Simulating the custom robot in Gazebo

To define robot and perform simulation in Gazebo we need to configure simulation components in SDF (Simulation Description Format) which follows standard XML (eXtensible Markup Language) format. Below is the brief description of both.

XML (eXtensible Markup Language)

- A structured language used to store and transport data.
- Uses tags (<tag>...</tag>) to define elements and their hierarchy.
- Commonly used in configuration files, including robot simulations.
- Human-readable and machine-parseable.

SDF (Simulation Description Format)

- An XML-based format specifically designed for robot simulations in Gazebo and Ignition Gazebo.
- Describes worlds, robots, sensors, lights, and physics properties.
- Files typically start with <sdf version="..."> and contain elements like <world>, <model>, and <plugin>.
- Enables realistic and detailed simulation environments.

Defining World and Plugin Systems: Starting with xml and sdf tags, we need to define the world where we will run our robot simulation with some necessary plugins as below.

```xml
<?xml version="1.0" ?>
<sdf version="1.8">
    <world name="car_world">
        <physics name="1ms" type="ignored">
            <max_step_size>0.001</max_step_size>
            <real_time_factor>1.0</real_time_factor>
        </physics>
        <plugin
            filename="libignition-gazebo-physics-system.so"
            name="ignition::gazebo::systems::Physics">
        </plugin>
        <plugin
            filename="libignition-gazebo-user-commands-system.so"
            name="ignition::gazebo::systems::UserCommands">
        </plugin>
        <plugin
            filename="libignition-gazebo-scene-broadcaster-system.so"
            name="ignition::gazebo::systems::SceneBroadcaster">
        </plugin>
```

Explanation:

World Definition (car_world): Sets up a named simulation environment where models, lights, and systems will be placed.

Physics Configuration

- max_step_size is set to 0.001 seconds (1ms) for fine-grained simulation steps.
- real_time_factor of 1.0 ensures the simulation matches real-time speed.

Loaded Plugins

- **Physics System Plugin**: Enables physical behavior like gravity, collisions, and dynamics.
- **User Commands Plugin**: Allows external commands (like model insertion or deletion) during simulation.
- **Scene Broadcaster Plugin**: Sends visual updates of the simulation scene to the GUI for visualization.

Defining Light System: Here we will add and simulate the light for our environment.

```xml
<light type="directional" name="sun">
            <cast_shadows>true</cast_shadows>
            <pose>0 0 10 0 0 0</pose>
            <diffuse>0.8 0.8 0.8 1</diffuse>
            <specular>0.2 0.2 0.2 1</specular>
            <attenuation>
                <range>1000</range>
                <constant>0.9</constant>
                <linear>0.01</linear>
                <quadratic>0.001</quadratic>
            </attenuation>
            <direction>-0.5 0.1 -0.9</direction>
   </light>
```

Explanation:

- Adds **sunlight** to the simulation for natural lighting.
- Enables **shadows** for more realistic visuals.
- Simulates **outdoor light direction and intensity**.

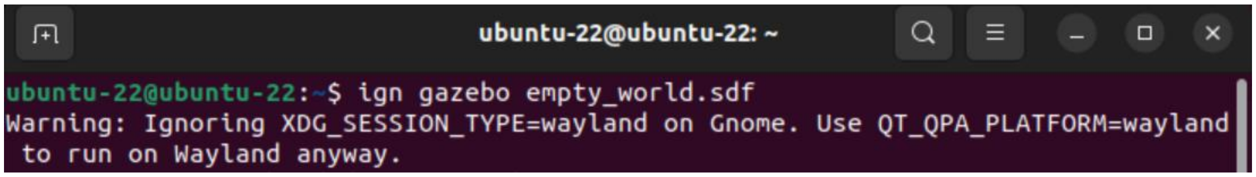
Defining Ground: Now we will define a surface for our robot to move on.

```
<model name="ground_plane">
            <static>true</static>
            <link name="link">
                <collision name="collision">
                <geometry>
                    <plane>
                    <normal>0 0 1</normal>
                    </plane>
                </geometry>
                </collision>
                <visual name="visual">
                <geometry>
                    <plane>
                    <normal>0 0 1</normal>
                    <size>100 100</size>
                    </plane>
                </geometry>
                <material>
                    <ambient>0.8 0.8 0.8 1</ambient>
                    <diffuse>0.8 0.8 0.8 1</diffuse>
                    <specular>0.8 0.8 0.8 1</specular>
                </material>
                </visual>
            </link>
        </model>
    </world>
</sdf>
```

Explanation:

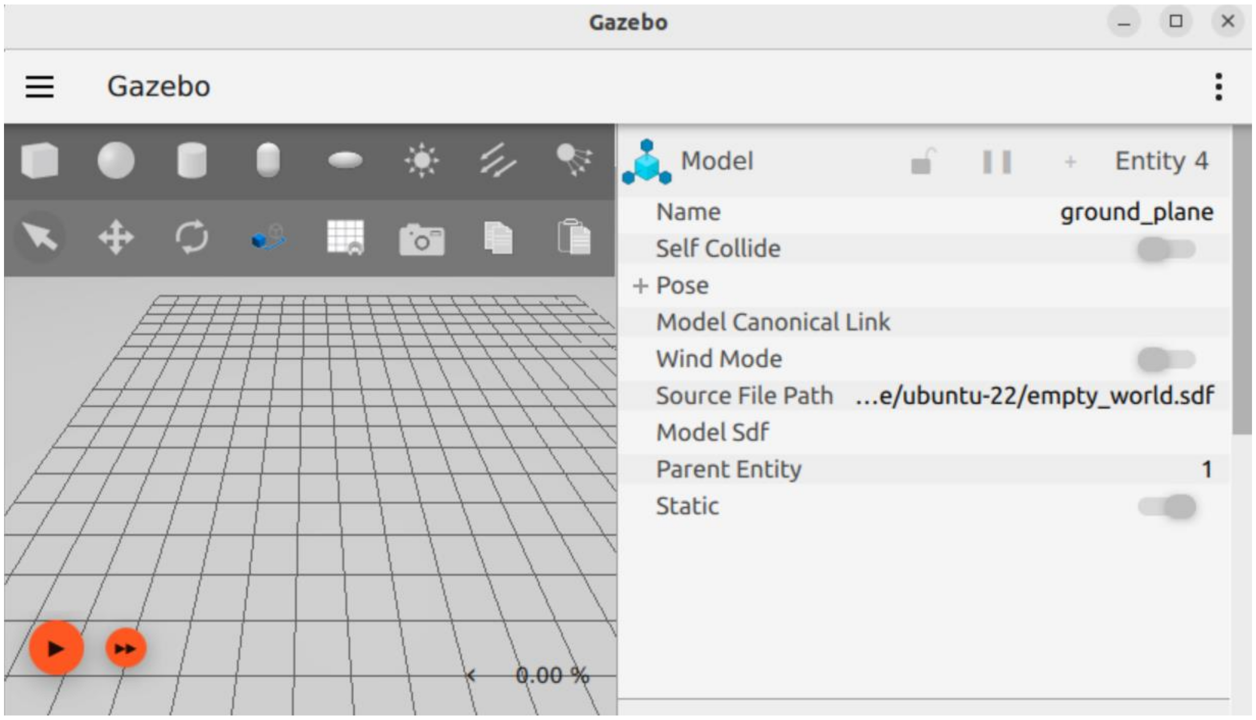
- Creates a **flat ground surface** for the robot or models to move on.
- Defined as **static**, meaning it doesn't move or react to physics.
- Includes **visual and collision properties** so it can be seen and interacted with in simulation.

Running our first simulation: To run and visualize the simulation just run the following command with your respective file name in the terminal below.



```
ubuntu-22@ubuntu-22:~$ ign gazebo empty_world.sdf
Warning: Ignoring XDG_SESSION_TYPE=wayland on Gnome. Use QT_QPA_PLATFORM=wayland
 to run on Wayland anyway.
```

Output: After running following window gazebo window will popup with the simulation configurations we have configured in our respective SDF file. Here we can view and interact with our simulated components.



Name	ground_plane
Self Collide	
+ Pose	
Model Canonical Link	
Wind Mode	
Source File Path	...e/ubuntu-22/empty_world.sdf
Model Sdf	
Parent Entity	1
Static	

Defining Vehicle: We will now configure our custom vehicle in existing SDF file with inertial properties for real-life simulation experience.

```
<model name='vehicle_blue' canonical_link='chassis'>
    <pose relative_to='world'>0 0 0 0 0 0</pose>
<link name='chassis'>
        <pose relative_to='__model__'>0.5 0 0.4 0 0 0</pose>
  <inertial> <!--inertial properties of the link mass, inertia matix-->
        <mass>1.14395</mass>
        <inertia>
            <ixx>0.095329</ixx>
            <ixy>0</ixy>
            <ixz>0</ixz>
            <iyy>0.381317</iyy>
            <iyz>0</iyz>
            <izz>0.476646</izz>
        </inertia>
    </inertial>
```

Explanation:

- **<model>** defines a movable object (e.g., a robot) with the name "vehicle_blue".
- **<pose relative_to='world'>** sets the model's starting position and orientation in the simulation.
- **<link name='chassis'>** represents the main physical body of the vehicle.
- **<pose relative_to='__model__'>** positions the chassis link within the model's own frame.
- **<inertial>** describes the physical properties of the chassis, such as mass and how it resists motion (inertia).

Visual and Collision of Vehicle:

```
    <visual name='visual'>
        <geometry>
            <box>
                <size>2.0 1.0 0.5</size>
            </box>
        </geometry>
        <!--let's add color to our link-->
        <material>
            <ambient>0.0 0.0 1.0 1</ambient>
            <diffuse>0.0 0.0 1.0 1</diffuse>
            <specular>0.0 0.0 1.0 1</specular>
        </material>
    </visual>

<collision name='collision'>
            <geometry>
                <box>
                    <size>2.0 1.0 0.5</size>
                </box>
            </geometry>
        </collision>
    </link>
</model>
```

Explanation

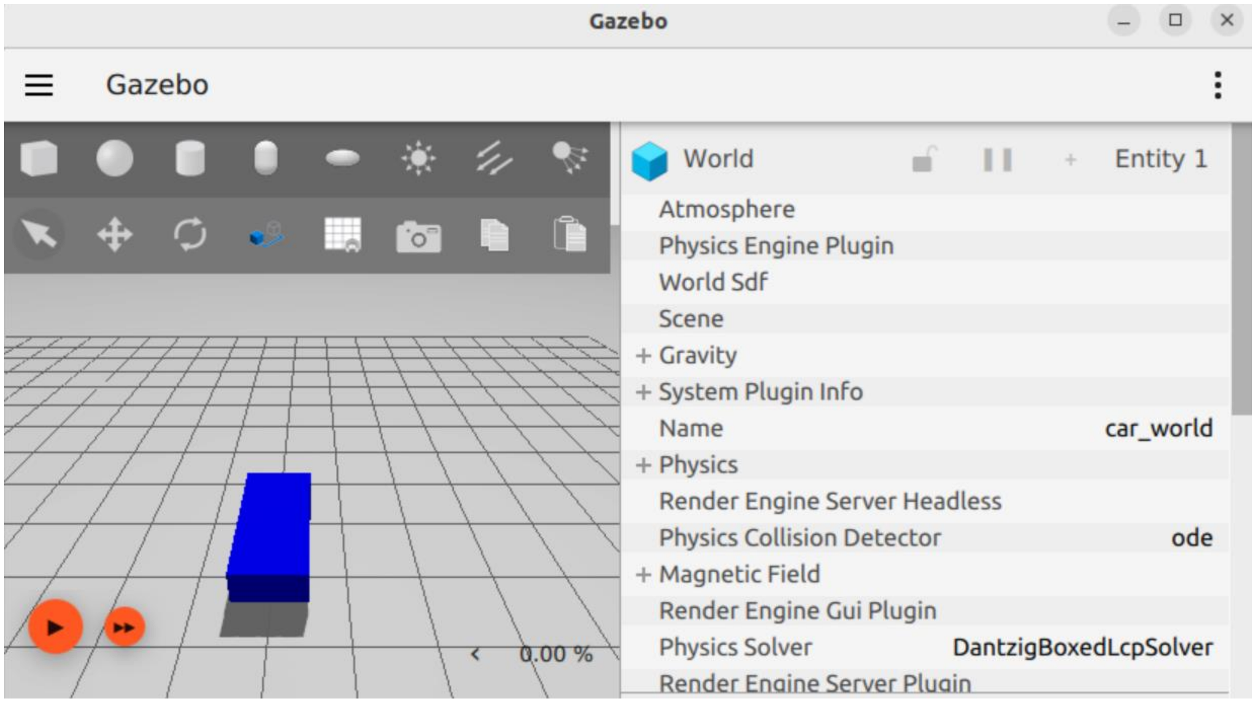
<visual>

- Defines the **visual appearance** of the link (e.g., shape and color).
- **<geometry>** specifies the **shape** like cylinder, circle (in this case, a box with given size).
- **<material>** defines the **color** properties: **ambient, diffuse, specular** control how the object reflects light (in this case, all set to blue).

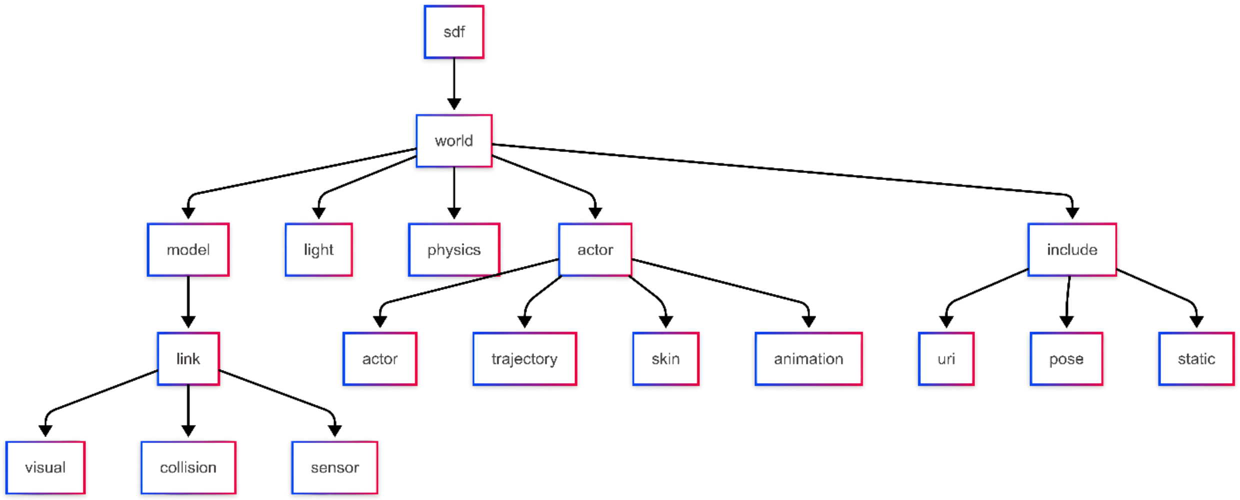
<collision>

- Specifies the **collision properties** of the link.
- Same shape and size as the visual part, but used for **physics calculations** (e.g., detecting when the object hits something).
- The **geometry** defined here is used to simulate the object's interactions in the physics engine, not its appearance.

Running simulation with chassis:



Atmosphere	
Physics Engine Plugin	
World Sdf	
Scene	
+ Gravity	
+ System Plugin Info	
Name	car_world
+ Physics	
Render Engine Server Headless	
Physics Collision Detector	ode
+ Magnetic Field	
Render Engine Gui Plugin	
Physics Solver	DantzigBoxedLcpSolver
Render Engine Server Plugin	



Defining Wheels and their joints: We will simply define two wheels here as a simple differential drive robot.

Defining Left Wheel

```
<link name='left_wheel'>
    <pose relative_to="chassis">-0.5 0.6 0 -1.5707 0 0</pose>
    <inertial>
        <mass>1</mass>
        <inertia>
            <ixx>0.043333</ixx>
            <ixy>0</ixy>
            <ixz>0</ixz>
            <iyy>0.043333</iyy>
            <iyz>0</iyz>
            <izz>0.08</izz>
        </inertia>
    </inertial>

<visual name='visual'>
        <geometry>
            <cylinder>
                <radius>0.4</radius>
                <length>0.2</length>
            </cylinder>
        </geometry>
        <material>
            <ambient>1.0 0.0 0.0 1</ambient>
            <diffuse>1.0 0.0 0.0 1</diffuse>
            <specular>1.0 0.0 0.0 1</specular>
        </material>
    </visual>
    <collision name='collision'>
        <geometry>
            <cylinder>
                <radius>0.4</radius>
                <length>0.2</length>
            </cylinder>
        </geometry>
    </collision>
</link>
```

Explanation: We defined link for left wheel as same for chassis, linked to chassis with some inertia, red material, visual and respective collision.

Defining Right Wheel

```xml
<!--The same as left wheel but with different position-->
<link name='right_wheel'>
    <pose relative_to="chassis">-0.5 -0.6 0 -1.5707 0 0</pose> <!--angles are
in radian-->
    <inertial>
        <mass>1</mass>
        <inertia>
            <ixx>0.043333</ixx>
            <ixy>0</ixy>
            <ixz>0</ixz>
            <iyy>0.043333</iyy>
            <iyz>0</iyz>
            <izz>0.08</izz>
        </inertia>
    </inertial>

    <visual name='visual'>
        <geometry>
            <cylinder>
                <radius>0.4</radius>
                <length>0.2</length>
            </cylinder>
        </geometry>
        <material>
            <ambient>1.0 0.0 0.0 1</ambient>
            <diffuse>1.0 0.0 0.0 1</diffuse>
            <specular>1.0 0.0 0.0 1</specular>
        </material>
    </visual>

    <collision name='collision'>
        <geometry>
            <cylinder>
                <radius>0.4</radius>
                <length>0.2</length>
            </cylinder>
        </geometry>
    </collision>
</link>
```

Adding frame for Caster wheel:

```xml
<frame name="caster_frame" attached_to='chassis'>
    <pose>0.8 0 -0.2 0 0 0</pose>
</frame>
```

Adding link for Caster wheel:

```
<link name='caster'>
    <pose relative_to='caster_frame'/>
    <inertial>
        <mass>1</mass>
        <inertia>
            <ixx>0.016</ixx>
            <ixy>0</ixy>
            <ixz>0</ixz>
            <iyy>0.016</iyy>
            <iyz>0</iyz>
            <izz>0.016</izz>
        </inertia>
    </inertial>
    <visual name='visual'>
        <geometry>
            <sphere>
                <radius>0.2</radius>
            </sphere>
        </geometry>
        <material>
            <ambient>0.0 1 0.0 1</ambient>
            <diffuse>0.0 1 0.0 1</diffuse>
            <specular>0.0 1 0.0 1</specular>
        </material>
    </visual>
    <collision name='collision'>
        <geometry>
            <sphere>
                <radius>0.2</radius>
            </sphere>
        </geometry>
    </collision>
</link>
```

Connecting links using joints: We need to define the joints to connect the links for caster, left and right wheel.

Right Wheel Joint:

```
<joint name='right_wheel_joint' type='revolute'>
    <pose relative_to='right_wheel'/>
    <parent>chassis</parent>
    <child>right_wheel</child>
    <axis>
        <xyz expressed_in='__model__'>0 1 0</xyz>
    </axis>
</joint>
```

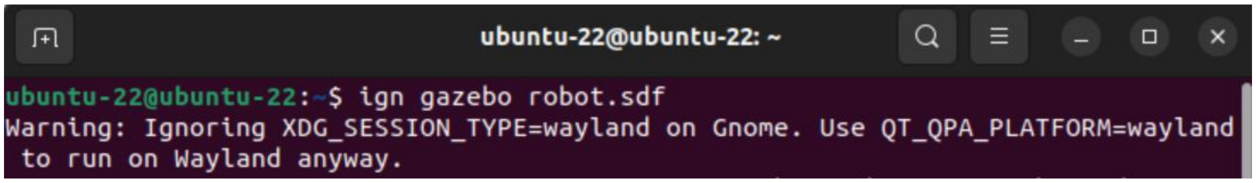
Left Wheel Joint

```
<joint name='left_wheel_joint' type='revolute'>
    <pose relative_to='left_wheel'/>
    <parent>chassis</parent>
    <child>left_wheel</child>
    <axis>
        <xyz expressed_in='__model__'>0 1 0</xyz>
    </axis>
</joint>
```

Caster Joint

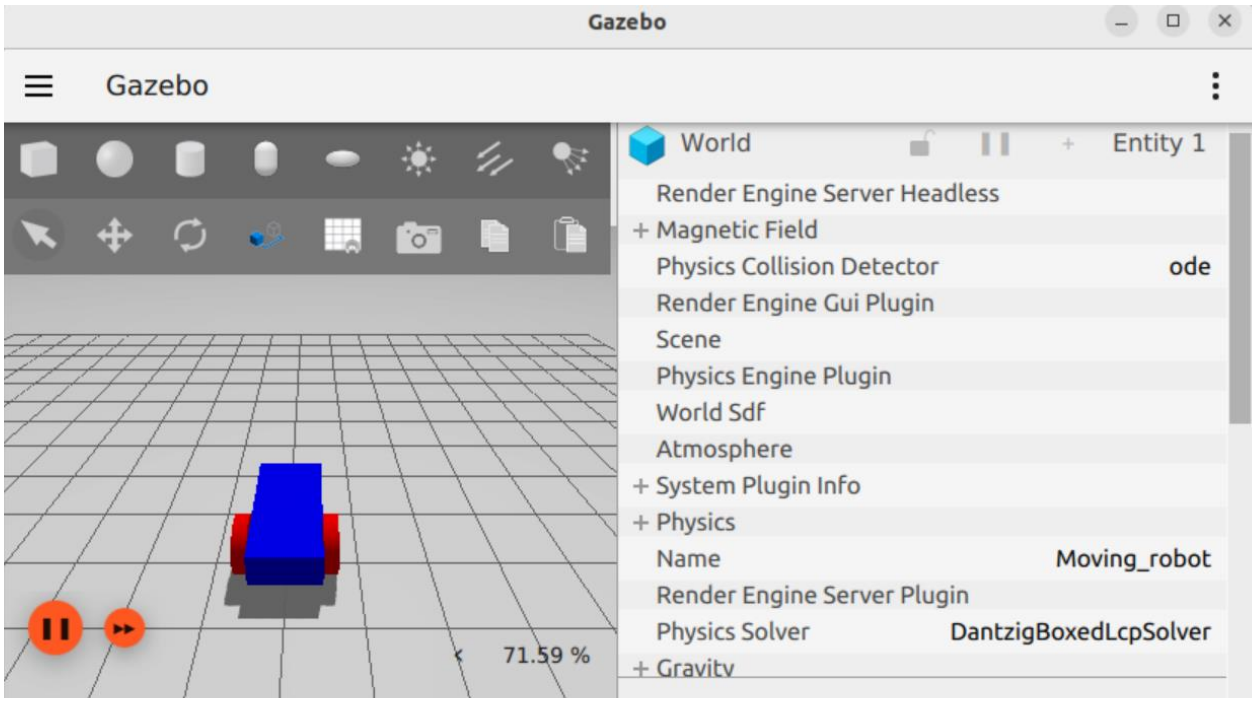
```
<joint name='caster_wheel' type='ball'>
    <parent>chassis</parent>
    <child>caster</child>
</joint>
```

Running simulation with complete robot:



```
ubuntu-22@ubuntu-22:~$ ign gazebo robot.sdf
Warning: Ignoring XDG_SESSION_TYPE=wayland on Gnome. Use QT_QPA_PLATFORM=wayland
 to run on Wayland anyway.
```

Output:



Render Engine Server Headless	
+ Magnetic Field	
Physics Collision Detector	ode
Render Engine Gui Plugin	
Scene	
Physics Engine Plugin	
World Sdf	
Atmosphere	
+ System Plugin Info	
+ Physics	
Name	Moving_robot
Render Engine Server Plugin	
Physics Solver	DantzigBoxedLcpSolver
+ Gravity	

Reference: https://gazebosim.org/docs/fortress/building_robot/

Github: https://github.com/gazebosim/docs/tree/master/fortress/tutorials