IRP LAB7 Deliverables

Jason Zhai

It contains everything for lab7

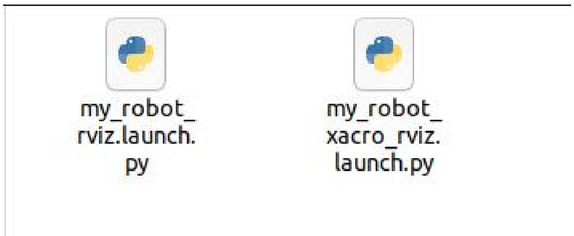
The deliverables are highlighted by the big red squares

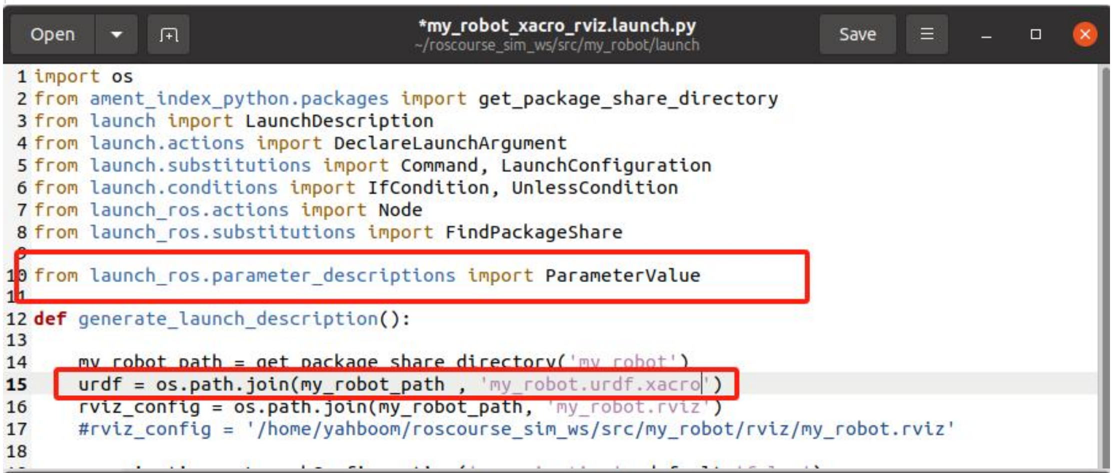
Task 1 - Adding Xacro Functionality to URDF

1. **Action:** Copy the URDF from the previous lab and rename it my_robot.urdf.xacro. (If you did the extended task with the arm, start with the version that is only the car first.)
2. **Note:** To be able to use the xacro functionality, we need to tell the file that we need this capability in the root tag. We do this using xmlns (xml namespace). It is kind of like an import statement which tells the specific meaning of tags so there is no ambiguity.
3. **Action:** Within the robot opening tag (after 'name' is specified and before >), add `xmlns:xacro="http://www.ros.org/wiki/xacro"`
   `<robot name="my_robot" xmlns:xacro="http://www.ros.org/wiki/xacro">`
4. **Action:** Make a copy of 'my_robot_rviz.launch.py' and rename it 'my_robot_xacro_rviz.launch.py'
5. **Action:** Make the following modifications to the file:
   - Add to imports: `from launch_ros.parameter_descriptions import ParameterValue`
   - Change the urdf path (line 14) to use the new xacro file name.
   - Replace
     ```
     with open(urdf,'r') as infp:
         robot_desc=infp.read()
     ```
     with
     ```
     robot_desc=ParameterValue(Command(['xacro ',urdf]),value_type=str)
     ```

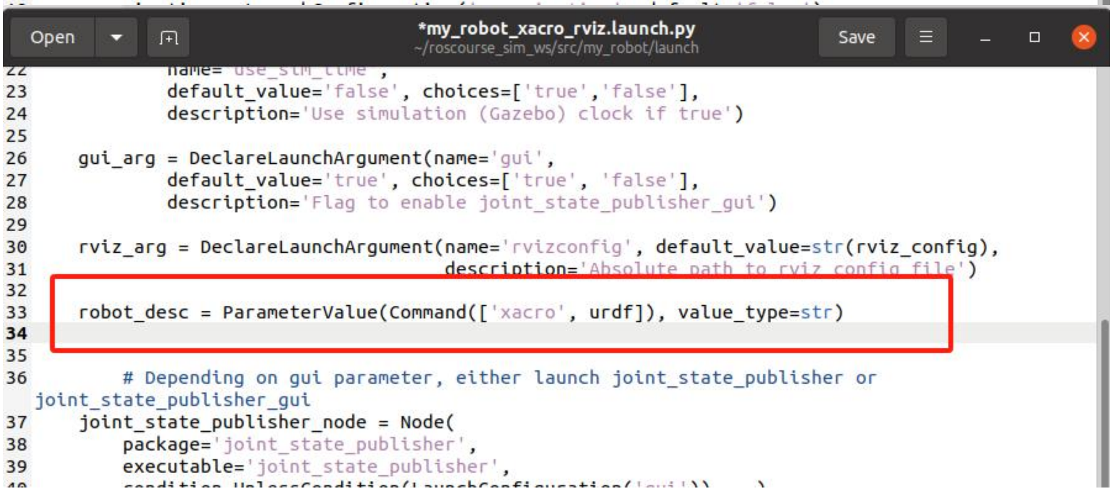
1


```
<?xml version="1.0"?>
<robot name="my_robot"xmlns:xacro="http://www.ros.org/wiki/xacro">
  <material name="green">
      <color rgba="0 0.6 0 1"/>
  </material>

  <material name="white">
      <color rgba="0.8 0.8 0.8 1.0"/>
  </material>

  <material name="gray">
      <color rgba="0.7 0.7 0.7 1.0"/>
  </material>


  <link name="base_link">
    <visual>
```






```
import os
from ament_index_python.packages import get_package_share_directory
from launch import LaunchDescription
from launch.actions import DeclareLaunchArgument
from launch.substitutions import Command, LaunchConfiguration
from launch.conditions import IfCondition, UnlessCondition
from launch_ros.actions import Node
from launch_ros.substitutions import FindPackageShare

from launch_ros.parameter_descriptions import ParameterValue

def generate_launch_description():

    my_robot_path = get_package_share_directory('my_robot')
    urdf = os.path.join(my_robot_path , 'my_robot.urdf.xacro')
    rviz_config = os.path.join(my_robot_path, 'my_robot.rviz')
    #rviz_config = '/home/yahboom/roscourse_sim_ws/src/my_robot/rviz/my_robot.rviz'

```




```
            name='use_sim_time',
            default_value='false', choices=['true','false'],
            description='Use simulation (Gazebo) clock if true')

    gui_arg = DeclareLaunchArgument(name='gui',
            default_value='true', choices=['true', 'false'],
            description='Flag to enable joint_state_publisher_gui')

    rviz_arg = DeclareLaunchArgument(name='rvizconfig', default_value=str(rviz_config),
                                     description='Absolute path to rviz config file')

    robot_desc = ParameterValue(Command(['xacro', urdf]), value_type=str)


        # Depending on gui parameter, either launch joint_state_publisher or
joint_state_publisher_gui
    joint_state_publisher_node = Node(
        package='joint_state_publisher',
        executable='joint_state_publisher',
```

6. **Note:** This ParameterValue(Command ...) will run the xacro command to process the URDF file. This is necessary to convert any math, variables, or functions that we define using xacro into a pure URDF format for Rviz to use.

7. **Note:** We need to indicate what parts of our file require xacro processing. To do this, we use `${expression}`. The dollar sign tells the interpreter that this part contains xacro code. It will evaluate the expression within the curly brackets. Xacro allows us to use some built-in variables such as pi, define our own variables, define macros (essentially functions), and do basic math such as +, -, *, /.

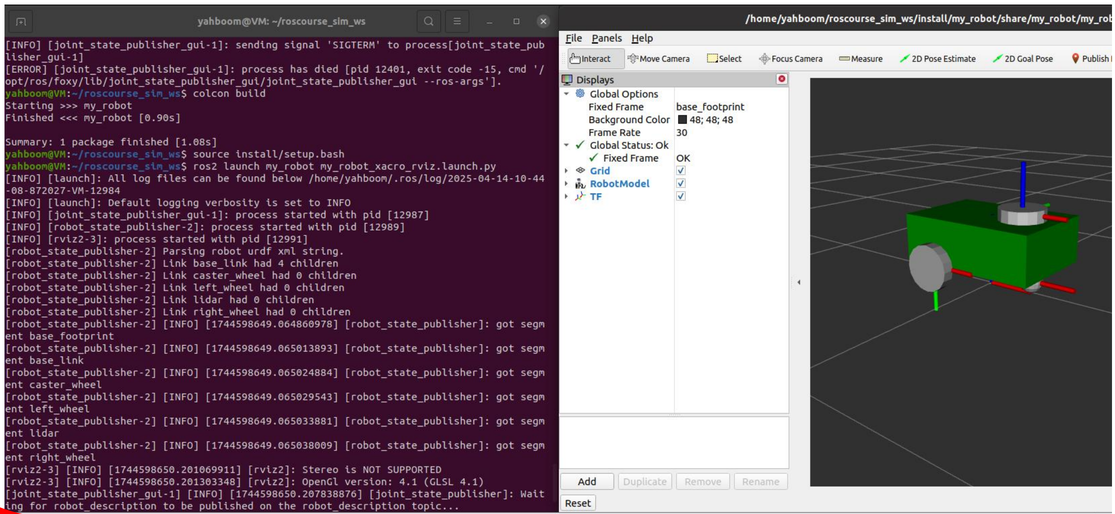
8. **Action:** In our new URDF, replace the -1.57 for the rpy in the wheel joints with `${-pi / 2.0}`. This will give us an exact value rather than our approximation.

```
<joint name="base_left_wheel_joint" type="continuous">
  <parent link="base_link"/>
  <child link="left_wheel"/>
  <origin xyz="-0.15 0.2 0" rpy="${-pi / 2.0} 0 0"/>
  <axis xyz="0 0 1"/>
</joint>

<joint name="base_right_wheel_joint" type="continuous">
  <parent link="base_link"/>
  <child link="right_wheel"/>
  <origin xyz="-0.15 -0.2 0" rpy="${-pi / 2.0} 0 0"/>
  <axis xyz="0 0 1"/>
</joint>
```

9. **Note:** It is good practice to include the decimal (2.0 rather than 2) to ensure math operations provide decimals. It is also good practice to add spacing around the math operators. This helps readability and ensures expressions are interpreted correctly. You can also use parentheses to group terms.

10. **Action:** Build and source. Then run our modified launch file:
    `ros2 launch my_robot my_robot_xacro_rviz.launch.py`

11. **Note:** Everything should look the same as it did at the end of the last lab. It is recommended to periodically <u>rebuild</u> and run this file as changes are made. It is easier to debug when there are fewer changes between runs. If there are errors, running the xacro command (xacro then filename with full path) can give better error feedback such as a line number to look at.



12. **Note:** We can now add some xacro variables that we define. **Place these after the opening robot tag but before any material, link, or joint definition.** This will make them easy to find for modification.

13. **Action:** Create a new xacro variable for the base length by adding
    `<xacro:property name="base_length" value="0.6" />`

14. **Action:** We can now use this variable in our URDF similar to how we used pi. Change the first value of the base_link visual to use ${base_length}

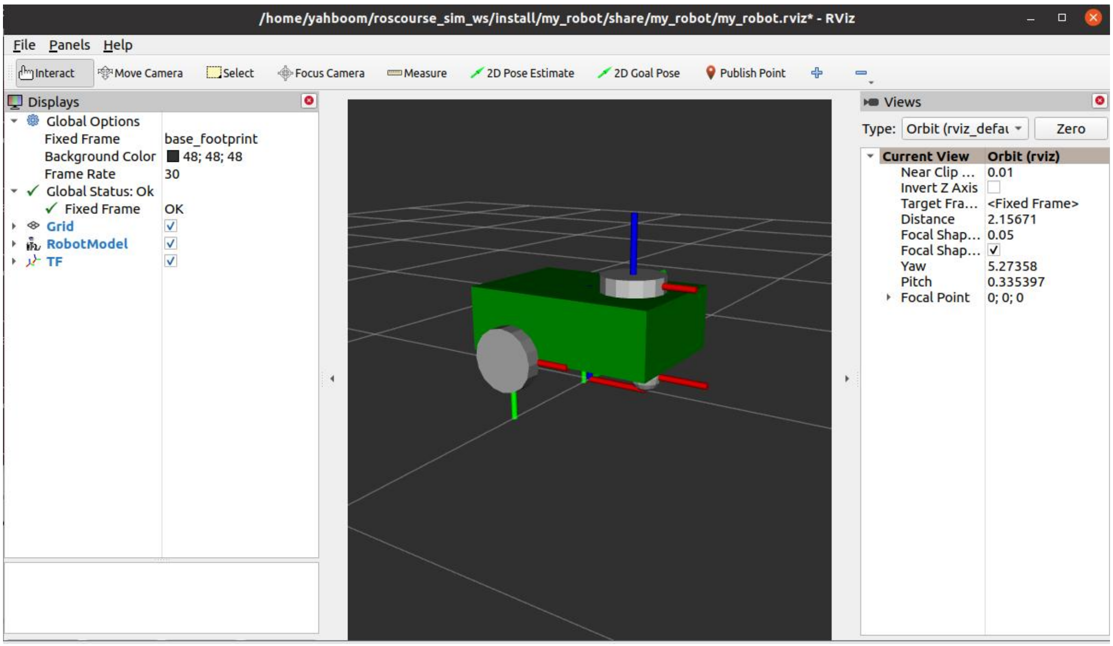
15. **Action:** We also used this value to compute the positions of the wheel and LIDAR joints. Modify the x position of these four joints to use the base_length variable and appropriate mathematical expressions.

Open | *my_robot.urdf.xacro ~/roscourse_sim_ws/src/my_robot/urdf | Save

```xml
<?xml version="1.0"?>
<robot name="my_robot" xmlns:xacro="http://www.ros.org/wiki/xacro">

  <xacro:property name="base_length" value="0.6" />

  <material name="green">
      <color rgba="0 0.6 0 1"/>
  </material>

  <material name="white">
      <color rgba="0.8 0.8 0.8 1.0"/>
  </material>
```

```xml


  <link name="base_link">
    <visual>
      <geometry>
        <box size="${base_length} 0.4 0.2" />
      </geometry>
      <origin xyz="0 0 0.1" rpy="0 0 0" />
      <material name="green"/>
    </visual>
  </link>

```

```xml
<joint name="base_lidar_joint" type="fixed">
  <parent link="base_link"/>
  <child link="lidar"/>
  <origin xyz="${base_length / 4.0} 0 0.225" rpy="0 0 0"/>
</joint>

<joint name="base_left_wheel_joint" type="continuous">
  <parent link="base_link"/>
  <child link="left_wheel"/>
  <origin xyz="${-base_length / 4.0} 0.2 0" rpy="${-pi / 2.0} 0 0"/>
  <axis xyz="0 0 1"/>
</joint>

<joint name="base_right_wheel_joint" type="continuous">
  <parent link="base_link"/>
  <child link="right_wheel"/>
  <origin xyz="${-base_length / 4.0} -0.2 0" rpy="${-pi / 2.0} 0 0"/>
  <axis xyz="0 0 1"/>
</joint>

<joint name="base_caster_wheel_joint" type="fixed">
  <parent link="base_link"/>
  <child link="caster_wheel"/>
  <origin xyz="${base_length / 3.0} 0 -0.05" rpy="0 0 0"/>
</joint>
```



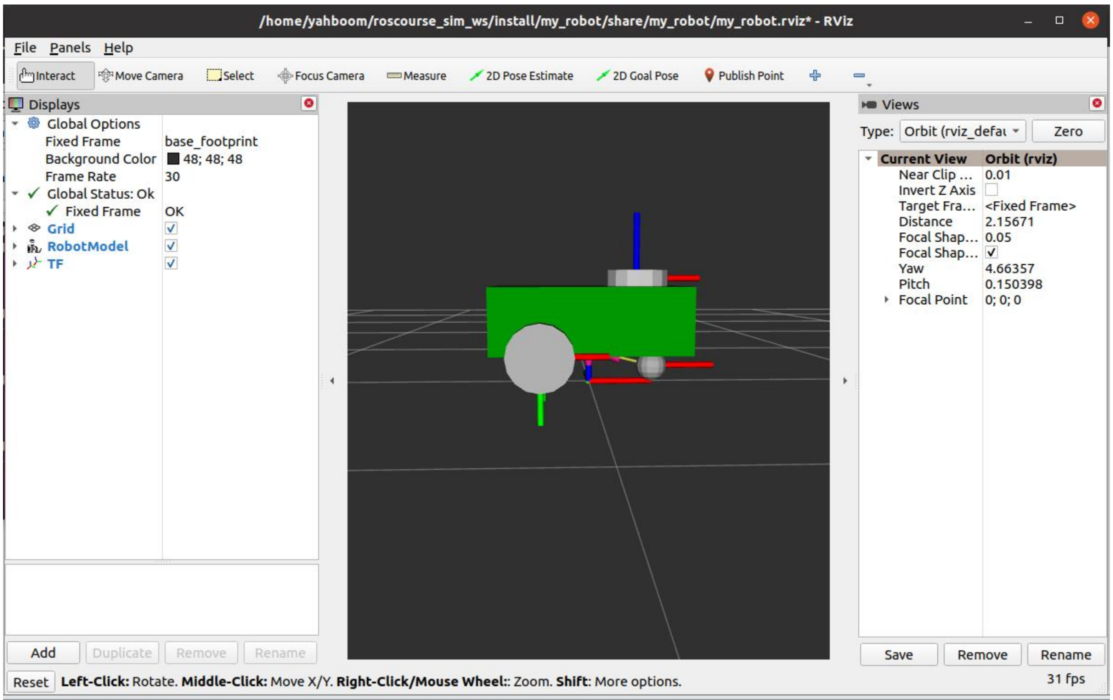
16. **Challenge:** Create additional xacro properties that are the wheel radius, wheel length, base_width, and base_height. Use these variables to update the following:

- Use the variable wheel_radius to determine:
  - left, right, and caster wheel radii
  - base joint height
  - caster wheel joint height
- Use the variable wheel_length to determine:
  - left and right wheel dimension and origin
- Use the variable base_width to determine:
  - left and right wheel joint origins
- Use the variable base_height to determine:
  - base_link origin
  - base_lidar_joint origin

```xml
<link name="base_link">
  <visual>
    <geometry>
      <box size="${base_length} ${base_width} ${base_height}" />
    </geometry>
    <origin xyz="0 0 ${base_height / 2.0}" rpy="0 0 0" />
    <material name="green"/>
  </visual>
</link>

<link name="lidar">
  <visual>
    <geometry>
      <cylinder radius="${wheel_radius}" length="${wheel_length}" />
    </geometry>
    <origin xyz="0 0 0" rpy="0 0 0" />
    <material name="white"/>
  </visual>
</link>

<link name="left_wheel">
  <visual>
    <geometry>
      <cylinder radius="${wheel_radius}" length="${wheel_length}"/>
    </geometry>
    <origin xyz="0 0 ${wheel_length / 2.0}" rpy="0 0 0"/>
    <material name="gray"/>
  </visual>
</link>
```

```xml
<link name="right_wheel">
  <visual>
    <geometry>
      <cylinder radius="${wheel_radius}" length="${wheel_length}"/>
    </geometry>
    <origin xyz="0 0 ${-wheel_length / 2.0}" rpy="0 0 0"/>
    <material name="gray"/>
  </visual>
</link>

<link name="caster_wheel">
  <visual>
    <geometry>
      <sphere radius="${wheel_radius}"/>
    </geometry>
    <origin xyz="0 0 0" rpy="0 0 0"/>
    <material name="gray"/>
  </visual>
</link>


<joint name="base_lidar_joint" type="fixed">
  <parent link="base_link"/>
  <child link="lidar"/>
  <origin xyz="${base_length / 4.0} 0 ${base_height + wheel_length / 2.0}" rpy="0 0 0"/>
</joint>

<joint name="base_left_wheel_joint" type="continuous">
  <parent link="base_link"/>
  <child link="left_wheel"/>
  <origin xyz="${-base_length / 4.0} ${base_width / 2.0} 0" rpy="${-pi / 2.0} 0 0"/>
  <axis xyz="0 0 1"/>
</joint>
```

```xml
<joint name="base_lidar_joint" type="fixed">
  <parent link="base_link"/>
  <child link="lidar"/>
  <origin xyz="${base_length / 4.0} 0 ${base_height + wheel_length / 2.0}" rpy="0 0 0"/>
</joint>

<joint name="base_left_wheel_joint" type="continuous">
  <parent link="base_link"/>
  <child link="left_wheel"/>
  <origin xyz="${-base_length / 4.0} ${base_width / 2.0} 0" rpy="${-pi / 2.0} 0 0"/>
  <axis xyz="0 0 1"/>
</joint>

<joint name="base_right_wheel_joint" type="continuous">
  <parent link="base_link"/>
  <child link="right_wheel"/>
  <origin xyz="${-base_length / 4.0} ${-base_width / 2.0} 0" rpy="${-pi / 2.0} 0 0"/>
  <axis xyz="0 0 1"/>
</joint>

<joint name="base_caster_wheel_joint" type="fixed">
  <parent link="base_link"/>
  <child link="caster_wheel"/>
  <origin xyz="${base_length / 3.0} 0 ${-wheel_radius}" rpy="0 0 0"/>
</joint>

<link name="base_footprint" />

<joint name="base_joint" type="fixed">
  <parent link="base_footprint"/>
  <child link="base_link"/>
  <origin xyz="0 0 ${base_height / 2.0}" rpy="0 0 0"/>
</joint>
```



Task 2 - Add Gazebo Required Tags

1. **Note:** In order to simulate the robot, the links need to have inertia and collision tags. (Some CAD software will allow you to export the inertia matrix. This is useful for custom mesh links.)

Inertial Tags

2. **Note:** The following resources will be useful for this section:
   - http://wiki.ros.org/urdf/Tutorials/Adding%20Physical%20and%20Collision%20Properties%20to%20a%20URDF%20Model
   - https://en.wikipedia.org/wiki/List_of_moments_of_inertia
3. **Note:** If we look at the second resource, we can see how to compute the inertia matrix (3D inertia tensors) for some objects. We are going to use the following: solid sphere, solid cuboid, solid cylinder.
4. **Action:** We are going to define a macro to compute the inertia matrix of a cuboid (box). Add this after the xacro variable definitions:

```
<xacro:macro name="box_inertia" params="mass lx ly lz xyz rpy">
    <inertial>
        <origin xyz="${xyz}" rpy="${rpy}" />
        <mass value="${mass}" />
        <inertia ixx="${(mass / 12.0) * (ly * ly + lz * lz)}" ixy="0" ixz="0"
                 iyy="${(mass / 12.0) * (lx * lx + lz * lz)}" iyz="0"
                 izz="${(mass / 12.0) * (lx * lx + ly * ly)}" />
    </inertial>
</xacro:macro>
```

5. **Note:** Defining a macro is a like defining a function in Python. The name is the function name that we can reference elsewhere in the URDF. The params are the arguments to the function. This macro is going to generate the inertial tag using the supplied params.
6. **Action:** We will use this new macro for the base_link. After the </visual> tag and before the </link> tag, add the following:
```
<xacro:box_inertia mass="5.0" lx="${base_length}" ly="${base_width}"
lz="${base_height}" xyz="0 0 ${base_height / 2.0}" rpy="0 0 0" />
```
7. **Note:** We pulled the xyz, rpy from the visual origin. The xacro tag allows us to use our macro by name and supply the required params. Notice we are able to use our variables to define the params as well.
8. **Challenge:** Create two more macros: One named cylinder_inertia and one sphere_inertia. Use the solid object tensor definitions in the provided resource.
9. **Action:** Use the defined macros to add the inertial tags to each wheel (both sides and the caster). Use mass=1 for right/left wheels and mass=2 for the caster.
10. **Action:** For the LIDAR link, add
```
<xacro:cylinder_inertia mass="1.0" h="0.05" r="0.1" xyz="0 0 0" rpy="0 0 0"/>
```

```xml
<xacro:macro name="box_inertia" params="mass lx ly lz xyz rpy">
  <inertial>
    <origin xyz="${xyz}" rpy="${rpy}" />
    <mass value="${mass}" />
    <inertia ixx="${(mass / 12.0) * (ly * ly + lz * lz)}" ixy="0" ixz="0"
             iyy="${(mass / 12.0) * (lx * lx + lz * lz)}" iyz="0"
             izz="${(mass / 12.0) * (lx * lx + ly * ly)}" />
  </inertial>
</xacro:macro>

<xacro:macro name="cylinder_inertia" params="mass r h xyz rpy">
  <inertial>
    <origin xyz="${xyz}" rpy="${rpy}" />
    <mass value="${mass}" />
    <inertia ixx="${(1/12.0)*mass*(3*r*r + h*h)}" ixy="0" ixz="0"
             iyy="${(1/12.0)*mass*(3*r*r + h*h)}" iyz="0"
             izz="${0.5*mass*r*r}" />
  </inertial>
</xacro:macro>

<xacro:macro name="sphere_inertia" params="mass r xyz rpy">
  <inertial>
    <origin xyz="${xyz}" rpy="${rpy}" />
    <mass value="${mass}" />
    <inertia ixx="${(2/5.0)*mass*r*r}" ixy="0" ixz="0"
             iyy="${(2/5.0)*mass*r*r}" iyz="0"
             izz="${(2/5.0)*mass*r*r}" />
  </inertial>
</xacro:macro>
```

```xml
<link name="base_link">
  <visual>
    <geometry>
      <box size="${base_length} ${base_width} ${base_height}" />
    </geometry>
    <origin xyz="0 0 ${base_height / 2.0}" rpy="0 0 0" />
    <material name="green"/>
  </visual>
  <xacro:box_inertia mass="5.0" lx="${base_length}" ly="${base_width}" lz="${base_height}" xyz="0 0 ${base_height / 2.0}" rpy="0 0 0" />
</link>

<link name="lidar">
  <visual>
    <geometry>
      <cylinder radius="${wheel_radius}" length="${wheel_length}" />
    </geometry>
    <origin xyz="0 0 0" rpy="0 0 0" />
    <material name="white"/>
  </visual>
  <xacro:cylinder_inertia mass="1.0" h="${wheel_length}" r="${wheel_radius}" xyz="0 0 0" rpy="0 0 0"/>
</link>

<link name="left_wheel">
  <visual>
    <geometry>
      <cylinder radius="${wheel_radius}" length="${wheel_length}"/>
    </geometry>
    <origin xyz="0 0 ${wheel_length / 2.0}" rpy="0 0 0"/>
    <material name="gray"/>
  </visual>
  <xacro:cylinder_inertia mass="1.0" r="${wheel_radius}" h="${wheel_length}" xyz="0 0 ${wheel_length / 2.0}" rpy="0 0 0" />
</link>
```

```xml
<link name="left_wheel">
  <visual>
    <geometry>
      <cylinder radius="${wheel_radius}" length="${wheel_length}"/>
    </geometry>
    <origin xyz="0 0 ${wheel_length / 2.0}" rpy="0 0 0"/>
    <material name="gray"/>
  </visual>
  <xacro:cylinder_inertia mass="1.0" r="${wheel_radius}" h="${wheel_length}" xyz="0 0 ${wheel_length / 2.0}" rpy="0 0 0" />
</link>

<link name="right_wheel">
  <visual>
    <geometry>
      <cylinder radius="${wheel_radius}" length="${wheel_length}"/>
    </geometry>
    <origin xyz="0 0 ${-wheel_length / 2.0}" rpy="0 0 0"/>
    <material name="gray"/>
  </visual>
  <xacro:cylinder_inertia mass="1.0" r="${wheel_radius}" h="${wheel_length}" xyz="0 0 ${-wheel_length / 2.0}" rpy="0 0 0" />
</link>

<link name="caster_wheel">
  <visual>
    <geometry>
      <sphere radius="${wheel_length}"/>
    </geometry>
    <origin xyz="0 0 0" rpy="0 0 0"/>
    <material name="gray"/>
  </visual>
  <xacro:sphere_inertia mass="2.0" r="${wheel_length}" xyz="0 0 0" rpy="0 0 0" />
</link>
```

Collision Tags

11. **Note:** Collision geometries should be as simple as possible while still being sufficiently accurate for the application. This is important for links which have complex shapes (e.g. ones defined by meshes). For our case, all of the link visuals are simple geometries, so we can save ourselves a lot of work by copying from them.

12. **Action:** Add to the base_link the following: (by convention place after the visual and before the inertial tags)


```xml
    <collision>
        <geometry>
            <box size="${base_length} ${base_width} ${base_height}" />
        </geometry>
        <origin xyz="0 0 ${base_height / 2.0}" rpy="0 0 0" />
    </collision>

```

13. **Action:** Use this same procedure to add collision tags to the other links (lidar, right/left wheels, caster wheel)

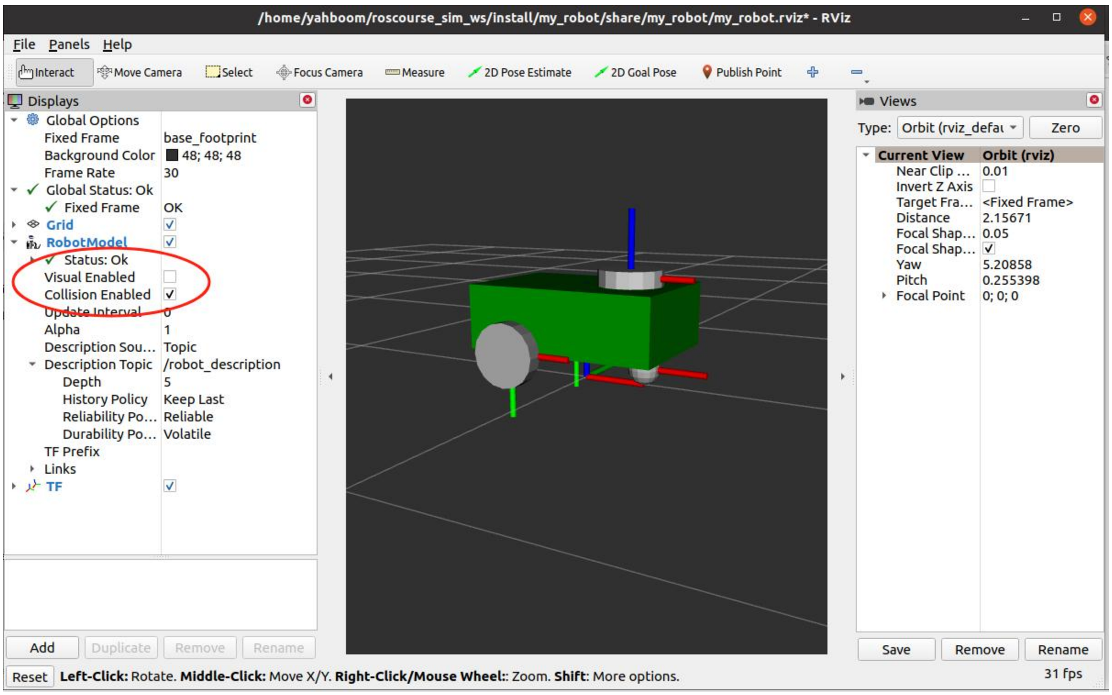
14. **Action:** After building and sourcing, use the my_robot_xacro_rviz.launch.py file to visualize the robot. If you turn off 'Visual Enabled' under 'RobotModel' and turn on 'Collision Enabled', it should look the same. (Note this is unique for our case since both geometries are the same.) It is good practice to do this before simulating to ensure all collision bodies are defined correctly.

```xml
<link name="base_link">
  <visual>
    <geometry>
      <box size="${base_length} ${base_width} ${base_height}" />
    </geometry>
    <origin xyz="0 0 ${base_height / 2.0}" rpy="0 0 0" />
    <material name="green"/>
  </visual>

  <collision>
    <geometry>
      <box size="${base_length} ${base_width} ${base_height}" />
    </geometry>
    <origin xyz="0 0 ${base_height / 2.0}" rpy="0 0 0" />
  </collision>

  <xacro:box_inertia mass="5.0" lx="${base_length}" ly="${base_width}" lz="${base_height}" xyz="0 0 ${base_height / 2.0}" rpy="0 0 0" />
</link>
```

```xml
<link name="lidar">
  <visual>
    <geometry>
      <cylinder radius="${wheel_radius}" length="${wheel_length}" />
    </geometry>
    <origin xyz="0 0 0" rpy="0 0 0" />
    <material name="white"/>
  </visual>

  <collision>
    <geometry>
      <cylinder radius="${wheel_radius}" length="${wheel_length}" />
    </geometry>
    <origin xyz="0 0 0" rpy="0 0 0" />
  </collision>

  <xacro:cylinder_inertia mass="1.0" h="${wheel_length}" r="${wheel_radius}" xyz="0 0 0" rpy="0 0 0"/>
</link>
```

```xml
<link name="left_wheel">
  <visual>
    <geometry>
      <cylinder radius="${wheel_radius}" length="${wheel_length}"/>
    </geometry>
    <origin xyz="0 0 ${wheel_length / 2.0}" rpy="0 0 0"/>
    <material name="gray"/>
  </visual>

  <collision>
    <geometry>
      <cylinder radius="${wheel_radius}" length="${wheel_length}" />
    </geometry>
    <origin xyz="0 0 ${wheel_length / 2.0}" rpy="0 0 0" />
  </collision>

  <xacro:cylinder_inertia mass="1.0" r="${wheel_radius}" h="${wheel_length}" xyz="0 0 ${wheel_length / 2.0}" rpy="0 0 0" />
</link>

<link name="right_wheel">
  <visual>
    <geometry>
      <cylinder radius="${wheel_radius}" length="${wheel_length}"/>
    </geometry>
    <origin xyz="0 0 ${-wheel_length / 2.0}" rpy="0 0 0"/>
    <material name="gray"/>
  </visual>

  <collision>
    <geometry>
      <cylinder radius="${wheel_radius}" length="${wheel_length}" />
    </geometry>
    <origin xyz="0 0 ${-wheel_length / 2.0}" rpy="0 0 0" />
  </collision>

  <xacro:cylinder_inertia mass="1.0" r="${wheel_radius}" h="${wheel_length}" xyz="0 0 ${-wheel_length / 2.0}" rpy="0 0 0" />
</link>
```

```xml
<link name="caster_wheel">
  <visual>
    <geometry>
      <sphere radius="${wheel_length}"/>
    </geometry>
    <origin xyz="0 0 0" rpy="0 0 0"/>
    <material name="gray"/>
  </visual>

  <collision>
    <geometry>
      <sphere radius="${wheel_length}" />
    </geometry>
    <origin xyz="0 0 0" rpy="0 0 0" />
  </collision>

  <xacro:sphere_inertia mass="2.0" r="${wheel_length}" xyz="0 0 0" rpy="0 0 0" />
</link>
```



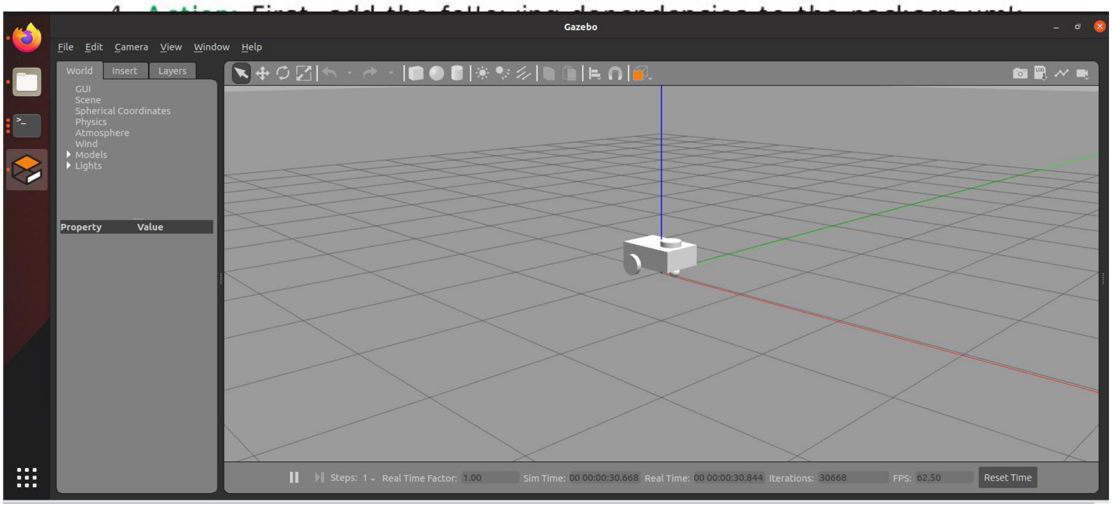
Task 3 - Augmenting the Launch File

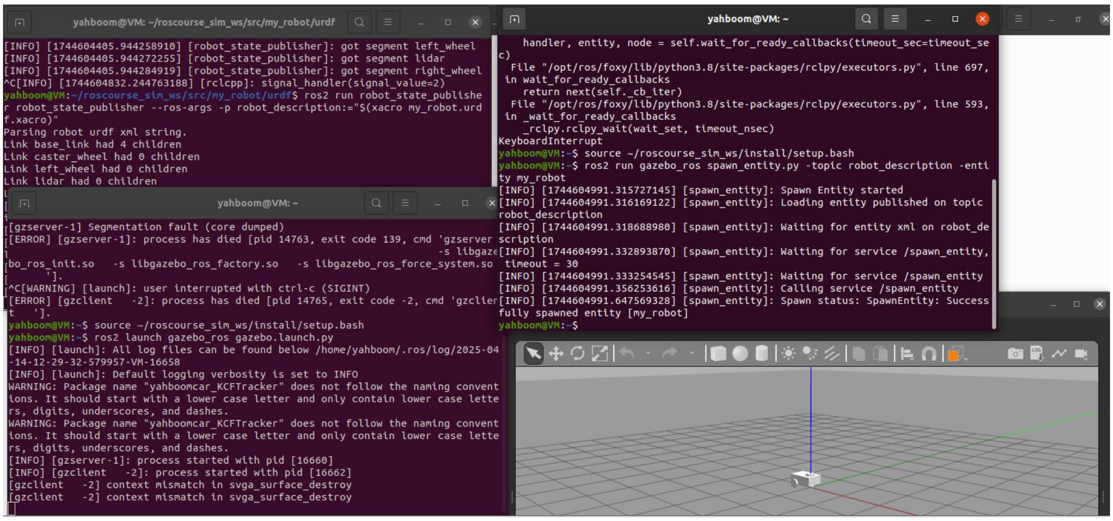
1. **Note:** First we will look at the steps needed to get our model loaded into Gazebo. Then we will modify our launch file to do these steps for us.
   To be able to use Gazebo, we will need to install the necessary packages (needed password is `yahboom`):
   `sudo apt install ros-foxy-gazebo-ros-pkgs`

2. **Action:** We will need three terminals to load our model into Gazebo. Execute the following:

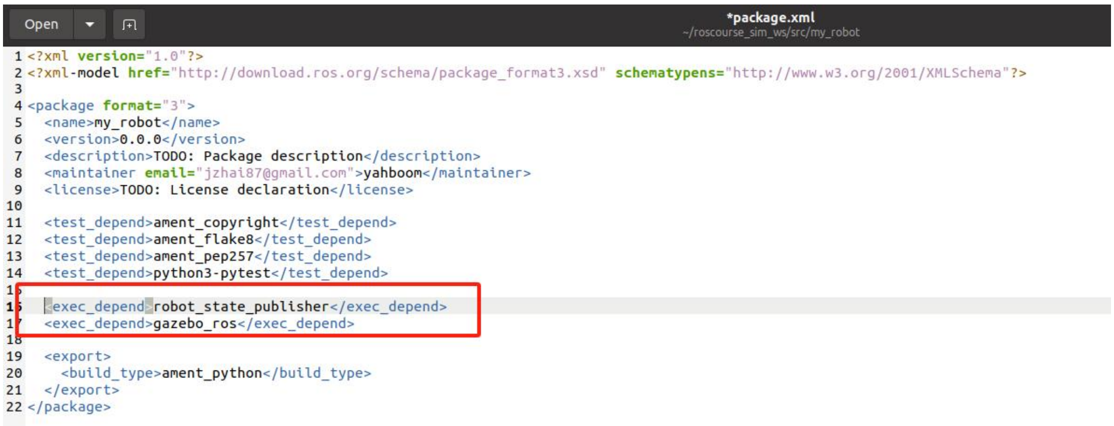
   - In terminal 1: Navigate into the folder containing your URDF and run
     `ros2 run robot_state_publisher robot_state_publisher --ros-args -p robot_description:="$(xacro my_robot.urdf.xacro)"`
     – We saw this package previously with Rviz. This is processing our URDF using the xacro command and publishing the result to the robot_description topic so that it is accessible by other nodes.
   - In terminal 2 run: `ros2 launch gazebo_ros gazebo.launch.py`
     – Note that Gazebo can take a little time to load (especially your first use).
   - In terminal 3: `ros2 run gazebo_ros spawn_entity.py -topic robot_description -entity my_robot`
     – This uses the spawn_entity node in the package gazebo_ros to load a model into the Gazebo environment. This node requires two arguments: the topic where the URDF is being published (check out `ros2 topic list` to verify) and the name of the entity being generated (to use as a reference in Gazebo).
   - **Note:** If the gazebo commands will not run, you may need to install the packages using `sudo apt install ros-foxy-gazebo-ros-pkgs`

3. **Note:** Now that we know all the additional components needed to launch and connect a model to Gazebo, we can augment our launch file.

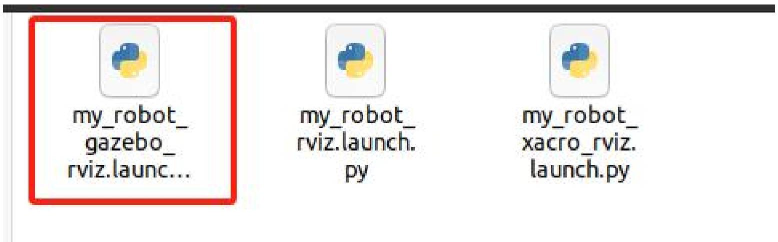




4. **Action:** First, add the following dependencies to the package.xml:
<exec_depend>robot_state_publisher</exec_depend>
<exec_depend>gazebo_ros</exec_depend>



5. **Action:** Create a copy of my_robot_xacro_rviz.launch.py and rename it my_robot_gazebo_rviz.launch.py



6. **Action:** Make the following edits to this file:

   - Remove gui_arg and all parts associated with joint_state_publisher. (We will be using Gazebo data rather than the GUI we had previously in Rviz.)
   - Change use_sim_time default_value to 'true'.
   - Add the following imports so we can call the Gazebo launch file:

```
from launch.launch_description_sources import PythonLaunchDescriptionSour
from launch.actions import IncludeLaunchDescription
```

   - Add the following after the rviz_node definition:

```
gazebo_launch = IncludeLaunchDescription(PythonLaunchDescriptionSource(os.
path.join(get_package_share_directory('gazebo_ros'),'launch/gazebo.launch.
py')))

gazebo_robot = Node(
    package='gazebo_ros',
    executable='spawn_entity.py',
    arguments=['-topic', 'robot_description', '-entity', 'my_robot'])
```

   - Add gazebo_launch and gazebo_robot to the LaunchDescription.

7. **Action:** Rebuild and source then run your new launch file.

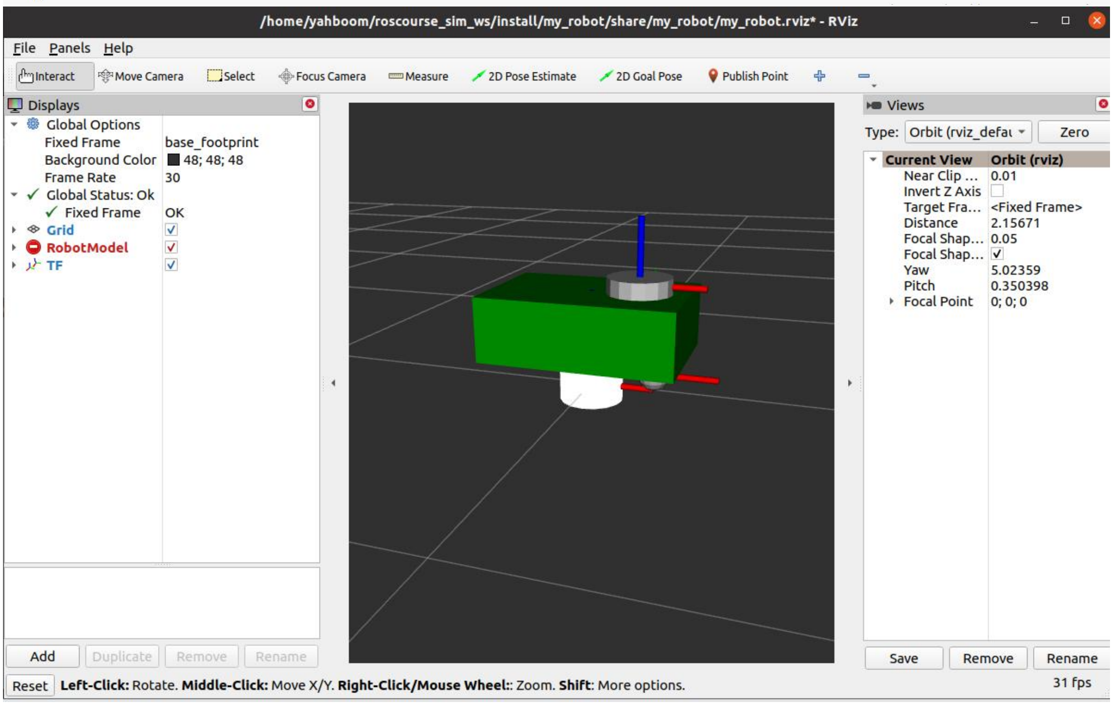
8. **Question:** What happened to the model in Rviz?

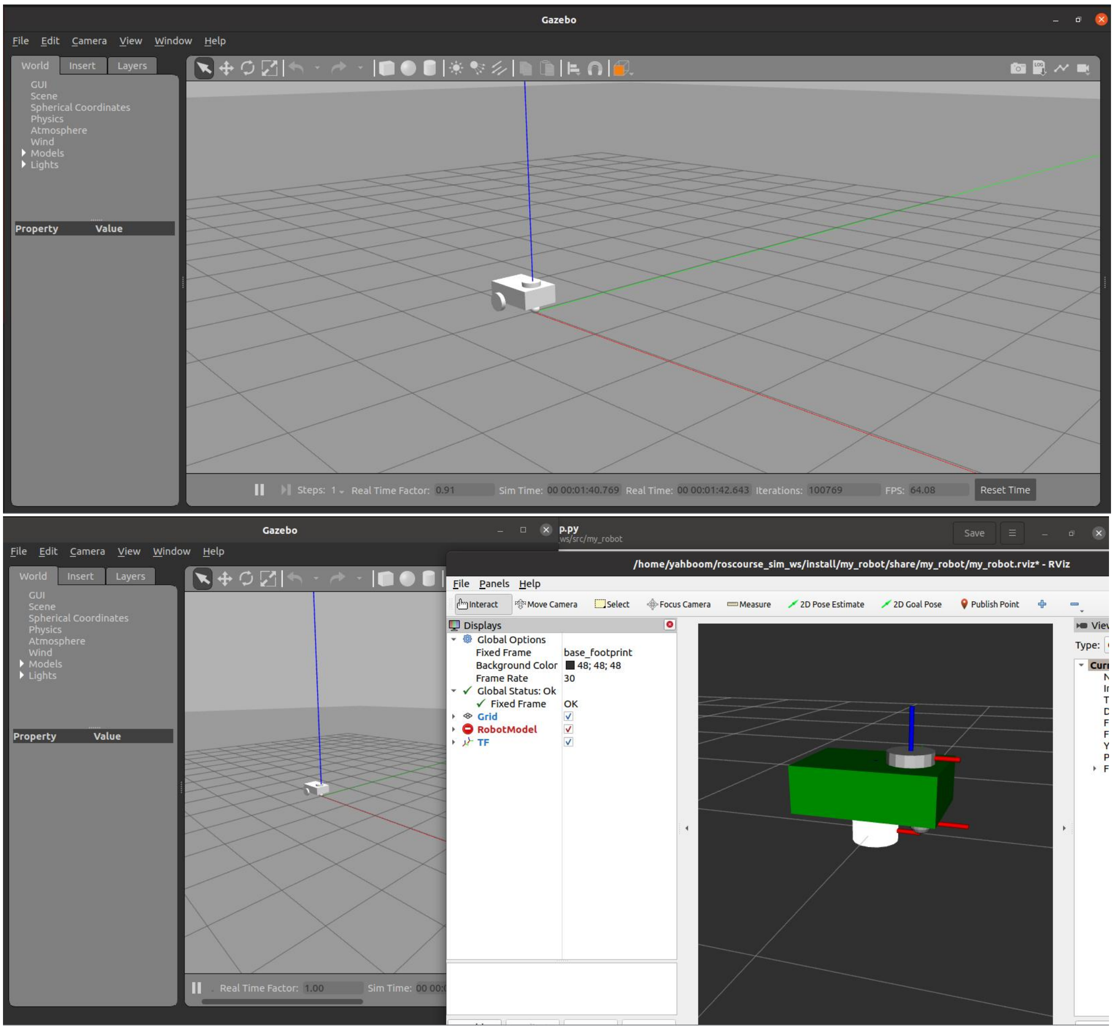
9. **Note:** Since we are not publishing the joint states, the wheel position is ambiguous. Since tf2 cannot compute the transform without the joint information, we have a problem visualizing in Rviz.

```
#    gui_arg = DeclareLaunchArgument(name='gui',
#            default_value='true', choices=['true', 'false'],
#            description='Flag to enable joint_state_publisher_gui')

    rviz_arg = DeclareLaunchArgument(name='rvizconfig', default_value=str(rviz_config),
                                     description='Absolute path to rviz config file')

    robot_desc = ParameterValue(Command(['xacro ',urdf]),value_type=str)


      # Depending on gui parameter, either launch joint_state_publisher or joint_state_publisher_gui
#    joint_state_publisher_node = Node(
#        package='joint_state_publisher',
#        executable='joint_state_publisher',
#        condition=UnlessCondition(LaunchConfiguration('gui'))    )

#    joint_state_publisher_gui_node = Node(
#        package='joint_state_publisher_gui',
#        executable='joint_state_publisher_gui',
#        condition=IfCondition(LaunchConfiguration('gui'))    )

    robot_state_publisher_node = Node(
```

```
    return LaunchDescription([
        sim_time_arg,
#        gui_arg,
        rviz_arg,
#        joint_state_publisher_node,
#        joint_state_publisher_gui_node,
        robot_state_publisher_node,
        rviz_node
    ])
```

```python
import os
from ament_index_python.packages import get_package_share_directory
from launch import LaunchDescription
from launch.actions import DeclareLaunchArgument
from launch.substitutions import Command, LaunchConfiguration
from launch.conditions import IfCondition, UnlessCondition
from launch_ros.actions import Node
from launch_ros.substitutions import FindPackageShare

from launch_ros.parameter_descriptions import ParameterValue

from launch.actions import IncludeLaunchDescription
from launch.launch_description_sources import PythonLaunchDescriptionSource

def generate_launch_description():

    my_robot_path = get_package_share_directory('my_robot')
    urdf = os.path.join(my_robot_path , 'my_robot.urdf.xacro')
    rviz_config = os.path.join(my_robot_path, 'my_robot.rviz')
    #rviz_config = '/home/yahboom/roscourse_sim_ws/src/my_robot/rviz/my_robot.rviz'

    use_sim_time = LaunchConfiguration('use_sim_time', default='true')

    rviz_node = Node(
        package='rviz2',
        executable='rviz2',
        name='rviz2',
        output='screen',
        arguments=['-d', LaunchConfiguration('rvizconfig')])

    gazebo_launch = IncludeLaunchDescription(PythonLaunchDescriptionSource(os.path.join(get_package_share_directory('gazebo_ros'), 'launch', 'gazebo.launch.py')))

    gazebo_robot = Node(
        package='gazebo_ros',
        executable='spawn_entity.py',
        arguments=['-topic', 'robot_description', '-entity', 'my_robot']

    return LaunchDescription([
        sim_time_arg,
#        gui_arg,
        rviz_arg,
#        joint_state_publisher_node,
#        joint_state_publisher_gui_node,
        robot_state_publisher_node,
        rviz_node,

        gazebo_launch,
        gazebo_robot

    ])
```




Gazebo
File Edit Camera View Window Help
World Insert Layers
GUI
Scene
Spherical Coordinates
Physics
Atmosphere
Wind
Models
Lights
Property Value
Steps: 1 Real Time Factor: 0.91 Sim Time: 00 00:01:40.769 Real Time: 00 00:01:42.643 Iterations: 100769 FPS: 64.08 Reset Time
/home/yahboom/roscourse_sim_ws/install/my_robot/share/my_robot/my_robot.rviz* - RViz
File Panels Help
Interact Move Camera Select Focus Camera Measure 2D Pose Estimate 2D Goal Pose Publish Point
Displays
Global Options
Fixed Frame base_footprint
Background Color 48; 48; 48
Frame Rate 30
Global Status: Ok
Fixed Frame OK
Grid
RobotModel
TF

Task 4 - Add Gazebo Materials and Plugins

Materials

1. **Note:** Observe that the model is not colored in Gazebo. Why? It does not recognize the material tags in the URDF. To include materials, we need to include gazebo tags. The Gazebo version of the material tag can contain things like color and surface parameters like friction coefficients. See: https://github.com/gazebosim/gazebo-classic/blob/master/media/materials/scripts/gazebo.material

2. **Action:** Let's put all of the gazebo tags at the end of the URDF (but before </robot>). Add the following:

```xml
<gazebo reference="base_link">
    <material>Gazebo/Green</material>
</gazebo>

```

3. **Note:** Notice that the tag includes a reference attribute. This tells Gazebo which link to apply the material properties to.

4. **Action:** Add material colors for the other links: caster_wheel, left_wheel, right_wheel, and lidar. You can use similar colors to our other material tags or pick new ones.

```xml
<link name="base_footprint" />

<joint name="base_joint" type="fixed">
  <parent link="base_footprint"/>
  <child link="base_link"/>
  <origin xyz="0 0 ${base_height / 2.0}" rpy="0 0 0"/>
</joint>


<gazebo reference="base_link">
  <material>Gazebo/Green</material>
</gazebo>

<gazebo reference="left_wheel">
  <material>Gazebo/Gray</material>
</gazebo>

<gazebo reference="right_wheel">
  <material>Gazebo/Gray</material>
</gazebo>

<gazebo reference="caster_wheel">
  <material>Gazebo/Gray</material>
</gazebo>

<gazebo reference="lidar">
  <material>Gazebo/White</material>
</gazebo>


</robot>
```

Plugins

5. **Note:** To add "hardware" to our simulation, we use plugins. There are some descriptions here: https://classic.gazebosim.org/tutorials?tut=ros_gz-plugins . However, this material is incomplete and some of the information is outdated.

6. **Note:** You can get examples of different plugins and how to use them here: https://github.com/ros-simulation/gazebo_ros_pkgs/blob/ros2/gazebo_plugins/include/gazebo_plugins/gazebo_ros_diff_drive.hpp . Make sure you are in the ros2 branch when you are looking at these .hpp files. The part of interest is where they give example usage. When adding new plugins, copying the example usage and modifying for your purposes is the easiest method to use.

7. **Action:** Add the following to the URDF file after the gazebo material tags.

```
  <gazebo>
    <plugin name="diff_drive_controller" filename="libgazebo_ros_diff_drive.so">
      <!-- Update rate in Hz -->
      <update_rate>50</update_rate>

```



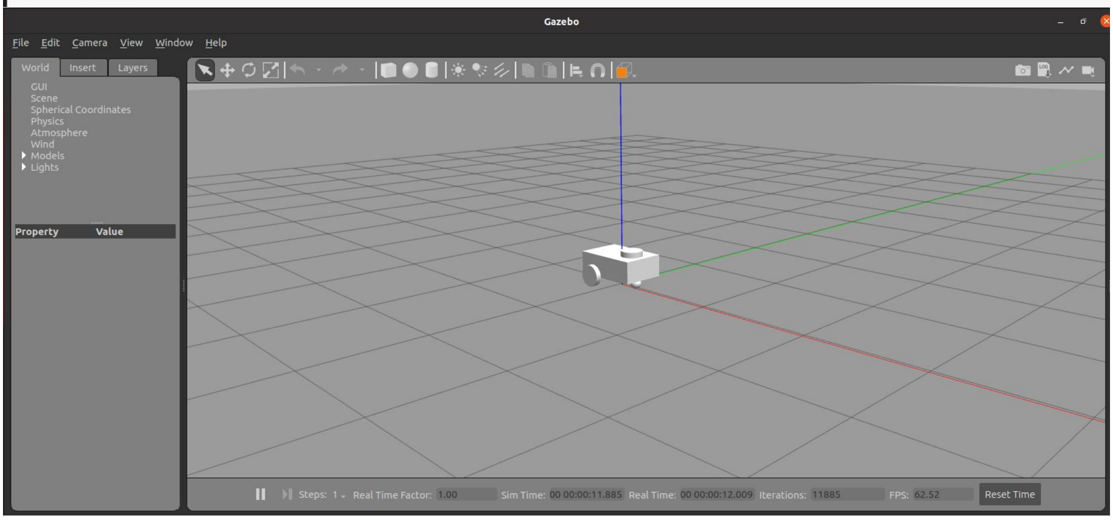
```
      <!-- wheels -->
      <left_joint>base_left_wheel_joint</left_joint>
      <right_joint>base_right_wheel_joint</right_joint>

      <!-- kinematics -->
      <wheel_separation>0.4</wheel_separation>
      <wheel_diameter>0.2</wheel_diameter>

      <!-- output -->
      <publish_odom>true</publish_odom>
      <publish_odom_tf>true</publish_odom_tf>
      <publish_wheel_tf>true</publish_wheel_tf>

      <odometry_topic>odom</odometry_topic>
      <odometry_frame>odom</odometry_frame>
      <robot_base_frame>base_footprint</robot_base_frame>

      <!-- input -->
      <command_topic>cmd_vel</command_topic>

    </plugin>
  </gazebo>

```

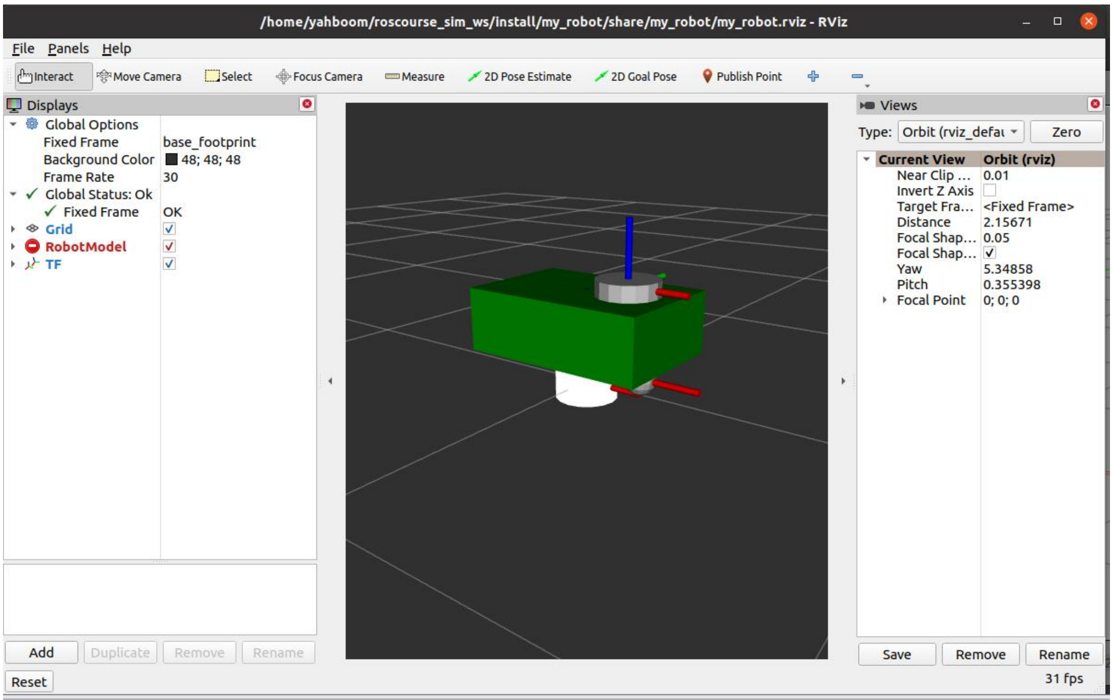
8. **Action:** Rebuild and run the launch file we wrote previously. Note that if you get errors, you may need to add the Gazebo plugin path to your environment variables (a Linux requirement) by inputting `export GAZEBO_PLUGIN_PATH=/opt/ros/foxy/lib` into a terminal.

9. **Action:** Type `ros2 topic list -t`. We can see that the plugin we added gives a new topic called /cmd_vel.

10. **Action:** Move the robot by publishing to this topic. (Reminders: ros2 topic pub, ros2 interface show)

```xml
<gazebo reference="lidar">
  <material>Gazebo/White</material>
</gazebo>

<gazebo>
  <plugin name="diff_drive_controller" filename="libgazebo_ros_diff_drive.so">
    <update_rate>50</update_rate>

    <!-- wheels -->
    <left_joint>base_left_wheel_joint</left_joint>
    <right_joint>base_right_wheel_joint</right_joint>

    <!-- kinematics -->
    <wheel_separation>0.4</wheel_separation>
    <wheel_diameter>0.2</wheel_diameter>

    <!-- output -->
    <publish_odom>true</publish_odom>
    <publish_odom_tf>true</publish_odom_tf>
    <publish_wheel_tf>true</publish_wheel_tf>
    <odometry_topic>odom</odometry_topic>
    <odom_frame>odom</odom_frame>
    <robot_base_frame>base_footprint</robot_base_frame>

    <!-- input -->
    <command_topic>cmd_vel</command_topic>
  </plugin>
</gazebo>


</robot>
```





I'm running into a few issues with my robot model in Gazebo, and I'm starting to think they might be related. First off, there's no color showing up on the robot, even though I added the Gazebo material tags for each link. I'm wondering if maybe the material names like Gazebo/Green or Gazebo/Gray aren't being recognized or applied correctly. On top of that, I'm not seeing the /cmd_vel topic that should be published by the diff_drive_controller plugin. That makes me think the plugin might not be loading properly, even though the .so file exists and the path looks right. Also, the wheels don't seem to be showing up or behaving correctly, and I keep getting those TF errors about left_wheel and right_wheel frames not existing. I'm thinking I should double-check the joint definitions, make sure the links are properly connected, and maybe verify that the robot description is being published correctly with all the necessary transforms.

I will fix this issue as soon as possible and update my deliverables