Dynamic Flight Plan Design for UAS Remote Sensing Applications

Marc Solé*, Enric Pastor†, Marcos Perez‡, Eduard Santamaria§Cristina Barrado†
Technical University of Catalonia, Barcelona, Spain

The development of Flight Control Systems (FCS) coupled with the availability of other Commercial Off-The Shelf (COTS) components is enabling the introduction of Unmanned Aircraft Systems (UAS) into the civil market. UAS have great potential to be used in a wide variety of civil applications such as environmental applications, emergency situations, surveillance tasks and more. In general, they are specially well suited for the so-called D-cube operations (Dirty, Dull or Dangerous).

Current technology greatly facilitates the construction of UAS. Sophisticated flight control systems also make them accessible to end users with little aeronautical expertise. However, we believe that for its successful introduction into the civil market, progress needs to be made to deliver systems able to perform a wide variety of missions with minimal reconfiguration and with reduced operational costs.

Most current flight plan specification mechanisms consist in a simple list of waypoints, an approach that has important limitations. This paper proposes a new specification mechanism with semantically richer constructs that will enable the end user to specify more complex flight plans. The proposed formalism provides means for specifying iterative behavior, conditional branching and other constructs to dynamically adapt the flight path to mission circumstances. Collaborating with the FCS, a new module on-board the UAS will be in charge of executing these plans.

This research also studies how the proposed flight plan structure can be tailored to the specific needs of remote sensing. For these type of applications well structured and efficient area and perimeter scanning is mandatory. In this paper we introduce several strategies focused to optimize the scanning process for tactical or mini UAS. The paper also presents a prototype implementation of this module and the results obtained in simulations.

Nomenclature

FCS	Flight Control System
FPM	Flight Plan Manager
MMa	Mission Manager
$RNAV$	Area Navigation
$SCXML$	State Chart XML
UAS	Unmanned Aircraft System
UAV	Unmanned Aerial Vehicle
$USAL$	UAS Service Abstraction Layer
VAS	Virtual Autopilot System
XML	Extensible Markup Language

*Assistant Professor, Computer Architecture Dept., C/ Jordi Girona 1 i 3, 08034 Barcelona, Spain.
†Associate Professor, Computer Architecture Dept., C/ Esteve Terradas 7, 08860 Castelldefels, Spain.
‡Research Assistant, Computer Architecture Dept., C/ Esteve Terradas 7, 08860 Castelldefels, Spain.
§Assistant Professor, Computer Architecture Dept., C/ Esteve Terradas 7, 08860 Castelldefels, Spain.

I. Introduction

Unmanned Aerial Systems (UAS) have been developed mainly in the military field. These systems have benefited from the development of Flight Control Systems (FCS), whose research started as a weapon feature introduced in many Air Force missiles, munitions and even in ejection seats. With the consolidation of FCS technologies and the incorporation of cheap Commercial Off-The-Shelf (COTS) electronics, the usage of UAS is now extending to the civil market. The availability of stable aircraft designs and autopilot systems greatly facilitates their construction. The sophistication of existing autopilots is also making these systems accessible to end users with little aeronautics expertise. However, much work remains to be done to deliver systems that can adapt to a wide array of missions with minimal reconfiguration and with reduced operational costs. A major effort has to be put in the formalism to define the aircraft behavior for pursuing the mission goals. While in many military missions it may suffice to reach a destination avoiding any obstacle, in most civil missions the final objective is observation of a given area.[1,2] The flight itself is only a means for gathering geographical information, more similar to a satellite mission, but with several advantages for end users' needs.

The introduction of UAS is clearly a new tool for remote sensing scientific applications. It has less costs compared to other aerial vehicles like conventional aircrafts or satellites. Among the benefits of using UAS there is the improvement of quality of the acquired geographical information. This is directly related to the quality (and cost) of sensors (i.e. CCD dynamic range, number of photodetectors, pixels resolution, in-camera imaging processing software, image file formats, etc.). It is not always possible to obtain the required image resolution from a low altitude satellite like the NASA TERRA satellite. With a cheaper sensor boarded on a UAS that flies three orders of magnitude lower, the resolution achieved can improve drastically.

But UAS have also the advantage of opportunity because end users can fly them whenever and wherever they need. Despite the huge number of orbiting satellites today, depending on their payload and position, the needed information for a given physical phenomena may not be available at the exact moment it is needed. Conventional aircrafts can be a solution to opportunity but only during light time, but not during darkness. Finally, safety and cost are the main reasons for the final introduction of UAS in the civil market. In dangerous situations and in highly repetitive surveillance operations the use of UAS is the most practical, flexible and economical solution.

This paper presents the advanced flying capabilities of a UAS as an extension of the FCS functionalities. Assuming a UAS with a FCS that ensures safe and stable maneuvers, we complement it with a user defined flight plan. The flight plan is characterized by offering semantically much richer constructs than those present in most current UAS autopilots,[3] which rely on simple lists of waypoints. This list of waypoints approach has several important limitations: it is difficult to specify complex trajectories and it does not support constructs such as conditional forks or iterations, small changes may imply having to deal with a considerable amount of waypoints and it provides no mechanism for adapting to mission time circumstances. To address these issues a new flight plan specification mechanism is proposed. The flight plan is organized as a set of stages, each one corresponding to a different flight phase. Each stage contains a structured collection of legs. The leg concept is inspired by current practices in Area Navigation (RNAV[4,5]). A leg describes the path the aircraft has to follow in order to reach a given destination waypoint. The leg concept is extended to accommodate higher level constructs for specifying iterations and forks. An additional leg type, referred to as parametric leg, is also introduced. The trajectory defined by a parametric leg is automatically generated as a function of mission variables, enabling dynamic behavior and providing a very valuable means for adapting the flight to the mission evolution. Another level of adaption is provided by the conditions governing the decision-making in intersection legs and the finalization of iterative legs.

In order to process and execute the user defined flight plan, a flight plan manager module is also proposed. The flight plan manager will interact with the on-board FCS to direct the aircraft according to the prescribed flight plan. A prototype of this module has been developed and tested as proof of the validity of the concept. In this paper we will focus on the development of fire fighting missions using UAS[6–8] as the motivating mission. We believe that the flight plan specification mechanism together with the flight plan manager will facilitate both the flight plan definition and execution processes. As a result the end user will be able to focus on the mission goals, i.e. the acquisition of useful data.

The mission fragment of a remote sensing oriented flight plan should include specific parametric legs focused to improve the area and perimeter scanning process. Based on the proposed flight plan structure we will introduce a new set of scanning legs specifically oriented to facilitate the area and perimeter scan

under several types of restrictions; e.g. terrain slope, obstacles, payload limitation, etc. This paper will demonstrate that the specific needs of remote sensing can be fulfilled by using a well structured and efficient area and perimeter scanning strategy.

The organization of the paper is as follows: Section II presents the USAL network centric architecture used as system baseline. Section III describes the flight plan specification formalism executed by the flight category of services inside USAL. Section IV describes the integration of the UAS base autopilot, the improved capacities provided by the Flight Plan Management and the payload activation of the Mission Management. Section V analyzes the remote sensing scanning requirements and introduces a number of perimeter scanning strategies to be implemented by the Flight Plan Manager according to the data gathered and analyzed on board the UAS. Finally, Section VI concludes the article and identifies some future developments.

II. USAL System Overview

This section describes the UAS architecture:[9] a distributed embedded system on board the aircraft that will operate as a payload/mission controller. Over the different distributed elements of the system we will deploy software components, called services, that will implement the required functionalities. These services cooperate to accomplish the UAS mission. They rely on a middleware layer[10] for managing communications.

II.A. Distributed Embedded Architecture

The proposed system is built as a set of embedded microprocessors, connected by a Local Area Network (LAN), in a distributed and scalable architecture. This approach offers a simple scheme with a number of benefits in our application domain that motivate its selection:

- Development simplicity: Inspired by Internet applications and protocols, the computational requirements can be organized as services that are offered to all possible clients connected to the network.
- Flexibility: We are free to select the actual type of processor to be used in each LAN module. Different processors can be used according to functional requirements, and they can be scaled according to computational needs of the application. LAN modules with processing capabilities are referred to as nodes.
- Easy node interconnection: In contrast with the complex interconnection schemes needed by end-to-end parallel buses, with a LAN based approach new nodes can be added with much less effort.

This architecture accommodates a number of software components that are organized following a Service Oriented Architectures (SOA). The main goal of a SOA is to achieve loose coupling among interacting components. We refer to the distributed components as services. A service is a unit of work, implemented and offered by a service provider, that carries out some task to the benefit of a service consumer. This architecture provides enhanced interoperability, flexibility and extensibility of the designed system and of their individual services.

Loose coupling among interacting services is achieved employing two architectural constraints. First, services shall define a small set of simple and ubiquitous interfaces, available to all other participant services, with generic semantics encoded in them. Second, each interface shall send, on request, descriptive messages explaining its operation and its capabilities. These messages define the structure and semantics provided by the services. The SOA constraints are inspired significantly by object oriented programming, which strongly suggests that you should bind data and its processing together.

In a network centric architecture like SOA, when some service needs some external functionality, it asks the system for the required service. If the system knows of a service that offers this capability, its reference is provided to the requester. Thus the former service can act as a client and consume that functionality using the common interface of the provider service. The interface of a SOA service must be simple and clear enough to be easy to implement in different platforms, both hardware and software. The development of services and specially their communication requires a running base software layer known as middleware.

II.B. UAV Service Abstraction Layer

The UAV Service Abstraction Layer (USAL)[11] is a set of services running on top of the middleware that give support to most types of remote sensing UAV missions (see Figure 1). The USAL can be compared to

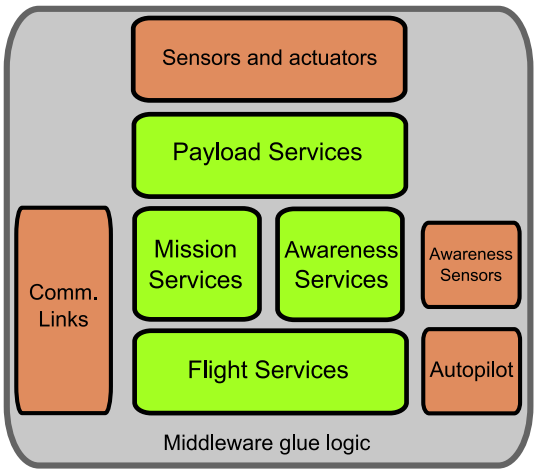

Figure 1: System architecture with services grouped according to USAL categories.

hardware drivers in an operating system. Computers have hardware devices used for input/output operations. Every device has its own particularities and the OS offers an abstraction layer to access such devices in a uniform way. The USAL treats sensors and in general all payload in a similar manner.

The USAL defines a collection of reusable services that comprises a minimum common set of elements that are needed in most UAV missions. The idea is to provide an abstraction layer that allows the mission developer to reuse these components and that provides guiding directives on how the services should interchange avionics information with each other. The available services cover an important part of the generic functionalities present in many missions. Therefore, to adapt our aircraft for a new mission it should be enough to reconfigure the services deployed to the UAV.

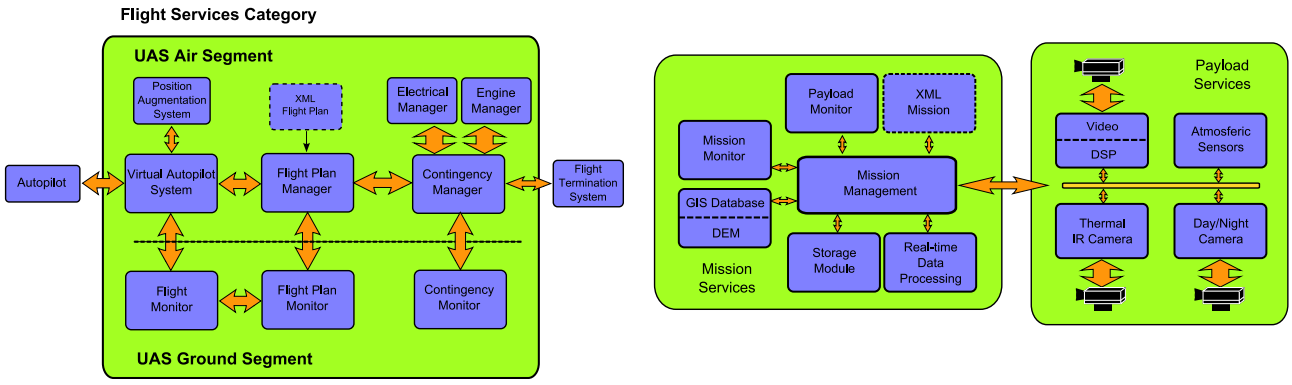

Figure 2: Detail of the USAL flight and mission service categories.

Available services have been classified into four categories:

1. Flight Services: all services in charge of basic UAS flight operations, including interaction with the autopilot, power system and engine monitoring, contingency management, etc.

2. Mission Services: all services in charge of developing the actual UAS mission, e.g. mission management and real time data processing.

3. Payload Services: specialized services interfacing with the input/output capabilities provided by the actual payload carried by the UAS.

4. Awareness Services: all services in charge of the safe operation of the UAS with respect to terrain avoidance and integration into non-segregated airspace.

There are three USAL services which are of special relevance to the work presented in this paper, namely the Mission Manager (MMa), the Flight Plan Manager (FPM) and the Virtual Autopilot System (VAS). The

FPM and the VAS belong to the Flight Services category, while the MMa belongs to the Mission Services category. As shown in Figure 3, there is a hierarchical relationship between them. The VAS is the closest to the on-board flight control system and it has two main objectives: (1) provide waypoint based navigation abstracting autopilot details from the rest of the system, and (2) extract internal sensor information from the autopilot and offer it to other services for its exploitation during the UAS mission. The FPM stands between the VAS and the MMa. It provides flight plan management capabilities for executing a leg-based flight plan. Finally, the MMa interacts with the FPM updating leg parameters and conditions contained in the flight plan to adapt the UAS trajectory to the needs of the ongoing mission.

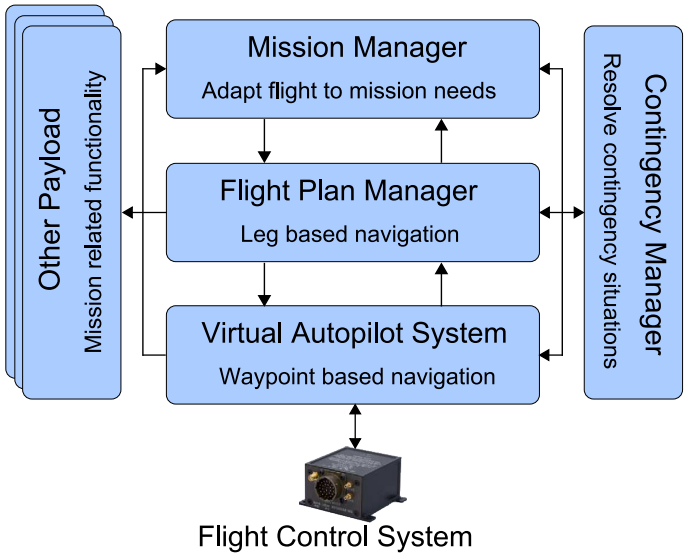

Figure 3: Hierarchical relationship between MMa, FPM and VAS.

Virtual Autopilot System The VAS directly interacts with the installed autopilot abstracting away the autopilot implementation details from its users. It also offers a number of information flows, such as UAS position, attitude, etc. to be exploited by other UAS services. Given that not all autopilots are equal, the VAS follows a contract between the VAS as a service provider and its potential clients. This means that all the information provided by this service is standardized independently of the actual autopilot being used. From the navigation point of view, the VAS provides waypoint based navigation plus a number of states to handle the different flight phases and contingency situations.

Flight Plan Manager The flight planning capabilities of most autopilots are generally limited to simple waypoint navigation. In some cases they include automatic take-off and landing modes. From the point of view of the actual missions or applications being developed by the UAS using a simple waypoint-based flight plan may be extremely restrictive. The FPM has been developed to implement much richer flight-plan capabilities on top of the functionality offered by the autopilot. The FPM enables structured flight plan definition using RNAV-inspired legs[12] as its main construction unit. Legs are organized in different stages with built-in emergency alternatives. Additional mission oriented legs with a high semantic level like repetitions, parameterized scans, etc. are also provided. Legs can be modified by other services in the UAS by changing a number of configuration parameters without having to redesign the actual flight plan; thus truly allowing cooperation between the autopilot operation and the specific UAS mission.

Given that, in general, the real autopilot capabilities are much simpler than those available in the flight plan manager, the FPM translates the flight plan legs into a sequence of waypoints to be processed by the VAS. The VAS dynamically feeds the autopilot with these waypoints during mission time, therefore transforming the FPM into a virtual machine capable of executing flight plans. As a result, combining both the abstraction mechanism provided by the VAS and the increased flight plan capabilities of the FPM, we obtain a highly capable platform.

Mission Manager The MMa orchestrates payload operation and interacts with the FPM to adapt the aircraft flight to the mission needs. An incremental approach has been followed in the design of the UAS

architecture so that the MMa is not strictly required for operating the system. However, with its addition the UAS benefits from a much improved level of autonomy. The UAS behavior is modeled using a state based control scheme based on Harel's statecharts.[13] This model is stored in an XML document which the MMa takes as its initial input. Events received by the MMa drive the execution of the model which, in turn, generates other events targeting UAS services. This interchange of information enables the MMa to effectively control the operation of the different UAS components in a coordinated manner.

III. Flight Plan and Mission Specification Formalisms

Previous section has introduced the UAS architecture. In this section we present the formalism that is used for describing flight plans. Most current UAS autopilot systems rely on lists of waypoints as the mechanism for flight plan specification and execution. This is a very restrictive approach: it is difficult to specify complex trajectories, changes to the flight plan may imply having to deal with a considerable amount of waypoints, there is no support for conditional or iterative behavior and it does not facilitate reuse of flight plan fragments. In short, current autopilots specialize in low level flight control and navigation is limited to very basic go to waypoint commands. For these reasons a new flight plan specification mechanism has been proposed[12] that provides higher level constructs, with richer semantics, and which enables adaption to mission progress.

Some ideas are based on current practices in civil aviation industry for the specification of Area Navigation (RNAV) procedures. Aircraft equipped with suitable RNAV systems can fly routes that can be defined between arbitrary waypoints and, therefore, not necessarily placed over radionavigation aids, as required with conventional navigation. This concept is possible thanks to the on-board Flight Management Systems (FMS) that continuously compute the position of the aircraft using data from one or several sensors. These new, and flexible, routes must be defined in an *Area* properly covered by one or more types of RNAV positioning aids (or sensors). RNAV procedures are composed of a series of smaller parts called legs, which describe a path (e.g., heading, course, track, etc.) and a termination point (e.g., the path terminates at an altitude, distance, fix, etc.). To facilitate this interpretation there is a database standard published by Aeronautical Radio, Inc. (ARINC): the standard 424.[14] This document details how navigation databases for FMSs are to be coded. Our specification mechanism borrows some leg types from this standard and extends them with additional constructs. New control constructs such as iterative legs and intersection legs are added and adaptability is increased by means of parametric legs. Further details are given in the next subsections, which describe the flight plan structure and its elements.

The flight plan specification enables to create plans that can be submitted to the UAS and executed by the Flight Plan Manager in collaboration with the Virtual Autopilot System. While these components alone already provide a very capable platform, the inclusion of a mission control layer on top of them will greatly improve the autonomy of the system. Again, we aim at obtaining a flexible system able to adapt to different mission scenarios.

III.A. Flight plan

The flight plan is represented by means of an XML document that contains the navigation instructions for the UAS. This document contains the main flight plan plus a number of alternatives for emergency situations. Each one of them is composed of stages, legs and waypoints hierarchically organized as seen in Figure 4.

Stages are the largest building blocks within a flight plan. They organize legs into different phases that will be performed in sequence. Legs specify the path that the plane must follow in order to reach a destination waypoint. Several primitives for leg specification are available. A waypoint is a geographical position defined in terms of latitude/longitude coordinates. A waypoint may also be accompanied by target altitude and speed indications. Optionally, a partial flight plan to be carried out if an emergency occurs can be associated to a flight plan. This emergency plan will be superseded by emergency plans specified at stage or leg level. 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.

III.B. Stages

Stages constitute high-level building blocks for flight plan specification and are used to group together legs that seek a common purpose. They correspond to flight phases that will be sequentially executed. Table 1

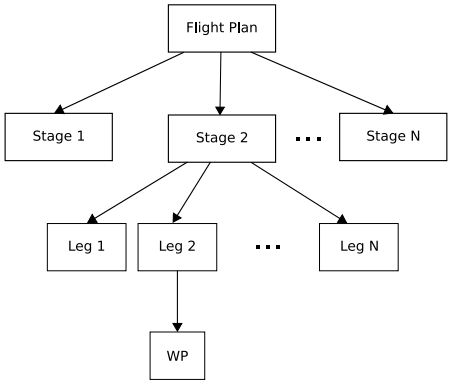

Figure 4: A flight plan is composed of stages, legs and waypoints

lists currently available stage types.

Table 1: Stage types

Taxi	Move to or return from runway.
Take-Off	Description of take off operations.
Departure	Legs flown after take off to reach the starting point of the next stage.
En-Route	Cruise to a destination area.
Mission	Series of legs that will be flown during main mission operations.
Arrival	Legs connecting the end of the route with the approach procedures.
Approach	Prepare for landing.
Land	Landing operation.

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 direction. 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 chosen take-off direction. There will be a one-to-one correspondence between the final legs of a given 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.

III.C. Legs

A leg specifies the flight path to get to a given waypoint. In general, legs contain a destination waypoint and a reference to their next. Most times legs will be flown in a single direction, but within iterative legs (see section III.C.2) reverse traversal is also supported. In this case a reference to the previous leg will be present too. Only intersection legs, which mark decision points, are allowed to specify more than one next and previous legs.

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 legs which end at the same waypoint, or a forking point where a decision on what leg to fly next can be made.
- Parametric legs: Specify legs whose trajectory can be computed given the parameters of a generating algorithm, e.g. a scanning pattern.

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 plan is to be carried out when an emergency occurs.

III.C.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.

- Initial Fix: 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: Corresponds to a straight trajectory from waypoint to waypoint. The initial position is the destination waypoint of the previous leg.
- Direct to a Fix: 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: 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 Pattern: Specifies a holding pattern path. There are three kinds of holding patterns which differ in how they are terminated. Hold to an Altitude terminates when a given altitude is reached. A Hold to a Fix is used to define a holding pattern path, which terminates at the first crossing of the hold waypoint after the entry procedure has been performed. The final possible type is the Hold to a Condition. In this case the holding pattern will be terminated after a given number of iterations or when a given condition is no longer satisfied (regardless of the number of iterations).

III.C.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. Every time the final leg is executed an iteration counter will be incremented. When a given count is reached or an specified condition no longer holds the leg will be abandoned proceeding to the next one.

Figure 5 shows the two different possibilities for iterative leg specification. Diagram 5a displays the case when holds to a fix are used to reverse the aircraft course and cycle back and forth. After entering the iterative leg, the legs forming its body are executed. Then a hold to a fix is found which reverses the aircraft direction. Now the body legs can be executed again, but in reverse direction, until another hold to a fix is found. In the holding patterns, the solid line represents the path followed by the aircraft in order to perform the turning maneuver. This back and forth behavior is only allowed when it is possible to obtain the inverse of all legs involved. Diagram 5b shows a simpler case when the legs of the body are executed one after another in a single direction.

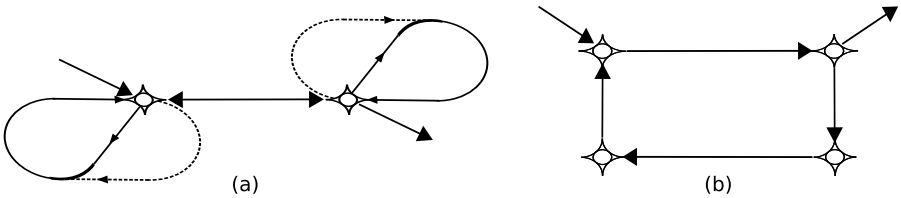

Figure 5: Iterative leg types.

III.C.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 figure 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.

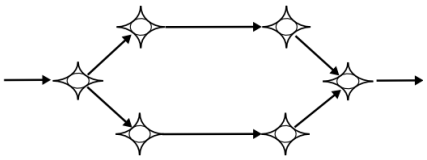

Figure 6: Intersection legs.

III.C.4. Parametric Legs

With parametric legs complex trajectories can 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. Figure 7 shows a number of possible patterns for exploring a given area, as in (a) and (b), or a more specific point, as in (c) that can be obtained using parametric legs.

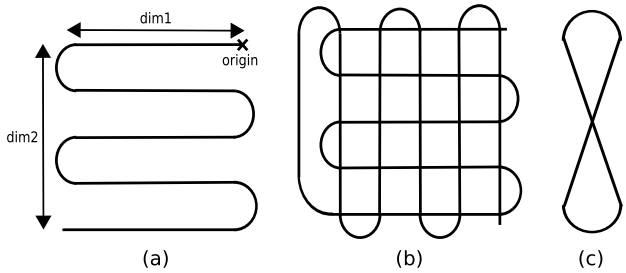

Figure 7: Scanning patterns.

Eventually a complete enough library of different parametric legs will be available so that a wide range of missions can be performed. 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 UAV path can dynamically adapt to the mission requirements.

III.D. Conditions

There are several points in the flight plan where conditions can be found: namely in holding patterns, iterative legs and intersection legs. For intersection legs, they are necessary in order to determine what path to follow next. For the rest of legs they will let the FPM know when to leave the current leg and proceed to the next one.

Conditions are not directly specified in the flight plan. Instead, each leg that depends on a condition contains an identifier, which is used to refer to the condition, and sometimes also a default value. Conditions are processed separately and when a condition is given a result, i.e. an integer value telling which is the selected path, the FPM dynamically recomputes affected waypoints.

III.E. Mission Modeling

The description of the UAS mission we propose the use of State Chart XML (SCXML).[15] SCXML is a working draft published by the World Wide Web Consortium, it provides a general purpose event-driven state machine language based on Harel's statecharts.[13] Statecharts can be used to model the behavior of a system by means of a finite number of states, transitions between those states, and actions. Statecharts extend traditional state machine diagrams with support for hierarchy, concurrency and communication. They enable modular descriptions of complex systems, provide constructs for expressing concurrency and a broadcast mechanism for communication between concurrent components. Statecharts are part of the Unified

Modeling Language (UML),[16,17] which is a widespread graphical modeling language used in industry and academia.

III.E.1. State Diagrams

A state diagram describes the behavior of a system in terms of states, events and transitions. A state reflects the current configuration of the system. A trigger that causes a transition to occur is called an event. A transition is a relationship between two states. It indicates that a system in the first state will enter the second state when a specified event occurs and the specified guard conditions are satisfied. Conditions are boolean expressions used to specify under what circumstances a given transition is permitted. Finally, transitions can be also accompanied by actions. An action specifies an executable computation that takes a very short amount of time to complete. Typical things actions are used for include causing another event to fire, updating some data structure and interact with the outside world. Fast execution times are required because ideally transitions should occur instantaneously.

Graphically, a state diagram is a collection of nodes representing states and arcs representing transitions. Each transition has a label that comes in three parts: event [guard]/action. All three parts are optional.

Harel's statecharts extend traditional state machines diagrams with support for hierarchy and concurrency. Hierarchical decomposition enables the mission designer to structure the behavior specification in smaller and more manageable pieces. The concurrency capabilities of statecharts can dramatically reduce the complexity produced by state explosion found in traditional state diagrams, where all state combinations need to be considered.

III.E.2. StateChart XML (SCXML)

SCXML is a state machine language being developed by the World Wide Web Consortium (W3C). A Working Draft[15] has been published which defines a complete specification to represent statechart diagrams. SCXML is organized in different modules, the most relevant ones are listed below:

- Core Module: contains the elements that define the basic Harel state machine. It provides elements to specify states, transitions and some executable content, e.g. raising an internal event. The set of elements which can appear as executable content is extended by other modules.
- External Communications Module: adds the capability of sending and receiving events from external entities, as well as invoking external services.
- Data Module: implements the capability of storing, reading and modifying a set of data that is internal to the state machine.

IV. Flight Plan and Mission Integration

Once the formalism used to specify the flight plan and the USAL architecture have been introduced, in this section we describe how they can be used in conjunction to carry out an example mission. Before that, for a better understanding of how the two pieces work together the interaction between the MMa and other services, with a main focus on the FPM and complementary planning services, is described. The MMa and the FPM services are respectively responsible for executing the flight plan and mission specifications, while a set of flight plan design services help computing in real time the most convenient scanning trajectories.

The example used to illustrate the flight plan and mission integration consists in the detection and analysis of potential hotspots once a wildfire has been suppressed. This task is necessary to make sure that no remaining hotspots can restart a fire. The use of UAS in this scenario can be very beneficial to firefighters because nowadays a significant number of resources must be allocated to the burned area to prevent a fire restart from happening.

To carry out the mission the UAS will fly a scanning patterns over the areas of interest, including the analysis of whole fire area and its perimeter. This operation will allow the system to detect potential hotspots at the fire perimeters or analyze the overall fire area to map the level of fire impact. Later on, hot-spots or other areas of interest could be analyzed performing an eight pattern over each one of them. Once all potential hotspots have been visited, the UAS may execute a holding pattern awaiting further instructions. To carry out this mission the UAS needs camera and sensor related services for inspecting the ground surface,

data processing services to analyze the acquired data, storage services, etc. We assume that all these are available. Each time a potential hotspot is detected the MMa will receive a notification together with its coordinates. In our simplified example, which concentrates on the mission and flight plan integration, this is the only needed information.

The tactical wilfire monitoring strategy here presented illustrates how an example mission can be performed in different ways using the same flight plan / mission specification. The overall strategy includes an initial high level exploration of the areas of interest, performed in order to determine with precision the extend of the low-level detailed scans that need to be subsequently executed.

A potential version A assumes that some time consuming on-board or ground image processing is required before identifying a potential hotspot. In this case we will first complete the overall scanning operation and, afterwards, visit the potential hotspots one by one. In version B we assume that more capable on-board services are able to detect potential hotspots almost immediately. Taking advantage of that, the UAS will fly the eight pattern upon hotspot detection and then resume the scanning of the area, thus exhibiting more advanced flight control capabilities.

Figure 10 shows the main states the UAS goes through to complete the mission. This statechart is common to both versions. The differences between version A and B appear when the Mission state gets refined.

IV.A. MMa and FPM Interaction

Figure 8 shows the overall structure of the message exchange between the MMa, the FPM and the auxiliar FP design services. Coordinated operation requires the MMa to continuously listen to the FPM and keep track of the flight evolution. At the same time, the MMa must be able to generate the necessary control messages to make the FPM adapt the flight trajectory to the current mission situation. The MMa also interacts with other services to receive mission related information (e.g. a potential hotspot has been detected) and control payload operation (e.g. activate a given sensor when the Mission state is entered). From this gathered information, detailed flight plan updates can be determined so that the scanning process is optimized. Table 2 summarizes the main messages involved in this process.

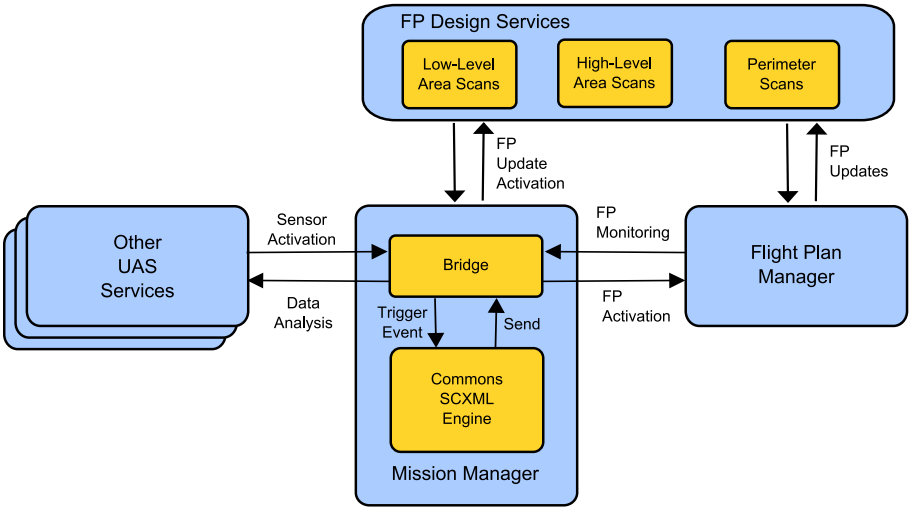

Figure 8: The MMa bridges communications between UAS services and the SCXML execution engine.

One of the advantages of using a representation which may eventually become an standard, as is the case for SCXML, is that we can benefit from already existing tools. As displayed in Figure 8, our MMa prototype makes use of Commons SCXML,[18] a library from the Apache Software Foundation that provides a generic event-driven state machine based execution environment, borrowing the semantics defined by SCXML. The MMa implements a bridge which enables communications between the UAS services and the execution engine provided by Commons SCXML. The MMa translates messages coming from the FPM, e.g. a notification that execution of a certain leg has started, to SCXML events. Symmetrically, messages sent by the execution engine, e.g. setting a new result value for a given condition, or requesting the computation of a scan update are also handled by the MMa.

Table 2: Main messages enabling MMa and FPM coordination

Direction	Message	Parameters	Purpose
FPM → MMa	currentLeg	leg id	Execution of a new flight plan leg has started.
	currentStage	stage id	Idem for flight plan stages.
MMa → FPM	setCondition	cond id, value	Set the result value of a flight plan condition.
	fpUpdate	update string	Modify the parameters of one or more legs. Update string contains modification instructions in XML form.
	gotoLeg	target leg id	Jump directly to the specified leg.
VAS → MMa	position	lon, lat, alt	UAS position.
other → MMa	hotspot	(lon, lat)	Potential hotspot location.
	perimeter	(lon, lat)*	Approximate fire perimeter
MMa → FPDS	area scan update	perimeter, leg id	High/low level area scan.
	perimeter scan update	perimeter, leg id	Low level perimeter scan.
	scan optimization	perimeter	Low-level area partition.
FPDS → FPM	fpUpdate	update string	Modify the parameters of one or more legs.
FPDS → MMa	updateConf	ack	FPM update confirmation.

We have covered only those messages that are effectively used in our example mission. The MMa will be able to interact with any service on the UAS network in a similar manner, the only restriction being that the appropriate translations must be provided. Another way to think about it is considering the MMa as a wrapper that handles all communications between the SCXML engine and the outside world. As such it determines the view of the system as seen by the SCXML engine.

IV.B. Flight Plan

The flight plan to perform the wildfire monitoring mission consists in a sequence of stages that take the UAS platform to the mission area, followed by the Mission stage and, finally, another sequence of stages to return to base. Most stages contain one or more Track to a Fix legs that describe the path for going to or returning from the mission area. The interesting part of the flight plan corresponds to the Mission stage, where the constructs that enable the adaption to the mission needs can be found. Figure 9 shows an schematic view of the Mission stage, which includes a number of different flight patterns used for the following purposes.

Within the initial Area Exploration phase:

- scanArea: Leg that follows a flight pattern that systematically scans an area determined by a interpolated sequence of ground fixes. The area is scanned at high level so that terrain is not an obstacle for the operation.
- navigation: Sequence of legs that bring the UAS from one area to be scanned to the following one.
- hold: Hold to a Fix leg used in conjunction with missLoop to wait until commanded on what to do next.

Within the initial Area Analysis phase:

- scanArea: Leg that follows a flight pattern that systematically scans an area determined by a interpolated sequence of ground fixes. The area is scanned at low level so that terrain should be considered for the operation.
- scanPerimeter: Leg that follows a flight pattern that systematically scans a perimeter determined by a interpolated sequence of ground fixes. The perimeter is scanned at low level so that terrain should be considered for the operation.
- scanPoint: Figure eight over an specific fix in order to monitor in more detail a given hotspot.

- hold: Hold to a Fix leg used in conjunction with missLoop to wait until commanded on what to do next.

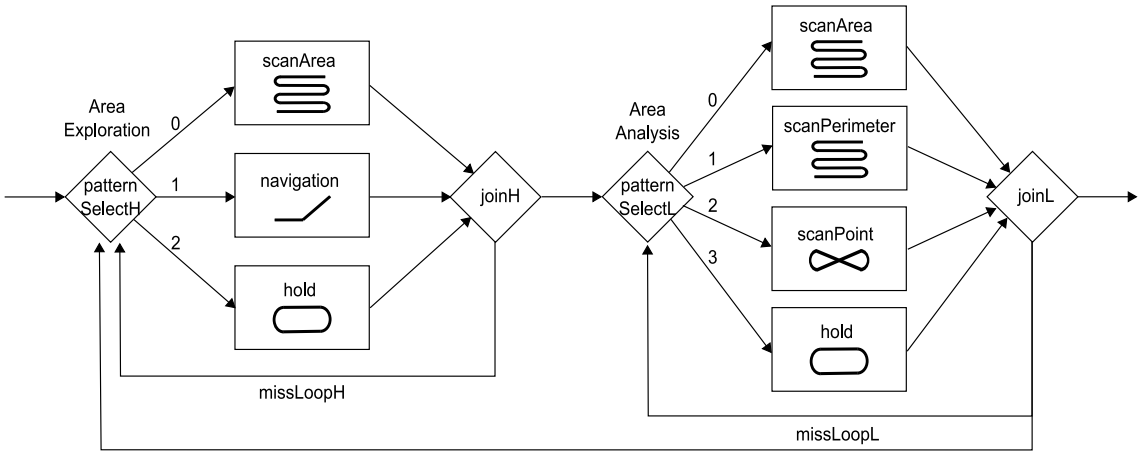

Figure 9: Organization of the legs contained in the Mission stage of the flight plan.

An intersection leg (patternSelectH / patternSelectL) is placed before these legs. The intersection leg contains a condition whose result governs the selection of the next leg to be flown. According to the numerical result of the conditions the corresponding scanning legs are selected. A second intersection leg (joinH / joinL respectively) is placed after the corresponding pattern legs to explicitly indicate that there is a single operative path from that point on. Iterative legs (missLoopH / missLoopL) are used in order to repeat this selection and to be able to alternate between the different leg types. Also note that both the intersection and the iterative legs are control structures that do not determine a flight path by themselves. With the setCondition message the MMa has control over what is going to be flown. Other means of interaction include the gotoLeg message and other commands not covered in this paper.

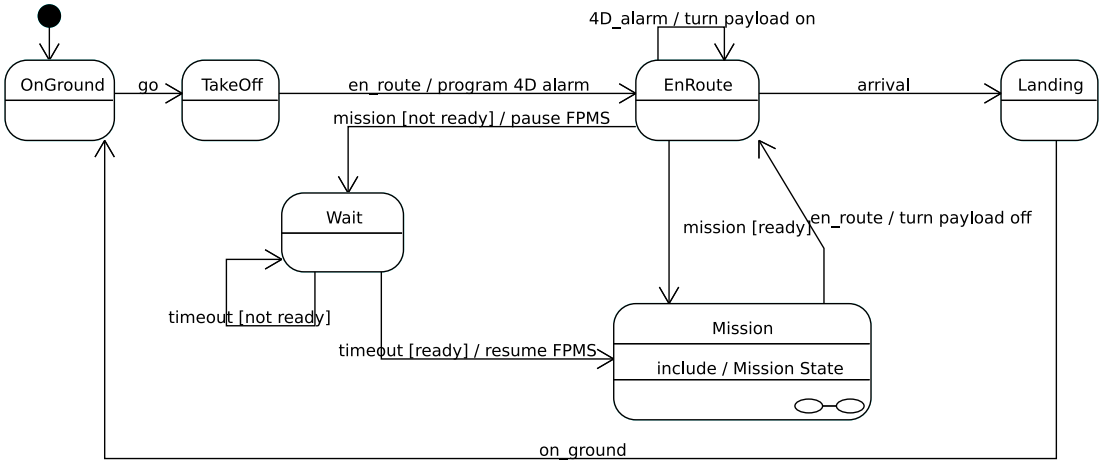

Figure 10: State chart for general monitoring operation. Mission details are refined later on.

Multiple variants of the monitoring mission example can be easily constructed through the MMa infrastructure.[19,20] A possible version is to assume that some time is required for analyzing the collected samples and decide that a given location should be analyzed in more detail; i.e. some is needed to realize that the UAS has passed upon a potential hotspot. The expected behavior for this version of the mission is to fully scan the target area first and, afterwards, visit each one of the potential hotspots. An alternative approach to the mission implementation is to assume that a potential hotspot can be detected as soon as it is approached. When such detection takes place, we expect the system to change its trajectory and perform an eight pattern over the point of interest. After that, the UAS should resume the scan of the area where it was left.

V. Flight plan design for perimeter scan

Along this paper we have identified the importance of properly plan area scan and perimeter scan operations. Within the USAL architecture, the FPM and the MMa interact to provide a powerful but at the same time flexible platform to implement these types of remote sensing missions. Given that in many cases the areas and perimeters to be scanned will be highly dynamic, generating optimized scan patterns is externalized to a dedicated set of USAL services called *FP Design Services*. These services will provide the computation support to determine the optimum strategies to perform high-level and low-level area scans, low-level perimeter scans, and low-level area scan partition in case of conflict with the terrain.

This section will develop the strategies and mathematics required to determine the best possible perimeter scanning patters to be flown by the UAS and feed them into the FPM according to the information acquired by the sensors on-board the vehicle.

V.A. Basic perimeter scan formulation

- It is as easy as possible to follow by a human pilot or an autopilot.
- It is feasible given the performaces of the aircraft, e.g. the minimum turning radius.
- It ensures that all the area is covered by images shot at probe points.
- It has the smallest possible length (to minimize fuel consumption and flight time).

To simplify the discussion we initially disregard the constraints that orography might impose on the trajectory. Later on we will see that these constraints are important, since aircraft typically have limitations on the rates of climb/descent they can achieve. Moreover, fuel consumption is also affected and following the trajectory becomes a more complex task. For now we assume that the scan has to be performed on a flat scenario.

The FP service analyzing this situation must be provided with several parameters:

- w is the width of the area that must be scanned around the perimeter,
- s_w, s_l, o_w and o_l. These are distances determined by the characteristics of the thermal camera used, its mount position, the desired quality and the scanning altitude. They are, respectively the width of the scanned area when a single image is taken, its length, the width offset between the the camera position and the center of the area and the length offset (see Figure 11). Note that parameters s_w and s_l are always positive, but not o_w and o_l, since we must distinguish the situation in which the camera is to the left/right up/down of the area center. By convention we chose the positive numbers to represent that the center is to the right/above the camera, like in Figure 11(a).

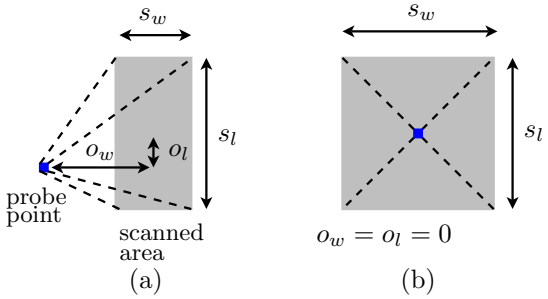

Figure 11: Possible scan areas for different cameras and mount positions.

All the area determined by the perimeter segment and the parameter w must be covered by some thermal image. Thus the problem is to find the correct probe points and define a suitable trajectory going through all of them.

To minimize the number of turns and simplify the trajectory, each area delimited by one segment of the perimeter (segment i has points p_i and p_{i+1} as limits) is scanned using straight trajectories that we call *rows*.

To obtain the number of rows r, and the number of probe points per row n, we simply divide the total area by the area covered by a single image.

$$r = \left\lceil \frac{w}{s_w} \right\rceil \qquad n = \left\lceil \frac{l_i}{s_l} \right\rceil$$

where l_i is the distance between points p_i and p_{i+1} disregarding the altitude, i.e. $l_i = \sqrt{(x_{i+1} - x_i)^2 + (y_{i+1} - y_i)^2}$. The r and n values allow computing two other important parameters. One is the distance Δ_n between consecutive centers of scanned areas, which is the same as the distance between consecutive probe points. The other is the distance Δ_r between adjacent *lanes*. A lane is the axis that traverses the center of all scan areas in a row (see Figure 14).

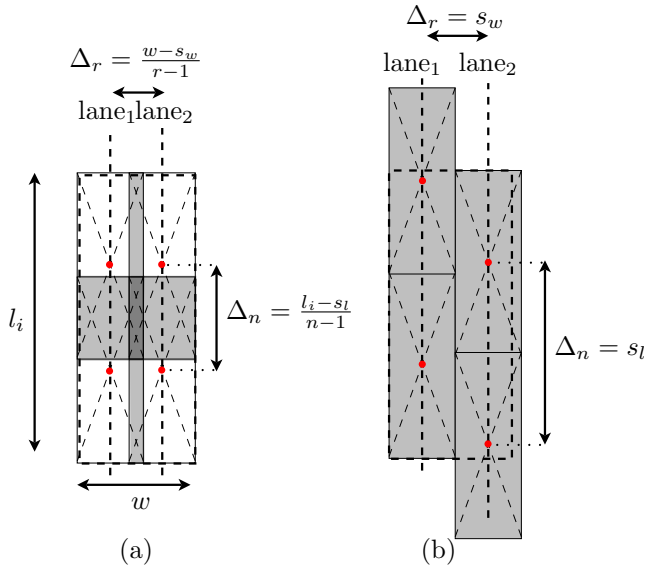

Figure 12: Overlapping policies inside a segment. (a) Distances between lanes and points is defined so that no area outside the segment frame is scanned. Overlapping areas are indicated in gray. (b) Distances defined so that overlapping does not happen, but areas outside the frame are scanned. Red dots indicate the centers of the scanned areas.

The Δ_n and Δ_r values depend on the overlapping policy adopted. For instance, one might distribute the overlapping evenly between all scanned areas –so that no area is scanned outside the segment frame, see Figure 12(a)–, or it might simply forbid the overlapping inside the fragment –and then some parts of the scanned area are outside the segment frame, see Figure 12(b).

$$\frac{l_i - s_l}{n-1} \leq \Delta_n \leq s_l \qquad \frac{w - s_w}{r-1} \leq \Delta_r \leq s_w$$

Different applications might have different preferred options, and even in the same mission preferences might change. For instance one might assign an overlapping policy for a segment considered as specially relevant, while non-overlapping would be a better option if the segment identification is suspected to be not very accurate and surrounding exploration is preferred. Note that if w is a multiple of s_w and l_i is a multiple of s_l, then both policies behave exactly the same way. In such case, if overlapping is desired then the overlapping policy must be used and n and w must be increased at will until the desired degree of overlapping is achieved.

We define the degree of overlapping as the percentage of surface scanned more than once. Horizontal overlapping is $\text{overlap}_w = \frac{l_i(r-1)(s_w - \Delta_r)}{l_i \cdot w}$. Vertical overlapping is $\text{overlap}_l = \frac{(n-1)(s_l - \Delta_n)w}{l_i \cdot w}$. Total overlapping is the sum of both overlappings minus the common zones:

$$\text{overlap}_t = \text{overlap}_l + \text{overlap}_w - \frac{(n-1)(r-1)(s_l - \Delta_n)(s_w - \Delta_r)}{l_i \cdot w}$$

As expected it becomes 0 whenever $\Delta_n = s_l$ and $\Delta_r = s_w$.

In Figure 13 we can see the scan of a single segment. Black dots indicate the perimeter points (in a 2D projection). Blue squares are the probe points. Different camera position yield different trajectories. In

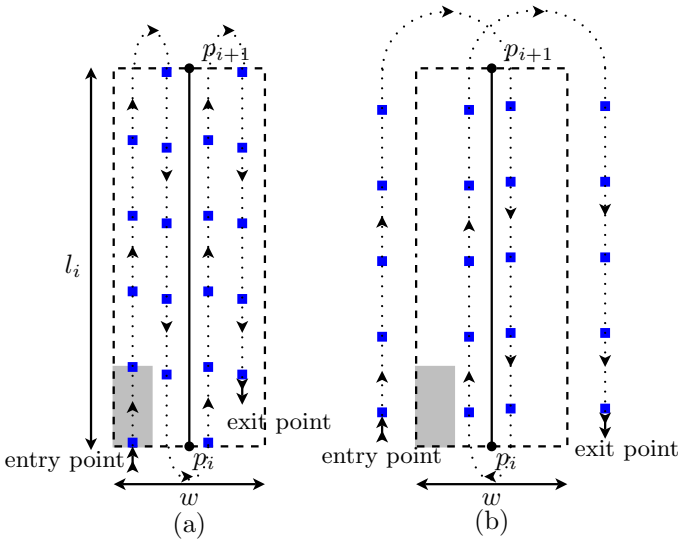

Figure 13: A perimeter scan of a single segment. The solid line is the segment delimited by points p_i and p_{i+1}. The area scanned is shown as a dashed rectangle. The trajectory is depicted as a dotted line. The grey boxes indicate the scan area associated to the first probe point. (a) The camera is frontally mounted. (b) The camera is right-mounted.

some cases this might produce too steep turns, depending on the distance between two adjacent rows. The minimum row separation is obtained from the following expression:

$$d_{\min} = \begin{cases} \Delta_r - 2\left|o_w\right|, & \text{if } 2\left|o_w\right| < \Delta_r \\ 2\left|o_w\right| - \Delta_r, & \text{otherwise} \end{cases}$$

if $r > 2$. The values are easily deduced from Figure 14.

For $r = 2$, depending on the starting orientation and the sign of the offset width it is possible that the minimum distance is, in fact, equal to $d_{\max}$. This is the case when $o_w > 0$ and the second row is on the right of the aircraft, or when $o_w < 0$ and the second row is on the left of the aircraft.

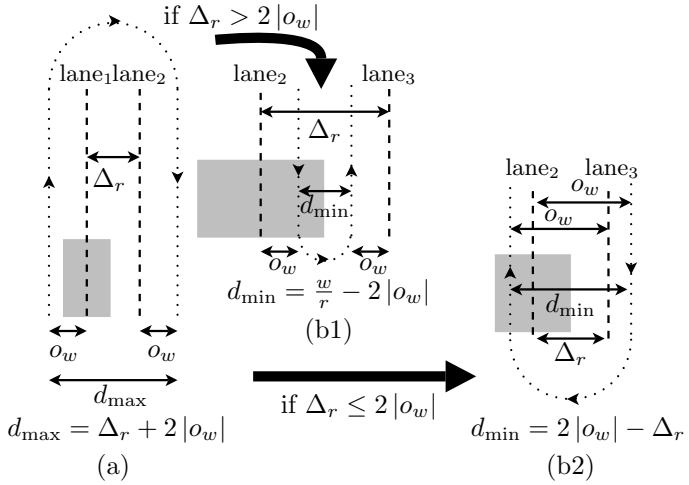

Figure 14: Distances between rows. If $o_w > 0$ and the scan is performed from left to right, then the first distance between rows is as shown in (a). The distance from the second to the third row depends on the values of the parameters, and two options are possible (b1, b2).

As seen in sectionIII, a RNAV capable flight control system is considered in this work. In particular we suppose that our flight plan will be formed by sequences of Track to a Fix (TF) and Radius to a Fix (RF)

legs. We use TF legs in order to define the rows. Thus, the waypoints defined by these legs –one leg per each probe point– are always Fly-Over (FO) type. We use RF legs in order to link each rows. Alternatively, we can suppose that non-RF legs are used in turns, but once a turn is performed we assume there is a length that must be used to correct the position and track for the next row. Typically the amount of error and the space required to fix the course depend on the speed of the aircraft, but we consider that either the speed is constant or that a conservative bound is given. We denote this repositioning length by o_p. The length of each row depends on the o_p parameter, as well as the camera offset o_l. Since we must connect two adjacent rows by a turn, we must keep flying until the entry point of the next row is surpassed. Considering this scenario, shown in Figure 15, the total length of each row is:

$$l_r = l_i - s_l + 2(o_l + o_p).$$

Remember that both s_l and o_p are always positive, but that o_l might be negative. Obviously previous equation is not complete, since by decreasing o_l at will, one can reduce l_r. If o_l is negative it might happen that the last probe point of the row establishes the limits to be flight (in Figure 15 the limits are imposed by the entry points of the adjacent rows). This happens when the distance from the last probe point to the frame, $l_i - (n-1)\Delta_n - \frac{s_l}{2} + o_l$, is in fact smaller than the distance from the frame to the entry point of the next row, $\frac{s_l}{2} - o_l - o_p$. Thus the complete equation that describes l_r is:

$$l_r = \begin{cases} l_i - s_l + 2(o_l + o_p) & \text{if } l_i - (n-1)\Delta_n \geq s_l - o_p - 2o_l \\ 2(n-1)\Delta_n - l_i + s_l - 2o_l & \text{otherwise} \end{cases}$$

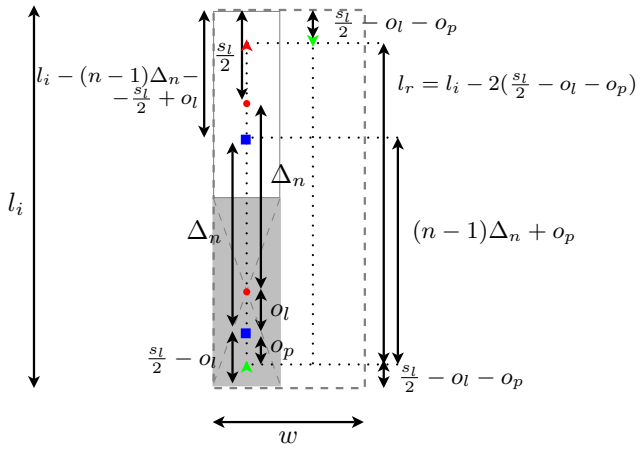

Figure 15: Length of a row (l_r). An overhead distance o_p is considered before the first probe point to allow correcting the positioning of the aircraft. After the last probe point, the plane must reposition to begin with next row. Green triangle indicates the entry point in the row, while red triangle shows the exit point.

For a single rectangular segment, the length of the trajectory is:

$$l_t = l_r \cdot r + \pi \sum_{i=0}^{r-1} t_i$$

where each t_i is the radius of the i-th turn. If we assume all turns are simple, the distance between the first two rows is $d_{\max}$, and r is even, then we can further simplify the expression:

$$l_t = l_r r + \pi \sum_{i=0}^{\frac{r-1}{2}} d_{\max} + \pi \sum_{i=0}^{\frac{r-1}{2}} d_{\min} = l_r r + \pi(d_{\max} + d_{\min})\frac{r-1}{2}$$

Substituting $d_{\max}$ and $d_{\min}$ by its definitions, we obtain:

$$l_t = l_r r + \pi(\Delta_r + 2\,|o_w| + \Delta_r - 2\,|o_w|)\frac{r-1}{2} = l_r r + \pi\Delta_r(r-1)$$

if $2|o_w| < \Delta_r$. Note that this expression for l_t is completely independent of the o_w parameter. On the other hand, if $2|o_w| \geq \Delta_r$, then the expression simplifies as follows

$$l_t = l_r r + 2\pi |o_w| (r-1)$$

that still depends on o_w.

Besides the considerations on o_w, there are also two possible additional scenarios. Let $t_{\min}$ be the minimum turning radius of the aircraft. Previous equations have assumed that $2t_{\min} \leq d_{\min}$, but still we have to compute l_t when $d_{\min} < 2t_{\min} \leq d_{\max}$ and when $2t_{\min} > d_{\max}$. When a turn is too steep, the aircraft must first open its trajectory to be able to complete the turn with a greater radius. Let $c_{\min}$ be the cost of such complex turn operation when two rows with a $d_{\min}$ separation have to be connected. Similarly, $c_{\max}$ is the cost of connecting rows separated a $d_{\max}$ distance. Since the more the turn tends to a complete circle, the more closer are the rows, we have $c_{\min} \geq c_{\max}$.

If $d_{\min} < 2t_{\min} \leq d_{\max}$, then only the turn that connect closer rows becomes a complex turn. In this case

$$l_t = l_r \cdot r + (\pi d_{\max} + c_{\min}) \frac{r-1}{2}$$

On the other hand, if $t_{\min} > d_{\max}$ then all turns are complex and

$$l_t = l_r \cdot r + (c_{\max} + c_{\min}) \frac{r-1}{2}$$

V.B. Interlaced perimeter scan

If the minimum turning radius does not allow to get directly aligned with the next row of the trajectory it is necessary to perform a more complex turn operation. This increases complexity and fuel consumption in each turn, thus it is not very efficient. A simple strategy is to skip some rows and go to the first one reachable by a simple turn. The skipped rows are visited when this first pass has finished but using a reversed order (i.e. if first pass was from left to right, the second is from right to left). In this way most of the turns are simple, except possibly the ones after each pass.

For instance, assume that the turns in Figure 13(a) are too steep for the aircraft. In Figure 16 we can see the interlaced version of the scan, in which most of the problematic turns have been removed.

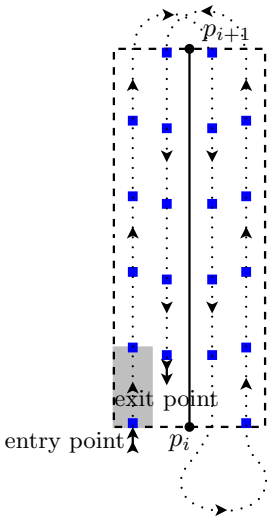

Figure 16: An interlaced perimeter scan of a single segment.

Interlacing is not always the best option. Since the length of the trajectory depends on how much displacement there is on the horizontal axis (i.e. the direction of w), and when we interlace we go back and forth this axis many times, trajectory length increases. It is only a better option if this increase is smaller than the increase introduced by $c_{\min}$ and $c_{\max}$.

Let f_i be the smallest integer such that

$$\begin{cases} f_i \frac{d_{\min}+d_{\max}}{2} \geq 2t_{\min} & \text{if } f_i \text{ is even} \\ (f_i - 1)\frac{d_{\min}+d_{\max}}{2} + d_{\min} \geq 2t_{\min} & \text{if } f_i \text{ is odd} \end{cases}$$

and $f_i \leq r-1$. We call f_i the *interlacing factor* and represents the minimum number of rows skipped in each turn to avoid a complex turn. It is also the number of passes that will be needed to complete the scanning of the segment.

V.C. Segment concatenation

There are two main strategies to explore a complete perimeter. Both are illustrated in Figure 17. The first one, named *segment-oriented*, is to explore each one of the segments independently, one after the other. This strategy might be useful when there are segments considered to be more interesting than others. For instance in the case of wildland fires it could be more relevant to check in-depth the segments where less resources are available and a fire revival is potentially more dangerous.

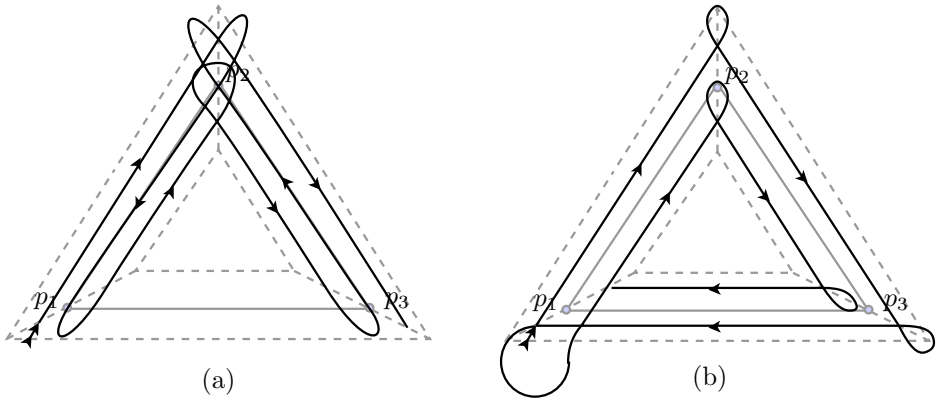

Figure 17: (a) Segment-oriented exploration policy (using trapezoid connections). For readability only the trajectory for the first two segments is shown. Although two rows could cover the surface, in order to obtain a suitable exit point in each segment, three rows are used instead. (b) Row-oriented exploration policy. Only two rows are needed, but many 270° turns are required to reach the entry point of each segment.

The second strategy, named *row-oriented*, is to follow the same row in all the segments until we reach the starting point, and then we change from row, until all the rows have been scanned. This other strategy is interesting in scenarios where we want a quick feedback, albeit incomplete, from all the perimeter. For instance one can scan from the outer lane to the inner lane to quickly check if some of the fronts of a fire has advanced since the perimeter was computed.

V.C.1. Segment-oriented policy

To chain several segments the number of rows must be odd to obtain an exit point in the opposite side of the scanned area with respect to the entry point. This avoids to go through the area to get to the entry point of next segment. If r is even we can increment the number of rows. However this is in fact more expensive than going directly to next area, because the speed when capturing images is usually slower than when no probing is needed. On the other hand, if images are captured the area overlapping increases, improving the analysis quality. Depending on the particular constraints of the mission each strategy has its own advantages.

There are several possible policies on how segment areas can be chained:

- **Overlapping**. Use rectangles and consider each fragment as isolated. The zone arround the perimeter points will then be scanned twice. Figure 18(b).
- **Trapezoid**. Use trapezoids. The non parallel sides are defined by the angle bisector of the two adjacent segments. Figure 18(c).

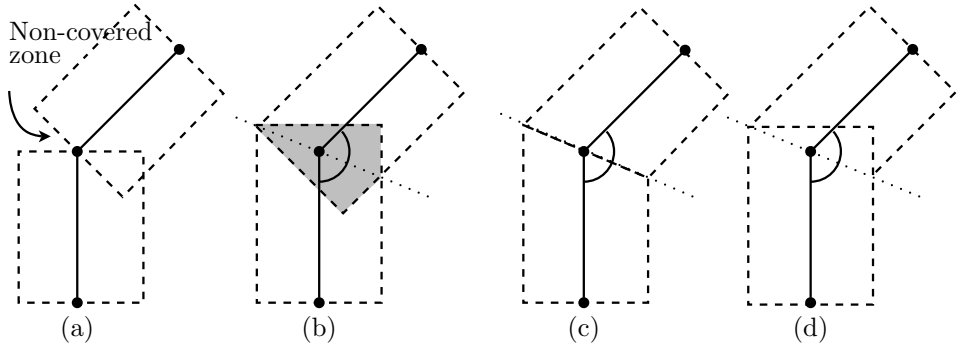

Figure 18: Chaining policies. (a) Overlapping without rectangle extension. It only guarantees to cover the inner part of the perimeter. (b) Overlapping with rectangles extended until the angle bisector. It guarantees a complete cover. (c) Trapezoid. (d) Subtractive policy.

- **Subtractive**. Use a rectangle-like shape in which one of the sides is irregular. The scanned area is subtracted from next area, and the remaining area is scanned by rows as usual. This has the advantage that overlapping is reduced, and that irregular points are only in one of the sides (this might help a human pilot because in the other side the turn always happens in the same zone). Figure 18(d).

We select the trapezoid policy because it offers the best tradeoff between overlapping and complexity.

V.C.2. Row-oriented policy

At first sight this policy seems to be more efficient. In terms of number of turns it is similar to the segment-oriented strategy, which performs $k \cdot (r-1)$ turns, where k is the number of segments. The row-oriented strategy needs $(k-1) \cdot r$ turns plus $r-1$ reposition operations to move to the next row. The advantage is that, apparently, turns are much more open than the 180° turns of the segment-oriented strategy. However, the bank in each turn plus the positioning offset o_p usually do not allow to connect rows in consecutive segments using an open turn. On the contrary, frequently a nearly complete turn is needed to accomplish the requirements that the aircraft reaches the entry point of the row leveled.

V.D. Trapezoid and arbitrarily-shaped segments

If a segment-oriented policy is used in conjunction with a trapezoid chaining of segments, then the basic scanning unit is a trapezoid. On the other hand, as we will see in the forthcoming sections, it is not unusual that the scanning of an arbitrary-shaped area is required. In this section we formulate the latter problem, so that the scan of a trapezoid reduces to a particular subcase of this general formulation.

Consider a polygon P and a vector $\vec{v}$ that defines the direction in which the scan is to be performed. To ease the presentation, assume $\vec{v}$ is always vertical. This can always be enforced by rotating the vector and the polygon P. Let w be the width of P, formally, if polygon P is formed by p_i points with coordinates (x_i, y_i), then $w = \max_{\forall i}(x_i) - \min_{\forall i}(x_i)$. Similarly to Section V, the number of rows r is obtained as $r = \left\lceil \frac{w}{s_w} \right\rceil$, so that the distance between adjacent lanes Δ_r is bounded by $\frac{w-s_w}{r-1} \leq \Delta_r \leq s_w$.

Once a Δ_r has been chosen according to the overlapping policy used, the corresponding lanes are defined following the direction of $\vec{v}$. Consider two parallel lines to each lane at distance $\frac{s_w}{2}$. Both parallels define an infinite strip intersecting P. Let $y_{\max}$ and $y_{\min}$ be the maximum and minimum y coordinates, respectively, of all perimeter points of P inside the strip. We define the length l_i of lane number i as the difference between $y_{\max}$ and $y_{\min}$, i.e. $l_i = y_{\max} - y_{\min}$.

The number of probe points in each lane, now becomes lane dependent, so that lane i requires $n_i = \left\lceil \frac{l_i}{s_l} \right\rceil$ probe points. This gives two possibilities to define the distance Δ_n between the center of consecutive scanned areas. First option is to make Δ_n lane dependent, so that lane i has distance $\Delta_n(i)$. Second option, is to use a common distance Δ_n for all lanes. However this latter option requires that

$$\max_{\forall i} \left(\frac{l_i - s_l}{n_i - 1} \right) \leq \Delta_n \leq s_l$$

and typically produces a different degree of overlapping in each lane.

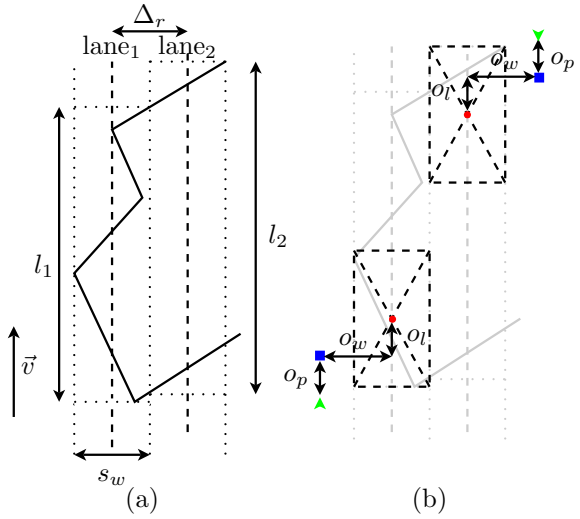

Figure 19: Scan of an arbitrary-shaped segment. Only the two first lanes are depicted. (a) Intersections between the polygon and the scanned area of each lane determines the length l_i. (b) Each scanned area center, together with the scanning sense of each lane, determines the location of its corresponding probe point. Entry points are obtained from the first probe point in each row and the o_p parameter (the offset to stabilize the plane).

Consider the centers of all the scanned area in a row. By definition these centers lie in the lane of the row and one can obtain the probe points of the row by simple addition/subtraction of the o_w and o_l parameters from the coordinates of the centers. If c is a center of a lane with coordinates (x_c, y_c) and the scanning sense is "upwards", then the corresponding probe point for c has coordinates $(x_c - o_w, y_c - o_l)$. On the contrary, if the scanning sense is "downwards", then the corresponding probe point has coordinates $(x_c + o_w, y_c + o_l)$.

V.E. Segment optimization

If one perimeter segment is small, then it is usually more efficient to avoid turns by merging this short segment with an adjacent segment. The merging process is illustrated in Figure 20. Although the resulting segment is wider than the two original ones, under appropriate conditions the overhead of this width increase is less than the cost of the removed turns.

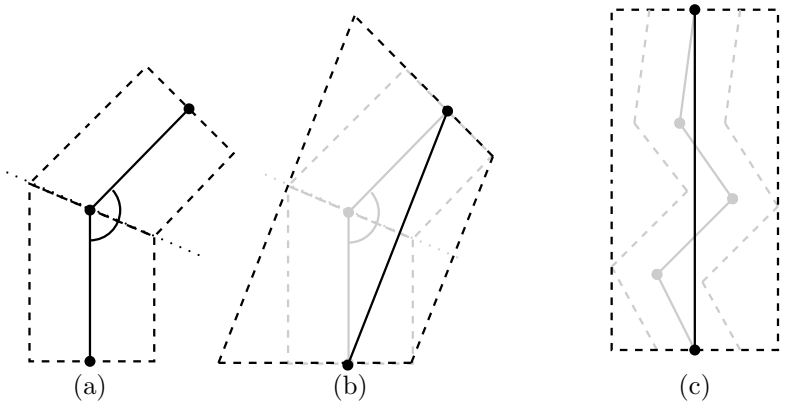

Figure 20: Merging two consecutive segments yields a wider segment but avoids around r or $2r$ turns (the exact number depends on the segment concatenation policy and the number of additional rows introduced). However, the repositioning operation to reach the entry point of either this segment or the next one is more complex than before, due to the difference in width of the adjacent segments.

The number of avoided turns is, at most, r under the row-oriented exploration policy, and $2r$ under the

segment-oriented policy. The reason we have only an upper limit is that, depending on the width increase and the overlapping policy used, it is possible that new rows have to be incorporated. For instance, if there was horizontal overlapping, so that $s_w \cdot r > w$, if new width $w' > w$ still satisfies $s_w \cdot r \geq w'$, then we can use the same number of rows to scan the new segment, by adjusting Δ_r. This reduces the degree of overlapping, but in the cases in which this factor needs to be stable or if $s_w \cdot r \ngeq w'$, then new rows have to be added.

An alternative strategy that prevents rows from being added, is to change the height of the aircraft, increasing the s_w and s_l parameters. However this approach is limited by the service ceiling of the aircraft and the desired accuracy of the pictures. Moreover two heading to altitude legs (VA) have to be incorporated before and after the segment to achieve the desired height, thus increasing the complexity of the flight and the value of o_p.

Despite the increase in complexity, in some particular situations the merging of segments is specially efficient, like when saw-like patterns appear, as in Figure 20(c). In this case, the merging process can involve many segments and the total number of avoided turns increases.

VI. Conclusion

In this paper an architecture for a UAS integrated mission management system has been described. This architecture achieves a high degree of flexibility by using XML based specifications for the flight plan and mission description. Therefore, the FPM and the MMa become engines which are able to process and execute the navigation and mission instructions included in the XML documents. The flight plan is described using a formalism specifically designed for carrying out UAS civil missions, while the mission design makes use of SCXML. The proposed system has been satisfactorily prototyped and tested in a simulation environment over a wildfire monitoring application.

Given the relevance of area and perimeter scanning during wildfire monitoring operations, this paper has proposed the design of a collection of supporting USAL services that externalize some of the required FP design operations. These services will provide optimized area and perimeter scan patters that will be feed in real time into the FPM. This paper introduces the operational and mathematic elements used to design optimized perimeter scan strategies. Various scan scheduling scenarios are described and analyzed in detail, all of them focusing on the simplification of the UAS operation and the fuel / flight time reduction. Future work will address full area scans and area partition for flight-time optimization and/or terrain avoidance.

Acknowledgments

This work has been partially funded by Ministry of Science and Education of Spain under contract CICYT TIN 2007-63927.

This work has been co-financed by the European Organisation for the Safety of Air Navigation (EUROCONTROL) under its CARE INO III programme. The content of the work does not necessarily reflect the official position of EUROCONTROL on the matter.

References

[1]NASA's Civil UAV Assessment Team, *Earth Observations and the Role of UAVs: A Capabilities Assessment. Version 1.1*, August 2006.

[2]RTCA, *DO-304: Guidance Material and Considerations for Unmanned Aircraft Systems*, March 2007.

[3]Haiyang, C., Yongcan, C., and YangQuan, C., "Autopilots for Small Fixed-Wing Unmanned Air Vehicles: A Survey," *International Conference on Mechatronics and Automation (ICMA)*, IEEE, Harbin, China, 2007, pp. 3144–3149.

[4]*Aeronautical Information Manual, Official Guide to Basic Flight Information and ATC Procedures*, U.S. Federal Aviation Administration, 2007.

[5]European Organisation for the Safety of Air Navigation, *Guidance Material for the Design of Terminal Procedures for Area Navigation*, 2003.

[6]Giglio, L., Descloitres, J., Justice, C., and Kaufman, Y., "An enhanced contextual fire detection algorithm for MODIS," *Photogrammetric Engineering and Remote Sensing*, Vol. 87, 2003, pp. 273–282.

[7]Wegener, V. A. S. and Brass, J., "The UAV Western States Fire Mission: Concepts, Plans and Developmental Advancements," *AIAA 3rd "Unmanned Unlimited" Technical Conference, Workshop and Exhibit*, AIAA, Chicago, Illinois, 2004.

[8]Casbeer, D., Kingston, D., Beard, R., and McLain, T., "Cooperative forest fire surveillance using a team of small unmanned air vehicles," *International Journal of Systems Science*, Vol. 37, No. 6, 2006, pp. 351–360, Publisher: Taylor and Francis Ltd.

[9]Pastor, E., Lopez, J., and Royo, P., "UAV Payload and Mission Control Hardware/Software Architecture," *IEEE Aerospace and Electronic Systems Magazine*, Vol. 22, No. 6, 2007.

[10]Lopez, J., Royo, P., Pastor, E., Barrado, C., and Santamaria, E., "A Middleware Architecture for Unmanned Aircraft Avionics," *ACM/IFIP/USEUNIX 8th Int. Middleware Conference*, NewPort, California, Nov. 2007.

[11]Royo, P., Lopez, J., Pastor, E., and Barrado, C., "Service Abstraction Layer for UAV Flexible Application Development," *46th AIAA Aerospace Sciences Meeting and Exhibit*, AIAA, Reno, Nevada, 2008.

[12]Santamaria, E., Royo, P., Barrado, C., Pastor, E., Lopez, J., and Prats, X., "Mission Aware Flight Planning for Unmanned Aerial Systems," *AIAA Guidance, Navigation and Control Conference and Exhibit*, AIAA, Honolulu, Hawaii, 2008.

[13]Harel, D., "Statecharts: A visual formalism for complex systems," *Science of Computer Programming*, Vol. 8, No. 3, 1987, pp. 231–274.

[14]ARINC, *Navigation System Database. ARINC specification 424.*, Aeronautical Radio Inc., Annapolis, Maryland (USA), 15th ed., Feb 2000.

[15]World Wide Web Consortium (W3C), *W3C Draft: State Chart XML (SCXML) State Machine Notation for Control Abstraction*, May 2008, http://www.w3.org/TR/scxml/.

[16]Object Management Group (OMG), *Introduction to OMG's Unified Modeling Language (UML)*, 2005, http://www.omg.org/gettingstarted/what_is_uml.htm.

[17]Booch, G., Rumbaugh, J., and Jacobson, I., *Unified Modeling Language User Guide, The (2nd Edition) (The Addison-Wesley Object Technology Series)*, Addison-Wesley Professional, 2005.

[18]The Apache Software Foundation, *Apache Commons SCXML*, 2008, http://commons.apache.org/scxml/.

[19]Santamaria, E., Barrado, C., and Pastor, E., "An Event Driven Approach for Increasing UAS Mission Automation," *AIAA Unmanned...Unlimited Conference*, AIAA, Seattle, WA, 2009.

[20]Santamaria, E., Royo, P., Barrado, C., and Pastor, E., "An Integrated Mission Management System for UAS Civil Applications," *AIAA Guidance, Navigation, and Control Conference*, AIAA, Chicago, IL, 2009.