quaternion \mathbf{q} được chuyển đổi từ θ).
- Diền thông tin về vận tốc tuyến tính (v'_x, v'_y) và vận tốc góc (ω_z) của robot.
- Publish message `Odometry` lên topic `/odom`.
- Broadcast transform TF từ frame `base_footprint` đến frame `odom` để biểu diễn vị trí và hướng của robot trong không gian.

5. Vòng lặp chính (Run):

- Duy trì node ROS hoạt động để liên tục xử lý dữ liệu cảm biến và publish thông tin odometry.

Phương pháp này cho phép tính toán odometry một cách linh hoạt, tùy thuộc vào cấu hình và các cảm biến có sẵn trên robot. Việc tích hợp dữ liệu từ nhiều nguồn cảm biến (ví dụ: encoder và IMU) thường giúp cải thiện độ chính xác và độ tin cậy của ước tính odometry. Tuy nhiên, việc tính toán odometry theo kiểu này đòi hỏi độ chính xác rất cao trong cả mô hình động học và dữ liệu cảm biến, bởi vì sai số nhỏ có thể dễ dàng tích lũy theo thời gian, dẫn đến các lỗi lớn hơn trong việc xây dựng bản đồ (Gmapping) và điều hướng (Navigation).

CHƯƠNG 3

THIẾT KẾ ROBOT 4 BÁNH OMNI

3.1 Thiết kế Solidwork

Ý tưởng thiết kế: Robot di động dạng omnidirectional car với một khẩu pháo được đặt trên thân và tích hợp các cảm biến hỗ trợ cho các tác vụ nhận thức môi trường và điều hướng.

3.1.1 Thân xe và cách bố trí bánh

Đối với robot di động sử dụng bánh xe omni, có hai kiểu bố trí trực bánh xe phổ biến:

- Bánh xe có trực vuông góc (hoặc song song) với trực chuyển động chính của robot.



Hình 3.1: Bố trí bánh xe omni vuông góc với trực

- Bánh xe được bố trí theo hình trục X, tạo một góc lệch so với các trục XY của hệ tọa độ thân robot.



Hình 3.2: Bố trí bánh xe omni theo hình trục X lệch góc

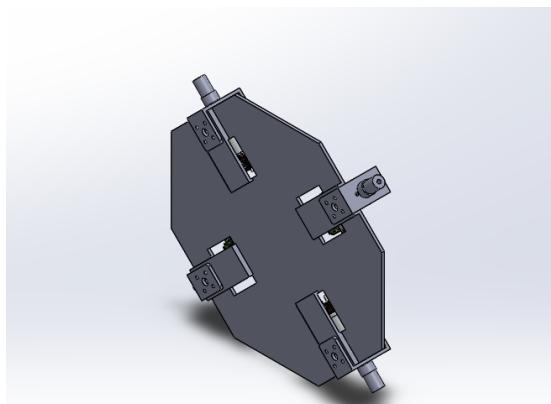
Trong thiết kế này, bạn em lựa chọn kiểu bố trí thứ hai:

- **Thân xe:** Sử dụng cấu trúc hình vuông với kích thước cạnh là 25 cm x 25 cm, mang lại sự cân bằng và không gian lắp đặt vừa đủ cho các thành phần khác của robot.
- **Bố trí bánh xe:** Bốn bánh xe omni được bố trí tại bốn góc của thân xe hình vuông. Điểm đặc biệt trong thiết kế này là trục của mỗi bánh xe không song song với các cạnh của thân xe mà tạo một góc lệch 45 độ so với trục gốc (base frame) của robot. Kiểu bố trí này cho phép robot di chuyển linh hoạt theo mọi hướng (tiến, lùi, trái, phải và xoay tại chỗ) một cách hiệu quả, tận dụng tối đa khả năng di chuyển đa hướng của bánh xe omni.

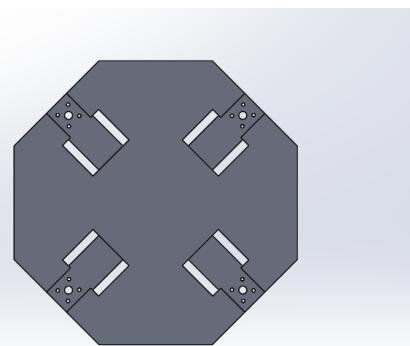
Việc lựa chọn bố trí bánh xe lệch 45 độ nhằm tối ưu hóa khả năng di chuyển đa hướng mượt mà và giảm thiểu hiện tượng trượt không mong muốn khi thực hiện các chuyển động phức tạp. Thiết kế thân xe hình vuông cung cấp một nền tảng ổn định để gắn khẩu pháo và các cảm biến, đồng thời dễ dàng trong việc phân bố trọng lượng và kết nối các thành phần điện tử.

3.1.2 Hình ảnh các bản thiết kế

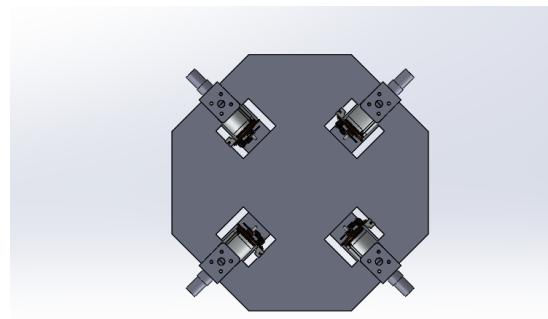
Thân dưới và động cơ:



(a) Thân dưới và động cơ - Góc 1



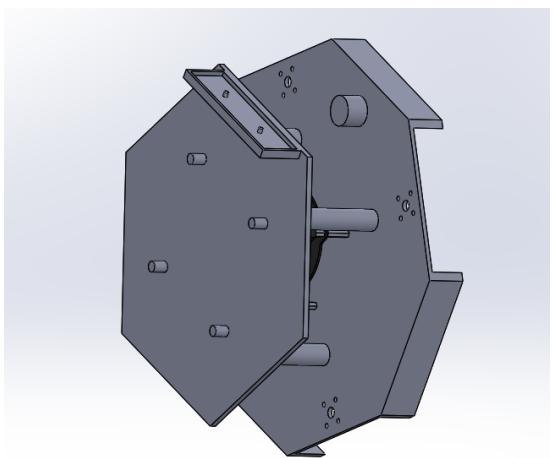
(b) Thân dưới và động cơ - Góc 2



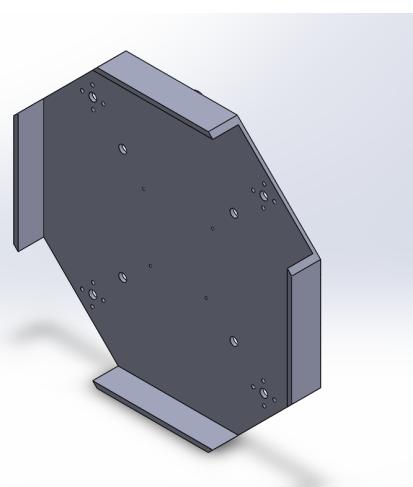
(c) Thân dưới và động cơ - Chi tiết

Hình 3.3: Thân dưới và động cơ

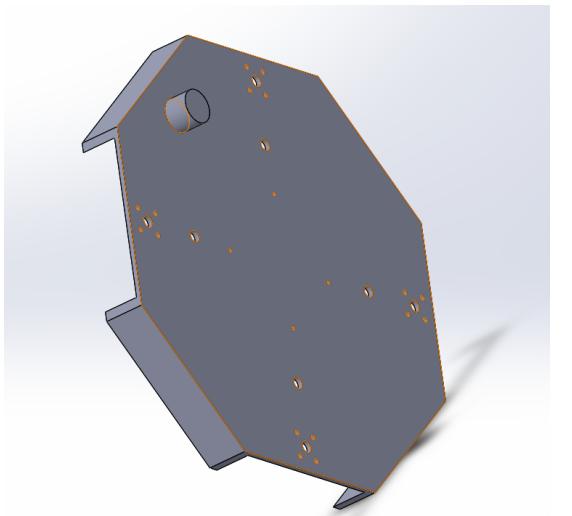
Thân trên:



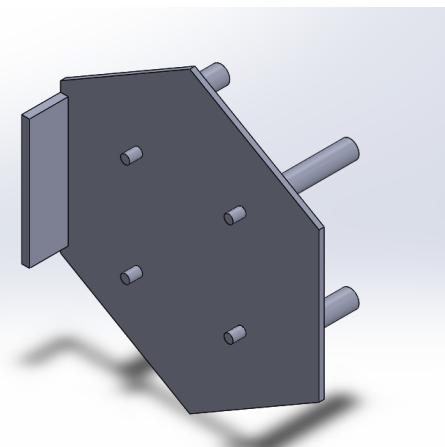
(a) Thân trên - Góc 1



(b) Thân trên - Góc 2



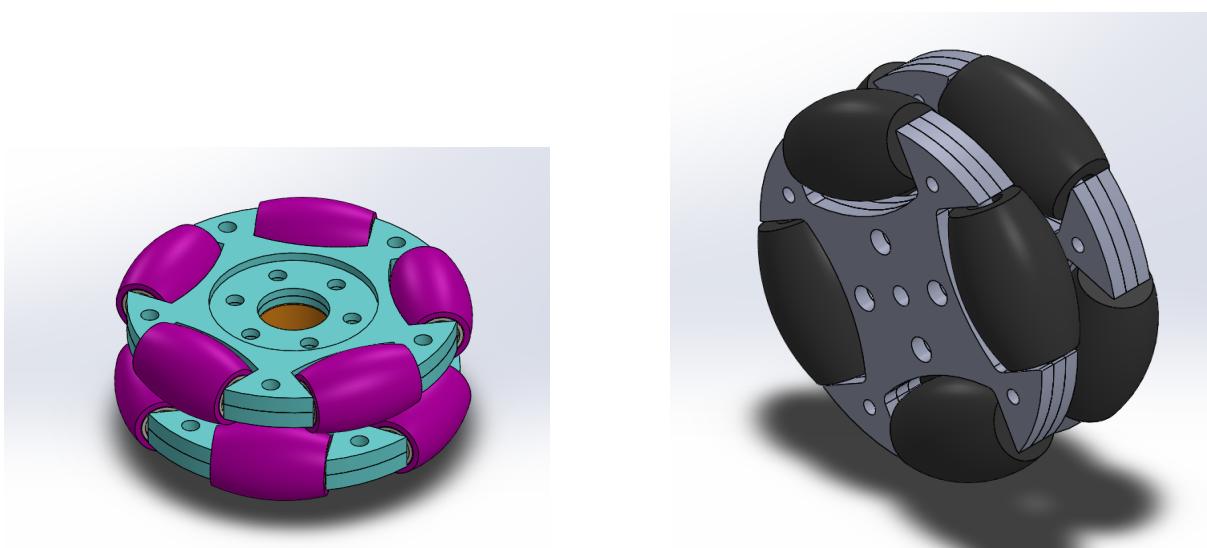
(c) Thân trên - Chi tiết 1



(d) Thân trên - Chi tiết 2

Hình 3.4: Thân trên

Bánh omni:

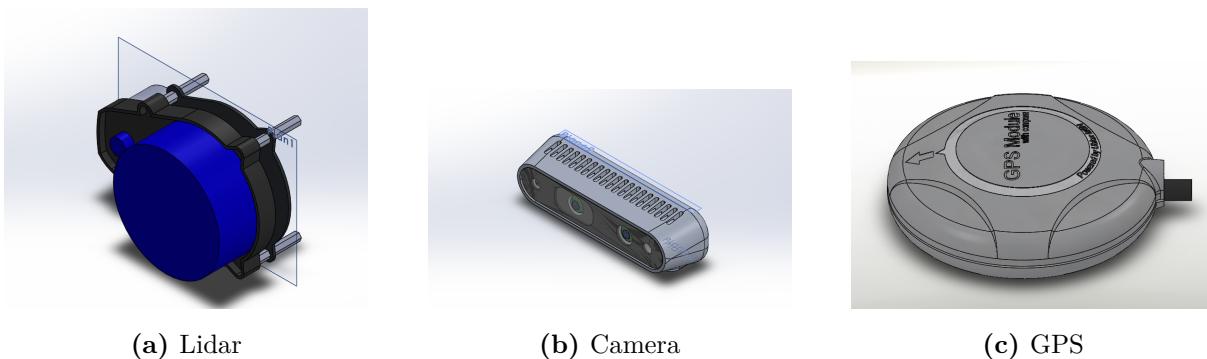


(a) Bánh omni - Kiểu 1

(b) Bánh omni - Kiểu 2

Hình 3.5: Kiểu bánh Omni

Các cảm biến:



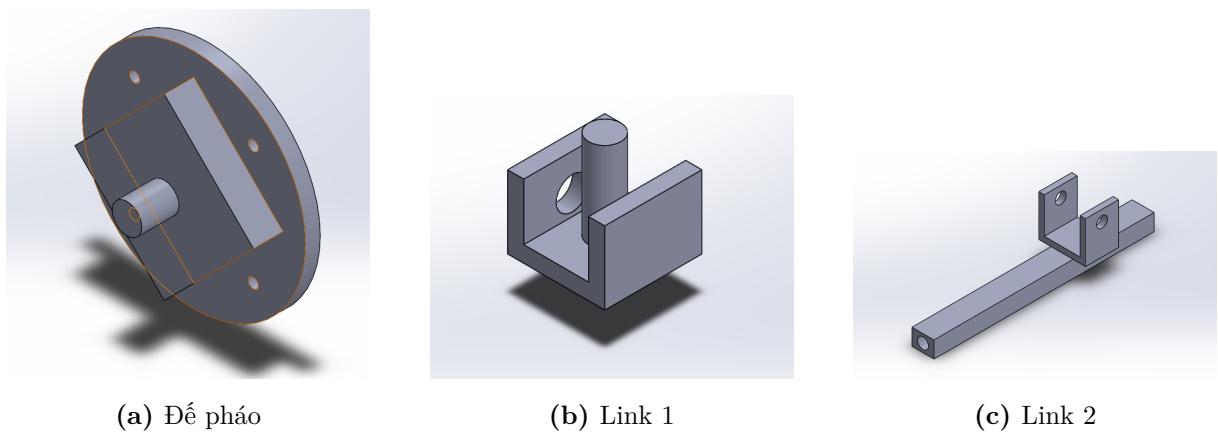
(a) Lidar

(b) Camera

(c) GPS

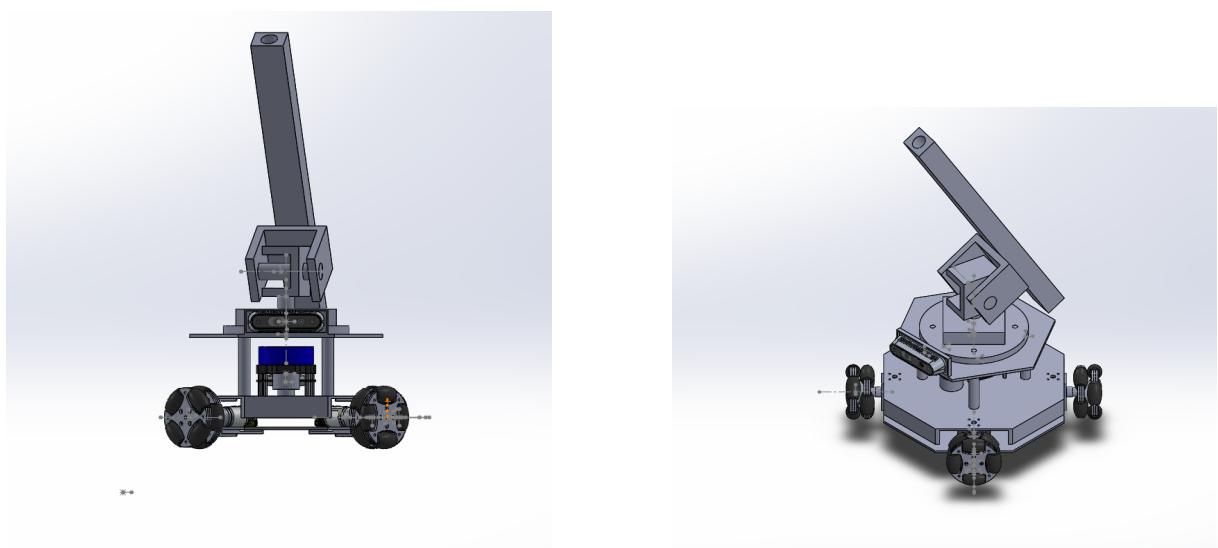
Hình 3.6: Sensor

Pháo (Tay máy hai bậc tự do):



Hình 3.7: Gun

Mô hình hoàn thiện:



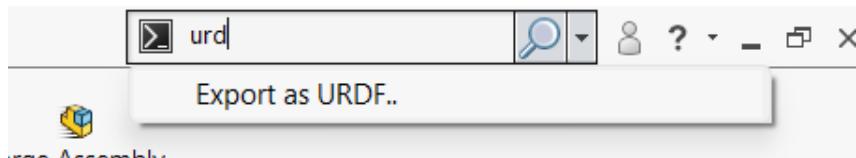
Hình 3.8: Omni car

3.2 Đặt trục và xuất file URDF

3.2.1 Cách đặt trục và xuất file URDF

Để đặt trục và xuất file URDF (Universal Robot Description Format) từ SolidWorks, bạn có thể thực hiện theo các bước sau:

1. Thêm Add-in Export as URDF vào SolidWorks:



Hình 3.9: Thêm Add-in Export as URDF vào SolidWorks

2. Thiết lập cấu trúc cho file URDF:

The image shows two side-by-side windows from the URDF Exporter add-in in SolidWorks.

(a) Thiết lập cấu trúc Parent-Child: This window shows the 'Configure and Organize Links' panel. It lists a 'base_link' node with several child links: 'rim_front_left_Link', 'lidar_Link', 'cam_Link', 'gps_Link', and 'de_Link'. The 'rim_front_left_Link' node has its own children: 'roller_e_rim_front_left_link', 'roller_ne_rim_front_left_link', 'roller_n_rim_front_left_link', 'roller_nw_rim_front_left_link', 'roller_w_rim_front_left_link', 'roller_sw_rim_front_left_link', 'roller_s_rim_front_left_link', and 'roller_se_rim_front_left_link'. A coordinate system icon is visible at the bottom.

(b) Chọn loại Joint: This window shows the 'URDF Exporter' panel. It is configuring a joint named 'roller_e_rim_front_left_joint' between 'rim_front_left_Link' and 'roller_e_rim_front_left_link'. The 'Joint Type' dropdown is set to 'continuous'. A tooltip 'Select the joint type' is visible over the dropdown. Below the dropdown is a numeric input field with value '0'.

(a) Thiết lập cấu trúc Parent-Child

(b) Chọn loại Joint

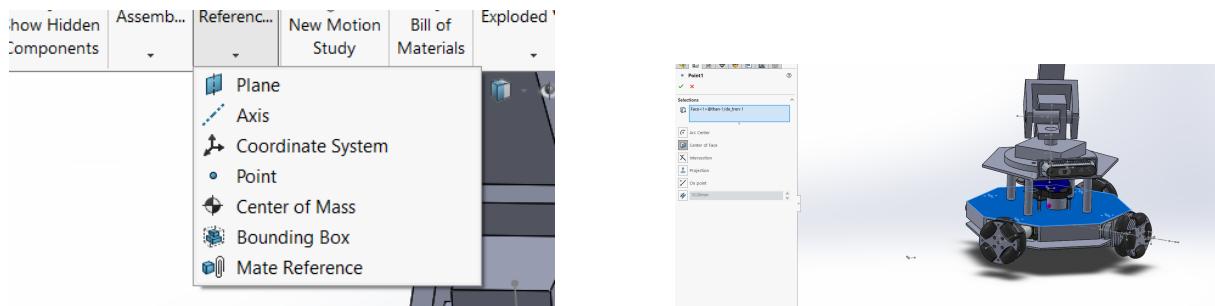
Hình 3.10: Thiết lập cấu trúc URDF

Lưu ý quan trọng: Để dễ dàng thiết lập cấu trúc URDF, nên tạo mỗi thành phần động học của robot (ví dụ: thân, bánh xe, các link của tay máy) thành một part riêng biệt trong SolidWorks trước khi assembly.

3. **Tiến hành Preview and Export:** Sau khi đã assembly và thiết lập các mối quan hệ Parent-Child cũng như loại khớp ban đầu, chọn chức năng "Preview and Export" trong add-in URDF để xem trước cấu trúc URDF sẽ được tạo ra.

4. Đặt lại trục cho các thành phần:

- **Tạo Point tham chiếu:** Tạo các điểm tham chiếu (Point) trong SolidWorks, thường là điểm chính giữa của mỗi part, để làm gốc cho trục tọa độ.

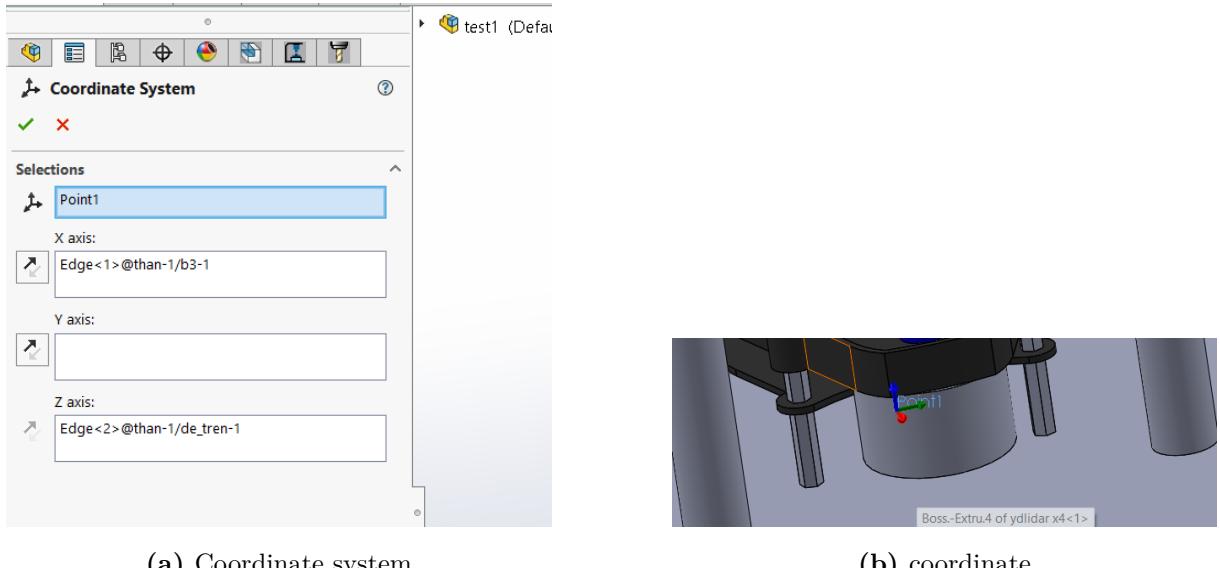


(a) Tạo Point tham chiếu cho trục

(b) Point reference

Hình 3.11: Tạo point

- **Xét trục theo Point và tham chiếu các trục chuẩn:** Thiết lập hướng của trục tọa độ cho mỗi link dựa trên các điểm tham chiếu đã tạo và tham chiếu đến các trục chuẩn X, Y, Z của SolidWorks:
 - **Trục X (màu đỏ):** Thường được chọn làm trục di chuyển theo phương tịnh tiến của khớp prismatic.
 - **Trục Y (màu xanh lá cây):**
 - **Trục Z (màu xanh dương):** Thường được chọn làm trục quay cho các khớp revolute hoặc continuous.

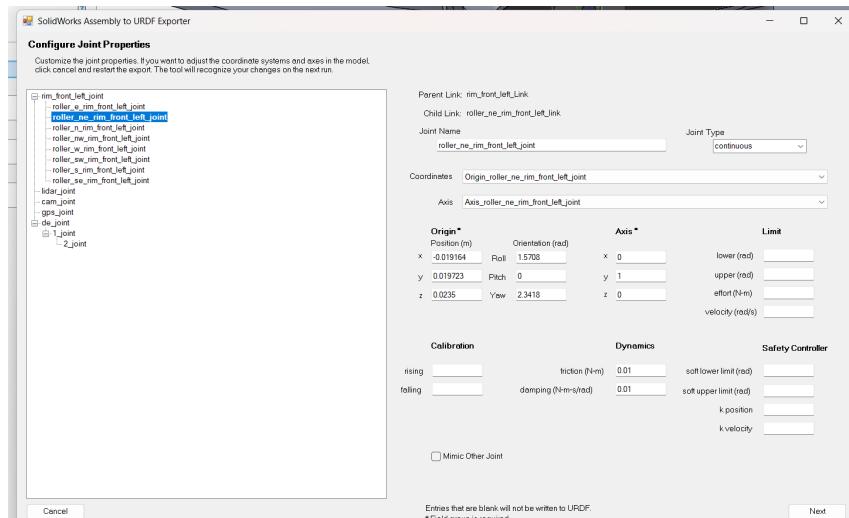


(a) Coordinate system

(b) coordinate

Hình 3.12: Đặt trục

5. Chỉnh các tham số Joint:

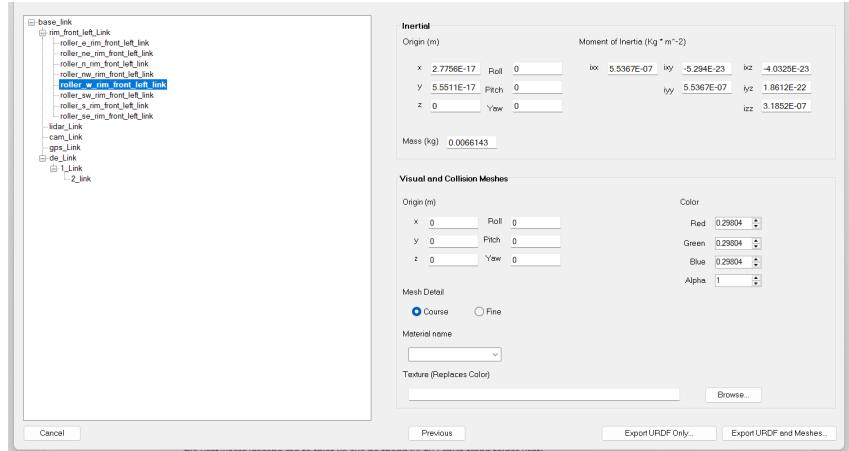


Hình 3.13: Chỉnh các tham số Joint

Sau khi hoàn tất việc đặt trục, chuyển sang bước chỉnh các tham số của khớp:

- **Axis* (Trục quay/trượt):** Đối với các khớp revolute hoặc prismatic, cần chỉ định trục nào là trục quay hoặc trượt bằng cách đặt giá trị tương ứng là 1 (hoặc -1 tùy theo hướng). Thông thường, lần xuất file URDF đầu tiên có thể gặp lỗi liên quan đến việc thiết lập trục này. Nếu xảy ra lỗi, hãy quay lại bước thiết lập cấu trúc và chỉnh sửa Axis cho đúng. SolidWorks thường lưu lại các thiết lập trước đó, giúp việc điều chỉnh trở nên dễ dàng hơn.

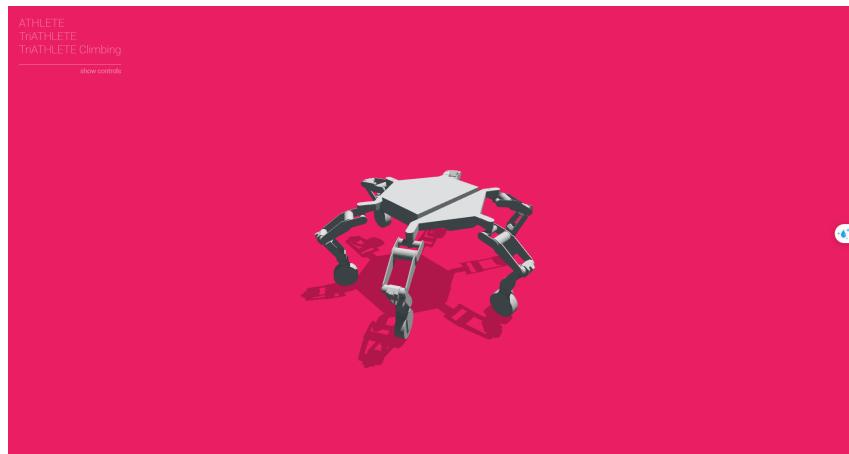
6. Tiếp tục chuyển sang chỉnh các tham số khác:



Hình 3.14: Chỉnh các tham số về khối lượng và quán tính

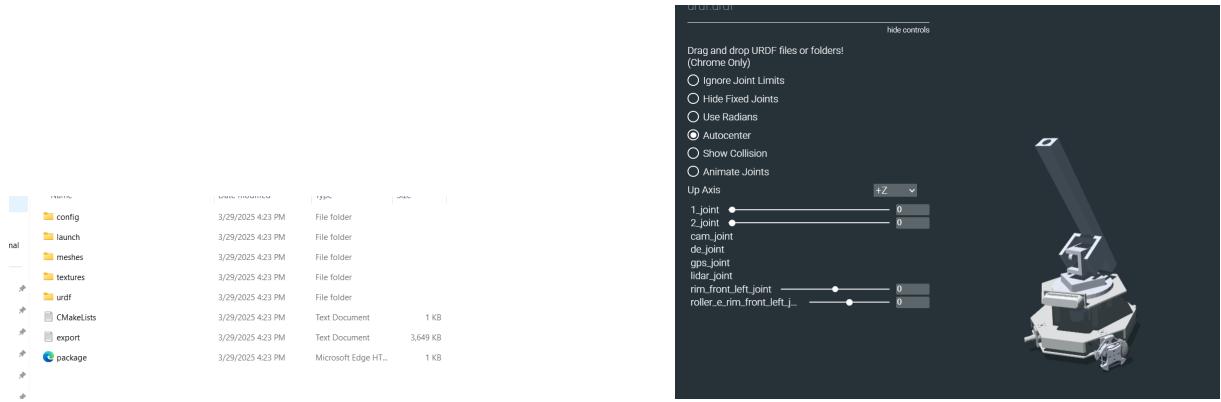
- Sau đó bấm Export URDF and Meshes... để xuất file URDF
-> Tạo ra 1 folder chứa các file

7. Bước cuối cùng check file URDF bằng:



Hình 3.15: Trình xem URDF trực tuyến

<https://gkjohnson.github.io/urdf-loaders/javascript/example/bundle/>



(a) Folder urdf

(b) Urdf car

Hình 3.16: Mô hình urdf

Kéo thả thư mục xuất ra được vào để check xem nêu lõi qua lại chỉnh axis.

Lưu ý về mô hình cụ thể: Do tính đối xứng của các bánh xe và roller trong mô hình của bạn, bạn có thể chỉ cần xuất file URDF cho một bánh xe và phần thân chính. Sau đó, bạn có thể chỉnh sửa trực tiếp file URDF để thêm định nghĩa cho các bánh xe còn lại bằng cách sao chép và điều chỉnh các thông số liên quan (ví dụ: vị trí tương đối).

CHƯƠNG 4

THIẾT LẬP SLAM GMAPPING CHO ROBOT OMNI

Mục Lục Mô Phỏng SLAM Gmapping Cho Robot Omni 4 Bánh

4.1 Giới Thiệu Thuật Toán Gmapping

4.1.1 Nguyên lý hoạt động cơ bản của Gmapping

Gmapping là một thuật toán SLAM dựa trên phương pháp **Particle Filter** (bộ lọc hạt). Nó hoạt động bằng cách duy trì một tập hợp các giả thuyết có thể về vị trí của robot và bản đồ môi trường xung quanh, được gọi là các "hạt". Mỗi hạt đại diện cho một trạng thái có thể của robot (vị trí và hướng) và một bản đồ lưới xác suất (occupancy grid map) tương ứng.

Quá trình hoạt động của Gmapping diễn ra theo các bước chính sau:

- **Khởi tạo:** Ban đầu, một số lượng hạt được tạo ra với vị trí và hướng ban đầu của robot (thường là gốc tọa độ). Mỗi hạt đi kèm với một bản đồ lưới xác suất rỗng.
- **Dự đoán (Prediction):** Khi robot di chuyển (dựa trên dữ liệu odometry từ encoder bánh xe), vị trí và hướng của mỗi hạt được cập nhật theo mô hình chuyển động của robot. Do sai số trong odometry, các hạt có xu hướng phân tán theo thời gian.
- **Cập nhật (Update):** Khi robot nhận được dữ liệu từ cảm biến (thường là lidar), thuật toán sẽ so sánh các phép đo cảm biến với bản đồ được duy trì bởi mỗi hạt. Các hạt có bản đồ phù hợp với các phép đo cảm biến sẽ được gán trọng số cao hơn, trong khi các hạt có bản đồ không phù hợp sẽ có trọng số thấp hơn.

- **Tạo lại mẫu (Resampling):** Để tránh sự suy giảm của tập hợp hạt (tức là tất cả các hạt đều có trọng số rất thấp trừ một vài hạt), thuật toán thực hiện quá trình tạo lại mẫu. Các hạt có trọng số cao hơn có nhiều khả năng được chọn để tạo ra các hạt mới, trong khi các hạt có trọng số thấp hơn có thể bị loại bỏ. Các hạt mới được tạo ra sẽ có vị trí và bản đồ gần giống với các hạt có trọng số cao.
- **Ước tính:** Vị trí và bản đồ cuối cùng của robot được ước tính dựa trên các hạt có trọng số cao nhất (hoặc trung bình có trọng số của tất cả các hạt). Bản đồ thường được chọn là bản đồ của hạt có trọng số cao nhất.

Điểm đặc biệt của Gmapping là nó xây dựng bản đồ **Lưới Xác Suất**. Mỗi ô trong lưới bản đồ lưu trữ xác suất mà ô đó bị chiếm bởi vật cản. Điều này cho phép xử lý sự không chắc chắn trong dữ liệu cảm biến và tạo ra bản đồ mượt mà hơn.

4.1.2 Ưu điểm và nhược điểm của Gmapping

Ưu điểm:

- Hiệu suất tốt trong môi trường nhỏ và vừa.
- Bản đồ lưới xác suất giúp xử lý nhiều cảm biến và tạo ra bản đồ rõ ràng hơn.
- Khả năng xử lý loop closure tốt (trong một số trường hợp).
- Thuật toán tương đối đơn giản để hiểu và triển khai.

Nhược điểm:

- Độ phức tạp tính toán cao, đặc biệt với số lượng hạt lớn.
- Khó khăn trong môi trường lớn và có nhiều đặc điểm lắp lại.
- Phụ thuộc vào chất lượng dữ liệu odometry.
- Ít hiệu quả với các cảm biến có trường nhìn hẹp.
- Không tích hợp IMU một cách trực tiếp.

4.2 Triển Khai SLAM Gmapping Trên Robot Omni 4 Bánh

4.2.1 Cấu Hình Gói Gmapping

Để triển khai Gmapping trong môi trường mô phỏng (ví dụ: ROS), bạn cần cấu hình gói gmapping. Các tham số quan trọng cần xem xét bao gồm:

- **scan topic:** Tên của topic mà robot publish dữ liệu quét lidar.
- **odom topic:** Tên của topic mà robot publish dữ liệu odometry.
- **Khung tọa độ (frames):**
 - **map_frame:** Tên của khung tọa độ cho bản đồ toàn cục (ví dụ: map).
 - **odom_frame:** Tên của khung tọa độ cho hệ tọa độ odometry (ví dụ: odom).
 - **base_frame:** Tên của khung tọa độ cho thân robot (ví dụ: base_link).
 - **scan_frame:** Tên của khung tọa độ cho cảm biến lidar (ví dụ: laser_frame).

Đảm bảo mối quan hệ giữa các khung tọa độ này được thiết lập chính xác thông qua **tf** (transformations).

- **Các tham số liên quan đến bộ lọc hạt:** particles, xmin, ymin, xmax, ymax, delta, maxUrange, maxRange.
- **Các tham số liên quan đến cập nhật bản đồ:** sigma, kernelSize, lstep, astep, iterations, lsigma, asigma, linearUpdate, angularUpdate, temporalUpdate.
- **Các tham số liên quan đến tạo lại mẫu:** resampleThreshold, transform_tolerance

Việc cấu hình các tham số này thường được thực hiện thông qua file launch của ROS hoặc trực tiếp qua command line khi chạy node **gmapping**.

4.2.2 Tích Hợp Dữ Liệu Cảm Biến (lidar và IMU)

Để Gmapping hoạt động, nó cần dữ liệu từ cảm biến lidar và dữ liệu odometry.

- **Lidar:** Node lidar mô phỏng cần publish dữ liệu dưới dạng message **sensor_msgs/LaserScan** trên topic đã cấu hình (ví dụ: /scan). Đảm bảo khung tọa độ của dữ liệu lidar (**scan_frame**) được thiết lập và broadcast chính xác so với khung tọa độ gốc của robot (**base_frame**) thông qua **tf**.

- **IMU (Inertial Measurement Unit):** Dữ liệu IMU (message `sensor_msgs/Imu`) có thể được sử dụng để cải thiện ước tính odometry bằng cách kết hợp với dữ liệu encoder bánh xe thông qua một node trung gian sử dụng bộ lọc Kalman hoặc các phương pháp fusion cảm biến khác. Node này sau đó publish odometry đã được cải thiện trên topic `/odom` mà Gmapping đang lắng nghe.

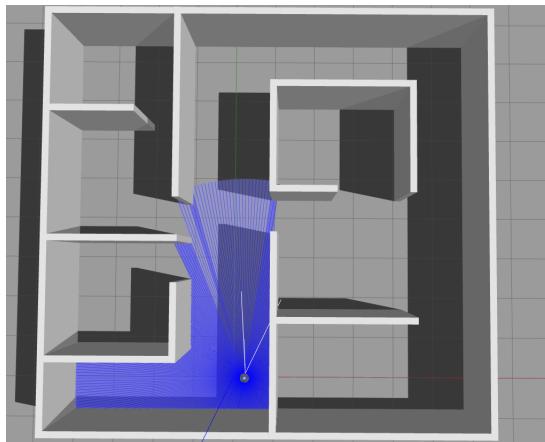
Lưu ý quan trọng: Đảm bảo tính nhất quán về đơn vị đo lường và hệ tọa độ của dữ liệu lidar và IMU. Việc thiết lập đúng các phép biến đổi tọa độ (`transforms`) thông qua `tf` là rất quan trọng.

4.2.3 Chạy Mô Phỏng SLAM Gmapping (Khởi chạy môi trường mô phỏng và các node liên quan)

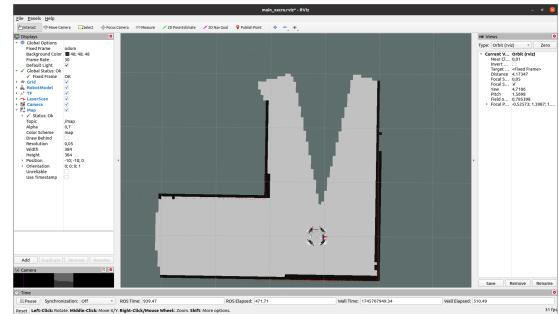
Quy trình chạy mô phỏng SLAM Gmapping thường bao gồm các bước sau:

1. Khởi chạy môi trường mô phỏng
2. Khởi chạy node điều khiển robot
3. Khởi chạy node lidar mô phỏng, publish dữ liệu trên topic `/scan`.
4. Khởi chạy node odometry, publish dữ liệu trên topic `/odom` (kết hợp dữ liệu IMU).
5. Khởi chạy node Gmapping:

```
rosrun gmapping slam_gmapping scan:=/your_laser_topic odom:=/your_odom
```



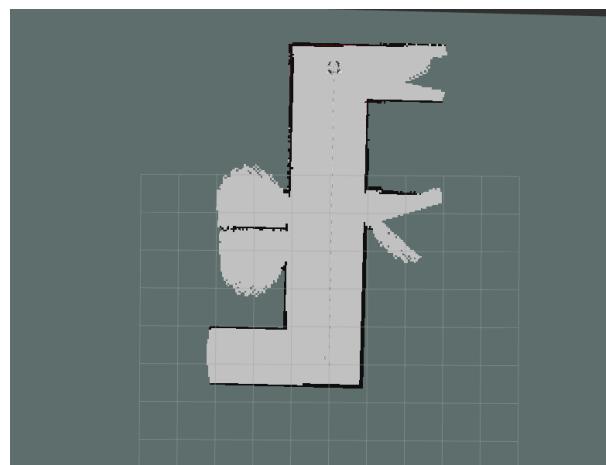
(a) spawn robot trong gazebo.png



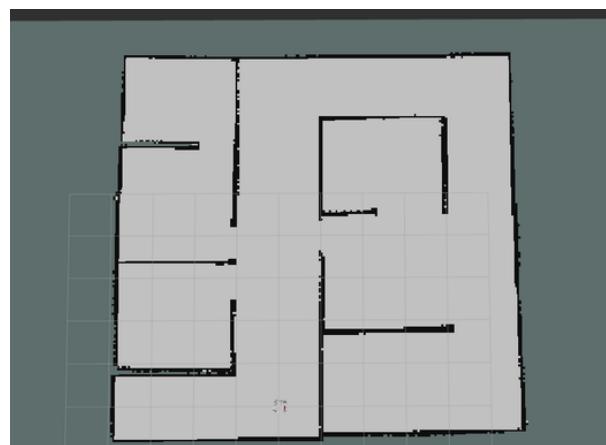
(b) spawn robot trong rviz.png

Hình 4.1: spawn robot

6. Quan sát quá trình xây dựng bản đồ bằng rviz (ví dụ: RViz).



Hình 4.2: Robot di chuyển quét map trong rviz



Hình 4.3: Map được tạo ra khi robot quét hoàn thành

4.2.4 Lưu Trữ Bản Đồ Đã Xây Dựng

Sử dụng service `map_saver` để lưu bản đồ được publish trên topic `/map`:

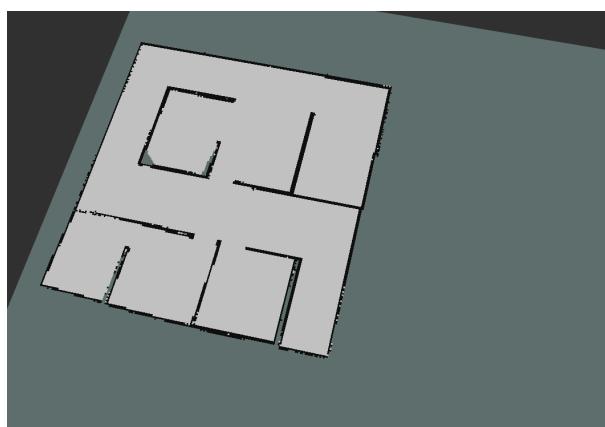
```
rosrun map_server map_saver -f map    ( cd vào cd ~/catkin_ws/src/car_4wd/maps
```

Lệnh này sẽ lưu bản đồ thành hai file: `your_map_name.pgm` (file hình ảnh) và `your_map_name.yaml` (file metadata).

4.3 Đánh Giá Và Phân Tích Kết Quả Mô Phỏng

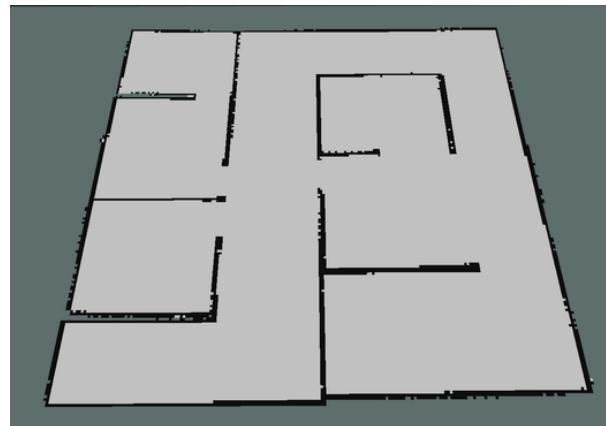
Quan sát video về kết quả mô phỏng slam Gmapping
click để xem video

- **Quan sát trực quan:** Đánh giá độ chính xác và tính nhất quán của bản đồ trong RViz. Ở đây em có thử sử dụng 2 odometry khác nhau để quét map và so sánh
- Odometry automatic.py : odom được tạo ra nhờ sự tính toán của hàm, không thể áp dụng trong thực tế nhưng độ chính xác trong mô phỏng cực cao :



Hình 4.4: Map tạo ra khi sử dụng odometry test

- Odometry test.py : odom được tạo ra do tính toán động học ngược như đã trình bày ở phần cơ sở lý thuyết. Nó được áp dụng nhiều trong thực tế. Tuy nhiên đối với kết quả mô phỏng của bạn em thì Map đã được vẽ ra đúng như thực tế. Tuy nhiên bởi vì odometry không được ổn định trong rviz nên lúc dùng navigation thì robot không thể xác định được vị trí hiện tại của bản thân nó khiến cho robot xoay tại chỗ khi tạo ra điểm goal cho nó.



Hình 4.5: Map được quét khi sử dụng odometry automatic

- So sánh 2 bản đồ thì ta thấy kết quả gần như tương đương.
- Đánh giá chất lượng map: Nhìn sơ qua thì mặc dù map khá rộng nhưng robot đều quét được hết các điểm và không có sự sai lệch nhiều vì odometry chuẩn xác.

CHƯƠNG 5

NAVIGATION CHO ROBOT

5.1 Mục tiêu

Báo cáo này trình bày quy trình xây dựng và triển khai hệ thống điều hướng (navigation) cho robot omni 4 bánh sử dụng ROS (Robot Operating System). Mục tiêu là giúp robot:

- Định vị chính xác vị trí trên bản đồ.
- Di chuyển tự động đến các điểm mục tiêu (waypoints) theo thứ tự.
- Tránh chướng ngại vật dựa trên dữ liệu từ cảm biến LiDAR.
- Sử dụng máy trạng thái (SMACH) để điều khiển hành vi robot.

5.2 Tổng quan về ROS Navigation Stack

ROS Navigation Stack là một bộ công cụ cho phép robot di chuyển tự động trong môi trường. Các thành phần chính bao gồm:

1. **AMCL (Adaptive Monte Carlo Localization)** giúp định vị robot trên bản đồ bằng cách so sánh dữ liệu cảm biến với bản đồ tĩnh.
2. **Global Planner** sẽ lập kế hoạch đường đi dài hạn từ vị trí hiện tại đến mục tiêu.
3. **Local Planner** có tác dụng tạo quỹ đạo ngắn hạn và lệnh vận tốc (`/cmd_vel`) để tránh chướng ngại vật.
4. **Costmap** là Bản đồ chi phí (global và local) biểu thị mức độ nguy hiểm của môi trường.

5. **Move Base** sẽ điều phối các thành phần trên để thực hiện điều hướng.
6. **SMACH** là máy trạng thái để điều khiển hành vi robot (ví dụ: di chuyển qua các điểm mục tiêu hoặc nghỉ).

5.3 Quy trình triển khai hệ thống điều hướng

Robot omni 4 bánh của tôi sẽ thực hiện các bước tuần tự, từ cấu hình hệ thống, định vị, lập kế hoạch, đến điều khiển robot.

5.3.1 Bước 1: Cấu hình Costmap

Costmap là bản đồ chi phí, được chia thành **global costmap** (lập kế hoạch đường đi dài hạn) và **local costmap** (tránh chướng ngại vật ngắn hạn). Các tệp cấu hình liên quan là:

A Costmap Common Parameters (`costmap_common_params.yaml`)

Tệp này định nghĩa các tham số chung cho cả global và local costmap.

- **obstacle_range**: 3.0: Khoảng cách tối đa (3 m) để phát hiện chướng ngại vật từ cảm biến LiDAR (topic `/scan`). Dữ liệu này được dùng để đánh dấu chướng ngại vật trên costmap.
- **raytrace_range**: 3.5: Khoảng cách tối đa (3.5 m) để kiểm tra vùng trống (không có chướng ngại vật). Giúp cập nhật các khu vực an toàn.
- **robot_radius**: 0.09: Robot được mô hình hóa như một hình tròn với bán kính 9 cm để tính toán va chạm.
- **inflation_radius**: 1.0: Bán kính mở rộng (1 m) xung quanh chướng ngại vật, tạo vùng đệm để robot giữ khoảng cách an toàn.
- **cost_scaling_factor**: 3.0: Hệ số điều chỉnh chi phí. Giá trị này làm giảm chi phí nhanh hơn khi robot xa chướng ngại vật, giúp ưu tiên các quỹ đạo an toàn.
- **observation_sources**: `scan`: Dữ liệu từ topic `/scan` (LiDAR), với:
 - **sensor_frame**: `laser_scan`: Khung tham chiếu của cảm biến.

- `data_type`: `LaserScan`: Kiểu dữ liệu là quét laser.
- `marking`: `true`: Đánh dấu chướng ngại vật trên costmap.
- `clearing`: `true`: Xóa chướng ngại vật (vùng trống) trên costmap.

B Global Costmap Parameters (`global_costmap_params.yaml`)

Global costmap được dùng để lập kế hoạch đường đi toàn cục.

- `global_frame`: `map`: Khung tham chiếu là bản đồ tĩnh (`map`), thường được tạo từ SLAM.
- `robot_base_frame`: `base_footprint`: Khung tham chiếu của robot, gắn với tâm robot.
- `update_frequency`: `10.0`: Tần số cập nhật costmap (10 Hz).
- `publish_frequency`: `10.0`: Tần số xuất bản costmap để các thành phần khác sử dụng (10 Hz).
- `transform_tolerance`: `0.5`: Dung sai thời gian (0.5 s) khi chuyển đổi giữa các khung tham chiếu (ví dụ: từ `map` sang `base_footprint`).
- `static_map`: `true`: Sử dụng bản đồ tĩnh, không thay đổi trong quá trình điều hướng.

C Local Costmap Parameters (`local_costmap_params.yaml`)

Local costmap được dùng để tránh chướng ngại vật và tạo quỹ đạo ngắn hạn.

- `global_frame`: `odom`: Khung tham chiếu là `odom`, theo dõi chuyển động cục bộ của robot dựa trên odometry.
- `robot_base_frame`: `base_footprint`: Khung tham chiếu của robot.
- `update_frequency`: `10.0`, `publish_frequency`: `10.0`: Tần số cập nhật và xuất bản (10 Hz).
- `transform_tolerance`: `0.5`: Dung sai thời gian (0.5 s).
- `static_map`: `false`: Không sử dụng bản đồ tĩnh, thay vào đó dùng dữ liệu cảm biến thời gian thực.

- `rolling_window`: `true`: Sử dụng cửa sổ di động, chỉ theo dõi khu vực xung quanh robot.
- `width`: `3`, `height`: `3`: Kích thước cửa sổ là $3\text{ m} \times 3\text{ m}$.
- `resolution`: `0.05`: Độ phân giải 5 cm mỗi ô, đảm bảo chi tiết cao để phát hiện chướng ngại vật nhỏ.

5.3.2 Bước 2: Cấu hình Move Base

Move Base là thành phần trung tâm, điều phối global planner, local planner, và costmap. Tệp `move_base_params.yaml` định nghĩa các tham số điều khiển.

- `shutdown_costmaps`: `false`: Không tắt costmap khi move_base không hoạt động, giúp tiết kiệm thời gian khởi động lại.
- `controller_frequency`: `10.0`: Tần số gửi lệnh vận tốc (`/cmd_vel`) là 10 Hz .
- `planner_patience`: `5.0`: Thời gian chờ (5 s) trước khi global planner thử lập kế hoạch lại nếu đường đi bị chặn.
- `controller_patience`: `15.0`: Thời gian chờ (15 s) trước khi local planner từ bỏ nếu không tạo được quỹ đạo khả thi.
- `conservative_reset_dist`: `3.0`: Khoảng cách (3 m) để xóa chướng ngại vật trong costmap khi reset, tránh việc robot bị kẹt do dữ liệu lỗi.
- `planner_frequency`: `5.0`: Tần số lập kế hoạch toàn cục (5 Hz).
- `oscillation_timeout`: `10.0`: Thời gian (10 s) để phát hiện trạng thái dao động (robot lắc qua lại mà không tiến).
- `oscillation_distance`: `0.2`: Khoảng cách tối thiểu (0.2 m) để coi là robot đang di chuyển, tránh reset dao động không cần thiết.

5.3.3 Bước 3: Cấu hình Local Planner

Hai local planner được cấu hình là **TrajectoryPlannerROS** và **DWAPlannerROS**, nhưng move_base sử dụng DWAPlannerROS làm mặc định.

A TrajectoryPlannerROS (`base_local_planner_params.yaml`)

TrajectoryPlannerROS sử dụng thuật toán *Trajectory Rollout*, tạo nhiều quỹ đạo dựa trên vận tốc và đánh giá chúng.

- **Robot Configuration Parameters:**

- `max_vel_x`: 0.18, `min_vel_x`: 0.08: Vận tốc tuyến tính tối đa và tối thiểu theo trục x (m/s). Robot di chuyển chậm để đảm bảo an toàn.
- `max_vel_theta`: 1.0, `min_vel_theta`: -1.0: Vận tốc góc tối đa và tối thiểu (rad/s, ~ 57 độ/s).
- `min_in_place_vel_theta`: 1.0: Vận tốc góc tối thiểu khi xoay tại chỗ (rad/s).
- `acc_lim_x`: 1.0, `acc_lim_y`: 0.0, `acc_lim_theta`: 0.6: Gia tốc tuyến tính (x), tuyến tính (y), và góc. `acc_lim_y`: 0.0 vì robot không di chuyển ngang.

- **Goal Tolerance Parameters:**

- `xy_goal_tolerance`: 0.10: Robot được coi là đạt mục tiêu nếu nằm trong vòng 10 cm từ điểm mục tiêu.
- `yaw_goal_tolerance`: 0.05: Dung sai góc là 0.05 rad (~ 2.86 độ).

- **Differential-drive Configuration:**

- `holonomic_robot`: false: Robot được cấu hình như dẫn động vi sai, không tận dụng khả năng toàn hướng của robot omni.

- **Forward Simulation Parameters:**

- `sim_time`: 0.8: Mô phỏng quỹ đạo trong 0.8 s.
- `vx_samples`: 18: Thử 18 giá trị vận tốc tuyến tính từ `min_vel_x` đến `max_vel_x`.
- `vtheta_samples`: 20: Thử 20 giá trị vận tốc góc từ `min_vel_theta` đến `max_vel_theta`.
- `sim_granularity`: 0.05: Các điểm trên quỹ đạo cách nhau 5 cm, đảm bảo độ chính xác cao.

Cơ chế hoạt động:

- Tạo nhiều quỹ đạo bằng cách kết hợp các giá trị vận tốc tuyến tính và góc.
- Dánh giá quỹ đạo dựa trên:
 - Khoảng cách đến chướng ngại vật (tránh va chạm).
 - Khoảng cách đến mục tiêu cục bộ.
 - Độ mượt mà của quỹ đạo.
- Chọn quỹ đạo tốt nhất và xuất lệnh vận tốc tương ứng.

B DWAPlannerROS (`dwa_local_planner_params.yaml`)

DWAPlannerROS sử dụng thuật toán *Dynamic Window Approach (DWA)*, tối ưu hơn trong môi trường động.

- **Robot Configuration Parameters:**

- `max_vel_x`: 0.5, `min_vel_x`: -0.5: Vận tốc tuyến tính tối đa và tối thiểu, cho phép robot lùi.
- `max_vel_y`: 0.0, `min_vel_y`: 0.0: Không di chuyển theo trục y, do cấu hình non-holonomic.
- `max_vel_trans`: 0.5, `min_vel_trans`: 0.13: Vận tốc chuyển động tổng tối đa và tối thiểu.
- `max_vel_theta`: 1.0, `min_vel_theta`: 0.2: Vận tốc góc tối đa và tối thiểu.
- `acc_lim_x`: 2.5, `acc_lim_y`: 0.0, `acc_lim_theta`: 3.2: Gia tốc tối đa, với `acc_lim_y`: 0.0 vì không di chuyển ngang.

- **Goal Tolerance Parameters:**

- `xy_goal_tolerance`: 0.05: Dung sai vị trí 5 cm, chính xác hơn so với TrajectoryPlannerROS.
- `yaw_goal_tolerance`: 0.17: Dung sai góc 0.17 rad (~9.74 độ).
- `latch_xy_goal_tolerance`: `false`: Không giữ dung sai vị trí sau khi đạt mục tiêu.

- **Forward Simulation Parameters:**

- `sim_time`: 1.0: Mô phỏng quỹ đạo trong 1 s.

- `vx_samples`: 20, `vy_samples`: 0, `vth_samples`: 40: Số mẫu vận tốc tuyến tính (x), tuyến tính (y), và góc.
- `controller_frequency`: 10.0: Tần số điều khiển 10 Hz.

- **Trajectory Scoring Parameters:**

- `path_distance_bias`: 64.0: Trọng số ưu tiên quỹ đạo bám sát đường đi toàn cục.
- `goal_distance_bias`: 24.0: Trọng số ưu tiên quỹ đạo gần mục tiêu cục bộ.
- `occdist_scale`: 0.02: Trọng số ưu tiên tránh chướng ngại vật (giá trị thấp, ít ưu tiên).
- `forward_point_distance`: 0.325: Khoảng cách (0.325 m) đến điểm tham chiếu phía trước để đánh giá quỹ đạo.
- `stop_time_buffer`: 0.2: Thời gian đệm (0.2 s) để dừng trước khi va chạm.
- `scaling_speed`: 0.25, `max_scaling_factor`: 0.2: Điều chỉnh vận tốc dựa trên khoảng cách đến chướng ngại vật.

- **Oscillation Prevention:**

- `oscillation_reset_dist`: 0.05: Khoảng cách (5 cm) để reset trạng thái dao động.

- **Debugging:**

- `publish_traj_pc`: true, `publish_cost_grid_pc`: true: Xuất bản quỹ đạo và lưới chi phí để trực quan hóa trong RViz.

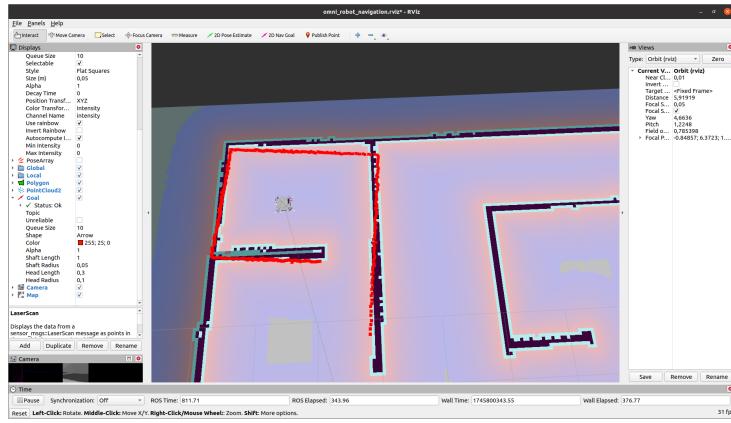
Cơ chế hoạt động:

- DWA giới hạn không gian vận tốc (dynamic window) dựa trên trạng thái hiện tại và gia tốc của robot.
- Tạo và đánh giá quỹ đạo dựa trên các trọng số (`path_distance_bias`, `goal_distance_bias`, `occdist_scale`).
- Chọn quỹ đạo tối ưu và gửi lệnh vận tốc qua `/cmd_vel`.

5.3.4 Bước 4: Định vị Robot với AMCL

AMCL sử dụng bộ lọc hạt (particle filter) để ước lượng vị trí robot trên bản đồ. Với cấu hình như sau:

- `min_particles`: 500, `max_particles`: 3000: Số hạt tối thiểu và tối đa, đại diện cho các giả thuyết về vị trí robot.
- `laser_max_range`: 3.5: Khoảng cách tối đa của LiDAR (3.5 m).
- `odom_model_type`: `diff`: Mô hình odometry cho robot dẫn động vi sai.
- `transform_tolerance`: 0.5: Dung sai thời gian khi chuyển đổi khung.



Hình 5.1: Robot đang xác định vị trí

Cơ chế hoạt động:

- AMCL lấy dữ liệu từ topic `/scan` (LiDAR) và `/odom` (odometry).
- So sánh dữ liệu cảm biến với bản đồ tinh để cập nhật trọng số các hạt.
- Xuất bản vị trí ước lượng (`/amcl_pose`) và chuyển đổi khung (`/tf`).

5.3.5 Bước 5: Gửi và Điều khiển Mục tiêu

Em đã tạo ra hai tệp mã nguồn chính được sử dụng để gửi mục tiêu và điều khiển robot là set goal action và navigation

A set_goal_action.py

Tệp này sử dụng `actionlib` để gửi mục tiêu điều hướng đến `move_base`.

- **Lớp GoalAction:**

- `__init__`: Khởi tạo client actionlib với server `move_base` và kiểu `MoveBaseAction`.
- `createGoal(x, y, th_deg)`: Tạo `PoseStamped` với:
 - * `header.frame_id: map`: Mục tiêu trong khung `map`.
 - * `pose.position.x, pose.position.y`: Tọa độ (`x, y`).
 - * `pose.orientation`: Góc `th_deg` (độ) được chuyển thành quaternion bằng `quaternion_from_euler`.
- `createGoalAction(x, y, th_deg)`: Tạo `MoveBaseGoal` chứa `PoseStamped`.
- `moveToGoal(x, y, th_deg, feedbackCallback)`: Gửi mục tiêu đến `move_base`, chờ server xử lý, và hỗ trợ callback để theo dõi tiến trình.
- `getStatus`: Lấy trạng thái mục tiêu (`GoalStatus.ACTIVE`, `GoalStatus.SUCCEEDED`, v.v.).
- `cancel`: Hủy mục tiêu.

- **Main:**

- Khởi tạo node `set_goal_node`.
- Tạo publisher dự phòng cho topic `/move_base_simple/goal`.
- In trạng thái mục tiêu mỗi 0.1 s.

Cơ chế hoạt động:

- Gửi mục tiêu qua giao thức actionlib, cho phép theo dõi trạng thái và hủy mục tiêu nếu cần.
- `move_base` nhận mục tiêu, lập kế hoạch, và điều khiển robot đến vị trí yêu cầu.

B `navigation_sm.py`

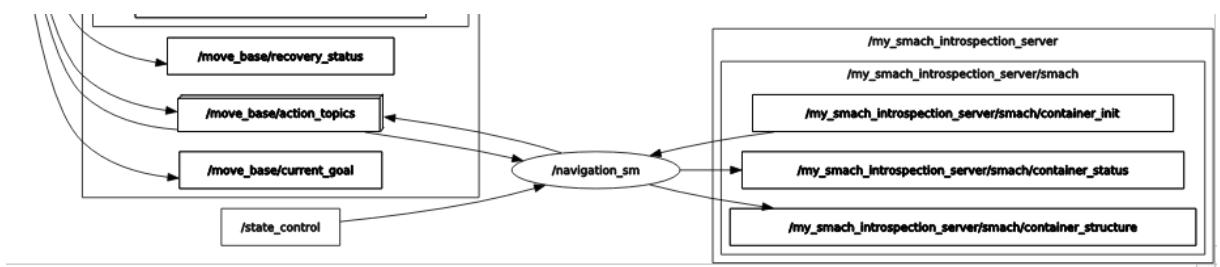
Tệp này sử dụng *SMACH* để điều khiển robot di chuyển qua một danh sách các điểm mục tiêu (waypoints).

- **Lớp State:** Định nghĩa hai trạng thái:

- `IDLE`: Robot nghỉ, chờ lệnh.
- `WAYPOINT_LOOP`: Robot di chuyển qua các điểm mục tiêu.

- **Lớp SharedData:** Lưu trữ trạng thái hiện tại (state).
- **Lớp IdleState:**
 - `__init__`: Khởi tạo với outcome `to_waypoint_loop`.
 - `execute`: Chờ cho đến khi `sharedData.state` chuyển sang `WAYPOINT_LOOP`, sau đó chuyển sang trạng thái `WaypointLoopState`.
- **Lớp WaypointLoopState:**
 - `__init__`: Khởi tạo với danh sách waypoints:

```
wp = [
    [2, 3, 0], [5, 0, 90], [7, 3, -30], [5, 3, 150],
    [5, 3, 45], [7.5, -3, 90], [5.5, -2.5, -90], [0, -2.5, 90]
]
```
 - `execute`:
 - * Nếu trạng thái là `IDLE`, hủy mục tiêu và chuyển về `IdleState`.
 - * Nếu mục tiêu hoàn thành (`GoalStatus.SUCCEEDED`), tăng chỉ số waypoint (`iWp`) và gửi mục tiêu mới.
 - * Nếu mục tiêu đang thực hiện (`GoalStatus.ACTIVE`), chờ 0.1 s và kiểm tra lại.
- **Hàm stateControlCallback:** Cập nhật `sharedData.state` dựa trên tin nhắn từ topic `/state_control` ("idle" hoặc "waypoint").
- **Main:**
 - Khởi tạo node `navigation_sm`.
 - Tạo state machine với hai trạng thái: `IDLE` và `WAYPOINT_LOOP`.
 - Khởi động server kiểm tra (introspection server) để debug SMACH.
 - Chạy state machine.



Hình 5.2: rqt graph cho navigation

Cơ chế hoạt động:

- Robot bắt đầu ở trạng thái IDLE, chờ lệnh từ topic /state_control.
- Khi nhận lệnh "waypoint", chuyển sang WAYPOINT_LOOP, gửi lần lượt các mục tiêu trong danh sách waypoints.
- Khi nhận lệnh "idle" hoặc hoàn thành tất cả waypoints, quay về IDLE.

5.3.6 Bước 6: Kiểm tra và Đánh giá

- **RViz:** Sử dụng RViz để trực quan hóa (tạo ra 1 file rviz riêng với khi dùng gmapping)
 - Global costmap và local costmap.
 - Đường đi toàn cục từ global planner.
 - Quỹ đạo từ local planner (xuất bản qua publish_traj_pc).
 - Vị trí robot (/amcl_pose) và dữ liệu LiDAR (/scan).
- **Log:** Theo dõi trạng thái mục tiêu từ set_goal_action.py và navigation_sm.py để kiểm tra lỗi.
- **Video kết quả mô phỏng:**
click để xem video

Kết quả mô phỏng thuật toán Navigation với 1 vài điểm khó ở trong map
Đánh giá chất lượng của thuật toán navigation :

- Kiểm tra robot di chuyển đúng đến các waypoints.
- Đảm bảo robot tránh được chướng ngại vật trong local costmap.
- Xác minh AMCL định vị chính xác bằng cách so sánh vị trí robot với bản đồ.

5.4 Phát triển trong tương lai

1. Tận dụng khả năng toàn hướng:

- Cập nhật holonomic_robot: true trong base_local_planner_params.yaml.
- Điều chỉnh max_vel_y, min_vel_y, acc_lim_y trong dwa_local_planner_params.yaml để hỗ trợ di chuyển ngang.

2. Tối ưu tham số:

- Giảm `vx_samples` và `vth_samples` trong `DWAPlannerROS` để tăng tốc độ tính toán.
- Tinh chỉnh `path_distance_bias` và `goal_distance_bias` để cân bằng giữa bám đường đi và đạt mục tiêu.

3. Cải tiến điều khiển:

- Thêm cơ chế thử lại mục tiêu nếu thất bại trong `set_goal_action.py`.

4. Kiểm tra thực tế:

- Thủ nghiệm robot trong môi trường có chướng ngại vật động để đánh giá hiệu quả của `DWAPlannerROS`.
- Sử dụng công cụ như `rosbag` để ghi lại dữ liệu và phân tích lỗi.

CHƯƠNG 6

KẾT QUẢ VÀ ĐÁNH GIÁ

Những công việc đã thực hiện trong đề tài

Đã thiết kế, mô phỏng và điều khiển thành công robot di động omni, bao gồm các chuyển động đa hướng cơ bản (tiến, lùi, trái, phải, xoay) dựa trên động học tính toán một cách tương đối chính xác và odometry cơ bản. Bước đầu triển khai gmapping và navigation.

Hướng phát triển

Tiếp tục hoàn thiện hệ thống điều khiển và nhận thức: tích hợp thêm cảm biến, nâng cao độ chính xác chuyển động đa hướng và khả năng định vị (odom), cải thiện hiệu suất gmapping và navigation, đồng thời tăng cường độ bền và khả năng vận hành trong các môi trường khác nhau.

CHƯƠNG 7

CÂU HỎI VÀ TRẢ LỜI CÂU HỎI

Trả lời câu hỏi từ Nhóm 8

Câu 1: Trong quá trình resampling, nếu số lượng effective particles (neff) quá thấp liên tục, bạn sẽ làm thế nào để điều chỉnh tham số hoặc thiết kế hệ thống nhằm tránh sự "degeneracy" (sự suy giảm hạt)?

Trả lời: Nguyên nhân chính:

Neff thấp → Đa số hạt có trọng số không đáng kể, chỉ một vài hạt chi phối
→ Mất đa dạng (sample impoverishment).

Thường xảy ra do:

- Resampling quá thường xuyên.
- Mô hình cảm biến (lidar) hoặc odometry kém chính xác.
- Số hạt ban đầu quá ít.

→ cách giải quyết:

- Điều Chỉnh Ngưỡng Resampling (*resampleThreshold*) để tránh mất hạt giống tốt.
- Tăng Số Lượng Hạt (*numParticles*) để duy trì đa dạng hạt.
- Thêm Nhiều Sau Resampling để khôi phục đa dạng hạt.
- Tối ưu odom và sensor.
- Adaptive Resample
- Tối ưu tham số Robot omni: *linearUpdate* (di chuyển linh hoạt) và *angularUpdate* (xoay tại chỗ nhanh)

Câu 2: Hãy giải thích cách bạn lựa chọn hoặc tối ưu các tham số như *resampleThreshold*, *linearUpdate*, và *angularUpdate* trong bối cảnh cụ thể của robot omni.

Trả lời: Các tham số *linearUpdate*, *angularUpdate*, và *resampleThreshold* được đặt mặc định như sau:

```
linearUpdate: 1.0      # (mét)
angularUpdate: 0.2     # (radian)
resampleThreshold: 0.5 # (tỷ lệ Neff)
```

Với các tham số mặc định này, chúng được điều chỉnh phù hợp cho các xe di chuyển kiểu differential drive. Trong trường hợp robot omni của nhóm em, khi điều khiển từ xa (teleop) và xuất bản vận tốc thẳng, robot di chuyển tương tự như robot differential drive theo hướng đó. Do đó, nhóm em quyết định giữ nguyên các tham số mặc định theo hướng dẫn.

Đặt câu hỏi cho nhóm 7

Câu 1: Ràng buộc từ scanmatching là là quá trình so khớp dữ liệu từ lượt quét laser hiện tại với các lượt quét trước đó hoặc một phần bản đồ đã xây dựng. và (Loop Closure Detection): là so sánh lượt quét hiện tại với các lượt quét trong quá khứ. có khác gì nhau (trước đó và quá khứ) ?

Câu 2: Tối ưu hóa ở đây là min edg nghĩa là sao ?

Tài liệu tham khảo

- [1] *gazebo_msgs/srv/GetModelState*, ROS 2 Documentation, https://docs.ros.org/en/iron/p/gazebo_msgs/interfaces/srv/GetModelState.html, accessed April 28, 2025.
- [2] *Mobile Robot Kinematics*, ROS 2 Control Documentation, https://control.ros.org/rolling/doc/ros2_controllers/doc/mobile_robot_kinematics.html, accessed April 28, 2025.
- [3] A. Phunopas and S. Inoue, *Motion Improvement of Four-Wheeled Omnidirectional Mobile Robots for Indoor Terrain*, [Journal/Conference name if available], [Year if available]. (Note: Please replace the bracketed information with the actual details of the publication if you have them.)
- [4] D. U. Rijalusalam and I. Iswanto, *Implementation Kinematics Modeling and Odometry of Four Omni Wheel Mobile Robot on The Trajectory Planning and Motion Control Based Microcontroller*, [Journal/Conference name if available], [Year if available]. (Note: Please replace the bracketed information with the actual details of the publication if you have them.)
- [5] M. S. S. Htay, K. W. Phy, S. Yadnar, and W. Y. Win, *Kinematics and Control A Three-wheeled Mobile Robot with Omni-directional Wheels*, [Journal/Conference name if available], [Year if available]. (Note: Please replace the bracketed information with the actual details of the publication if you have them.)
- [6] *gmapping*, ROS Wiki, <http://wiki.ros.org/gmapping>, accessed April 28, 2025.