Architectural Design of a Safe Mission Manager for Unmanned Aircraft Systems

Hector Usach*, Juan A. Vila

Universitat Politècnica de València, Camí de Vera s/n, València, 46022, Spain

Christoph Torens, Florian Adolf

German Aerospace Center (DLR), Institute of Flight Systems, Dept. Unmanned Aircraft, Lilienthalplatz 7, Braunschweig, 38108, Germany

Abstract

Civil Aviation Authorities are elaborating a new regulatory framework for the safe operation of Unmanned Aircraft Systems (UAS). Current proposals are based on the analysis of the specific risks of the operation as well as on the definition of some risk mitigation measures. In order to achieve the target level of safety, we propose increasing the level of automation by providing the on-board system with Automated Contingency Management functions. The aim of the resulting Safe Mission Manager System is to autonomously adapt to contingency events while still achieving mission objectives through the degradation of mission performance. In this paper, we discuss some of the architectural issues in designing this system. The resulting architecture makes a conceptual differentiation between event monitoring, decision-making on a policy for dealing with contingencies and the execution of the corresponding policy. We also discuss how to allocate the different Safe Mission Manager components to a partitioned, Integrated Modular Avionics architecture. Finally, determinism and predictability are key aspects in contingency management due to their overall impact on safety. For this reason, we model and verify the correctness of a contingency management policy using formal methods.

*Corresponding author
Email address: `hecusmo@doctor.upv.es` (Hector Usach)

Keywords: Software architecture, Automated Contingency Management, Formal methods, Partitioning, UAS

1. Introduction

Unmanned Aerial Systems (UAS) have been developing very quickly, thus presenting a challenge to traditional aviation. The European Aviation Safety Agency (EASA) is elaborating a new regulatory framework for the operation of UAS. The current proposal establishes three categories of UAS operation according to their risk levels [1, 2]. The *open category* is for low risk operations where safety is ensured through compliance with operational limitations, mass limitations, product safety requirements and a minimum set of operational rules. Authorization from a National Aviation Authority (NAA) is not required. The *specific category* is for medium risk operations and requires NAA authorization based on a risk assessment performed by the operator. A manual of operations lists the risk mitigation measures. Finally, the *certified category* is for large UAS flying in non-segregated airspace, the requirements for which are comparable to those for manned aviation. The International Civil Aviation Organization (ICAO) addresses this category in Doc. 10019 AN/507 [3]. According to that document, "only unmanned aircraft that are remotely piloted could be integrated alongside manned aircraft in non-segregated airspace and at aerodromes". This work is focused on Remotely Piloted Aircraft Systems (RPAS), a subclass of UAS.

The specific risks of an RPAS operation as compared to manned aviation are: *1)* reduced situational awareness of the remote pilot, and *2)* risk of losing the communication & control (C2) link between the remote pilot and the unmanned aircraft. In the former case, reduced situational awareness means that remote pilots, unlike pilots of manned aircraft in visual conditions, have reduced perception of environmental elements and events, which results in complex decision-making, especially during an emergency. In the latter case, the C2 link loss is a degradation or failure of the communication channel, which may result in the aircraft "flying not under command" [3].

UAS that aim to operate within the specific category, and ultimately within the certified category, are required to mitigate the aforementioned specific risks in order to achieve the *target level of safety*. This can be accomplished through several complementary approaches, such as setting the aforementioned operational limitations and even imposing certain functional requirements onto the on-board equipment. For example, some special technical equipment is often required to compensate for the reduced situational awareness, mainly Detect and Avoid (DAA) devices [3]. Another approach relies on operational flight planning and development of operations manuals with provisions for contingency handling.

In general, the functional requirements imposed on the on-board equipment exemplify the need for increased autonomous flight capabilities in RPAS. This is a focus of this paper. The software framework under development by the German Aerospace Center (DLR) for its research fleet of unmanned aircraft enables high level autonomous behaviors. One of its key software components is the automated Mission Planner and Execution (MiPlEx) system. MiPlEx performs real-time mission plan execution, 3-D world modeling, as well as algorithms for combinatorial motion planning and task scheduling [4, 5]. The Technical University of Valencia (UPV) is also developing a similar component based on the same architectural principles [6]. However, both *Mission Manager* implementations have so far only made use of operational limitations to achieve the target level of safety, e.g. operating in Very Low Level (VLL) or segregated airspace. As a collaboration between the two institutions, the goal is to safely increase the level of automation to develop a *Safe Mission Manager* System. This concept expands on the current Mission Manager by incorporating Automated Contingency Management (ACM) functions. The resulting system is expected to adapt autonomously to *contingencies*, while still achieving mission objectives by allowing some degradation on mission performance.

In this paper, we discuss the architectural design of the proposed Safe Mission Manager System. In addition, we also discuss how to allocate the different software components of the resulting system to an Integrated Modular Avionics (IMA) architecture. Finally, we propose using formal methods for specifying and verifying the contingency management policy. The rest of the paper is organized as follows: Sec. 2 presents related works in bibliography; Sec. 3 describes the initial Mission Manager System; Sec. 4 identifies the need for contingency management in RPAS; Sec. 5 discusses architectural considerations for integrating ACM functions into the previous Mission Manager; Sec. 6 presents the safety aspects relating to the software development of the resulting system; Sec. 7 develops the contingency management policy using formal methods; and finally, Sec. 8 concludes the paper.

2. Related work

The main topics of this paper are contingency management architectures, with special emphasis on UAS specific contingencies, and the use of formal methods in the software development process.

The primary guidelines for contingency management can be found in the proposals of regulatory frameworks for operating UAS currently being drawn up by Civil Aviation Authorities. These guidelines define risks and propose some risk mitigation procedures, among other things. UAS regulation in Europe is led by EASA, which has published the *Introduction of a regulatory framework for the operation of unmanned aircraft* [1], and the *Roadmap for the Integration of Remotely Piloted Aircraft Systems into the European Navigation System* [7]. A similar effort has been undertaken by the Unmanned Aircraft Systems Registration Task Force of the Federal Aviation Administration (FAA) in the United States [8, 9]. In addition, the ICAO has published the *Manual on RPAS* [3] to provide guidance on technical and operational issues applicable to the integration of RPAS in non-segregated airspace and at aerodromes.

There is an important research effort behind the regulatory proposals. The main research frameworks are the SESAR program in Europe [10] and the NextGen program in the United States. Some of the projects falling within these initiatives are also related to this work. One of the most relevant is the *Automated Contingency Management* (ACM) [11, 12, 13], which is a NASA-led research project in collaboration with Impact Technologies, LLC and Georgia Tech. ACM is designed to improve the reliability and survivability of safety-critical aerospace systems. The approach of ACM differs from the one presented in this paper in its focus on control optimization techniques rather than on the use of formal methods. One interesting extension to this approach is the work in [14] where human-machine interface considerations in contingency management are discussed. Another NASA project on drones is the Unmanned Aircraft System (UAS) Traffic Management (UTM). The UTM concept [15] was proposed as a traffic management scheme to enable civilian low-altitude UAS operations. This work's most relevant proposal with regard to contingency management is the level of automation. The proposed scheme ranges from a completely manual process relying on the operator (Build 1) to fully automatic, large-scale, system-wide contingency handling (Build 4).

The DLR has also conducted important research in the field of RPAS in the WASLA-HALE project for the High Altitude Long Endurance domain. Some research work focuses on the procedures and techniques for integrating UAS into controlled airspace [16, 17]. The proposed procedures are mainly related to C2 link failure conditions and communication with ATC. Another interesting aspect is the use of formal descriptions for enabling automatic reasoning on the consistency and correctness of the model requirements and the generation of on-line monitoring checks [18]. Case studies show that the process of formally writing down requirements is extremely helpful in understanding the domain inherent concepts [19].

The introduction of a Safety Monitor like the one in this paper is also suggested in [20]. The goal of the referenced work, however, is to expand the operational range and raise the autonomy level, rather than contingency handling. The work in [21] presents a predictive alerting method that uses multiple hypothesis prediction. It integrates all the onboard sensors and information sources with a stochastic estimator to obtain an accurate and reliable estimation of the aircraft state, which is key for contingency detection.

Regarding contingency management policies, C2 link loss is one of the most difficult to handle since any other contingency may also occur after it. The work in [22] presents a method for computing optimal lost-link policies for unmanned aircraft conducting surveillance alongside manned aircraft in a wildfire scenario. Another contingency handling policy especially important to UAS is collision avoidance. The work in the NextGen and SESAR programs led to the definition of a new Airborne Collision Avoidance System (ACAS) based on new logics, namely ACAS X. Its definition contains two particular variations: ACAS Xa for large aircraft, and ACAS Xu for unmanned aircraft. The work in [23] describes the specificities and challenges to the ACAS Xu system.

3. Initial Mission Manager architecture

The Mission Manager is the core system for performing the automatic guidance and control of the RPAS. Its functionality is based on the definition of a *Mission Plan* that basically specifies the RPAS route and payload actions. Both the MiPlEx framework and the Mission Manager developed at the UPV implement a software architecture based on the ideas of the three-tier (3T) architecture [24]. In general, a 3T architecture separates the intelligent control problem into three interacting layers named *Deliberative layer*, *Sequencing layer*, and *Reactive layer*. In this approach, the 3T concept has been applied from a flight guidance and control perspective, and the three layers have been renamed as *Path Planner*, *Guidance System*, and *Flight Director*, respectively, shown in Fig. 1:

The *Path Planner* is the high level component that has the ability to generate a reference trajectory for the Guidance System. As it is shown in Fig. 1, there exist multiple path planners that provide different path planning policies. The "Mission Planner" is a path planner that generates this trajectory based on the directives of the Mission Plan. In this approach, the Mission Plan is specified as a sequence of *flight legs* that implement the ARINC 424 *path terminators* [25].

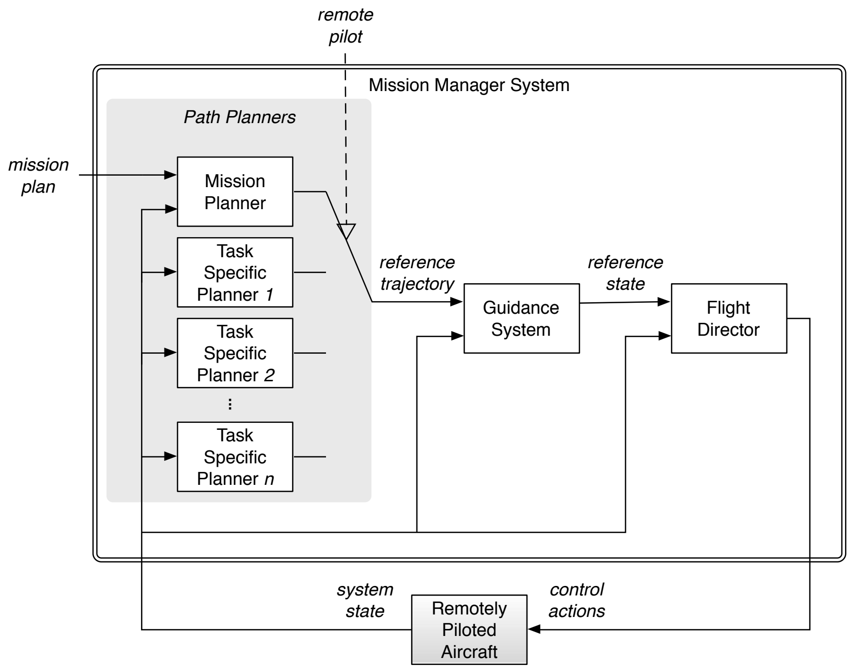


Figure 1: The initial Mission Manager architecture is structured into three layers: the Path Planners, the Guidance System, and the Flight Director.

Thus, the role of the Mission Planner is to provide each flight leg to the Guidance System in a sequential manner. In parallel to the Mission Planner, there exist some other *Task Specific Planners* for special tasks, such as the exploration of unknown terrain. From an abstract point of view, both the Mission Planner and the Task Specific Planners belong to a same class of objects with the ability to provide instructions for the Guidance System based on different criteria. The remote pilot should select the required Task Specific Planner manually in accordance with the current operational condition.

The *Guidance System* determines how to fly the reference trajectory provided by the active Path Planner and then activates the appropriate control modes of the Flight Director. To do so, the Guidance System uses a library of elemental maneuvers in the lateral plane (LNAV) and in the vertical plane (VNAV). LNAV maneuvers include straight maneuvers (with constant heading, with constant course, etc.) and turn maneuvers (with constant radius, with constant turn rate, etc). VNAV maneuvers include flight level maneuvers and climb/descent maneuvers (at constant speed, at constant vertical speed, etc). Thus, each time a new reference trajectory is received, the Guidance System plans a suitable sequence of maneuvers. The sequence of maneuvers in the LNAV plane is independent of the sequence of maneuvers in the VNAV plane.

Once the list of elemental maneuvers has been planned, the Guidance System activates the LNAV maneuvers and the VNAV maneuvers for carrying out the plan in a sequential manner. Only one LNAV maneuver and one VNAV maneuver can be active at a time. According to these active maneuvers, an interpreter activates the appropriate control modes of the Flight Director. The interpreter also computes the target values (the reference state) for the selected modes using different guidance algorithms. For example, the control mode for flying a turn with a constant radius is the heading control mode; and the algorithm that computes the target heading for this mode is based on the "carrot-chasing" algorithm in [26].

Each control mode is flown until some target event occurs. For example, the turn with a constant radius can be flown until the RPAS reaches a given waypoint, or until a given time-out occurs, for instance. When this target event is triggered, the Guidance System selects the next maneuver in the sequence. When the sequence of maneuvers is completed, the Guidance System notifies the Path Planner so that the high-level component will provide a new reference trajectory.

Finally, the *Flight Director* implements the control loops of the autopilot control modes. For example, the autopilot has the "heading control" mode, the "altitude control" mode, the "vertical speed control" mode, etc. The Flight Director can be commanded not only by the upper layers of the architecture, but also by the remote pilot directly.

3.1. Levels of automation

As can be seen in discussion above, the initial Mission Manager provides two levels of automation, named *manual operation* and *automatic operation*. Manual operation covers any mode in which the guidance actions are performed by the remote pilot. This includes direct control on the aircraft using the yoke, as well as giving the proper targets to the Flight Director. Manual operation is often used to execute flight procedures that are hard or unsafe to automate, such as take-off or landing procedures. By contrast, in the automatic operation, the guidance actions are performed by the upper layers of the architecture.

The level of automation plays a key role in aviation since allocating responsibilities between a human operator and an automatic system can lead to unsafe situations [27, 28]. Sheridan introduced a taxonomy with 10 automation levels [29] that characterize this interaction. According to this, manual operation is level 1, while automatic operation ranges from level 5 to 6. This means that the remote pilot is always in charge of decision making: he or she must always approve or reject the reference trajectory computed by the automatic system to ensure predictable behavior. As a result, automatic operation still puts some performance requirements on the C2 link, but not as stringently as it does in manual mode.

In this paper, we will describe the transformation of the previous Mission Manager into a *Safe Mission Manager* that handles contingencies; and we will discuss how to allocate the different software components of the resulting system to a partitioned environment based on the IMA concept.

4. Contingencies in RPAS

The aforementioned Mission Manager is able to perform the intended RPAS mission as long as it is executed in a nominal condition. However, at some point in the mission execution, a *contingency* may occur. Contingencies are unforeseen events that put other airspace users or people and facilities on the ground at risk [1, 2]. Several important contingencies have been identified in

RPAS [30]. Some of them are common to manned aviation –e.g. a traffic alert or the loss of control–, while others are specific to RPAS. Introducing contingency management functions into an RPAS means providing the on-board system with the ability to handle contingencies for the purpose of mitigating safety risks.

The source of contingencies are *faults*. These include both system component faults and human faults. Component faults occur when some aircraft component –such as an Inertial Measurement Unit (IMU), an engine, a Global Positioning System (GPS), or a barometric system– fails. Human faults refer to piloting errors, Air Traffic Control (ATC) errors, and any other faults related to inappropriate aircraft operation. There is a causal relation between faults and contingencies. For example, a GPS fault could cause inaccuracy in position determination, thus resulting in a mission boundary violation contingency. Another example is the loss of control due to a faulty IMU. Faults usually degenerate into contingencies after some short period of time. Early fault detection is key for effective contingency management.

Increasing the level of safety in an aircraft is usually accomplished using two complementary techniques: *fault tolerance* and *risk mitigation*. Fault tolerance is the ability to continue operating in the event of a failure. Risk mitigation is the process of incorporating defenses or preventive controls to lower the severity and/or likelihood of the projected consequence of a hazard. This is a two step process: when some fault is not tolerated and becomes a failure, then risk mitigation measures are needed. Faults affecting critical system components can be tolerated using redundancy. Full redundancy means replicating a component with exactly the same functionality and performance. Graceful degradation is the ability to maintain a limited or degraded functionality when some component fails. For example, GPS navigation could be replaced with dead-reckoning when the GPS fails, though at the cost of position accuracy. An important difference between system faults and human faults is that system faults can be tolerated to some extent by the use of redundancy, while human errors and inappropriate aircraft operation can only be handled through risk mitigation. One of the most encouraging measures proposed by EASA for risk mitigation

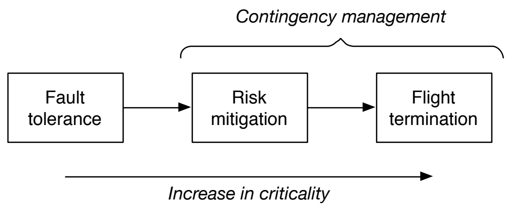


Figure 2: Framework of techniques for keeping safety in RPAS. Source: [31].

in RPAS is setting some boundary limits on RPAS operation. The goal here is to avoid the RPAS accidentally going out of its operations area or flying over dangerous or prohibited areas.

In the specific case of RPAS, there is a third barrier to enforcing safety in addition to fault tolerance and risk mitigation techniques: *flight termination*. Flight termination procedures make it possible to immediately ground the RPAS, for example by deploying a parachute, or using a self-destructive device. This option is crucial because it helps lower the severity of some risks and fault conditions in the safety assessment: the consequences associated with an unforeseen event can be minimized if it is possible to terminate the flight expeditiously. In any case, the flight termination procedure should be considered to be a last resort when no other option can mitigate the effect of the contingency, or when other actions have resulted ineffective for the situation being faced. A summary of the different techniques that can be applied for keeping an adequate level of safety can be found in Fig. 2, and is in agreement with the work in [31].

It is important to consider the most specific contingency in RPAS, which is the C2 link loss. Proper handling of this event is strictly required by ICAO for operating in non-segregated airspace [3]. Moreover, assuming that any combination of contingencies can happen in conjunction with C2 link loss, some sort of ACM ability is necessarily required to handle the situation. ACM functions make use of risk mitigation and flight termination techniques to generate recovery maneuvers or trajectories that effectively cope with a contingency situation without pilot intervention. Consequently, the level of automation of a Mission

Manager System performing ACM functions should be increased to provide *autonomous operation.* In this extended mode, the automatic system assumes flight guidance as well as decision-making responsibilities. This corresponds to a Sheridan level higher than 6, which goes beyond conventional FMSs [32, 33]. The next section discusses how to integrate ACM functions into the initial Mission Manager System.

5. Architectural considerations towards Automated Contingency Management in RPAS

The proposed Automated Contingency Management functions will be designed under the hypothesis that the RPAS has a Flight Termination System (FTS) that is able to terminate the flight expeditiously, and that this measure is an effective mechanism for enforcing safety. The need for this mechanism is supported by a number of aviation stakeholders [3, 32, 33, 34]. They also indicate that this action is to be triggered manually by the remote pilot as well as autonomously by the on-board system. This latter aspect implies that it is necessary to specify in the embedded software a list of predefined conditions for the automatic activation of the FTS.

Based on the initial hypothesis, the ability to engage the FTS becomes a safety-critical function. Critical software in aerospace is subject to strict validation and verification (V&V) processes defined by the DO-178 standard [35]. According to this document, a software component that cannot be completely verified at the design phase should not adversely affect safety. As a consequence, the list of predefined conditions that determines the automatic activation of the FTS must be hardcoded in the embedded software so that extensive testing can be performed at the design stage.

Commanding the flight termination action is a drastic decision, though. According to the safety framework in Fig. 2, less extreme risk mitigation measures could also be attempted for mitigating the risk inherent to contingencies in some circumstances. In these cases, we consider that the hardcoding of policies

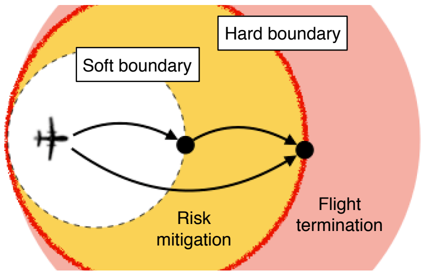


Figure 3: Safety thresholds managed by the Safety Monitor.

is not strictly necessary (because there is still a flight termination mechanism, and it can be thoroughly tested), and that having flexibility to specify some aspects of the contingency management policy would be beneficial to the final user. For this reason, we advocate separating ACM functions into two separate components, named the *Safety Monitor* and *Contingency Manager*, each with a different impact on the safety process.

Firstly, the role of the *Safety Monitor* is to check system behavior for unsafe states; and when an unsafe state is detected, to take the critical decision of whether a risk mitigation action is feasible, or whether the flight termination action is required instead. This decision depends on the criticality of the resulting state. Accordingly, the Safety Monitor manages two *safety thresholds*, represented in Fig. 3. When a first threshold is exceeded, the criticality is such that there is still a safety margin for attempting a risk mitigation action. The resolution of this state will be delegated to the Contingency Manager. But if the mitigation action fails and a second threshold is surpassed, the Safety Monitor will command the FTS to ensure that safety is not further compromised.

Secondly, when a risk mitigation measure is feasible, the *Contingency Manager* should react and *plan* the appropriate action for reducing the probability of infringing the second safety threshold, and ultimately attempting to recover the nominal condition of the RPAS. Mitigation actions, also called *contingency procedures*, replace the current reference trajectory of the RPAS with a new one that is more suitable for the contingency state being faced. For this reason, once a selection has been made, the Contingency Manager will notify the Mission Manager to execute the selected action.

However, defining a risk mitigation policy for dealing with contingencies is a complex matter. In some cases, there could exist multiple contingency procedures that could be effective, and the solution might depend on multiple factors. The extent to which this policy can be customized by the final user is another design issue that will be discussed later in this section. In any case, the fact that some aspects of this policy are open to modification by the final user implies that the Contingency Manager cannot be completely verified at the design stage, so it cannot be fully trusted.

As a result, the proposed differentiation between Safety Monitor and Contingency Manager provides an interesting tradeoff between safety and robustness: the Safety Monitor can enforce safety at any time during the mission, even when everything else fails; and the Contingency Manager enhances robustness of the system by providing solutions to contingency states. The implementation of the Safety Monitor must be hardcoded at the design phase so that extensive testing can be performed; in contrast, the verification of the Contingency Manager is subject to user modification. Safety aspects relating to the software development of this architecture will be further discussed in Sec. 6.

In summary, the execution of the proposed contingency management scheme will be composed of the following steps: *1)* monitoring the system behavior to detect and diagnose contingencies, *2)* deciding on a policy for dealing with contingencies, and *3)* executing the corresponding policy. In this scheme, the first step will be performed by the Safety Monitor; the second step will be performed at two levels by the Safety Monitor and by the Contingency Manager; and the third one will be performed by the Mission Manager or the Flight Termination System, depending on the selected policy.

Thus, the initial Mission Manager architecture will be extended with three new software components to perform ACM functions. The resulting *Safe Mission Manager* architecture is presented in Fig. 4. Note that the "Mission Manager System" box in this figure includes all the sub-components that were de-

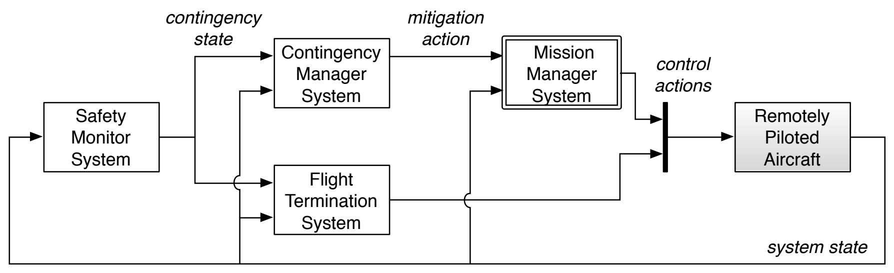


Figure 4: Safe Mission Manager architecture with four major components: Safety Monitor, Contingency Manager, Mission Manager and Flight Termination Systems.

picted in Fig. 1. In addition to the new software components, the Mission Manager module will also be internally redesigned to enable it to execute the *contingency handlers*. A contingency handler produces the reference trajectory of the mitigation actions that are enabled by the Contingency Manager. The sub-sections below will discuss the different components in more detail.

5.1. Detecting contingency states

Detecting unsafe states and triggering alarms is considered safety critical functionality. An interesting approach for this task is the use of formal specifications to derive *safety monitors*. The advantage of this methodology is that it enables very efficient monitors that can be verified because they are automatically derived from a formal specification. This technique has been an important research topic for checking software and hardware behavior in embedded systems [36]. We propose using similar techniques for system health management and the detection of unsafe conditions [37]. We discuss below how to integrate monitoring into the system architecture.

An aircraft is a distributed system consisting of a large number of independent subsystems. Critical system components are usually required to self-monitor, to perform fault detection and to report their faults. This is the case for the GPS subsystem, where Receiver Autonomous Integrity Monitoring (RAIM) systems are prescribed. Another example is the Airborne Collision Avoidance subsystem (ACAS), which is able to detect collision threats autonomously.

However, safety monitoring should be performed not only inside each subsystem but at the system level as well. This requires having access to the global system state because an unsafe situation can be formally specified as a predicate on the system state. We understand as a“global” state the aggregation of the states of a set of subcomponents of a distributed system [38]. As an example, consider the following unsafe condition: “the distance to the airport is greater than the mileage allowed by the reserve fuel remaining”. Checking this predicate involves knowledge as to the airport location, the aircraft position and the remaining fuel. Even if the subsystems that estimate these state variables are failure free and are not reporting any alarm, if these data are processed at a system level, then an unsafe situation can hold. In general, all contingencies derived from inappropriate aircraft operation involve the state of several system components.

Thus, online safety monitoring requires some centralized, high level component that coordinates all the distributed monitors and performs diagnoses at the system level. Performing this task implies that some knowledge about the normal behavior of the system is presented to the real-time reasoning. This knowledge can be typically developed using model-based or data-based techniques [39]: in model-based techniques, it is derived from theoretical models, while in data-based techniques, it is inferred from empirical experiments of fault-free operation.

One of the main problems is the reliability of the detection mechanisms. This refers to the probability of reporting false alarms or skipping true alarms. The main problem in creating a robust system is that sensors are imperfect and noisy. This results in uncertainty in the state determination. For this reason, the use of deterministic logic to define alarm conditions on the state variables does not guarantee reliable detection. Some proposed techniques for dealing with uncertainty are fuzzy logic [40], stochastic alarm detection techniques [41] and Markov decision processes [42]. Consequently, the Safety Monitor should also manage alarm thresholds.

Therefore, the centralized Safety Monitor should be modeled as a probabilis-

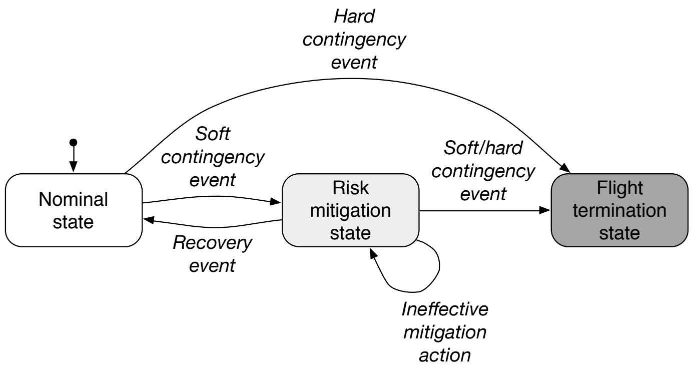


Figure 5: Generic model of a centralized Safety Monitor for RPAS.

tic state automaton [43]. However, current work is still at a conceptual level, so we will assume ideal detection mechanisms and avoid the uncertainty analysis. The resulting simplified model is depicted in Fig. 5. The state automaton in this figure starts at a nominal state where no contingency events have occurred. Within this state, the RPAS is flying the planned mission in a manual or automatic manner. According to the safety thresholds in Fig. 3, the Safety Monitor can trigger two types of *contingency events*: when the first safety boundary is exceeded, it triggers a “soft” contingency event. Such events make the system shift into a contingency state where a given mitigation action can be planned. In this state, the Contingency Manager will be enabled. When the second boundary is exceeded, the Safety Monitor will raise a “hard” contingency event, which results in a state where the only feasible action is flight termination; this will be handled by the Flight Termination System.

Once in a risk mitigation state, if the mitigation action turns out to be effective, a *recovery event* will bring the system back to the nominal state; otherwise, the system will remain in the same state and further mitigation actions can be planned. In addition, subsequent contingencies may also occur. However, we believe effective handling of nested alerts is highly unfeasible in most cases. For this reason, the approach for dealing with these conditions will be *prevention*: avoiding, whenever possible, the occurrence of new contingency events by using a safe and conservative design of the mitigation actions; otherwise, new events often lead to the flight termination state.

5.2. Defining a policy for dealing with contingencies

Once a risk mitigation state is entered, the next step is to decide on a policy for dealing with the resulting flight condition. The role of the *Contingency Manager* is to select a mitigation action that tries to partially complete the mission, probably in some degraded form, while maintaining safety. This requires addressing a decision-making problem that should balance the rewards with the risks associated with each possible action to maximize the probability of success, see Fig. 6. Once a decision has been made, the Contingency Manager will instruct the Mission Manager to execute this action.

In general, the decision-making process for selecting a mitigation action depends on not only the contingency event reported by the Safety Monitor, but also on other state variables. For example, the Contingency Manager could make different decisions about how to handle a traffic alert depending on whether the RPAS is flying in controlled or uncontrolled airspace. The decision is even more complex when several concurrent events are reported by the Safety Monitor. For this reason, the Contingency Manager also needs to have access to the global system state, as shown in Fig 4. Although the system state is huge and comprises many variables, it is often possible to limit the decision process

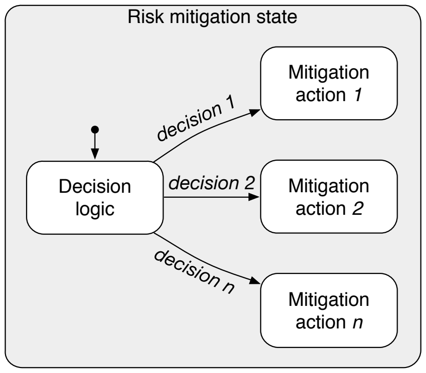


Figure 6: Decision logic for finding a feasible action after a contingency happens.

to the following subset: the aircraft position (including the segment of flight in which the contingency happens, and the airspace class) and the contingency event being faced.

As a result, the decision logic for selecting a suitable mitigation action can be also modeled as a state automaton. We call the specification of this automaton the *Contingency Plan*. It determines the behavior of the RPAS after a contingency occurs. The aim of this plan should be to reduce the time of flight of the Remotely Piloted Aircraft (RPA) experiencing the contingency [3]. The design of the Contingency Plan is a sensitive task because the resulting logic must: *a)* be deterministic, and ensure predictable behavior even without pilot intervention [3, 44]; and *b)* comply with current regulations (e.g. rules of the air [45] or procedures for emergency situations [46, 47]).

An interesting design issue is which level of customization is allowed in the design of the Contingency Plan; or, in other words, to what extent should the Contingency Plan be hardcoded into the embedded software. Two opposing trends affect this issue. One the one hand, having the flexibility to specify this plan on a mission basis –thus avoiding the hardcoding of policies– can make the remote pilot handle a contingency scenario in a more responsive way. On the other hand, since the specification of a state automaton with lots of states and transitions is a difficult and critical task, it must be verified; and the specification and verification of a safe state automation is considered to be beyond the scope of an RPAS operator.

For this reason, we have opted for the following mixed solution: the specification of the Contingency Plan is assigned to the system developer and is hardcoded into the embedded software; but if the mitigation actions have some *configuration parameters* defining *how* to execute this action, then they can be specified in a Mission Plan pre-flight. This solution implies that the validation of the Contingency Manager is subject to the validation of the Mission Plan at operation time: the Contingency Manager will be safe only if the Mission Plan specification is correct.

As an example, assume that "land at a designated landing site" is one possible mitigation action, and that the configuration parameters of this action are the list of suitable landing sites, as well as the routes towards these sites. Then, according to this proposal, the remote pilot will not be able to specify what state leads to the selection of this action, but he or she will be able to specify the possible landing sites, and the possible routes for reaching them. In addition, the decision on performing this procedure will be safe only if the Mission Plan has been approved by the corresponding aviation authority. Finally, the design of a Mission Plan specification that deals with contingency handling is a research problem that is currently under study.

5.3. Executing the selected mitigation action

To execute the selected policy, the corresponding *contingency handlers* must be provided to the on-board system. Contingency handlers implement the mitigation actions that will be executed in response to a contingency. In the proposed architecture, these handlers are executed by the Mission Manager because they can be seen as a special case of the Path Planners in Fig. 1. Thus, the goal of a contingency handler is to override the Path Planner guidance used during the nominal condition with some specific guidance based on safety concerns.

In general, the list of the required mitigation actions depends on the contingency events under consideration. As a general rule, the proposed contingency procedures should be similar to those for manned aviation [49]. For this reason, it is possible to classify them into two categories according to their impact on the planned route:

1. *Strategical contingency procedures* are suitable when the initial route is no longer feasible and thus a new mission has to be planned. The new reference path is often constructed with a more conservative design, with limited turns, vertical speeds, etc. This mechanism protects against exceeding the flight envelope during a contingency state (thus causing a nested alert), but also results in reduced aircraft performance due to the contingency being faced. One example of these procedures might be flying towards the alternative landing site.

2. *Tactical contingency procedures* are flight procedures that deviate the aircraft from the intended route temporarily, though the original mission may be resumed afterwards once the effect of the contingency has been mitigated. In contrast to strategical contingency procedures, tactical ones often demand high flight performances, like in a traffic avoidance maneuver.

Based on this differentiation, contingency handlers can be also classified into two classes according to their alternative guidance method: contingency handlers relying on the Mission Planner and contingency handlers requiring a Task Specific Planner. In general, all the tactical contingency procedures require a Task Specific Planner because the reference trajectory strongly depends on the type of contingency being faced. By contrast, strategical contingency procedures do not require a specific Path Planner be introduced but rather an alternative route definition be provided to the Mission Planner. How to perform smooth transitions between the guiding actions of different Path Planners is an interesting issue that exceeds the scope of this paper.

5.4. Performing the flight termination action

As was introduced in this paper, RPAS must often incorporate a Flight Termination System that is capable of safely bringing the vehicle back to the ground in case of severe contingencies. The work in [31] presents a survey of current and future technologies and procedures for performing a *controlled flight into terrain* (CFIT), including aerodynamic and ballistic terminations. Alternatives are self-destruct systems that allow an in-flight destruction to be performed without the loss of human lives [50]. However, this latter alternative may not be supported in RPAS that aim to operate within the certified category [49, 51].

In order to reduce the risk for people and ground facilities, the flight termination action should be performed in dedicated areas called Flight Termination Points (FTPs). These points should be specified in the Mission Plan, segregated by ATC and located in unpopulated areas or over the sea [17, 49]. For this reason, whenever possible, the execution of this action should be preceded by a strategical phase in which the RPAS tries to achieve the closest FTP.

6. Safety aspects relating to the software development

For the Mission Manager to be considered a *safe* system, it is necessary not only to provide it with ACM functions, but also develop it following a reliable software development process [52]. The software project for aerospace applications like the one presented in this paper shall demonstrate an adequate level of confidence in safety to comply with the aerospace standards for certification [34, 35]. The reference manual for the avionics industry is the DO-178 standard. It defines an explicit correlation between the severity of system hazards and the scrutiny to which that system is subjected. In particular, it establishes five *software levels* (Level A to E) related to the effect of five failure conditions (Catastrophic to No safety effect). The work in [53] particularizes the definition of these effects to the case of UAS. In DO-178, each level has a number of objectives that must be met in the software development process. In short, the verification effort increases with the software level of a component.

In regards to UAS, the EASA concept of operation states that system hazards are operation-centric [2]. According to this regulatory framework, unmanned aircraft operating in the open category are not subject to certification because the impact on safety of a software error is low: in this category, the aircraft is operated in visual line of sight (VLOS) and below 150 m, so a dedicated remote pilot is assumed to be present at all times of the operation. Accordingly, the remote pilot can take control of the vehicle at any point in the mission, and specifically after a contingency occurs. It can therefore be reasonably justified that the embedded software has no effect on the operation in terms of safety. However, large UAS operating in the specific category, and ultimately in the certified category, can eventually operate beyond the line of sight (BVLOS) of the remote pilot. In these cases, if the C2 link is lost, the software is essentially replacing the remote pilot; and, for this reason, it becomes safety-critical and must be verified [52].

In this section, we discuss the impact on safety of each architectural component of the proposed Safe Mission Manager system. We also propose the use of partitioning as a means of fault contention, and we allocate the different software components to a partitioning scheme that allows the software level of some architectural components to be downgraded. Finally, we propose the use of formal methods to facilitate the analysis and verification of the critical software.

6.1. Preliminary software level determination

Based on the safety framework in Fig. 2, it is possible to ensure safety as long as the RPAS has the ability to command the flight termination action in an expeditious manner, at any time and under any condition of the RPAS. In the ACM scheme proposed in this paper, the Safety Monitor is the software component with the ability to command this action autonomously, without the collaboration of any other system. Accordingly, in the safety assessment, a software error causing the loss of function of either the Safety Monitor or the Flight Termination System will have catastrophic effects. For this reason, the two systems will be considered the hardcore components for maintaining safety and should be assigned the highest software level.

In the case of the Contingency Manager and the Mission Manager components, the loss of any of these systems means that the flight path of the vehicle cannot be controlled. According to [53], this is thought to have hazardous effects on the operation of the RPAS as long as the vehicle is able to initiate a flight termination procedure; otherwise, such fault would have catastrophic effects. Consequently, assuming that the hardcore components of the architecture are able to safely terminate the flight, the Contingency Manager and the Mission Manager could be assigned a software level “B”.

The problem that emerges is that, according to the DO-178 standard, software components with common modes of failure cannot have different software levels [35]. That is, if a software error occurs at some component, but this error affects other functional components, then all these components should be assigned the software level associated with the most severe failure condition. In the case of the Safe Mission Manager in Fig. 4, if a serial implementation is conceived, then the loss of one component like the Mission Manager could cause a total system loss; and, as a result, all the components in the architecture should be assigned the same (highest) software level, which adds to the software development effort.

6.2. Architectural strategies for fault contention

To overcome this, the DO-178 standard proposes some architectural choices that can limit the impact of failures to ultimately demonstrate that sufficient independence between software components with respect to their failure modes exists [35]. If this is achieved, it is possible to separate safety-critical functions to independent modules with independent failure modes. Consequently, it is possible to downgrade the software level of some components by means of an appropriate architectural choice.

One of the possible architectural choices is *redundancy*. Redundant configurations mitigate hazards by replicating system components in different processors. So if a fault (either a software fault or a hardware fault) causes a system malfunction, the affected component can be replaced by a backup copy that provides the same functionality, but one executed on different hardware. The use of *dissimilar*, redundant components is required to avoid software development errors in this case. Redundant systems are common in aviation. Dual and triple redundancy is often required in critical aircraft systems [54, 55]. However, having dedicated hardware for each replicated application increases the system complexity, as well as development costs. Moreover, in the case of UAS, the reduced size and weight restrictions make redundancy hard to implement.

Another architectural choice that can limit the impact of failures is *partitioning*. Partitioned architectures, called Integrated Modular Avionics (IMA) architectures in aerospace [56], provide protection and separation among applications running on the same hardware. This way, failures occurring in one partition are not propagated to other partitions.

The support for IMA architectures is defined by ARINC-650 and ARINC-651 documents that specify general purpose hardware and software standards, and by ARINC-653, which specifies the application programming interface (API) [56]. Previous work by the authors presented an execution environment that supports the IMA concept [57]. The proposed framework relies on XtratuM, a hypervisor for real-time embedded systems developed at the UPV [58]. One of its guest real-time operating systems is LithOS [59], which is ARINC-653 compliant. The next section discusses how to exploit the IMA concept to allocate the different Safe Mission Manager components to the proposed execution framework.

6.2.1. Definition of the partitioning scheme

The definition of the partitioning scheme is a design issue. The simplest solution would be to allocate all the software components to a single partition. However, this does not exploit the IMA concept and implies that a fault occurring at some component can produce a total system loss. The opposite is allocating each function to a separate partition, but this unreasonably increases the system complexity. To overcome this, we propose an intermediate solution based on allocating software components according to their impact on overall safety.

Accordingly, the proposed partitioning scheme will be composed of two partitions, represented in Fig. 7. In this scheme, partition P0 allocates the software components that can contribute to failure conditions with catastrophic consequences; these are the Safety Monitor and the Flight Termination systems. Most of the software verification effort will fall on this partition. By contrast, partition P1 allocates the components whose failure is considered to have hazardous effects in the safety assessment, i.e. the Contingency Manager and the Mission Manager systems.

The advantage of this partitioning scheme is that a fault occurring at a component allocated to partition P1 will not be propagated to any component in

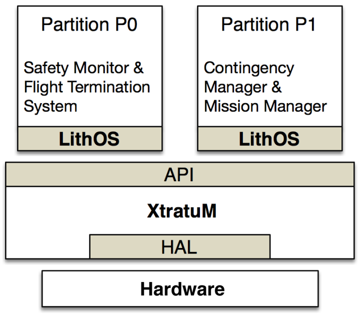

Partition P0
Safety Monitor & Flight Termination System
LithOS
Partition P1
Contingency Manager & Mission Manager
LithOS
API
XtratuM
HAL
Hardware

Figure 7: Safe Mission Manager partitioning scheme.

partition P0. In other words, a software error affecting partition P1 will not have catastrophic consequences because partition P0 is still able to perform the flight termination action (even if P1 fails). As a result, it is possible to downgrade the software level of the Contingency Manager and the Mission Manager from level “A” (in case of a serial implementation) to level “B” and reduce the number of verification objectives.

6.3. Formal methods for software verification

Along with fault contention mechanisms, reliable software design methodologies must also be followed in the software development process to assure that critical software is of high quality and error-free. Practices like *defensive programming* are often suggested for reducing software complexity and ultimately limiting the chance of introducing errors. For example, it is possible to limit the use of a programming language to a subset; this avoids the use of structures that could lead to non-deterministic behaviors. But the key mechanism for error prevention is the use of verification methods that ensure that errors entered into the software lifecycle get detected. In the previous version of the DO-178 standard (version B), the verification process mostly relies on generating a large set of test cases for different steps in the development process. However, the coverage of these tests cannot demonstrate the total absence of errors in the code [18]. Another shortcoming is that safety-related requirements are often difficult to test following such verification strategy [19, 60].

The last version of the DO-178 standard (version C) introduces the use of *formal methods* through the specific supplement DO-333 [61]. A formal method is defined as a formal analysis carried out on a formal model [35]. A formal model is a system description expressed with a formal specification language, i.e. with precise syntax and formal semantics. Such models can be useful in several phases of the development process, such as for the simulation of the system behavior or the reasoning over such system representation, among others. The most extended use is formal verification, however, which is the aim of the DO-333 supplement and of this work as well.

A formal specification makes it possible for system properties to be defined in a precise, consistent and complete way [62]. When used for verification, properties are deduced from the system requirements, and in the case of safety-critical system, from the safety requirements. Then, formal methods can be used to verify these properties against the model to reveal design errors or model inconsistencies, and ultimately to provide verification evidence for the certification process. One of the advantages of formal methods is that safety-critical properties can be checked more easily than with a conventional testing strategy [18, 19, 60].

The DO-333 supplement allows three classes of formal methods to satisfy certification objectives: theorem proving, abstract interpretation and model checking. This latter method is the focus of this work, while the remaining ones are omitted for brevity. In model checking, the model is represented as a finite-state machine (FSM), and the properties are formalized using temporal logic. Both the model and the properties are deduced manually from the system that is being verified. Then, the entire state space of the FSM is analyzed to check the validity of the formal properties, with the advantage that this analysis is fully automated. If a property is not satisfied, a counter-example is generated, and the model or the property can be refined and analyzed again. The resulting process is schematized in Fig. 8.

The application of model checking techniques to safety-critical avionics systems has increased in recent years [18, 19, 63]. The next section studies how to

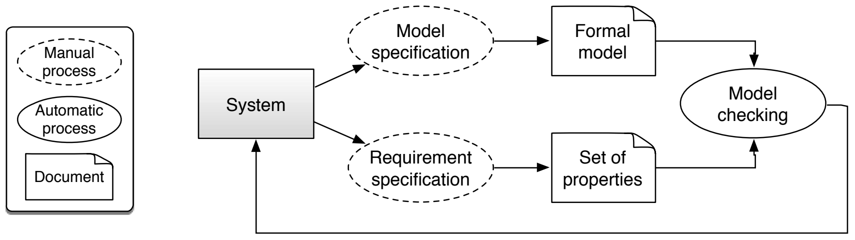


Figure 8: Model checking process. Source: [60].

exploit this method in the software development of the ACM functions presented in this work.

7. Formal design and verification of a contingency management policy: a case study

This section illustrates the use of model checking to verify the correctness of a contingency management policy. Based on the contingency management scheme in Sec. 5, contingency management policy depends on the Safety Monitor state automaton and on the Contingency Plan. Accordingly, we will develop a particular specification for these models, and we will identity some properties with which these models must comply. Then, we will translate both the models and the properties into a formal language and use the NuSMV model checking tool [64] to analyze the formal model with regards to the properties. The resulting process will ensure that the system design reaches a certain level of quality before it is implemented.

It is to be noted that the contingency management policy should respond to a system safety assessment. The type of contingency events to be handled and the required responses to these events often vary depending on the type of Remotely Piloted Aircraft (RPA) and the type of mission being performed. It is not the goal of this paper to present one given policy for a specific application. Rather, the purpose is to show how the development and verification of a realistic avionics application can benefit from formal methods. For this reason, the proposed policy will serve for demonstration purposes only. In any case, it will be developed in compliance with current airspace regulation and will aim to be generic enough so that it can be applied to a variety of RPA performing different mission types.

7.1. Specification of the Safety Monitor model

The proposed Safety Monitor model accounts for the occurrence of the five contingency events described below. These are the fault hypothesis of this case study, which are in line with the three key risk areas reported in [1]. Indeed, it is reasonable to expect these events to occur on any type of RPAS mission. If different fault hypothesis are assumed, then the subsequent design steps should also be rewritten.

a. *C2 link loss* is considered to be any situation in which the RPA can no longer be controlled by the remote pilot due to the degradation or total loss of the communication channel. This is likely even when redundant data link architectures are provided. Possible causes include screening terrain, ocean wave effects, malicious interferences, out of range, equipment failure and aircraft maneuvers.

   The C2 link mode of failure is not always a fail-stop failure (i.e. a “clean” failure). Usually, the C2 link experiences a degradation in which repetitive and intermittent unavailabilities or delays in the transaction time occur. It is therefore important to determine the sustained loss of link T_{sloss}, which is the time period at which the C2 link should be declared as being lost.

b. *GPS loss of performance* is due to the lack of satellite coverage or to a poor satellite signal. According to ICAO’s Performance-Based Navigation (PBN) [55], this performance degradation must be detected by RAIM systems. The navigation system continuously computes its Actual Navigation Performance (ANP), which is the maximum navigation error. When the ANP is higher than a specified Required Navigation Performance (RNP) then the system must signal a GPS loss.

c. *Loss of control* refers to situations where the pilot or the automatic guidance system are unable to control the aircraft, resulting in an unrecoverable deviation from the intended flight path. This is one of the most complex contingencies, involving numerous contributing factors that act individually or, more often, in combination. These factors include mechanical failures, weather conditions, sensor failures and ineffective aircraft control. Loss of control can be detected because the aircraft enters a flight regime outside its normal flight envelope. Ineffective control occurs when the guidance system provides the control loops with targets that cannot be achieved. Control systems attempt to prevent this situation by setting a flight envelope protection that puts some limits on attitude and speed targets; however, mechanical failures or extreme environmental conditions can make these preventive actions ineffective.

d. *Traffic alert* refers to the Detect and Avoid (DAA) capability which in RPAS relies on ACAS Xu equipment [23]. It provides two alarm thresholds: Traffic Advisories (TAs) and Resolution Advisories (RAs). TA is an indication alerting that a certain intruder is a potential threat. RA is an indication requiring the pilot to perform a quick maneuver deviating from the current flight path to provide separation from collision threats. False alerts are an important issue in collision avoidance systems. Variability in pilot behavior and aircraft dynamics make it difficult to predict where the intruder aircraft will be in the future. This requires a tradeoff between safety and operational considerations.

e. *Mission boundary limits violation* deals with two slightly different problems: *no-fly zones* and *geofencing*, see Fig. 9. On one hand, no-fly zones are locations where flight may be restricted by regulation or raise safety concerns. The RPAS shall not fly into these zones. On the other hand, *geofencing* consists of setting some boundaries or contention barriers to the area where the RPAS operation takes place and taking the proper measures to enforce these boundaries. The boundaries include both horizontal and vertical lim-

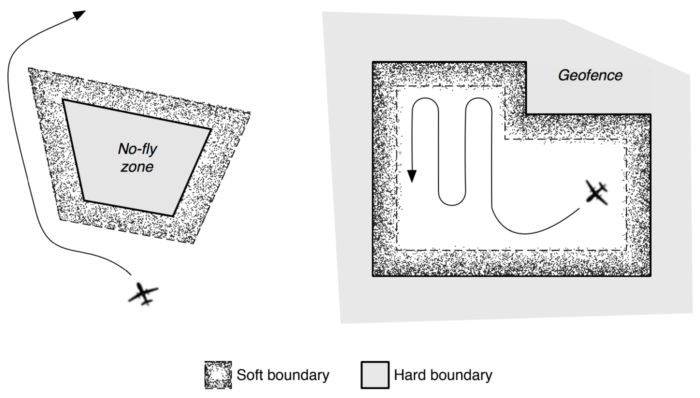


Figure 9: No-fly zones and geofencing.

its. The RPAS shall not fly out of these zones. When the whole mission takes place in segregated airspace, these limits also include the path from the aerodrome to and from the operations area.

No-fly zones and geofencing require continuous monitoring and checking of boundary violations. Two type of boundaries associated with the two alarm thresholds are usually considered. A *hard boundary* defines the limits that should never be trespassed. Violation of this boundary implies flight termination. A *soft boundary* defines the limits of the last chance to turn before violating mission boundaries. This limit strongly depends on the aircraft performance and the navigation performance.

7.1.1. Decision logic

The Safety Monitor model must diagnose each of the previous contingencies and decide whether the resulting state is to be handled by the Contingency Manager or by the Flight Termination System. In this case study, we determine that the occurrence of one single "soft" contingency can be addressed by the Contingency Manager, but any combination of nested contingencies or the occurrence of a "hard" event require instant flight termination. The resulting decision logic is modeled in Fig. 10. It shows an FSM with seven states: the nominal state (S_1), the flight termination state (S_7), and five risk mitigation states (S_2

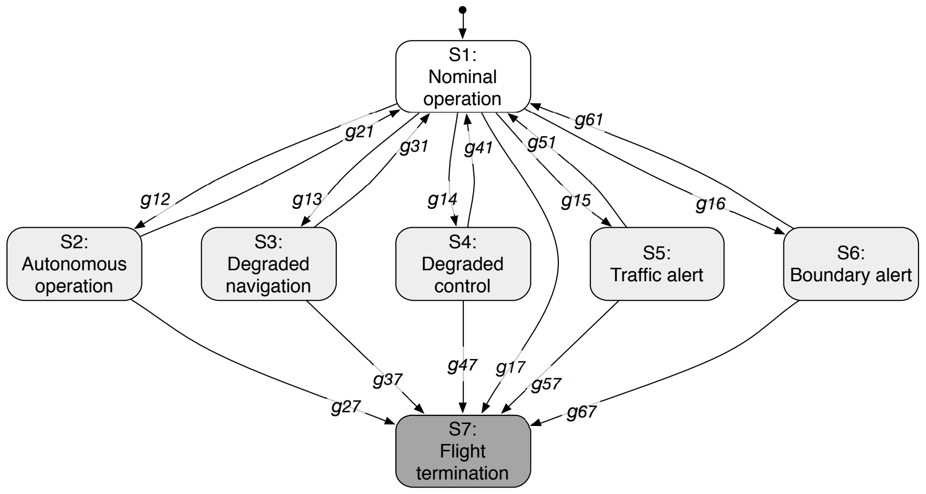


Figure 10: The Safety Monitor model presents one nominal state (in white) and several contingency states (in gray). State transitions g_{ij} are triggered by contingency events ($i < j$) and by recovery events ($i > j$).

to S_6, one per contingency under study). For example, *Autonomous operation* (S_2) describes *C2 link loss* conditions; *Degraded navigation* (S_3) implies reduced navigation capability due to *GPS loss of performance*, etc. Transitions between these states are labeled as g_{ij}, where i is the initial state and j is the resulting state. They can be triggered by contingency events (those where $i < j$) or by recovery events ($i > j$).

Some important properties of this model are: there should always be a transition for reaching the flight termination state in one step; the flight termination state should be a final state (i.e. one that has no successors); and one risk mitigation state must not be reached from another risk mitigation state in a direct manner.

Finally, note that if a different policy is conceived, then different states should be considered. For example, if risk mitigation measures are allowed for nested alert conditions, then additional states between single risk mitigation states (S_2 to S_6) and flight termination (S_7) should be added (for instance, *Autonomous traffic alert*).

7.2. Specification of the Contingency Plan model

The behavior of the Contingency Plan model is subject to the behavior of the Safety Monitor model: when a risk mitigation state is entered, the Contingency Plan determines an appropriate contingency procedure by means of a decision logic. The specification of the Contingency Plan model thus requires the definition of: *a)* a list of suitable contingency procedures for the risk mitigation states under study, and *b)* the corresponding decision logic. The list of procedures is, of course, dependent on the Safety Monitor states: if different states are considered, then the list of procedures should be reevaluated, as should the decision logic.

Based on the risk mitigation states in this case study (Fig. 10), we propose a series of procedures inspired by the emergency procedures for manned aviation and by the contingency options in Doc. 10019 AN/507 [3]. The list contains four contingency procedures of a tactical nature, including: *a.1) loitering*, *a.2) climbing to regain the signal* (either the GPS signal or C2 link signal), *a.3) avoidance maneuver*, and *a.4) reverting to manual control*; and two strategical contingency options: *b.1) landing* at a designated landing site, and *b.2) flight termination.*

The fact that the *flight termination* action is also available for the Contingency Plan warrants special attention. Even though the Safety Monitor has not entered the *Flight termination* state (meaning that there is still a safety margin for attempting a risk mitigation action), the Contingency Manager might be unable to find a more convenient option for a given contingency state. In these cases, *flight termination* will be commanded by the Contingency Manager, with the difference that it will not necessarily be performed expeditiously; by contrast, it can be preceded by a strategical phase in which the RPAS attempts to reach one of the FTPs specified in the Mission Plan, thus increasing safety.

Each of the aforementioned procedures can be executed under specified conditions only. For example, *revert to manual control* shall not be executed if the C2 link is lost. Another example is the *landing* procedure, which should not be executed after the *GPS loss* because it requires a high navigation accuracy. In order to develop such requirements, we propose using the Classification

Tree Method (CTM) [65], a graphical methodology that facilitates the requirement specification in the following manner: *1)* it helps to identify the relevant variables for the decision-making problem, as well as their possible values; *2)* it eases the analysis of nested alerts (if applicable); and especially, *3)* formal requirements can be directly extracted from the tree.

The resulting classification tree is presented in Fig. 11. It shows that the set of requirements are expressed in terms of the state variables introduced in Sec. 5.2; these are the tree branches. Therefore, the requirements are expressed as follows: *a)* when a specific value of one variable is required to select the procedure, it is marked with a black dot; in this case, the remaining values of this variable shall be left with no mark for this procedure. *b)* When the variable is not relevant for the selection of this procedure, then all their possible values are marked with a question mark.

7.2.1. Decision logic

Based on the previous list of options, we propose the following decision logic for the different risk mitigation states under study. Note that, for brevity, the following logic only accounts for contingencies occurring during the en-route phase, although the full policy can be easily extended:

a. After the *C2 link loss* event, the system is in the *Autonomous operation* state. In this state, the goal is to minimize the time of flight "not under command" [3]. According to ICAO, this can be achieved by either *landing* at a designated landing site, *climbing to regain the signal*, or performing the *flight termination* action. We assume that all of these options have some configuration parameters specified in the Mission Plan that determine the associated locations at which each procedure is to be performed, see Sec. 5.2. That is, the Mission Plan will specify all the possible airdromes and the allowed areas for climbing and terminating the flight. Based on this assumption, we suggest deciding whether to land or climb depending on which option is closer to the current position of the RPAS; and in case the resulting route is unfeasible (i.e. that the associated locations cannot be

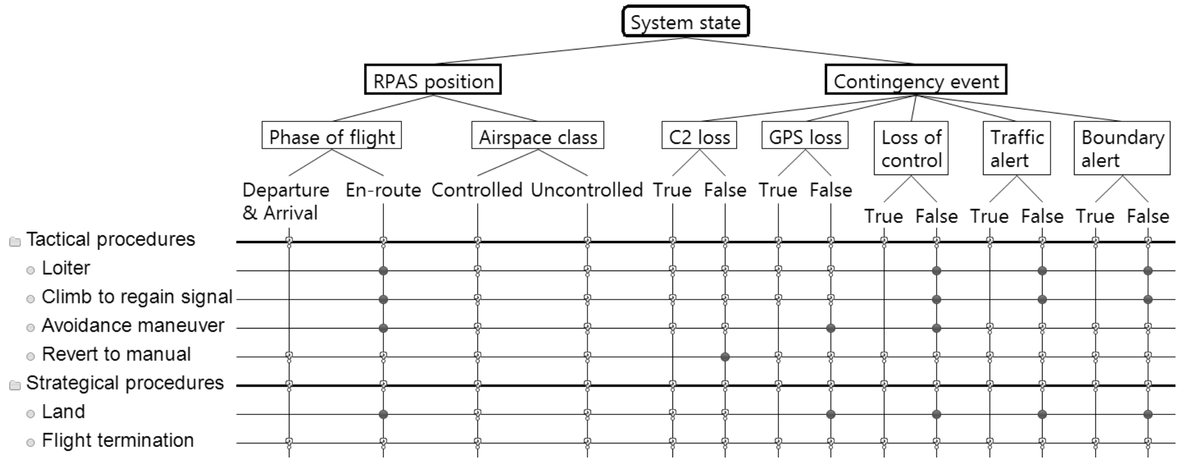


Figure 11: Preconditions for activating contingency procedures expressed using the Classification Tree Method.

reached), then *flight termination* will be selected.

b. A *GPS loss of performance* does not imply that the aircraft cannot determine its position by other means. Alternatives are ground-based navigation aids (if available) and dead-reckoning. However, the accuracy of these methods is lower than that of GPS and is probably insufficient for performing a given mission or complying with the RNP [55]. As such, after the loss of the GPS signal, the system evolves to the *Degraded navigation* state. In this state, if the RPAS is flying in controlled airspace, PBN specifications require it to *revert to manual control.* Otherwise, *climbing to regain the signal* or *loitering* are the most convenient options, depending on their associated locations and the current position of the RPAS.

c. The only suitable action for dealing with the *Degraded control* state is *reverting to manual control.* Note that, in order to maintain safety, this procedure should be performed without resulting in a transient that requires exceptional piloting skill or alertness from the RPA crew [66].

d. During a *Traffic alert*, the ACAS Xu equipment allows an automatic *avoidance maneuver* to be performed in order to regain the separation minima. ACAS Xu goes beyond conventional Traffic alert and Collision Avoidance Systems (TCAS) as it not only informs the pilot, but also executes the recommended evasion maneuver (though the remote pilot still retains the ability to override the proposed action).

e. Surpassing the soft boundary of a geofence or a no-fly zone causes a *Boundary alert* state. In this state, the RPAS starts an *avoidance maneuver* to go back inside the mission limits. If this is achieved, the original mission can be automatically resumed afterwards.

7.3. Formal specification and verification of the policy

In order to verify the proposed contingency management policy using the model checking techniques, it is necessary to translate the previous specification into a formal language. In this case, we will use the SMV language, and

NuSMV as the model checking tool. The resulting formal model of the contingency management policy will be composed of two modules: the Safety Monitor module, and the Contingency Plan module. An extract from these modules is shown in Figures 12 and 13, respectively.

As it can be observed, both modules start with a declaration of the state variables using the `VAR` command. This is followed by the specification of the FSM that describes the behavior of each model. In SMV, the FSM is declared using the `ASSIGN` command, followed by the initial value `init(state)` and the list of transitions `next(state)`. The last part of the modules specifies the properties with which the model must comply. In this case, the properties will be expressed using Computation Tree Logic (CTL) formulae, where `CTLSPEC` defines the start of a CTL expression, and "`->`" is the logical implication operator. The remaining notation is described in [64].

In the case of the Safety Monitor model, an extract of the FSM in Fig. 10 is shown in lines 58 to 72 of Fig. 12. In particular, it shows some transitions from the *Nominal* and *Autonomous operation* states. With respect to the formal properties, some relevant requirements presented in Sec. 7.1 are also formalized in this extract. For example, lines 121 to 125 specify that if a soft contingency event occurs when flying in a *Nominal* state, then the next state shall be a risk mitigation state; or that the *Flight termination* state shall be always reachable; and that it must be reachable in one step. Note that these properties have been specified using symbolic declarations like `recoveryEvent`, `hardEvent`, or `riskMitigationState`, which are omitted in the extract for brevity.

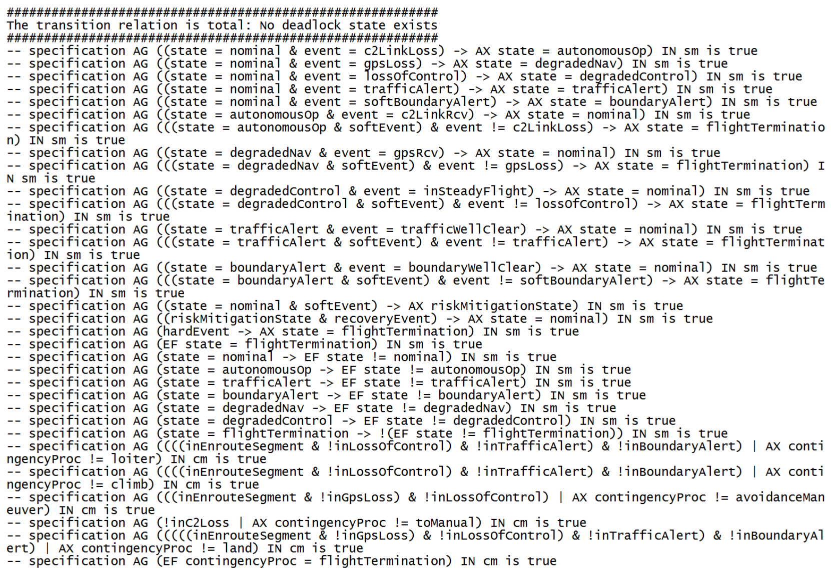
In the case of the Contingency Plan model, the corresponding module is invoked with four input parameters, see line 137 of Fig. 13. These are the variables on which the decision logic of the Contingency Plan relies: `sm_state` corresponds to the variable `state` of the Safety Monitor module; `shortestContingency-Route` specifies which contingency procedure has an associated location that is closer to the current position of the RPAS; `inEnrouteSegment` describes whether the current phase of flight is en-route or not; and `inControlled-Airspace` defines the airspace class. Note that the last two variables are modeled

```
MODULE safetyMonitor
VAR
  state : {nominal, autonomousOp, degradedNav, degradedControl, trafficAlert,
  boundaryAlert, flightTermination};
  event : {c2LinkLoss, gpsLoss, lossOfControl, trafficAlert,
  softBoundaryAlert, hardBoundaryAlert, c2LinkRcv, gpsRcv, inSteadyFlight,
  trafficWellClear, boundaryWellClear};
DEFINE
  softEvent := (event = c2LinkLoss | event = gpsLoss | event = lossOfControl |
  event = trafficAlert | event = softBoundaryAlert);
                                    ...
ASSIGN -- Safety Monitor state automaton
  init (state) := nominal;
  next (state) :=
    case
      --In S1: Nominal operation
      state = nominal & event = c2LinkLoss : autonomousOp; --g12
      state = nominal & event = gpsLoss : degradedNav; --g13
      state = nominal & event = lossOfControl : degradedControl; --g14
      state = nominal & event = trafficAlert : trafficAlert; --g15
      state = nominal & event = softBoundaryAlert : boundaryAlert; --g16
      --In S2: Autonomous operation
      state = autonomousOp & event = c2LinkRcv : nominal; --g21
      state = autonomousOp & (event = trafficAlert | event = softBoundaryAlert
      | event = gpsLoss | event = lossOfControl) : flightTermination; --g27
      --In S3: Degraded navigation
                                    ...
CTLSPEC AG (state = nominal & softEvent -> AX riskMitigationState);
CTLSPEC AG (riskMitigationState & recoveryEvent -> AX state = nominal);
CTLSPEC AG (hardEvent -> AX state = flightTermination);
CTLSPEC AG (state=flightTermination -> ! EF state != flightTermination);
CTLSPEC AG EF (state = flightTermination);
```

Figure 12: Extract of the Safety Monitor model in SMV language.

```
MODULE contingencyPlan (sm_state, shortestContingencyRoute, inEnrouteSegment,
inControlledAirspace)
VAR
  contingencyProc : {continueOriginalPlan, loiter, climb, avoidanceManeuver,
  toManual, land, flightTermination};
                                    ...
ASSIGN
  init (contingencyProc) := continueOriginalPlan;
  next (contingencyProc) :=
    case
      --In S1: Nominal operation
      sm_state = nominal & tacticalProc : continueOriginalPlan;
      --In S2: Autonomous operation
      sm_state = autonomousOp & inEnrouteSegment &
      shortestContingencyRoute = climb : climb;
      sm_state = autonomousOp & inEnrouteSegment &
      shortestContingencyRoute = land : land;
      sm_state = autonomousOp & inEnrouteSegment & shortestContingencyRoute
      != climb & shortestContingencyRoute != land : flightTermination;
      sm_state = autonomousOp & ! inEnrouteSegment : flightTermination;
      --In S3: Degraded navigation
                                    ...
CTLSPEC AG ((inEnrouteSegment & ! inLossOfControl & ! inTrafficAlert & !
inBoundaryAlert) | AX contingencyProc != loiter)
CTLSPEC AG ((inEnrouteSegment & ! inLossOfControl & ! inTrafficAlert & !
inBoundaryAlert) | AX contingencyProc != climb)
CTLSPEC AG ((inEnrouteSegment & ! inGpsLoss & ! inLossOfControl) | AX
contingencyProc != avoidanceManeuver)
CTLSPEC AG ((! inC2Loss) | AX contingencyProc != toManual)
CTLSPEC AG ((inEnrouteSegment & ! inGpsLoss & ! inLossOfControl & !
inTrafficAlert & ! inBoundaryAlert) | AX contingencyProc != land)
CTLSPEC AG EF contingencyProc=flightTermination;
```

Figure 13: Extract of the Contingency Plan model in SMV language.



```
##########################################################
The transition relation is total: No deadlock state exists
##########################################################
-- specification AG ((state = nominal & event = c2LinkLoss) -> AX state = autonomousOp) IN sm is true
-- specification AG ((state = nominal & event = gpsLoss) -> AX state = degradedNav) IN sm is true
-- specification AG ((state = nominal & event = lossOfControl) -> AX state = degradedControl) IN sm is true
-- specification AG ((state = nominal & event = trafficAlert) -> AX state = trafficAlert) IN sm is true
-- specification AG ((state = nominal & event = softBoundaryAlert) -> AX state = boundaryAlert) IN sm is true
-- specification AG ((state = autonomousOp & event = c2LinkRcv) -> AX state = nominal) IN sm is true
-- specification AG (((state = autonomousOp & softEvent) & event != c2LinkLoss) -> AX state = flightTerminatio
n) IN sm is true
-- specification AG ((state = degradedNav & event = gpsRcv) -> AX state = nominal) IN sm is true
-- specification AG (((state = degradedNav & softEvent) & event != gpsLoss) -> AX state = flightTermination) I
N sm is true
-- specification AG ((state = degradedControl & event = inSteadyFlight) -> AX state = nominal) IN sm is true
-- specification AG (((state = degradedControl & softEvent) & event != lossOfControl) -> AX state = flightTerm
ination) IN sm is true
-- specification AG ((state = trafficAlert & event = trafficWellClear) -> AX state = nominal) IN sm is true
-- specification AG (((state = trafficAlert & softEvent) & event != trafficAlert) -> AX state = flightTerminat
ion) IN sm is true
-- specification AG ((state = boundaryAlert & event = boundaryWellClear) -> AX state = nominal) IN sm is true
-- specification AG (((state = boundaryAlert & softEvent) & event != softBoundaryAlert) -> AX state = flightTe
rmination) IN sm is true
-- specification AG ((state = nominal & softEvent) -> AX riskMitigationState) IN sm is true
-- specification AG ((riskMitigationState & recoveryEvent) -> AX state = nominal) IN sm is true
-- specification AG (hardEvent -> AX state = flightTermination) IN sm is true
-- specification AG (EF state = flightTermination) IN sm is true
-- specification AG (state = nominal -> EF state != nominal) IN sm is true
-- specification AG (state = autonomousOp -> EF state != autonomousOp) IN sm is true
-- specification AG (state = trafficAlert -> EF state != trafficAlert) IN sm is true
-- specification AG (state = boundaryAlert -> EF state != boundaryAlert) IN sm is true
-- specification AG (state = degradedNav -> EF state != degradedNav) IN sm is true
-- specification AG (state = degradedControl -> EF state != degradedControl) IN sm is true
-- specification AG (state = flightTermination -> !(EF state != flightTermination)) IN sm is true
-- specification AG ((((inEnrouteSegment & !inLossOfControl) & !inTrafficAlert) & !inBoundaryAlert) | AX conti
ngencyProc != loiter) IN cm is true
-- specification AG ((((inEnrouteSegment & !inLossOfControl) & !inTrafficAlert) & !inBoundaryAlert) | AX conti
ngencyProc != climb) IN cm is true
-- specification AG (((inEnrouteSegment & !inGpsLoss) & !inLossOfControl) | AX contingencyProc != avoidanceMan
euver) IN cm is true
-- specification AG (!inC2Loss | AX contingencyProc != toManual) IN cm is true
-- specification AG (((((inEnrouteSegment & !inGpsLoss) & !inLossOfControl) & !inTrafficAlert) & !inBoundaryAl
ert) | AX contingencyProc != land) IN cm is true
-- specification AG (EF contingencyProc = flightTermination) IN cm is true
```

Figure 14: Verification results in the NuSMV console.

as boolean variables for simplicity.

Then, an extract of the FSM describing the decision logic of Sec. 7.2.1 is shown in lines 150 to 164 of Fig. 13. For example, the contingency procedure (`contingencyProc`) that will be selected during the *Autonomous operation* can be either `climb`, `land`, or `flightTermination`, depending on the RPAS condition. The last part of this extract shows the preconditions for activating each procedure, which were depicted in the classification tree of Fig. 11. In this case, they have been formalized using CTL formulae of the form `AG(s | AX !p)`, see lines 188 to 197; meaning that each occurrence of condition `p` (the activation of a procedure) is preceded by condition `s` (the required state condition) [67].

Finally, the previous modules shall be instantiated from a `main` module which is here omitted for brevity. The resulting SMV file can then be interpreted by NuSMV, which will check if the CTL specifications are satisfied by the model. The output of this program is shown in Fig. 14. It shows that the transition relation is total, and that all specifications hold. In summary, the results demonstrate the correctness of the proposed policy before it can be implemented.

8. Conclusions

Current proposals for a regulatory framework for UAS are operation-centric and rely on a risk analysis. In order to achieve the target level of safety, we proposed increasing the level of automation of the on-board system by inserting Automated Contingency Management functions. In this paper, we discussed the architectural design of the resulting system, which we called the Safe Mission Manager. The proposed solution provides an interesting balance between safety and robustness as it is able to adapt autonomously to contingencies by providing different risk mitigation policies before commanding the flight termination action.

We also discussed safety aspects related to the software development of this system. As a result of this analysis, we proposed the use of partitioning as a means for fault contention, and we allocated the different software components of the Safe Mission Manager to a particular partitioning configuration. This configuration is expected to reduce the software verification effort of the certification process as it makes it possible to downgrade the software level of some components of the architecture.

In addition, we proposed using formal methods for software verification. Formal methods, and particularly formal model checking, can help in analyzing the consistency, completeness and correctness of a software model. In this paper, we illustrated the model checking technique using the specification of a contingency management policy as a case study. We identified the most relevant contingencies for RPAS, proposed a list of mitigation actions and developed the corresponding decision logic. Finally, we modeled all these aspects using a formal specification and demonstrated that the software design is correct before implementation.

In the future, we plan to develop a novel Mission Plan specification that supports the definition of the configuration parameters of the different contingency options and to validate the proposed Safe Mission Manager System running in a simulation environment.

Acknowledgments

This work was supported by the Spanish Regional Government "Generalitat Valenciana" under contract ACIF/2016/197.

References

[1] European Aviation Safety Agency, Introduction of a regulatory framework for the operation of unmanned aircraft, 2015.

[2] European Aviation Safety Agency, Notice of Proposed Amendment 2017-05 (A). Introduction of a regulatory framework for the operation of drones: Unmanned aircraft system operations in the open and specific category, 2017.

[3] International Civil Aviation Organization, Doc. 10019, AN/507: Manual on Remotely Piloted Aircraft Systems (RPAS), 1st ed., ICAO, Montréal, Canada, 2015.

[4] F. Adolf, F. Thielecke, A Sequence Control System for Onboard Mission Management of an Unmanned Helicopter, in: AIAA Infotech @ Aerospace, AIAA SciTech, AIAA, Rohnert Park, CA, USA, 2007, pp. 2769–2780. doi:10.2514/6.2007-2769.

[5] F. Adolf, F. Andert, S. Lorenz, L. Goormann, J. Dittrich, An Unmanned Helicopter for Autonomous Flights in Urban Terrain, in: T. Kröger, F. Wahl (Eds.), Advances in Robotics Research: Theory, Implementation, Application, volume 9, Springer, Berlin, Heidelberg, 2009, pp. 275–285.

[6] H. Usach, J. Vila, A. Crespo, P. Yuste, A Highly-automated RPAS Mission Manager for Integrated Airspace, in: 5th International Conference on Application and Theory of Automation in Command and Control Systems, ATACCS '15, ACM, 2015, pp. 11–20. doi:10.1145/2899361.2899363.

[7] European RPAS Steering Group, Roadmap for the integration of civil Remotely Piloted Aircraft Systems into the European Aviation System, European Commission, 2013.

[8] Federal Aviation Administration, Integration of Civil Unmanned Aircraft Systems (UAS) in the National Airspace System (NAS), 1st ed., U.S. Department of Transportation, 2013.

[9] Unmanned Aircraft System (UAS) Registration Task Force (RTF) Aviation Rulemaking Committee (ARC), Task Force Recommendations Final Report, UAS Task Force, 2015.

[10] R. Román, F. J. Sáez-Nieto, C. Cuerno, RPAS Integration in Non-segregated Airspace: the SESAR Approach, in: 4th SESAR Innovation Days, SESAR Joint Undertaking, Madrid, Spain, 2014.

[11] L. Tang, A. Saxena, M. E. Orchard, G. J. Kacprzynski, G. Vachtsevanos, A. Patterson-Hine, Simulation-based Design and Validation of Automated Contingency Management for Propulsion Systems, in: Aerospace Conference, IEEE, Big Sky, MT, USA, 2007, pp. 1–11.

[12] A. Saxena, M. E. Orchard, B. Zhang, G. Vachtsevanos, L. Tang, Y. Lee, Y. Wardi, Automated Contingency Management for Propulsion Systems, in: European Control Conference (ECC), IEEE, Kos, Greece, 2007, pp. 3515–3522.

[13] L. Tang, G. Kacprzynski, K. Goebel, J. Reimann, M. E. Orchard, A. Saxena, B. Saha, Prognostics in the Control Loop, in: AAAI Fall Symposium on Artificial Intelligence for Prognostics, AAAI, Arlington, VA, USA, 2007.

[14] J. Li, G. Vachtsevanos, Human-machine interface: A framework for contingency management of complex aerospace systems, in: IEEE AUTOTESTCON, IEEE, National Harbor, MD, USA, 2015, pp. 80–86. doi:10.1109/AUTEST.2015.7356470.

[15] P. Kopardekar, Safely enabling UAS operations in low-altitude airspace, in: Unmanned Aerial Systems Traffic Management (UTM) Convention, NASA, Moffett Field, CA, USA, 2015.

[16] B. Korn, A. Udovic, File and Fly: Procedures and techniques for integration of UAVs in controlled airspace, in: 25th Congress of International Council of the Aeronautical Sciences, ICAS, Hamburg, Germany, 2006.

[17] M. Finke, Defining RPAS emergency procedures for controllers, remote pilots and automatic on-board systems, in: Deutscher Luft- und Raumfahrtkongress (DLRK), Braunschweig, Germany, 2016.

[18] C. Torens, F. Adolf, Using Formal Requirements and Model-Checking for Verification and Validation of an Unmanned Rotorcraft, in: AIAA Infotech @ Aerospace, AIAA SciTech, AIAA, 2015, pp. 1645–1657. doi:10.2514/6.2015-1645.

[19] D. Cofer, S. P. Miller, Formal Methods: Case Studies for DO-333, Technical Report NASA/CR-2014-218244, NF1676L-18435, NASA Langley Research Center, Hampton, VA, USA, 2014.

[20] A. Frey, T. Hanti, Expanding the operational range of UAS with an onboard supervisory instance, in: 34th Digital Avionics Systems Conference (DASC), IEEE/AIAA, Prague, Czech Republic, 2015, pp. 1–12. doi:10.1109/DASC.2015.7311437.

[21] M. U. de Haag, P. Duan, A multiple hypothesis predictive alerting (MHPA) method for improved aircraft state awareness, in: 34th Digital Avionics Systems Conference (DASC), IEEE/AIAA, Prague, Czech Republic, 2015, pp. 1–15. doi:10.1109/DASC.2015.7311443.

[22] Y. Kim, M. J. Kochenderfer, J. Grana, J. Bono, D. Wolpert, Optimal lost-link policies for unmanned aircraft, in: 34th Digital Avionics Systems Conference (DASC), IEEE/AIAA, Prague, Czech Republic, 2015, pp. 1–13. doi:10.1109/DASC.2015.7311430.

[23] G. Manfredi, Y. Jestin, An introduction to ACAS Xu and the challenges ahead, in: 35th Digital Avionics Systems Conference (DASC), IEEE/AIAA, Sacramento, CA, USA, 2016, pp. 1–9. doi:10.1109/DASC.2016.7778055.

[24] R. P. Bonasso, R. J. Firby, E. Gat, D. Kortenkamp, D. P. Miller, M. G. Slack, Experiences with an architecture for intelligent, reactive agents, Journal of Experimental & Theoretical Artificial Intelligence 9 (1997) 237–256. doi:10.1080/095281397147103.

[25] Aeronautical Radio, Inc., ARINC specification 424-15. Navigation System Data Base, 2000.

[26] G. J. Ducard, Fault-tolerant flight control and guidance systems, 1st ed., Springer-Verlag, London, 2009.

[27] K. Berry, M. Sawyer, E. Austrian, Human Factors Assessment of RNAV Approach and Departure Procedures, Technical Report, FAA Human Factors Division (ANG-C1), 2013.

[28] C. W. Johnson, The hidden human factors in Unmanned Aerial Vehicles, in: 26th International Conference on Systems Safety, International Systems Safety Society, Vancouver, Canada, 2008.

[29] T. B. Sheridan, W. L. Verplank, Human and computer control of undersea teleoperators, Technical Report, Massachusetts Institute of Technology, 1978.

[30] G. Wild, J. Murray, G. Baxter, Exploring civil drone accidents and incidents to help prevent potential air disasters, MDPI Aerospace 3 (2016). doi:10.3390/aerospace3030022.

[31] R. Stansbury, T. Wilson, W. Tanis, A Technology Survey of Emergency Recovery and Flight Termination Systems for UAS, in: AIAA Infotech @ Aerospace, AIAA SciTech, AIAA, Seattle, WA, USA, 2009, pp. 2038–2045. doi:10.2514/6.2009-2038.

[32] European Organisation for the Safety of Air Navigation, EUROCONTROL Specifications for the Use of Military Remotely Piloted Aircraft as Operational Air Traffic Outside Segregated Airspace, 2nd ed., 2012.

[33] A. P. Williams, P. D. Scharre (Eds.), Autonomous Systems: Issues for Defense Policymakers, NATO Supreme Allied Command Transformation, Norfolk, VA, USA, 2015.

[34] NATO Standardization Agency, STANAG 4671: Unmanned Aerial Vehicles Systems Airworthiness Requirements (USAR), NATO, 2009.

[35] Radio Technical Commission for Aeronautics (RTCA), DO-178C/ED-12C Software Considerations in Airborne Systems and Equipment Certification, Washington, D.C., USA, 2011.

[36] M. Leucker, C. Schallhart, A Brief Account of Runtime Verification, The Journal of Logic and Algebraic Programming 78 (2009). doi:10.1016/j.jlap.2008.08.004.

[37] F. Adolf, P. Faymonville, B. Finkbeiner, S. Schirmer, C. Torens, Stream Runtime Monitoring on UAS, in: S. Lahiri, G. Reger (Eds.), Runtime Verification. Lecture Notes in Computer Science, volume 10548, Springer, Cham, 2017. doi:10.1007/978-3-319-67531-2_3.

[38] F. Mattern, Virtual time and global states of distributed systems, in: Parallel and Distributed Algorithms, North-Holland, 1989, pp. 215–226.

[39] A. Dheedan, Distributed Online Safety Monitor Based on Multi-Agent System and AADL Safety Assessment Model, in: Q. A. Memon (Ed.), Distributed networks: Intelligence, security, and applications, CRC Press, 2008, pp. 317–345.

[40] E. Kiyak, F. Caliskan, Application of Fuzzy Logic in Aircraft Sensor Fault Diagnosis, International Journal of Systems Applications, Engineering & Development 6 (2012).

[41] K. Blin, M. Akian, F. Bonnans, E. Hoffman, C. Martini, K. Zeghal, A Stochastic Conflict Detection Model Revisited, in: 18th Applied Aerodynamics Conference, AIAA, Denver, CO, USA, 2000, pp. 4270–4279. doi:10.2514/6.2000-4270.

[42] M. J. Kochenderfer, J. E. Holland, J. P. Chryssanthacopoulos, Next-Generation Airborne Collision Avoidance System, Lincoln Laboratory Journal 19 (2012).

[43] M. O. Rabin, Probabilistic automata, Information and Control 6 (1963) 230–245. doi:10.1016/S0019-9958(63)90290-0.

[44] M. T. DeGarmo, Issues concerning integration of unmanned aerial vehicles in civil airspace, Technical Report, MITRE Center for Advanced Aviation System Development, 2004.

[45] International Civil Aviation Organization, Annex 2 to the Convention on International Civil Aviation: Rules of the Air, 10th ed., ICAO, Montréal, Canada, 2005.

[46] International Civil Aviation Organization, Annex 6 to the Convention on International Civil Aviation: Operation of Aircraft, 8th ed., ICAO, Montréal, Canada, 2001.

[47] Federal Aviation Administration, Airplane Flying Handbook (FAA-H-8083-3A), 2004.

[48] H. Usach, J. Vila, Reconfigurable Mission Plans for RPAS, 2018. Manuscript submitted for publication.

[49] Civil Air Navigation Services Organisation, ANSP Considerations for RPAS Operations, 2014.

[50] C. M. Eaton, E. K. P. Chong, A. A. Maciejewski, Multiple-scenario unmanned aerial system control: a systems engineering approach and review of existing control methods, MDPI Aerospace 3 (2015). doi:10.3390/aerospace3010001.

[51] R. J. Shively, A. Hobbs, B. Lyall, C. Rorie, Human performance considerations for Remotely Piloted Aircraft Systems (RPAS), Technical Report, Remotely Pilot Aircraft Systems Panel (RPASP), 2015.

[52] C. Torens, F. Adolf, L. Goormann, Certification and Software Verification Considerations for Autonomous Unmanned Aircraft, Journal of Aerospace Information Systems 11 (2014) 649–664. doi:10.2514/1.I010163.

[53] K. Hayhurst, J. Maddalon, P. Miner, G. Szatkowski, M. Ulrey, M. DeWalt, C. Spitzer, Preliminary considerations for classifying hazards of unmanned aircraft systems, Technical Report NASA/TM-2007-214539, NASA, 2007.

[54] European Aviation Safety Agency, Certification Specifications for Large Aeroplanes CS-25 Amendment 5, 2008.

[55] International Civil Aviation Organization, Doc. 9613, AN/937: Performance-based Navigation (PBN) Manual, 4th ed., ICAO, Montréal, Canada, 2013.

[56] Aeronautical Radio, Inc., ARINC specification 653-1. Avionics Application Software Standard Interface, 2003.

[57] H. Usach, J. Vila, A. Crespo, P. Yuste, Automatic Deployment of an RPAS Mission Manager to an ARINC-653 Compliant System, Journal of Intelligent & Robotic Systems (2017). doi:10.1007/s10846-017-0694-3.

[58] M. Masmano, I. Ripoll, A. Crespo, J. Metge, XtratuM: a hypervisor for safety critical embedded systems, in: 11th Real-Time Linux Workshop, Dresden, Germany, 2009.

[59] M. Masmano, Y. Valiente, P. Balbastre, I. Ripoll, A. Crespo, J. Metge, LithOS: a ARINC-653 guest operating for XtratuM, in: 12$^{\text{th}}$ Real-Time Linux Workshop, Nairobi, Kenia, 2010.

[60] R. Oliveira, Formal Specification and Verification of Interactive Systems with Plasticity: Applications to Nuclear-Plant Supervision, Ph.D. thesis, Université Grenoble Alpes, 2015.

[61] Radio Technical Commission for Aeronautics (RTCA), DO-333/ED-216 Formal Methods Supplement to DO-178C and DO-278A, Washington, D.C., USA, 2011.

[62] V. B. Mišić, D. M. Velašević, Formal specifications in software development: An overview, The Yugoslav Journal of Operations Research 7 (1997) 79–96.

[63] M. Webster, N. Cameron, M. Jump, M. Fisher, Towards certification of autonomous unmanned aircraft using formal model checking and simulation, in: IAIAA Infotech @ Aerospace, Garden Grove, CA, USA, 2012, pp. 1–15. doi:10.2514/6.2012-2573.

[64] A. Cimatti, E. M. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, A. Tacchella, NuSMV 2: An OpenSource Tool for Symbolic Model Checking, in: E. Brinksma, K. G. Larsen (Eds.), Computer Aided Verification, Springer, Berlin, Heidelberg, 2002, pp. 359–364. doi:10.1007/3-540-45657-0_29.

[65] M. Grochtmann, J. Wegener, K. Grimm, Test case design using classification trees and the classification-tree editor (CTE), in: European Quality Week, European Organization for Quality, 1995, pp. 30–39.

[66] European Aviation Safety Agency, Certification Specification: CS VLA - Subpart B, 2016.

[67] C. Baier, J.-P. Katoen, Principles of model checking, The MIT press, 2008.