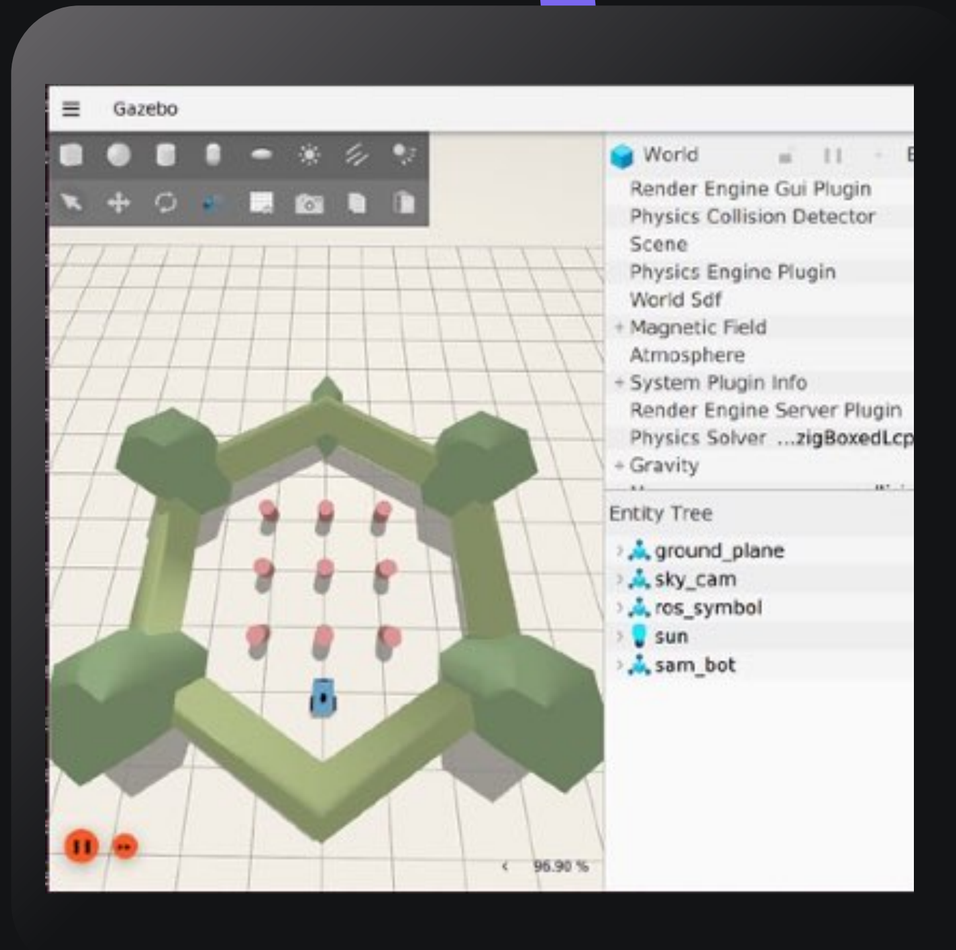
Getting Started with Gazebo Fortress: Installation & Setup

By Fiona Opiyo

21 August 2025

Introduction to Simulation

Robotics simulation allows us to test robots in a safe and repeatable virtual environment before working with real hardware.



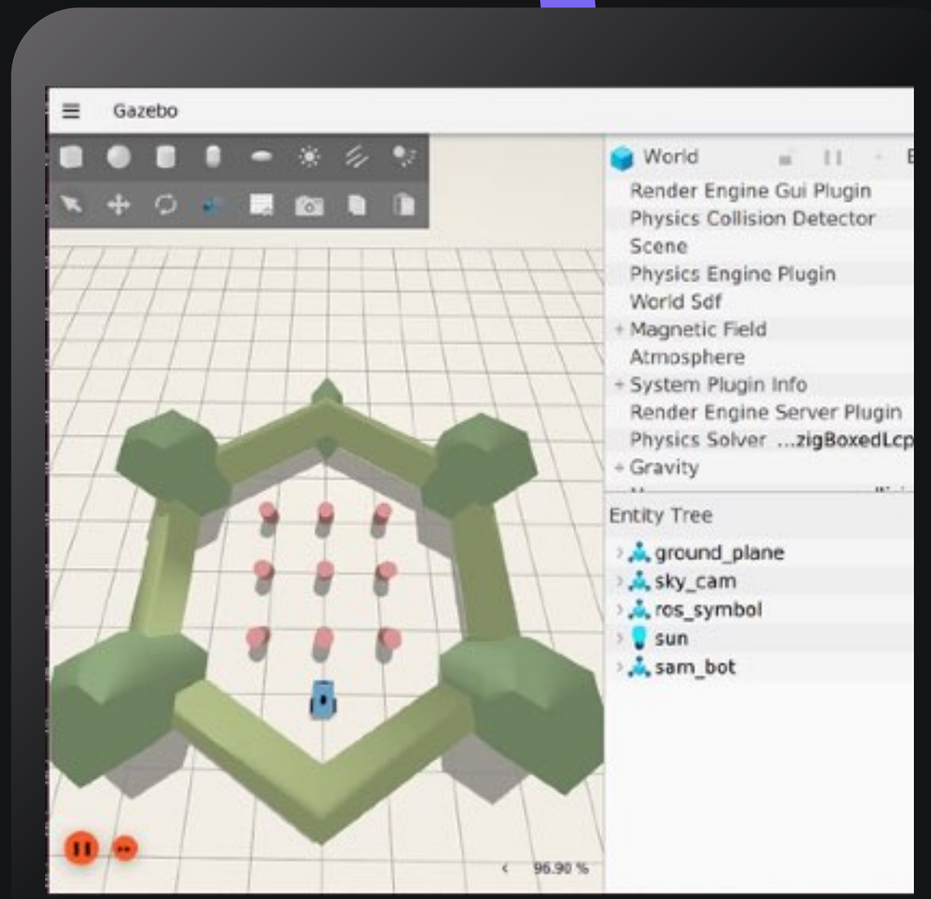
Benefits:

- Prevents hardware damage during early testing
- Scales easily to multiple robots and scenarios
- Saves costs by reducing trial-and-error on physical robots

Introduction to Simulation

Prevents hardware damage

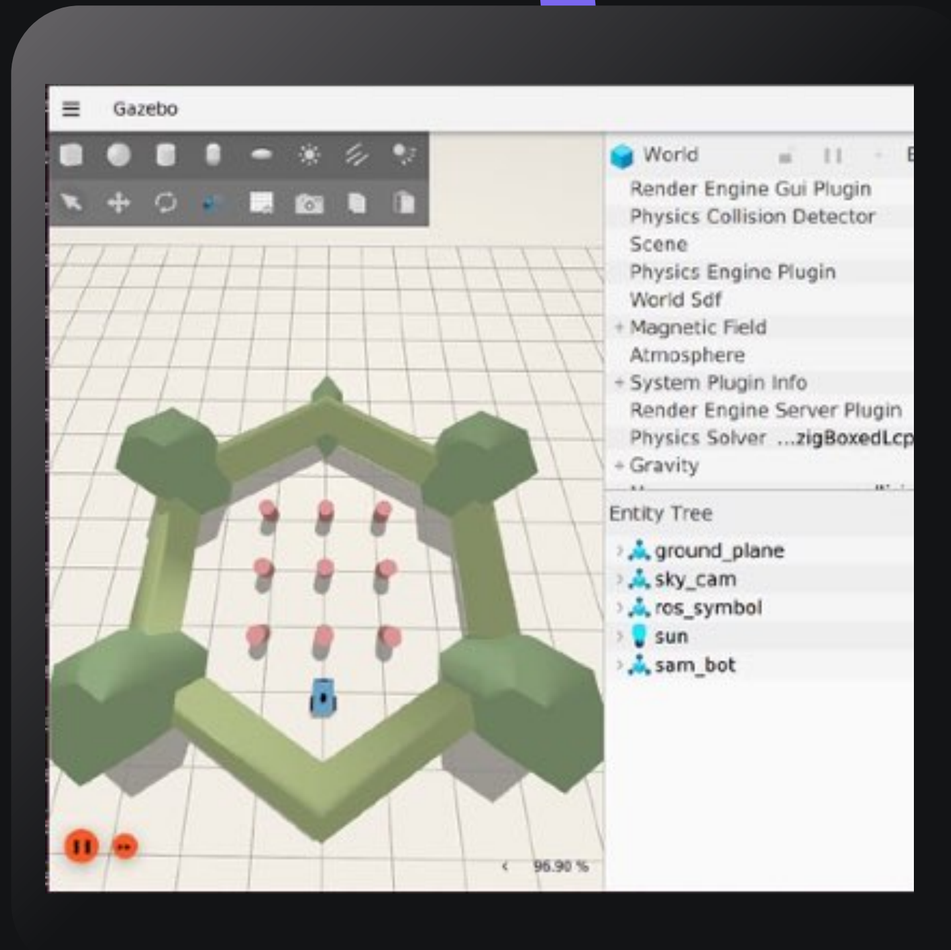
→ Imagine testing a new driving code on your robot, but instead of moving smoothly, it slams into a wall. In real life, that could break your sensors or wheels. In simulation, nothing gets damaged—you just restart.



Introduction to Simulation

Scales easily

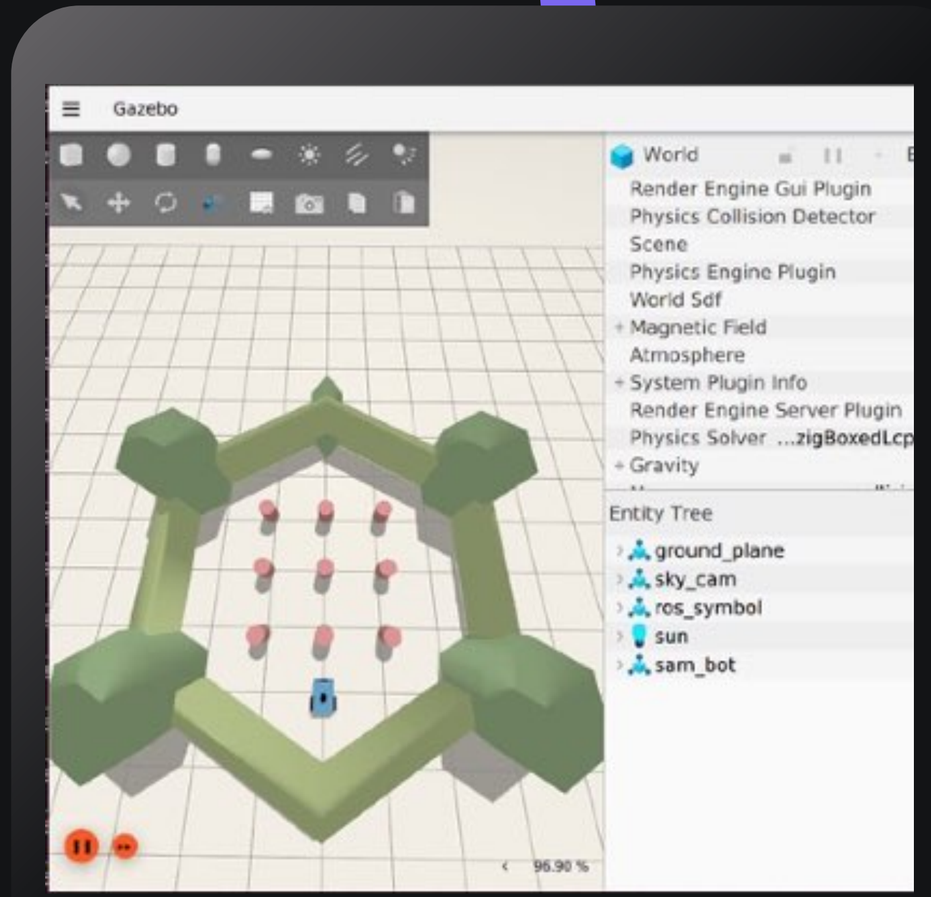
→ With real robots, you’d need to buy 5 robots to test team coordination. In simulation, you just copy-paste the robot model and suddenly you have 5 robots running together.



Introduction to Simulation

Saves Money

→ Every crash, broken part, or drained battery costs money and time. By testing in simulation first, you find mistakes early. When you finally move to the real robot, things work better and you spend less fixing problems.



Why Use Gazebo?

Gazebo is an open-source, high-fidelity robotics simulator.

- Supports advanced physics engines: ODE, Bullet, DART
- Ideal for testing: SLAM, navigation, manipulation, multi-robot systems
- Integrates directly with ROS 2 nodes and topics, bridging real and simulated robots

Why Use Gazebo?

ODE (Open Dynamics Engine)

- Focuses on rigid body dynamics (collisions, contacts, joints).
- Lightweight, stable for general robotics simulation.
- Commonly used for wheeled robots and simple articulated systems.

https://classic.gazebosim.org/tutorials?tut=physics_params

Why Use Gazebo?

Bullet Physics

- Known for real-time simulation and gaming use.
- Handles rigid and soft bodies (cloth, deformables).
- Better at handling complex contact dynamics and high-speed collisions.

Why Use Gazebo?

DART (Dynamic Animation and Robotics Toolkit)

- A modern physics engine integrated in Gazebo Ignition/Fortress.
- Provides accurate rigid body dynamics and articulated robot simulation.
- Strong in handling complex kinematics, constraint solving, and gradient-based optimization (useful for control and planning research).
- Often chosen when you need higher accuracy than ODE and Bullet, especially for humanoids or manipulators.

https://dartsim.github.io

Define geometry, sensors, actuators, and dynamics

- Build the robot’s shape (links, joints, collisions, visuals).
- Attach sensors (LiDAR, camera, IMU) for perception.
- Add actuators (motors, servos) for motion.
- Specify dynamics (mass, inertia, friction) for realistic behavior.

Supports LiDAR Sensors, Cameras, Motors

- Simulate LiDAR scans for mapping & obstacle detection.
- Use cameras for vision-based tasks.
- Test motors/actuators for movement control.
- Extend with IMUs, GPS, force/torque sensors, and others.

ROS 2 Integration

- Simulation publishes data on ROS 2 topics (e.g., /scan, /camera).
- Robots can be controlled via ROS 2 nodes as if they were real.
- Bridges the gap between simulation and real-world deployment.

Why Fortress?

- Long-Term Support (LTS) for stability till 2027
- Advanced physics & rendering for realism
- Rich sensor support out of the box
- Designed for ROS 2 Humble integration
- Supports multi-robot scalability

Gazebo versions

This table includes all currently supported versions of ROS and Gazebo. All other ROS and Gazebo releases are end of life and we do not recommend their continued use.

	OZ Citadel (LTS)	OZ Fortress (LTS)	OZ Garden	OZ Harmonic (LTS)	Oz Ionic
ROS 2 Rolling	❌	❌	⚡	⚡	✅
ROS 2 Jazzy (LTS)	❌	❌	⚡	✅	❌
ROS 2 Iron	❌	✅	⚡	⚡	❌
ROS 2 Humble (LTS)	❌	✅	⚡	⚡	❌
ROS 2 Foxy (LTS)	✅	❌	❌	❌	❌
ROS 1 Noetic (LTS)	✅	⚡	❌	❌	❌

- ✅ - Recommended combination
- ❌ - Incompatible / not possible.
- ⚡ - Possible, *but use with caution*. These combinations of ROS and Gazebo can be made to work together, but some effort is required.

Robots in Gazebo

Robots in Gazebo are described using SDF (Simulation Description Format).

URDF vs SDF (Robot Descriptions)

URDF (Unified Robot Description Format):
Defines robot structure only

```
<robot name="demo_bot">
  <link name="base_link"/>
  <joint name="base_to_wheel" type="continuous">
    <parent link="base_link"/>
    <child link="wheel_link"/>
  </joint>
</robot>
```

URDF vs SDF (Robot Descriptions)

SDF (Simulation Description Format): Defines robots + environments

```
<sdf version="1.8">
  <model name="demo_bot">
    <link name="base_link"/>
    <joint name="base_to_wheel" type="continuous">
      <parent>base_link</parent>
      <child>wheel_link</child>
    </joint>
  </model>
</sdf>
```

Difference in transformation representation

URDF defines transformations inside the joint element, while SDF splits them between the parent and child links. We'll see an example in the next slide.

EXAMPLE OF URDF

- The <origin> inside the <joint> element specifies the transform between the parent and child links.

```
<link name="base_link">
    <visual>
    <origin xyz="0 0 0.05" rpy="0 0 0"/>
    <geometry>
        <box size="2.5 1.5 0.1" />
    </geometry>
    <material name="green">
        <color rgba="0.2 1 0.2 1"/>
    </material>
    </visual>

    <collision>
        <origin xyz="0 0 0.05" rpy="0 0 0"/>
        <geometry>
            <box size="2.5 1.5 0.1" />
        </geometry>
    </collision>

    <inertial>
        <origin xyz="0 0 0.05" rpy="0 0 0"/>
        <mass value="12" />
        <inertia ixx="2.26" ixy="0.0" ixz="0.0" iyy="6.26" iyz="0.0" izz="8.5" /
    </inertial>
</link>

<joint name="slider_joint" type="prismatic">
    <origin xyz="-1.25 0 0.1" rpy="0 0 0"/>
    <parent link="base_link"/>
    <child link="slider_link"/>
    <axis xyz="1 0 0"/>
    <limit lower="0" upper="2" velocity="100" effort="100"/>
</joint>
```

EXAMPLE OF SDF

- Each <link> has its own <pose> element, so the link’s transform is expressed relative to its parent (or the model frame, depending on use_parent_model_frame).

- The <joint> itself in SDF doesn’t carry the transform — it just connects the parent and child links.

```
<link name='base'>
  <inertial>
    <mass>4</mass>
    <inertia>
      <ixx>0.00610633</ixx>
      <ixy>0</ixy>
      <ixz>0</ixz>
      <iyy>0.00610633</iyy>
      <iyz>0</iyz>
      <izz>0.01125</izz>
    </inertia>
  </inertial>
  <collision name='collision'>
  <pose>0 0 0.019 0 -0 0</pose>
    <geometry>
      <cylinder>
        <radius>0.075</radius>
        <length>0.038</length>
      </cylinder>
    </geometry>
  </collision>
  <visual name='visual'>
    <geometry>
      <mesh>
        <uri>model://ur10/meshes/base.dae</uri>
      </mesh>
    </geometry>
  </visual>
</link>

<joint name='wrist_1_wrist_2' type='revolute'>
  <child>wrist_2</child>
  <parent>wrist_1</parent>
  <axis>
    <xyz>3.58979e-09 0 -1</xyz>
    <limit>
      <lower>-6.28319</lower>
      <upper>6.28319</upper>
      <effort>54</effort>
      <velocity>3.2</velocity>
    </limit>
    <use_parent_model_frame>1</use_parent_model_frame>
  </axis>
</joint>
```

Difference in transformation representation

```
<!-- URDF -->
<joint name="slider_joint" type="prismatic">
    <parent link="base_link"/>
    <child link="slider_link"/>
    <origin xyz="-1.25 0 0.1" rpy="0 0 0"/>
</joint>

<!-- SDF -->
<link name="base_link">
    <pose>0 0 0 0 0 0</pose>
</link>

<link name="slider_link">
    <pose>some values here</pose>
</link>

<joint name="slider_joint" type="prismatic">
    <parent>base_link</parent>
    <child>slider_link</child>
</joint>
```

Installing Gazebo Fortress

Installing Gazebo Fortress

```
sudo apt update
sudo apt install lsb-release gnupg
```

Installing Gazebo Fortress

Step 2 – Add OSRF repo & key

```
sudo curl https://packages.osrfoundation.org/gazebo.gpg \
--output /usr/share/keyrings/pkgs-osrf-archive-keyring.gpg
```

```
echo "deb [arch=$(dpkg --print-architecture) signed-by=/usr/share/keyrings/pkgs-osrf-archive-keyring.gpg] \
http://packages.osrfoundation.org/gazebo/ubuntu-stable $(lsb_release -cs) main" | \
sudo tee /etc/apt/sources.list.d/gazebo-stable.list > /dev/null
```

Installing Gazebo Fortress

Step 3 – Update & Install

```
sudo apt update
sudo apt install ignition-fortress
```

Installing Gazebo Fortress

Step 4 – Run

```
ign gazebo
```

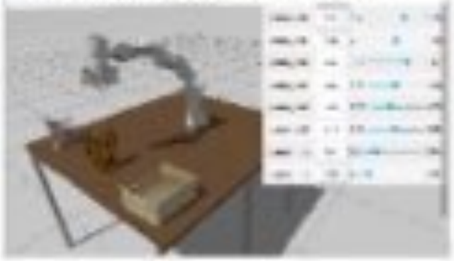
Gazebo quick start

GAZEBO

Fortress
v 6.17.0

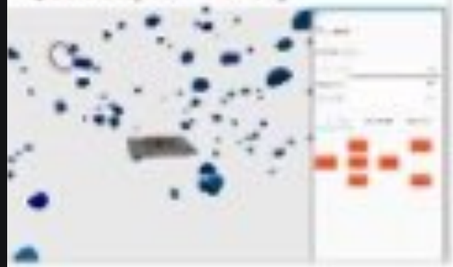
NAO Joint Control

Panda Joint Control World



Prius On Sonoma Raceway

Quadcopter Teleop



Tugbot In Warehouse

Empty

3k_shapes.sdf

ackermann_steering...

actor.sdf

actor_crowd.sdf

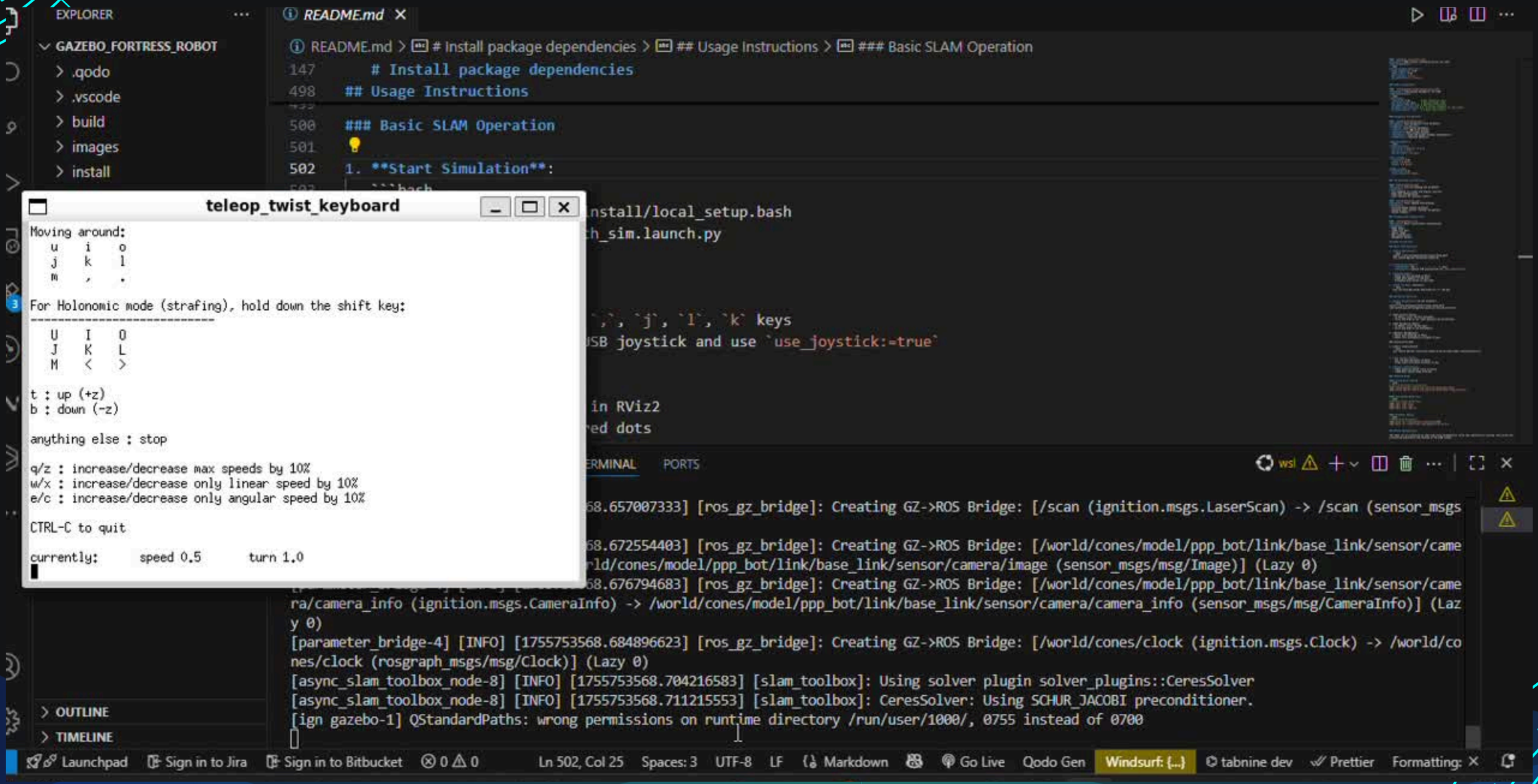
actors_population.sdf

apply_joint_force.sdf

Don't show this dialog again

RUN

Video of Simulation in Action



References

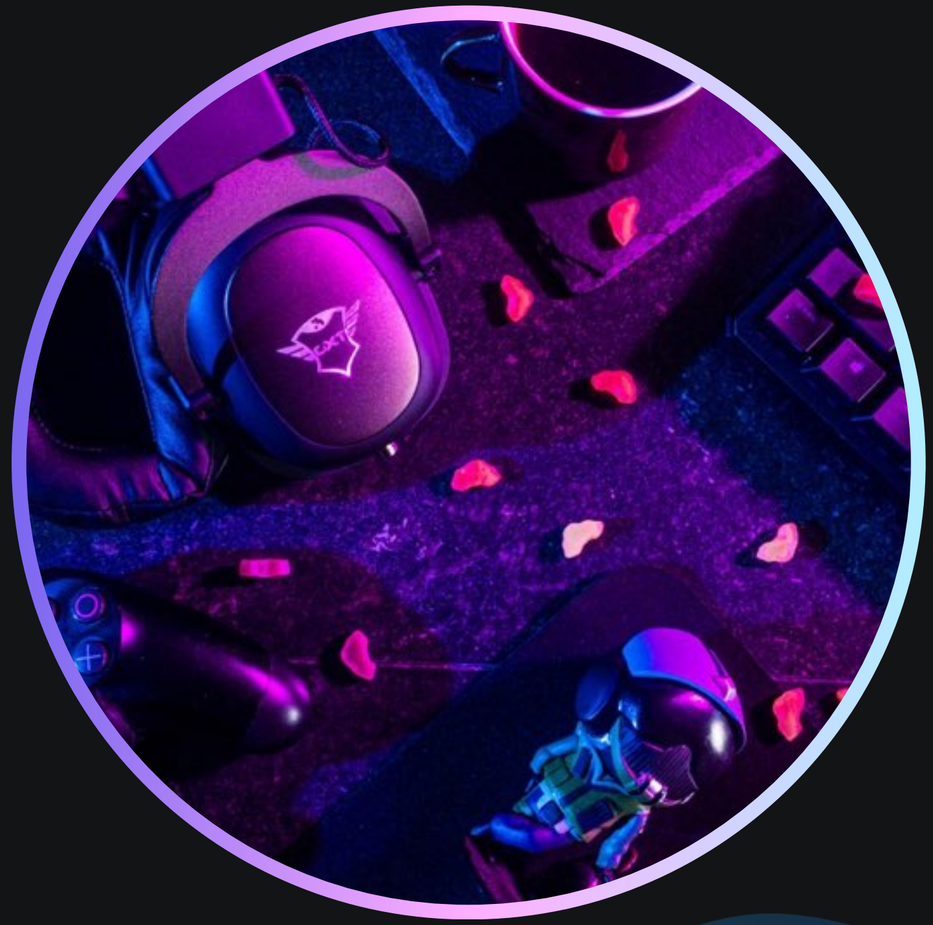
https://roboticsdojo.substack.com/p/getting-started-with-gazebo-fortress

https://roboticsdojo.substack.com/p/understanding-urdf-building-the-blueprint

https://roboticsdojo.substack.com/p/introduction-to-worlds-in-robotics

https://roboticsdojo.substack.com/p/introduction-to-simultaneous-localization

https://github.com/FionaMich/gazebo_ignition_fortress **(Kindly Review this Repo to Practice before next session on simulation next week Monday)**



Thank You