VORC Technical Guide

2020 Virtual Ocean Robotics Challenge

1. Introduction

The purpose of this document is to provide Virtual Ocean Robotics Challenge (VORC) teams with the information necessary to successfully prepare for and participate in the VORC. It covers the following topics: the robotic platform, propulsion and sensor configuration, the general structure of the competition tasks, runs and environmental envelopes, the application programming interface (API), and instructions for submitting your code for the competition.

For a description of competition goals, scoring, phases, and the details of the individual competition tasks, please see the following related document:

- 2020 Virtual Ocean Robotics Challenge Task Descriptions

2. Robotic Platform

The VORC will be executed using the Gazebo simulation environment. All teams must use the simulated version of the Common Raft (CoRa) distributed with the VORC software. The boat, as supplied by the VORC software, is standard for all teams. No modification of the standard platform model (URDF description, etc.) is allowed during competition.

The simulated boat includes models of the rigid body dynamics, hydrodynamics, external forcing (waves and wind) and propulsion. The rigid body dynamics are captured via the Gazebo physics engine. The hydrodynamics, external forcing and propulsion forces are generated by VORC plugins with fixed parameters. As with the platform model itself, no modification of the standard VORC model parameters is allowed during competition.

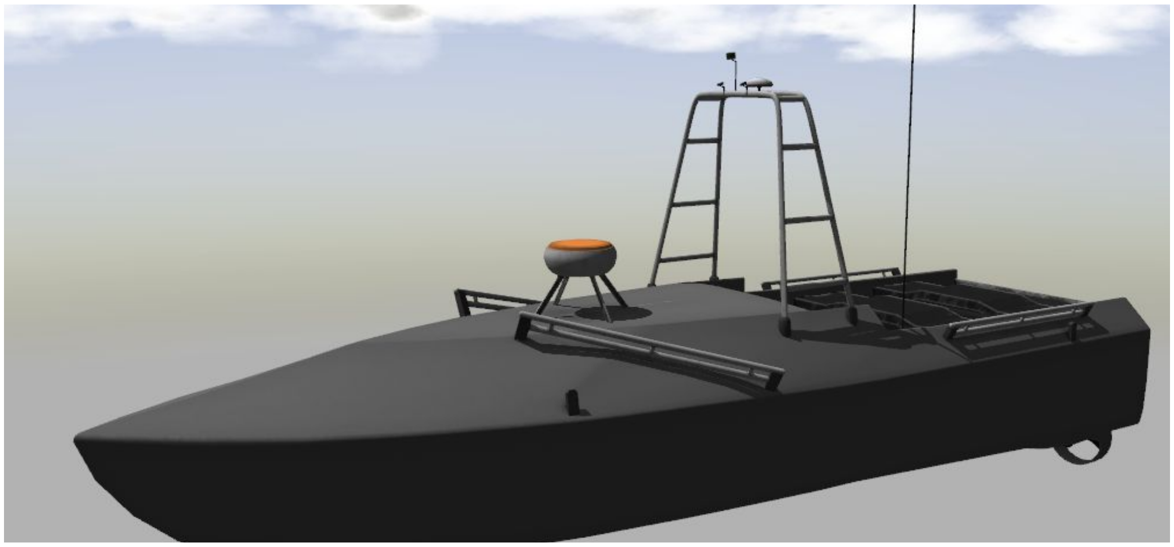

Figure 1: Simulated CoRa model

3. Propulsion configuration

The boat is equipped with two identical thrusters configured in "H" differential configuration (illustrated in **Figure 2: CoRa "H" thruster configuration**). The thrusters are fixed at the chassis of the boat. It is not possible to change their yaw angle.

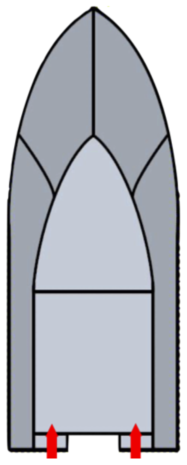

Figure 2: CoRa "H" thruster configuration

4. Sensor configuration

Much like the propulsion system, the sensor configuration is also common across all competitors, and the boat is equipped with a set of standard sensors. Standard sensors include navigation sensors (GPS and IMU) and perception sensors (cameras and lidars).

Table 1: Standard sensors included with the boat

Sensor type	Number of instances	Pose
Camera	2	(-0.61, ±0.2, 4.7, 0, 0, 0.261799)
Lidar	1	(-0.595, 0, 5, 0, 0, 0.139626)
GPS	1	(-1, 0, 4.6, 0, 0, 0)
IMU	1	(0.3, -0.2, 1.0, 0, 0, 0)
Acoustic	1	(0, 0, 0, 0, 0, 0)

The available sensor types and their performance characteristics have been chosen to reflect commonly used sensors. These characteristics do not represent any specific sensor choice but are meant to be representative of typical hardware options.

Note: While the sensor performance specifications below (update rates, noise values, etc.) are detailed, the exact values of these specifications may change before the final release of this document.

4.1. Navigation Sensor

A single standard navigation sensor, representing a GPS-aided IMU, will be used for VORC. This single sensor is simulated through the use of two Gazebo plugins (see the hector_gazebo_plugins) which generate GPS information (position and velocity) and IMU information (attitude, attitude rate and accelerations). While these two measurements are presented separately, the characteristics of the measurements are consistent with a sensor that estimates a complete navigation solution, e.g., a GPS-aided IMU.

Table 2: Characteristics of VORC Navigation Sensor

GPS-Aided IMU		
GPS	Update Rate	20 Hz
	Horizontal Position Noise[1]	0.85 m
	Vertical Position Noise	2.0 m
	Velocity Noise	0.1 m/s
IMU	Update Rate	100 Hz
	Acceleration Offset/Bias	+/- 0.002 g
	Acceleration Noise	0.275 g
	Attitude Rate Noise	0.08 degrees/s
	Heading Noise	0.8 degrees

4.2. Camera Sensor

A standard camera is simulated via the Gazebo camera plugin.

Table 3: Characteristics of VORC Camera Sensor

Camera	
Update Rate	30 Hz
Resolution	1280x720 px
Color format	R8G8B8

4.3. Acoustic Sensor

An acoustic sensor providing a RangeBearing message with the location of the beacon at a given update rate. The 3D location of the beacon is relative to the sensor, in the form of range and two angles (bearing and elevation) .

Table 4: Characteristics of VORC Acoustic Sensor

Acoustic sensor	
Update Rate	1 Hz

4.4. Lidar Sensor

One type of lidar sensor is provided for VORC: 16 beam.

Table 5: Characteristics of VORC Lidar Sensors

	16 Beam
Update Rate [Hz]	10
Lasers (Number of beams)	16
Samples (Number of horizontal rotating samples)	1875

[1] Unless otherwise noted, noise values are specified as one standard deviation and represent a Gaussian distribution.

Min Range [m]	0.9
Max Range [m]	130
Noise [m]	0.01
Min. Horizontal Angle [rad]	$-\pi$
Max. Horizontal Angle [rad]	π
Min. Vertical Angle [rad]	$-\frac{\pi}{12}$
Max. Vertical Angle [rad]	$\frac{\pi}{12}$

4.5. Task Information

Teams face multiple independent tasks during VORC. These tasks require different actions from the participant's controller code. During every task, the status of the task is published as a custom ROS Task message over a ROS topic. The message provides the following information about the status of the simulated task. Please, refer to the VORC API section of this document for further details.

Table 6: ROS Task message definition

Field Name	Description
`name`	Unique task name (e.g.: "wayfinding", "perception").
`state`	The current task state = {initial, ready, running, finished}. See the *Task States* section for more information.
`ready_time`	Simulation time at which the task transitions to the *ready* state.
`running_time`	Simulation time at which the task transitions to the *running* state.
`elapsed_time`	Elapsed time since the start of the task (since running_time).
`remaining_time`	Remaining task time.
`timed_out`	Whether the task timed out or not.
`score`	Current task score.

Teams are expected to subscribe to the task ROS topic and select their appropriate robot behavior given the current task under execution. In addition, teams need to react to the task states appropriately. The *initial* state is only used to stabilize the vehicle, allow for initial transients to decay and make sure that all the software blocks are ready. While the system is in the initial state, teams receive sensor information, but the robot control will be very limited. In the *ready* state, teams have full control of their robot and we expect them to get ready for the start of the task. Teams need to monitor the simulation time published over the clock ROS topic and compare it with the `ready_time` and `running_time` to be prepared to take control of the vehicle and start the task. Once the task is in the *finished* state, teams can still control the vehicle, but the score will not change.

5. Task States

A task can be in one of four different states. The task state is set by Gazebo according to the task configuration. The current state is included in the task message periodically published on the task information ROS topic.

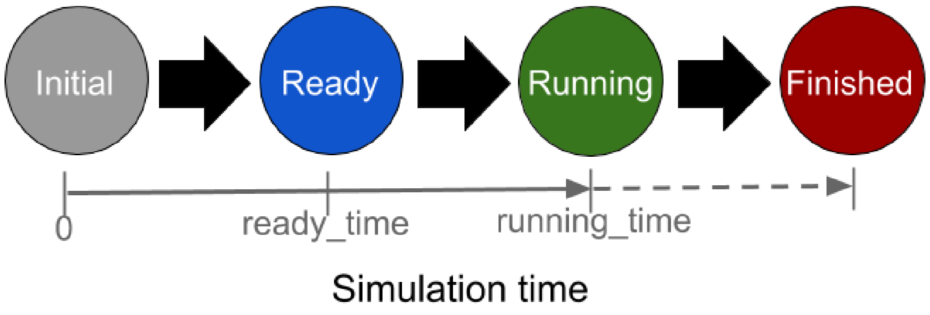

Figure 3: Task states as a function of time

5.1. Initial

After Gazebo starts, the task is in the *initial* state. The robot's motion is fixed in the X (surge), Y (sway) and yaw degrees of freedom, but allowed to move in Z (heave), pitch and roll degrees of freedom. Thus, the robot is pushed up and down by the waves and wind and will change its orientation (except in yaw) but stays in the same 2D position. The purpose of this initial state is to allow for simulation startup transients to decay and for all the user's software to have sufficient time to initialize.

5.2. Ready

The task transitions to *ready* when simulation time reaches the value ready_time. In the *ready* state, the robot motion is free in all degrees of freedom and is under the participant's full control. While in the ready state no scoring is performed.

5.3. Running

The task transitions to *running* when simulation time reaches the value running_time. In the *running* state, the task officially starts. The scoring and the task timer are enabled.

5.4. Finished

The task transitions to *finished* when the remaining_time field of the task message reaches 0 or when the task is considered complete. If all task time has been consumed, but the task has not been fully solved, the field timed_out of the task message will be set to true. The score will not be updated in this state.

6. Runs and Environmental Envelopes

In the competition, each team shall conduct multiple runs per task, where each run will use a different set of environmental conditions. Though conditions will be distinct from run to run, these distinct configurations will be identical for each team; i.e., each team will see the same set of conditions as the other teams in the competition. The following elements of the simulation will change between runs:

6.1. Object Location and Orientation

Relevant objects will be moved between runs to prevent training the team's controllers to handle known geometry across runs. This will include the placement of obstacles (for example, buoys or docks), as well as the starting pose of the robot itself.

6.2. Fog

Gazebo will optionally simulate fog with different densities and colors. The two images below illustrate the addition of fog to the visual scene.

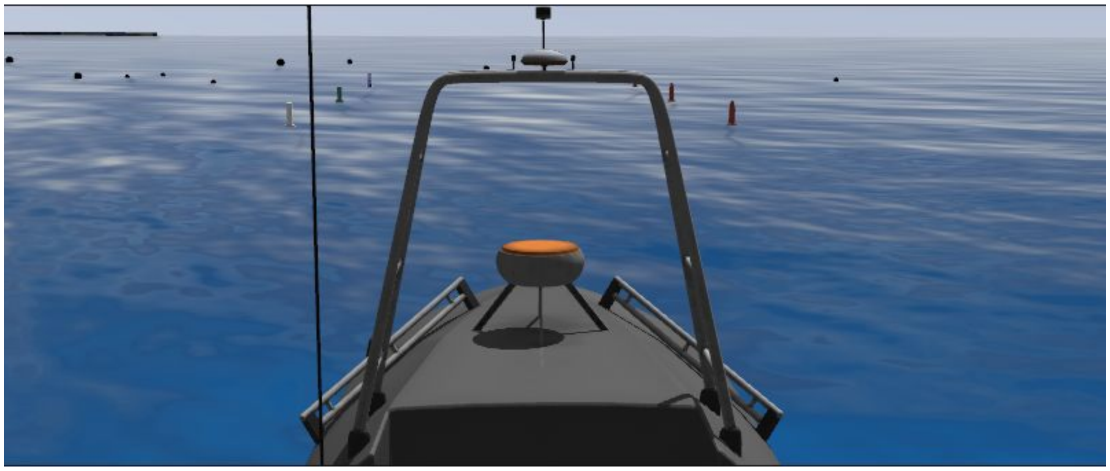

Figure 4: Visual scene with no fog

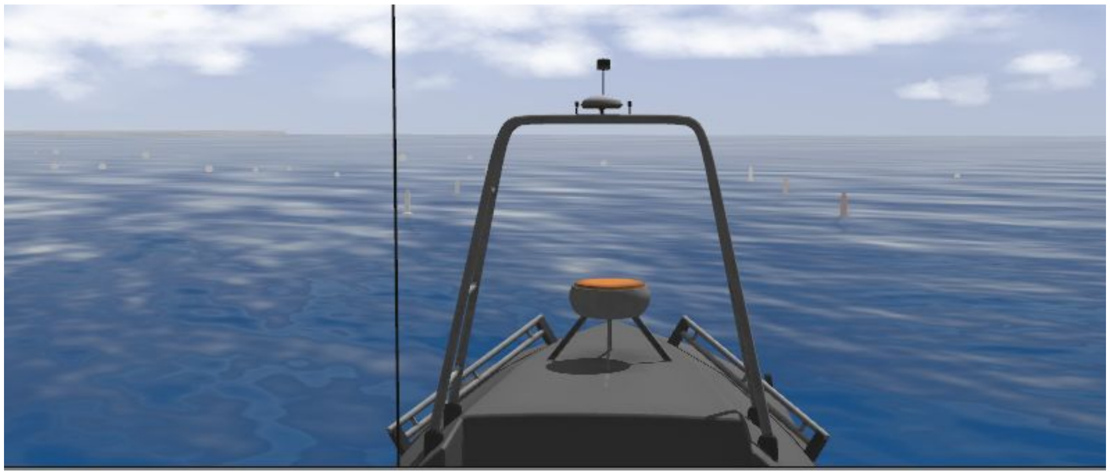

Figure 5: Visual scene with fog

6.3. Wind

Wind exerts a force on objects in the VORC environment. The total wind velocity is a combination of a constant mean velocity component and a variable wind speed (i.e., gusting). The variable component of the wind speed is modeled as a first-order linear spectrum defined by two components: the variability gain and the variability time constant. The variability gain specifies the magnitude (root-mean-square) of the variable component of the wind speed, and the variability time constant specifies how rapidly the wind speed changes with time. For details on the wind model and implementation see the VRX Documentation Wiki: Gazebo Plugins. For examples of how to change the wind parameters see the VRX Tutorials Wiki: Changing Simulation Parameters.

6.4. Waves

Surface waves affect the motion of objects in the simulated environment. The simulated sea state is generated using a summation of individual regular waves to create the three-dimensional water surface geometry as a function of time. The amplitudes of the individual waves are determined from sampling a Pierson-Moskowitz (P-M) ocean wave spectrum. The pertinent parameters associated with specifying a particular sea state are as follows:

- Peak Period (T_p) – wave period with the highest energy.
- Gain (γ) – constant multiplier applied to the individual wave amplitudes
- Wave Direction – direction of travel of the wave component corresponding to the peak period
- Wave Angle – angular difference in direction between component waves

The combination of T_p and γ parameters determine the energy in the specific sea state. For the VORC competition combinations of T_p and γ as illustrated in Figure 7 will be used for evaluation.

For details on the sea state model see the VRX Documentation Wiki: Gazebo2019 Plugins.

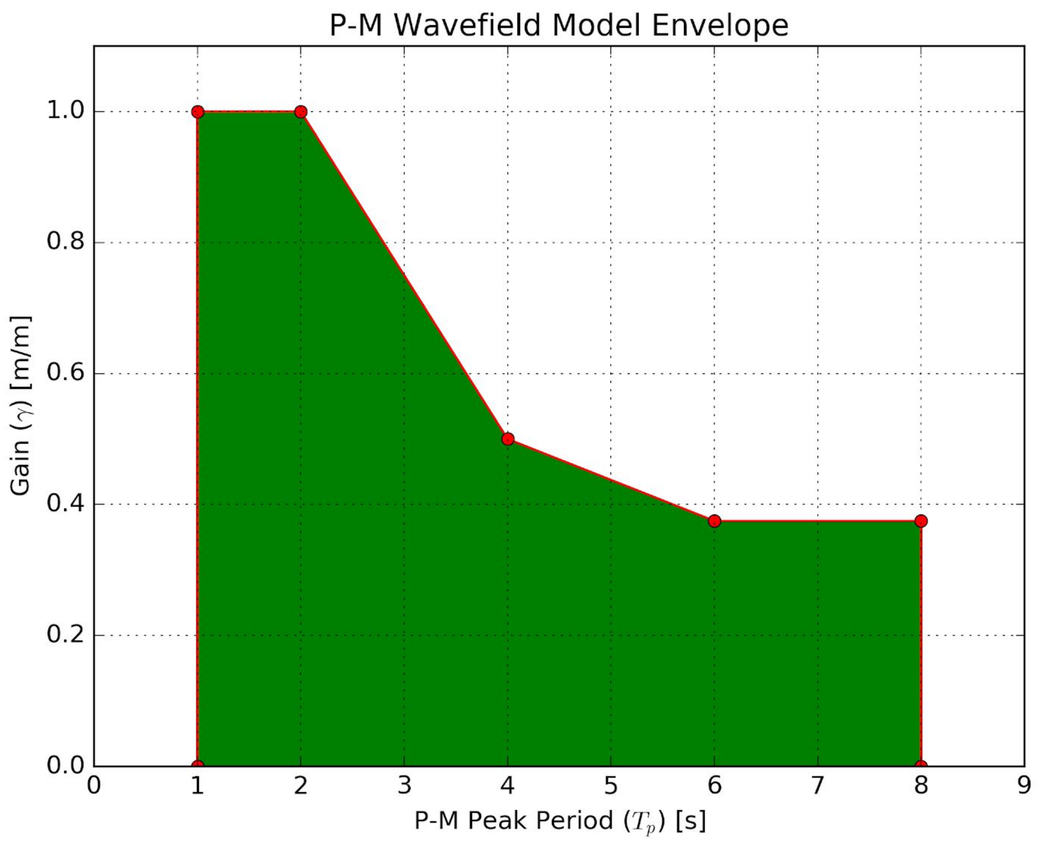

Figure 6: Envelope for sea state parameters used in VORC evaluation

6.5. Ambient Light

The color of the ambient light. The two images below illustrate changes to the ambient lighting conditions.

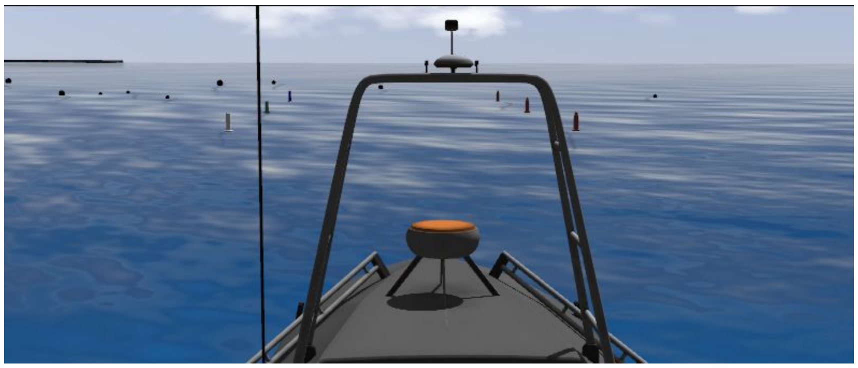

Figure 7: Visual scene with reduced ambient light

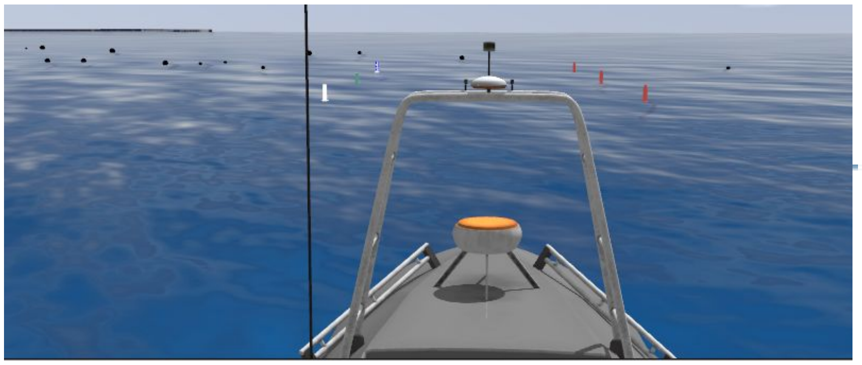

Figure 8: Visual scene with regular ambient light

The following table summarizes the range of all the parameters that can change during runs:

Table 7: Environmental variable parameters

Gazebo Parameter	Minimum value	Maximum value
`scene::fog::color`	[0.7, 0.7, 0.7, 1]	[0.9, 0.9, 0.9, 1]
`scene::fog::density`	0	0.1
`scene::ambient`	[0.3, 0.3, 0.3, 1]	[1, 1, 1, 1]
`wamv_gazebo::wind_mean_velocity`	0	60
`wamv_gazebo::wind_variance_gain`	0	40
`wamv_gazebo::wind_variance_time`	2	20
`wamv_gazebo::wind_direction`	0	360
`wamv_gazebo::wave_period`	See Figure 6.	
`wamv_gazebo::wave_gain`		
`wamv_gazebo::wave_direction`	0	360
`wamv_gazebo::wave_angle`	0	360

The characteristics of the simulated environment will be varied during competition runs. Final specifications for the exact characteristics will be released as part of the final technical specification; however, teams can expect values to be within the ranges described in the table above.

7. VORC API

VORC provides a ROS interface to the teams for controlling all available actuators, reading sensor information and sending/receiving notifications. The use of ROS as the interface between the team's software and the simulation environment does not require that the team's software internally use ROS. The intention of the competition is to be technology agnostic with regard to solution architecture and implementation. However, a single standard interface is required for the feasibility of the virtual competition. Every effort will be made to offer all teams support implementing the ROS interface to their software. For teams not familiar with ROS we highly recommend going through http://wiki.ros.org/ROS/Tutorials to get familiar with ROS and, in particular, with ROS topics and services.

The next table summarizes the ROS API used for the competition. Note that all topic names used for propulsion and sensors are configurable via their respective YAML files.

Table 8: Thruster actuation API

Thruster Actuation	
Topic Name	**Description**
`/cora/thrusters/left_thrust_cmd`	Next power command for the left thruster
`/cora/thrusters/right_thrust_cmd`	Next power command for the right thruster

Table 9: Sensor information API

Sensor Information	
Topic Name	**Description**
`/cora/sensors/cameras/front_left_camera`	Front left camera
`/cora/sensors/cameras/front_right_camera`	Front right camera
`/cora/sensors/lidars/front_lidar`	Front 3D lidar
`/cora/sensors/gps/gps`	GPS
`/cora/sensors/imu/imu`	IMU

Table 10: Task information API

Tasks[2]	
Topic Name	**Description**
`/clock`	Simulation time
`/vorc/task/info`	Task information

Table 11: Debug information API (not available during competition)

Debug[3]	
Topic Name	**Description**
`/vorc/debug/wind/direction`	Wind vector (units in degrees and ENU coordinate)
`/vorc/debug/wind/speed`	Magnitude of the wind
`/vorc/debug/contact`	CoRa collisions

The interface described is generic to the entire competition, including all tasks. The task-specific elements of the interface are described in the Virtual RobotX Competition and Task Descriptions, which details the additional ROS topics and services used to support individual task execution.

[2] Each task can use an additional set of ROS topics/services. Please consult the 2020 VORC task Descriptions document for additional information.

[3] All Gazebo topics to query ground truth information and other simulation aspects are available for debugging.

8. Submission and Code Execution

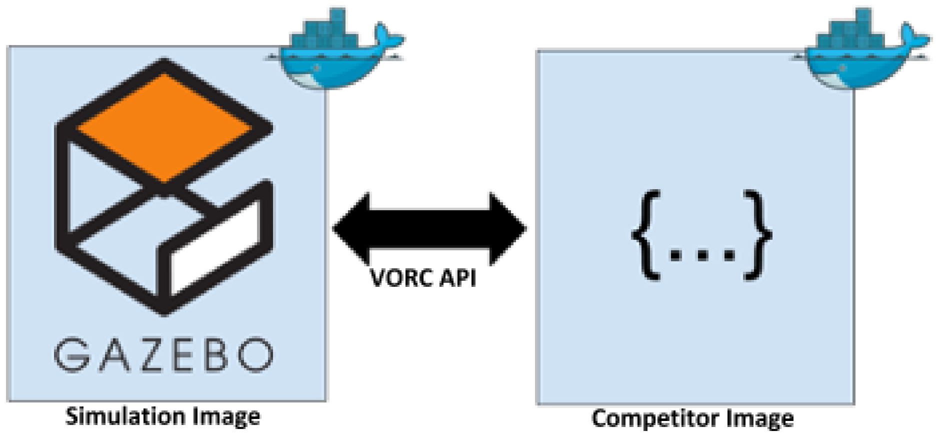

Figure9: Architecture used to execute competitor code

We expect to receive a file from each competitor prior to each event. This file will specify the team's controller. Please, see the VORC wiki page for detailed instructions about how to submit a solution for a given event.

Performance for each task will be evaluated as follows:

1. A Docker container running the VORC simulation image will be executed. This container will execute Gazebo with the VORC environment configured to run a particular task. Additionally, Gazebo will be configured to record a log of the execution.
2. A ROS bag (log file) will capture all task messages containing the score.
3. A team's Docker container (running the team's image) will be executed. It's expected that the entry point of this Docker instance spawns all the necessary elements of the team's code.
4. The competitor's code should interact with the simulation via the VORC API, determine the current task via the VORC API, and try to solve it.
5. When the task has been completed or has timed out, the Gazebo log file and the ROS bag will be saved and tagged appropriately.

This process will be repeated for each run of each task and for all the teams participating in the event. This architecture allows the execution of the entire competition in batch mode. Teams will be able to run a competition locally using the same set of tools that the organization will use during the official events. The automatic evaluation tool is available in the vrx-docker repository. We encourage all teams to use it for testing their solutions.