

Báo cáo giữa kỳ ROS

Họ và tên: Nguyễn Đức Khánh Huyền

Lớp: K67-RE

MSV: 22027503

Yêu cầu: Xây dựng Robot gồm có:

+ Robot di chuyển: hai bánh vi sai (***diffirential drive***)

+ Tay máy:

– Khớp 1: Khớp xoay- ***Rotation***

– Khớp 2: Khớp tịnh tiến- ***Prismatic***

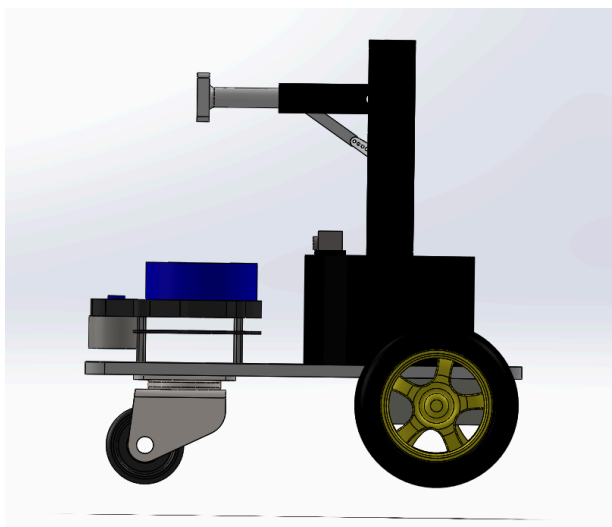
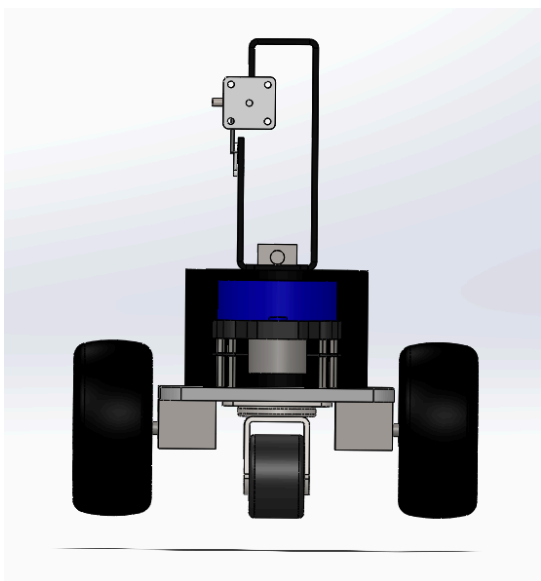
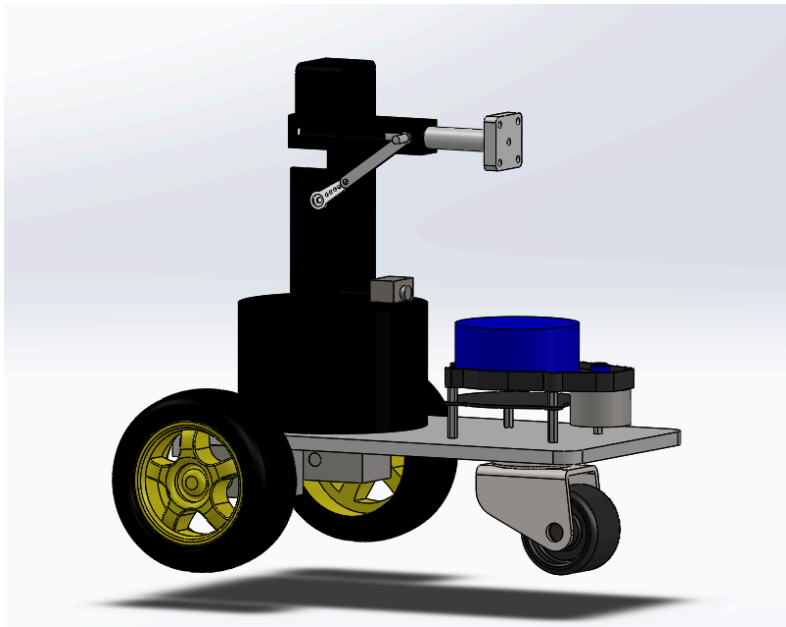
+ Cảm biến: ***LiDAR, Camera*** và ***encoder***

I. Giới thiệu thiết kế Robot

1. Tổng quan

- Robot được thiết kế sử dụng phần mềm Solidworks, gồm ba thành phần chính: Robot di chuyển, Tay máy và Cảm biến, đáp ứng các yêu cầu của đề bài (mô tả trên)

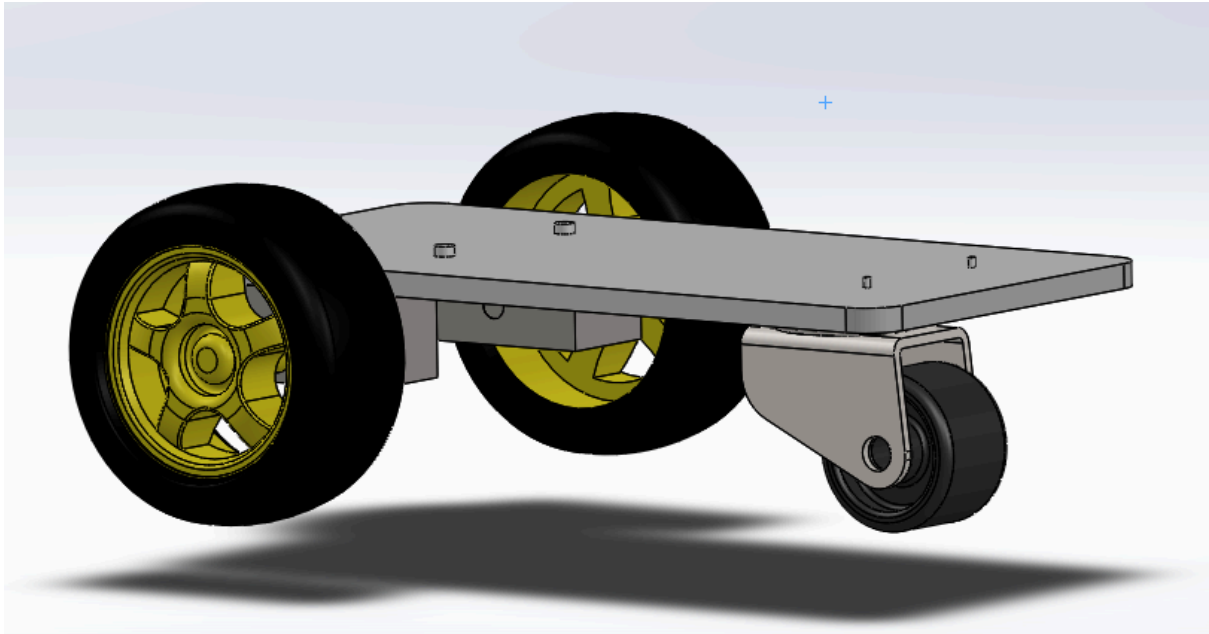
- Mô hình Robot 3D:



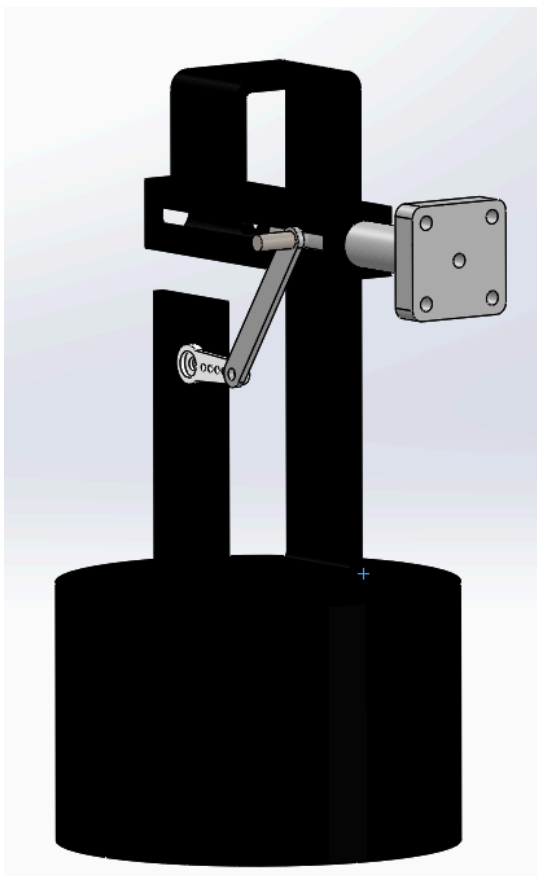
2. Chi tiết và mô tả thiết kế

2.1. Mô hình Robot di chuyển

- Gồm: Đế, 2 motor gắn với động cơ vàng và 1 bánh đĩa hướng



2.2. Mô hình Tay máy (cơ cấu truyền động)



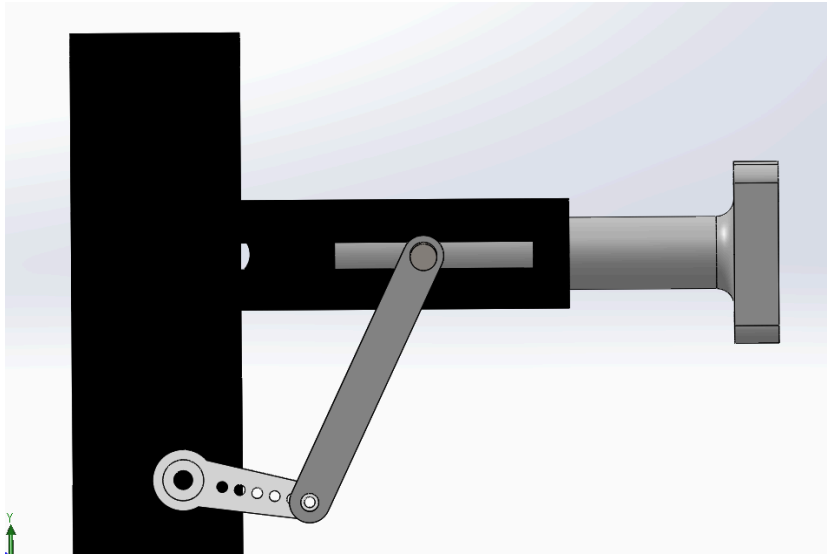
- Tay máy gồm có 2 khớp:

- + Khớp 1: Khớp quay: Đế tay máy, động cơ (đơn giản hóa bằng một hình trụ tròn thể thiện đầu khớp quay)

- + Khớp 2: Khớp quay, được gắn thêm một khớp nối để biến chuyển động quay của motor thành chuyển động tịnh tiến của end-effector.

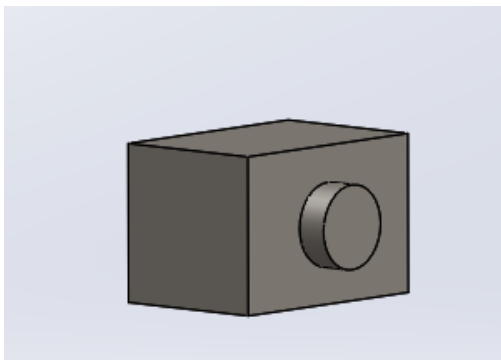
Gồm có một khung đứng đỡ động cơ (lược bỏ để đơn giản thiết kế), một thanh nối, một khung trượt và đỉnh ốc cố định thanh nối và khung trượt với end-effector

- End-effector có dạng là con dấu, được giới hạn trong khoảng trượt [0-3cm]

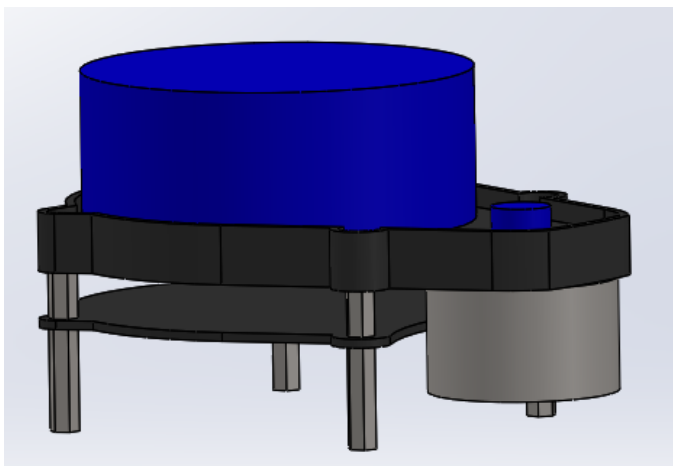


2.3. Cảm biến

- Camera:



- LiDAR:

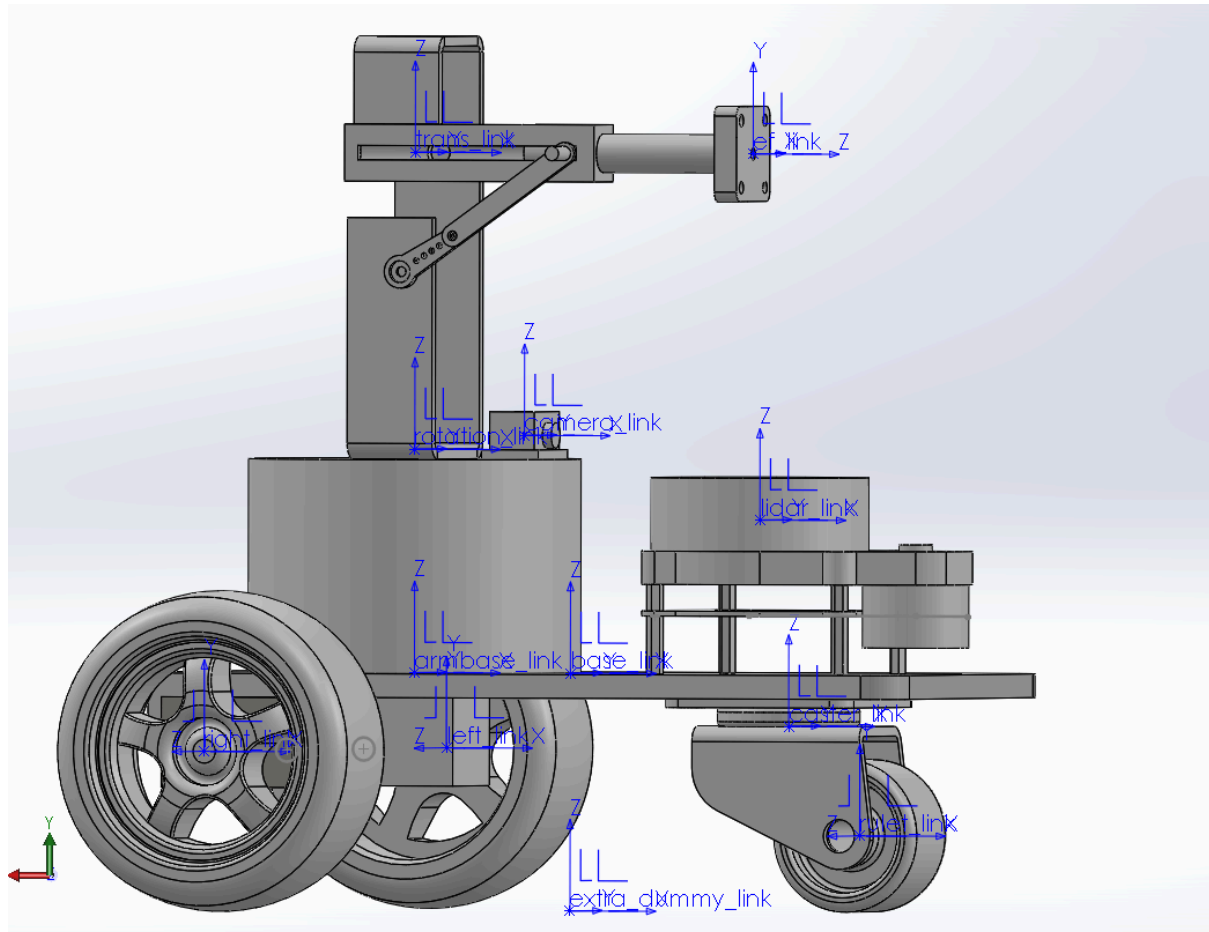


II. Phân tích động học Robot

1. Hệ trục tọa độ

- Mô tả cách đặt các hệ trục tọa độ của Robot (xuất file pdf)
- Hệ trục tọa độ của tay máy được đặt theo quy tắc

Denavit-Hartenberg

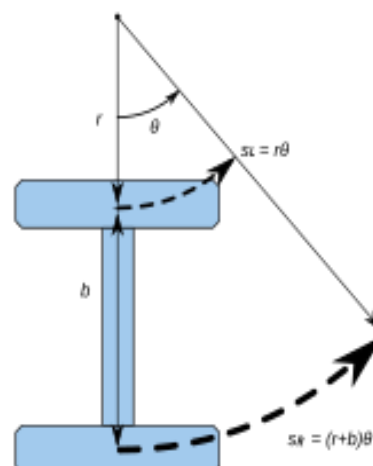


2. Robot di chuyển

- Robot chỉ chuyển được thiết kế theo mô hình xe hai bánh vi sai với một bánh đa hướng giữ thẳng bằng

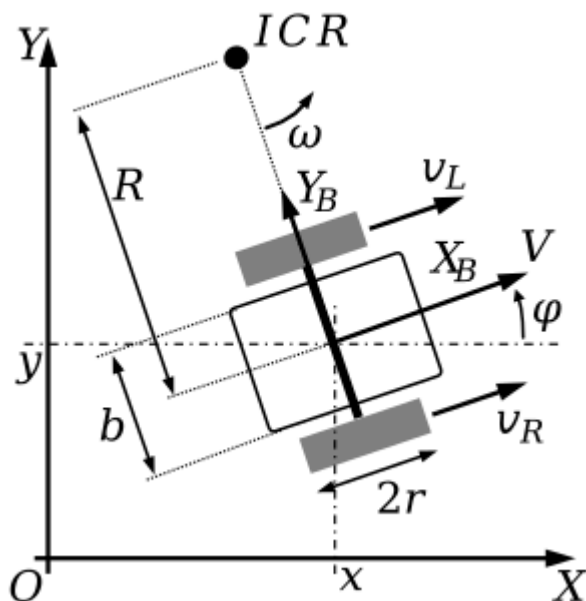
2.1. Cơ chế chuyển động

- Xe hai bánh vi sai chuyển động dựa trên việc điều khiển hai bánh trái/ phải riêng biệt.



- Điều hướng chuyển động của xe bằng cách thay đổi tốc độ quay tương đối của hai bánh:
- + Hai bánh xe cùng di chuyển theo một hướng với tốc độ bằng nhau
-> Xe tịnh tiến theo đường thẳng
- + Hai bánh xe quay khác tốc độ/ hướng
-> Xe quay quanh một điểm trung tâm nằm trên trục bánh

2.2. Mô hình động học



Chú thích:

- Y_B và X_B : vị trí của xe trong hệ tọa độ cố định Oxy
- φ : hướng của Robot trong hệ tọa độ cố định Oxy
- V : vận tốc xe
- V_R, V_L : vận tốc bánh phải, trái
- ω : vận tốc quay xe
- ω_R, ω_L : vận tốc bánh phải, trái
- R, r : bán kính của quỹ đạo xe và bán kính bánh xe
- ICR: tâm quay của quỹ đạo
- b : chiều ngang xe

- Thông qua việc tính toán và phân tích mô hình động học, ta rút ra được phương trình động học của Robot di chuyển như sau:

$$\begin{bmatrix} \dot{x}_B \\ \dot{y}_B \\ \dot{\varphi} \end{bmatrix} = \begin{bmatrix} v x_B \\ v y_B \\ \omega \end{bmatrix} \stackrel{v=r\omega}{=} \begin{bmatrix} \frac{r}{2} & \frac{r}{2} \\ 0 & 0 \\ -\frac{r}{b} & \frac{r}{b} \end{bmatrix} \begin{bmatrix} \omega_L \\ \omega_R \end{bmatrix} \quad (1)$$

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\varphi} \end{bmatrix} = \begin{bmatrix} \cos \varphi & 0 \\ \sin \varphi & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} V \\ \omega \end{bmatrix} \quad (2)$$

*Tham khảo: https://en.wikipedia.org/wiki/Differential_wheeled_robot

2.3. Thông số điều khiển

- Từ (1) và (2), ta có được thông số điều khiển hai bánh xe như sau:

$$\omega_R = \frac{V + \omega \cdot b/2}{r}$$

$$\omega_L = \frac{V - \omega \cdot b/2}{r}$$

Trong mô phỏng Gazebo, thực tế ta sử dụng gói plugin differential_drive_controller (áp dụng các kết quả trên) để tính toán tham số điều khiển truyền vào các động cơ.

3. Tay máy

3.1. Mô hình động học

- Áp dụng quy tắc Denavit-Hartenberg để tính toán động học cho tay máy sau khi đặt các hệ trục tọa độ như sau:

+ Bảng Denavit-Hartenberg:

Joint i	θ_i	d_i	a_i	α_i
1. extra_dummy_link -> base_link	0	L1	0	0
2. base_link -> armbase_link	0	0	-L2	0
3. armbase_link -> rotation_link	θ^*	L3	0	0
4. rotation_link -> trans_link	0	L4	0	0
5. trans_link -> ef_link	$-\pi/2$	d^*	-L5	$\pi/2$

Trong đó: θ^* và d^* là biến điều khiển khớp rotation và prismatic

L1= 0.05m

L2= 0.035m

L3= 0.045m

L4= 0.0645m

L5= 0.01185m

- Ma trận chuyển đổi đồng nhất từ end-effector ef_link so với extra_dummy_link:

$$T_0^5 = \begin{bmatrix} -\sin \theta^* & \cos \theta^* & 0 & -L_2 - L_5 \cos \theta^* \\ -\cos \theta^* & -\sin \theta^* & 0 & L_5 \sin \theta^* \\ 0 & 0 & 1 & L_1 + L_3 + L_4 + d^* \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (3)$$

3.2. Thông số điều khiển

- Bài toán động học thuận:

Từ (3), ta rút ra vị trí của end-effector so với thân tay máy là:

$$p = \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} -L_2 - L_5 \cos \theta^* \\ L_5 \sin \theta^* \\ L_1 + L_3 + L_4 + d^* \end{bmatrix} \quad (4)$$

- Bài toán động học ngược:

Giả sử bài toán đã có $p = [X_e, Y_e, Z_e]$, ta tính toán được các thông số điều khiển θ^* , d^* như sau:

$$\theta^* = \tan^{-1} \left(\frac{Y_e}{-(X_e + L_2)} \right) \quad (5)$$

$$d^* = Z_e - (L_1 + L_3 + L_4) \quad (6)$$

III. Phân tích URDF

1. Định nghĩa

- URDF (Unified Robot Description Format) là một tệp XML mô tả các thuộc tính vật lý quan trọng của Robot: link (liên kết), joint (khớp nối), hình dạng, màu sắc, collision (va chạm),... để mô phỏng Robot trong ROS.

- Một số thuộc tính được sử dụng nhiều trong file URDF:

<link> Mô tả một khâu gồm:

+ <name>: tên link

+ <mass>: khối lượng
+ <visual>: hình dạng hiển thị và <collision>: hình dạng để xác định va chạm trong môi trường vật lý:
<origin xyz="0 0 0" rpy="0 0 0" /> (giữ nguyên vị trí và hướng của mô hình)
<geometry>: mô tả hình dạng, với visual thể hiện bằng file mesh, với collision có thể chọn hình dạng đơn giản hơn
<color>: màu sắc
<material>: vật liệu
<link/>
<joint> Mô tả một khớp gồm
+ <origin>: vị trí và hướng
+ <parent_link>: khâu đầu
+ <child_link>: khâu cuối
+ <axis>: trục quay
<joint/>

2. Cấu trúc

- Để mô tả được các đặc tính của Robot theo yêu cầu, file URDF gồm có 4 phần chính sau đây:

2.1. Các link/ joint thường

- Cấu trúc chung:

<link> <link/>

<joint> <joint/>

(1) extra_dummy_link => thêm vào để sửa lỗi #1.VI, riêng link này chỉ khai báo <name>

(2) base_link => không có joint (Vì ban đầu khi export URDF chọn link này làm reference link)

(3) caster_link và (4) rulet_link => thể hiện bánh đa hướng

2.2. Các link/ joint gắn với động cơ

- Cấu trúc chung:

<link>

<joint>

<transmission> Bộ truyền động kết nối với động cơ và khớp

<name>

<type> transmission_interface/SimpleTransmission, mô phỏng đơn giản, giả thiết không có mất mát hoặc phi tuyến

<joint> Khớp

<name>

<hardwareInterface>

<actuator> Động cơ

<name>

<mechanicalReduction> 1, tỉ số truyền động

(4) left_link và (5) right_link

<joint> Được gắn thêm tag:

+ <initial_velocity>0 -> đảm bảo tại trạng thái khởi tạo Robot không di chuyển

<hardwareInterface>hardware_interface/VelocityJointInterface, bộ điều khiển sẽ gửi tốc độ góc tới khớp

(6) rotation_link và (7) ef_link

<joint> Được gắn thêm tag:

+ <initial_velocity>0-> đảm bảo tại trạng thái khởi tạo Robot không di chuyển

+ <dynamics damping="0.1" friction="0.5"/>: Hệ số giảm chấn damping và hệ số ma sát khớp

+ < limit>

_ lower & upper: Giới hạn chuyển động (rad hoặc m)

_ effort: Momen xoắn/ lực tác dụng lớn nhất

_ velocity: Vận tốc lớn nhất (rad/s hoặc m/s)

<hardwareInterface> hardware_interface/PositionJointInterface, bộ điều khiển sẽ gửi tốc độ góc tới khớp nếu là revolute, còn vị trí đúng nếu là prismatic

2.3. Các plugin điều khiển motor trong gazebo

(8) `Differential Drive Controller` - plugin điều khiển robot vi sai hai bánh, giúp tính toán vận tốc từng bánh V_R , V_L dựa trên vận tốc V của xe.

- Điều khiển vận tốc bằng cách gửi data tới `<commandTopic>cmd_vel</commandTopic>`, xuất giá trị của từng khớp bánh tới `<odometryTopic>odom</odometryTopic>` và `<publishWheelJointState>true</publishWheelJointState>`

(9) `Gazebo ROS Control` - plugin phân tích các bộ truyền động `<transmission>` và tải giao diện phần cứng `<hardwareInterface>` cũng như bộ quản lý điều khiển `<controller_manager>` cần thiết

2.4. Các cảm biến

- Cấu trúc chung:

`<link> </link>`

`<joint> </joint>`

`<gazebo>`

`<sensor>` Khai báo các thông số cảm biến

`<plugin>` Khai báo plugin điều khiển trong gazebo, gồm một số tag quan trọng như: `<frameName>` - khâu gắn với cảm biến, và `<topicName>` topic chứa dữ liệu cảm biến

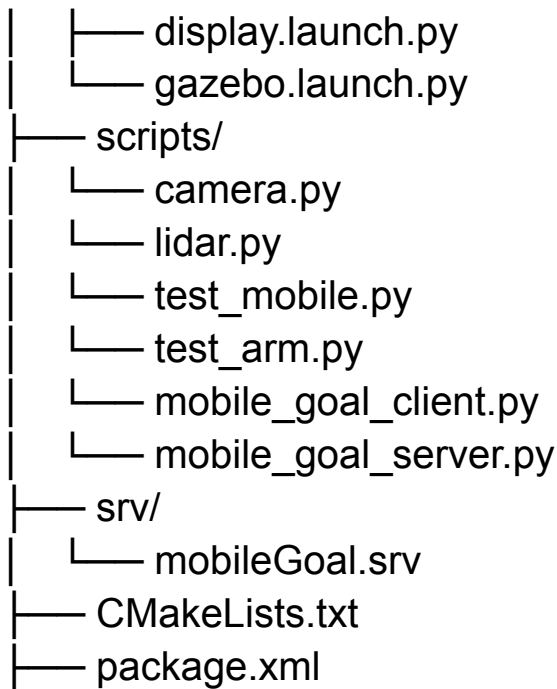
(10) `lidar_link`: với `<collision>`, lidar được định nghĩa `<geometry>` là một hình trụ đơn giản để tránh phức tạp hóa việc mô phỏng

(11) `camera_link`

V. Cấu trúc folder dự án

robot1/

```
|— urdf/
|   |— robot1.urdf
|— meshes/
|— config/
|   |— controller_manager.yaml
|— launch/
```



1. URDF

- Mô tả Robot để mô phỏng trong gazebo

2. Meshes

- Chứa các file 3D của các khâu của Robot

3. Config

- controller_manager.yaml: mô tả các bộ điều khiển trong gazebo, gồm có:
 - + robot_state_publisher
 - + rotation_controller
 - + prismatic_controller

4. Launch

- display.launch: thể hiện model trong rviz, dùng joint_state_publisher để kiểm tra hoạt động của các khớp
- gazebo.launch: khai báo các tham số gazebo và các node sau để chạy cùng lúc các node khi mô phỏng gazebo:
 - + <camera>- xử lý dữ liệu camera
 - + <lidar>- in dữ liệu lidar
 - + <robot_state_publisher>
 - + <robot_description>- tải urdf
 - + <controller_spawner>- tải các bộ điều khiển

- + `<include file="$(find gazebo_ros)/launch/empty_world.launch">`- khởi chạy gazebo

5. Scripts

- camera.py:
 - + Subscribe tới topic robot1/camera1/image_raw => chuyển đổi message ảnh ROS sang dạng message OpenCV dùng package cv_bridge.
 - + In ra màn hình kết quả ảnh camera thu được thông qua một cửa sổ OpenCV.
- lidar.py:
 - + Subscribe tới topic /scan => nhận dữ liệu data và in ra màn hình số điểm quét được, khoảng cách tới điểm đầu tiên
- test_mobile.py: kiểm tra chuyển động của xe bằng cách đẩy dữ liệu tới /cmd_vel
- test_arm.py: kiểm tra hoạt động của tay máy, đẩy dữ liệu tới /joint_states, dùng bộ điều khiển rotation_controller và prismatic_controller
- mobileGoal_server.py và mobileGoal_client.py (thử nghiệm):
 - + Dùng gói service mobileGoal.srv, tạo một node server và client: khi client chạy và nhận được vị trí mong muốn-goal, server sẽ nhận goal và điều khiển robot chạy tới goal bằng việc so sánh giữa vị trí thật của robot (được xử lý bởi encoder và đẩy lên odom/ (plugin Differential Drive Controller)), từ đó tính toán vận tốc di chuyển cho hai bánh

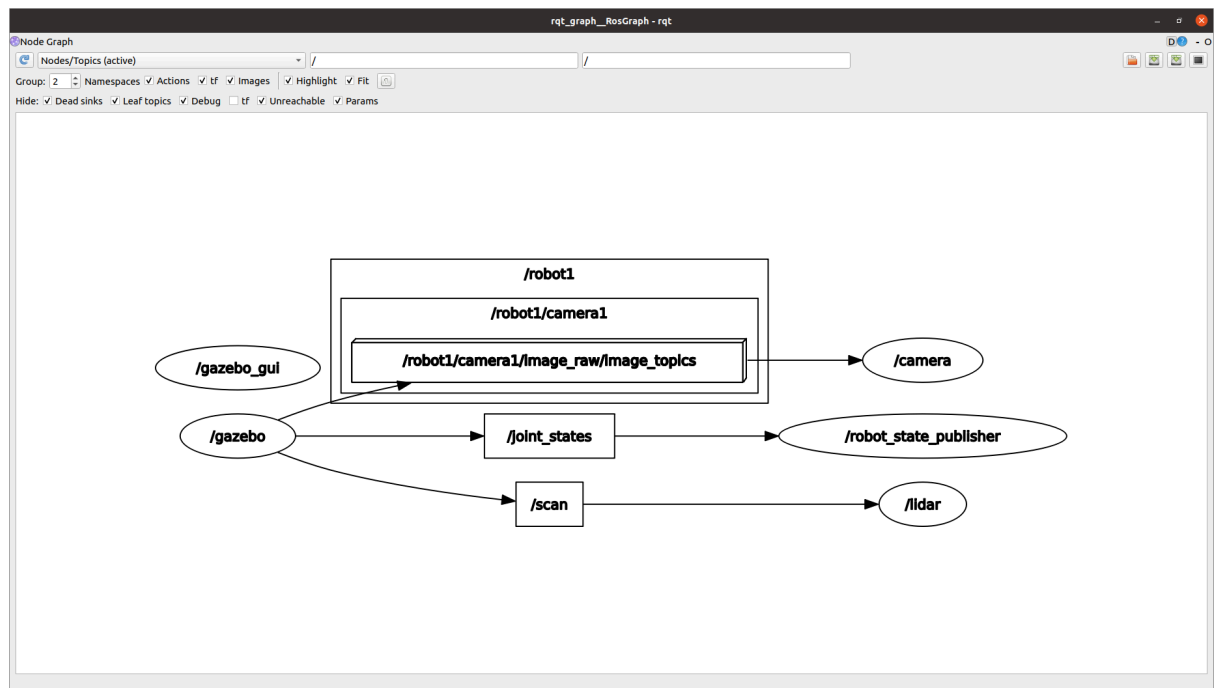
6. Srv

- mobileGoal.srv: service xác định xem robot đã thành công di chuyển tới vị trí mong muốn chưa.
 - + mobileGoal: float32 x, float32 y (vị trí)
 - + mobileGoalResponse: bool success (true -> thành công, false -> thất bại)

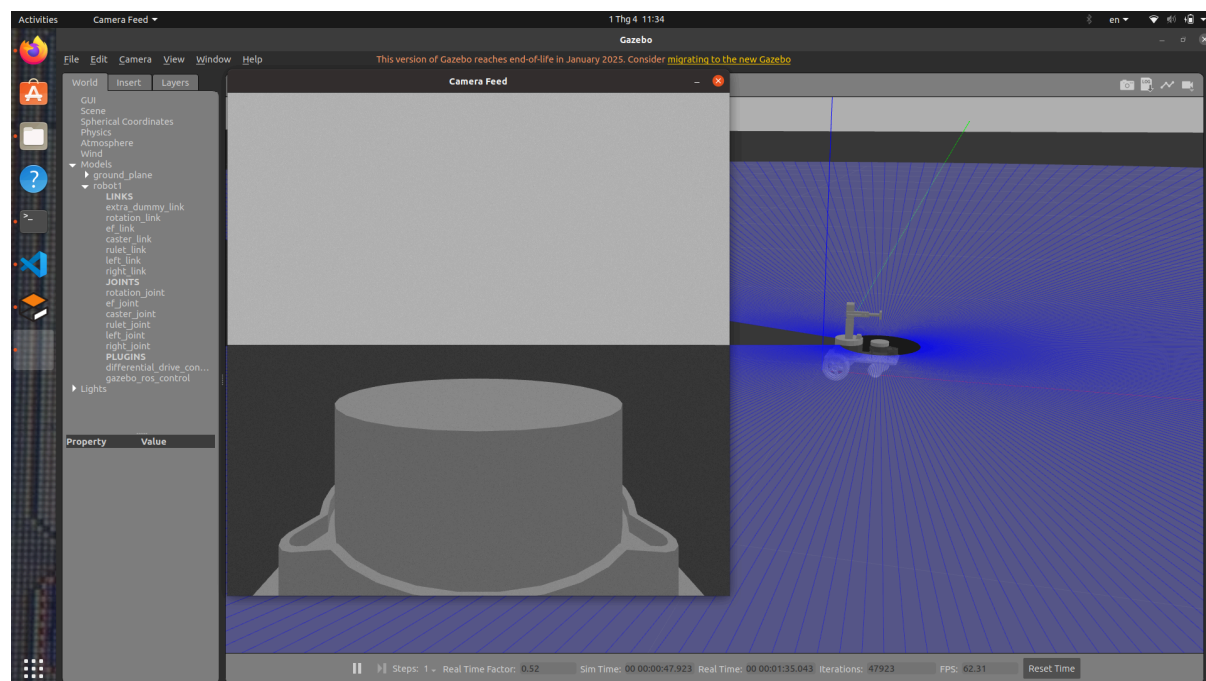
7. CMakeList.txt, package.xml và export.log

IV. Cơ chế điều khiển Gazebo

- Khi khởi chạy gazebo.launch, các node sau sẽ hoạt động (rqt_graph)



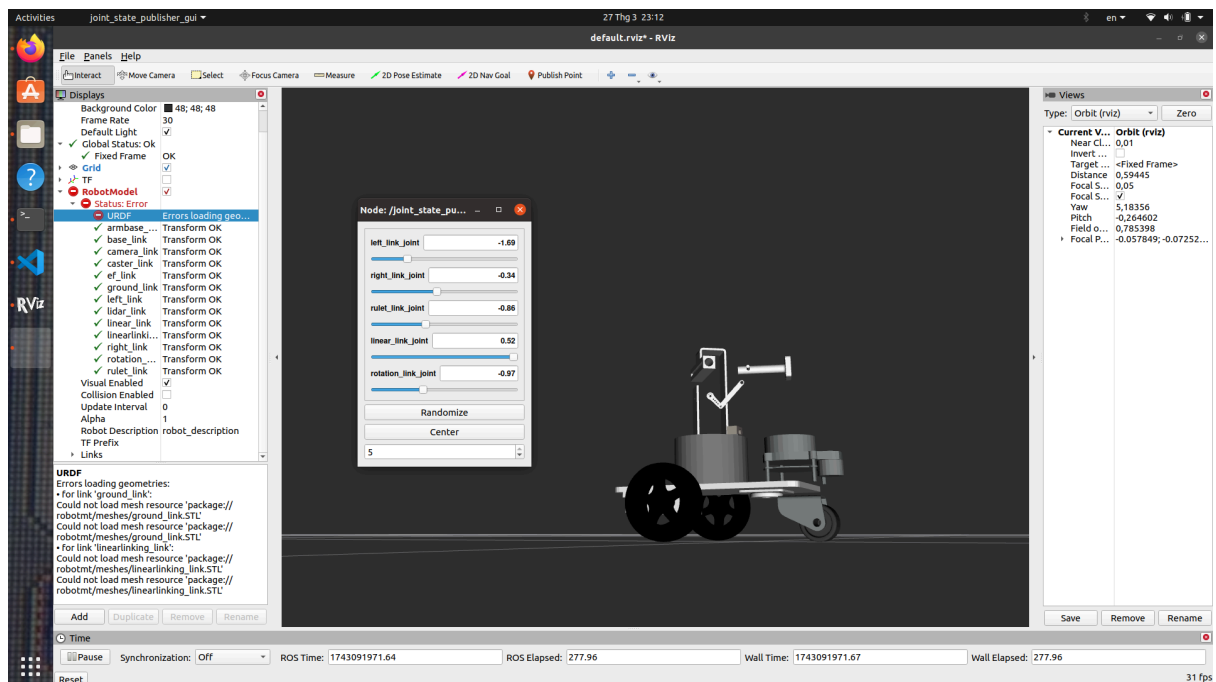
- Kết quả thu được từ các cảm biến:
 - + Camera:



+ LiDAR:

```
Activities Terminal 1 Thg 4 11:18 /home/huyen/ros1_gk_ws/src/robot1/launch/gazebo.launch http://localhost:11311
/home/huyen/ros1_gk_ws/src/robot1/launch/gazebo.launch http://local... huyen@nduckh29: ~/catkin_ws huyen@nduckh29: ~
[INFO] [1743481094.636122, 3.756000]: Receive Lidar's data including 360 points,
[INFO] [1743481094.640447, 3.758000]: First distance: 0.112411 meter
[INFO] [1743481094.835736, 3.956000]: Receive Lidar's data including 360 points,
[INFO] [1743481094.837578, 3.857000]: First distance: 0.112407 meter
[INFO] [1743481095.022243, 3.956000]: Receive Lidar's data including 360 points,
[INFO] [1743481095.026726, 3.958000]: First distance: 0.112412 meter
[INFO] [1743481095.218732, 4.056000]: Receive Lidar's data including 360 points,
[INFO] [1743481095.220786, 4.058000]: First distance: 0.112412 meter
[INFO] [1743481095.404717, 4.156000]: Receive Lidar's data including 360 points,
[INFO] [1743481095.409444, 4.158000]: First distance: 0.112412 meter
[INFO] [1743481095.592302, 4.256000]: Receive Lidar's data including 360 points,
[INFO] [1743481095.598179, 4.258000]: First distance: 0.112411 meter
[INFO] [1743481095.779087, 4.356000]: Receive Lidar's data including 360 points,
[INFO] [1743481095.781026, 4.357000]: First distance: 0.112412 meter
[INFO] [1743481095.989963, 4.456000]: Receive Lidar's data including 360 points,
[INFO] [1743481095.992262, 4.457000]: First distance: 0.112412 meter
[INFO] [1743481096.189894, 4.556000]: Receive Lidar's data including 360 points,
[INFO] [1743481096.194950, 4.558000]: First distance: 0.112412 meter
[INFO] [1743481096.374589, 4.656000]: Receive Lidar's data including 360 points,
[INFO] [1743481096.378857, 4.657000]: First distance: 0.112413 meter
[INFO] [1743481096.564195, 4.756000]: Receive Lidar's data including 360 points,
[INFO] [1743481096.565971, 4.757000]: First distance: 0.112405 meter
[INFO] [1743481096.737226, 4.856000]: Receive Lidar's data including 360 points,
[INFO] [1743481096.739486, 4.858000]: First distance: 0.112413 meter
[INFO] [1743481096.916932, 4.956000]: Receive Lidar's data including 360 points,
[INFO] [1743481096.918734, 4.957000]: First distance: 0.112410 meter
[INFO] [1743481097.085709, 5.056000]: Receive Lidar's data including 360 points,
[INFO] [1743481097.089986, 5.058000]: First distance: 0.112412 meter
[INFO] [1743481097.261149, 5.156000]: Receive Lidar's data including 360 points,
[INFO] [1743481097.266193, 5.159000]: First distance: 0.112410 meter
[INFO] [1743481097.429778, 5.256000]: Receive Lidar's data including 360 points,
[INFO] [1743481097.431539, 5.257000]: First distance: 0.112411 meter
[INFO] [1743481097.603848, 5.356000]: Receive Lidar's data including 360 points,
[INFO] [1743481097.605733, 5.357000]: First distance: 0.112410 meter
[INFO] [1743481097.779875, 5.456000]: Receive Lidar's data including 360 points,
[INFO] [1743481097.781817, 5.457000]: First distance: 0.112413 meter
[INFO] [1743481097.943310, 5.556000]: Receive Lidar's data including 360 points,
[INFO] [1743481097.949325, 5.558000]: First distance: 0.112411 meter
[INFO] [1743481098.118271, 5.656000]: Receive Lidar's data including 360 points,
[INFO] [1743481098.123835, 5.658000]: First distance: 0.112413 meter
[INFO] [1743481098.298062, 5.756000]: Receive Lidar's data including 360 points,
[INFO] [1743481098.303035, 5.758000]: First distance: 0.112413 meter
```


- + Các khớp bị lệch nhau, đặc biệt là các part gắn với base_link bị tách rời khỏi model
- + Không điều khiển được các joint có định dạng khác continuous



- + Lỗi chạy gazebo vì root link có quán tính

```

ROS_MASTER_URI=http://localhost:11311

process[gazebo-1]: started with pid [20049]
process[gazebo_gui-2]: started with pid [20054]
process[tf_footprint_base-3]: started with pid [20059]
process[spawn_model-4]: started with pid [20060]
process[fake_joint_calibration-5]: started with pid [20061]
process[robot_state_publisher-6]: started with pid [20066]
process[joint_state_publisher-7]: started with pid [20068]
process[controller_spawner-8]: started with pid [20069]
[ WARN] [1743349887.542893830]: The root link base_link has an inertia specifie
d in the URDF, but KDL does not support a root link with an inertia. As a work
around, you can add an extra dummy link to your URDF.
[INFO] [1743349888.004384, 0.000000]: Loading model XML from file /home/huyen/r
os1_gk_ws/src/robot1/urdf/robot1.urdf
[INFO] [1743349888.006910, 0.000000]: Waiting for service /gazebo/spawn_urdf_mo
del
[ INFO] [1743349888.190264097]: Finished loading Gazebo ROS API Plugin.
[ INFO] [1743349888.191297419]: waitForService: Service [/gazebo/set_physics_pr

```

-> Đặt lại trục, với base_link gắn với tâm của thân xe làm reference link, sau đó thêm một extra_dummy_link gắn với mặt đất, không có mass hoặc inertia làm root link

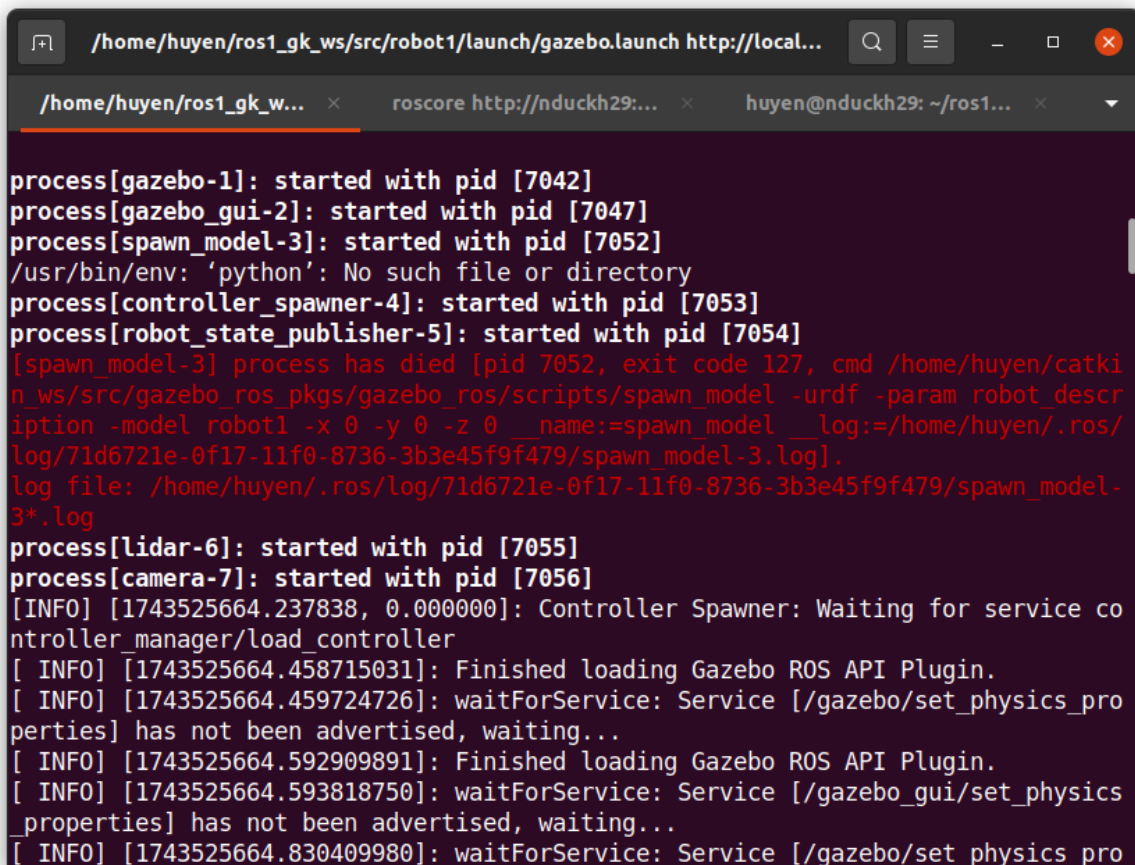
2. Lỗi điều khiển tay máy:

+ Khi chạy test_arm.py, hoặc publish dữ liệu dùng rostopic pub, truyền vào rotation_joint góc 1.57rad và ef_joint đoạn 0.02m thì mô hình trong gazebo bị giật lắc và crash hoặc không di chuyển

+ Kiểm tra: Dữ liệu không được truyền đến model trong gazebo dù có publish lên topic, hoặc có nhận được dữ liệu nhưng gây ra chuyển động sai => Lỗi crash

-> Mô tả sai hàm điều khiển trong file controller_manager.yaml (đã sửa lại và chạy được)

3. Lỗi khi xuất model lên gazebo



```
process[gazebo-1]: started with pid [7042]
process[gazebo_gui-2]: started with pid [7047]
process[spawn_model-3]: started with pid [7052]
/usr/bin/env: 'python': No such file or directory
process[controller_spawner-4]: started with pid [7053]
process[robot_state_publisher-5]: started with pid [7054]
[spawn_model-3] process has died [pid 7052, exit code 127, cmd /home/huyen/catkin_ws/src/gazebo_ros_pkgs/gazebo_ros/scripts/spawn_model -urdf -param robot_description -model robot1 -x 0 -y 0 -z 0 __name:=spawn_model __log:=/home/huyen/.ros/log/71d6721e-0f17-11f0-8736-3b3e45f9f479/spawn_model-3.log].
log file: /home/huyen/.ros/log/71d6721e-0f17-11f0-8736-3b3e45f9f479/spawn_model-3*.log
process[lidar-6]: started with pid [7055]
process[camera-7]: started with pid [7056]
[INFO] [1743525664.237838, 0.000000]: Controller Spawner: Waiting for service controller_manager/load_controller
[ INFO] [1743525664.458715031]: Finished loading Gazebo ROS API Plugin.
[ INFO] [1743525664.459724726]: waitForService: Service [/gazebo/set_physics_properties] has not been advertised, waiting...
[ INFO] [1743525664.592909891]: Finished loading Gazebo ROS API Plugin.
[ INFO] [1743525664.593818750]: waitForService: Service [/gazebo_gui/set_physics_properties] has not been advertised, waiting...
[ INFO] [1743525664.830409980]: waitForService: Service [/gazebo/set_physics_pro
```

-> Sửa lỗi: sudo ln -s /usr/bin/python3 /usr/bin/python

4. Lỗi không nhận ra service (chưa sửa được):

+ Ban đầu, khi dùng service mobileGoal.srv thì chạy lệnh rossrv show hiển thị đúng gói này, tuy nhiên khi import từ file srv vào

trong code thì gặp lỗi không xác định được `mobileGoal` và `mobileGoalResponse`

- + Check các khai báo trong `Cmakelist.txt` và `package.xml` đều có đầy đủ các khai báo yêu cầu cho service, tuy nhiên chưa tìm được lý do chạy lỗi