Soft Error Assessment of UAV Control Algorithms Running in Resource-constrained Microprocessors

Alex Hanneman*, Jonas Gava†, Paulo Vancin‡, Aqsa Kk Kaim-Khani*, Sam Amiri*, Rafael Garibotti†,
Fernando Moraes‡, Ney Calazans†, Ricardo Reis†, Luciano Ost*
* Wolfson School, Loughborough University, United Kingdom
{*A.D.Hanneman, A.B.Kaim-Khani, S.Amiri, l.ost*}*@lboro.ac.uk*
† PGMicro/PPGC/UFRGS, Universidade Federal do Rio Grande do Sul, Porto Alegre, Brazil
{*jonas.gava, rfgaribotti, nlvcalazans, reis*}*@inf.ufrgs.br*
‡ Pontifical Catholic University of Rio Grande do Sul (PUCRS), School of Technology, Porto Alegre, Brazil
paulo.vancin@acad.pucrs.br, fernando.moraes@pucrs.br

***Abstract*—The landscape of Unmanned Aerial Vehicles (UAVs) or drones technology and their applications is increasingly evolving across various sectors. UAVs are subject to radiation-induced single event upset (SEUs); assessing their vulnerability is thus vital to avoid possible catastrophic events. This paper's original contribution is to assess the SEU vulnerability of four UAV control algorithms running in a resource-constrained microprocessor. Results suggest the PID for attitude and height control has the best relative trade-off between SEU vulnerability, memory-saving, performance, and power efficiency. Results also show that a self-tuning PID controller using fuzzy theory presented fewer silent data corruption (SDCs) than all other algorithms.**

I. Introduction

The versatility of Unmanned Aerial Vehicles (UAVs) aligned with technological advancements is driving innovations across several sectors, including agriculture, healthcare, and security, among others [1]. UAVs, also known as drones, differ in size, weight, flight autonomy (e.g., decision-making and endurance capabilities), components, and even the regulations such systems are subject to. The selection of a UAVs' components (e.g., battery, processor) depends on several factors (e.g., altitude and area of operation, application requirements), and the most optimal configuration will likely vary depending on the unique use case and mission goals.

UAV technology is evolving fast, enabling it to expand its application spectrum. For instance, high-altitude drones are a reality that becomes achievable through significant advancements in electronic components miniaturisation, autonomous navigation, and even solar power technologies, enabling long-distance flights at a lower environmental impact [2]. The possibility of drones flying in the stratosphere brings notable challenges, such as minimising the impact of single event upset (SEUs) on their operation. For instance, the occurrence of SEUs in navigational sensors or during the execution of attitude estimation (AE) algorithms can lead to dangerous situations, since underlying systems share airspace with civil air traffic. The resulting scenario calls for sophisticated diagnosis methods for faults in the UAV hardware components (e.g., sensors and actuators) and the adopted software stack.

To minimise the effect of SEUs on drone performance, an accurate assessment of the system susceptibility to SEUs must be made; such an assessment can then be used to adjust a given mitigation strategy to strike an acceptable balance between reliability, memory-usage, power, and performance. This paper contributes by assessing the impact of radiation-induced soft errors on the behaviour of four control algorithms running in a resource-constrained microprocessor that is present in several drone platforms. To ensure a meaningful and statistically significant assessment, this work uses a multi-board debug-based fault injection (FI) platform to quantify the SEU reliability of the selected control algorithms. Gathered results suggest that the PID algorithm for attitude and height control has the best relative trade-off between SEU vulnerability, memory-saving, performance, and power efficiency.

This work was partially funded by: UK EPSRC (EP/R513088/1) and CNPq (grants no. 407477/2022-5, 311587/2022-4 and 309605/2020-2).

II. Related Works in SEU Vulnerability of UAVs

With the advance of high-altitude drones, authors started to investigate the impact of radiation-induced soft errors on different UAV software stack components [3]–[8], as summarised in Table I. Authors have exclusively concentrated on the SEU vulnerability of attitude estimation (AE) and control algorithms. Except for prior work [8], others employ either customised processor model architectures [3], [5] or the execution of AE algorithms in the Arm Cortex-A72 processor [4], [6], [7]. While authors from [3], [5] consider satellite applications, the works described in [4], [6], [7] assess the soft error reliability of AE algorithms. Most works use well-known Kalman filter variations, such as the NQKF ([4], [6], [7]), EKF ([7]), IKF ([7]), and the versions in [3] (i.e., CKF, FKF, FTFKF). In [5], Li et al. use simulation to evaluate a custom solution for fault-tolerant AE in nanosatellites. In turn, results obtained from neutron radiation experiments show the susceptibility of AE algorithms to SEUs when executed in Arm Cortex-A72 [6], [7], and Cortex-M microprocessors [8]. This work distinguishes itself from previous works essentially in three aspects:

- It is the first work to assess the SEU reliability of UAV control algorithms in a resource-bound microprocessor.
- It explores the relative performance, memory utilisation, power consumption and reliability trade-offs of four control algorithms running in the selected microprocessor.
- It also considers the impact of compiler optimization flags (i.e.,*-O0, -O3*), which is altogether ignored in all other works listed in Table I.

TABLE I: Related works in the soft error reliability assessment of UAV attitude estimation (AE) and control algorithms.

Work	Target Algorithm	Assessment Approach	Processor	Trade-off Analysis			
				Performance	Power	Mem. Usage	Reliability
Zhou *et al.* [3] (2016)	**AE**: CKF, FKF, FTFKF**	Simulation-based	Customised architecture	✓			✓
Brum *et al.* [4] (2021)	**AE**: NQKF, On-Chip Execution	Emulation-based	Arm Cortex-A72				✓
Li *et al.* [5] (2022)	**AE**: Distributed fusion nanosatellite FTAE***	Simulation-based	Customised architecture	✓			✓
Sartori *et al.* [6] (2022)	**AE**: NQKF	Neutron irradiation	Arm Cortex-A72				✓
Sartori *et al.* [7] (2023)	**AE**: EKF, IKF, NQKF, gradient	Neutron irradiation	Arm Cortex-A72				✓
Gava *et al.* [8] (2024)	**AE**: EKF, IKF, NQKF	Neutron irradiation	Arm Cortex-M4 & M7	✓			✓
ThisWork	**Control**: PID, PID with GPS, PID + Fuzzy, LQR	Emulation-based	Arm Cortex-M7	✓	✓	✓	✓

**CKF: centralized Kalman filter, FKF: federated Kalman filter, FTFKF: fault-tolerant FKF
***FTAE: fault-tolerant attitude estimation

III. Selected Control Algorithms

Four control algorithms often adopted in UAVs are considered here. These rely on two of the most used linear control methods: the Proportional-Integral-Derivative (PID) control and the Linear Quadratic Regulator (LQR) control.

The **PID control** is a control loop feedback mechanism that directly adjusts control values with a closed-form formula based on proportional, integral and derivative gains (resp. K_p, K_i, K_d) [9]. It can be computed as defined by Zulu and John in [10], based on the measured error signal ($e(t)$):

$$u_c(t) = K_p * e(t) + K_i * \int^{\tau} e(t)dt + K_d * \frac{de}{dt} \quad (1)$$

The **LQR control** is an optimal method that operates a dynamic system at a minimum cost [11]. A quadratic function defines a linear differential equation representing the system's cost. This cost function is minimised to provide the best control signal. The function is formulated by Equation 2.

$$J = \int_{t^0}^{\infty} (x^T Q x + u^T R u)dt. \quad (2)$$

The LQR control in fact relies on the Q and R matrices. Carefully selecting these is paramount to derive optimal feedback gain values aiming to improve the performance of the target system [12].

A. The Four Control Algorithms

Three of the algorithms are based on the PID control method and one on the LQR control method, as described next.

1- LQR for Attitude and Height Control: This LQR implementation, is designed to exclusively control the quadrotor attitude and height. LQR is more complex than PID control to implement, but the former deals better with disturbances without requiring an external controller (see the fourth algorithm).

2- PID for Attitude and Height Control: this algorithm is used to run a quadrotor with a stabilised attitude state (i.e.,roll, pitch and yaw at $0°$) and at a predefined height, using a simple PID controller.

3- Cascaded PID for XY Position Control: being an under-actuated system (i.e., having more controllable states than actuators), the quadrotor needs an alternative strategy for controlling the attitude, height, and XY position. In this implementation the outer-loop PID controls the XY position and the inner-loop PID regulates the UAV attitude and height.

4- Fuzzy PID Controller for XY Position Control and Gain Optimisation: This is a fuzzy controller, developed based on a set of predefined rules, which controls the PID gains while the system is operating. Although it adds another complexity layer to the processing, it compensates for attenuating possible external disturbance effects.

IV. Soft Error Assessment Methodology

A. Developed Multi-board FI Platform

This work uses the Debate-FI [13], a multi-board debug-based fault injection (FI) platform to assess the SEU reliability of the developed control algorithms. The developed multi-board platform, depicted in Figure 1 includes FI targeting functionality that relies on a function lifespan technique to generate fault lists from a simulation profile. This reduces the FI spectrum, by limiting the insertion time to those small intervals where the target control algorithm is active. The debug-based FI platform is then used to assess the outcomes of the generated fault scenarios.

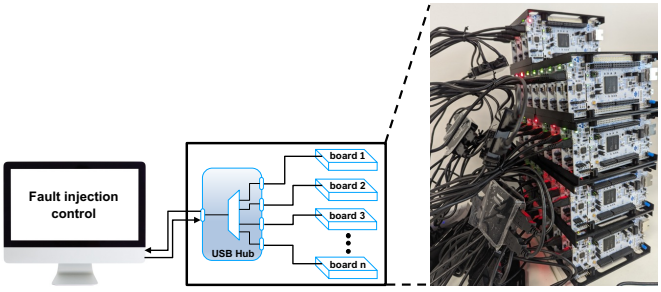

Fig. 1: Multi-board FI platform.

B. Adopted Assessment Flow

The adopted semi-automated flow comprises seven steps. In the first step, **(1) Software Stack Configuration**, the software engineer configures the software stack and selects the cross-compiler based on the target platform. Then, in step **(2) Cross-Compilation** the application's source files are compiled to the target platform considering the parameters previously defined. This step generates the simulation binary version and the board binary version of the application. In the third step, **(3) Function Profile**, the SOFIA framework [14] is used to simulate and verify the correctness execution of the UAV's software stack. Then, a profiling tool with the trace flag enabled captures function call information between the FI points (e.g., begin and end). Next, the **(4) Register Profile** step extracts registers' information of each executed instruction for

each function call, which is then used to generate the final fault list with precise target points to FI. The **(5) Fault Injection Configuration** step consists of the fault injector application setup. The input data files to be sent to the test nodes are specified, together with the test format. The AE algorithms are programmed on the board and the fault list is selected. In the next stage, **(6) Fault Injection Control**, an FI scenario and input dataset are allocated to the board and run according to the parameters set in the prior step. During each test, serial communications received from the boards are logged along with register dumps at the end of the FI zone. Each test consists of two runs, the first one runs without injecting a fault and provides a golden reference to compare with the second run, which uses the specified FI scenario to run to a predetermined injection point; after this, it runs until the end of the program. In stage **(7) Results Collection and Analysis** the serial log files generated from the FI campaigns are parsed; for each scenario, the outputs of the FI run are checked to verify that the outcomes were obtained; otherwise, this is recorded as a crash. If FI results are present, then these are compared against the reference run results, both the outputs of the algorithm and the internal states at the end of the algorithm execution; if there is a mismatch, these are classified as an SDC, otherwise, it is considered masked.

C. Fault Classification and Reliability Metrics

The obtained soft error results are classified as follows: *Masked:* the injected bit-flip did not affect the algorithm execution, and the output is correct; *Silent Data Corruption (SDC)*, the algorithm terminates without any error indication, but the output is affected; *Crash:* the algorithm suffers from abnormal termination or hangs.

This work uses the Mean Work to Failure (MWTF) metric, which measures the tradeoff between soft error reliability improvement and runtime overhead. The Architectural Vulnerability Factor (AVF) quantifies the likelihood of a visible system error (e.g., SDC) occurring when a storage cell experiences a bit-flip, and it is calculated using fault injection emulation. The normalised MWTF is used to improve the comparison between all scenarios. Equation (3) shows how this is calculated.

$$MWTF = \frac{1}{AVF_{SDC} * execution\ time} \tag{3}$$

V. Results

A. Experimental Setup

Table II shows the adopted experimental setup. Soft error reliability results are obtained by injecting random bit-flips into the general-purpose register file (r0-r15) or on special control registers (sp, lr, pc) of an Arm Cortex-M7 processor. To ensure the results' statistical significance, this work injects 1k faults per campaign, generating a margin of error of 5% with a 99.8% confidence level, following Leveugle *et al.* [15].

TABLE II: FI Experimental Setup

Processor (clock MHz)	Arm Cortex-M7 (216MHz)
Control Algorithms	**A1**-LQR, **A2**-PID for attitude, **A3**-PID for position, **A4**-PID + Fuzzy,
OS, Compiler, and opt. flag	Bare metal, GCC 9.3.1 with $-O0, -O3$
Target FI	General-purpose registers
FI per campaign	1k

B. Soft Error Reliability Analysis

This Section considers the soft error classification (i.e., SDC and Crash) and the MTWF metric defined in Section 4.3. When considering the soft error vulnerability of the algorithms, two categories of soft error outcome are considered: SDC, where the outputs or the internal state of the algorithm are changed from the expected values, and crashes, where the system fails to complete the execution of the algorithm. There is a significant variation in soft error performance across the algorithms, as can be seen in columns 5-7 of Table III; there are also some notable trends that occur when considering the unoptimised (O0) and optimised (O3) variants of the algorithms' implementation.

Firstly, the LQR algorithm, in either optimised and unoptimised forms, is the poorest implementation from a soft error vulnerability perspective, considering both SDCs and crash outcomes. This is in part due to it having the longest execution time in both levels of optimisation of any of the algorithms, significantly longer than any other for the O3 variants. In addition to this, it also has the highest absolute number of crashes of any of the algorithms, and these combine to give it an exceptionally poor $MWTF_{crash}$. Considering SDCs, the LQR algorithm fares marginally better in absolute terms, with the number of SDCs for the O3 variant being notably low; however, considering the significantly higher execution time for the algorithm, it still gives a poor $MWTF_{SDC}$.

The PID attitude and PID position control algorithms show very similar performances in terms of soft error susceptibility. Both give high resilience to crash outcomes w.r.t. the LQR solution, especially the highly optimised versions and increased resilience to SDC outcomes, with the unoptimised version performing best for this classification. These algorithms perform similarly in terms of absolute numbers of soft error outcomes and similar runtimes, suggesting a similar susceptibility profile, which is to be expected as these, although serving different functions, are the most similar in implementation. The Fuzzy PID controller displays a more interesting pattern. In absolute terms, it performs significantly worse, moving from the unoptimised version to the optimised one for both SDC and crash outcomes, especially so with SDCs. The reduced execution time offsets the absolute increase somewhat but still results in the $MWTF_{SDC}$ falling from a fairly high relative value of $25.83\times$ for the unoptimised version to a poor $2.97\times$; the $MWTF_{crash}$ does increase with optimisation, but not nearly as drastically.

Figure 2 shows the number of executed instructions (red dots associated with the right y-axis) and the instruction type for each algorithm and optimisation level represented by the stacked bars (left y-axis). The instruction types are classified in *Memory access:* load/store, *Control:* branches, and *Data processing:* arithmetic and logic operations (e.g., add, mul, div, or) and mov instructions. The optimisation level has a significant effect on the soft error resiliency of the system. A universal trend, in terms of the MWTF metric,

TABLE III: Summary of the emulation test results for the evaluated UAV control algorithms.

Algorithm	Optimisation Level	Runtime* [μs]	Energy* [μJ]	Events (MWTF**) Masked	SDC	Crash	Memory Usage [kB] RAM	Flash
A1 - LQR for attitude & heigh control	O0	190.59	58.89	725	33 (1.49)	242 (1.00)	25.2	36.2
	O3	118.46	32.33	554	79 (1.00)	367 (1.06)	25.2	31.2
A2 - PID for attitude & heigh control	O0	7.77	2.33	805	35 (34.43)	160 (37.12)	25.2	35.4
	O3	2.59	0.84	536	137 (26.38)	223 (79.89)	25.2	30.3
A3 - PID for position control	O0	9.36	2.65	807	30 (33.31)	162 (30.41)	25.2	35.9
	O3	2.73	0.87	574	138 (24.88)	211 (80.19)	25.2	30.0
A4 - PID + fuzzy for attitude control	O0	181.18	54.48	860	2 (25.83)	128 (1.99)	25.3	42.8
	O3	41.51	12.33	676	76 (2.97)	186 (5.97)	25.3	33.9

*Runtime and energy values are the average of 5001 executions.
**The Mean Work to Failure metric (the higher the better) is normalised by the lowest value between all algorithms.

is that increased optimisation increases the likelihood of SDC outcomes while reducing the chances of a crash outcome, i.e., a higher $MWTF_{crash}$. An explanation for this effect can be derived from Figure 2. Here it can be seen that for all of the algorithms there is a significant decrease in the proportion of memory access instructions from the unoptimised to the optimised implementations. Of all the algorithms, the LQR has the least proportional reduction in memory access operations and this partially explains its poor decrease in runtime.

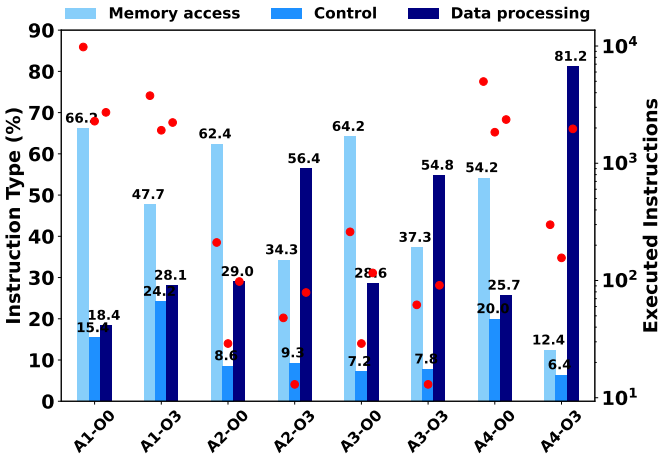

Fig. 2: Dynamic Instructions proportion (bars) and total number (red dots).

Figure 3 shows the effect of optimisation on the susceptibility of the PID for attitude and height control algorithm per register. For the unoptimised version, the vulnerability was generally confined to the special control registers, in particular for crashes and the lower register file registers, which mainly resulted in SDC outcomes. For the O3 algorithm, the vulnerability of the registers was more equally distributed, with most registers showing some level of susceptibility.

Considering the data processing instructions, these increase proportionally from O0 to O3, again for all algorithms. The trend can explain the increase in $MWTF_{SDC}$ from O0 to O3, being intuitive that an increase in the proportion of data processing operations is likely to result in more susceptibility in the data flow of the program. Except for PID+Fuzzy, the proportion of control instructions also increases with O3, which may lead to a higher number of crash events. Even with lower control instructions, the PID+Fuzzy algorithm executes a significant number of *compare* and *select* instructions to guide the control flow. However, as can be seen by the absolute number of masked events between the optimised and unoptimised versions of all algorithms, while the optimisation increases the criticality of the program in absolute terms, the reduced execution time offered by the higher optimisation, offsets this, by reducing the chance that the program will be running at the point when a radiation strike occurs.

C. Power-efficiency Analysis

The flight duration of a UAV is an interplay of multiple factors, including weather conditions (e.g., wind), flight movement and speed, hardware components (e.g., battery capacity), and the adopted software stack (e.g., communication protocols, flight controller firmware, etc.) [16]. Well-tuned control algorithms, for instance, can effectively contribute to energy savings and improve UAV's endurance. This work uses the STLINK-V3PWR [17], a source measurement unit (SMU), to quantify the power consumption of the four control algorithms running in the Cortex-M7 microprocessor. The STM32CubeMonitor power acquisition configuration was set to consider a current pulse of 50 μs (i.e., 5 samples per pulse), a current threshold of 1000 μA, sampling frequency of 20 kHz, and an input voltage of 3.3V. To ensure a comprehensive and equitable comparison, the power acquisition time considers 5001 executions of each control algorithm. Thus, the power cost of board programming, initialisation, and the use of on-board debug functions are not relevant. Figure 4 shows the power acquisition waveform of the developed Fuzzy PID controller for XY position control and gain optimisation.

The fourth column in Table III shows the average energy consumption of the four algorithms considering the two compilation flags. It is evident from the results that PID-based algorithms surpass the LQR alternative in power efficiency.

D. Trade-off Analysis

This Section provides a general trade-off analysis considering all data collected for each algorithm version. Table III includes SEU vulnerability ($MWTF_{sdc}$, and $MWTF_{crash}$), memory-saving, performance, and power efficiency for the four control algorithms considering two optimisation flags (i.e., O0 and O3), and using values ranging from 0 to 5

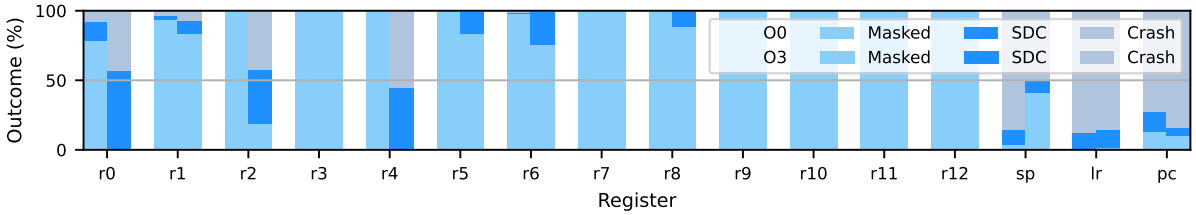

Fig. 3: PID for attitude & height control outcome for each injected register with O0 and O3 optimisation levels.

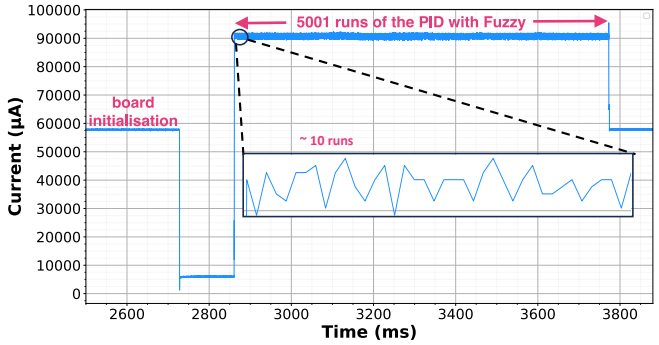

Fig. 4: Power acquisition example considering 5001 PID with Fuzzy executions when compiled with the O0 flag.

normalised by the highest value. As expected, O3 significantly improved the performance and, consequently, the power-saving. However, results are very different when comparing the LQR/PID+Fuzzy with the other two PID algorithms. LQR has some performance issues, mainly due to the reasonable use of external libraries that can not be optimised, such as memcpy, showing a high number of memory operations (i.e., load/store instructions) that consume more power and runtime. The Fuzzy PID controller holds the more complex implementation; but it provides the most remarkable performance/power improvement (i.e., about $4.4\times$) with the massive use of vector instructions when compiled with O3. The other two PID algorithms are simpler, showing similar results with slightly better performance and power-saving for the PID for attitude and height control. All algorithms present similar memory usage, except for the Fuzzy PID controller, which possesses the largest code implementation – values range from 55.2kB to 68.1kB (i.e., a 23% increase). For O3, the memory usage difference drops to 13% (i.e., 30 to 33.9kB). Regarding $MWTF_{sdc}$, even with fewer SDCs for O0, the PID for attitude and height control with O3 has the best trade-off overall. The PID for position control presented similar results but slightly worse ones, even considering the error margin for SDC events (i.e., 30 ± 1.5 against 35 ± 1.75). In terms of $MWTF_{crash}$, the PID for attitude and height control and PID for position control with O3 presented the best trade-off, and both algorithm results can be considered equal when taking into account the error margin for crash events (i.e., 223 ± 11.15 against 211 ± 10.55).

VI. Conclusion and Future Works

This paper conducted a detailed trade-off assessment between soft error resilience, power-saving, performance, and memory-saving for four UAV control algorithms running on a resource-constrained microprocessor under several fault injection scenarios. Results suggest that the PID for attitude and height control with O3 optimisation level has the best trade-off in general for the $MWTF_{sdc}$ and $MWTF_{crash}$. However, if performance and energy efficiency limitations are not an issue, a lower optimisation level could be ideal to minimise the occurrence of SDC events, which can lead to critical failures.

Future works include: (i) assessing the soft error reliability of fault-tolerant and reinforcement learning-based controllers; (ii) conducting neutron and atmospheric spectrum experiments considering the adopted flight control algorithms.

References

[1] R. Clarke *et al.*, "The regulation of civilian drones' impacts on public safety," *Computer Law & Security Review*, vol. 30, no. 3, pp. 263–285, 2014.

[2] S. Evans, *Solar-Powered Drone Reaches Stratosphere*, 2023. [Online]. Available: https://www.iotworldtoday.com/security/solar-powered-drone-reaches-stratosphere.

[3] J. Zhou *et al.*, "A scheme of satellite multi-sensor fault-tolerant attitude estimation," *Transactions of the Institute of Measurement and Control*, vol. 38, no. 9, pp. 1053–1063, 2016.

[4] J. P. Brum *et al.*, "Evaluation of Attitude Estimation Algorithm under Soft Error Effects," in *IEEE LATS*, 2021, pp. 110–114.

[5] L. Li *et al.*, "Distributed fusion fault-tolerant attitude estimation scheme for nanosatellite using commercial off-the-shelf sensors," *Advances in Space Research*, vol. 70, no. 11, pp. 3540–3551, 2022.

[6] T. K. S. Sartori *et al.*, "Assessment of Radiation Effects on Attitude Estimation Processing for Autonomous Things," *IEEE Transactions on Nuclear Science*, vol. 69, no. 7, pp. 1610–1617, 2022.

[7] T. K. S. Sartori *et al.*, "Effectiveness of Attitude Estimation Processing Approaches in Tolerating Radiation Soft Errors," *IEEE Transactions on Nuclear Science*, vol. 70, no. 8, pp. 1658–1665, 2023.

[8] J. Gava *et al.*, "Soft error assessment of attitude estimation algorithms running on resource-constrained devices under neutron radiation," *IEEE Transactions on Nuclear Science*, pp. 1–1, 2024.

[9] T. Xiao, *A Literature Review of Learning and Optimization Methods Applied to Quadrotor Control*, 2016. [Online]. Available: https://tedxiao.me/pdf/literature_review.pdf.

[10] A. Zulu *et al.*, "A Review of Control Algorithms for Autonomous Quadrotors," *Open Journal of Applied Sciences*, vol. 4, no. 14, pp. 547–556, 2014.

[11] S. Khatoon *et al.*, "PID & LQR control for a quadrotor: Modeling and simulation," in *IEEE ICACCI*, 2014, pp. 796–802.

[12] O. A. Dhewa *et al.*, "Model of Linear Quadratic Regulator (LQR) Control Method in Hovering State of Quadrotor," *Journal of Telecommunication, Electronic and Computer Engineering (JTEC)*, vol. 9, no. 3, pp. 135–143, 2017.

[13] A. Hanneman *et al.*, "DeBaTE-FI: A Debugger-Based Fault Injector Infrastructure for IoT Soft Error Reliability Assessment," in *IEEE World Forum on Internet of Things (WF-IoT)*, 2023.

[14] J. Gava *et al.*, "SOFIA: An automated framework for early soft error assessment, identification, and mitigation," *Journal of Systems Architecture*, vol. 131, p. 102 710, 2022.

[15] R. Leveugle *et al.*, "Statistical Fault Injection: Quantified Error and Confidence," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2009, pp. 502–506.

[16] H. V. Abeywickrama *et al.*, "Comprehensive Energy Consumption Model for Unmanned Aerial Vehicles, Based on Empirical Studies of Battery Performance," *IEEE Access*, vol. 6, pp. 58 383–58 394, 2018.

[17] STMicroelectronics, *Source measurement unit (SMU) and debugger/programmer for STM32 microcontrollers*, 2023. [Online]. Available: https://www.st.com/resource/en/data_brief/stlink-v3pwr.pdf.