Fuzz Testing of UAV Configurations Based on Evolutionary Algorithm

Yuexuan Ma[1], Xiao Yu[1(✉)], Yuanzhang Li[2], Li Zhang[3], Yifei Yan[1], and Yu-an Tan[2]

[1] School of Computer Science and Technology, Shandong University of Technology, Zibo 255049, Shandong, China
yuxiao8907118@163.com

[2] School of Computer Science and Technology, Beijing Institute of Technology, Beijing 100081, China

[3] Department of Media Engineering, Communication University of Zhejiang, Hangzhou 310018, Zhejiang, China

Abstract. With the widespread application of Unmanned Aerial Vehicle (UAV) technology, its security issues have also attracted much attention, among which the configuration attack against the UAV flight control system is one of the current research hotspots. Attackers always upload seemingly normal configuration combinations and cause an imbalance in the UAV state by exploiting configuration item verification vulnerabilities. This paper accumulates flight data through simulation, generates configuration combinations within the security range using differential evolution-based fuzz testing, uses neural networks to guide configuration item variants, and applies these configuration combinations to the AutoTest of UAV flight control systems. The experimental results show that the configuration combinations generated by fuzz testing can guide the UAV to course deviation, spin, crash and other unstable states; the code coverage and function coverage of the position and attitude code library base in the flight control system have also reached a high level.

Keywords: UAV · Configuration Security · Fuzz Testing · Differential Evolution · Neural Network · Code Coverage

1 Introduction

UAV is a type of aircraft that does not require direct maneuvering by personnel. Due to its excellent reconnaissance and strike capabilities, it was initially used for military purposes at first. In recent years, technological advances and cost reductions have led to the widespread use of UAVs in civil applications such as agricultural engineering, environmental monitoring, commercial performances, and aerial photography. However, as the functionality of UAV flight control systems continues to expand and the complexity of the structure continues to increase, configuration errors can accumulate throughout the system. These problems may lead to the unavailability of system functions or severe safety incidents.

J. Chen and Z. Xia (Eds.), BlockTEA 2023, LNICST 577, pp. 41–56, 2024.
https://doi.org/10.1007/978-3-031-60037-1_3

Fuzz testing is a testing technique to deal with such software security issues. By injecting a large number of unexpected test cases into the target program, fuzz testing seeks to cover all the running states of the program to discover as many configuration errors in the system as possible. It has the advantages of relatively low cost, high efficiency in finding errors, and automatic execution. Common fuzz testing tools include AFL [1], Peach, Radamsa, etc. These tools generate test cases by changing existing data samples or modeling program inputs, and adjust test cases according to program status feedback to detect program errors with loopholes. Usually, flight control system has thousands of configuration items, and the configuration combination will form a huge input space; and there is currently no unified judgment condition to determine whether the flight control system is faulty, so the above fuzz testing tools are mainly used for traditional software such as applications, and cannot be directly applied to embedded dedicated systems such as ArduPilot [2].

In response to this problem, Kim et al. used the control-guided testing method to discover the input verification vulnerability of the UAV flight control system [2]; and Han et al. proposed a fuzz testing tool LGDFuzz for the UAV flight control system [3], the test cases are generated by the genetic algorithm, and the effect of the fuzz testing is judged by the state of the drone. However, the above-mentioned fuzz testing methods for UAVs do not take the coverage rate as the guidance of the fuzz testing. Using the coverage index instead of observing the state of the UAV can evaluate the results of the fuzz testing more intuitively and accurately. Therefore, this paper proposes a fuzz testing method based on differential evolutionary algorithm to fuzz test some configuration items of ArduCopter, a flight control system under ArduPilot, and analyzes the configuration errors and security loopholes in the system using the coverage rate and the failure rate as the evaluation indexes. The main contributions of this paper are as follows:

(1) For mainstream fuzz testing tools only designed for the scope of application of general-purpose software, a fuzz testing tool adapted to the ArduCopter flight control system is proposed. Evolutionary algorithm fuzz testing for finding configuration errors in flight control systems.
(2) For the problem that the combination of fuzz testing based on the differential evolutionary algorithm and UAV flight control system leads to high computational cost, a neural network model TALA is established that integrates convolutional neural network, recurrent neural network and attention mechanism. This model predicts the state of UAV to accelerate the computation of fitness function in differential evolutionary algorithm and improve the efficiency of fuzz testing.

2 Related Works

2.1 Attack Methods Against UAVs

The widespread use of UAVs exposes them to many attacks, including physical attacks on sensors, network attacks on communication protocols, and software attacks on flight control systems. At the physical attack level, laser blinding attacks against vision sensors [4, 5] may lead to incorrect decisions by the UAV, which in turn may lead to more serious consequences; acoustic resonance attacks against UAV navigation sensors [6] can cause the UAV to lose position data and thus crash uncontrollably; spoofing attacks

against GPS [7, 8] can force UAVs to receive false navigation information, thereby inducing dangerous behavior. At the level of communication protocol security, as the MAVLink communication protocol commonly used by UAVs does not support encrypted communication and authentication authorization, the attackers can intercept and modify MAVLink information between the drone and the ground station, and easily launch man-in-the-middle attacks or replay attacks. Kwon et al. [9] performed ICMP flooding attack and malicious packet injection attack on UAVs by analyzing the MAVLink protocol vulnerability, causing the mission UAVs to stop and hover due to deleting mission information. At the software security level, the lack of validation of configuration item inputs by the flight control system makes it possible for seemingly normal inputs to cause anomalies in the UAV, i.e., an input validation vulnerability exists. RVFuzzer [2] proposed by Kim et al. looks for input validation errors in the flight control system through control-guided input variants, and LGDFuzz [3] proposed by Han et al. similarly exploits input validation vulnerabilities to perform configuration attacks on UAVs. The work in this paper is also developed based on this classical vulnerability to attack the UAV flight control system by mutating the configuration items of the UAV flight control system within the legitimate range.

2.2 Fuzz Testing for UAVs

As an embedded special system, fuzz testing for UAVs has difficulties such as complex data interaction, high environmental dependency, and strict safety and reliability requirements. As a result, general-purpose fuzz testing tools that cannot be directly used for UAV flight control system. However, their seed mutation methods, seed selection and other ideas can still be used as a reference. As one of the most representative fuzz testing tools, AFL [1] and its improved versions are mutation-based file fuzz testing tools, which generate test cases through genetic algorithms and track the execution path of the code under test by using instrumentation technology. Record the code coverage of the input samples, and use them as feedback to mutate the input samples to improve the coverage, thereby enhancing the possibility of finding vulnerabilities. Honggfuzz [10] is also a mutation-based fuzz testing tool that generates test cases and detects vulnerabilities in target programs through dynamic binary instrumentation and automated symbol execution techniques. Peach is a model-based file fuzz testing tool that automatically generates random test cases that meet the target program's expectations according to the input data's format and structure and records the coverage results for analysis. This method can ensure test cases' passability and improve fuzz testing's efficiency and accuracy. LibFuzzer [11] is a memory-based fuzz testing tool that uses instrumentation technology to track code coverage. By establishing each fuzz testing in the same process, the injection of the test case does not need to wait for the restart of the target program, which greatly improves the efficiency of the fuzz testing.

The above general fuzz testing tools' design ideas and implementation schemes have laid the foundation for applying fuzz testing on UAVs. RVFuzzer [8] uses control-guided input mutators to generate test cases and uses the detection results of the UAV's state as feedback to fuzz testing the flight control system to detect input validation vulnerabilities. LGDFuzz [9] uses genetic algorithm-based fuzz testing to automatically

detect errors in UAV configuration items for range specification errors and uses multi-objective optimization to calculate the most suitable parameter value range for each configuration item. PGFuzz [12] proposes a policy-oriented fuzz testing framework for robotic vehicles, which expresses temporal logic formulas with time constraints, and uses this constraint as a guiding metric for fuzzing. In this paper, ArduCopter, a multi-rotor UAV flight control system under ArduPilot, is fuzzed for its configuration items using the differential evolutionary algorithm, and the coverage rate and failure rate are used as evaluation metrics for fuzz testing.

2.3 Neural Networks for Fuzz Testing

Traditional fuzz testing usually uses randomly generated test cases to test software programs. This method can cover a relatively complete test space, but it is difficult to guarantee the efficiency and accuracy of the test. With the rise of machine learning, fuzz testing technology integrated with deep learning has gained a wide range of development space. NEUZZ [13] uses a feed-forward neural network to learn a smooth approximation of the branching behavior of the target program to generate effective test cases; and uses a gradient-guided search strategy to guide mutation positions to maximize the detection of errors in the target program. Faster Fuzzing [14] uses Generative Adversarial Networks (GAN) as AFL's seed reinitialization strategy, which can find more unique code paths than LSTM or random enhancement strategies, and enhances the fuzzing effect of AFL. SmartSeed [15] proposed by Lyu et al. uses Wasserstein Generative Adversarial Networks (WGAN) to generate valuable seed files, which are used as the input of AFL for fuzz testing. LGDFuzz [9] uses LSTM to calculate the fitness function in the fuzz testing based on the genetic algorithm to estimate the state of the UAV. In this paper, a neural network model combining TCN, LSTM and Attention is established in the evolutionary algorithm, which can predict the state of the drone with a lower error and accelerate the fitness function calculation.

3 UAV Fuzz Testing Framework

3.1 UAV Fuzz Testing Overall Framework

This paper proposes a fuzz testing method based on coverage and failure rate evaluation. First, configure the UAV mission file and select the configuration items to participate in the fuzz testing. Then, the flight log of the UAV is collected by the emulator in the way of simulated flight, and the characteristic data of the log are extracted to construct a data set for the neural network and fuzz testing. The differential evolutionary algorithm-based fuzz testing uses a neural network to calculate fitness function and generates many deformed configuration items using a mutation data set. Finally, the results of fuzz testing are applied to simulation verification, and the coverage and failure rates of specific components are calculated to evaluate the performance of fuzz testing and guide the selection of configuration items. The overall framework of the UAV fuzz testing method is shown in Fig. 1.

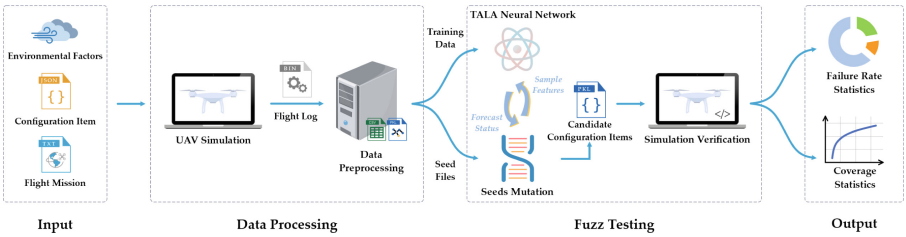

Fig. 1. UAV fuzz testing overall framework.

3.2 UAV Simulation

Operating Platform and Flight Mission. Both the acquisition of UAV flight logs and the verification of configuration items need to be carried out on the flight control system. In this paper, SITL simulator with lower cost but higher efficiency and safety is selected to simulate and run UAVs. Different configuration items are randomly created within the ArduPilot published configuration item safety range [16], and the AVC2013 [17], FitCollection, Copter-Mission, and SDUT-Map flight missions are repeatedly executed with these configuration item groups to collect flight logs. Among them, AVC2013 is the flight route used in SparkFun's fifth Autonomous Vehicle Competition. The UAV needs to take off independently, pass through the designated waypoint and land in the landing area to earn points; FitCollection is the flight route used by LGDFuzz [9], which is an improved version of AVC2013; Copter-Mission is the flight route used by the ArduPilot autotest framework, taking off from the airport of the Canberra Model Aircraft Club and returning to the airport after a simple flight. All of the above are public flight routes, while SDUT-Map belongs to this paper's own planned flight routes, aiming to evaluate UAV fuzz testing's adaptability to complex routes. During log collection process, abnormal logs are eliminated through data consistency inspection to prevent the training data set of the neural network from being polluted.

Configuration Item Selection. ArduPilot achieves specific functions by decoupling different system libraries, including core library, navigation library, control library, sensor library and motor library, etc. Different configuration items control various libraries. Due to the large number of configuration items involved in the system library and the lack of correlation, this article only selects the configuration items that affect the flight status of UAVs under the automatic flight state, while the configuration items of non-closely related components such as remote control and camera are not considered in the fuzz testing. ArduPilot mainly uses the position controller and attitude controller to control the flight state of the UAV, and the related configuration items are located primarily in the AC_AttitudeControl library. In the position controller, the horizontal position controller converts the position error into a target horizontal velocity, and the velocity controller converts the velocity error into a desired acceleration, which is then converted into a desired tilt angle and sent to the attitude controller. In the position controller, the horizontal position controller converts the position error into a target horizontal velocity, and the velocity controller converts the velocity error into a desired acceleration, which is then converted into a desired tilt angle and sent to the attitude controller. The vertical

position controller converts the position error into a target vertical velocity, the velocity controller converts the velocity error into a desired acceleration, and the acceleration PID controller converts the acceleration error into a desired throttle, which is then sent to the attitude controller. The position controller is controlled by three configuration items of speed loop, position loop and acceleration loop. In this paper, 14 configuration items involving PID gain and feedforward gain in each loop are selected. In the attitude controller, the AP controller converts the error between the target angle and the actual angle into a desired rotation rate. Then the attitude PID controller converts the rotation rate error into high-level motor commands to control the state of the UAV. The attitude controller is controlled by the configuration items of the angular rate loop and the angle loop. In this paper, 23 configuration items involving PID gain, feedforward gain and limiter are selected in each loop, which have the most direct impact on the position and attitude of the UAV in AUTO flight mode.

3.3 Data Preprocessing

The UAV log data needs to be preprocessed to improve the neural network's performance and the effect of fuzz testing. After collecting enough flight logs, extract three angular states (Roll Angle, Pitch Angle, Yaw Angle), three angular rates (Roll Angle Rate, Pitch Angle Rate, Yaw Angle Rate), 12 sensor data (Three-axis Accelerometer Original Value, Three-axis Gyroscope Original Rotation Rate, Three-axis Compass Original Magnetic Field Value, Three-axis Main Accelerometer Output Standard Deviation) and 42 configuration items, merged into a log dataset in CSV format, and create a copy of the dataset. A normalization operation is applied to the features of one of the datasets so that different features have the same scale to optimize the speed of gradient descent to find the optimal solution. Finally, the data set is transformed into a supervised learning problem. The training set and test set are divided for subsequent neural network training. After sampling another data set, it is directly converted into a supervised learning data set, which is used to build a seed pool for fuzz testing.

3.4 Fuzz Testing

General Framework of Fuzz Testing. Fuzz testing is a method of discovering vulnerabilities by providing many malformed inputs to the target system and observing the abnormal results exhibited by the system. Relatively low cost, high efficiency of configuration error mining, and automatic execution make fuzz testing a common method for software black-box, white-box or gray-box testing.

According to the acquisition method of test samples, fuzz testing can be divided into the mutation mode based on changing the existing data samples and the generation method based on program input modeling. However, the traditional method of randomly generating test samples has a poor purpose, leading to low program testing efficiency. In the fuzzification of configuration items, this paper uses the Different Evolution Algorithm (DE) [18, 19] to mutate the configuration items, which is a heuristic optimization algorithm based on the genetic algorithm, and its process is shown in Algorithm 1. Firstly, the default value of each configuration item is used as the initial population, and each

individual in it corresponds to a group of configuration items. Then calculate the fitness of individuals in the population, perform evolutionary operations of selection, mutation, and crossover on individuals, and continue to iterate until the termination condition. The last generation population is used as the target configuration item population to construct the output of the fuzz testing.

Algorithm 1: Differential evolution algorithm.

Input: $P(G)$, MAX

Output: $P(G_{MAX})$

1: **while** $G < MAX$ **do**
2: $f(P(G)) \leftarrow Fitness(P(G))$
3: $P(G) \leftarrow$ Selection($f(P(G))$, $P(G)$)
4: $P(G) \leftarrow$ Mutations($P(G)$)
5: $P(G) \leftarrow$ Crossover($P(G)$)
6: $f(P(G)) \leftarrow Fitness(P(G))$
7: $P(G+1) \leftarrow$ Environmental Selection($f(P(G))$, $P(G)$)
8: **end while**

Population Initialization. The differential evolution algorithm constructs an initial candidate solution population $P(0)$, which contains a number of individuals $X_{i,j}$ composed of chromosomes, each individual obeys the random deviation of the standard normal distribution plus the uniform distribution between the lower bounds, and each chromosome in the individual All are candidate solutions for the current problem. The populations $P(0)$ and individuals $X_{i,j}$ are shown below:

$$\begin{cases} \left\{ P(0) | x_{i,j}^{(L)} \le x_{i,j}(0) \le x_{i,j}^{(U)}; i = 1, 2, \cdots, MAX; j = 1, 2, \cdots, D \right\} \\ x_{i,j}(0) = RND[0, 1] * \left(x_{i,j}^{(U)} - x_{i,j}^{(L)} \right) + x_{i,j}^{(L)} \end{cases} \tag{1}$$

where i is the individual number, j represents the j th dimension; $x_{i,j}^{(L)}$ and $x_{i,j}^{(U)}$ represent the lower and upper bounds of the i th individual in the j dimension; MAX represents the population size; D represents the independent variable dimension; $RND[0, 1]$ represents a random number in the interval [0, 1]. The defined chromosome's objective function value in the candidate solution population is the configuration item's default value.

Fitness Function. The fitness function is an important evaluation index for the survival of the fittest, which directly affects the convergence speed of the evolutionary algorithm and whether it can find the optimal solution. The fitness function can be expressed as $Fitness = |PredictiveValue - TrueValue|$, which is the absolute value of the deviation between the predicted and true values of the three angular states (Roll, Pitch, Yaw) and three angular rates (RateRoll, RatePitch, RateYaw) of the UAV. In this paper, the neural network is used to calculate the predicted value of the UAV state, and the specific information is shown in Sect. 3.5. The larger the fitness function is, the more likely this group of configurations causes the possible state of the UAV to deviate from the real state, and the more likely it is an abnormal configuration item group, so it is more likely to be saved as the optimal individual in the next generation population.

Evolutionary Operations. Evolutionary operations include selection, mutation, crossover, and environmental selection.

Selection. The selection operator adopts the elite selection method, directly selecting a certain number of individuals with high fitness and directly participating in the subsequent evolutionary operation without modification. In contrast, the unselected individuals are directly discarded and will not enter the evolutionary iteration.

Mutations. The difference vector difference is scaled after different individuals are obtained by selection. Then, the vector is fused with the individual to be mutated to obtain a new mutated individual. Aggressive mutation probabilities can lead to many invalid inputs, while more conservative mutation probabilities can avoid falling into local optimal solutions. The operation of differential mutation is as follows:

$$V_{i,g} = x_{i,g} + F * \left[\left(x_{r1,g} - x_{r2,g}\right) + \left(x_{best,g} - x_{i,g}\right)\right] \tag{2}$$

where $V_{i,g}$ is the mutant individual, g represents the evolution algebra, and F is the variation scaling factor, which is used to control the scaling of the difference vector; $r1$ and $r2$ are unequal random numbers in the interval $[0, NP]$, $x_{r1,g}$ is a randomly selected individual in the population, $x_{r2,g}$ is the randomly selected individual in the current population and the external archive collection, and $x_{r1,g} - x_{r2,g}$ constitutes the difference vector; $x_{best,g}$ is the best individual in the current population, $x_{i,g}$ is the parent individual. After the differential mutation operation, each parent individual can generate a mutant individual.

Crossover. To further increase the diversity of individuals, the binomial distribution crossover method crosses the mutant individuals with the parent individuals to generate new test individuals. The operation of the binomial crossover is as follows:

$$U_{i,j}(g) = \begin{cases} V_{i,j}(g); \textit{if } \textit{rand}(0,1) \leq \textit{XVOR} \\ x_{i,j}(g); \textit{if } \textit{rand}(0,1) > \textit{XVOR} \end{cases} \tag{3}$$

where $U_{i,j}(g)$ is a new test individual, and $V_{i,j}(g)$ is the j th parameter value of the i th configuration in the next-generation individual after crossover, $x_{ij}(g)$ is the j th parameter value of the i th configuration in the g th generation individual before crossover, and *XVOR* is the crossover probability.

Environmental Selection. One-to-One Survivor Selection (OTOS) is performed between the test individuals and the parent individuals in index order, and the next generation retains the elite individuals with higher fitness to generate a new population generation. The operation of the environmental selection is shown below:

$$X_i(g+1) = \begin{cases} U_i(g); \textit{if } \textit{Fitness}(U_i(g)) \leq \textit{Fitness}(X_i(g)) \\ X_i(g); \textit{if } \textit{Fitness}(U_i(g)) > \textit{Fitness}(X_i(g)) \end{cases} \tag{4}$$

where $X_i(g+1)$ is the selected next-generation individual, $U_i(g)$ is the test individual, $X_i(g)$ is the parent individual, and *Fitness* is the fitness function.

Generating New Generation Populations. If the maximum evolutionary generation or the convergence condition of the algorithm is not reached, the generated new generation

population will continue to the next round of evolution; otherwise, some individuals with the highest fitness will be selected from the last generation population to construct the configuration population as the test case output by the fuzz testing.

3.5 UAV Status Prediction

The ArduPilot flight control system collects sensor data such as Inertial Measurement Unit (IMU), GPS, and Compass and uses the Extended Kalman Filter (EKF) to estimate information such as the position, attitude, and velocity of the UAV. In the fitness function of the evolutionary algorithm, to efficiently calculate the predicted value of each state of the UAV, we use a TCN-Attention-LSTM-Attention (TALA) neural network to estimate the angular state and angular rate of the UAV. The input layer of the model consists of 3 angular states of the UAV, three angular rates, 12 sensor data, and 42 configuration items for the first four-time steps. In the hidden layer, the TCN layer extracts the UAV state and configuration item features in the time dimension and updates the weights of different feature information by the first layer of Attention. The LSTM prediction layer further extracts time series features and then updates the different features by the second layer of Attention. After paying attention to the information, the output layer outputs the maximum conditional probability prediction of the UAV's three angular states and three angular rates in the next time step. The overall framework of the neural network is shown in Fig. 2.

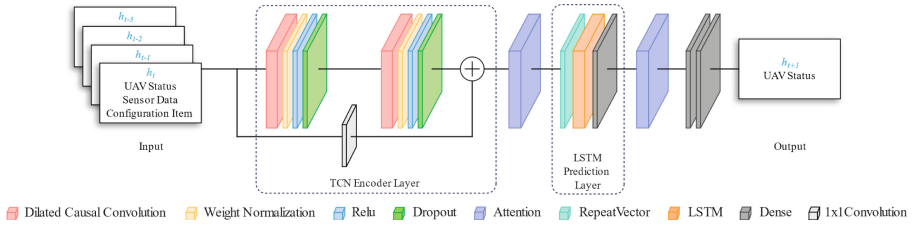

Fig. 2. The general framework of neural networks.

The Temporal Convolution Network (TCN) [20] uses Casual Dilated Convolutions to model the time series of UAV angular states, angular rates, sensor data, and configuration item information. In which causal convolution only performs one-way processing of past and current moment data and does not introduce future information, avoiding the traditional CNN data leakage problem; the dilated convolution can flexibly adjust the receptive field of the neural network, so that the network can handle the long-term dependence characteristics of the flight log sequence. The weights of the feature information affecting the UAV state change dynamically in different time periods, and the Attention mechanism [21] enables the neural network to assign higher weights to the focused information related to the output UAV state in the input UAV state, sensor information and configurations, thus increasing the network's attention to the focused information and reducing the computational burden of the network. LSTM uses input gate, forget

gate and output gate mechanisms to deal with long-term dependencies in sequences by selectively forgetting and updating past information.

The neural network model that mixes TCN, LSTM and Attention can more comprehensively capture the long-term and short-term dependencies in the time series, and focus on important features related to the UAV state in the sequence, thereby improving the accuracy and stability of time series modeling.

3.6 Simulation Verification

Simulation Validation. Although each configuration item constructed by the fuzz testing is within the range specified by ArduPilot, it may be very deformed data after combination, which will cause various abnormal states of the UAV. We use simulation to verify whether these configuration items will cause different abnormal UAV states. The possible exception states of the UAV and their meanings are described in Sect. 3.6. As a flight control system, ArduPilot has error reporting and protection programs for abnormalities in the UAV. Unexpected configuration items can explore more edge codes and achieve higher code coverage.

The simulation verification module first uploads the configuration item groups constructed by the fuzz testing to the SITL simulation environment, uses the sim_vehicle script to start ArduCopter and cyclically verifies each group of configuration items, and checks the abnormal status of the UAV by listening to the MAVLink message sent by the UAV. This test method needs to restart the SITL simulator to verify the next configuration items, which is less efficient for testing. This script only repeats the same flight mission with a new set of configuration items, covering fewer test categories, so it is only used to count the abnormal state of the UAV during the flight. It should not be used for coverage detection. After verifying the state of the UAV caused by each set of configuration items, the configuration item group that enables the UAV to perform tasks normally is eliminated and then passed to the AutoTest suite. The AutoTest suite creates repeatable unit tests and functional tests for ArduCopter. This testing method reduces the time spent on running the same scenario repeatedly. Testing is more efficient and more categories are covered. Under the premise of using the same configuration items, we combine the two configuration item verification methods of sim_vehicle and AutoTest to comprehensively calculate the code coverage, function coverage and failure rate of the ArduPilot related code base.

Abnormal State of the UAV. Unexpected configuration items may lead to abnormal states of the UAV, and the quality of test cases generated by fuzz testing can be measured by monitoring these abnormal states. High-quality test cases can guide UAVs to appear in various abnormal states.

The abnormal state of the UAV can be analyzed by downloading log files or monitoring the communication data between the UAV and the ground station. However, the log files of the UAV usually need to be downloaded after a complete flight, which has great limitations in time. By monitoring the MavLink communication protocol and analyzing the heartbeat packets transmitted by the UAV to the ground station in real time, the abnormal state of some UAVs can be judged immediately. The state categories of UAVs are shown in Table 1.

Table 1. Status categories of UAVs.

UAV Status	Implications
Pass	If the UAV can complete the scheduled flight mission with a relatively normal attitude, this flight mission can be defined as Pass
Timeout	Incorrect configuration items may cause the UAV to hover or stay in a certain position. When not idling on the ground or hovering in the air, if the moving distance within a certain period is less than the threshold, it will be considered as Timeout
PreArm Failed	Flight control system will refuse to unlock the motors when it detects sensor abnormalities in sensors or some configurations before takeoff
Potential Thrust Loss	When certain configurations are wrong, the load is too large, or the ambient wind speed is too high, even if the throttle is saturated, the UAV cannot reach the desired state
Course Deviation	Incorrect configuration items may cause the UAV in Auto mode to deviate from the established route
Yaw Imbalance	The higher the yaw imbalance, the weaker the UAV's control over the yaw action, and the more likely it is to fall into a state of swinging left and right in the yaw direction. Conversely, UAVs have more room for stability control
Spin	The UAV will fall into a fast spin when the yaw imbalance exceeds a certain threshold. This abnormal state usually cannot be controlled or stopped
Crash	Abnormal conditions such as excessive landing speeds, consistently exceeding angular limits, loss of thrust, yaw imbalance, and spin can all lead to a collision or crash of the UAV

3.7 Result Statistics

Failure discovery rate refers to the ratio of the number of test cases that can trigger vulnerabilities and errors to the total number of test cases. It is one of the important indicators to measure the effect of fuzz testing. We counted the number and proportion of the test cases generated by the fuzz testing that can cause the abnormal state of the UAV described in Sect. 3.6 to judge the quality of the test cases.

Code coverage and function coverage are common metrics for fuzz testing methods. The higher the coverage, the more comprehensive the testing of the program, the more likely to find defects in the code. We use GCOV as a statistical tool for code coverage. When compiling the source code of the target program, we insert probes into the entry and exit of the basic block to determine whether the basic block is covered. After running enough tests, generate a source file of coverage statistics in GCOV format and then use LCOV to convert it into a visual coverage report in HTML format. The coverage report reflects the code coverage and function coverage of specific files. It even includes the execution times of each line of code, which can intuitively evaluate the effect of fuzz testing.

4 Experiments

4.1 Experiment Environment

The hardware environment of this experiment is Intel i7-12700KF CPU, NVIDIA RTX 3080 GPU, 64 GB RAM; the software environment is Ubuntu 22.04 operating system, ArduPilot V4.3.7 open source autopilot software, Python 3.10.6 development language, TensorFlow 2.12.0 deep learning framework.

4.2 Experimental Evaluation Index

Neural Network Evaluation Index. The deviation between true and predicted values usually evaluates the regression problem. This paper uses Mean Square Error (MSE) as the loss function of the neural network. The formula for MSE is shown below:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}\left(Y_i - \widehat{Y}_i\right)^2 \tag{5}$$

where n is the size of the neural network batch, Y_i is the true value of the i th sample, and $\widehat{Y}_i$ is the predicted value of the i th sample. The smaller the value of MSE means the smaller the error between the true value and the predicted value, the better the model is fitted.

Fuzz Testing Evaluation Index. The failure discovery rate is an important evaluation index of fuzz testing, which is used to measure the ratio of the test cases generated by fuzz testing that can trigger the failure or crash of the UAV flight control system. The higher the failure discovery rate, the more effective the fuzz testing is in discovering the security holes or defects in the program. When using the configuration items constructed by the fuzz testing, according to the state of the drone defined in Sect. 3.6, record the results caused by each group of configuration items, and count the ratio of the configuration items that cause failures to the total test cases.

Coverage is the most important evaluation index of fuzz testing. Tests driven by coverage can quantify the progress of experiments and evaluate the pros and cons of fuzz testing methods. Code coverage includes indicators such as statement coverage, decision coverage, and branch coverage. We use statement coverage as the criterion for code coverage, that is, to detect whether all executable statements in the source code are executed. However, it is impossible to determine whether all logic functions have been tested with the help of code coverage alone, and the combination of function coverage can provide more accurate evaluation indicators for fuzz testing. The function coverage rate only counts the functions called. It ignores the statistics of the internal code of the function, which is used to detect whether the fuzz testing covers the functional characteristics required by the design specification. In the ArduPilot, the library that is strongly related to the flight state of the multi-rotor UAV is AC AttitudeControl, which includes the attitude and position control for the UAV. By adjusting the types and values of configuration items, the kinds of flight tasks, and the environment, we seek to improve the code coverage and function coverage of relevant source files in this code base to evaluate the adequacy of the fuzz testing method.

4.3 Analysis of Experimental Results

Analysis of Neural Network Results. The experiments compare the fitting ability of the LSTM and TCN used in LGDFuzz [3] and the TALA network model proposed in this paper to the UAV state under different flight missions. The specific performance difference results are shown in Table 2.

Table 2. Comparison of MSE of different models on test sets for different flight missions.

Flight Missions	MSE		
	LSTM	TCN	TALA
AVC2013	2.6307e−04	2.3051e−04	1.6591e−04
FitCollection	2.8905e−04	1.9475e−04	1.7528e−04
Copter-Mission	5.0408e−04	3.0376e−04	2.7091e−04
SDUT-Map	3.5327e−04	2.8599e−04	2.3811e−04

It can be seen from the table that the TALA neural network model has the smallest error value on the test set, which has certain advantages over other models. Specifically, in the AVC2013 mission, the MSE of TALA is reduced by 0.9716e−04 compared with the LSTM model; compared with the TCN model, the MSE is reduced by 0.6460e−04. For the SDUT-Map task with more complex flight paths, the MSE of TALA is reduced by 1.1516e−04 compared with the LSTM model; compared with the TCN model, the MSE is reduced by 0.4788e−04. The above results show that the TALA neural network model is effective for predicting UAV flight status, and the prediction accuracy is higher than other models. The TALA neural network can also maintain good generalization with lower error for different data sets.

Analysis of Fuzz Testing Results. For the failure discovery rate, we count the number and proportion of the UAV flight control system failures caused by the test cases generated by 4000 sets of fuzz testing in the sim_vehicle verification phase. The specific experimental results are shown in Table 3.

Table 3. Failure discovery rate.

UAV Status	Frequency	Proportion	Severity
Pass	361	9.0%	6
Timeout	814	20.4%	2
PreArm Failed	480	12.0%	0
Potential Thrust Loss	997	24.9%	0
Course Deviation	416	10.4%	4
Yaw Imbalance	367	9.2%	0
Spin	44	1.1%	0
Crash	521	13.0%	0

This table shows that among the test cases generated by the fuzz testing, the configuration items that can trigger the failure or crash of the UAV flight control system account for as high as 91%, and the proportion that can cause abnormal states after the UAV takes off reaches 88%. Only 361 groups of false positive configuration items can make the UAV complete the flight mission normally. The statistical results of the failure discovery rate fully prove the effectiveness of fuzz testing and provide support for the AutoTest verification phase.

In terms of coverage, we count the code coverage and function coverage of the AC_AttitudeControl.cpp, AC_AttitudeControl_Multi.cpp and AC_PosControl.cpp files in the AC_AttitudeControl pose control library. They represent the coverage levels of Universal Attitude Controller, Multi-rotor UAV Attitude Controller and Position Controller respectively. The comparison between our experimental results and the official results of ArduPilot is shown in Table 4.

Table 4. Coverage statistics for different files.

Controller Name	Code Coverage		Function Coverage	
	ArduPilot	Our Approach	ArduPilot	Our Approach
AC_AttitudeControl.cpp	75.8%	81.3%	89.2%	86.5%
AC_AttitudeControl_Multi.cpp	94.0%	97.7%	100%	100%
AC_PosControl.cpp	90.6%	91.3%	90.6%	90.6%

Among them, the code coverage and function coverage of the Universal Attitude Controller have reached more than 80%. Compared with the official coverage provided by ArduPilot [22], the code coverage has increased by 5.1%, but the function coverage has a difference of 2.7%. The code coverage rate of the human-machine attitude controller has risen by 3.7%, and the function coverage rate has reached 100%; the code coverage rate of the position controller has increased by 0.7%, and the function coverage rate is the same as the official one.

For files that do not reach 100% coverage first, to ensure that the ArduPilot can completely accept the test cases generated by the fuzz testing, all generated configuration items are within the security range officially defined by ArduPilot. This also resulted in some out-of-bounds checks and processing-related code not being executed. Second, part of the code is not subject to the configuration items involved in fuzz testing in this paper, so the relevant code lines are not covered. Finally, some redundant or unreachable codes are in the system but still participate in the coverage statistics. For the libraries that are not included in the coverage statistics, most of them are not related to the position and attitude of the drone; while the files that are not included in the coverage statistics in the library are mostly because the models are not related or the flight modes are not related. Complete code coverage testing is very difficult to achieve, and special test cases need to be manually designed, which cannot be achieved by fuzz testing some configuration items, and specific problems need to be analyzed in detail.

5 Conclusions

The tool performs fuzz testing based on the differential evolution algorithm for configuration items that affect the position and attitude of the UAV and performs coverage statistics on related code libraries to guide the selection of configuration items. In calculating the fitness function of the differential evolution algorithm, the TALA neural network is used to calculate the estimated value of the UAV state. The experimental results show that the TALA neural network can better extract the features of the flight logs and predict the state of the UAV with less error. Test cases generated by fuzz testing based on differential evolutionary algorithms are able to guide the UAV to seven abnormal states, in the verification stage. The coverage rate of the code library related to the position and attitude control of the UAV can exceed the data provided by the official ArduPilot, which proves the effectiveness of the method.

Acknowledgment. This study was supported by the National Key Research and Development Program of China (2020YFB1005704).

References

1. American Fuzzy Lop. https://lcamtuf.coredump.cx/afl/. Accessed 16 Dec 2022
2. Kim, T., et al.: RVFuzzer: finding input validation bugs in robotic vehicles through control-guided testing. In: 28th USENIX Security Symposium (USENIX Security 19). USENIX Association, Santa Clara, CA, pp. 425–442 (2019)
3. Han, R., et al.: Control parameters considered harmful: detecting range specification bugs in drone configuration modules via learning-guided search. In: 2022 IEEE/ACM 44th International Conference on Software Engineering (ICSE), Pittsburgh, PA, USA, pp. 462–473 (2022)
4. Petit, J., Stottelaar, B., Feiri, M., Kargl, F.: Remote attacks on automated vehicles sensors: experiments on camera and LiDAR. Black Hat Europe **11**, 995 (2015)
5. Fu, Z., Zhi, Y., Ji, S., Sun, X.: Remote attacks on drones vision sensors: an empirical study. IEEE Trans. Depend. Secure Comput. **19**, 3125–3135 (2022)
6. Yunmok, S., et al.: Rocking drones with intentional sound noise on gyroscopic sensors. In: Proceedings of the 24th USENIX Conference on Security Symposium, pp. 881–896 (2015)
7. Zou, Q., Huang, S., Lin, F., Cong, M.: Detection of GPS spoofing based on UAV model estimation. In: IECON 2016 - 42nd Annual Conference of the IEEE Industrial Electronics Society, Florence, pp. 6097–6102 (2016)
8. Bethi, P., Pathipati, S., Aparna, P.: Stealthy GPS spoofing: spoofer systems, spoofing techniques and strategies. In: 2020 IEEE 17th India Council International Conference (INDICON), New Delhi, India, pp. 1–7 (2020)
9. Young-Min, K., Jaemin, Y., Byeong-Moon, C., Yongsoon, E., Kyung-Joon, P.: Empirical analysis of MAVLink protocol vulnerability for attacking unmanned aerial vehicles. IEEE Access **6**, 43203–43212 (2018)
10. Honggfuzz. https://honggfuzz.dev/. Accessed 5 Jan 2023
11. The LLVM Compiler Infrastructure. https://llvm.org/. Accessed 10 Jan 2023
12. Kim, H., Ozmen, M.O., Bianchi, A., Celik, Z.B., Xu, D.: PGFUZZ: policy-guided fuzzing for robotic vehicles. In: NDSS (2021)

13. She, D., Pei, K., Epstein, D., Yang, J., Ray, B., Jana, S.: NEUZZ: efficient fuzzing with neural program smoothing. In: 2019 IEEE Symposium on Security and Privacy (SP), pp. 803–817 (2019)
14. Nichols, N., Raugas, M., Jasper, R.J., Hilliard, N.: Faster fuzzing: reinitialization with deep neural models. arXiv, abs/1711.02807 (2017)
15. Lv, C., et al.: SmartSeed: smart seed generation for efficient fuzzing. arXiv, abs/1807.02606 (2018)
16. Complete Parameter List. https://ardupilot.org/copter/docs/parameters-Copter-stable-V4.3.7.html. Accessed 16 June 2023
17. AVC 2013. https://avc.sparkfun.com/2013. Accessed 16 June 2023
18. Storn, R., Price, K.: Differential evolution: a simple and efficient heuristic for global optimization over continuous spaces. J. Glob. Optim. **11**(4), 341–359 (1997)
19. Geatpy: The genetic and evolutionary algorithm toolbox with high performance in python. http://www.geatpy.com/. Accessed 2 Mar 2023
20. Bai, S., Kolter, J.Z., Koltun, V.: An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. arXiv preprint arXiv:1803.01271 (2018)
21. Luong, T., Pham, H., Manning, C.D.: Effective approaches to attention-based neural machine translation. In: Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, Lisbon, Portugal, pp. 1412—1421 (2015)
22. ArduPilot LCOV - code coverage report. https://firmware.ardupilot.org/coverage/. Accessed 16 June 2023