DARPA TRIAGE CHALLENGE

Interface Control Document
Virtual Competition

Revision 2
July 2024

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.

This iteration of the document will mention both current standards and future standards. Current standards apply to the release of the Virtual Testbed used for the Fall 2024 competition. Future standards are expected to apply in Phase 2 releases and beyond. Any future standard specified in this iteration is subject to change, but changes should be minimal. If, at any point, current and future standards conflict, this document will explicitly clarify which standards are applicable to which Virtual Testbed releases.

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 competition uses CARLA as the simulator and the CARLA ROS Bridge allows 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.

In the Phase 1 competition only, candidate solutions are not responsible for providing a full autonomy stack and fully controlling virtual drone(s). Instead, competitors will be provided a list of predefined waypoints in the virtual environment and will submit their desired order of waypoint visits. The details for specifying waypoints are laid out in 3.2.3

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:

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

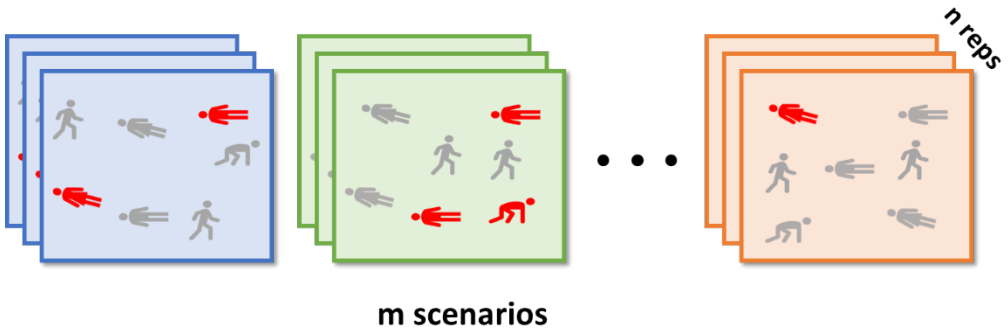

Figure 1. - Virtual Competition approach to scoring competition scenarios

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 more details see the DTC Virtual Competition Rules document.

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.

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. Further information for generating and submitting container images will be provided on the DTC Virtual Portal.

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 will require 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. Details for deployment configuration are available in the Virtual Testbed's READMEs.

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 ID>", 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

The following CARLA sensor topics will be available:

- `/carla/<Vehicle ID>/rgb`
  - See RGB Camera sensor documentation
- `/carla/<Vehicle ID>/depth`
  - See Depth Camera sensor documentation
- `/carla/<Vehicle ID>/ir`
  - See RGB Camera sensor documentation
- `/carla/<Vehicle ID>/imu`
  - See IMU sensor documentation
- `/carla/<Vehicle ID>/gps`
  - See GPS sensor documentation
- `/carla/<Vehicle ID>/radar`
  - See Radar sensor documentation
- `/carla/<Vehicle ID>/lidar`
  - See Lidar sensor documentation
- `/carla/<Vehicle ID>/audio_file_name`
  - See below

The Virtual Testbed operates on a recent version of CARLA built upon Unreal Engine 5 that does not yet contain support for audio sensors. For Phase 1, audio will be provided in a custom fashion on the audio_file_name topic. Messages on the audio topic will be standard String messages containing the filename of an audio WAV file – e.g. "97f107.wav". The Virtual Testbed will mount the audio files into `/mnt/wavs` directory in competitor containers. Competitor containers will be responsible for loading

and analyzing the audio within the WAV files (e.g. "`/mnt/wavs/97f107.wav`") to determine verbal alertness.

3.2.2 Control Topics

For Phase 1 candidate solutions will not have control over the simulated vehicles. Additional CARLA topics (TBD) will be used in Phase 2 and beyond to send control signals to simulated vehicles.

Instead of leveraging autonomy, teams will be allowed to specify a series of waypoints for simulated vehicles to follow for each scenario run prior to the Phase 1 competition. Waypoint selection will be done for each virtual environment used in the competition. Selectable waypoints will be shown on a top-down map of the scenario environment. Chosen waypoints for each candidate will be included as part of the overall competition configuration. For local testing candidates will be able to specify their own waypoint configuration using the scripts provided in the Virtual Testbed repository.

There are two topics for tracking the status of the simulation `/dtcvc/simulation_start` and `/dtcvc/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 publish an empty message to `/dtcvc/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 Waypoint Selection

Waypoints are defined as part of a simple YAML object that is incorporated into the full configuration of a scenario to simulate. Competitors need to provide:

1. The map for which the waypoint list applies
2. The list of waypoints

Each waypoint is defined by a zone number from the map and a dwell time in seconds. An example looks like the following:

```
map: train_derailment
waypoints:
  - zone: 14
    dwell_time: 12
  - zone: 8
    dwell_time: 9
```

Valid values for the "map" property will be released at competition time along with the top-down views of the maps with numbered waypoints. The numbers on the map representing waypoints are also called zones, hence the "zone" property in the yaml.

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. Health assessments do not need to be full and complete; they may include a subset of the scorable fields. For instance, competitors may report heart rate early and subsequently report on injuries and traumas. Per casualty, teams can submit multiple reports for a single scorable field, however, only the last report received within a run for each field will be scored. Casualty reports can be submitted by publishing a triage report message to a ROS topic (/competitor/drone_results) monitored by the Virtual Testbed's scorekeeper service. At the end of a run, all submitted reports are aggregated and tallied for a final score. See the DTC Competition Rules document for more details on how reports are scored. The following documentation explains how to create and publish triage reports.

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"})))
```

3.4 Casualty Report Message Types

Fields with an asterisk are required. Fields with double asterisks indicate a "one-of" requirement where a leaf property (aka terminal property) of one of the fields must be provided. For instance, a report might include vitals.heart_rate, injuries.severe_hemorrhage, or injuries.trauma.upper_extremity.

Table 1. Properties of casualty report

Field	Type	Allowed Values	Description
*observation_start	number	x >= 0	The simulation time, in seconds, at which observation for this report started
*observation_end	number	x >= 0	The simulation time, in seconds, at which observation for this report ended
*assessment_time	number	x >= 0	The simulation time, in seconds, at which the scorekeeper should compare this report to ground truth. Typically, this will be the same as

			observation_end or between observation_start and observation_end
*casualty_id	int	x >= 0	The ID of the casualty being reported. This is a value created by the competitor and is only used by the scorekeeper for aggregating disjointed reports
*location	Location	(see Table 2)	The location of the casualty being reported on
drone_id	int	x >= 0	The ID of the drone making the observation. This is for records only and does not affect scoring
**vitals	Vitals	(see Table 3)	The reported vitals of the casualty
**injuries	Injuries	(see Table 4)	The reported injuries of the casualty

Table 2. Properties of casualty location

Field	Type	Allowed Values	Description
*lon	number	-180 < x <= 180	The longitude, in degrees, of the casualty
*lat	number	-90 <= x <= 90	The latitude, in degrees, of the casualty
*alt	number	All reals	The altitude, in meters, of the casualty

Table 3. Properties of casualty vitals

Field	Type	Allowed Values	Description
heart_rate	number	x >= 0	Estimated heart rate of the casualty in beats/min.
respiration_rate	number	x >= 0	Estimated respiration rate of the casualty in respirations/min.

Table 4. Properties of casualty injuries

Field	Type	Allowed Values	Description
severe_hemorrhage	bool		Whether the casualty has a severe hemorrhage
respiratory_distress	bool		Whether the casualty is suffering from respiratory distress
**trauma	Trauma	(see Table 5)	Observed traumas or lack thereof
**alertness	Alertness	(see Table 6)	Observed degrees of casualty alertness

Table 5. Properties of casualty trauma

Field	Type	Allowed Values	Description
head	string	Enum{wound|normal}	Status of trauma to casualty head
torso	string	Enum{wound|normal}	Status of trauma to casualty torso
upper_extremity	string	Enum{wound|normal|amputation}	Status of trauma to casualty upper extremities
lower_extremity	string	Enum{wound|normal|amputation}	Status of trauma to casualty lower extremities

- All Regions
  - "normal" indicates there is no injury present on the body region
  - “wound” indicates an injury is present
- Extremities
  - “amputation” indicates there has been an amputation of an extremity in the body region

Table 6. Properties of casualty alertness

Field	Type	Allowed Values	Description
ocular	string	Enum{open|closed|nt}	Observed state of casualty’s ocular alertness
verbal	string	Enum{normal|abnormal|absent|nt}	Observed state of casualty’s verbal alertness
motor	string	Enum{normal|abnormal|absent|nt}	Observed state of casualty’s motor alertness

- Ocular
  - “open” indicates that eyelids are open – blinking included
  - “closed” indicates the eyelids are closed – blinking does not count
  - “nt” indicates eyelid status was not testable
- Verbal
  - “normal” indicates the casualty is responsive to speech with coherent speech
  - “abnormal” indicates the presence of non-speech or incoherent vocalization
  - “absent” indicates the absence of any vocalization
  - “nt” indicates alertness status was not testable
- Motor
  - “normal” indicates purposeful movement of limbs, response to commands to move, or walking
  - “abnormal” indicates minimal movement, twitching of limbs, or involuntary movement suggesting injury or pain
  - “absent” indicates no movement of limbs
  - “nt” indicates motor status was not testable

An example of a full assessment may look like the following:

```
{
    "drone_id": 19,
    "casualty_id": 0,
    "observation_start": 82.81012895485598,
    "assessment_time": 84.79181373133218,
    "observation_end": 94.12918343809109,
    "vitals": {
        "heart_rate": 119.26083171153168,
        "respiration_rate": 18.288651779837502
    },
    "injuries": {
        "severe_hemorrhage": false,
        "respiratory_distress": true,
        "trauma": {
            "head": "wound",
            "torso": "normal",
            "upper_extremity": "wound",
            "lower_extremity": "normal"
        },
        "alertness": {
            "ocular": "closed",
            "verbal": "abnormal",
            "motor": "nt"
        }
    },
    "location": {
        "lon": -109.456,
        "lat": 33.123,
        "alt": 19.22733433732779
    }
}
```


3.5 Logs

Logs are output into a known location in the scorekeeper container. During testing and deployment, a host system directory can be mounted to that location to extract logs. There will be one JSON log per run mapping each casualty ID to an object containing all reports submitted for that casualty, the final aggregated assessment, and the final score achieved for that casualty. The Virtual Testbed's READMEs contain details on how to configure mounted directories.

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
- GPS 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