An Exploratory Investigation of Log Anomalies in Unmanned Aerial Vehicles

Dinghua Wang*
dinghua.wang@student.uts.edu.au
University of Technology Sydney, Australia, Southern University of Science and Technology, China

Shuqing Li
sqli21@cse.cuhk.edu.hk
The Chinese University of Hong Kong, China

Guanping Xiao
gpxiao@nuaa.edu.cn
Nanjing University of Aeronautics and Astronautics, China

Yepang Liu†*
liuyp1@sustech.edu.cn
Research Institute of Trustworthy Autonomous Systems, Department of Computer Science and Engineering, Southern University of Science and Technology, China

Yulei Sui
y.sui@unsw.edu.au
The University of New South Wales, Australia

Pinjia He†
hepinjia@cuhk.edu.cn
School of Data Science, The Chinese University of Hong Kong, Shenzhen (CUHK-Shenzhen), China
Shenzhen Research Institute of Big Data, China

Michael R. Lyu
lyu@cse.cuhk.edu.hk
The Chinese University of Hong Kong, China

ABSTRACT

Unmanned aerial vehicles (UAVs) are becoming increasingly ubiquitous in our daily lives. However, like many other complex systems, UAVs are susceptible to software bugs that can lead to abnormal system behaviors and undesirable consequences. It is crucial to study such software bug-induced UAV anomalies, which are often manifested in flight logs, to help assure the quality and safety of UAV systems. However, there has been limited research on investigating the code-level patterns of software bug-induced UAV anomalies. This impedes the development of effective tools for diagnosing and localizing bugs within UAV system code.

To bridge the research gap and deepen our understanding of UAV anomalies, we carried out an empirical study on this subject. We first collected 178 real-world abnormal logs induced by software bugs in two popular open-source UAV platforms, i.e., PX4 and Ardupilot. We then examined each of these abnormal logs and compiled their common patterns. In particular, we investigated the most severe anomalies that led to UAV crashes, and identified their features. Based on our empirical findings, we further summarized the challenges of localizing bugs in system code by analyzing anomalous UAV flight data, which can offer insights for future research in this field.

CCS CONCEPTS

• **General and reference** → *Empirical studies*; • **Software and its engineering** → **Software defect analysis**.

KEYWORDS

UAV Anomaly, Software Bug, Crash, Code Pattern, Empirical Study

ACM Reference Format:
Dinghua Wang, Shuqing Li, Guanping Xiao, Yepang Liu, Yulei Sui, Pinjia He, and Michael R. Lyu. 2024. An Exploratory Investigation of Log Anomalies in Unmanned Aerial Vehicles. In *2024 IEEE/ACM 46th International Conference on Software Engineering (ICSE '24), April 14–20, 2024, Lisbon, Portugal.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3597503.3639186

1 INTRODUCTION

In recent years, UAVs have become more and more popular in our daily lives and played an increasingly important role. On the one hand, UAVs have a wide range of applications [24, 30], such as photography, parcel delivery, firefighting, and pesticide spraying. On the other hand, UAVs have more and more complex functions, such as automatic obstacle avoidance [16, 26], autonomous navigation [2, 34], and path planning [3, 87, 93]. However, with the diversification of applications and functions, the abnormal UAV behaviors caused by software bugs have also become more diverse and frequent, resulting in serious financial and even life losses. It is essential to study the characteristics of such abnormal UAV behaviors induced by software bugs to help people better judge UAV status and develop more reliable UAV systems.

*Dinghua Wang is a student in the joint Ph.D. program of UTS and SUSTech. This work was done when he was a visiting Ph.D. student at CUHK-Shenzhen.
†Yepang Liu and Pinjia He are the corresponding authors.

ICSE '24, April 14–20, 2024, Lisbon, Portugal

ACM ISBN 979-8-4007-0217-4/24/04…$15.00
https://doi.org/10.1145/3597503.3639186

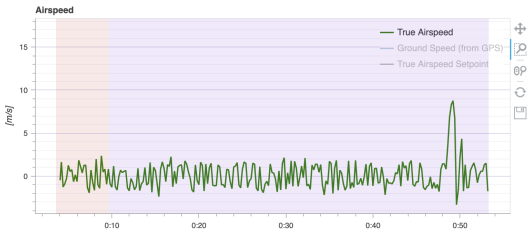

Figure 1: A UAV Log of Airspeed Changes

UAVs, as a typical type of cyber-physical systems (CPSs), employ a multitude of sensors to sense and respond to the dynamics of the physical environment, while also possessing the capability to establish network connections to interact with other UAVs and humans [11, 39, 41, 81]. Sensors in UAVs monitor the UAVs' states in real-time, such as speed, altitude, temperature, and battery power. Such information is often recorded in logs in popular UAV systems. In this paper, we focus on studying abnormal data in UAV logs that resulted from software bugs. We refer to such abnormal data in UAV logs as *UAV anomalies* for ease of presentation.

Figure 1 shows a typical UAV log that records airspeed changes during a flight. We can see that the UAV log is a collection of data points that vary over time, i.e., a time series. Existing research has pointed out that developers usually locate and fix software bugs in UAV systems by manually analyzing abnormal logs [41]. However, analyzing UAV anomalies is a challenging task that requires rich expertise in different domains, such as mechanics and control engineering. Moreover, UAV logs often contain noises and interference from the physical environment, which differ significantly from traditional software logs [90]. These difficulties impede the effective diagnosis of UAV anomalies. There is also some research that has explored the abnormal behaviors of UAVs, but much of the research has only classified UAV anomalies without investigating their patterns in depth. For example, when developing an anomaly detection tool, Li [41] analyzed anomalies in aviation logs but did not thoroughly investigate the causes of such anomalies at the system code level. Ultimately, experts still need to manually analyze anomalies case by case to determine unknown errors. Additionally, some studies [23, 86, 90] have focused on categorizing software bugs in UAV systems. However, the relationship between software bugs and UAV anomalies still remains largely unknown.

To fill the above-mentioned research gaps, we aim to analyze and understand the characteristics of UAV anomalies, disclosing their causes at the system code level, to provide developers with more guidance in combating software bugs via analyzing UAV logs. Particularly, we pay special attention to UAV crash anomalies, as they can result in significant damages. For example, in October 2021, a UAV show was disrupted by signal interference, causing many UAVs to fall from the sky and incurring serious property losses [98]. It is important and urgent to demystify such crash anomalies in UAV systems. In addition, before fixing software bugs that induce UAV anomalies, developers often need to locate the faulty source code in the UAV system. However, the difficulties of fault localization through UAV anomaly analysis are still unknown. Gaining insights into these difficulties is instrumental in designing effective UAV fault localization tools.

To achieve the above research goals, we conducted an empirical analysis of real-world UAV anomalies. Specifically, we aimed to summarize systematic knowledge about UAV anomalies by analyzing the available flight logs of two open-source UAV platforms: PX4 [54] and Ardupilot [8]. For the study, we collected 2,036 bugs in PX4 and Ardupilot, 178 of which were accompanied by abnormal flight logs. We manually analyzed each of these logs and relevant data, including source code, patches, bug reports, and so on. Our analysis identified six common types of UAV anomalies and their corresponding anomaly-inducing code patterns, as shown in Figure 2. We also summarized the features of UAV crash anomalies by analyzing the associated logs and observed four prominent challenges of fault localization in UAV systems via analyzing UAV anomalies.

Our work can help developers and researchers better understand UAV anomalies and the patterns of anomaly-inducing system code. It can also assist developers and researchers in performing effective fault localization by utilizing UAV anomalies, as well as in developing more stable and reliable UAV systems.

In summary, our work makes the following contributions:

- We conducted the first empirical study of software bug-induced UAV anomalies. Our investigation revealed six common types of UAV anomalies and their patterns at the system code level.
- We systematically discussed the difficulties of fault localization in UAV systems via anomalies in flight logs.
- We released a replication package and our dataset at https://doi.org/10.5281/zenodo.8208253 to facilitate future research.

The remaining part of this paper is organized as follows. In Section 2, we introduce the background of CPSs, UAVs, UAV logs, UAV log analysis, and pattern classification. The research questions driving our study are listed in Section 3. In Section 4, we present our analysis methodology. Sections 5, 6, and 7 delve into each of the three research questions and discuss our findings. We review the prior work related to our study in Section 8. In Section 9, we discuss the potential threats to the validity of our findings. Finally, Section 10 gives a conclusion of this work.

2 BACKGROUND

2.1 CPSs and UAV Systems

CPSs refer to systems formed by the tight integration of computer-controlled physical processes and networked communication systems. A common design objective of CPSs is to enhance system performance, efficiency, and security. These systems typically encompass sensor networks, real-time control systems, and automation systems [89, 91].

UAVs are an important type of CPSs. Within a typical UAV system, the embedded computer and controller manage the UAV's attitude and flight status, while sensors collect various flight data and environmental information to enable intelligent control and optimization of UAVs [5, 50]. In recent years, UAV technology has continued to advance rapidly, and several open-source UAV platforms have also emerged. For example, PX4 [27, 48, 54, 75] and Ardupilot [8, 14, 44, 49] are open-source UAV control software platforms, offering high performance, reliability, and flexibility. They support various vehicles, including multi-copters, fixed-wing aircraft, and vertical take-off and landing (VTOL) aircraft, with modular architectures.

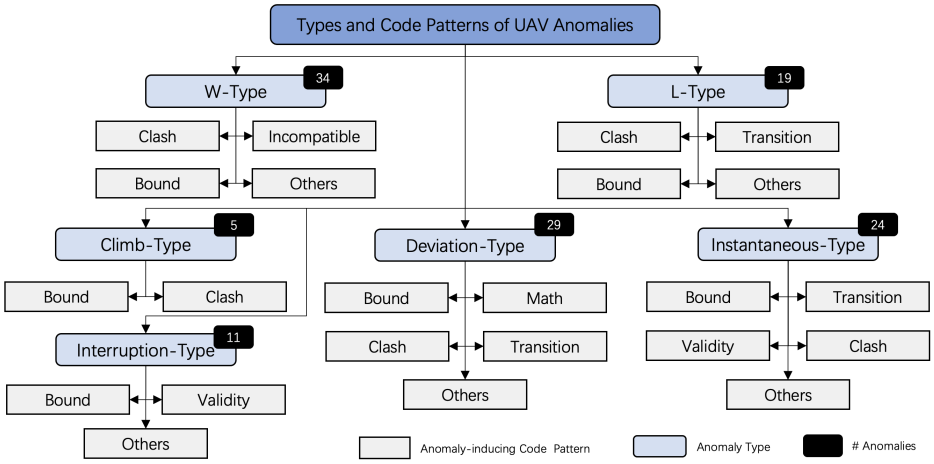

Figure 2: Types and Code Patterns of UAV Anomalies

2.2 UAV Logs

Logs play a critical role in recording system operations, aiding in troubleshooting, performance analysis, and optimization [95–97]. This is particularly important in the realm of UAVs, where analyzing flight data and system status is vital for safety and performance. Prominent UAV systems like PX4 and ArduPilot offer robust logging capabilities to capture sensor data, control outputs, and system messages [9, 57]. Typically, UAV log files encompass various data types, including time series data [79, 88] that provide continuous recordings over time, such as sensor measurements, flight status, and control outputs. Time series data is organized by timestamps, serving as a crucial component for researchers and developers to analyze UAV performance and diagnose UAV issues.

2.3 UAV Log Analysis and Pattern Classification

UAV log analysis is vital for understanding and optimizing UAV operations. Various general data analysis techniques can be employed for UAV log analysis, which mainly include three categories: statistical [7, 17, 28, 33, 37], classification-based [18, 85], and clustering-based [6, 15, 52] methods. Statistical methods involve modeling data distributions, often using the Gaussian model. Classification-based methods require labeled training data to learn and create boundaries between different classes. Clustering-based methods group data points into clusters, identifying anomalies as those not fitting any cluster.

Pattern classification techniques are crucial for identifying anomalies in UAV flight logs, which can also be divided into three main categories [1, 84]: feature-based (FB) [51], model-based (MB) [36], and distance-based (DB) methods [35]. In FB methods, time series data is transformed into feature vectors and classified using a conventional classifier such as a decision tree. MB methods assume shared underlying models within a class, assigning new series based on the best-fitting model. DB methods involve defining distance measures, which are then integrated into distance-based classification methods such as the k-nearest neighbor classifier.

3 RESEARCH QUESTIONS

Our empirical study investigates three research questions (RQs):

- **RQ1 (Anomaly Types and Anomaly-inducing Code Patterns):** Are there common types of UAV anomalies? What are the anomaly-inducing patterns at the system code level?
- **RQ2 (Characteristics of Crash Anomalies):** Which anomaly-inducing code pattern is most prone to cause UAV crashes? What are the underlying reasons behind the crash anomalies?
- **RQ3 (Difficulties of Fault Localization):** What are the difficulties in fault localization via analyzing UAV anomalies?

By investigating RQ1, we aim to classify UAV anomalies into different types and understand how UAV anomalies arise due to bugs in the system code. In RQ2, we dig deeper into the causes of UAV crashes based on the results of RQ1. In RQ3, we explore the obstacles of fault localization through anomaly analysis.

4 METHODOLOGY

4.1 Data Collection

To construct a dataset of UAV bugs and logs for analysis, we considered the following three requirements: (1) To ensure that our research results are useful, we should choose popular and active UAV systems with a sufficient number of anomalies as our study subjects. (2) To investigate the code-level patterns behind UAV anomalies, we should have access to the system source code. Therefore, the selected systems should be open-source. (3) For a comprehensive understanding of UAV anomalies (i.e., from the origin of the anomalies to their resolution), it is essential to know the details of the software bugs causing the anomalies, as well as their fixes. In accordance with requirement 1, we chose PX4 and ArduPilot from a wide range of open-source UAV systems as our study subjects. Other open-source UAV platforms, such as OpenPilot and Paparazzi, lack available issue data [90] for analysis. Based on requirements 2 and 3, we devised the following data collection criteria:

Table 1: The Statistics of the Two Subjects: PX4 and Ardupilot

Project Name	# Stars on GitHub	# Commits	Lines of Code	# Files	# Closed Issues	# Bugs
PX4	6,400	44,348	1,139,400	3,384	6,236	1,384
Ardupilot	8,400	75,857	2,123,463	3,291	5,085	652

- **Closed Issue Reports:** Resolved issues are often represented by closed reports, which are essential for our analysis. That is, we exclude open issues.
- **Inclusion of Log Data in Issue Reports:** In the open-source community, users or developers may not always adhere to issue reporting standards and may omit crucial log data when reporting abnormal UAV behaviors. Such reports hold little value for our study and are thus excluded.
- **Bug-tagged Issue Reports with a Corresponding Patch:** Open-source project maintainers typically assign appropriate labels to issue reports. In line with requirement 3, we collected issue reports tagged with a "bug" label and accompanied by a patch that resolves the issue.

Based on the above criteria, we manually collected 2,036 real bugs, of which 178 contain logs, from 11,321 closed issues in the projects PX4 and Ardupilot on GitHub. As shown in Table 1, PX4 contains 6,236 closed issues and 1,384 bugs, while Ardupilot has 5,085 closed issues and 652 bugs. It can also be seen from the table that these two projects contain more than two million lines of code and over 110,000 commits.

4.2 Types of UAV Anomalies

To gain a precise understanding of the collected anomalies, we begin by classifying them into different types. To facilitate manual analysis, we first employ the Flight Review tool [58] provided by PX4 and the UAV Log Viewer tool [10] from Ardupilot to visualize the UAV log files collected from the respective platforms. Since UAV logs consist of time series data, we also employ a widely adopted time series anomaly classification system [19, 22] to categorize the anomalies in UAV logs into three types, as shown in Figure 3:

Point Anomalies: Individual data points deviating significantly from the expected trend. Causes include random noise, data entry errors, or genuine unusual events.

Contextual Anomalies: Data points deviating from the expected pattern within a specific context, such as a particular day or season. These anomalies may appear normal in isolation but abnormal within context.

Collective Anomalies: A group of data points exhibiting unusual behavior collectively, although each point may not be anomalous individually. This type typically suggests underlying changes in the system generating the data, pattern shifts, or unknown variable relationships.

Following the aforementioned classification and the prior research [42] on anomaly detection in flight logs[1], we classify point and contextual anomalies as instantaneous-type anomalies in this work. Furthermore, we will provide a detailed classification scheme for collective anomalies in Section 4.3.

[1]Previous work [47] also explored anomaly classification in CPSs. However, as their studied systems differ greatly from UAVs, their method cannot be used in our work.

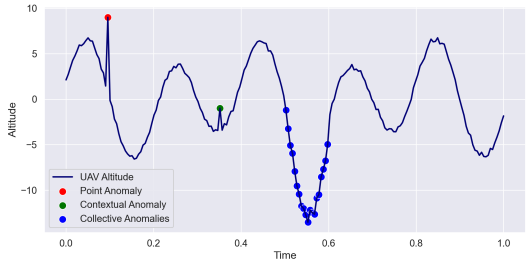

Figure 3: UAV Altitude with Three Types of Anomalies

4.3 Classifying Anomaly Types and Anomaly-inducing Code Patterns

To achieve an objective classification of UAV anomaly types and anomaly-inducing code patterns, we employed an open-coding procedure [40, 43, 90], a widely adopted data analysis method in empirical research. Specifically, two authors of this paper engaged in an eight-month iterative labeling process on the collected data in this study. Both authors have more than three years of research and development experience in the CPS domain. Particularly, prior to this work, they investigated hundreds of bugs in open-source CPS projects and deployed tens of buggy PX4 versions on either real UAVs or in emulators. During the manual analysis, through continuous discussion and refining of classification strategies, they ultimately reached a consensus on the classification results. In the following, we present the details of the manual analysis process.

In the *first iteration*, spanning a period of four months, the two authors separately analyzed all the UAV anomalies by checking logs, source code, patches, issue reports, and comments from the project developers, and provided descriptions and labels for the code-level patterns and types of UAV anomalies based on their own understanding. In this step, no restrictions were set on the content or format of the descriptions. For labels, we requested the two authors to describe them concisely, using as few words as possible to avoid excessive length.

Upon completing the above task, the two authors carried out their first discussion. They initially compared and discussed their results for each anomaly, preserving those with consistent or similar labels. For labels with discrepancies, they analyzed the reasons behind these differences, exploring factors such as whether the discrepancies stem from one party's misunderstanding, whether the discrepancies are due to different descriptive methods, or whether both parties misunderstand the anomalies. With these discussions, the two authors established a preliminary classification and labeling strategy.

In the *second iteration*, the two authors re-labeled all UAV anomalies based on the devised strategy and compared the outcomes. However, discrepancies persisted in the labeling results. This was because certain anomalies could be associated with multiple labels,

suggesting that the boundaries between different classes were unclear and the labeling strategy was ambiguous. To address these discrepancies, the two authors discussed again and clarified the boundaries of different classes, ensuring that each anomaly would be exclusively associated with a single label.

After the second refinement of the classification and labeling strategy, the two authors carried out a *third iteration* and ultimately achieved a consistent classification of anomaly types and anomaly-inducing code patterns.

With the three iterations, every UAV anomaly was assigned to one unique class within our taxonomy of anomaly types and anomaly-inducing code patterns. To facilitate the replication of our work, we discuss some special cases encountered during our classification process:

(1) Since UAV logs include more than 20 different indicators, such as speed, position, and temperature, the same anomaly may manifest in different indicators. For example, a crash may be identified as a sudden vertical drop in altitude, while also appearing as extremely irregular fluctuations in the UAV's posture. When developers do not specify the specific abnormal indicators in the issue reports, we would study the anomalies related to the physical trajectory of the UAV by referring to previous studies [32].

(2) Some logs contain a chain of abnormal events. For example, in PX4 Issue 12071 [59], a UAV experienced a loss of control that ultimately caused it to run out of power and crash. In such situations, we would study the initial anomaly that led to the loss of control rather than the crash anomaly that occurred later.

5 RQ1: ANOMALY TYPES AND ANOMALY-INDUCING CODE PATTERNS

5.1 Different Types of UAV Anomalies

After conducting iterative classification, in addition to the instantaneous type anomalies mentioned in Section 4.2, we additionally identified five other types of UAV anomalies based on their visual features: W-type, L-type, Climb-type, Deviation-type, and Interruption-type.

The **W-type** anomalies refer to those UAV anomalies exhibiting a waveform-like oscillation pattern with fluctuations. W-type anomalies rarely exhibit standard interval fluctuations. Instead, most of them have varying amplitudes and directions of oscillation, which can be broadly classified into horizontal, upward, and downward oscillations.

The **L-type** anomalies refer to those UAV anomalies that exhibit a sudden steep decline from normal states, followed by a gradual flattening with little or no significant fluctuation.

The **Climb-type** anomalies, in contrast to the L-type anomalies, are characterized by a rapid abnormal ascent from the normal state, followed by a gradual leveling off with little to no apparent fluctuation. The resulting shape resembles a step-like pattern as seen in the log data.

The **Deviation-type** anomalies refer to deviations between the UAV logs and the originally planned flight states. In contrast to the above types of anomalies, the Deviation type requires a comparison between two log entries to determine. For instance, in Figure 4, the UAV's flight appears normal by solely examining the yaw angle

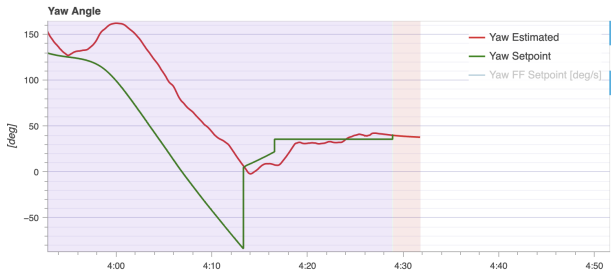

Figure 4: A Deviation-Type UAV Anomaly

estimated. However, by observing the yaw setpoint, we can identify a rightward deviation of the yaw angle in the UAV's flight path.

The **Interruption-type** anomalies in UAV logs refer to discontinuous or interrupted events. These anomalies are characterized by gaps or interruptions in the UAV's log data.

5.2 Code-Level Patterns of UAV Anomalies

Through our iterative classfication, we identified six major anomaly-inducing patterns at the code level. These patterns include bound, clash, incompatible, math, transition, and validity. In the following, we provide a detailed discussion of each pattern.

5.2.1 Bound. In UAV systems, there are various numerical boundaries related to physical parameters, time, UAV functionalities, and others. Failing to adhere to these boundaries (e.g., exceeding a limit) can result in anomalies or malfunctions of the UAV.

- **Bound of Parameters.** In UAV systems, there are thousands of parameters, most of which have physical boundaries. For example, the `HDRIFT` parameter represents the horizontal drift speed to use GPS, and its numerical bound is 0.1 to 1. If the program violates this boundary when using GPS, the UAV may malfunction. In addition, there are also some issues related to the use and optimization of parameter boundaries. For example, setting unnecessary boundaries or improper boundaries (e.g., too large or too small) can also cause UAV anomalies. In Issue 11420 [60], some users attempted to adjust the `MPC_XY_VEL_MAX` parameter to decrease the speed of the UAV. However, this approach led to an issue where the UAV's flight controller attempted to pull the aircraft back to the limited speed when the `MPC_VEL_MANUAL` or `MPC_XY_CRUISE` parameters exceeded the maximum speed. This resulted in the W-type anomaly depicted in Figure 5. Ultimately, the problem was resolved by verifying whether the parameters exceeded the maximum speed, as illustrated in Listing 1.
- **Bound of Filters.** Signal transmission within a UAV typically requires the use of filters to ensure signal quality by filtering out noises that are out of certain boundaries. High-quality signals, i.e., those with low noises, can ensure stable operation of the UAV. In practice, improper filtering may cause abnormal UAV behaviors. For example, in Issue 9150 [61], after achieving a stable takeoff, developers observed unstable oscillations in the UAV flight as shown in Figure 6. The occurrence of this anomaly was due to the disabling of the D-term filter[2] in the `mc_att_control` module

[2]The D-term filter is a component of the PID controller in UAVs and is responsible for fine-tuning the controller's output.

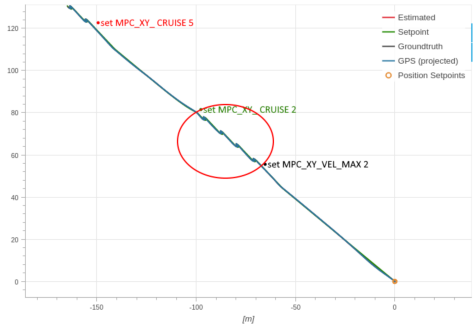

Figure 5: A W-Type UAV Anomaly of Issue #11420

```
    // Check that the design parameters are inside
    // the absolute maximum constraints
        if (_param_mpc_xy_cruise.get() >
            _param_mpc_xy_vel_max.get()) {
          _param_mpc_xy_cruise.set(_param_mpc_xy_vel_max
              .get());
          _param_mpc_xy_cruise.commit();
          mavlink_log_critical(&_mavlink_log_pub,
    "Cruise speed has been constrained by max speed")
        }
```

Listing 1: The Fix of Issue #11420

```
- PARAM_DEFINE_FLOAT(MC_DTERM_CUTOFF, 0.f);
+ PARAM_DEFINE_FLOAT(MC_DTERM_CUTOFF, 30.f);
```

Listing 2: The Fix of Issue #9150

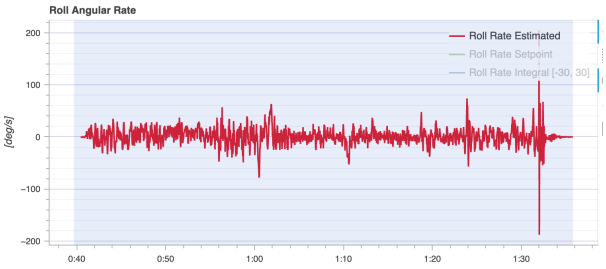

Figure 6: A W-Type UAV Anomaly of Issue #9150

of the system. This resulted in a high cutoff frequency, which negatively impacted the UAV's smooth operation. The solution to this issue is to enable the D-term filter, as shown in Listing 2.

- **Bound of Time.** Due to the limited processing power of UAVs and the need to handle a large number of tasks, sometimes small time boundaries can also cause UAV anomalies. For example, in Issue 15810 [62], the land detector module subscribed to a frequency of 1Hz, resulting in a timeout value that was too small. As a result, the UAV was unable to execute failsafe landing tasks in a timely manner before it shuts down in the air. Then, an L-type anomaly occurred, as shown in Figure 7.

There are other bound-related anomalies in our dataset, but due to the page limit, we present only the above three typical cases in this paper. All anomalies can be found in our released dataset.

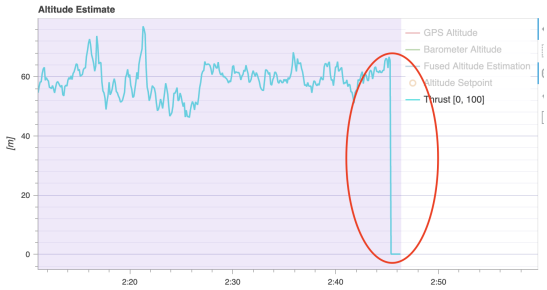

Figure 7: A L-Type UAV Anomaly of Issue #15810

```
+ // Accumulate autopilot gyro data across the same
      time interval as the flow sensor
+ _imu_del_ang_of += _imu_sample_delayed.delta_ang -
      _state.gyro_bias;
+ _delta_time_of += _imu_sample_delayed.delta_ang_dt;
```

Listing 3: The Fix of Issue #6772

5.2.2 Clash. The pattern *Clash* refers to the presence of two clashing objects, causing UAV anomalies. These objects include but are not limited to methods, libraries, commands, parameters, and reference coordinate systems.

- **Clash Between Commands and Flight Modes.** UAV systems are equipped with commonly used flight modes, such as takeoff, landing, and follow modes. PX4 includes 17 flight modes, each responding to different user commands. The diverse range of flight modes offered by UAV systems can cause clashes between user commands and flight modes, leading to unintended consequences. For example, in Issue 12029 [63], the use of the command `SET_ACTUATOR_CONTROL_TARGET` resulted in the disregard of subsequent commands, which ultimately led to a crash when the UAV attempted to fly in offboard mode using `body_rate` and thrust setpoints (the `SET_ACTUATOR_CONTROL_TARGET` command clashes with the offboard flight mode).
- **Clash of Timer Interval.** Some sensors have different timers, which can cause clashes in sensor data. For example, in Issue 6772 [64], the UAV's inertial measurement unit (IMU) data of gyro were not being accumulated over the same timer interval as the flow sensor data. This caused the optical flow configuration to exhibit slow drift over time. The problem was fixed by aligning the timers, as shown in Listing 3.
- **Clash Between Coordinates and Setpoints.** UAV systems involve various reference coordinate systems, including map coordinate systems, Earth coordinate systems, camera coordinate systems, global coordinate systems, and more. Different coordinate systems are associated with different setpoints. As a result, the reference coordinate system may clash with the setpoints. For instance, in Issue 12517 [65], the position control of a fixed-wing UAV uses a global coordinate system, but the position control in PX4 uses non-global position setpoints. This results in the fixed-wing UAV being unable to follow position setpoints for flight missions. As shown in Figure 8, the UAV flies directly in a straight line (i.e., the blue line) in offboard mode instead of following the setpoints.

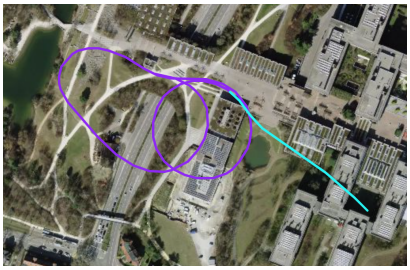

Figure 8: A Deviation-Type UAV Anomaly of Issue #12517

```
  pwm_out_sim mode_pwm
  sensors start
  commander start
- land_detector start multicopter
+ land_detector start vtol
  navigator start
```

Listing 4: The Fix of Issue #7737

5.2.3 Math. As a control system, UAVs involve a multitude of intricate mathematical calculations, encompassing tasks like angular velocity computation, position calculation, attitude determination, flight control algorithms, and more. Developers may make mistakes when implementing the calculations. Here is a typical example of this pattern: In Issue 18595 [66], the fixed-wing landing's loiter exit was incorrectly calculated during counterclockwise loiters, which resulted in the aircraft flying an approach vector with an offset. As a result, the UAV experienced a rightward deviation anomaly, as shown in Figure 4. The issue was eventually resolved by modifying the calculation formula.

5.2.4 Incompatible. There are many types of UAVs, such as fixed-wing and multi-copter, each corresponding to different flight control algorithms. If the flight control algorithm does not match the UAV type, it can lead to abnormal UAV behaviors. In Issue 7737 [67], the developer mistakenly used the `land_detector` for VTOL UAVs on a fixed-wing UAV, resulting in the UAV having very little throttle during takeoff, which eventually led to a failed takeoff. The developer fixed the bug by adjusting to the appropriate UAV type, as shown in Listing 4.

5.2.5 Transition. In UAVs, flight missions and modes can be switched using commands at runtime. However, due to the varying implementations of different missions or modes, many bugs can occur during the switching process. For example, in Issue 12910 [68], there is no control in the fixed wing after transitioning from altitude control. This results in the UAV's throttle being immediately cut to zero, as shown in Figure 9.

5.2.6 Validity. In UAV systems, data validity requires multiple verifications beyond simple numerical and data type checks. For instance, in Issue 15037 [69], the tailsitter's quaternion input must be completely normalized because the `acosf()` function used by the system module to calculate the control attitude requires input values to be within the range of -1 to 1. Invalid quaternion input resulted in the UAV failing to generate a pitch set value, which in turn caused the tailsitter VTOL front/back transmission to fail. The fix for this issue is shown in Listing 5.

Figure 9: A Instantaneous-Type UAV Anomaly of Issue #12910

```
+ // ensure input quaternions are exactly
      normalized because acosf(1.00001)==NaN
+ _q_trans_sp.normalize();
```

Listing 5: The Fix of Issue #15037

5.2.7 Others. There are other anomaly-inducing code patterns such as hardware support, and driver issues that we consider less important to be discussed in this paper since similar problems also exist in general software systems. Therefore, we will not elaborate on them due to the page limit.

5.3 Discussions

5.3.1 Comparison with General CPSs. Given that UAV systems are a typical type of CPSs, it is not surprising to find that many anomalies and code patterns observed in UAV systems also appear in other traditional CPSs. For instance, the *bound* pattern associated with UAV parameters can also be observed in robot systems [92, 94], which often enforce various parameter bounds. Furthermore, a noteworthy point is that some UAV systems such as PX4 directly leverage technologies from the Robot Operating System (ROS) [76] for obstacle avoidance and collision prevention, thereby naturally causing many similarities between these UAV systems and robot systems. Nonetheless, we found that UAV systems exhibit distinct features when compared to general CPS counterparts:

Severity of Crash Anomalies. UAVs may suffer from crash anomalies where altitude abruptly dropped to zero due to their unique operation in the airspace. We have observed twelve crash cases in our dataset where altitude unexpectedly drops to zero, causing severe consequences. In comparison, crashes in many other CPSs [47] do not manifest such pronounced changes in altitude.

Code Pattern Distributions. While anomaly-inducing code patterns can exhibit similarities between UAV systems and other CPSs, their distribution characteristics deviate due to the disparities in system functionality and hardware. According to a recent empirical study [94] on CPS bugs, bugs tied to numerical ranges accounted for 2.75%, whereas in our dataset, bound-related bugs comprised a substantial 34.8%.

Impact of the Control Algorithm. Flight control algorithms are a distinctive feature of UAV systems. UAVs demand exceptionally precise and real-time flight control algorithms [77] and bugs within these algorithms can lead to critical flight issues. In contrast to conventional CPSs, the system code governing UAV flight control involves a complex interplay of mathematical and physical calculations [55]. As exemplified in Section 5.2.1, even a minute deviation can culminate in a catastrophic UAV crash.

5.3.2 Detecting UAV Anomalies and Software Bugs. In RQ1, we reveal the causes of UAV anomalies at the system code level. To inspire future research, we further discuss several possible ideas to detect the three most common types of software bugs that induce abnormal UAV behaviors.

The Bound Pattern: Our studied UAV systems involve more than one thousand parameters, whose values should be within specific boundaries [90]. The number of parameters may further expand with software or hardware updates. Essentially, the state of a UAV flight is determined by a set of diverse parameter values. Therefore, it is possible to employ fuzzing techniques to continuously fly a UAV to detect whether each set of parameter values is within proper boundaries. However, the challenge arises from the multitude of parameter combinations, leading to a potential search space explosion. One viable solution to this problem is to utilize evolutionary algorithms to select the parameters to fuzz in a guided manner. Future research can investigate the effectiveness of this approach or leverage other search and optimization algorithms to reduce the search space when detecting bugs of the bound pattern.

The Clash Pattern: According to our observation of the clash of timer intervals in RQ1, it is evident that two distinct sensors may possess different timer intervals. When values from sensors with disparate timer intervals intersect within the program, conflicts may arise. To address and detect such issues, we recommend using symbolic execution. This method allows for the identification of potential intersection points among variables storing values from different sensors, thereby facilitating the detection of clashes.

The Transition Pattern: Building upon our observations of real anomalies, it is possible to craft specialized testing strategies to detect bugs of the transition pattern. For instance, one can continuously generate commands for UAVs to switch flight modes (i.e., *transition-specific fuzzing*) to detect potential issues arising during mode transitions within the system.

6 RQ2: CHARACTERISTICS OF CRASH ANOMALIES

Given the severe consequences of UAV crashes, it is important to understand the characteristics of the crash anomalies in real UAV systems to guide future research. For the first sub-question of RQ2, we focus on discerning the most common anomaly-inducing code pattern that can result in UAV crashes. The second sub-question investigate how such bugs in system code cause UAV crashes.

For the first sub-question, we observed that ***the bound pattern is most likely to cause a UAV to lose control and crash.*** We did statistical analysis on the code patterns causing UAV crash anomalies based on the results of RQ1. As shown in Figure 10, the bound pattern clearly caused the highest number of crash anomalies. This suggests that future research effort may focus on developing effective techniques to identify improper or missing boundary checks in UAV system code to avoid the catastrophe of UAV crashes.

To answer the second sub-question, we examined the fixing code for each bound-related issue in our dataset. Specifically, by manual code inspection, we analyzed which system module was affected by each bound-related issue and why such issues ultimately led to UAV crashes. We made two major observations through the analysis:

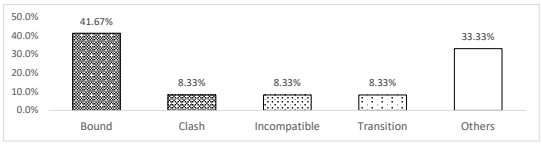

Figure 10: Code Pattern Distribution of Crash Anomalies

1) Most issues with bound patterns are related to control algorithms. In our dataset, 80% bound-related issues are in control-related modules, such as position control, altitude control, attitude control[3], navigation, and so on. This is understandable since the flight control algorithm has a direct impact on the stability of the UAV [53]. We observe that control algorithms can act like an amplifier that magnifies and even superimposes the effects of bound-related problems, ultimately leading to the loss of control. For example, in Issue 7202 [70], the problem arose from temperature fluctuations, which affected the accuracy of the Inertial Measurement Unit (IMU), leading to a minor deviation in `IMU.AccZ` (i.e., the Z-axis acceleration). Initially, this deviation did not impact the UAV's flight performance. However, as the flight mission continued, this deviation accumulated and there was no boundary check for the value. Consequently, the accumulated deviation grew beyond the Extended Kalman Filter's (EKF) capacity, resulting in a catastrophic UAV crash. This entire process is a kind of "butterfly effect" within the UAV, with the flight control algorithm acting as the accumulator.

2) The boundary checking in our studied UAV systems is insufficiently effective. Our manual code analysis unveils that all bound-related issues resulting in UAV crashes were due to the lack of proper boundary checks. In a UAV system, important value boundaries should be correctly enforced to ensure that the UAV's operation complies with safety requirements or restrictions. However, when building real UAV systems, it is difficult to implement sound error protection measures (i.e., certain values may not be checked properly at runtime). Once some values exceed their expected boundaries, the system may not be able to take corrective measures to prevent the UAV from exhibiting undesired behaviors. This could make the UAV lose control and ultimately crash. For example, in Issue 10757 [71], the UAV's setpoint was only slightly beyond a reasonable radius, but the system could not prevent the UAV from crashing by correcting the setpoint.

Although boundary checking is critical to ensuring the safety of UAVs, the frequent occurrences of crash anomalies caused by issues with the bound pattern highlights the inadequacy of boundary checking in real UAV systems. Ideally, whenever a new input is received or a new variable is introduced, the system should verify whether the value of the variable falls within a reasonable boundary. Otherwise, issues may arise. Unfortunately, when implementing UAV systems, developers need to consider a substantial amount of boundaries (e.g., there are more than one thousand boundaries related to parameters). It is highly challenging for them to avoid bound-related issues effectively.

In summary, we observed that bound-related issues are the primary reason for UAV crashes and boundary checking in real UAV

[3]The attitude control of a UAV refers to the precise control of its flight orientation, including pitch, roll, and yaw.

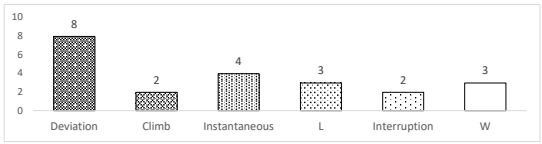

Figure 11: The Number of Distinct Anomaly-Inducing Code Patterns for Each Anomaly Type

systems is far from comprehensive. To prevent disastrous consequences, UAV systems should have more effective error protection measures such as self-monitoring, real-time fault diagnosis, and self-healing capabilities. This will help improve the safety and reliability of UAVs, ensuring their stable operation in various environments and tasks. Nonetheless, the computational resources of UAVs are often constrained, and incorporating additional error-protection features may inevitably affect UAV performance. Exploring efficient approaches for handling crash anomalies while mitigating the performance impact becomes a compelling avenue for future research in this domain.

7 RQ3: DIFFICULTIES OF FAULT LOCALIZATION

To address RQ3, we analyzed the UAV anomalies in our dataset to explore two questions: (1) For each type of UAV anomalies, do the anomaly-inducing code segments exhibit common patterns in terms of their location (e.g., whether the code segments are often located within a common system module)? (2) Are there obvious correlations between the types of UAV anomalies and the anomaly-inducing code patterns? If any of the two questions has a positive answer, it would mean that developers can have useful clues when performing fault localization via examining anomalies in flight logs. Unfortunately, by analyzing the collected anomalies and their patches, we found that the answer to both questions is "no", suggesting that locating buggy code in UAV systems is challenging even when anomalous flight logs are available. Developers may need more information besides the easily obtainable logs to perform effective fault localization. In the following, we discuss the observations we made during the analysis process.

Observation 1: Locating software bugs causing deviation-type anomalies is particularly challenging. Figure 11 gives the number of distinct anomaly-inducing code patterns for each anomaly type. As we can see, deviation-type anomalies exhibit the most diverse anomaly-inducing code patterns. Among 20 deviation-type anomalies, we identified a total of eight distinct anomaly-inducing code patterns (four are included in the others category in Figure 2). Besides, deviation-type anomalies are distributed across 12 different system modules, making fault localization very challenging.

Since it is difficult to locate bugs causing deviation-type anomalies, it would be very interesting to investigate how developers actually deal with deviation-type anomalies in their debugging process. Based on our investigation of GitHub issue reports and fixing commits, we observed two strategies adopted by developers to debug deviation-type anomalies:

Strategy 1: Examining both deviation-type anomalies and other log items, not just the anomalous items. The logs of a UAV contain over twenty items, which can be used to determine the UAV's status. We observed that in addition to anomalous log items, some developers also checked normal log items to obtain useful information for debugging deviation-type anomalies, as shown in Issue #14735 [72]. This is because deviation-type anomalies usually mean that the UAV is yawing but still operational, so many log items are in normal states. By analyzing normal log items one by one, developers can effectively rule out certain possibilities when pinpointing faulty modules.

Strategy 2: Adjusting parameters to observe the performance of the UAV. Some developers chose to observe the status of the UAV by constantly adjusting the relevant parameters to identify a minimal set of parameters and related data flows that truly affected the performance of the UAV. For example, in Issue 5110 [73], the developers mentioned such an attempt. While the developers did not immediately succeed in resolving the issue, their description highlighted that parameter adjustment was employed as a troubleshooting strategy.

While developers can address deviation-type anomalies through various strategies, we found that fixing bugs that cause deviation-type anomalies in PX4 was time-consuming, taking an average of 49.5 days. In comparison, on average, it took developers around 34 days to fix other bug-tagged issues in PX4 [25]. This also indicates the challenges of dealing with deviation-type anomalies.

Observation 2: There is a many-to-many relationship between UAV anomaly types and anomaly-inducing code patterns. Figure 2 shows that in our dataset, each anomaly type is associated with multiple anomaly-inducing code patterns, and conversely, each code pattern is found across multiple anomaly types. Such a many-to-many relationship makes fault localization difficult: (1) When multiple anomaly types are related to the same anomaly-inducing code pattern, identifying the specific anomaly type causing an error can be challenging; (2) On the other hand, when the same anomaly type is associated with multiple anomaly-inducing code patterns, inferring the specific code pattern responsible for an error can be challenging. In both of the two scenarios, further analysis is necessary to determine the true anomaly type and identify the buggy code segments in the UAV system. Contextual information and domain knowledge are essential to locate the faults accurately in such cases. Since the many-to-many relationship between anomaly types and anomaly-inducing code patterns can be highly complex in real UAV systems with large codebases, automated tools and techniques are desirable to help locate faults effectively. Future research may explore the possibility of leveraging deep learning algorithms to learn the complex relationship between UAV anomalies and their triggering bugs.

Observation 3: UAV behaviors are often affected by environmental factors. Such uncertainties make fault localization challenging. In our datasets, we observed that UAV flight logs are frequently influenced by environmental factors, which can pose challenges for analysis. Specifically, 34.27% of the analyzed logs were subject to the influence of environmental factors.

The physical environment can affect the analysis of the UAV flight logs in several ways. First, the quality of the UAV logs may be affected in certain environments. For example, high- or low-temperature environments can cause a decrease in the performance of UAV hardware and sensors, thereby affecting log quality, especially for some temperature-sensitive sensors, as we mentioned

in Section 6-A. Secondly, some anomalies can only occur under specific environmental conditions. For instance, in Issue 13956 [74], developers remarked: *"This is an issue that we have seen constantly, with the PixRacer and K66, FMUv5. We are seeing this issue when the wind is 6 m/s and over."* This anomaly only becomes evident when the wind speed is 6 m/s and over; however, it vanishes from the log when the wind speed is low. This leads to a situation where developers are unable to observe and analyze user-reported UAV anomalies, hindering fault localization. Furthermore, some physical environments can also result in UAV communication interruptions, leading to abnormal termination of log data. This also causes obstacles to fault localization.

Observation 4: It is often difficult to distinguish between hardware error-caused anomalies and software bug-caused anomalies in UAVs. Our study focuses on anomalies caused by software bugs. However, quite a few (around 11.8%) real-life issues mentioned hardware in their issue reports. In UAV systems, there are close interconnections between hardware and software components. This tight coupling often results in similar anomalies with different causes (e.g., unstable flight and abnormal landings can be caused by both hardware and software problems), making it challenging for developers to differentiate between hardware and software issues.

The interaction between multiple components further complicates fault localization. For instance, sensor readings that affect the behavior of the control system may result in problems that appear to be software errors. Here is a representative developer comment from a real issue [74] in our dataset: *"And actually the battery is still enough, I only set the minimum voltage per cell a bit higher. When I started Auto Mode, it flew a couple meter then changed to Land Mode due to the UAV detect low battery."* The difficulty of fault localization stems from the complexity of distinguishing whether the problem is caused by battery damage, or a software bug through log analysis.

When hardware is damaged, UAV systems typically record error logs. However, these abnormal logs may exhibit similarities with anomalies caused by software bugs. In UAV systems, distinguishing between anomalies caused by hardware errors and those caused by software errors requires developers to consider various factors, such as error manifestations, component interactions, error logs, and environmental conditions. Future research can further look investigate this problem to design effective solutions. It might be possible to leverage learning algorithms to identify the boundaries between hardware and software-related issues.

8 RELATED WORK

Anomaly Detection. Several surveys have been written on the topic of anomaly detection [56]. For example, Chandola et al. [19] conducted a comprehensive survey of anomaly detection tools, categorizing existing techniques based on their underlying approach and identifying key assumptions that can be used to differentiate between normal and abnormal behaviors. They also discussed the computational complexity of the techniques. Hodge and Austin [33] presented a survey of anomaly detection techniques, highlighting their motivations and distinguishing their advantages and disadvantages. Agyemang et al. [4] discussed the applications of anomaly detection and provided a taxonomy for categorizing anomaly detection techniques. Besides the general surveys, recent studies have also advanced the field of anomaly detection in UAVs. Markou and Singh [46] presented a review of statistical approaches to anomaly detection and discussed the novelty of NN-based anomaly detection. They also highlighted the importance of anomaly detection in computer vision, pattern recognition, and robotics, which is highly related to UAV system reliability. A substantial amount of research on outlier detection has been done in the area of statistics and has been reviewed in several books [31, 78] and articles [12, 13].

As for anomaly detection approaches, Shar et al. [80] developed DronLomaly, a deep learning method with LSTM models for real-time anomaly detection in drones, leveraging flight log analysis. Silalahi et al. [82] introduced a sentiment analysis method using fine-tuned large language models for anomaly detection in drone flight logs. Studiawan et al. [83] proposed the use of Sigma rules in drone forensic timelines, aiding in the identification of anomalous drone activities. Ma et al. [45] introduced graphical normalizing flows, a novel deep learning model for efficient anomaly detection in small unmanned aerial systems. Moreover, Cleland-Huang et al. [21] focused on enhancing the safety of small UAVs in airspace with a multi-pronged approach that combines data analytics and deep learning techniques for anomaly analysis in both real-time and post-mortem scenarios.

Evaluation of Anomaly Detectors. There are also many studies focusing on evaluating anomaly detection tools. Lavin and Ahmad [38] evaluated real-time anomaly detection algorithms using the Numenta Anomaly Benchmark (NAB), which provides a controlled and repeatable environment for evaluating anomaly detectors. Markus and Seiichi [29] evaluated 19 different unsupervised anomaly detection algorithms on 10 different datasets from multiple application domains, highlighting the strengths and weaknesses of different approaches. Varun and Vipin [20] evaluated seven anomaly detection techniques on ten public datasets collected from three diverse application domains, presenting a novel way to generate sequence data with desired characteristics for further understanding of the performance of anomaly detectors.

Empirical Studies of UAV Bugs. Taylor et al. [86] studied the reported bugs in Ardupilot and PX4, two widely used open-source control firmware for UAV systems, and characterized their root causes, severity, and position in the firmware architecture. Despite the adoption of rigorous software engineering practices, bugs were common in the two systems and often had severe symptoms. Wang et al. [90] performed the first large-scale empirical study of UAV-specific bugs, also focusing on PX4 and Ardupilot. By thoroughly analyzing 569 bugs, they successfully identified eight distinct types of UAV-specific bugs and provided insights into their root causes.

Differing from the above studies, our work aims to provide guidance for locating software bugs in UAV systems by analyzing anomalies manifested in UAV logs. Although there is a plethora of research on anomaly detection for UAV systems, further analysis and identification of bugs in the system code still requires significant manual effort. To fill this gap, our work offers a more comprehensive characterization of UAV anomalies and reveals the relationship between anomalies and the underlying software bugs. We also point out the difficulties of fault localization through analyzing UAV logs to shed light on future research.

9 THREATS TO VALIDITY

9.1 External Threats

Our empirical study only included two open-source UAV projects, and the UAV anomalies we collected may not be representative or comprehensive. Therefore, our findings may not be applicable to other UAV systems. We made an effort to find more projects, but many are either closed-source or lack sufficient data for our analysis. Future studies may further investigate more projects to enhance the community's understanding of UAV anomalies and software bugs.

9.2 Internal Threats

The internal threats mainly come from bug selection and manual analysis. We only collected issues labeled as "bugs" from PX4 and Ardupilot for our research. Although the project developers and maintainers label bugs properly, there is still a chance that we missed some real bugs that have caused severe UAV anomalies but were not labeled. One possible way to mitigate this threat is to investigate GitHub issues with other labels or no labels. As this process is labor-intensive, we leave it as our future work.

Besides, similar to many other empirical studies [86, 90], manual analysis has often been a potential threat to the validity of research results, as subjective judgment can come into play. To reduce this threat, we adopted the widely used data analysis method in empirical research known as "open coding" [40, 43, 90]. To enhance objectivity in the classification process, multiple iterations of analysis were conducted by two authors. They held many discussions when analyzing and categorizing anomaly patterns. They also looked for references from the comments in issue reports when constructing the anomaly type and anomaly-inducing code pattern taxonomy. The results have also been cross-validated by two other authors and released for public scrutiny.

10 CONCLUSION AND FUTURE WORK

In summary, in this work, we delved into the realm of UAVs to examine the impact of software bugs on UAV anomalies. We collected and analyzed 178 real-world abnormal logs stemming from software bugs in two widely used open-source UAV platforms, PX4 and Ardupilot. Our research primarily focused on the identification of code-level patterns associated with these anomalies. We paid special attention to crash anomalies to understand their causes and also investigated the challenges of localizing anomaly-inducing code in UAV systems. The observations and insights gained from our study can shed light on research on diagnosing and localizing bugs within UAV system code. Based on our empirical findings, in the future, we plan to design useful techniques to help developers enhance the security and reliability of UAV systems. We will also collect more issues from real-world projects to make our empirical study more comprehensive.

ACKNOWLEDGMENTS

This work is supported by the National Natural Science Foundation of China (Grant Nos. 61932021, 62002163, and 62102340), Natural Science Foundation of Jiangsu Province (Grant No. BK20200441), Science and Technology Innovation Committee Foundation of Shenzhen (Grant No. ZDSYS20210623092007023), and the Research Grants Council of the Hong Kong Special Administrative Region, China (No. CUHK 14210920 of the General Research Fund).

REFERENCES

[1] Amaia Abanda, Usue Mori, and Jose A Lozano. 2019. A review on distance based time series classification. *Data Mining and Knowledge Discovery* 33, 2 (2019), 378–412.
[2] Wilbert G Aguilar, Vinicio S Salcedo, David S Sandoval, and Bryan Cobeña. 2017. Developing of a video-based model for UAV autonomous navigation. In *Computational Neuroscience: First Latin American Workshop, LAWCN 2017, Porto Alegre, Brazil, November 22–24, 2017, Proceedings*. Springer, 94–105.
[3] Wilbert G Aguilar, Vinicio S Salcedo, David S Sandoval, and Bryan Cobeña. 2017. Developing of a video-based model for UAV autonomous navigation. In *Computational Neuroscience: First Latin American Workshop, LAWCN 2017, Porto Alegre, Brazil, November 22–24, 2017, Proceedings*. Springer, 94–105.
[4] Malik Agyemang, Ken Barker, and Rada Alhajj. 2006. A comprehensive survey of numeric and symbolic outlier mining techniques. *Intelligent Data Analysis* 10, 6 (2006), 521–538.
[5] Foisal Ahmed and Maksim Jenihhin. 2022. A Survey on UAV Computing Platforms: A Hardware Reliability Perspective. *Sensors* 22, 16 (2022), 6286.
[6] James Allan, Jaime G Carbonell, George Doddington, Jonathan Yamron, and Yiming Yang. 1998. Topic detection and tracking pilot study final report. (1998).
[7] Frank J Anscombe. 1960. Rejection of outliers. *Technometrics* 2, 2 (1960), 123–146.
[8] Ardupilot. 2023. *Ardupilot Doc.* Retrieved March 25, 2023 from https://ardupilot.org/ardupilot/
[9] Ardupilot. 2023. *Ardupilot Log.* Retrieved March 25, 2023 from https://ardupilot.org/copter/docs/common-logs.html
[10] Ardupilot. 2023. *UAV Log Viewer.* Retrieved March 25, 2023 from https://plot.ardupilot.org/#/
[11] Radhakisan Baheti and Helen Gill. 2011. Cyber-physical systems. *The impact of control technology* 12, 1 (2011), 161–166.
[12] Zuriana Abu Bakar, Rosmayati Mohemad, Akbar Ahmad, and Mustafa Mat Deris. 2006. A comparative study for outlier detection techniques in data mining. In *2006 IEEE conference on cybernetics and intelligent systems*. IEEE, 1–6.
[13] Richard J Beckman and R Dennis Cook. 1983. Outlier..... s. *Technometrics* 25, 2 (1983), 119–149.
[14] He Bin and Amahah Justice. 2009. The design of an unmanned aerial vehicle based on the ArduPilot. *Indian Journal of Science and Technology* 2, 4 (2009), 12–15.
[15] Richard J Bolton, David J Hand, et al. 2001. Unsupervised profiling methods for fraud detection. *Credit scoring and credit control VII* (2001), 235–255.
[16] Almira Budiyanto, Adha Cahyadi, Teguh Bharata Adji, and Oyas Wahyunggoro. 2015. UAV obstacle avoidance using potential field under dynamic environment. In *2015 International Conference on Control, Electronics, Renewable Energy and Communications (ICCEREC)*. IEEE, 187–192.
[17] Simon Byers and Adrian E Raftery. 1998. Nearest-neighbor clutter removal for estimating features in spatial point processes. *J. Amer. Statist. Assoc.* 93, 442 (1998), 577–584.
[18] Joao BD Cabrera, Lundy Lewis, and Raman K Mehra. 2001. Detection and classification of intrusions and faults using sequences of system calls. *Acm sigmod record* 30, 4 (2001), 25–34.
[19] Varun Chandola, Arindam Banerjee, and Vipin Kumar. 2009. Anomaly detection: A survey. *ACM computing surveys (CSUR)* 41, 3 (2009), 1–58.
[20] Varun Chandola, Varun Mithal, and Vipin Kumar. 2008. Comparative evaluation of anomaly detection techniques for sequence data. In *2008 Eighth IEEE international conference on data mining*. IEEE, 743–748.
[21] Jane Cleland-Huang, Nitesh Chawla, Myra Cohen, Md Nafee Al Islam, Urjoshi Sinha, Lilly Spirkovska, Yihong Ma, Sulil Purandare, and Muhammed Tawfiq Chowdhury. 2022. Towards Real-Time Safety Analysis of Small Unmanned Aerial Systems in the National Airspace. In *AIAA AVIATION 2022 Forum*. 3540.
[22] Andrew A Cook, Göksel Mısırlı, and Zhong Fan. 2019. Anomaly detection for IoT time-series data: A survey. *IEEE Internet of Things Journal* 7, 7 (2019), 6481–6494.
[23] 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.
[24] Paul G Fahlstrom, Thomas J Gleason, and Mohammad H Sadraey. 2022. *Introduction to UAV systems.* John Wiley & Sons.
[25] Maria Lucia Ferramosca. 2021. *Safety Assessment of UAV Systems: Field Data Analysis.* Ph. D. Dissertation. Politecnico di Torino.
[26] Allen Ferrick, Jesse Fish, Edward Venator, and Gregory S Lee. 2012. UAV obstacle avoidance using image processing techniques. In *2012 IEEE International Conference on Technologies for Practical Robot Applications (TePRA)*. IEEE, 73–78.

[27] Jesús García and Jose M Molina. 2022. Simulation in real conditions of navigation and obstacle avoidance with PX4/Gazebo platform. *Personal and Ubiquitous Computing* 26, 4 (2022), 1171–1191.

[28] Robert D Gibbons, Dulal K Bhaumik, and Subhash Aryal. 2009. *Statistical methods for groundwater monitoring*. John Wiley & Sons.

[29] Markus Goldstein and Seiichi Uchida. 2016. A comparative evaluation of unsupervised anomaly detection algorithms for multivariate data. *PloS one* 11, 4 (2016), e0152173.

[30] Lav Gupta, Raj Jain, and Gabor Vaszkun. 2015. Survey of important issues in UAV communication networks. *IEEE communications surveys & tutorials* 18, 2 (2015), 1123–1152.

[31] Douglas M Hawkins. 1980. *Identification of outliers*. Vol. 11. Springer.

[32] Zhijian He, Yao Chen, Enyan Huang, Qixin Wang, Yu Pei, and Haidong Yuan. 2019. A system identification based oracle for control-cps software fault localization. In *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*. IEEE, 116–127.

[33] Victoria Hodge and Jim Austin. 2004. A survey of outlier detection methodologies. *Artificial intelligence review* 22, 2 (2004), 85–126.

[34] Nursultan Imanberdiyev, Changhong Fu, Erdal Kayacan, and I-Ming Chen. 2016. Autonomous navigation of UAV by using real-time model-based reinforcement learning. In *2016 14th international conference on control, automation, robotics and vision (ICARCV)*. IEEE, 1–6.

[35] Hüseyin Kaya and Şule Gündüz-Öğüdücü. 2015. A distance based time series classification framework. *Information Systems* 51 (2015), 27–42.

[36] Alexios Kotsifakos and Panagiotis Papapetrou. 2014. Model-based time series classification. In *Advances in Intelligent Data Analysis XIII: 13th International Symposium, IDA 2014, Leuven, Belgium, October 30–November 1, 2014. Proceedings 13*. Springer, 179–191.

[37] Jorma Laurikkala, Martti Juhola, Erna Kentala, N Lavrac, S Miksch, and B Kavsek. 2000. Informal identification of outliers in medical data. In *Fifth international workshop on intelligent data analysis in medicine and pharmacology*, Vol. 1. 20–24.

[38] Alexander Lavin and Subutai Ahmad. 2015. Evaluating real-time anomaly detection algorithms–the Numenta anomaly benchmark. In *2015 IEEE 14th international conference on machine learning and applications (ICMLA)*. IEEE, 38–44.

[39] Edward A Lee. 2008. Cyber physical systems: Design challenges. In *2008 11th IEEE international symposium on object and component-oriented real-time distributed computing (ISORC)*. IEEE, 363–369.

[40] Sarah Lewis. 2015. Qualitative inquiry and research design: Choosing among five approaches. *Health promotion practice* 16, 4 (2015), 473–475.

[41] Lishuai Li. 2013. *Anomaly detection in airline routine operations using flight data recorder data*. Ph. D. Dissertation. Massachusetts Institute of Technology.

[42] Lishuai Li. 2013. *Anomaly detection in airline routine operations using flight data recorder data*. Ph. D. Dissertation. Massachusetts Institute of Technology.

[43] Shuqing Li, Yechang Wu, Yi Liu, Dinghua Wang, Ming Wen, Yida Tao, Yulei Sui, and Yepang Liu. 2020. An exploratory study of bugs in extended reality applications on the web. In *2020 IEEE 31st International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 172–183.

[44] Zongtong Luo, Xianbo Xiang, and Qin Zhang. 2019. Autopilot system of remotely operated vehicle based on Ardupilot. In *Intelligent Robotics and Applications: 12th International Conference, ICIRA 2019, Shenyang, China, August 8–11, 2019, Proceedings, Part III 12*. Springer, 206–217.

[45] Yihong Ma, Md Nafee Al Islam, Jane Cleland-Huang, and Nitesh V. Chawla. 2023. Detecting Anomalies in Small Unmanned Aerial Systems via Graphical Normalizing Flows. *IEEE Intell. Syst.* 38, 2 (2023), 46–54. https://doi.org/10.1109/MIS.2023.3252810

[46] Markos Markou and Sameer Singh. 2003. Novelty detection: a review—part 2:: neural network based approaches. *Signal processing* 83, 12 (2003), 2499–2521.

[47] Reza Matinnejad, Shiva Nejati, Lionel C Briand, and Thomas Bruckmann. 2016. Automated test suite generation for time-continuous simulink models. In *proceedings of the 38th International Conference on Software Engineering*. 595–606.

[48] Lorenz Meier, Dominik Honegger, and Marc Pollefeys. 2015. PX4: A node-based multithreaded open source robotics framework for deeply embedded platforms. In *2015 IEEE international conference on robotics and automation (ICRA)*. IEEE, 6235–6240.

[49] Julio Alberto Mendoza-Mendoza, Victor Javier Gonzalez-Villela, Gabriel Sepulveda-Cervantes, Mauricio Mendez-Martinez, Humberto Sossa-Azuela, Julio Alberto Mendoza-Mendoza, Victor Gonzalez-Villela, Gabriel Sepulveda-Cervantes, Mauricio Mendez-Martinez, and Humberto Sossa-Azuela. 2020. ArduPilot Working Environment. *Advanced Robotic Vehicles Programming: An Ardupilot and Pixhawk Approach* (2020), 19–46.

[50] Ana Carolina B Monteiro, Reinaldo P Franca, Vania V Estrela, Sandro R Fernandes, Abdeldjalil Khelassi, R Jenice Aroma, Kumudha Raimond, Yuzo Iano, and Ali Arshaghi. 2020. UAV-CPSs as a test bed for new technologies and a primer to Industry 5.0. *Imaging and sensing for unmanned aircraft systems* 2 (2020), 1.

[51] Alex Nanopoulos, Rob Alcock, and Yannis Manolopoulos. 2001. Feature-based classification of time-series data. *International Journal of Computer Research* 10, 3 (2001), 49–61.

[52] Raymond T Ng and Jiawei Han. 1994. Efficient and Effective Clustering Data Mining Methods for Spatial. In *Proceedings of the 20th VLDB Conference, Santiago, Chile*. 12–15.

[53] Hoa T Nguyen, Toan V Quyen, Cuong V Nguyen, Anh M Le, Hoa T Tran, and Minh T Nguyen. 2020. Control algorithms for UAVs: A comprehensive survey. *EAI Endorsed Transactions on Industrial Networks and Intelligent Systems* 7, 23 (2020), e5–e5.

[54] Khoa Dang Nguyen, Cheolkeun Ha, and Jong Tai Jang. 2018. Development of a new hybrid drone and software-in-the-loop simulation using px4 code. In *Intelligent Computing Theories and Application: 14th International Conference, ICIC 2018, Wuhan, China, August 15-18, 2018, Proceedings, Part I 14*. Springer, 84–93.

[55] Kaan Taha Öner, Ertuğrul Çetinsoy, EFE Sirimoğlu, Cevdet Hançer, Mustafa Ünel, Mahmut Faruk Akşit, Kayhan Gülez, and Ilyas Kandemir. 2012. Mathematical modeling and vertical flight control of a tilt-wing UAV. *Turkish Journal of Electrical Engineering and Computer Sciences* 20, 1 (2012), 149–157.

[56] Animesh Patcha and Jung-Min Park. 2007. An overview of anomaly detection techniques: Existing solutions and latest technological trends. *Computer networks* 51, 12 (2007), 3448–3470.

[57] PX4. 2023. *Flight Log*. Retrieved March 25, 2023 from https://docs.px4.io/main/en/log/flight_log_analysis

[58] PX4. 2023. *Flight Review Tool*. Retrieved March 25, 2023 from https://review.px4.io

[59] PX4. 2023. *Issue Report*. Retrieved March 25, 2023 from https://github.com/PX4/PX4-Autopilot/issues/12071

[60] PX4. 2023. *Issue Report*. Retrieved March 25, 2023 from https://github.com/PX4/PX4-Autopilot/issues/11420

[61] PX4. 2023. *Issue Report*. Retrieved March 25, 2023 from https://github.com/PX4/PX4-Autopilot/issues/9150

[62] PX4. 2023. *Issue Report*. Retrieved March 25, 2023 from https://github.com/PX4/PX4-Autopilot/issues/15810

[63] PX4. 2023. *Issue Report*. Retrieved March 25, 2023 from https://github.com/PX4/PX4-Autopilot/issues/12146

[64] PX4. 2023. *Issue Report*. Retrieved March 25, 2023 from https://github.com/PX4/PX4-Autopilot/issues/6772

[65] PX4. 2023. *Issue Report*. Retrieved March 25, 2023 from https://github.com/PX4/PX4-Autopilot/issues/12517

[66] PX4. 2023. *Issue Report*. Retrieved March 25, 2023 from https://github.com/PX4/PX4-Autopilot/issues/18595

[67] PX4. 2023. *Issue Report*. Retrieved March 25, 2023 from https://github.com/PX4/PX4-Autopilot/issues/7737

[68] PX4. 2023. *Issue Report*. Retrieved March 25, 2023 from https://github.com/PX4/PX4-Autopilot/issues/12910

[69] PX4. 2023. *Issue Report*. Retrieved March 25, 2023 from https://github.com/PX4/PX4-Autopilot/issues/15037

[70] PX4. 2023. *Issue Report*. Retrieved March 25, 2023 from https://github.com/PX4/PX4-Autopilot/issues/7202

[71] PX4. 2023. *Issue Report*. Retrieved March 25, 2023 from https://github.com/PX4/PX4-Autopilot/issues/10757

[72] PX4. 2023. *Issue Report*. Retrieved March 25, 2023 from https://github.com/PX4/PX4-Autopilot/issues/14735

[73] PX4. 2023. *Issue Report*. Retrieved March 25, 2023 from https://github.com/PX4/PX4-Autopilot/issues/5110

[74] PX4. 2023. *Issue Report*. Retrieved March 25, 2023 from https://github.com/PX4/PX4-Autopilot/issues/13956

[75] PX4. 2023. *User Guide*. Retrieved March 25, 2023 from https://docs.px4.io/main/en/

[76] PX4. 2023. *User Guide-ROS*. Retrieved March 25, 2023 from https://docs.px4.io/main/en/ros/

[77] Matthes Rieke, Theodor Foerster, Jakob Geipel, and Torsten Prinz. 2012. High-precision positioning and real-time data processing of UAV-systems. *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences* 38 (2012), 119–124.

[78] Peter J Rousseeuw and Annick M Leroy. 2005. *Robust regression and outlier detection*. John wiley & sons.

[79] Tabea Schmidt, Florian Hauer, and Alexander Pretschner. 2020. Automated Anomaly Detection in CPS Log Files: A Time Series Clustering Approach. In *Computer Safety, Reliability, and Security: 39th International Conference, SAFECOMP 2020, Lisbon, Portugal, September 16–18, 2020, Proceedings 39*. Springer, 179–194.

[80] Lwin Khin Shar, Wei Minn, Ta Nguyen Binh Duong, Jiani Fan, Lingxiao Jiang, and Daniel Lim Wai Kiat. 2022. DronLomaly: Runtime Detection of Anomalous Drone Behaviors via Log Analysis and Deep Learning. In *29th Asia-Pacific Software Engineering Conference, APSEC 2022, Virtual Event, Japan, December 6-9, 2022*. IEEE, 119–128. https://doi.org/10.1109/APSEC57359.2022.00024

[81] Jianhua Shi, Jiafu Wan, Hehua Yan, and Hui Suo. 2011. A survey of cyber-physical systems. In *2011 international conference on wireless communications and signal processing (WCSP)*. IEEE, 1–6.

[82] Swardiantara Silalahi, Tohari Ahmad, and Hudan Studiawan. 2023. Transformer-based Sentiment Analysis for Anomaly Detection on Drone Forensic Timeline. In *11th International Symposium on Digital Forensics and Security, ISDFS 2023, Chattanooga, TN, USA, May 11-12, 2023*. IEEE, 1–6. https://doi.org/10.1109/

ISDFS58141.2023.10131749

[83] Hudan Studiawan, Ahmad Firdaus, Baskoro A Pratomo, and Tohari Ahmad. 2023. Anomaly Detection on Drone Forensic Timeline with Sigma Rules. In *2023 International Conference on Emerging Smart Computing and Informatics (ESCI)*. IEEE, 1–5.

[84] Gian Antonio Susto, Angelo Cenedese, and Matteo Terzi. 2018. Time-series classification methods: Review and applications to power systems data. *Big data application in power systems* (2018), 179–220.

[85] David MJ Tax and Robert PW Duin. 1999. Support vector domain description. *Pattern recognition letters* 20, 11-13 (1999), 1191–1199.

[86] Max Taylor, Jayson Boubin, Haicheng Chen, Christopher Stewart, and Feng Qin. 2021. A study on software bugs in unmanned aircraft systems. In *2021 International Conference on Unmanned Aircraft Systems (ICUAS)*. IEEE, 1439–1448.

[87] John Tisdale, ZuWhan Kim, and J Karl Hedrick. 2009. Autonomous UAV path planning and estimation. *IEEE Robotics & Automation Magazine* 16, 2 (2009), 35–42.

[88] Darren Turner, Arko Lucieer, and Steven M De Jong. 2015. Time series analysis of landslide dynamics using an unmanned aerial vehicle (UAV). *Remote Sensing* 7, 2 (2015), 1736–1757.

[89] Michael Vierhauser, Jane Cleland-Huang, Sean Bayley, Thomas Krismayer, Rick Rabiser, and Pau Grünbacher. 2018. Monitoring CPS at runtime-A case study in the UAV domain. In *2018 44th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*. IEEE, 73–80.

[90] 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*. 20–31.

[91] Haijun Wang, Haitao Zhao, Jiao Zhang, Dongtang Ma, Jiaxun Li, and Jibo Wei. 2019. Survey on unmanned aerial vehicle networks: A cyber physical system perspective. *IEEE Communications Surveys & Tutorials* 22, 2 (2019), 1027–1070.

[92] Jun Wu, Jinsong Wang, and Zheng You. 2010. An overview of dynamic parameter identification of robots. *Robotics and computer-integrated manufacturing* 26, 5 (2010), 414–419.

[93] Liang Yang, Juntong Qi, Jizhong Xiao, and Xia Yong. 2014. A literature review of UAV 3D path planning. In *Proceeding of the 11th World Congress on Intelligent Control and Automation*. IEEE, 2376–2381.

[94] Fiorella Zampetti, Ritu Kapur, Massimiliano Di Penta, and Sebastiano Panichella. 2022. An empirical characterization of software bugs in open-source cyber–physical systems. *Journal of Systems and Software* 192 (2022), 111425.

[95] Lei Zeng, Yang Xiao, Hui Chen, Bo Sun, and Wenlin Han. 2016. Computer operating system logging and security issues: a survey. *Security and communication networks* 9, 17 (2016), 4804–4821.

[96] Tianzhu Zhang, Han Qiu, Gabriele Castellano, Myriana Rifai, Chung Shue Chen, and Fabio Pianese. 2023. System Log Parsing: A Survey. *IEEE Transactions on Knowledge and Data Engineering* (2023).

[97] Jiang Zhaoxue, Li Tong, Zhang Zhenguo, Ge Jingguo, You Junling, and Li Liangxiong. 2021. A survey on log research of aiops: methods and trends. *Mobile Networks and Applications* 26, 6 (2021), 2353–2364.

[98] zhihu. 2023. *news*. Retrieved March 25, 2023 from https://zhuanlan.zhihu.com/p/423739596