Universidade do Minho
Escola de Engenharia

José Miguel Alves Pires

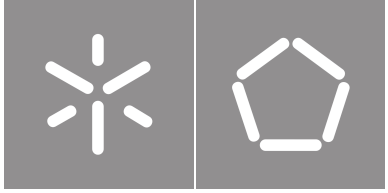
Trustworthy Open-Source Reference Software Stack for UAV applications

June, 2024

Trustworthy Open-Source Reference Software Stack for UAV applications

Jose Pires

UMinho | 2024



Universidade do Minho
Escola de Engenharia

José Miguel Alves Pires

Trustworthy Open-Source Reference Software Stack for UAV applications

Master Thesis
Master in Industrial Electronics and Computers Engineering

Work developed under the supervision of:

Professor Doctor Sandro Pinto

June, 2024

This document was created with the (pdf/Xe/Lua)LaTeX processor and the NOVAthesis template (v7.1.27) [1].

Acknowledgements

Acknowledgments are personal text and should be a free expression of the author.

However, without any intention of conditioning the form or content of this text, I would like to add that it usually starts with academic thanks (instructors, etc.); then institutional thanks (Research Center, Department, Faculty, University, FCT / MEC scholarships, etc.) and, finally, the personal ones (friends, family, etc.).

But I insist that there are no fixed rules for this text, and it must, above all, express what the author feels.

STATEMENT OF INTEGRITY

I hereby declare having conducted this academic work with integrity. I confirm that I have not used plagiarism or any form of undue use of information or falsification of results along the process leading to its elaboration.

I further declare that I have fully acknowledged the Code of Ethical Conduct of the Universidade do Minho.

_____, _____

(Place) (Date)

_____

(José Miguel Alves Pires)

”

“You cannot teach a man anything; you can only help him discover it in himself.”

— Galileo, Somewhere in a book or speach
(Astronomer, physicist and engineer)

Resumo

Pilha de software de referência para aplicações UAV

Independentemente da língua em que a dissertação está escrita, geralmente esta contém pelo menos dois resumos: um resumo na mesma língua do texto principal e outro resumo numa outra língua.

A ordem dos resumos varia de acordo com a escola. Se a sua escola tiver regulamentos específicos sobre a ordem dos resumos, o template (LaTeX) NOVAthesis LaTeX (novathesis) irá respeitá-los. Caso contrário, a regra padrão no template novathesis é ter em primeiro lugar o resumo *no mesmo idioma do texto principal* e depois o resumo *no outro idioma*. Por exemplo, se a dissertação for escrita em português, a ordem dos resumos será primeiro o português e depois o inglês, seguido do texto principal em português. Se a dissertação for escrita em inglês, a ordem dos resumos será primeiro em inglês e depois em português, seguida do texto principal em inglês. No entanto, esse pedido pode ser personalizado adicionando um dos seguintes ao arquivo `5_packages.tex`.

```
\abstractorder(<MAIN_LANG>):={<LANG_1>,...,<LANG_N>}
```

Por exemplo, para um documento escrito em Alemão com resumos em Alemão, Inglês e Italiano (por esta ordem), pode usar-se:

```
\ntsetup{abstractorder={de={de,en,it}}}
```

Relativamente ao seu conteúdo, os resumos não devem ultrapassar uma página e frequentemente tentam responder às seguintes questões (é imprescindível a adaptação às práticas habituais da sua área científica):

1. Qual é o problema?
2. Porque é que é um problema interessante/desafiante?
3. Qual é a proposta de abordagem/solução?
4. Quais são as consequências/resultados da solução proposta?

Palavras-chave: Primeira palavra-chave, Outra palavra-chave, Mais uma palavra-chave, A última palavra-chave

Abstract

Trustworthy Open-Source Reference Software Stack for UAV applications

Regardless of the language in which the dissertation is written, usually there are at least two abstracts: one abstract in the same language as the main text, and another abstract in some other language.

The abstracts' order varies with the school. If your school has specific regulations concerning the abstracts' order, the novathesis (LaTeX) template will respect them. Otherwise, the default rule in the novathesis template is to have in first place the abstract in *the same language as main text*, and then the abstract in *the other language*. For example, if the dissertation is written in Portuguese, the abstracts' order will be first Portuguese and then English, followed by the main text in Portuguese. If the dissertation is written in English, the abstracts' order will be first English and then Portuguese, followed by the main text in English. However, this order can be customized by adding one of the following to the file `5_packages.tex`.

```
\ntsetup{abstractorder={<LANG_1>,...,<LANG_N>}}
\ntsetup{abstractorder={<MAIN_LANG>={<LANG_1>,...,<LANG_N>}}}
```

Concerning its contents, the abstracts should not exceed one page and may answer the following questions (it is essential to adapt to the usual practices of your scientific area):

1. What is the problem?
2. Why is this problem interesting/challenging?
3. What is the proposed approach/solution/contribution?
4. What results (implications/consequences) from the solution?

Keywords: One keyword, Another keyword, Yet another keyword, One keyword more, The last keyword

Contents

List of Figures

List of Tables

Listings

Acronyms

ABI Application Binary Interface *(p. 34)*
ADC Analog to Digital Converter *(p. 29)*
API Application Programming Interface *(pp. 19, 32, 34)*
ASIL Automotive Safety and Integrity Level *(p. 4)*

BLDC Brushless Direct Current *(pp. 17, 24)*
BOM Bill Of Materials *(p. 26)*
BSD Berkeley Software Distribution *(pp. 26, 29, 33, 35)*
BVLOS Beyond Visual Line-Of-Sight *(p. 13)*

CAD Computer-Aided Design *(p. 21)*
CAN Controller Area Network *(pp. 26, 29)*
COTS Commercial Off-The-Shelf *(p. 26)*
CPU Central Processing Unit *(pp. 5, 7, 9, 10)*
CSI Camera Serial Interface *(p. 29)*
CSP Context Security Policy *(p. 32)*

DAL Design/Development Assurance Level *(p. 4)*
DC Direct Current *(p. 17)*
DDoS Distributed Denial Of Service *(p. 19)*
DoS Denial of Service *(pp. 10, 19)*

EEPROM Electrically Erasable Programmable Read-Only Memory *(p. 34)*
ESC Electronic Speed Controller *(p. 17)*

FAA Federal Aviation Administration *(pp. 13, 19)*

FCS Flight Control System *(pp. 18, 19, 23, 24, 26, 31, 35–37)*
FMU Flight Management Unit *(pp. 26, 29)*
FPGA Field-Programmable Gate Array *(pp. 28, 29)*
FPV First-Person View *(pp. 21, 36)*

GCS Ground Control Station *(pp. 16–18, 21, 25, 29, 32, 34, 36)*
GIC Generic Interrupt Controller *(p. 11)*
GNSS Global Navigation Satellite System *(pp. 16, 17, 29)*
GPL GNU Public License *(pp. 26, 34)*
GPOS General-Purpose Operating System *(p. 32)*
GPS Global Positioning System *(pp. 16, 17, 20, 21, 24, 26, 29, 34)*
GSI Generic Serial Interface *(p. 29)*

HAL Hardware Abstraction Layer *(pp. 32, 34)*
HAP High Altitude Platform *(p. 14)*
HDL Hardware Description Language *(p. 26)*
HFC Hydrogen Fuel Cell *(p. 14)*
HW Hardware *(pp. 2, 4, 5, 7–11, 16, 19, 21–23, 26, 27, 29, 32, 33)*

I/O Input/Output *(pp. 8, 10, 11, 24–26, 29)*
I2C Inter-Integrated Circuit *(pp. 26, 28, 29)*
IMU Inertial Measuring Unit *(pp. 16, 17, 24, 26, 28, 29)*
IOMMU Input/Output Memory Management Unit *(p. 8)*
ISA Instruction Set Architecture *(p. 10)*

LAP Low Altitude Platform *(p. 14)*
LiPo Lithium Polymer *(p. 14)*
LLC Last-Level Cache *(pp. 10, 11)*
LOS Line-Of-Sight *(p. 24)*
LTE Long-Term Evolution *(p. 29)*

MCS Mixed-Criticality System *(pp. 2, 4–7, 9, 10, 37)*
MCU Micro Controller Unit *(pp. 26, 28)*
MMU Memory Management Unit *(p. 8)*

NASA National Aeronautics and Space Administration *(p. 13)*
novathesis NOVAthesis LaTeX *(pp. vi, vii)*

OS Operating System *(pp. 5–10, 19, 26, 32, 34, 35)*
OSH Open-Source Hardware *(pp. 25, 26, 34)*
OSS Open-Source Software *(pp. 32, 34, 35)*
OTA Over-The-Air *(p. 2)*

PCB Printed Circuit Board *(p. 26)*
PID Proportional Integrative Derivative *(p. 24)*
PV Photovoltaic *(p. 14)*
PWM Pulse-Width Modulation *(pp. 26, 29)*

QoS Quality of Service *(p. 7)*

RAM Random Access Memory *(pp. 26, 29)*
RC Radio-Controlled *(pp. 18, 24, 29, 34)*
RID Remote IDentifier *(p. 19)*
ROS Robotic Operating System *(pp. 19, 32, 33)*
RT Real-Time *(pp. 5–7)*
RTC Real-Time Communication *(p. 29)*
RTOS Real-Time Operating System *(pp. 32, 35–37)*
RTPS Real-Time Publish Subscribe *(pp. 32, 34)*

SD Storage Disk *(p. 29)*
SDK Software Development Kit *(pp. 14, 19, 32, 35)*
SDR Software-Defined Radio *(p. 21)*
SIL Safety Integrity Level *(p. 4)*
SoC System on Chip *(pp. 28, 29)*
SPH Static Partitioning Hypervisor *(p. 2)*
SPI Serial Peripheral Interface *(pp. 26, 28, 29)*
SRAM Static Random Access Memory *(p. 26)*
SW Software *(pp. 2, 4, 5, 7, 10, 16, 19, 21, 22, 26, 28, 29, 31, 32, 34, 36)*
SWaP-C Size, Weight, Power and Cost *(pp. 2, 4)*

TCB Trusted Computing Base *(pp. 10, 11)*
TEE Trusted Execution Environment *(p. 10)*
TLB Translation Lookaside Buffer *(p. 11)*
TOF Time-of-Flight *(p. 17)*

UART Universal Asynchronous Receiver-Transmitter *(pp. 26, 28, 29)*
UAS Unmanned Aerial System *(pp. 1, 11)*
UAV Unmanned Aerial Vehicles *(pp. 1–4, 11–24, 29, 31–37)*
UDP User Datagram Protocol *(p. 29)*
UGV Unmanned Ground Vehicle *(p. 11)*
UI User Interface *(p. 18)*
USB Universal Serial Bus *(pp. 26, 28, 29)*
USV Unmanned Surface Vehicle *(p. 11)*
UTM UAS Traffic Management *(p. 13)*
UUV Unmanned Under water Vehicle *(p. 11)*
UV Unmanned Vehicle *(pp. 11, 33, 34)*

VM Virtual Machine *(pp. 5–11, 37)*
VMM Virtual Machine Monitor *(p. 7)*
VPN Virtual Private Network *(p. 29)*
VTOL Vertical Take-Off and Landing *(p. 13)*

WCET Worst-Case Execution Time *(pp. 2, 5)*

XML eXtensible Markup Language *(p. 34)*

1 Introduction

"To be, or not to be, that is the question."
William Shakespeare

Unmanned Aerial Vehicles (UAV)s, Unmanned Aerial System (UAS)s, or more commonly drones, are a class of unmanned robotic vehicles that can execute flying missions and carry payloads, guided either by remote control stations or in an autonomous way [2, 3].

The Unmanned Aerial Vehicles (UAV)s market is booming. In 2017 the North America market had a revenue of 737 million USD dollars and a eight-fold increase is expected for 2026 with a staggering 6.7 billion USD dollars, with strong contributions from the sectors of agriculture and farming, and security and law enforcement [4].

The versatility and utility of Unmanned Aerial Vehicless (UAVs) are well displayed is by its wide range of applications, performing tasks with high added value and that would be somewhat hard or impossible for a person to achieve: rescue operations and saving lifes, agriculture and farming, building structures, pipeline inspections, delivering goods and medical supplies, video capturing and filming, surveying, inventory management, providing telecommunications in remote areas, among others [2].

However, only recently regulations have been explicitly enforced on UAVs, with many countries allowing drones to fly over populated areas (at altitudes lower than 150 m) [5]. This poses several safety and security concerns. First, because a drone's crash, unintentional or not, can severely harm people and goods. Secondly, because they are cyber-physical systems which are able to record sensitive and private information. Thus, it is critical to ensure the security of an UAV system.

Furthermore, UAV's applications have real-time constraints but with mixed criticality levels, i.e., the risk of failure is more severe in some case than others. For example, lets consider an automotive infotainment system that displays navigation and rear-view camera information. The navigation information is a commodity for the user, and if, for example, the radio volume suddenly increases, although uncomfortable, it poses no safety risk for the user. On the other hand, the rear-view camera information assists the user in parking maneuvers, and, if a failure occurs, the vehicle may collide causing damage or harming people.

This is an example of a Mixed-Criticality System (MCS), i.e., a system comprised of computer Hardware (HW) and Software (SW) that executes several tasks/applications of different criticality (safety) levels. Furthermore, as this example illustrates, an increasingly relevant trend in the design of real-time and embedded systems is the integration of mixed-criticality components onto a common hardware platform while trying to meet the stringent Size, Weight, Power and Cost (SWaP-C) and safety requirements provided by safety-critical standards, such as those for automotive [6], avionics [7], and railway [8] industries.

As a result of this integration complexity increases, requiring higher computational power, and thus, these HW platforms are migrating from single cores to multi-cores and in the future many-core architectures [9]. This leads to the fundamental question of how to reconcile the conflicting requirements of *partitioning* for (safety) assurance and *sharing* for efficient resource usage. Consequently, problems arise in the modelling, design, implementation, and verification of the required HW and SW [9].

Two major areas of research stem from this dichotomy: a more theorethical one, largely focused on using criticality-specific Worst-Case Execution Times (WCETs) to schedule systems at each criticality level, but at the expenses of high processor utilization; a more practical one, focused on the safe partitioning of a system with the sharing of computational and communication resources, but with increased complexity. Unfortunately, the combination of both areas is not easy: flexible scheduling requires, at least, dynamic partitioning, whereas certified systems require complete separation or, at least, static partitioning. This mismatch needs to be addressed in future work [9].

1.1 Goals

The main goal of the present work is the development of a trustworthy open-source software stack for UAV applications, where security and safety are paramount, thus closing the gap between open-source and commercial solutions and contributing for the widespread adoption of secure and safe features.

To this end several main objectives have been outlined:

1. Identify the UAV application requirements and select the target HW platform with several security primitives (e.g, virtual memory, memory protection, TrustZone, criptographic accelerators, etc.).

2. Design an open-source SW software stack using as reference a Static Partitioning Hypervisor (SPH), namely Bao.

3. Provide support for autonomous mode (autopilot).

4. Implement the device drivers for UAV's control.

5. Perform the remote update of the system, through an Over-The-Air (OTA) mechanism.

6. Assess and demonstrate the security advantages of the UAV's prototype over others open-source competitors.

1.2 Document structure

This thesis is organised as follows: In Chapter 2, some background is provided concerning mixed criticality systems, its relevance and the challenges it poses, alongside with the current approach to address its complexity. Next, UAVs are discussed, namely the reference hardware and software, both as open-source and commercial solutions. Lastly, the related work is presented, showing the current research directions and topics.

2

Background and Related Work

In this chapter some background is provided concerning mixed criticality systems, its relevance and the challenges it poses, alongside with the current approach to address its complexity. Next, Unmanned Aerial Vehicles (UAV) are discussed, namely the reference hardware and software, both as open-source and commercial solutions, and a gap analysis is performed. Lastly, the related work is presented, showing the current research directions and topics.

2.1 Mixed criticality systems

Embedded systems have stringent requirements both on timing and criticality of the tasks it performs. The former concerns the task's timing deadlines requirement (firm, soft, hard), whereas the latter is a designation of the level of assurance against failure needed for a system component [9]. Typical names for the criticality levels are Automotive Safety and Integrity Levels (ASILs), Design/Development Assurance Levels (DALs), and Safety Integrity Levels (SILs) [9].

For example, lets consider an automotive infotainment system that displays navigation and rear-view camera information. The navigation information is a commodity for the user, and if, for example, the radio volume suddenly increases, although uncomfortable, it poses no safety risk for the user. On the other hand, the rear-view camera information assists the user in parking maneuvers, and, if a failure occurs, the vehicle may collide causing damage or harming people.

This is an example of a MCS, i.e., a system comprised of computer HW and SW that executes several tasks/applications of different criticality (safety) levels. Furthermore, as this example illustrates, an increasingly relevant trend in the design of real-time and embedded systems is the integration of mixed-criticality components onto a common hardware platform while trying to meet the stringent SWaP-C and safety requirements provided by safety-critical standards, such as those for automotive [6], avionics [7], and railway [8] industries.

As a result of this integration complexity increases, requiring higher computational power, and thus, these HW platforms are migrating from single cores to multi-cores and in the future many-core architectures [9]. This leads to the fundamental question of how to reconcile the conflicting requirements of

partitioning for (safety) assurance and *sharing* for efficient resource usage. Consequently, problems arise in the modelling, design, implementation, and verification of the required HW and SW [9].

Two major areas of research stem from this dichotomy: a more theorethical one, largely focused on using criticality-specific WCETs to schedule systems at each criticality level, but at the expenses of high processor utilization; a more practical one, focused on the safe partitioning of a system with the sharing of computational and communication resources, but with increased complexity. Unfortunately, the combination of both areas is not easy: flexible scheduling requires, at least, dynamic partitioning, whereas certified systems require complete separation or, at least, static partitioning. This mismatch needs to be addressed in future work [9].

Another important topic in the MCS field is the the incorporation of other shared resources, e.g. communication, particulary on multi-core or many-core platforms: can a shared bus provide the required separation, or is a Network-on-Chip protocol required? These issues are only now beginning to be addressed [9].

2.1.1 Virtualization

Mixed critical systems require the integration of software components with disparate criticality levels on a shared HW platform. As aforementioned, the most promising approach is the safe partitioning of the system with shared computation resources. This leads to concept of virtualization, a logic abstraction of HW resources with isolation guarantees, using different approaches such as hypervisors, Operating System (OS)-level virtualization or unikernels [10].

Fig. 1 depicts a map of concepts related to the virtualization topic, which can be used as reference through this section.

The major distinction between OS- and hypervisor-level virtualization is the former does not provide emulation of the physical HW, where the virtual domain, called container, has its own virtual HW resources (Central Processing Unit (CPU), memory, filesystem, network), and process and user management. This virtual domain acts as an traditional process of an OS, with its own independent resources. OS-level virtualization, or containerization, is mostly used in cloud environments to avoid the replication of the OS stack, further improving application consolidation on the same hardware [10]. Although lacking the HW emulation, this type of virtualization is gaining momentum also in the Real-Time (RT) systems [11, 12] due to some desirable features like tenant isolation, high-availability, fault-tolerance, software migration, and recovery techiques. However, these solutions introduce sources of undeterminism, e.g. dynamic resources assignment, have an extensive code base, and lack security features, difficulting its certification and testing [10].

Another option that is gaining popularity is the unikernel-based virtualization, where it is possible to run a single application in its virtual domain to increase isolation, performance and security. The full software stack of the system (OS components, libraries, language runtime, and application) is compiled into a single Virtual Machine (VM) — unikernel or library OS — which can be run on the physical hardware,

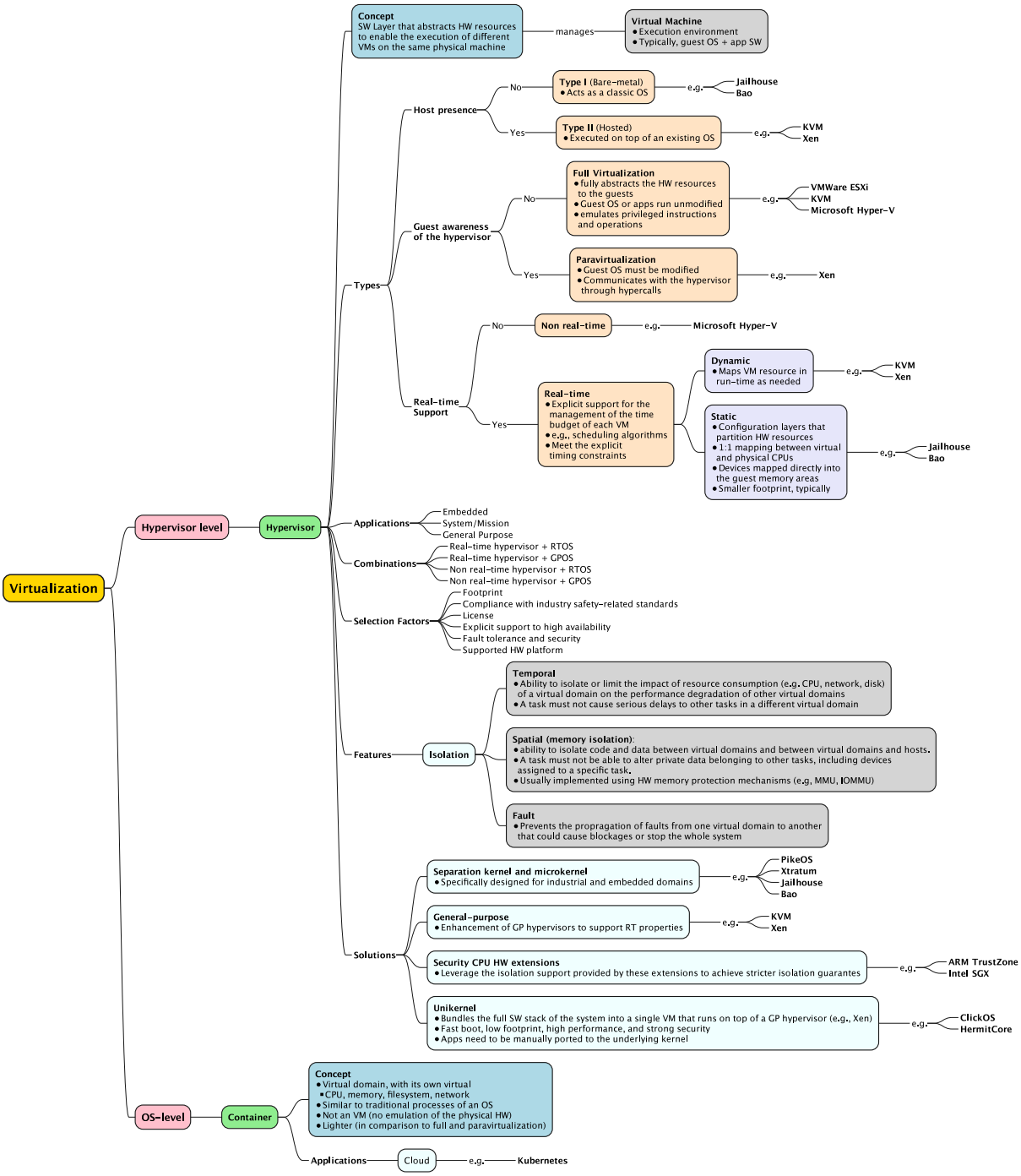


Figure 1: Virtualization mind map

or more commonly, on top of a general-purpose hypervisor [10]. Unikernels provide a fast boot time, low memory footprint, high performance, and strong security by leveraging the underlying host hypervisor. However, they lack flexibility as applications need to be manually ported to the underlying unikernel. These solutions require further studies to enable its adoption in MCSs, especially regarding certification and dependability support [10]. Fig 2 illustrates some examples of virtualization approaches: the application, RT and libraries, and the OS are bundled as a VM and run on top of an hypervisor (hypervisor); a

container is formed only with the first two components and run on top of an OS (OS-level virtualization); all components are compiled into a unikernel and run on top of hypervisor (typically) (unikernel-virtualization).

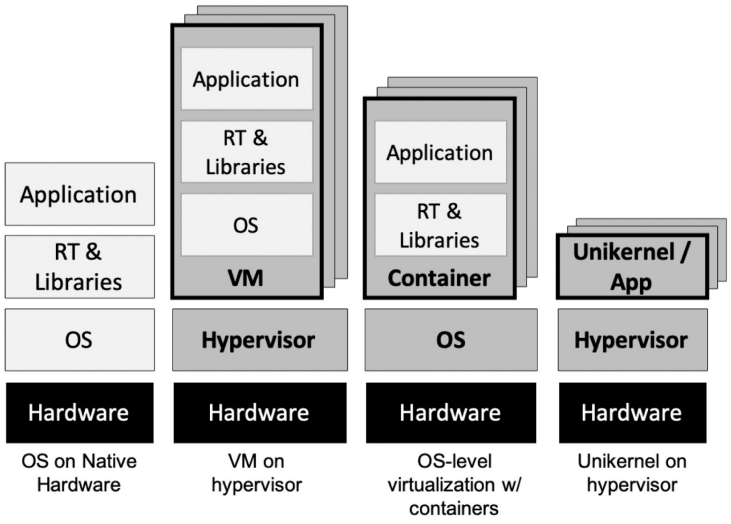


Figure 2: Examples of virtualization approaches [10][1]

Cinque et. al [10] identified the relationship between hypervisor and guest OS combinationsi, i.e. the type of guest that run of a hypervisor, and the associated applications (Fig. 3). On the left quadrants we have the guest OS running on top of a general purpose hypervisor, mainly used for server consolidation in cloud environments (lower-left), and functional testing and prototyping (upper-left). On the right quadrants we have a guest OS on top of a RT hypervisor, mainly used for Quality of Service (QoS) and performance analysis (lower-right), and safety-critical applications (upper-right). The region of interest — the MCS — is located in the intersection of these right quadrants, emphasizing the need of a RT hypervisor. Thus, the hypervisors will be discussed in more detail in the next section.

2.1.2 Hypervisors

In simple terms a hypervisor is a Virtual Machine Monitor (VMM), a software layer that abstracts the HW resources in order to run distinct and isolated application environments, called VMs or guests, on the same physical machine (see Fig. 2). A VM is the execution environment typically comprised of an OS, or guest OS, and the application SW.

A hypervisor must guarantee isolation in the following domains [10]:

- **Temporal**: ability to isolate or limit the impact of resource consumption (e.g. CPU, network, disk) of a virtual domain on the performance degradation of other virtual domains. Thus, a task from one virtual domain must not cause serious delays to other tasks in other virtual domain.

[1]Used with permission from Elsevier: licence nr. 5457890117132
[2]Used with permission from Elsevier: licence nr. 5457890117132

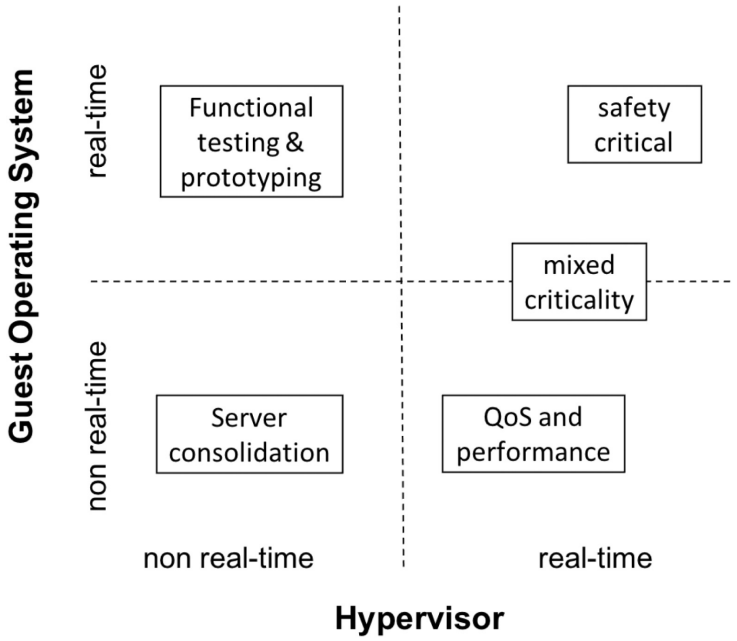


Figure 3: Hypervisor and OS combinations with related applications [10][2]

- **Spatial** (memory-isolation): ability to isolate code and data between virtual domains and between virtual domains and hosts. Thus, a task must not be able to modify private data from other tasks, including devices assigned to a specific task. It is usually implementated using HW memory protection mechanisms (e.g., Memory Management Unit (MMU), Input/Output Memory Management Unit (IOMMU)).

- **Fault**: prevents the propagation of faults from one virtual domain to another that could cause blockages or stop the whole system.

A hypervisor can be classified in numerous ways (see Fig. 1), as follows:

1. **Host presence**: If the hypervisor is executed on top of an existing *host* OS — hosted — is classified as type II, otherwise as type I — bare-metal —, running directly on the HW, acting as a classic OS. The bare-metal hypervisor controls directly the hardware resources (e.g. Jailhouse [13], Bao [14]), whereas the hosted manages it indirectly (e.g. KVM [15], Xen).

2. **Guest awareness of the hypervisor**: A fully virtualized hypervisor abstracts completely the HW resources to the guest, emulating privileged instructions and Input/Output (I/O) operations. Thus, it allows a guest OS or an application to run unmodified, as they were running directly on the physical machine. Typical examples are KVM [15] and Microsoft Hyper-V [16]. Conversely, in a paravirtualized hypervisor (e.g., Xen [17]) the guest OS must communicate with the underlying hypervisor through hypercalls, which is cumbersome, since it requires the guest OS to be modified.

3. **Real-time support**: this refers to the explicit support for the management of the time budget of each VM (e.g. scheduling algorithms) which must meet the explicit timing constraints [10]. These

hypervisors can be further decomposed in dynamic (e.g. KVM[15], Xen [17]) or static (e.g. Jailhouse[13], Bao[14]), depending on the VM resources assignment timing: in run-time, as needed, for the former; in instantiation time for the latter. Static solutions are often employed for MCSs due to higher tolerance to failure and less overhead. Moreover, as they usually have a small code base they are easier to test and certify according to industrial standards [10].

4. **Application type**: embedded, if targeted for a specific application, system, or mission, otherwise they are general-purpose [18].

Solutions based on hypervisors can be grouped in four categories [10]:

1. **Separation kernel and microkernel**: specially designed for industrial and embedded domains. A separation kernel is a special type of a very small bare-metal hypervisor defining fixed VMs and managing information flows, relegating device drivers, user model, shell access and dynamic memory to the guest OS. This simple architecture results in a minimal implementation, ideal for a MCS. Examples are PikeOS [19], Xtratum [20], Jailhouse [13], and Bao [14].

2. **General-purpose**: enhancement of general-purpose hypervisors (e.g. KVM [15], Xen [17]) to support real-time features.

3. **Security CPU HW extensions**: leverage the isolation support provided by these hardware extensions (ARM TrustZone, Intel SGX) to attain stricter isolation guarantees. For example, ARM TrustZone eanbles virtual virtualization thanks to dual world execution model. However, these solutions are strictly linked to the specific platforms. Examples are LTZVisor/RTZVisor [21, 22] and VOSYSMonitor [23, 24].

4. **Unikernel**: runs on top of an hypervisor to enable the execution of a single application in its virtual domain, providing isolation, performance, and security. Examples are ClickOS [25] and HermitCore [26].

The selection factors for a hypervisor with industrial standards are its footprint, the compliance with industry safety-related standard, the software license, the explicit support to high availability, fault tolerance and security, and the supported HW platform [10]. Cinque et. al provides an extensive list of these solutions applied to the MCS in order to meet the industrial standards [10].

2.1.2.1 Bao

Bao (from Mandarin Chinese “bǎohù”, meaning “to protect”) is a security and safety-oriented, lightweight bare-metal hypervisor, developed by the ESRGv3 team at University of Minho, targeting the embedded real-time domain and especially the MCSs for Armv8 and RISC-V platforms. It follows the pioneer static partioning architecture of *Jailhouse* [13], and improves it by discarding the need of the Linux Kernel to boot and manage its VMs. From a security and safety perspective, this dependency compromises

the system by bloating the system Trusted Computing Base (TCB) and intercepting the chain of trust in secure boot mechanisms [14]. The MCSs certification process is also hindered by the size of and monolithic architecture of such OSs.

Jailhouse's breakthrough consists in a minimal software layer that leverages HW-assisted virtualization technology to statically partition all platform resources and assign each one exclusively to a single VM instance at instantiation time. The scheduler can then be discarded as each virtual core is statically associated to a single physical CPU, and the complex semantic services are relegated for the VMs, further decreasing the size and complexity. It provides strong isolation and real-time guarantees but at the expense of efficient resource usage requirement. However, the Linux dependency is a liability for safety/security and performance too.

Furthermore, the static partioning approach is not enough *per se*, due to HW resources sharing across partitions such as Last-Level Caches (LLCs), interconnects, and memory controllers, which breach temporal isolation, hurting performance and determinism [17, 27]. A malicious VM can exploit this by increasing their usage of a share resource (Denial of Service (DoS) attack) or by indirectly accessing other glsvm's data thorugh the implicit timing side-channels [28]. To tackle this issue, some techniques were implemented at both the OS and hypervisor level such as cache partioning (via locking or coloring), and memory bandwidth reservations [14].

Taking this into account Bao was implemented as clean-slate hypervisor (see Fig. 4), comprising only a minimal thin layer of privileged SW leveraging Instruction Set Architecture (ISA) virtualization support to implement static partioning of HW resources. Its most relevant features are: memory is statically assigned using 2-stage translation; I/O is pass-through only; 1:1 mapping of virtual to physical CPUs (no scheduler required); no external dependencies (except for standard platform management SW); provides a basic mecanism for inter-VM communication; Trusted Execution Environment (TEE) support for increased security [14, 29].

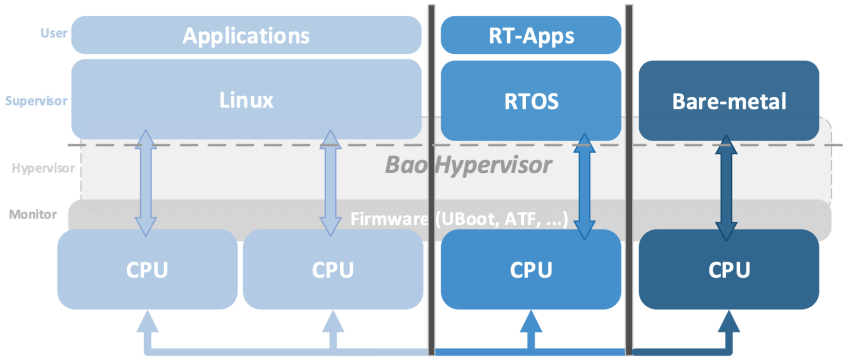


Figure 4: Hypervisor Bao architecture [14][3]

Guest isolation is critical for a MCS. Bao achieves complete logical temporal isolation due to the exclusive CPU mapping, discarding the scheduler, and the availability of per-CPU architectural timers

[3]Used under the terms of the Creative Commons BY 3.0 licence.

directly managed by the guests. Spatial isolation is provided by the 2-stage HW virtualization support for the logical address space isolation. The translation overhead, page table and Translation Lookaside Buffer (TLB) pressure is minimized using superpages, whenever possible,faciliting speculative fetch for potential guest performance improvement. A page coloring mechanism was implemented to enable LLC cache partioning independently for each VM, but at the expenses of memory waste and fragmentation, and increased boot time [14].

The I/O are directly assigned to guests in a pass-through only configuration. For the memory-mapped IO architectures, as the ones supported, it uses the existing memory mechanism and 2-stage translation provided by the virtualization support to implement this for free. A peripheric can be shared among guests, as exclusive assignment is not checked [14].

The interrupt virtualization support is restricted to the Arm Generic Interrupt Controller (GIC)v2, which does not support directly interrupt delivery to guest partitions. All interrupts are dispatched to the hypervisor, which must re-inject it in the VM using a limited set of pending registers, leadinf to unavoidable increase in both interrupt latency and the complexity of interrupt management code [14]. This was fixed in the newest version of the specification GICv4 which bypasses the hypervisor for guest interrupt delivery [30].

It is also noteworthy to mention that Xen has recently introduced *Dom0-less*, eliminating the Linux dependency to boot and execute its hypervisor and VMs. Nonetheless, Bao currently provides the same features but with a smaller TCB and with clean security features. Moreover, and although being in its infancy, preliminary evaluation demonstrate only minimal virtualization overhead [14]. Lastly, Bao is open-source [31] due to the developing team's strong belief that security requires transparency. For all the aforementioned reasons, Bao is the hypervisor of choice to use in this work. Tab. 1 provides a summary of the Bao hypervisor.

2.2 Unmanned Aerial Vehicles

Unmanned Aerial Vehicles (UAV)s, Unmanned Aerial System (UAS), or more commonly drones, are a class of unmanned robotic vehicles that can execute flying missions and carry payloads, guided either by remote control stations or in an autonomous way [2, 3]. They belong to a broader class of Unmanned Vehicles (UVs), alongside with Unmanned Ground Vehicles (UGVs), Unmanned Surface Vehicles (USVs) (e.g., boats) and Unmanned Under water Vehicles (UUVs) [3].

UAVs date back to as far as the 18^{th} century. In 1783, France, the first uncrewed aircraft — a hot air balloon — was publicly displayed [2, 32]. In 1896, Alfred Nobel created the first camera-based UAV (rocket) and launched it. In 1935, U.K., the first modern UAV was developed, a low cost radio controlled aircraft. The term *drone* was presumably coined by Lieutenant Commander Fahrney, who as in charge of U.S. Navy program *Radio Controlled Aircraft* [32]. The first UAV designed for surveillance and scouting was developed in 1973 in Israel, and the *Gulf War* was the first conflict utilizing UAVs. Starting from 2003,

Table 1: Bao hypervisor summary

Architecture	**Supported platforms**	**Size**	**License**	**Version**
Type I, Static	Armv8 RISC-V (experimental)	Small ∼5 kLOC C + ∼500 LOC ASM	GPLv2	0.1.1.
Features	**Isolation**			
Static partioning IO pass-thorugh only 1:1 mapping of virtual to physical CPUs (no scheduler required) Virtual interrupts directly mapped to physical ones No external dependencies (except for standard platform management firmware) Provides a basic mecanism for inter-VM communication Trusted Execution Environment support for increased security State-of the art partioning mechanisms to be implemented (e.g., memory throttling)	**Spatial**		**Temporal**	
	Provided by a 2-stage translation HW virtualization support Translation overhead is minimized using superpages whenever possible, faciliting speculative fetch for potential guest performance improvement It uses cache coloring to enable LLC cache partioning independently for each VM, but at the expenses of memory waste and fragmentation and increased boot time		Exclusive CPU assignment discards the scheduler Per-CPU architecture timers are available to and directly managed by the guests	
IO	**Interrupts**		**Related Work**	
Directly assigned to guest in a pass-through only IO configuration For memory-mapped IO it uses the existing memory mechanism and 2-stage translation provided by the virtualization support A peripheric can be shared among guests (exclusive assignment is not checked)	Currently supports only Arm GICv2 Interrupts are forwarded to the hypervisor which must re-inject them in the VM using a limited set of pending registers Fixed in GICv4 which bypasses the hypervisor for guest interrupt delivery		Jailhouse: pioneer in the static partioning adopted by Bao but requires the Linux Kernel to boot Xen Dom0-less: eliminates the Linux dependency to boot and execute Bao provides the same features from the previous ones, but has a smaller TCB and implements clean security features	

the UAV commercial market started to emerge with the release of the *Amazon Prime* [2]. However, it was until 2006 that the UAVs were first permitted in the U.S. civilian space. More recently, in 2010, the first smartphone controlled quadcopter was developed, and in 2013 camera equipped UAVs entered the consumer market [33].

From then on, the selling price dropped significantly, alongside with the emergence of some of the first open-source projects on UAV control — ArduPilot in 2008 [34] and Dronecode [35] (now PX4) in 2011 — led to a boom in the commercial UAV market. In 2017 the North America market had a revenue of 737 million USD dollars and a eight-fold increase is expected for 2026 with a staggering 6.7 billion USD dollars, with strong contributions from the sectors of agriculture and farming, and security and law enforcement [4].

The versatility and utility of UAVs are well displayed is by its wide range of applications, performing tasks with high added value and that would be somewhat hard or impossible for a person to achieve: rescue operations and saving lifes, agriculture and farming, building structures, pipeline inspections, delivering goods and medical supplies, video capturing and filming, surveying, inventory management, providing telecommunications in remote areas, among others [2].

However, only recently regulations have been explicitly enforced on UAVs, with many countries allowing drones to fly over populated areas (at altitudes lower than 150 m) [5]. For example, in 2019, the

E.U. stated that all drones under 25 kgs are able to fly without prior authorization under some constraints. On the other hand, it imposed that all drones must register in their respective states, and that each state must define no-fly zones where drones are forbidden to enter [36]. Broadly, five important categories must be considered when analyzing the UAV regulations [37, 38]:

1. **Applicability**: refers to the applicable scope of UAV regulations, typically including type, weight, and role of the UAV;
2. **Operational limitations**: restricts the locations for UAV operations.
3. **Administrative and legal requirements**: set of rules and regulations to monitor the use of UAVs.
4. **Technology specifications/requirements**: mechanical, communications and control capabilities that ensure its safe operation.
5. **Moral and ethics**: refers to the privacy and security of people in general.

Furthermore, no global regulation for UAV standardization exists. Thus, in 2020 Federal Aviation Administration (FAA) in collaboration with National Aeronautics and Space Administration (NASA) and other agencies published the UAS Traffic Management (UTM) 2.0, describing protocols for enabling multiple, Beyond Visual Line-Of-Sight (BVLOS) drone operations with the same airspace.

2.2.1 Classification

An UAV can be classified in multiple ways:

- **Thrust forces & Flight principles**: UAVs can be lighter than air, e.g., a balloon or a blimp if it uses a motorized propulsion system. On the other hand, if they are heavier than air, we have gliders, rotor-crafts, and birds [4].
- **Airframe**: probably the most noticeable external feature of a UAV is its chassis, or airframe. Several airframes exist for different applications from three main types: fixed-wing, rotor, and hybrid. Fixed-wing can travel faster than all other types of UAVs due to its aerodynamics and propulsion system, can carry heavy payloads, and have long autonomy. However, they require takeoff and landing from a runway. They are typically used for power line inspections and aerial mapping. Single rotor and multirotor UAV can hover and do Vertical Take-Off and Landing (VTOL). The single rotor long autonomy, but are expensive, require skilled operators, and are mechanically complex and vulnerable to vibrations. Multirotors, on the other hand, are the cheapest and most manufactured ones, but they typically have low autonomy since they consume more power. They can have a varied number of propelers, such as tricopter, quadcopter, hexacopter, and octocopter [4].

- **Altitude**: Broadly, UAVs are divided into Low Altitude Platforms (LAPs) and High Altitude Platforms (HAPs) . A LAP maximum altitude ranges from 3 to 9 kms and is typically used to support cellular communications. HAPs, on the other hand, are deployed above the 9 km altitude and are used to support cellular communication but with wider coverage, endorsed by companies such as Google and Facebook [4].

- **Overall weight**: UAVs can have few grams of weight or hundreds of kilograms. For example, Australia labels them as *micro* (below 100 g), *very small* (from 100 g to 2 kg), **small** (from 2 to 25 kgs), **medium** (from 25 to 150 kgs), and **large** (over 150 kgs) [2].

- **Power source**: UAVs can be powered using electricity, fuel, or a hybrid solution. Except for the fuel-only powered ones (e.g., Nitro Stingray, and Goliath Quadcopter [39]), all the others use electric motors. These motors can be powered through batteries (e.g., Parrot[40], Dji Mavic 3 [41]), typically Lithium Polymer (LiPo), Hydrogen Fuel Cells (HFCs) (e.g., Energyor H2Quad 1000 [42]), and solar power. The hybrid solutions use fuel + batteries (e.g., Flaperon MX8 [43]) or HFC + batteries, and are a compromise between the two power sources. Batteries typically have the shortest autonomy and are heavy and bulky, while fuel, although not a clean power source, has the highest power density, leading to higher autonomy. The hydrogen fuel cells are a intermediate solution, but are typically more expensive than batteries and have more complex power management.

- **Target audience**: UAVs are developed with a specific target audience in mind, namely the recreational/hobbyist, the commercial, and military one.

- **User-modifiable**: In 2021, 26% of all commercial drones sold were open-source (e.g., Parrot) [44]. There is an increased trend for open-source adoption due to higher transparency, extensability and usage from the end-user perspective, and due to bootstrapping opportunity since the developers can leverage from the ecosystem, e.g., using the flight control algorithms which are hard and costly to develop. On the side of the spectrum, we have proprietary drones (e.g., Dji) with the highest market share. Transparency is compromised and extensibility is typically provided using a Software Development Kit (SDK) [45].

Fig. 5 illustrates the several types of UAVs, focusing on airframe and power source.

The most prominent UAV's characteristics are speed, autonomy, payload, range, and altitude. The speed depends mainly on the propulsion system, aerodynamics, weather conditions and power usage. Typically, a small UAV can reach speeds up to 50 km/h, while a large one can reach speeds up to 360 km/h[4]. The autonomy or endurance, refers to the maximum flight time with a single charge (battery, fuel, or both), varying from a 20–30 minutes (small UAVs) to several hourse (large UAVs). Size, weight and weather conditions have a strong impact on the autonomy. Some technologies are currently being develop to support in-flight recharging through renewalable sources (Photovoltaic (PV) cells) or wireless power techniques (e.g., laser emission)[4]. The payload refers to the lifting capability of the UAV for load

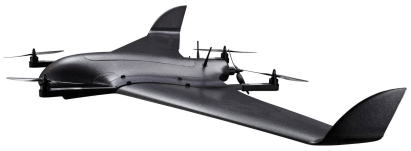
carrying, varying from few grams (small UAVs) to hundreds of kilograms (large UAVs). Most common payloads are sensors and video cameras [4]. The range defines the distance from where the UAV can be controlled remotely and depends on the communication technology and network, and the weather conditions[4]. The altitude is the height a drone can fly, weather by technological constraints or by legal ones [4].

Fig. 6 depicts the UAV's generic overview, summarizing its main concepts.

(a) Firebird: Fixed-wing [46]

(b) Velos: single-rotor [47]

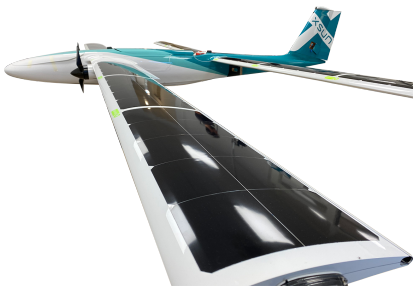
(c) Parrot: multirotor [40]



(d) DeltaQuad Pro: fixed-wing hybrid [48]

(e) Energyor Quad 1000: hybrid (HFC + batteries) [42]

(f) HAMR: hybrid (fuel+batteries) [49]



(g) Solar X One: solar powered [50]

Figure 5: UAV types

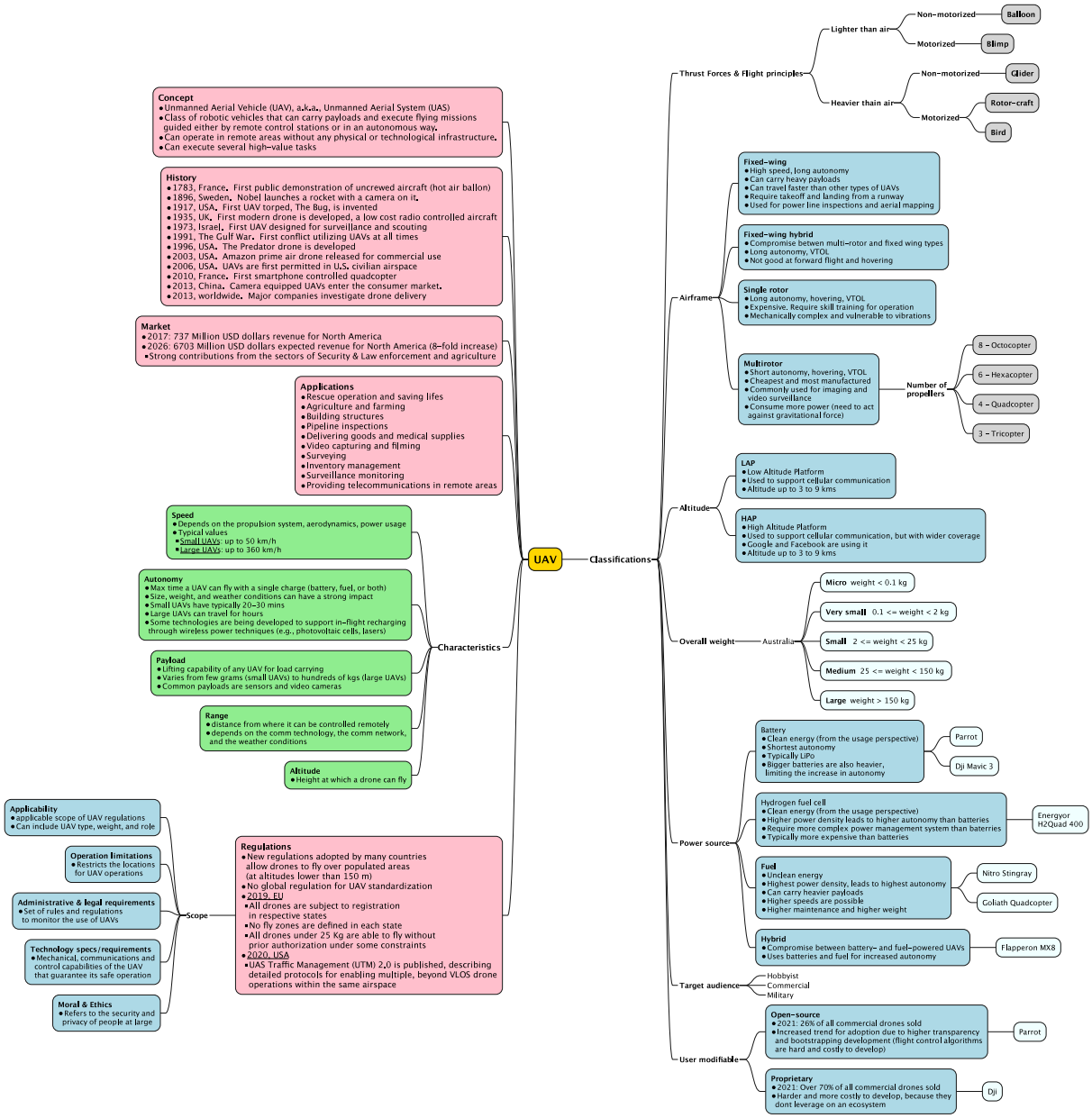


Figure 6: UAV mind map: generic overview

2.2.2 System overview

Fig. 7 shows an overview of the UAV system and its ecosystem. At the center *(1)* we have the UAV and its flight control HW and SW. Its main tasks are path planning, communication management, data acquisition, and mission [51]. The path planning works in combination with the Ground Control Station (GCS) to assist in the navigation, finding the optimal path, while providing environmental awareness through weather and climate monitoring, and controlling the motion and speed for obstacle avoidance. To achieve this, the on-board flight controller collects data from multiple sensors (e.g., ultrasonic/infrared sensors for obstacle avoidance, Inertial Measuring Unit (IMU), barometer, Global Navigation Satellite System (GNSS)/Global Positioning System (GPS) module, etc.) and, together with the commands received

from the GCS, implements attitude estimation and the control law (e.g., Kalman filter) to drive the propulsion system (motors) (2). The flight controller also manages the communications between three types of links [51]: UAV to GCS (5) — radio communication used for transmitting instrument readings such as audio or video and for human remote control (7) (8); UAV to Satellite (4) — carries weather, climate, and GPS information required for accurate UAV navigation; UAV to UAV (6) — can be used for cooperative missions or to provide environmental awareness (e.g., signal danger). The mission refers to the flight's goal, e.g., video capturing, topographic mapping, etc., and is directly related to the payload (3) (e.g., camera, LIDAR, etc.).

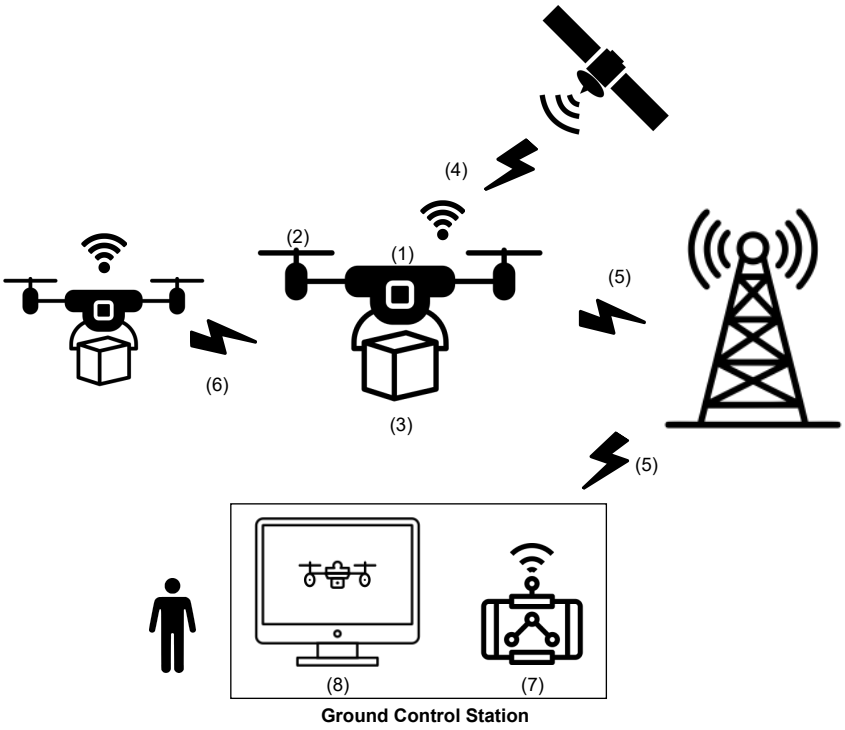


Figure 7: UAV system overview

Sensors can be typically categorized into obstacle avoidance, payload, and navigation [52]. Ultrasonic and infrared sensors are generally used to avoid collisions with obstacle, but Time-of-Flight (TOF) sensors like LIDAR can also be used, although more expensive and more complex. The navigational sensors include the IMU — used to estimate orientation and heading of the vehicle, the barometer — used to estimate of the UAV with a precision of few centimeters, and the GNSS/GPS module — used to estimate the drone's geolocation for navigation and autonomous flight with a precision of up to 5 meters, thus requiring the IMU and barometer to improve accuracy [53].

The actuators depend on the propulsion system, although electronic drives are always used. For a typical multi-rotor UAV, Brushless Direct Currents (BLDCs) motors are used for rotors, with an Electronic Speed Controller (ESC) — an glsac/Direct Current (DC) power converter and high-frequency variable motor speed controller. Fixed-wing hybrids include also servomotors for flaps [54].

Fig. 8 depicts the concept map of UAV's tasks and components.

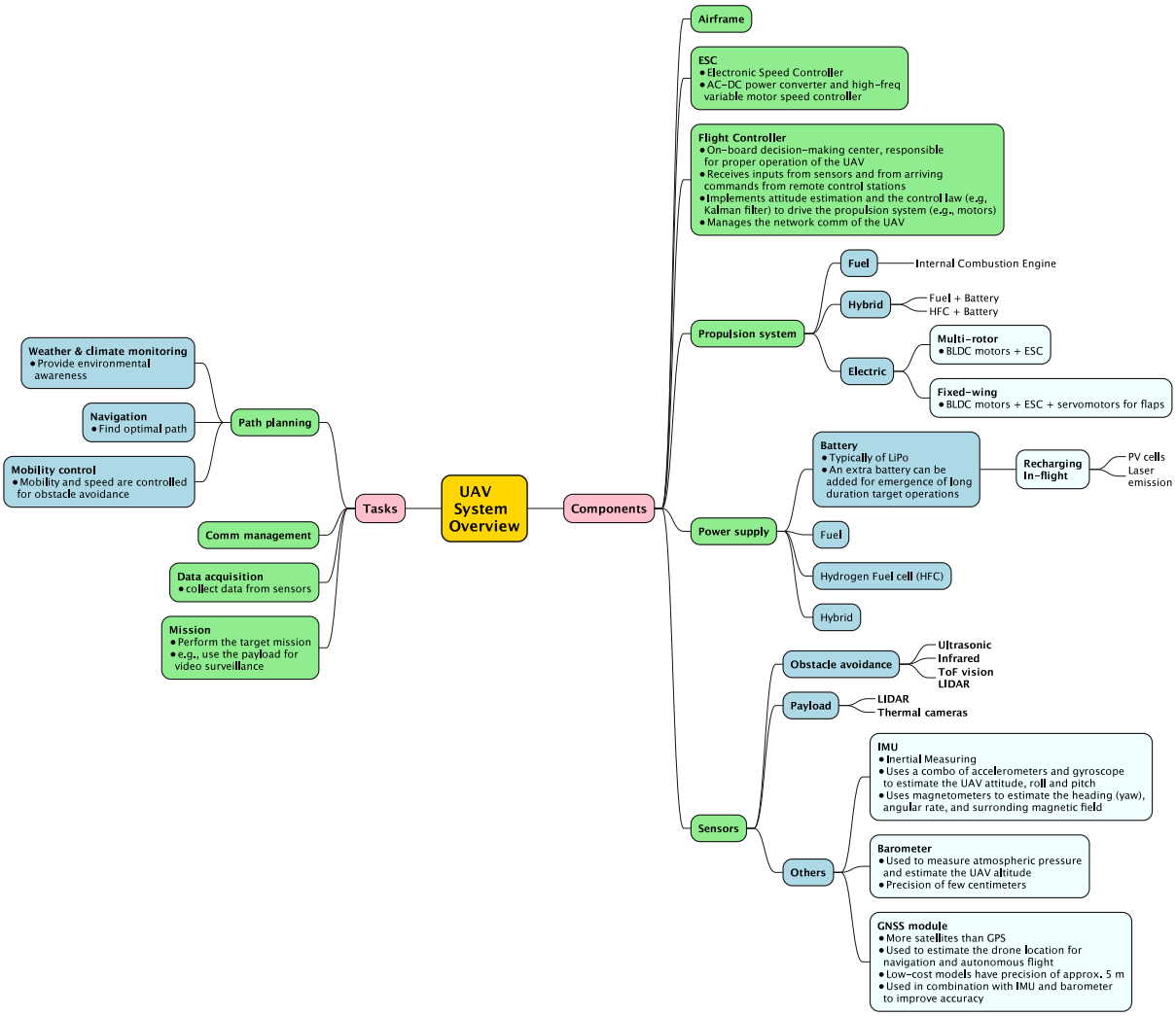


Figure 8: UAV mind map: System overview — Tasks and components

UAVs can be organized into a top-down hierarchy of layers [3], as follows (see Fig. 9):

1. **Flight Supervision**: the topmost layer handles flight's management, safety/authorization, and remote UAV's identification. Flight management is achieved through the transmission of commands from the GCS to the UAV's Flight Control System (FCS). The GCS is software running on ground-control devices, and they typically provide path planning, mapping and real-time flight statistics superimposed on a map. The most notable example is probably `QGroundControl`, which provides full flight control and setup for UAV vehicles, cross-platform, and with a mobile-styled User Interface (UI). It supports actions to be triggered in case of failures (fail-safe features), such as geofencing (limits the permissible fly areas). Safety/authorization handles the authorization to fly (if required) and its safety. Although autopilots, like PX4 and ArduPilot, and the GCS exist, providing several safety features, the pilot in control is ultimately responsible for ensuring the safe flight of any UAV. As such, it must enforce the use of failsafe actions, like in the case of low battery, loss of Radio-Controlled (RC) signal, data link failures, or geofence violations, and request authorization

to fly if legally demanded. For example, pilots can use `LAANC` Application Programming Interfaces (APIs) or `Google Wing OpenSky` to receive FAA authorization for entering controlled airspace. On the other hand these softwares provides the air traffic professionals drones operations' visibility. Remote identification will be required for all UAVs beginning in 2023. Thus, all UAVs must be equipped to a Remote IDentifier (RID), an unique electronic identifier comparable to a vehicle license plate, with direct broadcasting of the RID to anyone within range of signal.

2. **Command & Control**: the second layer ensures drones can fly safely, through the transmission of commands to the UAV, yielded by an human operator of computer-generated via autopilot. The command is sent to the UAV via proprietary protocols or open APIs (e.g., MAVLink SDK or Parrot SDK). The autopilot consists in SW stack running on the FCS. Noticeable open-source autopilots are `PX4`, `ArduPilot`, and `Paparazzi`.

3. **System Simulation**: UAV behavior is analyzed in respect to different environment and conditions, mainly through flight modeling, traffic modelling — models real-world traffic interactions (land and air) — and network communication modelling.

4. **Operating Systems**: OSs are used to interface the electronic HW that controls UAV operation through a tractable abstraction. UAVs can a myriad of OSs, both open-source — such as Robotic Operating System (ROS), `Linux`, `FreeRtos`, `NuttX`, `ChibiOS` — or proprietary, like `VxWorks`.

5. **Physical HW**: comprises the actual electronic HW of the UAV, such as sensors, motors, communications, etc.

2.2.3 Security and Safety

Another very important feature, often overlooked, is the UAV's ecosystem security, and alongside with it safety of people and goods [55]. Security is often not embedded in the system's design, posing all sorts of problems. For example, UAVs often include onboard wireless communication modules that use open, unencrypted, and unauthenticated channels, exposing them to a variety of cyber-attacks [56, 57]. The problem is not restricted to commercial applications, with the U.S. army banning Dji drones — the most widely used ones by the army — for cybersecurity concerns in 2017 [58]. Hacking of drones is another major concern, exposing sensitive information and control of the UAV. In fact, several incidents have been reported in the media where drones were weaponized [59–61], and the volume and risk of such incidents are likely to increase significantly with the expected drone market growth in the foreseable future [4]and the new regulations adopted by many countries which allow drones to fly over populated areas [38].

The most common attacks to the UAV are DoS and Distributed Denial Of Service (DDoS), causing resource availability challenges which can be exploited to drain the batteries, overload the processing units, and flood the communications link, leading to huge services' interruption [4]. But more sophisticated

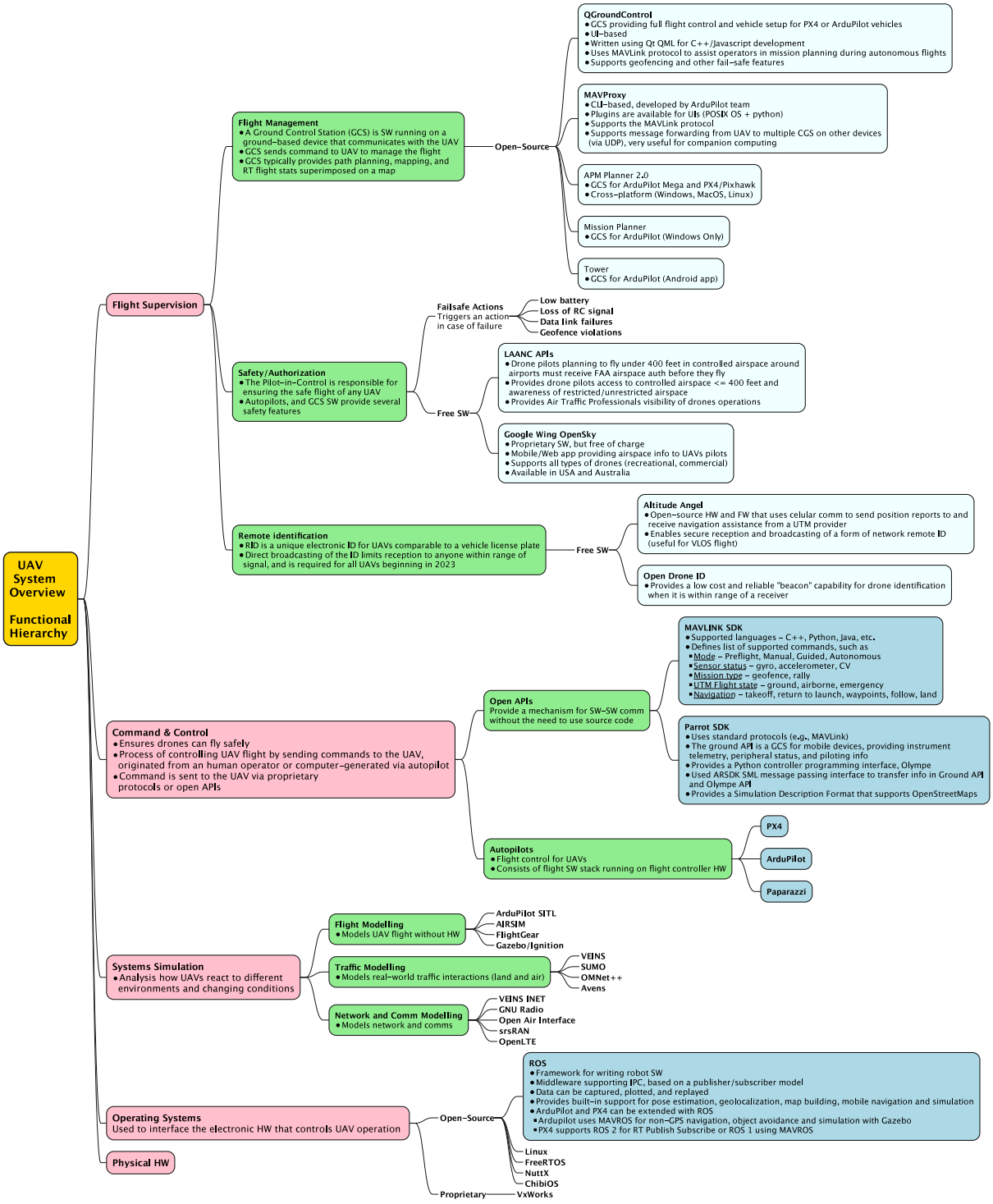


Figure 9: UAV mind map: System overview — Functional Hierarchy

attacks are used too, like GPS spoofing — impersonating as a valid GPS satellite to provide false data, GPS jamming, instrument spoofing (gyroscope, compass), or even killing the main process [5]. Ground control stations are a target too, since the attacker can indirectly obtain access to the UAV, usually through key loggers, viruses, and malwares, and steal all of its data or send malicious and erroneous commands

to the UAV [4].

Nassi et al. [5] conducted a thorough and very extensive survey on the security and privacy issues associated with UAVs, the attacks and countermeasures. The authors divided the security attacks on two categories:

- **Targetting UAVs**: corresponds to an attack an ill-intentioned civilian performs on an UAV using, e.g., an Software-Defined Radio (SDR), a computer or a commercial laser. Six targets were identified — the UAV electronic HW, the UAV chassis and package (e.g., propelers and cargo), the GCS, the First-Person View (FPV) channel, the pilot and cloud services/servers. Attacks on the UAV HW include installation of fake firmware, instrument spoofing and GPS jamming. Countermeasures vary with each attack, but, for example, the installation of fake firmware on the device can be opposed by requiring the digital signature of the firmware before installing. Direct attacks to interrupt UAV's flight control and commiunications links to modify mission parameters can be mitigated through onboard SW and HW mechanisms, such as real-time monitoring, instantaneous estimation of the controller, alert warning and immediate action on any alteration from the intended controller model [4]. Attacks on the chassis and package include fake Computer-Aided Design (CAD) files to undermine its fabrication and deployment, especially critical for open-source HW, nets, and bullets. They can be opposed by requiring digital signature and by using parachutes, respectively. The FPV channel is the radio communication channel between the UAV and the GCS, which enables the pilot to fly as if it was on board and consists of a downlink — used for video streaming — and an uplink — used to control the UAV via the GCS. The most known methods are pilot's deauthentication, taking and dowloading videos and pictures, killing the main process, or remote control of the drone, which can be mitigated through the usage of an encrypted network protocol. Jamming is also an issue and can be remediated using channel hopping. Lastly, cloud services and servers expose the UAV indirectly to hackers, because, although most UAVs are not connected to the Internet, the GCS is, sending the telemetry of flights to cloud servers for storage and analysis. The methods used to explore this link are deanonymizing pilots and extracting flight history, which can be both mitigated through the implementation of authorization and two-factor authentication mechanisms.

- **Targetting people**: UAVs can attack people, on a individual, organization or nation base. The countermeasures include detection and tracking, assessment, and interdiction. Several detection methods exist, such as radars, video and infrared cameras, LIDAR, acoustics, acoustics and optics, etc. However, it is important to note that none of these methods are fail-proof. For example, specific radars have to be used with very specialized operators, since common radars are designed for large aircraft detection and drones uses materials which are not very radio reflective. Another example is the acoustics, where sensors exist to detect the specific spectrum of the drone's propellers, but can be easily circumvented by going into silent (stealth) mode [62]. Assessment concerns the rating and identification of hostile drones, especially important in areas where drones are used

for both legal and illegal activies. The most used methods are based on UAV's classication, to identify the manufacturer and model of the drone. In the event of the detection and assessment of a hostile drone, interdiction must be applied, i.e., the drone should be disabled. For this purpose, several methods can be used like bullets and nets, which are dangerous and not very effective, commercial jammers, predator birds and laser cannons.

Safety failures are also very serious, since they pose risks on the integrity of people and goods. They can be categorized as follows [63]:

- **Damague due to calculation errors**: occurs when a UAV is more dangerous than planned. For example, in 2020 in the U.K., an activist group piloted hundreds of drones with the 3-mile depletion zone to interrupt the flights, which could have been catastrophic.

- **Sensor failures**: UAV sensors do not easily detect delicate objects, such as wires, tree branches, and transparent surface such as buildings windows. Falls by collisions with these types of objects are very common due to undetected obstacle or pilot's excess of confidence.

- **Obstacle deviation**: collisions and crashes caused by object deviation are common, from building to trees.

- **Direct attacks**: these attacks are performed to harm people, for personal or ideological reasons, or to steal the UAV payload, e.g., an organ, food, or vaccines. The recreational/hobbyist UAVs are the most commonly targets.

- **Accidental damage**: can occur when a UAV gets out of control, due to legitimate loss of control by the owner, cyber attack, or SW or HW malfunction. Indepently of the cause, the extent of the damage on people as goods can be severe, due to crash of UAV falling from the sky.

Clearly, security and safety are fundamental aspects of the UAV's operation, but they require more effort to decrease the attack vectors and the attack surface. Ferrão proposed that UAV's design considers simultaneously both aspects, as they strongly influence each other [63]. Fig. 10 illustrates the UAV's security and safety taxonomy.

2.2.4 UAV Reference Hardware

In this section the reference hardware for UAVs is discussed. An overview and the HW architecture is presented. Then, the open-source and commercial solutions are discussed, and compared in the gap-analysis.

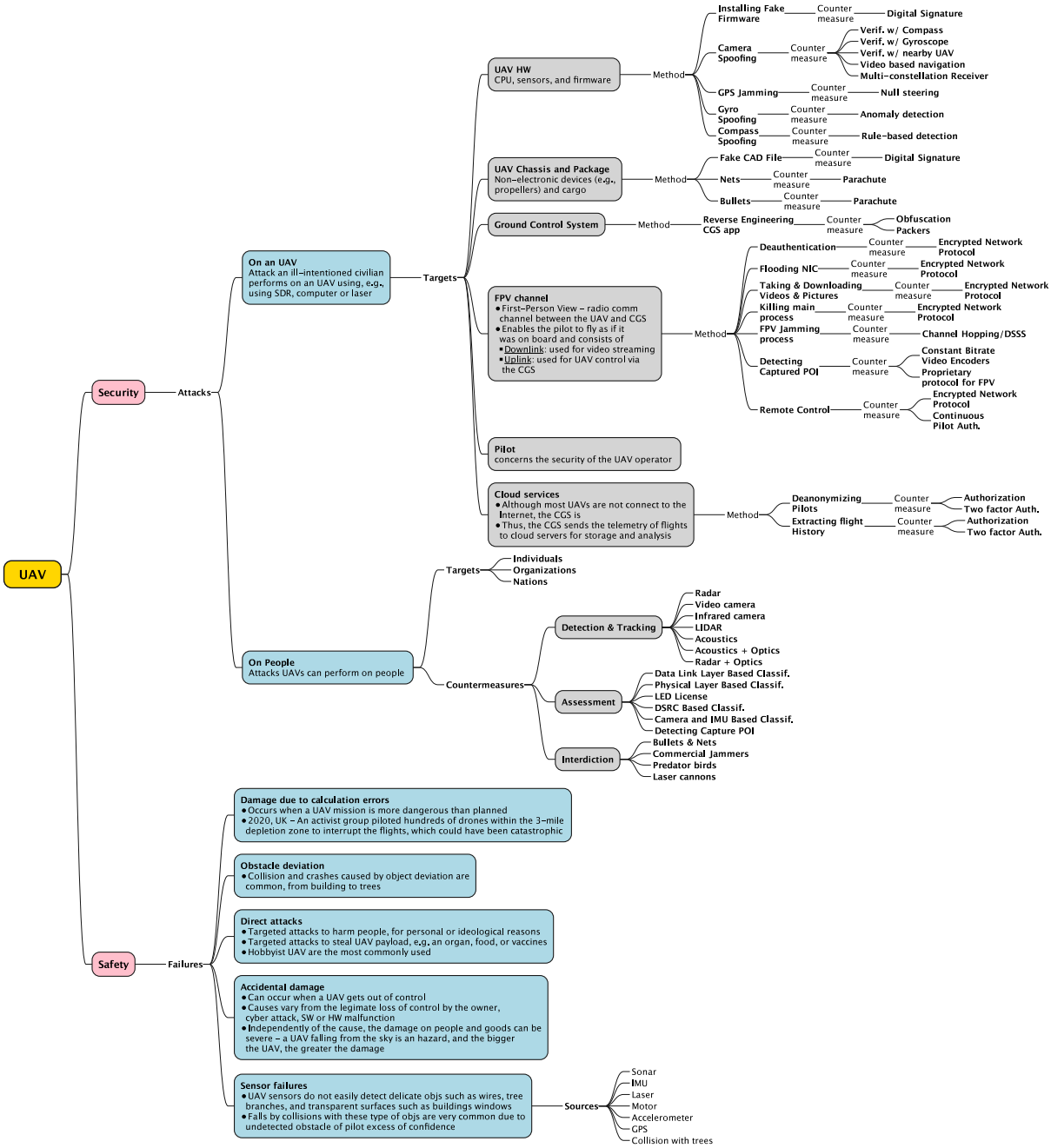


Figure 10: UAV mind map: security and safety

2.2.4.1 Overview and Architecture

Fig. 12 illustrates the high-level abstraction of the UAV HW architecture [55, 64]. In orange, we have the main computing platforms: the flight controller (FCS), e.g., Pixhawk, and, optionally, the companion computer, which provides extra functionalities like navigation — avoidance and collision prevention, mission-related features — associated to the payload, e.g., camera surveillance, topographic mapping, etc. — or telemetry data processing, off-loading the FCS.

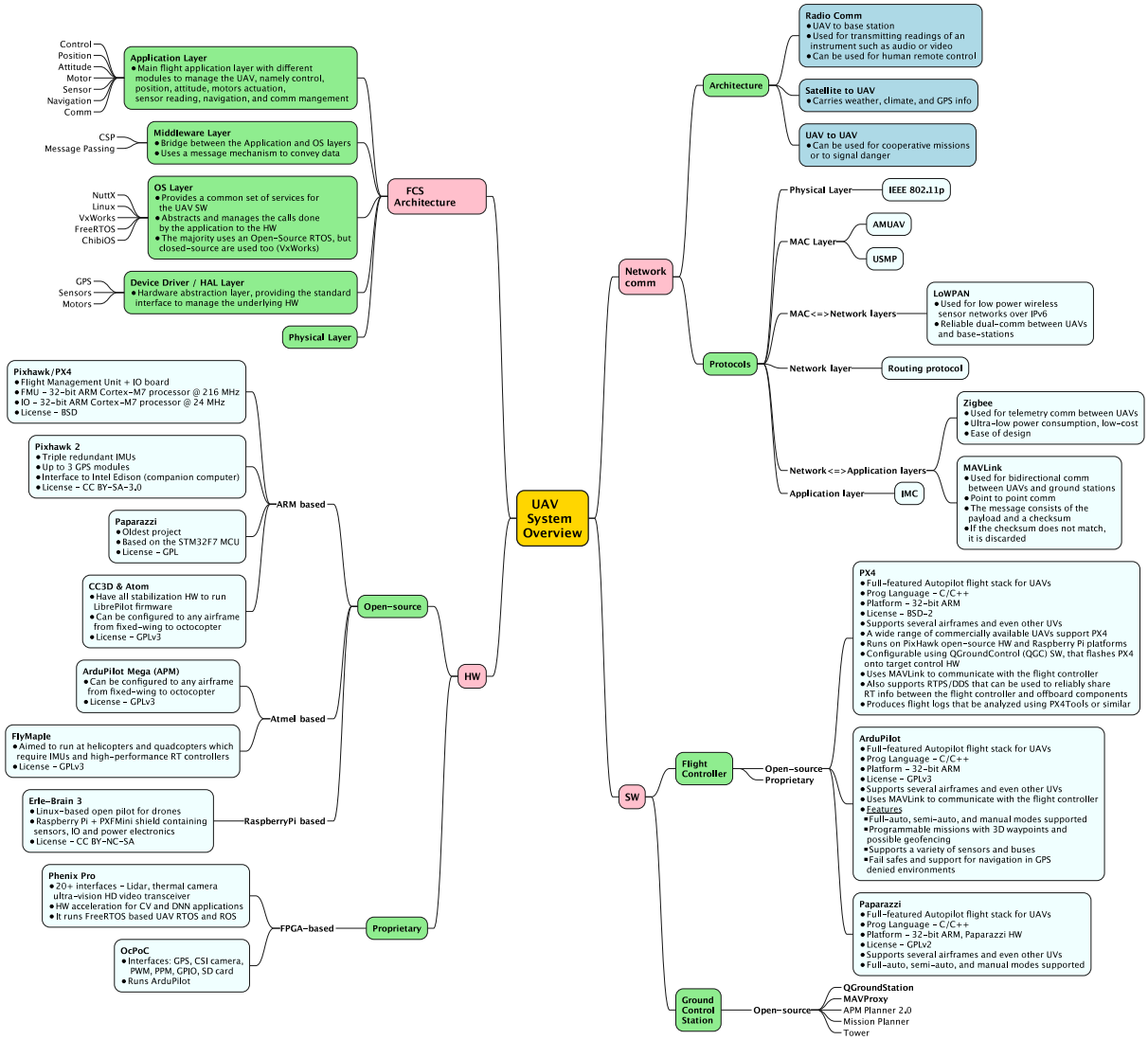


Figure 11: UAV mind map: System overview — Architecture, HW, SW and communications

The FCS is typically composed of a main processor, for UAV control, and a optionally fall-safe coprocessor, dedicated to fail-safe features, i.e., the actions that must be triggered in case of a failure, e.g., returning to base when the battery reaches the low threshold. The main processor handles the communications — commands arriving from the manual RC controller or between the companion computer, the control of the aircraft — typically using Proportional Integrative Derivative (PID) models and/or estimation methods (e.g., Kalman filter), and the I/O — to interface the critical sensors, such as IMU, barometer, gyroscope, and compass, or the actuators, such as BLDC motors and servomotors. The power system supplies the required energy for computing and I/O.

As aforementioned, the flight controller does not explicitly require the companion computer, since the commands sent manually by the pilot through the RC controller are sufficient to control the aircraft flight (Line-Of-Sight (LOS) navigation). However, for autonomous missions — autopilot — the non-critical sensors like the GPS, and obstacle avoidance sensors, are required. In this mode, the companion computer sends

telemetry data to the GCS, which are typically superimposed on a map for straightforward navigation and receives the navigational and mission-related commands, processing them and dispatching the lower level commands to the flight controller or I/O system.

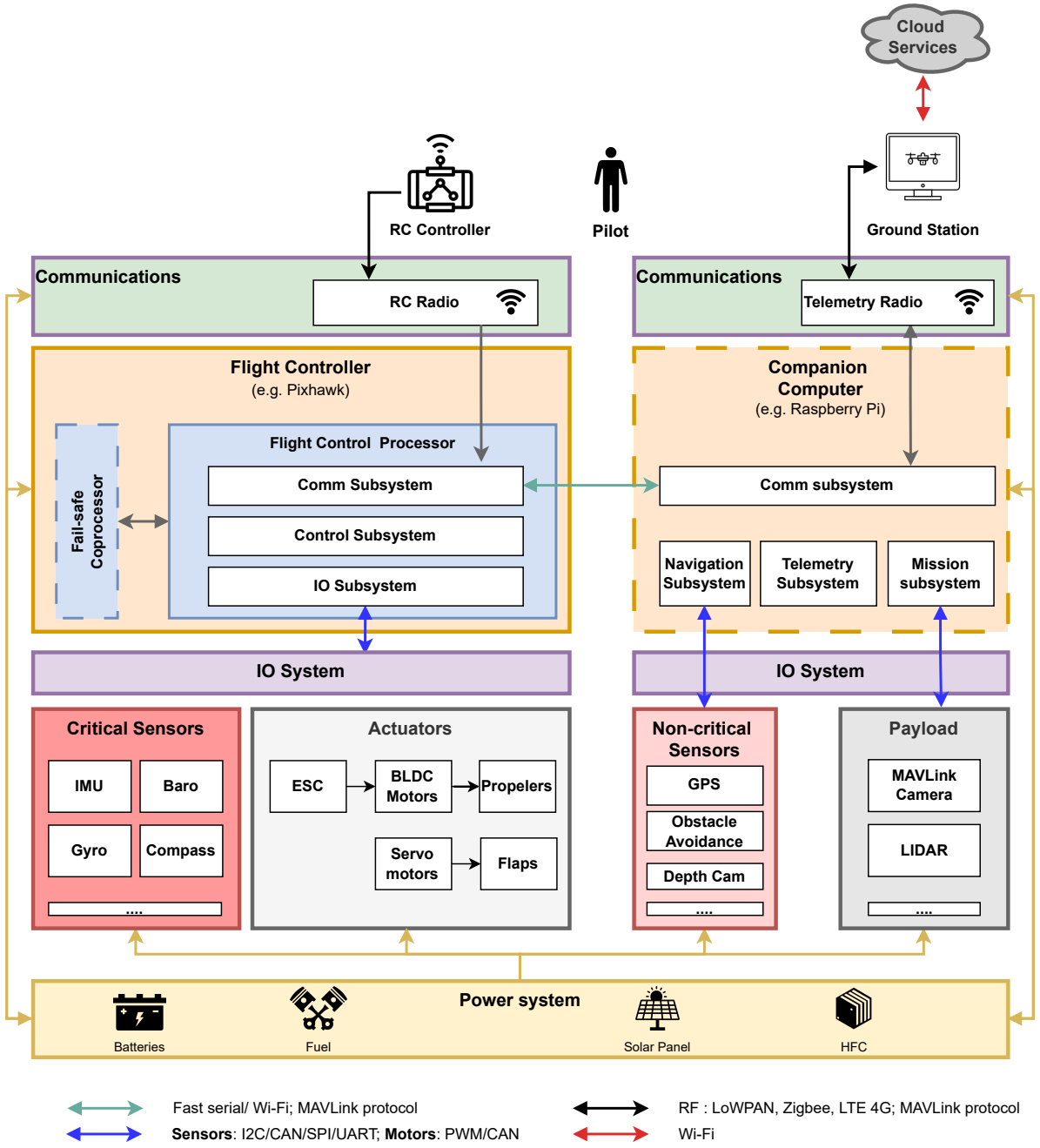


Figure 12: UAV HW architecture: high-level abstraction

2.2.4.2 Open-source solutions

Open-Source Hardware (OSH) solutions have been developed to provide more transparency, extensibility, flexibility, maintenability, while trying to be cost-effective. Open-Source Hardware (OSH) means that the

HW design (i.e., mechanical drawings, schematics, Bill Of Materials (BOM),Printed Circuit Board (PCB) layout data, Hardware Description Language (HDL) source code), and the SW that drives the HW, are all released under free/libre terms [65]. The user can purchase or manufacture the HW components individually or in Commercial Off-The-Shelf (COTS) kits.

The OSH solutions fall into three main categories: ARM-based platforms, Atmel-based platforms, and RaspberryPi-based platforms [53]. However, the currently active OSH projects are based only in ARM platforms, such as, `Pixhawk 4`, `Paparazzi Chimera`, `CC3D`, and `CUAV v5 Plus`, among others.

Pixhawk is a company that develops open standards for drone hardware, providing readily available HW specifications and guidelines for drone development. `Pixhawk 4` is an advanced autopilot designed in collaboration with Holybro HW manufacturer and the PX4 (open-source autopilot SW) teams [66], released under the Berkeley Software Distribution (BSD) license (see Fig. 13). It is optimized to run PX4 v1.7 and laterm and is suitable for academic and commercial developers. It runs PX4 on the NuttX OS [66]. It consists of two 32-bit ARM processors for the FCS (Flight Management Unit (FMU), 32-bit Arm Cortex M7 @ 216 MHz, 2 MB memory, 512 KB Random Access Memory (RAM)) and I/O (32 bit Arm Cortex-M3 @ 24 MHz, 8 KB Static Random Access Memory (SRAM)), respectively [66]. It comes with on-board sensors (accelerometers/gyroscopes, magnetometer, and barometer) and a GPS module. It has Pulse-Width Modulation (PWM), Controller Area Network (CAN) Inter-Integrated Circuit (I2C), Universal Asynchronous Receiver-Transmitter (UART), Serial Peripheral Interface (SPI), and telemetry interfaces (radio and companion computer) [66].

The `Paparazzi Chimera` is an OSH flight controller released under the GNU Public License (GPL) license [67] (see Fig. 14). This flight controller integrates all I/O in the same board, and it uses the ARM Cortex-M7 STM32F767 @ 216 MHz (2 MB Flash, 512 Kb SRAM) Micro Controller Unit (MCU). It comes with on-board sensors (IMU, barometer, and pressure sensure), an XBEE modem holder for communications, and a dedicated serial link and power supply for the companion computer (e.g., Beaglone, RaspberryPi, etc.). It has PWM, CAN I2C, UART, SPI, and servomotors interfaces [66].

The `CC3D` is an OSH flight controller released under the GPL license [68] (see Fig. 15), running the `OpenPilot` firmware. It uses a STM 32-bit MCU @ 90 MHz (128 kB Flash and 20 kB SRAM). It comes with on-board gyroscope and accelerometers, support to serial and Universal Serial Bus (USB) telemetry, up to ten motors, and provides camera stabilization.

`CUAV v5 Plus` is an advanced autopilot supporting the `ArduPilot` firmware [69], released under the BSD license (see Fig. 13). It consists of a 32-bit ARM processor for the FCS (FMU, 32-bit Arm Cortex M7 @ 216 MHz, 2 MB Flash, 512 KB RAM) and a 32-bit IOMCU coprocessor. It comes with on-board sensors (accelerometers/gyroscopes, magnetometer, and barometer) and a GPS module. It has PWM, CAN I2C, UART, SPI interfaces [69].

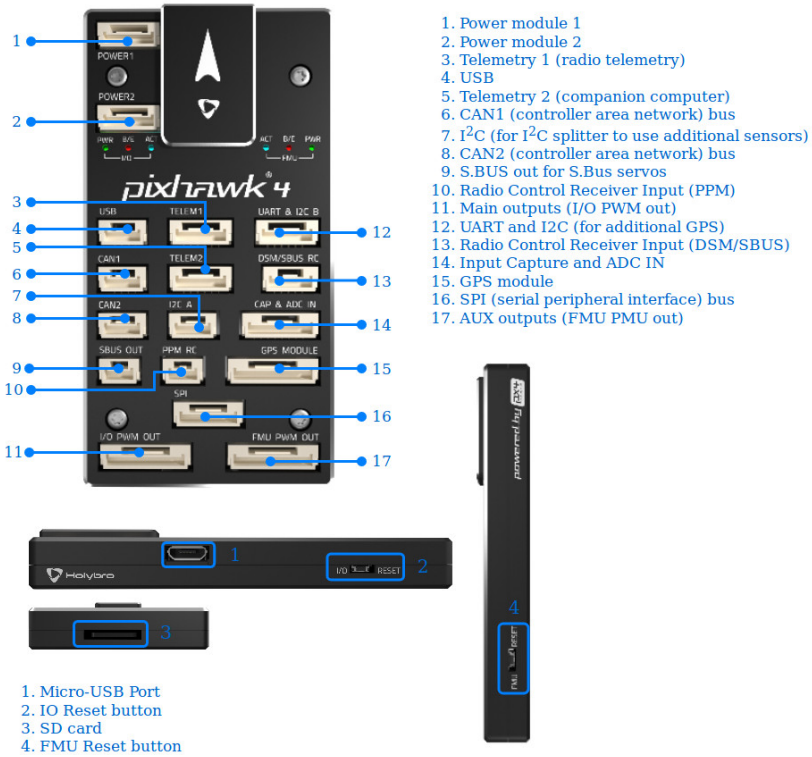


Figure 13: Pixhawk 4 flight controller [66]

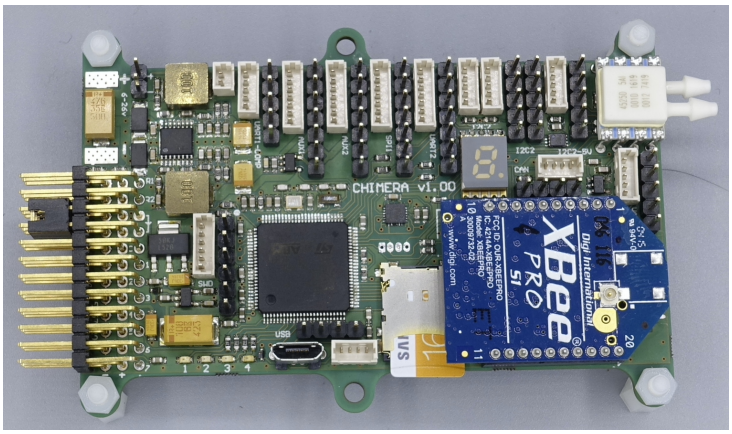


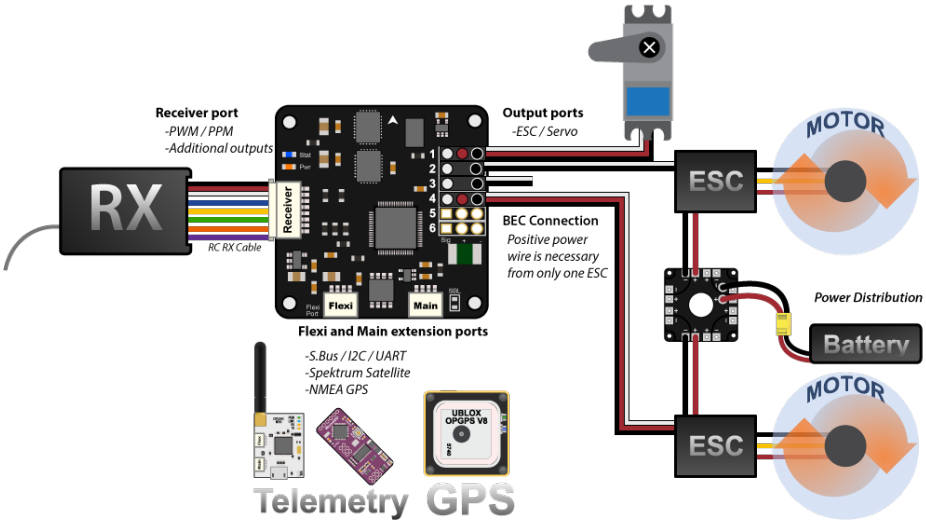
Figure 14: Paparazzi Chimera flight controller [67]

2.2.4.3 Commercial solutions

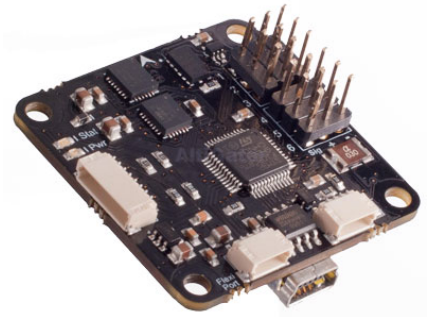
The commercial term here refers to the proprietary nature of the HW, opaque to the end-user. The lack of transparency can be an issue, raising suspicions, like the on-going ban of the U.S. Army to the Dji drones [58, 70]. Nonetheless, proprietary solutions represent 70% of the market share, with Dji being the biggest manufacturer [44].

The commercial solutions can be classified into the following categories:

1. **ARM-based platorms**: The `SPRacing H7 Extreme` is a low-end flight controller running



(a) Connection diagram



(b) flight controller board

Figure 15: CC3D flight controller [68]

Figure 16: CUAV v5 Plus flight controller [69]

`ArduPilot` SW stack (see Fig. 17). It features an STM32F750 ARM 32-bit MCU @ 400 MHz, and comes with on-board sensors (dual IMU, barometer). It supports I2C, UART, SPI, micro USB, and camera interfaces [71].

2. **Field-Programmable Gate Array (FPGA)-based solutions**: The `Aerotenna OcPoC-Zynq Mini` is a FPGA + ARM System on Chip (SoC) based flight control platform (Xilinx Zynq Z-7010),

which supports ArduPilot and PX4 SW stacks [72] (see Fig. 18). The FPGA /glsio's flexibility allows for rapid sensor integration and customization of the flight controller HW, allowing for capabilities such as triple redundancy in GPS, magnetometers, and IMUs. The FPGA is an Artix-7 with 28k logic cells, and the ARM SoC is ARM A9 dual-core @ 667 MHz, including 512 MB RAM and 128 MB of flash memory. It requires an Storage Disk (SD) card of 16 GB for Linux booting and Data logging. Includes also on-board sensors (dual IMU, barometer), 16 programmable tri-pin I/O and 10 programmable I/Os supporting the I2C, USB, SPI, CAN, Camera Serial Interface (CSI), and Generic Serial Interface (GSI) interfaces.

3. **Companion computer-based platforms**: The Navio2 is a RaspberryPi-based autopilot, which supports ArduPilot and PX4 SW stacks [73] (see Fig. 19). Basically, it is an UAV extension board (shield) for the RaspberryPi 2 or later, a quad-core 1 GHz computer running a real-time Linux and ArduPilot flight stack. The shield includes on-board sensors (dual IMU, barometer), a GNSS receiver, an RC I/O coprocessor, extension ports exposing Analog to Digital Converter (ADC), I2C, and UART interfaces for sensors and radios, 14 PWM servomotors outputs, and a triple redundant power supply.

   The PixC4-Jetson (Fig. 20) is a high-end FMU, powerful single board computer and peripheral support system (USB, MIPI, Ethernet, M.2 slot, etc.) in a small form factor, designed to be integrated into end-user platforms [74]. It supports the ArduPilot and PX4 SW stacks. The term "PixC4" is derived from the Pixhawk, on which the FMU design is based (FMUv5) and C4, representing Command, Control, Compute and Communication. It is available as a turnkey solution including software pre-flashed on the Nvidia Jetson companion [74], enabling the following features: User Datagram Protocol (UDP) telemetry (MAVLink); Long-Term Evolution (LTE) connection management with Layer-2 peer to peer Virtual Private Network (VPN); multiple-endpoint video encoding pipelines, web interface for configuration and remote terminal access; scalable and secure cloud connectivity to Horizon31's U.S. servers and optional access to their cloud GCS and low-latency webReal-Time Communication (RTC) video distribution system. It feaures the STM32H743 processor with STM IO coprocessor, integrated Nvidia Jetson companion computer, ethernet switch (3 port), USB 2.0 Hub (7 port), cellular/LTE Modem support, and on-board sensors (IMU, barometer, and compass). It supports the PWM, I2C, UART, SPI, CAN, and CSI. It is important to note that the FMU derives from the Pixhawk FMUv5, but can be sold commercially as a proprietary solution because it is released under the BSD license, which does not obligate to make the modifications to the design open to the public. Obviously, this is very attractive for companies who want to commercialize drones, but do not want to invest the time and money to developed an end-to-end solution.

Comparison between the open-source and commercial solutions

Table with multiple dimensions

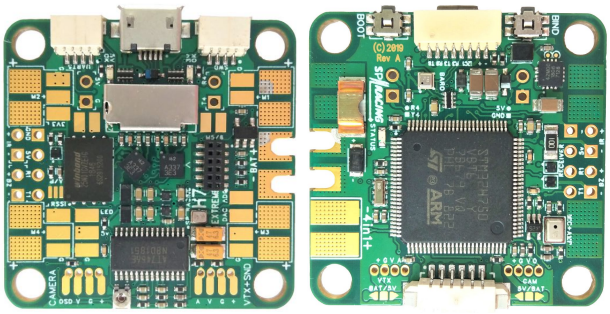


Figure 17: SPRacing H7 extreme flight controller [71]

Figure 18: Aerotenna OcPoc-Zynq Mini flight controller [72]

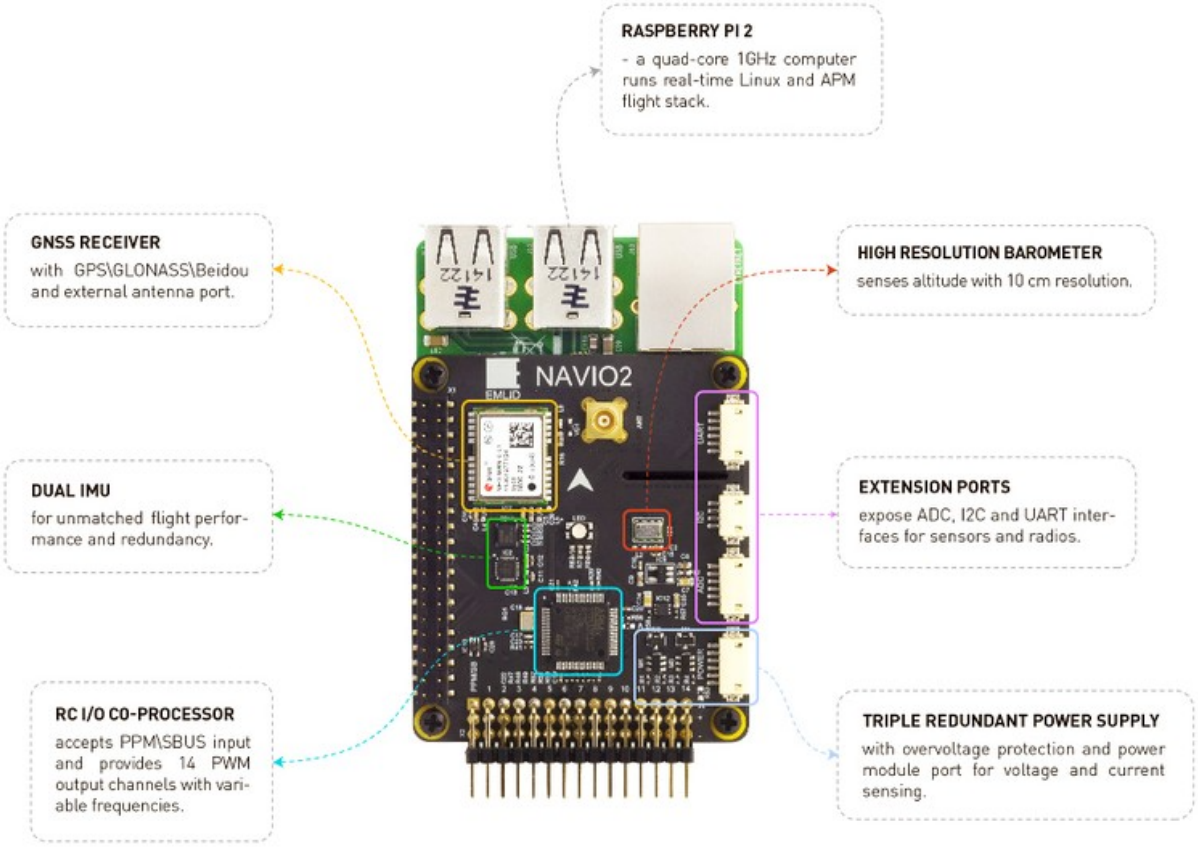
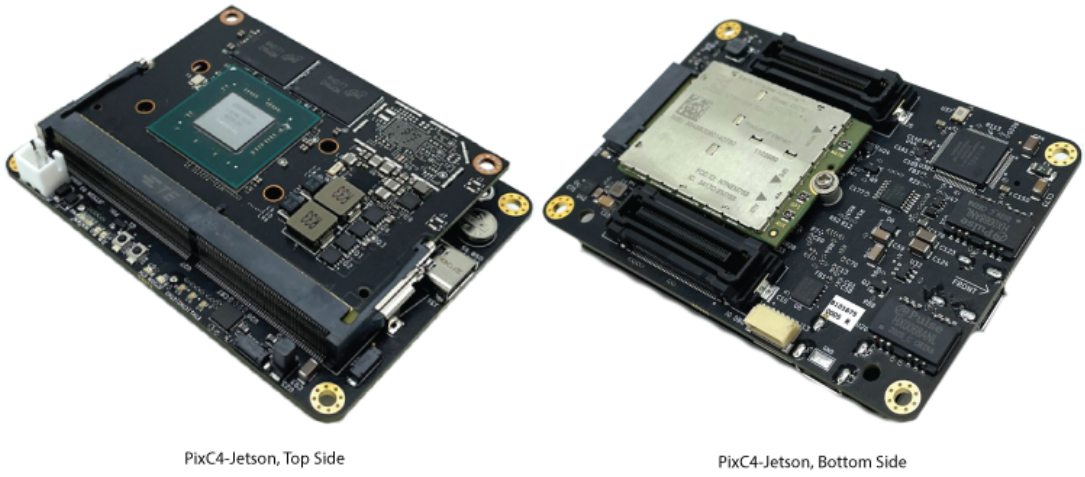
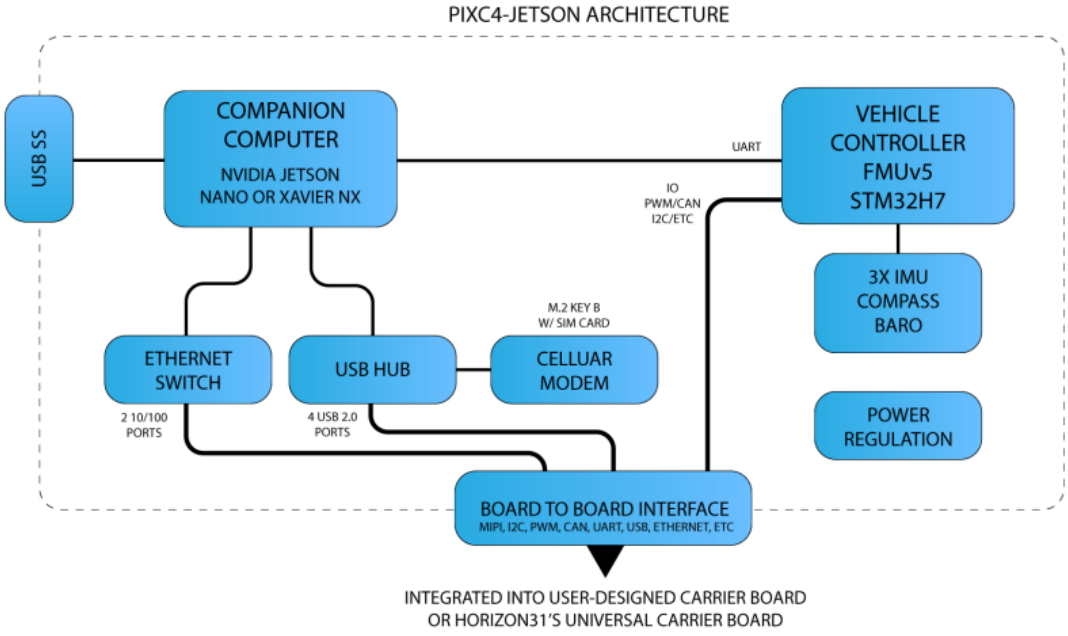


Figure 19: NAVIO2 flight controller [73]



(a) Board



(b) HW architecture

Figure 20: Horizon31 PixC4-Jetson flight controller [74]

2.2.5 UAV Reference Software

In this section the reference software for UAVs is discussed. An overview and the SW architecture is presented. Then, the open-source and commercial solutions are discussed, and compared in the gap-analysis.

2.2.5.1 Overview and Architecture

Fig. 21 illustrates the high-level abstraction of the most common structure of UAV SW architecture [55, 64]. In orange, we have the main computing platforms — the flight controller (FCS), e.g., Pixhawk, the

GCS (e.g., a desktop/laptop computer), and, optionally, the companion computer, e.g., a RaspberryPi — in purple the physical HW — sensors, actuators, and payload — in green the layers of the SW stack, and the arrows indicate the communication link and associated protocols.

The SW stack structure is the same for the flight controller and the companion computer:

- **Application Layer**: this layer is where the applications/tasks reside, providing the high-level functionalities of the system. In the flight controller we have the critical tasks, namely, *Controller*, *Commander*, *Navigator*, *Logger*, or *Communications*, and in the companion computer the secondary tasks, providing mission and navigation extra features, such as, *Collision Avoidance*, *Collision Prevention*, *Safe Landing*, *Odometry*, and *Telemetry*.

- **Middleware Layer**: the middleware is an intermediate layer, responsible for abstracting the interface with the OS and the communications through message passing — using real-time and or asynchronous protocols, such as Real-Time Publish Subscribe (RTPS), MAVLink, or `uORB` — and for providing an extra layer of security enforcing Context Security Policy (CSP) mechanisms.

- **OS Layer**: the OS provides the basic services for system's operation and a straightforward environment for application's development. The flight controller hosts a Real-Time Operating System (RTOS) to meet the stringent requirements and deadlines of the UAV's control, such as, `NuttX`, `FreeRTOS`, `VxWorks`, and `ChibiOS`. On the other hand, the companion computer hosts a General-Purpose Operating System (GPOS), as the applications on top have soft real-time requirements. Furthermore, it provides a much better bootstrapping environment for "general" software development than a RTOS, e.g., ROS-based avoidance libraries are available for Linux [64].

- **Device Driver Layer (Hardware Abstraction Layer (HAL))**: this layer abstracts the underlying HW, providing a tractable interface for the OS for data acquisition and motors' actuation.

The GCS SW (e.g., *QGroundControl*) typically runs on a desktop/laptop computer or mobile device providing real-time monitoring of the flight superimposed on a map and UAV's control functionalities via companion computer communication. The SW interface between both systems is achieved through the use of off-board APIs or SDKs, like ROS or MAVSDK [64].

2.2.5.2 Open-source solutions

Open-Source Software (OSS) solutions, like their HW counterparts, have been developed to provide more transparency, extensibility, flexibility, maintenability, while trying to be cost-effective. Open-Source Software (OSS) means that the SW's source code is released under free/libre terms [65].

The three most prominent OSS solutions at the moment are `PX4`, `ArduPilot`, and `Paparazzi UAS`.

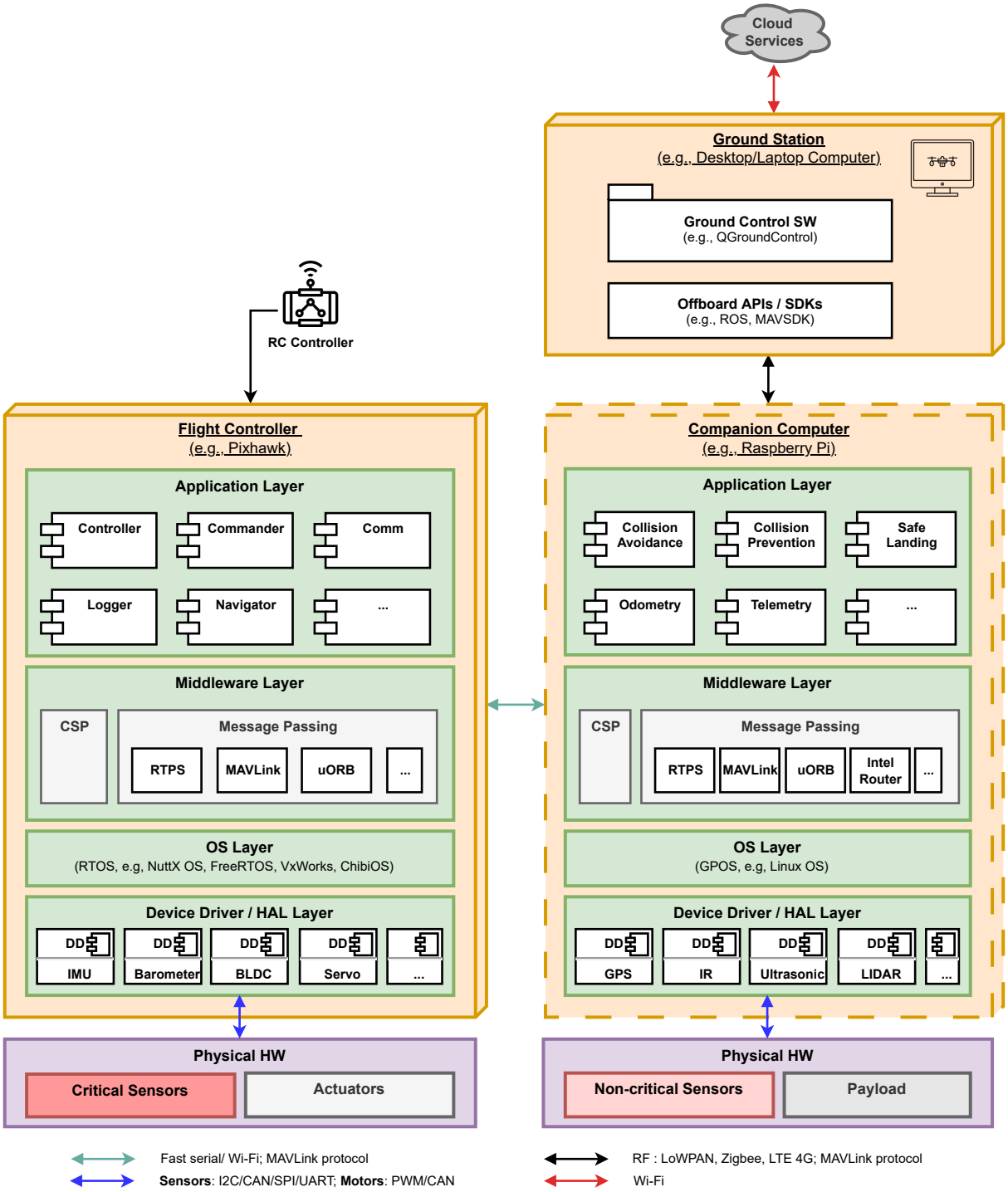


Figure 21: UAV SW architecture: most common structure

PX4 PX4 is an autopilot flight stack for drones, started in 2012, released under the permissible BSD-2 license, supporting different UAV's airframes and even others UVs. The source code core is on C/C++ and it is available on Github [75]. Several commercially available drones have adopted PX4, and a wider range supports it [76]. It supports NuttX and ROS operating systems [77].

PX4 is loaded onto the flight controller HW using the *QGroundControl* application. *QGroundControl* is

also used to configure PX4 and interface the flight controller. A RC unit can be used to manually control the drone.

PX4 uses the MAVLink protocol to communicate with the flight controller, but also supports the RTPS protocol. It uses its message API — `uORB` — for data transfer between its internal modules. It is automatically started on bootup, and messages are defines as separate `.msg` files in the `msg/` folder [77]. PX4 uses several sensors for UAV's control or autonomous operation. The minimum set of sensors are the accelerometers/gyroscopes, magnetometer and barometer, with a GPS being required for automatic modes. PX4 produces logs durint flights, which can be analyzed using *Flight Review*, or *Px4Tools* [3].

ArduPilot ArduPilot is an autopilot flight stack for drones, started in 2009, released under the more restrictable GPLv3 license, supporting different UAV's airframes and even others UVs. The source code core is on C/C++ and it is available on Github [78]. It supports *Linux* and *ChibiOS* operating systems [77].

It is installed on more than a million vehicles and runs on a large number of OSH platforms such as *Pixhawk*. *ArduPilot* features include: multiple flight modes (manual, semi- and full-autonomous) with multiple stabilization options; progammable missions with 3D waypoints and optional geofencing; support for multiple sensors and buses; fail-safe actions and support for navigation in GPS denied environments [79].

ArduPilot supports the MAVLink protocol for communication with GCSs and companion computers [79]. Mission commands are stored in Electrically Erasable Programmable Read-Only Memory (EEPROM) and executed one-by-one when the vehicle is switched into *Auto* mode.

Paparazzi UAS *Paparazzi UAS* is an autopilot flight stack for drones, started in 2003, released under the GPLv2 license, supporting different UAV's airframes UVs [80]. The source code core is on C (and a little C++) and it is available on Github [81]. It supports the *ChibiOS* operating system [77].

It was designed mainly for autopilot applications, for portability, and flexibility, with the ability to control multiple aircraft systems within the same system [80].

Paparazzi UAS uses its own *PPRZLINK* protocol for communication with GCSs and companion computers [77]. It uses a proprietary middleware Application Binary Interface (ABI) — *AirBorne Ivy* — to transfer data between modules in a straightforward way. Messages are defined in a eXtensible Markup Language (XML) file with an unique name and *id*. *Auto* mode.

Jargalsaikhan et al. [77] conducted a survey on the flight control software portability, more specifically at a module-level basis, analyzing several academia and OSS solutions features — general structure, messaging mechanisms, and supported OSs — and comparing them with the portability requirements — simplicity, moduarity, centralized messages, standard interfaces, HAL's implementation, and documentation. It found out that, from the aforementioned OSS solutions: only *ArduPilot* and *PX4* implement a HAL; *ArduPilot* does not implement a message centralization mechanism; although open-source, all use proprietary messaging interfaces, which hinders portability; and that although well documented, sometimes is hard to understand and/or is partially outdated. Fig. 22 illustrates the SW stack layers and modules

for the abovementioned OSS solutions and the *mpFCS* — the portable FCS developed by Jargalsaikhan et al. [77].

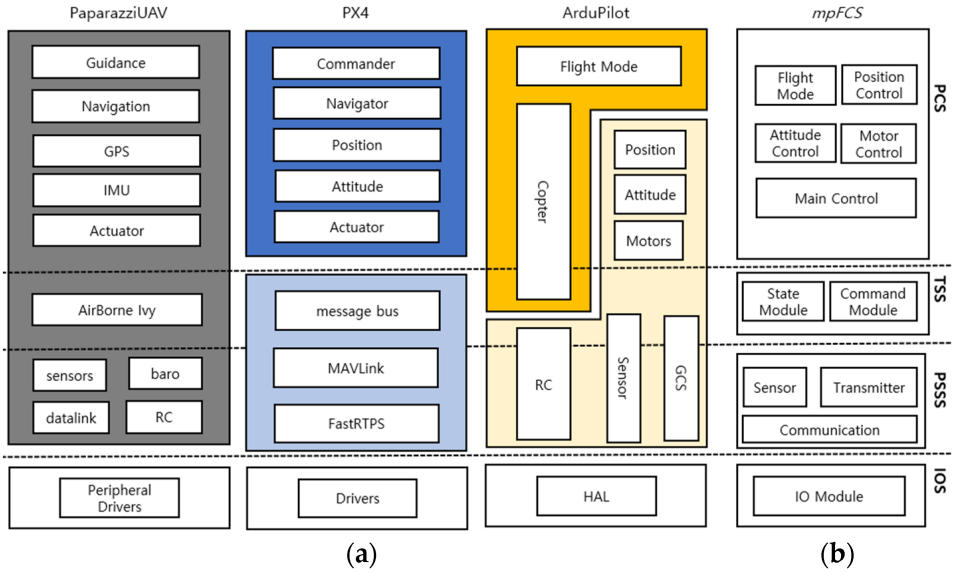


Figure 22: Analysis of the OSS modules and the mpFCS: (a) OSS (Paparazzi UAS, PX4, ArduPilot); (b) mpFCS [77][4]

2.2.5.3 Commercial solutions

Typically, the commercial proprietary solutions do not disclose information about the product. They provide extensibility through the use of SDKs, like the Dji drones [45], enabling the end-user to use extra features, but requires specialized knowledge.

On the other hand, and as aforementioned, some commercial solutions can explore more permissible OSS licenses, like the BSD license provided by PX4, to add extra functionalities and bundle it as a closed, commercial, product.

If, however, we expand the search to include any proprietary software component, such as the `VxWorks` RTOS then the results' list becomes significantly larger, namely, the Northrot Grumman X-47B UAV [82], the Airbus Helionix [83], and the Airbus Atlante [84].

On academia, *VxWorks* was also used as RTOS for UAV applications. Chong and Li [85] designed a FCS for small UAVs using *VxWorks* as an RTOS to support tasks partitioning meeting the real-time requirements. Fig. 23 illustrates the software framework designed considering three main layers: functional layer (application + middleware), OS layer, and device driver layer. The authors claimed remarkable improvements in the reliability and real-time capability of the FCS using `VxWorks`, due to its multitasking capabilities, but did not provide direct comparison with other RTOSs.

Comparison between the open-source and commercial solutions

Table with multiple dimensions

[4]Used with permission from MDPI: licence nr. 5457890117132
[5]Used with permission from IEEE: licence nr. 5457890117132

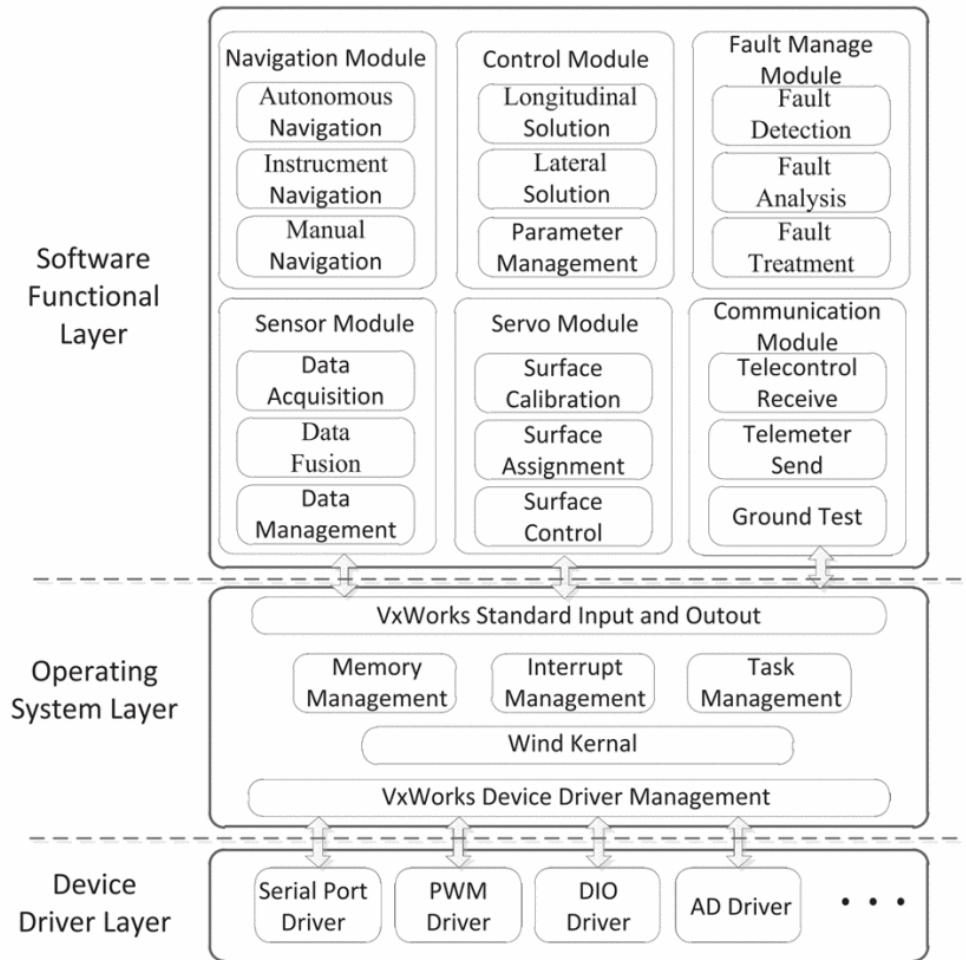


Figure 23: SW framework for an UAV application based on the VxWorks RTOS [85][5]

2.3 Related work

Papers de UAVs/drones no contexto de solucoes, arquiteturas, security, etc...

UAVs's market is booming, but the security and safety is often overlooked in the design [63]. Several vulnerabilities have been identified [4, 5, 56], with a extensive attack surface area — FCS, GCS, chassis, FPV channel, and the cloud services.

The commercial SW solutions do not provide the security mechanisms required, and, furthermore lack transparency. Buquerin [86] conducted a security evaluation on the VxWorks 7 RTOS — a commercial professional-level RTOS — for avionic systems. The author reported that basic attacks such as buffer overflows or string vulnerabilities led to a noticeable performance decrease in the system. Moreover, no security mechanism protects the system from malware or command injection vulnerabilities. This poses a major threat on the system if no supervising technology, like virtualization, is used. On the other hand, good security practices are used in areas such as cryptography and privilege management.

With the open-source solutions, comes transparency, but, nonetheless, it is insufficient to meet the mixed-criticality and security requirements. Zhang et al. [87] compared the two main RTOSs for drone's

applications in the open-source domain, Nuttx and ChibiOS, with the latter emerging as the best. Not only ChibiOS outperforms NuttX in most temporal tests, but also, and even more importantly, succeeds in the priority inversion avoidance test with mutexes rather than binary semaphores. The context switch overhead is greater for the NuttX, augmenting the probability of missing deadlines. Furthermore, NuttX does not provide priority inheritance using mutexes, which may lead to faulty designs, and has a larger codebase. Since UAVs have stringent real-time constraints, ChibiOS appears to the be the best RTOS for FCSs, and therefore the migration of ArduPilot's from NuttX to ChibiOS is probably a smart decision.

This leads to some research being conducted on changing the paradigm entirely. Alladi et al. [2] propose a completely different approach — the use of a decentralized architecture to increase security in UAVs application through the application of blockchain — especially in cooperative environments. Blockchain features such as smart contracts and consensus mechanisms and the inherent cryptographic foundations, enabling automation and providing security, according to the authors. Overall, the blockchain technology helps in overcoming many problems such as coordination, security, collision avoidance, privacy, decision marking, and signal jamming. Although it presents great potential, the real-life implementation seems a little distant.

Virtualization emerges as a viable solution to handle the UAV's mixed-criticality requirements and security, through spatial, temporal, and fault isolation. Nonetheless, the number of works in these field is still scarce. Faultrel et al [88] proposed an hypervisor based approach for mixed critical real-time UAV applications to meet its stringent timing and criticality requirements and allow FCS certification. The authors used an iterative approach for the construction of scheduling tables for the VM, as they use dynamic partioning.

However, dynamic partioning increases the latency and undeterminism of the system, hampering the mixed-criticality requirements. Thus, a static partioning approach appears to be the best-compromise solution for MCSs in UAVs' applications.

Bibliography

[1] J. M. Lourenço. *The NOVAthesis LaTeX Template User's Manual*. NOVA University Lisbon. 2021. url: https://github.com/joaomlourenco/novathesis/raw/main/template.pdf (cit. on p. ii).

[2] T. Alladi et al. "Applications of blockchain in unmanned aerial vehicles: A review". In: *Vehicular Communications* 23 (2020), p. 100249. issn: 2214-2096. doi: https://doi.org/10.1016/j.vehcom.2020.100249. url: https://www.sciencedirect.com/science/article/pii/S2214209620300206 (cit. on pp. 1, 11, 12, 14, 37).

[3] J. Glossner, S. Murphy, and D. Iancu. "An overview of the drone open-source ecosystem". In: *arXiv preprint arXiv:2110.02260* (2021) (cit. on pp. 1, 11, 18, 34).

[4] S. A. H. Mohsan et al. "Towards the unmanned aerial vehicles (UAVs): A comprehensive review". In: *Drones* 6.6 (2022), p. 147 (cit. on pp. 1, 12–15, 19, 21, 36).

[5] B. Nassi et al. "SoK: Security and Privacy in the Age of Commercial Drones". In: *2021 IEEE Symposium on Security and Privacy (SP)*. 2021, pp. 1434–1451. doi: 10.1109/SP40001.2021.00005 (cit. on pp. 1, 12, 20, 21, 36).

[6] ISO. *Product development: software level, ISO 26262: Road Vehicles - Functional safety 6* (cit. on pp. 2, 4).

[7] R. (S. 167. *Software considerations in airborne systems and equipment certification*. RTCA, Incorporated, 1992 (cit. on pp. 2, 4).

[8] E. Cenelec. "50128-Railway applications-Communication, signalling and processing systems-Software for railway control and protection systems". In: *Book EN* 50128 (2012) (cit. on pp. 2, 4).

[9] A. Burns and R. I. Davis. "Mixed criticality systems-a review:(february 2022)". In: *Department of Computer Science, University of York, Tech. Rep* (2022) (cit. on pp. 2, 4, 5).

[10] M. Cinque et al. “Virtualizing mixed-criticality systems: A survey on industrial trends and issues”. In: *Future Generation Computer Systems* 129 (2022), pp. 315–330. issn: 0167-739X. doi: https://doi.org/10.1016/j.future.2021.12.002. url: https://www.sciencedirect.com/science/article/pii/S0167739X21004787 (cit. on pp. 5–9).

[11] Xilinx. *RunX*. https://github.com/Xilinx/runx. accessed: 2022-11-27. 2020 (cit. on p. 5).

[12] V. Struhár et al. “Real-time containers: A survey”. In: *2nd Workshop on Fog Computing and the IoT (Fog-IoT 2020)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik. 2020 (cit. on p. 5).

[13] S. A.G. *Jailhouse hypervisor source code*. https://github.com/siemens/jailhouse. accessed: 2022-11-30 (cit. on pp. 8, 9).

[14] J. Martins et al. “Bao: A Lightweight Static Partitioning Hypervisor for Modern Multi-Core Embedded Systems”. In: *Workshop on Next Generation Real-Time Embedded Systems (NG-RES 2020)*. Ed. by M. Bertogna and F. Terraneo. Vol. 77. OpenAccess Series in Informatics (OASIcs). Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2020, 3:1–3:14. isbn: 978-3-95977-136-8. doi: 10.4230/OASIcs.NG-RES.2020.3. url: https://drops.dagstuhl.de/opus/volltexte/2020/11779 (cit. on pp. 8–11).

[15] A. Kivity et al. “kvm: the Linux virtual machine monitor”. In: *Proceedings of the Linux symposium*. Vol. 1. 8. Dttawa, Dntorio, Canada. 2007, pp. 225–230 (cit. on pp. 8, 9).

[16] M. Corporation. *Hyper-V*. https://learn.microsoft.com/en-us/windows-server/virtualization/hyper-v/hyper-v-technology-overview. accessed: 2022-11-30. 2020 (cit. on p. 8).

[17] P. Barham et al. “Xen and the art of virtualization”. In: *ACM SIGOPS operating systems review* 37.5 (2003), pp. 164–177 (cit. on pp. 8–10).

[18] G. Heiser. “The role of virtualization in embedded systems”. In: *Proceedings of the 1st workshop on Isolation and integration in embedded systems*. 2008, pp. 11–16 (cit. on p. 9).

[19] SysGO. *PikeOS product overview*. https://www.sysgo.com/fileadmin/user_upload/www.sysgo.com/redaktion/downloads/pdf/data-sheets/SYSGO-Product-Overview-PikeOS.pdf. accessed: 2022-11-30 (cit. on p. 9).

[20] M. Masmano et al. “Xtratum: a hypervisor for safety critical embedded systems”. In: *11th Real-Time Linux Workshop*. Citeseer. 2009, pp. 263–272 (cit. on p. 9).

[21] S. Pinto et al. “Towards a TrustZone-assisted hypervisor for real-time embedded systems”. In: *IEEE computer architecture letters* 16.2 (2016), pp. 158–161 (cit. on p. 9).

[22] J. Martins et al. “µTZVisor: A Secure and Safe Real-Time Hypervisor”. In: *Electronics* 6.4 (2017). issn: 2079-9292. doi: 10.3390/electronics6040093. url: https://www.mdpi.com/2079-9292/6/4/93 (cit. on p. 9).

[23] P. Lucas et al. “VOSYSmonitor, a TrustZone-based Hypervisor for ISO 26262 Mixed-critical System”. In: *2018 23rd Conference of Open Innovations Association (FRUCT)*. 2018-11, pp. 231–238. doi: 10.23919/FRUCT.2018.8588018 (cit. on p. 9).

[24] P. Lucas et al. “Vosysmonitor, a low latency monitor layer for mixed-criticality systems on armv8-a”. In: *29th Euromicro Conference on Real-Time Systems (ECRTS 2017)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik. 2017 (cit. on p. 9).

[25] J. Martins et al. “{ClickOS} and the Art of Network Function Virtualization”. In: *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)*. 2014, pp. 459–473 (cit. on p. 9).

[26] S. Lankes, S. Pickartz, and J. Breitbart. “HermitCore: a unikernel for extreme scale computing”. In: *Proceedings of the 6th International Workshop on Runtime and Operating Systems for Supercomputers*. 2016, pp. 1–8 (cit. on p. 9).

[27] A. Bansal et al. “Evaluating the memory subsystem of a configurable heterogeneous mpsoc”. In: *Workshop on Operating Systems Platforms for Embedded Real-Time Applications (OSPERT)*. Vol. 7. 2018, p. 55 (cit. on p. 10).

[28] Q. Ge et al. “A survey of microarchitectural timing attacks and countermeasures on contemporary hardware”. In: *Journal of Cryptographic Engineering* 8.1 (2018), pp. 1–27. doi: 10.1007/s13389-016-0141-6. url: https://doi.org/10.1007/s13389-016-0141-6 (cit. on p. 10).

[29] J. Martins and S. Pinto. “Bao: a modern lightweight embedded hypervisor”. In: *Embedded World Conference*. Nuremberg, Germany. 2020 (cit. on p. 10).

[30] C. Dall. *The Design, Implementation, and Evaluation of Software and Architectural Support for ARM Virtualization*. Columbia University, 2018 (cit. on p. 11).

[31] B. Hypervisor. *Bao project repo*. https://github.com/bao-project/bao-hypervisor. accessed: 2022-11-27. 2020 (cit. on p. 11).

[32] B. G. Blundell. “Empowering Technology: Drones”. In: *Ethics in Computing, Science, and Engineering: A Student’s Guide to Doing Things Right*. Cham: Springer International Publishing, 2020, pp. 489–578. isbn: 978-3-030-27126-8. doi: 10.1007/978-3-030-27126-8_7. url: https://doi.org/10.1007/978-3-030-27126-8_7 (cit. on p. 11).

[33] ConsortIQ. *A not-so-short history of Unmanned Aerial Vehicles (UAV)*. https://consortiq.com/uas-resources/short-history-unmanned-aerial-vehicles-uavs. accessed: 2022-11-30. 2022 (cit. on p. 12).

[34] ArduPilot. *History of ArduPilot*. https://ardupilot.org/copter/docs/common-history-of-ardupilot.html. accessed: 2022-11-30. 2022 (cit. on p. 12).

[35] Auterion. *The story of PX4 and Pixhawk*. https://auterion.com/company/the-history-of-pixhawk/. accessed: 2022-11-30. 2022 (cit. on p. 12).

[36] Z. Ullah, F. Al-Turjman, and L. Mostarda. “Cognition in UAV-Aided 5G and Beyond Communications: A Survey”. In: *IEEE Transactions on Cognitive Communications and Networking* 6.3 (2020), pp. 872–891. doi: 10.1109/TCCN.2020.2968311 (cit. on p. 13).

[37] A. Fotouhi et al. “Survey on UAV Cellular Communications: Practical Aspects, Standardization Advancements, Regulation, and Security Challenges”. In: *IEEE Communications Surveys & Tutorials* 21.4 (2019-10), pp. 3417–3442. issn: 1553-877X. doi: 10.1109/COMST.2019.2906228 (cit. on p. 13).

[38] C. Stöcker et al. “Review of the Current State of UAV Regulations”. In: *Remote Sensing* 9.5 (2017). issn: 2072-4292. doi: 10.3390/rs9050459. url: https://www.mdpi.com/2072-4292/9/5/459 (cit. on pp. 13, 19).

[39] Flyability. *Gas powered drone: A guide*. https://www.flyability.com/gas-powered-drone. accessed: 2022-11-30. 2022 (cit. on p. 14).

[40] Parrot. *Parrot drones*. https://www.parrot.com/us/drones. accessed: 2022-11-30. 2022 (cit. on pp. 14, 15).

[41] Dji. *Dji Mavic 3*. https://www.dji.com/pt/mavic-3. accessed: 2022-11-30. 2022 (cit. on p. 14).

[42] Energyor. *Energyor H2Quad 1000*. http://energyor.com/products/detail/h2quad-1000. accessed: 2022-11-30. 2022 (cit. on pp. 14, 15).

[43] Flaperon. *Flaperon MX8*. https://flaperon.com/. accessed: 2022-11-30. 2022 (cit. on p. 14).

[44] D. Analyst. *The rise of open-source drones*. https://droneanalyst.com/2021/05/30/rise-of-open-source-drones. accessed: 2022-11-30. 2021 (cit. on pp. 14, 27).

[45] Dji. *Dji Software Development Kit*. https://developer.dji.com/. accessed: 2022-11-30. 2022 (cit. on pp. 14, 35).

[46] T. Verge. *Yuneec announces Typhoon H Plus alongise first fixed-wing and racing drones*. https://www.theverge.com/2018/1/9/16867090/yuneec-typhoon-h-plus-firebird-fpv-hd-racer-drones-ces-2018. accessed: 2022-11-30. 2022 (cit. on p. 15).

[47] V. Rotors. *Velos UAV*. https://www.velosuav.com/velosv3/. accessed: 2022-11-30. 2022 (cit. on p. 15).

[48] DeltaQuad. *DeltaQuad Pro VTOL UAV*. https://www.deltaquad.com/. accessed: 2022-11-30. 2022 (cit. on p. 15).

[49] A. A. Company. *Hybrid Advanced Multi-Rotor (HAMR)*. https://advancedaircraftcompany.com/hamr/. accessed: 2022-11-30. 2022 (cit. on p. 15).

[50] XSun. *Solar X One*. https://xsun.fr/autonomous-drone/. accessed: 2022-11-30. 2022 (cit. on p. 15).

[51] S. Aggarwal and N. Kumar. "Path planning techniques for unmanned aerial vehicles: A review, solutions, and challenges". In: *Computer Communications* 149 (2020), pp. 270–299. issn: 0140-3664. doi: https://doi.org/10.1016/j.comcom.2019.10.014. url: https://www.sciencedirect.com/science/article/pii/S0140366419308539 (cit. on pp. 16, 17).

[52] T. Vogeltanz. "A Survey of Free Software for the Design, Analysis, Modelling, and Simulation of an Unmanned Aerial Vehicle". In: *Archives of Computational Methods in Engineering* 23.3 (2016), pp. 449–514. doi: 10.1007/s11831-015-9147-y. url: https://doi.org/10.1007/s11831-015-9147-y (cit. on p. 17).

[53] E. Ebeid, M. Skriver, and J. Jin. "A Survey on Open-Source Flight Control Platforms of Unmanned Aerial Vehicle". In: *2017 Euromicro Conference on Digital System Design (DSD)*. 2017, pp. 396–402. doi: 10.1109/DSD.2017.30 (cit. on pp. 17, 26).

[54] D. L. Gabriel, J. Meyer, and F. du Plessis. "Brushless DC motor characterisation and selection for a fixed wing UAV". In: *IEEE Africon '11*. 2011, pp. 1–6. doi: 10.1109/AFRCON.2011.6072087 (cit. on p. 17).

[55] M. Leccadito et al. "A survey on securing UAS cyber physical systems". In: *IEEE Aerospace and Electronic Systems Magazine* 33.10 (2018), pp. 22–32 (cit. on pp. 19, 23, 31).

[56] C. G. L. Krishna and R. R. Murphy. "A review on cybersecurity vulnerabilities for unmanned aerial vehicles". In: *2017 IEEE International Symposium on Safety, Security and Rescue Robotics (SSRR)*. 2017, pp. 194–199. doi: 10.1109/SSRR.2017.8088163 (cit. on pp. 19, 36).

[57] K. Mansfield et al. "Unmanned aerial vehicle smart device ground control station cyber security threat model". In: *2013 IEEE International Conference on Technologies for Homeland Security (HST)*. 2013, pp. 722–728. doi: 10.1109/THS.2013.6699093 (cit. on p. 19).

[58] sUAS News. *US Army calls for units to discontinue use of DJI equipment*. https://www.suasnews.com/2017/08/us-army-calls-units-discontinue-use-dji-equipment/. accessed: 2022-12-05. 2017 (cit. on pp. 19, 27).

[59] T. Independent. *Man arrested for landing 'radioactive' drone on japanese prime minister's roof*. https://www.independent.co.uk/news/world/asia/man-arrested-for-landing-radioactive-drone-on-japanese-prime-ministers-roof-10203517.html. accessed: 2022-12-05. 2015 (cit. on p. 19).

[60] N. Times. *Venezuelan President Target By Drone Attack, Officials say*. https://www.nytimes.com/2018/08/04/world/americas/venezuelan-president-targeted-in-attack-attempt-minister-says.html. accessed: 2022-12-05. 2018 (cit. on p. 19).

[61] T. Drive. *Russia Offers New Details About Syrian Mass Drone Attack, Now Implies Ukrainian Connection*. https://www.thedrive.com/the-war-zone/17595/russia-offers-new-details-about-syrian-mass-drone-attack-now-implies-ukrainian-connection. accessed: 2022-12-05. 2019 (cit. on p. 19).

[62] D. Sathyamoorthy. "A review of security threats of unmanned aerial vehicles and mitigation steps". In: *J. Def. Secur* 6.1 (2015), pp. 81–97 (cit. on p. 21).

[63] I. G. Ferrão et al. "STUART: ReSilient archiTecture to dynamically manage Unmanned aeriAl vehicle networks undeR atTack". In: *2020 IEEE Symposium on Computers and Communications (ISCC)*. IEEE. 2020, pp. 1–6 (cit. on pp. 22, 36).

[64] P. Autopilot. *PX4 System Architecture*. https://docs.px4.io/main/en/concept/px4_systems_architecture.html. accessed: 2022-12-08. 2022 (cit. on pp. 23, 31, 32).

[65] F. S. Foundation. *GNU Operating System*. https://gnu.org/philosophy/free-hardware-designs.en.html. accessed: 2022-12-05. 2022 (cit. on pp. 26, 32).

[66] P. Autopilot. *Pixhawk 4*. https://docs.px4.io/main/en/flight_controller/pixhawk4.html. accessed: 2022-12-07. 2022 (cit. on pp. 26, 27).

[67] Paparrazi. *Paparazzi Chimera v1.00*. https://wiki.paparazziuav.org/wiki/Chimera/v1.00. accessed: 2022-12-07. 2022 (cit. on pp. 26, 27).

[68] O. Wiki. *Copter Control / CC3D / Atom Hardware Setup*. https://opwiki.readthedocs.io/en/latest/user_manual/cc3d/cc3d.html. accessed: 2022-12-07. 2022 (cit. on pp. 26, 28).

[69] A. Wiki. *CUAV v5 Plus Overview*. https://ardupilot.org/copter/docs/common-cuav-v5plus-overview.html. accessed: 2022-12-07. 2022 (cit. on pp. 26, 28).

[70] DroneDJ. *After product ban, the US DoD formally blacklists drone giant DJI (Update)*. https://dronedj.com/2022/10/07/dji-dod-drone/. accessed: 2022-12-08. 2022 (cit. on p. 27).

[71] A. Wiki. *SPRacing H7 extreme*. https://ardupilot.org/copter/docs/common-spracingh7-extreme.html. accessed: 2022-12-08. 2022 (cit. on pp. 28, 30).

[72] Dronecode. *Aerotenna OcPoC-Zynq Mini Flight Controller*. https://docs.px4.io/v1.9.0/en/flight_controller/ocpoc_zynq.html. accessed: 2022-12-08. 2022 (cit. on pp. 29, 30).

[73] A. Wiki. *Navio2 Overview*. https://ardupilot.org/copter/docs/common-navio2-overview.html. accessed: 2022-12-07. 2022 (cit. on pp. 29, 30).

[74] A. Wiki. *Horizon31 PixC4-Jetson*. https://ardupilot.org/copter/docs/common-horizon31-pixc4-jetson.html. accessed: 2022-12-08. 2022 (cit. on pp. 29, 31).

[75] P. Github. *PX4-Autopilot*. https://github.com/PX4/PX4-Autopilot. accessed: 2022-12-08. 2022 (cit. on p. 33).

[76] P. Autopilot. *Opens Source Autopilot for Drone Developers*. https://px4.io/. accessed: 2022-12-08. 2022 (cit. on p. 33).

[77] T. Jargalsaikhan et al. "Architectural Process for Flight Control Software of Unmanned Aerial Vehicle with Module-Level Portability". In: *Aerospace* 9.2 (2022), p. 62 (cit. on pp. 33–35).

[78] A. Github. *ArduPilot Autopilot*. https://github.com/ArduPilot/ardupilot. accessed: 2022-12-08. 2022 (cit. on p. 34).

[79] ArduPilot. *ArduPilot, Home*. https://ardupilot.org/. accessed: 2022-12-08. 2022 (cit. on p. 34).

[80] Paparazzi. *Paparazzi Home*. https://wiki.paparazziuav.org/wiki/Main_Page. accessed: 2022-12-08. 2022 (cit. on p. 34).

[81] P. U. Github. *Paparazzi UAS Autopilot*. https://github.com/paparazzi/paparazzi. accessed: 2022-12-08. 2022 (cit. on p. 34).

[82] W. Blog. *Northrop Grumman X-47B UCAS-D Running on Wind River VxWorks Catapults from Aircraft Carrier*. https://blogs.windriver.com/wind_river_blog/2013/05/historic-milestone-for-northrops-x-47b-and-wind-river/. accessed: 2022-12-09. 2013 (cit. on p. 35).

[83] A. Internation. *Wind River VxWorks 653 Providing Power for Airbus Helionix*. https://www.aviationtoday.com/2016/05/11/wind-river-vxworks-653-providing-power-for-airbus-helionix/. accessed: 2022-12-09. 2016 (cit. on p. 35).

[84] sUAS News. *Wind River Technology Powers Airbus Group's Innovative Unmanned Aerial Vehicle ATLANTE*. https://www.suasnews.com/2014/09/wind-river-technology-powers-airbus-groups-innovative-unmanned-aerial-vehicle-atlante/. accessed: 2022-12-09. 2014 (cit. on p. 35).

[85] M. Chong and L. Chuntao. "Design of flight control software for small unmanned aerial vehicle based on VxWorks". In: *Proceedings of 2014 IEEE Chinese Guidance, Navigation and Control Conference*. 2014, pp. 1831–1834. doi: 10.1109/CGNCC.2014.7007459 (cit. on pp. 35, 36).

[86] K. K. G. Buquerin. "Security Evaluation for the Real-time Operating System VxWorks 7 for Avionic Systems". PhD thesis. Technische Hochschule Ingolstadt, 2018 (cit. on p. 36).

[87] M. Zhang et al. "Which Is the Best Real-Time Operating System for Drones? Evaluation of the Real-Time Characteristics of NuttX and ChibiOS". In: *2021 International Conference on Unmanned Aircraft Systems (ICUAS)*. IEEE. 2021, pp. 582–590 (cit. on p. 36).

[88] T. Fautrel et al. “An hypervisor approach for mixed critical real-time UAV applications”. In: *2019 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*. IEEE. 2019, pp. 985–991 (cit. on p. 37).