RouthSearch: Inferring PID Parameter Specification for Flight Control Program by Coordinate Search

SIAO WANG, Fudan University, China
ZHEN DONG*, Fudan University, China
HUI LI, Fudan University, China
LIWEI SHEN, Fudan University, China
XIN PENG, Fudan University, China
DONGDONG SHE, Hong Kong University of Science and Technology, China

Flight control programs are widely used in unmanned aerial vehicles (UAVs) to manage and maintain UAVs' flying behaviors dynamically. These flight control programs include a PID control module that takes three user-configurable PID parameters: Proportional (P), Integral (I), and Derivative (D). Users can also adjust these PID parameters during flight to suit the needs of various flight tasks. However, flight control programs do not have sufficient safety checks on the user-provided PID parameters, leading to a severe vulnerability of UAV—input validation bug. It happens when the user misconfigures PID parameters and causes the UAV to enter a dangerous state, such as deviation from the expected path, loss of control, or even crash.

Prior works use random testing approaches like fuzzing to identify invalid PID parameters from user input. However, they are not effective in the three-dimensional search space of PID parameters. Meanwhile, each dynamic execution of the UAV test is very expensive, further affecting the performance of random testing.

In this work, we address the problem of PID parameter misconfiguration by combining the Routh-Hurwitz stability criterion with coordinate search, introducing a method called RouthSearch. Instead of identifying misconfigured PID parameters in an ad-hoc fashion, RouthSearch principledly determines valid ranges for three-dimensional PID parameters. We first leverage the Routh-Hurwitz Criterion to identify a theoretical PID parameter boundary. We then refine the boundary using an efficient coordinate search. The valid range of three-dimensional PID parameters determined by RouthSearch can filter out misconfigured PID parameters from users during flight and further help to discover logical bugs in popular flight control programs.

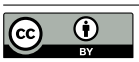
We evaluated RouthSearch across eight flight modes in two popular flight control programs, PX4 and ArduPilot. The results show that RouthSearch can determine the valid ranges of the three-dimensional PID parameters with an accuracy of 92. 0% when compared to the ground truth. In terms of the total number of misconfigured PID parameters, RouthSearch discovers 3,853 sets of PID misconfigurations within 48 hours, while the STOA work PGFuzz only discovers 449 sets of PID misconfigurations, significantly outperforming prior works by 8.58 times. Additionally, our method helps to detect three bugs in ArduPilot and PX4.

CCS Concepts: • **Software and its engineering → Search-based software engineering**.

Additional Key Words and Phrases: UAV Testing, Misconfiguration, PID Control, Specification Inference

*Corresponding author

Authors' Contact Information: Siao Wang, Fudan University, Shanghai, China, 22110240039@m.fudan.edu.cn; Zhen Dong, Fudan University, Shanghai, China, zhendong@fudan.edu.cn; Hui Li, Fudan University, Shanghai, China, 22210240023@m.fudan.edu.cn; Liwei Shen, Fudan University, Shanghai, China, shenliwei@fudan.edu.cn; Xin Peng, Fudan University, Shanghai, China, pengxin@fudan.edu.cn; Dongdong She, Hong Kong University of Science and Technology, Hong Kong, China, dongdong@cse.ust.hk.



ACM 2994-970X/2025/7-ARTISSTA029
https://doi.org/10.1145/3728904

ACM Reference Format:
Siao Wang, Zhen Dong, Hui Li, Liwei Shen, Xin Peng, and Dongdong She. 2025. RouthSearch: Inferring PID Parameter Specification for Flight Control Program by Coordinate Search. *Proc. ACM Softw. Eng.* 2, ISSTA, Article ISSTA029 (July 2025), 23 pages. https://doi.org/10.1145/3728904

1 Introduction

Unmanned Aerial Vehicles (UAVs) are widely used in various domains, such as agriculture [9], meteorology [34], search and rescue operations [30], and modern warfare [55]. They rely on flight control programs to manage the UAV's position and attitude during flight. Flight control programs include a PID control module with three *user-configurable* PID parameters: Proportional (P), Integral (I), and Derivative (D). The PID parameters can be dynamically adjusted in flight to meet the requirements of various UAV tasks. However, there are no sufficient safety checks on the PID parameters before they are dynamically adjusted [14, 15]. The lack of configuration safety checks on the PID parameters leads to a serious security problem in the UAV flight control program, named *input validation bug* [27]. The input validation bug occurs when a user passes a set of misconfigured PID parameters to the flight control program, further causing the UAV to turn into a dangerous flight state, such as deviations from the intended flight paths, collisions, lost control, and crashes [5, 12, 31, 44]. As flight control programs like PX4 [39] and ArduPilot [1] are widely used in UAVs, the input validation bug is becoming a key problem in UAV safety. A protection technique that can identify dynamically misconfigured PID parameters during flight is urgently needed.

Prior work PGFuzz [24] uses random fuzzing to detect PID parameters misconfiguration in an ad-hoc fashion. RVFuzzer [27] and LGDFuzzer [15] identify valid ranges for each individual PID parameter through binary search and machine learning-based prediction, respectively. However, these approaches treat each parameter independently, overlooking their interdependencies and thus are ineffective in identifying valid ranges when taking these relationships into account.

Identifying valid ranges for PID parameters is challenging. First, the classic stability model that can reason the validity of PID parameters is not directly applicable in real-world scenarios. The stability theory only reasons the mathematical model of the PID control and gives a theoretical boundary that separates the valid and invalid PID parameters [4, 6]. In practice, the flying behavior of UAVs can be influenced by many factors other than the underlying mathematical model, such as the implementation of the flight control program and environmental noise. These factors cause a *drastic shift* on its theoretical PID parameter boundary from the real one. Second, the search space for PID parameters is three-dimensional and includes millions of possible parameter sets. The runtime cost to validate each set of PID parameters is very high, taking hundreds of seconds to run on a UAV simulator.

To address these challenges, an efficient dynamic technique is desired because it is lightweight and resistant to unknown physical noise. We observe that despite the gap between the theoretical PID parameter boundary and the real one being unknown, the real PID parameter boundary is continuous and fluctuates around the theoretical one. Therefore, we start from the theoretical boundary, then use the coordinate search [56] to quickly depict such a gap and construct an accurate PID parameter boundary.

In this paper, we propose a principled approach called RouthSearch to efficiently determine the valid range for the three-dimensional PID parameters. Given these valid ranges, we can detect PID misconfigurations of UAVs on the fly. Specifically, we first leverage the Routh-Hurwitz criterion [17] from stability theory to derive a theoretical PID parameter boundary as the starting line. Then, we perform an efficient coordinate search to quickly refine the boundary by identifying the unknown boundary shift. To detect whether a PID parameter configuration is valid, we implement a UAV misbehavior validator as an oracle. We record the UAV flight log and analyze it against Metric

Temporal Logic encoding the UAV specification. If a UAV's flight behavior violates the MTL specification, we classify the configuration as invalid. RouthSearch can identify valid ranges of PID control parameters in different modes defined in mainstream flight control program. RouthSearch also helps users avoid misconfiguring PID parameters and even discover logic bugs in implementing a widely adopted flight control program, ArduPilot.

We evaluated RouthSearch under the eight typical flight modes of two popular flight control programs (i.e., PX4 and ArduPilot). We measure the performance of RouthSearch by computing the ratio of true PID parameter misconfigurations out of all predicted PID parameter misconfigurations. Our experimental results show that RouthSearch can identify the valid range of PID parameters with an accuracy of 92.0%. We also evaluate RouthSearch's ability to discover individual PID parameter misconfigurations. The result shows that RouthSearch discovers an average of 3,853 misconfigurations within 48 hours, while the baseline tool PGFuzz has only discovered 449 misconfigurations.

Our key contributions are summarized as follows:

- We propose a principled method to identify the valid range of three-dimensional PID parameters. The valid range can help prevent users from misconfiguring PID parameters during flight.
- We use an efficient coordinate search to identify individual PID parameter misconfigurations on the UAV flight control program.
- We evaluate our method under 8 flight modes on two popular open-source flight control programs. The evaluation result shows that RouthSearch identifies the boundary of valid PID parameter regions with 92.0% accuracy.
- We open-source our tool to help users properly set up the PID parameters and further foster future research in this domain. We have released our implementation in the web at https://github.com/SciC0d3m4xOfW/RouthSearch.

The rest of the paper is organized as follows. Section 2 summarizes the background of the UAV PID control and the Routh-Hurwitz Stability Criterion. Section 3 explains the challenges of identifying valid ranges for PID parameters and the insights behind our method. Section 4 introduces our methodology in detail. We present our experimental results in Section 5 and discuss the limitations of our method in Section 6. We conclude with a summary of related work in Section 7 and a conclusion in Section 8.

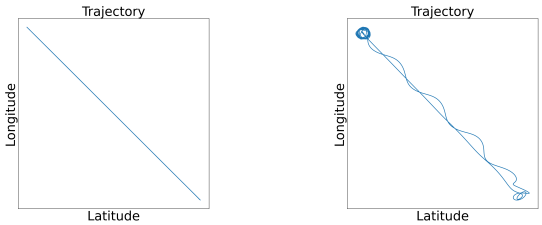
2 Background

2.1 PID Control Module

The PID control module regulates the UAV's position and attitude to maintain its control and stability. It consists of two sub-modules: 1) position sub-module to manage UAV's position and velocity; 2) attitude sub-module to manage UAV's attitude and the angular velocity [29, 33]. The PID control module has three user-configurable PID parameters, corresponding to Proportional (P), Integral (I), and Derivative (D). The three user-configured PID parameters are used to dynamically maintain the UAV's stability during flight.

The error in a control system is typically defined as the difference between the desired state and the actual state of the system. A control system is stable when its error approaches zero as time approaches infinity. A stable UAV indicates that both the position and attitude sub-modules are properly configured and have around zero errors [11]. While misconfigured PID parameters can lead to large errors and an unstable UAV (e.g., oscillating or diverging from the expected path).

To illustrate the significant impact of PID parameter configurations, Figure 1 provides an example of UAV trajectory under valid/invalid PID configurations. Figure 1a demonstrates a stable flight



(a) Valid PID Config (b) Invalid PID Config

Fig. 1. Comparison of UAV trajectories before and after setting invalid PID configurations. After the invalid PID parameter is set, the UAV trajectory becomes waving and unstable as shown in Figure 1b.

trajectory from the bottom-right corner to the top-left corner and back. While Figure 1b depicts an unstable flight trajectory, characterized by unstable and waving movements and localized circular patterns, which highlight the instability of the UAV resulting from invalid PID configurations.

2.2 Routh-Hurwitz Stability Criterion

The Routh-Hurwitz Criterion is a mathematical condition used to determine the stability of a control system [17]. For a closed-loop PID control system, we can have the following equation:

$$\frac{\mathrm{d}^2x(t)}{\mathrm{d}t^2} + a_2\frac{\mathrm{d}x(t)}{\mathrm{d}t} + a_1x(t) = u(t) \tag{1}$$

$$u(t) = k_p e(t) + k_i \int_0^t e(\tau)\,d\tau + k_d \frac{\mathrm{d}e(t)}{\mathrm{d}t} \tag{2}$$

where $x(t)$ and $u(t)$ denote the system output and the control input at time step t respectively. $e(t)$ represents the error at time t. a_1 and a_2 are constant coefficients set by the system. k_p, k_i, and k_d are the PID control's proportional, integral, and derivative parameters. According to the Routh-Hurwitz Criterion, when the following condition is satisfied:

$$k_p + a_1 > 0 \tag{3}$$

$$k_d + a_2 > 0 \tag{4}$$

$$k_i > 0 \tag{5}$$

$$(k_p + a_1)(k_d + a_2) > k_i \tag{6}$$

the system is stable [17, 59]. k_p, k_i, and k_d are manually configured by users. Due to a lack of sufficient safety checks on these PID parameters in flight, incorrect configurations by users can lead to severe consequences such as deviation from the expected path, collisions, loss of control, and crashes. Note that the Routh-Hurwitz Criterion is a theoretical model of a low-level control system and hence cannot be directly applied to UAV systems. There exists a gap between the theoretical boundary derived by the Routh-Hurwitz Criterion and the real one. We will explain the root cause of such boundary shifts in the following section.

3 Challenges and Insight

Routh-Hurwitz stability analysis shows that there exists a boundary that separates valid and invalid PID parameter configurations [59]. However, we cannot directly use the Routh-Hurwitz Criterion to derive a valid PID parameter boundary due to an unknown gap between the theoretical and real boundaries. We first explain three challenges in determining a valid PID parameter boundary. We then introduce our insights using an efficient coordinate search to quickly depict the unknown gap and construct the valid PID parameter boundary.

Challenge: Flight Mode Influence. Flight mode can greatly influence the validity of the PID parameter. Popular flight control program defines various flight modes. Each mode represents a unique flight behavior that imposes corresponding constraints on the PID control module. A stable PID parameter configuration satisfying the Routh-Hurwitz Criterion may fail to meet these additional constraints defined in flight mode, resulting in unexpected flying behaviors or failed flight missions. Hence, there is no silver bullet PID parameter set that is valid across diverse flight modes. A valid PID parameter range must be *mode-specific*. For example, a PID parameter configuration (p, i, d) satisfies the Routh-Hurwitz Criterion, and hence, the UAV configured with (p, i, d) can fly stably. However, under a certain mode, the UAV configured with (p, i, d) could fail to accomplish the required flying behaviors and lead to a crash [3].

Challenge: External Noise. External noises in sensors and environments contribute random variance to the valid PID parameter range. PID control modules are easily affected by high-frequency noise during a flying mission. Even a slight disturbance from sensor input or the environment can cause a significant oscillation in the UAV [42]. Although the developers have spent a considerable amount of time modeling and performing stability analysis of a UAV to derive a valid PID parameter range, the UAV may still lose control or crash due to improper PID parameter configuration. Consequently, the PID parameter range guidance in existing documentation is prone to erroneous. As we can see from Section 5.10.2, a PID parameter configuration falling in the suggested parameter ranges in the document can still cause an unexpected UAV malfunction.

Challenge: Huge PID Parameter Space. Given the unknown gap between the theoretical and real boundaries, an ideal way to determine the real boundary is to use a dynamic testing approach that can adapt to diverse flight modes and random noise. However, the PID parameter space is a three-dimensional space, composed of a huge number of possible PID parameter configurations. For instance, the search space of a popular UAV flight control program, ArduPilot, is [0.1, 6], [0.02, 1], and [0.0, 1]. We set the search step size for each dimension as 0.1, 0.01, and 0.001, respectively. There are a total of around 6 million parameter sets. Meanwhile, the runtime cost to verify the validity of each PID parameter configuration is very high, either through a real UAV device or a simulator.

Insight: Coordinate Search. Coordinate search is an efficient search algorithm that can perform a coordinate-wise search to minimize a specified objective function or other search goals. Empirically, we observe that although the unknown gap exists between the theoretical and real boundaries of valid PID parameter configurations, the real boundary remains continuous and oscillates around the theoretical one. Therefore, we can leverage the coordinate search algorithm to depict the unknown gap with an efficient dynamic search. We start from the theoretical boundary derived by the Routh-Hurwitz Criterion and refine it with the gap identified by the coordinate search. In the end, we can efficiently construct the valid PID parameter boundary.

4 Methodology

4.1 Overview

The RouthSearch consists of two components: (1) Boundary Identification and (2) Misbehavior Validation, as illustrated in Figure 2. The boundary identification module first takes in PID parameters and their corresponding value range from the official documents of UAV flight control programs. It then efficiently searches the PID parameter space to find a classification boundary between valid and invalid configurations. During the search, the boundary identification module continuously queries the misbehavior validation module to determine whether a given PID parameter configuration is valid. In the end, it outputs the classification boundary.

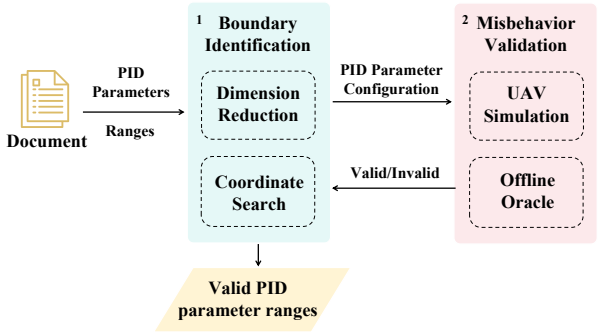


Fig. 2. RouthSearch Overview: RouthSearch includes Boundary Identification module and Misbehavior Validation module. The Boundary Identification module takes in PID parameters along with their ranges from the document and then explores the parameter space to find boundaries between valid and invalid configurations. The Boundary Identification module queries the Misbehavior Validation module to validate PID parameter configurations during the search process. Finally, it outputs valid PID parameter ranges.

4.2 Boundary Identification

In this section, we explain the efficient search algorithm that can identify the classification boundary within a three-dimensional PID parameter space. Our approach can generalize to diverse modes of mainstream flight control programs with external noise.

Dimension Reduction. We analyze the flight control program documentation to extract PID parameters, their corresponding value ranges, and incremental step sizes. We define the three-dimensional PID parameter space as $(k_p, k_i, k_d) \in [p_{\min}, p_{\max}] \times [i_{\min}, i_{\max}] \times [d_{\min}, d_{\max}]$, where k_p, k_i, and k_d denotes three PID parameter values. $p_{\min}$, $i_{\min}$, and $d_{\min}$ denotes lower bound of PID parameters. Similarly, $p_{\max}$, $i_{\max}$, and $d_{\max}$ denotes the upper bound of PID parameters. We represent the incremental step size as $step_p$, $step_i$, and $step_d$. We first divide the three-dimensional PID parameter space (k_p, k_i, k_d) into n two-dimensional planes (k_i, k_d). Each plane has a unique k_p value. The goal of such dimension reduction is that we can run our efficient coordinate search procedure as shown in Line 14-23 in Algorithm 1 at each two-dimensional plane in $O(n)$. Hence, the total runtime overhead of our approach is $O(n^2)$. While a brutal-force approach has to enumerate all the possible cases within the three-dimensional PID parameter space to find an accurate boundary, whose runtime overhead is $O(n^3)$.

As shown in Equation 6, the Routh-Hurwitz Criterion shows that a classification boundary separating the stable PID parameter values and unstable PID parameter values exists. On a plane (k_i, k_d) with $k_p = p_{const}$, we can have a linear boundary as follows:

$$(p_{const} + a_1)(k_d + a_2) = k_i \tag{7}$$

Note that a_1 and a_2 are unknown constants determined by the physical system and the specific flight mode. Ideally, we only need two data points of (k_i, k_d) to determine the value of a_1 and a_2, and further identify the linear boundary. However, real-world UAVs often have *complex non-linear* boundaries due to the unknown boundary shift (see Section 3). Figure 3b shows a noisy boundary of a control system highlighted in red. We can see that the boundary is a messy sawtooth wave rather than a clear linear boundary, while the valid/invalid parameter region remains approximately continuous with no disjoint regions observed empirically. To mitigate this problem, we design an efficient algorithm based on coordinate search that is robust to the boundary shift.

Algorithm 1: Boundary Identification Algorithm

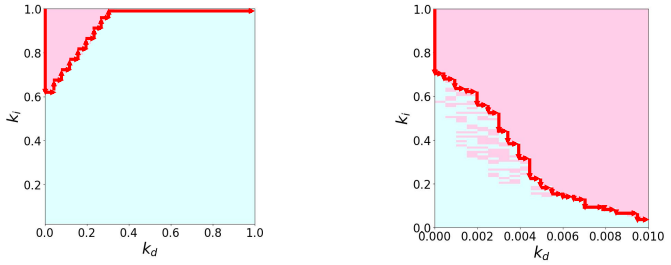
Input: PID range: (p_{min}, p_{max}), (i_{min}, i_{max}), (d_{min}, d_{max})
Incremental step: $\{step_p, step_i, step_d\}$
Flight Mission: M
Output: Classification boundary:
$BL = \{(p_1, i_1, d_1), (p_2, i_2, d_2), \cdots, (p_n, i_n, d_n)\}$

```
/* Search dimension I to find a valid config that is nearest to the partition line. */
Func Search (p, i, d, direction, step_i):
    is_valid = False
    while is_valid == False and i is valid do
        is_valid = Oracle(p, i, d, M)
        if direction == down then
            update = −step_i                          ▷ Search downwards
        else
            update = step_i                           ▷ Search upwards
        i = i + update
    i_save = i − update                               ▷ Restore i to last valid config
    return (p, i_save, d)
/* Dimension reduction: first fix P, then search (I, D) */
for p = p_min to p_max by step_p do
    /* Coordinate search over plane (I, D) */
    i = i_max                                         ▷ Start from upper-left corner
    for d = d_min to d_max by step_d do
        if Oracle(p, i, d, M) == False then
            (p, i_save, d) = Search(p, i, d, down, step_i)   ▷ Search dimension i downwards
        else
            (p, i_save, d) = Search(p, i, d, up, step_i)     ▷ Search dimension i upwards
        BL.Insert(p, i_save, d)                       ▷ Save boundary line
        i = i_save
```

Coordinate Search. We describe the algorithm using coordinate search to find the boundary on a given two-dimensional plane (k_i, k_d). In real-world UAVs, the boundary shape varies due to unpredictable external noises from the physical world and flight mode influence. For example, Figure 3a shows an approximately linear boundary as defined in Equation 7. But the boundary in Figure 3b turns into a drastically nonlinear sawtooth wave. Unlike conventional coordinate search aiming to find a particular data point [56], our approach identifies the entire boundary line composed of a sequence of continuous data points.

We start from an edge point (e.g., the upper-left corner) of the two-dimensional plane (k_i, k_d). Then, we fix one dimension and search for another by a small step size to generate a new (k_i, k_d) pair. We call the function `Search()` to perform this coordinate search. Note that the unknown gap can be extremely large, such that it distorts the slope direction of the theoretical boundary. We, therefore, adaptively search downwards and upwards in function `Search()` to ensure the discovery of the real boundary. After generating a candidate of the PID parameter configuration, we send it to the misbehavior validation module to check if it is valid. The feedback information from the oracle is used to guide the future search direction. Once we find the first valid PID control parameter nearest the partition line, we save it to the result list. Different from conventional coordinate search, our algorithm continues the search after finding the first partition point. In the end, our algorithm terminates when it reaches the boundary of the two-dimensional plane (k_i, k_d). Since it's an efficient coordinate search, the runtime overhead is $O(n)$, where n denotes the number of steps along each dimension.

Next, we explain the implementation details of our approach as shown in Algorithm 1. Lines 14-24 show the procedure of coordinate search. We iteratively search dimension k_d with step size $step_d$. We then search dimension k_i with the function `Search()` to find the valid PID control parameter that is nearest to the partition line and save it into the boundary line.



(a) Linear Boundary Line (b) Noisy Boundary Line

Fig. 3. Different PID parameter boundaries of UAVs. The light blue and pink represent valid and invalid PID configurations, respectively. We highlight the boundary line in **red**.

4.3 Misbehavior Validation

Our misbehavior validation module classifies a PID parameter configuration as invalid if it leads to abnormal UAV flight behavior. Accurate validation of such misbehavior is crucial for identifying the valid range of PID parameters. Inspired by PGFuzz [24], we used Metric Temporal Logic (MTL) [28] formula ϕ to precisely describe the specified behavior of UAVs from the document. If a UAV's flight behavior contradicts the MTL specification, the misbehavior validation module will classify the configuration as invalid. As Equation 8 shows, by evaluating the flight trajectory ω against the MTL formula ϕ, We can verify whether the UAV's behavior conforms to the expected norms. Symbol $\models$ indicates that trajectory ω conforms to MTL-formula ϕ.

$$Verif(\omega, \phi) = \begin{cases} true & \text{if } \omega \models \phi \\ false & \text{otherwise} \end{cases} \tag{8}$$

For example, we use MTL formula $\square\{(ALT_t < RTL\_ALT) \wedge (Mode_t = RTL) \rightarrow (ALT_{t-1} < ALT_t)\}$ to formally describe the expected UAV behavior during the ascending stage under ArduPilot `RTL` mode. ALT_t and ALT_{t-1} denote the altitude of the UAV at time steps t and $t-1$, respectively. RTL_ALT denotes a threshold defined in the `RTL` mode. This MTL formula states that when the UAV operates in `RTL` mode, and its altitude is below RTL_ALT, it should strictly increase its altitude. Any flight trajectory under ArduPilot `RTL` mode violating this MTL formula would be considered as UAV misbehaviors.

The misbehavior validation module comprises two main components: UAV simulation and offline oracle. The UAV simulation operates with a set of PID control parameters and predefined flight commands. During the simulation, we record the UAV's flight log. Next, we leverage an offline oracle to examine this log and check whether any misbehavior is observed.

We use the offline oracle because it can validate the entire flight trajectory and offer better validation accuracy than the online oracle. For flight modes that exhibit a long trajectory, the online oracle cannot accurately validate them. The reason is that the online oracle relies on a fixed-size sliding window, which cannot completely capture the UAV's flight trajectory. For example, ArduPilot's `Circle` mode requires the oracle to verify whether the flight trajectory exhibits a

repeating circle. The trajectory is too long to fit into a fixed-size sliding window. Consequently, the online oracle often yields wrong results when given a partial UAV's flight trajectory. In contrast, the offline oracle can verify the MTL against the entire trajectory, which results in better accuracy.

5 Evaluation

In this section, we evaluate the effectiveness of RouthSearch by answering the following research questions:

RQ1: Can RouthSearch identify the accurate range of valid PID parameter configurations given a flight mode?

RQ1 aims to evaluate the accuracy of the identified parameter range. In this context, we seek to determine whether there are any misconfigurations that our method fails to identify, as well as to verify if the PID configurations within the parameter region we've designated as invalid are indeed invalid.

RQ2: How does RouthSearch perform in spotting individual invalid PID parameter configurations?

RQ2 focuses on evaluating the efficacy of our method in discovering PID misconfigurations. As misconfigurations provide developers with valuable information about flight control program vulnerabilities, we want to evaluate our method's ability to detect misconfigurations.

RQ3: Has our strategy for mitigating the influence of noise improved the accuracy of identified boundaries?

RQ3 evaluates the effectiveness of the downward search strategy implemented within our coordinate search methodology. To mitigate the influence of noise, we incorporated this strategy into our search algorithm. Our objective is to quantify the extent to which this search strategy enhances the overall performance of our approach.

RQ4: How does the step size of the RouthSearch search algorithm affect the accuracy of boundary identification?

The purpose of RQ4 is to examine the relationship between step size and boundary identification accuracy in the RouthSearch algorithm. The step size parameter is hypothesized to have a dual effect: smaller values may enhance accuracy by allowing for finer-grained boundary delineation, while potentially increasing computational time and reducing efficiency. This study systematically examines how accurately boundaries are identified with different step sizes to find the best balance between precision and computational efficiency in the RouthSearch algorithm.

RQ5: Does the sampling-based evaluation influence the RouthSearch's performance?

RQ5 aims to assess the impact of sampling-based evaluation on the performance of RouthSearch. In our experiment, we sample the parameter space with a step size to evaluate the performance of RouthSearch due to the limited computational resources. This study investigates whether this sampling-based evaluation may affect the evaluation by comparing it against the exhaustive evaluation setting.

RQ6: Is offline oracle more accurate than online oracle in UAV misbehavior validation?

RQ6 evaluates the accuracy of an offline oracle over an online oracle in the validation of UAV misbehavior. The motivation for the offline oracle is that it improves the accuracy of validation by recording the flight states of all time steps, particularly in some flight modes (such as `AP:Circle`) where an accurate validation requires a long flight history. However, an online oracle with a sliding window cannot capture the entire flight trajectory and lead to an accurate validation. We compare the accuracy of the online and offline oracle.

RQ7: Can other search algorithms (e.g., hill climbing and genetic algorithm) achieve comparable performance to coordinate search for boundary identification and misconfiguration detection?

RQ7 investigates the application of other search algorithms (e.g., hill climbing and genetic algorithms) in RouthSearch. In our evaluation, we found coordinate search performed empirically the best among all three search algorithms in RouthSearch.

5.1 Experimental Settings

We conducted four studies to address our research questions (RQs). Each study is designed to provide insights into specific aspects of our method's performance and effectiveness. We conducted an experiment on an openEuler release 20.03 LTS machine with 80 CPU cores (16 processors), 251 GB RAM, and 870 GB disk space.

Subjects. We studied two popular flight control programs: ArduPilot (ArduCopter V4.4.1)[1] and PX4 (V1.15.0 a1cce7e) [39]. We evaluated our method in the SITL Simulator [2] of ArduPilot and the jMAVSim [38] of PX4 with default settings. Our evaluation evaluated eight typical flight modes, encompassing both simple and complex flight tasks. From ArduPilot, we selected `RTL` [48], `Zigzag` [49], `Circle` [47], and `Brake` [46]. From PX4, we chose `Orbit` [52], `Return` [53], `Land` [51], and `Hold` [50].

Experiment 0: Obtain Ground Truth. To evaluate the accuracy of these identified boundaries, we obtain the ground truth benchmark of the invalid PID parameter configurations. The ground truth was obtained through a systematic exploration of the PID parameter space available in the document for each flight mode. Each PID parameter configuration was tested within its corresponding flight mode. Our misbehavior validation module then classified each configuration as valid or invalid. This process, requiring approximately 7000 CPU hours, yielded a labeled benchmark of PID parameter configurations for each flight mode.

Experiment 1: Boundary Identification. This experiment addresses RQ1. We utilize RouthSearch to identify the boundaries of invalid PID parameter regions. To evaluate the accuracy of these identified boundaries, we compare the invalid PID parameter configurations within these regions against the ground truth benchmark established in experiment 0. We choose this approach over comparing our method with other leading methods such as RVFuzzer and LGDFuzzer. We do not select RVFuzzer as a baseline because it is not open-sourced. Meanwhile, RVFuzzer's problem setting is not compatible with ours as it only outputs the value range for a single PID parameter, while our problem requires identifying a multi-dimensional PID parameter range. We do not select LGDFuzzer as our baseline, because in the context of a multi-dimensional parameter space, LGDFuzzer outputs the Cartesian product of these single-dimensional ranges. This results in a cuboid-like invalid parameter region, which does not match our polyhedron-like invalid parameter region in our problem definition. As a result, they are not suitable as our baselines.

Experiment 2: Misconfiguration Discovery. This experiment addresses RQ2 by evaluating the effectiveness of our method in discovering PID parameter misconfigurations. We compare our method with PGFuzz, a state-of-the-art misconfiguration detection method. Both methods were run for 48 hours, and their performance was assessed based on the number of misconfigurations discovered. Our method utilizes identified boundaries of invalid PID parameter regions. Each parameter within these regions is tested under eight flight modes, and our misbehavior validation module classifies it as either a misconfiguration or a valid configuration. PGFuzz uses a random search with simple guidance to find misconfigurations. For a fair comparison, we align PGFuzz's search space with ours, excluding parameters irrelevant to PID control modules. We also replace PGFuzz's online oracle with our offline misbehavior validator. We ran the PGFuzz experiment three times to ensure robust results, given the variability of the random method.

Experiment 3: Design Choice. This experiment addresses RQ3. We evaluate the impact of the downward search strategy on identifying valid PID parameter regions. We developed a variant method that omits the downward search during coordinate search and applied both the original and

variant methods to eight flight modes. We then compared the boundaries identified by each method. This comparison assesses the downward search strategy's contribution to boundary accuracy.

Experiment 4: Step Size Analysis. This experiment addresses RQ4, focusing on step size impact. We modified the original incremental step size by factors of 10x and 100x and then applied our search algorithm with these modified step sizes. We compare the resulting Miss Rate and Hit Rate to those obtained using the original step size. The comparison helps quantify the step size's influence on search accuracy and efficiency.

Experiment 5: Sample-based Evaluation Influence. This experiment addresses RQ5. We exhaustively evaluated RouthSearchon all possible configurations of ArduPilot `RTL` mode. We then compared the miss rate and hit rate obtained from that exhaustive experiment setting with the rates from the sampling-based experiment setting.

Experiment 6: Justification for Offline Oracle Design. This experiment addresses RQ6. We compared the accuracy of the offline oracle and online oracle to justify our selection of the offline oracle. We execute flight tasks across 8 flight modes under various PID configurations, and use online and offline oracles, respectively, to validate their behaviors. We calculated the accuracy of the different oracles by comparing the results of different oracle settings with human-verified results.

Experiment 7: Investigating Other Search Algorithms. This experiment addresses RQ7. We replaced the Coordinate Search component of RouthSearch with Hill Climbing (HC) and Genetic Algorithm (GA), creating RouthSearch-HC and RouthSearch-GA variants. These variants were applied to identify boundaries and detect misconfigurations with the same experiment settings of the original RouthSearch. We compare the miss rate, the hit rate, and the number of detected misconfigurations of two variants and the original RouthSearch.

5.2 Evaluation Metrics

We use Miss Rate and Hit Rate to evaluate our method. Let GT represent the set of all misconfigurations in the ground truth, and RS denotes the set of all misconfigurations generated by the RouthSearch method proposed in this study. The metrics employed in this paper are as follows:

MR. The Miss Rate (MR) quantifies the extent to which the boundaries identified by the proposed method fail to capture misconfigurations. A lower miss rate indicates a higher accuracy in identifying boundaries. It is defined as:

$$MR = \frac{|GT - RS|}{|GT|}$$

HR. The Hit Rate (HR) measures the accuracy of the misconfigurations generated according to the boundaries identified by the proposed method. A higher hit rate signifies a more efficient generation of misconfigurations during the invalid configuration generation phase. It is defined as:

$$HR = \frac{|RS \cap GT|}{|RS|}$$

5.3 Results: Experiment 1

Table 1 shows the boundary identification performance of RouthSearch. Column "Total" shows the total ground truth misconfigurations. "Identified" is the count of misconfigurations identified by RouthSearch. "Accurate" is the count of correctly identified misconfigurations. "MR" and "HR" are the miss rate and hit rate, respectively.

The result shown in Table 1 demonstrates our method's high effectiveness in boundary identification. The average hit rate (HR) of misconfigurations generated by our method is 92.0%. This means that, on average, most of the PID parameter misconfigurations identified by RouthSearch are

Table 1. Boundary identification accuracy of RouthSearch. This table shows the accuracy of the misconfigurations identified by the boundaries for different flight modes in ArduPilot (AP) and PX4 systems, measured in terms of total misconfigurations, identified misconfigurations, accurate detections, miss rate (MR), and hit rate (HR). Higher HR and lower MR indicate better detection capability.

Mode	Total	Identified	Accurate	MR	HR
AP:Zigzag	28,162	21,107	19,367	31.2%	91.8%
AP:Brake	36,584	33,357	32,902	10.1%	98.6%
AP:RTL	27,267	26,624	26,545	2.6%	99.7%
AP:Circle	20,477	23,016	17,022	16.9%	74.0%
PX4:Orbit	8,979	8,103	6,630	26.2%	81.8%
PX4:Return	9,592	8,604	8,429	12.1%	98.0%
PX4:Land	9,610	9,123	8,885	7.5%	97.4%
PX4:Hold	4,930	4,394	4,165	15.5%	94.8%
Average	18,200	16,791	15,493	15.3%	92.0%

correct. The miss rate (MR), representing misconfigurations not detected by our method, averages 15.3%. In other words, only a few PID parameter misconfigurations are not detected by our method. The high average Hit rate and the low average Miss Rate demonstrate the high accuracy of our boundary identification method.

Notably, in ArduPilot's `RTL` mode, RouthSearch achieves exceptional performance with a 99.7% hit rate and only a 2.6% miss rate, significantly surpassing the overall average performance of 92.0% hit rate and 15.3% miss rate across all flight modes. This indicates that our method is particularly effective in identifying PID parameter misconfigurations in this specific flight mode, with `RTL` mode showing the most precise detection capabilities among all tested scenarios.

The number of PID parameter misconfigurations varies across flight modes, suggesting the boundary shift of invalid PID parameter regions under different flight modes. This variation highlights the importance of considering flight modes when identifying PID parameter misconfigurations. A significant boundary shift is observed between different flight modes within ArduPilot: ArduPilot's `Brake` mode exhibits the highest number of invalid PID parameter configurations (36,584), while the `Circle` mode shows the lowest (20,477). This difference suggests that the `Brake` mode may be more sensitive to PID parameter misconfigurations compared to the `Circle` mode.

Result 1: RouthSearch achieves an average 92.0% hit rate and 15.3% miss rate across various flight modes. This result indicates the effectiveness of our method in accurately identifying PID parameter misconfigurations.

5.3.1 Root Cause Analysis of Worst Case. The performance of our method in the worst-case scenario is primarily influenced by two factors: the settling time of the PID control system [35] and the misbehavior validator's sensitivity.

Long Settling Time. ArduPilot's `Zigzag` mode exhibits an elevated MR (31.2%), attributed to long settling times. The Routh-Hurwitz criteria guarantee stability as time approaches infinity but do not account for settling time, defined as the time for a system to stabilize [35]. Small P values can lead to prolonged settling times, causing the system to slowly reach or even fail to reach the target [32, 57]. Consequently, our proposed method may overlook misconfigurations caused by these long settling times.

Over-Sensitive Misbehavior Validator. Our method, when applied to PX4's `Orbit` mode and ArduPilot's `Circle` mode, exhibits a low hit rate (HR) of 81.8% and 74.0%. This discrepancy arises

from the misbehavior validator's sensitivity to the inherent deviations present in `Orbit` mode trajectories. While `Orbit` mode ideally maintains a circular flight path, factors like sensor inaccuracies introduce minor deviations that, although practically insignificant, impact the detector's ability to differentiate between valid and invalid trajectories. Consequently, valid PID parameter configurations are misclassified as invalid, as illustrated in Figure 4, leading to an inaccurate boundary identified by our algorithm.

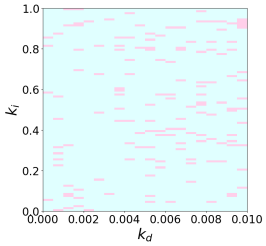


Fig. 4. False negative reported by the misbehavior validator under PX4 `Orbit` mode. The pink marks false positive misconfigurations caused by the detector's sensitivity to deviation.

5.4 Results: Experiment 2

Table 2 compares misconfiguration detection by RouthSearch and PGFuzz. Column "RouthSearch" is the number of RouthSearch's detections, and "$PGFuzz_{\{1,2,3\}}$" are PGFuzz detections over three runs.

Table 2. Comparison of number of misconfigurations discovered by RouthSearchand PGFuzz under three runs ($PGFuzz_{\{1,2,3\}}$) across flight modes in ArduPilot (AP) and PX4. Higher numbers indicate better performance.

Mode	RouthSearch	$PGFuzz_1$	$PGFuzz_2$	$PGFuzz_3$
AP:Zigzag	3,390	557	617	518
AP:Brake	11,820	406	289	262
AP:RTL	4,969	1,319	1,325	1,286
AP:Circle	3,533	93	78	76
PX4:Orbit	1,909	290	236	207
PX4:Return	1,092	299	278	256
PX4:Land	2,037	635	495	576
PX4:Hold	2,070	305	193	171
Average	3,853	488	439	419

As shown in Table 2, regarding misconfiguration generation, RouthSearch demonstrates superior effectiveness compared to the PGFuzz method. Our method discovered an average of 3,853 PID parameter misconfigurations, whereas PGFuzz detected a modest 449 misconfigurations (averaged across three experimental iterations). In the most successful instance, our method discovered 11,820 PID parameter misconfigurations in ArduPilot's `Brake` mode. In contrast, PGFuzz discovered an average of only 319 PID parameter misconfigurations, based on results from three iterations (406, 289, and 262 misconfigurations, respectively).

Result 2: RouthSearch outperforms PGFuzz in PID parameter misconfiguration discovery, discovering 8.58 times more misconfigurations (3,853 vs 449 on average).

5.5 Results: Experiment 3

Table 3 shows the misconfiguration detection comparison between RouthSearch and RouthSearch-DSOff (Downward Search Off). Column "Total" represents the number of misconfigurations in ground truth, Column "RouthSearch-DSOff" and Column "RouthSearch" show the number of detected misconfigurations, Miss Rate, and Hit Rate for each.

Table 3. The total number of misconfigurations and accuracy discovered by RouthSearch and variant RouthSearch-DSOff (Downward Search Off) for eight flight modes. Higher numbers, higher HR, and lower MR indicate better detection capability.

Mode	Total	RouthSearch-DSOff			RouthSearch		
		Number	MR	HR	Number	MR	HR
AP:Zigzag	28,162	531	98.1%	97.3%	19,367	31.2%	91.8%
AP:Brake	36,584	32,515	11.1%	98.2%	32,902	10.1%	98.6%
AP:RTL	27,267	26,531	2.7%	99.7%	26,545	2.6%	99.7%
AP:Circle	20,477	15,678	23.4%	93.1%	17,022	16.9%	74.0%
PX4:Orbit	8,979	3,779	57.9%	99.9%	6,630	26.2%	81.8%
PX4:Return	9,592	5,340	44.3%	99.7%	8,429	12.1%	98.0%
PX4:Land	9,610	6,737	29.9%	98.7%	8,885	7.5%	97.4%
PX4:Hold	4,930	3,175	35.6%	99.5%	4,165	15.5%	94.8%
Average	18,200	11,786	37.9%	98.3%	15,493	15.3%	92.0%

As shown in Table 3, RouthSearch achieves significantly better results than RouthSearch-DSOff in terms of the number of identified misconfigurations and the miss rate. On average, RouthSearch identifies 15,493 misconfigurations and a 15.3% miss rate. RouthSearch-DSOff identifies 11,786 misconfigurations and 37.9% miss rate. That means the downward search strategy can significantly decrease the miss rate (from 37.9% to 15.3%). RouthSearch-DSOff's hit rate slightly improved because the RouthSearch-DSOff doesn't explore the noise area, where invalid and valid configurations are mixed. In other words, the downward search strategy is effective in identifying configuration boundaries.

Result 3: The downward search strategy is effective in identifying configuration boundaries, decreasing the miss rate from 37.9% to 15.3%.

5.6 Results: Experiment 4

Table 4. The influence of step size in algorithm 1 on the misconfiguration detection and boundary identification accuracy of RouthSearch. The step size ranges from 1x (finest) to 100x (coarsest). Results are measured in terms of total misconfigurations, identified misconfigurations, accurate detections, miss rate (MR), and hit rate (HR). Higher HR and lower MR indicate better performance.

Step Size	Total	Identified	Accurate	MR	HR
1x	374,578	330,727	325,300	13.2%	98.4%
10x	36,584	33,357	32,902	10.1%	98.6%
100x	4,012	3,812	3,673	8.4%	96.4%

Table 4 examines how step size affects detection performance. The "Total" column represents the overall misconfiguration ground truth. "Identified" indicates the count of misconfigurations RouthSearch invalidates. "Accurate" reflects the number of correctly detected invalid misconfigurations. "MR" and "HR" present the Miss Rate and Hit Rate evaluation metrics.

The results reveal that the 10x scale step size achieves the highest hit rate (98.6%), while the 100x scale step size yields the lowest (96.4%). However, the difference between these extremes is merely 2.2%. Regarding miss rate, the 100x scale step size performs best (8.4%), with the 1x scale step size showing the poorest performance (13.2%). The difference here is 4.8%. These relatively small variations indicate that step size has a limited impact on the method's overall performance.

Considering both accuracy and computational efficiency, we empirically determined that a 10x scale step size provided the optimal trade-off for our experiments. This choice balances the high hit rate (98.6%) with a reasonably low miss rate (10.1%), while significantly reducing the configurations to be tested compared to the 1x scale.

Result 4: The maximum differences in the MR and HR across various step sizes were 4.8% and 2.2% respectively. These small variations indicate that our method's performance is relatively insensitive to step size, with the 10x scale step size offering the most balanced selection.

5.7 Results: Experiment 5

Table 5 presents a comparison between computing HR and MR using exhaustive and sampling evaluation. "Exhaustive" (evaluation over all possible configurations) and "Sampling-based" (evaluation over sampled configurations). Column "CPU hours" means total runtime.

Table 5. Exhaustive evaluation vs Sampling-based evaluation; Exhaustive means evaluation over all possible configurations; Sampling-based means evaluation over sampled configurations. CPU hours mean total runtime.

Mode	Experimental Setting	MR	HR	CPU hours
AP: RTL	Exhaustive	6.7%	98.9%	9,067
	Sampling-based	2.6%	99.7%	907

As Table 5 shows, we found that the difference in MR and HR is marginal, which indicates that the sampling-based evaluation is reliable. The variance in MR (6.7% vs 2.6%) and HR (98.9% vs 99.7%) is minor. Meanwhile, our sampling-based evaluation requires only one-tenth total runtime compared with exhaustive evaluation.

Result 5: Our sampling-based evaluation is reliable with minimal influence on the MR and HR, while also requiring only one-tenth of the total runtime compared with exhaustive evaluation.

5.8 Results: Experiment 6

Table 6 compares agreement rate with expert classification (%) of "Online" and "Offline" Oracles. Higher percentages indicate better performance.

Table 6. Online/Offline method Detection Accuracy Comparison (evaluated by agreement rate with human-verified results).

Mode	Online	Offline	Mode	Online	Offline
AP:Brake	96.5%	99.4%	PX4:Hold	100%	100%
AP:Circle	72.7%	96.5%	PX4:Land	100%	100%
AP:RTL	99.7%	99.7%	PX4:Return	100%	100%
AP:Zigzag	97.1%	97.1%	PX4:Orbit	28.5%	97.3%

Table 6 shows that the offline oracle generally outperforms the online oracle for UAV misbehavior validation, due to its use of the entire trajectory context. For the long-term `PX4:Orbit` mode, the offline oracle is significantly better, reaching 97.3% agreement compared to online's 28.5% (68.8% higher). This gap in `PX4:Orbit`, a long-duration mode, highlights the online oracle's limited window size, failing to capture the long-term context needed for complex misbehavior validation. The offline oracle overcomes this by using the entire trajectory. For short-term flight mode, the offline oracle's accuracy is also on par with the online oracles' accuracy. For instance, both online and offline oracles achieve high agreement (96.5%-100%) in modes like `AP:Brake` and `PX4:Hold/Land/Return`. In conclusion, Table 6 empirically justifies offline oracle for RQ6, as it better detects UAV misbehavior, especially those requiring the entire trajectory context other than limited slide windows.

Result 6: The offline oracle outperforms the online oracle for UAV misbehavior validation, especially for long-term patterns (`PX4:Orbit` 97.3% vs 28.5% agreement rate, 68.8% higher). For short-term patterns, both methods show similar high performance (96.5–100%).

5.9 Results: Experiment 7

Table 7 compares the performance of RouthSearch variants: Coordinate Search (CS), Hill Climbing (HC), and Genetic Algorithm (GA). The columns show Miss Rate (MR), Hit Rate (HR), and the number of misconfigurations identified ('number') for each method.

Table 7. Comparison performance between RouthSearch-CS(Routh Search with Coordinate Search), RouthSearch-HC(Routh Search with Hill Climbing) and RouthSearch-GA(Routh Search with Genetic Algorithm). Higher numbers, higher HR, and lower MR indicate better detection capability.

Mode	RouthSearch-CS			RouthSearch-HC			RouthSearch-GA		
	MR	HR	number	MR	HR	number	MR	HR	number
AP:Zigzag	31.2%	91.8%	3390	98.3%	25.1%	481	93.0%	20.2%	1928
AP:Brake	10.1%	98.6%	11,820	93.5%	49.3%	2373	89.9%	35.5%	3688
AP:RTL	2.6%	99.7%	4,969	95.0%	37.9%	1379	88.4%	30.3%	3204
AP:Circle	16.9%	74.0%	3,533	95.4%	45.6%	934	90.2%	19.1%	1874
PX4:Orbit	26.2%	81.8%	1,909	98.0%	22.8%	173	96.4%	42.2%	319
PX4:Return	12.1%	98.0%	1,092	98.3%	37.6%	162	96.4%	40.3%	345
PX4:Land	7.5%	97.4%	2,037	96.3%	40.8%	358	95.8%	39.3%	401
PX4:Hold	15.5%	94.8%	2,070	96.8%	21.9%	158	94.5%	29.8%	270
Average	13.0%	92.0%	3853	96.5%	35.1%	752	93.1%	32.1%	1504

Table 7 demonstrates that Coordinate Search(CS) is better than Hill Climbing (HC) and Genetic Algorithms (GA) in RouthSearch. CS only requires binary feedback to decide on whether to search upward or downward, which aligns perfectly with the binary feedback, achieving near-perfect precision (e.g., 2.6% MR and 99.7% HR in `AP:RTL`) and 2.5 times higher misconfiguration counts. In contrast, HC exhibits excessively high miss rates (e.g., 98.3% MR in `AP:Zigzag`) and low hit rates (e.g., 21.9% HR in `PX4:Hold`). This limitation arises because hill climbing lacks quantitative feedback, degenerating into a local random search that produces narrow, inaccurate boundaries. GA also exhibits high miss rates and low hit rates. Although GA is slightly better than HC (for example, 89. 9% vs. 93. 5% in AP: Brake) through a greater exploration, but remains trapped in undirected randomness due to missing fitness-guided refinement, resulting in persistently low HR (19.1–42.2%) and suboptimal misconfiguration detection (max: 3,688 vs. 11,820 for RouthSearch-CS).

Result 7: Coordinate search significantly surpasses hill climbing and genetic algorithms, achieving superior HR and MR and detecting 2.5-5 times more misconfigurations on average. This result validates the selection of coordinate search due to its effective use of binary feedback.

5.10 Case Study

By utilizing our method in the ArduPilot and PX4 flight control programs, We obtained many failed cases and spotted several bugs. Here are some bug cases we have found.

5.10.1 Bug 1. As the ArduPilot document said, the `Brake` mode stops the vehicle as soon as possible. Once invoked, this mode does not accept any input from the pilot. We have found that PID misconfigurations could cause the `Brake` mode to fail to stop the vehicle and move around. After misconfiguring the PID parameters of the ArduPilot position control sub-module into $(0.1, 0.8, 0.8)$, we commanded the UAV to fly for a certain distance and then issued a command to enter `Brake` mode. The UAV attempted to stop quickly and regain its balance. However, due to the misconfiguration, the more the flight control program attempts to utilize the PID control module to quickly bring the UAV to a stop, the more the unstable PID control module fails to achieve the objective. This results in the UAV continuously rotating, as shown in Figure 5.

To make it even worse, due to the refusal of `Brake` mode to accept commands sent by the sticks of the transmitter. Thus, the user can't stop the UAV manually while the UAV is in a perilous state. As Listing 1 shows, ArduPilot is trying to stop the UAV by setting a zero velocity as the target to the PID control module. While the UAV is unstable because of the misconfiguration, the velocity is always greater than zero, which results in a rotating UAV that never stops. This issue represents a logical flaw in the system, where there is inadequate error handling for scenarios in which the speed fails to reach zero.

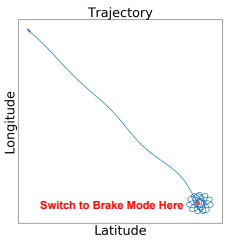


Fig. 5. UAV under ArduPilot `Brake` mode fails to stop at the destination marked in **red** and gets stuck in circles endlessly due to a logic bug in the flight control program.

```
void brake_mode() {
    while(true) {
        pos_pid_control.set_target_velocity(0);
        /*Logical Bug: Missed error handling.*/
        sleep(0.1);
    }
}
```

Listing 1. The logical bug in `Brake` mode of ArduPilot. The ArduPilot tries to stop the UAV by setting a zero velocity as the target to the PID control module, but the unstable UAV fails to achieve that due to the misconfiguration. That results in the UAV continuously rotating, because the flight control program forgets to handle the error case when the function call `set_target_velocity(0)` fails.

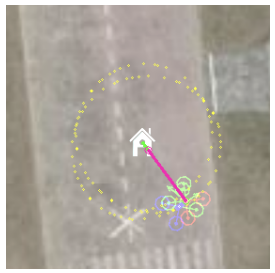


Fig. 6. Flight trajectory (marked in yellow) of a misconfigured UAV (marked in green) under RTL mode. The UAV, stuck in a circle around the home location and failed to land, was caused by misconfigured PID parameters.

```
1  // Check if the UAV reaches the destination
2  bool reached_dst() {
3      dist = compute_dist(curr_loc, dst);
4      if(dist < radius)
5          return true;
6      return false;
7  }
8
9  // Attempt to land after the UAV reaches the destination
10 void switch_to_land() {
11     state_complete = reached_dst();
12     if(state_complete)
13         perform_land();
14     /* Logical Bug: Missed error handling. */
15 }
```

Listing 2. Logical bug in ArduPilot RTL mode. ArduPilot checks the UAV's approach to the descent location by calculating the distance to it. Misconfigured PID parameters cause the UAV to circle continuously, preventing it from getting close enough to reduce the distance under threshold `radius`. In that case, `state_complete` is always false. The flight control program forgets to handle the error case. Therefore, the landing action is not invoked, and the UAV fails to land at the home location.

5.10.2 Bug 2. RTL (`Return-To-Launch`) is an ArduPilot flight mode that commands the UAV to fly towards its home location and land after hovering above for a while. A logic bug occurs in RTL mode when users set the PID parameters with a value that would cause the UAV to misbehave. Specifically, users set the PID parameters of the position control sub-module as $(0.1, 0.8, 0.4)$, all of which fall within the value range of the PID control parameter in the official document of ArduPilot. However, these PID parameters accidentally lead to an unstable control state of the UAV under RTL mode. As a result, the UAV kept circling around the home location without reaching a suitable descending spot. Figure 6 shows the flight trajectory (marked in yellow) of the UAV (green). Given that the goal of RTL mode is to perform a landing at the home location, the misconfigured UAV under RTL mode is trapped in a circle and fails to accomplish the landing task.

Listing 2 shows the logical bug in ArduPilot. In Line 2, the function `reached_dst()` checks if the UAV reaches a suitable descent location by measuring the distance between the UAV's current position and the descent location. The distance is shown in **purple** in Figure 6. Due to misconfigured PID control parameters, the UAV is stuck in a circle and fails to reduce the distance under the threshold `radius`. In Line 12, the `state_complete` is always false. The landing action in Line 13 is, therefore, not invoked. Ultimately, the UAV hovers in a circle and fails to land at the home location. It's worth noting that we have already brought this issue to the attention of the developers, who have acknowledged the bug.

5.10.3 Bug 3. As the PX4 document said, the `Hold` mode causes the vehicle to stop and maintain its current position and altitude [50]. As Figure 7 shows, we have found that PID misconfigurations could cause the `Hold` mode to fail to hold the UAV's position and attitude and lead the UAV to crash to the ground. After commanding the UAV to take off to a certain altitude, we misconfigured the PID parameters of the PX4 attitude control submodule into $(0.01, 0.1, 0.0005)$ and then issued a command to enter `Hold` mode. As Figure 7 shows, once we have misconfigured PID parameters, the UAV becomes unstable and quickly crashes to the ground due to the loss of balance.

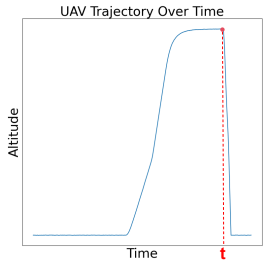


Fig. 7. UAV trajectory of PX4 under `Hold` mode. UAV crashes after **time t** when a user misconfigures a PID parameter.

5.10.4 Discussion. In three 48-hour experiment runs, RouthSearch exclusively discovered Bugs 1 and 2, which PGFuzz failed to discover. While PGFuzz occasionally found Bug 3 in only two of the three runs and required 8.86 hours for detection in those instances, RouthSearch consistently identified the same bug in just 1.5 minutes. Furthermore, RouthSearchdiscovered 2, 002 PID misconfigurations, compared to the 7 found by PGFuzz. The random and isolated parameter mutation of PGFuzz hindered the detection of multidimensional trigger conditions (required by Bugs 1 and 2). In contrast, RouthSearch's principled PID parameter analysis and theoretical guidance enabled systematic misconfiguration detection. Bug 3, requiring a single-dimensional trigger (a low value of p), can be occasionally found by PGFuzz, but RouthSearch can detect it more consistently and efficiently.

6 Limitations and Discussion

Handling Misconfigurations Caused by Long Settling Time. The settling time problem occurs when both the p value and i value are simultaneously small, the PID control module fails to achieve the desired output, and causes the system under control to move too slowly to reach the target. That results in PID misconfigurations that our method may overlook. A candidate fix is to examine the configurations where both the p value and i value are simultaneously small in pre-processing.

Misbehavior Validator's Sensitivity. The detector's over-sensitivity affects boundary identification accuracy. It sometimes misclassifies valid PID configurations as invalid. This can lead to inaccurate boundary identification in corner cases. A potential solution is to use multiple misbehavior validators and perform majority voting. This approach could mitigate the impact of deviations on the detector's output.

Trade-off between Accuracy and Efficiency. A large search step size in boundary identification improves RouthSearch's efficiency but may reduce boundary accuracy. Conversely, a small step size increases accuracy at the expense of time. Selecting an appropriate step size that balances the trade-off between these requirements should be considered in practice.

7 Related Works

PID Parameters Optimization. In the field of control theory, researchers have long focused on strategies for selecting optimal PID control parameters. Classical approaches, including Ziegler-Nichols [32], Åström-Hägglund [57], and Cohen-Coon [22], are widely employed to determine suitable PID control parameters within specified ranges for various control systems. Building upon these classical methods, more sophisticated techniques have emerged to address diverse control challenges. Computational intelligence approaches, such as genetic algorithms [18, 20] and particle swarm optimization [16, 23], offer robust tuning strategies for complex systems where traditional methods may fall short. Data-driven tuning approaches [21, 43] generate tuning strategies through learning processes, combining system models and operational data to optimize control parameters. Recent advancements [13, 19, 37] focus on robust and adaptive tuning strategies to manage system uncertainties and time-varying dynamics, addressing limitations of static tuning methods.

Based on these theories, software developers have implemented numerous tools for fine-tuning PID control parameters across various applications, such as the `AutoTune mode` [45] in ArduPilot. Nevertheless, these tools may pose challenges for normal users lacking substantial expertise in tuning control parameters [40].

UAV Parameter Misconfiguration Detection. The misconfiguration of parameters in UAVs frequently results in abnormal flight behavior, jeopardizing flight safety. In recent years, ensuring the flight safety of unmanned aerial vehicles (UAVs) has become a focal point of considerable attention from researchers [10, 36, 54].

Fuzz testing has been adapted for UAVs to uncover bugs arising from misconfigured parameters. Unlike traditional software fuzzing that detects crashes or memory corruption, bugs in UAV systems typically manifest in distinct error types [26, 41]. Recent UAV fuzzing methods address this using policies based on physical behavior to guide testing [8, 25]. PGFuzz [24], for example, implements temporal logic policies to guide the search for erroneous UAV parameters. These policy-driven fuzzing methods excel in uncovering bugs that conventional methods may overlook. Concurrently, learning-guided fuzzing techniques, used in testing Cyber-Physical Systems [7, 58], are also used to identify UAV parameter bugs. LGDFuzzer [15], for instance, employs a learned model that effectively guides the search.

8 Conclusion

This paper proposes RouthSearch, a novel method leveraging Routh-Hurwitz stability and coordinate search to accurately determine the valid 3D ranges for critical PID parameters in the UAV flight control program. RouthSearch identifies boundaries between valid and invalid PID parameter configurations, helping prevent users from misconfiguring PID parameters during flight. Evaluated on ArduPilot and PX4 across 8 flight modes, it achieved 92.0% accuracy, detected 3,850 misconfigurations per 48 hours (significantly outperforming prior work), and uncovered 3 bugs.

9 Data Availability

We provide our tool RouthSearch, an implementation of the boundary identification module and misbehavior validation module in web at https://github.com/SciC0d3m4xOfW/RouthSearch.

Acknowledgments

This work was supported by the National Key R&D Program of China under Grant No. 2022YFB4501803. This work was also supported by the National Natural Science Foundation of China (Grant No. 62472100).

References

[1] ArduPilot Dev Team. 2024. ArduPilot - Versatile, Trusted, Open. https://ardupilot.org/. Accessed: 2024-10-29.

[2] ArduPilot Dev Team. 2024. SITL Simulator (Software in the Loop). https://ardupilot.org/dev/docs/sitl-simulator-software-in-the-loop.html. Accessed: 2024-10-29.

[3] ArduPilot Discourse. 2024. Wild ride after successful Auto-tune - Copter 4.4 - ArduPilot Discourse. https://discuss.ardupilot.org/t/wild-ride-after-successful-auto-tune/113718. Accessed: 2024-10-29.

[4] Karl Johan Åström, Chang C Hang, and Per Persson. 1989. Towards intelligent PID control. *Annual Review in Automatic Programming* 15 (1989), 53–58. doi:10.1016/B978-0-08-040185-0.50014-2

[5] Australian Transport Safety Bureau (ATSB). 2023. Docklands drone swarm accident. https://www.atsb.gov.au/media/2023/docklands-drone-swarm-accident. Accessed: 2023-11-17.

[6] Jose J Castillo-Zamora, Jose E Hernández-Díez, Islam Boussaada, Juan Escareno, and Jonatan U Alvarez-Muñoz. 2021. A preliminary parametric analysis of pid delay-based controllers for quadrotor uavs. In *2021 International Conference on Unmanned Aircraft Systems (ICUAS)*. IEEE, Athens, Greece, 28–37. doi:10.1109/ICUAS51884.2021.9476821

[7] Yuqi Chen, Christopher M Poskitt, Jun Sun, Sridhar Adepu, and Fan Zhang. 2019. Learning-guided network fuzzing for testing cyber-physical system defences. In *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, San Diego, CA, USA, 962–973. doi:10.1109/ASE.2019.00093

[8] 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 2020 ACM SIGSAC Conference on Computer and Communications Security*. Association for Computing Machinery, New York, NY, USA, 263–278. doi:10.1145/3372297.3417249

[9] Jaime Del Cerro, Christyan Cruz Ulloa, Antonio Barrientos, and Jorge de León Rivas. 2021. Unmanned aerial vehicles in agriculture: A survey. *Agronomy* 11, 2 (2021), 203. doi:10.3390/agronomy11020203

[10] Andrea Di Sorbo, Fiorella Zampetti, Aaron Visaggio, Massimiliano Di Penta, and Sebastiano Panichella. 2023. Automated identification and qualitative characterization of safety concerns reported in uav software platforms. *ACM Transactions on Software Engineering and Methodology* 32, 3 (2023), 1–37. doi:10.1145/3564821

[11] Isabelle Fantoni, Rogelio Lozano, and Farid Kendoul. 2008. Asymptotic stability of hierarchical inner-outer loop-based flight controllers. *IFAC Proceedings Volumes* 41, 2 (2008), 1741–1746. doi:10.3182/20080706-5-KR-1001.00298

[12] FossilRider et al. 2024. Unstable Flight While Attempting PID Tweak Ending In Catastrophic Crash. https://discuss.ardupilot.org/t/unstable-flight-while-attempting-pid-tweak-ending-in-catastrophic-crash/118665. Accessed: 2023-10-26.

[13] Seyyed Morteza Ghamari, Fatemeh Khavari, and Hasan Mollaee. 2023. Lyapunov-based adaptive PID controller design for buck converter. *Soft Computing* 27, 9 (2023), 5741–5750. doi:10.21203/rs.3.rs-2354753/v1

[14] 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. doi:10.1109/TSE.2024.3354739

[15] Ruidong Han, Chao Yang, Siqi Ma, JiangFeng Ma, Cong Sun, Juanru Li, and Elisa Bertino. 2022. Control parameters considered harmful: Detecting range specification bugs in drone configuration modules via learning-guided search. In *Proceedings of the 44th International Conference on Software Engineering*. Association for Computing Machinery, New York, NY, USA, 462–473. doi:10.1145/3510003.3510084

[16] Teguh Herlambang, Dinita Rahmalia, and T Yulianto. 2019. Particle swarm optimization (pso) and ant colony optimization (aco) for optimizing pid parameters on autonomous underwater vehicle (auv) control system. In *Journal of Physics: Conference Series*, Vol. 1211. IOP Publishing, East Java, Indonesia, 012039. doi:10.1088/1742-6596/1211/1/012039

[17] Ming-Tzu Ho, Aniruddha Datta, and Shankar P Bhattacharyya. 1998. An elementary derivation of the Routh-Hurwitz criterion. *IEEE Trans. Automat. Control* 43, 3 (1998), 405–409. doi:10.1109/9.661607

[18] Muhammed A Ibrahim, Ausama Kh Mahmood, and Nashwan Saleh Sultan. 2019. Optimal PID controller of a brushless DC motor using genetic algorithm. *Int J Pow Elec & Dri Syst ISSN* 2088, 8694 (2019), 8694. doi:10.11591/ijpeds.v10.i2.pp822-830

[19] Arman Javadian, Nader Nariman-Zadeh, and Ali Jamali. 2023. Evolutionary design of marginally robust multivariable PID controller. *Engineering Applications of Artificial Intelligence* 121 (2023), 105938. doi:10.1016/j.engappai.2023.105938

[20] A Jayachitra and R Vinodha. 2014. Genetic algorithm based PID controller tuning approach for continuous stirred tank reactor. *Advances in Artificial Intelligence* 2014, 1 (2014), 791230. doi:10.1155/2014/791230

[21] Hozefa Jesawada, Amol Yerudkar, Carmen Del Vecchio, and Navdeep Singh. 2022. A Model-Based Reinforcement Learning Approach for Robust PID Tuning. In *2022 IEEE 61st Conference on Decision and Control (CDC)*. IEEE, Cancun, Mexico, 1466–1471. doi:10.1016/j.isatra.2020.12.033

[22] EA Joseph and OO Olaiya. 2017. Cohen-coon PID tuning method; A better option to Ziegler Nichols-PID tuning method. *ENginerring Research* 2, 11 (2017), 141–145.

[23] Abhishek Kumar Kashyap and Dayal R Parhi. 2021. Particle Swarm Optimization aided PID gait controller design for a humanoid robot. *ISA transactions* 114 (2021), 306–330.
[24] Hyungsub Kim, Muslum Ozgur Ozmen, Antonio Bianchi, Z Berkay Celik, and Dongyan Xu. 2021. PGFUZZ: Policy-Guided Fuzzing for Robotic Vehicles.. In *NDSS*. doi:10.14722/ndss.2021.24096
[25] Seulbae Kim and Taesoo Kim. 2022. Robofuzz: Fuzzing robotic systems over robot operating system (ros) for finding correctness bugs. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. Association for Computing Machinery, New York, NY, USA, 447–458. doi:10.1145/3540250.3549164
[26] Taegyu Kim, Aolin Ding, Sriharsha Etigowni, Pengfei Sun, Jizhou Chen, Luis Garcia, Saman Zonouz, Dongyan Xu, and Dave (Jing) 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* (Portland, Oregon) *(MobiSys '22)*. Association for Computing Machinery, New York, NY, USA, 69–83. doi:10.1145/3498361.3538938
[27] 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 *(SEC'19)*. USENIX Association, USA, 425–442.
[28] Ron Koymans. 1990. Specifying real-time properties with metric temporal logic. *Real-time systems* 2, 4 (1990), 255–299.
[29] Ivan Lopez-Sanchez and Javier Moreno-Valenzuela. 2023. PID control of quadrotor UAVs: A survey. *Annual Reviews in Control* 56 (2023), 100900. doi:10.1016/j.arcontrol.2023.100900
[30] Ignacio Martinez-Alpiste, Gelayol Golcarenarenji, Qi Wang, and Jose Maria Alcaraz-Calero. 2021. Search and rescue operation using UAVs: A case study. *Expert Systems with Applications* 178 (2021), 114937. doi:10.1016/j.eswa.2021.114937
[31] Graeme Massie. 2022. Amazon delivery drone sparked fire when it crashed into Oregon field, says FAA. https://www.independent.co.uk/news/world/americas/amazon-delivery-drone-field-fire-b2043644.html. Accessed: 2024-10-29.
[32] PM Meshram and Rohit G Kanojiya. 2012. Tuning of PID controller using Ziegler-Nichols method for speed control of DC motor. In *IEEE-international conference on advances in engineering, science and management (ICAESM-2012)*. IEEE, Nagapattinam, India, 117–122.
[33] Luca Mottola and Kamin Whitehouse. 2018. Fundamental concepts of reactive control for autonomous drones. *Commun. ACM* 61, 10 (2018), 96–104. doi:10.1145/3264417
[34] Christos Mourgelas, Evangelia Micha, Emmanouil Chatzistavrakis, and Ioannis Voyiatzis. 2023. Classification of Unmanned Aerial Vehicles in meteorology: A survey. *Environmental Sciences Proceedings* 26, 1 (2023), 135. doi:10.3390/environsciproc2023026135
[35] Katsuhiko Ogata. 2020. *Modern control engineering*. Prentice Hall, New Jersey, USA.
[36] 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*. Association for Computing Machinery, New York, NY, USA, 547–563. doi:10.1145/3607199.3607221
[37] Salil Purandare, Urjoshi Sinha, Md Nafee Al Islam, Jane Cleland-Huang, and Myra B Cohen. 2023. Self-Adaptive Mechanisms for Misconfigurations in Small Uncrewed Aerial Systems. In *2023 IEEE/ACM 18th Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*. IEEE, Melbourne, Australia, 169–180. doi:10.1109/SEAMS59076.2023.00030
[38] PX4 Autopilot. 2024. jMAVSim with SITL. https://docs.px4.io/main/en/sim_jmavsim/. Accessed: 2024-10-29.
[39] PX4 Autopilot. 2024. Open Source Autopilot for Drones - PX4 Autopilot. https://px4.io/. Accessed: 2024-10-29.
[40] sauravhobyy et al. 2023. 20kg payload quad copter Auto Tune Results- advise changes. https://discuss.ardupilot.org/t/20kg-payload-quad-copter-auto-tune-results-advise-changes/105268. Accessed: 2023-10-26.
[41] Nico Schiller, Merlin Chlosta, Moritz Schloegel, Nils Bars, Thorsten Eisenhofer, Tobias Scharnowski, Felix Domke, Lea Schönherr, and Thorsten Holz. 2023. Drone Security and the Mysterious Case of DJI's DroneID.. In *NDSS*. San Diego, CA, USA. doi:10.14722/ndss.2023.24217
[42] Thomas Schuhmann, Wilfried Hofmann, and Ralf Werner. 2011. Improving operational performance of active magnetic bearings using Kalman filter and state feedback control. *IEEE Transactions on Industrial Electronics* 59, 2 (2011), 821–829. doi:10.1109/TIE.2011.2161056
[43] Serhat Sonmez, Simone Martini, Mathew J Rutherford, and Kimon P Valavanis. 2024. Reinforcement Learning Based PID Parameter Tuning and Estimation for Multirotor UAVs. In *2024 International Conference on Unmanned Aircraft Systems (ICUAS)*. IEEE, Chania - Crete, Greece, 1224–1231. doi:10.1109/ICUAS60882.2024.10557032
[44] Corrado Steri et al. 2019. 24KG Quad light crash after manual PID tuning advice needed. https://discuss.ardupilot.org/t/24kg-quad-light-crash-after-manual-pid-tuning-advice-needed/39953. Accessed: 2023-10-26.
[45] ArduPilot Dev Team. 2024. AutoTune Mode — Copter Documentation. https://ardupilot.org/copter/docs/autotune.html. Accessed: 2024-10-29.

[46] ArduPilot Dev Team. 2024. Brake Mode — Copter Documentation. https://ardupilot.org/copter/docs/brake-mode.html. Accessed: 2024-10-29.
[47] ArduPilot Dev Team. 2024. Circle Mode — Copter Documentation. https://ardupilot.org/copter/docs/circle-mode.html. Accessed: 2024-10-29.
[48] ArduPilot Development Team. 2024. RTL Mode Documentation. https://ardupilot.org/copter/docs/rtl-mode.html. Accessed: 2024-05-16.
[49] ArduPilot Dev Team. 2024. ZigZag Mode — Copter Documentation. https://ardupilot.org/copter/docs/zigzag-mode.html. Accessed: 2024-10-29.
[50] PX4 Autopilot Dev Team. 2024. Hold Mode (Multicopter). https://docs.px4.io/main/en/flight_modes_mc/hold.html. Accessed: 2024-10-29.
[51] PX4 Autopilot Dev Team. 2024. Land Mode (Multicopter). https://docs.px4.io/main/en/flight_modes_mc/land.html. Accessed: 2024-10-29.
[52] PX4 Autopilot Dev Team. 2024. Orbit Mode (Multicopter). https://docs.px4.io/main/en/flight_modes_mc/orbit.html. Accessed: 2024-10-29.
[53] PX4 Autopilot Dev Team. 2024. Return Mode (Multicopter). https://docs.px4.io/main/en/flight_modes_mc/return.html. Accessed: 2024-05-16.
[54] 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*. Association for Computing Machinery, New York, NY, USA, 20–31. doi:10.1145/3468264.3468559
[55] Huifang Wang, Hongjun Cheng, and Heyuan Hao. 2020. The use of unmanned aerial vehicle in military operations. In *International Conference on Man-Machine-Environment System Engineering*. Springer, Singapore, 939–945. doi:10.1007/978-981-15-6978-4_108
[56] Stephen J Wright. 2015. Coordinate descent algorithms. *Mathematical programming* 151, 1 (2015), 3–34. doi:10.1007/s10107-015-0892-3
[57] Celaleddin Yeroglu, Cem Onat, and Nusret Tan. 2009. A new tuning method for PI λ D μ controller. In *2009 International Conference on Electrical and Electronics Engineering-ELECO 2009*. IEEE, Bursa, Turkey, II–312.
[58] Shaohua Zhang, Shuang Liu, Jun Sun, Yuqi Chen, Wenzhi Huang, Jinyi Liu, Jian Liu, and Jianye Hao. 2021. FIGCPS: Effective failure-inducing input generation for cyber-physical systems with deep reinforcement learning. In *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, Melbourne, Australia, 555–567. doi:10.1109/ASE51524.2021.9678832
[59] Cheng Zhao and Lei Guo. 2017. PID controller design for second order nonlinear uncertain systems. *Science China Information Sciences* 60 (2017), 1–13. doi:10.1007/s11432-016-0879-3

Received 2024-10-31; accepted 2025-03-31