

ĐẠI HỌC QUỐC GIA HÀ NỘI
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ

Khoa: Điện tử viễn thông



BÁO CÁO GIỮA KỲ: LẬP TRÌNH ROBOT VỚI ROS

Họ và tên:
Nguyễn Văn Bằng - 22027505

Hà Nội – 2024

Tóm tắt (Abstract - from 5 to 10 lines)

Dự án này tập trung vào việc thiết kế và mô phỏng một robot di chuyển omnidirectional sử dụng cơ cấu mecanum 3 bánh trong môi trường ROS (Robot Operating System). Robot được trang bị ba cảm biến chính: LiDAR để quét và lập bản đồ môi trường, IMU để đo hướng và tốc độ, cùng Encoder để theo dõi chính xác chuyển động của bánh xe. Mô hình 3D của robot được xây dựng chi tiết, sau đó chuyển đổi thành định dạng URDF/Xacro để hiển thị và kiểm tra trong RViz. Trong Gazebo, robot được mô phỏng với khả năng di chuyển linh hoạt theo mọi hướng nhờ cơ chế mecanum, đồng thời tích hợp dữ liệu từ các cảm biến để đảm bảo hoạt động ổn định. Báo cáo trình bày quá trình thiết kế, tính toán, tích hợp cảm biến và xử lý lỗi, đáp ứng đầy đủ yêu cầu của bài kiểm tra giữa kỳ.

Từ khóa (Keywords)

Robot Omnidirectional

Mecanum Wheels

ROS (Robot Operating System)

LiDAR

IMU (Inertial Measurement Unit)

Encoder

URDF/Xacro

Gazebo Simulation

RViz Visualization

Sensor Integration

Document History

	Họ và tên (Full name)	Mã SV (ID)	Đóng góp (Contribution)
Thành viên 1 (Member 1)	Nguyễn Văn Bằng	22027505	Toàn bộ dự án
Tên/Địa chỉ Repo trên Github	https://github.com/EBang2k4/omni3_pkg		

Mục lục (Table of Contents)

Document History

Mục lục (Table of Contents)

1. Giới thiệu (Introduction)

- 1.1. Mục tiêu dự án
- 1.2. Yêu cầu đề bài
- 1.3. Tổng quan về robot omni 3 bánh

2. Thiết kế mô hình robot

- 2.1. Mô tả tổng quan về robot
- 2.2. Thiết kế cơ khí (mô hình 3D)
 - 2.2.1. Chassis (base_link)
 - 2.2.2. Chiều cao tổng thể
 - 2.2.3. Bánh xe (mecanum)
 - 2.2.4. Cơ cấu tay máy

3. Cấu trúc hình học và mô hình toán của robot

- 3.1. Cấu trúc hình học
- 3.2. Phương trình động học
- 3.3. Phương trình động lực học

4. Mô hình hóa trong ROS (URDF/Xacro)

4.1. Hệ trục tọa độ trong SolidWorks

4.2. URDF

 4.2.1. Cấu trúc thư mục

 4.2.2. Mô tả URDF file

 4.2.2.1. Mô tả chung về file

 4.2.2.2. Plugin cảm biến

 - Lidar

 - IMU

5. Điều khiển trong ROS

5.1. ROS Control

5.2. Sử dụng Plugin `gazebo_ros_control`

5.3. Định nghĩa Transmission trong URDF

5.4. Cấu hình file YAML

5.5. File launch

5.6. Scripts Python điều khiển bánh xe

5.7. Scripts Python điều khiển tay máy

5.8. Hiển thị encoder

6. Mô phỏng trong Gazebo và RViz

6.1. Hiển thị LIDAR

6.2. Hiển thị IMU

6.3. Hiển thị Encoder

6.4. Hiển thị điều khiển bánh xe

6.5. Hiển thị điều khiển tay máy

7. Motion Planning tay máy bằng MoveIt

8. Kết luận

1. Giới thiệu

1.1. Mục tiêu dự án

Mục tiêu chính của dự án này là thiết kế và mô phỏng một robot di chuyển omni-directional sử dụng ba bánh mecanum trong môi trường ROS (Robot Operating System). Robot được trang bị các cảm biến **LiDAR**, **IMU** và **Encoder** để đạt được khả năng điều hướng chính xác và nhận thức môi trường. Thông qua dự án, tôi hướng đến việc chứng minh khả năng di chuyển linh hoạt của robot theo mọi hướng, đồng thời tích hợp và xử lý dữ liệu từ các cảm biến để đảm bảo robot hoạt động hiệu quả trong môi trường mô phỏng.

1.2. Yêu cầu đề bài

Theo yêu cầu của đề bài, robot phải đáp ứng các tiêu chí sau:

- Sử dụng cơ cấu ba bánh mecanum để hỗ trợ di chuyển omni-directional.
- Tích hợp ba loại cảm biến: LiDAR để quét và lập bản đồ môi trường, IMU để đo hướng và gia tốc, và encoder để theo dõi chuyển động của bánh xe.
- Được mô hình hóa đầy đủ bằng URDF/Xacro để hiển thị và mô phỏng trong ROS.
- Hoạt động trong môi trường ROS, với các luồng dữ liệu cảm biến hoạt động ổn định và khả năng điều khiển thông qua lệnh vận tốc.

1.3. Tổng quan về robot omni 3 bánh

Robot được thiết kế với khung chassis hình tròn, cho phép bố trí ba bánh xe mecanum đều nhau ở ba vị trí, mỗi bánh cách nhau 120 độ, đảm bảo khả năng di chuyển đa hướng (omni-directional) linh hoạt. Cấu trúc của robot bao gồm ba tầng, mỗi tầng được tối ưu hóa cho các chức năng riêng biệt:

- **Tầng 1 (dưới cùng):** Gồm IMU (Inertial Measurement Unit) để đo gia tốc và định hướng của robot, cùng với ba bánh xe mecanum tích hợp encoder, hỗ trợ theo dõi chính xác tốc độ và vị trí của từng bánh.
- **Tầng 2:** Được trang bị cảm biến LiDAR, đặt tại vị trí trung tâm, cho phép quét môi trường xung quanh 360 độ, phục vụ cho việc lập bản đồ và phát hiện chướng ngại vật.

- **Tầng 3 (trên cùng):** Gắn một tay máy với hai khớp rotation, mang lại khả năng thực hiện các thao tác linh hoạt như nhặt hoặc đặt vật thể.

Thiết kế ba tầng này không chỉ đảm bảo tính ổn định và khả năng di chuyển vượt trội mà còn hỗ trợ tích hợp hiệu quả các thành phần như cảm biến và tay máy, đáp ứng tốt nhu cầu điều hướng và tương tác trong các ứng dụng robotics phức tạp.

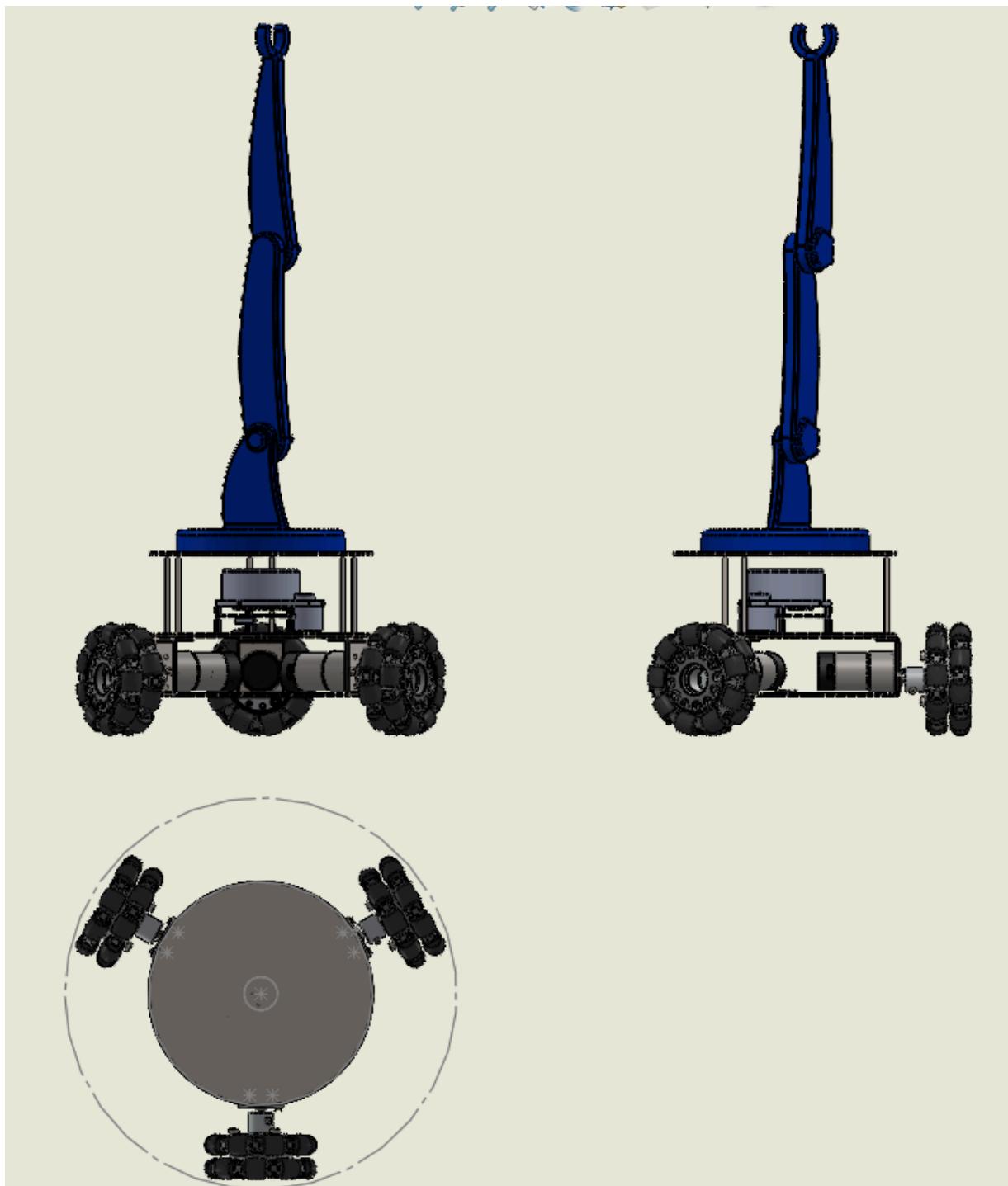
2. Thiết kế mô hình robot

2.1. Mô tả tổng quan

Robot được thiết kế với cấu trúc tích hợp giữa phần thân (base) di chuyển kiểu car-like và tay máy hai khớp, nhằm đáp ứng khả năng di chuyển linh hoạt và thực hiện các tác vụ tương tác với môi trường. Cấu trúc robot được chia thành các phần chính như sau:

- **Phần thân (base):** Robot có dạng chassis hình tròn, sử dụng ba bánh xe mecanum để di chuyển theo kiểu omni-directional, cho phép di chuyển ngang, dọc hoặc xoay tại chỗ mà không cần thay đổi hướng của thân. Thiết kế này khác biệt so với kiểu car-like truyền thống (di chuyển giống xe hơi với hai bánh dẫn động và bánh phụ), nhưng mang lại sự linh hoạt vượt trội trong không gian hẹp.
- **Tay máy:** Được gắn ở phần trên cùng của robot, tay máy gồm hai khớp rotation, cho phép thực hiện các thao tác nhu nhặt, đặt hoặc di chuyển vật thể. Hai khớp này cung cấp khả năng xoay linh hoạt theo các trục khác nhau, tối ưu hóa phạm vi hoạt động.
- **Vị trí gắn cảm biến:**
 - Cảm biến IMU (Inertial Measurement Unit) được đặt ở phần thân dưới cùng, gần bánh xe, để thu thập dữ liệu về tốc độ và định hướng của robot.
 - Cảm biến LiDAR được bố trí ở vị trí trung tâm phía trên chassis, cho phép quét môi trường xung quanh 360 độ để phát hiện chướng ngại vật và lập bản đồ.

Cấu trúc này được sắp xếp khoa học để đảm bảo các thành phần hoạt động hài hòa, không gây cản trở lẫn nhau trong quá trình vận hành.



Hình 1. 2D Drawing của robot 3 bánh

2.2. Thiết kế cơ khí

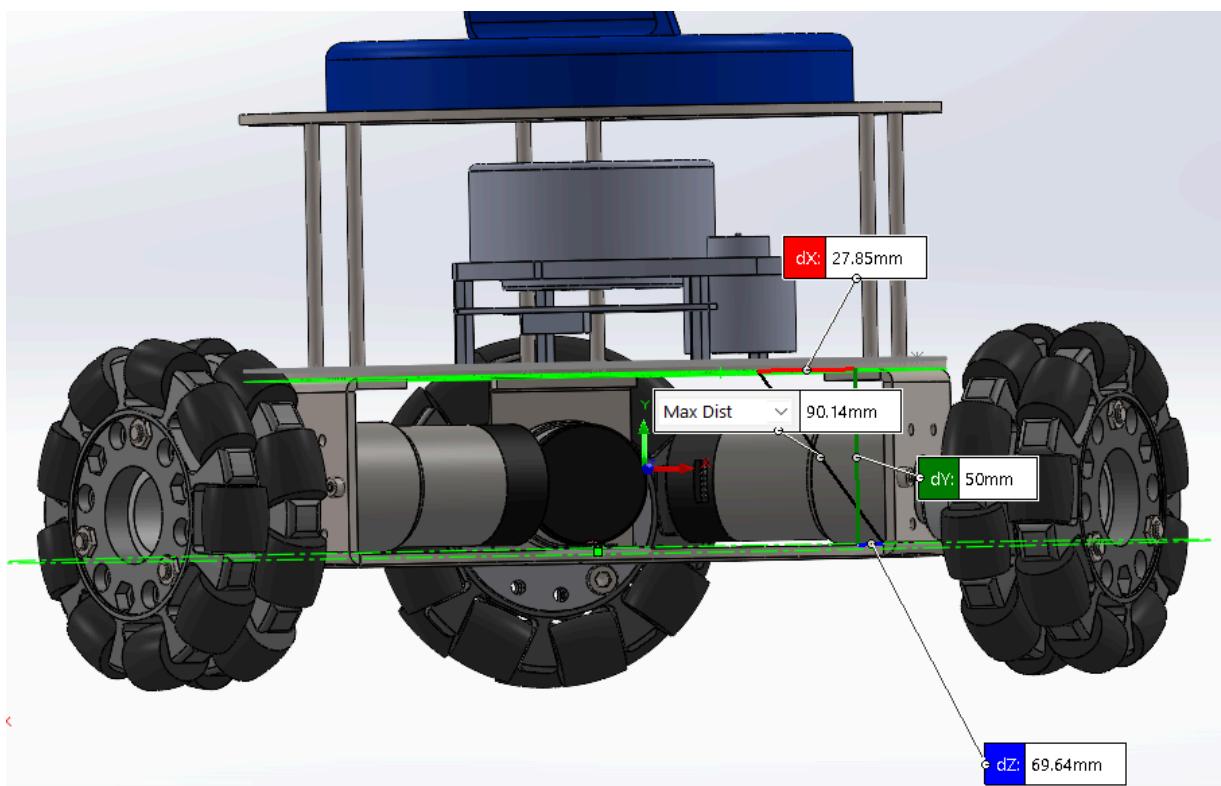
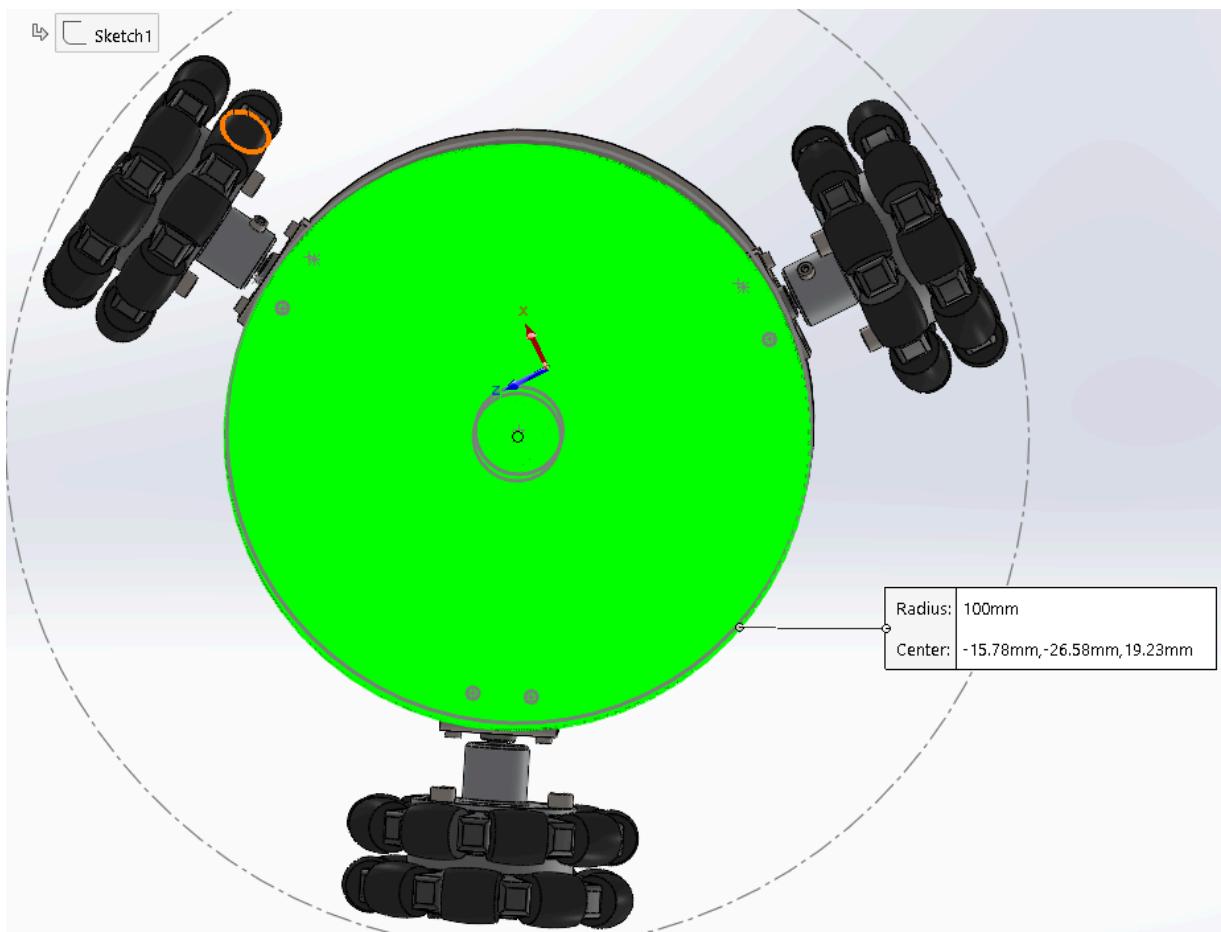
Mô hình 3D của robot được thiết kế bằng phần mềm **SolidWorks**, một công cụ phổ biến cho phép tạo ra các bản vẽ kỹ thuật chính xác và tương thích với các hệ thống mô phỏng như ROS (Robot Operating System). Dưới đây là mô tả chi tiết về các thông số kỹ thuật dựa trên bản vẽ SolidWork và URDF File:

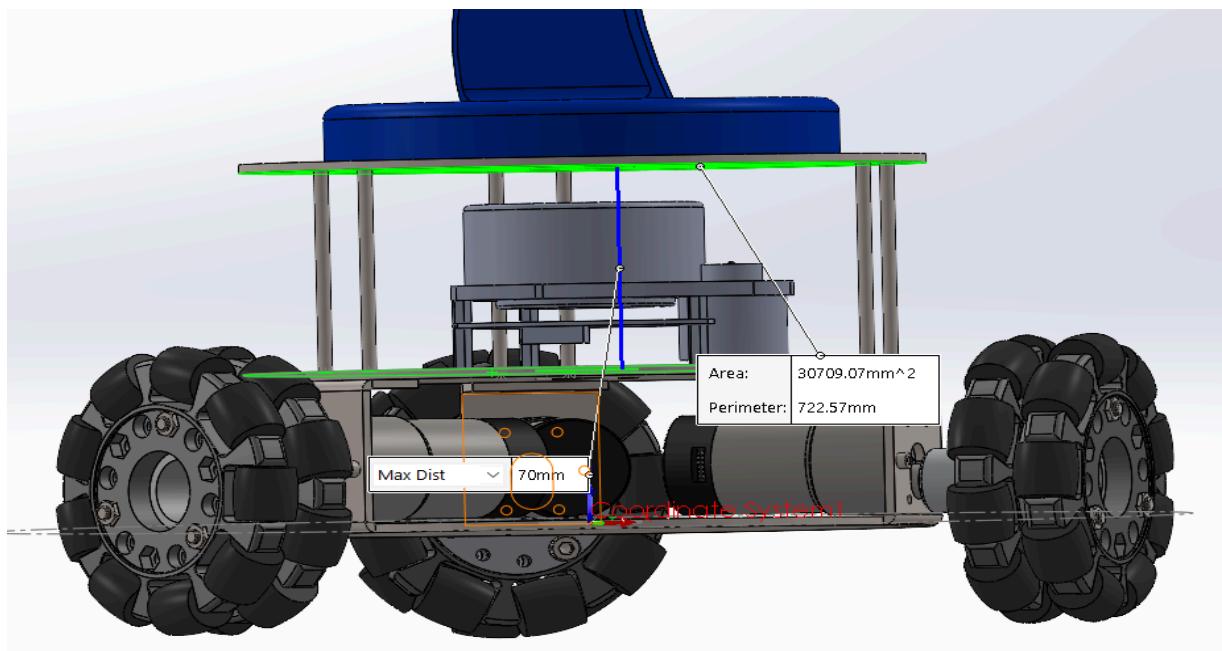
2.2.1. Chassis (base_link):

- Đường kính: **20 cm**.
- Độ dày đế: **3 mm**.
- Chiều cao tầng 1 (bao gồm bánh xe và IMU): **5 cm**.
- Chiều cao tầng 2 (bao gồm LiDAR): **7 cm**.

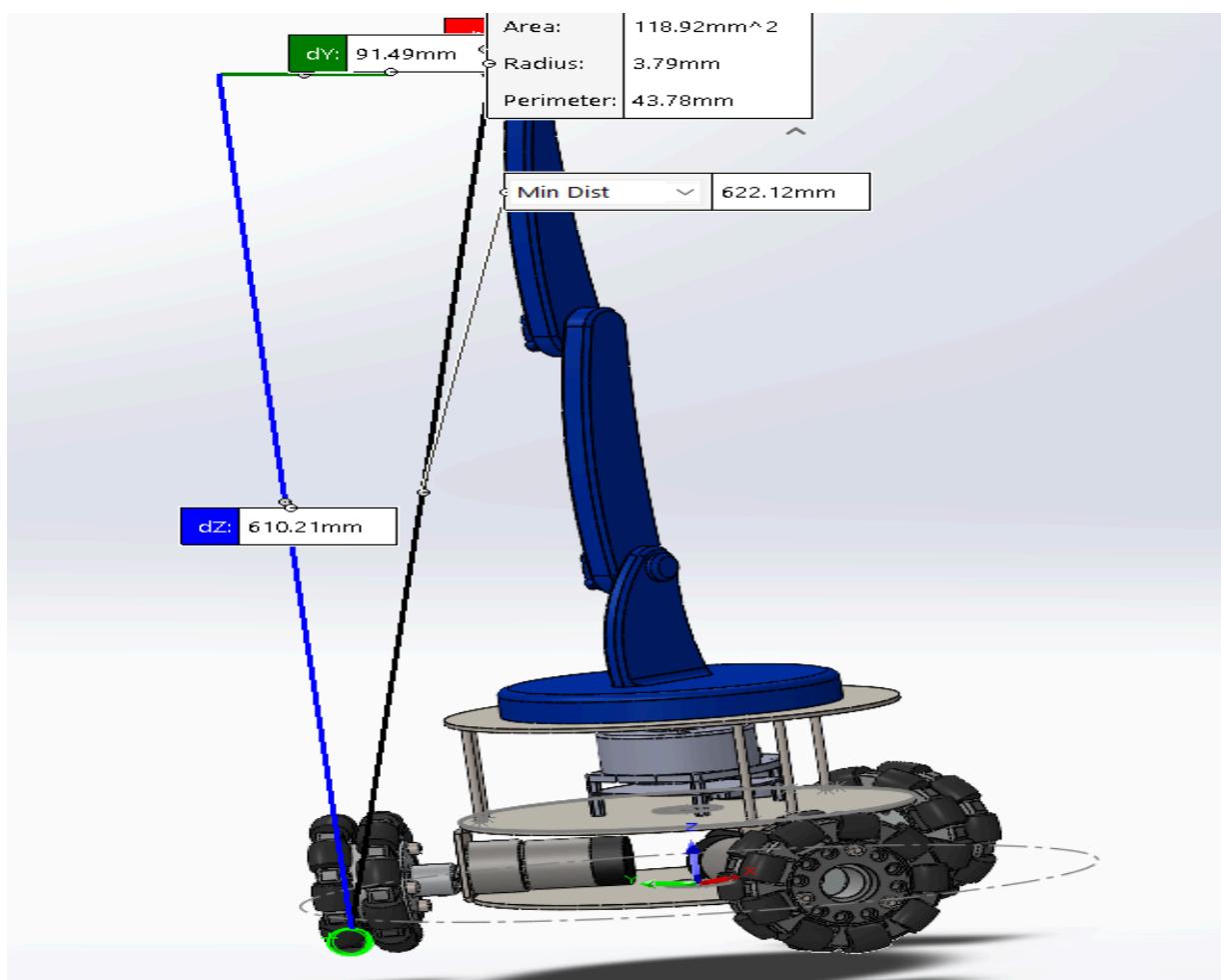
2.2.2. Chiều cao tổng thể:

- Từ mặt đất đến đỉnh tay máy: **63 cm (ở trạng thái duỗi thẳng đứng)**.
- Tầng 1 (base + bánh xe + IMU): **2 cm**.
- Tầng 2 (LiDAR): Đỉnh LiDAR cách mặt đất khoảng **7 cm**.
- Tầng 3 (tay máy): Thêm khoảng **8 cm**.





Hình 2. Khoảng cách của robot



Hình 3. Chiều cao của robot

2.2.3. Bánh xe (mecanum):

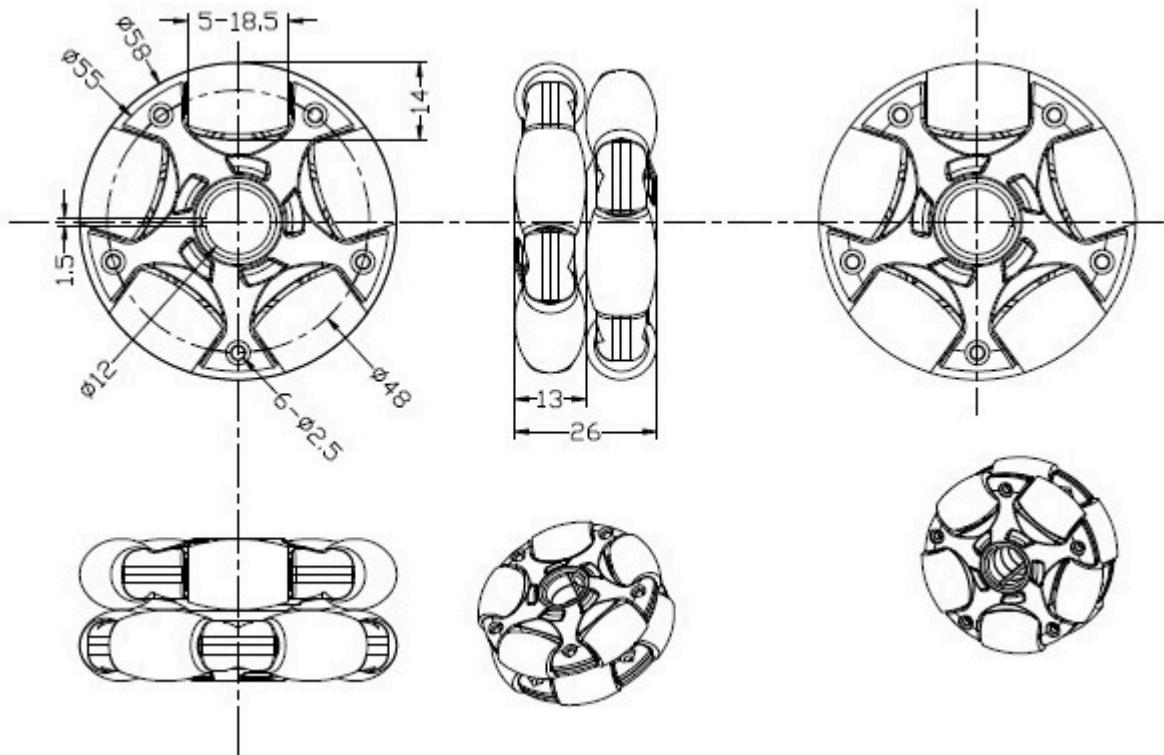
Trên cơ sở làm xe 3 bánh omni và để phù hợp với kích thước xe đã lựa chọn , chúng em chọn loại bánh Omni 58mm nhựa.



Hình 4. Bánh Mecanum

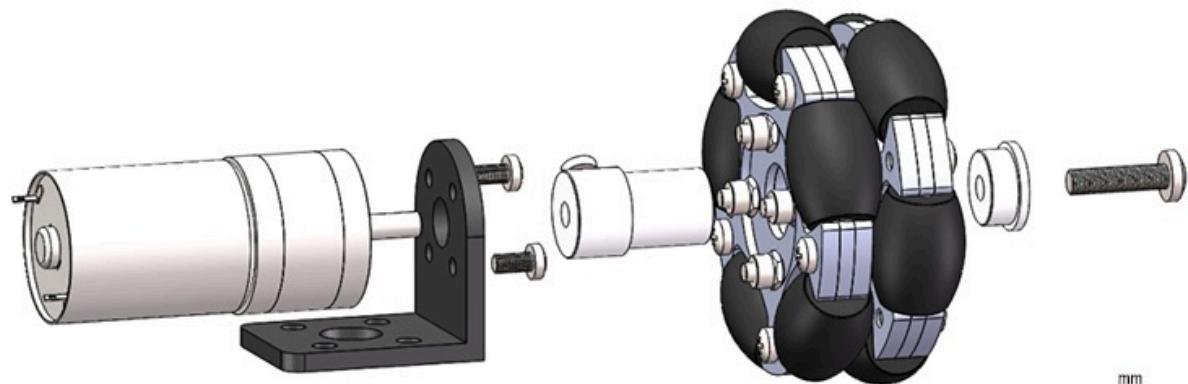
Một số thông số của bánh :

- Đường kính bánh xe: 58mm
- Đường kính trục: 13mm
- Chất liệu: Nhựa ABS + Cao su
- Trọng tải: 3Kg
- Trọng lượng: 60g
- Bánh đôi với mỗi nửa có 5 con lăn



Hình 5. Bản vẽ kỹ thuật bánh mecanum

Khớp nối:



Hình 6. Khớp nối bánh xe

2.2.4. Cơ cấu tay máy:

- Tay máy gồm hai khớp rotation:
 - **Khớp 1:** Quay quanh trục với phạm vi 180 độ, cho phép tay quét ngang.

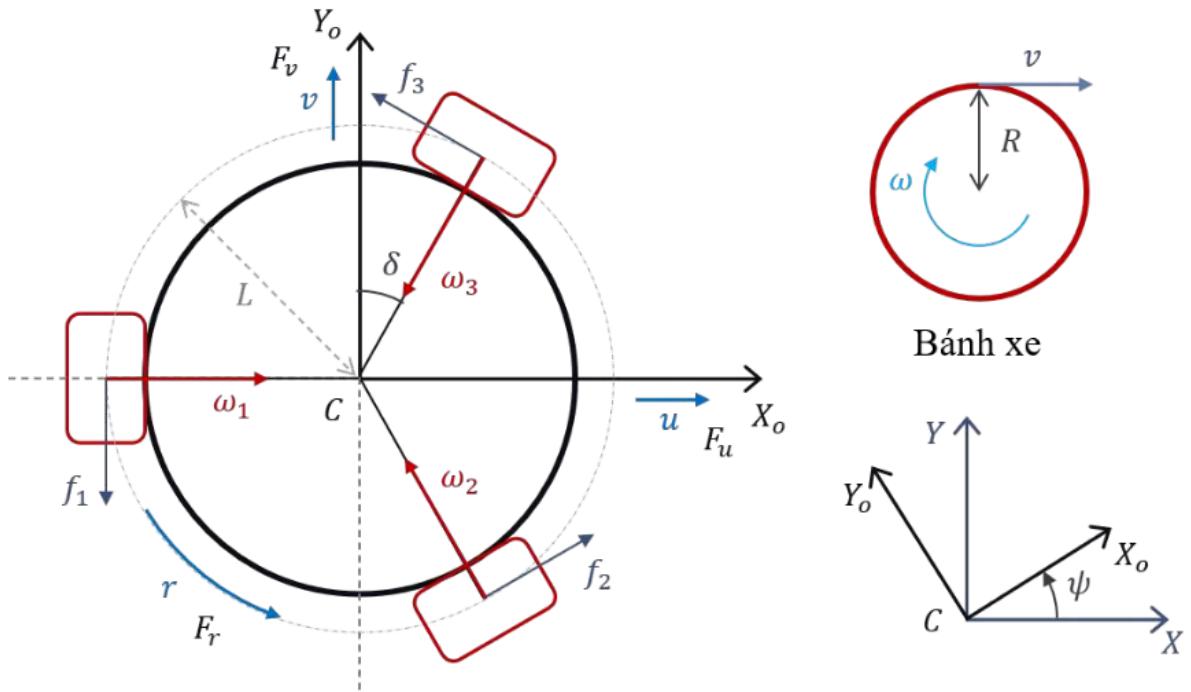
- **Khớp 2:** Quay đồng trực với khớp 1, phạm vi 180 độ, hỗ trợ nâng hạ vật thể.
- Tay máy được gắn trên một bệ xoay ở phần trên cùng của robot, đảm bảo hoạt động độc lập với chuyển động của chassis.



Hình 7. Tay máy

3. Cấu trúc hình học và mô hình toán của đối tượng

3.1. Cấu trúc hình học



Trong đó:

- CXY là hệ tọa độ tham chiếu toàn cục, giúp biểu diễn vị trí của Robot trên mặt phẳng (đồ án sẽ không xét đến việc điều khiển lên xuống dốc hoặc các dạng di chuyển vượt khỏi mặt phẳng 2D)
- C là tâm Robot. (Trong điều khiển bám lộ trình, coi như chúng ta đang điều khiển điểm C này bám theo lộ trình định trước)
- CX0Y0 là hệ tọa độ tham chiếu cục bộ gắn liền với Robot.
- ψ (rad/s) là góc lệch giữa 2 hệ tọa độ tham chiếu Toàn cục và Cục bộ
- r (rad/s) là tốc độ góc quay quanh trục của robot xét trên hệ quy chiếu cục bộ.
- u (m/s) là tốc độ thay đổi vị trí theo trục XO xét trên hệ quy chiếu cục bộ.
- v (m/s) là tốc độ thay đổi vị trí theo trục YO xét trên hệ quy chiếu cục bộ.
- v_1, v_2, v_3 vận tốc của mỗi bánh xe (m/s)
- $\omega_1, \omega_2, \omega_3$ vận tốc góc của mỗi bánh xe (rad/s)
- f_1, f_2, f_3 lực kéo của mỗi bánh xe (N)

Ngoài ra, mô hình Robot còn có một số ký hiệu vật lý sau:

- L là khoảng cách từ tâm bánh xe đến điểm C (m)

- R là bán kính bánh xe Omni (m)
- δ là góc định hướng bánh xe ($\delta=30^\circ$)
- m là khối lượng Robot (kg)
- JZ là Momen quán tính của Robot (N.m)

Vị trí của Omni Robot trong hệ tọa độ tham chiếu toàn cục được xác định bởi tọa độ X, Y và góc lệch ψ .

3.2. Phương trình động học

Ma trận quay RT (ψ) chuyển từ hệ tọa độ cục bộ sang hệ tọa độ toàn cục được thể hiện như sau:

$$R_T(\psi) = \begin{bmatrix} \cos(\psi) & -\sin(\psi) & 0 \\ \sin(\psi) & \cos(\psi) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Như vậy, để dễ dàng tính toán và biểu diễn các phương trình động học, động lực học, mối quan hệ giữa Vector vị trí Robot trên hệ quy chiếu toàn cục với vector tốc độ thay đổi vị trí Robot trên hệ quy chiếu cục bộ được thể hiện như sau:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\psi} \end{bmatrix} = \begin{bmatrix} \cos(\psi) & -\sin(\psi) & 0 \\ \sin(\psi) & \cos(\psi) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u \\ v \\ r \end{bmatrix}$$

Việc xác định mối quan hệ giữa Vận tốc vi của mỗi bánh xe với 3 đại lượng thành phần vận tốc u, v, r được thực hiện bằng phép cộng và nhân vector (do đã bỏ qua hiện tượng trượt bánh và rê bánh khi xe di chuyển). Ta được hệ phương trình sau:

$$v1 = -v + Lr$$

$$v2 = u \cos \delta + v \sin \delta + Lr$$

$$v3 = -u \cos \delta + v \sin \delta + Lr$$

Với $v_i = R\omega_i$ và $\delta = 30^\circ$, Ta có phương trình quan hệ giữa Vận tốc v_i của mỗi bánh xe với 3 đại lượng thành phần vận tốc u, v, r

$$\begin{bmatrix} u \\ v \\ r \end{bmatrix} = R \begin{bmatrix} 0 & -1 & L \\ \cos \delta & \sin \delta & L \\ -\cos \delta & \sin \delta & L \end{bmatrix}^{-1} \begin{bmatrix} \omega_1 \\ \omega_2 \\ \omega_3 \end{bmatrix}$$

Thay số và biến đổi phương trình, ta được:

$$\begin{bmatrix} u \\ v \\ r \end{bmatrix} = \frac{R}{3} \begin{bmatrix} 0 & \sqrt{3} & -\sqrt{3} \\ -2 & 1 & 1 \\ 1/L & 1/L & 1/L \end{bmatrix} \begin{bmatrix} \omega_1 \\ \omega_2 \\ \omega_3 \end{bmatrix}$$

Kết hợp phương trình, ra được phương trình động học của hệ thể hiện quan hệ giữa tốc độ góc của 3 bánh xe Omni với vị trí của Robot trong hệ quy chiếu toàn cục:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\psi} \end{bmatrix} = \frac{R}{3} \begin{bmatrix} \cos(\psi) & -\sin(\psi) & 0 \\ \sin(\psi) & \cos(\psi) & 0 \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 0 & \sqrt{3} & -\sqrt{3} \\ -2 & 1 & 1 \\ 1/L & 1/L & 1/L \end{bmatrix} \begin{bmatrix} \omega_1 \\ \omega_2 \\ \omega_3 \end{bmatrix}$$

3.3. Phương trình động lực học

Theo phương pháp Newton, các phương trình động lực học xét đến tác dụng của lực ma sát nhót và ma sát Coulomb:

$$\begin{cases} m \frac{du}{dt} = \sum F_u - F_{Bu} - F_{Cu} \\ m \frac{dv}{dt} = \sum F_v - F_{Bv} - F_{Cv} \\ J \frac{dr}{dt} = \sum F_r - F_{Br} - F_{Cr} \end{cases}$$

Trong đó:

- $F_{Bu} = B_u u$, $F_{Bv} = B_v v$, $F_{Br} = B_r r$: Lực ma sát nhót theo phương Xo,

Yo và phương quay của Robot.

- $F_{Cu} = C_u \text{sgn}(u)$, $F_{Cv} = C_v \text{sgn}(v)$, $F_{Cr} = C_r u \text{sgn}(r)$: Lực ma sát Coulomb theo phương Xo, Yo.

Phân tích vecter tương tự với vận tốc, ta có phương trình quan hệ giữa 3 đại lượng thành phần lực với lực kéo của các bánh xe thể hiện như sau:

$$\begin{bmatrix} F_u \\ F_v \\ F_r \end{bmatrix} = \begin{bmatrix} 0 & \frac{\sqrt{3}}{2} & -\frac{\sqrt{3}}{2} \\ -1 & 0.5 & 0.5 \\ L & L & L \end{bmatrix} \begin{bmatrix} f_1 \\ f_2 \\ f_3 \end{bmatrix}$$

Kết hợp 2 phương trình, ta được hệ phương trình tường minh như sau:

$$\begin{cases} m \frac{du}{dt} + B_u u + C_u \text{sgn}(u) = \frac{\sqrt{3}}{2} f_2 - \frac{\sqrt{3}}{2} f_3 \\ m \frac{dv}{dt} + B_v v + C_v \text{sgn}(v) = -f_1 + \frac{1}{2} f_2 + \frac{1}{2} f_3 \\ J_z \frac{dr}{dt} + B_r r + C_r \text{sgn}(r) = L f_1 + L f_2 + L f_3 \end{cases}$$

Quan hệ giữa lực kéo của mỗi bánh xe và momen xoắn do mỗi động cơ sinh ra thể hiện bởi phương trình: $\tau_i = R f_i$. Viết lại hệ phương trình với đại lượng Momen, ta được:

$$\begin{cases} m \frac{du}{dt} + B_u u + C_u sgn(u) = \frac{\sqrt{3}}{2R} \tau_2 - \frac{\sqrt{3}}{2R} \tau_3 \\ m \frac{dv}{dt} + B_v v + C_v sgn(v) = \frac{-1}{R} \tau_1 + \frac{1}{2R} \tau_2 + \frac{1}{2R} \tau_3 \\ J_z \frac{dr}{dt} + B_r r + C_r sgn(r) = \tau_1 \frac{L}{R} + \frac{L}{R} \tau_2 + \frac{L}{R} \tau_3 \end{cases}$$

Từ đó, ta có phương trình động lực học của Robot có dạng như sau:

$$M_{(q)} \begin{bmatrix} \dot{u} \\ \dot{v} \\ \dot{r} \end{bmatrix} + C \begin{bmatrix} u \\ v \\ r \end{bmatrix} + G sgn \begin{bmatrix} u \\ v \\ r \end{bmatrix} + \tau_d = B \begin{bmatrix} \tau_1 \\ \tau_2 \\ \tau_3 \end{bmatrix}$$

Trong đó:

- $M_{(q)}$ là ma trận khối lượng và Momen quán tính
- C là ma trận hệ số ma sát nhớt
- G là ma trận hệ số ma sát coulomb
- B là ma trận hệ số momen

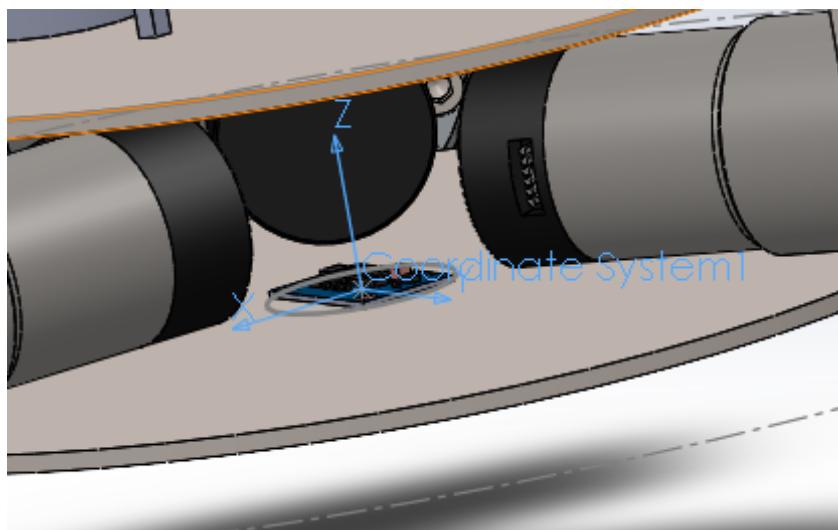
4. Mô hình hóa trong ROS (URDF/Xacro)

Mô hình robot được thiết kế trên SolidWorks, đảm bảo đúng tỷ lệ thực tế và có thể xuất sang URDF để tích hợp vào ROS. Hệ trục tọa độ được xác định như sau:

- Trục X: Hướng về phía trước của robot.
- Trục Y: Hướng sang trái hoặc phải của robot.
- Trục Z: Hướng lên trong không gian 3D.

Tiến hành gắn các trục cho xe(global), các bánh , tay máy, imu, camera

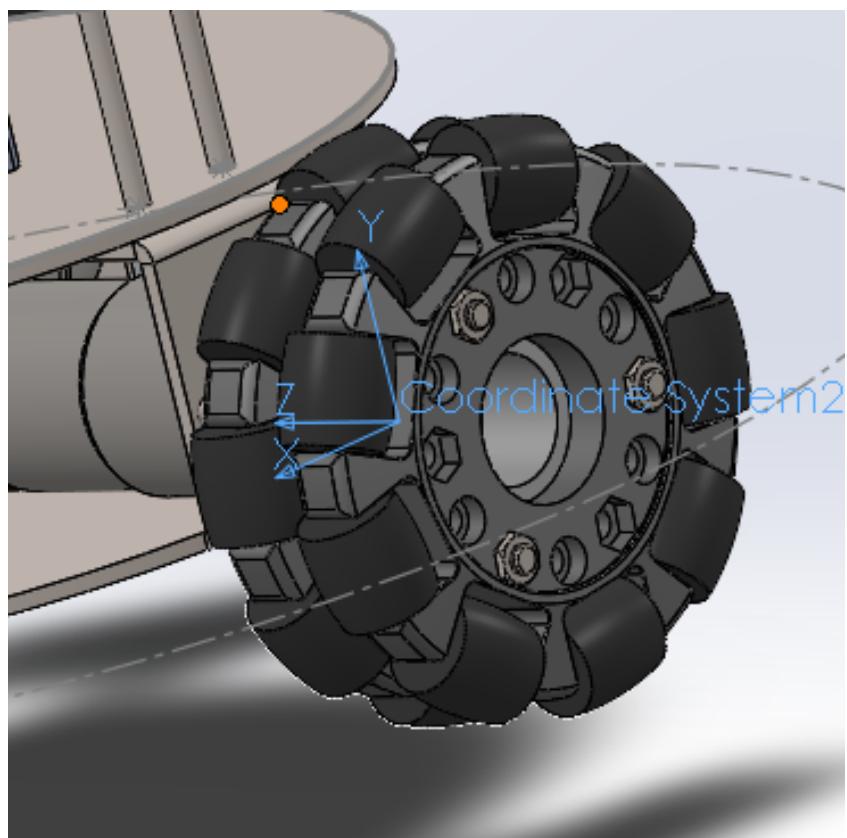
4.1. Hệ tọa độ global - Base_link



Hình 7. Trục tọa độ base_link

- Trục z hướng lên
- Trục x theo chiều tiến của robot

4.2. Hệ tọa độ Bánh xe



Hình 8. Hệ tọa độ bánh xe

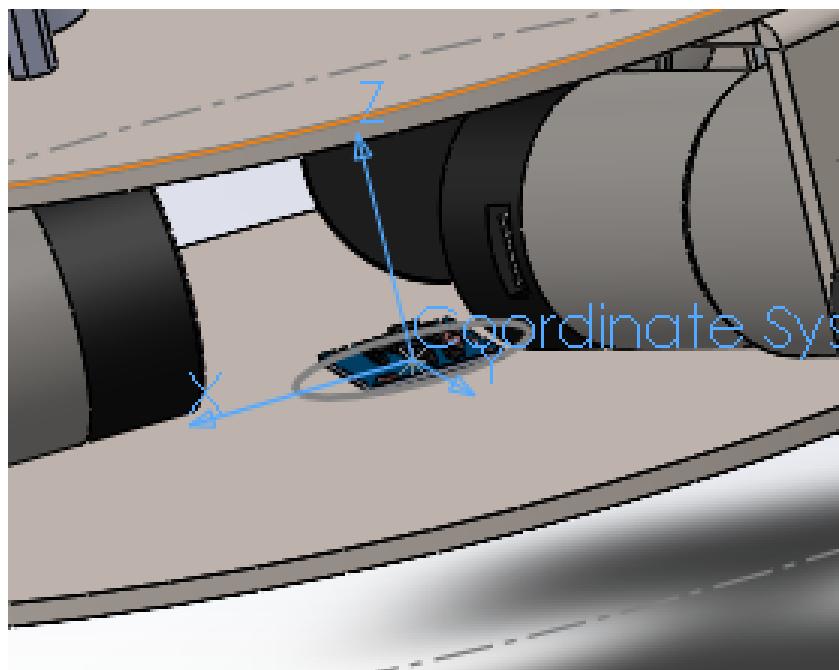
- Trục z là chiều quay của bánh xe



Hình 9. Trục axis bánh xe

Bánh xe quay quanh trục z và Axis 1 và tương ứng cho từng xe

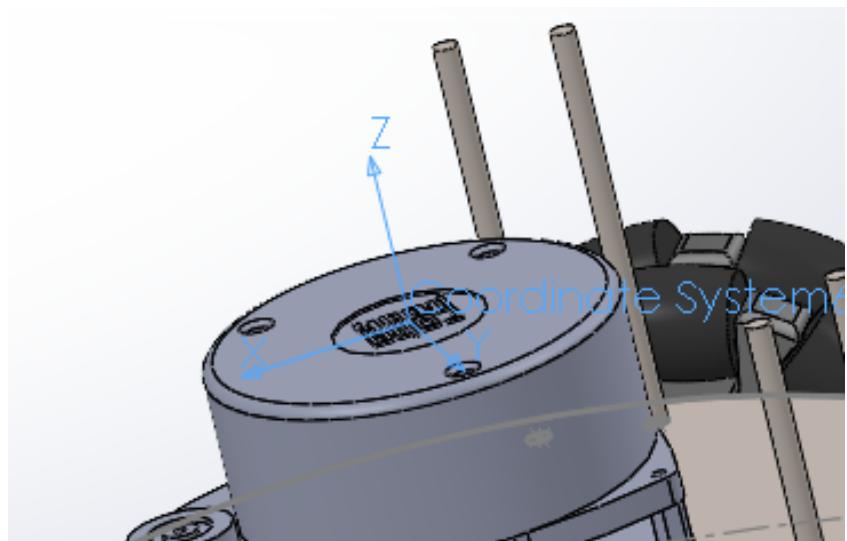
4.2. Hệ tọa độ IMU



Hình 10. Hệ tọa độ IMU

Trùng với base_link

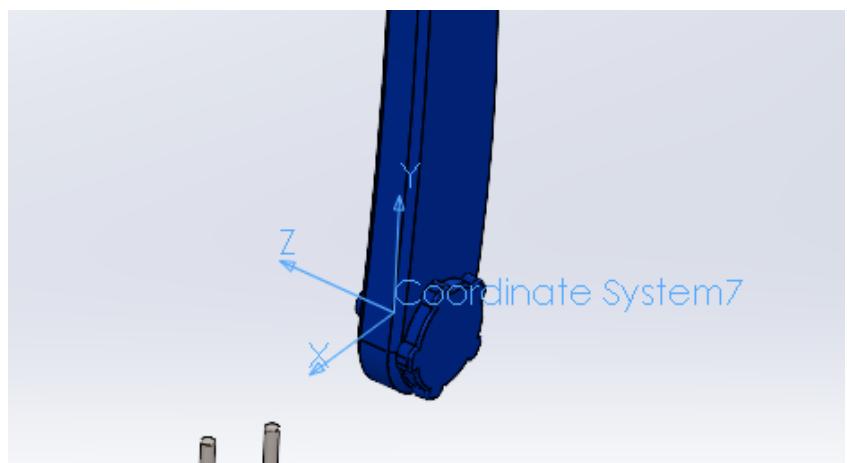
4.3. Hệ tọa độ LIDAR



Hình 11. Hệ tọa độ IMU

Trùng với base_link

4.4. Hệ tọa độ tay máy



Hình 12. Hệ tọa độ tay máy

2 tay máy được đặt đồng trục, chiều quay của tay máy là quanh trục z

4.2. URDF

4.2.1. Cấu trúc thư mục

Name	Last commit message	Last commit date
..		
config	Add files via upload	3 days ago
launch	Add files via upload	4 minutes ago
meshes	Add files via upload	3 days ago
rviz	Add files via upload	3 days ago
scripts	Add files via upload	4 minutes ago
urdf	Add files via upload	3 days ago
CMakeLists.txt	Add files via upload	4 minutes ago
export.log	Add files via upload	3 days ago
package.xml	Add files via upload	3 days ago

Hình 12. Cấu trúc thư mục

Để mô tả cấu trúc của robot trong ROS, **URDF/Xacro** được sử dụng với các thành phần chính:

Cấu trúc thư mục main mo_hinh_robot:

- **config/**: Có thể chứa các file cấu hình (như thông số PID, bộ lọc Kalman, v.v.).
- **launch/**: Chứa các file `.launch` để khởi chạy mô phỏng, RViz, Gazebo, hoặc node điều khiển.
- **meshes/**: Chứa các file mô hình 3D (ở dạng `.stl`) của robot.
- **rviz/**: Có thể chứa các cấu hình hiển thị cho **RViz**.
- **scripts/**: Chứa các script Python, có thể dùng để điều khiển hoặc test robot.
- **urdf/**: Chứa file mô tả robot **URDF/Xacro**.
- **CMakeLists.txt & package.xml**: Các file cần thiết để build package trong ROS.

Name	Last commit message	Last commit date
..	Add files via upload	3 days ago
config	Add files via upload	3 days ago
launch	Add files via upload	3 days ago
CMakeLists.txt	Add files via upload	3 days ago
package.xml	Add files via upload	3 days ago

Hình 13. Cấu trúc thư mục Moveit

Cấu trúc thư mục motion planning arm robot move_it_arm_sim:

- **config/**: Có thể chứa các file cấu hình.
- **launch/**: Chứa các file .launch để khởi chạy mô phỏng, RViz, Gazebo, hoặc node điều khiển.

4.2.2. Mô tả URDF File:

4.2.2.1. Mô tả chung về file:

File URDF (Unified Robot Description Format) là một định dạng XML được sử dụng trong ROS (Robot Operating System) để mô tả cấu trúc của robot. File này chứa thông tin về các **link** (bộ phận cứng), **joint** (khớp nối), và các thuộc tính bổ sung như quán tính, hình học, va chạm, cảm biến, và tích hợp với Gazebo để mô phỏng.

Trong file URDF này, robot được đặt tên là "**mo_hinh_robot**". Đây là một robot Omni 3 bánh với các thành phần chính bao gồm:

- **base_link**: Thân chính của robot.
- **left_wheel, right_wheel, front_wheel**: Ba bánh xe Omni.
- **lidar_link**: Cảm biến LiDAR.
- **imu_link**: Cảm biến IMU (Inertial Measurement Unit).
- **arm1, arm2**: Hai phần của cánh tay robot.

File URDF cũng bao gồm các phần mô phỏng trong Gazebo, như plugin điều khiển, cảm biến, và vật liệu hiển thị.

```
<link name="left_wheel">
  <inertial>
    <origin xyz="1.26531361333351E-06 -1.8579774514766E-06 -0.0154233151501505" rpy="0 0 0"/>
    <mass value="0.220455124017994"/>
    <inertia ixz="4.59354203353536E-05" ixy="-1.64874741472444E-09" ixz="3.11694735802205E-09"
          iyy="4.59367152538397E-05" iyz="-4.569390606387E-09" izz="8.21417212128144E-05"/>
  </inertial>
  <visual>
    <origin xyz="0 0 0" rpy="0 0 0"/>
    <geometry>
      <mesh filename="package://mo_hinh_robot/meshes/left_wheel.STL"/>
    </geometry>
    <material name="">
      <color rgba="0.752941176470588 0.752941176470588 0.752941176470588 1"/>
    </material>
  </visual>
  <collision>
    <origin xyz="0 0 0" rpy="0 0 0"/>
    <geometry>
      <mesh filename="package://mo_hinh_robot/meshes/left_wheel.STL"/>
    </geometry>
  </collision>
</link>

<joint name="left_wheel_joint" type="continuous">
  <origin xyz="-0.0643580948556634 0.111471490168346 0.0122233569354912" rpy="1.5707963267949 0 0.523598775598299"/>
  <parent link="base_link"/>
  <child link="left_wheel"/>
  <axis xyz="0 0 -1"/>
  <limit effort="10" velocity="10"/>
  <dynamics damping="0.1" friction="0.01"/>
</joint>
```

Hình 14. Ví dụ urdf của bánh trái

Link: left_wheel

- **Inertial:**
 - Tâm khối lượng: (1.265E-06, -1.858E-06, -0.01542) m, không xoay.
 - Khối lượng: 0.22045 kg.
 - Quán tính: ixx=4.594E-05, iyy=4.594E-05, izz=8.214E-05 (đối xứng cao).
 - Ý nghĩa: Mô tả phân bố khối lượng của bánh xe.
 - **Visual:**
 - Hình dạng: File STL left_wheel.STL.
 - Màu: Xám nhạt (0.753, 0.753, 0.753, 1).
 - Ý nghĩa: Quy định hình ảnh hiển thị.
 - **Collision:**
 - Hình dạng: File STL left_wheel.STL.
 - Ý nghĩa: Xác định vùng va chạm.
-

Joint: left_wheel_joint

- **Type:** continuous (quay vô hạn).
- **Origin:**
 - Vị trí: (-0.06436, 0.11147, 0.01222) m.
 - Hướng: (90°, 0°, 30°) (radian: 1.571, 0, 0.524).
 - Ý nghĩa: Gắn bánh xe lệch trái thân robot.
- **Parent:** base_link.
- **Child:** left_wheel.
- **Axis:** (0, 0, -1) (quay quanh trục Z ngược).
- **Limit:** Effort=10, Velocity=10.
- **Dynamics:** Damping=0.1, Friction=0.01.
- **Ý nghĩa:** Cho phép bánh xe quay tự do, hỗ trợ chuyển động Omni.

4.2.2.2. Plugin cảm biến

a. Lidar

```

<gazebo reference="lidar_link">
  <sensor type="ray" name="lidar_sensor">
    <pose>0 0 0 0 0 0</pose>
    <visualize>true</visualize>
    <update_rate>30</update_rate>
    <ray>
      <scan>
        <horizontal>
          <samples>720</samples>
          <resolution>1</resolution>
          <min_angle>-1.5708</min_angle>
          <max_angle>1.5708</max_angle>
        </horizontal>
      </scan>
      <range>
        <min>0.1</min>
        <max>30.0</max>
        <resolution>0.01</resolution>
      </range>
      <noise>
        <type>gaussian</type>
        <mean>0.0</mean>
        <stddev>0.01</stddev>
      </noise>
    </ray>
    <plugin name="gazebo_ros_laser" filename="libgazebo_ros_laser.so">
      <topicName>/scan</topicName>
      <frameName>lidar_link</frameName>
    </plugin>
  </sensor>
</gazebo>

```

Hình 15. Plugin LiDAR

- **Name:** gazebo_ros_laser - Tên plugin.
- **Filename:** libgazebo_ros_laser.so - File thư viện thực thi plugin.
- **TopicName:** /scan - Topic ROS xuất dữ liệu LiDAR.
- **FrameName:** lidar_link - Khung tham chiếu của cảm biến.

Tác dụng

- **Kết nối Gazebo và ROS:** Plugin này tích hợp cảm biến LiDAR trong Gazebo với ROS, xuất dữ liệu quét (scan) lên topic /scan.
- **Định khung:** Liên kết dữ liệu với lidar_link, đảm bảo vị trí và hướng của cảm biến trong mô phỏng được đồng bộ.
- **Hỗ trợ mô phỏng:** Cho phép robot sử dụng dữ liệu LiDAR (khoảng cách, góc) để điều hướng hoặc lập bản đồ trong ROS.

b. IMU

```

<gazebo reference="imu_link">
  <sensor name="imu_sensor" type="imu">
    <always_on>true</always_on>
    <update_rate>100</update_rate>
    <visualize>true</visualize> <!-- Bật hiển thị mũi tên -->
    <plugin filename="libgazebo_ros_imu_sensor.so" name="imu_plugin">
      <topicName>/imu</topicName>
      <bodyName>imu_link</bodyName>
      <frameName>imu_link</frameName>
      <gaussianNoise>0.01</gaussianNoise>
      <xyzOffset>0 0 0</xyzOffset>
      <rpyOffset>0 0 0</rpyOffset> <!-- Không offset góc dữ liệu -->
      <initialOrientationAsReference>false</initialOrientationAsReference>
    </plugin>
    <pose>0 0 0 0 1.5708 0</pose> <!-- Xoay 90 độ quanh trục Y để mũi tên hướng theo X -->
  </sensor>
</gazebo>

```

Hình 15. Plugin IMU

- **Name:** imu_plugin - Tên plugin.
- **Filename:** libgazebo_ros_imu_sensor.so - File thư viện thực thi plugin.
- **TopicName:** /imu - Topic ROS xuất dữ liệu IMU.
- **BodyName/FrameName:** imu_link - Khung tham chiếu của cảm biến.
- **GaussianNoise:** 0.01 - Nhiêu Gaussian cho dữ liệu.

5. Điều khiển trong ROS

5.1. ROS Control

Để mô phỏng robot Omni 3 bánh trong **Gazebo** với các tính năng phức tạp (như điều khiển bánh xe Omni và cánh tay robot), tôi sử dụng giao diện dựa trên **pluginlib** để tích hợp **Gazebo** với **ROS Control**. Phương pháp này cho phép điều khiển các khớp của robot (bánh xe và tay máy) một cách linh hoạt và chính xác. Dưới đây là các bước triển khai:

a. Sử Dụng Plugin `gazebo_ros_control`

Plugin `gazebo_ros_control` kết nối Gazebo với ROS Control thông qua một giao diện phần cứng mô phỏng. Để tùy chỉnh cho robot Omni 3 bánh, tôi tạo một lớp con của **omni 3 bánh** và chỉ định nó trong file URDF của robot. Đoạn mã XML sau là một ví dụ tích hợp plugin:

```

<gazebo>
  <plugin name ="gazebo_ros_control" filename = "libgazebo_ros_control.so">
    | | <robotNamespace>/</robotNamespace>
  </plugin>
</gazebo>

```

name: Tên của plugin, ở đây là gazebo_ros_control.

filename: Thư viện thực thi của plugin (libgazebo_ros_control.so).

robotNamespace: Không gian tên của robot trong ROS (Ở đây là /).

b. Định nghĩa Transmission trong URDF

```

<!-- Transmission cho bánh xe -->
<transmission name="left_wheel_trans">
  <type>transmission_interface/SimpleTransmission</type>
  <joint name="left_wheel_joint">
    | <hardwareInterface>hardware_interface/VelocityJointInterface</hardwareInterface>
  </joint>
  <actuator name="left_wheel_motor">
    | <mechanicalReduction>1</mechanicalReduction>
  </actuator>
</transmission>

```

Hình 15. Transmission cho bánh trái

Name: left_wheel_trans - Transmission cho left_wheel

Type: transmission_interface/SimpleTransmission - Truyền động đơn giản.

Joint:

- Tên: left_wheel_joint.
- Hardware Interface: VelocityJointInterface - Giao diện điều khiển vận tốc.

```

<transmission name="arm1_trans">
  <type>transmission_interface/SimpleTransmission</type>
  <joint name="arm1_joint">
    <hardwareInterface>hardware_interface/PositionJointInterface</hardwareInterface>
  </joint>
  <actuator name="arm1_motor">
    <!-- <hardwareInterface>hardware_interface/PositionJointInterface</hardwareInterface> -->
    <mechanicalReduction>1</mechanicalReduction>
  </actuator>
</transmission>

```

Hình 15. Transmission cho khớp tay 1

Name: arm1_trans - Transmission cho cánh tay 1.

Type: transmission_interface/SimpleTransmission - Truyền động đơn giản.

Joint:

- *Tên:* arm1_joint.
- *Hardware Interface:* PositionJointInterface - Điều khiển vị trí.

Mục đích của **transmission** trong file URDF là:

- **Kết nối khớp và bộ truyền động:** Nó liên kết một khớp (joint) với một bộ truyền động (actuator, thường là motor), cho phép điều khiển chuyển động của robot (vị trí, vận tốc, hoặc lực).
- **Xác định giao diện điều khiển:** Quy định cách ROS giao tiếp với khớp qua các giao diện phần cứng như PositionJointInterface (điều khiển vị trí) hoặc VelocityJointInterface (điều khiển vận tốc).
- **Mô phỏng và thực tế:** Đảm bảo khớp hoạt động nhất quán trong mô phỏng (như Gazebo) và trên phần cứng thực tế bằng cách mô tả cơ chế truyền động (tỷ lệ giảm tốc, loại truyền động).

c. Cấu hình file YAML

```

joint_state_controller:
  type: "joint_state_controller/JointStateController"
  publish_rate: 50
# Controller to control robot base joints
left_wheel_joint_velocity_controller:
  type: "velocity_controllers/JointVelocityController"
  joint: "left_wheel_joint"
  pid: {p: 100.0, i: 0.1, d: 10.0}

right_wheel_joint_velocity_controller:
  type: "velocity_controllers/JointVelocityController"
  joint: "right_wheel_joint"
  pid: {p: 100.0, i: 0.1, d: 10.0}

front_wheel_joint_velocity_controller:
  type: "velocity_controllers/JointVelocityController"
  joint: "front_wheel_joint"
  pid: {p: 100.0, i: 0.1, d: 10.0}
# Controller to control robot arm joints
arm_1_joint_controller:
  type: "position_controllers/JointPositionController"
  joint: "arm1_joint"
  pid: {p: 100.0, i: 0.1, d: 5.0}

arm_2_joint_controller:
  type: "position_controllers/JointPositionController"
  joint: "arm2_joint"
  pid: {p: 100.0, i: 0.1, d: 5.0}

```

Hình 15. File config YAML

- **joint_state_controller:** Xuất trạng thái khớp (vị trí, vận tốc) với tần suất 50 Hz.
- **velocity_controllers:** Điều khiển vận tốc cho 3 bánh xe.
- **position_controllers:** Điều khiển vị trí cho 2 khớp tay máy.

d. File launch

File launch (mo_hinh_robot/launch/gazebo.launch) khởi động các controller:

```

<!-- Khởi động bộ điều khiển -->
<node
    name="wheel_controller_spawner"
    pkg="controller_manager"
    type="spawner"
    args="  joint_state_controller
           left_wheel_joint_velocity_controller
           right_wheel_joint_velocity_controller
           front_wheel_joint_velocity_controller
           arm_1_joint_controller
           arm_2_joint_controller" />

```

Hình 15. Truyền controller vào launch

Sau khi chạy file gazebo.launch, các controller đã được khởi tạo và ta có thể viết các file python hoặc C++ để điều khiển các khớp tay máy hoặc bánh xe bằng cách publish data thông qua controller/command.

e. Scripts Python điều khiển bánh xe

```

class OmniKeyboardControl:
    def __init__(self):
        # Khởi tạo node ROS
        rospy.init_node('omni_keyboard_control', anonymous=True)

        # Publisher cho các lệnh vận tốc bánh xe
        self.pub_left = rospy.Publisher('/left_wheel_joint_velocity_controller/command', Float64, queue_size=10)
        self.pub_right = rospy.Publisher('/right_wheel_joint_velocity_controller/command', Float64, queue_size=10)
        self.pub_front = rospy.Publisher('/front_wheel_joint_velocity_controller/command', Float64, queue_size=10)

        # Tốc độ tối đa (radian/s)
        self.max_speed = 10.0 # Giới hạn từ URDF: velocity="10"

        # Vận tốc hiện tại của từng bánh
        self.left_speed = 0.0
        self.right_speed = 0.0
        self.front_speed = 0.0

```

- **rospy.init_node:** Khởi tạo một node ROS với tên 'omni_keyboard_control'. Tham số anonymous=True đảm bảo tên node là duy nhất nếu có nhiều instance chạy cùng lúc.
- **Publisher:** Tạo 3 publisher để gửi lệnh vận tốc đến các controller của bánh xe:
 - /left_wheel_joint_velocity_controller/command: Điều khiển bánh trái.
 - /right_wheel_joint_velocity_controller/command: Điều khiển bánh phải.
 - /front_wheel_joint_velocity_controller/command: Điều khiển bánh trước.
 Mỗi publisher gửi message kiểu Float64 với queue_size=10.
- **self.max_speed:** Đặt tốc độ tối đa của bánh xe là 10.0 radian/s, dựa trên giới hạn trong file URDF của robot.
- **Vận tốc ban đầu:** Khởi tạo vận tốc của từng bánh (left_speed, right_speed, front_speed) bằng 0.0.

```

def run(self):
    rospy.loginfo("Điều khiển robot omni 3 bánh bằng bàn phím:")
    rospy.loginfo("w: Tiết, s: Lùi, a: Trái, d: Phải, q: Xoay trái, e: Xoay phải")
    rospy.loginfo("x: Dừng và thoát")

    while not rospy.is_shutdown():
        key = self.get_key()

        # Đặt lại vận tốc về 0 trước khi tính toán
        self.left_speed = 0.0
        self.right_speed = 0.0
        self.front_speed = 0.0

        # Điều khiển chuyển động
        if key == 'w': # Tiết
            self.left_speed = self.max_speed
            self.right_speed = -self.max_speed
            self.front_speed = 0.0
        elif key == 's': # Lùi
            self.left_speed = -self.max_speed
            self.right_speed = self.max_speed
            self.front_speed = 0.0
        elif key == 'a': # Sang trái
            self.left_speed = self.max_speed
            self.right_speed = self.max_speed
            self.front_speed = -self.max_speed
        elif key == 'd': # Sang phải
            self.left_speed = -self.max_speed
            self.right_speed = -self.max_speed
            self.front_speed = self.max_speed
        elif key == 'q': # Xoay trái
            self.left_speed = -self.max_speed
            self.right_speed = -self.max_speed
            self.front_speed = -self.max_speed
        elif key == 'e': # Xoay phải
            self.left_speed = self.max_speed
            self.right_speed = self.max_speed
            self.front_speed = self.max_speed
        elif key == 'z': # Dừng và thoát
            self.left_speed = 0.0
            self.right_speed = 0.0
            self.front_speed = 0.0
        elif key == 'x': # Dừng và thoát
            self.left_speed = 0.0
            self.right_speed = 0.0
            self.front_speed = 0.0
            self.pub_left.publish(self.left_speed)
            self.pub_right.publish(self.right_speed)
            self.pub_front.publish(self.front_speed)
            rospy.loginfo("Dừng robot và thoát")
            break

        # Gửi lệnh vận tốc
        self.pub_left.publish(self.left_speed)
        self.pub_right.publish(self.right_speed)
        self.pub_front.publish(self.front_speed)

        # Hiển thị trạng thái
        rospy.loginfo(f"Left: {self.left_speed:.2f}, Right: {self.right_speed:.2f}, Front: {self.front_speed:.2f}")

        rospy.sleep(0.1) # Tránh đọc phím quá nhanh

```

Vòng lặp chính:

- Chạy liên tục cho đến khi ROS bị tắt (rospy.is_shutdown()) hoặc người dùng nhấn 'x'.
- Trong mỗi lần lặp:
 - Đọc phím:** Gọi get_key() để lấy phím người dùng nhấn.
 - Đặt lại vận tốc:** Đặt vận tốc của cả 3 bánh về 0 trước khi xử lý phím mới.
 - Xử lý phím:**

- 'w' (Tiến): Bánh trái quay thuận (max_speed), bánh phải quay ngược (-max_speed), bánh trước dừng.
- 's' (Lùi): Ngược lại với tiến.
- 'a' (Sang trái): Cả 3 bánh quay để di chuyển ngang sang trái.
- 'd' (Sang phải): Ngược lại với sang trái.
- 'q' (Xoay trái): Các bánh quay để robot xoay tại chỗ theo chiều kim đồng hồ.
- 'e' (Xoay phải): Ngược lại với xoay trái.
- 'z' (Đứng): Tất cả bánh dừng.
- 'x' (Thoát): Dừng bánh, gửi lệnh cuối cùng, và thoát vòng lặp.
- **Gửi lệnh vận tốc:** Sử dụng publish để gửi giá trị vận tốc đến từng bánh xe qua các publisher.
- **Hiển thị trạng thái:** In ra vận tốc hiện tại của từng bánh để người dùng theo dõi.
- **Ngủ:** rospy.sleep(0.1) tạm dừng 0.1 giây để tránh đọc phím quá nhanh.

f. Scripts Python điều khiển tay máy (Tương tự với bánh xe)

```

def __init__(self):
    # Khởi tạo node ROS
    rospy.init_node('keyboard_arm_control', anonymous=True)

    # Biến lưu trữ góc hiện tại của các khớp
    self.joint1_pos = 0.0 # arm1_joint
    self.joint2_pos = 0.0 # arm2_joint

    # Publisher cho lệnh điều khiển khớp
    self.pub_joint1 = rospy.Publisher('/arm_1_joint_controller/command', Float64, queue_size=10)
    self.pub_joint2 = rospy.Publisher('/arm_2_joint_controller/command', Float64, queue_size=10)

    # Subscriber để nhận trạng thái khớp
    rospy.Subscriber('/joint_states', JointState, self.joint_state_callback)

    # Tốc độ thay đổi góc (radian mỗi lần nhảy)
    self.step = 0.1

    # Giới hạn góc khớp (theo URDF của bạn: -1.57 đến 1.57 rad)
    self.min_angle = -1.57
    self.max_angle = 1.57

```

- **Biến lưu trữ góc:**
 - self.joint1_pos: Góc hiện tại của khớp arm1_joint, khởi tạo là 0.0 radian.
 - self.joint2_pos: Góc hiện tại của khớp arm2_joint, khởi tạo là 0.0 radian.
- **Publisher:**
 - self.pub_joint1: Publisher gửi lệnh góc đến topic /arm_1_joint_controller/command cho khớp arm1_joint.

- self.pub_joint2: Publisher gửi lệnh góc đến topic /arm_2_joint_controller/command cho khớp arm2_joint.
- Cả hai publisher đều gửi message kiểu Float64 với queue_size=10.
- **Subscriber:** Đăng ký lắng nghe topic /joint_states để nhận trạng thái hiện tại của các khớp. Callback joint_state_callback sẽ xử lý các message nhận được.
- **self.step = 0.1:** Bước thay đổi góc mỗi lần nhấn phím là 0.1 radian.
- **Giới hạn góc:**
 - self.min_angle = -1.57: Góc tối thiểu (-90°).
 - self.max_angle = 1.57: Góc tối đa (90°).
- g. Hiển thị encoder

```

class WheelEncoderReader:
    def __init__(self):
        # Khởi tạo node ROS
        rospy.init_node('wheel_encoder_reader', anonymous=True)

        # Thông số bánh xe (giả định)
        self.wheel_radius = 0.015423 # Bán kính bánh xe (m), từ URDF: diameter=0.030846
        self.pulses_per_revolution = 360 # Số xung trên mỗi vòng quay (giả định, thay đổi nếu có encoder thực tế)

        # Biến lưu trữ dữ liệu
        self.left_pos = 0.0 # Vị trí góc (rad) của bánh trái
        self.right_pos = 0.0
        self.front_pos = 0.0
        self.left_vel = 0.0 # Vận tốc góc (rad/s) của bánh trái
        self.right_vel = 0.0
        self.front_vel = 0.0

        # Subscriber cho topic /joint_states
        rospy.Subscriber('/joint_states', JointState, self.joint_state_callback)

    def joint_state_callback(self, msg):
        # Lấy dữ liệu từ topic /joint_states
        try:
            # Tìm chỉ số của các khớp bánh xe trong message
            left_idx = msg.name.index('left_wheel_joint')
            right_idx = msg.name.index('right_wheel_joint')
            front_idx = msg.name.index('front_wheel_joint')

            # Cập nhật vị trí và vận tốc
            self.left_pos = msg.position[left_idx]
            self.right_pos = msg.position[right_idx]
            self.front_pos = msg.position[front_idx]
            self.left_vel = msg.velocity[left_idx]
            self.right_vel = msg.velocity[right_idx]
            self.front_vel = msg.velocity[front_idx]

        except ValueError as e:
            rospy.logwarn(f'Không tìm thấy một số khớp trong /joint_states: {e}')

    def calculate_pulses(self, position):
        # Tính số xung từ vị trí góc (rad)
        revolutions = position / (2 * math.pi) # Số vòng quay
        pulses = revolutions * self.pulses_per_revolution # Số xung
        return int(pulses) # Lấy tròn về số nguyên

    def run(self):
        rate = rospy.Rate(10) # 10 Hz
        rospy.loginfo("Đang đọc dữ liệu encoder từ các bánh xe...")

        while not rospy.is_shutdown():
            # Tính số xung cho từng bánh
            left_pulses = self.calculate_pulses(self.left_pos)
            right_pulses = self.calculate_pulses(self.right_pos)
            front_pulses = self.calculate_pulses(self.front_pos)

            # Tính quãng đường di chuyển (m) từ vị trí góc
            left_distance = self.left_pos * self.wheel_radius
            right_distance = self.right_pos * self.wheel_radius
            front_distance = self.front_pos * self.wheel_radius

            # Hiển thị thông tin
            rospy.loginfo("== Dữ liệu Encoder ==")
            rospy.loginfo(f"Left Wheel: Pulses={left_pulses}, Pos={self.left_pos:.3f} rad, Vel={self.left_vel:.3f} rad/s, Distance={left_distance:.3f} m")
            rospy.loginfo(f"Right Wheel: Pulses={right_pulses}, Pos={self.right_pos:.3f} rad, Vel={self.right_vel:.3f} rad/s, Distance={right_distance:.3f} m")
            rospy.loginfo(f"Front Wheel: Pulses={front_pulses}, Pos={self.front_pos:.3f} rad, Vel={self.front_vel:.3f} rad/s, Distance={front_distance:.3f} m")

        rate.sleep()

```

Đoạn mã sử dụng ROS để đọc dữ liệu từ topic /joint_states, lấy vị trí và vận tốc góc của 3 bánh xe robot (trái, phải, trước). Từ đó, nó tính toán:

- **Số xung:** Dựa trên vị trí góc và số xung mỗi vòng quay (giả định 360).

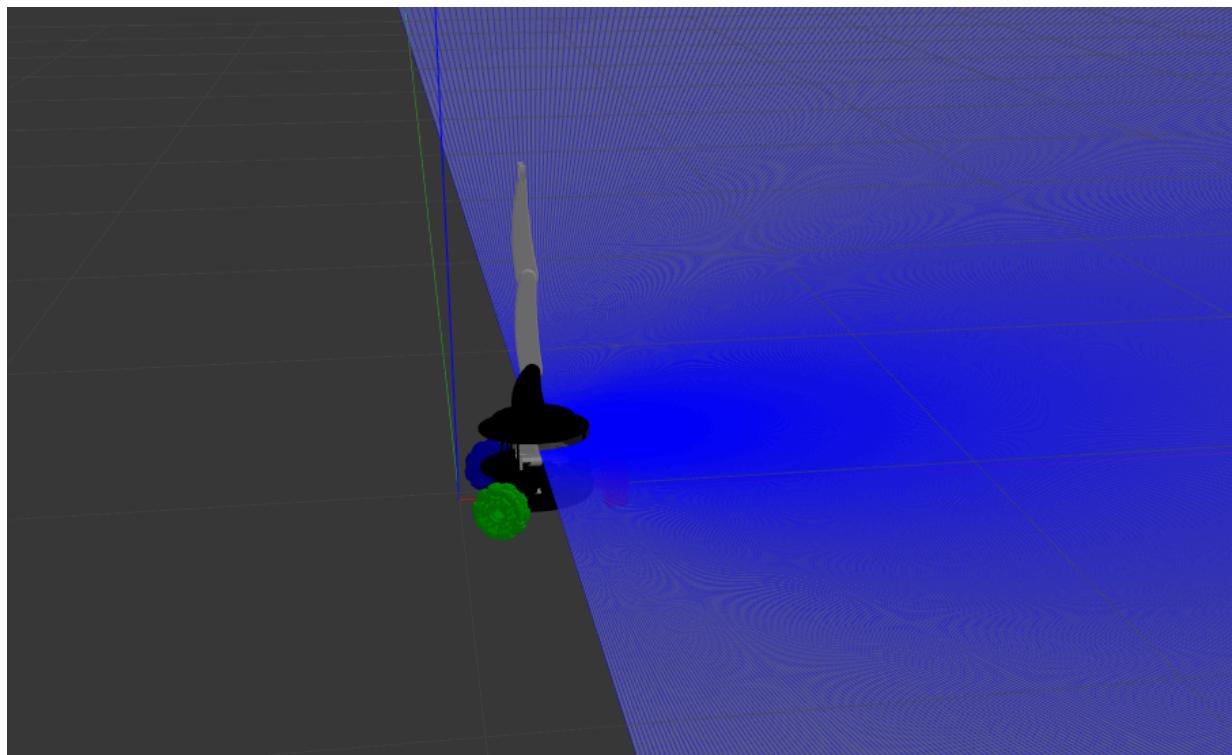
- **Quãng đường:** Nhân bán kính bánh xe (0.015423 m) với vị trí góc.

Chương trình chạy vòng lặp 10 Hz, in ra:

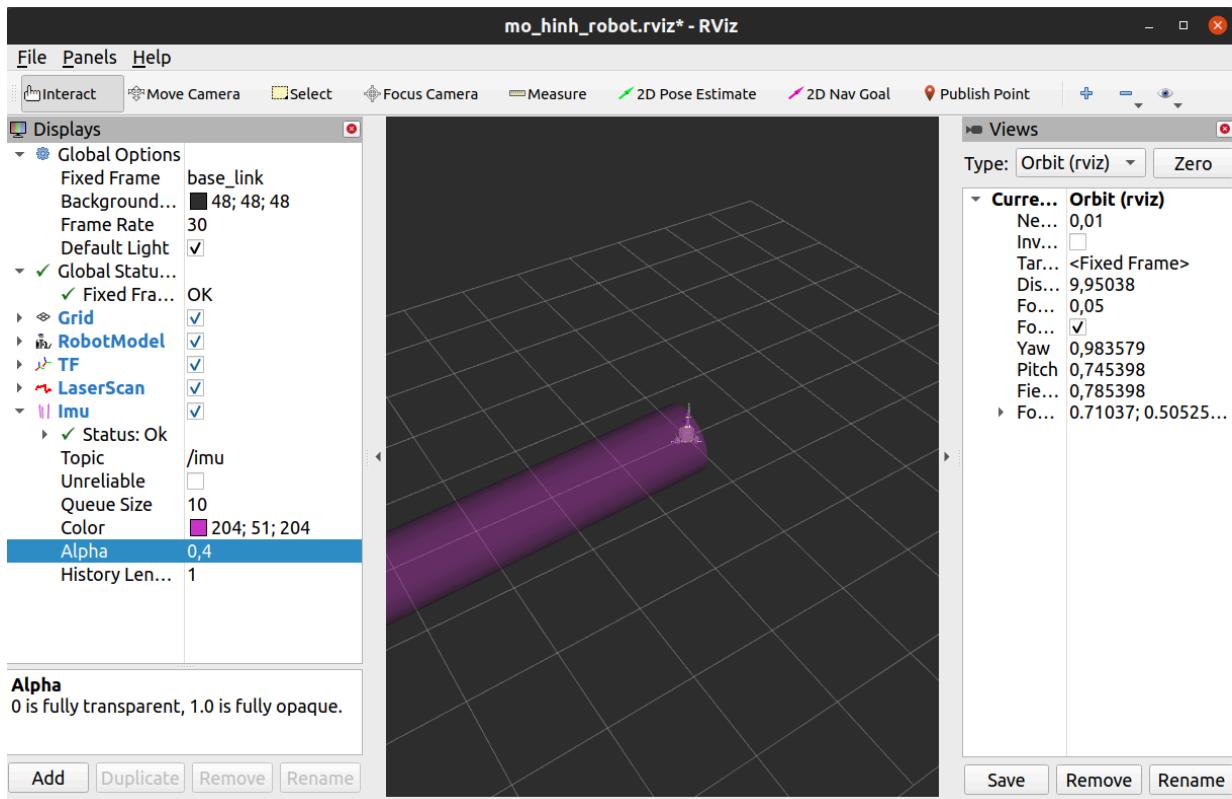
- Số xung.
- Vị trí góc (rad).
- Vận tốc góc (rad/s).
- Quãng đường (m) của từng bánh.

Encoder chỉ theo dõi và in dữ liệu, không điều khiển robot, hữu ích cho việc phân tích chuyển động.

6. Hiển thị trong Gazebo và Rviz

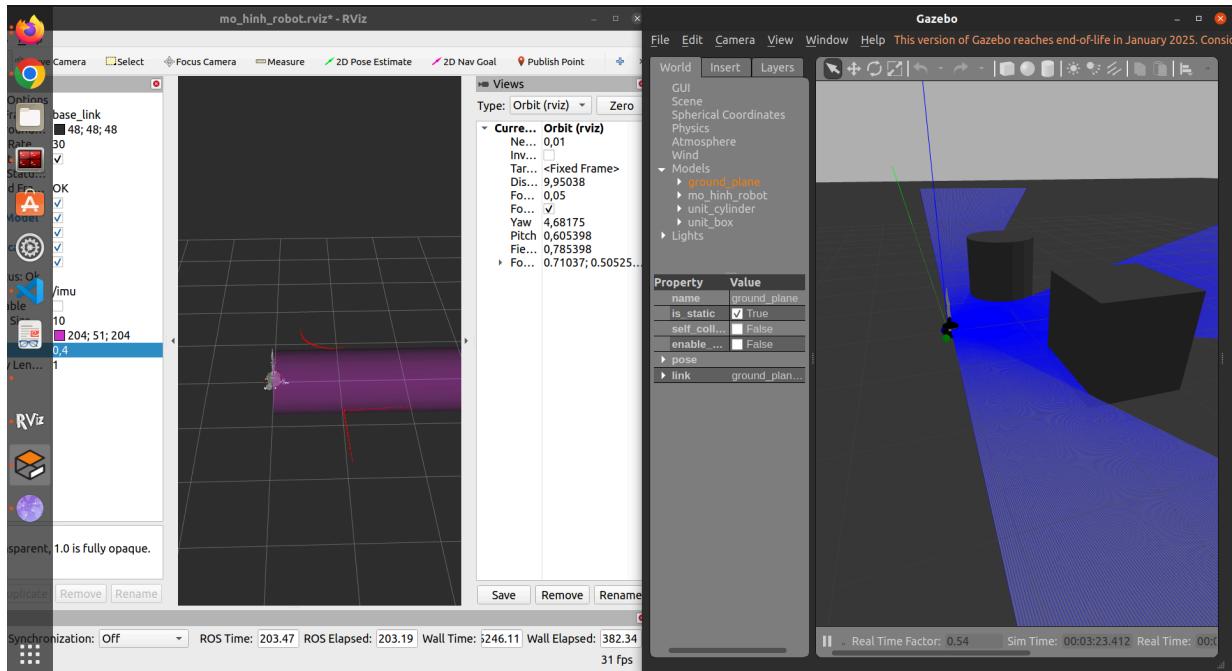


Hình 16. Hiển thị Gazebo



Hình 17. Hiển thị RViz

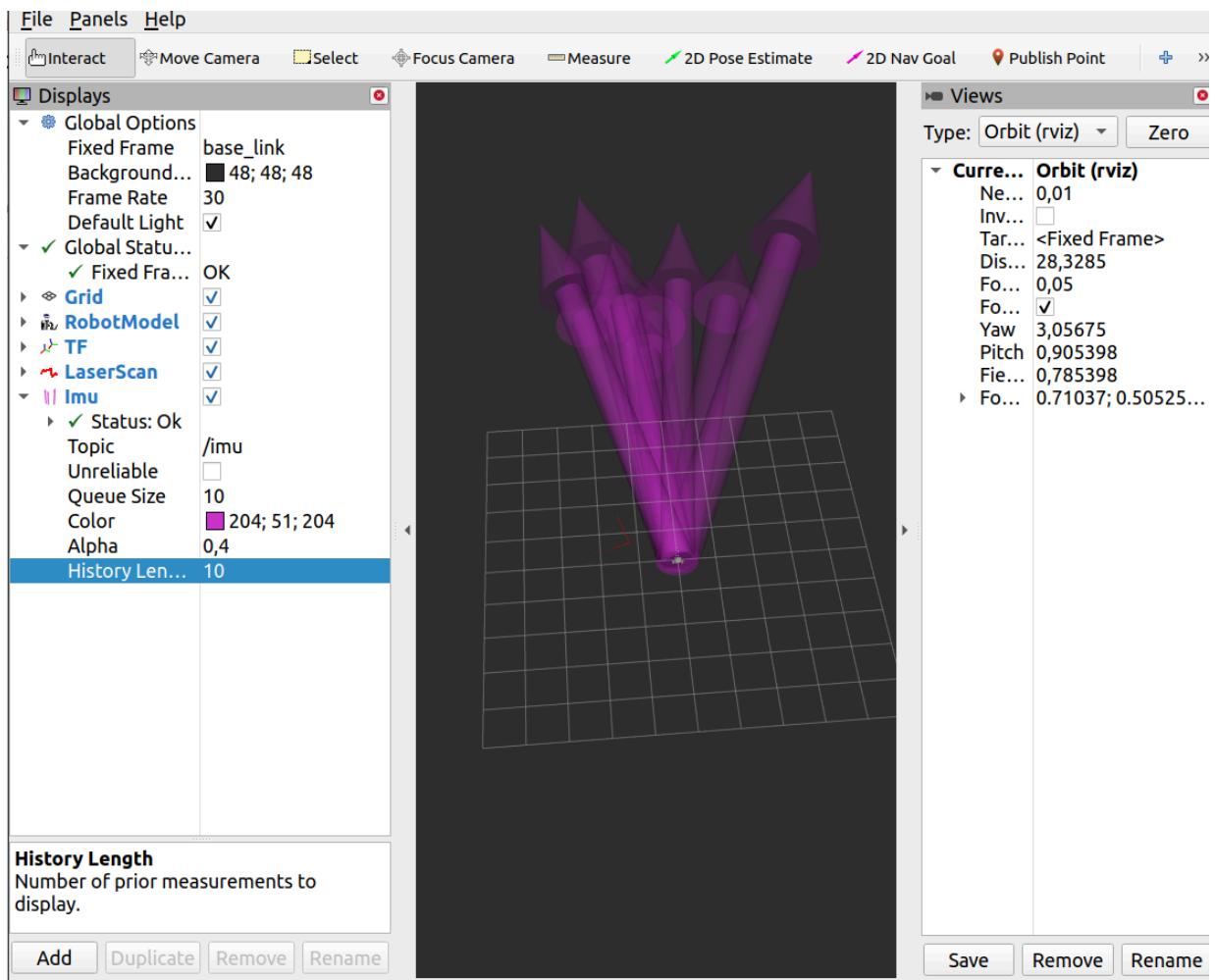
1. Hiển thị LIDAR



Hình 18. Hiển thị LIDAR

Khi đặt vật thể trong tầm quét của robot, bên Rviz xuất hiện vật thể

2. Hiển thị IMU



Hình 19. Hiển thị IMU

3. Hiển thị Encoder

```

bang@bang-Nitro-AN515-58: ~/test_ws
[urdf_spawner-7] process has finished cleanly
log file: /home/bang/.ros/log/189ecb8c-0d8d-11f0-914a-673e2d945be3/urdf_spawner-7*.log

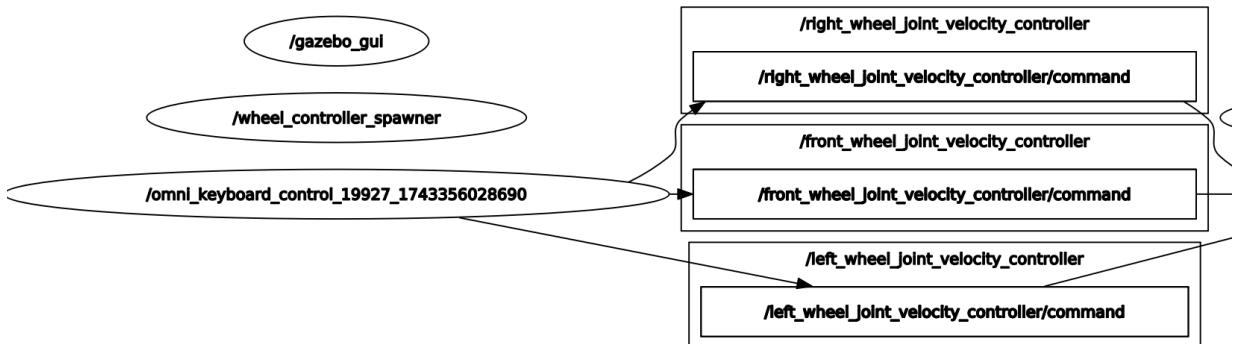
bang@bang-Nitro-AN515-58: ~/test_ws 107x8
[INFO] [1743355827.367813, 507.200000]: Left: -10.00, Right: 10.00, Front: 10.00
[INFO] [1743355977.832491, 585.918000]: Dùng robot và thoát
bang@bang-Nitro-AN515-58:~/test_ws$ rosrn mo_hinh_robot omni_keyboard_control.py
[INFO] [1743356028.821776, 9.496000]: Điều khiển robot omni 3 bánh bằng bàn phím:
[INFO] [1743356028.822548, 9.496000]: w: Tiền, s: Lùi, a: Trái, d: Phải, q: Xoay trái, e: Xoay phải
[INFO] [1743356028.823180, 9.497000]: x: Dừng và thoát
[INFO] [1743356030.469038, 10.380000]: Left: 10.00, Right: -10.00, Front: -10.00

bang@bang-Nitro-AN515-58: ~/test_ws 107x8
[INFO] [1743356065.075120, 28.700000]: left_wheel_joint: Position = 0.000 rad, Velocity = 0.000 rad/s
[INFO] [1743356065.076024, 28.700000]: right_wheel_joint: Position = 0.000 rad, Velocity = 0.000 rad/s
[INFO] [1743356065.076607, 28.700000]: arm1_joint: Position = 0.000 rad, Velocity = 0.000 rad/s
[INFO] [1743356065.077137, 28.701000]: arm2_joint: Position = 0.000 rad, Velocity = 0.000 rad/s
[INFO] [1743356065.094082, 28.710000]: arm1_joint: Position = -0.000 rad, Velocity = -0.000 rad/s
[INFO] [1743356065.094971, 28.710000]: arm2_joint: Position = 0.000 rad, Velocity = 0.000 rad/s
[INFO] [1743356065.095475, 28.710000]: left_wheel_joint: Position = 178.861 rad, Velocity = 10.000 rad/s
[INFO] [1743356065.095938, 28.711000]: right_wheel_joint: Position = -180.542 rad, Velocity = -10.000 rad/s

```

Hình 19. Hiển thị Encoder

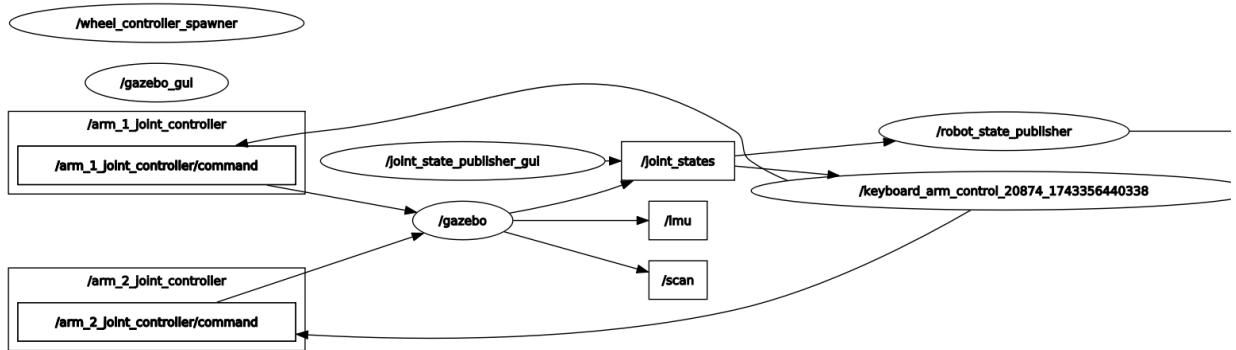
4. Hiển thị điều khiển bánh xe:



Hình 20. Hiển thị điều khiển bánh xe

Hàm scripts Python omni_keyboard_control trỏ đến 3 bánh

5. Hiển thị điều khiển tay máy:



Hình 21. Hiển thị điều khiển tay máy

7. Motion Planning tay máy bằng Moveit

Sau khi viết file YAML và Transmission cho tay máy, ta viết file launch và Setup trạng thái cho tay máy trong Moveit:

```

<launch>
    <arg name="arg_x" default="0.00" />
    <arg name="arg_y" default="0.00" />
    <arg name="arg_z" default="0.00" />
    <arg name="arg_R" default="0.00" />
    <arg name="arg_P" default="0.00" />
    <arg name="arg_Y" default="0.00" />
    <!-- Khởi động Gazebo -->
    <include file="$(find gazebo_ros)/launch/empty_world.launch">
        <arg name="world_name" value="$(find mo_hinh_robot)/worlds/empty.world"/>
        <arg name="paused" value="false"/>
        <arg name="use_sim_time" value="true"/>
        <arg name="gui" value="true"/>
        <arg name="headless" value="false"/>
        <arg name="debug" value="false"/>
    </include>

    <!-- Load mô tả robot -->
    <param name="robot_description" command="$(find xacro)/xacro --inorder '$(find mo_hinh_robot)/urdf/mo_hinh_robot.urdf'" />

    <!-- Spawn robot vào Gazebo -->
    <node name="urdf_spawner" pkg="gazebo_ros" type="spawn_model" respawn="false" output="screen"
          args="-urdf -model mo_hinh_robot -param robot_description -x $(arg arg_x) -y $(arg arg_y) -z $(arg arg_z) -Y $(arg arg_R)" />

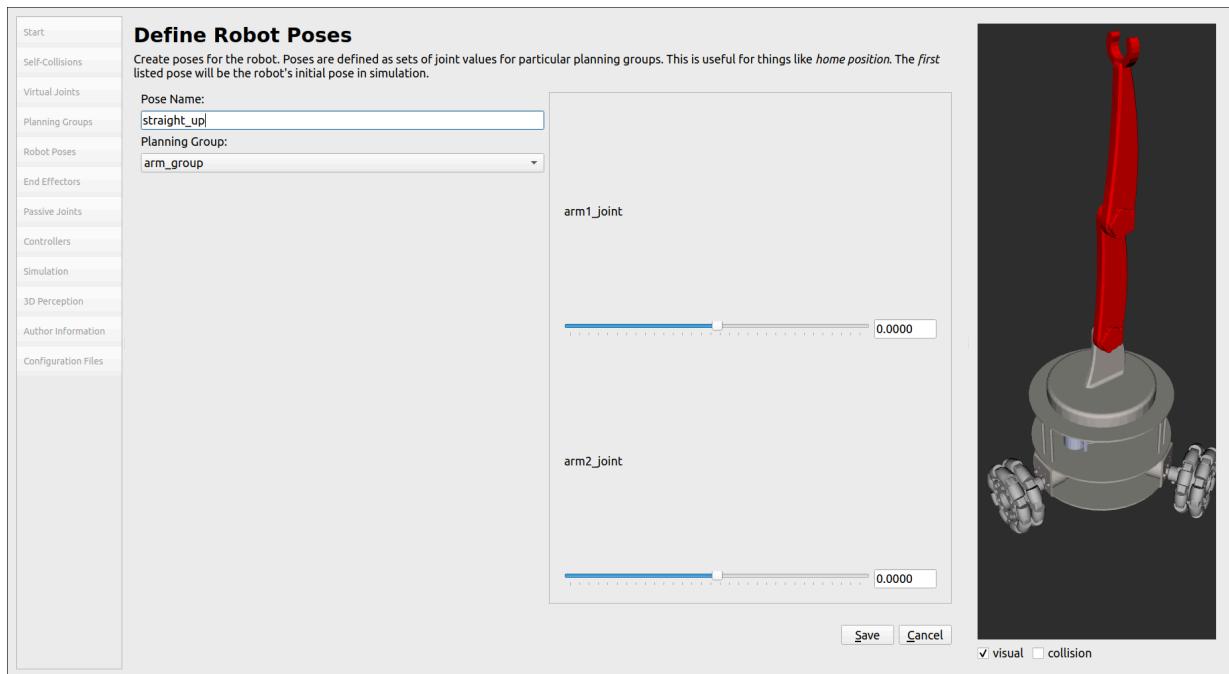
    <!-- Tải thông số joint -->
    <rosparam command="load" file="$(find mo_hinh_robot)/config/joint_trajectory_controller.yaml" />

    <!-- Khởi động bộ điều khiển -->
    <node name="controller_spawner" pkg="controller_manager" type="spawner" respawn="false" output="screen"
          args="joint_state_controller robot_arm_controller"/>

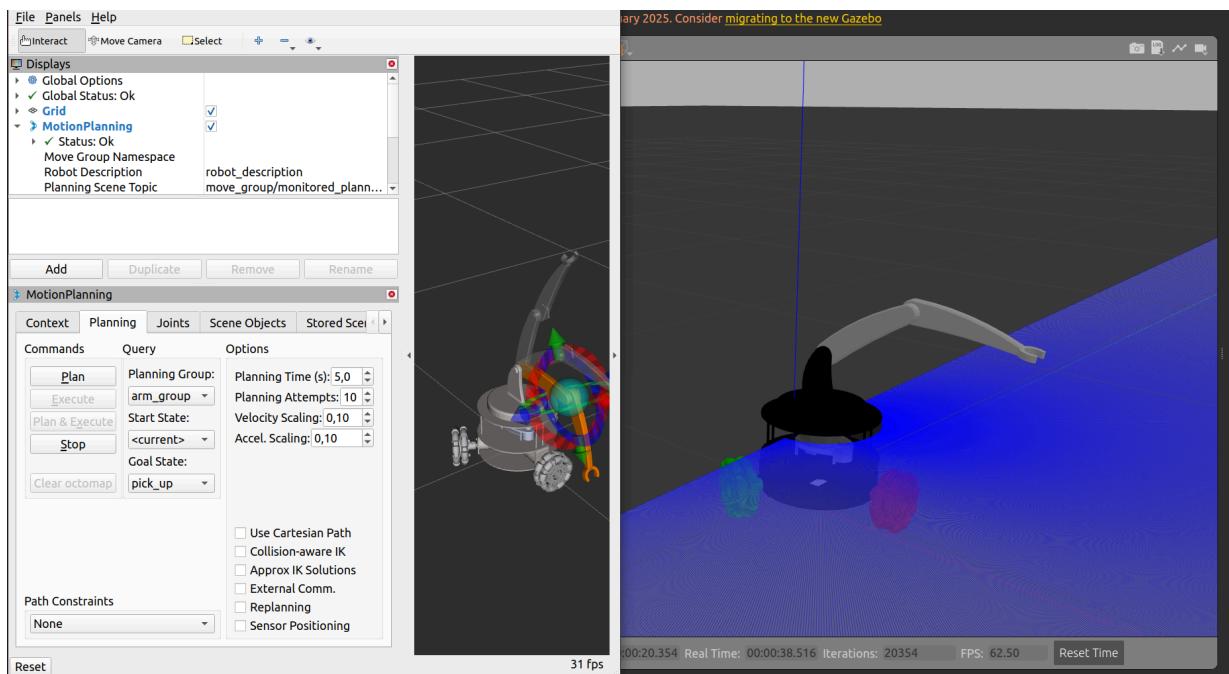
    <!-- Robot State Publisher -->
    <node name="robot_state_publisher" pkg="robot_state_publisher" type="robot_state_publisher" respawn="false" output="screen"/>
</launch>

```

Hình 22. Hàm arm_urdf.launch



Hình 23. Setup Pose trong Moveit



Hình 23. Motion Planning của tay máy

8. Kết luận

File URDF của robot omni 3 bánh mô tả chi tiết cấu trúc, liên kết, cảm biến (LiDAR, IMU) và tích hợp Gazebo, đảm bảo khả năng di chuyển linh hoạt và mô phỏng thực tế. Với các tham số được định nghĩa rõ ràng, file này là nền tảng vững chắc cho thiết kế, thử nghiệm và ứng dụng robot.