I-AM: Intelligent Air Mobility

The I-AM system is an Agent Based Modeling and Simulation framework for studying Unmanned Aircraft Systems (UAS) and the related UAS Traffic Management (UTM) system. A simulation includes a variety of agent types:

* ATOC: Air Traffic Operations Center
	+ Monitors USS and UAS activity
	+ Provides visualization of simulation
	+ Provides a Matlab movie of the simulation
* GRS: Grid Reservation System
	+ Given a flight path, provides the grid element indexes which are flown over
	+ Registers which USS are operating over which grid elements
	+ Given a list of grid elements, provides a list of USS operating in each grid element
* USS: UAS Service Supplier
	+ Deconflicts flights
	+ Gets authorization for flight
	+ Provides UAS with flight info
	+ Informs USS of contingencies in system
* UAS: aircraft agent
	+ Flies mission
	+ Provides telemetry data throughout the flight
* SIM: although the simulation is not strictly an agent, it communicates with agents as necessary by passing messages

# Functions and Associated Data Structures

## CS6380\_load\_ABMS\_data: provides common data to all agents

% load\_ABMS\_data - script provides common variable definitions to agents

%

% communication info

BROADCAST = '\*';

TELEMETRY = 'TELEMETRY';

ANNOUNCE\_SELF = 'ANNOUNCE\_SELF';

SIM\_TYPE = 'SIM';

USS\_TYPE = 'USS';

UAS\_TYPE = 'UAS';

ATOC\_TYPE = 'ATOC';

GRS\_TYPE = 'GRS';

REQUEST\_GRID = 'REQUEST\_GRID';

GRID = 'GRID';

IN\_GE = 'IN\_GE';

NOT\_GE = 'NOT\_GE';

USS\_GE = 'USS\_GE';

REQUEST\_FLIGHTS = 'REQUEST\_FLIGHTS';

GE\_FLIGHTS = 'GE\_FLIGHTS';

CFB = 'CFB';

BID = 'BID';

ACCEPT\_BID = 'ACCEPT\_BID';

REJECT\_BID = 'REJECT\_BID';

ACCEPT\_CON = 'ACCEPT\_CON';

REJECT\_CON = 'REJECT\_CON';

AWARD\_CON = 'AWARD\_CON';

% agent constraints

Z\_MIN = 10;

Z\_MAX = 20;

MAX\_SPEED = 10;

DIST\_THRESH = 0.1;

H\_T = 1;

% characters

UNDERSCORE = '\_';

## CS6380\_drive\_A4\_ABMS: sets up simulation and calls simulation function

function [res,M] = CS6380\_drive\_A4\_ABMS(max\_t,draw,film,dump)

% CS6380\_drive\_A4\_ABMS - overall driver for A4 ABMS

% On input:

% max\_t (float): max simulation time

% draw (Boolean): draw during simulation

% film (Boolean): make a movie (not implemented)

% dump (Boolean): dump messages to a file named ‘popo’

% On output:

% res (struct vector): results

% M (Matlab movie): movie of flight simulations

% Call:

% [r1,M] = CS6380\_drive\_A4\_ABMS(100,1,1,1);

Sets up the lane structure (5x5 grid with teleports at each grid crossing)

This function uses a global variable (g\_fig) to close the agent table figure after the simulation is done.

## CS6380\_4\_ABMS: name of I-AS simulator (!!)

function [res,M] = CS6380\_A4\_ABMS(fnames,ports,max\_t,del\_t,draw,film,dump)

% CS6380\_A4\_ABMS – A4 ABMS simulator

% On input:

% fnames (struct vector): names of agent function (filenames)

% ports (nx2 array): launch/land ports (x1 y1)

% max\_t (float): max time to simulate

% del\_t (float): time step increment

% draw (Boolean): display each simulation step

% film (Boolean): make a movie from data

% dump (Boolean): dump messages to file ‘popo’

% On output:

% res (struct vector): agent info at each step

% .agents (px9 array): info for each agent

% col 1: agent type (1: USS; 2: UAS: 3: ATOC)

% col 2: x coord

% col 3: y coord

% col 4: z coord

% col 5: dx heading in x

% col 6: dy heading in y

% col 7: dz heading in z

% col 8: ground speed

% col 9: last time called

% M (Matlab movie): movie of flight simulation

% Call:

% [r1,M] = CS6380\_A4\_ABMS(fnames,ports,max\_t,del\_t,1,1,1);

This function uses the name: SIM\_tom\_1 in the From field for messages. It sets up a table of all the agents. Outputs messages to a file named “popo” if the “dump” variable is 1. Draws a step by step visualization if the variable “draw” is set to 1; and makes Matlab movie if the variable “film” is set to 1.

Communicates with the ATOC agent by sending a “DRAW” message type if vis required, and a “FILM” message type if a movie is to be made. Before exiting it sends a “SEND\_FILM” message to ATOC if movie required (this is done through the percept message variable and the movie is returned as the Data variable).

Simulator sets up percept for each agent, calls it and gets action; accumulates all actions (including messages), and then calls each agent in turn to update the state of the world based on its actions.

## CS6380\_ATOC\_tom\_1: Air Traffic Operations Center

Gets the grid info from GRS agent so it can draw the visualization. Handles simulator requests for visualization and movie. Keeps track of all USS and UAS and prints a table of them. Reads telemetry data.

## CS6380\_USS\_tom\_1: UAS Service Supplier (example)

Will generate flights with a fixed probability. Handles GRS messages: gets grid, provides grid element usage info; assigns one of its fleet of UAS to a flight; gets UAS telemetry data.

# Data Structures

## percept fields:

* x: agent x coord
* y: agent y coord
* z: agent z coord [x;y;z] is location vector
* dx: agent x heading
* dy: agent y heading
* dz: agent z heading [dx;dy;dz] is direction vector
* speed: agent speed in direction vector
* time: current time
* messages (mx1 struct vector): each message has fields:
	+ To: name of agent for whom message is intended (\* for BROADCAST)
	+ From: agent sending message
	+ Type: indicates type of message
		- REQUEST\_GRID = 'REQUEST\_GRID';
		- GRID = 'GRID';
		- IN\_GE = 'IN\_GE';
		- NOT\_GE = 'NOT\_GE';
		- USS\_GE = 'USS\_GE';
		- REQUEST\_FLIGHTS = 'REQUEST\_FLIGHTS';
		- GE\_FLIGHTS = 'GE\_FLIGHTS';
		- CFB = 'CFB';
		- BID = 'BID';
		- ACCEPT\_BID = 'ACCEPT\_BID';
		- REJECT\_BID = 'REJECT\_BID';
		- ACCEPT\_CON = 'ACCEPT\_CON';
		- REJECT\_CON = 'REJECT\_CON';
		- AWARD\_CON = 'AWARD\_CON';
		- DRAW = 'DRAW';
		- FILM = 'FILM';
		- SEND\_FILM = 'SEND\_FILM';
* .realx (float): provides simulator with UAS launch location x coord
* .realy (float): provides simulator with UAS launch location y coord
* .realz (float): provides simulator with UAS launch location z coord

## Action fields:

* dx: x heading
* dy: y heading
* dz: z heading
* speed: magnitude of velocity
* messages (as above)

## flights: keep info about all flights; fields:

* id: index in flights array
* traj: flight trajectory
* speed: flight speed
* start\_time\_or: originally requested start time
* start\_time: actual start time
* stop\_time: scheduled arrival time
* UAS: name of UAS making flight
* telemetry: data for flight
* status: status of flight: 1 initial; 2 waiting for bids; 3 all bids received; 4 accepted contract; 5 awarded contract

# USS Process

The USS generates flights probabilistically, and keeps the flight info in a flights data structure. In addition, it has a couple of lists to know which UAS are assigned and free. UAS\_assigned has the indexes (into the UAS data structure which has UAS info) of the UAS assigned to a flight. UAS\_free has the indexes of the available UAS. USS\_tom\_1 loads initialization info using a script (CS6380\_load\_USS\_tom\_1) which lets it know the number of UAS at it disposal (num\_UAS). The names of the UAS will be UAS\_tom\_100 through UAS\_<last>. A constant called BASE is used as the offset for the USS (99 in this case). The USS assigns a UAS to any waiting flights first, then generates a new flight and assigns it a UAS if possible. If not, the flight waits for a free UAS. If the flight start time is missed, the flight is marked as unsuccessful and will not fly.

# UAS Process

The UAS process is described by Figure 1:



Figure 1. UAS State Machine

The UAS initializes variables with the script CS6380\_load\_UAS\_tom:

% state names (for state machine)

HANDLE\_PERCEPTS = 1;

SETUP\_FLIGHT = 2;

ANALYZER = 3;

FILTER = 4;

EXECUTE\_PLAN = 5;

WRAP\_UP = 6;

EXIT = 7;

% logical atoms

num\_atoms = 10;

NOMINAL = 1;

IN\_LANE = 2;

ON\_HEADING = 3;

SPEED\_OK = 4;

ASSIGNED = 5;

IN\_FLIGHT = 6;

AT\_START = 7;

AT\_FINISH = 8;

AT\_NEXT\_WAYPT = 9;

LAST\_LANE = 10;

% Behavior variables

IN\_LANE\_THRESH = 1;

ON\_HEADING\_THRESH = 10\*pi/180; % 10 degrees

MAX\_SPEED = 10;

SPEED\_THRESH = 1;

AT\_NEXT\_WAYPT\_THRESH = 0.1;

% plan variables

P\_WAIT = 1;

P\_GO\_TO\_LANE = 2;

P\_CORRECT\_HEADING = 3;

P\_CORRECT\_SPEED = 4;

P\_FOLLOW\_LANE = 5;

P\_WRAP\_UP = 6;

The states perform the following actions:

* HANDLE\_PERCEPTS: updates messages, and converts percepts to beliefs
* SETUP\_FLIGHT: sets variables needed for the flight
* ANALYZER: unused at this time
* FILTER: picks immediate goal and plan
* EXECUTE\_PLAN: gets actions from plan
* WRAP\_UP: sets up for next flight assignment
* EXIT: set action fields and returns to simulator

Goals are handled as shown in Figure 2:



Figure 2. UAS Goal Handling Algorithm

The intention\_stack maintains the current set of goals (and sub-goals) which are being made true. Basically, the UAS waits to be assigned, then waits until it is time to launch, then attempts to follow the trajectory making use of plans to get back in the lane, or on the correct heading or at the correct speed.