ĐẠI HỌC QUỐC GIA HÀ NỘI
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ

Trần Thanh Tuấn

MSSV: 22023506

4 wheeled-robot with 2-link robot

Báo cáo giữa kì ROS

HÀ NỘI - 2025

0.1 Yêu cầu

- 4 bánh mecanum + 2 arms robot (rotation) + lidar, camera, encoder.

0.2 Hướng tiếp cận

- Thực hiện vẽ mô hình 3D theo yêu cầu (xe + tay máy + sensor).
- Trích xuất thông tin mô hình 3D dưới dạng URDF, XACRO.
- Sử dụng các plugin có sẵn cho các sensor cùng chức năng.
- Điều khiển xe:
  - C1: Điều khiển bằng vi sai thông thường (teleop, cung cấp thông tin từ người dùng).
  - C2: Điều khiển bằng động học, cung cấp chính xác thông tin cho từng bánh (teleop).
  - C3: Điều khiển tự động sử dụng lidar né vật cản (cung cấp thông tin từ người dùng).
- Điều khiển tay máy hoạt động.

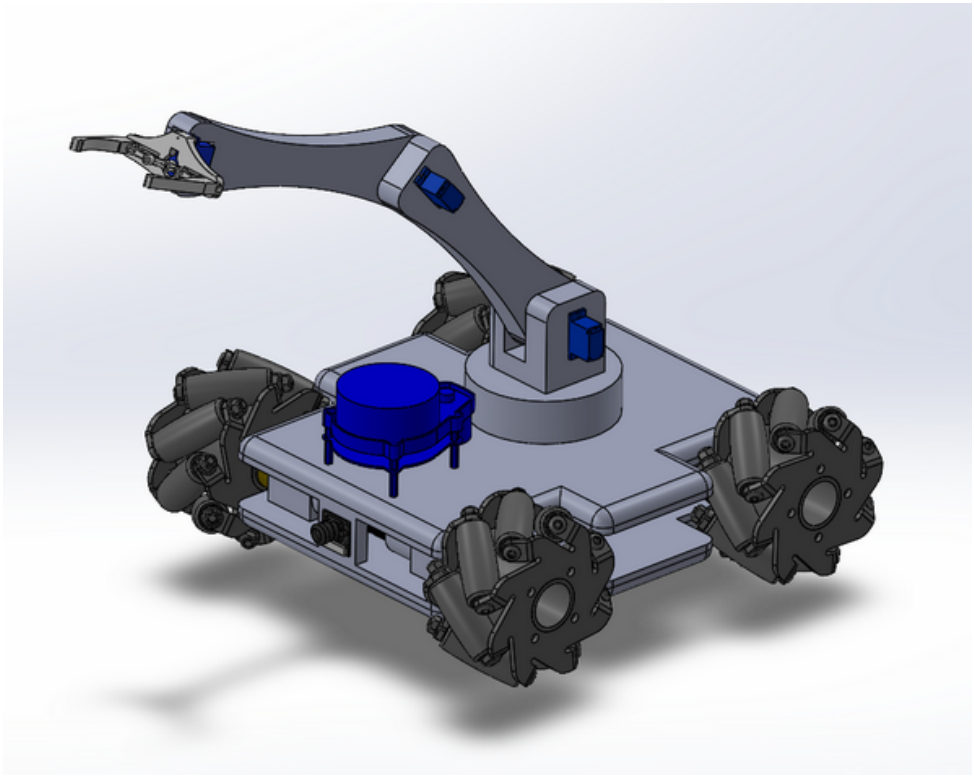
0.3 Nội dung chính

0.3.1 Thiết kế

Sử dụng phần mềm SolidWorks 2023 để thiết kế chi tiết mô hình 3D.

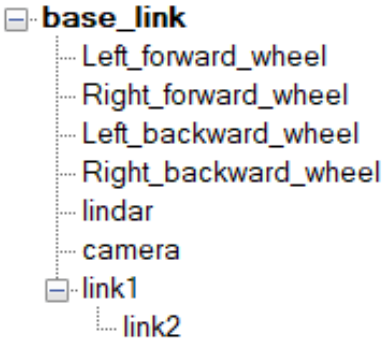
Bộ phận 3D *(wheel-mecanum, servo, lidar)* của robot sử dụng file có sẵn (*xác định vị trí chính xác trước - nếu cần thay đổi trong quá trình mô phỏng, giúp định hướng tốt vị trí cần đặt*), nguồn từ: `https://grabcad.com/library`.

Trục các components của robot được đặt như khi thiết lập ma trận DH. Trục quay của động cơ sẽ là trục z, trục x được đặt về phía trước giúp xe di chuyển tịnh tiến, trục y đặt tùy chỉnh.



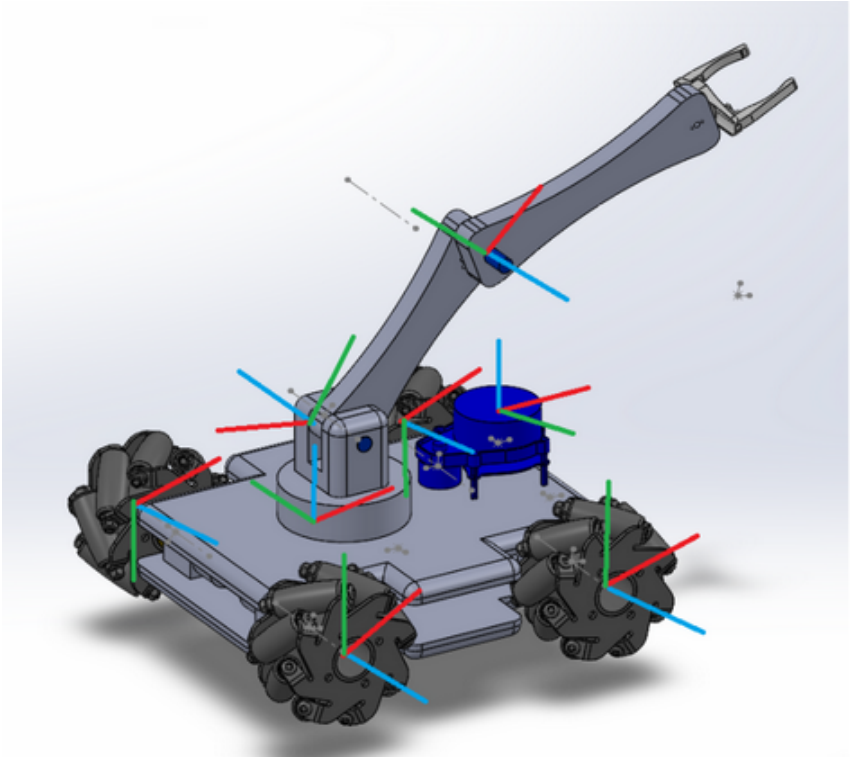
Hình 1: Hình ảnh robot được thiết kế

Cấu trúc parent-child của các components: Joint của 4 bánh, link1 của tay máy, camera và lidar sẽ liên kết trực tiếp với *base link*, trong khi đó link2 của tay máy sẽ liên kết với link1 do chuyển động của link2 phụ thuộc vào link1.



Hình 2: Cấu trúc của parent-child các components của robot

Trục tọa độ của robot: Trục quay của động cơ sẽ là trục z (màu xanh dương), trục x (màu đỏ) được đặt về phía trước giúp xe di chuyển tịnh tiến, trục y (màu xanh lá) đặt tùy chỉnh.



Hình 3: Hình ảnh trục của robot

0.3.2 Mô phỏng

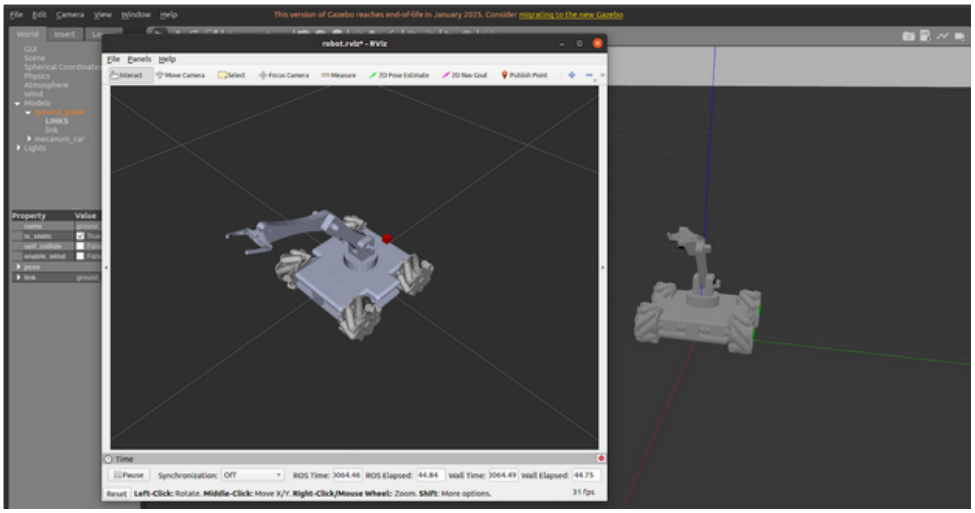
Mô hình 3D đã được mô tả cụ thể các thông tin dưới dạng 1 file URDF được trích xuất bằng các công cụ sẵn có trong SolidWorks.

Thuộc tính joint của các bộ phận

- **4 wheels**: continuous
- **2 cánh tay**: revolute
- **camera**: fixed
- **lidar**: fixed

Hình 4: Tệp file URDF được trích xuất

Mô hình robot 3D được mô phỏng thông qua hai nền tảng **RViz** và **Gazebo**.

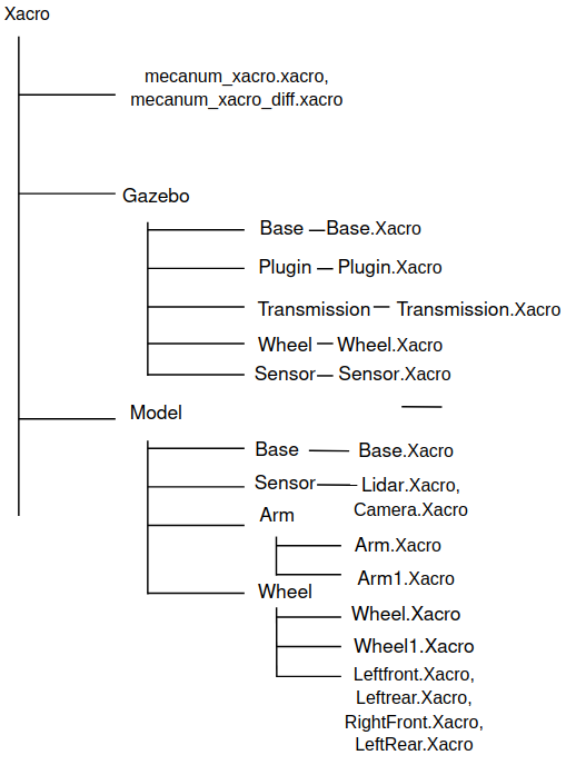


Hình 5: Hình ảnh robot được mô phỏng thông qua RViz và Gazebo

Chia nhỏ file URDF thành Xacro

Việc chia nhỏ file URDF thành các tệp Xacro giúp:

- Dễ dàng quản lý các thành phần của robot.
- Thuận tiện sửa đổi thông tin của robot thông qua macro.



Hình 6: Hình ảnh minh họa các nhánh file khi chia file Xacro

Cấu trúc thư mục mô phỏng robot

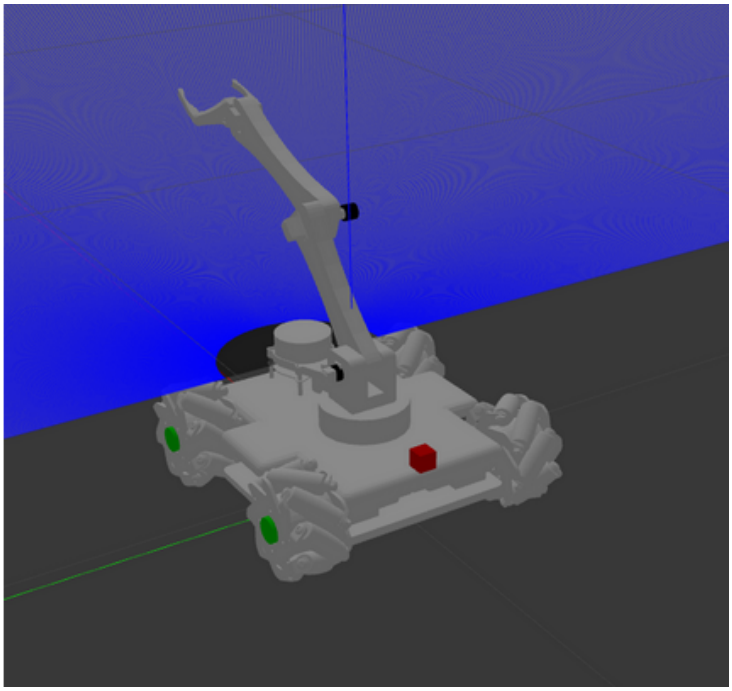
Bên trong các file xacro chính cho arm và wheel của robot:

- **File arm:** Chứa các thông tin 3D về cánh tay cho file arm1.
- **File arm1:** Chứa thông tin điều khiển link, joint, vật liệu `<material>` và ma sát `<mu>`, tạo 2 thuộc tính hình học dạng cylinder mô phỏng servo cho 2 cánh tay.
- **File wheel:** Chứa các thông tin 3D về bánh cho file wheel1.
- **File wheel1:** Khai báo vật liệu `<material>`, ma sát `<mu>`, vận tốc tối đa `<maxVel>`, độ sâu tối thiểu `<minDepth>` cho từng bánh xe, đồng thời tạo 4 thuộc tính hình học dạng cylinder mô phỏng encoder, điều khiển 4 bánh.

Thuộc tính hình học cho servo, camera, encoder

- Servo (cánh tay): Cylinder, kích thước 0.05m, màu đen.
- Encoder (bánh xe): Cylinder, kích thước 0.07m, màu xanh lá.
- Camera:
  - Hình dạng: Hộp (0.02x0.02x0.02 m), màu đỏ.
  - FOV: 1.3962634.
  - Kích thước ảnh: 800x800.
  - Định dạng: R8G8B8.

Do trong file 3D đã có thực hiện vẽ lidar, nên ở trong quá trình thực hiện mô phỏng, mô hình lidar 3D này có thể được sử dụng lại để mô phỏng chính mô hình đó trong khi thực hiện chạy các nền tảng mô phỏng. Đồng thời, tôi thực hiện thay đổi vị trí và hình dạng của camera. Kết quả sau khi thực hiện thêm các chức năng giúp điều khiển được robot được minh họa tại 7

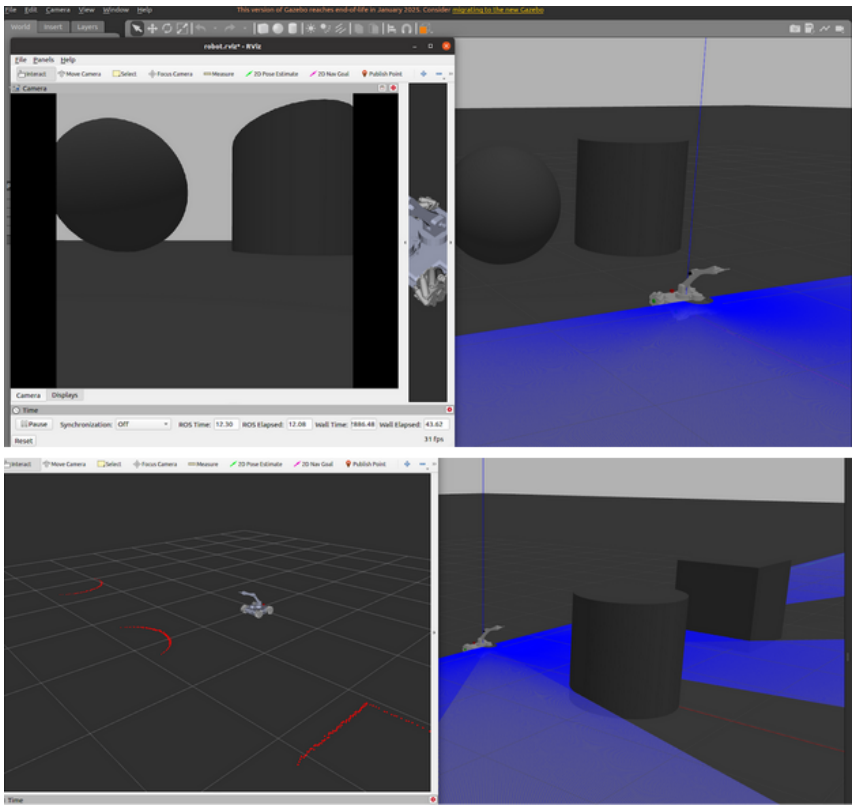


Hình 7: Hình ảnh encoder, camera, lidar tự custom trên robot. Màu xanh lá: encoder, màu đỏ: camera, màu đen: servo.

Plugin sử dụng cho các components của robot:

- 4 bánh xe: libgazebo ros control.so
- 2 cánh tay: libgazebo ros joint position control.so
- Lidar: libgazebo ros laser.so

- Camera: libgazebo ros camera.so



Hình 8: Hình ảnh hoạt động của camera (bên trái) và lidar (màu xanh - bên phải và bên dưới).

0.3.3 Điều khiển

Transmission: Định nghĩa cơ chế truyền động cho components của robot trong quá trình mô phỏng:

- 4 truyền động bánh xe. Mỗi bánh xe có một động cơ riêng (`wheel_motor1` → `wheel_motor4`) và sử dụng giao diện điều khiển vận tốc (`VelocityJointInterface`) hay bộ truyền động (actuator) cho các khớp của 4 bánh.
- 2 truyền động khớp. Mỗi khớp có một động cơ (`motor1`, `motor2`) và sử dụng giao diện bộ truyền động vị trí (`PositionJointInterface`).

Điều này giúp khi mô phỏng trong Gazebo, mô phỏng các cơ chế truyền động của các components của robot, hỗ trợ điều khiển vận tốc bánh xe và vị trí khớp.

Yaml: Cấu hình bộ điều khiển và các giá trị cài đặt cho actuator, controller:

- Sử dụng bộ điều khiển vi sai "DiffDriveController" để điều khiển cho 4 bánh.
- Tại đó, tôi thực hiện khai báo 4 joints của 4 bánh xe, cấu hình các thông tin như bán kính bánh, chiều dài và giới hạn vận tốc, gia tốc tịnh tiến và quay. Ở đây tôi giới hạn vận tốc và gia tốc một bánh có thể đạt tối đa là $3m/s$, $3m/s^2$ và $3rad/s$, $3rad/s^2$.
- Với điều khiển động học từng bánh, tôi sử dụng bộ điều khiển "JointVelocityController" cho từng joints của 4 bánh xe, cấu hình các thông tin như bán kính bánh, chiều dài và giới hạn vận tốc, gia tốc tịnh tiến và quay. Ở đây tôi giới hạn vận tốc và gia tốc một bánh có thể đạt tối đa là $3m/s$, $3m/s^2$ và $3rad/s$, $3rad/s^2$.
- Tương tự, tôi sử dụng bộ điều khiển vị trí "JointPositionController" để điều khiển và cấu hình các thông tin của servo sử dụng cho tay máy với các thông tin về joint tương ứng với từng tay máy đã biết.

- Tốc độ cập nhật (publish_rate): 50 Hz cho cả 2 bộ controller.

Cách điều khiển thứ nhất: điều khiển động học

Các topic được sử dụng trong quá trình điều khiển: **Bánh xe:**

- /meca/left_forward_controller/command
- /meca/left_backward_controller/command
- /meca/right_forward_controller/command
- /meca/right_backward_controller/command

Tay máy:

- /meca/servo1_controller/command
- /meca/servo2_controller/command

```
IF key == "w":  # Move Forward
        SET twist.linear.x = velocity
        SET twist.linear.y = 0
        SET twist.angular.z = 0

ELSE IF key == "s":  # Move Backward
        SET twist.linear.x = -velocity
        SET twist.linear.y = 0
        SET twist.angular.z = 0

ELSE IF key == "a":  # Rotate Left
        SET twist.linear.x = 0
        SET twist.linear.y = 0
        SET twist.angular.z = angular_velocity

ELSE IF key == "d":  # Rotate Right
        SET twist.linear.x = 0
        SET twist.linear.y = 0
        SET twist.angular.z = -angular_velocity

ELSE IF key == "l":  # Decrease Speed
        DECREASE velocity (minimum 0.1)
        DECREASE angular_velocity (minimum 0.1)
        PRINT "Speed Decreased"

ELSE IF key == "u":  # Increase Speed
        INCREASE velocity
        INCREASE angular_velocity
        PRINT "Speed Increased"

ELSE IF key == "x":  # Stop
        SET twist.linear.x = 0
        SET twist.linear.y = 0
        SET twist.angular.z = 0
        PRINT "Stopping"
```

Hình 9: Mã giả để điều khiển robot thông qua teleop.

Hình số 9 mô tả đoạn mã dùng để điều khiển robot thông qua teleoperation keyboard với nội dung:

- **w**: đi thẳng
- **s**: đi lùi
- **a**: rẽ trái
- **d**: rẽ phải

- **x**: dừng
- **u**: tăng tốc 0.001
- **l**: giảm tốc 0.001

Động học cho từng bánh xe của robot:

$$\begin{bmatrix} w_{fl} \\ w_{fr} \\ w_{rl} \\ w_{rr} \end{bmatrix} = \frac{1}{r} \begin{bmatrix} 1 & -1 & -(l_x + l_y) \\ 1 & 1 & (l_x + l_y) \\ 1 & 1 & -(l_x + l_y) \\ 1 & -1 & (l_x + l_y) \end{bmatrix} \begin{bmatrix} v_x \\ v_y \\ w_z \end{bmatrix} \tag{1}$$

Trong đó:

- w_{fl}, w_{fr}, w_{rl}, w_{rr} là vận tốc góc của 4 bánh xe,
- r là bán kính của bánh xe,
- l_x, l_y lần lượt là khoảng cách từ tâm đến tâm bánh xe theo trục x và y,
- v_x, v_y là vận tốc dài theo trục x và y,
- w_z là góc quay theo trục z của xe.

Trong trường hợp này:

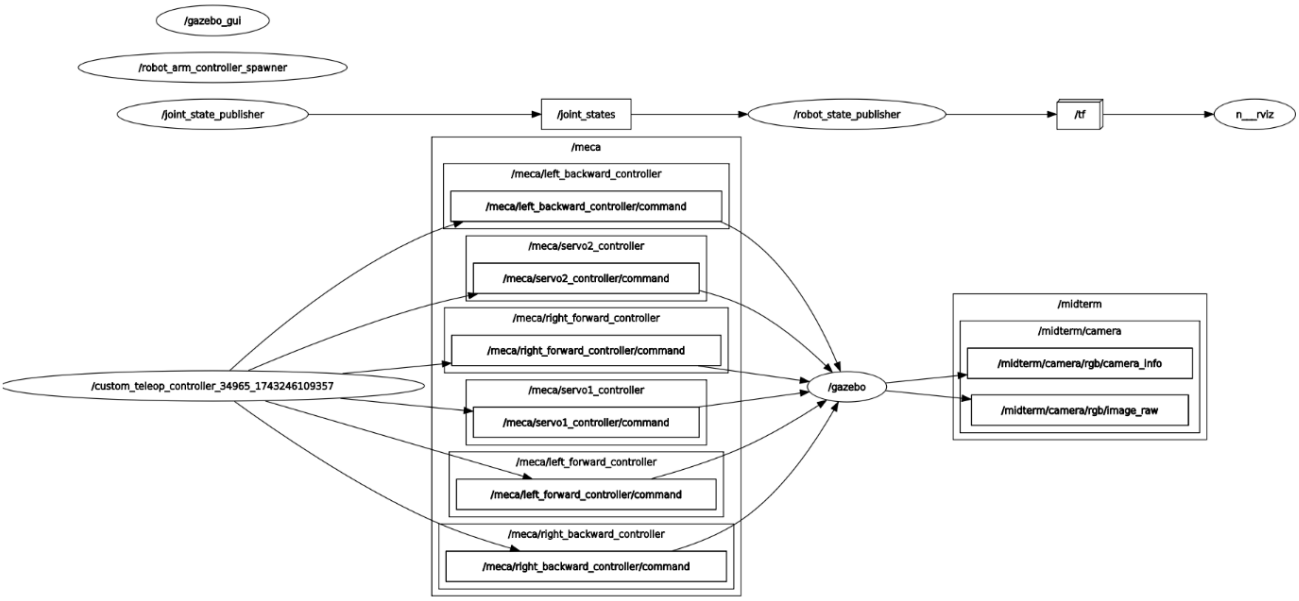
$$r = 0.05983 \text{ m}, \quad l_x = 0.148 \text{ m}, \quad l_y = 0.075 \text{ m} \tag{2}$$

```
BEGIN CustomTeleopController

    INITIALIZE ROS node "custom_teleop_controller"

    CREATE publishers for topics:
        - Left front wheel velocity
        - Left back wheel velocity
        - Right front wheel velocity
        - Right back wheel velocity
        - Servo 1 position
        - Servo 2 position

    SET control loop rate to 10 Hz

    DEFINE robot physical parameters:
        - lx (X-axis distance)
        - ly (Y-axis distance)
        - r (Wheel radius)

    INITIALIZE speed control variables:
        - velocity (default = 0.2)
        - velocity_step (0.001)
        - max_velocity (1.0)
        - min_velocity (-1.0)

    PRINT "Custom Teleop Control Ready!"

    FUNCTION servo_oscillation:
        SET time t = 0
        WHILE ROS is running:
            COMPUTE servo1_value = 0.5 + 0.4 * sin(t)
            COMPUTE servo2_value = 0.1 + 0.4 * cos(t)
            PUBLISH servo1_value
            PUBLISH servo2_value
            INCREMENT t by 0.1
            WAIT 0.1 seconds
        END WHILE

    FUNCTION inverse_kinematics(vx, vy, omega):
        COMPUTE L = lx + ly
        COMPUTE wheel_velocities using inverse Jacobian matrix
        RETURN wheel_velocities

    FUNCTION publish_wheel_speeds(vx, vy, omega):
        COMPUTE wheel velocities using inverse_kinematics
        PUBLISH wheel velocities to ROS topics
```

Hình 10: Mã giả để điều khiển teleop cho robot

Mã giả 10, thực hiện điều khiển robot thông qua teleop với nội dung chính:

- Thực hiện khởi tạo một publish node tên: "custom_teleop_controller".
- Tạo các thuộc tính publish từ class `Publish` cho từng joint của bánh xe và tay máy đã được định nghĩa.
- Khởi tạo các thuộc tính vật lý của xe theo động học (r, l_x, l_y), và tốc độ của xe.
- Tay máy sẽ được thực hiện điều khiển tự động không phụ thuộc vào động học mà sẽ được thực hiện ở một thread khác cùng với chương trình điều khiển xe, đảm bảo sự hoạt động đồng bộ của xe và tay máy.
- Link thứ nhất được giới hạn hoạt động từ $(0 \rightarrow 0.6)$ rad trong khi đó tay máy thứ hai hoạt động từ $(0 \rightarrow 0.9)$ rad.
- Định nghĩa hàm thành viên cho động học ngược của xe, giá trị trả về sẽ là tốc độ góc của 4 bánh xe cần di chuyển theo yêu cầu.
- Cuối cùng, hàm điều khiển hay hàm chính sử dụng tất cả các chức năng đã được định nghĩa trước đó. Do điều khiển bằng teleop, nội dung được trình bày rõ ở hình số 9

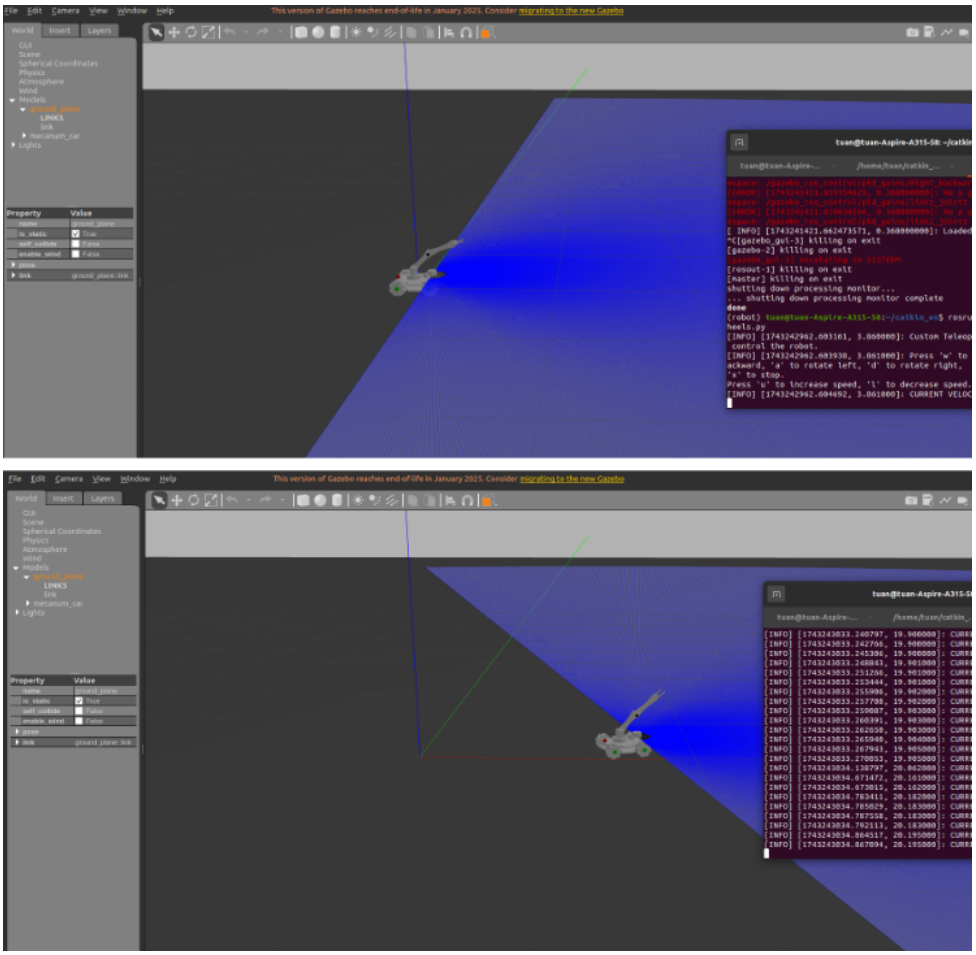


Hình 11: rpt_graph của cách điều khiển thứ nhất

Trong hình 11, sơ đồ `rqt_graph` minh họa quá trình truyền thông tin từ node "custom_ teleop_controller" đến các topic điều khiển của bốn bánh xe và hai khớp tay máy. Cụ thể, node này thực hiện gán thông tin điều khiển được nhận trực tiếp từ quá trình tương tác giữa người dùng, máy tính thông qua bàn phím và *publish* dữ liệu lên các topic tương ứng để vận hành robot trong môi trường mô phỏng Gazebo.

Bên cạnh đó, Gazebo cũng cung cấp dữ liệu hình ảnh từ camera thông qua các topic "camera_info" và "image_raw". Node "/custom_teleop_controller" tiếp nhận tín hiệu điều khiển từ người dùng, dựa trên các phím bấm đã được định nghĩa trước cùng với chức năng điều khiển tương ứng (ví dụ: phím "w" để di chuyển thẳng). Các tín hiệu này sau đó được xử lý và *publish* lên các topic điều khiển của bánh xe.

Trong khi đó, cánh tay robot được điều khiển tự động với các giá trị trạng thái được cập nhật liên tục theo thời gian, đảm bảo sự chuyển động linh hoạt của các khớp tay máy. Cuối cùng, toàn bộ thông tin đã được cập nhật được gửi đến Gazebo để thực hiện điều khiển robot trong môi trường mô phỏng. Kết quả được mình họa tại hình 12



Hình 12: Trạng thái xe trong quá trình điều khiển. Hình ảnh xe khi ban đầu mới được khởi tạo (phía trên) và xe sau khi điều khiển rẽ trái và đi thẳng (phía dưới), tay máy đồng thời được hoạt động.

Cách điều khiển thứ hai

Các topic được sử dụng trong quá trình điều khiển:

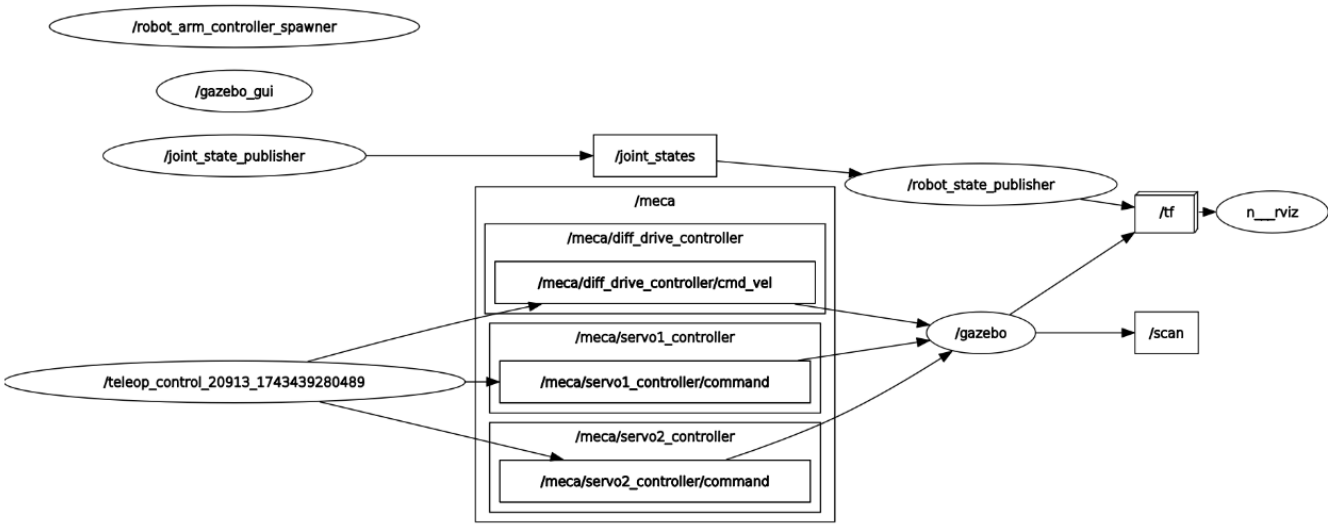
- Bánh xe:
  - `/meca/diff_drive_controller/cmd_vel`
- Tay máy:
  - `/meca/servo1_controller/command`
  - `/meca/servo2_controller/command`
- Vị trí:
  - `/midterm/odom`

```
BEGIN TeleopController

    INITIALIZE ROS node "teleop_control"

    CREATE publishers for:
        - Robot movement (cmd_vel topic)
        - Servo 1 position
        - Servo 2 position

    INITIALIZE:
        - Twist message for velocity control
        - velocity = 0.5 (default linear speed)
        - angular_velocity = 0.5 (default angular speed)

    PRINT control instructions (w, s, q, e, x, u, l)

    FUNCTION get_key:
        CAPTURE single keyboard input
        RETURN key

    FUNCTION servo_oscillation:
        SET t = 0
        WHILE ROS is running:
            COMPUTE servo1_value = 0.2 + 0.4 * sin(t)
            COMPUTE servo2_value = 0.5 + 0.4 * cos(t)
            PUBLISH servo1_value
            PUBLISH servo2_value
            INCREMENT t by 0.1
            WAIT 0.1 seconds
        END WHILE

    FUNCTION run:
        START servo_oscillation in a separate thread
```

Hình 13: Mã giả cho các điều khiển thứ hai, điều khiển bằng teleoperation.

3.2.2.1 Điều khiển thông qua teleop:

- Thực hiện khởi tạo một publish node tên: "teleop_control".
- Tạo các thuộc tính publish từ class `Publish` cho từng joint của bánh xe và tay máy đã được định nghĩa.
- Thuộc tính ban đầu của xe như vận tốc, góc quay.
- Tay máy sẽ được thực hiện điều khiển tự động không phụ thuộc vào động học mà sẽ được thực hiện ở một thread khác cùng với chương trình điều khiển xe, đảm bảo sự hoạt động đồng bộ của xe và tay máy.
- Link thứ nhất được giới hạn hoạt động từ $(0 \rightarrow 0.6)$ rad trong khi đó tay máy thứ hai hoạt động từ $(0 \rightarrow 0.9)$ rad.
- Cuối cùng, hàm điều khiển hay hàm chính sử dụng tất cả các chức năng đã được định nghĩa trước đó, do điều khiển bằng teleop, tôi đã thực hiện định nghĩa tại hình 9

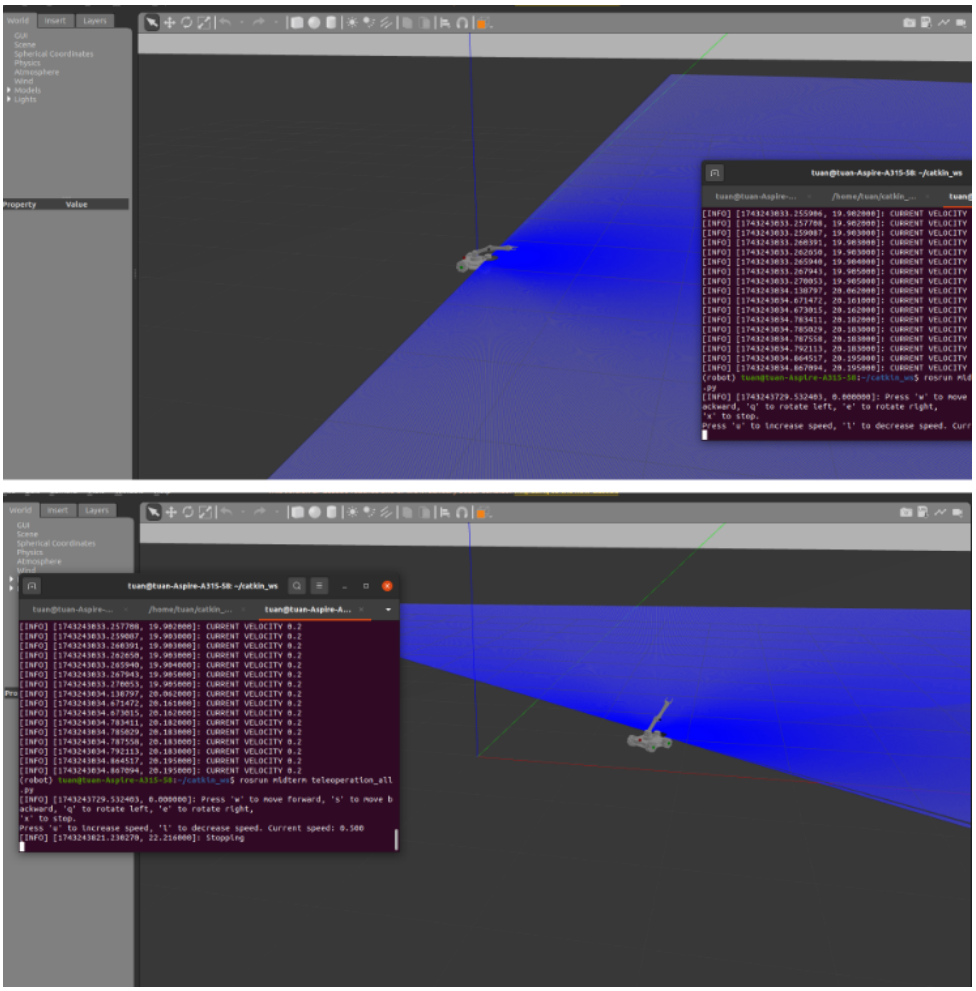


Hình 14: Rqt_graph của điều khiển bằng cách thứ hai, qua teleop.

Trong hình ảnh thứ 14, `rqt_graph` minh họa quá trình truyền thông tin từ node "/teleop_control..." đến các topic điều khiển của bốn bánh xe và hai khớp tay máy. Cụ thể, node này thực hiện gán thông tin điều khiển và publish dữ liệu lên các topic tương ứng để vận hành robot trong môi trường mô phỏng Gazebo.

Ngoài ra, Gazebo cũng cung cấp thông tin hình ảnh từ camera thông qua các topic "camera_info" và "image_raw". Node "/teleop_control..." nhận tín hiệu điều khiển từ người dùng, được xác định thông qua các phím bấm đã được định nghĩa trước (ví dụ: phím "w" để di chuyển thẳng, ...). Các tín hiệu điều khiển này sau đó được xử lý và publish giá trị tương ứng lên các topic của bánh xe.

Trong khi đó, cánh tay robot được điều khiển tự động với các giá trị liên tục cập nhật theo thời gian, tạo ra sự chuyển động nhịp nhàng cho các khớp tay máy. Cuối cùng, tất cả các thông tin đã được cập nhật sẽ được gửi đến Gazebo để thực hiện quá trình điều khiển robot. Kết quả của quá trình này được minh họa trong hình 15.



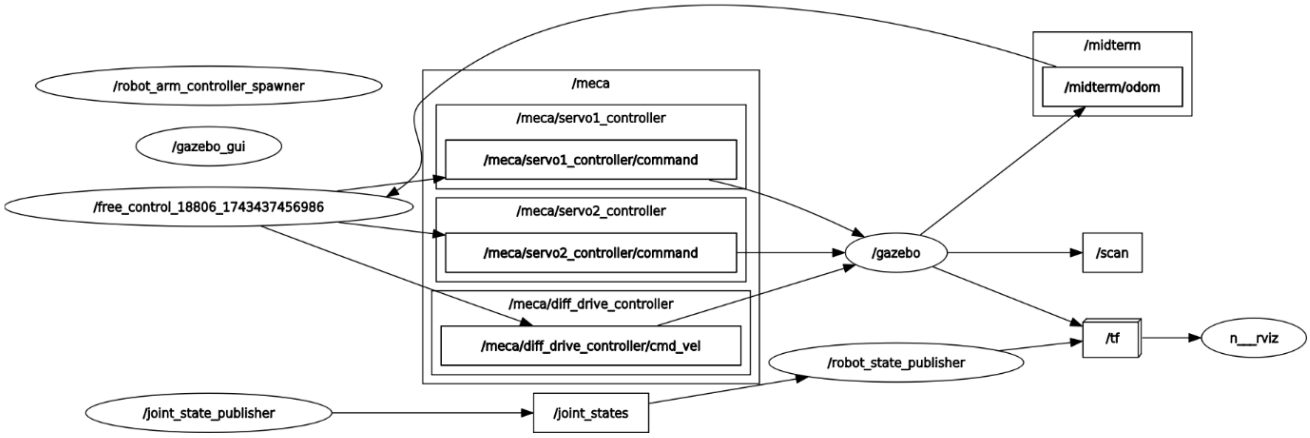
Hình 15: Hình ảnh xe khi ban đầu mới được khởi tạo (phía trên) và xe sau đi điều khiển rẽ trái và đi thẳng (phía dưới), tay máy đồng thời được hoạt động.

```
BEGIN TeleopController

INITIALIZE ROS node "free_control"

CREATE publishers for:
  - Robot movement (cmd_vel topic)
  - Servo 1 position
  - Servo 2 position
CREATE subscriber for:
  - Odom

INITIALIZE:
  - Twist message for velocity control
  - odom_x = 0 (default linear speed)
  - odom_yaw = 0 (default angular speed)

FUNCTION get_odom:
  Distance = (robot_x - odom_x)
  Angle = (robot_yaw - odom_yaw)
  IF Distance < 0.2
  Elif Angle < 0.09
  ELSE
  RETURN RUN
  RETURN STOP

FUNCTION servo_oscillation:
  SET t = 0
  WHILE ROS is running:
    COMPUTE servo1_value = 0.2 + 0.4 * sin(t)
    COMPUTE servo2_value = 0.5 + 0.4 * cos(t)
    PUBLISH servo1_value
    PUBLISH servo2_value
    INCREMENT t by 0.1
    WAIT 0.1 seconds
  END WHILE

FUNCTION run:
  START servo_oscillation in a separate thread
  RUN robot
```

Hình 16: Mã giả cho cách điều khiển thứ hai thông qua giá trị đầu vào từ người dùng.

3.2.2.2 Điều khiển thông qua giá trị đầu vào từ người dùng Mã giả 16 thực hiện điều khiển robot thông qua việc nhập giá trị đầu vào:

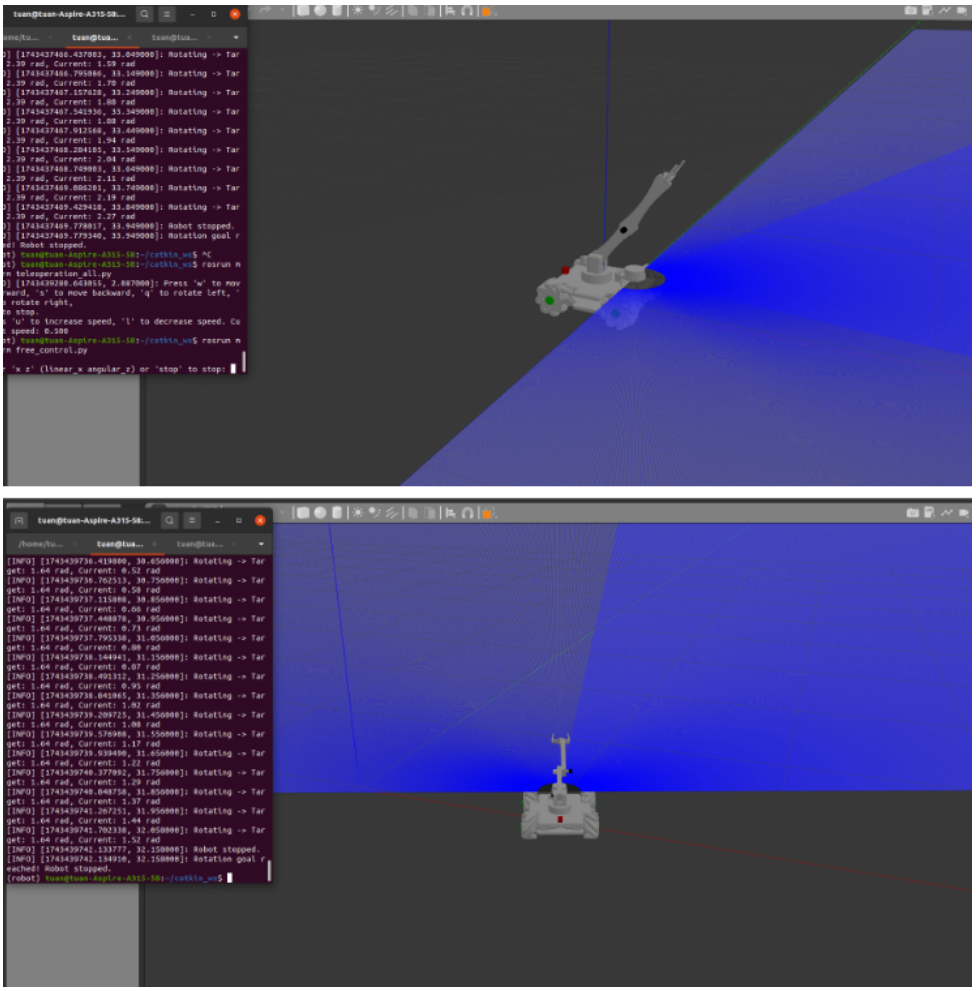
- Thực hiện khởi tạo 1 publish node tên: "free control".
- Tạo các thuộc tính publish từ class `Publish` cho từng joint của bánh xe và tay máy đã được định nghĩa, khởi tạo 1 subscriber cho topic `odom`, dùng để xác định vị trí của xe.
- Thuộc tính ban đầu của odom như vị trí và góc quay (giá trị mặc định của xe lúc khởi tạo).
- Tay máy sẽ được thực hiện điều khiển tự động không phụ thuộc vào động học mà sẽ được thực hiện ở 1 thread khác cùng với chương trình điều khiển xe, đảm bảo sự hoạt động đồng bộ của xe và tay máy.
- Link thứ nhất được giới hạn hoạt động từ $(0 \rightarrow 0.6)$ rad trong khi đó tay máy thứ hai hoạt động từ $(0 \rightarrow 0.9)$ rad.
- Cuối cùng, hàm điều khiển hay hàm chính sử dụng tất cả các chức năng đã được định nghĩa trước đó. Đầu vào người dùng cung cấp giá trị vận tốc, góc quay theo yêu cầu, xe thực hiện di chuyển và đồng thời subscriber lấy liên tục các giá trị từ `odom` để đảm bảo khi xe đạt đến vị trí mong muốn sẽ dừng. Logic điều khiển này được sử dụng đồng thời cho vị trí tịnh tiến và góc quay của robot.



Hình 17: rpt graph của phương thức điều khiển thứ hai.

Hình minh họa 17 mô tả quá trình gửi dữ liệu điều khiển phương tiện thông qua node "free_control..", do tôi khởi tạo. Node này thực hiện khai báo và quản lý các topic cần thiết để điều hướng và điều chỉnh góc quay của robot. Cụ thể, bốn bánh xe sử dụng chung một topic điều khiển vận tốc `/cmd_vel`, trong khi hai khớp cánh tay được điều khiển thông qua các topic riêng biệt: `servo1_controller/command` và `servo2_controller/command`.

Thông tin về hướng di chuyển và góc quay của robot được gửi tới các topic này và thực thi trong môi trường mô phỏng Gazebo. Ngoài ra, một subscriber được khai báo để theo dõi vị trí hiện tại của robot thông qua topic `/midterm/odom`. Giá trị này phản ánh vị trí thực tế của robot trong môi trường mô phỏng. Node "free_control.." liên tục cập nhật dữ liệu từ `odom` và so sánh với vị trí mục tiêu do người dùng nhập vào. Khi robot đạt đến vị trí mong muốn, hệ thống tự động dừng lại, đảm bảo tính chính xác của quá trình điều khiển. Hình ảnh minh họa giai đoạn robot thay đổi trạng thái, thực hiện di chuyển được mô tả cụ thể tại 18



Hình 18: Hình ảnh xe khi ban đầu mới được khởi tạo (phía trên) và xe sau khi điều khiển đi thẳng 1m và quay 1.57 rad (phía dưới), tay máy đồng thời được hoạt động.

0.4 Cách điều khiển thứ ba: Điều khiển động học

0.4.1 Các topic được sử dụng trong quá trình điều khiển

Bánh xe:

- `/meca/left_forward_controller/command`
- `/meca/left_backward_controller/command`
- `/meca/right_forward_controller/command`
- `/meca/right_backward_controller/command`

Tay máy:

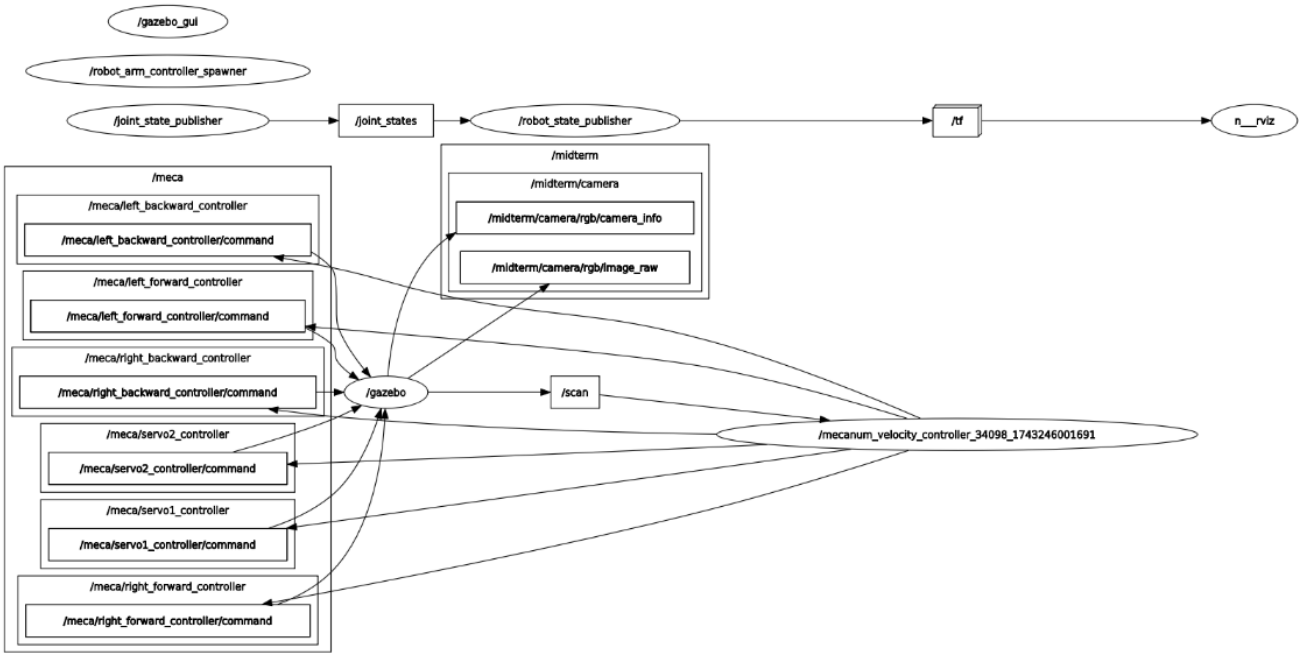
- `/meca/servo1_controller/command`
- `/meca/servo2_controller/command`

Phương trình động học cho robot tại phương thức điều khiển này được mô tả cụ thể tại 1, thực hiện cung cấp cho robot vận tốc dài và góc quay, từ đó thực hiện tính toán động học ngược và trả ra các giá trị giúp robot di chuyển.

```
BEGIN Controller

    INITIALIZE ROS node "mecanum_velocity_controller"

    CREATE publishers for:
        - Left front wheel velocity
        - Left back wheel velocity
        - Right front wheel velocity
        - Right back wheel velocity
        - Servo motor 1
        - Servo motor 2

    SUBSCRIBE to LiDAR topic "/scan"

    INITIALIZE:
        - scan_data = None  # Stores LiDAR scan data
        - rate = 10 Hz
        - Robot parameters: lx, ly, r (dimensions and wheel radius)
        - OBSTACLE_DISTANCE_THRESHOLD = 0.5 meters
        - LEFT_REGION = angles 90 to 150
        - RIGHT_REGION = angles 210 to 270

    FUNCTION lidar_callback(msg):
        UPDATE scan_data with LiDAR ranges

    FUNCTION get_obstacle_direction:
        IF scan_data is None:
            RETURN None  # No LiDAR data

        FILTER valid distances for LEFT and RIGHT regions
        FIND minimum distance in LEFT and RIGHT regions

        IF both distances < OBSTACLE_DISTANCE_THRESHOLD:
            RETURN "both"
        ELSE IF left distance < OBSTACLE_DISTANCE_THRESHOLD:
            RETURN "left"
        ELSE IF right distance < OBSTACLE_DISTANCE_THRESHOLD:
            RETURN "right"
        ELSE:
            RETURN "clear"

    FUNCTION servo_oscillation:
        SET t = 0
        WHILE ROS is running:
            COMPUTE servo1_value = 0.5 + 0.4 * sin(t)
            COMPUTE servo2_value = 0.1 + 0.4 * cos(t)
            PUBLISH servo1_value and servo2_value
            INCREMENT t by 0.1
            WAIT 0.1 seconds
        END WHILE
```

Hình 19: Mã giả cho điều khiển tự động né vật cản.

- Thực hiện khởi tạo tương tự 6 Publishers cho 6 topics là 4 bánh và 2 cánh tay.
- Đồng thời, khởi tạo 1 Subscriber để lấy thông tin từ lidar (`/scan`).
- Do phạm vi hoạt động của lidar bị giới hạn do vị trí đặt của cánh tay, nên giới hạn phạm vi hoạt động của lidar từ $(0 \rightarrow 180)$ độ, trong đó:
  - Phạm vi trái của lidar được định nghĩa từ $(90 \rightarrow 150)$ độ.
  - Phạm vi phải của lidar từ $(210 \rightarrow 270)$ độ.
- Logic hoạt động của lidar và xe tuân theo tín hiệu rẽ (**left, right, both**). Giá trị nhỏ nhất từ 2 buffers `left` và `right` được tìm kiếm:
  - Nếu vật cản nằm ở bên trái, xe sẽ rẽ phải.
  - Nếu vật cản nằm ở bên phải, xe sẽ rẽ trái.
  - Nếu vật cản xuất hiện ở cả hai phía, xe sẽ tự động dừng lại.
- Thông tin cung cấp cho robot ngay từ ban đầu sẽ là vận tốc dài theo hướng x, y hoặc cả góc quay theo trục z của xe.

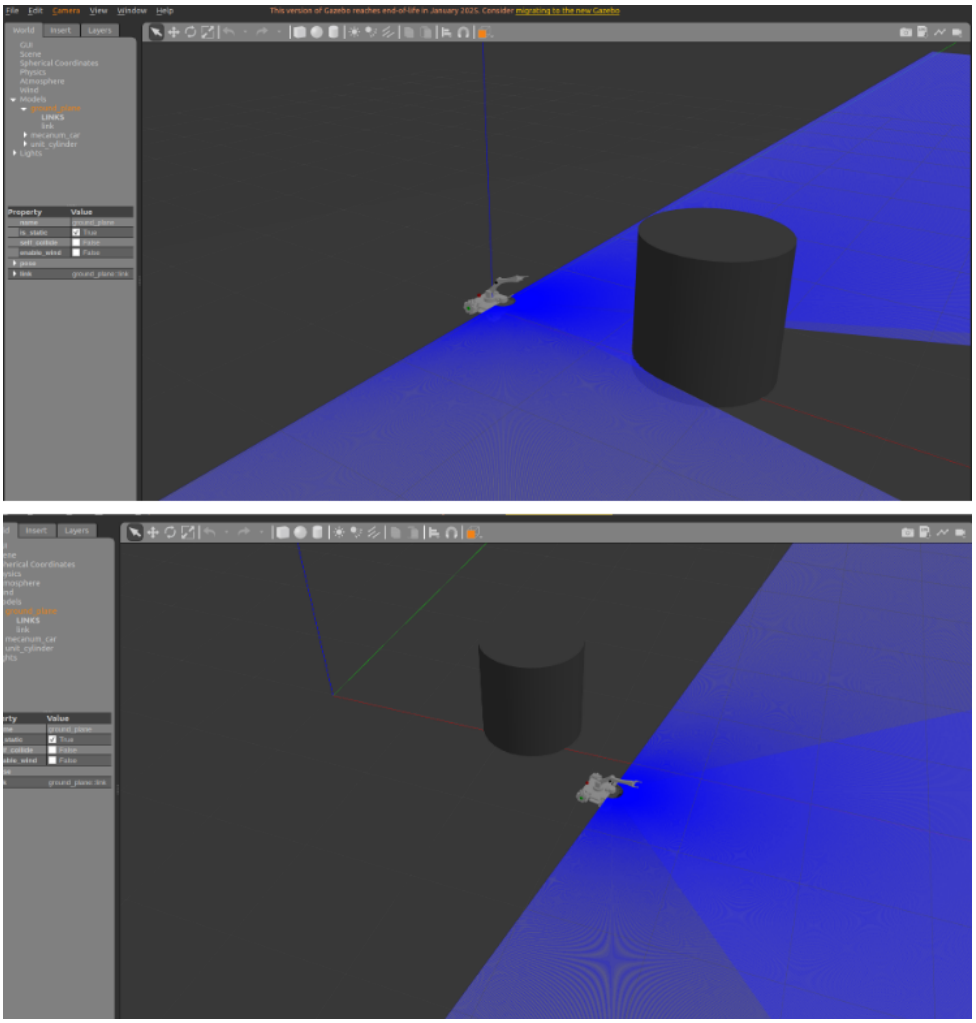


Hình 20: rqt Graph của phương thức điều khiển thứ ba.

Đồ thị 20, tôi khởi tạo node "mecanum_vel..." nhằm thực hiện chức năng tự động né vật cản và điều hướng chuyển động của xe. Logic điều khiển được mô tả tại 19

Tôi Khởi tạo các *Pulisher* cho các topic được sử dụng để *publish* bao gồm 4 topic điều khiển 4 bánh xe và 2 topic điều khiển 2 khớp của tay máy. Đồng thời, tôi cũng khởi tạo một *subscriber* để thu thập thông tin từ môi trường mô phỏng, cụ thể là dữ liệu từ cảm biến LiDAR.

Logic điều khiển sẽ được tự động kích hoạt dựa trên thông tin di chuyển do người dùng cung cấp. Xe sẽ tự động điều hướng và né tránh vật cản bằng cách xử lý dữ liệu từ LiDAR, đảm bảo khả năng di chuyển linh hoạt trong môi trường mô phỏng. Hình minh họa kết quả được mô tả tại 21



Hình 21: Kết quả hoạt động của xe tự động né vật cản dùng lidar.