2022 29th Asia-Pacific Software Engineering Conference (APSEC) | 978-1-6654-5537-4/22/$31.00 ©2022 IEEE | DOI: 10.1109/APSEC57359.2022.00024

DronLomaly: Runtime Detection of Anomalous Drone Behaviors via Log Analysis and Deep Learning

Lwin Khin Shar, Wei Minn, Nguyen Binh Duong Ta
Singapore Management University
Singapore
Email: {lkshar, weiminn, donta}smu.edu.sg

Jiani Fan
Nanyang Technological University
Singapore
Email: jiani001@e.ntu.edu.sg

Lingxiao Jiang, Daniel Lim Wai Kiat
Singapore Management University
Singapore
Email: {lxjiang, waikiat.lim.2019}@smu.edu.sg

***Abstract*—Drones are increasingly popular and getting used in a variety of missions such as area surveillance, pipeline inspection, cinematography, etc. While the drone is conducting a mission, anomalies such as sensor fault, actuator fault, configuration errors, bugs in controller program, remote cyber-attack, etc., may affect the drone's physical stability and cause serious safety violations such as crashing into the public. During a flight mission, drones typically log flight status and state units such as GPS coordinates, actuator outputs, accelerator readings, gyroscopic readings, etc. These log data may reflect the above-mentioned anomalies. In this paper, we propose a novel, deep learning-based log analysis approach for detecting anomalies in the drone log that could lead to physical instabilities. We train a LSTM-based deep learning model on the normal flight logs produced by a baseline drone. Essentially, the model learns the sequential patterns of flight state units and correlations among them. The model can then be used to detect anomalies in the state units as the log entries are being recorded by the drone's control program at runtime. In our experiments, we built detection models based on several logs produced by 3 different drone control programs, namely DJI, ArduPilot and PX4, and used them to detect anomalies in the logs. On average, our approach achieves 0.968 recall and 0.963 precision, and it can detect anomalies during runtime within a few milliseconds.**

***Index Terms*—Drone security; anomaly detection; log analysis; deep learning**

I. Introduction

Drones refer to aerial vehicles that do not have a human pilot on board. They have been increasingly used in missions such as military operations, surveillance, infrastructure inspection, package delivery, crop dusting, first aid, emergency response, etc. As more and more drones are flying in public airspace, safety concerns are also rising. In the past decade, drones have often been reported as causing hazards to aircraft or train, or to people and property on the ground [1]. For example, in 2016, a drone collided with a travelling train in United Kingdom [2].

While conducting a flight mission, several factors such as an input validation vulnerability in the drone controller program, a sensor fault that incorrectly reads the surrounding obstacles, or a remote cyber-attack that intentionally produces noises to confuse the sensors may affect the drone physical stability, resulting in a crash into public area or in the wrong hand. For example, Kim et al. [3] and other studies [4], [5] found multiple input validation bugs in ArduPilot and PX4 control programs. Son et al. [6] demonstrated that drone's gyroscopic sensors can be incapacitated by producing malign sound noises with resonant frequencies. They also showed that the sensors' frequencies can be easily scanned by a consumer-grade speaker. Hence, conducting flight missions in populated areas poses serious risks, as also highlighted by several incidents [7], [8]. Therefore, anomaly detection of drone physical states when a drone is conducting its mission is important.

Existing approaches are not suitable for detection such anomalies at runtime. State-of-the-art drone fuzzing approaches such as RVFuzz [3], PGFuzz [4], and LGDFuzzer [5] are effective at finding input validation and semantic bugs in control programs. LGDFuzzer, in particular, is a learning-based approach that uses an flight state predictor based on an LSTM model. However, the state predictor is generated at the start using prior flight logs and there's no more learning during the fuzzing process i.e. it is an offline approach. As such, certain bugs may go undetected as the fuzzer is not trained to recognized anomalies in new experiences and circumstances.

Drones, especially industrial and military grade drones, usually record flight data at runtime. The log contains multiple entries; each entry typically contains timestamp, flight status and state units. Given a baseline data, these runtime log data can be leveraged to detect potential physical instabilities of the drone. There are log analysis approaches such as LogAnomaly [9], DeepLog [10], and LogRobust [11] which can analyse and detect anomalies in the logs online. But these approaches have been designed specifically for different kinds of logs such as print logs from web-based micro-services, network access logs, etc., and for detecting software anomalies such as abnormal API usages or network intrusion detection. Therefore, these approaches are not suitable for detecting unstable physical states of the drone.

In this paper, we develop a novel log analysis-based deep learning approach called *DronLomaly*, specifically designed for detecting flight log anomalies that could result in physical

978-1-6654-5537-4/22/$31.00 ©2022 IEEE
DOI 10.1109/APSEC57359.2022.00024

instabilities online (while the drone is conducting a flight mission). More specifically, *DronLomaly* takes as input a list of flight logs of a good drone which successfully executed a given flight mission repeatedly, for training the model. Each log entry contains a timestamp, flight status (e.g., flight mode, battery status, configuration values), and state units (e.g., GPS coordinates, acceleration and rotation on X, Y and Z axis). Essentially, state units indicate the physical conditions of the drone. *DronLomaly* learns a LSTM-based deep learning model on those logs. LSTM has the ability to learn long-term dependencies over sequences, which makes it suitable for our context where it is required to predict the next possible state given the sequence of past states. The input to our LSTM model is a history of recent flight states and the output is the next possible flight state (predicted state). If the actual state recorded in the log significantly deviates from the predicted state, anomaly is reported. It is possible that the reported anomaly is a false alarm. For example, the drone may encounter environmental conditions that were not present in training missions and as a result, the actual flight state may deviate from the states observed during the training. While such deviation itself may not be desirable (for example, the mission is sensitive and requires the drone to keep the states as close as possible to those observed during the training), it may not cause the physical instability of the drone. To mitigate this, we design *DronLomaly* to incrementally learn the new normal patterns, without requiring re-training from scratch.

We evaluated *DronLomaly* on several logs produced by 3 control programs — DJI, ArduPilot and PX4. DJI is a closed source control program and to our knowledge, it is not well investigated in the context of anomaly or bug detection, even though DJI holds 76% of the global drone market according to Statista [12]. ArduPilot and PX4 are open source control programs which are widely used by drone manufacturers such as Parrot and Mamba. In total, our experiments include 427,938 log entries, containing 4,181 anomalies. We train the models on 80% of the log entries that do not include anomalies and tested on the other 20% of the log entries. The results show that our approach has 0.968 recall and 0.963 precision, and it can detect the anomalies during runtime within milliseconds.

Our contributions in this paper include:

1) We designed and implemented *DronLomaly* to detect drone physical instabilities at real time. We use a LSTM deep learning model to learn the sequential log data that characterize normal physical states. We then apply the model to predict the next possible states given a set of recent log entries. If the actual states significantly deviates from the predicted states, anomaly is reported.
2) We also provide a mechanism for users to update *DronLomaly* using the false positives and improve its accuracy.
3) We also deployed and tested the trained models on a Raspberry Pi device, a companion computer for autonomous drones. We compared the performance of model inference on different types of logs (DJI, ArduPilot and PX4), discussed the feasibility of deploying model in real-life use cases.
4) We have open sourced *DronLomaly* at [13]; the site makes available the tool and the dataset.

II. Background and Motivation

A. Drone System

Typically, a drone system consists of Ground Control Station (GCS), Remote Controller (RC), Companion Computer, Flight Controller, Sensors, and Motors. Figure 1 shows a block diagram of a drone system.

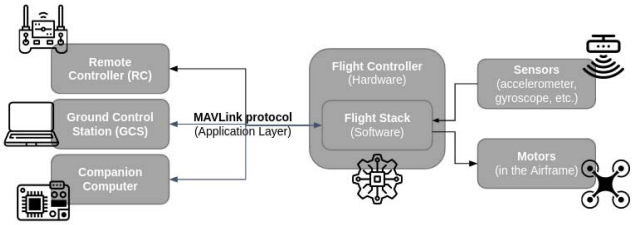

Fig. 1: A typical drone system

Flight Stack (software side of the Flight Controller) is the heart of the drone that controls the drone's movements and operations by using information from the sensors that represents the physical state of the drone such as Global Positioning System (GPS), Barometer, Accelerometer, Gyroscope, and Thermometer, and orchestrating the motors on the airframe accordingly. Most open-source Flight Stacks expose APIs that support the MAVLink protocol (de-facto standard for UAV communication on the Application Layer) that is used by Remote Controllers (Wirelessly), Ground Control Stations (Wirelessly) and Companion Computers (Wired and physically attached to the drone and typically powered by the drone's batteries). Companion Computer is usually a lightweight computer such as Raspberry PI inside which the custom-written business application invokes high-level SDKs such as MAVSDK that uses MAVLink protocol to communicate to the Flight Controller to achieve its business objectives such as Waypoint Missions, Object Detection and Structural Inspections, and other payload operations such as camera actions and robotic hands.

As an example, gyroscope detects angular velocity in three axis. So it can detect rate of change of angle in pitch, roll and yaw. Likewise, accelerometer measures a drone's acceleration along the three axis X, Y, and Z. If the drone is stationary in horizontal position then its X and Y axis will give $0g$ output whereas z-axis will give $1g$ output. $1g$ is the gravity which is experienced by every object on earth. During the tilt X, Y and Z will give output which lies between $0g$ and $1g$. The values can then be applied to trigonometry formulas to arrive at the tilt angle of drone. As such, these sensor values can be used to detect the physical states of the drones e.g., position, velocity, attitude, magnetic field interference, vibration level, etc.

Drones are expected to be stable and do not shake, wobble or tilt unexpectedly. Otherwise, it will lose its balance and fall down. It is very important for the Flight Controller to receive accurate sensor readings from such sensors so that it

can send adequate actuator commands to stabilize the drone. Sensors that are faulty or unable to function well under certain conditions (e.g., extreme weather or incorrectly configured parameter values) may cause drone incidents. For example, a drone may be tilting when hovering. This is quite a common issue that occurs when the accelerometer or the gyroscope was not properly calibrated, e.g., it was not calibrated on a flat surface. In such events of critical component failures, flight controllers are programmed to trigger Fail-Safe mode which comprises performing automated procedures such as Emergency Landing or Return-to-Home in order to protect the drone from sustaining or incurring physical damages in the current erratic state, especially during autonomous modes which does not involve the supervision of a human drone operator.

B. Flight Logs in the Drone

Majority of the drones, especially industrial drones, logs flight data during the flight. This data typically includes flight status, sensor readings, and other information such as configuration parameter values. It essentially serves as the electronic flight recorder for the drone. Different types of drones may use different file format (e.g., DAT, TXT, CSV, etc.) and data structure to record the data. As an example, Figure 2 shows the data structure of DJI flight log files. "Packet Type" header indicates the type of the packet. There are eight different packet types, which are GPS, motor, home point, remote control, tablet location, battery, attitude, and flight status. "Payload" contains the actual information according to the packet type. 'Motor' payload indicates the speed and the load of motors on the drone. 'Home point' states the home point coordinates of the drone. 'Remote control' states the remote control status such as throttle, rudder, and elevator. 'Tablet location' reports the latitude and longitude of the tablet during a "Follow Me" mission. 'Battery' indicates the battery status such as battery capacity, temperature, current, and voltage. 'Attitude' provides the pitch, roll and yaw readings. 'GPS' provides the GPS location of the drone, altitude, 3-axis acceleration, gyro, and other sensor readings. 'Flight status' provides flight mode (autopilot, go home, hover, etc.), other configuration parameters (gain, agility, max speed limit, etc.), and also the flight time in milliseconds since the start of the flight.

In the context of our work, we extract 'GPS', 'Attitude', 'Battery', and 'Flight status' from DJI logs. Figure 3 shows an excerpt of a log extracted from a DJI drone used in our experiments. Same kind of information can also be extracted from other types of drones although it may be in different format and data structure.

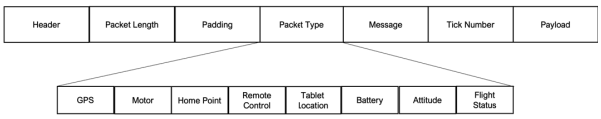

Fig. 2: Packet Structure of DJI DAT files

velocity_x	velocity_y	velocity_z	status_mode	rtk_connect_status
1.49978	1.00E-06	-0.001255	6	1
1.499783	2.00E-06	-0.001428	6	1
1.499787	2.00E-06	-0.001608	6	1
1.499792	3.00E-06	-0.001815	6	1

Fig. 3: A sample DJI log. 'velocity_x/y/z' values are in m/s; 'status_mode' refers to the mode the drone is currently in ('6' means simulation); 'rtk_connect_status' refers to the GPS connection status.

C. Threat Model

Our threat model includes threats that may cause drone incidents. According to drone incidents reported in the drone incident databases [1], [14], [15], the causes of drone incidents can be categorized into the following:

1) Human error: drone operator loses control of the drone
2) Violation of regulation: e.g. flying over non-fly zone or flying above the allowed altitude
3) Communication issue: e.g. loss of communication between ground control station (GCS) and the drone
4) Hardware fault: e.g., faulty sensors or interferences
5) Software configuration issue: setting invalid configuration parameter values
6) Bad weather: e.g. extreme temperature and wind

In principal, online flight log anomaly detection when the drone is flying for a particular flight mission can deal with these issues, except human error. This is because those issues will most likely manifest into deviations from normal physical states that are observed when a good drone executes the same flight mission without any issue[1], and we can expect the companion computer that is running online log anomaly detection to detect those deviations and perform appropriate fail-safe actions to shepherd the drone to safety. Therefore, the last five categories represents our threat model and serves as our motivation for this work.

As an example, Figure 4 shows two scenarios. In scenario (a), the norm of the magnetic sensor measurement is a constant throughout the whole flight regardless of the drone's thrust. This is a normal scenario. In scenario (b), the two measurements now seem to be correlated; the current drawn by the motors may be influencing the magnetic field. And this may lead to incorrect yaw estimation, and possibly to drone crashing. Hence, scenario (b) represents an abnormal case, which deviates from the normal scenario. Our log anomaly analysis approach aims to detect such cases.

III. Methodology

A. Overview

Figure 5 shows the overall workflow of our approach. We use the bidirectional Long Short-Term Memory (LSTM) to build the prediction model. Given a sequence h of feature

[1]Regarding regulatory violations, it is assumed that the flight mission by a good drone was conducted without violating regulatory constraints.

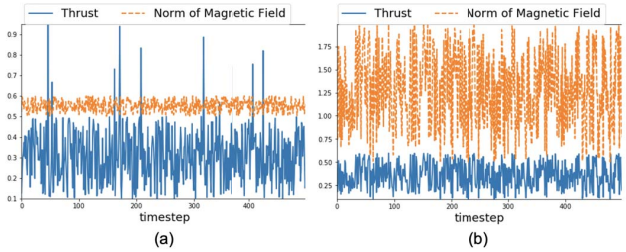

Fig. 4: Plot of a drone's thrust and norm of magnetic field, which have (a) no correlation (normal), and (b) strong correlation

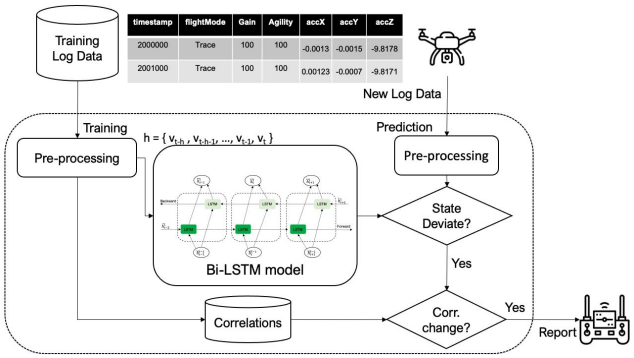

Fig. 5: The workflow of our approach

vectors that characterise the physical states of the drone, it predicts the next possible feature vector (physical state).

More specifically, each feature vector v consists of flight status f and state units a, e.g., $v = \{$flight Mode, Gain, accelX, accelY, accelY$\}$. Flight status refers to flight status information such as flight mode and Gain configuration value; state units refers to sensor readings such as acceleration at X-axis, Y-axis, and Z-axis. In short, we shall refer to this vector as $v = \{f, a\}$. This vector is normalized so that the values fall within the range of 0 and 1.

Our model takes as input a sequence h of pre-processed feature vectors with timestamp t (i.e., $h = \{v_{t-h}, v_{t-h-1}, \ldots, v_{t-1}, , v_t\}$). It then predicts the next state units a'_{t+1}. If an actual state unit (e.g., a sensor reading) observed in the current log is different from the predicted state unit, the corresponding state unit is flagged. *DronLomaly* then performs correlation analysis between the flagged state unit and other units, based on the whole current log data. If there is any significant change among the correlations, in comparison with the baseline correlation data, anomaly is reported. The rationale of incorporating correlation analysis in addition to LSTM-based prediction is to reduce the false positive cases due to new stable states that were not observed in the training stage. Furthermore, LSTM model makes prediction based on h most recent log entries whereas correlation analysis considers the whole log and hence, they are complementary.

In the subsequent subsections, we discuss the details of our approach.

B. Prediction Model

LSTM stands for Long Short-Term Memory Model, which is a class of deep neural networks tailored to deal with time-series or sequential data. It has two key vectors — one represent the short-term state which keeps the output at the current time step and another represents the long-term state which stores, reads, and rejects items meant for the long-term while passing through the network. The decision of reading, storing, and writing is based on some activation functions, e.g. ReLU or Sigmoid. The output from those activate functions is a value between (0, 1). Bidirection LSTM model is an extension of the LSTM model. In addition to learning from the sequence of the input as it is, it also learns from the reverse

Step	Sequence		Next
3	[f_0 a_0 f_1 a_1 f_2 a_2]	=>	a_3
4	[f_1 a_1 f_2 a_2 f_3 a_3]	=>	a_4
5	[f_2 a_2 f_3 a_3 f_4 a_4]	=>	a_5
6	[f_3 a_3 f_4 a_4 f_5 a_5]	=>	a_6

Fig. 6: Sequential patterns of flight log data, used for training the Bidirectional LSTM model

of that sequence. Since it learns from both the past and the future, it may be able to learn the context better. As such, LSTM model makes it suitable for our context where it is required to predict the next possible state given the sequence of past states.

Idea. Our idea is that normal flight logs containing the records of a (successful) flight mission have sequential patterns of flight states. Hence, for a given sequence of feature vectors that characterise flight states, its next flight state would be predictable if no anomaly occurs.

Input. Let $\Omega = \{(f_1, a_1), (f_2, a_2), \ldots (f_n, a_n)\}$ be the whole set of feature vectors, extracted from a given log of a flight mission. The sequence for detection is a sliding window of the h most recent feature vectors. This is the input to our prediction model. As an example, Figure 6 shows the sequential patterns of the flight states, where $h = 3$. As we can observe from Figure 6, our data is a multi-variate time series data where (bidirection) LSTM model can be applied. We normalize the values in each vector by the average and the standard deviation of all values from the same parameter position from the training data.

Prediction Model. Our model has one bidirectional LSTM layer with a drop out of 0.1, one fully-connected linear layer with 128 neurons and 'ReLU' activation function, and one output layer. For the multi-variate time series data, the training process tries to adjust the weights of its LSTM model in order to minimize the error between a prediction and an observed feature vector. Mean Squared Error (MSE) loss is used to minimize the error. This process is done in an iterative manner, which is determined by the Epoch value. We set *Epoch* to 2000

with *Early Stopping* which is reached when the validation loss does not decrease below 0.001 continuously for 20 iterations.

Optimal values of h and model hyper-parameter values are determined through updating them iteratively and checking whether the predicted state values are closer to the ground truth.

Output. The output is a real value vector as a prediction for the next state units, based on a sequence of feature vectors from recent history.

Anomaly Detection. Instead of setting a magic error threshold for anomaly detection purpose in an ad-hoc fashion, we compute the mean and the standard deviation σ of all values of each state unit observed in the training. Assuming a normal distribution, 99% of the values should be within 3σ of the mean value and therefore, we consider that if a value lies outside of 3σ, it is an outlier (anomaly). We therefore define a threshold called *deviation threshold* $\varsigma = 3\sigma$. At deployment (in detection phase), if the error between the predicted state unit and the observed state unit is larger than ς, the particular state unit is flagged as a likely anomaly.

Since the feature vector characterises the physical states of the drone, this method is able to detect physical instabilities of the drone, which may be caused by hardware fault, software configuration issue, violation of regulation, or bad weather. For example, due to an acceleration sensor fault, the drone's control program may not receive the correct reading of the drone's acceleration. Therefore, it may try to correct the deviation and yet the sensor fault may prevent a successful correction. And as a result, the control program may continue to correct, resulting in physical instability of the drone and causing a crash. Figure 7 shows a timeseries plot depicting the deviation from the expected acceleration on X-axis around the timesteps 40 and 100.

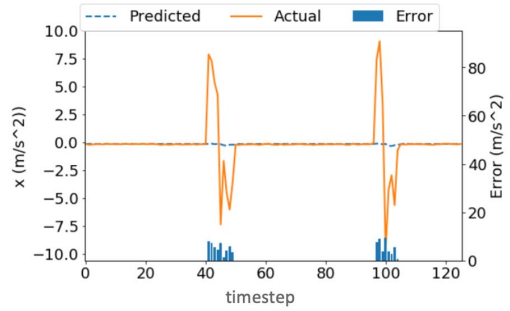

Fig. 7: Actual values vs predicted values of a state unit 'acceleration on X-axis'. The histogram at the bottom indicates the errors between these values.

When an anomaly of particular sensor unit is detected in this phase, instead of sending an alert to the user immediately, *DronLomaly* performs the correlation analysis with regards to that unit, to reduce false alarms.

C. Correlation Analysis

Several factors such as different environmental conditions and configurations may result in different drone states, which are not observed in the training logs but are considered stable. Our prediction model may yield those cases as abnormal, which would be false alarms. To reduce such cases, Pearson correlation analysis is used to augment the prediction model. Pearson correlation measures the strength of the linear relationship between two variables. It has a value between -1 to 1, with a value of -1 meaning a total negative linear correlation, 0 being no correlation, and +1 meaning a total positive correlation.

DronLomaly keeps a baseline record of Pearson correlations among each pair of sensor readings, e.g., acceleration sensor readings on X-axis vs Y-axis, which are computed based on the training logs. A matrix structure is used to store the pairwise correlation coefficients. Figure 8 shows an example, derived from a DJI log.

	AccX	AccY	AccZ
AccX	1.0	0.035	0.0346
AccY	0.035	1.0	0.08
AccZ	0.0346	0.08	1.0

Fig. 8: Correlations among pairs of acceleration sensor readings on X, Y, Z-axis

Here, we define another threshold called *correlation threshold* $\rho = 0.1$[2]. When deviation of a sensor unit is detected by the prediction model, *DronLomaly* then takes the whole current log and computes the correlations between the flagged sensor unit and each of the other state units. If any correlation change is larger than ρ (i.e., ± 0.1 correlation coefficient), anomaly is reported. In an actual deployment, alert message would be sent back to the GCS so that the operator can take an appropriate action.

D. Update of the Anomaly Detection Model

The training data may not cover all possible stable states. Several factors such as different environmental conditions and configurations may result in new different drone states which may not be abnormal but might still be flagged as abnormal by our model as it has not been trained on those states. Hence, it is important that our prediction model can incrementally update its baseline with newly observed stable states. *DronLomaly* provides a mechanism in which the user upon observing a false positive, can adjust the weights of the trained model to adapt to the new stable state. The update process is just like the training process discussed in III-B, where h sequence of the recently recorded drone state (false positive cases) is fed into the Bidirectional LSTM model as the new training data. The weights of the model are adjusted to form an improved baseline for prediction with lower the error between the model output and the actual observed values from the false positive

[2]This threshold was determined through some preliminary experiments.

cases. Afterwards, the prediction model is saved as a multi-dimensional weight vector (in the form of *.h5 file), which is reloaded to perform further predictions and online learning.

IV. Evaluation

We consider that *DronLomaly* would be practically useful if it can effectively detect anomalous states. And it is also important that the detection model is efficient (low runtime overhead) so that it can be deployed in the actual drone. As a result, the following research questions are investigated in this section:

- *RQ1* - Effectiveness: can *DronLomaly* effectively detect anomalies in the flight logs? That is, what is the recall, precision, and F-measure?
- *RQ2* - Efficiency: can *DronLomaly* be run in the computing device that can actually be used to control/fly the drones? More specifically, how long does it take for *DronLomaly* to read the current log entry and detect anomalies in such a device?

A. Experiment Design

Datasets. We use 3 sets of flight logs generated by DJI, ArduPilot, and PX4 control programs. DJI is a closed source control program. ArduPilot and PX4 are open source control programs. ArduPilot logs are taken from LGDFuzzer [5] which were generated by repeatedly flying a flight mission called *AVC2013* [16]. We obtained PX4 logs by flying a few flight missions in the Software-In-The-Loop (SITL) setup that comprises Gazebo simulator, PX4 and QGroundControl running on Ubuntu 20.04 environment [17]. Inspired by AVC2013, we further augmented the generalizability of the PX4 logs data by randomly generating the waypoints inside the perimeter that is defined by GPS coordinates of a set of Polygon corners as demonstrated by [18]. We collected DJI logs by flying a physical DJI drone, DJI Matrice 300. Firstly, we used the waypoint generator provided in DJI OSDK [19] to generate GPS Longitude and Latitude waypoints in a shape of a 10-sided polygon with 5 repeats. We then uploaded the generated waypoints into the drone. During the flight, we use the telemetry program to extract the logs.

We manually injected faulty values in the logs in a way that those values reflect faulty sensor readings, actuator outputs, GPS and Magnetic field interferences, and communication signal losses. We discussed how this is done in more detail in Section IV-B. Table I shows the details of the datasets. Essentially, DJI dataset contains faulty velocity sensor readings and communication signal losses; ArduPilot dataset contains faulty accelerator and gyroscope sensor readings. PX4 dataset contains faculty accelerator and gyroscope sensor readings, actuator faults, and strong GPS and Magnetic field interferences.

Training and Testing. Given each dataset (Table I), we split its log entries into 80% and 20%, ensuring that 80% of log entries contain only the normal entries. We then use the 80% set for training (no anomalies were used in the training) and the 20% set for testing.

TABLE I: Datasets. 'entries' refers to the total number of log entries including both normal and anomalous entries; 'acc', 'gyro', 'vel' refer to the number of anomalous entries that represent faulty accelerator, gyroscope, and velocity senor readings, respectively. Similarly, 'actu' refers to the number of drone actuator faults; 'gps' and 'mag' refer to strong GPS and magnetic field interferences, respectively; 'com' refers to the number of communication signal losses among 'entries'.

Dataset	entries	acc	gyro	vel	actu	gps	mag	com
DJI	229,328	0	0	500	0	0	0	708
ArduPilot	144,176	500	500	0	0	0	0	0
PX4	54,434	887	887	0	75	12	112	0

Hardware. Training of the models were conducted on a Mac OS machine with i7 2.6 GHz CPU and 16 GB memory. The trained models were tested on Raspberry Pi 4 Model B with 8 GB memory (for the sake of RQ2).

We implemented *DronLomaly* with Python 3.7 and Keras 2.4.

B. RQ1: *Effectiveness*

To measure the effectiveness of *DronLomaly*, we use Recall, Precision, and F-measure; which are standard measures typically used for evaluating anomaly detection approaches [9]–[11], [20]. They are defined as follows:

- *Recall* $Pr = tp/(tp + fn)$
- *Precision* $Pd = tp/(tp + fp)$
- *F-measure* $= 2 \times (Pr \times Pd)/(Pr + Pd)$.

where tp is the true positives (detected anomaly), fn is the false negatives (missed anomaly), and fp is the false positives (non-anomaly reported as anomaly). Recall measures the ability of the model to find all the relevant cases (true positives – anomaly) within a dataset. Precision measures the ability of the model to identify *only* the relevant cases. *F-measure* is the harmonic mean of precision and recall. It reports a balance between precision and recall.

Table II shows the results. On average across all types of anomaly, it achieves 0.968 recall, 0.963 precision, and 0.956 F-measure. In all types of anomaly, it achieves the F-measure of 0.9 and above. Therefore, this answers RQ1 that *DronLomaly* is highly effective at detecting anomalies in drone logs.

TABLE II: Results

Type of Anomaly	Recall	Precision	F-measure
DJI-vel	0.986	0.948	0.967
DJI-com	1.0	0.999	0.999
ArduPilot-acc	0.998	0.919	0.956
ArduPilot-gyro	0.904	0.946	0.924
PX4-acc	1.0	0.994	0.997
PX4-gyro	0.997	0.939	0.9
PX4-actuator	0.907	1.0	0.951
PX4-gps	0.917	1.0	0.957
PX4-mag	1.0	0.918	0.957

In the following, we discuss the results in more detail. We explain our fault injection strategy to achieve abnormal physical states that are realistic. We also discuss how *DronLomaly* dealt with the threat model that we discussed in Section II.

High vibration levels. When the drone's vibration level is too high, it could be due to hardware fault, software configuration issue, or bad weather. High level of vibration is one of the most common problems for multirotor drones. This can lead to problems such as motors heating up, sensor clipping, or position estimation failures that result in fly-aways. The good news is that this problem often manifests into abnormal sensor measurement values such as overlapping accelerator measurements along Z-axis and X-/Y-axis, which can be detected by anomaly detection models like ours. An example of such a manifestation is shown in Figure 9. In our experiments, we simulated this behavior with PX4 dataset. *DronLomaly* achieved 1.0 recall and 0.994 precision for this particular anomaly (refer to 'PX4-acc' in Table II). We observed that *DronLomaly* can detect the cases well especially when the fault injection resulted in a high correlation between acceleration at Z-axis and acceleration at X-/Y-axis.

Sensor calibration errors. Sensors often need to be calibrated, especially after flying through extreme environmental conditions. When the sensors are not calibrated or improperly calibrated, it may causes instabilities. This can be considered as both hardware fault and software configuration issue. This could also manifest into abnormal sensor measurement values such as the one shown in Figure 9. Similar to the above, fault injection was done to accelerometer, gyroscope, and velocity sensor values in Ardupilot, PX4, and DJI datasets. *DronLomaly* generally achieved good recall and precision (refer to 'DJI-vel', 'ArduPilot-acc', 'ArduPilot-gyro', 'PX4-gyro' in Table II).

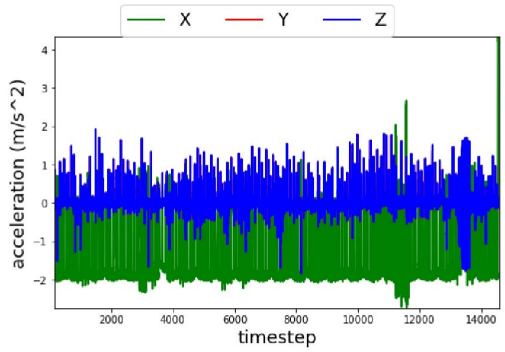

Fig. 9: High correlation between acceleration at Z-axis and acceleration at X-/Y-axis could indicate high level of vibration (highly unstable)

Magnetic field interference. The magnetic field should be uncorrelated with the thrust of the drone. If it is correlated, it is likely that the current drawn by the motors is influencing the magnetic field, which is an abnormal physical state as it could lead to incorrect yaw estimation. We injected faults that reflect such a scenario in PX4 dataset (see example in Figure 4b). *DronLomaly* detected all injected faults and produced 10 false alarms (refer to 'PX4-mag' in Table II).

GPS signal interference. GPS signal tends to be weak and thus it can be interfered/jammed by components transmitting or radiating in a frequency used by the GPS. To simulate this signal interference behavior, we injected faults in the PX4 log file by increasing the noise indicator values at certain time-steps, as shown in an example in Figure 10. Our model detected most of the injected cases except one (refer to 'PX4-gps' in Table II).

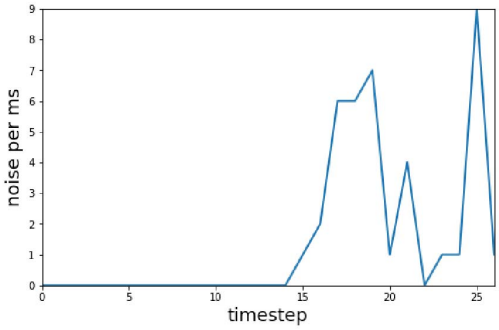

Fig. 10: GPS signal noise

Actuator faults. If the actuator outputs are outside the normal range for a long period of time, it could be an indication that the drone is imbalanced, the controller runs into saturation, or the drone is carrying a weight that requires more than the thrust it can support. Similar to the GPS signal interference case, we created log file by injecting the actuator output values above the normal range for 100 continuous time-steps. Our model detected most of the cases except seven cases (refer to 'PX4-actuator' in Table II). The 7 false negatives were due to some of the original values (non-anomalies) in the dataset also fall outside of the actuator's normal range at certain timesteps. Hence, when our model encounters the injected abnormal values initially, it does not consider them as anomalies until this continues for a few continuous timesteps.

Communication issues. Due to various problems such as large distance, encapsulated surroundings, faulty communication component, or signal jamming, there may be a loss of communication between the GCS and the drone, or between the GPS satellite and the drone. Such issues can be easily detected by monitoring the state units such as GPS coordinates, rc_signal and mavlink_signal. For example, if GPS coordinate values in the log indicate zero for a certain period of time, our model will flag this as anomaly, since it would observe that GPS values are generally not zero in the normal logs. We injected such faults in the DJI dataset. Our model detected all the injected cases and produced one false alarm (refer to 'DJI-com' in Table II).

C. RQ2: *Efficiency*

To evaluate the efficiency of *DronLomaly*, i.e., the runtime performance when we deployed *DronLomaly* on a Raspberry Pi 4 Model B with 8 GB memory, which can actually be used to control/fly a drone. We then ran a script to load our models and use the models to predict each log entry in the test set, in

the Raspberry Pi device. We collect the time taken for loading the models and predicting as each log entry is fed to the model at real time. On average, *DronLomaly* takes about 2 minutes to load a model. However, this is a one time cost. Figure 11 shows the boxplot of the time taken for predicting each of the log entries in the test datasets. We can observe that it takes a few milliseconds to analyze a log entry, which we deem is very reasonable.

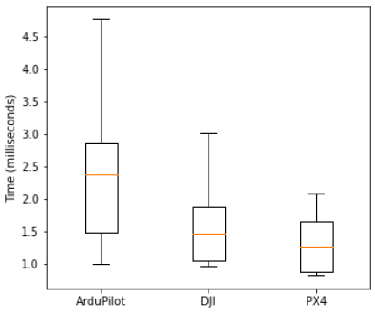

Fig. 11: Boxplot of *DronLomaly*'s runtime performance when deployed in Raspberry Pi

D. Limitations

In our preliminary experiments, we observed that *DronLomaly*'s performance varies with different h values and the larger h results in a better performance. But we did not investigate further about how to determine the optimal sliding window size in the context of drone anomaly detection. Likewise, we also did not investigate much about the impact of using different thresholds (i.e., *deviation threshold* and *correlation threshold*) for reporting anomalies. Also, when flying the drones to generate DJI and PX4 logs, we used default configuration parameters. Therefore, we did not consider the impact of varying configuration parameter values. In this work, we dealt with 7 types of fault data in the logs. But there are also other kinds of faults that we did not consider such as the faults that may be induced by invalid or arbitrary flight commands given by human users. We plan to address these problems as future work.

V. Related Work

A. Log-based anomaly detection for software systems

Log analysis for anomaly detection is a well-known problem in software engineering. Existing approaches such as LogAnomaly [9], DeepLog [10], and LogRobust [11] analyse and detect anomalies in the runtime logs of web-based microservices and networked devices. In particular, LogAnomaly models semi-structured system logs as natural language sequences using a word embedding-inspired template representation. It then trains a LSTM network to automatically detect both sequential and quantitative anomalies. LogRobust [11] shows that in practice, log data usually exhibits unseen events and sequences, which is the key reason why prior methods assuming log data is stable over time do not perform well. LogRobust uses an attention-based Bi-LSTM model to capture the importance of different log events and to handle unstable events. The approach has been evaluated on logs from Hadoop and an actual Microsoft online service. DeepLog [10] also makes use of the LSTM model to learn log patterns from normal software executions as baselines. It can then detect anomalies when log patterns deviate from these baselines. In addition, DeepLog builds system workflows to help with root cause analysis when anomalies are detected. We note that these approaches mainly target software anomalies such as abnormal API usages or intrusion detection in computing systems. Our work instead focuses on detecting unstable physical states in drones using log files.

The latest approaches (2022) in this area include DeepTraLog [20], and an empirical study by Le and Zhang [21]. DeepTraLog leverages deep learning and builds a graph representation to capture the complexity of logs produced by many different related microservices to perform anomaly detection. As a result, DeepTraLog reports high precision and recall measures, with an average increase of 0.37 in F-measure over existing approaches. In [21], the authors perform an extensive empirical study of several state-of-the-art deep learning models designed for anomaly detection in software systems. They use different public log datasets collected from actual deployments of distributed computing systems such as Hadoop on Amazon EC2, the Blue Gene/L supercomputer, etc. They found that aspects such as training data selection and different characteristics of the datasets have significant impacts on the results. More importantly, the performance of these studied models are usually not as good as expected; so log-based anomaly detection is still an open problem.

B. Anomaly detection and defense approaches for drones

Drones, especially industrial and military grade drones, records flight data at runtime in the form of logs. Such logs might contain entries for timestamps, sensor readings and other statuses. Physical instabilities could be detected using these runtime log data, if a given baseline, i.e., normal flight data, could be obtained. This problem is similar to log differencing in software systems [22]. However, not much work has been done on log-based anomaly detection in the context of drones, especially when we need to perform such detection at runtime. For instance, in [23], the authors design a forensics framework to enable the examination of a drone's activities via its log after the flight. The work presented in [24] analyzes positioning data in drones made by three different vendors, namely DJI, Parrot and Yuneec, to carry out flight path reconstructions. In [25], the authors implement a digital forensic method for analyzing drone data based on self-organizing maps, and evaluate it with ArduPilot DIY Drone and DJI Phantom 4 images. These existing work have mostly dealt with after-the-fact drone forensics analyses, e.g., to find evidence for the court of law, etc. To the best of our knowledge, no work has been done regarding log-based anomaly detection in drones during runtime.

To detect bugs in drone control programs, several drone fuzzing approaches have been proposed, in particular RV-Fuzz [3], PGFuzz [4], and LGDFuzzer [5]. RVFuzz deals with input validation bugs affecting drone's control parameter inputs. Those bugs can be exploited easily via normal ground control commands, leading to physical disruptions of drones. LGDFuzzer aims to detect incorrect configurations of drone control parameters which can be set by normal users, or by attackers. It leverages machine learning, fuzzing, genetic search and multi-objective optimization to identify wrong configurations and suggest feasible ranges of control parameters. PGFuzz is a fuzzing approach based on predetermined safety and functional policies. Such policies are expressed in temporal logic formulas, which include user commands, configurations, and drone physical states. Fuzzing inputs are then generated considering a distance measuring how close the state of the drone to a policy violation. PGFuzz has been validated on popular drone control programs such as ArduPilot and PX4. We note that the above mentioned fuzzing approaches could be effective at finding input validation and semantic bugs in drone control programs, but it is not possible for them to detect all the potential bugs. Therefore, fuzzing approaches are useful, but by themselves they cannot handle unstable physical states in drones when undetected bugs are activated at runtime.

Protecting drones against various cyber-attacks is a growing research area [26]. Existing approaches mostly consider the system perspectives by employing network intrusion detection systems (IDS), or securing drone wireless communication, etc. For instance, [27] describes a Moving Target Defense (MTD) approach focusing on communication network security and intrusion detection. In [28], Zhang et al. implements algorithms at the physical network layer to secure wireless channels used in drones to ground communication in 5G networks. In [29], a rule-based IDS is employed to protect drone systems against various cyber-attacks. Such systems needs human experts to configure the rules used in detection, and may not work when encountering new types of attacks. [30] discusses anomaly detection techniques focusing on distributed denial of service attacks in drone networks. Our work in this paper on log-based detection of anomalies in drones during runtime can complement existing defense approaches which have been focusing more on the physical infrastructure of drone systems.

VI. Conclusion

In this work, we address the problem of detecting anomalies that could cause physical instabilities of drones. Drones experiencing physical instabilities while conducting a fight mission could cause serious safety concerns. Drones usually log flight data at runtime. Such logs typically contains flight information such as GPS, actuator outputs, accelerator readings, gyroscopic readings, together with the timestamps. We leverage the normal logs recorded by the drones to train a bidirectional LSTM-based deep learning model. This model is then used for detecting anomalies as the new log entries are recorded at runtime. We evaluated this approach based on the logs produced by three drone control programs — one closed source and two open source. We injected the logs with seven types of faults that reflect realistic scenarios such as malfunctioning sensors, high vibration levels, signal interferences, etc. In terms of accuracy, our prediction models achieved 0.968 recall and 0.963 precision. In terms of efficiency, our prediction models, when deployed on a Raspberry Pi device that can actually be used to control a drone, were able to generate a prediction within a few milliseconds on average. In future, we plan to extend this work by investigating the impact of various thresholds and varying configuration parameters. We also plan to look into other kinds of faults that were not considered in this work.

Acknowledgment

This research / project is supported by the National Research Foundation Singapore, under the National Satellite of Excellence in Mobile System Security and Cloud Security (NRF2018NCR-NSOE004-0001).

References

[1] "List of uav-related incidents," https://en.wikipedia.org/wiki/List_of_UAV-related_incidents, Accessed 2022.

[2] R. Alex, "Drone hits flying scotsman on north yorkshire moors," https://www.yorkpress.co.uk/news/ryedale/14351263.drone-hits-flying-scotsman-on-north-yorkshire-moors/, March 2016.

[3] T. Kim, C. H. Kim, J. Rhee, F. Fei, Z. Tu, G. Walkup, X. Zhang, X. Deng, and D. Xu, "Rvfuzzer: Finding input validation bugs in robotic vehicles through control-guided testing," in *28th USENIX Security Symposium (USENIX Security 19)*. USENIX Association, 2019, pp. 425–442.

[4] H. Kim, M. O. Ozmen, A. Bianchi, Z. B. Celik, and D. Xu, "Pgfuzz: Policy-guided fuzzing for robotic vehicles," in *Network and Distributed Systems Security (NDSS) Symposium*, 2021.

[5] R. Han, C. Yang, S. Ma, J. Ma, C. Sun, J. Li, and E. Bertino, "Control parameters considered harmful: Detecting range specification bugs in drone configuration modules via learning-guided search," in *International Conference on Software Engineering (ICSE 22)*, 2022.

[6] Y. Son, H. Shin, D. Kim, Y. Park, J. Noh, K. Choi, J. Choi, and Y. Kim, "Rocking drones with intentional sound noise on gyroscopic sensors," in *24th USENIX Security Symposium (USENIX Security 15)*, 2015, pp. 881–896.

[7] T. N. Y. Times, "F.A.A. opens inquiry after baby hurt in drone crash," https://www.nytimes.com/2015/09/23/business/drone-crash-injures-baby-highlighting-faa-concerns.html, 2015.

[8] S. Shankland, "Facebook drone investigation: Wind gust led to broken wing," https://www.cnet.com/tech/services-and-software/facebook-drone-investigation-wind-gust-led-to-broken-wing, 2016.

[9] W. Meng, Y. Liu, Y. Zhu, S. Zhang, D. Pei, Y. Liu, Y. Chen, R. Zhang, S. Tao, P. Sun *et al.*, "Loganomaly: Unsupervised detection of sequential and quantitative anomalies in unstructured logs." in *IJCAI*, vol. 19, no. 7, 2019, pp. 4739–4745.

[10] M. Du, F. Li, G. Zheng, and V. Srikumar, "Deeplog: Anomaly detection and diagnosis from system logs through deep learning," in *Proceedings of the 2017 ACM SIGSAC conference on computer and communications security*, 2017, pp. 1285–1298.

[11] X. Zhang, Y. Xu, Q. Lin, B. Qiao, H. Zhang, Y. Dang, C. Xie, X. Yang, Q. Cheng, Z. Li *et al.*, "Robust log-based anomaly detection on unstable log data," in *27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2019, pp. 807–817.

[12] D. Slotta, "Leading global drone manufacturers 2021, by share of sales volume," https://www.statista.com/statistics/1254982/global-market-share-of-drone-manufacturers/, June 2022.

[13] W. Minn, "Dronlomaly: Runtime detecting of anomalous drone behaviors," https://github.com/weiminn/DronLomaly, 2022.

[14] "Dedrone: Worldwide drone incidents," https://www.dedrone.com/resources/incidents/all, Accessed 2022.

[15] "Drone incidents: A survey of legal cases," https://dronecenter.bard.edu/drone-incidents/, Accessed 2022.
[16] A. Team, "Sparkfun autonomous vehicle competition 2013," https://avc.sparkfun.com/2013, Accessed 2022.
[17] "Gazebo simulation," https://docs.px4.io/main/en/simulation/gazebo.html, 2022.
[18] "A quick trick to create random lat/long coordinates in python (within a defined polygon)," https://medium.com/the-data-journal/a-quick-trick-to-create-random-lat-long-coordinates-in-python-within-a-defined-polygon-e8997f05123a, 2021.
[19] "Dji onboard sdk (osdk) 4.1.0," https://github.com/dji-sdk/Onboard-SDK, Accessed 2022.
[20] C. Zhang, X. Peng, C. Sha, K. Zhang, Z. Fu, X. Wu, Q. Lin, and D. Zhang, "Deeptralog: Trace-log combined microservice anomaly detection through graph-based deep learning," 2022.
[21] H. Van Le and H. Zhang, "Log-based anomaly detection with deep learning: How far are we?" in *In 2022 IEEE/ACM 44th International Conference on Software Engineering (ICSE)*, 2022.
[22] L. Bao, N. Busany, D. Lo, and S. Maoz, "Statistical log differencing," in *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2019, pp. 851–862.
[23] A. L. S. Renduchintala, A. Albehadili, and A. Y. Javaid, "Drone forensics: digital flight log examination framework for micro drones," in *2017 International Conference on Computational Science and Computational Intelligence (CSCI)*. IEEE, 2017, pp. 91–96.
[24] R. Kumar and A. K. Agrawal, "Drone gps data analysis for flight path reconstruction: A study on dji, parrot & yuneec make drones," *Forensic Science International: Digital Investigation*, vol. 38, p. 301182, 2021.
[25] S. H. Mekala and Z. Baig, "Digital forensics for drone data–intelligent clustering using self organising maps," in *International Conference on Future Network Systems and Security*. Springer, 2019, pp. 172–189.
[26] J.-P. Yaacoub, H. Noura, O. Salman, and A. Chehab, "Security analysis of drones systems: Attacks, limitations, and recommendations," *Internet of Things*, vol. 11, 2020.
[27] C. Gudla, M. S. Rana, and A. H. Sung, "Defense techniques against cyber attacks on unmanned aerial vehicles," in *Proceedings of the international conference on embedded systems, cyber-physical systems, and applications (ESCS)*, 2018, pp. 110–116.
[28] G. Zhang, Q. Wu, M. Cui, and R. Zhang, "Securing uav communications via joint trajectory and power control," *IEEE Transactions on Wireless Communications*, vol. 18, no. 2, pp. 1376–1389, 2019.
[29] R. Mitchell and R. Chen, "Adaptive intrusion detection of malicious unmanned air vehicles using behavior rule specifications," *IEEE transactions on systems, man, and cybernetics: systems*, vol. 44, no. 5, pp. 593–604, 2013.
[30] J.-P. Condomines, R. Zhang, and N. Larrieu, "Network intrusion detection system for uav ad-hoc communication: From methodology design to real test validation," *Ad Hoc Networks*, vol. 90, p. 101759, 2019.