Gazebo and ROS 2

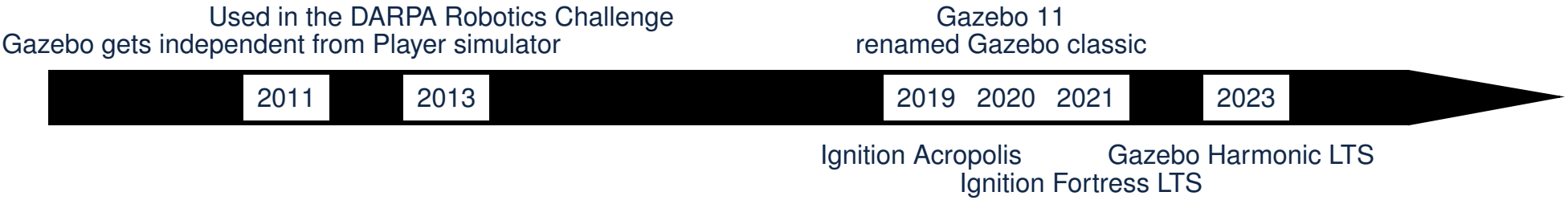
Olivier Kermorgant

ROSCon FR 2024

CENTRALE NANTES

LS2N

A short history of Gazebo



Features

- Dynamic simulator (ODE / Bullet)
- 3D rendering (OGRE)
- Supported formats: URDF and SDF
- Plugin-based architecture

Link with ROS 2

- Classic: plugins are ROS nodes
- New: ROS-Gz bridge
- Can also be used without ROS

Compatibility between ROS and Gazebo

From `https://gazebosim.org/docs/harmonic/ros_installation`

ROS	IGN Citadel (LTS)	IGN Fortress (LTS)	GZ Garden	GZ Harmonic (LTS)
ROS 2 Jazzy (LTS)	❌	❌	⚡	✅
ROS 2 Rolling	❌	✅	⚡	⚡
ROS 2 Iron	❌	✅	⚡	⚡
ROS 2 Humble (LTS)	❌	✅	⚡	⚡
ROS 2 Galactic	❌	❌	❌	❌
ROS 2 Foxy (LTS)	✅	❌	❌	❌

How to install ROS-GZ bridge?

- ✅ ROS packages: `ros-${ROS_DISTRO}-ros-gz`
- ⚡ OSRF packages: `ros-${ROS_DISTRO}-ros-gz$GZ_VERSION`
- ❌ from source (might of course fail)

Known to work for this tutorial

- Galactic + Fortress (Garden+ will uninstall GZ classic)
- Humble + Garden or Harmonic (recommended)
- Jazzy + Harmonic (just released)

Compatibility between ROS and Gazebo

From `https://gazebosim.org/docs/harmonic/ros_installation`

ROS	IGN Citadel (LTS)	IGN Fortress (LTS)	GZ Garden	GZ Harmonic (LTS)
ROS 2 Jazzy (LTS)	❌	❌	⚡	✅
ROS 2 Rolling	❌	✅	⚡	⚡
ROS 2 Iron	❌	✅	⚡	⚡
ROS 2 Humble (LTS)	❌	✅	⚡	⚡
ROS 2 Galactic	❌	❌	❌	❌
ROS 2 Foxy (LTS)	✅	❌	❌	❌

How to install ROS-GZ bridge?

- ✅ ROS packages: `ros-${ROS_DISTRO}-ros-gz`
- ⚡ OSRF packages: `ros-${ROS_DISTRO}-ros-gz$GZ_VERSION`
- ❌ from source (might of course fail)

Known to work for this tutorial

- Galactic + Fortress (Garden+ will uninstall GZ classic)
- Humble + Garden or Harmonic (recommended)
- Jazzy + Harmonic (just released)

Pure SDF

- world file includes models
- uses Gazebo environment variables
- difficult to link with ROS
- somehow quite static scenario

Separate world and models

- SDF for empty world file
- then spawn models from `robot_description`
- direct link with ROS and `/tf`
- spawn pose can be changed through launch files

1 Run Gazebo with a basic world (launch file / bash)

```
ros2 launch ros_gz_sim gz_sim.launch.py gz_args:=/path/to/world.sdf
```

2 Spawn models (from URDF) at desired poses (launch file)

```
# using simple_launch
ns = 'turret'
with sl.group(ns = ns):
    sl.robot_state_publisher('simple_launch', 'turret.xacro', xacro_args={'prefix': ns+'/'})
    sl.spawn_gz_model(ns)
```

3 Run ROS-GZ bridges (launch file)

```
bridges = [GazeboBridge.clock()]

# cmd_vel for joints
for joint in ('joint1', 'joint2', 'joint3'):
    bridges.append(GazeboBridge(f'{ns}/{joint}_cmd_vel', f'{joint}_cmd_vel',
                                'std_msgs/Float64', GazeboBridge.ros2gz))

# joint state feedback on /world/<world>/model/<model>/joint_state
gz_js_topic = GazeboBridge.model_prefix(ns) + '/joint_state'
bridges.append(GazeboBridge(gz_js_topic, 'joint_states',
                            'sensor_msgs/JointState', GazeboBridge.gz2ros))

# image
bridges.append(GazeboBridge(f'{ns}/image', 'image', 'sensor_msgs/Image', GazeboBridge.gz2ros))

sl.create_gz_bridge(bridges)
```

An empty world with sun and sensors

```
<?xml version="1.0" ?>
<sdf version="1.6">
  <world name="floatgen">

    <physics name="10ms" type="ode">
      <max_step_size>0.01</max_step_size>
      <real_time_factor>1.0</real_time_factor>
    </physics>
    <plugin filename="gz-sim-physics-system" name="gz::sim::systems::Physics"/>
    <plugin filename="gz-sim-user-commands-system" name="gz::sim::systems::UserCommands"/>
    <plugin filename="gz-sim-scene-broadcaster-system" name="gz::sim::systems::SceneBroadcaster"/>
    <plugin filename="gz-sim-imu-system" name="gz::sim::systems::Imu"/>
    <plugin filename="gz-sim-sensors-system" name="gz::sim::systems::Sensors">
      <render_engine>ogre2</render_engine>
    </plugin>

    <light type="directional" name="sun">
      <cast_shadows>true</cast_shadows>
      <pose>0 0 10 0 0 0</pose>
      <diffuse>1 1 1 1</diffuse>
      <specular>0.5 0.5 0.5 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>
  </world>
</sdf>
```

no ground plane... ok for maritime / aerial

An empty world with sun and sensors, and a ground plane

```
<?xml version="1.0" ?>
<sdf version="1.6">
  <world name="empty">

    <!-- same as previous world -->

      <model name="ground_plane">
      <static>true</static>
      <link name="link">
        <collision name="collision">
          <geometry>
            <plane>
              <normal>0 0 1</normal>
              <size>100 100</size>
            </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>
```

- model
  - links
    - visual → useful (mesh / basic shape)
    - inertia → mandatory (links without inertia are ignored)
    - collision → if relevant
  - joints
    - limit → required anyway
    - dynamics (damping) → if relevant
  - sensors
    - usually related to a particular link
  - plugins
    - related to full model
    - related to a particular joint (attached controller)

Plugins or sensors may open internal Gazebo topics

ROS-GZ bridge will then link Gazebo topics to ROS topics

Sensor examples

```
<gazebo reference="lidar_link">
  <sensor type="gpu_ray" name="lidar">
    <update_rate>5</update_rate>
    <topic>scan</topic>
    <visualize>true</visualize>
    <gz_frame_id>lidar_link</gz_frame_id>
    <ray>
      <scan>
        <horizontal>
          <samples>360</samples>
          <resolution>1</resolution>
          <min_angle>0.0</min_angle>
          <max_angle>6.28319</max_angle>
        </horizontal>
         <vertical>
            <samples>10</samples>
            <resolution>1</resolution>
            <min_angle>-0.1</min_angle>
            <max_angle>0.1</max_angle>
          </vertical>
      </scan>
      <range>
        <min>0.1</min>
        <max>20.</max>
        <resolution>0.015</resolution>
      </range>
    </ray>
  </sensor>
</gazebo>
```

```
# the corresponding bridge from Gazebo to ROS
ros2 run ros_gz_bridge parameter_bridge scan@sensor_msgs/msg/PointCloud2[gz.msgs.PointCloudPacked
```

Model plugin examples

```
<gazebo>
  <plugin filename="ignition-gazebo-joint-state-publisher-system"
          name="ignition::gazebo::systems::JointStatePublisher"/>

  <plugin filename="ignition-gazebo-odometry-publisher-system"
          name="ignition::gazebo::systems::OdometryPublisher">
      <odom_frame>odom</odom_frame>
      <dimensions>3</dimensions>
      <robot_base_frame>base_link</robot_base_frame>
      <odom_publish_frequency>10</odom_publish_frequency>
  </plugin>

  <plugin filename="ignition-gazebo-pose-publisher-system"
          name="ignition::gazebo::systems::PosePublisher">
  </plugin>
</gazebo>
```

```
# the corresponding bridges from Gazebo to ROS
ros2 run ros_gz_bridge parameter_bridge joint_state@sensor_msgs/msg/JointState[gz.msgs.Model

ros2 run ros_gz_bridge parameter_bridge odom@nav_msgs/msg/Odometry[gz.msgs.Odometry

ros2 run ros_gz_bridge parameter_bridge pose@geometry_msgs/msg/Pose[gz.msgs.Pose
```

Controller plugin examples

```
<gazebo>
  <plugin filename="ignition-gazebo-joint-position-controller-system"
          name="ignition::gazebo::systems::JointPositionController">
        <joint_name>tilt</joint_name>
        <p_gain>10.</p_gain>
        <i_gain>0.2</i_gain>
        <d_gain>0.1</d_gain>
        <i_max>10</i_max>
        <i_min>-10</i_min>
  </plugin>
</gazebo>

<gazebo>
  <plugin
        filename="ignition-gazebo-thruster-system"
        name="ignition::gazebo::systems::Thruster">
        <namespace>${ns}</namespace>
        <joint_name>thruster${thruster_id}</joint_name>
        <topic>thruster${thruster_id}/cmd</topic>
        <thrust_coefficient>0.011</thrust_coefficient>
        <fluid_density>${density}</fluid_density>
        <propeller_diameter>${2*prop_r}</propeller_diameter>
  </plugin>
</gazebo>
```

```
# the corresponding bridge from ROS to Gazebo (first plugin for tilt command)
ros2 run ros_gz_bridge parameter_bridge tilt/0/cmd_pos@std_msgs/msg/Float64]gz.msgs.Double
```

Command-line syntax is a bit strange

```
ros2 run ros_gz_bridge parameter_bridge <gz topic>@<ros msg>]<gz msg>
```

- ROS topic is the same as Gazebo: use remapping if needed
- Only some combinations of ROS and GZ messages make sense
-] for GZ to ROS, [for the opposite, @ for both

Configuration file

```
- ros_topic_name: "ros_chatter"
  gz_topic_name: "gz_chatter"
  ros_type_name: "std_msgs/msg/String"
  gz_type_name: "gz.msgs.StringMsg"
  direction: GZ_TO_ROS # or ROS_TO_GZ or BIDIRECTIONAL
```

- Only some combinations of ROS and GZ messages make sense

Wrapped into `simple_launch`

```
sl.create_gz_bridge(GazeboBridge('gz_chatter', 'ros_chatter', 'std_msgs/String', GazeboBridge.gz2ros))
```

- Generates ad-hoc configuration file at launch

Command-line syntax is a bit strange

```
ros2 run ros_gz_bridge parameter_bridge <gz topic>@<ros msg>]<gz msg>
```

- ROS topic is the same as Gazebo: use remapping if needed
- Only some combinations of ROS and GZ messages make sense
-] for GZ to ROS, [for the opposite, @ for both

Configuration file

```
- ros_topic_name: "ros_chatter"
  gz_topic_name: "gz_chatter"
  ros_type_name: "std_msgs/msg/String"
  gz_type_name: "gz.msgs.StringMsg"
  direction: GZ_TO_ROS # or ROS_TO_GZ or BIDIRECTIONAL
```

- Only some combinations of ROS and GZ messages make sense

Wrapped into `simple_launch`

```
sl.create_gz_bridge(GazeboBridge('gz_chatter', 'ros_chatter', 'std_msgs/String', GazeboBridge.gz2ros))
```

- Generates ad-hoc configuration file at launch

Command-line syntax is a bit strange

```
ros2 run ros_gz_bridge parameter_bridge <gz topic>@<ros msg>]<gz msg>
```

- ROS topic is the same as Gazebo: use remapping if needed
- Only some combinations of ROS and GZ messages make sense
-] for GZ to ROS, [for the opposite, @ for both

Configuration file

```
- ros_topic_name: "ros_chatter"
  gz_topic_name: "gz_chatter"
  ros_type_name: "std_msgs/msg/String"
  gz_type_name: "gz.msgs.StringMsg"
  direction: GZ_TO_ROS # or ROS_TO_GZ or BIDIRECTIONAL
```

- Only some combinations of ROS and GZ messages make sense

Wrapped into `simple_launch`

```
sl.create_gz_bridge(GazeboBridge('gz_chatter', 'ros_chatter', 'std_msgs/String', GazeboBridge.gz2ros))
```

- Generates ad-hoc configuration file at launch

Command-line syntax is a bit strange

```
ros2 run ros_gz_bridge parameter_bridge <gz topic>@<ros msg>]<gz msg>
```

- ROS topic is the same as Gazebo: use remapping if needed
- Only some combinations of ROS and GZ messages make sense
-] for GZ to ROS, [for the opposite, @ for both

Configuration file

```
- ros_topic_name: "ros_chatter"
  gz_topic_name: "gz_chatter"
  ros_type_name: "std_msgs/msg/String"
  gz_type_name: "gz.msgs.StringMsg"
  direction: GZ_TO_ROS # or ROS_TO_GZ or BIDIRECTIONAL
```

- Only some combinations of ROS and GZ messages make sense

Wrapped into `simple_launch`

```
sl.create_gz_bridge(GazeboBridge('gz_chatter', 'ros_chatter', 'std_msgs/String', GazeboBridge.gz2ros))
```

- Generates ad-hoc configuration file at launch

Command-line syntax is a bit strange

```
ros2 run ros_gz_bridge parameter_bridge <gz topic>@<ros msg>]<gz msg>
```

- ROS topic is the same as Gazebo: use remapping if needed
- Only some combinations of ROS and GZ messages make sense
-] for GZ to ROS, [for the opposite, @ for both

Configuration file

```
- ros_topic_name: "ros_chatter"
  gz_topic_name: "gz_chatter"
  ros_type_name: "std_msgs/msg/String"
  gz_type_name: "gz.msgs.StringMsg"
  direction: GZ_TO_ROS # or ROS_TO_GZ or BIDIRECTIONAL
```

- Only some combinations of ROS and GZ messages make sense

Wrapped into `simple_launch`

```
sl.create_gz_bridge(GazeboBridge('gz_chatter', 'ros_chatter', 'std_msgs/String', GazeboBridge.gz2ros))
```

- Generates ad-hoc configuration file at launch

Command-line syntax is a bit strange

```
ros2 run ros_gz_bridge parameter_bridge <gz topic>@<ros msg>]<gz msg>
```

- ROS topic is the same as Gazebo: use remapping if needed
- Only some combinations of ROS and GZ messages make sense
-] for GZ to ROS, [for the opposite, @ for both

Configuration file

```
- ros_topic_name: "ros_chatter"
  gz_topic_name: "gz_chatter"
  ros_type_name: "std_msgs/msg/String"
  gz_type_name: "gz.msgs.StringMsg"
  direction: GZ_TO_ROS # or ROS_TO_GZ or BIDIRECTIONAL
```

- Only some combinations of ROS and GZ messages make sense

Wrapped into `simple_launch`

```
sl.create_gz_bridge(GazeboBridge('gz_chatter', 'ros_chatter', 'std_msgs/String', GazeboBridge.gz2ros))
```

- Generates ad-hoc configuration file at launch

Command-line syntax is a bit strange

```
ros2 run ros_gz_bridge parameter_bridge <gz topic>@<ros msg>]<gz msg>
```

- ROS topic is the same as Gazebo: use remapping if needed
- Only some combinations of ROS and GZ messages make sense
-] for GZ to ROS, [for the opposite, @ for both

Configuration file

```
- ros_topic_name: "ros_chatter"
  gz_topic_name: "gz_chatter"
  ros_type_name: "std_msgs/msg/String"
  gz_type_name: "gz.msgs.StringMsg"
  direction: GZ_TO_ROS # or ROS_TO_GZ or BIDIRECTIONAL
```

- Only some combinations of ROS and GZ messages make sense

Wrapped into `simple_launch`

```
sl.create_gz_bridge(GazeboBridge('gz_chatter', 'ros_chatter', 'std_msgs/String', GazeboBridge.gz2ros))
```

- Generates ad-hoc configuration file at launch

Command-line syntax is a bit strange

```
ros2 run ros_gz_bridge parameter_bridge <gz topic>@<ros msg>]<gz msg>
```

- ROS topic is the same as Gazebo: use remapping if needed
- Only some combinations of ROS and GZ messages make sense
-] for GZ to ROS, [for the opposite, @ for both

Configuration file

```
- ros_topic_name: "ros_chatter"
  gz_topic_name: "gz_chatter"
  ros_type_name: "std_msgs/msg/String"
  gz_type_name: "gz.msgs.StringMsg"
  direction: GZ_TO_ROS # or ROS_TO_GZ or BIDIRECTIONAL
```

- Only some combinations of ROS and GZ messages make sense

Wrapped into `simple_launch`

```
sl.create_gz_bridge(GazeboBridge('gz_chatter', 'ros_chatter', 'std_msgs/String', GazeboBridge.gz2ros))
```

- Generates ad-hoc configuration file at launch

Files (e.g. meshes) as `package://ros_package/meshes/some.dae`

- `ros_package` is a ROS package, will be resolved by RViz
- Gazebo does **not** know about ROS packages

Gazebo looks for files in `GZ_SIM_RESOURCE_PATH`

- Meshes
- Libraries (plugins)

Easy to handle with `ament_environment_hooks`

```
# installed resources are in share
prepend-non-duplicate;GZ_SIM_RESOURCE_PATH;share
# installed libraries are in lib/project_name
prepend-non-duplicate;GZ_SIM_RESOURCE_PATH;lib/@PROJECT_NAME@
```

Files (e.g. meshes) as `package://ros_package/meshes/some.dae`

- `ros_package` is a ROS package, will be resolved by RViz
- Gazebo does **not** know about ROS packages

Gazebo looks for files in `GZ_SIM_RESOURCE_PATH`

- Meshes
- Libraries (plugins)

Easy to handle with `ament_environment_hooks`

```
# installed resources are in share
prepend-non-duplicate;GZ_SIM_RESOURCE_PATH;share
# installed libraries are in lib/project_name
prepend-non-duplicate;GZ_SIM_RESOURCE_PATH;lib/@PROJECT_NAME@
```

Files (e.g. meshes) as `package://ros_package/meshes/some.dae`

- `ros_package` is a ROS package, will be resolved by RViz
- Gazebo does **not** know about ROS packages

Gazebo looks for files in `GZ_SIM_RESOURCE_PATH`

- Meshes
- Libraries (plugins)

Easy to handle with `ament_environment_hooks`

```
# installed resources are in share
prepend-non-duplicate;GZ_SIM_RESOURCE_PATH;share
# installed libraries are in lib/project_name
prepend-non-duplicate;GZ_SIM_RESOURCE_PATH;lib/@PROJECT_NAME@
```

Files (e.g. meshes) as `package://ros_package/meshes/some.dae`

- `ros_package` is a ROS package, will be resolved by RViz
- Gazebo does **not** know about ROS packages

Gazebo looks for files in `GZ_SIM_RESOURCE_PATH`

- Meshes
- Libraries (plugins)

Easy to handle with `ament_environment_hooks`

```
# installed resources are in share
prepend-non-duplicate;GZ_SIM_RESOURCE_PATH;share
# installed libraries are in lib/project_name
prepend-non-duplicate;GZ_SIM_RESOURCE_PATH;lib/@PROJECT_NAME@
```

Files (e.g. meshes) as `package://ros_package/meshes/some.dae`

- `ros_package` is a ROS package, will be resolved by RViz
- Gazebo does **not** know about ROS packages

Gazebo looks for files in `GZ_SIM_RESOURCE_PATH`

- Meshes
- Libraries (plugins)

Easy to handle with `ament_environment_hooks`

```
# installed resources are in share
prepend-non-duplicate;GZ_SIM_RESOURCE_PATH;share
# installed libraries are in lib/project_name
prepend-non-duplicate;GZ_SIM_RESOURCE_PATH;lib/@PROJECT_NAME@
```

Files (e.g. meshes) as `package://ros_package/meshes/some.dae`

- `ros_package` is a ROS package, will be resolved by RViz
- Gazebo does **not** know about ROS packages

Gazebo looks for files in `GZ_SIM_RESOURCE_PATH`

- Meshes
- Libraries (plugins)

Easy to handle with `ament_environment_hooks`

```
# installed resources are in share
prepend-non-duplicate;GZ_SIM_RESOURCE_PATH;share
# installed libraries are in lib/project_name
prepend-non-duplicate;GZ_SIM_RESOURCE_PATH;lib/@PROJECT_NAME@
```

Files (e.g. meshes) as `package://ros_package/meshes/some.dae`

- `ros_package` is a ROS package, will be resolved by RViz
- Gazebo does **not** know about ROS packages

Gazebo looks for files in `GZ_SIM_RESOURCE_PATH`

- Meshes
- Libraries (plugins)

Easy to handle with `ament_environment_hooks`

```
# installed resources are in share
prepend-non-duplicate;GZ_SIM_RESOURCE_PATH;share
# installed libraries are in lib/project_name
prepend-non-duplicate;GZ_SIM_RESOURCE_PATH;lib/@PROJECT_NAME@
```