European Research Council
Established by the European Commission

Self-assessment Oracles for Anticipatory Testing

TECHNICAL REPORT: TR-Precrime-2023-06

Sajad Khatiri[a,b], Sebastiano Panichella[b], Paolo Tonella[a]
[a]Università della Svizzera italiana, [b]Zurich University of Applied Sciences

Simulation-based Test Case Generation for Unmanned Aerial Vehicles in the Neighborhood of Real Flights

Project no.:	787703
Funding scheme:	ERC-2017-ADG
Start date of the project:	January 1, 2019
Duration:	60 months

Technical report num.:	TR-Precrime-2023-06
Date:	January, 2023
Organization:	Università della Svizzera italiana
Authors:	Sajad Khatiri[a,b], Sebastiano Panichella[b], Paolo Tonella[a] [a]Università della Svizzera italiana, [b]Zurich University of Applied Sciences
Dissemination level:	Public
Revision:	1.0

Università della Svizzera Italiana (USI)

Principal investigator:	Prof. Paolo Tonella
E-mail:	paolo.tonella@usi.ch
Address:	Via Buffi, 13 – 6900 Lugano – Switzerland
Tel:	+41 58 666 4848
Project website:	https://www.pre-crime.eu/

Abstract

Unmanned aerial vehicles (UAVs), also known as drones, are acquiring increasing autonomy. With their commercial adoption, the problem of testing their functional and non-functional, and in particular their safety requirements has become a critical concern. Simulation-based testing represents a fundamental practice, but the testing scenarios considered in software-in-the-loop testing may not be representative of the actual scenarios experienced in the field.

In this paper, we propose **SURREALIST** (te**S**ting **U**AVs in the neighbo**R**hood of **REAL** fl**I**ght**S**), a novel 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, thereby improving the realism and representativeness of the simulation-based tests. This is done in two steps: first, SURREALIST faithfully replicates the given UAV flight in the simulation environment, generating a simulation-based test that mirrors a pre-logged real-world behavior. Then, it smoothly manipulates the replicated flight conditions to discover slightly modified test cases that are challenging or trigger misbehaviors of the UAV under test in simulation. In our experiments, we were able to replicate a real flight accurately in the simulation environment and to expose unstable and potentially unsafe behavior in the neighborhood of a replicated flight, which even led to crashes.

Contents

1 Introduction

With the boost of *cyber-physical systems* (CPS) in both academia and industry over the past decade, we have witnessed impressive advancements in the technology available in healthcare, avionics, automotive, railway, and robotics sectors [67, 16]. Unmanned Aerial Vehicles (UAVs) [72] or drones equipped with onboard cameras and sensors have already demonstrated that autonomous flights are possible in real environments. This sparked great interest in a plethora of application scenarios, with crop monitoring [15], surveillance [8], medical and food delivery [19], and search and rescue in disaster areas [18] representing only some of the relevant applications of UAVs.

Support for UAV developers has increased over the years, with open-access projects for the software (i.e., firmware) and hardware (e.g., flight controller). Well-known examples are Ardupilot [6] and PX4 [41] (autopilot software) and Pixhawk [50] (open standards for UAV hardware). On the other hand, automated testing of UAVs (and in general, CPS) to ensure their proper behavior represents still an open research challenge [4, 37, 58, 20, 61]. Simulation-based testing is a promising direction to improve UAV testing practices [3, 63, 66]. Researchers proposed the use of *digital-twins*, i.e., virtual representations of real-time, physical objects or processes, to simulate and test CPS in diversified scenarios [30, 14, 49, 11, 44, 13], and to support test automation [5, 69]. However, it is challenging to capture the same bugs as physical tests in simulation [66, 4] and to generate representative simulated test cases that expose realistic bugs [3].

To better illustrate the problem statement, let us consider the following scenario: *Bob* is a UAV customer using a quad-copter based on PX4 [41] (a popular open-source UAV firmware enabling autonomous flight, path planning, and obstacle avoidance) for crop monitoring missions over various croplands. Since some of these lands are close to or include trees, buildings, roads, or other populated areas and facilities, he is particularly concerned about the safety and reliability of his quad-copter during the missions. He has already tested the UAV in a specific scenario over one of these lands: an autonomous flight from a starting point S to a destination point D, crossing a small building on the way. *Bob* observed that the UAV reached the destination safely while avoiding obstacles in the scene, but he is not yet convinced that it will be the case in other possible scenarios and over other lands. Specifically, he is interested to know if the UAV would still complete the mission safely, even if the scenario was a bit different, e.g.,different building sizes, planned paths, or weather conditions.

Since he does not have the budget to test the UAV in all such variations manually in the field, *Bob* contacts *Alice*, an experienced PX4 developer, to help in the (safety) assessment of his UAV in such diversified scenarios. As the first step, *Alice* asks *Bob* for the *Flight Logs* of his field tests, as the logs include valuable information about how the environment was perceived by the UAV during the flights, e.g., all sensor readings, received commands, and motor control signals.

Now, *Alice* has the challenge of manually analyzing the flight logs, interpreting the results of the test, and investigating ways to make a proper assessment of the drone in alternative, neighboring flight scenarios. As a practical and viable strategy, *Alice* decides to use simulators (e.g., Gazebo[36]) to replicate the real test flights in simulation, and to identify close-related scenarios that could potentially fail in the real world. However, the problem of replicating the logged real flight in simulation with high fidelity remains a big issue for *Alice*. In the context of our work, we assume that if the UAV behavior in simulation is in line with its behavior in the field (e.g., the trajectory during the flight), the simulated replication can be considered *realistic*. Here, the questions are: 1) how to enable *Alice* to faithfully replicate a real UAV flight in simulation, by analyzing the flight log coming from an unknown environment? and 2) how to enable *Alice* to test the UAV in a set of diversified possible scenarios in the neighborhood of the given field test?

Software engineering researchers proposed several automated solutions to generate test cases reproducing the crashes of software-only systems [7, 43, 60, 32, 22, 59]. However, to the best of our knowledge, no existing approach for CPS/UAV testing [27, 63, 61, 71] addresses the problem of test scenario replication and test case generation, where the execution state to be reproduced is not only the state of the program, but it involves also the state of the real world.

In this paper, we propose **SURREALIST** [1](te**S**ting **U**AVs in the neighbo**R**hood of **REAL** fl**I**ght**S**), a novel search-based approach that automatically generates simulation-based test cases in the neighborhood of previously logged real-world UAV flights, thereby improving the realism of software-in-the-loop testing. Using our approach, the work done by *Alice* in the previous scenario is drastically reduced by simply giving *Bob*'s flight log as an input to SURREALIST. Following the steps described in Section 3, our approach first analyses the flight log, extracts any available UAV and environment configurations, and searches for optimal values for the unknown configurations to replicate the real-world UAV behavior in the simulation environment. Then, it smoothly manipulates the replicated configurations (flight conditions) to discover related test scenarios that can potentially trigger unsafe behavior of the UAV in simulation, which can also guide *Bob* toward potential corner cases for field testing.

This paper provides the following contributions:

- A generic approach for automatically generating a simulation-based test case that replicates a real flight scenario, by searching for optimal simulation environment configurations using only the flight log.
- A generic approach that automatically modifies a (replicated) simulation-based test case to generate more challenging test scenarios.
- An empirical evaluation of a specific instance of the generic approaches, for optimal placement of obstacles in the simulation environment during an autonomous flight.
- A replication package [34] on Github including SURREALIST implementation and experiments data and results.

2 Background

This section provides an overview of the UAV Architecture and UAV Firmware and Software used in our research.

2.1 UAV Architecture

UAVs are characterized by the Perception, Planning, and Control [47] software components, and the hardware components that interact with the environment and the UAV software. The **Perception component** is responsible for the UAV's understanding and modeling of the surrounding environment based on sensor signals. The functionalities of this component include *state estimation* [26, 55] and *mapping* [45]. State estimation recreates the drone state in the environment and enables navigation and autonomous movement [55], while mapping strategies compute obstacle distances, to create a model of the surrounding area [45]. The **Planning component** aims at finding an optimal trajectory from starting point to the destination, e.g., by computing *polynomial trajectories* [42, 57] and then applying *trajectory optimization* [25]. The **Control component** determines the actuator control commands to be executed by the UAV to safely navigate the environment and enables the autopilot (onboard commands) and/or the ground-control station (commands from a remote station) modalities [64, 47, 17].

[1]Surrealism is an art movement aimed at resolving the previously contradictory conditions of dream and reality into an absolute reality, a super-reality, or surreality.

2.2 UAV Firmware and Software

2.2.1 PX4 Platform

PX4 [41] open-source flight control platform is often used to implement a UAV system. PX4 supports Software In-the-Loop (SIL) simulation to safely execute UAV flights in simulation environments, with the purpose of checking novel control algorithms before actually flying the UAV, limiting the risk of damaging the vehicle. It also supports Hardware In-the-Loop (HIL) simulation, by providing simulation inputs to the firmware deployed on a real flight controller board.

2.2.2 PX4 Simulation Environments

Simulators allow PX4 to control a modeled vehicle in a *simulated world*. Hence, PX4 communicates with a simulator (e.g., Gazebo [36]) to receive sensor data from the simulated world and send actuator control commands back. In this setting, the UAV pilot (user), similarly to a real vehicle, can interact with the simulated vehicle using a ground control station (GCS), a radio controller (RC) or an offboard API (e.g. ROS), both to send control commands and to receive telemetry data. PX4 supports several HIL and SIL simulators [51]. In the context of our work, we considered Gazebo [36] as PX4's reference 3D simulation environment since it is particularly suitable for testing its obstacle avoidance and computer vision functionalities.

2.2.3 Flight Logs

PX4 logs any message communicated between RC or GCS and UAVs, or between its internal modules [53]. 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. Logs are stored on the UAV file system after each flight, to help investigate issues encountered during a flight and their root causes [53]. A sample flight log [33] we used in our experiments is visualized in the footnote link[2].

2.2.4 Flight Modes

Flight modes define how the autopilot responds to RC input, and how it manages the vehicle movements during fully autonomous flights. Flight modes 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. Flight modes can be divided into *manual* and *autonomous* modes. Manual modes allow the user to control the vehicle movement via the RC sticks, while autonomous modes are fully controlled by the autopilot, with no pilot/RC input. During an autonomous flight, obstacle avoidance [54] can be enabled to let the UAV locate any obstacle on its path using its onboard cameras, and navigate around them safely to reach the destination.

3 Approach

The de-facto standard testing process of UAVs relies on *manually-written system-level tests* for testing UAVs *in the field*. These tests are defined as software *configurations* in a given *physical environment* and a set of runtime *commands* that make the UAV fly with a specific observable *behavior* (e.g., flight trajectory, speed, distance to obstacles). We model a UAV-simulated test case with the following *test properties*:

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

- *UAV Configuration: Autopilot parameters* [3] *set at startup, configuration files (e.g., mission plan) required.*
- *Environment Configuration:* Simulation settings such as simulation world (e.g., surface material, UAV's initial position), surrounding objects (e.g., obstacles size, position), weather condition (e.g., wind, lighting).
- *Runtime Commands:* Timestamped external commands sent from GCS or RC to the UAV during the flight (e.g., changing the flight mode, flying in a specific direction, starting autonomous flight).

Since the physical attributes of the *simulated* and *real* UAVs and the surrounding environments are often not identical, simply replaying the same set of commands sent to a physical UAV (as recorded in the logs) would not always result in the same observable behavior in simulators. For instance, sending a command for going forward with full power for 1 second, will likely not bring the real and simulated UAVs to have the same speed and acceleration, and to cover the same distance. This is typically due to the differences in the UAV's real vs simulated characteristics (e.g., weight, motors power, and sensors accuracy) and to unpredictable environmental factors (e.g., wind and other disturbances).

Given a field test log, SURREALIST aims to generate simulated test cases that replicate, as closely as possible, real-world observations (e.g., flight trajectory). This is done by finding the best combination of the above-mentioned test properties, so as to minimize some distance measure between the sensor readings of the field test and its simulated counterpart (e.g., the Euclidean distance between the two flight trajectories).

Starting from this replicated simulation test, SURREALIST generates variants in the close neighborhood of the test case, with the goal of creating potentially more challenging scenarios. This is achieved by updating the test properties, with the purpose of increasing the difficulty (or risk level) of the generated test cases. The test difficulty is measured according to a given *fitness function*, e.g., the minimum distance of the UAV to the obstacles during the flight. To achieve these goals, we propose a generic search-based approach that generates simulation-based test cases that minimize a given *distance measure* (or maximize a given fitness function) by iteratively manipulating the corresponding test properties.

In the following sections, we first describe the generic approach to generate test cases that optimize a given fitness function. Then, we instantiate it for replicating the *flight trajectory* of autonomous flights, and for generating challenging test cases for maintaining *safe distance* to the obstacles, by manipulating *obstacles* in the simulation environment. It is important to note that the generic approach can also be instantiated to replicate other UAV behaviors (e.g., speed, acceleration, outputs to motors), or generate challenging test cases w.r.t other requirements (e.g., UAV stability, mission duration, power consumption), and by manipulating other test properties (e.g., wind, planned waypoints, runtime commands) which are out of the scope of this paper.

3.1 Generic Approach

3.1.1 Context

The proposed generic approach can be used in two different contexts:

A) ***[Flight Replication]*** Given a real flight log, the goal is to generate a simulation-based test case that replicates the flight w.r.t. specific UAV behavioral properties. The behavioral properties to reproduce can be any logged variable, such as outputs to the actuators (motors thrust), raw inputs coming from sensors, or higher-level variables calculated from them (e.g., UAV position in the 3D space), with the replication accuracy measured by a *distance metric (or similarity measure)*. For

[3] https://docs.px4.io/main/en/advanced_config/parameter_reference.html

instance, by choosing to replicate the 3D space position variables we create a simulated flight with a similar trajectory as that recorded in the real-world log.

B) ***[Test Generation]*** Given a simulation-based test case, the goal is to generate variants of such test that are more *challenging* w.r.t specific *difficulty measures*. The difficulty of the test case is calculated based on the risk level of violating (safety) requirements, such as flying too close to obstacles.

We *formulate both problems as a search problem* focused on finding the optimal test properties that maximize a relevant *fitness function*.

3.1.2 Search Algorithm

Algorithm 1 details our approach. Overall, the search is an iterative process that finds the best mutations to apply to the current solution at any given step.

The process starts with an initial *seed solution* (test properties). In the context of flight replication, the seed is available directly from the raw data in the original flight log. It includes the logged drone configurations, RC command series, and potential obstacle information that can be extracted from distance sensors. In the case of test generation, the seed consists of an existing simulation-based test case, from which the algorithm generates more challenging variants.

The second input, *fitness_func*, is the function computing the fitness of the solutions. It gets the simulated flight logs as input, and computes a fitness value according to the given goal (see section 3.1.4). For flight replication, it consists of a distance metric between the original flight sensor values and the simulated ones, as described in detail in section 3.2. For test generation, it consists of a risk assessment measure for the given test case, as described in detail in section 3.3.

The third input is the *budget* assigned to the search process: the maximum number of test case evaluations performed before returning the final solution found during the search. We measure the search budget as the number of *allowed test case evaluations* since evaluating a candidate test is the most expensive operation performed by the proposed algorithm, as it consists of a full simulation of the UAV behavior. The final input, *min_rounds* corresponds to the *minimum ensured rounds of global search* (the while loop in lines 6-11).

The initial best solution is set at line 2 as the seed. At line 3, we initialize an empty dictionary, named *evaluation_hash* which will record all the evaluated solutions and their fitness values as the algorithm proceeds. *evaluation_hash* will be used by the function *EVALUATE* (described in section 3.1.3, pseudo-code not shown for space reasons) to implement *memoization*, i.e., to skip the simulations if the same test properties have been already evaluated in previous iterations and directly set the fitness value as obtained from the dictionary. The *EVALUATE* function is also in charge of updating the execution budget, which is decreased by one when a simulation is needed, while it is left unchanged if the fitness for the requested execution is available from *evaluation_hash* (the last parameter, which is NULL at line 4, is a local budget updated similarly to the global one).

After evaluating the initial seed solution, which consumes one simulation from the budget, the main optimization loop is entered at line 6. The loop terminates when the local search, invoked at line 10, is unable to find a better solution for all parameters that can be mutated in the test properties. We assume here that test properties come with *mutators*, i.e., parameterized operators that can be applied individually to each test property. For example, the property $obstacle.position.x$ can be modified by an additive mutator, which moves the current position of the obstacle in the simulation environment along the x axis by a parameter value called $MOVE\_X$.

The *for* loop at line 9 iterates over all mutators available for the given test properties and tries to optimize each of them individually by invoking a local optimization procedure at line 10. The local budget is obtained by uniformly distributing the available budget across all mutators in the remaining ensured rounds (line 8), with each local search not necessarily consuming entirely its local budget. Hence, when re-entering the while loop at line 6, the residual global budget might still be available for the next iteration.

Algorithm 1: GENERIC-TEST-PROPERTIES-SEARCH

```
Input: seed: original test properties
       fitness_func: fitness function to maximize
       budget: global search budget (max num of simulations)
       min_rounds: minimum ensured round of global search
Result: best: test properties that optimize the fitness
1  begin
2      best = seed
3      evaluation_hash = {}
4      EVALUATE(fitness_func, best, evaluation_hash, budget, NULL)
5      improved = true
6      while improved do
7          improved = false
8          local_budget = budget/(|seed.mutators|×min_rounds)
9          for mutator in seed.mutators do
10             improved = improved ∨ LOCAL-TEST-PROPERTIES-SEARCH(best, mutator,
                 evaluation_hash, local_budget, budget)
11         min_rounds = min_rounds - 1
12     return best
```

Algorithm 2 shows the details of the local search. This algorithm can be classified as an *adaptive greedy algorithm* that searches the parameter space of each mutator. We chose a greedy technique to reduce the number of simulations needed to reach the optimum (executing an entire simulation is computationally expensive). At the same time, to avoid the choice of a sub-optimal greedy optimization step, we adapt the *step* parameter as the algorithm progresses.

Each mutator comes with a default value and a default step. The default value is a value that leaves the test properties unchanged. For a multiplicative mutator, it is 1; for an additive mutator, 0. The default step is mutator specific. For example, the mutator that moves the obstacle along the x axis has a default value of 0, because it is additive, and has a default step of 4 meters. The default step is defined specifically for each mutator parameter, based on the expected parameter range. Its value is not critical, as the local search adjusts it in an adaptive way. At lines 2-3 the default step and parameter value for the given mutator are assigned to the variables *step*, *param*.

The optimization loop starts at line 7, with 2 termination conditions: the local budget expired, or no improvement in the best solution for more than *MAX_IT* iterations.

In lines 8-9, we create two mutated solutions by either increasing or decreasing the mutator parameter by the current optimization step. These candidate solutions are evaluated at lines 10-11 (function *EVALUATE* will skip simulation if the test properties can be found in *evaluation_hash*). Then, if either the first mutated solution or the second one improves the current best solution by a margin higher than ϵ, the new best solution is recorded. Otherwise, if we are in a plateau (line 29) the optimization loop stops, while if both mutations *mu1, and mu2* have decreased the fitness value by an amount greater than ϵ (*else* case at line 28 with a *false* condition at line 29), the optimization step is halved adaptively (line 31).

If the same step is applied multiple times in the positive (resp. negative) direction, at line 19 (resp. 27) the optimization step is doubled, to converge more quickly to the final solution. The local search terminates by assigning the new best solution to the input/output variable *best*, which is eventually returned by Algorithm 1. It returns *true* if a better solution was found during the local search; *false* otherwise.

Algorithm 2: LOCAL-TEST-PROPERTIES-SEARCH

```
InOut:  best: best solution found so far
Input : mutator: test property mutator
InOut:  evaluation_hash: memory of past evaluations
Input : local_budget: max simulations for current mutator
InOut:  budget: overall max simulations allowed
Result: improved: previous best solution was improved
begin
    step = mutator.default_step
    param = mutator.default_value
    positive_moves = negative_moves = 0
    iter_with_no_improvements = 0
    new_best = best
    while $local\_budget > 0 \wedge iter\_with\_no\_improvements < MAX\_IT$ do
        mu1 = MUTATE(best, mutator, param + step)
        mu2 = MUTATE(best, mutator, param - step)
        EVALUATE(mu1, evaluation_hash, budget, local_budget)
        EVALUATE(mu2, evaluation_hash, budget, local_budget)
        if $mu1.fitness > new\_best.fitness + \epsilon \wedge mu1.fitness > mu2.fitneess$ then
            new_best = mu1
            param = param + step
            positive_moves += 1
            negative_moves = 0
            iter_with_no_improvements = 0
            if $positive\_moves > MAX\_SEQ\_IT$ then
                step = step · 2
        else if $mu2.fitness > new\_best.fitness + \epsilon$ then
            new_best = mu2
            param = param - step
            negative_moves += 1
            positive_moves = 0
            iter_with_no_improvements = 0
            if $negative\_moves > MAX\_SEQ\_IT$ then
                step = step · 2
        else
            if $|new\_best.fitness - mu1.fitness| < \epsilon \wedge |new\_best.fitness - mu2.fitness| < \epsilon$ then
                break
            step = step / 2
            positive_moves = negative_moves = 0
            iter_with_no_improvements += 1
    improved = false
    if $new\_best \neq best$ then
        best = new_best
        improved = true
    return improved
```

3.1.3 Solution Evaluation

To evaluate a search solution, i.e., the candidate test properties, we generate and execute the corresponding simulated test case automatically. The test case automates all necessary 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 the UAV configurations from the test case properties at startup. Then, the test case commands are scheduled and sent to the UAV, the flight is monitored for any issues, and after test completion, the flight log file is extracted. Due to the nature of the control mechanisms and the surrounding environment, the UAV behavior (both in simulation and in the real world) can be non-deterministic. To eliminate the effects of outliers in our experiments, we run each test case n times, extract the logs, and use the average of the variables recorded in the logs for computing the fitness function.

3.1.4 Fitness Function

Our search algorithm has the overall goal of maximizing the given fitness function (*fitness_func*) provided as input to the algorithm with the following signature:
$fitness\_func(flight\_logs : List < Log >) \rightarrow float$
The input is a list of flight logs, obtained from multiple executions of the same simulation-based test case. The output is a numeric value measuring the overall fitness of the test case w.r.t. our goal. Since the fitness function is maximized by the local search algorithm (see lines 12, 20), when the goal is to minimize some metric value (e.g., the distance d between trajectories) we supply the negation of the metric value ($-d$) as the fitness function.

3.2 Flight Replication in Autonomous Mode

3.2.1 Context

Given the flight log and the mission configuration of an autonomous flight that includes a given number of N ($\geqslant 1$) obstacles, with unknown size and position, we generate a simulated test case that includes the optimized size and position of the obstacles, with similar UAV trajectory in simulation. We propose an instance of our generic flight replication algorithm for this problem. In PX4's autonomous mode, the mission is uploaded to the UAV in advance. After the mission *start* command, the UAV follows the mission waypoints in a completely autonomous way. If obstacle avoidance [54] is enabled, the UAV will use its distance sensors and camera to locate any obstacle on the way and will automatically find its way to the next waypoint beyond the obstacle.

3.2.2 Fitness Function

Since the relevant logged variables are time series, the fitness function must be able to compare two time series, measuring the distance between the sequence of replicated states from the *simulation log* (intermediate solutions of the search algorithm) and the sequence of expected states from the *original log* (real-world flight to be replicated). As a general-purpose fitness function that could potentially work on any metrics from the original log, we use Dynamic Time Warping (DTW) [9], a well-known distance measure for multi-dimensional time series of different lengths that has already been used for comparing UAV flight trajectories [2]. DTW is based on a dynamic programming algorithm that matches the elements appearing in two sequences s, t by finding the pairing that minimizes the overall cost. The minimum cost for pairing the element in position i from s with the element in position j from t is recursively determined as:

$$\begin{aligned} DTW[i,j] = d(s_i, t_j) + \min\{&DTW[i-1,j], \\ &DTW[i,j-1], DTW[i-1,j-1]\} \end{aligned} \tag{1}$$

where $d(s_i, t_j)$ is the distance between the metric values appearing at position i (resp. j) in the two logs (in the simplest case, just the Euclidean distance $\|s_i - t_j\|$), while the recursive choice of the minimum DTW value corresponds respectively to advancing the pairwise comparison by one position along the first list only, the second only, or both. Here, the compared sequences s and t are UAV trajectory points $\langle x, y, z \rangle$.

Since we run each test scenario multiple times to eliminate outliers' effect, we also need an *aggregation function* to group the logs obtained from multiple runs of the same simulated test scenario into one, coherent time series to be compared by DTW. Here, we adopt *DTW Barycenter Averaging* [48], an averaging method for time series data that determines and keeps the shape of the series, while computing the average.

3.2.3 Test Case Properties

The physical world is assumed to be a plain area with N obstacles of predefined shape (e.g., box). Since the flight is operated in autonomous mode, the only variable properties of the generated test cases are the obstacle properties, i.e., the size $(length, width, height)$, position (x, y, z), and rotation angle (r) of each obstacle.

3.2.4 Mutation Operators

To search for the optimal obstacle properties, we use the following mutation operators for any of the N obstacles individually:

Move: This mutation operator moves the obstacle from the previous location in the simulated world by Δ_x, Δ_y. We ignore the z dimension since we assume the boxes are always placed on the ground.

Resize: This mutation operator resizes the obstacle in place (keeping the center position) by $\Delta_l, \Delta_w, \Delta_h$.

Rotate: This mutation operator rotates the obstacle around its geometric center in the x,y plane by Δ_r degrees.

3.2.5 Seed

If no information on the placement and size of the N obstacles is available in the original log, we create an initial seed solution by randomly placing N obstacles in positions that intersect with the mission flying area.

3.3 Test Case Generation in Autonomous Mode

To find challenging and buggy UAV fly conditions, we designed another instance of the generic approach (Algorithms 1, 2), which generates challenging test scenarios, starting from the output of flight replication. The search seeds are the final solutions of the algorithms instantiated in Section 3.2.

3.3.1 Context

Given a simulated test case configuration for autonomous flight (the mission waypoints and obstacle locations and sizes), we want to generate a more challenging simulated test case by introducing an additional obstacle, to force the UAV to get too close to the obstacle (i.e., having a distance below a predefined safety threshold) while still completing the mission. This will create a risky environment for the UAV to operate the mission.

Most of the algorithm is identical to the algorithm instance described in Section 3.2 for autonomous flight replication, with few modifications. Specifically, we use the same test case properties (location, size, and rotation of the additional obstacle) and the same mutation operators (move, resize and rotate). The fundamental distinction is in the fitness function.

3.3.2 Fitness Function

We define the fitness such that the algorithm is guided to get the drone close to all the obstacles in general and close to the border of one obstacle in particular. Correspondingly, the fitness function has two components:

$$\begin{aligned}
\mathit{sum\_dist} &= \min_{p \in \text{trj.points}} \sum_{o \in \text{obs}} d(p, o) \\
\mathit{min\_dist} &= \min_{o \in \text{obs}, p \in \text{trj.points}} d(p, o) \\
\mathit{fitness} &= \mathit{sum\_dist} + 2 \times \mathit{min\_dist}
\end{aligned} \tag{2}$$

sum_dist accounts for the minimum distance of a single trajectory point to all of the obstacles combined (favoring obstacles closer to each other), while *min_dist* accounts for the minimum distance of the trajectory to any of the obstacles (to be compared against the safety distance). We give the *min_dist* component a weight that is twice the weight of the *sum_dist* component, because it is expected to contribute the most to the generation of risky test scenarios.

In this case, since in multiple runs of the same test case (parallel simulations), the flight trajectories can differ significantly from each other in corner cases (test cases with low *min_dist*), instead of taking the average trajectory, we take the trajectory with the lowest fitness value when returning the fitness value for a given mutation of a test scenario.

4 Experimental Results

The *goal* of our empirical evaluation is to assess SURREALIST's ability to replicate a logged UAV flight and to manipulate it to expose challenging flight conditions. In this section, we elaborate on the research questions, evaluation scenarios, and the results obtained when evaluating our approach.

4.1 Research Questions

4.1.1 RQ$_1$ [Flight Replication]

Can we generate simulated test cases that faithfully replicate autonomous flight trajectories? The goal is to replicate the test environment in a way that makes the simulated flight trajectory as similar as possible to the original one, using only logged data as input information. The variable test properties are the environment configurations where the UAV flies a predefined mission autonomously. We investigate an environment setup where placing an obstacle on the map can influence the trajectory, making it more or less similar to the logged one.

4.1.2 RQ$_2$ [Test Generation]

Can we modify simulated test case properties of autonomous flights to make them more challenging for the UAV autonomous controller? The goal is to investigate the possibility of generating more challenging test cases based on an existing one, replicated from a real-world test. Specifically, we investigate a similar environment setup as RQ$_1$ where placing an additional obstacle properly can result in unsafe or faulty behavior of the UAV.

4.2 Subject and Original Flights

The *subject* of our experiments is PX4's [41] Autopilot controlling a quad-copter. For RQ_1, we consider an original flight conducted in a real-world environment containing an obstacle along the mission route. We set up the *PX4 Vision Autonomy Development Kit*[4], enable module PX4-Avoidance [54], and define a survey mission consisting of taking off to 3 meters altitude, flying towards a waypoint at about 20m distance, and landing. The test field is an empty parking area, with a cargo container sized about $2.5m \times 12m \times 2.5m$ placed in the middle. The original flight trajectory, as extracted from the flight log [33] is shown in Figure 1 (left) as a blue line.

For RQ_2, we consider a simulated test case in Gazebo [52] with a similar setup (taking off to 10 meters altitude, flying towards a waypoint at about 50m distance, and flying back to the landing point, about 12m to the left of the starting point). We put a box-like obstacle (representing a small building) sized $8m \times 5m \times 20m$ on the direct route towards the destination. Then, we ran SURREALIST to generate more challenging environment configurations, obtained by adding a second obstacle to the environment. The flight trajectory, as extracted from the flight log, is shown in Figure 2 (middle).

4.3 Metrics and Experimental Procedure

To run our experiments, we set the hyper-parameters of Algorithms 1,2 (see Section 3.1) to the values reported in Table 1. To evaluate our approach, we run SURREALIST to replicate the flight trajectory (RQ_1) or generate test cases (RQ_2), applying 10 repetitions with the same configurations, to gain statistical validity of our results. To deal with the non-determinism of simulated trajectories, at each step of the algorithms, we run multiple simulations in parallel (5 for RQ_1 and 10 for RQ_2 because of the higher non-determinism in the corner test cases; see parameter *sim. runs* in Table 1). We run SURREALIST inside Docker containers in a Kubernetes cluster, with main algorithm containers limited to 1.5 virtual CPUs (VCPUs) and 15GB of Ram and simulation containers (running PX4 and Gazebo) limited to 6 VCPUs and 2 GB.

For the best found solutions of each algorithm repetition, we computed the metrics in Table 2. We measure the reduction in DTW distance from the original (*org*) when flight reproduction is achieved by SURREALIST (*best*) with respect to the search *seed*, as well as the reduction in Fréchet distance [24] for RQ_1. Fréchet distance is defined as the maximum distance observed when traveling through the two trajectories (original and replicated flights), considering an optimal mapping of the points visited along the two trajectories. For RQ_2, the seed for the additional obstacle has the same size as the first obstacle and is placed $15m$ to the right side of it, so that it does not affect the flight trajectory compared to the original test (figure 2, middle). To ensure the realism of the second obstacle, we kept its initial size and angle and used only the *Obstacle Move* mutation. For RQ_2, we measure the reduction of the minimum distance between the UAV and obstacles as well as the percentage of failing (crashing to obstacle) and unsafe (getting closer than 1.5m) simulations for the best generated test cases. We also report the needed evaluation budget (solution evaluations through simulation) to reach the best solutions and the average evaluation time for each solution (run the parallel simulations and process the logs). For each of these metrics, we report the average across 10 repetitions.

Once we have collected all the data, we used statistical tests to verify whether there is a statistically significant difference between the seed and the best solution for both RQs across the algorithm repetitions. We employed parametric tests since the Shapiro-Wilk test revealed that the distributions across all experiments follow a Gaussian distribution ($p \gg 0.05$). Hence, we used the one-way Anova test with a p-value threshold of 0.05. We also computed the effect-size of the observed differences using the Vargha-Delaney ($\hat{A}_{12}$) statistic [65]. The Vargha-Delaney ($\hat{A}_{12}$) statistic classifies the quantitative effect size values into four qualitative levels (*negligible*, *small*, *medium*, and *large*),

[4] https://docs.px4.io/v1.12/en/complete_vehicles/px4_vision_kit.html

Table 1: Experiment Hyper-parameters

RQ	Parameter	Value
$RQ_{1,2}$	repetition	10
$RQ_{1,2}$	MAX_SEQ.	5
$RQ_{1,2}$	MAX_IT	5
$RQ_{1,2}$	default_step	4m (MOVE, RESIZE), 30° (ROTATE)
$RQ_{1,2}$	default_value	0m (MOVE, RESIZE), 0° (ROTATE)
$RQ_{1,2}$	ϵ	0
RQ_1	budget	200 (seed 1), 100 (seed 2)
RQ_1	min_rounds	4 (seed 1), 2 (seed 2)
RQ_1	sim. runs	5
RQ_2	budget	50
RQ_2	min_rounds	2
RQ_2	sim. runs	10

Table 2: Experiment Evaluation Metrics

Metric	Definition
DTW Reduction (RQ_1)	$= 1 - \frac{DTW(best,org)}{DTW(seed,org)}(\%)$
Fréchet Reduction (RQ_1)	$= 1 - \frac{Frechet(best,org)}{Frechet(seed,org)}(\%)$
Min_dist Red. (RQ_2)	$= 1 - \frac{Min\_dist(best)}{Min\_dist(seed)}(\%)$
Crash & Unsafe Rate (RQ_2)	= % of crash & unsafe in the simulations of best
P-value	comparing seed and best fitness distributions
Effect Size	comparing seed and best fitness distributions
Needed Budget	= # of evaluations to reach best
Eval. Time	= average time for each solution evaluation

which are easier to interpret.

4.4 Results

4.4.1 RQ_1 [Flight Replication]

The seed obstacle is a small ($3m \times 3m \times 3m$) box, aligned with the direct path between takeoff and landing position as extracted from the original log. We do the experiments with two different positioning of this obstacle as seed, illustrated in Figure 1 (left). One is placed right below the center of the direct path (seed 1) so that the UAV is probable to fly around the obstacle from the right side, and the other is placed on the opposite side (seed 2) so that the UAV is more likely to fly to the left side. The two choices aim to analyze if the algorithm is equally effective when the starting trajectory is on different sides of the obstacle.

As can be seen in Figure 1, showing the best final solution across 10 runs for Seed 1, SURREALIST was able to position and size the obstacle very well, so that the trajectory of the replayed flight is almost identical to the original one (with less than 75cm Fréchet distance). During this specific run, the obstacle was moved 2m upwards, rotated 30° clockwise, and the height was increased by 2.8m by the algorithm over the iterations. Although the final obstacle properties are not identical over the 10 runs, the very low DTW between the simulated and original trajectories shows that we do not need to replicate the exact same obstacle configurations to be able to test the UAV in the same manner.

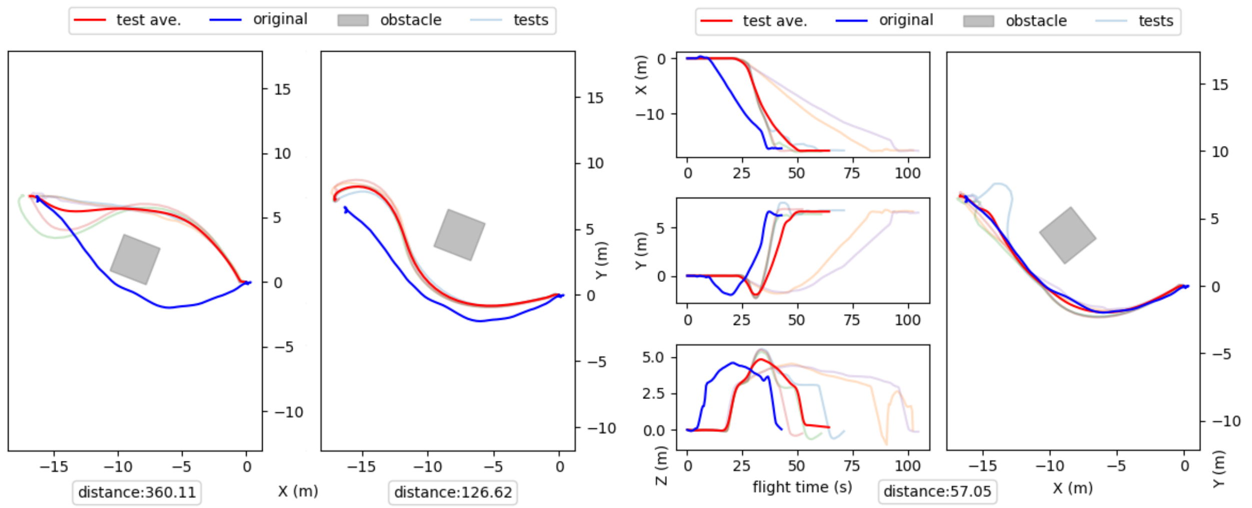
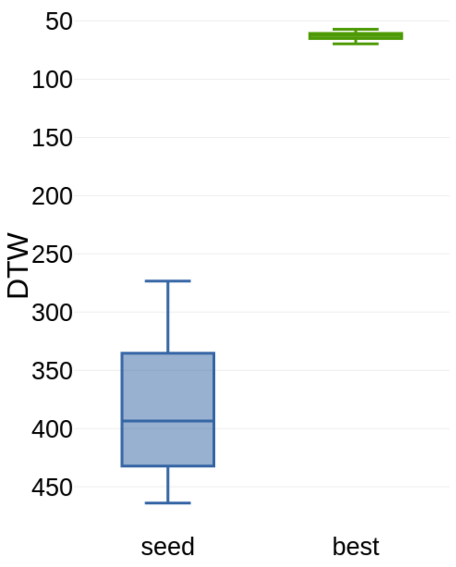


Figure 1: RQ$_1$ Seeds 1 and 2 (left) and Final (right) solutions (simulated in Gazebo) compared to the real-world flight

Table 3: RQ$_1$ Evaluation metrics for both seeds

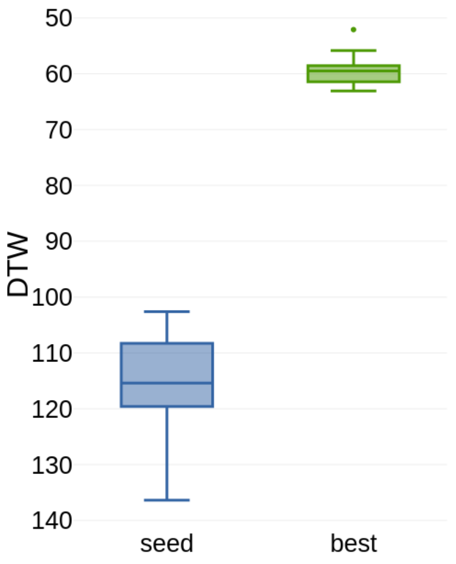
seed 1

Metric	Ave.
Seed DTW	383.5
Best DTW	63.15
DTW Red.	83.1%
Seed Fréchet	6.87 (m)
Best Fréchet	1.14 (m)
Fréchet Red.	83.4 %
P-value	1.9 e-12
Effect Size	7.9
Needed Budget	74.8
Eval. Time	152.9 (s)



seed 2

Metric	Ave.
Seed DTW	116.1
Best DTW	59.1
DTW Red.	48.8%
Seed Fréchet	2.25 (m)
Best Fréchet	1.05 (m)
Fréchet Red.	51.6 %
P-value	1.2 e-12
Effect Size	8.13
Needed Budget	65
Eval. Time	158.9 (s)



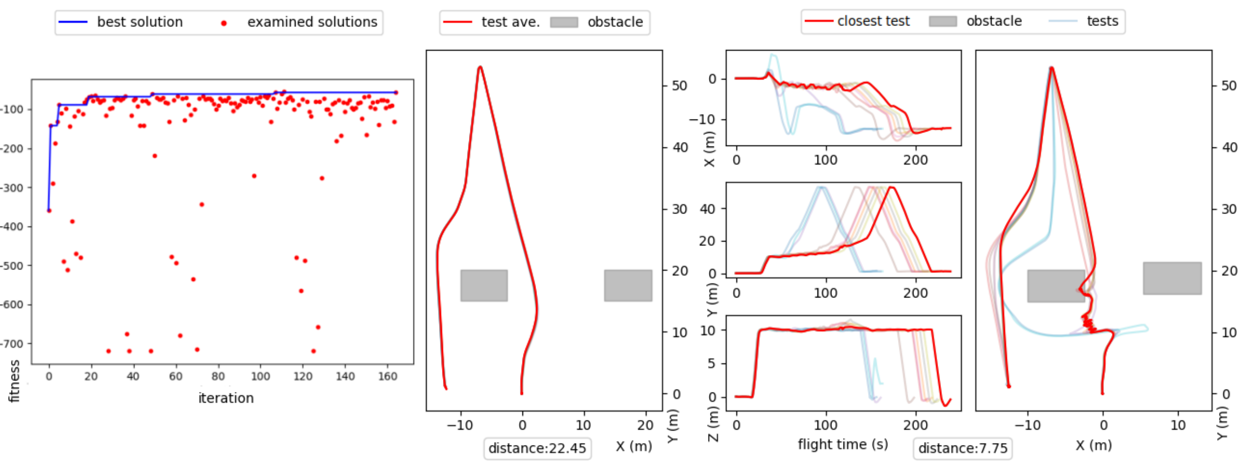


Figure 2: RQ$_1$ Fitness progress over iterations (left), RQ$_2$ Seed (middle) and Final solutions (right)

As reported in Table 3, the algorithm was able to locate and size the obstacles consistently well in all 10 runs for both seeds, finding solutions with an average DTW of 63 and 59 respectively, which corresponds to an almost identical trajectory, while reducing the seed DTW by 83% and 48% with respect to the distance obtained with the seed obstacles. The two compared DTW distributions are Gaussian ($p > 0.3$ in the Shapiro-Wilk test), so we could use the Anova test: a low p-value ($\ll 0.01$) and a large effect size ($7.9 \gg 0.8$) suggest that the improvement achieved by the algorithm is statistically significant and extremely large.

Table 3 reports both the average required budget (i.e., the average number of evaluations) and the evaluation time (in seconds). With the search budget set to 200 simulations for Seed 1 (100 for Seed 2), on average the final solution was found after 75 (65) evaluations and about 3.5 minutes were used to run all the computations (parallel simulations and distance calculations) at each solution evaluation, adding up to almost 5 hours for each run of the experiment. Figure 2 (left) shows the convergence of the fitness over the iterations. The initial round of mutations (iterations 1-58) are contributing to most improvements in the fitness function, while during the second and third rounds (59-117, 118-165) only marginal improvements are observed.

Discussion on minimum feasible distance: As can be seen in the flight trajectories in Figure 1 (right), the individual runs of the exact same test case can have slightly different trajectories due to the simulation randomness. Although we already mitigated this with the averaging method discussed in Section 3.2.2, it also limits the minimum DTW distance that can be possibly reached from the original flight. To estimate the potential lower bound due to the simulation randomness, we took the average flight trajectories from the seed solutions of all the 10 runs, and computed their pairwise DTW distance to each other. These distances were in the range $[15.9 - 121.3]$ with an average of 58.5. This means that, due to the simulation randomness, even replicating a simulated flight in the same simulator, by putting the exact same test configurations, could reach a DTW distance within this range. Hence, this average (58.5) can be considered as the bottom-line for our optimization process, which indeed reached an average DTW distance of 63.15 when starting from Seed 1; 59.1 when starting from Seed 2.

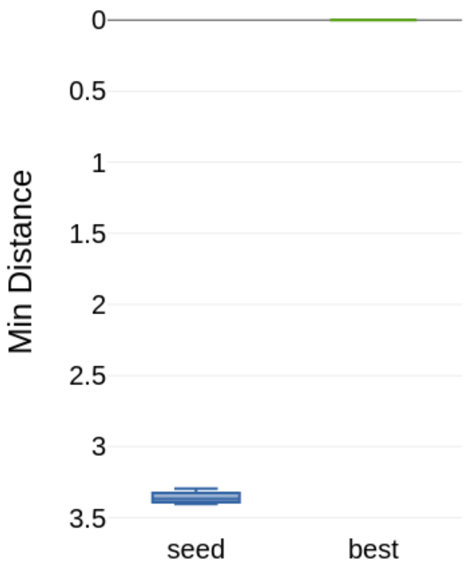
RQ$_1$: The information available in the flight logs allows searching for optimal test properties that faithfully replicate UAV *autonomous* flight trajectories in simulation.

4.4.2 RQ$_2$ [Test Generation]

As can be seen from the best final solution across 10 runs in Figure 2 (right), SURREALIST was able to position and size the second obstacle in a way that the UAV (i) was forced to behave in a non-deterministic way across the parallel simulations; (ii) experienced an unsafe behavior, often too

Table 4: RQ_2 Evaluation metrics

Metric	Ave.
Seed Min_dist	3.36
Best Min_dist	0
Min_dist Red.	100%
Crash Rate	25 %
Unsafe Rate	84 %
P-value	6.7 e-12
Effect Size	119.9
Needed Budget	36.8
Eval. Time	388.1 (s)



close to the first obstacle; (iii) even worse, occasionally crashed into the obstacle in some simulations. The second obstacle was moved by SURREALIST to almost $8m$ to the left and $1.1m$ upwards, making it increasingly harder for the UAV to follow the path. Interestingly, if we position the obstacles even slightly closer to each other, the UAV would act in a safe and deterministic way, always flying risk-free around the left of the first obstacle. Indeed, we have found a bug regarding the violation of a safety requirement, i.e., maintaining a minimum distance to the obstacles.

As reported in Table 4, the algorithm was able to find crashing test cases consistently in all 10 runs, forcing the UAV to get as close as $0m$ to the obstacle, down from the $3.3m$ safe distance for the seed. Also, on average, the UAV crashed into the obstacle in around 3 out of the 10 simulations for the best test cases found, and got unsafely close (<1.5m) in 5 more.

The two compared distance distributions are Gaussian ($p > 0.69$ in the Shapiro-Wilk test), so we could use the Anova test: a low p-value ($\ll 0.01$) and a large effect size ($119.9 \gg 0.8$) suggests that the improvement achieved by the algorithm is statistically significant and extremely large.

RQ_2: Modifying a simulation-based test case allows generating challenging test cases that can expose the UAV to unsafe behaviors or even crashes.

4.5 Threats to validity

Threats to *construct validity* concern the metrics used to draw a relation between theory and observation. The distance between the trajectory reproduced in the simulator and the original log's trajectory is affected by randomness, due to various sources of noise and non-determinism that affect the simulation environment (e.g., the effect of the wind or the multi-threaded execution in the simulator). Hence, we cannot consider one solution to be closer to the log than another if their trajectories have a small difference (see "Discussion on minimum feasible distance" in Section 4.1). To address this threat we introduced a distance threshold *MIN_DIST* and two trajectories are regarded as equivalent if their distance is lower than *MIN_DIST*. When comparing a simulated trajectory to the log data, we take the average trajectory over 5 simulations to reduce the effects of non-determinism. To gather statistically significant results, we repeated our experiments 10 times. For what concerns the choice of the distance metric used to compare trajectories during the evaluation, we adopted DTW [9] and Fréchet[24], well-known metrics that have already been used before for comparing UAV flight trajectories [2].

Threats to *internal validity* concern the technologies used to generate the UAV scenarios and tests. To increase the generality of our results we could have used also other supported simulators (e.g., jMAVSim). However, it is acceptable to use Gazebo as PX4's reference simulation environment since it is suitable for testing the obstacle avoidance functionalities. Another threat that affects the internal validity is the choice of the seeds for the obstacles used to answer RQ_1 and RQ_2. We used two different seeds for RQ_1 with the flight trajectory being on different sides of them, and a seed additional obstacle for RQ_2 that does not affect the flight trajectory. While our choices were aimed to minimize the effect of the seed solution on the evaluations, a replication with other obstacle types and seed position/size is needed to corroborate our findings.

Threats to *external validity* concern the generalization of our findings. Although we experimented with a widely used UAV firmware and simulator (PX4 and Gazebo), we cannot claim that our results can be generalized to other UAV platforms or other CPS domains. Therefore, further replications and studies considering more CPS domains are desirable.

Conclusion validity threats concern our conclusions about the improvement brought by our algorithm. After verifying its applicability conditions, we used the parametric Anova test to verify that there is a statistically significant difference between the fitness values of seed and final solutions.

5 Related Works

Wang *et al.* [66] studied UAV software bugs from UAV Autopilot platforms (PX4 [41] and Ardupilot [6]) and created a taxonomy of UAV bugs and identified their root causes. They report that developers mainly use simulators for reproducing bugs, but setting up realistic-enough simulation environments that capture the same bugs as physical tests is a hard and expensive task. Afzal *et al.* [4, 2] studied the challenges of testing robotic systems and recognize the *engineering complexity* of the test environment, including the design of realistic inputs to test the system, as one of the biggest challenges. Afzal *et al.* [3] surveyed robotic practitioners on their use of simulators for testing robots and identified the *reality gap* of the robot behavior in simulation and the *reproducibility* of issues encountered in real or simulated tests in simulation as the top challenges they face. Timperly *et al.* [63] conducted an empirical study on fixed bugs in Ardupilot [6] and found that many bugs can be potentially detected before field tests if proper simulation-based testing is in place.

Lindval *et al.* [38] developed a framework for automated testing of autonomous drones in simulation with the aim of solving the test oracle definition problem. Recently, Woodlief et al. proposed PHYS-FUZZ [68], a fuzzing approach tailored specifically for testing mobile robots, taking into account the physical attributes and hazards of such robots. To address the simulation-reality gap, Hildebrandt *et al.* [29] propose a mixed-reality approach for testing UAVs. Their approach, called *world-in-the-loop* simulation, integrates and mixes sensor data from both the simulated and real world, and feeds these mixed sensor inputs to the system under test (UAV).

Complementary, we address UAV simulation-based testing challenges concerning realistic test cases, engineering complexity, and field test reproducibility, by an approach that faithfully replicates real-world test scenarios in simulation and generates similar but challenging variants of these test cases.

Testing of Deep Learning (DL) based systems is a research area that has attracted a growing interest in the last few years. Traditional testing techniques have been adapted to the specific features of machine learning components, addressing problems such as test input generation [40, 70, 21, 23, 39], test oracle definition [62, 31], and test adequacy [46, 40, 35].

In the existing DL testing literature, most related works deal with automated test data generation. Only a few input generators are model-based and expose failures within a simulated environment. Gambi et al. [28], Stocco et al. [62]. Riccio et al. [56], Birchler et al. [11, 12, 10], and Abdessalem et al. [1] test self-driving cars by generating failure-inducing driving scenarios. We share with them the usage of a simulator to control the environment in which an autonomous vehicle is tested, but differently from these works, we aim at generating realistic simulated test cases by first reproducing flying conditions experienced in the field and then manipulating such conditions to identify failure scenarios.

6 Conclusion and Future Work

Simulation-based testing of UAVs is an important quality assurance step before systems can be released to production. We have proposed a generic adaptive greedy algorithm that can be instantiated to replicate a flight trajectory observed in the field and manipulate it in order to expose misbehaviors. Our experimental results show that SURREALIST, implementing our approach, can achieve faithful flight replication by reconstructing the obstacles encountered along the mission's path. After replication, SURREALIST is also able to manipulate the obstacles in the environment to find challenging conditions that lead to unsafe behavior of the UAV.

In our future work, we plan to investigate surrogate models that can predict the behavior of the UAV without actually running any simulations. Such models can guide our adaptive greedy search algorithm at low computational cost, making the search more efficient and potentially capable of exploring more complex critical scenarios. We also intend to experiment with additional UAV models and environment configurations, including e.g. different weather conditions and obstacle types.

References

[1] Raja Ben Abdessalem, Shiva Nejati, Lionel C. Briand, and Thomas Stifter. Testing vision-based control systems using learnable evolutionary algorithms. In *International Conference on Software Engineering*, pages 1016–1026. ACM, 2018.

[2] Afsoon Afzal. *Automated Testing of Robotic and Cyberphysical Systems*. PhD thesis, Carnegie Mellon University, 2021.

[3] Afsoon Afzal, Deborah S Katz, Claire Le Goues, and Christopher S Timperley. Simulation for robotics test automation: Developer perspectives. In *Conference on Software Testing, Verification and Validation*, pages 263–274. IEEE, 2021.

[4] Afsoon Afzal, Claire Le Goues, Michael Hilton, and Christopher Steven Timperley. A study on challenges of testing robotic systems. In *International Conference on Software Testing, Validation and Verification*, pages 96–107. IEEE, 2020.

[5] Miguel Alcon, Hamid Tabani, Jaume Abella, and Francisco 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*, pages 426–433. IEEE, 2021.

[6] Ardupilot.org. Ardupilot – versatile, trusted, open, 2007. accessed: 07.02.2022.

[7] Shay Artzi, Sunghun Kim, and Michael D. Ernst. Recrash: Making software failures reproducible by preserving object states. In Jan Vitek, editor, *Object-Oriented Programming, European Conference*, volume 5142 of *Lecture Notes in Computer Science*, pages 542–565. Springer, 2008.

[8] Eulalia Balestrieri, Pasquale Daponte, Luca De Vito, Francesco Picariello, and Ioan Tudosa. Sensors and measurements for UAV safety: An overview. *Sensors*, 21(24):8253, 2021.

[9] Donald J Berndt and James Clifford. Using dynamic time warping to find patterns in time series. In *KDD workshop*, volume 10, pages 359–370. Seattle, WA, USA:, 1994.

[10] Christian Birchler, Nicolas Ganz, Sajad Khatiri, Alessio Gambi, and Sebastiano Panichella. Cost-effective simulation-based test selection in self-driving cars software. *Science of Computer Programming (SCP)*, 2022.

[11] Christian Birchler, Nicolas Ganz, Sajad Khatiri, Alessio Gambi, and Sebastiano Panichella. Cost-effective simulation-based test selection in self-driving cars software with sdc-scissor. In *the 29th IEEE International Conference on Software Analysis, Evolution, and Reengineering*, 2022.

[12] Christian Birchler, Sajad Khatiri, Bill Bosshard, Alessio Gambi, and Sebastiano Panichella. Machine learning-based test selection for simulation-based testing of self-driving cars software. *Empirical Software Engineering*, 2022.

[13] Christian Birchler, Sajad Khatiri, Pouria Derakhshanfar, Sebastiano Panichella, and Annibale Panichella. Single and multi-objective test cases prioritization for self-driving cars in virtual environments. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 2022.

[14] Kinga Bojarczuk, Natalija Gucevska, Simon M. M. Lucas, Inna Dvortsova, Mark Harman, Erik Meijer, Silvia Sapora, Johann George, Maria Lomeli, and Rubmary Rojas. Measurement challenges for cyber cyber digital twins: Experiences from the deployment of facebook's WW simulation system. In *International Symposium on Empirical Software Engineering and Measurement*, pages 2:1–2:10. ACM, 2021.

[15] Carlos Carbone, Dario Albani, Federico Magistri, Dimitri Ognibene, Cyrill Stachniss, Gert Kootstra, Daniele Nardi, and Vito Trianni. Monitoring and mapping of crop fields with UAV

swarms based on information gain. In *Distributed Autonomous Robotic Systems - International Symposium*, volume 22 of *Springer Proceedings in Advanced Robotics*, pages 306–319. Springer, 2021.

[16] Hong Chen. Applications of cyber-physical system: A literature review. *Journal of Industrial Integration and Management*, 02(03):1750012, 2017.

[17] Ying Chen and Néstor O. Pérez-Arancibia. Controller synthesis and performance optimization for aerobatic quadrotor flight. *IEEE Transactions on Control Systems Technology*, 28(6):2204–2219, 2020.

[18] Sudipta Chowdhury, Omid Shahvari, Mohammad Marufuzzaman, Xiaopeng Li, and Linkan Bian. Drone routing and optimization for post-disaster inspection. *Comput. Ind. Eng.*, 159:107495, 2021.

[19] Raffaello D'Andrea. Guest editorial can drones deliver? *IEEE Trans Autom. Sci. Eng.*, 11(3):647–648, 2014.

[20] Rodrigo Delgado, Miguel Campusano, and Alexandre Bergel. Fuzz testing in behavior-based robotics. In *International Conference on Robotics and Automation*, pages 9375–9381. IEEE, 2021.

[21] Samet Demir, Hasan Ferit Eniser, and Alper Sen. Deepsmartfuzzer: Reward guided test generation for deep learning. In *Workshop on Artificial Intelligence Safety 2020 (IJCAI-PRICAI 2020)*, volume 2640 of *CEUR Workshop Proceedings*, pages 134–140. CEUR-WS.org, 2020.

[22] Pouria Derakhshanfar, Xavier Devroey, Annibale Panichella, Andy Zaidman, and Arie van Deursen. Botsing, a search-based crash reproduction framework for java. In *International Conference on Automated Software Engineering*, pages 1278–1282. IEEE, 2020.

[23] Swaroopa Dola, Matthew B. Dwyer, and Mary Lou Soffa. Distribution-aware testing of neural networks using generative models. In *International Conference on Software Engineering (ICSE)*, pages 226–237, 2021.

[24] Thomas Eiter and Heikki Mannila. Computing discrete fréchet distance. *Citeseer*, 1994.

[25] Philipp Foehn, Angel Romero, and Davide Scaramuzza. Time-optimal planning for quadrotor waypoint flight. *Sci. Robotics*, 6(56):1221, 2021.

[26] Christian Forster, Matia Pizzoli, and Davide Scaramuzza. Svo: Fast semi-direct monocular visual odometry. In *International Conference on Robotics and Automation (ICRA)*, pages 15–22, 2014.

[27] Alessio Gambi, Tri Huynh, and Gordon Fraser. Generating effective test cases for self-driving cars from police reports. In *ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 257–267. ACM, 2019.

[28] Alessio Gambi, Marc Müller, and Gordon Fraser. Automatically testing self-driving cars with search-based procedural content generation. In *ACM SIGSOFT International Symposium on Software Testing and Analysis*, pages 318–328. ACM, 2019.

[29] Carl Hildebrandt and Sebastian Elbaum. World-in-the-loop simulation for autonomous systems validation. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 10912–10919. IEEE, 2021.

[30] Ziqi Huang, Yang Shen, Jiayi Li, Marcel Fey, and Christian Brecher. A survey on ai-driven digital twins in industry 4.0: Smart manufacturing and advanced robotics. *Sensors*, 21(19):6340, 2021.

[31] Gunel Jahangirova, Andrea Stocco, and Paolo Tonella. Quality metrics and oracles for autonomous vehicles testing. In *Conference on Software Testing, Verification and Validation*, pages 194–204. IEEE, 2021.

[32] Wei Jin and Alessandro Orso. Bugredux: Reproducing field failures for in-house debugging. In *International Conference on Software Engineering*, pages 474–484. IEEE Computer Society, 2012.

[33] Sajad Khatiri, Sebastiano Panichella, and Paolo Tonella. Flight log for "obstacle avoidance with a simple mission and a cargo container as obstacle", 2023.

[34] Sajad Khatiri, Sebastiano Panichella, and Paolo Tonella. "surrealist: Simulation-based test case generation for uavs in the neighborhood of real flights", 2023.

[35] Jinhan Kim, Robert Feldt, and Shin Yoo. Guiding deep learning system testing using surprise adequacy. In *International Conference on Software Engineering*, pages 1039–1049. IEEE / ACM, 2019.

[36] Nathan P. Koenig and Andrew Howard. Design and use paradigms for gazebo, an open-source multi-robot simulator. In *International Conference on Intelligent Robots and Systems*, pages 2149–2154. IEEE, 2004.

[37] Rui Li, Huai Liu, Guannan Lou, James Xi Zheng, Xiao Liu, and Tsong Yueh Chen. Metamorphic testing on multi-module UAV systems. In *International Conference on Automated Software Engineering*, pages 1171–1173. IEEE, 2021.

[38] Mikael Lindvall, Adam Porter, Gudjon Magnusson, and Christoph Schulze. Metamorphic model-based testing of autonomous systems. In *International Workshop on Metamorphic Testing*, pages 35–41. IEEE, 2017.

[39] Lei Ma, Felix Juefei-Xu, Minhui Xue, Bo Li, Li Li, Yang Liu, and Jianjun Zhao. DeepCT: Tomographic combinatorial testing for deep learning systems. In *International Conference on Software Analysis, Evolution and Reengineering*, pages 614–618. IEEE, 2019.

[40] Lei Ma, Felix Juefei-Xu, Fuyuan Zhang, Jiyuan Sun, Minhui Xue, Bo Li, Chunyang Chen, Ting Su, Li Li, Yang Liu, Jianjun Zhao, and Yadong Wang. Deepgauge: Multi-granularity testing criteria for deep learning systems. In *International Conference on Automated Software Engineering*, pages 120–131. ACM, 2018.

[41] Lorenz Meier, Dominik Honegger, and Marc Pollefeys. Px4: A node-based multithreaded open source robotics framework for deeply embedded platforms. In *international conference on robotics and automation*, pages 6235–6240. IEEE, 2015.

[42] Daniel Mellinger and Vijay Kumar. Minimum snap trajectory generation and control for quadrotors. In *2011 IEEE International Conference on Robotics and Automation*, pages 2520–2525, 2011.

[43] Satish Narayanasamy, Gilles Pokam, and Brad Calder. Bugnet: Continuously recording program execution for deterministic replay debugging. In *International Symposium on Computer Architecture*, pages 284–295. IEEE Computer Society, 2005.

[44] Vuong Nguyen, Stefan Huber, and Alessio Gambi. SALVO: automated generation of diversified tests for self-driving cars from existing maps. In *International Conference on Artificial Intelligence Testing*, pages 128–135. IEEE, 2021.

[45] Helen Oleynikova, Zachary Taylor, Marius Fehr, Roland Siegwart, and Juan I. Nieto. Voxblox: Incremental 3d euclidean signed distance fields for on-board MAV planning. In *International Conference on Intelligent Robots and Systems*, pages 1366–1373. IEEE, 2017.

[46] Kexin Pei, Yinzhi Cao, Junfeng Yang, and Suman Jana. Deepxplore: Automated whitebox testing of deep learning systems. *Commun. ACM*, 62(11):137?145, October 2019.

[47] Scott Drew Pendleton, Hans Andersen, Xinxin Du, Xiaotong Shen, Malika Meghjani, You Hong Eng, Daniela Rus, and Marcelo H. Ang. Perception, planning, control, and coordination for autonomous vehicles. *Machines*, 5(1), 2017.

[48] François Petitjean, Alain Ketterlin, and Pierre Gançarski. A global averaging method for dynamic time warping, with applications to clustering. *Pattern recognition*, 44(3):678–693, 2011.

[49] Andrea Piazzoni, Jim Cherian, Mohamed Azhar, Jing Yew Yap, James Lee Wei Shung, and Roshan Vijay. Vista: a framework for virtual scenario-based testing of autonomous vehicles. In *International Conference on Artificial Intelligence Testing*, pages 143–150. IEEE, 2021.

[50] Pixhawk.org. Pixhawk | the hardware standard for open-source autopilots., 2021.

[51] PX4. Px4 simulation. `https://docs.px4.io/v1.12/en/simulation/`, 2021.

[52] PX4.io. Gazebo simulation | px4 user guide, 2022. accessed: 07.02.2022.

[53] PX4.io. Log analysis using flight review | px4 user guide, 2022. accessed: 07.02.2022.

[54] PX4.io. Obstacle avoidance | px4 user guide, 2022. accessed: 07.02.2022.

[55] Tong Qin, Peiliang Li, and Shaojie Shen. Vins-mono: A robust and versatile monocular visual-inertial state estimator. *IEEE Trans. Robotics*, 34(4):1004–1020, 2018.

[56] Vincenzo Riccio and Paolo Tonella. Model-based exploration of the frontier of behaviours for deep learning system testing. In *ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 876–888, 2020.

[57] Charles Richter, Adam Bry, and Nicholas Roy. Polynomial trajectory planning for aggressive quadrotor flight in dense indoor environments. In *Robotics Research - International Symposium ISRR*, volume 114 of *Springer Tracts in Advanced Robotics*, pages 649–666. Springer, 2013.

[58] Mrinmoy Sarkar, Xuyang Yan, Shamila Nateghi, Bruce J. Holmes, Kyriakos G. Vamvoudakis, and Abdollah Homaifar. A framework for testing and evaluation of operational performance of multi-uav systems. In *Intelligent Systems and Applications - Proceedings of the 2021 Intelligent Systems Conference, IntelliSys*, volume 294 of *Lecture Notes in Networks and Systems*, pages 355–374. Springer, 2021.

[59] Mozhan Soltani, Annibale Panichella, and Arie van Deursen. A guided genetic algorithm for automated crash reproduction. In *International Conference on Software Engineering*, pages 209–220. IEEE / ACM, 2017.

[60] Mozhan Soltani, Annibale Panichella, and Arie van Deursen. Search-based crash reproduction and its impact on debugging. *IEEE Trans. Software Eng.*, 46(12):1294–1317, 2020.

[61] Andrea Di Sorbo, Fiorella Zampetti, Corrado A. Visaggio, Massimiliano Di Penta, and Sebastiano Panichella. Automated identification and qualitative characterization of safety concerns reported in uav software platforms. *Transactions on Software Engineering and Methodology*, 2022.

[62] Andrea Stocco, Michael Weiss, Marco Calzana, and Paolo Tonella. Misbehaviour prediction for autonomous driving systems. In *International Conference on Software Engineering*, pages 359–371, 2020.

[63] Christopher Steven Timperley, Afsoon Afzal, Deborah S Katz, Jam Marcos Hernandez, and Claire 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)*, pages 331–342. IEEE, 2018.

[64] Chris Cole Drone Wars UK. Accidents will happen - a review of military drone crash data as the uk considers allowing large military drone flights in its airspace, 2019.

[65] András Vargha and Harold D Delaney. A critique and improvement of the cl common language effect size statistics of mcgraw and wong. *Journal of Educational and Behavioral Statistics*, 25(2):101–132, 2000.

[66] Dinghua Wang, Shuqing Li, Guanping Xiao, Yepang Liu, and Yulei 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*, pages 20–31, 2021.

[67] Chathurika S. Wickramasinghe, Daniel L. Marino, Kasun Amarasinghe, and Milos Manic. Generalization of deep learning for cyber-physical system security: A survey. In *44th Annual Conference of the IEEE Industrial Electronics Society (IECON)*, pages 745–751, 2018.

[68] Trey Woodlief, Sebastian Elbaum, and Kevin Sullivan. Fuzzing mobile robot environments for fast automated crash detection. In *International Conference on Robotics and Automation*, pages 5417–5423. IEEE, 2021.

[69] Franz Wotawa. On the use of available testing methods for verification & validation of ai-based software and systems. In *Workshop on Artificial Intelligence Safety*, volume 2808 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2021.

[70] Xiaofei Xie, Lei Ma, Felix Juefei-Xu, Minhui Xue, Hongxu Chen, Yang Liu, Jianjun Zhao, Bo Li, Jianxiong Yin, and Simon See. Deephunter: A coverage-guided fuzz testing framework for deep neural networks. In *ACM SIGSOFT International Symposium on Software Testing and Analysis*, pages 146–157. Association for Computing Machinery, 2019.

[71] Fiorella Zampetti, Ritu Kapur, Massimiliano Di Penta, and Sebastiano Panichella. An empirical characterization of software bugs in open-source cyber–physical systems. *Journal of Systems and Software*, 192:111425, 2022.

[72] Xuejun Zhang, Yang Liu, Yu Zhang, Xiangmin Guan, Daniel Delahaye, and Li Tang. Safety assessment and risk estimation for unmanned aerial vehicles operating in national airspace system. *Journal of Advanced Transportation*, 2018.