ĐẠI HỌC QUỐC GIA HÀ NỘI

TRƯỜNG ĐẠI HỌC CÔNG NGHỆ

----------***----------

BÀI BÁO CÁO

MÔN HỌC: LẬP TRÌNH ROBOT VỚI ROS

Họ và tên sinh viên: Đinh Mạnh Quân

MSV: 22027522

Hà Nội, ngày 1 tháng 4 năm 2025

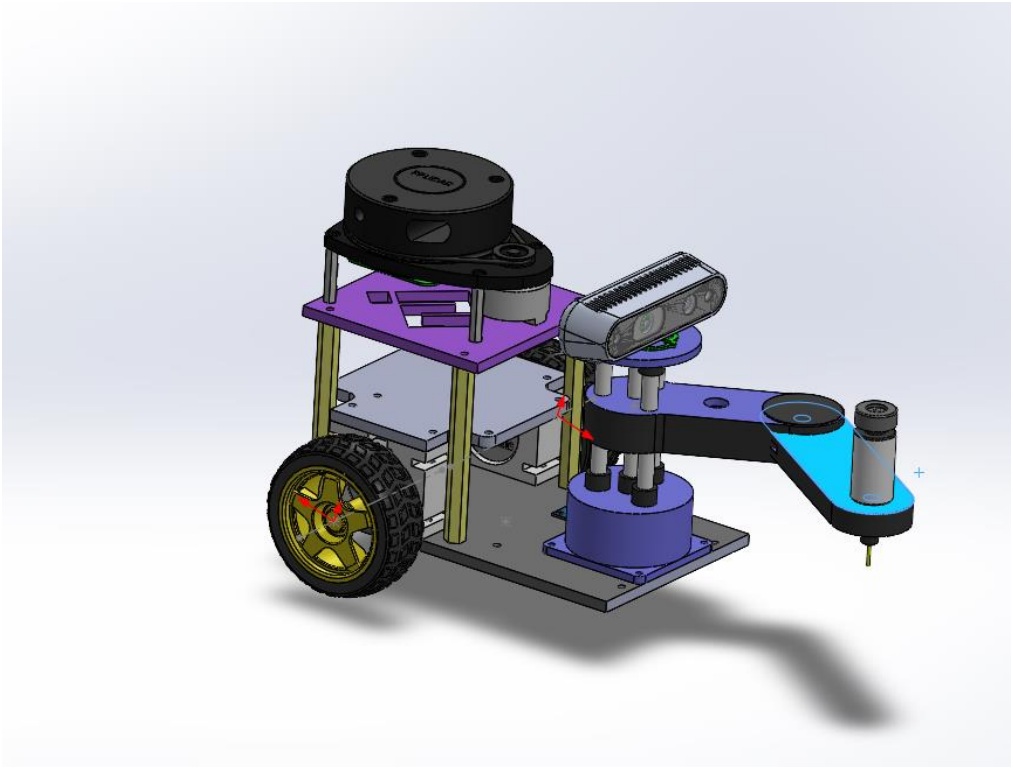
MỤC LỤC

1 Tổng quan

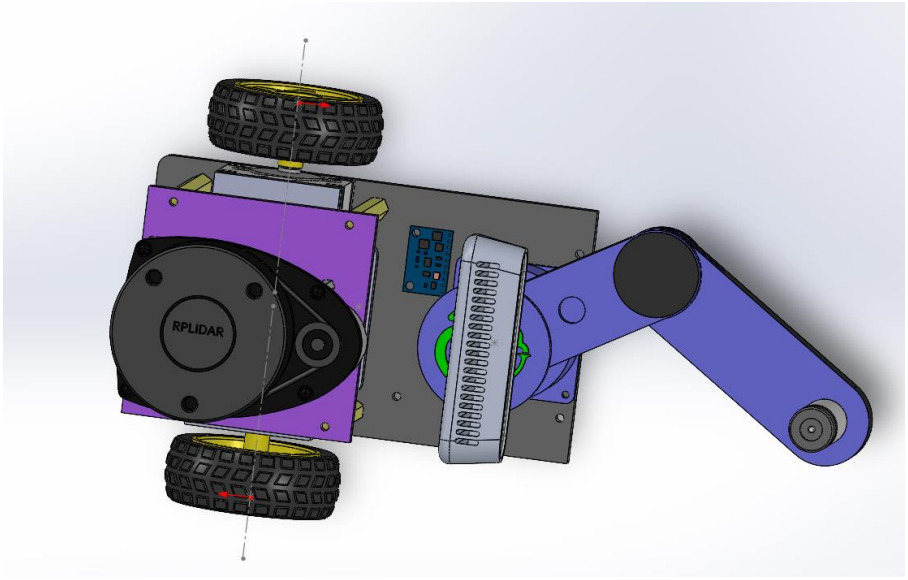
- Sử dụng phần mềm solidworks để thiết kế mô hình 3D với đề tài Robot hai bánh vi sai(differential drive) với hai khớp tay máy bao gồm khớp 1 sử dụng khớp tịnh tiến và khớp 2 sử dụng khớp xoay. Ngoài ra đề tài sử dụng 3 loại cảm biến phổ biến trong thực tế là Lidar, camera và imu.
- Sau khi xây dựng mô hình trên solidworks thực hiện quá trình export to urdf để nhúng mô hình vào trong ros(gazebo) để điều khiển robot.
- Tên package: test1

2 Thiết kế mô hình

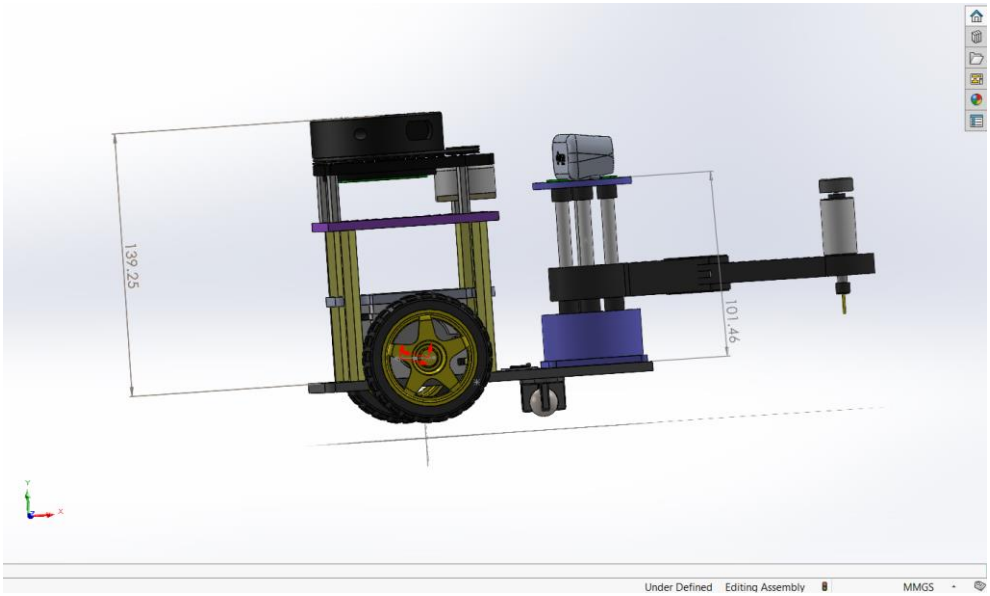
2.1. Thiết kế



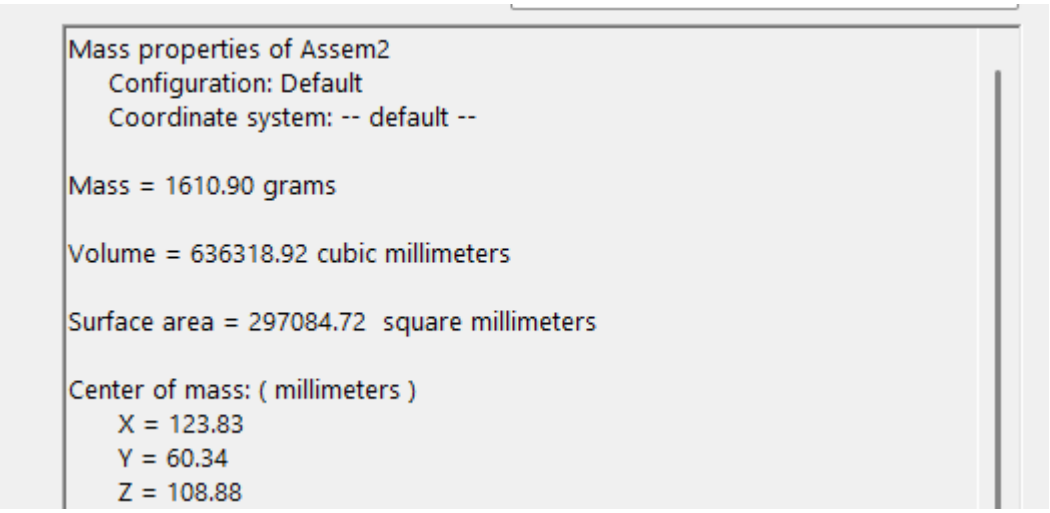
- Về mặt thiết kế, mô hình dựa theo cách thiết kế của con burger trong turtlebot 3. Tay máy gồm khớp 1 là khớp tịnh tiến, khớp 2 là khớp xoay được lấy ý tưởng từ tay máy scara.



- Hình ảnh từ mặt trên nhìn xuống trên solidworks với lidar được đặt ở vị trí cao nhất để có thể định vị, quét không gian 280 độ và tính toán khoảng cách, xác định vật cản. Camera được đặt trên cánh tay máy để có thể nhận diện vật cản, xuất hiện đối tượng ở phía trước robot. Imu được đặt ở vị trí khung robot để xác định trạng thái, xác định gia tốc theo x-y-z



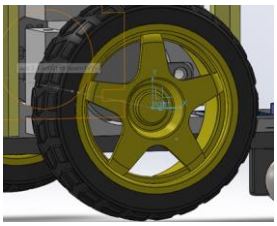
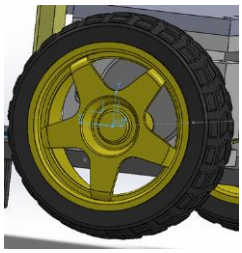
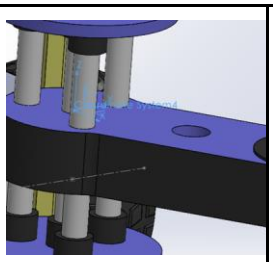
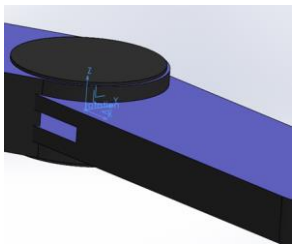
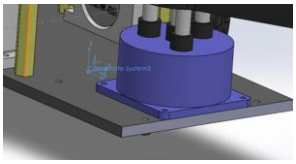
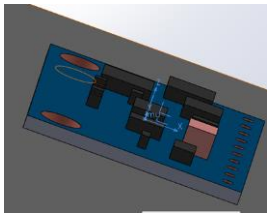
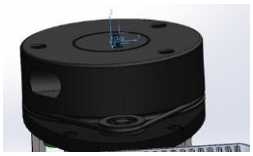
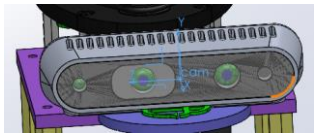
- Chiều cao chiếc xe và độ cao của thanh trượt theo đơn vị mm



- Chất liệu của khung xe, 2 bánh và 2 khớp tay máy được thiết kế bằng nhựa nên khối lượng nó khá nhẹ khoảng 1.6kg.

2.2. Gắn trục/export to urdf

a, Đặt trục

 Bánh phải	 Bánh trái	 Khớp 1	 Khớp 2
 Khung xe	 imu	 LIdar	 Camera

b, Export to urdf

- Base_link: để khung xe(phần cố định) cùng với thanh trượt của scara
- Từ base_link: tạo ra 6 child_link bao gồm left_link(bánh trái), right_link(bánh phải), lidar_link, camera_link, imu_Link, prismatic(Link 1) - tạo thêm child_link từ khớp prismatic là rotation(Link 2).
- lidar_link, imu_Link, camera_link: type fixed, axis: automatically generate
- left_link, right_link: type-continuous.

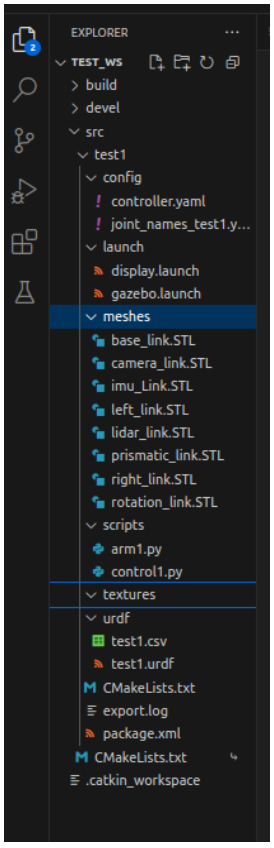
- prismatic: type-prismatic.
- rotation: type-revolute.

3 Mô phỏng quá trình thực hiện chi tiết điều khiển robot trong Ros/gazebo

3.1. Mô hình trong gazebo/rviz

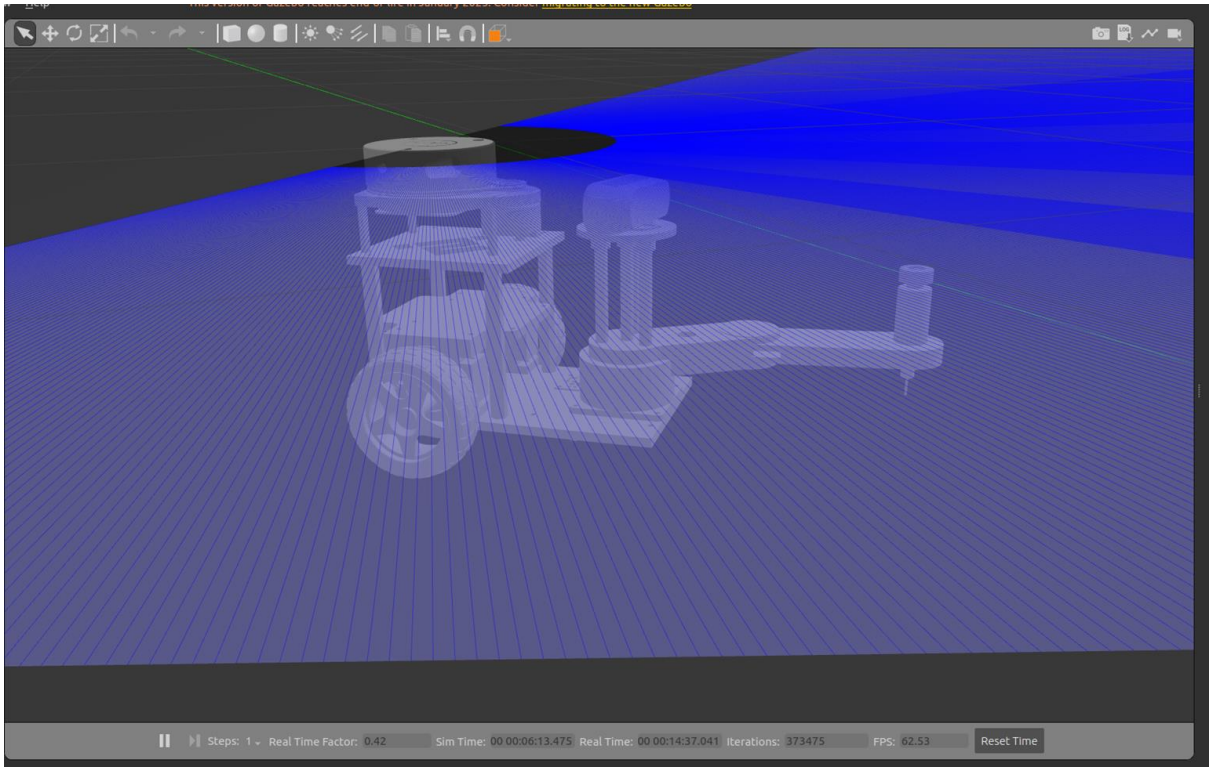
a, gazebo

- cấu trúc các thư mục build trong mô hình:



- sử dụng lệnh: **roslaunch test1 gazebo.launch** (mở gazebo)

-> mở gazebo và tự động add mô hình với phạm vi hoạt động của lidar với góc đã được setup trong file urdf góc 180 độ

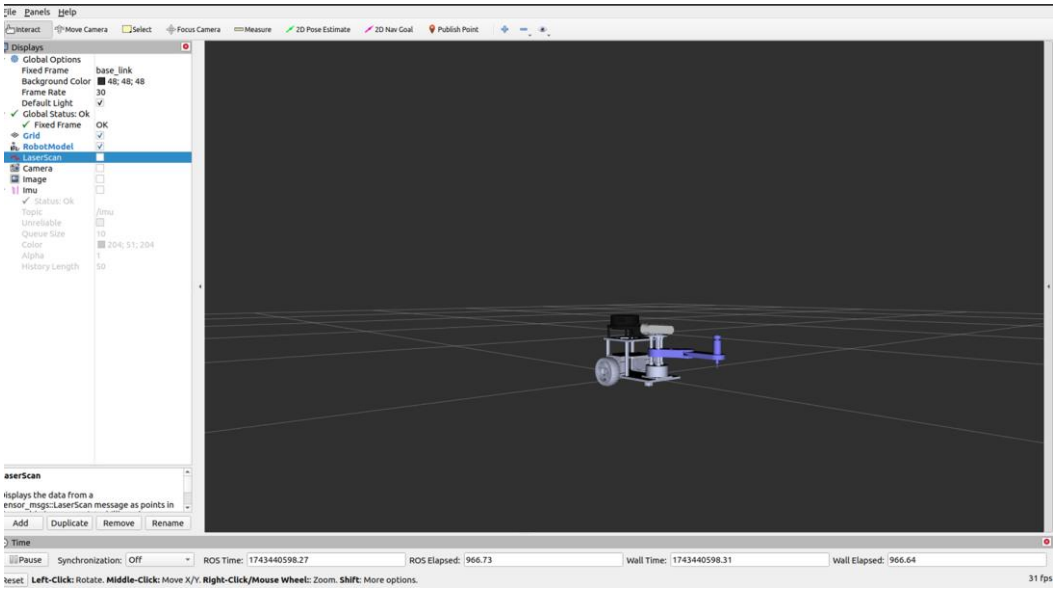


b, rviz

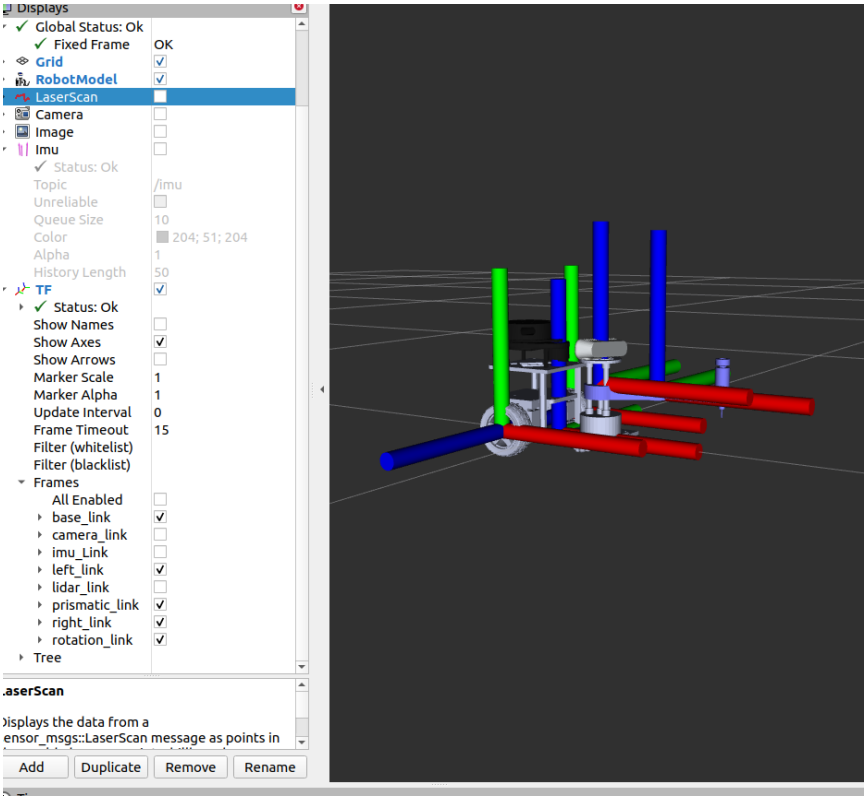
- Sử dụng lệnh: **roslaunch test1 display.launch** để mở rviz

Add > Robot model(TF)

Fixed frame: base_link



- TF:



c, meshes

-> Chứa các file mô hình 3d dưới dạng file .stl bao gồm: link gốc và các link con(các cảm biến, tay máy)

3.2. Các loại cảm biến

***** Lệnh:

- **roslaunch test1 display.launch**: mở rviz
- Add > Robot model (sau đó chọn các loại cảm biến như lidar, camera, imu)
- Fixed frame: base_lịnk

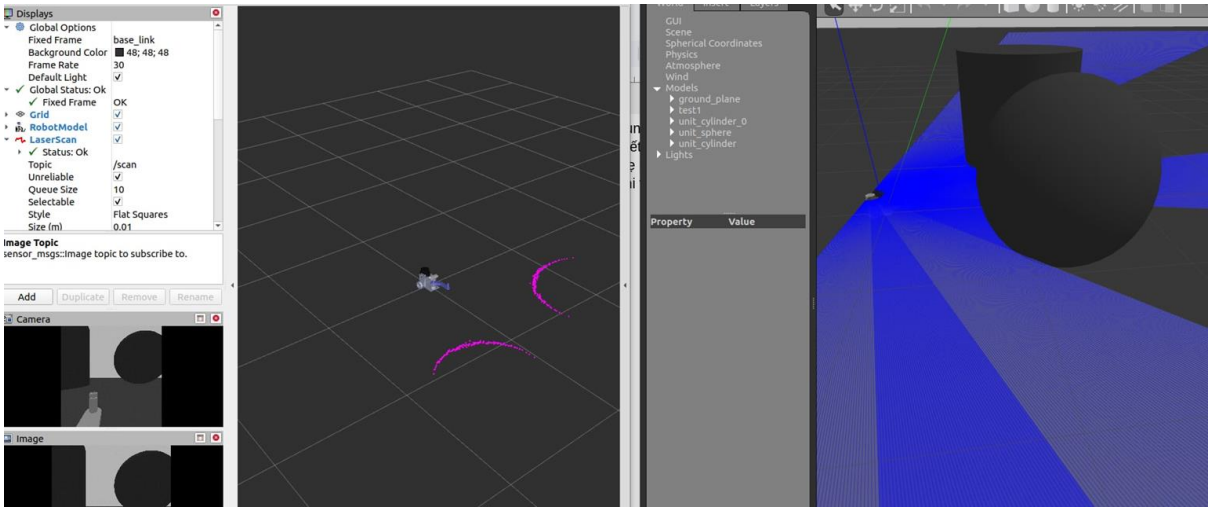
a, Lidar

- Add plugin lidar vao trong file urdf

```
<!-- Plugin lidar -->
 <gazebo reference="lidar_link">
  <sensor type="ray" name="lidar_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>-1.5708</min_angle>
        <max_angle>1.5708</max_angle>
      </horizontal>
    </scan>
    <range>
      <min>0.1</min>
      <max>30.0</max>
      <resolution>0.01</resolution>
    </range>
    <noise>
      <type>gaussian</type>
      <mean>0.0</mean>
      <stddev>0.01</stddev>
    </noise>
     </ray>
     <plugin name="gazebo_ros_laser"
filename="libgazebo_ros_laser.so">
    <topicName>/scan</topicName>
    <frameName>lidar_link</frameName>
     </plugin>
   </sensor>
    </gazebo>
```

- Add > LaserScan (Topic > /scan)
- Kết quả:



b, Camera

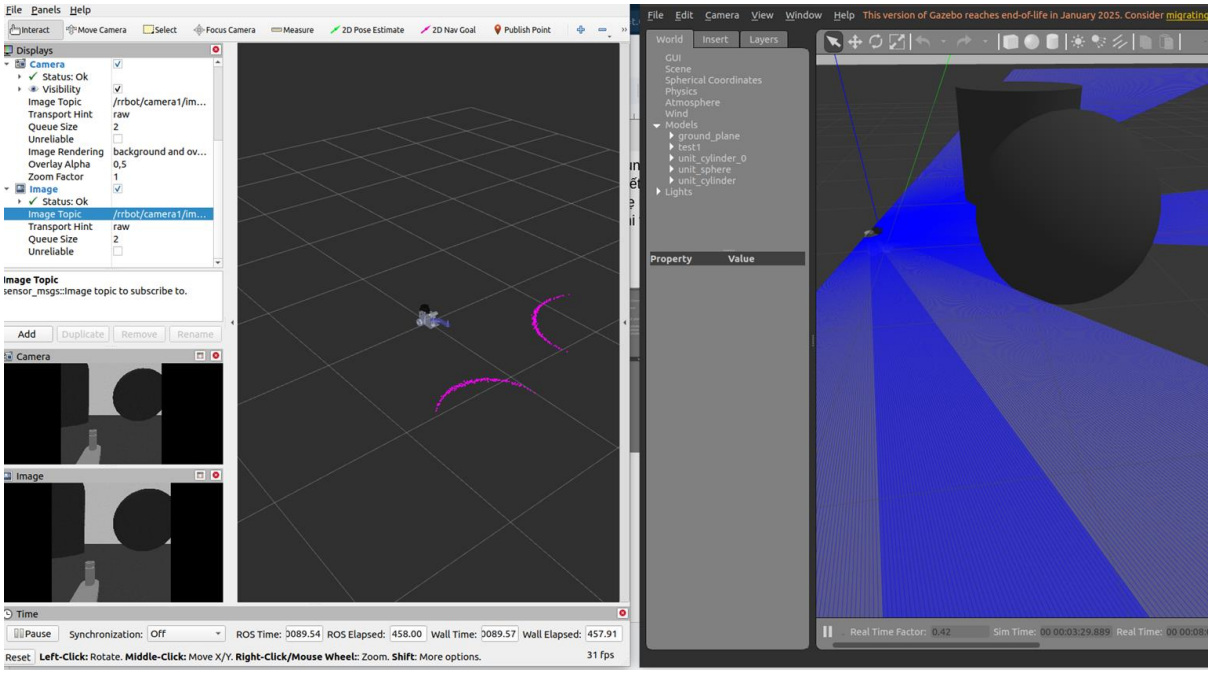
-Add plugin camera trong file urdf:

```
<!-- Camera Plugin -->
 <gazebo reference="camera_link">
   <sensor type="camera" name="camera1">
     <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>
         <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/camera1</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>
```

- Add > Camera (Topic > /rrbot/camera1/image_raw)

- Add > Image (Topic > /rrbot/camera1/image_raw)

-> Kết quả:



c, imu

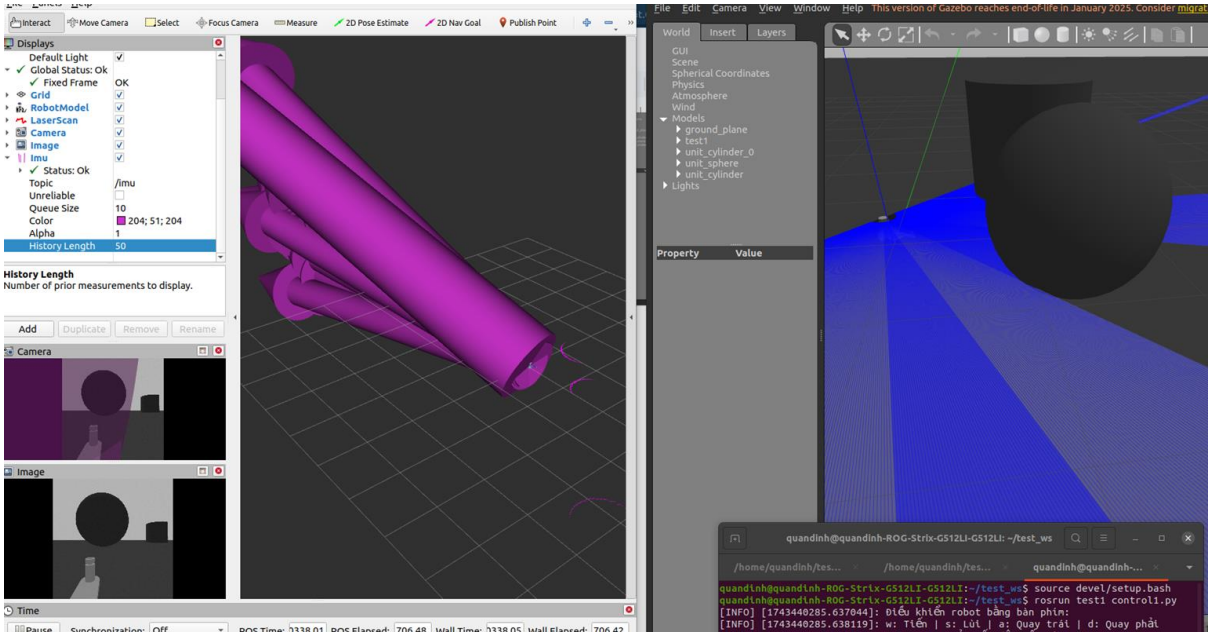
- Add plugin imu trong file urdf

```
<gazebo reference="imu_Link">
 <sensor name="imu_sensor" type="imu">
   <always_on>true</always_on>
   <update_rate>100</update_rate>
   <visualize>true</visualize> <!-- Bật hiển thị mũi tên -->
   <plugin filename="libgazebo_ros_imu_sensor.so" name="imu_plugin">
     <topicName>/imu</topicName>
     <bodyName>imu_Link</bodyName>
     <frameName>imu_Link</frameName>
     <gaussianNoise>0.01</gaussianNoise>
     <xyzOffset>0 0 0</xyzOffset>
     <rpyOffset>0 0 0</rpyOffset> <!-- Không offset góc dữ liệu -->

<initialOrientationAsReference>false</initialOrientationAsReference>
   </plugin>
   <pose>0 0 0 0 1.5708 0</pose> <!-- Xoay 90 độ quanh trục Y để mũi
tên hướng theo X -->
 </sensor>
</gazebo>
```

-Add > imu (Topic > /imu)

-Điều khiển xe di chuyển trong bản đồ sau đó imu sẽ tự tự động cập nhật trạng thái gia tốc của chiếc xe
-> Kết quả:



3.3. Điều khiển xe 2 bánh vi sai qua keyboard

- Chạy lệnh: **rosrun test1 control1.py**

(test1: tên pkg, control1.py: file code bằng python điều khiển robot)

- Cấp quyền qua lệnh chmod +x
- Code control1.py

```
#!/usr/bin/env python3
import rospy
from std_msgs.msg import Float64
import sys
import tty
import termios

class KeyboardControl:
    def __init__(self):
        # Khởi tạo node ROS
        rospy.init_node('keyboard_control', anonymous=True)

        # Publisher cho các lệnh vận tốc bánh xe
```

```
        self.pub_left =
rospy.Publisher('/left_wheel_joint_velocity_controller/command',
Float64, queue_size=10)
        self.pub_right =
rospy.Publisher('/right_wheel_joint_velocity_controller/command',
Float64, queue_size=10)

        # Tốc độ giới hạn
        self.max_speed = 10  # Giới hạn vận tốc tối đa (radian/s)
        self.min_speed = -10  # Giới hạn vận tốc tối thiểu (radian/s)
        self.speed_step = 1  # Mức tăng/giảm tốc độ

        # Vận tốc ban đầu
        self.linear_speed = 0.0  # Tốc độ tiến/lùi
        self.angular_speed = 0.0  # Tốc độ quay

    def get_key(self):
        """Đọc phím từ bàn phím mà không cần nhấn Enter"""
        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

    def run(self):
        rospy.loginfo("Điều khiển robot bằng bàn phím:")
        rospy.loginfo("w: Tiến | s: Lùi | a: Quay trái | d: Quay phải")
        rospy.loginfo("q/z: Tăng/Giảm tốc độ tiến/lùi")
        rospy.loginfo("e/c: Tăng/Giảm tốc độ quay")
        rospy.loginfo("x: Dừng | f: Dừng và thoát")

        while not rospy.is_shutdown():
            key = self.get_key()

            # Điều khiển tốc độ tuyến tính
            if key == 'w':  # Đi thẳng
                left_speed = -self.linear_speed
                right_speed = -self.linear_speed
            elif key == 's':  # Đi lùi
```

```
                left_speed = self.linear_speed
                right_speed = self.linear_speed
            elif key == 'a':  # Quay trái
                left_speed = self.angular_speed
                right_speed = -self.angular_speed
            elif key == 'd':  # Quay phải
                left_speed = -self.angular_speed
                right_speed = self.angular_speed
            elif key == 'x':  # Dừng
                left_speed = 0.0
                right_speed = 0.0
            elif key == 'f':  # Dừng và thoát
                left_speed = 0.0
                right_speed = 0.0
                self.pub_left.publish(left_speed)
                self.pub_right.publish(right_speed)
                rospy.loginfo("Dừng robot và thoát")
                break
            elif key == 'q':  # Tăng tốc độ tiến/lùi
                self.linear_speed = min(self.linear_speed +
self.speed_step, self.max_speed)
                rospy.loginfo(f"Tốc độ tiến/lùi:
{self.linear_speed:.2f}")
                continue  # Không cần gửi lệnh vận tốc vì không thay
đổi trạng thái di chuyển
            elif key == 'z':  # Giảm tốc độ tiến/lùi
                self.linear_speed = max(self.linear_speed -
self.speed_step, 0.0)
                rospy.loginfo(f"Tốc độ tiến/lùi:
{self.linear_speed:.2f}")
                continue
            elif key == 'e':  # Tăng tốc độ quay
                self.angular_speed = min(self.angular_speed +
self.speed_step, self.max_speed)
                rospy.loginfo(f"Tốc độ quay:
{self.angular_speed:.2f}")
                continue
            elif key == 'c':  # Giảm tốc độ quay
                self.angular_speed = max(self.angular_speed -
self.speed_step, 0.0)
                rospy.loginfo(f"Tốc độ quay:
{self.angular_speed:.2f}")
```

```
                continue
            else:
                continue  # Nếu không nhấn phím hợp lệ, bỏ qua vòng lặp này

            # Gửi lệnh vận tốc
            self.pub_left.publish(left_speed)
            self.pub_right.publish(right_speed)

            # Hiển thị trạng thái
            rospy.loginfo(f"Left: {left_speed:.2f}, Right:
{right_speed:.2f}")

            rospy.sleep(0.1)  # Tránh đọc phím quá nhanh

if __name__ == "__main__":
    try:
        controller = KeyboardControl()
        controller.run()
    except rospy.ROSInterruptException:
        pass
```

- Khởi tạo node keyboard_control
- Publisher các lệnh vận tốc cho bánh xe với left_wheel_joint_velocity_controller(bánh trái), right_wheel_joint_velocity_controller(bánh phải)
- Sử dụng bàn phím máy tính để có thể di chuyển robot với
w tiến, s lùi, a quay trái, d quay phải, x dừng, f thoát
q/z: tăng/giảm tốc độ tiến/lùi
e/c: tăng/giảm tốc độ quay của bánh(để rẽ trái/phải)
- Trước khi tiến lùi hoặc rẽ thì cần phải tăng tốc độ tiến và tốc độ quay vì setup ban đầu để tốc độ với góc quay ban đầu là 0(rad/s).
- Add transmission cho từng bánh xe trong file urdf và thêm plugin di chuyển trong gazebo:

```
!-- Transmission for left_joint -->
 <transmission name="left_wheel_trans">
   <type>transmission_interface/SimpleTransmission</type>
   <joint name="left_joint">
```

```
<hardwareInterface>hardware_interface/VelocityJointInterface</hardwareInterface>
    </joint>
    <actuator name="left_wheel_motor">
      <mechanicalReduction>1</mechanicalReduction>
      <!--
<hardwareInterface>hardware_interface/VelocityJointInterface</hardwareInterface> -->
    </actuator>
  </transmission>

  <!-- Transmission for right_joint -->
  <transmission name="right_wheel_trans">
    <type>transmission_interface/SimpleTransmission</type>
    <joint name="right_joint">

<hardwareInterface>hardware_interface/VelocityJointInterface</hardwareInterface>
    </joint>
    <actuator name="right_wheel_motor">
      <mechanicalReduction>1</mechanicalReduction>
      <!--
<hardwareInterface>hardware_interface/VelocityJointInterface</hardwareInterface> -->
    </actuator>
  </transmission>

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

-> Plugin gazebo_ros_control trong Gazebo kết nối hệ thống điều khiển ROS với Gazebo, cho phép điều khiển robot qua các controller như JointEffortController, JointPositionController và JointVelocityController. Nó tạo các topic như /robot/joint_states và /robot/controller_manager, giúp ROS gửi lệnh điều khiển tới các joint robot. Ngoài ra, plugin này còn hỗ

trợ mô phỏng động học, bao gồm lực, vận tốc và vị trí của các khớp khi điều khiển robot.

--> Vid điều khiển robot qua bàn phím: IMG_0735.MOV

3.4. Điều khiển 2 khớp của cánh tay máy

- Chạy lệnh: **rosrun test1 arm1.py**

(test1: tên pkg, arm1.py: file code điều khiển 2 links)

- Cấp quyền qua lệnh chmod +x
- Code arm1.py:

```
#!/usr/bin/env python3
import rospy
from std_msgs.msg import Float64
from pynput.keyboard import Listener, Key

class ArmController:
    def __init__(self):
        # Khởi tạo ROS node
        rospy.init_node('arm_keyboard_controller', anonymous=True)

        # Tạo publisher cho mỗi khớp
        self.prismatic_pub =
rospy.Publisher('/prismatic_controller/command', Float64,
queue_size=10)
        self.rotation_pub =
rospy.Publisher('/rotation_controller/command', Float64,
queue_size=10)

        # Thiết lập vận tốc ban đầu
        self.linear_vel = 0.05  # m/s cho khớp tịnh tiến
        self.angular_vel = 1.0  # rad/s cho khớp xoay

        # Vận tốc ban đầu
        self.prismatic_vel = 0.0
        self.rotation_vel = 0.0

        # Tần số vòng lặp
        self.rate = rospy.Rate(10)  # 10 Hz
```

```
        # Khởi động listener cho bàn phím
        self.listener = Listener(on_press=self.on_press,
on_release=self.on_release)
        self.listener.start()

    def on_press(self, key):
        """Xử lý khi nhấn phím"""
        try:
            if key.char == 'w':  # Di chuyển lên
                self.prismatic_vel = self.linear_vel
            elif key.char == 's':  # Di chuyển xuống
                self.prismatic_vel = -self.linear_vel
            elif key.char == 'a':  # Xoay trái
                self.rotation_vel = self.angular_vel
            elif key.char == 'd':  # Xoay phải
                self.rotation_vel = -self.angular_vel
            elif key.char == 'q':  # Tăng tốc độ lên/xuống
                self.linear_vel += 0.01
                print(f"Tốc độ tịnh tiến: {self.linear_vel}")
            elif key.char == 'z':  # Giảm tốc độ lên/xuống
                self.linear_vel = max(0.01, self.linear_vel - 0.01)
                print(f"Tốc độ tịnh tiến: {self.linear_vel}")
            elif key.char == 'e':  # Tăng tốc độ xoay
                self.angular_vel += 0.1
                print(f"Tốc độ xoay: {self.angular_vel}")
            elif key.char == 'c':  # Giảm tốc độ xoay
                self.angular_vel = max(0.1, self.angular_vel - 0.1)
                print(f"Tốc độ xoay: {self.angular_vel}")
            elif key.char == 'f':  # Thoát chương trình
                rospy.signal_shutdown("User requested shutdown")
        except AttributeError:
            pass

    def on_release(self, key):
        """Xử lý khi nhả  phím"""
        try:
            if key.char in ['w', 's']:
                self.prismatic_vel = 0.0
            elif key.char in ['a', 'd']:
                self.rotation_vel = 0.0
        except AttributeError:
            pass
```

```
    def control_loop(self):
        """Vòng lặp chính điều khiển cánh tay robot"""
        print("Keyboard Control for Robotic Arm")
        print("W: Lên, S: Xuống, A: Quay trái, D: Quay phải")
        print("Q/Z: Tăng/Giảm tốc độ tịnh tiến, E/C: Tăng/Giảm tốc độ
xoay")
        print("F: Thoát chương trình")

        while not rospy.is_shutdown():
            # Gửi lệnh vận tốc
            self.prismatic_pub.publish(self.prismatic_vel)
            self.rotation_pub.publish(self.rotation_vel)
            self.rate.sleep()

if __name__ == '__main__':
    try:
        controller = ArmController()
        controller.control_loop()
    except rospy.ROSInterruptException:
        pass
```

- Khởi tạo node: arm_keyboard_controller
- Publisher cho mỗi khớp: prismatic_controller(khớp tịnh tiến: link1), rotation_controller(khớp xoay: link 2)
- Sử dụng bàn phím để có thể di chuyển các khớp của tay máy với:
w/s: di chuyển khớp 1 lên/xuống
a/d: xoay khớp 2 quay sang trái/phải
q/z: tăng/giảm tốc độ lên xuống của tay khớp 1
e/c: tăng/giảm tốc độ quay của tay khớp 2
f: thoát
- Trước khi di chuyển lên xuống của khớp 1 hoặc quay trái/phải của khớp 2 thì cần phải tăng tốc độ lên/xuống và tốc độ xoay vì setup ban đầu để tốc độ với góc quay ban đầu có giá trị là 0.
- Add transmission cho từng khớp với khớp tịnh tiến và khớp xoay trong file test1.urdf

```
<!-- Transmission for prismatic joint -->
```

```
<transmission name="prismatic_trans">
 <type>transmission_interface/SimpleTransmission</type>
 <joint name="prismatic">

<hardwareInterface>hardware_interface/VelocityJointInterface</hardwa
reInterface>
 </joint>
 <actuator name="prismatic_motor">
   <mechanicalReduction>1</mechanicalReduction>
 </actuator>
</transmission>

<!-- Transmission for rotation joint -->
<transmission name="rotation_trans">
 <type>transmission_interface/SimpleTransmission</type>
 <joint name="rotation">

<hardwareInterface>hardware_interface/VelocityJointInterface</hardwa
reInterface>
 </joint>
 <actuator name="rotation_motor">
   <mechanicalReduction>1</mechanicalReduction>
 </actuator>
</transmission>
```

--> Vid điều khiển tay máy: IMG_0737.MOV

3.5. Config (chứa các file định dạng yaml)

- Controller.yaml: định dạng controller điều khiển robot với 2 bánh trái/phải của xe vi sai, chuyển động của từng tay máy và các thông số pid:

```
joint_state_controller:
 type: "joint_state_controller/JointStateController"
 publish_rate: 50
# Controller to control robot base joints
left_wheel_joint_velocity_controller:
 type: "velocity_controllers/JointVelocityController"
 joint: "left_joint"
 pid: {p: 100.0, i: 0.1, d: 10.0}
```

```
right_wheel_joint_velocity_controller:
 type: "velocity_controllers/JointVelocityController"
 joint: "right_joint"
 pid: {p: 100.0, i: 0.1, d: 10.0}

# Controller for Prismatic Joint
prismatic_controller:
 type: velocity_controllers/JointVelocityController
 joint: prismatic
 pid: {p: 50.0, i: 0.05, d: 5.0}

# Controller for Rotation Joint
rotation_controller:
 type: velocity_controllers/JointVelocityController
 joint: rotation
 pid: {p: 50.0, i: 0.05, d: 5.0}
```

3.6. Launch

- gazebo.launch: file launch mở mô hình trong gazebo bao gồm các phần như khởi động gazebo, load mô tả robot, các thông số của từng bánh, spawn mô hình vào trong gazebo và khởi động bộ điều khiển 2 bánh và tay máy.

```
<launch>
   <!-- Khởi động Gazebo -->
   <include file="$(find gazebo_ros)/launch/empty_world.launch">
       <arg name="world_name" value="$(find
test1)/worlds/empty.world"/>
       <arg name="paused" value="false"/>
       <arg name="use_sim_time" value="false"/>
       <arg name="gui" value="true"/>
       <arg name="headless" value="false"/>
       <arg name="debug" value="false"/>
   </include>

   <!-- Load mô tả robot -->
```

```
    <param name="robot_description" command="$(find xacro)/xacro --
inorder '$(find test1)/urdf/test1.urdf'" />

    <!-- Spawn robot vào Gazebo -->
    <node name="urdf_spawner" pkg="gazebo_ros" type="spawn_model"
respawn="false" output="screen"
          args="-urdf -model test1 -param robot_description"/>

    <!-- Tải thông số wheels -->
    <rosparam command="load" file="$(find
test1)/config/controller.yaml" />
    <!-- Khởi động bộ điều khiển -->
    <node
        name="controller_spawner"
        pkg="controller_manager"
        type="spawner"
        args="joint_state_controller
left_wheel_joint_velocity_controller
right_wheel_joint_velocity_controller prismatic_controller
rotation_controller" />
</launch>
```

- display.launch khởi động rviz

4 Kết quả

--> Video và pkg test1 mô phỏng quá trình trong gazebo/rviz(điều khiển robot di chuyển và điều khiển tay máy qua bàn phím)

Ros_gk_test1

--> Các thư viện/pks cần install: pynput, gazebo-ros, gazebo-ros-control, controller-manager, velocity-controllers, effort-controllers