BÁO CÁO GIỮA KỲ ROS

Họ và tên: Lâm Việt Anh
Mã sinh viên: 22027553
Mã môn học: RBE3017

I. Giới thiệu

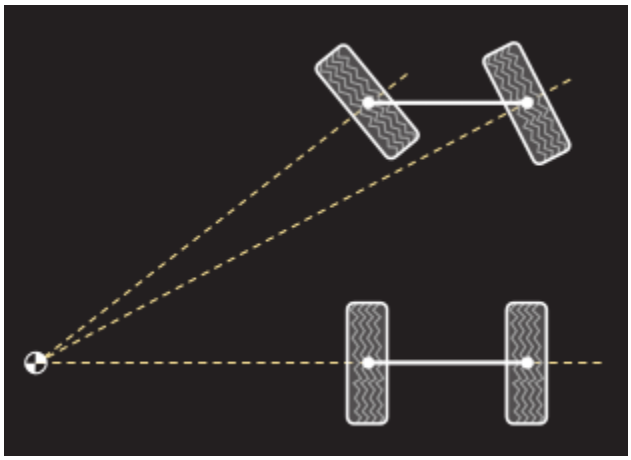
1. Đề tài

Đề tài giữa kỳ được lựa chọn là điều khiển một robot AGV bằng ROS kèm theo các cảm biến định vị. Robot di chuyển bằng bốn bánh theo thiết kế Ackermann Steering. Tay máy hai khớp được thiết kế với khớp đầu là khớp quay, khớp thứ hai là khớp tịnh tiến. Ba cảm biến được lựa chọn gồm có Lidar, IMU và Encoder.

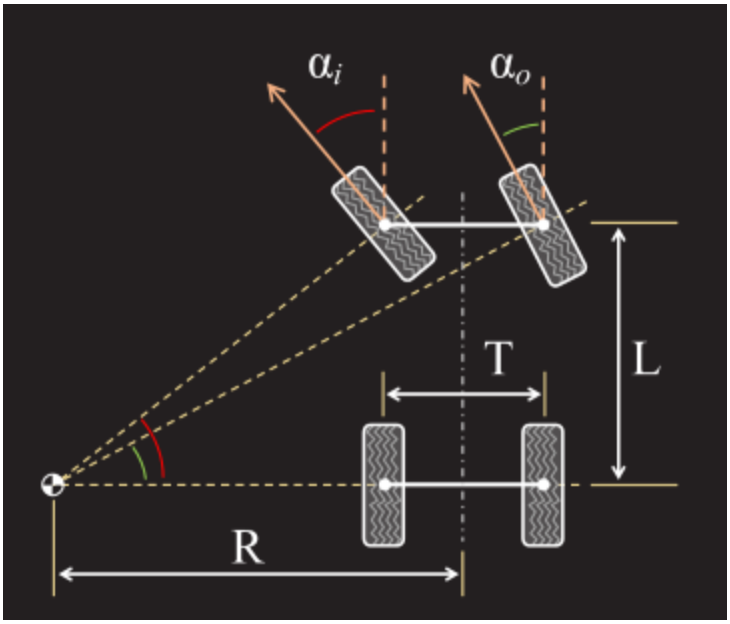
2. Ackermann Steering

Ackermann Steering là một nguyên lý điều khiển dựa trên kết nối của các link trong xe. Mục đích của Ackermann Steering là để cho các bánh dẫn đường của xe có bán kính quay khác nhau khi vào dẫn cua nhằm giảm hiện tượng trượt ngang và tăng độ ổn định. Ứng dụng phổ biến nhất của Ackermann Steering là dùng trong xe đua..

Hai bánh trước dẫn đường của Ackermann Steering khi di chuyển đều quay chung quanh một tâm. Hướng quay và bán kính quay của hai bánh là khác nhau.



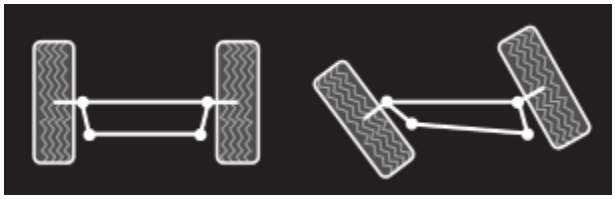
Trong điều khiển lý tưởng bánh trong gần tâm quay nhất sẽ có góc quay lớn hơn bánh ngoài. Đặt trục như hình vẽ.

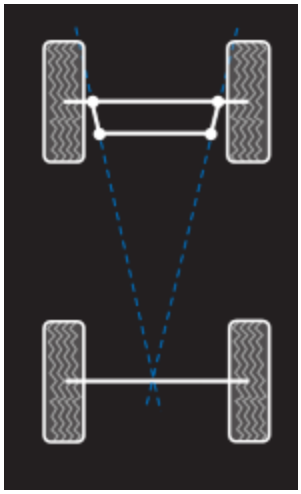


Gọi L là khoảng cách giữa trục trước và trục sau, T là khoảng cách giữa hai bánh trong cùng trục, R là bán kính quay của xe lấy đến trung điểm của trục bánh sau ta có được công thức tính góc quay của từng bánh trước.

$$\propto_i = \tan^{-1}\left(\frac{L}{R - {}^{T}/_{2}}\right) \qquad \propto_o = \tan^{-1}\left(\frac{L}{R + {}^{T}/_{2}}\right)$$

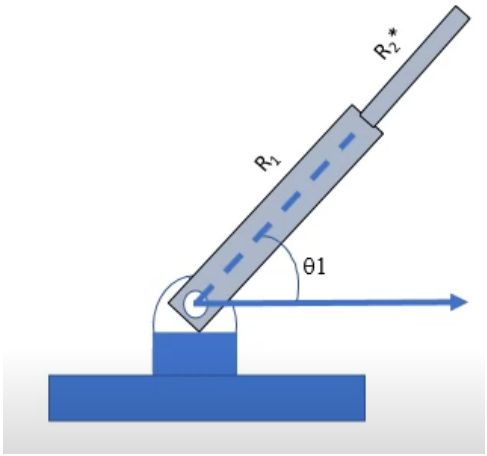
Để điều khiển cho hai bánh dẫn đường quay được với góc quay khác nhau, hai bánh được nối với nhau bằng một chuỗi động học hình thang cân. Hai cạnh bên của hình thang nếu kéo dài ra sẽ gặp nhau tại trung điểm trục bánh xe sau.



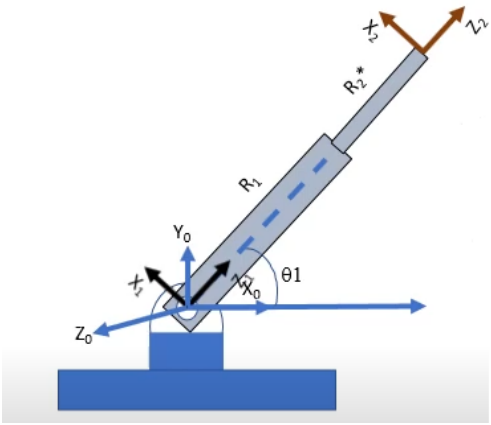


3. Tay máy RP

Tay máy RP hay tay máy Rotation - Prismatic là tay máy hai khớp với khớp đầu là khớp xoay và khớp thứ hai là khớp tịnh tiến.



Đặt các trục cho tay máy như hình vẽ ta có thể xác định được bảng D-H cho tay máy như sau.



Transformation	ai	αi	di	θι
0-1	0	90	0	90+θ1
1-2	0	0	R1+R2*	0

Dựa trên bảng D-H đã xác định ta tính được các ma trận chuyển đổi đơn vị cho từng khớp và ma trận chuyển đổi đồng nhất cho cả tay máy.

$$A_0^1=\begin{bmatrix} -S1 & 0 & C1 & 0 \\ C1 & 0 & S1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad A_1^2=\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & R1+R2 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$A_0^3= A_0^1 A_1^2 = \begin{bmatrix} -S1 & 0 & C1 & (R1+R2)C1 \\ C1 & 0 & S1 & (R1+R2)S1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Kết quả động học thuận và động học ngược cuối cùng của tay máy sẽ là.

$$X = (R1+R2)C1$$
$$Y = (R1+R2)S1.$$

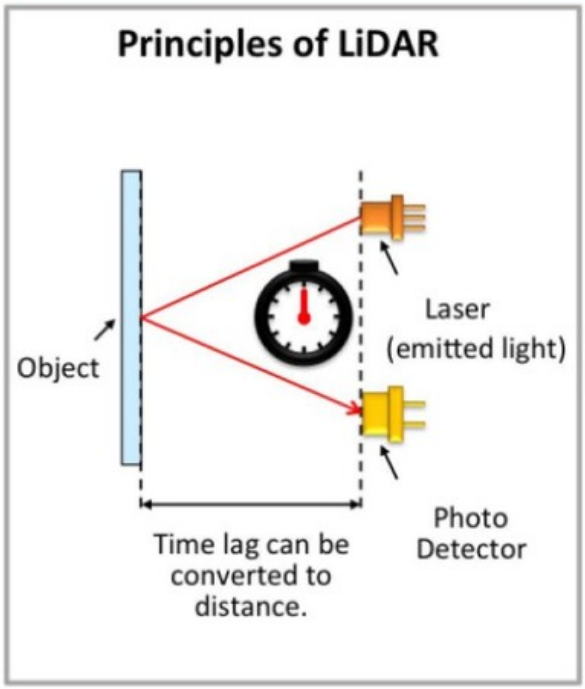
$$R2 = SQRT(X^2 + Y^2)-R1$$

$$\theta 1 = atan2(Y,X)$$

4. Cảm biến Lidar

Cảm biến LiDAR - Light Detection And Ranging là công nghệ sử dụng ánh sáng tia laser đo khoảng cách và xây dựng bản đồ 3D của vật thể. Bằng cách phát ra chùm tia laser có công suất thấp tới môi trường và nhận về rồi tiếp nhận ánh sáng phản chiếu ngược lại phần cứng để xử lý.

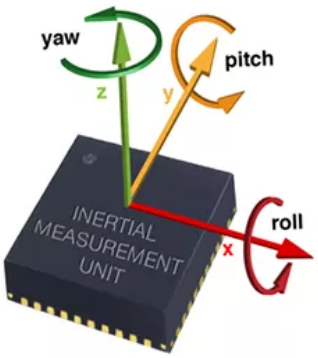
Cảm biến LiDAR hoạt động bằng cách phát ra tia laser và đo thời gian mà tia này phản xạ lại từ các vật thể xung quanh để xác định khoảng cách và hình dạng của chúng. Bằng cách quét laser từ nhiều góc độ khác nhau, LiDAR có thể tạo ra một hình ảnh 3D chính xác của môi trường trong thời gian thực.



5. Cảm biến IMU

Cảm biến IMU - Inertial Measurement Unit là sự kết hợp giữa cảm biến Gyroscope còn gọi là cảm biến con quay hồi chuyển và cảm biến Accelerometer còn gọi là cảm biến gia tốc. Cảm biến có chức năng là cung cấp thông tin về độ nghiêng và góc quay.

Cảm biến gia tốc đo gia tốc tuyến tính theo ba trục x, y, z, giúp phát hiện các lực tác động lên vật thể như gia tốc trọng trường và các chuyển động theo quán tính. Trong khi đó, con quay hồi chuyển đo tốc độ góc quay quanh ba trục, cho phép xác định sự xoay của vật thể. Khi kết hợp lại, hai cảm biến này tạo ra một hệ thống mạnh mẽ giúp theo dõi và phân tích các chuyển động phức tạp.

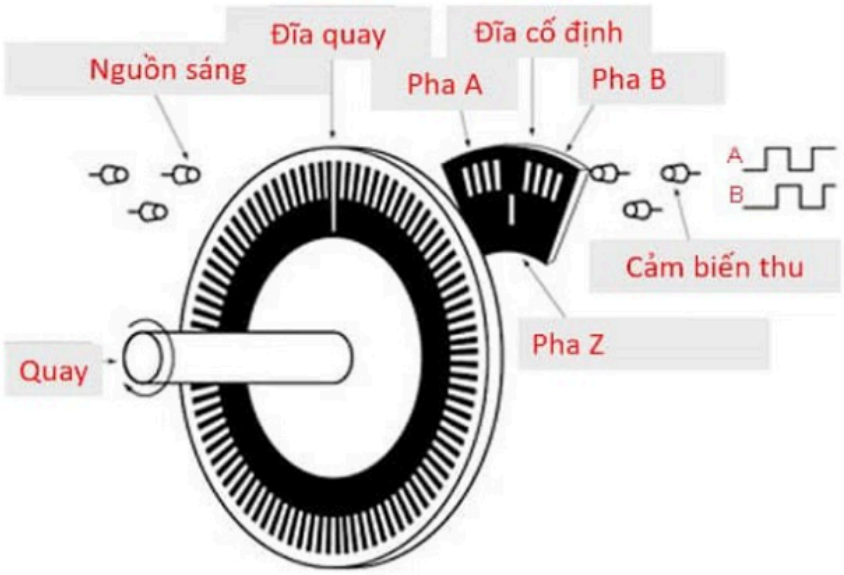


6. Cảm biến Encoder

Cảm biến encoder là thiết bị điện tử dùng để đo lường vị trí, tốc độ và hướng chuyển động của trục quay hoặc hệ thống cơ khí. Encoder hoạt động dựa trên nguyên lý phát hiện sự thay đổi vị trí góc thông qua các tín hiệu xung, từ đó chuyển đổi chuyển động cơ học thành tín hiệu điện tử để hệ thống điều khiển có thể xử lý. Có hai loại encoder phổ biến: encoder tuyệt đối (absolute encoder) xác định vị trí chính xác ngay cả khi mất điện và encoder tương đối (incremental encoder) chỉ đo sự thay đổi vị trí dựa trên số xung đếm được.

Encoder hoạt động theo nguyên lý đĩa quay quanh trục. Trên đĩa mã hóa có các rãnh nhỏ để nguồn phát sáng chiếu tín hiệu quang qua đĩa. Chỗ có rãnh thì ánh sáng xuyên qua được, chỗ không có rãnh ánh sáng không xuyên qua được.

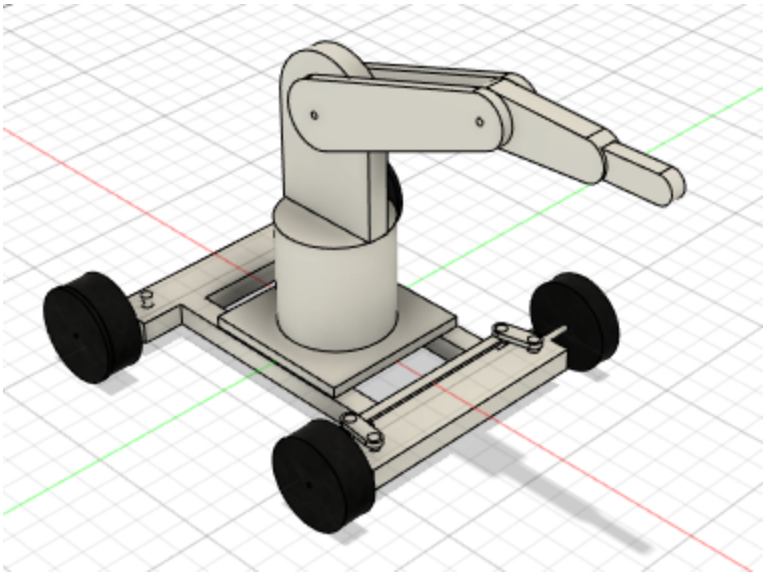
Với các tín hiệu có, hoặc không có ánh sáng chiếu qua, người ta ghi nhận được đèn led có chiếu qua lỗ hay không. Số xung đếm được và tăng lên được tính bằng số lần ánh sáng bị cắt.



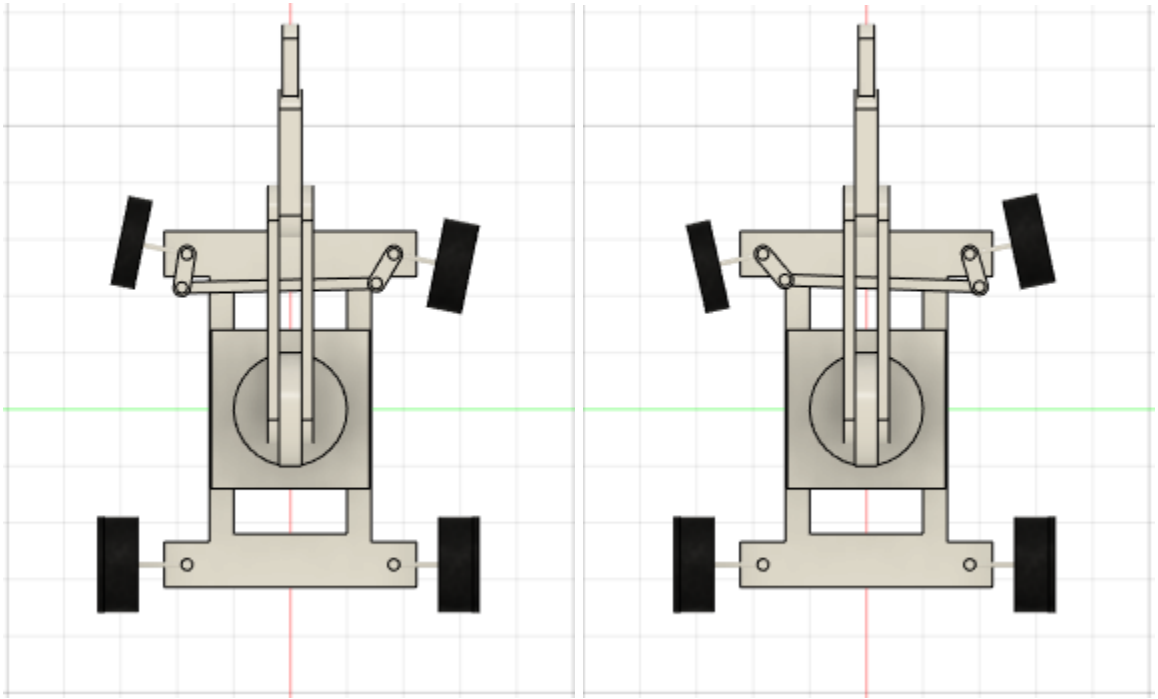
II. Thực hiện

1. Thiết kế mô hình 3D

Bước đầu tiên trong việc xây dựng robot AGV là việc thiết kế mô hình 3D của cả xe và tay máy hai khớp gắn trên xe. Xe sẽ được chọn là xe bốn bánh, hai bánh dẫn đường hai bánh trượt theo. Hai bánh dẫn đường khớp được nối với nhau bằng chuỗi khớp động hình thang cân. Tay máy hai khâu phẳng được đặt trên một bệ cao. Dưới đây là hình ảnh thiết kế 3D của robot.



Hai bánh dẫn đường sử dụng chuỗi khâu hình thang dùng để thực hiện quay đồng thời hai bánh theo chiều trái hoặc phải



Mô hình robot được đặt trục trong Solidwork để xuất ra file URDF. Gốc tọa độ của các hệ tọa độ con được đặt ở khớp nối của các link với nhau. Các bánh xe và khớp quay được đặt sao cho trục z cùng chiều với hướng vecto quay. Riêng đối với khớp tịnh tiến thì trục z được đặt cùng chiều với chiều tịnh tiến.

2. Thiết kế URDF
    a. URDF của thiết kế 3D

File URDF được bắt đầu xây dựng từ mô hình robot 3D đã thiết kế. URDF gồm có các link chính sau: Base_link gồm thân xe và các bộ phận cố định nối với bốn bánh xe (motor_1_link, motor_2_link, motor_3_link, motor_4_link) và tay máy (arm_1_link và arm_2_link). Ngoài ra còn có một dummy_link là parent của base_link dùng để sửa lỗi biến đổi động học trong Gazebo do Gazebo định nghĩa base_link không được có momen quán tính.

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

Các khâu trong file URDF đều được khai báo theo một cấu trúc. Mỗi khâu được đặt một tên riêng để phân biệt. Thông số quán tính của từng khâu gồm có trọng tâm khâu theo tọa độ xyz và hướng roll-pitch-yaw, khối lượng tổng thể của khâu và momen quán tính theo 9 trục trong không gian. Để hiển thị được khâu là hai thành phần visual và collision. Visual là để xác định hình dạng hiển thị của khâu, collision là xác định hình dạng va chạm dùng cho tính toán vật lý, mô phỏng. Cả visual và collision đều chung gốc tốc độ đặt vật và chỉ chung về đường dẫn file STL của link đó.

```
<link
   name="base_link">
   <inertial>
     <origin
       xyz="0.00959630430752534 -4.59753131624718E-05 0.0914512861464635"
       rpy="0 0 0" />
     <mass
       value="2.09964115614805" />
     <inertia
       ixx="0.019201784582518"
       ixy="-4.29296488428788E-07"
       ixz="-0.00346486881609445"
       iyy="0.0272604310157993"
       iyz="-6.56054217370797E-13"
       izz="0.0142151743467295" />
   </inertial>
   <visual>
     <origin
       xyz="0 0 0"
       rpy="0 0 0" />
```

```
    <geometry>
      <mesh
        filename="package://ros_lva/meshes/base_link.STL" />
    </geometry>
    <material
      name="">
      <color
        rgba="0.964705882352941 0.964705882352941 0.952941176470588 1" />
    </material>
  </visual>
  <collision>
    <origin
      xyz="0 0 0"
      rpy="0 0 0" />
    <geometry>
      <mesh
        filename="package://ros_lva/meshes/base_link.STL" />
    </geometry>
  </collision>
</link>
```

Các khâu được nối với base_link bằng các khớp, mỗi khớp được định nghĩa cho từng khâu (motor_1_joint, motor_2_joint, motor_3_joint, motor_4_joint, arm_1_joint và arm_2_joint). Bốn khớp quay cho bánh xe là bốn khớp quay không giới hạn (continuous joint). Khớp đầu tiên của tay máy thì là khớp quay có giới hạn (revolute joint) còn khớp thứ hai thì là khớp tịnh tiến (prismatic joint). Giới hạn của từng khớp đều được nêu trong file URDF.

Các khớp được định nghĩa chung trong URDF với các thành phần: tên riêng biệt của từng khớp, khâu cha và khâu con mà khớp đó nối với nhau, kiểu khớp và tọa độ đặt khớp đó.

```
<joint
  name="arm_1_joint"
  type="revolute">
  <origin
    xyz="0.175 0.02 0.295"
    rpy="-1.5707963267949 0 0" />
  <parent
    link="base_link" />
  <child
    link="arm_1_link" />
  <axis
```

```
    xyz="0 0 1" />
  <limit lower="-3.14159" upper="3.14159" effort="100" velocity="10" />
  <initial_position>0.0</initial_position>
  <dynamics damping="0.1" friction="0.5"/>
</joint>
```

b. Plugin của URDF

Để điều khiển được xe và tay máy file URDF phải có Plugin điều khiển "gazebo-ros-control". Đối với mỗi khớp của tay máy, điều khiển dựa theo vị trí là hợp lý nhất nhằm đảm bảo chuyển động chính xác. Thế nên mỗi khớp trong tay máy còn có một plugin "Transmission" với phân loại điều khiển phần cứng là "PositionJointInterfacel".

```
<gazebo>
 <plugin name="gazebo_ros_control" filename="libgazebo_ros_control.so"/>
 </gazebo>
```

```
<transmission name="arm_1_trans">
 <type>transmission_interface/SimpleTransmission</type>
 <joint name="arm_1_joint">

<hardwareInterface>hardware_interface/PositionJointInterface</hardwareInte
rface>
 </joint>
 <actuator name="arm_1_motor">
   <mechanicalReduction>1</mechanicalReduction>
 </actuator>
 </transmission>
```

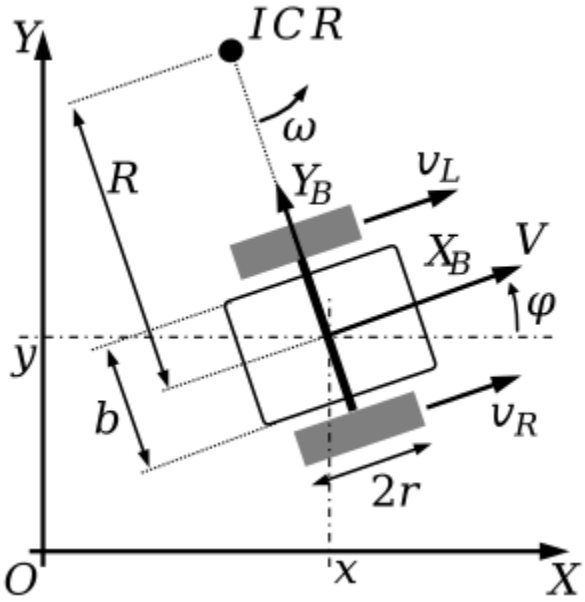
Còn đối với bốn bánh xe điều khiển đồng thời với nhau. Bốn bánh xe cùng dùng chung nhau plugin điều khiển kiểu bánh vi sai. Lý do lựa chọn plugin điều khiển vi sai là vì khó khăn thực hiện điều khiển bốn bánh ban đầu bằng phương pháp Ackermann Steering.

```
<gazebo>
 <plugin name="differential_drive_controller_1"
filename="libgazebo_ros_diff_drive.so">
   <leftJoint>motor_4_joint</leftJoint>
   <rightJoint>motor_3_joint</rightJoint>

   <commandTopic>cmd_vel</commandTopic>
```

```
   <odometryTopic>odom</odometryTopic>
   <odometryFrame>odom</odometryFrame>
   <odometrySource>world</odometrySource>
   <publishOdomTF>true</publishOdomTF>
   <publishWheelTF>false</publishWheelTF>
   <publishTf>true</publishTf>
   <publishWheelJointState>true</publishWheelJointState>
   <legacyMode>false</legacyMode>
   <updateRate>50</updateRate>
   <wheelSeparation>0.266137</wheelSeparation>
   <wheelDiameter>0.0852</wheelDiameter>
   <robotBaseFrame>base_link</robotBaseFrame>
   <wheelAcceleration>1</wheelAcceleration>
   <wheelTorque>10</wheelTorque>
   <rosDebugLevel>na</rosDebugLevel>
 </plugin>
 </gazebo>
  <gazebo>
 <plugin name="differential_drive_controller_22"
filename="libgazebo_ros_diff_drive.so">
   <leftJoint>motor_2_joint</leftJoint>
   <rightJoint>motor_1_joint</rightJoint>

   <commandTopic>cmd_vel</commandTopic>
   <odometryTopic>odom</odometryTopic>
   <odometryFrame>odom</odometryFrame>
   <odometrySource>world</odometrySource>
   <publishOdomTF>true</publishOdomTF>
   <publishWheelTF>false</publishWheelTF>
   <publishTf>true</publishTf>
   <publishWheelJointState>true</publishWheelJointState>
   <legacyMode>false</legacyMode>
   <updateRate>50</updateRate>
   <wheelSeparation>0.260605</wheelSeparation>
   <wheelDiameter>0.08</wheelDiameter>
   <robotBaseFrame>base_link</robotBaseFrame>
   <wheelAcceleration>1</wheelAcceleration>
   <wheelTorque>10</wheelTorque>
   <rosDebugLevel>na</rosDebugLevel>
 </plugin>
```

```
</gazebo>
```

Điều khiển vi sai là phương pháp di chuyển xe dựa vào sự chênh lệch vận tốc giữa các bánh hai bên. Điều khiển vi sai phổ biến với xe hai bánh do dễ tạo chênh lệch vận tốc. Vì thế trong trường hợp xe bốn bánh, hai bánh bên trái sẽ quay cùng vận tốc, hai bánh bên phải tương tự.



Gọi bán kính bánh xe vi sai là r, chiều rộng bánh xe hai khoảng cách giữa các bánh hai bên là b, vận tốc góc của các bánh là omega. Để điều khiển cho xe di chuyển quanh quỹ đạo của một đường tròn đường kính R vận tốc bánh xe trái và bánh xe phải được tính như sau.

$$\omega \cdot (R + b/2) = v_R$$
$$\omega \cdot (R - b/2) = v_L$$

Biết được vận tốc của bánh xe bên trái và bên phải ta có thể tính ngược lại ra được vận tốc quay của bánh xe và bánh kính của quỹ đạo chuyển động, đồng thời cùng như vận tốc dài của toàn xe.

$$\omega = (v_R - v_L)/b$$
$$R = b/2 \cdot (v_R + v_L)/(v_R - v_L)$$

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

c. URDF các cảm biến

Để thực hiện mô phỏng các cảm biến Lidar và IMU cho xe, các cảm biến phải được tạo khâu tương và khớp tương ứng và gắn cố định trên thân xe. Riêng đối với cảm biến Encoder thì giá trị của cảm biến có thể đọc được bằng cách truy cập vào topic /odom hoặc /joint_states nên không cần phải thêm vào cảm biến trong URDF. Khâu của cảm biến Lidar và cảm biến IMU được khai báo như các khâu khác của robot. Trong đó cảm biến Lidar được đặt trên đầu robot để có quan sát toàn diện, còn cảm biến IMU thì được đặt dưới đáy xe.

Plugin cho cảm biến Lidar gồm các thành phần: tần số cập nhật thông tin, loại cảm biến là loại tia laser, chế độ hiển thị các tia, số lượng tia quét, góc quét, độ phân giải góc, khoảng cách tối thiểu và tối đa mà tia quét có thể phát hiện được vật và cuối cùng gói ros hỗ trợ chạy cảm biến.

```
<gazebo reference="lidar_link">
 <sensor type="ray" name="lidar_sensor">
   <pose>0 0 0 0 0 0</pose>
   <visualize>true</visualize>
   <update_rate>50</update_rate>
   <ray>
     <scan>
       <horizontal>
         <samples>360</samples>
         <resolution>1</resolution>
         <min_angle>-3.14</min_angle>
         <max_angle>3.14</max_angle>
       </horizontal>
     </scan>
     <range>
       <min>0.1</min>
       <max>10.0</max>
       <resolution>0.01</resolution>
     </range>
   </ray>
   <plugin name="lidar_controller" filename="libgazebo_ros_laser.so">
     <topicName>/scan</topicName>
     <frameName>lidar_link</frameName>
   </plugin>
 </sensor>
</gazebo>
```

Plugin cảm biến IMU thì gồm: chế độ bật luôn luôn bật, tần số cập nhật thông tin, độ lệch của cảm biến về các hướng x, y, z cũng như các góc roll-pitch-yaw và gói ros dùng để hỗ trợ cảm biến.

```
<gazebo reference="imu_link">
   <gravity>true</gravity>
   <sensor name="imu_sensor" type="imu">
     <always_on>true</always_on>
     <update_rate>50</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>5.0</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>
```

3. Thiết kế điều khiển

a. Điều khiển xe

Xe được điều khiển bằng tạo publisher với topic /cmd_vel với kiểu message gửi là Twist. Code điều khiển xe được xây dựng dựa bằng gói package “teleop_twist_keyboard” nên cũng giống như gói package điều khiển xe có thể thực hiện được bằng 9 phím. Trong đó “w” và “s” là di chuyển tiến/lùi, “a” và “d” là quay trái/phải tại chỗ. Các phím còn lại là các tổ hợp của vừa đi tiến/lùi vừa quay trái/phải. Riêng phím “x” thì là phím dừng lại. Ngoài ra còn có các tổ hợp phím 1/2/3/4/5/6 dùng để tăng, giảm vận tốc.

```
msg = """
Reading from the keyboard  and Publishing to Twist!
---------------------------
Moving around:
  q    w    e
```

```
  a    s    d
  z    x    c

anything else : stop

1/2 : increase/decrease max speeds by 10%
3/4 : increase/decrease only linear speed by 10%
5/6 : increase/decrease only angular speed by 10%

CTRL-C to quit
"""
```

Luồng tin nhắn mà code gửi để điều khiển là một luồn được tạo riêng và tiên được gửi liên tục tới ROS. Các tham số đầu vào mà code quan tâm đến gồm có tọa độ x, y, z, góc quay th và tốc độ di chuyển tịnh tiến speed cũng như tốc độ quay turn.

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

       # Set timeout to None if rate is 0 (causes new_message to wait
forever
       # for new data to publish)
       if rate != 0.0:
           self.timeout = 1.0 / rate
       else:
           self.timeout = None

       self.start()
```

Các giá trị này có thể được cập nhật liên tục nhờ vào phím ấn. Sau khi cập nhật được giá trị, thì kết quả cuối cùng sẽ được cập nhật vào message Twist là độ dời theo các tọa độ x, y, z sẽ bằng tọa độ hiện tại

nhân với tốc độ di chuyển tịnh tiến. Còn góc quay cuối sẽ bằng góc quay hiện tại nhân với tốc độ quay. Message này sau đó sẽ được gửi vào topic để điều khiển xe di chuyển kiểu vi sai.

```
def run(self):
        twist = Twist()
        while not self.done:
            self.condition.acquire()
            self.condition.wait(self.timeout)
            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()

            self.publisher.publish(twist)

        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)
```

Ngoài ra bên cạnh đó code cũng có các chương trình khác để lấy tín hiệu đọc được từ bàn phím cũng như là chương trình để cập nhật thông tin mỗi khi bàn phím được ấn.

b. Điều khiển tay máy

Tay máy cũng sẽ được điều khiển bằng bàn phím giống như xe. Nhưng thay vì là chỉ cần truyền lên một topic chung là điều khiển được bốn bánh, tay máy sẽ do mỗi khớp hoạt động độc lập nên từng khớp sẽ có một topic riêng để gửi.

```
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
   print("u: Tăng vị trí arm_1_joint")
   print("i: Giảm vị trí arm_1_joint")
   print("o: Tăng vị trí arm_2_joint")
   print("p: Giảm vị trí arm_2_joint")
   print("y: Thoát")
   rate = rospy.Rate(50)
```

Quá trình điều khiển tay máy tương đối đơn giản. Do sử dụng "PositionJointInterface" nên mỗi khi nhấn phím thích hợp vị trí của từng khâu sẽ tăng thêm theo một khoảng đã định sẵn.

```
while not rospy.is_shutdown():
       key = get_key()
       if key == 'u':
           arm_1_pos += 0.5
           if arm_1_pos > 3.14:
               arm_1_pos = 3.14
       elif key == 'i':
           arm_1_pos -= 0.5
           if arm_1_pos < -3.14:
               arm_1_pos = -3.14
       elif key == 'o':
           arm_2_pos += 0.01
           if arm_2_pos > 0.07:
               arm_2_pos = 0.07
       elif key == 'p':
           arm_2_pos -= 0.01
           if arm_2_pos < 0.0:
               arm_2_pos = 0.0
       elif key == 'y':
           break
       arm_1_pub.publish(arm_1_pos)
       arm_2_pub.publish(arm_2_pos)
       rospy.loginfo("Arm 1 position: %.2f rad, Arm 2 position: %.2f m",
arm_1_pos, arm_2_pos)
       rate.sleep()
```

Tương tự như code điều khiển xe, code điều khiển tay máy cũng có hàm hỗ trợ đọc đầu vào từ bàn phím.

c. Đọc giá trị Lidar

Code đọc giá trị từ Lidar sử dụng kiểu message của sẵn của sensor_msgs đó là LaserScan. Sau đó thông qua một hàm callback đơn giản Lidar có thể đọc được số lượng điểm phát ra từ cảm biến Lidar.

```
#!/usr/bin/env python3
import rospy
from sensor_msgs.msg import LaserScan

def lidar_callback(data):
    rospy.loginfo("Nhận dữ liệu LiDAR với %d điểm", len(data.ranges))

if __name__ == '__main__':
    rospy.init_node('lidar_processor')
    rospy.Subscriber("/scan", LaserScan, lidar_callback)
    rospy.spin()
```

d. Đọc giá trị IMU

Tương tự như code đọc giá trị từ Lidar, code đọc IMU cũng dùng một hàm callback đơn giản để lấy trực tiếp dữ liệu các hướng từ topic /imu/data.

```
#!/usr/bin/env python3
import rospy
from sensor_msgs.msg import Imu

def imu_callback(data):
    orientation = data.orientation
    angular_vel = data.angular_velocity
    linear_accel = data.linear_acceleration

    rospy.loginfo("IMU Data - Orientation: %s", str(orientation))

if __name__ == '__main__':
    rospy.init_node('imu_processor')
    rospy.Subscriber("/imu/data", Imu, imu_callback)
    rospy.spin()
```

e. Đọc giá trị encoder

Code đọc encoder đọc trực tiếp giá trị về vị trí và vận tốc của một motor, ở dưới đây là motor_1_joint trực tiếp thông qua topic /joint_states

```
def joint_state_callback(msg):
   joints_to_display = ['[motor_1_joint', 'motor_2_joint',
'motor_3_joint', 'motor_4_joint']
   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_processor', anonymous=True)
   rospy.Subscriber('/joint_states', JointState, joint_state_callback)
   rospy.spin()
```

f. Cấu hình control.yaml

Sau khi xây dựng xong được các code điều khiển thì để điều khiển được thành công cả xe và tay máy chạy, thay đổi các giá trị cảm biến thì ta cần phải xây dựng các controller để điều khiển các khớp. Có năm controller cần phải xây dựng. Control_manager điều khiển tất cả các controller khác. Joint states controller dùng để xuất bản các thông tin về vị trí, vận tốc của các khớp lên topic /joint_states. Arm_1_controller và arm_2_controller dùng để điều khiển hai khớp của tay máy. Và cuối cùng diff_drive_controller dùng để điều khiển xe chạy vi sai.

```
controller_manager:
 ros:
   auto_start: true
   update_rate: 50

joint_state_controller:
 type: joint_state_controller/JointStateController
 publish_rate: 50
 joints:
   - motor_1_joint
   - motor_2_joint
```

```
   - motor_3_joint
   - motor_4_joint
   - arm_1_joint
   - arm_2_joint

diff_drive_controller_1:
 type: "diff_drive_controller_1/DiffDriveController"
 left_wheel: 'motor_4_joint'
 right_wheel: 'motor_3_joint'
 wheel_separation: 0.266137
 wheel_radius: 0.0426
 cmd_vel_topic: cmd_vel
 odom_topic: odom
 publish_rate: 50
 base_frame_id: base_link
 odom_frame_id: odom
 enable_odom_tf: true

diff_drive_controller_2:
 type: "diff_drive_controller_2/DiffDriveController"
 left_wheel: 'motor_2_joint'
 right_wheel: 'motor_1_joint'
 wheel_separation: 0.260605
 wheel_radius: 0.04
 cmd_vel_topic: cmd_vel
 odom_topic: odom
 publish_rate: 50
 base_frame_id: base_link
 odom_frame_id: odom
 enable_odom_tf: true

arm_1_controller:
 type: position_controllers/JointPositionController
 joint: arm_1_joint

arm_2_controller:
 type: position_controllers/JointPositionController
 joint: arm_2_joint
 gazebo_ros_control:
 pid_gains:
```

```
  motor_1_joint: {p: 100.0, i: 10.0, d: 1.0}
  motor_2_joint: {p: 100.0, i: 10.0, d: 1.0}
  motor_3_joint: {p: 100.0, i: 10.0, d: 1.0}
  motor_4_joint: {p: 100.0, i: 10.0, d: 1.0}
  arm_1_joint: {p: 1.0, i: 0.1, d: 0.01}
  arm_2_joint: {p: 1.0, i: 0.1, d: 0.01}
```

g. Cấu hình khởi động trong Gazebo

Để chạy robot trong Gazebo, yêu cầu phải tạo một file launch trong đó yêu cầu định sẵn các tham số sẵn của Gazebo, khởi tạo thành công một thế giới trống sau đó spawn được robot của mình vào thế giới đó. Ngoài ra khởi tạo đồng thời cùng với robot là các node điều khiển và đọc cảm biến cần sử dụng.

```
<!-- Tải các bộ điều khiển -->
 <rosparam file="$(find ros_lva)/config/control.yaml" command="load"/>

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

 <!-- Node teleop -->
 <node name="teleop_wasd_keyboard" pkg="ros_lva" type="teleop_wasd.py"
output="screen">
   <remap from="cmd_vel" to="/cmd_vel"/>
 </node>

 <!-- Node điều khiển cánh tay -->
 <node name="arm_teleop_keyboard" pkg="ros_lva" type="arm_teleop.py"
output="screen" if="$(arg enable_keyboard)"/>
  <!-- Robot State Publisher -->
 <node name="robot_state_publisher" pkg="robot_state_publisher"
       type="robot_state_publisher" respawn="false" output="screen"/>

 <!-- Các node xử lý cảm biến -->
 <node name="lidar_processor" pkg="ros_lva" type="lidar_processor.py"
output="screen">
```

```
    <remap from="scan" to="/scan"/>
  </node>

  <node name="imu_processor" pkg="ros_lva" type="imu_processor.py"
output="screen">
    <remap from="imu_data" to="/imu/data"/>
  </node>
   <node name="encoder_processor" pkg="ros_lva" type="encoder_processor.py"
output="screen">
    <remap from="encoder_1" to="/joint_states"/>
  </node>
```

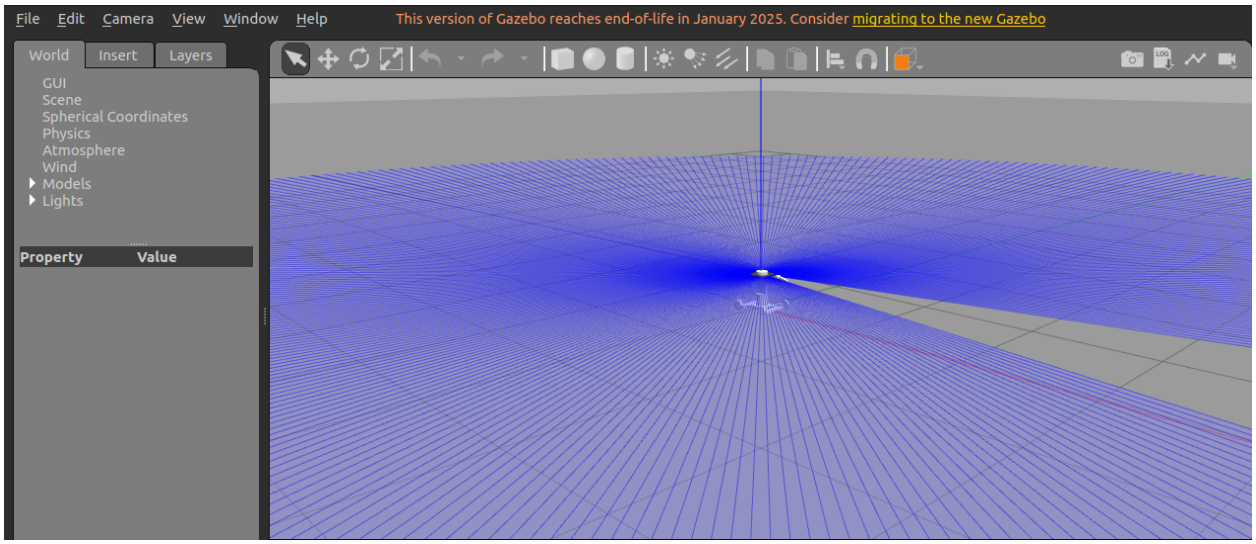
h. Cấu hình khởi động trong Rviz

Khởi tạo trong Rviz cũng giống như trong Gazebo là phải khởi tạo được thành công thế giới rồi cho thêm robot thiết kế vào. Nhưng Rviz khác với Gazebo ở hai chỗ là Rviz hỗ trợ thêm khởi tạo Joint GUI dùng để điều khiển trực tiếp các khớp bằng thanh trượt và các topic cảm biến có thể được add thủ công vào Rviz. Các topic này add vào xong sẽ được lưu lại vào trong file rviz config cho cả những lần sau mở ra.

```
<node
    name="joint_state_publisher_gui"
    pkg="joint_state_publisher_gui"
    type="joint_state_publisher_gui" />
```

III. Kết quả

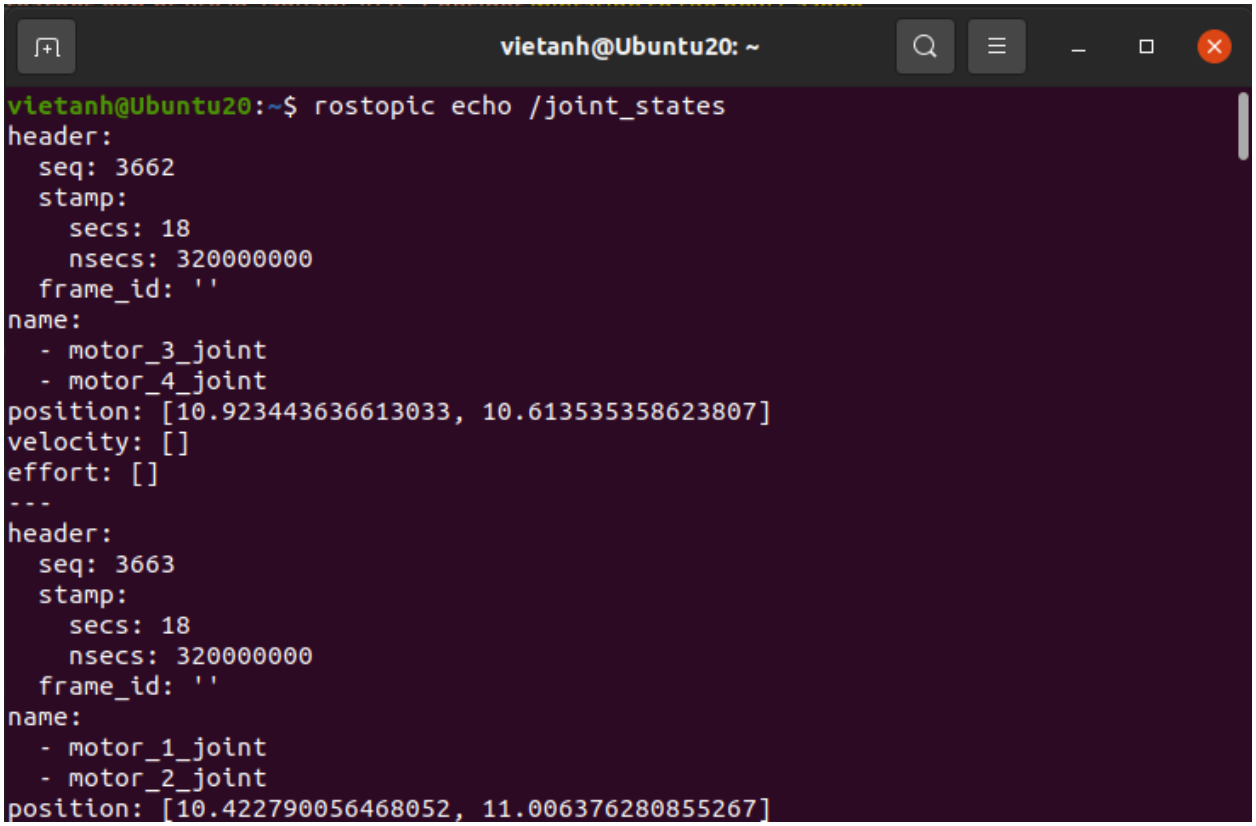
1. Hiển thị và chạy trong Gazebo
    a. Hiển thị robot trong Gazebo

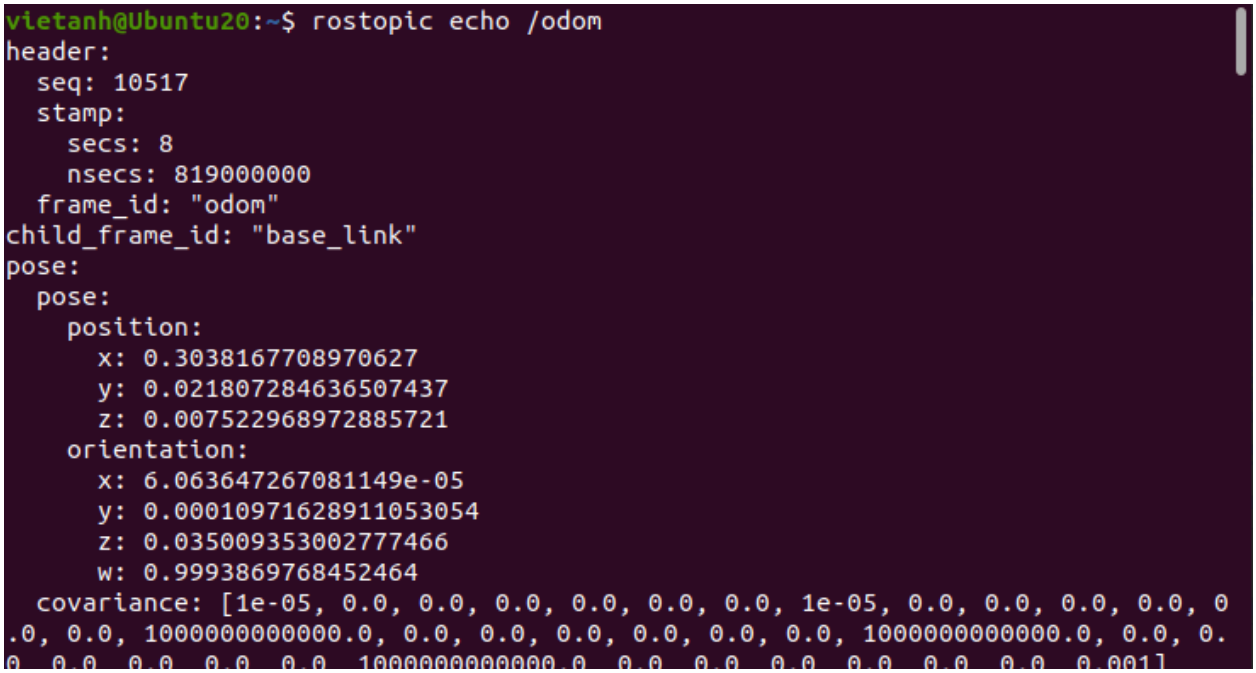


b. Đọc các giá trị cảm biến

```
vietanh@Ubuntu20:~$ rostopic echo /scan
header:
  seq: 2299
  stamp:
    secs: 46
    nsecs: 217000000
  frame_id: "lidar_link"
angle_min: -3.140000104904175
angle_max: 3.140000104904175
angle_increment: 0.01749303564429283
time_increment: 0.0
scan_time: 0.0
range_min: 0.10000000149011612
range_max: 10.0
ranges: [inf, inf, inf, inf, inf, inf, inf, inf, inf, inf, inf, inf, inf, inf, i
nf, inf, inf, inf, inf, inf, inf, inf, inf, inf, inf, inf, inf, inf, inf, inf, i
```

```
vietanh@Ubuntu20:~$ rostopic echo /imu/data
WARNING: no messages received and simulated time is active.
Is /clock being published?
header:
  seq: 61
  stamp:
    secs: 62
    nsecs: 268000000
  frame_id: "imu_link"
orientation:
  x: -0.0001431043713332573
  y: 0.0001297381127640361
  z: 0.02717925377568774
  w: 0.9996305571826812
orientation_covariance: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
angular_velocity:
  x: -0.0019556885686176115
  y: 0.00034046223961917095
  z: -0.0010890696865891128
angular_velocity_covariance: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
linear_acceleration:
```


vietanh@Ubuntu20: ~
vietanh@Ubuntu20:~$ rostopic echo /joint_states
header:
  seq: 3662
  stamp:
    secs: 18
    nsecs: 320000000
  frame_id: ''
name:
  - motor_3_joint
  - motor_4_joint
position: [10.923443636613033, 10.613535358623807]
velocity: []
effort: []
---
header:
  seq: 3663
  stamp:
    secs: 18
    nsecs: 320000000
  frame_id: ''
name:
  - motor_1_joint
  - motor_2_joint
position: [10.422790056468052, 11.006376280855267]


vietanh@Ubuntu20:~$ rostopic echo /odom
header:
  seq: 10517
  stamp:
    secs: 8
    nsecs: 819000000
  frame_id: "odom"
child_frame_id: "base_link"
pose:
  pose:
    position:
      x: 0.3038167708970627
      y: 0.021807284636507437
      z: 0.007522968972885721
    orientation:
      x: 6.063647267081149e-05
      y: 0.00010971628911053054
      z: 0.035009353002777466
      w: 0.9993869768452464
  covariance: [1e-05, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1e-05, 0.0, 0.0, 0.0, 0.0, 0
.0, 0.0, 1000000000000.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1000000000000.0, 0.0, 0.

2. Đọc và hiển thị trong Rviz