VRX Technical Guide

1. Introduction

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

Related documents include the following:

- 2019 Virtual RobotX Competition and Task Descriptions

2. Robotic Platform

The VRX competition will be executed using the Gazebo simulation environment. All teams must use the simulated version of the Wave Adaptive Modular Vessel (WAM-V) surface craft distributed with the VRX software. The WAM-V as supplied by the VRX software is standard for all teams. No modification of the standard platform model (URDF description, etc.) is allowed during competition.

The simulated WAM-V 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 hydrodynamic, external forcing and propulsion forces are generated by VRX plugins with fixed parameters. No modification of the standard VRX model parameters is allowed during competition.

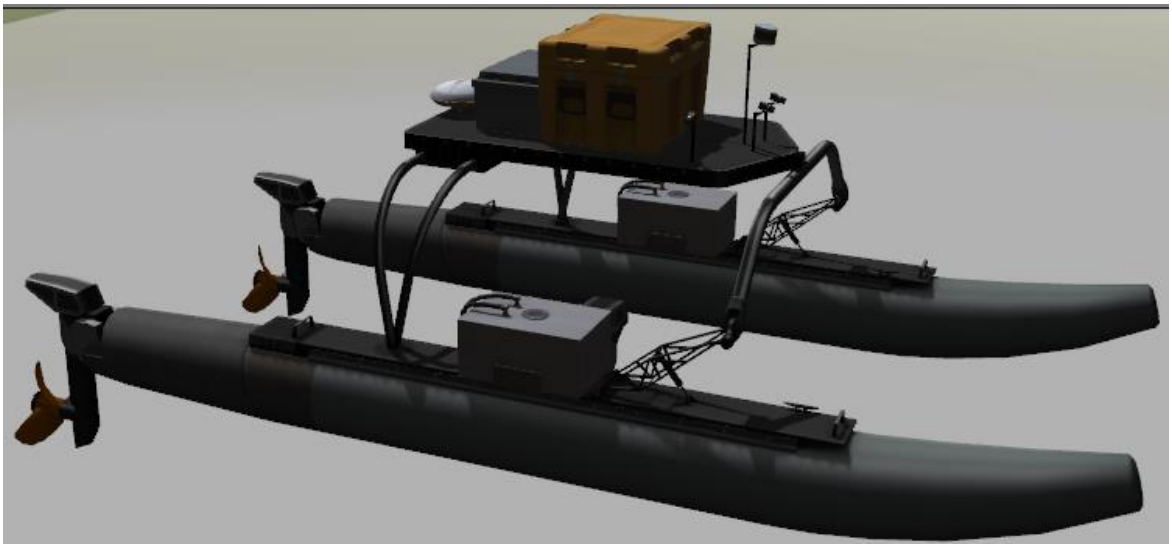

Figure 1: Simulated WAM-V model

3. Propulsion configuration

As part of the competition, teams may customize the propulsion system with which their WAM-V is equipped. Vehicle propulsion is provided by a set of identical thrusters. The details of the thruster model are available on the documentation page of the VRX website. The maximum number of allowed thrusters is four. The pose (position and attitude relative to the vessel) of each thruster is configurable via a YAML file.

In order to preserve realism, not all possible thruster positions will be accepted. The valid set of allowed positions are represented in the Figure 2 as five red bounding boxes. In order for a thruster configuration to

be considered valid, the origin of each thruster must be contained within one of these bounding boxes and each bounding box must contain no more than one thruster. Additionally, all thrusters have the ability to dynamically rotate and thus redirect the direction of its thrust.

Teams can visit the tutorials section of the VRX website for information about how to customize the WAM-V for running simulations locally. The same tutorials describe the format of the YAML file used for propulsion configuration.

During an actual event, teams will need to provide a valid YAML file named `propulsion.yaml` with their propulsion configuration. Working examples of YAML configuration files with typical propulsion configurations are described in the VRX tutorials.

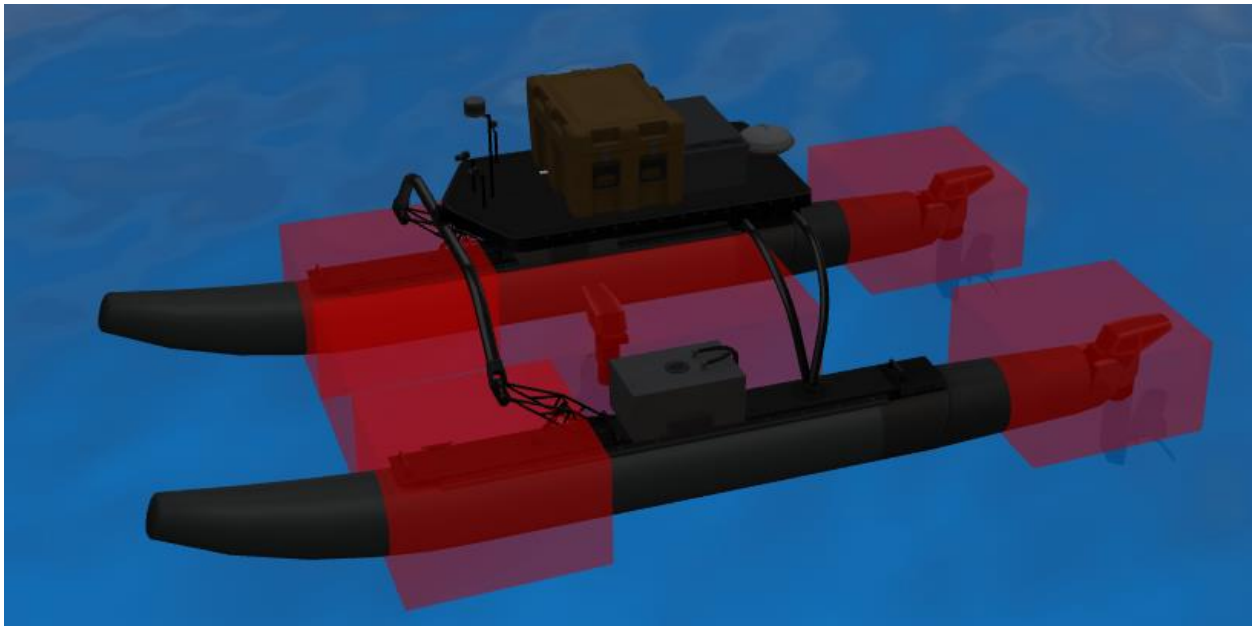

Figure 2: Allowed thruster regions in WAM-V

Table 1: Dimensions of Bounding Boxes

Name	Position and Orientation of Center of Bounding Box (x y z R P Y) (Relative to the Origin of the WAM-V)	Size (x y z)
Back Left	-2.25 1 0 0 0 0	1 1 1.2
Back Right	-2.25 -1 0 0 0 0	1 1 1.2
Front Left	1 1 0 0 0 0	1 1 1.2
Front Right	1 -1 0 0 0 0	1 1 1.2
Center	0.25 0 0 0 0 0	2.5 1 1.2

4. Sensor configuration

Similarly, to the propulsion system, teams should customize the sensor system with which their WAM-Vs are equipped by choosing the type of sensors and placement of those sensors. Standard sensors include navigation sensors (GPS and IMU) and perception sensors (cameras and lidars). In addition to configuring sensor types, teams may also customize the sensor placement.

Table 2: Maximum number of sensors allowed per sensor type

Sensor type	Maximum number of instances
Camera	3
Lidar	2
GPS/IMU	1

The available sensor types and their performance characteristics have been chosen to reflect commonly used sensors from the RobotX physical competition. 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 VRX. 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 3: Characteristics of VRX 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

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

4.2. Camera Sensor

A standard camera is simulated via the Gazebo camera plugin. Teams may provision their system with up to three cameras.

Table 4: Characteristics of VRX Camera Sensor

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

4.3. Lidar Sensors

Two types of lidar sensors are provided for VRX: 16 beam and 32 beam. Teams may provision their system with up to two lidars.

Table 5: Characteristics of VRX Lidar Sensors

	16 Beam	**32 Beam**
Update Rate [Hz]	10	10
Lasers (Number of beams)	16	32
Samples (Number of horizontal rotating samples)	1875	2187
Min Range [m]	0.9	0.9
Max Range [m]	130	130
Noise [m]	0.01	0.01
Min. Horizontal Angle [rad]	$-\pi$	$-\pi$
Max. Horizontal Angle [rad]	π	π
Min. Vertical Angle [rad]	$-\frac{\pi}{12}$	-0.186
Max. Vertical Angle [rad]	$\frac{\pi}{12}$	0.54

4.4. Sensor Placement

The pose of each sensor is configurable via a YAML file. In order to preserve realism, not all possible sensor positions will be accepted. The valid set of allowed positions are represented in the next figure as green bounding boxes. The origin of each sensor needs to be contained within one of these bounding boxes to be considered valid. Note that the specification of the sensor is not customizable; all teams share the same sensor types.

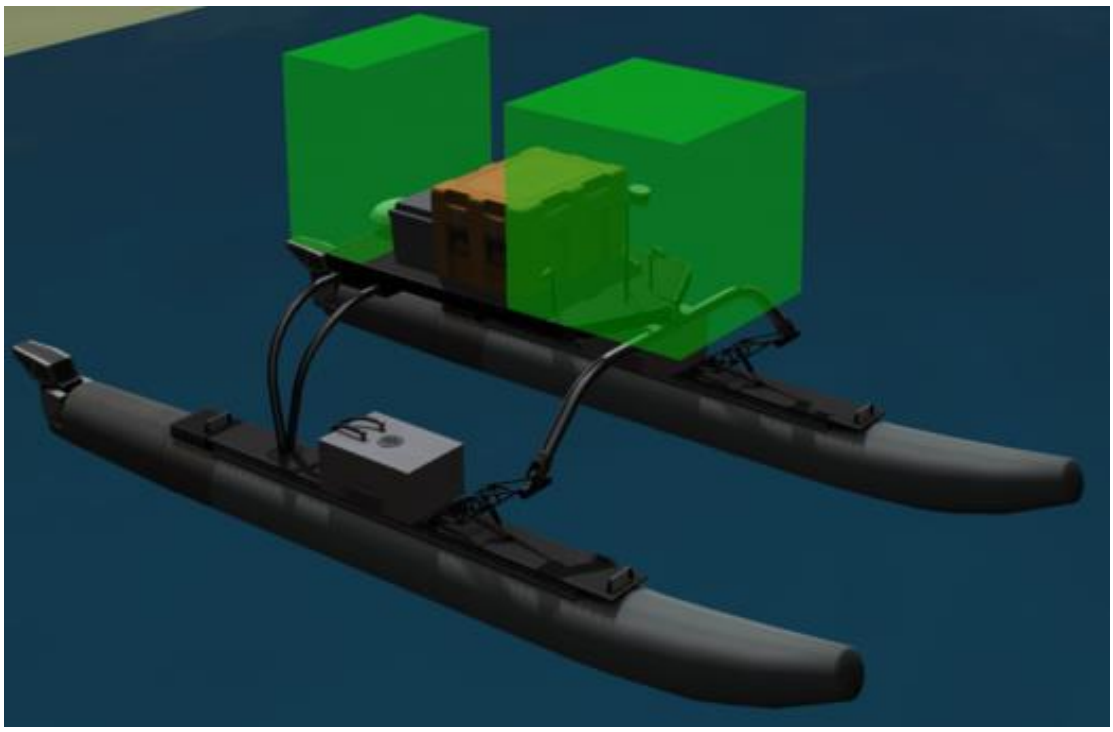

Figure 3: Allowed sensor regions in WAM-V

Teams can visit the tutorials section of the VRX website for information about how to customize their WAM-Vs for running simulations locally. The same tutorial describes the format of the YAML file used for sensor configuration.

During an actual event, teams will need to provide a valid YAML file named `sensors.yaml` with their sensor configuration.

5. Task Information

Teams face multiple independent tasks during VRX. 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 VRX API section of this document for further details.

Table 6: ROS Task message definition

Field Name	Description
`name`	Unique task name (e.g.: "scan_and_dock", "navigation_course").
`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 this 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 accordingly. 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 respectively. Once the task is in the *finished* state, teams still can control the vehicle, but the score will not change.

6. Task States

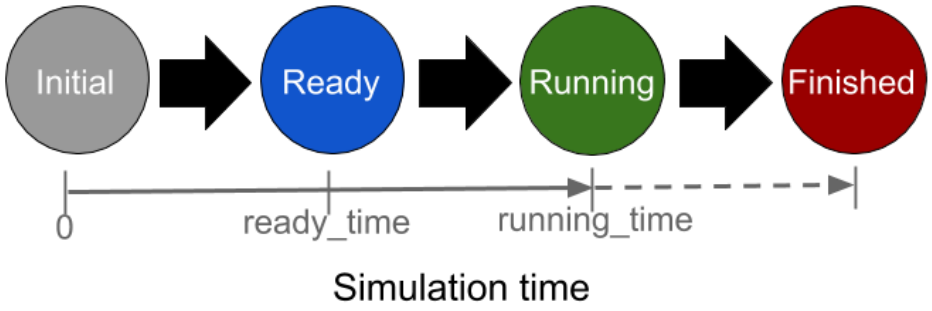

Figure 4: Task states as a function of time

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.

6.1. 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.

6.2. 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.

6.3. 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.

6.4. 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.

7. 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:

7.1. 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.

7.2. Fog

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

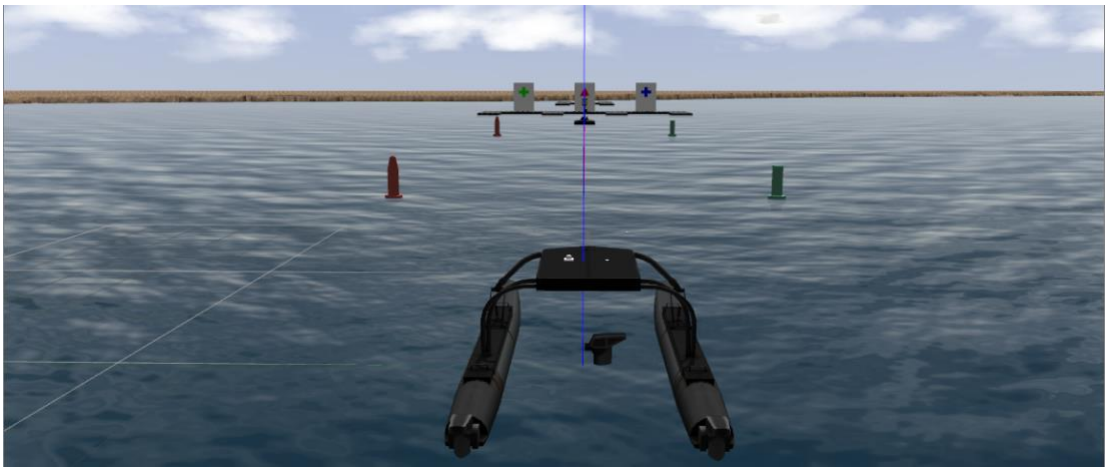

Figure 5: Visual scene with no fog

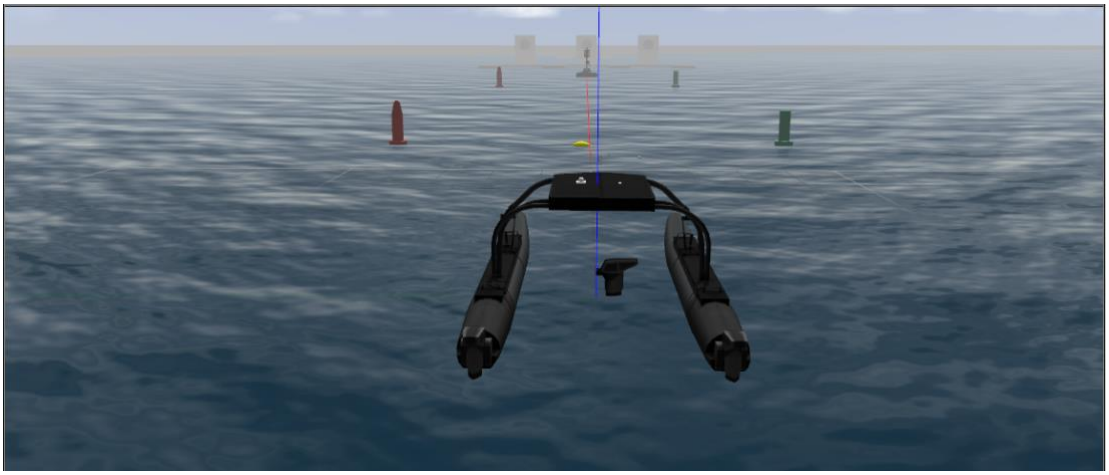

Figure 6: Visual scene with fog

7.3. Wind

Wind exerts a force on objects in the VRX 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.

7.4. 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 VRX competition combinations of T_p and γ as illustrated in Figure 7 will be used for evaluation.

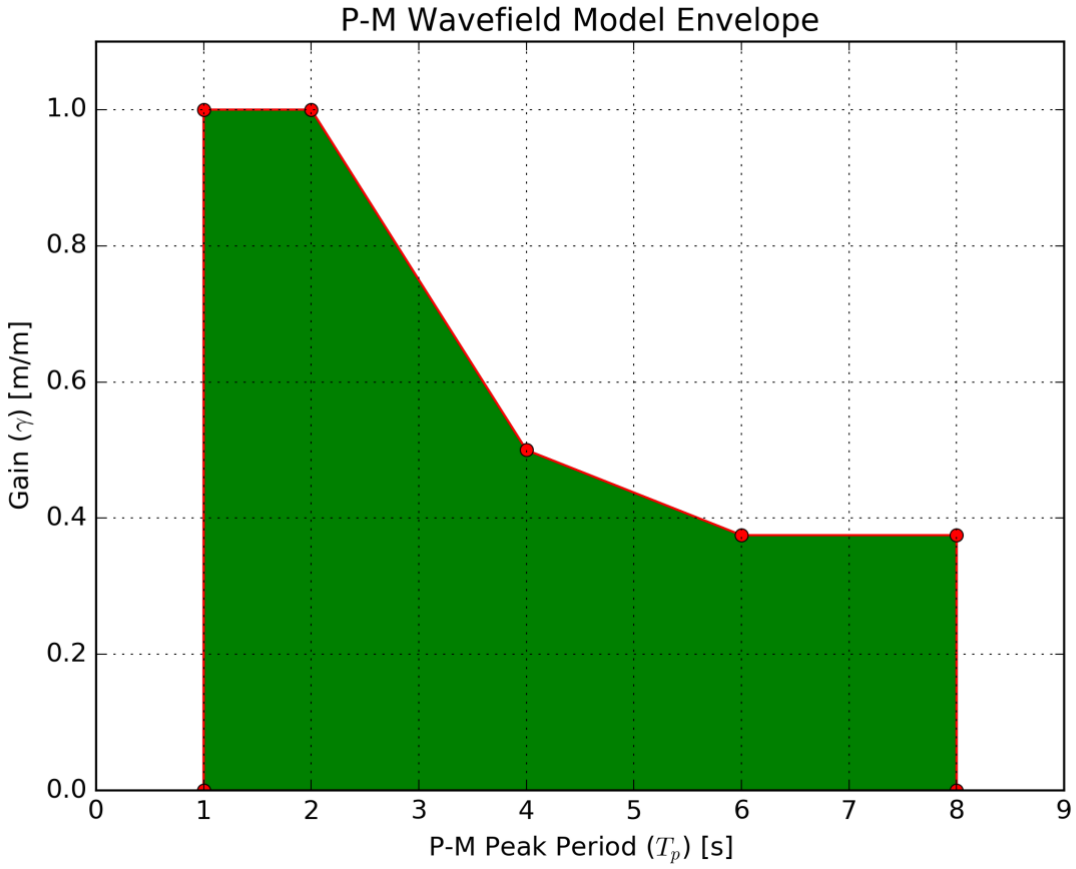

Figure 7: Envelope for sea state parameters used in VRX evaluation.

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

7.5. Ambient Light

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

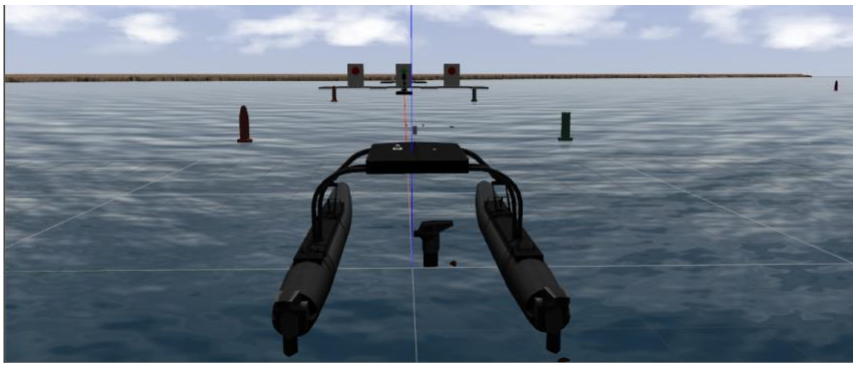

Figure 8: Visual scene with reduced ambient light

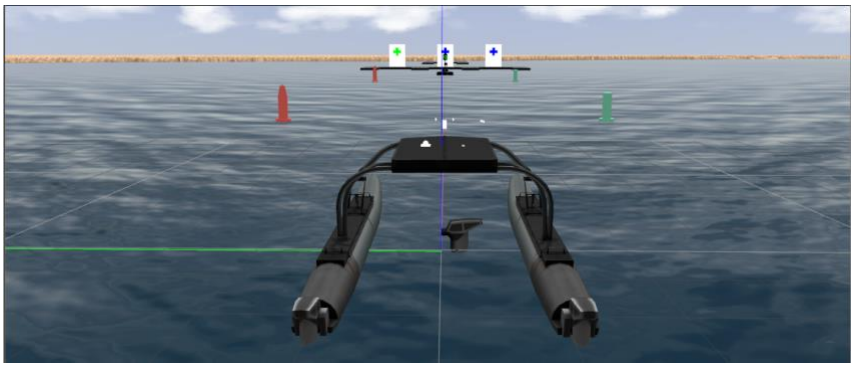

Figure 9: Visual scene with regular ambient light

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	8
`wamv_gazebo::wind_variance_gain`	0	8
`wamv_gazebo::wind_variance_time`	2	20
`wamv_gazebo::wind_direction`	0	360
`wamv_gazebo::wave_period`	See Figure 7.	
`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.

8. VRX API

VRX 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 (default topics)

Thruster Actuation	
Topic Name	**Description**
/wamv/thrusters/<thrustername>_cmd	Next power command for the <thrustername> thruster
/wamv/thrusters/<thrustername>_angle	Next angle command for the <thrustername> thruster

Table 9: Sensor information API (default topics)

Sensor Information	
Topic Name	**Description**
/wamv/sensors/cameras/<sensorname>	<sensorname> camera
/wamv/sensors/lidars/<sensorname>	<sensorname> 3D lidar
/wamv/sensors/gps/gps	GPS
/wamv/sensors/imu/imu	IMU

Table 10: Task information API

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

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

Debug[3]	
Topic Name	**Description**
/vrx/debug/wind/direction	Wind vector (units in degrees and ENU coordinate)
/vrx/debug/wind/speed	Magnitude of the wind
/vrx/debug/contact	WAM-V 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 2019 Virtual RobotX Competition and Task Descriptions document for additional information.

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

9. Submission and Code Execution

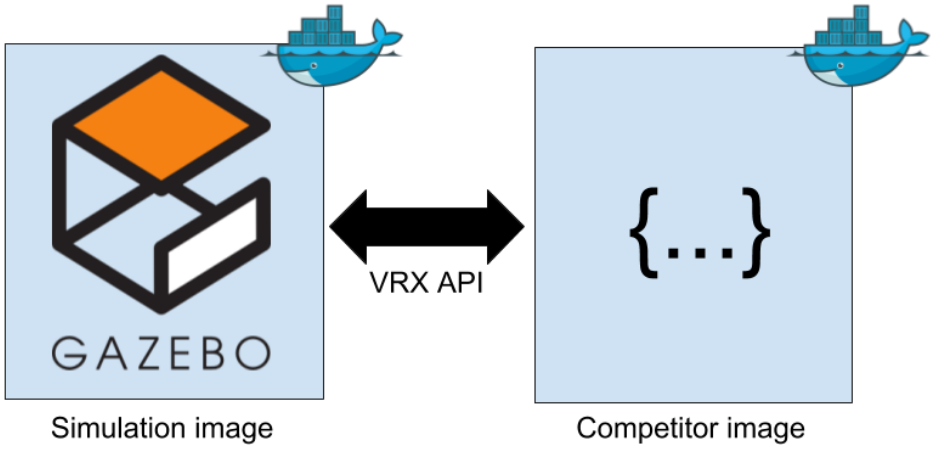

Figure 9: Architecture used to execute competitor code

We expect to receive multiple files from each competitor prior to each event. These files will specify different aspects of the WAM-V configuration, as well as the team's controller. Please, see the VRX 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. Each team's customized WAM-V will be generated.
2. A Docker container running the VRX simulation image will be executed. This container will execute Gazebo with the VRX environment configured to run a particular task. Additionally, Gazebo will be configured to record a log of the execution.
3. A ROS bag (log file) will capture all task messages containing the score.
4. 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.
5. The competitor's code should interact with the simulation via the VRX API, determine the current task via the VRX API, and try to solve it.
6. 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.