ĐẠI HỌC QUỐC GIA HÀ NỘI

TRƯỜNG ĐẠI HỌC CÔNG NGHỆ

KHOA ĐIỆN TỬ - VIỄN THÔNG

BÁO CÁO GIỮA KỲ

Môn Lập Trình Robot Với ROS

ĐIỀU KHIỂN XE ROBOT OMNI 4 BÁNH (MECANUM) KẾT HỢP CAMERA, LIDAR, GPS

Người thực hiện:

Vũ Thị Thu Trang	22027515
QH-2022-I/CQ-R	Khoa Điện tử Viễn thông

Hà Nội, 2024

<u>Yêu cầu</u>:

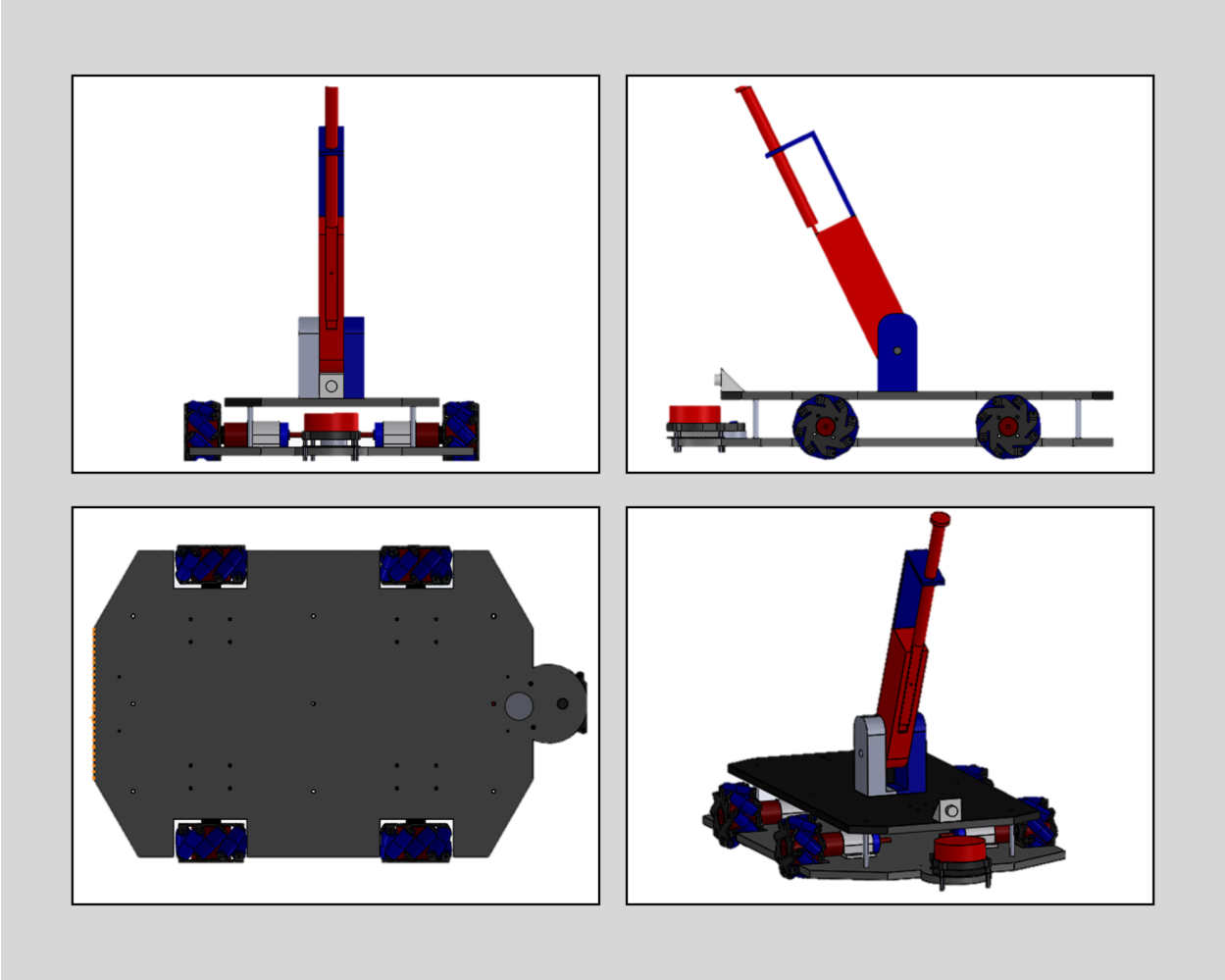
Xây dựng mô hình mô phỏng Robot

- Robot 4 bánh Omni (**Mecanum**)
- Kết hợp 1 tay máy 2 khớp:
    + Khớp 1: **Rotation**
    + Khớp 2: **Prismatic**
- Gắn các cảm biến: **Camera, LIDAR, GPS**

I. Tổng quan về thiết kế Robot

1. Tổng thể thiết kế SOLIDWORKS

Xe robot được thiết kế với hệ thống bánh Omni, cho phép di chuyển linh hoạt theo nhiều hướng mà không cần quay đầu. Xe được trang bị các cảm biến hiện đại như Camera, GPS và LIDAR giúp thu thập dữ liệu môi trường và điều hướng chính xác trong không gian 3D.



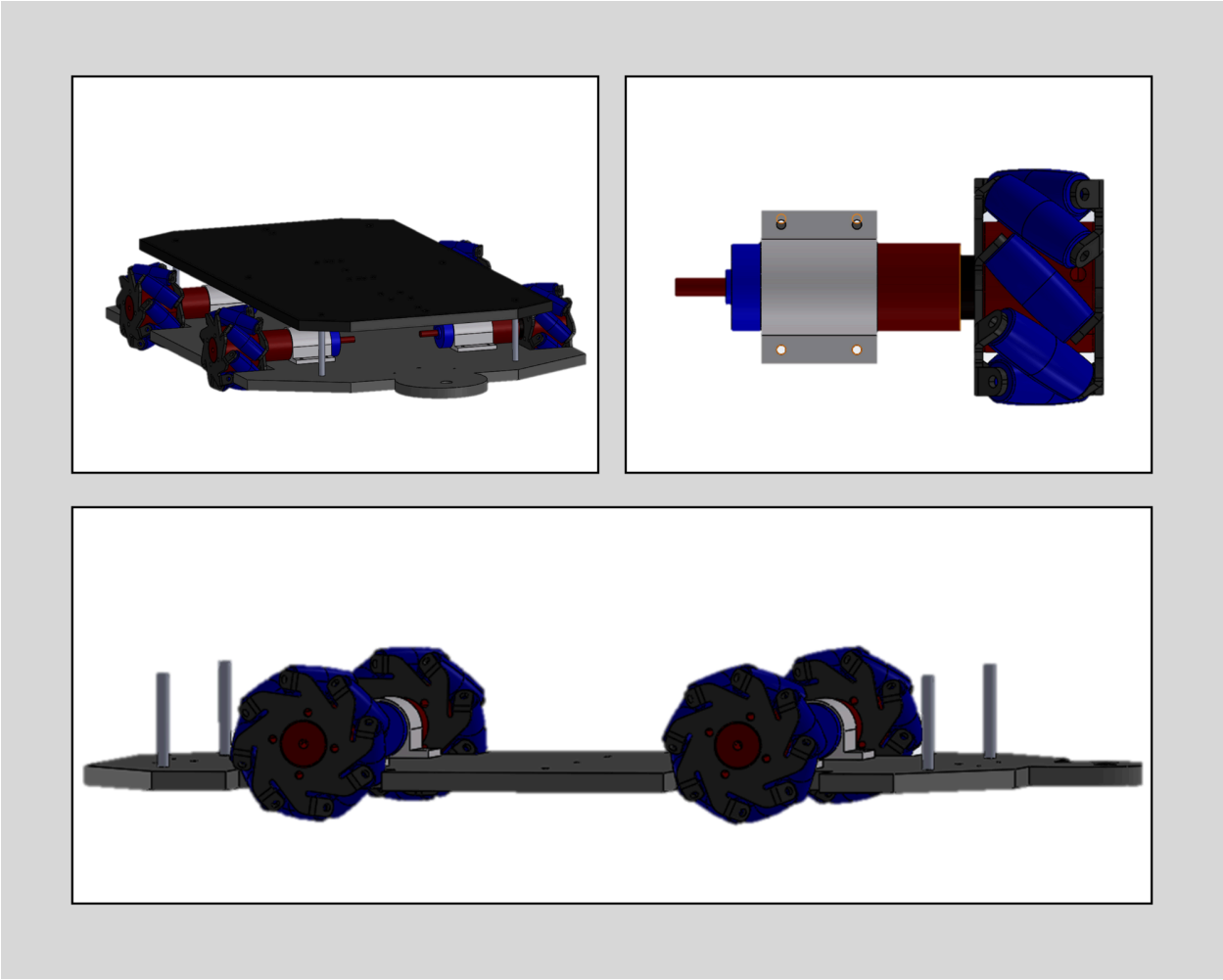
Hình 1.1. Mô hình 3D

2. Chi tiết thiết kế

2.1. Mô hình di chuyển Robot 4 bánh Omni (mecanum)

Hệ thống sử dụng hệ di chuyển Robot gồm có:

- 4 bánh xe Omni gắn với 4 động cơ Servo
- Đế xe 2 tầng để dễ dàng mở rộng mô hình



Hình 2.1. Mô hình 3D Robot di chuyển

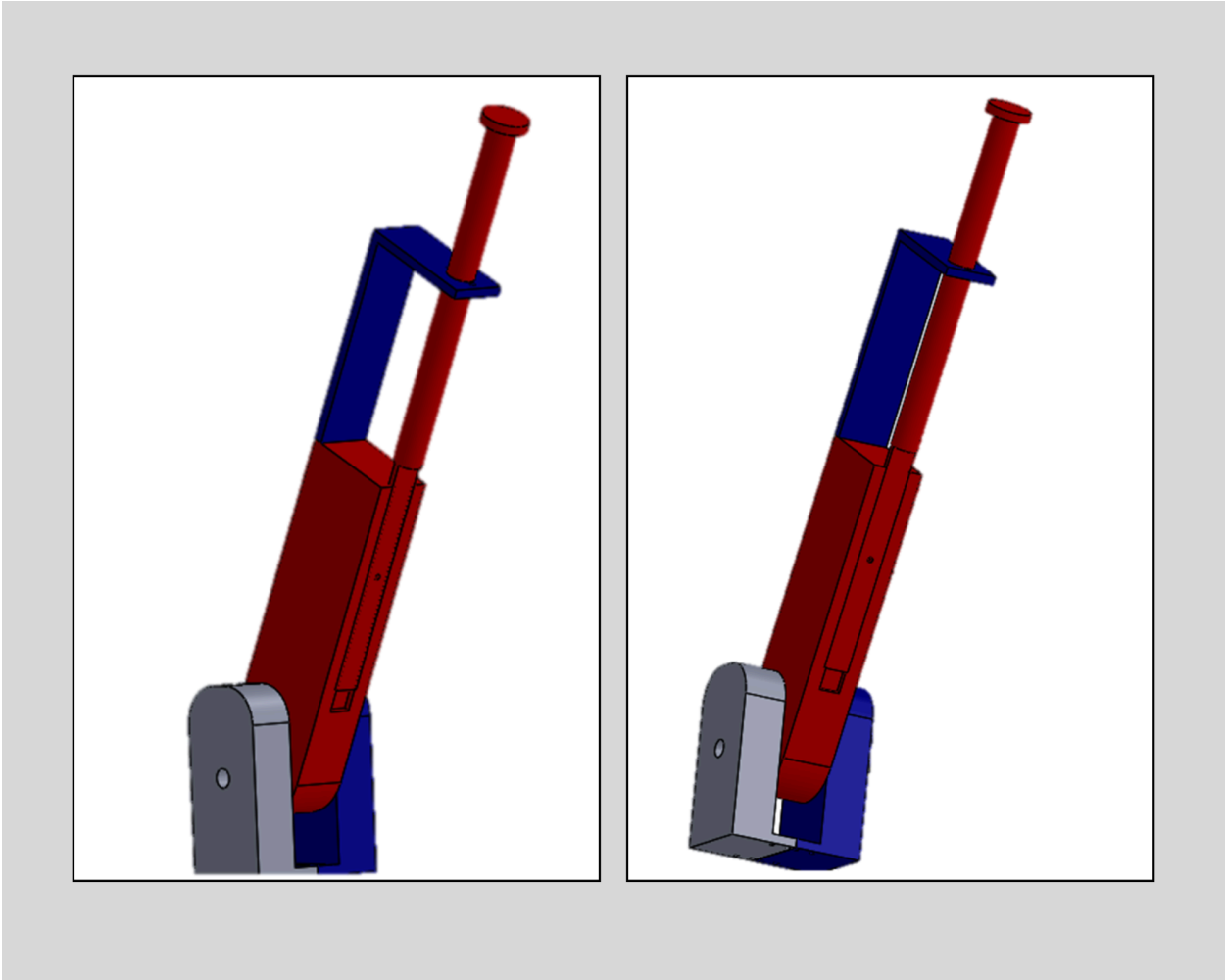
2.2. Mô hình tay máy 2 khớp

Cấu trúc tay máy gồm các phần chính:

- Phần base: gắn cố định vào thân xe, đóng vai trò như trục xoay hoặc bản lề để điều chỉnh góc xoay của tay máy
- Phần khớp:

+ Khớp 1: Khớp quay (revolute joint) cho phép tay máy xoay quanh một trục cố định 180 độ. Kết hợp khớp 1 với hệ thống đỡ trục tịnh tiến của khớp 2,
+ Khớp 2: Khớp tịnh tiến (prismatic joint) có thể tăng độ dài bằng cơ chế trượt, giới hạn trượt là 0,13m. (Đơn giản hoá để dễ mô phỏng. Thực tế sẽ dùng 1 động cơ Servo chuyển từ chuyển động quay sang chuyển động tịnh tiến)

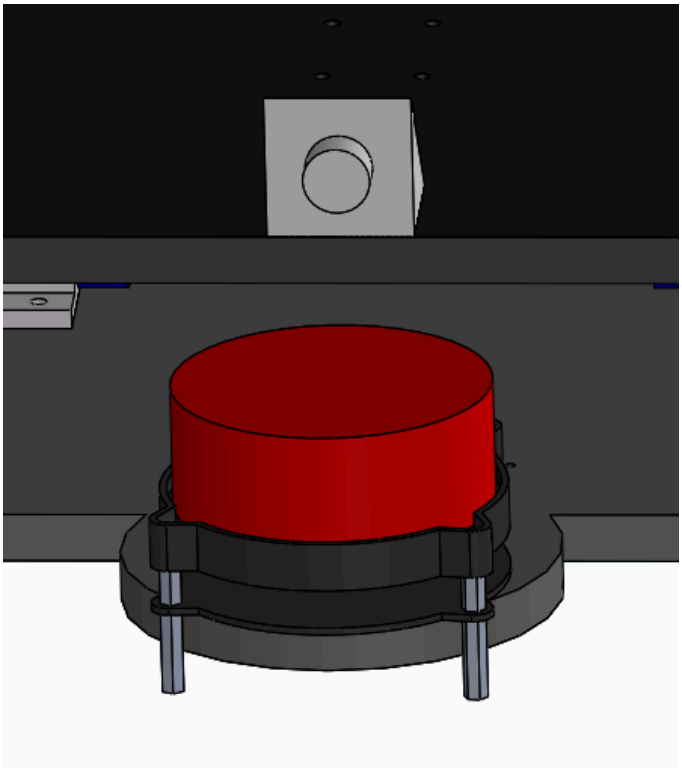
- Phần End-effector: Có thể được trang bị cơ cấu kẹp hoặc các công cụ khác tùy vào nhiệm vụ cụ thể.



Hình 2.2. Mô hình 3D tay máy

2.3. Hệ thống cảm biến

- Camera: gắn ở mặt trên của đế xe để không bị vướng khi tay máy và xe điều khiển, dễ quan sát
- LIDAR: gắn ở phần nhô ra ở đế dưới, có thể quét được 180 độ (chỉ cần từng này để phát hiện được các vật thể trước mặt)
- GPS: gắn giữa thân xe để dễ dàng xác định được toạ độ của xe



Hình 2.3. Hệ cảm biến mở rộng

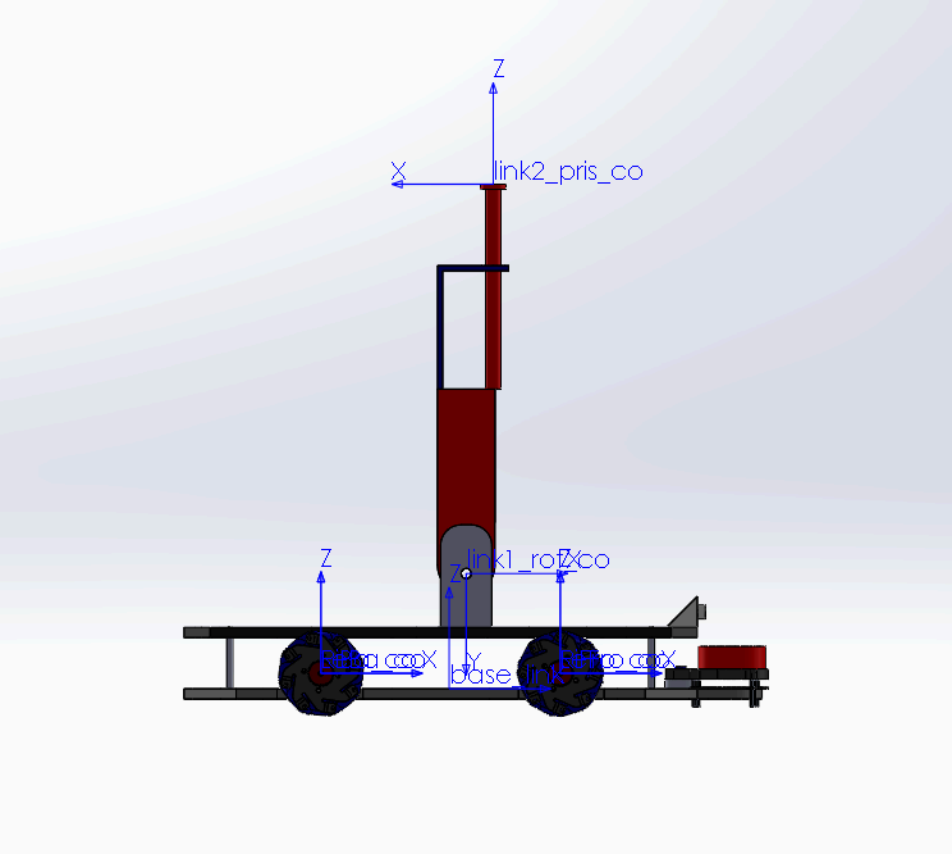
II. Tính toán động học

1. Đặt hệ trục toạ độ

Đặt vị trí ban đầu của Robot khi khớp xoay vuông góc với thân xe và khớp tịnh tiến ở vị trí tịnh tiến 0m. Đặt hệ trục toạ độ của Robot theo quy tắc Denavit-Hartenberg như Hình 1.1.

Hệ gồm có:

- Base (gốc tọa độ)
- Khớp 1 (Revolute Joint - quay quanh trục Z)
- Khớp 2 (Prismatic Joint - tịnh tiến theo trục Z)



Hình 1.1. Đặt hệ trục toạ độ

2. Tính toán động học thuận, ngược của tay máy

Joint	θ_i (rad)	d_i (mm)	a_i (mm)	α_i (rad)
1 (Revolute)	θ_1	d_1	0	$\frac{\pi}{2}$
2 (Prismatic)	0	d_2	0	0

Hình 2.3. Bảng D-H của hệ

Trong đó:

θ_i – *góc quay quanh trục* z_{i-1}

d_i – *khoảng tịnh tiến dọc theo* z_{i-1}

a_i – *khoảng cách giữa* z_{i-1} *và* z_i *dọc theo* x_{i-1}

α_i – *góc giữa* z_{i-1} *và* z_i *quanh* x_{i-1}

a) Động học thuận:

Ma trận biến đổi đồng nhất:

Từ base đến link1:

$$T_1^0 = \begin{bmatrix} \cos\theta_1 & 0 & \sin\theta_1 & 0 \\ \sin\theta_1 & 0 & -\cos\theta_1 & 0 \\ 0 & 1 & 0 & d_1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Từ link1 đến link2:

$$T_2^1 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d_2 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Ma trận tổng:

$$T_2^0 = T_1^0 \cdot T_2^1$$

$$= \begin{bmatrix} \cos\theta_1 & 0 & \sin\theta_1 & d_2\sin\theta_1 \\ \sin\theta_1 & 0 & -\cos\theta_1 & -d_2\cos\theta_1 \\ 0 & 1 & 0 & d_1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Toạ độ của EF trong hệ toạ độ base:

x = d2 sin θ1

y = −d2 cos θ1

z = d1

b) Động học ngược

Cho trước vị trí mong muốn của đầu tay máy (x_d, y_d, z_d) từ phương trình từ động học thuận ta có:

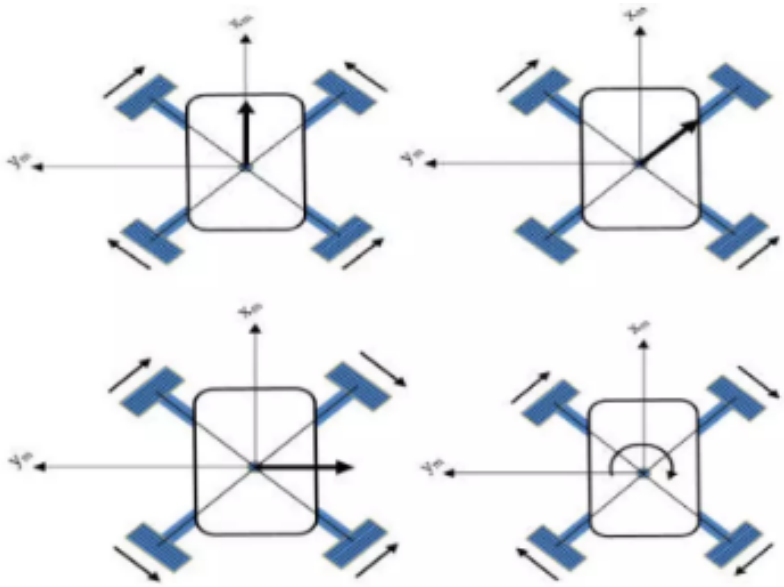
$$\theta_1 = \tan^{-1}\left(\frac{x_d}{-y_d}\right) \qquad d_2 = \sqrt{x_d^2 + y_d^2}$$

3. Tính toán động học thuận, ngược của hệ di chuyển

Mô hình động học mô tả mối quan hệ giữa vận tốc của robot và vận tốc góc của từng bánh xe. Trong FWOMR, các bánh xe Omni thường được bố trí

lệch so với trục tọa độ động một góc 45 độ, và các bánh được đặt cách nhau một góc 90 độ. Điều này cho phép robot di chuyển theo mọi hướng trên mặt phẳng mà không cần quay thân. Bánh xe Omni được thiết kế với các con lăn nhỏ xung quanh, cho phép chuyển động theo cả hai hướng vuông góc. Khi các bánh xe quay với vận tốc và hướng khác nhau, robot có thể thực hiện các chuyển động phức tạp như:

★ **Di chuyển thẳng:** Tất cả các bánh xe quay cùng tốc độ và cùng hướng.
★ **Di chuyển ngang:** Các bánh xe quay theo hướng và tốc độ phù hợp để tạo ra chuyển động ngang.
★ **Quay tại chỗ:** Các bánh xe quay theo cách tạo ra mô-men xoắn quanh trục trung tâm của robot, cho phép robot xoay mà không di chuyển vị trí.



Hình 1.2. Cơ chế chuyển động của omni 4 bánh

a) Động học thuận

Mối quan hệ giữa vận tốc của robot và vận tốc góc của bánh xe được xác định bởi ma trận động học:

$$\begin{bmatrix} V_x \\ V_y \\ \omega \end{bmatrix} = \frac{r}{4} \begin{bmatrix} 1 & 1 & 1 & 1 \\ -1 & 1 & 1 & -1 \\ \frac{1}{R} & -\frac{1}{R} & \frac{1}{R} & -\frac{1}{R} \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ w_3 \\ w_4 \end{bmatrix}$$

b) Động học ngược

Để tìm vận tốc góc của bánh xe từ vận tốc của robot, ta lấy nghịch đảo ma trận:

$$\begin{bmatrix} w_1 \\ w_2 \\ w_3 \\ w_4 \end{bmatrix} = \frac{1}{r} \begin{bmatrix} 1 & -1 & R \\ 1 & 1 & -R \\ 1 & 1 & R \\ 1 & -1 & -R \end{bmatrix} \begin{bmatrix} V_x \\ V_y \\ \omega \end{bmatrix}$$

III. Mô tả dự án

1. Mô tả file URDF

Mô hình được xây dựng với base_link làm khung gầm chính, trên đó gắn các bánh xe omni và các bộ phận khác.

1.1. Khung gầm (Base Link)

★ Tên: base_link
★ Khối lượng: 3.41696 kg
★ Quán tính: Được định nghĩa trong phần <inertial>
★ Hình dạng: Sử dụng tệp STL (base_link.STL) để mô tả hình dạng trực quan.
★ Màu sắc: Xanh pastel (0.7 0.9 1 1).

1.2. Hệ thống bánh xe omni

Robot có 4 bánh xe với các joint kiểu "continuous":

★ **LeFro (Bánh trước trái)**
- Vị trí: (0.111, 0.181, 0.015) m
- Trục quay: trục Y âm

★ **LeBa (Bánh sau trái)**
- Vị trí: (-0.125, 0.181, 0.015) m
- Trục quay: trục Y âm

★ **RiFro (Bánh trước phải)**
- Vị trí: (0.109, -0.180, 0.015) m

- Trục quay: trục Y dương

★ **RiBa (Bánh sau phải)**

- Vị trí: (-0.127, -0.181, 0.015) m
- Trục quay: trục Y dương

Các bánh xe này được gắn vào base_link thông qua các khớp quay (continuous joint), cho phép di chuyển linh hoạt theo các hướng khác nhau.

1.3. Cánh tay robot

Gồm 2 khớp:

★ **Khớp quay (link1_rot)**

- Kiểu: revolute
- Giới hạn: -1.57 đến 1.57 radian (-90° đến 90°)
- Vị trí: (0.017, 0, 0.113) m
- Trục quay: trục Z âm

★ **Khớp tịnh tiến (link2_pris)**

- Kiểu: prismatic
- Giới hạn: -0.5 đến 0.5 m
- Trục chuyển động: trục Z âm

1.4. Các cảm biến

a. Camera

★ Tên link: camera_link

★ Vị trí: (0.228, 0.001, 0.075) m

★ Cấu hình trong Gazebo:

- Độ phân giải: 640x480
- FOV: 1.396 rad (~80°)
- Tần suất: 30Hz
- Topic: /front_camera/image_raw

b. RPLIDAR

★ Tên link: laser

★ Vị trí: (0.25, 0.001, 0.005) m

★ Cấu hình trong Gazebo:

- Góc quét: -90° đến 90°
- Khoảng cách: 0.1-12m
- Tần suất: 30Hz

- Topic: /scan

c. GPS

★ Tên link: gps_link
★ Vị trí: (0, 0, 0.04) m
★ Cấu hình trong Gazebo:
- Tần suất: 10Hz
- Topic vị trí: /gps/fix
- Topic vận tốc: /gps/fix_velocity
- Tọa độ tham chiếu: Hà Nội (21.028511, 105.804817)

1.5. Cấu hình Gazebo

★ Điều khiển bánh xe
- Plugin: libgazebo_ros_diff_drive.so
- Topic điều khiển: /cmd_vel
- Topic odometry: /odom
- Khoảng cách giữa bánh xe: 0.3623m
- Đường kính bánh xe: 0.1m

★ Điều khiển cánh tay
- Sử dụng PositionJointInterface
- Có transmission cho từng khớp

★ ROS Control
- Plugin: libgazebo_ros_control.so

2. Mô tả cơ chế điều khiển trên Gazebo

File URDF (myomnicar.urdf) định nghĩa cấu trúc vật lý và động học của robot, bao gồm:

➔ Các link: Đại diện cho các thành phần vật lý của robot (ví dụ: thân robot, bánh xe).
➔ Các joint: Kết nối các link và cho phép chuyển động (ví dụ: quay bánh xe).
➔ Plugin Gazebo: Tích hợp các thành phần vật lý với Gazebo để mô phỏng.

- **Nhận lệnh vận tốc:**
  - Người dùng gửi lệnh vận tốc đến topic /cmd_vel (ví dụ: từ bàn phím hoặc joystick).
  - Node omni_keyboard_control.py chuyển đổi lệnh vận tốc thành vận tốc bánh xe.

- **Điều khiển bánh xe:**
    - Vận tốc bánh xe được gửi đến các controller thông qua các topic /LeFro_joint_velocity_controller/command, v.v.
    - Plugin diff_drive_controller trong Gazebo mô phỏng chuyển động của robot dựa trên vận tốc bánh xe.
- **Điều khiển tay máy:**
    - Node arm_controller.py sử dụng giao thức **FollowJointTrajectoryAction** để gửi lệnh đến controller của tay máy.
        - joint1: Điều khiển khớp quay (radian).
        - joint2: Điều khiển khớp tịnh tiến (mét).
    - arm_controller.py nhận lệnh từ bàn phím và gửi lệnh điều khiển đến các controller của tay máy.
- **Xuất dữ liệu odometry:**
    - Plugin diff_drive_controller mô phỏng chuyển động của robot dựa trên vận tốc bánh xe.
    - Plugin gazebo_ros_control mô phỏng chuyển động của tay máy dựa trên lệnh điều khiển từ các controller.
- **Hiển thị trên RViz:**
    - RViz hiển thị trạng thái của robot và tay máy, bao gồm:
        - Dữ liệu odometry của robot.
        - Vị trí và chuyển động của tay máy.
- **Chuyển đổi GPS sang kiểu Odometry:**
    - Nhận dữ liệu GPS:
        - Node navsat_transform_node nhận dữ liệu từ topic /gps/fix (chuẩn ROS message sensor_msgs/NavSatFix).
        - Dữ liệu GPS bao gồm: latitude, longitude, altitude.
    - Nhận dữ liệu IMU (nếu có):
        - Node cũng có thể nhận dữ liệu từ IMU (Inertial Measurement Unit) qua topic /imu/data để cải thiện độ chính xác.
    - Chuyển đổi sang odometry:
        - Node sử dụng thông tin GPS và IMU để tính toán vị trí và vận tốc trong hệ tọa độ Cartesian (x, y, z).
        - Dữ liệu odometry được xuất ra topic /odometry/gps

IV. Mô tả cấu trúc Package dự án

Cấu trúc dự án

```
├── CMakeLists.txt
├── config
│   ├── arm_control.yaml
│   ├── configfinal.rviz
│   ├── config_sensors.rviz
│   ├── controllers.yaml
│   └── joint_names_myomnicar.yaml
├── export.log
├── launch
│   ├── display.launch
│   └── gazebo.launch
├── meshes
│   ├── base_link.STL
│   ├── LeBa_Link.STL
│   ├── LeFro_Link.STL
│   ├── link1_rot_Link.STL
│   ├── link2_pris_Link.STL
│   ├── RiBa_Link.STL
│   └── RiFro_Link.STL
├── package.xml
├── scripts
│   ├── arm_controller.py
│   ├── omni_keyboard_control.py
│   └── sensor_srv.py
├── textures
└── urdf
        ├── myomnicar.csv
        └── myomnicar.urdf
```

a. URDF

- Mục đích: Mô tả cấu trúc vật lý và động học của robot, bao gồm các link, joint, và plugin tích hợp với Gazebo.
- Các thành phần chính:
- Link: Đại diện cho các bộ phận vật lý của robot (ví dụ: thân robot, bánh xe, tay máy).

- Joint: Kết nối các link và cho phép chuyển động (quay hoặc tịnh tiến).
- Plugin:
    - diff_drive_controller: Điều khiển chuyển động của robot dựa trên vận tốc bánh xe.
    - gazebo_ros_control: Tích hợp các controller ROS với Gazebo.

b. File cấu hình controller (arm_control.yaml và controllers.yaml)

- Mục đích: Định nghĩa các controller ROS để điều khiển các joint của robot.
- Các thành phần chính:
- Controller cho bánh xe:
    - LeFro_joint_velocity_controller: Điều khiển vận tốc bánh trước trái.
    - RiFro_joint_velocity_controller: Điều khiển vận tốc bánh trước phải.
- Controller cho tay máy:
    - joint1_position_controller: Điều khiển vị trí khớp quay.
    - joint2_position_controller: Điều khiển vị trí khớp tịnh tiến.
- PID: Các tham số PID để điều chỉnh phản hồi của controller.

c. Node Python (omni_keyboard_control.py)

- Mục đích: Nhận lệnh vận tốc từ người dùng (qua topic /cmd_vel) và chuyển đổi thành vận tốc bánh xe.
- Các thành phần chính:
    - Hàm twist_to_wheel_speeds: Chuyển đổi lệnh vận tốc tuyến tính và góc (Twist) thành vận tốc bánh xe.
    - Publisher: Gửi vận tốc bánh xe đến các topic controller (/LeFro_joint_velocity_controller/command, v.v.).
    - Subscriber: Nhận lệnh vận tốc từ topic /cmd_vel.

d. Node Python (arm_controller.py)

- Mục đích: Điều khiển tay máy (robot arm) thông qua giao thức FollowJointTrajectoryAction.
- Các thành phần chính:
    - Action Client: Gửi lệnh điều khiển đến controller của tay máy (/arm_controller/follow_joint_trajectory).
    - Hàm move_arm: Tạo và gửi lệnh điều khiển vị trí cho các joint của tay máy.

- Giới hạn joint: Đảm bảo các giá trị điều khiển nằm trong phạm vi cho phép.

e. Node Python (sensor_srv.py)

- Mục đích: Tạo service gọi callback của các cảm biến
- Các thành phần chính: các hàm callback

f. File launch (Display.launch)

- Mục đích: Khởi chạy toàn bộ hệ thống, bao gồm:
  - Tải mô hình robot từ URDF.
  - Tải cấu hình controller.
  - Khởi chạy các node điều khiển robot và tay máy.
  - Khởi chạy Gazebo và RViz để mô phỏng và hiển thị.
- Các thành phần chính:
  - Node navsat_transform_node:
    - Chuyển đổi dữ liệu GPS thành odometry.
  - Node controller_spawner:
    - Khởi chạy các controller cho bánh xe và tay máy.
  - Node điều khiển:
    - omni_keyboard_control.py: Điều khiển robot di chuyển.
    - arm_controller.py: Điều khiển tay máy.

g. Plugin navsat_transform_node

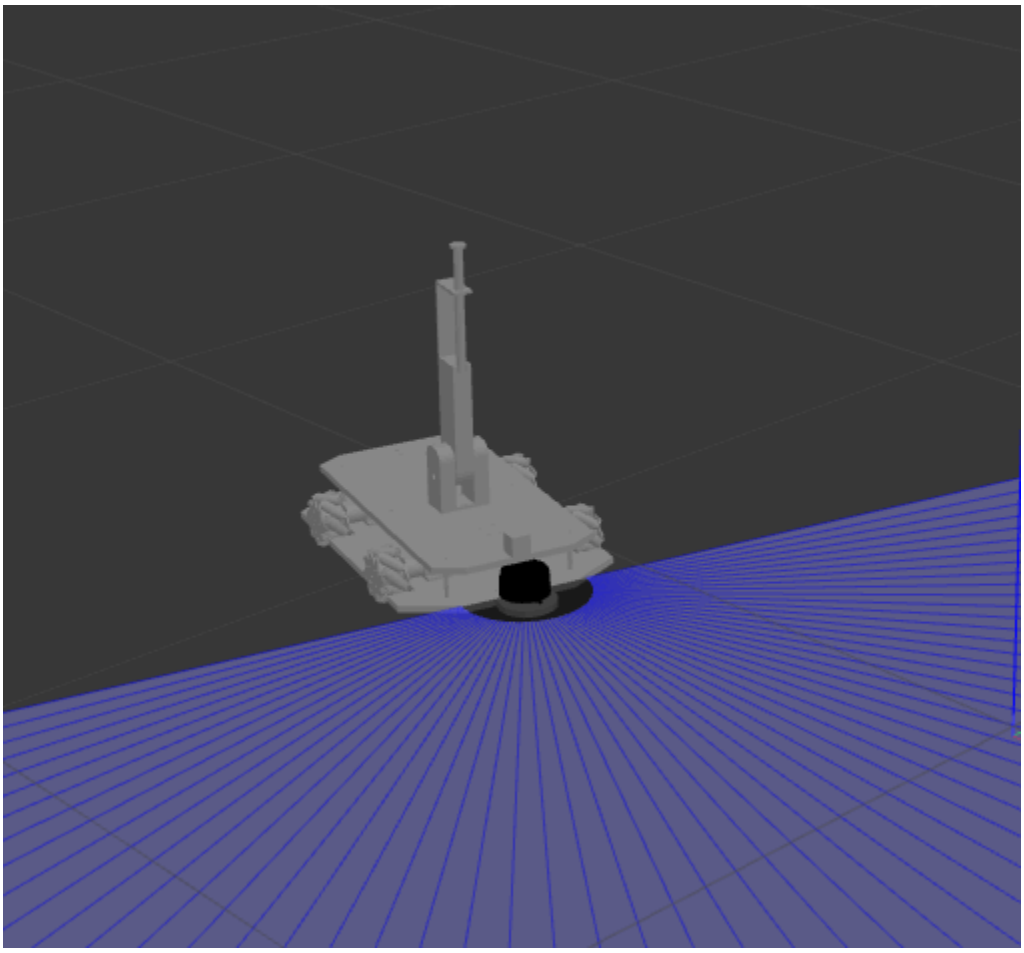
- Mục đích: Chuyển đổi dữ liệu GPS thành dữ liệu odometry.
- Các thành phần chính:
  - Input:
    - Dữ liệu GPS từ topic /gps/fix.
    - Dữ liệu IMU từ topic /imu/data (nếu có).
  - Output:
    - Dữ liệu odometry trên topic /odometry/gps.

h. File Config (config_sensors.rviz): file rviz đã tích hợp các cảm biến và kết nối với topic của từng loạioại

V. Kết quả chạy mô phỏng, các lỗi gặp phải và cách xử lý

A. Kết quả chạy:

- Mô phỏng được điều khiển tay máy khá mượt mà trong Gazebo
- Mô phỏng được điều khiển bánh xe chạy

- Mô phỏng các cảm biến khá ổn, Camera và LiDAR ra tín hiệu chuẩn
- Gọi được các hàm service lấy callback của cảm biến
- 

B. Lỗi và cách giải quyết:

- Lúc đầu em chạy code bị lỗi không nhận được joint, có topic nhưng tay máy không quay. **Cách giải quyết:** bị thiếu tag trong .urdf
- Vì chạy nhiều node cùng lúc nên bị quá tải, khi chạy in topic của gps/fix thì hiện là node chưa được published. **Cách giải quyết:** Thử tải docker để kết nối nhiều máy với nhau, pub được nhiều node nhưng không hiệu quả. Chỉ khi pub thủ công ở terminal thì mới hiện giá trị đọc được
- Tay máy khớp xoay ổn nhưng lúc đầu khớp tịnh tiến không chạy. **Cách giải quyết:** thay đổi limit dài hơn

- Lỗi khi chạy gazebo và rviz bị cái trước cái sau và phải gọi node nhiều lần. **Cách giải quyết**: Em viết gộp chạy các node cũng như chạy gazebo rviz vào file display.launch.
- Rviz chạy không mở được cái config em đã find (chưa fix được nên phải mở tay)
- Có lỗi dummy_link inertial nhưng em không fix vì lúc fix dummy vào thấy pkg chạy bị lỗi nhiều hơn. Chỉ bị Warn thôi nên có thể không cần.
- Xe lúc đầu chạy rất mượt nhưng khi cho hàm omni vào thì 2 bánh sau không thấy quay khiến xe đi vòng vòng.
- Chạy trong Gazebo rất mượt nhưng khi vào Rviz thì tay máy bị giật giật có thể do PID nhưng em đã thử nhưng vẫn giật.