

Đại học Quốc gia Hà Nội
Trường Đại học Công nghệ

Khoa Điện Tử Viễn Thông



BÁO CÁO GIỮA KỲ MÔN LẬP TRÌNH ROBOT VỚI ROS

Đề tài : DIFFERENTIAL ROBOT

Họ và tên : Nguyễn Thanh Đức
Mã số sinh viên : 22027544

MỤC LỤC

1.TỔNG QUAN VỀ DỰ ÁN

- 1.1 Mục tiêu dự án
- 1.2 Các công cụ thực hiện trong dự án

2.THIẾT KẾ

- 2.1. Cảm biến
- 2.2. Thiết kế 3D
- 2.3. Cách gán hệ trục tọa độ cho các bộ phận
- 2.4. Xuất file URDF từ SolidWorks
- 2.5. Cấu trúc thư mục
- 2.6. Mô phỏng trong Gazebo

3.MÔ HÌNH ĐỘNG HỌC

- 3.1. Động học di chuyển
- 3.2. Điều khiển trong ROS
- 3.3 Các ros nodes và topics

4.TỔNG KẾT

- 4.1. Kết quả
- 4.2. Tài liệu tham khảo

1. TỔNG QUAN VỀ DỰ ÁN

1.1 Mục tiêu dự án

Dự án thiết kế một robot 2 bánh vi sai có khả năng di chuyển linh hoạt trong môi trường giả lập Gazebo. Robot cần được tích hợp các cảm biến như camera, IMU, LiDAR và có hệ thống tay máy 2 khớp xoay để thực hiện các nhiệm vụ mô phỏng thao tác vật thể. Ngoài ra, robot phải có khả năng điều khiển từ xa thông qua bàn phím hoặc giao diện ROS.

1.2. Các công cụ thực hiện trong dự án

1.2.1. Robot Operating System (ROS)

ROS là một framework cung cấp một bộ phần mềm cốt lõi để vận hành robot có thể được mở rộng bằng cách tạo hoặc sử dụng các gói hiện có. Có hàng nghìn gói có sẵn trên mỗi bản phân phối ổn định đóng gói các thuật toán và trình điều khiển cảm biến, trong số những gói khác. Các chương trình ROS có thể được tạo chủ yếu bằng hai ngôn ngữ lập trình: Python và C++. Ưu điểm chính của ROS là nó là một mã nguồn mở và miễn phí, cho phép viết phần mềm robot có thể được sử dụng lại trên các nền tảng phần cứng riêng biệt. Ngoài ra, ROS có sẵn cho nhiều loại robot, chẳng hạn như robot di động, người thao túng, robot hình người, xe tự hành, v.v. ROS cung cấp kiến trúc ngang hàng cho phép các thành phần 'master' hoặc 'slave' đối thoại trực tiếp với nhau, đồng bộ hoặc không đồng bộ. Cuối cùng, ROS rất dễ sử dụng vì các trình điều khiển hoặc thuật toán của nó được chứa trong các tệp thực thi độc lập, giúp giảm kích thước của nó

1.2.2. Gazebo

Gazebo là một công cụ mô phỏng nhiều robot có thể cung cấp mô phỏng chính xác cho robot, cảm biến và vật thể trong không gian 3D. Nó tạo ra phản hồi cảm biến thực tế và có một công cụ vật lý mạnh mẽ để tạo ra các tương tác giữa các vật thể, robot và môi trường. Gazebo cũng được cung cấp miễn phí như một phần mềm độc lập nhưng cũng đã được đóng gói cùng với ROS như một công

cụ mô phỏng . Ưu điểm chính của Gazebo là nó cung cấp đồ họa chất lượng cao và giao diện người dùng đồ họa và lập trình phù hợp.

2. THIẾT KẾ

2.1. Cảm biến

1. 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.

2. Camera

Trong hệ thống robot sử dụng ROS, cảm biến camera đóng vai trò quan trọng trong việc cung cấp thông tin thị giác cho robot. Dữ liệu từ camera có thể được sử dụng cho nhiều mục đích khác nhau, bao gồm:

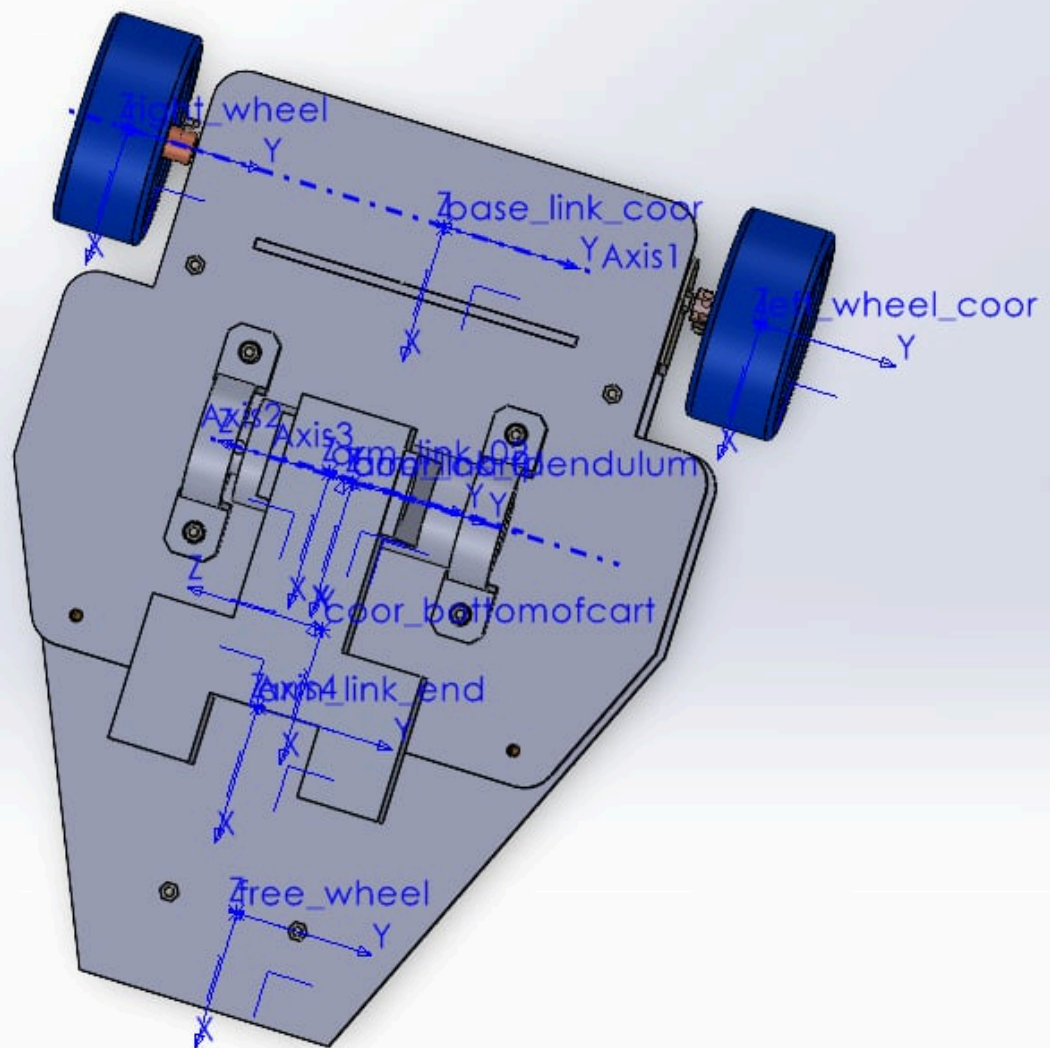
- **Nhận dạng và phân loại đối tượng:** Phát hiện và xác định các vật thể trong môi trường.
- **Theo dõi đối tượng:** Theo dõi chuyển động của các đối tượng quan tâm.
- **Dẫn đường và định vị:** Hỗ trợ robot di chuyển và xác định vị trí của nó trong không gian.
- **Lập bản đồ 3D:** Tạo bản đồ ba chiều của môi trường xung quanh.
- **Tương tác người-robot:** Nhận diện cử chỉ hoặc khuôn mặt người.
- **Kiểm tra và giám sát:** Đảm bảo chất lượng sản phẩm hoặc giám sát một khu vực cụ thể.

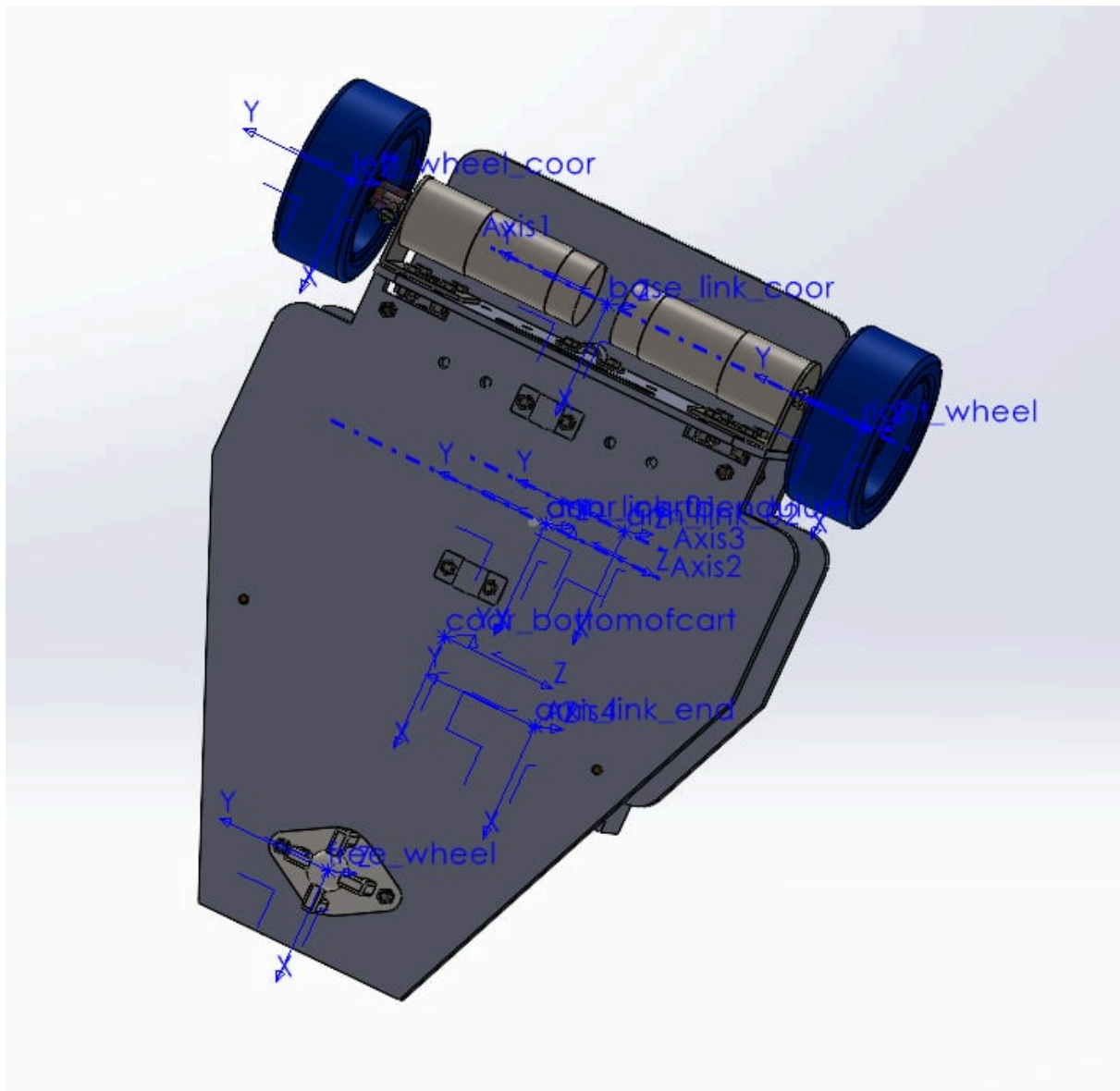
3. IMU

IMU là một thiết bị điện tử đo lường và báo cáo các lực quán tính cụ thể, vận tốc góc và đôi khi cả từ trường xung quanh nó. Trong hệ thống robot sử dụng ROS, IMU đóng vai trò quan trọng trong việc cung cấp thông tin về:

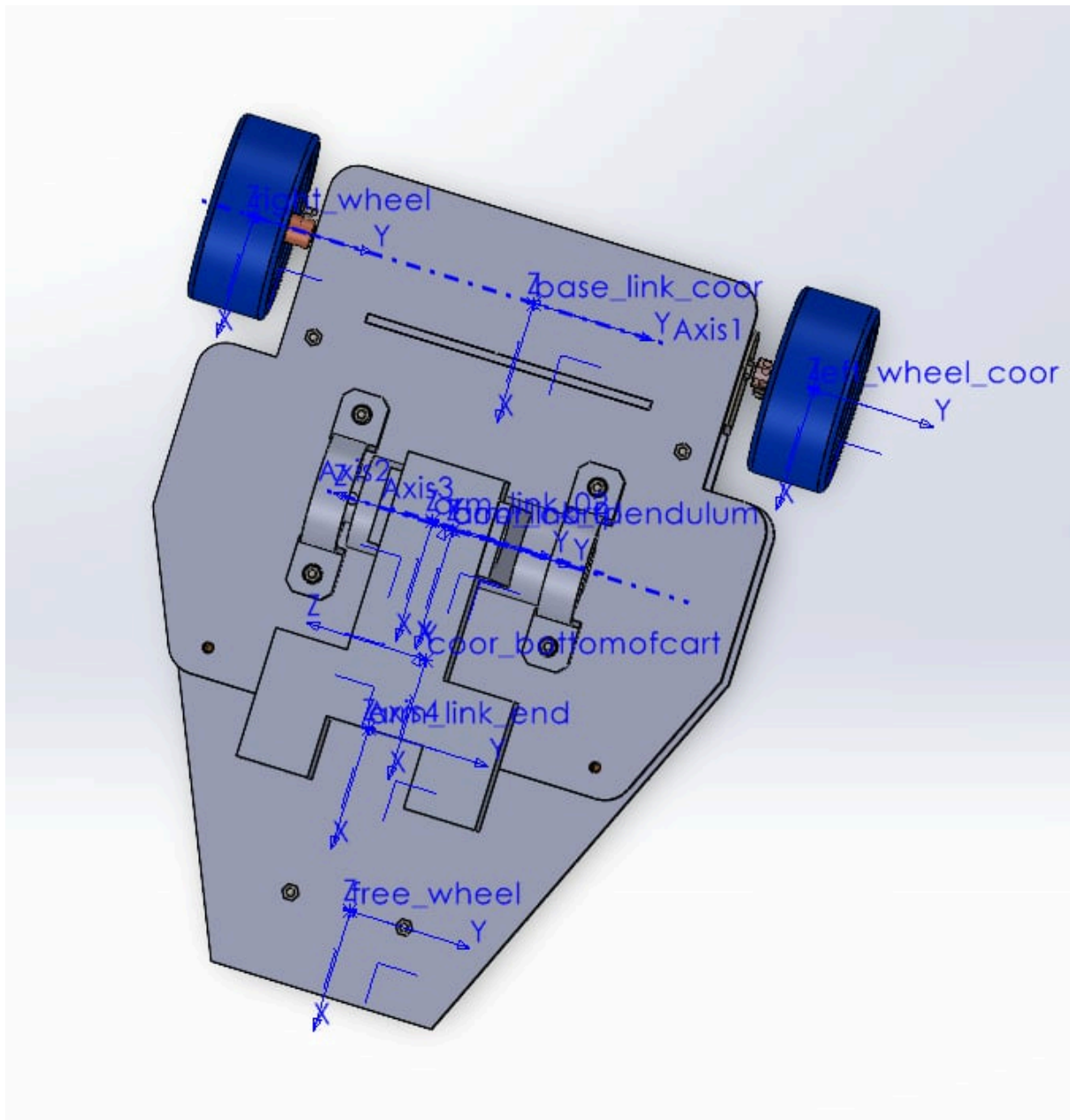
- **Gia tốc tuyến tính:** Đo lường sự thay đổi vận tốc theo thời gian trên các trục x, y, z.

Mô hình xe





Mặt dưới



Mặt trên

2.3. Cách đặt trục tọa độ

Hệ trục tọa độ được xác định như sau:

- Trục X: Hướng về phía trước của robot.
- Trục Z: Hướng lên trong không gian 3D.
- Trục Y: Được xác định theo trục X và trục Z theo quy tắc bàn tay phải

Cách đặt trục tay máy dựa theo DH:

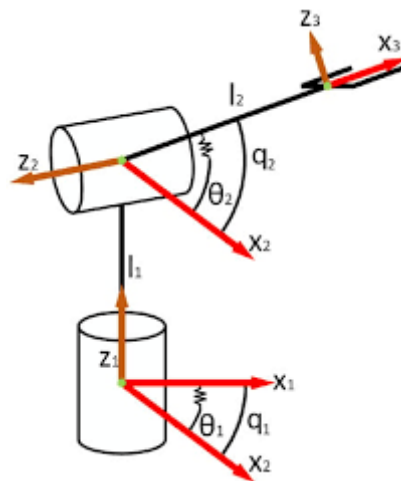
Mô hình robot được thiết kế trên SolidWorks, đảm bảo đúng tỷ lệ thực tế và có thể xuất sang URDF để tích hợp vào ROS. Hệ trục tọa độ được xác định như sau:

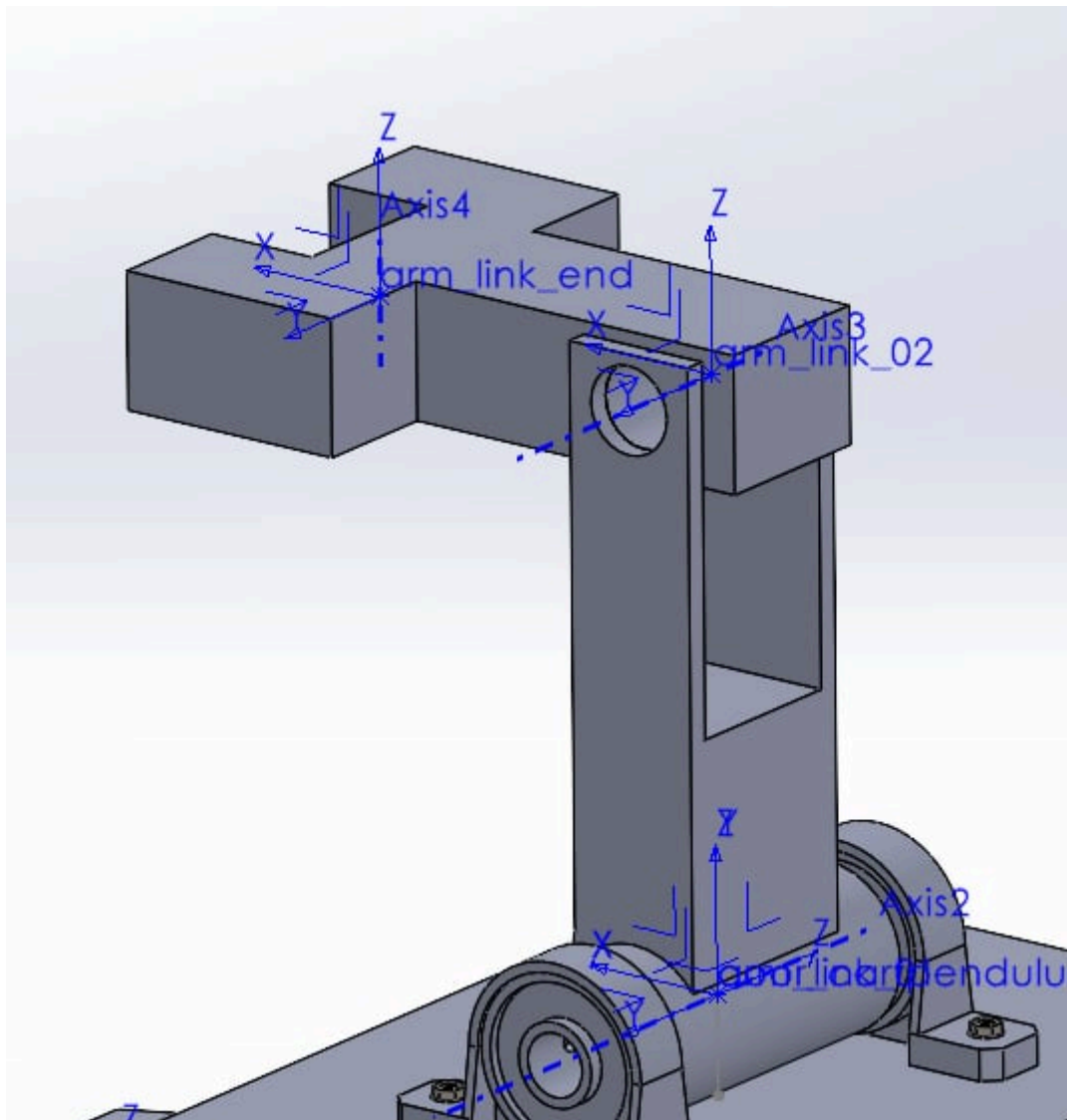
Trục X: Hướng về phía trước của robot.

Trục Z: Hướng lên trong không gian 3D.

Trục Y: Được xác định theo trục X và trục Z theo quy tắc bàn tay phải

Cách đặt trục tay máy dựa theo DH:





Tay máy

Vì không vẽ cảm biến ngay từ trong SolidWorks nên việc đặt trục và khối được thực hiện trực tiếp bằng code ở file Urdf sau khi được xuất :

LiDAR :

```

<link name="lidar_link">

<visual>
|   <geometry>
|   |   <cylinder length="0.01" radius="0.01"/>
|   </geometry>
</visual>

</link>

<joint
|   name="lidar_joint"
|   type="fixed">
|   <origin
|   |   xyz="0.2 0 0.001"
|   |   rpy="0 0 0" />
|   <parent
|   |   link="base_link" />
|   <child
|   |   link="lidar_link" />
|   <axis
|   |   xyz="0 0 0" />
</joint>

```

IMU:

```

<link name="imu_link">
</link>

<joint
|   name="limu_joint"
|   type="fixed">
|   <origin
|   |   xyz="0.0 0 0.1"
|   |   rpy="0 0 0" />
|   <parent
|   |   link="base_link" />
|   <child
|   |   link="imu_link" />
|   <axis
|   |   xyz="0 0 0" />
</joint>

```

Camera:

```

<link name="camera_link">
  <visual>
    <geometry>
      <box size="0.01 0.02 0.01"/>
    </geometry>
  </visual>
</link>













<joint
  name="camera_joint"
  type="fixed">
  <origin
    xyz="0.15 0 0.04"
    rpy="0 0 0" />
  <parent
    link="base_link" />
  <child
    link="camera_link" />
  <axis
    xyz="0 0 0" />
</joint>

```

2.4. Xuất file URDF từ SolidWorks

Để chuyển đổi mô hình từ SolidWorks sang định dạng URDF sử dụng trong ROS, công cụ URDF Exporter được áp dụng. Quá trình này giúp giảm thời gian chỉnh sửa và đảm bảo tính chính xác của mô hình khi đưa vào Gazebo.

2.5. Cấu trúc thư mục

 ThanhDuck Add files via upload		1739b71 · 2 minutes ago	 13 Commits
 config	Delete config/rviz.rviz	2 minutes ago	
 launch	Delete launch/gazebo.launch	10 hours ago	
 meshes	Add file	19 hours ago	
 rviz	Add files via upload	2 minutes ago	
 scripts	Add files via upload	19 hours ago	
 urdf	Add files via upload	19 hours ago	
 CMakeLists.txt	Add files via upload	37 minutes ago	
 README.md	Initial commit	19 hours ago	
 export.log	Add files via upload	27 minutes ago	
 package.xml	Add files via upload	37 minutes ago	

Để mô tả cấu trúc của robot trong ROS thì cấu trúc thư mục gồm :

- config/: Có thể chứa các file cấu hình (như thông số PID, bộ lọc Kalman, v.v.).
- launch/: Chứa các file .launch để khởi chạy mô phỏng, RViz, Gazebo, hoặc node điều khiển.
- meshes/: Chứa các file mô hình 3D (ở dạng .stl) của robot.
- rviz/: Có thể chứa các cấu hình hiển thị cho RViz.
- scripts/: Chứa các script Python, có thể dùng để điều khiển hoặc test robot.
- urdf/: Chứa file mô tả robot URDF/Xacro.
- CMakeLists.txt & package.xml: Các file cần thiết để build package trong ROS.

2.6. Mô phỏng trong Gazebo

Để mô phỏng Robot cũng như cảm biến trong Gazebo ta thêm vào các plugins vào file URDF :

- Tích hợp plugin cho xe : mô phỏng xe 2 bánh vi sai (điều khiển độc lập tốc độ của 2 bánh)

```
<gazebo>
  <plugin name="robot_base" filename="libgazebo_ros_diff_drive.so">
    <commandTopic>cmd_vel</commandTopic>
    <odometryTopic>odom</odometryTopic>
    <odometryFrame>odom</odometryFrame>
    <odometrySource>world</odometrySource>
    <publishOdomTF>true</publishOdomTF>
    <robotBaseFrame>base_link</robotBaseFrame>
    <publishWheelTF>false</publishWheelTF>
    <publishTf>true</publishTf>
    <publishWheelJointState>true</publishWheelJointState>
    <legacyMode>false</legacyMode>
    <updateRate>30</updateRate>
    <leftJoint>left_wheel_joint</leftJoint>
    <rightJoint>right_wheel_joint</rightJoint>
    <wheelSeparation>0.160</wheelSeparation>
    <wheelDiameter>0.066</wheelDiameter>
    <wheelAcceleration>1</wheelAcceleration>
    <wheelTorque>50</wheelTorque>
    <rosDebugLevel>na</rosDebugLevel>
  </plugin>
</gazebo>
```

- Tích hợp plugin Camera : Cho phép robot quan sát môi trường xung quanh trong mô phỏng.

```

<!-- camera -->
<gazebo reference="camera_link">
  <sensor type="camera" name="camera1">
    <update_rate>30.0</update_rate>
    <visualize>true</visualize>
    <camera name="head">
      <horizontal_fov>1.3962634</horizontal_fov>
      <image>
        <width>480</width>
        <height>240</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>/camera</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>
</gazebo>

```

- Tích hợp plugin IMU : Mô phỏng gia tốc kế và con quay hồi chuyển để đo vận tốc và hướng.

```

<gazebo reference="imu_link">
  <gravity>true</gravity>
  <sensor name="imu_sensor" type="imu">
    <always_on>true</always_on>
    <update_rate>10</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>

```

- Tích hợp plugin LiDAR : Tạo ra đám mây điểm (point cloud) mô phỏng việc quét laser.


```

<gazebo reference="lidar_link">
  <sensor type="ray" name="head_hokuyo_sensor">
    <pose>0 0 0 0 0 0</pose>
    <visualize>true</visualize>
    <update_rate>40</update_rate>
    <ray>
      <scan>
        <horizontal>
          <samples>720</samples>
          <resolution>1</resolution>
          <min_angle>-1.570796</min_angle>
          <max_angle>1.570796</max_angle>
        </horizontal>
      </scan>
      <range>
        <min>0.10</min>
        <max>30.0</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>/scan</topicName>
      <frameName>lidar_link</frameName>
    </plugin>
  </sensor>
</gazebo>

```

- Tích hợp Plugin cho tay máy : cho phép mô phỏng, điều khiển và tương tác với tay máy này trong môi trường ROS.

```

<!-- Transmission -->
<transmission name="j1">
  <type>transmission_interface/SimpleTransmission</type>
  <joint name="arm_01_joint">
    <hardwareInterface>hardware_interface/PositionJointInterface</hardwareInterface>
  </joint>
  <actuator name="M1">
    <hardwareInterface>hardware_interface/PositionJointInterface</hardwareInterface>
    <mechanicalReduction>1</mechanicalReduction>
  </actuator>
</transmission>

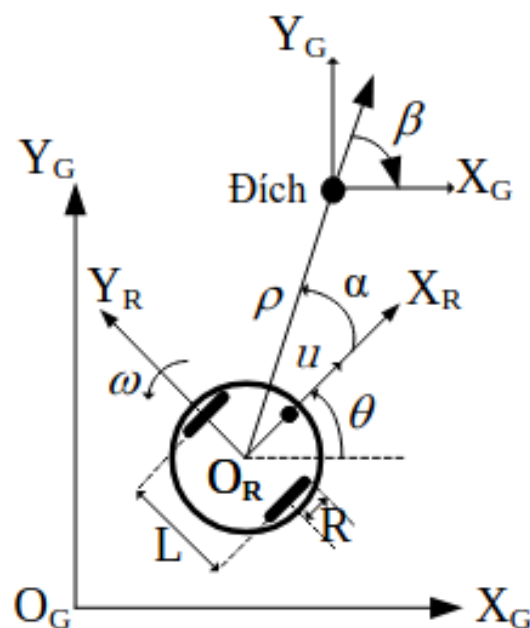
<transmission name="j2">
  <type>transmission_interface/SimpleTransmission</type>
  <joint name="arm_02_joint">
    <hardwareInterface>hardware_interface/PositionJointInterface</hardwareInterface>
  </joint>
  <actuator name="M2">
    <hardwareInterface>hardware_interface/PositionJointInterface</hardwareInterface>
    <mechanicalReduction>1</mechanicalReduction>
  </actuator>
</transmission>

```

3. MÔ HÌNH ĐỘNG HỌC

3.1. Động học di chuyển

Robot di động hai bánh vi sai bao gồm hai bánh chuyển động và một bánh dẫn động. Hoạt động vi sai là sự chênh lệch tốc độ quay giữa hai bánh xe làm robot chuyển động theo một cung tròn có tâm nằm trên trục bánh xe. Mô hình hình học của robot di động hai bánh vi sai trong hệ tọa độ Đề các được thể hiện như Hình



Hình : Mô hình robot di động hai bánh vi sai

Trong đó (X_G, Y_G) là hệ tọa độ toàn cục, (X_R, Y_R) là hệ tọa độ gắn với robot. Tư thế của robot được biểu diễn thông qua biến trạng thái (x, y, θ) , với (x, y) là vị trí và θ là góc hướng của robot so với hệ tọa độ toàn cục. Hai biến vận tốc liên quan đến chuyển động của robot là vận tốc tuyến tính u và vận tốc góc ω . Mô hình động học biểu diễn sự thay đổi trạng thái của robot thông qua biến lỗi vào vận tốc như sau:

$$\begin{cases} \dot{x}(t) = u(t) \cos(\theta(t)) \\ \dot{y}(t) = u(t) \sin(\theta(t)) \\ \dot{\theta}(t) = \omega(t) \end{cases} \quad (1)$$

Mô hình (1) với ràng buộc không khả tích, vì thế khi điều khiển chuyển động ổn định từ một điểm bất kỳ tới một điểm đích, mô hình (1) sử dụng thêm các biến: $\rho > 0$ là khoảng cách từ vị trí hiện tại (x_k, y_k) tới vị trí đích (x_d, y_d) của robot; α là góc lệch giữa véc tơ khoảng cách và véc tơ hướng; β là góc hướng của robot khi về đích.

3.2. Điều khiển trong ROS

- **ROS Control:** Hỗ trợ điều khiển chính xác tốc độ và hướng di chuyển của bánh xe.

```
<gazebo>
  <plugin name="gazebo_ros_control"
    filename="libgazebo_ros_control.so">
    <robotNamespace>/robot_arm</robotNamespace>
    <robotSimType>gazebo_ros_control/DefaultRobotHWSim</robotSimType>
    <legacyModeNS>true</legacyModeNS>
  </plugin>
</gazebo>
```

Plugin gazebo_ros_control trong Gazebo có nhiệm vụ kết nối hệ thống điều khiển của ROS với mô phỏng Gazebo, giúp bạn điều khiển robot bằng các controller từ ROS.

Cụ thể, plugin này:

Kết nối Gazebo với ROS Control: Nó cho phép sử dụng ros_control để điều khiển các joint của robot trong Gazebo thông qua các controller như :

- **effort_controllers/JointEffortController**
- **position_controllers/JointPositionController**
- **velocity_controllers/JointVelocityController**

Cấp quyền điều khiển cho ROS: Khi plugin này được thêm vào mô hình, các topic như /robot/joint_states và /robot/controller_manager sẽ được tạo, giúp ROS có thể gửi lệnh điều khiển đến các joint của robot.

Mô phỏng hệ thống động lực học: Nếu bạn có một robot di động hoặc cánh tay robot, plugin này giúp mô phỏng chính xác lực, vận tốc và vị trí của các khớp khi điều khiển.

```
robot_arm:

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

  joint_1_pos_controller:
    type : position_controllers/JointPositionController
    joint : arm_01_joint
    pid : { p: 100, i: 0.0, d: 10}
    gainn : 0.1

  joint_2_pos_controller:
    type : position_controllers/JointPositionController
    joint : arm_02_joint
    pid : { p: 100, i: 0.0, d: 10}
    gainn : 0.1

  arm_controller:
    type : position_controllers/JointTrajectoryController
    joints :
      - arm_01_joint
      - arm_02_joint
```

File .yaml định nghĩa các bộ phận điều khiển khớp của robot

File control.py được viết để điều khiển robot bằng cách nhận lệnh từ bàn phím hoặc từ topic /cmd_vel của ROS. Các lệnh di chuyển như tiến, lùi, xoay

trái/phải, và dịch chuyển trái phải được chuyển thành vận tốc bánh xe và gửi đến controller tương ứng.

- Thiết lập ROS node và tạo các publs

```
# Publishers for controlling joints
class PublishThread(threading.Thread):
    def __init__(self, rate):
        super(PublishThread, self).__init__()
        self.publisher = rospy.Publisher('cmd_vel', TwistMsg, queue_size=1)
        self.joint_1_pub = rospy.Publisher('/robot_arm/joint_1_pos_controller/command', Float64, queue_size=1)
        self.joint_2_pub = rospy.Publisher('/robot_arm/joint_2_pos_controller/command', Float64, 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.joint_1_pos = 0.0
        self.joint_2_pos = 0.0
        self.condition = threading.Condition()
        self.done = False

        if rate != 0.0:
            self.timeout = 1.0 / rate
        else:
            self.timeout = None

        self.start()

    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")
```

Đọc phím nhấn từ bàn phím

```
def getKey(settings, timeout):
    if sys.platform == 'win32':
        key = msvcrt.getwch()
    else:
        tty.setraw(sys.stdin.fileno())
        rlist, _, _ = select([sys.stdin], [], [], timeout)
        if rlist:
            key = sys.stdin.read(1)
        else:
            key = ''
        termios.tcsetattr(sys.stdin, termios.TCSADRAIN, settings)
    return key
```

Lưu lại các cài đặt hiện tại của terminal. Mục đích là để có thể khôi phục lại các cài đặt này sau khi chương trình đã thay đổi chúng

```
def restoreTerminalSettings(old_settings):
    if sys.platform == 'win32':
        return
    termios.tcsetattr(sys.stdin, termios.TCSADRAIN, old_settings)
```

Tạo một chuỗi thông báo hiển thị tốc độ tuyến tính và tốc độ góc hiện tại

```
def vels(speed, turn):
    return "currently:\tspeed %s\tturn %s " % (speed, turn)
```

Cấu hình file launch

Để chạy được 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, sau đó mở rviz đã có sẵn config của cảm biến và nạp các thông số cần thiết

launch

```
<include file="$(find gazebo_ros)/launch/empty_world.launch" />

<param name="robot_description" textfile="$(find robot_arm_01)/urdf/robot_arm_01.urdf" />

<node name="spawn_model" pkg="gazebo_ros" type="spawn_model" args="-file $(find robot_arm_01)/urdf/robot_arm_01.urdf -urdf -model robot_arm -x 0 -y 0 -z 0" output="screen" />

<rosparam command="load" file="$(find robot_arm_01)/config/arm_robot_config.yaml" />
<rosparam command="load" file="$(find robot_arm_01)/config/joint_state_controller.yaml" />

<node name="controller_spawner" pkg="controller_manager" ns="/robot_arm" type="spawner" args="joint_state_controller joint_1_pos_controller joint_2_pos_controller" />

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

<node name="joint_state_controller_spawner" pkg="controller_manager" type="controller_manager" args="spawn joint_state_controller" respawn="false" output="screen" />

<node name="robot_state_publisher_01" pkg="robot_state_publisher" type="robot_state_publisher" respawn="false" output="screen">
  <remap from="/joint_states" to="/robot_arm/joint_states" />
</node>


<node name="rviz" pkg="rviz" type="rviz" args="-d $(find robot_arm_01)/urdf.rviz" />


</launch>
```


4.TỔNG KẾT

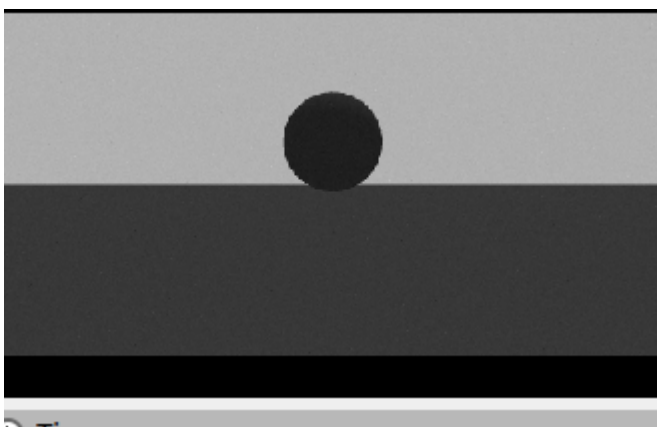
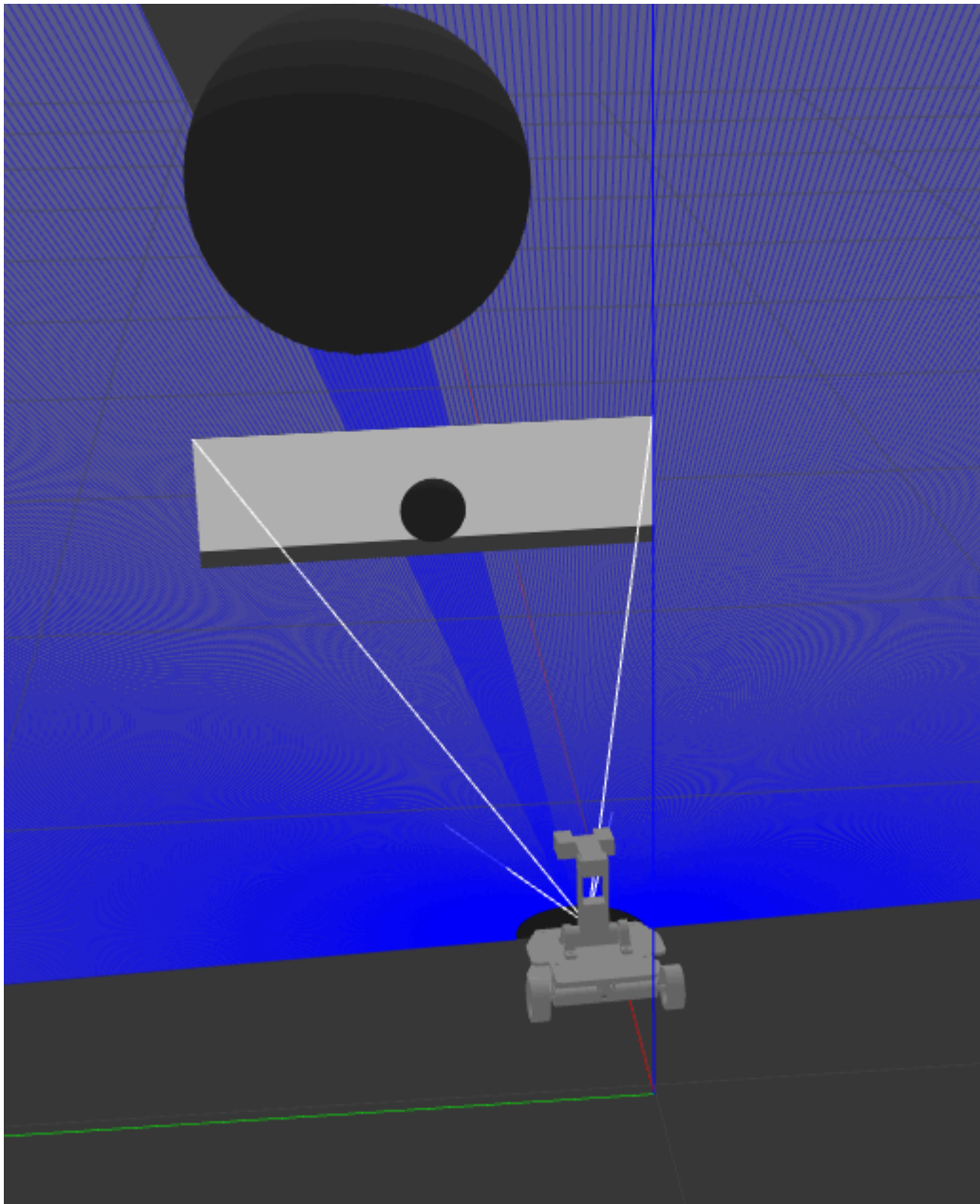
4.1. Kết quả

Hiện ra các topic trên Rviz

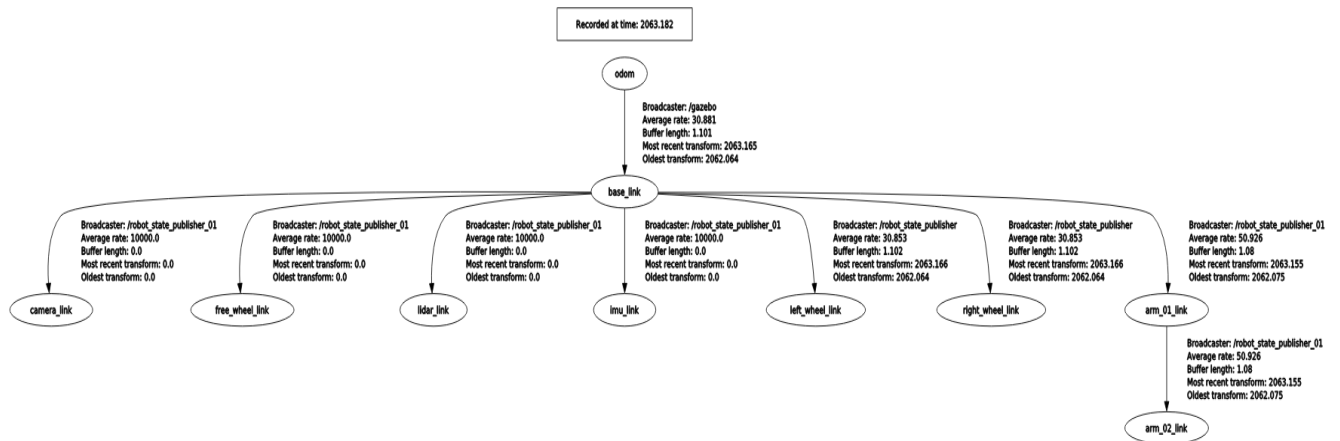
▼  Image	<input checked="" type="checkbox"/>
▶ ✓ Status: Ok	
Image Topic	/camera/image_raw
Transport Hint	raw
Queue Size	2
Unreliable	<input type="checkbox"/>

▼  LaserScan	<input checked="" type="checkbox"/>
▶ ✓ Status: Ok	
Topic	/scan
Unreliable	<input type="checkbox"/>
Queue Size	10
Selectable	<input checked="" type="checkbox"/>
Style	Flat Squares
Size (m)	0,01
Alpha	1
Decay Time	0
Position Transf...	XYZ
Color Transfor...	Intensity
Channel Name	intensity
Use rainbow	<input checked="" type="checkbox"/>
Invert Rainbow	<input type="checkbox"/>
Min Color	■ 0; 0; 0
Max Color	<input type="checkbox"/> 255; 255; 255
Autocompute I...	<input checked="" type="checkbox"/>
Min Intensity	0
Max Intensity	0

▼  Imu	<input checked="" type="checkbox"/>
▶ ✓ Status: Ok	
Topic	/imu
Unreliable	<input type="checkbox"/>
Queue Size	10
Color	■ 204; 51; 204
Alpha	1
History Length	1



Rqt_tf_tree



LINK DEMO :  DEMO.mkv

4.2. Tài liệu tham khảo

[1]<https://ieeexplore.ieee.org/abstract/document/1248848>

[2]https://eprints.uet.vnu.edu.vn/eprints/id/eprint/3261/1/REV-ECIT_2018_paper_65.pdf