TG1 OSP User Guide

The Open Simulation Platform (OSP), is an open-source industry initiative for co-simulation of maritime equipment, systems and entire ships. Building on the Functional Mockup Interface (FMI) standard, the OSP can build simulations with FMUs exported from different simulation tools.

This guide provides information on how to connect existing OSP simulations to Shipware. This guidance will cover the following topics: Requirements and General Information, Structure of the OSP Simulation, and Interfaces between OSP_bridge and Shipware.

1. Requirements and General Information

(1) System Requirements

The updated Shipware is developed on Ubuntu 20.04 and ROS noetic. For the installation guide of both Shipware and the OSP, please refer to the GitHub repository: https://github.com/Autonoship/Autonoship_simulation

The FMUs can be built with software that supports FMU export, such as MATLAB Simulink, OpenModelica, etc. **Note that if you are using MATLAB Simulink to produce the FMUs, please work on a Linux system so that the exported FMUs can be executed on Linux systems**. If an FMU provides a "*.so" file, it can be run on a Linux system. If you are using MATLAB Simulink, a version later than R2020a is recommended. For details on constructing your own FMUs with MATLAB, please refer to our guide for FMI modules: https://github.com/Autonoship/Autonoship_simulation/blob/master/FMU_module_guidance.pdf.

The OSP alone can run simulations with Co-Simulation FMUs. An independent demo using the three FMUs for OSP-only simulation can be found here: https://github.com/Autonoship/Autonoship_simulation/tree/master/osp_ros_demo/osp_simulation_model_TG1. In Shipware, we have provided a ROS node called "OSP_bridge", which helps to connect the OSP simulation to the Gazebo environment. For a detailed introduction to OSP simulation and ROS-OSP connection, please look at the "OSP_module_guidance.pdf".

(2) Introduction to OSP

The OSP aims to create a maritime industry ecosystem for co-simulation of "black-box" simulation models and "plug and play" configuration of systems. OSP relies on the FMI standard and the new OSP interface specification (OSP-IS: https://opensimulationplatform.com/specification/) for the simulation models interfaces.

OSP provide a C++/C library called libcosim/libcosimc to construct simulations with FMUs. They also provide a command line interface (CLI: https://open-simulation-platform.github.io/cosim) built with libcosim to run simulations defined with OSP-IS.

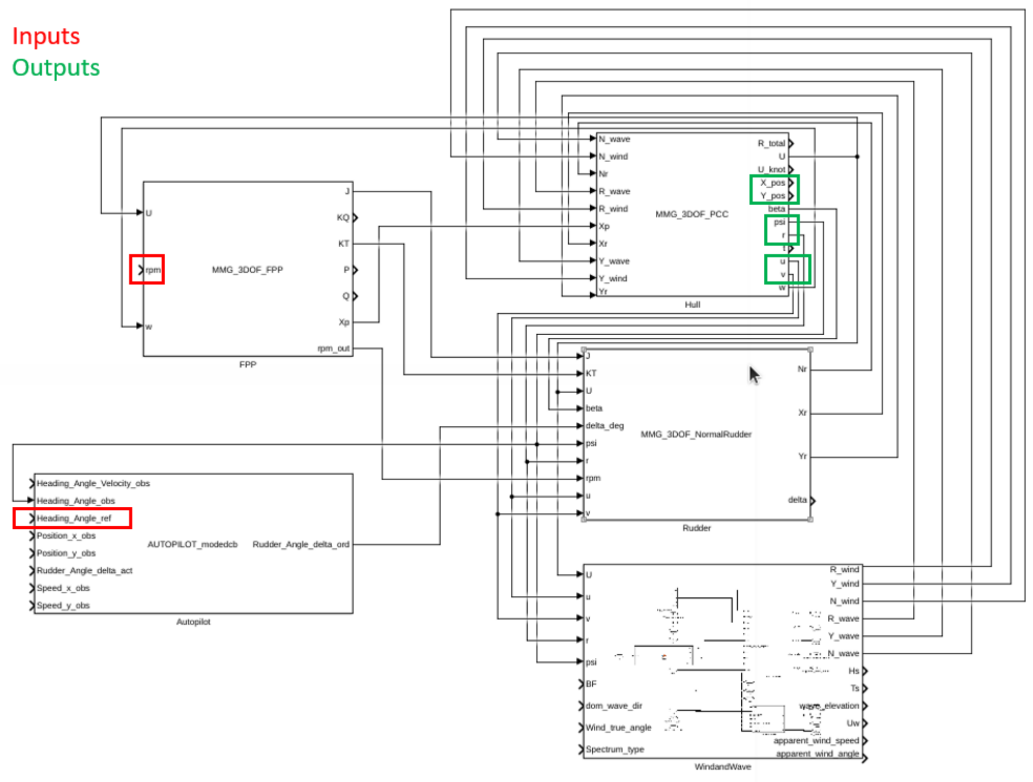
By defining the connection among FMUs with an OspSystemStructure file, one can run the simulation with the CLI. Additional features include testing scenario and logging definition. By providing the required configuration files to OSP, OSP will run your testing scenarios and log the outputs to the target directory.

2. Structure of the OSP Simulation

The sample simulation contains five FMUs: MMG_3DOF_PCC.fmu, MMG_3DOF_NormalRudder.fmu, MMG_3DOF_FPP.fmu, WindandWave.fmu and AutoPilot.fmu. The FMUs and their description files are in the folder *"/osp_ros_demo/osp_simulation_model_TG1/fmu/"*.

The connection of the FMUs are described in "/osp_ros_demo/osp_simulation_model_TG1/OspSystemStructure.xml"

The demo simulation accepts two inputs: target course and propeller rpm. The structure is shown as follows:



This figure is created in MATLAB Simulink and cannot be accessed in Shipware

3. Interfaces between OSP_bridge and Shipware

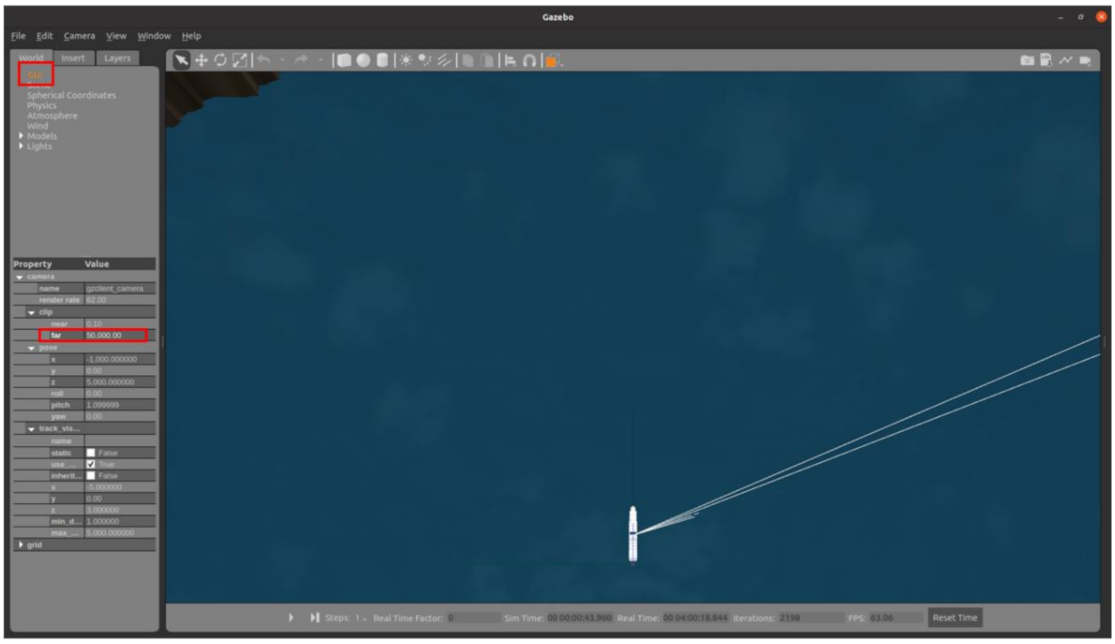
(1) Introduction to OSP-bridge

The OSP-bridge is a ROS node that constructs an OSP simulation and provides a service for other nodes to interact with the OSP simulation. For details, please look at "OSP_module_guidance.pdf".

(2) Connection between Shipware and OSP
To run the demonstration simulation, run in the command line:

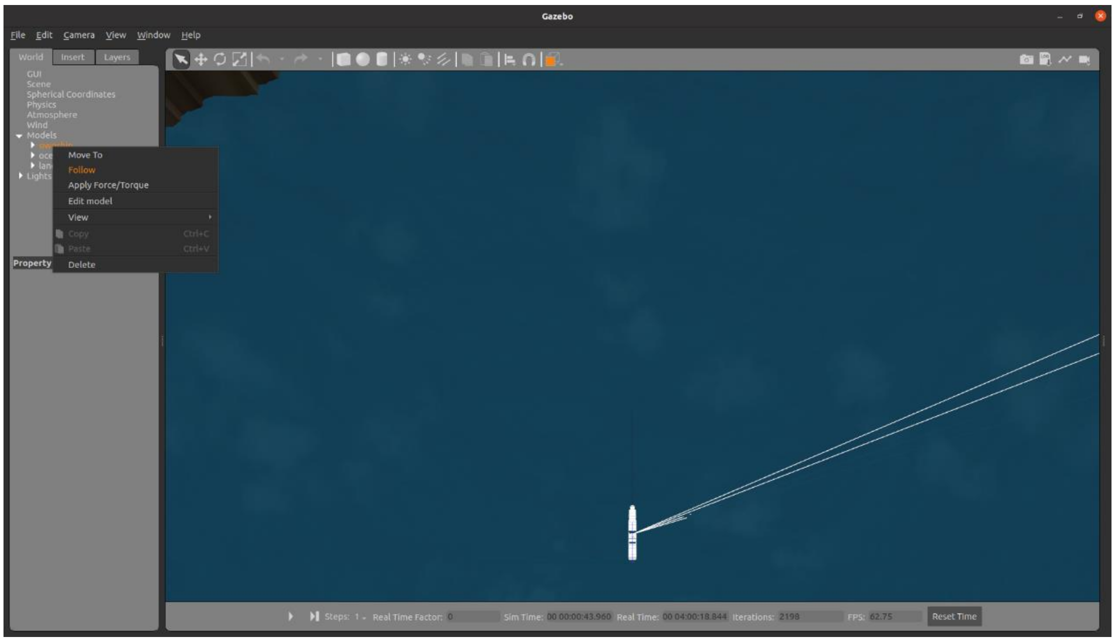
roslaunch osp_ros_demo osp_autonoship_gazebo.launch scenario:=scenario18 spawn_land:=true

This command will run the simulation with the OSP model discussed above and the path planning module activated. After the simulation has started, you need to set the range of the camera view. Left click the "GUI" on the left panel and set the "clip->far" to 50,000 to do so.

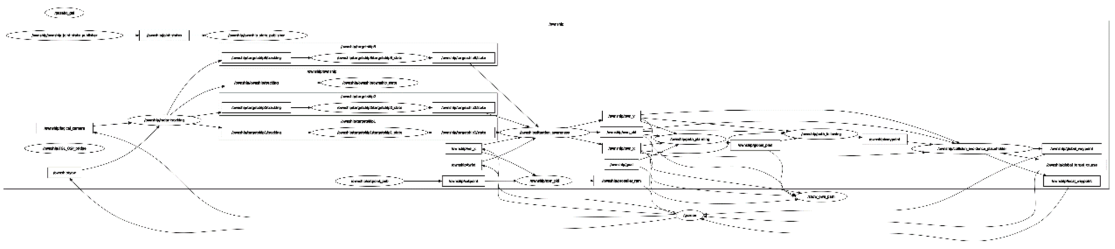


Property	Value
camera	
name	gzclient_camera
render rate	62.00
clip	
near	0.10
far	50,000.00
pose	
x	-1,000.000000
y	0.00
z	5,000.000000
roll	0.00
pitch	1.099999
yaw	0.00
track_vis...	
name	
static	False
use_...	✓ True
inherit...	False
x	-5.000000
y	0.00
z	3.000000
min_d...	1.000000
max_...	5,000.000000
grid	

To let the camera follow the ownship, click the "Models" on the left panel, right click on the "ownship" and select "Follow".



The /gazebo node will subscribe to the propeller rpm and target course information from other Shipware modules and call the service in the OSP_bridge to run the simulation. The results of the simulation is returned to /gazebo and the pose of the ownship is then updated in the simulation.
The following figure shows the ROS graph of the demo simulation.



You can also find this figure during simulation by the command:

rqt_graph

(3) Use your own OSP simulation

To use your own controller and dynamics model, you can build your own OSP simulation and connect it to the Shipware. The Shipware expects a model that contains:

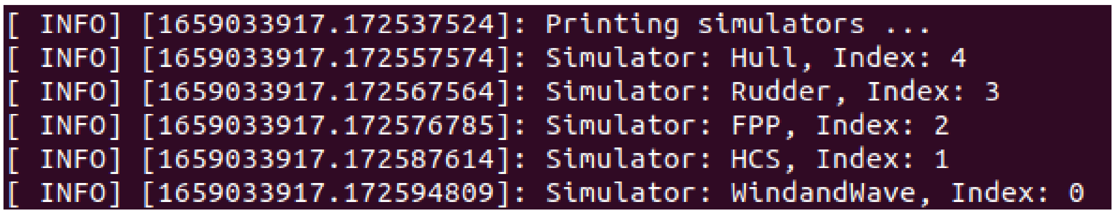
- A ship dynamics model that contains in the outputs:
  - x: x position
  - y: y position
  - psi: heading angle
  - r: angular speed

    - u: surge speed
    - v: sway speed
- An autopilot model that contains in the inputs either:
    - target course angle, or
    - target waypoint (xg, yg) in the global frame, or
    - target waypoint (xl, yl) in the local frame of the ship
- A model that takes the propeller RPM as the input:

To connect your own model, you can replace the content in the *"/osp_ros_demo/osp_simulation_model_TG1/"* with your own OSP simulation. After that, run the command:

roslaunch osp_ros_demo print_sim_id.launch

and find the simulator ID printed on the screen.



```
[ INFO] [1659033917.172537524]: Printing simulators ...
[ INFO] [1659033917.172557574]: Simulator: Hull, Index: 4
[ INFO] [1659033917.172567564]: Simulator: Rudder, Index: 3
[ INFO] [1659033917.172576785]: Simulator: FPP, Index: 2
[ INFO] [1659033917.172587614]: Simulator: HCS, Index: 1
[ INFO] [1659033917.172594809]: Simulator: WindandWave, Index: 0
```

Then, modify the simulator IDs and variables' reference IDs in the launch file *"/osp_ros_demo/launch/TG1_spawn_ship.launch".* There parameters are specified in the node "*TG1_OSP_bridge*":

```
<node name="TG1_OSP_bridge" pkg="osp_ros_demo" type="osp_ros_TG1" >
  <param name="ship_dynamics_id" value="4" />
  <param name="X_pos_id" value="27" />
  <param name="Y_pos_id" value="35" />
  <param name="psi_id" value="44" />
  <param name="r_id" value="45" />
  <param name="u_id" value="47" />
  <param name="v_id" value="48" />

  <param name="propeller_id" value="2" />
  <param name="rpm_id" value="7" />

  <param name="autopilot_id" value="1" />
  <param name="target_course_id" value="6" />
  <param name="local_waypoint_x_id" value="-1" />
  <param name="local_waypoint_y_id" value="-1" />
  <param name="global_waypoint_x_id" value="-1" />
  <param name="global_waypoint_y_id" value="-1" />
  <param name="k_id" value="23" />
  <param name="D_id" value="22" />
  <param name="w_id" value="24" />
  <param name="k" value="10.0" />
  <param name="D" value="1.0" />
  <param name="w" value="0.1" />
</node>
```

The “ship_dynamics_id” is the ID number of the FMU that contains (x, y, psi, r, u, v) in the outputs we printed in the previous step. In this case, the ID number is 4.

The “propeller_id” is the ID number of the FMU that contains (rpm) in the inputs we printed in the previous step. In this case, the ID number is 2.

The “autopilot_id” is the ID number of the FMU that contains either (target_course) or (x_waypoint_global, y_waypoint_global) or (x_waypoint_local, y_waypoint_local) in the outputs we printed in the previous step. In this case, the ID number is 1.

The IDs of variables can be found in the model description file of the FMUs. For example, you can find that the valueReference of “X_pos” in MMG_3DOF_PCC.fmu is 27. Thus, the value of “X_pos_id” is set to 27 in the launch file.

```
<ScalarVariable
  name="X_pos"
  valueReference="27"
  variability="continuous"
  causality="output"
  initial="exact">
  <Real start="0.0" unit="m"/>
</ScalarVariable>
```

The autopilot can take three different types of inputs. To activate the target_course, you need to set the id of local_waypoint and global_waypoint to -1. If you hope to use local_waypoint as input, you need to set the id of target_course_id and global_waypoint_id to -1, and set local_waypoint_id to the values you found in the FMU model description file.

In this case, the autopilot uses a second-order controller and contains (k, D, w) as parameters. These parameters are set at the beginning of the OSP simulation by the function *update_secondOrder(k, D, w)*. If your model has other parameters, you can pass them into the simulator constructor by the same process by calling "*nh.getParam("name", variable)*" and "*update_secondOrder(k, D, w)*". After you modified the "*osp_ros_TG1.cpp*", run "catkin_make" at the directory "/home/<user_name>/autonoship/" to recompile.

To run the demonstration simulation, you can run the same command in the command line:

roslaunch osp_ros_demo osp_autonoship_gazebo.launch scenario:=scenario18 spawn_land:=true