Reconfigurable Mission Plans for RPAS

Hector Usach*, Juan A. Vila

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

Abstract

This paper deals with the problem of formally defining and specifying Mission Plans for Remotely Piloted Aircraft Systems (RPAS). Firstly, the profile of RPAS missions is highly variable and different from those of commercial flights. Route variability from the planned route is frequent due to operating conditions and, especially, contingencies. For this reason, RPAS Mission Plans should be reconfigurable: they should allow the nominal plan to be modified during flight time. Secondly, aviation authorities may require the ability to operate in an autonomous mode in response to Command and Control (C2) link losses. As a result, RPAS Mission Plans should specify all possible routings and behaviors in greater detail. The Reconfigurable Mission Plan concept introduced in this paper expands on current flight plans by providing a level of description that improves predictability and allows for reconfiguration, contingency handling, and higher levels of automation and pilot assistance. The paper presents a detailed discussion of RPAS contingency handling and develops a formal specification of the Reconfigurable Mission Plan concept. The paper also develops algorithms for dynamically configuring Mission Plan routes that might mitigate the effect of contingencies. Finally, the whole proposal is validated with a prototype implementation and a proof of concept.

Keywords: Flight planning, Mission replanning, Flight automation, Automated contingency management, RPAS

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

1. Introduction

The increasing number of civil applications using Unmanned Aircraft Systems (UAS) presents a challenge for the Air Traffic System. Remotely Piloted Aircraft Systems (RPAS) are a subclass of UASs that excludes all those "autonomous" vehicles for which no human action is necessary after take-off. According to the International Civil Aviation Organization (ICAO), "only unmanned aircraft that are remotely piloted (i.e. RPAS) could be integrated alongside manned aircraft in non-segregated airspace and at aerodromes" [1].

Civil Aviation Authorities are currently developing a new regulatory framework to ensure the safe insertion of RPAS into the civil airspace. To this aim, the ICAO published the guidance material in Doc. 10019 AN/507. From this document, it can be extracted that RPAS need to: *1)* demonstrate an Equivalent Level Of Safety (ELOS) to that of a manned aircraft, *2)* operate in compliance with existing aviation regulations, and *3)* appear transparent to other airspace users. These guidelines require, among other things, the use of flight plans for RPAS missions.

This paper addresses the problem of designing the formal specification of a *Mission Plan* for RPAS. RPAS Mission Plans are devised as a generalization of the *flight plan* concept used in manned aviation for traffic planning and traffic control purposes standardized by ICAO in Doc. 4444 [2]. In our proposal, flight plans should serve additional purposes: to provide automatic guidance whenever required and to specify RPAS behavior in case of contingencies so as to be predictable and suitable for automatization in case of a Communication and Control (C2) link loss event.

To meet the previous goals, RPAS missions should fulfill new requirements: *a) RPAS Mission Plans should be flexible to allow different mission profiles to be specified.* RPAS missions have a wide variety of profiles and are quite different from a typical "transport mission" between one origin and one destination. It is, however, difficult to define a "typical RPAS" profile. *b) RPAS Mission Plans should specify the flight segments to be covered in controlled and non-*

controlled areas. This is because Mission Plans serve additional purposes, such as automation. Additionally, RPAS can fly under non-conventional Air Traffic Control (ATC) services not included in controlled areas. One example is the NASA proposal for the airspace below 400 ft known as Unmanned Aircraft System Traffic Management (UTM) [3]. Another example would be an ATC unit specifically for the operations area, similar to the one used to access the operations area in firefighting. *c) RPAS Mission Plans should allow for dynamic trajectory replanning*. RPAS missions usually have a preferred or nominal route, but contingencies or some other flight conditions may require the current plan to be abandoned and replaced by an alternate one. This will be termed mission *reconfiguration* or *replanning*.

In accordance with these requirements, this paper develops the concept of *Reconfigurable Mission Plans*. These plans enable the nominal route to be modified at flight time and replaced with an alternate plan that might be regarded as a kind of "degraded mission", the goal of which is to mitigate the risks of a contingency. Mission replanning is unusual in commercial aviation, so conventional flight plans do not include it. The only alternate routes that conventional flight plans take into consideration are "alternate airports". However, aircraft behavior in case of some contingency is not specified any further, nor is it automated in flight plans of manned aviation: it is the responsibility of the pilot-in-command to make a decision and execute it. We believe these alternate plans need to be specified in the Mission Plan because RPAS can fly in a completely autonomous way; and, under this condition, any alternate route must be still predictable. If RPAS are operated in semi-automatic or manual mode, alternate plans can also be used to suggest possible options to the remote pilot.

Reconfigurable Mission Plans improve the specification of an RPAS mission by providing a level of description that allows for reconfiguration, contingency handling and higher levels of automation and pilot assistance. The Mission Plan proposal of this work will be modeled using Graph Theory and formalized using the Unified Modeling Language (UML) [4]. The approach for this proposal is to extend the capabilities of current trends and technologies in manned

aviation. Concepts like Performance-based Navigation (PBN) [5] or standards like ARINC-424 [6] will be used in so far as possible. ICAO recommendations will be also taken into consideration [1], especially when it comes to contingency handling. In accordance with this, the system should seamlessly accept conventional transport missions as a particular case.

The rest of the paper is organized as follows. Sec. 2 presents related works in the literature. Sec. 3 explains the need for performing automated contingency management functions onboard the RPAS and discusses how the specification of a contingency management scheme influences the design of the Mission Plan. Sec. 4 presents the RPAS Mission Management System which will fly Reconfigurable Mission Plans. Sec. 5 develops the Reconfigurable Mission Plan concept and models it using Graph Theory and UML. Sec. 6 presents the algorithms for dynamically configuring Mission Plan routes according to the current operational condition. Sec. 7 validates the proposed concepts in an RPAS mission example. Finally, Sec. 8 concludes the paper.

2. Related work

Two of the most widely used models for defining UAS missions are: *a)* a declaration with the list of waypoints of a mission, and *b)* a behavior-based description of the flight procedures of the mission. The first approach consists of setting a number of waypoints and associated commands to define the mission route. This is usually done through a Graphical User Interface (GUI). Navigation commands are used to specify movements to and around waypoints. Payload-related commands are used for setting options like the camera trigger distance or setting a servo value. A survey of autopilots using this type of specification can be found in [7]. On the other hand, the behavior-based paradigm is based on using a set of behaviors, which are a high level description of the flight procedures of a mission plan [8, 9, 10]. Behaviors are structured in a hierarchical way: complex behaviors can be built on top of lower-level ones. Such mission plans are usually system-specific, so they cannot be easily generalized.

None of the previous approaches assumes a controlled airspace where a number of well defined flight procedures are required by navigation charts or databases. The work by [11, 12] aims to enhance the level of automation of an RPAS in a controlled airspace using a formal specification of Mission Plans with semantically richer constructs to enable the definition of more complex flight plans and new RPAS-specific features. The Mission Plan is specified using the Extensible Markup Language (XML). The authors propose some Area Navigation (RNAV) leg types extensions for complex paths, as well as some control structures for repetitive and conditional behavior. Although these proposals have the same goal as this paper's and share the approach of extending current navigation concepts and technologies, they concentrate on complex routing and do not address in detail the topic of dynamically reconfiguring a flight plan and dealing with contingencies.

Regarding the use of autonomous systems in defense, the compilation work done by the North Atlantic Treaty Organization (NATO) [13] provides a good discussion on the definition of the levels of automation that can be introduced into an aircraft and their associated risks. It covers the challenging legal, ethical, policy, operational, and technical issues of autonomous systems from a multidisciplinary point of view. In [14], the authors present the development of the Intelligent Mission Planner and Pilot. The goal is to reduce operator workload and increase the level of automation. This proposal shares some goals with this paper, although in practice they concentrate on testing some previously proposed algorithms in the robotics field, like C-Space and collision avoidance, in a flight simulation environment.

Finally, in the field of software architectures for mission planners, one of the most significant for us has been the automated Mission Planner and Execution (MiPlEx) system developed at the German Aerospace Center (DLR) [15, 16, 17]. This architecture combines the main ideas of the behavior-based paradigm [18, 19] and the three-tier architecture defined by NASA [20]. The collaboration with this group has provided some ideas on how to introduce contingency management into the proposed architecture [21, 22]. Other software architectures

for mission planning have been proposed in the work cited above [11].

3. RPAS missions and the need for contingency management

RPAS are mainly used on missions involving dirty, dangerous, or dull tasks (too long, or too tedious for a crew). Some examples of typical RPAS missions include surveillance, image acquisition, and firefighting. Certainly, some of these missions have been traditionally flown using manned aircraft. Reasons for flying these missions using RPAS may be economical in nature, but an even stronger reason is *safety*: some risks and failure conditions can be less severe if RPAS is used instead of manned aviation. For example, a catastrophic failure condition in a manned aircraft is "one that would prevent continued safe flight and landing" [23]; however, the severity level of not landing in an RPAS can be lowered if it has some sort of *Flight Termination System* (FTS). FTSs are capable of safely bringing the vehicle back to ground using parachutes, for instance, or even destroying it without any loss in human life. This opens up new possibilities for contingency handling in RPAS.

Contingencies in RPAS are unsafe conditions that put other airspace users or people and facilities on the ground at risk. The work of [24] analyzes the most notorious contingencies in RPAS: some of them are common to manned aviation –such as the loss of control in-flight, or a collision alert–, whilst others are specific to RPAS. Among these, the loss of performance in the Command and Control (C2) link between the remote pilot and the unmanned aircraft is the most remarkable example.

The *C2 link loss* event is a key factor in the design of Mission Management Systems for RPAS. Even though the RPAS category excludes completely autonomous vehicles, aviation authorities require an *autonomous mode* (completely automatic) in some circumstances. According to ICAO and EUROCONTROL [1, 25], this mode is necessary in the last level of a hierarchy of collision avoidance mechanisms that get activated when Air Traffic Control (ATC) actions and the remote pilot actions fail, for instance. As a result, assuming that

any combination of contingencies can happen along with the C2 link loss, the Mission Management System should incorporate *automated contingency management* functions. Contingency management refers to the ability to handle contingencies, in an automatic or semiautomatic way, in order to maintain an acceptable safety level during the entire mission. Performing contingency management often demands a dynamic reconfiguration of the Mission Plan.

The problem with operating in an autonomous condition is keeping up the Target Level of Safety (TLS). The TLS of an automated system is achieved through a combination of automated functions and a human operator. According to NATO, the specified TLS for a fully automated system is always higher than one for a system with human intervention [13]. Specifically, the TLS with no human operator involved is increased by a factor of 10. For example, in the application of the Required Navigation Performance (RNP) concept to automatic landings, the probability of a catastrophic accident per landing is specified as 10^{-9}. In the risk analysis, the pilot is credited for reducing risk by a factor of 10. In this sense, in an Instrument Landing System (ILS) CAT II approach, where the pilot can veto up to a decision height of 100 ft, the automated system is required to achieve a TLS of 10^{-8}. However, if the level of automation is increased in a way that the automated system executes the flight procedure automatically, and simply informs the human about the decision of executing it, such a reduction cannot be applied and the TLS of the automated system will be 10^{-9}.

3.1. Design of a contingency management scheme

In previous work, the authors of this paper presented a contingency management scheme for RPAS [21, 22]. It was developed on the basis of avoiding as much as possible any hardcoding of policies: it should provide the right mechanisms to allow for different contingency options. For this reason, we proposed decoupling the following tasks: *1)* the detection and triggering of *contingency events*, *2)* the selection of a suitable *contingency handler*, and *3)* the execution of this handler. A contingency handler is a flight procedure that can mitigate the

risk posed by the contingency. For example, a contingency handler can implement some of the behaviors specified by ICAO, like "land at nearest designated landing site", "climb to regain the C2 link signal", or "flight termination" [1]. In this section, we analyze how the specification of this scheme influences the design of a suitable Mission Plan specification for RPAS.

To start with, we take as fault hypothesis five of the most representative contingencies covering the key risk areas reported by the European Aviation Safety Agency [26]. These are *loss of control*, *traffic alert*, *mission boundary limits violation*, *C2 link loss*, and *GPS loss of performance* (Global Positioning System). The nature of these events has already been discussed in the seminal works.

Then, in order to mitigate the risk caused by the aforementioned contingencies, we propose several contingency handlers. The required handlers vary depending on the type of contingencies under study; according to our fault hypothesis, we envision the following six handlers, classified according to their impact on the planned route:

- *Tactical contingency handlers* are flight procedures that deviate the aircraft from the intended route until the effect of the contingency has been mitigated; after which, the planned route is likely to be resumed. Among them, we envision avoidance maneuvers, loiter procedures, climbing in order to regain GPS signal or C2 link, and reverting to manual control.
- *Strategic contingency handlers* replace the original flight path with a new route. The new route often implies a degradation of the nominal plan. Among them, we include landing at a designated landing site and flight termination.

The last aspect of the contingency management scheme is how to carry out the mapping between contingency events and contingency handlers. In general, the selection of one handler depends on the system state. Although the system state is huge and comprises many variables, we can limit the selection process to some subset of variables: the aircraft position, the operational mode, and

the contingency event being faced. This way, the required decision logic for selecting a contingency handler can be modeled as a state automaton. We call the specification of this automaton the *Contingency Plan*. As was deduced in Sec. 3, the goal of the Contingency Plan is to minimize the flight time of the RPAS experiencing a contingency.

The key design issue is which part of this decision logic should be hardcoded into the embedded software in the design phase, and which part could be specified and customized on a mission basis in the Mission Plan. Two opposing trends affect this issue. On one hand, specifying a state automaton with lots of states and transitions is a difficult and critical task; and given that safety critical aspects in aviation are subject to the certification process, it seems inadequate to specify the automaton's behavior in the Mission Plan. On the other hand, if the remote pilot has the ability to specify the RPAS behavior after a contingency happens, he or she will handle the situation in a more responsive way.

For this reason, we opted for a mixed solution represented in Fig. 1: the specification of the Contingency Plan is assigned to the system developer and is pre-programmed into the embedded software; but if the contingency handlers have some configuration parameters, then they can be specified in the Mission Plan pre-flight. This means that the remote pilot cannot specify what state leads to the selection of one handler, but he or she can designate *how* to execute this handler.

The configuration parameters of most handlers include the target destination and the possible routes towards this destination. For example, in the case of the "landing" procedure, the remote pilot can specify the possible landing sites and the suitable routes for reaching them. The same occurs with the "loiter" procedure, the "regain signal" procedure, and the "flight termination" procedure. By contrast, "avoidance" maneuvers and "reverting to manual control" are contingency handlers whose required trajectory depends on in-flight conditions and thus cannot be customized in the Mission Plan pre-flight. Section 5 develops a Mission Plan specification that brings together all these aspects.

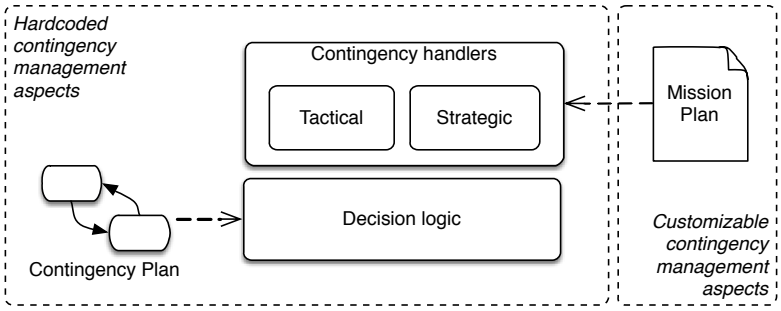


Figure 1: Contingency management specification scheme.

4. A Mission Management System for RPAS

In a conventional airliner, the *Flight Management System* (FMS) is the on-board system that performs the automatic flight guidance and control based on the directives of a flight plan. In general, this system is organized into the following two-layer scheme:

- A *Flight Manager Computer* (FMC) responsible for the Strategic Operation. It can handle the entire route and usually includes the Lateral Navigation (LNAV) mode and the Vertical Navigational (VNAV) mode.
- A *Flight Director* (FD) responsible for the Tactical Operation. It controls the current maneuver by programming the autopilot and auto-throttle modes for the speed, course, and altitude control. It can be commanded by the upper layer (the FMC) or directly by the pilot when the FMC is not engaged.

In the case of an UAS, the equivalent system is called the *Mission Management System*, which emphasizes the difference between flight plan and mission plan. It can be assumed that this system follows the same scheme as the Flight Management System. However, as was deduced in previous sections, such a system should enhance the Flight Management System functionality by providing an autonomous mode with automated contingency management functions. To this end, we shall first examine the required levels of automation and the

Table 1: Proposed operational modes in a Mission Management System for RPAS.

Operational mode	Responsible agent	
	Routing decisions	Guidance actions
Manual	Remote pilot	Remote pilot
Automatic	Remote pilot	Automatic system
Autonomous	Automatic system	Automatic system

role of the remote pilot in each mode; and then, we shall develop a software architecture that extends the previous layers in accordance with the proposed contingency management scheme.

4.1. Levels of automation

Much research has been devoted in the literature to address the *levels of autonomy* of an automatic system and the interaction between the automatic system and a human operator [13, 27, 28, 29, 30]. The work by Sheridan [27] is one of the seminal ones and for the first time it defined ten different levels of autonomy. A slightly different classification for the Autonomy Levels for Unmanned Rotorcraft Systems (ALFURS) is presented in [28]. This classification is closer and more suited to unmanned aircraft. In short, the ALFURS framework introduces a taxonomy with eleven automation levels according to the automated functionalities: guidance, navigation and control functions. Level 0 represents remote control, and Level 10 is fully autonomous capability.

Relating the levels of autonomy to the existing two-layer scheme of automation of conventional aircraft, we propose deriving a Mission Management System with three *operational modes*: *Manual*, *Automatic* and *Autonomous*, summarized in Table 1. The difference between these modes is made attending to decision-making at two different levels: *a)* who decides the route to be followed and makes strategic decisions (in the long term), and *b)* who is in charge of the guidance actions at the tactical level (in the short term), like flight level changes, turns, or speed selections. The proposed automation modes are described below in more detail.

- The *Manual mode* is the lowest level of automation, that corresponds to an ALFURS level 0. In this mode, all decisions in the long and short term are taken by the remote pilot. It is similar to navigating in a conventional airliner with the FMC disengaged and the pilot exercising direct control over the aircraft by either using the yoke or by setting the proper control targets in the FD. This mode demands high performance from the C2 link (bandwidth, delays, etc.) because real-time imaging and control are required for manual control.

- The *Automatic mode* is an intermediate automation level (ALFURS levels 1 to 4) where route selection is made by the pilot and guidance actions are performed by the automatic system. It is similar to navigating in a conventional airliner where the route has been programmed by the pilot in the FMC and then the FMC decides about the commands that will be delivered to the FD to follow the desired route. In this mode, the pilot is responsible for the strategic decision-making by choosing between different alternative routings, or by approving/rejecting proposals of the automatic system. This mode must also include some capability to override the actions of the automatic system using manual control or to abort some decision of the automatic system and reverting to manual control when necessary. For example, if the pilot has decided an automatic landing, the pilot is allowed to abort the maneuver before a deadline if something goes wrong. The performance requirements on the C2 link are not as stringent as in the previous mode.

- The *Autonomous mode* is the highest level of automation (ALFURS levels 5 and above). This mode does not exist in conventional airliners. It is intended for completely automating the aircraft mission when all communication links are down or some other serious contingency prevents the pilot to take control of the aircraft. In this mode, the on-board system must assume both strategic and tactical decision making. Autonomous operation can solve an out-of-control situation perhaps in a better or more

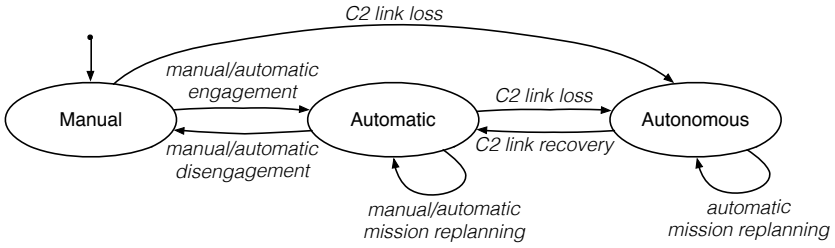


Figure 2: Operational modes and transitions between modes.

efficient way than a flight termination.

The resulting automatic system can be represented as a state automaton, as in Fig. 2. One important aspect to this machine is how transitions occur. Most often, these transitions are triggered manually by the remote pilot; however, they can also occur automatically after a contingency occurs. This is discussed below:

- Transitions *from manual to automatic* mode often occurs upon the pilot's decision to engage this mode. However, some contingencies, such as a collision threat, could be handled in an automatic collision avoidance mode without requiring the pilot's approval (although suppressing this approval is an important design decision). In other cases, these transitions require a pilot's decision on a rerouting to be followed and how the transition to the new route is to be performed. If no suitable route exists, then a stabilizing mode that maintains course, altitude and speed should be engaged.

- Transitions *from automatic to manual* mode can occur for three reasons: programmed disengagements, pilot decisions, or contingency handling strategies. Programmed disengagements occur when a transition to manual mode is pre-programmed in the middle of a sequence of automatic maneuvers, or at the end of this sequence. A disengagement can also be forced by pilot decision at any time: this can be done by simply acting on any flight control. Finally, if the autopilot control becomes unstable, it may cause the *loss of control* event. This is a severe contingency of-

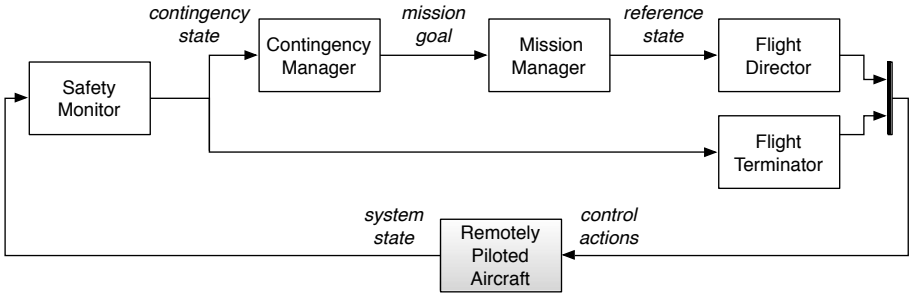


Figure 3: Software architecture of a Mission Management System for RPAS.

ten handled by reverting to manual control, an automatic (unexpected) autopilot disengagement procedure.

- Transitions *from manual or automatic to autonomous* mode are always triggered by the loss of all C2 links. All further contingencies must be handled in this mode. A recovery of the C2 link triggers a transition *from autonomous to automatic* mode.

4.2. On-board software architecture

In previous works, the authors presented the software architecture of a Mission Management System for RPAS [21, 22]. The proposed architecture extends the two-layer scheme of a conventional airliner to provide higher levels of automation and to enable automated contingency handling. The resulting architecture is structured into five software components named *Safety Monitor*, *Contingency Manager*, *Mission Manager*, *Flight Director*, and *Flight Terminator*, see Fig. 3. Each of the components is briefly introduced next.

The Safety Monitor and the Contingency Manager are responsible for the strategic decision-making. The role of the *Safety Monitor* is to check the system state looking for contingency states. When an contingency state is detected, it decides between a risk mitigation or a flight termination action. This decision is critical, so that is why the Safety Monitor is implemented as a completely separated component that has a failure independent mode from the rest of the components.

The referred decision is made based on the criticality of the contingency state. When there is a safety margin for attempting a risk mitigation option, the resolution of this state will be delegated to the Contingency Manager. But if the contingency state is considered safety-critical, i.e, there is a low probability of successfully handling the contingency, then the Safety Monitor will command the Flight Terminator to act. If this occurs, the Contingency Manager, the Mission Manager and the Flight Director will be instantly disengaged.

When risk mitigation is a plausible option, the *Contingency Manager* should react and decide a handling strategy. This way, the Contingency Manager can be also seen as a component that tries to minimize the probability of flight termination by defining a new *mission goal*. This goal usually consists of partially completing the mission in a degraded form, while ensuring safety. When no contingency has been declared, the mission goal is simply the *nominal goal*: to perform the intended mission. After a contingency occurs, the Contingency Manager interrupts the current mission execution to set a new goal and execute the corresponding actions. The selection of a goal is made in accordance with the Contingency Plan. Once a selection has been made, if the RPAS is operated in manual or automatic mode, the Contingency Manager informs the remote pilot about this goal. The pilot can decide between accepting the proposed contingency handling or changing to manual mode. If the RPAS is operated in autonomous mode, the Contingency Manager instructs the Mission Manager directly to execute the selected goal.

The *Mission Manager* is the architectural component responsible for the tactical decision-making: it defines a RPAS trajectory that allows to perform the current mission goal. It is thus equivalent to the FMC in a conventional airliner. The internal design of the Mission Manager (not represented in Fig. 3) is structured into two layers: the *Path Planner* and the *Guidance System*. A Path Planner can be considered as an abstract object with the ability to provide a reference trajectory to meet a goal. There exist multiple instances of Path Planners in the Mission Manager, but there is only one active instance at a time. The different instances use different criteria depending of the goal type.

A particularly important instance of the Path Planner is the "Mission Planner" which determines the trajectory based on the directives of a Mission Plan. Additionally, there are some "Task Specific Planners" which provide trajectory guidance under special contingency conditions such as, for example, collision avoidance. These planners usually provide deviations on the nominal path for a short period of time. Once a reference trajectory is defined, the *Guidance System* decomposes the trajectory into the corresponding horizontal (LNAV) and vertical (VNAV) profiles. Finally, these profiles generate targets for the *Flight Director*, which executes the control loops.

The last architectural component is the *Flight Terminator*. This system is responsible for conducting the flight termination action when no other option is effective for mitigating the effect of a contingency. The flight termination system is a mandatory equipment in most RPAS safety requirements. It can be implemented in different manners. Self-destruction systems, or parachute release systems are the most popular ones. Emergency landing systems are not considered a flight termination system in our proposal. In order to increase flight safety, this system must be completely independent from the rest of the systems and should be triggered by the Safety Monitor or by the remote pilot [13, 31, 32].

The proposed architecture has been prototyped in Matlab/Simulink, tested in a simulation environment, and deployed on a partitioned architecture based on XtratuM. XtratuM is a hypervisor for real-time embedded systems developed in our research group [33]. It is based on the Integrated Modular Avionics (IMA) concept and provides compliance with the ARINC-653 standard [34] by means of LithOS: a guest real-time operating system also developed in our research group [35].

5. Reconfigurable Mission Plans

Reconfigurable Mission Plans are based on the idea that an RPAS has a preferred or *nominal route*, though it is possible to modify this route at flight

time to respond to contingencies or pilot decisions. The alternate routes that result from changing the nominal route must be deduced from the specifications in the Mission Plan. To do so, all possible deviations from the nominal route should be specified in advance in the Mission Plan. The aim is to achieve completely predictable behavior, especially when the RPAS is in autonomous mode. In manual mode, the remote pilot can always override the Mission Plan if necessary.

As an example, consider a mission where the current goal is reaching an airfield via a preferred route. If the C2 link is lost, the Mission Plan can be reconfigured so the new mission goal can be to regain the C2 link signal. If the C2 link is restored, the original goal can be resumed, perhaps by using a different route. This way, the notion that a mission goal can be achieved through different alternative routes is another feature of Reconfigurable Mission Plans.

From the above discussion, a Reconfigurable Mission Plan is defined as a specification of all the possible *goals* of a mission and the associated routes that can be used to achieve these goals. One of these goals will be designated as the *nominal goal*; it defines the intended behavior if no contingency occurs. When some contingency occurs, the Mission Plan also defines alternative goals and one or more routes to manage the contingency. Alternative goals make use of different contingency handlers, such as the ones proposed in the contingency management scheme. In manual or automatic mode, it is the remote pilot who decides on following the Mission Plan directions. In autonomous mode, the Contingency Manager always uses the Mission Plan specifications to guide the aircraft.

The flight path of a route is structured into *segments*. A segment is a sequence of *legs* that correspond to a specific phase of flight or to a flight procedure (e.g. departure, arrival, etc). The Mission Plan comprises the set of all possible segments that can be used in a mission: different departure procedures, different routes, different payload related procedures, different landing procedures, etc. The set of all possible routes in a Reconfigurable Mission Plan can be deduced from a graph that results from the union of these segments.

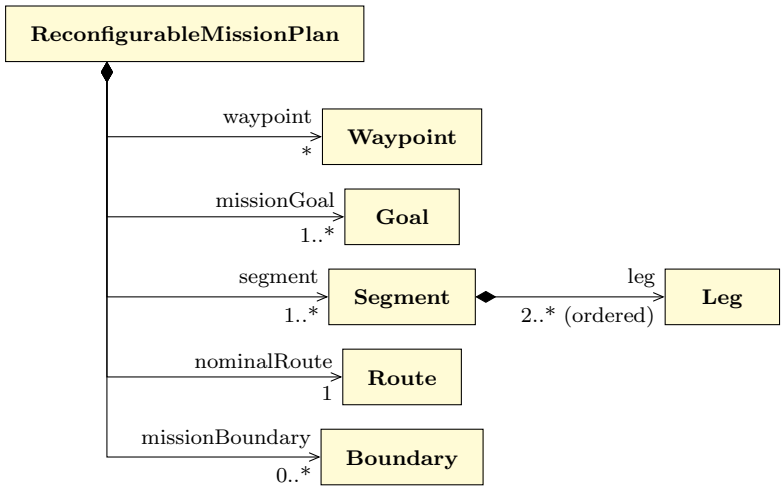


Figure 4: Structure diagram of a Reconfigurable Mission Plan object.

The formal description of a Reconfigurable Mission Plan and all its components is formally introduced below by using Graph Theory concepts and UML. At present time, we have not addressed the subject of designing a specific language or syntax to formally specify Mission Plans other than UML. Nevertheless, UML is a powerful representation that allows to derive a different specification or representation through a code generation process. This representation can be generated for a specific target system. For example, in our case we test Mission Plans on a Mission Manager prototype in Matlab/Simulink; so we need to translate the Mission Plan specification in UML to the Matlab language. This process can be automatized using a code generator. However, we will always refer throughout this paper to the original UML specification.

5.1. UML model

The UML model of a *ReconfigurableMissionPlan* object is shown in Fig. 4. The structural components of the plan are detailed below.

Attribute *waypoint* in Fig. 4 is the declaration of a set of relevant waypoints used in other Mission Plan attributes. This attribute is explained further in Sec. 5.2.

Attribute *missionGoal* is the definition of the set of all mission goals that a mission may have, one of which is the nominal one (see Sec. 5.3).

Attribute *segment* is the set of all possible segments that a mission may have. A segment is the description of a phase of flight or a flight procedure (see Sec. 5.5). The possible routes allowed in the Mission Plan are derived from this set of segments. One of these routes is the *nominalRoute*, which is the only route statically declared in the Mission Plan (see Sec. 5.6). The remaining routes will be dynamically specified, as will be shown in Sec. 6.

Attribute *missionBoundary* is a set of relevant airspace volumes that a mission may have (see Sec. 5.7). Examples are geofenced areas (like the area of operations) and no-fly zones. The reason the specification of these volumes are entered into the Mission Plan is to detect mission boundary limits violations. This is the basis for any sort of geofencing or warning system.

5.2. Waypoints

A waypoint is the most elementary object in a Mission Plan. It identifies a *point* over the earth surface using its WGS-84 coordinates. These points are used by other Mission Plan objects, like goals, routes, etc. In PBN (Performance Based Navigation), all fixes in a flight procedure are specified as waypoints and not by their relative position to some ground navaid.

The waypoint declaration of a Mission Plan identifies a set of well defined points, similar to how they are declared in a navigation database. A variant of the waypoint class is the *variable waypoint* class. This class is used to specify waypoints whose coordinates may vary depending on aircraft performance, and for this reason they do not appear in the initial waypoint declaration of the mission plan. These types of points will be used by some leg types in Sec. 5.4.

5.2.1. UML model

The UML model of a point is shown in Fig. 5. It shows an abstract class *Point* which has two two subclasses or specializations: the *Waypoint* class, and the *VariableWaypoint* class. Both subclasses present two attributes: an *id*(entifier)

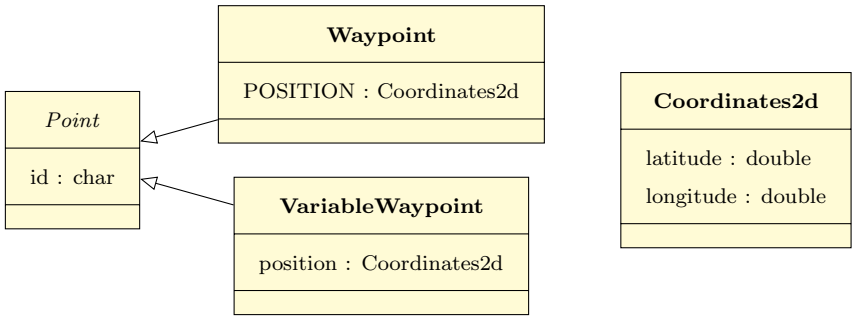


Figure 5: Structure diagram of a Waypoint object.

and a *position*. The attribute *position* is a *Coordinates2d* object for expressing latitude and longitude coordinates in decimal degrees. The difference between each subclass is that in the *Waypoint* class, the *position* attribute is constant (emphasized using capital letters), while in the *VariableWaypoint* class, the coordinates of the position can vary.

5.3. Mission goals

A mission goal represents a kind of "motivation" that leads RPAS behavior and trajectory. Examples of such goals may be flying to a location, landing, loitering, etc. In a Reconfigurable Mission Plan, the required goal types are mostly derived from the ICAO recommendations on contingency handling cited in Sec. 3.1. Once in flight, there is only one active goal at a time, which is selected by the remote pilot or the Contingency Manager, depending on the system configuration.

Formally, a goal has an *associated location* (a loiter point, an airport, etc.) that must be reached to achieve the goal, and an *enabled procedure* that is flown upon reaching this location. Table 2 shows some goal types with their associated locations and procedures. For example, the "loiter" goal is associated with a loiter point; reaching a loiter point allows the RPAS to perform a holding procedure.

One of the mission goals specified in a Reconfigurable Mission Plan is designated as the *nominal goal*. This goal describes the primary intended RPAS

Table 2: Goal types in the proposed Reconfigurable Mission Plan.

Goal type	Associated location	Enabled procedure
Fly-over	Waypoint	None
Loiter	Loiter point	Hold position
Regain signal	Waypoint	Climb trying to regain the C2 link or the GPS signal
Land	Airport IAF	Approach procedure
Flight termination	Flight termination point	Flight termination action

mission and it is the default goal to be achieved.

A goal can consist either of a single goal or a sequence of sub-goals or *stages*. For example, a goal may have an initial stage to fly to the operations area, a stage to perform the payload task inside the operations area, a stage to exit the operations area and a final stage to land at some given location. Stages can be also used to meet route constraints. For example, imagine accomplishing a "land" goal requires flying a standard arrival procedure with some required intermediate waypoints. This could be modeled as a sequence of "fly-over" stages followed by a "land" stage.

The *associated location* of a goal can be specified as a single point or as a set of alternative points. For example, a "land" stage may have a set of alternative Initial Approach Fixes (IAFs); each IAF corresponding to a different landing procedure to be chosen according to the airport configuration. Similarly, the operations area may have several entry points or several exit points that the pilot can select. In general, only one of the listed points can be enabled at a time: the decision maker agent (usually the pilot) must choose between the different options according to the current state or condition.

5.3.1. UML model

The UML model of a *Goal* object is shown in Fig. 6. Their attributes are an *id*(entifier), a *nominal* boolean to indicate if it is the nominal goal, and a sequence (ordered set) of *stage* objects. The UML's composition operator

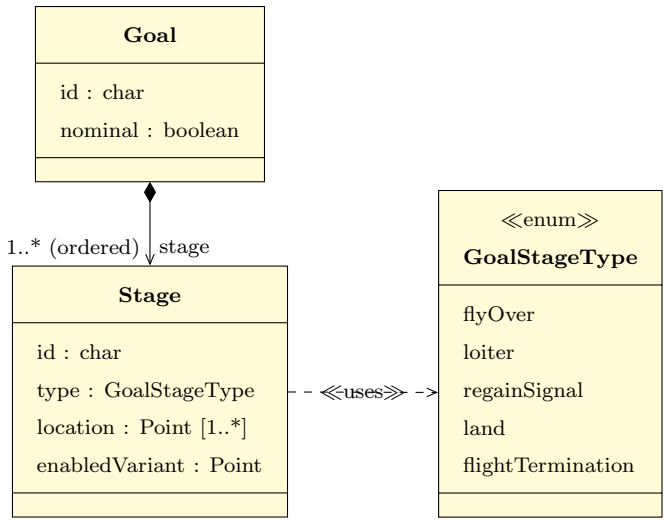


Figure 6: Structure diagram of a Goal object.

represents a parent-child relationship between a goal and the list of stages with a strong lifecycle dependency. The parent-child relationship means that the goal object has exclusive ownership over the stage object; and the lifecycle dependency states that if a goal object is deleted, then all its stages will also be deleted.

A *Stage* object is labeled with a *type* attribute, which is one of the enumerated values in Table 2. Stages have an associated *location* that has to be reached. Recall that several alternative points can be associated with one stage, representing different alternatives, so they are mutually exclusive. In this case, one of them has to be selected as the *enabledVariant*.

5.4. Legs

As it was introduced at the beginning of this section, a Reconfigurable Mission Plan route is structured into segments. A segment is defined as a sequence of legs. Legs are the most basic maneuvers in a segment. In conventional airliners, legs are specified in the FMC using the *path terminator* concept of ARINC-424. ARINC-424 is a standard that defines the format of navigation databases [6]. The standard defines a number of entities (like waypoints, airports, airways,

navaids, path terminators, etc), their attributes and their format. ARINC-424 path terminators have been adopted by ICAO in Doc. 8168 OPS/611 [36] to describe the basic flight maneuvers of RNAV flight procedures.

Path terminators provide higher flexibility and richness of behaviors than simple "waypoint navigation" used by most Mission Plan proposals. A path terminator is defined using two letters, the first one describing the type of *path*, and the second, the *termination condition*. For example, the most common path terminator is the *track to fix*, coded as "TF". This coding defines a maneuver that the FMC must implement as navigating a *path* that is the shortest orthodromic track between two fixes. The *terminator* indicates that the maneuver ends when the target fix is reached. But path terminators provide additional path types (like arcs with constant radius, paths with constant heading, paths with constant course, etc), and additional types of terminators (reaching an altitude, manual termination, etc).

The advantage of using path terminators is that it allows flight procedures to be defined using PBN standards [5] and, ultimately, provides compatibility with commercial FMC technologies. Path terminators were originally intended to describe airliner maneuvers. We propose to extend this concepts to RPAS. This requires to introduce new path terminators to describe RPAS specific maneuvers in a way that is close to the ARINC-424 standard. Our proposal is a resulting set of path terminators referred to as *Extended Path Terminators* (EPTs). New features introduced by EPT's are:

a) *New types of paths and new types of terminations*, gathered in Table 3. Examples of new paths include the "S-path" to describe scanning patterns around a geographical area; while new terminators include the "L-terminator" (used to specify a maximum number of laps) or the "T-terminator" (used to specify a given time limit). Accordingly, some examples of new EPTs would be: SL (perform a scanning pattern for a number of laps), ST (scan for a given time) or SM (scan to manual termination).

b) *Combinations of paths and terminators not included in the standard.* Exam-

Table 3: Extended Path and Terminator codes.

Path		Terminator	
Code	Description	Code	Description
S	Scanning pattern	L	Lap limit
		T	Time limit

ples of new combinations of existing paths and terminators not defined in the standard include: RA (helicoidal ascents or descents to an altitude), or RM (orbit until pilot intervention).

On the other hand, EPTs also restrict some features defined by the ARINC-424 standard: since RPAS navigation is mostly based on GNSS, path terminators based on the use of ground beacons (like VORs, ILS, etc) not usually available in RPAS will be restricted in practice.

Table 4 summarizes the EPTs that will be used in this approach. It includes 18 EPT codings: 11 out of the 14 standard path terminators defined for RNAV operations [5], plus 7 extended procedures for RPAS. As this table shows, each EPT requires a number of *definition parameters*. For example, the *course to altitude* (CA) needs the following parameters: the reference course, the altitude limit, and optionally the speed limit and the required vertical path angle.

Note that EPTs can lead the plane to a well defined waypoint or to a variable waypoint. The termination point is a waypoint in the case that the EPT specifies a fix as the terminator. If the EPT specifies an altitude or some other condition, then the termination point is a variable waypoint since different planes can reach that altitude at different points depending on their performance.

5.4.1. UML model

The UML model of legs shown in Fig. 7 is derived from the previous discussion. There is an abstract class *Leg*, and the different EPTs are subclasses or specializations of this class. Each EPT has its own set of attributes, which basically corresponds to the parameters of Table 4.

Table 4: Extended Path Terminators and definition parameters.

Description	coding	waypoint1	waypoint2	fly-over	course	path length	altitude1	altitude2	speed limit	radius	turn direction	limit value	vertical angle
Course to altitude	CA				✓				O			1	O
Course to fix	CF	✓		O	✓		O	O	O				O
Direct to fix	DF	✓		O			O	O	O				
Fix to an altitude	FA	✓			✓				O			1	O
Fix to manual	FM	✓			✓		O	O	O				
Initial fix	IF	✓					O	O	O				
Radius to fix	RF	✓	2			✓	O	O	O		✓		O
Track to fix	TF	✓		O		✓	O	O	O				O
Heading to altitude	VA				3				O			1	O
Heading to intercept	VI				3		O	O	O				
Heading to manual	VM				3		O	O	O				
Radius to altitude	RA	2							O	✓	✓	1	O
Radius to lap number	RL	2					O	O	O	✓	✓	4	
Radius to manual	RM	2					O	O	O	✓	✓		
Radius to time	RT	2					O	O	O	✓	✓	5	
Scan to lap number	SL	✓	✓		6		O	O	O	✓		4	
Scan to manual	SM	✓	✓		6		O	O	O	✓			
Scan to time	ST	✓	✓		6		O	O	O	✓		5	

✓– Required

O – Optional

1 – Altitude limit (at or above)

2 – Arc center

3 – Heading not course

4 – Laps limit

5 – Time limit

6 – Initial course

Shaded – Not applicable field

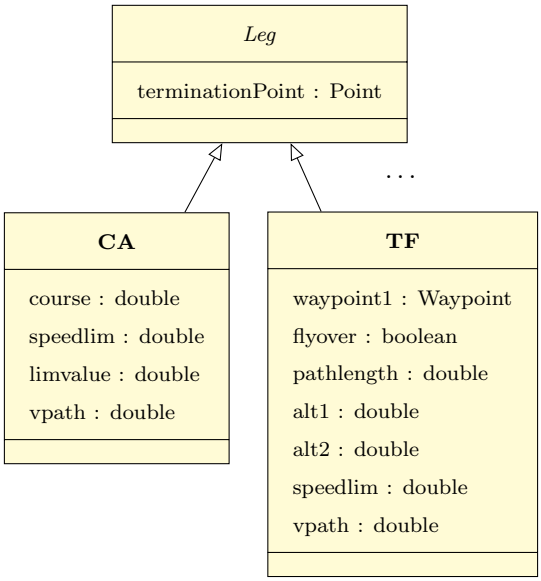


Figure 7: Structure diagram of two of the possible specializations of a Leg object.

In addition, the abstract class *Leg* has an attribute common to all EPTs, which is the *terminationPoint*. When the termination condition of the EPT is a fix (IF, RF, etc), the termination point is the *Waypoint* (fix) specified as the terminator of the EPT (attribute *waypoint1* in Table 4). Otherwise, this attribute is a *VariableWaypoint*. Since every leg has an associated termination point, there is a bijective relationship between legs and their corresponding termination waypoints.

5.5. Segments

A *segment* is defined as a sequence of *legs* that correspond to a phase of flight or to a flight procedure. As stated in the Mission Plan definition, the Mission Plan specifies the set of all possible segments of a mission.

Every segment is tagged with a segment *type*. Six types of segments have been defined according to the phases of flight of an RPAS mission [37]: departure, en-route, operation (i.e. performing the payload task in the operations area), ingress/egress (transitions between en-route and operation), and arrival.

Table 5: Valid initial and final legs per segment type.

Segment type	Initial leg	Final leg
Departure	IF	CF, DF, RF, TF
En-route	IF	TF
Ingress	IF	CF, DF, RF, TF
Operations	IF	CF, DF, RF, TF
Egress	IF	CF, DF, RF, TF
Arrival	IF	CF, RF, TF

The specification of the sequence of legs is subject to the following constraints:

Definition 1. (Well formed segment) *A segment is said to be* well formed *iff its sequence of legs meets the following rules derived from the ARINC-424 standard*[1]*:*

WFS1 *Permitted legs per segment type.* Not all EPT types are permitted in every segment type. For example, en-route segments can be composed of legs coded as IF and TF only. Similarly, EPTs for performing scanning patterns are limited to operations segments.

WFS2 *Permitted beginning and ending leg types.* The ARINC-424 standard [6] defines a table of valid initial and final path terminators. Table 5 is an extension of the ARINC-424 proposal considering the new phases of flight proposed by EUROCONTROL for RPAS [37].

WFS3 *Permitted leg sequences.* The ARINC-424 standard [6] also defines a table of valid sequences of legs. The general rule is that RF and TF legs should be preceded by legs whose target point is a waypoint. For example, the CA/TF sequence is not allowed because the TF requires a previous fix for defining the flight path. Another prohibited sequence is DF/RF

[1] They are also the basis for the corresponding rules in ICAO Doc. 8168 OPS/611 [36].

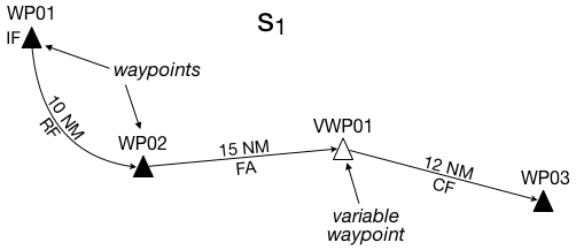
because the resulting flight path is not predictable. We have derived an equivalent table of permitted EPT sequences that has been omitted for brevity.

Regarding Table 5, it must be noted that the initial leg of every segment type is coded as IF. This initial leg must be associated with a waypoint with well defined coordinates (not a variable waypoint), except in departure segments. In such segments, the starting condition is any point along the runway, so this point must be identified using a variable waypoint. The *de facto* starting leg in departures will be the second leg which must be coded as CA, CF, VA, or VI [36]; but the IF leg is still required to support the definition of the segment, and particularly the *segment graph*, as it will be discussed below. Note also that the permitted final legs in Table 5 are always EPTs ending at a fix; this will allow segments to be connected at a well defined waypoint (not a variable waypoint), thus making the flight procedure more deterministic.

From the point of view of Graph Theory, a well formed segment can be modeled as a *directed path* [38]. A directed path (sometimes called *dipath*) is a sequence of edges which connect a sequence of nodes, but with the added restriction that the edges all be directed in the same direction. In our case, the graph nodes represent the *termination points* of the legs of the segment (either a well defined or a variable waypoint), and directed edges are the path between these points. Figure 8 shows an example of a segment graph as compared with the segment representation in a navigation chart.

Segment nodes will be named $w_{i,j}$, where i is the segment identifier, and j is the identifier of the segment leg (or more precisely its associated termination point). Notation $w_{1,1} \rightarrow w_{1,2}$ expresses that there is a directed edge from node $w_{1,1}$ to node $w_{1,2}$. This means that waypoint $w_{1,2}$ is *reached* (navigated) right after $w_{1,1}$ in segment 1.

By definition of directed path, the set of nodes of a segment are are totally ordered. As a result, a segment node can be reached from any node that precedes it in the ordering. If the in-degree of a node is the number of ingoing edges, the



(a) Navigation chart view.

(b) Graph view.

Figure 8: Example of segment representation.

in-degree of all segment nodes is 1, except in the first node (called the source node), where it is 0. In the same way, if the out-degree of a node is the number of outgoing edges, the out-degree of all segment nodes is 1, except in the last node (called the sink node), where it is 0.

In order to optimize *route reconfigurations*, the edges of a segment graph are *weighted*, with the weight representing the cost of flying between two consecutive nodes. In this work, the cost is measured in terms of the resulting *path length*, but it could also represent flight time, fuel consumption, etc. In order to compute the path length between two consecutive nodes, it is necessary to distinguish between the following cases:

a) When both nodes are waypoints, the path is defined completely so the length can be computed using spherical geometry functions.

b) When some of the nodes are a variable waypoint, the path depends on the aircraft performance, so the length should be estimated using trajectory planning functions like EUROCONTROL's Base of Aircraft Data (BADA) libraries [39].

A special case where the cost cannot be easily estimated is an EPT with manual

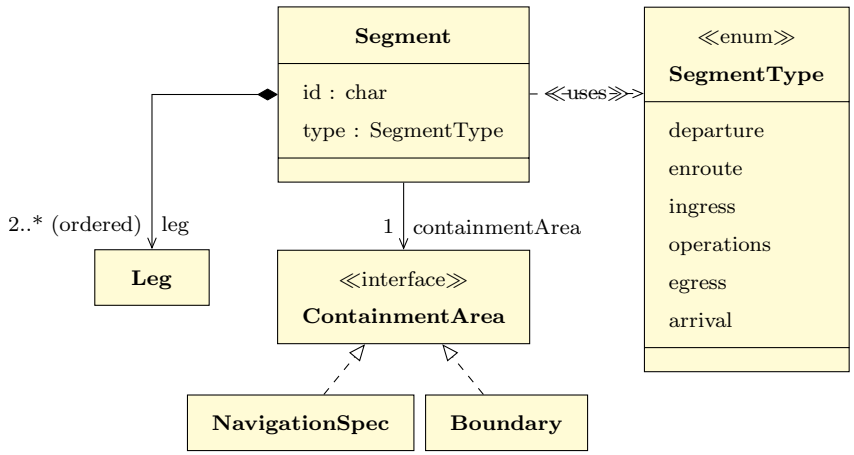


Figure 9: Structure diagram of a Segment object.

termination, since the termination point completely depends on pilot decision. A maximum weight should be assigned in this case.

5.5.1. UML model

The UML model of a *Segment* object is shown in Fig. 9. There are two basic attributes: the segment's *id*(entifier) and the segment's *type*. The attribute *leg* is the sequence of legs of the segment, where each leg is a subclass of the abstract class *Leg*.

In addition, the attribute *containmentArea* represents the protected volume that encircles the defined flight path. This attribute describes the flight tolerance error allowed in order to fly the segment. As shown in Fig. 9, the containment area is modeled as an interface that can be realized in two ways: by means of a *NavigationSpec* object (a PBN navigation specification that defines the allowed cross track error); or by defining a *Boundary* object (an airspace volume). For example, a *NavigationSpec* can be an RNP-1 specification that defines a containment area of 1 NM at each side of the intended flight path. *Boundary* objects define the containment area by its geographical limits, see Sec. 5.7.

5.6. Routes

As it was introduced at the beginning of this section, Reconfigurable Mission Plans allow to specify all the different routes that an aircraft can fly. One among all the possible routes must be declared as the *nominal route*. This route is the only one that is statically declared in the Mission Plan. The remaining alternative routes are *dynamically* derived from the segments declared in the Mission Plan. The set of all the possible routes is described by the *Mission Graph*. The Mission Graph is defined as the *union* of all the segments of the Mission Plan.

The union operator defines the conditions that allow to fly from one segment to another. The *necessary condition* for setting a path between two segments is that they shall have a waypoint in common. This is not a *sufficient condition*, though. In order for the resulting path to be consistent, it is also necessary to account for the position of the common waypoint in the sequence of legs of each segment, and for the phase of flight of the segments under consideration. Therefore, the definition of the union operator is similar to the notion of union in Graph Theory, but it should be particularized to cope with some restrictions on the path construction:

Definition 2. (Segment union) *Given the graph of two well formed segments s_a and s_b, the* union *of s_a with s_b, denoted as $s_a \cup s_b$, is the directed graph that results out of performing the following two operations:*

SU1 *Performing the disjoint union of both segment graphs.* This implies that, in the resulting graph, the nodes of s_a and the nodes of s_b will have no elements in common (even though they may reference the same waypoints).

SU2 *Creating additional edges between nodes associated with the same waypoint, iff the following conditions hold:*

SU2.1 The waypoint is not associated with the termination point of the first leg of s_a.

Table 6: Permitted phase of flight transitions.

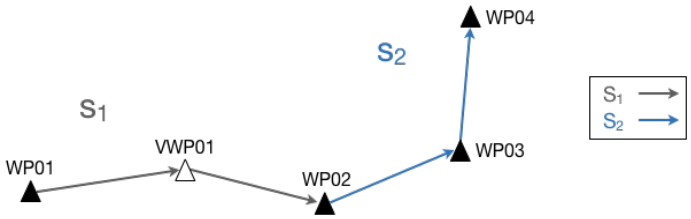
		Next segment (s_b)					
		Departure	*En-route*	*Ingress*	*Operations*	*Egress*	*Arrival*
Current seg. (s_a)	*Departure*	–	✓	✓	✓	–	✓
	En-route	–	✓	✓	✓	–	✓
	Ingress	–	–	–	✓	✓	–
	Operations	–	✓	–	✓	✓	✓
	Egress	–	✓	✓	✓	–	✓
	Arrival	–	–	–	–	–	–

SU2.2 The waypoint is not associated with the termination point of the last leg of s_b.

SU2.3 The phase of flight transition from s_a to s_b is permitted by Table 6.

The new edges created when condition **SU2** holds will be called *transition edges* because they allow to fly between segments. An important remark is that, since the nodes connected by a transition edge are geographically co-located, the cost of flying a transition edge will be zero.

To illustrate this definition, imagine the two segments s_1 and s_2 in Fig. 10a; assume that both of them correspond to the en-route phase. The graph obtained after performing the disjoint union of these segments (operation **SU1**) is shown in Fig. 10b. As it can be observed, the resulting graph is not connected: there are two subgraphs, each one representing the graph of a segment, but they have no elements in common. In order to connect these graphs, it is necessary to perform operation **SU2**. The resulting graph is shown in Fig. 10c. As it can be observed, there exists an additional edge from node $w_{1,3}$ to node $w_{2,1}$ (depicted as a dashed line) since both nodes are associated with the same waypoint (WP02) and conditions **SU2.1**, **SU2.2** and **SU2.3** hold. This way, it is possible to fly from s_1 to s_2 through this transition edge.



(a) Navigation chart view.

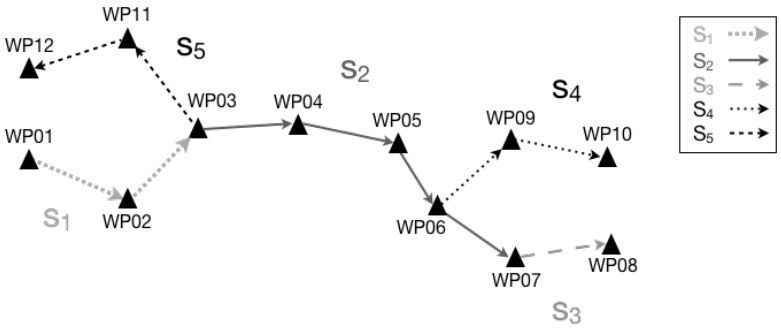
(b) Disjoint union of segments s_1 and s_2 (operation **SU1**).

(c) Union of segments s_1 and s_2 (operations **SU1** and **SU2**).

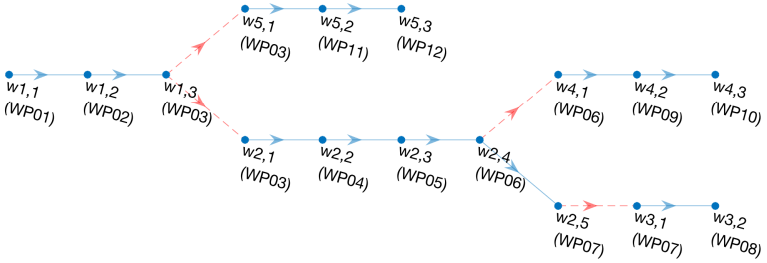
Figure 10: Example of segment union. Leg codings, path distances, and edge weights are omitted for simplicity.

An important property of the segment union is that the union operator is not commutative: $s_a \cup s_b \neq s_b \cup s_a$. In the above example, the union of s_2 with s_1 ($s_2 \cup s_1$) does not allow to fly from s_2 to s_1 through the common waypoint WP02 because conditions **SU2.1** and **SU2.2** fail in this case. Therefore, in order to obtain the Mission Graph, it is necessary to perform the segment union of each segment declared in the Mission Plan with all the remaining segments.

Another important property of the segment union is that, although the graph of a segment is totally ordered, the graph that results out of performing the segment union might be partially ordered. This might occur if the resulting graph has multiple source nodes and/or multiple sink nodes; or if it presents



(a) Navigation chart view.



(b) Graph view.

Figure 11: Example of Mission Graph.

cycles. To illustrate this, imagine a more complex example like the one in Fig. 11a. In this case, there are 5 segments which connect 12 waypoints. The Mission Graph that results out of performing the union of all these segments is presented in Fig. 11b. As it can be observed, it presents one source ($w_{1,1}$) but three sinks ($w_{3,2}$, $w_{4,3}$, and $w_{5,3}$), so it is not possible to define a total order.

Once the Mission Graph has been defined, we can define mission *routes* based on the concept of *reachability* between nodes of the Mission Graph:

Definition 3. (Node reachability) *Let $w_{a,i}$ and $w_{b,j}$ be two nodes of the Mission Graph. Node $w_{b,j}$ is said to be* reachable *from node $w_{a,i}$, denoted as $w_{a,i} \Rightarrow w_{b,j}$, iff there is* at least *one directed path from $w_{a,i}$ to $w_{b,j}$ in the Mission Graph.*

Accordingly, a route between two nodes $w_{a,i}$ and $w_{b,j}$ of the Mission Graph exists iff $w_{a,i} \Rightarrow w_{b,j}$. The resulting route is a directed path in the Mission Graph that can traverse the nodes of different segments. The set of all the nodes traversed by a route is totally ordered. The node in which the directed path reaches a segment is called the *entry point* of that segment in this route. The node in which the directed path leaves a segment is called the *exit point* of that segment in this route.

As a result, a route can be specified as the sequence of the segments traversed by a directed path in the Mission Graph, along with the entry point and the exit point of each segment in the sequence. This will be denoted as:

$$r = \langle \text{WP0} \xrightarrow{s_0} \text{WP1} \xrightarrow{s_1} \text{WP2} \ldots \rangle \tag{1}$$

Where WP0 and WP1 are the entry and exit points of s_0, and WP1 and WP2 are the entry and exit points, respectively, of s_1. For example, a possible route in the Mission Graph example of Fig. 11 is:

$$r_0 = \langle \text{WP01} \xrightarrow{s_1} \text{WP03} \xrightarrow{s_2} \text{WP06} \xrightarrow{s_4} \text{WP10} \rangle \tag{2}$$

Finally, a route is considered to be *effective* for achieving a specific mission goal if it traverses all the associated locations of that goal. For example, if we assume that a mission goal for the mission example in Fig. 11 is declared as "fly over waypoint WP05; then perform the landing procedure associated to waypoint WP10", then r_0 in Eq. (2) will be effective in achieving this because it traverses both waypoints in the required order.

5.6.1. UML model

The UML model of a *Route* object is shown in Fig. 12. The attribute *goal* specifies the mission goal for which the route is intended. The attribute *routeSegment* specifies the flight path of the route as a sequence of *RouteSegment* objects. Each *RouteSegment* is associated with a *Segment* instance of the Mission Plan. In addition, the *RouteSegment* also specifies the *entryPoint* and the

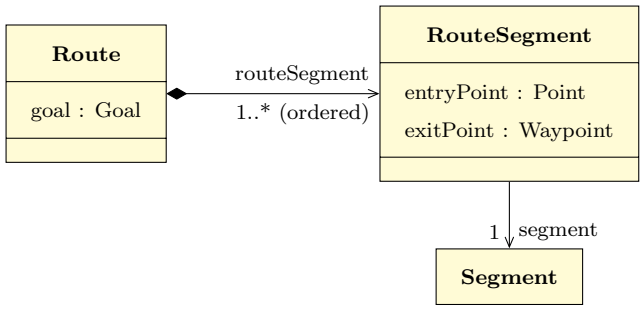


Figure 12: Structure diagram of a Route object.

exitPoint of this segment. These attributes are specified using a *Point* object. Note that all *entryPoints* and *exitPoints* are instances of class *Waypoint*, except the entry point of a departure segment, which is an instance of class *Variable-Waypoint*.

The UML's direct association between the *RouteSegment* and the *Segment* object has an important implication on the route definition: as depicted in Fig. 4, the ownership of the *Segment* objects of a route is the *Reconfigurable-MissionPlan* class, not the *Route* class itself. In other words: routes are built using segments defined by the Mission Plan. For this reason, a segment used in the nominal route can also be used in another route.

5.7. Mission boundaries

Apart from the route information, the Mission Plan allows mission boundaries to be specified. A mission boundary is an airspace volume with well-specified limits which are monitored so as to produce a contingency if the limits are close to being trespassed. There are two main *types* of boundaries: no-fly zones, and geofenced areas. In no-fly zones, the contingency is produced when the aircraft gets close to entering this area. In geofenced areas, the contingency is produced when the aircraft gets close to exiting this area. From the ATC point of view, a no-fly zone is an area subject to special authorization, while a geofenced area can be a segregated area used as the operations area of an RPAS, for example. Both of them can be published as NOTAMs.

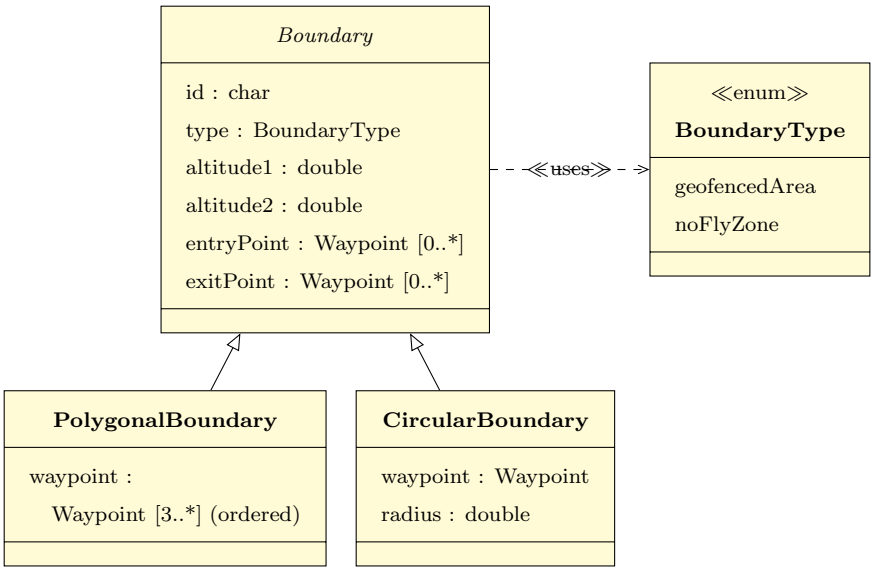


Figure 13: Structure diagram of a Boundary object.

5.7.1. UML model

The UML model of a *Boundary* object is shown in Fig. 13. According to this figure, a *Boundary* class has two possible specializations or subclasses: circular boundaries and polygonal boundaries. The specification of a circular boundary requires a waypoint object be provided and the corresponding radius defined; while a polygonal boundary is formed from three or more waypoints in some given order.

Boundary objects may also have *vertical limits*: an upper limit, a lower limit, or a given altitude window. Attribute *type* is used to indicate whether it is a no-fly zone or a geofenced area. Finally, *Boundary* objects may be associated with a list of *entry* and *exit points*. These points are waypoints that allow an ingress/egress segment to be connected with an operation segment that is geofenced.

6. Dynamic route configuration

As introduced in Sec. 5, Reconfigurable Mission Plans provide two types of routes: static and dynamic routes. Static routes are statically declared in the Mission Plan pre-flight; by contrast, dynamic routes are not statically declared, but rather generated at flight time as a function of the Mission Graph, the current position, and the active mission goal. The problem in dynamically (re)configuring a Mission Plan route can be stated as the problem of *finding the lowest cost route that is effective for achieving the active mission goal from the current position of the RPAS*.

As an example, let us return the mission example in Fig. 11. Assume that the nominal goal is declared as "fly over waypoint WP05; then land at WP10", and that the nominal route is r_0 in Eq. (2). There are also two alternate goals for performing the "flight termination" at waypoints WP08 and WP12, for instance. Now, once in flight time, the RPAS is flying the nominal route, somewhere in between waypoints WP02 and WP03, and a contingency occurs. If the decision maker agent sets the new goal type to "flight termination" (for instance), then the Mission Management System can reconfigure the plan to fly any of the following routes:

$$r_1 = \langle \text{WP02} \xrightarrow{s_1} \text{WP03} \xrightarrow{s_5} \text{WP12} \rangle$$
$$r_2 = \langle \text{WP02} \xrightarrow{s_1} \text{WP03} \xrightarrow{s_2} \text{WP07} \xrightarrow{s_3} \text{WP08} \rangle$$

If the cost of flying a route r is $C(r)$, and we assume that $C(r_1) < C(r_2)$, then the optimal (shortest) route would be r_1. However, if the contingency had occurred after reaching WP03, then waypoint WP12 would have not been reachable, so the only suitable route in the Mission Graph would have been r_2.

In this section, we develop a series of tools for dynamically configuring a Mission Plan route. These tools rely on Graph Theory to exploit features like graph analysis and shortest path algorithms [38]. In order to develop these tools, we have first prototyped the UML models of a Reconfigurable Mission Plan object using object-oriented programming (OOP) in Matlab. Then, based

on this model, it is possible to perform the following actions: *1)* construct the Mission Graph object; *2)* find a path in the Mission Graph that is effective for achieving some given type of goal; and *3)* specify a path in the graph as a *Route* object. The following subsections provide more detail on these topics.

6.1. Constructing the Mission Graph object

Constructing the Mission Graph object consists of performing the union of all the segments in a Reconfigurable Mission Plan object. The pseudocode of Listing 1 performs this process based on rules **SU1** and **SU2**. In short, the pseudocode that implements rule **SU1** (lines 4 to 7) reads the Mission Plan data structure `missionObj`. Then, it gets the nodes (`s`, `t`) and the edge weights (`w`) of all the segments in `missionObj` using the `getSegmentNodes` procedure. Finally, these nodes are added to the Mission Graph structure `G`. Note that this code exploits functions of the Matlab toolkit "Graph and Network Algorithms".

Then, the pseudocode for rule **SU2** (lines 9 to 16) shows a double `for` loop that iterates for every couple of segments in the `missionObj` and checks if some pair of waypoints meet conditions **SU2.1**, **SU2.2** and **SU2.3** (procedure `getTransitionEdgeNodes`). If so, a transition edge is added to the digraph `G`. Recall that the associated weight of these edges is `w=0`.

6.2. Finding a path that is effective for achieving one given mission goal type

Once the Mission Graph object `G` has been created, the next problem is finding a route in this graph that allows some given type of goal to be achieved. As deduced in Sec. 5.6, a mission route is a directed path that connects one given source with one given destination. In this case, the source represents the initial position of the RPAS, and the destination is a location that is associated with the target goal. When this goal is composed of multiple sub-goals, the route should connect the origin with the destination, with the difference that this route must traverse all the required intermediate positions. According to this, the problem of finding a route in the graph consists of two major steps in the general case: *1)* locating all the nodes of the Mission Graph that are

Listing 1: Pseudocode for constructing the Mission Graph.

```
function G = getMissionGraph(missionObj)
G=digraph(); %Initialize digraph object
%1.- Perform disjoint union of segments (SU1)
for i=1:getNumSegments(missionObj)
    [s,t,w]=getSegmentNodes(missionObj.segmentObj(i));
    G=addedge(G,s,t,w);  %Add nodes to graph object
end
%2.- Create transition edges (SU2)
for i=1:getNumSegments(missionObj)
    for j=1:getNumSegments(missionObj)
        currentSegmentObj=missionObj.routeSegment(i);
        nextSegmentObj=missionObj.routeSegment(j);
        [s,t,w]=getTransitionPointNodes(currentSegmentObj, ...
            nextSegmentObj);
        G=addedge(G,s,t,w); %Add nodes to graph object
    end
end
end
```

associated with that goal, and *2)* finding a path in the Mission Graph that connects the source node with all the required nodes.

In Graph Theory, the problem of finding a path between nodes is known as the *shortest path problem*: the problem of finding the path that minimizes the sum of its edges' weights. There exist multiple algorithms that solve this problem [38]; for the particular case of a Mission Graph (a weighted, directed graph, where all weights are positive), Dijkstra's algorithm is suggested. Based on this algorithm, we have developed a procedure for finding dynamic routes that is structured into the following three levels:

a) *Find the path for achieving one given goal type.* The highest level of this problem requires all the goals in the Mission Plan matching the target goal type be found (for example, all the "loiter" goals, or the "flight termination"

Listing 2: Pseudocode for finding the path to achieve one given goal type.

```
function [POut,dOut] = getPathToGoalsByType(G,missionObj, ...
    srcNode,trgGoalType)
goalObjArr=getGoalsByType(missionObj,trgGoalType);
for i=1:length(goalObjArr)
    [P,d]=getPathToGoal(G,srcNode,goalObjArr(i) );
    POut{i,:}=P; %Add path to output struct
    dOut=[dOut d]; %Add cost to output struct
end
[POut,dOut]=sort(POut,dOut); %Sort paths by cost
end
```

goals, etc), and then the path for achieving each of these goals is computed. The algorithm `getPathToGoalsByType` of Listing 2 performs this task for a given Mission Graph `G` and a Mission Plan data structure `missionObj`. The remaining input arguments are the source node `srcNode` and the target goal type `trgGoalType`. The algorithm returns the paths for achieving these goals (`POut`), as well as the length of these paths (their associated cost, `dOut`). These paths are computed by invoking the `getPathToGoal` procedure multiple times, as described next.

b) *Find the path for achieving one given mission goal.* This problem requires a path traversing all the associated locations of a given goal be found. The algorithm `getPathToGoal` of Listing 3 performs this task for a given input goal `goalObj`. When the goal is a single stage goal, the problem is trivial and is solved as computing the path from the source node `srcNode` to a node associated with the target location. This path is computed using the `getPathToWaypoint` procedure, which will be introduced below. In multiple stage goals, we assume that the intermediate stages are to be flown in some given order, so the problem can be solved by invoking the `getPathToWaypoint` procedure between every two consecutive stage way-

Listing 3: Pseudocode for finding the path to achieve one given goal.

```
function [POut,dOut] = getPathToGoal(G,srcNode,goalObj)
s0=srcNode; %Initialize source node
n=getNumStages(goalObj);
for i=1:length(n)
    trgWaypoint=goalObj.stage(i).enabledVariant;
    [P,d]=getPathToWaypoint(G,s0,trgWaypoint);
    if isempty(P)
        return;
    end
    POut=[POut P(1)]; %Append path to output struct
    dOut=dOut+d(1); %Update path cost
    s0=POut(end); %Update source node
end
end
```

points and appending this sub-path to the output path structure `POut`. The overall path distance (`dOut`) is then computed as the sum of the distances of each sub-path.

c) *Find the path towards one given waypoint.* The most elementary problem consists of finding *all* the nodes of the Mission Graph associated with a given waypoint, and then computing the path from a given source node to these target nodes. The algorithm `getPathToWaypoint` of Listing 4 performs this task for a given input waypoint `waypointObj`. In this algorithm, the path between nodes is computed using the Matlab procedure `shortestpath`, setting the `'Method'` attribute to `'positive'`. This method implements Dijkstra's algorithm in Matlab. The `getPathToWaypoint` procedure returns the path from the source node to all these target nodes (`POut`) and sorts these paths according to their cost.

Listing 4: Pseudocode for finding the path towards one given waypoint.

```
function [POut,dOut] = getPathToWaypoint(G,srcNode,waypointObj)
trgNodeArr=getNodesInWaypoint(G,waypointObj);
for i=1:length(trgNodeArr)
    [P,d]=shortestpath(G,srcNode,trgNodeArr(i),'Method','positive')
    POut{i,:}=P; %Add path to output struct
    dOut=[dOut d]; %Add cost to output struct
end
[POut,dOut]=sort(POut,dOut); %Sort paths by cost
end
```

6.3. Specifying a path in the graph as a Route *object*

The last step of the dynamic route configuration process is converting one path described as a sequence of nodes of the Mission Graph to a *Route* object like in Fig. 12. This step requires all the segments traversed by the path be identified, as well as the points in which the path traverses from one segment to another segment (i.e. the entry points and the exit points of each segment, see Sec. 5.6). The advantage of using the disjoint union in the graph creation is that this task is straightforward.

7. A Reconfigurable Mission Plan design and specification example

In order to illustrate the process for both designing Reconfigurable Mission Plans and validating the algorithms for dynamic route construction, the following example is proposed. An RPAS mission seeks to perform some direct observations over the Albufera natural park in Spain. The operations area is defined as the boundary of this natural park, which coincides with the protected area F15B of the Spanish Aeronautical Information Service (AIS)[2]. This area is located within the Controlled Traffic Region (CTR) of the Valencia Airport

[2]Available online at `https://ais.enaire.es/aip/` (accessed on June 2018).

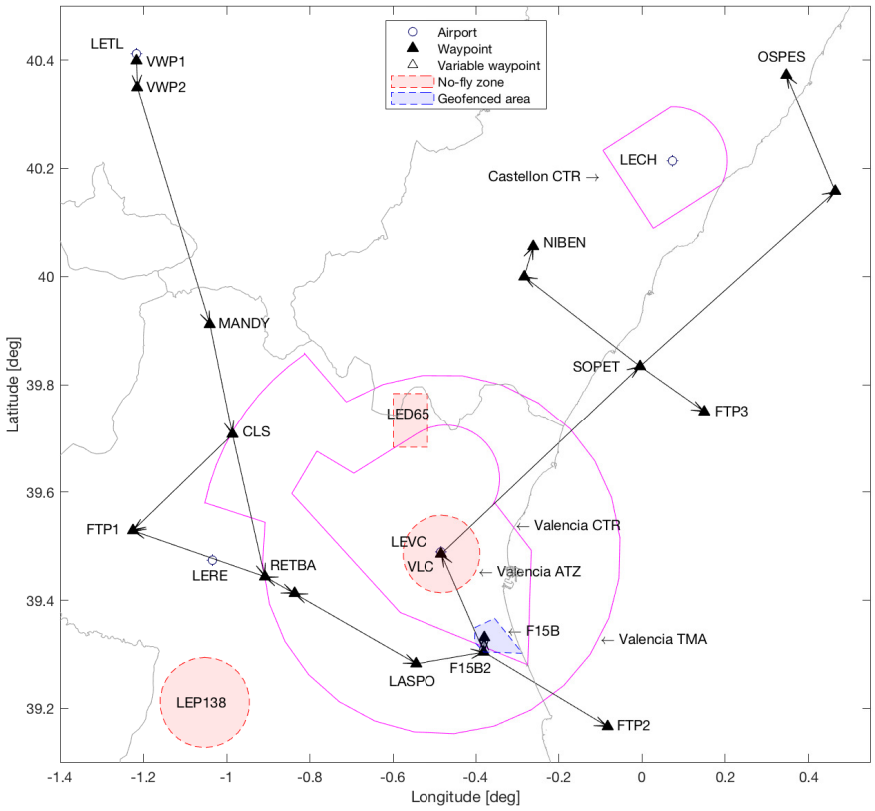


Figure 14: Reconfigurable Mission Plan example: navigation chart view. Note that some departure and arrival segments have been omitted for clarity.

(ICAO code LEVC), so the mission will require special permission from ATS authorities. The planned mission departs at the uncontrolled airport Teruel (LETL) and the expected arrival site is Castellón (LECH), a controlled one. Alternative landing sites are LETL and the Requena aerodrome (LERE). There are three no-fly zones in this mission: the nuclear plant LEP138, the military zone LED65, and the Aerodrome Traffic Zone (ATZ) around LEVC. The overall picture is presented in Fig. 14. The proposed Mission Plan will be designed in compliance with the flight charts and airspace information available in the AIS.

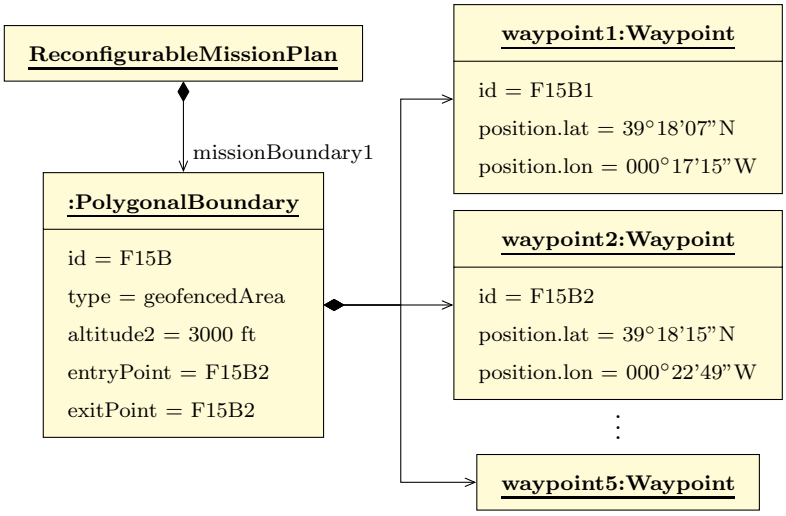


Figure 15: Reconfigurable Mission Plan example: specification of the operations area.

7.1. Mission Plan specification

Reconfigurable Mission Plans are formally specified as UML object diagrams and, as previously said, we have not addressed the subject of designing a specific language or syntax to formally specify Mission Plans other than UML. Recall that the high-level view of the Reconfigurable Mission Plan object diagram follows the structure described in Fig. 4: a declaration of the waypoints used by other components of the Mission Plan; a declaration of the possible goals (and degraded goals) of the mission; a declaration of the segments used to build the routes of a mission; a declaration of the nominal route; and a declaration of the mission boundaries. Next, we describe the specification of the different objects that conform the Reconfigurable Mission Plan example in UML.

To start with, Fig. 15 shows the specification of the *Boundary* object describing the operations area. As shown, the operations area is geofenced and has a polygonal shape outlined by five waypoints. The boundary has vertical limits to avoid conflicts with upper traffic, and the entry and the exit point of this area is the same waypoint F15B2. The other *Boundary* objects for the no-fly zones are specified in a similar manner, so they are omitted for brevity.

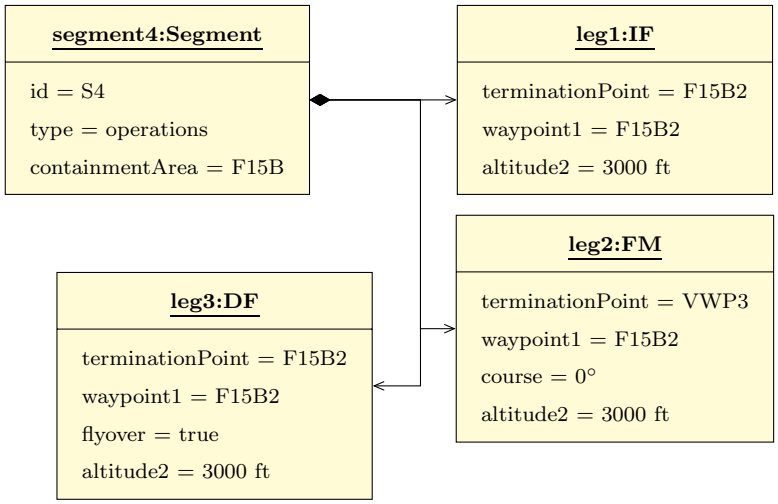


Figure 16: Reconfigurable Mission Plan example: specification of the operations segment.

The segments declared in the Mission Plan include segments performed in controlled and uncontrolled areas. Based on the AIS charts, the nominal route will be composed of the following seven segments, represented in Fig. 14:

s_1: Departure segment s_1 describes the departure phase from LETL runway 18 to MANDY, the first waypoint of the en-route phase. The flight path is constructed as a sequence of three legs, coded as IF, CA, and DF (where the IF simply supports the graph definition).

s_2: En-route segment s_2 traverses the lower ATS route R29 from MANDY to RETBA, and then the RNAV airway M871 from RETBA to LASPO. All legs are coded as IF and TF, as required by rule **WFS1**.

s_3: Ingress segment s_3 connects LASPO with waypoint F15B2, the entry point of the operations area.

s_4: Operations segment s_4 is linked to the operations area F15B. The flight path of s_4 is specified using three legs coded as IF, FM, and DF, see Fig. 16. One of these legs has a manual termination condition as the mission task to be performed manually by the remote pilot. Afterwards, a DF directs the aircraft towards the exit point of this area.

s_5: Egress segment s_5 connects the exit point with waypoint VLC, traversing

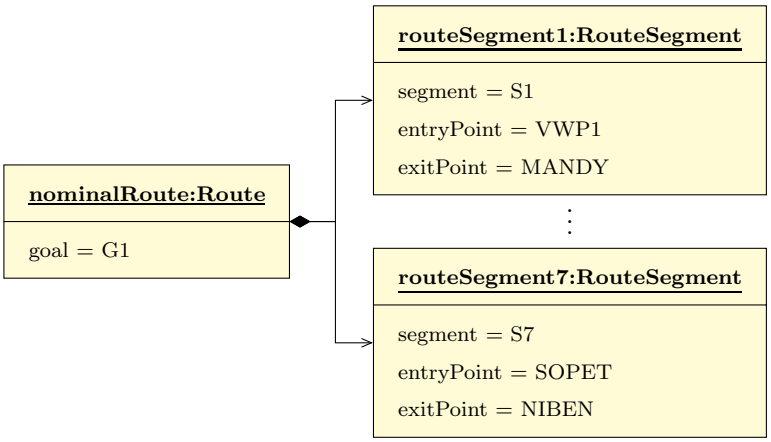


Figure 17: Reconfigurable Mission Plan example: specification of the nominal route.

the VFR corridor of the Valencia CTR. The segment legs have well specified vertical limits to ensure that the boundaries of the Valencia ATZ are not infringed.

s_6: En-route segment s_6 flies airway B26 from VLC to SOPET.

s_7: Arrival segment s_7 describes the standard arrival procedure SOPET1S from SOPET to waypoint NIBEN, the IAF for LECH runway 06.

The resulting nominal route is specified as $r_0 = \langle \text{VWP1} \xrightarrow{s_1} \text{MANDY} \xrightarrow{s_2} \text{LASPO} \xrightarrow{s_3} \text{F15B2} \xrightarrow{s_4} \text{F15B2} \xrightarrow{s_5} \text{VLC} \xrightarrow{s_6} \text{SOPET} \xrightarrow{s_7} \text{NIBEN} \rangle$, where "VWP" denotes a variable waypoint. The UML diagram of this route is schematized in Fig. 17 as a composition of 7 *routeSegment* objects that are associated with the previous segments. The goal in this figure will be detailed below.

Along with the previous segments, the proposed Mission Plan also declares additional segments for alternative routings. For example, there is an alternate departure segment s_8 for departing in the opposite runway direction of LETL; however, this segment is not used in the assumed airport configuration. There is also an arrival segment s_9 towards OSPES, the IAF for the alternate airport configuration of LECH, as well as other arrival segments towards the alternative landing sites ($s_{10} \ldots s_{13}$). With respect to alternate en-route segments, s_{15} connects the operations area with the emergency landing site LERE. This segment

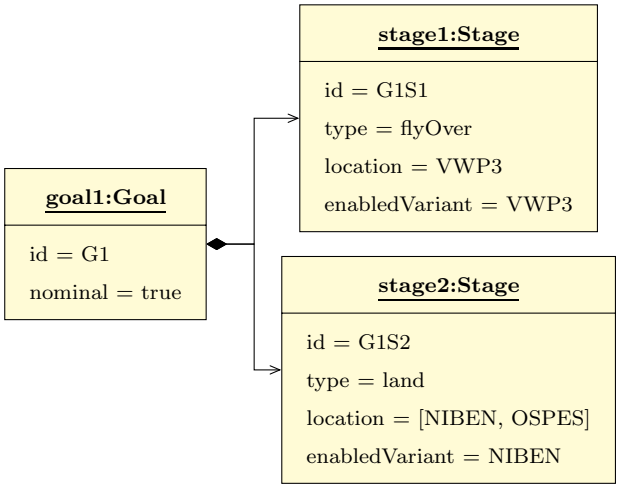


Figure 18: Reconfigurable Mission Plan example: specification of the nominal goal.

goes below airway M871 following a dedicated flight corridor because M871 is a single direction airway. The remaining segments allow the nominal route to be connected with the proposed flight termination points (FTPs) ($s_{16} \ldots s_{19}$).

Based on the previous discussion, the nominal goal can be stated as: "to perform the manual task in the operations area and then land at LECH". Accordingly, the nominal goal is specified as a sequence of the two stages represented in Fig. 18. The first stage is a "fly-over" stage to fly over the termination point of the FM leg in s_4. This way, the first stage will be considered to be completed when the remote pilot ends the manual control. The second stage is a "land" stage in which the associated location is the set of IAFs of LECH (waypoints NIBEN and OSPES), where the current enabled variant is assumed to be NIBEN.

The remaining alternate goals are all single staged. In particular, we have specified: *a)* five "loiter" goals associated to waypoints MANDY, CLS, RETBA, F15B2, and SOPET; *b)* three "land" goals for performing the landing at LETL (associated locations LETL18IAF and LETL36IAF), at LERE (LERE12IAF and LERE30IAF), and at LECH (NIBEN and OSPES); and *c)* three "flight termination" goals for performing the flight termination action at waypoints

FTP1, FTP2, and FTP3. Note that, in this example, the "regain signal" goal has been omitted for simplicity.

In summary, the specification of this Reconfigurable Mission Plan example required 32 waypoints, 5 variable waypoints, 19 segments, 12 mission goals, 4 mission boundaries, and 1 static route.

7.2. Mission analysis

Based on the previous specification, the Mission Graph can be automatically constructed using the algorithm in Listing 1. The resulting graph is depicted in Fig. 19. The correspondence between the node names of this figure and their associated waypoint is presented in Table 7. As can be observed in the figure, the Mission Graph is a connected, directed graph with cycles. It has 2 sources (one per runway direction at the departure site), and 10 sinks (each one associated with a different goal). The nominal route, shown as a solid line, has an estimated length of 149 NM plus the manual segment. Note that the weight of the edges between variable waypoints has been estimated using a simplified point-mass model; and the weight of the manual termination leg has been set to 50 NM, an arbitrary maximum weight.

It is possible to use the graph analysis tools in Sec. 6.2 to compute some relevant safety metrics of the Mission Graph. For example, function `getPathToGoalsByType` can be used to check what goal types are achievable from each waypoint in the mission and what the required route length is in each case. Based on these results, we can also compute the average distance to achieve a given goal type; or determine what is the farthest waypoint from a landing site, the farthest waypoint from a flight termination point, etc. Table 8 summarizes these safety metrics for the particular mission under study.

As can be observed, the "loiter" goal can be achieved at 64.0% of the nodes in the Mission Graph[3]; the average route distance for achieving this goal is 7.3

[3] Excluding the nodes relative to the operations segment, where these metrics are not significant because they carry the weight penalty of the legs with manual termination

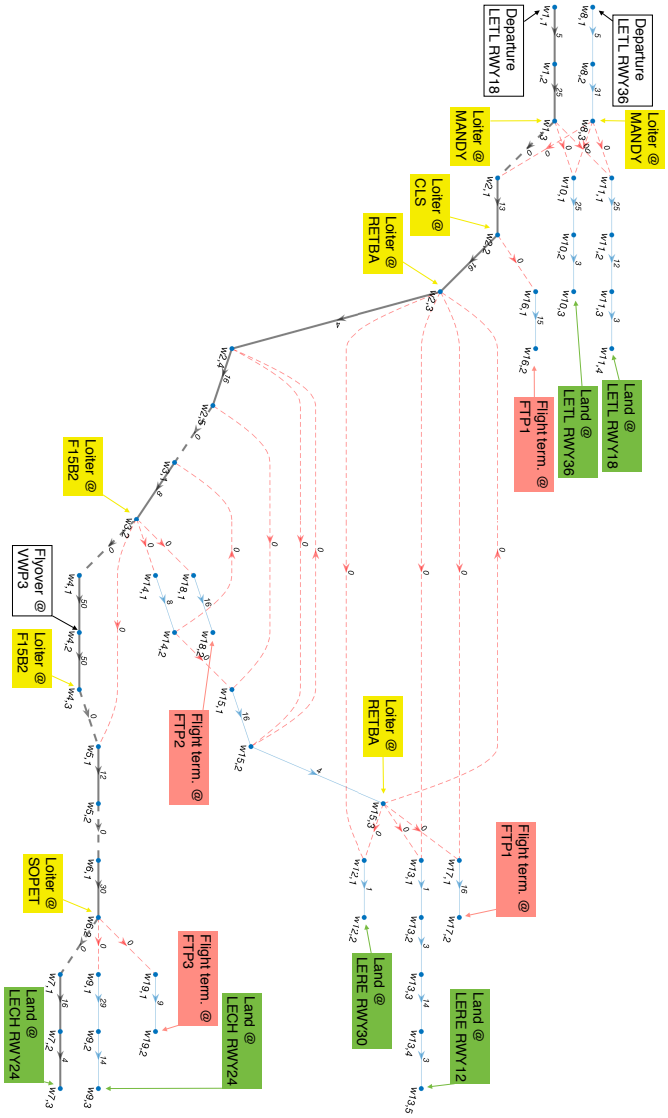


Figure 19: Reconfigurable Mission Plan example: nominal route in the Mission Graph.

Table 7: Correspondence between nodes and their associated waypoints.

Node	Waypoint	Node	Waypoint	Node	Waypoint	Node	Waypoint
$w_{1,1}$	VWP1	$w_{5,2}$	VLC	$w_{10,3}$	LETL36IAF	$w_{15,1}$	LASPO
$w_{1,2}$	VWP2	$w_{6,1}$	VLC	$w_{11,1}$	MANDY	$w_{15,2}$	MOPIR
$w_{1,3}$	MANDY	$w_{6,2}$	SOPET	$w_{11,2}$	LETLAUX1	$w_{15,3}$	RETBA
$w_{2,1}$	MANDY	$w_{7,1}$	SOPET	$w_{11,3}$	LETLAUX2	$w_{16,1}$	CLS
$w_{2,2}$	CLS	$w_{7,2}$	TATOS	$w_{11,4}$	LETL18IAF	$w_{16,2}$	FTP1
$w_{2,3}$	RETBA	$w_{7,3}$	NIBEN	$w_{12,1}$	RETBA	$w_{17,1}$	RETBA
$w_{2,4}$	MOPIR	$w_{8,1}$	VWP4	$w_{12,2}$	LERE30IAF	$w_{17,2}$	FTP1
$w_{2,5}$	LASPO	$w_{8,2}$	VWP5	$w_{13,1}$	RETBA	$w_{18,1}$	F15B2
$w_{3,1}$	LASPO	$w_{8,3}$	MANDY	$w_{13,2}$	LERE30IAF	$w_{18,2}$	FTP2
$w_{3,2}$	F15B2	$w_{9,1}$	SOPET	$w_{13,3}$	LEREAUX1	$w_{19,1}$	SOPET
$w_{4,1}$	F15B2	$w_{9,2}$	LECHAUX1	$w_{13,4}$	LEREAUX2	$w_{19,2}$	FTP3
$w_{4,2}$	VWP3	$w_{9,3}$	OSPES	$w_{13,5}$	LERE12IAF		
$w_{4,3}$	F15B2	$w_{10,1}$	MANDY	$w_{14,1}$	F15B2		
$w_{5,1}$	F15B2	$w_{10,2}$	LETLAUX1	$w_{14,2}$	LASPO		

NM ($2\sigma = 23.6$ NM); and the farthest node from a loiter point is VWP4, which is 36.0 NM away from the loiter point associated with MANDY. Similar results can be extracted from the remaining goals under analysis. Note that these metrics are not applicable to the "fly over" goal because it is used to specify intermediate route constraints, not for safety reasons.

Finally, these metrics can also be used to refine some aspect of a particular mission design. For example, if the average route distance for achieving the "flight termination" goal is considered to be above some given safety threshold, then an additional flight termination point can be proposed. However, such conclusions go beyond the scope of this example.

7.3. In-flight contingency management

The last part of this example aims at illustrating the advantages of the Reconfigurable Mission Plan concept with respect to automated contingency management. Imagine an RPAS is flying the nominal route of the proposed

Table 8: Safety metrics of the proposed Reconfigurable Mission Plan example.

Goal type	Nodes where goal type is achievable (%)	Average route distance (NM)	Worst case route distance (NM)
Fly-over	N/A	N/A	N/A
Loiter	64.0	7.3 ± 23.6 (95%)	36.0 (VWP4)
Regain signal	N/A	N/A	N/A
Land	66.0	23.8 ± 42.3 (95%)	66.0 (VWP4)
Flight termination	60.0	25.2 ± 35.4 (95%)	64.0 (VWP4)

N/A – Not applicable

mission, that the active mission goal is the nominal goal, and that the RPAS is about to enter the operations area somewhere in between waypoints LASPO and F15B2, see Fig. 20. If a contingency occurs in this scenario and the mission is not replanned, the remaining route length would be 70 NM plus the manual segment in the operations area (see blue dotted line in Fig. 20). In order to reduce the risk caused by the contingency, the mission can be replanned as follows.

a) If the C2 link is up, the Contingency Manager informs the remote pilot of a possible type of mission goal that could mitigate the effect of the contingency (for example, a "land" goal). This suggestion is made in accordance with the state machine of the Contingency Plan, which is hardcoded into the embedded software (Sec. 3.1). However, the decision maker agent is still the remote pilot, so he or she can select the new mission goal type at will. If we assume that the selected mission goal type is indeed "land", then the Mission Manager computes all the possible routes in the Mission Graph that are effective at achieving this. This task is performed using the `getPathToGoalsByType` procedure described in Sec. 6. For the particular scenario under study, the procedure returns the following routes, along with

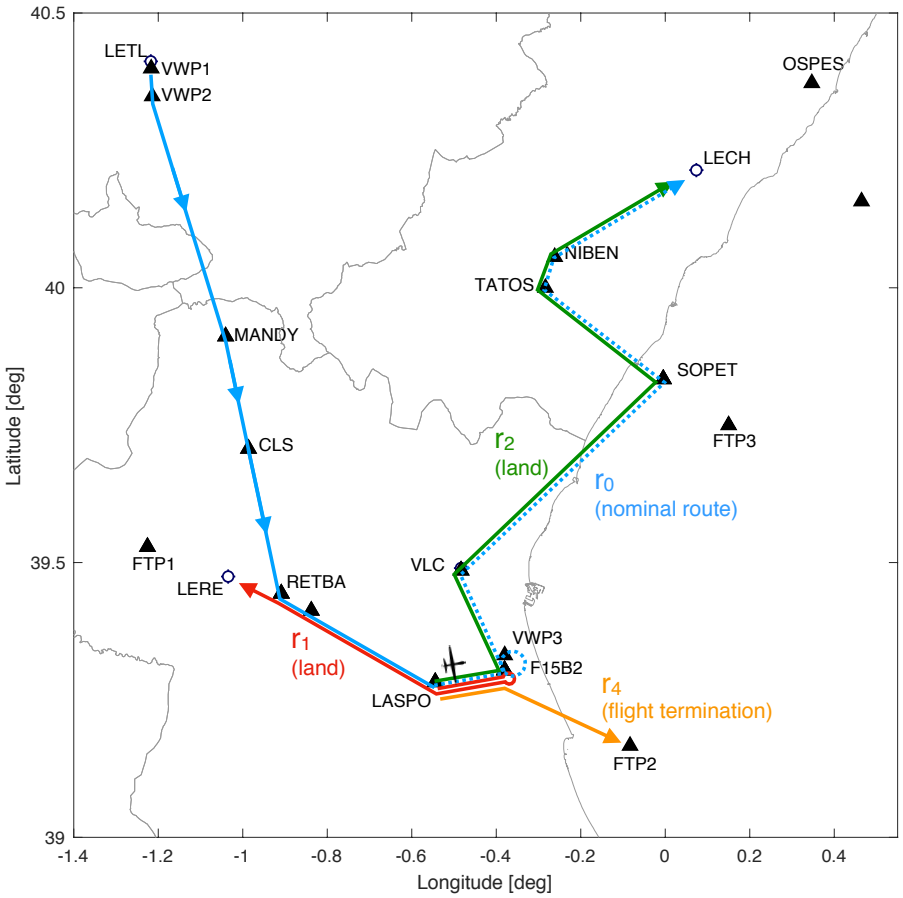


Figure 20: Reconfigurable Mission Plan example: dynamic route configuration options in a contingency scenario.

their associated route distance (cost):

$$r_1 = \langle \text{LASPO} \xrightarrow{s_3} \text{F15B2} \xrightarrow{s_{14}} \text{LASPO} \xrightarrow{s_{15}} \text{RETBA} \xrightarrow{s_{12}} \text{LERE30IAF} \rangle;$$

$$r_2 = \langle \text{LASPO} \xrightarrow{s_3} \text{F15B2} \xrightarrow{s_5} \text{VLC} \xrightarrow{s_6} \text{SOPET} \xrightarrow{s_7} \text{NIBEN} \rangle$$

The first route r_1 (the red path in Fig. 20) has an associated cost of 37 NM and is effective for landing at the alternate landing site LERE. The second route r_2 (green path) has a cost of 70 NM and is effective for landing at the main destination site LECH. Note that, although r_2 steers towards the nominal destination, this route is not effective for achieving the nominal goal

since it does not fly over VWP3 (the first stage in Fig. 18). Also note that, in this case, the `getPathToGoalsByType` procedure is unable to find a route towards the other alternative landing site LETL because this point is not reachable from the current position of the RPAS.

Then, the remote pilot must also select one of the previous dynamic routes for achieving the new mission goal type (probably the shortest one).

b) Should the C2 link be lost, the Contingency Plan can select a different (more conservative) contingency option than when the C2 link is up because the remote pilot is now unable to intervene in the operation. As a result, the mission goal type determined by the Contingency Manager (the current decision maker) is provided directly to the Mission Manager and the shortest route for achieving this goal is automatically selected.

Finally, in the case where the selected goal type had been "loiter" or "flight termination", the corresponding routes returned by the `getPathToGoalsByType` procedure would have been $r_3 = \langle \text{LASPO} \xrightarrow{s_3} \text{F15B2} \rangle$, and $r_4 = \langle \text{LASPO} \xrightarrow{s_3} \text{F15B2} \xrightarrow{s_{18}} \text{FTP2} \rangle$ (where r_4 is the orange path in Fig. 20). Their associated cost is 8 NM, and 24 NM, respectively. Thus, any of the possible alternative routes is shorter than the nominal route in this contingency scenario. Consequently, it is possible to affirm that the proposed Reconfigurable Mission Plan design allows the flight time of the RPAS experiencing a contingency to be reduced, which is the major guideline of the ICAO contingency management policy [1].

8. Conclusions

A novel Mission Plan specification that overcomes the limitations of the conventional flight plans for describing RPAS missions has been introduced in this paper. The proposed concept of Reconfigurable Mission Plan is based on the idea that the RPAS has a nominal route, but this route can be changed at flight time due to operational conditions or contingencies. The novelty is that Reconfigurable Mission Plan allows detailed specification of all the possible routes the aircraft can fly. This improves predictability and increases the level

of automation by enabling automatic reconfiguration of the intended plan when a contingency or some other event occurs.

In this work, the Reconfigurable Mission Plan specification has been formalized using UML models. A series of algorithms for dynamically configuring a Mission Plan route that is expected to handle the contingency state being faced have been also presented in this paper. Lastly, the resulting specification and the algorithms for handling Reconfigurable Mission Plans have been prototyped and validated in a representative operational scenario.

Results show that the proposed specification is able to describe the specificities of an RPAS mission. Results also demonstrate that the proposed contingency management scheme is a good mechanism for reducing the flight time of an RPAS experiencing a contingency, a key aspect for maintaining an adequate level of safety. Future work is to integrate the algorithms for flying Reconfigurable Mission Plans in the proposed Mission Management System architecture and to test the complete system working in a simulation environment.

Conflict of interest statement

There is no conflict of interest.

Acknowledgments

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

References

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

[2] International Civil Aviation Organization, Doc. 4444, ATM/501: Procedures for Air Navigation Servicies: Air Traffic Management, 16th ed., ICAO, Montréal, Canada, 2016.

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

[4] J. Rumbaugh, I. Jacobson, G. Booch, The Unified Modeling Language. Reference Manual, Adisson Wesley Longman Inc., 1999.

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

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

[7] H. Chao, Y. Cao, Y. Chen, Autopilots for small fixed-wing unmanned air vehicles: A survey, in: International Conference on Mechatronics and Automation, IEEE, Harbin, China, 2007, pp. 3144–3149. doi:10.1109/ICMA.2007.4304064.

[8] M. Barbier, E. Chanthery, Autonomous mission management for unmanned aerial vehicles, Aerospace Science and Technology 8 (2004). doi:10.1016/j.ast.2004.01.003.

[9] F. Adolf, M. M. Carneiro, Behavior-based High Level Control of a VTOL UAV, in: AIAA Infotech @ Aerospace Conference, AIAA, Seattle, WA, USA, 2009, pp. 1–13. doi:10.2514/6.2009-1977.

[10] M. Kao, G. Weitzel, X. Zheng, M. Black, A simple approach to planning and executing complex AUV missions, in: Symposium on Autonomous Underwater Vehicle Technology, IEEE, 1992, pp. 95–102. doi:10.1109/AUV.1992.225188.

[11] E. Santamaria, C. Barrado, E. Pastor, An Event Driven Approach for Increasing UAS Mission Automation, in: AIAA Infotech @ Aerospace, Seattle, WA, USA, 2009, pp. 1–21. doi:10.2514/6.2009-2044.

[12] E. Santamaria, C. Barrado, E. Pastor, P. Royo, E. Salami, Reconfigurable automated behavior for UAS applications, Aerospace Science and Technology 23 (2012) 372–386. doi:10.1016/j.ast.2011.09.005.

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

[14] I. A. McManus, R. A. Clothier, R. A. Walker, Highly Autonomous UAV Mission Planning and Piloting for Civilian Airspace Operations, in: 11th Australian International Aerospace Congress (AIAC-11), Melbourne, Australia, 2005.

[15] J. S. Dittrich, A. Bernatz, F. Thielecke, Intelligent systems research using a small autonomous rotorcraft testbed, in: 2nd AIAA Unmanned Unlimited Conference, Workshop and Exhibit, AIAA, San Diego, CA, USA, 2003, pp. 6561–6572. doi:10.2514/6.2003-6561.

[16] 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.

[17] 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.

[18] R. Brooks, A robust layered control system for a mobile robot, IEEE Journal on Robotics and Automation 2 (1986) 14–23. doi:10.1109/JRA.1986.1087032.

[19] C. Flanagan, D. Toal, R. Strunz, Subsumption architecture for the control of robots, in: Polymodel 16: Applications of Artificial Intelligence, Sunderland, UK, 1995, pp. 150–158.

[20] 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.

[21] H. Usach, C. Torens, F. Adolf, J. Vila, Architectural Considerations Towards Automated Contingency Management for Unmanned Aircraft, in: AIAA Infotech @ Aerospace, AIAA SciTech, AIAA, Grapevine, TX, USA, 2017, pp. 1–13. doi:10.2514/6.2017-1293.

[22] H. Usach, J. Vila, C. Torens, F. Adolf, Architectural design of a Safe Mission Manager for Unmanned Aircraft Systems, Journal of Systems Architecture 90 (2018) 94–108. doi:10.1016/j.sysarc.2018.09.003.

[23] F. De Florio, Airworthiness: An Introduction to Aircraft Certification and Operations, 3rd ed., Butterworth-Heinemann, 2016.

[24] 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.

[25] 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.

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

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

[28] F. Kendoul, Survey of advances in guidance, navigation, and control of unmanned rotorcraft systems, Journal of Field Robotics 29 (2012) 315–378. doi:10.1002/rob.20414.

[29] H.-M. Huang, E. Messina, J. Albus, Autonomy levels for Unmanned Systems (ALFUS) Framework - Volume II: Framework Models, Technical Report, National Institute of Standards and Technology (NIST), 2007.

[30] B. T. Clough, Metrics, schmetrics! How the heck do you determine a UAV's autonomy anyway?, in: Performance Metrics for Intelligent Systems (PerMIS) Conference, Gaithersburg, MD, USA, 2002.

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

[32] Civil Air Navigation Services Organisation, Air Navigation Service Provider (ANSP) Considerations for RPAS Operations, 2014.

[33] 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.

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

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

[36] International Civil Aviation Organization, Doc. 8168, OPS/611: Procedures for Air Navigation Services: Aircraft Operations, 5th ed., ICAO, Montréal, Canada, 2006.

[37] European Organisation for the Safety of Air Navigation, EUROCONTROL Specification for the application of the Flexible Use of Airspace (FUA), 2009.

[38] J. Bang-Jensen, G. Z. Gutin, Digraphs: Theory, Algorithms and Applications, 2nd ed., Springer, 2008.

[39] EUROCONTROL Experimental Centre, User manual for the Base of Aircarft Data (BADA) revision 3.12, EUROCONTROL, 2014.