ROS2/Gazebo - Marine Simulator v1.1

An international university campus across 1300 acres of land

Professor Yihan Xing
Study program leader, MSc Marine and Offshore

October 13, 2024

Updates from MARSIM v1.0

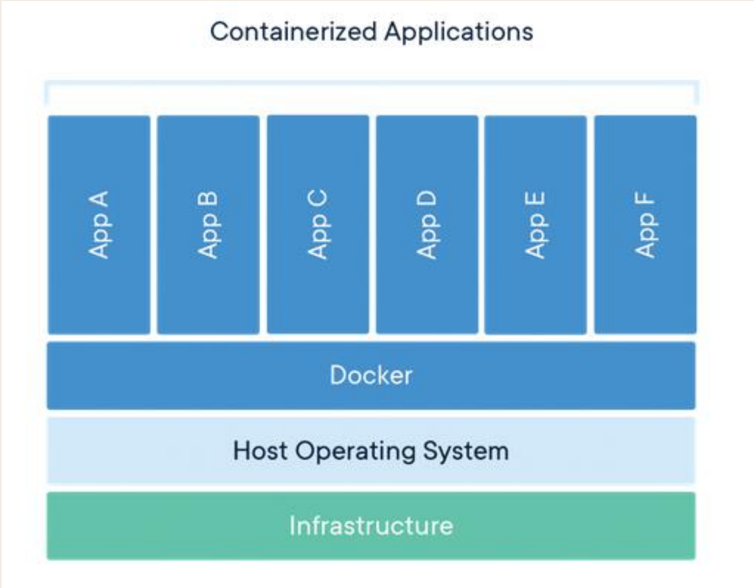
- Improved documentation
  - Theory behind extended karman filter (EKF)
  - References folder
- Upgraded marsim_gazebo package
  - Updated models
    - Added camera and depth camera to models/bluerov2
    - Created namespace for models/bluerov2
    - worlds/empty.world has now vessel_g in the middle
  - New models
    - models/sst
    - models/vessel_g
- Added marsim_control package
  - scripts/bluerov2_pid_pose_z_control.py
- Added matlab_tools
  - Supplementary tools for tuning of controllers, estimation of hydrodynamic coefficients, etc.

Agenda

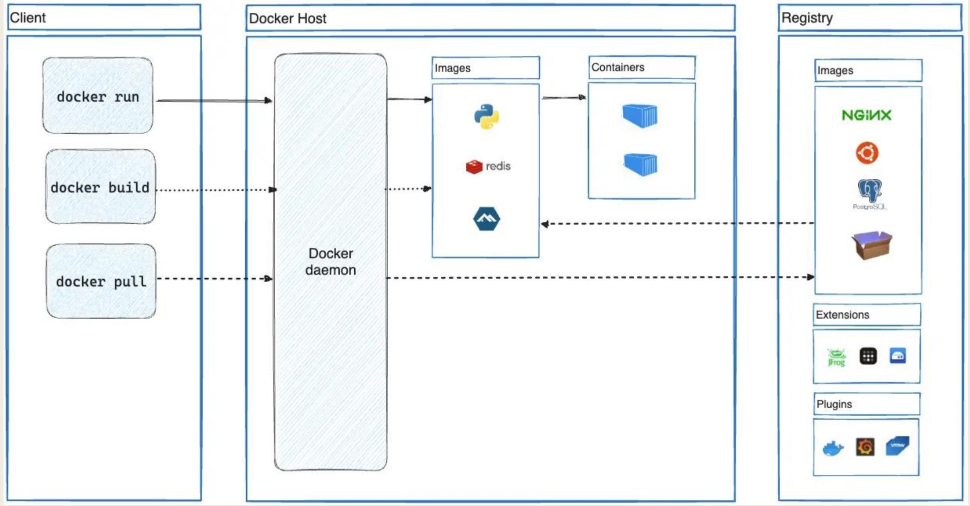
- Docker Engine
- ROS2
- Gazebo
- Marine Simulator Gazebo (marsim_gazebo) package
- Marine Simulator Control (marsim_control) package

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 defintion 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 > ...

```
# ----- Define base image -----
FROM nvidia/cuda:12.6.1-base-ubuntu24.04

# ----- Define environmental variables -----
ENV HOME=/home
ENV TZ=Europe/Oslo
RUN ln -snf /usr/share/zoneinfo/$TZ /etc/localtime && echo $TZ > /etc/timezone

# ----- Install ROS2 Jazzy -----
RUN apt update && apt install -y locales
RUN locale-gen en_US en_US.UTF-8
RUN update-locale LC_ALL=en_US.UTF-8 LANG=en_US.UTF-8
ENV LANG=en_US.UTF-8
RUN apt install -y software-properties-common
RUN add-apt-repository universe
RUN apt update && apt install curl -y
RUN curl -sSL https://raw.githubusercontent.com/ros/rosdistro/master/ros.key -o
RUN echo "deb [arch=$(dpkg --print-architecture) signed-by=/usr/share/keyrings/
RUN apt update && apt install -y ros-dev-tools
RUN apt update -y && apt upgrade -y
RUN apt install -y ros-jazzy-desktop
RUN echo "source /opt/ros/jazzy/setup.bash" >> ~/.bashrc

# ----- Install Gazebo Harmonic -----
RUN apt-get update
RUN apt-get install -y lsb-release gnupg
RUN curl https://packages.osrfoundation.org/gazebo.gpg --output /usr/share/keyri
RUN echo "deb [arch=$(dpkg --print-architecture) signed-by=/usr/share/keyrings/
RUN apt-get update
RUN apt-get install -y gz-harmonic

# ----- Install Gazebo/ROS pairing -----
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

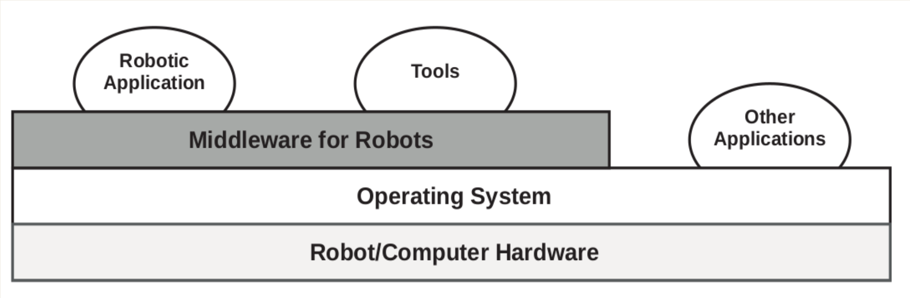
```
$ load_dock.sh
#! /usr/bin/bash

xhost +local:docker

docker run \
-it --rm \
--privileged \
--gpus all \
--name marsim_cont \
--env="XDG_RUNTIME_DIR=/tmp/runtime-root" \
--env="DISPLAY=$DISPLAY" \
--env="QT_X11_NO_MITSHM=1" \
--env="MESA_GL_VERSION_OVERRIDE=3.3" \
--volume="/tmp/.X11-unix:/tmp/.X11-unix" \
--volume="/home/yhxing/Docker_WS/marsim1:/home/marsim_ws:rw" \
--runtime=nvidia \
marsim1_dock
```

access_cont.sh

```
$ access_dock.sh
#! /usr/bin/bash

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

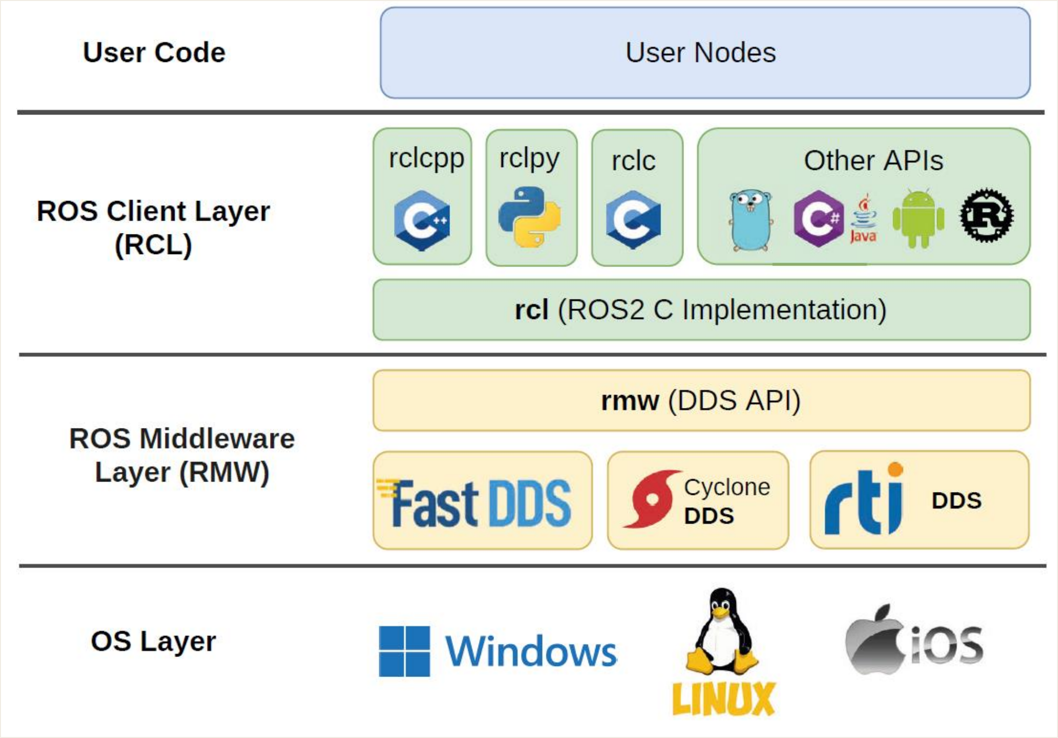


Source: Rico, F. M. (2022). A concise introduction to robot programming with ROS2. Chapman and Hall/CRC

Distro	Release date	Logo	EOL date
Jazzy Jalisco	May 23rd, 2024		May 2029
Iron Irwini	May 23rd, 2023		November 2024
Humble Hawksbill	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

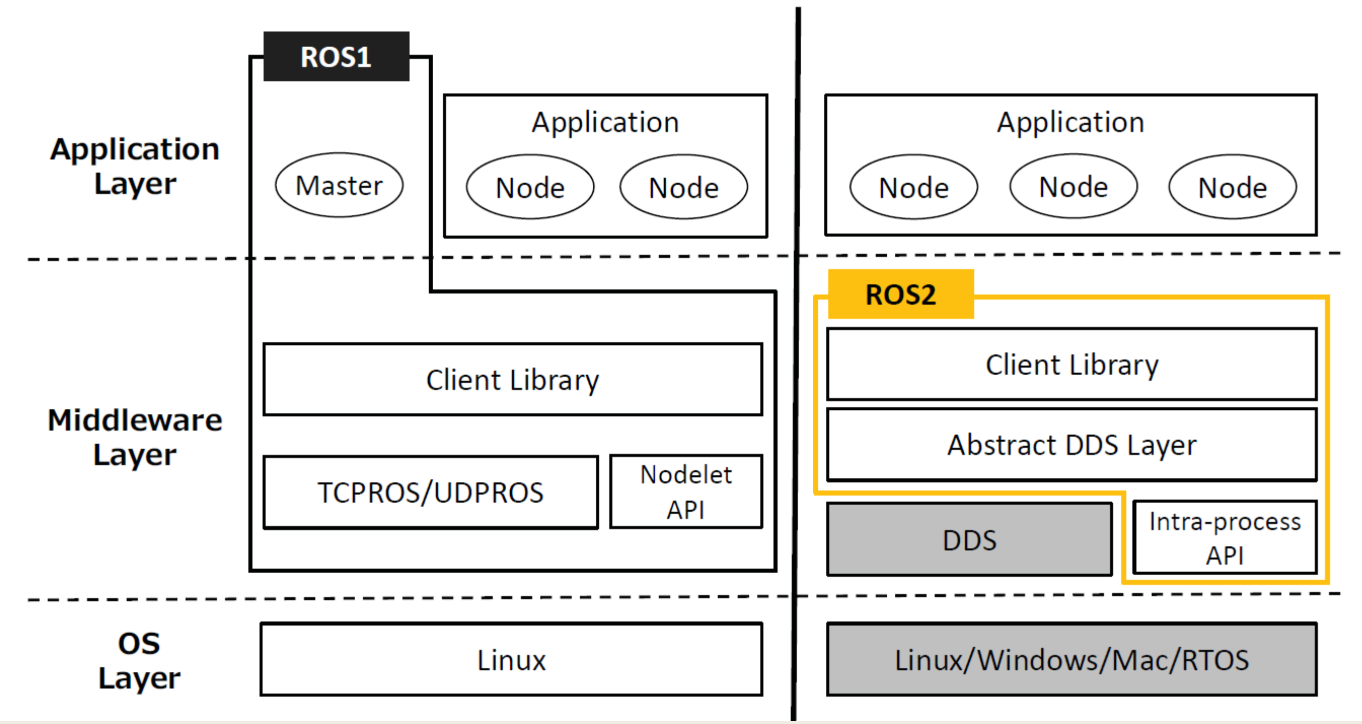
- User nodes interface with ROS2 via RCL's APIs
- rcl is the core of ROS2 and is written in C
- rclcpp, rclpy, rclc, etc binds different programming languages to rcl offering the same behaviour, i.e., they are not independent implementations
- ROS2 is a distributed system using data distribution system (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 seperate libraries roscpp and rospy (ROS1)
- Features
  - QoS, multi-threaded exectution 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

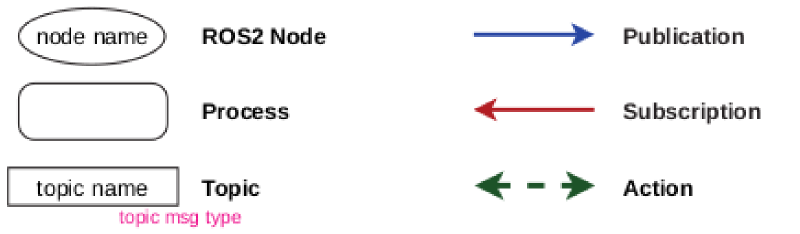


Figure 1.3: Description of symbols used in computer graph diagrams.

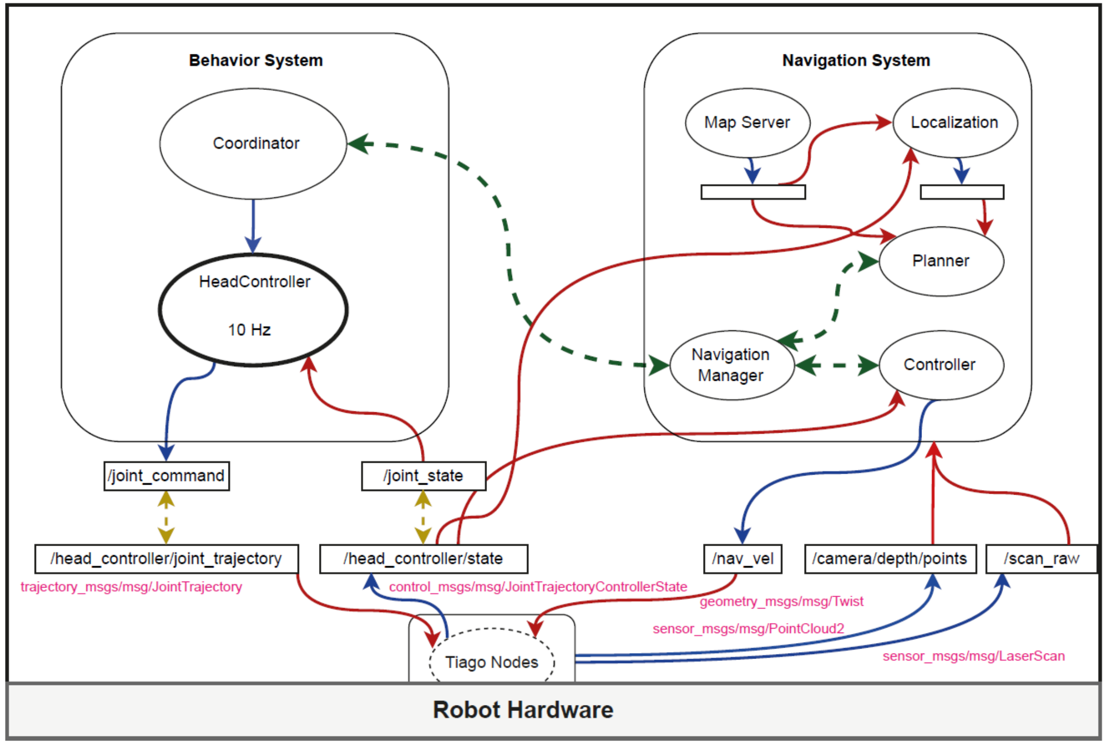
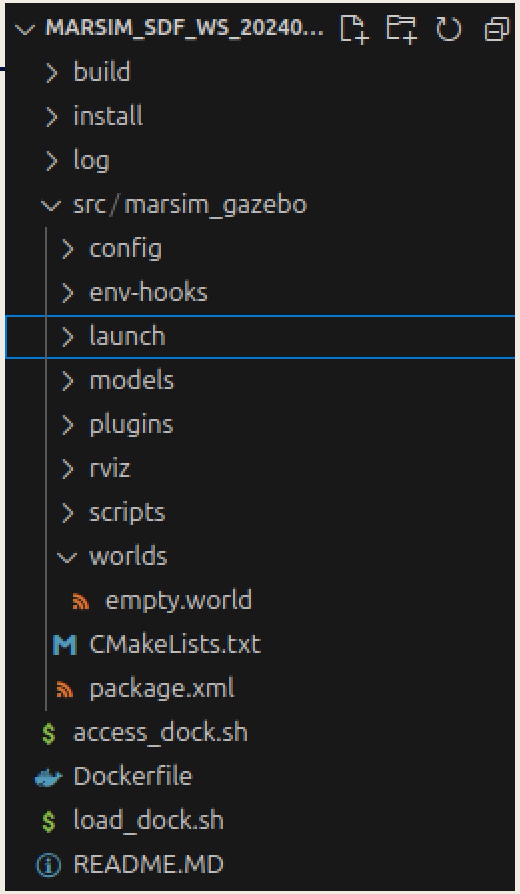


Figure 1.6: Computing graph of behavior-based application for the Tiago robot that uses a navigation subsystem.

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

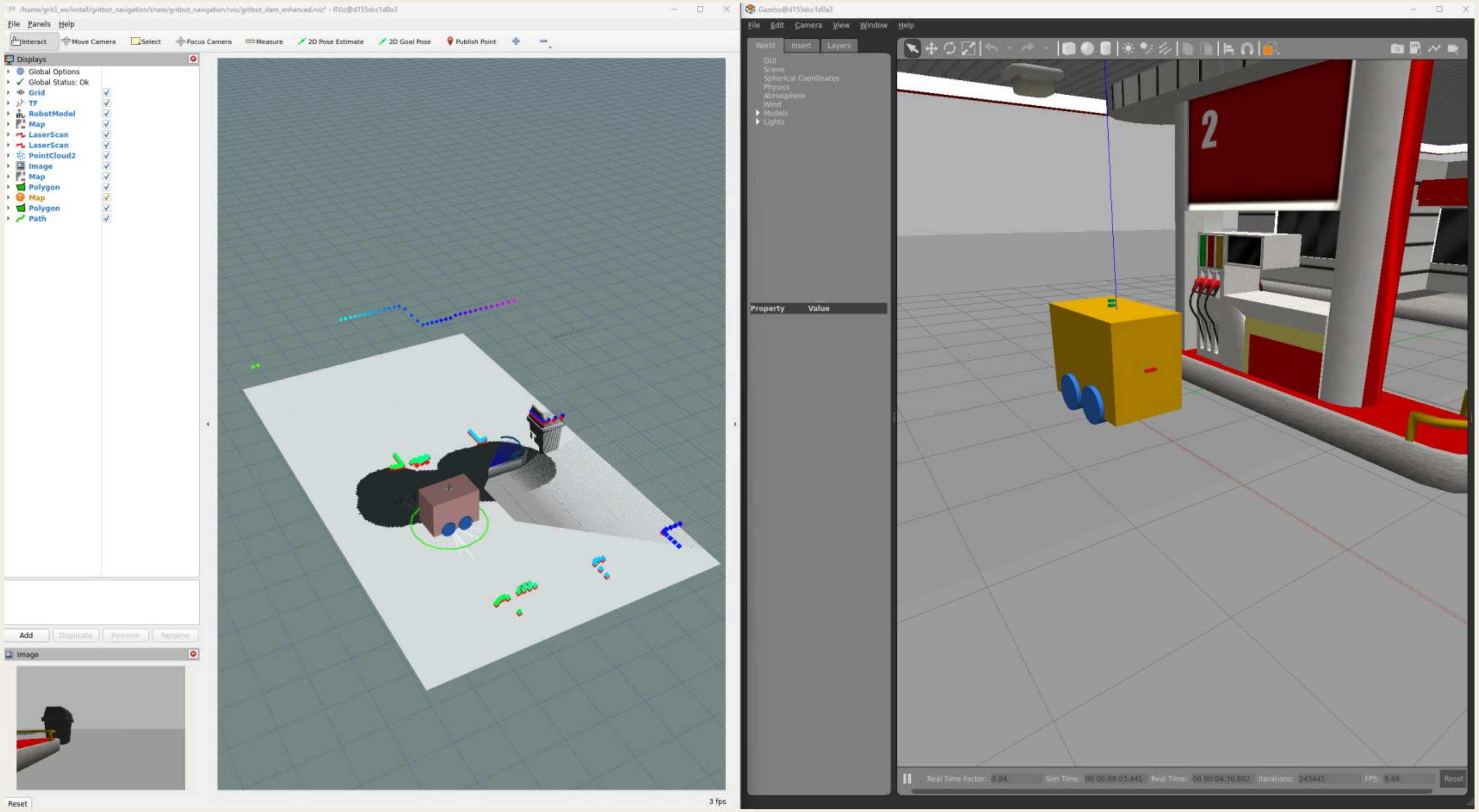
- Must be placed at the root of the package directory
- Defines the build tool, folders to be included, dependencies, etc

```cmake
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
<?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

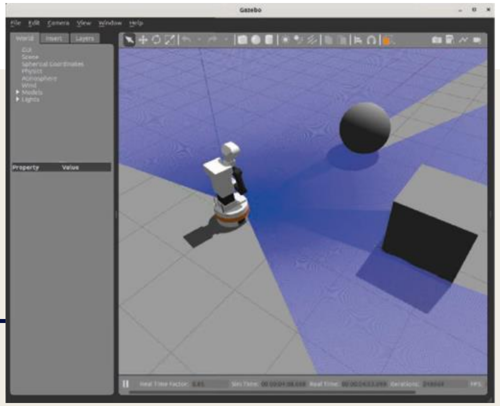
- Port of Rviz to ROS2
- Installed by ROS2 by default
- Provides graphic interface for users to view the robot, sensor data, maps, etc



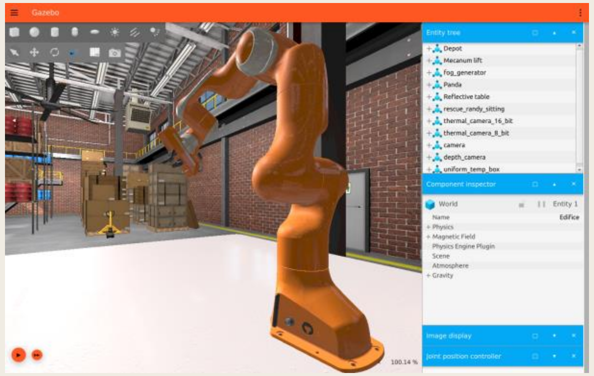
University of Stavanger

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/gazebosim

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

```yaml
- ros_topic_name: "clock"
  gz_topic_name: "clock"
  ros_type_name: "rosgraph_msgs/msg/Clock"
  gz_type_name: "gz.msgs.Clock"
  subscriber_queue: 10      # Default 10
  publisher_queue: 10       # Default 10
  lazy: true                # Default "false"
  direction: GZ_TO_ROS      # Default "BIDIRECTIONAL" - Bridge both directions
                            # "GZ_TO_ROS" - Bridge Gz topic to ROS
                            # "ROS_TO_GZ" - Bridge ROS topic to Gz

- ros_topic_name: "/bluerov2/ocean_current"
  gz_topic_name: "/model/bluerov2/ocean_current"
  ros_type_name: "geometry_msgs/msg/Vector3"
  gz_type_name: "gz.msgs.Vector3d"
  subscriber_queue: 10      # Default 10
  publisher_queue: 10       # Default 10
  lazy: true                # Default "false"
  direction: GZ_TO_ROS    # Default "BIDIRECTIONAL" - Bridge both directions
                            # "GZ_TO_ROS" - Bridge Gz topic to ROS
                            # "ROS_TO_GZ" - Bridge ROS topic to Gz

- ros_topic_name: "/bluerov2/joint_states"
  gz_topic_name: "/world/ocean/model/bluerov2/joint_state"
  ros_type_name: "sensor_msgs/msg/JointState"
  gz_type_name: "gz.msgs.Model"
  subscriber_queue: 10      # Default 10
  publisher_queue: 10       # Default 10
  lazy: true                # Default "false"
  direction: GZ_TO_ROS    # Default "BIDIRECTIONAL" - Bridge both directions
                            # "GZ_TO_ROS" - Bridge Gz topic to ROS
                            # "ROS_TO_GZ" - Bridge ROS topic to Gz
```

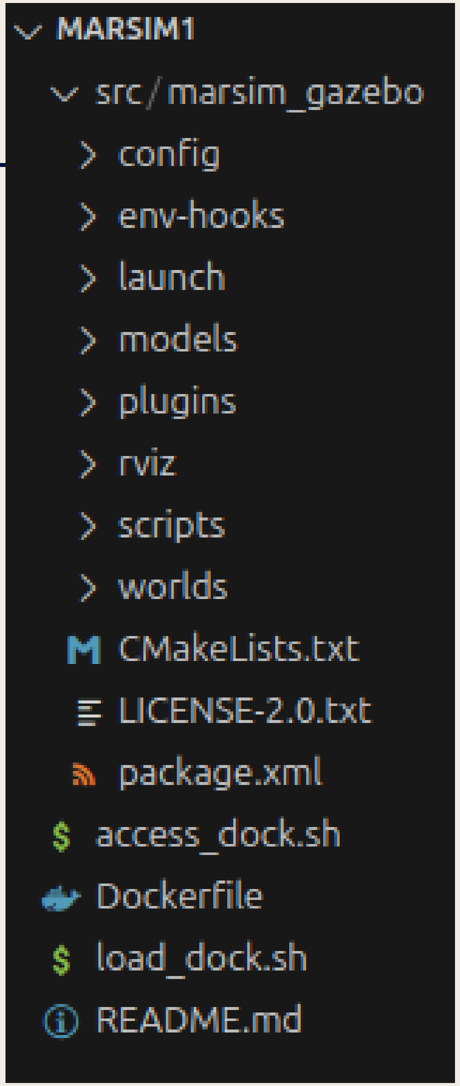
Marine Simulator Gazebo (marsim_gazebo) package

- 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
- Cuurently only BlueROV2 model - main body with 6 thursters
  - 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

marsim1.1 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

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

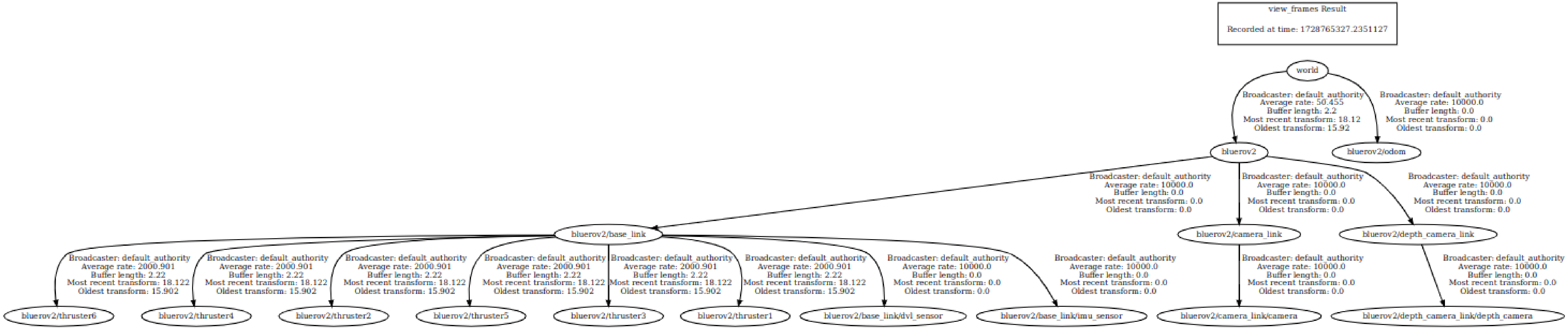
```yaml
- ros_topic_name: "clock"
  gz_topic_name: "clock"
  ros_type_name: "rosgraph_msgs/msg/Clock"
  gz_type_name: "gz.msgs.Clock"
  subscriber_queue: 10       # Default 10
  publisher_queue: 10        # Default 10
  lazy: true                 # Default "false"
  direction: GZ_TO_ROS       # Default "BIDIRECTIONAL" - Bridge both directions
                             # "GZ_TO_ROS" - Bridge Gz topic to ROS
                             # "ROS_TO_GZ" - Bridge ROS topic to Gz

- ros_topic_name: "/bluerov2/ocean_current"
  gz_topic_name: "/model/bluerov2/ocean_current"
  ros_type_name: "geometry_msgs/msg/Vector3"
  gz_type_name: "gz.msgs.Vector3d"
  subscriber_queue: 10       # Default 10
  publisher_queue: 10        # Default 10
  lazy: true                 # Default "false"
  direction: GZ_TO_ROS     # Default "BIDIRECTIONAL" - Bridge both directions
                             # "GZ_TO_ROS" - Bridge Gz topic to ROS
                             # "ROS_TO_GZ" - Bridge ROS topic to Gz

- ros_topic_name: "/bluerov2/joint_states"
  gz_topic_name: "/world/ocean/model/bluerov2/joint_state"
  ros_type_name: "sensor_msgs/msg/JointState"
  gz_type_name: "gz.msgs.Model"
  subscriber_queue: 10       # Default 10
  publisher_queue: 10        # Default 10
  lazy: true                 # Default "false"
  direction: GZ_TO_ROS     # Default "BIDIRECTIONAL" - Bridge both directions
                             # "GZ_TO_ROS" - Bridge Gz topic to ROS
                             # "ROS_TO_GZ" - Bridge ROS topic to Gz
```

```
[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/TwistWithCovar
iance)] (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

- Namespace: bluerov2
- 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

University of Stavanger

EKF – Robot Localization package

- 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 defintion of the covariances

EKF - Introduction

- Purpose: State estimation for nonlinear systems
  - Estimates the state of a dynamic system from noisy measurements
  - Crucial for robotics, autonomous systems, and control applications
- Extends the linear Kalman Filter to nonlinear systems
  - Uses local linearization around the current state estimate
- Key applications:
  - Robot localization and navigation
  - Sensor fusion in autonomous vehicles
  - Spacecraft attitude determination
  - Target tracking in radar systems
- Addresses challenges of:
  - System nonlinearity
  - Measurement and process noise
  - Multi-sensor integration

EKF algorithm

- The EKF algorithm aims to estimate the full 3D pose and velocity (6 pose states and 6 velocity states) of the robot over time
- Consists of two steps: prediction and update
- For more details, see Moore, T., & Stouch, D. (2016). A generalized extended kalman filter implementation for the robot operating system. In Intelligent Autonomous Systems 13: Proceedings of the 13th International Conference IAS-13 (pp. 335-348). Springer International Publishing.

EKF algorithm – Robot state and measurements

- Robot state: $\boldsymbol{x}_k = f(\boldsymbol{x}_{k-1}) + \boldsymbol{w}_{k-1}$
  - $\boldsymbol{x}_k$ is the 12-dimensional state vector (x, y, z, roll, pitch, yaw, vx, vy, vz, wx, wy, wz) at time *k*
  - f is the nonlinear state transition function
  - $\boldsymbol{w}_{k-1}$ is the process noise at time *k-1* which is assumed to be normally distrubuted
- Measurements: $\boldsymbol{z}_k = h(\boldsymbol{x}_k) + \boldsymbol{v}_k$
  - $\boldsymbol{z}_k$ is the measurement at time k
  - h is the nonlinear sensor model that maps the states into the measurement space
  - $\boldsymbol{v}_k$ is the normally distributed sensor noise

EKF algorithm – Prediction step

- Project the current state estimate and error covariance forward in time:
- State prediction: est($\boldsymbol{x}_k$) = $f(\boldsymbol{x}_{k-1})$
  - est($\boldsymbol{x}_k$) is the estimate of the state at time k
  - f is robot model or nonlinear state transition function
  - $\boldsymbol{x}_{k-1}$ is the state at time $k-1$
- Covariance prediction: est($\boldsymbol{P}_k$) = $\boldsymbol{F}\boldsymbol{P}_{k-1}\boldsymbol{F}^T + \boldsymbol{Q}$
  - est($\boldsymbol{P}_k$) is the estimated error covariance projected via $\boldsymbol{F}$ and perturbed by $\boldsymbol{Q}$
  - $\boldsymbol{F}$ is the Jacobian of f
  - $\boldsymbol{Q}$ is the process noise covariance

EKF algorithm – Correction step

- Kalman gain calculation: $\boldsymbol{K} = \mathrm{est}(\boldsymbol{P}_k)\boldsymbol{H}^T(\boldsymbol{H}\,\mathrm{est}(\boldsymbol{P}_k)\boldsymbol{H}^T + \boldsymbol{R})^{-1}$
  - $\boldsymbol{K}$ is the Kalman gain
  - $\boldsymbol{H}$ is the observation matrix, the Jacobian of h, the observation model function or the nonlinear sensor model. To support a wide array of sensors, $\boldsymbol{H}$ is an identity matrix in the robot_localization package.
  - $\boldsymbol{R}$ is the measurement covariance
- State update: $\boldsymbol{x}_k = \mathrm{est}(\boldsymbol{x}_k) + \boldsymbol{K}(\boldsymbol{z} - \boldsymbol{H}\,\mathrm{est}(\boldsymbol{x}_k))$
- Covariance update: $\boldsymbol{P}_k = (\boldsymbol{I} - \boldsymbol{KH})\,\mathrm{est}(\boldsymbol{P}_k)(\boldsymbol{I} - \boldsymbol{KH})^T + \boldsymbol{KRK}^T$

EKF algorithm – Prediction step (more explanations)

- State prediction: est($\boldsymbol{x}_k$) = $f(\boldsymbol{x}_{k-1})$
  - This equation projects the state estimate forward in time
  - *f* represents the robot's motion model. For example, if the robot is moving at a certain velocity, *f* would use this information to predict where the robot will be at the next time step.
  - The key point here is that *f* can be nonlinear, which is what distinguishes the Extended Kalman Filter from the regular Kalman Filter. This allows it to handle more complex systems and motions.

EKF algorithm – Prediction step (more explanations) (2)

- Covariance prediction: est($\boldsymbol{P}_k$) = $\boldsymbol{F}\boldsymbol{P}_{k-1}\boldsymbol{F}^T + \boldsymbol{Q}$
  - $\boldsymbol{F}$ is the Jacobian of f and represents how changes in the previous state affect the current state. It's used to transform the previous covariance into the new time step. The use of the Jacobian $\boldsymbol{F}$ is another key aspect of the Extended Kalman Filter. It's a linear approximation of how the nonlinear function f behaves near the current estimate. This linearization is what allows the EKF to apply the Kalman Filter framework to nonlinear systems.
  - $\boldsymbol{F}\boldsymbol{P}_{k-1}\boldsymbol{F}^T$ propagates the previous uncertainty through the state transition model.
  - $\boldsymbol{Q}$ adds additional uncertainty due to imperfections in our model or external factors we can't account for.

EKF algorithm – Correction step (more explanations)

- Kalman gain calculation: $\boldsymbol{K} = \mathrm{est}(\boldsymbol{P}_k)\boldsymbol{H}^T(\boldsymbol{H}\,\mathrm{est}(\boldsymbol{P}_k)\boldsymbol{H}^T + \boldsymbol{R})^{-1}$
  - $\mathrm{est}(\boldsymbol{P}_k)\boldsymbol{H}^T$ represents how much we expect the measurement to change based on changes in our state estimate
  - $\boldsymbol{H}\,\mathrm{est}(\boldsymbol{P}_k)\boldsymbol{H}^T + \boldsymbol{R}$ represents the total expected variance in our measurement (predicted measurement uncertainty + actual measurement noise).
  - The inverse at the end turns this into a ratio - effectively, "how much should we trust this measurement compared to our prediction?
- State update: $\boldsymbol{x}_k = \mathrm{est}(\boldsymbol{x}_k) + \boldsymbol{K}(\boldsymbol{z} - \boldsymbol{H}\,\mathrm{est}(\boldsymbol{x}_k))$
  - $\boldsymbol{z} - \boldsymbol{H}\,\mathrm{est}(\boldsymbol{x}_k))$ is the difference between the actual measurement $\boldsymbol{z}$ and what we expected to measure $\boldsymbol{K}(\boldsymbol{z} - \boldsymbol{H}\,\mathrm{est}(\boldsymbol{x}_k))$. This is often called the "innovation" or "measurement residual".
  - We multiply this difference by $\boldsymbol{K}$ to determine how much to adjust our prediction. If K is large, we adjust our estimate more towards the new measurement. If K is small, we trust our prediction more.

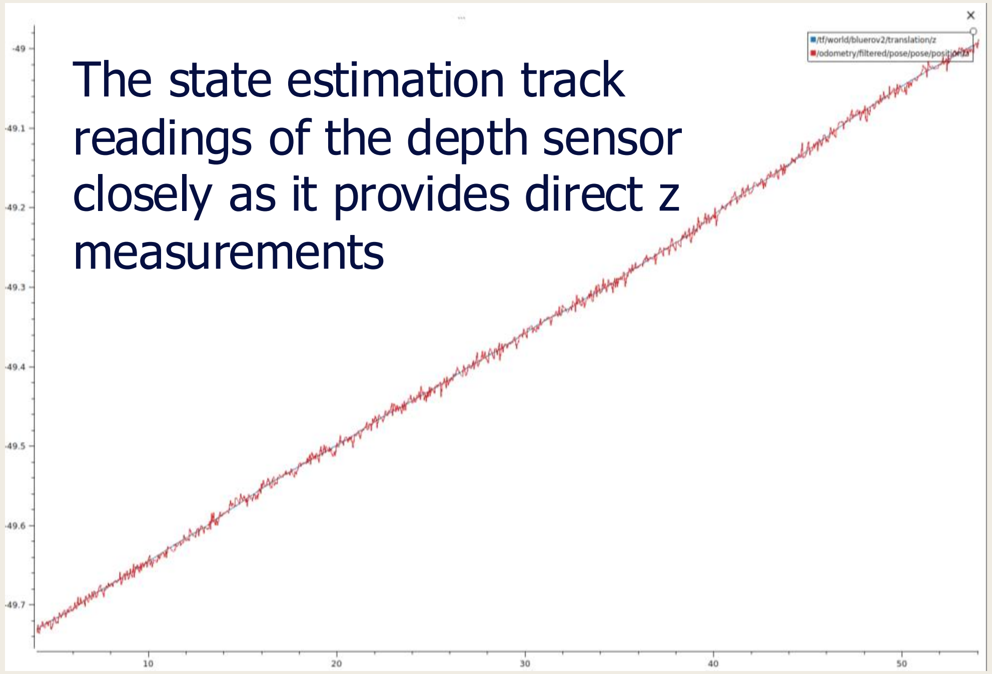
EKF algorithm – Correction step (more explanations) (2)

- Covariance update: $\boldsymbol{P}_k = (\boldsymbol{I} - \boldsymbol{KH})\ \mathrm{est}(\boldsymbol{P}_k)(\boldsymbol{I} - \boldsymbol{KH})^T + \boldsymbol{KRK}^T$
  - $\boldsymbol{I} - \boldsymbol{KH}$ represents how much of our previous uncertainty estimate we're keeping. If $\boldsymbol{K}$ is large (we trusted the measurement a lot), this term becomes smaller.
  - $\mathrm{est}(\boldsymbol{P}_k)(\boldsymbol{I} - \boldsymbol{KH})^T$ reduces our uncertainty based on how much information we gained from the measurement
  - $\boldsymbol{KRK}^T$ adds back some uncertainty based on how noisy our measurement was

State estimations of position z when different combination of sensors are used

Red line = estimated depth; Blue line = actual depth

IMU + DVL + Depth sensors

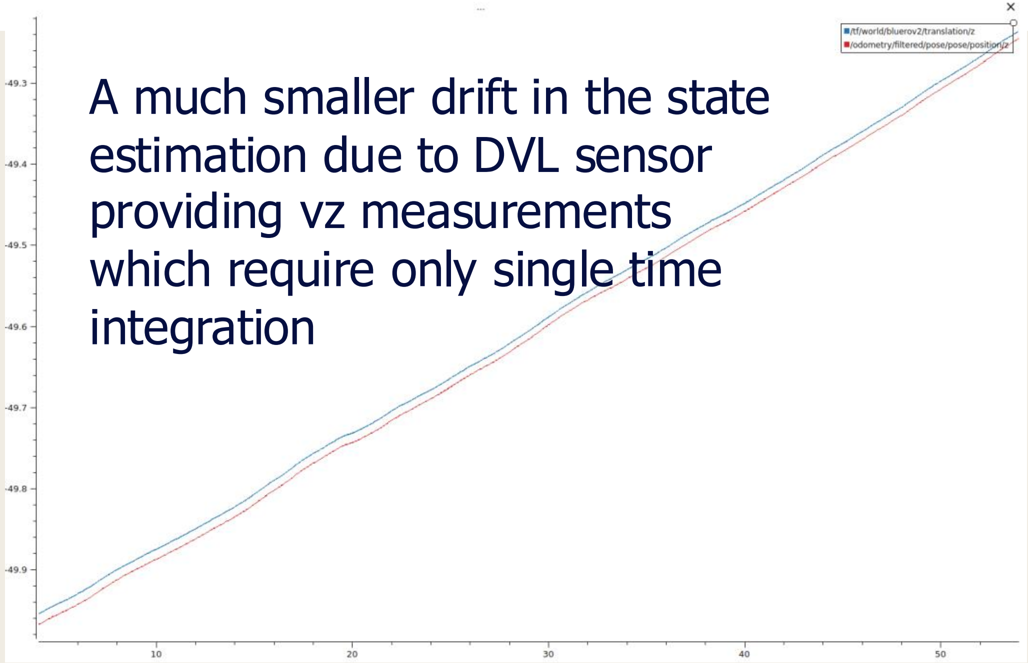


The state estimation track readings of the depth sensor closely as it provides direct z measurements

Measurements from the sensors:

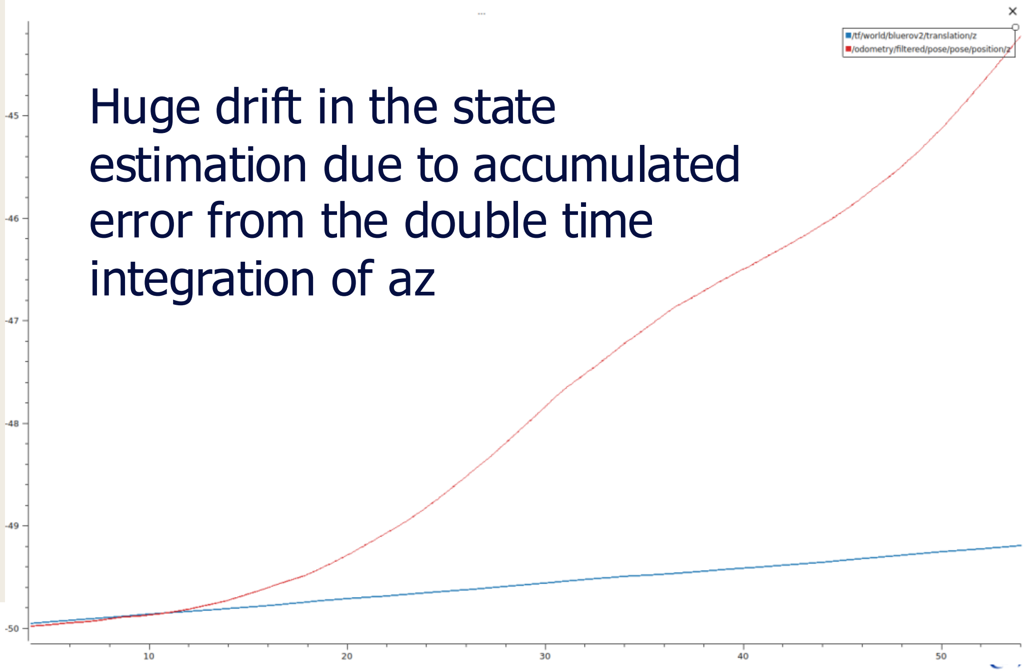
IMU: ax, ay, az, ωx, ωy, ωz
DVL: vx, vy, vz
Depth: z

IMU + DVL sensors

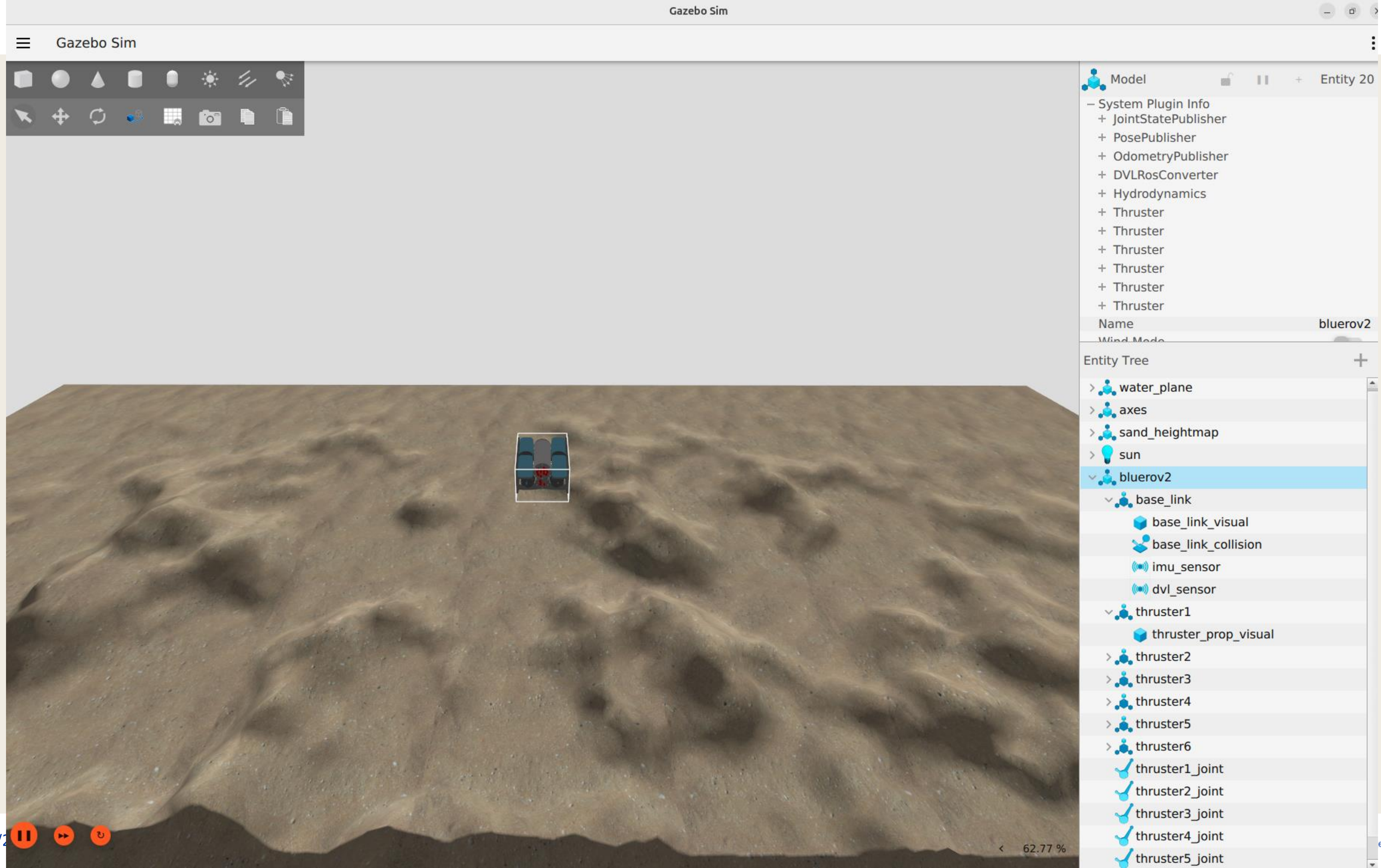


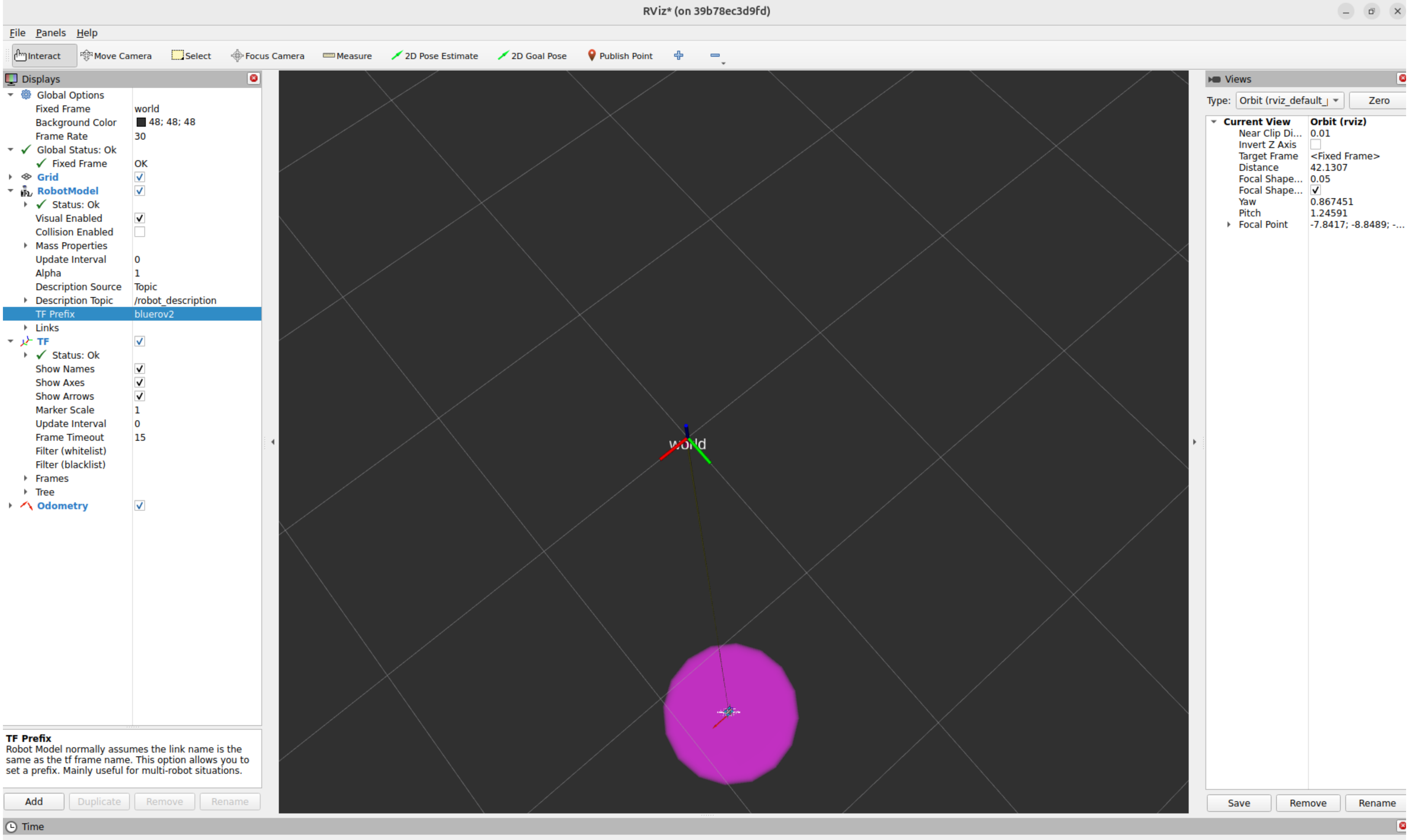
A much smaller drift in the state estimation due to DVL sensor providing vz measurements which require only single time integration

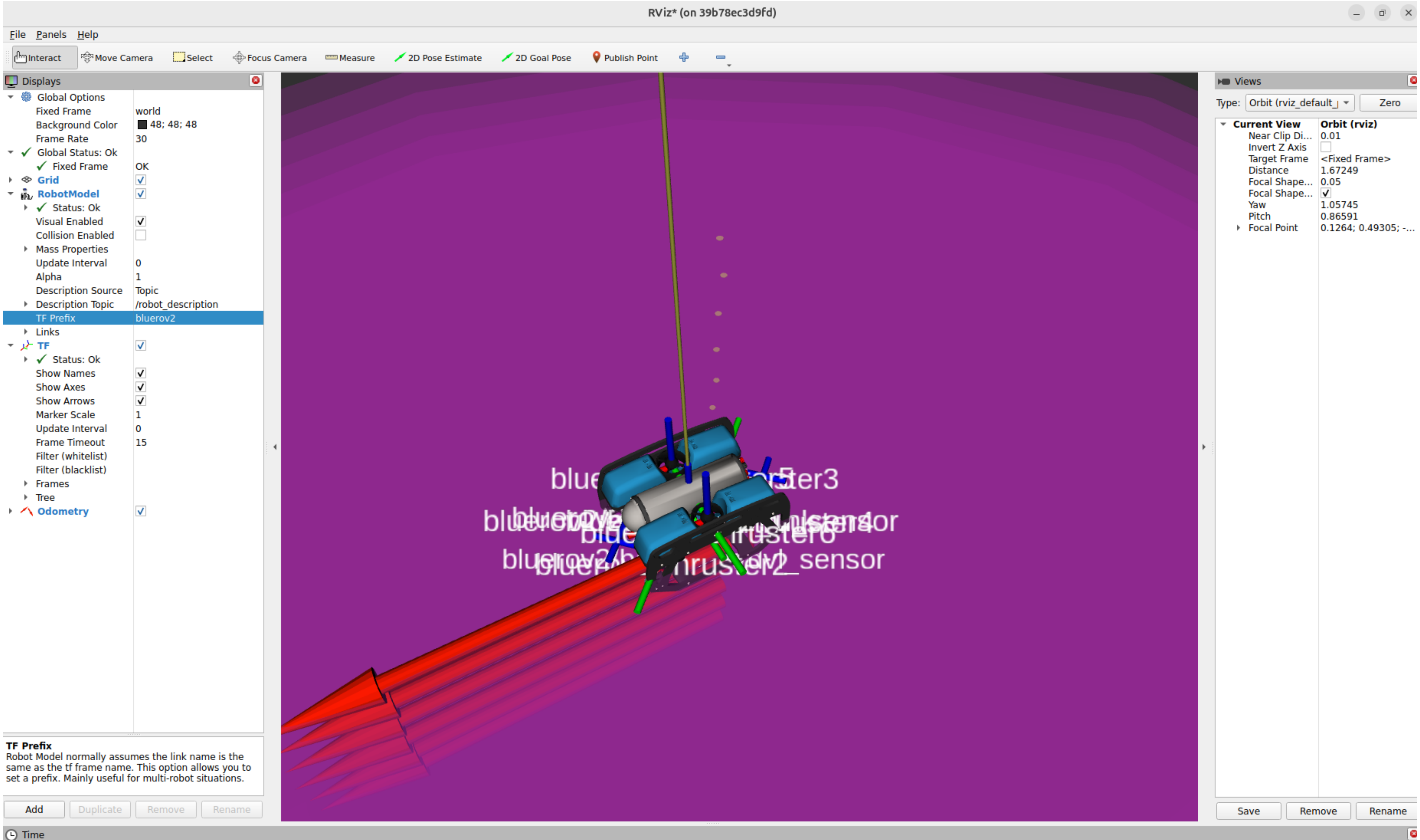
IMU sensor only



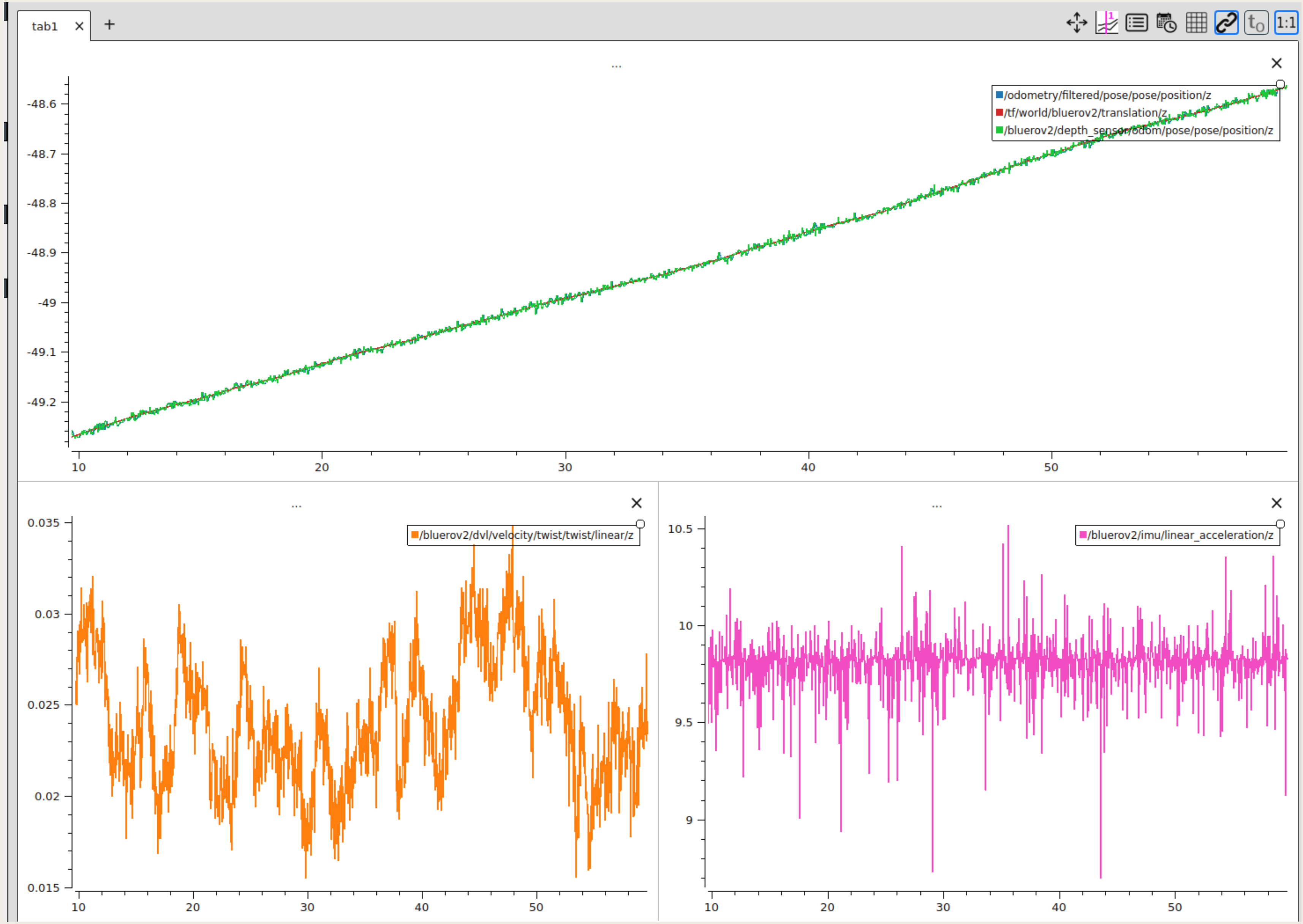
Huge drift in the state estimation due to accumulated error from the double time integration of az


Gazebo Sim
Gazebo Sim
Model
Entity 20
System Plugin Info
JointStatePublisher
PosePublisher
OdometryPublisher
DVLRosConverter
Hydrodynamics
Thruster
Thruster
Thruster
Thruster
Thruster
Thruster
Name
bluerov2
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
62.77 %

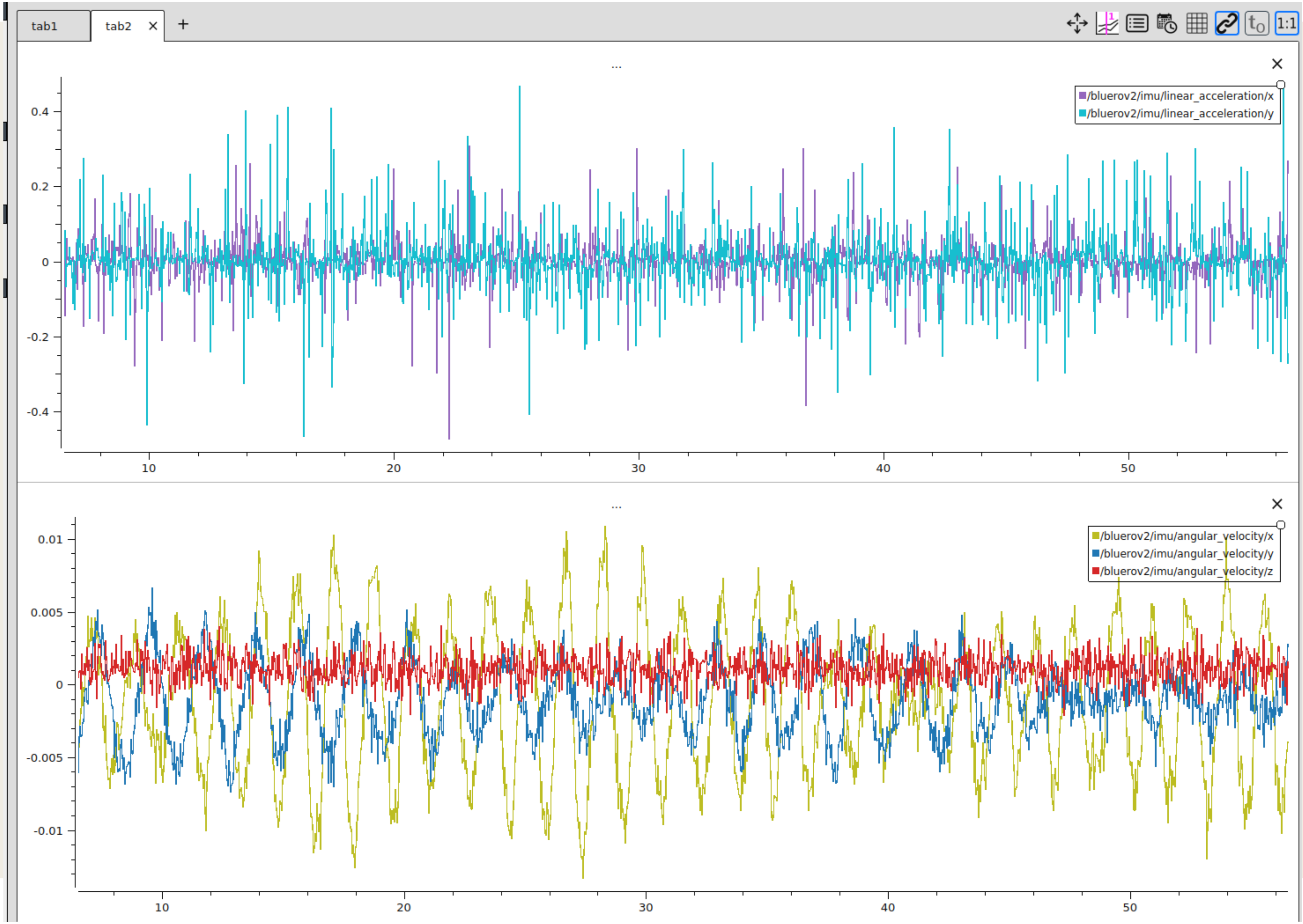

RViz* (on 39b78ec3d9fd)
File
Panels
Help
Interact
Move Camera
Select
Focus Camera
Measure
2D Pose Estimate
2D Goal Pose
Publish Point
Displays
Global Options
Fixed Frame
world
Background Color
48; 48; 48
Frame Rate
30
Global Status: Ok
Fixed Frame
OK
Grid
RobotModel
Status: Ok
Visual Enabled
Collision 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
world
Views
Type:
Orbit (rviz_default_
Zero
Current View
Orbit (rviz)
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
Time



RViz* (on 39b78ec3d9fd)
File
Panels
Help
Interact
Move Camera
Select
Focus Camera
Measure
2D Pose Estimate
2D Goal Pose
Publish Point
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_
Zero
Current View
Orbit (rviz)
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
Time

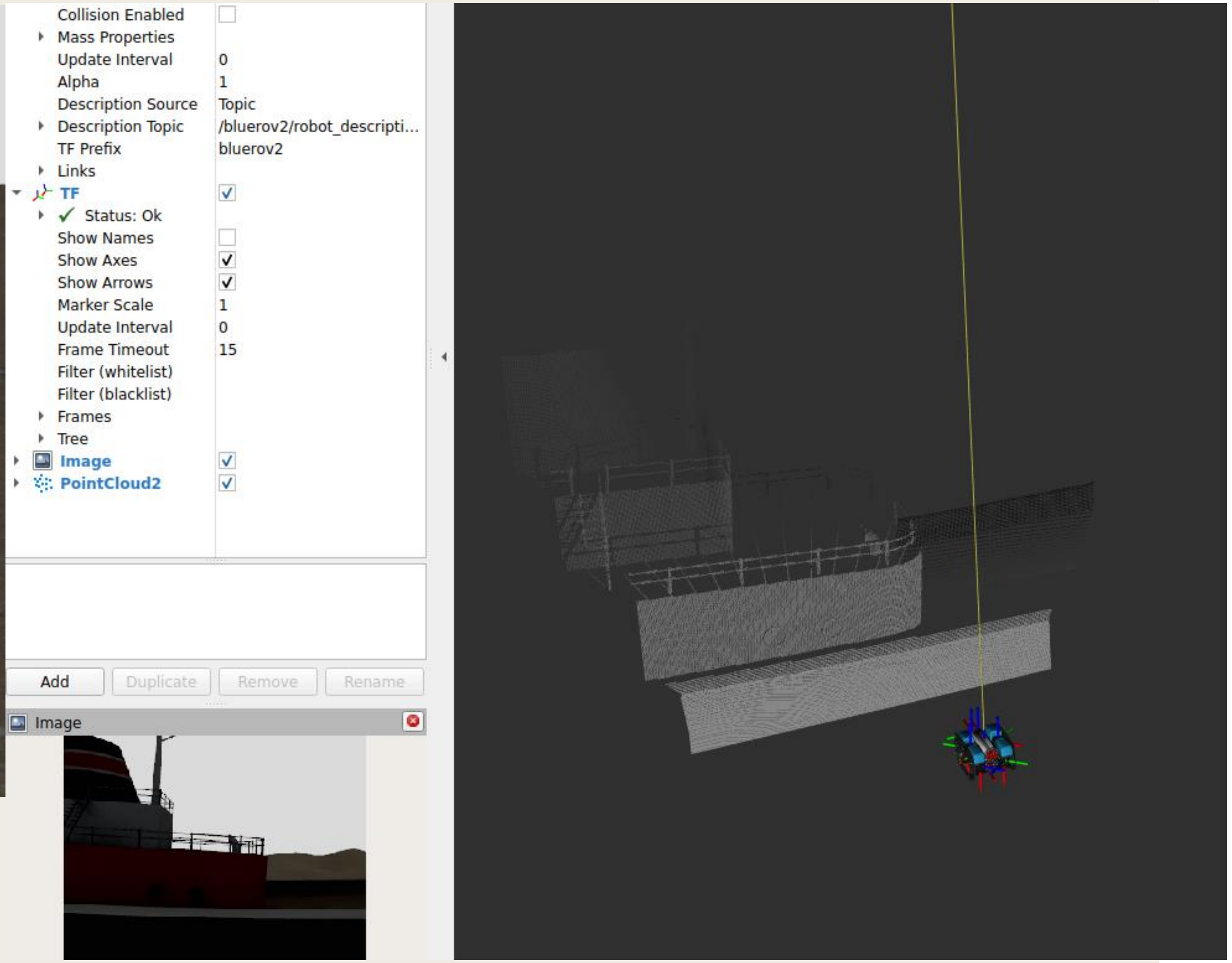


tab1
/odometry/filtered/pose/pose/position/z
/tf/world/bluerov2/translation/z
/bluerov2/depth_sensor/odom/pose/pose/position/z
-48.6
-48.7
-48.8
-48.9
-49
-49.1
-49.2
10
20
30
40
50
/bluerov2/dvl/velocity/twist/twist/linear/z
0.035
0.03
0.025
0.02
0.015
/bluerov2/imu/linear_acceleration/z
10.5
10
9.5
9



tab1
tab2
/bluerov2/imu/linear_acceleration/x
/bluerov2/imu/linear_acceleration/y
/bluerov2/imu/angular_velocity/x
/bluerov2/imu/angular_velocity/y
/bluerov2/imu/angular_velocity/z

With camera and depth camera sensors



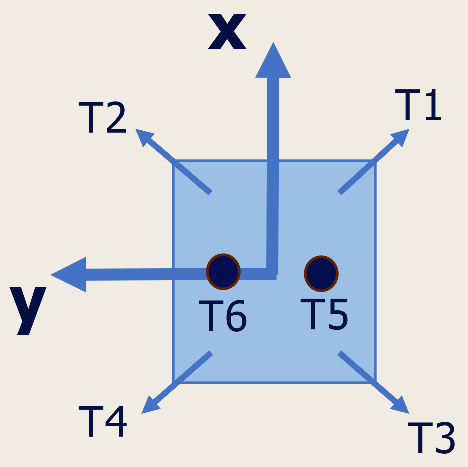
Marine Simulator Control (marsim_control) package

- Package where control-related source code will be placed
- Common scripts where common functions are placed
  - marsim_control/bluerov2_utils.py
  - marsim_control/marsim_utils.py
- scripts/bluerov2_pid_pose_z_control.py

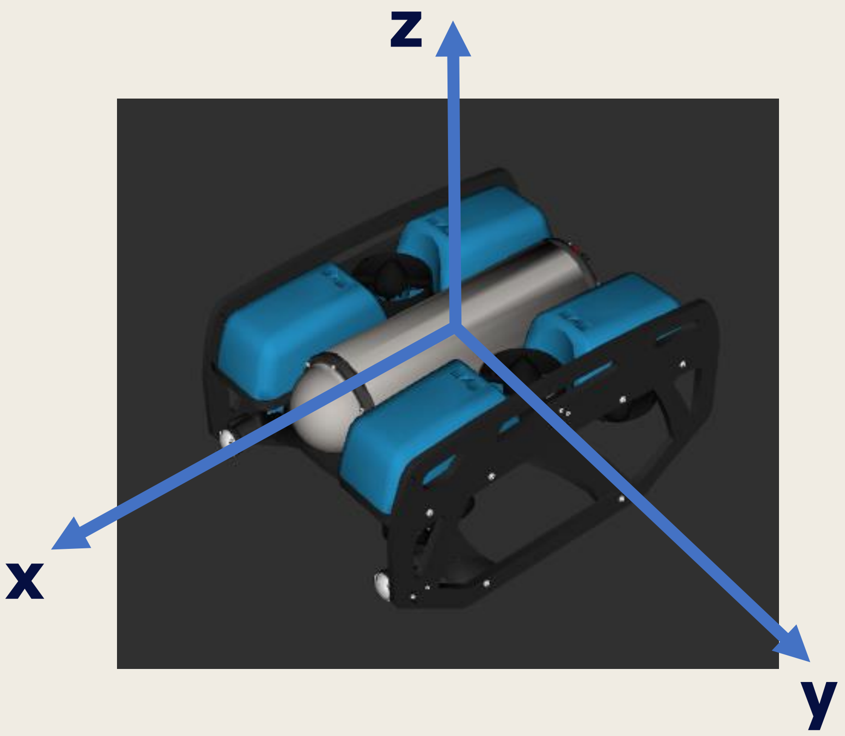
BlueROV2 thruster control configuration

- For thruster control, the BlueROV2 has:
- 12 State vector components:
  - Position (x, y, z)
  - Orientation (roll, pitch, yaw)
  - Linear velocity (vx, vy, vz)
  - Angular velocity (ωx, ωy, ωz)
- 6 input vector components:
  - Commands for 6 thrusters

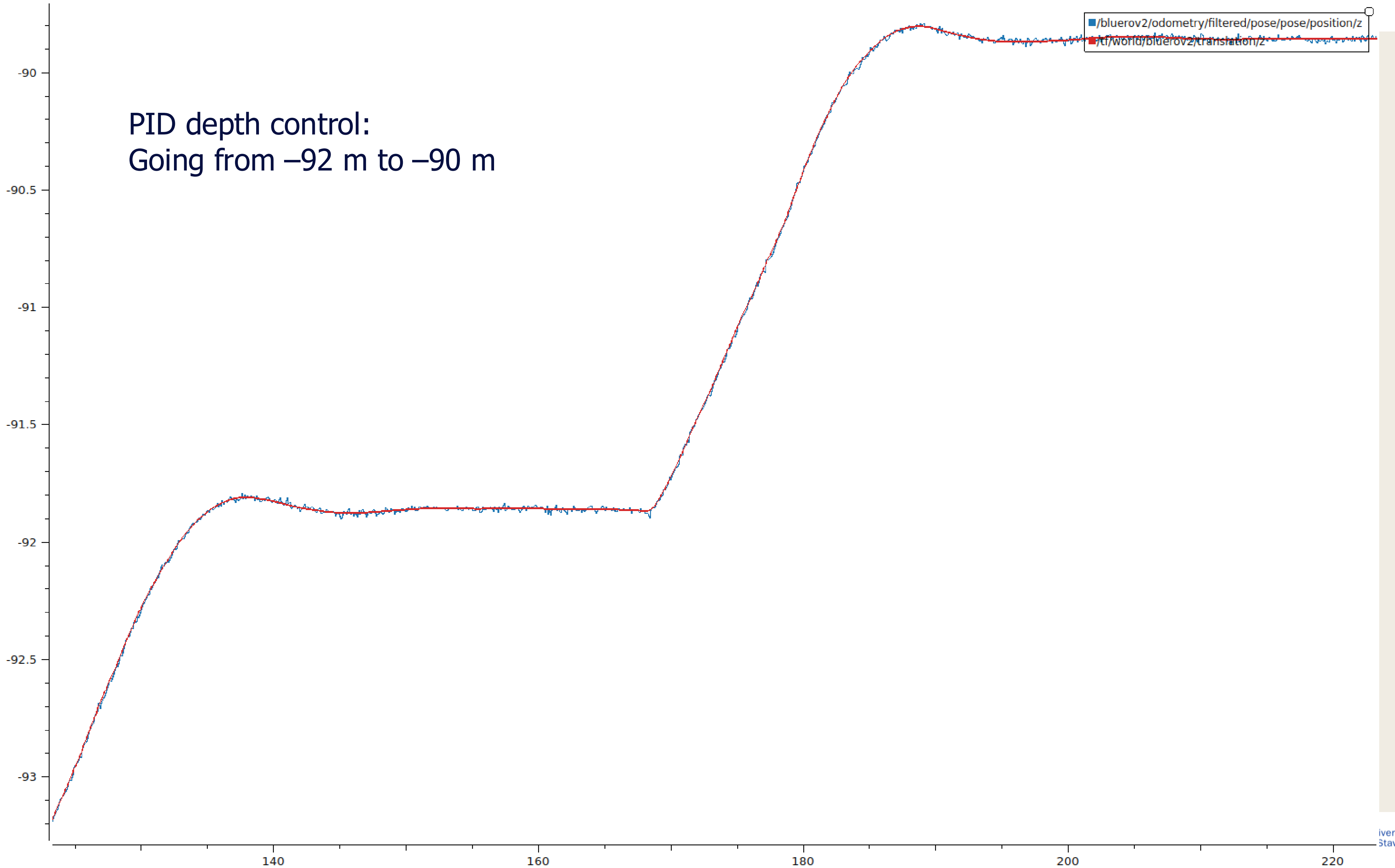
Positive thrust for T5 and T6 is pointing upwards, i.e., pushes the ROV downwards



Z-axis pointing out of the paper



PID depth control:
Going from –92 m to –90 m



Planned updates for MARSIM v1.2

- marsim_gazebo
  - Add EKF with imu, dvl and depth sensors for models/sst
- marsim_control
  - Add displacement request and position hold for bluerov2 and sst

Thank you for your attention!