Journal of Simulation

ISSN: (Print) (Online) Journal homepage: https://www.tandfonline.com/loi/tjsm20

Cloud DEVS-based computation of UAVs trajectories for search and rescue missions

Juan Bordón-Ruiz, Eva Besada-Portas & José A. López-Orozco

To cite this article: Juan Bordón-Ruiz, Eva Besada-Portas & José A. López-Orozco (2022): Cloud DEVS-based computation of UAVs trajectories for search and rescue missions, Journal of Simulation, DOI: 10.1080/17477778.2022.2053311

To link to this article: https://doi.org/10.1080/17477778.2022.2053311

Published online: 05 Apr 2022.

JOURNAL OF SIMULATION
https://doi.org/10.1080/17477778.2022.2053311

RESEARCH ARTICLE

Cloud DEVS-based computation of UAVs trajectories for search and rescue missions

Juan Bordón-Ruiz, Eva Besada-Portas and José A. López-Orozco

Department of Computer Architecture and Automation, Universidad Complutense de Madrid, Madrid, Spain

ABSTRACT

This paper presents a new Cloud-deployable DEVS-based framework for optimising UAV trajectories and sensor strategies in target-search missions. DEVS provides it with a well-established, flexible, and verifiable modelling strategy to include different models for the UAV, sensor, and target dynamics; the target and sensor uncertainty; and the optimising process. Its Cloud deployability speeds up the evaluations/simulations required to optimise this NP-hard problem, which involves computationally heavy models when solving real-world missions. The framework, designed to handle different types of target-search missions, currently optimises, using a multi-objective Genetic Algorithm, free-shape trajectories of multiple UAVs,eqquiped with several static/movable sensors to detect a target within a search area. It is implemented in xDEVS and deployable over a set of containers in the Google Cloud Platform. The results show that our deployment policy speeds up the computation up to 3.35 times, letting the operator simultaneously optimise several search strategies for agiven scenario.

ARTICLE HISTORY
Received 30 April 2021
Accepted 19 February 2022

KEYWORDS
Simulation in Cloud; Discrete Event System Specification; Model-Based Systems Engineering; Bayesian Search; Multi-Objective Path Planning

1. Introduction

Unmanned Aerial Vehicles (UAVs) and their sensorial systems are useful mobile platforms to gather information in large-scale environments of real-world applications. Examples include wildlife monitoring, area patrolling, persistent surveillance, tactical reconnaissance, fire fighting, and search and rescue missions (Ivić et al., 2020; Linchant et al., 2015; Nigam, 2014; Shakhatreh et al., 2019; Skorobogatov et al., 2020; Yeong et al., 2015; Yuan et al., 2015). How to determine the best way of proceeding, i.e., how to compute the best UAV trajectory and sensing strategy for a given application, is an open question, answered by the research community by providing new approaches, algorithms, and heuristics that take 1) into account different aspects of each problem and 2) advantage of the computation advances and capabilities of their moment. In this regard, this paper focuses on developing a new framework, for optimising UAV-based search and rescue operations that considers realistic features of the mission and the computational opportunities that Model-Based System Engineering (MBSE) and Cloud Computing currently offer.

Search and rescue operations can be formulated, at least partially, as target detection problems where the object or person under search is located in an unknown location within a given area, moves randomly, and is observable from imperfect sensors on board UAVs. Moreover, the UAVs and sensor trajectories can be obtained by optimising probabilistic utility functions that simultaneously handle the uncertainty in the target (location and movements) and sensor (observations). Examples of these functions and of works that present strategies to optimise them, are the *entropy* (Yang et al., 2002), the *information gain* (Carpin et al., 2013; Grocholsky et al., 2006; Hu et al., 2014), the *probability of detection* (Delle Fave et al., 2010; Fedorov, 2019; Kratzke et al., 2010; Lanillos et al., 2014; Li et al., 2021; Saadaoui et al., 2018; Tisdale et al., 2009; Wang et al., 2017; Wong et al., 2005; Yao et al., 2017, 2019) and the *expected time of detection* (Lanillos et al., 2013; Perez-Carabaza et al., 2017, 2016; Pérez-Carabaza, Besada-Portas et al., 2019; Pérez-Carabaza, Scherer et al., 2019; Riehl et al., 2011). The last two utility functions are especially interesting in missions where: a) it is possible to know and exploit a probability distribution of the initial target location and b) critical to detect the target as soon as possible.

Sometimes, the optimiser also considers other criteria associated with mission constraints and objectives. In this regard, some works take into account the existence of non-flying zones, UAV collisions, communication requirements, the UAV fuel consumption, the length or smoothness of the UAV trajectories, and/or the area coverage (Carpin et al., 2013; Li et al., 2021; Perez-Carabaza et al., 2017, 2016; Pérez-Carabaza, Besada-Portas et al., 2019; Pérez-Carabaza, Scherer et al., 2019; Yang et al., 2002). Alternatively, and in order to shorten the gap between the solutions (UAVs and sensor trajectories) proposed by the optimiser and its application to real-world search and rescue missions, a few optimisers include the UAV and sensor

CONTACT Juan Bordón-Ruiz jubordon@ucm.es Universidad Complutense de Madrid, Madrid, Spain

dynamical motion models (Delle Fave et al., 2010; Lanillos et al., 2014; Perez-Carabaza et al., 2017, 2016; Pérez-Carabaza, Besada-Portas et al., 2019; Tisdale et al., 2009; Wong et al., 2005; Yao et al., 2019) or non-ideal/non-constant likelihood functions to model the observations uncertainty within the sensor footprint (Delle Fave et al., 2010; Kratzke et al., 2010; Lanillos et al., 2014; Perez-Carabaza et al., 2017, 2016; Pérez-Carabaza, Besada-Portas et al., 2019; Riehl et al., 2011; Wong et al., 2005). These additional functions and models bring realism to the optimisers, usually at the expenses of hardening the evaluation and optimisation of the UAVs and sensor trajectories.

In any case, the detection problem, even just with the probability-based utility functions, is already NP-hard for a single UAV (Trummel & Weisinger, 1986) and can become NEXP-Complete for multiples ones (Yang et al., 2002). Hence, the optimisers are often any-time iterative suboptimal algorithms (e.g., gradient-descendent, Evolutionary Algorithms, Cross-Entropy or Ant Colony Optimisation). Furthermore, they usually reduce the space of possible solutions to the problem by setting up only a few target-search strategies (e.g., with one or two UAVs or pre-fixing the sensors onboard) over a given mission or by lowering the scale of the search area. Hence, to tackle a more significant number of alternatives and bigger scale environments efficiently, this paper proposes a new framework for solving target-search problems that eases the inclusion of new possibilities and that can be straightforwardly deployed, without editing the source code of the model under test and accordingly to their computation requirements, in a Personal Computer (PC) or the Cloud.

The framework presented in this paper has been developed using the Model-Based System Engineering (MBSE, Wymore, 2018) methodology and the Discrete Event System Specification formalism (DEVS, Zeigler et al., 2018). On the one hand, MBSE provides our framework with a clear separation between the (high-level) specifications of the models and their final (computational) implementation. This improves the understanding of the system behaviour and favours the incremental design and refinement of its models. On the other hand, DEVS supports multi-resolution and the integration of the different types of models (e.g., probabilistic and deterministic, continuous and discrete, synchronous, and asynchronous) required by the framework. Both paradigms also help us to debug, verify, and validate the framework, and facilitate its scalability, maintainability, and reusability. Last but not least, xDEVS, which is the DEVS engine where the framework is implemented, provides profiling tools and support for Cloud simulation. We require both capabilities to detect the most consuming models of the framework and to *distribute* their computation in the Cloud (Mittal et al., 2017; Risco-Martín et al., 2017).

The distributed simulation capabilities of xDEVS have been exploited as they provide a good trade-off between scalability and cost. Besides, xDEVS makes use of a straightforward distributed architecture, based on client/server patterns and standard sockets, that streamlines the distribution task and that is most suitable for our problem than other distributed DEVS implementations, such as DEVS/SOA (Mittal et al., 2009) or CD++ (Al-Zoubi & Wainer, 2009), which are based on the concept of Simulation as a Service (SaaS) with stateless models, and PyPDEVS (Van Tendeloo & Vangheluwe, 2015), which requires the user to slightly modify the model source code to distribute the simulation.

Finally, in this work, distributed simulations have been preferred to parallel simulations, because the last ones, typically faster, demand monolithic computer platforms, at prohibitive costs when the computation requirements are increased.

At this point, it is worth noting that several works use DEVS for problems related to ours. The closest contribution, by Bordón-Ruiz et al. (2021), introduces the evaluator for UAV-based target-search strategies that is used by the optimiser presented in this paper to determine which UAV and sensor trajectories are the best for a given scenario. Also related to the target-search problem are the works by Holman et al. (2010) and by Happe and Berger (2010). The first work models the target-search problem with Cell-DEVS and cellular automata that combine diffusion rules to update the probability map and high-climbing algorithms to optimise the UAV search-pattern. In contrast, the second one introduces a multi-UAV cooperative search path planner focused on coordination strategies and information-sharing policies. Our approach is different from both since it is intended to provide a common Cloud-deployable framework for determining the best UAV/sensor strategies for different types of target-search problems. In other words, within our framework we are planning to support different variants of target search problems and to efficiently compute the best UAV/sensor strategies for them, by taking advantage of xDEVS modelling flexibility and distribution capabilities. In addition, Hall (1997) introduces a simulator to evaluate the effectiveness of military missions involving autonomous vehicles; Moreno, de la Torre, Risco-Martin, Besada-Portas, Aranda et al. (2011) model the evaluation of UAV trajectories in hostile environments; Zeigler and Kim (2019) tackle multi-resolution modelling for exploratory analysis of complex and adaptive UAV service systems; and Pecker-Marcosig et al. (2020) present a unified DEVS-based platform to model and simulate hybrid control systems which is tested in mission involving UAVs. Hence, they are focused on other types or aspects of problems involving UAVs, while

this work is focused on presenting a distributable framework for target search missions. Finally, and as our DEVS-based framework also implements the optimisation of the UAV and sensor trajectories, we want to mention a few works that have successfully used DEVS to optimise problems of other fields of research (Cárdenas et al., 2020; Moreno, de la Torre, Risco-Martin, Besada-Portas, Aranda et al., 2011; Ntaimo et al., 2008; Pérez, 2017; Risco et al., 2008; Zeigler et al., 1996). This last group of works already shows how implementing the optimiser within DEVS is helpful to unify the modelling and implementation under the same paradigm.

After all this discussion, we want to highlight the main contributions of this work:

- *A new DEVS-based framework for optimizing different types of UAV and sensor strategies for different variants of target-search problems in large-scale realistically-modeled scenarios*. Although to test the framework, we have had to select a particular variant of the problem and implement specific behaviors for the sensors, UAVs and targets, the framework is designed to easily incorporate other behaviors and elements in the future.

- *A framework deployable in PCs or the Cloud*, without needing to modify its hierarchical structure or re-code the couplings and behaviors of its models. Hence, it can be used, depending on the computational requirements of the involved models, in different platforms.

- *A flexible and straightforward distribution allocation policy*, based on DEVS microservices and containerization paradigms. While the use-case is specific, this policy can be generalized to support the Cloud containerization and simulated of other complex systems modelled in DEVS.

The organisation of this paper is the following. This first section has introduced the paper and conducted a review of the state of the art related to it. Section 2 introduces the technologies that support the distribution of DEVS-based models in the Cloud. Section 3 describes the architectural and behavioural specifications of the framework. Section 4 presents the tools and policy followed to deploy the framework in the Cloud. Section 5 analyzes the results obtained by the framework over a realistically modelled scenario and the computational speedup associated with its Cloud distribution. Finally, Section 6 draws the conclusions and introduces future lines of research.

2. Foundational technologies

Our framework must be able to execute simulations and optimisations in a distributed environment. To this end, we have selected a container-based distributed architecture due to its potential and configuration simplicity. In this section, we describe the technologies involved in performing distributed simulations based on microservices, containerisation paradigms, and DEVS.

2.1. Microservices paradigm

A microservice is a basic element that results from the architectural decomposition of an application into loosely coupled patterns that consist of self-contained services, which communicate with each other primarily through asynchronous event-driven mechanisms. To do it, they use a standard communication protocol and a set of well-defined Application Programming Interfaces (APIs) independent of any vendor, product, or technology (Mittal et al., 2017).

Moreover, any microservices-based architecture has to address two fundamental issues: distributed data management (to store the state of the microservice locally) and shared event processing (to facilitate the information exchange between stateless microservices). Finally, to execute their inherent business logic, the information from the local data and the event processing inside the microservice are used together.

2.2. Containerisation paradigm

A container is an independent self-sufficient runtime environment for a software application. It wraps a piece of software in a complete file-system that contains everything needed to run. Relevant features of these containers are: (1) lightweight; (2) sharing the same OS kernel, common files and layering; (3) based on open standards; (4) running on all major Operating Systems (OS) and on top of any infrastructure; (5) isolating applications from others and from the underlying infrastructure; (6) composable; and (5) predictable (opensource.com, 2021).

The architectural approach of containerisation is different from that of virtual machines. The differences are illustrated in Figure 1. In particular, Figure 1a shows how a virtual machine stores the whole Operating System (OS), libraries, needed binaries and applications, requiring a huge memory space in the host machine. Figure 1b shows how a container is composed by libraries, required binaries and applications; and how all the containers share the same OS kernel. This makes containers a lightweight and flexible tool for packaging, delivering, and deploying software and applications.

2.3. DEVS distributed architecture

The parallel Discrete Event System Specification (DEVS) is a modular and hierarchical formalism for discrete event system modelling based on set theory (Zeigler et al., 2018). It includes two types of models, atomic and coupled, which have an interface consisting of input (X) and output (Y) ports to communicate with other models. Additionally, in atomic models, every model state (S) is associated with the time advance function ta, which determines the duration in which the state remains unchanged. Once the time

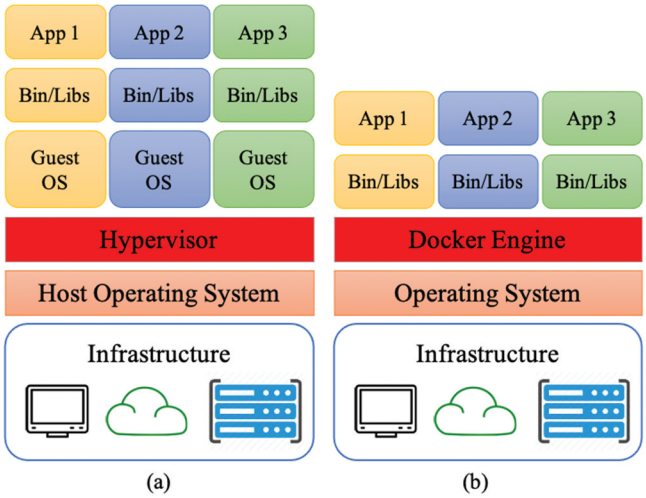


Figure 1. Graphical representation of (a) virtual machine and (b) docker container architectures.

assigned to the state has passed, an internal transition function ($\delta_{\text{int}} : S \rightarrow S$) is fired and an internal transition is triggered, producing a local state change ($\delta_{\text{int}}(s) = s'$). At that moment, the model execution results are spread through the model's output ports by activating an output function (λ). Furthermore, input external events (received from other models) are collected in the input ports. An external transition function ($\delta_{\text{ext}} : S \times e \times X \rightarrow S$) specifies how to react to those inputs, using the current state (s), the elapsed time since the last event (e) and the input value (x) ($\delta_{\text{ext}}((s, e), x) = s'$). Parallel DEVS introduces a confluent function ($\delta_{\text{con}}((s, ta(s)), x) = s'$), which decides the next state in cases of collision between external and internal events. Coupled models are the aggregation/composition of two or more models (atomic and/or coupled), connected by explicit couplings. This makes DEVS closed under coupling and allows to use networks of systems as components in larger coupled models, leading to hierarchical and modular constructions. Overall, DEVS provides a framework for information modelling that has several advantages to analyse and design complex systems: completeness, verifiability, extensibility, and maintainability.

Once a system is described according to DEVS theory, it can be easily implemented using one of the many DEVS Modelling and Simulation (M&S) engines that have come into existence in the last decades. All of them offer a programmer-friendly API to define new models using a high-level language, but only a few provide a user-friendly API for parallel or distributed model simulation. Among them, xDEVS (Mittal et al., 2017) offers a good alternative to parallelise or distribute simulations in the Cloud, following the microservices architecture and containerisation detailed above. As a result, any DEVS model can be parallelised or distributed by assigning resources (threads or processes) to different transition and output functions as parallel or distributed functional programming.

The microservices-based DEVS simulation execution is explained with the help of the classic Experimental Frame – Processor (EF-P) model. This model, represented in Figure 2a, contains two components: the Experimental Frame (EF) coupled model and the Processor (P) atomic model. In order to simulate it in the Cloud, this hierarchical model is automatically flattened by xDEVS, removing all the coupled models, in order to obtain the Generator – Processor – Transducer (GPT) equivalent model depicted in Figure 2b. Next, and according to Figure 2c, each model in GPT is mapped to a DEVS container following the instructions of a simulation configuration file. Moreover, one container can allocate one or more DEVS atomic models with their corresponding local states. In the example illustrated in Figure 2c, a different container/simulator is created for each atomic model of the GPT, and one of the containers is designated as the root coordinator responsible of driving the whole simulation. Furthermore, models' state transitions and output events are always managed by their corresponding simulators, and model output events are propagated to their neighbours using socket communication between model-simulator pairs. Finally, note that since the distributed architecture is implemented at the simulation layer of xDEVS, any xDEVS model can be simulated in the Cloud without changing its code. Finally, it is worth mentioning that although the parallelisation or distribution is oriented to transition or output functions, DEVS closure-under-coupling property allows us to easily rearrange parallel or distributed layouts. Following the EF-P example, the EF

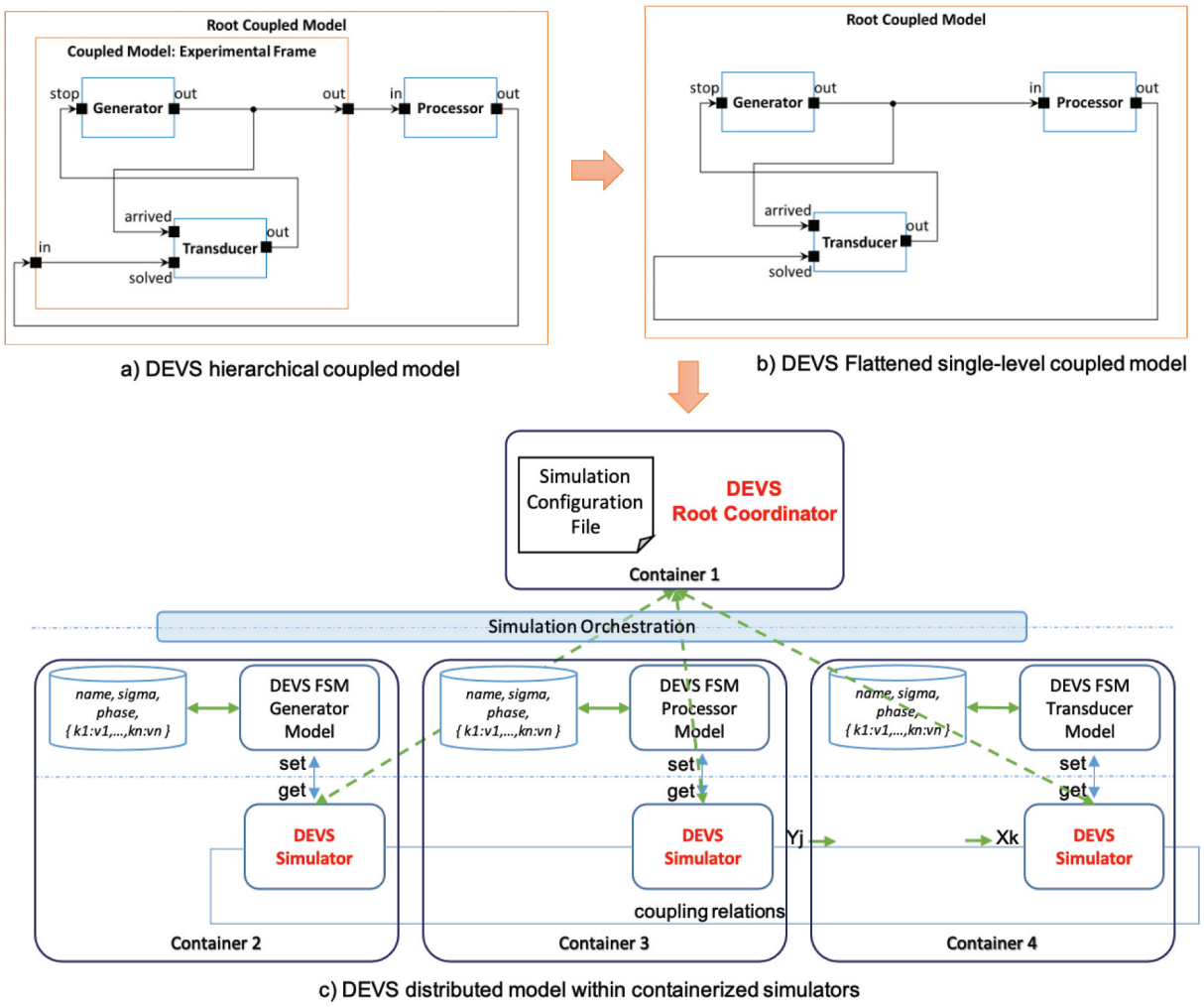


Figure 2. DEVS containers within microservices and containerisation paradigms. adapted from (Mittal et al., 2017).

coupled model could be transformed into a single atomic using the Coupled2Atomic xDEVS wrapper, which allows us to parallelise the EF transition or output functions as a whole, instead of as two atomic function sets.

3. Models description

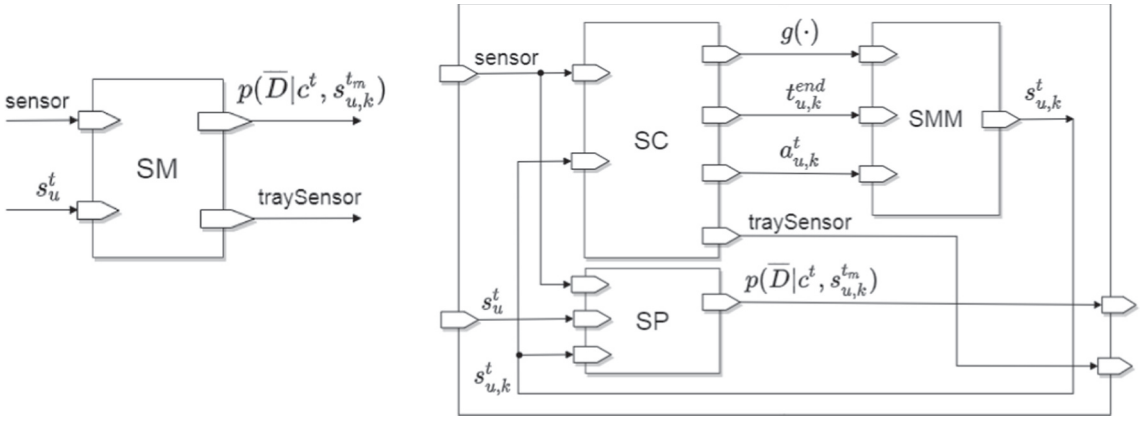
This section presents the main characteristics of the models of our framework from a bottom-up perspective. As the framework incorporates the modules of the evaluation process already presented in Bordón-Ruiz et al. (2021), the descriptions in Sections 3.1, 3.2, 3.3, 3.4, 3.5 and 3.6 are lighter, while the descriptions in Sections 3.7 and 3.8 of the new modules required to optimise the UAV and sensor trajectories are deeper.

3.1. Sensor Model (SM)

In the target-search problem, sensors on board UAVs are used to obtain information from the search area Ω, which is discretised in a grid G of w_x x w_y rectangular cells. The detection capabilities of the sensor are defined by the likelihood model $p(D|c^t, s_{u,k}^{t_m})$, which states how probable is to observe the target placed at cell c^t at time t from the UAV location and sensor pose $s_{u,k}^{t_m}$ at measuring time t_m. Additionally, to describe the behaviour of moving sensors (e.g., a gimballed camera), a deterministic motion model $s_{u,k}^t = g(s_u^t, s_{u,k}^{t-T_{u,k}}, a_{u,k}^{t-T_{u,k}}, T_{u,k}, \in^{t-T_{u,k}})$ obtains the new sensor pose $s_{u,k}^t$ based on its UAV location s_u^t, its previous pose $s_{u,k}^{t-T_{u,k}}$, its control signal $a_{u,k}^{t-T_{u,k}}$ and duration $T_{u,k}$, and the environment condition $\in^{t-T_{u,k}}$.

The specification of each sensor is represented by the Sensor coupled Model (SM) of Figure 3a, while its internal structure is depicted in Figure 3b and described below:

- The Sensor Payload atomic model (SP) computes $p(D|c^t, s_{u,k}^{t_m})$ and returns its complementary function $p(\overline{D}|c^t, s_{u,k}^{t_m})$ at every measuring time t_m of the sensor.
- The Sensor Motion atomic model (SMM) executes $g(\cdot)$ at the time lapses defined by the user to output the new sensor poses $s_{u,k}^t$.



(a) Specification (b) Internal Structure

Figure 3. Sensor coupled model (SM).

• The Sensor Control atomic model (SC) handles the sensor moving control list, receives every new $s^t_{u,k}$, and wraps them into the sensor pose trajectory traySen that is output when the simulation ends.

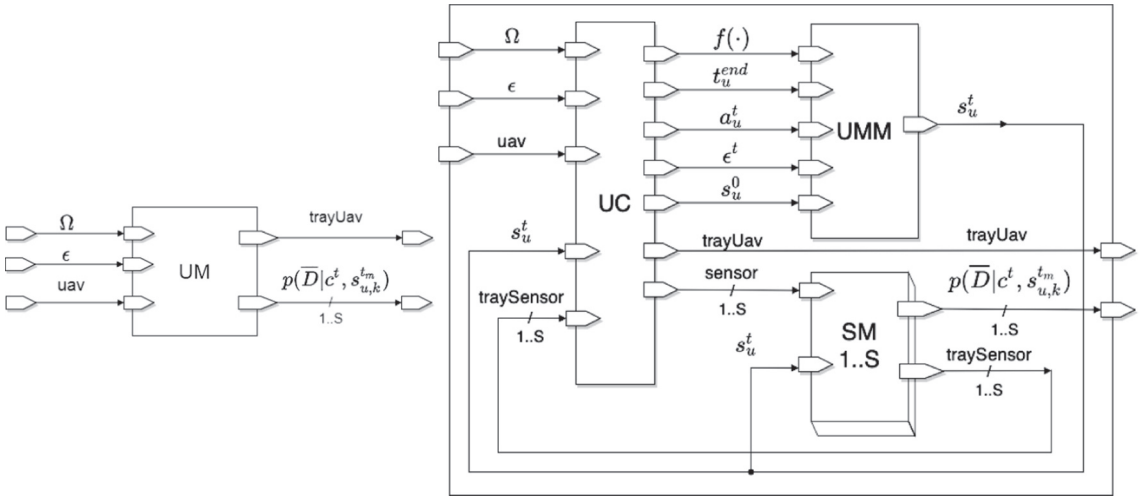
In addition, as static sensors do not require motion models and control lists, their SM models are simplified into SP models.

Eventually, it is worth noting that evaluating $p(D|c^t, s^{t_m}_{u,k})$ for realistically modelled sensors is often more computationally demanding than calculating $g(\cdot)$. In particular, the camera likelihood in this paper is high-time consuming (because it evaluates $p(D|c^t, s^{t_m}_{u,k})$ in many points within the sensor footprint for every sensor measurement), the radar likelihood is lighter (as it reuses the likelihoods of previous time steps, displaced to the new sensor observation area), the camera motion $g(\cdot)$ computation time is negligible, and the radar motion ignored.

3.2. UAV Model (UM)

UAVs overfly Ω looking for the target. Their motion dynamics are summarised in $\in^{t-T_{u,k}}$, which allow to obtain the new UAV location s^t_u from its previous one $s^{t-T_u}_u$, its control signals $a^{t-T_u}_u$ and durations T_u, and $\in^{t-T_u}$.

Each UAV in the scenario creates an instance of the UAV coupled model (UM). Its inputs are, according to Figure 4a, the search area Ω, the environmental state $\in$ and the encapsulated UAV definition uav. Its outputs are the simulated UAV trajectory trayUAV, which includes traySensor of its moving sensors, and $p(D|c^t, s^{t_m}_{u,k})$. Its internal structure, in Figure 4b, includes two atomic models: UAV Control (UC) and UAV Motion (UMM), conceptually equivalent to the sensor SC and SMM models. Additionally, it aggregates as many SM models as detection sensors have the UAV.



(a) Specification (b) Internal Structure

Figure 4. UAV coupled model (UM).

Finally, the computational requirements of $f(\cdot)$ are highly dependent on the complexity of the UAV dynamical modelled within it. In particular, in this paper $f(\cdot)$ implements a 4th Order Runge Kutta to integrate a lightweight model with streamlined dynamics for the UAV height, speed, and lateral displacements.

3.3. Flight Simulator (FS)

The Flight Simulator coupled model (FS) is defined, as Figure5 shows, by aggregating as many UM models as UAVs. It computes all UAV trajectories and sensor likelihoods, and inherits the computational requirements of its components (UAVs and sensors).

3.4. Target Model (TM)

The uncertainty about the target location is modelled by a probability map or belief $b(c^t)$ over each cell $c^t \in G$. The initial belief $b(c^0)$ is known and the target evolution is computed using the prediction and assimilation step of the Recursive Bayesian Filters (RBF, Bourgault et al. (2004)) over the "remaining or unobserved probability" $p(c^t)$, related to $b(c^t) = p(c^t)/\sum_{g\in G} p(c^t = g)$. In particular, the prediction $p(c^t) \leftarrow \sum_{c^{t-T}\in G} p(c^t|c^{t-T_\tau})p(c^{t-T_\tau})$ considers the target uncertainty motion model $p(c^t|c^{t-T_\tau})$, while the assimilation $p(c^t) \leftarrow p(\overline{D}|c^t, s^{t_m}_{u,k}) \cdot p(c^t)$ incorporates the computed likelihoods.

The operations are carried out by the Target coupled model (TM) of Figure 6a, which contains the two atomic models of Figure 6b:

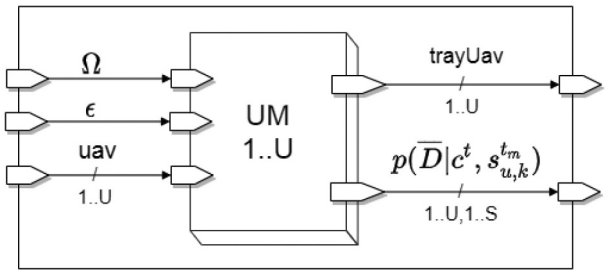


Figure 5. Flight Simulator coupled model (FS).

• The Target Control (TC) updates $p(c^t)$ by: a) performing the assimilation step integrating the receive $p(\overline{D}|c^t, s^{t_m}_{u,k})$ in $p(c^t)$, or b) receiving the $p(c^t)$ predicted by the Target Motion model. It also wraps $p(c^t)$ into the encapsulated target evaluation eTarget.

• The Target Motion (TMM) performs the prediction step, applying the target motion model. It is only used by dynamic targets and removed with its corresponding couplings for static targets.

The computational demands of these models grow with the resolution (number of cells in G) of the search space, the number of sensors and measurements taken, and the number of target displacements.

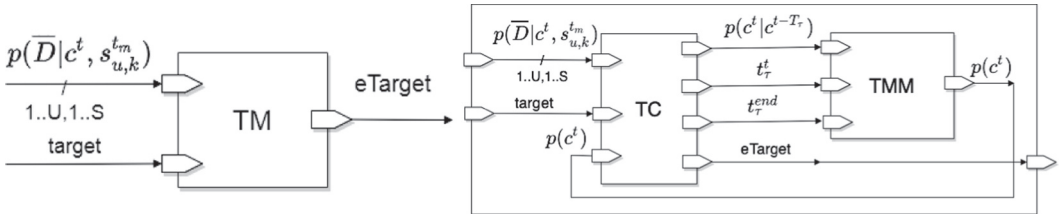
3.5. Evaluator Function (EF)

To evaluate the simulated UAV trajectories and sensor poses, different utility functions are used. They are computed in the Evaluator Function atomic model (EF), which receives the simulated trayUAV and eTarget, and calculates the probability of target detection ($P_d(t) = 1 - \sum_{c^t\in G} p(c^t)$), the Expected Time of detection ($ET(t) = \sum_{l=1:1:t/T_\tau}(1 - P_d(l \cdot T_\tau))T_\tau$), the number of Non-Flying Zones overflights ($\text{NFZ} = \sum_{t=1}^{T}\sum_{u=1}^{U} WithinNFZ(s^t_u)$), the number of UAV collisions ($\text{COL} = \sum_{t=1}^{T}\sum_{u=1}^{U}\sum_{l=k+1}^{U} Collision(s^t_u, s^t_l, d_{COL})$) and the MYOpia criterium at certain times of the simulation ($\text{MYO}(t) = \sum_{c^t\in G} l(c^t, s^t_{1:U,1:K})p(c^t)$ with $l(c^t, s^t_{1:U,1:K}) = \prod_{u=1:U,k=1:S} h(c^t, s^t_{u,k})$) and $h(c^t, s^t_{u,k})$ a radial monotonically increasing function in [0,1], whose minimum is centred at $s^t_{u,k}$ and whose purpose is to inform how good are the final UAV and sensor locations for collecting the remaining $p(c^t)$).

At last, note that the computational requirements of this module are moderated, as it just performs some final operations over the results provided by the Flight Simulator and Target Model.

3.6. Evaluator (EV)

The Evaluator coupled model (EV) encapsulates the TM and EF models, as Figure 7 shows, inheriting their computational requirements (especially those of TM).



(a) Specification (b) Internal Structure

Figure 6. Target coupled model (TM).

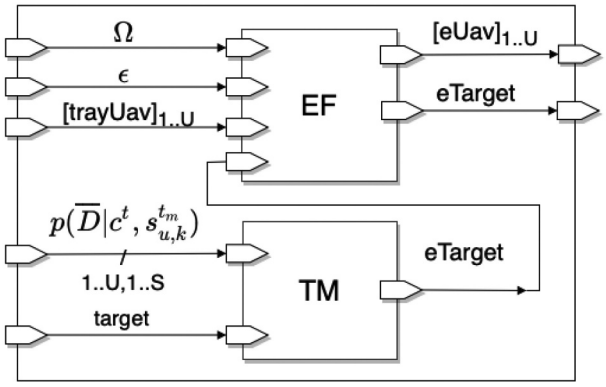


Figure 7. Evaluator coupled model (EV).

In short, it is where the target and utility functions are evaluated. Finally, it returns the encapsulated result of the evaluation (eUAV and eTarget).

3.7. Optimiser (OP)

The Optimiser coupled model (OP) introduced in this paper extends the capabilities of the evaluation framework of UAV strategies for target-search scenarios by Bordón-Ruiz et al. (2021), built over the coupling of the Flight Simulator (FS) and Evaluator Function (FC) models. The new OP model is in charge of automatically constructing the UAV and sensor trajectories, by implementing the steps of an iterative population-based optimisation algorithm capable of handling the complexity of target-search problems.

The OP model, depicted in Figure 8, encapsulates the generic steps of the optimiser in the Algorithm Controller atomic model (AC), while the UAV-strategy target-search evaluation process is performed by the FS and EV models. In particular, at every iteration of the algorithm implemented within AC, a population of N_{pop} solutions (UAV and sensor trajectories) is generated and sent to the aggregation of N_{pop} FS and EV models, which in pair carry out the evaluation of each solution.

In addition, the outer loop of the current version of OP works as the receding horizon-controller in Pérez-Carabaza, Besada-Portas et al. (2019), which a) optimises all the control signals to be applied to the UAVs and sensors during the decision horizon T_{DH}, and b) considers the final $p(c^t)$, $s^t u$ and $s^t_{u,k}$ of each trajectory section, the initial values of the next. This philosophy, commonly used in target-search, breaks down the problem into smaller pieces that are optimised sequentially, reducing the space of possible solutions.

Finally, note that there is not a direct feedback among EV and FS (or between the UM within FS), since their purpose is to evaluate the solutions proposed by the optimisation approach implemented within AC. Besides, our approach consists in including the optimiser within the model-based framework, as it helps us unify all the process in DEVS and distribute it in the Cloud.

3.8. Algorithm Controller (AC)

The AC model receives the scenario specification, which includes the definition of every element of the mission (e.g., Ω dimensions; target probability models; number, initial location/pose and models of the UAVs/sensors, and mission duration $t^{end}_{mission}$).

Besides, it receives the optimiser specification, which includes the type and parameters of the optimisation algorithms, the optimisation and feasibility criteria to be used (ET, $P_d(t)$, MYO, NFZ and COL), and the decision horizon parameters (T_{DH}, number of iterations and computation time for each trajectory section). At last, when $T_{DH} < t^{end}_{mission}$, the AC model optimises the UAV and sensor trajectory by sequences, and MYO should be included to determine how well the search can continue from the final state of the current trajectory section.

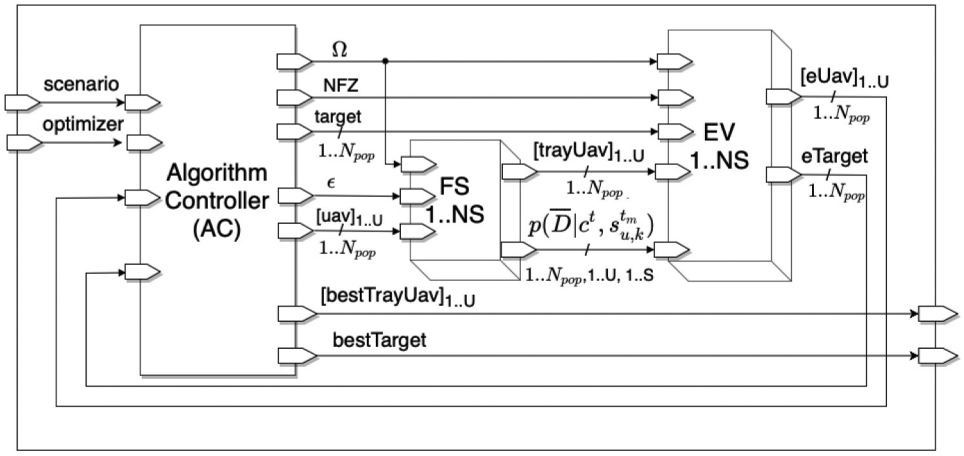


Figure 8. Optimiser coupled model.

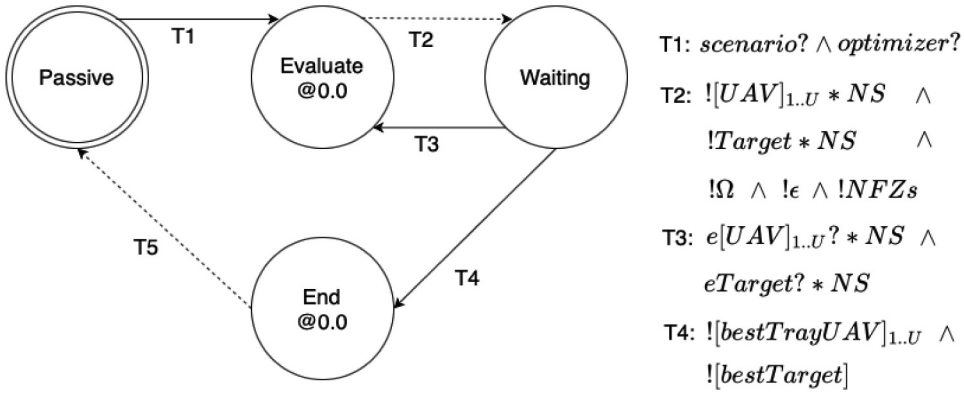


Figure 9. Algorithm Controller States.

The AC behaviour is described in Figure 9. Initially, the model waits for the scenario and optimisation definitions in a passive state. At their reception, AC prepares the initial sequence by analysing the scenario specification and determining the trajectories of which UAVs and sensors should be optimised already. Next, for each UAV u and sensor u_k included in the current section, AC generates N_{pop} solutions, consistent on the list of random reference signals (UAV orientation, height and speed, and sensor azimuth and elevation angles respect the UAV body-axes) to be applied periodically (at a fixed rate) or at the time-steps randomly generated by AC. When this last behaviour is enabled, each solution of the population can have a different number of decision variables. Once the first set of N_{pop} solutions is ready, a δ_{ext} transition takes AC to the evaluation state, where an instant δ_{int} transition is programmed to output via the λ function the N_{pop} solutions to be evaluated by their FS and EV models. Next, AC remains in a waiting state until the all the evaluation results are received. At this stage, AC checks if the maximal number of iterations and execution time of the current decision horizon has been reached. If it has not, AC iterates the operations of the optimisation algorithm to produce another group N_{pop} solutions, sends them to be evaluated and waits for the results. Otherwise, AC terminates the optimisation of the current sequence, stores the best solution so far, and uses it to obtain the initial state of the next sequence. When the optimisation of the last sequence is finished, AC transitions to its end state, outputting the whole best solution via the λ function.

The previous behaviour is designed to operate as a receding controller capable of incorporating the optimisation steps of different population-based algorithms. For this paper, it uses a Genetic Algorithm (GA) that implements: 1) binary tournament selection, 2) a crossover that determines the cutting point of each parent based on the crossing time (selected from a uniform probability between the times of the decision horizon), 3) a two-level mutation (consisting of two Gaussian increments of the decision variables, the first in a few uniformly-randomly selected decision variables and the second one – with smaller variance – in all of them), and 4) the recombination step of NSGA-II (Deb et al., 2002). Further details of steps (1), (3) and (4) are presented in (Pérez-Carabaza, Besada-Portas et al., 2019), while the new step (2) is conceptually similar to the ones in (Andres-Toro et al., 2004; Kiam et al., 2021). In addition, within our framework, step (4) is implemented before step (1), because the optimisation step iterates every time the whole set of N_{pop} solutions is received.

Finally, note that the computational requirements of the AC model are low, as the evaluation process is performed by the FS and EV models placed outside it. Moreover, as AC requests the N_{pop} FS and EV coupled models to evaluate the partial solutions at each iteration of the optimisation of each segment of the UAV and sensor trajectories, the overall computational cost of FS and EV grows proportionally to the number of segments and optimisation iterations within them.

4. Cloud simulation deployment

This section describes the Cloud deployment architecture. Broadly speaking, the original DEVS model is flattened, next each atomic model is allocated into a Cloud container, and finally, the simulation is executed into a container-based distributed infrastructure. In the following, we detail how this procedure, summarised in Figure 10, is carried out over our framework, which has been developed using the xDEVS simulation engine (Risco-Martín et al., 2017).

First, an XML description of a flattened version of the original model is generated with xDEVS. This XML file contains all the atomic models and coupling relations that are obtained after rearranging the connections of the coupled models, which are removed because they accumulate deployment difficulties in distributed simulations (Risco et al., 2008). Next, the user must edit this XML file to configure the allocation of all

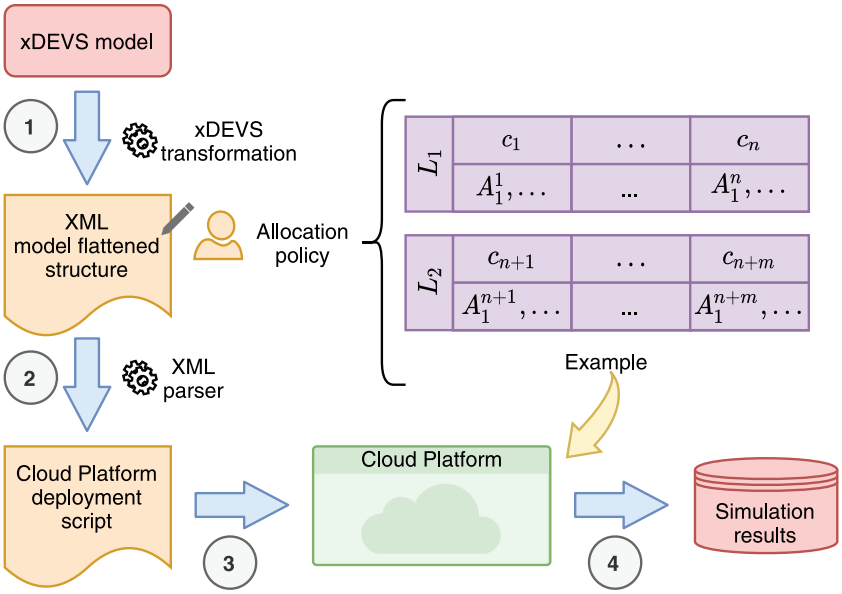


Figure 10. Cloud deployment scheme.

the atomic models. This step is important because it decides the distributed simulation structure. Although each atomic model can be allocated into a single container, this option is not operative since a DEVS model can contain hundreds of atomic models, many of them computational light in terms of CPU cycles. Thus, it is better to group several atomic models per container. Figure 10 shows the 2-level allocation policy followed in this paper, consistent of n containers, placed at level 1 (L_1), reserved for those atomic models with high computational demands; plus m containers, placed at level 2 (L_2), reserved for the others. In general, $n >> m$ in order to follow a coarse-grain allocation policy that a) exploits the modeller knowledge about which atomic models consume more CPU, and b) avoids a computing-intensive profiling phase.

Second, once the allocation policy is described inside the XML file, a script (specific for the cloud platform where the distributed model is going to be simulated) is generated with an XML parser. In a xDEVS distributed simulation, each atomic model is executed inside its simulator as an isolated process, while the coordinator marks the beginning and end of the simulation, as described in Figure 2. The communication between simulators is made using sockets. As a result, any distributed architecture is possible (e.g., using a set of computers, a set of containers, a set of virtual machines, or a combination of all of them). For this particular work, we have developed a parser that generates the script to deploy a distributed simulation over a set of containers in the Google Cloud Platform (GCP), using the Google Kubernetes Engine (GKE).

Third, the script is executed against the selected cloud platform, and the model is deployed using the selected allocation policy. In this step, every container stores the whole flattened atomic model. Still, each container will run one or more distributed xDEVS simulators, each one with its corresponding atomic model as defined in the XML file. Besides, one particular container runs the distributed xDEVS root coordinator, and for simplicity, even between the simulators executed in the same container, messages are passed using sockets.

Finally, once the simulation ends, the results are stored in a distributed way, as each atomic model can have a different mechanism to save data. Our recommendation is to have a *Transducer* atomic model (Zeigler et al., 2018) to collect the relevant information to store it, if necessary, in the container where it was allocated.

To anticipate the dimensions of the four strategies simulated in this work (named S1, S2, S3, and S4 here after) and understand how they are distributed among the containers, Table 1 shows the number (#) of atomic models instantiated in the xDEVS model and the percentage (%) of the accumulated execution time of all the atomic models that belong to each class. It also shows that Target Control and Radar Sensor

Table 1. Percentage of execution time/scenario and atomic class.

Atomic Class	S1		S2		S3		S4	
	#	%	#	%	#	%	#	%
Radar Sensor Payload (SP)	50	**26**	50	6	50	6	50	5
Camera Sensor Payload (SP)	NA	NA	100	**34**	100	**35**	100	**35**
Camera Sensor Control (SC)	NA	NA	100	0	100	0	100	0
Camera Sensor Motion (SMM)	NA	NA	100	1	100	1	100	1
UAV Control (UC)	50	1	100	1	100	1	100	1
UAV Motion (UMM)	50	2	100	1	100	1	100	1
Target Motion (TMM)	50	10	50	2	50	2	50	2
Target Control (TC)	50	**55**	50	**50**	50	**52**	50	**52**
Evaluator Function (EV)	50	6	50	5	50	2	50	3
Algorithm Controller (AC)	1	0	1	0	1	0	1	0
TOTAL	301	100	701	100	701	100	701	100

Payload assume more than 75 % of the total execution time for S1, while Camera Sensor Payload and Target Control do something similar for the others. Based on this information, we have decided to add all the atomic models that belong to these three classes to level L_1 of our allocation policy, and the remaining atomic models to L_2. In the following section, we will see that adding containers to L_1 has a high impact on the execution time, while adding more containers to L_2 has almost no effect.

Finally, note that the previous statistics have been calculated using the Coordinator named CoordinatorProfile in xDEVS, which registers the execution times for each DEVS function and atomic model. However, to distribute larger models quickly, we recommend to follow a coarse grain classification of atomic classes as we are applying in this example because the simulation of these models can take several hours, but the modeller usually knows (as can be observed in Section 3) which classes are more CPU demanding.

5. Results

The simulations conducted in this paper are focused on demonstrating the benefits of deploying complex simulation models in the Cloud. To do it, we optimise different strategies over a real-world inspired scenario and compare the simulation time required by distributing their models over different numbers of containers. We have also executed fully parallel simulations on two different shared memory machines to compare performance and cost against the proposed distributed approach.

5.1. Scenario

The scenario is inspired by a search and rescue mission at the sea, where a boat is missing within a search area of 60 × 60 km^2, discretised into 200 × 200 cells of 300 x 300^2.

The initial target $b(c^0)$ is presented in Figure 11 (a), while its displacement towards the north-east is modelled with a $p(c^t|c^{t-T_\tau})$ applicable every $T_\tau =$ 150 s. The effects of the target displacement after $t^{end}_{mission} = 3000$ s are represented in Figure 11(b), although difficult to appreciate as the target barely moves 6 km.

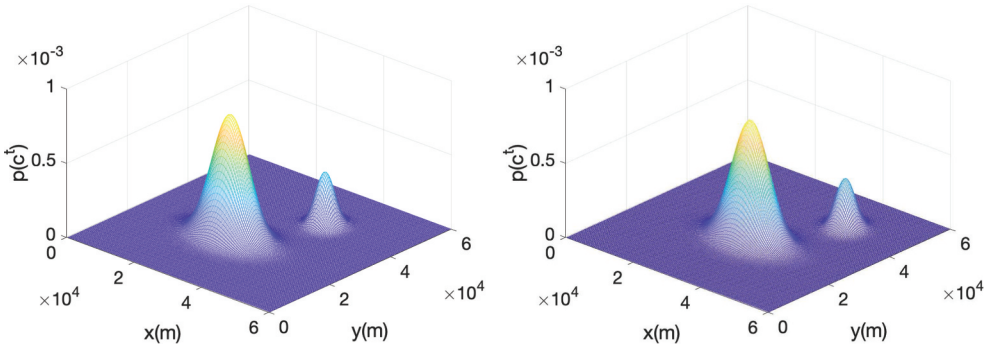
Two UAVs are available to carry out the mission. Their main characteristics are summarised in Table 2. Their motion model is described in (Pérez-Carabaza, Besada-Portas et al., 2019) and integrated every 1 s.

Regarding the sensors, UAV1 is equipped with a continuous-wave radar and UAV2 has two high-resolution cameras. Its first camera is static and placed at the UAV noise pointing ahead with a fixed elevation (from the aircraft longitudinal axis) of 30 deg, while the second one is mounted at the UAV bottom on a gimbal, whose motion model is integrated with a 4th Order Runge-Kutta every 2 s, has a maximal slew rate of 5 deg/s at an azimuth range of 360 deg, and a fixed elevation of 70 deg. Their $p(D|\tau^t, s^t_u)$ depends, as Figure 12 shows, on the distance between τ^t and s^t_u and, in the camera case, on the sensor footprint that is modifiable by the sensor field of view (fixed at 7.5deg), UAV height and sensor pose.

5.2. Search strategies and optimiser setup

In order to determine the best UAV and sensor trajectories for the previous scenario, we set up the following four strategies:

- Strategy 1 (S1) uses only UAV1, a decision horizon $T_{DH} =$ 1000s and a mission duration $t^{end}_{mission} =$ 3000s.
- Strategy 2 (S2) incorporates also UAV2 to the mission and makes $T_{DH} =$ 500 s.
- Strategy 3 (S3) is similar to S2, but with $T_{DH} =$ 1500 s.



(a) Initial Belief $b(c^0)$ (b) Final $p(c^t)$ at $t = 3000$s

Figure 11. Mission Belief.

Table 2. UAVs characterisation.

Description	UAV1	UAV2
Flying height range (ft)	3000	3048
Flying speed range (kts)	77.8	87.5
Operating range (nm)	135	80
Flying autonomy (hours)	15	6.5
Motion Model	Runge-Kutta (1s)	Runge-Kutta (1s)
Payload	Radar (4s)	Camera 1 (1s), Camera 2 (4s)

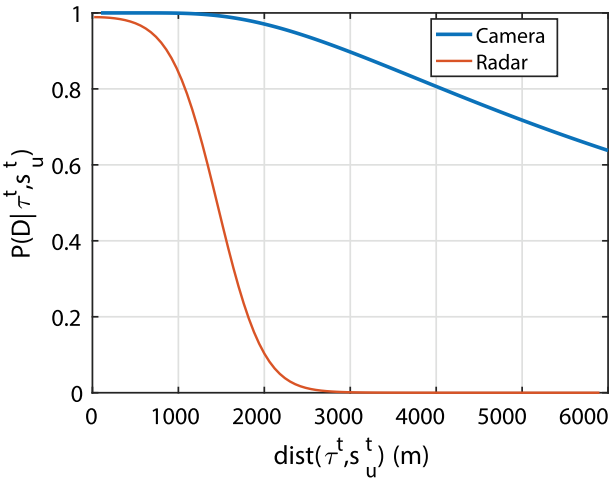


Figure 12. Sensor Likelihoods.

Table 3. GA characterisation per decision horizon.

Parameter	Configuration
Population Size (N_{pop})	50
Iterations	100
Crossover Probability	$p_x = 0.8$
Mutation Probabilities	$p_{mut} = 1/\#_{\text{Decision Variables}}$, $\sigma_1 = 200$, $\sigma_2 = 1$

• Strategy 4 (S4) is similar to S3, but with T_{DH} = 1000 s and $t^{end}_{mission} = 4000$ s.

In short, we set up S1 to see how well UAV1 can accomplish the mission on its own, and the other strategies to see the effect of T_{DH} in the quality of the solutions, and if increasing T_{DH} and $t^{end}_{mission}$ in S4 allows to obtain a better $P_d(t)$ and ET.

The remaining parameters of the optimiser, common for all the strategies, are:

• The decision variables of the optimiser are:

° The headings of both UAVs, defined as aperiodic incremental control variables within [−30, +30]deg.

° The camera azimuth, defined as a control signal with absolute value within [−90, +90]deg that changes every 10s.

• The constraint is COL and the optimisation criteria (in the given preference order for deciding which is the best final solution of each decision horizon) are MYO, ET and P_d.

• The GA properties in Table 3.

5.3. Computation cost analysis

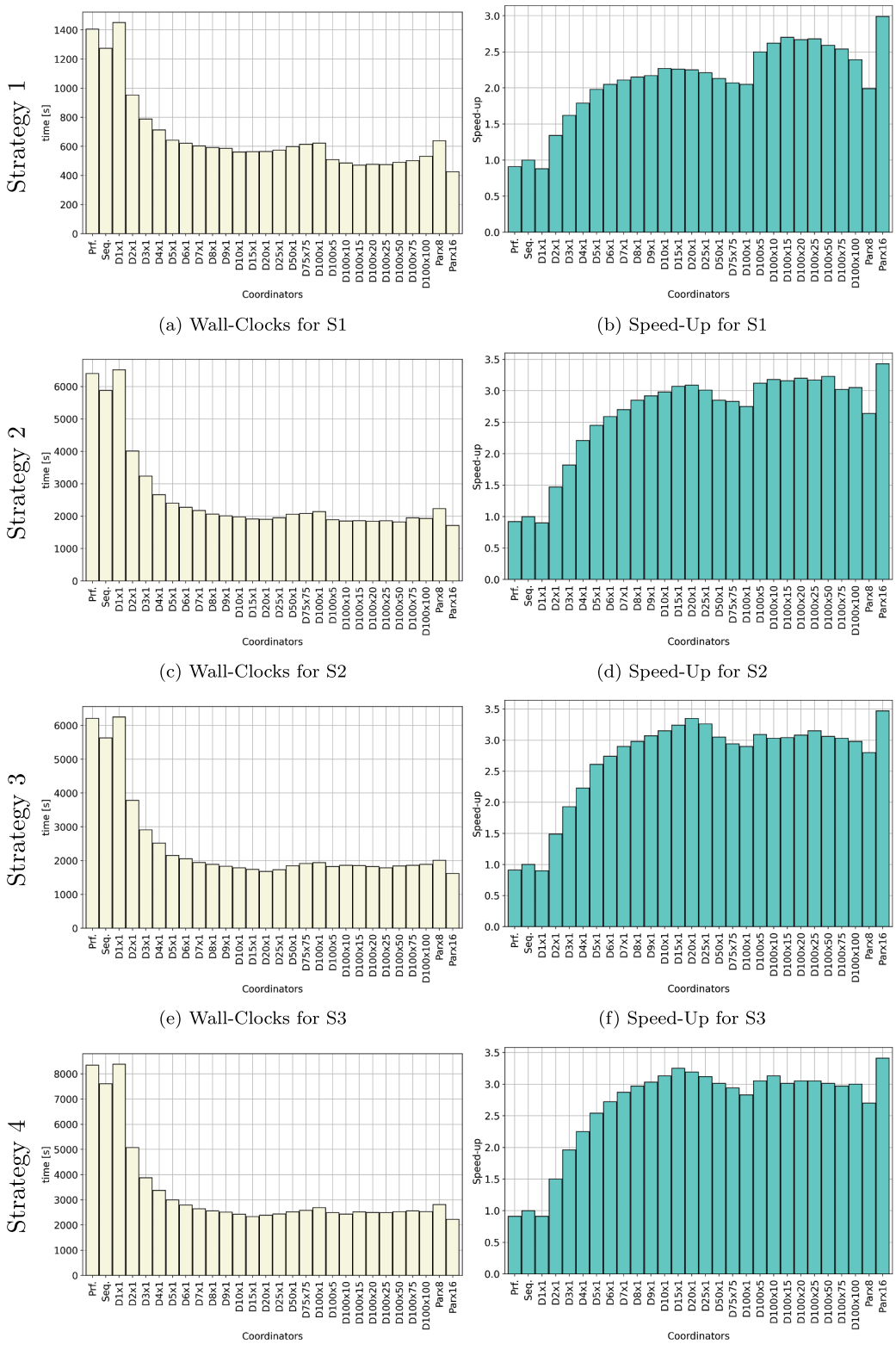
The analysis in this section is based on the containers distribution explained in Section 4. Likewise, the optimisation of the four strategies is carried out on GKE.

To determine how the Cloud deployment affects the computational time, an incremental container strategy is followed by 1) increasing the number of L_1 containers from 1 to 100 using a single L_2 container and 2) fixing 100 containers in L_1 while increasing L_2 from 1 to 100. We increase first the containers of L_1 because the models with higher computational costs are placed in that layer. Besides, these distributed simulations have been performed using a Kubernetes cluster with 16 Intel(R) Xeon(R) CPU @ 2.30 GHz and 4 GiB RAM nodes.

To compare the distributed simulations against parallel ones, we have simulated the same scenarios in parallel, using 8 and 16 Intel(R) Xeon(R) CPU @ 2.30 GHz shared memory machines, with 32 and 64 GiB of RAM, respectively. Hence, the 16 CPU machine was set up to have as many CPUs and the same total RAM as the sum of the resources distributed in the cluster.

Figure 13 illustrates the results of the analysis (further detailed in Table 4) using each row of graphics for a different strategy, the left column for the Wall-Clocks (WCs) and the right one for the Speed-Up (SU). The ordinate axis of the graphics at the left show that the WC of S2, S3, and S4 is significantly higher than those of S1, since the last three cases require computing the likelihoods and target updates of the two cameras. Besides, within each graphic, the first two bars (labelled *Prf.* and *Seq.*) correspond to regular/sequential simulations, executed in GKE without exploiting its distribution possibilities, respectively, using a) xDEVS CoordinatorProfile to perform a profiling of the simulation, and b) xDEVS Coordinator to obtain the reference to compute the speedup. For the following bars, named *Dixj* with $i = ContainersIn(L_1)$ and $j = ContainersIn(L_2)$, the simulation is executed using xDEVS CoordinatorDistributed. The last two bars, labelled Parx8 and Parx16, show the results of the fully parallel simulations in 8 and 16 CPUs, respectively, both using xDEVS CoordinatorParallel class, which takes the root coupled model, flattens it and parallelises the executions of its transition and output functions, similarly as in the distributed simulation but using the available threads, with no levels.

Within the group of distributed simulations *Dixj*, we can observe that as soon as the containers in L_1 grows, the WCs are significantly reduced and the SU increased. The improvements associated with increasing L_1 containers stop when the communication workload of the distributed models becomes significant. The optimal configuration with a single container in L_2 happens when there are 10/20/20/15



(a) Wall-Clocks for S1 (b) Speed-Up for S1

(c) Wall-Clocks for S2 (d) Speed-Up for S2

(e) Wall-Clocks for S3 (f) Speed-Up for S3

(g) Wall-Clocks for S4 (h) Speed-Up for S4

Figure 13. Computational analysis.

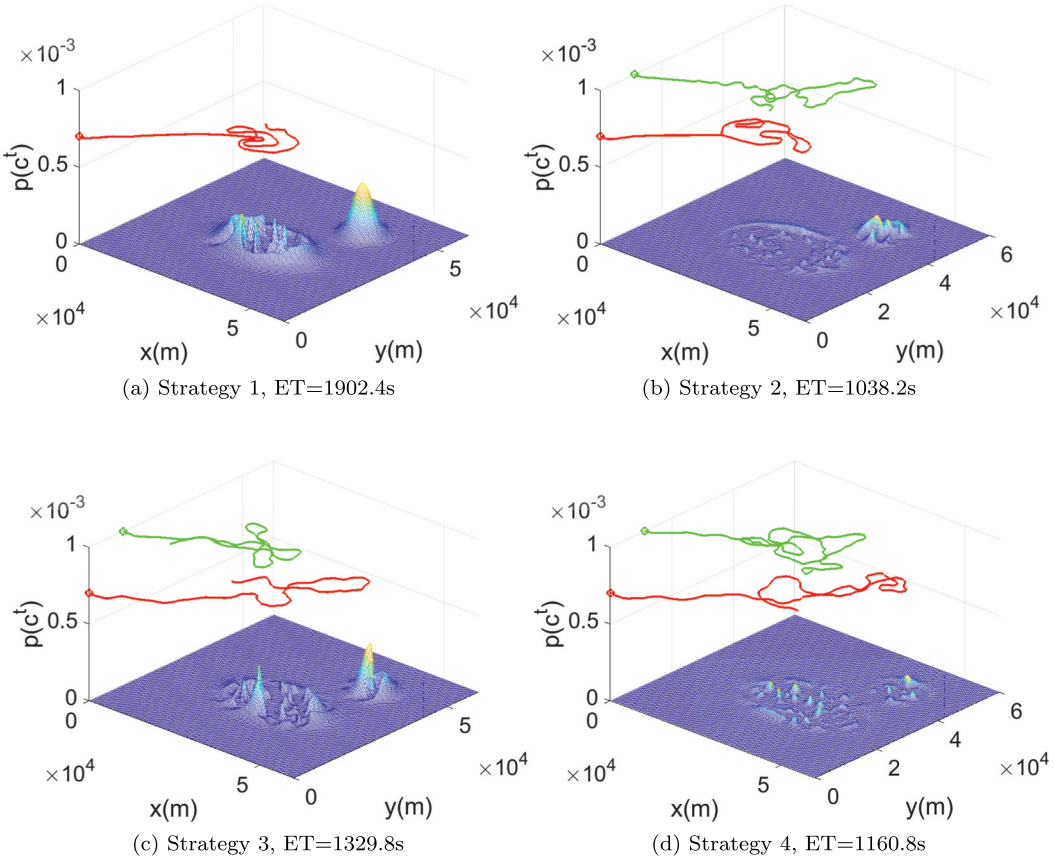
containers in L_1 for S1/S2/S3/S4. Beyond this point, there are no benefits in increasing L_1 containers without increasing L_2 containers. Nevertheless, increasing L_2 containers only improves the performance slightly in S1 and S2, as the benefits of distributing models of low computational costs cannot compensate for their communication workload. As a result, complex models can be straightforwardly distributed, simulating all of them simultaneously and with a significant speedup.

When comparing the distributed and parallel simulations, we observe that the SU of the best distributed one within each strategy is greater than the SU of Parx8 and lower than the SU of Parx16. However, if

Table 4. Computational analysis: numerical results.

Coordinator	S1		S2		S3		S4	
	time (s)	speedup	time (s)	speedup	time (s)	speedup	time (s)	speedup
Prf.	1406.49	0.91	6401.21	0.92	6209.55	0.91	8346.38	0.91
Seq.	1273.69	1.00	5879.81	1.00	5632.22	1.00	7603.96	1.00
D1x1	1451.47	0.88	6516.46	0.90	6250.44	0.90	8383.01	0.91
D2x1	953.38	1.34	4007.62	1.47	3784.46	1.49	5074.52	1.50
D3x1	787.52	1.62	3233.41	1.82	2911.95	1.93	3876.61	1.96
D4x1	713.15	1.79	2659.70	2.21	2520.25	2.23	3375.57	2.25
D5x1	642.62	1.98	2401.46	2.45	2155.08	2.61	2997.17	2.54
D6x1	621.28	2.05	2271.72	2.59	2055.88	2.74	2791.12	2.72
D7x1	603.33	2.11	2175.13	2.70	1944.79	2.90	2645.47	2.87
D8x1	593.08	2.15	2062.93	2.85	1889.53	2.98	2557.97	2.97
D9x1	586.99	2.17	2012.30	2.92	1836.91	3.07	2512.61	3.03
D10x1	561.93	**2.27**	1974.12	2.98	1786.98	3.15	2426.67	3.13
D15x1	563.92	2.26	1913.23	3.07	1740.03	3.24	2336.78	**3.25**
D20x1	564.99	2.25	1902.99	**3.09**	1680.77	**3.35**	2387.37	3.19
D25x1	575.37	2.21	1952.92	3.01	1729.27	3.26	2435.62	3.12
D50x1	598.46	2.13	2062.30	2.85	1844.93	3.05	2526.40	3.01
D75x75	614.60	2.07	2080.47	2.83	1913.82	2.94	2582.20	2.94
D100x1	621.85	2.05	2141.94	2.75	1939.52	2.90	2684.79	2.83
D100x5	509.85	2.50	1886.62	3.12	1823.49	3.09	2492.32	3.05
D100x10	486.24	2.62	1850.94	3.18	1860.35	3.03	2431.53	3.13
D100x15	471.03	**2.70**	1860.30	3.16	1852.52	3.04	2526.93	3.01
D100x20	476.60	2.67	1837.65	3.20	1825.77	3.08	2495.45	3.05
D100x25	475.34	2.68	1857.40	3.17	1788.84	3.15	2490.98	3.05
D100x50	490.88	2.59	1821.82	**3.23**	1841.74	3.06	2528.78	3.01
D100x75	502.07	2.54	1946.98	3.02	1859.59	3.03	2556.95	2.97
D100x100	532.36	2.39	1927.30	3.05	1889.01	2.98	2535.17	3.00
Parx8	639.39	1.99	2229.97	2.64	2010.73	2.80	2812.17	2.70
Parx16	426.43	2.99	1714.44	3.43	1621.26	3.47	2229.91	3.41

we include in the comparison the cost/month of each setup (calculated as 208.32, 338.36, and 416.63 $/month for the 8 parallel, 16 distributed, and 16 parallel CPUs), the distributed simulations offer a good trade-off between flexibility, performance, and cost.



(a) Strategy 1, ET=1902.4s

(b) Strategy 2, ET=1038.2s

(c) Strategy 3, ET=1329.8s

(d) Strategy 4, ET=1160.8s

Figure 14. Representative solutions of the optimised strategies.

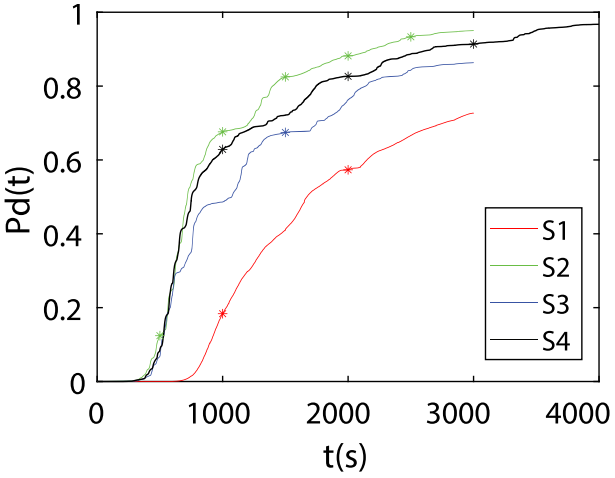


Figure 15. $P_d(t)$ of the representative solutions

5.4. Representative solutions

To illustrate the importance of optimising multiple strategies, we show a representative solution for each of them and its associated ET in Figure 14, and their corresponding $P_d(t)$ curves in Figure 15. In general terms, the solution obtained for S2 is the best, as it reaches the lowest ET and the highest P_d at t = 3000s. This happens because using two UAVs is often better than using only one, and because S2 T_{DH} lets the optimiser handle the correct number of decision variables within each section for this scenario. Besides, the longer $t^{end}_{mission}$ of S4 lets it reach a slightly higher P_d 1000 s later, worsening the ET of S2 solution.

As determining before-hand the best strategy to optimise is not straightforward (even for experienced UAV operators in target-search scenarios), our framework and its Cloud-deployment is useful to speed up the simulations, launch several simulations in a streamlined way, and efficiently perform systematical analysis of the parameters, behaviours, and structures of the models.

6. Conclusions

This paper presents a new DEVS-based framework, available in (Juan et al., 2022), that extends the capabilities of the evaluation framework by Bordón-Ruiz et al. (2021), by 1) adding an optimiser module to automate the generation of UAV and sensors search-strategies for target-search problems, and by 2) supporting its straightforward Cloud deployment due to its xDEVS distributed implementation (Mittal et al., 2017). This has a direct impact on the computational cost of the optimisation framework, where hundreds of UAV trajectories and sensor strategies are evaluated to find the best solution. The results show that the Cloud deployment allows speedups up to 3.35. This is very useful when complex models characterise the behaviour of the target-search elements, to increase the simulation realism and make the framework applicable in real-world scenarios. Moreover, cloud-computing also allows to simultaneously launch several optimisations (under different search strategies/parameterisations), providing operators with different solutions and a deeper insight into each mission.

Finally, taking advantage of the MBSE methodology and DEVS formalism, we plan to extend our framework capabilities by: 1) incorporating other probability models for the target (e.g., particle filters), 2) optimising other types of UAV trajectories (e.g., Dubin curves), 3) incorporating other optimisers in the framework to build a generalised Island model (Izzo et al., 2012), and 4) exploring new Cloud deployment policies.

Acknowledgments

This work is supported by the Spanish Ministry of Science and Innovation (MICINN) under AMPBAS (RTI2018-098962- B-C21) and by the Madrid Regional Government under IA-GES-BLOOM-CM (Y2020/TCS-6420). The computation has been supported by the Google Cloud Research Credits programme with award GCP19980904.

Disclosure statement

No potential conflict of interest was reported by the author(s).

References

Al-Zoubi, K., & Wainer, G. (2009). Performing distributed simulation with RESTful webservices. In *Proceedings of the 2009 winter simulation conference. (WSC)*, 1323–1334.

Andres-Toro, B., Giron-Sierra, J. M., Fernandez-Blanco, P., Lopez-Orozco, J. A., & Besada- Portas, E. (2004). Multiobjective optimization and multivariable control of the beer fermentation process with the use of evolutionary algorithms. *Journal of Zhejiang University- ScienceA*, *5*(4), 378–389. https://doi.org/10.1631/jzus.2004.0378

Bordón-Ruiz, J. B., Besada-Portas, E., Risco-Martín, J. L., & López-Orozco, J. A. (2021). DEVS-based evaluation of UAVs-based target-search strategies in realistically-modeled missions. *ACM SIGSIM Conference on Principles of Advanced. discrete simulation (PADS)*.

Bourgault, F., Furukawa, T., & Durrant-Whyte, H. F. (2004). Decentralized Bayesian negotiation for cooperative search. In *IEEE/RSJ international conference on intelligent robots and systems (IROS)* (Vol. 3, pp. 2681–2686). IEE.

Cárdenas, R., Arroba, P., Blanco, R., Malagón, P., Risco-Martín, J. L., & Moya, J. M. (2020). Mercury: A modeling, simulation, and optimization framework for data stream-oriented IoT applications. *Simulation Modelling Practice and Theory*, *101*, 102037. https://doi.org/10.1016/j.simpat.2019.102037

Carpin, S., Burch, D., Basilico, N., Chung, T. H., & Kölsch, M. (2013). Variable resolution search with quadrotors: Theory and practice. *Journal of Field Robotics*, *30* (5), 685–701. https://doi.org/10.1002/rob.21468

Deb, K., Pratap, A., Agrawal, S., & Meyarivan, T. (2002). A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary. Computing*, *6*, 182–197. doi:10.1109/4235.996017. https://ieeexplore.ieee.org/document/996017

Delle Fave, F., Xu, Z., Rogers, A., & Jennings, N. R. (2010). Decentralised coordination of unmanned aerial vehicles for target search using the max-sum algorithm. In *Proceedings of the workshop on agents in real time and. environment*, 35–44.

Fedorov, A. (2019). Path planning for uav search using growing area algorithm and clustering. In *Fourth Conference on Software Engineering and Information Management*.

Grocholsky, B., Keller, J., Kumar, V., & Pappas, G. (2006). Cooperative air and ground surveillance. *IEEE Robotics & Automation Magazine / IEEE Robotics & Automation Society*, *13*(3), 16–25. https://doi.org/10.1109/MRA.2006.1678135

Hall, S. (1997). A DEVS based simulation architecture for analysis of multi-vehicle interactions. In *Proceedings of SPIE - the International Society for Optical Engineering*, 287–294.

Happe, J., & Berger, J. (2010). CoUAV: A multi-UAV cooperative search path planning simulation environment. In *Summer computer simulation. conference*, 86–93.

Holman, K., Kuzub, J., & Wainer, G. (2010). UAV search strategies using Cell-DEVS. In *Spring simulation multiconference*.

Hu, J., Xie, L., Xu, J., & Xu, Z. (2014). Multi-agent cooperative target search. *Sensors*, *14*(6), 9408–9428. https://doi.org/10.3390/s140609408

Ivić, S., Crnković, B., Arbabi, H., Loire, S., Clary, P., & Mezić, I. (2020). Search strategy in a complex and dynamic environment: The MH370 case. *SciRep*, *10*.

Izzo, D., Rucinski, M., & Biscani, F. (2012). *The generalized Island model*.

Juan, B., Bordón-Ruiz, José, A. López-Orozco & Eva Besada Portas. (2022). DEVS/SAR: DEVS Search And Rescue M \&S\&O framework. https://github.com/iscar-ucm/devs-sar/releases/tag/2022-jos

Kiam, J. J., Besada-Portas, E., & Schulte, A. (2021). Hierarchical mission planning with a GA-optimizer for unmanned high altitude pseudo-satellites. *Electronics*, *21* (5), 36.

Kratzke, T., Stone, L., & Frost, J. R. (2010). Search and rescue optimal planning system. In *13th international conference on information fusion*.

Lanillos, P., Gan, S., Besada-Portas, E., Pajares, G., & Sukkarieh, S. (2014). Multi-UAV target search using decentralized gradient-based negotiation with expected observation. *Information Science*, *282*, 92–110. https://doi.org/10.1016/j.ins.2014.05.054

Lanillos, P., Yañez-Zuluaga, J., Ruz, J., & Besada-Portas, E. (2013). A Bayesian approach for constrained multi-agent minimum time search in uncertain dynamic domains. In *The genetic and evolutionary computation. conference*, 391–398.

Li, L., Zhang, X., Yue, W., & Liu, Z. (2021). Cooperative search for dynamic targets by multiple UAVs with communication data losses. *ISA. Transactions*

Linchant, J., Lisein, J., Semki, J., Lejeune, P., & Vermeulen, C. (2015). Are unmanned aircraft systems (UAS) the future of wildlife monitoring? a review of accomplishments and challenges. *Mammal Review*, *45* (4), 4. https://doi.org/10.1111/mam.12046

Mittal, S., Risco-Martín, J. L., Masuda, T., & Mittal, S. K. (2017). DEVSML 3.0 stack: Rapid deployment of DEVS farm in distributed cloud environment using microservices and containers. In *Spring simulation multiconference (SpringSim 2017). Diseases of the Esophagus: Official Journal of the International Society for Diseases of the Esophagus*, *30*(6), 1–19. https://doi.org/10.1093/dote/dox048

Mittal, S., Risco-Martín, J. L., & Zeigler, B. P. (2009). DEVS/SOA: A cross-platform framework for net-centric modeling and simulation in devs unified process. *SIMULATION*, *85*(7), 419–450. https://doi.org/10.1177/0037549709340968

Moreno, A., de la Torre, L., Risco-Martin, J. L., Besada-Portas, E., & Aranda, J. (2011). DEVS-based validation of UAV path planning in hostile environments. In *The International Defense and Homeland Security Simulation Workshop* (p. 135–140).

Moreno, A., de la Torre, L., Risco-Martin, J. L., Besada-Portas, E., Aranda, J., & Ayala, J. L. (2011). DEVS-based parallel framework for multi-objective evolutionary algortithms. In *The Fourth International Workshop on Parallel Architectures and Bioinspired Algorithms*.

Nigam, N. (2014). The multiple unmanned air vehicle persistent surveillance problem: A review. *Machines*, *2*(1), 13–72. https://doi.org/10.3390/machines2010013

Ntaimo, L., Hu, X., & Sun, Y. (2008). DEVS-FIRE: Towards an integrated simulation environment for surface wildfire spread and containment. *Simulation*, *84*(4), 137–155. https://doi.org/10.1177/0037549708094047

opensource.com. (2021). *What is docker?* https://opensource.com/resources/what-docker. (Accessed April. 30, 2021)

Pecker-Marcosig, E., Zudaire, S., Garrett, M., Uchitel, S., & Castro, R. (2020). Unified DEVS- based platform for modelling and simulation of hybrid control systems. *Winter Simulation Conference*, 1051–1162.

Pérez, E. (2017). Integrating mathematical optimization in DEVS for nuclear medicine patient and resource scheduling. *Winter Simulation Conference*, 398–407.

Perez-Carabaza, S., Bermudez-Ortega, J., Besada-Portas, E., Lopez-Orozco, J. A., & de la Cruz, J. M. (2017). A multi-UAV minimum time search planner based on ACOR. In *Proceedings of the genetic and evolutionary computation. conference*, 35–42.

Perez-Carabaza, S., Besada-Portas, E., Lopez-Orozco, J. A., & de la Cruz, J. M. (2016). A real world multi-UAV evolutionary planner for minimum time target detection. In *Proceedings of the genetic and evolutionary computation conference 2016*. 981–988.

Pérez-Carabaza, S., Besada-Portas, E., López-Orozco, J., & Pajares, G. (2019). Minimum time search in real-world scenarios using multiple UAVs with onboard orientable cameras. *Journal of Sensors*, *2019*, 22. https://doi.org/10.1155/2019/7673859

Pérez-Carabaza, S., Scherer, J., Rinner, B., López-Orozco, J., & Besada-Portas, E. (2019). UAV trajectory optimization for minimum time search with communication constraints and collision avoidance. *Engineering Applications of Artificial Intelligence*, *85*, 357–371. https://doi.org/10.1016/j.engappai.2019.06.002

Riehl, J. R., Collins, G. E., Hespanha, J. P., Xie, Q., Li, P., Chen, J., & Yao, S. (2011). Cooperative search by UAV teams: A model predictive approach using dynamic graphs. *IEEE Transactions on Aerospace and Electronic*

Systems. *Chemical Communications (Cambridge, England)*, *47*(4), 2637–2656. https://doi.org/10.1039/c0cc05188h

Risco-Martín, J. L., Mittal, S., Fabero, J. C., Zapater, M., & Hermida, R. (2017). Reconsidering the performance of DEVS modeling and simulation environments using the DEVStone benchmark. *Simulation*, *93*(6), 459–476. https://doi.org/10.1177/0037549717690447

Risco, J. L., Mittal, S., Atienza, D., Hidalgo, J. I., & Lanchares, J. (2008). Optimization of dynamic data types in embedded systems using DEVS/SOA-based modeling and simulation. In *Proceedings of the 3rd international icst conference on scalable information*. *Systems 2009*, 1–11.

Saadaoui, H., El Bouanani, F., Dobson, F. S., Saadaoui, H., Viblanc, V. A., & Bize, P. (2018). Information sharing based on local PSO for UAVs cooperative search of unmoved targets. In *International conference on advanced communication technologies and networking*. *Ecology and Evolution*, *8*(2), 1084–1095. https://doi.org/10.1002/ece3.3677

Shakhatreh, H., SawalMeh, A. H., Al-Fuqaha, A., Dou, Z., Almaita, E., Khalil, I., . . . Guizan, M. (2019). Unmanned aerial vehicles (UAVs): A survey on civil applications and key research challenges. *IEEE Access* .

Skorobogatov, G., Barrado, C., & Salami, E. (2020). Multiple UAV systems: A survey. *Unmanned Systems*, *8*(2), 149–169. https://doi.org/10.1142/S2301385020500090

Tisdale, J., Kim, Z., & Hedrick, J. K. (2009). Autonomous UAV path planning and estimation. *IEEE Robotics Automation Magazine*, *16*(2), 35–42. https://doi.org/10.1109/MRA.2009.932529

Trummel, K. E., & Weisinger, J. R. (1986). The complexity of the optimal searcher path problem. *Operational Research*, *34*(2), 324–327. https://doi.org/10.1287/opre.34.2.324

Van Tendeloo, Y., & Vangheluwe, H. (2015). PythonPDEVS: A distributed parallel DEVS simulator. *Springsim (TMS-DEVS)*, 844–851.

Wang, Y., Zhang, M. X., & Zheng, Y. J. (2017). A hyper-heuristic method for UAV search planning. In *International conference on swarm intelligence*.

Wong, E., Bourgault, F., & Furukawa, T. (2005). Multi-vehicle Bayesian search for multiple lost targets. In *IEEE international conference on robotics and automation* (pp. 3169–3174).

Wymore, A. W. (2018). *Model-based systems engineering* (Vol. 3). CRC press.

Yang, Y., Minai, A., & Polycarpou, M. (2002). Decentralized cooperative search in UAV's using opportunistic learning. In *AIAA guidance, navigation, and control conference and exhibit*.

Yao, P., Want, H., & Ji, H. (2017). Gaussian mixture model and receding horizon control for multiple UAV search in complex environment. *Nonlinear Dynamics*, *88*(2), 903–919. https://doi.org/10.1007/s11071-016-3284-1

Yao, P., Xie, Z., & Ren, P. (2019). Optimal UAV route planning for coverage search of stationary target in river. *IEEE Transactions on Control Systems Technology*, *27*(2), 822–829. https://doi.org/10.1109/TCST.2017.2781655

Yeong, S. P., King, L. M., & Dol, S. S. (2015). A review on marine search and rescue operations using unmanned aerial vehicles. *International Journal of Marine and Environmental Sciences*, *9*(2).

Yuan, C., Zhang, Y., Liu, Z., Zhang, Y., & Liu, Z. (2015). A survey on technologies for automatic forest fire monitoring, detection and fighting using UAVs and remote sensing techniques. *Canadian Journal of Forest Research*, *45*(7), 783–792. https://doi.org/10.1139/cjfr-2014-0347

Zeigler, B. P., & Kim, D. (2019). Multi-resolution modeling for adaptive UAV service systems. In *Proceedings of spring simulation*. *conference*, 1–12.

Zeigler, B. P., Moon, Y., Lopes, V. L., & Kim, J. (1996). DEVS approximation of infiltration using genetic algorithm optimization of a fuzzy system. *Mathematical and Computer Modelling*, *23*(11–12), 215–228. https://doi.org/10.1016/0895-7177(96)00074-X

Zeigler, B. P., Muzy, A., & Kofman, E. (2018). *Theory of modeling and simulation: Discrete event & iterative system computational foundations*. Academic press.