

**ĐẠI HỌC QUỐC GIA HÀ NỘI  
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ  
KHOA ĐIỆN TỬ - VIỄN THÔNG**



**BÀI GIỮA KỲ THỰC HÀNH ROS  
MÔ PHỎNG XE OMNI 4 BÁNH**

**Người thực hiện:**

Nguyễn Kiều Đức Phú	22027540
QH-2022-I/CQ-R	Khoa Điện tử Viễn thông

Hà Nội, 2025

<b>Khái quát chung</b>	<b>3</b>
a. Đề bài	3
b. Bánh omni đa hướng	3
c. Cánh tay máy 2 khớp	5
d. Camera	6
e. Encoder	7
f. Cảm biến IMU	8
<b>Quá trình làm bài</b>	<b>9</b>
a. Solidwork	9
b. Xuất URDF và chỉnh sửa	12
c. Code để điều khiển xe	16
d. Code để điều khiển tay máy	18
e. Code cho cảm biến IMU	19
f. Code điều khiển cho yaml	20
g. Code lại cho gazebo	21
<b>KẾT QUẢ</b>	<b>22</b>
1. Di chuyển xe:	22
2. Di chuyển tay máy	23
3. Chạy camera	24
4. Chạy IMU	24
5. Chạy Encoder	25

## **Khái quát chung**

### **a. Đề bài**

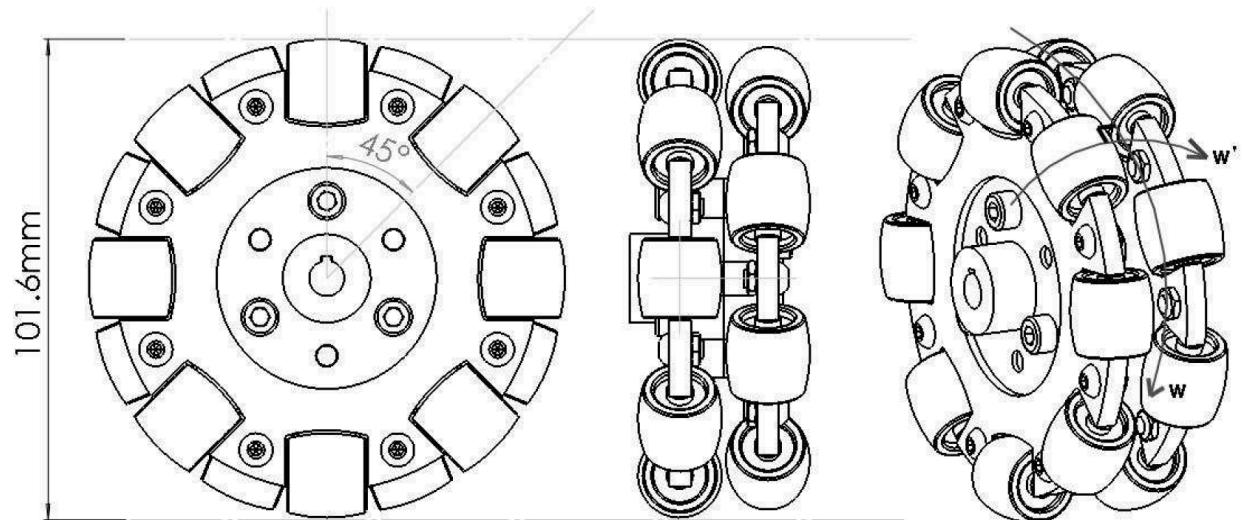
Đề bài giữa kì được giao sẽ là điều khiển 1 con Robot với 4 bánh omni đa hướng cộng với đó sẽ là 3 loại cảm biến: - Camera

- IMU
- Encoder

Đồng thời sẽ phải điều khiển cánh tay robot có 2 khớp, khớp 1 sẽ là khớp tĩnh tiến và khớp 2 sẽ là khớp xoay

### **b. Bánh omni đa hướng**

Robot di động có bánh omni được sử dụng rộng rãi trong công nghiệp và dịch vụ do khả năng di chuyển đa hướng. Các nghiên cứu truyền thống tập trung vào cấu trúc bánh đối xứng để đảm bảo tính ổn định. Tuy nhiên, bố trí bánh xe bất đối xứng có thể giúp robot ổn định hơn, giảm nguy cơ lật và thu gọn kích thước.





### Mô-men vận tốc (Velocity Moment)

Lực đẩy của bánh xe khó đo chính xác do cấu trúc phức tạp của các con lăn. Giả định:  
Lực đẩy của bánh xe tỷ lệ thuận với vận tốc tiếp xúc của bánh xe với mặt đất.

Tốc độ quay của bánh xe:

$$\omega = V/r$$

Trong đó - V là vận tốc tiếp xúc của bánh xe  
- r là bán kính của chiếc xe

Ta có I' là khoảng cách từ vector vận tốc của bánh xe đến tâm quay của robot

$$l' = x \cdot \sin\theta - y \cdot \cos\theta = l \cdot \sin(\theta - \beta)$$

Trong đó - (x,y) là vị trí của bánh xe so với tâm robot.

-  $\theta$ i là góc định hướng của bánh xe.

- $l$  là khoảng cách từ bánh xe đến tâm robot.
- $\beta$  là góc hướng của bánh xe so với tâm robot.

Ta có Phương trình động học ngược của bánh Omni:

$$\begin{pmatrix} V_1 \\ V_2 \\ V_3 \\ \vdots \\ V_n \end{pmatrix} = - \begin{pmatrix} \cos(\theta_1) & \sin(\theta_1) & l_1 \sin(\theta_1 - \beta_1) \\ \cos(\theta_2) & \sin(\theta_2) & l_2 \sin(\theta_2 - \beta_2) \\ \cos(\theta_3) & \sin(\theta_3) & l_3 \sin(\theta_3 - \beta_3) \\ \vdots & \vdots & \vdots \\ \cos(\theta_n) & \sin(\theta_n) & l_n \sin(\theta_n - \beta_n) \end{pmatrix} \begin{pmatrix} V_x \\ V_y \\ \Omega \end{pmatrix}$$

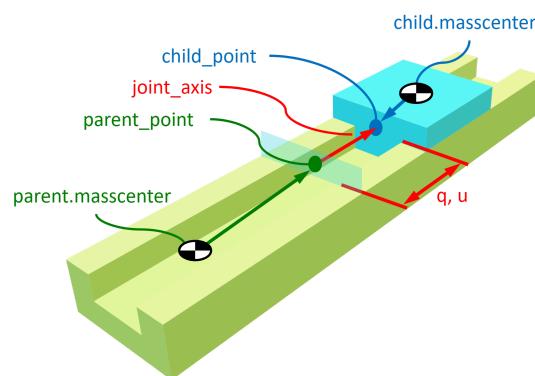
Trong khi đó:  $V$  là Tốc độ của bánh xe khi robot di chuyển.

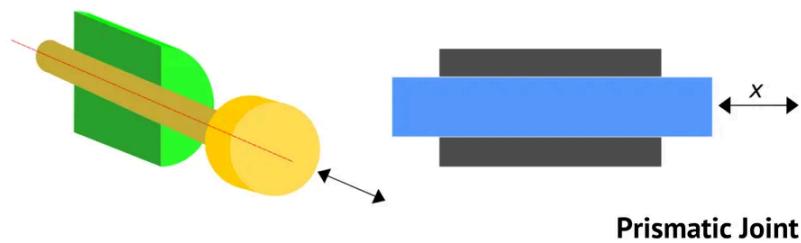
$\Omega$  (Vận tốc góc của khối tâm robot): Tốc độ quay của toàn bộ robot.

=> Dấu âm (-) xuất hiện do bánh Omni tạo ra các lực phản ứng ngược với phương chuyển động mong muốn. Phương trình này cho phép chúng ta tính toán vận tốc của từng bánh xe dựa trên chuyển động mong muốn của robot

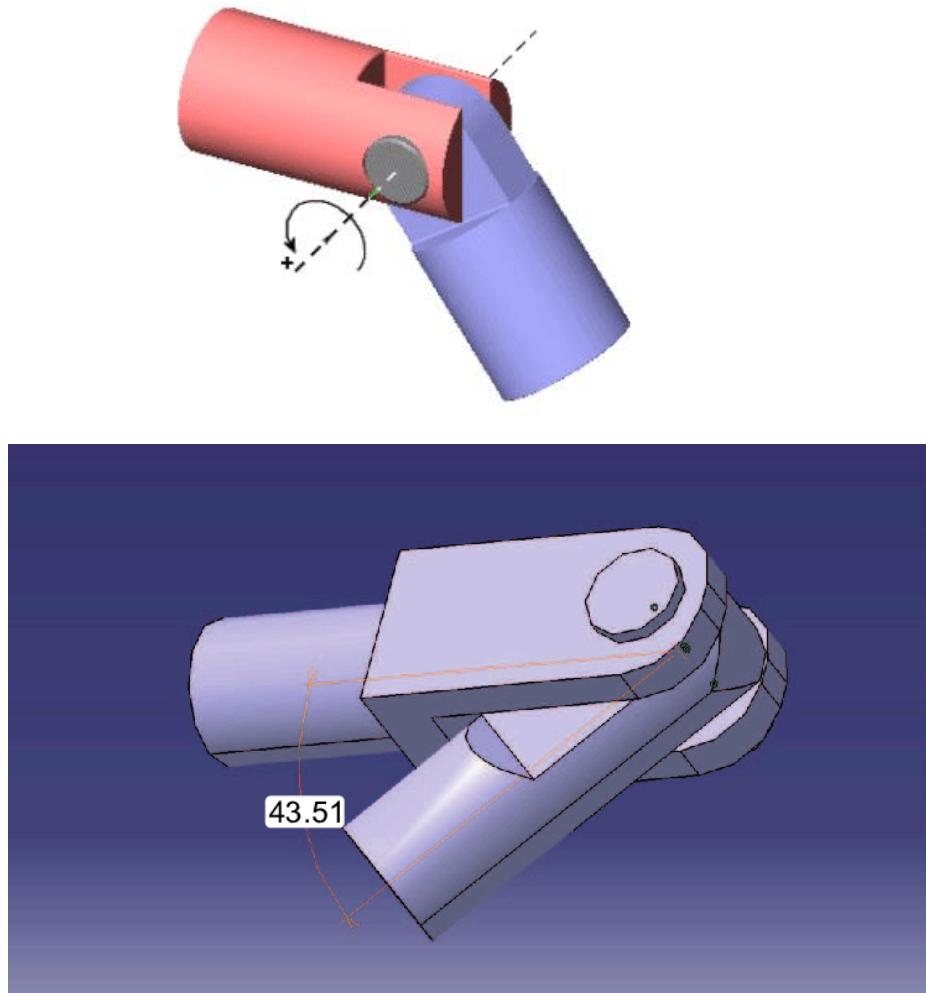
### c. Cánh tay máy 2 khớp

Khớp đầu tiên là khớp tĩnh tiến Là loại khớp cho phép chuyển động thẳng theo một hướng duy nhất mà không quay. Độ tự do (DOF): 1 DOF, chỉ di chuyển theo trục được xác định.





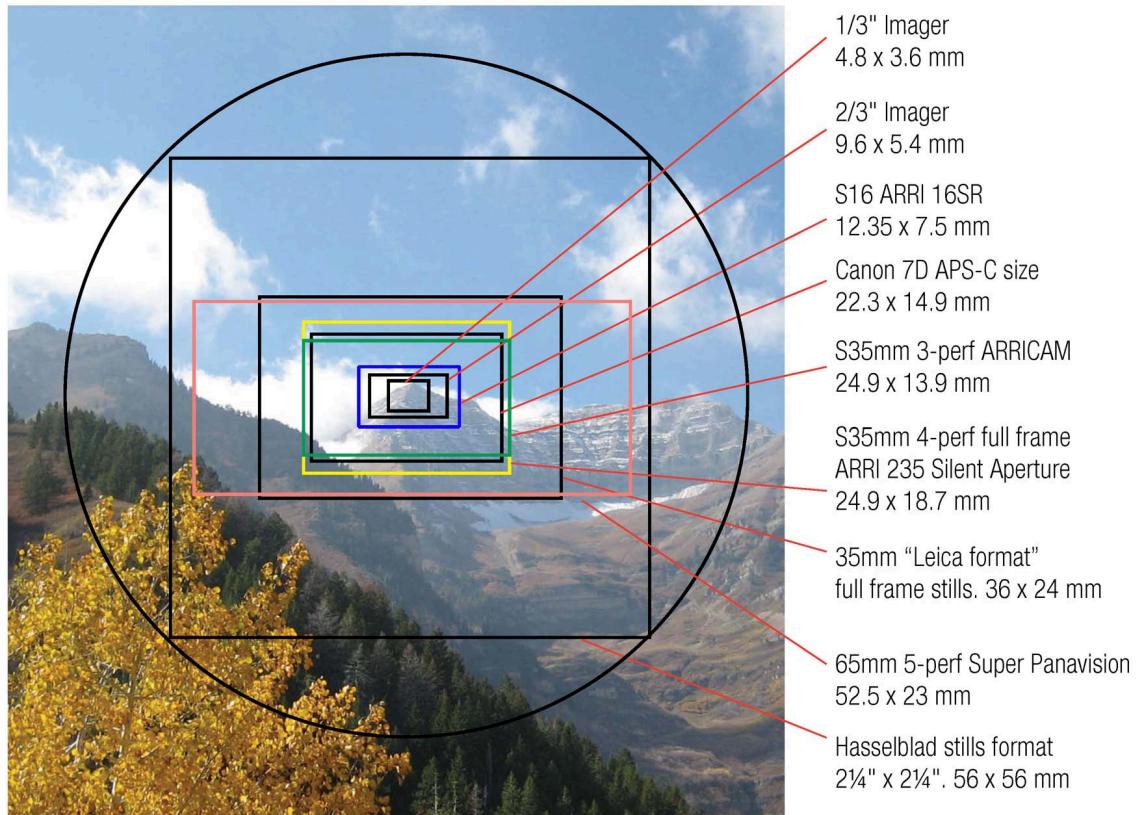
Khớp thứ 2 là khớp xoay Là loại khớp cho phép quay quanh một trục cố định.  
Độ tự do (DOF): 1 DOF, chỉ quay quanh một trục duy nhất.



#### d. Camera

Cảm biến Camera bao gồm có cảm biến hình ảnh và cảm biến ánh sáng, Cảm biến hình ảnh là một trong những bộ phận quan trọng nhất của máy ảnh. Cảm biến hình ảnh

này được sản xuất từ silic là chính, (tương tự như việc sản xuất chip xử lý trên máy tính), chế tạo thành các miếng wafer siêu mỏng được đúc đẽo tinh vi theo công nghệ của từng hãng.



### e. Encoder

Encoder (Rotary Encoder) là một bộ cảm biến chuyển động cơ học tạo ra tín hiệu analog hoặc tín hiệu kỹ thuật số (digital) đáp ứng với chuyển động. Loại thiết bị cơ điện này có khả năng biến đổi chuyển động (chuyển động tịnh tiến, chuyển động quay của trục, ...) thành tín hiệu đầu ra số hoặc xung. Encoder hay Bộ mã hóa vòng quay được ứng dụng chủ yếu để phát hiện vị trí, hướng di chuyển, tốc độ... của động cơ bằng cách đếm số vòng mà trục quay được. Bạn hãy liên tưởng rằng Encoder đóng vai trò như là bộ phận công tơ mét trên xe máy hay ô tô.



### f. Cảm biến IMU

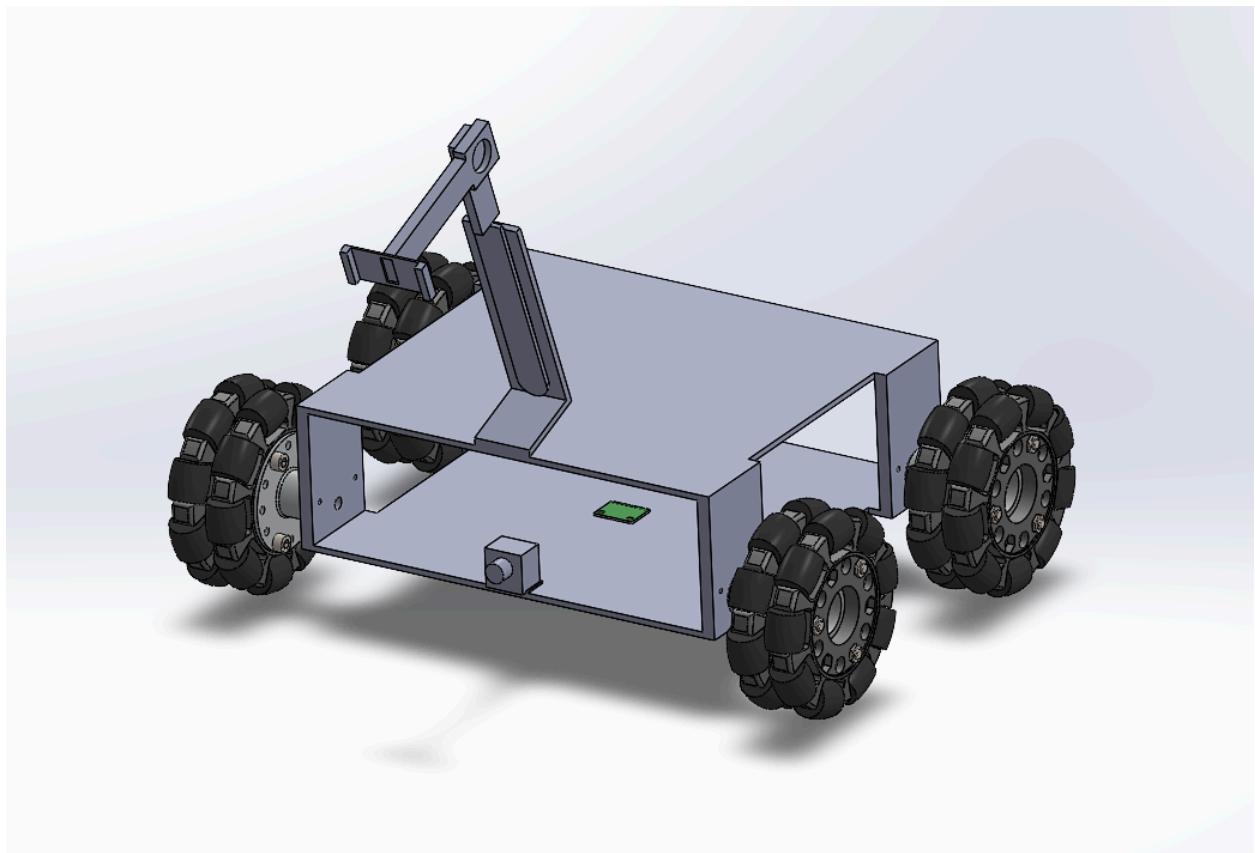
Cảm biến IMU là một thiết bị điện tử được sử dụng để tính toán và báo cáo lực chính xác của cơ thể, tốc độ góc cũng như hướng của cơ thể, điều này có thể đạt được bằng cách sử dụng sự kết hợp của 3 cảm biến như con quay hồi chuyển, từ kế và gia tốc kế. Những cảm biến này thường được sử dụng để lập kế hoạch cho máy bay bao gồm UAV (phương tiện bay không người lái), giữa nhiều thứ khác, và tàu vũ trụ, bao gồm cả tàu đổ bộ và vệ tinh. Những phát triển hiện đại cho phép sản xuất các thiết bị GPS dựa trên IMU.

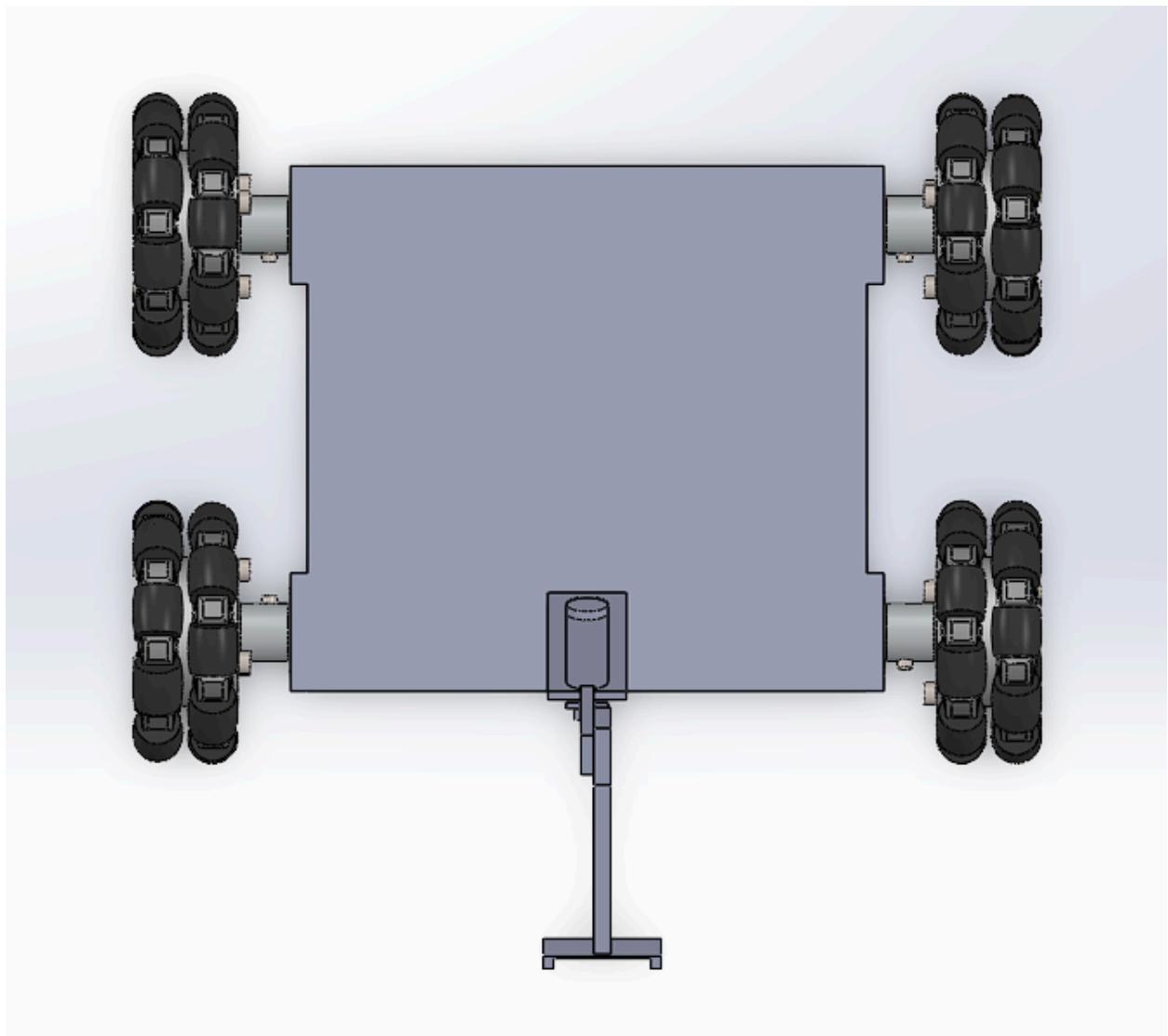
Hoạt động của một đơn vị cảm biến IMU có thể được thực hiện bằng cách nhận biết gia tốc tuyến tính với sự trợ giúp của một hoặc nhiều gia tốc kế và tốc độ quay có thể được phát hiện bằng cách sử dụng một hoặc nhiều con quay hồi chuyển. Một số cũng chứa từ kế có thể được sử dụng làm tham chiếu hướng. Cảm biến này bao gồm một số cấu hình thông thường bao gồm con quay hồi chuyển, một gia tốc kế và từ kế cho mỗi trục được sử dụng cho mỗi trong ba trục của phương tiện như lăn, quay và nghiêng.

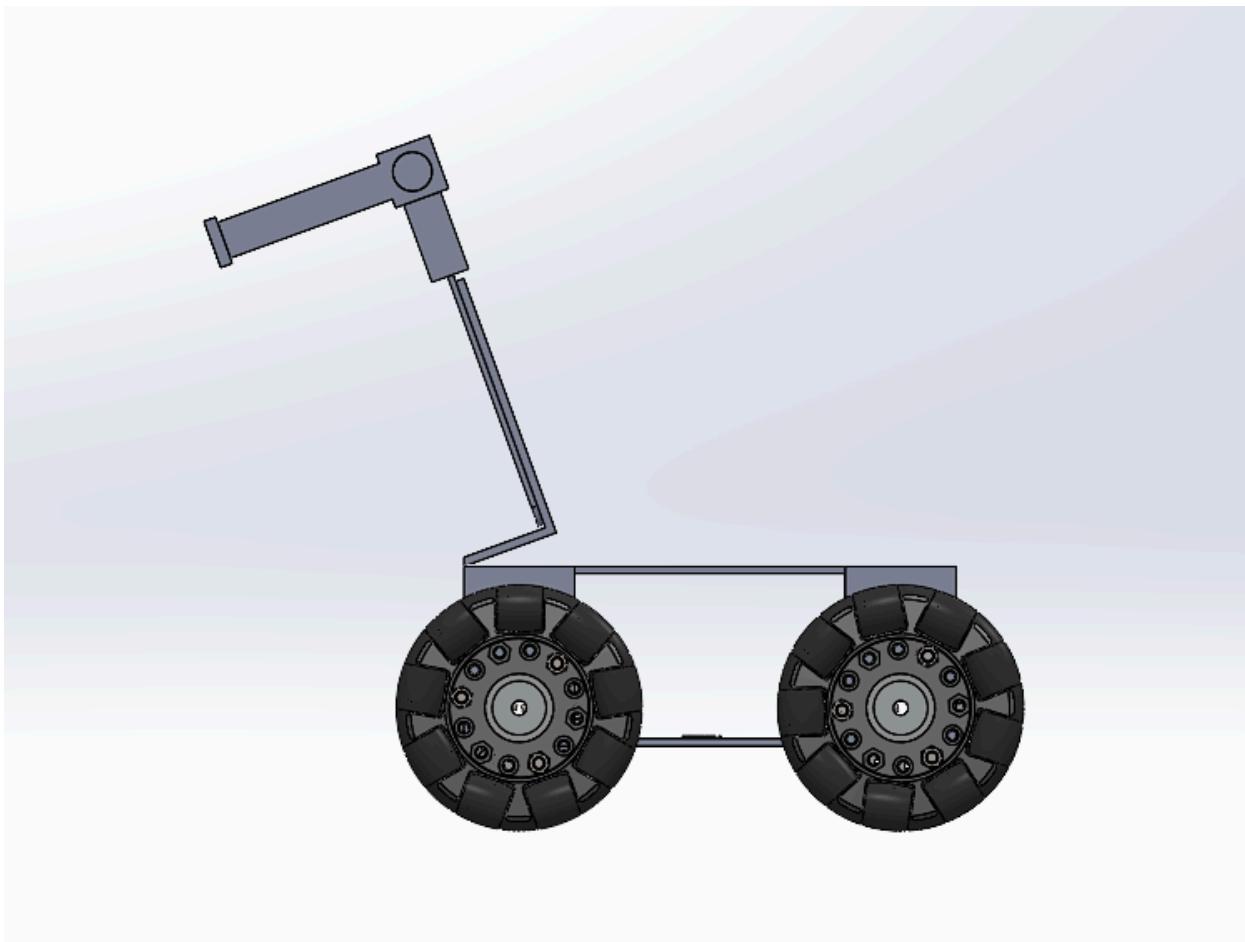
## **Quá trình làm bài**

### **a. Solidwork**

Dưới đây là mô hình Robot







## b. Xuất URDF và chỉnh sửa

Chọn Export as URDF ta sẽ có file của URDF

Name	Date modified	Type
config	3/29/2025 8:01 PM	File folder
launch	3/29/2025 8:01 PM	File folder
meshes	3/29/2025 8:01 PM	File folder
textures	3/29/2025 8:01 PM	File folder
urdf	3/29/2025 8:01 PM	File folder
CMakeLists.txt	3/29/2025 8:01 PM	Text Document
export.log	3/29/2025 8:01 PM	Text Document
package.xml	3/29/2025 8:01 PM	Microsoft Edge

Khi đọc các file urdf ta sẽ có các tên cùng với các link và vị trí được định nghĩa đầy đủ trong file urdf

```
<link
  name="base_link">
  <inertial>
    <origin
      xyz="0.00239506949251722 -1.11022302462516E-16 -0.0337760800101865"
      rpy="0 0 0" />
    <mass
      value="0.34555530282193" />
    <inertia
      ixx="0.00235597995164024"
      ixy="2.34668227270661E-18"
      ixz="-6.35299748605835E-05"
      iyy="0.00185302876841725"
      iyz="1.65981291586631E-18"
      izz="0.00337461610296602" />
  </inertial>
  <visual>
    <origin
      xyz="0 0 0"
      rpy="0 0 0" />
    <geometry>
      <mesh
        filename="package://rosstkfinal/meshes/base_link.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://rosstkfinal/meshes/base_link.STL" />
    </geometry>
  </collision>
</link>
```

Ví dụ: Tên: base\_link

Góc tại: 0.002...

...

Đưa sang ROS ta sẽ thêm gốc cho odom

```
<link name="odom">
  <visual>
    <origin xyz="0 0 0" rpy="0 0 0" />
    <geometry>
      <box size="0.01 0.01 0.01" /> <!-- Simple box to represent the link -->
    </geometry>
    <material name="default">
      <color rgba="0.5 0.5 0.5 1" /> <!-- Neutral gray color -->
    </material>
  </visual>
  <collision>
    <origin xyz="0 0 0" rpy="0 0 0" />
    <geometry>
      <box size="0.01 0.01 0.01" />
    </geometry>
  </collision>
</link>

<joint name="odom_to_base" type="fixed">
  <parent link="odom" />
  <child link="base_link" />
  <origin xyz="0 0 0" rpy="0 0 0" />
</joint>
```

Mục đích để tạo odom là gắn với mặt đất hoặc một mốc cố định trong không gian. Khi robot di chuyển, bộ đo quán tính (IMU), encoder bánh xe, hoặc cảm biến khác sẽ cập nhật vị trí của robot tương đối so với odom. Điều này giúp robot biết được nó đã di chuyển bao xa và xoay bao nhiêu kể từ khi khởi động.

```

<gazebo reference="camera_link">
  <sensor type="camera" name="cameral">
    <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>
        <!-- Noise is sampled independently per pixel on each frame.
            That pixel's noise value is added to each of its color
            channels, which at that point lie in the range [0,1]. -->
        <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/cameral</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>
      <distortionT1>0.0</distortionT1>
      <distortionT2>0.0</distortionT2>
    </plugin>
  </sensor>

```

Ta sẽ có resolution của camera sẽ là 800x800. Số lần mỗi giây mà một hình ảnh mới được camera chụp trong Gazebo. 30 là tốc độ cập nhật tối đa mà cảm biến sẽ có găng đạt được trong quá trình mô phỏng, nhưng có thể chậm hơn nếu tốc độ mô phỏng vật lý nhanh hơn tốc độ tạo ảnh của cảm biến. Tại đây, chúng ta xác định các **rostopic** mà camera sẽ xuất bản dữ liệu hình ảnh và thông tin camera.

```

<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>

```

Đoạn code trên mô phỏng một cảm biến IMU trong Gazebo và xuất dữ liệu sang ROS. Cụ thể:

Gắn cảm biến IMU vào liên kết imu\_link và chịu ảnh hưởng của trọng lực. Cập nhật dữ liệu với tần số 100Hz trong Gazebo, nhưng chỉ gửi dữ liệu lên ROS với 10 Hz. Xuất dữ liệu IMU lên topic /imu trong ROS. Không có nhiễu, không có độ lệch vị trí/góc quay. Sử dụng plugin libgazebo\_ros\_imu\_sensor.so để ROS có thể đọc dữ liệu.

```

<gazebo>
  <plugin name="skid_steer_drive_controller" filename="libgazebo_ros_skid_steer_drive.so">
    <updateRate>100.0</updateRate>
    <robotNamespace>/</robotNamespace>
    <leftFrontJoint>left_front.omni</leftFrontJoint>
    <rightFrontJoint>right_front.omni</rightFrontJoint>
    <leftRearJoint>left_back.omni</leftRearJoint>
    <rightRearJoint>right_back.omni</rightRearJoint>
    <wheelSeparation>0.4</wheelSeparation>
    <wheelDiameter>0.215</wheelDiameter>
    <robotBaseFrame>base_link</robotBaseFrame>
    <torque>20</torque>
    <topicName>/cmd_vel</topicName>
    <broadcastTF>false</broadcastTF>
  </plugin>
</gazebo>

```

Đây là dòng lệnh cho phép di chuyển đơn giản cho bánh xe trong Gazebo

### c. Code để điều khiển xe

Đầu tiên ta sẽ tạo hàm điều khiển bánh xe: control.py (Code này được tham khảo từ code teleop\_twist\_key.py)

Đầu tiên ta sẽ tạo ra các bàn phím sẽ dùng để điều khiển. Sau đó tạo 1 class để chạy nền mục đích là để xuất lệnh vận tốc liên tục lên topic cmd\_vel bao gồm tọa độ di chuyển và tốc độ

```
moveBindings = {
    'q': (1, 0, 0, 0),
    'e': (-1, 0, 0, 0),
    'a': (0, 1, 0, 0),
    'd': (0, -1, 0, 0),
    'w': (0, 0, 0, 1),
    's': (0, 0, 0, -1),
    'f': (0, 0, 0, 0),
}

speedBindings={
    'q':(1.1,1.1),
    'z':(.9,.9),
    'w':(1.1,1.0),
    'x':(.9,1),
    'e':(1,1.1),
    'c':(1,.9),
}

class PublishThread(threading.Thread):
    def __init__(self, rate):
        super(PublishThread, self).__init__()
        self.publisher = rospy.Publisher('cmd_vel', TwistMsg, queue_size = 1)
        self.x = 0.0
        self.y = 0.0
        self.z = 0.0
        self.th = 0.0
        self.speed = 1
        self.turn = 0.0
        self.condition = threading.Condition()
        self.done = False
```

Tiếp đến ta có hàm chờ subscriber cho đến khi kết nối với cmd\_vel và sau khi đã kết nối code sẽ chuyển đến phần tiếp theo là cập nhật dữ liệu và gửi lệnh vận tốc và vị trí lại cho topic cmd\_vel

```
def wait_for_subscribers(self):
    i = 0
    while not rospy.is_shutdown() and self.publisher.get_num_connections() == 0:
        if i == 4:
            print("Waiting for subscriber to connect to {}".format(self.publisher.name))
        rospy.sleep(0.5)
        i += 1
        i = i % 5
    if rospy.is_shutdown():
        raise Exception("Got shutdown request before subscribers connected")
```

```

def update(self, x, y, z, th, speed, turn):
    self.condition.acquire()
    self.x = x
    self.y = y
    self.z = z
    self.th = th
    self.speed = speed
    self.turn = turn
    # Notify publish thread that we have a new message.
    self.condition.notify()
    self.condition.release()

def stop(self):
    self.done = True
    self.update(0, 0, 0, 0, 0, 0)
    self.join()

def run(self):
    twist_msg = TwistMsg()

    if stamped:
        twist = twist_msg.twist
        twist_msg.header.stamp = rospy.Time.now()
        twist_msg.header.frame_id = twist_frame
    else:
        twist = twist_msg

    while not self.done:
        if stamped:
            twist_msg.header.stamp = rospy.Time.now()
        self.condition.acquire()
        # Wait for a new message or timeout.
        self.condition.wait(self.timeout)

        # Copy state into twist message.
        twist.linear.x = self.x * self.speed
        twist.linear.y = self.y * self.speed
        twist.linear.z = self.z * self.speed
        twist.angular.x = 0
        twist.angular.y = 0
        twist.angular.z = self.th * self.turn

        self.condition.release()

        # Publish.
        self.publisher.publish(twist_msg)

    # Publish stop message when thread exits.
    twist.linear.x = 0
    twist.linear.y = 0
    twist.linear.z = 0
    twist.angular.x = 0
    twist.angular.y = 0
    twist.angular.z = 0
    self.publisher.publish(twist_msg)

```

Sau đó ta sẽ có đọc bàn phím và gửi lệnh di chuyển sau khi đó nếu đã dừng robot thì sẽ xử lý để thoát chương trình

#### d. Code để điều khiển tay máy

Đầu tiên ta sẽ có hàm nhận từ bàn phím

```

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

```

Tiếp đó là hàm chính trong việc điều khiển tay máy sau đó ta sẽ gọi mỗi tay máy 1 topic riêng vì sau đó ta sẽ gọi từng hàm topic để điều khiển tay máy trong mô phỏng

```

def arm_controller():
    rospy.init_node('arm_teleop_keyboard', anonymous=True)
    arm_1_pub = rospy.Publisher('/arm_1_controller/command', Float64, queue_size=10)
    arm_2_pub = rospy.Publisher('/arm_2_controller/command', Float64, queue_size=10)
    arm_1_pos = 0.0
    arm_2_pos = 0.0
    rate = rospy.Rate(50)
    while not rospy.is_shutdown():
        key = get_key()
        if key == 'u':
            arm_2_pos += 0.5
            if arm_2_pos > 3.14:
                arm_2_pos = 3.14
        elif key == 'i':
            arm_2_pos -= 0.5
            if arm_2_pos < -3.14:
                arm_2_pos = -3.14
        elif key == 'j':
            arm_1_pos += 0.1
            if arm_1_pos > 0.6:
                arm_1_pos = 0.6
        elif key == 'k':
            arm_1_pos -= 0.1
            if arm_1_pos < 0.0:
                arm_1_pos = 0.0
        elif (key == 'p'):
            break
        arm_1_pub.publish(arm_1_pos)
        arm_2_pub.publish(arm_2_pos)
        rate.sleep()

```

### e. Code cho cảm biến IMU

Code được viết ra để bắt đến topic joint\_states và đọc trực tiếp giá trị và vận tốc của motor

```

// encoder.py > Q: encoder_display
#!/usr/bin/env python3
import rospy
from sensor_msgs.msg import JointState

def joint_state_callback(msg):
    joints_to_display = ['right_front.omni', 'right_back.omni', 'left_back.omni', 'left_front.omni']

    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
            rospy.loginfo("%s: Position = %.3f rad, Velocity = %.3f rad/s", joint_name, position, velocity)

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

    rospy.Subscriber('/joint_states', JointState, joint_state_callback)

    rospy.loginfo("Hiển thị giá trị encoder của các khớp...")
    rospy.spin()

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

```

## f. Code điều khiển cho yaml

```

controller_manager:
  ros:
    auto_start: true
    update_rate: 100

  joint_state_controller:
    type: joint_state_controller/JointStateController
    publish_rate: 50
  arm_1_controller:
    type: position_controllers/JointPositionController
    joint: joint1

  arm_2_controller:
    type: position_controllers/JointPositionController
    joint: joint2

  gazebo_ros_control:
    pid_gains:
      right_front.omni: {p: 1.0, i: 0.0, d: 0.0}
      left_front.omni: {p: 1.0, i: 0.0, d: 0.0}
      right_back.omni: {p: 1.0, i: 0.0, d: 0.0}
      left_back.omni: {p: 1.0, i: 0.0, d: 0.0}
      joint1: {p: 1, i: 0.01, d: 0.0}
      joint2: {p: 1, i: 0.01, d: 0.0}

```

Nó sẽ cung cấp thông tin từ joint\_state\_controller và sau đó cung cấp hàm điều khiển của joint1 và joint2 bằng bộ điều khiển

Tiếp đến là phần cung cấp PID cho từng khâu như bánh trái trên, bánh phải trên, bánh trái dưới, bánh phải dưới, khớp 1, khớp 2 để giúp cho điều khiển dễ dàng hơn

## g. Code lại cho gazebo

```
<launch>
  <!-- Parameters for Gazebo -->
  <arg name="paused" default="false"/>
  <arg name="use_sim_time" default="true"/>
  <arg name="gui" default="true"/>
  <arg name="headless" default="false"/>
  <arg name="debug" default="false"/>
  <arg name="enable_keyboard" default="true"/>

  <!-- Launch Gazebo -->
  <include file="$(find gazebo_ros)/launch/empty_world.launch">
    <arg name="paused" value="$(arg paused)"/>
    <arg name="use_sim_time" value="$(arg use_sim_time)"/>
    <arg name="gui" value="$(arg gui)"/>
    <arg name="headless" value="$(arg headless)"/>
    <arg name="debug" value="$(arg debug)"/>
  </include>

  <param name="robot_description" textfile="$(find rosgkfinal)/urdf/rosgkfinal.urdf" />

  <node name="spawn_model" pkg="gazebo_ros" type="spawn_model"
    args="-file $(find rosgkfinal)/urdf/rosgkfinal.urdf -urdf -model rosgkfinal"
    output="screen" />

  <include file="$(find rosgkfinal)/launch/display.launch"/>

  <rosparam file="$(find rosgkfinal)/config/joint_names_rosgkfinal.yaml" command="load"/>

  <node name="control" pkg="rosgkfinal" type="controller.py" output="screen">
    <remap from="cmd_vel" to="/cmd_vel"/>
    <param name="speed" value="0.5"/>
    <param name="turn" value="1.0"/>
  </node>

  <node name="jointcontrol" pkg="rosgkfinal" type="jointcontrol.py" output="screen" if="$(arg enable_keyboard)"/>

  <node name="robot_state_publisher" pkg="robot_state_publisher"
    type="robot_state_publisher" respawn="false" output="screen"/>

  <node name="controller_spawner" pkg="controller_manager" type="spawner"
    args="joint_state_controller arm_1_controller arm_2_controller"
    output="screen"
    launch-prefix="bash -c 'sleep 5; $0 $@'" />

  <node name="controller_manager" pkg="controller_manager" type="controller_manager" output="screen" />

  <node name="tf_footprint_base" pkg="tf" type="static_transform_publisher"
    args="0 0 0 0 0 base_link base_footprint 40" />

  <node name="fake_joint_calibration" pkg="rostopic" type="rostopic"
    args="pub /calibrated std_msgs/Bool true" />

  <node name="camera_view" pkg="rqt_image_view" type="rqt_image_view"
    args="/rrbot/camerai/image_raw" />

  <!-- <node name="encoder" pkg="rosgkfinal" type="encoder.py" output="screen">
    <remap from="encoder_1" to="/joint_states"/> -->
  <!-- </node> -->
</launch>
```

Mở đầu sẽ là để mở gazebo và add model vào trong gazebo tiếp đến là nạp file điều khiển: điều khiển khớp yaml; điều khiển bánh xe đuôi .py; điều khiển cánh tay máy .py Xuất trạng thái khớp: Node robot\_state\_publisher giúp xuất thông tin về trạng thái khớp của robot lên ROS. Dữ liệu này có thể được sử dụng trong Rviz để hiển thị trạng thái robot. Khởi động bộ điều khiển

Chạy controller\_spawner để kích hoạt các bộ điều khiển khớp. Các bộ điều khiển được kích hoạt:

joint\_state\_controller → Xuất trạng thái khớp.

arm\_1\_controller → Điều khiển khớp 1.

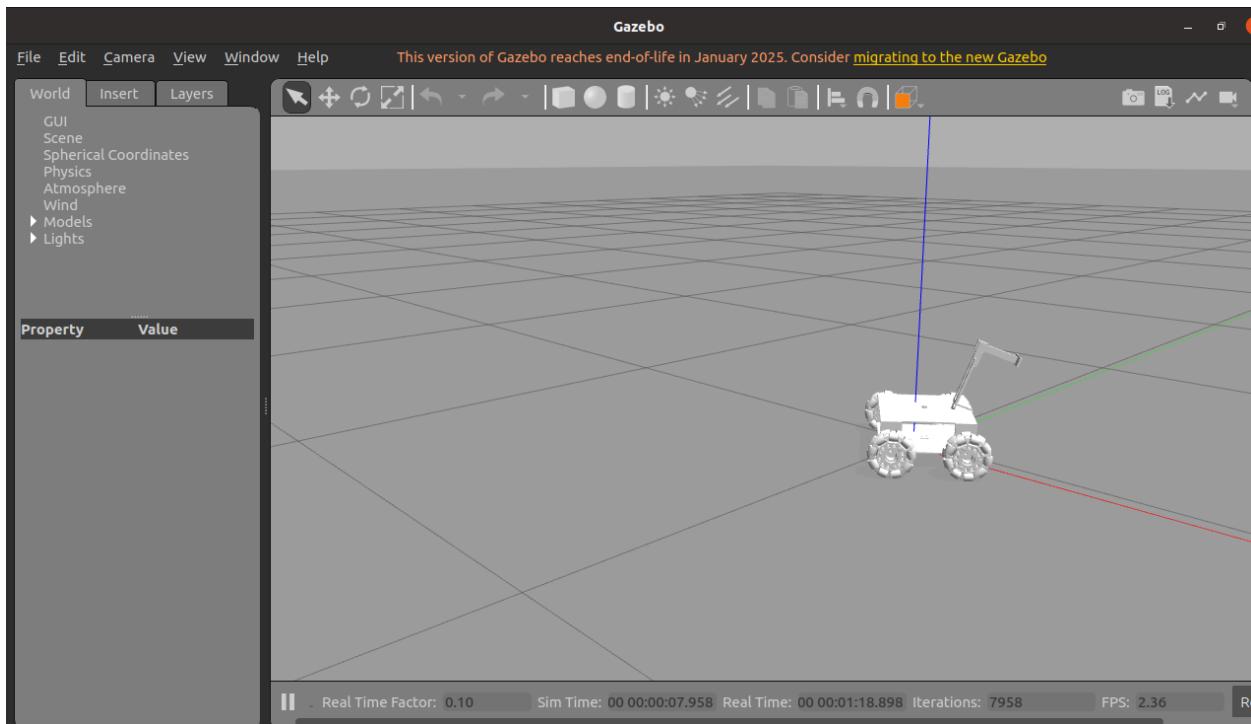
arm\_2\_controller → Điều khiển khớp 2.

Quản lý bộ điều khiển được khởi động node controller\_manager để quản lý tất cả các bộ điều khiển. Xuất TF giữa các khung tọa độ, xuất một TF tĩnh giữa base\_link và base\_footprint với tốc độ 40 Hz. TF này giúp định vị robot trong không gian.

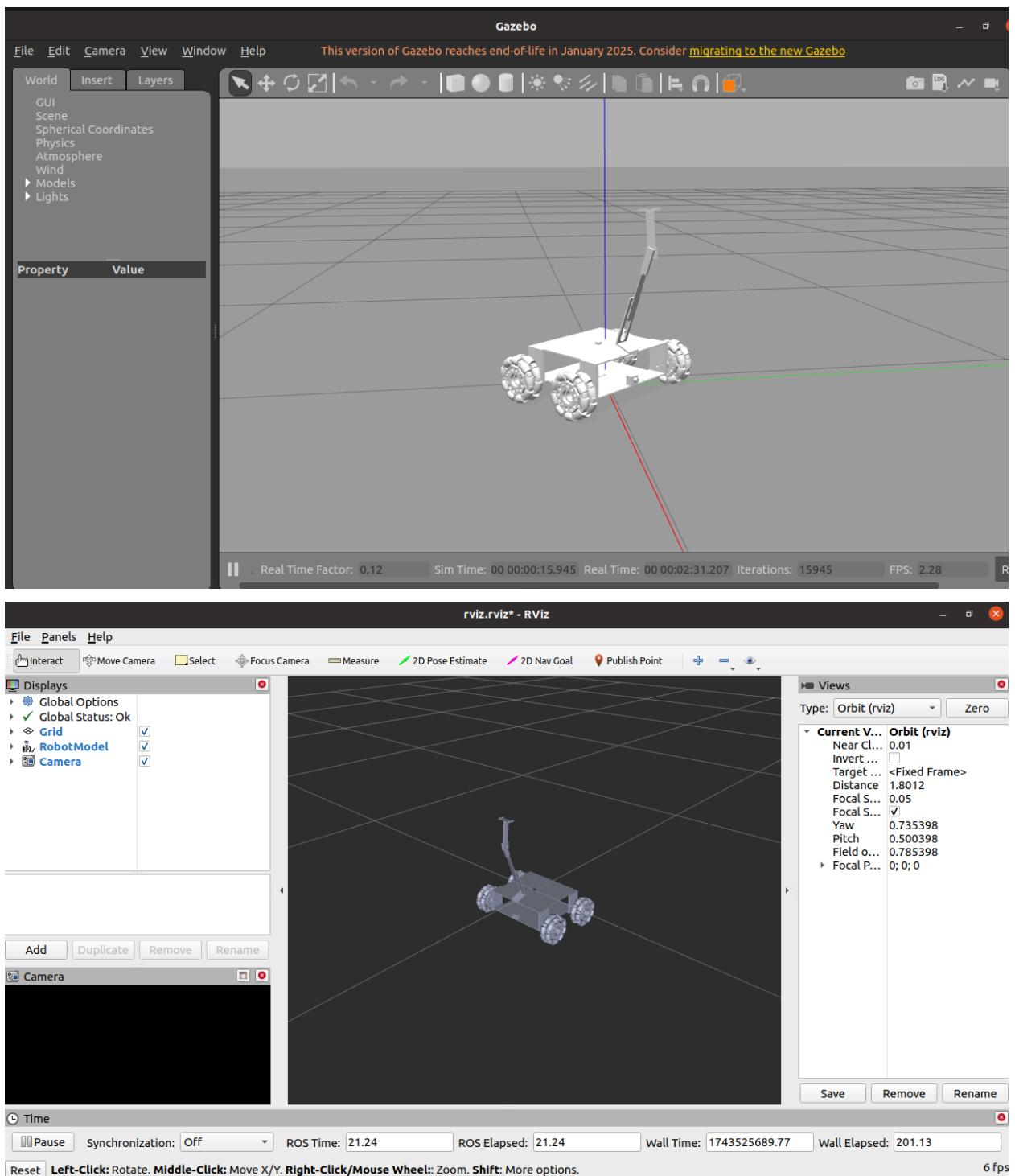
Hiệu chỉnh khớp (giả lập) sẽ gửi một giá trị true đến topic /calibrated để báo hiệu rằng khớp đã được hiệu chỉnh. Hiển thị Camera bằng cách mở cửa sổ rqt\_image\_view để xem hình ảnh từ camera của robot.

## KẾT QUẢ

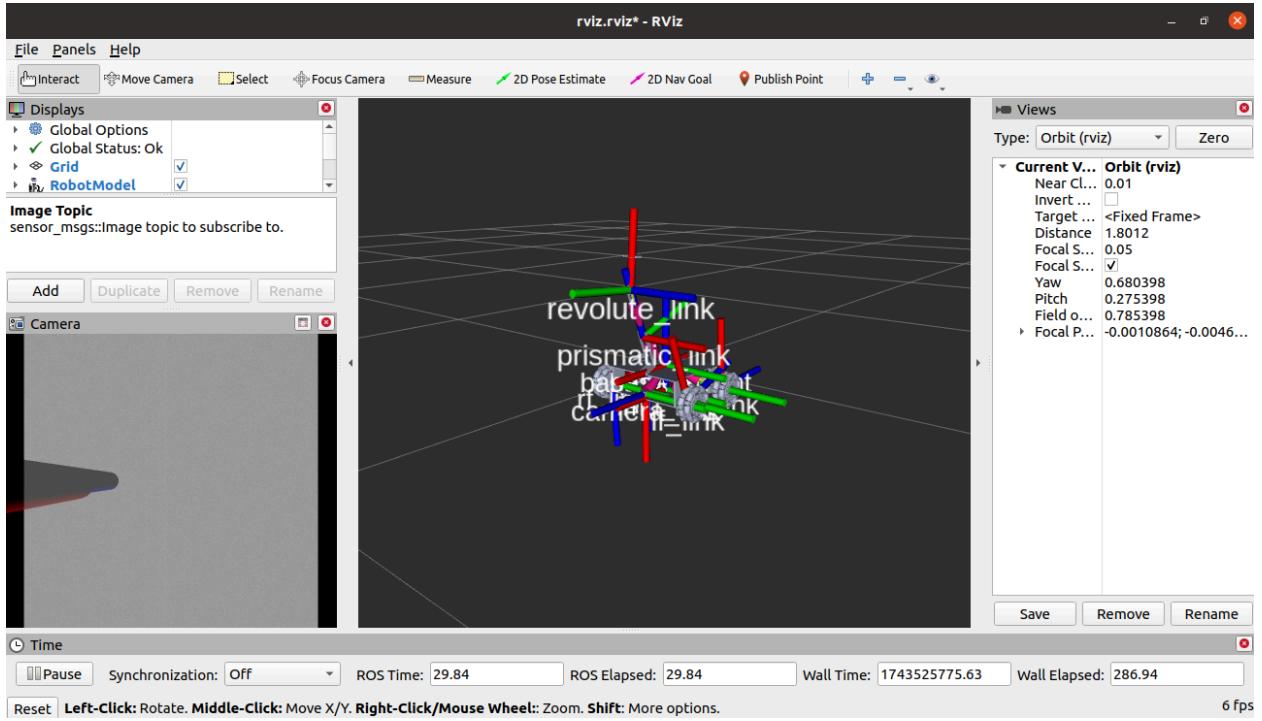
### 1. Di chuyển xe:



## 2. Di chuyển tay máy



### 3. Chạy camera



### 4. Chạy IMU

```
header:  
  seq: 5  
  stamp:  
    secs: 37  
    nsecs: 198000000  
    frame_id: "imu_link"  
orientation:  
  x: -0.23206099836940086  
  y: 0.6679496283732743  
  z: 0.23205449891808583  
  w: 0.6679383927606485  
orientation_covariance: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]  
angular_velocity:  
  x: 0.00539816302424247  
  y: 0.031808476635306226  
  z: 0.016064612253162254  
angular_velocity_covariance: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]  
linear_acceleration:  
  x: -10.174258537655529  
  y: -0.09624501204002447  
  z: -0.04605802355258433  
linear_acceleration_covariance: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
```

## 5. Chay Encoder

```
ocity = -0.000 rad/s
[INFO] [1743525904.919878, 42.567000]: left_front_omni: Position = 1.217 rad, Ve
locity = -0.000 rad/s
[INFO] [1743525904.921979, 42.567000]: right_back_omni: Position = 3.160 rad, Ve
locity = 0.000 rad/s
[INFO] [1743525904.926112, 42.567000]: right_front_omni: Position = 3.189 rad, V
elocity = 0.000 rad/s
[INFO] [1743525905.089654, 42.587000]: left_back_omni: Position = 1.560 rad, Vel
ocity = 0.041 rad/s
[INFO] [1743525905.092773, 42.587000]: left_front_omni: Position = 1.217 rad, Ve
locity = -0.000 rad/s
[INFO] [1743525905.095996, 42.587000]: right_back_omni: Position = 3.160 rad, Ve
locity = 0.000 rad/s
[INFO] [1743525905.098586, 42.587000]: right_front_omni: Position = 3.189 rad, V
elocity = 0.000 rad/s
[INFO] [1743525905.353378, 42.613000]: left_back_omni: Position = 1.560 rad, Vel
ocity = 0.009 rad/s
[INFO] [1743525905.355218, 42.613000]: left_front_omni: Position = 1.217 rad, Ve
locity = -0.000 rad/s
[INFO] [1743525905.359327, 42.613000]: right_back_omni: Position = 3.160 rad, Ve
locity = 0.000 rad/s
[INFO] [1743525905.362890, 42.614000]: right_front_omni: Position = 3.189 rad, V
elocity = -0.007 rad/s
```