An Autonomous UAV Architecture for Remote Sensing and Intelligent Decision Making

Juan Boubeta-Puig*[1], Enrique Moguel[2], Fernando Sánchez-Figueroa[2], Juan Hernández[2], and Juan Carlos Preciado[2]

[1] Department of Computer Science and Engineering, University of Cádiz, Spain

[2] Quercus Software Engineering Group, University of Extremadura, Spain

---

Abstract:

Recently, the US Department of Transportation's Federal Aviation Administration and other international organizations have proposed a set of requirements for small unmanned aerial vehicles (UAVs) to operate for nonrecreational purposes. However, existing UAV architectures fulfill only some of the established requirements, and not all in one solution. This article presents an event-driven service-oriented architecture that allows autonomous UAVs to satisfy all these requirements and to detect critical situations, performing real-time decision making.

Keywords:

Computer architecture, Decision making, FAA, Unmanned aerial vehicles, Real-time systems, Autonomous vehicles

---

* Corresponding author: juan.boubeta@uca.es

Publisher version:

Boubeta-Puig, J., Moguel, E., Sánchez-Figueroa, F., Hernández, J., & Preciado, J. C. (2018). An Autonomous UAV Architecture for Remote Sensing and Intelligent Decision-making. *IEEE Internet Computing*, *22*(3), 6–15. https://doi.org/10.1109/MIC.2018.032501511

1. Introduction

Unmanned aerial vehicles (UAVs) –also called drones– are currently playing a prominent role. While regulations may still hinder their broader adoption, some technological advances are fueling their use:

- electronic improvements for the miniaturization of multiple sensory I/O devices;
- the appearance of multiple wireless-communication mechanisms;
- the possibility of transporting significant computational resources for processing and storing edge data, and
- the consolidation of cloud computing for providing software as a service as well as storing and sharing data.

Nowadays, UAVs are finding not only military uses but also civilian ones: environmental control, cattle raising and farming, and rescue in catastrophes, just to name a few.

While UAV functionality is increasing, there are other features such as adaptability, autonomy, efficiency, reliability, safety, and usability that must be improved.

The real-time decision-making process in autonomous UAVs is at the heart of many of the aforementioned features, and it is one of the main challenges to be addressed in most of the emergent application fields. Indeed, the lack of appropriate real-time decision-making strategies is the cause of many accidents involving UAVs.[1] Nevertheless, the fact is that the basic architecture of UAVs presents several limitations (e.g., limited onboard computing) to deal with these kinds of decisions.[2]

The main goal of this article is to provide an unprecedented UAV architecture extension for supporting onboard real-time decision making in autonomous UAVs. The core of this architecture is based on the use of complex event processing (CEP) onboard. The results obtained involve advances in terms of the number of events processed per second, response time, ease of use for nontechnological users, and code reconfiguration before or during the UAV flight. These results have been validated by implementing the architecture.

2. UAV Architecture Design for Autonomy: Requirements and Related Work

Real-time decision making in autonomous UAVs usually leads to an alteration of flight plans. These alterations are based on events that can be broadly grouped into three main categories, depending on the concern they are related to:[3]

- *Data acquisition or processing*. Onboard data acquisition and processing analysis may lead to changes in the navigation plan. For example, consider an eolic park (wind farm) that is formed by one or more electric substations. Each substation is formed by a set of wind turbines.

Damage to one of these turbines may affect a planned route and may require either acquiring new data on other substation or carefully reviewing the damaged turbine.[4]

- *Health management*. UAV health management addresses monitoring and sensing strategies for enhancing the UAV's useful life and mitigating potential catastrophic events.[5] For instance, the onboard prediction of the remaining useful life of UAV components (e.g., the battery) or data obtained from onboard sensors (e.g., wind speed or rain) may lead to changes in the navigation plan in order to take the appropriate corrective actions (e.g., return-to-home operation).
- *Regulations*. Reliability and safety are critical concerns in UAVs. The safe integration of a UAV into a common airspace requires UAV autonomous behavior to comply with current legal regulations. For example, operating higher than 400 feet is illegal for a drone.[5]

There might also be simultaneous events related to the previous categories that conflict each other. For example, changes in the navigation plan may conflict with battery life, or the pilot's intent or commands may conflict with legal issues, and so on. For those cases, a prioritization of the events belonging to the aforementioned categories must be established.

2.1 Requirements

Since the number of incidents or accidents involving UAVs has dramatically increased in past years,[1] the US Department of Transportation's Federal Aviation Administration (FAA) has proposed a set of 33 requirements for small UAVs to operate for nonhobby or nonrecreational purposes. These are grouped into four main categories: operational limitations, operator certification and responsibilities, aircraft requirements, and model aircraft.

Operational limitations are the set of requirements needed for real-time decision making in autonomous UAVs, since these limitations cover the UAV event categories previously mentioned: data acquisition or processing, health management, and legal issues. Considering these requirements together with the ones proposed by Clarke[3] and by Evertsz and colleagues,[6] the complete list of requirements for onboard real-time decision making can be established as follows:

1. data collection or acquisition from heterogeneous sources and different domains;
2. onboard processing, analyzing, and correlating of a continuously arriving huge amount of fine-grained data;
3. automated detection of relevant events;
4. dealing with multiple priorities among events that conflict with each other and event hierarchies in which some events depend on others;
5. real-time triggering of appropriate actions, notifying both users and the UAV itself, as a result of the decision-making process;
6. supporting flexible configuration at both design time and runtime; and
7. providing a friendly automated interface and conflict resolution between the UAV and the domain experts. This interface must facilitate the graphical modeling of event patterns

(conditions to be met and actions to be carried out) and automatically transform them into their implementation code.

2.2 Related Work

UAV architectures can be classified mainly into two types, depending on the way in which data can be processed: off-board and onboard processing.

Off-board processing requires data transmission between the UAV and the server, which is responsible for processing the incoming data and then sending the response back to the UAV.[7] All this process may be affected by communication overhead or even communication failure, involving a delay that, in many cases, is excessive and can jeopardize the integrity of the UAV. Thus, off-board processing architectures do not fulfill the second, third, and fifth established requirements.

Onboard processing facilitates data capture, processing, and the decision-making process almost immediately. Nevertheless, several onboard architectures are closed,[8–9] thus not allowing the modification and the inclusion of new sensors or actuators. This impedes modification of existing action rules or even the addition of new ones. So, these UAVs are proposed and implemented for a particular application domain, contrary to the first requirement. Moreover, decision-making rules are usually simple and without temporal restrictions, and they do not support event prioritization or an event hierarchy in which some events depend on others,[10] thus not satisfying the fourth requirement.

Additionally, there are onboard architectures similar to the one proposed in this article,[11] but with the difference that these make use of specific methods or algorithms for particular tasks without the capability of modifying the flight plan or allowing UAVs to self-manage themselves. Therefore, these architectures do not fulfill the sixth requirement. Remarkably, most of these methods are inherently computationally complex or inefficient in UAV real-time mode.[12]

Both off-board and onboard architectures require domain experts to have a somewhat advanced knowledge of programming languages and environments in order to program a UAV mission and to define the situations of interest and actions to be taken. For example, knowledge of Matlab Simulink may be needed.[13] So, the seventh requirement is not completely fulfilled.

Therefore, existing architectures for decision making in UAVs fulfill only some of the established requirements, and not all in one solution, unlike the architecture proposed in this article.

3. An Autonomous-UAV Architecture

An autonomous-UAV architecture for remote sensing and intelligent decision making is proposed in this section. This proposal is based on the integration of CEP technology with an event-driven service-oriented architecture (SOA 2.0).

CEP allows the processing, analyzing, and correlating of a huge amount of data in the form of events, with the aim of detecting critical situations in real time.[14] The use of this technology enables the architecture to satisfy the second to fourth requirements. By integrating CEP with SOA 2.0, this real-time detection can be done using data coming from heterogeneous sources and different domains, and the execution of appropriate actions is supported. So, the first and fifth requirements are also fulfilled. Additionally, combining CEP-based SOA 2.0 with a model-driven approach, which facilitates the definition and automatic code generation of event patterns, permits satisfaction of the seventh requirement. Altogether, these software components therefore fulfill the sixth requirement.

This autonomous UAV architecture incorporates both hardware and software components. Figure 1 illustrates these components, grouped into two tiers: the basic-architecture tier (A) and extended-architecture tier (B). The hardware components are orange; the software components are blue.

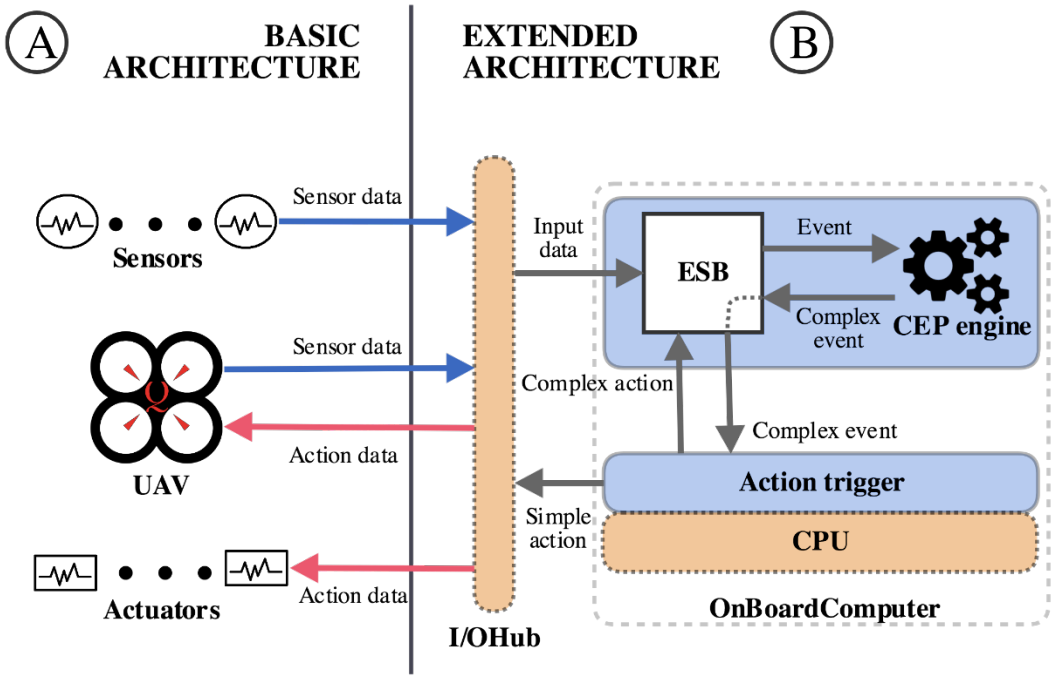

Figure 1. An autonomous-UAV (unmanned aerial vehicle) architecture for remote sensing and intelligent decision making. The hardware components are orange; the software components are blue. ESB = enterprise service bus; CEP = complex event processing.

The hardware components of the architecture are as follows.

The *I/OHub* gathers the data received from the UAV itself (e.g., GPS, altitude, velocity, and optical-detector data) and from the onboard sensors (e.g., humidity, temperature, air pollution, and noise data). The I/OHub sends these data ("Input data" in Figure 1) to the CEP engine through an enterprise service bus (ESB) that transforms them into events. When any of the situations of interest (complex events) are detected by the CEP engine, the I/OHub could receive from the action trigger some actions ("Simple action" in Figure 1) to be performed by the drone (e.g., a command for the autopilot to change the route) or the onboard actuators (e.g., switching the LEDs on), with the purpose of addressing such

detected situations. Thus, the I/OHub is crucial not only for heterogeneous data collection from sensors and the UAV itself but also for action distribution among actuators and the UAV.

The *OnBoardComputer* is the architecture's brain, responsible for the execution of the ESB, the CEP engine, and the action trigger at the same time. It adds the extra processing capabilities needed by a UAV.

The software components of the architecture are as follows.

The *ESB* is responsible for orchestrating the gathered heterogeneous data coming from multiple devices and in diverse formats. Several tasks take place at this component:

1. *Data reception*. The ESB receives the input data from the I/OHub.
2. *Event transformation*. All the input data are transformed into a common format to facilitate the event's processing by the CEP engine.
3. *Event enrichment*. Extra information (retrieved from a database, for example) can be added to these events.
4. *Domain dynamic generation*. Before the ESB sends each event to the CEP engine, the event's type will be analyzed to check if it is already recognized by the engine. If not, the new event type will be registered in the CEP engine.

The *CEP engine* is the software used to match event patterns over continuous event streams and to raise alerts about complex events created when detecting such patterns. These patterns are implemented using specific languages developed for this purpose, known as *event processing languages* (EPLs). In this work, patterns can be categorized into three different groups, according to the classification previously given: data acquisition, health management, and regulations. A priority level is assigned to each kind of pattern: low, medium, or high, respectively. This priority will be used to solve conflicts when more than one pattern is detected at the same time. These patterns can be specified by the user before flying a UAV or even at runtime –if a communication channel is available. By analyzing the events generated by the ESB and depending on the defined patterns (e.g., *batteryLevel* < 30%), the engine could create complex events (e.g., *Return_To_Home*). These complex events are received by the ESB and consumed by the action trigger.

The *action trigger* contains the code for dealing with the different complex events detected in the CEP engine. The response to a complex event can be either a single action (e.g., activating LEDs) or a complex action (e.g., going to the next location). Complex actions are sent as events to the CEP engine, through the ESB, to check if these complex actions violate any restriction. This new cycle is performed because a complex action could lead to new complex events. Several cycles could be possible, but, finally, a simple action or a set of simple actions is triggered. The actions taken by the action trigger are defined as follows: *WHEN (complex event) THEN Actions (Simple XOR Complex actions)*. The action trigger is connected to the I/OHub, in which simple actions are transformed into action data for specific actuators.

The *event pattern editor* facilitates for any domain expert the graphical definition of UAV event patterns and their detection, using real-time information flowing through the proposed architecture.

4. Architecture Implementation

The architecture has been implemented using a DJI F-450 UAV chassis with 750-watt rotors and an ArduPilot APM 2.6 autopilot. The hardware components chosen for the proposed architecture are the following:

- *The I/OHub*. An Arduino Uno microcontroller board was used. The programming of this microcontroller is simple, and it has libraries for a multitude of external components. It is connected by means of a USB port to the OnBoardComputer.
- *The OnBoardComputer*. A Raspberry Pi 2 microcomputer board was chosen. The processing capacities of this board are enough to support the execution of the ESB, the CEP engine, and the action trigger at the same time. It is connected to the I/OHub through a USB port.
- Data transmission between both components is done by using a Message Queue Telemetry Transport (MQTT) connector that supports the publish/subscribe ISO standard.

The following software components were chosen:

- *The ESB*. Mule was chosen because of its ability to integrate itself with cloud platforms as well as multiple tools and domain scenarios.
- *The CEP engine*. Esper was chosen as one of the best-known and most widely used open source CEP engines because of its ability to process thousands of events per second.
- *The action trigger*. The action trigger is implemented in DroneKit-Python, the most widely used software developer's kit by the UAV developer community because of its ability to facilitate application communication with UAVs over MAVLink.
- *The event pattern editor*. MEdit4CEP was used since it facilitates the definition and automatic code generation of event patterns (e.g., situations of interest to be detected in drones) through a graphical editor,[14] hiding the implementation details necessary to define such patterns from domain experts.

5. Application scenario

An application scenario is presented with the aim of illustrating the UAV architecture's functionality.

5.1 Scenario Description

Consider an area near an airport in which it is necessary to measure the environmental noise pollution produced by the takeoff and landing of aircraft. Since the measurement points can change from time to time, a solution based on UAVs (see Figure 2), instead of fixed sensors, is considered.

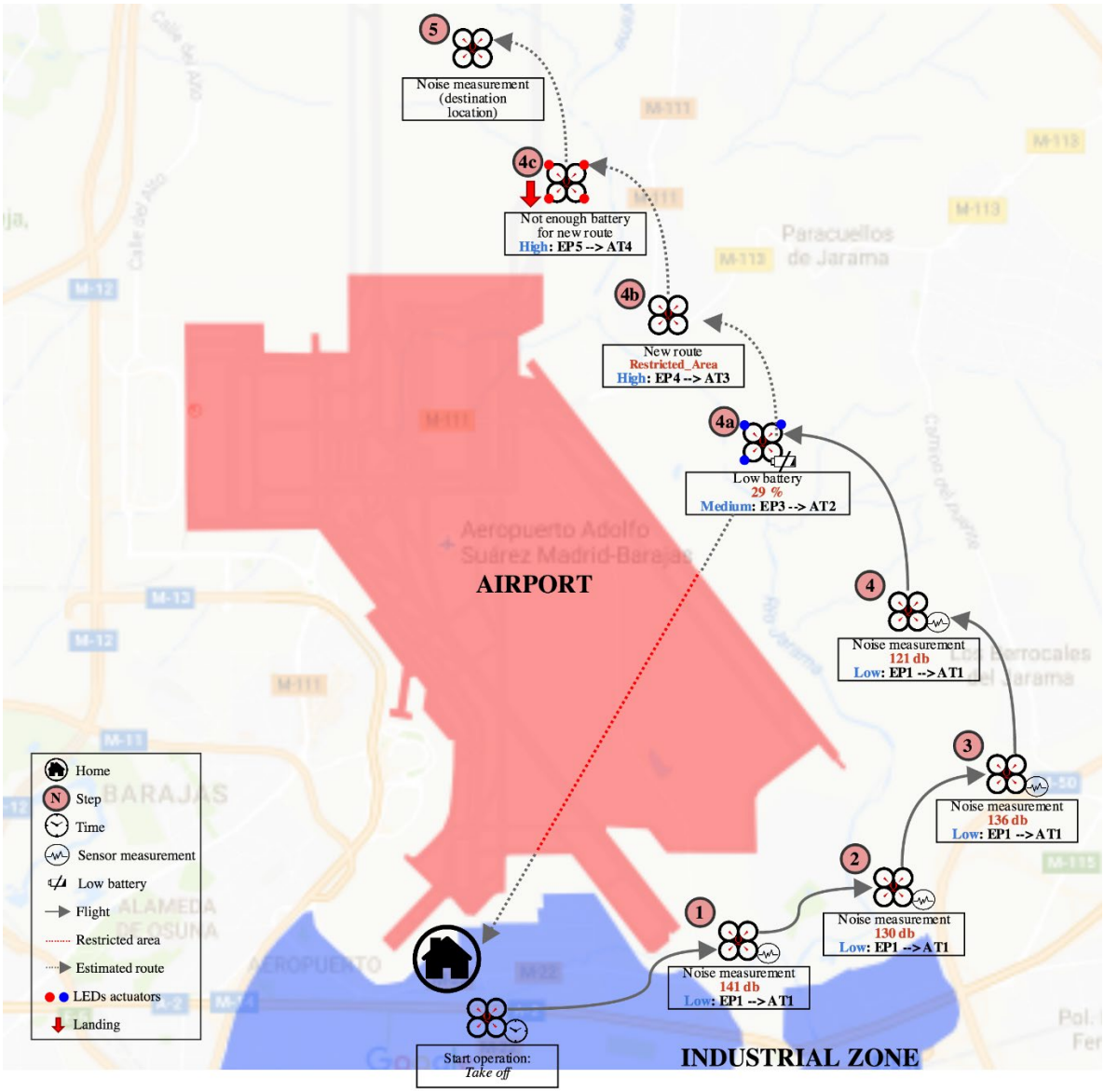

Figure 2. A practical example of applying the UAV real-time decision-making approach.

Each UAV is equipped with:

- a quadcopter with autopilot and GPS (one-second latitude, longitude, and altitude measurements),
- a sound meter,
- a saveLiPo sensor (a LiPo [lithium polymer] battery voltage tester) that checks the battery level every 10 s,

- light actuators,
- an I/OHub, and
- an OnBoardComputer executing the ESB, the CEP engine, and the action trigger.

Five points located around the area are monitored (numbered 1 to 5 in Figure 2). Noise monitoring starts at point 1 at a predefined time, in which the UAV takes off from home. To take a measure, the UAV must have previously landed.

If a noise level below 110 dB is detected, then the UAV waits. If the following measurement within 10 minutes at that location does not increase above 110 dB, then the UAV returns home. In contrast, if a measurement above 110 dB is detected, then it is necessary to monitor the related point. For the sake of clarity and to show the whole process in this example, the noise level is above 110 dB in all the considered points.

The example illustrates not only how event patterns can be defined but also how the planned route of the UAV can be changed in an autonomous way owing to the appearance of events with different priorities and types (health management and regulations).

5.2 Event Pattern Definition

Before starting the route for noise monitoring, it is necessary to define:

- the UAV parameters related to the available sensors and actuators,
- the data and their types coming from sensors to be collected by the I/O Hub and sent to the ESB,
- the event patterns, and
- their associated actions.

Since the main contribution of the proposed architecture extension is the use of the ESB and CEP engine, the example focuses on the event patterns and their associated actions. It is noteworthy that both the event patterns and their associated actions may be changed during the UAV flight, but, for simplicity, this is not illustrated in the example.

Five event patterns (EPs) have been proposed for this application scenario. These have been grouped according to their priority level.

Low priority:

- EP1 (high noise level). *WHEN(noise > 110 db) THEN Next_Location_Measurement*
- EP2 (low noise level). *WHEN(noise <= 110 db) THEN Return_To_Home*

Medium priority:

- EP3 (low battery level). *WHEN(batteryLevel < 30%) THEN Return_To_Home*

High priority:

- EP4 (restricted airspace). *WHEN(NewRoute IS Restricted_Airspace) THEN Recalculate_Route*
- EP5 (low battery level). *WHEN(Recalculate_Route) AND (batteryNeededForNewLocation>batteryLevel) THEN Landing*

The actions for the different complex events, created when detecting such patterns, are the following:

- AT1. *WHEN(Next_Location_Measurement) THEN goTo(nextLocation) [complexAction] → goTo(x, y) [simpleAction]*
- AT2. *WHEN(Return_To_Home) THEN NewRoute(Home) [complexAction] → goTo(x,y) [simpleAction] AND LEDs(blue) [simpleAction]*
- AT3. *WHEN(Recalculate_Route) THEN recalculateNewRoute [complexAction] → goTo(x,y) [simpleAction]*
- AT4. *WHEN(Landing) THEN LandingUAV [simpleAction] AND LEDs(red) [simpleAction]*

6. Results and Discussion

A video showing and explaining step by step the application scenario can be seen at http://qapps.unex.es/AutonomousUAV. The video makes use of an SITL (software in the loop) simulator to ease understanding.

Since we are not allowed to operate UAVs near airports according to current legal regulations, the application scenario was tested by operating the UAV around a large manufacturing company, instead of near an airport.

We obtained the following results, which can serve as a measure of the proposal's suitability.

6.1 The number of events generated per second

As a result of the complete execution of the described application scenario, 997 events were produced in total. In particular, a GPS location (latitude, longitude, and altitude) event was generated every second, a battery level (saveLiPO) event was produced every 10 s, and a noise/sound meter (soundmeter) event was created at each of the waypoints marked in the flight plan. Since the test duration was 14 min and 50 s, 1.12 events/s were processed.

The event-processing rate was noticeably much lower than the one supported by the Esper CEP engine: 500,000 events/s (www.espertech.com/esper). This high processing rate may be required in more complex scenarios –e.g., the Internet of drones.[15]

Moreover, additional tests were conducted with other types of sensors (luminosity, water, UV index, carbon monoxide, formaldehyde, ultrasound, temperature, humidity, and pressure) for other application scenarios. These tests produced 40 events/s approximately, without delaying or altering the system because of the number of events produced.

In the contexts in which the system can be used, the performance of the proposed architecture is adequate. The complete log file of the generated events is available at http://qapps.unex.es/AutonomousUAV.

6.2 The response time for notifying detected patterns

The pattern notification response time of the CEP engine, once the events were received and the conditions were fulfilled, was between 3 and 10 µs. Thus, it was insignificant, and this delay did not affect the performance expected from a UAV.

6.3 Off-board vs. onboard processing

The response delay for off-board processing was between 2 and 7 s, depending on the transmission size and the distance between the UAV and the ground control station.[7] In contrast, the data reception and processing performed onboard the UAV allowed a response of less than 0.7 s. This was enough to allow autonomous UAVs to detect critical situations and to automatically execute the appropriate actions.

6.4 Energy consumption

Energy consumption is the price that must be paid. The UAV used in the application scenario, without the proposed architecture, weighs 785 g plus 390 g for a 4,000 mAh LiPo battery. This UAV has an average consumption of 950 mAh with an electric potential of 3.7 V. Thus, under normal wind conditions (less than 5 Km/h) and pressure (1,000 hPa), the battery can last approximately 27 min at a 25 km/h constant speed.

By integrating this UAV basic architecture with the extended architecture proposed in this article, the UAV weighs 1,374 g total, with an average consumption of 1,372 mAh and autonomy for approximately 18 minutes under the same conditions. This drawback could be removed by adding lighter and lower-consumption devices, at the price of making the prototype more expensive.

7. Conclusion

Real-time decision making has become a crucial task to be performed by autonomous UAVs in many application fields. However, the limited processing capabilities of current UAV architectures make it difficult to accomplish this task. A CEP-based extension of the UAV basic architecture has been

proposed in this article. This approach satisfies the requirements for real-time decision making in autonomous UAVs proposed by international organizations, such as the FAA, as well as by the academic community.

The approach's benefits are as follows:

- It supports onboard real-time decision making in an autonomous way.
- The architecture facilitates for any nontechnological user the graphical definition of situations of interest to be detected and actions to be performed in UAVs, as well as their automatic code generation and deployment before or during the UAV flight.
- The planned UAV route can be dynamically changed upon the detection of prioritized situations summarizing UAV health problems or regulation violations.

To validate the approach's feasibility, a multipurpose UAV for dealing with several real-world application scenarios was proposed and built. Furthermore, a practical example of applying this UAV for measuring the environmental noise pollution produced by the takeoff and landing of aircraft has been described in this article.

Future work will include real-time decision making by coordinating several UAVs, which implies having a master UAV receiving events and distributing actions. This further work will be tested in the context of a project for fire detection and extinguishing control.

Acknowledgments

This work has been partially funded by the Spanish MINECO/FEDER funds (projects TIN2015-65845-C3-3-R, TIN2015-69957-R, and TIN2016-81978-REDT), Junta de Extremadura (project GR15098), and the POCTEP 2014-2020 program (project 0045-4IE-4-P). Juan Boubeta-Puig thanks the Quercus Research Group for their hospitality when he visited them at the University of Extremadura, where part of this work was developed.

References

1. Federal Aviation Administration, "FAA Releases Updated UAS Sighting Reports," 25-Mar-2016. [Online]. Available: https://www.faa.gov/news/updates/?newsId=85229. [Accessed: 15-Jan-2018].

2. E. Moguel, J.M. Conejero, F. Sánchez-Figueroa, J. Hernández, J.C. Preciado, and R. Rodríguez-Echeverría, "Towards the Use of Unmanned Aerial Systems for Providing Sustainable Services in Smart Cities", *Sensors*, vol. 18, p. 64, 2018.

3. R. Clarke, "What drones inherit from their ancestors," *Computer Law & Security Review*, vol. 30, no. 3, pp. 247–262, Jun. 2014.

4. E. Moguel, J. C. Preciado, F. Sanchez-Figueroa, M. A. Preciado, and J. Hernandez, "Multilayer Big Data Architecture for Remote Sensing in Eolic Parks," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 8, no. 10, pp. 4714–4719, Oct. 2015.

5. K. P. Valavanis and G. J. Vachtsevanos, "UAV Health Management Issues: Introduction," in *Handbook of Unmanned Aerial Vehicles*, K. P. Valavanis and G. J. Vachtsevanos, Eds. Springer Netherlands, 2015, pp. 995–997.

6. R. Evertsz, J. Thangarajah, N. Yadav, and T. Ly, "A framework for modelling tactical decision-making in autonomous systems," *Journal of Systems and Software*, vol. 110, pp. 222–238, Dec. 2015.

7. M. Asadpour, D. Giustiniano, K. A. Hummel, S. Heimlicher, and S. Egli, "Now or Later?: Delaying Data Transfer in Time-critical Aerial Communication", in *Proceedings of the Ninth ACM Conference on Emerging Networking Experiments and Technologies*, New York, NY, USA, 2013, pp. 127–132.

8. B. H. Y. Alsalam, K. Morton, D. Campbell, and F. Gonzalez, "Autonomous UAV with vision based on-board decision making for remote sensing and precision agriculture," in *2017 IEEE Aerospace Conference*, 2017, pp. 1–12.

9. C.-E. Hrabia, M. Berger, A. Hessler, S. Wypler, J. Brehmer, S. Matern, and S. Albayrak, "An Autonomous Companion UAV for the SpaceBot Cup Competition 2015," in *Robot Operating System*, Springer, Cham, 2017, pp. 345–385.

10. H. Kim and K. Choi, "A modular wireless sensor network for architecture of autonomous UAV using dual platform for assisting rescue operation," in *2016 IEEE Sensors*, 2016, pp. 1–3.

11. I. Sa, M. Kamel, M. Burri, M. Bloesch, R. Khanna, M. Popovic, J. Nieto, and R. Siegwart, "Build Your Own Visual-Inertial Drone: A Cost-Effective and Open-Source Autonomous Drone," *IEEE Robotics & Automation Magazine*, vol. 25, no. 1, pp. 89–103, 2017.

12. W. Liu, Z. Zheng, and K.-Y. Cai, "Bi-level programming based real-time path planning for unmanned aerial vehicles," *Knowledge-Based Systems*, vol. 44, pp. 34–47, May 2013.

13. C. D. Fulford, N. H. M. Lie, E. J. P. Earon, R. Huq, and C. A. Rabbath, "The Vehicle Abstraction Layer: A simplified approach to multi-agent, autonomous UAV systems development," in *7th International Conference on System Simulation and Scientific Computing*, Beijing, 2008, pp. 483–487.

14. J. Boubeta-Puig, G. Ortiz, and I. Medina-Bulo, "MEdit4CEP: A model-driven solution for real-time decision making in SOA 2.0," *Knowledge-Based Systems*, vol. 89, pp. 97–112, Nov. 2015.

15. R. J. Hall, "An Internet of Drones," *IEEE Internet Computing*, vol. 20, no. 3, pp. 68–73, May 2016.