Using MATLAB for STK Automation

Problem Statement

In this exercise you will be using MATLAB to automate STK for a hypothetical MQ-1 Predator training mission in the vicinity of Mt. St. Helens. In the training mission, the flight route will remain the same but the target locations will be initially unknown, or "random". The SAR Radar and FLIR sensors used for surveillance and reconnaissance will be modeled with a single simple conic sensor here, and the exercise will go over some basic scenario generation and automation approaches using both the STK Object Model and Connect.

Solution

Write a script in MATLAB using both Connect and Object Model that automates the following functions:

- Create a scenario using MATLAB
- Generate a facility, satellite and great arc aircraft
- Add terrain and imagery
- Generate random targets in an automated fashion
- Point a sensor at the random targets based on minimum range
- Compute chain access between the target, aircraft, satellite and facility
- Set the camera in the 3D Graphics Window to view the mission

Note: A completed script is available in the STK install here:

<STK install folder>/CodeSamples/Automation/Matlab/STK_Automation_Tutorial.m

Create a MATLAB Script

Begin by creating a MATLAB Script which you'll be using to generate, populate and programmatically customize your STK scenario. The script file can be saved and ran over and over again for the varying generated results given the purposes of mission planning and demonstration.

1. Open MATLAB (any version, 32 or 64 bit).
2. Click the "New Script" button on the Home tab.
3. In the Editor tab, save your script file as a MATLAB Code File (*.m) named PredatorMission in a location where you can find it later (e.g. Desktop).
4. At the top of your script on line 1, type `clc, clear` so that with each new run of the script your command window will be cleared of previous entries.

Using the Object Model to Open STK and Create a Scenario

Now that you have your script file open and ready to be edited, we can begin by initializing the STK application and creating a new scenario.

1. Type the following code in as shown, each line below as its own respective line in the MATLAB script. These lines will initialize and launch STK.
```
app = actxserver('STK11.application');
root = app.Personality2;
```
2. For clarity and ease of editing it is recommended that you type %% for each segment of code.
3. Type the code below to create a new scenario with a custom time period. Remember to note that the ending of a single line is denoted by the suppressor semicolon (;) and that, if copying and pasting the code, a proper line of code should be formatted to remain on a single line.

```
scenario = root.Children.New('eScenario','MATLAB_PredatorMission');
scenario.SetTimePeriod('24 Feb 2012 16:00:00.000','25 Feb 2012 16:00:00.000');
scenario.StartTime = '24 Feb 2012 16:00:00.000';
scenario.StopTime = '25 Feb 2012 16:00:00.000';
root.ExecuteCommand('Animate * Reset');
```

Run the Script

With your first lines of code implemented, you can test to see if the MATLAB run script will actually open STK and create a scenario.

1. With the MATLAB script open on the Editor tab, click the “Run” button and either minimize MATLAB or move it to the side a bit.
2. Wait for a moment, and STK should open and generate a scenario with all of our defined properties so far.
3. Close STK and click “No” when prompted if you want to save the changes. Don’t worry about saving the scenarios themselves, because you have the script to generate whole new ones any time you want.
4. If minimized or moved away, bring MATLAB to the front again.

Instantiating Objects and Defining Object Properties

The scenario is going to use a variety of objects using the STK Object Model.

1. In a new section of code (by using %%), type or copy the following lines of code to instantiate a facility and satellite.
```
facility = scenario.Children.New('eFacility','GroundStation');
facility.Position.AssignGeodetic(36.1457,-114.5946,0);
satellite = scenario.Children.New('eSatellite','GeoSat');
```
2. In addition to simply creating default objects, the Object Model allows for the editing of properties. For this mission we want to model a notional geostationary satellite called “GeoSat”. You can edit the propagator, the properties of orbit, and finally propagate the new orbit by typing in the lines below.

```
keplerian = satellite.Propagator.InitialState.Representation.ConvertTo('eOrbitStateClassical');
keplerian.SizeShapeType = 'eSizeShapeAltitude';
keplerian.LocationType = 'eLocationTrueAnomaly';
keplerian.Orientation.AscNodeType = 'eAscNodeLAN';
keplerian.SizeShape.PerigeeAltitude = 35788.1;
keplerian.SizeShape.ApogeeAltitude = 35788.1;
keplerian.Orientation.Inclination = 0;
keplerian.Orientation.ArgOfPerigee = 0;
keplerian.Orientation.AscNode.Value = 245;
keplerian.Location.Value = 180;
satellite.Propagator.InitialState.Representation.Assign(keplerian);
satellite.Propagator.Propagate;
```

3. Save your script and Run the file. Are the desired objects now implemented in our STK scenario generation process? Remember, if you run into any errors upon trying to run the code, check the Command Window in MATLAB to see what the problem may and be sure that the lines above are implemented correctly.

Creating a Great Arc Aircraft and Adding Waypoints

With two of the important pieces of the communication network for this scenario in place, you can now instantiate the predator aircraft and add its waypoints over Mt St Helens.

1. Type the code below, which will instantiate the aircraft object and give it a predator 3D model to add realism to the scenario.

```
aircraft = scenario.Children.New('eAircraft','Predator');
model = aircraft.VO.Model;
model.ModelData.Filename = 'STKData\VO\Models\Air\rq-1a_predator.mdl';
```

2. For this scenario you are going to use the Great Arc propagator, so use the following code below to set the propagator and preliminarily begin creating the route.

```
aircraft.SetRouteType('ePropagatorGreatArc');
route = aircraft.Route;
route.Method = 'eDetermineTimeAccFromVel';
route.SetAltitudeRefType('eWayPtAltRefMSL');
```

3. Now you can begin adding waypoints to the route. Type the code below to do so.

```
waypoint = route.Waypoints.Add();
waypoint.Latitude = 46.098;
waypoint.Longitude = -122.0823;
waypoint.Altitude = 4.5; % km
waypoint.Speed = .075; % km/sec
waypoint.TurnRadius = 0; % km
```

4. Using the syntax above, add in three additional waypoints with the following values.

Latitude	Longitude	Altitude	Speed	Turn Radius
46.269	-122.192	4.5	0.075	0.25
46.251	-122.248	4.5	0.075	0.25
46.076	-122.131	4.5	0.075	0

5. Finally, use the line below to propagate the route of all of the waypoints you've added.

```
route.Propagate;
```

Try running the code now and seeing if it was implemented correctly.

Adding Terrain and Imagery

For this mission it would be nice to simulate the terrain and imagery for visual purposes, but it is especially important to demonstrate realistic altitudes and access constraints. Here we will use a combination of the Object Model and Connect in MATLAB.

1. Use the code below to create the 3D terrain for the Mt St Helens area and enable use in analysis.

```
%% Get the path to the STK install directory
installDirectory = root.ExecuteCommand('GetDirectory / STKHome').Item(0);

%Instantiates the SceneManager object for use
manager = scenario.SceneManager;

%Adds Terrain in for analysis
cmd = ['Terrain * Add Type PDTT File "' installDirectory 'STKData\VO\Textures\St Helens.pdtt"'];
root.ExecuteCommand(cmd);

%Visually implements the terrain to our 3D Graphics
earthTerrain = manager.Scenes.Item(0).CentralBodies.Earth.Terrain;
terrainTile = earthTerrain.AddUriString([installDirectory 'STKData\VO\Textures\St Helens.pdtt']);
terrain.UseTerrain = true;
```

2. With the terrain implemented, you can add in the Mt St Helens imagery by typing the code below. It is also going to disable the Aerial.ve imagery from Bing Maps, as we only want to see our local Mt St Helens imagery for this scenario.

```
%Visually implements the imagery to our 3D Graphics
earthImagery = manager.Scenes.Item(0).CentralBodies.Earth.Imagery;
imageryTile = earthImagery.AddUriString([installDirectory 'STKData\VO\Textures\St Helens.jp2']);
extentImagery = imageryTile.Extent;

disp('Imagery boundaries: ');
disp(['LatMin: ' num2str(extentImagery{1}) ' LatMax: ' num2str(extentImagery{3})]);
disp(['LonMin: ' num2str(extentImagery{2}) ' LonMax: ' num2str(extentImagery{4})]);

%Enables 3D Graphics Window label declutter
root.ExecuteCommand('VO * Declutter Enable On');
```

Creating Targets with Random Locations Using Automation

Something MATLAB or other languages have as an advantage as an automation tool is the ability to generate simulation-like scenarios to test constraints and predetermined object responses to “new and unexpected” (randomly generated) information in order to better simulate and analyze real-world missions. For this mission scenario, we’re just going to add in one target with a randomized location. For a more involved and interesting version of this code, please see the code sample at <STK Install Folder> \CodeSamples\Automation\Matlab\STK_Automation_Tutorial.m

1. Type the code below, which creates pseudorandom numbers that correspond to a possible area of latitude and longitude surrounding Mt St Helens.

```
targetlocations_lat = 46.1991 - .035 + .07*rand;
targetlocations_long = -122.1864 - .035 + .07*rand;
```

2. Now you can add the target and change some of its properties to continue to add realism to the mission scenario. Use the code below to instantiate the target, enable use of terrain and set an AzEl Mask.

```
target1 = scenario.Children.New('eTarget','Target1');
target1.Position.AssignGeodetic(targetlocations_lat,targetlocations_long,0);
target1.UseTerrain = true;
target1.SetAzElMask('eTerrainData',0);
```

3. Run the script a few times and see how the target is placed in a random location each time. With random generation in use, now you can move on to utilize the automation capabilities of MATLAB. Delete the six lines of code you typed from steps 1. & 2. and instead use the for loop below which utilizes MATLAB's cell arrays to create multiple objects within an array.

```
targetlocations_lat = 46.1991 - .035 + .07*rand(1,5);
targetlocations_long = -122.1864 - .035 + .07*rand(1,5);
for i = 1:5
      tname = ['Target' num2str(i)];
      target(i) = scenario.Children.New('eTarget',tname);
      target(i).Position.AssignGeodetic(targetlocations_lat(1,i),targetlocations_long(1,i),0);
      target(i).UseTerrain = true;
      target(i).SetAzElMask('eTerrainData',0);
end
```

4. Run the script now and you should end up with 5 targets all placed randomly along Mt St Helens. If not for the loop used in automating the creation of these objects, doing this same task would require 22 lines of code rather than just 9. To further demonstrate the use of this, 30 targets with random locations will now be populated with little to no effort. Delete the lines of code used in step 3. and type in the new code below.

```
targetlocations_lat = 46.1991 - .035 + .07*rand(1,30);
targetlocations_long = -122.1864 - .035 + .07*rand(1,30);
for i = 1:30
      tname = ['Target' num2str(i)];
      target(i) = scenario.Children.New('eTarget',tname);
      target(i).Position.AssignGeodetic(targetlocations_lat(1,i),targetlocations_long(1,i),0);
      target(i).UseTerrain = true;
      target(i).SetAzElMask('eTerrainData',0);
end
```

5. Again, run the script and you will find that with just 9 lines of code you could complete that much work within STK. Given that this mission planning does not require 30 targets, you can bring that number down to just 2. To do it this time, instead of deleting your lines of code, simply change the three "30" numbers in your nine lines of code above to "3".
6. Because achieving access is an important part of this mission, you're going to enable 3D graphics for the AzEl Masks that interact with the terrain. To do this for your three targets, type the code

below. Note that this code inside the loop below could simply be added to the loop you're using for target creation.

```
for n = 1:3
    tname = ['Target' num2str(n)];
    azelMask(n) = target(n).Graphics.AzElMask;
    azelMask(n).RangeVisible = true;
    azelMask(n).NumberOfRangeSteps = 10;
    azelMask(n).DisplayRangeMinimum = 0;   % km
    azelMask(n).DisplayRangeMaximum = 1;  % km
    azelMask(n).RangeColorVisible = true;
    azelMask(n).RangeColor = 16776960; % cyan
    root.ExecuteCommand(['VO */Target/' tname ' AzElMask ShowCompassLabels Off']);
    root.ExecuteCommand(['SetConstraint */Target/' tname ' AzElMask On']);
end
```

Retrieving Data Providers

Something MATLAB can do very easily is retrieve Data Provider information from STK.

1. To analyze the aircraft flight plan, you'll retrieve the Latitude, Longitude and Altitude information at intervals of 30 seconds. To do this, use the code below.

```
root.UnitPreferences.Item('DateFormat').SetCurrentUnit('EpSec');
llaFixed = aircraft.DataProviders.Item('LLA State').Group.Item('Fixed');
aircraftPosDP = llaFixed.Exec(scenario.StartTime,scenario.StopTime,30);
aircraftLat = cell2mat(aircraftPosDP.DataSets.GetDataSetByName('Lat').GetValues);
aircraftLon = cell2mat(aircraftPosDP.DataSets.GetDataSetByName('Lon').GetValues);
aircraftAlt = cell2mat(aircraftPosDP.DataSets.GetDataSetByName('Alt').GetValues);
disp([aircraftLat aircraftLon aircraftAlt])
```

2. If you haven't been running the code after each section, now is a good time to save your script, run the code and see if the scenario is correctly created and populated and if your command window has output the Data Provider information about the predator aircraft.

Adding a Sensor and Defining Pointing

Now that we have defined our aircraft and our targets, we can add a sensor object to model the predators FLIR sensor (notional).

1. As stated in the problem statement, the exercise will just be using a simple conic sensor to represent the actual various sensors of a predator UAV. The code below will create a pointing sensor and add your target as the object it will be pointing at.

```
sensor = aircraft.Children.New('eSensor', 'Targeted');
pattern1 = sensor.Pattern;
pattern1.ConeAngle = 5;
sensor.SetPointingType('eSnPtTargeted');
pointing1 = sensor.Pointing;
pointing1.Targets.AddObject(target(1));
```

Targeting the Sensor at the Closet Target

STK does not have a pre-defined method to automatically point sensors at the closest targets. STK will point the sensor at the first target it has access to, and continue pointing at that object until it loses access; then it will switch to the next object and so on. However, custom logic can be incorporated in MATLAB, which can be used to define the sensor pointing in STK. This custom logic is beyond the scope of this tutorial, however the code sample shows how to implement sample custom pointing logic.

Computing Access and Chain Access

1. Below you will find code that generates a basic access object between the satellite and the facility, and uses this access to retrieve data providers.

```
access = satellite.GetAccessToObject(facility);
access.ComputeAccess;
accessDP = access.DataProviders.Item('Access Data').Exec(scenario.StartTime,scenario.StopTime);
accessStartTimes = accessDP.DataSets.GetDataSetByName('Start Time').GetValues;
accessStopTimes = accessDP.DataSets.GetDataSetByName('Stop Time').GetValues;
llaFixed = satellite.DataProviders.Item('LLA State').Group.Item('Fixed');
satelliteDP = llaFixed.ExecElements(accessStartTimes{1},accessStopTimes{1},30,{'Time';'Alt'});
satellitealtitude = satelliteDP.DataSets.GetDataSetByName('Alt').GetValues;
```

2. Now you can create a chain and compute the access. Type the code below to create the chain, add objects to it and compute access.

```
chain = scenario.Children.New('eChain', 'SensorInfoNetwork');
chain.Objects.AddObject(target(1));
chain.Objects.AddObject(sensor);
chain.Objects.AddObject(aircraft);
chain.Objects.AddObject(satellite);
chain.Objects.AddObject(facility);
chain.ComputeAccess();
```

With all of this code implemented, running it will now show you a scenario that should feel much more like a “real” scenario for mission planning than what you started with towards the beginning of this exercise. With continued implementation of the more unique you will find that you can end up with very sophisticated scenarios that can be used for a variety of specific purposes – not to mention that automation in circumstances with many objects and many steps is highly efficient, given that you can change a high volume of objects with a just few edits to your code.

Setting Camera View and Animating the Scenario

Now that your baseline code is written, and your scenario is ready to be used in the predator mission planning, you can create a custom view and animate the scenario to get a better feeling for the circumstances the UAV will be dealing with.

1. Below is an example of setting a camera view on an object with offsets using the Object Model. For this exercise you are going to use a bird’s eye view.

```
axes = aircraft.Vgt.Axes.Item('TopoCentric');
point = aircraft.vgt.Points.Item('Center');
offset = {-5.1575; -0.75; 6.0};
upDirection = {-0.0906; -0.6908; 0.7173};
manager = scenario.SceneManager;
camera = manager.scenes.Item(0).Camera;
camera.ViewOffsetWithUpAxis(axes , point, offset, upDirection);
camera.ConstrainedUpAxis = 'eStkGraphicsConstrainedUpAxisZ';
camera.FieldOfView = 45;
camera.LockViewDirection = false;
manager.Render;
```

2. This code will use Connect to set the view and zoom out a bit. The Object Model lines will edit the animation time step to 0.5 seconds, and the final Connect line starts the animation. Now whenever you run your script, the scenario will be created, populate itself and change various properties, and then play out the animation of the newly generated scenario for you.

```
path = [installDirectory 'STKData/VO/Textures/St Helens.jp2"'];
cmd = ['VO * ViewFromTo Normal From "' path];
root.ExecuteCommand(cmd);
root.ExecuteCommand('VO * View Zoom WindowID 1 FractionofCB -0.0015');
animation = scenario.Animation;
animation.AnimStepValue = 0.5;
root.ExecuteCommand('Animate * Start End');
```

3. Given that all of your code has been implemented correctly, save your script, run it, and minimize or move the MATLAB window out of the way from where the 3D Graphics Window in STK will be. The scenario should be generated and should do everything we set out to do in the problem statement.

Additional STK / MATLAB resources

As it has been stated throughout the exercise, the uses of MATLAB and other languages in conjunction with STK go far beyond the basic functions that are portrayed here. Please continue to explore the code sample for this exercise, along with the additional code samples in the following folder:

<STK install folder>/CodeSamples/Automation/Matlab

Additionally, there are more code snippets in the STK Programming Interface help documentation.

STK Programming Interface Help > Using Core Libraries > STK Object Model > MATLAB Code Snippets