Interface Control Document
Virtual Competition

Revision 5
February 19, 2025

Defense Advanced Research Projects Agency
Biological Technologies Office
675 North Randolph Street
Arlington, VA 22203-2114
TriageChallenge@darpa.mil

Contents

1 Introduction

This document describes the interfaces to the DARPA Virtual Testbed, the DARPA Triage Challenge (DTC) Virtual Competition's automated submission evaluation system. The intent is to convey the overall concept of operations for interaction with the Virtual Testbed during competition and to describe the software interfaces necessary to successfully submit DTC candidate solutions to be scored during the competition portion of each DTC project phase. This document covers interoperability with the Virtual Testbed only, for the Command Post interface for the Systems Competition or Data competition, please refer to their respective ICD documents.

2 Overview and Concept of Operations

DTC Virtual Competition participants will submit their candidate solutions as Docker containers that use the Robot Operating System v2 (ROS2) for communication with the Virtual Testbed's simulator and scoring system. The Virtual Testbed uses CARLA for simulation and the CARLA ROS Bridge for allowing competitors to interact with the simulator through ROS topics. The Virtual Testbed's scoring system also uses ROS topics to receive health assessments from competitors when running a scenario.

2.1 Evaluation

The Virtual Testbed will evaluate each candidate solution over several runs, varying the environment and distribution of virtual casualties, to produce a final aggregate score. During each run candidate solutions must control one or more simulated vehicle(s), identify virtual casualties in simulated vehicle sensor feeds, and submit health assessments for identified casualties. Each run will perform a fresh deployment of the candidate solution container alongside the competition simulator and other Virtual Testbed containers. After each run of the simulation completes, the deployment will be cleaned up to avoid issues with lingering state and processes. At the end of the evaluation for a candidate solution, the Testbed generates a final score report which includes all casualty reports and scores for each run and the final aggregate score over all runs.

2.2 Scoring

Candidate solutions will be evaluated against ***m*** number of competition scenarios to test the versatility of the solutions. Each competition scenario will, in turn, be evaluated over ***n*** replications (reps) to account for random variability. See Figure 1 for a graphical depiction. The Event Score of the ***m*** x ***n*** runs is given by:

$$Event\ Score = \frac{1}{m}\sum_{i=1}^{m}\left(\frac{\sum_{j=1}^{n} run\ score_{ij}}{n}\right)$$

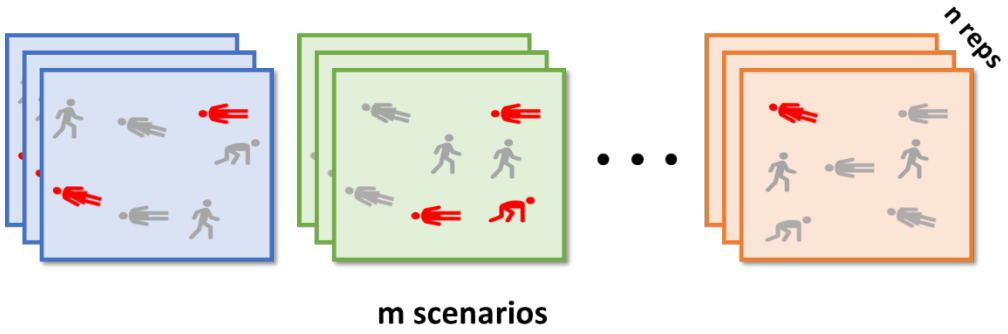

Figure 1. - Virtual Competition approach to scoring competition scenarios

Competitors will be scored based on their ability to correctly evaluate casualties in simulation. Casualties will be presented at different severities with both physical indicators of injury and severity (i.e. lacerations and burns) as well as underlying physiological manifestations of injury (i.e. changes in respiration and heart rate). Casualties will be scannable in multiple sensing modalities, beginning with RGB, thermal, and audio at a minimum for Phase 1. For specific details about scoring, see the DARPA Triage Challenge Competition Rules.

3 Candidate Solution Interfaces

This section provides technical specifications of the interfaces between the candidate solution and the Virtual Testbed evaluation system.

3.1 Container Requirements

Candidate solutions must be submitted as Docker containers which will be automatically deployed by the Virtual Testbed. A sample candidate solution container Dockerfile and supporting scripts will be provided for reference in each release of the Virtual Testbed. This will include READMEs, scripts, and Docker compose files to perform local deployments for testing. In Phase 2, the Virtual Testbed will utilize Docker Swarm to leverage simulating capabilities of multiple machines. The READMEs will include details and/or references on how to create Swarm clusters for multi-machine deployments. To allow the same container to be used for multiple simulated vehicles, candidate solutions must use environment variables to configure the container behavior for each simulated vehicle.

Base container images will be published to Amazon's ECR so participants will not have to build the required images from scratch for local testing. The container for the sample candidate solution is derived from a ROS base container with the casualty reporting ROS message types installed (see below). Participants are not required to base their solution on the sample container.

Candidate solutions should be submitted by uploading container images to ARA's Amazon ECR repository at competition time. ARA will provide a Nextcloud drop as a failsafe for competitors if they face issues with the ECR approach.

Other requirements for candidate solution containers:

1. Container should have a default entry point (no arguments) which launches all required processes (typically using ROS2 launch configurations as shown in the sample solution).
2. On startup subscribe to CARLA topics for vehicle sensors.
3. Publish casualty reports while processing sensor feeds.
4. Graceful startup and shutdown by sending and receiving lifecycle messages
5. Environment variables will be set in the deployed container to pass metadata and parameters for each run of the candidate solution. Candidate submissions must include a JSON file containing the environment variable values required to configure the container for each simulated vehicle. Details for deployment configuration are available in the Virtual Testbed's READMEs.

Supplementing the container submission, competitors will need to provide a vehicle configuration(s) YAML file. Details on the schema and semantics of the configuration are available in the Virtual Testbed READMEs provided in each release.

3.2 CARLA Topics and Message Types

The CARLA ROS Bridge exposes vehicles and their sensors as ROS topics of the form "`/carla/<Vehicle ID>/<Sensor_Name>`", with additional subtopics depending on the sensor types. For details on how to subscribe to these topics and consume the message consult the ROS documentation and references provided in the appendix. The sensors all use standard ROS message types, see linked documentation for details.

3.2.1 Sensor Topics

All sensors are now spawnable through the scenario file and are loaded in with predefined parameterizations. The topics made by the sensor will use the logic below:

- `/carla/<Vehicle ID>/<Sensor Name>`

And for some sensors, such as the camera, may produce multiple sensor messages, like the image and info. Below is an example of this format:

- `/carla/<Vehicle ID>/<Sensor Name>/camera_info`
- `/carla/<Vehicle ID>/<Sensor Name>/image`

See the appendix for the list of sensors supported by the DTC team. For more information on the Carla Sensor packages the following:

- Sensors reference - CARLA Simulator
- CARLA sensor reference - CARLA Simulator

Additionally, a new audio sensor has been added to the simulation. This follows the same parameterization and topic logic as the other sensors. It uses the new CarlaAudio msg format defined below:

```
# PortAudio configurations are used for this message.
# Refer to https://www.portaudio.com/docs/v19-doxydocs/portaudio_8h.html

uint32 PA_INT16 = 8

std_msgs/Header header
uint32 sample_format    # The Unreal Engine outputs data in int16 format so this will always be paInt16
uint32 num_channels     # Number of audio channels (1 for mono or 2 for stereo)
uint32 sample_rate      # Number of samples per second (ex. 1600)
uint32 chunk_size       # Size of each audio frame (ex. 4096)
int16[] data            # Audio data
```

3.2.2 Control Topics

There are two topics for tracking the status of the simulation `/dtc/simulation_start` and `/dtc/simulation_stop`. These topics receive ROS Empty messages and simply indicate the start of the simulation. After receiving from simulation_start, it is safe to begin measuring and reporting. After receiving from simulation_stop, containers should clean up and shutdown immediately before being forcefully terminated.

Competitor submissions must also repeatedly publish an empty message to `/dtc/competitor_ready` to indicate that the container is ready to measure and report. The simulation will not start until the competitor container has sent a ready signal.

3.2.3 Vehicle Control

Vehicles may be controlled by publishing a control message to the vehicles' control topics. The topics and message formats are as described:

- Quad Copter:
  - Topic: `/carla/<Vehicle ID>/multirotor_control_cmd`
  - Format:
    - `std_msgs/Header header`
    - `float32[] throttle`
  - The throttle vector should contain four floating point values, one for each rotor. The rotor indices are assigned according to Figure 2.

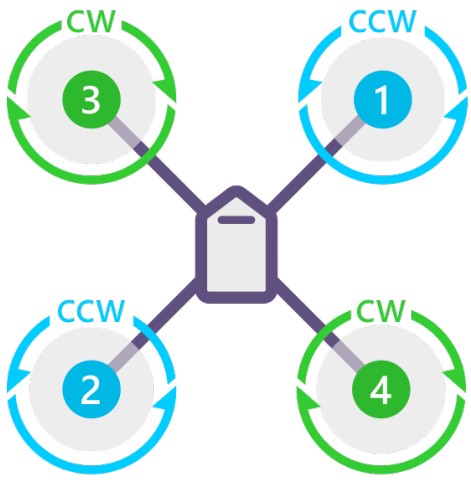

Figure 2. Quadcopter rotor indices.

- Skid Steer: In Progress
- Ackerman: In Progress
- 4-Legged: In Progress

3.3 Format and Distribution of Casualty Reports

Casualty reports must identify the location of the casualty to be reported on, the simulation time associated with the report, and a health assessment. Details on the rules of health reports can be found in the DARPA Triage Challenge Competition Rules. All casualty reports are submitted as JSON embedded in ROS's standard String type message.

```
import json
from rclpy.node import Node
from std_msgs.msg import String

class DtcvcScoringTestCompetitor(Node):
    def __init__(self):
        super().__init__('dtcvc_scoring_test_competitor_node')
        publisher = self.create_publisher(String, '/competitor/drone_results',
10)
        publisher.publish(String(data=json.dumps({"foo": "bar"})))
```

Figure 3. Example code of submitting/publishing a JSON object to a topic

3.3.1 Initial Reports

Initial reports for newly identified casualties.

The request content must be a JSON-encoded object with the following format:

```
{
        "casualty_id": <int>,
        "team": <string>,
        "system": <string>,
        "location":
        {
                "latitude": <float>,
                "longitude": <float>,
                "time_ago": <int>
        },
        "severe_hemorrhage": <int>,
        "respiratory_distress": <int>,
        "hr": <value_at_time>,
        "rr": <value_at_time>,
        "temp": <value_at_time>,
        "trauma_head":  <int>,
        "trauma_torso": <int>,
        "trauma_lower_ext": <int>,
        "trauma_upper_ext": <int>,
        "alertness_ocular": <value_at_time>,
        "alertness_verbal": <value_at_time>,
        "alertness_motor": <value_at_time>
}
```

where **value_at_time** is a sub-object with the following format:

```
{
        "value": <int>,
        "time_ago": <int>
}
```

Each field has the following definitions:

- *casualty_id* is a new, unique integer identifier for the casualty whose condition is being reported
- *team* is a unique string identifier for the team submitting the report
- *system* is a unique string identifier for the unmanned system responsible for the identification of the casualty
- *location* is the coordinates of the reported casualty expressed as *latitude* and *longitude* at time *time_ago* relative to the time the report was submitted.
- *severe_hemorrhage* indicates whether the casualty shows signs of severe hemorrhage or not
- *respiratory_distress* indicates whether the casualty shows signs of respiratory distress or not
- *hr* indicates the heart rate in beats per minute for the reported casualty at time *time_ago* relative to the time the report was submitted.

- *rr* indicates the respiration rate in breaths per minute for the reported casualty at time *time_ago* relative to the time the report was submitted.
- *temp* indicates the core temperature in degrees Fahrenheit for the reported casualty at time *time_ago* relative to the time the report was submitted.
- *trauma_head* indicates presence of injury on the head for the reported casualty.
- *trauma_torso* indicates presence of injury on the torso for the reported casualty.
- *trauma_lower_ext* indicates the presence and type of injury on one or both lower extremities for the reported casualty.
- *trauma_upper_ext* indicates the presence and type of injury on one or both upper extremities for the reported casualty.
- *alertness_ocular* indicates the presence and type of alertness based on eye movement at time *time_ago* relative to the time the report was submitted.
- *alertness_verbal* indicates the presence and type of alertness based on speech and vocalizations at time *time_ago* relative to the time the report was submitted.
- *alertness_motor* indicates the presence and type of alertness based on gross motor movement at time *time_ago* relative to the time the report was submitted.

The following fields are required:

- *casualty_id*
- *team*
- *system*
- *location*
- At least three of the health assessment fields: *severe_hemorrhage, respiratory_distress, hr, rr, temp, trauma_head, trauma_torso, trauma_lower_ext, trauma_upper_ext, alertness_oculatr, alertness_verbal, alertness_motor*

Remaining health assessment fields are optional. Any included fields must contain all sub-fields; for example, *alertness_verbal* must contain both *value* and *time_ago* sub-fields.

For all relative timestamps, *time_ago* is a non-negative integer representing the elapsed time (in seconds) since location was estimated. This relative timestamp is used to determine what ground truth to score the reported value against. *time_ago* may differ between fields in the same report, and it will only pertain to the field it is contained within. For example, *time_ago* within the *location* field will only be used to determine the time at which to compare reported casualty location to the ground truth location.

Each field takes specific acceptable values, with semantics for each casualty report field as follows:

- *location*
    - *latitude*: float-precision number with range from -90 and 90 in degrees
    - *longitude*: float-precision number with range from -180 and 180 in degrees
    - *time_ago*: non-negative integer in seconds
- *severe_hemorrhage*
    - *1* indicates the casualty shows signs of the severe hemorrhage

    - *0* indicates the casualty does not show signs of severe hemorrhage
- *respiratory_distress*
    - *1* indicates the casualty shows signs of the respiratory distress
    - *0* indicates the casualty does not show signs of respiratory distress
- *hr*
    - *value*: non-negative integer indicating heart rate in beats per minute
    - *time_ago*: non-negative integer in seconds
- *rr*
    - *value*: non-negative integer indicating respiratory rate in breaths per minute
    - *time_ago*: non-negative integer in seconds
- *temp*
    - *value*: non-negative integer indicating core temperature in degrees Fahrenheit
    - *time_ago*: non-negative integer in seconds
- *trauma_head*
    - *0 (normal)* indicates absence of injury on the head
    - *1 (wound)* indicates presence of one or more injuries on the head
    - *2 (not testable)* indicates assessment of head injury was not possible
- *trauma_torso*
    - *0 (normal)* indicates absence of injury on the torso
    - *1 (wound)* indicates presence of one or more injuries on the torso
    - *2 (not testable)* indicates assessment of torso injury was not possible
- *trauma_upper_ext*
    - *0 (normal)* indicates absence of injury on both upper extremities
    - *1 (wound)* indicates presence of non-amputation injury on one or both upper extremities
    - *2 (amputation)* indicates amputation of one or both upper extremities
    - *3 (not testable)* indicates assessment of upper extremity injury was not possible
- *trauma_lower_ext*
    - *0 (normal)* indicates absence of injury on both lower extremities
    - *1 (wound)* indicates presence of non-amputation injury on one or both lower extremities
    - *2 (amputation)* indicates amputation of one or both lower extremities
    - *3 (not testable)* indicates assessment of lower extremity injury was not possible
- *alertness_ocular*
    - *0 (open)* indicates eyelids are open and/or blinking spontaneously
    - *1 (closed)* indicates eyelids are closed
    - *2 (not testable)* indicates eyelid assessment was not possible
- *alertness_verbal*
    - *0 (normal)* indicates responsiveness to speech prompts with coherent and relevant speech
    - *1 (abnormal)* indicates confused or pain-/distress-related speech and/or vocalizations
    - *2 (absent)* indicates absence of vocalization
    - *3 (not testable)* indicates speech and vocalization assessment was not possible
- *alertness_movement*
    - *0 (normal)* indicates walking/standing/sitting unsupported with coordinated movement
    - *1 (abnormal)* indicates lying or sitting supported with minimal limb movement
    - *2 (absent)* indicates absence of limb movement

- *3 (not testable)* indicates movement assessment was not possible

Complete definitions of report fields can be found in the DARPA Triage Challenge Competition Rules.

3.3.2 Update Report

Reports to update casualty information of a previously identified casualty.

The request content must be a JSON-encoded object with the following format:

```
{
        "casualty_id": <int>,
        "team": <string>,
        "system": <string>,
        "location":
        {
                "latitude": <float>,
                "longitude": <float>,
                "time_ago": <int>
        },
        "hr": <value_at_time>,
        "rr": <value_at_time>,
        "alertness_ocular": <value_at_time>,
        "alertness_verbal": <value_at_time>,
        "alertness_motor": <value_at_time>
}
```

The semantics of the fields follow the standards set in Section 6.2 with the following exception: *casualty_id* is a previously reported integer identifier used to associate this update report with the initial report on the same casualty

3.4 Logs

With the default setup, final score reports are output to `deployment/data/output/testbed/logs` relative to the Virtual Testbed root directory. Comprehensive logs for the whole swarm are *not* automatically stored in any location.

3.5 Video Outputs

As described in the DARPA Triage Challenge Competition Rules, candidate solutions must provide visual representations of their algorithms during the competition. Solutions shall write video files into the `/mnt/candidate_user_interface/` directory, which will be mounted into each running agent. The same host folder will be mounted, so unique filenames must be used across all running agents.

4 Appendix A – Standards/References

- Robot Operating System v2 - https://docs.ros.org/en/rolling/index.html
- CARLA - https://carla.org/
- CARLA ROS Bridge - https://carla.readthedocs.io/projects/ros-bridge/en/latest/run_ros/
- RGB Camera sensor documentation - https://carla.readthedocs.io/projects/ros-bridge/en/latest/ros_sensors/#rgb-camera
- Depth Camera sensor documentation - https://carla.readthedocs.io/projects/ros-bridge/en/latest/ros_sensors/#depth-camera
- IMU sensor documentation - https://carla.readthedocs.io/projects/ros-bridge/en/latest/ros_sensors/#imu
- GNSS sensor documentation - https://carla.readthedocs.io/projects/ros-bridge/en/latest/ros_sensors/#gnss
- Radar sensor documentation - https://carla.readthedocs.io/projects/ros-bridge/en/latest/ros_sensors/#Radar
- Lidar sensor documentation - https://carla.readthedocs.io/projects/ros-bridge/en/latest/ros_sensors/#Lidar
- Odometry sensor documentation - https://carla.readthedocs.io/projects/ros-bridge/en/latest/ros_sensors/#odometry-sensor