

# Hotpatching on the Fly - Mitigating Drone Incidents Arising from Incorrect Configuration

Ruidong Han<sup>✉</sup>, Juanru Li, *Member, IEEE*, Zhuo Ma<sup>✉</sup>, *Member, IEEE*, David Lo,<sup>✉</sup> *Fellow, IEEE*, Arash Shaghaghi<sup>✉</sup>, *Member, IEEE*, JianFeng Ma<sup>✉</sup>, *Member, IEEE*, and Siqi Ma<sup>✉</sup>, *Member, IEEE*,

**Abstract**—Manufacturers offer adjustable control parameters for flight control systems to accommodate diverse environments and missions. To ensure flight safety, they also develop established boundaries, i.e., range specification for parameter values. However, even when the configuration parameters fall within the prescribed manufacturer range, it could still lead to instability or even severe incidents like crashes, which are *Range Specification Bugs*. Prior research has suggested shrinking the range of parameter values to protect drones from the adverse effects of such bugs. However, narrowing the range of parameters may only reduce the probability of errors and could potentially limit the adaptability of the drone. To overcome this limitation, we present an online approach that analyzes a sequence of flight states to detect any potential triggering of bugs and rectify the drone by dynamically adjusting its parameters. We implemented the rectification approach, CONFIX, and applied it in current prevalent flight control systems, ArduPilot and PX4. The results demonstrated that CONFIX achieved a rectification success at 80% on average.

## I. INTRODUCTION

Drones have been widely used for personal entertainment and commercial purposes (e.g., photo shooting and goods delivery) and military purposes (e.g., target surveillance). Manufacturers design custom flight control systems integrated with configurable features to fulfill various flight missions and environmental changes. Specifically, by selecting parameter values from the suggested ranges, the user sets a configuration (parameter combination) to match their flight requirements, such as establishing the maximum flight angle. However, recent research demonstrated that the suggested value ranges are inaccurate (i.e., *Range Specification Bugs*) [1, 2, 3]. Even if their parameter values are selected from the suggested value ranges, some incorrect configurations may trigger severe incidents (such as thrust loss, crash, and trajectory offset), thereby disrupting the mission.

Ruidong Han is with the School of Computing and Information Systems, Singapore Management University, Singapore, and with the School of Cyber Engineering, Xidian University, Xi'an (e-mail: ruidonghanxd@outlook.com).

Juanru Li is with Feiyu Security, Shanghai China (e-mail: mail@lijuanru.com).

Zhuo Ma and JianFeng Ma are with the School of Cyber Engineering, Xidian University, Xi'an (e-mail: mazhuo@mail.xidian.edu.cn; jfma@mail.xidian.edu.cn).

David Lo is with the School of Computing and Information Systems, Singapore Management University (e-mail: davidlo@smu.edu.sg).

Arash Shaghaghi and Siqi Ma are with the School of Computer Science and Engineering, UNSW Sydney, Australia (e-mail: a.shaghaghi@unsw.edu.au; siqi.ma@unsw.edu.au).

This work was supported by the National Natural Science Foundation of China (Key Program 62232013)

Unlike other drone bugs [4, 5, 6, 7, 8], such as the ones related to sensor data (e.g., sensor spoofing attacks) or system disruption (e.g., buffer overflow attacks), range specification bugs arise because of inadequate implementation logic of configuration mechanism. Consequently, program analysis methodologies [9, 10, 11, 12, 13, 14, 15] may be invalid for addressing this bug at the code level. Moreover, a control system usually contains hundreds or even thousands of parameters, and each parameter commonly has a range containing more than 100 values. Faced with almost infinite combinations of parameters, developers can not provide an impeccable range that can avoid bugs. Some existing research [1, 2, 3] suggested shrinking the value range of each parameter to achieve a safer flight. However, constrained by the excessive scale of combinations, they can only decrease the probability of bug triggers, in which incidents may still occur and can not be remedied during flight. Additionally, range-shrinking methods also restrict the adaptive capabilities because fewer selections of configurations can be set.

Some methods [16, 17] utilize dynamic schemes that reset the configuration to default or average parameter values derived from historical flight data upon detecting instability. However, the variability in complex environments and the need for flexible mission adjustments can render default or average values insufficient for current flight. This inadequacy may lead to mission interruptions or even trigger system bugs.

To address *Range Specification Bugs*, and especially avoid severe incidents, we develop an online rectification approach, CONFIX. It dynamically monitors the drone and initiates rectification in the event of instability. It generates a new configuration tailored to accommodate the current flight, maintaining stability. Intuitively, CONFIX utilizes a state predictor for identifying instability and facilitating subsequent rectification, which is trained using past flight data encompassing various configurations. By referencing the current state and configuration, this predictor can forecast changes in the drone's state (e.g., angular roll, pitch, and yaw).

During a flight execution, CONFIX initially detects potential instability by observing deviations between the actual drone achieved state and the predictions made by the predictor. Once instability is identified, it utilizes a learning-guided configuration optimizer to generate an appropriate configuration. The state predictor estimates which configuration is better during optimization. It guides the optimizer in selecting the best configuration, allowing for adjustments without requiring actual flight response, thereby enhancing responsiveness and safety. This capability enables the rapid generation of configurations

on the fly. Once the optimal configuration is determined, the system uploads the final evaluated configuration as the rectification to the drone

We assessed the effectiveness and efficiency of CONFIX using two flight control systems: ArduPilot and PX4. In terms of state rectification performance, when the number of examined parameters is constrained to no more than 12, CONFIX achieves an average success rate of 80% in rectifying the state.

### Contributions:

- We introduce a dynamic rectification approach that effectively addresses instability and prevents incidents caused by range specification bugs.
- We designed and implemented CONFIX, an approach that can be used for flight control systems without code modification, even though their vehicle models are different.
- We applied CONFIX to the widely used flight control systems, ArduPilot and PX4, to demonstrate the performance of our solution. CONFIX successfully identified the instability and rectified it during flight. And the source code is available at <https://github.com/BlackJocker1995/LGDFix>

## II. CONFIGURATION AND RANGE SPECIFICATION BUGS

This section introduces some background about flight control system and range specification bugs. Additionally, we present the threat model.

### A. Configuration for Flight System

Manufacturers provide customizable control schemes with various parameters to adapt to different environments and mission requirements. For instance, the `ANGLE_MAX` parameter in Ardupilot influences the maximum tilt angle of the drone, which can range from 10 to 80 degrees. A higher tilt angle allows the drone to achieve greater speeds, enhancing performance in speed-critical tasks. However, this increased speed also introduces risks, such as challenges in braking and maneuverability. Therefore, users may need to adjust the `ANGLE_MAX` based on their specific mission requirements. If a mission prioritizes speed, a higher value may be appropriate. Conversely, if safety and control are paramount, a lower value might be more suitable to mitigate risks and ensure stable flight.

However, previous investigations [1, 2, 3] demonstrate that certain parameter combinations (i.e., incorrect configurations), despite being within the prescribed limits, can still lead to drones instability. This phenomenon is referred to as *Range Specification Bugs*, which indicates that the drone’s developer has not accurately formulated the range limitations. They result in significant instability, leading to severe incidents, which we categorize as follows:

- **Trajectory Offset.** The drone’s actual flight trajectory can not follow the planned trajectory and has a continuous deviation.
- **Thrust Loss.** The drone requires more motor power for state rectification even though the current throttle has already saturated the motor up to 100%, raising the thrust loss warning.

- **Flight Freeze.** The drone moves forward/backward or wanders around a position within a very small range.
- **Crash.** The drone hits an object or rolls over on its side and eventually crashes.

### B. Threat Models

The threats posed by drones arise from both external attacks and internal misconfigurations. The steps associated with these threats are illustrated in Figure 1. In the context of an external attack, we postulate that the attacker mimics incorrect configurations through simulated testing and can exploit vulnerabilities in wireless or radio communications [18, 19, 20]. The attacker does not need to resort to malicious code injection, memory corruption, or sensor spoofing. Instead, they can devise and transmit configurations directly to the drone via the compromised wireless or radio channel. Unlike malicious commands, these commands induce incident by exploiting their configuration mechanisms rather than invading external systems. Consequently, such an attack may mislead users into considering it as a routine accident resulting from user operations or system bugs.

In the case of internal misconfiguration, we assume that a drone user, lacking familiarity with the appropriate configuration methods, sets inappropriate parameter values. This misconfiguration can effects similar to those caused by external attacks. Without foundational knowledge, the user may struggle to identify the root cause of the issue, further complicating the resolution process and potentially leading to unsafe flight conditions.

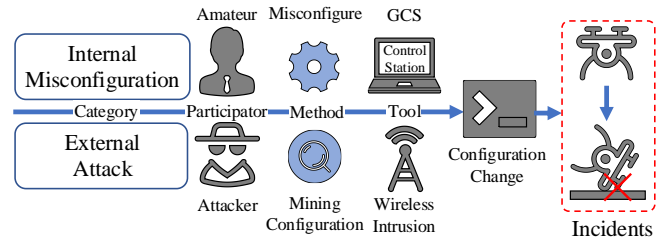


Fig. 1: Threat models of two situations.

## III. CONFIX DESIGN

In this section, we present the design and implementation of CONFIX, dynamically rectifying the instability caused by range specification bugs.

### A. Overview of CONFIX

Figure 2 demonstrates the workflow of CONFIX. During flight, CONFIX actively monitors the flight state of the drone (①). CONFIX dynamically measures data as a segment (called *data segment* below) (②). For each data segment, CONFIX utilizes the state predictor to forecast desired states and subsequently measures the deviations between these desired states and the drone’s achieved states. And CONFIX identifies instability referring to the accumulated deviation in the segment (③). If CONFIX identifies the drone is unstable and the

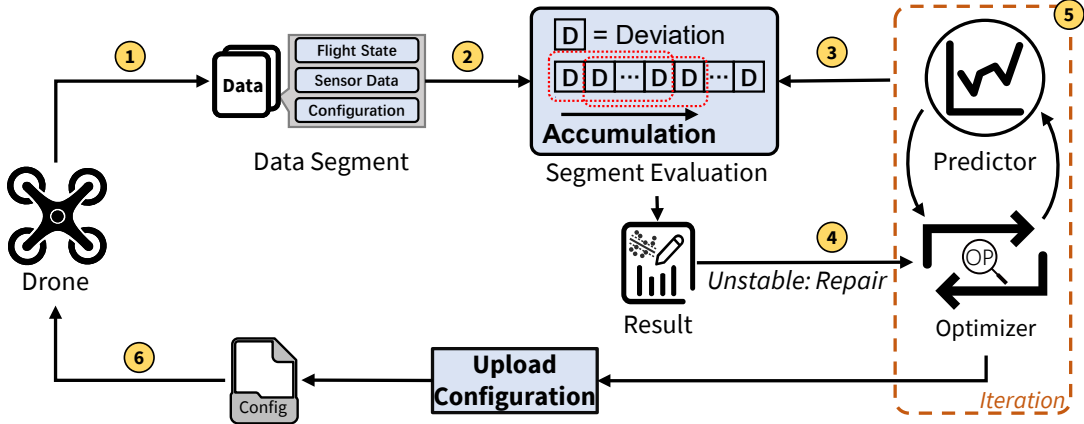


Fig. 2: Overview of CONFIX.

configuration has been changed, it then generates a rectification to adjust the flight (④). For rectification, CONFIX uses a learning-guided optimizer to mutate the current configuration loaded to obtain the most adaptive configuration (⑤). Finally, CONFIX uploads this configuration to the drone (⑥).

### B. Flight Data Capture

CONFIX is an implementation independent of the control system. It achieves data communication and configuration uploading through wireless communication. While in flight, the drone broadcasts its flight states, sensor data, and current configuration wirelessly. Consequently, the CONFIX is capable of capturing this information. Specifically, CONFIX acquires (1) **Flight State**  $u$ : angle (Roll, Pitch, Yaw) and angular rate (Roll\_Rate, Pitch\_Rate, Yaw\_Rate) of attitude. (2) **Sensor Data**  $s$ : raw sensor measurement obtained from gyroscopes (X, Y, Z axis) and accelerometers (X, Y, Z axis). (3) current **Configuration**  $x$ . Since different drone components operate at varying update frequencies, we synchronized these data streams to a common frequency of 10Hz, i.e., each second consists of 10 entries. Therefore, every flight data captured at timestamp  $t$  can be denoted as  $v(t) = \{u(t), s(t), x(t)\}$ , where the timestamp unit is 0.1 second.

### C. State Predictor

The state predictor implements a function that forecasts the future state based on the contextual correlations among the flight state, sensor data, and configuration in the time series. Specifically, at a timestamp  $t$ , state predictor leverages  $h$  consecutive flight data  $V = \{v(t-h+1), v(t-h+2), \dots, v(t)\}$  to forecast the flight state at the next timestamp  $t+1$ , i.e.,  $u'(t+1)$ . In stable flight, the predicted state  $u'(t+1)$  should be close to the actual state  $u(t+1)$ ; deviations will occur in instances of instability.

We implement a predictor with Temporal Convolutional Network (TCN) [21]. We conducted multiple flight missions, AVC2013 [22], while setting various configurations to train it. For these data logs, we discarded those that cause incidents (contain instability) for the following two reasons: (1)

By learning patterns from flight data devoid of instability, the predictor will exhibit a deviation in predictions when encountering instability. Leveraging this deviated property, CONFIX can detect instability. (2) When employing incident-free flight data for learning, rectification experiences utilized by the learning-guided optimizer to generate configuration are considered positive.

### D. Incorrect Configuration Alert

CONFIX monitors the flight state and assesses whether the drone is unstable (indicating the current configuration is incorrect). As our state predictor is trained on flight data devoid of instability, it yields larger predicted deviations when the drone experiences instability, whereas stable flights do not exhibit such deviations. CONFIX employs the state predictor to analyze deviation in the segment (comprising  $m$  flight data) to detect instability.

Specifically, suppose a segment in timestamp  $t$  is  $V(t) = \{V(i) \mid i \in [t, t+m-1]\}$ . Through the state predictor, CONFIX can obtain  $m-h$  predicted result (desired states)  $U' = \{u'(i) \mid i \in [t+h, t+m-1]\}$ . CONFIX matches each predicted result with its corresponding real flight state and calculates their deviations:

$$d(i) = \|u(i) - u'(i)\|, i \in [t+h, t+m-1] \quad (1)$$

Since deviations tend to escalate during instability, CONFIX employs a fine-grained window sliding approach to detect this trend. A window size that is too small fails to adequately reflect changes in the drone's status, while an excessively large window increases computational overhead. Based on the data collection frequency of 0.1 seconds (10 Hz), we manually set the window size as  $10-h$  to capture the state deviation occurring every second and analyze the deviation trend. CONFIX count the accumulated deviation  $d_w$  occurring in every window:

$$d_w(i) = \sum_i^{i+10-h} d(i), i \in [t+h, t+m+h-11] \quad (2)$$

CONFIX finally selects the maximum and the minimum accumulated deviations and computes their average value to represent the *deviation trend value (DTV)*.

CONFIX employs a detection threshold  $TH$  (we experimentally determined this  $TH$ ) to evaluate the trend value. If DTV exceeds  $TH$ , CONFIX labels the data segment as “unstable”, indicating that the current configuration is incorrect and should launch a rectification. Conversely, CONFIX labels the data segment as “stable” and checks the next segment.

### E. Rectification Configuration

When a data segment is identified as unstable, CONFIX utilizes a learning-guided optimizer to generate configuration and rectifies the flight accordingly. Suppose the segment is  $V(t)$ . The following describes the optimization process and how configurations are evaluated.

1) *Configuration Optimizer*: The optimizer initializes a population of size  $NP$ ; each individual in the population is a configuration whose parameter initial values are identical to those of the segment’s setting. The optimizer iteratively mutates this population and searches for the best configuration.

Specifically, for iteration  $g$ -th ( $g \in [1, G_{max}]$ ), the corresponding population, denoted as  $pop_c(g) = \{x_i(g) \mid i \in [1, NP]\}$ . The optimizer applies *differential evolution* technique [23] to the current population to generate a variant population  $pop_v(g) = \{y_i(g) \mid i \in [1, NP]\}$ . And each individual  $y_i(g)$  is generated by:

$$x_i(g) + F * (x_{best}(g) - x_i(g)) + F * (x_{r1}(g) - x_{r2}(g)) \quad (3)$$

where  $x_{r1/r2}(g)$  are random configurations,  $x_{best}(g)$  is the best fitness configuration, and  $F$  is the scaling factor. Then, CONFIX applies a *binomial distribution crossover* in the *differential evolution* to  $pop_c(g)$  and  $pop_v(g)$  to produce a new experimental population  $pop_e(g) = \{e_i(g) \mid i \in [1, NP]\}$ .

Using our *evaluation function*  $Fit(\cdot)$  (Sec. III-E2), the optimizer evaluates the fitness of each configuration both in  $pop_c(g)$  and  $pop_e(g)$ . Based on the fitness results, the optimizer retains the configuration with the least fitness, i.e., the smallest summation of deviations, from the two parallel populations  $pop_c(g)$  and  $pop_e(g)$ . This selected is used to generate the next generation population  $pop_c(g+1)$ , and its individual  $x_i(g+1), i \in [1, NP]$  is:

$$x_i(g+1) = \begin{cases} x_i(g), & Fit(x_i(g)) < Fit(e_i(g)) \\ e_i(g), & otherwise \end{cases} \quad (4)$$

Ultimately, when the fitness of each individual (configuration) no longer decreases, or the maximum number  $G_{max}$  is reached, the optimizer stops and selects the best configuration, that is, the one with the smallest fitness in the latest generation, as the rectification.

2) *Evaluation Function*: The optimizer utilizes the state predictor to evaluate the fitness of each configuration. Suppose a candidate configuration is  $x_{cad}\{p_1, p_2, \dots, p_d\}$ , where  $d$  is

the number of its parameters. CONFIX initially replaces the parameter values to generate a new vector set  $V_{new}$ :

$$\begin{aligned} V(i) &= \{u(i), s(i), x(i)\}, \\ \{u(i), s(i), x(i)\} &\rightarrow \{u(i), s(i), x_{cad}\}, \\ v_{new}(i) &= \{u(i), s(i), x_{cad}\}, \\ V_{new} &= \{v_{new}(i) \mid i \in [t, t+m-1]\} \end{aligned} \quad (5)$$

CONFIX applies its predictor to create the predicted results  $V'_{new}$  and get deviations:

$$d(i) = \|u(i) - u'_{new}(i)\|, i \in [t+h, t+m-1] \quad (6)$$

Unlike detection, which relies on multiple accumulated values, the optimizer aims to select the most suitable configuration that causes the minimum deviation in the segment. Hence, the optimizer calculates the deviation summation as the fitness of the candidate configuration  $x_{cad}$ :

$$Fit(x_{cad}) = \sum_{i=t+h}^{t+m-h} d(i) \quad (7)$$

According to the fitness, the optimizer assesses whether the deviation can be reduced when loading the candidate configuration to the current data segment.

## IV. EVALUATION

This section presents the experimental setup of evaluating CONFIX and the search results.

### A. Experiment Setup

**Program and Vehicle.** Our experiments involves two popular flight control program ArduPilot (4.2.0) and PX4 (1.13). To conduct the majority of test cases, we employed simulators, i.e., *APM* and *Gazebo* for ArduPilot and *Jmavsim* for PX4. Additionally, we employ a physical drone, namely the *CUAV ZD550*, to evaluate the effectiveness in real-world scenarios.

**CONFIX Implementation.** We implement CONFIX with Python 3.9 on the ground control station, i.e., a laptop with Ubuntu 20.04 in our experiments. In virtual experiments, CONFIX connects the virtual drone using inter-process communication. In real flight experiments, CONFIX connects to the drone with a WIFI connection. Both establish communication with drone’s flight control system through *Mavlink* [24] protocol (its commands can be found in Web<sup>1</sup>). Leveraging this communication, CONFIX captures essential data, including flight states, sensor data, and the current configuration of the drone. It also sends new configurations as needed. After takeoff, CONFIX asynchronously acquires a series of data to construct a segment for instability detection. When an issue arises, the system generates a suitable configuration that can mitigate the instability estimated by the predictor and transmits it to the drone, facilitating real-time adjustments to ensure stability.

**GA and Predictor Setting.** We implement the GA optimizer with *Geatpy*. The population size  $NP$  is set to 40, the maximum number of evolutionary generations  $G_{max}$  is set

<sup>1</sup><https://mavlink.io/en/services/command.html>

to 80, its evolutionary stagnation judgment threshold is 0.1, and the scaling factor  $F$  is 0.4. The segment size set  $m = 30$ . Our TCN predictor consists of a TCN Cell, a dropout layer (on 0.1 drop rate), a Dense layer with ReLU activation, and one fully connected output layer.

**Parameter Selection.** In theory, our solution can support a larger number of parameters. However, the unlimited escalation of parameters necessitates the deployment of additional drones for data collection. Considering the constraints of experimental costs, we manually chose some example parameters that directly impact flight states, such as angular position and speed. Based on the control parameter descriptions provided by the manufacturers, 20 parameters from ArduPilot and 14 parameters from PX4 are chosen. Table I demonstrates the parameters we chose in ArduPilot configuration experiments. In these parameters, PSC\* are advanced parameters to control the velocity gain, ATC\_ANG\* influence the gain of angle controller, ATC\_RAT\* influence the gain of rate controller, and WPNAV\* influence the flight related to missions. Table II demonstrates the parameters we chose in PX4 configuration experiment. In these parameters, MC\*\_P are proportional gain for angle gain, MPC\_XY/Z\_P are proportional gain for horizontal/vertical position error, MC\*\_RATE\_P are proportional rate gain, MPC\_TILTMAX\_AIR are maximum tilt angle in air, MIS\*\_ERR are max \* angle error in degrees needed for waypoint heading acceptance, MPC\_Z\_VEL\_MAX\_DN/UP are maximum descent/ascent velocity.

TABLE I: Parameters of ArduPilot.

Parameter	Range	Default
PSC_VELXY_P	[0.10, 6.00]	2.0
PSC_VELXY_I	[0.02, 1.00]	1.0
PSC_VELXY_D	[0.00, 1.00]	0.5
PSC_ACCZ_P	[0.20, 1.50]	0.5
PSC_ACCZ_I	[0.00, 3.00]	1.0
ATC_ANG_RLL_P	[3.00, 12.0]	4.5
ATC_RAT_RLL_P	[0.01, 0.50]	0.135
ATC_RAT_PIT_I	[0.01, 2.00]	0.135
ATC_RAT_YAW_D	[0.00, 0.02]	0
WPNAV_SPEED	[20, 2000]	500
WPNAV_ACCEL	[50, 500]	100
ANGLE_MAX	[1000, 8000]	4500
ATC_RAT_RLL_D	[0.00, 0.05]	0.0036
ATC_ANG_PIT_P	[0.00, 12.0]	4.5
ATC_RAT_PIT_P	[0.01, 0.50]	0.135
ATC_RAT_PIT_I	[0.01, 2.0]	0.135
ATC_RAT_PIT_D	[0.00, 0.05]	0.0036
ATC_ANG_YAW_P	[3.00, 12.0]	4.5
ATC_RAT_YAW_P	[0.10, 2.50]	0.18
ATC_RAT_YAW_I	[0.01, 1.00]	0.018

### B. Predictor Performance

We assessed the performance of the state predictor based on its ability to forecast state changes when provided with current flight data accurately. The experiment initially establishes the base dataset through multiple flights with varying configurations. Using a multi-threaded implementation (6 threads), we spent about 7 hours to separately collect 969,838 and 327,617 flight data from ArduPilot and PX4 by simulators, where 90% is for training and 10% serves as the test data for

TABLE II: Parameters of PX4.

Parameter	Range	Default
MC_ROLL_P	[0.00, 12.0]	6.5
MC_PITCH_P	[0.00, 12.0]	6.5
MC_YAW_P	[0.00, 5.0]	2.8
MC_YAW_WEIGHT	[0.00, 1.00]	0.4
MPC_XY_P	[0.00, 2.00]	0.9
MPC_Z_P	[0.00, 1.50]	1.0
MC_PITCHRATE_P	[0.01 0.60]	0.15
MC_ROLLRATE_P	[0.01, 0.50]	0.15
MC_YAWRATE_P	[0.00, 0.60]	0.2
MPC_TILTMAX_AIR	[20.0, 89.0]	45.0
MIS_YAW_ERR	[0.00, 90.0]	12.0
MPC_Z_VEL_MAX_DN	[0.5, 4.0]	1.0
MPC_Z_VEL_MAX_UP	[0.5, 8.0]	3.0
MPC_TKO_SPEED	[1.0, 5.0]	1.5

performance. The accuracy outcomes of the predictions are presented in Table III. The experimental results demonstrate that our state predictor consistently achieves an accuracy of over 96%. We selected the best performance predictors for subsequent experiments, i.e.,  $h = 4$  for ArduPilot and  $h = 5$  for PX4. The predictor is specific to the control system used. Vehicles that share the same flight control system and physical frame (e.g., identical four-motor configurations) can utilize the same trained model. However, the accuracy of the model decreases significantly when applied to different flight control systems.

TABLE III: Predictor accuracy trained with different input lengths.

Input $h$	2	3	4	5	6
Ardupilot	96.98%	97.14%	97.18%	96.65%	97.14%
PX4	96.12%	96.43%	96.95%	97.45 %	97.26%

### C. Detection Accuracy

To evaluate the detection effectiveness of CONFIX, we collected data to calculate the threshold values in Sec. III-D and evaluate the performance.

1) *Threshold Calculation:* To choose the appropriate  $TH$  value, we analyze experimental data comprising stable and unstable segments. Stable segments are randomly selected from incident-free historical segments. To acquire the most representative pattern data, unstable segments are extracted from the state data recorded three seconds prior to the incidents. Specifically, we launched 1,000 flight missions containing 500 stable flights and 500 incidents on ArduPilot and PX4, respectively. In terms of the incidents, for ArduPilot, we collected data about 158 Trajectory Offset, 191 Thrust Loss, 142 Flight Freeze, and 9 Crash; for PX4, we collected data about 316 Trajectory Offset, 173 Flight Freeze, and 11 Crash. Noted that PX4 does not support thrust loss warnings. Then, we calculate each set's accumulated deviation and determine the threshold by taking the median value between their average accumulated deviation. Table IV presents the minimum and maximum accumulated deviations observed in both stable and unstable segments, along with the corresponding threshold value.

TABLE IV: Accumulated deviation of different segment.

Program	Type	Min	Max	Average	TH
Ardupilot	Stable	1.12	3.61	1.89	2.83
	Unstable	1.22	12.15	3.78	
PX4	Stable	0.44	1.46	0.77	2.06
	Unstable	0.41	15.45	4.18	

2) *Unstable Segment Detection*: To evaluate the reasonability of the established threshold, we gathered 960 stable segments and 998 unstable segments from ArduPilot, 931 stable segments, and 820 unstable segments from PX4. If the segment's DTV surpassed the threshold  $TH$ , CONFIX classified it as unstable. The detection outcomes are presented in Table V. CONFIX successfully identified 948 out of 998 ArduPilot unstable segments, achieving a F1 of 87.13%; and 780 out of 820 PX4 unstable segments, achieving a F1 of 86.61%. While the precision achieved by CONFIX may not be sufficiently high, reaching 80.48% and 79.51%, it does not significantly impact the drone. Due to eliminating negative experiences (severe incidents) during the training of the state predictor, the rectification resulting from false positives will not bring a negative impact.

TABLE V: Performance of unstable detection.

	Report	Truth	Precision	Recall	F1
Ardupilot	1,178	948	80.48%	94.99%	87.13%
PX4	981	780	79.51%	95.12%	86.61%

After manually reviewing some false detection data, we identified two specific cases our detection mechanism could not effectively cover. One case is when the drone's body remains stable but experiences continuous minor offset. In this scenario, the flight may appear stable initially, but the accumulation of small offset eventually triggers the trajectory offset check. Another extreme case where our detection mechanism falls short is when the drone moves at a low speed. In such instances, the flight states may not be perceived as unstable, but due to the small movements, they can be incorrectly identified as freeze flights over an extended period. Although CONFIX has not been able to detect such cases, the drone maintains a relatively stable flight in these cases, giving the users enough time to notice and solve them.

#### D. Rectification Performance

As CONFIX is a real-time rectification system, we evaluated its performance by testing the rectifications' success rate when facing instability caused by range specification bugs.

1) *Test Configuration Generation*: When devising the test scenarios for CONFIX, we utilized *ICSearcher* [3], a fuzzing technique capable of searching incorrect configurations that lead to range specification bugs. Table VI enumerates the total number of incorrect configurations searched by *ICSearcher*, along with the incidents triggered by these configurations. The table shows that the number of modified parameters is proportional to the number of incorrect configurations *ICSearcher*

can find. PX4 does not report thrust loss warnings, so our experiment does not consider it.

TABLE VI: Configuration search result of *ICSearcher*.

	Num#	Total Search	Traj* Offset	Thrust Loss	Flight Freeze	Crash
Ardu*	2	32	4	0	28	0
	4	149	9	10	129	1
	8	163	7	10	144	2
	12	340	19	9	310	2
	16	805	100	308	393	4
	20	1511	910	518	70	13
PX4	2	16	16	-	0	0
	4	26	19	-	0	0
	8	122	70	-	47	5
	12	133	61	-	56	16
	14	258	155	-	82	21

**Num#** is number of parameters the configuration construed with. **Ardu\*** is **ArduPilot** control program. **Traj\*** is **Trajectory**.

2) *Performance in Rectification*: We conducted flight missions (AVC2013 [22]) with these incorrect configurations to assess whether CONFIX can effectively prevent incidents. In advance, we first created four labels to evaluate the analysis results of CONFIX:

- **Rectified**: CONFIX rectifies the instability by uploading a configuration, and thus, the drone finally completes the flight mission successfully (i.e., landed at the intended destination).
- **Failed**: CONFIX successfully uploads a configuration in time; however, the unstable trend persists, leading to an incident.
- **Missed**: CONFIX does not report any instability before the incident occurs.
- **Interrupted**: CONFIX reports instability but fails to generate a configuration to prevent the incident.

TABLE VII: Rectifications result of CONFIX.

	Num*	Rectified (Rate)	Missed	Inter*	Failed
Ardu*	2	31 (96.8%)	0	0	1
	4	137 (91.9%)	0	0	12
	8	148 (90.7%)	0	0	15
	12	279 (82.1%)	0	0	61
	16	403 (50.1%)	1	26	375
	20	376 (25.3%)	29	82	1,024
PX4	2	14 (81.3%)	0	0	2
	4	19 (73.1%)	0	0	7
	8	72 (60.0%)	0	2	46
	12	55 (42.9%)	5	13	60
	14	100 (41.8%)	19	34	105

\* **Num\*** is number of parameters the configuration construed with. **Ardu\*** is **ArduPilot** control program. **Inter\*** is number of **Interrupted**.

Table VII illustrates the test results. In summary, we found that:

- **ArduPilot**: CONFIX successfully achieved a high success rate of flight rectification (i.e., over 90%) when the number of analyzed parameters is less than or equal to eight. When the involved parameters increased to 16, rectification success

rates reduced, where CONFIX failed to rectify the drone in 375 flight missions. For 26 missions, it successfully detected instability but could not rectify it in time. Reduced success rate arises from two aspects: (1) Incorrect configurations with excessive internal parameter modifications often result in more severe instabilities, quickly causing incidents that the system cannot rectify within a short time frame. (2) The escalating quantity of parameters also results in an exponential expansion of the exploration space required by the optimizer, which complicates the search for the most suitable configuration.

- **PX4:** The performance of CONFIX is similar for PX4. Specifically, within eight parameters, the success rate of rectifications is higher than 60%. By comparing the programs of ArduPilot and PX4, we found that the flight control system of PX4 is more complicated, which makes the configuration prediction more difficult. Consequently, the success rate of rectifications performed on PX4 is lower than the performance on ArduPilot. Besides, for the parameters that fulfill similar functions, the value range provided by the PX4 manufacturer is more extensive than the one set by the ArduPilot manufacturer; thus, CONFIX requires more time to test the parameter values. For example, parameters `ATC_ANG_PIT_P` of *Ardupilot* and `MC_PITCH_P` of PX4 are designed for converting the discrepancy between the desired and the actual pitch angle into a desired pitch rate. Differently, the range of `ATC_ANG_PIT_P` is [3, 12], and the range of `MC_PITCH_P` is (0, 12].
- **Incident Discussion:** By analyzing the details for each type of incident (Table VIII), CONFIX performed stable when rectifying the instability that led to **Trajectory Offset**, achieving a success rate over 78% if the modified parameters are no more than 12. Nonetheless, CONFIX performed worse on PX4 when proceeding with the instability leading to **Flight Freeze**. Through our manual inspection, we found the major feature that affects performance is the selected parameters. Some parameters included in ArduPilot (e.g., `PSC_ACCZ_*`) directly manage the acceleration of the drone, which can adjust the flight state easily. For the resting two incident types, CONFIX performed unsteadily. We observed that the major interrupted cases were caused by rectifying these two incidents. By reproducing the rectification, we found that the modification of parameters `ATC_RAT_*` in ArduPilot and `MC_*RATE_*` in PX4 complicate the attitude estimation of the control logic.

3) *Time Requirement:* We recorded the time required for CONFIX to detect each segment and the time to generate rectification in the experiment. Table IX shows the average result for those times. We observe that the time consumption is within 0.29 per second for detecting instability. CONFIX can provide timely rectification within an average of 3.61 seconds. Since CONFIX operates independently of the flight control system, it does not affect the consumption of drones. As for communication (capture data or upload configuration) consumption, virtual experiments took less than 1ms and real experiments took 32ms on average.

## E. Workflow Component Performance

To verify the impact of each component in CONFIX on the entire performance, we evaluated the effectiveness of each component to pursue potential improvement.

1) *Assessment of Training Dataset:* In our design of CONFIX, we trained the state predictor without utilizing any mission data that resulted in severe incidents to achieve accuracy detection and rectification. To verify such a statement, we conducted an experiment that trained a hybrid predictor using data from stable flights and incidents. As CONFIX performed similarly on ArduPilot and PX4, we only applied the rectification experiments on ArduPilot.

We additionally process flight data that appear before the incident and add 126, 513 data entries in the training dataset. Subsequently, we embedded the hybrid predictor into CONFIX to test the success rate for comparison. The outcomes are presented in Table X, where “Without Unstable” represents the state predictor trained using stable flight data, and “With Unstable” refers to the hybrid predictor. Intuitively, the success rate experienced a decrease upon integration with the hybrid predictor. While the performance appeared comparable as the adjusted parameters reached a value of 16, the success rate of CONFIX utilizing the original predictor remained superior to that of the hybrid predictor when the modified parameters were below 12. The presence of unstable logs may hinder the effectiveness of our rectification measures.

2) *Accumulated Deviation:* CONFIX conducted accumulated deviations to determine whether a segment of states will lead to an incident. To verify whether such an approach could describe the unstable trend before incidents, we verified two types of flight states (i.e., stable and instability that ultimately trigger irregular vibration and crash). We leveraged the state predictor to generate referenced states to calculate the accumulated deviations.

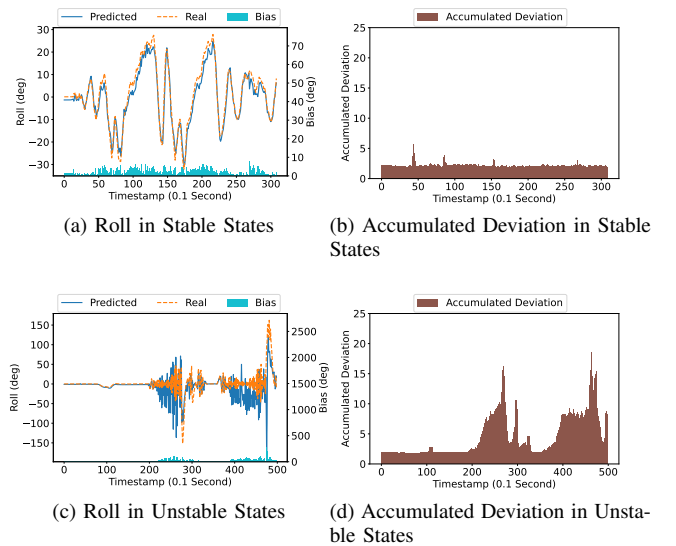


Fig. 3: Roll bias and accumulated deviation in different situations.

Take the roll angle change in states as an example, Fig-

TABLE VIII: Distribution of Rectification results.

	Num#	Distribution {Rectified - Total — Success Rate }							
		Trajectory Offset		Thrust Loss		Flight Freeze		Crash	
Ardupilot	2	4 - 4	100%	0 - 0	-	27 - 28	96.4%	0 - 0	-
	4	8 - 9	88.8%	0 - 0	-	128 - 139	92.1%	1 - 1	100%
	8	7 - 7	100%	5 - 10	50.0%	135 - 144	93.7%	1 - 2	50.0%
	12	16 - 19	84.2%	4 - 9	44.4%	258 - 310	83.2%	2 - 2	100%
	16	50 - 100	50.0%	98 - 308	31.8%	251 - 393	63.8%	4 - 4	100%
	20	255 - 910	28.0%	92 - 518	17.8%	24 - 70	34.3%	5 - 13	38.4%
PX4	2	13 - 16	81.3%	-	-	0 - 0	-	0 - 0	-
	4	19 - 26	73.1%	-	-	0 - 0	-	0 - 0	-
	8	55 - 70	78.5%	-	-	7 - 47	14.8%	3 - 5	60.0%
	12	31 - 61	50.8%	-	-	19 - 56	33.9%	5 - 16	31.25%
	14	72 - 155	46.4%	-	-	26 - 82	31.7%	2 - 21	9.5%

\* Num# is the number of parameters the configuration construed with.

TABLE IX: Average time cost of CONFIX.

Number	2	4	8	12	16	20
Detection (s)	0.231	0.231	0.235	0.262	0.271	0.282
Rectification (s)	2.28	2.32	2.41	2.85	3.32	3.61

TABLE X: Comparison of success rate with different predictors.

Num#	2	4	8	12	16	20
WOU	96.8%	91.9%	90.7%	82.1%	50.1%	25.3%
WU	93.75%	88.41%	85.7%	71.7%	49.8%	23.6%

\* Num# is number of parameters the configuration construed with.  
**WOU** means predictor is trained without unstable data. **WU** means predictor is trained with unstable data.

ure 3 demonstrates the predicted reference and real roll, and accumulated deviations. Figure 3a clearly shows the bias between the predicted reference and real roll. When proceeding with the stable states, there is no big bias between them. The corresponding accumulated deviations in Figure 3b remained stable and kept within a small range (i.e., less than three). Nevertheless, within the context of the unstable state (Figure 3c), the drone exhibited notable bias from predicted reference states, leading to substantial accumulation deviations (Figure 3d). Following such a pattern, CONFIX can identify stable and unstable states effectively.

3) *Population Size and Maximum Iteration*: In light of the potential impact of population size and maximum iteration on the optimization of rectification, we conducted experiments to validate their respective influences. Experiments adopt 8 Ardupilot parameters in Table VII to test the rectification capability. The distinguishing factor was the application of varying population sizes for the initialization of the optimizer, i.e., 20, 40, 80, 160, 320. The rectified success rate and time consumption results are exhibited in Figure 4a, where the blue bar is the average rectification time, and the line is the rectified success rate. It can be observed that there is a slight increase (float around 90%) in the success rate as the population size increases, but with that comes an exponential growth in time consumption. Although some unstable states will not cause damage to the drone in a short period, e.g.,

**Flight freeze**, the rectifications of others, e.g., **Trajectory Offset** and **Thrust Loss**, have high requirements for real-time as their severe and rapid consequences. Therefore, our system adopts a 40 population size to balance the success rate and time consumption.

Furthermore, the maximum number of iterations mainly influences the optimization efficiency. Therefore, we conducted additional tests to assess its impact. Similarly, experiments adopt 8 Ardupilot parameters and apply different maximum numbers of iterations, i.e., 40, 80, 160, 320, 400. Figure 4b demonstrates the result. Unlike the population size, augmenting the maximum number of iterations did not result in significant time consumption or a substantial increase in the success rate. This implies that, within our optimization process, the iterations mostly operated within the prescribed limit. In summary, the maximum number of iterations does not influence the performance of CONFIX.

#### F. Cases Study

We selected a representative rectified example to show the process and characteristics of CONFIX, which launched test cases in both reality and simulation. The cases show two examples that separately cause **Thrust Loss** and **Flight Freeze**, where **Thrust Loss** is tested in simulation. Figure 5 shows its state change about angular roll, pitch, and yaw during the flight.

In a previous flight experiment for this configuration, it was impossible to avoid the loss of control when the thrust loss message was raised. With the rectification by CONFIX, the drone successfully finished the mission shown in the video<sup>2</sup>. Following takeoff, the drone experienced initial instability in its attitude, leading to an uncontrollable rotation. CONFIX promptly detected this issue and triggered a rectification. The new configuration was subsequently uploaded to the drone, effectively addressing the problem. Figure 5a shows a part of state change. It can be observed from the figure that with an inappropriate configuration loaded (red area), the desired states of the drone have a large deviation from the achieved state. In particular, the change in yaw angle shows that the drone has rotational movements. When the new configuration is updated

<sup>2</sup><https://youtu.be/NbS1hwS40wg>

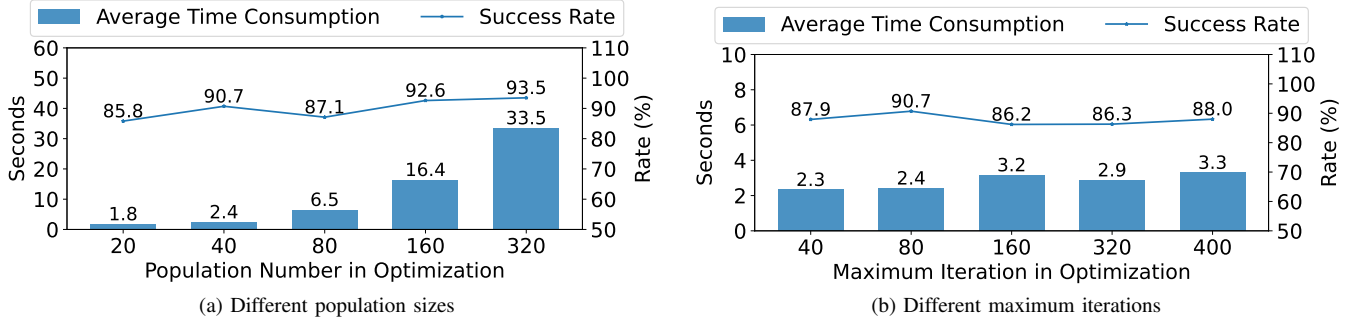


Fig. 4: Performance of CONFIX in different conditions.

(green area), the deviation becomes smaller, thus making the drone stable, which means the rectification of CONFIX for the current state was successful.

In an actual experiment for this **Freeze Flight** incident, the drone is stuck at a waypoint and cannot move on. The drone was stuck at a mission waypoint with a slight vibration. Then, with an updated rectification, the drone stops vibrating and continues its forward mission. Similarly, Figure 5b shows the state change of a real drone. After our rectification is uploaded, the drone returns to a stable flight and finally completes the mission.

## V. DISCUSSION

CONFIX achieved acceptable performance. In this section, we discuss some scenarios or factors that may limit the performance of CONFIX.

### Increasing the parameters involved in the configuration.

To accommodate additional parameters, CONFIX must undertake further experiments to gather training data, thereby ensuring the predictor’s accuracy. However, as the number of parameters escalates, the volume of required experimental data increases exponentially. This process will be highly time-consuming, and it becomes challenging to guarantee the diversity of the data, ultimately impacting the predictor’s accuracy. Furthermore, as the number of parameters in the configuration increases, the GA process exponentially expands the exploration space demanded by the optimizer, making searching for the optimal configuration more complex and time-consuming. This increases the occurrence of “Interrupted” and “Failed” outcomes. Nevertheless, from the user’s perspective, unless confronted with exceptionally challenging tasks (such as navigating highly complex environments), users generally do not adjust many parameters simultaneously. From the attacker’s perspective, the cost of generating configurations also increases exponentially with the number of parameters involved. Consequently, CONFIX is less likely to undergo simultaneous adjustments of multiple parameters.

**Influence of predictor.** CONFIX relies on the predictor’s outcomes to establish the rectification configuration. In most instances, this configuration serves to alleviate instability. However, certain inaccurate forecasts may exacerbate instability. For instance, the current configuration may cause the drone

to stray from its intended course. When the system identifies an instability and formulates a new configuration, it may prove inadequate, leading to increased oscillation and potentially resulting in the drone’s crash. Due to the variability in prediction accuracy, this situation cannot be entirely mitigated.

**Practical application.** In practical applications, CONFIX requires developers or users to compile a diverse array of flight data to generate the predictor. Conducting numerous experiments is challenging, as these trials may compromise the drone’s structural integrity, potentially leading to crashes. However, this effort is largely a one-time investment.

**Further works.** As the number of modified parameters increases, the success rate of CONFIX repairs will consequently decline. Subsequent efforts could employ more efficient search methods to reduce the duration of the GA process, thereby improving the success rate of repairs. Moreover, the current unstable detection scheme employs a statistical threshold approach, which lacks sufficient accuracy for complex scenarios such as flight attitudes. Future work should consider utilizing more advanced techniques to enhance the precision of detection.

## VI. RELATED WORK

Many approaches have been developed for drone security, mainly focusing on three aspects: identifying vulnerabilities in drone control systems, potential attack detection, and recovery.

### A. Vulnerability in Drone Program

From the software design perspective, drone programs have some system bugs, which may lead to unpredictable results or be exploited when attacking drones. Wang et al. [25] conducted a large-scale empirical study and summarized common bug patterns and repair strategies by analyzing drone-specific bugs and related root causes by investigating drone error reports, source code, patches, and historical data. Liang et al. [26] conducted an empirical study to understand the safety-critical concerns of software from a bottom-up developer-in-the-field perspective. They focused on the runtime checking of the Bounding Function (BF) in control software, classified BF instances with static methods, and dynamically evaluated the impact of BF usage through differential methods. Dash et al. [8] demonstrated the vulnerabilities in control-based intrusion detection and proposed an automated algorithm that

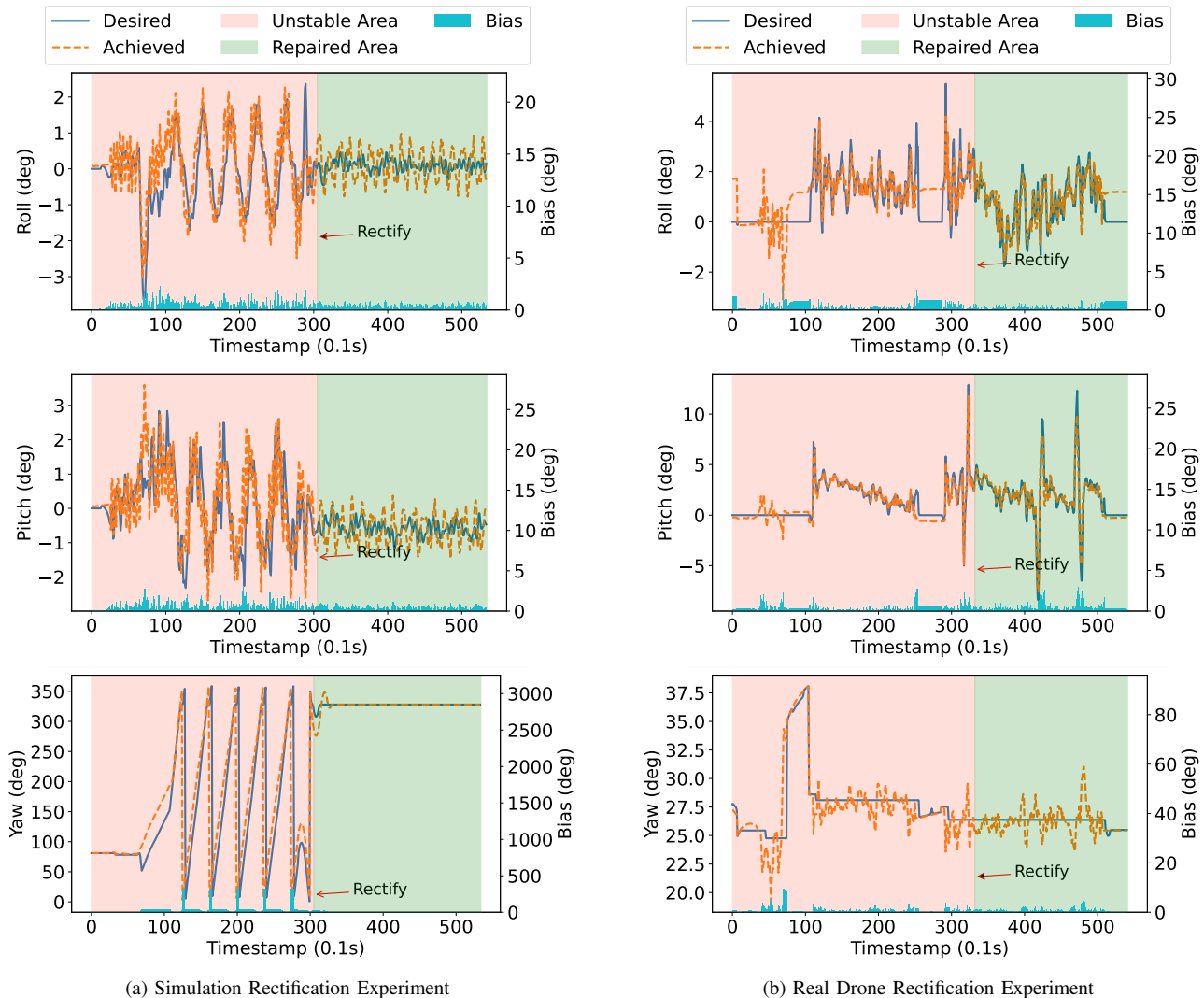


Fig. 5: Flight state change about angular roll, pitch, and yaw. The solid curve denotes the degree desired by the controller. The dotted curve denotes the degree achieved by the drone. The histogram at the bottom (the cyan bar) indicates the bias between the two curves. The red area indicates that the incorrect configuration was set. The green area indicates that rectification was set.

can attack robotic vehicles without knowing much about them. *PGFUZZ* [27] is a policy-guided fuzzing framework that verifies that the drone program follows the defined security and functional policies. Choi et al. [28] proposed a combination of program analysis, vehicle modeling, and search-based testing to identify a specific type of vulnerability called Cyber-Physical Inconsistency. This vulnerability may cause the safety checks to fail to report real physical accidents or false alarms.

### B. Potential Attack Detection

For external attacks, detection schemes have been proposed to protect drones. Fei et al. [29] proposed a framework termed *BlueBox* that can be retrofitted to the drone. Then, a binary reverse analysis is executed on the drone source code to detect cyber-physical attacks without changing the original control system. Choi et al. [30] proposed a vetting system that

detects control system misbehaviors by observing (simulating) the physical operations of the robotic vehicle (RV) based on the control model. It uses control invariant (CI) to detect common attacks by drones. Jansen et al. [31] designed *Crowd-GPS-Sec*, which monitors the GPS location ads regularly broadcast by drones for air traffic surveillance through the detection and evaluation of the actual data from crowdsourced sensors and analyzes the content and arrival time of these surveillance messages received by different sensors on the ground. Sindhvani et al. [32] used machine learning methods and models to generate a novel anomaly detection framework by training on the drones' flight parameters of sensors. Based on three components, namely an attack detection algorithm, a counterattack model that includes a powerful stealth attacker, and an implementation that displays minimal performance overhead in actual hardware, Quinonez et al. [33] introduced *SAVIOR*, a stealth attack system with Physics-Based Attack

Detection (PBAD) robust physical provide security for Autonomous Vehicles.

### C. Repair and Restoration

Automated methods have recently been proposed for drone protection from attacks identified by past research. *MAYDAY* [34] can be deployed in drones for monitoring flight records and tracing bugs and causes of drone accidents. Choi et al. [35] proposed a technique to recover RVs from attacks by alternating with software sensors. When an attack on a physical sensor is detected, the sensor is isolated and replaced by a software sensor with a quick recovery from the damaged internal state. *PID-Piper* [36] is a forward feedback-based ML model proposed to recover RVs from physical attacks, which avoids over-compensation and high colinearity between input parameters. *PGPATCH* [37] proposes a policy-guided program repair framework for RV control systems to identify patch classifications for a given logical error and generate patches without human intervention. Li et al. [38] integrates adaptive control into *ArduPilot*, to address state-dependent alterations and unstructured uncertainties like unmodelled dynamics and environmental disturbances. The adaptation features are incorporated without changing the original architecture, enabling users to operate the autopilot as usual. *MRS ArduPilot* [39] is an adaptive autopilot designed to enhance the flexibility and performance of drones across various platforms while addressing uncertainties in operation. By integrating model reference stabilization (MRS) into the existing *ArduPilot* architecture. The approach bridges the gap between autopilot developers and researchers in adaptive control.

In contrast, our solution for range specification bugs adopts state detection to identify the unstable trends before incidents and an online optimization with a learning-based state predictor to create a better configuration to rectify the drone.

## VII. CONCLUSION

When a drone configuration is incorrectly set because of user mistakes, worst-case scenarios may disrupt drone flight. We propose a rectification system that efficiently and effectively detects the instability caused by range specification bugs while providing appropriate configuration. *CONFIX* uses the accumulated deviation between the predicted desired state and ground-truth state to detect when flight states are unstable and leverages an optimizer to search for a suitable configuration. The experimental results show that our system has good repair performance.

## REFERENCES

- [1] 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 *Proceedings of the 28th USENIX Security Symposium (USENIX Security)*. USENIX Association, Aug. 2019, pp. 425–442.
- [2] 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 IEEE/ACM 44th International Conference on Software Engineering (ICSE)*. IEEE, 2022, pp. 462–473.
- [3] R. Han, S. Ma, J. Li, S. Nepal, D. Lo, Z. Ma, and J. Ma, "Range specification bug detection in flight control system through fuzzing," *IEEE Transactions on Software Engineering*, 2024.
- [4] Y. Son, H. Shin, D. Kim, Y. Park, J. Noh, K. Choi, J. Choi, and Y. Kim, "Rocking drones with intentional sound noise on gyroscopic sensors," in *Proceedings of the 24th USENIX Security Symposium (USENIX Security 15)*, 2015, pp. 881–896.
- [5] T. Trippel, O. Weisse, W. Xu, P. Honeyman, and K. Fu, "WALNUT: Waging doubt on the integrity of MEMS accelerometers with acoustic injection attacks," in *Proceedings of the 2017 IEEE European symposium on security and privacy (EuroS&P)*. IEEE, 2017, pp. 3–18.
- [6] M. S. bin Mohammad Fadilah, V. Balachandran, P. K. K. Loh, and M. W. J. Chua, "DRAT: A Drone Attack Tool for Vulnerability Assessment," in *Proceedings of 20th ACM Conference on Data and Application Security and Privacy, New Orleans, LA, USA, March 16-18, 2020*. ACM, 2020, pp. 153–155.
- [7] C. Zhou, Q. Yan, Y. Shi, and L. Sun, "DoubleStar: Long-Range Attack Towards Depth Estimation based Obstacle Avoidance in Autonomous Systems," in *Proceedings of the 31st USENIX Security Symposium (USENIX Security)*, 2022, pp. 1885–1902.
- [8] P. Dash, M. Karimibiuki, and K. Pattabiraman, "Stealthy attacks against robotic vehicles protected by control-based intrusion detection techniques," *Digital Threats: Research and Practice*, vol. 2, no. 1, pp. 1–25, 2021.
- [9] O. M. Alhawi, M. A. Mustafa, and L. C. Cordiro, "Finding security vulnerabilities in unmanned aerial vehicles using software verification," in *Proceedings of the 2019 International Workshop on Secure Internet of Things (SIOT)*. IEEE, 2019, pp. 1–9.
- [10] M. Taylor, H. Chen, F. Qin, and C. Stewart, "Avis: In-Situ Model Checking for Unmanned Aerial Vehicles," in *Proceedings of the 51st Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, 2021, pp. 471–483.
- [11] S. K. Bhoi, K. K. Jena, S. K. Panda, H. V. Long, R. Kumar, P. Subbulakshmi, and H. B. Jebreen, "An Internet of Things assisted Unmanned Aerial Vehicle based artificial intelligence model for rice pest detection," *Microprocessors and Microsystems*, vol. 80, p. 103607, 2021.
- [12] M. Zeng, Y. Wu, Z. Ye, Y. Xiong, X. Zhang, and L. Zhang, "Fault Localization via Efficient Probabilistic Modeling of Program Semantics," in *Processing of the IEEE/ACM 44th International Conference on Software Engineering (ICSE)*. ACM, 2022, pp. 958–969.
- [13] D. Reid, M. Jahanshahi, and A. Mockus, "The Extent of Orphan Vulnerabilities from Code Reuse in Open Source Software," in *Processing of the 44th IEEE/ACM 44th International Conference on Software Engineering (ICSE)*. ACM, 2022, pp. 2104–2115.

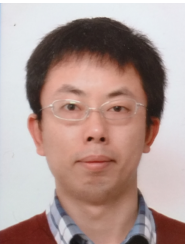
- [14] K. Nguyen and T. Nguyen, “GenTree: Using decision trees to learn interactions for configurable software,” in *Processing of the 43rd IEEE/ACM International Conference on Software Engineering (ICSE)*. IEEE, 2021, pp. 1598–1609.
- [15] X. Li, C. Yang, J. Ma, Y. Liu, and S. Yin, “Energy-efficient side-channel attack countermeasure with awareness and hybrid configuration based on it,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 25, no. 12, pp. 3355–3368, 2017.
- [16] S. Purandare, U. Sinha, M. N. Al Islam, J. Cleland-Huang, and M. B. Cohen, “Self-adaptive mechanisms for misconfigurations in small uncrewed aerial systems,” in *Proceedings of the 18th Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*. IEEE, 2023, pp. 169–180.
- [17] S. Park, Y. Kim, and D. H. Lee, “Scvmon: Data-oriented attack recovery for rvs based on safety-critical variable monitoring,” in *Proceedings of the 26th International Symposium on Research in Attacks, Intrusions and Defenses (RAID)*, 2023, pp. 547–563.
- [18] Y.-M. Kwon, J. Yu, B.-M. Cho, Y. Eun, and K.-J. Park, “Empirical analysis of mavlink protocol vulnerability for attacking unmanned aerial vehicles,” *IEEE Access*, vol. 6, pp. 43 203–43 212, 2018.
- [19] N. M. Rodday, R. d. O. Schmidt, and A. Pras, “Exploring security vulnerabilities of unmanned aerial vehicles,” in *Proceedings of the 16th IEEE/IFIP Network Operations and Management Symposium*. IEEE, 2016, pp. 993–994.
- [20] M. Vanhoef and F. Piessens, “Key reinstallation attacks: Forcing nonce reuse in WPA2,” in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, 2017, pp. 1313–1328.
- [21] S. Bai, J. Z. Kolter, and V. Koltun, “An empirical evaluation of generic convolutional and recurrent networks for sequence modeling,” *arXiv preprint arXiv:1803.01271*, 2018.
- [22] A. Team, “Sparkfun autonomous vehicle competition 2013,” <https://avc.sparkfun.com/2013>.
- [23] K. Fleetwood, “An introduction to differential evolution,” in *Proceedings of Mathematics and Statistics of Complex Systems (MASCOS) One Day Symposium, 26th November, Brisbane, Australia*, 2004, pp. 785–791.
- [24] L. Meier, J. Camacho, B. Godbolt, J. Goppert, L. Heng, M. Lizarraga *et al.*, “Mavlink: Micro air vehicle communication protocol,” 2013.
- [25] D. Wang, S. Li, G. Xiao, Y. Liu, and Y. Sui, “An exploratory study of autopilot software bugs in unmanned aerial vehicles,” in *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE)*, 2021, pp. 20–31.
- [26] X. Liang, J. H. Burns, J. Sanchez, K. Dantu, L. Ziarek, and Y. D. Liu, “Understanding Bounding Functions in Safety-Critical UAV Software,” in *Proceedings of the IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*, 2021, pp. 1311–1322.
- [27] H. Kim, M. O. Ozmen, A. Bianchi, Z. B. Celik, and D. Xu, “PGFUZZ: Policy-Guided Fuzzing for Robotic Vehicles,” in *Proceedings of the the 28th The Network and Distributed System Security Symposium (NDSS)*, 2021.
- [28] H. Choi, S. Kate, Y. Aafer, X. Zhang, and D. Xu, “Cyber-physical inconsistency vulnerability identification for safety checks in robotic vehicles,” in *Proceedings of the 27th ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2020, pp. 263–278.
- [29] F. Fei, Z. Tu, R. Yu, T. Kim, X. Zhang, D. Xu, and X. Deng, “Cross-layer retrofitting of UAVs against cyber-physical attacks,” in *Proceedings of the 17th IEEE International Conference on Robotics and Automation (ICRA)*, 2018, pp. 550–557.
- [30] H. Choi, W.-C. Lee, Y. Aafer, F. Fei, Z. Tu, X. Zhang, D. Xu, and X. Deng, “Detecting attacks against robotic vehicles: A control invariant approach,” in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security (CCS)*. ACM, 2018, pp. 801–816.
- [31] K. Jansen, M. Schäfer, D. Moser, V. Lenders, C. Pöpper, and J. Schmitt, “Crowd-gps-sec: Leveraging crowdsourcing to detect and localize gps spoofing attacks,” in *Proceedings of the 39th IEEE Symposium on Security and Privacy (S&P)*, 2018, pp. 1018–1031.
- [32] V. Sindhvani, H. Sidahmed, K. Choromanski, and B. Jones, “Unsupervised anomaly detection for self-flying delivery drones,” in *Proceedings of the 19th IEEE International Conference on Robotics and Automation (ICRA)*, 2020, pp. 186–192.
- [33] R. Quinonez, J. Giraldo, L. Salazar, E. Bauman, A. Cardenas, and Z. Lin, “SAVIOR: Securing autonomous vehicles with robust physical invariants,” in *Proceedings of the 29th USENIX Security Symposium (USENIX Security)*, 2020, pp. 895–912.
- [34] T. Kim, C. H. Kim, A. Ozen, F. Fei, Z. Tu, X. Zhang, X. Deng, D. J. Tian, and D. Xu, “From Control Model to Program: Investigating Robotic Aerial Vehicle Accidents with MAYDAY,” in *Proceedings of the 29th USENIX Security Symposium (USENIX Security)*, 2020, pp. 913–930.
- [35] H. Choi, S. Kate, Y. Aafer, X. Zhang, and D. Xu, “Software-based realtime recovery from sensor attacks on robotic vehicles,” in *Proceedings the 23rd International Symposium on Research in Attacks, Intrusions and Defenses (RAID)*, 2020, pp. 349–364.
- [36] P. Dash, G. Li, Z. Chen, M. Karimibiuki, and K. Patabiraman, “Pid-piper: Recovering robotic vehicles from physical attacks,” in *Proceedings of the 51st Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, 2021, pp. 26–38.
- [37] H. Kim, M. O. Ozmen, Z. B. Celik, A. Bianchi, and D. Xu, “PGPATCH: Policy-Guided Logic Bug Patching for Robotic Vehicles,” in *Proceedings of the 43rd IEEE Symposium on Security and Privacy (S&P)*, 2022.
- [38] P. Li, D. Liu, X. Xia, and S. Baldi, “Embedding adaptive features in the ardupilot control architecture for unmanned aerial vehicles,” in *2022 IEEE 61st Conference*

on *Decision and Control (CDC)*. IEEE, 2022, pp. 3773–3780.

- [39] D. Sun, P. Li, X. Xia, D. Liu, and S. Baldi, “Mrs ardupilot: An adaptive ardupilot architecture based on model reference stabilization,” in *2024 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, 2024, pp. 2723–2728.



**Ruidong Han** received his B.Eng. degree in Computer Science from Northwest A&F University in 2018, China. He received his Ph.D. degree in the School of Cyber Engineering, at Xidian University. His research interests include IoT security, trusted computing, and intelligent system security.



**Juanru Li** is the director with the Group of Software Security In Progress (G.O.S.S.I.P). His research interests cover a wide array of systems and software security problems, with an emphasis on designing practical solutions and techniques that help computer systems stay secure.



**Zhuo Ma** received the Ph.D. degree in computer architecture from Xidian University, Xi’an, China, in 2010. Now, he is an associate professor with the School of Cyber Engineering, at Xidian University. His research interests include cryptography, machine learning in cyber security, and Internet of Things security.



**David Lo** (Fellow, IEEE) received the PhD degree in computer science from National University of Singapore, Singapore, in 2008. He is currently fellow of Automated Software Engineering, ACM distinguished member, and professor of Computer Science at Singapore Management University. His research interests include the intersection of software engineering and data science, encompassing socio-technical aspects and analysis of different kinds of software artefacts, with the goal of improving software quality and developer productivity. His work

has been published in premier and major conferences and journals in the area of software engineering, AI, and cybersecurity.



**Arash Shaghghi** is a Senior Lecturer in Cyber Security at the School of Computer Science and Engineering (CSE) of UNSW Sydney. He is a member of the UNSW Institute for Cyber Security (IFCYBER). He completed his PhD in Computer Science and Engineering at UNSW Sydney, MSc in Information Security at University College London (UCL), and BSc at Heriot-Watt University



**Jianfeng** received a B.S. degree in computer science from Shaanxi Normal University in 1982, an M.S. degree in computer science from Xidian University in 1992, and a Ph.D. degree in computer science from Xidian University in 1995. Currently, he is a Professor at the School of Computer Science and Technology, at Xidian University. He has published over 150 journal and conference papers. His research interests include information security, cryptography, and network security.



**Siqi Ma** received her Ph.D. degree in information systems from Singapore Management University in 2018. She is currently a lecturer in the School of Engineering and Information Technology at the University of New South Wales. Her research interests are mobile security, web security, and IoT security.