



WHITEPAPER

# Innoslate/STK/MATLAB Co-Simulation

# Introduction

This guide explains how to co-simulate STK and MATLAB® with Innoslate to refine design engineering results through operational scenario models by following an operational model of a lunar rover.

Ansys Systems Tool Kit (STK) allows you to model complex systems in a realistic mission context using a dynamic physics-based simulation environment. STK is integrated with Innoslate's Action Diagram to improve the fidelity of the model by passing values, such as the time duration of dynamic STK objects, from STK to Innoslate.

MathWorks® MATLAB® utilizes math, graphics, and programming together to allow users to analyze data, develop algorithms, and create models. MATLAB® is integrated with Innoslate's Action Diagram to improve the model simulation by passing values between MATLAB® and Innoslate to incorporate MATLAB® functions and results in the simulation.

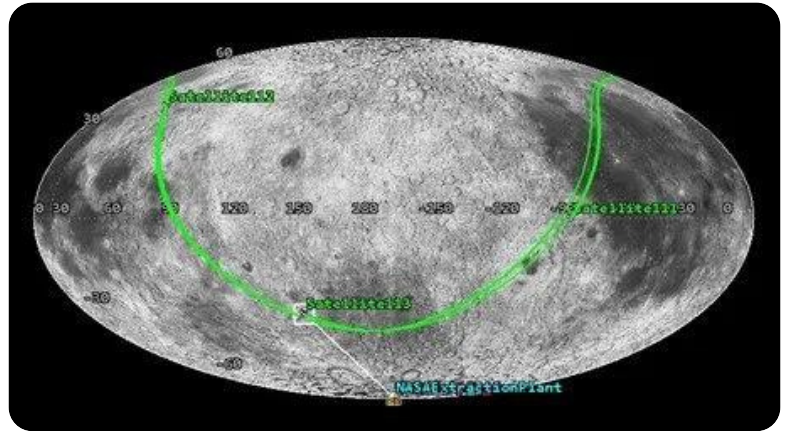
The following section, "Co-Simulate STK Models in Innoslate," will discuss the process of integrating Innoslate with STK, and "Co-Simulate MATLAB Functions in Innoslate" will discuss the process of integrating Innoslate with MATLAB®. A co-simulation using STK, MATLAB®, and Innoslate's Action Diagram was performed during the Lunar Rover project to simulate the lunar rover's mission on the lunar surface to excavate icy regolith and deliver extracted water for 365 days or until the mission goal of collecting 10,000 kg of water is reached.

# Co-Simulate STK Models in Innoslate

Models created in STK were integrated with Innoslate through Action diagrams. Once simulated, the results produced in Innoslate provided a more detailed analysis of the lunar rover's mission on the surface of the Moon. The following sections describe the processes conducted to integrate STK models into Innoslate simulations for the Lunar Rover project.

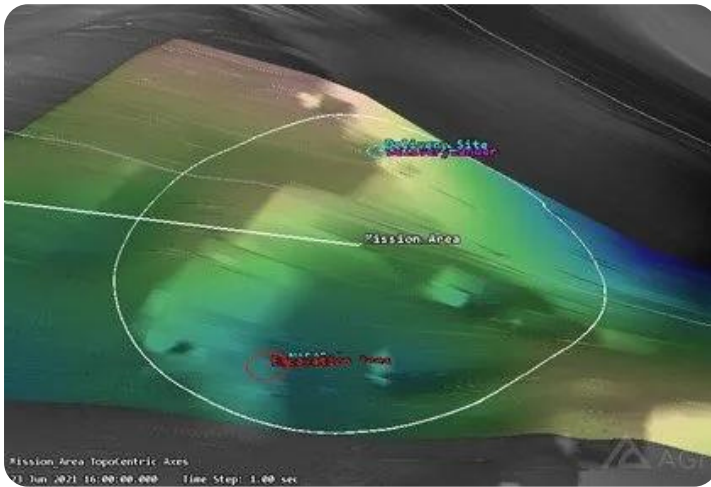
## Create a Model in STK

A model was built in STK to represent the lunar mission environment. Each aspect of the mission had to be instantiated as various objects in STK. The Moon was represented by declaring it as a central body in the STK application window. The NASA site locations (Extraction Plant, Excavation, and Delivery Sites) were plotted using the lunar latitude and longitude coordinates provided by the Break the Ice Challenge. Communication satellites and infrastructure around the Moon were also added to illustrate contact with mission control on Earth.



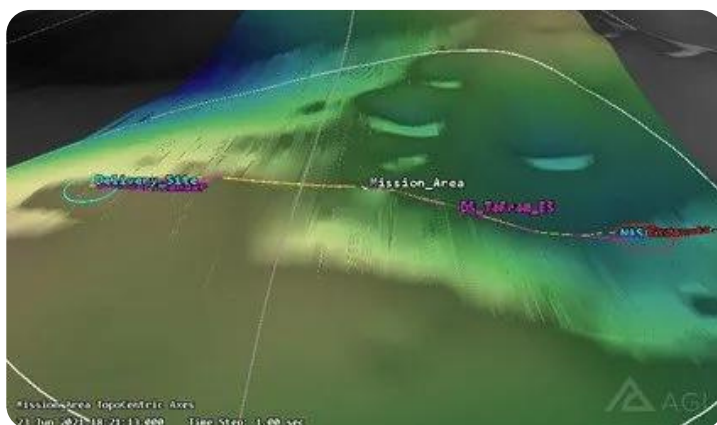
STK Satellite and Location Mapping

The fidelity of the STK model was increased by importing publicly available lunar terrain information for the South Pole. This data was obtained from the NASA Lunar Reconnaissance Orbiter (LRO) missions. By incorporating lunar terrain details, visuals for the terrain elevation, obstacles, and cratered regions were incorporated in the mission area of the STK model. The figure on the following page is the resulting high-fidelity STK model.

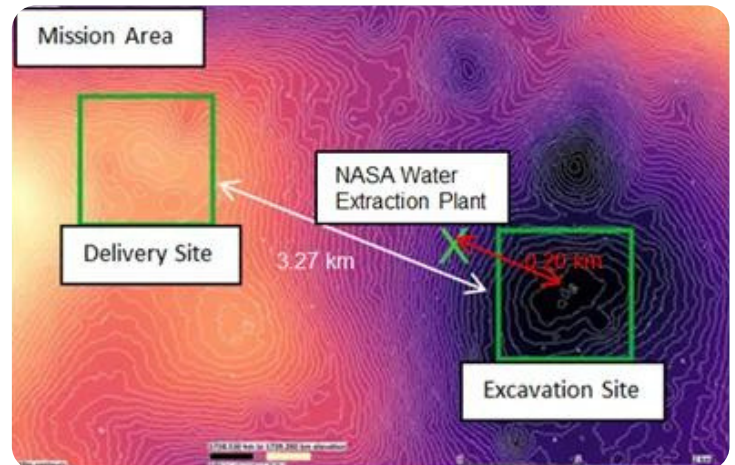


STK Lunar Environment Mapping

Viable travel routes for the lunar rover to follow while navigating between the mission sites were then visually plotted. This was done while considering the rover's terrain incline limitations; it was nominally determined to plot a route with inclines less than 30 degrees. The travel routes were verified using topography maps provided by the Break the Ice Challenge.



STK Model



Break the Ice Challenge Topography Map

Lastly, a constant velocity was defined for the rover by declaring an STK Ground Vehicle object. This speed is later used to calculate the lunar rover's travel time values.

## Create an Operational Scenario in Innoslate

An operational scenario was created in Innoslate using Action diagrams. The model below describes the lunar rover's mission on the lunar surface, which involves excavating icy regolith and delivering extracted water for 365 days or until the mission goal of collecting 10,000 kg of water is achieved. Each Action in the diagram represents a specific process or capability the rover will perform.

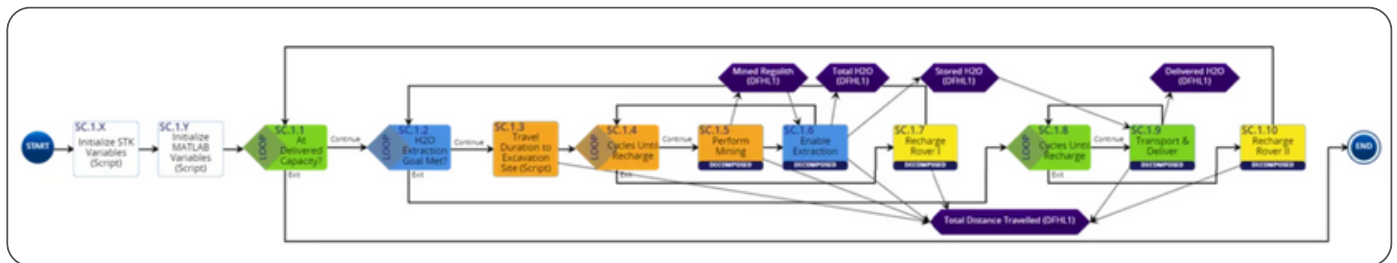


The operational scenario is described from the beginning of the mission, when the rover lands on the lunar surface, to the end of the mission, when the goal of collecting 10,000 kg of water is achieved or 365 days have passed, whichever occurs first. In this model, there is one rover on the surface of the Moon that is responsible for excavating regolith, transporting regolith to the extraction plant, and delivering the extracted water to the designated delivery site.

At the start of the scenario, the rover performs multiple cycles of excavating regolith. Once its storage container has reached maximum capacity, the rover then delivers the collected regolith to the NASA Water Extraction Plant. After

the plant extracts water from the regolith, the second phase of operation begins. The rover collects the water from the Extraction Plant and transports it to the Delivery Site. If the rover's battery does not need to be charged or the equipment warmed up, then the lunar rover will return to the Excavation Site and repeat these processes.

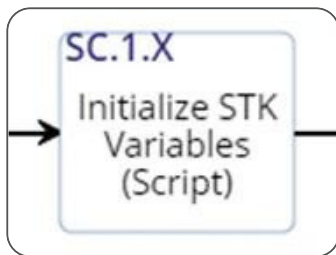
The rover system will continue these actions until the mission goal of delivering 10,000 kg of water has been met or 365 Earth days have passed on the Moon. At the end of the mission, the lunar rover deactivates and retires.



One Rover Asymmetrical Sequence Scenario

## Use Innoslate/STK Integration API

These two models from STK and Innoslate were then combined to create a cohesive co-simulation of the mission events.



STK Initialization  
Action Entity

A new Action entity, "Initialize STK Scenario Variables", was created and added to the beginning of the operational scenario Action diagram in Innoslate. This entity serves as an "initialization" block to hold Innoslate/STK API scripting. The scripts added to the initialization block serve to initialize and run the STK model and create global variables for storing data acquired from STK in Innoslate. Duration and velocity vector components were calculated using a combination of Innoslate/STK Javascript methods with the Ground Vehicle object acting as the rover in STK. NOTE: Units and unit conversions must be handled carefully, as the user must

be aware of the units that output from STK (STK Connect default units). STK automatically uses default values for each dimension, such as time, distance, and velocity (seconds, meters, and meters/seconds); however, in Innoslate, it is the user's responsibility to declare units and perform any unit conversion as needed. This is done within the Action block using scripting.

The figure on the following page is a screenshot of the scripting used to calculate the duration components, e.g., start & end times in milliseconds. This data was extracted from the STK model, and it represents the time the rover takes to traverse the lunar rover route from the Excavation Site to the Water Extraction Plant to the Delivery Site. Duration values were then calculated from the duration components (i.e., subtracting start times from end times in milliseconds), and each was assigned to a unique global variable that can be recalled later on in the simulation.

```

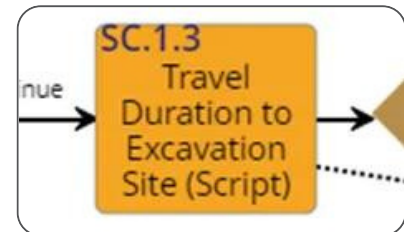
</ Edit Script
Script
1 = function onStart(){
2   STK.initialize("http://localhost:8080/integrations/STKIntegration?", "C:\\Users\\kayla\\Documents\\STK
3
4   var tempStart = 0;
5   var tempEnd = 0;
6
7   //-----Route 1-----
8   //STK values to get to Extraction Plant from Delivery Site
9   tempEnd = STK.getTimePeriod("DS_ToFrom_EP").end_Time.millisecond;
10  tempStart = STK.getTimePeriod("DS_ToFrom_EP").start_Time.millisecond;
11  var driveExtractFromDeliver = (tempEnd - tempStart)/1000; //convert milliseconds to seconds
12  print("STK - Time to drive to Extraction Plant from Delivery Site (minutes): " + driveExtractFromDel
13  globals.put("driveExtractFromDeliver", driveExtractFromDeliver);
14
15  //-----Route 2-----
16  //STK values to get to Delivery Site from Extraction Plant
17  tempEnd = STK.getTimePeriod("DS_ToFrom_EP").end_Time.millisecond;
18  tempStart = STK.getTimePeriod("DS_ToFrom_EP").start_Time.millisecond;
19  var driveDeliverFromExtract = (tempEnd - tempStart)/1000;
20  print("STK - Time to drive to Delivery Site from Extraction Plant (minutes): " + driveDeliverFromExtr
21  globals.put("driveDeliverFromExtract", driveDeliverFromExtract);
22
23  //-----Route 3-----
24  //STK values to get to Excavation Site from Delivery Site
25  tempEnd = STK.getTimePeriod("DS_ToFrom_ES").end_Time.millisecond;
26  tempStart = STK.getTimePeriod("DS_ToFrom_ES").start_Time.millisecond;
27  var driveExcavateFromDeliver = (tempEnd - tempStart)/1000;
28  print("STK - Time to drive to Excavation Site from Delivery Site (minutes): " + driveExcavateFromDeli
29  globals.put("driveExcavateFromDeliver", driveExcavateFromDeliver);
30
31  //-----Route 4-----
32  //STK values to get to Delivery Site from Excavation Site
33  tempEnd = STK.getTimePeriod("DS_ToFrom_ES").end_Time.millisecond;
34  tempStart = STK.getTimePeriod("DS_ToFrom_ES").start_Time.millisecond;
35  var driveDeliverFromExcavate = (tempEnd - tempStart)/1000;
36  print("STK - Time to drive to Delivery Site from Excavation Site (minutes): " + driveDeliverFromExcav
37  globals.put("driveDeliverFromExcavate", driveDeliverFromExcavate);
38
39  //-----Route 5-----
40  //STK values to get to Extraction Plant from Excavation Site
41

```

Script to Enable Innoslate/STK Initialization

Additional scripting was then added to all the Action entities in the Innoslate operational scenario that refer to the rover travel time. This was done by manipulating the Innoslate scripting API to assign the time duration values to equal the global variables that were created for the STK duration components.

An example of an Action entity that required additional scripting to enable realistic process times, "Travel Duration to Excavation Site," is shown to the right.



"Travel Duration to Excavation Site" Action Entity

In the screenshot of the script, the Innoslate Simulator API was used to assign duration values derived from STK, via the Innoslate global variables, to the respective Action Entity.

```

</ Edit Script
Script
1 = function onStart(){
2   STK.initialize("http://localhost:8080/integrations/STKIntegration?", "C:\\Users\\Perseus\\Documents\\STK
3   //Grab Action entity using local ID from self
4   var driveExcavateFromDeliver = globals.get("driveExcavateFromDeliver");
5   //Perform attribute manipulation of the self Action entity for 'Duration' (Innoslate API)
6   var entity = Sim.getEntityById(56);
7   if(entity){
8     var newDur = entity.attributes().get("Duration");
9     newDur.value = driveExcavateFromDeliver;
10    newDur.units = "SECONDS";
11  }
12
13

```

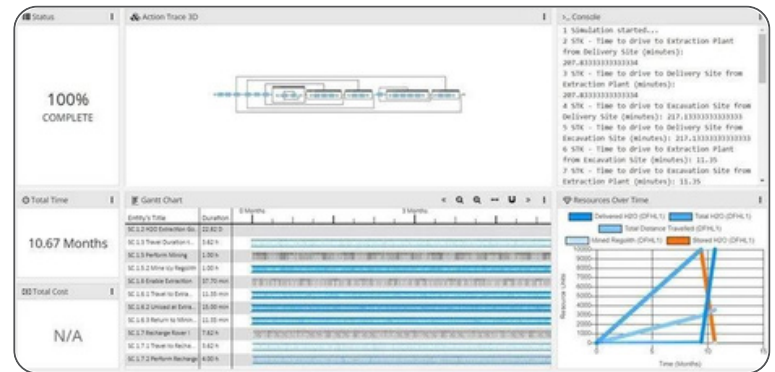
Script to Enable STK Time Duration Values in Innoslate

## Interpret Innoslate/STK Co-Simulation Results

Before the operational scenario Action diagram was simulated, the following assumptions were defined:

- The Delivery Site holds up to 10,000 kg of water for long-term storage.
- The rover travels the same route between the three mission sites for every excavation (Excavation Site to Water Extraction Plant to Delivery Site back to Excavation Site).
- The rover has a maximum load capacity of 100 kg.
- The rover has an average velocity of 0.3 m/s.
- The rover has an excavation rate of 100 kg of regolith/hr.
- The rover charges its battery at the Delivery Site.
- The rover can nominally perform 10 excavation cycles before having to recharge.
- The rover can nominally perform 20 delivery cycles before having to recharge.
- The rover has a charge time of 4 hours.
- All unloading processes nominally take 15 minutes to complete.
- All loading processes nominally take 1 hour to complete.

Once the Innoslate/STK API scripts were added to the Action diagram and the assumptions stated, the operational scenario simulator was executed. The figure below displays the Innoslate/STK co-simulation results.



Innoslate-STK Co-Simulation Results

From the co-simulation of the Innoslate Action diagram and STK model, it was calculated that the mission will take a total time of 10.67 months to reach the mission goal of collecting 10,000 kg of water within 365 Earth days.



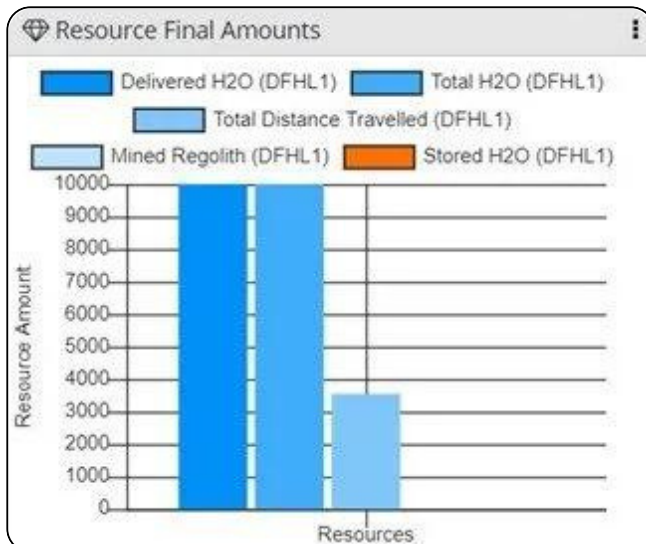


Resources Over Time Simulation Results

By tracking the resources used throughout Innoslate's model execution, see the 'Resources Over Time' panel, it was concluded a majority of the rover's mission time will be dedicated to the excavation and extraction processes.

Looking at the final state of the resources, 10,000 kg of water was successfully extracted and delivered by the end of the mission. In addition, the total distance traveled by the rover. This value has great implications for rover performance, maintenance, and reliability characteristics, especially since no rover has ever traveled this distance in the past.

These simulation results help determine early on whether the lunar rover design is feasible to complete its mission given the constraints provided by the NASA Break the Ice Challenge.



Resource Final Amounts Simulation Results

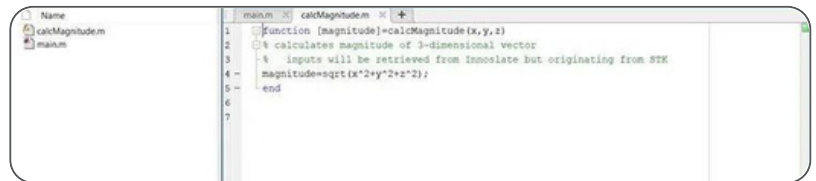
# Co-Simulate MATLAB Functions in Innoslate

Functions created in MATLAB were integrated with Innoslate through Action diagrams. Once simulated, the results produced in Innoslate provided a more detailed analysis of the lunar rover's mission on the surface of the Moon. The following sections describe the processes conducted to integrate MATLAB functions into Innoslate simulations for the Lunar Rover project.

## Create a Function in MATLAB

Once the two models created in Innoslate and STK were co-simulated, MATLAB was used to verify the calculations. Velocity vectors in the X, Y, and Z planes were retrieved from STK through Innoslate and used to calculate a magnitude value representing the rover's velocity.

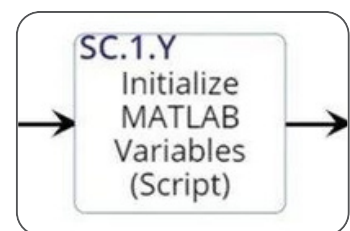
The Figure to the right displays the function script written in MATLAB to compute the magnitude of velocity given the three input parameters, the velocity vectors in the X, Y, and Z planes.



MATLAB Function Script

## Use Innoslate/MATLAB Integration API

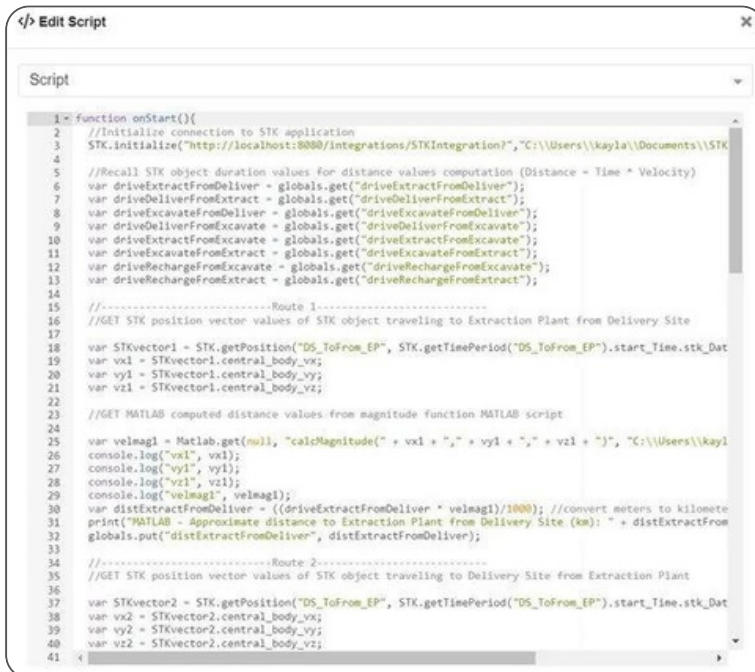
After the function script was written in MATLAB, MATLAB integration APIs were added to the Innoslate operational scenario that had been previously created. This was done using scripting in another initialization Action entity, "Initialize MATLAB Variables", that was added to the beginning of the Action diagram.



MATLAB Initialization Action Entity

Within the MATLAB initialization Action entity, the velocity vector components were extracted from the STK model in the X, Y, and Z planes relative to their central body, the Moon, at the initial start time.

Then, they were used to calculate a velocity magnitude in MATLAB via a GET request. The distance value was calculated by multiplying the time duration by the velocity magnitude and assigned to a unique global variable for later use.



```

1= function onStart(){
2 //Initialize connection to STK application
3 STK.initialize("http://localhost:8080/integrations/STKIntegration?", "C:\\Users\\kayla\\Documents\\STK
4
5 //Recall STK object duration values for distance values computation (Distance = Time * Velocity)
6 var driveExtractFromDeliver = globals.get("driveExtractFromDeliver");
7 var driveDeliverFromExtract = globals.get("driveDeliverFromExtract");
8 var driveExcavateFromDeliver = globals.get("driveExcavateFromDeliver");
9 var driveDeliverFromExcavate = globals.get("driveDeliverFromExcavate");
10 var driveExtractFromExcavate = globals.get("driveExtractFromExcavate");
11 var driveExcavateFromExtract = globals.get("driveExcavateFromExtract");
12 var driveRechargeFromExcavate = globals.get("driveRechargeFromExcavate");
13 var driveRechargeFromExtract = globals.get("driveRechargeFromExtract");
14
15 //-----Route 1-----
16 //GET STK position vector values of STK object traveling to Extraction Plant from Delivery Site
17 var STKvector1 = STK.getPosition("DS_ToFrom_EP", STK.getTimePeriod("DS_ToFrom_EP").start_Time.stk_Dat
18 var vx1 = STKvector1.central_body_vx;
19 var vy1 = STKvector1.central_body_vy;
20 var vz1 = STKvector1.central_body_vz;
21
22 //GET MATLAB computed distance values from magnitude function MATLAB script
23
24 var velmag1 = Matlab.get(null, "calcMagnitude(" + vx1 + "," + vy1 + "," + vz1 + ")", "C:\\Users\\kayl
25 console.log("vx1", vx1);
26 console.log("vy1", vy1);
27 console.log("vz1", vz1);
28 console.log("velmag1", velmag1);
29 var distExtractFromDeliver = ((driveExtractFromDeliver * velmag1)/1000); //convert meters to kilomete
30 print("MATLAB - Approximate distance to Extraction Plant from Delivery Site (km): " + distExtractFrom
31 globals.put("distExtractFromDeliver", distExtractFromDeliver);
32
33 //-----Route 2-----
34 //GET STK position vector values of STK object traveling to Delivery Site from Extraction Plant
35
36 var STKvector2 = STK.getPosition("DS_ToFrom_EP", STK.getTimePeriod("DS_ToFrom_EP").start_Time.stk_Dat
37 var vx2 = STKvector2.central_body_vx;
38 var vy2 = STKvector2.central_body_vy;
39 var vz2 = STKvector2.central_body_vz;
40
41

```

Script to Enable Innoslate/ MATLAB Initialization

Once the MATLAB scripts were added to the MATLAB initialization Action entity, additional scripting was also added to all the Action entities within the Action diagram that refer to rover travel durations. This is done to continuously record and update the total distance the rover has traveled on the surface of the Moon. A Resource entity, "Total Distance Travelled", served as a

counter to compute the total distance the rover traveled during its mission.



"Total Distance Traveled" Resource Entity

The figure on the next page displays the script used for the Action entity "Travel Duration to Excavation Site". Within the script, the global variable containing the distance value is retrieved and manually added to the Resource entity, "Total Distance Travelled", to update the current total distance traveled by the lunar rover.



```

1= function onStart(){
2 STK.initialize("http://localhost:8080/integrations/STKIntegration?", "C:\\Users\\Perseus\\Documents\\STK
3 //Grab Action entity using local ID from so4
4 var driveExcavateFromDeliver = globals.get("driveExcavateFromDeliver");
5 //Perform attribute manipulation of the self Action entity for "Duration" (Innoslate API)
6 var entity = Sim.getEntityById(360);
7 if(entity){
8 var newDur = entity.attributes().get("Duration");
9 newDur.value = driveExcavateFromDeliver;
10 newDur.units = "SECONDS";
11
12 //Perform Resource entity manipulation to add to current value (Innoslate API)
13 var addDist = globals.get("distExcavateFromDeliver");
14 var currDist = Sim.getAmountById(501);
15 Sim.setResourceById(511, currDist + addDist);
16
17

```

Script to Enable Total Distance Traveled Computation

Once the MATLAB scripting was added, the final step was to enable the Innoslate/MATLAB integration by attaching the current location of the MATLAB Java Web Application (JWA) to the MATLAB Integration URL

field (e.g.,  
“http://localhost:8080/integrations/  
MatlabServlet”) found in the  
Innoslate simulation settings, shown  
below. Without it, the simulation will  
not properly execute the  
Innoslate/MATLAB integration APIs.

Settings

Display Mode

Light

Calendar Mode

False

Speed

10000x

Decisions

Prompt on no script

Start

Normal

Hours per year

Random seed

MatLab Integration URL

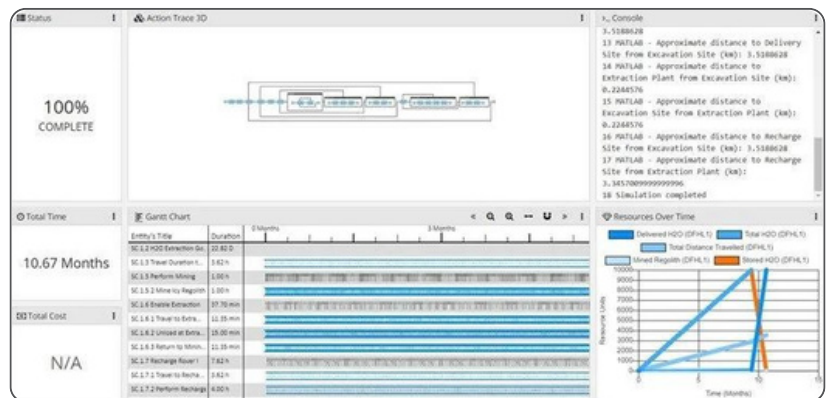
http://localhost:8080/integrations/MatlabServlet

Reset Settings

Save Settings

## Interpret Innoslate/MATLAB Co-Simulation Results

After completing all necessary scripting and enabling the Innoslate/MATLAB integration APIs, the operational scenario Action diagram was executed in the simulator. During simulation, MATLAB calculated distance values with the help of Innoslate and STK, allowing for a realistic estimate of the total rover travel distance. The figure below is a screenshot of the Innoslate/MATLAB/STK co-simulation results and a screenshot of the real-time console output of the MATLAB calculations.



Innoslate Simulation Results – MATLAB

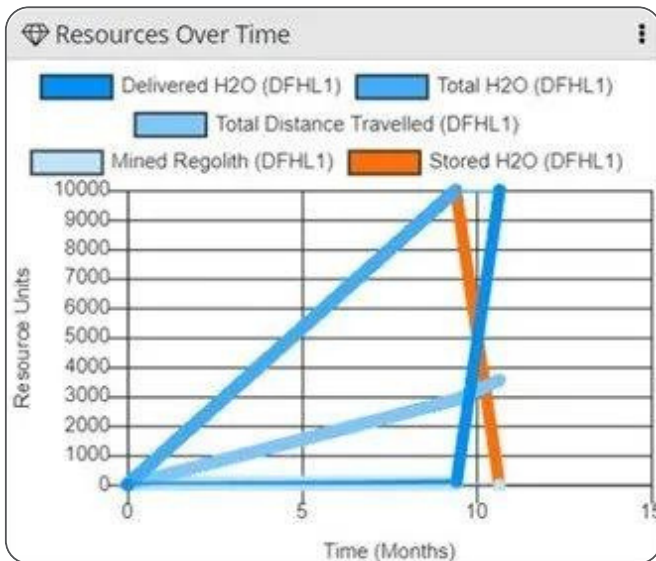
```

>_ Console
3.5188628
13 MATLAB - Approximate distance to Delivery
Site from Excavation Site (km): 3.5188628
14 MATLAB - Approximate distance to
Extraction Plant from Excavation Site (km):
0.2244576
15 MATLAB - Approximate distance to
Excavation Site from Extraction Plant (km):
0.2244576
16 MATLAB - Approximate distance to Recharge
Site from Excavation Site (km): 3.5188628
17 MATLAB - Approximate distance to Recharge
Site from Extraction Plant (km):
3.3457009999999996
18 Simulation completed
  
```

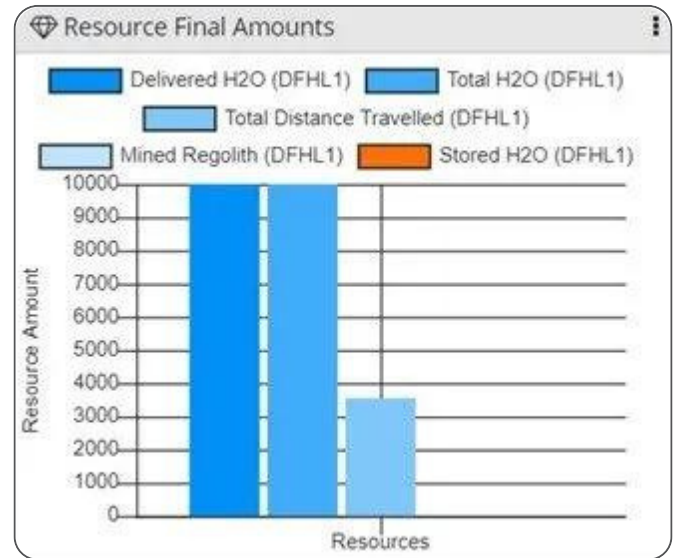
MATLAB Simulation  
Results Console



From the co-simulation of the Innoslate Action diagram and MATLAB function, it was calculated that the mission will take a total time of 10.67 months to reach the mission goal of collecting 10,000 kg of water within 365 Earth days. These results verify the conclusions calculated during the Innoslate/STK co-simulation.



Resources Over Time Simulation Results



Resource Total Amounts Simulation Results

Observing the "Total Distance Travelled" Resource entity in the Resources Over Time and Resource Final Amounts graphs, it was concluded that the total distance traveled reached roughly 3,500 km. This value has great implications for rover performance, maintenance, and reliability characteristics, especially since no rover has ever traveled this distance in past missions.

These simulation results from MATLAB help consider potential expected and unexpected system behaviors during the mission, which can be used to further refine requirements, confirm system solutions, and aid risk management throughout the project.



# Summarize Co-Simulation Results

To summarize the results of the Innoslate/STK/MATLAB co-simulation:

- STK provides accurate times the simulated rover travels to navigate between the various sites on the surface of the Moon.
- Innoslate's Action diagram adds additional processes such as communication, maintenance, operator-in-the-loop, and expected failure conditions.
- JavaScripting in the Innoslate Action entities linked together the STK model with the Action diagram.
- When the Innoslate Action diagram is simulated, global variables are used to retrieve data from the STK model, and they are traced throughout the diagram's execution to calculate the time to collect 10,000 kg of water and the total distance traveled during that mission.
- Functions in MATLAB can also be implemented in the Action entities via JavaScripting to execute with the Innoslate Action diagram. This co-simulation verifies the Innoslate/STK co-simulation if the same results are calculated.
- Performance characteristics can be manipulated to run multiple scenarios using this co-simulation process. In the STK model, change the speed of the rover or modify the route the rover follows to complete its mission, then run the simulation again in Innoslate to determine the impact on mission results.

