BÁO CÁO PROJECT GIỮA KỲ

THIẾT KẾ , MÔ PHỎNG VÀ ĐIỀU KHIỂN MÔ HÌNH ROBOT BỐN BÁNH KẾT HỢP TAY MÁY VÀ CẢM BIẾN VỚI ROS VÀ GAZEBO

Thành viên	Họ và tên: Nguyễn Bảo Long	Mã sinh viên (ID): 22027546	
Mô hình robot	Loại chuyển động: *Car-like (ackerman-steering)*	Loại tay máy: *Rotation-rotation - grip*	Loại cảm biến: *Encoder- IMU-Lidar*
Tên/Địa chỉ Repo trên Github	*Github*.		
File mô hình Soliwork	*Drive*.		

Mục lục (Table of Contents)

1. Tổng quan (Introduction).

Báo cáo này trình bày về mô hình robot di chuyển theo nguyên lý Car-like (Ackermann Steering), kết hợp với một tay máy có hai khớp quay (rotation). Robot được trang bị các cảm biến IMU, LiDAR và Encoder để hỗ trợ điều hướng, nhận thức môi trường và kiểm soát chuyển động. IMU giúp đo lường gia tốc, xác định góc quay chính xác ; LiDAR có khả năng nhận biết môi trường xung quanh; Encoder giúp xác định chính xác vị trí của xe, đảm bảo điều khiển ổn định. Mô hình này sẽ được xây dựng và mô phỏng trên nền tảng ROS với sự hỗ trợ từ Gazebo.

2. Thiết kế mô hình trong SolidWork (Design)

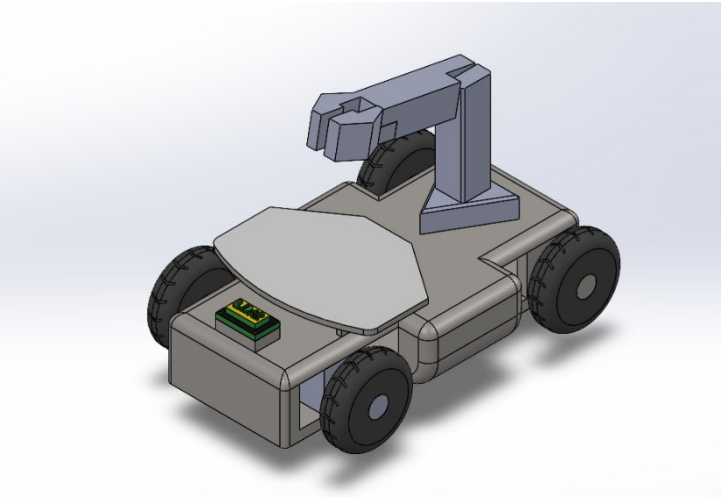
2.1. Yêu cầu đối với thiết kế.

- Robot cần có kích thước phù hợp, đảm bảo cân đối và ổn định khi di chuyển, đồng thời số lượng chi tiết nên được tối ưu để giảm độ phức tạp trong thiết kế và mô phỏng.

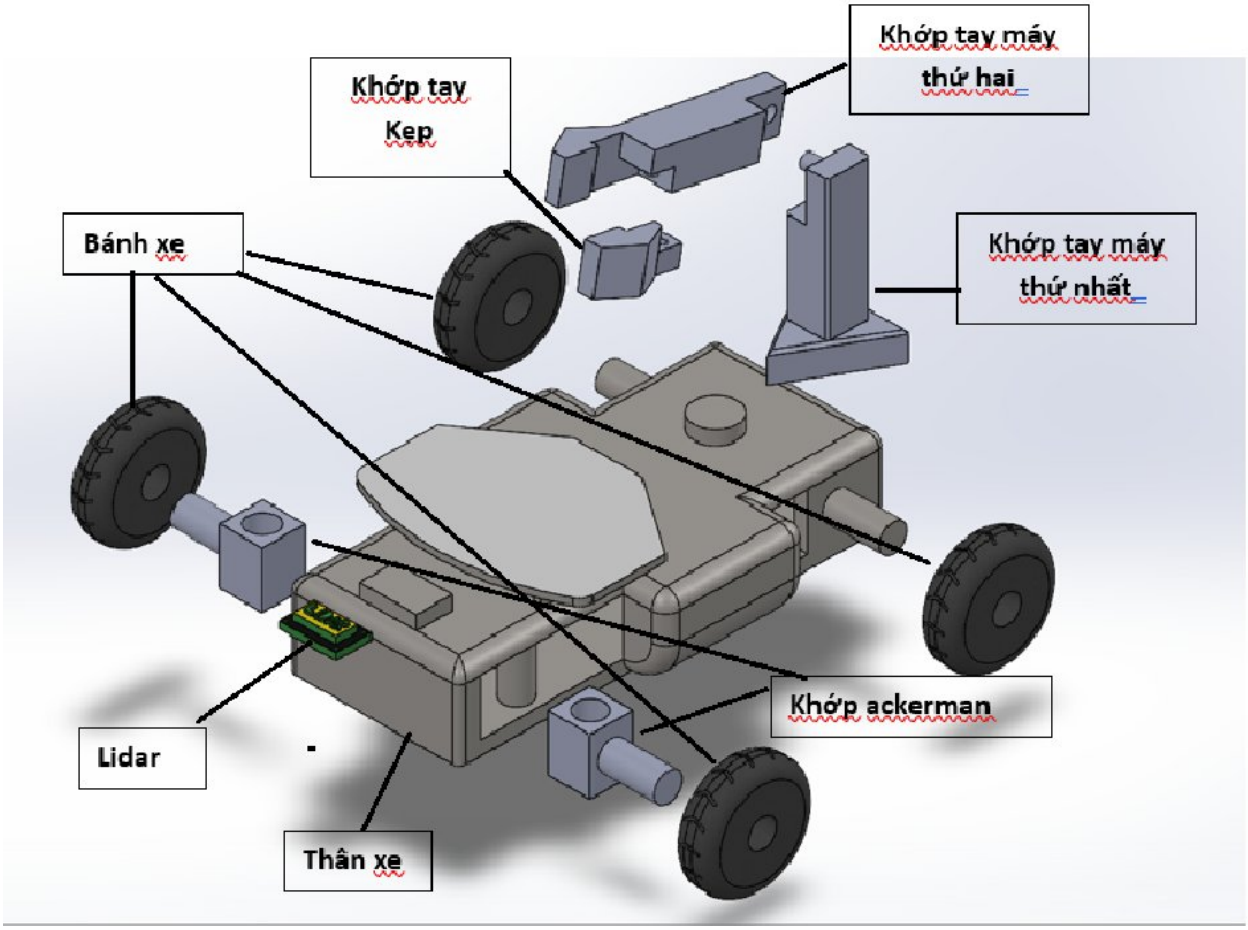
- Các cảm biến có kích thước lớn và phức tạp như LiDAR cần được tích hợp ngay từ đầu , trong khi các cảm biến nhỏ như IMU và Encoder có thể được thêm vào sau khi hoàn thiện mô hình URDF.

2.2. Chi tiết bản thiết kế trên SolidWork.

Mô hình thân xe được thiết kế lại dựa theo ý tưởng về robot di động Kuka Youbot , với phần cánh tay với 2 khớp xoay (revolute) khác hướng, đồng thời được tích hợp theo một tay gắp (gripper) với thiết kế đơn giản. Hệ thống bánh lái Ackerman Steering phức tạp được thay thế bằng hai trục quay dọc tại hai bên nhằm đơn giản hóa trong quá trình chuyển đổi sang URDF.



Chi tiết các bộ phân được thể hiện ở hình dưới đây:

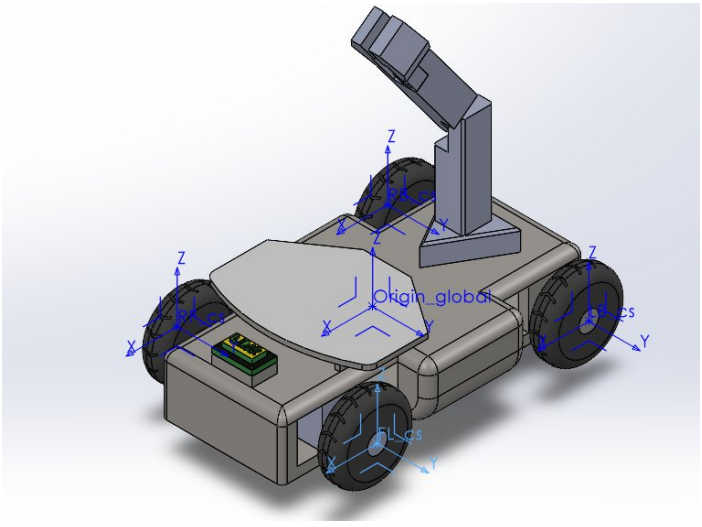


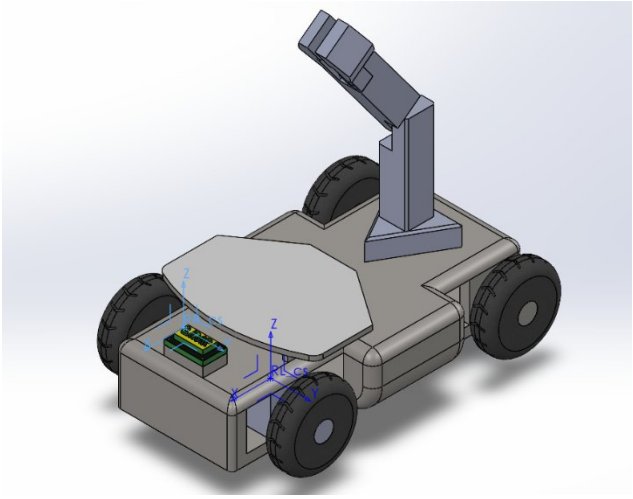
2.3. Cách đặt hệ trục tọa độ

Hệ trục tọa độ cho mô hình để xuất ra file urdf được đặt như sau:

- Với bốn bánh xe:

Đặt hệ trục tọa độ với tâm ở chính giữa bánh xe, trục x dọc theo thân xe theo chiều từ sau ra trước xe, trục z đặt dọc xe theo chiều từ dưới lên trên, với từng trục tham chiếu nằm ở chính giữa những bánh xe.



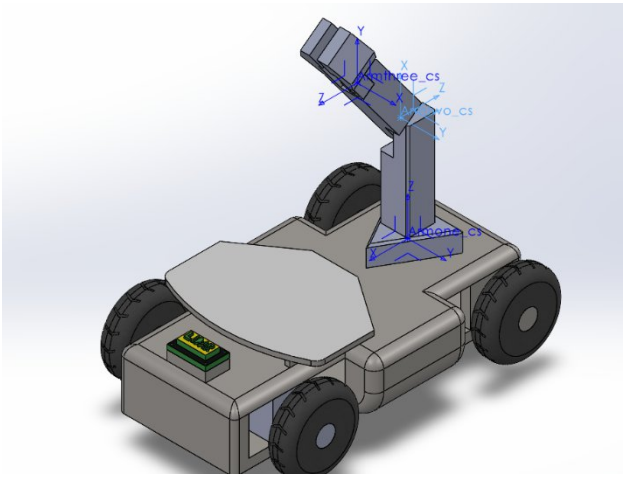


- Với trục Ackerman steering:

Đặt hệ trục tọa độ với tâm nằm ở chính giữa trục, trục z hướng dọc theo thân xe từ dưới lên trên, trục x dọc theo thân xe từ trước ra sau với trục tham chiếu đặt dọc theo trục z.

- Với tay máy:

Tay máy được đặt hệ trục tọa độ và trục tham chiếu dựa trên nguyên tắc đặt trục của Denavit-Hatenbert.

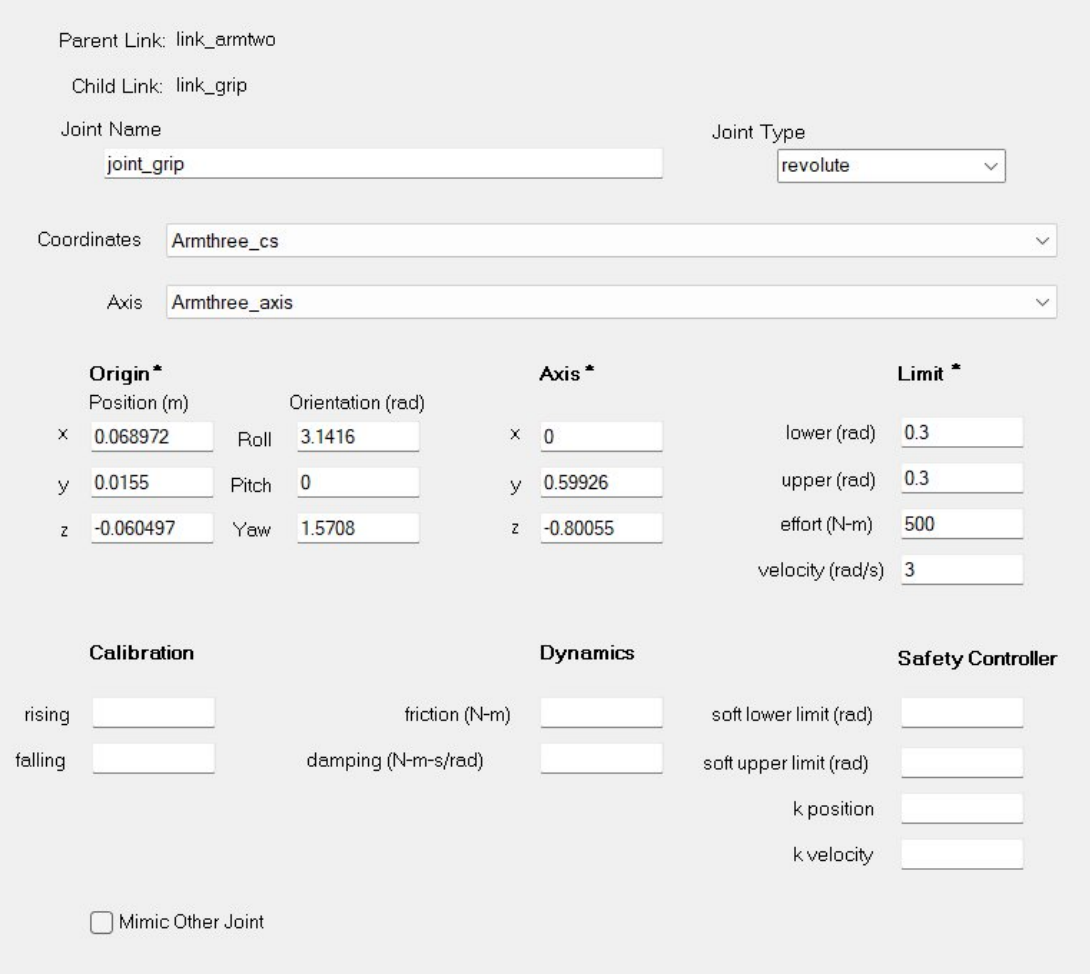


3. Điều chỉnh mô hình URDF (Config URDF).

3.1. Chuyển đổi từ file SolidWork sang URDF.

- Mở mô hình và chạy URDF Exporter.
- Định nghĩa các liên kết (links) và khớp nối (joints) trong giao diện của plugin.
- Chọn các trục quay, kiểu khớp (bốn bánh xe, continuous, hai trục ackerman steering và các khớp tay máy - revolutionn).

- Gán thông số vật lý (khối lượng, quán tính) và giới hạn góc quay cho từng bộ phận.



3.2. Các khớp và thuộc tính trong URDF.

(Toàn bộ file URDF có thể được xem tại đây: link)

3.2.1. Cấu trúc tổng thể:

Thẻ <robot name="ten">: Xác định tên của robot là "ten".

Tệp sử dụng định dạng XML với mã hóa UTF-8.

```
<?xml version="1.0" encoding="utf-8"?>
<robot name="ten">
```

3.2.2. Các link và các joint:

Mỗi <link> đại diện cho một thành phần vật lý của robot, bao gồm thông tin về:

- ***Inertial***: Đặc tính quán tính (khối lượng, moment quán tính) để mô phỏng động lực học. (*các link trong mô hình được điều chỉnh để có khối lượng khoảng 1kg/ cho mỗi chi tiết lớn và 0.5 kg cho mỗi chi tiết nhỏ*)
- ***Visual***: Hình dạng hiển thị trong mô phỏng (thường là tệp STL hoặc hình học cơ bản như hình trụ).
- ***Collision***: Hình dạng dùng để tính toán va chạm. (*trong file urdf này ta không khai báo thuộc tính này*)
- ***Gazebo***: Cấu hình bổ sung cho Gazebo (*như màu sắc*).

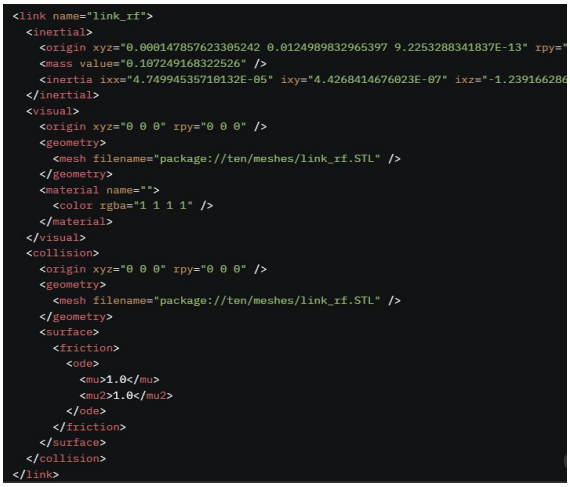
Các link:

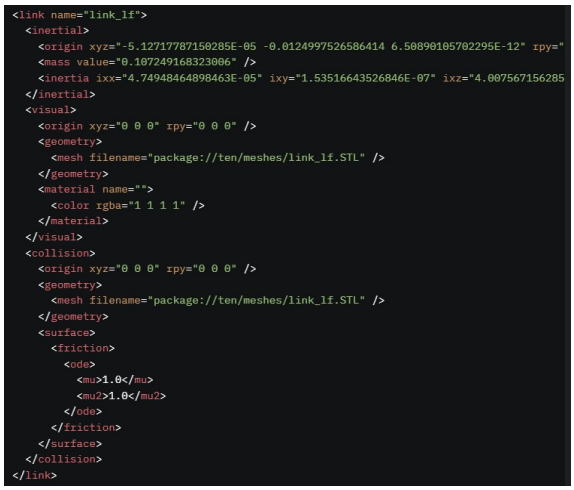
```
<link name="base_link">
  <inertial>
    <origin xyz="-0.00234108810956965 -2.0130803930053E-05 -0.0160690551998181" rpy="0
    <mass value="1.35087539538164" />
    <inertia ixx="0.0039080349016898" ixy="2.56534541584962E-06" ixz="-0.000216074327
  </inertial>
  <visual>
    <origin xyz="0 0 0" rpy="0 0 0" />
    <geometry>
      <mesh filename="package://ten/meshes/base_link.STL" />
    </geometry>
    <material name="1">
      <color rgba="0.792156862745098 0.819607843137255 0.933333333333333 1" />
    </material>
  </visual>
  <collision>
    <origin xyz="0 0 0" rpy="0 0 0" />
    <geometry>
      <mesh filename="package://ten/meshes/base_link.STL" />
    </geometry>
  </collision>
</link>
<gazebo reference="base_link">
  <material>Gazebo/LightGrey</material>
</gazebo>
```

```
<link name="link_rb">
  <inertial>
    <origin xyz="-3.80415143830248E-12 0.0124998578121212 -6.34490029960411E-12" rpy="
    <mass value="0.107249168322434" />
    <inertia ixx="4.74942167970264E-05" ixy="8.73960312790841E-15" ixz="8.433318469953
  </inertial>
  <visual>
    <origin xyz="0 0 0" rpy="0 0 0" />
    <geometry>
      <mesh filename="package://ten/meshes/link_rb.STL" />
    </geometry>
    <material name="">
      <color rgba="1 1 1 1" />
    </material>
  </visual>
  <collision>
    <origin xyz="0 0 0" rpy="0 0 0" />
    <geometry>
      <mesh filename="package://ten/meshes/link_rb.STL" />
    </geometry>
    <surface>
      <friction>
        <ode>
          <mu>1.0</mu>
          <mu2>1.0</mu2>
        </ode>
      </friction>
    </surface>
```

```
<link name="link_lb">
  <inertial>
    <origin xyz="3.12937176172312E-12 -0.0124998578121187 6.3729473087637E-12" rpy
    <mass value="0.107249168323003" />
    <inertia ixx="4.7494216791172E-05" ixy="9.81169308698789E-15" ixz="4.101868362
  </inertial>
  <visual>
    <origin xyz="0 0 0" rpy="0 0 0" />
    <geometry>
      <mesh filename="package://ten/meshes/link_lb.STL" />
    </geometry>
    <material name="">
      <color rgba="1 1 1 1" />
    </material>
  </visual>
  <collision>
    <origin xyz="0 0 0" rpy="0 0 0" />
    <geometry>
      <mesh filename="package://ten/meshes/link_lb.STL" />
    </geometry>
    <surface>
      <friction>
        <ode>
          <mu>1.0</mu>
          <mu2>1.0</mu2>
        </ode>
      </friction>
```

```
<link name="link_rr">
  <inertial>
    <origin xyz="-0.000144515368454431 -0.0122164500164231 -0.0199999999999994" rpy="
    <mass value="0.0360000000000001" />
    <inertia ixx="1.75187601980783E-05" ixy="-1.34621907445705E-07" ixz="1.3543381179
  </inertial>
  <visual>
    <origin xyz="0 0 0" rpy="0 0 0" />
    <geometry>
      <mesh filename="package://ten/meshes/link_rr.STL" />
    </geometry>
    <material name="">
      <color rgba="0.792156862745098 0.819607843137255 0.933333333333333 1" />
    </material>
  </visual>
  <collision>
    <origin xyz="0 0 0" rpy="0 0 0" />
    <geometry>
      <mesh filename="package://ten/meshes/link_rr.STL" />
    </geometry>
  </collision>
</link>
```





Mỗi <joint> mô tả cách các link được kết nối với nhau:

- ***Type***: Loại khớp (continuous: xoay vô hạn, revolute: xoay có giới hạn, fixed: cố định) (*tại mô hình này ta đặt bốn bánh xe là loại continuous, các joint còn lại là revolute*).
- ***Origin***: Vị trí và hướng của khớp so với link cha. (*đã được quy ước trong SolidWork to URDF*)
- ***Parent & Child***: Link cha và link con mà khớp kết nối. (*đã được quy ước trong SolidWork to URDF)*
- ***Axis***: Trục quay của khớp. (*đã được quy ước trong SolidWork to URDF*)
- ***Limit***: Giới hạn góc, lực, vận tốc .

Các joint:

```xml
<joint name="joint_rb" type="continuous">
  <origin xyz="-0.115 -0.1 -0.02" rpy="0 0 0" />
  <parent link="base_link" />
  <child link="link_rb" />
  <axis xyz="0 1 0" />
</joint>
<gazebo reference="link_rb">
  <material>Gazebo/White</material>
</gazebo>
```

```xml
<joint name="joint_lb" type="continuous">
  <origin xyz="-0.115 0.1 -0.02" rpy="0 0 0" />
  <parent link="base_link" />
  <child link="link_lb" />
  <axis xyz="0 1 0" />
</joint>
<gazebo reference="link_lb">
  <material>Gazebo/White</material>
</gazebo>
```

```xml
<joint name="joint_rr" type="revolute">
  <origin xyz="0.095 -0.045 0" rpy="0 0 0" />
  <parent link="base_link" />
  <child link="link_rr" />
  <axis xyz="0 0 -1" />
  <limit lower="0" upper="0" effort="5" velocity="2" />
</joint>
<gazebo reference="link_rr">
  <material>Gazebo/LightGrey</material>
</gazebo>
```

```xml
<joint name="joint_rf" type="continuous">
  <origin xyz="-0.00065058 -0.054996 -0.02" rpy="0 0 0" />
  <parent link="link_rr" />
  <child link="link_rf" />
  <axis xyz="0.011829 0.99993 0" />
</joint>
<gazebo reference="link_rf">
  <material>Gazebo/White</material>
</gazebo>
```

```xml
<joint name="joint_rl" type="revolute">
  <origin xyz="0.095 0.045 0" rpy="0 0 0" />
  <parent link="base_link" />
  <child link="link_rl" />
  <axis xyz="0 0 -1" />
  <limit lower="0" upper="0" effort="5" velocity="2" />
</joint>
<gazebo reference="link_rl">
  <material>Gazebo/LightGrey</material>
</gazebo>
```

```xml
<joint name="joint_lf" type="continuous">
  <origin xyz="0.0002256 0.055 -0.02" rpy="0 0 0" />
  <parent link="link_rl" />
  <child link="link_lf" />
  <axis xyz="-0.0041018 1 0" />
</joint>
<gazebo reference="link_lf">
  <material>Gazebo/White</material>
</gazebo>
```

```xml
<joint name="joint_armone" type="revolute">
  <origin xyz="-0.09 0 0.02" rpy="0 0 0" />
  <parent link="base_link" />
  <child link="link_armone" />
  <axis xyz="0 0 1" />
  <limit lower="-1.57" upper="1.57" effort="500" velocity="200" />
</joint>
<gazebo reference="link_armone">
  <material>Gazebo/LightGrey</material>
</gazebo>
```

```xml
<joint name="joint_grip" type="revolute">
  <origin xyz="0.091469 0 0.008" rpy="1.5708 0 0" />
  <parent link="link_armtwo" />
  <child link="link_grip" />
  <axis xyz="0.081822 -0.020685 -0.99643" />
  <limit lower="-0.2" upper="0.2" effort="500" velocity="2" />
</joint>
```

3.2.3. Các plugin cảm biến:

- IMU (cảm biến góc và gia tốc):

```xml
<gazebo reference="base_link">
  <sensor name="imu_sensor" type="imu">
    <always_on>true</always_on>
    <update_rate>20.0</update_rate>
    <pose>0 0 0 0 -1.5708 0</pose>
    <plugin name="imu_plugin" filename="libgazebo_ros_imu_sensor.so">
      <topicName>imu/data</topicName>
      <frameName>base_link</frameName>
      <gaussianNoise>0.0</gaussianNoise>
      <initialOrientationAsReference>false</initialOrientationAsReference>
    </plugin>
  </sensor>
</gazebo>
```

Vị trí: Cảm biến IMU được gắn vào base_link (phần thân chính của robot), như được chỉ định bởi thẻ <gazebo reference="base_link">.

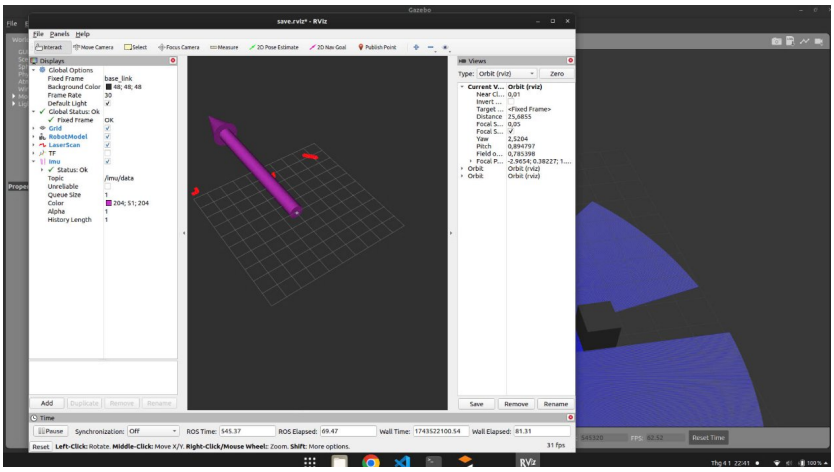
Cấu hình:

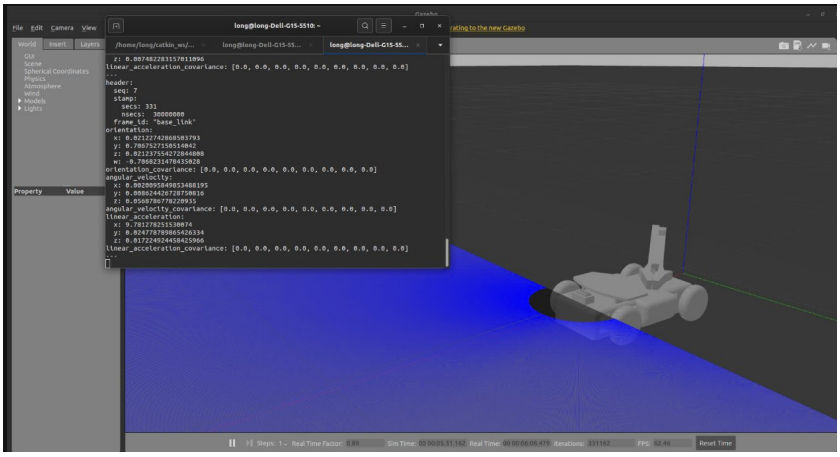
- Luôn bật (always_on=true), cập nhật 20 Hz (update_rate=20.0).
- Vị trí: Gốc của base_link (xyz=0 0 0), quay dọc theo chiều thân xe từ dưới lên trên.(rpy=0 -1.5708 0).
- Tham số: Không nhiễu (gaussianNoise=0.0), định hướng không cố định ban đầu (initialOrientationAsReference=false).

Để xuất dữ liệu từ cảm biến , ta xuất dữ liệu từ topic /imu/data:

- rostopic echo imu/data

Kết quả ta sẽ thu được dữ liệu như hình sau:





- Lidar:

```
<gazebo reference="lidar_link">
  <sensor type="ray" name="lidar_sensor">
    <pose>0 0 0 0 0 0</pose>
    <visualize>true</visualize>
    <update_rate>180</update_rate>
    <ray>
      <scan>
        <horizontal>
          <samples>720</samples>
          <resolution>1</resolution>
          <min_angle>-1.52</min_angle>
          <max_angle>1.52</max_angle>
        </horizontal>
      </scan>
      <range>
        <min>0.1</min>
        <max>5.0</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>
```

Cảm biến LIDAR (lidar_sensor) gắn vào lidar_link, quét khoảng cách bằng tia laser.

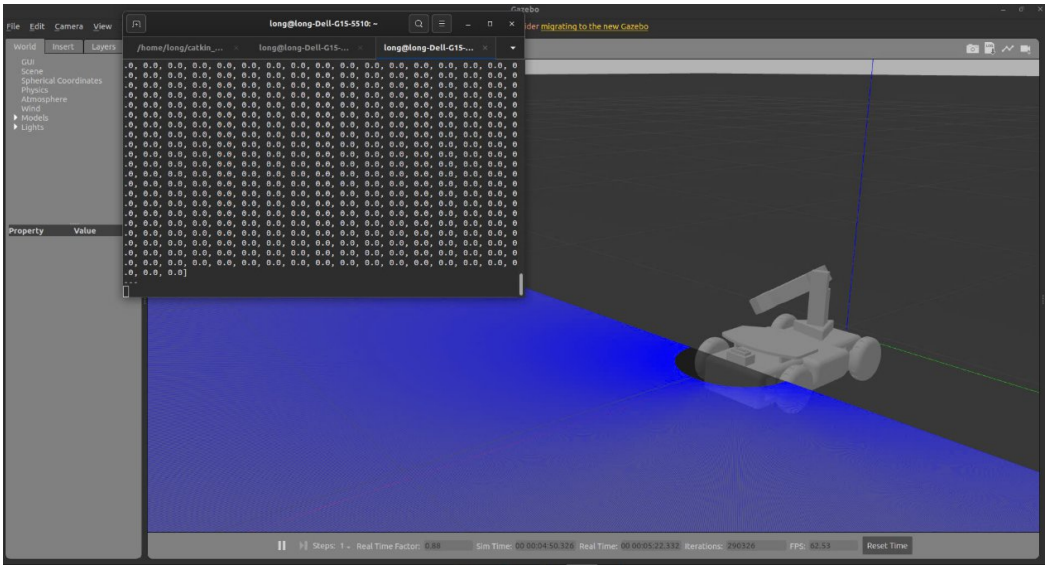
Cấu hình:

- Vị trí gốc lidar_link (pose=0 0 0 0 0 0), hiển thị tia (visualize=true), cập nhật 180 Hz.
- Quét ngang 174° (-1.52 đến 1.52 rad), 720 mẫu, phạm vi 0.1-5 m, nhiễu Gaussian 1 cm.

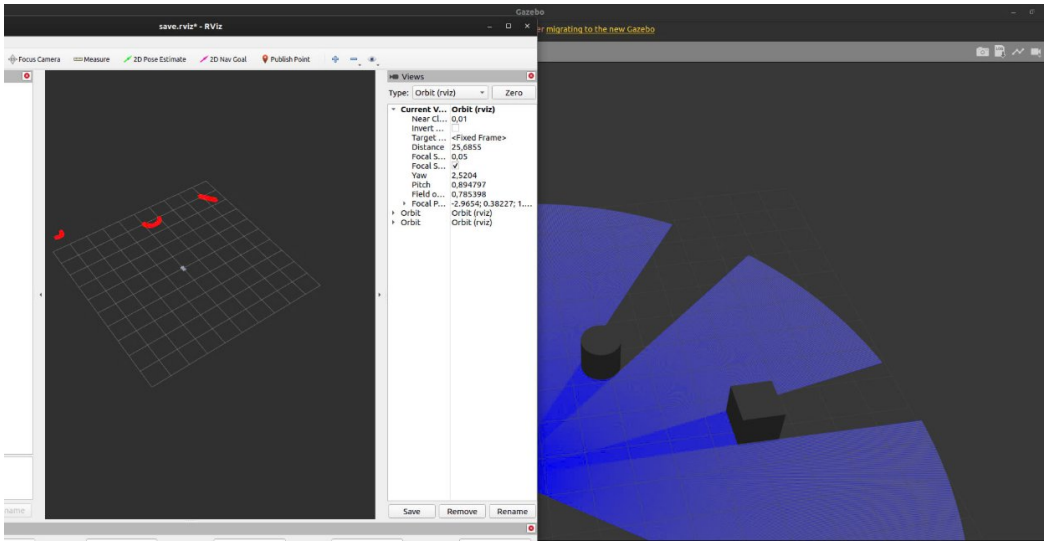
Để xuất dữ liệu từ cảm biến, ta xuất dữ liệu từ topic: /scan:

- rostopic echo /scan

Kết quả ta sẽ thu được dữ liệu như hình sau:



Trực quan hoá qua Rviz:



➢ Encoder:

```
<gazebo>
    <plugin name="diff_drive_controller" filename="libgazebo_ros_diff_drive.so">
      <leftJoint>joint_lb</leftJoint>
      <rightJoint>joint_rb</rightJoint>
      <wheelSeparation>0.4</wheelSeparation>
      <wheelDiameter>0.215</wheelDiameter>
      <robotBaseFrame>base_link</robotBaseFrame>
      <odomTopic>/odom</odomTopic>
      <publishWheelTF>false</publishWheelTF>
      <publishOdomTF>true</publishOdomTF>
    </plugin>
</gazebo>
<gazebo>
  <plugin name="joint_state_publisher" filename="libgazebo_ros_joint_state_publisher.
    <robotNamespace>/</robotNamespace>
    <updateRate>100.0</updateRate>
    <jointName>joint_rf, joint_lf, joint_rb, joint_lb, joint_armone, joint_armtwo, jo
    <topicName>/joint_states</topicName>
  </plugin>
</gazebo>
```

Plugin diff_drive_controller trong Gazebo điều khiển vi sai và xuất dữ liệu odometry cho robot "ten".

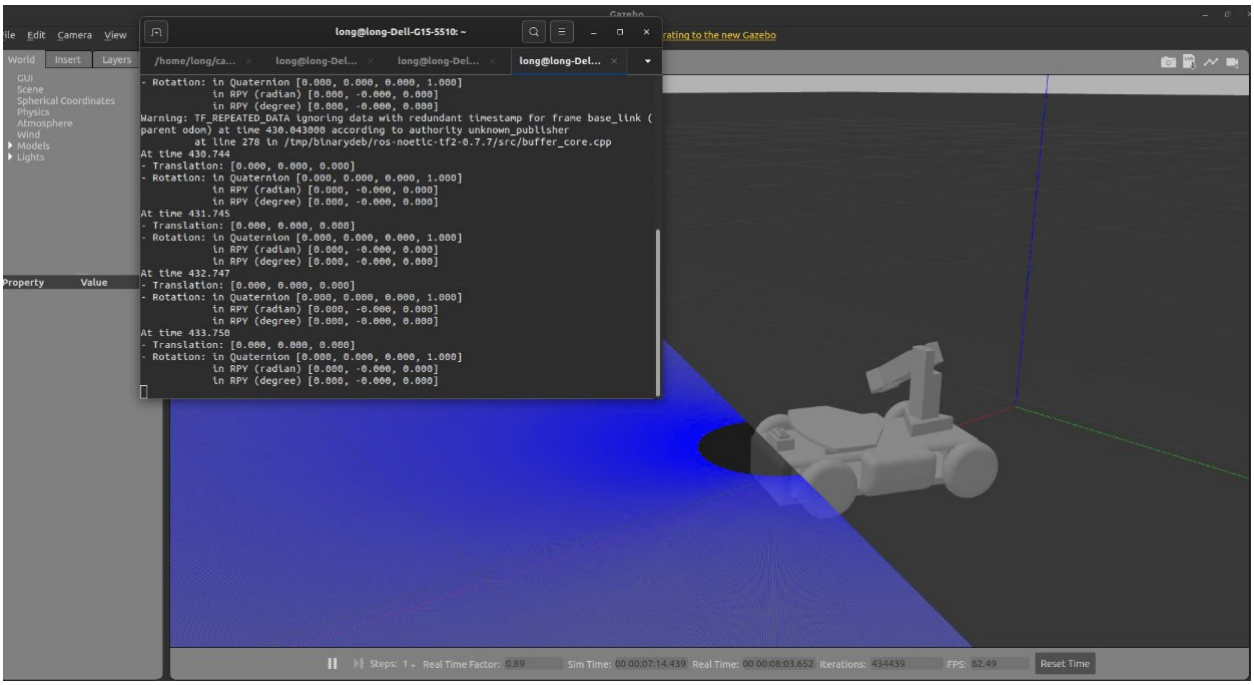
Cấu hình:

- Sử dụng libgazebo_ros_diff_drive.so, điều khiển bánh trái joint_lb và phải joint_rb.
- Khoảng cách bánh 0.4 m (wheelSeparation), đường kính 0.215 m (wheelDiameter).

Để xuất dữ liệu, ta lấy dữ liệu từ topic /odom.

- rostopic echo /odom.

Dữ liệu thu được sẽ có dạng sau:



3.2.4. Các plugin điều khiển.

- Plugin điều khiển tay máy:

```
<gazebo>
  <plugin name="arm_controller" filename="libgazebo_ros_joint_pose_trajectory.so">
    <robotNamespace>/</robotNamespace>
    <updateRate>100.0</updateRate>
    <topic禁止禁止

    <topicName>arm_controller/command</topicName>
    <jointName>joint_armone, joint_armtwo, joint_grip</jointName>
  </plugin>
</gazebo>
```

Plugin arm_controller trong Gazebo điều khiển tay robot "ten".

Cấu hình:

- Sử dụng libgazebo_ros_joint_pose_trajectory.so, không gian tên / (robotNamespace).
- Cập nhật 100 Hz (updateRate=100.0).

- Điều khiển các khớp: joint_armone, joint_armtwo, joint_grip.

Để điều khiển tay máy, ta gửi dữ liệu qua topic: /arm_controller/command.

➢ Plugin điều khiển xe:

```
<gazebo>
  <plugin name="skid_steer_drive_controller" filename="libgazebo_ros_skid_steer_drive.
    <updateRate>100.0</updateRate>
    <robotNamespace>/</robotNamespace>
    <leftFrontJoint>joint_lf</leftFrontJoint>
    <rightFrontJoint>joint_rf</rightFrontJoint>
    <leftRearJoint>joint_lb</leftRearJoint>
    <rightRearJoint>joint_rb</rightRearJoint>
    <wheelSeparation>0.4</wheelSeparation>
    <wheelDiameter>0.215</wheelDiameter>
    <robotBaseFrame>base_link</robotBaseFrame>
    <torque>200</torque>
    <topicName>cmd_vel</topicName>
    <gaussianNoise>0.0</gaussianNoise>
    <broadcastTF>false</broadcastTF>
  </plugin>
</gazebo>
```

Plugin skid_steer_drive_controller trong Gazebo điều khiển robot "ten" kiểu skid-steer.

Cấu hình:

Sử dụng libgazebo_ros_skid_steer_drive.so, cập nhật 100 Hz (updateRate=100.0), không gian tên / (robotNamespace).

Điều khiển bánh: trước trái joint_lf, trước phải joint_rf, sau trái joint_lb, sau phải joint_rb.

Khoảng cách bánh 0.4 m (wheelSeparation), đường kính 0.215 m (wheelDiameter), lực xoắn 200 Nm (torque).

Chức năng: Nhận lệnh vận tốc qua topic /cmd_vel, không nhiễu (gaussianNoise=0).

4. Cơ chế điều khiển.(Control)

4.1. Tổng quan ý tưởng:

Tạo ra một tệp script có khả năng điều khiển tay máy và robot di chuyển theo đúng mô hình AckermanSteering, để thực hiện yêu cầu đó, ta cần làm được những việc sau:

- Đọc được dữ liệu được nhấn từ phím trong thời gian thực, không cần nhấn Enter
- Điều khiển robot di chuyển qua topic /cmd_vel, đồng thời điều khiển 2 khớp ackerman quay khi nhận được lệnh rẽ trái phải qua topic /armcontroller /command.
- Điều khiển cánh tay robot di chuyển khi nhận lệnh qua topic /armcontroller/command.

4.2. Thực tế và mã nguồn:

Ta điều khiển mô hình di chuyển với phím WASD, tang giảm tốc độ với phím E/R, điều khiển tay máy với phím số 2/4/6/7/8/9.

Mã nguồn thực tế:

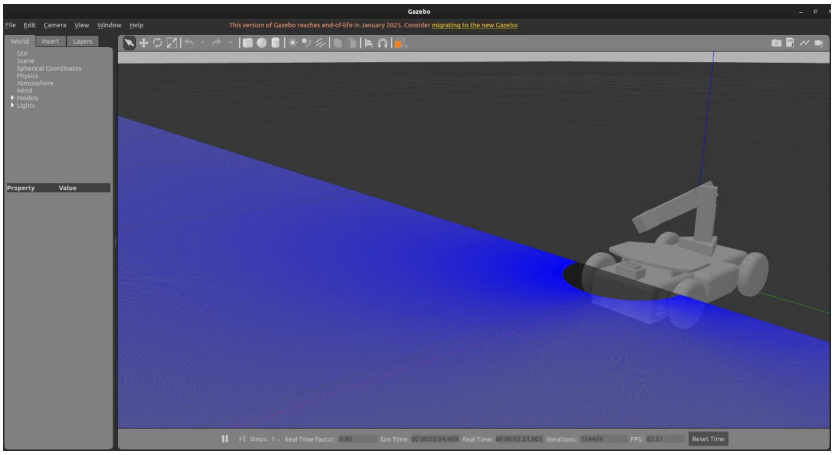
```
import rospy
from trajectory_msgs.msg import JointTrajectory, JointTrajectoryPoint
from geometry_msgs.msg import Twist
from std_msgs.msg import Header
import sys
import tty
import termios

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

def send_home_pose(pub_arm):
    traj = JointTrajectory()
    traj.header = Header()
    traj.header.frame_id = "base_link"
    traj.header.stamp = rospy.Time.now()
    traj.joint_names = ['joint_armone', 'joint_armtwo', 'joint_grip']

    point = JointTrajectoryPoint()
    point.positions = [0.13, 0.6, 0.0]  # Home pose
    point.velocities = [0.1, 0.1, 0.1]
    point.time_from_start = rospy.Duration(2.0)
    traj.points.append(point)

    rospy.loginfo("Moving arm to home pose (0.13, 0.6, 0.0)...")
    pub_arm.publish(traj)
    rospy.sleep(2)
```

```
    rospy.loginfo("Moving arm to home pose (0.13, 0.6, 0.0)...")
    pub_arm.publish(traj)
    rospy.sleep(2)

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

    pub_arm = rospy.Publisher('/arm_controller/command', JointTrajectory, queue
    pub_vel = rospy.Publisher('/cmd_vel', Twist, queue_size=10)

    rospy.loginfo("Waiting for Gazebo to be ready...")
    rospy.wait_for_service('/gazebo/spawn_urdf_model')
    rospy.sleep(2)

    send_home_pose(pub_arm)

    armone_pos = 0.13
    armtwo_pos = 0.6
    grip_pos = 0.0
    step = 0.1

    arm_min_limit = -1.57
    arm_max_limit = 1.57
    grip_min_limit = -0.2
    grip_max_limit = 0.2

    linear_speed = 1.0
    angular_speed = 1.0

    print("Điều khiển robot:")
    print("  W: Tiến lên | S: Lùi lại | A: Quay trái | D: Quay phải")
    print("  E: Tăng tốc độ (+1 m/s) | R: Giảm tốc độ (-1 m/s)")
    print("Điều khiển tay máy:")
    print("  4/6: joint_armone trái/phải | 8/2: joint_armtwo lên/xuống")
    print("  7/9: Đóng/mở joint_grip | Q: Thoát")


rate = rospy.Rate(10)
while not rospy.is_shutdown():
    key = get_key().lower()

    twist = Twist()

    if key == 'w':
        twist.linear.x = linear_speed
        twist.angular.z = 0.0
        rospy.loginfo(f"Moving forward at {linear_speed} m/s")
    elif key == 's':
        twist.linear.x = -linear_speed
        twist.angular.z = 0.0
        rospy.loginfo(f"Moving backward at {linear_speed} m/s")
    elif key == 'a':
        twist.linear.x = 0.0
        twist.angular.z = angular_speed
        rospy.loginfo("Turning left")
    elif key == 'd':
        twist.linear.x = 0.0
        twist.angular.z = -angular_speed
        rospy.loginfo("Turning right")
    elif key == 'e':
        linear_speed += 1.0
        rospy.loginfo(f"Speed increased to {linear_speed} m/s")
    elif key == 'r':
        linear_speed = max(0.0, linear_speed - 1.0)
        rospy.loginfo(f"Speed decreased to {linear_speed} m/s")
    else:
        twist.linear.x = 0.0
        twist.angular.z = 0.0
```

```
if key == '4':
    armone_pos = max(arm_min_limit, armone_pos - step)
    arm_moved = True
elif key == '6':
    armone_pos = min(arm_max_limit, armone_pos + step)
    arm_moved = True
elif key == '8':
    armtwo_pos = min(arm_max_limit, armtwo_pos + step)
    arm_moved = True
elif key == '2':
    armtwo_pos = max(arm_min_limit, armtwo_pos - step)
    arm_moved = True
elif key == '7':
    grip_pos = max(grip_min_limit, grip_pos - step)
    arm_moved = True
elif key == '9':
    grip_pos = min(grip_max_limit, grip_pos + step)
    arm_moved = True
elif key == 'q':
    rospy.loginfo("Exiting...")
    break

if arm_moved:
    traj = JointTrajectory()
    traj.header = Header()
    traj.header.frame_id = "base_link"
    traj.header.stamp = rospy.Time.now()
    traj.joint_names = ['joint_armone', 'joint_armtwo', 'joint_grip']

    point = JointTrajectoryPoint()
    point.positions = [armone_pos, armtwo_pos, grip_pos]
    point.velocities = [0.1, 0.1, 0.1]
    point.time_from_start = rospy.Duration(0.1)
    traj.points.append(point)


            rospy.loginfo(f"Sending arm command: joint_armone={armone_pos:.2f}, joint_
            pub_arm.publish(traj)

        pub_vel.publish(twist)

        rate.sleep()

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

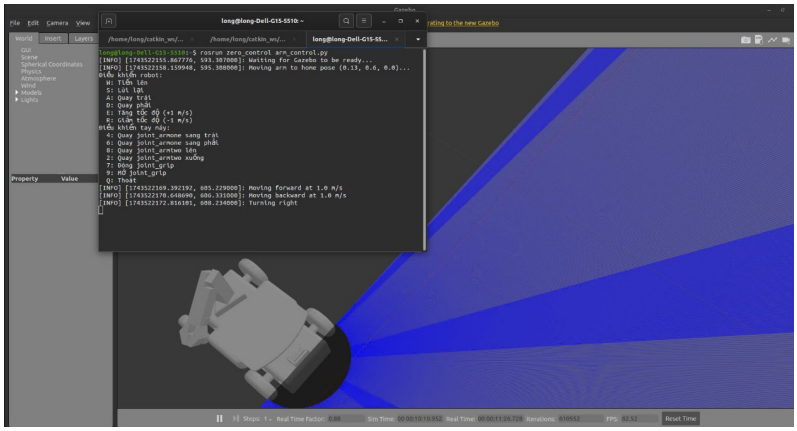
4.3. Kết quả thực tế:

Mô hình trong Gazebo:

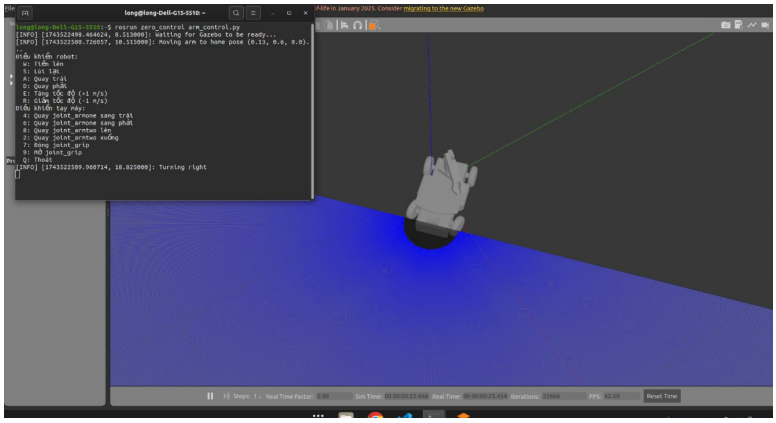


Khi khởi chạy chương trình điều khiển:

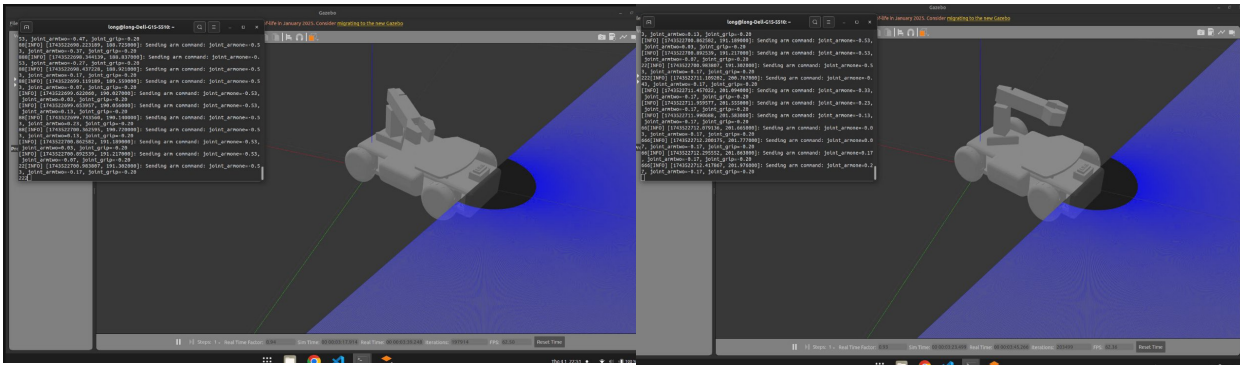
Đi thẳng:

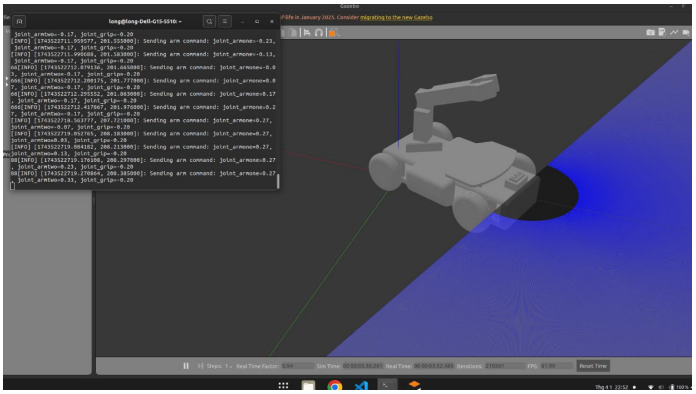


Rẽ phải:



Điều khiển tay máy:





5. *Tổng kết (Sumary):*

Qua quá trình thực hiện bài tập lớn, em tự nhận thấy đã hoàn thành tương đối đầy đủ yêu cầu đề bài đặt ra,mô hình vận hành tương đối ổn định, các chức năng chính hoạt động đúng như mong đợi, tuy nhiên vẫn còn một số lỗi nhỏ của tay máy vẫn chưa được khắc phục triệt để. Em sẽ cố gắng khắc phục và phát triển thêm tính năng vào project cuối kỳ.