Towards a Metamorphic Testing Architecture for Software-Defined Drone Systems

Erik M. Fredericks
College of Computing
Grand Valley State University
Allendale, MI, USA
frederer@gvsu.edu

Mallory Jacobs
College of Computing
Grand Valley State University
Allendale, MI, USA
jacobmal@mail.gvsu.edu

Byron DeVries
College of Computing
Grand Valley State University
Allendale, MI, USA
devrieby@gvsu.edu

***Abstract*—Safety-critical systems, such as drones, are vulnerable to effects of uncertainty, where uncertainty can manifest as adverse weather conditions, unexpected human interactions, and misconfigured system settings. Moreover, it is humanly infeasible to test for all possible combinations of conditions that a system may experience through its lifetime during design. Additionally, drones are difficult to exhaustively test prior to deployment as real-world validation is costly in terms of time and equipment. Search-based testing is an approach for discovering new situations that a system may experience, however such tests can suffer from automatically inferring incorrect expected values/outcomes without domain knowledge (i.e., the oracle problem). As such, we propose a metamorphic testing framework for software-defined drone systems that uses exploratory search and aims to minimize the oracle problem at both design time and run time. We demonstrate our framework through a motivating example that illustrates each step of the process.**

***Index Terms*—software-defined systems, drones, evolutionary computation, metamorphic relations, metamorphic testing, cyber-physical systems**

I. Introduction

Drone systems are often subjected to uncertainty in the form of unanticipated weather scenarios, unforeseen human behaviors/interactions, and problems with system configuration (e.g., mistranslated from requirements, incorrectly updated at run time) [1]–[3]. Such systems are considered *safety critical* in that failure can result in catastrophic damage to the drone, significant monetary impact in the event of a crash, and injury/death of human bystanders. Metamorphic testing (MT) provides a validation strategy for mapping input scenarios to additional input scenarios where outputs can be directly compared [4]. Given that the operating contexts (i.e., instantiated combinations of internal and external parameters) that a system may experience can significantly change over time [5], an automated approach is necessary to both discover unknown situations and support a mapping strategy of inputs to outputs. This paper presents *DroneMR*, a proof of concept, evolutionary search-based technique for enhancing drone assurance against uncertainty.

It is humanly-infeasible to catalogue and test all possible combinations of operating contexts that a drone may experience over its lifetime [1], [5]–[7]. Previous techniques have aimed to address uncertainty at both design time and run time in terms of requirements monitoring, exploratory testing, and contextual metamorphic relations [5], [8]–[10]. Moreover, exploratory techniques for generating adverse environmental scenarios have also been explored to understand how systems react to uncertain situations [9], [11]–[14]. Such techniques are useful for anticipating and understanding how uncertainty can impact systems, however behaviors may be expressed that are not explicitly defined or intended according to their specifications. Metamorphic relations (MRs) can be derived to understand how an input or set of inputs impact related outputs, however they typically are limited to a small set of functions that can be managed by a domain expert. MRs are also traditionally difficult to automatically derive and often require a human to enumerate [15].

DroneMR generates a suite of novel operating contexts specific to drone systems using evolutionary computation, where representations of contexts are clustered and mapped to equivalency sets to reduce overhead. *DroneMR* infers a suite of MRs using an input set of requirements and derived utility functions to measure drone performance during simulation. MRs are then mapped to a set of MTs that are next validated in a real-world system. A test specification comprising the set of MTs are then provided as output for use in validation activities.

We describe and illustrate *DroneMR* using motivating examples from a micro-drone research application, where the drones are required to satisfy varying mission objectives. The remainder of this paper is organized as follows. Section II presents relevant background information and related work. Section III then presents *DroneMR*, and Section IV summarizes our efforts and presents future directions for research.

II. Background and Related Work

This section presents relevant background information and related work on safety-critical systems, evolutionary computation, and MRs/MTs.

A. Drones as Safety-Critical Systems

Safety-critical systems typically operate in conditions in which failure can result in catastrophic loss, including a significant financial impact, damage to the system or environment, and injury/death to human participants [16] As such, drones can be categorized as safety-critical systems.

Drones (i.e., unmanned aerial vehicles (UAVs)) are a type of aircraft that operate without a human pilot onboard. They are commonly equipped with different types of sensors, rotors, and other components that enable autonomous or remote-controlled flight [17]. Typical sensor capabilities include cameras, GPS, accelerometers, gyroscopes, magnetometers, barometers, and thermal/infrared sensors. Such sensors are used to support a wide range of tasks, from aerial photography to complex environmental monitoring and data collection.

A standard drone typically has multiple rotors that provide the necessary lift and maneuverability for flight. Common configurations include quadcopters (four rotors), hexacopters (six rotors), and octocopters (eight rotors). The choice of configuration affects the drone's stability, payload capacity, and power consumption [17]. Despite their advanced capabilities, drones are susceptible to various types of uncertainty that can impact their performance and mission success. These uncertainties can include:

- *Environmental Conditions*: Wind, rain, fog, and other weather conditions can affect a drone's flight stability and sensor accuracy [18].
- *Human Interactions*: Unexpected human activities can interfere with a drone's operations, such as obstacles suddenly appearing in its flight path. For example, the authors tested the obstacle avoidance algorithm for the drone shown in Figure 1 by tossing a sponge football in its path.
- *System Failures*: Hardware or software malfunctions, such as motor failure, sensor errors, or connectivity issues, can disrupt a drone's mission [19].
- *Configuration Errors*: Mistakes in the setup or programming of the drone can lead to incorrect behavior during flight [19].

Recognition of such uncertainties can be a non-trivial task, depending on the diversity and quality of equipped sensors. While dedicated sensors can alleviate specific instances (e.g., battery monitor for power consumption, humidity/rain sensor for wet conditions, etc.), some uncertainties must be inferred from fusing sensor data. For example, a wind event can be inferred from analysis of positional/rotational data, where a drone whose pose (i.e., position/rotation in space) drastically changes may have been impacted by an external event. Additionally, a drone that is having difficulty maintaining altitude may have issues in powering/controlling its rotors. Regardless, the drone controller must include introspection capabilities for both recognizing when its key objectives are being unsatisfied and moreover define thresholds for taking action (e.g., performing a self-reconfiguration, entering a failsafe state, etc.).

Drones are often equipped with advanced algorithms and systems for real-time monitoring and adaptive response to mitigate and/or avoid these uncertainties. For instance, obstacle avoidance systems use optical flow and other sensor data to detect and navigate around obstacles. Autonomous navigation algorithms enable drones to adjust their flight paths dynamically in response to environmental changes. Additionally, redundancy in critical components (e.g., multiple sensors and fail-safe mechanisms) ensures that the drone can continue to operate even if one component fails [20].

Figure 1 shows the bitcraze Crazyflie 2.1 micro-drone platform that we use as part of our research testbed [21]. Available sensors include a multi-ranger deck (i.e., obstacle avoidance) and optical flow deck (i.e., distance from ground). For this paper we motivate the use of both drone decks, however other sensor platforms and mission objectives may be installed/defined as needed.

Fig. 1. CrazyFlie 2.1 Micro-Drone [21].

Drone simulators have been previously created for assessing behavior in varying situations [22]–[24]. For this paper we use the Webots environment with a CrazyFlie plugin for visualizations and proof of concept [23], however a simulator specific to the Crazyflie (CrazySim [24]) that supports both Python and the Robot Operating System (ROS) may reduce the issues presented by the simulation gap (i.e., Webots code does not directly translate to the physical system). Other examples of using software-defined systems for simulation include those in the Internet of Things space, such as SDIoT [25], leveraging software-defined networking as a mechanism for implementing a fog architecture (i.e., the space between edge and cloud computing) [26], and in abstracting the hardware and mechanisms necessary for specifying a cloud-based system [27]. Additionally, software-defined systems have been used in validating safety critical systems, such as testing edge/fog systems involving cooperation between live systems and digital twins [28] and in managing the energy consumption of software-defined systems when performing testing [29].

B. Evolutionary Computation

Evolutionary computation (EC) is a stochastic approach for searching through massive solution spaces to solve complex problems where an optimal solution is not readily available. Many variants of EC have been developed, including genetic algorithms [30], genetic programming [31], and multi-

objective optimization [32]. Most rely upon encoding candidate solutions (i.e., individuals) in a relevant data structure (e.g., vector, tree, bitstring, etc.), one or more fitness functions to assess the quality of each solution, and evolutionary operators to evolve candidates over time (e.g., mutation, crossover, selection). Crossover generates new individuals by combining genetic material (i.e., components of the data structure) from existing individuals, ideally resulting in better performing solutions. Mutation generates new individuals by randomly modifying attributes or components of an existing individual to attempt to improve diversity within the set of candidates. Selection determines which individuals are most fit to continue within the evolutionary process and may use different methods of picking candidates, from simply choosing the "best" performing individuals to ranking fitness objectives with Lexicase [33]. This process continues until either an acceptable fitness value is attained or the number of specified generations is met.

For the purposes of *DroneMR* we leave the exact specification of the evolutionary algorithm open as multiple types of exploratory search may be desired by the user. For example, if there is only a single objective then a genetic algorithm may best serve the search process, whereas if there are multiple competing objectives then a multi-objective algorithm such as NSGA-II [32] or many-objective selection operator such as Lexicase [33] may be more appropriate. In this paper we motivate *DroneMR* with a novelty search-driven approach, where a candidate solution's fitness is a combination of optimality and diversity and an archive of the most diverse solutions is maintained throughout the evolutionary process [34].

C. Metamorphic Relations and Testing

MRs can be used to generate MTs [35]. Unlike traditional tests, MTs overcome the oracle problem by comparing the system-under-test's behavior to other behavior rather than an already known expected result. For example, the length of the shortest path from two nodes (e.g., A and B) in a graph can be compared against the reverse (e.g., B and A). When two different sets of inputs provide the same result, the metamorphic relationship is an input-based relationship. However, if two different sets of inputs provide different results that are comparable (e.g., one is always less than the other) the relationship is an output-based relationship. Additional MTs can be generated by selecting different sets of comparable inputs.

Typically MRs are always valid for the system-under-test. However, when MRs are only applicable within specific contexts (e.g., specific modes) a contextual MR [10] can be employed where a logically defined context implies the MR only when the relation is contextually relevant.

The relationship between MRs and requirements can be illustrated with an example drone requirement as follows in `R1`:

"`R1`: The drone shall maintain a height of 1 meter from the ground."

A sample utility function can be then derived for `R1` using fuzzy logic membership functions:

$$util_{R1} = AS\_CLOSE\_AS\_POSSIBLE\_TO (dist_{ground}, 1m) \quad (1)$$

where $util_{R1}$ is normalized on $[0.0, 1.0]$ and a value of 1.0 would indicate that `R1` is completely satisfied (i.e., the drone is hovering at 1m exactly) and a value of 0.0 indicates a violation (i.e., the drone's height is measured outside of a pre-defined tolerance), and any value in between indicates a degree of satisficement. Assessing $util_{R1}$ at both design time and run time can provide critical feedback to the designers and system itself on the performance of the drone during its mission. Additionally, assuming a sensor exists on the drone that directly reports $dist_{ground}$, assessing $util_{R1}$ at run time would not induce any significant overhead (however, if sensor fusion activities are required to derive a utility function then a processing overhead may be incurred).

Next, an MR may be derived using `R1` and $util_{R1}$ based upon the drone's goal of maintaining a height of $1m$. The drone can be subjected to multiple operating contexts, where such contexts can include unexpected moving obstacles, varying wind speeds, and misconfigured rotor speeds. The drone's behavior can be monitored throughout executions to determine which behaviors are expressed as a result of each context. For instance, a drone released from a height of $5m$ might immediately fall to $0.25m$ before its rotors can exert an appropriate amount of thrust to maintain stable flight. Monitoring its altitude in this fashion would show that it did not immediately attain its objective and as a result $util_{R1}$ would be temporarily satisficed before settling into a steady state. Therefore, a contextual MR can be derived as follows in Equation 2:

$$(target_{1m} \leq currentHeight \leq start_{5m}) \rightarrow MR_{descent} \quad (2)$$

where $target_{1m}$ denotes the target altitude for the drone to maintain and $start_{5m}$ denotes the height at which the drone is initially released. $MR_{descent}$ then denotes the contextual MR that comprises the behavior of initially correcting altitude before its goal is satisfied. The $currentHeight$ remains with the range of the start and target heights, inclusively, while the $MR_{descent}$ is contextually applicable.

III. Approach

In this section we detail our framework for inferring MTs in uncertain environments and illustrate each step with a motivating example. Figure 2 presents a data flow diagram of our approach. We first describe expected inputs and outputs and then discuss each step in turn.

A. Inputs, Outputs, and Assumptions

DroneMR requires as input a drone system with known capabilities and sensors that can be simulated in a software-defined environment (i.e., simulator). Additionally, a set of

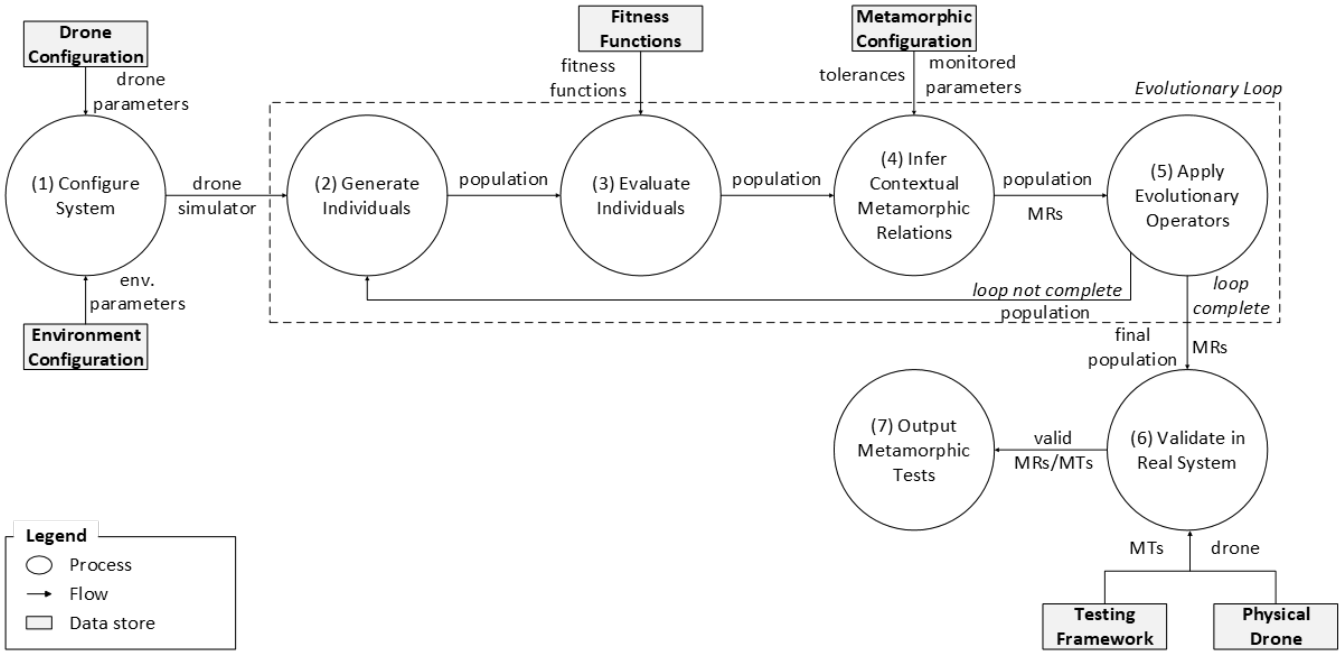

Fig. 2. Data flow diagram of metamorphic testing architecture.

requirements and utility functions derived from the operational specifications must also be provided to assess drone performance within the simulation and on the physical system. A configurable environment for the simulator must also be provided to evaluate the drone in multiple simulations. Finally, both the EC and metamorphic inference parameters must be provided as well to enable evolutionary search and MR derivation, respectively.

DroneMR will provide a set of MRs and associated MTs as output for use in enhancing the validation process. Additionally, *DroneMR* will output the set of clustered operating contexts that trigger its respective set of MRs to be used in further validation and can additionally be used as part of a self-adaptive feedback loop for self-reconfiguration at run time [36].

Lastly, we assume that the simulation environment reasonably reflects reality (i.e., the physics simulation mimics a real drone and that environmental scenarios can be recreated outside the simulation) and that the EC process will discover scenarios that may occur during normal operations.

B. Inferring Metamorphic Relations from Uncertain Environments

We next describe and motivate each step of the *DroneMR* process as shown in Figure 2.

(1) Configure System. First, all configurable parameters must be defined. All software-defined drone parameters must be set, including number of rotors, available sensors and precisions, and mission objective(s) to determine success or failure. The mission objective is derived from the algorithm governing the drone's behavior.

EC parameters, such as search type (single objective, multiple objective, novelty search, etc.), number of generations, population size, crossover rate, and mutation rate must be set for determining how the fitness landscape is to be explored. Simulation parameters for EC to use, such as obstacles for the drone to avoid, weather conditions to manage, and number of timesteps to evaluate must be configured as well. Lastly, the MR inference parameters must be set to determine how tolerant *DroneMR* will be when clustering inputs and expressed outputs.

Motivating Example. A single four-rotor micro-drone's mission objective is to maintain a height of $1m$ from the ground, starting from multiple heights (i.e., $0m$ and $5m$). The drone supports an infrared sensor underneath and an array of object detection sensors on its sides. As such, an autonomous flight algorithm that includes obstacle avoidance (i.e., moving in the opposite direction of a detected obstacle) and is configured to hover at the specified height of $1m$ is loaded into the drone's controller. Controller code is automatically activated upon the start of the simulation.

The EC process is configured to use novelty search for discovering as many diverse operating contexts and MRs/MTs as possible. Novelty search [34] is a variant of the genetic algorithm [30] that is typically single objective and maintains a novelty archive of diverse solutions, governed by a novelty metric that specifies how "different" solutions are from each other.

In this instance, a solution comprises the number of obstacles (i.e., rectangular objects, c.f., Figure 3) that can be instantiated during the simulation, including their respective sizes and the rate and pattern at which each obstacle moves.

Environmental scenarios are configured to include their chance of activation, intensity, and any specific parameters to that scenario. Here, wind can be configured in terms of its direction (i.e., angle to impart the wind force) and velocity throughout the duration of the simulation. An additional rate of change can be specified to vary each parameter as well.

Metamorphic input/output tolerances are loosely-defined to yield a smaller set of MRs for the designer to later verify. When inferring MRs in (4), *DroneMR* will collapse the environmental stimuli, drone configuration, and drone behavior into a set of numerical values that are then clustered. In this example, an 80% correlation rate would govern the number of solutions, within a specified distance from the cluster center, that belongs to a cluster expresses a correlated MR.

(2) Generate Individuals. Steps (2) through (5) serve as part of the evolutionary loop, running until either an acceptable set of solutions are discovered or the number of generations is met. In (2), a set of individuals is randomly generated that serve as potential solutions for the search problem. Here, the problem lies in creating a novel set of operating contexts (i.e., combinations of system and environmental parameters) that induce different behaviors within the drone system. The individuals will comprise a population of genomes that express a set of states for the simulation environment to execute.

If the evolutionary loop is not complete in (5) and the current number of individuals is less than the specified population size (due to selection), then a randomly generated set of individuals is created to fill out the population.

Motivating Example. Assuming that the evolutionary process was configured to run for 100 generations with 100 individuals per generation, *DroneMR* would randomly instantiate 100 individuals that comprise a set of states for the simulation environment: drone configuration, drone mission, initial environment settings, and changes to the environment over the duration of the simulation (e.g., moving items) as noted in (1).

(3) Evaluate Individuals. Each individual within the population is evaluated by parsing its genome and instantiating the simulation environment. The drone simulator then executes for a configured amount of time and its progress is recorded according to its specified mission objectives. All relevant behaviors and metrics are logged for post-processing.

Upon completion of the simulation, each individual is scored according to the fitness function(s) provided by the designer. Such functions are intended to provide a mathematical representation of how well the individual performed during evaluation and are used for later evolutionary operators.

For example, a relevant fitness function could combine the novelty of all generated MRs, the diversity of the environmental scenarios in which the drone operates, and the aggregated performance of the drone across all its key objectives to guide the search process.

Motivating Example. Assume a four-rotor drone as specified in (1) is instantiated with a mission objective of hovering at one meter for a period of one minute while maintaining positive movement along the x-axis. The environment is configured to randomly instantiate a number of boxes of varying widths/heights/depths for the drone to avoid. The boxes will follow a sinusoidal pattern on the z-axis to provide further issues for the drone. If the drone were to crash or be unable to maintain a one meter height, then the simulation is considered a failure and the resulting metrics are logged. Likewise, if the drone succeeds in its objectives then relevant output would be logged as well. Any values in between, such as attaining the height and then needing to correct its position due to transient problems (e.g., a box moving into the drone's position) would be considered *satisficement* and a normalized metric demonstrating how well the drone performed would be recorded. Such metrics would be tied to an input set of utility functions that mathematically quantify how well an objective is satisfied [8].

Upon completion, the individual's fitness score is calculated. A fitness score of 0.0 would be assigned if the drone crashed or violated any of its key objectives/invariants and a score of 1.0 would be assigned if the drone perfectly satisfied all key objectives. A normalized value in between $(0.0, 1.0)$ is assigned based on a calculation of the relationship between all fitness objectives.

For example, a novelty search-configured process will yield a suite of operating contexts that are as different as possible while still satisfying key objectives (i.e., a novelty archive), where a multi-objective approach would aim to balance a series of competing objectives and yield a pareto front of optimal solutions. In this context, the drone would be instantiated as specified by its starting criteria in (1) (i.e., at $0m$ and $5m$) and its height from both the simulator and drone sensor would be recorded throughout the duration of the simulation. During the simulation, any events that occur (e.g., wind, rain, etc.) are included in the log with a timestamp, tag (i.e., type of event), and any additional information relevant to the event (e.g., wind strength/direction, amount of rain, etc.). Upon simulation completion, the mission objective would be measured as to how long it stayed at its target height ($1m$) and if any critical violations occurred (e.g., crash, outside of range tolerance, etc.). Fitness would be calculated based on measured mission objectives as well as the measured diversity (if desired) of the operating contexts (e.g., Manhattan distance of all simulation parameters). Figure 3 demonstrates a simulated Crazyflie within the Webots simulation environment with a number of randomly-instantiated obstacles (boxes) that move vertically during the simulation.

(4) Infer Contextual Metamorphic Relations. Output logs from (3) are then evaluated to determine a set of behaviors that the drone expressed in all simulations and the events that led to the behaviors. Behaviors are clustered according to the MR tolerance specified in (1) and can be inferred based on steady-state or drastic changes in recorded values, as well as any user-defined tags that were provided (e.g., crash, trigger obstacle avoidance, etc.). For instance, a steady-state may show the drone hovering at a particular height for two seconds or a drastic change shows the drone's orientation flipping around an axis to denote uncontrolled flight. Given a set of output events, logs would then be scanned again

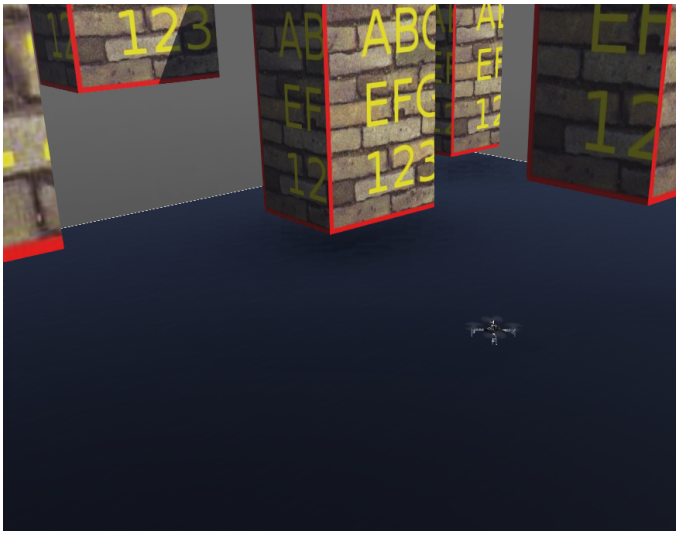

Fig. 3. Webots simulation of Crazyflie and randomly-instantiated obstacles.

to determine the cause of those behaviors and added to a related input set, where the tags provided in (2) can serve as a starting point for analysis (or to serve as a naive relation). This input/output set is considered to be the suite of potential MRs that require further validation. Given the current focus on automatic identification of MRs in multiple domains, a tool such as AutoMR [37] or GenMorph [38] could be used for a deeper analysis of input/output relations. The MRs can be considered contextual, given that they were inferred with respect to the operating context in which the drone was active (i.e., contextual MRs [10]). Moreover, the number and novelty of identified contextual MRs can be used as a fitness function to further aid the search process if desired.

Motivating Example. *DroneMR* will parse the logs of the novelty archive (i.e., optimal set of outputs as novelty search was the EC process) from (2) to infer a set of MRs. First, the path, orientation and actions of the drone are scanned to determine the events that occurred during the simulation. Note that, since recorded values typically use floating points a threshold must be specified to determine how similar events are to each other. For example, the drone may be hovering at $0.9m$ and $1.12m$ and still meet its mission criteria based on a threshold of maintaining height within $0.15m$ of the target. Moreover, the robot's z-axis orientation may not necessarily impact its goal as well, given that the altitude is the only metric considered for success. Another example is that the drone can be initially positioned at $5m$ and is released at the start of the simulation, immediately dropping below the $1m$ threshold before rectifying its position. A monitored behavior would be the increased thrust triggered for all motors before easing off at the desired altitude. For each output behavior, *DroneMR* will scan the relevant output logs up to the point of the behavior to determine which inputs/situations are relevant, where significant events were provided with tags during (2). For example, if the drone's steady state position is determined to fluctuate then a backwards analysis of logs may show that a northern-blowing wind was instantiated in advance of the output behavior.

An idealized log with accompanying utility values for $util_{R1}$ is as follows in Table I. For presentation purposes we truncate the log to applicable values.

Timestamp	Z (height, m)	$util_{R1}$
0:00	5.00	0.0
0:01	4.90	0.0
...	...	...
0:05	0.80	0.0
0:06	0.90	0.1
0:07	0.95	0.6
0:08	1.01	1.0
0:09	1.00	1.0
0:10	0.99	1.0

TABLE I
SAMPLE LOG WITH CALCULATED UTILITY VALUES FOR $util_{R1}$.

The values for $util_{R1}$ in Table I are defined in Equation 1 with a threshold of 0.1. Any height value outside the bounds of $[0.9m, 1.1m]$ is considered a violation and receives a utility value of 0.0, a height within $[0.99, 1.01]$ is considered fully satisfied, and any height in between scales within $(0.0, 1.0)$. While scanning logs, *DroneMR* will tolerate settling times to minimize false positives. As shown previously, $util_{R1}$ starts in violation as its $5m$ launch height is outside of the accepted value. However, within 8 seconds an ideal altitude is reached.

(5) Apply Evolutionary Operators. Next, crossover, mutation, and selection evolutionary operators are applied to the population as configured in (1). Crossover in this domain could select two individuals from the population and swap their genomes at a randomly-selected index, assuming the data structure is a list or vector (i.e., single-point crossover). This process would yield two new individuals for the population. Mutation then would select one gene within a chosen individual's genome and randomly mutate it to yield a new individual. For instance, a real-valued gene might be modified with a random number in a range of $[-\alpha, \alpha]$ or a Boolean gene may have its state flipped. Depending on the selection mechanism, individuals are selected for continuation within the evolutionary loop based on their calculated fitness score from (4), whereas all who are not selected are removed from consideration. In the case of novelty search [34] a novelty metric is also calculated for each individual to determine how "different" each are from other members of the population. The most diverse, according to a novelty threshold, are maintained within a novelty archive throughout the duration of the search.

Motivating Example. Following the configuration in (1) and assuming that crossover and mutation were specified to consider 40% and 30% of the population, respectively, a set of candidate parent genomes are randomly chosen for each process. Given there are 100 individuals within the population, 40 parent individuals are chosen to perform crossover (where pairs of individuals will generate two child individuals from their respective genomes) and 30 will perform mutation. Following, the selection operator will cull the population to 100 for those individuals who have been evaluated. For

this example we use tournament selection, where groups of k individuals are randomly selected and the winner (i.e., the individual with the highest fitness) is selected. Upon completion of the generational loop (i.e., 100 generations have been evaluated) the final population, along with the novelty archive, are provided as output. If the number of generations is not met then the process returns to Step (2).

(6) Validate in Real System. The automatically-inferred MRs from (4) are then formatted into a test report of inputs, expected outputs/behaviors, and scenario configurations for validation in a comparable drone system. This step both converts MRs to metamorphic test cases and aims to reduce the digital divide between a digital twin and a real system. A test engineer then must replicate the scenario as best as possible to validate the test case. Ideally a scenario will have multiple associated tests to minimize the time spent reconfiguring the testing area, however all generated scenarios should be feasible based on the input configuration in (1). Moreover, if possible test cases should be scanned and mapped to equivalency sets to minimize effort required by the test engineer. Relations marked as valid can then be added to a test specification for future design time validation or run time introspection activities.

Motivating Example. The inferred MRs focusing on reaching a stable altitude are translated into a test suite and validated in the physical drone. Specifically, the contextual MR in Equation 2 is translated into a set of test cases as shown in Table II. Given that there are a number of confounding factors that can serve as inputs/expected outputs to test cases, we truncate our test specification for presentation purposes. However, a test case such as $TC_{1.1}$ would additionally include all factors that served as inputs, such as individual rotor speed/direction, wind speed/direction, and altitude jitter when settling into its target height (i.e., $1m$). As such, we present the starting height and expected time to settle into the target range in Table II.

ID	Set	Inputs	Expected outputs	Threshold
$TC_{1.1}$	1	$Height_{init}$: $4.9m$	$Height_{stable}$ $3.0s$	$\pm 0.1s$
$TC_{1.2}$	1	$Height_{init}$: $5.1m$	$Height_{stable}$ $3.0s$	$\pm 0.1s$
...	...	...	...	...
$TC_{2.1}$	2	$Height_{init}$: $0.0m$	$Height_{stable}$ $1.0s$	$\pm 0.1s$
$TC_{2.2}$	2	$Height_{init}$: $0.1m$	$Height_{stable}$ $1.0s$	$\pm 0.1s$
...	...	...	...	...

TABLE II
SAMPLE TEST CASES FOR $MR_{descent}$.

Given real-world fluctuations in sensor readings and motor quality, a tolerance can be set for acceptance in each test case. In this example, a tolerance of 0.1 seconds is allowed for the test case to either pass or fail in its time to reach stable height. Additionally, the test cases shown in Table II are specified to be part of equivalency sets that result in the same settling time for stable flight given different starting altitudes. Their outputs are the same in that their governing requirement/utility functions (i.e., `R1` and $util_{R1}$) are eventually satisfied given the tolerance, however other inputs (e.g., wind speed) may impact the exact values required to satisfy the test case. Such an equivalency set may be collapsed into a smaller number of test cases at the discretion of the test engineer. The physical drone is then validated according to the test specification to ensure that the derived values are in agreement with those found in the software-defined simulation. If the results do not agree with the simulator's results then an additional review process is necessary to rectify inconsistencies between simulation and reality (i.e., the digital divide).

(7) Output Metamorphic Tests. The test specification from (6) is provided as output from *DroneMR*, where the specification comprises a set of metamorphic test cases (comprising input scenarios/configurations, output behaviors, and pass/fail criteria if known) and correlated operating contexts for which each MT is valid.

IV. DISCUSSION

This paper has presented *DroneMR*, our proof of concept framework for automatically inferring MTs in uncertain environments for safety-critical drones. *DroneMR* uses EC for discovering operating contexts in which a drone will express varying behaviors. These behaviors are then used as a basis for discovering MRs and refining them into MTs that can be used for validating expected behaviors at design time and run time. We motivated the use of *DroneMR* in the context of a drone's mission objective for maintaining a specific altitude.

Future directions for this project include experimental validation of *DroneMR*, instantiation of multiple drone types comprising heterogeneous sensors, and exploration of different search algorithms to discover a more diverse set of operating contexts.

ACKNOWLEDGMENT

We gratefully acknowledge support from the Michigan Space Grant Consortium (award number 80NSSC20M0124) and Grand Valley State University.

REFERENCES

[1] B. H. C. Cheng, P. Sawyer, N. Bencomo, and J. Whittle, "A goal-based modeling approach to develop requirements of an adaptive system with environmental uncertainty," in *Proc. of the 12th International Conference on Model Driven Engineering Languages and Systems*. Berlin, Heidelberg: Springer-Verlag, 2009, pp. 468–483.

[2] J. Cleland-Huang, N. Chawla, M. Cohen, M. N. Al Islam, U. Sinha, L. Spirkovska, Y. Ma, S. Purandare, and M. T. Chowdhury, "Towards real-time safety analysis of small unmanned aerial systems in the national airspace," in *AIAA AVIATION 2022 Forum*, 2022.

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

[4] S. Segura, D. Towey, Z. Q. Zhou, and T. Y. Chen, "Metamorphic testing: Testing the untestable," *IEEE Software*, vol. 37, no. 3, pp. 46–53, 2018.

[5] B. H. C. Cheng, R. Lemos, H. Giese, P. Inverardi, J. Magee, and et al., "Software engineering for self-adaptive systems: A research roadmap," in *Software engineering for self-adaptive systems*. Berlin, Heidelberg: Springer-Verlag, 2009, ch. Software Engineering for Self-Adaptive Systems: A Research Roadmap, pp. 1–26.

[6] P. McKinley, S. Sadjadi, E. Kasten, and B. H. C. Cheng, "Composing adaptive software," *Computer*, vol. 37, no. 7, pp. 56 – 64, July 2004.

[7] P. Sawyer, N. Bencomo, J. Whittle, E. Letier, and A. Finkelstein, "Requirements-aware systems: A research agenda for re for self-adaptive systems," in *Requirements Engineering Conference (RE), 2010 18th IEEE International*, 2010, pp. 95 –103.

[8] W. E. Walsh, G. Tesauro, J. O. Kephart, and R. Das, "Utility functions in autonomic systems," in *Proceedings of the First IEEE International Conference on Autonomic Computing*. IEEE Computer Society, 2004, pp. 70–77.

[9] G. Fraser and A. Arcuri, "Evosuite: automatic test suite generation for object-oriented software," in *Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering*, ser. ESEC/FSE '11. Szeged, Hungary: ACM, 2011, pp. 416–419.

[10] B. DeVries and E. M. Fredericks, "Triggering adaptation via contextual metamorphic relations," in *2024 IEEE International Conference on Software Quality, Reliability and Security (QRS)*. IEEE, 2024.

[11] J. H. Andrews, T. Menzies, and F. C. Li, "Genetic algorithms for randomized unit testing," *IEEE Trans. on Software Engineering*, vol. 37, no. 1, pp. 80–94, January 2011.

[12] M. Lajolo, L. Lavagno, and M. Rebaudengo, "Automatic test bench generation for simulation-based validation," in *Proceedings of the Eighth International Workshop on Hardware/Software Codesign*. San Diego, California, United States: ACM, 2000, pp. 136–140.

[13] E. M. Fredericks, "Automatically hardening a self-adaptive system against uncertainty," in *Proceedings of the 11th International Workshop on Software Engineering for Adaptive and Self-Managing Systems*, ser. SEAMS '16, 2016, pp. 16–27.

[14] C. Lu, H. Zhang, T. Yue, and S. Ali, "Search-based selection and prioritization of test scenarios for autonomous driving systems," in *International Symposium on Search Based Software Engineering*. Springer, 2021, pp. 41–55.

[15] H. Liu, X. Liu, and T. Y. Chen, "A new method for constructing metamorphic relations," in *2012 12th International Conference on Quality Software*, 2012, pp. 59–68.

[16] S. Mitra, T. Wongpiromsarn, and R. Murray, "Verifying cyber-physical interactions in safety-critical systems," *IEEE Security Privacy*, vol. 11, no. 4, pp. 28–37, 2013.

[17] M. Hassanalian and A. Abdelkefi, "Classifications, applications, and design challenges of drones: A review," *Progress in Aerospace Sciences*, vol. 91, pp. 99–131, 2017. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0376042116301348

[18] M. Gao, C. H. Hugenholtz, T. A. Fox, M. Kucharczyk, T. E. Barchyn, and P. R. Nesbit, "Weather constraints on global drone flyability," *Scientific reports*, vol. 11, no. 1, p. 12092, 2021.

[19] S. Park, H. T. Kim, S. Lee, H. Joo, and H. Kim, "Survey on anti-drone systems: Components, designs, and challenges," *IEEE access*, vol. 9, pp. 42 635–42 659, 2021.

[20] U. Ermağan, B. Yıldız, and F. S. Salman, "A learning based algorithm for drone routing," *Computers Operations Research*, vol. 137, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S030505482100263X

[21] Bitcraze. (2024) Crazyflie 2.1. Accessed July 17, 2024. [Online]. Available: https://store.bitcraze.io/products/crazyflie-2-1

[22] E. B. Putnam, D. N. Senarath, G. R. Urquijo, C. Tunc, and R. Bryce, "A lightweight drone simulator," in *2023 Tenth International Conference on Software Defined Systems (SDS)*. IEEE, 2023, pp. 52–59.

[23] O. Michel, "Webots: Professional mobile robot simulation," *Journal of Advanced Robotics Systems*, vol. 1, no. 1, pp. 39–42, 2004. [Online]. Available: http://www.ars-journal.com/International-Journal-of- Advanced-Robotic-Systems/Volume-1/39-42.pdf

[24] C. Llanes, Z. Kakish, K. Williams, and S. Coogan, "Crazysim: A software-in-the-loop simulator for the crazyflie nano quadrotor," in *2024 IEEE International Conference on Robotics and Automation (ICRA)*, 2024.

[25] Y. Jararweh, M. Al-Ayyoub, A. Darabseh, E. Benkhelifa, M. Vouk, and A. Rindos, "Sdiot: a software defined based internet of things framework," *Journal of Ambient Intelligence and Humanized Computing*, vol. 6, pp. 453–461, 2015.

[26] S. Tomovic, K. Yoshigoe, I. Maljevic, and I. Radusinovic, "Software-defined fog network architecture for iot," *Wireless Personal Communications*, vol. 92, pp. 181–196, 2017.

[27] Y. Jararweh, M. Al-Ayyoub, A. Darabseh, E. Benkhelifa, M. Vouk, and A. Rindos, "Software defined cloud: Survey, system and evaluation," *Future Generation Computer Systems*, vol. 58, pp. 56–74, 2016. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167739X15003283

[28] M. Ficco, C. Esposito, Y. Xiang, and F. Palmieri, "Pseudo-dynamic testing of realistic edge-fog cloud ecosystems," *IEEE Communications Magazine*, vol. 55, no. 11, pp. 98–104, 2017.

[29] F. Wedyan, R. Morrison, and O. S. Abuomar, "Integration and unit testing of software energy consumption," in *2023 Tenth International Conference on Software Defined Systems (SDS)*. IEEE, 2023, pp. 60–64.

[30] J. H. Holland, *Adaptation in Natural and Artificial Systems*. Cambridge, MA, USA: MIT Press, 1992.

[31] J. R. Koza, "Genetic programming: On the programming of computers by means of natural selection (complex adaptive systems)," *The MIT Press*, December 1992.

[32] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: Nsga-ii," *Evolutionary Computation, IEEE Transactions on*, vol. 6, no. 2, pp. 182 –197, apr 2002.

[33] L. Spector, "Assessment of problem modality by differential performance of Lexicase selection in genetic programming: A preliminary report," in *Proceedings of the 14th Annual Conference Companion on Genetic and Evolutionary Computation*. Philadelphia, Pennsylvania, USA: ACM, 2012, pp. 401–408.

[34] J. Lehman and K. O. Stanley, "Novelty search and the problem with objectives," in *Genetic programming theory and practice IX*. Springer, 2011, pp. 37–56.

[35] T. Chen, S. Cheung, and S. Yiu, "Metamorphic testing: a new approach for generating next test cases. technical report hkust-cs98-01," *Hong Kong Univ. of Science and Technology*, 1998.

[36] J. Kephart and D. Chess, "The vision of autonomic computing," *Computer*, vol. 36, no. 1, pp. 41 – 50, January 2003.

[37] B. Zhang, H. Zhang, J. Chen, D. Hao, and P. Moscato, "Automatic discovery and cleansing of numerical metamorphic relations," in *2019 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 2019, pp. 235–245.

[38] J. Ayerdi, V. Terragni, G. Jahangirova, A. Arrieta, and P. Tonella, "Genmorph: Automatically generating metamorphic relations via genetic programming," *IEEE Transactions on Software Engineering*, vol. 50, no. 7, pp. 1888–1900, 2024.