

Autonomous Robotic Inspection and Maintenance on Ship Hulls and Storage Tanks

Deliverable report – D1.3


Context		
Deliverable title	Simulation Environment and Middleware Integration	
Lead beneficiary	CNRS	
Author(s)	Cédric PRADALIER (CNRS)	
Work Package	WP1	
Deliverable due date	March 2020	
Document status		
Version No.	1	
Type	REPORT	
Dissemination level	PUBLIC	
Last modified	03 May 2020	
Status	RELEASED	
Date approved	04 May 2020	
Approved by Coordinator	Prof. Cédric Pradalier (CNRS)	Signature: 
Declaration	Any work or result described therein is genuinely a result of the BugWright2 project. Any other source will be properly referenced where and when relevant.	





Table of Contents

List of figures.....	1
Referenced documents.....	1
History of changes	2
Abbreviations.....	2
1. Abstract	3
2. Middleware	3
1. Existing frameworks	3
2. Existing expertise.....	4
3. Choice.....	4
3. Simulation environment.....	4
1. Scene design.....	4
2. Drone simulation.....	6
3. Crawler simulation	8
4. ROV simulation.....	9
5. Joint simulations.....	11
6. Dissemination within the consortium	11

LIST OF FIGURES

Figure 1: Two ship models	5
Figure 2: Flow chart of the task_manager_uav framework	7
Figure 3: Three drones managed by task_manager_uav	8
Figure 4: Simplified Altiscan's 3D model provided by Roboplanet	8
Figure 5: Close-up view of an Altiscan on ship hull in Gazebo	9
Figure 6: Three crawlers on the same hull in Gazebo.....	9
Figure 7: 3D model of Blueye's Pioneer	10
Figure 8: Close-up on a Pioneer in the SecondShip3P.world	11
Figure 9: Three Pioneers in the SecondShip.world	11
Figure 10: Simulation environment with 3 crawlers, 3 Pioneers, and 3 drones	11

REFERENCED DOCUMENTS

/



HISTORY OF CHANGES

Date	Written by	Description of change	Approver	Version No.

ABBREVIATIONS

ASV	<i>Autonomous Surface Vehicle</i>
ROV	<i>Remotely Operated Vehicle</i>
AUV	<i>Autonomous Underwater Vehicle</i>
UAV	<i>Unmanned Air Vehicle</i>
QoS	<i>Quality of Service</i>
IMU	<i>Inertial Measurement Unit</i>
EKF	<i>Extended Kalman Filter</i>



1. Abstract

This deliverable 1.3 describes the design choices regarding the middleware used for integration in the BugWright2 project. It also reports on the design of the simulation environment as of the third month of the project.

2. Middleware

1. Existing frameworks

Within the Robotic community, we will only consider three middlewares: ROS, ROS2 and DUNE. Other frameworks such as DDS, YARP, GENOM, DDX, etc. are either obsolete or much less prominent. Furthermore, we focus only on the middlewares for which there is a significant expertise within the consortium. The list below summarise the main information about the three considered middlewares.

1. ROS (www.ros.org): ROS is the de-facto standard in the robotic community. It includes a communication middleware, a set of standardized data types, a packaging system and an outstanding set of tools going from visualization to debugging, or from localization to artificial intelligence. It is used on many robotic systems around the world, from drones to ground robots and even underwater systems. The main drawback of ROS is its somewhat large footprint in terms of data storage and its sub-optimal transport layer which is designed for high-bandwidth communication channels and permanently connected subsystems.
2. ROS2 (<https://index.ros.org/doc/ros2/>): ROS2 is the evolution of ROS (also known as ROS1). It intends to take advantage of the good things in ROS1 while improving what needs to be improved. Although there is a strong interest in ROS2 from the industrial community, ROS2 has still to convince the masses of roboticist around the world. If needed transitioning from ROS1 to ROS2 requires a certain effort but the path from one to the other is fairly well defined. The main advantage of ROS2 is a much improved transport layer relying on the industrial standard DDS, allowing in particular a better control of Quality of Service (QoS) requirements.
3. DUNE (<https://lsts.fe.up.pt/toolchain/dune>): DUNE is a middleware developed by the LSTS team at the University of Porto. DUNE is the on-board software running on the vehicle, which is responsible not only for every interaction with sensors, payload and actuators, but also for communications, navigation, control, maneuvering, plan execution and vehicle supervision. It is CPU architecture independent as well as operating system independent. Thanks to its modularity and versatility, DUNE runs on LSTS ASVs, ROVs, AUVs and UAVs, but also in their Manta communication gateways. Its role is similar to ROS but with a much leaner communication protocol, well adapted to underwater communications. Its main drawback is the availability of a smaller software basis from contributor around the world. A strong advantage of DUNE is its link with NEPTUS, the multi-robot mission management developed by the LSTS team as well.



2. Existing expertise

This section matches the expertise of the various robotic experts in the consortium with the middleware listed above.

- **ROS:** ROS is used daily for research and teaching at CNRS, UIB, UNI-KLU, LSL, INSA, NTNU. It is the main middleware running on Blueeye's pioneers. Although this is not their main research tool, UPORTO and RWTH have some proficiency with the framework, UPORTO having developed bridges between DUNE and ROS in the past.
- **ROS2:** Although most of the partners listed above have considered using transitioning to ROS2, only UNI-KLU is currently using it on a daily basis for decentralized multi-agent synchronization and coordination on ground robots.
- **DUNE:** As its main developer and maintainer, the LSTS team at UPORTO has obviously a very large expertise in DUNE. This has been transmitted in part to the NTNU team in the context of joint projects on underwater robotics.

Other partners dealing with robotic development in the consortium use either no middleware or their own framework which is not publicly available.

3. Choice

Based on the situation described above, the consortium decided to select ROS as the main middleware for the project. Where necessary, in particular for the autonomous navigation systems for Blueeye's ROV, existing DUNE components will be integrated and linked with the other systems through existing DUNE-ROS bridges.

3. Simulation environment

The purpose of a simulation environment will be to support the development of individual components of the BugWright2 system, in particular for partners without regular access to the robotic systems and the testing infrastructure. This is particularly important for the development of the multi-robot system and the visualization framework.

Among the robotic simulation frameworks available, one can cite Gazebo, CoppeliaSim (previously known as V-Rep), Unreal Engine, Moose. Because of the good integration with ROS and its large availability on linux system, Gazebo (version 9) was selected as the main simulation environment.

1. Scene design

The most important element of the simulation scenes will be the ship structure. For Gazebo, a Collada file (.dae) is required. For computational geometry (e.g. CGAL), OFF files are required. Most free models available on the web are delivered in OBJ, STL, 3DS formats. Meshlab provides tools to switch between one representation and the others.

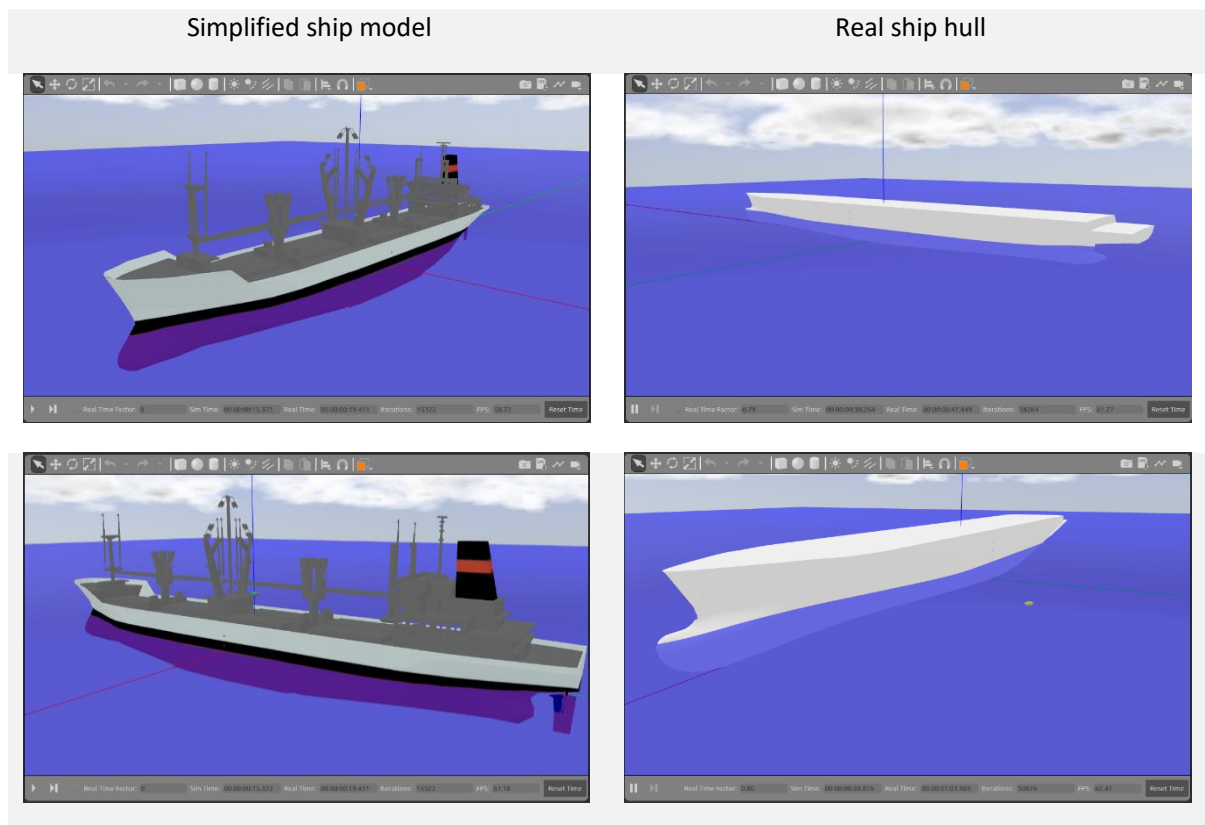
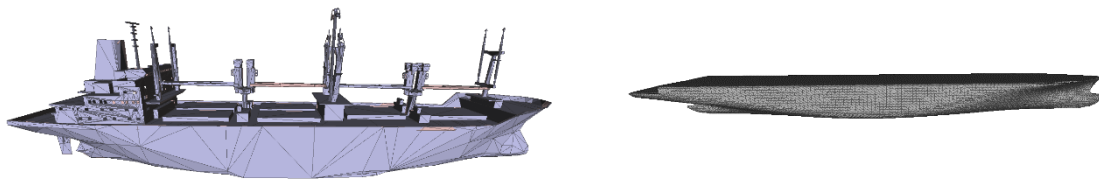
When considering meshes, the distinction between collision check and display needs to be considered. The model used for display may be highly complex, including texture and colours. The model used for collision check needs to have a minimal amount of triangles while respecting the general topology of the ship. Given the context of our project, the collision model only needs to include the hull. Meshlab also



provides tools to pre-process a complete model and reduce its complexity. This process is tedious but necessary for the real-time performance of the simulator.

The mesh pre-processing needs also to include the preparation for use by CGAL. CGAL requires a very clean mesh with well oriented triangles, no degenerated triangles, no non-manifold edges. The preparation of such a mesh with Meshlab is also a significant effort.

At the time of this writing, two ship models have been made available to the consortium. The first one is a generic model downloaded from internet. The second one has been pre-processed from data provided by Danaos.



Gazebo renderings

Figure 2: two ship models

Although the ship meshes presented above are sufficient to support the first project development, they do not contain a sufficient level of details to implement complex navigation algorithms, in particular for the crawlers. These ship hulls are extremely smooth, without positive or negative obstacles. Future iterations of the simulation environment will focus on expanding these designs and integrating complex ship 3D models in the environment, using anonymised files provided by the project end-users.



2. Drone simulation

The drone simulation `task_manager_uav` is, like the crawler and ROV simulation, based on the `ros_task_manager`¹ by CNRS (Cédric Pradalier) for the mission execution and (local) task planning. The modelling of the drone is based on the RotorS² simulation framework, which can easily be integrated in the proposed Gazebo simulation environment for BugWright2. RotorS already includes models for the former multi-copter systems by Ascending Technologies such as the Hummingbird, Firefly, and Pelican. Additional models can be generated based on these examples. Similarly, different sensor modalities such as Inertial Measurement Unit (IMU), cameras, position sensors are provided and ready to use on the platform models. The provision of these standard elements, simple integration in Gazebo, ease of adaptation and exchange of different modules (controller, estimator, models), and being well tested already in the community with a decent documentation made RotorS a good choice as base-frame for the drone simulation part.

As low level control, a standard attitude controller acting on the speeds of the single motors is implemented. Thus, the multi-rotor drone models have to include mass inertia information and other dynamic aspects besides their geometry. For position control, the geometric controller presented in (Lee et al 2013³) is used. For its input, either ground truth simulation-trajectory values or pose estimates generated by UNI-KLU's state estimation framework and the simulated sensor inputs can be used.

For state estimation, UNI-KLU implemented `aaucns_ssf` as a simple Extended Kalman Filter EKF based GNSS-IMU fusion with realistic noise in the framework based on (Weiss et al. 2013⁴). The IMU is considered as propagation information whereas GPS is used as correction. This proof-of-concept implementation forms the base for future, more complex estimation processes based on the same framework. The output `/[sensor_type]/odom` is fed into the controller part and the trajectory planner, whereas `[sensor_type]` can be any (combination of) future sensor modalities fused with IMU as long as the `./odom` message output format is preserved.

¹<https://hal.archives-ouvertes.fr/hal-01435823>

²https://github.com/ethz-asl/rotors_simulator/wiki

³<https://arxiv.org/pdf/1109.4457.pdf>

⁴<http://e-collection.library.ethz.ch/eserv/eth:5889/eth-5889-02.pdf>



For navigation, a dynamically feasible trajectory based on the multi-copter performance parameters (maximum angular velocities, maximum linear acceleration, etc.) is generated as a piecewise polynomial path based on the `mav_trajectory_generation` framework⁵ using the approaches in (Richter et al. 2016⁶) and (Burri et al. 2015⁷). Besides the ability to fly such a predefined path, the implemented drone simulation allows online waypoint adjustment with subsequent trajectory re-planning. For this, a single waypoint in the existing list can be selected via its ID and its changed value can be pushed to the task server. The replaced waypoint insertion triggers a re-calculation of the trajectory. In this way, local obstacle avoidance can be tackled focusing on individual waypoints without changing all other mission

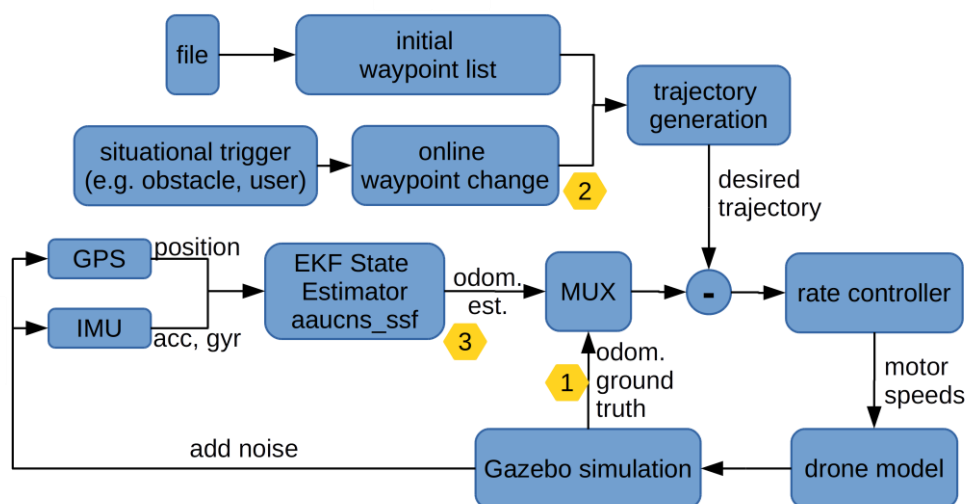


Figure 2: Flow chart of the `task_manager_uav` framework. Demo scripts in the repository show its use with ground truth data and fix trajectory (1), with online adapted trajectory (2), and with realistic state estimation instead of ground truth (3).

waypoints. The computed desired trajectory is then compared with the current location of the drone to generate corresponding control inputs. The source of information for the current location can be chosen between the ground truth (from Gazebo simulation) and a realistic signal coming from the `aaucns_ssf` state estimator.

The above flow-chart depicts the interplay of the above mentioned elements. In the BugWright2 GIT repository, the `task_manager_uav` is the core package for the above framework and contains detailed instructions and demonstration scripts in particular to show-case the three elements of 1) regular operation with ground truth current drone pose from the simulation and non-adaptive input trajectory, 2) inclusion of adaptive waypoint change and online trajectory recalculation, and 3) replacing ground-truth with an estimated current drone pose using the `aaucns_ssf` GPS-IMU estimator. The three cases are marked in the diagram with yellow hexagons.

While easy to use, the above framework, due to the dependence on the simplified controller and model used, lacks on details in:

⁵https://github.com/ethz-asl/mav_trajectory_generation

⁶https://link.springer.com/chapter/10.1007/978-3-319-28872-7_37

⁷<https://ieeexplore.ieee.org/abstract/document/7353622>



- dynamic modelling as the current implementation assumes all frames (sensor, centre of mass, body, aerodynamic centre) to be aligned. In particular with larger sensor payload, this will not be true in practice. Adaptations to control and model/estimator may be required for improved system performance
- photo-realistic scenery. While Gazebo is easy connectible with ROS, compared to other engines (e.g. Unity3D, Unreal), it lacks on realistic texture. Further adaptations may be required to the framework once camera sensors are integrated.

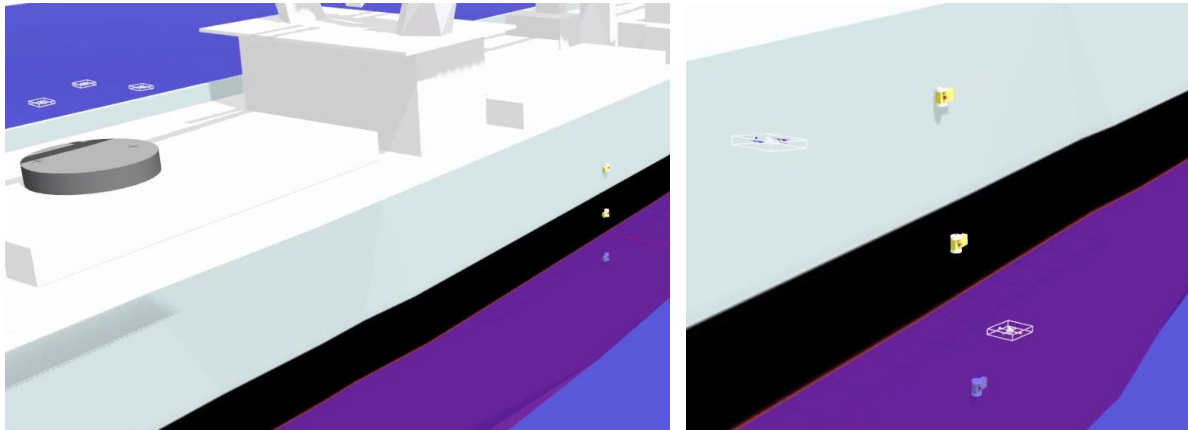


Figure 3: Three drones managed by `task_manager_uav`.
Left: just after take-off. Right: following dynamically feasible trajectories

3. Crawler simulation

Physically, Roboplanet's Altiscan can be seen as a differential drive robot, with two magnetic traction wheels and a magnetic castor wheel on the back for stability. Gazebo does not include an easy way to simulate the magnetic attraction but this can be simulated by adding a constant force pointing downward in the Altiscan frame. This force needs to be strong enough to create enough wheel traction when the robot is on the hull surface, despite the poor friction simulation in Gazebo.

Graphically, a model provided by Roboplanet and depicted below is attached to the physical model.



Figure 4: Simplified Altiscan's 3D model provided by Roboplanet

The Gazebo simulation is currently implemented in the `gz_crawler_plugin` within the `bugwright_ws` git repository. The plugin is mostly responsible for generating the constant downward force. At the time of



this writing the crawler plugin includes the following sensors exported as ROS topic prefixed by the crawler name defined in the world file:

- RGB camera: `/crawler_name/image_raw`, `/crawler_name/camera_info`.
- IMU: `/crawler_name/imu` (sensor_msgs::Imu)
- Global position: `/crawler_name/uwb` (geometry_msgs::PointStamped)

In addition, each crawler can receive a geometry_msgs::Twist on the following topic:

- Twist in body frame: `/crawler_name/cmd_vel`

Two Gazebo worlds are currently available in the gazebo_world/world package:

- SecondShip3C.world: uses the SecondShip.dae mesh and instantiates 3 crawlers
- RealShip3C.world: uses the real_hull.dae mesh provided by Danaos and instantiates 3 crawlers

Before starting this world without a launch file, one needs to source `gz_crawler_plugin/exports.sh` to define the proper environment variables for Gazebo to find the meshes, models and plugins.

Alternatively, the gazebo_world package provides two launch files to start Gazebo with the appropriate world files and environment:

- SecondShip3C.launch
- RealShip3C.launch

The images below demonstrate the simulated crawlers on the SecondShip3C world.

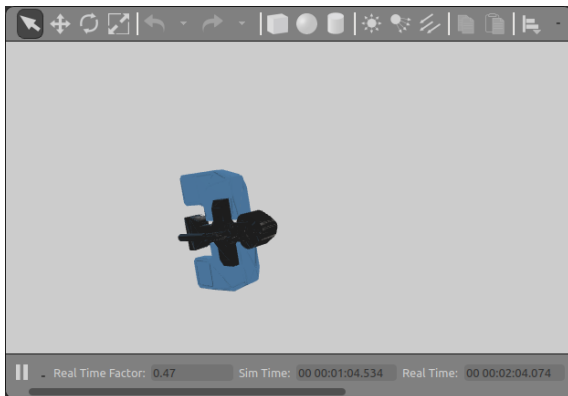


Figure 5: Close-up view of an Altiscan on ship hull in Gazebo

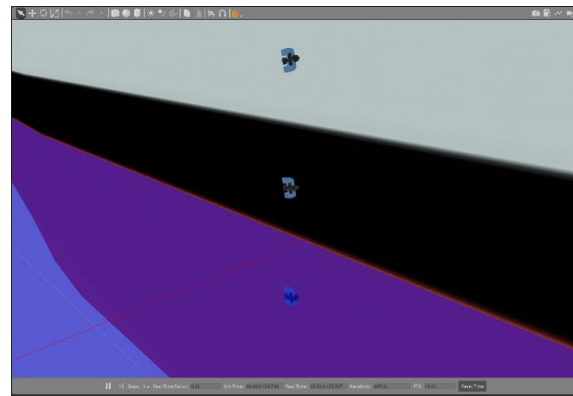


Figure 6: Three crawlers on the same hull in Gazebo

4. ROV simulation

The Blueye Pioneer simulation is physically considered as a floating solid for Gazebo. As with a real system, the challenge is to define a centre of buoyancy slightly above the centre of gravity so as to achieve a natural roll and pitch stability. Additionally, the Pioneer is slightly positively buoyant, which means that its buoyancy force is slightly higher than its weight as long as it is submerged. Buoyancy decreases proportionally to the part of the body above water.



To the contrary of the Altiscan, the Pioneer cannot be controlled in velocity. Instead, the model account for an input wrench including vertical, longitudinal and lateral forces in the body frame, as well as torque around the vertical axis.

The graphical model of the Pioneer has been provided by Blueye and is displayed below. A lower resolution version produced by Meshlab is used as a collision model for the simulation.



Figure 7: 3D model of Blueye's Pioneer

The Gazebo simulation is currently implemented in the `gz_blueye_plugin` within the `bugwright_ws` GIT repository. The plugin is mostly responsible for generating the buoyancy force, applying a water friction force and applying the input wrench. At the time of this writing the crawler plugin includes the following sensors exported as ROS topic prefixed by the crawler name defined in the world file:

- RGB camera: `/pioneer_name/image_raw`, `/crawler_name/camera_info`.
- IMU: `/pioneer_name/imu` (`sensor_msgs::Imu`)
- Global position: `/pioneer_name/usbl` (`geometry_msgs::PointStamped`)
- Global heading (i.e. compass): `/pioneer_name/heading` (`std_msgs::Float32`)

In addition, each Pioneer can receive a `geometry_msgs::Wrench` on the following topic:

- Twist in body frame: `/pioneer_name/cmd_wrench`

Two Gazebo worlds are currently available in the `gazebo_world/world` package:

- `SecondShip3P.world`: uses the `SecondShip.dae` mesh and instantiates 3 Pioneers
- `SecondShip3C3P.world`: uses the `SecondShip.dae` mesh and instantiates 3 Pioneers and 3 Altiscans.

Before starting this world without a launch files, one needs to source `gz_pioneer_plugin/exports.sh` to define the proper environment variables for Gazebo to find the meshes, models and plugins. Note that this also sources `gz_crawler_plugin/exports.sh`.

Alternatively, the `gazebo_world` package provides two launch files to start Gazebo with the appropriate world files and environment:

- `SecondShip3C.launch`
- `SecondShip3C3P.launch`

The images below demonstrate the simulated pioneers on the `SecondShip3P` world.

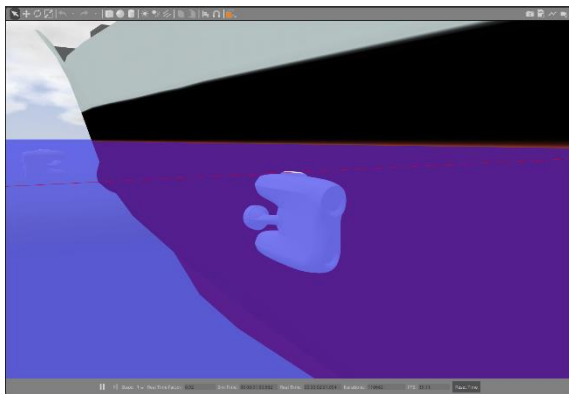


Figure 8: Close-up on a Pioneer in the SecondShip3P.world

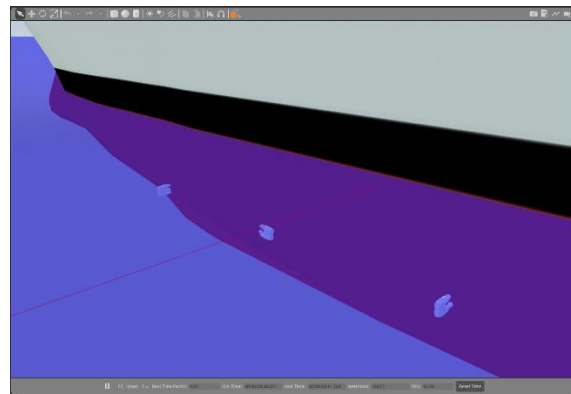


Figure 9: Three Pioneers in the SecondShip.world

5. Joint simulations

Integrating all the models above does not present any particular challenge. The gazebo_world package provides a world and launch file including the three instances of the three types of rovers.

- SecondShip3C3P3M.world describes the world with all the robot models.
- SecondShip3C3P3M.launch launches the Gazebo simulation as well as all the helper files requires to control the RotorS simulation.

The picture below depicts a situation where the nine robots are visible in the simulation. Note that, this particular simulation requires a significant computing power. A dedicated GPU is strongly recommended.

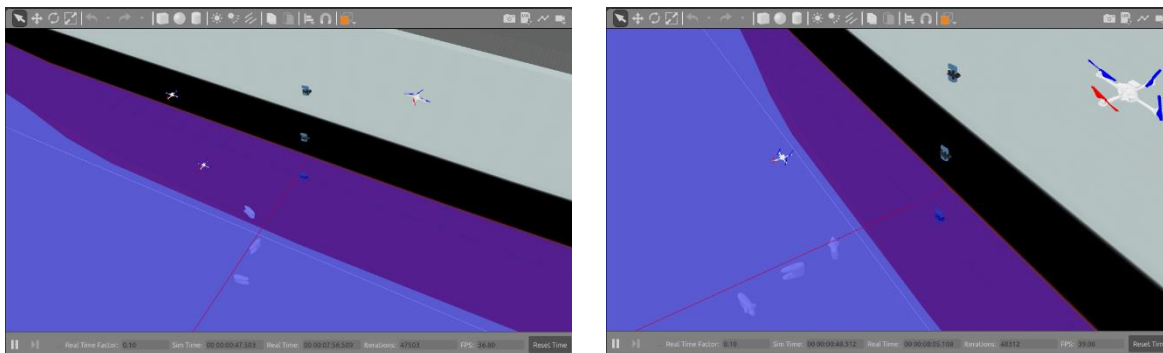


Figure 10: Simulation environment with 3 crawlers, 3 Pioneers, and 3 drones

6. Dissemination within the consortium

All the packages described in this document have been made available within the shared git repository for the consortium: <https://gitlab.georgiatech-metz.fr/bugwright2/bugwright2-ws> (restricted access).

At the time of this writing, the simulator has been deployed at CNRS, UNI-KLU, INSA, LSL, and RWTH, which are the partners most in need of a simulation environment to develop WP4, WP5, WP6 and WP7.



On the other side, the HIL (Hardware in the Loop) simulation and development framework of UIB⁸ is incorporating the features required to be compatible with the simulations described in this document. This HIL framework has been developed under the ROS environment, employs Gazebo and allows the simulation of the motion of the real platform, as well as its functionalities and behaviour at the mid- and high-control levels.

The use of ROS and Gazebo in both simulation environments –the above-described environment and the UIB framework– is to allow for compatibility regarding software development within the two environments. This can be accomplished just by agreeing on the messages being exchanged between ROS nodes, either if ROS standard messages are used or if proprietary messages are defined for certain tasks and their details are shared within the consortium. Besides, the same vessel models that have been mentioned in the previous sections can be incorporated within the UIB HIL framework, and this allows for software debugging, testing and experimentation for the agreed use-cases and environments, and for functionality development to be performed under the same conditions as the rest of the consortium.

⁸ The HIL development and simulation framework of UIB comes at the end of a number of control software developments completed within the context of UIB participation in previous projects involving MAV-based vessel inspection (FP7 MINOAS, FP7 INCASS & H2020 ROBINS). This framework has been employed in a number of field trials to perform inspections onboard real vessels of different nature: bulk carriers, ro-ro vessels, containerhips, etc.