International Research Journal of Computer Science (IRJCS)
Issue 5, Volume 2 (May 2015)
ISSN: 2393-9842
www.irjcs.com

Mitigation Settings and their Execution Scenario in Drone Firmware

Hoijin Yoon
Department of Computer Engineering, Hyupsung University

Abstract— The autopilot on an Unmanned Aerial Vehicle (UAV) should be a safety-critical system. Various risk mitigation approaches and behavioural testing are requested to support autopilot safety. We attempted to analyse a method for setting the data that affect the autopilot of a UAV. In this paper, we narrow down our efforts to battery-related data because battery problems can be responsible for most failures. This paper analyses the battery-related parameter settings using a real drone model manufactured by 3D robotics. The present analysis can be used in the design of test cases for the black box testing of the battery parts of the autopilot firmware used in a drone vehicle.

Keywords— Mitigation, Autopilot, UAV, Safety, Pixhawk, Mission Planner

I. INTRODUCTION

Techniques for implementing UAVs are developing quite rapidly these days. Google cars have driven on the streets in California. Of course, this was only an experimental demonstration, but it shows that this dream technology is coming to fruition. Another type of UAV is a drone. Drones were originally used for military purposes, but they have spread into other parts of our lives including delivery services, and fashion shows. Amazon has been planning and is currently using drones for deliveries. Drones should be able to pilot to accurate waypoints and drop at the proper place successfully without crashing.

Safety is one of the main qualities of the autopilot used in a drone. An autopilot system is handled through firmware, which refers to the related-parameters and waypoints, which are set by the developers. Based on the settings of the parameters and waypoints, the autopilot function is implemented in firmware. Mitigation options are also included in the parameter settings. The results described herein may be a good source for the design of safety test cases on battery risk. According to the crash history of drones, battery risk can result in drone crashes.

We examined an X8 as a sample drone, which is manufactured by 3D Robotics, which is a leading company in the drone industry. Qualcomm and some other companies have invested significant funds into 3D Robotics for mobile-phone based drone control [1].

II. RELATED WORKS

A. *Drone Crashes*

In 2014, some meaningful data were published at the Drone Wars site, which lists historical drone crashes that have occurred since January 1st, 2007 [2]. The site focuses solely on military drones, and excludes hobby-like drones. Table 1 shows some of the crashes listed. As the column, “cause details,” shows, various failures have occurred. Mechanical failures, lost links, or power failures can result from battery problems.

TABLE I.
SOME CASES OF DRONE CRASHES [2]

Date	Operator	Drone Type	Cause details	Location
Apr. 26, 2014	US Air Force	MQ-1B Predator	Oil Leak	Afghanistan
Jan.28, 2014	US DHS	MQ-1B Predator	Mechanical Failure	California, US
Oct.16, 2013	US Army	MQ-1C Gray Eagle	Lost link	Afghanistan
May 10, 2012	Schiebel	Schiebel S-100	Lost link	South Korea
Aug.18, 2010	US Army	MQ-1B Warrior	Lost link	Iraq
Dec. 17, 2007	US Air Force	MQ-1B Predator	Power failure	Iraq

Incorrect battery management and mitigation are important causes of drone crashes. In fact, the mitigation options for battery risk are selected by users before flying a drone. The mitigation options are referred to by firmware during autopilot control.

B. Safety Mitigation

Interest in risk mitigation has increased with the emergence of safety as an important issue. With respect to the various risks to drone flights, problems may occur when these risks are actually realized. Through traditional risk management and measurements, the effects of reducing the risk probability or the risk impact are obtained. This effort is focused on the mitigation of risks. For these mitigation efforts, many patterns can be found in existing works [3, 4, 5], such as a rollback, roll-forward, immediate fixing, deferred fixing, retry, compensation, and go-to-failsafe state. Among them, the mitigation setup shown in Fig. 3 corresponds to the go-to-failsafe state. This mitigation pattern implies that a system is transferred into a mitigation state to avoid dangerous effects and stops.

III. BATTERY-RELATED MITIGATION SETTINGS

A. Autopilot Implementation

An autopilot system is used to make a drone operate as programmed. It includes the waypoints that the drone should visit, and a mitigation plan under expected risky conditions. In addition, an RC controller can be used to control a drone manually. Figure 1 (a) shows an RC controller. However, an RC controller works only within a certain range of distance between the controller and drone. To support long-distance flying, an autopilot function is necessary, and a specific device mounted on the drone is needed to store the autopilot program. Figure 1 (b) shows a Pixhawk, which includes device drivers and Firmware. Pixhawk can be mounted on a drone for autopilot control [6].

(a) RC Controller (b) Pixhawk

Fig. 1. Devices used for controlling a drone.

RC controllers have been used a great deal for flying hobby-like drones. Such controllers have two drawbacks: a short working distance and inaccurate waypoints. In addition, Pixhawk supports long-distance flying and accurate pre-defined GPS-based waypoints. Therefore, the autopilot function is one of the necessities in the current trends of drone use in our lives; for example, Amazon is using drones in their deliveries. Pixhawk supports the X8 drone manufactured by 3D Robotics. Herein, we describe how to implement the autopilot function of Pixhawk for an X8 drone in detail.

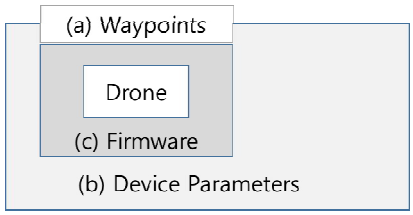

Figure 2. The parts supporting a drone autopilot.

Figure 2 shows the parts implemented in a drone autopilot system. In Figure 2, a drone is located at the center of the image. Around the drone are the parameters, waypoints, and firmware. First, the firmware is built on Pixhawk, which is mounted on the drone [7]. The firmware in (c) of the figure controls the drone as programmed by referring to the parameters of devices, shown in (b), and the waypoints, shown in (a), that the drone should visit. We build the firmware source codes in a host computer, and port it onto Pixhawk. In addition, we generate the waypoints and set the parameters of the devices using a specific tool called Mission Planner. Mission Planner is a ground control station for Plane, Copter and Rover [8]. Developers can also utilize Mission Planner for setting the waypoints and sending the waypoint information to Pixhawk. Figure 3 shows a sample screen shot of Mission Planner.

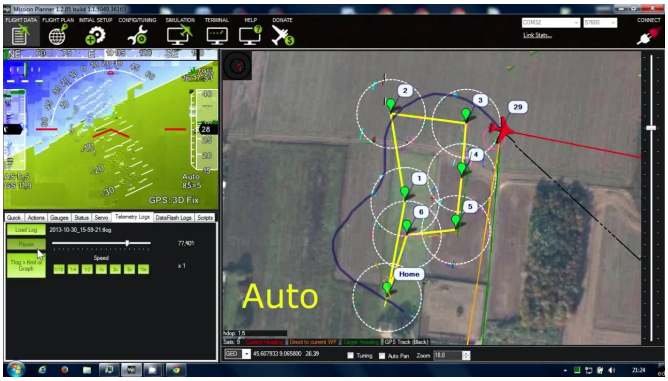

Figure 3. Mission Planner

Sections 3.2 and 3.3 analyze the relations between the firmware and its reference parameters. The analysis provides basic information for designing test cases for the failsafe behaviors of a drone.

B. Mitigation Settings and Execution Scenario

Mission Planner manages various parameters of every peripheral device used in a drone, such as the battery, GPS receiver, air speed, accelerometer, gyroscope, magnetometer, and barometer. We selected the battery-related parts because battery problems have been the direct causes of major crashes, as mentioned in Section 1. We narrowed down the battery-related settings to the failsafe option. Figure 4 shows a portion of the failsafe option setting screen of Mission Planner.

Figure 4. A portion of the failsafe options for battery use of Mission Planner

In Figure 4, the Mission Planner settings are “Low Battery,” “Reserved MAH,” and its mitigation action, which is “Land” in this case. These settings indicate that the drone should land if its battery level falls below 10.5 V. The “Land” setting is a mitigation action for a low-battery risk. Once this setting value is sent to Pixhawk, the firmware refers to the value of 10.5 V and the “Land” mitigation while executing its autopilot operation. The parameters shown in Figure 4 are a part of Figure 2 (b).

The firmware used is an open-source product released at pixhawk.org. It includes the working modules of each device, as well as the documentations, diagrams, and Make files. Of the modules used, the “Sensor” module handles battery-related parameters, manages risky situations, and makes a decision regarding the mitigation. Figure 5 shows a snippet of Sensor.cpp, which defines the important parameters and uses them to collect the battery levels and decide on the applied mitigation.

```
……

#define    BATT_V_LOWPASS     0.001f
….
void
Sensors::adc_poll(struct sensor_combined_s &raw)
{
        /* only read if publishing */
        if (!_publishing) {
                return;
        }

        hrt_abstime t = hrt_absolute_time();

        /* rate limit to 100 Hz */
        if (t - _last_adc >= 10000) {
                /* make space for a maximum of twelve channels (to ensure reading all channels at
once) */
                struct adc_msg_s buf_adc[12];
                /* read all channels available */
                int ret = read(_fd_adc, &buf_adc, sizeof(buf_adc));
…..
```

Figure 5. A code snippet of Sensor.cpp in firmware source codes.

Sensor.cpp defines BATT_V_LOWPASS, which is a filtering coefficient. In the real world, the battery level is very bouncing. For this reason, the battery level should be filtered to a certain acceptance ratio. The firmware sets the ratio to 0.001, which means that it accepts only 1/1000th of the battery level collected from its sensor. This value must be defined by a battery expert. In addition, the frequency used for reading the battery level from the sensor is set to 100 Hz, which means that it reads the battery level 100 times per second. In the code, the sentence, if (t - _last_adc >= 10000) handles the frequency. These two attributes, the filtering coefficient and frequency, are hard-coded into the firmware source codes.

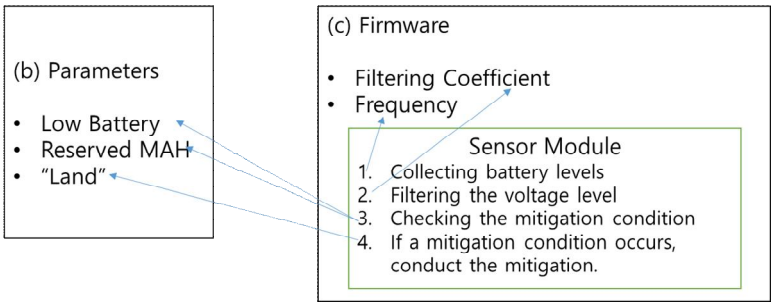

Figure 6. Scenario of parameter referencing.

In terms of mitigation for a battery-related risk, the firmware collects the battery levels based on the frequency defined in the firmware. Once the current battery level is collected, it should be recalculated by multiplying the filtering coefficient as defined in the firmware source codes. The firmware then compares the filtered level and Low Battery level set in Mission Planner. Actually, it should satisfy the Low Battery and Reserved MAH parameters concurrently. If the filtered level is below that of the Low Battery or Reserved MAH parameter, it executes a mitigation action assigned in Mission Planner. In the example, the "Land" mitigation was selected. Figure 6 shows the scenario and interaction of the firmware and parameters defined in Mission Planner. Figures 6 (b) and 6 (c) are the same as (b) and (c) of Figure 2 in Section 3.1.

IV. CONCLUSIONS

This research analyzed the parameter settings and method for operating an autopilot system by referring to the parameters of the firmware used. We narrowed down the parameters of various peripheral devices of a drone to those related to the battery: (a) "Low Battery," "Reserved MAH," and mitigation actions, which are set through a tool called Mission Planner. In addition, we described a couple of important variables hard-coded into the firmware: (b) "Filtering Coefficient" and "Frequency." These parameters are referenced and affect the drone's flying while running on autopilot. Both (a) and (b) have an influence on the choice of battery-related mitigation. In addition, (a) is open to users, who can set the parameters by inputting their preferred values into Mission Planner. However, (b) is hard-coded into the firmware source codes, which means that users cannot change the variables for "Filtering Coefficient" or "Frequency."

These parameters are all used to determine whether the drone should take a mitigation action, and so the referring relation of these parameters will be used in designing test cases for failsafe behavioural testing.

ACKNOWLEDGMENT

This research was supported by the Basic Science Research Program through the National Research Foundation of Korea (NRF), which is funded by the Ministry of Science, ICT & Future Planning (NRF-2014R1A1A3051827).

REFERENCES

[1] F. Bi, "Drone Maker 3D Robotics Raises $50 Million in Latest Round," Forbes, 2015
[2] C. Cole, "What 200 military drone crashes tells us about the drone wars," Drone Wars UK, Feb.2015
[3] F. Ye and T. Kelly, "Component failure mitigation according to failure type," in Proceedings of the 28th Annual International Computer Software and Applications Conference(COMPSAC 2004), pp. 258–264, 2004.
[4] Donmez, L. Boyle and J. D. Lee, "Taxonomy of mitigation strategies for driver distraction," in Proceedings of the Human Factors and Ergonomics Society Annual Meeting, vol. 47, no. 16. pp. 1865–1869, 2003.
[5] S. Subramanian, L. Elliott, R. Vishnuvajjala, W. Tsai, and R. Mojdehbakhsh, "Fault mitigation in safety-critical software systems," in Proceedings Ninth IEEE Symposium on Computer-Based Medical Systems, pp. 12–17, 1996.
[6] "Pixhawk Autopilot," http://pixhawk.org/modules/pixhawk
[7] "PX4 source code," https://pixhawk.org/firmware/source_code
[8] Mission Planner Home, http://planner.ardupilot.com/