



**An international university campus across 1300 acres of land**



**Professor Yihan Xing**  
Study program leader, MSc Marine and Offshore

# ROS2/Gazebo - Marine Simulator v1

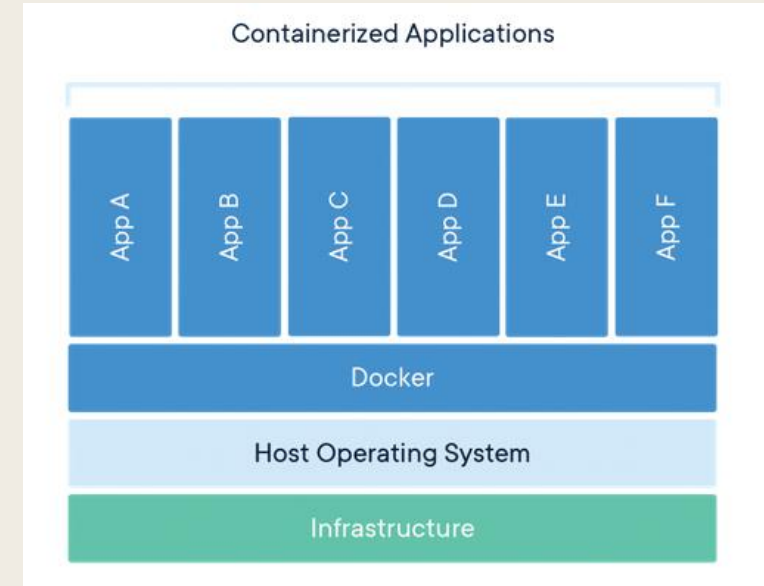
# Agenda

---

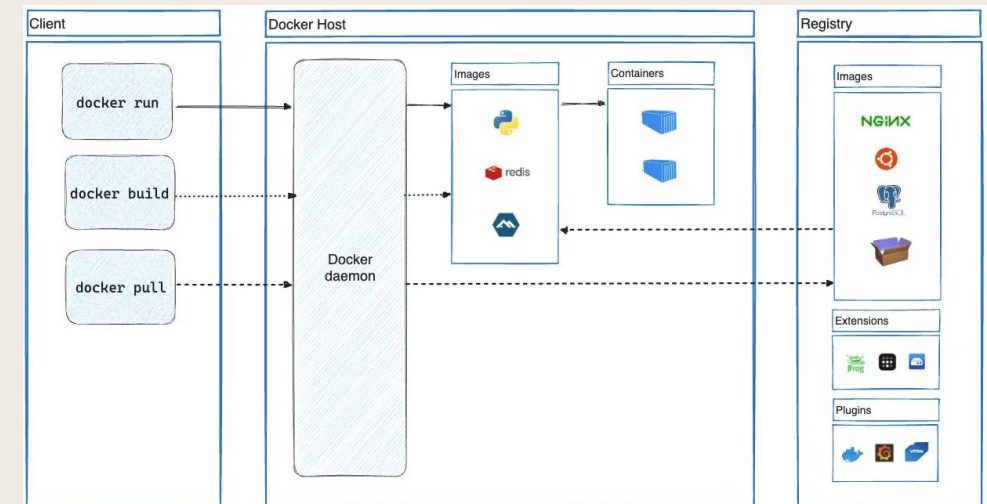
- Docker Engine
- ROS2
- Gazebo
- BlueROV2 ROS2/Gazebo model

# Docker Engine

- Open-source containerization tool
- Docker container:
  - Run-time environment with the applications to be run and their dependencies
- Docker image:
  - Standalone and executable file that is used to create the container
  - Contains all files required (application, libraries, dependencies, etc.)
  - Sharable and portable



Source: <https://www.docker.com/resources/what-container/>



Source: <https://docs.docker.com/guides/docker-overview/#docker-architecture>

# Docker workflow process

---

- Step 1: Build the Docker image (Dockerfile)
  - Use a Dockerfile for easier definition of the image to be built
- Step 2: Load the Docker image as a container (load\_dock.sh)
  - This is the main terminal of the container. Exiting from this terminal will unload the container.
  - No files will be saved in the container upon unloading.
  - Load and use a share volume from the host machine for saving work files.
- Step 3: Access the running Docker container if more additional terminals are required (access\_cont.sh)
- Step 4: Exit from the container
  - Exiting from the main terminal will unload the container

# DockerFile

---

## ○ Base image

- Example: nvidia/cuda:12.6.1-base-ubuntu24.04
- <https://hub.docker.com/r/nvidia/cuda>
- Reason: I am using a laptop that has NVIDIA RTX 2000 Ada. The Cuda base image has already many of the required graphics drivers pre-installed and configured.

## ○ Install packages

- Sudo not required; the build process runs in root
- Install ROS2 jazzy, Gazebo harmonic, Gazebo/ROS pairing, misc ROS2 tools and misc tools

```
Dockerfile > ...
1 # ----- Define base image -----
2 FROM nvidia/cuda:12.6.1-base-ubuntu24.04
3
4 # ----- Define environmental variables -----
5 ENV HOME=/home
6 ENV TZ=Europe/Oslo
7 RUN ln -snf /usr/share/zoneinfo/$TZ /etc/localtime && echo $TZ > /etc/timezone
8
9 # ----- Install ROS2 Jazzy -----
10 RUN apt update && apt install -y locales
11 RUN locale-gen en_US en_US.UTF-8
12 RUN update-locale LC_ALL=en_US.UTF-8 LANG=en_US.UTF-8
13 ENV LANG=en_US.UTF-8
14 RUN apt install -y software-properties-common
15 RUN add-apt-repository universe
16 RUN apt update && apt install curl -y
17 RUN curl -sSL https://raw.githubusercontent.com/ros/rosdistro/master/ros.key -o /usr/share/keyrings/ros-archive-keyring.gpg
18 RUN echo "deb [arch=$(dpkg --print-architecture) signed-by=/usr/share/keyrings/ros-archive-keyring.gpg] http://packages.ros.org/ros-archive/ubuntu jammy main" > /etc/apt/sources.list.d/ros-list.txt
19 RUN apt update && apt install -y ros-dev-tools
20 RUN apt update -y && apt upgrade -y
21 RUN apt install -y ros-jazzy-desktop
22 RUN echo "source /opt/ros/jazzy/setup.bash" >> ~/.bashrc
23
24 # ----- Install Gazebo Harmonic -----
25 RUN apt-get update
26 RUN apt-get install -y lsb-release gnupg
27 RUN curl https://packages.osrfoundation.org/gazebo.gpg --output /usr/share/keyrings/gazebo-archive-keyring.gpg
28 RUN echo "deb [arch=$(dpkg --print-architecture) signed-by=/usr/share/keyrings/gazebo-archive-keyring.gpg] http://packages.osrfoundation.org/gazebo/ubuntu-stable $(lsb_release -cs) main" > /etc/apt/sources.list.d/gazebo-list.txt
29 RUN apt-get update
30 RUN apt-get install -y gz-harmonic
31
32 # ----- Install Gazebo/ROS pairing -----
33 RUN apt-get install -y ros-jazzy-ros-gz
```



# Load and run Docker container

## ○ load\_dock.sh

- **xhost +local:docker**: Allows the Docker container to access the host's X server for GUI applications.
- Container options: **-it**: Interactive mode with a pseudo-TTY; **--rm**: Automatically remove the container when it exits; **--privileged**: Gives extended privileges to this container; **--gpus all**: Allows the container to use all available NVIDIA GPUs; **--name marsim\_sdf\_cont**: Names the container "marsim\_sdf\_cont".
- Environment variables: **XDG\_RUNTIME\_DIR=/tmp/runtime-root**: Sets the runtime directory; **DISPLAY=\$DISPLAY**: Passes the host's DISPLAY environment to the container; **QT\_X11\_NO\_MITSHM=1**: Disables MIT-SHM extension for Qt applications; **MESA\_GL\_VERSION\_OVERRIDE=3.3**: Forces OpenGL version 3.3 for Mesa drivers.
- Volume mounts: **/tmp/.X11-unix:/tmp/.X11-unix**: Mounts X11 socket for GUI support; **/home/yhxing/Docker\_WS/marsim1:/home/marsim\_ws:rw**: Mounts a local directory into the container with read-write access;
- **--runtime=nvidia**: Specifies to use the NVIDIA Container Runtime.

## ○ access\_cont.sh

load\_dock.sh

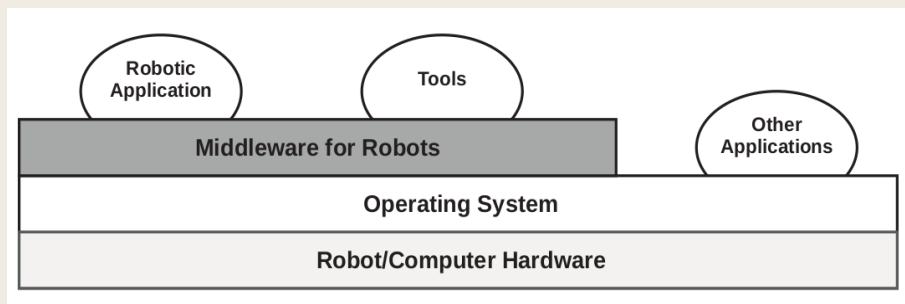
```
$ load_dock.sh
1  #!/usr/bin/bash
2
3  xhost +local:docker
4
5  docker run \
6  -it --rm \
7  --privileged \
8  --gpus all \
9  --name marsim_cont \
10 --env="XDG_RUNTIME_DIR=/tmp/runtime-root" \
11 --env="DISPLAY=$DISPLAY" \
12 --env="QT_X11_NO_MITSHM=1" \
13 --env="MESA_GL_VERSION_OVERRIDE=3.3" \
14 --volume="/tmp/.X11-unix:/tmp/.X11-unix" \
15 --volume="/home/yhxing/Docker_WS/marsim1:/home/marsim_ws:rw" \
16 --runtime=nvidia \
17 marsim1_dock
```

access\_cont.sh











```
$ access_dock.sh
1  #!/usr/bin/bash
2
3  docker exec -it marsim_sdf_cont /bin/bash
```

# Robot Operating System 2 (ROS2)

- Material is taken from Rico, F. M. (2022). A concise introduction to robot programming with ROS2. Chapman and Hall/CRC.
- ROS2 is not an operating system; it is a middleware developed and maintained by the Open Robotics Foundation
- Middleware is a layer of software between the operating system and the user applications
- ROS2 is the 2nd generation of ROS

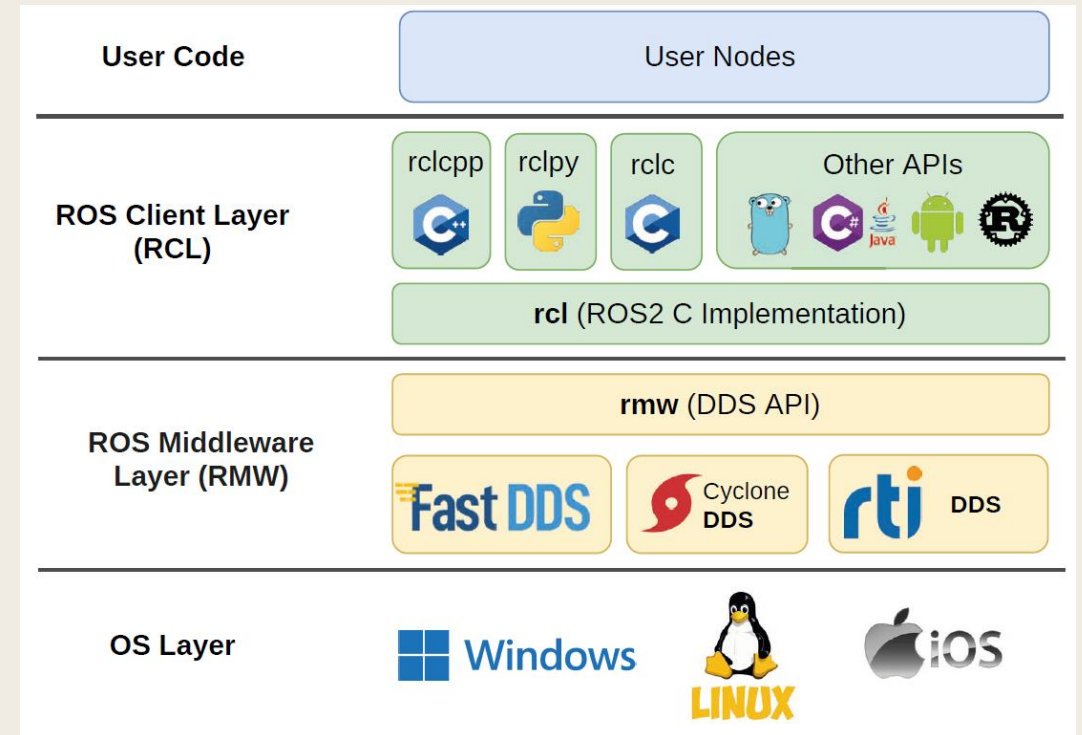


Source: Rico, F. M. (2022). A concise introduction to robot programming with ROS2. Chapman and Hall/CRC

Distro	Release date	Logo	EOL date
Jazzy Jalisco	May 23rd, 2024		May 2029
Iron Irtwin	May 23rd, 2023		November 2024
Humble Hawkbill	May 23rd, 2022		May 2027
Galactic Geochelone	May 23rd, 2021		December 9th, 2022
Foxy Fitzroy	June 5th, 2020		June 20th, 2023
Eloquent Elusor	November 22nd, 2019		November 2020
Dashing Diademata	May 31st, 2019		May 2021
Crystal Clemmys	December 14th, 2018		December 2019
Bouncy Bolson	July 2nd, 2018		July 2019
Ardent Apalone	December 8th, 2017		December 2018

# ROS2 architecture

- User nodes interface with ROS2 via RCL's APIs
- rcl is the core of ROS2 and is written in C
- rclcpp, rclpy, rcl, etc binds different programming languages to rcl offering the same behaviour, i.e., they are not independent implementations
- ROS2 is a distributed system using data distribution system (DDS) as its communications layer
- There are several DDS vendors; ROS2 Humble uses Cyclone DDS



Source: Rico, F. M. (2022). A concise introduction to robot programming with ROS2. Chapman and Hall/CRC



# ROS2 vs ROS1

## ○ Architecture

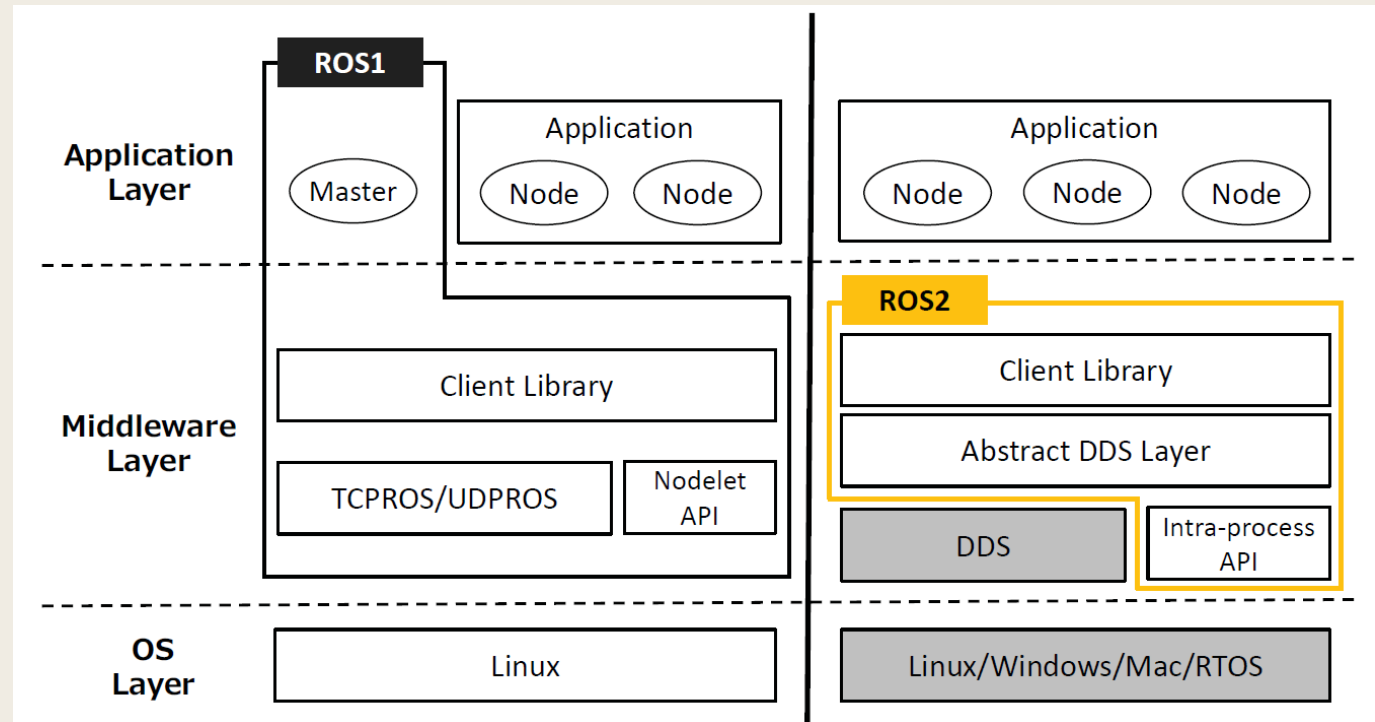
- DDS (ROS2) vs Master-Slave Architecture and the XML-RPC middleware (ROS1)
- rcp with binders to any programming language (ROS2) vs two separate libraries roscpp and rospy (ROS1)

## ○ Features

- QoS, multi-threaded execution and real-time processing

## ○ Tooling/Ecosystem

- Not backward compatible, but interface with ROS1 can be done via ROSbridge



Maruyama, Y., Kato, S., & Azumi, T. (2016, October). Exploring the performance of ROS2. In Proceedings of the 13th international conference on embedded software (pp. 1-10).

# Components of ROS2

---

- ROS2 is open source
- The community
  - Large community of users and developers contributing software packages
- Computation graph
  - ROS2 runs as a computational graph consisting of nodes and arcs
- Workspace
  - This is the user workspace that consists of the set of software installed on the robot or computer and the programs that the user develops

# The computational graph

- Node: single purpose executable program
- Topic: a named bus over which nodes exchange messages
- Publication/subscription: nodes sending/receiving messages on a topic
- Service: a synchronous request/response interaction between nodes: must be completed before the next one begins
- Action: an asynchronous request/response interaction with feedback; can be performed concurrently without waiting for each other to complete



Figure 1.3: Description of symbols used in computer graph diagrams.

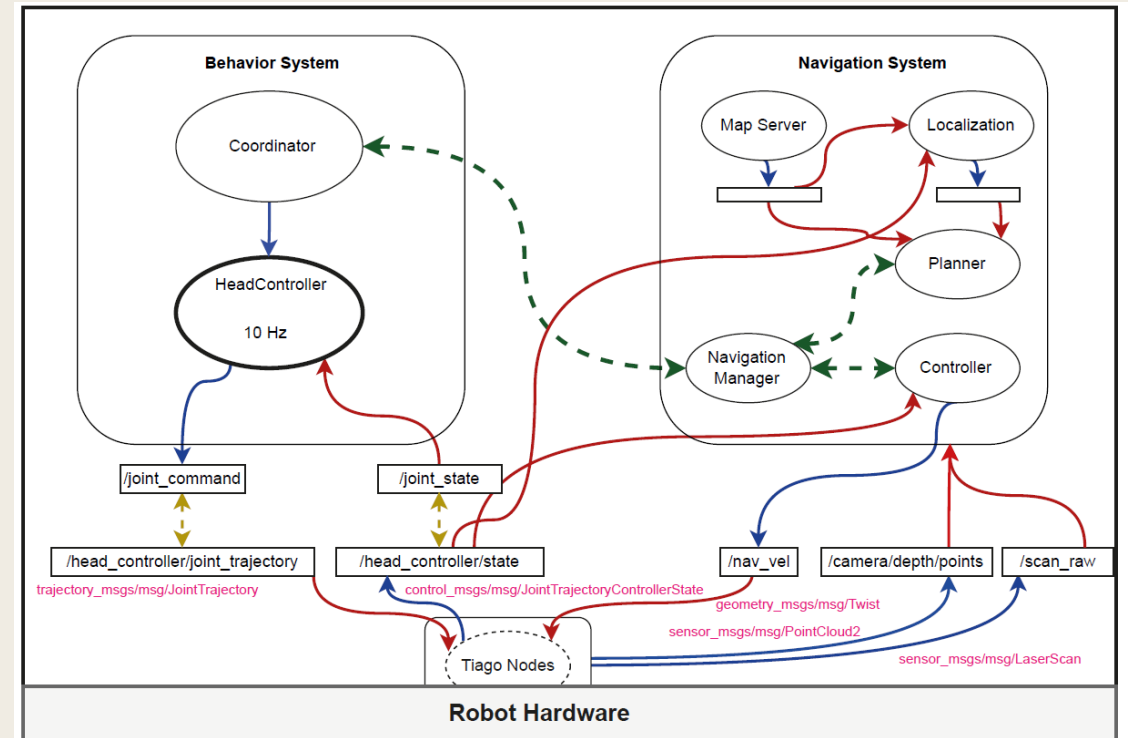


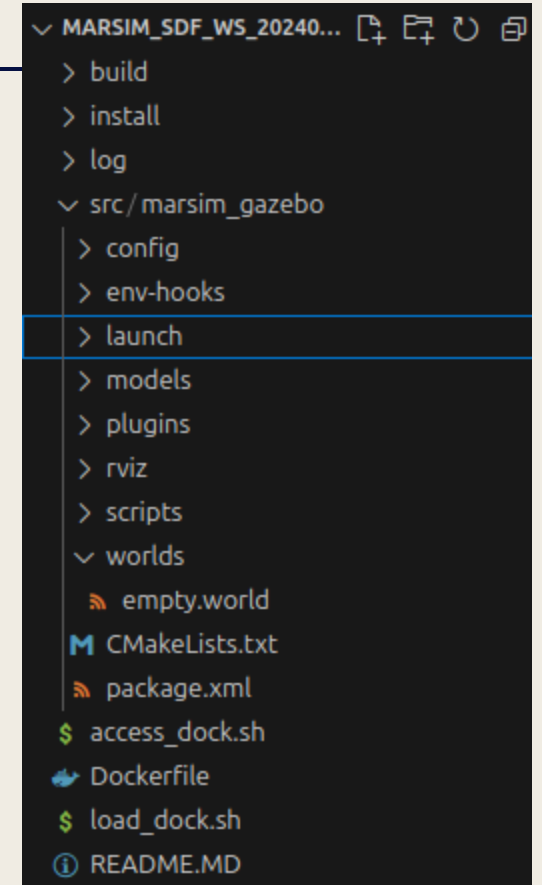
Figure 1.6: Computing graph of behavior-based application for the Tiago robot that uses a navigation subsystem.

Source: Rico, F. M. (2022). A concise introduction to robot programming with ROS2. Chapman and Hall/CRC

# ROS2 workspace

---

- Workspace directory
- Source code of user programs (packages) are stored in **src**
  - The workspace can contain multiple packages
  - Each package is stored in 1 directory with **CMakeList.txt** and **package.xml**
- Workspace needs to be built using **colcon build**
  - **build**, **install** and **log** directories are created
- To run the packages, the workspace needs to be sourced after building
  - **source install/setup.bash**



# CMakeList.txt and package.xml

---

- Must be placed at the root of the package directory
- Defines the build tool, folders to be included, dependencies, etc

```
cmake_minimum_required(VERSION 3.5)
project(gritbot_description)

find_package(ament_cmake REQUIRED)
find_package(urdf REQUIRED)

install(
  DIRECTORY launch urdf rviz
  DESTINATION share/${PROJECT_NAME}
)

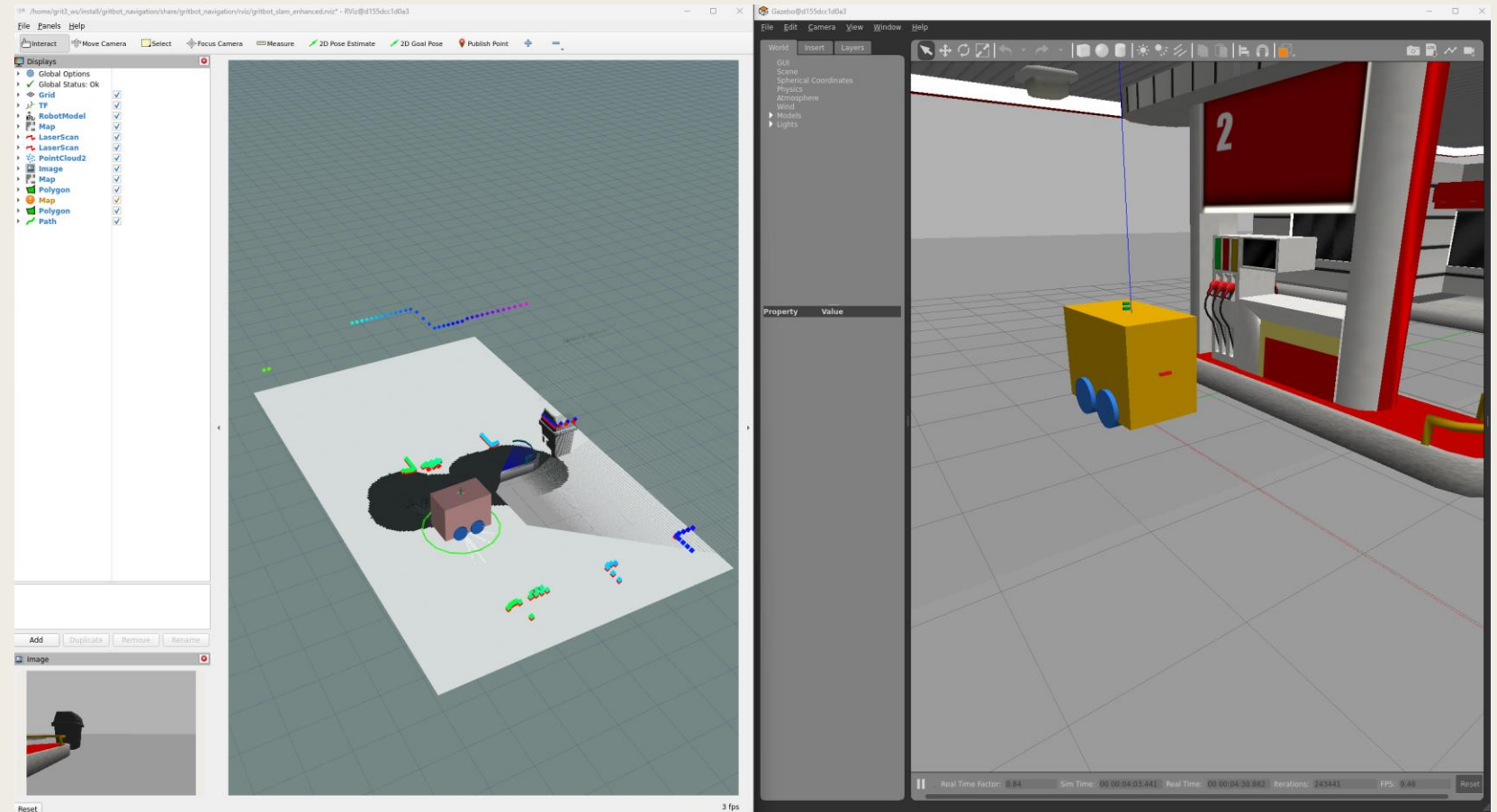
ament_package()
```

```
<?xml version="1.0"?>
<?xml-model href="http://download.ros.org/schema/package_format3.xsd" schematypens="http://www.w3.org/2001/XMLSchema"?>
<package format="3">
  <name>gritbot_description</name>
  <version>0.0.0</version>
  <description>Gritbot Description Package</description>
  <maintainer email="yihan.xing@uis.no">Yihan Xing</maintainer>
  <license>Apache 2.0</license>
  <url type="website">TBA</url>
  <url type="repository">TBA</url>
  <url type="bugtracker">TBA</url>
  <buildtool_depend>ament_cmake</buildtool_depend>
  <exec_depend>joint_state_publisher</exec_depend>
  <exec_depend>robot_state_publisher</exec_depend>
  <exec_depend>rviz2</exec_depend>
  <exec_depend>xacro</exec_depend>
  <export>
    <build_type>ament_cmake</build_type>
  </export>
</package>
```



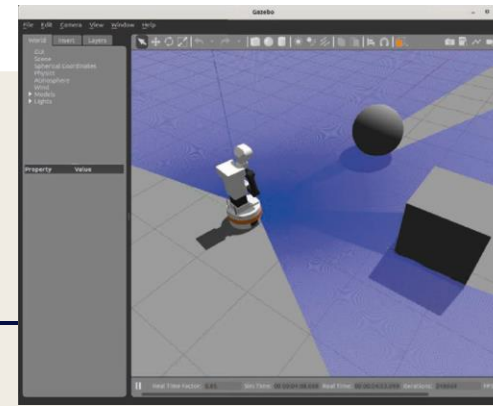
# Rviz2

- Port of Rviz to ROS2
- Installed by ROS2 by default
- Provides graphic interface for users to view the robot, sensor data, maps, etc

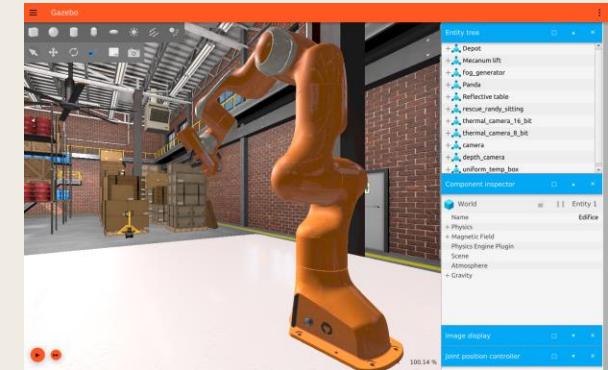


# Gazebo

- Open-source 3D robotics simulator developed and maintained by the Open Robotics Foundation
- Commonly used together with ROS and ROS2
- Features:
  - Physics engines: e.g., ODE, Bullet, Simbody and DART
  - 3D graphics: uses Object-Oriented Graphics Rendering Engine
  - Sensors and actuators: e.g., lidar, IMU, cameras, GPS, etc
  - Plugins: easily extendable for custom functionalities
  - World and environment: allows for complex and dynamic environments
  - Multi-robot simulation
- Gazebo classic and Gazebo (previously Ignition Gazebo)
- We are using **Gazebo Harmonic**



Gazebo Classic. Source: Rico, F. M. (2022). A concise introduction to robot programming with ROS2. Chapman and Hall/CRC



Gazebo. Source: <https://github.com/gazebo-sim>

Feature	Gazebo Classic	Gazebo (Ignition Gazebo)
Maturity	Established, widely adopted	Newer, growing ecosystem
Architecture	Monolithic	Modular
Performance	Good, but less optimized	Improved performance and scalability
Integration	Strong ROS 1 and ROS 2 support	Strong ROS 2 support
Plugins	Extensive, but less modular	Improved, modular plugins
Physics Engines	Multiple engines supported	Enhanced physics capabilities
Rendering	OGRE-based	Advanced rendering with modern APIs
Cross-Platform Support	Primarily Linux, limited Windows/macOS	Better cross-platform support
Development	Stable, less active development	Actively developed and growing

# ros\_gz\_bridge

- A network bridge which enables the exchange of messages between ROS 2 and Gazebo Transport
- We need a bridge because the robot is simulated in Gazebo, while the robot is controlled in ROS2; the messages of the robot from Gazebo must be ported to ROS2
- Limited to only certain message types
- There are several ways to configure the bridge; we will use a yaml file, which is the easiest way

```
src > marsim_gazebo > config > ! bluerov2_ros_gz_bridge.yaml
1  - ros_topic_name: "clock"
2    gz_topic_name: "clock"
3    ros_type_name: "rosgraph_msgs/msg/Clock"
4    gz_type_name: "gz.msgs.Clock"
5    subscriber_queue: 10      # Default 10
6    publisher_queue: 10      # Default 10
7    lazy: true                # Default "false"
8    direction: GZ_TO_ROS     # Default "BIDIRECTIONAL" - Bridge both directions
9                               # "GZ_TO_ROS" - Bridge Gz topic to ROS
10                              # "ROS_TO_GZ" - Bridge ROS topic to Gz
11
12 - ros_topic_name: "/bluerov2/ocean_current"
13   gz_topic_name: "/model/bluerov2/ocean_current"
14   ros_type_name: "geometry_msgs/msg/Vector3"
15   gz_type_name: "gz.msgs.Vector3d"
16   subscriber_queue: 10      # Default 10
17   publisher_queue: 10      # Default 10
18   lazy: true                # Default "false"
19   direction: GZ_TO_ROS     # Default "BIDIRECTIONAL" - Bridge both directions
20                              # "GZ_TO_ROS" - Bridge Gz topic to ROS
21                              # "ROS_TO_GZ" - Bridge ROS topic to Gz
22
23 - ros_topic_name: "/bluerov2/joint_states"
24   gz_topic_name: "/world/ocean/model/bluerov2/joint_state"
25   ros_type_name: "sensor_msgs/msg/JointState"
26   gz_type_name: "gz.msgs.Model"
27   subscriber_queue: 10      # Default 10
28   publisher_queue: 10      # Default 10
29   lazy: true                # Default "false"
30   direction: GZ_TO_ROS     # Default "BIDIRECTIONAL" - Bridge both directions
31                              # "GZ_TO_ROS" - Bridge Gz topic to ROS
32                              # "ROS_TO_GZ" - Bridge ROS topic to Gz
33
```

# BlueROV2 ROS2/Gazebo model

---

- World
  - Standard Gazebo physics, user commands, scene broadcaster, imu, sensors, dvl system and buoyancy plugins
  - Seabed modelling using a heightmap model
  - Custom-written Gauss Markov ocean current plugin
- ROV main body with 6 thrusters
  - Buoyancy modelled using a rectangular collision geometry
  - Standard Gazebo hydrodynamic, joint state publisher, pose publisher and odometry publisher plugins
  - Thruster attached to main body using revolute joints with standard Gazebo thruster plugins
  - Standard Gazebo IMU and DVL sensors with a custom-written ccp plugins and python scripts for post processing of measurements
  - Simulated depth sensor using standard Gazebo odometry publisher with added noise using a custom-written python script
  - ROS2 robot description and robot localization packages
  - Custom-written python script for robot keyboard controller

# Custom-written plugins and scripts

---

- Source codes are stored in /plugins and /scripts
- marsim\_gazebo::GaussMarkovOceanCurrent
- marsim\_gazebo::DVLRosConverter
- depth\_sensor\_from\_odom.py
- noisy\_odometry.py
- ros\_keyboard\_controller.py
- twist\_stamper.py



# marsim1 ROS2/Gazebo model

- Currently only the **marsim\_gazebo** package
- **config**: holds the configuration files such as the roz\_gz\_bridge yaml file
- **env-hooks**: holds the files required to define paths
- **launch**: holds the python launch files
- **models**: holds folders which contain the models in sdf format
- **plugins**: holds the cpp source codes (cc and hh files) for custom-made Gazebo plugins
- **rviz**: holds the Rviz2 configuration yaml files
- **scripts**: holds scripts that can be used to control the BlueROV2
- **worlds**: holds sdf world files

```
▼ MARSIM1
  ▼ src/marsim_gazebo
    > config
    > env-hooks
    > launch
    > models
    > plugins
    > rviz
    > scripts
    > worlds
    M CMakeLists.txt
    ≡ LICENSE-2.0.txt
    📄 package.xml
    $ access_dock.sh
    🐳 Dockerfile
    $ load_dock.sh
    ⓘ README.md
```

# Starting up the model

---

- `ros2 launch marsim_gazebo empty_world.launch.py`
- `ros2 launch marsim_gazebo spawn_bluerov2.launch.py`
- `ros2 launch marsim_gazebo start_bridge.launch.py`
- `ros run marsim_gazebo ros_keyboard_controller bluerov2`

# ROS2-GZ bridge

---

- We need a bridge because ROS2 and Gazebo are separate programs
- Recap:
  - ROS2 (Robot Operating System 2):
    - A flexible framework for writing robot software
    - Provides a collection of tools, libraries, and conventions for robot development
    - Handles communication between different parts of a robotic system
    - Supports multiple programming languages (C++, Python, etc.)
    - Designed for distributed computing and real-time systems
    - Used for both simulated and physical robots
  - Gazebo:
    - A 3D dynamic simulator for robots
    - Provides accurate simulation of robots in complex indoor and outdoor environments
    - Offers realistic sensor feedback and physically plausible interactions
    - Can simulate multiple robots
    - Often used in conjunction with ROS/ROS2 for testing and development

# ROS2-GZ bridge

- Configured using bluerov2\_ros\_gz\_bridge.yaml
- Gazebo is the simulator, and ROS2 is the robot computer
- Environment and robot measurements: Gazebo ---> ROS2
- Robot commands: ROS2 ---> Gazebo

```
src > marsim_gazebo > config > ! bluerov2_ros_gz_bridge.yaml
1  - ros_topic_name: "clock"
2    gz_topic_name: "clock"
3    ros_type_name: "rosgraph_msgs/msg/Clock"
4    gz_type_name: "gz.msgs.Clock"
5    subscriber_queue: 10      # Default 10
6    publisher_queue: 10      # Default 10
7    lazy: true                # Default "false"
8    direction: GZ_TO_ROS      # Default "BIDIRECTIONAL" - Bridge both directions
9                                # "GZ_TO_ROS" - Bridge Gz topic to ROS
10                               # "ROS_TO_GZ" - Bridge ROS topic to Gz
11
12  - ros_topic_name: "/bluerov2/ocean_current"
13    gz_topic_name: "/model/bluerov2/ocean_current"
14    ros_type_name: "geometry_msgs/msg/Vector3"
15    gz_type_name: "gz.msgs.Vector3d"
16    subscriber_queue: 10      # Default 10
17    publisher_queue: 10      # Default 10
18    lazy: true                # Default "false"
19    direction: GZ_TO_ROS      # Default "BIDIRECTIONAL" - Bridge both directions
20                                # "GZ_TO_ROS" - Bridge Gz topic to ROS
21                               # "ROS_TO_GZ" - Bridge ROS topic to Gz
22
23  - ros_topic_name: "/bluerov2/joint_states"
24    gz_topic_name: "/world/ocean/model/bluerov2/joint_state"
25    ros_type_name: "sensor_msgs/msg/JointState"
26    gz_type_name: "gz.msgs.Model"
27    subscriber_queue: 10      # Default 10
28    publisher_queue: 10      # Default 10
29    lazy: true                # Default "false"
30    direction: GZ_TO_ROS      # Default "BIDIRECTIONAL" - Bridge both directions
31                                # "GZ_TO_ROS" - Bridge Gz topic to ROS
32                               # "ROS_TO_GZ" - Bridge ROS topic to Gz
33
```

```
[parameter_bridge-1] [INFO] [ros_gz_bridge]: Creating GZ->ROS Bridge: [clock (gz.msgs.Clock) -> clock (rosgraph_msgs/msg/Clock)] (Lazy 1)
[parameter_bridge-1] [INFO] [ros_gz_bridge]: Creating GZ->ROS Bridge: [/model/bluerov2/ocean_current (gz.msgs.Vector3d) -> /bluerov2/ocean_current (geometry_msgs/msg/Vector3)] (Lazy 1)
[parameter_bridge-1] [INFO] [ros_gz_bridge]: Creating GZ->ROS Bridge: [/world/ocean/model/bluerov2/joint_state (gz.msgs.Model) -> /bluerov2/joint_states (sensor_msgs/msg/JointState)] (Lazy 1)
[parameter_bridge-1] [INFO] [ros_gz_bridge]: Creating GZ->ROS Bridge: [/model/bluerov2/pose (gz.msgs.Pose_V) -> /tf (tf2_msgs/msg/TFMessage)] (Lazy 1)
[parameter_bridge-1] [INFO] [ros_gz_bridge]: Creating GZ->ROS Bridge: [/model/bluerov2/pose_static (gz.msgs.Pose_V) -> /tf_static (tf2_msgs/msg/TFMessage)] (Lazy 1)
[parameter_bridge-1] [INFO] [ros_gz_bridge]: Creating GZ->ROS Bridge: [/model/bluerov2/odometry_with_covariance (gz.msgs.OdometryWithCovariance) -> /odometry (nav_msgs/msg/Odometry)] (Lazy 1)
[parameter_bridge-1] [INFO] [ros_gz_bridge]: Creating GZ->ROS Bridge: [/model/bluerov2/imu (gz.msgs.IMU) -> /bluerov2/imu (sensor_msgs/msg/Imu)] (Lazy 1)
[parameter_bridge-1] [INFO] [ros_gz_bridge]: Creating GZ->ROS Bridge: [/model/bluerov2/dvl_ros_compatible/velocity (gz.msgs.TwistWithCovariance) -> /bluerov2/dvl/velocity_notStamped (geometry_msgs/msg/TwistWithCovariance)] (Lazy 1)
[parameter_bridge-1] [INFO] [ros_gz_bridge]: Creating GZ->ROS Bridge: [/model/bluerov2/dvl_ros_compatible/range (gz.msgs.Float) -> /bluerov2/dvl/range (std_msgs/msg/Float32)] (Lazy 1)
[parameter_bridge-1] [INFO] [ros_gz_bridge]: Creating ROS->GZ Bridge: [/bluerov2/thruster1/cmd_thrust (std_msgs/msg/Float64) -> /model/bluerov2/joint/thruster1_joint/cmd_thrust (gz.msgs.Double)] (Lazy 1)
[parameter_bridge-1] [INFO] [ros_gz_bridge]: Creating ROS->GZ Bridge: [/bluerov2/thruster2/cmd_thrust (std_msgs/msg/Float64) -> /model/bluerov2/joint/thruster2_joint/cmd_thrust (gz.msgs.Double)] (Lazy 1)
[parameter_bridge-1] [INFO] [ros_gz_bridge]: Creating ROS->GZ Bridge: [/bluerov2/thruster3/cmd_thrust (std_msgs/msg/Float64) -> /model/bluerov2/joint/thruster3_joint/cmd_thrust (gz.msgs.Double)] (Lazy 1)
[parameter_bridge-1] [INFO] [ros_gz_bridge]: Creating ROS->GZ Bridge: [/bluerov2/thruster4/cmd_thrust (std_msgs/msg/Float64) -> /model/bluerov2/joint/thruster4_joint/cmd_thrust (gz.msgs.Double)] (Lazy 1)
[parameter_bridge-1] [INFO] [ros_gz_bridge]: Creating ROS->GZ Bridge: [/bluerov2/thruster5/cmd_thrust (std_msgs/msg/Float64) -> /model/bluerov2/joint/thruster5_joint/cmd_thrust (gz.msgs.Double)] (Lazy 1)
[parameter_bridge-1] [INFO] [ros_gz_bridge]: Creating ROS->GZ Bridge: [/bluerov2/thruster6/cmd_thrust (std_msgs/msg/Float64) -> /model/bluerov2/joint/thruster6_joint/cmd_thrust (gz.msgs.Double)] (Lazy 1)
```

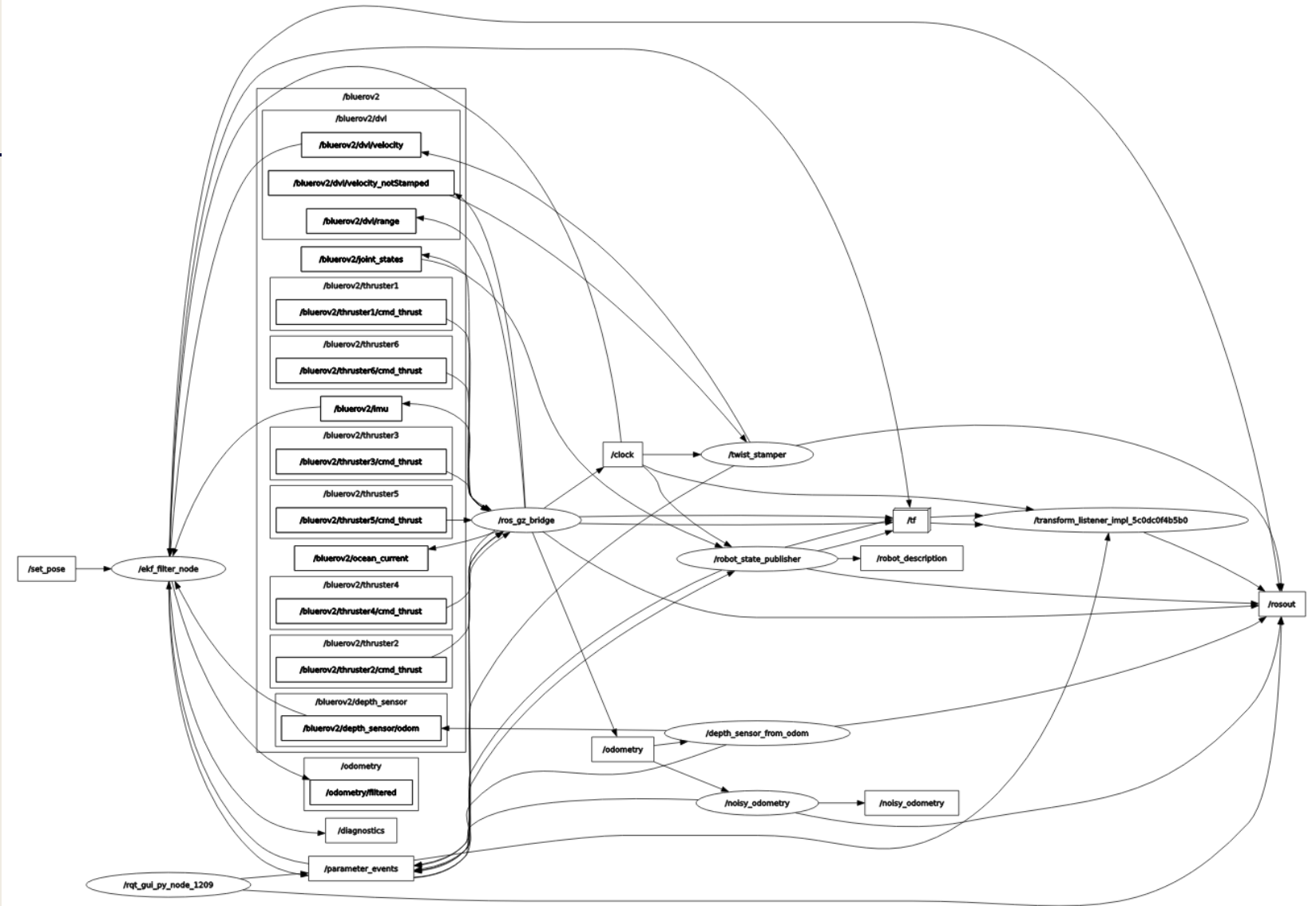
# rqt\_graph

## ○ Important nodes

- /ros\_gz\_bridge
- /ekf\_filter
- /robot\_state\_publisher
- /depth\_sensor\_from\_odom
- /noisy\_odometry
- /twist\_stamper

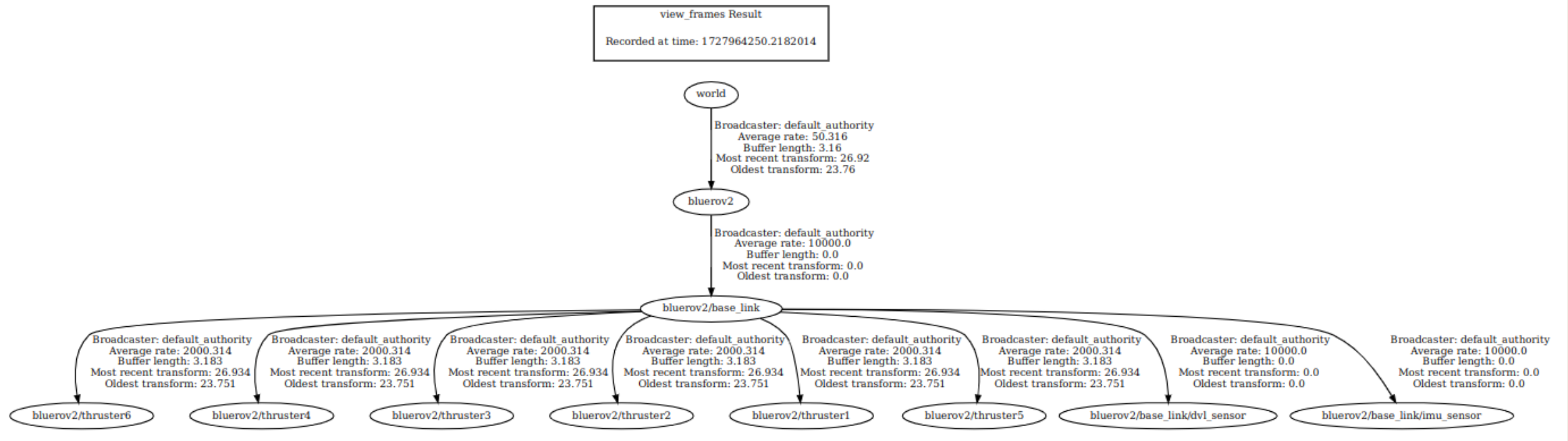
## ○ Important topics

- /bluerov2
- /tf
- /robot\_description
- odometry
- odometry/filtered





# TF2 tree

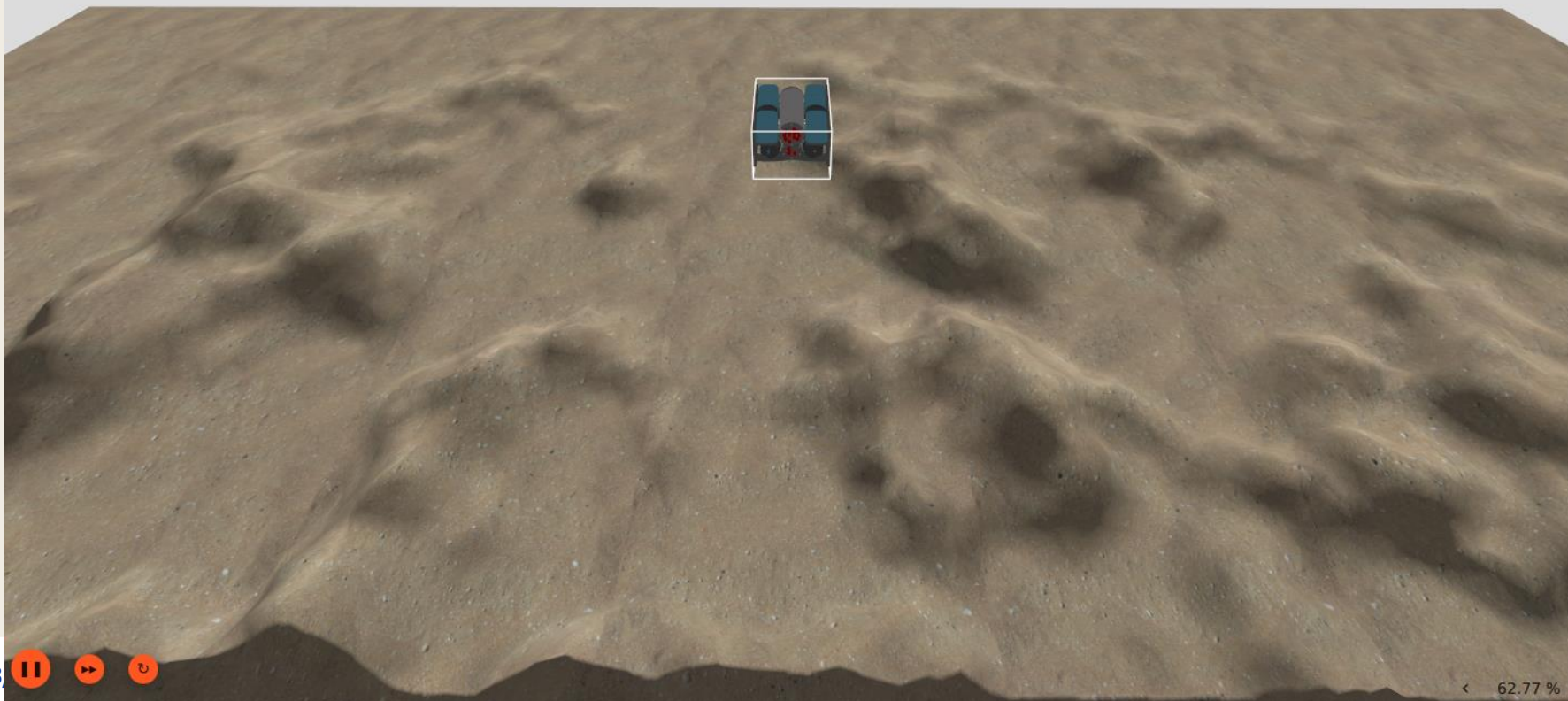


- No bluerov2/base\_link/depth\_sensor because the depth sensor is simulated using pose z measurements from /odometry with gaussian noise added

# EKF filter

---

- Implemented using ROS2 robot localization package
- Configured using bluerov2\_ekf\_config.yaml
- imu0: standard Gazebo IMU sensor
  - Orientations, angular velocities, linear accelerations
- twist0
  - Linear velocities
- odom0
  - Only pose z
- Can be tuned through the definition of the covariances



Model + Entity 20

- System Plugin Info
  - + JointStatePublisher
  - + PosePublisher
  - + OdometryPublisher
  - + DVLRosConverter
  - + Hydrodynamics
  - + Thruster
  - + Thruster
  - + Thruster
  - + Thruster
  - + Thruster
  - + Thruster

Name	bluerov2
Wind Model	

Entity Tree +

- > water\_plane
- > axes
- > sand\_heightmap
- > sun
- ▼ bluerov2
  - ▼ base\_link
    - base\_link\_visual
    - base\_link\_collision
    - imu\_sensor
    - dvl\_sensor
  - ▼ thruster1
    - thruster\_prop\_visual
  - > thruster2
  - > thruster3
  - > thruster4
  - > thruster5
  - > thruster6
  - ✓ thruster1\_joint
  - ✓ thruster2\_joint
  - ✓ thruster3\_joint
  - ✓ thruster4\_joint
  - ✓ thruster5\_joint

**Displays**

- Global Options
  - Fixed Frame: world
  - Background Color: 48; 48; 48
  - Frame Rate: 30
- Global Status: Ok
  - Fixed Frame: OK
- Grid
  - ☒
- RobotModel
  - Status: Ok
  - Visual Enabled: ☒
  - Collision Enabled: ☐
  - Mass Properties
    - Update Interval: 0
    - Alpha: 1
  - Description Source: Topic
  - Description Topic: /robot\_description
  - TF Prefix: bluerov2
  - Links
- TF
  - Status: Ok
  - Show Names: ☒
  - Show Axes: ☒
  - Show Arrows: ☒
  - Marker Scale: 1
  - Update Interval: 0
  - Frame Timeout: 15
  - Filter (whitelist)
  - Filter (blacklist)
  - Frames
  - Tree
- Odometry
  - ☒

**TF Prefix**

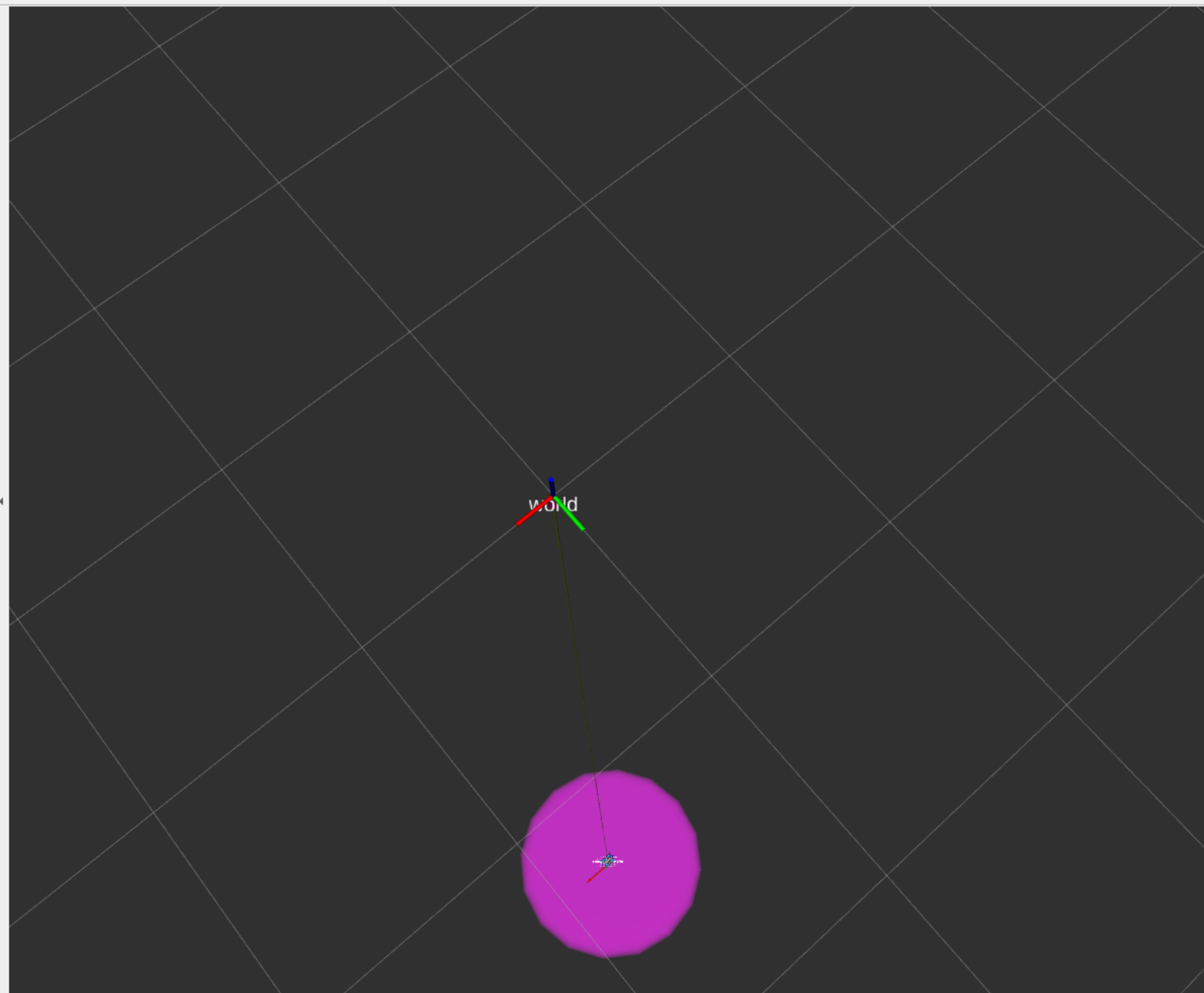
Robot Model normally assumes the link name is the same as the tf frame name. This option allows you to set a prefix. Mainly useful for multi-robot situations.

Add

Duplicate

Remove

Rename



**Views**

Type: Orbit (rviz\_default\_1) Zero

**Current View**

- Near Clip Di...: 0.01
- Invert Z Axis: ☐
- Target Frame: <Fixed Frame>
- Distance: 42.1307
- Focal Shape...: 0.05
- Focal Shape...: ☒
- Yaw: 0.867451
- Pitch: 1.24591
- Focal Point: -7.8417; -8.8489; -...

Save Remove Rename

## Displays

- Global Options
  - Fixed Frame world
  - Background Color 48; 48; 48
  - Frame Rate 30
- Global Status: Ok
  - Fixed Frame OK
- Grid
  - ☒
- RobotModel
  - Status: Ok
  - Visual Enabled ☒
  - Collision Enabled ☐
  - Mass Properties
    - Update Interval 0
    - Alpha 1
  - Description Source Topic
  - Description Topic /robot\_description
  - TF Prefix bluerov2
  - Links
- TF
  - Status: Ok
  - Show Names ☒
  - Show Axes ☒
  - Show Arrows ☒
  - Marker Scale 1
  - Update Interval 0
  - Frame Timeout 15
  - Filter (whitelist)
  - Filter (blacklist)
  - Frames
  - Tree
- Odometry
  - ☒

## TF Prefix

Robot Model normally assumes the link name is the same as the tf frame name. This option allows you to set a prefix. Mainly useful for multi-robot situations.

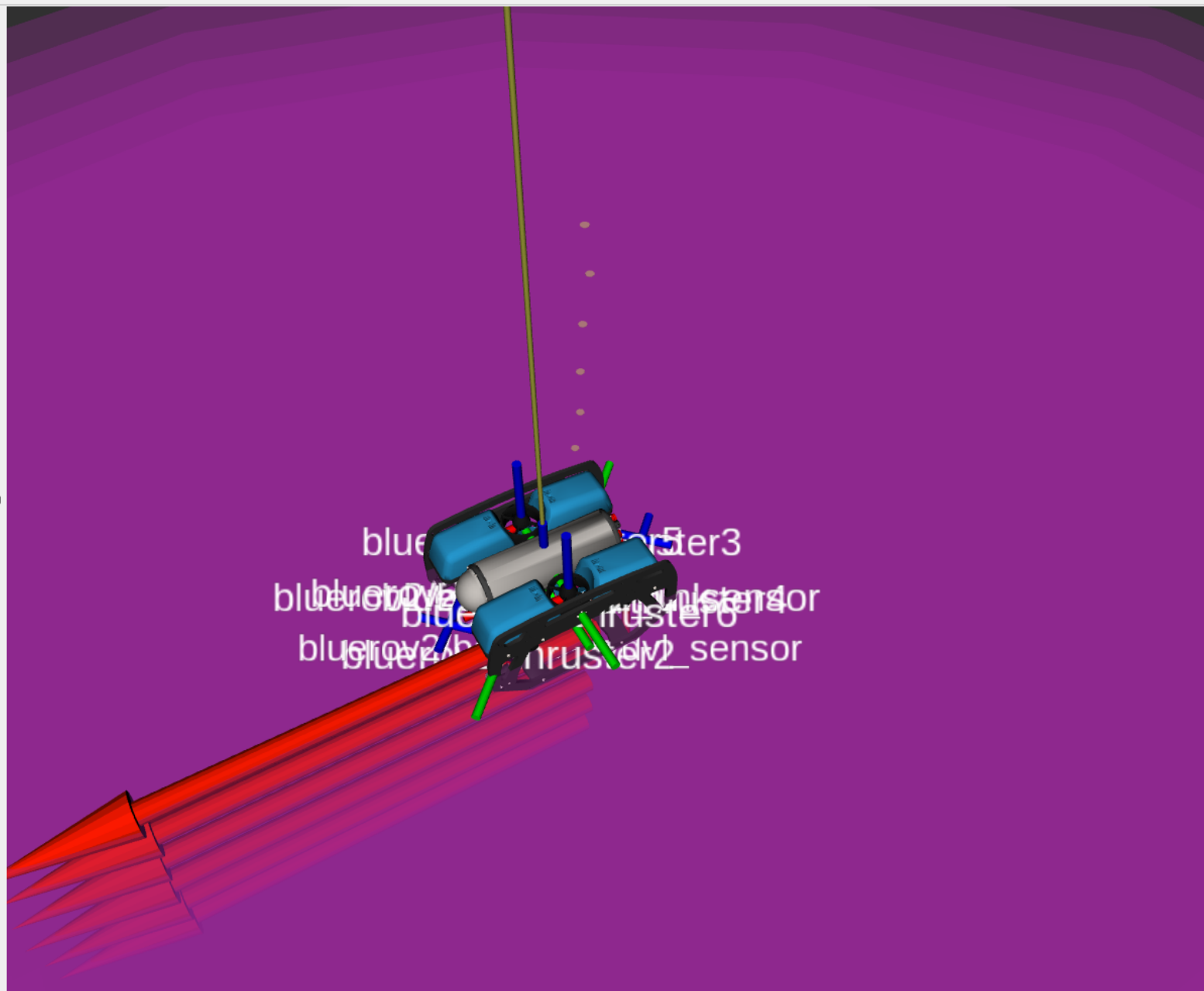
Add

Duplicate

Remove

Rename

Time



## Views

Type: Orbit (rviz\_default\_1) Zero

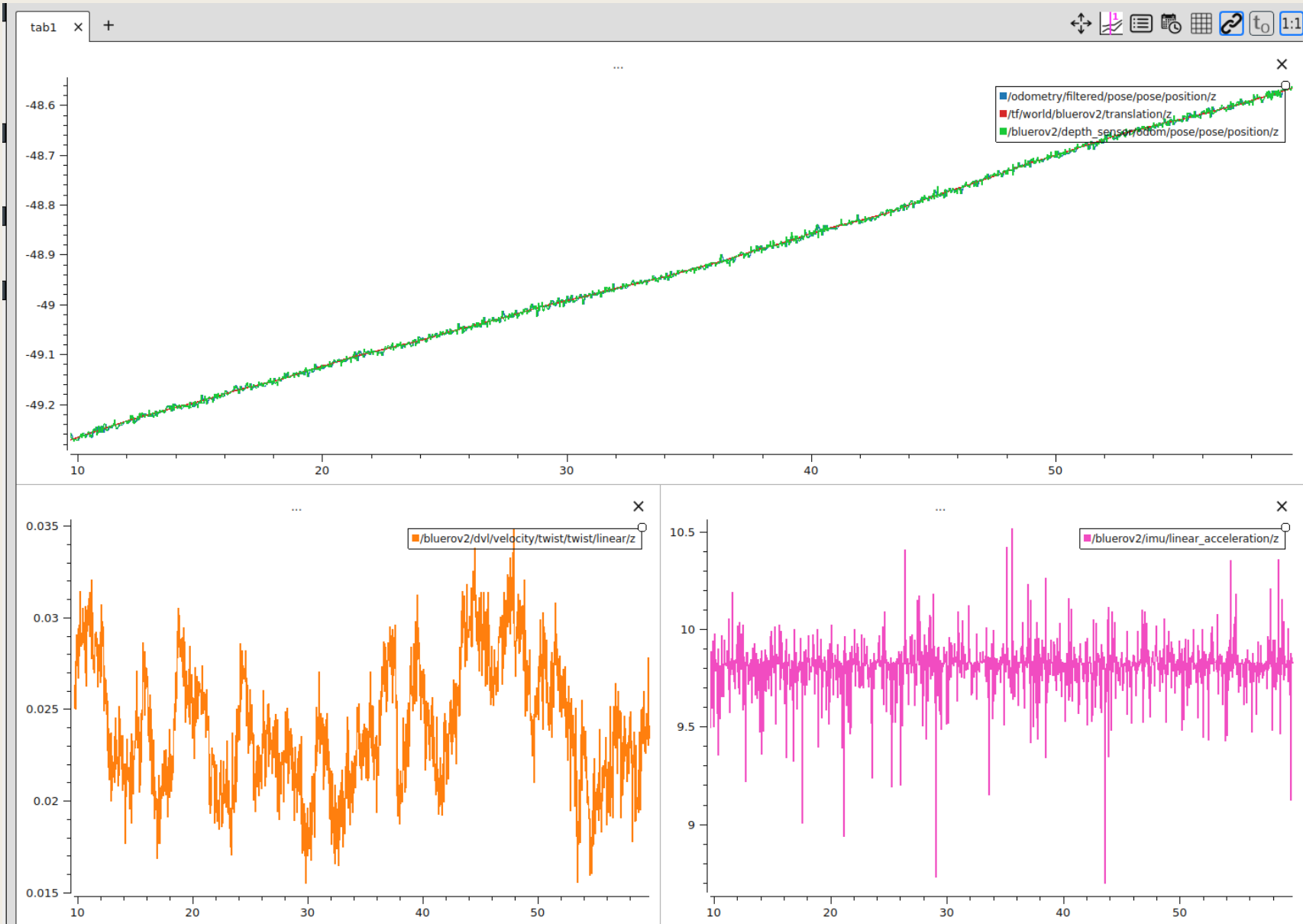
- Current View
  - Near Clip Di... 0.01
  - Invert Z Axis ☐
  - Target Frame <Fixed Frame>
  - Distance 1.67249
  - Focal Shape... 0.05
  - Focal Shape... ☒
  - Yaw 1.05745
  - Pitch 0.86591
  - Focal Point 0.1264; 0.49305; -...

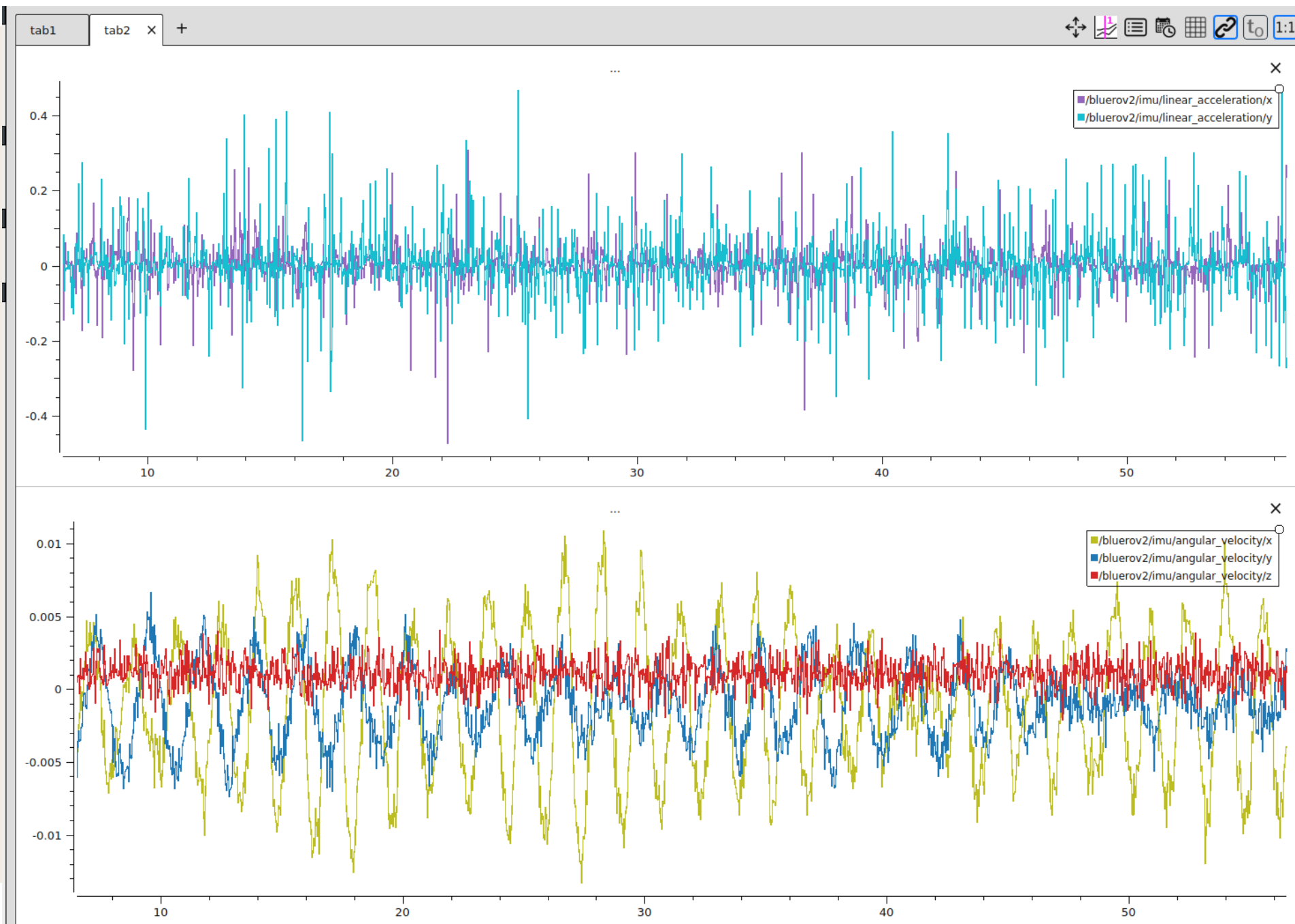
Save

Remove

Rename







An aerial photograph of a coastal city, likely Stavanger, Norway. In the foreground, a lush green hill features a prominent white tower with a red roof and a spire. The city is spread out below the hill, with a mix of residential and commercial buildings. The city is situated along a coastline with a large body of water and several islands in the background. The sky is blue with scattered white clouds.

**Thank you for your attention!**