Technical Guide

2019 Virtual RobotX Competition *www.RobotX.org*

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. The current release of this technical guide is a preliminary draft intended to provide teams with an overview of the structure of the technical implementation of VRX and to solicit feedback from the teams and other stakeholders. Many of the technical details will change based on community feedback and continued development. These modifications will be captured in the final release of this document.

This document provides a technical guide to the VRX competition. It covers the following topics: the robotic platform, competition tasks, runs and environmental envelopes, sensors, the application programming interface (API), and instructions for submitting your code for the competition.

Related documents include the following:

- VRX 2019_Task Descriptions-PRELIM_v1.0

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.

The propulsion configuration for the WAM-V used in the VRX competition will consist of two stern mounted thrusters at fixed angles and one lateral/bow thruster, as shown in the figure below.

Figure 1: Simulated WAM-V model illustrating propulsion configuration

3. 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 for further details.

Table 1: 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 for taking control of the vehicle and start the task respectively. Once the task is in *finished* state, teams still can control the vehicle, but the score will not change.

4. Task States

A task can be in 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.

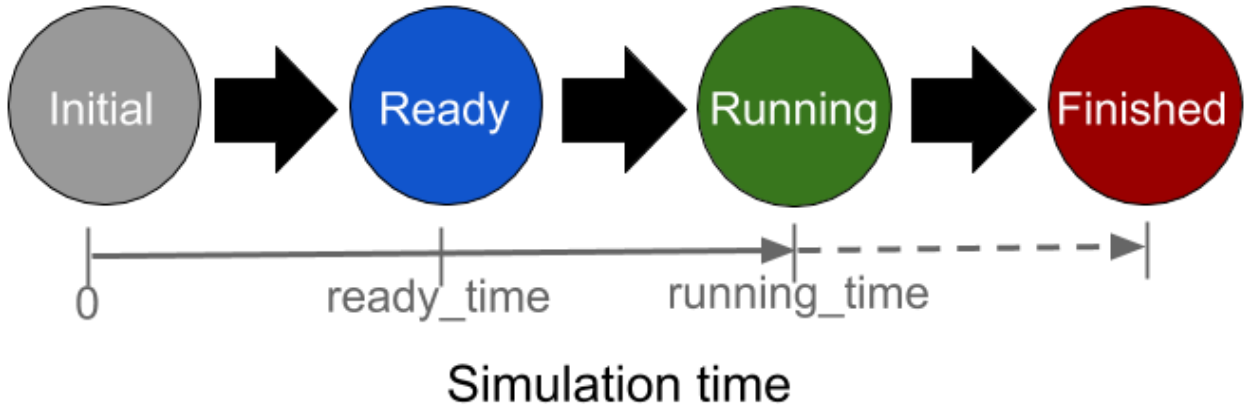

Figure 2: Task states as a function of time.

4.1. Initial

After Gazebo starts, the task is in *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.

4.2. 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.

4.3. 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.

4.4. 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.

5. Runs and Environmental Envelopes

For the competition, each team shall conduct multiple runs per task, where each run will use a different set of environmental conditions. Conditions will be distinct from run to run, but 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:

5.1. 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.

5.2. Wind

The magnitude and the direction of the wind velocity vector.

5.3. Waves

The wave field characteristics.

5.4. Fog

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

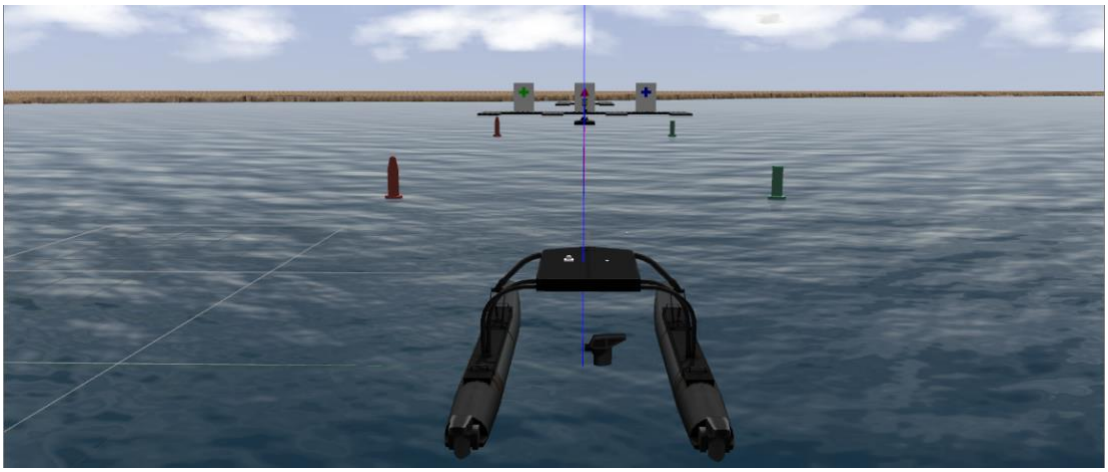

Figure 3: Visual scene with no fog

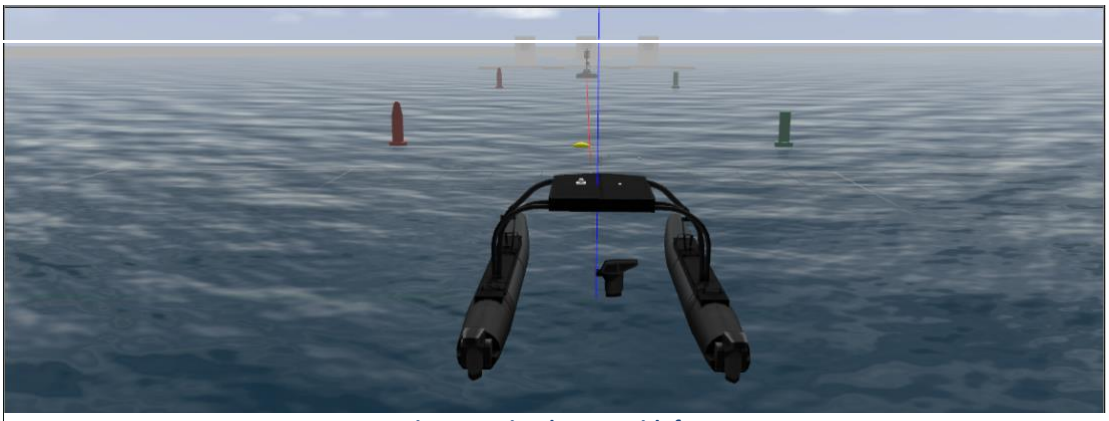

Figure 4: Visual scene with fog

5.5. Ambient Light

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

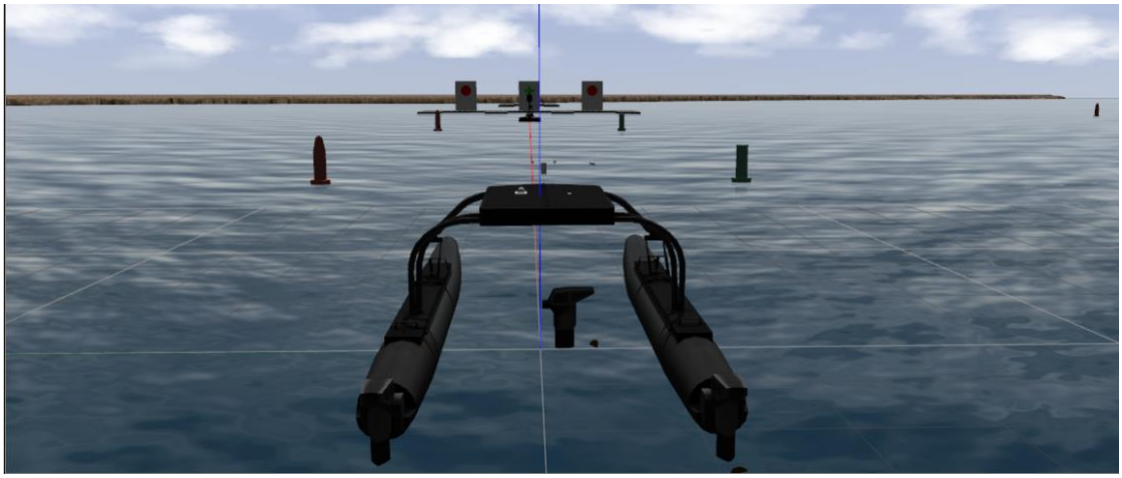

Figure 5: Visual scene with reduced ambient lighting

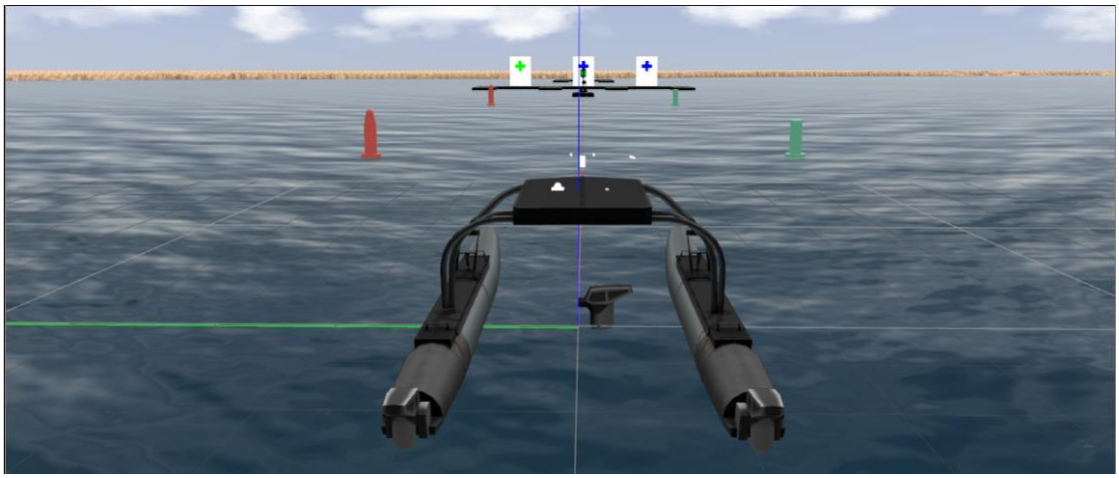

Figure 6: Visual scene with increased ambient light

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

Table 2: 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_velocity_vector	[0, 0, 0]	[8, 8, 8]
wamv_gazebo::wave_amp0	TBD	TBD
wamv_gazebo::wave_period0	TBD	TBD
wamv_gazebo::wave_direction0	TBD	TBD
wamv_gazebo::wave_amp1	TBD	TBD
wamv_gazebo::wave_period1	TBD	TBD
wamv_gazebo::wave_direction1	TBD	TBD
wamv_gazebo::wave_amp2	TBD	TBD
wamv_gazebo::wave_period2	TBD	TBD
wamv_gazebo::wave_direction2	TBD	TBD

The simulated wind and wave environment will be variable during competition runs. Final specifications for the exact characteristics will be released as part of the final technical specification, however the following are general range values teams can expect.

- Nominal wind velocity from 0 to 8 m/s in any direction.
- Significant wave height from 0 to 0.5 m where component waves are generated from a fully developed sea spectrum.

6. Sensors

The WAM-V vessel must be used with a fixed sensor suite configuration during competitions.

The following table details the sensor suite and its specification:

Table 3: Front Stereo Camera specification

Front Stereo Camera	
Update Rate	30 Hz
Resolution	800x800 px
Color format	R8G8B8

Table 4: Starboard camera specification

Starboard Camera	
Update Rate	30 Hz
Resolution	800x800 px
Color format	R8G8B8

Table 5: GPS specification

GPS	
Update Rate	1 Hz

Table 6: IMU specification

IMU	
Update Rate	15 Hz

Table 7: Front 3D LIDAR specification

Front 3D Lidar	
Update Rate	10 Hz
Vertical beams	16
Samples	1800
Resolution	0.1 deg
Min. horizontal angle	$-\pi$ rad
Max. horizontal angle	π rad
Min. vertical angle	$-\frac{\pi}{12}$rad
Max. vertical angle	$\frac{\pi}{12}$rad

The noise specifications of all these sensors will be fixed and announced in future versions of this document.

7. 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:

Table 8: Thruster actuation API

Thruster Actuation	
Topic Name	**Description**
/left_thrust_cmd	Next command for the left thruster
/right_thrust_cmd	Next command for the right thruster
/lateral_thrust_cmd	Next command for the lateral thruster

Table 9: Sensor information API

Sensor Information	
Topic Name	**Description**
/front_left_camera/	Front left camera
/front_right_camera/	Front right camera
/middle_right_camera	Starboard camera
/gps	GPS
/lidar_wamv	3D lidar
/imu	IMU

Table 10: Task information API

Tasks[1]	
Topic Name	**Description**
/clock	Simulation time
/vrx/task/info	Task information

The interface described above is generic to the entire competition, including all tasks. The task-specific elements of the interface are described in the VRX Preliminary Task Description document, which details the additional ROS topics and services used to support individual task execution.

[1] Each task can use an additional set of ROS topics/services. Please, consult the 2019 Virtual RobotX rules document for additional information.

8. Submission and Code Execution

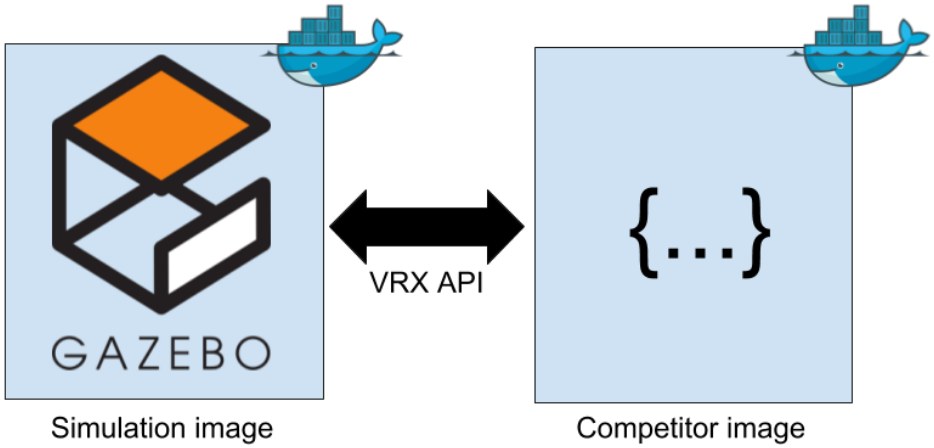

Figure 7: Architecture used to execute competitor code

We expect to receive a `Dockerfile` from each competitor prior to an event. The mechanism to communicate the `Dockerfile` will be announced before the event. This file should build a competitor Docker image and run all the competitor's code as an entry point. The organization will create a similar Docker image with all the simulation and VRX plugins.

Performance for each task will be evaluated as follows:

1. A Docker container running the 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.
2. A ROS bag (log file) will capture all task messages containing the score.
3. A Docker container running the competitor image will be executed. It's expected that the entry point of this Docker instance spawns all the necessary elements of the competitor's code.
4. The competitor's code should interact with the simulation via the VRX API, introspect the current task, 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 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. Currently, we are planning different events (qualifiers, semi-finals, finals), that will incrementally walk participants towards the setup needed for the finals. Additional details, including tutorials and working examples, will be provided to help teams submit their solutions.