CEAS-GNC-2024-058

EuroGNC Navigation Guidance Control CEAS Bristol, UK June 11th-13th 2024
University of BRISTOL ROYAL AERONAUTICAL SOCIETY AIAA

Adaptive Avionics System for Multi-agent Computing Resource Sharing under Failures

Chitoshi Tamaoki Visiting Scholar, Department of Aerospace Engineering, Pennsylvania State University, University Park, Pennsylvania, United States, 16802, and M.S. student, University of Stuttgart, Institute of Aircraft Systems, Stuttgart, Germany, cft5405@psu.edu

Thanakorn Khamvilai Assistant Research Professor, Department of Aerospace Engineering, Pennsylvania State University, University Park, Pennsylvania, United States, 16802, thanakorn.khamvila@psu.edu

Eric Johnson Professor, Department of Aerospace Engineering, Pennsylvania State University, University Park, Pennsylvania, United States, 16802, eric.johnson@psu.edu

Johannes Reinhart Ph.D candidate, Institute of Aircraft Systems, University of Stuttgart, Stuttgart, Germany. johannes.reinhart@ils.uni-stuttgart.de

Bjoern Annighoefer Professor, Institute of Aircraft Systems, University of Stuttgart, Stuttgart, Germany. bjoern.annighoefer@ils.uni-stuttgart.de

ABSTRACT

In the foreseeable future, the collaboration and information sharing between unmanned aerial vehicles (UAVs) and conventional aircraft is expected to improve the efficiency and effectiveness of aviation operations. The scope of this article is a proof-of-concept demonstration of a multi-vehicle self-adaptive avionics system where computing resources can be shared across multiple vehicles with the same mission. This increases the flexibility and adaptivity to changing multi-vehicle missions, and failure tolerance of the whole system of interconnected vehicles. Furthermore, it allows optimized usage of resources. Those new features will be achieved by Plug and Fly Avionics (PAFA), a self-organizing avionics platform, developed by the Institute of Aircraft Systems at the University of Stuttgart. To demonstrate the concept, a scenario with a sudden loss of a mission device is simulated. Two separate mission devices with PAFA are used to operate a single drone on which one of the computers is mounted. When the mounted mission device fails, the other mission device detects the failure and seamlessly takes over the mission task to continue the flight. The flight test was successfully conducted and the drone had safely recovered from the incident.

Keywords: Distributed computation, Adaptive avionics system

Nomenclature

PAFA	=	Plug & Fly Avionics
PSU-GUST	=	Penn State/Georgia Tech UAV Simulation Tool
GCS	=	Ground Control Station
FCS	=	Flight Control System

Except where otherwise noted, content of this paper is licensed under a Creative Commons Attribution 4.0 International License.

The reproduction and distribution with attribution of the entire paper or of individual pages, in electronic or printed form, including some materials under non-CC-BY 4.0 licenses is hereby granted by the respective copyright owners.

1 Introduction

The sharing of computing resources can be an important aspect of future aviation, owing to its adaptability, flexibility, and enhanced ability to withstand failures. Rather than relying on centralized computers, interconnected devices are able to share their computational resources. This enables the dynamic allocation of functions, allowing for optimal operation based on the current circumstances. Such adaptive and flexible operation provides increased tolerance for failures, ensuring the continuous functioning of aviation systems in the event of hardware or software malfunctions. This leads to more reliability for future aviation.

In recent years, there has been increasing research on cooperative and self-adaptive UAVs aimed at improving failure tolerance. One notable study by Ziquan Yu et al. [1] proposed a theoretical approach to fault-tolerant cooperative control. This approach involves healthy neighbor UAVs to detect software or hardware malfunctions and provide support through information sharing. The assistant is integrated into the control loop. Another study by Corey Ippolito et al. [2] conducted a collaborative flight test using polymorphic control systems. These systems utilize polymorphic reconfiguration to recover from critical situations, such as sensor malfunctions during landing. During the flight test, a ground vehicle equipped with onboard sensors determined the position of an aircraft with sensor malfunction during landing. Additionally, Anawat Pongpunwattana et al. [3] developed effective real-time planning and cooperation algorithms for autonomous team decision-making. These algorithms adapt the configuration of the UAVs to changing circumstances. Xiangwang Hou et al. [4] performed automatic task allocation among drones using distributed algorithm and confirmed its effectiveness. In their demonstration scenario, computational energy minimization problem was solved with constraints of latency and reliability.

The above mentioned studies showed the effectiveness of self-adaptive system for increased failure tolerance, flexibility to circumstances and optimum usage of resources in the application field of UAVs. In this particular study, Plug & Fly Avionics (PAFA) platform [5] [6] is utilized that is under development by the Institute of Aircraft Systems at the University of Stuttgart. Unlike current systems, PAFA features dynamic reconfiguration during its operation period, enabling greater flexibility in the distribution of functionalities across devices. This seamless and immediate reconfiguration eliminates any downtime for critical operations, making it a significantly important feature for aviation applications.

So far, the PAFA concept was considered only for using shared computing resources within a single vehicle. We suggest broadening this concept by sharing computing resources between multiple cooperating vehicles or multiple computing units. This will be advantageous for future aviation scenarios, where autonomous flying vehicles will cooperate to fulfill common mission tasks, such as object detection [7], more efficient forest fire fighting [8], infrastructure monitoring application [9]. By sharing computing resources, the system of cooperating vehicles becomes more flexible, adaptive and may continue its mission despite failures. To prove the capability of PAFA in such situations, a drone flight test was conducted as part of this study.

In chapter 2, the state of art of PAFA is presented, along with its technical features. In chapter 2.1, a concept to increase failure tolerance through the collaboration of interconnected devices and dynamic reconfiguration is proposed. This concept is to be proven with a drone flight test, in which a critical device loss is simulated. In chapter 3, its technical realization is described. The preparation and results of this flight test are discussed in chapter 5. Finally, in the last chapter, the conclusion and outlook are provided.

2 State of the Art of Plug & Fly Avionics

Current integrated modular avionics (IMA) is a computing platform for software-based system functions. It offers a more efficient, flexible, and cost-effective approach to avionics systems, providing

Except where otherwise noted, content of this paper is licensed under a Creative Commons Attribution 4.0 International License.

The reproduction and distribution with attribution of the entire paper or of individual pages, in electronic or printed form, including some materials under non-CC-BY 4.0 licenses is hereby granted by the respective copyright owners.

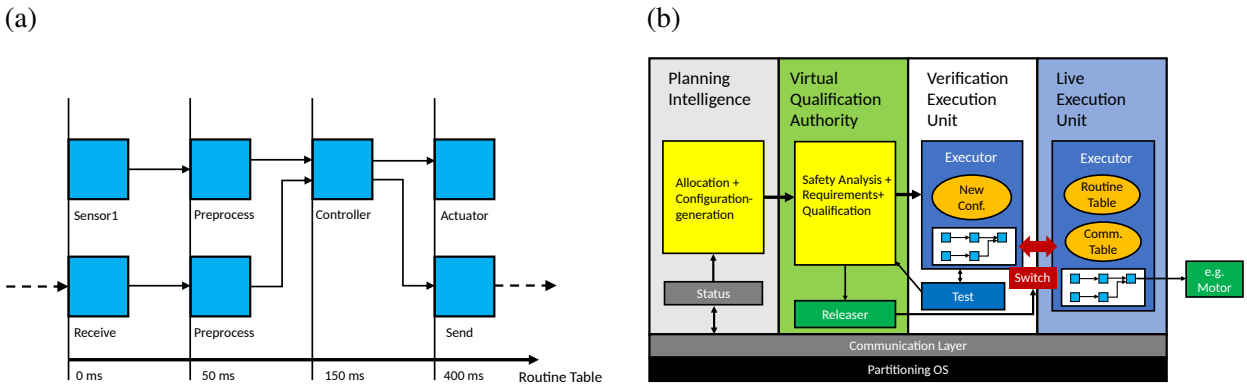


Fig. 1. (a) Example of combination of basic task blocks, (b) Abstract illustration of partitions of PAFA

benefits in terms of modularity, scalability, reliability, and maintainability compared to traditional separate and complex system architecture. However, the system is configured once during integration and may not change while the aircraft is in flight. This limitation reduces the flexibility of functionalities over computing devices, necessitating significant configuration effort, and making it error prone due to the large number of manual configuration parameters involved.

The situation with a vast number of configuration parameters is often referred to as the software complexity crisis[10]. The only solution to this issue is the concept of autonomic computing. In this framework, computing systems manage themselves autonomously. To achieve this self-management, four key aspects need to be considered: self-configuration, self-optimization, self-healing, and self-protection.

The Institute of Aircraft Systems at the University of Stuttgart currently develops novel type of avionics platform, called Plug & Fly Avionics (PAFA). The platform possesses two notable features [5]: automatic allocation of configuration parameters, and seamless in-operation reconfiguration to realize the four key aspects for autonomic computing concept in aviation. The PAFA platform is designed to be self-organizing, automatically allocating configuration parameters based on a portable application specification [11] including the intended function, communication requirements between functions, and prescribed redundancy and safety levels. During this process, the platform determines message paths and function hardware distribution. In order for new configurations to take effect, they must be applied to all participating devices. The switching mechanism within one cycle allows for a seamless transition between old and new configurations, ensuring a smooth and uninterrupted operation. The PAFA concept is specifically designed for safety criticality and aviation certification.

The executable unit of each device in PAFA consists of basic task blocks and configuration. Each basic task block has its own function to execute and can have inputs and outputs that can be connected to other task blocks. Each task block undergoes testing and qualification for the required safety level. When combined, multiple task blocks can perform a specific system function for the device, as shown in Figure 1(a). A group of task blocks working together to execute a system function is referred as cluster. Clusters of task blocks within a device can communicate with other clusters across devices through send and receive blocks. The schedule of execution and the connections of inputs and outputs for the task blocks are described in a routine table. Each device is configured with a specified routine table to perform the assigned system functions. The inter-device and inter-cluster communication is controlled by a communication table, which contains information such as the IP address or CAN ID of the target device, message length, and buffer. The send and receive blocks in the routine table refer to this communication table to establish communication.

The platform architecture is designed with a partition-based approach, consisting of four main partitions: Live Execution Unit, Planning Intelligence, Virtual Qualification Authority, and Verification Execution Unit, as shown in Figure 1(b). The Live Execution Unit is responsible for executing the current configuration, which includes task blocks performing according to the routine table and communication table. On the other hand, the Planning Intelligence partition is responsible for automatically allocating

Except where otherwise noted, content of this paper is licensed under a Creative Commons Attribution 4.0 International License.

The reproduction and distribution with attribution of the entire paper or of individual pages, in electronic or printed form, including some materials under non-CC-BY 4.0 licenses is hereby granted by the respective copyright owners.

functions and generating new configurations through optimization methods. This allows for the real-time generation of new configurations that are best-effort for the current system status. The Virtual Qualification Authority is responsible for qualifying the newly generated configurations from the Planning Intelligence. In the background, the Verification Execution Unit runs with a new configuration with its outputs are passivated to verify the proper operation of the new configuration. If the configuration in the Live Execution Unit is deemed inappropriate for the current situation in an airplane, a switching occurs between the Live Execution Unit and the Verification Execution Unit within a single execution cycle. This means that the Verification Execution Unit takes over the position of the Live Execution Unit, while the former Live Execution Unit becomes the Verification Execution Unit. Through this process, the new configuration can be seamlessly applied. The communication is supported by PAFA middleware. This abstracts communication, such that individual components can communicate with any of available communication technologies (currently for PAFA ethernet/Wifi or CAN). PAFA is currently developed with C programming language. The switching is technically implemented with changing the pointer to the new configuration at the beginning of an execution cycle.

The compilation of all necessary essential information for automated allocation in PAFA can be accomplished using a GUI tool based on Web technology, as discussed in [12]. However, a significant challenge lies in developing a qualification method to assess the safety level of newly generated configurations in Planning Intelligence [6] [13]. This aspect of development is particularly challenging as it involves considering the overall safety level of PAFA.

2.1 Concept

PAFA stands out for its dynamic reconfiguration capability, which greatly enhances optimized resource usage. Initially, the PAFA concept has only been explored for utilizing shared computing resources within a single vehicle. The effectiveness of this concept in an unmanned multicopter is shown in [11] as a part of PAFA developement. If PAFA can be mounted on hardware components as middleware, micro-controllers without operation system can be also a part of PAFA platform. In that study, each motor is equipped with a micro-controller with ARM Cortex-M7 core which enables the motor as a part of PAFA system on the UAV. However, we propose expanding this concept to include resource sharing between multiple collaborating vehicles as shown in Figure 2. With its ability to detect failures and optimize configurations, PAFA enables the system to recover from failures by redistributing critical functions across different hardware components. This is particularly beneficial for multi-vehicle systems like drone swarms, where all drones are connected through PAFA and actively share information. In the event of a device failure within the swarm, the system's configuration is updated, allowing other devices to assist the affected drone. Consequently, the failure tolerance of the swarm is significantly increased.

2.2 Use-Case to Demonstrate the Concept

To demonstrate in-flight dynamic reconfiguration of inter-connected devices, a use-case scenario was derived. In this scenario, a drone equipped with a Flight Control System (FCS) autonomously navigates itself based on waypoints generated by other mission devices. Although PAFA is capable of redistribution of tasks to micro-controllers which are often used for flight controllers with high execution frequency, in this particular study, only the mission relevant tasks are the targets of reconfiguration.

The drone and its FCS are solely responsible for stabilizing and following the waypoints provided by the mission devices. The mission devices have an interface to receive telemetry information from the drone and create new waypoint commands to guide its movement. To showcase the collaboration of interconnected devices, we utilize two mission devices equipped with PAFA. One is mounted on the drone (referred to as mission device 1) and the other is located on the ground (referred to as mission

Except where otherwise noted, content of this paper is licensed under a Creative Commons Attribution 4.0 International License.

The reproduction and distribution with attribution of the entire paper or of individual pages, in electronic or printed form, including some materials under non-CC-BY 4.0 licenses is hereby granted by the respective copyright owners.

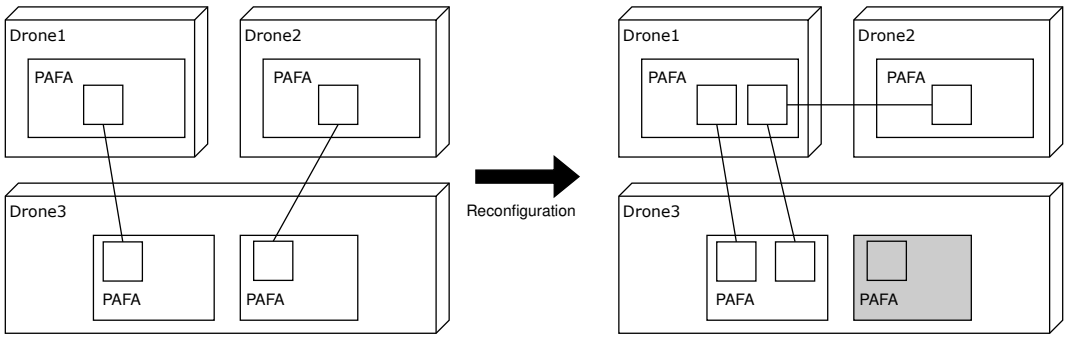


Fig. 2. The concept of resource sharing among multiple collaborating flying vehicles using PAFA. The square markers depict system functions assigned to each vehicle, and the lines represent signal routes. In the initial phase (left), drone 1 and drone 2 communicate with drone 3 individually. In the subsequent phase (right), drone 3 encounters a malfunction in its communication with drone 2. Consequently, tasks and signals are reallocated to establish a new communication route, ensuring the preservation of system function within the entire swarm.

device 2). This placement of the mission device 2 on the ground is intentional. It serves to lessen the difficulty and workload associated with the flight test, while also enhancing safety at the experiment.

In the first phase, new commands are generated by mission device 1, while mission device 2 monitors its activity. In the second phase, mission device 1 is intentionally disabled, and mission device 2 detects this change and sends a request for a reconfiguration to ensure the mission continues. Consequently, mission device 2 takes over the responsibility of generating commands.

3 Technical Realization

A hexacopter drone was prepared to demonstrate the concept. The basic configuration of the devices and function allocations are depicted in Figure 3. The drone is equipped with various components, including motors, sensors, Pixhawk with PX4, a Raspberry Pi 4B, and a Raspberry Pi 3B. An essential software component utilized in this setup is Penn State/Georgia Tech UAV Simulation Tool (PSU-GUST) [14][15]. PSU-GUST is a high-level dynamics simulation tool that enables the simulation and the development of multi-rotor UAVs. An onboard version of PSU-GUST is utilized, which can optimize the flight path to given flight plans and has an interface to a low-level flight controller PX4. The Raspberry Pi 4B hosts PSU-GUST, as well as "PSU-GUST Comm", an interface of PSU-GUST for a third-party software such as PAFA. It also hosts PAFA Reconfigurator which sends new configurations to PAFA-participating devices and a reconfiguration command to initiate the switch mechanism. This reconfigurator is implemented in Python programming language. This is one of the functions of Planning Intelligence partition. This partition is still under development and there is only an experimental Python implementation. As for mission devices, the executable of PAFA is hosted by Raspberry Pi 3B on the drone, acting as mission device 1. The mission device 2, also equipped with PAFA, is hosted on a Linux laptop. The Linux operation system on the laptop and two Raspberry Pis is to reduce the implementation difficulty, especially aimed to support the parallel running of PAFA Reconfigurator and PSU-GUST with different programming language implementations. Due to the availability in the laboratory, two different kinds of Raspberry Pi are used. It is representative of a mission computer, that in a real use case could be also airborne on another flying vehicle. All the devices are interconnected via a single WiFi network (G-type with 2.4 GHz), and communication between them is established using IP addresses and ports. The WiFi network is used to facilitate communication between the Raspberry Pi 4B and mission device 1 for the sake of simple development.

The entire message paths are illustrated in Figure 4. The communication between PSU-GUST and PAFA is facilitated by the PSU-GUST Comm, which is connected to PSU-GUST via a PSU-GUST

Except where otherwise noted, content of this paper is licensed under a Creative Commons Attribution 4.0 International License.

The reproduction and distribution with attribution of the entire paper or of individual pages, in electronic or printed form, including some materials under non-CC-BY 4.0 licenses is hereby granted by the respective copyright owners.

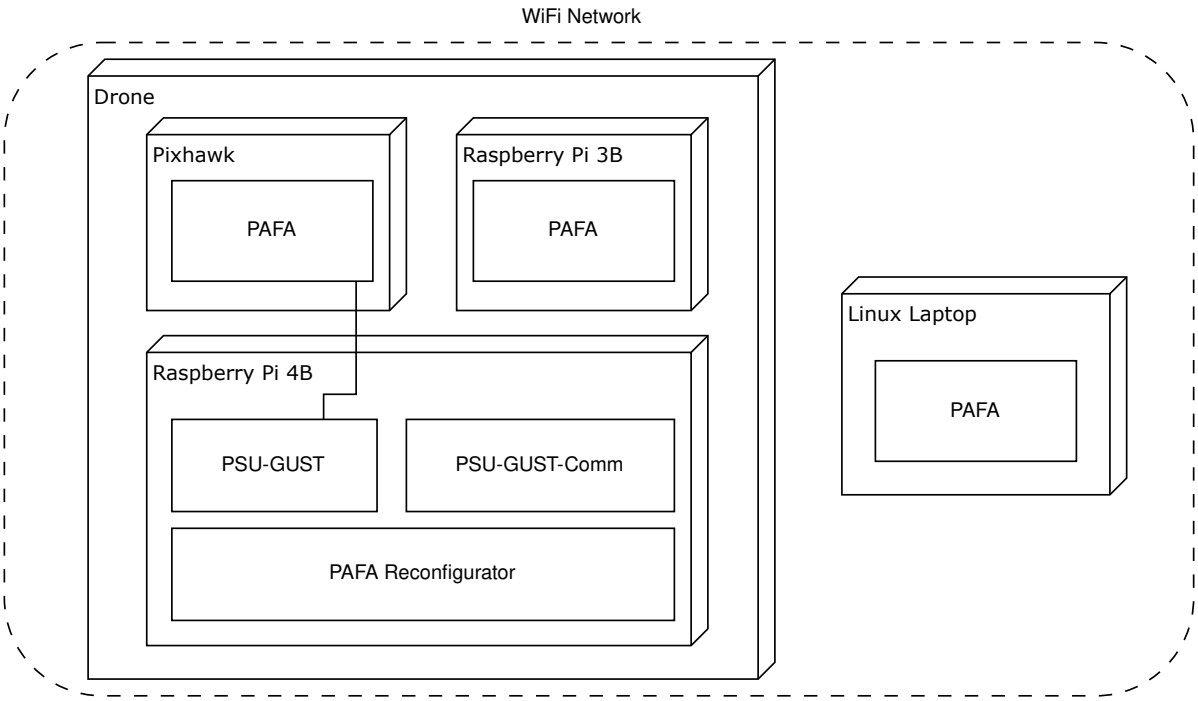


Fig. 3. The configuration of devices and functions assigned to them. All devices are interconnected via a WiFi network except the line which represents a physical connection.

datalink. The connection between PSU-GUST Comm and PAFA is established using UDP to establish asynchronous inter-process communication. Firstly, the raw telemetry data from the drone is transmitted from PX4 to PSU-GUST, where it undergoes data processing such as Kalman filter for state estimation. The processed data is then forwarded to PSU-GUST Comm, which distributes it to PAFA over UDP. In the case of command data, PAFA sends the new command back to PSU-GUST through UDP. Additionally, PAFA engages in communication with other devices PAFA is mounted on, supported by the PAFA communication method. This method is facilitated by send and receive blocks that serve as interfaces for communication. During the configuration process, the PAFA reconfigurator also employs the PAFA communication method to send a new configuration.

4 System Mechanism

This chapter provides a detailed explanation of the system mechanism used for the demonstration, which is also demonstrated in Figure 5. While PAFA has the capability for automatic allocation of functions based on device-function capability assessment, for the purpose of safety, two prescribed and tested configurations were utilized for the demonstration, as PAFA lacks operational Virtual Qualification Authority.

4.1 Step 1

Firstly, PX4 and PSU-GUST are activated prior to the activation of mission critical functions, such as PAFA, as they are capable of stabilizing the drone on their own. Following this, PSU-GUST Comm is activated to establish the interface between PSU-GUST and PAFA. Once the communication between PSU-GUST and PSU-GUST Comm is confirmed, PAFA is started on both mission devices. These activation processes are manually carried out on the Linux laptop using terminal windows over SSH connection. At this stage, no configuration is assigned to them. After ensuring proper activation, the PAFA Reconfigurator is executed by hand to send the initial configurations, including routine tables and communication tables and a reconfiguration command, as depicted in Figure 5(a). Each configuration

Except where otherwise noted, content of this paper is licensed under a Creative Commons Attribution 4.0 International License.

The reproduction and distribution with attribution of the entire paper or of individual pages, in electronic or printed form, including some materials under non-CC-BY 4.0 licenses is hereby granted by the respective copyright owners.

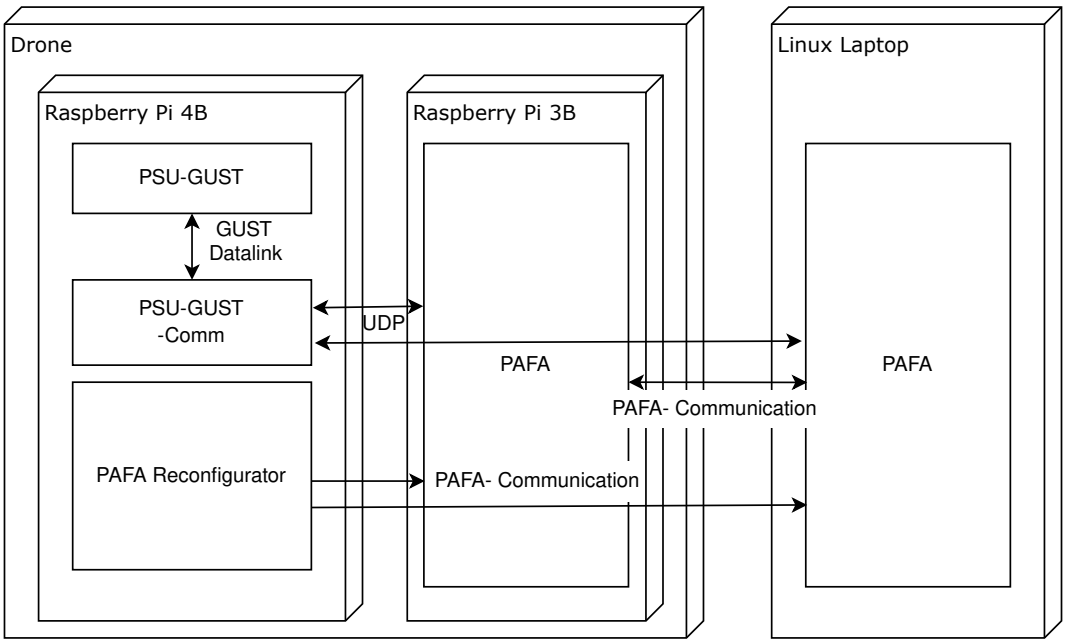


Fig. 4. The communication paths between functions. The physical connections depicted in Figure 3 are not shown here.

information is transmitted separately and stored in the RAM of the mission devices. Upon completion of this configuration process, a reconfiguration command is sent to both mission devices to make the new configuration effective.

4.2 Step 2

The first configuration is illustrated in Figure 5(b). All the processes in this step are automatically executed after the reconfiguration command in step 1. The main task of Raspberry Pi 3B as mission device 1 is generation of new commands. It is equipped with four task blocks that are executed based on the routine table. The feedback transfer block serves as an interface to receive telemetry data from PSU-GUST Comm. It receives this data as a UDP message and transfers it to the command generation block. The command generation block determines whether a new command needs to be generated based on the received telemetry data. In our case, the command represents a waypoint coordinate. If the drone is still en route to the latest waypoint, the command generation block does not generate a new command. When a new command is generated, it is passed on to the command transfer block. The command transfer block converts the new command into a UDP message, which is then sent to PSU-GUST Comm. The transferred command in PSU-GUST Comm is transmitted to PSU-GUST via the PSU-GUST datalink, and the drone begins to pursue the new waypoint. The send block transmits the message to the mission device 2. The target IP address is specified by the communication table sent in step 1. The message sent by the send block is a counter, which will be further explained later.

On the other hand, the laptop as mission device 2 has been assigned the task of monitoring. This is achieved through the use of a receive block and a watchdog block. The received message within the receive block is then transmitted to the watchdog block. The role of the watchdog block is to monitor the activity of the command generation block on another mission device. Although it is possible to send a message status along with the message content, the watchdog block is unable to detect any loss of messages as the receive block continuously transfers the message within its buffer. To avoid this problem, the command generation block on mission device 1 sends a counter that increments at each cycle. If the device is terminated for any reason and does not send new messages, the counter within the buffer remains the same. If the watchdog block does not receive a new counter within a specific period of time,

Except where otherwise noted, content of this paper is licensed under a Creative Commons Attribution 4.0 International License.

The reproduction and distribution with attribution of the entire paper or of individual pages, in electronic or printed form, including some materials under non-CC-BY 4.0 licenses is hereby granted by the respective copyright owners.

it will conclude that the command generation block is not functioning correctly and thus a new swarm configuration is required. The same decision is made when the message status is not appropriate.

4.3 Step 3

In order to simulate a mission device loss during the demonstration, the mission device 1 on the drone will be intentionally terminated. This termination is triggered by sending a "signal kill" command using the keyboard shortcut "ctrl + C". Subsequently, no new counter information is transmitted to the mission device 2. As a result of this termination, the watchdog block promptly detects the failure.

4.4 Step 4

After the prescribed time-out period elapses, the watchdog block sends reconfiguration request as a UDP messsage to the PAFA reconfigurator.

4.5 Step 5

The PAFA reconfigurator receives a UDP message containing the request and performs the identical configuration process as described in step 1. In order to proceed with the flight mission, the feedback transfer block, command generation block, and command transfer block are allocated to mission device 2. Since there is currently only one mission device, no send block is utilized. This new configuration is loaded while the old configuration remains operational. If PAFA on mission device 2 has other tasks that are executed in parallel, they are not affected by this loading configuration process.

4.6 Step 6

After all the new configuration is sent, the PAFA reconfigurator sends the reconfiguration command. This command switches the execution configuration of the second mission device with PAFA within one cycle. The mission device 2 is now responsible for generating new command and the drone continues its original mission.

5 Flight Test and Results

In order to ensure safety during the demonstration, the mission was designed to involve ascending and descending to predetermined altitudes. When the drone reaches the specified altitude, a mission device generates a new altitude command that is then sent to PSU-GUST as a waypoint. For the test flight, the high altitude command is set at 4 feet, while the low altitude command is set at 2 feet. The system frequency is set as 1 second, although more than 100 times faster processing is possible. This is to make it easy to monitor the system behavior.

The devices utilized for the flight test are presented in Figure 6. Starting from the left, there is a GCS laptop equipped with an antenna module and a gaming controller. The laptop runs PSU-GUST for GCS, which enables the visualization of telemetry and waypoints of the drone, as well as access to settings. The gaming controller, connected to the GCS laptop, allows the GCS operator to guide the drone with the assistance of PSU-GUST easily and stably. In the bottom middle, there is a radio controller operated by a safety pilot. The safety pilot's role is to intervene in drone control when the autopilot does not perform as expected. The drone itself is depicted in the upper middle. It is equipped with a Raspberry Pi 4B, which hosts PSU-GUST, PSU-GUST Comm, PAFA Reconfigurator, and a Raspberry Pi 3B as mission device 1 with PAFA. The laptop on the far right serves as mission device 2 with PAFA and is also used to operate vehicle functions via SSH connections.

Except where otherwise noted, content of this paper is licensed under a Creative Commons Attribution 4.0 International License.

The reproduction and distribution with attribution of the entire paper or of individual pages, in electronic or printed form, including some materials under non-CC-BY 4.0 licenses is hereby granted by the respective copyright owners.

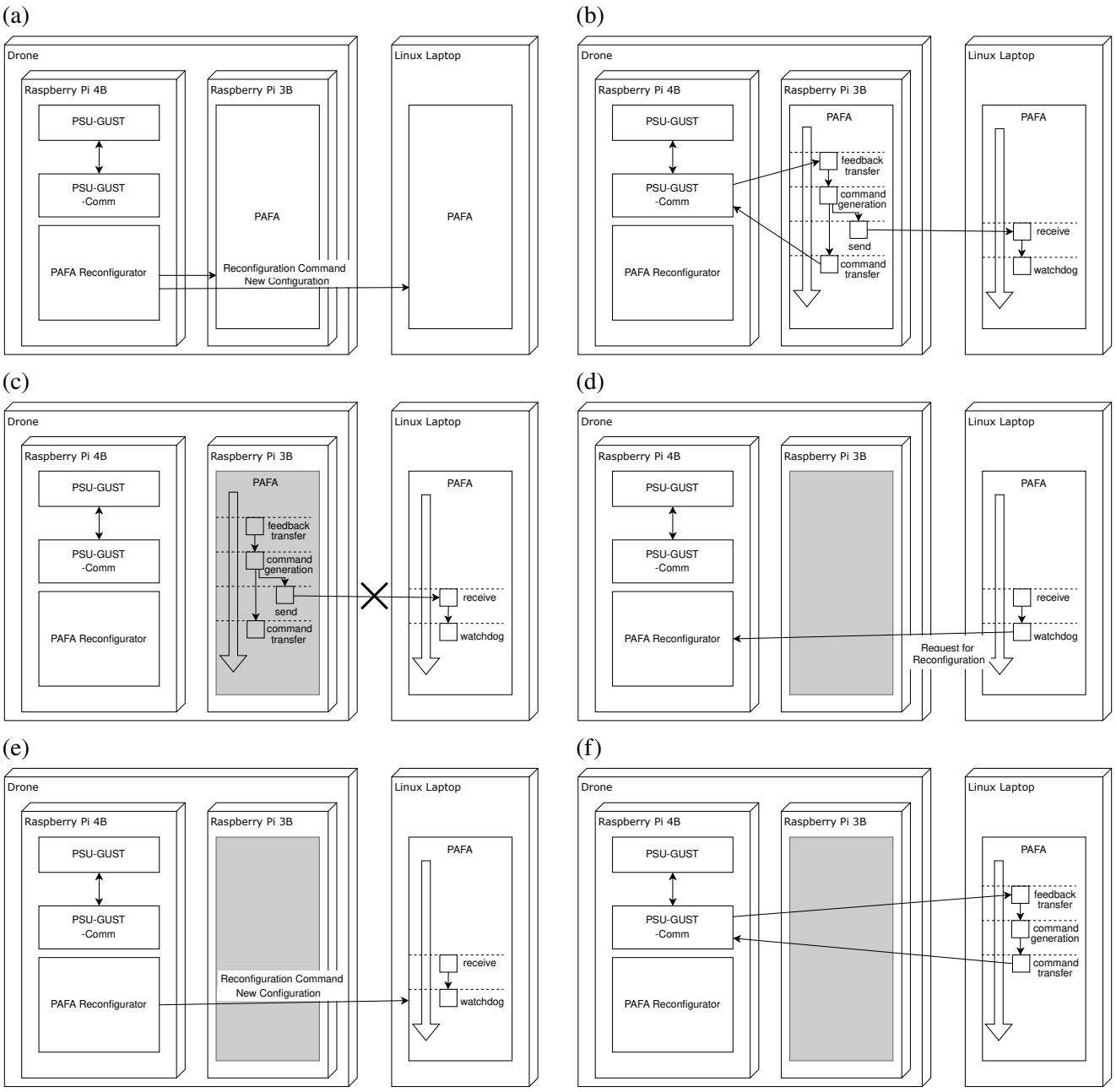


Fig. 5. Visualizations of the sequential steps in the process. (a) Step 1, (b) Step 2, (c) Step 3, (d) Step 4, (e) Step 5, and (f) Step 6

5.1 Results

The flight log for vertical movement is shown in Figure 7. Due to the common coordinate system in aviation, the negative value indicate positive altitude. The safety pilot manually lifted the drone off the ground and ensured it was stable in the air. Once the PAFA was activated on both mission devices, the first reconfiguration was carried out to simulate a failure-free scenario as described in chapter 4.1. The first two cycles of movement in the log (yellow area on the left) represent this scenario. After observing the up-down maneuvers, producing two peaks in the graph, the mission device 1 on the drone was manually terminated. Due to this failure, the drone did not follow the sinewave pattern anymore, resulting in the plateu between 20 and 30 seconds in the blue area. Upon detection of this failure by the watchdog on mission device 2, a reconfiguration request was automatically sent to the PAFA Reconfigurator. The new configuration was successfully transmitted to mission device 2. The plateau between 20 and 30 seconds in the blue area area represents the termination of mission device 1, the detection of the failure, and the subsequent reconfiguration of mission device 2. Since mission device 2 operated without any

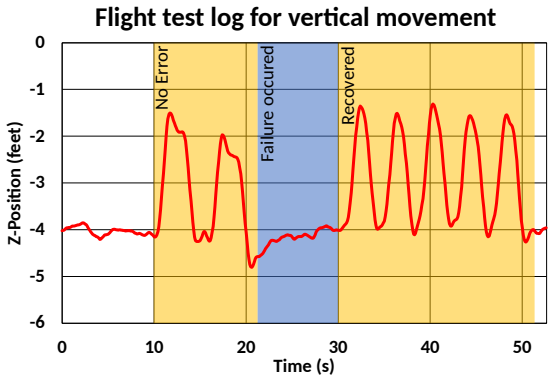
Except where otherwise noted, content of this paper is licensed under a Creative Commons Attribution 4.0 International License.

The reproduction and distribution with attribution of the entire paper or of individual pages, in electronic or printed form, including some materials under non-CC-BY 4.0 licenses is hereby granted by the respective copyright owners.

Fig. 6. The devices used for the flight test. From left to right: Ground Control Station (GCS) laptop, gaming controller connected to the GCS laptop, drone, radio controller for the safety pilot, and Linux laptop.

issues during this time period, other functions remained intact and operational if they were configured and running concurrently. The drone regained its mission capability, as evidenced by the multiple cycles of up-down maneuvers after the plateau period depicted in the log (yellow area on the right).

(a) (b)



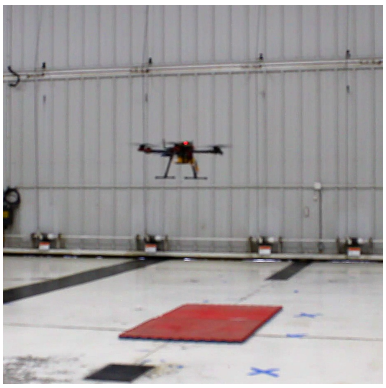


Fig. 7. (a) Flight test log for Z-axis position. The negative values indicate positive altitude due to its coordinate system. Mission device 1 was terminated after two cycles of up and down maneuvers. After a time-out and reconfiguration, the drone successfully resumed the maneuver. (b) Photograph captured during the flight test. A video is published with the following link: `https://youtu.be/MPfDm9BYCQY`

6 Conclusion and Outlook

The collaboration and information sharing among UAVs are expected to play a key role to enhance the mission capability of vehicles and increase failure tolerance. To achieve this, a standardized middleware is essential to facilitate inter-device communication and dynamic reconfiguration of connected devices.

Except where otherwise noted, content of this paper is licensed under a Creative Commons Attribution 4.0 International License.

The reproduction and distribution with attribution of the entire paper or of individual pages, in electronic or printed form, including some materials under non-CC-BY 4.0 licenses is hereby granted by the respective copyright owners.

PAFA enables easy device connectivity and seamless in-flight reconfiguration. This allows for self-detection and self-healing of the entire system on the PAFA platform.

In this article, an in-flight reconfiguration of two interconnected mission critical devices equipped with PAFA is demonstrated. During the demonstration, one device on a drone was terminated and lost its ability to create a new waypoint command. The other devices detected this and was reconfigured to take over the command generation position. As a result, the drone was able to continue its mission.

Although the technical possibility of the application of PAFA to multiple UAVs are performed in this study, several improvements are required to utilize PAFA in the real UAV application field. In this demonstration, PAFA was only applied to guidance function which is a mission relevant function and not for safety critical function such as flight controller. However, as demonstrated in [11], the idea of PAFA can be applied to more safety critical functions. Generally, such functions are scheduled periodically. If the system function needs to meet real-time requirements, one configuration could already account for redundancies, to allow for a seamless continuation of functionality in the event of a failure. The reconfiguration mechanisms by PAFA could then, in a following step, restore redundandacies by utilizing different, still functioning, computing resources.

Another open point is the communication method. In this study, WiFi network was used to facilitate the communication between components, since PAFA is utilized for CAN and communication over IP networks. Depending on timing constraints and safety targets, more reliable or robust communication methods might be required. Moreover, for the real application of swarm drones, another close distance communication must be used which does does not require central network provider.

Also, a sophisticated error detection mechanism is to be developed. In this demonstration, only time-managed error detection was implemented. Among with very low execution frequency (1Hz), it took nearly 9 seconds to conduct the reconfiguration. However, in a safety critical error situation, there is not sufficient time to detect the error. Logical approaches are required to reduce the required time to detect the error.

In the future, the use-case can be expanded to larger swarm of devices, including not only flying vehicles but also ground station devices, thus creating an avionics cloud. This will further increase failure tolerance and enhance overall system performance.

Acknowledgements

The authors would like to thank the members of the Pennsylvania State Unmanned Aerial Research Laboratory for enabling flight tests.

References

[1] Ziquan YU, Youmin ZHANG, Bin JIANG, Jun FU, and Ying JIN. A review on fault-tolerant cooperative control of multiple unmanned aerial vehicles. *Chinese Journal of Aeronautics*, 35(1):1–18, 2022. ISSN: 1000-9361. DOI: https://doi.org/10.1016/j.cja.2021.04.022.

[2] Corey Ippolito, Sungmoon Joo, Khalid Al-Ali, and Yoo Hsiu Yeh. Polymorphic control reconfiguration in an autonomous uav with ugv collaboration. In *2008 IEEE Aerospace Conference*, pages 1–14, 2008. DOI: 10.1109/AERO.2008.4526291.

[3] Anawat Pongpunwattana, Richard Wise, Rolf Rysdyk, and Anthony J. Kang. Multi-vehicle cooperative control flight test. In *2006 ieee/aiaa 25TH Digital Avionics Systems Conference*, pages 1–11, 2006. DOI: 10.1109/DASC.2006.313717.

Except where otherwise noted, content of this paper is licensed under a Creative Commons Attribution 4.0 International License.

The reproduction and distribution with attribution of the entire paper or of individual pages, in electronic or printed form, including some materials under non-CC-BY 4.0 licenses is hereby granted by the respective copyright owners.

[4] Xiangwang Hou, Zhiyuan Ren, Jingjing Wang, Shuya Zheng, Wenchi Cheng, and Hailin Zhang. Distributed fog computing for latency and reliability guaranteed swarm of drones. *IEEE Access*, 8:7117–7130, 2020.

[5] Bjoern Annighoefer, Johannes Reinhart, Matthias Brunner, and Bernd Schulz. The concept of an autonomic avionics platform and the resulting software engineering challenges. In *2021 International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, pages 179–185, 2021. DOI: 10.1109/SEAMS51251.2021.00031.

[6] Bjoern Annighoefer, Johannes Reinhart, Matthias Brunner, and Bernd Schulz. Requirements and concept for a self-organizing plugfly avionics platform. In *2021 IEEE/AIAA 40th Digital Avionics Systems Conference (DASC)*, pages 1–10, 2021. DOI: 10.1109/DASC52595.2021.9594505.

[7] Jinwen Hu, Lihua Xie, and Jun Xu. Vision-based multi-agent cooperative target search. In *2012 12th International Conference on Control Automation Robotics Vision (ICARCV)*, pages 895–900, 2012. DOI: 10.1109/ICARCV.2012.6485276.

[8] Juan Jesús Roldán-Gómez, Eduardo González-Gironda, and Antonio Barrientos. A survey on robotic technologies for forest firefighting: Applying drone swarms to improve firefighters' efficiency and safety. *Applied Sciences*, 11(1), 2021. ISSN: 2076-3417. DOI: 10.3390/app11010363.

[9] Bardienus P. Duisterhof, Shushuai Li, Javier Burgués, Vijay Janapa Reddi, and Guido C. H. E. de Croon. Sniffy bug: A fully autonomous swarm of gas-seeking nano quadcopters in cluttered environments. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 9099–9106, 2021. DOI: 10.1109/IROS51168.2021.9636217.

[10] J.O. Kephart and D.M. Chess. The vision of autonomic computing. *Computer*, 36(1):41–50, 2003. DOI: 10.1109/MC.2003.1160055.

[11] Matthias Brunner, Johannes Reinhart, Bernd Schulz, Erik Preissing, Stefan Moennikes, and Bjoern Annighoefer. Hardware-independent self-discovery of peripherals and modules of a self-adaptive avionics platform. In *2022 IEEE/AIAA 41st Digital Avionics Systems Conference (DASC)*, pages 1–10, 2022. DOI: 10.1109/DASC55683.2022.9925770.

[12] Bjoern Annighoefer, Matthias Brunner, Julian Schoepf, Bastian Luettig, Matthieu Merckling, and Peter Mueller. Holistic ima platform configuration using web-technologies and a domain-specific model query language. In *2020 AIAA/IEEE 39th Digital Avionics Systems Conference (DASC)*, pages 1–10, 2020. DOI: 10.1109/DASC50938.2020.9256726.

[13] Bjoern Annighoefer, Marc Riedlinger, and Oliver Marquardt. How to tell configuration-free integrated modular avionics what to do?! In *2017 IEEE/AIAA 36th Digital Avionics Systems Conference (DASC)*, pages 1–10, 2017. DOI: 10.1109/DASC.2017.8102064.

[14] Eric Johnson and Sumit Mishra. *Flight Simulation for the Development of an Experimental UAV*. DOI: 10.2514/6.2002-4975.

[15] Eric N. Johnson and Suresh K. Kannan. Adaptive trajectory control for autonomous helicopters. *Journal of Guidance, Control, and Dynamics*, 28(3):524–538, 2005. DOI: 10.2514/1.6271.

Except where otherwise noted, content of this paper is licensed under a Creative Commons Attribution 4.0 International License.

The reproduction and distribution with attribution of the entire paper or of individual pages, in electronic or printed form, including some materials under non-CC-BY 4.0 licenses is hereby granted by the respective copyright owners.