The URBAN Large-scale UAS Flight Management System

Thomas C. Henderson
University of Utah

School of Computing
University of Utah
Salt Lake City, UT 84112 USA

6 August 2019

Abstract

This document details the algorithms and data structures of the *URBAN* Large-scale UAS Flight Management System.

1 Introduction

URBAN is a large-scale USA flight management simulation system written in Matlab. The current set of functions includes:

```
An example simulation set:

  UR_URBAN_sim1
    * UR_gen_roads_fixed1
    * UR_flight_manager
      - UR_gen_airways
        + UR_gen_corridors
      - UR_comms
      - UR_find_fp
        + UR_shortest_path_cor
        + UR_possible_times_int
          o UR_OK_sched_req_enum
```

```
          o UR_merge_intervals
    * UR_run_USS
    * UR_error

Miscellaneous functions:

  UR_URBAN_sim_SLC
  UR_URBAN_sim_Renyi
  UR_sow_airways3D
  UR_display_flights_cor
  UR_gen_roads_fixed
  UR_gen_roads_fixed3x3
  UR_gen_roads_X
  UR_USS_17
  UR_USS_23
  UR_USS_111
  UR_save_session_data
  UR_MW_vis
```

The simulation is organized around a set of agents:

- flight manager
- comms agent
- USS agents
- UAV agents

The flight manager and all USS agents first register with the comms agent in order to get a comms code and to enable transmission of messages to other agents. Once USS's are registered with the comms agent, they must register with the flight manager. After USS agents have received a registration acknowledgment from the flight manager, they can send flight scheduling requests. Such a request has a message with the following information:

```
.data: [USS_ID, speed, Op_id, pf_id, launch_vertex, land_vertex,
        time_start, time_end, request_number]
```

1.1 Flight Manager

The flight manager:

- initializes corridors (given a set of roads and intersections)
- acquires launch/land sites
- registers with comms agent
- registers USS's
- schedules flights for USS's

1.2 USS Agents

USS's handle flights for operators:

- gets launch and land sites
- registers with comms agent
- registers with flight manager
- generates flight requests
- requests flight schedules from flight manager
- receives telemetry info from UAS's (not implemented)
- sends contigency info to UAS's and flight manager (not implemented)

1.3 UAS Agents

Each USS agent will ultimately be a separate Matlab function with its own custom data and experiences. It will run a percept-action agent loop and will have state (e.g., $x, y, z, \dot{x}, \dot{y}, \dot{z}$, heading, ground speed) as well as sensor data (i.e., percepts like GPS, altimeter, ground speed, cameras, lidar, etc.).

- has own platform info
- has communication info
- has agent framework
  - receives percepts and messages
  - provides actions and messages
  - handles contingencies

1.4 Mirror World Agent

The MirrorWorld agent takes the simulation information and provides a visualization; it may also provide a physical simulation for each platform:

- receives flight launch info and agent function names
- gives percept data to UAS agents
- gives messages to agents
- receives actions from agents
- updates UAS state based on requested action and environmental conditions
- handles communication between agents

1.5 System Flow Chart

The process flow is as shown in Figure 1.

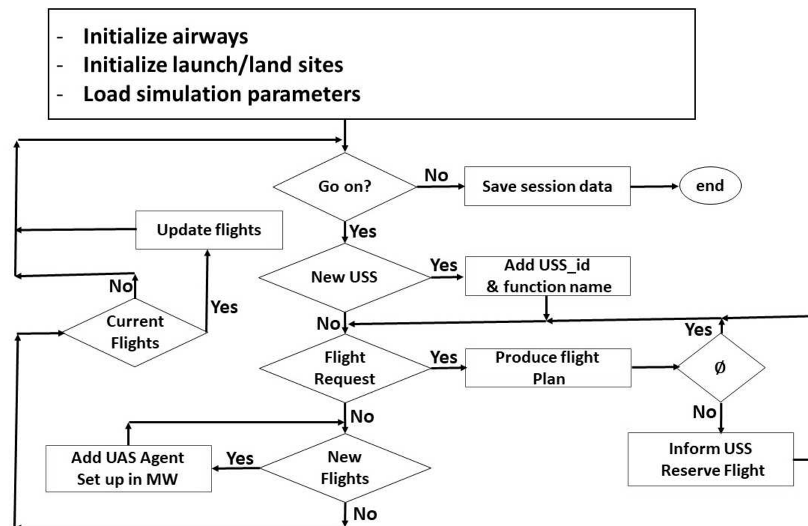

Figure 1: Flow of Simulation Processing.

2 Data Structures and Associated Files

2.1 Global Data Structures

The global data structures are defined in *UR_flight_manager*:

```
global g_radius g_offset g_del
       g_radius: radius for roundabout circles
       g_offset: offset from circle into between vertex lanes
       g_del: perpendicular offset onto circle

global g_USS_info
       g_USS_info: describes USS's

global g_airways
       g_airways: describes airways

global g_z_upper g_z_lower g_z_circle g_z_launch g_z_land
       g_z_upper: altitude for upper lane between intersections
       g_z_lower: altitude for lower lane between intersections
       g_z_circle: altitude for roundabouts
       g_z_launch: altitude for launch site (on ground)
       g_z_land: altitude for landing site (on ground)

global g_speed
       g_speed: unused

global g_flights g_flight_plans
       g_flights: flight info by corridor
       g_flight_plans: flight plan info by UAS

global g_launch_vertexes g_land_vertexes
       g_launch_vertexes: indexes of road vertexes for launch sites
       g_land_vertexes: indexes of road vertexes for land sites

global g_ht
       g_ht: min headway time between flights
```

2.2 g_airways

Airways are described by the following fields:

```
.vertexes: road intersection locations (2D) [x_i,y_i]
.edges: road connections [index_i1,index_i2]
.launch_vertexes: indexes (road) of launch site vertexes
.land_vertexes: indexes (road) of land site vertexes
.corridors: corridor info for all corridors
  [x1 y1 z1 x2 y2 z2 zone c_ind1 c_ind2 v_ind1 v_ind2]
.cor_vertexes: 3D corridor vertex data [x_i,y_i,z_i]
.cor_edges: corridor edges [index_i1,index_i2]
```

2.3 g_USS_info

USS_info provides info about the USS's.

```
g_USS_info(k) has the fields:
  .id (comms code)
  .requests
  .flights
  .uname: function name (e.g., 'UR_USS_111')
```

2.4 g_flights

g_flights gives the scheduled flights ordered by corridor:

```
g_flights(c).flights:  c is the corridor number
   [t1 t2 speed corr USS_id Op_id pf_id launch_vertex land_vertex]
```

Some other data structures used to handle flights include:

```
flight_plans(k).data: corridor sequence for a flight
 [t11 t12 speed corr1 USS_id Op_id pf_id launch_vertex land_vertex,
  t21 t22 speed corr2 USS_id Op_id pf_id launch_vertex land_vertex,
  ...
  tn1 tn2 speed corrn USS_id Op_id pf_id launch_vertex land_vertex]
```

An individual flight plan is put into the flight_info data structure.

3 Time

Time is currently handled as integer steps. Some work was done to use current date and time functions and encode them in an index (by minute from some start time, e.g., 1 January 2019).

```
cur_date = date;
cur_time = clock;
time_index = UR_time_index(cur_date,cur_time(4),cur_time(5));
```

4 Communications

Communications is handled so as to simulate radio transmissions to some extent. The *UR_comms* function allows 4 actions:

- DIRECTORY: request comm code for known entities
- SEND: send a message
- RECEIVE: receive messages
- REGISTER: register to get a comms code

UR_comms takes the parameters: (1) action, (2) params, and (3) mess. These variables take on different values according to the requested action.

4.1 Directory Action

The DIRECTORY action returns either the flight manager comms code or that of the comms agent.

action: 1

params: can either be 'flight-manager' or 'comms' – it is not possible to get the comms code for any other entity

mess: [] (empty)

4.2 Send Action

The SEND action sets up a message to be relayed to another agent.

action: 2

params: $[< my_comms_code >, < other_comms_code >]$

mess: struct with fields:

```
.type (int)
  == 1:  USS2FM
  == 2:  FM2USS

.code: sender's comms code

.action:
 == 1: register
 == 2: request

.data: depends on type, code and action
```

```
 type   code   action    data
-----------------------------------------------------------
    1     -       1       USS_name (string)
    1     -       2       [USS_id, speed, Op_id, pf_id,
                           launch_vertex, land_vertex,
                           t_start, t_end, req_no]
-----------------------------------------------------------
    2  <FM_code>  1       USS index in flight manager
    2  <FM_code>  2       flight plan:
                           [t_start, t_end, speed, corridor,
                            USS_id, Op_id, pf_id, launch_vertex,
                            land_vertex,]
```

4.3 Receive Action

The RECEIVE action allows an agent to obtain all messages sent to it since its last request.

action: 3

params: $[< my_comms_code >, < other_comms_code >]$

mess: [] (empty)

4.4 Register Action

The REGISTER action allows an agent to register with the comms agent.

action: 4

params: $< my_name >$ (string

mess: [] (empty)

4.5 Internal Message Representation

UR_comms represents messages in a vector struct:

```
messages(k).sender:     <comms_code>
           .receiver:   <comms_code>
           .mess        message with fields:
             .type
             .code
             .action
             .data

The message format is:
```

```
.type: 1: USS to flight manager
       2: Flight manager to USS

.code: <sender comms code>

.action: 1: register (from USS) / ACK (from flight manager)
         2: request (from USS) / ACK (from flight manager)

.data  <appropriate to type and action>
```

4.6 Examples

Flight manager registers with comms agent:

```
FM_comm = UR_comms(COMM_DIRECTORY,'flight-manager',[]);
```

Flight manager requests messages (i.e., a RECEIVE action);

```
messages = UR_comms(COMM_RECEIVE,[FM_comm,0],[]);
```

Flight manager acknowledges USS registration:

```
mess_out.type = M_FM2USS;
mess_out.code = FM_comm;
mess_out.action = M_REGISTER;
mess_out.data = len_g_USS_info;
UR_comms(COMM_SEND,[FM_comm,mess.code],mess_out);
```

USS registers with comms agent:

```
my_comm = UR_comms(COMM_REGISTER,my_name,[]);
```

USS gets flight manager comms code:

```
FM_comm = UR_comms(COMM_DIRECTORY,'flight-manager',[]);
```

USS registers with flight manager:

```
mess.type = M_USS2FM;
mess.code = my_comm;
mess.action = M_REGISTER;
mess.data = my_name;
UR_comms(COMM_SEND,[my_comm,FM_comm],mess);
```

USS requests messages (RECEIVE action):

```
message = UR_comms(COMM_RECEIVE,[my_comm,FM_comm],[]);
```

USS requests flight to be scheduled:

```
mess.type = M_USS2FM;
mess.code = my_comm;
mess.action = M_REQUEST;
mess.data = [USS_ID,speed,op_id,platform_id,launch_vertex,...
    land_vertex,t_start,t_end,req_no];
UR_comms(COMM_SEND,[my_comm,FM_comm],mess);
```

5 Simulation Data File

Each simulation requires a data file containing values of parameters used to set up the airway corridors (airlanes), and required headway. The file consists of 9 lines specifying:

Parameter Index	*Meaning*
1	radius (roundabout radius)
2	offset (into lane from center of roundabout)
3	del (minimum distance between roundabout points)
4	z_upper (upper airlane altitude)
5	z_lower (lower airlane altitude)
6	z_circle (roundabout altitude)
7	z_launch (launch altitude)
8	z_land (land altitude)
9	g_ht (minimum headway time)