# Experience Report: Application of Falsification Methods on the UxAS System*

Cumhur Erkan Tuncali[1], Bardh Hoxha[2], Guohui Ding[3], Georgios Fainekos[1], and Sriram Sankaranarayanan[3]

**1.** Arizona State Univ., {etuncali,fainekos}@asu.edu **2.** Southern Illinois Univ., {bhoxha}@siu.edu, **3.** Univ. of Colorado, Boulder, {firstname.lastname}@colorado.edu

**Abstract.** In this report, we present our experiences in applying falsification methods over the Unmanned Systems Autonomy Services (UxAS) system. UxAS is a collection of software modules that enables complex mission planning for multiple vehicles. To test the system, we utilized the tool S-TaLiRo to generate mission scenarios for both UxAS and the underlying vehicle simulators, with the goal of finding behaviors which do not meet system specifications.

## 1 Introduction

Testing and verification of Cyber-Physical Systems (CPS) with respect to their functional or safety requirements is a critical and difficult problem. The difficulty for testing mainly arises from the fact that the generally large input and state space of most complex systems makes it challenging to identify the values of the inputs and the initial system states which will lead unexpected behaviors.

Among different testing methodologies, requirements-based boundary-value testing is an approach where the system is tested for the boundary values extracted from the requirements. Although it is a very widely used approach in the industry for testing safety-critical systems, it does not cover the input space of the CPS well. Hence, it may fail to find the failure cases which are not at the boundaries of the requirements. On the other hand, fuzzing [10], where the tests are randomly sampled from the input space of the system, provides a better coverage of the input space. However, the input space is generally infinitely large, especially when it is on real-valued inputs. If an unexpected behavior for the system is caused by a small region in the input space, then, in general, there is a very small probability to hit that small region with randomly generated test cases. Optimization-based test generation/falsification approaches utilize global optimization methods to guide the tests towards a possibly small region in the input space that lead to an incorrect system behavior. Falsification can be defined as the task of discovering counterexamples, i.e., the system behaviors that do not satisfy the given safety or functional requirements.

In this work we use optimization-based falsification for identifying the conditions that cause an unexpected system behavior. In particular, we used S-TaLiRo which is

* This research was supported by the Summer of Innovation 2017 program organized by AFRL and Wright Brothers Institute in Dayton, OH.

a robustness-guided automatic test case generation tool [4, 9]. S-TaLiRo simulates the system with generated input signals and computes a robustness value for the simulated system trajectory. The robustness value is basically a measure of how close is a system trajectory to a set of unsafe behaviors [5]. While a positive robustness value indicates that the trajectory satisfies the system requirements, a negative robustness value means that the trajectory does not satisfy (falsify) at least one system requirement. After computing the robustness value, S-TaLiRo utilizes stochastic optimization techniques [8, 1] to update test cases in order to minimize the robustness value. The search continues until it finds a negative robustness value, i.e., a system trajectory falsifying (failing to satisfy) the requirements, or until it exceeds the maximum number of simulations. The stochastic nature of S-TaLiRo helps it to obtain a better coverage of the input space of the system compared to the boundary-value testing, while the robustness-guided search approach helps it to smartly guide the tests towards risky areas. This allows exploring the failure cases with a smaller number of simulations compared to random testing, or given a finite time, finding more failure cases than the random testing.

## 2 Problem Statement

UxAS is a publicly available task automation software for Unmanned Aerial Vehicles (UAVs), designed as a set of modular services by the US Air Force Research Laboratory (AFRL) [3]. UxAS computes optimal or close-to-optimal execution plans for a given set of tasks for multiple UAVs, and allows cooperative decision making between the UAVs. The search and surveillance tasks UxAS can handle include *point inspection*, *line (path) search*, *area search*, *spiral search* and *sector search* as described by Kingston et al [6]. The UxAS distribution contains some example scenarios that can be used as a base for these tasks. However, scenarios involve numerous parameters that are specific to an individual mission. Additionally, the missions are carried out under *operating region* constraints that describe *Keep In* regions that a particular aircraft must always remain inside and *Keep Out* regions that an aircraft must keep out of. Finally, the dynamics of the flight are subjected to wind and GPS disturbances.

*Formal System Requirements:* In order to mathematically evaluate whether a trajectory of the system satisfies its requirements, we need formal, mathematical representations of the system requirements. We utilize Metric Temporal Logic (MTL) specifications to formally express the system requirements [7]. MTL extends common temporal operators such as *Finally*, *Globally* and *Until* with time intervals that restrict how these

**Table 1.** Examples of parameters describing various mission types in UxAS.

| Mission Type | Parameters |
|---|---|
| Point Search | GPS Coordinate, view angles distance bounds |
| Line Search | Line coordinates, view angles max distance |
| Area Search | Region, desired resolution, angles |

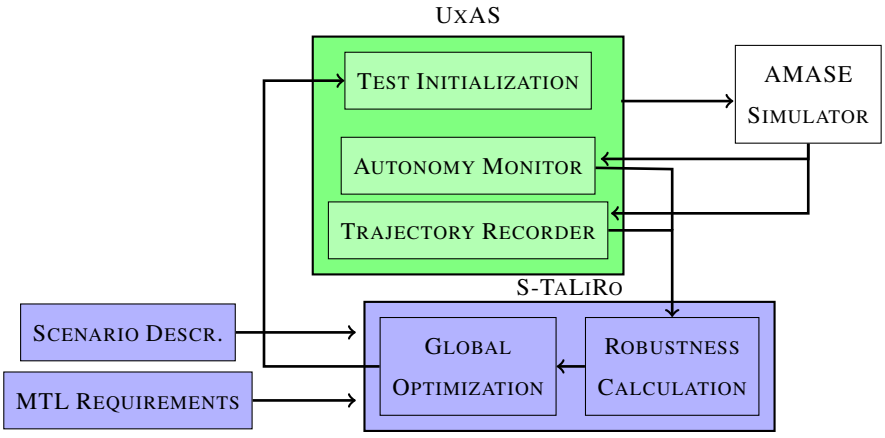


**Fig. 1.** An overview of the various components involved in the test generation setup for UxAS

operators are applied to a timed trace. We refer the reader to the original paper by Koymans [7] or our earlier work [1] for a detailed exposition of MTL. Using MTL, we can eliminate ambiguity in the requirements, and we can mathematically reason about the system behavior with respect to its requirements.

Thus, the overall problem is as follows:

INPUTS: Mission parameter ranges, operating region parameter ranges, wind and GPS disturbance parameters, additional MTL requirements.

OUTPUT: Concrete parameter values and operating regions, wind and GPS disturbance patters so that the resulting plan executions violate mission requirements, operating region requirements or additional MTL constraints.

## 3 Test Generation for UxAS

A basic overview of our test generation environment architecture is illustrated in Fig. 1. In this section we will briefly explain the main blocks of this environment.

*AMASE Simulator* OpenAMASE is an openly available aircraft dynamics simulation toolset developed by the US Air Force Research Laboratory [2]. It communicates with UxAS through a middleware interface and implements simulators for UAV platforms and disturbances such as wind. We have additionally incorported GPS disturbances into this framework. OpenAMASE is initialized by providing descriptions of the various air vehicles and their initial positions. Next, the UxAS system provides a series of waypoints to the aircrafts to follow. AMASE roughly simulates the behavior of the Piccolo autopilot over these waypoints and periodically publishes aircraft positions and headings back to UxAS as the simulation progresses.

*Scenario Description* Scenarios are described externally through an XML file that is read and transmitted to the UxAS system through a middleware layer. The message to the UxAS system contains mission information that includes aircraft descriptions and configurations, initial aircraft states, target tasks, Keep In/Out zone parameters and weather conditions which can change during the simulation. In our approach, we mainly generate new test cases by modifying the parameters defined in these messages. We add additional fields to the XML structure to define the ranges of the parameters.

S-TaLiRo*/UxAS Interface* The testing process is started by a Matlab script. This startup script reads the scenario description XML files and extracts the ranges for the variable parameters. It further extracts the information on the path search tasks and the operating zones. If there is a path search task to be randomized, a random path for the task is generated. Similarly for any keep out zone, it randomly places the Keep Out zone around the path such that the keep out zone does not intersect with the path. The MTL requirements for the system are also specified in the startup script.

After basic configuration is read, S-TaLiRo is called with the parameter ranges, the ranges for the coordinates of the Keep Out Zones and the system requirements. Fig. 1 gives an overview of our test generation approach. After it starts, S-TaLiRo randomly samples from the parameter ranges, and communicates with the test services located in the UxAS over TCP/IP sockets to send the sampled parameter values and to receive the system trajectory at the end of the simulation. The received trajectories are used for computing the robustness value for the current execution. The sampled values are updated, and the simulation is executed again until a negative robustness value is obtained or until the user defined maximum number of simulation executions is reached. In this case study, we utilize the Simulated Annealing optimization method to search for parameters that minimize the robustness value.

*Autonomy monitors* The autonomy monitoring service is implemented inside UxAS to monitor the positions of vehicles over time and decide if a task has completed or failed. Furthermore, it computes the robustness value of the trajectory with respect to the task requirements. First, we define specific monitors for each type of task and operating region constraints in our overall mission specification. The monitors receive periodic timestamped messages containing the positions and headings of the various airplanes. It then updates the current completion status for each task and operating region constraints. It then publishes success or failure messages along with robustness values to S-Taliro.

We now briefly describe the operation of S-Taliro to generate test cases. This includes randomized generation of paths for various search tasks and the random generation of operation zone constraints.

*Random Path Generation* A random path is specified by its starting and ending points: $p_{start}, p_{end}$, minimum and maximum distances between various segments of the path $d_{min}, d_{max}$, maximum angular difference between segments $\theta_{max}$, and the standard deviation of angle difference, $\theta_{\sigma}$. We generate a list of coordinates $p_0 : p_{start}, \ldots, p_N : p_{end}$ with the line joining the coordinates specifying the overall path.

Initially, the partial path just consists of $p_{start}$. We then sample a point at a sample distance $d_{sample}$ and angle $\theta_{sample}$. $d_{sample}$ is chosen randomly from the given range, and $\theta_{sample}$ is chosen at random with the mean value centered around the line joining the last point to the end point $p_{end}$ and specified standard deviation. This process continues until the last point in the path so far is close enough to the end point. At this stage, the point $p_{end}$ is added to the list and the process terminates. The green paths in Fig. 2 are generated by this algorithm for a path search task.

*Scenario Description* Scenarios are described externally through an XML file that is read and transmitted to the UxAS system through a middleware layer. The message to the UxAS system contains mission information that includes aircraft descriptions and configurations, initial aircraft states, target tasks, Keep In/Out zone parameters and weather conditions which can change during the simulation. In our approach, we mainly generate new test cases by modifying the parameters defined in these messages. We add additional fields to the xml structure to define the ranges of the parameters.

*Random Placement of Keep Out Zone* A keep out zone specifies a region that an aircraft cannot enter during the mission. To test the system, we randomly place Keep Out zones ensuring that the overall mission remains feasible in doing so: i.e, no keep out zone intersects a path or target point in the mission specification. Our approach starts by computing the interval hull of the generated path and sampling a point in the hull. We then place the keep out region $R$ at this point and test for an intersection with the path. If an intersection occurs, we then find a point of intersection and choose a direction along with we translate the region $R$ by the minimum possible distance to move the current intersection point outside the region $R$. We repeat this process until we are free of intersections. However, this process need not terminate in all cases. To aid termination, we place maximum limits on the number of iterations and restart afresh. Examples of randomly generated paths (in green) with a randomly placed keep out zones (in red) are shown in Fig. 2.

### 3.1 Case-Study

The scenario in our case study involves three UAVs with ID 400, 500 and 600, for which we denote the positions by $p_{V400}, p_{V500}, p_{V600}$. A line search task arrives, and UxAS generates an optimal plan for one of the UAVs to perform the task of obtaining surveillance video that covers the path to be searched. We also add a keep out zone $Z_1$ at random such that it does not intersect with the search path. Since the aircrafts are not supposed to fly over keep out zones, we require that "Whenever any of UAV 400, 500 or 600 enters the keep out zone $Z_1$, it should exit the zone in 10 seconds". The MTL representation for this requirement can be given as

$$\square(r_1 \implies \Diamond_{[0,10]} \neg r_1) \wedge \square(r_2 \implies \Diamond_{[0,10]} \neg r_2) \wedge \square(r_3 \implies \Diamond_{[0,10]} \neg r_3),$$

where $r_1$ is a predicate which evaluates to True ($\top$) when the UAV 400 enters the keep out zone $Z_1$, i.e., $r_1 := p_{V400} \in Z_1$. Similarly, $r_2 := p_{V500} \in Z_1$ and $r_3 := p_{V600} \in Z_1$.

We leave the shape of the path to be searched, the nominal speed values for each of the aircrafts, wind speed and wind directions that can change 6 times over the simulation

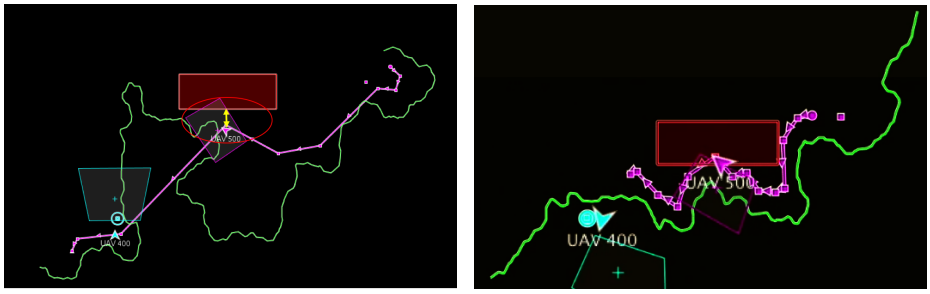


**Fig. 2.** Randomly generated search paths and Keep Out zones along with simulations: (left) satisfying requirements and (right) violation.

time as the variable parameters of the scenario. We use OpenAMASE to simulate the aircrafts. In our test generation approach, we randomly sample a path as the line search task and use S-TALIRO to search over the variable scenario parameters to discover the system behaviors that do not satisfy the MTL specification which is given above. The system under test can be considered as the UxAS tool together with the UAVs in the scenario.

The aircrafts start from their given initial positions. If an aircraft never enters zone $Z_1$ during the simulation, then the robustness value (for this example) is the minimum distance between the aircraft trajectory and the keep out zone boundaries. If an aircraft actually enters zone $Z_1$, then the robustness value for the violation reduces to how far inside $Z_1$ the UAV flies beyond the 10 second time limit. Moreover, in the latter case, the robustness value is negative indicating that the requirement has been violated.

In the execution example given in Fig. 2 (left) the green path is the one to be searched and the purple path connects waypoints generated by UxAS. In this case, the robustness value is the length represented by the yellow arrow which is the point where the aircraft comes closest to the keep out zone, as illustrated by the yellow arrow.

In our case study, S-TALIRO discovered cases where UxAS generates waypoints inside a keep out zone for a path search task and the vehicles fly into this zone (see Fig. 2 (right)) and stay inside the zone for more than 10 seconds which is against the system requirements.

## 4 Conclusion and Future Work

We have developed a framework which can be used to automatically generate test cases which can discover scenarios that lead to unexpected behaviors. Although the level of automation can be increased, we have automated most of the process by extracting data from existing scenario files. This ability would save human effort spent for generating test cases. Furthermore, because the automatic test generation framework does not have a developer's / tester's bias, it can discover unforeseen conditions leading to failure.

As a future work, we propose to use Simulink aircraft simulation models to apply coverage guided test generation techniques. Furthermore, we plan to utilize simplified

system dynamics for the aircrafts and the environment to compute functional gradient descent directions as described in our earlier work [11].

## References


1. H. Abbas, G. Fainekos, S. Sankaranarayanan, F. Ivančić, and A. Gupta. Probabilistic temporal logic falsification of cyber-physical systems. *ACM Transactions on Embedded Computing Systems (TECS)*, 12(2s):95, 2013.
2. Air Force Research Laboratory, Aerospace System Directorate, Power and Control Division. OpenAMASE, Aerospace Multi-agent Simulation Environment, Dec. 2017. Available at https://github.com/afrl-rq/OpenAMASE/.
3. Air Force Research Laboratory, Aerospace System Directorate, Power and Control Division. OpenUXAS, Project for multi-UAV cooperative decision making, Dec. 2017. Available at https://github.com/afrl-rq/OpenUxAS.
4. Y. Annpureddy, C. Liu, G. E. Fainekos, and S. Sankaranarayanan. S-taliro: A tool for temporal logic falsification for hybrid systems. In *TACAS*, volume 6605, pages 254–257. Springer, 2011.
5. G. E. Fainekos and G. J. Pappas. Robustness of temporal logic specifications for continuous-time signals. *Theoretical Computer Science*, 410(42):4262–4291, 2009.
6. D. Kingston, S. Rasmussen, and L. Humphrey. Automated UAV tasks for search and surveillance. In *Control Applications (CCA), 2016 IEEE Conference on*, pages 1–8. IEEE, 2016.
7. R. Koymans. Specifying real-time properties with metric temporal logic. *Real-Time Systems*, 2(4):255–299, 1990.
8. T. Nghiem, S. Sankaranarayanan, G. Fainekos, F. Ivancić, A. Gupta, and G. J. Pappas. Monte-carlo techniques for falsification of temporal properties of non-linear hybrid systems. In *HSCC '10*, pages 211–220, New York, NY, USA, 2010. ACM.
9. S-TaLiRo, Dec. 2017. Available at https://sites.google.com/a/asu.edu/s-taliro/s-taliro.
10. M. Sutton, A. Greene, and P. Amini. *Fuzzing: brute force vulnerability discovery*. Pearson Education, 2007.
11. C. E. Tuncali, S. Yaghoubi, T. P. Pavlic, and G. Fainekos. Functional gradient descent optimization for automatic test case generation for vehicle controllers. In *Automation Science and Engineering (CASE), 2017 IEEE International Conference on*. IEEE, 2017.