**ĐẠI HỌC QUỐC GIA HÀ NỘI**

**TRƯỜNG ĐẠI HỌC CÔNG NGHỆ**

----------.***----------

# BÁO CÁO GIỮA KÌ

**Môn: Lập trình Robot với ROS**

Họ và tên sinh viên: Hoàng Văn Cường

MSV: 22027549

*- Ngày 1 tháng 4 năm 2025 –*

| Tóm tắt |
| --- |
| Dự án tập trung vào thiết kế và mô phỏng di chuyển của robot 2 bánh vi sai trong môi trường ROS (Robot Operating System). Robot được trang bị 3 cảm biến chính là: cảm biến Lidar, Encoder và Camera. Mô hình 3D của Robot được xây dựng chi tiết trong Solidworks sau đó được chuyển sang định dạng URDF để hiển thị kiểm tra trong Rviz. Tiếp theo sẽ mô phỏng di chuyển đồng thời tích hợp các cảm biến giúp Robot thực hiện nhiều chức năng trong Gazebo. Báo cáo đã trình bày quá trình thiết kế, tính toán và liên kết cảm biến đáp ứng đầy đủ theo yêu cầu của đề bài đưa ra. |

Mục lục

# 1. Giới thiệu

## *1.1. Mục tiêu của đề tài*

Đề tài được đưa ra với mục tiêu thiết kế và mô phỏng được Robot 2 bánh vi sai di chuyển trong môi trường ROS (Robot Operating System) với những cảm biến được trang bị là cảm biến Lidar, Encoder và Camera với mong muốn điều hướng chính xác và nhận diện được môi trường xung quanh. Dự án chỉ đang hướng đến mô phỏng khả năng di chuyển và làm việc của robot trong môi trường mô phỏng, thu thập về những dữ liệu mà cảm biến nhận được từ đó phát triển thêm những dự án thực tế trong tương lai.

## *1.2. Yêu cầu đề bài*

Đề bài yêu cầu Robot phải đáp ứng các tiêu chí :

- Sử dụng cơ cấu 2 bánh vi sai để di chuyển Robot
- Tích hợp 3 loại cảm biến: Lidar, Encoder và Camera
- Mô hình hóa đầy đủ bằng URDF để hiển thị và mô phỏng trên ROS
- Hoạt động trên môi trường ROS với các cảm biến và di chuyển được Robot và tay máy thông qua các nút điều khiển

## *1.3. Tổng quan về Robot 2 bánh vi sai*

Robot được thiết kế hợp lý dạng hình nón phù hợp với việc bố trí hai bánh vi sai và một bánh phụ phía trước để giữ cân bằng và đảm bảo khả năng di chuyển. Cấu trúc của Robot gồm 2 tầng với từng khả năng riêng:

- Tầng 1: gồm 2 động cơ và 2 bánh xe cùng với một bánh phụ ở mặt dưới giúp robot có thể di chuyển linh hoạt. Phía trên được đặt một tay máy với 2 khớp xoay (Rotation) có thể xoay linh hoạt giúp robot có thể thực hiện những yêu cầu phức tạp hơn. Ngay phía trước Robot được trang bị camera giúp Robot thu thập được hình ảnh từ bên ngoài môi trường linh hoạt theo chuyển đoọng của nó.
- Tầng 2: Tầng 2 được thiết kế cao hơn và được đặt một cảm biến Lidar có thể quét 180 độ giúp cho Robot có thể phát hiện được những vật cản ở môi trường xung quanh.

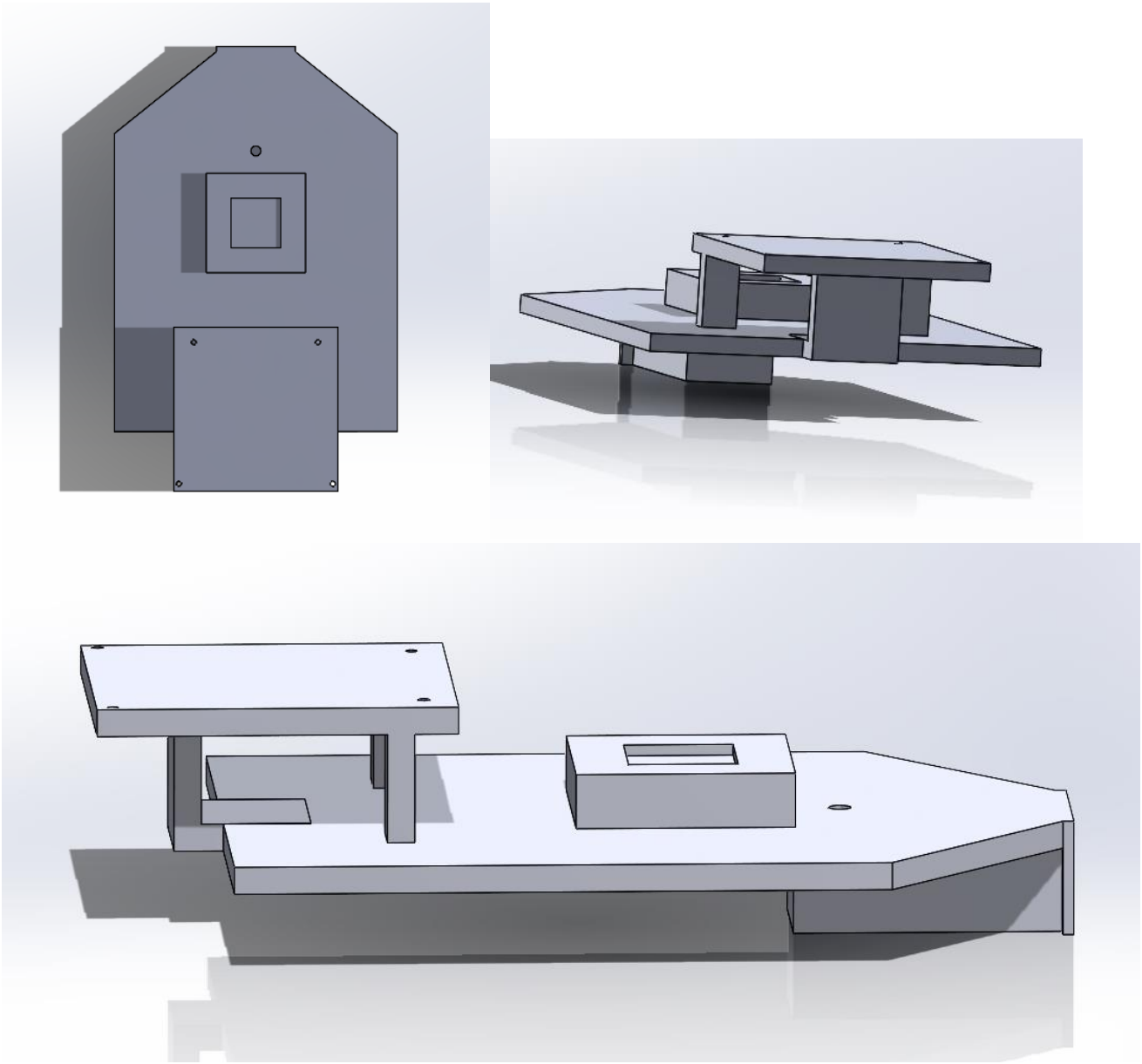
# 2. Thiết kế mô hình Robot

## *2.1. Thiết kế cơ khí*

Tổng quan Robot được thiết kế phần thân di chuyển kiểu 2 bánh vi sai kèm theo bánh phụ phía trước kết hợp khả năng linh hoạt của tay máy 2 khớp giúp cho Robot có thể di chuyển cũng như thực hiện nhiệm vụ một cách linh hoạt.

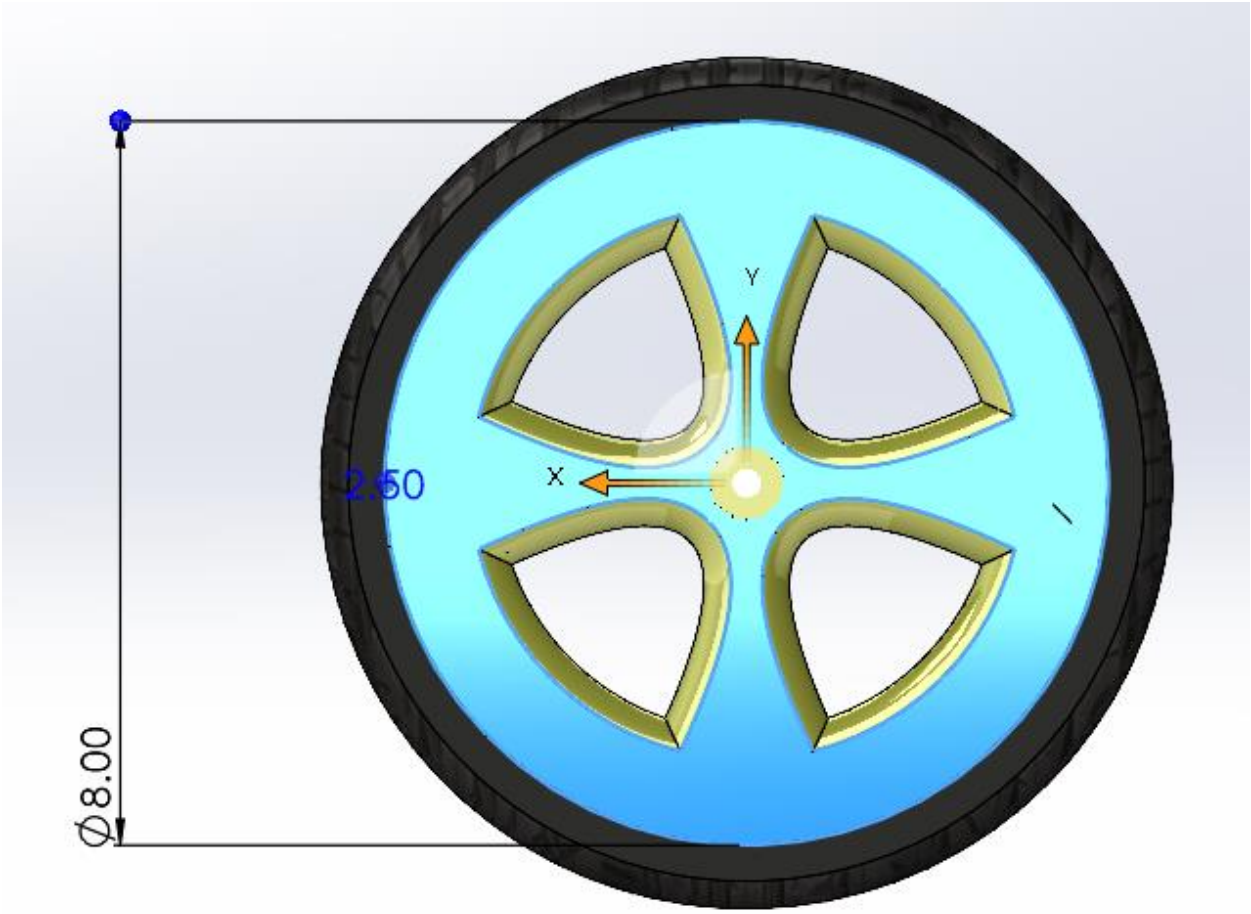
Mô hình 3D của robot được thiết kế trên Solidworks với các thông số phần cứng :

- Base_link:



+ Chiều dài : 15cm
+ Chiều rộng : 12cm
+ Chiều cao : 7cm(bao gồm cả Lidar)

## *2.2. Bánh xe và động cơ*

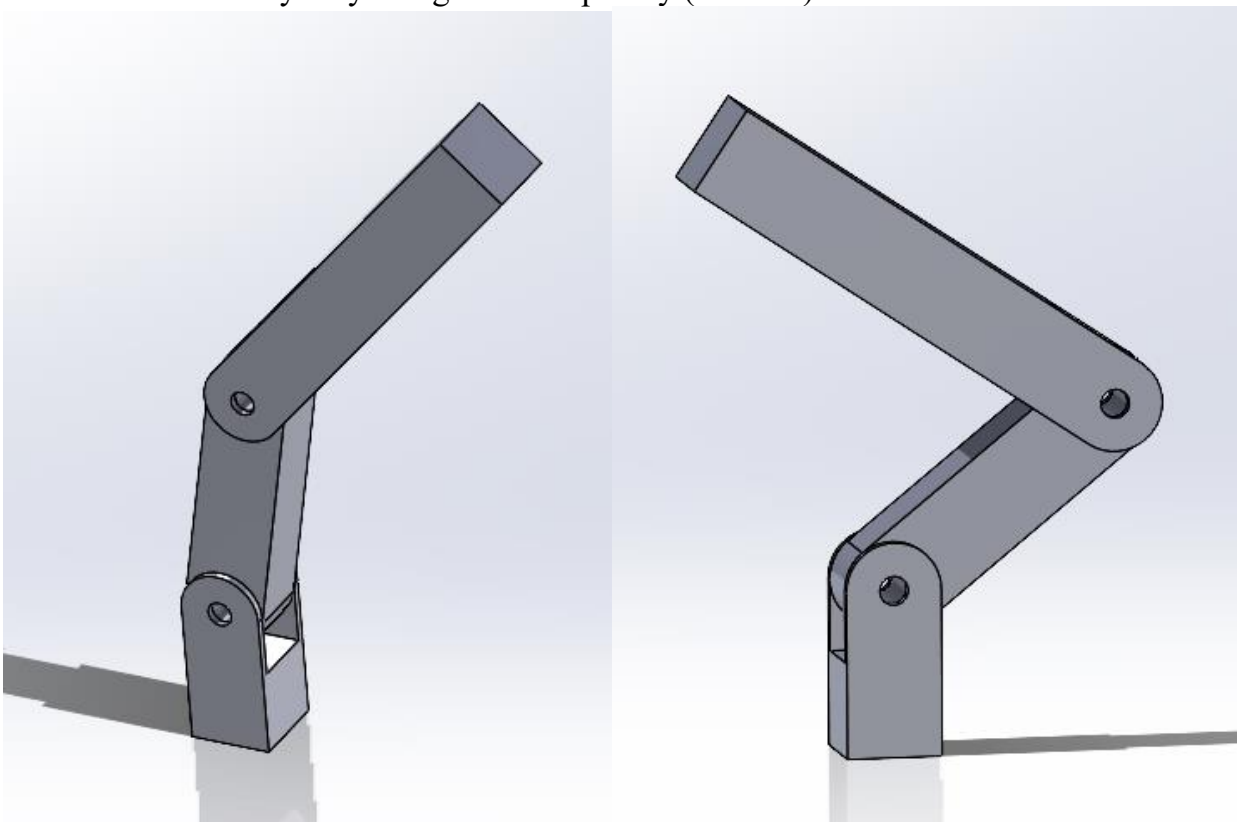


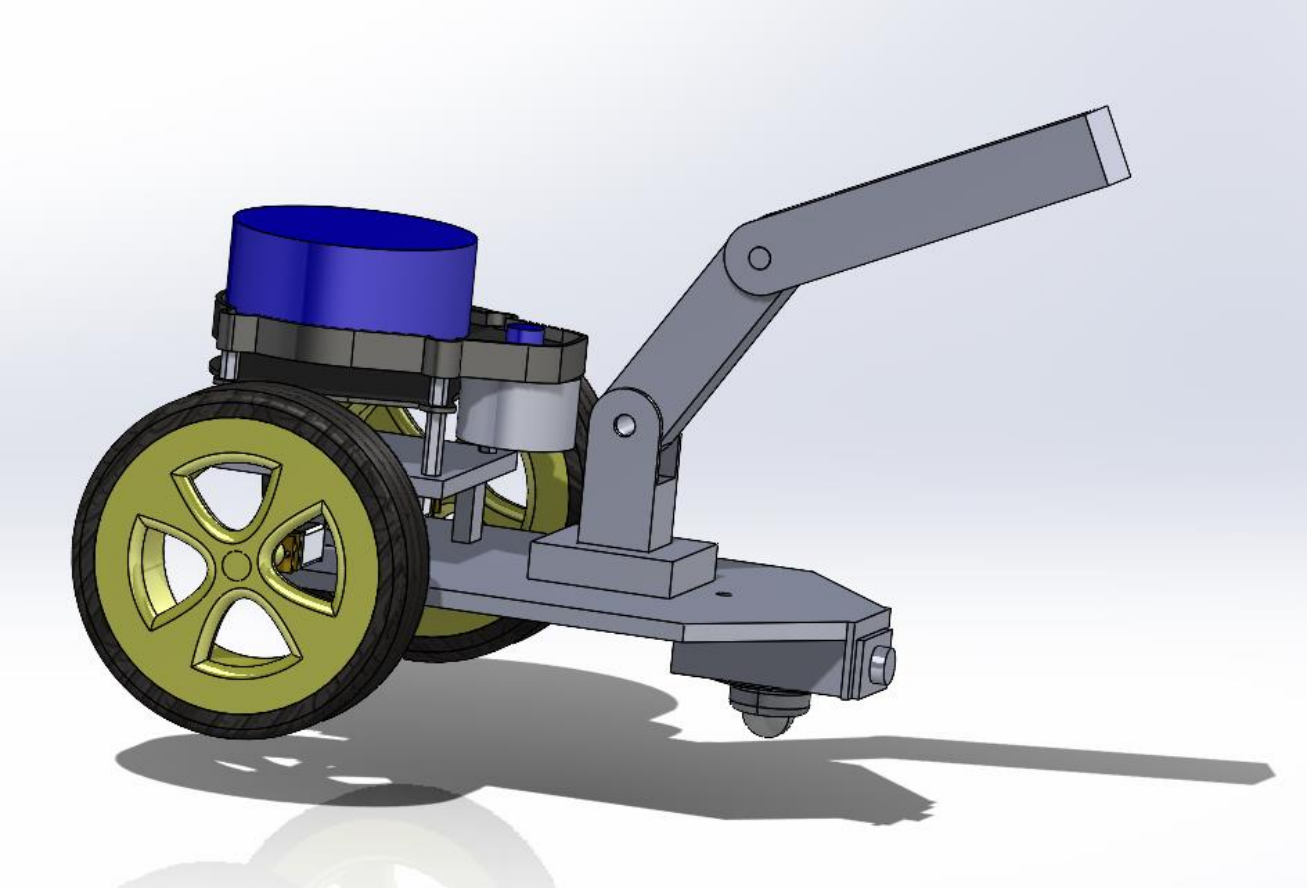

+ Bán kính bánh xe: 4.5cm
+ Bán kính trục: 0.5cm
+ chất liệu: nhựa ABS và cao su

- Kết hợp motor và bánh xe:

## *2.3. Kết cấu tay máy*

- Tay máy bao gồm 2 khớp xoay (rotatuin)



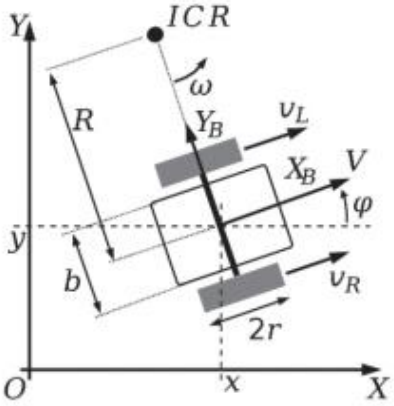


+ Khớp 1: Quay quanh trục với phạm vi 180, kết nối giữa trụ với arm2
+ Khớp 2: Quay đồng trục với khớp 1 và hỗ trợ nâng hạ vật

## 3. Cấu trúc hình học và mô hình toán học Robot

### *3.1. Mô hình động học robot 2 bánh vi sai*

- Hệ tọa độ:




Tốc độ tiếp xúc với mặt đất của bánh xe bên trái là VL và bánh xe bên phải VR dẫn đến

xe chuyển động quay với vận tốc góc ω. Theo định nghĩa vận tốc góc, ta thu được:

$$\omega = \frac{v_R - v_L}{b}$$

$$R = \frac{b.(v_R + v_L)}{2(v_R - v_L)}$$

Sử dụng phương trình vận tốc góc, vận tốc tức thời V của điểm nằm giữa các bánh xe của robot được tính bởi:

$$V = \omega.R = \frac{v_R + v_L}{2}$$

Vận tốc tiếp tuyến của bánh xe cũng có thể được viết là:

$$v_R = r.\omega_L$$

$$v_L = r.\omega_L$$

Trong đó: ωR và ωL - Vận tốc góc trái và phải của các bánh xe quanh trục của chúng. Do đó, động học của robot trong tọa độ cơ thể cục bộ có thể được viết là:

$$\begin{bmatrix} \dot{x}_B \\ \dot{y}_B \\ \dot{\varphi} \end{bmatrix} = \begin{bmatrix} \frac{r}{2} & \frac{r}{2} \\ 0 & 0 \\ -\frac{r}{b} & \frac{r}{b} \end{bmatrix} \begin{bmatrix} \omega_L \\ \omega_R \end{bmatrix}$$

Bằng cách sử dụng phép biến đổi tọa độ (phép xoay trục), cuối cùng có thể thu được mô hình động học của robot theo tọa độ tổng thể như:

$$\begin{bmatrix} \dot{x}_B \\ \dot{y}_B \\ \dot{\varphi} \end{bmatrix} = \begin{bmatrix} \cos\varphi & 0 \\ \sin\varphi & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} V \\ \omega \end{bmatrix}$$

Sử dụng quan hệ R = V/ω và ωR = vR/r ta thu được phương trình tính vận tốc góc của

bánh xe bên phải ωR :

$$\omega_R = \frac{V + \omega.b/2}{r}$$

Quy trình tương tự có thể được áp dụng để tính vận tốc góc của bánh xe bên trái ωL .

$$\omega_R = \frac{V - \omega.b/2}{r}$$

## 4. Mô hình hóa trong ROS (URDF)

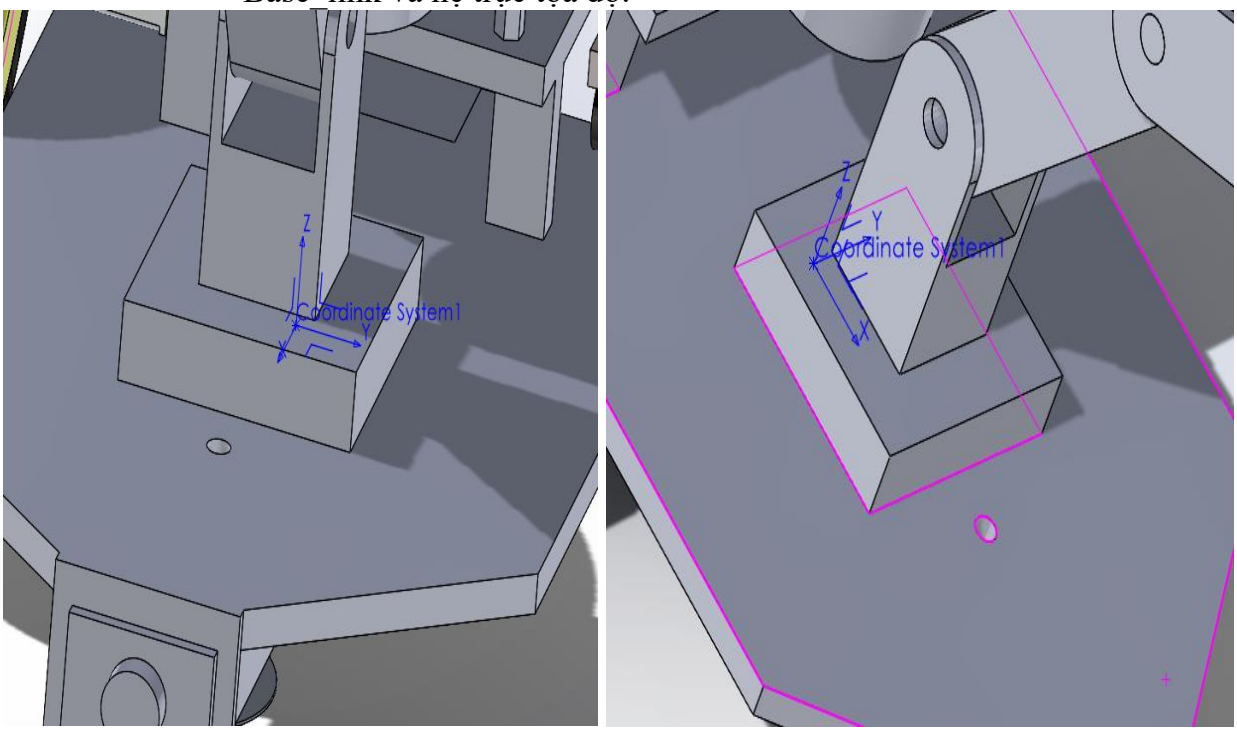
Mô hình được thiết kế trong Solidworks sao cho đúng với tỉ lệ thực tế nhất có thể từ đó đặt hệ trục tọa độ theo quy ước:

- Trục X: hướng về phía trước của Robot
- Trục Y: hướng về bên phải của Robot
- Trục Z: hướng lên trên của Robot

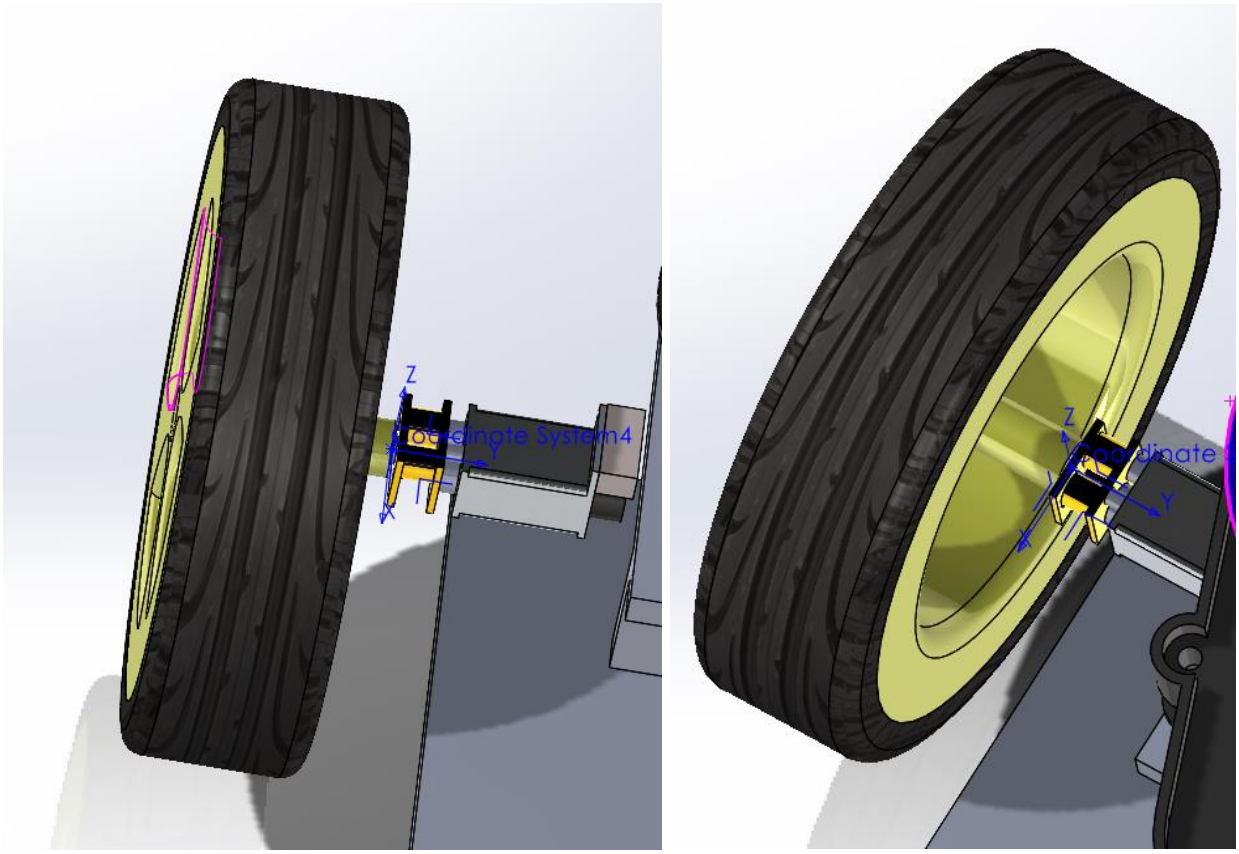
Tất cả các bánh xe và cảm biến đều được gắn với quy ước này.

### *4.1. Hệ trục tọa độ trong Solidworks*

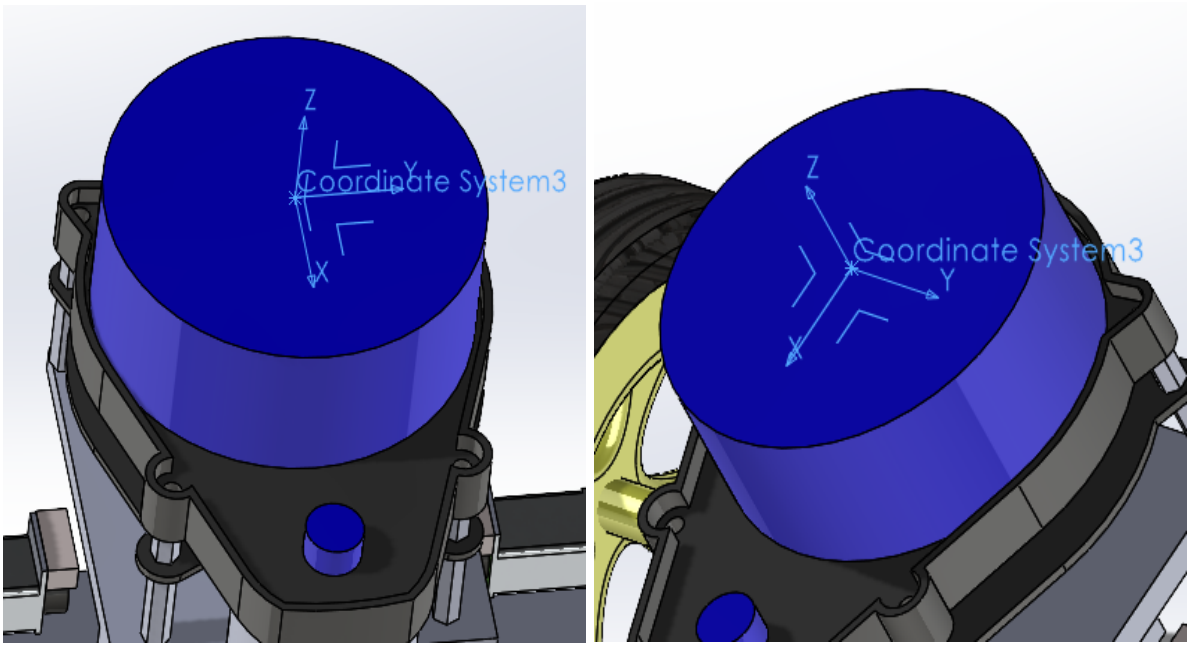
- Base_link và hệ trục tọa độ:



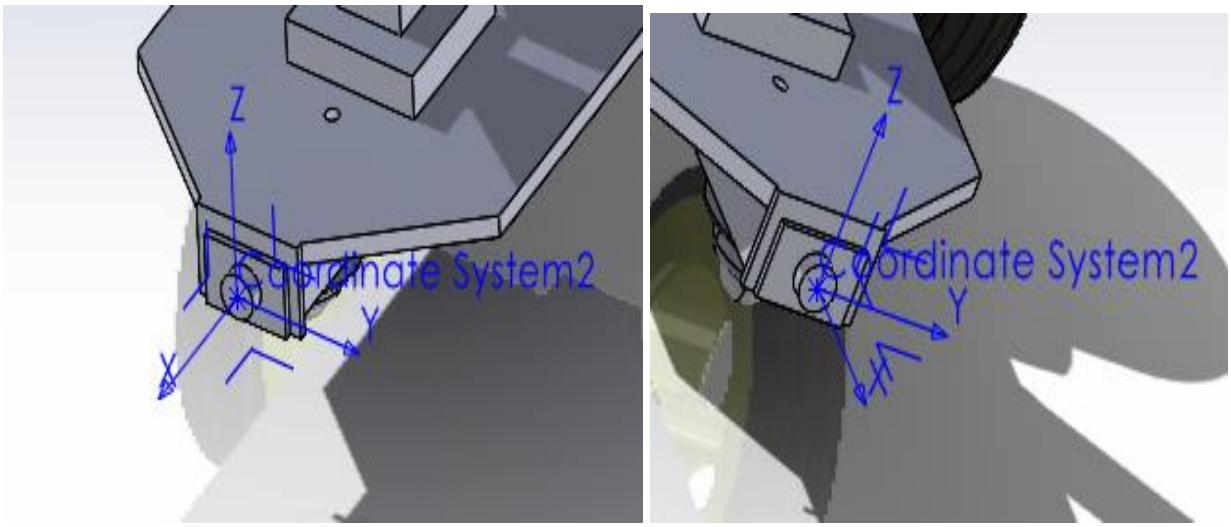
- Wheel_link và hệ trục tọa độ:



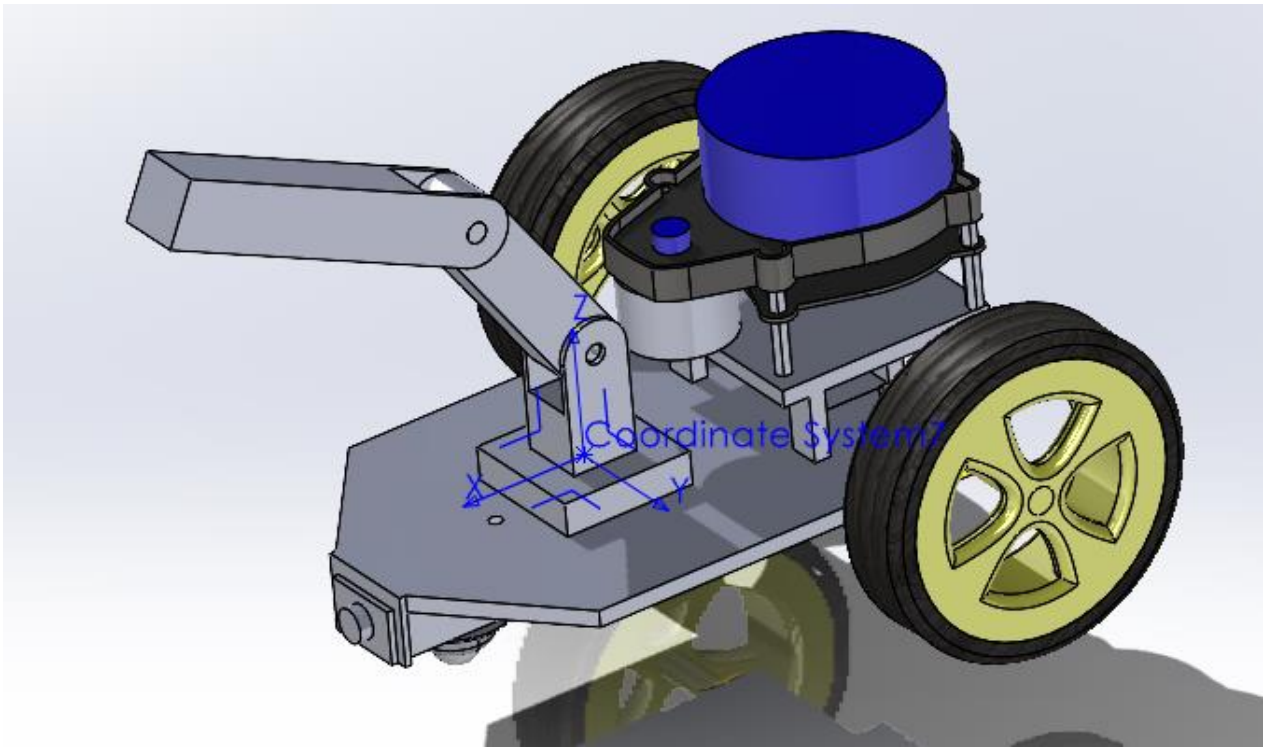

- Lidar_link và hệ trục tọa độ:

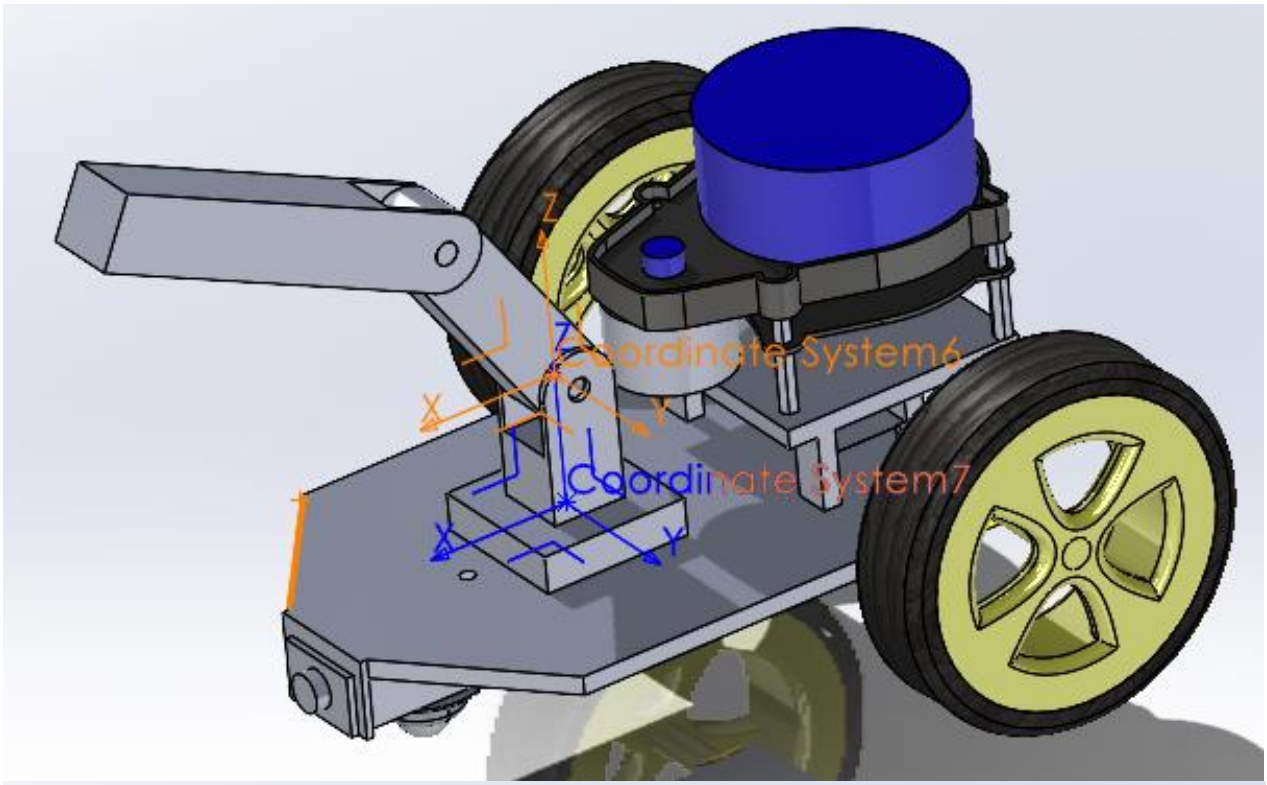


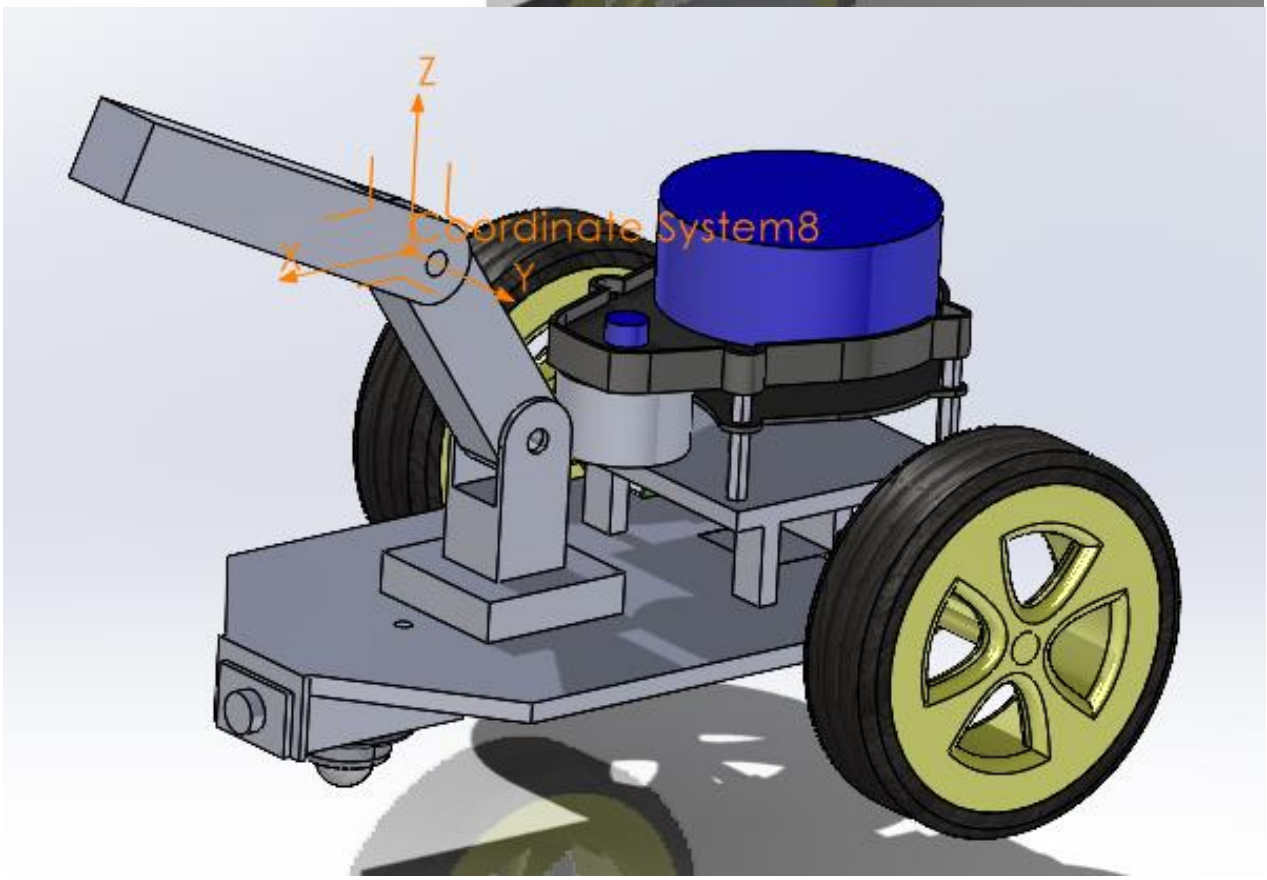
- Camera_link và hệ trục tọa độ:




- Arm_link và trục tọa độ:

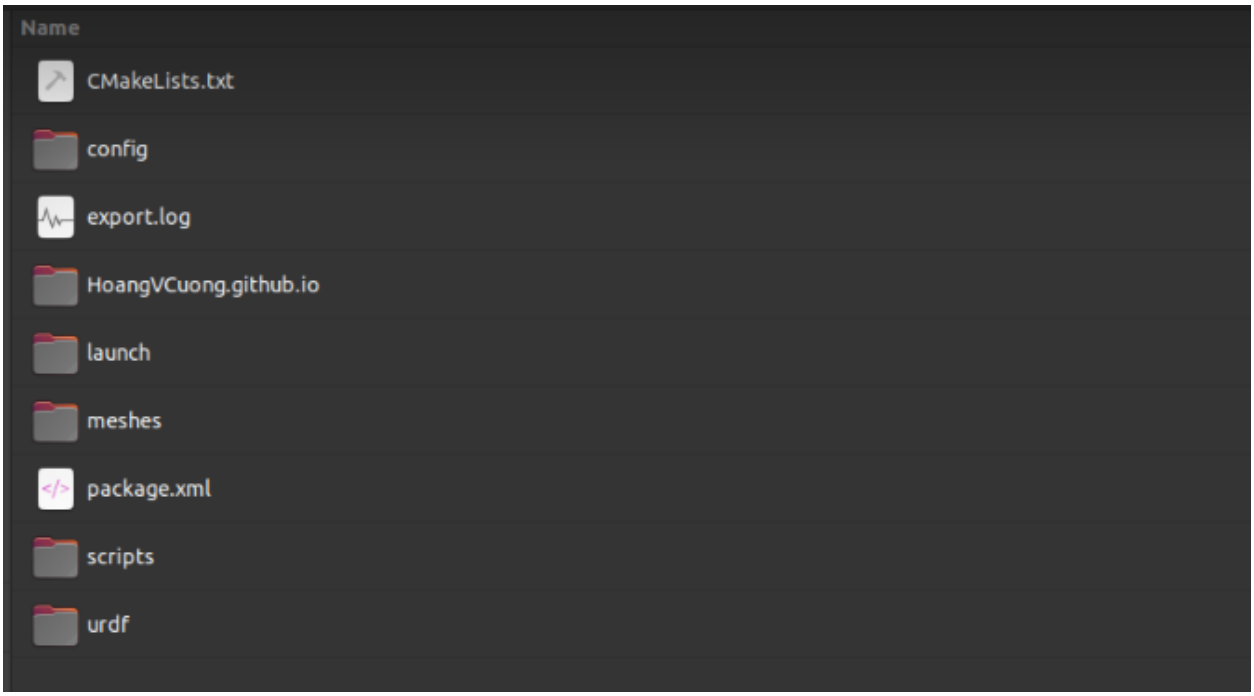



Z
Z
Coordinate System6
X
Y
Coordinate System7
X
Y


Z
Coordinate System8
X
Y

### *4.2. URDF*

- Cấu trúc các thư mục:

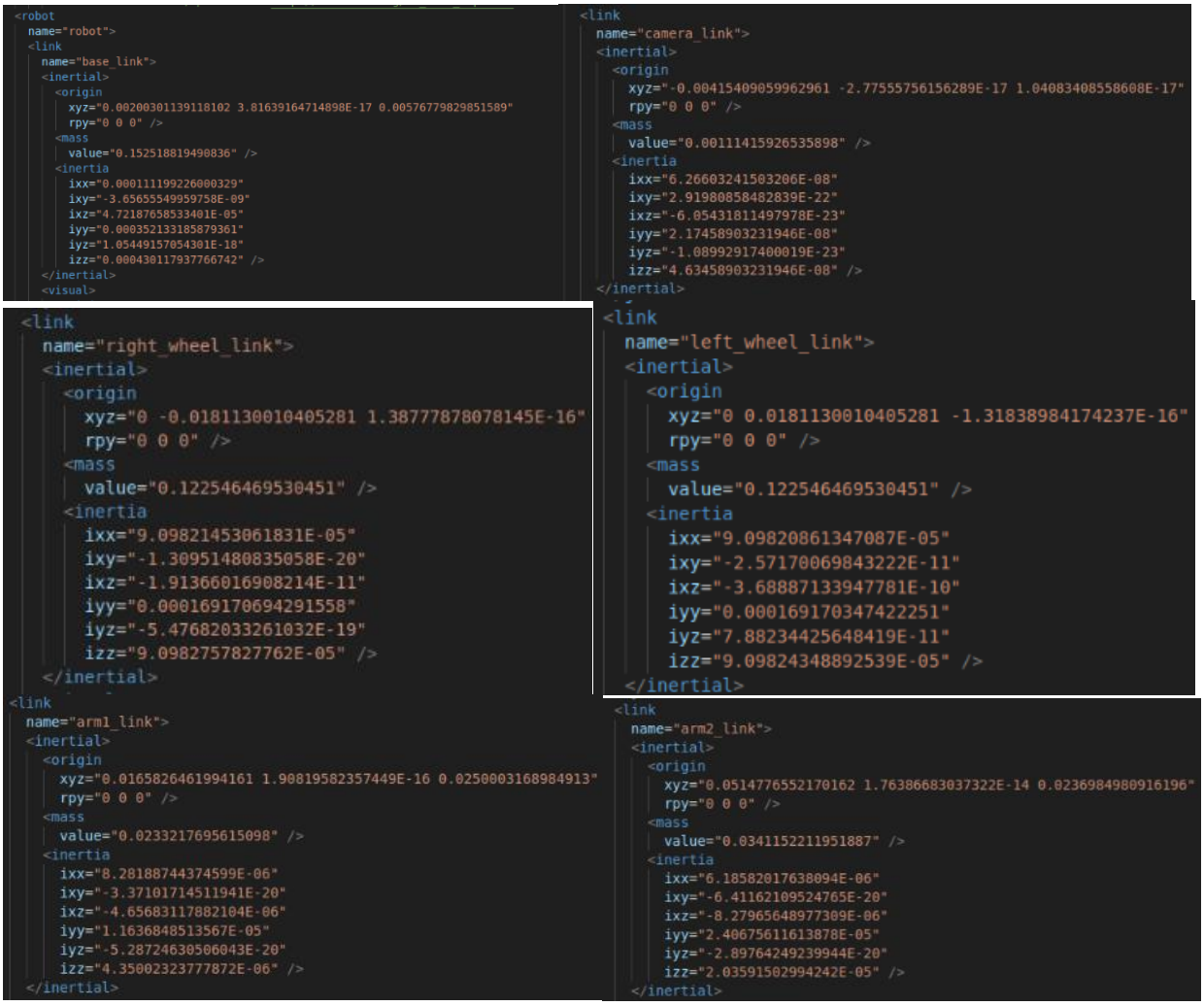



+ **Config**: chứa các file YAML (control.yaml...), các file cấu hình
+ **launch:** chứa các file .launch để khởi chạy mô phỏng Gazebo, Rviz và các node điều khiển.
+ **Meshes:** chứa các file mô hình 3D ( .stl) của Robot
+ **Scripts:** Chứa các file python (control.py, encoder.py...) dùng để điều khiển Robot
+ **urdf:** chứa file mô tả Robot
+ **CmakeLists.txt & package.xml:** các file cần thiết để buil package trong ROS

- ***Phân tích file URDF***: File này chứa các thông tin về các bộ phận cứng (link), các khớp nối (joint) và các thuộc tính về hình học, quán tính, va chạm, cảm biến và các liên kết với Gazebo để mô phỏng.

Trong file urdf của tôi (robot.urdf) có các thành phần chính:

+ **base_link**: thân chính của robot
+ **right_wheel_link, left_wheel_link**: 2 bánh vi sai
+ **lidar_link**: cảm biến Lidar
+ **camera_link:** Camera
+ **arm1_link, arm2_link**: 2 tay máy của Robot




### *4.3. Các plugin cảm biến*

a. Cảm biến Lidar

```
<!-- 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>
```

Mục đích:

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

b. Camera

```
<!-- camera -->
<gazebo reference="camera_link">
  <sensor type="camera" name="camera1">
    <update_rate>30.0</update_rate>
    <camera name="head">
      <horizontal_fov>1.3962634</horizontal_fov>
      <image>
        <width>800</width>
        <height>800</height>
        <format>R8G8B8</format>
      </image>
      <clip>
        <near>0.02</near>
        <far>300</far>
      </clip>
      <noise>
        <type>gaussian</type>
        <mean>0.0</mean>
        <stddev>0.007</stddev>
      </noise>
    </camera>
    <plugin name="camera_controller" filename="libgazebo_ros_camera.so">
      <alwaysOn>true</alwaysOn>
      <updateRate>0.0</updateRate>
      <cameraName>rrbot/camera1</cameraName>
      <imageTopicName>image_raw</imageTopicName>
      <cameraInfoTopicName>camera_info</cameraInfoTopicName>
      <frameName>camera_link</frameName>
      <hackBaseline>0.07</hackBaseline>
      <distortionK1>0.0</distortionK1>
      <distortionK2>0.0</distortionK2>
      <distortionK3>0.0</distortionK3>
```

Mục đích:

- Giúp robot có khả năng thu thập hình ảnh/video từ môi trường mô phỏng.
- Cho phép điều chỉnh độ phân giải, tần số khung hình, góc nhìn, tiêu cự
- Hỗ trợ mô phỏng nhiễu, méo hình, và các hiệu ứng quang học khác.
- Gửi hình ảnh/video qua topic ROS để xử lý thị giác máy tính.
- Dùng để huấn luyện mô hình nhận diện, điều hướng, và xử lý hình ảnh.
-

## 5. Điều khiển trong ROS

### *5.1. Plugin gazebo_ros_control*

Plugin gazebo_ros_control kết nối Gazebo với ROS Control thông qua một giao diện phần cứng mô phỏng.

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

### 5.2. *Plugin điều khiển 2 bánh xe*

```
<!-- wheel -->
<gazebo>
  <plugin name="differential_drive_controller" filename="libgazebo_ros_diff_drive.so">
    <updateRate>10</updateRate>
    <leftJoint>left_wheel_joint</leftJoint>
    <rightJoint>right_wheel_joint</rightJoint>
    <wheelSeparation>0.1334</wheelSeparation>
    <wheelDiameter>0.2410</wheelDiameter>
    <wheelAcceleration>1.0</wheelAcceleration>
    <wheelTorque>20</wheelTorque>
    <commandTopic>cmd_vel</commandTopic>
    <odometryTopic>odom</odometryTopic>
    <odometryFrame>odom</odometryFrame>
    <robotBaseFrame>base_link</robotBaseFrame>
    <odometrySource>1</odometrySource>
    <publishWheelTF>false</publishWheelTF>
    <publishOdom>true</publishOdom>
    <publishWheelJointState>true</publishWheelJointState>
    <legacyMode>false</legacyMode>
    <rosDebugLevel>Debug</rosDebugLevel>
  </plugin>
</gazebo>
```

### 5.3. *Định nghĩa các Tranmission trong URDF*

- Định nghĩa các tranmission tay máy:

```
  <!-- transmission -->
<transmission name="arm1_transmission">
    <type>transmission_interface/SimpleTransmission</type>
    <joint name="arm1_joint">
        <hardwareInterface>hardware_interface/PositionJointInterface</hardwareInterface>
    </joint>
    <actuator name="arm1_actuator">
        <mechanicalReduction>1.0</mechanicalReduction>
    </actuator>
</transmission>

<transmission name="arm2_transmission">
    <type>transmission_interface/SimpleTransmission</type>
    <joint name="arm2_joint">
        <hardwareInterface>hardware_interface/PositionJointInterface</hardwareInterface>
    </joint>
    <actuator name="arm2_actuator">
        <mechanicalReduction>1.0</mechanicalReduction>
    </actuator>
</transmission>
```

Mục đích:

- 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).
- 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).

- 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).

### *5.4. Cấu hình các file YAML*

```yaml
arm_1_joint_controller:
  type: "position_controllers/JointPositionController"
  joint: "arm1_joint"
  pid: {p: 10.0, i: 0.05, d: 1.0}

arm_2_joint_controller:
  type: "position_controllers/JointPositionController"
  joint: "arm2_joint"
  pid: {p: 10.0, i: 0.05, d: 1.0}

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

- position_controllers: Điều khiển vị trí cho 2 khớp tay máy.

### *5.5. Các file Launch*

Lệnh khởi chạy gazebo và Rviz : ***roslaunch robot display.launch***

File gazebo.launch:

launch > gazebo.launch

```xml
<launch>
  <include
    file="$(find gazebo_ros)/launch/empty_world.launch" />
  <node
    name="tf_footprint_base"
    pkg="tf"
    type="static_transform_publisher"
    args="0 0 0 0 0 0 base_link base_footprint 40" />
  <node
    name="spawn_model"
    pkg="gazebo_ros"
    type="spawn_model"
    args="-file $(find robot)/urdf/robot.urdf -urdf -model robot"
    output="screen" />
  <node
    name="fake_joint_calibration"
    pkg="rostopic"
    type="rostopic"
    args="pub /calibrated std_msgs/Bool true" />
</launch>
```

File display.launch:

```
launch > display.launch
<launch>
  <include
    file="$(find gazebo_ros)/launch/empty_world.launch" />
  <arg
    name="model" />
  <param
    name="robot_description"
    textfile="$(find robot)/urdf/robot.urdf" />
  <!-- <node
    name="joint_state_publisher_gui"
    pkg="joint_state_publisher_gui"
    type="joint_state_publisher_gui" /> -->
  <node
    name="joint_state_publisher"
    pkg="joint_state_publisher"
    type="joint_state_publisher" />
  <node
    name="robot_state_publisher"
    pkg="robot_state_publisher"
    type="robot_state_publisher" />
  <node
    name="rviz"
    pkg="rviz"
    type="rviz"
    args="-d $(find robot)/config/chay.rviz" />
  <node
    name="spawn_model"
    pkg="gazebo_ros"
    type="spawn_model"
    args="-file $(find robot)/urdf/robot.urdf -urdf -model robot"
    output="screen" />
  <node
    name="fake_joint_calibration"
    pkg="rostopic"
    type="rostopic"
    args="pub /calibrated std_msgs/Bool true" />
  <rosparam file="$(find robot)/config/control.yaml" command="load"/>

<node name="controller_spawner" pkg="controller_manager" type="spawner" respawn="false" output="screen"
      args="joint_state_controller arm_1_joint_controller arm_2_joint_controller" />

</launch>
```

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

### *5.6. Scripts Python điều khiển*

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

import rospy
from geometry_msgs.msg import Twist
from std_msgs.msg import Float64
import sys, termios, tty

LINEAR_SPEED = 0.2
ANGULAR_SPEED = 1

ARM_STEP = 0.1

NAMESPACE = ""

KEY_MAPPING = {
    'w': ('wheel', [LINEAR_SPEED, 0.0]),    # tien
    's': ('wheel', [-LINEAR_SPEED, 0.0]),   # lui
    'a': ('wheel', [0.0, ANGULAR_SPEED]),   # Quay trai
    'd': ('wheel', [0.0, -ANGULAR_SPEED]),  # Quay phai
    'i': ('arm1', ARM_STEP),                # Tang arm1
    'k': ('arm1', -ARM_STEP),               # Giam arm1
    'j': ('arm2', ARM_STEP),                # Tang arm2
    'l': ('arm2', -ARM_STEP),               # Giam arm2
    'q': ('exit', None)                     # out
}
```

```python
def get_key():
    fd = sys.stdin.fileno()
    old_settings = termios.tcgetattr(fd)
    try:
        tty.setraw(sys.stdin.fileno())
        key = sys.stdin.read(1)
    finally:
        termios.tcsetattr(fd, termios.TCSADRAIN, old_settings)
    return key

def teleop_control():
    rospy.init_node("teleop_robot", anonymous=True)

    # Publisher
    wheel_pub = rospy.Publisher(f"{NAMESPACE}/cmd_vel", Twist, queue_size=10)
    arm1_pub = rospy.Publisher(f"{NAMESPACE}/arm_1_joint_controller/command", Float64, queue_size=10)
    arm2_pub = rospy.Publisher(f"{NAMESPACE}/arm_2_joint_controller/command", Float64, queue_size=10)
```

```python
rate = rospy.Rate(10)
rospy.sleep(1)

if wheel_pub.get_num_connections() == 0:
    rospy.logwarn("Khong co subscriber nao ket noi toi /cmd_vel!")
else:
    rospy.loginfo("da ket noi toi /cmd_vel")
if arm1_pub.get_num_connections() == 0:
    rospy.logwarn("khong co subscriber nao ket noi toi /arm_1_joint_controller/command!")
else:
    rospy.loginfo("da ket noi toi /arm_1_joint_controller/command")
if arm2_pub.get_num_connections() == 0:
    rospy.logwarn("khong co subscriber nao ket noi toi /arm_2_joint_controller/command!")
else:
    rospy.loginfo("da ket noi toi /arm_2_joint_controller/command")

rospy.loginfo("=== TELEOP CONTROL ===")
rospy.loginfo("W - Tien | S - Lui | A - Quay trai | D - Quay phai")
rospy.loginfo("I - Tang arm1 | K - Giam arm1 | J - Tang arm2 | L - Giam arm2")
rospy.loginfo("Q - out")

twist = Twist()
arm1_pos = 0.0
arm2_pos = 0.0
```

```
    twist = Twist()
    arm1_pos = 0.0
    arm2_pos = 0.0

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

        if key in KEY_MAPPING:
            action_type, value = KEY_MAPPING[key]

            if action_type == 'exit':
                twist.linear.x = 0.0
                twist.angular.z = 0.0
                wheel_pub.publish(twist)
                rospy.loginfo("Thoát chương trình!")
                break

            elif action_type == 'wheel':
                twist.linear.x = value[0]
                twist.angular.z = value[1]
                wheel_pub.publish(twist)
                rospy.loginfo(f"Wheel - Linear: {twist.linear.x}, Angular: {twist.angular.z}")

            elif action_type == 'arm1':
                arm1_pos += value
                arm1_pos = max(-1.57, min(1.57, arm1_pos))
                arm1_pub.publish(Float64(arm1_pos))
                rospy.loginfo(f"Arm1 position: {arm1_pos:.2f} rad")

            elif action_type == 'arm2':
                arm2_pos += value
                arm2_pos = max(-1.57, min(1.57, arm2_pos))
                arm2_pub.publish(Float64(arm2_pos))
                rospy.loginfo(f"Arm2 position: {arm2_pos:.2f} rad")

        elif key == '\x03':  # Ctrl+C
            twist.linear.x = 0.0
            twist.angular.z = 0.0
            wheel_pub.publish(twist)
            rospy.loginfo("Thoát bằng Ctrl+C!")
            break

        rate.sleep()

if __name__ == "__main__":
    try:
        teleop_control()
    except rospy.ROSInterruptException:
        rospy.loginfo("da dung boi ROS Interrupt")
    except Exception as e:
        rospy.logerr(f"Loi xay ra: {str(e)}")
```

Giải thích code python:

- Dùng bàn phím để điều khiển bánh xe và tay máy
- Hàm def get_key(): Lấy ký tự đầu vào từ bàn phím mà không cần nhấn "Enter"
- Hàm def teleop_control(): khởi tạo node ROS với tên teleop_robot.
- Khởi tạo các Publisher: - ***wheel_pub****: Gửi lệnh vận tốc (Twist) đến* ***topic /cmd_vel****.*

***- arm1_pub****: Gửi lệnh góc (Float64) đến* ***topic/arm_1_joint_controller/command****.*

***- arm2_pub****: Gửi lệnh góc (Float64) đến* ***topic/arm_2_joint_controller/command****.*

- Kiểm tra kết nối Subscriber:

```
rospy.sleep(1)

if wheel_pub.get_num_connections() == 0:
   rospy.logwarn("Khong co subscriber nao ket noi toi /cmd_vel!")
else:
   rospy.loginfo("da ket noi toi /cmd_vel")
```

- Vòng lặp điều khiển:

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

   if key in KEY_MAPPING:
      action_type, value = KEY_MAPPING[key]
```

- Xử lý điều khiển: *+ Dừng robot và thoát vòng lặp*
  *+ Cập nhật tốc độ tuyến tính (linear.x) hoặc góc (angular.z).*
  *+ Gửi lệnh đến topic /cmd_vel.*
  *+ Cập nhật góc arm1, giới hạn từ -1.57 đến 1.57 rad.*
  *+ Gửi lệnh đến /arm_1_joint_controller/command.*
- Chạy chương trình: Gọi teleop_control(), bắt lỗi nếu có.

### *5.7. Hiển thị encoder*

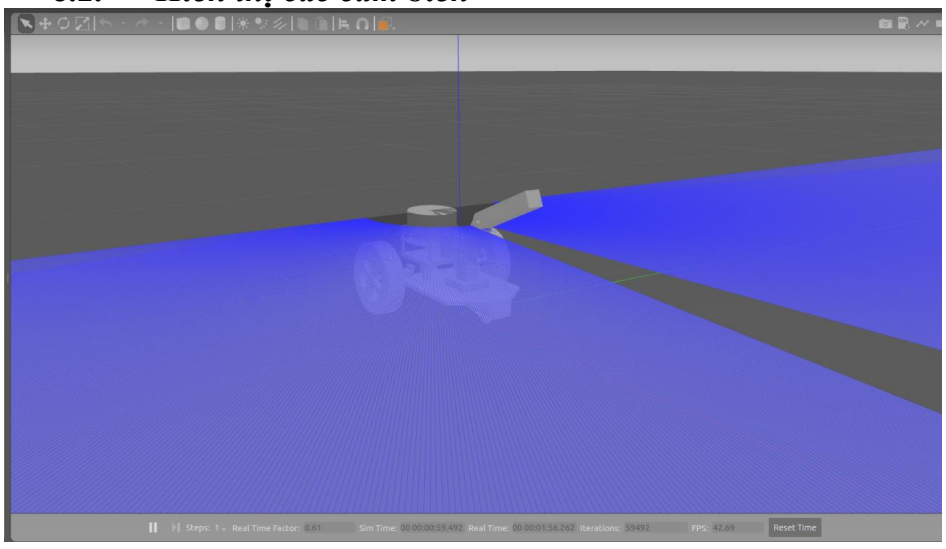
```python
scripts > encoder.py > ...
#!/usr/bin/env python3

import rospy
from nav_msgs.msg import Odometry

def encoder_callback(msg):
    # Lay du lieu tu encoder (Odometry)
    x = msg.pose.pose.position.x
    y = msg.pose.pose.position.y

    # Lay giu lieu van toc tu encoder
    linear_velocity = msg.twist.twist.linear.x
    angular_velocity = msg.twist.twist.angular.z

    # hien thi thong tin
    rospy.loginfo("Vị trí: x = {:.3f}, y = {:.3f}".format(x, y))
    rospy.loginfo("Vận tốc: Linear = {:.3f}, Angular = {:.3f}".format(linear_velocity, angular_velocity))

def encoder_listener():
    rospy.init_node('encoder_listener', anonymous=True)

    rospy.Subscriber("/odom", Odometry, encoder_callback)

    rospy.spin()

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

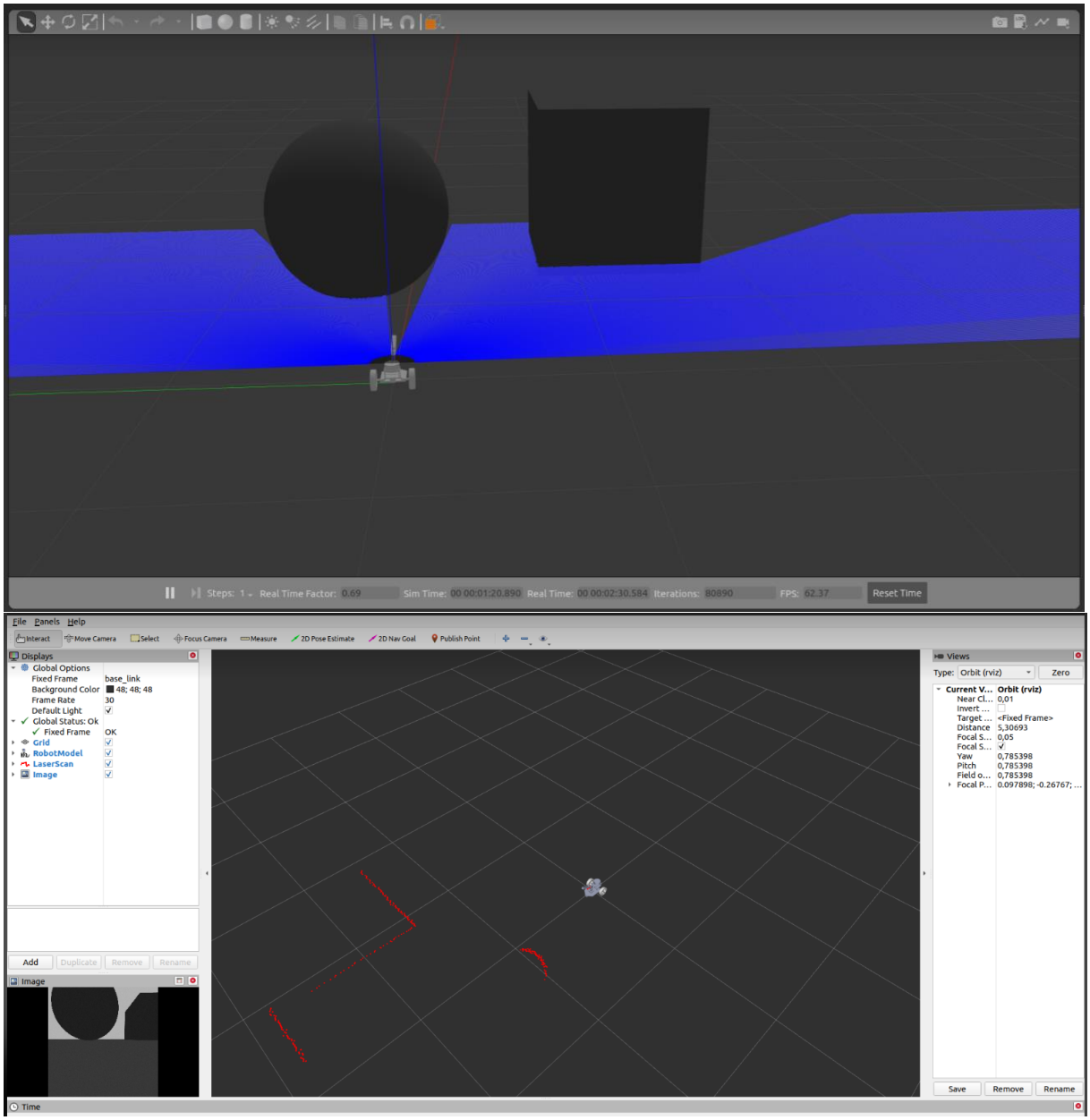
Giải thích code:

- Thêm các thư viện cần sử dụng
- Hàm def encoder_callback(msg):
  *+ Lấy tọa độ hiện tại (x, y) của robot từ* ***pose.pose.position****.*
  *+ Lấy vận tốc tuyến tính (linear.x) và vận tốc góc (angular.z) từ* ***twist.twist****.*
  *+ Hiển thị thông tin vị trí và vận tốc lên terminal với 3 chữ số thập phân.*
- *Hàm def encoder_listener(): :*
  *+ Khởi tạo node ROS với tên* ***encoder_listener.***
  *+ anonymous=True: Cho phép ROS tự*
  *động đổi tên node nếu có node khác cùng tên.*
  *+ Đăng ký Subscriber để lắng nghe* ***topic /odom*** *và gọi hàm* ***encoder_callback()*** *khi có dữ liệu mới*
  *+ Duy trì vòng lặp ROS để lắng nghe*
  *dữ liệu liên tục.*

## 6. Mô phỏng trong Rviz và Gazebo

### *6.1. Hiển thị các cảm biến*

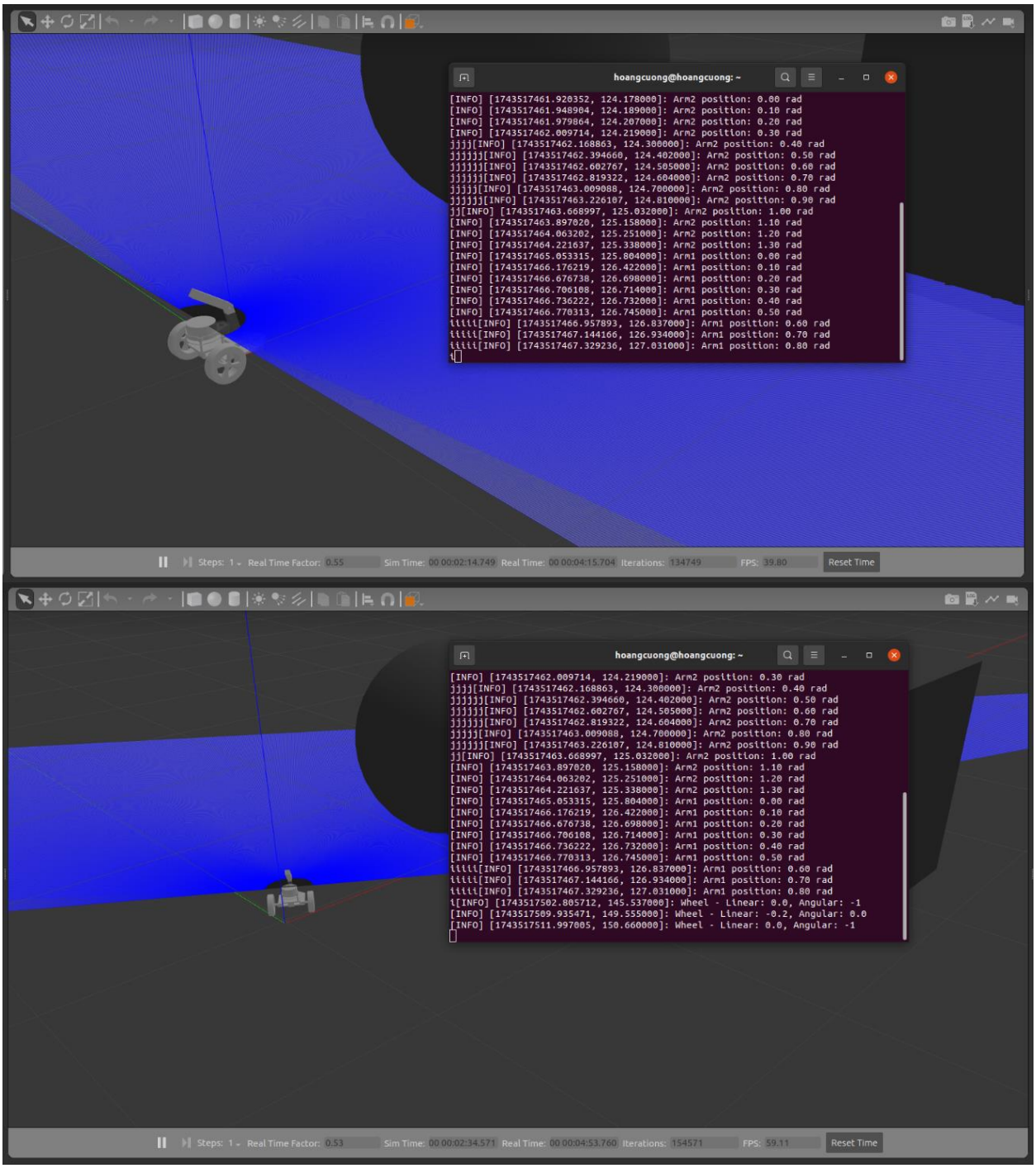





## 6.2. *Hiển thị encoder*

```
hoangcuong@hoangcuong:~$ rosrun robot encoder.py
[INFO] [1743517283.331470, 28.761000]: Vị trí: x = -0.003, y = -0.001
[INFO] [1743517283.339317, 28.765000]: Vận tốc: Linear = -0.000, Angular = -0.000
[INFO] [1743517283.531031, 28.861000]: Vị trí: x = -0.003, y = -0.001
[INFO] [1743517283.542265, 28.868000]: Vận tốc: Linear = -0.000, Angular = 0.000
[INFO] [1743517283.729745, 28.961000]: Vị trí: x = -0.003, y = -0.001
[INFO] [1743517283.734140, 28.964000]: Vận tốc: Linear = -0.000, Angular = 0.000
[INFO] [1743517283.925237, 29.061000]: Vị trí: x = -0.003, y = -0.001
[INFO] [1743517283.934055, 29.062000]: Vận tốc: Linear = -0.000, Angular = 0.000
[INFO] [1743517284.116918, 29.161000]: Vị trí: x = -0.003, y = -0.001
[INFO] [1743517284.122258, 29.162000]: Vận tốc: Linear = -0.000, Angular = 0.000
[INFO] [1743517284.312876, 29.262000]: Vị trí: x = -0.003, y = -0.001
[INFO] [1743517284.320426, 29.265000]: Vận tốc: Linear = -0.000, Angular = 0.000
[INFO] [1743517284.512868, 29.360000]: Vị trí: x = -0.003, y = -0.001
[INFO] [1743517284.519455, 29.365000]: Vận tốc: Linear = -0.001, Angular = 0.002
[INFO] [1743517284.711039, 29.461000]: Vị trí: x = -0.003, y = -0.001
[INFO] [1743517284.720946, 29.464000]: Vận tốc: Linear = -0.000, Angular = 0.000
[INFO] [1743517284.903097, 29.560000]: Vị trí: x = -0.003, y = -0.001
[INFO] [1743517284.910835, 29.564000]: Vận tốc: Linear = -0.000, Angular = 0.000
[INFO] [1743517285.113961, 29.662000]: Vị trí: x = -0.003, y = -0.001
[INFO] [1743517285.120398, 29.665000]: Vận tốc: Linear = -0.000, Angular = 0.000
```

## *6.3. Hiển thị điều khiển bánh xe và tay máy*



## 7. Kết luận

- Bài báo cáo đã hoàn thiện và đảm bảo yêu cầu đề ra, đảm bảo được khả năng di chuyển linh hoạt trên mô phỏng từ đó là nền tảng cho những dự án lớn hơn tiếp theo.
- Bài báo cáo cũng giúp cho người thực hiện hiểu rõ hơn khi được làm việc với những cảm biến như Lidar, Camera và Encoder cũng như điều khiển được bánh xe và tay máy.