DEVS-BASED VALIDATION OF UAV PATH PLANNING IN HOSTILE ENVIRONMENTS

Alejandro Moreno [(a)], Luís de la Torre [(a)], José L. Risco-Martin [(b)], Eva Besada-Portas [(b)] Joaquín Aranda [(a)]

[(a)]Dpto. Informática y Automática, Univ. Nacional de Educación a Distancia (Madrid, Spain)
[(b)]Dpto. Arquitectura de Computadores y Automática, Univ. Complutense de Madrid (Madrid, Spain)

[(a)]amoreno@bec.uned.es, ldltorre@bec.uned.es, jaranda@dia.uned.es [(b)]jlrisco@dacya.ucm.es, evabes@dacya.ucm.es

ABSTRACT

Discrete Event Specification (DEVS) is a sound formal modeling and simulation framework based on concepts derived from dynamic systems theory. DEVS provides a framework for information modeling with several advantages to analyze and design complex systems: completeness, verifiability, extensibility, and maintainability. Unmanned Aerial Vehicles (UAVs) are aircrafts without onboard pilots that can be controlled remotely or fly autonomously based on pre-programmed flight routes. They are used in a wide variety of fields, both civil and military. This research work is focused on taking advantage of DEVS simulation framework to build models that simulate a complex military problem. The simulator is used to validate the results of a route planner for multiple UAVs. The path planner uses several approximations to compute solutions in affordable time, whereas the simulator uses accurate models to validate those results.

Keywords: DEVS, UAV, simulation, path planning

1. INTRODUCTION AND RELATED WORK

Unmanned Aerial Vehicles are aircrafts without onboard pilots that can be controlled remotely or fly autonomously based on pre-programmed flight routes (Stevens and Lewis 2004). They are used in a wide variety of fields, both civil and military, such as surveillance, reconnaissance, geophysical survey, environmental and meteorological monitoring, aerial photography, and search-and-rescue tasks.

In military missions they work in dangerous environments, where it is vital to fly along routes which keep the UAVs away from any type of threat and prohibited zone. The threats of our problem are ADUs, which consists on detection radar to discover the UAVs, a set of tracking radars to follow their trajectories and a set of missiles to destroy them. The prohibited zones, also known as Non Flying Zones (NFZs), are certain regions that the UAVs cannot visit due to mission restrictions.

The best routes for the UAVs are those which minimize the risk of destruction of each UAV and optimize some planning criteria (such as flying time and path length) while fulfilling all the physical constraints of the UAVs and its environment, plus the restrictions imposed by the selected mission (such as forcing the UAVs to visit some points of the map).

Therefore, the motivation of this research is to validate the results of a route planner for multiple UAVs (Besada-Portas et al. 2011) applying DEVS formalism. The path planner uses several approximations to compute solutions in affordable time, whereas the simulator uses accurate models to validate those results.

In order to evaluate the quality of the planner before using it in real missions, we decide to validate the routes in multiple experiments against a simulator that contains models for all the elements of the problem.

In this problem, those elements are the lists of way points (WPs), the UAVs, the radars and the missiles; as well as the terrain, and the controllers coupled with the UAVs, this last group responsible for translating the WPs in maneuverability instructions. The models of the radars and missiles are non-deterministic, incorporating stochastic behaviors related with the probability of detection and destruction of the UAVs. So, two simulations for the same experiment and optimal trajectories can return different results.

The simulation symbolizes a scenario where one or more unmanned aerial vehicles must follow a given trajectory trying to avoid flying within air defense unit's visibility range. The trajectories are calculated prior to the simulation, each trajectory consists of a sequence of way points, each way point may also be a getaway.

To properly simulate this set of elements, correct models have to be used as the basis of the simulation. Traditional flight simulators, such as Microsoft Flight Simulator, FlightGear and X-Plane have very accurate aerodynamics models incorporated in their programs, but they do not include other elements like Air Defense Units or UAV's embedded radars.

In addition, these programs need a lot of memory and computing time to accurately calculate UAV's position and attitude. In this regard, a number of multi-UAV simulations have already been developed. In Rasmussen and Chandler (2002), the authors propose a Matlab-based model of multiple UAVs. However, their model can only be run for up to eight elements. To avoid these limitations, some approaches based on cellular automatas have been presented Glickstein and Stiles (1992), Shem, Mazzuchi and Sarkani (2008),

ISBN 978-88-903724-3-8

Holman, Kuzub and Wainer (2010). For example, in the latter, the 3D cell space is divided into three 2D layers, where the first layer stores UAV position and previous travel path, the second one stores the target location probability terrain, and the last level stores the UAV restricted boundary information. Other approaches consist of agent-based models, like Lundell et al. (2006) and Karim, Heinze and Dunn (2004). However, this paper is focused in the employment of DEVS methodology for development of UAV simulators using the Component Based Development (CBD) process, which is a software development paradigm to assemble applications from reusable, executable software pieces called Components. Once developed, a component is repeatedly used in many projects via well-defined CBD interfaces, which greatly reduces the software development cost and increases the reliability (Kim et al. 2007).

This paper is organized as follows. Section 2 collects some relevant aspects of DEVS. Section 3 describes the modeling of the upper mentioned scenario. Section 4 presents the experiments and results of the simulation. Finally, in section 6 some conclusions are drawn.

2. DEVS

The Discrete Event System Specification is a general formalism for discrete event system modeling based on set theory (Zeigler et al. 2000). It allows representing any system by three sets and five functions: input set (X), output set (Y), state set (S), external transition function (δ_{ext}), internal transition function (δ_{int}), confluent function (δ_{con}), output function (λ), and time advanced function (ta). DEVS provides a framework for information modeling with several advantages to analyze and design complex systems: completeness, verifiability, extensibility, and maintainability. DEVS can also approximate continuous systems using numerical integration methods. Thus, simulation tools based on DEVS are potentially more general than others including continuous simulation tools (Kofman 2004).

DEVS defines system behavior as well as system structure. System behavior in DEVS is described using input and output events as well as states. To this end, DEVS has two kinds of models to represent systems: atomic model and coupled model. The atomic model is the irreducible model definition that specifies the behavior for any modeled entity. The coupled model is the aggregation/composition of two or more atomic and coupled models connected by explicit couplings between ports. The coupled model itself can be a component in a larger coupled model system giving rise to a hierarchical DEVS model construction. The top-level coupled model is usually called the root coupled model.

DEVS models can be simulated with a simple ad-hoc program written in any language. In fact, the simulation of a DEVS model is not much more complicated than the simulation of a Discrete Time Model. The problem arises with models composed by many subsystems where ad-hoc programming becomes very hard. One of the simplest ways to implement these complex models is writing a program with a hierarchical structure equivalent to the hierarchical structure of the model to be simulated. This is the method used in (Zeigler et al. 2000), where a class called Simulator is associated to each atomic DEVS model and a different class called Coordinator is related to each coupled DEVS model. At the top of the hierarchy there is a Coordinator, usually called the Root Coordinator that manages the global simulation time.

3. MODEL

To develop a more formal simulator, we redefine the behavior of all the elements of the system following the DEVS modeling formalism. Although different DEVS tools can be used for this purpose, from the modeling point of view, they are based on atomic and compound model definitions presented in section 2.

For this problem, each example is constructed upon multiple DEVS atomic components that exemplify an UAV dynamics and behavior, together with multiple DEVS coupled components that characterize the line of action of an ADU (see Figure 1). Each ADU is composed by various heterogeneous DEVS atomic components: detection radar; several tracking radars; and certain number of missiles. Wiring rules are depicted by Figure 2.

Below, we describe first the couplings type, and afterwards the structure, behavior and couplings layout of each model.

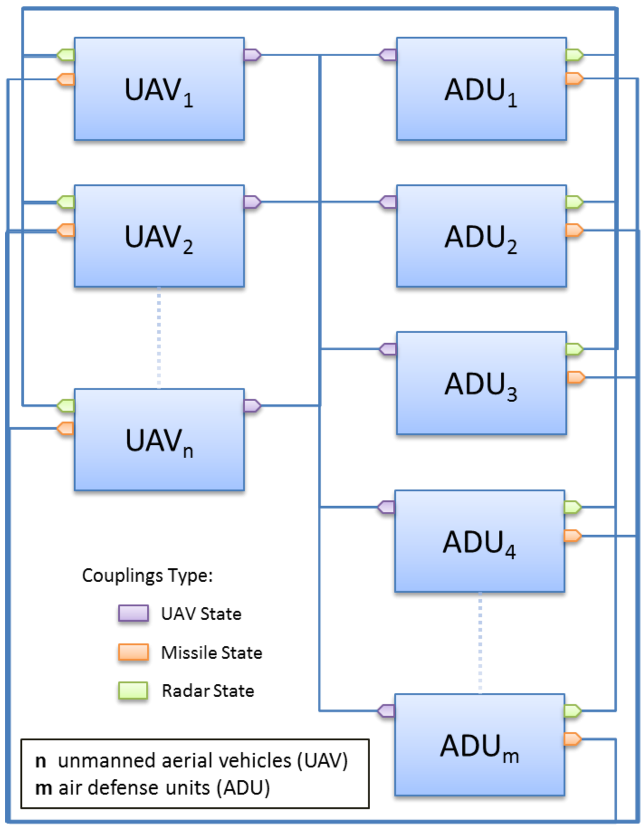


Figure 1 Root Coupled Model

ISBN 978-88-903724-3-8

```
For each uav in UAVs
    For each adu in ADUs
        coupling (UAV State) from uav out(0) to adu in(0);
        coupling (Radar State) from adu out(0) to uav in(0);
        coupling (Missile State) from adu out(1) to uav in(1);
```

Figure 2 Couplings pseudo code

3.1. Couplings

Couplings are directed edges that link output ports to input ports, defined by a type that bounds the data exchanged between models. The DEVS model described in this document uses up to four different coupling types.

- **UAV state**: UAV data type that encapsulates the necessary information to know who, when, where and how.
- **Missile state**: carries the missile information as an UAV, current missile phase and target identifier.
- **Radar tracking state**: covers the tracking radar identifier, the ADU where it belongs, and the data of the UAV to track.
- **Lost target**: the identifier of the UAV lost during the tracking process.

```
UavState {
 String id;              //identifier
 Time t;                 //computed in time
 Double X,Y,Z;           //X, Y and Z coordinates
 Double Vx,Vy,Vz;        //X, Y and Z velocities
 Double theta,phi,psi;     //roll,pitch,yaw
 Double Vtheta,Vphi,Vpsi;//roll,pitch,yaw vels
}
MissileState {
 UavState uav;      // unmanned aerial vehicle
 String phase;      // phase
 String target;     // target identifier
}
RadarTrackingState {
 String id;         //identifier
 String adu;        //air defense unit
 UavState uav;      //unmanned aerial vehicle
}
String lostTarget: // identifier of lost target
```

Figure 3 Couplings types

3.2. UAV

Unmanned aerial vehicles are represented as models assembled with two input ports that receive states of tracking radars and missiles of each ADU correspondingly. One output port that sends its state gathering the computed time, identifier, position, orientation and velocities. Additionally, keeps an internal state variable with an array of these UAV states reflecting changes in time of the UAV dynamics. Whenever an UAV realizes that is been detected by an ADU, starts if possible an evasion maneuver to escape from the ADU fire power range and prevent from being shot down.

Basically, the UAV model works in following way. Every time the internal time event function is triggered (simulation time is equal to sigma), the next necessary collection of states is computed, unless this collection is not empty or the UAV has reached the end of the trajectory. These states store intermediate values of position, orientation and velocities that describe the UAV's movement across the current coordinate to the next trajectory point. Then, sigma (time of next internal time event) is updated to the next computed time or set to ∞ only if the UAV reached the end of the assign path. Whenever the external transition is executed (received an input), the UAV verifies if any radar is tracking his path and whether the distance from a missile aimed at overthrowing it, is less than the established minimum. If the former case is positive, attempts to escape through an intersecting trajectory to flee away from the corresponding ADU and afterwards updates sigma. If the latter is positive and according to a certain probability of destruction, the UAV is destroyed and sigma is set to ∞. On every occasion that the output function is activated the current UAV state is sent thought the output port. Figures 4 and 5 depict this behavior.

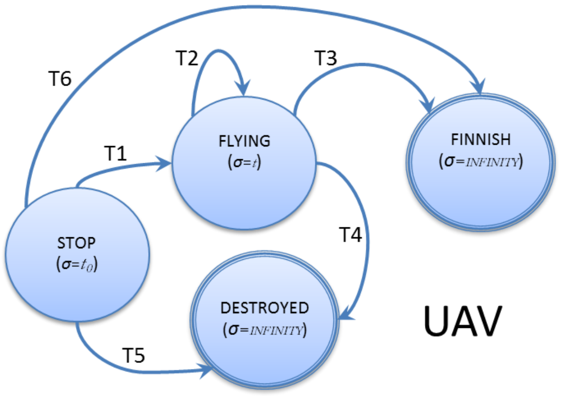


Figure 4 UAV State Diagram

⇨	FLYING	FINNISH	DESTROYED
STOP	T1	T6	T5
FLYING	T2	T3	T4

	δ_{int}	δ_{ext}
T1	*states is NOT empty*	
T2	*states is NOT empty*	*dist. to missile is > MIN*
T3	*states is empty*	
T4		*dist. to missile is <= MIN*
T5		
T6	*states is empty*	

Figure 5 UAV State Transitions

ISBN 978-88-903724-3-8

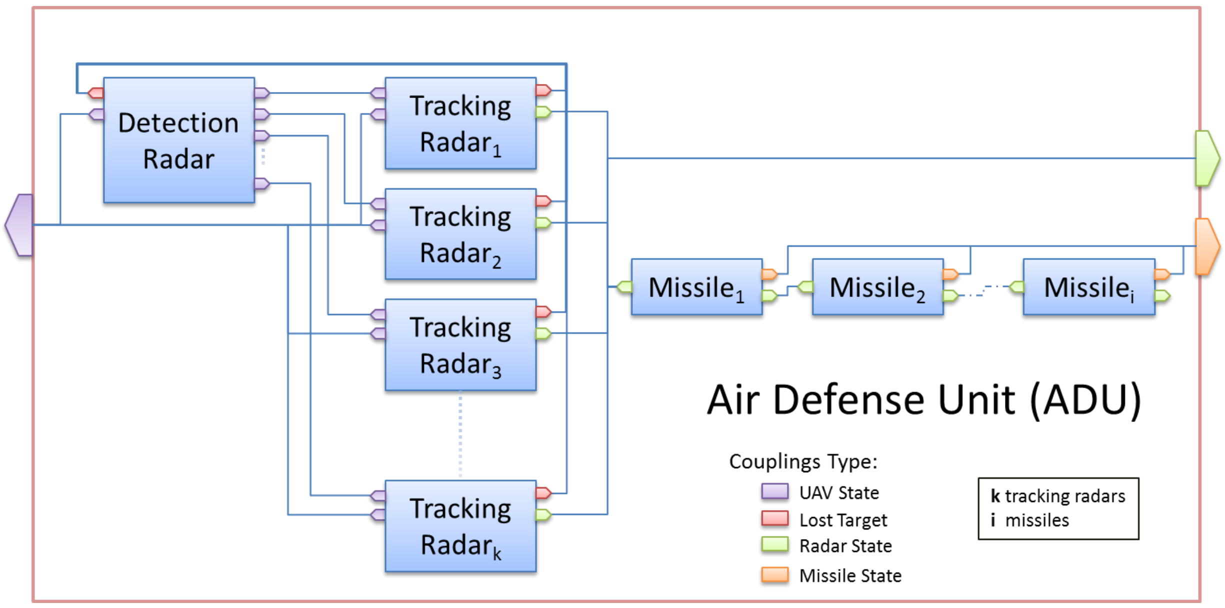


Figure 6 ADU DEVS model

3.3. ADU

Air defense units are coupled models formed by one detection radar, multiple tracking radars and multiple serial connected missiles. Figure 6 depicts the DEVS based ADU model structure and Figure 7 illustrates the pseudo code that clarifies how the wiring is set given the atomic models of all mentioned elements. Detection radars scan the skies seeking for UAVs. If a detection radar detects an UAV in its proximity then looks for task-free tracking radars. Assigns the spotted target UAV to an unoccupied tracking radar. The tracking radar attempts to detect the target UAV, if successful, alerts the first missile model in the row with the corresponding target. If the missile state is already fired, it hands the target to next missile in the row. In essence, on every occasion an ADU detects an UAV, after a specified period of time, shoots a missile to attempt to knock down the UAV. In the subsequent sections, each component is described in more detail.

```
p = 1;
For each tr in Tracking Radars {
    coupling (UAV State) from DetectionRadar out(p) to tr in(0);
    coupling (Lost Target) from tr out(0) to DetectionRadar in(0);
    coupling (Radar State) from rt out(1) to Missile$_1$ in(0);
    coupling (Radar State) from rt out(1) to ADU out(0);
    p++;
}
j=1;
For each m in Missiles {
    coupling (Missile State) from m out(0) to ADU out(1)
    if (j < i)
        coupling (Radar State) from m out(1) to Missile$_{j+1}$ in(0)
    j++
}
```

Figure 7 ADU couplings pseudo code

3.3.1. Detection Radar

The radar detection component consist of one input port that receives UAV's states, another input port that alerts if any tracking radar has lost its target, and one output port per each tracking radar model to transmit the UAV state to follow.

The radar operation is based on its inputs, stores as an internal state variable a collection that maps the assignment between incoming UAVs to tracking radars. When the detection radar receives one or more UAV's states, first, attempts to discover them in its visibility field, if they are within its range and according to a certain probability of detection, checks if they have not been already assign to any tracking radar, and then, in that case, searches for a task free tracking radars to send them, and sets sigma to a certain response time. Otherwise, when it receives notification of a lost target removes the mapping relationship from memory.

3.3.2. Tracking Radar

The tracking radar DEVS based model is design to operate as follows. Through the input port linked to detection radar of the corresponding ADU, obtains the state of a UAV to track, stores its value as an internal state variable and waits for the reception of the same UAV state from the coupling wired to UAV's models. Then, verifies whether the UAV is within its detection field, if it fails and the elapsed time doesn't exceed the defined maximum, estimates its position, orientation and velocities and finally sends the UAV state to the first model of the series of missiles. Otherwise, reports to the Detection Radar that the target has been lost.

3.3.3. Missile

Missiles models are composed by one input port that accepts states of UAVs intended to be blown down. One output port to give over the UAV state to the next missile only if their status is "fired". And another output port to communicate its state to the UAVs so they can

ISBN 978-88-903724-3-8

check whether they are destroyed or not. Like UAVs, they also keep an internal state variable with an array of missile states (same as UAVs) reflecting changes in time of the missile dynamics.

Essentially, as seen on Figures 8 and 9, missiles wait for an external command from any tracking radar model of its corresponding ADU to shift from the initial state stop to be fired. Then, sigma is updated from infinity to the next immediate state time. Afterwards, every time the internal time event function is triggered, jumps to the next computed state. Sigma is updated to the next computed time, unless the array of states is empty, reached its goal, or exceed limits or distance sigma is set to ∞. This behavior is exemplified by Figures 8 and 9.

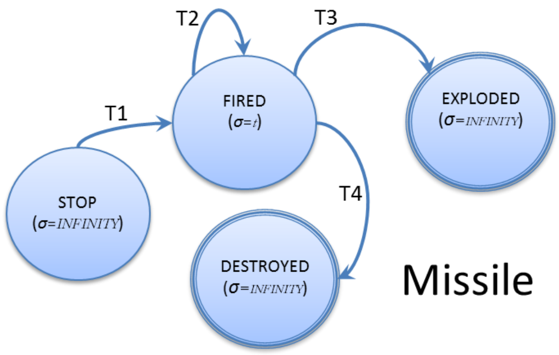
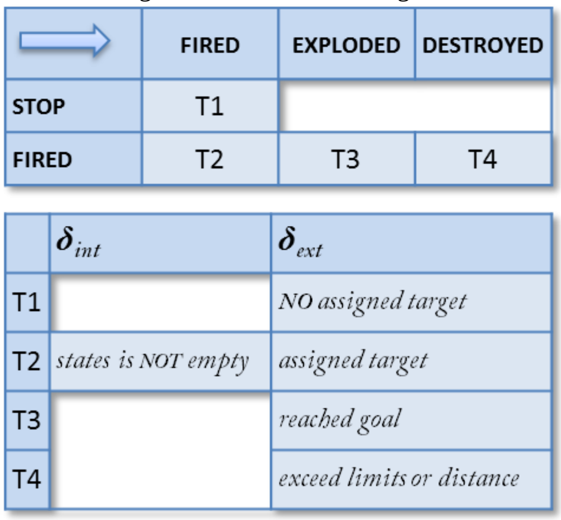


Figure 8 Missile State Diagram



⇨	FIRED	EXPLODED	DESTROYED
STOP	T1		
FIRED	T2	T3	T4

	δ_{int}	δ_{ext}
T1		*NO assigned target*
T2	*states is NOT empty*	*assigned target*
T3		*reached goal*
T4		*exceed limits or distance*

Figure 9 Missile State Transitions

4. SIMULATION

In order to evaluate the quality of the planner before using it in real missions, we decide to validate the routes in multiple experiments against a simulator that contains models for all the elements of the problem.

The models of the radars and missiles are non-deterministic, incorporating stochastic behaviors related with the probability of detection and destruction of the UAVs. So, two simulations for the same experiment and optimal trajectories can return different results.

The simulation symbolizes a scenario where one or more unmanned aerial vehicles must follow a given trajectory trying to avoid flying within ADUs visibility range. Trajectories are calculated prior to the simulation, each trajectory consists of a sequence of way points, each way point may also be a getaway, i.e., an intersection between the current trajectory and an alternative trajectory intended to be used in evasion maneuvers. Whenever an UAV apprehends that is been detected by an ADU, starts if possible an evasion maneuver to escape from the ADU fire power range and prevent from being shot down. Correspondingly, every time an ADU detects an UAV, after a specified period of time, shoots a missile to attempt to knock down the UAV.

4.1. Experiments

The experiment consists of 3 UAVs and different number of ADUs and NFZs. They also differ in the initial and final positions of the UAVs, in the position of the ADUs and NFZs, and in the number of initially known ADUs. There are 4 different types (A, B, C, and D) schematized in Figure 11. The initial and final positions of each UAV are represented as green and magenta crosses. The yellow crosses in experiment type C represent intermediate points that the UAVs are forced to visit. The ADUs are represented by the big blue dashed circles which show the maximum distance of detection of their radars, and by the small red solid circles which enclose the zones where the probability of destroying an UAV can be greater than 0. The NFZs are represented with the rectangular green areas. For this experiment, the offline path planner is configured to consider no ADU's and the initial routes are only by NFZs, terrain and UAVs maneuverability.

Figure 11 represent the 4 experiment types (A, B, C and D), the offline routes for the experiments, and one of the online alternative routes.

4.2. Results

In this section we analyze the results of the aforementioned experiments. For each of the 4 cases, we carry out 30 simulations. Based on their results, we measure the consistency of the simulations. The consistency is characterized by the percentage of successful arrivals of each UAV in each of the 4 cases during the 30 simulations. The performance is measured according with the total simulation time needed in each of the 4 cases.

Table 1 Percentage of success for each example type

	UAV_1	UAV_2	UAV_3
A	100%	100%	10%
B	100%	100%	30%
C	100%	77%	100%
D	13%	100%	17%

The results presented in Table I depend on the UAV, experiment type (A, B, C, or D). For instance, if we focus on the experiment B (second row of Table I), UAV_2 always survives because its initial trajectory is always safe, UAV_3 has a good chance to be destroyed

ISBN 978-88-903724-3-8

because its initial trajectory stays in the non-safe zone too long. Similar explanations apply to the rest of the UAVs in the remaining experiments.

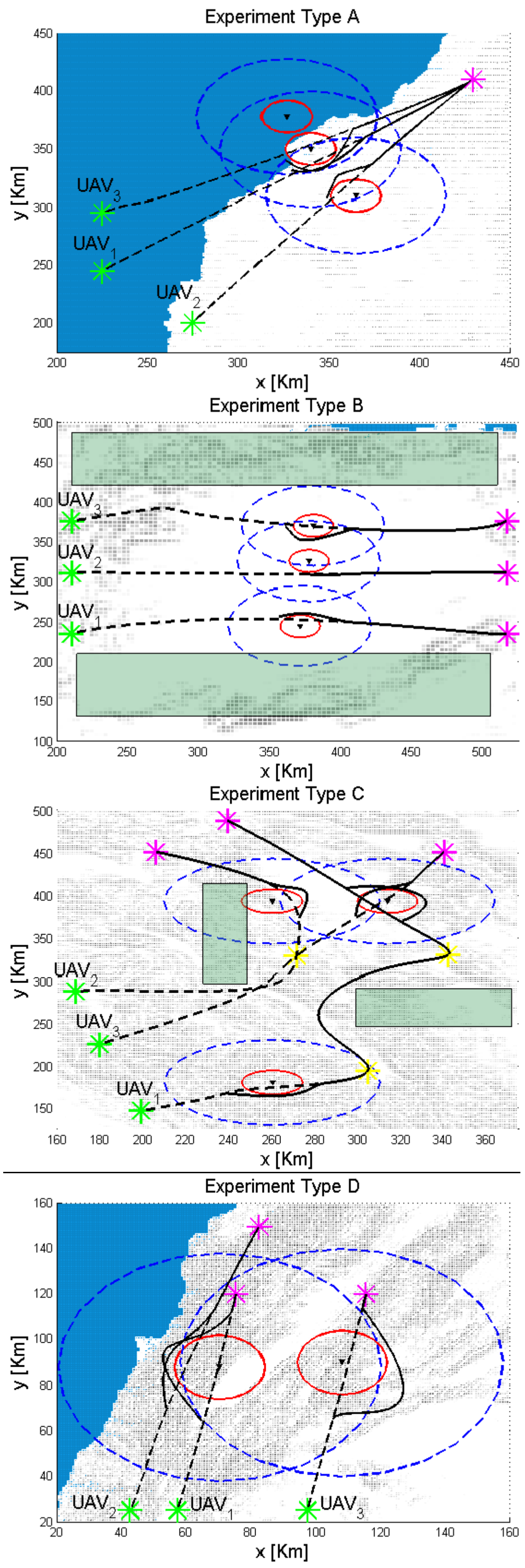


Figure 10 Experiments A, B, C and D

5. CONCLUSIONS

The work presented in this document takes advantage of DEVS simulation framework to build models that simulate a complex military problem. The simulator validates successfully the results of a route planner for multiple UAVs. Builds accurate models to verify the computed solutions of the path planner obtained through several approximations.

ACKNOWLEDGMENTS

This work is supported by the Spanish grants DPI2009-14552-C02-01 and DPI2009-14552-C02-02.

REFERENCES

Stevens, Brian L., and Lewis, Frank L., 2004. *Aircraft Control and Simulation*. 2nd ed. Wiley.

Besada-Portas E., de la Torre L., Moreno A., and Risco-Martín J.L., *Performance Analysis of Multiobjective Bio-inspired UAV Path Planners*. In GECCO 2011.

Rasmussen S. and Chandler P., 2002. Multiuav: a multiple uav simulation for investigation of cooperative control. *Proceedings of the Winter*, vol. 1, pp. 869–877.

Glickstein I. and Stiles P., 1992. Situation assessment using cellular automata paradigm. *Aerospace and Electronic Systems Magazine*, IEEE, vol. 7, no. 1, pp. 32–37.

Shem A., Mazzuchi T., and Sarkani S., 2008. Addressing uncertainty in uav navigation decision-making. *Aerospace and Electronic Systems, IEEE Transactions*, vol. 44, no. 1, pp. 295 –313.

Holman K., Kuzub J., and Wainer G.A., 2010. Uav search strategies using cell devs. *Proceedings of 2010 Spring Simulation Conference*, ANSS Symposium, pp. 192–199.

Lundell M., Tang J., Hogan T., and Nygard K., 2006. An agent-based heterogeneous uav simulator design. *Proceedings of the 5th WSEAS International Conference on Artificial Intelligence, Knowledge Engineering and Data Bases*. WSEAS, pp. 453–457.

Karim S., Heinze C., and Dunn S., 2004. Agent-based mission management for a uav. *Intelligent Sensors, Sensor Networks and Information Processing Conference*, pp. 481 – 486.

Kim J.H, Kim T.G., and Jeong J., 2007. Embedding devs methodology in cbd process for development of war game simulators. *Proceeding of summer computer simulation conference*, pp. 35:1–35:8.

Bernard P. Zeigler and T. Kim and H. Praehofer, 2000. *Theory of Modeling and Simulation: Integrating Discrete Event and Continuous Complex Dynamic Systems*. Academic Press.

Kofman E., 2004. Discrete Event Simulation of Hybrid Systems. *SIAM Journal on Scientific Computing*, 25(5), 1771-1797.

ISBN 978-88-903724-3-8