J Intell Robot Syst (2012) 67:155–181
DOI 10.1007/s10846-011-9648-3

Flight Plan Specification and Management for Unmanned Aircraft Systems

Eduard Santamaria · Enric Pastor · Cristina Barrado · Xavier Prats · Pablo Royo · Marc Perez

Received: 8 June 2011 / Accepted: 30 November 2011 / Published online: 17 December 2011

Abstract This paper presents a new concept for specifying Unmanned Aircraft Systems (UAS) flight operations that aims at improving the waypoint based approach, found in most autopilot systems, by providing higher level fligh plan specification primitives. The proposed method borrows the leg and path terminator concepts used in Area Navigation[1] (RNAV). Several RNAV leg types are adopted and extended with new ones for a better adaptation to UAS requirements. Extensions include the addition of control constructs that enable repetitive and conditional behavior, and also parametric legs that can be used to generate complex paths from a reduced number of parameters. The paper also covers the design and implementation of a software component that manages execution of the flight plan. To take advantage of current off-the-shelf flight control systems the constructs included in the flight plan are translated to waypoint navigation commands. In this way, the advanced capabilities provided by the flight plan specification language are implemented as a new layer on top of existing technologies. The benefits and the feasibility of the proposed approach for UAS flight plan management are demonstrated by means of a simulated mission that performs the flight inspection of Radio Navigation Aids.

Keywords Unmanned aircraft systems (UAS) · Flight plan specification · Flight plan management

[1] Area Navigation (RNAV) is a method of Instrument Flight Rules (IFR) navigation that allows an aircraft to follow any course within a network of navigation beacons, rather than navigating directly to and from the beacons.

E. Santamaria (✉) · E. Pastor · C. Barrado · P. Royo · M. Perez
Department of Computer Architecture, Universitat Politècnica de Catalunya - BarcelonaTech (UPC), Esteve Terradas, 7, 08860, Castelldefels, Spain
e-mail: esantama@ac.upc.edu

E. Pastor
e-mail: enric@ac.upc.edu

C. Barrado
e-mail: cristina@ac.upc.edu

P. Royo
e-mail: proyo@ac.upc.edu

M. Perez
e-mail: mpbatlle@ac.upc.edu

X. Prats
Escola d'Enginyeria de Telecomunicació i Aeroespacial de Castelldefels - EETAC, Universitat Politècnica de Catalunya - BarcelonaTech (UPC), Esteve Terradas, 5, 08860, Castelldefels, Spain
e-mail: xavier.prats@upc.edu

1 Introduction

Unmanned Aircraft Systems (UAS) are rapidly gaining attention due to the increasing potential of their applications in the civil domain. UAS can provide great value performing environmental applications, during emergency situations, as monitoring and surveillance tools, and operating as communication relays, among other uses. In general, they are specially well suited for the so-called D-cube operations (Dirty, Dull or Dangerous).

With no human pilot being on-board, the flight control system that guides the UAS during the mission becomes one of several critical components of the system. In their survey of autopilots for small fixed-wing UAS, Chao et al. describe the main features of a typical off-the-shelf autopilot [9]. The paper presents the details of a number of autopilots and identifies waypoint based navigation as a common capability provided by all of them. In waypoint navigation, to specify the aircraft's trajectory, the user introduces a list of waypoints, defined in terms of their latitude and longitude coordinates. These waypoints are then flied in sequence by the aircraft. Supported capabilities vary between autopilots. Features that may be available are autonomous take-off and landing, loiter and rally modes, a failsafe mode for contingencies, or the capacity to change waypoints in flight, among others. Some autopilots even provide some kind of language to specify the aircraft's behavior in a way that resembles a computer program.

For a future integration of UAS into shared airspace, other users must be taken into account. The reuse of concepts that are already familiar to airspace stakeholders will facilitate this integration. In commercial aviation, a specification method to encode Area Navigation (RNAV) procedures [12] has been in use for a long time. RNAV procedures are composed of a series of smaller parts called legs. Yet, to translate RNAV procedures into a code suitable for navigation systems the industry has developed the "Path and Termination" concept. Path Terminators provide the means to translate the text and the routes depicted on the charts into code readable by the aircraft's Flight Management System. A Path Terminator defines a specific flight path (e.g., heading, course, track, etc.) and a specific type of termination for each leg (e.g., the path terminates at an altitude, distance, fix, etc.). Our system takes this concept and brings it to the UAS domain by providing a subset of the legs available in RNAV.

In this article we present our work on flight plan specification and management for UAS. One goal of our research is to provide a flight plan specification mechanism that can be used across multiple autopilots, thus avoiding lock-in to a single solution and allowing its use on different platforms. The proposed flight plan specification and execution mechanisms also add capabilities to systems that lack some features, e.g., they allow to specify ground track around corners, provide support for specifying repetitive and conditional behavior, and overcome the limitations on the number of waypoints in the flight plan. Parametric legs, an extension that enables the specification of complex trajectories, for instance a lawnmower pattern, using a reduced number of parameters, facilitate the construction of the flight plan. Parametric legs also simplify the process of making changes, since the need to change a possibly quite large list waypoints, one at a time, is eliminated. This is specially valuable when changes need to be made during mission time. A very important aspect of the proposal, that will be key for enabling airspace integration, is its ability to include emergency routes and contingency reactions as part of the flight plan.

Another distinct characteristic of our flight plan specification language is the use of concepts inspired by current practices in commercial aviation. However, since civil UAS missions, which typically involve observation and monitoring of specific areas, can greatly differ from point-to-point transportation missions of commercial aircraft, extensions such as the ability to specify repetitive and conditional behavior have been added. We believe that the result is a very capable system that takes into account airspace integration issues, and that can be deployed in a wide range of scenarios.

The rest of this paper is organized as follows. In Section 2 an overview of related work is provided. Section 3 outlines the UAS architecture that accommodates the FPM, the VAS and the other

UAS services that may be present during a mission. Section 4 presents the main elements of the proposed language for flight plan specification. Section 5 explains how the Flight Plan Manager operates and the way in which waypoint generation takes place. Afterwards, the results of a simulated mission performed using prototype implementations of the proposed technologies are provided. Finally, Section 7 presents some conclusions and future work.

2 Related Work

Electronics were introduced in flight control under the fly-by-wire concept in 1050s. They evolved into Flight Control Systems (FCS), a set of automatic control algorithms, in a close-loop structure, to facilitate interaction of the pilot with the flight surfaces such as ailerons, elevators and rudders. Today, most commercial aircraft are also equipped with Flight Management Systems (FMS) which, by means of a FCS, provide automatic navigation functionalities [3]. FMS are large software components (up to one million lines of code) devoted to increasing safety and optimizing routing to save fuel. The challenge for the FMS is to compute a safe and optimum path, and direct the aircraft, via the FCS, to stay on the designed path under constantly changing wind direction and/or speed.

Seven manufacturers (CMC Electronics, GE Aviation, Thales, Honeywell, Rockwell Collins, Universal Avionics, Garmin) provide over 90% of the civil FMS in service today [15], mainly in commercial aviation. Commercial aviation includes passenger and cargo flights operating on regularly scheduled routes. Standard performance-based public RNAV instrument procedures (RNP) describe the path options available for these aircraft routes. The procedures, published in Aeronautical Information Publications (AIPs) by state agencies, are stored in the FMS as navigation databases using the ARINC Navigation Systems Database Specification 424 [2]. ARINC 424 is an industry standard for airborne navigation system data bases and flight plan preparation.

Among other functionalities, the FMS accesses the pre-programmed routes, identifies the next waypoint, calculates the aircraft position and provides inputs to the FCS to fly it. In conjunction with the Aircraft Communication Addressing and Reporting System (ACARS), the flight plan of an aircraft can be updated even while in service. In a near future, FMS will increase their functionalities with 4D trajectories, time based trajectories, required time of arrival, etc. In this domain we find proposals on new formal languages to encode the future aircraft trajectories. The Aircraft Intent Description Language [18] (AIDL) aims at enabling inter-operation among future ATM automation components and will include 4-D profiles when the required air and ground infrastructure exists. However, the semantic content of the language, with instructions such as *hold bearing* or *throttle up*, is very poor.

Unmanned civil aviation is evolving in the same direction that commercial aviation did before. Autopilots are being first developed at the control level, like FCS, and then extended to incorporate navigation functionalities, like FMS. An important difference between commercial manned aviation and unmanned civil aviation lies in the types of missions that will be performed. While commercial aviation is mainly concerned about carrying people and/or goods from one place to another one, in most cases, UAS will be used as sensing platforms that require a more complex behavior.

UAS research encompasses topics that range from intelligent behavior to low level stabilization and control algorithms, going through payload operation and data processing. In every practical application there is a need to specify the behavior of the UAS, either in a formal way or in a more interactive manner.

Some projects that deal with artificial intelligence problems are the ProCoSA project, the work of Caveney et al., and the work of Miller et al.

In ProCoSA [4], the mission evolves according to a set of Petri nets [23] that describe the behavior of the aircraft. A program called “Petri Player” runs the Petri nets and manages operation of the planning, guidance and data subsystems and their communications. This system aims at achieving a high level of autonomy. On-board computing resources are responsible for flight planning and

re-planning. In ProCoSA the flight plan is embedded within the Petri net specification, which identifies the different flight phases, e.g., navigate to mission area, but does not reflect actual trajectories, which are created in real-time. The system supports several trajectory patterns for aerial observation.

The goal of Caveney's work [8] is to define applications for multi-agent collaborative control of fixed-wing unmanned aerial vehicles. The applications are described through three basic behaviors: Traveling, Watching, and Tracking. These three behaviors are constructed using two primitive actions: going somewhere, and holding at a particular location. The system is built in two layers: a Collaboration layer, and a Trajectory Generation layer. These layers are respectively responsible for assigning behaviors and computing the nominal path. Miller et al. [22] follow a similar approach. Their system also features on-board path planning and operates in three different modes: FlightPath, Investigate and Standby. Both systems feed the autopilot with automatically generated waypoints that define the desired maneuvers. Caveney and Miller implement their systems using the Piccolo [10] autopilot from Cloud Cap Technology. A flight plan of the Piccolo autopilot is made of linked lists of waypoints up to a maximum of 100. Each waypoint encodes latitude, longitude, altitude, and the index of the next waypoint. Different options allow waypoints to be marked as pre-turn (to avoid overshoot), as an orbit waypoint, as slope for constant rate climb or descend, and can have associate actions like lights, chute and drop.

Another group of projects could be characterized as research works whose focus is to build an observation platform for data acquisition and processing. Many of these projects use a COTS autopilot as part of their prototype platforms, and the system relies on the capabilities of the autopilot to define the desired route. In [5], Bendea et al. describe a low cost UAV for post-disaster assessment. The system uses the MicroPilot autopilot [21], which features a a script language that can be used to describe the flight plan in a way that resembles a computer program. The available navigation commands in MicroPilot are climb, waitClimb, flyTo, fromTo, turn and circle. Operations for performing calculations are also available, e.g., add, sub, div and mult. The script language has a mechanism for defining patterns and also features commands, such as repeat, skipEqual, skipNotEqual, and others, to control the flight. Other works, such as the ones presented in [35] and [11] provide additional examples, this time making use of the Kestrel autopilot [29] by Procerus Technologies. With the Kestrel autopilot, three types of commands are used for specifying the flight plan: Waypoint, Loiter and Goto. A group of flight commands defines a route, which is always ended with a Goto command that tells the UAS which waypoint to jump to in order to restart the route. Execution of a new route is commanded by the UAS operator by selecting an item on the new route. The ground control station facilitates the definition of rectangular search areas by automatically generating the waypoints that cover an area specified by the user, either before takeoff or during flight.

Other research, like Mcmanus' work [20], focuses on automatic path planning. In the cited article a mission planning system, which takes into account airspace integration concerns, is presented. Given the mission objectives and situational awareness information, the flight path planner computes a list of waypoints to be flown by a COTS autopilot. Interesting aspects of the system are its ability to execute collision avoidance maneuvers, and its support for a number of commands for instructing the aircraft to circle, perform a holding pattern, an eight figure or a grid search. Similarly to our approach, this commands are translated to sequences of waypoints. However, the concept of a flight plan as a document in electronic format which could be transmitted to other stakeholders is not present.

Finally, the Paparazzi Project [7] represents an effort where a flight plan specification language is available. The language supports commands for requesting the system to keep a certain altitude, a given heading, go to a given waypoint and circle around a waypoint, among others. All these flight plan elements are grouped in blocks. Constructs to repeat parts of the plan and to directly go to and fly a specific part of it are also provided.

Although both the Paparazzi specification language and the one presented in this paper are

based on XML [6], there are several important differences between them. The Paparazzi solution targets μUAS and tries to take full advantage of a particular autopilot, which is intended for very small systems and has very limited processing capabilities. Its specification language contains constructions that resemble the C programming language, with which it is highly integrated. Finally, all the contents of the flight plan are compiled into a binary program. This binary program is then transmitted to the autopilot for its execution during the UAS flight. This approach limits the capabilities for making changes to the flight plan during the mission to only moving existing waypoints to new locations or to skip or go directly to a flight plan block.

As conclusion, we believe that the history of FMS provides insights on how the capabilities of current UAS autopilots will evolve. Also, FMS are the systems governing the flight of many aircraft flying today. Therefore, from an airspace integration point of view, it seems reasonable to try to describe UAS flight plans using primitives already used and well known. However, since UAS missions differ from those of commercial aviation, some extensions/adaptations need to be done. With regard to research projects in the field of UAS, their goals and approaches greatly vary. We believe that there is value in those approaches that try to build highly intelligent and autonomous systems, but in order to be able to share airspace with other users, UAS systems need to be more predictable. At the other end, systems that rely on the capabilities of current COTS autopilots lack features and the whole system is dependent on a very specific product. While some autopilots exhibit more advanced capabilities, these are either expressed as pilot commands, as autopilot modes or as embedded program instructions, and are not fully integrated in a self-contained flight plan specification. A common functionality of all autopilots is the capability to fly a predefined sequence of waypoints. Taking advantage of this common capability, our system implements a flight management service that dynamically translates high level primitives, as the ones found in our flight plan specification language, into a sequence of waypoints that can be handled by virtually any autopilot.

3 System Architecture

The main elements of our architecture that are involved in the execution of the flight plan are the Flight Plan Manager (FPM), the Virtual Autopilot System (VAS), and the COTS autopilot (see Fig. 1). We refer to the FPM, the VAS, and other software components as services. A document containing the flight plan specification is submitted to the FPM. The FPM processes the leg based flight plan description and generates waypoint commands that are sent to the VAS. During flight, the FPM can receive flight plan updates and other commands to adapt UAS operation to the mission needs. The role of the VAS is to operate as an intermediate service that isolates the FPM from the particularities of each autopilot solution. To this end, the VAS provides a standardized interface for the FPM to interact with. It is also responsible for extracting telemetry data from the flight control system and making these data available to other services in a compatible way.

The flight management services presented in this paper are part of a wider set of services organized following the architecture proposed by Pastor et al. in [26]. This architecture conceives a UAS as a distributed system, where a number of software components use a common communications infrastructure to exchange information and collaborate. Each computational node can run one or more services. Communication between services follows a publish/subscribe model and is managed by a middleware layer [19]. There is a collection of services that have been identified as

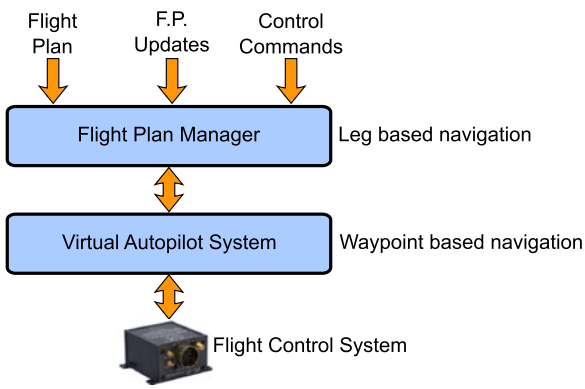


Fig. 1 Elements involved in the execution of the flight plan

necessary to perform a wide range of missions. These services are standardized by what is called the UAS System Abstraction Layer (USAL) [31].

The USAL concept can be compared to the way operating systems handle device drivers. Computers have hardware devices used for input/output operations, each one having its own particularities. The operating system offers an abstraction layer to access such devices in a uniform way. In a similar fashion, the USAL publishes an Application Programming Interface (API) that provides end-users with a standardized way to access hardware elements. The USAL makes use of the communication primitives provided by the underlying service-oriented middleware layer.

Another goal of the USAL is to provide a set of components that can be reused across different missions. The available services will cover an important part of the generic functionalities present in many missions. Therefore, in many cases, to adapt the system to a new mission it should be enough to reconfigure the services deployed onto the UAS.

3.1 USAL Services

In Fig. 2 the set of services that we envision as forming part of the UAS are shown. These services can be organized in four categories: Flight, Awareness, Mission and Payload services.

- *Flight Services*: This is arguably the most important category, since the success of the mission and its safety depend, to a great extent, on the system's ability to follow the intended flight path. The system relies on the installed autopilot for low level flight control, but there are a number of services that add capabilities on top of that. One such service is the Virtual Autopilot System (VAS), that manages all interactions with the autopilot. The VAS provides waypoint navigation capabilities and a number of telemetry flows regarding the UAS position, attitude, autopilot status, etc. It also isolates the autopilot from the rest of the system, thus avoiding dependence on a particular autopilot solution. The waypoint navigation primitives of the VAS are used by the Flight Plan Manager in order to govern the UAS flight. Other services included in this category, such as the Electrical Manager, the Engine Manager and the Contingency Manager, help to improve safety and reliability.
- *Mission services*: Mission services are those responsible for the actual execution of the mission. The Mission Manager orchestrates operation of flight and mission related services in order to achieve the mission goals. The Mission Manager listens to system events and responds in a purely reactive fashion. Services that store and analyze sensed data are also found in this category. Planning services also fall into this category.
- *Payload services*: Services that handle operation of sensors and actuators belong to this category. There are many kinds of sensors

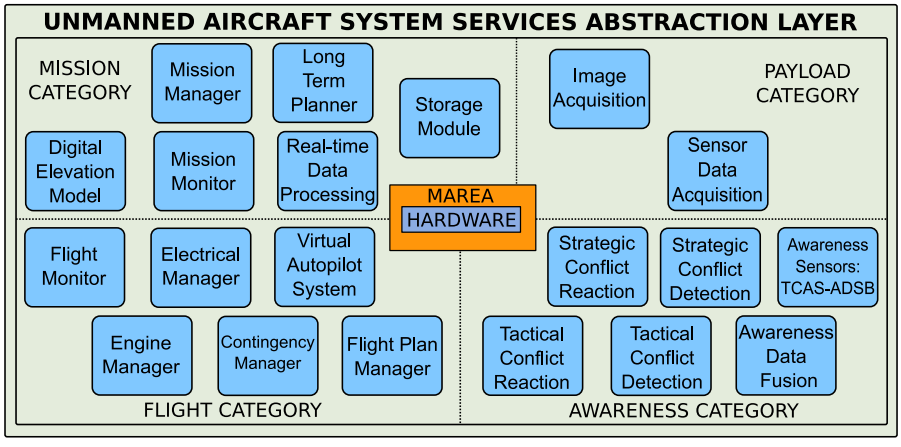


Fig. 2 General view of USAL architecture

that we may need to consider: GPS, IMU, anemometers, visual, infra-red and radiometric cameras, chemical and temperature sensors, radars, etc.

– *Awareness services*: This category includes those services that gather information about the environment the UAS is operating in. These services are critical for a successful integration of UAS into shared airspace. Awareness services handle interaction with cooperative aircraft through transponders, TCAS or data-link systems and try to detect non-cooperative aircraft through visual or other kinds of sensors. Services in this category will also take control and command emergency maneuvers in critical situations where an immediate response is required.

Although the USAL is composed of a large set of services, not all of them need to be present at all times. Only those required for a given configuration/mission should be present and/or activated in the UAS.

In this article we focus on the *Flight Plan Manager* service. The *Flight Plan Manager* is a service designed to provide flight management functions that go beyond following a predefined sequence of waypoints. The FPM offers structured flight-plan phases with built-in emergency alternatives, leg based navigation and constructs to enable forking, repetition and generation of complex trajectories. The supported ability to specify alternative procedures for contingency and emergency situations will be key for enabling airspace integration and, consequently, it is a very important aspect of the proposal. However, a detailed description of the concept is beyond the scope of this article. A more thorough explanation of the contingency and emergency reaction concept can be found in [27].

Next section describes the main elements of the flight plan specification language used to describe the flight plans that the FPM executes.

4 Flight Plan Specification Language

The flight plan specification language provides the mechanism to describe the flight path that the unmanned aircraft should follow. One of the core concepts the flight plan specification language builds on is the notion of leg, which is already used in commercial aviation for the specification of Area Navigation (RNAV) procedures [12]. RNAV legs are specified using Path Terminators, which define a specific flight path and a specific type of termination for each leg. Our system takes this concept and brings it to the UAS domain. A subset of the legs available in RNAV are implemented and extended with new constructs for a better adaptation to UAS needs.

4.1 Flight Plan Document Structure

As shown in Fig. 3, the proposed specification language organizes the UAS flight into stages,

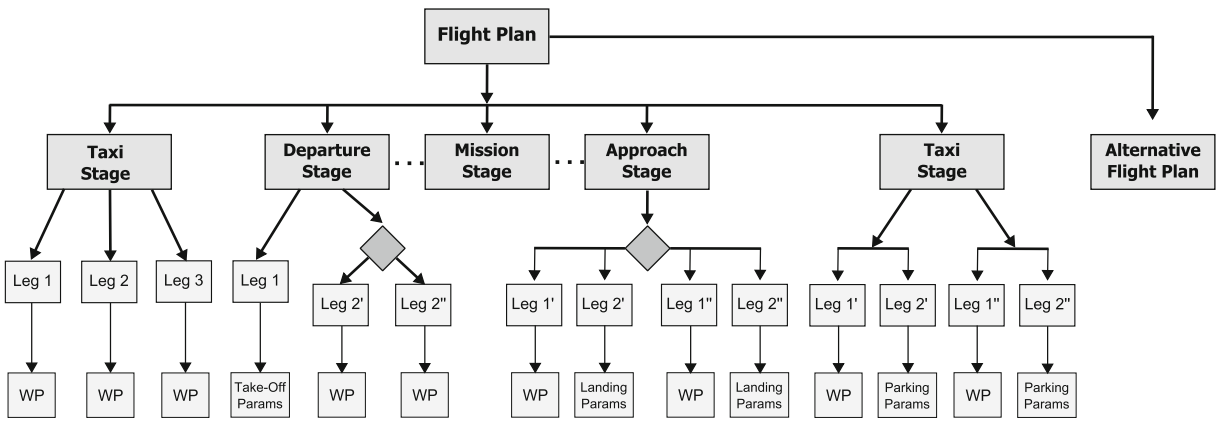


Fig. 3 Flight plan structure

each one representing a different flight phase. A stage contains a number of legs that specify the flight path during the execution of the stage. Since support for emergency flight plans is also provided, the specification document may contain more than one flight plan: The main flight plan will describe the whole mission under normal circumstances, and emergency ones will indicate alternatives to be flown when some contingency occurs. All of them are organized in the same way, the main difference being that the emergency plans will not require some initial stages. Other information that can be specified during the construction of the flight plan are locale data, indicating which units are being used, and fixes, which represent specific locations with a name associated to them.

An XML document, whose upper level elements are shown in Listing 1, is used to store the flight plan related data.

```
<FlightPlan
    xmlns='http://icarus.upc.es/schema/FlightPlan/1.1'>
    <!-- units and separators -->
    <Locale> ... </Locale>
    <!-- specific named locations -->
    <Fixes> ... </Fixes>
    <!-- emergency flight plans -->
    <EmergencyPlans> ... </EmergencyPlans>
    <!-- main flight plan -->
    <MainFP> ... </MainFP>
</FlightPlan>
```

Listing 1 XML flight plan document structure

The elements that can be found within the main flight plan are shown in Listing 2. Stages are the largest building blocks used in the flight plan. The most important element contained inside a stage is its list of legs. Each leg indicates how to reach a given waypoint. A waypoint consists of a geographical position defined in terms of latitude/-longitude coordinates, that can be accompanied by target values for the altitude and speed of the aircraft at that waypoint. Therefore, changes of speed and altitude are specified at a waypoint level. The way in which the target speed/altitude is reached is controlled by the installed autopilot. However, the translation process from legs to waypoints may require the addition of extra waypoints, and speed and altitude values will need to be set for them. The details on how these values are assigned is discussed in Section 5.2, which deals with the waypoint generation process.

A stage can be entered from different points and, depending on the choices made during its execution, can finish at different locations. The initial and final legs of the stage are respectively found in the *initialLegs* and *finalLegs* elements. Optionally, emergency plans can be indicated at the flight plan, stage or leg level. Emergency flight plans found deeper in the document hierarchy will have higher priority. A partial flight plan follows the same structure as the main flight plan but contains only those stages necessary to fly from the current position to the landing runway of choice.

```
<MainFP id="FPID">
    <name>Name of the flight plan</name>
    <description>Text describing the flight plan</description>
    <!-- List of stages that form the flight plan follows -->
    <stages>
        .
        .
        <stage id="STID" type="Departure" manualOnly="false">
            <name>Name of the stage</name>
            <description>Text describing the stage</description>
            <!-- Legs that belong to this stage -->
            <legs> ... </legs>
            <!-- Space separated list of leg ids -->
            <initialLegs>LStart</initialLegs>
            <!-- Space separated list of leg ids -->
            <finalLegs>LEnd</finalLegs>
            <!-- Emergency flight plans -->
            <emergency> ... </emergency>
        </stage>
        .
        .
    </stages>
    <emergency>EmergencyFP1 EmergencyFP2 ... </emergency>
</MainFP>
```

Listing 2 XML description of main flight plan

Following sections provide more details about each of these elements. A more thorough description of the flight plan specification language, together with the details on how it is processed and executed, can be found in [32].

4.2 Stages

A stage groups together legs that seek a common purpose. During its total flight, the aircraft can perform different types of stages. The available stage types are listed in Table 1.

Every stage, except for the first and last stages, has a single predecessor and a single successor. A stage may have more than one final leg. For instance, a take-off stage may end at different points depending on the selected take-off runway. Also, a stage may have more than one initial leg as could be the case for departure procedures that start at different positions depending on the executed take-off operation. There will be a one-to-one correspondence between the final legs of a given

Table 1 Stage types

Taxi	Ground operations to go to/move away from the active runway.
TakeOff	Take-off run and initial climb phases.
Departure	Specific legs joining the take-off with the starting point of the EnRoute stage.
EnRoute	Cruise to a destination area.
Mission	Series of legs that will be flown during main mission operations.
Arrival	Specific legs transitioning from the EnRoute stage to the Approach stage.
Approach	Lateral and vertical specific maneuver to place the aircraft in a safe position and altitude to land at the active runway.
Land	Landing operation.

stage and the initial legs of the next one. Thus providing a seamless transition between stages. There are constructs that enable the flight plan designer to provide this one-to-one correspondence.

4.3 Legs

A leg specifies the flight path to get to a given waypoint. Most legs contain a destination waypoint and a reference to the next leg. Only intersection legs, which mark decision points, are allowed to specify more than one next leg.

There are four different kinds of legs:

- Basic legs: Specify leg primitives such as 'Direct to a Fix', 'Track to a Fix', etc.
- Iterative legs: Allow for specifying repetitive sequences.
- Intersection legs: Provide a junction point for converging trajectories, or a forking point where a decision on what leg to fly next can be made.
- Parametric legs: Are used to specify complex trajectories from a reduced number of input parameters. As an example consider a scanning pattern, given its geometry and a few additional parameters, an algorithm can automatically generate the desired trajectory.

Intersection legs differ from the rest in that they may be reached from more than one predecessor and may lead to more than one successor. All legs have an optional parameter indicating what emergency flight plans are candidates to be carried out when an emergency occurs.

A brief description of the available leg types follows. Sample code showing how some of them are encoded using the XML based specification language can be found in Listings 4 and 5.

4.3.1 Basic Legs

This section describes the basic legs available to the flight plan designer. They are referred to as basic legs to differentiate them from control structures like iterative or intersection legs and parametric legs. All of them are based on already existing ones in RNAV. Its original name is preserved.

Figure 4 illustrates the available basic legs. A brief description of each one follows:

- Initial Fix (IF): Determines an initial point. It is used in conjunction with another leg type (e.g. TF) to define a desired track.
- Track to a Fix (TF): Corresponds to the great circle track over ground joining two way-

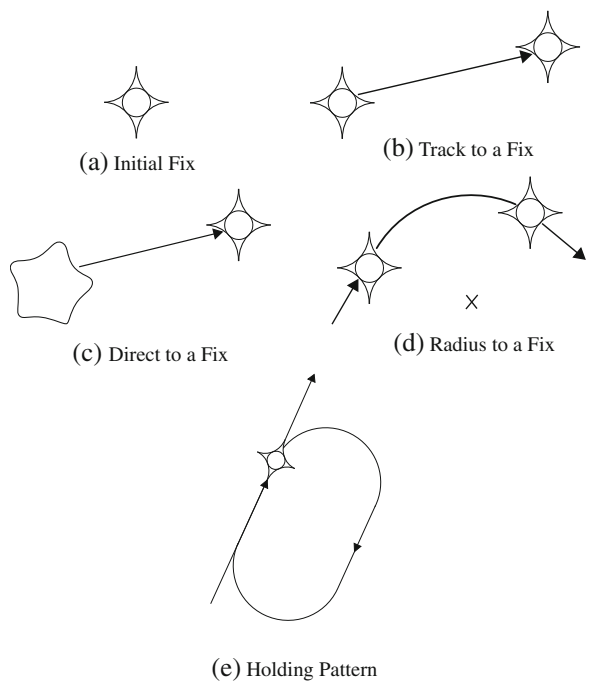


Fig. 4 Basic leg types available

points. The initial position is the destination waypoint of the previous leg.
- Direct to a Fix (DF): Is a path described by an aircraft's track from an initial area direct to the next waypoint, i.e., fly directly to the destination waypoint whatever the current position is.
- Radius to a Fix (RF): Is defined as a constant radius circular path around a defined turn center that terminates at a waypoint. It is characterized by its turn center and turn direction.
- Holding to a Fix (HF): An HF is used to define a holding pattern path. The leg terminates when the hold waypoint is crossed after a given number of iterations or when a given condition is no longer satisfied (regardless of the number of iterations). The shape of the holding pattern is determined by a number of parameters that specify the distance between the two turn centers, the turn diameter, the angle that its sides form with respect to North, and a turn direction. Setting the distance between turn centers to 0 results in an orbit pattern.

4.3.2 Iterative Legs

A complex trajectory may involve iteration, thus the inclusion of iterative legs. An iterative leg has a single entry (i.e., its body can be entered from a single leg), a single exit, and includes a list with the legs that form its body. An iterative leg also includes an upper bound indicating the number of repetitions and, optionally, a condition that controls its termination.

The structure of an iterative leg is shown in Fig. 5. An inbound arrow indicates the point where the iterative leg execution starts. Each time

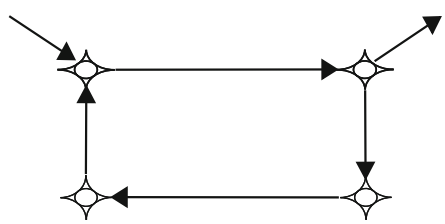


Fig. 5 Iterative leg

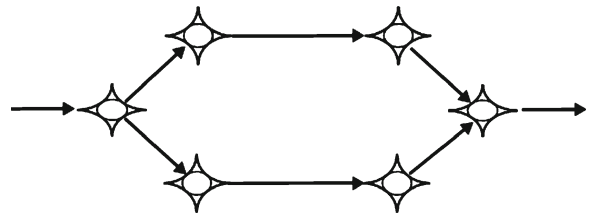


Fig. 6 Intersection legs

the final leg (the one that leads to the exit point) is executed, a counter is incremented. When a given count is reached, or a specified condition no longer holds, the leg will be abandoned proceeding to the next one.

4.3.3 Intersection Legs

Intersection legs are used in situations where there is more than one possible path to follow and a decision needs to be made (see Fig. 6). This leg type contains a list with the different alternatives and a condition for picking one of them. Intersection legs are also used to explicitly indicate where two or more different paths meet.

Together with parametric and iterative legs, intersection legs provide a powerful means for adapting the flight as best suited to the ongoing mission circumstances.

4.3.4 Parametric Legs

Parametric legs permit complex trajectories to be automatically generated from a reduced number of input parameters. If the actual values of these parameters change, the resulting trajectory will be dynamically recomputed. In this way, the aircraft trajectory can be modified depending on the evolution of mission variables. Parametric legs provide an increased level of adaptation to changes that occur during mission time. Possible patterns that could be obtained using parametric legs are shown in Fig. 7. While (a) and (b) could be used to explore a given area, (c) could be used as a pattern for inspecting a more specific point.

With the use of parametric legs two goals are achieved. First, complex trajectories can be generated with no need to specify a possibly quite long list of legs. Second, the UAS path can dynamically adapt to the mission requirements.

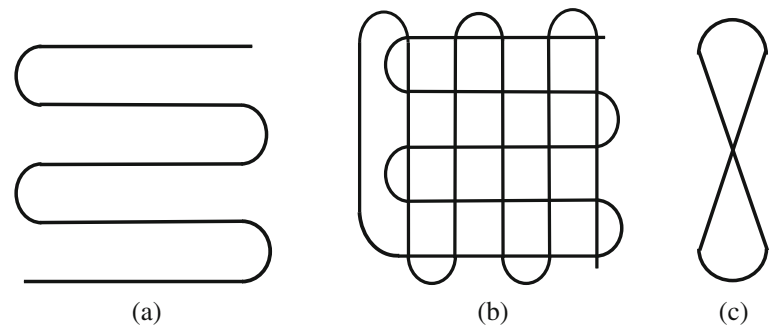


Fig. 7 Different flight patterns: basic scan (**a**), complex scan (**b**) and eight pattern (**c**)

4.4 Conditions

There are several points in the flight plan where conditions can be found: namely in holding patterns, iterative legs and intersection legs. The condition of an intersection leg specifies which path to follow, whereas for the holding pattern and the iterative leg the condition specifies when the aircraft should leave the current leg and proceed to the next one.

Conditions are represented in the flight plan as an identifier with an associated integer value. Each leg that depends on a condition contains a condition identifier. The path selection that the condition governs is determined by the value assigned to the condition. This value can be provided by a UAS operator or by one of the UAS services. When the value associated to a condition changes, waypoints are dynamically recomputed to reflect the new choice.

4.5 Flight Plan Updates

The flight plan described using the proposed specification language can be modified by means of flight plan updates. A flight plan update consists of an XML message with a syntax that closely resembles the one used in the flight plan specification language.

```
<Change>
   <MainFP targetId="FPID">
      <stages>
         <stage targetId="MyMissionStage">
            <legs>
               <leg targetId="LegA" xsi:type="fp:DFLeg">
                  <dest>
                     <coordinates>
                        41.580203095 1.7369781057
                     </coordinates>
                  </dest>
               </leg>
            </legs>
         </stage>
      </stages>
   </MainFP>
</Change>
```

Listing 3 Example of simple update message

The purpose of the updates is to enable the flight plan to be modified during the execution of the mission. The simplest modification consists of changing an attribute of a given leg, e.g., the coordinates of its destination waypoint. In this case, the update message would contain the information required to identify the leg and the new values for its destination waypoint (see Listing 3). Figure 8 illustrates the changes resulting from such an update. A more aggressive approach may involve restructuring the flight plan with the addition of new legs and the removal of existing

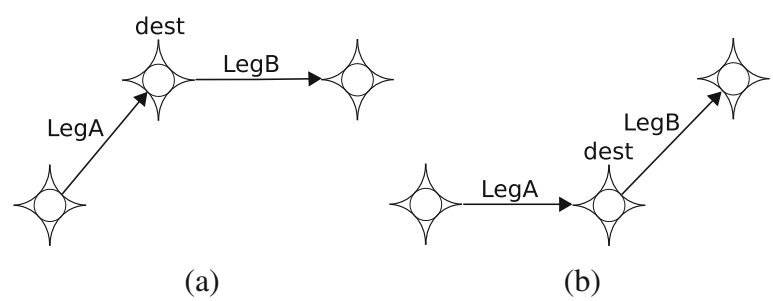


Fig. 8 Example legs before (**a**) and after (**b**) an update

ones. All updates should refer to forthcoming legs, updating the leg that is under execution is not allowed.

5 Flight Plan Management

The previous section described the language used for specifying flight plans. This section presents the Flight Plan Manager (FPM), which is the service responsible for their processing and execution. It forms part of a wider set of services that, together, provide the UAS with all its capabilities. The FPM collaborates with some of those services to perform the execution of the flight plan.

The FPM can be seen as a translator of legs to waypoints. This translation process enables leg based navigation on systems that only support waypoint navigation. From the VAS or autopilot perspective, the FPM can be seen as a provider of waypoints to fly to. From a mission related perspective, the FPM is the service that the UAS operator talks to in order to control the flight progress and make it adapt to the mission needs. There are multiple possibilities of interaction with the FPM, the primary ones being setting condition values, sending updates to flight plan elements and triggering execution of emergency plans.

5.1 FPM Service Capabilities

The main responsibility of the FPM is generating the waypoints that will make the aircraft follow the flight path described using our flight plan specification language. As an example, in Fig. 9, the waypoints that would be generated for executing a scanning pattern are shown. This maneuver appears in the flight plan as a single parametric leg (see Listing 4). The values of *dim1* and *dim2* together with *angle* determine the geometry of the area that needs to be scanned. The *dest* parameter indicates the destination waypoint of the maneuver, which serves as a reference to place the area on the map. The vertex opposite to the parametric leg's destination is the starting point of the scanning pattern. If this vertex does not coincide with the destination of the previous leg, the aircraft will perform a DF from the destination of the previous leg to the starting point of the parametric leg. The *separation* parameter indicates separation between passes across the area. This value should

Fig. 9 Generated waypoints for a scanning pattern

be set taking into account both the capabilities of embarked sensors and aircraft performances.

```
<leg id="scanleg" xsi:type="BasicScanLeg">
    <dest>
        <coordinates>
            41.5493424917977 1.77254310685181
        </coordinates>
        <speed>60</speed>
    </dest>
    <dim1>6000</dim1>
    <dim2>5500</dim2>
    <angle>90</angle>
    <separation>800</separation>
</leg>
```

Listing 4 XML description of a parametric leg

When processing the parametric leg, the FPM computes all the waypoints necessary to execute a series of TF legs connected by RF legs. The detail of a constant radius turning maneuver is shown in Fig. 10. As explained in Section 5.2, with the assumption that the underlying autopilot only supports waypoint navigation, RF legs are approximated by a sequence of waypoints. The use of parametric legs eliminates the need for having to manually introduce each waypoint of the scanning pattern. Moreover, if the area of interest changes, all the waypoints can be automatically recomputed.

Fig. 10 Detail of a constant radius turning maneuver from Fig. 9

The basic requests that must be handled by the FPM are:

- Receive and initiate execution of a flight plan.
- Assign new values to conditions that govern selection between alternative routes.
- Receive and process updates to the initial flight plan.
- Trigger execution of an emergency plan.

Additional functions provide more control over the aircraft maneuvers and the state of the FPM:

- Skip the leg under execution, i.e., immediately start execution of the next leg.
- Jump directly to a leg located further in the flight path, therefore ignoring some intermediate legs (in later sections we refer to this function as the *Goto* command).
- Pause flight plan execution while performing a holding pattern.
- Switch to a standby state, which is going to happen when the UAS is under manual control or controlled by another UAS service.
- Resume operation after a pause, or once control is regained.

Together with the navigation commands sent to the VAS, the FPM also generates several information flows that can be exploited by other services. This data includes the position of the aircraft in flight plan terms, i.e., what is the current stage, the current leg, and other leg-related information such as current iteration of an iterative leg. It also periodically publishes what emergency plans are available, which may depend on the stage or leg being flown, and their estimated duration. Finally, information relative to the current operating state of the service is also provided.

Figure 11 depicts a simplified diagram with the main messages interchanged between the Ground Station (GS), the FPM and the VAS. UAS operation starts by uploading the flight plan definition to the Flight Plan Manager (*UploadFlightPlan*). Validation that the flight plan is collision free and feasible according to performances is done before upload. The FPM will parse the flight plan and

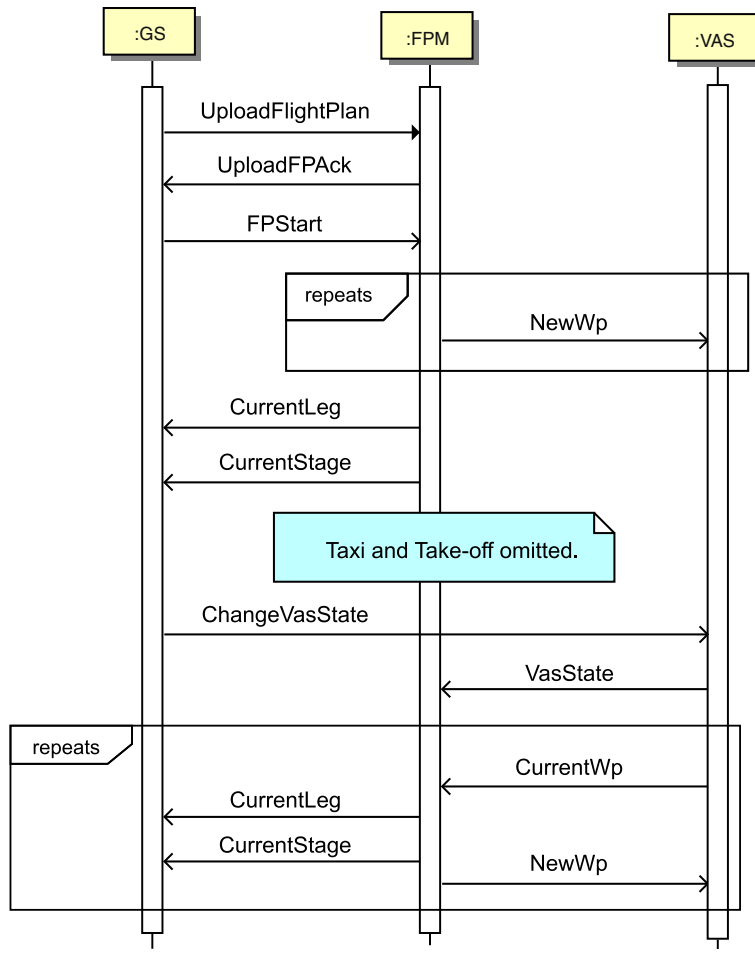


Fig. 11 Simplified diagram of navigation messages between FPM and VAS

check that it is syntactically correct and that it complies with a number of constraints, e.g., that stages come in valid order. After confirmation that the flight plan has been received (*UploadFPAck*), the Flight Plan Manager can be started (*FPStart*). At this point, the FPM starts generating waypoints which are progressively sent to the VAS (*NewWp*). Only a limited number of waypoints is transferred at a time from the FPM to the VAS. This waypoint window is used to ensure that the specified number of waypoints is always available to the VAS. Limiting the number of waypoints also helps keeping communications cost penalties low when old waypoints need to be discarded and replaced by new ones. Such situation may occur due to changes in the flight plan. The initial set of waypoints is immediately transferred, then additional waypoints are sent as the old ones get flown. Each time a waypoint is flown, the VAS generates an event (*CurrentWp*) to inform the FMS and other services. At the same time, the FPM informs other services of those legs and stages that are being flown (*CurrentLeg* and *CurrentStage*). Waypoint navigation will only start after the VAS switches to the *Navigation* state (*ChangeVasState*). The waypoint generation process keeps going until the landing phase, which is directly implemented by the VAS.

5.2 Waypoint Generation

The flight plan submitted to the FPM is parsed and translated to a tree-like internal representation. A flight plan has a number of stages that, in turn, contain one or more legs each. These legs

can take different forms depending on its type, but all of them (except iterative and intersection legs) have a destination waypoint.

Flight plan objects are organized forming a tree structure whose root node represents the complete plan (see Fig. 12). Stages are located at the second level with legs following. At this point, some degree of recursion may be found due to iterative legs, whose children legs form the body of the iterative structure. This representation is traversed and waypoints are generated for the encountered legs.

The FPM implements a producer-consumer model. A waypoint generator object has the role of the producer and stores waypoints in a queue. A controller object, responsible for handling interactions of the FPM with other services, acts as the consumer. It follows the consumer role when it takes waypoints from the queue of generated waypoints and sends them to the VAS.

These two objects operate in a decoupled manner: the producer continually generates waypoints ahead of time until the end of the flight plan is reached. The controller uses a configurable window size to retrieve generated waypoints and send them to the VAS. Each time a reached notification is received, a new waypoint is taken from the queue and forwarded to the VAS. In this way the VAS always holds a minimum number of waypoints to fly to.

In order to support the different types of requests, each generated waypoint has some extra information associated to it that enables the FPM to tell which leg this waypoint belongs to, the iteration it was generated in, etc. This control data is used to resume waypoint generation at the right place when a change in the flight plan invalidates waypoints that have already been generated.

Figure 13 illustrates how the waypoint generation process takes place. The initial step is taking the first leg for which we want to generate waypoints. Structural representation of the flight plan is kept apart from waypoint generation classes following a separation of concerns principle, therefore the taken leg knows nothing about what waypoints will come out of it. The next step is then obtaining an object that knows how to generate waypoints for that leg. This generator object is selected taking two elements into account: (1) what type of leg are we handling and (2) what type of autopilot are we targeting. Now the actual generation of waypoints can take place. As seen in Fig. 13b, some parameters are required to perform this computation:

- *Initial position* is the position of the aircraft at the beginning of *current_leg*.
- *Heading* is the entry path angle the aircraft is following when reaching the *initial position*.
- *Speed & Altitude* are the estimated values the *initial position* is reached at.
- *Aircraft parameters* are the aircraft's bank angle and a correction factor to account for the transition time required to reach the bank angle.

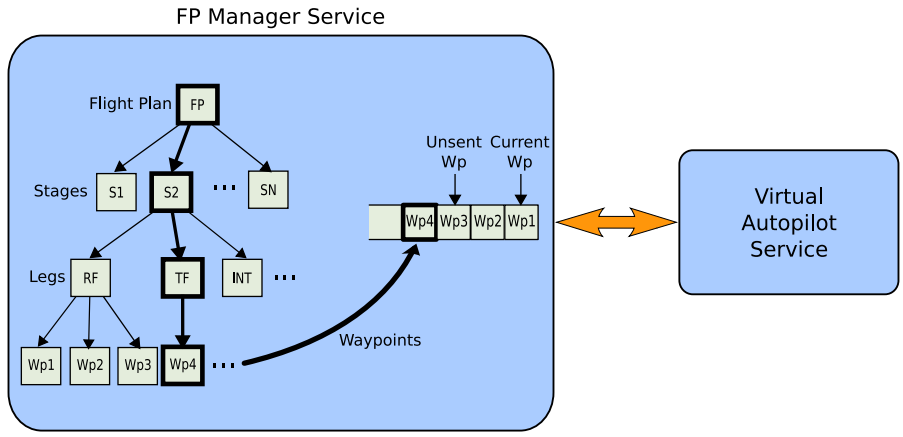


Fig. 12 Flight plan execution

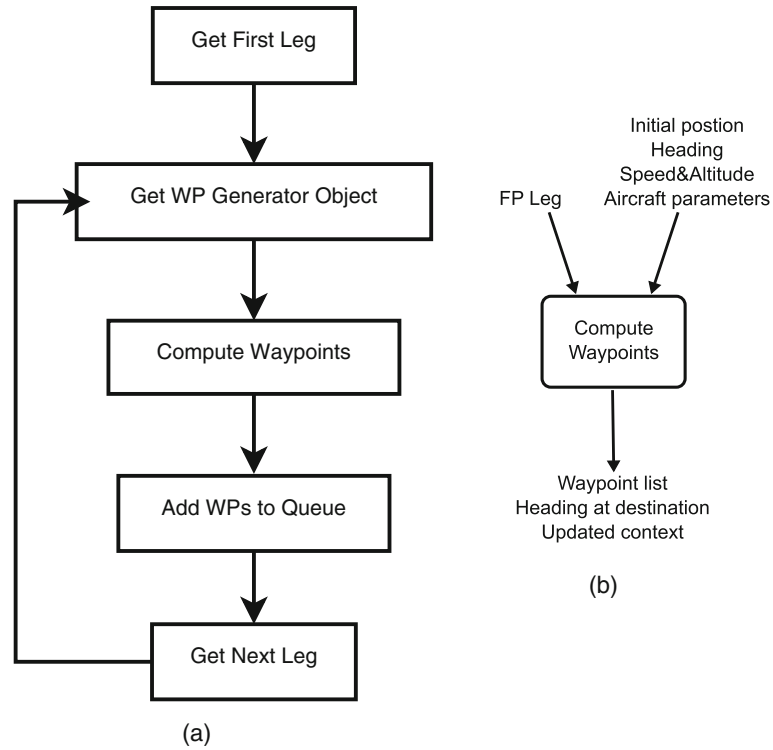


Fig. 13 Waypoint generation process (**a**) and inputs/outputs of *Compute Waypoints* step (**b**)

As a result of the computation, a list of waypoints is obtained together with the predicted heading at the last waypoint. The list of waypoints is added to the waypoint queue and the new heading is used in the computation of the next leg.

Although not yet considered in the current implementation of the FPM, other parameters, such as wind speed and direction, should also be taken into account. The inclusion of wind effects is a very important requirement for the system to be able to operate in the field, otherwise the actual trajectory of the aircraft could greatly differ from the planned one. Previous work exists that addresses the waypoint generation problem taking wind into account, see for example [25] and [13]. We believe that the internal design of the FPM is flexible enough to allow the addition of estimated wind data as an additional parameter during the waypoint computation process. Techniques as the ones described in the previously referenced papers could be used to determine the placement of the generated waypoints in a way that the effects of wind are compensated and the desired trajectory is achieved. Up to now, our research efforts have been focused on the development of the flight management concept, the proposed flight plan specification language and the software architecture of the system. The inclusion of wind data into the waypoint generation algorithms is part of our future work.

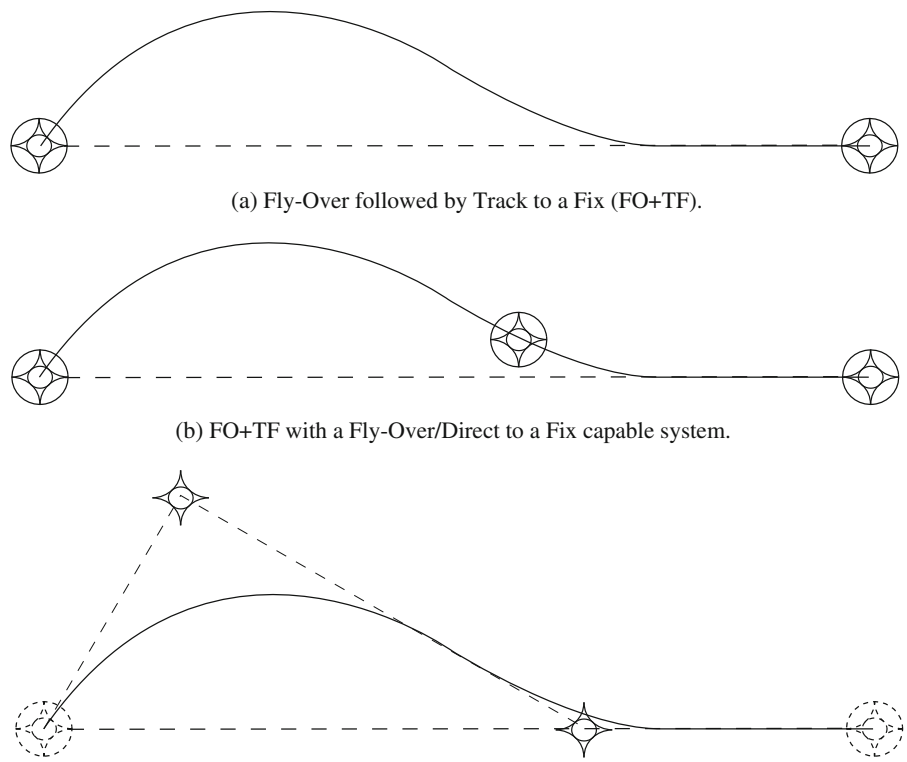
Our flight plan specification language and waypoint generation process emphasize lateral navigation. The way in which vertical navigation takes places will depend on the underlying UAS autopilot. During the waypoint generation process, when multiple waypoints are generated from a given leg, and the initial altitude differs from the altitude at the destination, altitude of intermediate waypoints will linearly be increased or decreased according to the distance along the track between each pair of consecutive waypoints. The idea behind this approach is to obtain a smooth climb/descent profile. However, depending on the autopilot capabilities, one could actually get a suboptimal step by step climb/descent. Therefore, this simple approach is not completely satisfactory and should be revised in future versions. A similar problem is encountered with regard to changes of

speed. The approach taken in this case has been to set the speed of intermediate waypoints to the same value indicated in the destination waypoint of the leg under consideration.

The system supports five types of basic legs: Initial Fix, Track to a Fix, Direct to a Fix, Radius to a Fix and holding patterns. In some cases, waypoint generation is trivial, as an example, generating waypoints for an Initial Fix is accomplished by just adding the fix to the queue of generated waypoints. In other cases, waypoint generation is far more complex. Waypoint generation for a holding pattern takes into account that different entry procedures [17] may need to be executed depending on the direction the aircraft comes from.

Another aspect that adds complexity to the generation process is that, while waypoint based navigation is a common denominator of the vast majority of UAS autopilots, there can be variations with regard to their capabilities to capture and hold to a given track or to perform both fly-by and fly-over waypoints. An example illustrating how these restrictions can be overcome with smart waypoint generation techniques, that take into account the system capabilities, is shown in Fig. 14. As a direct benefit from keeping structural leg data and waypoint computation separated in different classes, we are able to pick the generation algorithm that best suits each situation.

In Fig. 14a, we see what should be the required trajectory for performing a Track to a Fix having a fly-over waypoint as the initial aircraft position. Once the starting waypoint has been overflown, the aircraft turns right in order to intercept the track. Figure 14b and c respectively illustrate how this same trajectory can be obtained with an autopilot system that only supports Direct to a Fix navigation and with one that only implements a fly-by. In both cases, additional waypoints are strategically added and others removed so that the intended trajectory is achieved. In Fig. 14b, an extra waypoint is added so that the aircraft is forced to turn twice in order to reach the destination. In Fig. 14c, the two fly-over waypoints at the



(a) Fly-Over followed by Track to a Fix (FO+TF).

(b) FO+TF with a Fly-Over/Direct to a Fix capable system.

(c) FO+TF with a Fly-By/Track to a Fix capable system.

Fig. 14 Waypoint generation depending on autopilot capabilities

beginning and the end of the trajectory are replaced by two fly-by ones at different positions. The details on how to carry out the actual computations for these and other cases can be found in [28].

The replacement of the first waypoint in Fig. 14c can actually be seen as taking the destination of the previous leg and moving it further along the leg's course. Therefore, the final waypoint of a given leg may actually depend on what happens in the next leg. To handle this situation, the generation process for a given leg is performed in two steps:

1. Generate waypoints for the current leg regardless of what happens next.
2. Every time a new leg is added to the waypoint list, check if the destination of the previous one needs to be corrected.

To implement execution of RF legs on an autopilot with no support for this type of turn, the system will approximate the turning maneuver with a sequence of waypoints (see examples on Figs. 9 and 10). If the autopilot supports fly-over waypoints, the generated waypoints will be placed at regular intervals on the desired path. If the system makes use of fly-by waypoints, the generated waypoints will be slightly displaced from the desired trajectory. The distance between consecutive waypoints is directly proportional to the aircraft's turning radius which, in turn, depends on the aircraft's speed and its bank angle, the latter value being a constant that characterizes the aircraft.

Holding patterns are generated by concatenation of TF and RF legs and, in that respect, are very similar to the way parametric legs are generated.

Iterative and Intersection legs represent control constructs that, by themselves, do not determine a trajectory. The former groups together a number of legs that may be executed several times, while the latter can be used to select between alternative paths.

Dealing with iterative legs implies that the waypoint queues will contain waypoints coming from different instantiations of a single leg. Moreover, iterative legs can be nested so that two given waypoints may have been created for the same iteration of an outer leg but different iterations of an inner one and vice versa. For this reason, all waypoints are tagged with context data. This enables the FPM to determine what iterations a given waypoint, and its corresponding leg instantiation, belong to. Therefore, each enqueued waypoint contains all data directly related to the waypoint, such as parent leg, latitude, longitude, etc. plus a stack of integers. Each time a new iterative leg is entered an integer value of zero is added on top of the stack. This value is incremented at each iteration and popped out when no more iterations are left.

Keeping account of the context information is crucial due to the iterative nature of the waypoint generation algorithm. Flight plan legs are taken one at a time and knowing what leg are we generating waypoints for does not suffice. It is mandatory to know what exact iterations of enclosing iterative legs have already been processed and which are the current ones, otherwise, we would not be able to tell when waypoint generation for a given iterative leg has finished. To leave a trace of the presence of the iterative leg in the waypoint queue, a fake waypoint is added. Fake waypoints are not sent to the autopilot.

When an intersection leg is found only waypoints belonging to the selected path are generated. Should the value of its governing condition change, all waypoints found in the FPM's queues from that point onwards will be discarded and new waypoints will be generated starting at the intersection.

Each time an intersection leg is found, a fake waypoint is added to the waypoint queue. In this way, when its condition changes it can easily be found. We take advantage of the control information associated to each waypoint to be able to properly restart the generation process.

6 Simulation Results

During the development of the flight plan specification language and the FPM service many different tests have been performed, but there are two main simulated mission scenarios that have been developed with a high level of detail. The first one, is the Radio Navigation Aid (navaid)

flight inspection mission, whose results are presented in this section. The other one, is a hot spot detection mission, where the unmanned aircraft performs a scanning pattern over a supposedly burned area and reacts when a potential hot spot is detected (see [33] for further details).

The use of UAS for performing flight inspections of navaids has been proposed by Ramírez et al. in [30]. This section presents the experimental results that have been obtained in a simulation environment based on FlightGear Flight Simulator [1]. Some additional details about the experiment can be found in [34].

6.1 Navaids Flight Inspection Mission

The current Air Transportation System relies on the use of navaids to provide the capability to fly, in a safe manner, even with unfavorable visibility conditions. These navaids are subject to inspections that verify the adequacy of the radio-frequency emission with reference to a standard. While some of these inspections could be conducted on the ground, others require a set of in-flight measurements.

The flight inspection is performed in coordination with the Air Navigation Service Providers (ANSPs), who provide navigation services with the inspected navaid. In large airports, where restricting traffic is extremely expensive, the flight inspection aircrafts are inserted in normal air traffic. A requirement for the flight inspection is to minimize the impact on the rest of airspace users. In a conventional inspection platform, this requirement can be satisfied thanks to the operational agility provided by a human pilot. The UAS flight inspection platform needs to provide an equivalent level of operational agility.

During the flight inspection of a navaid, different measurements of the signal must be taken following procedures as detailed in ICAO doc 8071 [16]. The acquisition of the physical magnitudes is performed flying specific trajectories, such as an orbit around the navaid, a straight line over the facility, etc. The overall set of trajectories and measurements for a specific VOR navaid is shown in Fig. 15. This figure illustrates the complexity of the simulated flight inspection mission whose results are given in the following sections. This mission corresponds to a real facility located at Huesca, Spain.

6.2 Inspection Procedures

This section presents the specific procedures for the periodic flight inspection of the VOR/DME located at Huesca, Spain. VOR/DME refers to a combined radio navigation station for aircraft, which consists of two beacons, placed together: a VHF Omnidirectional Range (VOR) and a Distance Measuring Equipment (DME). A VOR indicates the magnetic bearing from the station to the aircraft (the radial). A DME indicates the distance from the beacon to the aircraft.

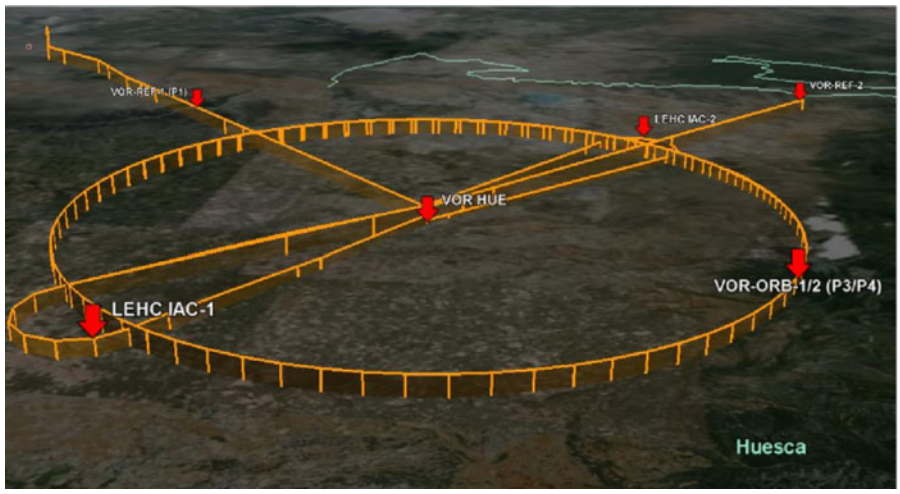


Fig. 15 VOR navaid at Huesca, Spain

In order to fulfill the mission requirements the system must be able to:

- Fly all procedures as well as or better than conventional flight inspection systems.
- Repeat any procedure or part of it if the results are not satisfactory.
- Interrupt the pre-established flight plan.
- Continue the flight inspection procedures where they were interrupted according to efficiency issues.

In a traditional inspection mission aircraft navigation is based on the inspected navaid while the actual aircraft position is obtained using a precise positioning system. Note that our inspection method inverts the terms and assumes satellite based navigation for performing the inspection. Therefore the intent is to fly the desired trajectory using RNAV legs and compare the VOR readings with the expected values.

Table 2 lists the procedures that have to be flown in order to perform a periodic flight inspection of a VOR/DME navaid [14, 16, 24]. These procedures are described in terms of three basic legs: Direct to a Fix (DF), Track to a Fix (TF) and Radius to a Fix (RF).

The procedure order is determined by two factors. Two reference radials have to be flown first because they test vital parameters. The other procedures are ordered according to efficiency. A brief description of each procedure follows.

1. *Reference Radial Flight (VOR-REF-1, VOR-REF-2):* This procedure consists of flying a VOR navaid radial at constant altitude. The main objective of this procedure is comparing vital parameters (such as magnetic deviation) with the record of previous inspections. It is flown twice because the VOR navaid has two transmitters due to redundancy aspects. Both of them transmit at the same band, hence, they have to be tested separately.
2. *Orbital Flight 360 Degrees (VOR-ORB-1, VOR-ORB-2):* This procedure is an orbital flight with constant radius. The center of the orbit is the navaid position. Its main objective is to determine if the signal coverage is between the established limits. Other parameters are also tested. Like the Reference Radial Flight, this procedure also needs to be flown twice.
3. *Radial Flights (VOR-RAD)* In order to ensure the correct reception of VOR signal, all the

Table 2 Procedures for a periodic flight inspection

Procedure	Start	Finish	Height	Leg types
Reference radial flight	20 NM away from aid	Aid	1500 ft (AGL[a]) or minimum safe altitude	TF
Orbital flight	Anywhere in the aid centered orbit	Overlapping area between 5–20 degrees from the initial point	Same as reference radial	RF
Terminal radial flight	Published maximum range	Aid	100 ft below published altitude	TF
En-route radial flight	As published in the AIP[b]	Minimum altitude published in the AIP or 1000 ft above minimum obstacle clearance altitude		TF
Approach	As published in the AIP	As published in the AIP	As published in the AIP	TF, RF
SID,[c] STAR[d]	As published in the AIP	As published in the AIP	As published in the AIP	TF, DF

[a]AGL: above ground level

[b]AIP: aeronautical information publications

[c]SID: standard instrumental departure

[d]STAR: standard terminal arrival route

VOR radials that are used to define airways shall be tested by flying them 100 ft below the specified altitude (terminal radial) or at the minimum secure altitude (en-route radial).

4. *Approaches (VOR-APP)* An Instrument Approach Procedure (IAP) is a type of air navigation that allows pilots to land an aircraft in reduced visibility or to reach visual conditions permitting a visual landing. In order to ensure the correct reception of VOR signal in these procedures, periodic flight inspection includes the flight of all approach procedures based in the inspected navaid.
5. *Standard Instrumental Departure (VOR-SID)* Standard Instrument Departure (SID) routes, also known as Departure Procedures (DP) are published flight procedures followed by aircraft on an IFR flight plan immediately after taking off from an airport.

6.3 Flight Plan Specification

This section describes how the previous procedures are translated to our flight plan specification language taking into account the requirements of the mission. As explained in Section 4, the flight plan is organized in a number of stages that group together legs that belong to different flight phases. In this section we focus on the *Mission* stage, which is where the inspection procedures are placed.

To encode the inspection procedures, we make use of the legs available in the specification language. For instance, a reference radial can be encoded using two Track to a Fix legs: one for placing the UAS at the beginning of the procedure and another one for executing the procedure. For the specification of orbital procedures, Radius to a Fix legs are used. It is possible that the orbital cannot be completed in a single pass. To limit the extent that needs to be flown more than once in case this happens each orbital is composed of several consecutive Radius to a Fix legs. The XML specification of the second radial followed by two RF legs that describe part of the first orbital is shown in Listing 5. A DF leg is used to connect both procedures. In a similar fashion, the rest of procedures can be specified combining and connecting the different available leg types. The result is a sequence of legs that can be flown from start to end to perform a complete execution of the inspection mission.

```
...
<leg id="VOR-REF-2-A" xsi:type="fp:TFLeg">
  <dest>
    <coordinates>42.0733 -0.3189</coordinates>
    <fly-over>true</fly-over>
    <altitude>2000</altitude>
    <speed>200</speed>
  </dest>
  <next>VOR-REF-2-B</next>
</leg>
<leg id="VOR-REF-2-B" xsi:type="fp:TFLeg">
  <dest>
    <coordinates>42.2821 -0.6684</coordinates>
    <fly-over>true</fly-over>
    <altitude>2000</altitude>
    <speed>200</speed>
  </dest>
  <next>VOR-VOID-2-A</next>
</leg>
<leg id="VOR-VOID-2-A" xsi:type="fp:DFLeg">
  <dest>
    <coordinates>42.2173 -0.4318</coordinates>
    <altitude>1470</altitude>
    <speed>200</speed>
  </dest>
  <next>VOR-ORB360R10-1-A</next>
</leg>
<leg id="VOR-ORB360R10-1-A" xsi:type="fp:RFLeg">
  <dest>
    <coordinates>42.1572 -0.1254</coordinates>
    <altitude>2000</altitude>
    <speed>200</speed>
  </dest>
  <next>VOR-ORB360R10-1-B</next>
  <center>42.0733 -0.3189</center>
  <direction>Right</direction>
</leg>
<leg id="VOR-ORB360R10-1-B" xsi:type="fp:RFLeg">
  <dest>
    <coordinates>41.9276 -0.2104</coordinates>
    <altitude>2000</altitude>
    <speed>200</speed>
  </dest>
  <next>VOR-ORB360R10-1-C</next>
  <center>42.0733 -0.3189</center>
  <direction>Right</direction>
</leg>
...
```

Listing 5 Excerpt from the flight plan specification

Since the flight inspection requirements demand supporting interruptions and restarts, some additional legs need to be added to the flight plan. In particular, an iterative leg is added to enable repetition of the inspection legs. In addition to that, an intersection leg provides an alternative path that leads to the execution of a holding pattern. This holding pattern is going to be flown when the aircraft is requested to move away from the area where it operates. The structure of the resulting flight plan is shown in Fig. 16. With these new legs, the UAS operator will be able to repeat parts of the plan and switch between execution of

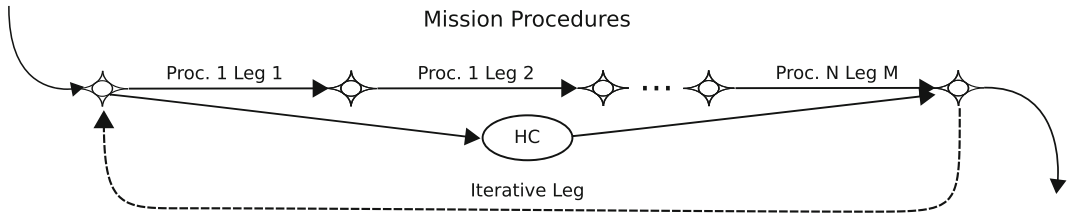


Fig. 16 Flight plan organization

the inspection procedures and the holding pattern. A *Goto* command, which is a function provided by the FPM, can be used to directly jump to the desired leg and proceed from there.

6.4 Results of the Simulation

In order to demonstrate the feasibility and benefits of the proposed approach for UAS flight management a simulation environment has been set up. The aircraft behavior is simulated using the FlightGear Flight Simulator [1], an open-source project licensed under the GNU General Public License. The simulation environment takes advantage of the VAS capability to isolate the underlying autopilot from the rest of services. With the VAS handling all interactions with FlightGear, the FPM is completely unaware of the fact that the flight is simulated. Google Earth® is used for tracking the UAS flight and provide real time visualization of the mission evolution.

Fig. 17 Beechcraft B1900D

For the purpose of the simulation, a Beechcraft B1900D (see Fig. 17) has been used. This is one of the aircraft available in FlightGear's models database and is commonly used in manned flight inspection operations. Some of the simulation parameters are shown in Table 3. A typical bank angle of 30 degrees has been considered for all turning maneuvers. A roll factor of 14 seconds accounts for the time it takes for the aircraft to reach this bank angle at the considered cruise speed. The simulation has been run without wind.

The first requirement we need to validate is that the UAS is actually capable of performing the inspection mission at the same level of performance as a manned aircraft. The resulting flight from a complete execution of the inspection procedures is displayed in Fig. 18. Labeled arrows indicate what procedure different parts of the flight belong to. This is a long flight plan and the simulation confirms that it was well defined and executed.

Following examples demonstrate the system's capability to successfully interrupt and resume the flight inspection. Figure 19 illustrates how the system responds to a situation where an Air Traffic Controller (ATC) requests the UAS to move away from the mission area until further notice. The flight trajectory in Fig. 19 is numbered to indicate the chronological order of the different steps of the maneuver. Downward arrows mark some relevant points. In "1" the UAS is initiating the flight of the *VOR-REF-1* procedure (Reference

Table 3 Aircraft and simulation parameters

Aircraft	Beechcraft B1900D
Cruise speed	230 kt (425 km/h)
Bank angle	30°
Roll factor	14 s
Wind	No wind
Mission duration	1 h 30 m aprox.

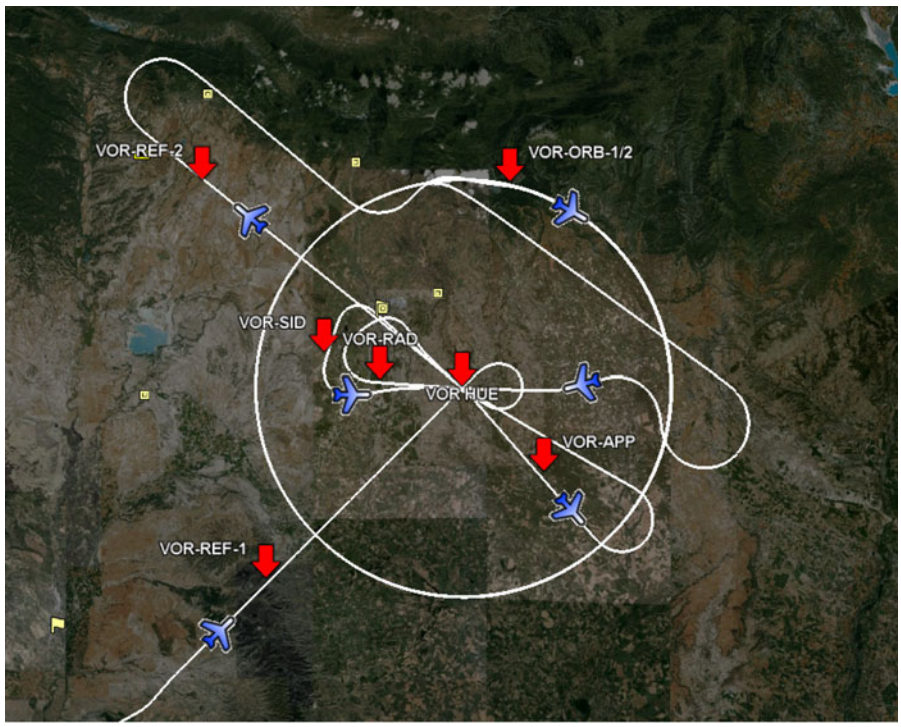


Fig. 18 Flight path of complete inspection

Radial Flight). At the *ATC Interrupt Order* arrow an interruption is requested and the vehicle leaves its trajectory to fly the agreed holding pattern (step "2"). The position of this holding procedure (*Agreed HF* arrow) can be planned with ATC before the mission execution or decided in real time sending a flight plan update to the FPM. Once the ATC decides that the inspection mission can continue, the UAS returns to the beginning of the leg that was interrupted (step "3") to perform its execution and then proceed with the rest of the flight (step "4").

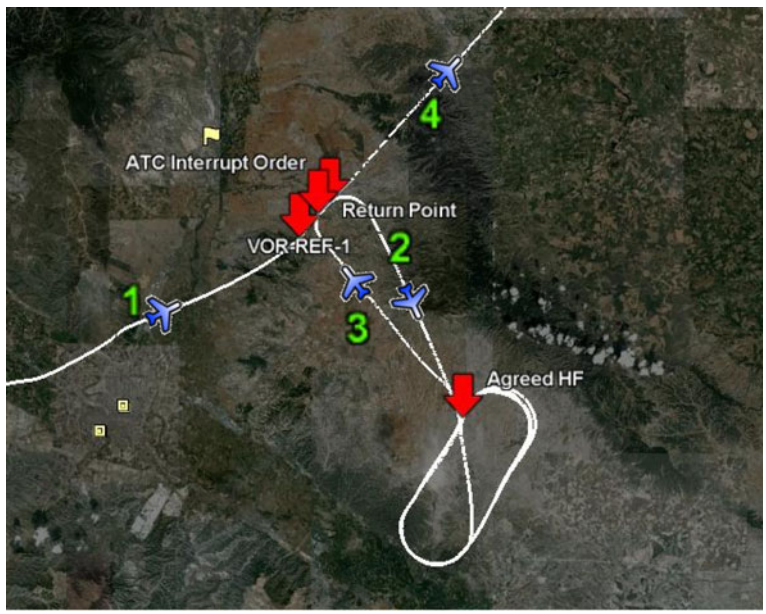


Fig. 19 ATC interruption

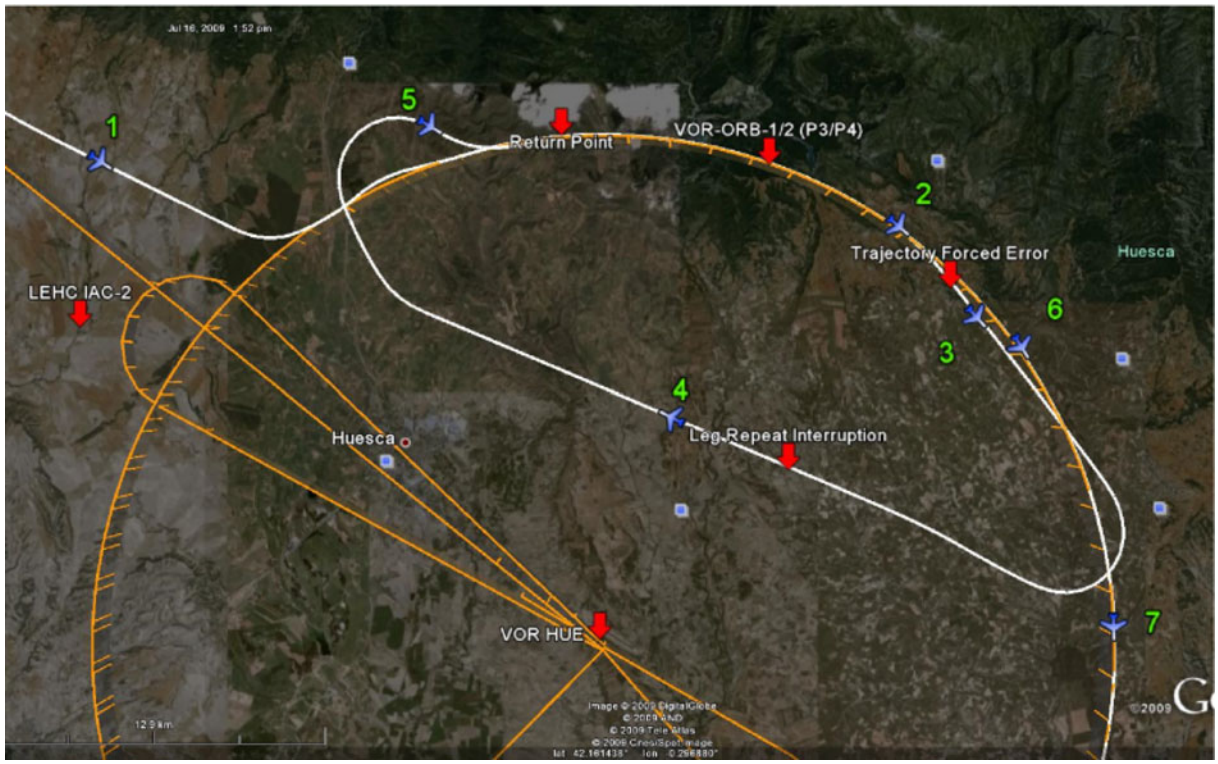


Fig. 20 Leg repetition

Figure 20 displays a situation where a leg needs to be repeated due to a deviation from the planned trajectory. This trajectory error has been manually induced for the purpose of the simulation. As shown in the figure, initially the UAS is flying to intercept *VOR-ORB-1* procedure (step "1"). It flies properly (step "2") until a trajectory error occurs (step "3"), hence measures are not going to be correct. The leg has to be interrupted and flown again. In step "4" we see how the UAS interrupts normal mission execution an flies back to a previous return point (step "5"). From there on the flight continues as planned (steps "6" and "7").

Table 4 lists the legs that are used during the mission stage of the navaid flight inspection. This information is accompanied by the number of waypoints generated by the Flight Plan Manager. The quantity of waypoints per leg depends on the nature of the leg, the capabilities of the autopilot and the type of the destination waypoint (see [28] for further details). In the simulation it is assumed that the autopilot is able to perform fly-by waypoints followed by tracks to a fix. It

Table 4 Number of waypoints for each leg of the mission phase

Procedure	Legs	Waypoints
Reference radial flights	TF × 2	6
Orbital flight	RF × 8	354
Terminal radial flight	TF	3
En-route radial flight	Covered by reference radial flights	0
Approach	TF × 3 + RF	25
SID	TF × 2 + DF	5
Holding pattern	TF × 2 + RF × 2	25
Transitions	DF × 6 + TF × 5	22
Total	TF × 13 + DF × 7 + RF × 9	440

should be noted that each of the two orbital flights around the navaid is broken up into four smaller legs so that, if the mission is interrupted during execution of an orbital, its continuation does not imply having to repeat all the parts that were already flown. Another aspect worth noting is that, although specified as a single leg in the flight plan, the holding pattern is actually translated into two tracks to a fix and two radius to a fix legs.

The benefits of the proposed approach are more evident when a given leg results in a high number of waypoints. A radius to a fix, a holding pattern and legs for performing lawnmower, eight or other patterns, the latter ones not used in the example mission, are clear examples of that. In such cases it is much easier to use a higher level abstraction, like the one provided by the leg concept, than to specify all waypoints one at a time. This benefit is more important when the flight plan needs to be changed, which could even happen during the execution of the mission. With our system, moving a holding pattern from one area to another can be achieved by changing the position of its associated waypoint.

Besides the practical aspects of the waypoint generation process, the use of the leg concept also provides a way to specify the aircraft maneuvers in a more meaningful way. A single waypoint does not provide much information about its purpose, while a leg, which defines a trajectory and can be given a descriptive name, is much more informative.

The goal of the system is not limited to facilitating the waypoint specification process. The simulated flight inspection mission shows how the inclusion of iterative and intersection legs, coupled with the capabilities of the Flight Plan Manager, provides a very flexible tool. During the simulation, we have been able to repeat parts of the flight and to respond to the hypothetical requests from an ATC in continuous contact with the UAS operator. We believe that many UAS civil applications, e.g., natural disaster monitoring, surveillance, and search and rescue operations, among others, can benefit from these features

Another aspect worth noting is the ability to include emergency alternatives in the specification of the flight plan. For a system that operates autonomously, it is important to be able to specify alternative routes to take the aircraft to a place where a reasonably safe landing can be carried out.

Some systems, e.g. the Paparazzi autopilot [7], provide capabilities comparable to the ones presented in this article, but they are tied to a particular autopilot solution. In our case, we propose a software layer that takes advantage of the waypoint navigation capabilities present in most autopilots, and extends them without requiring the autopilot to be modified. As outlined in Section 2, other significant differences relate to the general concept and the targeted platforms.

7 Conclusion

In this article a new concept for the specification of UAS flight plans has been presented. Flight plans following such specification are processed and executed by the Flight Plan Manager service. To enable the service to operate with currently available autopilot systems, flight plan primitives are dynamically translated to waypoints. The service also provides a number of flight management functions with commands for setting condition

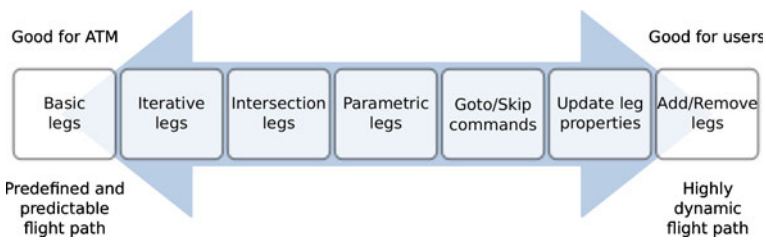


Fig. 21 Different levels of usage provided by flight plan manger

results, skipping legs and modifying the flight plan during the mission. The benefits of the proposed approach for UAS flight plan management have been demonstrated by means of a simulated mission consisting of the flight inspection of Radio Navigation Aids.

We believe that the combination of the flight plan language with the FPM, together with the other services supporting the mission, provides a highly flexible and capable platform that has the potential to be applied in a wide range of civil missions. Figure 21 outlines the main features of the system with an emphasis on the degree of flexibility they provide. At one end of the spectrum one may follow a very conservative approach and define a very predictable flight path with basic legs as the main construction unit. This can be very convenient in situations where the aircraft is operating in controlled airspace. At the other end of the spectrum, a more aggressive approach, perhaps with automated updates to the flight plan, may be followed. This way of operating can be very useful when performing a complex mission in a situation where airspace segregation guarantees no conflicts with other airspace users. Of course, both approaches can be applied to different phases of a single mission.

At its current state of development, the Flight Plan Manager should be seen as a proof of concept prototype. Aspects that need to be addressed as future work are a better model for vertical navigation, validation of update messages to ensure that the resulting flight plan can be performed and is collision free, and the inclusion of wind effects during the waypoint computation process. We also aim at implementing an enhanced model of the aircraft performance that could be used to better predict the behavior of the UAS and obtain Estimated Times of Arrival (ETA) for the different flight phases. These enhancements will enable operation of the UAS in a wider set of circumstances and provide valuable time estimations to the parties involved in its operation.

Acknowledgements This work has been partially funded by the Ministry of Science and Education of Spain under contract CICYT TIN 2010-18989.

The work presented in this paper has been performed with support from the Innovative Studies Programme of the EUROCONTROL Experimental Centre.

References

1. Flightgear Flight Simulator: http://www.flightgear.org. Last visited: May 2011
2. ARINC: Navigation system database. ARINC specification 424, 15 edn. Aeronautical Radio Inc., Annapolis, Maryland, USA (2000)
3. Avery, D.: The evolution of flight management systems. In: IEEE Software, pp. 11–13. IEEE (2011)
4. Barbier, M., Chanthery, E.: Autonomous mission management for unmanned aerial vehicles. Aerosp. Sci. Technol. **8**(4), 359–368 (2004). doi:10.1016/j.ast.2004.01.003. http://www.sciencedirect.com/science/article/B6VK2-4C5VMN4-1/2/3d5501e35d8ee30a3bf95a256bb3734b
5. Bendea, H., Boccardo, P., Dequal, S., Tonolo, F.G., Marenchino, D., Piras, M.: Low cost uav for post-disaster assessment. In: Proceedings of The XXI Congress of the International Society for Photogrammetry and Remote Sensing (2008)
6. Bray, T., Paoli, J., Sperberg-McQueen, C.M., Maler, E., Yergeau, F., Cowan, J.: Extensible Markup Language (XML) 1.1, 2nd edn. World Wide Web Consortium (W3C) (2006). http://www.w3.org/TR/xml11/
7. Brisset, P., Drouin, A., Gorraz, M., Huard, P.S., Tyler, J.: The paparazzi solution. MAV2006 (2006). http://www.recherche.enac.fr/paparazzi/papers_2006/mav06_paparazzi.pdf
8. Caveney, D., Sengupta, R.: Architecture and application abstractions for multi-agent collaboration projects. In: Proceedings of the 44th IEEE Conference on Decision and Control, and the European Control Conference (2005)
9. Chao, H., Cao, Y., Chen, Y.: Autopilots for small fixed-wing unmanned air vehicles: a survey. In: International Conference on Mechatronics and Automation (ICMA), pp. 3144–3149. IEEE, Harbin, China (2007)
10. Cloud Cap Technology: Piccolo user's guide v2.1.2 (2011). http://www.cloudcaptech.com/download/Piccolo/User%20and%20Integration%20Guides/Version%202.1.2/Piccolo%20User%27s%20Guide.pdf
11. Erdos, D., Watkins, S.: Uav autopilot integration and testing. In: Region 5 Conference, 2008 IEEE, pp. 1–6, (2008). doi:10.1109/TPSD.2008.4562731
12. FAA: Aeronautical information manual, official guide to basic flight information and ATC procedures. Federal Aviation Administration. U.S. Department of Transportation (2008)
13. Farrell, S.M., Jacques, D.R.: Waypoint generation based on sensor aimpoint. In: European Micro Air Vehicle 2009 Conference (2009). http://www.emav09.org/EMAV-final-papers/paper_58.pdf

14. Federal Aviation Administration: U.S. Department of Transportation: Aviation System Standards. Flight Inspetion Operations Group. Flight Inspetion Handbook. TI 8200.52 (2007)
15. Herndon, A.A., Cramer, M., Sprong, K.: Analysis of advanced flight management systems (fms), flight management computer (fmc) field observations trials, radius-to-fix path terminators. In: IEEE Digital Avionics Systems Conference. IEEE (2008)
16. ICAO: Manual on Testing of Radio Navigation Aids, doc. 8071, 4th edn. (2000)
17. ICAO: Procedures for Air Navigation Services - Aircraft Operations (PANS-OPS), vol. I, , 5th edn. Flight Procedures. International Civil Aviation Organisation, Montreal, Canada (2006). Doc. 8168
18. Lopez-Leones, J., Vilaplana, M., Gallo, E., Navarro, F., Querejeta, C.: The aircraft intent description language: a key enabler for air-ground synchronization in trajectory-based operations. In: Digital Avionics Systems Conference, 2007. DASC '07. IEEE/AIAA 26th, pp. 1.D.4–1–1.D.4–12, (2007). doi:10.1109/DASC.2007.4391836
19. López, J., Royo, P., Barrado, C., Pastor, E.: Applying marea middleware to uas communications. In: Proceedings of the AIAA Infotech@Aerospace Conference and AIAA Unmanned Unlimited Conference 2009. Seattle, Washington, USA (2009)
20. Mcmanus, M.I.A., Clothier, M.R., Rodney, D., Walker, A.: Highly autonomous UAV mission planning and piloting for civilian airspace operations. In: AIAC-11 Eleventh Australian International Aerospace Congress, 2005 (2005)
21. MicroPilot: Mp2028 series autopilots. http://www.micropilot.com/products-mp2028-autopilots.htm. Last visited: May 2011
22. Miller, J.A., Minear, P.D., Niessner, A.F., Delullo, A.M., Geiger, B.R., Long, L.N., Horn, J.F.: Intelligent unmanned air vehicle flight systems. In: AIAA InfoTech@Aerospace Conference (2005)
23. Murata, T.: Petri nets: properties, analysis and applications. Proc. IEEE **77**(4), 541–580 (1989). doi:10.1109/5.24143
24. NATO Research and Technology Organisation: Flight testing of radio navigation systems (les Essais en vol des systès de radionavigation) (2000)
25. Osborne, J., Rysdyk, R.: Waypoint guidance for small uavs in wind. In: Proceedings of the IEEE Conference on Decision and Control, pp. 709–714. IEEE (2006)
26. Pastor, E., López, J., Royo, P.: UAV payload and mission control hardware/software architecture. IEEE Aerosp. Electron. Syst. Mag. **22**(6), 3–8 (2007). doi:10.1109/MAES.2007.384074
27. Pastor, E., Royo, P., Santamaria, E., Prats, X., Barrado, C.: In-flight contingency management for unmanned aerial vehicles. In: Proceedings of the AIAA Unmanned...Unlimited Conference. AIAA, Seattle, Washington, USA (2009). http://hdl.handle.net/2117/6849
28. Prats, X., Santamaria, E., Delgado, L., Trillo, N., Pastor, E.: Enabling leg-based guidance on top of waypoint-based autopilots for UAS. Aerosp. Sci. Technol. (2011). doi:10.1016/j.ast.2011.09.006. http://www.sciencedirect.com/science/article/pii/S1270963811001477 Available online 24 Sept 2011, ISSN 1270-9638
29. Procerus Technologies: Kestrel user guide. Procerus Technologies, version 2.0 edn. (2008). http://www.procerusuav.com/Downloads/Manuals/Kestrel_User_Guide.pdf
30. Ramírez, J., Barrado, C., Pastor, E.: A proposal for using UAS in radio navigation aids flight inspection. In: Proceedings of the 47th AIAA Aerospace Sciences Meeting. AIAA, Orlando, Florida, USA (2009)
31. Royo, P., López, J., Pastor, E., Barrado, C.: Service abstraction layer for UAV flexible application development. In: Proceedings of the 46th AIAA Aerospace Sciences Meeting and Exhibit. AIAA, Reno, Nevada, USA (2008)
32. Santamaria, E.: Formal mission specification and execution mechanisms for unmanned aircraft systems. Ph.D. thesis, Technical University of Catalonia (UPC), Castelldefels, Catalonia, Spain (2010)
33. Santamaria, E., Barrado, C., Pastor, E., Royo, P., Salami, E.: Reconfigurable automated behavior for UAS applications. Aerosp. Sci. Technol. (2011). doi:10.1016/j.ast.2011.09.005. http://www.sciencedirect.com/science/article/pii/S1270963811001465. Available online 24 Sept 2011, ISSN 1270-9638
34. Santamaria, E., Pérez-Batlle, M., Ramírez, J., Barrado, C., Pastor, E.: Mission formalism for UAS based navaid flight inspections. In: Proceedings of the 9th AIAA Aviation Technology, Integration, and Operations (ATIO) Conference. AIAA, Hilton Head, South Carolina, USA (2009)
35. Watts, A.C., Perry, J.H., Smith, S.E., Burgess, M.A., Wilkinson, B.E., Szantoi, Z., Ifju, P.G., Percival, H.F.: Small unmanned aircraft systems for low-altitude aerial surveys. J. Wildl. Manage. **74**(7), 1614–1619 (2010). doi:10.2193/2009-425