Real-time Rectifying Flight Control Misconfiguration Using Intelligent Agent

RUIDONG HAN, Singapore Management University, Singapore, Singapore and Xidian University, Xian, China
SHANGZHI XU, The University of New South Wales, Sydney, Australia
JUANRU LI, Feiyu Security, Shanghai, China
ELISA BERTINO, Purdue University, West Lafayette, IN, USA
DAVID LO, Singapore Management University, Singapore, Singapore
JIANFENG MA, Xidian University, Xian, China
SIQI MA, The University of New South Wales, Sydney, Australia

Configurations are supported by most flight control systems, allowing users to control a flying drone adapted to complexities such as environmental changes or mission alterations. Such an advanced functionality also introduces a significant problem—misconfiguration settings. It may cause drone instability, threaten drone safety, and potentially lead to substantial financial loss. However, detecting and rectifying misconfigurations across different flight control systems is challenging because (1) (mis)configuration-related code snippets might be syntactically correct and thus hard to identify through traditional code analysis; (2) the response to each configuration varies under different flying scenarios.

In this article, we propose and implement a novel rectification approach, Nyctea, to detect instability caused by misconfigurations and conduct an on-the-fly rectification. Nyctea first continuously inspects state changes over consecutive time intervals and calculates the overall deviations to determine whether a drone is in a transition of instability to control loss. When a potential instability is reported, Nyctea instantly invokes a pre-trained intelligent agent to automatically generate proper configurations and then re-configure the drone against entering a state of loss of control. This process of reconfiguration is conducted iteratively until the instability is eliminated. We integrated Nyctea with the widely used flight control system, *Ardupilot* and *PX4*. The simulated and practical experiment results showed that Nyctea successfully eliminates instabilities caused by 85% of misconfigurations. For each misconfiguration, Nyctea averagely generated 4 to 5 configurations to achieve a successful rectification.

CCS Concepts: • **Software and its engineering → Software maintenance tools**; • **Hardware → Safety critical systems**;

Ruidong Han completed this work at Xidian University.

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

Authors' Contact Information: Ruidong Han (corresponding author), Singapore Management University, Singapore, Singapore and Xidian University, Xian, China; e-mail: ruidonghanxd@outlook.com; Shangzhi Xu, The University of New South Wales, Sydney, Australia; e-mail: mxu490469@gmail.com; Juanru Li, Feiyu Security, Shanghai, China; e-mail: mail@lijuanru.com; Elisa Bertino, Purdue University, West Lafayette, IN, USA; e-mail: bertino@purdue.edu; David Lo, Singapore Management University, Singapore, Singapore; e-mail: davidlo@smu.edu.sg; JianFeng Ma, Xidian University, Xian, China; e-mail: jfma@mail.xidian.edu.cn; Siqi Ma, The University of New South Wales, Sydney, Australia; e-mail: siqi.ma@unsw.edu.au.

ACM 1557-7392/2025/4-ART92
https://doi.org/10.1145/3702994

Additional Key Words and Phrases: Drone security, instability rectification, reinforcement learning

ACM Reference format:
Ruidong Han, Shangzhi Xu, Juanru Li, Elisa Bertino, David Lo, JianFeng Ma, and Siqi Ma. 2025. Real-time Rectifying Flight Control Misconfiguration Using Intelligent Agent. *ACM Trans. Softw. Eng. Methodol.* 34, 4, Article 92 (April 2025), 29 pages.
https://doi.org/10.1145/3702994

1 Introduction

Drones (also known as unmanned aerial vehicles) have become widely used for personal, commercial, and military purposes (e.g., photo shooting, surveillance, and delivery) [4, 8, 23, 24, 50]. Unlike other smart devices (e.g., smartphones and IoT devices) controlled by humans, drones are usually controlled autonomously by flight control systems to handle various flight missions and adapt to environmental changes.

To better accommodate unpredictable changes and offer users a degree of flexibility in controlling drones, manufacturers design a configuration setting feature comprised of hundreds of control parameters. By adjusting control parameter values, users can operate a drone remotely according to their preferences and the current flight states. In addition, manufacturers also provide official manuals introducing the suggested value range of each control parameter. With the assistance of official manuals and well-implemented control parameters in the flight control system, users ideally should be capable of achieving various goals and preventing potential threats by configuring control parameters as needed. However, in practice, the intricate variations in environmental changes and the complexity of setting control parameters make proper configuration settings a difficult task [14]. Some studies [22, 36, 58] have shown numerous instances of misconfigurations, resulting in adverse incidents (e.g., trajectory deviations, flight hovering, flight freezing, crashes), even when users select parameter values from the suggested range.

In contrast to other bugs that may arise from incorrect code implementations (e.g., code logic execution bugs [9, 27, 32, 33]) or malicious attacks (e.g., sensor attacks [11, 25, 26, 57]), misconfigurations may be triggered under two scenarios: (1) *improper configurations*—that configurations consisting of multiple improper/incorrect parameter values impact the system and lead directly to instability, from which the issue originates from the configuration itself. (2) *inappropriate settings*—the configurations are proper/correct, yet inappropriate for the current flying conditions (e.g., environment and mission), leading to instability, from which the issue arises due to external flying conditions.

These misconfigurations can not be identified through code analysis because functions related to control parameters may be correctly implemented at the code level [31]. Hence, the approaches for identifying defect codes do not apply when identifying misconfigurations. Apart from that, control parameters defined in different flight control systems are inconsistent in achieving different capabilities, rendering the (mis)configuration study even more challenging and unscalable. Instead of analyzing the implementation code, *LGDFuzzer* [22], *ICSearcher* [21], and *RVFuzzer* [36] leverage either fuzzing or a combination of machine learning and fuzzing approaches to search potential misconfigurations automatically and further verify whether configurations will lead to unstable flight states via a simulator. However, the solution they proposed entails restricting the range of user-configurable parameters, which not only diminishes the drone's flexibility but also fails to eliminate the misconfiguration that continues to exist within that range. The other post-incident detection approaches [11, 35, 52] can only be confirmed after instability or control loss has occurred, but fail

to prevent the occurrence of current unforeseen incidents. They may lead to severe consequences, such as physical drone crashes and disruption of military missions.

To potentially eliminate the instability, some approaches [44, 46] involve dynamic revert-to-baseline schemes, which reset misconfigurations to the default configuration or the average value of each control parameter obtained from the historical flight data. A fundamental assumption of these approaches is that the default configuration or average value is universally suitable for all flying conditions. However, this assumption fails when faced with complicated environmental changes and the need for flexible mission adjustments in practical scenarios. Lack of accounting for dynamic conditions encountered during the flight, the revert-to-baseline schemes may result in mission interruption or trigger further instabilities.

We propose a novel real-time approach to identify (mis)configurations and prevent the occurrence of unforeseen incidents. By continuously monitoring flight states and eliminating instability in real-time, the approach prevents a drone from losing control by only adjusting the configuration. In brief, we first capture and analyze flight data in segments to check flight state changes over a series of times. Next, we calculate a segment deviation value by comparing the physical and the desired states to assess whether instability has occurred. When instability is identified, we employ a pre-trained intelligent agent to eliminate instability through a real-time re-configuration. By incorporating reinforcement learning, the intelligent agent utilizes the re-configuration setting to generate a proper configuration that suits the current flying condition for state rectification without interrupting or terminating the flight mission. This re-configuration rectification is conducted iteratively until instability is fully eliminated. The detection and rectification process *not only fulfills a continuous and successful completion of flight missions but also easily adapts to various flight control systems.*

We implemented a prototype of this flight state rectification system, Nyctea. As Nyctea is built on top of the configuration setting scheme supported by most flight control systems, it can be easily integrated with the existing flight control systems via the wireless communication protocol (e.g., MAVLink). In our experiment, we integrated it with two widely used flight control systems, *Ardupilot* and *PX4* to study how Nyctea performed in the real-world systematically. To simulate diverse physical scenarios, we collected a variety of (mis)configurations resulting in different unstable consequences and flew a drone with these configurations to execute flight missions. During each flight, we employed a misconfiguration to trigger instability and monitored whether Nyctea could eliminate instability, finally leading the drone to complete the mission successfully. As a result, Nyctea successfully identified and eliminated over 85% instability led by (mis)configurations. For each instability, Nyctea averagely generated 4 to 5 configurations to achieve a successful rectification. We also assessed the individual time cost of generating each configuration, Nyctea only took an average of 0.12 ms to generate each configuration, which is efficient.

Contributions.

- *A learning-based approach to generate proper configurations automatically.* We utilize reinforcement learning to construct an adaptive and intelligent agent. By comprehensively exploring correlations between configurations and the corresponding flight states in each flight control system, the agent generates proper configurations based on the physical flight state and the current operational requirements.
- *A real-time detection and rectification approach, Nyctea, to ensure successful completion of each flight mission.* We implemented Nyctea, an automated detection and rectification approach, that identifies instability during the flight and re-configures the drone to eliminate instability in real-time, which achieves on-the-fly rectification without interruption or terminating the ongoing flight missions.

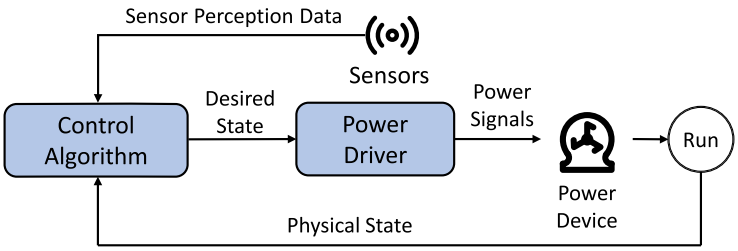


Fig. 1. Workflow of the drone control system.

— *An adaptable approach that can be applied across different flight control systems.* We applied NYCTEA to the prevalent flight control systems, both *Ardupilot* and *PX4*. NYCTEA successfully eliminated over 85% instability caused by (mis)configurations. We open-sourced the dataset, the source code of NYCTEA, and details of our results at https://github.com/BlackJocker1995/nyctea.

Article Organization. The remainder of the article is organized as follows. First, Section 2 provides background information on the drone configuration and presents a motivating example of misconfigurations. Section 3 introduces the architecture of NYCTEA. The detailed processing, including training and rewards, is described in Section 4. Section 5 discusses the performance of NYCTEA through several experiments. Section 6 outlines existing studies on bug detection in robotic vehicles, their mitigation, and configuration adjustment. Finally, we conclude the article in Section 7.

2 Background and Motivation

In this section, we first introduce workflow of a flight control system. Then, we clarify characteristics of the issue caused by misconfigurations and elaborate on the motivations for designing NYCTEA.

2.1 Workflow of Flight Control System

The flight control system is a firmware that enables the control of the movements of a drone during the flight. Figure 1 illustrates the typical workflow of a flight control system. During the flight, the control algorithm unit deconstructs the planned trajectory and processes data from various sources, including sensor data and corresponding physical states. Leveraging these data, the algorithm estimates the next desired flight state to be achieved. Based on the desired state, the flight control system generates power signals for the motors to direct the drone toward that state. The physical state of a drone may not always match the desired state calculated by the control algorithm. In regular flights, the deviation between them remains within a minimal range. However, some research [10, 11, 21, 22] indicate that these deviations intensify as instability impacts the drone's ability to achieve its target. Consequently, this deviation can serve as an indicator for monitoring the drone's flight health.

2.2 Characteristics of Configurations

To adapt to mission-specific requirements, the flight control system supports a set (often hundreds) of control parameters to construct configurations. In alignment with the specific requirements, users have flexibility by selecting different values (within the suggested range) for each control parameter. We outline the characteristics of configurations below, highlighting why misconfigurations can be easily introduced yet are difficult to eliminate.

Feature 1: Parameter design and parameter value ranges are inconsistent in different flight control systems. Due to the lack of standardized regulations, manufacturers may incorporate different capabilities into their flight control systems utilizing distinct control parameter elements. Even for

Table 1. Parameters (Have Range) in Different Control Systems

	Ardupilot	*PX4*
Total	2,990	2,407
Same name, functionality and range	66	
Same functionality and range	16	
Same functionality	249	
Attitude-Related	105(3.5%)	101(3.5%)
Same name, functionality and range	0	
Same functionality and range	0	
Same functionality	37	

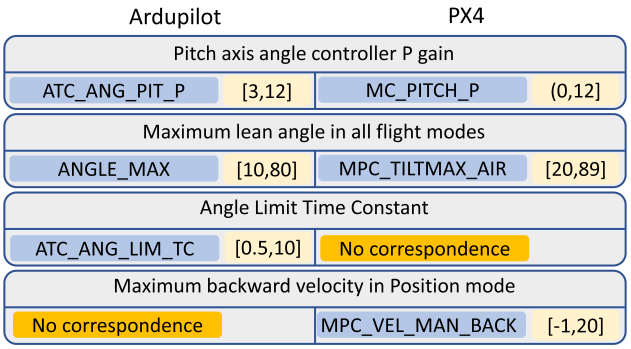


Fig. 2. Parameter sample in flight control systems.

parameter elements that fulfill the same goal, manufacturers might use diverse parameter names. In addition, value ranges associated with the parameters with the same functionality may also vary in different flight control systems. Table 1 shows a comparison of the control parameter elements supported by two prevalent flight control systems, *Ardupilot* [3], and *PX4* [15]. In particular, both flight control systems contain over two thousand parameters, but only 249 achieve the same functionalities. Besides, each flight control system has around one hundred attitude-related control parameters controlling physical flight states, but only 37 (35%) fulfill the same functionalities. Due to such inconsistencies, users may experience confusion when operating different drones simultaneously, leading to potential misconfiguration [9, 21, 22, 36] through inappropriate settings of control parameter values.

Figure 2 illustrates the differences in control parameter elements between *Ardupilot* and *PX4*. Specifically, both *Ardupilot* and *PX4* can convert the error between the desired pitch angle and the actual angle to a desired pitch rate; however, the parameters are defined in distinct names (i.e., `ATC_ANG_PIT_P` and `MC_PITCH_P`) and different parameter value ranges (i.e., $[3, 12]$ and $(0, 12]$). Similarly, when controlling the maximum lean angle in all flight modes, *Ardupilot* and *PX4* separately use `ANGLE_MAX` and `MPC_TILTMAX_AIR` with a corresponding range of $[10, 80]$ and $[20, 89]$. In terms of the capability differences, *Ardupilot* uses `ATC_ANG_LIM_TC` to control the rate at which lean angles are limited to preserve the altitude. Nonetheless, *PX4* does not provide any similar parameter. On the contrary, *PX4* uses a parameter `MPC_VEL_MAN_BACK` to set the maximum backward velocity at a certain position, but *Ardupilot* does not support any similar functionality.

Feature 2: Inter-dependencies among control parameters are complicated, and the impacts can not be predicted. Control parameters are intricately intertwined with complex mutual dependencies because all components in a drone are highly interactive [14]. Thus, effectively managing these dependencies is essential for correctly setting configurations and eliminating instability led by misconfigurations. However, following our manual inspection of the source code of flight control systems, we observed that initialization and range check codes associated with control parameters are usually implemented independently, which is insufficient when proceeding with the real-world variant conditions.

Listing 1 shows the implementation of parameters PSC_VELXY_P and PSC_VELXY_I in *Ardupilot*. Both parameters are used for velocity control gains to refine the acceleration adjustment. However, the parameter implementations only include the processing of parameter values, without describing the correlations between them. If there is a flight mission with multiple turnings, the drone can only complete the turnings when both parameters are configured as PSC_VELXY_P = 2.0 and PSC_VELXY_I = 0.5. Alternatively, it will result in a significant trajectory deviation and further lead to a drone crash if PSC_VELXY_P = 0.2 and PSC_VELXY_I = 0.94. Nevertheless, these values perform well when processing the other flying conditions such as flying straight.

```
1  // @Range: 0.1 6.0
2  {Parameters::k_param_pi_vel_xy, 0, AP_PARAM_FLOA,"PSC_VELXY_P"}
3  // @Range: 0.02 1.00
4  {Parameters::k_param_pi_vel_xy, 1, AP_PARAM_FLOA,"PSC_VELXY_I"}
```

Listing 1. Code Samples on Parameter Declaration.

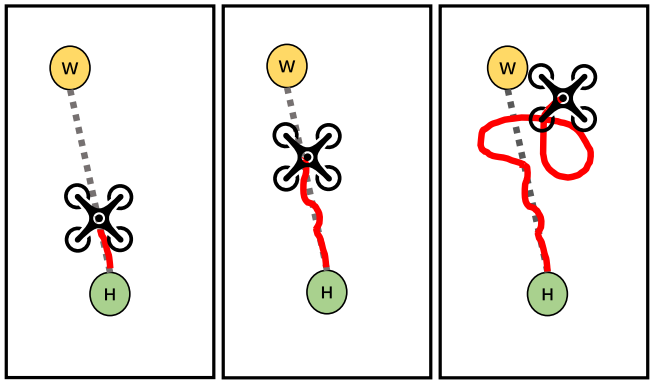
Given such complex value combinations of control parameters, it is infeasible to test all combinations (i.e., configurations) practically and, thus difficult to pinpoint misconfigurations efficiently.

Feature 3: Configurations are affected by uncertainty of the execution environments with unexpected adverse effects. Due to the open nature of the execution environments, the performance of flight states resulting from configurations may deviate from the expected one. While some research approaches attempted to handle such environmental uncertainty by establishing prior knowledge to describe the potential execution conditions, the increased complexity of unforeseen alterations in the real-world makes the creation of comprehensive predefined cases impractical. Even though the attitude-related control parameters may be implemented and set correctly, it is difficult to estimate the environmental changes when flying a drone and how the changes impact the attitude-related parameters.

2.3 Motivating Example of Misconfigurations

To study how misconfigurations adversely affect the flight mission, We launched an execution with a misconfiguration on a drone equipped with an open-source flight control system, *Ardupilot* [3]. The drone was expected to fly from one point to another, and the misconfiguration caused flight hovering to interrupt the execution.

Figure 3 demonstrates the trajectory changes before and after the misconfiguration was deployed, where the green and orange dots separately represent the take-off points and the waypoints, the dashed black lines are the desired trajectories, and the red lines are the physical trajectories. While specifically inspecting the state changes during the flight, we noted the fluctuation of the angular changes in Roll, Pitch, and Yaw, respectively (shown in Figure 4). The solid blue line represents the drone's physical state; the dashed orange line denotes the desired state as calculated by the algorithm; the cyan histogram illustrates the deviation between the two, and the red line corresponds to the deviation curve.



(a) Stage1 (b) Stage2 (c) Stage3

Fig. 3. Trajectory caused by misconfiguration.

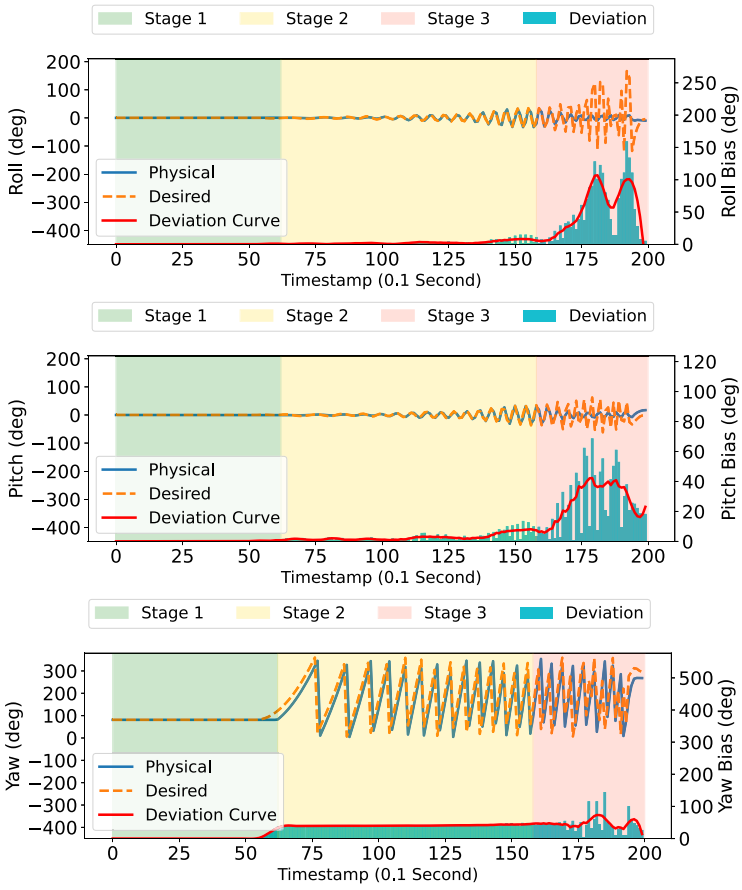


Fig. 4. Angular change in physical state.

Through the deviation, we observed that:

— *0 to 62 (Stage 1, Figure 3(a))*. The drone was stable; that is, its physical roll, pitch, and yaw of each state matched with the desired ones. Even when a minor deviation occurred, the deviation remained within an acceptable range; thus, the flight control system could rectify it through its internal rectification scheme, i.e., **Extended Kalman Filter (EKF)**. The deviation bars remain small, indicating a minimal variation.
— *63 to 158 (Stage 2, Figure 3(b))*. The drone began to sway and spin, exhibiting initial deviations, indicating an alert of potential instability was triggered. However, at this stage, the drone still maintained its correct trajectory. The deviation bars increase, illustrating a potential variation between the physical and the desired states.
— *After 159 (Stage 3, Figure 3(c))*. The drone was hovering, and significant deviations occurred. Simultaneously, the flight control system also reported an error.

In practice, the deviations at Stage 2 are usually unable to be resolved by EKF because of the limited capability of the internal rectification scheme supported by flight control systems. However, when proper configurations are uploaded promptly, the adverse effects can still be eliminated. After the drone entered Stage 3 (i.e., significant deviations were reported), the flight states can no longer be rectified through user commands. In our work, we aim to monitor flight states and identify instability at Stage 2 for further rectification.

2.4 Challenges in Detecting and Rectifying Misconfigurations

Unlike other bugs (e.g., memory corruptions [12], cryptographic misuses [20]), it is challenging to detect and eliminate adverse effects caused by misconfiguration through traditional code analysis techniques.

Static Code Analysis. The inconsistent design of flight control systems introduces differences in control parameter elements, parameter names, and the corresponding value ranges. These variations make the existing code-based bug detection approaches [37, 48] ineffective and inefficient in detecting misconfigurations. While comprehensive dependency graphs correlate to each other, the control parameters could be built, and environmental impacts may also affect practical executions, which is a factor that can not be anticipated at the code development level. Hence, we aim to identify misconfiguration during the executions by considering code execution and environmental variations. Such an approach provides information about the impact that one or more control parameters have on other parameters and the impact of real-world environments.

Dynamic Code Analysis. Previous researches [9, 14, 22, 32, 34, 36, 40] have proposed dynamic analysis approaches to find bugs. Unfortunately, these approaches can only confirm a bug when negative consequences arise, such as drone crashes or task interruptions. For example, when misconfiguration occurs, the drone may deviate from the desired trajectory, leading to a loss of control. This could result in the drone descending into a crowded area, posing a threat to pedestrians underneath. Some research works [44, 46] identify instability and restore parameters to their previous values or average settings before incidents. However, these settings may not align with the current flying conditions, and could potentially trigger incidents. Therefore, our goal is to establish an intelligent system able to identify misconfigurations in real-time, and mitigating their adverse effects through flight state rectification.

3 Overview

In this article, we propose a novel rectification approach that eliminates flight instability by only adjusting the (mis)configurations. Most drone control algorithms are well-designed and capable of accomplishing routine tasks when properly configured [29, 55]. However, misconfigurations

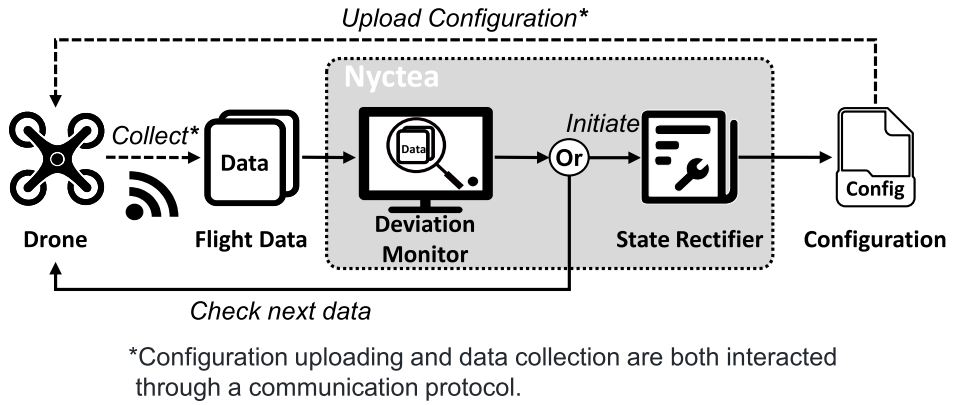


Fig. 5. Overview of Nyctea.

introduce instability or even a crash [21, 22, 36]. In response, our proposed rectification approach captures instability arising from misconfiguration and automatically generates a new configuration to co-work with the existing control algorithm to keep the drone stable. While the official guidelines do not suggest changes to configuration parameters during flight, a misconfiguration can result in physical damage or even a catastrophic failure, where its practical effects can only be evaluated during actual flight. Therefore, to prevent such damage during flight, a well-handling method is to update the configuration, even if it may contradict official guidelines. We deem the risk of configuration modification during flight acceptable when weighed against the potential for direct physical damage. Naturally, if the configuration does not induce any instability, the approach will refrain from making active rectification.

We implement a prototype approach, Nyctea. Figure 5 shows how Nyctea works with the flight control system in a physical drone to achieve state monitoring and rectification. Specifically, Nyctea continuously collects flight data and calculates the deviations to verify whether the drone is experiencing potential instability or merely encountering brief turbulence. If instability is confirmed, Nyctea promptly initializes the rectification process to generate new configurations and send the new configurations to the flight control system through the wireless communication protocol (e.g., MAVLink) until the instability is eliminated. Otherwise, Nyctea continues collecting the subsequent flight data for further analysis.

The core insight of Nyctea is that it utilizes a pre-trained intelligent agent to fulfill an automated, real-time re-configuring against the drone. Unlike traditional operations that require manually configuring the control parameters of the drone before a flight, Nyctea automatically generates a proper configuration at any time by utilizing the agent to analyze the flight state. More importantly, the system continuously monitors the drone's state during the flight and once the originally generated configuration is not proper due to an environmental change, it instantly re-configures the drone to prevent it from entering a state of loss of control.

We designed Nyctea to meet the following targets:

Real-Time Drone Re-Configuration. Typically, users need to interrupt a flight and re-configure the drone if the previously offered control parameters are not appropriate. Nyctea conducts an *on-the-fly rectification* to avoid such task interruptions. It continuously monitors the flight state changes and rectifies improper configuration (and it caused instability) in real-time. Nyctea consists of two modules, a *deviation monitor* (Section 4.2) and a *state rectifier* (Section 4.3), to detect potential instability induced by misconfigurations and re-generate proper configurations to adjust physical

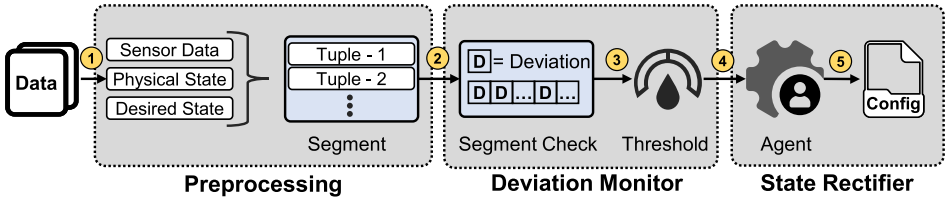


Fig. 6. Workflow of Nyctea.

flight states, respectively. The *deviation monitor* collects the flight data from the drone, and the *state rectifier* utilizes an intelligent agent to guarantee a new configuration could be instantly generated before the drone enters a state of loss of control.

Adaptive Integration with Control Systems. Nyctea aims to support various types of drones. However, drone manufacturers implement their flight control systems in different ways. To support a wider range of drones, Nyctea does not modify any flight control systems (and their flight control algorithms). Instead, it simply adjusts the control parameters and the parameter value ranges of the drone to fulfill the task of flight state stabilization. The insight here is that most drones provide a native communication interface and corresponding communication protocols to help users access their *state perception module* and *configuration-based state control module* [6, 43]. By communicating with the *state perception module*, Nyctea could capture essential flight states calculated by the flight control system, and flight-related information (e.g., flight state, location, accelerometer sensor data, warnings). By reusing the configuration setting functionality provided by the *configuration-based state control module*, Nyctea could work with most flight control systems without considering their specific implementations.

4 Nyctea

Figure 6 shows the workflow of Nyctea. It preprocesses the collected flight data by compiling them into a sequence to create a *segment* (①). Then, Nyctea conducts a segment check where it calculates a sum deviation by comparing all physical and desired flight states within a segment to (②). If the sum deviation exceeds a threshold of secure deviation range (③), Nyctea launches a rectification process (④) for re-configuration (⑤).

4.1 Preprocessing

Nyctea captures flight data broadcast by the flight control system and preprocesses the flight data by generating a segment for further analysis. In detail, at each timestamp t, Nyctea separately retrieves the physical flight state a_t and the desired flight state a'_t from the broadcast messages. Specifically, flight states contain angle and angular rate, each characterized by three Euler angles (roll, pitch, and yaw). As a drone can complete a 360-degree roll, simply recording the flight states in decimal and calculating the differences between physical and desired flight states are inaccurate. We recorded the values of physical and desired flight states as radian-based. Additionally, Nyctea obtains sensor data e_t from the gyroscope and accelerometer measurements, and each provides three values corresponding to the x, y, and z axes. Each flight data captured at timestamp t can be represented by a 3-tuple, i.e., $s_t = \{a_t, a'_t, e_t\}$. Nyctea gathers consecutive flight data to form a segment S to explore the deviation changes. A segment recording the data collected from a timestamp t is denoted as follows:

$$S_t = \{s_j \mid j \in [t, t + m - 1]\}, \tag{1}$$

where m represents the number of flight data in the segment.

4.2 Deviation Monitor

With each segment, Nyctea monitors drone instability by checking whether the segment deviation between desired and physical states exceeds the acceptable range. The physical A_t and desired A'_t states in the segment S_t are:

$$\begin{aligned} A_t &= \{\ a_j \mid j \in [t, t+m-1]\ \} \\ A'_t &= \{\ a'_j \mid j \in [t, t+m-1]\ \}. \end{aligned} \tag{2}$$

Nyctea first computes the Manhattan distance between each physical flight state and its corresponding desired flight state:

$$d_j = \parallel a_j - a'_j \parallel, j \in [t, t+m-1]. \tag{3}$$

It then computes a segment deviation value Dv_t by summing up the individual deviations

$$Dv_t = \sum_{t}^{t+m-1} d_j. \tag{4}$$

Nyctea identifies an instability if Dv_i is larger than a threshold TH (Section 5.2 presents our approach to set the value to be used for this threshold) and then activate the state rectifier module for rectification; otherwise, Nyctea continues capturing and analyzing the next segment.

4.3 State Rectifier

When an instability is reported, Nyctea takes the segment as input and utilizes a configuration-based control scheme to rectify the state and eliminate the instability. The rectification process uses a pre-trained intelligent agent, gradually eliminating instability.

4.3.1 Segment and Action Constraint. The constraint for the rectifier is defined by two aspects: (1) segment constraints, which represent the range of segment values that the rectifier may take as input and (2) action constraints, which delineate the parameters range of configurations that the rectifier can select as output.

Segment Space. The segment space $\mathbb{S}$ is contingent upon internal elements, such as state and sensor management. As previously addressed, all states are transformed into a radian-based format, with the state value range spanning from $-\pi$ to π. The specific device determines the constraints of sensor measurements, and the value range of a device is not limitless. Consequently, the 3-tuple $s_t = \{a_t, a'_t, e_t\}$ represents a finite space, with its constituent segments forming finite spaces.

Action Space. To ensure the agent reacts to instability properly when deployed on actual drones, we predefined an action space, including all possible actions an agent may take. Since we aim to generate new configurations to rectify flight states, each action c is a configuration selected by the agent. The action space $\mathbb{C}$ contains all combinations of control parameter values constrained within the value ranges suggested by manufacturers, thus:

$$c = \{\ p'_1, p'_2, ..., p'_n\ \}, c \in \mathbb{C}, \tag{5}$$

where $p'_i (i = 1, ..., n)$ is the value of the ith control parameter, and n is the total number of control parameters involved in configuration generation. Note that the number of control parameters is flexible in Nyctea. Users can select arbitrary control parameter(s). The more control parameters are involved, the more correlation information of control parameters needs to be explored; consequently, it incurs higher costs for training.

4.3.2 Actor Model Training. Having the predefined agent actions, NYCTEA trains an actor model to generate configurations for rectification and enhance the model based on a formulated reward function. The reward function can direct the actor model toward selecting configurations that can enhance the stability of the physical flight state while discouraging those that induce instability. To create flight execution for model training, we execute a drone with misconfigurations to fulfill a flight mission *AVC2013* [53] (a sample test mission). In each execution, we start the flight using a misconfiguration. NYCTEA proceeds the following steps to train the actor model:

(1) NYCTEA captures a segment S_t at a timestamp t and calculates its deviation Dv_t. If the deviation exceeds the threshold (i.e., $Dv_t > TH$), indicating that there is an instability, NYCTEA proceeds to Step (2); otherwise, NYCTEA continues capturing the subsequent segments.
(2) NYCTEA utilizes the actor model $\mu(\cdot)$ of the intelligent agent to generate a proper configuration c_t.
(3) At the timestamp tt, NYCTEA sends c_t to the flight control system and simultaneously captures the segment S_{tt}. It further calculates the corresponding deviation Dv_{tt} to investigate how c_t affects flight states.
(4) NYCTEA computes a deviation change $Dc_t = Dv_t - Dv_{tt}$ to determine whether c_t enhances the stability or instability. The objective of each generated configuration is to enhance the stability, i.e., reduce the deviation. Hence, $Dc_t > 0$ indicates that configuration c_t properly eliminates the current instability (i.e., enhances the stability) while $Dc_t \leq 0$ indicates that c_t is inappropriate, exacerbating instability. If c_t is considered as proper, NYCTEA continues to Step (5) to calculate a reward r_t; otherwise, NYCTEA sets the reward as $r_t = Dc_t$ and go to Step (6). Even worse, when c_t causes the flight control system to report an incident or a warning, NYCTEA sets $r_t = 2 \times Dc_t$, representing a double negative reward as a punishment, and then goes to Step (6). Hence, even for some cases where $Dc_t > 0$, it will regarded as negative because an incident is triggered.
(5) Our rectification goal is to move the drone from instability to stability. Thus, each configuration is expected to rectify the deviation to a maximum extent, and the physical states are expected to be brought as close to the desired states as possible. Therefore, each configuration leading to a higher Dc_t with a lower Dv_{tt} will receive a better reward. NYCTEA calculates the reward for the proper configuration c_t as follows:

$$r_t = \frac{Dc_t * acc_t}{max(1, Dv_{tt})}, \tag{6}$$

where acc_t is an acceleration scale indicating whether the drone is currently moving. acc_t is the average acceleration value within the segment. As we found that the reward scheme resulted in a tricky situation, i.e., NYCTEA may generate a configuration causing a frozen flight (i.e., not moving), acc_t can prevent such an issue. Note that drones lack sensors that can directly measure the current flight velocity. Instead, as an alternative, we utilize acceleration, which can be directly obtained from sensors. This value indirectly suggests that the drone is in motion within a segment rather than remaining stationary. Furthermore, when $Dv_{tt} \in (0, 1)$, the corresponding reward will be extremely large; thus we utilized $max(1, Dv_{tt})$ to constrain the reward value within a reasonable range.
(6) NYCTEA stores all outputs as an experience entry, represented by a 4-tuple: (S_t, c_t, r_t, S_{tt}) into an experience buffer. NYCTEA leverages the buffer and **Deep Deterministic Policy Gradient (DDPG)** algorithm [39] to update the weights within the actor model $\mu(\cdot)$ as well as the critic model (auxiliary models in DDPG, which are not directly involved in configuration generation). During updating, assuming the actor model comprises N_a weight parameters and the critic model comprises N_c weight parameters, the computational complexities associated

with this update process are as follows: (1) The forward pass for the actor requires $O(N_a)$, while the forward pass for the critic requires $O(N_c)$. (2) The backpropagation process updates the parameters in both models, culminating in $O(N_a + N_c)$.

The intelligent agent iteratively executes Steps (1)–(6) until the deviation is restored to an acceptable range. During the training process, we regard the instability caused by each misconfiguration as successfully eliminated if the agent can rectify the flight execution with the misconfiguration three times and then continue to the next misconfiguration.

4.4 Instability Elimination

Having the trained, intelligent agent, Nyctea takes as input the segment that results in instability and generates a configuration for rectification. Nyctea then sends the configuration to the flight control system and simultaneously captures a new segment. If the segment deviation after re-configuration is lower than the threshold, Nyctea considers that the instability is eliminated. Alternatively, Nyctea promptly engages the intelligent agent again for the subsequent reconfiguration. Note that the primary aim of Nyctea is not to establish an optimal configuration for the drone but to rectify instabilities arising from various misconfigurations during flight. The final configuration maintains its segment deviation value within a specified threshold, ensuring it can successfully complete its mission.

5 Evaluation

We evaluate the performance of Nyctea by answering the following **Research Questions (RQs)**:

- *RQ1: Detection Accuracy.* Can Nyctea identify instability effectively in real-time?
- *RQ2: Rectification Performance.* Can Nyctea eliminate instability to prevent potential threats successfully? Does the number of control parameters involved affect the success rate of rectification?
- *RQ3: Time Cost.* How long does Nyctea take to generate a configuration and finally eliminate instability?

5.1 Experimental Preparation

This section describes our experimental setup and the implementation of Nyctea for further evaluation.

5.1.1 Experimental Setup. We carry out experiments by integrating Nyctea with the prevalent flight control systems and constructing a comprehensive dataset containing safe configurations and misconfigurations.

Implementation. Nyctea is independently deployed on a desktop computer with Python. It can be connected to a flight control system through the wireless communication protocol supported by each flight control system. We specifically integrate it with two mainstream open-source flight control systems, *ArduPilot* (4.2.0) [3] and *PX4* (1.13) [15], using *Mavlink* [42] protocol. Hence, Nyctea sets an inter-process communication with the simulator and connects with a physical drone through Wi-Fi. To achieve data monitoring and rectification function, Nyctea applies a third-party package *Pymavlink* [1]. While in flight, the flight control systems broadcast sensor data and flight states encompassing both the physical and the desired flight data. Nyctea receives this information for monitoring and decides whether to launch rectification.

Program Settings. Since our experimental hardware writes memory data into the NAND flash at 10 Hz, we used a unified sampling rate of 10 Hz, that is, our data capturing interval is 0.1s. Our manual testing showed that a flight control system commonly takes more than two seconds to react

Table 2. Parameters for Experiments

Description	Ardupilot	PX4
Roll angle P gain	ATC_ANG_RLL_P	MC_ROLL_P
Pitch angle P gain	ATC_ANG_PIT_P	MC_PITCH_P
Yaw angle P gain	ATC_ANG_YAW_P	MC_YAW_P
Roll rate P gain	ATC_RAT_RLL_P	MC_ROLLRATE_P
Pitch rate P gain	ATC_RAT_PIT_P	MC_PITCHRATE_P
Yaw rate P gain	ATC_RAT_YAW_P	MC_YAWRATE_P
Roll rate I gain	ATC_RAT_RLL_I	
Roll rate D gain	ATC_RAT_RLL_D	
Pitch rate I gain	ATC_RAT_PIT_I	
Pitch rate D gain	ATC_RAT_PIT_D	
Yaw rate I gain	ATC_RAT_YAW_I	
Yaw rate D gain	ATC_RAT_YAW_D	
Velocity P gain	PSC_VELXY_P	
Velocity I gain	PSC_VELXY_I	
Velocity D gain	PSC_VELXY_D	
Acceleration P gain.	PSC_ACCZ_P	
Acceleration P gain.	PSC_ACCZ_I	
Yaw weight		MC_YAW_WEIGHT
Horizontal error gain		MPC_XY_P
Vertical error gain		MPC_Z_P
Maximum tilt angle		MPC_TILTMAX_AIR
Takeoff climb rate		MPC_TKO_SPEED

to a misconfiguration. Hence, we consecutively captured each segment for two seconds. Given that Nyctea captures flight data every 0.1*s*, each round of analysis involves a segment with a size of 20, i.e., 20 flight data. Regarding the intelligent agent settings, we set the maximum capacity of the buffer as 20,000, the soft update factor as 0.02, the discount factor as 0.99, and the batch size as 64. The intelligent agent (DDPG) was implemented using *PyTorch* [45]. The networks of actor and critic have the same network structure, which contains three liner layers with a 256 hidden size.

Testing Devices. Since instability might cause risks of physical damage to drones and potential accidents involving pedestrians, we conducted the majority of experiments using simulators. Specifically, we used *APM* equipped with *Ardupilot* and *Jmavsim* equipped with *PX4*. For some test cases resulting in minor risks, we integrated these cases into a physical drone, *CUAV ZD550*, equipped with *Ardupilot*, to assess the performance of Nyctea in real-world scenarios.

Parameter Selection. Given the significant differences in control parameters across various flight control systems (ranging from achieved capabilities to value selection ranges), Nyctea has been designed to be flexible to allow users to incorporate any control parameters into the state rectifier module for new configuration generation. Since our system necessitates misconfigurations for training, we leverage prior fuzzing approaches [21, 22] for misconfigurations generation. To reduce experimental costs, we reference the parameters chosen in their fuzzing studies and select the attitude control parameters relevant to attitude adjustment. In total, we chose 17 parameters from *Ardupilot* and 11 parameters from *PX4*, listed in Table 2.

Table 3. Distribution of Configurations

Program	Set	Safe	Misconfiguration	(Actuator related	Mission execution	System failsafe)
Ardupilot	Total	1,781	4,764	(1,719	1,975	1,070)
	Train	1,000	2,785	(1,003	1,037	745)
	Test	781	1,979	(716	938	325)
PX4	Total	1,367	3,462	(1,163	1,567	732)
	Train	1,000	2,228	(847	769	612)
	Test	367	1,234	(316	798	120)

5.1.2 Configuration Dataset Construction. To assess the performance of Nyctea, we constructed a configuration dataset containing safe configurations and misconfigurations, as well as the corresponding flight states. Specifically, we first launched *LGDFuzzer* [22] and *ICSearcher* [21] to explore potentially safe configurations and misconfigurations. Then we executed a drone embedded with the flight control system (i.e., either *Ardupilot* or *PX4* in our experiments) to carry out a flight mission, *AVC2013* [53], while intentionally introducing a misconfiguration. Finally, we leveraged *PyMavlink* to examine whether each flight mission was completed successfully. If so, the configuration was confirmed as "safe"; otherwise, "misconfigurations."

We found that misconfigurations will result in diverse performance outcomes in flight states. Based on the triggered outcomes, we classified misconfigurations into three categories:

- **Actuator-related error (AR)**: It causes the flight control system to report actuator-related warnings, such as potential thrust loss, throttle error, yaw imbalance, and crash.
- **Mission execution error (ME)**: It deviates flight trajectory from the expected paths, such as trajectory deviation and hovering.
- **System failsafe error (SF)**: It causes the flight control system to report failsafes, such as EKF failsafe (unhealthy state of the position and attitude estimation system) and sensor failsafe (inconsistent sensor measurement value).

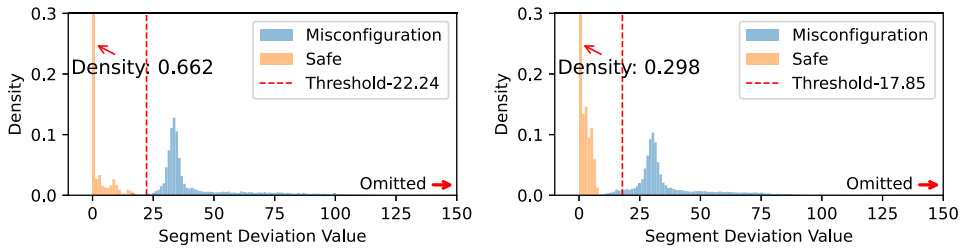
In total, we collected 6,545 configurations from *Ardupilot*, including 1,781 safe configurations and 4,764 misconfigurations, and 4,829 configurations from *PX4*, including 1,367 safe configurations and 3,462 misconfigurations. As Nyctea requires a threshold TH to determine drone instability and relies on an intelligent agent to generate proper configurations, we split the dataset into training and testing sets. The training set was used to determine the threshold and train the agent, while the testing set was used to evaluate Nyctea. Specifically, we randomly selected 1,000 safe configurations in *Ardupilot* and *PX4*, respectively, and separately selected 2,785 and 2,228 misconfigurations from *Ardupilot* and *PX4*. Dataset details are listed in Table 3.

5.2 Instability Threshold Setting

We thus manually launched flight executions using safe configurations and misconfigurations in the training dataset to determine TH. Specifically, for each flight execution with a safe configuration, we randomly started capturing segments at arbitrary timestamps. Nonetheless, for flight execution that received misconfigurations, we randomly captured segments after an error or a warning was reported. Subsequently, we calculated their average deviation values collected from the execution with safe configurations $Dv_{(safe,avg)}$ and those collected from the execution with misconfigurations

Table 4. Segment Deviation Value of Different Types

Program	Type	Min	Max	Average	*TH*
Ardupilot	Safe	0.03	17.41	1.73	22.24
	Misconfiguration	3.22	318.41	42.76	
PX4	Safe	0.04	15.46	1.53	17.85
	Misconfiguration	2.41	305.45	34.18	



(a) Ardupilot (b) PX4

Fig. 7. Segment deviation histogram value of different type.

$Dv_{(mis,avg)}$. Finally, the threshold TH is computed by the following expression:

$$TH = (Dv_{(safe,avg)} + Dv_{(mis,avg)})/2. \tag{7}$$

Figure 7 demonstrates distributions of segment deviation value for *Ardupilot* (Figure 7(a)) and PX4 (Figure 7(b)), where the horizontal axis represents the segment deviation value, and the vertical axis represents its probability density. Data greater than 150 are omitted because they are too few. Specifically, Table 4 lists the minimum and maximum deviation values and the threshold TH. In both *Ardupilot* and *PX4*, the minimum and maximum deviation values show significant differences, indicating that segments when setting safe and misconfigurations are extremely distinct. In our experiments, we set $TH = 22.24$ for *Ardupilot* and $TH = 17.85$ for *PX4*.

5.3 Nyctea Assessment

We utilized the test dataset, i.e., 2,760 configurations, including 781 safe configurations and 1,979 misconfigurations from *Ardupilot* and 1,601 configurations, including 367 safe configurations and 1,234 misconfigurations from *PX4*, to assess the effectiveness of Nyctea in detecting and rectifying instability. Similar to the previous flight execution, we also launched the testing flight execution to accomplish the flight mission *AVC2013* and sent misconfigurations.

5.3.1 Metrics. We define the following metrics to evaluate detection and rectification accurately:

— *Detected* represents the number of unstable executions identified by Nyctea.
— *Correct* is the number of unstable executions that will result in a loss of control if the states are not rectified timely.
— *Missed* represents the number of unstable executions that were not detected by Nyctea before the drone lost control.
— *Eliminated* is the number of reported unstable executions that are successfully rectified by Nyctea, thus leading the drone back into a stable deviation range and accomplishing the flight mission.

Table 5. Detection Results of Unstable Executions

	Detected	Correct	Missed	Precision	Recall	F1-score
Ardupilot	1,964	1,962	17	99.89%	99.14%	99.51%
PX4	1,222	1,219	15	99.75%	98.78%	98.76%

—*Failed* represents the number of *Detected* unstable executions that cannot be rectified, resulting in a control loss.

5.3.2 RQ 1: Detection Accuracy. Table 5 reports the performance of Nyctea in detecting instability caused by misconfigurations (i.e., detecting the number of misconfigurations). Specifically, in *Ardupilot*, Nyctea successfully detected 1,962 out of 1,979, achieving a F1 of 99.51%; and in *PX4*, Nyctea successfully identified 1,219 out of 1,234, achieving a F1 of 98.76%.

Upon manual examination of unreported executions (i.e., 17 and 15 executions in *Ardupilot* and *PX4*, respectively), we identified two specific situations that Nyctea missed:

—Partial angle deviation: As flight states are controlled by the values of three coordinates (i.e., Roll, Yaw, and Pitch), there are 12 instances where only one coordinate's angular value deviates significantly while the others remain steady. Under these scenarios, despite segment deviations potentially staying within an acceptable range, the drone may still experience a loss of control.

—Slightly deviation change: Nyctea relies on deviation changes within a certain period to determine instability. Assuming that drone incidents usually happen within a short period, we collected flight data every two seconds in the experiment. However, we observed that in 20 instances, the drone only had a slight deviation every two seconds; thus, the corresponding segment was considered stable.

In addition, we manually inspected five stable executions misreported as "unstable." We found that these executions have a rare change in a turning angle at a specific timestamp, leading to a significant increment in the segment deviation value. However, misreports did not affect subsequent flights. The new configuration did not increase the segment deviation value for each instance. Since the drone is relatively stable, the agent makes minimal adjustments to the configuration. The maximum adjustment for any single parameter does not exceed ±8% of its current value.

5.3.3 RQ 2: Rectification Performance. Given the identified instability, Nyctea applies the intelligent agent to generate configurations iteratively and send configurations to the flight control system to eliminate instability.

Success Rate of Rectification. Table 6 shows the results of rectification. In the table, *ANC* represents the average number of configurations sent to eliminate instability successfully. In *Ardupilot*, Nyctea successfully rectified 1,784 unstable executions out of the identified 1,962, including 637 caused by AR, 857 caused by ME, and 290 caused by SF, achieving a success rate of 90.74%. In *PX4*, Nyctea successfully rectified 1,080 unstable executions out of the 1,219 detected, including 280 caused by AR, 694 caused by ME, and 106 caused by SF, achieving a success rate of 89.49%. After analyzing the total number of configurations generated for rectification, Nyctea averagely generated 4.32 and 4.57 configurations for each unstable flight execution in *Ardupilot* and *PX4*, respectively.

Through our manual analysis of all the failed cases, we found the major reason is the time cost of rectification. If the time interval between transitioning from identifying misconfiguration to

Table 6. Rectifications Result of Nyctea

	Misconfiguration Type	Eliminated (Rate)	Failed	ANC
Ardupilot	Total	1,784 (90.74%)	178	4.32
	Actuator-related	637 (89.46%)	75	4.43
	Mission execution	857 (92.15%)	73	4.23
	System failsafe	290 (90.62%)	30	4.30
PX4	Total	1,080 (89.45%)	139	4.57
	Actuator-related	280 (90.03%)	31	4.55
	Mission execution	694 (87.73%)	97	4.54
	System failsafe	106 (90.59%)	11	4.62

ANC is the average number of configurations sent to rectify instability.

drone losing control is too short, Nyctea fails to generate sufficient configurations to eliminate the instability. For 162 failed instances, Nyctea reported the instability a bit late (almost reaching the uncontrollable state), resulting in insufficient time for rectification. We then modified the threshold TH to a lower value so that these failed instances could be rectified successfully. However, a lower threshold TH may cause high false positives by reporting lots of "stable" executions as "unstable," further triggering the state rectification module frequently. Even though rectifying a stable state does not harm the original flight, it will waste lots of time, posing a risk of overlooking real instability during this period. Therefore, we still select TH that can correctly identify the most instability.

Impact of Parameter Numbers. Apart from assessing the rectification effectiveness of Nyctea with the selected control parameters listed in Table 2, we also evaluated whether the number of control parameters would affect the success rate of rectification. In detail, we separately selected 2, 4, 8, 12, and 17 control parameters from *Ardupilot*, and 2, 4, 8, 11 from *PX4* to construct sub-experiments. For instance, *Ardupilot* #4 indicates that in this sub-experiment, the misconfigurations generated by *LGDFuzzer* are constructed from the first four *Ardupilot* parameters listed in Table 2. Similarly, #8, #12, #17 indicate that the sub-experiment considers the first 8, 12, and 17 parameters. Similar to the previous experiments, we generated the relevant misconfigurations associated with these control parameters and executed the flight mission with these misconfigurations. Table 7 shows the rectification results when involving different numbers of control parameters. Intuitively, the number of control parameters negatively impacts the rectification performance. When the number of parameters increases, the success rate tends to decrease. But Nyctea can still maintain a success rate of rectification exceeding 85%.

Performance Stability of Rectification. Utilizing the same configurations employed in the *Ardupilot* #12 experiment, we assess the stability of rectification performance across multiple trial experiments. Table 8 illustrates five extended results and their variations compared to the original *Ardupilot* #12 experiment. "Failed -> Eliminated" signifies that the misconfiguration was effectively corrected in this round compared to the original test. Conversely, "Eliminated -> Failed" indicates that the additional failed instances have increased in this round despite having been successfully rectified in the original test. Overall, the success rate exhibited minimal fluctuation; however, the reconfiguration outcomes of certain configurations varied, with several instances of reversed

Table 7. System Performance for Different Parameter Number Misconfigurations

Mis* Num#	2#	4#	8#	12# or 11#	17#
Ardupilot	4/4 100%	16/16 100%	73/73 100%	390/430 90.7%	1047/1184 88.4%
PX4	4/4 100%	22/24 91.6%	69/74 93.24%	392/459 85.4%	-

Mis* Num# is the parameter number in misconfiguration. Each element is $\frac{Eliminated/Total}{SuccessRate}$.

Table 8. Multiple Rounds Verification of Rectification

Change	Round-1	Round-2	Round-3	Round-4	Round-5
Failed -> Eliminated	3	8	1	4	3
Eliminated -> Failed	3	2	3	6	4
$\frac{Eliminated/Total}{SuccessRate}$	390/430 90.7%	396/430 92.1%	388/430 90.2%	388/430 90.2%	389/430 90.4%

Table 9. Multiple Rounds of Verification in Varying Wind Speeds

Wind Impact (Degree, Speed)	45°, 3 m/s	120°, 5 m/s	180°, 9 m/s	30°, 11 m/s
Failed -> Eliminated	2	2	4	1
Eliminated -> Failed	1	5	4	3
$\frac{Eliminated/Total}{SuccessRate}$	391/430 90.9%	387/430 90.0%	390/430 90.7%	388/430 90.2%

repair results. This is due to the system's dynamic rectification strategy. Even with the same misconfiguration, the segments identified as unstable (the initial segment for further rectification) are not identical on each occasion, resulting in different rectified configurations provided by the agent and affecting subsequent iterations. Therefore, although the success rate will fluctuate, our system performance remains stable.

Impact of Environmental Wind. We examine the impact of environmental wind on Nyctea, specifically investigating whether wind speed affects rectification performance. We employed the same configurations in the *Ardupilot* #12 experiment while incorporating varying wind conditions during flight execution. We tested four seniors at varying speeds ranging from 3 m/s to 11 m/s, each facing different horizontal directions. The test results are presented in Table 9. It can be observed that the performance did not decline, as control systems can mitigate environmental wind through its EKF function [7, 28, 41]. However, the practical capacity to withstand wind is contingent upon its framework's design and power capabilities.

Statistics of Configuration Rectification. Taking *Ardupilot* as an example, we depict the distribution of the generated configurations in Figure 8. The figure demonstrates that, in the majority (84.66%), Nyctea could successfully eliminate each unstable execution using six generated configurations.

Furthermore, we recorded the variations in deviation values of four sample instances to gain a better understanding of how Nyctea adjusts misconfigurations (shown in Figure 9). In most instances, we observed a gradual decrease in segment deviation values after uploading new configurations, eventually reaching stable states. Elaborately, Table 10 illustrates the alterations in

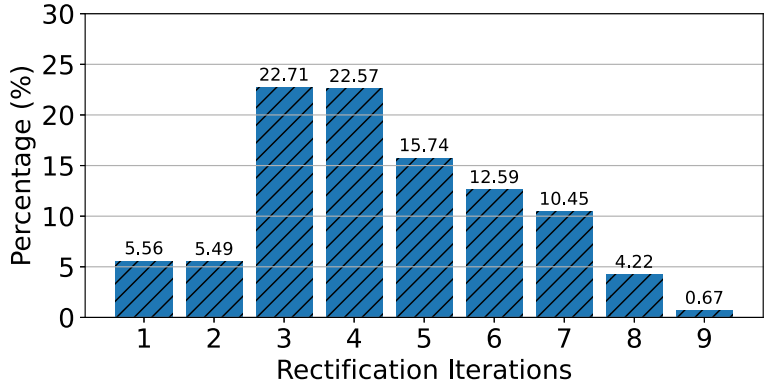


Fig. 8. Distribution of the count of configurations sent to eliminate instability successfully.

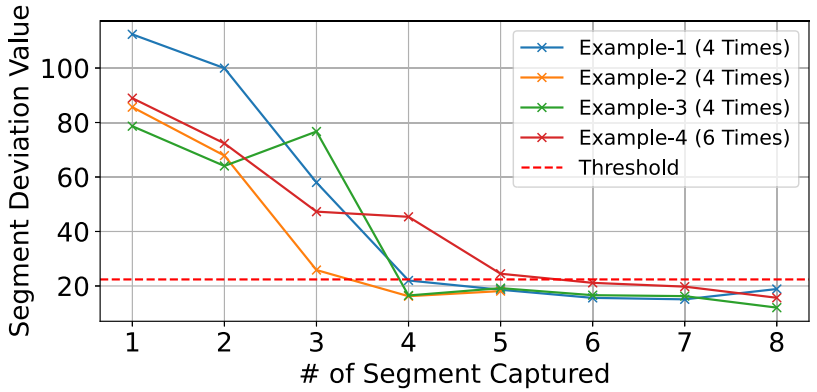


Fig. 9. Example of deviation variants.

parameter values in Example-2. This example begins with a misconfiguration, and NYCTEA used four configurations to restore the drone's stability. The initial two adjustments resulted in significant changes, whereas the subsequent two configurations led to more minor refinements. From the third to the fourth adjustment, only four parameters were modified. For Example-3, although NYCTEA initially introduces an adverse impact, the subsequent configurations gradually rectify the deviation, bringing the drone back to stability.

5.3.4 RQ 3: Time Cost. We assessed the efficiency of NYCTEA by separately calculating the time cost of training the intelligent agent and mitigating unstable states.

- —*Agent Training*. When training the actor model in the intelligent agent, NYCTEA requires 22 hours to mitigate instability caused by all the training misconfigurations for *Ardupilot* and *PX4*. As the model training cost is a one-time cost, 22-hour is an acceptable time cost.
- —*State Rectification*. We individually assessed the time consumption of generating each configuration. Particularly, we ran the intelligent agent in both *Ardupilot* and *PX4* 500 times to create configurations. The agents of NYCTEA in both *Ardupilot* and *PX4* cost 0.12 ms on average to generate one configuration.

It is essential to note that NYCTEA does not impose any additional execution cost on a physical drone, aside from a slight increase in battery usage when transmitting configurations. It is because

Table 10. Parameter Values Change in Experiments

Parameter	Mis*	Change-1	Change-2	Change-3	Change-4
ATC_ANG_RLL_P	11.9	9.4	8.9	8.3	8.3
ATC_ANG_PIT_P	3.0	5.4	5.6	6.7	6.7
ATC_ANG_YAW_P	3.0	8.4	9.7	9.6	9.6
ATC_RAT_RLL_P	0.020	0.395	0.330	0.365	0.365
ATC_RAT_PIT_P	0.495	0.220	0.270	0.295	0.295
ATC_RAT_YAW_P	2.495	1.700	1.900	1.555	1.550
ATC_RAT_RLL_I	1.995	1.579	0.880	0.820	0.820
ATC_RAT_RLL_D	0.001	0.023	0.034	0.030	0.030
ATC_RAT_PIT_I	1.99	1.34	1.60	1.58	1.56
ATC_RAT_PIT_D	0.05	0.028	0.036	0.034	0.034
ATC_RAT_YAW_I	0.07	0.78	0.78	0.72	0.72
ATC_RAT_YAW_D	0.02	0.01	0.012	0.012	0.012
PSC_VELXY_P	0.3	1.1	1.7	2.1	2.0
PSC_VELXY_I	0.97	0.47	0.54	0.53	0.53
PSC_VELXY_D	0.001	0.877	0.840	0.812	0.812
PSC_ACCZ_P	0.25	0.499	0.664	0.637	0.637
PSC_ACCZ_I	0.2	0.579	1.41	1.10	1.00

Mis* is misconfiguration.

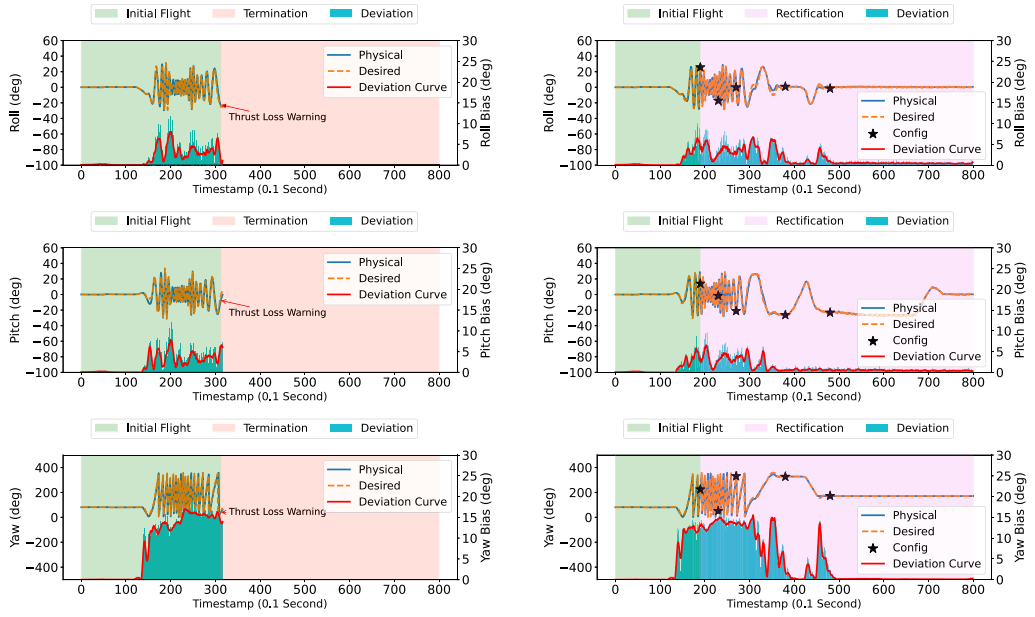
Nyctea is deployed independently on a desktop to connect with drone hardware via Wi-Fi and Pymavlink, ensuring that the drone's operational efficiency remains largely unaffected.

5.4 Case Study

We conducted two comparable flight executions to demonstrate how Nyctea eliminates instability. We implemented a drone with *Ardupilot* to execute the flight mission *AVC2013* in the simulator *APM*. We first executed the flight mission without integrating Nyctea, shown in Figure 10(a), where the solid blue line represents the drone's physical state; the dashed orange line denotes the desired state as calculated by the algorithm; and the cyan histogram illustrates the deviation between the two, and the red line corresponds to the deviation curve.

The drone initialized the flight and took off (in the green area). After taking off, we uploaded a misconfiguration that would potentially cause a thrust loss warning. Then, the drone started shaking and rotating, and the deviation increased gradually. At 31.2s, the flight mission was terminated (in the red area), and the drone raised a thrust loss warning. In the second execution, we integrated Nyctea with *Ardupilot* and replicated the first experiment. Figure 10(b) shows the detailed state changes of Roll, Pitch, and Yaw. The results in the figure clearly show that Nyctea identified instability at 18.9s and initiated the state rectifier module for rectification (in the purple area). Then Nyctea uploaded five configurations at 18.9s, 23s, 27.1s, 38.3s, and 47.8s (marked as black stars). While the initial configuration did not significantly alleviate the instability, the following four configurations gradually reduced the deviations to a small range. Upon completing the fifth rectification upload, the observed deviation gradually decreased, stabilizing the drone's state (i.e., roll, pitch, and yaw). We uploaded the demo of this case [17].

We also conducted a real-world experiment, and the demo is uploaded [18]. We implemented a real-world flight execution using a physical drone *CUAV ZD550*. This drone was embedded with



(a) Drone state change without Nyctea.

(b) Drone state change with Nyctea.

Fig. 10. State change of rectification.

Ardupilot, integrating with Nyctea. Figure 11 shows pitch changes before and after rectification. When the drone took off, we uploaded a misconfiguration that initiated oscillatory and triggered a failsafe warning. After receiving the misconfiguration, Nyctea was initiated when detecting unstable and vibrated states at around 27s (in the purple area). Then, three configurations were uploaded at 27.3s, 36.4s, and 52.1s for rectification (marked as black stars).

5.5 Discussion

In this section, we discuss the application scenarios, platform scalability, and limitations of Nyctea.

5.5.1 Application Scenario. There are three primary scenarios in practical applications where misconfiguration may occur and require our system:

Internal Misconfiguration. Users of drones who are unfamiliar with the correct configuration methods may inadvertently lead to misconfiguration. Most importantly, it is not easily ascertainable through code checks alone in cyberspace whether the configuration will induce instability, leaving users unable to predict the accuracy of their configurations. This requires the actual execution of the configuration to observe its real effects during flight in physical space. In this context, the user necessitates the system to rectify the configurations they set.

External Attack. Additionally, some attackers may exploit this vulnerability to construct misconfigurations to attack the drone deliberately. Since the vulnerability arises from a logical flaw in the flight system, attackers can conceal their intent by disguising the attack as a system bug. In this scenario, the user needs Nyctea to counteract the attack through reconfiguration.

Scenario Change. One configuration is not universally suitable for all flight missions and environments. Consequently, switching scenarios may render a configuration inappropriate, leading to unstable results similar to those mentioned previously. In this scenario, Nyctea can update the configuration to adapt to the current flight mission.

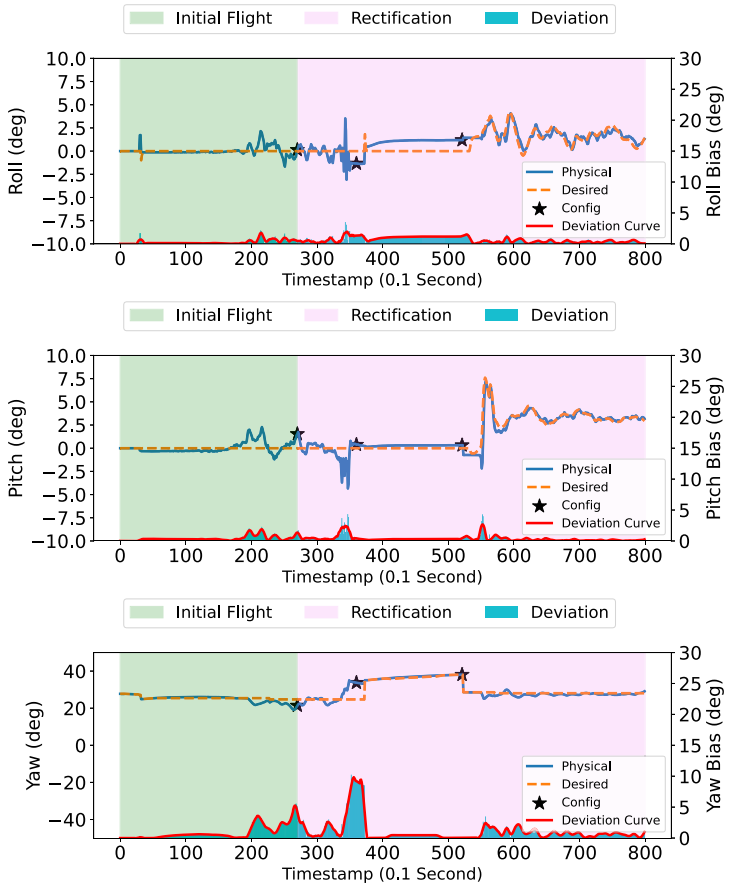


Fig. 11. Rectification example of Nyctea for a real drone.

5.5.2 Platform Scalability. Our experiments applied Nyctea to two widely-used flight control systems. It can be adapted for other platforms or unmanned vehicles, such as guided vehicles and unmanned ships, with some practical modifications. Specifically, developers have three target functions to realize:

(1) *Communication*: Developers must enable Nyctea to obtain the state information of the devices. Unlike drones, other devices may have distinct state structures that must accurately represent the current physical condition of the equipment. Additionally, it is advisable to employ asynchronous implementation to ensure that it does not interfere with the operation of the control system.
(2) *Configuration list*: Developers should initialize the configuration list, which encompasses ranges for the system, where selected parameters are designed to affect the physical actions of the device.
(3) *Training*: The training of Nyctea necessitates an execution environment. Developers need to construct an experimental environment that includes virtual or physical devices to enhance the agent's capabilities, generating configurations in various settings. This process entails numerous execution experiments, which may potentially damage the device.

5.5.3 Limitation.

Interpretability of Parameter Dependencies. Ideally, fine-grained analysis may flag all parameter inter-dependencies among parameters. However, the cross-file analysis is challenging since the code size is substantial and the parameters are defined in disparate locations, such as the parameters of various modules in *Ardupilot*, defined in their respective source files. Our approach does not concentrate on static relationships; instead, it examines the influence of configuration (the combination of parameters) on attitude during flight. The agent in DDPG is a black-box model that directly outputs the appropriate configuration for a given segment. While the agent model may establish inter-dependencies among parameters, elucidating how they influence the decision-making process remains difficult due to the black-box training process.

Initial Consumption. To accommodate additional parameters, Nyctea must undertake further experiments to train the agent for acquiring data across varied scenarios. However, as the number of parameters increases, the volume of experimental data required also escalates. This process can be time-consuming and challenges ensuring data diversity, ultimately impacting performance. As the number of parameters involved in the configuration grows, the rectification process experiences an exponential expansion of the action space, complicating the iterations necessary for achieving stabilization, resulting in a higher incidence of "Failed" outcomes. However, from the user's perspective, unless confronted with exceptionally challenging missions (such as highly complex environments), users generally do not modify such many parameters simultaneously.

6 Related Work

6.1 Bug or Anomaly Detection in Robotic Vehicles

Existing research works on detecting bugs or anomalies in robotic vehicles generally rely on predefined policies/patterns for identification.

6.1.1 Bugs Detection. *PGFuzz* [31] detects code implementations that violate safety policies, such as deploying the parachute at a low altitude. By mutating operation inputs, *PGFuzz* assesses the proximity of current flight states to predefined policies and subsequently identifies the root cause of each violation. Similarly, Liang et al. [38] and Choi et al. [9] identified bugs through pattern match. Liang et al. [38] identified inappropriate usage of bounding functions in the flight control system by developing a differential simulation to illustrate the impact of BF using an offset range on drone behaviors. Choi et al. [9] detected cyber-physical inconsistency vulnerabilities in robotic vehicles by matching the source code semantic goals with the real physical processing feedback. Wang et al. [56] investigated drone error reports, source code, patches, and historical data to summarize patterns of the common bugs and repair strategies for bug analysis. However, these detection approaches are limited to specific bugs that can be matched with the pre-defined patterns/policies. Also, the corresponding code snippets may be correctly implemented for misconfigurations, which are difficult to exploit through pattern-based code analysis.

Different from the above approaches that exploit bugs through code analysis, *MAYDAY* [35] investigates the problematic controller after accident occurred. Starting from the problematic controller, *MAYDAY* conducts the program analysis to trace the root cause of the control system that led to an accident. However, it can only protect against known bugs, and the severe consequences may have already been triggered. Similar to Nyctea, Choi et al. [11] also relied on the physical state analysis during the flight to determine issues (i.e., anomaly control inputs). They focused on studying bugs in flight controllers and could not mitigate adverse effects; instead, they reported issues. *Avis* [52] explores fault handling logic by adopting an aerial-vehicle in-situ model checker to simulate operating mode transactions and inject failure and then identifying sensor bugs caused by narrow failure

Table 11. Comparison with State-of-the-Art Researches

	Repair Method	Online Support	Pre Training	Complete Repair	No Code Insert	No Mission Limitation
Nyctea	Dynamic adjust	✓	✓	✗	✓	✓
DisPatch [34]	Patch	✗	✗	✓	✗	✓
PGPatch [32]	Patch	✗	✗	✓	✗	✓
PID-Piper [13]	Redundantion	✓	✓	✗	✗	✓
SSR [10]	Redundantion	✓	✓	✗	✗	✓
LGDFuzzr [22]	Restriction	✗	✓	✗	✓	✗
ICSearcher [21]	Restriction	✗	✓	✗	✓	✗
RVFuzzr [36]	Restriction	✗	✓	✗	✓	✗
CICADA [46]	Res*	✓	✓	✗	✓	✓
SCVMON [44]	Res*	✓	✓	✗	✗	✓

Res* is restore to default value.

handling logic in control firmware. Although *SAVIOR* [47] also learns physical invariant parameters, it aims to defend against stealthy attacks that can only be triggered under specific conditions.

6.1.2 Anomaly Detection. Some researchers implement anomaly detection through learning models or deviation checks. Galvan et al. [19] discussed an anomaly detection system for drones, addressing security concerns from external attacks and internal failures. It leverages CNNs to analyze real-time sensor data for detecting abnormal UAV statuses. Sindhwani et al. [51] employ machine learning models trained on millions of flight log measurements, using a new robust regression algorithm to identify and exclude abnormal missions. This unsupervised approach reveals normal flight patterns without prior knowledge of aerodynamics or manual labeling. DronLomaly [49] is a deep learning-based approach for detecting anomalies in drone log data that could compromise flight stability. It utilizes an LSTM model trained on normal flight logs from a baseline drone to learn sequential patterns and correlations among flight state units. This model enables real-time anomaly detection during flight. Mithra [2] is an unsupervised Oracle Learning technique that leverages existing telemetry data. It processes unlabeled telemetry logs generated during operation, applying unsupervised multi-step clustering to create behavioral clusters that represent unique contextual behaviors. An execution trace is assessed for anomalies based on its similarity to these clusters, with deviations treated as errors.

6.2 Mitigation in Robotic Vehicles

Approaches to mitigating issues in robotic vehicles can be classified as static or dynamic. Table 11 shows the comparison with other state-of-the-art approaches that identify or repair issues impacting robotic vehicles. We describe these approaches and list the differences between them and our solution.

6.2.1 Static Approaches. Most existing static repair approaches target a specific bug type and then define predefined patterns to repair bugs of this type. In particular, *DisPatch* [34], and *PGPatch* [32] address logic bugs in the source code of robotic vehicles by utilizing predefined patterns to fix the bugs correspondingly. Although such approaches can mitigate instability caused by some logic bugs, the fix patterns cannot cover all possible cases. Also, such approaches cannot address all the other types of bugs.

6.2.2 Dynamic Approaches. There are two common ways for dynamically addressing security issues: assistance involvement and buggy case removal. *PID-Piper* [13] and *SSR* [10] leverage additional modules to replace the compromised ones. *PID-Piper* utilizes a model to implement consistent functions for the flight controller, which learns the relationship between control input and output. When anomalies are detected, the redundant model is involved as a controller to recover robotic vehicles from physical attacks. *SSR* contains an additional sensor algorithm that can transform desired states into appropriate sensor readings and replace compromised ones. However, such module insertions require code modification, which makes them less adaptable. Furthermore, these approaches cannot eliminate instability caused by misconfigurations either. Different from them, *RVFuzzer* [36], *ICsearcher* [21], and *LGDFuzzer* [22] introduce limitations to narrow down the value selection ranges to mitigate range specification bugs. However, reducing the value range restricts the adaptability of robotic vehicles to various flight requirements. CICADA [46] restores all current parameters to default settings when detecting unstable flight caused by configuration. Based on the observation that data-oriented attacks inevitably change the values of some variables in RV control systems, *SCVMON* [44] identifies the safety-critical variables that can impact the safety of RVs. For recovery, *SCVMON* resets the safety-critical variables to an average value of each control parameter after analyzing all historical data.

Compared to those approaches, Nyctea is more user-friendly and adaptive because it can dynamically process issues through its built-in mechanism without any code and functionality modifications.

6.3 Configuration Adjustment

Some studies have proposed configuration adjustment schemes to maintain the system's stability. SURREALIST [30] is a search-based method that automatically generates simulation-based test cases based on previously logged real-world UAV flights, enhancing software-in-the-loop testing realism. SURREALIST analyzes the flight log, extracts UAV and environment configurations, and searches for optimal values for unknown configurations to replicate real-world UAV behavior in simulations. It then manipulates these configurations to discover test scenarios that may trigger unsafe UAV behavior. Valle et al. [54] presents an automated methodology for rectifying misconfigurations in Cyber-Physical Systems. They have redefined the misconfiguration repair issue as a many-objective search problem. Their approach introduces a singular population-based algorithm and implements a strategy that allows for the assessment of the suspiciousness of each parameter. Ultimately, it incorporates a Pareto-optimal archive-based strategy to select and evolve potential misconfiguration patches. Elsisi [16] introduces a new tuning method for adaptive model predictive control in autonomous vehicles, utilizing an improved grey wolf optimizer to enhance performance and effectively handle system uncertainties. DTLF-MPC [5] introduces a hybrid model predictive controller using discrete-time Laguerre functions and a dandelion optimizer to improve steering angle adjustments in autonomous vehicles.

7 Conclusion

We proposed and implemented Nyctea, a novel rectification approach that can detect instability led by misconfigurations by analyzing a series of flight data, and then eliminate instability in real-time by adjusting (mis)configurations to prevent a flight mission from disruption. By combining the configuration scheme and the reinforcement learning approach, we constructed an intelligent agent that can automatically investigate correlations among control parameters and flight data. Such an automation feature ensures that Nyctea can be adapted to different flight control systems easily by using a new set of control parameters and flight data to update the agent model. We integrated Nyctea with the prevalent flight control systems and assessed its success rate of rectification. The

results demonstrated that Nyctea successfully eliminated over 85% instability by generating an average of 4.5 configurations for rectification.

Regarding future work, we intend to further minimize manual involvement and reduce system resource consumption. Specifically, the parameter selection in the system currently requires manual intervention. We aim to integrate Natural Language Processing or Large Language Models to automatically identify which configuration parameters are most beneficial for adjusting the drone's flight attitude. The identification of instability is based on statistical methods. Furthermore, nonlinear methods may be employed to replace the instability detection in our system. While our experimental results indicate current detection had a low false positive rate, the utilization of nonlinear approaches could intuitively further reduce false positives, thereby minimizing unnecessary resource consumption.

References

[1] GitHub. 2024. Pymavlink – A python implementation of the MAVLink protocol. Retrieved from https://github.com/ArduPilot/pymavlink

[2] Afsoon Afzal, Claire Le Goues, and Christopher Steven Timperley. 2021. Mithra: Anomaly detection as an oracle for cyberphysical systems. *IEEE Transactions on Software Engineering* 48, 11 (2021), 4535–4552.

[3] Ardupilot. 2024. Ardupilot – Versatile, trusted, open autopilot software for drones and other autonomous systems. Retrieved from https://ardupilot.org

[4] Antonios Banos, Jim Hayman, Tom Wallace-Smith, Benjamin Bird, Barry Lennox, and Thomas B. Scott. 2020. An assessment of contamination pickup on ground robotic vehicles for nuclear surveying application. *Journal of Radiological Protection* 41, 2 (2020), 179.

[5] Shimaa Bergies, Shun-Feng Su, and Mahmoud Elsisi. 2022. Model predictive paradigm with low computational burden based on dandelion optimizer for autonomous vehicle considering vision system uncertainty. *Mathematics* 10, 23 (2022), 4539.

[6] Mitch Campion, Prakash Ranganathan, and Saleh Faruque. 2018. UAV swarm communication and control architectures: A review. *Journal of Unmanned Vehicle Systems* 7, 2 (2018), 93–106.

[7] Federico Cassola and Massimiliano Burlando. 2012. Wind speed and wind energy forecast through Kalman filtering of Numerical Weather Prediction model output. *Applied Energy* 99 (2012), 154–166.

[8] Dominique Chabot. 2018. Trends in drone research and applications as the Journal of Unmanned Vehicle Systems turns five. *Journal of Unmanned Vehicle Systems* 6, 1 (2018), vi–xv.

[9] Hongjun Choi, Sayali Kate, Yousra Aafer, Xiangyu Zhang, and Dongyan Xu. 2020. 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)*, 263–278.

[10] Hongjun Choi, Sayali Kate, Yousra Aafer, Xiangyu Zhang, and Dongyan Xu. 2020. Software-based realtime recovery from sensor attacks on robotic vehicles. In *Proceedings the 23rd International Symposium on Research in Attacks, Intrusions and Defenses (RAID)*, 349–364.

[11] Hongjun Choi, Wen-Chuan Lee, Yousra Aafer, Fan Fei, Zhan Tu, Xiangyu Zhang, Dongyan Xu, and Xinyan Deng. 2018. Detecting attacks against robotic vehicles: A control invariant approach. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS)*. ACM, New York, NY, 801–816.

[12] Tobias Cloosters, Michael Rodler, and Lucas Davi. 2020. {TeeRex}: Discovery and exploitation of memory corruption vulnerabilities in {SGX} enclaves. In *Proceedings of the 29th USENIX Security Symposium (USENIX Security)*, 841–858.

[13] Pritam Dash, Guanpeng Li, Zitao Chen, Mehdi Karimibiuki, and Karthik Pattabiraman. 2021. 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, 26–38.

[14] Aolin Ding, Matthew Chan, Amin Hassanzadeh, Nils Ole Tippenhauer, Shiqing Ma, and Saman Zonouz. 2023. Get your cyber-physical tests done! data-driven vulnerability assessment of robotic vehicle. In *Proceedings of the International Conference on Dependable Systems and Networks (DSN)*.

[15] DroneCode. 2024. PX4 – An Open Source Flight Control Software for Drones and Other Unmanned Vehicles. Retrieved from https://px4.io/

[16] Mahmoud Elsisi. 2024. Optimal design of adaptive model predictive control based on improved GWO for autonomous vehicle considering system vision uncertainty. *Applied Soft Computing* 158 (2024), 111581.

[17] Simulator Example. 2024. Thrust Loss. Retrieved from https://youtu.be/nsdNRPihif8

[18] Real Flight Example. 2024. Real Flight. Retrieved from https://youtu.be/ZAidyll9z8A

[19] Julio Galvan, Ashok Raja, Yanyan Li, and Jiawei Yuan. 2021. Sensor data-driven uav anomaly detection using deep learning approach. In *Proceedings of IEEE Military Communications Conference (MILCOM)*. IEEE, 589–594.

[20] Ruidong Han, Huihui Gong, Siqi Ma, Juanru Li, Chang Xu, Elisa Bertino, Surya Nepal, Zhuo Ma, and JianFeng Ma. 2023. A credential usage study: Flow-aware leakage detection in open-source projects. *IEEE Transactions on Information Forensics and Security* 19 (2023), 722–734.

[21] Ruidong Han, Siqi Ma, Juanru Li, Surya Nepal, David Lo, Zhuo Ma, and JianFeng Ma. 2024. Range specification bug detection in flight control system through fuzzing. *IEEE Transactions on Software Engineering* 50, 3 (2024), 461–473.

[22] Ruidong Han, Chao Yang, Siqi Ma, JiangFeng Ma, Cong Sun, Juanru Li, and Elisa Bertino. 2022. Control parameters considered harmful: Detecting range specification bugs in drone configuration modules via learning-guided search. In *Proceedings of the IEEE/ACM 44th International Conference on Software Engineering (ICSE)*. IEEE, 462–473.

[23] Vikas Hassija, Vinay Chamola, Adhar Agrawal, Adit Goyal, Nguyen Cong Luong, Dusit Niyato, Fei Richard Yu, and Mohsen Guizani. 2021. Fast, reliable, and secure drone communication: A comprehensive survey. *IEEE Communications Surveys & Tutorials* 23, 4 (2021), 2802–2832.

[24] Xiaoyan Huang, Supeng Leng, Sabita Maharjan, and Yan Zhang. 2021. Multi-agent deep reinforcement learning for computation offloading and interference coordination in small cell networks. *IEEE Transactions on Vehicular Technology* 70, 9 (2021), 9282–9293.

[25] Joon-Ha Jang, Mangi Cho, Jaehoon Kim, Dongkwan Kim, and Yongdae Kim. 2023. Paralyzing drones via EMI signal injection on sensory communication channels. In *Proceedings of the Network and Distributed System Security Symposium (NDSS)*.

[26] Jinseob Jeong, Dongkwan Kim, Joon-Ha Jang, Juhwan Noh, Changhun Song, and Yongdae Kim. 2023. Un-rocking drones: Foundations of acoustic injection attacks and recovery thereof. In *Proceedings of the Network and Distributed System Security Symposium (NDSS)*.

[27] Jiasheng Jiang, Jingzheng Wu, Xiang Ling, Tianyue Luo, Sheng Qu, and Yanjun Wu. 2023. APP-miner: Detecting API misuses via automatically mining API Path patterns. In *Proceedings of the IEEE Symposium on Security and Privacy (SP)*. IEEE Computer Society, 43–43.

[28] Xiaofei Jing, Jiarui Cui, Hongtai He, Bo Zhang, Dawei Ding, and Yue Yang. 2017. Attitude estimation for UAV using extended Kalman filter. In *Proceedings of the 29th Chinese Control And Decision Conference (CCDC)*. IEEE, 3307–3312.

[29] Belkacem Kada and Y. Ghazzawi. 2011. Robust PID controller design for an UAV flight control system. In *Proceedings of the World congress on Engineering and Computer Science*, Vol. 2, 1–6.

[30] Sajad Khatiri, Sebastiano Panichella, and Paolo Tonella. 2023. Simulation-based test case generation for unmanned aerial vehicles in the neighborhood of real flights. In *Proceedings of IEEE Conference on Software Testing, Verification and Validation (ICST)*. IEEE, 281–292.

[31] Hyungsub Kim, Muslum Ozgur Ozmen, Antonio Bianchi, Z. Berkay Celik, and Dongyan Xu. 2021. PGFUZZ: Policy-guided fuzzing for robotic vehicles. In *Proceedings of the 28th the Network and Distributed System Security Symposium (NDSS)*.

[32] Hyungsub Kim, Muslum Ozgur Ozmen, Z Berkay Celik, Antonio Bianchi, and Dongyan Xu. 2022. PGPATCH: Policy-guided logic bug patching for robotic vehicles. In *Proceedings of the 43rd IEEE Symposium on Security and Privacy* (S & P).

[33] Seulbae Kim, Major Liu, Junghwan "John" Rhee, Yuseok Jeon, Yonghwi Kwon, and Chung Hwan Kim. 2022. Drivefuzz: Discovering autonomous driving bugs through driving quality-guided fuzzing. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 1753–1767.

[34] Taegyu Kim, Aolin Ding, Sriharsha Etigowni, Pengfei Sun, Jizhou Chen, Luis Garcia, Saman Zonouz, Dongyan Xu, and Dave Tian. 2022. Reverse engineering and retrofitting robotic aerial vehicle control firmware using dispatch. In *Proceedings of the 20th Annual International Conference on Mobile Systems, Applications and Services*. 69–83.

[35] Taegyu Kim, Chung Hwan Kim, Altay Ozen, Fan Fei, Zhan Tu, Xiangyu Zhang, Xinyan Deng, Dave Jing Tian, and Dongyan Xu. 2020. From control model to program: Investigating robotic aerial vehicle accidents with MAYDAY. In *Proceedings of the 29th USENIX Security Symposium (USENIX Security)*, 913–930.

[36] Taegyu Kim, Chung Hwan Kim, Junghwan Rhee, Fan Fei, Zhan Tu, Gregory Walkup, Xiangyu Zhang, Xinyan Deng, and Dongyan Xu. 2019. RVFuzzer: Finding input validation bugs in robotic vehicles through control-guided testing. In *Proceedings of the 28th USENIX Security Symposium (USENIX Security)*. USENIX Association, 425–442.

[37] Xiangyu Li, Chaoqun Yang, Jiangsha Ma, Yongchang Liu, and Shujuan Yin. 2017. Energy-efficient side-channel attack countermeasure with awareness and hybrid configuration based on it. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 25, 12 (2017), 3355–3368.

[38] Xiaozhou Liang, John Henry Burns, Joseph Sanchez, Karthik Dantu, Lukasz Ziarek, and Yu David Liu. 2021. Understanding bounding functions in safety-critical UAV software. In *Proceedings of the IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*, 1311–1322.

[39] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. 2015. Continuous control with deep reinforcement learning. arXiv:1509.02971. Retrieved from https://arxiv.org/abs/1509.02971

[40] Taifeng Liu, Chao Yang, Xinjing Liu, Ruidong Han, and Jianfeng Ma. 2023. RPAU: Fooling the eyes of UAVs via physical adversarial patches. *IEEE Transactions on Intelligent Transportation Systems* 25, 3 (2023), 2586–2598.

[41] Petroula Louka, Georges Galanis, Nils Siebert, Georges Kariniotakis, Petros Katsafados, Ioannis Pytharoulis, and George Kallos. 2008. Improvements in wind speed forecasts for wind power prediction purposes using Kalman filtering. *Journal of Wind Engineering and Industrial Aerodynamics* 96, 12 (2008), 2348–2362.

[42] Lorenz Meier, J. Camacho, B. Godbolt, J. Goppert, L. Heng, and M. Lizarraga. 2013. Mavlink: Micro Air Vehicle Communication Protocol.

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

[44] Sangbin Park, Youngjoon Kim, and Dong Hoon Lee. 2023. 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)*, 547–563.

[45] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. 2019. PyTorch: An imperative style, high-performance deep learning library. In *Proceedings of the 33rd International Conference on Neural Information Processing Systems*, 8024–8035.

[46] Salil Purandare, Urjoshi Sinha, Md Nafee Al Islam, Jane Cleland-Huang, and Myra B. Cohen. 2023. Self-adaptive mechanisms for misconfigurations in small uncrewed aerial systems. In *Proceedings of the 18th Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*. IEEE, 169–180.

[47] Raul Quinonez, Jairo Giraldo, Luis Salazar, Erick Bauman, Alvaro Cardenas, and Zhiqiang Lin. 2020. SAVIOR: Securing autonomous vehicles with robust physical invariants. In *Proceedings of the 29th USENIX Security Symposium (USENIX Security)*, 895–912.

[48] Mohammed Sayagh, Noureddine Kerzazi, and Bram Adams. 2017. On cross-stack configuration errors. In *Processing of the 39th IEEE/ACM International Conference on Software Engineering (ICSE)*. IEEE, 255–265.

[49] Lwin Khin Shar, Wei Minn, Nguyen Binh Duong Ta, Jiani Fan, Lingxiao Jiang, and Daniel Lim Wai Kiat. 2022. DronLomaly: Runtime detection of anomalous drone behaviors via log analysis and deep learning. In *Proceedings of 29th Asia-Pacific Software Engineering Conference (APSEC)*. IEEE, 119–128.

[50] Abhishek Sharma, Pankhuri Vanjani, Nikhil Paliwal, Chathuranga M. Wijerathna Basnayaka, Dushantha Nalin K. Jayakody, Hwang-Cheng Wang, and P. Muthuchidambaranathan. 2020. Communication and networking technologies for UAVs: A survey. *Journal of Network and Computer Applications* 168 (2020), 102739.

[51] Vikas Sindhwani, Hakim Sidahmed, Krzysztof Choromanski, and Brandon Jones. 2020. Unsupervised anomaly detection for self-flying delivery drones. In *Proceedings of the 19th IEEE International Conference on Robotics and Automation (ICRA)*, 186–192.

[52] Max Taylor, Haicheng Chen, Feng Qin, and Christopher Stewart. 2021. 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, 471–483.

[53] ArduPilot Team. 2013. Autonomous vehicle competition 2013. Retrieved from https://github.com/dronekit/ardupilot-releases/blob/master/Tools/autotest/copter_AVC2013_mission.txt

[54] Pablo Valle, Aitor Arrieta, and Maite Arratibel. 2023. Automated misconfiguration repair of configurable cyber-physical systems with search: An industrial case study on elevator dispatching algorithms. In *Proceedings of IEEE/ACM 45th International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*. IEEE, 396–408.

[55] Ramon Vilanova, Víctor M. Alfaro, and Orlando Arrieta. 2012. Robustness in PID control. In *PID Control in the Third Millennium: Lessons Learned and New Approaches*. Springer, 113–145.

[56] Dinghua Wang, Shuqing Li, Guanping Xiao, Yepang Liu, and Yulei Sui. 2021. 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)*, 20–31.

[57] Ce Zhou, Qiben Yan, Yan Shi, and Lichao Sun. 2022. DoubleStar:Long-range attack towards depth estimation based obstacle avoidance in autonomous systems. In *Proceedings of the 31st USENIX Security Symposium (USENIX Security)*, 1885–1902.

[58] Yuan Zhou, Yang Sun, Yun Tang, Yuqi Chen, Jun Sun, Christopher M. Poskitt, Yang Liu, and Zijiang Yang. 2023. Specification-based autonomous driving system testing. *IEEE Transactions on Software Engineering* 49, 6 (2023), 3391–3410.

Received 26 June 2024; revised 8 September 2024; accepted 15 October 2024