A Simplex Architecture for Intelligent and Safe Unmanned Aerial Vehicles

Prasanth Vivekanandan†, Gonzalo Garcia⋆, Heechul Yun†, Shawn Keshmiri⋆
† Electrical Engineering & Computer Science
⋆ Aerospace Engineering
University of Kansas
{vivekanandan, gagarcia, heechul.yun, keshmiri}@ku.edu

***Abstract*—Unmanned Aerial Vehicles (UAVs) are increasingly demanded in civil, military and research purposes. However, they also possess serious threats to the society because faults in UAVs can lead to physical damage or even loss of life. While increasing their intelligence, for example, adding vision-based sense-and-avoid capability, has a potential to reduce the safety threats, increased software complexity and the need for higher computing performance create additional challenges—software bugs and transient hardware faults—that must be addressed to realize intelligent and safe UAV systems.**

In this paper, we present a fault tolerant system design for UAVs. Our proposal is to use two heterogeneous hardware and software platforms with distinct reliability and performance characteristics: High-Assurance (HA) and High-Performance (HP) platforms. The HA platform focuses on simplicity and verifiability in software and uses a simple and transient fault tolerant processor, while the HP platform focuses on intelligence and functionality in software and uses a complex and high-performance processor. During the normal operation, the HP platform is responsible for controlling the UAV. However, if it fails due to transient hardware faults or software bugs, the HA platform will take over until the HP platform recovers.

We have implemented the proposed design on an actual UAV using a low-cost Arduino and a high-performance Tegra TK1 multicore platform. Our case-studies show that our design can improve safety without compromising performance and intelligence of the UAV.

I. Introduction

The use of Unmanned Aerial Vehicles (UAVs) is rapidly increasing in recent years due to diverse recreational, commercial, and military applications. However, UAVs also pose serious threats to the society because they can cause physical damage or even loss of life. Therefore, there is an increasing demand for intelligent UAV systems that are cognizant of the surrounding environment and perform sophisticated tasks including collision avoidance.

An intelligent UAV system requires integration of advanced sensor packages (e.g. vision) and high computational performance to process the enormous amount of real-time sensor data and to execute complex algorithms in a timely manner. The rapidly increasing computing capacity of modern embedded computing platforms—multiple CPU cores, GPU, and other accelerators—makes it feasible to develop such a UAV system while satisfying size, weight, and power (SWaP) requirements of UAVs.

However, the push for higher intelligence in UAV systems also creates serious side effects in terms of safety and reliability. First, the complexity of software systems is rapidly increasing, which makes it difficult to weed out software bugs. For example, an intelligent flight control system with vision based collision avoidance capability, which in itself can be complex and difficult to verify, may also depend on complex middleware packages (e.g., Robot Operating System, ROS [24]) and the OS (e.g., Linux), each of which may be comprised of multi-million lines of code.

Fig. 1: DG 808S with our custom built avionics

Second, to achieve high intelligence, the use of high-performance computing platforms is necessary. However, high performance computing platforms are increasingly prone to transient hardware faults (soft errors) due to environmental effects such as single-event upsets (SEUs) [7]. SEUs are caused by high energy particle strikes from cosmic rays [29] which result in bit flips. The technology treads to develop efficient and high-performance processors—shrinking dimensions and operating voltage, and increased frequency and density—have dramatically increased the possibilities of SEUs [7], which could result in unexpected failures in the system [15], [28].

There has been a large body of research in the control systems community regarding the design of fault tolerant control of UAV systems [26]. While these fault tolerant controllers are designed to handle structural damage, actuator and sensor failures, they typically do not handle system-level failure such as on-board computing platform malfunction, which prevents execution of the control algorithms in the first place. System-level reliability can be generally improved by redundancy. A well-known technique is triple modular redundancy (TMR) [18] in which three identical systems produce control outputs in order to survive from failures of any one of the system. However, the size, weight, and power

considerations as well as the cost make the TMR solution undesirable, especially in small UAVs.

To address the safety and reliability challenges of UAV systems, we present a UAV system design and implementation based on Simplex architecture [27]. The main idea of the Simplex architecture is that a simple verifiable controller provides safety of the system, while a complex, high-performance controller strives to achieve high system performance. The choice between the two controllers is determined by a decision module, which constantly assesses the safety status of the system and makes the decision.

Our contribution is a novel application of the Simplex architecture to develop both intelligent and reliable UAV systems in a cost effective manner. Specifically, we realize the Simplex architecture using two heterogeneous hardware platforms with distinct reliability and performance characteristics. The idea is that we use a reliable but less performant hardware platform, which we call a High-Assurance (HA) platform, to be responsible for safety while we use a more performant, but potentially less reliable platform, which we call a High-Performance (HP) platform, for performance and intelligence of the UAV. Only the HA platform—both software and hardware—forms the trusted computing base (TCB) of the UAV. The HP platform, on the other hand, can be affected by software bugs and SEUs, which could result in failures (e.g., crashes), but the whole system safety will still be insured by the safety platform.

We applied the proposed design in implementing a custom fixed-wing UAV system, shown in Figure 1, using a low-cost Arduino as our HA platform and a Tegra TK1 as HP platform. We demonstrate the ability to recover from crashes through a set of fault-injection experiments in a hardware-in-the-loop simulation setup. The results suggest that the proposed design can substantially increase safety of intelligent UAVs.

The rest of the paper is organized as follows. Section II discusses various fault types in a UAV and related background. Section III presents the proposed UAV simplex architecture. Section IV describes our implementation of a fixed-wing UAV based on the proposed architecture. Section V presents our evaluation results. We discuss related work in Section VI and conclude in Section VII.

II. Background

In this section, we discuss various types of faults that can occur in a UAV and provide some necessary background.

An UAV is a cyber-physical system, which includes cyber part (computing hardware and software) and physical part (sensors, actuators, UAV frame, and etc). Faults in physical components of an UAV and their handling has been well studied in the control and aerospace communities. For example, structural damage or bias in sensor readings can be tolerated by advanced adaptive control algorithms that take such effects into account. A comprehensive review on the topic can be found in [26]. However, advanced control algorithms must be realized in software binaries running on computing platforms—i.e., the cyber system.

Faults in the cyber system can arise from a number of different reasons. The most common type of faults are *logical faults* in software—i.e., software bugs. As the demand for higher intelligence and functionality increase, the size and complexity of flight control software keeps increasing [20]. While model-based design and verification methodologies have made great progress [13], [12], [8], it is still difficult to completely weed out all bugs.

Another type of faults are *temporal faults*, which occur when the real-time requirements (i.e., deadlines) of various tasks in the system are not met. Temporal correctness is difficult to guarantee especially in multicore architecture because of uncontrolled sharing of many performance critical hardware resources—such as cache and DRAM—among concurrently executing tasks can cause highly variable timing [17], [22].

Third, often overlooked but increasingly important type of faults are *transient hardware faults* due to single event upsets (SEUs). SEUs are caused by cosmic radiation and alpha particles [29], [19] resulting bit flips. Because SEUs can occur anywhere in SRAM, registers, and combinational logic, it can cause, for example, an invalid instruction exception, a parity error, a memory access violation, a wrong conditional branch, and ultimately a system crash [28]. As technology scaling continues (i.e., more transistors in a chip), sensitivity to radiation has dramatically increased [7]. Traditional circuit-level solutions—special circuit design and manufacturing process—are not only expensive but also often lag several generations behind the state-of-the-art processors [10]. This is a serious impediment to develop intelligent UAVs that require cutting edge high performance computing capabilities.

In summary, as the demand for higher intelligence in UAVs increases, the importance of reliable computing software and hardware platform—the cyber system—increases. However, the increased software complexity and the use of high-performance multicore processors, while necessary, would increase all three types of faults—logical, temporal, and transient hardware faults—of the cyber system of an UAV.

III. UAV Simplex Architecture

In this section, we review the Simplex architecture [27] and describe our two heterogeneous platforms based approach.

The Simplex architecture is composed of three components: a safety controller, a performance controller, and a decision logic [27]. Normally, the performance controller drives the plant (in our case, the UAV) as it offers higher control performance. However, if the safety conditions of the plant are to be violated, as determined by the decision logic, the safety controller will assume the control of the plant until the performance controller is recovered (for example, by restarting it). In this way, faults in the performance controller does not cause safety failure of the system.

In the original Simplex applications, however, all three components share the same computing hardware (processor) and software platform (OS, middleware) [27]. This means that system-level faults in the shared hardware and software platform—SEUs in the processor, bugs in the OS and middleware—could still compromise the safety of the system.

To overcome the limitations, we propose to realize the Simplex architecture by using two platforms with distinct reliability and performance characteristics, as shown in Table I. The High-Assurance (HA) platform focuses on safety and verifiability over performance and functionality. For hardware,

TABLE I: Platform characteristics

Platform	High-Assurance (HA)	High-Performance (HP)
Hardware	SEU resistant	SEU susceptible
Software	Verifiable	Unverifiable

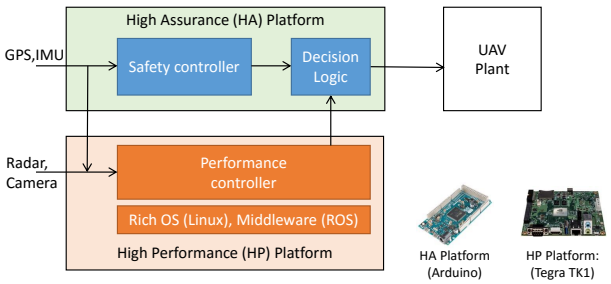

Fig. 2: UAV simplex architecture

we assume that it uses chips that are more tolerant to SEUs. While a number of techniques can be used to reduce the SEU rate of a chip—ECC, manufacturing process, and etc.—using simple, low-density chips running at low operating frequency could also help reduce the overall SEU rate [7]. For software, we assume that the platform uses a small RTOS with proven (verified) reliability. For example, the seL4 micro-kernel's functional correctness was formally verified [16]. Also, there are many other commercial/open-source RTOSs that have been used in critical applications.

On the other hand, the High-Performance (HP) platform focuses on performance and functionality over safety. For hardware, it may use a complex, high-performance (multicore) processor, which runs at multi-gigahertz frequencies and is composed of multi-billion transistors. Generally, such a processor suffers more SEUs than a simpler, low-performance one because higher density and operating frequency negatively affect SEUs [7]. For software, it may use a rich OS (e.g., Linux) and middleware solutions (e.g., Robot Operating System: ROS) to offer sophisticated capabilities needed to implement high intelligence. However, each of the software package may be comprised of many million lines of code, which makes it very difficult, if not impossible, to verify its correctness.

Among the three components in the Simplex architecture, we assign the safety controller and the decision logic on the HA platform, while assign the performance controller on the HP platform as shown in Figure 2.

The performance controller may implement an advanced control algorithm and intelligent sensing capabilities, such as vision-based collision avoidance, that require high computing performance and rich OS and middleware support. The HP platform offers such computing power and functionality but it does not guarantee safety and reliability.

On the other hand, the safety controller implements a simple and proven control algorithm that is always ready to take over the performance controller, if necessary, as determined by the decision logic. In our design, as in the original Simplex architecture, correct functioning of the safety controller and the decision logic is required. In addition, the HA platform provides a safe and reliable execution environment although it may not provide high computing performance or rich middleware support.

A. Fault Model

We assume that faults can occur only in the HP platform. In other words, we trust the correct functioning of the HA platform—the safety controller and decision logic. To ensure this, hardware of the HA platform must be resistant to transient hardware faults, and the complexity of its software must be limited to a level that can be rigorously tested and possibly verified.

On the other hand, we do not trust both hardware and software of the HP platform—i.e., the OS can crash, the performance controller may crash or produce invalid outputs (e.g., NaN output) or simply miss the deadline due to resource contention. Also, the HP platform can suffer transient hardware faults (SEUs) which lead to system crash or application failure.

Lastly, in this paper, we do not consider physical faults, such as structural damage, bias in sensor readings, sensor and actuator malfunction, and so on. We assume that these physical issues are tolerated by adaptive control algorithms, which have been extensively studied in the aerospace engineering community [26].

IV. Implementation

In this section, we detail the hardware and software architecture of the flight system for a DG808S UAV (See Figure 1).

A. Hardware

1) Sensors: We categorize sensors into two groups: basic and advanced sensors. Basic sensors include GPS/IMU, airspeed, and pressure sensors and they are essential to flight. For GPS/IMU, we use a VectorNav VN-200 [4] module. For airspeed and pressure, we use an AMS 5812 pressure sensor and a pitot tube. The basic sensors are shared by both safety and performance controllers. On the other hand, advanced sensors are only used by the performance controller and considered not essential to flight although they may increase performance of the system. The advanced sensors include, for example, radars and cameras, which can be used to implement advanced sense-and-avoid capabilities. We are currently implementing a vision and radar based sense-and-avoid system.

2) High-Performance (HP) Platform: The HP platform is responsible for real-time processing of advanced sensors (e.g., vision and radar) and sophisticated control algorithms to achieve high control performance and sense-and-avoid capabilities. To satisfy the performance demand, we use Nvidia's Tegra K1 processor, which equips four ARM Cortex-A15 cores, running at 2.3 GHz, and 192 Kepler based GPU cores [23].

3) High-Assurance (HA) Platform : The HA platform is ultimately responsible for safety of the system and therefore must be simple and highly resistant to transient hardware faults. For this, we use an Arduino Due platform. The platform equips a single-core Cortex-M3 processor with 80 MHz maximum operating frequency. It is based on simple in-order architecture and the number of transistors of the chip is much smaller than that of the Tegra K1. These characteristics make

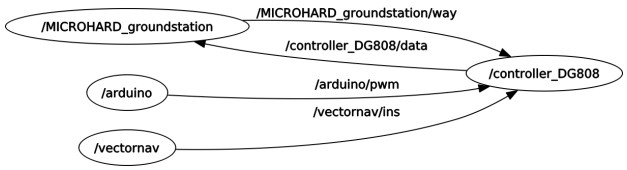

Fig. 3: ROS based software architecture on the high-performance platform (Tegra K1)

the HA platform less susceptible to SEUs. Also, the platform supports numerous I/O options—GPIO, PWM, I2C, and etc—that are needed to connect various basic sensors.

B. Software

1) Performance Controller: The performance controller is implemented on the Tegra K1. We use Ubuntu 12.04 Linux as the OS and the Robot Operating System (ROS) as the middleware framework. In ROS, the system, a robot, is composed of a set of *nodes*, each of which is a separate Linux process participating in the communication network for the robot. A node can publish messages to a communication channel, called a *topic*, which can be subscribed by other nodes to receive the published messages.

Figure 3 shows the nodes and topics of the performance controller. In the figure, the controller_DG808 node is the main control node. The node subscribes the `/vectornav/ins` topic, to receive GPS and IMU sensor values, and the `/arduino/pwm`, to receive commands from the remote controller (RC) and pressure sensor values. (The RC is used for manual control of the UAV). The controller node publishes control outputs to the `servo_op` topic, which is then subscribed by the arduino node and is forwarded to the HA platform via serial. The MICORHARD_groundstation node is responsible for communicating with the remote ground station on the laptop PC. We use the open-source Qgroundcontrol [2] as the ground station software. We use a microhard modem for long range communication [1] between the UAV and the ground station to send/receive statistics (UAV to groundstation) and waypoints (groundstation to UAV)

Currently, the performance controller only uses the basic sensors, and it does not utilize radar and vision sensors; we are actively developing radar and vision-based sense-and-avoid capability in the performance controller.

2) Safety Controller: The safety controller runs on the Arduino Due platform. The control algorithm is a simple PID based one [14] and it is modeled and validated using Matlab Simulink [3]. Figure 4 shows the simulink model. We then generate C code of the controller using Matlab Simulink Coder. In this way, we minimize the possibility of bugs in the safety controller, although it does not guarantee the absence of bugs in the model. In fact, our initial model contained a divide zero bug, which was manifested only in a certain sensor input value ranges. Nevertheless, model-based design is highly desired for a safety controller as it can significantly reduce the possibility of coding mistakes and other common software bugs [12], [8].

3) Decision Logic: The decision logic also runs on the Arduino Due, along with the safety controller. In fact, the generated C code of the safety controller is merged with the

```
1  void loop()
2  {
3      // basic sensor input
4      sensor_data = read_sensors();
5      send_to_HPP(sensor_data);
6
7      // execute safety controller
8      out_hap = safety_controller(sensor_data);
9
10     // wait for the performance controller
11     out_hpp = receive_from_HPP(timeout);
12
13     // decision logic
14     if (decision_check(out_hpp));
15        run_servo(out_hpp);
16     else {
17        run_servo(out_hap);
18        // recover HPP
19        try_recover_hpp();
20     }
21
22     sleep_until_next_period();
23 }
```

Fig. 5: Decision logic on Arduino.

TABLE II: Fault detection

Observed behaviors	Faults in the HP platform
No output	OS/controller crash
Delayed output	Deadline miss
Unsafe output	Bugs, SEUs, bad controller design, etc.

decision logic. The safety controller is directly called by the decision logic while the output of the performance controller is received asynchronously from the Tegra K1 via serial. Which output is used to actuate the UAV is then determined by the decision logic. Figure 5 shows the overall process of the decision logic. Note that each loop is executed periodically at a regular interval (20Hz in our current implementation). The implementation assumes that the safety controller always completes within the interval without errors—i.e., `out_hap` is always valid and computed within the interval (deadline). On the other hand, the output of the performance controller can be deemed unsafe or invalid due to a number of reasons.

4) Fault Detection and Recovery: The decision logic detects faults in the HP platform by observing its outputs (See Line 14 in Figure 5). Once a fault is identified, the decision logic switches its control to the safety controller and tries to recover the HP platform by restarting the system.

Table II shows the observations by the decision logic and the corresponding faults at the HP platform. First, 'No output' means that the decision logic does not receive performance controller's control outputs. This can be caused by software faults such as OS or controller crashes. In our current implementation, the decision logic waits one more control period before it makes a switch to the safety controller. If the outputs are received in the next period, the decision logic's observation is 'Delayed output' and the system continues to use the performance controller. Also, we categorize the certain outputs as 'Unsafe', if they are clearly out of the valid ranges or significantly off-track the given waypoints.

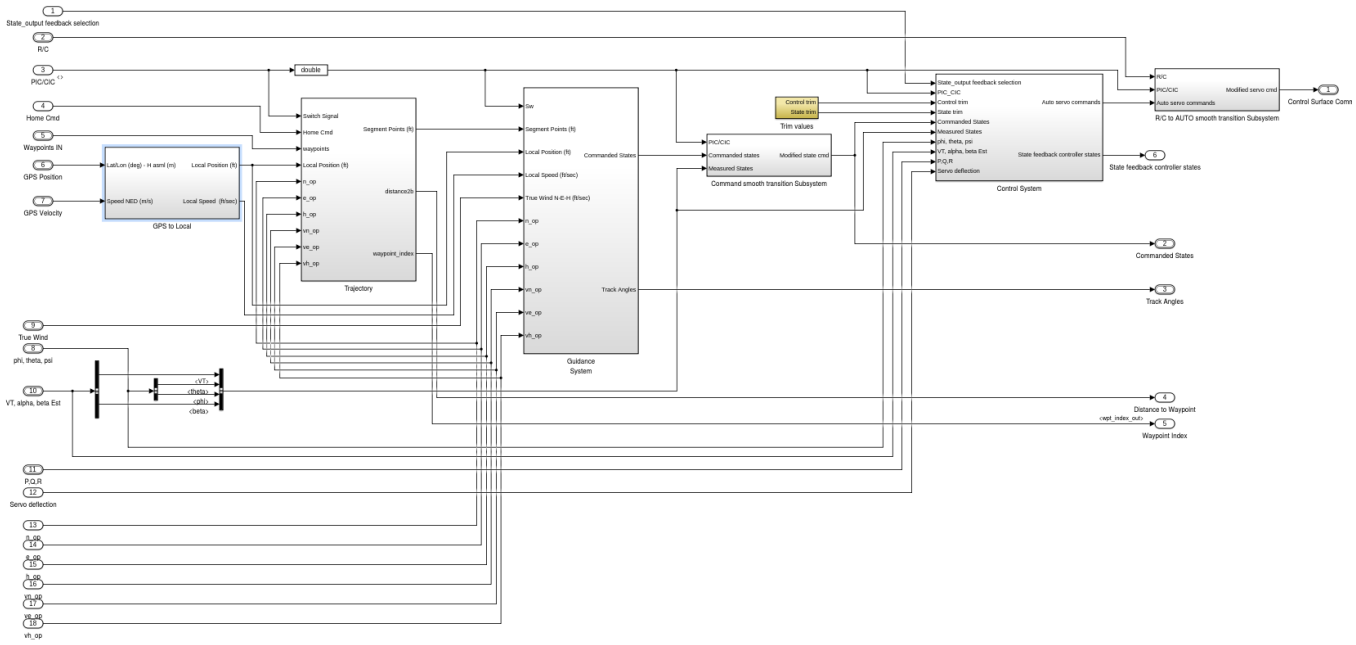

Fig. 4: Autonomous controller block diagram

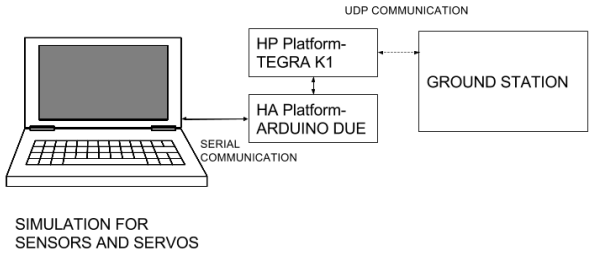

Fig. 6: Experiment setup for UAV simplex

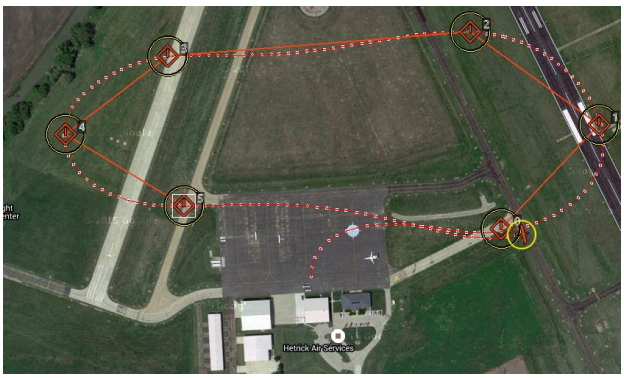

Fig. 7: The flight path

V. Evaluation

A. Setup

Figure 6 shows the hardware-in-the-loop (HIL) experiment setup that we used to evaluate our Simplex based UAV system. The experiment setup consists of an Arduino Due, a Tegra TK1, and a laptop. The Arduino Due executes the safety controller and decision logic, while the Tegra TK1 platform executes the performance controller. Together, they form the cyber system of the UAV. On the other hand, the laptop runs the ground station and a custom built flight simulator. The flight simulator, which is implemented in Matlab, provides simulated sensor values to both the controllers and in turn receives the control command surface outputs from the controllers and calculate the next inputs.

B. Case study: performance controller crash

In this case-study, we intentionally crash the performance controller in the HP platform to evaluate the system's ability to detect and recover from the fault.

Figure 8 shows the outputs of the performance controller (top), the safety controller (middle), and the decision logic (bottom), collected over 200 seconds duration (Note that the figure shows only one of the four control outputs.) For the first 100 seconds, both safety and performance controllers are working in parallel on HA and HP platforms, respectively. In this normal operation mode, the decision logic chooses to use the outputs of the performance controller. At 100 seconds, however, we inject a fault by manually terminating the performance controller in the HP platform. Because the decision logic observes no outputs from the performance controller, it switches to use the outputs from the safety controller. At about 130 seconds, the performance controller is restarted and produces control outputs. However, the decision logic does not immediately switch back to the performance controller because the outputs of the performance controller is not

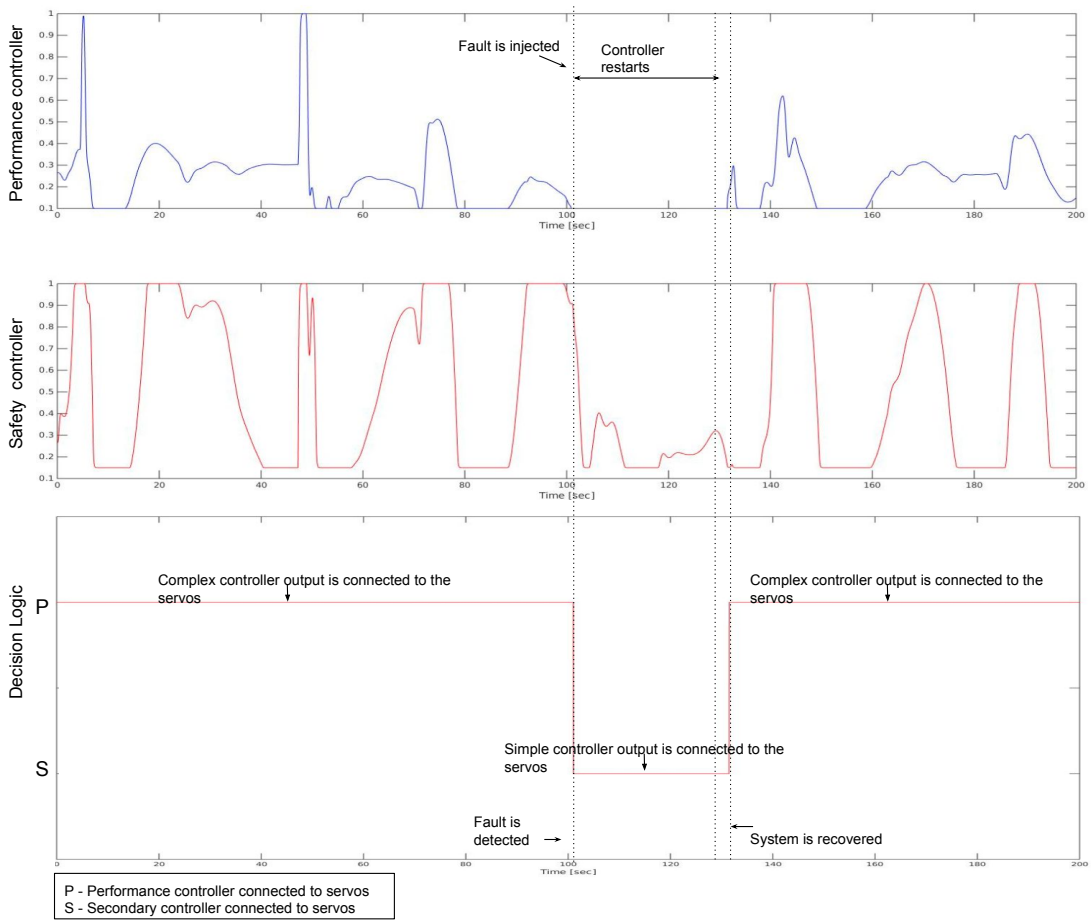

Fig. 8: Outputs of safety controller, performance controller, and decision logic. A fault is injected at time 100 second.

stabilized yet. The stability factor depends on the convergence of the filters and the guidance logic of the controller. Once the outputs are stabilized, the decision logic switches back to use performance controller's outputs. Figure 7 shows the ground station tracking of the UAV system during the test [1]. The solid orange lines denote the waypoint trajectory uploaded to the UAV system from the ground station and the red lines show the trajectory followed by the flight and the white dots represent the position update from the UAV system on the ground station. Our UAV system is designed to update the ground station at 10Hz.

VI. Related Work

Software reliability have long been a major concern in safety-critical cyber-physical systems. For example, software in commercial airplanes must follow certain standards (e.g., DO-178C [25] and ARINC-653 [5]) and be certified by certification authorities. At the application-level, model-based designs and formal-methods [12], [8], [13] have made significant advance, it is still difficult to verify the correctness of the software at the code-level [21]. Furthermore, the applications also rely on many other system software components, including the OS and middleware, each of which may contain bugs.

System-level reliability and fault-tolerance techniques, especially in the context of intelligent cyber-physical systems, have been studied to mitigate possible software bugs. SAFER is a middleware framework that uses software-level redundancy to enhance system-level reliability. However, such a middleware solution may not survive from an OS failure (e.g., BSOD). The seL4 is a micro-kernel where its functional correctness was formally verified [16]. However, functional correctness does not means its temporal correctness is guaranteed. Nor it guarantees correct operations in the presence of transient hardware faults (SEUs), which are becoming more prevalent as technology scaling continues [10]. C'Mon focuses on detecting OS-level timing faults (e.g., deadline misses due to scheduling) that are caused by SEUs [28]. To guard the OS code and data against SEUs, dOSEK proposed to use a special encoding in storing OS code and data, which enables detection and recovery from the SEUs [15]. However, its software-based approach comes at a significant performance penalty.

SEUs can be reduced at each hardware component-level by, for example, adopting ECC and other so called "hardening" techniques [7]. Also, hardware redundancy solutions, such as Triple-modular redundancy (TMR) [18] or dual-redundancy, can protect the system from transient hardware faults [7].

[1]The video of our experiment can be found in the following link https://youtu.be/p6BT3UN8qJE

In today's commercial airplanes and satellites, the physical redundancy based techniques are used due to their high safety-critical requirements [11]. However, these hardware techniques comes at substantial space and performance penalties and high cost and development time—unsuitable for UAVs. Also, the processors used in these applications often several generations behind the stat-of-the-art COTS processors [17], [9]. This is a serious problem for intelligent UAVs where high computing performance is a key to achieve high intelligence.

The simplex architecture [27] is a special form of redundancy in which safety and performance are decoupled to safety and performance controllers, respectively. It was originally implemented at the application software-level in a single computing platform and used to upgrade/verify controllers without stopping the system [27]. More recently, Bak et al. implemented the safety-critical parts of the system—safety controller and decision logic—in FPGA so that it can tolerate system-level faults in the computer system running the performance controller for an inverted pendulum and a cardiac pacemaker [6]. Our work is inspired by their work, but we use an Arduino platform instead of FPGA to enable low-cost safety system design in the context of UAV systems. We also consider the implications of our simplex design in terms of hardware transient faults.

VII. Conclusion

We have presented a fault-tolerant UAV design based on the Simplex [27] architecture. The proposed design uses two heterogeneous platforms with distinct reliability and performance characteristics.

Our main idea is that we use a reliable but less performant hardware platform, we call a High-Assurance (HA) platform, to be responsible for safety while we use a more performant, but potentially less reliable platform, which we call a High-Performance (HP) platform, for performance and intelligence of the UAV. During the normal operation, the HP platform is responsible for controlling the UAV. However, if it fails due to transient hardware faults or software bugs, the HA platform will immediately take over until the HP platform recovers. As such, our design provides a much needed fail-operational property in the UAV.

We have implemented the proposed design on an actual UAV using a low-cost Arduino and a high-performance Tegra TK1 platform. We have demonstrated the system's ability to detect and recover from failures through a set of experiments in a hardware-in-the-loop simulation setting. In the future, we plan to implement more sophisticated vision sense-and-avoid capability and perform real flight tests in an outdoor setting.

Acknowledgement

This work is supported by the National Aeronautics and Space Administration's (NASA's) Leading Edge Aeronautics Research for NASA (LEARN) fund under grant number NNX15AN94A and Paul G. Allen Family Foundation (PGAFF) grant number KUAE#40956.

The authors would like to thank George Blake for his support on hardware development and validation of the UAV.

References

[1] Microhard. http://www.microhardcorp.com/n920.php.
[2] Qground station. http://qgroundcontrol.org/.
[3] Simulink. http://www.mathworks.com/products/simulink/.
[4] Vn-220 rugged gps/ins. http://www.vectornav.com/products/vn200-rugged/.
[5] Aeronautical Radio Inc. *Avionics Application Standard Software Interface (ARINC) 653*, 2013.
[6] S. Bak, D. Chivukula, O. Adekunle, M. Sun, M. Caccamo, and L. Sha. The system-level simplex architecture for improved real-time embedded system safety. In *Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 99–107. IEEE, 2009.
[7] R. Baumann. Soft errors in advanced computer systems. *Design & Test of Computers, IEEE*, 22(3):258–266, 2005.
[8] J. Bengtsson, K. Larsen, F. Larsson, P. Pettersson, and W. Yi. *UPPAAL: a tool suite for automatic verification of real-time systems*. Springer, 1996.
[9] P. Bieber, F. Boniol, M. Boyer, E. Noulard, and C. Pagetti. New Challenges for Future Avionic Architectures. *AerospaceLab Journal*, (4), 2012.
[10] S. Borkar. Designing reliable systems from unreliable components: the challenges of transistor variability and degradation. *Micro, IEEE*, 25(6):10–16, 2005.
[11] D. Brière and P. Traverse. AIRBUS A320/A330/A340 electrical flight controls-A family of fault-tolerant systems. In *Fault-Tolerant Computing*, pages 616–623. IEEE, 1993.
[12] J. B. Dabney and T. L. Harman. *Mastering simulink*. Pearson/Prentice Hall, 2004.
[13] P. Fritzson. *Principles of object-oriented modeling and simulation with Modelica 2.1*. John Wiley & Sons, 2010.
[14] G. A. Garcia, S. Keshmiri, and R. Colgren. Advanced h-infinity trainer autopilot. *AIAA Modeling and Simulation Technologies Conference*, August 2010.
[15] M. Hoffmann, F. Lukas, C. Dietrich, and D. Lohmann. dosek: the design and implementation of a dependability-oriented static embedded kernel. In *Real-Time and Embedded Technology and Applications Symposium (RTAS), 2015 IEEE*, pages 259–270. IEEE, 2015.
[16] G. Klein, K. Elphinstone, G. Heiser, J. Andronick, D. Cock, P. Derrin, D. Elkaduwe, K. Engelhardt, R. Kolanski, M. Norrish, et al. sel4: Formal verification of an os kernel. In *Symposium on Operating Systems Principles (SOSP)*, pages 207–220. ACM, 2009.
[17] O. Kotaba, J. Nowotsch, M. Paulitsch, S. Petters, and H. Theilingx. Multicore in real-time systems temporal isolation challenges due to shared resources. In *Industry-Driven Approaches for Cost-effective Certification of Safety-Critical, Mixed-Criticality Systems*, 2013.
[18] R. E. Lyons and W. Vanderkulk. The use of triple-modular redundancy to improve computer reliability. *IBM Journal of Research and Development*, 6(2):200–209, 1962.
[19] T. C. May and M. H. Woods. Alpha-particle-induced soft errors in dynamic memories. *IEEE Transactions on Electron Devices*, 26(1):2–9, Jan 1979.
[20] D. Montalk and J. Potocki. Computer software in civil aircraft. In *Digital Avionics Systems Conference, 1991. Proceedings., IEEE/AIAA 10th*, pages 324–330. IEEE, 1991.
[21] Y. Moy, E. Ledinot, H. Delseny, V. Wiels, and B. Monate. Testing or formal verification: Do-178c alternatives and industrial experience. *Software, IEEE*, 30(3):50–57, 2013.
[22] J. Nowotsch and M. Paulitsch. Leveraging multi-core computing architectures in avionics. In *Dependable Computing Conference (EDCC)*, pages 132–143. IEEE, 2012.
[23] NVIDIA. *NVIDIA Tegra K1 Mobile Processor, Technical Reference Manual Rev-01p*, 2014.
[24] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng. Ros: an open-source robot operating system. In *ICRA workshop on open source software*, volume 3, page 5, 2009.
[25] RTCA. *DO-178C Software Considerations in Airborne Systems and Equipment Certification*, 2011.
[26] I. Sadeghzadeh and Y. Zhang. A review on fault-tolerant control for unmanned aerial vehicles (uavs). *Infotech@ Aerospace*, 2011.
[27] L. Sha. Using simplicity to control complexity. *IEEE Software*, 18(4):20–28, 2001.
[28] J. Song and G. Parmer. C'mon: a predictable monitoring infrastructure for system-level latent fault detection and recovery. In *Real-Time and Embedded Technology and Applications Symposium (RTAS), 2015 IEEE*, pages 247–258. IEEE, 2015.
[29] J. F. Ziegler. Terrestrial cosmic rays. *IBM journal of research and development*, 40(1):19–39, 1996.