Using MATLAB to build RF scenarios for the Colosseum Network Emulator

Channel simulator Instruction Manual

Miead Tehrani Moayyed

1. Introduction

This document explains how to use the developed pipeline that utilizes MATLAB ray-tracing capability to generate RF scenarios to be run on Colosseum, the largest wireless network emulator in the world with hardware-in-the-loop.

Key steps are described in detail in the next sections that include: (1) Obtaining a geographic area of interest in the simulator; (2) defining the wireless devices that will be used in the emulation, including their mobility patterns (3) scenario simulation to characterize its RF footprint through MATLAB-based raytracing and determining the Finite Impulse Response non-zero filter coefficients needed for emulation in Colosseum. These steps are demonstrated through the RF scenario simulation example of NU campus LTE small cell and WIFI coexistence.

2. Mobile channel simulation

2.1 Obtain the wireless environment

The core of our channel simulator is the MATLAB ray-tracer which supports the OSM files as the wireless environment for simulating the wireless channel using the ray-tracing algorithm.

Follow the steps below to download the 3D model of the geographic area of interest from the open street map. These steps are shown in the figure below.

1- Open the URL “openstreetmap.org”

2- In the search box enter the address of the geographic area of interest

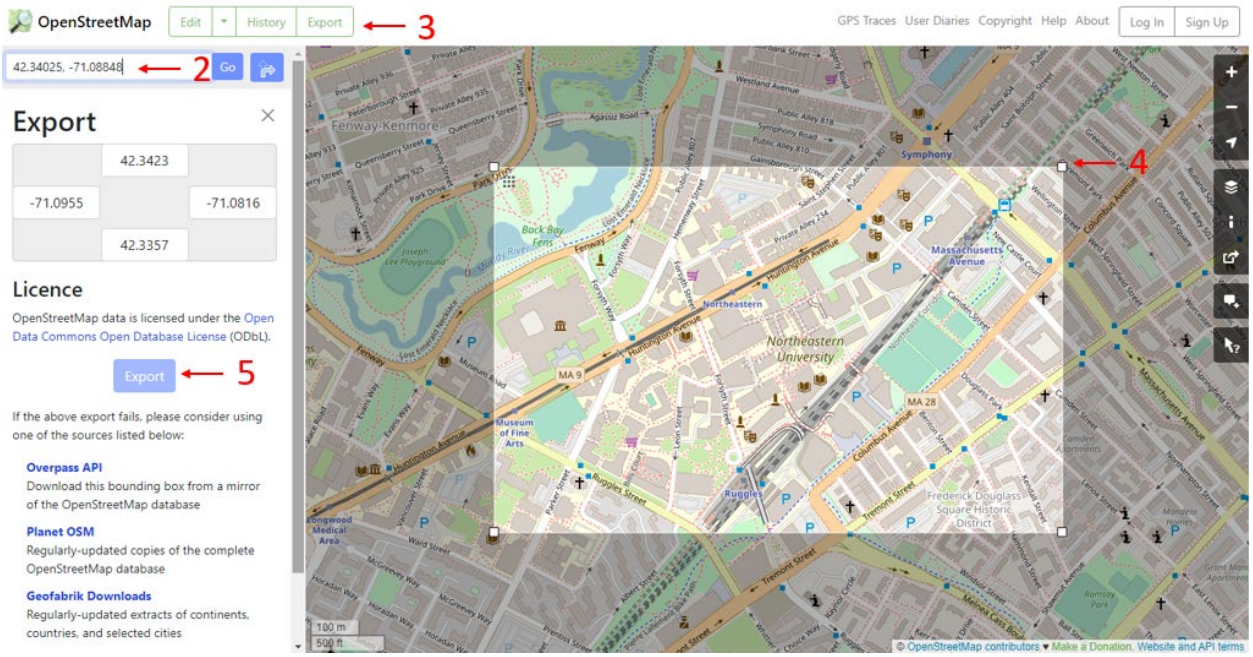
   You can also search by latitude and longitude

3- Click on the “Export” button on the top left and click on the “Manually select a different area” link on the left panel of the map

4- Determine the geographic area of interest on the map by clicking on the corner of the map selected area and modifying the region

5- Click on the “Export” button in the left panel and download the OSM file that includes the 3D model of the environment.

Acknowledgements: This work is supported by MathWorks under the Development-Collaboration Research Grant



2.2 Setting up channel simulator and its software structure

Download the channel simulator from this link and unzip the content in a folder. This folder contains the MATLAB function listed in the table below as well as a live script named “chSimulatorDemo.mlx”. This file provides flexible test and debug, and we also include its MATLAB M file script “chSimulator.m” for accelerating the simulation process.

channelSimulator.m	Main function for simulating mobile wireless channels using ray-tracing propagation model
createRoute.m	Internal function utilized by channelSimulator to sample the mobile nodes’ channels in their trajectory
generatePLmatrix.m	Calculates the path loss of the channel matrix, output of the channelSimulator function
channelApproximation.m	Approximate the simulated channels to the four non-zero taps for emulation scenario using Colosseum
channelSimulatorDemo.m	A script to demonstrate generating an RF emulation scenario for Colosseum toolchain

2.3 Channel Simulator Configuration

The channel simulator read input parameters in a structure explained below. The configuration includes simulation parameters, setting up properties of the nodes, and ray-tracing configuration.

Simulation parameters specify the origin, propagation environment, and properties for simulating mobility in the simulator. These parameters are listed in the table below.

.origin	Wireless environment origin, a 1 by 2 array with latitude and longitude

.osmfile	Wireless environment 3D Model OSM file name and location, a string
.fc	Carrier frequency in Hz, scalar value
.Ts	Mobile channel sampling interval in second
.scenarioDuration	Scenario duration in second

Wireless nodes' properties define the wireless parameters, geographical location, and mobility characteristics required for modeling the radio nodes in the simulator. These parameters are listed in the table below.

.nodes.names	Nodes names, cell array of strings
.nodes.powers	Transmit power in dBm, cell array of scalar values
.nodes.coordinates	Nodes coordinates or control points for stationary or mobile nodes
.nodes.heights	Height of nodes
.nodes.antenna	Nodes antenna type, a cell array of antenna objects
.nodes.antAngle	Nodes antenna azimuth angles in degree
.nodes.velocity	Nodes velocity in m/s], scalar values cell
.nodes.mobilityType	Nodes mobility type, can be 'stationary', 'route' or 'custom'

Ray-tracing propagation model parameters configure the MATLAB ray-tracing algorithm to compute the multipath between the transmitter and receiver. They also include properties of the building and terrain surfaces as well as polarization. These parameters are listed in the table below.

.rt.nReflections	Number of reflections in raytracing
.rt.BuildingsMaterial	Type of buildings material, takes one of these types: "perfect-reflector", "concrete", "brick", "wood", "glass", "metal", or "custom". More details in this link
.rt.BuildingsMaterialPermittivity	Building material 3ermittivity for custom material
.rt.BuildingsMaterialConductivity	Building material conductivity for custom material
.rt.TerrainMaterial	Type of terrain material, more details in this link
.rt.TerrainMaterialPermittivity	Terrain material 3ermittivity for custom material
.rt.TerrainMaterialConductivity	Terrain material conductivity for custom material
.rt.polarization	Polarization {"H" or "V"}, refer to this link for more details

Visualization parameters are used to visualize the nodes and their mobility in the propagation environment. These parameters also include the transmitter and receiver of interest for displaying propagation paths find by the ray-tracing algorithm. These parameters are listed in the table below.

.visualization.flag	Visualization flag, {1 or 0}, activate visualization of mobility simulation and ray-tracing propagation paths
.visualization.TxId	Define TX Index for the transmitter in the visualizations

.visualization.RxId	Define RX Index for the receiver in the visualizations

2.4 Channel simulator output

The channel simulator output three matrices, channel matrix, nodes' coordinates, and time instants of the channel snapshots.

The "chMatrix" output is a tensor of cells with the dimension of the number of the nodes by the number of nodes by the number of snapshots and each cell element is a table that specifies the temporal and spatial parameters of the Multi-Path Components of the channel between the particular transmitter and receiver.

The "coordinates" output is also a tensor of cells with the dimension definition of the number of nodes by four by the number of snapshots. For each snapshot, the matrix has four columns that specify node index, X, Y, and Z coordinates respectively.

The "Time" output is an array that records the time instants of the channel snapshots in milliseconds.

Finally, The channel matrix output needs to be adapted for emulation. To this end, the "chApproximation" function is used to approximate the channels. The structure of approximated channel matrix along with the required metadata for the Colosseum scenario generator toolchain are documented in *the New scenario file format requirements Freshdesk* page.

3. Mobile channel simulation example

This section demonstrates an example of generating the required output for creating a realistic scenario on the Colosseum.

NU campus LTE small cell and WIFI coexistence

This example simulates a scenario that has two networks of LTE small cell Base Station (BS) and WiFi outdoor Access Point (AP). The LTE network has two LTE users in mobile vehicles and the WiFi network has one user that represents a student walking on the Northeastern campus. The details of this scenario can be found on this page.

The MATLAB code of this example can be found in the "chSimulatorDemo.mlx" file. The simulation parameters of this example to configure the channel simulator are the following.

```
% Parameters Configuration
parameters.origin = [42.34025, -71.08848];              %Wireless environment
origin
parameters.osmfile = "NUcampus.osm";                    %Wireless environment 3D
Model OSM file
parameters.fc = 5.8e9;                                  %LTE ISM band Center
freq. (Hz)
parameters.nodes.names = {'LTE' 'WIFI' 'UE#1' 'UE#2' 'WIFIUE'};  %Nodes names
parameters.nodes.powers = {30, 22, 20, 20, 15};         %Transmit power (dBm)
```

```
parameters.nodes.coordinates = {[42.3405645965253,-71.0892925339939],...
%stationary nodes coordinates
                               [42.3398095978303,-71.0884277156665],...
                               [42.341482,-71.086650; 42.341102,-71.087238;
42.339903,-71.090416],...
                               [42.339757,-71.090343; 42.341045,-71.087006;
42.341412,-71.086518]};%Control points for route
parameters.nodes.heights = {17, 17, [1.5, 1.5, 1.5], [1.5, 1.5, 1.5], 1.5};
%Height of nodes
parameters.nodes.antenna = {'isotropic' 'isotropic' 'isotropic' 'isotropic'
'isotropic'};           %Nodes antenna type
parameters.nodes.antAngle = {0 0 0 0 0};                %Nodes antenna azimuth
angles
parameters.nodes.velocity = {0 0 11.18 5.6 nan};        %Nodes velocity [m/s]
parameters.nodes.mobilityType = {'stationary' 'stationary' 'route' 'route'
'custom'};
parameters.Ts = 1000e-3;                                %Mobile channel sampling
interval [s]
parameters.scenarioDuration = 70;                       %Scenario duration [s]
parameters.rt.nReflections = 3;                         %Number of reflections in
raytracing
parameters.rt.BuildingsMaterial = "custom";             %Type of buildings
material
parameters.rt.BuildingsMaterialPermittivity = 5.31;     %Building material
permitivity for custom material
parameters.rt.BuildingsMaterialConductivity = 0.1353;   %Building material
conductivity for custom material
parameters.rt.TerrainMaterial = "custom";               %Type of terrain material
parameters.rt.TerrainMaterialPermittivity = 5.31;       %Terrain material
permitivity for custom material
parameters.rt.TerrainMaterialConductivity = 0.1353;     %Terrain material
conductivity for custom material
parameters.rt.polarization = "H";                       %Polarization
parameters.visualization.flag = 1;                      %Visualization request
parameters.visualization.TxId = 1;                      %TX Index for
visualization
parameters.visualization.RxId = 4;                      %RX Index for
visualization

% Define the custom mobility pattern - RWP in this case
RWP = load('RWPmobility.mat');
parameters.nodes.coordinates {5} = [RWP.snapshots.lat RWP.snapshots.lon];
```

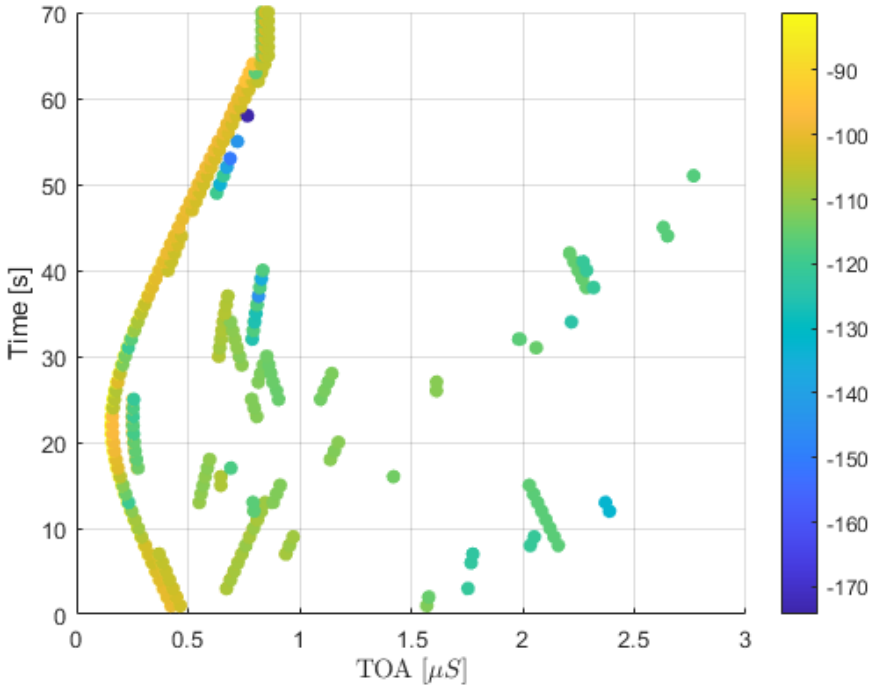
Simulate the wireless scenario using ray-tracing propagation model.

```
[chMatrix, coordinates, Time] = channelSimulator(parameters);
```

Plot time evolution of the paths between the LTE BS and LTE UE#2.

```
toa_scale_factor = 1e6;

figure
TxID = parameters.visualization.TxId;
RxID = parameters.visualization.RxId;
for snapshotIdx = 1:size(chMatrix,3)
    scatter( chMatrix{TxID,RxID,snapshotIdx}.tau * toa_scale_factor, ...
             ones(size(chMatrix{TxID,RxID,snapshotIdx}.tau)) * parameters.Ts *
snapshotIdx, [], ...
             mag2db(abs(chMatrix{TxID,RxID,snapshotIdx}.h)) , 'filled');
    hold on
end
colorbar
grid on
xlabel('TOA [$\mu S$]','Interpreter','latex');
ylabel('Time [s]')
```



Approximate the channels to the four taps for emulation on Colosseum.

```
[chMatrix_hat, plMatrix_hat] = chApproximation(chMatrix,'elevation','NLOS');
```

Wrap the metadata required for Colosseum scenario generator toolchain

```
origin.lat = parameters.origin(1);
origin.long = parameters.origin(2);

NodesName = parameters.nodes.names;

chMatrix = chMatrix_hat;
plMatrix = plMatrix_hat;

% save the required variable for Colosseum multi-tap toolchain
save("RFscenario.mat","chMatrix","plMatrix","NodesName","coordinates","Time","origin");
```

The “RFscenario.mat” file contains all the required variables for the Colosseum scenario generator toolchain standardized in this document to request this new scenario.