ĐẠI HỌC QUỐC GIA HÀ NỘI
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ

BÁO CÁO GIỮA KỲ LẬP TRÌNH ROS

Họ và tên: Vương Ngọc Đạt

Mã sinh viên: 2202754

1. Giới thiệu

Dự án thiết kế một robot di động sử dụng hệ dẫn động vi sai với hai bánh, hoạt động trong môi trường mô phỏng Gazebo. Robot được trang bị các cảm biến như Camera, Lidar và GPS để thu thập dữ liệu môi trường và hỗ trợ điều hướng. Ngoài ra, hệ thống điều khiển cho phép robot di chuyển từ xa thông qua bàn phím hoặc giao diện ROS, phục vụ cho các nhiệm vụ mô phỏng liên quan đến di chuyển và tránh vật cản.

1.1 Mô tả dạng robot

Robot di động sử dụng 2 bánh dẫn động kết hợp 1 bánh đa hướng giúp robot di chuyển linh hoạt. Cấu trúc gồm:

- Thân robot: là giá đỡ cho tay máy, nơi gắn bánh xe và nơi đặt các cảm biến.

- Bánh xe: 2 bánh dẫn động kết hợp với bánh đa hướng được đặt ở trước.

- Cảm biến: sử dụng cảm biến Camera, Lidar và GPS.

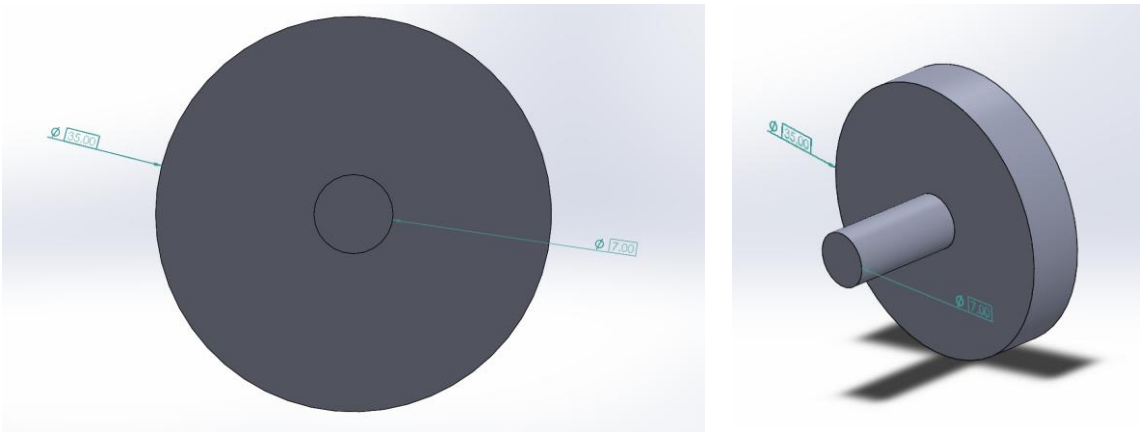
- Tay máy: tay máy gồm 2 joints và 3 links, 2 joints đều là khớp xoay.

2. Thiết kế và đặt trục cho robot

2.1 Thiết kế

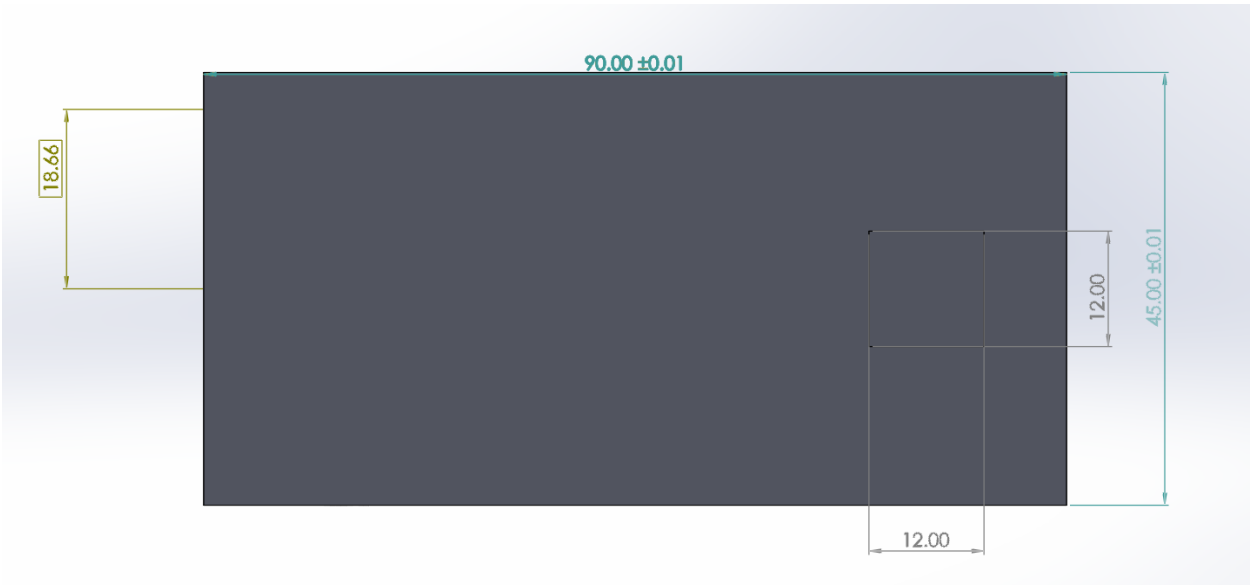
Thiết kế được thực hiện và vẽ trên SOLIDWORKS, các kích thước đều được tính theo đơn vị cm.

- Bánh xe và trục của bánh

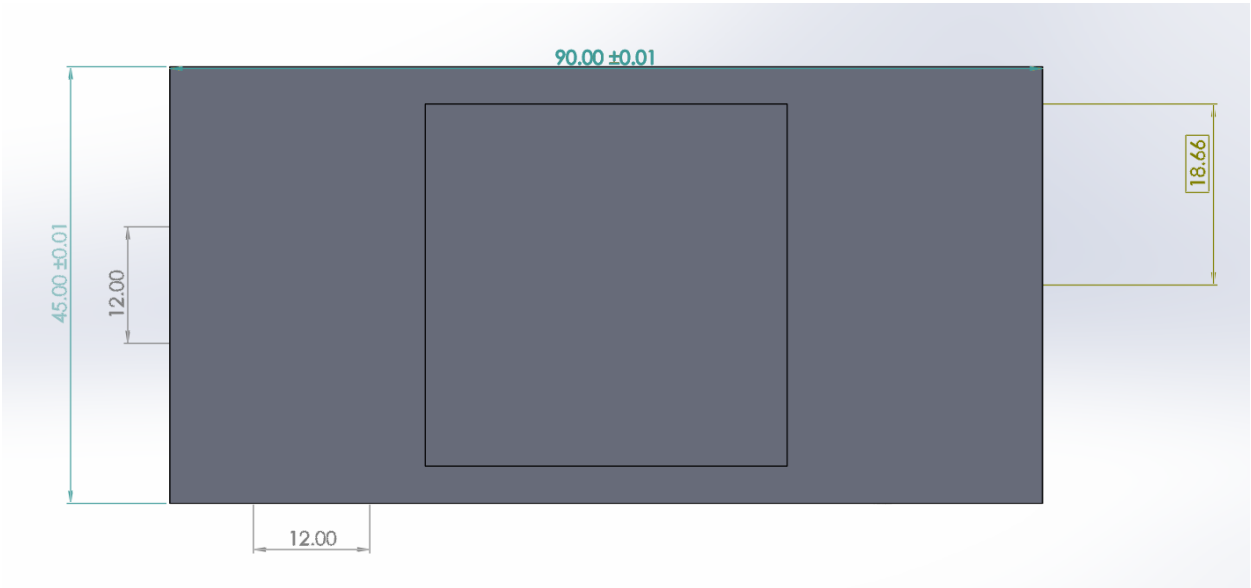


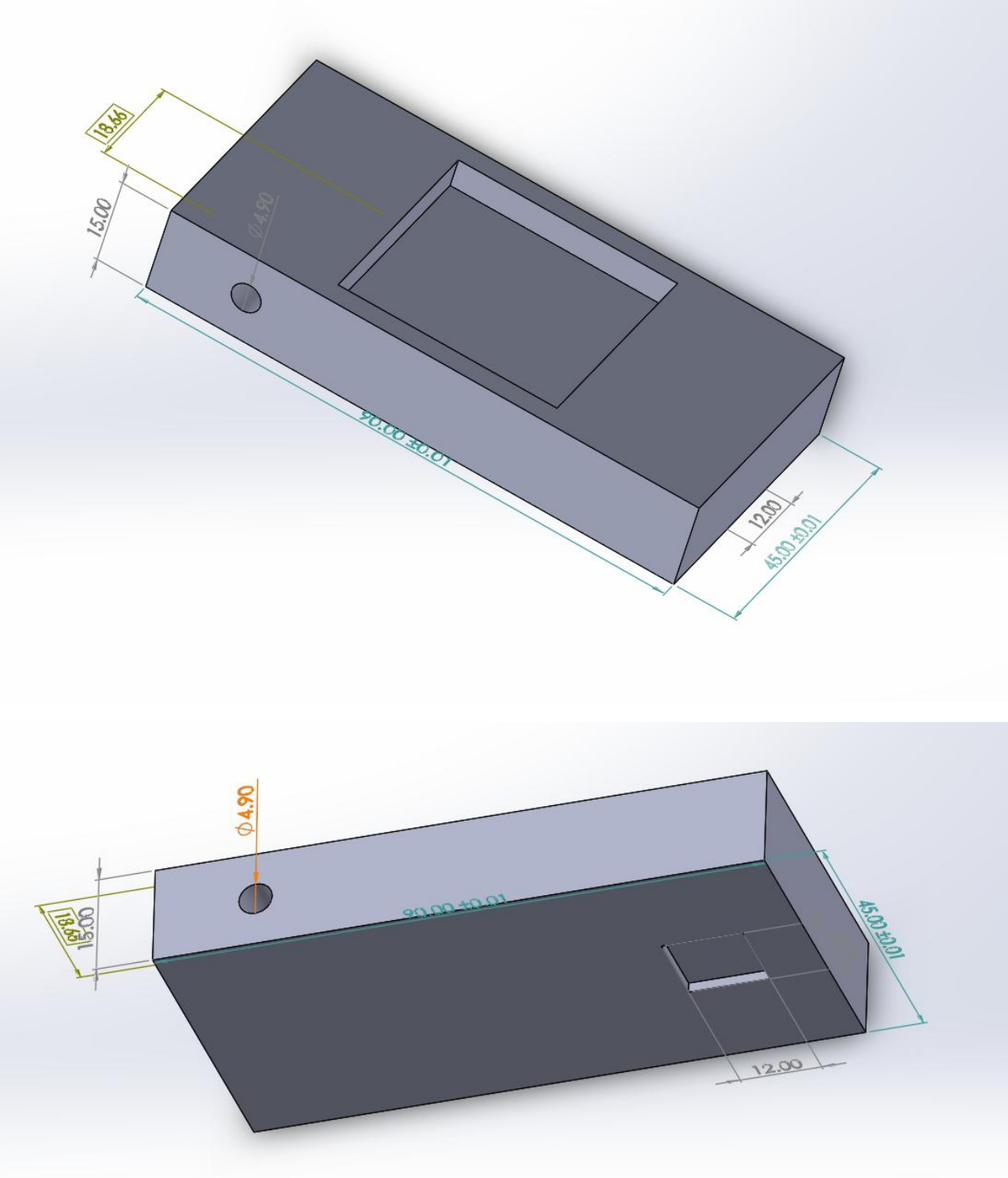
Hình 1: Bánh xe và trục của bánh xe

- Khung xe


90.00 ±0.01
18.66
12.00
45.00 ±0.01
12.00


⌀4.90
15.00

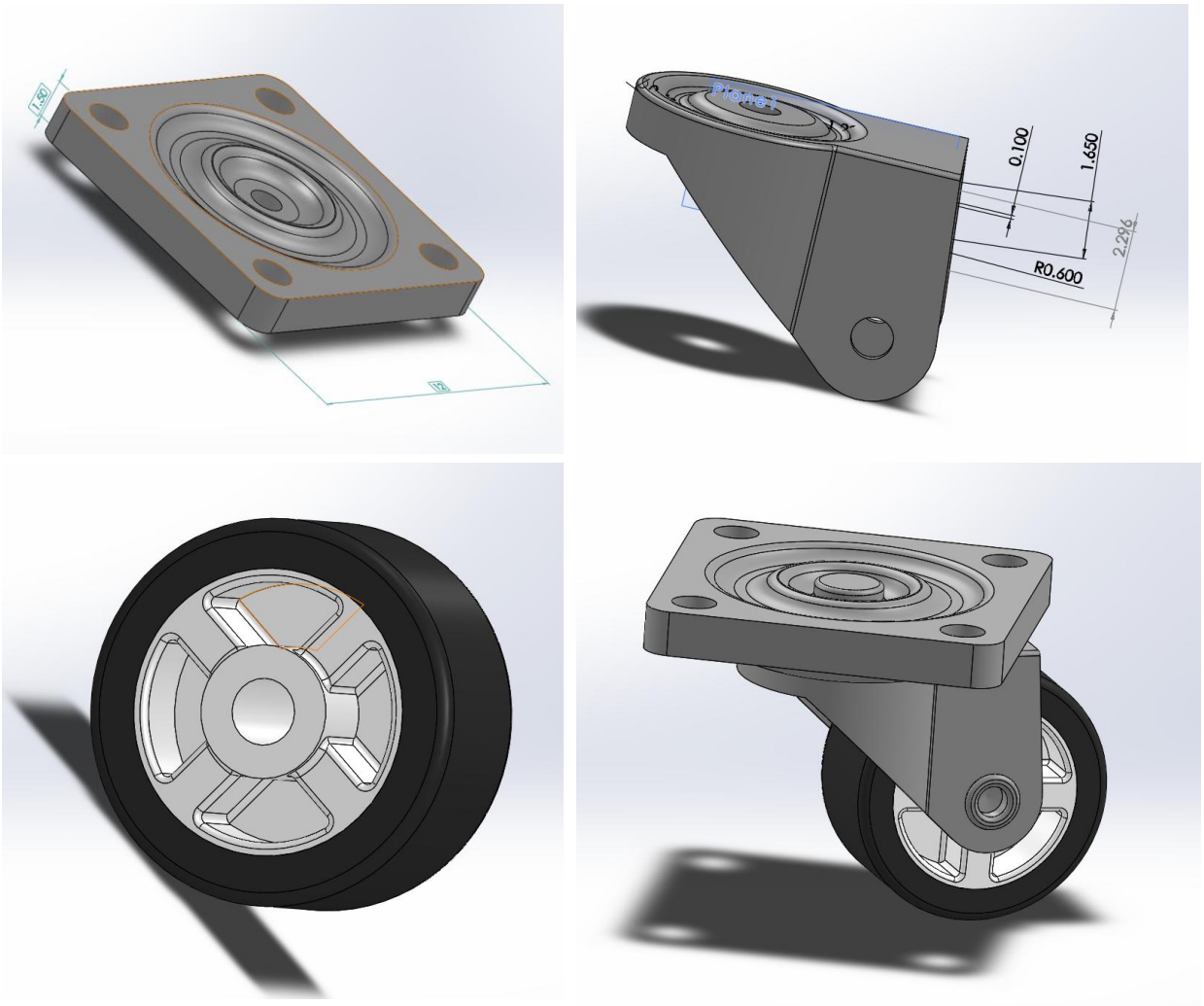

90.00 ±0.01
18.66
45.00 ±0.01
12.00
12.00



Hình 2: Khung xe

**Chú thích: khung xe bao gồm 1 phần trụ rỗng để đặt 2 bánh xe, 1 phần lõm hình vuông ở trên chính giữa để đặt tay máy và 1 phần lõm phía trước ở dưới để đặt bánh đa hướng.

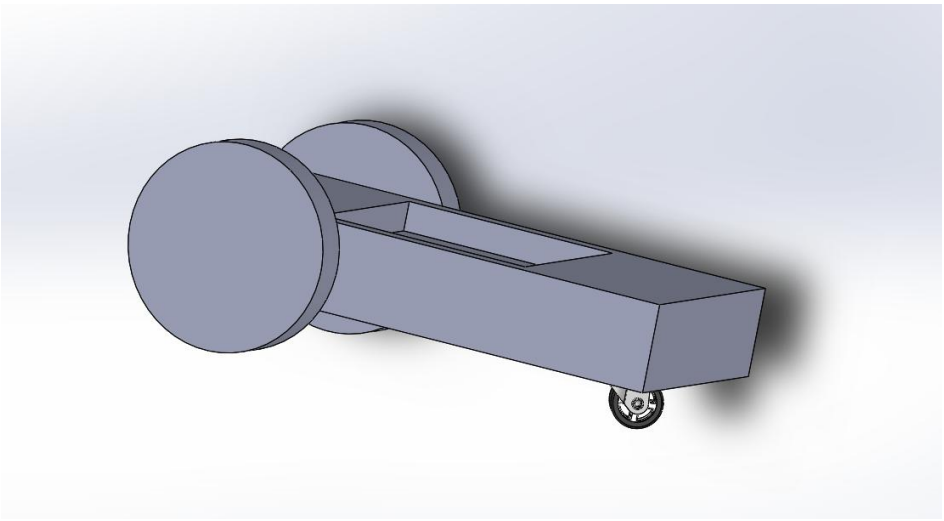
- Bánh đa hướng



Hình 3: Bánh đa hướng

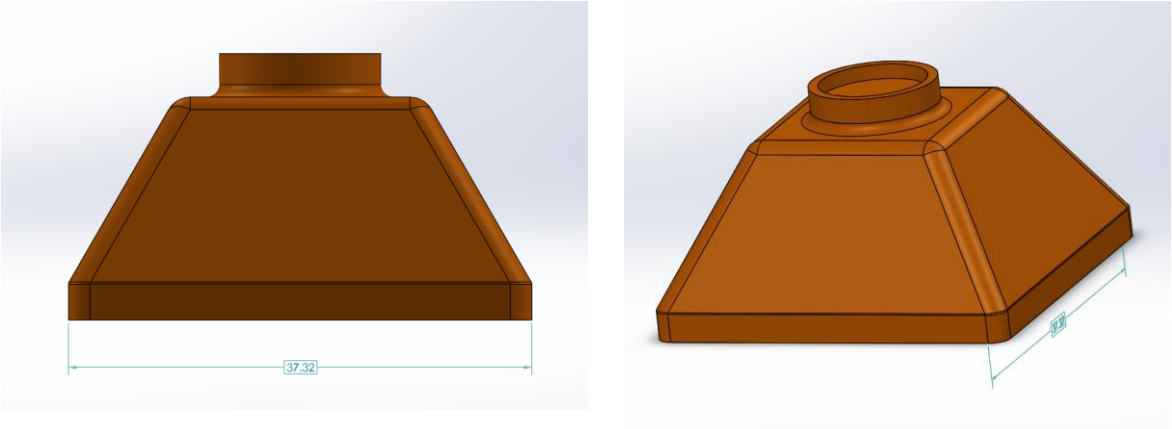
**Chú thích: chọn bánh đa hướng thay vì mắt trâu vì mắt trâu sẽ có ma sát với mặt phẳng chuyển động khi mô phỏng trong gazebo từ đó khiến robot bị khựng khi với chuyển động nhỏ.

- Mô hình xe

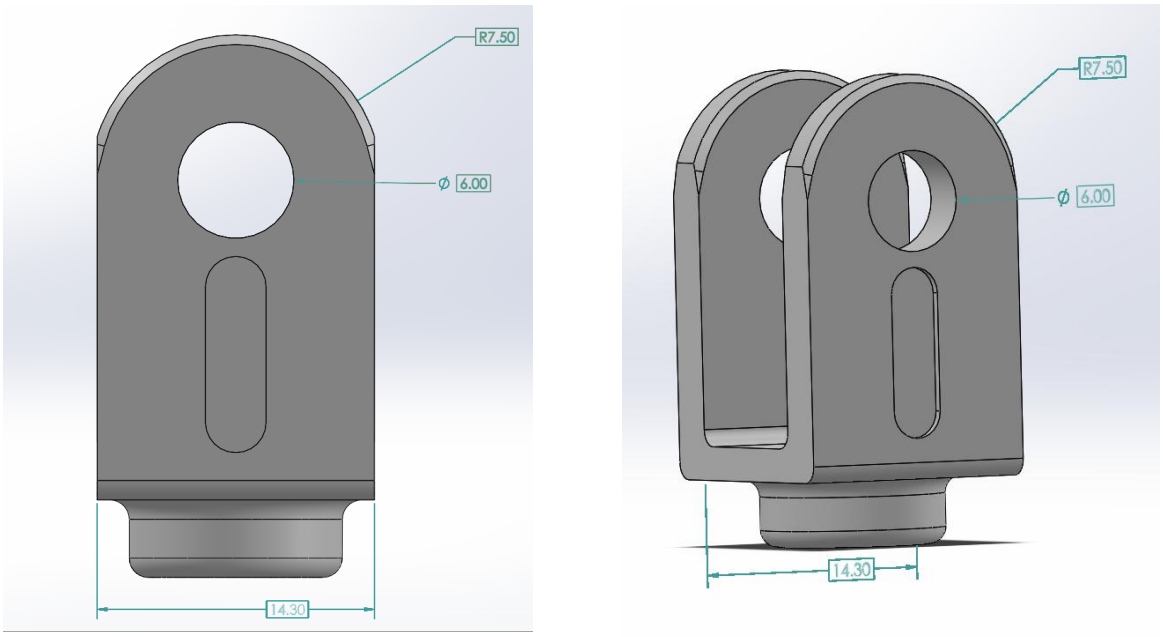


Hình 4: Mô hình xe khi ghép

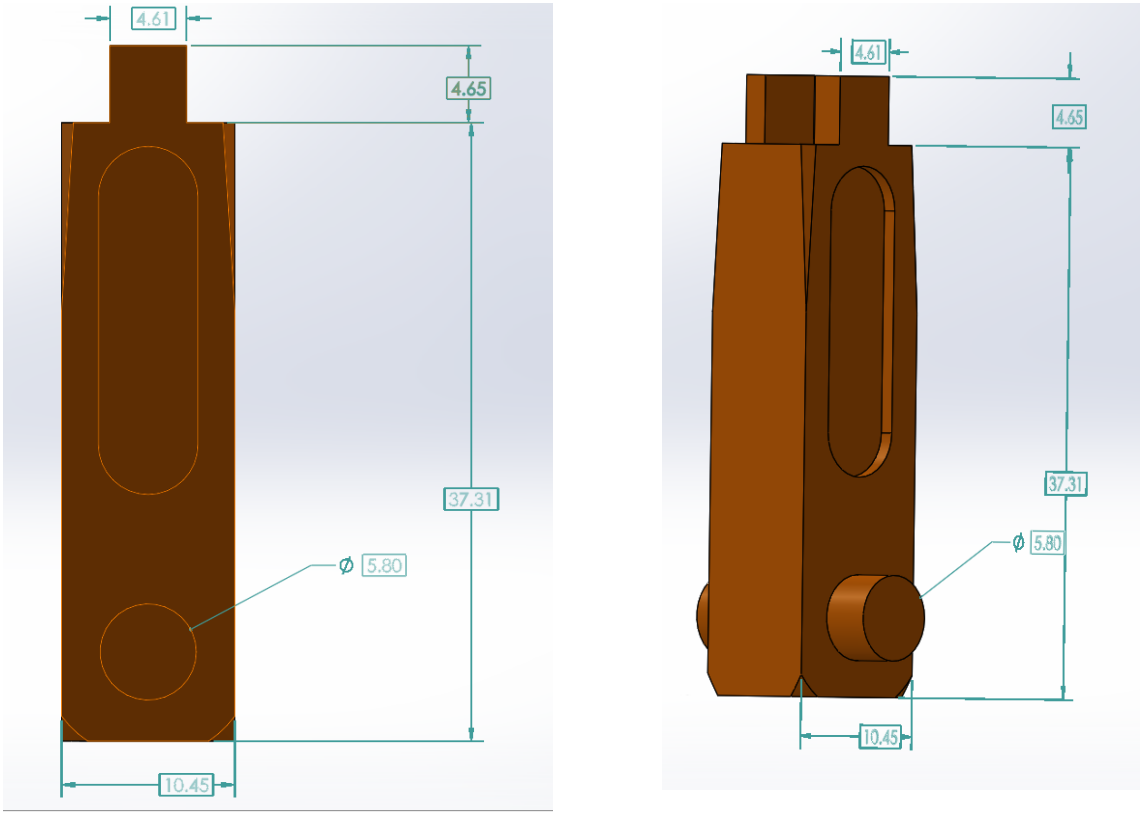
- Tay máy



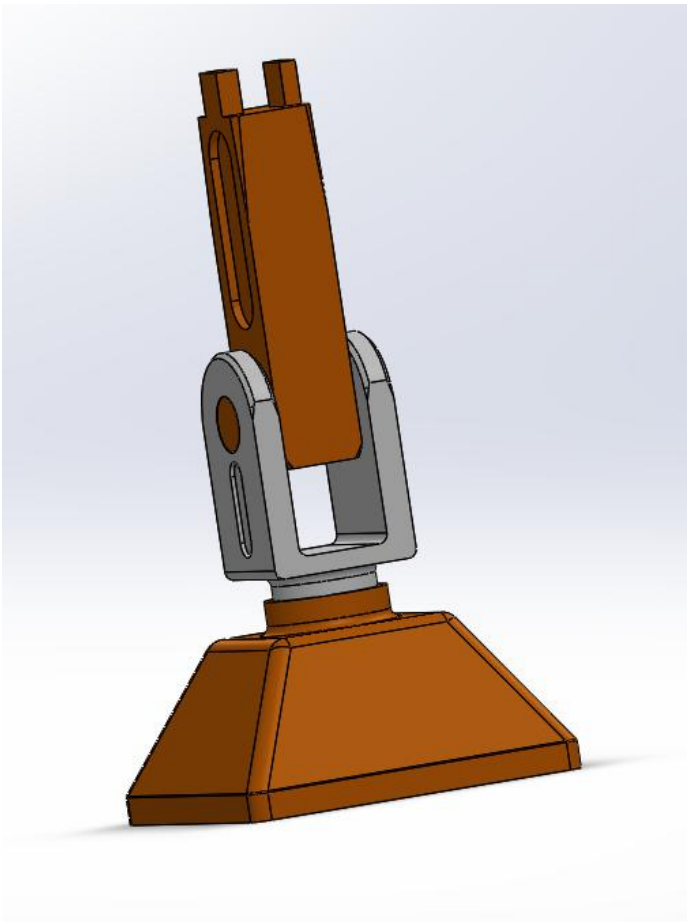
Hình 5: base_link của tay máy (phần sẽ được gắn với khung xe)



Hình 6: link_1 của tay máy

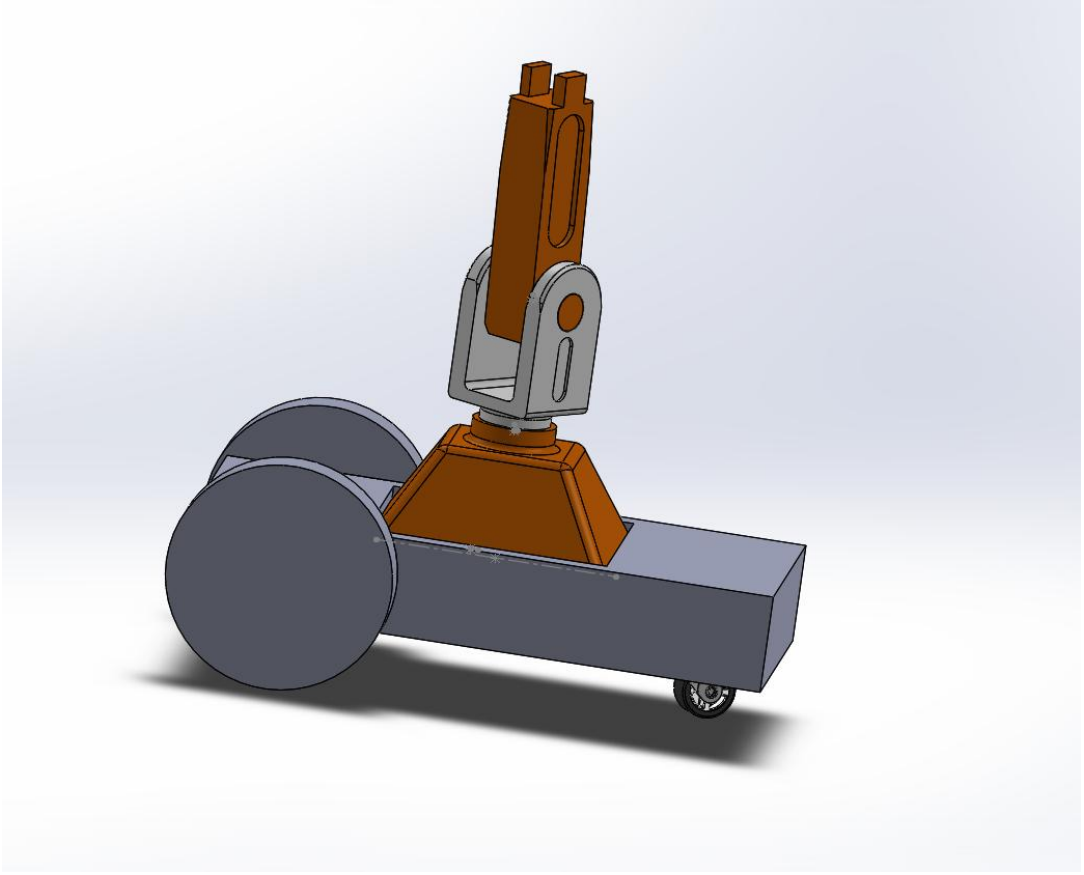


Hình 7: link_2 của tay máy



Hình 8: Mô hình tay máy sau khi ghép

- Mô hình toàn bộ xe



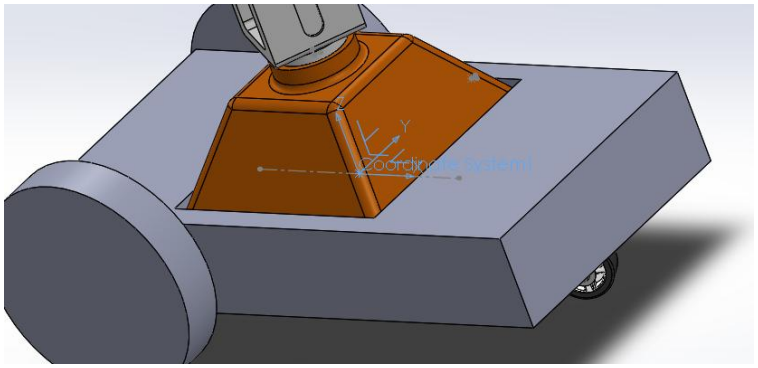
Hình 10: Mô hình toàn bộ xe sau khi ghép

2.2 Đặt trục và hệ

Trục base_link sẽ đặt cho khung xe và base_link của tay máy để dễ dàng đồng bộ khi điều khiển.

2.2.1 Đặt trục và hệ cho khung xe

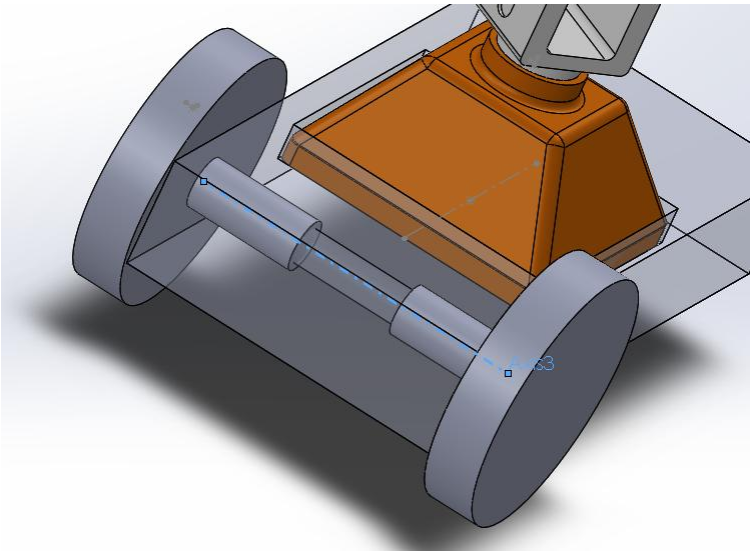
- Đặt trục cho khung xe



Hình 11: Hệ trục cho khung xe

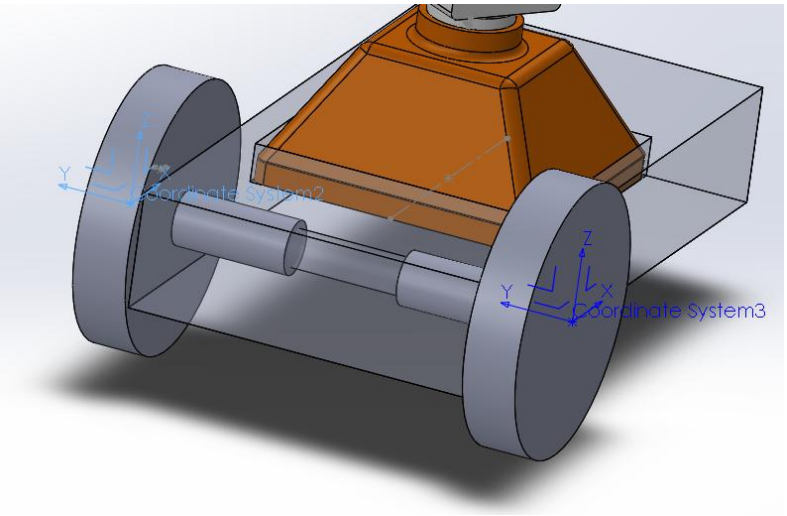
- Đặt trục cho bánh xe

+) Đặt trục xoay cho bánh xe: 2 bánh xe sẽ lấy trục này làm trục xoay



Hình 12: Trục xoay của bánh xe

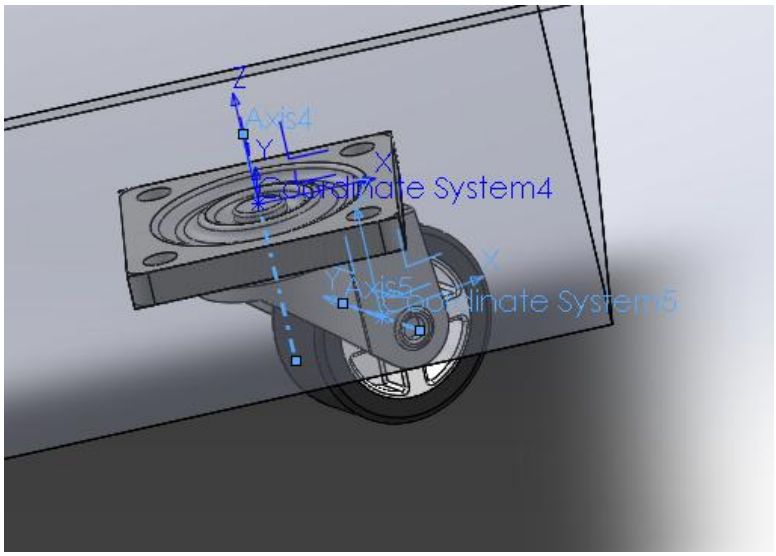
+) Đặt hệ cho 2 bánh xe



Hình 13: Hệ trục cho hai bánh xe

**Chú thích: hệ bánh xe có trục X luôn hướng về trước để khi di chuyển sẽ trả về x dương khi đi thẳng và x âm khi đi lùi

- Đặt trục cho bánh xe đa hướng

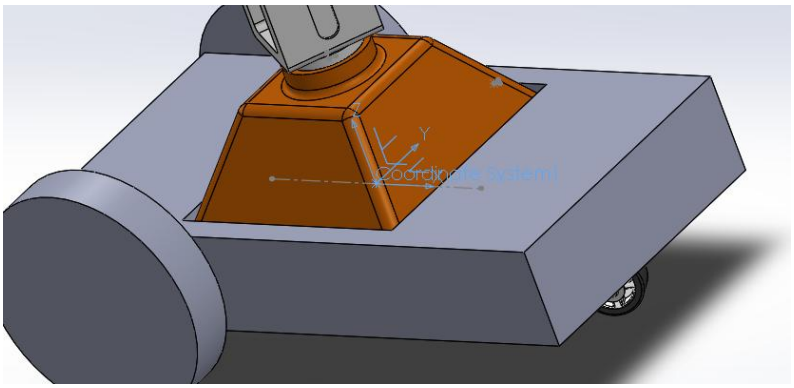


Hình 14: Đặt trục cho bánh xe đa hướng

**Chú thích: trục Axis4 để bánh xe tự quay quanh trục dọc của nó giúp cho việc thay đổi hướng của bánh xe, Axis5 bánh xe quay quanh trục ngang của nó giúp bánh xe di chuyển tiến lùi. Đặt 2 hệ cho bánh xe có thể giúp ta kiểm soát nếu cần.

- Đặt trục cho tay máy

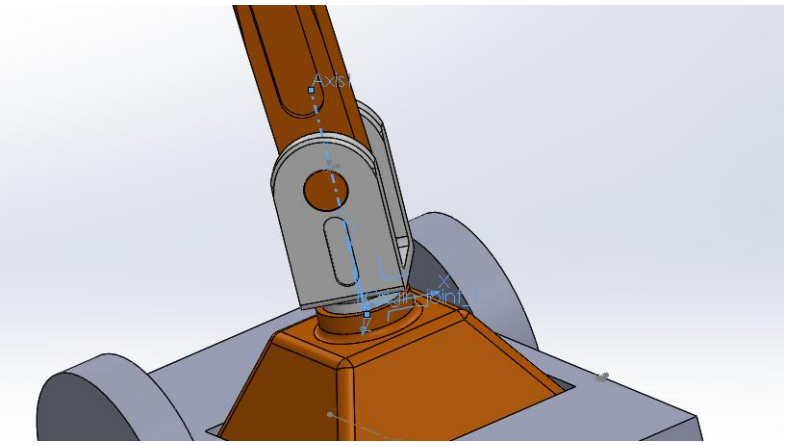
+) Đặt trục và hệ cho base_link



Hình 15: Đặt hệ cho tay máy

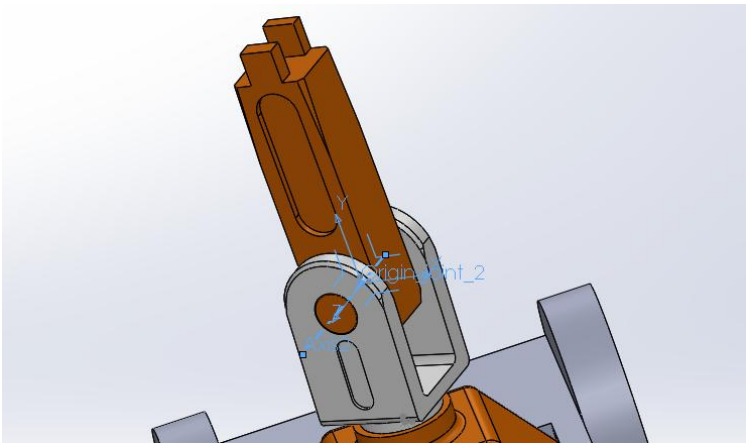
**Đặt hệ tay máy ở base_link trùng với hệ của xe giúp dễ dàng kiểm soát vị trí tương đối giữa xe và tay máy.

+) Đặt trục và hệ cho link_1



Hình 15: Trục và hệ cho link_1

- Đặt trục và hệ cho link_2

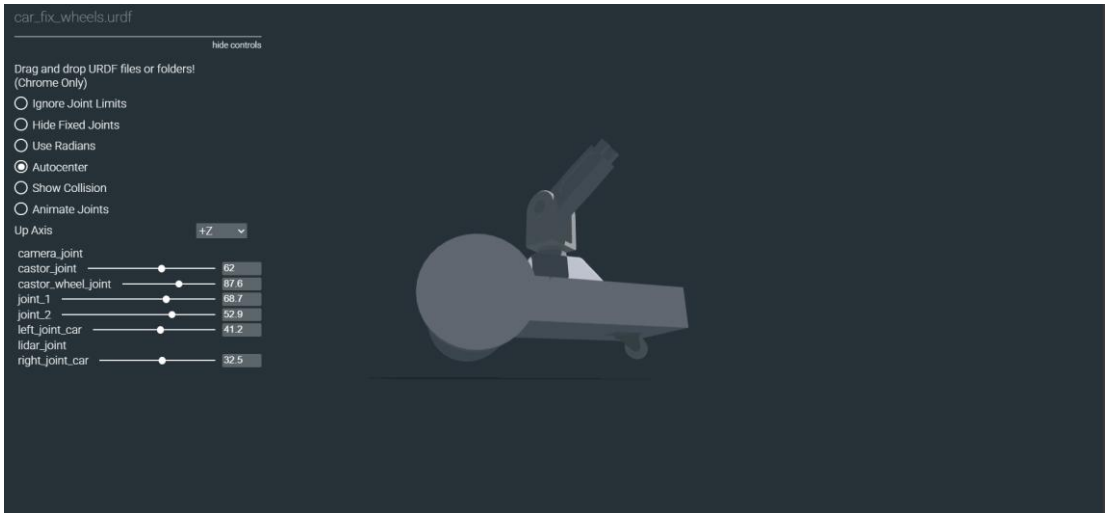


Hình 16: Trục và hệ cho link_2

3. Xuất file URDF và mô phỏng trong Gazebo

- Xuất file URDF: đối với joint_1 và joint_2 đặt revolute và 2 bánh xe đặt là continuous

- Sau khi xuất file xong ta có thể kiểm tra các khớp hoạt động bằng web sau: Links



Hình 17: Kết quả kiểm tra sau khi xuất file URDF

- Cấu trúc file URDF

config	first_commit	2 days ago
launch	fix display.launch	10 hours ago
meshes	first_commit	2 days ago
rviz	fix lidar and controllers	2 days ago
scripts	fix display.launch	10 hours ago
urdf	fix lidar and controllers	2 days ago
.gitattributes	Initial commit	2 days ago
CMakeLists.txt	first_commit	2 days ago
README.md	Update README.md	4 hours ago
export.log	first_commit	2 days ago
package.xml	first_commit	2 days ago

Hình 18: Cấu trúc file URDF sau khí xuất file URDF

+) config: Lưu trữ các tập tin cấu hình, bao gồm các tham số cho bộ điều khiển (ví dụ: PID), bộ lọc dữ liệu (như Kalman) hoặc các cài đặt khác cần thiết cho hệ thống.

+) launch: Chứa các tập tin .launch, dùng để khởi động các mô phỏng trên Gazebo, thiết lập môi trường RViz, hoặc chạy các node điều khiển robot.

+) meshes: Lưu các mô hình 3D của robot dưới định dạng .stl, dùng để hiển thị hình dạng của robot trong mô phỏng.

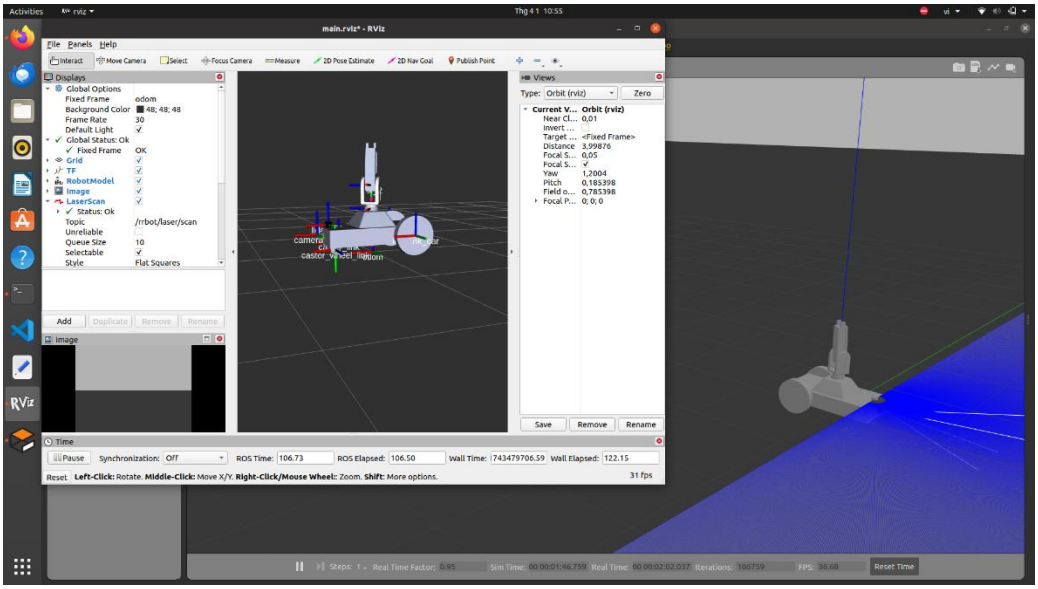
+) rviz: Chứa các tập tin cấu hình hiển thị dành cho RViz, giúp tùy chỉnh giao diện trực quan khi quan sát robot.

+) scripts: Thư mục này chứa các tập tin script viết bằng Python, hỗ trợ việc kiểm tra, điều khiển hoặc lập trình thêm cho robot.

+) urdf: Nơi lưu trữ các tệp mô tả robot sử dụng định dạng URDF hoặc Xacro, giúp định nghĩa cấu trúc và động học của robot.

+) CmakeLists.txt và package.xml: biên dịch và thiết lập package trong ROS

- Kết quả mô phỏng trong Gazebo và Rviz

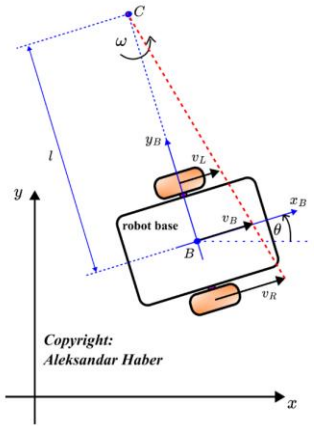


Hình 19: Hình ảnh mô hình xe và các trục đã đặt trong SOLID

4. Mô hình động học, điều khiển trong Gazebo và cảm biến

4.1 Mô hình động học và điều khiển xe

- Mô hình động học:



Hình 20: Mô hình robot 2 bánh trên hệ Oxy (nguồn: Haber)

+) Động học thuận:

Cho một robot hai bánh với:

- R là bán kính bánh xe
- L là khoảng cách giữa hai bánh
- ω_L và ω_R là tốc độ góc của bánh trái và bánh phải

Vận tốc tuyến tính của từng bánh:

$$v_L = R \cdot \omega_L$$

$$v_R = R \cdot \omega_R$$

Vận tốc của tâm robot:

$$v = \frac{v_R + v_L}{2} = \frac{R(\omega_R + \omega_L)}{2}$$

Vận tốc góc của robot:

$$\omega = \frac{v_R - v_L}{L} = \frac{R(\omega_R - \omega_L)}{L}$$

Phương trình động học:

$$\dot{x} = v \cos\theta$$

$$\dot{y} = v \sin\theta$$

$$\dot{\theta} = \omega$$

Hình 21: Phương trình tính toán động học thuận

+) Động học ngược:

Cho trước vận tốc mong muốn của robot v và ω, ta cần tính tốc độ bánh xe:

$$\omega_L = \frac{2v - \omega L}{2R}$$

$$\omega_R = \frac{2v + \omega L}{2R}$$

Hình 22: Phương trình động học ngược

**Áp dụng động học ngược vào điều khiển xe

- Ta có các thông số xe và vận tốc mong muốn, ta sử dụng phương trình động học ngược để tính ngược ra vận tốc cần điều khiển.

```
# Thông số xe
WHEEL_RADIUS = 0.35  # Bán kính bánh xe (m)
WHEEL_BASE = 0.45     # Khoảng cách giữa hai bánh xe (m)
LINEAR_SPEED = 0.5   # Vận tốc tuyến tính (m/s)
ANGULAR_SPEED = 1.0  # Vận tốc góc (rad/s)
```

```python
def compute_wheel_velocities(linear_x, angular_z):
    """Tính toán vận tốc bánh xe trái và phải từ vận tốc xe."""
    v_l = (linear_x - (angular_z * WHEEL_BASE / 2)) / WHEEL_RADIUS
    v_r = (linear_x + (angular_z * WHEEL_BASE / 2)) / WHEEL_RADIUS
    return v_l, v_r
```

Hình 23: Hàm tính toán vận tốc 2 bánh dựa trên công thức

- Ta cần add plugin cho 2 bánh xe để có thể publish vào 2 bánh xe đó

```
<gazebo>
  <plugin name="differential_drive_controller" filename="libgazebo_ros_diff_drive.so">

    <!-- Plugin update rate in Hz -->
    <updateRate>10</updateRate>

    <!-- Name of left joint, defaults to `left_joint` -->
    <leftJoint>left_joint_car</leftJoint>

    <!-- Name of right joint, defaults to `right_joint` -->
    <rightJoint>right_joint_car</rightJoint>

    <!-- The distance from the center of one wheel to the other, in meters, defaults to 0.34 m -->
    <wheelSeparation>0.5380</wheelSeparation>

    <!-- Diameter of the wheels, in meters, defaults to 0.15 m -->
    <wheelDiameter>0.2410</wheelDiameter>

    <!-- Wheel acceleration, in rad/s^2, defaults to 0.0 rad/s^2 -->
    <wheelAcceleration>1.0</wheelAcceleration>

    <!-- Maximum torque which the wheels can produce, in Nm, defaults to 5 Nm -->
    <wheelTorque>20</wheelTorque>

    <!-- Topic to receive geometry_msgs/Twist message commands, defaults to `cmd_vel` -->
    <commandTopic>cmd_vel</commandTopic>

    <!-- Topic to publish nav_msgs/Odometry messages, defaults to `odom` -->
    <odometryTopic>odom</odometryTopic>

    <!-- Odometry frame, defaults to `odom` -->
    <odometryFrame>odom</odometryFrame>

    <!-- Robot frame to calculate odometry from, defaults to `base_footprint` -->
    <robotBaseFrame>base_link</robotBaseFrame>

    <!-- Odometry source, 0 for ENCODER, 1 for WORLD, defaults to WORLD -->
    <odometrySource>1</odometrySource>

    <!-- Set to true to publish transforms for the wheel links, defaults to false -->
    <publishWheelTF>false</publishWheelTF>

    <!-- Set to true to publish transforms for the odometry, defaults to true -->
    <publishOdom>true</publishOdom>

    <!-- Set to true to publish sensor_msgs/JointState on /joint_states for the wheel joints, defaults to false -->
    <publishWheelJointState>true</publishWheelJointState>

    <!-- Set to true to swap right and left wheels, defaults to true -->
    <legacyMode>false</legacyMode>
  </plugin>
</gazebo>
```

Hình 24: Plugin để điều khiển xe

4.2 Mô hình động học và điều khiển tay máy

- Động học thuận

Tọa độ của đầu cuối (End-Effector) tính theo góc quay θ_1 và θ_2:

$$x = L_1 \cos\theta_1 + L_2 \cos(\theta_1 + \theta_2)$$

$$y = L_1 \sin\theta_1 + L_2 \sin(\theta_1 + \theta_2)$$

Ma trận Jacobian (J):

Để tính vận tốc của đầu cuối dựa trên tốc độ góc $\dot{\theta}_1, \dot{\theta}_2$:

$$J = \begin{bmatrix} -(L_1 \sin\theta_1 + L_2 \sin(\theta_1 + \theta_2)) & -L_2 \sin(\theta_1 + \theta_2) \\ L_1 \cos\theta_1 + L_2 \cos(\theta_1 + \theta_2) & L_2 \cos(\theta_1 + \theta_2) \end{bmatrix}$$

Vận tốc đầu cuối:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \end{bmatrix} = J \begin{bmatrix} \dot{\theta}_1 \\ \dot{\theta}_2 \end{bmatrix}$$

Hình 25: Mô hình động học thuận cho tay máy

- Động học ngược

Cho trước tọa độ mong muốn của đầu cuối (x, y), ta tìm θ_1, θ_2.

Bước 1: Tính θ_2

$$\cos\theta_2 = \frac{x^2 + y^2 - L_1^2 - L_2^2}{2L_1L_2}$$

Nếu $|\cos\theta_2| > 1$, thì điểm (x, y) nằm ngoài tầm với.

$$\theta_2 = \pm \cos^{-1}\left(\frac{x^2 + y^2 - L_1^2 - L_2^2}{2L_1L_2}\right)$$

Bước 2: Tính θ_1

$$\theta_1 = \tan^{-1}\left(\frac{y}{x}\right) - \tan^{-1}\left(\frac{L_2 \sin\theta_2}{L_1 + L_2 \cos\theta_2}\right)$$

Hình 26: Mô hình động học ngược tay máy

- Điều khiển tay máy

+) Cấu hình fila yaml để load thông số cho tay máy

```yaml
arm_1_joint_controller:
  type: "position_controllers/JointPositionController"
  joint: "joint_1"
  pid: {p: 10.0, i: 0.05, d: 1.0}

arm_2_joint_controller:
  type: "position_controllers/JointPositionController"
  joint: "joint_2"
  pid: {p: 10.0, i: 0.05, d: 1.0}

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

Hình 27: File yaml để cấu hình thông số cho tay máy

+) Thêm plugin và transmission

```xml
<gazebo>
  <plugin name="gazebo_ros_control" filename="libgazebo_ros_control.so">
    <robotNamespace>/</robotNamespace>
    <robotSimType>gazebo_ros_control/DefaultRobotHWSim</robotSimType>
    <legacyModeNS>false</legacyModeNS>
    <controlPeriod>0.001</controlPeriod>
  </plugin>
</gazebo>
```

```xml
<transmission name="arm_1_joint_transmission">
  <type>transmission_interface/SimpleTransmission</type>
  <joint name="joint_1">
    <hardwareInterface>hardware_interface/PositionJointInterface</hardwareInterface>
  </joint>
  <actuator name="link1">
    <mechanicalReduction>1.0</mechanicalReduction>
  </actuator>
</transmission>

<transmission name="arm_2_joint_transmission">
  <type>transmission_interface/SimpleTransmission</type>
  <joint name="joint_2">
    <hardwareInterface>hardware_interface/PositionJointInterface</hardwareInterface>
  </joint>
  <actuator name="link2">
    <mechanicalReduction>1.0</mechanicalReduction>
  </actuator>
</transmission>
```

Hình 28: plugin và transmission cho tay máy

**Chú thích: Plugin để truyền topic vào cho tay máy còn transmission để định nghĩa cơ chế truyền động của mỗi joint.

+) Điều khiển tay máy

```
rospy.init_node('arm_teleop')
joint1_pub = rospy.Publisher('/arm_1_joint_controller/command', Float64, queue_size=10)
joint2_pub = rospy.Publisher('/arm_2_joint_controller/command', Float64, queue_size=10)
rospy.sleep(1)
```

Hình 29: Publish để điều khiển các góc ở 2 joint

**Chú thích: sau khi load được cấu hình từ file yaml, ta publish các góc vào joint1 và joint2 để điều khiển tay máy.

4.3 Camera

- Định nghĩa link và joint để đặt Camera lên xe

```
<link name="camera_link">
  <collision>
    <origin xyz="-0.05 0 -0.01" rpy="0 0 0"/>
    <geometry>
  <box size="0.05 0.05 0.05"/>
    </geometry>
  </collision>

  <visual>
    <origin xyz="-0.05 0 -0.01" rpy="0 0 0"/>
    <geometry>
  <box size="0.05 0.05 0.05"/>
    </geometry>
    <material name="gray"/>
  </visual>

  <inertial>
    <mass value="1e-5" />
    <origin xyz="0 0 0" rpy="0 0 0"/>
    <inertia ixx="1e-6" ixy="0" ixz="0" iyy="1e-6" iyz="0" izz="1e-6" />
  </inertial>
</link>
<joint name="camera_joint" type="fixed">
  <axis xyz="0 1 0" />
  <origin xyz="0.5 0 -0.01" rpy="0 0 0"/>
  <parent link="base_link"/>
  <child link="camera_link"/>
</joint>
```

Hình 30: Vị trí và hướng của Camera khi được đặt lên base_link

- Thêm plugin để hiển thị Camera

```
<gazebo reference="camera_link">
  <material>Gazebo/Red</material>

  <sensor name="camera" type="camera">
      <pose> 0 0 0 0 0 0 </pose>
      <visualize>true</visualize>
      <update_rate>10</update_rate>
      <camera>
          <horizontal_fov>1.089</horizontal_fov>
          <image>
              <format>R8G8B8</format>
              <width>640</width>
              <height>480</height>
          </image>
          <clip>
              <near>0.05</near>
              <far>8.0</far>
          </clip>
      </camera>
      <plugin name="camera_controller" filename="libgazebo_ros_camera.so">
          <frame_name>camera_link_optical</frame_name>
      </plugin>
  </sensor>
</gazebo>
```

Hình 31: Plugin của camera

4.4 Lidar

- Định nghĩa link và joint để đặt Lidar lên xe

```
<link name="lidar_link">
  <visual>
    <geometry>
      <cylinder length="0.05" radius="0.03"/>
    </geometry>
    <material name="black">
      <color rgba="0.0 0.0 0.0 1.0"/>
    </material>
  </visual>
  <collision>
    <geometry>
      <cylinder length="0.05" radius="0.03"/>
    </geometry>
  </collision>
  <inertial>
    <mass value="0.1"/>
    <origin xyz="0 0 0"/>
    <inertia ixx="0.001" ixy="0.0" ixz="0.0" iyy="0.001" iyz="0.0" izz="0.001"/>
  </inertial>
</link>

<joint name="lidar_joint" type="fixed">
  <parent link="base_link"/>
  <child link="lidar_link"/>
  <origin xyz="0.4 0 0.07" rpy="0 0 0"/>
</joint>
```

Hình 32: Vị trí và hướng của Lidar khi được đặt lên base_link

- Thêm plugin để hiển thị Lidar

```
<gazebo reference="lidar_link">
  <sensor type="ray" name="head_hokuyo_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>-1.57</min_angle>
          <max_angle>1.57</max_angle>
        </horizontal>
      </scan>
      <range>
        <min>0.10</min>
        <max>20.0</max>
        <resolution>0.01</resolution>
      </range>
      <noise>
        <type>gaussian</type>
        <!-- Noise parameters based on published spec for Hokuyo laser
             achieving "+-30mm" accuracy at range < 10m.  A mean of 0.0m and
             stddev of 0.01m will put 99.7% of samples within 0.03m of the true
             reading. -->
        <mean>0.0</mean>
        <stddev>0.01</stddev>
      </noise>
    </ray>
    <plugin name="gazebo_ros_head_hokuyo_controller" filename="libgazebo_ros_laser.so">
      <topicName>/rrbot/laser/scan</topicName>
      <frameName>lidar_link</frameName>
    </plugin>
  </sensor>
</gazebo>
```

Hình 33: Plugin để hiển thị Lidar

4.5 GPS

Do GPS gắn trực tiếp vào xe lên ta không cần link và joint, chỉ cần gắn trực tiếp vào base_link

- Thêm plugin và gắn vào xe cho GPS

```xml
<gazebo>
  <plugin name="gps_plugin" filename="libhector_gazebo_ros_gps.so">
    <alwaysOn>true</alwaysOn>
    <updateRate>10.0</updateRate>
    <bodyName>base_link</bodyName> <!-- Link gắn GPS -->
    <topicName>/gps/fix</topicName>
    <velocityTopicName>/gps/fix_velocity</velocityTopicName>
    <referenceLatitude>21.028511</referenceLatitude>
    <referenceLongitude>105.804817</referenceLongitude>
    <referenceAltitude>10.0</referenceAltitude>
    <drift>0.001 0.001 0.001</drift>
    <gaussianNoise>0.05 0.05 0.05</gaussianNoise>
  </plugin>
</gazebo>
```

Hình 34: Plugin cho GPS

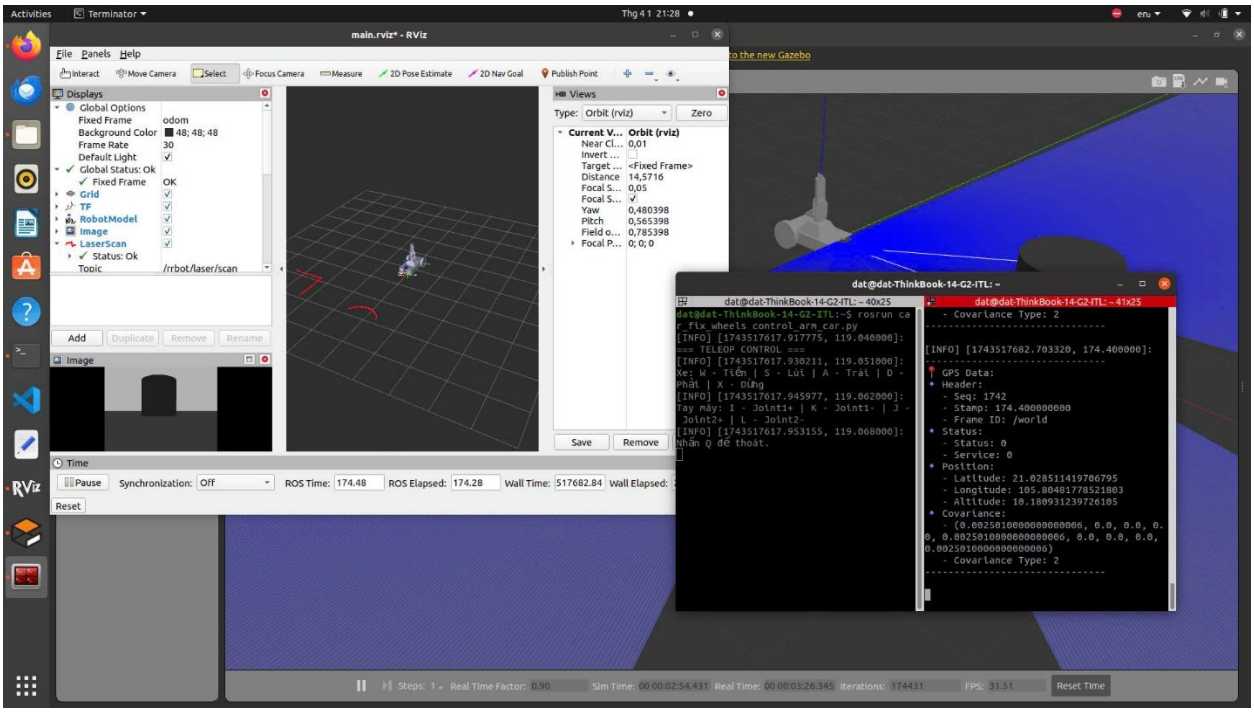
** Mở rộng ứng dụng của GPS: ta sử dụng GPS để điều khiển cho robot đi chính xác bao nhiêu mét

```python
class MoveWithGPS:
    def __init__(self, distance_goal, speed=0.5):
        rospy.init_node('move_with_gps', anonymous=True)
        self.cmd_vel_pub = rospy.Publisher('/cmd_vel', Twist, queue_size=10)
        rospy.Subscriber('/gps/fix', NavSatFix, self.gps_callback)

        self.start_latlon = None
        self.current_latlon = None

        self.distance_goal = abs(distance_goal)
        self.speed = speed if distance_goal > 0 else -speed
        self.moving = False

    def gps_callback(self, msg):
        """Hàm callback để cập nhật vị trí từ GPS."""
        self.current_latlon = (msg.latitude, msg.longitude)

        if self.start_latlon is None:
            self.start_latlon = self.current_latlon
            rospy.loginfo(f"📍 Điểm bắt đầu: {self.start_latlon}")

        if self.moving and self.start_latlon is not None:
            distance_traveled = geopy.distance.distance(self.start_latlon, self.current_latlon).m
            rospy.loginfo(f"🚗 Đã đi: {distance_traveled:.2f}m / {self.distance_goal}m")

            if distance_traveled >= self.distance_goal - 0.05:
                self.stop()
                rospy.loginfo("✅ Xe đã đi đủ khoảng cách!")
```

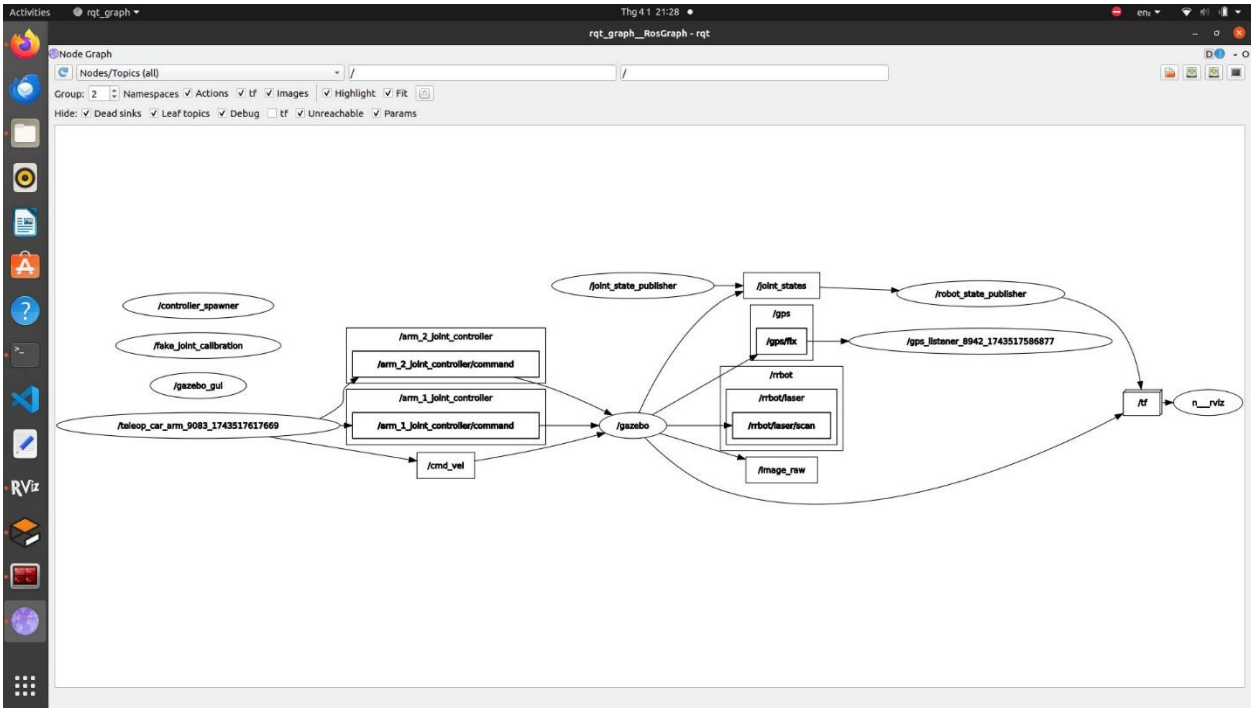
Hình 35: Class để điều khiển chính xác

(ta cần thêm điều kiện distance_traveled >= distance_goal – 0.05 là do 0.05 là khoảng cách để xe từ lúc đang đi đến khi dừng đúng tại điểm ta yêu cầu)

5. Kết quả



Hình 36: Hiển thị các cảm biến Lidar, Camera, GPS



Hình 37: rqt_graph giữa các node điều khiển xe

- Kết quả mô phỏng: Link

- Link source: Link