Simulation-based Testing of Unmanned Aerial Vehicles with AERIALIST

Sajad Khatiri
Università della Svizzera italiana &
Zurich University of Applied Sciences
Winterthur, Switzerland
sajad.mazraehkhatiri@{usi,zhaw}.ch

Sebastiano Panichella
Zurich University of Applied Sciences
Winterthur, Switzerland
sebastiano.panichella@zhaw.ch

Paolo Tonella
Università della Svizzera italiana
Lugano, Switzerland
paolo.tonella@usi.ch

ABSTRACT

Simulation-based testing is crucial for ensuring the safety and reliability of unmanned aerial vehicles (UAVs), especially as they become more autonomous and get increasingly used in commercial scenarios. The complexity and automated nature of UAVs requires sophisticated simulation environments for effectively testing their safety requirements. The primary challenges in setting up these environments pose significant barriers to the practical, widespread adoption of UAVs. We address this issue by introducing AERIALIST (unmanned **AERIAL** veh**I**cle te**ST** bench), a novel UAV test bench, built on top of PX4 firmware, that facilitates or automates all the necessary steps of definition, generation, execution, and analysis of system-level UAV test cases in simulation environments. Moreover, it also supports parallel and scalable execution and analysis of test cases on Kubernetes clusters. This makes AERIALIST a unique platform for research and development of test generation approaches for UAVs. To evaluate AERIALIST's support for UAV developers in defining, generating, and executing UAV test cases, we implemented a search-based approach for generating realistic simulation-based test cases using real-world UAV flight logs. We confirmed its effectiveness in improving the realism and representativeness of simulation-based UAV tests.

Code Repository: https://github.com/skhatiri/Aerialist
Demo Video: https://youtu.be/k_bqYpWItSg

CCS CONCEPTS

• **Software and its engineering → Software verification and validation**.

KEYWORDS

Unmanned Aerial Vehicles, Test Generation, Simulation

ACM Reference Format:
Sajad Khatiri, Sebastiano Panichella, and Paolo Tonella. 2024. Simulation-based Testing of Unmanned Aerial Vehicles with AERIALIST. In *2024 IEEE/ACM 46th International Conference on Software Engineering: Companion Proceedings (ICSE-Companion '24), April 14–20, 2024, Lisbon, Portugal.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3639478.3640031

1 INTRODUCTION

Unmanned Aerial Vehicles (UAVs), equipped with onboard cameras and sensors have demonstrated the possibility of autonomous flights in real environments, leading to great interest in various application scenarios: crop monitoring, surveillance, medical and food delivery [1]. Over the years, support for UAV developers has increased with open-access projects for software and hardware such as the autopilot support provided by Ardupilot [2] and PX4 [3].

The complexity and automated nature of UAVs require methods to systematically and effectively test their safe operation in dynamic environments. Researchers proposed the use of *digital twins*, i.e., virtual representations (simulations) of real-time, physical objects or processes, to simulate and test generic *cyber-physical systems* (CPS) in diversified scenarios [4] and support test automation [5].

Empirical studies have proven that UAV bugs can be potentially detected before field tests if proper simulation-based testing is in place [6–8]. This suggests the need for further research on setting up advanced simulation environments that test UAVs' behavior in real-world scenarios [9]. However, the *engineering complexity* of UAV test environments [10, 11], and the difficulty of setting up *realistic-enough* simulation environments that can capture the same bugs as physical tests [12] represent relevant obstacles. The design of *realistic test cases* [10, 11] that can leverage the system in *diversified scenarios* [9] and *reproduce real-world issues* [13] require the aforementioned advances in the UAV testing environments.

In this paper, we introduce **AERIALIST** (unmanned **AERIAL** veh**I**cle te**ST** bench), a novel test bench for UAV software, that automates all necessary UAV testing steps: setting up the test environment, building and running the firmware code, configuring the simulator with the simulated world properties, connecting the simulated UAV to the firmware and applying proper UAV configurations at startup, scheduling and executing runtime commands, monitoring the UAV at runtime for any issues, and extracting the flight log file after test completion.

With AERIALIST, we aim to provide software testing researchers with a popular UAV case study and an easy-to-use test automation and analysis platform to facilitate their onboarding in the UAV domain. This allows them to conveniently experiment with approaches that overcome the above-mentioned UAV simulation-based testing challenges. AERIALIST's adoption as the platform for the first UAV Testing Competition [14] organized by the Search-Based and Fuzz Testing (SBFT) workshop [15] is such an initiative designed to inspire and encourage the software testing community to direct their attention toward UAVs as a rapidly emerging and crucial domain[14].

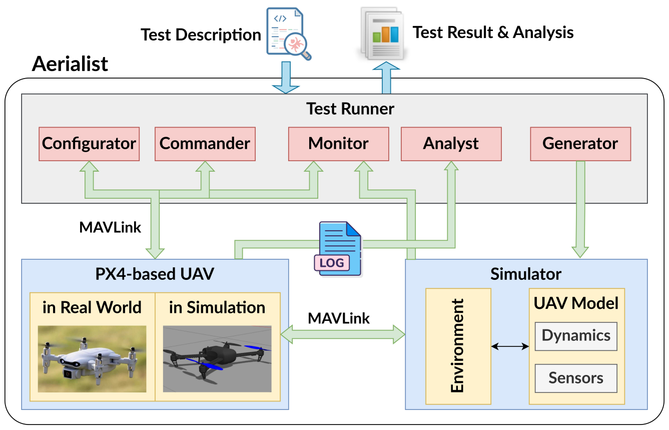

Figure 1: High-level architecture of Aerialist

We evaluated Aerialist's practical usefulness for UAV developers by experimenting with a search-based approach that analyses the logs from real UAV flights and automatically generates simulation-based test cases in the neighborhood of such real flights [9]. During our experiments, we observed that one of the challenging aspects of test case generation for UAVs is represented by the necessity of parallel and scalable running of the test cases (specifically when using search-based approaches which require executing test cases with several configurations). The Aerialist's support for the definition and execution of large-scale simulation experiments on Kubernetes clusters addresses these problems with increased reliability and scalability of UAV test outcomes.

2 THE AERIALIST TOOL

Aerialist is a modular and extensible test bench for UAV software and it aims to facilitate and automate all the necessary steps of definition, generation, execution, and analysis of system-level test cases for UAVs. Figure 1 demonstrates its architecture, with the implementation [16] currently supporting the PX4 platform [3](a widely used open-source UAV firmware), and the potential to be extended to support other UAV platforms.

The input is a *Test Description* file which defines the UAV and environment configurations and the test steps. The *Test Runner* subsystem, which abstracts any dependencies to the actual UAV, its software platform, and the simulation environment prepares the environment for running the test case as described in the test description. After setting up the simulation environment (if testing a simulated UAV), the *Test Runner* connects to the (simulated or physical) UAV and configures it according to the startup instructions. Then, it sends runtime commands, monitors the UAV's state during the flight, and extracts flight logs at the end of the test for future analysis. Each module is detailed in the following sections.

2.1 UAV Test Description

The de-facto testing standard of UAVs relies on *manually-written system-level tests* to test UAVs *in the field*. These tests are defined as software *configurations* (using parameters, config files, etc.), in a specific *environment* setup (e.g., obstacle placement, lighting conditions), and a set of runtime *commands*. The runtime commands received during the UAV flight (e.g., from a remote controller) make the UAV fly with a specific human observable *behavior* (e.g., trajectory, speed, distance to obstacles).

Hence, Aerialist models a UAV test case with the following set of *test properties* and uses a *yaml* structure (see Aerialist's repository [16] for details) to describe the test.

- *Drone Settings*: UAV configuration (i.e., all parameter values and configuration files) required to start the simulation.
- *Environment Settings*: Simulation configurations (e.g., used simulator, obstacles' position/shape, wind speed).
- *Commands*: Timestamped external commands from the ground control station (GCS) or the remote controller (RC) to the UAV during the flight (e.g., change flight mode, go in a specific direction, enter mission mode).
- *Expectation* (optional): time series of sensors' reading that the test flights are expected to follow closely.

2.2 UAV Software Platform

2.2.1 PX4. Aerialist aims to abstract low-level technical dependencies to the actual UAV software used to implement the UAV under test. PX4 [3] open-source flight control platform is often used to implement a UAV system. PX4 supports various flight modes, which provide different levels of autopilot assistance, ranging from automation of common tasks (e.g., takeoff and landing) or flying a preplanned path, to mechanisms that make it easier to hold a certain altitude level or position when needed.

PX4 supports Software In-the-Loop (SIL) simulation [17] to safely execute UAV flights in simulation environments and check novel control algorithms before actually flying the UAV, limiting the risk of damaging the vehicle. Since the communication to both real and simulated UAVs are technically identical, Aerialist can easily automate tests in both the real world and the simulation environment.

2.2.2 UAV Simulator. Simulators allow PX4 to control a modeled vehicle in a pre-defined *simulated world*. PX4 communicates with a physics simulator to receive sensor data and send actuator control commands back. The UAV pilot can also interact with the simulated vehicle (similar to a real vehicle) using a GCS, RC, or an offboard API (e.g. ROS). Aerialist currently supports test execution using two PX4-supported simulators [17] (Gazebo and jMAVSim) and can be extended to support others.

2.2.3 Flight Logs. PX4 logs any message communicated between RC or GCS and UAVs, or between its internal modules. This includes the sensor outputs, location, other estimations based on sensor readings, the commands sent to the UAV, and the errors/warnings from the internal modules. The stored logs are used by developers to investigate issues encountered during the flight. A sample flight log used in our experiments is available online[1].

2.3 Aerialist's Test Runner

2.3.1 Generator. The Generator module deals with setting up the simulated world before testing UAVs in SIL mode. It sets up and prepares the simulation environment as described in the test description, in a specific simulator (e.g., Gazebo, jMAVSim), along with the described static and dynamic objects and simulated UAV.

[1]https://logs.px4.io/plot_app?log=f986a896-c189-4bfa-a11a-1d80fa4b9633

2.3.2 Configurator. This module is responsible for setting up and initializing the UAV under test (simulated or real) before flying the UAV, according to the test description. This includes building the code, connecting to the UAV via MAVLink, setting the parameters, uploading any needed resources, etc.

2.3.3 Commander. This module is responsible for all the runtime communications to the UAV, including scheduling and sending the RC commands (e.g., manual sticks, flight mode changes, arm/disarm), communications from GCS, or the offboard commands coming from a companion computer.

2.3.4 Monitor. The monitor is responsible for runtime analysis of UAV state during the flight. Using MAVLink, we are able to subscribe to any messages communicated between PX4 modules, including sensor values. This allows monitoring of any potential runtime checks described in the test description.

2.3.5 Analyst. The module is responsible for the post-flight analysis, mostly based on the extracted flight log. It parses the log files and extracts any relevant data to analyze test results based on the given expectations in the test description.

3 USING AERIALIST

3.1 Command Line Interface

Aerialist can be used as a Python command-line utility, which needs proper setup and configurations of the PX4 platform for execution. To simplify this process, we have included a Dockerfile that sets up all the requirements:

```
docker build . -t aerialist
docker run -it aerialist bash
```

This will open a bash terminal to a Docker container that directly supports the execution of UAV test cases. The corresponding Docker image (available at Dockerhub[2]) includes all the necessary tools and configurations including Gazebo, PX4, and the Aerialist CLI. After each test execution, Aerialist gathers the flight logs from the simulation containers and stores them on the host machine.

Test Description. Aerialist uses a model called *UAV test description* for describing the tests. A sample *yaml* file used to describe the UAV test, and how to execute and evaluate it, is given below.

```
drone:
  port: sitl # conected drone type {sitl, ros, cf}
  params_file: path/to/params.csv # params file address
  mission_file: path/to/mission.plan # mission file addr.

simulation:
  simulator: gazebo # simulation engine {gazebo,jmavsim,
     ros}
  speed: 1 # simulator speed relative to real-time
  headless: true # execute without simulator GUI
  obstacles: [l,w,h, x,y,z, r] # box size, position, and
     rotation
  home_position: [lat,lon,alt] # drone's initial position

test:
  commands_file: path/to/commands.csv # runtime commands
     file address

assertion:
  log_file: path/to/log.ulg # reference log file address
  variable: trajectory # flight behavior to compare

agent:
  engine: docker # test execution environment {k8s,
     docker, local}
  count: 1 # no. of parallel runs (only for k8s)
```

Using such descriptions, one can easily run a UAV test case with the following command:

```
python3 aerialist exec --test path/to/test.yaml
```

Alternatively, users can execute the same test case by providing proper CLI arguments instead of the *yaml* description:

```
python3 aerialist exec --drone sitl --params path/to/
     params.csv --mission path/to/mission.plan --
     simulator gazebo --commands path/to/commands.csv --
     log path/to/log.ulg  --engine docker
```

More sample test description files, as well as corresponding command line options, can be found in the repository [16].

3.2 Python Package

Since we target to have Aerialist as a platform and building block for future research on testing UAVs, we put effort into the extensibility and usability of the tool as a software library that can be integrated into other applications, such as test generators. Specifically, we fully documented the tool and provided a public and self-contained Python package[3] and Docker image[2], as well as sample code snippets for its usage as a library for UAV test definition and execution[4] and complete test generation approaches built on top of its functionalities[5].

3.3 Usage Scenarios

Aerialist supports automating both autonomous (mission) and manual (remote-controlled) flights. It also supports executing the tests locally using local PX4 dependencies or inside pre-configured docker containers, as well as deploying the test execution workload in a Kubernetes cluster. Thus, Aerialist can be used in various settings as detailed below. It can be used as a local **Test Bench** during the development phase of UAV systems. It can also be integrated into their DevOps pipeline as a **CI Test Runner**. Most importantly, Aerialist can be used by UAV testing researchers as an **Experiment Platform** to evaluate their testing strategies.

3.3.1 UAV Test Bench. Aerialist can be used as a command line tool for locally executing simulation-based test cases for drones, while the graphical interface of the simulators can be used to visually follow the UAV behavior at runtime. It can also be used to replay a pre-logged flight, which could be quite handy when debugging certain failures, or when using a single case study for improving control algorithms.

3.3.2 DevOps Test Runner. Currently, the testing stage of the Continuous Integration (CI) pipeline of many CPS and UAV systems lacks an effective system-level testing solution. Although simulations have been used to facilitate testing, there are still technological challenges in automating simulation-based testing in standard CI

[2]https://hub.docker.com/repository/docker/skhatiri/aerialist

[3]https://pypi.org/project/aerialist/

[4]https://github.com/skhatiri/UAV-Testing-Competition/tree/master/snippets

[5]https://github.com/skhatiri/Surrealist

platforms. Aerialist is Dockerized and easy to integrate with modern CI platforms, provides a simple model to describe the test cases, and a proper CLI to automatically execute and evaluate the test cases. Hence, it can be used by UAV practitioners to fill this gap.

3.3.3 UAV Testing Research Platform. In recent years, software testing researchers have shown more interest in CPSs as a new domain. However, during our previous study [9] on test case generation for UAVs, we realized that the community lacks a proper tool that supports large-scale experiments. Hence, we extended Aerialist into an experimental platform suitable for research on different aspects of testing UAV systems (e.g., test generation).

Scalability: To ensure easy and fast execution and analysis of various UAV tests, we enable Aerialist users to run multiple tests in parallel in a Kubernetes cluster rather than on local machines. The flight simulations are transformed into Kubernetes Jobs and executed inside isolated Docker containers. After the test executions, the flight logs are uploaded to a cloud storage and processed centrally by the CLI.

Reliability: Due to the non-deterministic nature of the control mechanisms and the surrounding environment, the UAV behavior can be non-deterministic in both simulation and real-world settings. Specifically, given the exact same test scenario, the UAV can behave slightly differently on each test execution. In specific corner cases (see figure 2), the difference could be more severe and important, potentially failing the test in some runs and passing it in others. Moreover, the performance of the UAV in simulation heavily relies on the processing resources of the computer or Docker container running Aerialist. To ensure reliable test outcomes, Aerialist provides facilities to run multiple executions of the same test case in parallel (each with a fixed resource utilization) to eliminate outliers in results. For example, by running each test case *n* times, users can extract logs and compute the *average flight trajectory*.

4 EVALUATION

In our recent research [9], we evaluated Aerialist as an experimental platform to propose a novel search-based approach to automatically generate simulation-based test cases in the neighborhood of real-world UAV flights, improving the realism of SIL testing. Our approach initially analyzes the flight log, extracts available *test description* properties, and searches for optimal values of unknown properties to replicate the real-world UAV's behavior in the simulation environment. Then, it smoothly manipulates the replicated test description to identify related test cases that could potentially trigger unsafe UAV behavior in simulation. Figure 2 illustrates a challenging and flaky test case we generated for the autonomous flight in the presence of simple obstacles.

Our search-based approach starts with a simple and easy placement of the right side obstacle, and then step-by-step moves it in different directions to potentially make the path planning harder for the UAV. During this search process, an average of 50 different obstacle placements were evaluated using Aerialist, i.e., the flight was simulated and analyzed. Each of these test cases was executed 10 times in parallel to eliminate the outlier effect, account for the non-deterministic behaviors, and identify flaky tests. To ensure a statistically significant result, the whole process was also repeated 10 times with each repetition taking about 5 hours. In total, with

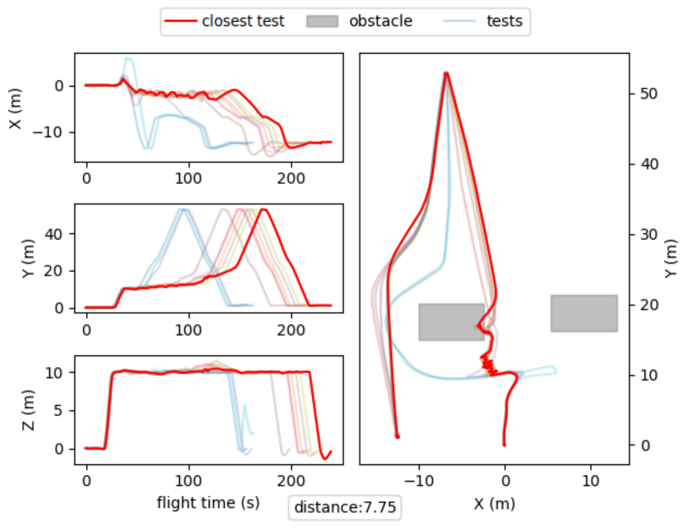

Figure 2: A flaky test identified using Aerialist (10 runs)

our limited Kubernetes resources (60 VCPUs and 80 GBs of RAM), it required us about 2 full days to conduct all 5,000 simulations. This would have taken at least 10 times more (20 days) on a single machine, while there is still the potential to speed up the whole process up to 10 times (5 hours) by running all the 10 repetitions in parallel given enough Kubernetes processing resources.

5 CONCLUSION

The development of Aerialist offers a promising solution to the challenges of creating advanced simulation environments for testing the safety requirements of unmanned aerial vehicles. With its automated functionalities and support for reliable and scalable test case executions, Aerialist provides a comprehensive experiment platform to support future research on UAV testing tools in both simulated and real UAV scenarios. Our evaluations show that the use of Aerialist can improve the feasibility and ease of implementation of search-based test case generation approaches for UAVs, and can significantly improve the required time for running the experiments thanks to its capability to run multiple tests in parallel in a Kubernetes cluster.

CRediT AUTHOR STATEMENT

Sajad Khatiri: Conceptualization, Methodology, Software, Validation, Investigation, Writing – Original Draft. **Sebastiano Panichella**: Conceptualization, Methodology, Writing – Review & Editing, Supervision, Project Administration, Funding Acquisition. **Paolo Tonella**: Conceptualization, Methodology, Writing – Review & Editing, Supervision.

ACKNOWLEDGMENT

We thank the Horizon 2020 (EU Commission) support for the project COSMOS, Project No. 957254-COSMOS.

REFERENCES

[1] X. Zhang, Y. Liu, Y. Zhang, X. Guan, D. Delahaye, and L. Tang, “Safety assessment and risk estimation for unmanned aerial vehicles operating in national airspace system,” *Journal of Advanced Transportation*, 2018.

[2] Ardupilot.org, “Ardupilot – versatile, trusted, open,” 2007, accessed: 07.02.2022. [Online]. Available: https://ardupilot.org/

[3] L. Meier, D. Honegger, and M. Pollefeys, “Px4: A node-based multithreaded open source robotics framework for deeply embedded platforms,” in *international conference on robotics and automation.* IEEE, 2015, pp. 6235–6240.

[4] C. Birchler, S. Khatiri, P. Derakhshanfar, S. Panichella, and A. Panichella, “Single and multi-objective test cases prioritization for self-driving cars in virtual environments,” *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 2022.

[5] M. Alcon, H. Tabani, J. Abella, and F. J. Cazorla, “Enabling unit testing of already-integrated AI software systems: The case of apollo for autonomous driving,” in *Euromicro Conference on Digital System Design.* IEEE, 2021, pp. 426–433. [Online]. Available: https://doi.org/10.1109/DSD53832.2021.00071

[6] C. S. Timperley, A. Afzal, D. S. Katz, J. M. Hernandez, and C. Le Goues, “Crashing simulated planes is cheap: Can simulation detect robotics bugs early?” in *2018 IEEE 11th International Conference on Software Testing, Verification and Validation (ICST).* IEEE, 2018, pp. 331–342.

[7] A. D. Sorbo, F. Zampetti, C. A. Visaggio, M. D. Penta, and S. Panichella, “Automated identification and qualitative characterization of safety concerns reported in uav software platforms,” *Transactions on Software Engineering and Methodology*, 2022.

[8] F. Zampetti, R. Kapur, M. Di Penta, and S. Panichella, “An empirical characterization of software bugs in open-source cyber–physical systems,” *Journal of Systems and Software*, vol. 192, p. 111425, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0164121222001315

[9] S. Khatiri, S. Panichella, and P. Tonella, “Simulation-based test case generation for unmanned aerial vehicles in the neighborhood of real flights,” in *16th IEEE International Conference on Software Testing, Verification and Validation (ICST)*, 2023.

[10] A. Afzal, C. Le Goues, M. Hilton, and C. S. Timperley, “A study on challenges of testing robotic systems,” in *International Conference on Software Testing, Validation and Verification.* IEEE, 2020, pp. 96–107.

[11] A. Afzal, “Automated testing of robotic and cyberphysical systems,” Ph.D. dissertation, Carnegie Mellon University, 2021.

[12] D. Wang, S. Li, G. Xiao, Y. Liu, and Y. Sui, “An exploratory study of autopilot software bugs in unmanned aerial vehicles,” in *ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2021, pp. 20–31.

[13] A. Afzal, D. S. Katz, C. Le Goues, and C. S. Timperley, “Simulation for robotics test automation: Developer perspectives,” in *Conference on Software Testing, Verification and Validation.* IEEE, 2021, pp. 263–274.

[14] S. Khatiri, P. Saurabh, T. Zimmermann, C. Munasinghe, C. Birchler, and S. Panichella, “SBFT tool competition 2024 - cps-uav test case generation track,” in *IEEE/ACM International Workshop on Search-Based and Fuzz Testing, SBFT@ICSE 2024*, 2024.

[15] S. organizers, “"the 17th intl. workshop on search-based and fuzz testing",” 2023. [Online]. Available: https://sbft24.github.io

[16] S. Khatiri, S. Panichella, and P. Tonella, “"aerialist: Uav test bench",” 2023. [Online]. Available: https://github.com/skhatiri/Aerialist

[17] PX4, “Px4 simulation,” https://docs.px4.io/v1.12/en/simulation/, 2021.