Real-world Faults and Their Injection into Autonomous Unmanned Aerial Vehicles

Haotian Chen[1], James Cyriac[1], Saurabh Jha[2], Subho S. Banerjee[2],
Zbigniew T. Kalbarczyk[1], and Ravishankar K. Iyer[1,2]
[1]Department of Electrical and Computer Engineering, University of Illinois Urbana-Champaign, Urbana, IL, USA
[2]Department of Computer Science, University of Illinois Urbana-Champaign, Urbana, IL, USA

***Abstract*—Unmanned aerial vehicles (UAVs) are being rapidly deployed in many real-world applications, such as policing, logistics, search, aerial-survey, and -filming. However, recent headline-making UAV accidents highlight the challenge of ensuring the safety of such cyber-physical systems. Hence, a methodology for the meticulous safety assessment of UAVs is a primary concern for developing such technology, directly influencing its public acceptance and adoption. In this paper, we perform (i) an assessment of field failure reports from four multi-national databases relating to safety incidents involving UAVs; and (ii) inspired by the failure study, we demonstrate the use of a simple fault injection framework, UAVFI, to demonstrate the impact of a few key-faults on the performance and failures of UAVs in photo-realistic simulated environments. We draw several conclusions. For example, we observe that a only 8.1% of real-world accidents in UAVs are caused by computer/software malfunctions. Fault models dealing with sensor and mechanical failures are key in developing reliable UAV systems. UAVFI can leverage such insight to create fault injection campaigns that cause as much as 75.9% increase in mission completion time, as well as 3.5× increase in accident incidence.**

***Index Terms*—Unmanned Aerial Vehicle, Fault Injection, Assessment, Field Data, Failure Diagnosis, Dependable Cyber-Physical System**

I. Introduction

Unmanned aerial vehicles (UAVs) are being rapidly deployed in many real-world applications, such as policing [1], package delivery, searching, surveying, and filming [2]. However, recent headline-making UAV accidents (e.g., [3]–[5]) highlight the challenge of ensuring the safety of such cyber-physical systems. Hence, a methodology for the rigorous safety assessment of UAVs is a primary concern for developing such technology, directly influencing its public acceptance and adoption. The purpose of this practical experience report is to share insight from a study that combines data from field failures of UAVs and constructs a fault injection engine, UAVFI to recreate realistic scenarios to test and better illustrate the failure characteristics of UAV systems. Studying real-world UAV failures gives us an opportunity to learn fault patterns and understand the reliability and safety challenges of UAVs.

We collected and analyzed accident records from 2015 to 2020 from four multinational databases, spanning the US, UK and Australia [6]–[9], to identify common failure modes associated with UAVs across a variety of appellations (e.g, aerial survey, filming, policing). Each of record in our dataset was curated and validated by the legally designated national safety body (e.g., NTSB in the United States). We cleaned, parsed, and normalized the failure data into a single database which can be used to jointly analyze the latest trends in UAV reliability. Compared to past work in real-world UAV failure analysis, e.g., [10], [11], which analyze 4 and 40 accidents respectively, our combined dataset has over 160 accidents across as many as 989916 UAVs flown.[1] To the best of our knowledge this makes it largest field failure study conducted on UAVs.

It is expensive to wait for real-world accidents to occur in order to learn about UAV safety and reliability issues. Therefore, to scalably test UAV systems, we have built UAVFI, a fault injection framework that can take advantage of two different photo-realistic world simulators [13], [14] to capture different aspects of the simulated world and the model the incidence and propagation of faults in the UAV, encompassing the mechanical components of the UAV, the sensors, hardware and software components of the on-board computer, as well as key algorithmic aspects of the ob-board controller (e.g., route planning and its dynamics). Hence, UAVFI allows us to: (i) experiment with fault models, (ii) understand fault propagation and (iii) assess fault manifestation in a more cost-effective way than collecting field data. UAVFI leverages our insights from the field data analysis to identify fault models that are critical for UAV safety.

Prior fault injection approaches [15]–[17] have focused on showing that control system simulators can be used to inject faults into an aircraft's feedback system components (e.g., an aileron stuck at an angle). However, modern UAVs rely significantly on a combination of perception algorithms to infer their surrounding environments [18]–[20], thereby reducing the viability of those injectors. In contrast, UAVFI is based on simulators such as AirSim [13] or Gazebo [14], to enable high-fidelity graphics-based (i.e., with cameras, RADAR, LIDAR) simulation of the surrounding environment and UAVs, thereby allowing them to be used in modern perception algorithms. However, the fault models and injection mythologies are not well established for these platforms. This paper presents a fault-injection-based testbed on the AirSim and the Gazebo [14], which are UAV simulators that are widely used in the robotics community. By enabling fault injection on these two common UAV simulators, we hope to promote itegration of UAVFI into many existing UAV research and development projects. Thus, the injection tools we developed for those simulation platforms are readily usable

[1]Registered model UAVs with FAA in the U.S. [12]

by the UAV development community.

The main contributions of this paper can be summarized as follows:

1) We surveyed and analyzed a multinational accident database detailing UAV system failures over the last five years. We found several conclusions. For example, the top two non-human fault sources are sensors and mechanical components (21% of the total faults) and communication link failures (15% of the total faults). In Section II, We present challenges and lessons learned to ensure UAV safety, based on the latest real-world data.
2) We developed a fault injection framework, UAVFI, which enables end-to-end resilience assessment of autonomous UAVs. Our framework supports the latest UAV features driven by computer-vision technology. The framework was developed for both the AirSim and Gazebo simulators, and is open-source for public access.
3) Using UAVFI, we created fault injection campaigns that cause as much as 75.9% increase in mission completion time, as well as 3.5× increase in accident incidence. Further we showed that UAV control algorithms built without explicit fault handling is deficient for providing reliability. Our findings call for the consideration of real-world faults in the algorithm development phase.

II. Multinational Civil UAV Accident Reports

A. Data Sources and Collection

To reveal the reliability challenges of current UAV systems, we surveyed real-world UAV accident data from a multinational database, including data from the following sources:

- NASA and the FAA's Aviation Safety Reporting System (ASRS) [6].
- U.S. Aviation Safety Communiqué (SAFECOM) data [7].
- Australian Transport Safety Bureau (ATSB) aviation safety investigations & reports. [8]
- UK Air Accident Investigation Branch (AAIB) [9] data.

A total of 160 accident reports from 2015 to 2020 were collected and documented in an integrated database. We used key words such as *UAV*, *UAS*, *UA*, *SUA*, *UAM*, and *drone* to query relevant reports. Further, we applied a rule-based natural language processing (NLP) which extracts key words to obtain attributes (such as accident results, root causes, and flight phases) from the original reports, which were recorded in narrative form. However, because the accident reports (in narrative form) do not share a consistent structure across organizations, our pipeline involved manual effort to standardize and verify the accident database. As an illustration, we present sample records from the database in Table I[2].

To the best of our knowledge, this is the largest and most up-to-date multinational data analysis of UAV failures. In [11], Wild et al. collected and analyzed 152 incident and accident records, but only 40 records deal with UAV accidents, and the data date back to the 2006–2015 period. The dataset we analyze here is substantially larger and captures the latest trends.

[2]The attribute fields in the database include data source, accident year, month, location, phase of flight, aircraft model, flight hour history, flight duration in mission before accident, altitude, weather, mission being conducted, root cause, and accident outcome.

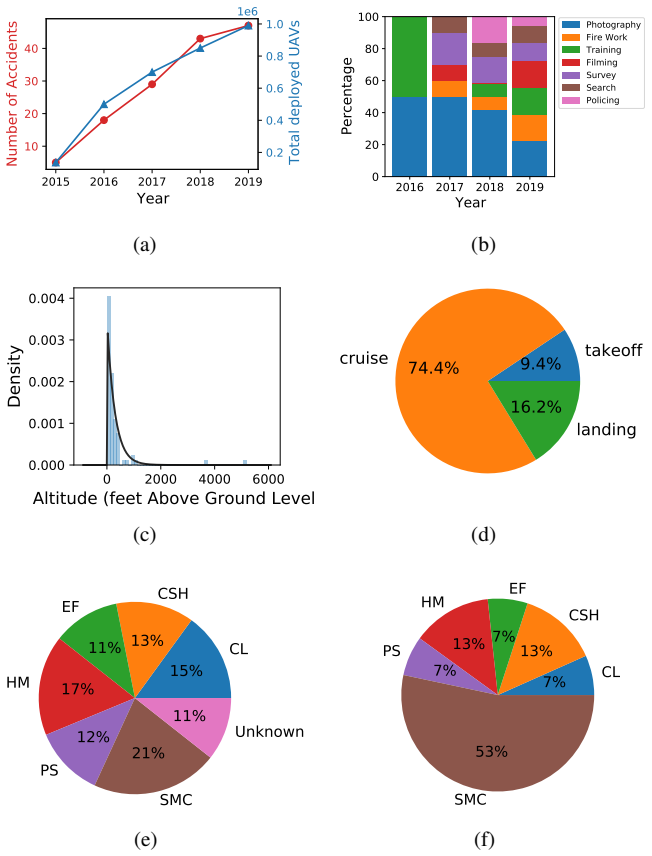


Fig. 1. Results from real-world accident data: (a) Reported number of accidents per year; (b) Breakdown of the UAV application (58 records involved); (c) Distribution of the flight altitude; (d) Breakdown of the flight phase (160 records involved); (e) Breakdown of the root causes for all accidents (160 records involved); (f) Breakdown of the root causes of accidents that happened at the takeoff phase (15 records involved). The acronyms used for root causes are defined at Table. II

B. Data analysis of UAV failures

As depicted in Fig. 1(a), the number of accidents continually increased over the period from 2015 to 2019. However unlike other autonomous systems, such as self-driving cars [21], UAVs do not show a distinct trend of decreasing the number of incidents with time. Instead it stays near constant at 0.0000365 accidents/UAV[3] with less than 0.000582% standard deviation over period from 2015 to 2019.

Fig. 1(b) presents a breakdown of the types of missions conducted by UAVs. It can be observed that compared to ground autonomous vehicles, which are mainly used for transportation, UAVs have much more diverse applications, and such diversity is increased over the years.

Fig. 1(c) shows that the altitude at which failures occur follows an exponential distribution, which indicates that more safety consideration should be given to low-altitude flight operations and monitoring.

Fig. 1(d) shows a breakdown of the flight phases during which failures occurred. Notably, the takeoff and landing stages account for 25.6% of accidents. Given that the takeoff and landing stages account for only a small fraction of the overall flight time, it is evident that these two flight stages are

[3]The total number is from FAA's report on registered model UAVs [12]

TABLE I
SAMPLE ENTRIES WITH INTERESTING ATTRIBUTES FROM THE DATABASE.

Data Source	Year	Accident Causes	Phase of Flight	Mission Conducted	Altitude	Accident Result
ATSB	2019	Blockage in the pitot-static system	Cruise	N/A	N/A	Collision with terrain
AAIB	2018	Failure of ESC	Cruise	Targeting vehicle crime	50 m	Crashed into a building
ASRS	2017	Loss of radio connection	Cruise	Aerial photography	N/A	Loss of vehicle
SAFECOM	2020	Human pilot failed to see a tree line	Landing	Aerial ignition	15 ft	Collision with a tree
SAFECOM	2018	Propeller broken	Takeoff	Aerial photography	40 ft	Tumbled to the ground

relatively accident-prone.

Fig. 1(e) shows the overall breakdown of accident causes; the top three risk factors are (1) malfunction of sensors & mechanical components (SMC), (2) human misoperation (HM), and (3) disrupted communication link (CL). The distribution of accident causes appears to be uniform across causes. The cruise and landing stages have more uniformly distributed root causes which is in contrast with takeoff phase where mechanical component & sensor malfunctions account for 53% of the accident causes (as shown in Fig. 1(f)). The difference suggests that errors due to incorrect sensor calibration or sensors' fragile structures tend to manifest at the beginning of the flight, while errors in path planning and navigation are more likely to happen at later stages.

The root cause categories we used in Fig. 1(e) and 1(f) is based on the taxonomy we developed, which incorporates both internal and external factors. Internal factors include failures of sensors & mechanical components, power systems, and computer software & hardware; external factors include failures due to communication link disruption, human misoperation, and environmental disruption. The detailed decomposition and example failure causes can be found in Table II.

C. Rendering of real-world UAV accidents

The rendering of real-world accidents helps us understand how the faults propagate and manifest and answer what-if questions to expand the failure scenarios for investigation. It also serves as a platform for verifying fault mitigation strategies. To achieve high fidelity, we recreated accident scenarios in the AirSim [13] simulator by injecting faults. We discuss the details of the injection tools for AirSim in Section III-B. As shown in Fig 2, we selected and rendered three representative accident scenarios (covering diverse flying environments and both internal and external accident causes). Video-recordings of these rendered scenarios are available at [22].

1) **Failure of obstacle avoidance system.** This accident happened at Mill Lane, Ludlow, UK, 03/2020. As the UAV was following the planned trajectory, its collision avoidance system failed to detect an obstacle to the right of the drone, in part because the camera was not pointing in the direction of flight and the obstacle was not visible in the camera's view. It resulted in a collision of the drone with a tree, causing extensive damage to the motor arms, propellers, batteries, gimbal bracket, and cameras.
2) **Adverse weather conditions.** This accident happened at Region 05 Pacific Southwest Region, US, 08/2017. In the landing phase, the UAV encountered a fierce wind gust that caused it to crash into the ground, damaging its propellers.
3) **Positioning error, poor GPS and compass signal strength.** The accident happened at Sampson House, London, UK, 02/2020. During flight, the UAV suffered position errors, as well as poor GPS and compass signal strength. The UAV went out of control and collided with a concrete structure on the site.

The recreations of accident scenarios demonstrate the feasibility of testing UAVs in the AirSim simulator, which is able to cover diverse failure sources. As a cyber-physical system exposed to the wild, UAVs are highly susceptible to peripheral and environmental disruptions. Hence, relevant fault models should be evaluated during the design phase before deployment in the wild.

D. Lessons learned for reliability analysis

The UAV accident reports disclose reliability challenges in real-world deployments and lessons on how to enhance the reliability of UAV systems.

1) UAV **faults are more diverse** than those of traditional piloted aircraft and require more comprehensive resilience test cases. Compared with traditional commercial air transport (CAT), for which the human factor accounts for 75% of accident causes [11], [23], the root causes of UAV accidents are more diverse and are distributed more uniformly (as shown in Fig. 1(e)) across different vehicle components as well as the human factor.
2) The top two non-human fault sources are sensors and mechanical components (21% of the total faults) and communication link failures (15% of the total faults). Common fault models, such as weak GPS signal, corrupted compass measurement, and disconnection with the ground control station (GCS), as revealed by real-world accident data, should be emphasized in resilience engineering.
3) Multi-purpose UAV applications raise a demand for domain-specific assessments rather than generic assessments. Compared with ground autonomous vehicles, unmanned aerial vehicles have much more diverse applications (as shown in Fig. 1(b)). UAVs in use for different applications have different safety or dependability requirements. For instance, package delivery UAVs can have different performance metrics and working conditions from those used for object tracking or searching.
4) As hazardous phases for UAVs, the takeoff and landing phases have skewed distribution of fault causes (as seen in the Fig. 1(f)). Therefore, specialized test cases should be created to cover takeoff and landing scenarios separately from the cruise phase.
5) Vehicle control algorithms should be assessed in conjunction with the whole system (i.e., end-to-end safety

TABLE II
TAXONOMY OF FAILURE CAUSES BASED ON REAL-WORLD UAV ACCIDENTS.

Relative to UAV Agent	Fault Category	Subcategory
Internal factors	Sensors & mechanical components (SMC)	Motor (including its control board) Propeller & blade Structural (e.g., bonding) Sensors (camera, IMU, GPS, etc.) Unknown
	Power system (PS)	Battery Power regulation firmware
	Computing software & hardware (CSH)	System on a chip (SOC) Erroneous flight plan Autopilot Electronic speed controller (ESC)
External factors	Communication link (CL)	Loss link; no response; loss control
	Human misoperation (HM)	Improper calibration Aggressive/risky operation Inexperienced piloting (e.g., misjudgment)
	Environmental disruption (ED)	Weather (wind, rain/wet) Magnetic interference GPS interference Radio interference Animal attacks

(a) (b) (c)

Fig. 2. Rendering and investigating real-world accidents with AirSim: (a) Forest; (b) Suburb; (c) Urban city.

assessment). Although its embedded algorithm may be flawless, a UAV can still crash because of failures in peripheral components. The computer software accounts for only 8.1% of the total faults. Common faults and environmental interruptions should be considered in the algorithm development phase.

III. FAULT INJECTIONS FOR UAVS

Real-world UAV failures reveal the need to consider comprehensive fault models for safety assessment. Meanwhile, in developing robust UAV autonomy algorithms, realistic faults need to be integrated into the simulator for fault-tolerance evaluation. This section presents UAVFI, a fault-injection-based testbed we developed to facilitate testing of autonomous UAVs. It provides a platform to take real-world failure scenarios, explore the underlying failure causes, and study the manifestation and propagation of the faults. We characterized failures seen in accidents by using fault models, and evaluated a state-of-the-art navigation algorithm [24] with the testbed.

Existing fault injection work [15]–[17] assess the reliability of UAV control algorithms in low-fidelity environments which do not provide standalone real-world models that the UAV can interact with. They neither support computer vision algorithms to perceive the surrounding environment, nor assess AI based decision making algorithms, which are required for the autonomy. Since the major challenges in autonomous UAV development involve building robust perception, decision and control systems, it is imperative that fault injection frameworks be integrated into high fidelity simulators that enable UAVs to operate in realistic, dynamic and interactive environments. Our fault injection framework aims to address these limitations by building a fault injector for high-fidelity simulators AirSim and Gazebo. These platforms also allow the end-to-end testing of computer-vision and planning algorithms required for autonomous operation. To this extent, we show how are fault injection tool can be used to assess autonomous navigation algorithms on the two simulators. These experiments are more representative of real world autonomous UAVs than those in prior work.

A. High-fidelity UAV simulation and autonomous flight agents

High-fidelity flight simulators provide a cost-effective method for testing UAVs while maintaining a realistic vehicle model. Traditional simulators such as JSBSim and Matlab/Simulink focus on the hardware and dynamic control of UAVs, but do not provide a photo-realistic environment. Recent simulators, however, enable the testing of computer-vision-based perception modules, which enables collision avoidance and path planning. A number of vision-based navigation algorithms have been developed, and their robustness needs careful assessment. Because of their popularity in the industrial and academic sectors, as well as their compatibility with visual cameras, we adopted the AirSim and Gazebo simulators in the present work, and we built our fault injection tools upon these two simulators.

Microsoft AirSim is an open-source cross-platform simulator

built on Epic Games' Unreal Engine 4. It provides high-fidelity visual and physical simulation for autonomous vehicles [13]. AirSim has 2 main modules: the client and the server. The autonomous navigation algorithm is run in the client, which takes sensor data and outputs the actuation commands. The server leverages Unreal Engine (popularly used in video games) as its physics and high-fidelity rendering engine. It supports prevalent UAV sensors, such as cameras, barometers, IMU, GPS, magnetometers, and Lidar. Common weather effects, like rain and wind, are tunable in AirSim. In addition, a Python API and several baseline vehicle maneuver algorithms are provided to interface with the server. The agent run in our experiments is based on the navigation algorithm provided in the AirSim code repository [25], which perceives obstacles by using depth images.

Gazebo and FASTER is an open-source 3D dynamic simulator that generates realistic sensor feedback and physical interactions between objects. It is widely used in robotic research and has been integrated with the Robot Operating System (ROS). Moreover, the ROS and Gazebo platform has a larger collection of existing UAV autonomy algorithms than AirSim. The Fast and Safe Trajectory Planner for Flights in Unknown Environments (FASTER) [24] is a novel planning algorithm for generating high-speed trajectories to a goal point in an unknown environment while maintaining safety. FASTER does so by enabling the local planner to optimize in both the free-known and unknown spaces. However, FASTER assures safety by having a safe fallback trajectory that is in the free-known space at every replanning step. Thus, if the fast trajectory is not feasible, the UAV can take the safe trajectory until another feasible fast trajectory is found, or stop safely if no path is found.

One common factor among these navigation algorithms is that they do not handle failures that might occur in the UAV that they are controlling. Traditionally, fault tolerance was addressed separately from the algorithms, through use of redundancy or duplication of hardware, software, and sensor components. However, that is not a viable approach for fault tolerance in UAVs because of cost, weight, and other mechanical constraints. Hence, fault tolerance needs to be integrated into the algorithms that are responsible for controlling UAVs.

B. Design of the testbed using UAVFI

The testbed design for autonomous UAVs is shown in Fig. 3 and consists of (a) a world simulator for modeling vehicle physics and rendering the environment (e.g., AirSim, Gazebo); (b) an autonomous flight engine, which plans the UAV's trajectory and performs real-time collision avoidance; and (c) the UAVFI fault-injection-based resilience assessment engine.

Fault models and injectors: The fault injection manager parameterizes each fault model and offers flexibility on (a) when the faults manifest, (b) where the faults localize, (c) what types of fault to inject, and (d) the strength of faults. We have developed a wide spectrum of faults, as follows:

- *Data Faults:* The measurement data can be corrupted due to faulty sensors or environmental disruptions, such as lightning, electromagnetic interference, and occlusions on a camera lens. UAVFI emulates these faults by perturbing sensor readings (e.g., camera, IMU, compass) and manipulating weather conditions (e.g., sunny, rainy, windy).
- *Hardware Faults:* UAVFI emulates hardware faults in sensors, memory, processors, and communication devices. Typical errors include single-bit flips and multi-bits in electronic circuits. A sample fault injection might consist of intercepting the yaw control command, performing bit error injection, and then sending the perturbed command to the drone running in the simulator.
- *Timing Faults:* UAVFI incorporates the following timing faults that represent clock and synchronization errors: (a) delay of data delivery, (b) loss of data, and (c) out-of-order data delivery.
- *Algorithmic Faults:* The algorithms used to make decisions in autonomous UAV agents are also susceptible to faults, wherein a small alteration of the input or intermediary states results in erroneous outputs. For example, it has been shown that neural networks are susceptible to adversarial attacks [26]. In the FASTER algorithm [24] used in this paper, the intermediate waypoints are perturbed with an algorithm-oriented injector.

Performance metrics: We utilized the following metrics to assess the resiliency of autonomous UAVs under faults.

- *Time before collision (TBC)* is the UAV flight duration time before the first collision happens. A smaller TBC value indicates a more hazardous UAV safety condition.
- *Exploration space (ES)* measures the effective flight distance of the UAV. It is defined by

$$D = \|(x_{max}, y_{max}) - (x_{min}, y_{min})\|$$

where x and y are coordinates in horizontal space obtained from GPS or IMU. The ES metrics capture the degradation in autonomous performance. For example, if the UAV stops or circles around locally, the TBC is still zero, but the ES will be abnormally small.

To enable fault injection in Gazebo, we set up an additional ROS node as a fault injection campaign manager. It subscribes to the existing topics, injects the faults, and then publishes perturbed data to corrupt the operation of UAVs.

IV. Experiment Results

This section presents some fault tolerance experiments we performed on two UAV simulation platforms: AirSim and Gazebo. The autonomy algorithm run on AirSim is based on [25] while the algorithm run on Gazebo is [24]. The goal of the fault-tolerance experiments discussed in this section was not to point out the weaknesses of the algorithms, but to raise awareness of the dependability issues that can arise when these algorithms are deployed in UAVs in the real world.

The failure modes aim to demonstrate deficiencies in fault tolerance rather than empirically evaluate the probability that the accidents can occur in real life and their effects.

A. Demonstrating UAVFI on AirSim

With the fault models described in Section III-B, it is possible to experiment on a large space of faults in AirSim. For demonstration purposes, we present a case study in which

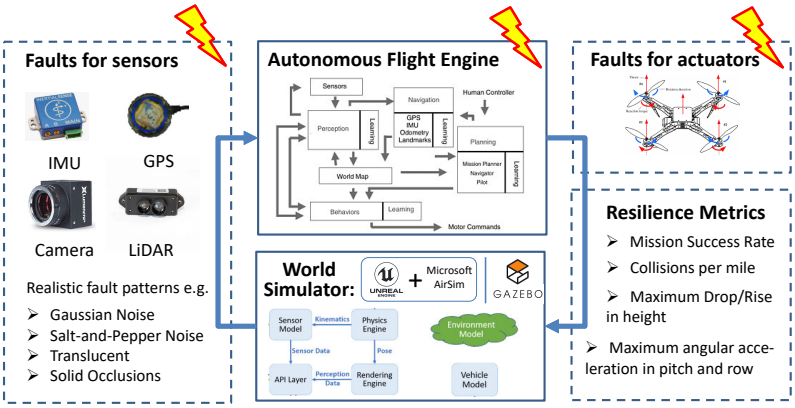
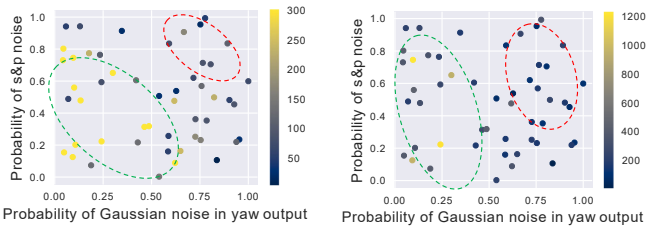


Fig. 3. UAVFI: Multi-platform fault injection framework for autonomous UAVs.



(a) (b)

Fig. 4. Performance of collision avoidance algorithm under 2-D fault injections, evaluated with (a) a time-before-collision metric, and (b) an exploration-space metric.

two types of faults are simultaneously injected: (1) salt-and-pepper noise injected in the depth sensor, and (2) Gaussian noise injected at the output yaw maneuver. Those two faults have semantic meaning to the navigation algorithm running on AirSim because the algorithm utilizes a depth camera to perceive obstacles and uses output yaw commands for collision avoidance.

We used a probability parameter that ranged from 0 to 1 to alter the salt-and-pepper noise for the depth camera. Meanwhile, the Gaussian noise for the output yaw control had a standard deviation parameter that could be altered between 0 and 1. In each run, the parameters of the two fault models were uniformly sampled. With the two faults injected, we ran the UAV for 5 minutes (per run) in the simulator and measured the TBC and ES metrics (as described in Section III-B).

In Fig. 4(a), we present the results in terms of the TBC metric. The value of the TBC (in seconds) is represented on the blue-to-yellow color scale, on which the maximum flight time (300 seconds) is represented by yellow. The x axis and y axis present the severity of faults. The data points at the bottom left corner (with less significant perturbation) tend to have larger TBCs, while the data points at the top right tend to have smaller TBCs. This result shows that the fault injections with greater perturbations increase the possibility of collisions. Experimental results for the ES metric are presented in Fig. 4(b), in which smaller ES values can be observed on the right of the graph. This indicates that the injection into the output yaw degraded the UAV's ability to explore the world.

One important feature of our injector is its capability to simulate faults in vision sensors. Preliminary results using our

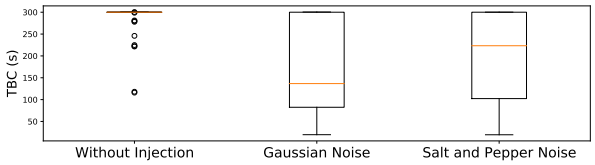


Fig. 5. Box plot for TBC under various fault models in AirSim.

fault injector show promising results. Fig. 5 shows the increase in variance as well as a significant decrease in the mean value of the TBC when Gaussian and salt-and-pepper noise were injected into the camera of the UAV in Airsim. Note that it can not be concluded from the box plot that Gaussian noise is more effective than salt-and-pepper noise, because they are parameterized differently. Our fault injection campaign resulted in 54.4% degradation in the median TBC value for Gaussian noise case and 25.6% for salt-and-pepper noise. Such a metric can not be found with traditional injectors that do not simulate high-fidelity vision environments. Moreover, the probability increases by 3.5 × when using the Gaussian noise fault model and 2.9 × when using the salt-and-pepper noise fault model.

B. Demonstrating UAVFI on Gazebo using FASTER

To demonstrate the algorithm, [24] uses the Gazebo simulator to provide the environment with static obstacles as well as a model of the UAV. The drone in the simulator was fitted with a forward-facing depth camera. FASTER interfaces with Gazebo via the Robot Operating System (ROS). At a high level, there are three key modules that enable the drone to navigate safely and quickly from the starting point to the goal point in the simulation environment: 1) the global mapper, which takes in the sensor readings from the depth camera and reconstructs the obstacles in a 3D point cloud representation; 2) the FASTER algorithm, which takes in the reconstructed point cloud, current drone position, and goal point to generate fast and safe trajectories; and 3) the Gazebo simulator, which takes the trajectory from FASTER, moves the drone, and provides the depth image for the next time-step. The current implementation assumes that the drone perfectly tracks the trajectories generated by FASTER.

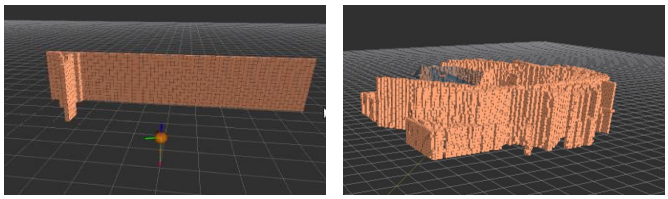
For our experiments, we considered two potential threats to fault tolerance. The first threat affects the global mapper

module, such that noise in the depth camera readings (i.e., the salt-and-pepper noise) causes spurious obstacles to appear in the global point cloud. The second threat affects the FASTER trajectory planner such that there is uncertainty in the starting position of the next replanning step (Point A in the FASTER algorithm). We evaluate how the FASTER algorithm responds to those threats in simulation and present the scenarios below as three cases.

Case 1: Salt-and-pepper noise in depth camera. To simulate salt-and-pepper noise in depth camera images, we intercepted the depth camera images from the simulator, perturbed the depth image, and forwarded the perturbed depth image to the global mapper. We injected into a frame with a probability of 0.4 and perturbed a pixel in the depth image (to a distance of either 0 m or the sensor's maximum range, in this case 5 m) with a probability of 0.8. Fig. 6(a) shows the effect of the salt-and-pepper noise on the global mapper. The orange blocks represent obstacles detected using the depth camera. The noise injected in the depth images resulted in generation of a number of spurious (fake) obstacles in the 3D point cloud that was sent to FASTER for planning. The results from the simulation show that despite the spurious obstacles in the 3D point cloud, the drone was able to reach the destination, although the flight time increased because of the effort of navigating around the fake obstacles. Because of the nature of the depth image to global point cloud mapping algorithm, once an obstacle has been registered in 3D space, it will take multiple consecutive frames from the depth camera showing that the 3D point is unoccupied for the mapper to deregister the obstacle. Although the algorithm parameters can be adjusted to specify how many frames are required to register/deregister obstacles, we can still conclude that the global mapping algorithm is sensitive to noise in the camera.

In the above scenario, fault tolerance by redundancy (e.g., duplicating the depth camera or the mapping algorithm) will not solve the issue, because both cameras are susceptible to noise. Ideally, fault tolerance needs to be built into the mapping algorithm.

Case 2: Gaussian noise in waypoints used in planning by FASTER. In planning trajectories, the accuracy of the computed waypoints is critical. However, multiple factors, such as uncertainty in the current drone position, velocity, and environmental factors such as wind, can result in uncertainty in the computed waypoints. In this case study, we investigated the effect of uncertainty in the starting waypoint, A, on the FASTER planning algorithm by adding Gaussian noise $N(\mu = 0, \sigma = 1)$ to A's coordinates at every time-step. As a result of a small variance, the drone was still able to reach the destination, although it took significantly longer. It was able to do so as long as A wasn't perturbed to be inside obstacles defined by the 3D point cloud. The preliminary result shows that the planning algorithm has some built-in fault tolerance to a small amount of noise in the position of A, mainly because point A is the starting point/input to the replanning step, and any unsafe changes can be, to some extent, mitigated by planning a new trajectory that is still safe. However, some intermediate points, like R (where the safe and fast trajectories diverge) may be more susceptible to faults and needs further exploration to understand the sensitivity to noise/errors.



(a) (b)

Fig. 6. Results of the injections on Gazebo: (a) Perceived illusion wall; (b) Enclosure made by a virtual obstacle barrier.

Again, adding redundancy through duplication will not help in addressing the reliability issue, since noise is not eliminated through duplication. The algorithm needs to be tuned to handle noises/errors to provide fault tolerance.

Case 3: Combining Cases 1 and 2. By combining salt-and-pepper noise in the depth camera and Gaussian noise in the replanning starting waypoint, A, we were able to uncover a situation in which the drone was unable to navigate to the destination because it enclosed itself in a virtual obstacle barrier, as shown in Fig. 6(b). As the drone in the simulation perfectly tracks the trajectory output by FASTER, the drone's position is also shifted by $N(\mu = 0, \sigma = 1)$ when we perturb waypoint A at each time-step. When the Gaussian noise was combined with the salt-and-pepper noise, the net effect was to create additional fake obstacles in the 3D point cloud, creating a situation in which it trapped itself in virtual closed space.

Additional feature of the fault injector is to show that the s&p noise causes the mission completion time to increase (by 75.9%) (no noise case <median:23.7s> vs salt-and-pepper noise <median:41.7s>) with injection at the waypoints A and E, our results are also significant (<median:87.3s> and <median:29.9s>)

V. Conclusion

This paper studied real-world UAV failures and provided the lessons learned from root causes, phases of flight, and other aspects of the accidents. The data showed that faults in UAVs are diversely distributed among different components of the system, emphasizing the need to develop comprehensive test cases for the end-to-end resilience evaluation of UAVs. We developed UAVFI to assess critical UAV control algorithms against failure modes and faults derived from our preliminary study. The injection experiments on the FASTER algorithm revealed that malfunctions could be triggered by faults, resulting in safety hazards (e.g., shortened TBC) and performance degradation. Another key insight is that the collision avoidance algorithm in AirSim, despite its simplicity, is more robust to sporadic salt-and-pepper noise in the depth images than the FASTER algorithm is. This is because FASTER preserved the fake obstacles for some consecutive frames, while the algorithm run in AirSim only use the latest frame to perceive the environment. Our findings on both algorithms show it is crucial that faults be considered and addressed in the algorithm development phase, and our open-source computer-vision-supporting testbed provides a viable solution to assess the end-to-end robustness of the navigation and control algorithms in UAVs.

REFERENCES

[1] A. Beg, A. R. Qureshi, T. Sheltami, and A. Yasar, "Uav-enabled intelligent traffic policing and emergency response handling system for the smart city," *Personal and Ubiquitous Computing*, pp. 1–18, 2020.

[2] Fact sheet – general aviation safety, faa. [Online]. Available: https://www.faa.gov/news/fact_sheets/news_story.cfm?newsId=21274

[3] D. Hambling, "Drone crash due to GPS interference in U.K. raises safety questions," *Forbes*. [Online]. Available: https://www.forbes.com/sites/davidhambling/2020/08/10/investigation-finds-gps-interference-caused-uk-survey-drone-crash/?sh=2e18e8a8534a

[4] "Norwich police drone crashed 'with a second's notice'," *BBC*. [Online]. Available: https://www.bbc.com/news/uk-england-norfolk-51049500

[5] C. Matlock, "Unmanned aircraft crashes into airport hangar, catches fire." [Online]. Available: https://www.wcbi.com/unmanned-aircraft-crashes-airport-hangar-catches-fire/

[6] "Aviation safety reporting system," *NASA,FAA*. [Online]. Available: https://asrs.arc.nasa.gov/

[7] "Aviation safety reporting system." *The Department of the Interior (DOI) and the U.S. Forest Service (USFS)*. [Online]. Available: https://www.safecom.gov/about

[8] "ATSB aviation safety investigations," *Australian Transport Safety Bureau (ATSB)*. [Online]. Available: https://www.atsb.gov.au/publications/safety-investigation-reports/?mode=Aviation

[9] Australian Transport Safety Bureau (ATSB), "ATSB aviation safety investigations." [Online]. Available: https://www.atsb.gov.au/publications/safety-investigation-reports/?mode=Aviation

[10] C. M. Belcastro, R. L. Newman, J. Evans, D. H. Klyde, L. C. Barr, and E. Ancel, "Hazards identification and analysis for unmanned aircraft system operations," in *17th AIAA Aviation Technology, Integration, and Operations Conference*, 2017, p. 3269.

[11] G. Wild, J. Murray, and G. Baxter, "Exploring civil drone accidents and incidents to help prevent potential air disasters," *Aerospace*, vol. 3, no. 3, p. 22, 2016.

[12] "Report on registered uavs." [Online]. Available: https://www.faa.gov/data_research/aviation/aerospace_forecasts/media/Unmanned_Aircraft_Systems.pdf

[13] S. Shah, D. Dey, C. Lovett, and A. Kapoor, "AirSim: High-fidelity visual and physical simulation for autonomous vehicles," Springer, pp. 621–635, 2018.

[14] N. Koenig and A. Howard, "Design and use paradigms for Gazebo, an open-source multi-robot simulator," in *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)(IEEE Cat. No. 04CH37566)*, vol. 3. IEEE, 2004, pp. 2149–2154.

[15] S. Gong, S. Meng, B. Wang, and D. Liu, "Hardware-in-the-loop simulation of UAV for fault injection," in *2019 Prognostics and System Health Management Conference (PHM-Qingdao)*. IEEE, 2019, pp. 1–6.

[16] C. Bo, W. Benkuan, M. Yuntong, and P. Yu, "A fault injection platform for multirotor UAV PHM," in *2019 14th IEEE International Conference on Electronic Measurement & Instruments (ICEMI)*. IEEE, 2019, pp. 954–959.

[17] J. Wen, H. Ji, H. Wang, M. Zhang, D. Li, and J. Wu, "Design of a real-time UAV fault injection simulation system," in *2019 IEEE International Conference on Unmanned Systems (ICUS)*. IEEE, 2019, pp. 767–772.

[18] M. Iacono and A. Sgorbissa, "Path following and obstacle avoidance for an autonomous UAV using a depth camera," *Robotics and Autonomous Systems*, vol. 106, pp. 38–46, 2018.

[19] P. H. Nguyen, M. Arsalan, J. H. Koo, R. A. Naqvi, N. Q. Truong, and K. R. Park, "LightDenseYOLO: A fast and accurate marker tracker for autonomous UAV landing by visible light camera sensor on drone," *Sensors*, vol. 18, no. 6, p. 1703, 2018.

[20] S. Lange, N. Sünderhauf, P. Neubert, S. Drews, and P. Protzel, "Autonomous corridor flight of a UAV using a low-cost and light-weight RGB-D camera," in *Advances in Autonomous Mini Robots*. Springer, 2012, pp. 183–192.

[21] S. S. Banerjee, S. Jha, J. Cyriac, Z. T. Kalbarczyk, and R. K. Iyer, "Hands off the wheel in autonomous vehicles?: A systems perspective on over a million miles of field data," in *2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, 2018, pp. 586–597.

[22] Rendering real-world UAV accidents in AirSim with fault injections. [Online]. Available: https://youtube.com/playlist?list=PL9Vs78H1P-cpSJr0vo5ZUbUjV94z26WuP

[23] "Annual safety review 20," *EASA*. [Online]. Available: https://cdn.aviation-safety.net/airlinesafety/industry/reports/EASA-Annual-safety-review-2014.pdf

[24] J. Tordesillas, B. T. Lopez, and J. P. How, "FASTER: Fast and safe trajectory planner for flights in unknown environments," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2019.

[25] Multirotor maneuver algorithms provided with AirSim. [Online]. Available: https://github.com/microsoft/AirSim/tree/master/PythonClient/multirotor

[26] N. Akhtar and A. Mian, "Threat of adversarial attacks on deep learning in computer vision: A survey," *IEEE Access*, vol. 6, pp. 14 410–14 430, 2018.