LEMANS User Guide

March 2021

LEMANS_V1 is a set of functions to enable the study of Urban Air Mobility methods. This guide aims to help users develop custom simulations based on their specific scenarios.

Overview

A scenario specification defines:

- Underlying roads
- The UTM airways
- A set of flight requests, and with their start times
- A set of reservations through the lanes
- A simulation time interval

LEMANS_V1 allows both lane-based and FAA-NASA scenarios. Lane-based means that a set of lanes are defined along which all flights' travel, whereas FAA-NASA means each flight creates its own trajectory independently of other flights. Both must be strategically deconflicted before execution of the flight.

Consider the following simple example of a lane-based scenario (see Appendix A for complete listing of the function). First, a road set is defined:

```
roads = LEM_gen_grid_roads(0,50,0,50,10,10);
```

Consider This sets up a grid ranging from 0 to 50 in steps of 10 units (see Figure 1 created by call: LEM_show_roads(roads,1)).

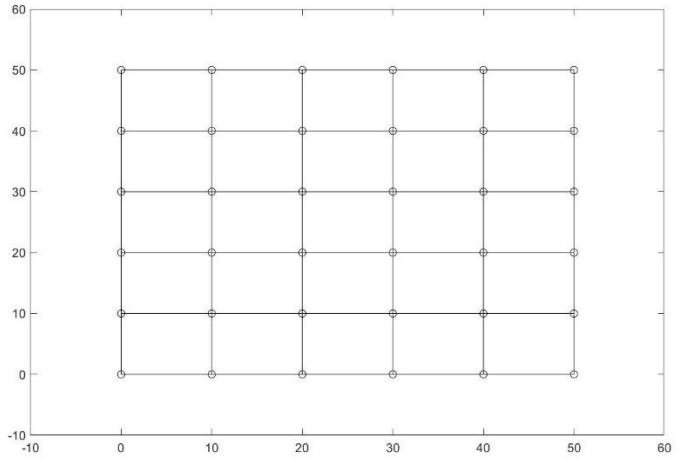

Figure 1. Road Layout for Simple LBSD Simulation.

Next, the airways are established (see Figure 2 created with LEM_show_airways3D(airways,[])). Launch and landing sites are specified in terms of road vertex indexes (here: launch sites are the first 6 vertexes, while landing sites are the last 6).

```
launch_sites = [1:6]; % picks ground vertexes 1 to 6 as launch sites
land_sites = [31:36]; % picks groundvertexes 31 to 36 as land sites
min_lane_len = 2; % sets minimum air lane length to 2 units (feet)
min_altitude = 467; % sets lower lanes' altitude
max_altitude = 534; % sets upper lanes' altitude
airways = LEM_gen_airways(roads,launch_sites,land_sites,min_lane_len,…
                          min_altitude,max_altitude);
```

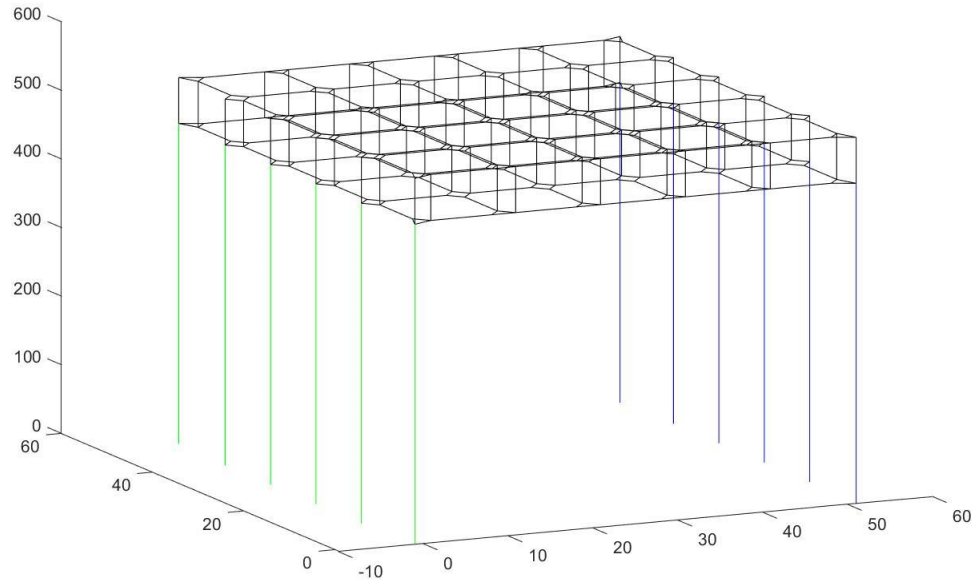

Figure 2. Airway Layout for Simple LBSD Simulation.

Next, a set of 100 flight requests are generated over the time interval [0,100] with a launch time interval of 120 seconds, and with the speed for the entire flight fixed at 10 units per second.

```
num_flights = 100;
time_start = 0;
time_end = 100;
launch_window = 120;
speeds = [10,10];
requests = LEM_gen_requests_LBSD(time_start,time_end,airways,num_flights,…
launch_window,speeds);
[reservations,flights] = LEM_requests2reservations(airways,requests,1);
```

The most important result of the simulation is the calculation of performance statistics, including the delay times for each flight (i.e., the difference between the actual start time and the requested start

time), and the wall clock time for deconfliction; P is an nx5 array with delay in column 2, and deconfliction time in column 4.

```
P = LEM_performance(flights);
```

In order to run a simulation, it is necessary to create the flight trajectories with a call:

```
for f = 1:100
    fo = LEM_gen_traj(flights(f),0.1);
    flights(f).traj = fo.traj;
end
```

For this simulation, the mean flight delay is 0.0101 seconds per flight, while the mean deconfliction time is 0.0051 secs. Finally, the simulation provides a visualization of the flights as scheduled and stores it in a file called sim1_LBSD.mp4 created by the call:

```
LEM_run_flights(airways,flights,1,1,'sim11Mar21');
```

Figure 3 shows one time step from the simulation video.

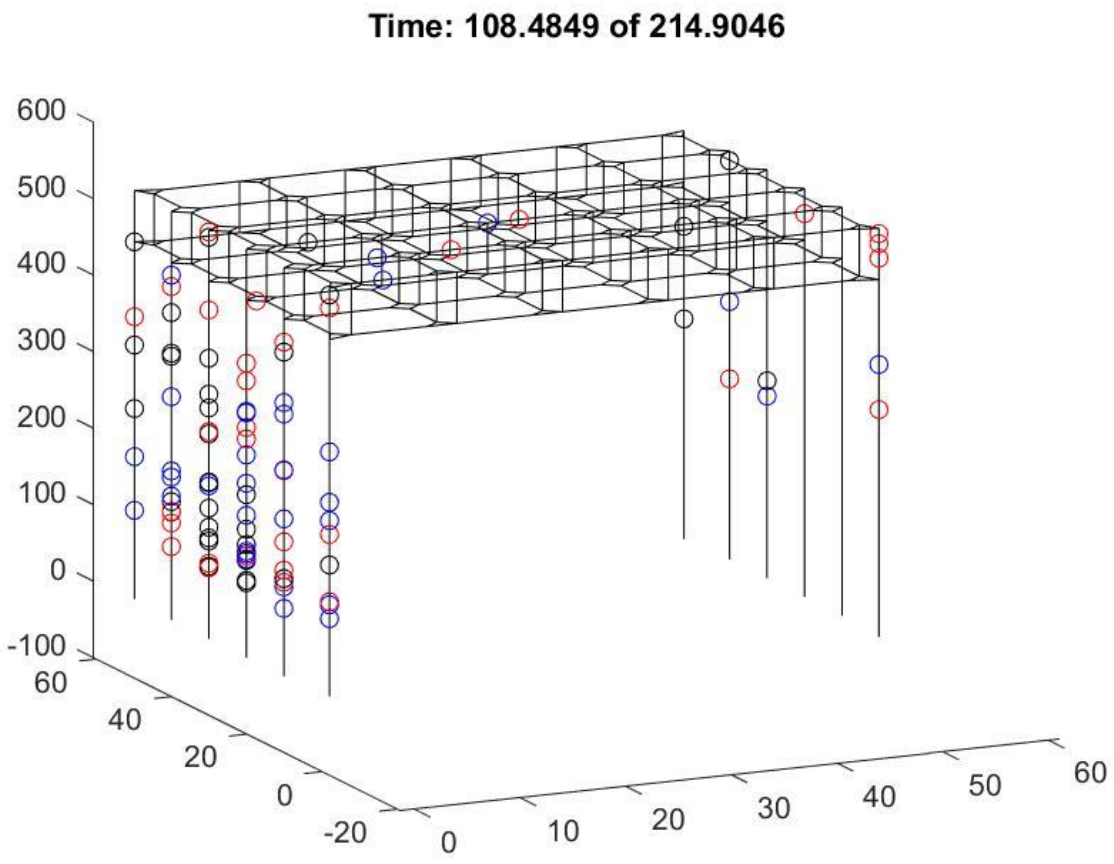

Figure 3. Time Instant 103.1902 in the Flight Simulation.

Figure 4. Time step 91.1902 during the FAA_NASA Simulation.

Appendix A: Example function for lane-based method

```
function res = LEM_sim1_LBSD
% LEM_sim1_LBSD - basic small-scale simulation of lane
based method
% On input:
%     N/A
% On output:
%     sim1_LBSD (mp4 file): visualization of flights
%     res (results struct): results info
%       .roads (roads struct): roads info
%       .airways (airways struct): airways info
%       .requests (requests struct): LBSD requests
%       .reservations (reservations struct): reservations
info
%       .flights (flights struct): flights info
%       .failures_LBSD (n by 1 vector): 0 if LBSD flight
success; else 1
%       .flights_cells (flights struct): cells flight info
%       .cells (cells struct): FAA-NASA cells corresponding
to LBSD layout
%       .requests_cells (requests_cells struct): FAA-NASA
flight requests
%       P_LBSD (nx5 array): performance info for LBSD
%         col 1 (float):
%         col 2 (float): delay (actual_start -
requested_min_start)
%         col 3 (float):
%         col 4 (float): deconflict time
%         col 5 (float):
%     P_cells (nx5 array): performance info for cells
%        cols: same as LBSD
%       .failures_cells (n by 1 vector): 0 if cells flight
success; else 1
% Call:
%     res = LEM_sim1_LBSD;
% Author:
%     T. Henderson
```

```
%     UU
%     Fall 2020
%

FAILED = 0;

res = [];

rng('default');
roads = LEM_gen_grid_roads(0,50,0,50,10,10);
launch_sites = [1:6];
land_sites = [31:36];
airways =
LEM_gen_airways(roads,launch_sites,land_sites,2,467,534);

num_flights = 100;
requests =
LEM_gen_requests_LBSD(0,100,airways,num_flights,120,[10,10]
);
[reservations,flights] =
LEM_requests2reservations(airways,requests,1);

del_t = 0.1;
for f = 1:num_flights
    f_out = LEM_gen_traj(flights(f),del_t);
    flights(f).traj = f_out.traj;
end

LEM_run_flights(airways,flights,1,1,'sim1_LBSD');
P = LEM_performance(flights);

res.roads = roads;
res.airways = airways;
res.requests = requests;
res.reservations = reservations;
res.flights = flights;
failures_LBSD = zeros(num_flights,1);
for f = 1:num_flights
    if flights(f).type==FAILED
        failures_LBSD(f) = 1;
    end
end
res.failures_LBSD = failures_LBSD;
res.mp4 = 'sim1_LBSD.mp4';
```

```
res.P = P;
```

Appendix C: Data Structures

roads (vector struct): ground road layout

.vertexes (n by 3 array): [x,y,z] of road lane endpoints

.edges (p by 2 array): [index1,index2] indexes into vertexes

airways (vector struct): airways layout

.vertexes (n_1x3 array): [x,y,z] road vertexes

.edges (n_2 ×2 array): [index1,index2] road edges

.launch_vertexes (1×n_3 vector): (road-based) indexes for launch sites

.land_vertexes (1×n_4): (road-based) indexes for land sites

.min_lane_len (float): minimum length for any lane

.g_z_upper (float): highest altitude for airway lanes

.g_z_lower (float): lowest altitude for airway lanes

.roundabouts_up (1×n_1 struct):

.roundabouts_dn (1×n_1 struct):

.lanes (n_5×6 array): [x1,y1,z1,x2,y2,z2] endpoints of lanes

.lane_vertexes (n_6×3 array): lane 3D vertexes

.lane_edges (n_5×2 array): [index1,index2] lane vertex indexes for lanes

.G (digraph): directed graph for lane alyout

.launch_lane_vertexes (1×n_3 int): lane indexes for launch lanes

.land_lane_vertexes (1×n_3 int): lane indexes for land lanes

.lane_lengths (n_5×1 vector): lane lengths

requests (vector struct): flight request info

.request_time (float): time request received

.launch_interval (1x2 vector): [t1,t2] earliest and latest possible launch times

.speed (float): desired average speed through flight

.path (1xn vector): lane sequence for requested flight

.path_vertexes (nx6 array): entry exit vertexes for each lane

.launch_index (int): launch index

.land_index (int): land_index

flights (vector struct): individual flight info

.type (int): planned, rogue, etc.

.decon_time (float): time for deconfliction (secs)

.plan (nx4 array): [t1,t2,speed,lane] – entry time, exit time, speed in lane, lane index

.route (nx9 array): [x1,y1,x1,x2,y2,z2,t1,t2,speed] lane endpoints, entry time, exit time, speed in lane

.telemetry (mx7 array): [x,y,z,dx,dy,dz,speed] – location, direction, speed

.start_interval (1x2 vector): requested time interval for launch

.speed (float): requested speed (through all lanes)

.path (1xn vector): lane indexes for flight

.path_vertexes (nx6 array): entry and exit vertexes for lane

.launch_index (int): index into launch indexes

.land_index (int): index into land indexes

.start_time (float): assigned launch time

.end_time (float): assigned landing time

.traj (px4 array): locations through time of flight [x y z t

reservations (vector struct): reservations information

.flights (nx4 array): [flight_id, entry time, exit time, speed]

.hd (float): headway distance