Virtual Autonomous System Training (VAST) Test Plan and Report

MSIM Capstone Team 2018-2019

This document is the test plan and report for the Virtual Autonomous System Training (VAST) project. It describes an overview of the system (Section I) and the requirements for the project to be considered complete. Additionally, the unit test procedures and associated measures for each requirement are presented (Section II). The proof-of-concept (integration tests) and its test cases are be described (Section III). The limitations to these unit tests are also presented (Section IV). Finally, any references to external documents will be presented (Section V).

I. System Overview

Virtual Autonomous System Training (VAST) is comprised of the AV Testbed and the User Interface (UI) with its data and visualization components. User-defined parameters and metrics will be entered into the User Interface, and the system will generate a visualization based on these parameters and metrics. Each auxiliary module (simulation, AV / software, and UI) will send and receive information to and from the AV Testbed. The system architecture can be seen in Figure 1 below.

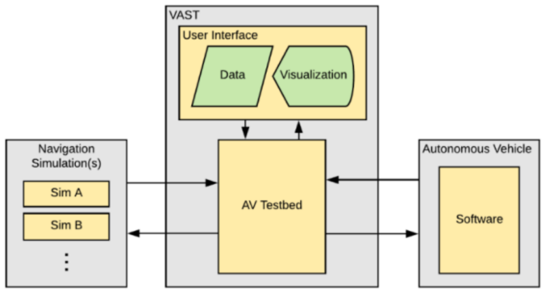


Figure 1: System Architecture

II. Unit Tests

Verification of VAST must be performed to ensure that it meets all project requirements. Each requirement will have a test description, criteria for accepting that the requirement is met, and description on how these requirements solve the problem statement. Criteria will include measures that may be thresholded or un-thresholded. A thresholded measure has a specific valued condition that must be met as a condition for the system to pass the test; in other words, it is a critical measure. However, an un-thresholded measure means that the value is calculated and tracked but is not required for the system to pass the test. This format is seen in Figure 1 below:

Requirement Name
Supporting Module in VAST
- a. Module Description
- b. Test Procedure
  - i. Test Category
    1. Test Cases
       - a. Pass/Fail Criteria
       - b. Expected Output
    2. Completion Date
       - a. Actual Output

Figure 1: Requirement Test Description Format

Critical Operational Issues (COIs) will be addressed as well in the test plan; these include effectiveness (mission support) and suitability (reliability, maintainability, logistic supportability, and availability) determinations. This plan must be developed in accordance with the requirements and its acceptance criteria, which are described in Table 1 below.

Table 1: System Requirements

Requirement	Acceptance Criteria
AV Testbed Process	
1.1: AV Testbed shall accurately represent data input to clients.	With sensor parameters provided, the simulation must be able to understand the AV's decision. This decision must be representative of the AV's actual decision.
1.2: AV Testbed shall visualize the environment accurately.	The virtual environment visualized must represent AVs and other obstacles' position and geometry accurately.
AV Testbed Inputs	
2.1: AV Testbed shall be abstracted to handle multiple sensor and vehicle types that can be inputted by the user. These sensor and vehicle types shall have acceptable and realistic parameters.	The user must be able to add new sensor types with different parameters and functionality. The sensor parameters must provide the simulation with enough information to understand how the sensor will act including what types of data are associated with it.
2.2: AV Testbed shall receive and store scene simulation information.	The simulation information must be abstracted to handle multiple types of simulations.
AV Testbed Outputs	
3.1: AV Testbed shall accurately state the output made by the AV.	The geometric output may include information about the state of the environment surroundings, the AV, the sensor readings, and the resulting output of the vehicle.
3.2: AV Testbed shall evaluate whether a failure/operational mission failure has occurred and output its result.	A failure is attributed to any software issue that results in unpredicted behavior. An operational mission failure (OMF) is a failure that prevents the entire system from being used (e.g. crash).
3.3: AV Testbed shall report relevant performance measures	Performance measures include quantitative and qualitative information regarding the overall performance of the AV. Relevant measures are described in Section 2.3.

Table 2 below lists the basic unit test procedures, associating each requirement with specific components within the AV Testbed. Any interaction with other components may not be included in these procedures unless they are part of the requirement. Many requirements will have a reliability measure associated with it.

Commented [MP1]: within the*

Commented [MP2]: Not sure I understand this definition.

Table 2: Basic Unit Test Procedures and Associated Measures

Requirement	Basic Unit Test Procedures	Associated Measures
AV Testbed Process		
1.1: AV Testbed shall accurately represent data input to clients.	-Tester will run scenarios that require data transfer between AV testing environment and the simulation/AV. -Information regarding data accuracy and transfer rate is recorded and analyzed.	-Data format reliability -Data transfer success rate
1.2: AV Testbed shall visualize the environment accurately.	-Generate a virtual environment based on the simulation and AV interaction. -Determine if the AV position in the AV testing environment scene graph is reliable.	-AV position reliability in visualization
AV Testbed Inputs		
2.1: AV Testbed shall be abstracted to handle multiple sensor and vehicle types that can be inputted by the user. These sensor and vehicle types shall have acceptable and realistic parameters.	-Tester selects multiple sensor types. -AV testing environment is expected to instantiate these sensors and associate them with the AV. -Sensor list is printed out and checked against the expected sensor listing.	-Sensor list reliability
2.2: AV Testbed shall receive and store scene simulation information.	-Tester runs a scenario with a chosen simulation. -Scene information is stored internally to AV testing environment.	-Scene object position reliability -Scene update rate

	-Stored scene information is compared to the expected scene information (specifically object position and scene update).	
AV Testbed Outputs		
3.1: AV Testbed shall accurately state the output made by the AV.	-Tester runs multiple scenarios. -AV testing environment sends information to the AV and receives input back from the AV. -Output is compared to the expected output.	-Output reliability
3.2: AV Testbed shall evaluate whether a failure/operational mission failure has occurred and output its result.	-Tester runs scenarios that force collisions between the AV and other vehicles. -AV testing environment outputs the collision information.	-Collision detection rate
3.3: AV Testbed shall report relevant performance measures	-Tester selects metrics. -AV testing environment runs a test scenario with already-known metrics. -AV testing environment generates a metrics report. -Tester compares the generated metrics with the known metrics.	-Proper number of measures are reported -Measures are correct for the scenario

AV Testbed Process Requirements

<u>AV Testing Environment shall accurately represent data input to clients.</u>

class VType

a. This requirement is the need to maintain data format reliability and a demonstrated data transfer success rate. Accurate data will be tested in VAST program source code unit tests, using GoogleTest unit test framework for C++. The data transmitted between VAST system clients (AV and navigation simulation) shall be demonstrated as versatile, but safe types. The base class VType was designed to be the common ancestor of all data types transmitted in the VAST ecosystem.
b. Unit Test
    i. Test_VType

Commented [MP3]: This VType module doesn't seem to follow the format used in Figure 1

1. IsVType - Tests that each extended VType can be stored or moved as a VType, and reflects its type in the type field.
    a. Pass: All objects can be stored and types can be verified.
    b. Fail: An object cannot be stored or a type is not correct.

Get Value - Tests that each extended VType's value can be retrieved through the value() function.

- a. Pass: All objects return their passed value.
- b. Fail: The constructor does not set the value properly in the object.

Set Value - Tests that each extended VType's value can be changed. Also tests that the default constructor generates the default value.

- a. Pass: All objects' values can be modified with value(...) call.

b. Fail: An object value cannot be modified correctly.

Special String Functions - Tests that each special String function operates as predicted. These include a length() accessor and a concat(...) mutator.

- a. Pass: Functions length and concat are implemented correctly.

b. Fail: Functions length and concat are not implemented correctly, and/or a new String is not returned from concat.

Special Double Functions - Tests that each special Double function operates like a double would.

- a. Pass: Resulting doubles are the correct values, and the original Double values are untouched. A Double can be set with these results.

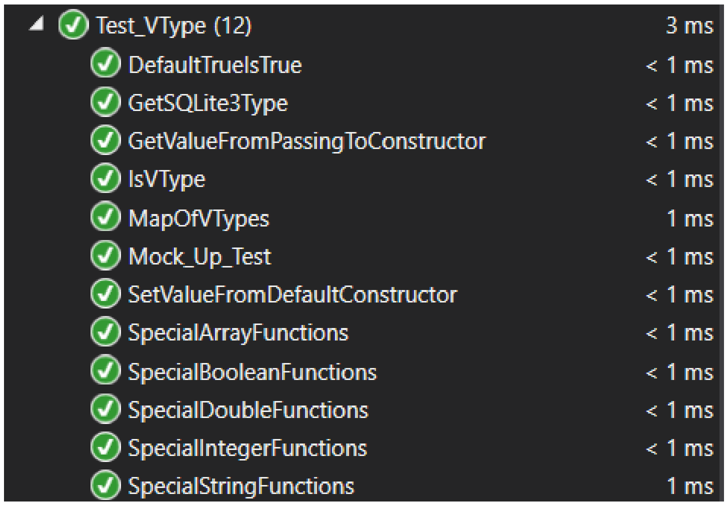
b. Fail: Arithmetic was incorrect or functions are not implemented.

Special Integer Functions - Tests that each special Integer function operates like an integer would.

- a. Pass: For Arithmetic overloads, resulting integers are the correct values, and the original Integer values are untouched. An Integer can be set with these results. For asDouble(), the original integer is still integer, new Double matches double cast value of original integer

b. Fail: Arithmetic was incorrect or functions are not implemented.

2. Completion Date: 3/29/2019
   a. Actual Results:



```
◢ ✓ Test_VType (12)                          3 ms
    ✓ DefaultTrueIsTrue                     < 1 ms
    ✓ GetSQLite3Type                        < 1 ms
    ✓ GetValueFromPassingToConstructor      < 1 ms
    ✓ IsVType                               < 1 ms
    ✓ MapOfVTypes                             1 ms
    ✓ Mock_Up_Test                          < 1 ms
    ✓ SetValueFromDefaultConstructor        < 1 ms
    ✓ SpecialArrayFunctions                 < 1 ms
    ✓ SpecialBooleanFunctions               < 1 ms
    ✓ SpecialDoubleFunctions                < 1 ms
    ✓ SpecialIntegerFunctions               < 1 ms
    ✓ SpecialStringFunctions                  1 ms
```

<u>AV Testing Environment shall visualize the environment accurately</u>

Post-Sim Visualization

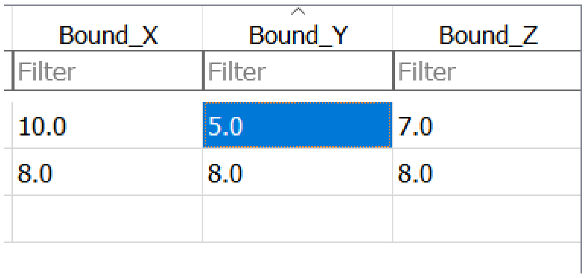
a. This VAST module maintains AV position reliability in the Unity visualization. Upon reading the database, a virtual environment is generated based on the simulation and AV interaction. The objective of this test is to determine if the AV position in the Unity scene graph is reliable.
b. The code is unit tested with multiple test categories and cases that validate that the database is being queried properly.
   i. The test category is to read the number of AVs from database. The database is queried by the code and the count is stored in a variable. The value of this variable is outputted to the debug log.
      1. Test Cases
         a. Test case: 1 AV
            i. Expected outcome: Debug message displaying “1”
            ii. PASS criteria: value of numAVs variable is 1
            iii. FAIL criteria: value of numAVs is a number other than 1
         b. Test case: 2 AVs PASSED

i. Expected outcome: Debug message displaying “2”
ii. PASS criteria: value of numAVs variable is 2
iii. FAIL criteria: value of numAVs is a number other than 2

c. Test case: 3 AVs PASSED
   i. Expected outcome: Debug message displaying “3”
   ii. PASS criteria: value of numAVs variable is 3
   iii. FAIL criteria: value of numAVs is a number other than 3

2. Completion Date: 4/13/2019

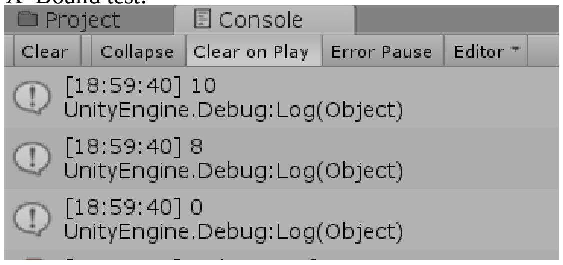
ii. The test category is to read the bounds of each AV. The test procedures are the same for this test category, except the query is slightly modified to read in the entire AV configuration table. The contents of this table will be used to determine the value of multiple other relevant variables. The test also validates that blank fields in the table will not break the code.
   1. Test Cases
      a. X, Y, and Z bounds values are set to the following values:



Bound_X	Bound_Y	Bound_Z
Filter	Filter	Filter
10.0	5.0	7.0
8.0	8.0	8.0

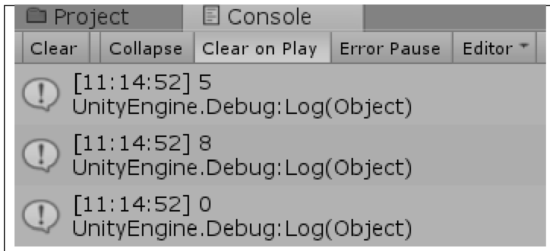
   2. Completion Date: 4/13/2019

X_Bound test:



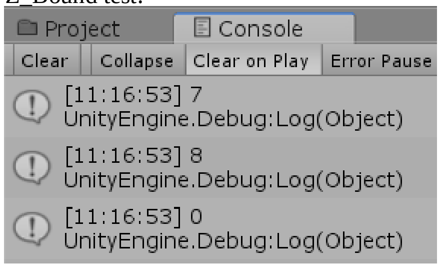
```
[18:59:40] 10
UnityEngine.Debug:Log(Object)
[18:59:40] 8
UnityEngine.Debug:Log(Object)
[18:59:40] 0
UnityEngine.Debug:Log(Object)
```

Y_Bound test:



```
[11:14:52] 5
UnityEngine.Debug:Log(Object)
[11:14:52] 8
UnityEngine.Debug:Log(Object)
[11:14:52] 0
UnityEngine.Debug:Log(Object)
```

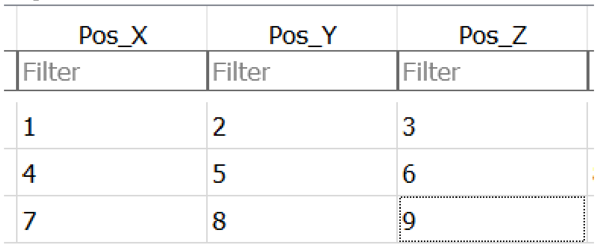
Z_Bound test:



```
[11:16:53] 7
UnityEngine.Debug:Log(Object)
[11:16:53] 8
UnityEngine.Debug:Log(Object)
[11:16:53] 0
UnityEngine.Debug:Log(Object)
```

iii. This test category is to get the positions of AVs (the x, y, and z values).

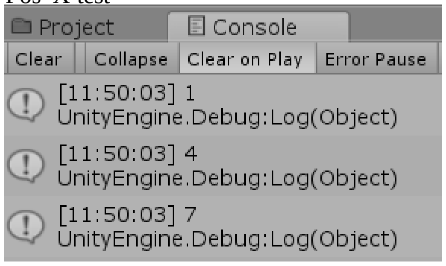
1. Test Cases
   a. Expected Values



Pos_X	Pos_Y	Pos_Z
Filter	Filter	Filter
1	2	3
4	5	6
7	8	9

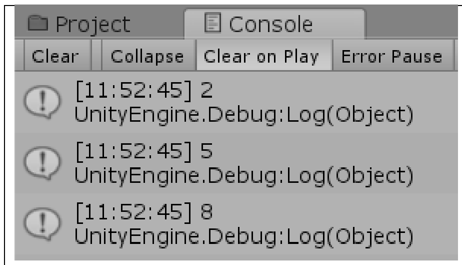
2. Completion Date: 4/13/2019

Pos_X test



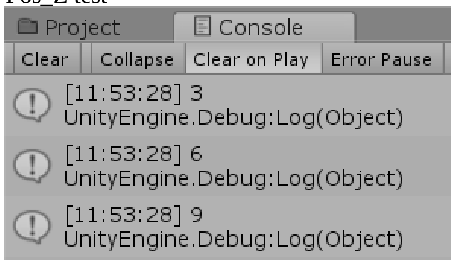
```
[11:50:03] 1
UnityEngine.Debug:Log(Object)
[11:50:03] 4
UnityEngine.Debug:Log(Object)
[11:50:03] 7
UnityEngine.Debug:Log(Object)
```

Pos_Y test


Project Console
Clear Collapse Clear on Play Error Pause

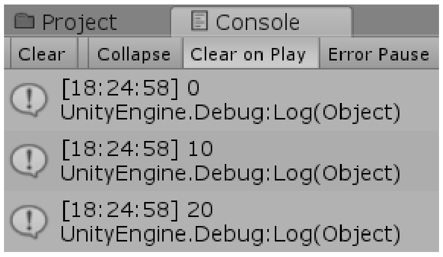
```
[11:52:45] 2
UnityEngine.Debug:Log(Object)
[11:52:45] 5
UnityEngine.Debug:Log(Object)
[11:52:45] 8
UnityEngine.Debug:Log(Object)
```

Pos_Z test


Project Console
Clear Collapse Clear on Play Error Pause

```
[11:53:28] 3
UnityEngine.Debug:Log(Object)
[11:53:28] 6
UnityEngine.Debug:Log(Object)
[11:53:28] 9
UnityEngine.Debug:Log(Object)
```

iv. This test category is to determine the time difference between the vehicles.
   1. Test Cases
      a. Expected Values: 0, 10, 20
   2. Completion Date: 4/13/2019


Project Console
Clear Collapse Clear on Play Error Pause

```
[18:24:58] 0
UnityEngine.Debug:Log(Object)
[18:24:58] 10
UnityEngine.Debug:Log(Object)
[18:24:58] 20
UnityEngine.Debug:Log(Object)
```

SUMO Example Environment Class

a. SUMO determines the AVs and other obstacles' position and geometry. SUMO connects with VAST through TraCI. These are the main functions for SUMO connection.
b. Unit Tests
   i. Testing the main functions for a SUMO connection
      1. Test Cases

         Constructor – This test is testing the constructor of SUMO TraCI Setting up
         a. Pass: SUMO TraCI set up successfully

   b. Fail: SUMO TraCI did not set up sucessfully

Destructor – This tests the destructor of SUMO TraCI Setting up
   a. Pass: Destruct SUMO TraCI successfully
   b. Fail: Failed to destruct SUMO TraCI

Initialize – Tests initialization of SUMO TraCI connection
   a. Pass: Initialize SUMO TraCI connection succesfully
   b. Fail: Initializing SUMO TraCI connection was not successful

RunSumo – Tests that SUMO can run
   a. Pass: SUMO runs successfully
   b. Fail: SUMO cannot run

RunClient – Test Client works correctly
   a. Client works
   b. Client does not work

2. Completion Date: 4/21/2019

AV Testbed Inputs Requirements

<u>AV Testing Environment shall be abstracted to handle multiple sensor and vehicle types that can be inputted by the user. These sensor and vehicle types shall have acceptable and realistic parameters</u>

a. This requirement is the need to maintain Sensor list reliability.
b. Tester selects multiple sensor types. AV testing environment is expected to instantiate these sensors and associate them with the AV. Sensor list is printed out and checked against the expected sensor listing. These are main functions for sensor.
   i. GetSensorType
      1. Procedure: not finish yet
      2. Pass: Get sensor type successfully
      3. Fail: Fail to get sensor type.
   ii. GetDataType
      1. Procedure: not finish yet
      2. Pass: Get data type successfully.
      3. Fail: Get data type unsuccessfully.
c. The associated measures for these tests are passing GoogleTests
d. Completion date: 4/25

Commented [MP4]: This looks like it needs to be updated.

<u>AV Testing Environment shall receive and store scene simulation information</u>

Collision detection module

a. These requirements are the need to maintain Scene object position reliability and Scene update rate

b. Tester runs a scenario with a chosen simulation. Scene information is stored internally to AV testing environment. Stored scene information is compared to the expected scene information (specifically object position and scene update). VAST can add and get obstacle through environment file, and obstacle and environment data will save in database. These are main function about environment construction.

i. Environment Constructor Test
   1. Procedure: Create an Env object and pass dummy parameters into it: vector3{0,0,0} and then 0 for number of obstacles. Check first parameter sets the vector dimensions value. Check second parameter sets the number of obstacles
   2. Pass: Construct environment successfully.
   3. Fail: Construct environment unsuccessfully.

ii. Environment Add and Get Obstacle Test
   1. Procedure: using a combined approach with the add and the get obstacle, test that the obstacle create can be added to the environment object, and then retrieved as the SAME object. Check environment object is same as obstacle.
   2. Pass: Add and get obstacle successfully.
   3. Fail: Add and get obstacle not successfully.

iii. Obstacle Constructor Test
   1. Procedure: Create an Environment object and pass dummy parameters into it: vector3{0,0,0}. Check obstacle parameters are same as sets the vector dimensions value.
   2. Pass: construct obstacle successfully.
   3. Fail: construct obstacle unsuccessfully.

These are the main functions of database:

i. Opendatabase-test program can open database
   1. Procedure: Create a new char name of database, and call open database function, which comes from SQLite API. Check function's result is true or false.
   2. Pass: The result of function is true. Open or create database file successfully.
   3. Fail: The result of function is false. Open or create database file not successfully.

ii. Createtable-test program can create table

1. Procedure: Generate sql statement, and input sql to sqlite3_exec function, which is an SQLite API function. Check result is SQLITE_OK.
2. Pass: The test result is SQLITE_OK, generate table successfully.
3. Fail: The test result isn't SQLITE_OK, generate tables unsuccessfully.

iii. Insertdata-test program can insert data

1. Procedure: Generate sql statement, and input sql to sqlite3_exec function, which is SQLite API function. Check result is SQLITE_OK.
2. Pass: The test result is SQLITE_OK, generate table successfully.
3. Fail: The test result isn't SQLITE_OK, generate tables unsuccessfully.

iv. Showdata -test program can output data

1. Procedure: Generate sql, input sql into showdata function and check result is SQLITE_OK.
2. Pass: The test result isn't SQLITE_OK, generate tables successfully.
3. Fail: The test result isn't SQLITE_OK, generate tables unsuccessfully

c. The associated measures for these tests are passing GoogleTests

d. Completion date: 4/21

AV Testbed Outputs Requirements

<u>AV Testing Environment shall accurately state the output made by the AV</u>

EventTree.cpp

a. This requirement is the need to maintain output reliability

b. Tester runs multiple scenarios. AV testing environment sends information to the AV and receives input back from the AV. Output is compared to the expected output.

i. EventTreeConstructor- Tests that the EventTree constructors properly create an EventTree and throw the InvalidArgumentException where arguments do not follow the proper input requirements.
   1. Procedure: Create EventTrees with improper arguments.
   2. Pass: All expected exceptions were thrown, and all expected allowable inputs did not throw an exception. Getters retrieve constructor inputs again.
   3. Fail: Exceptions that were expected were not thrown, getters returned the wrong results.

ii. EventTreeRegisterComponent - Tests that the EventTree can register the component data maps.
   1. Procedure: create EventTree and verify registration of VComponents..

2. Pass: All expected exceptions were thrown, and all expected allowable inputs did not throw an exception. Getters retrieve constructor inputs again.
3. Fail: Exceptions that were expected were not thrown, getters returned the wrong results.

iii. EventTreeStartAndStopClock-Test the stop clock
1. Procedure: create new EventTree and push the clock forward indirectly, it should stop
2. Pass: stop clock works correctly.
3. Fail: stop clock works uncorrectly

iv. EventTreeFasterThanRealTime-Test EventTree can run fast than real time
1. Procedure: set the timeslice to 0.1 s, ratio smallest it can be set 0.01 (one real second per hundred seconds), run length 1.0 s. the run duration should be 0.01 in real time
2. Pass: EventTree can run fast than real time.
3. Fail: EventTree cannot run fast than real time.

v. EventTreeSeveralComponentsAndEvents-Test EventTree Several Components and Event
1. Procedure: define update maps, test update values. Create event creators, the EventTree can be initialized and can register the components creator.
2. Pass: update maps works correctly.
3. Fail: update maps not work.

Commented [MP5]: Seems to be missing a completion date.

AV Testing Environment shall evaluate whether a failure/operational mission failure has occurred and output its result

- Collision detection publishing to database with proper collision detection rate
  - Needs to report the following to the database: run number, AV ID, time of collision, position of the collision within the environment, object AV collided with
  - Test scenarios
    - Post-simulation example to file (in Unity Editor)
      - Need: Unity demo run
      - Value added: validates the detection of a collision with visualization being rendered
    - Post-simulation example to database (in Unity Application)
      - Need: Unity demo with database
      - Value added: validates the publication to the database

        - VAST generates data and stores data in the database and Post-simulation connect database and output visualization of AV system. This requirement relies on the data of the AV and other vehicles.
        - Generate a virtual environment based on the simulation and AV interaction. Determine if the AV position in the AV testing environment scene graph is reliable.
        -
    - Faster-than-real-time example to file (in Unity Editor)
        - Need: Unity demo without running camera
        - Value added: validates the ability to detect a collision without rendering the visualization
    - Faster-than-real-time example to database (in Unity Application)
        - Procedures: Unity demo without running camera and active database connection
        - Value added: validates the ability to publish collision information without rendering the visualization

Commented [MP6]: This whole section seems out of place. Does it belong somewhere else?

<u>AV Testing Environment shall report relevant performance measures</u>

ScenarioMetric.h / ScenarioMetric.cpp

a. The requirement is that the proper number of measures are reported, and measures are correct for the scenario
b. Tester selects metrics. AV testing environment runs a test scenario with already-known metrics. V testing environment generates a metrics report. Tester compares the generated metrics with the known metrics. This data will be tested in VAST program source code unit tests, using GoogleTest unit test framework for C++.
    I. AverageSpeed-test calculating average speed.
        1. Procedure: Create speed values, 1,2,3 store speed values into Calculate function expect result is 2.
        2. Pass: Average speed calculation works correctly.
        3. Fail: Average speed calculation works not correctly.
    II. MaximumAcceleration-test getting maximum acceleration.
        1. Procedure: Create acceleration values, 1,2,3, shore these into calculate function expect result is 3.
        2. Pass: Maximum Acceleration calculation works correctly.
        3. Fail: Maximum Acceleration calculation works not correctly.
    III. MinimumAcceleration-test getting minimum acceleration.
        1. Procedure: Create acceleration values, 1,2,3, shore these into calculate function expect result is 1.

2. Pass: Minimum Acceleration calculation works correctly.
3. Fail: Minimum Acceleration calculation works not correctly.

IV. AverageAcceleration-test calculating average acceleration.

1. Procedure: Create Acceleration values, 1,2,3 store these into Calculate function expect result is 2.
2. Pass: Calculating average acceleration function works correctly.
3. Fail: Calculating average acceleration function works not correctly.

V. XYZCoordinates-test getting x,y,z coordinates.

1. Procedure: Create coordinates values (1,1,1), Check output is equal (1,1,1)
2. Pass: Get x,y,z coordinates successfully.
3. Fail: Get x,y,z coordinates unsuccessfully.

c. The associated measures for these tests are passing GoogleTests

d. Completion date: 4/21

III. Integration Tests

Commented [OE7]: Needs to include information about proof-of-concept demo (can use paper and user guide as a basis)

The Proof-of-Concept demonstration will be used to validate that the entire system works together properly; thus, it serves as the integration test for VAST. Each component added to Master undergoes an integration test, including the additions of VType and VComponent.

The primary method for controlling a vehicle in SUMO would be to use the MoveToXY command, which would require an update every time-step to continue moving the vehicle. Although using MoveToXY may be feasible, the control methods in other simulation software products are far more intuitive.

Another consideration is that not all navigation simulations have the capability to detect collisions, which is an integral part of training and testing an AV. To account for this, the Proof-of-Concept will also include Unity-driven collision detection.

The final component of the Proof-of-Concept involves the generation of a post-run visualization. This functionality allows users the opportunity to visualize a scenario after it has been run and interact with it: increase and decrease update speed, pause the simulation, rewind and fast-forward one frame, toggle camera views, and increase as well as decrease run number. This is highly useful in cases where the navigation simulation does not support 3D visualization inherently.

Configuration Wizard

-Parser Tests related to instantiating objects

-Configuration could connect with VAST

Basic procedure:

-- Creating a test case for a sample xml file format and adding in information

-- Checking the xml file to make sure it matches the expected output.

Result: The xml file matches the expected output.

<u>AV</u>

-Information should be passed from the Ground AV class to VAST via the Event Tree

-Sensor should report data to AV

-AV can connect with VAST

Basic procedure:

--Construct Event Tree and verify registration of VComponents.

--Check to see if the Unit Tests pass.

--Check sensor work well

Actual Output: TBD

Result: TBD

<u>VAST database system</u>

-Sent data to database

-Get different value type

Basic procedure:

--Construct Database and check functionality of database operation

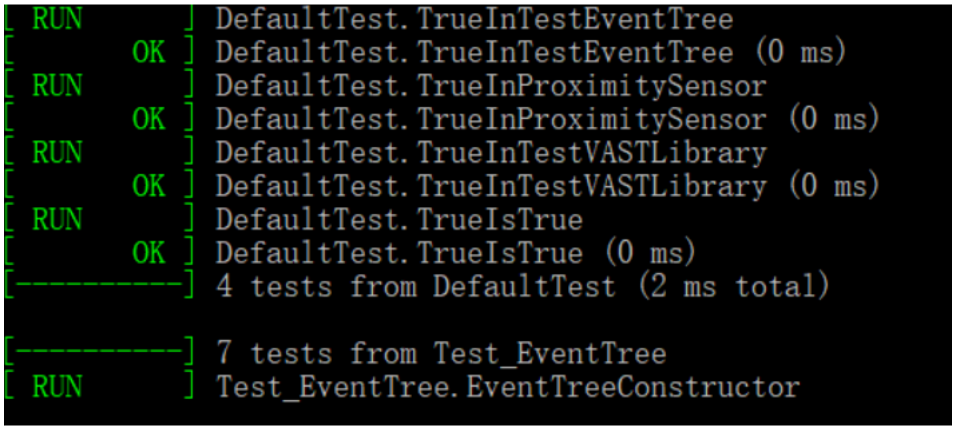
--Check Vtype work well

--Check VAST library work well.

--Check EventTree work well

Actual Output:

Result: Test failure.


```
[ RUN      ] DefaultTest.TrueInTestEventTree
[       OK ] DefaultTest.TrueInTestEventTree (0 ms)
[ RUN      ] DefaultTest.TrueInProximitySensor
[       OK ] DefaultTest.TrueInProximitySensor (0 ms)
[ RUN      ] DefaultTest.TrueInTestVASTLibrary
[       OK ] DefaultTest.TrueInTestVASTLibrary (0 ms)
[ RUN      ] DefaultTest.TrueIsTrue
[       OK ] DefaultTest.TrueIsTrue (0 ms)
[----------] 4 tests from DefaultTest (2 ms total)

[----------] 7 tests from Test_EventTree
[ RUN      ] Test_EventTree.EventTreeConstructor
```

Environment

-Information should be passed from the SUMO Environment class to VAST via the Event Tree

-Environment could connect with VAST

Basic procedure:

--Set up Traci and connect with SUMO

--Check Unit Test whether to pass

Actual Output:

Pass Unit Test

Result: Test passed.

Commented [MP8]: This doesn't seem complete

Collision Detection

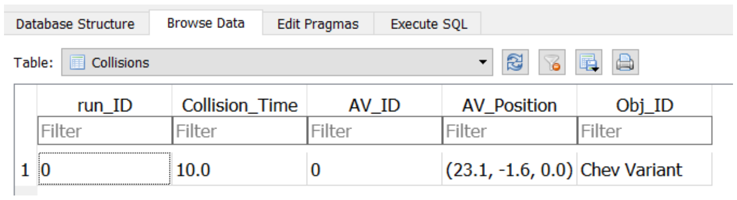
-Collisions need to be properly reported to database

Basic procedure:

--Simulate collision

--Check that all data report is correct

Actual Output:



Database Structure | Browse Data | Edit Pragmas | Execute SQL

Table: Collisions

	run_ID	Collision_Time	AV_ID	AV_Position	Obj_ID
	Filter	Filter	Filter	Filter	Filter
1	0	10.0	0	(23.1, -1.6, 0.0)	Chev Variant

Result: Test passed.

<u>Visualization</u>

-Visualization needs to be properly launched

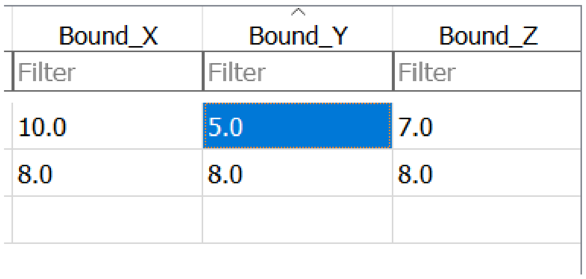
-Visualization could communicate with VAST

Basic procedure:

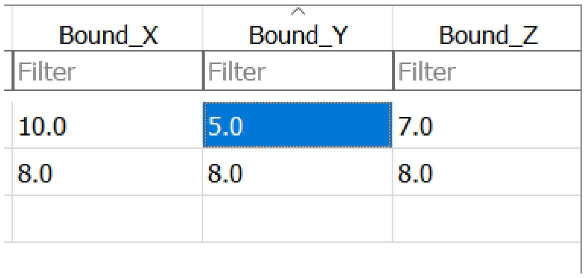
-- Reading the database, generated Unity graph based on the simulation and AV interaction.

--Check the output is same as database.

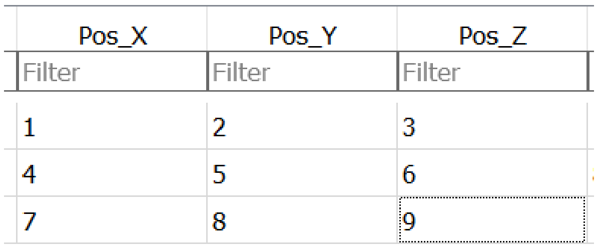
Actual Output:



Bound_X	Bound_Y	Bound_Z
Filter	Filter	Filter
10.0	5.0	7.0
8.0	8.0	8.0



Bound_X	Bound_Y	Bound_Z
Filter	Filter	Filter
10.0	5.0	7.0
8.0	8.0	8.0



Pos_X	Pos_Y	Pos_Z
Filter	Filter	Filter
1	2	3
4	5	6
7	8	9

Result: Test passed.

IV. Limitations to Tests

Overall, there will be limitations to testing VAST. The major limitation is the fact that this test does not include hardware-in-the-loop, which means that this validation is purely on the software. Even if this software is fully validated, new problems are likely to arise once both hardware and software are tested in conjunction. Specifically, in defense acquisition, multiple test events are conducted. In an Operational Test (OT) report from Fiscal Year (FY) 17, 64 OTs were conducted with Director, Operational Test and Evaluation (DOT&E) oversight. 34.4% OTs conducted in FY17 discovered critical, new problems in addition to old problems. 15.7% discovered only new, critical problems [1].

Additionally, there is no way a system can account for every scenario without testing how the system of systems (SOS) integrates with the system under test (SUT). This includes how the hardware interacts with the software (impossible to test the hardware) along with other simulation types.

V. References

[1] Problem Discovery Affecting OT&E