An Evaluation of Self-Adaptive Mechanisms for Misconfigurations in Small Uncrewed Aerial Systems

SALIL PURANDARE, Iowa State University, USA
MD NAFEE AL ISLAM, University of San Diego, USA
URJOSHI SINHA, Lawrence Berkeley National Laboratory, USA
JANE CLELAND-HUANG, University of Notre Dame, USA
MYRA B. COHEN, Iowa State University, USA

Small uncrewed aerial systems, sUAS, provide an invaluable resource for performing a variety of surveillance, search, and delivery tasks in remote or hostile terrains which may not be accessible by other means. Due to the critical role sUAS play in these situations, it is vital that they are well configured in order to ensure a safe and stable flight. However, it is not uncommon for mistakes to occur in configuration and calibration, leading to failures or incomplete missions. To address this problem, we propose a set of self-adaptive mechanisms and implement them into a self-adaptive framework, *CICADA*, for Controller Instability-preventing Configuration Aware Drone Adaptation. *CICADA* dynamically detects unstable drone behavior during flight and adapts to mitigate this threat. We have built a prototype of *CICADA* using a popular open source sUAS flight control software and experimented with a large number of different configurations in simulation. We then performed a case study with physical drones to determine if our framework will work in practice. Experimental results show that *CICADA*'s adaptations reduce controller instability and enable the sUAS to recover from up to 33.8% of poor configurations. In cases where we cannot complete the intended mission, invoking alternative adaptations may still help by allowing the vehicle to loiter or land safely in place, avoiding potentially catastrophic crashes. These safety-focused adaptations can mitigate unsafe behavior in 52.9% to 64.7% of dangerous configurations. We further show that rule-based approaches can be leveraged to automatically select an appropriate adaptation strategy based on the severity of instability encountered, with up to a 14.2% improvement over direct adaptation. Finally, we introduce a variation of our primary adaptation strategy designed to allow more cautious adaptation with limited configuration information, which gets within 6.7% of our primary adaptation strategy despite not requiring an optimal knowledge base.

CCS Concepts: • **Computer systems organization → Embedded and cyber-physical systems**; • **Software and its engineering → Error handling and recovery**.

Additional Key Words and Phrases: self-adaptive software, configurability, uncrewed aerial vehicles

1 INTRODUCTION

Small Uncrewed Aerial Systems (sUAS) are increasingly being deployed into unknown environments to support diverse missions – often in response to natural or man-made disasters such as floods, earthquakes, or fires [6, 55]. In such scenarios, the operators need to dispatch the sUAS as expeditiously and safely as possible to detect survivors, deliver food and water or medical items, or to provide critical intelligence to human rescuers. In order to

Authors' addresses: Salil Purandare, Iowa State University, Ames, Iowa, USA, 50011, salil@iastate.edu; Md Nafee Al Islam, University of San Diego, San Diego, California, USA, 92110, mislam@sandiego.edu; Urjoshi Sinha, Lawrence Berkeley National Laboratory, Berkeley, California, USA, 94720, usinha@lbl.gov; Jane Cleland-Huang, University of Notre Dame, Notre Dame, Indiana, USA, 46556, janehuang@nd.edu; Myra B. Cohen, Iowa State University, Ames, Iowa, USA, 50011, mcohen@iastate.edu.

ACM 1556-4703/2024/12-ART
https://doi.org/10.1145/3707643

be deployed safely and reliably, sUAS must be configured properly for their intended purpose. Misconfigurations can be caused unintentionally by inexperienced users or even maliciously by external bad-actors [53], either of which can impact the stability, safety, and success of the flight. To be effective in such environments, sUAS need the ability to detect the effects of unfavorable configurations as they emerge, evaluate the impact of those configurations on the stability of their operations, and ultimately adapt their behavior to mitigate the problem.

Almost all commercial sUAS flight controllers include basic failsafe mechanisms for detecting and responding to system failures such as low-battery, loss-of-signal, or geofence breaches. However, these failsafes are limited in their scope. Current sUAS are not inherently well-equipped to detect and mitigate the effects of configuration errors, which is especially problematic when they are to be deployed in emergency response scenarios.

A key to system dependability of an sUAS is the ability of its controller to prevent the vehicle from moving in unexpected ways. This can be partially achieved by tuning sets of parameters which work together to constrain the physics of the vehicle. Therefore, most sUAS controllers provide a wide range of configuration parameters that can be tuned and customized for different vehicles, flying conditions, or even individual missions (e.g. when speed is of the utmost importance vs. battery preservation). Initial configurations are typically set by sUAS manufacturers, but can be reconfigured by Remote Pilots in Command (RPICs) or sUAS technicians prior to flight, often as part of recalibrating the flight controller when prearming checks fail. It is also possible to modify configurations at runtime. The runtime exposure of parameters is meant to provide flexibility, but incorrectly modifying them can lead to errors that have been labeled as *input* or *range specification bugs* [30, 44]. These are parameter settings that may cause the vehicles to become unstable, leading to crashes, deviations from the flight path, or unresponsiveness.

Kim et al. have shown that slight changes to parameter settings at runtime can lead to critical flight failures [44]. They suggest restricting certain parameter settings to guard an RPIC from incorrectly changing parameters during flight. However, this would need to be vehicle and environment specific and is unlikely to have a general solution. A quick search of user forums shows that users are changing parameters, and that use of incorrect parameter settings is a common problem (e.g., [56]). Furthermore, there is little preventing a malicious actor from accessing and modifying control parameters and recent work [43, 72] has reported faults in controllers which lack checks on incorrect parameters.

While some research has explored the ability to *detect* or *predict* combinations of parameters, or to find root causes of faults [30, 32], they lack a proposed solution for overall sUAS dependability. One of the most popular software controllers, PX4 [51] has over 1,200 parameters, many with a wide range of potential values, all of which can be manipulated. The possible search space (much larger than $2^{1,200}$) for finding failing scenarios is simply infeasible to cover exhaustively, especially given that failure cases are vehicle and situation (e.g. mission or use case) dependent.

In this work we take a different approach to sUAS dependability. We investigate whether it is possible to automatically detect and adapt to and recover from failures, allowing critical missions to complete in the event of misconfiguration-related flight instability. We propose extending a common self-adaptive, MAPE-K (Monitor, Analysis, Planning, Execution and Knowledge) framework to improve overall system dependability [7, 37, 49, 52]. Many self-adaptive systems monitor and adapt based on quality attributes such as time and bandwidth, rather than discrete events (e.g. failures). However, prior research has also proposed self-adaptation for failure avoidance [24, 66]. This is the approach taken here.

Recently, Braberman et al. [7] presented a MAPE-K reference architecture for uncrewed aerial vehicles which considers different types of adaptations; those which change system configurations to adapt the vehicle's capabilities and those which adapt the behavior of the vehicle through flight commands. We propose adaptations of both types, achieving adaptation through modification of configuration parameters, and by sending flight commands. We further introduce a lightweight method of monitoring flight stability outside of standard controller error reporting with the aim to automatically detect emergent instabilities and to trigger adaptations.

We have built a framework called *CICADA*, which stands for *Controller Instability-preventing Configuration Aware Drone Adaptation*, with a prototype for experimentally validating our approach. *CICADA* subscribes to and monitors time-series data representing the vehicle's physical state (e.g. its roll, pitch, and yaw) and detects significant deviations from the expected norm. When deviations are detected, *CICADA* triggers an immediate adaptation.

We built a prototype of *CICADA* and conducted an initial evaluation of this work (presented in [58]). We began with a series of experiments on the widely used Gazebo flight simulator to explore part of the PX4 control parameter space in order to understand how parameter changes impact the sUAS during flight. We found a wide range of behaviors, including many failure-causing configurations. Utilizing this analysis, we then evaluated *CICADA*'s adaptation mechanisms. Our first adaptation approach returns to a predefined baseline configuration and attempts to continue the mission. When this fails we invoke adaptations that abandon the mission while attempting to increase stability. These include (a) loitering in place, and (b) landing in place. Our initial results are promising, showing that we can adapt to successfully complete up to 34.5% of otherwise failing missions, and stabilize to prevent dangerous behavior in 72.7% of cases that otherwise tend to result in severe behavior such as crashes or fly-aways.

This paper presents an extension of that original work [58] and introduces two new adaptations. First, we present a strategy in which we apply a rule-based metric to automatically select a suitable adaptation mechanism. Based on what we learned from our initial experiments, it uses heuristics to either (1) revert configurations towards known default values in an attempt to recover and resume the mission in cases of non-extreme instability, or (2) in more severe cases to sacrifice the mission and simply try to stabilize the drone. We further introduce an approach to configuration adaptation that does not require prior knowledge about configurations for any given drone model. It uses incremental *nudging*, iteratively pushing the configuration values in one direction or the other, using instability feedback to choose its next nudge. This is an important improvement over our initial work because ideal configuration information is rarely available for most real-world drone systems.

One limitation of the original work was that many of the tests were run only a single time in order to cover a broad swathe of configurations, and as such, the results were susceptible to flakiness in mission outcomes. In this work, we address this threat to validity by running multiple trials of all of our initial experiments. This is important to minimize the impact of potential non-determinism which can appear during flight due to timing issues. We capture the flaky behavior in our results to ensure that no data is lost, while still preventing the flakiness from producing misleading results. We further perform a study in which we re-run each flaky configuration in our dataset thirty times to compare the outcomes to the initial runs and discuss the causes of the flaky behavior in greater depth.

Finally, in this extension we present a case study with *CICADA* in the field using real drones. In this study we demonstrate a successful application of *CICADA* using its revert-to-baseline strategy. During takeoff the misconfigured sUAS begins to visibly wobble and is heading towards failure, yet *CICADA* automatically detects this and reconfigures, after which the vehicle visually stabilizes and proceeds to complete the intended mission.

The contributions of this work are as follows:

(1) We systematically explore a large parameter space of PX4, a popular flight control software to determine the impact of configurations on flight behavior.
(2) We present a configuration-aware self-adaptive framework, *CICADA*, for sUAS which is triggered by monitoring the vehicle's physical state during flight.
(3) We perform a series of experiments evaluating *CICADA*'s ability to avoid flight failures caused by configuration problems, and for mitigating the risk posed by dangerous configurations.
(4) We extend the initial adaptation protocols to introduce two smart adaptation strategies for *CICADA* and evaluate them with our prototype implementation.

(5) We conduct a case study on physical drone systems in which we demonstrate *CICADA*'s ability to detect configuration-related instability and successfully adapt to maintain stable flight behavior in the real world.

The rest of this paper is organized as follows. In the next section we present some motivation for *CICADA*. In Section 3 we present an overview of *CICADA* and then Sections 4 and 5 describe the experiments we conducted and report results. We then present our case study on a real drone (Section 6). We then discuss our findings in more detail (Section 7) and point to some interesting follow on investigations. We close by discussing related work (Section 8) and finally presenting our conclusions and future work in Section 9.

2 MOTIVATING EXAMPLES

Misconfigurations can be introduced in many different ways. First, hobbyists and technicians build and configure sUAS; however, this does not always work out as intended, and the resulting configurations can reduce flight stability and result in crashes [56]. Second, software bugs in sUAS applications can inadvertently cause inappropriate configurations. Third, the sUAS may be configured to fly in certain environmental conditions or with a specific payload; but may be launched under different conditions without appropriate reconfiguration. Finally, despite attempts to secure the communications infrastructure, a hacker might change the sUAS' configuration during flight [28].

We present three real-world examples of configurations errors here. First, we recently took delivery of new sUAS from a highly qualified manufacturer. During initial tests the sUAS experienced intermittent takeoff failures resulting in several crashes. This was not easy to debug, but eventually the root cause was attributed to a configuration error in one of the flight controller parameters. What we found interesting about this scenario is that the manufacturer was proficient in sUAS tuning, yet still delivered an sUAS with a problematic configuration. The fact that the problem was intermittent led us to look for other causes at first, and it was not until we had spent considerable time that we realized we had a subtle misconfiguration in our system.

In our second example, during preflight setup the operator accidentally swiped a screen in the RC transmitter, which modified the value for 'MOT_SPIN_ARM', causing the drone to have insufficient lift to maintain flight. Fortunately, we noticed the error and were able to reset it to the value established during pid tuning, thereby avoiding a potential flight-time problem. However, this incident shows how easy it is for operators to make accidental changes to parameters.

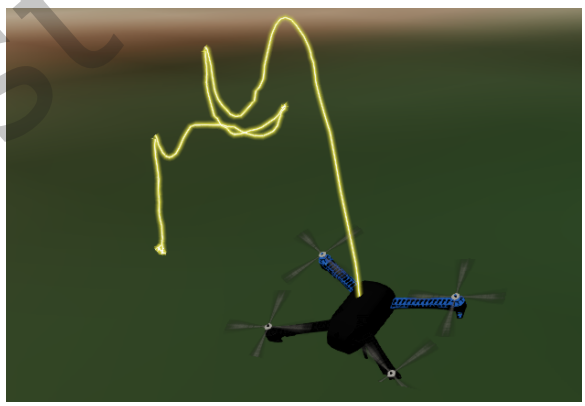


Fig. 1. Configuration-related multicopter crash reported by a PX4 user. This behavior was caused by the MC_ROLLRATE_P and MC_ROLL_P parameters.

For our third example we look to the PX4 user forum [56]. Users report many problems related to misconfigurations on this site, but we present one. In this instance they reported a crash of a Harrier D7 multicopter. The sUAS operator was attempting to modify the `MC_ROLLRATE_P` (controls output for angular speed error) and `MC_ROLL_P` (the desired angular speed) parameters of the controller. These parameters are ones that we manipulate in our experiments and case study. Figure 1 shows the uploaded flight path from this report. As can be seen, the flight took off and then flew in spiral like formation before crashing (shown left to right). In our tests with the same parameters, we found that there were values that caused similar unstable flight behavior for other drone models as well and often prevented take off. However, we also observed during some initial trials, that timely reconfiguration to known *good* parameter values allows the sUAS to regain stability and continue flying safely.

Another common scenario in which flight parameters can play an important role is during loss of signal. This is a relatively common problem that is typically caused by building obstructions, electromagnetic interference from high-power lines, interference from other devices, including those hosted on the sUAS itself, hot and/or humid weather conditions, or malicious attacks [33, 53, 70]. For example, a UK survey drone and a subsequent 'recovery' drone both experienced compass interference that were likely caused by deliberate signal jamming [28]. Once loss-of-signal is detected, the default response is to activate a RTH (Return to Home) failsafe mechanism. For example, in the PX4 firmware settings, the `COM_RCL_ACT_T` parameter defines the delay in seconds between loss of signal and failsafe activation. The optimal setting for this delay might differ when flying in remote rural areas versus urban ones. Similarly, fog or rain can reduce signal, potentially necessitating an increased tolerance for loss of signal duration. The parameter should therefore be configured according to geolocation and weather conditions and changing this parameter can have a large impact on mission success. sUAS are typically designed to detect loss-of-signal with radio-controllers; however, signal interference problems can also be detected directly by use of a Radio Spectrum sensor such as RadioHound [46]. Once detected, the sUAS can adapt accordingly – typically by flying higher to avoid interference and/or by switching radio frequencies or channels.

Last, hacking a drone is unfortunately all too simple as many sUAS use unencrypted communication channels. Whilst sUAS engineers can reduce the chance of hacking by deploying sUAS over VPNs, securing encryption channels, and limiting access to the sUAS to a single radio controller; it is still possible for a determined and skilled hacker to interfere with the correct operation of the sUAS by transmitting new waypoints or by changing the sUAS' configuration during flight. Monitoring both the underlying configuration of the sUAS and its mission plan to ensure that it remains within acceptable ranges provides the opportunity to detect problems, reconfigure parameters to safe ranges, and even to intercept and deny certain reconfiguration commands.

These examples of the ease of misconfigurations, and our initial tests to see if we could stabilize a vehicle when this happens, motivated us to develop a framework for instability-preventing configuration-aware adaptation. Understanding the configuration space, monitoring the sUAS' behavior during flight, and re-configuring when needed can potentially increase sUAS dependability. We present our vision for achieving this goal next.

3 THE *CICADA* FRAMEWORK

We now present an introduction to the *CICADA* framework. We begin with an overview and then present more detail on the adaptations.

3.1 *CICADA* Overview

CICADA is based on MAPE-K [39], and builds upon two existing frameworks. The Rainbow framework [11] is an architecture that provides runtime, self-adaptive capabilities for monitoring, detecting, decision-making, and enactment. Rainbow's control loop continuously monitors the properties of a system to identify problems. When a problem is detected, it performs an adaptation to bring the system back to a stable state. MORPH [7] is an adaptation architecture geared towards robotic systems like sUAS which differentiates between configuration

and behavioral adaptations. We follow the Rainbow architecture by partitioning the system into different layers and separating out the monitoring, analysis, and action components, and we leverage MORPH by incorporating its differentiation of configuration-based and behavioral adaptations.

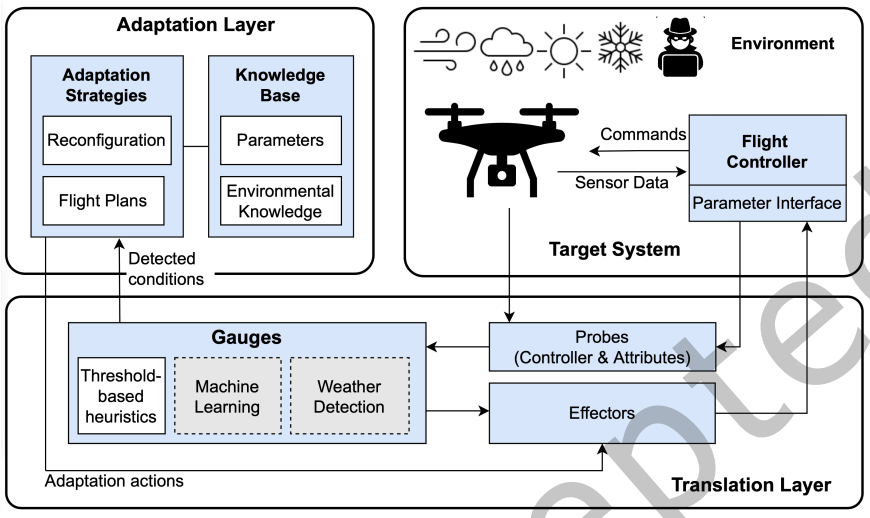


Fig. 2. *CICADA* Architecture: The flight controller in the target system communicates with the sUAS to send mission commands and modify configuration parameters. The sUAS is subject to environmental conditions. The translation layer contains probes, gauges and effectors which interact with the target system and the adaptation layer. Grey boxes indicate future work. The adaptation layer uses its internal knowledge base to select and plan reconfiguration strategies.

CICADA consists of three layers illustrated in Figure 2. These include the target system, which contains the physical sUAS and sensors, as well as their simulated variants. The translation layer monitors the target system and decides when to adapt, and then executes any necessary adaptations. It interfaces with both the target system (via probes, gauges and effectors) and the adaptation layer. Probes in the translation layer monitor raw data from the target system, gauges aggregate the data and trigger an adaptation, and effectors perform the reconfiguration. The adaptation layer analyzes and selects the reconfiguration strategy. In a MAPE-K system the K stands for knowledge about the system; for example, information about flight control parameters used to support the analyses and reconfiguration strategy. The adaptation layer then sends the reconfiguration strategy to the effectors (in the translation layer) which implement the adaptation.

At a more detailed level, *CICADA*'s *target system* (upper right) contains the flight controller (e.g., Ardupilot [5], PX4 [51] or Paparazzi [25]) and the sensors integrated into the flight controller or attached externally to the sUAS. The flight controller interfaces directly with the sUAS' sensors, and *CICADA* supports these integrated sensors as well as external ones. For example, *CICADA* can monitor data generated by the flight controller to detect unstable conditions or may deploy specialized environmental sensors or a camera to detect weather conditions [1]. Finally, the target system interacts with the environment, which may include externally applied changes to the controller parameters.

The translation layer consists of *probes*, *gauges* and *effectors*. Probes collect the realtime data from the controller, aggregate it, perform analysis, and detect emergent problems in the environment and/or onboard the sUAS. Gauges my be of different types. We envision ones based on computer vision, data analytics, and configuration

checks. Initially, we focus on the heuristics implemented in this paper, which monitor *controller instability thresholds*; however, more sophisticated approaches based on deep-learning are also feasible and will be included in future work [34]. Together, these elements make up the *monitor* and *analyze* components of the MAPE-K feedback loop. The last part of the translation layer are the effectors, part of the *execute* component, which interact with the adaptation layer and perform the adaptation in the target system.

Finally, our adaptation layer is where we reason about the adaptations, leveraging knowledge of the way different parameters impact stability. This is the *plan* component of our feedback loop. The knowledge base can be grown experimentally using the simulator to discover preferred adaptations as well as parameter changes to be avoided (guards), such as those described in the work of Swanson et al. [66].

In order for the sUAS to adapt to changes in flight behavior, it needs the ability to assess current conditions at runtime. *CICADA* accomplishes this with its gauges, using an onboard analytics component to analyze real-time sensor data from the probes to check for potential instabilities. In our first prototype of *CICADA* we use threshold violations which are relatively lightweight and fast to compute. The sensor data consists of information about the expected and actual roll, pitch, and yaw of the sUAS. Figure 2 has two additional gauges (greyed out) which we plan to build for future work. The first uses machine learning predictions to detect potential flight instabilities and the second uses weather detection as a trigger for adaptation.

CICADA also monitors configuration parameters to check that they are within acceptable bounds. This is particularly important if the sUAS is flying in a populated area in which malicious attacks are more likely to occur. As shown by Kim et al. [44], the use of parameter values outside safe ranges is often not constrained and is therefore open for attack. *CICADA* therefore first checks that all parameters are within-range prior to flight and reruns these checks during flight if instabilities are detected.

3.2 Adaptations

We initially designed three basic adaptations for *CICADA*. In this paper, we additionally present two new *smart adaptations* which build on the earlier strategies. Our adaptations can be sorted into two categories as defined by Braberman et al. [7]: *configuration-based* and *behavioral*. A configuration-based adaptation targets only the flight control parameters in the system, while a behavioral adaptation can make use of other mission commands, e.g. initiating landing. Our adaptations are defined below.

3.2.1 *Configuration-based Adaptations.*

Revert-to-baseline: This adaptation attempts to stabilize the sUAS so that it can complete its flight (albeit with some noise and potential drift). This adaptation requires knowledge of a baseline (or default) set of configuration parameters. This type of adaptation can be used when we have a stable start-up configuration (or known stable default set of parameters), and involves changing parameters during flight. We save the initial (stable) configuration and then re-set all changed parameters back to the known baseline values as established during earlier PID tuning. It is therefore particularly useful when parameters are accidentally changed by the operator via a user interface such as QGroundControl, or maliciously modified as a result of a security attack or breach. We call this *revert-to-baseline*.

Nudging: In this paper, we introduce a gradual approach which leverages the existing knowledge base to perform an incremental variation of the revert-to-baseline adaptation. We apply this approach when the ideal baseline is unknown, by gradually nudging the values in either direction. In our study, we focus on nudging towards default values. The advantage of this adaptation is that it requires little prior knowledge about the configurations of the sUAS being used, making it useful for a far greater variety of drone models.

3.2.2 *Behavioral Adaptations.*

Our next two adaptations are behavioral. For particularly problematic types of failures where harm to the drone, nearby property, or people in the vicinity is likely, attempting to continue the mission is not useful. These

behavioral adaptations allow the mission to abort, but attempt to modify the flight path to avoid a catastrophic consequence.

Loiter: The first of these strategies is to send the sUAS into *loiter* mode, which in the case of a rotor-copter, forces it to hover in place. The aim of this adaptation is two-fold – first, to give the drone operator time to regain manual control, and secondly to potentially reduce problems such as vibration or attitude fluctuations that may occur in more rigorous flight patterns.

Land-in-place: This strategy forces the sUAS to land at its current location. The goal when this adaptation is used is to quickly return a potentially out-of-control drone to the ground without causing serious damage. However, this strategy comes with additional challenges of preventing unnecessary landings on water or in a corn field.

3.2.3 ***Hybrid Adaptation****.*

Rule-based adaptation: We combine the earlier adaptations and our knowledge base with a rule-based strategy which uses data from the probes to determine the severity of instability and then automatically applies an appropriate adaptation in response. The goal for this adaptation is to ensure safety for a wider range of configurations, from recoverable minor misconfigurations to more severe, and potentially unsafe situations.

3.3 *CICADA* Instantiation

We built *CICADA* to work both in a simulation environment and on our physical drones. The simulation allows us to safely experiment with environmental and configuration factors; however, since *CICADA* is fully compatible with our sUAS hardware platform it can be used for deployment on physical sUAS. The PX4-Autopilot controller supports hardware deployments as well as software-in-the-loop. We demonstrate the use of *CICADA* on our physical drones in the case study (Section 6). Once we have gained a deeper understanding of dependable and safe reconfigurations we plan to build a more robust version of *CICADA* and perform additional experimentation, however, the cost and potential damage from failure in the physical environment has led us to focus mostly on simulation in this work. We now describe each part of *CICADA*.

3.3.1 Controller. The PX4-Autopilot software [51] is an open source flight control software compatible with many different flight control boards including Pixhawk 4, VOXL Flight, and ControlZero. It uses the MAVLink messaging protocol to send mission plans and control commands to the sUAS' hardware flight controller, and is also used by the sUAS to send status updates to the Ground Control System (GCS). Control software can be hosted onboard the sUAS (e.g., on an onboard Jetson) or offboard on a GCS.

3.3.2 Configuration Parameters. The PX4-Autopilot flight controller has over 70 categories of parameters, and around 1,200 configurable properties [21]. Furthermore, many parameters often have a large range of possible values, all of which can be individually configured. While a subset of the parameters are specific to different types of vehicles (e.g., fixed wing vs. copters), applicable to specific hardware devices (e.g., gimbal), relevant only in simulation environments, or can only be configured prior to activation of the sUAS, there are a large number that are common across all vehicles, relevant to both simulation versus hardware environments, and which can be manipulated statically (requiring a restart) as well as dynamically (taking effect immediately). In this work, we focus primarily on the dynamic configuration parameters which can be leveraged for runtime adaptation.

3.3.3 Probes. CICADA's initial monitoring component is plugged into the PX4 flight controller. PX4 uses an uORB messaging protocol which is an asynchronous publish-subscribe API, to publish various uORB topics associated with different sensor data. For instance, the uORB topic '*sensor_accel*' contains accelerometer data which gives the acceleration across x,y and z axes. While it is possible to monitor data from a wide range of sensors, we start with only a few that are relevant for evaluating aspects of flight stability. *CICADA* subscribes to *vehicle_attitude* and *vehicle_attitude_setpoint* and monitors and aggregates the actual and estimated *roll*, *pitch*

and *yaw* data in real time. *CICADA* can also monitor the configuration data using the controller's parameter settings functions.

3.3.4 Gauges. Gauges aggregate the data from probes and from the PX4 parameter interface. For this instantiation we use two types of gauges (see Section 3.1). The first aggregates acceleration and attitude data, including roll, pitch, and yaw values, and detects a variance from the expected and actual values. This is a simple mechanism to detect instabilities and will trigger an adaptation. The second collects current parameter settings. Adaptation is only triggered by the first gauge when it detects a deviation from the expected values. Instability information is shared with the adaptation layer.

3.3.5 Effectors. *CICADA*'s effectors apply decisions made in the adaptation layer by sending updated parameter commands to the flight controller in the target system. They also communicate with the adaptation layer to determine what changes to make. In *CICADA* we support the five adaptations mentioned, (1) **revert-to-baseline**, (2) **loiter**, (3) **land-in-place**, (4) **rule-based**, and (5) **nudging**.

4 EXPERIMENTAL EVALUATION

We have built a prototype of *CICADA* and perform an evaluation of this in simulation. Our study seeks to answer the following research questions:[1]

- **RQ1:** What is the impact of configurations on flight success?
- **RQ2:** How well does the *revert-to-baseline* adaptation recover from instability-causing configurations?
- **RQ3:** How effective are the *loiter* and *land* strategies at stabilizing problematic configurations?
- **RQ4:** Can our *rule-based* strategy improve the adaptation success rate?
- **RQ5:** How well does *nudging* work in the absence of an ideal knowledge base?

4.1 Configuration Space Model

We first selected a set of parameters based on the work of Kim et al. [44] since the focus of their work was also on flight instability. We then retrieved additional information from three sources; the PX4 discussion forums [61], formal PX4 documentation [21], and the PX4 bug repository [60]. In the online discussion forums, experts suggested ways to tune and optimize specific parameters to avoid poor calibrations that caused high vibration, insufficient thrust, and other negative outcomes on flight quality. We identified a set of 13 core parameters that appeared most frequently in discussions and in the relevant literature, which we refer to as our **core parameters** since they were specially selected for their relevance to controller stability. We then added 26 more parameters (labeled the **extended set**), for a total of 39 parameters providing a larger exploration space.

The top portion of Table 1 shows the set of 13 core parameters, while the bottom portion shows the extended set. We partitioned each parameter into five choices within its valid range. We selected the minimum (MIN), maximum (MAX), and default value (bold) for each parameter as specified in the official PX4 documentation. We then added (OP1), a value approximately midway between min and default (OP2), and a value approximately midway between max and default (OP3). For `MPC_THR_MAX` the default value is also its MAX so we modified the partition scheme to use values dispersed between MIN and MAX. During this process, we uncovered missing online documentation. The maximum values for three of the parameters, `MC_PITCHRATE_D`, `MC_PITCHRATE_I` and `MC_ROLLRATE_I` were not specified. Hence, we used the largest maximum value from the documented parameters and tried to include the maximum value of other similar parameters as well. For those parameters, we also experimented with significantly higher values, up to and beyond 1800, the highest value of any parameter in the set, and found the behavior was not noticeably different than with the maximum value.

[1]Supplemental data is at https://sites.google.com/iastate.edu/cicada/taas

Table 1. PX4 parameters depicting the minimum and maximum values and three additional parameters(OP) for each. Boldface denotes default values. The first 13 parameters represent the core parameters. The rest are the extended set. Values denoted by * have been excluded from 2-way data.

	Parameter	MIN	OP1	OP2	OP3	MAX
CORE	MC_PITCHRATE_P	0.01*	0.08	**0.15**	0.38	0.6
	MC_PITCH_P	0.0*	3.3	**6.5**	9.3	12.0
	MC_ROLLRATE_P	0.01*	0.08	**0.15**	0.33	0.5
	MC_ROLL_P	0.0*	3.3	**6.5**	9.3	12.0
	MC_PITCHRATE_D	0.0	0.0015	**0.003**	0.01	12.0*
	MC_PITCHRATE_I	0.0	0.1	**0.2**	0.6	12.0
	MC_PITCHRATE_K	0.01*	0.505	**1.0**	3.0	5.0
	MC_ROLLRATE_D	0.0	0.0015	**0.003**	0.0065	0.01
	MC_ROLLRATE_I	0.0	0.1	**0.2**	0.6	12.0
	MC_ROLLRATE_K	0.01*	0.505	**1.0**	3.0	5.0
	MPC_THR_MAX	0.0*	0.25*	0.5*	0.75*	**1.0**
	MC_PITCHRATE_MAX	0.0*	110.0	**220.0**	1010.0	1800.0
	MPC_THR_MIN	0.05*	0.085	**0.12**	0.56	1.0*

	Parameter	MIN	OP1	OP2	OP3	MAX
EXTENDED	MC_YAWRATE_P	0	0.1	**0.2**	0.4	0.6
	MC_YAWRATE_I	0	**0.1**	0.2	0.4	0.6
	MC_YAWRATE_D	**0**	0.1	0.2	0.4	0.6
	MC_YAWRATE_K	0	0.5	**1**	3	5
	COM_ARM_IMU_ACC	0.1	0.4	**0.7**	0.85	1
	COM_ARM_IMU_GYR	0.02	0.135	**0.25**	0.275	0.3
	MC_PITCHRATE_FF	**0**	0.0015	0.003	0.01	12*
	MC_ROLLRATE_FF	**0**	0.0015	0.003	0.0065	0.01
	MC_ROLLRATE_MAX	0*	110	**220**	1010	1800
	MC_YAWRATE_FF	**0**	0.1	0.2	0.4	0.6
	MC_YAW_P	0	1.4	**2.8**	3.9	5
	MIS_YAW_ERR	0	6	**12**	39	90
	MPC_TILTMAX_AIR	20	32.5	**45**	67	89
	MPC_XY_P	0	0.475	**0.95**	1.475	2
	MPC_Z_P	0*	0.5	**1**	1.25	1.5
	COM_POS_FS_EPH	0*	3	**5**	7	10
	EKF2_ABL_LIM	0*	0.2	**0.4**	0.6	0.8
	MOT_SLEW_MAX	**0**	1	2	3*	4*
	SENS_BOARD_ROT	**0**	10*	20*	30*	40*
	COM_VEL_FS_EVH	**1**	2	3	4	5
	MC_PR_INT_LIM	0	0.15	**0.3**	0.45	0.6
	MPC_ACC_HOR	2	2.5	**3**	9	15
	MPC_ACC_HOR_MAX	2	3.5	**5**	10	15
	MPC_XY_VEL_I_ACC	0	0.2	**0.4**	30.2	60
	SENS_BARO_QNH	500	756.65	**1013.25**	1256.65	1500
	MPC_Z_VEL_D_ACC	**0**	0.5	1	1.5	2

4.2 Sampling

The configuration set has 39 parameters, each with 5 values. This leads to 5^{39} possible configurations, which is too large to exhaustively explore. Therefore, we performed experiments with two different strategies for systematic exploration. The first approach was creating a one-hop sample. For each configuration, the algorithm uses default values for all but one parameter, and then systematically evaluates each of the four non-default values for that

parameter. This is repeated for all configuration parameters. This sample has 13×4 or 52 configurations for the core set and 26×4 or 104 configurations for the extended set (and 156 for the complete set).

While one-hop testing covers the entire range of values for every individual parameter, we also wanted explore interactions between parameters, using tests with two different parameters at a time set to non-default values. However, a two-hop algorithm created a sample that was too large. Therefore, we leveraged pairwise (or 2-way) combinatorial testing [15, 54]. This is one approach to explore a large configuration space systematically. 2-way combinatorial testing ensures that all pairs of parameter values are contained at least once in the sample used for testing, and attempts to minimize the number of configurations needed to satisfy this goal. We used a simulated annealing tool to create these samples [16].

We ran all pairwise samples and found that a large number of configurations failed. This is not unexpected because a pairwise sample ensures all single parameter values are combined with all other parameters, hence the failing parameters values will be heavily represented in the sample. We thus removed all parameter values that failed during one-hop testing from our pairwise sampling (these are starred in Table 1).

4.3 Metrics

We measure the maximum tilt observed during the flight as one metric of flight stability. We also retain the complete PX4 log files for each run, which allows us to analyze the flight in detail and view the exact flight path, and collect the number of instabilities and the outcome of the mission (success or failure).

Mission Success and Failure In order to establish a benchmark for flight success and failure, we built a test mission that would guide the sUAS through a realistic flight scenario. During the test mission, the sUAS is first commanded to arm, take off, and rise to an altitude of 10 meters. It then flies to a central waypoint 25 meters from the point of origin, and then moves to four other waypoints located five meters from the central waypoint in each of the cardinal directions. The sUAS must complete a path between these points to form a square shape before returning to the central waypoint and finally heading back to the point of origin to land. An example of a successful flight can be seen in Figure 3(a). The sUAS hovers briefly at each waypoint to approximate a more realistic real-world mission. For a run to be considered a success, the sUAS must reach all of these waypoints and return to the origin. If for any reason the sUAS fails to reach an expected waypoint within a predefined amount of time (which is set as the standard ROS timeout length of 15 seconds), the mission is classified as a failure.

4.4 Implementation Details

We implemented *CICADA* in Python on Ubuntu 20.04. The main *CICADA* program monitors the sensor and mission information over the course of the flight, determines when to adapt, and implements the required adaptation strategy. The probes were implemented as uORB plugins to the controller. To control the sUAS we use ROS-Noetic [65]. The Robot Operating System (ROS), is a widely used open-source suite for robotics applications. We used the PX4-Autopilot controller (version 1.12), and the Gazebo simulator [47]. We used the Iris multicopter airframe for all tests. The mission was defined in Python; flight commands and waypoints were transmitted to the sUAS via a MavROS interface. *CICADA* runs within a docker container. We used Docker Desktop version 4.4.2, on macOS BigSur 11.5.2. The container was allocated 2 CPUs and 2 GB of memory.

It took approximately 130 seconds to complete a mission, with an estimated 100 seconds of actual flight time. Around 30 additional seconds were required to launch the various components before the start of the mission.

4.5 Instabilities and Adaptation

To measure an adaptation-triggering instability, we used the previously described lightweight approach comparing the actual roll, pitch, and yaw of the vehicle against the expected roll, pitch, and yaw values as calculated by the flight control software. An instability message was triggered at any point that the difference between the

observed and expected values for any of these attributes exceeded a predefined threshold. The threshold was set to 10 degrees based on the authors' experience from observations of instabilities in real-world physical sUAS systems [14]. As soon as an instability was detected, the adaptation protocol was activated.

When the sUAS received the flight mission, it started with a set of configurations known to be safe for the Iris quadcopter, which we refer to as the set of *baseline* parameter values. This baseline set can be modified to accommodate other sUAS models to ensure that they are safe for any specific system being used. Immediately prior to flight, parameters were changed to the values decided for the current experiment. If an instability was detected, *CICADA* identified the parameter values that were currently set to non-baseline values in the flight control software. Action was then automatically taken based on the adaptation protocol was selected. In the case of the revert-to-baseline strategy, all parameters which were found to have been changed to potentially unsafe values were reset to their baseline values. For other adaptation strategies, the relevant protocol was put into effect immediately and the parameters were reset afterwards to ensure that the flight control software was in a safe state for the next run.

4.6 Threats to Validity

In order to maintain consistency across all our simulated experiments, we ran simulations using a single mission plan (the box), which means that we cannot guarantee generality. However, this test is complex and long enough that it will fail to complete when misconfigured. The experiments on all of the adaptations except for revert-to-baseline were also run only in a simulator, rather than on physical systems. However, based on hundreds of hours of simulation and physical flights in our prior sUAS work, we have observed that the simulator we used has high fidelity with respect to real-world flights, and furthermore, that the kinds of flight anomalies we observed matched real-world problems [3]. To reduce this threat we present a case study (see Section 6) on a real drone for the revert-to-baseline adaptation. We see that the same parameters which cause a misconfiguration in simulation, lead to instabilities in the real sUAS and we are able to adapt and restabilize the vehicle.

Only one sUAS model (the Iris quadcopter) was used in our simulations as this is the default model supported by the Gazebo simulator. It is likely that different models will require different parameter tuning, and as such, our results may not apply equally to other models. We note that our test harness allows the user to specify baseline parameter values for any sUAS, and is therefore compatible with other models as well. We have also introduced one new adaptation approach to help counter this limitation for different sUAS models, which is discussed in Section 5.5.

We explored only a limited set of parameter changes, so our experiments are not exhaustive in the parameter space. In order to obtain results that would be relevant for real-world users, we focused on parameters that literature and user reports on forums suggested would have a strong effect on flight stability. Finally, we note that misconfigurations are only one cause of instability in sUAS, and that not all flight problems can be attributed to or solved by configuration changes. We consider other causes of flight instability to be outside of the scope of this paper.

5 RESULTS

We now present the results of each research question in turn.

5.1 RQ1: Studying the Impact of Configurations

Kim et al. [44] used fuzzing to look for failing configurations; however, they did not focus on flight instability, did not systematically sample the valid configuration space, and were not able to identify combinations of parameters that are most likely to fail. This study provides a baseline for understanding interactions in the parameter space.

Furthermore, this is a first step in developing our knowledge base for adaptation. It can be leveraged later and integrated with learning.

Table 2. Comparison of results for 1-hop and 2-way sample sets for non-adaptive and revert-to-baseline adaptive tests. No. Trials is number of runs per sample.

Sample (set)	No. Trials	Failure Rate	Adaptive Failure Rate	Reduction in Failure Rate
1-hop (core)	5	23.1%	8.1%	15.0%
1-hop (extended)	5	11.3%	7.7%	3.6%
1-hop (total)	5	15.6%	9.5%	6.1%
2-way (core)	5	69.0%	35.2%	33.8%
2-way (complete)	5	93.2%	85.6%	7.6%

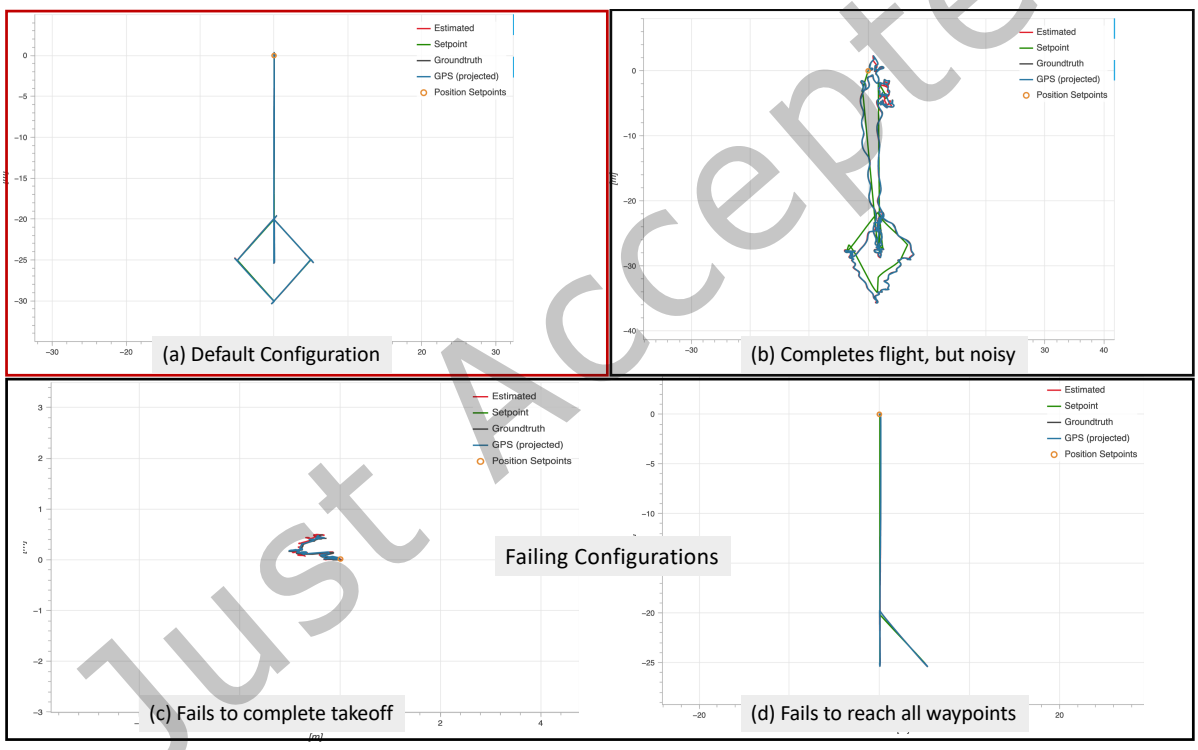


Fig. 3. Example flight paths under different configurations

Table 2 shows the samples and number of times we ran them (No. Trials). The third column displays the failure rates of each sample. This ranges from 11.3% in the 1-hop extended set to 93.2% in the 2-way covering array for the complete set. We will discuss columns 4 and 5 in the next research question. We observed that failing parameter values frequently lay at the extremes of the valid ranges. In many cases, the minimum value led to mission failure, which generally aligns with the findings of Kim et al. [44]. There were also several cases in

which the maximum value was the only one that failed. Figure 3 shows example flight paths for the (a) default configuration, a (b) noisy passing configuration, a (c) failing configuration that never takes off and (d) one that takes off but fails to reach all waypoints. We present a comparison of our findings with the findings of Kim et al. for the same set of parameters in Figure 4. They used a flight path deviation-based threshold using the integral

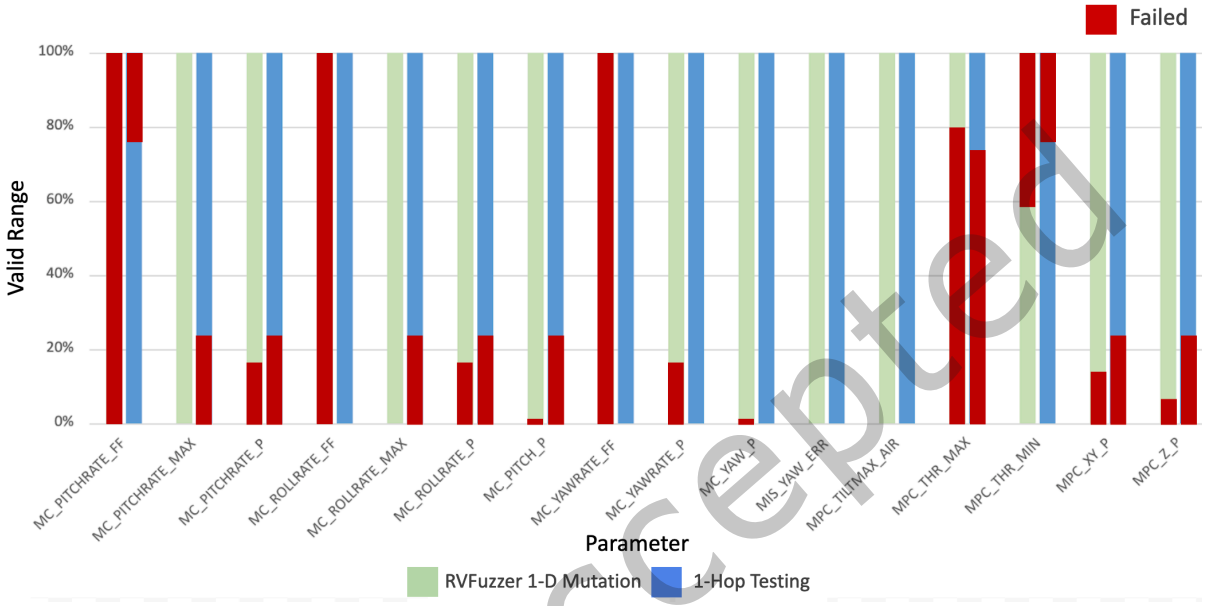
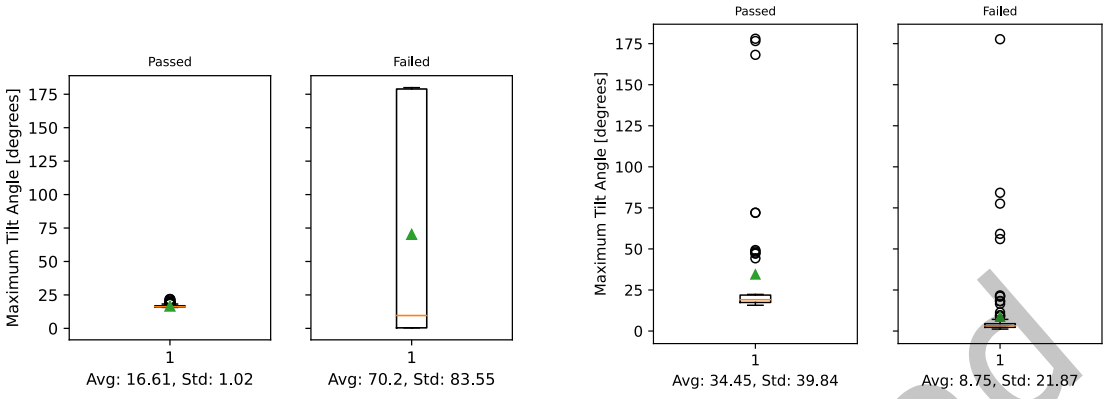


Fig. 4. Comparison of valid and invalid values with RVFuzzer 1-dimensional mutation. Failure-causing parameter values are marked in red.

absolute error formula for determining flight failure and success. We believe that this approach may be overly pessimistic, as we observed that in many cases, missions were able to complete successfully despite noisy flight paths. In addition, some of their parameters failed on all values, including defaults, e.g. `MC_PITCHRATE_FF` and `MC_YAWRATE_FF`, the feedforward pitch and yaw rates, which we did not observe. Wind and other environmental factors could also impact the accuracy of this metric.

We show boxplots of the maximum tilt angle for the one-hop and two-way core sets in Figure 5. We see in the one-hop set that failing cases had higher maximum tilt angles on average, as well as a much wider spread, despite the median actually being slightly lower than the passing cases. Our observations suggest that this high variance may be due to some failure-causing configurations inducing high tilt angle fluctuations if they actually fly, while others prevent the sUAS from taking off at all, resulting in very low tilt angles. In order to test for statistical significance, we first applied the D'Agostino-Pearson normality test [17] and determined that the tilt angle data did not follow a normal distribution. Based on this, we selected the nonparametric Mann-Whitney U test [50] to determine significance.

In the two-way sample, the maximum tilt angles were significantly lower in failing cases. The reason for this behavior is the same; with fewer successful takeoffs, there are less likely to be high tilt angles like there would be in flight. However, this trend is far more pronounced for the two-way set, likely because the interactions between multiple parameters in each flight lead to more immediate takeoff failures than in the one-hop set.



(a) One-hop core (b) Two-way core

Fig. 5. Maximum tilt angle for passing vs failing one-hop and two-way configurations.

We note that the failure rate is also higher for the pairwise sets. This is likely due to the greater number of configuration changes from the default. While 2-way combinatorial testing ensures that any given pair of parameter values is covered in at least one configuration in the sample, it does not mean that only that pair is being modified in that configuration. Rather, in most of the configurations, almost every parameter is changed from the default, in order to cover the sample space using a small number of configurations.

Summary of RQ1. We conclude that configuration parameters have a large impact on flight stability and highlight some configurations that induce failing behavior. When changing the values of frequently modified relevant parameters, we saw variation in the success of the mission, in flight paths, and in the maximum tilt angle during the flight.

5.2 RQ2: Adaptation using Revert-to-Baseline

Since an instability can occur at any time during a mission, once it is detected, it indicates that a problem has already occurred and the flight has been affected as a result. When a user attempts to take manual action upon receiving this information, it is often too slow to counteract the effect. An automatic adaptation mechanism that monitors and immediately responds to instabilities is preferable both because it can react more quickly to correct unstable flight behavior and also because it does not require the user's attention to be focused on the instability detector at all times.

Our first adaptation strategy, revert-to-baseline, relies on a knowledge base of parameter values considered to be safe based either on PX4 documentation or initial pid tuned configuration of that specific sUAS. We refer to this set of safe values as *baseline* values. Any parameters not currently set to their baseline value are updated to match the baseline value. The aim of this strategy is to restore stability to the flight and allow the sUAS to complete the current mission. We note that the baseline will be specific to a particular airframe, and environmental conditions. We show, for instance, the parameters which differ in the baseline for the simulated Iris compared with one of our physical hexcopters (HX18) during a real flight in Table 3. This baseline can be captured in our knowledge base or at arming.

Table 3. Comparison of non-matching baseline parameter values for the Iris quadcopter vs HX18 hexcopter along with their documented range. ? indicates an unspecified limit.

Parameter	Iris	HX18	Range
MC_PITCHRATE_D	0.003	0.0055	[0.0, ?]
MC_PITCHRATE_I	0.2	0.24	[0.0, ?]
MC_PITCHRATE_MAX	220	60	[0.0, 1800.0]
MC_PITCHRATE_P	0.15	0.2	[0.01, 0.6]
MC_PITCH_P	6.5	4.4	[0.0, 12]
MC_ROLLRATE_D	0.003	0.005	[0.0, 0.01]
MC_ROLLRATE_I	0.2	0.24	[0.0, ?]
MC_ROLLRATE_MAX	220	60	[0.0, 1800.0]
MC_ROLL_P	6.5	5.2	[0.0, 12]
MC_YAWRATE_MAX	200	45	[0.0, 1800.0]
MPC_ACC_HOR_MAX	10	5	[2.0, 15.0]
MPC_TILTMAX_AIR	45	20	[20.0, 89.0]

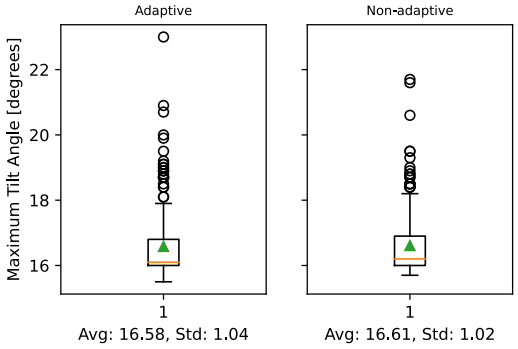
In order to determine the effectiveness of the revert-to-baseline adaptation strategy, we studied the flight behavior of missions with this adaptation mechanism enabled and disabled and compared the results for each of the sample sets. The results are summarized in the rightmost columns (4 and 5) of Table 2. The adaptive failure rate is the rate of failure when we use *CICADA* to detect and adapt using revert-to-baseline. The last column provides the reduction in failure rate from the non-adaptive variant of our experiments.

The revert-to-baseline adaptation strategy reduced the failure rate for all parameter sets we studied, improving by between 3.6% and 33.8% over the non-adaptive version (see Table 2). The greatest improvement in the one-hop tests was for the core parameters. The parameters in this set were selected because they were known to be some of the most impactful on flight stability. The extended set started with a lower failure rate and saw a smaller decrease in failure rate, which indicates that that set of parameters likely had a less meaningful effect on flight stability. The 2-way samples started with a much higher rate of failure. However, the improvement for these sets was also larger. The pairwise sample for the core set failed for 69% of configurations, but was able to more than double its success rate, reducing the failure rate of 33.8% with adaptation.

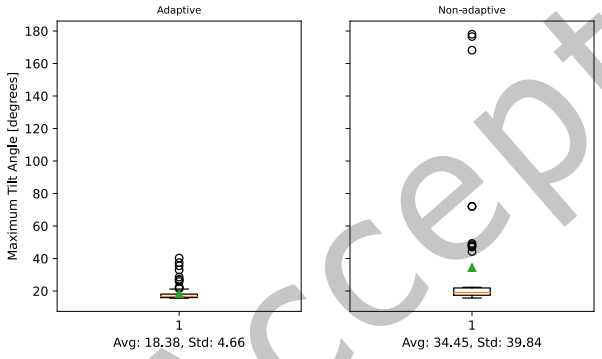
We also compared the maximum tilt angle from these trials with the revert-to-baseline adaptation to the nonadaptive case. We established in Section 5.1 that failing configurations have a higher maximum tilt angle on average, and that many more configurations failed without the revert-to-baseline adaptation protocol than when it was enabled. Therefore, in order to compare the maximum tilt angles for the adaptive and non-adaptive trials fairly, we only considered passing cases from both sets.

In our comparison, we found that there was very little difference in maximum tilt between the adaptive and nonadaptive flights for the one-hop set, as many of the passing one-hop configurations did not exhibit any behavior visibly different from the baseline. However, in our two-hop tests, we observed significantly higher maximum tilt in the nonadaptive trials. The average, median, and variance were all far greater than in the adaptive experiments, as visualized in Figure 6. This is likely due to the existence of configurations that didn't entirely prevent mission success, but still introduced a lot of instability during the flight, such as the configuration in path (b) of Figure 3. The revert-to-baseline strategy would have reacted to such configurations and modified them, thus reducing the number of high-tilt outliers. The difference between the one-hop and two-way behavior highlights the impact of interactions between multiple parameters. The 2-way sample for the complete set was not included in this analysis because it had too few passing configurations without adaptation to demonstrate spread. Our statistical tests of the core parameter sets are summarized in Table 4.

The pairwise sample had an initial failure rate of 93.2%, failing for nearly every configuration. However, the revert-to-baseline strategy still had a noticeable effect improving the success rate by over double compared to the non-adaptive case. Our observations suggest that one reason for the lower success rate compared to the one-hop



(a) One-hop core



(b) Two-way core

Fig. 6. Maximum tilt angle for adaptive vs nonadaptive one-hop and two-way configurations (passing only). Note that the Y-axis scale is not consistent across the one-hop and two-way plots.

Table 4. Overview of statistical tests of box plots in Figures 5 and 6. The p-value was calculated using the Mann-Whitney *U* test [50]. We consider results where $p < 0.05$ (in bold text) to be statistically significant.

	n	Average	Median	Std. Dev	p-value
1-hop (pass)	197	16.61	16.2	1.02	0.24
1-hop (fail)	60	70.2	9.6	83.55	
2-way (pass)	45	34.45	19.1	39.84	$\mathbf{3.59 \times 10^{-16}}$
2-way (fail)	99	8.75	3.2	21.87	
1-hop (adaptive)	238	16.58	16.1	1.04	0.68
1-hop (nonadaptive)	197	16.61	16.2	1.02	
2-way (adaptive)	93	18.38	16.7	4.66	$\mathbf{5.03 \times 10^{-6}}$
2-way (nonadaptive)	45	34.45	19.1	39.84	

tests for adaptation is the existence of hidden parameter interactions. In some cases, PX4 flight parameter values depend on the values of other parameters, making them more difficult to revert. For example, `MPC_THR_MIN`, which controls minimum thrust, has a hidden interaction with `MPC_THR_HOVER`, another thrust parameter, which is not in either of our test sets. If `MPC_THR_MIN`, which has a default value of 0.12, is set to 1.0, then `MPC_THR_HOVER`, which has a default value of 0.5, also automatically gets set to 1.0. However, if the adaptation protocol attempts to automatically set `MPC_THR_HOVER` back to 0.5, the controller throws a warning and the attempt to revert the parameter value fails, causing the parameters to stay the same, and eventually leading to mission failure. Some of this may be mitigated as we build up our knowledge base.

5.2.1 ***Flakiness in Mission Outcomes.***

During our testing, we observed that for certain configurations, the outcome of the mission was not consistent across all five trials. This is a byproduct of nondeterminism in the Gazebo simulator, which we used for all experiments. Notably, this problem is not unique to the simulation environment. For example, we previously experienced flakiness on physical drones, resulting in intermittent take-off failures. We invested significant time and effort to track down and remove the root-cause of the flakiness, which turned out to be a race condition associated with interactions between the flight-controller and our own onboard software [12]. Although Gazebo is a reliable, widely used simulator for sUAS, just like the flight control software running on physical drones, its implementation has subtle timing-based dependencies. More specifically, the physics, sensor generation, and navigation code are all run in separate threads or processes, and are therefore not synchronized [22, 23]. This means that no two flights are completely identical, even when there are no apparent differences without close analysis of the flight logs. These variations are too minor to have an effect in most cases. However, for certain problematic configurations, the parameter values can push the drone into a state that is very close to the boundary of passing and failing. These cases often exhibit high levels of instability. In such scenarios, Gazebo's nondeterminism has enough of an effect to swing the outcome one way or the other.

We performed five runs for each experiment and captured this flakiness in our results wherever it occurred. However, in order to understand if the flakiness demonstrated in our initial experiments was representative of the typical level of flakiness, we performed 30 additional re-runs for each flaky configuration. We present a comparison of these re-runs with the initial flaky experiments in Table 5. The first column describes the sample which the configuration was a member of, the second column is the configuration description (for the samples which have more than one change we use the number of the configuration in that sample. The third column provides the number (and ratio) of failing runs and the last column provides the number and ratio of failing runs for the re-run of 30 configurations. We also marked those configurations that exhibited a high instability rate with a star to see if this impacts our findings. In general we see similar rates of failures (with some variance) as the original 5 runs and the higher instability configurations do not seem to make a large difference. However, there are two configurations that were not flaky in this set of runs. One failed every time (configuration 30), and one did not fail at all (`MC_YAWRATE_P = 0.0`). The first case is less of a concern for validity since we always see a failure and in the five runs 4 out of 5 runs failed. For the one that did not fail in the reruns, we note it only failed once in the initial 5 runs, hence there could be some rare timing issue that led to a failure. This was one of the cases that exhibited very high instability even in passing cases, which makes this possibility more likely.

Summary of RQ2. We conclude that *CICADA*'s revert-to-baseline adaptation strategy is successful at recovering from many failures caused by misconfigurations using the flight instability trigger to adapt.

Table 5. Comparison of re-runs with initial experiments for each flaky configuration. The number in parenthesis indicates the ratio of failures. * indicates configurations that exhibited an abnormally high number of instabilities even in passing cases.

Sample	Config	Initial Failures	Rerun Failures
1hop core; adaptive	MC_PITCHRATE_D = 12.0	1/5 (.20)	9/30 (.30)
1-hop extended; nonadaptive	MOT_SLEW_MAX = 3	3/5 (.60)	6/30* (.20)
	MC_YAWRATE_P = 0.0	1/5 (.20)	0/30* (0.00)
2-way core; adaptive	config2	1/5 (.20)	1/30 (.03)
	config3	2/5 (.40)	6/30 (.20)
	config18	4/5 (.80)	26/30 (.87)
	config20	1/5 (.20)	1/30 (.03)
	config21	3/5 (.60)	28/30 (.93)
2-way complete; nonadaptive	config1	1/5 (.20)	8/30* (.27)
	config40	3/5 (.60)	19/30* (.63)
2-way complete; adaptive	config8	4/5 (.80)	18/30 (.60)
	config25	4/5 (.80)	29/30 (0.97)
	config27	4/5 (.80)	14/30 (.47)
	config30	4/5 (.80)	30/30 (1.0)
	config35	3/5 (.60)	15/30 (.50)

5.3 RQ3: Effectiveness of Different Adaptation Strategies

The revert-to-baseline adaptation strategy is effective in many cases in enabling an sUAS to continue and complete missions that would otherwise have been interrupted by configuration issues. However, there are also scenarios in which it isn't successful in counteracting these issues. Alternative adaptation strategies are required to prevent dangerous flight behavior. During our testing, we discovered a significant number of pairwise sample configurations that caused the sUAS to rapidly ascend to dangerous altitudes instead of following the mission path, which the revert-to-baseline strategy was unable to prevent. An example of this type of uncontrolled ascension is contrasted with a normal mission flight path in Figure 7.

Table 6. Takeoff and mission outcomes for each sample set. Values in parentheses refer to flaky configurations that caused failures in at least one of five trials, but not for all five. For example, 13 (2) indicates 13 failing configurations, with two of those 13 failures being flaky cases in which at least one of five trials failed.

Sample	Total configs	Failing Missions	Failing Takeoffs
1-hop non-adaptive (core)	52	12	8
1-hop adaptive (core)	52	5 (1)	5 (1)
1-hop non-adaptive (extended)	104	13 (2)	8 (2)
1-hop adaptive (extended)	104	8	6 (1)
2-way non-adaptive (core)	29	20	20
2-way adaptive (core)	29	13 (5)	13 (5)
2-way non-adaptive (complete)	50	48 (2)	44
2-way adaptive (complete)	50	44 (5)	38 (12)

Fig. 7. Comparison of expected mission flight path (left) with rapidly ascending flight behavior (right) caused by pairwise misconfigurations.

We study two adaptation strategies to address the limitations of the revert-to-baseline approach for such problematic configurations. The first of these is the *loiter* strategy, in which the sUAS reacts to an instability by hovering in place instead of continuing its previous flight path. The other is the *land-in-place* strategy, which commands the sUAS to land immediately at its current location. Importantly, unlike revert-to-baseline, neither of these strategies allow the sUAS to complete the current mission. Rather, the focus for this adaptation is on safety, as it's vital that the sUAS does not harm or damage people or objects in the vicinity if a configuration issue causes a flight to go astray.

To test these strategies, we began by identifying configurations that failed without adaptation which the revert-to-baseline approach was unable to recover from. We noted that many of the failures were cases where the sUAS was not able to take off successfully. This aligns with our real-world observations, where sUAS with extremely low thrust will just spin propellers, while sUAS with almost, but not quite, sufficient thrust to take off, tend to tip over and break propellers after approximately one minute. Therefore, if an sUAS fails to take off after 30 seconds of thrust, we can automatically kill the motors and disarm. This method can be applied to all cases for which takeoff fails entirely.

In order to test the loiter and land-in-place strategies on cases which weren't already covered by this protocol, we isolated configurations for which the takeoff succeeded and the sUAS reached the altitude of 10 meters specified by the mission. This also allowed us to evaluate the performance of each adaptation strategy on equal grounds (if the sUAS wasn't in the air at the time of adaptation, then the loiter strategy couldn't be fairly compared to the land-in-place strategy). We discarded any configurations for which the takeoff failed. Table 6 displays all takeoff outcomes for our dataset. We note that we obtain our shortlisted set of configurations for this RQ using nonadaptive/revert outcomes for each individual parameter as well as takeoff information, so the final number of configurations isn't the same as the total difference between the failing missions and failing takeoffs columns in Table 6.

We ultimately identified 17 configurations that fit our criteria. Of these, 14 were from the pairwise sample of the complete parameter set, and 3 were from one-hop testing with the `MPC_XY_P`, `MPC_Z_P`, and `EKF2_ABL_LIM` parameters, which represent rows J, K, and L respectively in Table 7, where the results are summarized. The configurations for which the uncontrolled ascent occurred are marked as *UnASC* in the table.

We found that both strategies were successful in preventing the uncontrolled vertical ascent in all cases where it previously occurred. However, not all of them were successes because there were instances where the flight behavior exhibited by the sUAS was different than expected. Namely, in several of the failures for the loiter strategy, the instability was detected before the sUAS took off, and the mitigation caused the sUAS to flip over on the ground instead of rising to takeoff altitude. We classified this as a failure because such situations often cause damage to the sUAS.

There were also some cases in which the hovering was not completely stationary and led to slight drift in the horizontal or vertical directions, but without leading to a crash or other risky behavior, or the sUAS failed to takeoff entirely. We do not consider these successes, but since the primary objective of the loiter adaptation is to prevent unsafe behavior, and that was achieved, they are simply listed with the descriptive labels "drift" or "no takeoff" in the table.

Table 7. Comparison of results for selected set of loiter and instant-land adaptive tests. UnASC are configurations with an uncontrolled ascent.

Config.	UnASC	Adaptation	Loiter	Land
A	✗	triggered	success	success
B		triggered	success	failure
C	✗	triggered	drift	success
D	✗	triggered	drift	success
E		triggered	success	success
F	✗	triggered	failure	success
G	✗	triggered	no takeoff	success
H	✗	triggered	failure	success
I	✗	triggered	failure	success
J		not triggered	failure	failure
K		not triggered	failure	failure
L		triggered	success	success
M	✗	triggered	failure	failure
N	✗	triggered	no takeoff	success
O	✗	triggered	failure	failure
P	✗	triggered	failure	failure
Q	✗	triggered	no takeoff	success
Fail Rate			47.1%	35.3%

The land-in-place strategy was effective in most cases, often detecting the configuration-related instabilities immediately when takeoff was attempted and taking action instantly to command a safe landing. We identified one failure of this approach in which the sUAS landed after an instability was detected, only to take off again and hover at a low altitude. We were unable to explain the cause of this behavior, but the authors have previously observed this behavior in physical sUAS as well. Configurations J and K failed for both strategies; however, these configurations were unique in that in both cases, no instability was ever reported, which meant the adaptation protocols were never put into place. We note that these cases were both from the one-hop experimental set, using the minimum values for the parameters `MPC_XY_P` and `MPC_Z_P`. We believe that these parameter values severely limited the attitude fluctuations of the sUAS, which both caused the mission to fail and prevented our gauges from detecting the error and triggering adaptation. However, ultimately, both strategies were broadly successful in countering the ascension issue and preventing the occurrence of risky flight behavior for this set of configurations.

Summary of RQ3. As we explored highly problematic configurations that the revert-to-baseline strategy was unable to address, we found that they could cause dangerous flight behavior. *CICADA* was able to prevent unsafe behavior in the majority of cases using the loiter and land-in-place strategies.

5.4 RQ4: Rule-based Adaptation

When an sUAS is deployed with one of the revert-to-baseline, loiter, or land-in-place strategies, our original adaptation module was limited to applying only the adaptation mechanism selected prior to the start of the mission. This can be a limitation if the user does not have enough knowledge about the mission or system parameters to predict the types of failures that are likely to occur. In order to address this problem, we propose a rule-based approach which attempts to automatically select an appropriate adaptation strategy based on the attitude information reported by the probes and gauges.

In situations where any adaptation strategy is likely to be successful, the revert-to-baseline strategy is generally preferred because it allows the mission to complete successfully instead of interrupting it to ensure safety like the loiter and land-in-place strategies. However, as discussed in Section 5.3, there are more severe cases which require the other adaptations. In our investigation into the different types of configuration-related failures, we observed a trend among configurations which caused more severe failures and required the use of the loiter or land-in-place strategies. Our instability detector collects data from multiple dimensions and uses an overall threshold to trigger an adaptation. We noticed that more problematic configurations were more likely to violate the attitude instability threshold along multiple dimensions of movement. For example, if an instability was detected and the gauges reported that both the pitch and yaw deviation thresholds were violated, the revert-to-baseline adaptation was more likely to be insufficient than if only the yaw deviation threshold was violated.

Based on these observations, we applied a simple rule-based adaptation strategy, which used the number of violated dimensions of movement to select an appropriate adaptation mechanism. We selected one *optimistic adaptation*, revert-to-baseline, and one *safer strategy*, land-in-place, to incorporate into our rule-based approach. We chose land-in-place over loiter due to its lower failure rate in the previously examined problematic configurations in Section 5.3.

Our adaptation rules are defined in Algorithm 1, where D is the set of dimensions of movement violated:

Algorithm 1 Rule-Based Adaptation

```
1: if |D| > 1 then
2:     revert and land-in-place
3: else
4:     revert-to-baseline
5: end if
```

When the rule-based protocol is in place, as soon as *CICADA* detects the occurrence of an instability, it checks how many dimensions of movement are violated. If only one dimension is violated, the revert-to-baseline adaptation is used, and if multiple dimensions are violated, a revert is performed, and the land-in-place strategy is applied immediately after.

To explore the effectiveness of the rule-based approach, we first selected all of the configurations from our sample sets that caused mission failures in RQ1 without adaptation. This gave us a total of 163 failing configurations, with 15 of these being flaky failures. We then tested the rule-based protocol on this "failure set". We performed one rule-based trial for each configuration. The rule-based approach yielded largely positive results, except for the 1-hop extended set, as summarized in Table 8. The rule-based approach provided the

greatest improvement over the revert strategy in the 2-way complete set. This may be due to the effectiveness of the land strategy in immediately stabilizing and returning the sUAS to the ground for more extreme, multiple movement dimension-violating configurations that are more likely to occur in this set and which the revert strategy is less likely to recover from. Rule-based adaptation was worse than the revert strategy for the 1-hop extended set. We believe this is due to multiple factors. First, the 1-hop extended set contains the fewest failing configurations of any set, so any failures will represent a larger percentage of the failure set. In this case, many of the configurations in the failure set for the extended parameter set were parameters like SENS_BOARD_ROT, which causes uncontrollable turning and flipping on the spot, which even landing in place cannot solve. The overhead for determining the rule may be enough lag in adaptation to lead to extra failures. Due to the simplicity of the revert adaptation, it can be deployed slightly more quickly and thus influence the drone earlier than the rule-based strategy, and with the complexity of the sUAS and the environment, even milliseconds can make a difference in adaptation success. Finally, we believe better heuristics are needed to determine when to land or revert, as the land in place strategy was not triggered very often in one-hop testing, even in cases when it would have likely have been the best option. We plan to continue exploring this direction of work.

Table 8. Comparison of failure rates for the revert-to-baseline, land-in-place, and rule-based adaptation strategies. These sample sets consist only of configurations that are known to fail without adaptation. Flaky configurations that failed at in least one trial out of five are included.

Sample (set)	Failure Rate	Revert Failure Rate	Rule-based Failure Rate	Rule-based Improvement
1-hop (core)	100%	41.7%	33.3%	8.4%
1-hop (extended)	100%	61.5%	69.2%	-7.7%
1-hop (total)	100%	52.0%	52.0%	0%
2-way (core)	100%	51.0%	50.0%	1%
2-way (complete)	97.1%	87.1%	72.9%	14.2%

Summary of RQ4. We conclude that a rule-based approach combining the revert-to-baseline and land-in-place strategies can be effective in improving the reliability of adaptation for sets of configurations that have a high starting failure rate while still enabling successful mission completions when possible for many configurations.

5.5 RQ5: Adaptation with Limited Knowledge Base

The standard revert-to-baseline approach assumes that a knowledge base is available which contains precise information about safe parameter values, based on either documentation or manufacturer tuning. However, in real-world applications, there may be situations in which this information is not available for the exact drone model being used. The PX4 documentation [21] contains parameter information for the Iris quadcopter airframe which we have used in all our simulated experiments up to this point. Although PX4 does support other airframes which come with some slightly modified configurations, the list of available airframes is quite limited and likely will not include the exact model required by the user.

We have already seen in Table 3 that optimal parameter values for different sUAS may vary slightly. In such cases, when parameter information isn't available for the model of drone being used, it may not be desirable for the user to fully trust the values provided by PX4 for a different model. For example, if the revert-to-baseline adaptation is triggered for an HX18 hexcopter and a parameter value is immediately set based on the Iris-centered

knowledge base, the new value may be suboptimal or even failure-inducing for the HX18. However, the occurrence of instability still signals that the current value is problematic and some form of adaptation is necessary.

To address this concern, we introduce a more cautious approach to the revert strategy, which leverages the existing knowledge base to gradually shift parameters towards safer values. This strategy makes use of the monotonic property of control instability described in [44], which states that parameter value changes in either direction will generally have a monotonic effect on the level of instability experienced by the sUAS during a flight. Starting from any current instability-causing parameter value, we aim to iteratively *nudge* the parameter towards a likely more stable value, based on the safe values in the knowledge base. When we don't have good data in the knowledge base we can provide the maximum and minimum parameter values from the documentation and experiment with partitioning and nudging in that way. With this approach, the sUAS is pushed more gradually towards a more stable configuration, with the parameter value being modified only to the extent necessary to mitigate the instability.

In our implementation, we selected five intermediate values distributed evenly between the current parameter value and the value stored in the knowledge base. Upon the first occurrence of instability, the parameter is changed from the current value to the first of the intermediate values. Once we have changed a parameter, due to the high reporting rate of the attitude data, it is not ideal to immediately nudge again when the next instability is reported, as the previous value will likely not have affected the stability of the sUAS by that point. Instead, we ignore several of the following instability reports and only adapt again when five more instabilities have been reported since the last parameter change. This gradual nudging prevents the parameter from immediately reaching the baseline value, which here is only being used as a general guide for the direction in which to adapt and not as an established safe value. It allows *CICADA* to cautiously shift the parameter value only as far as needed to restore stability.

We tested this approach on the failure set defined in Section 5.4. We found that the gradual reversion strategy was largely successful for many of the parameters in our dataset. The outcomes are summarized in Table 9. The delay in adaptation in comparison to the revert-to-baseline strategy prevented some configurations from being able to recover, however, we observed that many were still able to stabilize despite the delay, with the overall failure rate of 41.7% being within 7% of the revert-to-baseline failure rate (35.0%) for the core parameter set. For the extended set the revert-to-baseline failure rate was 61.5% and this increases to 76.9% (about 15% increase), however, it is still more than 10% lower than the failure rate without adaptation. In general, the nudging approach was successful for parameters with smaller failing value ranges, especially since the initial nudge was already enough to move the parameter out of the instability-causing range. It was less successful in cases in which a parameter had a relatively small stable range and the adaptation required a longer time to change from a failure-causing value at one extreme of the valid input range to a safe value at the other extreme of the range. For example, the `MC_PITCHRATE_FF` parameter has a baseline value equal to its minimum value in our parameter set. When nudging from a much higher failing value, the intermediate values reached during the gradual adaptation process were still much higher than the likely safe minimum value, so by the time the minimum was reached, it was too late for the drone to recover. This also supports our observations that timing plays an important role in successful adaptation. Since many more of the parameters in our parameter set had a large range of passing values, the nudging strategy was highly effective in stabilizing flights in our one-hop testing despite not requiring an exact knowledge base.

Summary of RQ5. We conclude that the gradual reversion approach, in which we gradually shift parameter values into a safer range, is successful, albiet less than the direct revert-to-baseline, in allowing sUAS to recover from misconfigurations in the absence of an ideal knowledge base.

Table 9. Comparison of outcomes for the standard revert-to-baseline strategy with a safer gradual reversion strategy using nudging in the absence of an ideal knowledge base. Only configurations that are known to fail without adaptation are included.

Parameter	Value	No Adaptation	Revert-to-Baseline	Nudging
MC_PITCH_P	0	failed	succeeded	succeeded
MPC_THR_MIN	1	failed	failed	failed
MPC_THR_MAX	0.25	failed	failed	failed
MPC_THR_MAX	0.5	failed	failed	failed
MPC_THR_MAX	0	failed	failed	failed
MC_PITCHRATE_D	12	failed	1/5 failed	failed
MC_PITCHRATE_P	0.01	failed	succeeded	succeeded
MC_PITCHRATE_K	0.01	failed	succeeded	succeeded
MC_PITCHRATE_MAX	0	failed	succeeded	succeeded
MC_ROLL_P	0	failed	succeeded	succeeded
MC_ROLLRATE_P	0.01	failed	succeeded	succeeded
MC_ROLLRATE_K	0.01	failed	succeeded	succeeded
Core failure rate		**100%**	**35.0%**	**41.7%**
MOT_SLEW_MAX	4	failed	succeeded	failed
MOT_SLEW_MAX	3	3/5 failed	succeeded	failed
MPC_XY_P	0	failed	failed	failed
SENS_BOARD_ROT	20	failed	failed	failed
SENS_BOARD_ROT	10	failed	failed	failed
SENS_BOARD_ROT	30	failed	failed	failed
SENS_BOARD_ROT	40	failed	failed	failed
EKF2_ABL_LIM	0	failed	failed	failed
COM_POS_FS_EPH	0	failed	failed	failed
MC_PITCHRATE_FF	12	failed	succeeded	succeeded
MPC_Z_P	0	failed	failed	failed
MC_ROLLRATE_MAX	0	failed	succeeded	succeeded
MC_YAWRATE_P	0	1/5 failed	succeeded	succeeded
Extended failure rate		**90.8%**	**61.5%**	**76.9%**

6 CASE STUDY: REAL WORLD APPLICATION OF REVERT-TO-BASELINE ADAPTATION STRATEGY

In Section 5.2, we demonstrated that *CICADA*'s revert-to-baseline strategy was capable of recovering from many failures caused by misconfigurations. Although a high-fidelity simulator was used for all experiments in that section, we acknowledge that there is still a gap between simulation and real-world usage. We also argued in Section 3.3 that *CICADA* is generalizable since PX4 supports both simulation and hardware in the loop environments. In an effort to bridge that gap, we present a case study on a real-world application of the revert-to-baseline strategy on physical sUAS hardware.

This study was performed using a hexcopter equipped with an mRo Control Zero H7 flight controller, using PX4-Autopilot version 1.13, and with probes implemented as MAVROS plugins. The rest of the architecture was similar to the simulated runs, but using a more recent version of the previously adopted DroneResponse platform

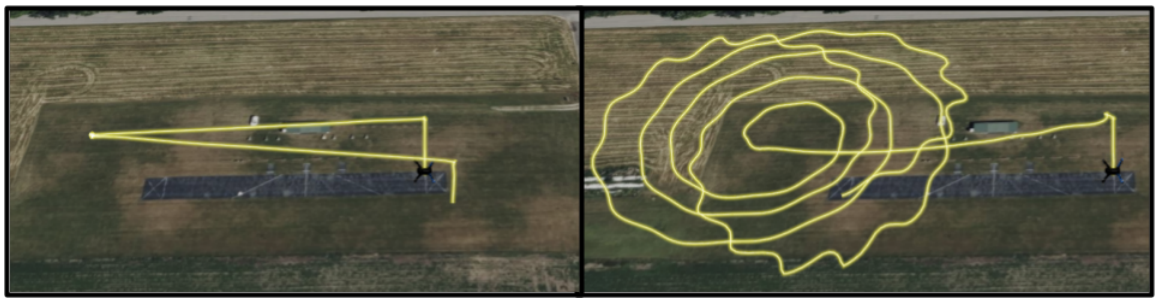


Fig. 8. Simulation results comparing the intended flight path for this mission with correct parameter tuning (left) versus the flight path observed in simulation without the benefit of adaptation when a faulty parameter value of MC_ROLL_P = 0.0 (right) was injected.

and architecture [35]. Flight parameters were tuned according to the manufacturer specifications for the drone, and *CICADA*'s knowledge base was updated accordingly.

The test mission used for these experiments consisted of a takeoff to an altitude of 20 meters, followed by brief hover leading into a slightly ascending straight line path for approximately 100 meters. At the end of the straight line path, which was the highest point of the mission, with an altitude of 33 meters, the drone reversed course and returned to a point close to the takeoff, mirroring the ascent with an equal descent, followed by a landing close to the takeoff location. This mission was designed to test the drone's ability to perform standard maneuvers, such as ascending, descending, hovering, and turning and changing direction.

For this study, we selected three failure-causing configurations from our parameter set to inject into the sUAS. Two of the parameters chosen, MC_ROLL_P and MC_PITCH_P, were from our core set, and the third, MC_ROLLRATE_MAX, was from the extended set. These parameters were specifically selected because our simulated experiments suggested that they would seriously impair flight ability and cause consistent failures, while still allowing the drone to take off, rather than causing low-to-the-ground flips or crashes, which many other parameters do. This was important because with no prior data available on injecting failing configurations into drones to test adaptation mechanisms, we needed to minimize the chances of an immediate drone-damaging crash near the ground. The chosen configurations made it possible to observe the impact of the parameters and adaptation in flight.

Prior to the field test, we performed a high fidelity simulation of our test mission with the selected configurations. To match the field test as closely as possible, we used the same DroneResponse architecture as deployed on the physical sUAS and used a more real-world accurate hexcopter airframe instead of the quadcopter airframe used in our empirical study. In these simulations, we observed that the faulty parameter values caused wildly diverging flight behavior, including off-line detours and extreme spiral paths instead of the expected straight flight lines. We show the results of the simulation when we set MC_ROLL_P to its default value (left) and a value of 0.0 (a known failing value from our experiments) in Figure 8. We verified the functionality of the revert-to-baseline adaptation in the high fidelity simulation environment before the field test.

In the field test, we first performed a baseline run with no parameters modified in order to verify the mission in the field and to have a point of comparison for the other runs. In the adaptation tests, we set a failure-inducing parameter value before takeoff and sent the sUAS on its mission. For all three faulty configurations, we observed a dangerous lateral swinging motion during the takeoff ascent which indicated that an imminent crash was likely. However, this deviation was detected by the probes and the revert-to-baseline mechanism was immediately engaged. As soon as the adaptation was triggered, the drone was able to recover from the unstable flight behavior

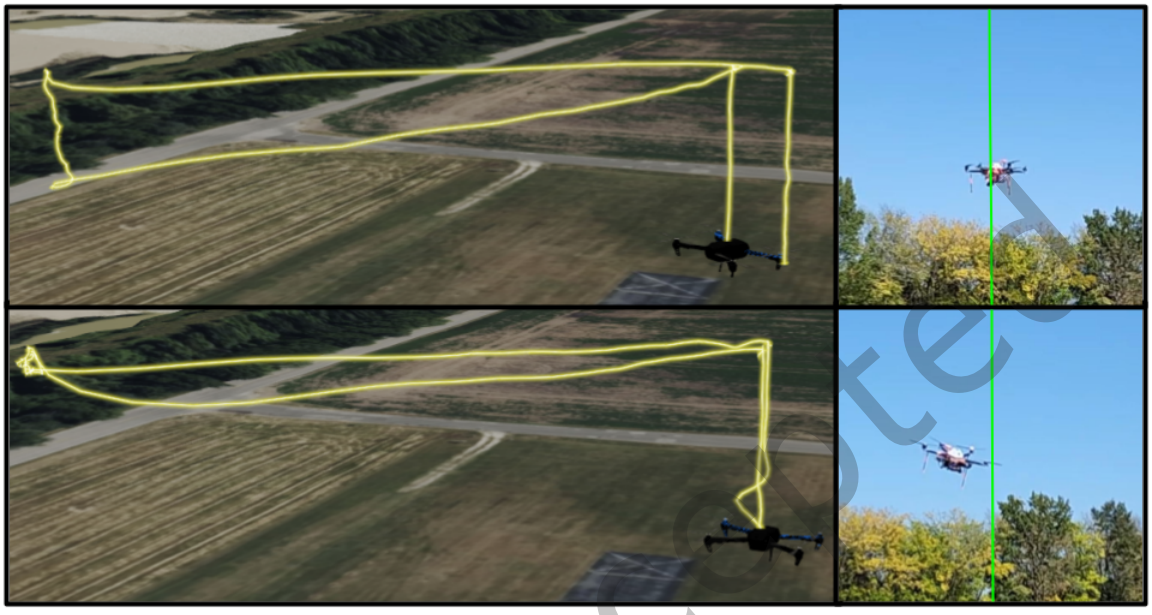


Fig. 9. A comparison of a run with with baseline parameters and a run with the MC_ROLL_P parameter set to a faulty value of 0.0 on real drone hardware. The instability caused by the configuration is apparent immediately during takeoff, but is detected by *CICADA*'s probes and the revert adaptation is triggered to restore stability. The moment of instability during takeoff in the second case is highlighted in the image on the bottom right.

and safely resume its mission. Figure 9 highlights the deviation and recovery of the failure-causing configuration in comparison to a baseline run. To strengthen our claim of adaptation, we repeated the same experiment for the misconfiguration of parameter MC_ROLL_P = 0.0, on three separate runs, on different days which meant the weather/environment was slightly different each time. We observed the same behavior for each run.

We show the baseline (good) flight on top and the misconfigured flight on the bottom. The figures on the left are the paths from the collected logs. The photos on the right are pictures of the actual sUAS in flight. A green line indicates the vertical takeoff expected for the flight. Both the log path and the photo shows the instability of the sUAS as it ascends. In the photo of the misconfigured sUAS we can see a large tilt angle and deviation from the green line. All flight logs from the field test, as well as video demonstrating the effect of the revert-to-baseline adaptation during the field test, have been made available on the supplemental site.

7 DISCUSSION

We now discuss some of the findings of this work and its implications. As observed in our experiments, sensing realtime data from the sUAS controller allows us to quickly detect and adapt to changing environmental conditions. We describe some of our takeaways here.

- **Additional gauges may improve adaptation.** Our implementation currently only takes attitude data from the sensors as input. However, not all misconfiguration-related issues will be directly reflected in the roll, pitch, and yaw information. By implementing more gauges into our framework, we may be able to detect other flight-threatening issues that the attitude sensors alone may not catch. These gauges may process different data streams, interface with other sensors, or make use of other components on board the vehicle, such as cameras. They may also leverage machine learning models for earlier or more accurate instability detection.
- **Scalability of rule-based adaptation** As we build on the rule-based adaptation approach, either by incorporating new gauges or more adaptation mechanisms, we will need to update our algorithm for choosing adaptations to deal with the increased complexity. Some research in this area have discussed applications of utility theory for selecting adaptations effectively [27, 71]. However, utility functions usually require continuous data, while our current gauges deal with discrete events (threshold violations). Therefore we follow a similar approach to Garvin et al. [24] and use a simplified rule-based strategy which only deals with discrete elements. In future work, we plan to incorporate more advanced learning-based gauges which may enable us to use a utility function with continuous data for effective decision making.
- **Fast detection is critical.** Our experiments with the the rule-based and nudging strategies, in which the delay in adaptation prevented successful stabilization for certain parameters, showed us that timing is a crucial component of effective adaptation, and faster instability detection may improve the adaptation success rate. One way to achieve this could be through lightweight detectors that require minimal sensor data to trigger adaptation.
- **Further parameter exploration is needed.** In this work we chose sets of parameters that we expected would make a difference in the vehicle stability during flight. However, we did not perform a systematic sampling of the entire PX4 parameter space. Exploring a larger set of parameters, as well as analyzing more complex interactions between parameters, would increase our understanding of the configuration space and allow users to be better informed about potential misconfigurations in their sUAS.
- **Implications.** The key takeaways from this study are that configurations play a large role in flight stability and success in sUAS, and that we can leverage flight control parameters to dynamically adapt to and recover from unstable flight behavior. Knowledge of the parameter space and immediate activation of the necessary adaptation protocols upon detection of flight instability is crucial for successful adaptation. Our case study demonstrates the potential of configuration-aware adaptation to help improve sUAS safety and reliability in the real world.

8 RELATED WORK

There is a large body of research on adaptive systems such as best practices, validation and challenges [10, 62, 63, 69] as well as research on using the MAPE-K loop [4, 11, 18, 45, 62, 66]. Several recent studies propose the use of MAPE-K in uncrewed aerial vehicles [7, 13, 42] and robot planning [36, 37, 59]. We discuss some closely related work to ours here.

Braberman et al. [7] proposed MORPH, a reference architecture for uncrewed aerial vehicles (UAVs). Like *CICADA* they use the Rainbow architecture and a MAPE-K loop, and split the architecture into (1) reconfigurations which change parameters to adapt the controller and (2) those that change the behavior via modification to the the mission plan or goal. However, while MORPH relies on the flight controller to report an error in order to trigger adaptation, *CICADA* directly monitors flight stability to independently detect issues during the flight,

since failures caused by misconfigurations are typically not automatically diagnosed and reported by the flight controller. Furthermore, their work describes an architecture, but they do not provide experimental results, while we have instantiated a prototype and evaluated *CICADA* for multiple use cases to determine feasibility.

While implementing the MAPE-K loop in self adaptive systems, Shmelkin [63] refers to interloop & intraloop communication as being the potential bottleneck for decentralized SASs. They argue the way to overcome this is to consider preprocessing for knowledge gain on the instance level to minimize the communication footprint. Jamshidi et al. [37] use machine learning to reduce a large configuration space, Elkhodary et al. [20] use feature modeling to reason about potential reconfigurations, and Swanson et al. [66] use both a feature model and aggregated data over time to learn about which re-configurations to choose and avoid. *CICADA* proposes to use knowledge from learning to create different types of gauges. In our initial implementation we only use limited knowledge of the configuration space, but we plan to implement more knowledge in future work similar to that of Swanson et al. Some adaptations proposed for uncrewed aerial vehicles involve changing the mission plan (e.g [7, 42]); however, our primary goal is to complete the mission via vehicle stabilization, and we only attempt other strategies when mission completion is known to be impossible with our primary adaptation approach. Several recent papers also propose learning to improve self-adaptation [2, 19, 37]. Our use of learning is for improving gauges.

In recent work Islam et al. [36] proposed a self-adaptive mechanism for anomaly detection on sUAS called ADAM (Adaptive Drone Anomaly Monitor). While runtime monitoring is important to detect the types of anomalies that trigger our adaptations in this work, it is resource intensive. Hence, ADAM adapts and selects a subset of detectors dynamically, based on its current environment. This work is orthogonal to ours. While we use a similar approach to ADAM for monitoring (detecting our anomalies), we use a single detector. Future work could incorporate an ADAM-like adaptive mechanism for monitoring as well as for adapting to fix misconfigurations.

The general problem of finding faults or identifying poor performance due to misconfigurations is well studied in traditional software. We point the reader to a representative sample of references on this topic [8, 26, 64, 73, 74]; however, none of these focus on the sUAS environment or self-adaptation. There has been some research on exploring misconfigurations on sUAS and other robotic systems (e.g. [32, 38, 43, 48, 67, 72]), or studying instability due to malicious threats [44]. We discuss a few of these robotic threads of work in more detail.

Kim et al. presented RVFuzzer [44] which looks for range implementation and specification bugs in sUAS. Their assumption is that the range of misconfiguration behavior is monotonically changing and the full range of passing behavior can be determined. RVFuzzer uses a binary search to refine valid parameter value ranges beyond those defined in documentation. However, it only focuses on finding parameter values attackers might use and does not perform adaptation. We have compared our exploration and have expanded on the configuration space over RVFuzzer. We have also observed in some recent work [57] that the monotonic property may not always hold.

Han et al. introduced LGDFuzzer [31] and ICSEARCHER [29], which use genetic algorithms to search for problematic configurations in sUAS. In that work they use machine learning to predict instabilities via multi-objective optimization. They do not, however, systematically explore the parameter space, and they do not consider adaptation to fix the instabilities observed. Chang et al. [9] also use a genetic algorithm-based approach called APFuzzer to search for configurations that cause incorrect states.

Taylor et al. examined faults due to faulty configurations in sUAS for human-robotic collaborative systems [67] and Jung et al. [38] presented SwarmBug which detects bugs in swarms of sUAS due to the configurations of the swarm algorithms. Neither of these focus on the controller parameters or threads perform adaptation.

Other research has looked more broadly at misconfigurations and testing of robotics systems (e.g. see [40, 41, 68] as examples). For instance, Khatiri et al. presented SURREALIST [40] and AERIALIST [41], a simulation-based test case generation tools for sUAS. These toos are able to replicate a given fight log in simulator and generate test cases (mutations of the environment). They do not focus on configurations of the fight controller. Last,

Timperley et al. [68] proposed a static analysis for finding architectural misconfigurations in general robotic systems, however, these are not controller parameters and they do not provide adaptation.

9 CONCLUSIONS AND FUTURE WORK

In this paper we presented *CICADA*, a framework for self-adaptation in small uncrewed aerial vehicles which aims to prevent flight instability caused by misconfigurations. We explored the configuration space of a widely used flight control software and found that controller parameters had a large impact on flight stability. We introduced a primary adaptation strategy to overcome misconfigurations mid-flight to complete the mission, and demonstrated the effectiveness of this strategy in recovering from configuration problems caused by both individually problematic parameter values and interactions between pairs of parameters. We further proposed two other safety-focused adaptation mechanisms to prevent exceptionally dangerous flight behavior from occurring in situations where the primary strategy was not viable. Our experiments showed that these strategies were effective in preventing dangerous flight behavior for many of these especially unsafe misconfigurations. Depending on the parameter sample sets, we observed between 3.6% and 33.8% improvement in failure rates with our primary strategy, while the safety-focused mechanisms allowed us to mitigate unsafe behavior for 52.9% to 64.7% of dangerous configurations. We then combined the other strategies to create a rule-based adaptation approach using data from the probes and gauges to automatically select an appropriate adaptation for the severity of instability detected, which resulted in up to a 14.2% improvement over the revert strategy. In addition, we introduced an iterative nudging configuration approach which can be used even with limited knowledge about the optimal configurations for a drone system. Overall nudging improved the failure rate over the non-adaptive approach, but lost some effectiveness seen in the revert-to-baseline adaptation, which was optimized for the airframe we were testing on. We saw as little as a 6.7% decrease in effectiveness for some parameters, but as high as a 15.4% decrease for the extended configuration set. While the reduction was noticeable, we did still improve greatly over the non-adaptive approach despite the lack of optimal knowledge base. Finally, we performed a case study on physical sUAS hardware to demonstrate the effectiveness of our primary adaptation strategy in the real world.

As future work we plan to devise more intelligent adaptation selection strategies to automatically select from different adaptation mechanisms that may be needed over the course of a flight. We will also expand our experiments to incorporate improved (ML based) gauges, perhaps incorporating some of the ADAM detectors into *CICADA*. We plan to experiment with larger and different parameter spaces, and run experiments under more diverse flight missions and environmental conditions (e.g. wind). We also plan to explore ways to handle flakiness both in simulation and real-world environments.

ACKNOWLEDGMENTS

The work in this paper was primarily funded under the USA National Aeronautics and Space Administration (NASA) Grant Number: 80NSSC21M0185 and the National Science Foundation (NSF) CNS-1931962 and CCF-1909688.

REFERENCES

[1] Sophia J. Abraham, Zachariah Carmichael, Sreya Banerjee, Rosaura G. VidalMata, Ankit Agrawal, Md Nafee Al Islam, Walter J. Scheirer, and Jane Cleland-Huang. 2021. Adaptive Autonomy in Human-on-the-Loop Vision-Based Robotics Systems. In *1st IEEE/ACM Workshop on AI Engineering - Software Engineering for AI, WAIN@ICSE 2021, Madrid, Spain, May 30-31, 2021*. IEEE, 113–120. https://doi.org/10.1109/WAIN52551.2021.00025

[2] Frank José Affonso, Gustavo Leite, Rafael AP Oliveira, and Elisa Yumi Nakagawa. 2015. A Framework Based on Learning Techniques for Decision-making in Self-adaptive Software. In *International Conference on Software Engineering and Knowledge Engineering*, Vol. 15. 1–6. https://doi.org/10.18293/SEKE2015

[3] Md Nafee Al Islam, Muhammed Tawfiq Chowdhury, Pedro Alarcon Granadeno, Jane Cleland-Huang , and Lilly Spirkovska. 2023. Towards an Annotated All-Weather Dataset of Flight Logs for Small Uncrewed Aerial Systems. In *AIAA AVIATION 2023 Forum.*

https://doi.org/10.2514/6.2023-3856

[4] Paolo Arcaini, Elvinia Riccobene, and Patrizia Scandurra. 2015. Modeling and Analyzing MAPE-K Feedback Loops for Self-Adaptation. In *2015 IEEE/ACM 10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*. 13–23. https://doi.org/10.1109/SEAMS.2015.10

[5] Ardupilot. Last Accessed 01/29/22. ArduPilot Open Source Autopilot. https://ardupilot.org/

[6] Bilel Benjdira, Taha Khursheed, Anis Koubaa, Adel Ammar, and Kais Ouni. 2019. Car Detection using Unmanned Aerial Vehicles: Comparison between Faster R-CNN and YOLOv3. In *2019 1st International Conference on Unmanned Vehicle Systems-Oman (UVS)*. 1–6. https://doi.org/10.1109/UVS.2019.8658300

[7] Victor Braberman, Nicolas D'Ippolito, Jeff Kramer, Daniel Sykes, and Sebastian Uchitel. 2015. MORPH: A reference architecture for configuration and behaviour self-adaptation. In *Proceedings of the 1st International Workshop on Control Theory for Software Engineering*. 9–16. https://doi.org/10.1145/2804337.2804339

[8] Mikaela Cashman, Myra B. Cohen, Priya Ranjan, and Robert W. Cottingham. 2018. Navigating the Maze: The Impact of Configurability in Bioinformatics Software. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering* (Montpellier, France) *(ASE '18)*. Association for Computing Machinery, New York, NY, USA, 757–767. https://doi.org/10.1145/3238147.3240466

[9] Zhiwei Chang, Hanfeng Zhang, Yue Yang, Yan Jia, Sihan Xu, Tong Li, and Zheli Liu. 2024. Fuzzing Drone Control System Configurations Based on Quality-Diversity Enhanced Genetic Algorithm. In *Artificial Intelligence Security and Privacy*, Jaideep Vaidya, Moncef Gabbouj, and Jin Li (Eds.). Springer Nature Singapore, Singapore, 499–512. https://doi.org/10.1007/978-981-99-9785-5_35

[10] Betty H. C. Cheng, Rogério de Lemos, Holger Giese, Paola Inverardi, Jeff Magee, Jesper Andersson, Basil Becker, Nelly Bencomo, Yuriy Brun, Bojan Cukic, Giovanna Di Marzo Serugendo, Schahram Dustdar, Anthony Finkelstein, Cristina Gacek, Kurt Geihs, Vincenzo Grassi, Gabor Karsai, Holger M. Kienle, Jeff Kramer, Marin Litoiu, Sam Malek, Raffaela Mirandola, Hausi A. Müller, Sooyong Park, Mary Shaw, Matthias Tichy, Massimo Tivoli, Danny Weyns, and Jon Whittle. 2009. *Software Engineering for Self-Adaptive Systems: A Research Roadmap*. Springer Berlin Heidelberg, Berlin, Heidelberg, 1–26. https://doi.org/10.1007/978-3-642-02161-9_1

[11] Shang-Wen Cheng, David Garlan, and Bradley Schmerl. 2009. Evaluating the effectiveness of the Rainbow self-adaptive system. *Proceedings of the 2009 ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems, SEAMS 2009*, 132–141. https://doi.org/10.1109/SEAMS.2009.5069082

[12] Jane Cleland-Huang. 2023. Truth or Dare: Real-World Fuzz Testing of UAVs in Flight. In *Keynote at the 16th International Workshop on Search-Based and Fuzz Testing (SBFT)*. Melbourne, Australia. Keynote Address.

[13] Jane Cleland-Huang, Ankit Agrawal, Michael Vierhauser, Michael Murphy, and Mike Prieto. 2022. Extending MAPE-K to support Human-Machine Teaming. *CoRR* abs/2203.13036 (2022). https://doi.org/10.1145/3524844.3528054

[14] Jane Cleland-Huang, Nitesh Chawla, Myra Cohen, Md Nafee Al Islam, Urjoshi Sinha, Lilly Spirkovska, Yihong Ma, Salil Purandare, and Muhammed Tawfiq Chowdhury. 2022. Towards real-time safety analysis of small unmanned aerial systems in the national airspace. In *AIAA AVIATION 2022 Forum*. 3540. https://doi.org/10.2514/6.2022-3540

[15] David M. Cohen, Siddhartha R. Dalal, Michael L. Fredman, and Gardner C. Patton. 1997. The AETG system: An approach to testing based on combinatorial design. *IEEE Transactions on Software Engineering* 23, 7 (1997), 437–444. https://doi.org/10.1109/32.605761

[16] M. B. Cohen, C. J. Colbourn, P. B. Gibbons, and W. B. Mugridge. 2003. Constructing test suites for interaction testing. In *Proc. of the Intl. Conf. on Soft. Eng.* 38–48. https://doi.org/10.1109/ICSE.2003.1201186

[17] Ralph D'Agostino and E. S. Pearson. 1973. Tests for Departure from Normality. Empirical Results for the Distributions of b2 and square root b1. *Biometrika* 60, 3 (1973), 613–622. https://doi.org/10.2307/2335012

[18] Rogério de Lemos, Holger Giese, Hausi A. Müller, Mary Shaw, Jesper Andersson, Marin Litoiu, Bradley Schmerl, Gabriel Tamura, Norha M. Villegas, Thomas Vogel, Danny Weyns, Luciano Baresi, Basil Becker, Nelly Bencomo, Yuriy Brun, Bojan Cukic, Ron Desmarais, Schahram Dustdar, Gregor Engels, Kurt Geihs, Karl M. Göschka, Alessandra Gorla, Vincenzo Grassi, Paola Inverardi, Gabor Karsai, Jeff Kramer, Antónia Lopes, Jeff Magee, Sam Malek, Serge Mankovskii, Raffaela Mirandola, John Mylopoulos, Oscar Nierstrasz, Mauro Pezzè, Christian Prehofer, Wilhelm Schäfer, Rick Schlichting, Dennis B. Smith, João Pedro Sousa, Ladan Tahvildari, Kenny Wong, and Jochen Wuttke. 2013. *Software Engineering for Self-Adaptive Systems: A Second Research Roadmap*. Springer Berlin Heidelberg, Berlin, Heidelberg, 1–32. https://doi.org/10.1007/978-3-642-35813-5_1

[19] Ivana Dusparic and Nicolás Cardozo. 2021. Adaptation to Unknown Situations as the Holy Grail of Learning-Based Self-Adaptive Systems: Research Directions. *CoRR* abs/2103.06908 (2021). arXiv:2103.06908 https://arxiv.org/abs/2103.06908

[20] Ahmed Elkhodary, Naeem Esfahani, and Sam Malek. 2010. FUSION: A Framework for Engineering Self-Tuning Self-Adaptive Software Systems. In *Proceedings of the Eighteenth ACM SIGSOFT International Symposium on Foundations of Software Engineering* (Santa Fe, New Mexico, USA) *(FSE '10)*. Association for Computing Machinery, New York, NY, USA, 7–16. https://doi.org/10.1145/1882291.1882296

[21] PX4 Development forum. Last Accessed 01/29/23. PX4 Documentation. https://docs.px4.io/master/en/

[22] ROS Answers forum poster. Last accessed 7/21/24; Posted July 2012. Gazebo simulations not repeatable. https://answers.ros.org/question/40208/gazebo-simulations-not-repeatable/

[23] Open Source Robotics Foundation. Last accessed 7/28/24. Gazebo - A dynamic multi-robot simulator. https://github.com/gazebosim/gazebo-classic

[24] Brady J Garvin, Myra B Cohen, and Matthew B Dwyer. 2013. Failure avoidance in configurable systems through feature locality. In *Assurances for Self-Adaptive Systems*. Springer, 266–296. https://doi.org/10.1007/978-3-642-36249-1_10

[25] Balazs Gati. 2013. Open source autopilot for academic research-the paparazzi system. In *2013 American Control Conference*. IEEE, 1478–1481. https://doi.org/doi:10.1109/ACC.2013.6580045

[26] Paul Gazzillo. 2020. Inferring and Securing Software Configurations Using Automated Reasoning. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (Virtual Event, USA) *(ESEC/FSE 2020)*. Association for Computing Machinery, New York, NY, USA, 1517–1520. https://doi.org/10.1145/3368089.3417041

[27] Thomas J. Glazier, Bradley R. Schmerl, Javier Cámara, and David Garlan. 2017. Utility Theory for Self-Adaptive Systems. In *Technical Report CMU-ISR-17-119*. http://reports-archive.adm.cs.cmu.edu/anon/isr2017/CMU-ISR-17-119.pdf

[28] David Hambling. 2020. Drone Crash Due To GPS Interference In U.K. Raises Safety Questions. (Aug. 2020). https://www.forbes.com/sites/davidhambling/2020/08/10/investigation-finds-gps-interference-caused-uk-survey-drone-crash/?sh=350a3e1d534a

[29] Ruidong Han, Siqi Ma, Juanru Li, Surya Nepal, David Lo, Zhuo Ma, and Jianfeng Ma. 2024. Range Specification Bug Detection in Flight Control System Through Fuzzing. *IEEE Transactions on Software Engineering* 50 (March 2024), 1–13. https://doi.org/10.1109/TSE.2024.3354739

[30] Ruidong Han, Chao Yang, Siqi Ma, JiangFeng Ma, Cong Sun, Juanru Li, and Elisa Bertino. 2022. Control Parameters Considered Harmful: Detecting Range Specification Bugs in Drone Configuration Modules via Learning-Guided Search. In *Proceedings of the International Conference on Software Engineering*. ACM. https://doi.org/10.1145/3510003.3510084

[31] Ruidong Han, Chao Yang, Siqi Ma, JiangFeng Ma, Cong Sun, Juanru Li, and Elisa Bertino. 2022. Control parameters considered harmful: detecting range specification bugs in drone configuration modules via learning-guided search. In *Proceedings of the 44th International Conference on Software Engineering*. 462–473. https://doi.org/10.1145/3510003.3510084

[32] Md Abir Hossen, Sonam Kharade, Bradley Schmerl, Javier Cámara, Jason M. O'Kane, Ellen C. Czaplinski, Katherine A. Dzurilla, David Garlan, and Pooyan Jamshidi. 2023. CaRE: Finding Root Causes of Configuration Issues in Highly-Configurable Robots. https://doi.org/10.48550/ARXIV.2301.07690

[33] Pilot Institute. Last Accessed: 01/29/2022. What Causes Signal Dropouts When Flying Drones? https://pilotinstitute.com/signal-dropout-causes/

[34] M. Al Islam, Y. Ma, P. Alarcon, N. Chawla, and J. Cleland-Huang. 2022. RESAM: Requirements Elicitation and Specification for Deep-Learning Anomaly Models with Applications to UAV Flight Controllers. In *2022 IEEE 30th International Requirements Engineering Conference (RE)*. 153–165. https://doi.org/10.1109/RE54965.2022.00020

[35] Md Nafee Al Islam, Muhammed Tawfiq Chowdhury, Ankit Agrawal, Michael Murphy, Raj Mehta, Daria Kudriavtseva, Jane Cleland-Huang, Michael Vierhauser, and Marsha Chechik. 2023. Configuring mission-specific behavior in a product line of collaborating Small Unmanned Aerial Systems. *J. Syst. Softw.* 197 (2023), 111543. https://doi.org/10.1016/J.JSS.2022.111543

[36] Md Nafee Al Islam, Jane Cleland-Huang, and Michael Vierhauser. 2024. ADAM: Adaptive Monitoring of Runtime Anomalies in Small Uncrewed Aerial Systems. In *Proceedings of the 19th International Symposium on Software Engineering for Adaptive and Self-Managing Systems* (Lisbon, AA, Portugal) *(SEAMS '24)*. Association for Computing Machinery, New York, NY, USA, 44–55. https://doi.org/10.1145/3643915.3644092

[37] Pooyan Jamshidi, Javier Cámara, Bradley Schmerl, Christian Käestner, and David Garlan. 2019. Machine learning meets quantitative planning: Enabling self-adaptation in autonomous robots. In *2019 IEEE/ACM 14th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*. IEEE, 39–50. https://doi.org/10.1109/SEAMS.2019.00015

[38] Chijung Jung, Ali Ahad, Jinho Jung, Sebastian Elbaum, and Yonghwi Kwon. 2021. Swarmbug: Debugging Configuration Bugs in Swarm Robotics. In *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (Athens, Greece) *(ESEC/FSE 2021)*. Association for Computing Machinery, New York, NY, USA, 868–880. https://doi.org/10.1145/3468264.3468601

[39] J.O. Kephart and D.M. Chess. 2003. The vision of autonomic computing. *Computer* 36, 1 (2003), 41–50. https://doi.org/10.1109/MC.2003.1160055

[40] Sajad Khatiri, Sebastiano Panichella, and Paolo Tonella. 2023. Simulation-based Test Case Generation for Unmanned Aerial Vehicles in the Neighborhood of Real Flights. In *2023 IEEE Conference on Software Testing, Verification and Validation (ICST)*. 281–292. https://doi.org/10.1109/ICST57152.2023.00034

[41] Sajad Khatiri, Sebastiano Panichella, and Paolo Tonella. 2024. Simulation-based testing of unmanned aerial vehicles with Aerialist. In *Proceedings of the International Conference on Software Engineering: Companion Proceedings*. 134–138. https://doi.org/10.21256/zhaw-29678

[42] Jinyong Kim, Jinho Lee, Jaehoon Jeong, Hyoungshick Kim, Jung-Soo Park, and Taeho Kim. 2016. SAN: Self-Adaptive Navigation for Drone Battery Charging in Wireless Drone Networks. In *2016 30th International Conference on Advanced Information Networking and Applications Workshops (WAINA)*. 248–251. https://doi.org/10.1109/WAINA.2016.103

[43] Seulbae Kim and Taesoo Kim. 2022. RoboFuzz: Fuzzing Robotic Systems over Robot Operating System (ROS) for Finding Correctness Bugs. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (Singapore, Singapore) *(ESEC/FSE 2022)*. Association for Computing Machinery, New York, NY, USA, 447–458. https:

//doi.org/10.1145/3540250.3549164

[44] Taegyu Kim, Chung Hwan Kim, Junghwan Rhee, Fan Fei, Zhan Tu, Gregory Walkup, Xiangyu Zhang, Xinyan Deng, and Dongyan Xu. 2019. RVFuzzer: Finding Input Validation Bugs in Robotic Vehicles through Control-Guided Testing. In *28th USENIX Security Symposium (USENIX Security 19)*. USENIX Association, Santa Clara, CA, 425–442. https://www.usenix.org/conference/usenixsecurity19/presentation/kim

[45] Cody Kinneer, Zack Coker, Jiacheng Wang, David Garlan, and Claire Le Goues. 2018. Managing uncertainty in self-adaptive systems with plan reuse and stochastic search. In *Proceedings of the 13th International Conference on Software Engineering for Adaptive and Self-Managing Systems*. 40–50. https://doi.org/10.1145/3194133.3194145

[46] Nikolaus Kleber, Jonathan D. Chisum, Aaron Striegel, Bertrand M. Hochwald, Abbas Termos, J. Nicholas Laneman, Zuohui Fu, and John Merritt. 2016. RadioHound: A Pervasive Sensing Network for Sub-6 GHz Dynamic Spectrum Monitoring. *CoRR* abs/1610.06212 (2016). arXiv:1610.06212 http://arxiv.org/abs/1610.06212

[47] N. Koenig and A. Howard. 2004. Design and use paradigms for Gazebo, an open-source multi-robot simulator. In *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*, Vol. 3. 2149–2154 vol.3. https://doi.org/10.1109/IROS.2004.1389727

[48] Guanpeng Li, Yiran Li, Saurabh Jha, Timothy Tsai, Michael Sullivan, Siva Kumar Sastry Hari, Zbigniew Kalbarczyk, and Ravishankar Iyer. 2020. AV-FUZZER: Finding Safety Violations in Autonomous Driving Systems. In *2020 IEEE 31st International Symposium on Software Reliability Engineering (ISSRE)*. 25–36. https://doi.org/10.1109/ISSRE5003.2020.00012

[49] Paulo Henrique Maia, Lucas Vieira, Matheus Chagas, Yijun Yu, Andrea Zisman, and Bashar Nuseibeh. 2019. Dragonfly: a tool for simulating self-adaptive drone behaviours. In *2019 IEEE/ACM 14th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*. IEEE, 107–113. https://doi.org/10.1109/SEAMS.2019.00022

[50] Henry B Mann and Donald R Whitney. 1947. On a test of whether one of two random variables is stochastically larger than the other. *The annals of mathematical statistics* (1947), 50–60.

[51] Lorenz Meier, Dominik Honegger, and Marc Pollefeys. 2015. PX4: A node-based multithreaded open source robotics framework for deeply embedded platforms. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*. 6235–6240. https://doi.org/10.1109/ICRA.2015.7140074

[52] Gabriel Moreno, Cody Kinneer, Ashutosh Pandey, and David Garlan. 2019. DARTSim: An exemplar for evaluation and comparison of self-adaptation approaches for smart cyber-physical systems. In *2019 IEEE/ACM 14th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*. IEEE, 181–187. https://doi.org/10.1109/SEAMS.2019.00031

[53] Thomas Multerer, Alexander Ganis, Ulrich Prechtel, Enric Miralles, Askold Meusling, Jan Mietzner, Martin Vossiek, Mirko Loghi, and Volker Ziegler. 2017. Low-cost jamming system against small drones using a 3D MIMO radar based tracking. In *European Microwave Week 2017: "A Prime Year for a Prime Event", EuMW 2017 - Conference Proceedings; 14th European Microwave Conference, EURAD 2017*, Vol. 2018-Janua. Institute of Electrical and Electronics Engineers Inc., 299–302. https://doi.org/10.23919/EURAD.2017.8249206

[54] Changhai Nie and Hareton Leung. 2011. A Survey of Combinatorial Testing. *ACM Comput. Surv.* 43, 2, Article 11 (Feb 2011), 29 pages. https://doi.org/10.1145/1883612.1883618

[55] Gonzalo Pajares. 2015. Overview and Current Status of Remote Sensing Applications Based on Unmanned Aerial Vehicles (UAVs). *Photogrammetric Engineering & Remote Sensing* 81 (04 2015), 281–330. https://doi.org/10.14358/PERS.81.4.281

[56] PX4 Forum poster. Last accessed 5/21/22; Posted Oct 2019. Strange Harrier D7 crash. https://discuss.px4.io/t/strange-harrier-d7-crash/13480

[57] Salil Purandare and Myra B. Cohen. 2024. Exploration of Failures in an sUAS Controller Software Product Line. In *Proceedings of the 28th ACM International Systems and Software Product Line Conference - Volume B* (Dommeldange, Luxembourg) *(SPLC '24)*. Association for Computing Machinery, New York, NY, USA, 125–135. https://doi.org/10.1145/3646548.3672597

[58] Salil Purandare, Urjoshi Sinha, Md Nafee Al Islam, Jane Cleland-Huang, and Myra B. Cohen. 2023. Self-Adaptive Mechanisms for Misconfigurations in Small Uncrewed Aerial Systems. In *2023 IEEE/ACM 18th Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*. 169–180. https://doi.org/10.1109/SEAMS59076.2023.00030

[59] Georg Püschel, Christian Piechnick, Sebastian Götz, Christoph Seidl, Sebastian Richly, Thomas Schlegel, and Uwe Aßmann. 2014. A combined simulation and test case generation strategy for self-adaptive systems. *Journal On Advances in Software* 7, 3&4 (2014), 686–696.

[60] PX4. Last Accessed 05/19/22. PX4 Bug Repository. https://github.com/PX4/PX4-Autopilot/issues

[61] PX4-Autopilot. Last Accessed 01/29/22. PX4 Online Discussion forum. https://discuss.px4.io/

[62] Mazeiar Salehie and Ladan Tahvildari. 2009. Self-Adaptive Software: Landscape and Research Challenges. *ACM Trans. Auton. Adapt. Syst.* 4, 2, Article 14 (May 2009), 42 pages. https://doi.org/10.1145/1516533.1516538

[63] Ilja Shmelkin. 2020. Monitoring for control in role-oriented self-adaptive systems. In *Proceedings of the IEEE/ACM 15th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*. 115–119. https://doi.org/10.1145/3387939.3391598

[64] Norbert Siegmund, Alexander Grebhahn, Sven Apel, and Christian Kästner. 2015. Performance-Influence Models for Highly Configurable Systems. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering* (Bergamo, Italy) *(ESEC/FSE 2015)*. Association

for Computing Machinery, New York, NY, USA, 284–294. https://doi.org/10.1145/2786805.2786845
[65] Stanford Artificial Intelligence Laboratory et al. [n. d.]. Robotic Operating System. https://www.ros.org
[66] Jacob Swanson, Myra B. Cohen, Matthew B. Dwyer, Brady J. Garvin, and Justin Firestone. 2014. Beyond the Rainbow: Self-Adaptive Failure Avoidance in Configurable Systems. In *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering* (Hong Kong, China) *(FSE 2014)*. Association for Computing Machinery, New York, NY, USA, 377–388. https://doi.org/10.1145/2635868.2635915
[67] Adam Taylor, Sebastian Elbaum, and Carrick Detweiler. 2016. Co-diagnosing configuration failures in co-robotic systems. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2934–2939. https://doi.org/10.1109/IROS.2016.7759454
[68] Christopher Steven Timperley, Tobias Dürschmid, Bradley R. Schmerl, David Garlan, and Claire Le Goues. 2022. ROSDiscover: Statically Detecting Run-Time Architecture Misconfigurations in Robotics Systems. In *19th IEEE International Conference on Software Architecture, ICSA 2022, Honolulu, HI, USA, March 12-15, 2022*. IEEE, 112–123. https://doi.org/10.1109/ICSA53651.2022.00019
[69] Frank Trollmann, Johannes Fähndrich, and Sahin Albayrak. 2018. Hybrid adaptation policies: towards a framework for classification and modelling of different combinations of adaptation policies. In *Proceedings of the 13th International Conference on Software Engineering for Adaptive and Self-Managing Systems*. 76–86. https://doi.org/10.1145/3194133.3194137
[70] Michael Vierhauser, Md Nafee Al Islam, Ankit Agrawal, Jane Cleland-Huang, and James Mason. 2021. Hazard analysis for human-on-the-loop interactions in sUAS systems. In *ESEC/FSE '21: 29th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Athens, Greece, August 23-28, 2021*, Diomidis Spinellis, Georgios Gousios, Marsha Chechik, and Massimiliano Di Penta (Eds.). ACM, 8–19. https://doi.org/10.1145/3468264.3468534
[71] W.E. Walsh, G. Tesauro, J.O. Kephart, and R. Das. 2004. Utility functions in autonomic systems. In *International Conference on Autonomic Computing, 2004. Proceedings.* 70–77. https://doi.org/10.1109/ICAC.2004.1301349
[72] Kai-Tao Xie, Jia-Ju Bai, Yong-Hao Zou, and Yu-Ping Wang. 2022. ROZZ: Property-based Fuzzing for Robotic Programs in ROS. In *2022 International Conference on Robotics and Automation (ICRA)*. 6786–6792. https://doi.org/10.1109/ICRA46639.2022.9811701
[73] Tianyin Xu, Long Jin, Xuepeng Fan, Yuanyuan Zhou, Shankar Pasupathy, and Rukma Talwadker. 2015. Hey, You Have given Me Too Many Knobs!: Understanding and Dealing with over-Designed Configuration in System Software. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering* (Bergamo, Italy) *(ESEC/FSE 2015)*. Association for Computing Machinery, New York, NY, USA, 307–319. https://doi.org/10.1145/2786805.2786852
[74] Tianyin Xu and Yuanyuan Zhou. 2015. Systems Approaches to Tackling Configuration Errors: A Survey. *ACM Comput. Surv.* 47, 4, Article 70 (Jul 2015), 41 pages. https://doi.org/10.1145/2791577

Received 18 December 2023; revised 24 August 2024; accepted 16 October 2024