ĐẠI HỌC QUỐC GIA HÀ NỘI

TRƯỜNG ĐẠI HỌC CÔNG NGHỆ

Họ và Tên: Đỗ Mạnh Đoan

MSSV: 22027542

Lớp : K67E-RE

ĐỀ TÀI: XÂY DỰNG, MÔ PHỎNG VÀ ĐIỀU KHIỂN ROBOT VỚI ROS

BÁO CÁO BÀI TẬP GIỮA KỲ

MÔN HỌC: LẬP TRÌNH ROBOT VỚI ROS

Giảng viên hướng dẫn: Lê Xuân Lực, KS. Dương Văn Tân

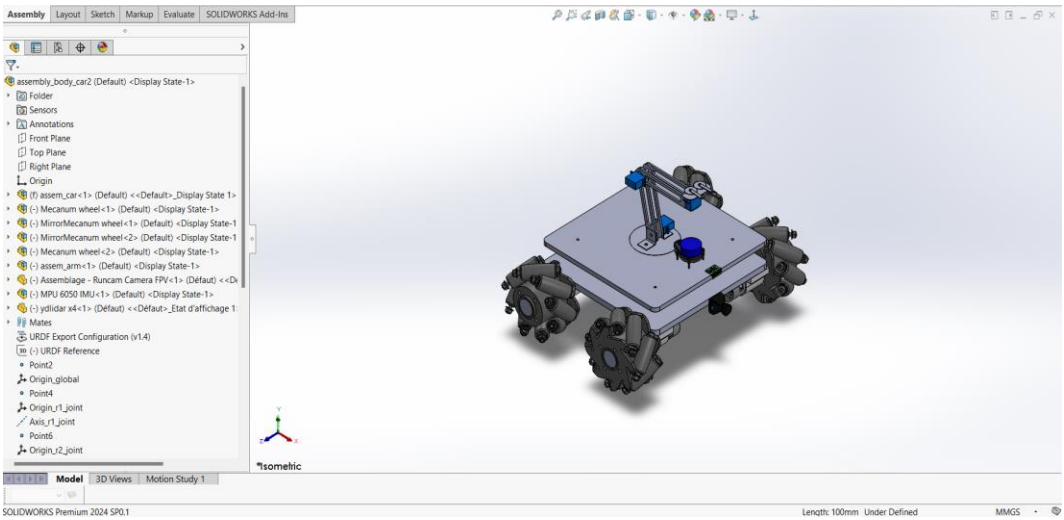
HÀ NỘI - 2025

Lời giới thiệu: Đây là bài tập lớn về thiết kế mô hình, mô phỏng và điều khiển Robot mecanum bốn bánh xe có khả năng di chuyển linh hoạt và trên thân xe có một tay máy với hai góc khớp quay. Robot này được tích hợp thêm các cảm biến Camera, Imu, Lidar để thu thập và hiển thị dữ liệu trong môi trường Gazebo và Rviz.

Nội dung chính:

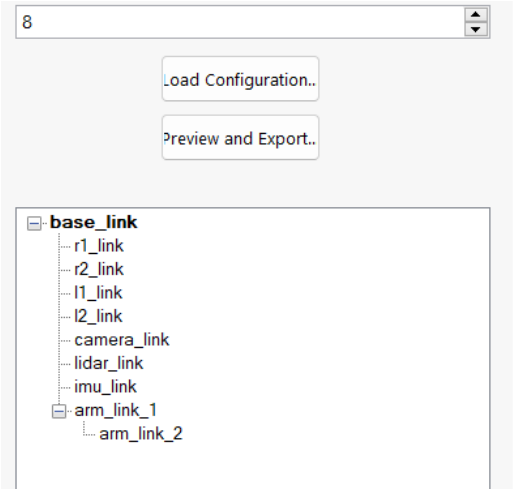
1. Dạng Robot, thiết kế solidwork và cách đặt trục tọa độ

- **Dạng Robot:** Trong bài tập giữa kỳ này, em chọn Robot gồm **4 bánh mecanum**; trên xe gồm tay máy **2 khớp xoay**; cảm biến: **Lidar, Camera và IMU**
  => Dựa vào bài toán trên, em tự thiết kế được Robot cơ bản và tinh gọn gồm các linh kiện như trên
- **Thiết kế solidworks**:



Hình ảnh mô hình Robot 3D Solidworks

+ Robot gồm các phần cơ bản như sau: miếng base dùng để đặt các bộ phận; 4 bánh mecanum, động cơ; giá đỡ động cơ gắn vào miếng base; tay máy 2 khớp xoay và 1 gripper; cảm biến lidar, camera và imu.
+ Cấu trúc link parent và link child như sau:



- **Kích thước xe:** Dài x rộng = 39x27 cm, đường kính bánh xe D=13.5cm

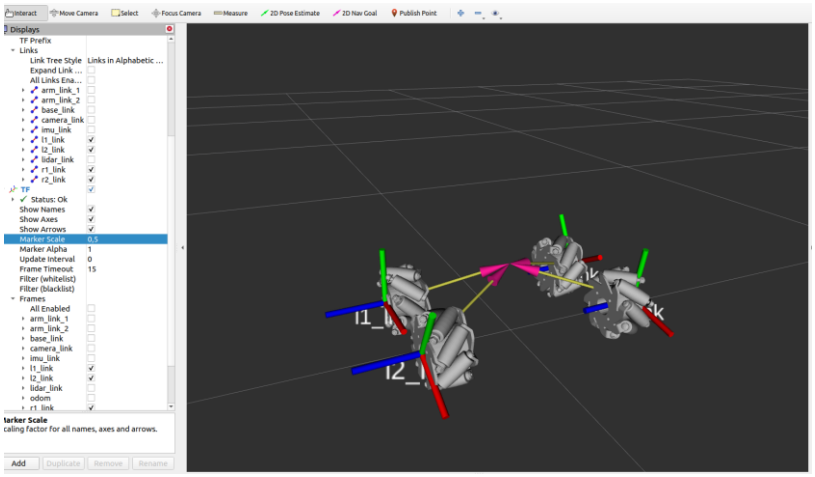
- **Cách đặt trục tọa độ:**

  + Đối với các bánh xe:

  Trục X hướng theo hướng chuyển động

  Trục Z hướng theo trục quay của bánh

  Trục Y theo quy tắc bàn tay phải



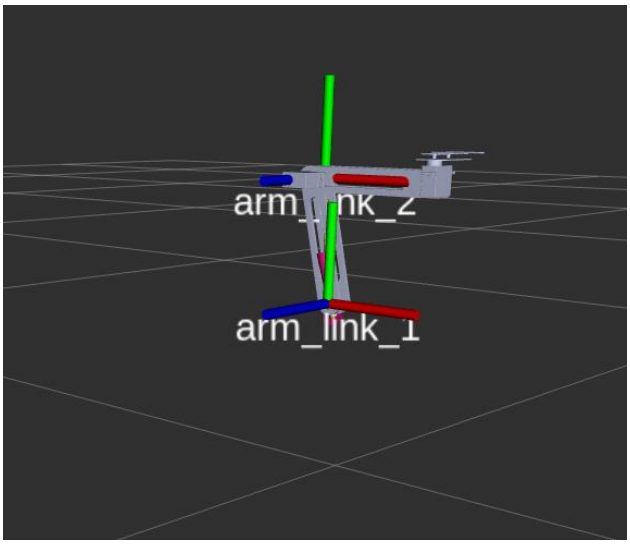
Hình ảnh trục tọa độ bánh xe

  + Đối với tay 2 khớp tay máy:

  Trục X hướng theo hướng bánh xe

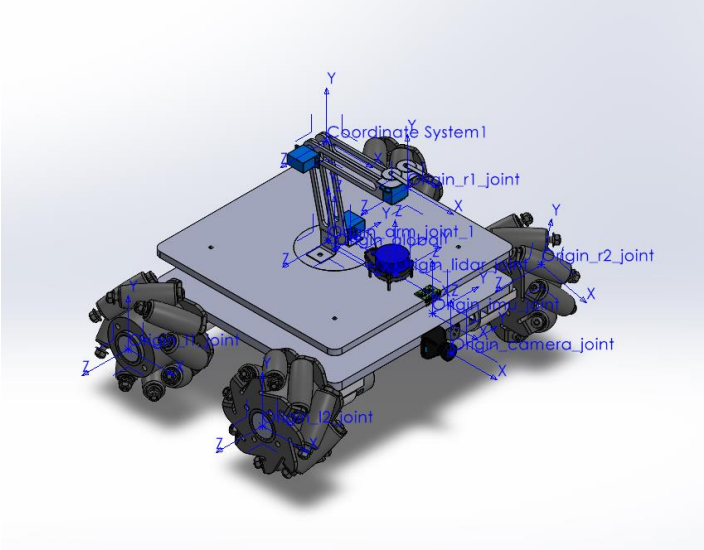
  Trục Z hướng theo trục quay của các khớp

  Trục Y theo quy tắc bàn tay phải



Hình ảnh trục tọa độ tay máy

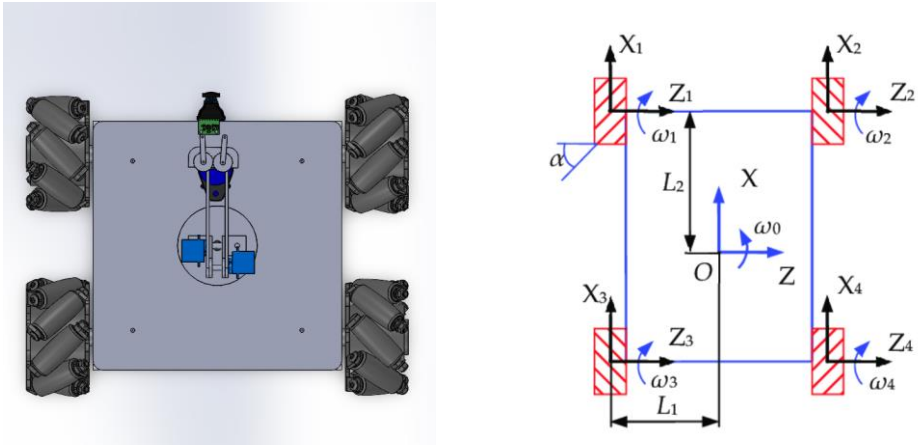
  + Đối với các cảm biến Lidar, Imu, Camera: Tương tự giống như bánh xe và tay máy



Hình ảnh sau khi đặt hết các trục tọa độ cho Robot

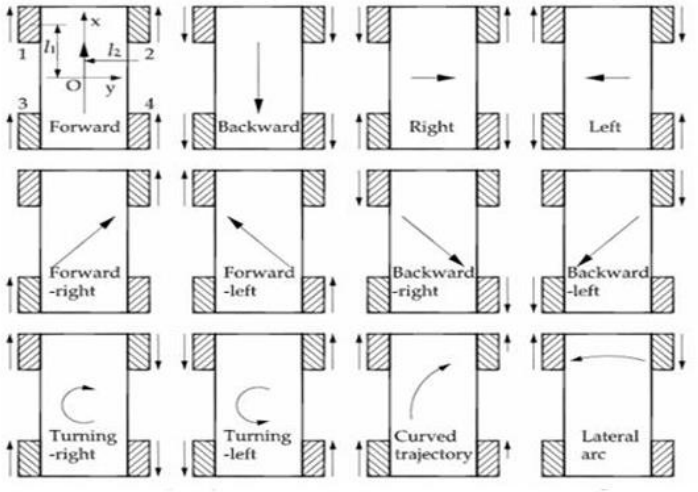
2. Động học Robot, cơ chế chuyển động

2.1: Đối với bánh xe



Hình ảnh Robot và trục tọa độ tương ứng

Mô tả: Bánh xe Mecanum có các con lăn nhỏ đặt chéo 45°. Khi bánh xe mecanum hoạt động, các con lăn trên chu vi bánh xe chuyển một phần lực theo hướng quay của bánh xe với lực pháp tuyến so với hướng của bánh xe.



Chuyển động của Robot dùng bánh mecanum theo hướng và vận tốc góc

Động học thuận: xác định vận tốc của robot dựa trên vận tốc của từng bánh xe

Xét xe mecanum 4 bánh như trên, ta có:

$$\begin{bmatrix} v_{FL} \\ v_{FR} \\ v_{RL} \\ v_{RR} \end{bmatrix} = \frac{1}{R} \begin{bmatrix} 1 & -1 & -(L_x + L_y) \\ 1 & 1 & (L_x + L_y) \\ 1 & 1 & -(L_x + L_y) \\ 1 & -1 & (L_x + L_y) \end{bmatrix} \begin{bmatrix} v_x \\ v_y \\ \omega \end{bmatrix}$$

công thức tính vận tốc bánh xe được xác định bởi hệ phương trình

Trong đó:

- FL- (Front Left): Bánh trước trái
- FR- (Front Right): Bánh trước phải
- RL- (Rear Left): Bánh sau trái
- RR- (Rear Right): Bánh sau phải
- Vx: vận tốc theo trục X
- Vy: vận tốc theo trục Y
- W: Vận tốc góc quanh trục Z
- Lx, Ly: Khoảng cách từ tâm đến bánh xe theo trục X và Y

Động học Nghịch:

Ngược lại, nếu biết vận tốc của từng bánh, ta có thể tính vận tốc robot:

$$\begin{bmatrix} v_x \\ v_y \\ \omega \end{bmatrix} = \frac{R}{4} \begin{bmatrix} 1 & 1 & 1 & 1 \\ -1 & 1 & 1 & -1 \\ -\frac{1}{L_x+L_y} & \frac{1}{L_x+L_y} & -\frac{1}{L_x+L_y} & \frac{1}{L_x+L_y} \end{bmatrix} \begin{bmatrix} v_{FL} \\ v_{FR} \\ v_{RL} \\ v_{RR} \end{bmatrix}$$

2.2: Đối với tay máy

Mô tả: Sử dụng tay máy gồm 2 khớp xoay (Joint1 và Joint2) có thể xoay linh hoạt tới vị trí của vật

Động học thuận:
Với chiều dài các đoạn tay máy là L1 và L2, tọa độ EF được xác định như sau:

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

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

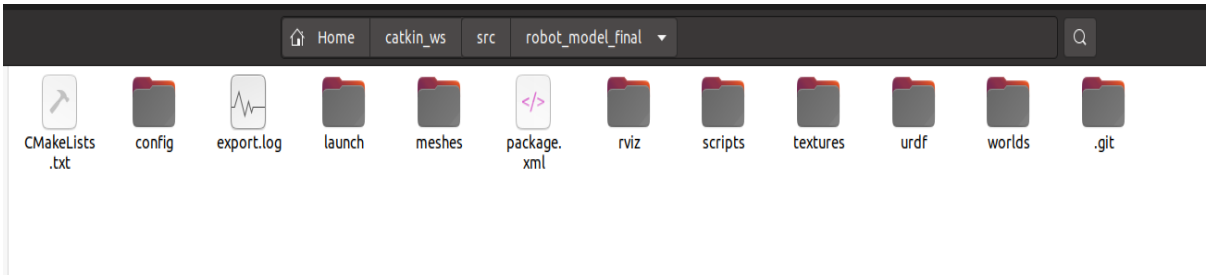
Động học nghịch: Tính toán góc quay cần thiết từ tọa độ mong muốn

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

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

3. Các thành phần chính của code, structure folder dự án

=> Sau khi convert sang urdf từ solidworks ta có:



Hình ảnh các files và folders trong package

Một vài file chính:

- **CMakeLists.txt:** Là file cấu hình để biên dịch package bằng catkin_make (gồm các định nghĩa về thư viện, dependencies, scripts cần build
- **Config:** Chứa các file cấu hình YAML cho bộ điều khiển và Robot
- **Launch:** Chứa các file .launch để chạy mô phỏng.
  + gazebo.launch: Mở Gazebo và Load model Robot từ file urdf
  + rviz.launch: Mở Rviz để hiển thị Robot, điều khiển gui state
  + control.launch : Mở rviz, gazebo, bộ điều khiển
- **Meshes** Chứa các mô hình 3D của Robot (file .STL gồm các link Robot sau khi xuất từ Solidworks sang URDF). Dùng để hiển thị hình dạng Robot trong rviz và gazebo

- **Package.xml:** Chứa metadata của package (tên package, dependencies, tác giả...). Ros dùng file này để xác định các thư viện cần cài
- **Rviz/mecanum.rviz:** lưu lại config của Robot trong rviz mà không cần add topic lại từ đầu
- **Scripts/:** Dùng để lưu trữ các file biên dịch python để điều khiển Robot
  + arm_Robot_keyboard.py: Điều khiển tay máy dùng keyboard
  + arm_control.py: Node publisher để điều khiển tay máy
  + imu.py: hiển thị giá trị cảm biến IMU
  + mecanum_control.py: Node điều khiển bánh xe mecanum
- **Urdf:** Chứa các file URDF mô tả cấu trúc Robot: gồm các link, joints, plugin, transmission ....
- **Worlds**: chứa file custom world trong gazebo tự tạo

4. Mô tả các files chính trong dự án

- **Đối với file URDF**

  + robot name = **"robot_model_final"**: là tên pkg này
  + link name = **"base_link"**: link này là parent link để kết nối các child_link
    <inertial>: Khai báo ma trận quán tính của base_link
    <mass>: Khối lượng (kg)
    <visual>: hiển thị 3D trong gazebo/rviz (file:meshes/base_link.STL
    <rgba>: vector màu<collision>: Xác định va chạm

  + **Lần lượt sau đó là 4 link_name**: "r1_link, r2_link, l1_link, l2_link" tương ứng với 4 link bánh xe mecanum. Các phần <intertial>,<visual>, <collision> cấu trúc tương tự như base_link

  + Khớp nối "joint" tương ứng của nó là "r1_joint", "r2_joint", "l1_joint", "l2_joint" là các khớp nối của bánh xe
    type = "continuos": thể loại khớp xoay vô hạn
    <origin>: vị trí của khớp so với base_link
    <parent> & <child>: liên kết giữa parent link và child link
    <axis xyz>: khớp joint quay quanh trục

  + Lần lượt sau đó là các link và joint cảm biến liên kết với base_link : "camera_link, imu_link, lidar_link" tương ứng với "camera_joint, imu_joint, lidar_joint": Giống với bánh xe tuy nhiên type = "fixed" để chế độ khớp xoay cố định => Các cảm biến không xoay

  + Tiếp đến là 2 link tay máy có link_name = "arm_link_1" và "arm_link_2". "arm_link_1" là child của "base_link" còn arm_link_2 là child của "arm_link_1" và các "arm_joint_1" và "arm_joint_2" tương ứng.
  Cấu trúc tương tự giống với bánh xe.

  Tuy nhiên ở joint của tay máy thì **type = "revolution"** -> Khớp xoay theo góc có giới hạn phạm vi xoay và động học của khớp qua **<limit > <dynamics>**
  + Tiếp đến là phần add **plugin** cho các sensor, bộ điều khiển và add **transmission** cho các khớp của Robot.

- **Giải thích phần transmission:** Transmission giúp ánh xạ giữa bộ điều khiển và (actuator, joint).ROS sử dụng **transmission_interface** để xác định cách động cơ (actuator) tác động lên khớp (joint)

  + **Truyền động** cho khớp arm_joint_1 của tay máy

```
<transmission name="arm_1_joint_transmission">
  <type>transmission_interface/SimpleTransmission</type>
  <joint name="arm_joint_1">
    <hardwareInterface>hardware_interface/PositionJointInterface</hardwareInterface>
  </joint>
  <actuator name="arm_motor_1">
    <mechanicalReduction>1.0</mechanicalReduction>
  </actuator>
</transmission>
```

  transmission_interface/SimpleTransmission: Mô hình truyền động đơn giản
  hardware_interface/PositionJointInterface: Điều khiển vị trí của khớp
  mechanicalReduction = 1.0: Không có tỉ số truyền, động cơ quay bao nhiêu thì khớp quay bấy nhiêu

  Làm tương tự với arm_joint_2 của tay máy

  + **Truyền động** cho khớp **l1_joint** của bánh xe

```
<!-- Add transmission cho 2 khớp tay máy và 4 khớp bánh xe-->

<transmission name="wheel_l1_trasmission">
  <type>transmission_interface/SimpleTransmission</type>
  <joint name="l1_joint">
    <hardwareInterface>hardware_interface/VelocityJointInterface</hardwareInterface>
  </joint>
  <actuator name="l1_motor">
    <mechanicalReduction>1</mechanicalReduction>
  </actuator>
</transmission>
```

  transmission_interface/SimpleTransmission: Mô hình truyền động đơn giản
  hardware_interface/VelocityJointInterface: Điều khiển tốc độ của khớp
  mechanicalReduction = 1.0: Không có tỉ số truyền, truyền vận tốc bao nhiêu thì khớp quay bấy nhiêu

  Làm tương tự đối với các khớp bánh xe còn lại

- **Đối với file YAML:** cấu hình bộ điều khiển cho bánh xe và tay máy **"mecanum_controller.yaml"**

src > robot_model_final > config > ! mecanum_controllers.yaml

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

wheel_1_joint_controller:
  type: "velocity_controllers/JointVelocityController"
  joint: "l1_joint"
  # pid: {p: 50.0, i: 0.1, d: 1.0}

wheel_2_joint_controller:
  type: "velocity_controllers/JointVelocityController"
  joint: "l2_joint"
  # pid: {p: 50.0, i: 0.1, d: 1.0}

wheel_3_joint_controller:
  type: "velocity_controllers/JointVelocityController"
  joint: "r1_joint"
  # pid: {p: 50.0, i: 0.1, d: 1.0}

wheel_4_joint_controller:
  type: "velocity_controllers/JointVelocityController"
  joint: "r2_joint"
  # pid: {p: 50.0, i: 0.1, d: 1.0}

arm_1_joint_controller:
  type: "position_controllers/JointPositionController"
  joint: "arm_joint_1"
  # pid: {p: 50.0, i: 0.01, d: 0.5}

arm_2_joint_controller:
  type: "position_controllers/JointPositionController"
  joint: "arm_joint_2"
  # pid: {p: 50.0, i: 0.01, d: 0.5}
```

+ **“joint_state_controller”** xuất thông tin các khớp với tần số 50Hz
+ Bộ điều khiển bánh xe **“velocity_controllers”**: dùng để điều khiển tốc độ góc của 4 bánh xe. **“JointVelocityController”** điều kiển tốc độ
+ Bộ điều khiển tay máy **“position_controllers”**: Điều khiển vị trí góc quay của các khớp tương ứng. **“JointPositionController”** điều khiển vị trí

- **Đối với file launch:**

src > robot_model_final > launch > control.launch > launch

```xml
<launch>
    <!-- Load Gazebo with Custom World -->
    <include file="$(find gazebo_ros)/launch/empty_world.launch">
        <arg name="world_name" value="$(find robot_model_final)/worlds/create_world.world" />
        <arg name="use_sim_time" value="true" />
    </include>

    <!-- Load Robot Model -->
    <param name="robot_description"
        command="$(find xacro)/xacro '$(find robot_model_final)/urdf/robot_model_final.urdf'" />
    <!-- Load display.launch -->
    <include file="$(find robot_model_final)/launch/display.launch" />
    <!-- Load Controllers -->
    <rosparam file="$(find robot_model_final)/config/mecanum_controllers.yaml" command="load" />
    <node name="controller_spawner" pkg="controller_manager" type="spawner"
        args="joint_state_controller arm_1_joint_controller arm_2_joint_controller
            wheel_1_joint_controller wheel_2_joint_controller wheel_3_joint_controller wheel_4_joint_controller" />
</launch>
```

+ Đầu tiên load gazebo với custom world tạo trong **worlds/create_world.world**
+ Sau đó load cấu hình Robot từ **/urdf/robot_model_final.urdf**
+ Mở rviz thông qua file launch display.launch (file này sẽ lưu các config rviz ở **/rviz/mecanum.rviz**)
+ **Load** bộ controllers trong **mecanum_controllers.yaml** để điều khiển

5. Mô tả cơ chế điều khiển

5.1: Điều khiển các khớp tay máy

Cài đặt: 2 khớp tay máy ở chế độ “revolution” (chế độ xoay), khi đó em giới hạn các khớp của tay máy như sau:

```
<parent
  link="base_link" />
<child
  link="arm_link_1" />
<axis
  xyz="0 0 -1" />
<limit
  lower="-0.35"
  upper="0.35"
  effort="100"
  velocity="10" />
<dynamics
  damping="0.01"
  friction="0.01" />
```

Giới hạn và động học arm_link_1

(Giới hạn này có thể tự do điều chỉnh tùy vào workspace và mục đích)
=> Như vậy giới hạn 2 khớp **arm_joint_2** và **arm_joint_2** là **(-0.35; 0.35) rad**

Viết Node điều khiển tay máy:

Node arm_control.py

src > robot_model_final > scripts > arm_control.py > ...

```python
#!/usr/bin/env python3
import rospy
from std_msgs.msg import Float64
import time

def move_arms():
    rospy.init_node("arm_control", anonymous=True)
    # Publisher cho 2 joints cua arm
    arm_joint_1_pub = rospy.Publisher("/arm_1_joint_controller/command", Float64, queue_size=10)
    arm_joint_2_pub = rospy.Publisher("/arm_2_joint_controller/command", Float64, queue_size=10)
    time.sleep(1)
    rate = rospy.Rate(1)

    while not rospy.is_shutdown():
        #  -0.35 và 0.35 do limit trong urdf -> lower : -0.35 rad; upper: 0.35 rad
        rospy.loginfo("Di chuyen 2 khop ra sau")
        arm_joint_1_pub.publish(-0.35)
        time.sleep(1)
        arm_joint_2_pub.publish(-0.35)
        time.sleep(1)

        rospy.loginfo("Di chuyen 2 khop len phia truoc")
        arm_joint_1_pub.publish(0.35)
        time.sleep(1)
        arm_joint_2_pub.publish(0.35)
        time.sleep(1)

        rate.sleep()

if __name__ == "__main__":
    try:
        move_arms()
    except rospy.ROSInterruptException:
        pass
```

- Khởi tạo node ROS "**arm_control**" và tạo hai **publisher tới arm_1_joint và arm_2_joint** thông qua bộ controller để gửi lệnh điều khiển đến các khớp (gửi vô hạn cho đến khi thoát thì thôi. Lệnh gửi:

- Đưa hai khớp ra sau (-0.35 rad) và đưa hai khớp ra trước (0.35 rad)
- Mỗi lần di chuyển chờ 1s để đảm bảo tay máy có thời gian phản ứng

Node arm_control_keyboard.py:

src > robot_model_final > scripts > arm_control_keyboard.py > move_arms_keyboard

```python
def move_arms_keyboard():
    rospy.init_node("arm_control_keyboard", anonymous=True)
    # Publisher cho 2 joints của arm
    arm_joint_1_pub = rospy.Publisher("/arm_1_joint_controller/command", Float64, queue_size=10)
    arm_joint_2_pub = rospy.Publisher("/arm_2_joint_controller/command", Float64, queue_size=10)
    rospy.sleep(1)

    # Bien luu trang thai cua 2 goc
    joint_1_angle = 0.0
    joint_2_angle = 0.0

    print("===== Key control =====")
    print("Nhan a va d de dieu khien Joint 1 ")
    print("Nhan q va e de dieu khien Joint 2 ")
    print("Nhan 's' de shutdown")

    while not rospy.is_shutdown():
        key = get_key()
        if key == 'a':
            joint_1_angle = max(joint_1_angle - STEP, -JOINT_LIMIT)
            rospy.loginfo(f"Joint 1: {joint_1_angle:.2f}")
            arm_joint_1_pub.publish(joint_1_angle)

        elif key == 'd':
            joint_1_angle = min(joint_1_angle + STEP, JOINT_LIMIT)
            rospy.loginfo(f"Joint 1: {joint_1_angle:.2f}")
            arm_joint_1_pub.publish(joint_1_angle)

        elif key == 'q':
            joint_2_angle = max(joint_2_angle - STEP, -JOINT_LIMIT)
            rospy.loginfo(f"Joint 2: {joint_2_angle:.2f}")
            arm_joint_2_pub.publish(joint_2_angle)

        elif key == 'e':
            joint_2_angle = min(joint_2_angle + STEP, JOINT_LIMIT)
            rospy.loginfo(f"Joint 2: {joint_2_angle:.2f}")
            arm_joint_2_pub.publish(joint_2_angle)

        elif key == 's':
            print("Shutdown")
            break

        rospy.sleep(0.1)
```

- **Khởi tạo node: "arm_control_keyboard**" và tạo **2 publisher** tới arm_joint_1 và arm_joint_2
- Sau đó, nhận ký tự từ bàn phím các nút “a, d, q, e” : để di chuyển các arm_1_joint_controller và arm_2_joint_controller và nút “s” để dừng. Giới hạn 2 khớp là (-0.35;0.35)rad, biến joint_1_angle và joint_2_angle để lưu góc hiện tại của khớp tay. Mỗi 1 lần tăng/giảm góc là 0.05rad

Kết quả: Mở cmd -> roslaunch Robot_control_model arm_control.py hoặc roslaunch Robot_control_model arm_control_keyboard.py

5.2: Điều khiển các khớp bánh xe

Sử dụng mô hình động học thuận để tính toán vận tốc (trình bày ở mục 2.1)

Viết node điều khiển xe: mecanum_control.py

```python
class MecanumKeyboardControl:
    def __init__(self):
        rospy.init_node('mecanum_keyboard_control', anonymous=True)
        self.pub_wheel_fl = rospy.Publisher('/wheel_4_joint_controller/command', Float64, queue_size=10)
        self.pub_wheel_fr = rospy.Publisher('/wheel_2_joint_controller/command', Float64, queue_size=10)
        self.pub_wheel_rl = rospy.Publisher('/wheel_3_joint_controller/command', Float64, queue_size=10)
        self.pub_wheel_rr = rospy.Publisher('/wheel_1_joint_controller/command', Float64, queue_size=10)

        # toc do
        self.max_speed = 1.0
        self.max_omega = 1.0

        # Chieu dai, rong, bk banh xe
        self.Lx = 0.39
        self.Ly = 0.27
        self.wheel_radius = 0.13
```

```python
    def run(self):
        rospy.loginfo("Mecanum Robot Keyboard Control: w/x (Tien/Lui), a/d (Xoay quanh truc), s (stop)")

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

            vx, vy, omega = 0.0, 0.0, 0.0

            if key == 'w':  # Tien
                vx = -self.max_speed
            elif key == 'x':  # Lui
                vx = self.max_speed
            elif key == 'a':  # Xoay quanh truc tu trai qua phai
                omega = -self.max_omega
            elif key == 'd':  # Xoay quanh truc tu phai qua trai
                omega = self.max_omega
            elif key == 's':  # Stop
                vx, vy, omega = 0.0, 0.0, 0.0
            elif key == '\x03':
                break

            # Vtoc banh xe dua vao dong hoc thuan
            v_fl = (1 / self.wheel_radius) * (vx - vy - (self.Lx + self.Ly) * omega)
            v_fr = (1 / self.wheel_radius) * (vx + vy + (self.Lx + self.Ly) * omega)
            v_rl = (1 / self.wheel_radius) * (vx + vy - (self.Lx + self.Ly) * omega)
            v_rr = (1 / self.wheel_radius) * (vx - vy + (self.Lx + self.Ly) * omega)

            self.pub_wheel_fl.publish(v_fl)
            self.pub_wheel_fr.publish(v_fr)
            self.pub_wheel_rl.publish(v_rl)
            self.pub_wheel_rr.publish(v_rr)

            rospy.loginfo(f"FL: {v_fl:.2f}, FR: {v_fr:.2f}, RL: {v_rl:.2f}, RR: {v_rr:.2f}")
            rospy.sleep(0.1)
```

Đầu tiên, khởi tạo node "mecanum_keyboard_control" và 4 publisher riêng biệt được tạo ra để gửi lệnh điều khiển các joints của bánh xe
Viết hàm get_key() để nhận ký tự từ bàn phím
Khai báo các parameters để điều khiển: max_speed: tốc độ tối đa, **max_omega**: tốc độ quay tối đa và **Lx, Ly, wheel_radius:** là chiều dài, rộng và đường kính của bánh xe
Hàm run(): sử dụng các phím "w,x,a,d,s" để di chuyển (các giá trị vx, vy, omega sẽ được gán tương ứng) -> tính ra được tốc độ của 4 bánh xe thông qua công thức động học thuận
Kết quả: Xe có thể tiến, lùi, xoay quanh trục của nó và dừng lại

6. Mô tả cảm biến (Lidar, Camera, Imu)

Trong mô phỏng Gazebo, các cảm biến đóng vai trò quan trọng trong việc thu thập dữ liệu để Robot hoạt động hiệu quả. Các cảm biến như Camera, Lidar và Imu được tích hợp và hoạt động trong Gazebo thông qua plugin

6.1: Mô tả cảm biến camera

Mô tả: Camera hoạt động giống như một camera thực tế, ghi lại hình ảnh từ môi trường xung quanh và truyền vào ROS dưới dạng topic. Thông số của camera bao gồm góc nhìn, độ phân giải, tầm nhìn ...

Plugin sử dụng: **libgazebo_ros_camera.so**

Phân tích:

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

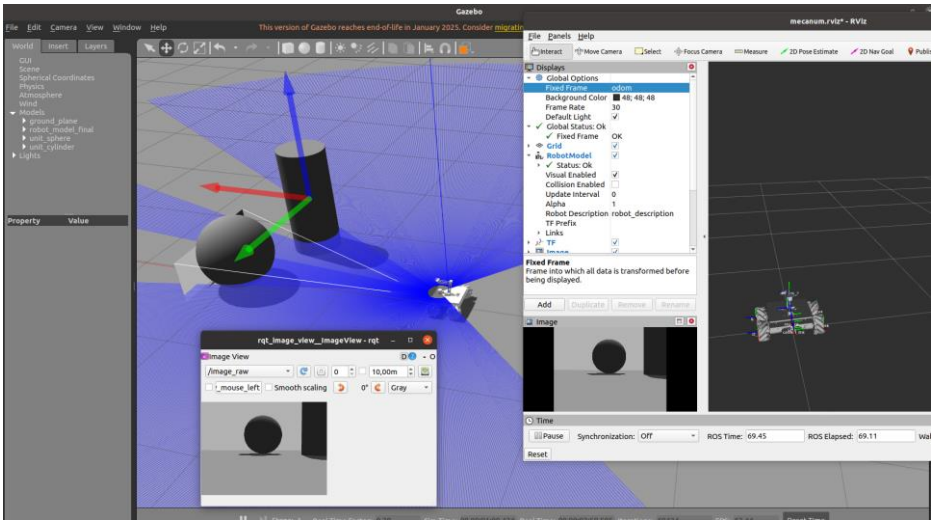
Plugin cho Camera

+ Reference = **“camera_link”**: Camera gắn vào link tên là “camera_link”
+ type="camera": Đây là một cảm biến camera.
+ pose="0 0 0 0 0 0": Camera nằm ngay chính giữa tâm của camera_link
+ visualize=true: Cho phép hiển thị camera trong giao diện Gazebo
+ update_rate=10: Cập nhật dữ liệu hình ảnh 10 lần/giây
+ Các phần ở dưới định dạng: góc nhìn ngang, định dạng màu ảnh RGB 8 bit, độ phân giải 640x480
+ Giới hạn tầm nhìn : phạm vi (0.05 - 8) m

- Dữ liệu được trả về:

-> Xuất hình ảnh theo topic: **/camera1/image_raw (trong rviz: Add -> By topic -> image)**

-> Xem dữ liệu trong ROS: **rqt_image_view**



Hình ảnh dữ liệu của cảm biến Camera

6.2: Mô tả cảm biến Lidar

Mô tả: Cảm biến Lidar dùng tia laser quét xung quanh và đo khoảng cách đến vật cản.Lidar phát ra nhiều tia theo góc quét và trả về tọa độ các điểm

Plugin sử dụng: **libgazebo_ros_laser.so**

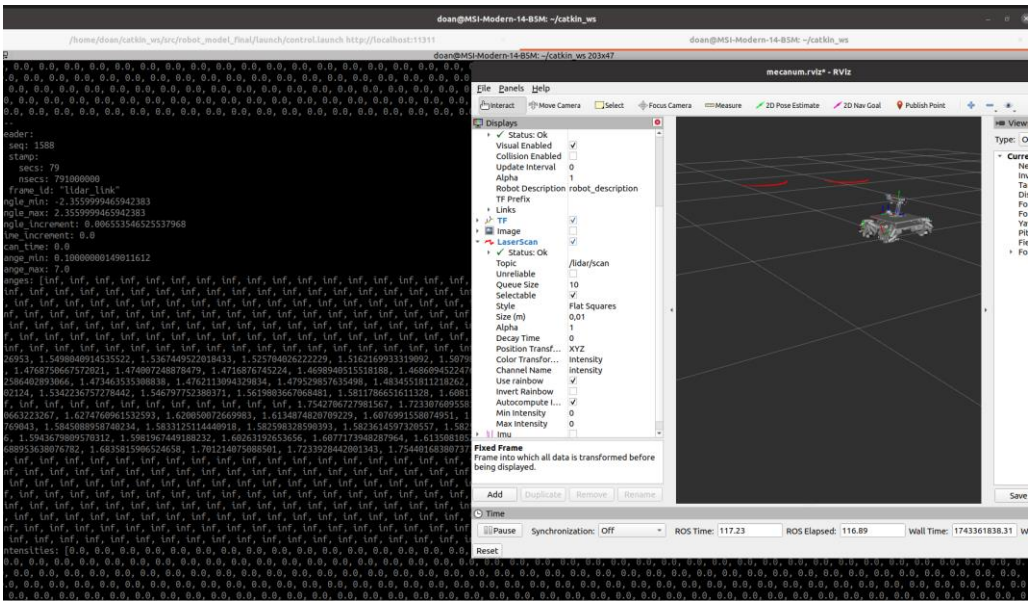
Phân tích:

```
<!-- Lidar plugin-->
<gazebo reference="lidar_link">
  <sensor type="ray" name="lidar_sensor">
    <pose>0 0 0 0 0 0</pose>
    <visualize>true</visualize>
    <update_rate>10</update_rate>
    <ray>
      <scan>
        <horizontal>
          <samples>720</samples>
          <resolution>1</resolution>
          <min_angle>-2.356</min_angle>
          <max_angle>2.356</max_angle>
        </horizontal>
      </scan>
      <range>
        <min>0.1</min>
        <max>7.0</max>
        <resolution>0.01</resolution>
      </range>
    </ray>
    <plugin name="lidar_controller" filename="libgazebo_ros_laser.so">
      <topicName>/lidar/scan</topicName>
      <frameName>lidar_link</frameName>
    </plugin>
  </sensor>
</gazebo>
```

Plugin cho Lidar

+reference = “lidar_link”: Lidar gắn vào link tên là “lidar_link”
+ type=**"ray"**: sử dụng phát tia Laser để đo khoảng cách
+ pose="0 0 0 0 0 0": Vị trí đặt LiDAR trùng với lidar_link
+ update_rate=10: Cập nhật dữ liệu 10 lần/giây
+ samples: 720 tia /lần quét
+ **Giới hạn phạm góc quét: (-135°,135°), khoảng cách quét (0,1;7) m**
+ topicName=/lidar/scan: **Dữ liệu Lidar sẽ được phát trên ROS topic /lidar/scan**

- Dữ liệu được trả về:
    -> Xuất dữ liệu theo topic: **/lidar/scan**
    -> Kiểm tra trong ROS: **rostopic echo /lidar/scan**



Hình ảnh dữ liệu của cảm biến Lidar

6.3: Mô tả cảm biến Imu

Mô tả: Cảm biến IMU đo gia tốc và tốc độ góc theo các trục X, Y, Z, giúp Robot nhận biết độ nghiêng và hướng quay

Plugin sử dụng: **libgazebo_ros_imu_sensor.so**

Phân tích:

```xml
<!-- IMU plugin -->
<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>
      <frameName>imu_link</frameName>
      <initialOrientationAsReference>false</initialOrientationAsReference>
    </plugin>
    <pose>0 0 0 0 0 0</pose>
  </sensor>
</gazebo>
```

Plugin cho IMU

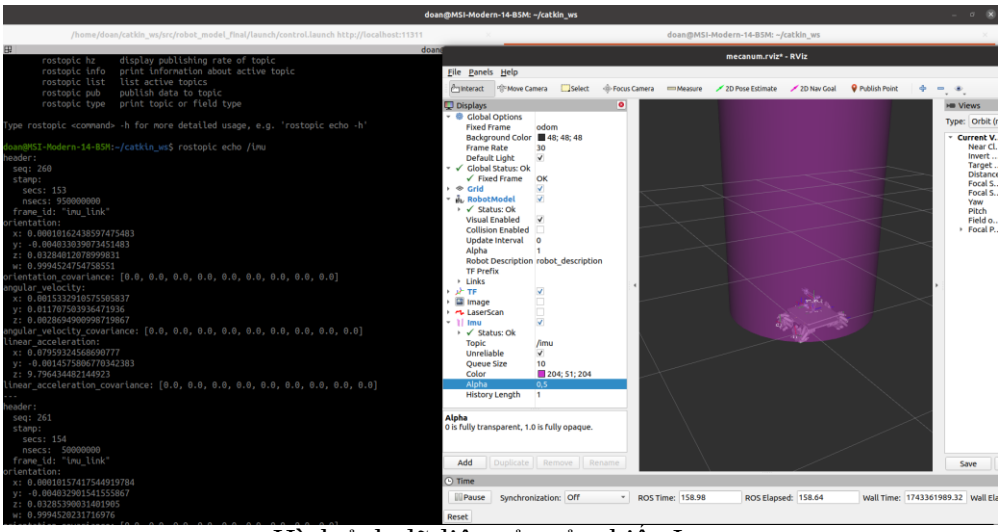
+ reference = "imu_link": Imu gắn vào link tên là "imu_link"
+ type="imu": đo gia tốc và góc quay
+ always_on=true: Luôn bật IMU trong mô phỏng
+ update_rate=10: Cập nhật dữ liệu 10 lần/giây
+ topicName=imu: Dữ liệu IMU sẽ phát trên ROS topic /imu
+ updateRateHZ=10.0: Cập nhật IMU 10 lần/giây

- Dữ liệu trả về:

-> theo topic: **/imu** (rviz: Add -> By topic ->Imu)

-> Kiểm tra trong ROS: **rostopic echo /imu**

(Có thể sử dụng node: **rosrun Robot_model_final imu.py** để theo dõi giá trị cho dễ)

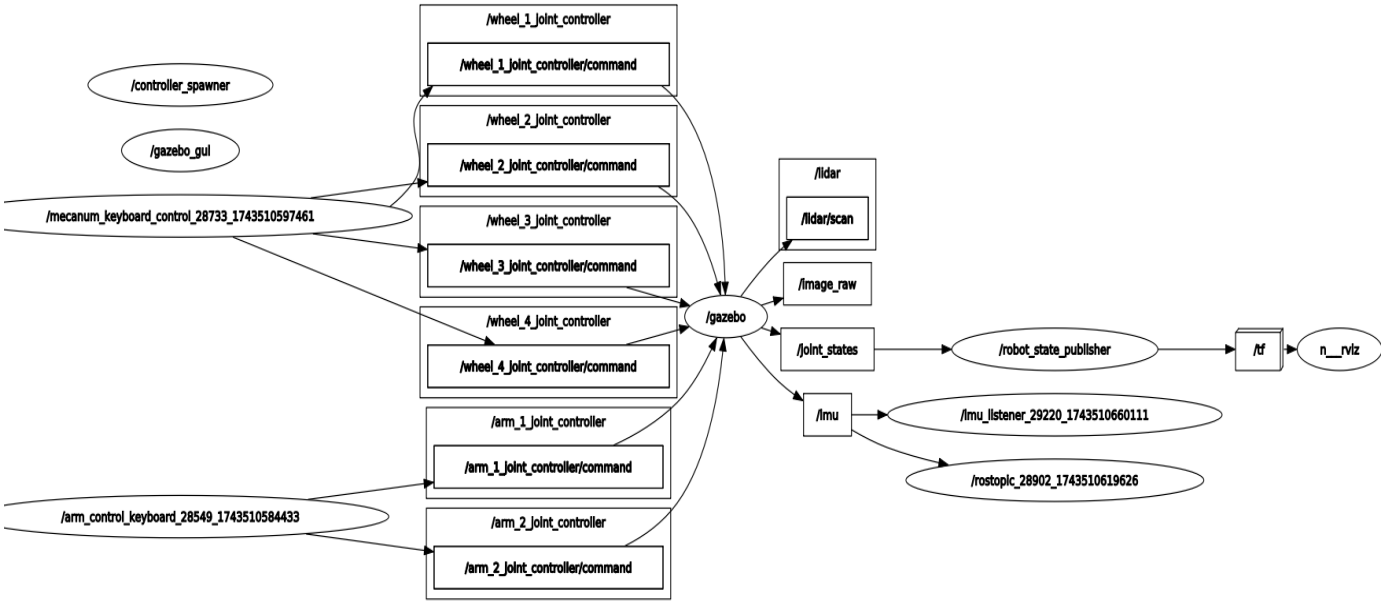


Hình ảnh dữ liệu của cảm biến Imu

7. Cơ chế hoạt động các Gazebo

Chạy các lệnh:

roslaunch robot_model_final control.py
Rosrun robot_model_final arm_control_keyboard.py
Rosrun robot_model_final mecanum_control.py
Rosrun robot_model_final imu.py
Rqt_graph



*** Các Node chính:**

- **Node điều khiển robot:**
  - **/mecanum_keyboard_control_...** Node xử lý điều khiển xe từ bàn phím. Node này gửi lệnh đến các controller của bánh xe
  - **/arm_control_keyboard_...** Node này điều khiển các joints của robot từ bàn phím, gửi lệnh đến controller của khớp cánh tay
- **Node điều khiển gazebo**
  - **/Gazebo ->** Node chính của gazebo, dùng để mô phỏng vật lý, xử lý cảm biến và cập nhật trạng thái robot
  - **/gazebo_gui** → Node này quản lý giao diện Gui của gazebo

*** Các Controller:**

- **Bánh Xe Mecanum:** /wheel_1_joint_controller -> /wheel_4_joint_controller: Mỗi controller bánh xe đều có topic /command để nhận lệnh từ mecanum_keyboard_control
- **Khớp tay Robotg /arm_1_joint_controller và /arm_2_joint_controller: Nhận lệnh từ arm_control_keyboard**

*** Các Topic:**

- **Từ Gazebo:**
  - **/joint_states:** Chứa trạng thái của tất cả khớp
  - **/lidar/scan:** Dữ liệu từ cảm biến Lidar
  - **/image_raw:** Dữ liệu ảnh từ Camera
  - **/imu:** Dữ liệu cảm biến Imu
- **Từ Robot State Publisher:**
  - **/tf:** Thông tin biến đổi khung tọa độ để hiển thị trong rviz

=> Các Node điều khiển gửi lệnh tới Controller của bánh xe và tay máy. Gazebo nhận lệnh từ controller và mô phỏng vật lý. Các dữ liệu cảm biến Lidar, Camera. Imu được gửi ra các topic tương ứng. Robot State Publisher nhận dữ liệu từ joint_states và gửi thông tin tọa độ lên /tf (trong rviz)

TỔNG KẾT

Đối với dự án lần này, em đã hoàn thiện được mô hình 3D hoàn chỉnh cho Robot như yêu cầu; xuất đúng mô hình sang URDF để mô phỏng và hiển thị trong Rviz và Gazebo; di chuyển được các bánh xe bằng bàn phím thông qua phương trình động học và các khớp tay máy; đồng thời hiển thị được các topic dữ liệu cho cảm biến đề ra.

Qua dự án trên, em đã học được rèn luyện rất nhiều các kỹ năng về xây dựng mô hình 3D, tìm hiểu được các kiến thức trong ROS. Em cảm ơn thầy và cảm ơn anh đã giải đáp các thắc mắc của em trong quá trình học tập và làm dự án này