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:
Hoàng Kim Trường - 22027530

Hà Nội – 2025

1. Giới thiệu

1.1. Mục tiêu dự án

Dự án thiết kế một robot omni 4 bánh có khả năng di chuyển linh hoạt trong môi trường giả lập Gazebo. Robot cần được tích hợp các cảm biến như camera, IMU, encoder và có hệ thống tay máy để thực hiện các nhiệm vụ mô phỏng thao tác vật thể. Ngoài ra, robot phải có khả năng điều khiển từ xa thông qua bàn phím hoặc giao diện ROS.

1.2. Mô tả robot omni 4 bánh

Robot omni 4 bánh là một loại robot sử dụng bốn bánh xe đa hướng (omni wheels), cho phép di chuyển theo bất kỳ hướng nào. Điều này giúp robot có khả năng di chuyển linh hoạt hơn so với các loại robot bánh thông thường. Cấu trúc chính của robot bao gồm:

- **Thân chính**: Khung gầm chứa toàn bộ hệ thống điện tử, cảm biến và bộ điều khiển.
- **Bốn bánh omni**: Gắn ở bốn góc của khung, giúp robot di chuyển theo nhiều hướng.
- **Động cơ**: Điều khiển tốc độ và hướng quay của từng bánh xe để tạo ra chuyển động mong muốn.
- **Cảm biến**: Bao gồm camera, IMU và encoder để thu thập dữ liệu về môi trường và trạng thái chuyển động của robot.
- **Tay máy**: Tay máy 2 khớp nối có thể thực hiện các thao tác như gắp và di chuyển vật thể.

2. Thiết kế Robot

2.1. Mô tả thiết kế Robot

Robot được thiết kế với hệ thống 4 bánh omni, cho phép di chuyển linh hoạt theo nhiều hướng khác nhau mà không cần quay đầu. Kết cấu chính của robot bao gồm:

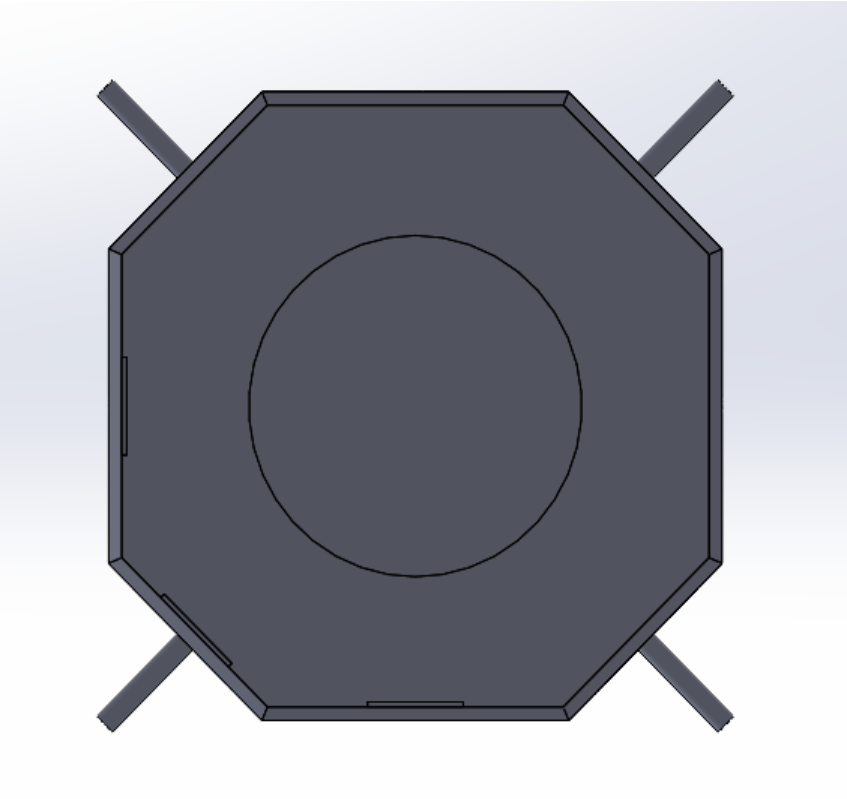
- **Thân chính**: Là phần khung chứa toàn bộ hệ thống và các cảm biến.

- **Bánh xe omni**: Cho phép di chuyển theo các phương ngang, dọc và chéo mà không cần quay toàn bộ thân robot.
- **Camera**: Hỗ trợ quan sát, nhận diện đối tượng và theo dõi môi trường xung quanh.
- **IMU (Inertial Measurement Unit)**: Cảm biến đo gia tốc và vận tốc góc, giúp robot xác định trạng thái chuyển động của mình
- **Encoder**: Được gắn vào động cơ của bánh xe để đo tốc độ quay, hỗ trợ điều khiển chính xác hơn.

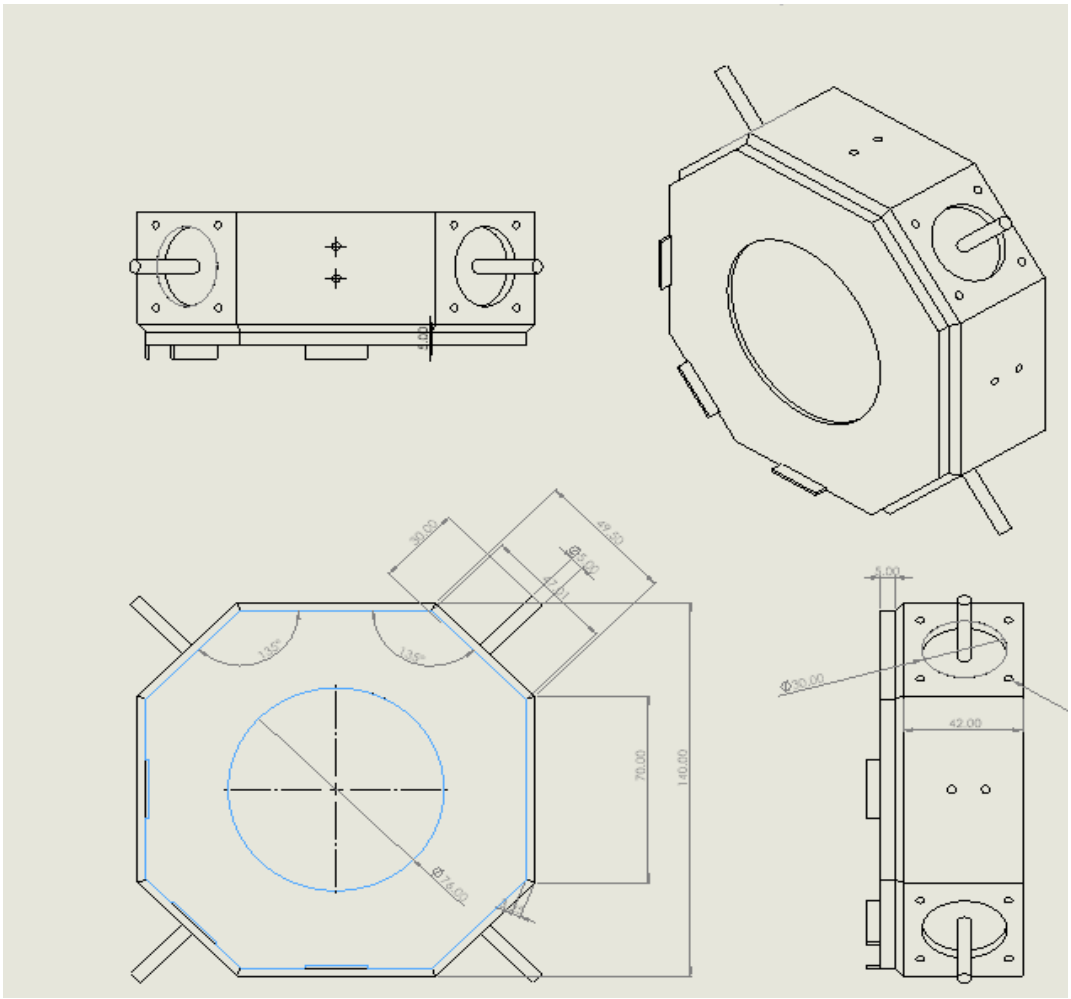
- **Tay máy**: Hệ thống tay máy có hai khớp nối (2 link), có thể thao tác

2.2. Thiết kế SolidWorks

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

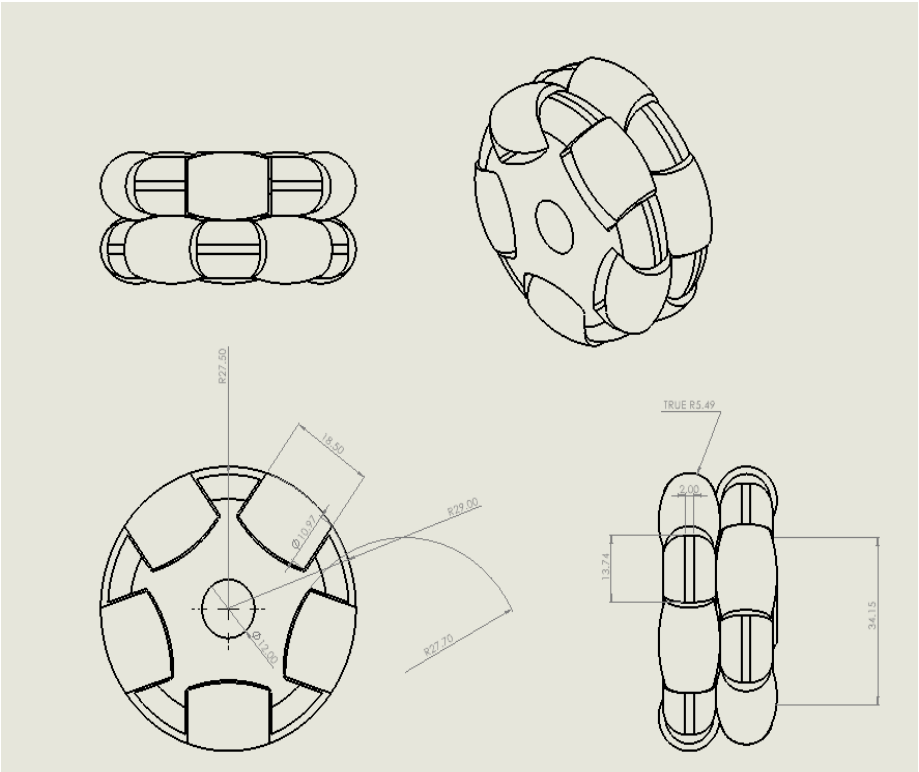


Hình 2.1 Thân chính



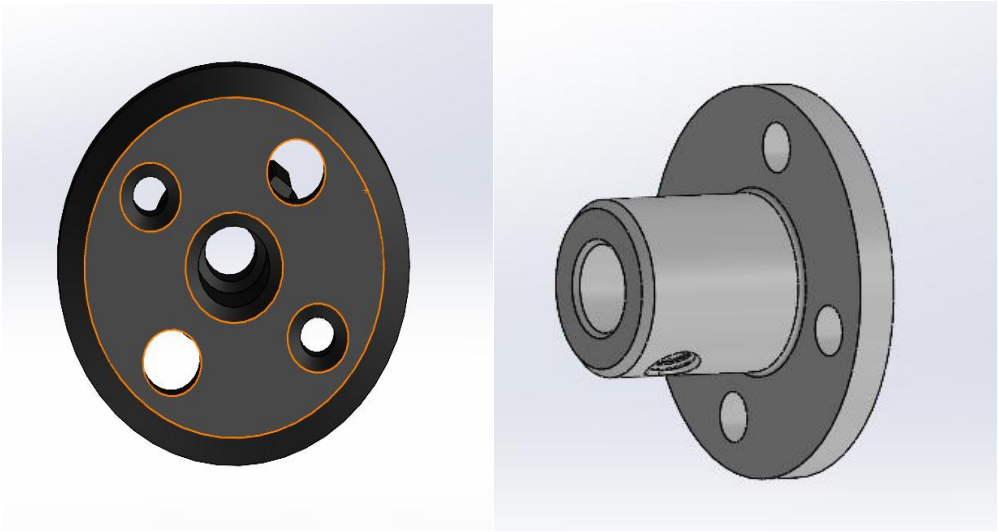
Hình 2.2 Thông số kích thước thân chính

Hình 2.3 Bánh Omni

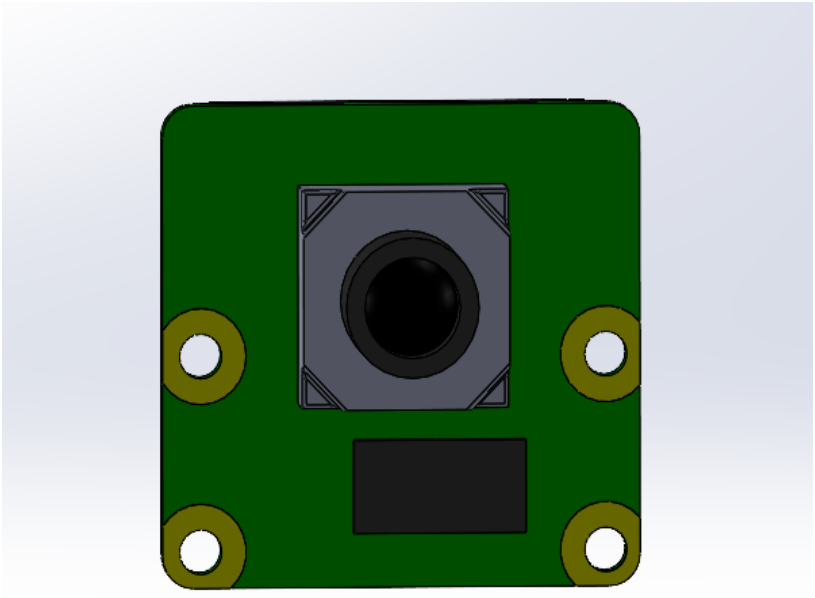


Hình 2.4 Thông số bản vẽ bánh Omni

Các thành phần đi kèm với bánh Omni

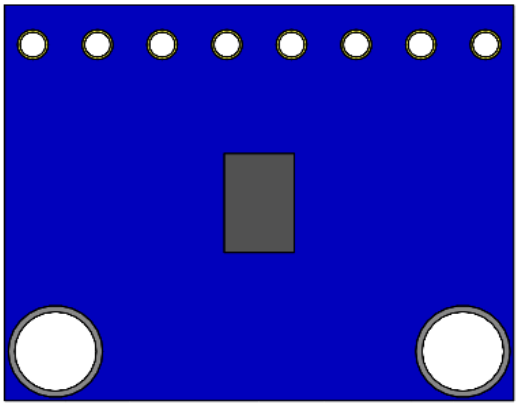


Hình 2.5 Wheel HUB

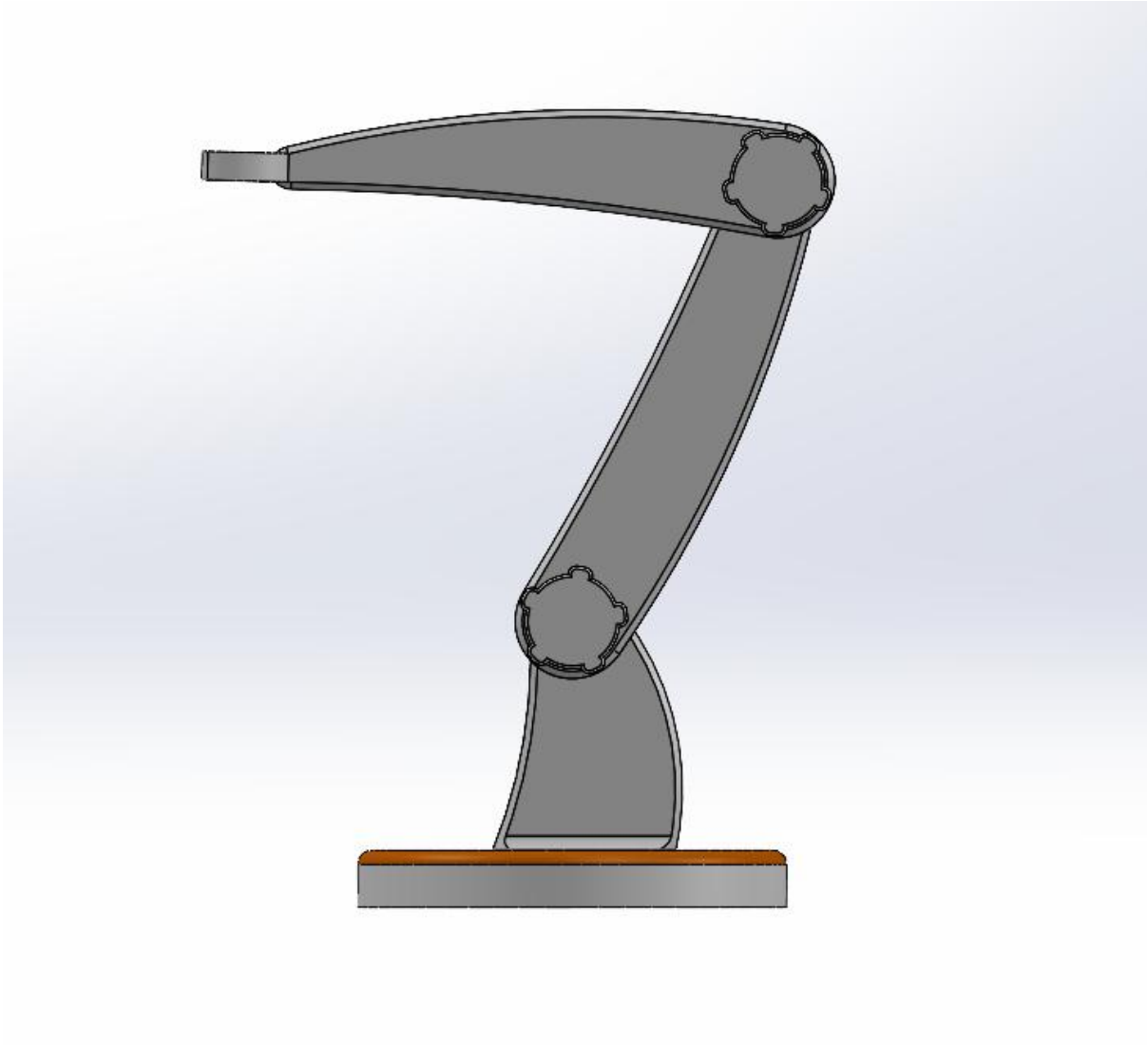


Hình 2.6 Camera

.

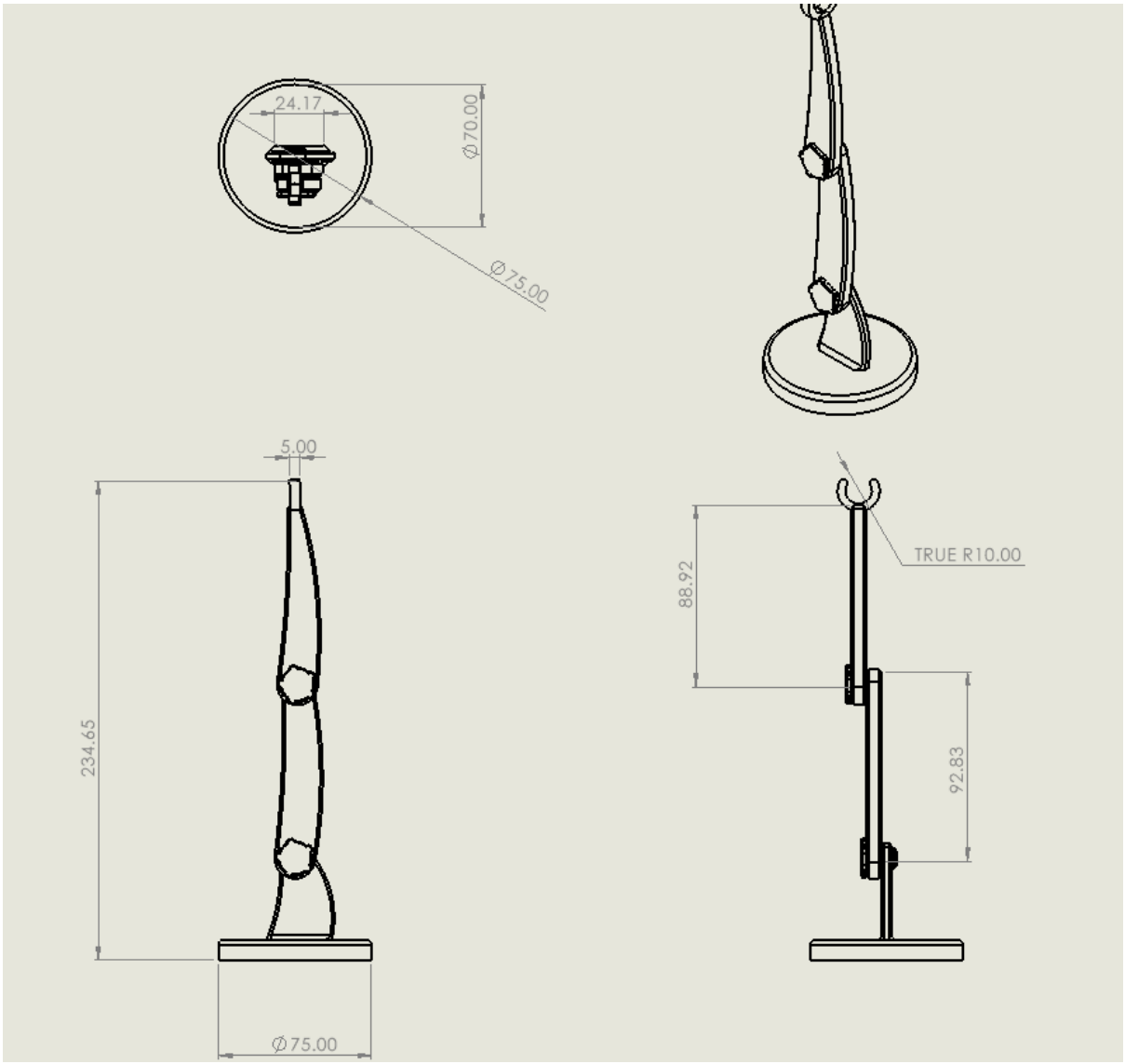


Hình 2.7 IMU

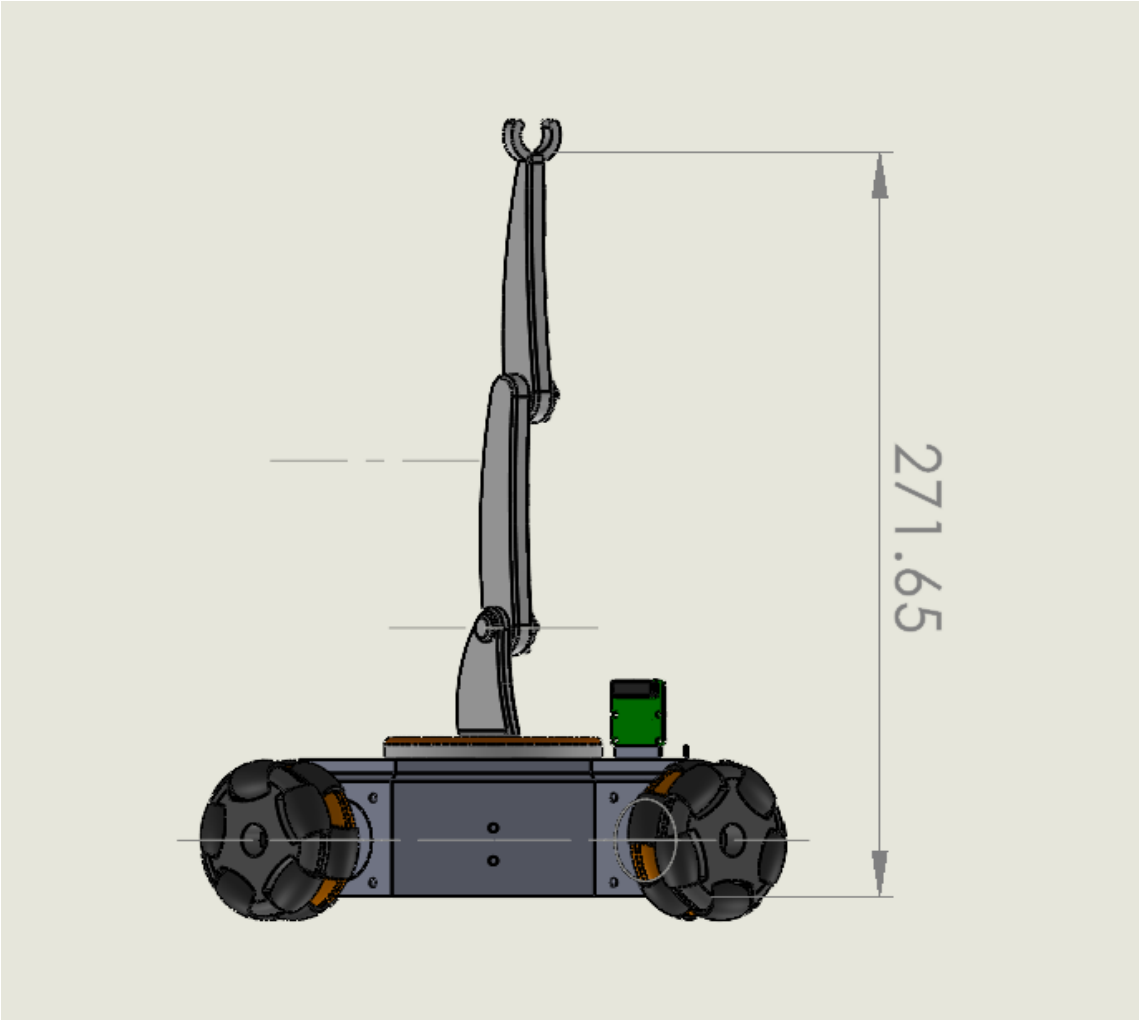


Hình 2.8 Tay máy

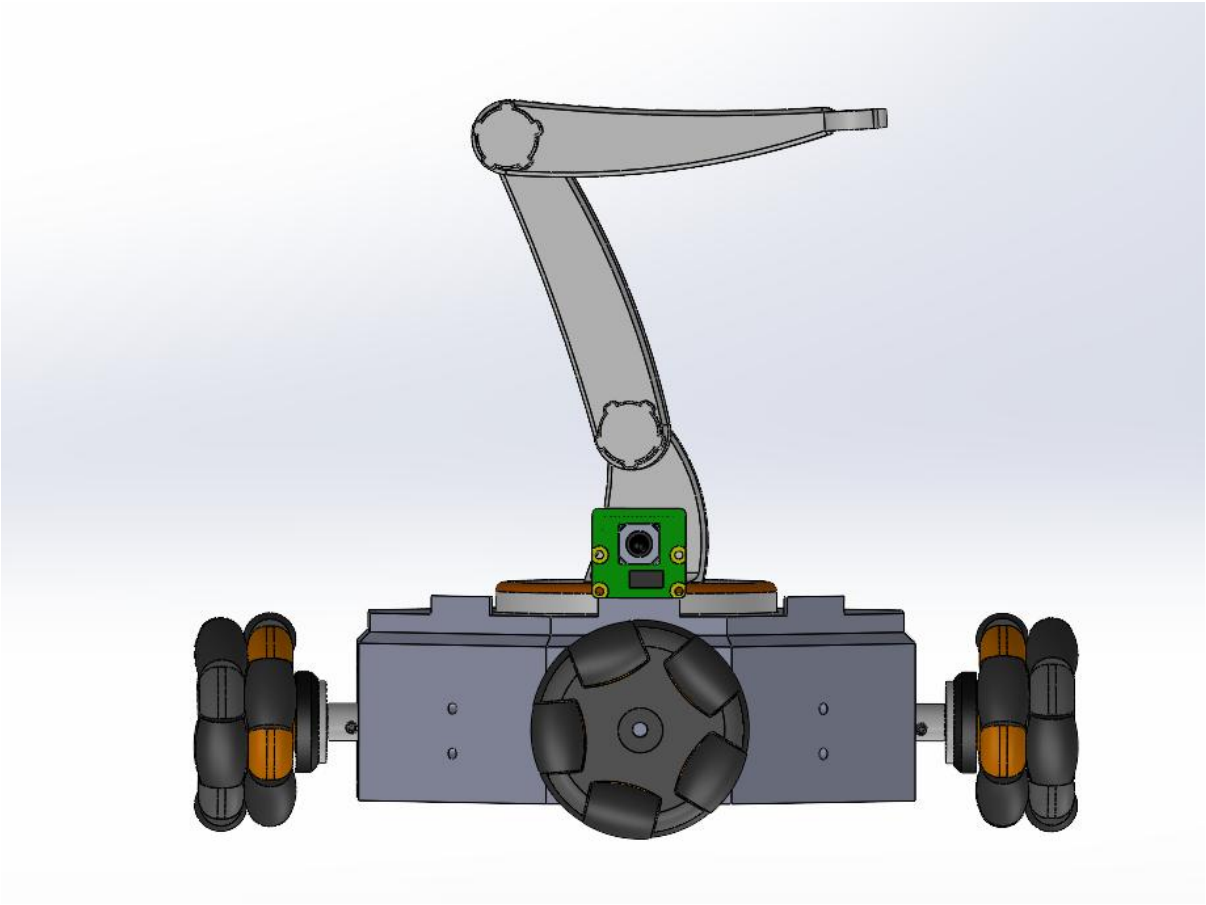
Ở đây , khớp thứ nhất được tính là cố định, chỉ dùng khớp thứ hai và thứ 3 để di chuyển



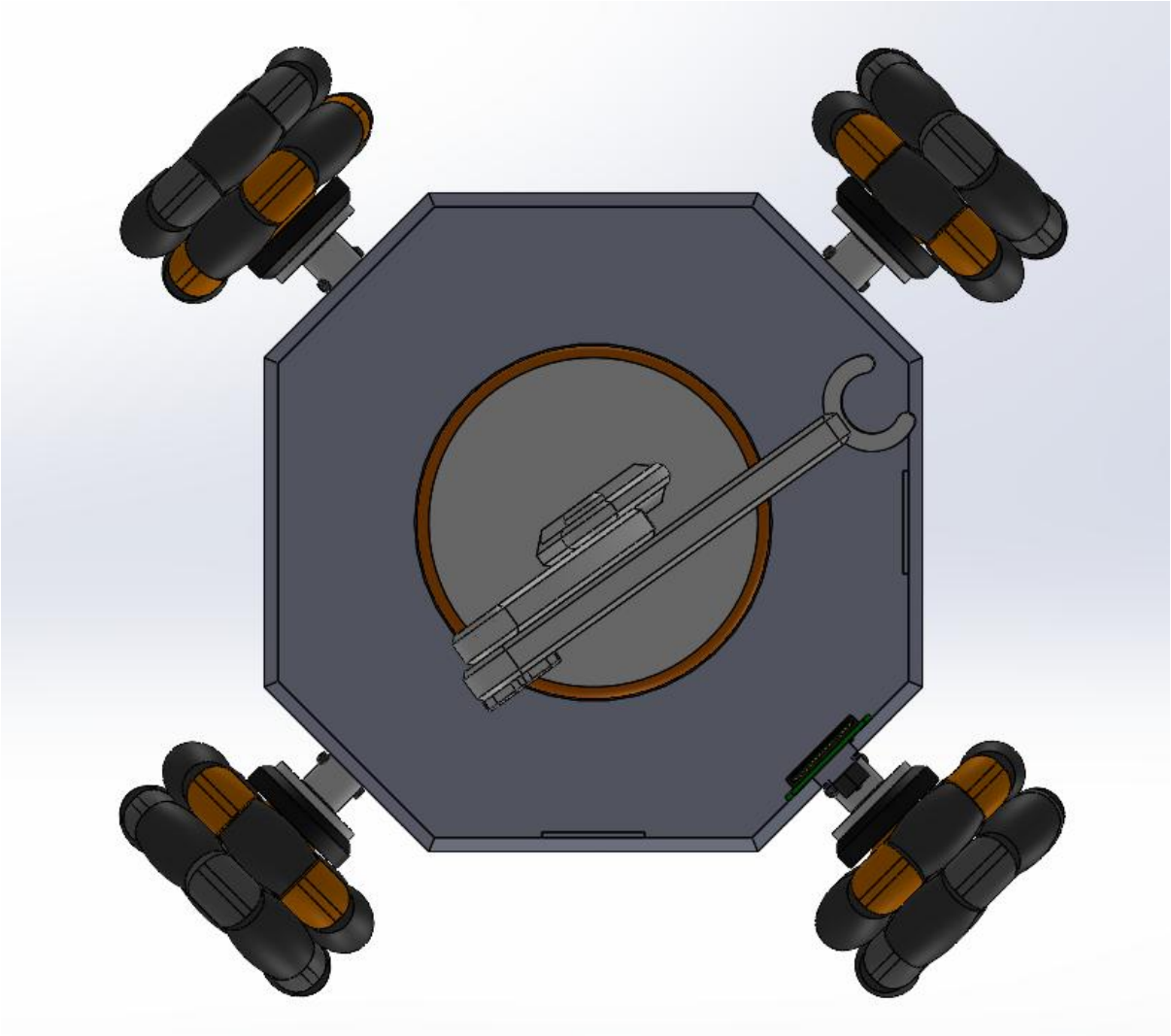
Hình 2.9 Thông số kỹ thuật tay máy



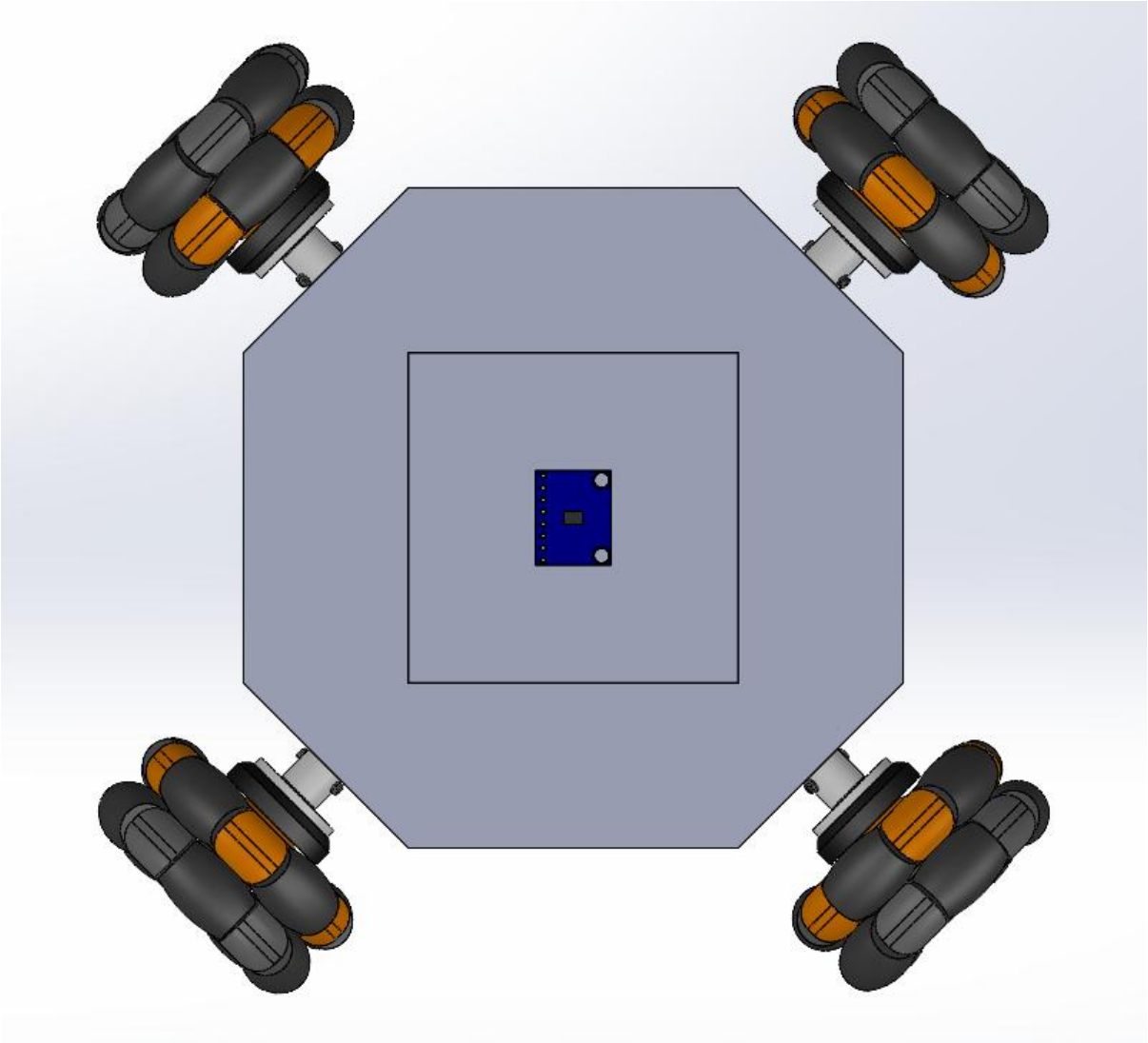
Hình 2.10 Mô Hình Robot hoàn chỉnh và chiều cao



Hình 2.11 Mặt trước robot nếu đi thẳng



Hình 2.12 Phía trên nhìn xuống



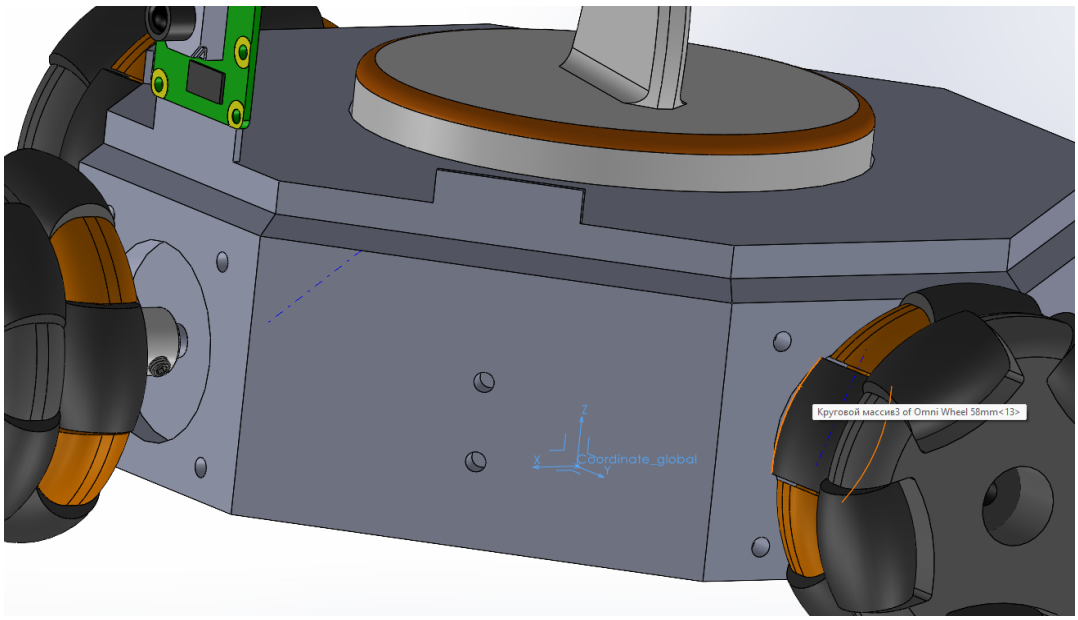
Hình 2.13 Mặt dưới Robot

3. Mô tả URDF và mô phỏng trong Gazebo

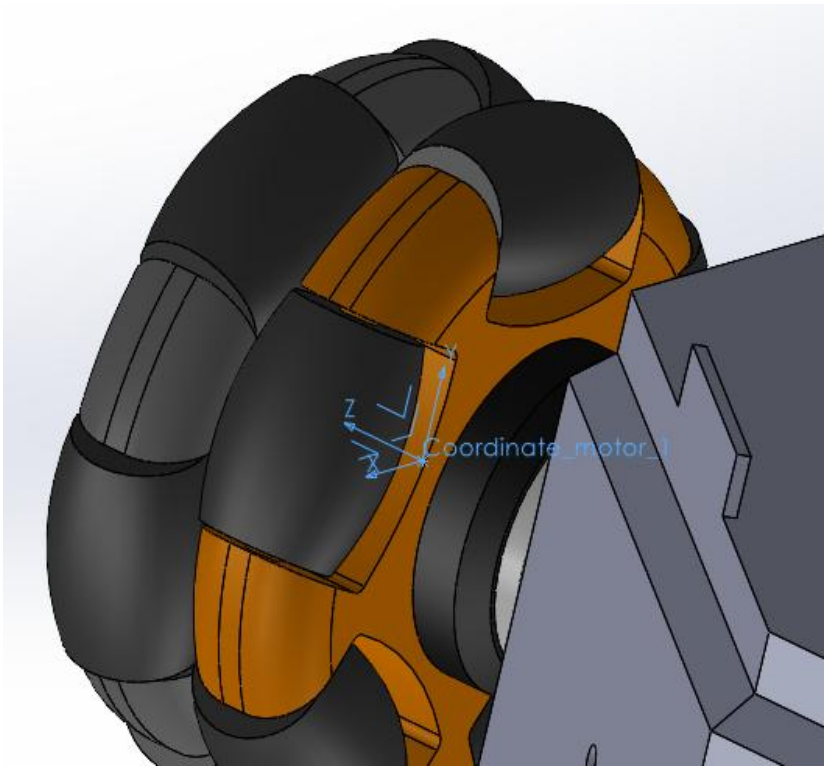
Mô hình robot được thiết kế trên **SolidWorks**, đảm bảo đúng tỷ lệ thực tế và có thể xuất sang URDF để tích hợp vào ROS. Hệ trục tọa độ được xác định như sau:

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

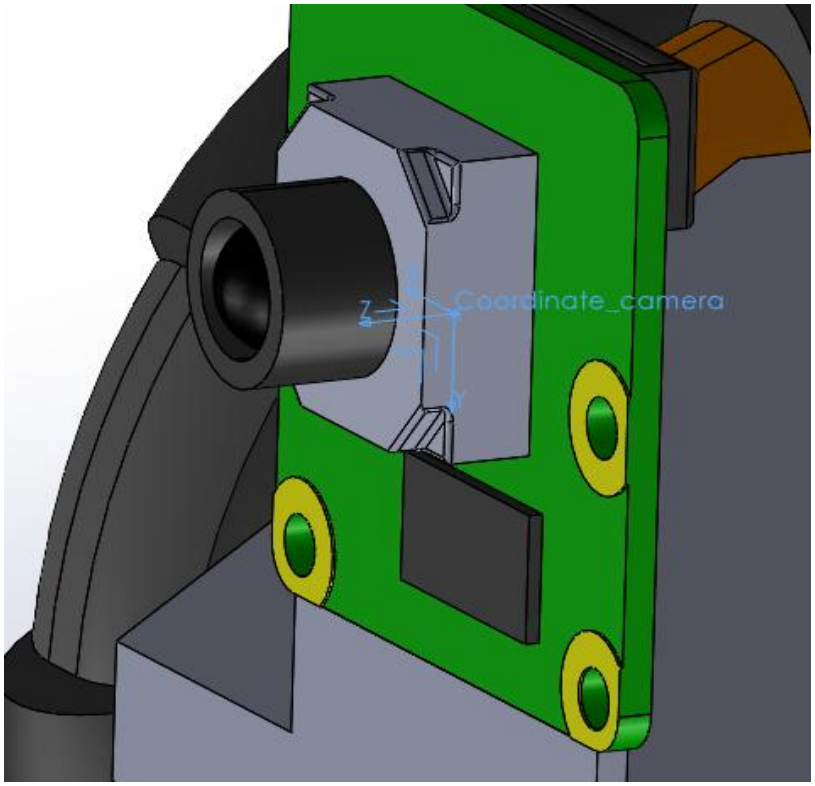
Tiến hành gắn các trục cho xe(global), các bánh , tay máy, imu, camera



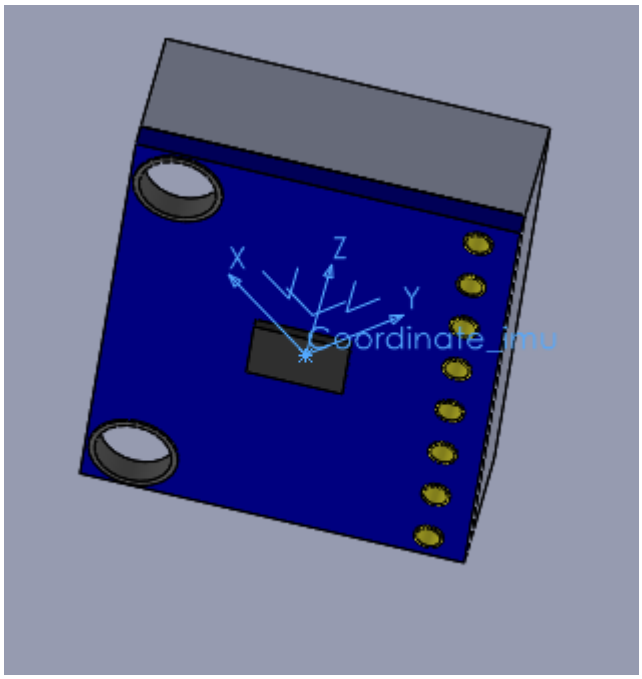
Hình 3.1 Coordinate_global



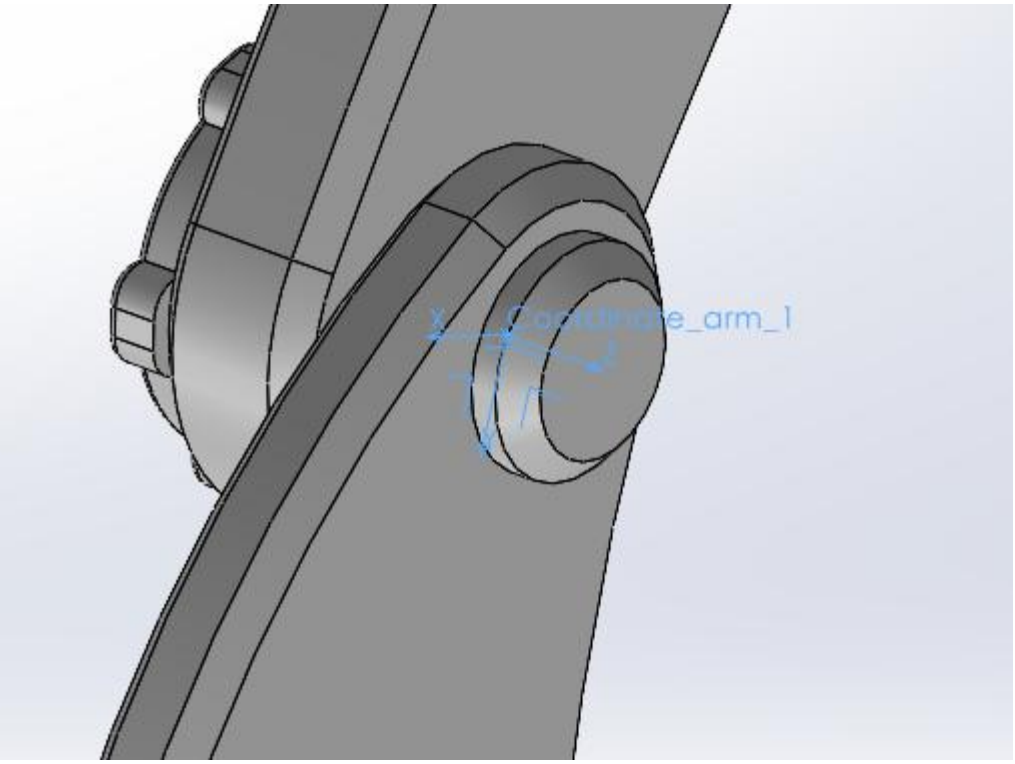
Hình 3.2 Coordinate_motor



Hình 3.3 Coordinate_camera



Hình 3.4 Coordinate_IMU

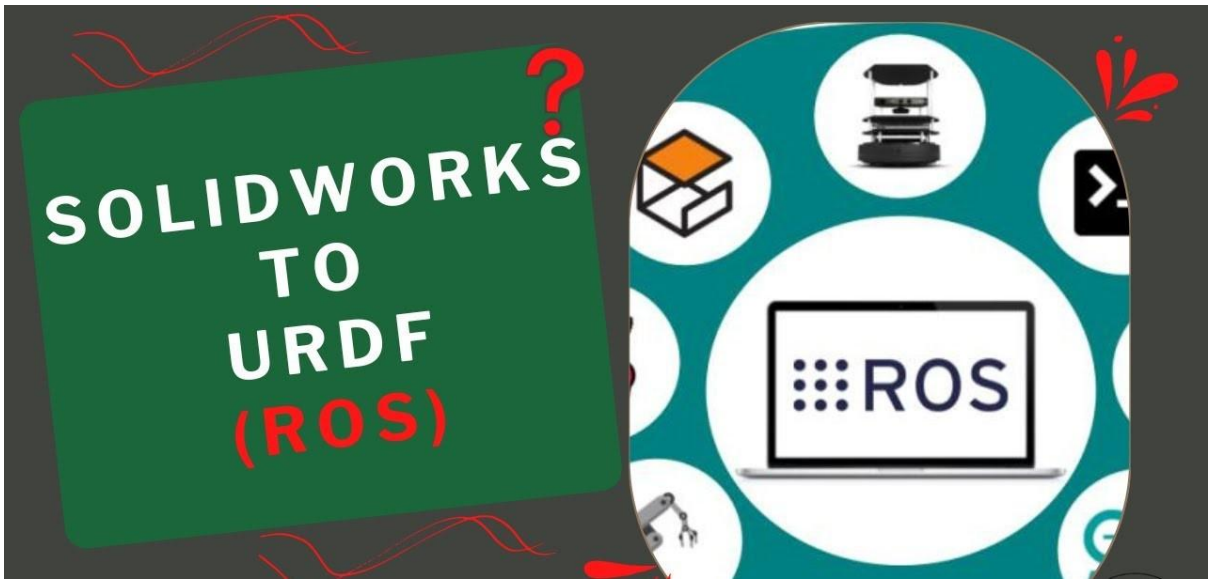


Hình 3.5 Coordinate_của các arm

=> Tiếp theo tiến hành xuất file URDF

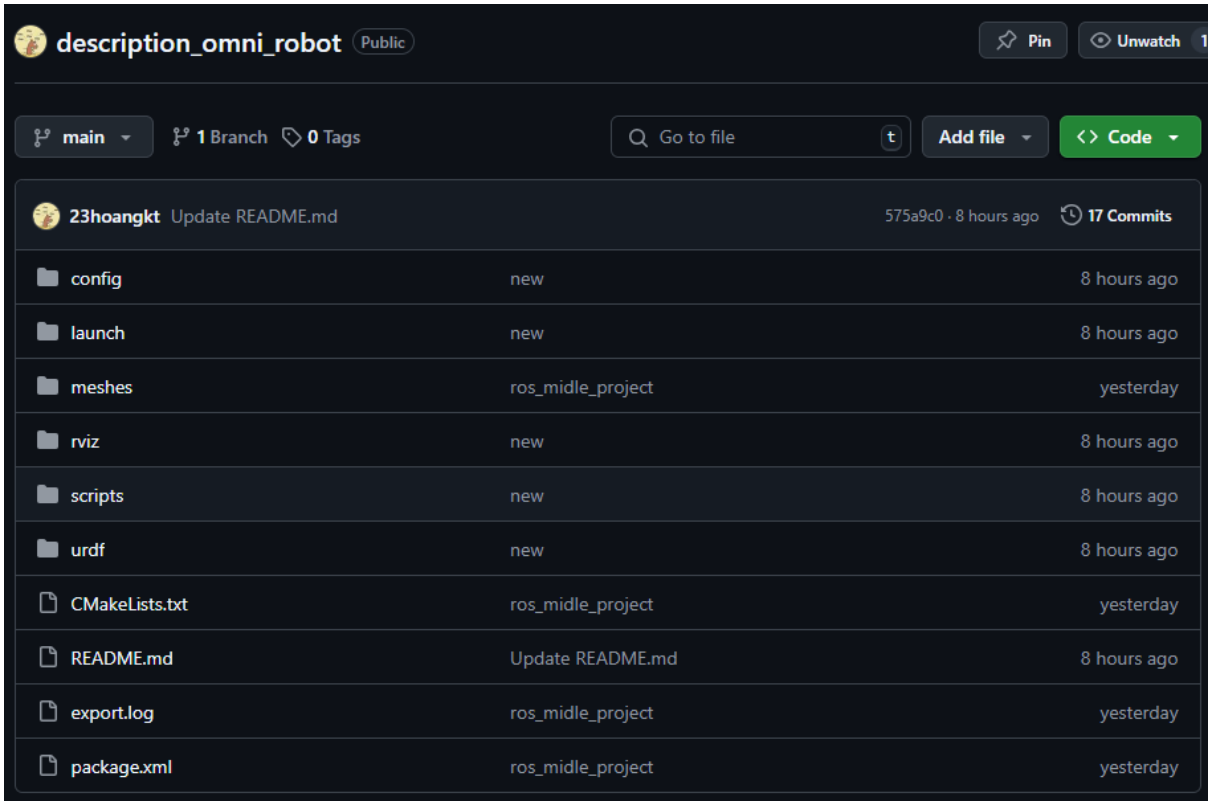
3.1. Xuất file URDF từ SolidWorks

Để chuyển đổi mô hình từ **SolidWorks** sang định dạng URDF sử dụng trong ROS, công cụ **URDF Exporter** được áp dụng. Quá trình này giúp giảm thời gian chỉnh sửa và đảm bảo tính chính xác của mô hình khi đưa vào Gazebo.

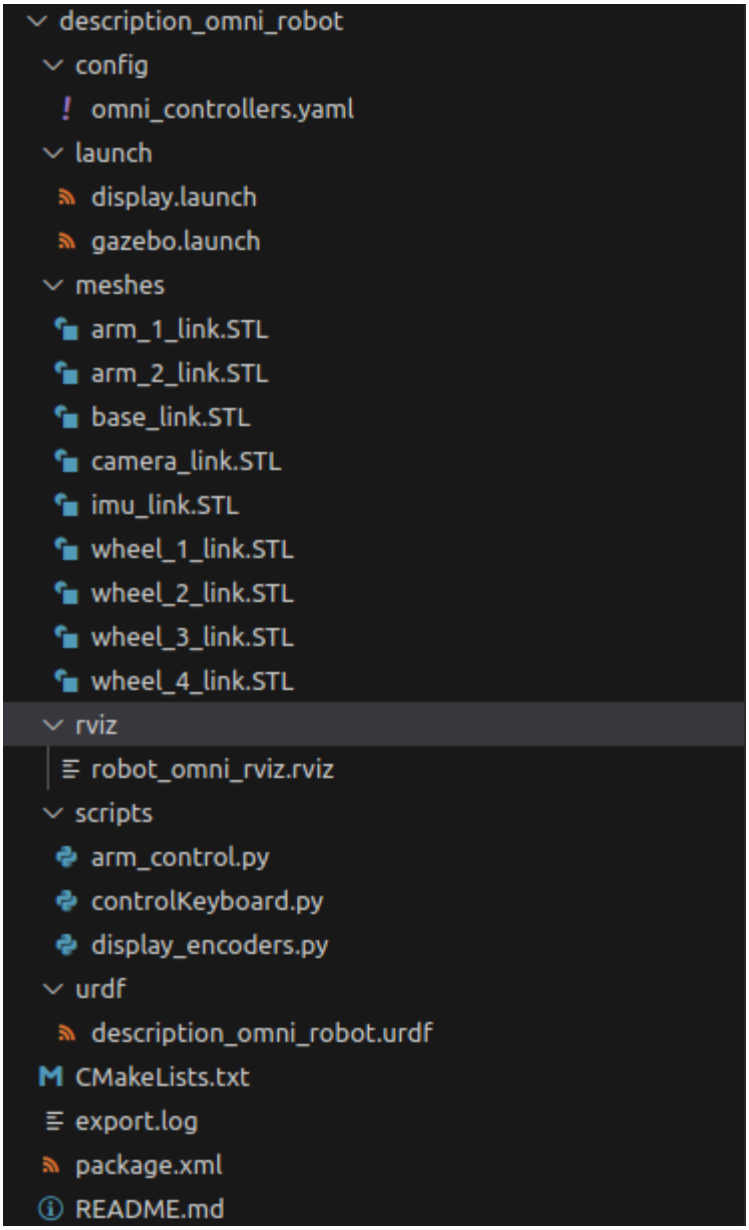


Hình 3.6 …

3.2. Cấu trúc URDF/Xacro



Hình 3.7. Cấu trúc file



Hình 3.8 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:

- **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.

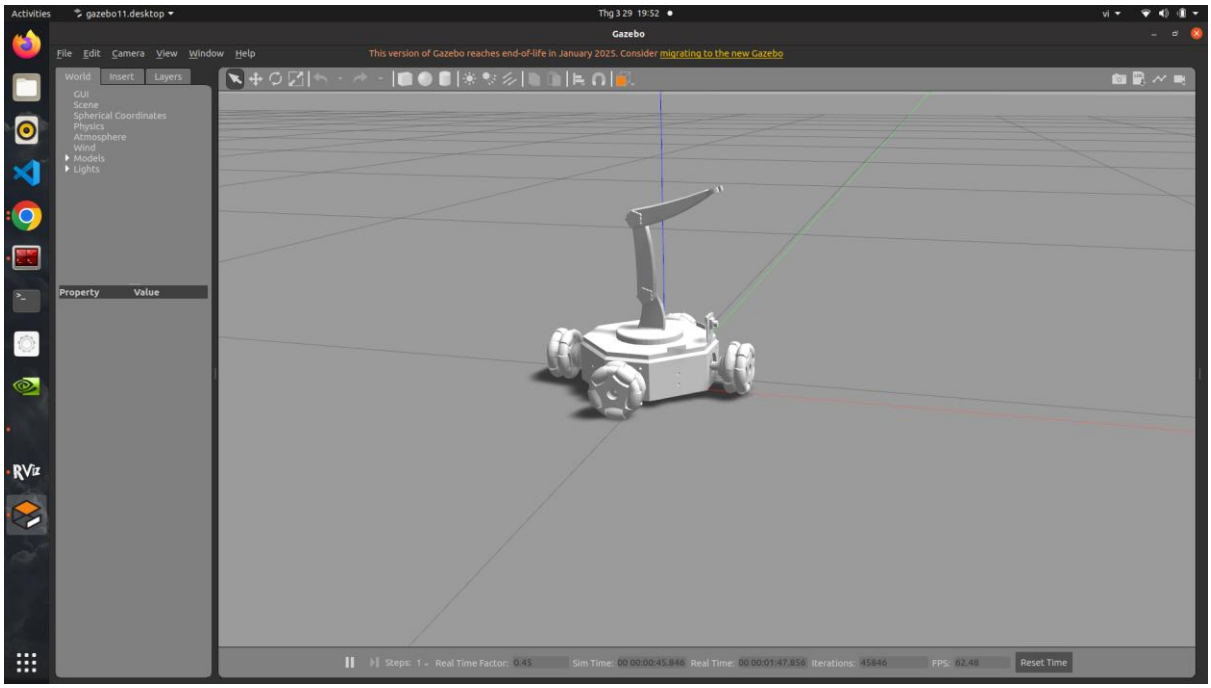
- **rviz/**: Có thể chứa các cấu hình hiển thị cho **RViz**.
- **scripts/**: Chứa các script Python, có thể dùng để điều khiển hoặc test robot.
- **urdf/**: Chứa file mô tả robot **URDF/Xacro**.
- **CMakeLists.txt & package.xml**: Các file cần thiết để build package trong ROS.

Ngoài ra có thể dùng **urdf viewer example** để mô phỏng thử xem có bị lỗi không trước khi cho vào Gazebo:

Hình 3.9 Kết quả urdf viewer example

3.3. Mô phỏng trong Gazebo

Sau khi hoàn thành URDF, robot được tích hợp vào **Gazebo** để kiểm tra và mô phỏng:



Hình 3.10 Kết quả khi cho mô hình lên Gazebo

Tiến hành thêm các Plugin vào file URDF:

- **Tích hợp plugin camera**: Cho phép robot quan sát môi trường xung quanh trong mô phỏng.

```
<!-- plugin cho camera -->
<gazebo reference="camera_link">
  <sensor name="camera_sensor" type="camera">
    <always_on>1</always_on>
    <update_rate>30</update_rate>
    <camera>
      <horizontal_fov>1.0472</horizontal_fov>
      <image_width>640</image_width>
      <image_height>480</image_height>
      <near>0.1</near>
      <far>100</far>
    </camera>
    <plugin name="camera_plugin" filename="libgazebo_ros_camera.so">
      <topicName>/camera/image_raw</topicName>
      <cameraName>camera</cameraName>
      <frameName>camera_link</frameName>
    </plugin>
  </sensor>
</gazebo>
```

Hình 3.11 Plugin cho camera

- **Tích hợp plugin IMU**: Cho phép robot lấy vị trí trong mô phỏng.

```
<!-- plugin cho imu -->
<gazebo reference="imu_link">
    <gravity>true</gravity>
    <sensor name="imu_sensor" type="imu">
      <always_on>true</always_on>
      <update_rate>100</update_rate>
      <visualize>true</visualize>
      <topic>__default_topic__</topic>
      <plugin filename="libgazebo_ros_imu_sensor.so" name="imu_plugin">
        <topicName>imu</topicName>
        <bodyName>imu_link</bodyName>
        <updateRateHZ>10.0</updateRateHZ>
        <gaussianNoise>0.0</gaussianNoise>
        <xyzOffset>0 0 0</xyzOffset>
        <rpyOffset>0 0 0</rpyOffset>
        <frameName>imu_link</frameName>
        <initialOrientationAsReference>false</initialOrientationAsReference>
      </plugin>
      <pose>0 0 0 0 0 0</pose>
    </sensor>
  </gazebo>
```

Hình 3.12 Plugin cho imu

- **Mô phỏng encoder**: Sử dụng plugin velocity để đo tốc độ quay bánh xe.

  Trong **ROS**, plugin transmission được sử dụng trong **URDF** để định nghĩa cách lực từ bộ điều khiển (controller) được truyền đến các khớp (joint) của robot. Nó giúp mô phỏng chính xác hơn các bộ truyền động

  Sử dụng các plugin transmission

```
<!-- transmission for 4 wheel -->

<transmission name="tran_wheel_1">
  <type>transmission_interface/SimpleTransmission</type>
  <joint name="wheel_1_joint">
    <hardwareInterface>hardware_interface/VelocityJointInterface</hardwareInterface>
  </joint>
  <actuator name="wheel_1_motor">
    <mechanicalReduction>1</mechanicalReduction>
  </actuator>
</transmission>
```

Hình 3.13 Transmissioon cho các bánh

⇨ Tương tự cho các bánh còn lại

- **Điều khiển tay máy**: Thông qua ROS topic, giúp thực hiện các thao tác cơ bản như nâng hạ và gắp vật thể.

  Sử dụng các plugin transmission cho tay máy

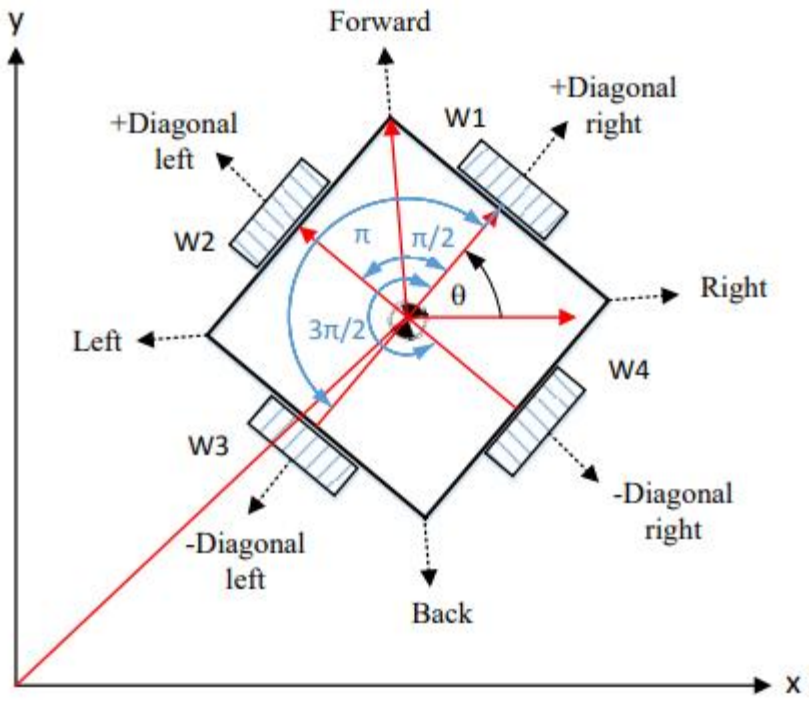
```
<!-- transmission for 2 arm  -->

<transmission name="arm_1_joint_transmission">
  <type>transmission_interface/SimpleTransmission</type>
  <joint name="arm_1_joint">
    <hardwareInterface>hardware_interface/PositionJointInterface</hardwareInterface>
  </joint>
  <actuator name="arm_1">
    <mechanicalReduction>1.0</mechanicalReduction>
  </actuator>
</transmission>

<transmission name="arm_2_joint_transmission">
  <type>transmission_interface/SimpleTransmission</type>
  <joint name="arm_2_joint">
    <hardwareInterface>hardware_interface/PositionJointInterface</hardwareInterface>
  </joint>
  <actuator name="arm_2">
    <mechanicalReduction>1.0</mechanicalReduction>
  </actuator>
</transmission>
```

Hình 3.14. Transmission cho arm

4. Điều khiển robot trong ROS

4.1. Mô Hình Động Học của Robot Bốn Bánh Omni

Robot di động omnidirectional với bốn bánh xe là một nền tảng holonomic, cho phép di chuyển linh hoạt theo nhiều hướng và thay đổi hướng nhanh chóng. Phần động học của robot bao gồm hai mô hình chính: động học thuận (forward kinematics) và động học ngược (inverse kinematics), được sử dụng để mô tả và điều khiển chuyển động của robot trong không gian hai chiều (XY) với ba bậc tự do (vị trí x, y và góc hướng θ).



Hình 4.1 minh họa vị trí của các bánh xe và các ký hiệu liên quan.

Động học ngược (Inverse Kinematics - IK)

Động học ngược được sử dụng để tính vận tốc tuyến tính của từng bánh xe (υ_ω) từ vận tốc của robot trong hệ tọa độ toàn cục ($V_{(g)} = [x^{\cdot}, y^{\cdot}, \omega]^T$). Công thức IK được biểu diễn như sau:

$$V_w = \begin{bmatrix} w_1 \\ w_2 \\ w_3 \\ w_4 \end{bmatrix} = \frac{1}{r} \begin{bmatrix} -sin\left(\theta + \frac{\pi}{4}\right) & cos\left(\theta + \frac{\pi}{4}\right) & R \\ -sin\left(\theta + \frac{3\pi}{4}\right) & cos\left(\theta + \frac{3\pi}{4}\right) & R \\ -sin\left(\theta + \frac{5\pi}{4}\right) & cos\left(\theta + \frac{5\pi}{4}\right) & R \\ -sin\left(\theta + \frac{7\pi}{4}\right) & cos\left(\theta + \frac{7\pi}{4}\right) & R \end{bmatrix} \begin{bmatrix} V_x \\ V_y \\ \dot{\theta} \end{bmatrix}$$

Hình 4.2 Inverse Kinematic

Ma trận T(θ) phản ánh cách vận tốc robot được phân bổ cho từng bánh xe dựa trên định hướng và vị trí hình học.

Động học thuận (Forward Kinematics - FK)

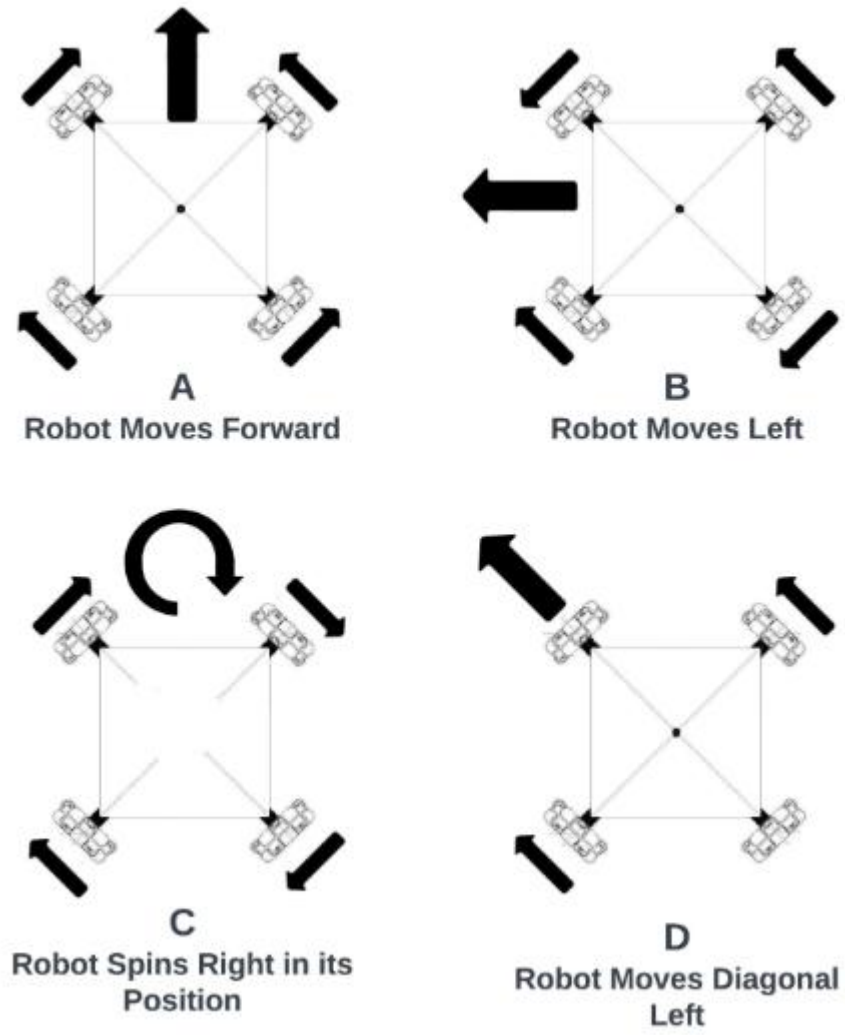
Động học thuận xác định vận tốc của robot dựa trên tốc độ quay của bốn bánh xe Công thức được biểu diễn như sau:

$$V_r = \begin{bmatrix} V_x \\ V_y \\ \dot{\theta} \end{bmatrix} = J \begin{bmatrix} w_1 \\ w_2 \\ w_3 \\ w_4 \end{bmatrix}$$

$$J = \frac{r}{2} \begin{bmatrix} -sin\left(\theta + \frac{\pi}{4}\right) & cos\left(\theta + \frac{\pi}{4}\right) & \frac{1}{2R} \\ -sin\left(\theta + \frac{3\pi}{4}\right) & cos\left(\theta + \frac{3\pi}{4}\right) & \frac{1}{2R} \\ -sin\left(\theta + \frac{5\pi}{4}\right) & cos\left(\theta + \frac{5\pi}{4}\right) & \frac{1}{2R} \\ -sin\left(\theta + \frac{7\pi}{4}\right) & cos\left(\theta + \frac{7\pi}{4}\right) & \frac{1}{2R} \end{bmatrix}^T$$

Hình 4.3 Forward Kinematic

Cách di chuyển của Robot Omni 4 bánh



Hình 4.4 Cách xe di chuyển

4.2. Điều khiển robot trong Gazebo

- **ROS Control**: Hỗ trợ điều khiển chính xác tốc độ và hướng di chuyển của bánh xe.

```
<!-- plugin gazebo control -->
<gazebo>
  <plugin name ="gazebo_ros_control" filename = "libgazebo_ros_control.so">
      <robotNamespace>/</robotNamespace>
  </plugin>
</gazebo>
```

Hình 4.5 Plugin ros control

Plugin gazebo_ros_control trong Gazebo có nhiệm vụ kết nối hệ thống điều khiển của ROS với mô phỏng Gazebo, giúp bạn điều khiển robot bằng các controller từ ROS.

Cụ thể, plugin này:
Kết nối Gazebo với ROS Control: Nó cho phép sử dụng **ros_control** để điều khiển các joint của robot trong Gazebo thông qua các controller như :

- **effort_controllers/JointEffortController**
- **position_controllers/JointPositionController**
- **velocity_controllers/JointVelocityController**

Cấp quyền điều khiển cho ROS: Khi plugin này được thêm vào mô hình, các topic như /robot/joint_states và /robot/controller_manager sẽ được tạo, giúp ROS có thể gửi lệnh điều khiển đến các joint của robot.

Mô phỏng hệ thống động lực học: Nếu bạn có một robot di động hoặc cánh tay robot, plugin này giúp mô phỏng chính xác lực, vận tốc và vị trí của các khớp khi điều khiển.

omni_controllers.yaml

config > omni_controllers.yaml

```yaml
arm_1_joint_controller:
  type: "position_controllers/JointPositionController"
  joint: "arm_1_joint"
  pid: {p: 10.0, i: 0.05, d: 1.0}

arm_2_joint_controller:
  type: "position_controllers/JointPositionController"
  joint: "arm_2_joint"
  pid: {p: 10.0, i: 0.05, d: 1.0}

joint_state_controller:
  type: "joint_state_controller/JointStateController"
  publish_rate: 50

wheel_1_joint_controller:
  type: "velocity_controllers/JointVelocityController"
  joint: "wheel_1_joint"
  pid: {p: 10.0, i: 0.1, d: 1.0}

wheel_2_joint_controller:
  type: "velocity_controllers/JointVelocityController"
  joint: "wheel_2_joint"
  pid: {p: 10.0, i: 0.1, d: 1.0}

wheel_3_joint_controller:
  type: "velocity_controllers/JointVelocityController"
  joint: "wheel_3_joint"
  pid: {p: 10.0, i: 0.1, d: 1.0}

wheel_4_joint_controller:
  type: "velocity_controllers/JointVelocityController"
  joint: "wheel_4_joint"
  pid: {p: 10.0, i: 0.1, d: 1.0}
```

Hình 4.6 File .yaml định nghĩa các bộ điều khiển cho các khớp của robot

- **Teleop_keyboard**: Nhận lệnh từ bàn phím và chuyển thành tín hiệu điều khiển robot.
  Các thành phần chính trong code điều khiển xe bằng teleop keyboard:

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

        # Publishers for wheel velocity commands (4 wheels)
        self.pub_wheel_1 = rospy.Publisher('/wheel_1_joint_controller/command', Float64, queue_size=10)
        self.pub_wheel_2 = rospy.Publisher('/wheel_2_joint_controller/command', Float64, queue_size=10)
        self.pub_wheel_3 = rospy.Publisher('/wheel_3_joint_controller/command', Float64, queue_size=10)
        self.pub_wheel_4 = rospy.Publisher('/wheel_4_joint_controller/command', Float64, queue_size=10)
```

  rospy.init_node('omni_keyboard_control', anonymous=True): Khởi tạo node ROS tên "omni_keyboard_control".

  self.pub_wheel_1, self.pub_wheel_2, self.pub_wheel_3, self.pub_wheel_4: Tạo 4 Publisher để gửi lệnh vận tốc tới từng bánh xe qua các topic **/wheel_X_joint_controller/command**

```
# Map keyboard inputs to desired velocities
if key == 'w':  # Forward
    vy = self.max_speed
elif key == 's':  # Backward
    vy = -self.max_speed
elif key == 'a':  # Left
    vx = -self.max_speed
elif key == 'd':  # Right
    vx = self.max_speed
elif key == 'q':  # Rotate left
    omega = self.max_speed
elif key == 'e':  # Rotate right
    omega = -self.max_speed
elif key == 'x':  # Stop
    vx = 0.0
    vy = 0.0
    omega = 0.0
elif key == '\x03':  # Ctrl+C (ASCII code 3)
    self.wheel_1_speed = 0.0
    self.wheel_2_speed = 0.0
    self.wheel_3_speed = 0.0
    self.wheel_4_speed = 0.0
    self.pub_wheel_1.publish(self.wheel_1_speed)
    self.pub_wheel_2.publish(self.wheel_2_speed)
    self.pub_wheel_3.publish(self.wheel_3_speed)
    self.pub_wheel_4.publish(self.wheel_4_speed)
    rospy.loginfo("Stopping robot and exiting")
    break
```

Các phím điều khiển

```
# Calculate wheel speeds for omni wheels (with corrected directions)
self.wheel_1_speed = -vy + omega * self.L
self.wheel_2_speed = vy + omega * self.L
self.wheel_3_speed = -vx + omega * self.L
self.wheel_4_speed = vx + omega * self.L

# Publish velocity commands
self.pub_wheel_1.publish(self.wheel_1_speed)
self.pub_wheel_2.publish(self.wheel_2_speed)
self.pub_wheel_3.publish(self.wheel_3_speed)
self.pub_wheel_4.publish(self.wheel_4_speed)
```

Tính vận tốc bánh xe (dựa trên mô hình động học) và publish lên các wheel

4.3. Điều khiển tay máy bằng teleop_keyboard

Người dùng có thể điều khiển tay máy thông qua bàn phím bằng **teleop_keyboard**. Lệnh điều khiển gửi qua ROS topic như sau:

Thành phần chính của code

```
def arm_controller():
    rospy.init_node('arm_controller', anonymous=True)
    arm_1_pub = rospy.Publisher('/arm_1_joint_controller/command', Float64, queue_size=10)
    arm_2_pub = rospy.Publisher('/arm_2_joint_controller/command', Float64, queue_size=10)
    arm_1_pos = 0.0
    arm_2_pos = 0.0
    step = 0.5
    print("Điều khiển tay máy:")
    print("1: Tăng vị trí arm_1")
    print("2: Giảm vị trí arm_1")
    print("3: Tăng vị trí arm_2")
    print("4: Giảm vị trí arm_2")
    print("q: Thoát")
    rate = rospy.Rate(50)
    while not rospy.is_shutdown():
        key = get_key()
        if key == '1':
            arm_1_pos += step
            if arm_1_pos > 1.57:
                arm_1_pos = 1.57
        elif key == '2':
            arm_1_pos -= step
            if arm_1_pos < -1.57:
                arm_1_pos = -1.57
        elif key == '3':
            arm_2_pos += step
            if arm_2_pos > 1.57:
                arm_2_pos = 1.57
        elif key == '4':
            arm_2_pos -= step
            if arm_2_pos < -1.57:
                arm_2_pos = -1.57
        elif key == 'q':
            break
        arm_1_pub.publish(arm_1_pos)
        arm_2_pub.publish(arm_2_pos)
        rospy.loginfo("Arm 1 position: %.2f rad, Arm 2 position: %.2f rad", arm_1_pos, arm_2_pos)
```

rospy.init_node('arm_controller', anonymous=True): Khởi tạo node ROS tên "**arm_controller**".

- **arm_1_pub, arm_2_pub**: Tạo 2 Publisher để gửi lệnh vị trí tới **/arm_1_joint_controller/command và /arm_2_joint_controller/command.**

arm_1_pos, arm_2_pos: Lưu trữ vị trí hiện tại của hai khớp (ban đầu bằng 0).

step = 0.5: Bước thay đổi vị trí mỗi lần nhấn phím (rad).

Giới hạn vị trí: Từ -1.57 đến 1.57 rad

arm_controller() : Điều khiển qua keyboard

4.4. Hiển thị encoder

```python
#!/usr/bin/env python3

import rospy
from sensor_msgs.msg import JointState

def joint_state_callback(msg):
    # Danh sách các khớp cần hiển thị
    joints_to_display = ['arm_1_joint', 'arm_2_joint', 'wheel_1_joint', 'wheel_2_joint', 'wheel_3_joint', 'wheel_4_joint']

    # Duyệt qua danh sách khớp và hiển thị giá trị
    for i, joint_name in enumerate(msg.name):
        if joint_name in joints_to_display:
            position = msg.position[i]
            velocity = msg.velocity[i] if i < len(msg.velocity) else 0.0  # Đảm bảo không lỗi nếu velocity rỗng
            rospy.loginfo("%s: Position = %.3f rad, Velocity = %.3f rad/s", joint_name, position, velocity)

def encoder_display():
    # Khởi tạo node
    rospy.init_node('encoder_display', anonymous=True)

    # Đăng ký subscriber để đọc topic /joint_states
    rospy.Subscriber('/joint_states', JointState, joint_state_callback)

    # Giữ node chạy
    rospy.loginfo("Hiển thị giá trị encoder của các khớp...")
    rospy.spin()

if __name__ == '__main__':
    try:
        encoder_display()
    except rospy.ROSInterruptException:
        pass
```

Khởi tạo ROS Node và Subscriber:

- **rospy.init_node('encoder_display', anonymous=True)**: Khởi tạo node ROS tên "**encoder_display**".
- **rospy.Subscriber('/joint_states', JointState, joint_state_callback):** Đăng ký lắng nghe topic **/joint_states** và gọi hàm **joint_state_callback** khi có dữ liệu.

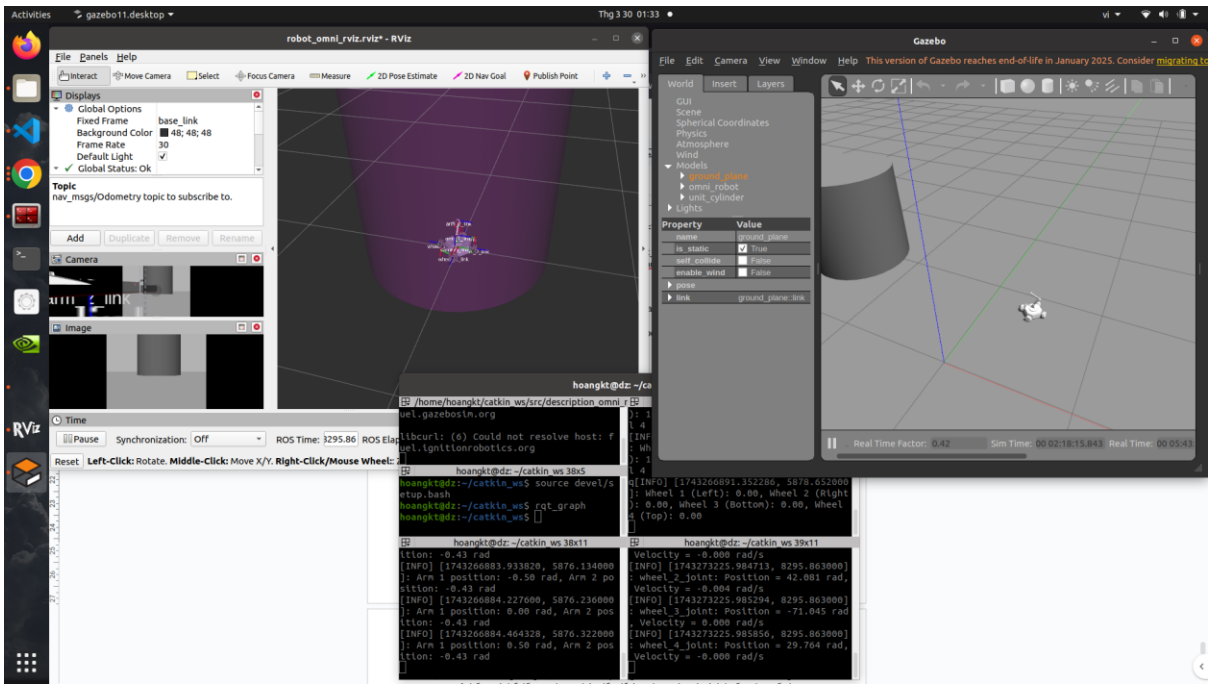
Hàm joint_state_callback(msg) - Xử lý dữ liệu:

- **Danh sách khớp cần hiển thị: joints_to_display = ['arm_1_joint', 'arm_2_joint', 'wheel_1_joint', ...]** (tay máy và 4 bánh xe).
- **Hiển thị thông tin**: Duyệt qua msg.name, lấy position (vị trí) và velocity (vận tốc) của từng khớp, in ra bằng **rospy.loginfo**.

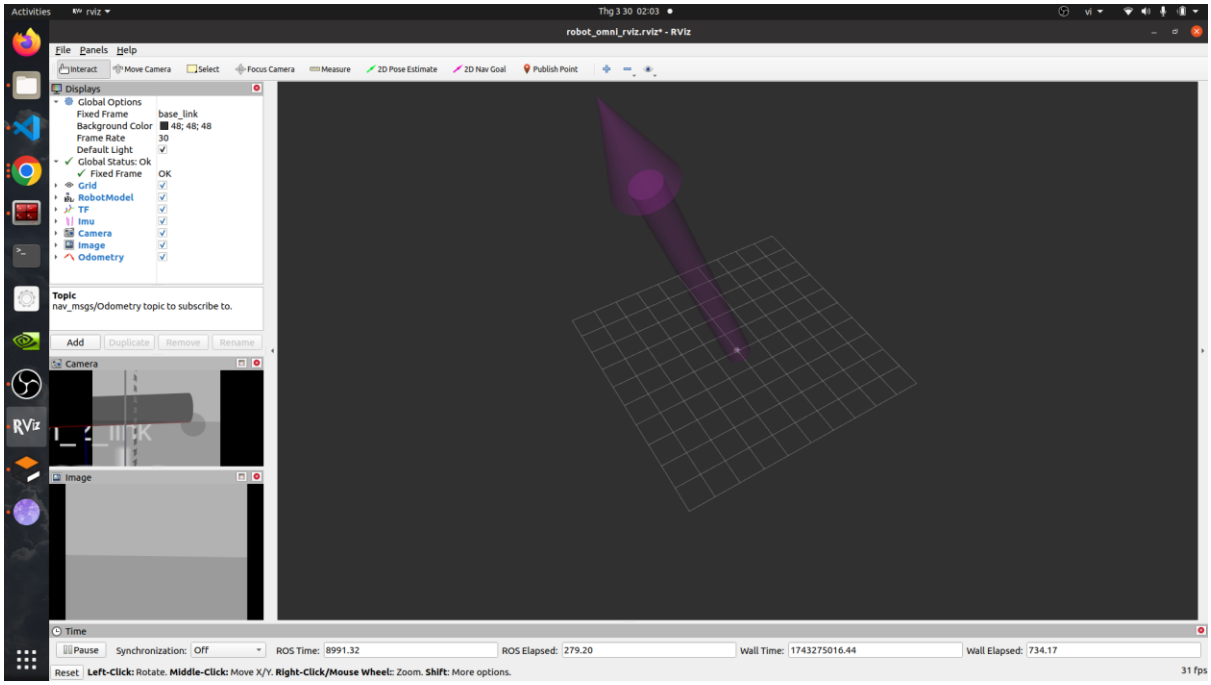
Hàm encoder_display():

- Khởi tạo node và subscriber, sau đó dùng **rospy.spin()** để giữ node hoạt động và liên tục nhận dữ liệu từ topic.

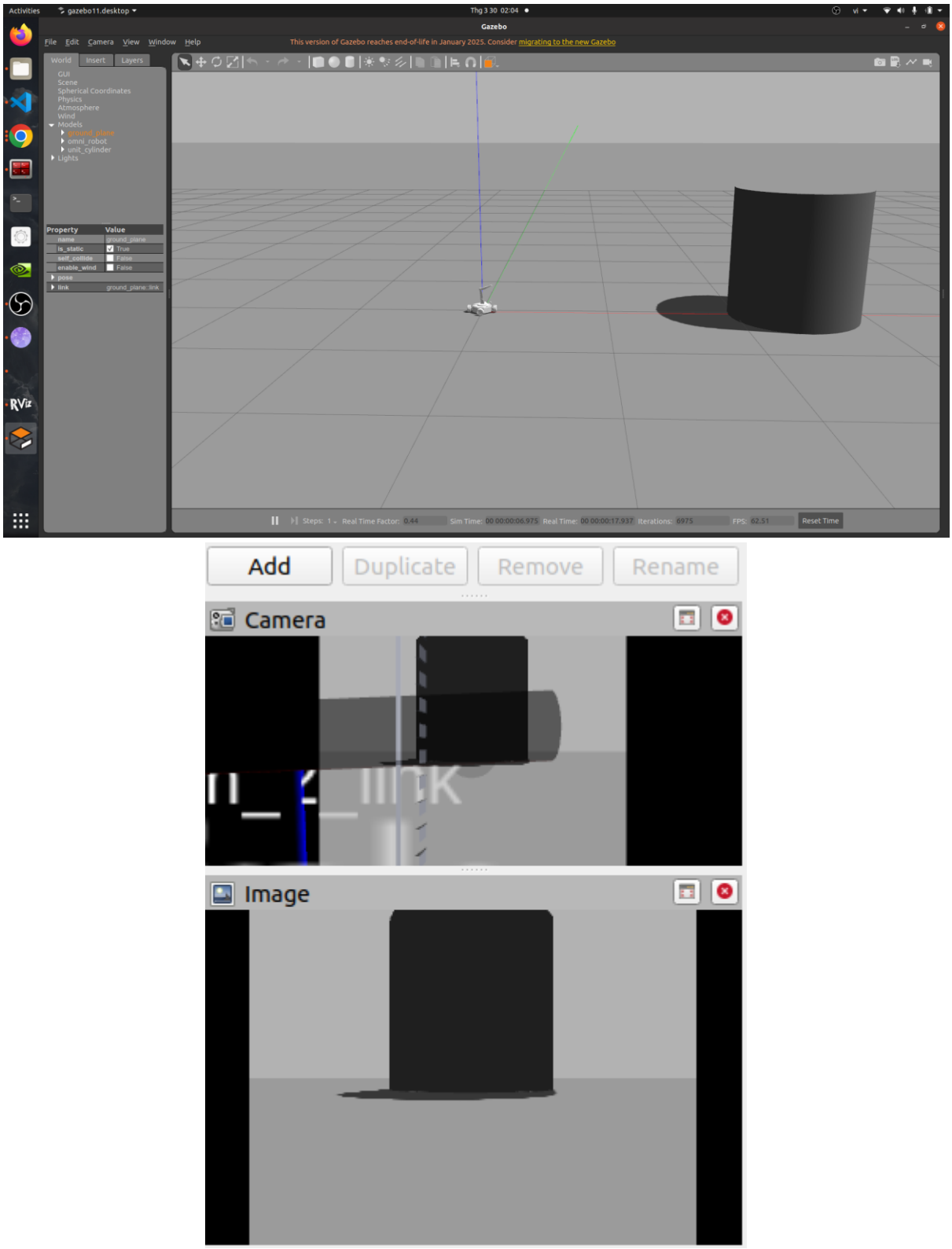
5. Kết quả



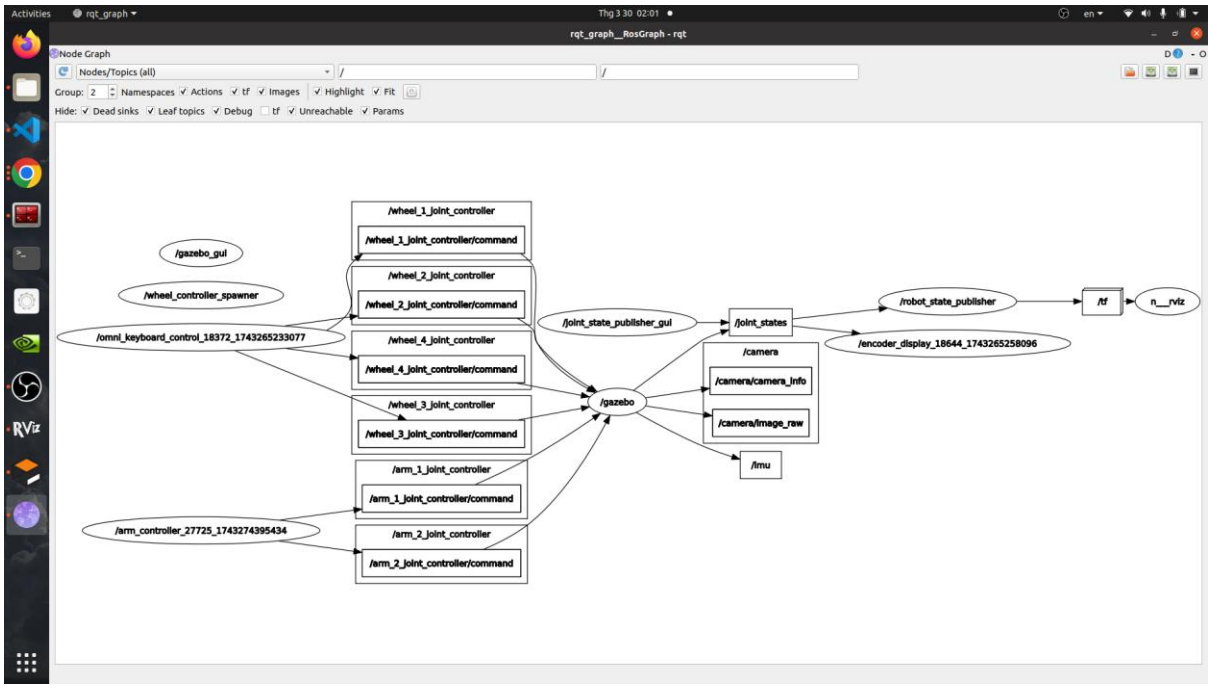
Hình 5.1 Khởi động chương trình cùng các node



Hình 5.2. Topic từ IMU



Hình 5.3 Topic từ camera



Rqt_graph

Robot hoạt động ổn định trong Gazebo, di chuyển linh hoạt và tay máy có thể thực hiện các thao tác chính xác, Youtube

6. Mã nguồn và tham khảo

Link Github : https://github.com/23hoangkt/description_omni_robot

https://www.researchgate.net/publication/324929202_Motion_Improvement_of_Four-Wheeled_Omnidirectional_Mobile_Robots_for_Indoor_Terrain/fulltext/5aebadf2458515f59981e3b1/Motion-Improvement-of-Four-Wheeled-Omnidirectional-Mobile-Robots-for-Indoor-Terrain.pdf

--- Hết ---