Self-Adaptive Mechanisms for Misconfigurations in Small Uncrewed Aerial Systems

Salil Purandare
Urjoshi Sinha
Dept. of Computer Science
Iowa State University
Ames, IA, USA
{salil,urjoshi}@iastate.edu

Md Nafee Al Islam
Jane Cleland-Huang
Dept. of Computer Science
University of Notre Dame
Notre Dame, IN, USA
{mislam2,janehuang}@nd.edu

Myra B. Cohen
Dept. of Computer Science
Iowa State University
Ames, IA, USA
mcohen@iastate.edu

***Abstract*—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 simulator and experimented with a large number of different configurations. Experimental results show that *CICADA*'s adaptations reduce controller instability and enable the sUAS to recover from a significant number 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.**

***Index Terms*—self-adaptive software, configurability, uncrewed aerial vehicles**

I. Introduction

Small Uncrewed Aerial Systems[1] (sUAS) are increasingly deployed into unknown environments to support diverse missions – often in response to natural or man-made disasters such as floods, earthquakes, or fires [2], [3]. 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 provide critical intelligence to human rescuers. In order to 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 [4], 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 to self-adapt their behavior to mitigate the problem.

[1]Terminology recommended by the USA Federal Aviation Authority [1].

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 quite 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 a 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* [5], [6]. 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 [5]. They suggest restricting certain parameter settings to guard an RPIC from incorrectly changing parameters during flight. However, 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., [7]). Furthermore, there is little preventing a malicious actor from accessing and modifying control parameters and recent work [8], [9] 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 [6], [10], they lack a proposed solution for sUAS

dependability. One of the most popular software controllers, PX4 [11] has over 1,200 parameters, many with a wide range of potential values, all of which can be manipulated. The possible search space for finding failing scenarios is simply too large 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 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 framework to improve dependability [12]–[15]. 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 [16], [17].

Recently, Braberman et al. [15] 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. In this paper, we propose adaptations of both types, achieving adaptation through modification of configuration parameters and by sending flight commands. We further introduce a method of monitoring flight stability outside of standard controller error reporting with the aim to automatically detect emergent instabilities and to trigger adaptations.

Following on from Braberman et al. we have built a framework called *CICADA*, or *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.

In a series of experiments on the widely used Gazebo flight simulator, we first explore part of the PX4 control parameter space to understand how parameter changes impact the sUAS during flight. We find a wide range of behaviors, including many failure-causing configurations. Utilizing this analysis, we evaluate *CICADA*'s adaptation mechanisms. Our first approach returns to a predefined baseline configuration and attempts to continue the mission. When this fails we invoke other adaptations that abandon the mission but should increase stability: (a) loitering in place, or (b) landing in place.

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 built a configuration-aware self-adaptive framework, *CICADA*, for sUAS which is triggered by monitoring the vehicle's physical state during flight.
3) We conduct 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.

In Section II we present some motivation for our research and an overview of *CICADA* and then Sections III and IV describe the experiments we conducted and report results. We then discuss our findings in more detail (Section V) and point to some interesting follow on investigations. We close by discussing related work (Section VI) and finally presenting our conclusions and future work in Section VII.

II. The *CICADA* Framework

To motivate *CICADA* we point out that configuration errors 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 [7]. 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 [18]. As a simple real-world example, we recently took delivery of new sUAS from a highly qualified manufacturer. However, during initial tests the sUAS experienced intermittent takeoff failures resulting in several crashes, and the root cause was ultimately attributed to a configuration error in one of the flight controller parameters. These examples suggest that understanding the configuration space, monitoring the sUAS' behavior during flight, and re-configuring when needed can potentially increase sUAS dependability.

A. *CICADA* Overview

CICADA is based on MAPE-K [19], and builds upon two existing frameworks. The Rainbow framework [20] is an architecture that provides runtime, self-adaptive capabilities for monitoring, detecting, decision-making, and enactment, while MORPH focuses entirely on sUAS [15], differentiating 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 build upon MORPH by differentiating configuration and behavioral adaptations.

CICADA consists of three layers illustrated in Figure 1. 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

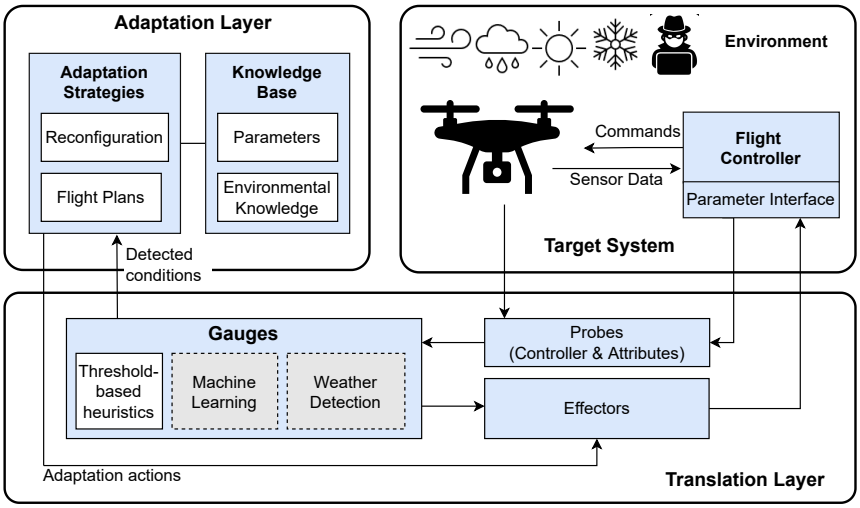

Fig. 1. *CICADA* Architecture: The flight controller in the target system communicates with the sUAS to send mission commands and modify configuration parameters. The sUAS is subject to environmental conditions. The translation layer contains probes, gauges and effectors which interact with the target system and the adaptation layer. Grey boxes indicate future work. The adaptation layer uses its internal knowledge base to select and plan reconfiguration strategies.

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 [21], PX4 [11] or Paparazzi [22]) 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 [23]. 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 [24].

The last part of the translation layer is the effector, which interacts with the adaptation layer and performs 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. 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. [17].

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 threshold violations. This sensor data consists of information about the expected and actual roll, pitch, and yaw of the sUAS. *CICADA* also monitors key 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. [5], 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.

B. Adaptations

We designed three adaptations for *CICADA* which we describe here. Our first adaptation attempts to stabilize the sUAS so that it can complete its flight (albeit with some noise and potential drift). To achieve this adaptation, we reset all changed parameters back to a known set of baseline values. We call this *revert-to-baseline*. Our other adaptations are behavioral. They allow the mission to fail, but attempt to avoid a catastrophic consequence. The first of these strategies is to send the sUAS into *loiter* mode, forcing it to hover in place. The second strategy forces the sUAS to *land* in place.

C. CICADA Instantiation

Our initial instantiation of *CICADA* is simulation-based. This allows us to safely experiment with environmental and configuration factors; however, *CICADA* is fully compatible with our sUAS hardware platform for future deployment on physical sUAS. Once we have gained a deeper understanding

of dependable and safe reconfigurations it should be transferrable since the PX4-Autopilot controller supports hardware deployments as well as software-in-the-loop.

Controller: The PX4-Autopilot software [11] 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.

Configuration Parameters: The PX4-Autopilot flight controller has over 70 categories of parameters, and around 1,200 configurable properties [25]. 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.

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.

Gauges: Gauges aggregate the data from probes and from the PX4 parameter interface. For this instantiation we use two types of gauges (see Section II-A). 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.

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 three adaptations mentioned, (1) **revert-to-baseline**, (2) **loiter**, and (3) **land**.

III. Experimental Evaluation

Our study seeks to answer the following three research questions:[2]

- **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?

A. Configuration Space Model

We first selected a set of parameters based on the work of Kim et al. [5] since the focus of their work was also on flight instability. We then retrieved additional information from three sources; the PX4 discussion forums [26], formal PX4 documentation [25], and the PX4 bug repository [27]. 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 I 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.

B. 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

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

TABLE I

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

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 [28], [29]. 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 [30].

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 I).

C. Metrics

We measure the duration of the mission, the maximum altitude reached, total distance flown, and the maximum tilt. We retain the PX4 logs for each run, which allows us to analyze the flight in detail and view the exact flight path. We also 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. 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.

D. 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 [31]. 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 [32]. 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.

E. 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. 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 that is 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.

F. Threats to Validity

In order to maintain consistency across all our experiments, we ran simulations using a single mission plan and a single sUAS model, which means that we cannot guarantee generality. The experiments were also run in a simulator, rather than on physical systems, due to the potential danger and cost of running sUAS hardware with instability-causing configurations. 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 [33].

Only one sUAS model (the Iris quadcopter) was used in our study 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 also explored 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.

IV. Results

We now present the results of each RQ in turn.

A. RQ1: Studying the Impact of Configurations

Kim et al. [5] 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 II
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%	7.7%	15.3%
1-hop (extended)	1	11.5%	8.6%	2.9%
1-hop (total)	1	16.9%	9.4%	7.5%
2-way (core)	1	72.4%	37.9%	34.5%
2-way (complete)	1	92.0%	84.0%	8.0%

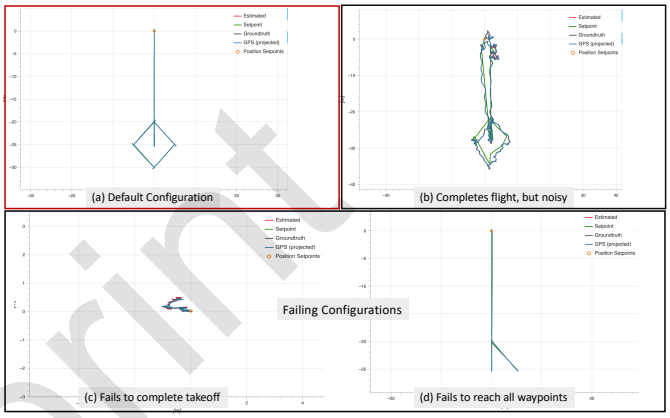

Fig. 2. Example flight paths under different configurations

Table II shows the samples and number of times we ran them (No. Trials). The third column are failure rates of the various samples. This ranges from 11% in the 1-hop extended set to 92% in the 2-way covering array for the complete set. 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. [5]. There were also several cases in which the maximum value was the only one that failed. Figure 2 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 3. They used a flight path deviation-based threshold using the integral 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 a boxplot of the maximum tilt angle for the one-hop core set in Figure 4. We see that failing cases had

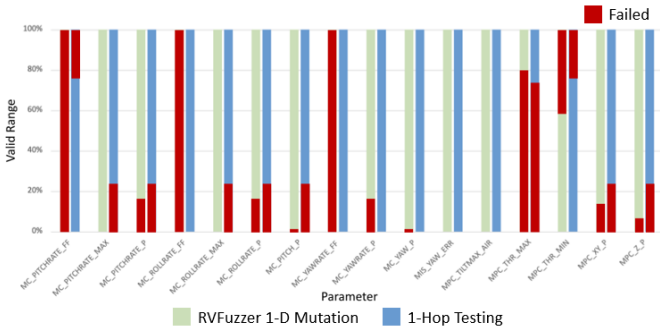

Fig. 3. Comparison of valid and invalid values with RVFuzzer 1-dimensional mutation. Failure-causing parameter values are marked in red.

significantly higher maximum tilt angles on average, as well as a much wider spread, despite the median only being slightly higher than the passing cases. Our observations suggest that this wider spread may be due to some failure-causing configurations inducing high tilt angle fluctuations if they actually fly, while in other configurations the sUAS struggles to take off at all, resulting in very low tilt angles.

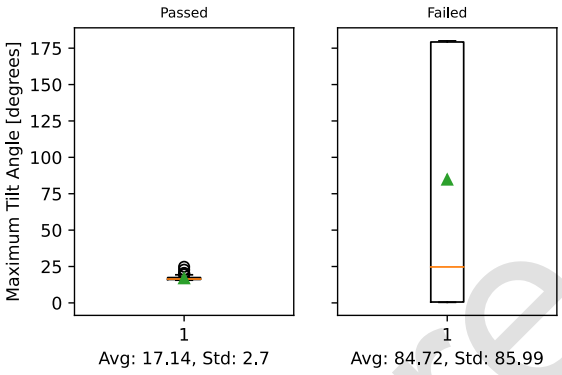

Fig. 4. Maximum tilt angle for passing vs failing 1-hop configurations.

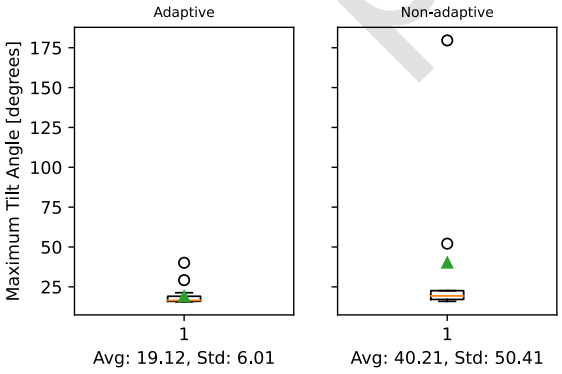

Fig. 5. Maximum tilt angle for 2-way sample configurations with and without the revert-to-baseline adaptation enabled.

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 the configuration parameters have a large impact on flight stability. 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.

B. 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 during a real flight in Table III. This baseline can be captured in our knowledge base or at arming.

TABLE III
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 of Table II.

The revert-to-baseline adaptation strategy reduced the failure rate for all parameter sets we studied, improving by between 2.9% and 34.5% over the non-adaptive version (see Table II). 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 over 70% of configurations, but was able to more than double its success rate, improving to a failure rate of 37.9% with adaptation.

We observed that the maximum tilt angle for the pairwise sample configurations was slightly lower on average when the revert-to-baseline protocol was in place. We established in Section IV-A that failing configurations have a higher maximum tilt angle on average, and that significantly 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. We observed that despite this consideration, the non-adaptive trials still showed slightly higher averages in terms of maximum tilt, and significantly larger spread, as visualized in Figure 5. This may be due to the existence of configurations that didn't entirely prevent mission success, but still introduced a significant amount of instability during the flight, such as the configuration in path (b) of Figure 2. The revert-to-baseline strategy would have reacted to such configurations and modified them, thus reducing the number of high-tilt outliers. The exception to this trend was the 2-way sample for the complete set, which had too few passing configurations without adaptation to demonstrate spread.

The pairwise sample had an initial failure rate of 98%, nearly failing for every configuration. However, the revert-to-baseline strategy still had a noticeable effect improving the success rate eightfold compared to the non-adaptive case. Our observations suggest that one reason for the lower success rate compared to the one-hop 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.

TABLE IV
TAKEOFF OUTCOMES. NUMBER OF FAILING TAKEOFFS AND MISSIONS

Sample	Total configs	Failing Missions	Failing Takeoffs
1-hop non-adaptive (core)	52	12	9
1-hop adaptive (core)	52	4	4
1-hop non-adaptive (extended)	104	12	8
1-hop adaptive (extended)	104	9	7
2-way non-adaptive (core)	29	20	19
2-way adaptive (core)	29	11	11
2-way non-adaptive (complete)	50	46	44
2-way adaptive (complete)	50	42	33

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.

C. 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 6.

Fig. 6. Comparison of expected mission flight path (left) with rapidly ascending flight behavior (right) caused by pairwise misconfigurations.

We study our 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 the revert-to-baseline approach was unable to recover from. We noted that many of the failures were cases which were 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 takeoff after 30 seconds of thrust, we can automatically kill the motors and disarm. This method can be applied to all cases in Table IV 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. We ultimately identified 11 configurations that fit this criteria. Of these, 9 were from the pairwise sample of the complete parameter set, and 2 were from one-hop testing with the MPC_XY_P and MPC_Z_P parameters. The results are summarized in Table V. 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 from happening in all cases where it was previously happening. 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 V
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
Fail Rate			45.5%	27.3%

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 especially 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.

V. 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. In this work we were limited by several factors which we describe here.

- **Additional gauges may improve the 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 could not catch.

- **Need for a Larger Parameter Exploration**. 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 can have a significant impact on 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 can help improve sUAS safety and reliability.

VI. Related Work

There is a large body of research on adaptive systems such as best practices, validation and challenges [34]–[37] as well as research on using the MAPE-K loop [17], [20], [37]–[40]. Several recent studies propose the use of MAPE-K in uncrewed aerial vehicles [15], [41], [42] and robot planning [12], [43]. We discuss some closely related work to ours here.

Braberman et al. [15] proposed a reference architecture (called MORPH) 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 [35] 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. [12] use machine learning to reduce a large configuration space, Elkhodary et al. use feature modeling to reason about potential reconfigurations [44] and Swanson et al. [17] 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 [15], [41]); 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 [12], [45], [46]. Our use of learning is for improving gauges.

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 [47]–[51]; however, none of these focuses on the sUAS environment or self-adaptation.

Finally, there has been recent work on fuzzing sUAS to find configuration problems (e.g. [8]–[10], [52], [53]) which focuses on finding and fixing configurations in sUAS, or studies instability due to malicious threats [5], [54]. However, none of this work proposes an automated, self-adaptive approach for recovery. We have incorporated many of the parameters from these studies in our core parameter set.

VII. 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. Finally, we 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.

As future work we plan to devise more intelligent adaptation strategies to automatically apply the best adaptation mechanism at any given instant of a flight, expand our experiments to incorporate improved (ML based) gauges, explore an even larger parameter space, and run experiments on a larger variety of flight missions and environmental conditions (e.g. wind). Last, we plan to incorporate *CICADA* onto our physical drone systems to demonstrate its potential for real-world usage.

Acknowledgments

The work in this paper was primarily funded under USA National Aeronautics and Space Administration (NASA) Grant Number: 80NSSC21M0185 and the National Science Foundation (NSF) CNS-1931962 and CCF-1909688. We thank A. Nicolellis for help collecting data for a preliminary study.

References

[1] "U.S. Government Accountability Office, Drone operations." [Online]. Available: https://www.gao.gov/uncrewed-aircraft-systems

[2] G. Pajares, "Overview and current status of remote sensing applications based on unmanned aerial vehicles (UAVs)," *Photogrammetric Engineering & Remote Sensing*, vol. 81, pp. 281–330, 04 2015.

[3] B. Benjdira, T. Khursheed, A. Koubaa, A. Ammar, and K. Ouni, "Car detection using unmanned aerial vehicles: Comparison between faster r-cnn and yolov3," in *2019 1st International Conference on Unmanned Vehicle Systems-Oman (UVS)*, 2019, pp. 1–6.

[4] T. Multerer, A. Ganis, U. Prechtel, E. Miralles, A. Meusling, J. Mietzner, M. Vossiek, M. Loghi, and V. Ziegler, "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., Jun 2017, pp. 299–302.

[5] T. Kim, C. H. Kim, J. Rhee, F. Fei, Z. Tu, G. Walkup, X. Zhang, X. Deng, and D. Xu, "RVFuzzer: Finding input validation bugs in robotic vehicles through Control-Guided testing," in *28th USENIX Security Symposium (USENIX Security 19)*. Santa Clara, CA: USENIX Association, Aug. 2019, pp. 425–442. [Online]. Available: https://www.usenix.org/conference/usenixsecurity19/presentation/kim

[6] R. Han, C. Yang, S. Ma, J. Ma, C. Sun, J. Li, and E. Bertino, "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, 2022. [Online]. Available: https://arxiv.org/abs/2112.03511

[7] P. F. poster, "Strange harrier d7 crash," Last accessed 5/21/22; Posted Oct 2019. [Online]. Available: https://discuss.px4.io/t/strange-harrier-d7-crash/13480

[8] S. Kim and T. Kim, "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*, ser. ESEC/FSE 2022. New York, NY, USA: Association for Computing Machinery, 2022, p. 447–458. [Online]. Available: https://doi.org/10.1145/3540250.3549164

[9] K.-T. Xie, J.-J. Bai, Y.-H. Zou, and Y.-P. Wang, "Rozz: Property-based fuzzing for robotic programs in ros," in *2022 International Conference on Robotics and Automation (ICRA)*, 2022, pp. 6786–6792.

[10] M. A. Hossen, S. Kharade, B. Schmerl, J. Cámara, J. M. O'Kane, E. C. Czaplinski, K. A. Dzurilla, D. Garlan, and P. Jamshidi, "Care: Finding root causes of configuration issues in highly-configurable robots," 2023. [Online]. Available: https://arxiv.org/abs/2301.07690

[11] L. Meier, D. Honegger, and M. Pollefeys, "PX4: A node-based multithreaded open source robotics framework for deeply embedded platforms," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, 2015, pp. 6235–6240.

[12] P. Jamshidi, J. Cámara, B. Schmerl, C. Käestner, and D. Garlan, "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, 2019, pp. 39–50.

[13] P. H. Maia, L. Vieira, M. Chagas, Y. Yu, A. Zisman, and B. Nuseibeh, "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, 2019, pp. 107–113.

[14] G. Moreno, C. Kinneer, A. Pandey, and D. Garlan, "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, 2019, pp. 181–187.

[15] V. Braberman, N. D'Ippolito, J. Kramer, D. Sykes, and S. Uchitel, "MORPH: A reference architecture for configuration and behaviour self-adaptation," in *Proceedings of the 1st International Workshop on Control Theory for Software Engineering*, 2015, pp. 9–16.

[16] B. J. Garvin, M. B. Cohen, and M. B. Dwyer, "Failure avoidance in configurable systems through feature locality," in *Assurances for Self-Adaptive Systems*. Springer, 2013, pp. 266–296.

[17] J. Swanson, M. B. Cohen, M. B. Dwyer, B. J. Garvin, and J. Firestone, "Beyond the rainbow: Self-adaptive failure avoidance in configurable systems," in *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, ser. FSE 2014. New York, NY, USA: Association for Computing Machinery, 2014, p. 377–388. [Online]. Available: https://doi.org/10.1145/2635868.2635915

[18] D. Hambling, "Drone crash due to GPS interference in U.K. raises safety questions," Aug. 2020. [Online]. Available: https://www.forbes.com/sites/davidhambling/2020/08/10/investigation-finds-gps-interference-caused-uk-survey-drone-crash/?sh=350a3e1d534a

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

[20] S.-W. Cheng, D. Garlan, and B. Schmerl, "Evaluating the effectiveness of the rainbow self-adaptive system," 05 2009, pp. 132–141.

[21] Ardupilot, "Ardupilot open source autopilot," Last Accessed 01/29/22. [Online]. Available: https://ardupilot.org/

[22] B. Gati, "Open source autopilot for academic research-the paparazzi system," in *2013 American Control Conference*. IEEE, 2013, pp. 1478–1481.

[23] S. J. Abraham, Z. Carmichael, S. Banerjee, R. G. VidalMata, A. Agrawal, M. N. A. Islam, W. J. Scheirer, and J. Cleland-Huang, "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, 2021, pp. 113–120. [Online]. Available: https://doi.org/10.1109/WAIN52551.2021.00025

[24] M. A. Islam, Y. Ma, P. Alarcon, N. Chawla, and J. Cleland-Huang, "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)*, Aug 2022, pp. 153–165.

[25] P. D. forum, "PX4 documentation," Last Accessed 01/29/23. [Online]. Available: https://docs.px4.io/master/en/

[26] PX4-Autopilot, "PX4 online discussion forum," Last Accessed 01/29/22. [Online]. Available: https://discuss.px4.io/

[27] PX4, "PX4 bug repository," Last Accessed 05/19/22. [Online]. Available: https://github.com/PX4/PX4-Autopilot/issues

[28] C. Nie and H. Leung, "A survey of combinatorial testing," *ACM Comput. Surv.*, vol. 43, no. 2, Feb 2011. [Online]. Available: https://doi.org/10.1145/1883612.1883618

[29] D. M. Cohen, S. R. Dalal, M. L. Fredman, and G. C. Patton, "The AETG system: An approach to testing based on combinatorial design," *IEEE Transactions on Software Engineering*, vol. 23, no. 7, pp. 437–444, 1997.

[30] M. B. Cohen, C. J. Colbourn, P. B. Gibbons, and W. B. Mugridge, "Constructing test suites for interaction testing," in *Proc. of the Intl. Conf. on Soft. Eng.*, May 2003, pp. 38–48.

[31] Stanford Artificial Intelligence Laboratory et al., "Robotic operating system." [Online]. Available: https://www.ros.org

[32] N. Koenig and A. Howard, "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, 2004, pp. 2149–2154 vol.3.

[33] M. Al Islam, M. Chowdhury, P. Granadeno, J. Cleland-Huang , and L. Spirkovska, "Towards an annotated all-weather dataset of flight logs for small uncrewed aerial systems," in *AIAA AVIATION 2023 Forum*.

[34] B. H. C. Cheng, R. de Lemos, H. Giese, P. Inverardi, J. Magee, J. Andersson, B. Becker, N. Bencomo, Y. Brun, B. Cukic, G. Di Marzo Serugendo, S. Dustdar, A. Finkelstein, C. Gacek, K. Geihs, V. Grassi, G. Karsai, H. M. Kienle, J. Kramer, M. Litoiu, S. Malek, R. Mirandola, H. A. Müller, S. Park, M. Shaw, M. Tichy, M. Tivoli, D. Weyns, and J. Whittle, *Software Engineering for Self-Adaptive Systems: A Research Roadmap*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 1–26. [Online]. Available: https://doi.org/10.1007/978-3-642-02161-9_1

[35] I. Shmelkin, "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*, 2020, pp. 115–119.

[36] F. Trollmann, J. Fähndrich, and S. Albayrak, "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*, 2018, pp. 76–86.

[37] M. Salehie and L. Tahvildari, "Self-adaptive software: Landscape and research challenges," *ACM Trans. Auton. Adapt. Syst.*, vol. 4, no. 2, May 2009. [Online]. Available: https://doi.org/10.1145/1516533.1516538

[38] C. Kinneer, Z. Coker, J. Wang, D. Garlan, and C. L. Goues, "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*, 2018, pp. 40–50.

[39] R. de Lemos, H. Giese, H. A. Müller, M. Shaw, J. Andersson, M. Litoiu, B. Schmerl, G. Tamura, N. M. Villegas, T. Vogel, D. Weyns, L. Baresi, B. Becker, N. Bencomo, Y. Brun, B. Cukic, R. Desmarais, S. Dustdar, G. Engels, K. Geihs, K. M. Göschka, A. Gorla, V. Grassi, P. Inverardi, G. Karsai, J. Kramer, A. Lopes, J. Magee, S. Malek, S. Mankovskii, R. Mirandola, J. Mylopoulos, O. Nierstrasz, M. Pezzè, C. Prehofer, W. Schäfer, R. Schlichting, D. B. Smith, J. P. Sousa, L. Tahvildari, K. Wong, and J. Wuttke, *Software Engineering for Self-Adaptive Systems: A Second Research Roadmap*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 1–32. [Online]. Available: https://doi.org/10.1007/978-3-642-35813-5_1

[40] P. Arcaini, E. Riccobene, and P. Scandurra, "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*, May 2015, pp. 13–23.

[41] J. Kim, J. Lee, J. Jeong, H. Kim, J.-S. Park, and T. Kim, "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)*, 2016, pp. 248–251.

[42] J. Cleland-Huang, A. Agrawal, M. Vierhauser, M. Murphy, and M. Prieto, "Extending MAPE-K to support human-machine teaming," *CoRR*, vol. abs/2203.13036, 2022. [Online]. Available: https://doi.org/10.1145/3524844.3528054

[43] G. Püschel, C. Piechnick, S. Götz, C. Seidl, S. Richly, T. Schlegel, and U. Aßmann, "A combined simulation and test case generation strategy for self-adaptive systems," *Journal On Advances in Software*, vol. 7, no. 3&4, pp. 686–696, 2014.

[44] A. Elkhodary, N. Esfahani, and S. Malek, "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*, ser. FSE '10. New York, NY, USA: Association for Computing Machinery, 2010, p. 7–16. [Online]. Available: https://doi.org/10.1145/1882291.1882296

[45] I. Dusparic and N. Cardozo, "Adaptation to unknown situations as the holy grail of learning-based self-adaptive systems: Research directions," *CoRR*, vol. abs/2103.06908, 2021. [Online]. Available: https://arxiv.org/abs/2103.06908

[46] F. J. Affonso, G. Leite, R. A. Oliveira, and E. Y. Nakagawa, "A framework based on learning techniques for decision-making in self-adaptive software." in *SEKE*, vol. 15, 2015, pp. 1–6.

[47] P. Gazzillo, "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*, ser. ESEC/FSE 2020. New York, NY, USA: Association for Computing Machinery, 2020, p. 1517–1520. [Online]. Available: https://doi.org/10.1145/3368089.3417041

[48] T. Xu and Y. Zhou, "Systems approaches to tackling configuration errors: A survey," *ACM Comput. Surv.*, vol. 47, no. 4, Jul 2015. [Online]. Available: https://doi.org/10.1145/2791577

[49] T. Xu, L. Jin, X. Fan, Y. Zhou, S. Pasupathy, and R. Talwadker, "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*, ser. ESEC/FSE 2015. New York, NY, USA: Association for Computing Machinery, 2015, p. 307–319. [Online]. Available: https://doi.org/10.1145/2786805.2786852

[50] M. Cashman, M. B. Cohen, P. Ranjan, and R. W. Cottingham, "Navigating the maze: The impact of configurability in bioinformatics software," in *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*, ser. ASE '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 757–767. [Online]. Available: https://doi.org/10.1145/3238147.3240466

[51] N. Siegmund, A. Grebhahn, S. Apel, and C. Kästner, "Performance-influence models for highly configurable systems," in *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, ser. ESEC/FSE 2015. New York, NY, USA: Association for Computing Machinery, 2015, p. 284–294. [Online]. Available: https://doi.org/10.1145/2786805.2786845

[52] A. Taylor, S. Elbaum, and C. Detweiler, "Co-diagnosing configuration failures in co-robotic systems," in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2016, pp. 2934–2939.

[53] C. Jung, A. Ahad, J. Jung, S. Elbaum, and Y. Kwon, "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*, ser. ESEC/FSE 2021. New York, NY, USA: Association for Computing Machinery, 2021, p. 868–880. [Online]. Available: https://doi.org/10.1145/3468264.3468601

[54] G. Li, Y. Li, S. Jha, T. Tsai, M. Sullivan, S. K. S. Hari, Z. Kalbarczyk, and R. Iyer, "Av-fuzzer: Finding safety violations in autonomous driving systems," in *2020 IEEE 31st International Symposium on Software Reliability Engineering (ISSRE)*, 2020, pp. 25–36.