Escola d'Enginyeria de Telecomunicació i Aeroespacial de Castelldefels
UNIVERSITAT POLITÈCNICA DE CATALUNYA

TREBALL DE FI DE CARRERA

TÍTOL DEL TFC: Mission Management for Unmanned Aircraft Systems

TITULACIÓ: Enginyeria Tècnica Aeronàutica, especialitat Aeronavegació

AUTOR: Marta Valenzuela Arroyo

DIRECTOR: Pablo Royo Chic

DATA: 4 de Juliol de 2011

Títol: Mission Management for Unmanned Aircraft Systems

Autor: Marta Valenzuela Arroyo

Director: Pablo Royo Chic

Data: 4 de Juliol de 2011

Resum

Un *Mission Manager* aporta flexibilitat, optimització i autonomia a tota missió amb vehicles aeris no tripulats (UAV). És l'encarregat d'interactuar amb les Categories de Vol i Càrrega de Pagament per a coordinar la missió. Per aconseguir-ho, executa un autòmat amb accions adjuntes.

El *Mission Manager* treballa conjuntament amb un *Mission Monitor*, una unió anomenada Categoria de Missió, per reduir la interacció entre l'home i la màquina durant el vol. A més a més, els beneficis que aporta aquesta Categoria van més enllà del dia de la missió, ja que amb la Categoria de Missió s'aconsegueix reduir el temps de disseny d'una missió.

Aquest treball fi de carrera té l'objectiu exprés de dissenyar, desenvolupar i implementar aquestes dues parts principals de la Categoria de Missió per un projecte de la UPC-ICARUS anomenat Sky-Eye.

El grup ICARUS està localitzat al Parc Mediterrani de la Tecnologia de Castelldefels i està bàsicament format per investigadors de la UPC i alguns dels seus estudiants.

Sky-Eye és un projecte subvencionat per la Generalitat de Catalunya per desenvolupar missions de inspecció de post-incendis amb UAV.

Per aconseguir els objectius, aquest treball fi de carrera ha investigat l'estat de l'art del projecte Sky-Eye, ha fet una proposta de disseny de la Categoria de Missió, la ha desenvolupada i finalment s'ha testejat la seva actuació amb simulacions fetes amb Categories que s'estan utilitzant actualment al projecte.

Els resultats que s'han obtingut podrien ser considerats excel·lents perquè les parts principals de la Categoria de Missió han estat introduïdes al projecte Sky-Eye i assoleixen tots els objectius principals que els pertoquen. Com a resultat, després d'aquest treball fi de carrera, la Categoria de Càrrega de Pagament és optimitzada i operada tenint en compte en tot moment les necessitats de l'usuari. Per la seva part, la Categoria de Vol és ara actualitzada automàticament segons les circumstàncies variants de la missió. A més, s'hi ha afegit algunes funcionalitats extres que donen un valor afegit a aquest treball, com la possibilitat de planejar una missió abans de l'actuació.

Title: Mission Management for Unmanned Aircraft Systems

Author: Marta Valenzuela Arroyo

Director: Pablo Royo Chic

Date: July, 4th 2011

Overview

A Mission Manager brings flexibility, optimization and autonomy to a mission with unmanned aircraft vehicles (UAV). It is in charge of interacting with the Flight and Payload Categories to coordinate the mission. For achieving it, it executes an automaton with attached actions.

It works conjunctly with a Mission Monitor, a union called Mission Category, to reduce the human-machine interaction during the flight. Furthermore, its benefits go beyond the day of the mission, as the Mission Category reuses efforts to reduce the mission designing time.

This project has the express objective of designing and implementing these two main parts of the Mission Category of a UPC-ICARUS project called Sky-Eye.

The ICARUS group is located in the Mediterranean Technology Park in Castelldefels and is basically formed by researchers of the UPC and some of their students.

Sky-Eye is a project subsidized for the Government of Catalonia for designing a post-fire inspection UAS mission.

In order to achieve its objectives, this final degree project has researched the State of Art of the Sky-Eye project, it has made a proposal of design of the Mission part, it has developed it and finally it has tested its performances with simulations with the Categories that are currently being used on the project.

The results obtained might be considered excellent because the main parts of the Mission Category have been introduced in the Sky-Eye's project and they fulfil all the main objectives of the Category. As a result, after this Final Degree Project, the Payload Category is optimized and operated taking into account the user needs. For its part, the Flight Category is now automatically updated according to the changing mission circumstances. Moreover, it has been added some extra functionalities that provides extra value to this project, like the possibility of planning a mission before the actuation.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ACRONIMS

ACK	*Acknowledgement*
AoI	*Area of Inspection*
AP	*Autopilot*
CC	*Camera Controller*
CMa	*Camera Manager*
COTS	*Commercial Off-The-Shelf*
CPU	*Central Processing Units*
CS	*Camera Service*
DEM	*Digital Elevation Model*
FMo	*Flight Monitor*
FP	*Flight Plan*
FPMa	*Flight Plan Manager*
FPMo	*Flight Plan Monitor*
GPS	*Global Positioning System*
ICARUS	*Intelligent Communications and Avionics for Robust Unmanned Aerial Systems*
LAN	*Local Area Network*
MAREA	*Middleware Architecture for Remote Embedded Applications*
MD	*Mission Director*
MMa	*Mission Manager*
MMo	*Mission Monitor*
MS	*Mission Setup*
MWF	*Microsoft Workflow Foundation*
PiC	*Pilot In Command*
RDC	*Reusable Dialog Components*
RTDP	*Real Time Data Processing*
SAR	*Save and Rescue*
SM	*Storage Module*
TBD	*To Be Determined*
TCP	*Transmission Control Protocol*
TO	*Take-Off*
UAS	*Unmanned Aircraft System*
UAV	*Unmanned Aerial Vehicle*
UDP	*User Datagram Protocol*
UPC	*Universitat Politècnica de Catalunya*
USAL	*UAS Service Abstraction Layer*
VAS	*Virtual Autopilot System*
XML	*Extensible Mark-up Language*

INTRODUCTION

The acronym UAS (Unmanned Aircraft System) identifies an aircraft that can fly without a pilot. That is, an airframe and a computer system, formed with sensors, a Global Positioning System (GPS), servos and a Central Processing Units (CPUs), which controls the plane with no direct human intervention.

Currently, UAS are mostly being used for military applications. However, the evolution of avionics technologies have opened up new challenges and introduced a new vision in several fields. One of them is the remote sensing applications, where the benefits of using UAS compared to those of other aircraft vehicles are remarkable. There are many ways to understand it. First of all, we could think in the quality of images that UAS bring for flying at lower altitudes compared to satellites. Another thing to take into account would be the possibilities they offer when, in Dull, Dirty and Dangerous (D^3) situations a low-cost, practical, flexible and safe operation is available. Moreover, we cannot forget about the cost. UAS, on the contrary of satellites, have no subscription fee.

Nevertheless, they still are not the perfect solution. On the market, there are not UAS prepared to serve different missions and the needs are solved with specific developments.

This problematic is being studied by the UPC research group ICARUS. The ICARUS group is subsidized for the Government of Catalonia. It is located in the Mediterranean Technology Park in Castelldefels and basically, it is formed by researchers of the UPC and some of their students.

The group is working with a project called Sky-Eye for designing a post-fire inspection UAS mission. Sky-Eye is the name for the unmanned aerial vehicle (UAV) employed, a helicopter built around existing COTS technology that can be immediately deployed at a reasonable cost. The helicopter is capable of operating from non-prepared terrains and performing surveillance flights. However, its architecture is under development.

At present, ICARUS has implemented two categories, the named Flight and Payload Category. The Flight Category refers to all the systems responsible of guiding and keeping the UAV on air while the Payload Category includes those systems that try to accomplish the UAV goals. Figure Intro.1 illustrates the operation of those categories. It can be observed an operator that it is charge of interacting with both Categories. Using ground workstations, this operator is capable of receiving information from the different Categories and of unlinking them the appropriate orders and configurations.

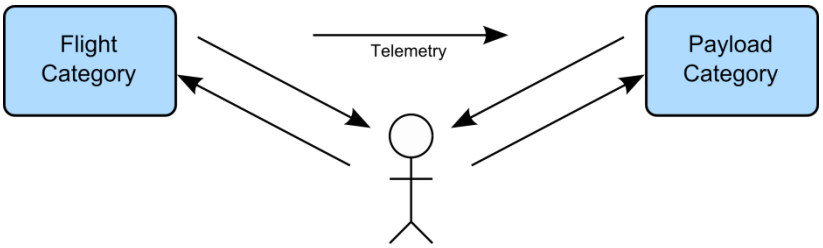


Figure Intro.1 Scenario without the Mission Category

Nonetheless, the aim of a UAS is to reduce the human-machine interaction. Therefore, it was found the necessity of adding another Category to the ICARUS architecture to achieve it, the called Mission Category.

The Mission Category is indispensable for bringing flexibility, optimization and autonomy to the system. It coordinates the overall operation supervising the Flight and Payload Categories, as can be seen in figure Intro.2. It could be said that this category acts as an orchestrate director of the others. The Mission Category executes a user-defined automaton with attached actions to achieve the reduction of the user interaction

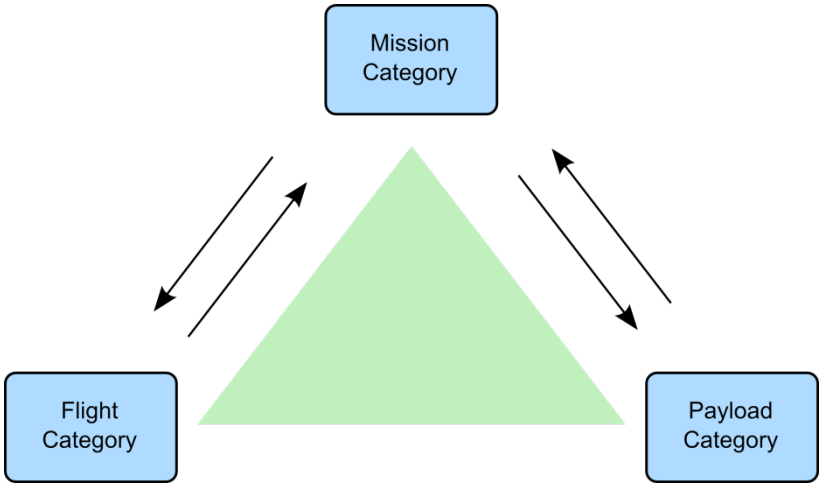


Figure Intro.2 Scenario with the Mission Category

Furthermore, the Mission Category has some additional benefits. When designing a new mission, the Mission Category reuse efforts to reduce the designing time, providing valuable flexibility as well.

The aim of this project is to design and implement the main parts of this new category into the Sky-Eye's project. In order to do so, this project has been divided into four chapters. On the first one, it will be researched the State of the Art of the Sky-Eye's Project. Then, on the second one, it will be explained the design of the Mission Category. After that, on the third, it will be described the development, to conclude, finally, with some use cases to test the Mission Category developed, on the fourth.

The results obtained might be considered excellent because the main parts of the Mission Category have been introduced in the Sky-Eye's project and they fulfil all the main objectives of the Category. As a result the Sky-Eye's project has gained in efficiency, flexibility and above all, autonomy. Before the implementations done by this final degree project, it was not possible to optimize the Payload Category taking into account the characteristics of the mission, to operate it regarding to the user needs in all moment or to adapt automatically the Flight Services to the changing mission circumstances. Now, after this Final Degree Project, the Mission Category developed converts all this into a reality and adds some extra functionalities, like planning a mission before the actuation.

MOTIVATION

The biggest motivation for doing this project was the fact that it had to be developed inside the research group ICARUS.

The group offered the possibility of researching about UAS and that was a door to make an interesting project.

On one hand, it was going to be developed inside a research department where there would be a work environment and everyone would have to cooperate with the others. In addition, it was going to be developed for a real project. Instead of researching in a field with no finalities, the Sky Eye's project has fixed objectives and its implementation is going to have practical uses.

On the other hand, it would let me deepen into the aeronautical studies. This is why UAS are faintly seen during the Aeronautical Engineering and they are an amazing field that could have huge market demands on the future.

Moreover during the Engineering we had programming subjects but they were mainly introductions. So, I found another benefit for choosing this project when I was told that I had to program. That would let me improve my programming skills, which they really interest me and are almost indispensable for any job these days.

Another important fact for having chosen this project also appeared in the introduction when talking about the market offers for UAS.

Indeed, nowadays there are not flexible solutions for different UAS missions and this is also a motivation and a challenge for doing this project: to point somewhere else when everyone is pointing in the same direction.

Finally, it is worth mentioning that the research is not related to commercial ends. On the contrary, it is focused on sensing applications that could help civil operations (e.g. forests inspection, sea rescues, etc.) to develop more efficiently their tasks.

CHAPTER 1. SKY EYE'S PROJECT STATE OF ART

Before start designing the main parts of the Mission Category we have to look to the State of Art of the UAS developed in ICARUS, as it is going to be our basis.

In order to do so, this section has been divided into two subsections: the system hardware [1] and the USAL services architecture [2].

1.1. Hardware architecture

The Sky-Eye's project needed a platform capable of overflying a forest area for certain periods of time with the capability of operating from non-prepared terrains under the following circumstances:

1. Surveillance flights during to gather information for the fire-fighters.
2. Early morning or late afternoon flight to monitor the evolution of post-fire-hot-spots during the days following the extinction.

The system proposed to accomplish the goals was called **Sky-Eye,** a UAV helicopter (figure 1.3) built around existing COTS technology that could be immediately deployed at a reasonable cost.

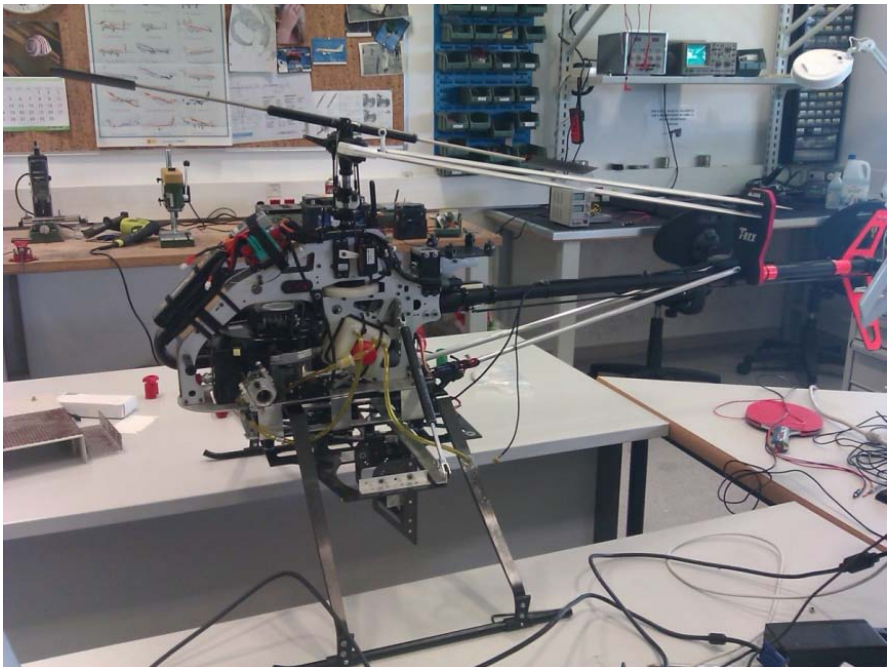


Figure 1.3 Sky-Eye's helicopter

Its architecture can be divided into three main parts:

1. **A helicopter and its computer system and on-board sensors**
   Set of sensors related to the mission. The information they provide is collected and partially processed by the on-board computer system.

2. **A mobile ground command and control station**
   A ground station managed by an operator in charge of supervising the data-gathering process and deciding actions.

3. **Communication infrastructure**
   Mixture of communication connections that must guarantee the continuous contact of the helicopter with the ground command and the control station.

1.2. USAL Service Architecture

The Mission Category is integrated among the Sky-Eye's services or USAL.

A service is a module that provisions a discrete function inside a system environment, in our case, the USAL.

USAL comes from UAS Service Abstraction Layer and it is formed by the set of services of the Sky-Eye project. The USAL links and translates the functional analysis of the UAV and the system architecture. It can be compared the USAL to an operating system. Computers have hardware devices used for input/output operations. Every device has its own particularities and the operative system offers an abstraction layer to access such devices in a uniform way. Hence, USAL is the abstraction layer for the Sky Eye's services.

Figure 1.4 shows the USAL Service Architecture of the Sky-Eye's forest inspection mission:

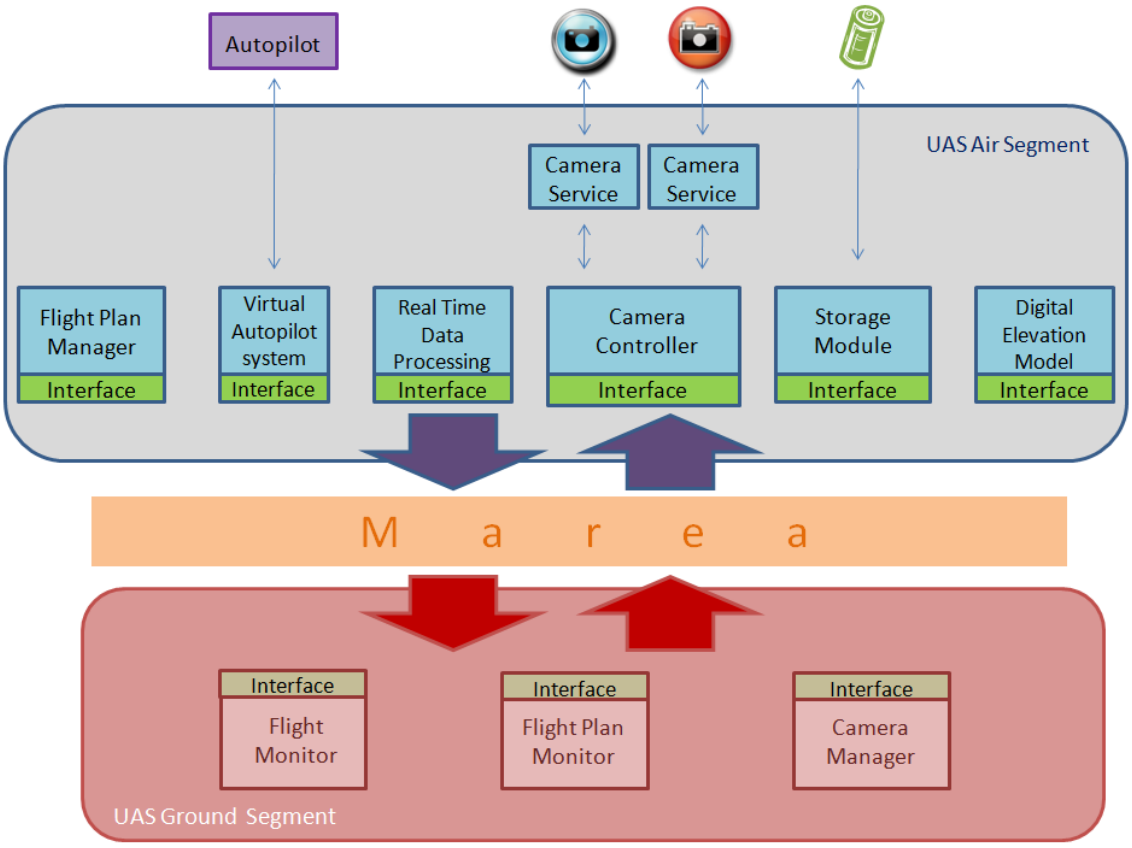


Figure 1.4 Sky-Eye's present USAL architecture

If we start from the left of the figure, it can be noticed the Flight Plan Manager (FPMa), the service responsible for processing and executing the proposed Flight Plan (FP). The FPMa has been designed to implement much richer FP capabilities on top of the available capabilities offered by the current autopilot (AP). The FPMA offers an almost unlimited number of waypoints, waypoint grouping, structured FP phases with built-in emergency alternatives, mission oriented legs with a high semantic level like repetitions, parameterized scans, etc. In addition, the FP can be introduced through Area Navigation XML formalism. Also, it is worth mentioning that all available highly semantic legs can be modified by other services by changing a number of configuration parameters without having to redesign the actual FP.

Right to the FPMa, it can be observed the VAS (Virtual Autopilot System). It receives the waypoints generated from the FPMa and interacts with the current AP to fly them. It gives flight information like angles, acceleration, rate of turn, ground speed, air speed, wind and altitude. It is especially suited to work in conjunction with the Flight Monitor (FMo) to provide information to the Pilot in Command (PiC).

Next to the VAS, there is the Camera Controller (CC). It is in charge of transmitting the order of taking photos from the Camera Manager (CMa) to the

Camera Service (CS) and to synchronize them taking into account some flight parameters. Later on the ground segment, the CMa displays the photos.

Once the photos have been taken, they are automatically stored on the on-board hard disk through the Storage Module (SM).

When enough samples have been stored, the hot-spot photo analysis starts by means of the Real-Time Data Processing (RTDP) service. This service acquires the images and carries out the hot-spots image searching with the help of the Digital Elevation Model (DEM) service, which includes the altitude parameter.

Finally, all the hot-spots coordinates detected are retrieved to the ground control station by the RTDP for individual review.

In order to link all the services and the different segments it is used what is called MAREA.

MAREA is a middleware layer that allows by means of a LAN, the communication between the running processes. These services understand each other applying an interface. MAREA allows sending messages containing variables inside containers. A container, as its name suggests, is a kind of a box where we send data. Each container is sent with an identifier. What is important to notice is that each information interchange must have the same form: first the identifier and then the data. Also it is remarkable that MAREA allows sending data by means of two protocols, the UDP and TPC protocol. Finally, just mention that all the parameters that a service wants to send or receive must be described in the services interface. Other way, there will be no connection between the service and the rest of the USAL.

CHAPTER 2. MISSION CATEGORY DESIGN

After many years of development, UAS are reaching the critical point in which they could be applied in a civil/commercial scenario. The Flight and Payload Categories may suffice for simple applications but not for more complex scenarios [3].

This lack of effective mission and payload management is studied by the Mission Category considering the following critical aspects:

- Too much human control from a ground station is still required. Flight control computers do not provide additional support beyond basic flight plan definition and operation. Additionally, payload should be also remotely operated with very little automation support.

- Economic efficiency requires the same UAS to be able to operate in different application domains. This necessity translates into more demanding requirements of the mission/payload management subsystems, with increased levels of flexibility and automation.

The two main services that compose the Mission Category are the Mission Manager and the Mission Monitor Service. The Mission Monitor (MMo) is a HMI that enables the interaction with the operator for configuring the mission and supervising it. The Mission Manager (MMa) is the core of the system. It is the one that takes action of the configured orders by interacting with other USAL services, and performing by itself mission actions. Figure 2.5 illustrates the explanation:

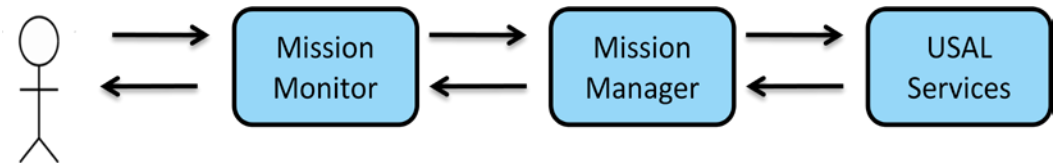


Figure 2.5 Mission Category components

Among the other services that form the Mission Category, it can be found the Mission Parameter Validator, the Mission Task Manager and the History Task Log. However as they are complimentary services and they are not fundamental to the Mission Category, for a matter of time they are not studied in this final degree project.

For explaining the design of the Mission Category, the chapter has been divided into two subsections. On the first one, it is explained the design of the MMa and in the second one, the design of the MMo.

2.1. Mission Manager Design

The concept of MMa is indispensable for achieving a flexible and optimum mission. The MMa is the responsible for supervising the Flight and the Payload services, as well as the coordination of the overall operation. It can be said that the MMa is the orchestrate director of the different services. The MMa executes a user-defined automaton with attached actions at each defined flight state or transition [4].

Hence, the main goals that MMa must fulfil are:

- Coordinate the USAL services.
- Interact between the Flight and Payload Category.
- Give mission autonomy, flexibility and optimization to the system.
- Reduce the mission planning time.

Next it is going to be explained the MMa inside the USAL architecture.

2.1.1. Mission Manager under USAL

The MMa is integrated in the Air Segment of the USAL architecture seen in figure 1.4 (see section 1.2.). With this new configuration it will be achieved the automation and efficiency required.

From the point of view of the Payload services, the main observable difference it is that now the MMa is who gives the order of taking photos to the CC, instead of the CMa. The MMa does not need to interact with any other Payload related service because the SM and the RTDP operate automatically to store the photos, analyze them and send them to ground. From the point of view of the Flight services, the MMa interacts with the FPMa to update the FP to operate accordingly to the circumstances of the mission. Again, it is the only Flight related service that the MMa interacts. This is why it is the FPMa who gives orders to the VAS and the DEM is autonomous. All the functionalities of the MMa can be found in section 2.1.4.

In order to establish the more efficient and appropriate functionalities and avoid consuming more than the exactly necessary we need to know the characteristics of the USAL systems, which are detailed in the next section.

2.1.1.1. USAL Systems characteristics

In this section it will be pointed out the main characteristics of some of the Sky-Eye's Air Segment services which could contain relevant information or limiting factors for the design of the MMa.

- **CC**

Electricity is one of the most critical parameters in a UAV and it cannot be wasted. For example, it would not be logic to establish that the cameras should be turn on when we are at 2 minutes time from the fire area if the cameras need 5 minutes to be ready.

So, the cameras should be turned on at the optimum time.

In order to do so, the following information is needed for each camera of the Sky-Eye:

Characteristic	FLIR (Thermal Camera)	Lumenera (Visual Camera)
Time to get the system ready to operate with all its characteristics once fed	30 seconds to operate and another 30 seconds to connect	Between 45 seconds and 1 minute
Optimum altitude to get the best resolution	TBD	TBD
Time between photos	0.5 seconds	At present, a little bit more than 0.5 seconds

Table 2.1 Sky-Eye's camera information

Taking into the account the information provided in this table, it would be a wise idea to turn on the cameras two minutes before arriving to the Area of Inspection (AoI). That would assure us that the cameras are connected and ready to start taking photos.

It could be also considered the maximum velocity or vibration to not get the photos moved. Fortunately, the Sky-Eye already has a solution to avoid that problem.

- **Real-Time Data Processing**

It should be known the processing time of the samples to determine the best scan pattern to perform and the optimum velocity of the UAS. This is why, for example, if our process time is high, we should not use a scan pattern where we look for some hot-spots and then, immediately, perform an eight pattern because in the meanwhile that the analysis is being processed, maybe the UAV has gone far away from the hot-spot.

Therefore, the needed information would be:

Characteristics	RTDP
Time to get the system ready to operate with all its	Almost immediate, the time that gets the code to be executed

characteristics once fed	
Time to process the analysis	The sum of all the processes carried out in the RTDP (among them loading images, segmentations, geolocations, fusions, hot-spots marking, saving processes...) needs a processing time of approximately 2.15s

Table 2.2 Sky-Eye's RTDP information

- **Flight Plan Manager, Virtual Autopilot System, Storage Module and Digital Elevation Model**

These services have been grouped because they all have the same important characteristic: they have to be always on.

Starting with the FPMA, it is the responsible for processing and executing the proposed FP and sending waypoints to the VAS, hence it is fundamental that it is on.

Regarding to the VAS, it has to be always on because apart from receiving the waypoints from the FPMa, it interacts with the current AP and delivers flight information. It could be considered a limiting factor some flight information like maximum operative altitude, maximum and minimum velocity, etc. Nevertheless, as the MMa does not interact directly with the VAS, if it happens that the MMa orders a type of scan with some parameters that cannot be performed, the FPMa would correct it.

In the case of the SM, it stores flight information and, consequently, it has to be always turned on. The maximum quantity of images that can be stored could be a limiting factor. However, images are variable in size and it is not controlled. Therefore, there will be no restrictions coming from the SM.

Finally, the DEM has to be always on because it indicates the UAV height and it could help the PiC to avoid collisions.

2.1.2. Mission Manager States

Along the document it has been mentioned several times that the MMa is in charge of coordinating the overall mission. We have to understand the concept of *mission* in detail to be able to design the MMa correctly.

It has been used two remote sensing missions for helping extracting conclusions, a forest inspection and a SAR mission [5]. What is important to realise after having studied the cases, is that a mission is formed by states. Moreover, the states that form the two missions are almost the same and they only differ in the tasks to develop, provided that they are focused on different matters. However, although the tasks are not the same, the global meaning of

the states is. Another important characteristic found is that some of the states are not fixed because their order can be changed and still the mission would be equally effective.

The conclusion that can be extracted is that even thought there are infinite state possibilities, the different remote sensing missions frequently use the same states and usually, the difference between missions resides in one simple state.

Therefore, figure 2.6 shows the states that it has been found that the remote sensing UAS missions follow. It can be observed three initial and final states and two Objectives states. The initial states are used to lead the UAV to the AoI and start performing the firsts Payload related tasks. The Objectives states are formed for a “Reconnaissance” and a “Detailed” state. These states manage Payload and Flight tasks to gather ground information. After the Objectives states, it is found the final states, which return the UAV back home and perform the last Payload related tasks.

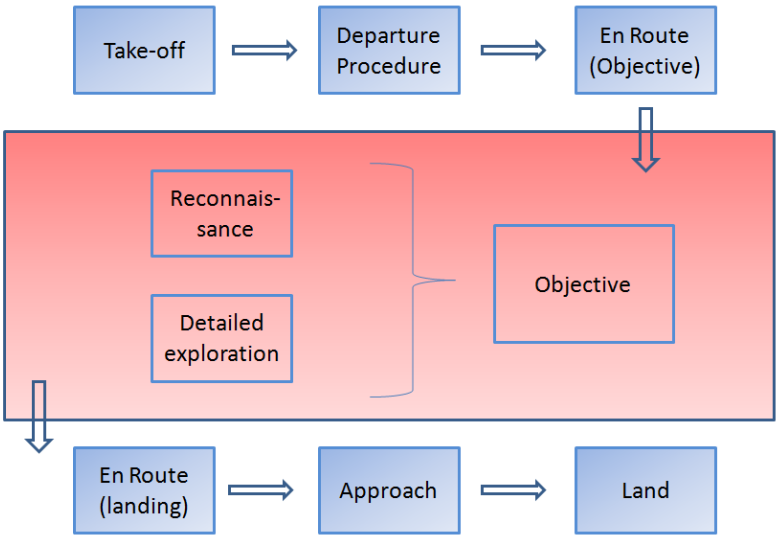


Figure 2.6 Remote sensing UAS mission

The fact that the remote sensing missions follow the same state patterns is what demonstrates that the MMa can reduce the mission design time: Why starting new projects from the beginning every time if we could just adapt the different states and requirements according to a new configuration? Therefore, the MMa proposed in this project would take into account that fact and it is going to be programmed for a future reuse of the states.

Continuing on the mission concept, now seen that it is formed by states, it is time to focus on them as it is going to be what is going to be implemented in the MMa. For the USAL architecture, a *Mission State* is a set of actions and variables that are exchanged between services for achieving one goal. The following example will explain the concept:

When a UAV has already done the take-off, and it has reached a safe altitude, the next step will possibly consist in reaching the AoI. From a flight oriented point of view, this procedure will be minimized to get to an area. However, when we think from a mission point of view, there are a set of actions to be performed in this time. In addition of getting to the zone (which is an important task), the UAV must also calibrate the systems that will be used once the area is reached and prepare itself for the first action.

Therefore, the general view of a *state* is:

- It is defined by a series of tasks, some implemented by the MMa, other delivered to other services.
- It has an entry condition (e.g. the take-off has to be over) and an exit condition (e.g. the UAS has to have reached the AoI).
- Its next state is defined (e.g. a state that begins the exploration).

The next section will explain how we can implement states in our MMa.

2.1.3. Automaton State of Art

As mentioned before the MMa implements an automaton to implement the states that will define a mission.

An automaton consists of a series of states (represented by circles or boxes), and transitions (represented by arrows). As the automaton sees a symbol of input, it makes a transition (or jump) to another state, according to its transition function (which takes the current state and the recent symbol as its inputs). Therefore, with an automaton, it is given to the MMa the necessary intelligence to be able to react to external events and adapt the UAV to the changing flight circumstances. Figure 2.7 shows an example of an automaton:

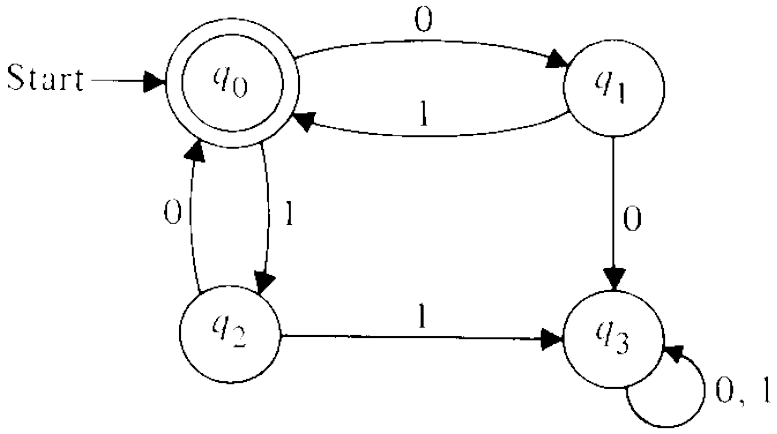


Figure 2.7 Example of automaton
http://www.lnds.net/blog/wp-content/uploads/2010/10/math-finite-state-machine-DHD.gif

The automaton proposed in this project is a **state machine**. Table 3 summarizes a little research carried out to find out what state machine fit better for the purposes of this project [6]:

Charac-teristics	Microsoft Workflow Foundation (MWF)	Petri Nets	SCXML
Design	• Flow diagrams with a high level of abstraction and visual representation, making it easy to understand. • Both sequential and state machines are possible.	• The big difference between Petri Nets and MWF is that the last one is always in exactly one state. On the contrary, in Petri nets, there can be more than one token state. The overall "state" of the Petri net is the distribution of tokens.	• It is a generic state machine execution environment based on Harel State Tables. • It basically lets the user construct state graphs and execute them.
Uses	• Any long running process which have different stages or states in their process.	• Simulation and verification of network protocols.	• Encapsulated speech modules. • Database access. • Business logic modules.
Pros	• It is easy to change the rules associated with states. • Core set activities can be reused. • Intuitive. • State machine can be persisted to a database when it becomes idle and reactivated when an external stimulus occurs. • Provides a robust, scalable environment for the execution.	• Parallel routing. • It is easy to verify its properties. • It is easy to simulate several executions. • Formal semantics. • Graphical notation. • Support for complex process constructions.	• Generalizes state diagrams notations which are already used in other XML contexts.
Cons	• More like a requirement, it is needed good level of programming skills to work with it.	• Conflict can occur between different states. • Workflow allocation.	• It is the less spread. • Some problems of robustness. • Neediness of automatic tests.

Support	• Numerous tutorials. • Examples of code.	• Community. • Forum. • Research.	• Forum. • Libraries. • Examples.
Status	• Stable. With Constant updates.	• Stable. With Constant updates.	• In development.
Debug-ger	• Yes.	• Yes.	• Yes.

Table 2.3 State of Art of state machines

Programming state machines is a relatively new programming tool in computers and given that the purpose of this short-time project is to develop a MMa which works with them, it is a wise idea to use the code which is more spread and easy to use to shorten the learning process time.

Before opting to the MWF option, which theoretically was the most appropriate, it was developed in this tool a little program that could develop some of the tasks that were required for this project. These tasks included the creation of a state machine that could change between different states when an external event occurred and the communication of the state machine and the USAL services using MAREA. The result of the test was excellent because in one week the program was finished and it developed all the tasks.

Therefore, given that MFW fit perfectly in the time schedule of this final degree project, it was intuitive and there were enough information to resolve problems, finally it was chosen to work with it.

2.1.4. Sky-Eye's state machine

Once all the pieces of a MMa have been seen, we can proceed with the architecture.

Following the remote sensing missions state pattern, figure 2.8 shows the proposed states for the MMa in the Sky-Eye's project:

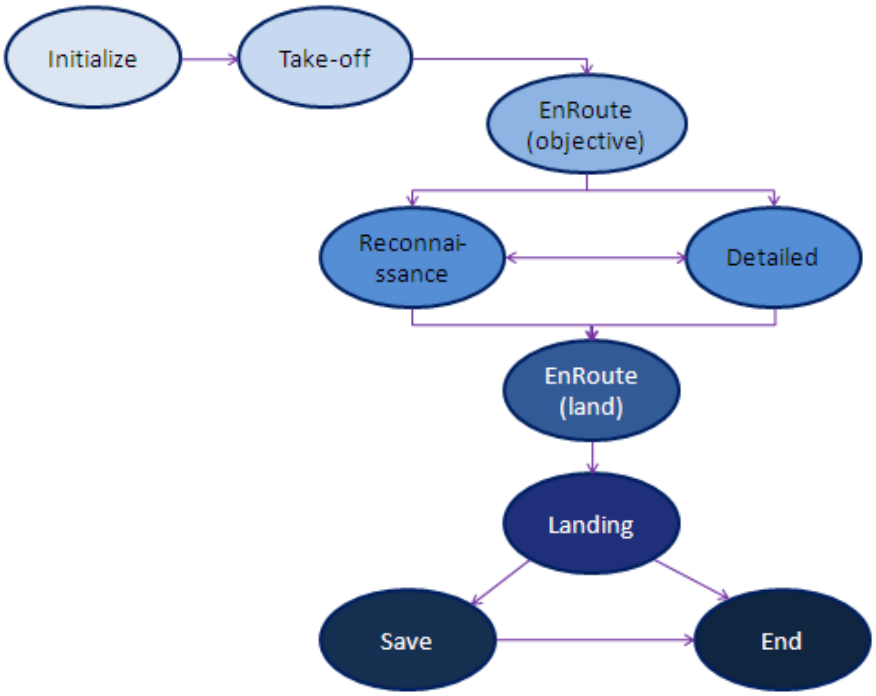


Figure 2.8 Mission state diagram

Next, it will be described the different states individually to know the main characteristics of each of them.

2.1.4.1. Initialize

This state, as its name suggests, initializes the whole mission.

Basically, it is in charge of asking the user which mission wants to develop, charging the mission parameters to the MMa, turning on the services that the user wants to use and checking if everything that is fundamental to fly is OK.

Some services will be turned on by default to assure the safety of the flight. This is the case of the DEM, the VAS, the FPMa, the SM and the MMa as explained in section 2.1.1.1.

Table 2.4 shows the characteristics of this state:

Pre-requirements	List with the services to turn on. List with the parameters to feed the MMa.
Goals	Check if everything is OK to develop the mission, ask the user which mission wants to perform, load the mission parameters, turn on USAL services and initialize the MMa.
Tasks	Turn on services and execute the defined code to load parameters to the MMa.
USAL Services	TBD by the user. Minimum MMa, VAS, DEM, FPMa and SM.

Deliverables	ACK showing if everything is OK.

Table 2.4 Initialize state characteristics

2.1.4.2. Take-off

Take-off is a departure state in which the MMa continues checking if everything is OK while reaching a safe flight altitude. Once a transition altitude has been reached it loads the parameters for the next state, the EnRoute(objective).

To enter this state one pre-requirement has to be met. It is that the Initialize state has passed an ACK representing that everything is OK. That would mean that the fundamental services are on and the flight is safe.

There is another pre-requirement but it has not to be satisfied before entering to the state necessarily. This requirement is the transition altitude, which determines the changing point from Take-off to EnRoute(objective).

So, the main characteristics of the Take-off state are:

Pre-requirements	OK from the Initialize state and parameters loaded.
Requirements	During the state, the transition altitude.
Goals	Wait to reach a safe altitude to develop the mission and check whether the aircraft is ready for it or not.
Tasks	Continue checking if the flight is safe.
Minimum USAL Services	FPMa, VAS, DEM, SM and MMa.
Deliverables	None.

Table 2.5 Take-off state characteristics

2.1.4.3. En Route(objective)

Once the Take-off has finished (transition altitude reached) the UAS will enter to the EnRoute(objective) state.

This state is basically in charge of turning on the services that are needed for the next chosen state while arriving to the AoI. It is worth mentioning that this fact is one of the biggest contributions of the MMa to optimize the mission.

The services that will be turned on will depend on the next state. The RTDP and the thermal camera, as they will be used for either next state, will be turned on. Regarding to the visual camera, its connection will depend. If the next state is

the Detailed or Reconnaissance with visual camera, it will be turned on. On the contrary, if the next state is the Reconnaissance without visual camera, it will not be.

Therefore, the main characteristics of EnRoute(objective) are:

Pre-requirements	Transition altitude reached and parameters loaded.
Requirements	Next state and AoI.
Goals	Turn on the USAL services needed for the next state of the mission and wait until the UAV reaches the AoI to change the state.
Tasks	Turn on services while arriving to the AoI.
Minimum USAL Services	FPMa, MMa, VAS, DEM, SM, RTDP and depending on the next state both cameras or only the thermal.
Deliverables	None.

Table 2.6 EnRoute(objective) state characteristics

For flexibility, the next state, as well as the AoI, can be defined any time. Nonetheless, some precaution has to be taken given that the tasks will not be performed until they have been defined. This is why, obviously, we cannot perform our operation of turning on USAL services on if we do not know what services will be needed or when they will be needed.

If our next state is the Reconnaissance (see 2.1.4.4.) without the visual camera, the tasks that will be developed during the EnRoute (objective) expressed in time would be:

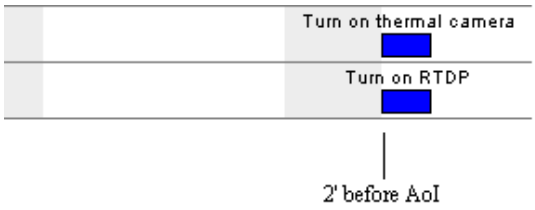


Figure 2.9 EnRoute(objetive) tasks in time

In other case that our next state is the Reconnaissance with the visual camera or the Detailed, the tasks would be the same than in figure 2.9 with the addition of turning on the visual camera.

The time of turning on the services two minutes before the AoI comes from the characteristics of the USAL Systems (see section 2.1.1.1.).

The end of EnRoute(objective) arrives when the AoI is reached. In that moment, we will enter to the state we have chosen (Reconnaissance or Detailed).

2.1.4.4. Reconnaissance

Reconnaissance, on a general basis, is the phase of the mission where we "recognize" the terrain. It has as a main objective detecting hot-spots and listing them to enter later in the Detailed state.

For electing the type of Reconnaissance that we want to perform, we must take into account what type of exploration we want to perform and the characteristics of our devices. For example, if we want to make some recognition on the terrain and we have a low RTPD time, we will find more effective using a Reconnaissance with the visual camera. However, if our SM has not a high capacity, we can use a Reconnaissance state without the visual camera to avoid overflowing the system.

It also has some pre-requirements to enter to this state. The first one is to have reached the AoI. The second one, to have turned on the appropriate services and the third is to have loaded its parameters.

However, what is more special in this state, conjunctly with the Detailed, is the optimization, flexibility and autonomy that is brought to the mission. This is thanks to the photo overlapping, the hot-spot false alarm detector and the type of scan.

Starting with the overlapping, before the implementation of the MMa, its value did not change the way of operating the UAS. The UAS speed and the photo frequency were related to compute which overlapping was being obtained. Now, on the contrary, the user will enter a value of overlapping wanted and the MMa will give the order of taking photos to keep constant that value taking into account the UAS speed, the height and the camera tilt.

Regarding to the hot-spot false alarm detector, it is achieved optimization to the mission given that if a new hot-spot comes from a known one, it will not be reported. Nevertheless, it is a reconfigurable option in case the user wants to report all hot-spots detected.

Finally, the type of scan that the UAV performs in the AoI is used to bring autonomy to the MMa. Now, the user will be able to plan what type of scan pattern will be automatically performed in the AoI. In this state, it will be available a typical scan or a scan with an eight every time a hot-spot is detected.

The main characteristics of the Reconnaissance state are summarized up as follows:

Pre-requirements	AoI reached, thermal camera on, visual camera on (if chosen), RTDP on, and parameters loaded.

Goals	Take ground information. List the coordinates where there is a hot-spot.
Tasks	Camera management (visual photos on demand), list hot-spots positions and update flight plan if necessary. Also, depending on the following state, turn on the visual camera or turn off both cameras and RTDP.
Minimum USAL Services	FPMa, VAS, CC, CS, DEM, MMa, SM and RTDP.
Deliverables	List of coordinates with hot-spots.
Optional	Reject hot-spot false alarms.

Table 2.7 Reconnaissance state characteristics

The task of taking photos is performed continuously while this state is running. Contrarily, the tasks of listing the hot-spots and rejecting false alarms are only performed when a hot-spot has been detected.

So, the process once a hot-spot has been detected would be:

- Reject hot-spot false alarms (if enabled).
- Perform an eight (if selected).

And if the hot-spot is accepted or the false alarm detector is disabled:

- Store the hot-spot into a list to report it later to the Detailed state.

When this state is finished, it will be performed the Detailed (if there are one or more hot-spots) or EnRoute(land) state as determined by the user.

2.1.4.5. Detailed

This state tries to get the maximum ground information possible by flying at a low altitude near areas where it has been detected a hot-spot.

This state machine proposed allows entering directly from the EnRoute(objective) to the Detailed state if it is already known the position of the hot-spots.

There have to be met some pre-requirements to enter to this state. The most important one is that it needs a list of coordinates to analyze. If not, this state will be not performed. This is why, if we do not have any coordinates to analyze, there is no reason to be of this state. The other pre-requirements are similar to the Reconnaissance state ones: the needed services have to be turned on and the Detailed parameters have to be loaded.

As said before, this state, conjunctly with the Reconnaissance, is where the MMa brings the maximum optimization, flexibility and autonomy to the mission.

In this case, besides the photo overlapping, the false alarm detector and the type of scan, another function is added: the photo fusion.

The photo fusion is a function on demand used to ask the user if he wants the thermal and visual photos fusion. If the user enables it, when it is detected a hot-spot and the images are send to ground, the images will be fusion. That would be especially useful to help the tasks of analyzing the images on the ground.

Also, it is worth mentioning that the type of scan performed in this state is different than the ones in the Reconnaissance. In the Detailed state the user will have the option of choosing between an eight or a hold pattern in the coordinates where there has been detected a hot-spot.

The following table summarizes the characteristics of the Detailed state:

Pre-requirements	AoI reached, thermal camera on, visual camera on, RTDP on and parameters loaded.
Goals	Inspect the hot-spots .
Tasks	Camera management and flight plan updates. Also, depending on the following state, turn off the cameras and RTDP or only turn off the visual camera.
Minimum USAL Services	FP, VAS, CC, CS, DEM, MMa, SM and RTDP.
Deliverables	If positive, send alarm and photos to ground.
Optional	Reject false alarms, order to fusion photos.

Table 2.8 Detailed state characteristics

The tasks developed chronologically are the same than the ones exposed in the Reconnaissance state, with the addition of the photo fusion that is performed continuously as well as the order of taking visual photos.

When this state is finished, it will be performed the Reconnaissance or EnRoute(land) as determined by the user.

2.1.4.6. En Route (land)

When we have finished the mission goals and the UAV has left the AoI, we will enter in the En Route (land) state.

The main objectives of this state are starting the way back to the origin and turning off the cameras and RTDP, as they will not be needed anymore.

The main characteristics of this state are summed up in the next table:

Pre-requirement	AoI left.
Goals	Return to the origin position and turn off the cameras and RTDP.
Tasks	Update the FP to return to the origin and turn off the cameras and RTDP.
Minimum USAL Services	FPMa, VAS, DEM, SM and MMa.
Deliverables	None.

Table 2.9 En Route(land) state characteristics

This state ends when the UAS has returned to the take-off position or it receives a Runway event from the FPMa. Once that position is achieved, the UAS will enter to the Landing state.

2.1.4.7. Landing

This state looks for the safe ending of the mission. It is a state thought for the future. At present there is no implementation to land with extra sensors to help the UAV landing safely. Thus, when those sensors or extra functionalities will be operative, they will be managed in this state.

For the moment, it only has one entry pre-requirement: to have the next state parameter loaded into its configuration.

Its characteristics can be found in the next table:

Pre-requirement	Next state.
Goals	Manage landing aids.
Tasks	For the moment, transfer the execution to the next selected state.
Minimum USAL Service	FPMa, VAS, DEM, SM and MMa.
Deliverables	None.

Table 2.10 Landing state characteristics

The election of the next state will depend on the user needs. This architecture proposes two cases, a Save state and an End state. If we want a state which gives us some time to download data from the air services, the Save state is our state. On the contrary, if the mission is ended because we already have all the data we need, we will go directly to the End state.

2.1.4.8. Save

As said before, the Save state gives us some time to call a function to download data (or any wanted operation) and waits until this function is over to end the mission. In other words, it is a state where the state machine is paused to let the user have some time to perform any wanted operation before the services are closed.

Its characteristics are summarized in table 2.11:

Pre-requirement	None.
Goals	Give some time to the user to download data from the services.
Tasks	Call the appropriate function to download data.
Minimum USAL Service	FPMa, VAS, DEM, SM and MMa.
Deliverables	ACK function.

Table 2.11 Save state characteristics

When it receives an ACK showing that the downloading operation has ended, it will give way to the End state.

2.1.4.9. End

This final state waits until the user notifies that the mission is over and then closes all the air services to end the mission.

The following table shows its characteristics:

Pre-requirement	None
Goals	Turn off remaining services and end mission
Tasks	Turn off all remaining services
USAL Service	FPMa, VAS, DEM, SM and MMa
Deliverables	ACK function

Table 2.12 End state characteristics

2.2. Mission Monitor Design

The mission operator will interact with a MMo for designing the mission, keeping control of it and interacting with the MMa by sending new orders or just updates.

This chapter will explain its design including its functionalities and architecture proposed [7].

2.2.1. Mission Monitor under USAL

As there is a FPMo and a FMo to deal with the peculiarities of their managers, the MMo will be the human machine interface between the MMa and the operator. The interaction with other USAL services will be done by the MMa. Therefore, the MMo will be integrated in the Ground Segment of the USAL architecture seen in figure 1.4 (see section 1.2.) to communicate with it.

2.2.2. Architecture

The MMo will have two differentiated parts: the **Mission Setup** and the **Mission Director.** The Mission Setup (MS) is where the user will define and configure the mission. Once the mission is defined, this part will be closed by the operator and will be no longer used. The Mission Director (MD) will be used to interact while the mission is being performed. The MD will allow changing values previously defined.

2.2.2.1. Mission Setup

As said before, the main goal of the MS is to define a mission and all the parameters that can be configured. Each screen that will appear in the MS will correspond to one mission state and will let the user configure its parameters.

The configurable parameters may be as general as which USAL services to use at a certain mission, or very particular such as the safe altitude of a UAV. The following table summarizes the parameters that will be configurable in each state:

State	Parameters
Initialize	Name of the mission, USAL services needed, camera characteristics (tilt and aperture angles) and operation characteristics (photo filter (see 3.2.2.)).
Take-Off	Transition altitude.
EnRoute (objective)	Time to turn on services, next state (Detailed only if there are hot-spots) and the AoI.
Reconnais-sance	Scan pattern, visual camera on/off, false alarm detector on/off, next state, reject distance for the false alarm detector and photo overlapping wanted.

Detailed	Scan pattern, photo fusion on/off, false alarm detector on/off, next state, reject distance for the false alarm detector, photo overlapping wanted and hot-spots positions if already known.
EnRoute (land)	No Parameters.
Landing	The next state.
Save	No Parameters.
End	No Parameters.

Table 2.13 MS parameters

Another characteristic of the MS is that it has to let the user choose where to save the Mission and with what file name. This will be especially useful later on when we have to choose a mission to develop.

2.2.2.2. Mission Director

The Mission Monitor Director (MD) will let keeping track of which state is the UAS currently flying and also making reconfigurations in the mission previously defined.

The reconfigurations will be possible in the state that the UAS is flying as well as in some other future states. This would be practical when, for example, it had been previously determined that the visual camera was not wanted in the Reconnaissance state and later, on flight, it is noticed that it would be better for the mission that the visual camera was on.

Furthermore, in the Objective states, Reconnaissance and Detailed, the MD will display what scan pattern the UAS is performing and some extra notifications. These last ones will inform the user that the false alarm detector, the fusion and/or the cameras are connected and how many hot-spots have been detected so far. The camera notifications will also appear when the UAV is in the EnRoute(objective) state.

CHAPTER 3. MISSION CATEGORY DEVELOPMENT

Once seen the design, this chapter will explain everything related to the development of the Mission Category designed.

In order to do so, the chapter has been divided into three subsections. The first one will correspond to a brief explanation of the software used while other two will correspond to the MMa and MMo development, respectively.

3.1. Software

Next, it is going to be shortly explained the MWF program to understand the tool in which the state machine was developed.

First, mention that it has been used the platform .NET 3.0 because it was more intuitive to use and the preparations to get used to the MWF were made under .NET 3.0. Therefore, changing to another platform would have meant a delay to the development.

Also mention that there was no discussion on the programming language. Inside the Sky-Eye's project it was being used c# to programming the other services, thus the Mission Category just used the same language.

The next figure shows the used MFW activities to interact with the state machine:

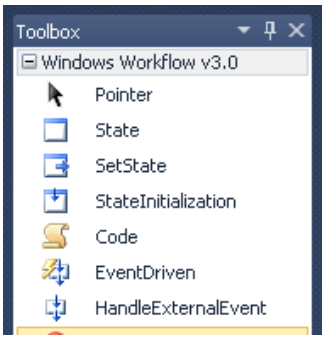


Figure 3.10 MWF state machine activities

The description of each one of these activities is [8]:

- **State:** This activity represents a state within a state machine.
- **SetState:** This activity is used to specify a transition to a new state within a state machine.

- **StateInitialization:** This activity is part of a state activity, and is made up of other activities that are executed when the state activity is first entered.

- **Code:** This activity allows writing code lines that will be executed when the workflow passes through it.

- **EventDriven:** Wraps an activity whose execution is initialized by an event.

- **HandleExternalEvent:** This activity stops the flow and waits for an external event. When the event is received, the flow proceeds to the execution of the next activity.

With these activities the state machine needed for the MMa could be developed. What has been done is using a state activity to define any of the states seen in the design part. Then, with a code activity inside a StateInitialization, it has been performed some of the tasks that the states have to complete like turning on USAL Services. Also, inside the state as well, it has been defined an EventDriven with a HandleExternalEvent that calls a SetState activity (figure 3.11). This is used for reacting to an external event and moving the state machine forward to the next defined state.

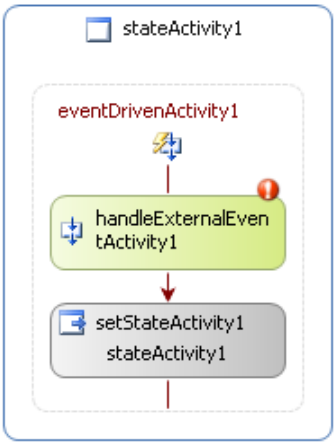


Figure 3.11 State activity example

3.2. Mission Manager Development

Figure 3.12 shows the state machine for the MMa implemented in the Sky-Eye's project. In it, it can be appreciated the 9 designed states with two state activities. On the upper side of the states there is the StateInitialization activity (remarked with squares in the figure) to develop each state task. In the lower side of the states (remarked with ellipses in the figure), there is the event driven activity used to listen to external events to change the state. This last activity has inside the code, HandleExternalEvent and SetState activities seen before.

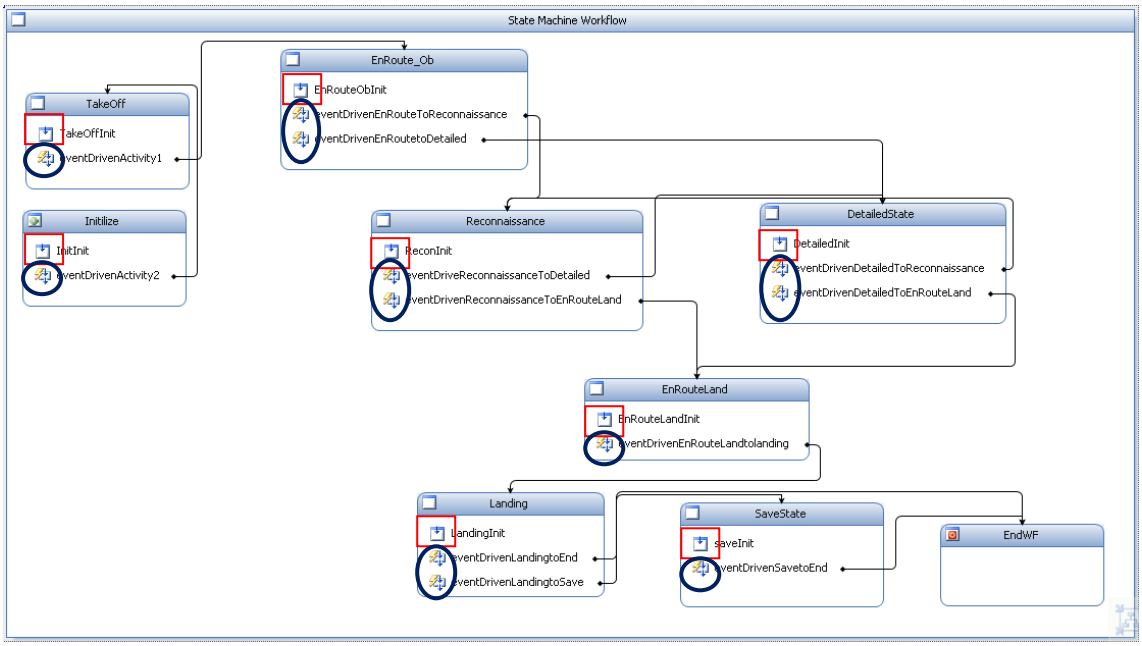


Figure 3.12 State machine implemented in the MMa

Each of the states that figure 3.12 shows has a code that performs its tasks and creates the deliverables. The states, as mentioned before, change when an external event occurs. This means that, if for example, we are inside the Take-Off state and we have arrived to the transition altitude, the Take-Off code will raise the event of changing to the EnRoute(objective) state. The state configuration is executed when we are inside the state. This is why there are general parameters for all the states (e.g. next state) instead of having a big amount of particular parameters for each one of them. Another important fact is that the parameters that the MMa uses are centralized. Rather than sending individually the parameters that arrive to the MMa to all the classes that needs them, they are stored in a central class. Then, when a class needs one of them, it goes to look for it. Some other important characteristic of the code developed is that the parameters are reconfigurable. The MMa has been developed taking into account that we could send parameters to the state configuration and change the predefined value. If it happens that we want to change a parameter in the actual state that the UAS is, it is not a problem because in that case, the actual parameter will be changed, not the configuration. Also it is worth mentioning that in order to improve the flexibility, in any moment a state can be changed to a logical forward one. In addition, if the entry requirements of the next state are not met, it is performed the required actions to met them.

The whole documentation of the code produced in the MMa can be found in the CD attached. Additionally, annex E.1. contains a summary of it.

Regarding to the operation of the MMa, it goes as follows. First the user plans the mission on the MS, which stores the parameters in an XML file specified by the user (see 3.3.1.). Then, the day of the mission, the MMa asks the user which mission wants to perform. When the appropriate XML file is selected, the

parameters defining the mission are read and loaded by the MMa into each state configuration. Then, the MMa goes executing the different states performing the tasks designed for each one of them.

The next sections explain the development in detail for some states, the interactions between the Flight and Payload Services and some remarkable function developments.

3.2.1. States Characteristics

- **Initialize and End State**

In the Initialize state, the MMa turns on the services that are specified in the XML mission file. Then, the End state closes the remaining turned on services. These functions of turning on/off services can be performed thanks to MAREA.

- **EnRoute(objective)**

In this state it is created an imaginary area that is found in the time that the user has decided (two minutes by default) from the AoI (this procedure can be found in Annex A.1).

We could think about what happens if we enter to the area where we are two minutes time and we turn on the required USAL Services but afterwards the UAV leaves the area. If that happens, the services will turn off because the MMa continues monitoring all time the UAV position. That means that, for example, if we enter into the area where we are a two minutes time from the AoI, we leave and then we re-enter, the cameras and the RTDP would have been connected, disconnected and finally connected.

- **EnRoute(land)**

The MMa uses two ways for knowing if the UAS has returned to the take-off position. The first should be a Runway event sent from the FPMa. However, as this message is not implemented for the moment, the Runway event is manually sent to the MMa. The second one is from the MMa itself. The MMa gets the take-off position and if it detects in this state that the actual position is the same than the previously stored, with a certain margin, it ends the EnRoute(land) state.

3.2.2. Interaction with Payload Services

During the Reconnaissance and Detailed states, one of the tasks that have to be developed is taking photos. This means that the MMa has to communicate to the CC to address the appropriate orders. The different messages used are:

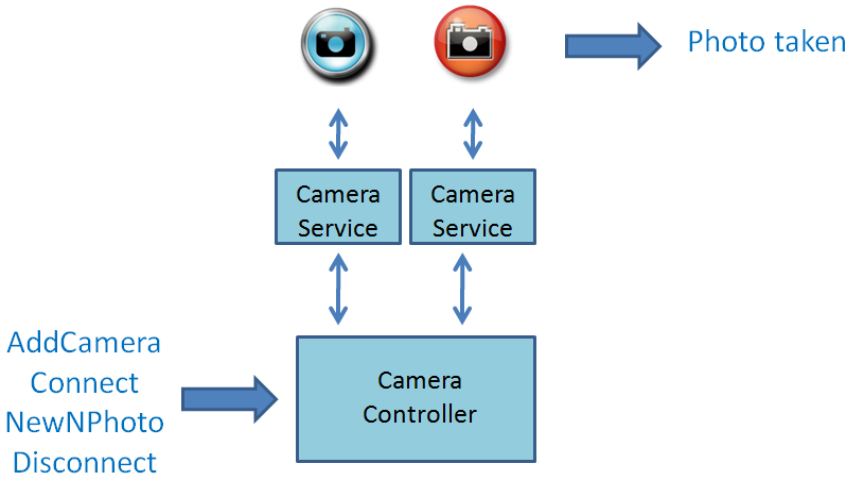


Figure 3.13 Payload interactions messages

The actions these messages produce are:

- ***Add Camera*:** Registers a camera to the system.
- ***Connect*:** Turns a camera on.
- ***NewNPhoto*:** Asks the CC to take one/several new pictures. The parameters of this message are the number of photos and the frequency wanted.
- ***Disconnect*:** Turns a camera off.

It is worth mentioning that all the procedures of taking photos, analyzing images, rejecting hot-spots, etc. are only performed if the UAV is inside the AoI. This means that, automatically, when the UAV is inside the Reconnaissance or Detailed state but outside the AoI, the MMa stops performing the tasks and it waits until the UAV re-enters into the area.

3.2.3. Interaction with Flight Services

A highly important task that is held in the Reconnaissance and Detailed states is dealing with the FPMa to adapt the FP to the circumstances of the mission.

Assuming that a FP is created, the messages exchanged between the MMa and the FPMa are just in case a modification is needed. However, in order to produce modifications, the MMa must know what the UAS is doing, and what flying patterns are charged in the FP.

Figure 3.14 shows what the FPMa is able to do. The three flight patterns needed for the proper development of the remote sensing mission (scan, hold, and eight) have been charged to the FP inside an iterative sequence. This means that until a number of loops have not been met, the UAS will be performing one of these flight patterns, whichever is selected.

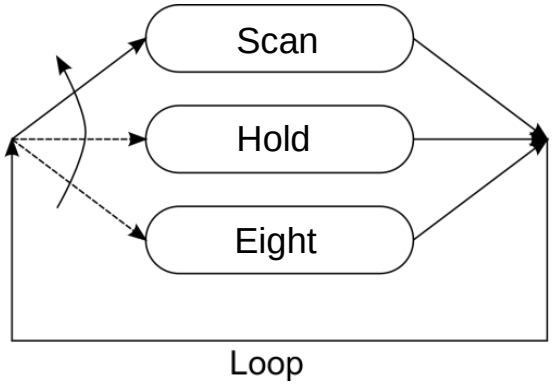


Figure 3.14 Iterative flight plan scheme

In order to produce modifications in the iterative sequence, the FPMa provides several messages that the MMa sends thanks to MAREA:

- **SetCondition:** Controls the selector of the FP. In the picture is represented as a non-lineal arrow at the left of the patterns.
- **Skip:** When performing a pattern, if the Skip is called, it will interrupt the current action, and make a loop for a new iteration.
- **Update:** An Update is called when the properties of the pattern charged at the FP must be changed.
- **GoToLeg:** Stops performing the iterative sequence. This is especially useful when it is wanted to return home. So, this message is used every time that it is reached the EnRoute(land) state.

These messages must have a specific form in order to be understood by the FPMa. The form and the variables can be found in Annex B.

With the iterative sequence and the messages provided for the FPMa, the MMa is able to fulfil its goal of adapting the FP to the circumstances of the mission. For example, the following example shows what messages the MMa will have to send to the FPMa in case that it is wanted to reproduce a scan with an eight when a hot-spot is detected:

```
update (SCAN);
setCondition(SCAN);
skip(HOLD);
while (!End_SCAN)
{
    if (hot_spot_detected)
    {
        update(EIGHT);
        setCondition(EIGHT);
        skip(SCAN);

        while (!End_EIGHT)
        {
        }

        update(SCAN);
        setCondition(SCAN);
        skip(EIGHT);
    }
}
```

Figure 3.15 Scan and eight pattern example

Unfortunately, for the moment the FPMa does not send any message of End Scan. Consequently, the MMa is manually noticed that the UAV has finished performing a scan pattern.

Finally, it is worth mentioning that every time that we send an event to the FPMa, it sends us an ACK informing whether it is correct or not. In case that it is not correct, the MMa resends the event.

3.2.4. Tasks Characteristics

One of the tasks of the MMa consists in keeping constant the photo overlapping. Therefore in this case, it is fundamental to determine the frequency of taking them (to see the procedure to compute the overlapping see annex A.2). In order to prevent constant changes in the frequency, a filter has been developed inside the MMa. This filter avoids altering the frequency when there is a change equal or less than 0.2 seconds. These 0.2 seconds correspond to the mean time to have a change in the overlapping of 10%. In other words, unless there is a change in the velocity, height or any other parameter high enough to change the overlapping a 10% or more, the frequency of taking photos will not be altered. This parameter is reconfigurable in case the user wants a change higher or lower than 10%.

Another important task is rejecting hot-spots with the false alarm detector. This function would let the MMa identify, for example, that a new hot-spot alarm is coming from a previous hot-spot detected and reject it. This function could be

disabled by the user to inform of all hot-spots detected (to see the procedure or rejecting photos see annex A.3).

Also, during the design part, it was told that the user could choice to fusion on demand the photos during the Detailed state. Unfortunately, this function is not implemented for any Payload Services and consequently it cannot be performed. Nevertheless, both the MMa and the MMo are prepared for performing this function and they will act as if this function operates. Hence, the day that the function will be operative it will just be seconds to implement it in the MMa and the MMo.

3.3. Mission Monitor Development

In this section it will be described the development of the MMo from two different points of view, the MS and the MD.

As well as in the MMa case, the whole documentation of the code produced in the MMo can be found in the CD attached. Also, annex E.2. contains a summary of it.

3.3.1. Mission Monitor Setup Development

The MD is formed by a sequence of Windows Forms each one corresponding to a state defined in the MMa. In those forms appear the parameters defined in the design part to allow the user to create a mission.

The following images show two examples of Forms used in the MS. Figure 3.16 corresponds to the Detailed state Form in which we define the characteristics of this state. In this case the user can check the different boxes to select the scan pattern to perform, to enable the false alarm detector and the fusion and to select the next state. There are also two labels to introduce the reject distance for the false alarm detector and to establish a value for the overlapping. Also it can be appreciated a button to introduce hot-spots and a continue button to move forward to the next configuration Form.

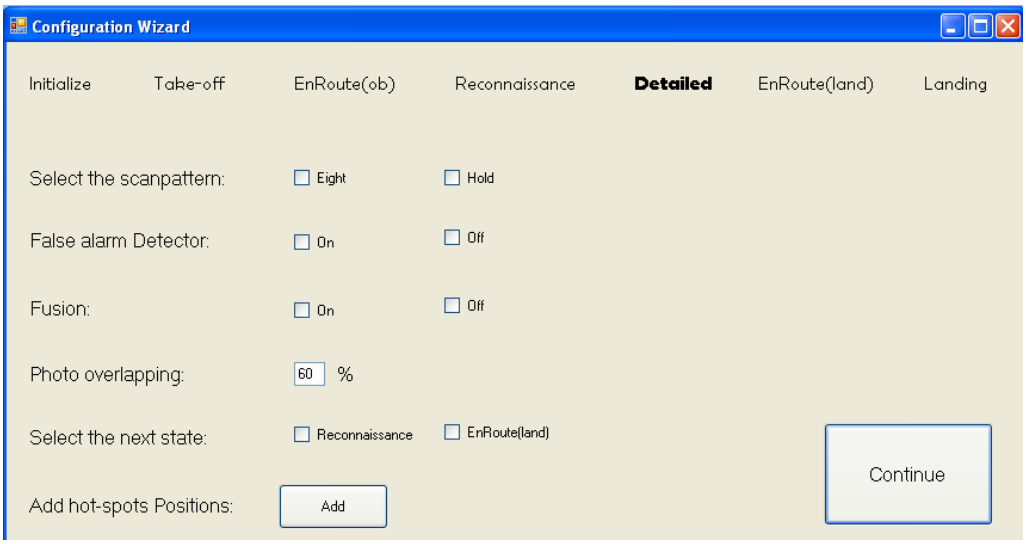


Figure 3.16 Configuration Wizard for the Detailed state

Figure 3.17, for its part, is the Form to select the AoI. It is worth mentioning that it allows the user to select the AoI by introducing the coordinates or by clicking on the map:

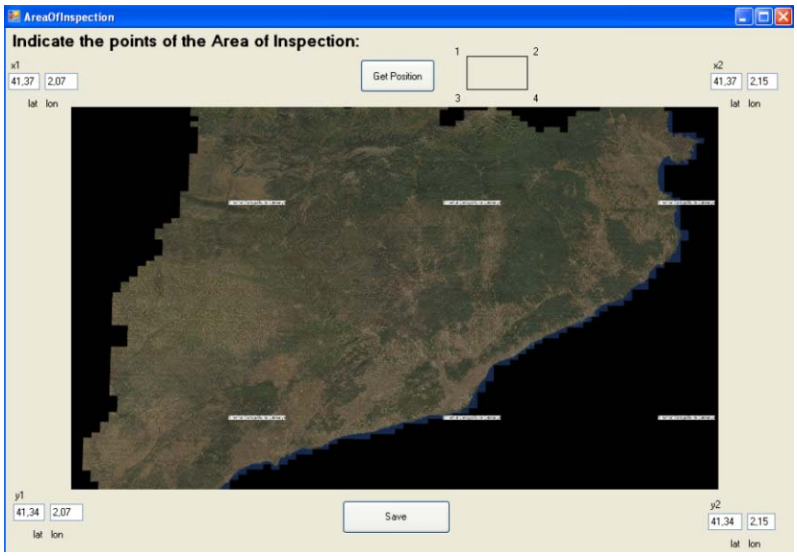


Figure 3.17 Configuration Wizard for the EnRoute(objective) state

What is important in the MS, is that when we arrive at the end of the Forms and we click on a save button, the mission is automatically saved into a XML format to be read later on the MMa. An example of a mission created and saved in a XML format can be found in Annex C.

3.3.2. Mission Monitor Director Development

The MD is also formed by Windows Forms. Although forming part of the MMo, the Forms have no relation with the MS, in exception of the Forms used to indicate the AoI and to introduce hot-spots.

Figure 3.18 shows the main Form of the MD:

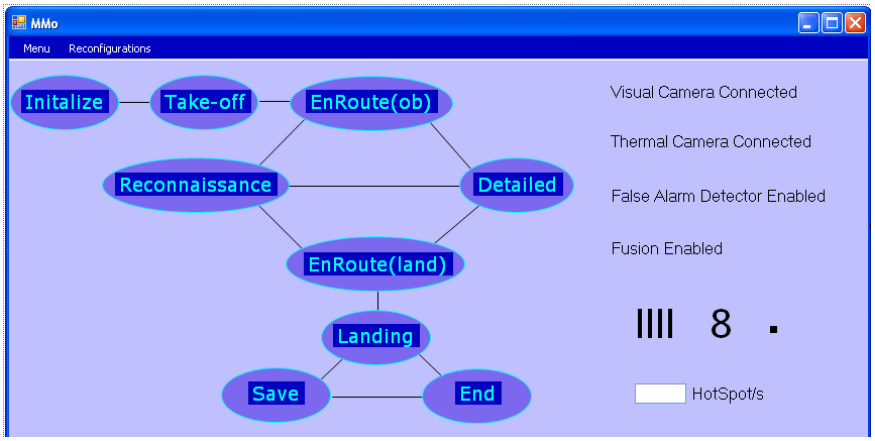


Figure 3.18 Mission Director Form

It can be observed, how the MD informs the user that the cameras, the false alarm detector and the fusion are connected and the type of scan pattern that is being performed during the Reconnaissance and Detailed states. To indicate which state the UAV is currently on, the MD highlights the label of the state with a green colour, like figure 3.19 shows:

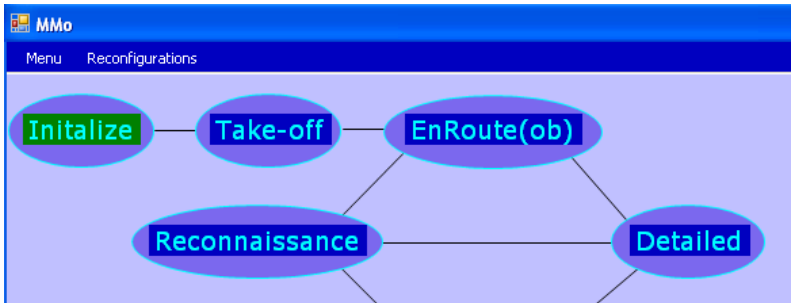


Figure 3.19 MD indicating which state is the UAV currently on

The MD knows these parameters because it is informed by the MMa. The messages used are:

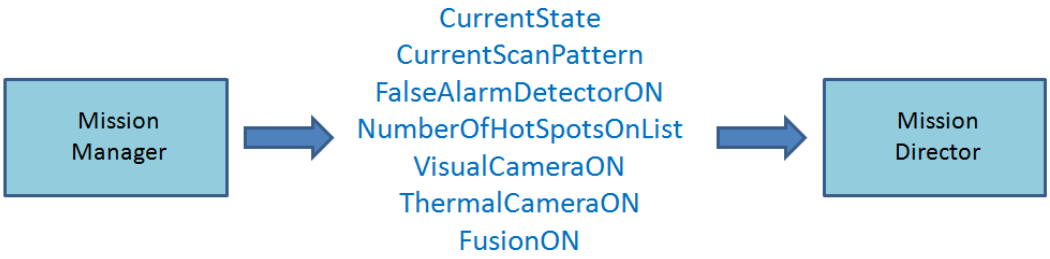


Figure 3.20 Messages from the MMa to the MD

On the upper left corner of the MD form it can be observed a drop-down menu. When we click on the Menu label and then Go to..., it opens the following options that are showed in figure 3.21:

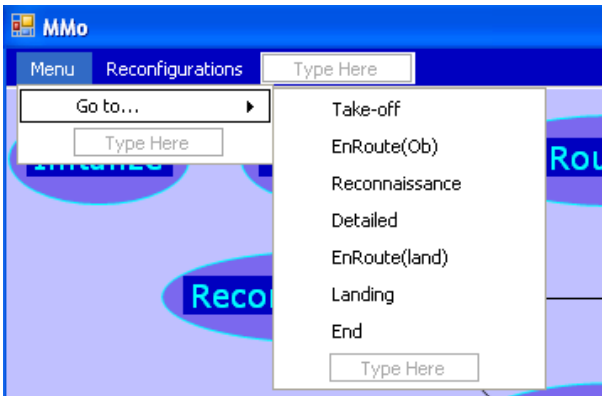


Figure 3.21 Menu options

The functionality of each one is very simple. If we are in the Reconnaissance state and we want to change to the Detalied state without waiting until the end of the scan or any other exit requirement, we will have to click on the Go to Detailed state.

Right to the Menu label there is another drop-down menu for the Reconfigurations. As said before, it allows reconfigurations of the parameters previously charged in the MMa. Figure 3.22 shows the reconfigurations for the next state after Detailed:

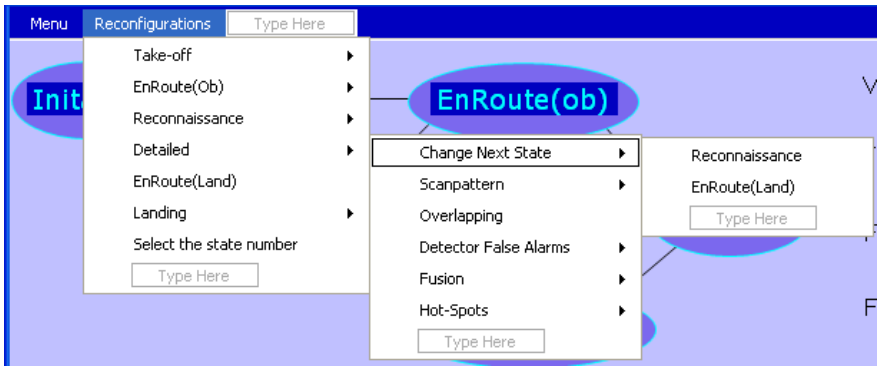


Figure 3.22 Detailed next state reconfigurations

When we click on a parameter that is defined numerically, like for example the Transition Altitude in the Take-Off state, it shows another form that lets the user introduce a number, like figure 3.23 shows:

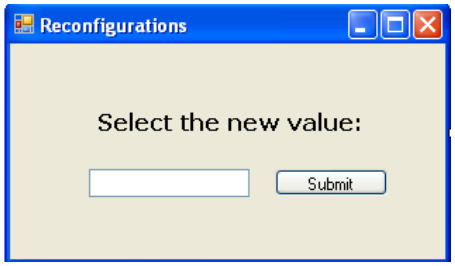


Figure 3.23 Number introduction in the MMo

Another special form is the one used to introduce hot-spots which can be observed in figure 3.24. It is worth mentioning that the Form allow submitting hot-spots, visualizing them and removing them in case that the user committed an error.

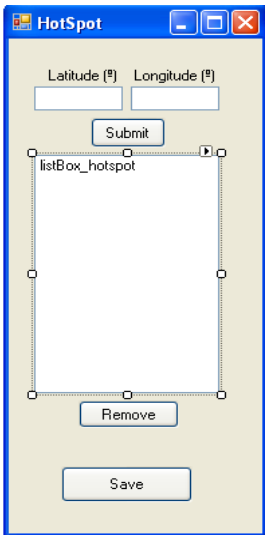


Figure 3.24 Hot-spot introduction form

Finally, figure 3.25 shows some of the messages that are exchanged from the MMo to the MMa in order to make reconfigurations are:

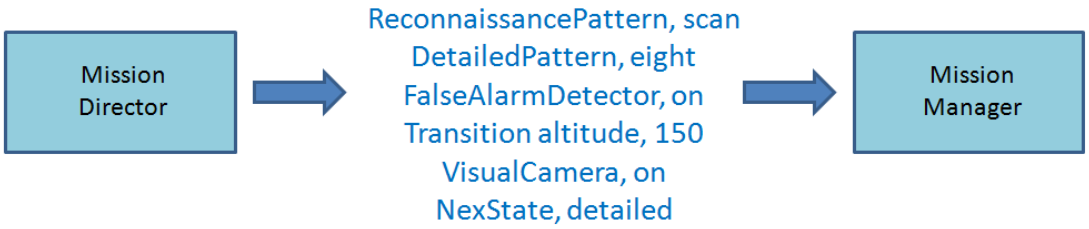


Figure 3.25 Messages from the MD to the MMa

3.4. Mission Interface

Given the big amount of parameters interchanged between the Mission Category it was found the necessity of adding a Mission Interface.

The Mission Interface is formed by empty objects containing variables that are commonly exchanged or that are appropriately grouped. The objects with its description can be found in annex D.

CHAPTER 4. USE CASES

In this chapter it is going to be tested the implementation and potential of the parts of the Mission Category developed by means of two use cases.

The first test will consist of a simple mission in which the MMa will have to perform its basic functions. The second one will be more complex and it will develop some extra functionalities of the Mission Category parts developed.

4.1. Use case 1

4.1.1. Goal

Test the basic functionalities of the parts of the Mission Category developed.

The MMa, minimum, will have to be able to turn on USAL Services, turn on the thermal camera before arriving to the AoI, send orders to take photos, update the FP to perform a scan pattern in the AoI, update again the FP to return home and turn off USAL Services.

The MMo will have to let us planning the mission in the MS and later on the simulation, the MD will have to inform us which state that the UAS is in, when the thermal camera is on and which type of scan pattern is being performed in the Reconnaissance state. Also, it will have to allow reconfiguring the mission planned to not perform the Save state.

4.1.2. Scenario

The next figure illustrates the FP that is proposed in this mission, where the square is the AoI:

Figure 4.26 Use cases FP

The UAV will take-off from Barcelona's Airport Runway 02 (W1). Then, it will go to the waypoint W2. After that, it will go the eight scan where if everything has gone fine it will start performing the scan pattern updated from the MMa. Then, when it is over, it will go to the W4 point. After that, W5, to end finally again in Barcelona's Airport L1.

4.1.3. Performance

It is going to be explained the mission performed from the point of view of the MS, the MD and the MMa.

Starting with the MS, this is the XML that have been created after following the MS explorer:

```
  </Mission>
- <Mission Name="Case">
- <State Name="Initialize">
- <Services>
  <Service>MMaService</Service>
  <Service>VirtualAutopilotLiteService</Service>
  <Service>FlightMonitorLiteService</Service>
  <Service>DEM</Service>
  <Service>FlightPlanMonitorService</Service>
  <Service>FlightPlanManagerService</Service>
  <Service>Storage</Service>
  <Service>RTDP</Service>
  </Services>
  <Tilt>15</Tilt>
  <X_Cam_angle>25</X_Cam_angle>
  <Y_Cam_angle>18.8</Y_Cam_angle>
  <Photo_filter>0.2</Photo_filter>
  </State>
- <State Name="Take-off">
  <TransitionAltitude>30</TransitionAltitude>
  </State>
- <State Name="EnRouteOb">
- <Area>
  <x1 latitude="41.6741664129136" longitude="2.22755280639818" />
  <x2 latitude="41.6785035145053" longitude="2.40657381935969" />
  <y2 latitude="41.526839" longitude="2.373159" />
  <y1 latitude="41.506571" longitude="2.198375" />
  </Area>
  <TimeServices>120</TimeServices>
  <NextStat>reconnaissance</NextStat>
  </State>
- <State Name="Reconnaissance">
  <ScanPattern>scan</ScanPattern>
  <VisualCam>false</VisualCam>
  <FalseAlarmDetector>false</FalseAlarmDetector>
  <Overlapping>60</Overlapping>
  <NextStat>detailed</NextStat>
  </State>
```

```
- <State Name="Detailed">
  <HotSpotList Value="False" />
  <ScanPattern>eight</ScanPattern>
  <Fusion>true</Fusion>
  <FalseAlarmDetector>true</FalseAlarmDetector>
  <RejectDistance>3</RejectDistance>
  <Overlapping>60</Overlapping>
  <NextStat>enrouteland</NextStat>
  </State>
  <State Name="EnRouteLand" />
- <State Name="Landing">
  <NextStat>Save</NextStat>
  </State>
  <State Name="Save" />
  <State Name="End" />
  </Mission>
```

The next table summarizes the performance of the Mission Category from the point of view of the MMa and the MD:

Table 4.14 Use Case 1 performance

State	Performance
Initialize	• When we enter to this state we are asked which mission we want to develop. • When we select the Mission Case, the MMa charges all the mission parameters planned and turns on the USAL Services specified.
Take-off	• The MMa changes the state when the transition altitude is reached
EnRoute (objective)	• When the UAV is two minutes before the AoI, the MMa turns on the thermal camera and the MMo informs us (figure 4.27). • Then, when it reaches the specified AoI, it changes the state to Reconnaissance.
Reconnais-sance	• When the UAV reaches the Eight Point, the MMa updates the FP to start performing the scan pattern. • The MMo informs us that the thermal camera is on and that the UAV is performing the mentioned scan (figure 4.28 left). • Also, in order to see how the MMa interacts with the CC, it has been used MAREA to show the messages of ordering to take photos (figure 4.28 right). • When the scan is over, as no hot-spots have been detected, the MMa updates again the FP to go to the L4 waypoint and changes the state to the EnRoute(land).
Detailed	• As designed, this state has not been performed as there has not been detected any hot-spot.
EnRoute (land)	• The MMa turns off the thermal camera because it will not be needed any more (figure 4.29). • As we have not detected any hot-spot, we apply one reconfiguration

	to say to the MMa that we do not want to go to the Save state. • When it receives an event of Runway, the MMa changes to the End state.
Save	• Provided that we have reconfigured EnRoute(land) to avoid entering to this state, the Save state has not been performed.
End	• This state is maintained until there is not an End Mission event. • When we say to the MMa that we have ended the mission it turns off all the USAL Services.

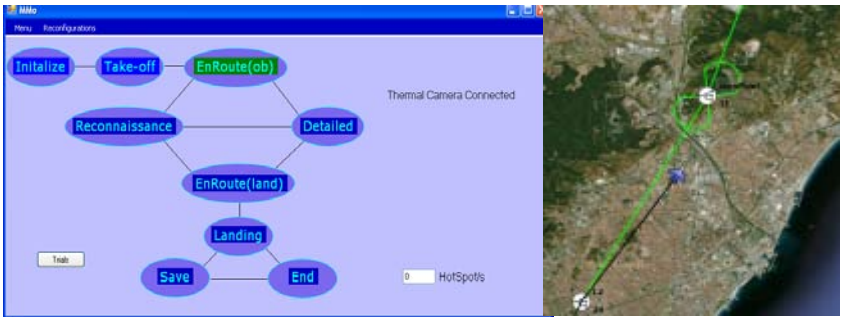


Figure 4.28 Use Case 1. EnRoute(objective) state

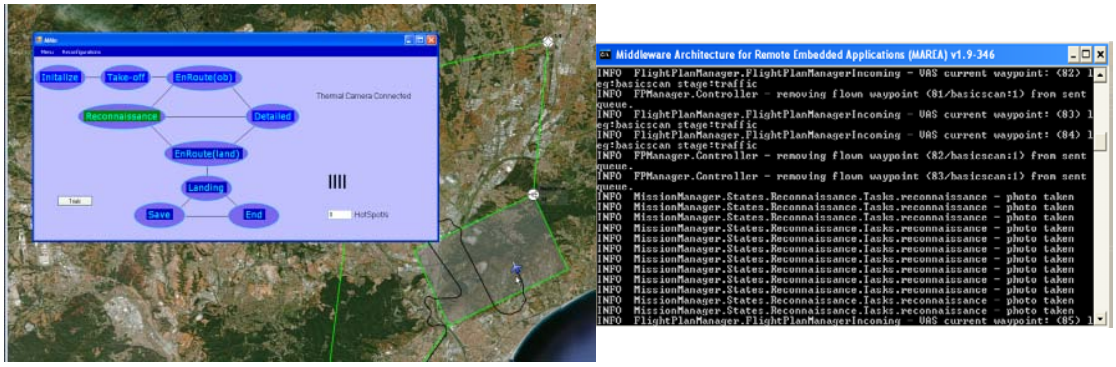


Figure 4.27 Use Case 1. Reconnaissance state

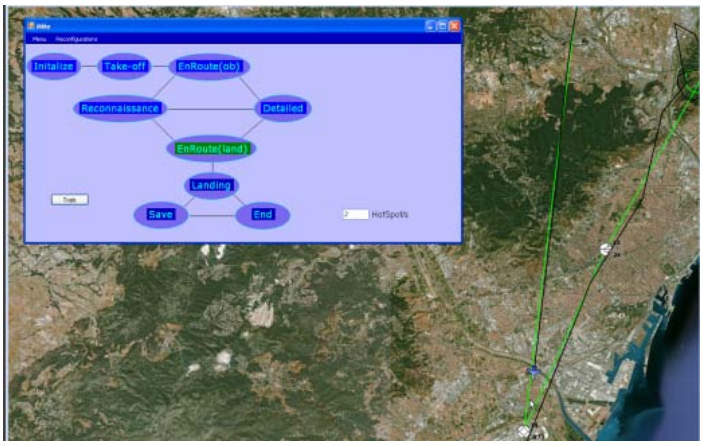


Figure 4.29 Use Case 1. EnRoute(land) state

4.2. Use case 2

4.2.1. Goal

Test more complex functionalities of the Mission Category developed.

In this occasion, it is going to be performed the same scenario than the use case 1 but the tasks to perform will be upgraded. In this second test, the MMa will have to be able to report to the Detailed state the list of waypoints after the scan has been performed in the Reconnaissance state. Then, in the Detailed state it will have to update again the FP to make the UAV perform a hold in the first hot-spot detected and an eight pattern in the second. Thus, both types of scans will be tested. Another extra functionality will be performed in the Detailed state too, because the MMa will have to be able to reject hot-spots with the false alarm detector. Also, we will alter the photo overlapping to test the correctness of the photo frequency.

Regarding to the MMo, the MS will perform the same action. However, the MD will have to allow again reconfigurations and will have to update the information that it shows. For example, in this second mission, it will be reconfigured the visual camera status to turn it on and it will be deactivated the fusion. Especial attention will have to be paid on the Detailed state as it will have to inform that the false alarm detector is on, as well as the eight pattern that have to be performed.

4.2.2. Scenario

The same scenario than in use case 1 (see section 4.1.2.) is performed.

4.2.3. Performance

As mentioned before, the mission is not altered so the MS returns the same mission than the one seen before.

The role of the MMa and the MD is not the same and table 4.15 summarizes the performance of the Mission Category in this second mission:

Table 4.15 Use Case 2 performance

State	Performance
Initialize	• Again, when we enter into this state, we are asked which mission we want to develop. • When we select the Mission Case, the MMa charges all the mission parameters planned and it turns on the USAL Services specified.

Take-off	• The MMa changes the state when the transition altitude is reached.
EnRoute (objective)	• We change the parameter of visual camera and we select that we want the visual camera on. • Then, when the UAV arrives approximately to the L2 waypoint, the MMa turns on both cameras (figure 4.30). • After that, when it reaches the specified AoI, it changes the state to Reconnaissance.
Reconnais-sance	• As seen in the Use Case 1, in this state the UAS starts performing the scan because the FP has been updated. • When it is over, it goes to the Detailed state because this time two hot-spots have been detected.
Detailed	• The UAV enters in the Detailed state and it updates the FP to perform an eight pattern. Figure 4.31 shows how the MD informs us that two hot-spots have been detected and that the cameras and the false alarm detector are on. The fusion has been reconfigured therefore, as planned, it is off. • To test the photo frequency, first, we annotate the photo frequency when the overlapping is of 60%. The MMa tells us that it sends one order to take photos every 4,87s. Then, we reconfigure the overlapping and we establish it at 80%. Now, the MMa says that the photo frequency is one photo every 2.44s. It seems logical because if we want more overlapping the periodicity must be higher. Indeed, after manually computing the photo overlapping with the telemetry of the VAS (it informs us that the altitude is 1037.725 m and the speed 30.28 m/s), it can be affirmed that the value is correct. • To test the false alarm detector performance, we send to the MMa another two hot-spots that are inside the reject distance. The behaviour of the false alarm detector is correct because neither of the two new hot-spots have been accepted and reported to the MD. • When the UAV has finished performing a hold in the first hot-spot (dark arrow in figure 4.31) and en eight pattern in the second (light arrow in figure 4.31), it goes to the L4 waypoint and the state is changed for the EnRoute(land) one.
EnRoute (land)	• In this state, the MMa turns off both cameras and the false alarm detector (figure 4.32). • This state is maintained until it receives a Runway event or it detects that it has returned to the TO position. This time we wait for the UAV to return to the TO position. • When we have reached that position the state changes to the Save state.
Save	• In this state we wait until the data has been downloaded. This function is not implemented for any service thus what we do is saying to the MMa that everything is OK to end the mission.
End	• When the MMa reaches this state, the same procedure seen in

	Use Case 1 is followed. • Therefore, all the services are turned off and the mission is concluded.

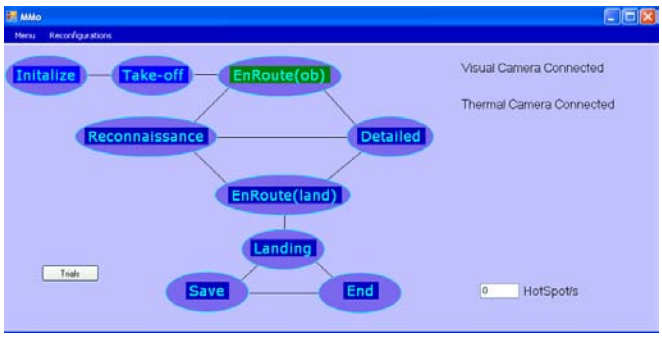


Figure 4.30 Use Case 2. EnRoute(objective) state

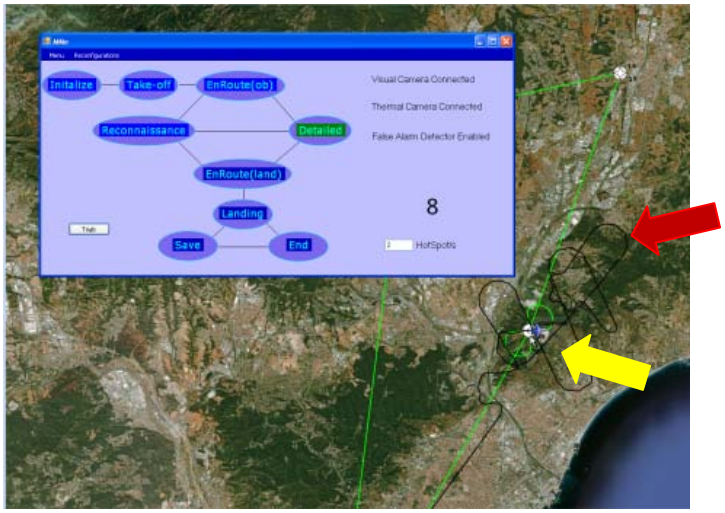


Figure 4.31 Use Case 2. Detailed state

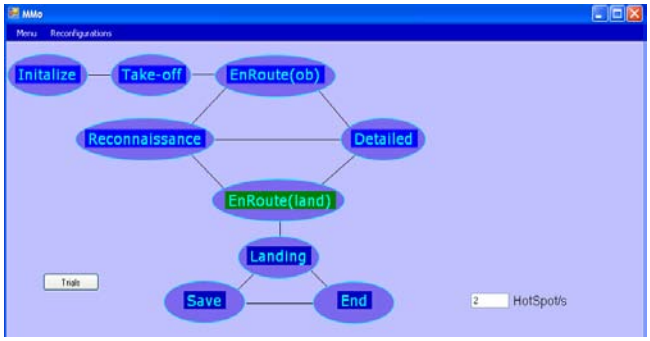


Figure 4.32 Use Case 2. EnRoute(land) state

CHAPTER 5. CONCLUSIONS AND FUTURE LINES

After having tested the performance of the Mission Category developed it can be said that the results are excellent.

On one hand, it has been designed, developed and implemented a Mission Manager that works with the Flight and Payload Sky-Eye's USAL Services. As a result, the project has improved its flexibility, optimization but above all it has gained valuable autonomy. Before this final degree project it not possible to turn the cameras on before arriving to the AoI, to control the photo overlapping or even to modify automatically the FP to adapt it to the mission circumstances among others. Now, these functions are operative and it could be said that all the objectives fixed for this project have been fulfilled.

On the other hand, it has been also produced a Mission Monitor, which was not planned on the initial objectives and represents an extra value to the work done. The Mission Monitor completes the human interface between the Mission Manager and the user and adds the benefit of reconfigurations, visualizations and reduction of the designing time. Before this final degree project it was not possible either to plan a mission before the actuation.

Nevertheless, this project has been the first attempt to create a Mission Category inside the Sky-Eye's project. For this reason, there are several things that could be improved in later versions. The following points summarize the future lines that the Mission Category could follow after this project:

- It could be a wise idea using different states machines to open up new visions in the mission design. For example Petri Nets would let us perform more than one state at the same time.
- The state machine could be generalized to wider its applications beyond remote sensing missions. Also, it could be interesting upgrading it to get back to previous executed states.
- It could be improved the code of the MMa to catch exceptions and inform the user of some of the fails that may appear.
- It would be a wise idea to take profit of the Save state and use it for downloading data or other operations.
- The photo fusion should be operative in the less time possible as well as the End Scan and Runway event from the FPMa.
- The MMo could be improved to display all the state parameters and the services that are connected.
- The human machine interaction between the user and the MMo should be simpler to avoid, for example, having to introduce first the state number and then the new value when making reconfigurations.
- When planning a mission in the MS, it could be interesting having a Form that allows the user to decide which states he wants and in what order.

REFERENCES

[1] P. ROYO, “Chapter VII- USAL Remote Sensing Missions Use Case”, *An Open Architecture for the Integration of UAV Civil Applications PhD Dissertation*, UPC, May 2010

[2] P. ROYO, “Chapter VII- USAL Remote Sensing Missions Use Case”, *An Open Architecture for the Integration of UAV Civil Applications PhD Dissertation*, UPC, May 2010

E. SANTAMARIA, *Reconfigurable Mission Management System for UAS Civil Applications*, Preprint submitted to Aerospace Science and Technology, 2010

[3] P. ROYO, “Chapter III - UAS Service Abstraction Layer”, *An Open Architecture for the Integration of UAV Civil Applications PhD Dissertation*, UPC, May 2010

[4] E. PASTOR, C. BARRADO, P.ROYO, J.LOPEZ AND E.SANTAMARIA, “Chapter 24- An Open Architecture for the Integration of UAV Civil Applications”, *Aerial Vehicles*, T.M.LAM, intechweb.org, January 2009

[5] C.E. NEHME, M.L. CUMMINGS, J.W. CRANDALL, *A UAV Mission Hierarchy, Massachusetts Institute of Technology*, December 2006, [online: accessed February 2011]
http://web.mit.edu/aeroastro/labs/halab/papers/HAL2006_09.pdf

J.R. MARTINEZ-DE-DIOS, L.MERINO AND A.OLLERO, *Multi-UAV Experiments: Applications to Forest Fires*, A. Ollero and I. Maza (Eds.). 2007

J.R. MARTINEZ-DE-DIOS, L.MERINO AND A.OLLERO, *Unmanned Aerial Vehicles as tools for forest-fire fighting*, V International Conference on Forest Fire Research, D. X. Viegas (Ed.), 2006, [online: accessed February 2011]
http://grvc.us.es/publica/congresosint/documentos/2006VICFFR_AOLLERO.pdf

Unmanned Aerial Vehicle (UAV) For Search and Rescue, [online: accessed February 2011]
http://www.u2learn.net/software_engineering/UAV_Project/proposal.html

[6] *K. MCNEISH, Windows Workflow Foundation Essentials*, 2007, [online: accessed February 2011]
http://www.code-magazine.com/Article.aspx?QuickID=0711071

Commons SCXML, [online: accessed February 2011]
http://commons.apache.org/scxml/

State Chart XML (SCXML): State Machine Notation for Control Abstraction W3C Working Draft, April 2011, [online: accessed February 2011]
http://www.w3.org/TR/scxml/

Project Open: Petri Nets in the Workflow Package, [online: accessed February 2011]
http://www.project-open.org/documentation/workflow_petri_nets

Petri Nets World, Frequently Asked Questions, [online: accessed February 2011]
http://www.informatik.uni-hamburg.de/TGI/PetriNets/

[7] J. LEMA, *Mission Category*, ICARUS, April 2011

[8] B.R. MYERS, *Foundations of WF and Introduction to Windows Workflow Foundation*, Apress, 2007

[9] J. LEMA, *Mission Category*, ICARUS, April 2011

E.SANTAMARIA, C.BARRADO, E.PASTOR, *A Reconfigurable Mission Management System for UAS Civil Applications*, UPC, September 2010

[10] Doxygen Documentation System, [online: accessed June 2011]
http://www.stack.nl/~dimitri/doxygen/

Escola d'Enginyeria de Telecomunicació i Aeroespacial de Castelldefels

UNIVERSITAT POLITÈCNICA DE CATALUNYA

ANNEXOS

TÍTOL DEL TFC: Mission Management for Unmanned Aircraft Systems

TITULACIÓ: Enginyeria Tècnica Aeronàutica, especialitat Aeronavegació

AUTOR: Marta Valenzuela Arroyo

DIRECTOR: Juan Manuel Lema Rosas

DATA: 4 de Juliol de 2011

ANNEX A: MISSION MANAGER PROCEDURES

A.1. Procedure to determine when the UAV is near the AoI

1. From the value of the speed given by the VAS, it is computed the distance that the UAS covers 120 seconds (reconfigurable).
2. Next, it is created a square expanding the AoI with the distance computed in 1.

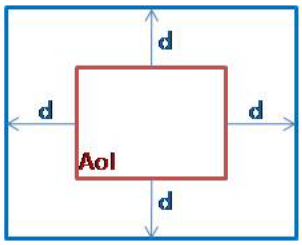


Figure Annex A.1.33 AoI expanded

3. Then, with a function that tells if the UAV is inside the bigger square, it is kept monitoring the actual position of the UAV. When the UAV enters into the square, as that means that the UAV is at 120 seconds from the AoI, the MMa gives the order of turning on the USAL Services needed.

A.2. Procedure to determine the frequency of taking photos

For computing the frequency of taking photos it is needed to know the characteristics of the photos that are obtained.

The first remarkable thing is that the photos are trapeziums. This is due to the x dimension of the photo that becomes bigger as the distance from the camera to the ground increases.

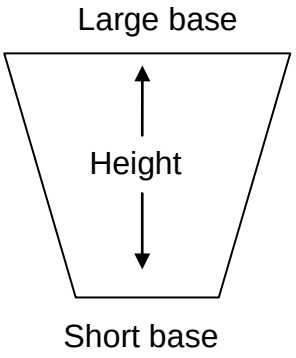


Figure Annex A.2.34 Example of a photo's shape

Nevertheless, after some calculations of the photos obtained (see table Annex A.2.16), we could state that the photos are almost rectangles.

Tilt (deg)	0		15		
Z(m)	**Width**	**Height**	**Short Base**	**Large Base**	**Height**
15	6,7	5,0	6,7	7,3	5,3
20	8,9	6,6	8,9	9,7	7,1
25	11,1	8,3	11,1	12,2	8,9
30	13,3	9,9	13,4	14,6	10,7
35	15,5	11,6	15,6	17,0	12,4
40	17,7	13,2	17,8	19,5	14,2
45	20,0	14,9	20,0	21,9	16,0
50	22,2	16,6	22,3	24,3	17,8
100	44,3	33,1	44,6	48,7	35,6
200	88,7	66,2	89,1	97,4	71,1
300	133,0	99,3	133,7	146,1	106,7
400	177,4	132,4	178,2	194,8	142,2

Table Annex A.2.16 Photo calculations

This fact has been taken profit by the MMa. A trapezoidal overlapping would not have been very practical because some parts of the photo will be overlapped while some others would not. Therefore, supposing that the photos are rectangles, the calculations become simpler and reliable. In addition, this

supposition would only represent an error on the overlapping of 4% in the nominal UAV height (100m).

To calculate the photo frequency, the MMa computes the overlapping vertically. Nonetheless, it needs to know both the short base of the photo (x dimension) and the length (y dimension). The x dimension is useful to know the separation distance between the different legs when performing the scan. For its part, the y dimension of the photo will be what determines the periodicity of taking photos.

The procedure used to determine the x photo length is based on trigonometric as follows:

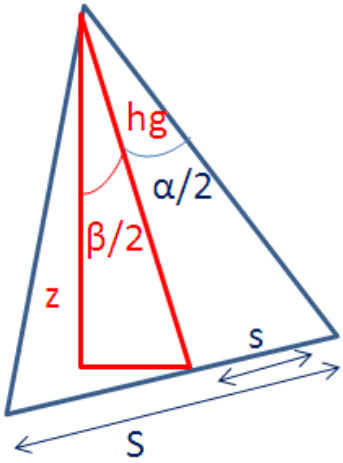


Figure Annex A.2.35 Procedure to determine the x dimension of the photos

$$hg = \frac{z}{\cos(\frac{\beta}{2})}$$

$$s = hg \cdot \tan(\frac{\alpha}{2})$$

$$\boldsymbol{x\ dimension} = S = \mathbf{2} \cdot \boldsymbol{s}$$

Using trigonometric too, the procedure to determine the y photo length is:

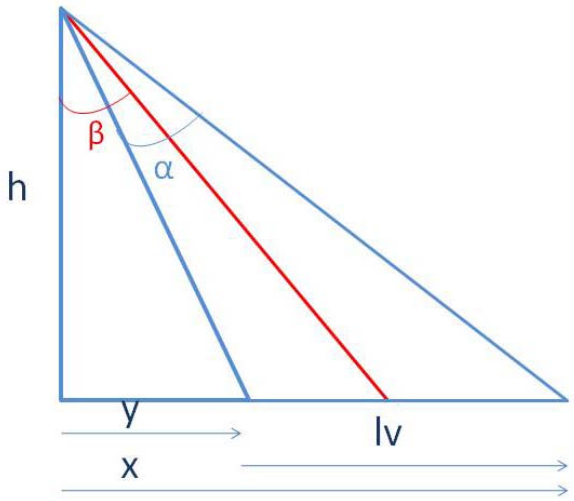


Figure Annex A.2.36 Procedure to determine the y dimension of the photos

$$\tan\left(\beta + \frac{\alpha}{2}\right) = \frac{x}{h}$$

$$\tan\left(\beta - \frac{\alpha}{2}\right) = \frac{y}{h}$$

$$\boldsymbol{y\ dimension} = lv = \boldsymbol{h} \cdot \left[\boldsymbol{tan}\left(\boldsymbol{\beta} + \frac{\boldsymbol{\alpha}}{\boldsymbol{2}}\right) - \boldsymbol{\tan}\left(\boldsymbol{\beta} - \frac{\boldsymbol{\alpha}}{\boldsymbol{2}}\right)\right]$$

With the y dimension, and taking into account the UAS velocity and the overlapping wanted, the frequency of taking photos is:

$$\frac{photo\ dimesion\ (y) \cdot (\ 100 - overlapping\ wanted\ (\%))}{100} = D_{between\ photos} = x$$

$$x \cdot \frac{1}{v_{uas}} = \boldsymbol{s/photo}$$

A.3. Procedure to reject hot-spots

The procedure for rejecting hot-spots goes as follows:

1. First of all it is asked the number of hot-spots detected (if the number is 0, it is accepted automatically the hot-spot as new).
2. For the first hot-spot on the list, the MMa creates a 3m (reconfigurable) side square around it and computes if the new hot-spot position is inside that area or not.
3. If the hot-spot is inside the area, it is rejected as a new. On the contrary, if the hot-spot is outside, the procedure of creating a square is repeated for the next hot-spot of the list.
4. The process ends when the MMa has a match (hot-spot rejected) or it has analysed all hot-spots and none of them were 3m near the new hot-spot (hot-spot is accepted as a new one).

ANNEX B: INTERACTION WITH THE FLIGHT CATEGORY

B.1. Scan Pattern Update

- Example of form:

```
<?xml version="1.0" encoding="utf-8" ?>
<fpu:FlightPlanUpdate
xmlns:fpu='http://icarus.upc.es/schema/FlightPlanUpdate/1.1'
xmlns:fp='http://icarus.upc.es/schema/FlightPlan/1.1'
xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'
xsi:schemaLocation='http://icarus.upc.es/schema/FlightPlanUpdate/1.1
file:C:\FlightPlanUpdate-1.1.xsd'>
  <Change>
    <MainFP targetId='FPBasico'>
      <stages>
        <stage targetId='traffic'>
          <legs>
            <leg targetId="basicscan" xsi:type="fp:BasicScanLeg">
              <dest>
                <coordinates>
                  72.20 3.75
                </coordinates>
                <altitude>2000</altitude>
                <speed>30</speed>
              </dest>
              <angle>0.18</angle>
              <dim1>500</dim1>
              <dim2>200</dim2>
              <separation>20</separation>
            </leg>
          </legs>
        </stage>
      </stages>
    </MainFP>
  </Change>
</fpu:FlightPlanUpdate>
```

- Parameters:

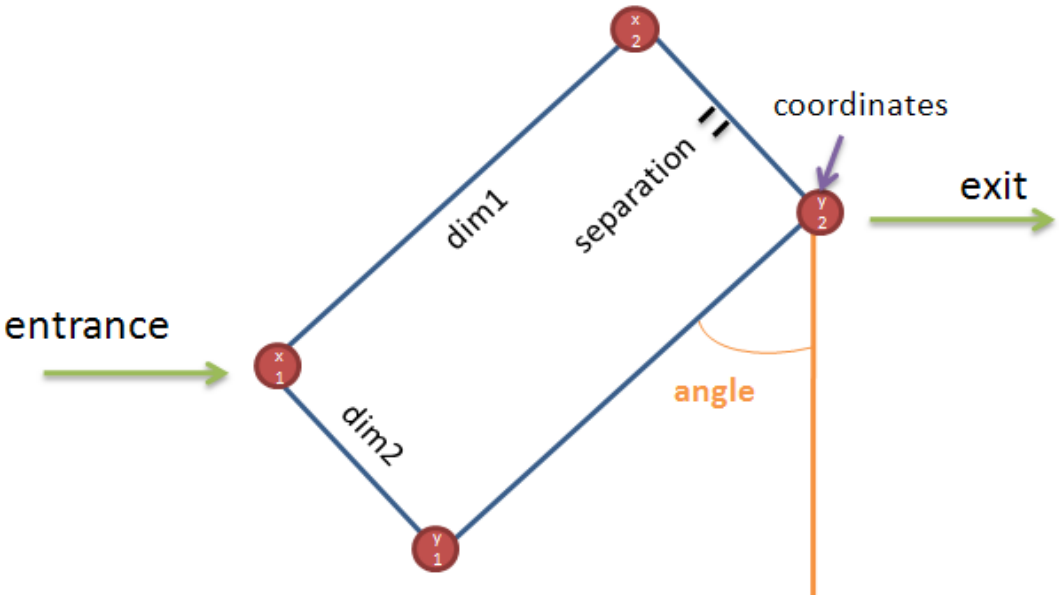


Figure Annex B.1.37 Scan pattern parameters

Parameter	Definition	Example	Value given for the MMa
MainFP	Name of the flight plan	MainFP targetId='FPBasico'	This parameter is defined by the FPMa in the moment of the creation of the flight plan
Stage	Name of the flight stage	stage targetId='traffic'	This parameter is defined by the FPMa in the moment of the creation of the flight plan
Leg	Name and type of leg	leg targetId="basicscan" xsi:type="fp:BasicScanLeg"	This parameter is defined by the FPMa in the moment of the creation of the flight plan
Coordinates	Last coordinate of the scan	72.20 43.75	The MMa sends the appropriate coordinate of the AoI
Altitude	Altitude to perform the scan in meters	2000	It is used the altitude that the UAV has in the moment of the update
Speed	Speed to perform the scan in meters/second	30	It is used the speed that the UAV has in the moment of the update
Angle	Angle of inclination of the scan pattern to respect the north in degrees	0.18	The MMa computes with trigonometric its value to send it in the update
dim1	Distance between the coordinate of entrance to the scan and the second one in meters	500	The MMa computes with trigonometric its value to send it in the update
dim2	Distance between the coordinate of entrance to the scan and the third one in meters	200	The MMa computes with trigonometric its value to send it in the update
Separation	Separation between the different legs when doing the scan in meters	20	The MMa computes its value (see Annex A.3) to send it in the update

Table Annex B.1.17 Scan pattern parameters

B.2. Hold Pattern Update

- Example of form:

```
<?xml version="1.0" encoding="utf-8" ?>
<fpu:FlightPlanUpdate
xmlns:fpu='http://icarus.upc.es/schema/FlightPlanUpdate/1.1'
xmlns:fp='http://icarus.upc.es/schema/FlightPlan/1.1'
xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'
xsi:schemaLocation='http://icarus.upc.es/schema/FlightPlanUpdate/1.1
file:C:\FlightPlanUpdate-1.1.xsd'>
 <Change>
  <MainFP targetId='FPBasico'>
   <stages>
    <stage targetId='traffic'>
     <legs>
      <leg targetId="hold" xsi:type="fp:HFLeg">
       <dest>
        <coordinates>
         4.2 2.88
        </coordinates>
        <altitude>2000</altitude>
        <speed>30</speed>
       </dest>
       <course>220</course>
       <direction>Right</direction>
       <d1>200</d1>
       <d2>250</d2>
      </leg>
     </legs>
    </stage>
   </stages>
  </MainFP>
 </Change>
</fpu:FlightPlanUpdate>
```

- Parameters:

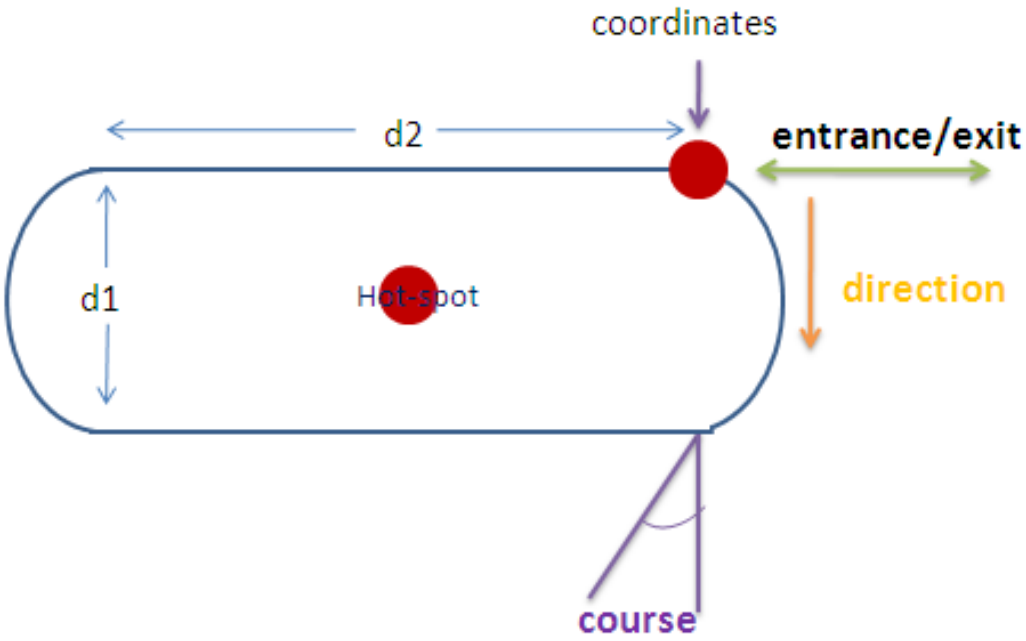


Figure Annex B.2.38 Hold pattern parameters

Parameter	Definition	Example	Value given for the MMa
MainFP	Name of the flight plan	MainFP targetId='FPBasico'	This parameter is defined by the FPMa in the moment of the creation of the flight plan
Stage	Name of the flight stage	stage targetId='depart'	This parameter is defined by the FPMa in the moment of the creation of the flight plan
Leg	Name and type of leg	leg targetId="**hold**" xsi:type="**fp:HFLeg**""	This parameter is defined by the FPMa in the moment of the creation of the flight plan.
Coordinates	Last coordinate of the hold	4.2 2.88	The MMa sends a coordinate of a hot-spot found or reported modified to convert it in the final hold coordinate
Altitude	Altitude to perform the scan in meters	2000	It is used a low altitude
Speed	Speed to perform the scan in meters/second	30	It is used the speed that the UAV has in the moment of the update
Course	Angle of inclination of the holding pattern to respect the north in degrees	200	The MMa uses some predefined value (0º)
Direction	Direction of the hold pattern	Right	The MMa uses some predefined value (Right)
d1	Distance between the coordinate of entrance to the scan and the second one in meters	200	The MMa uses some predefined value to perform the scan (200)
d2	Distance between the coordinate of entrance to the hold and the third one in meters	250	The MMa uses some predefined value to perform the scan (250)

Table Annex B.2.18 Hold pattern parameters

B.3. Eight Pattern Update

- Example of form:

```
<?xml version="1.0" encoding="utf-8" ?>
<fpu:FlightPlanUpdate
xmlns:fpu='http://icarus.upc.es/schema/FlightPlanUpdate/1.1'
xmlns:fp='http://icarus.upc.es/schema/FlightPlan/1.1'
xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'
xsi:schemaLocation='http://icarus.upc.es/schema/FlightPlanUpdate/1.1
file:C:\FlightPlanUpdate-1.1.xsd'>
 <Change>
  <MainFP targetId="FPBasico">
   <stages>
    <stage targetId="traffic">
     <legs>
      <leg targetId="scanPoint" xsi:type="fp:ScanPointLeg">
       <dest>
        <coordinates>
                41.455 2.175
               </coordinates>
        <altitude>200</altitude>
        <speed>100</speed>
       </dest>
       <course>220</course>
       <d1>200</d1>
       <d2>250</d2>
      </leg>
     </legs>
    </stage>
   </stages>
  </MainFP>
 </Change>
</fpu:FlightPlanUpdate>
```

- Parameters:

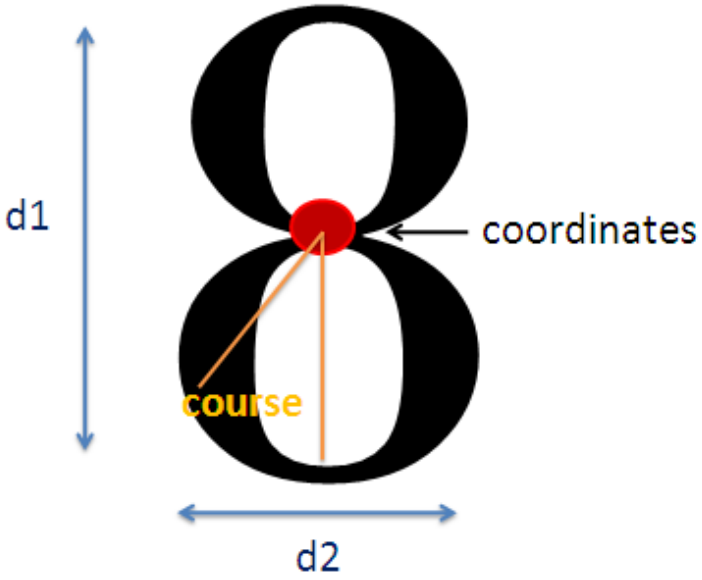


Figure Annex B.3.39 Eight pattern parameters

Parameter	Definition	Example	Value given for the MMa
MainFP	Name of the flight plan	MainFP targetId='FPBasico'	This parameter is defined by the FPMa in the moment of the creation of the flight plan
Stage	Name of the flight stage	stage targetId='traffic'	This parameter is defined by the FPMa in the moment of the creation of the flight plan
Leg	Name and type of leg	leg targetId="**scanpoint**" xsi:type="**fp:ScanPointLeg**""	This parameter is defined by the FPMa in the moment of the creation of the flight plan
Coordinates	Last coordinate of the hold	41.455 2.175	The MMa sends a coordinate of a hot-spot found or reported modified to convert it in the final hold coordinate
Altitude	Altitude to perform the scan in meters	200	It is used a low altitude
Speed	Speed to perform the scan in meters/second	100	It is used the speed that the UAV has in the moment of the update
Course	Angle of inclination of the eight pattern to respect the vertical in degrees	220	The MMa uses some predefined value (0º)
d1	Distance between the coordinate of entrance to the scan and the second one in meters	200	The MMa uses some predefined value to perform the scan (200)
d2	Distance between the coordinate of entrance to the hold and the third one in meters	250	The MMa uses some predefined value to perform the scan (250)

Table Annex B.3.19 Eight pattern parameters

B.4. SetCondition

In order to indicate to the FPMa the next branch of an iterative loop we have to send it a SetCondition.

In it, we have to introduce the name of the iterative leg (in the example condition 1) and the number of the pattern (3 in the example).

Setcondition.condId = "condition1"
Setcondition.condRes = 3

The number is defined in the FPMa. When a FP is created, the user specifies a sequence of scan patterns that can be performed in the iterative loop, for example hold scan eight. Then, in order to establish that an eight pattern is wanted, we have to indicate the number of introduction of the eight pattern in the FP. For example, in the list of hold scan eight that has just been seen, we have to indicate in the SetCondition that the number is 3.

B.5. Skip

Skip is used to end the actual branch of an iterative leg. The message to send to the FPMa is quite simple as we just have to send a skip message without any parameter.

B.6. GoToLeg

When we want to end an iterative loop we use a GoToLeg message. It is going to be used an example:

```
leg.legId = "L4"
leg.stageId = "traffic"
```

As it can be observed we have to indicate the name of the leg that goes after the iterative leg and the stage that is contained that leg. Both identifications are determined when the FP is created.

ANNEX C: MISSION XML

The next example shows a mission created with the Mission Monitor Setup:

```
<Mission>
- <Mission Name="Example of Mission">
  - <State Name="Initialize">
    - <Services>
      <Service>MMaService</Service>
      <Service>VirtualAutopilotLiteService</Service>
      <Service>FlightMonitorLiteService</Service>
      <Service>DEM</Service>
      <Service>FlightPlanMonitorService</Service>
      <Service>FlightPlanManagerService</Service>
      <Service>Storage</Service>
      <Service>RTDP</Service>
    </Services>
    <Tilt>15</Tilt>
    <X_Cam_angle>25</X_Cam_angle>
    <Y_Cam_angle>18.8</Y_Cam_angle>
    <Photo_filter>0.2</Photo_filter>
  </State>
  - <State Name="Take-off">
    <TransitionAltitude>30</TransitionAltitude>
  </State>
  - <State Name="EnRouteOb">
    - <Area>
      <x1 latitude="41.37" longitude="2.07" />
      <x2 latitude="41.37" longitude="2.15" />
      <y2 latitude="41.34" longitude="2.15" />
      <y1 latitude="41.34" longitude="2.07" />
    </Area>
    <TimeServices>120</TimeServices>
    <NextStat>reconnaissance</NextStat>
  </State>
  - <State Name="Reconnaissance">
    <ScanPattern>scan</ScanPattern>
    <VisualCam>false</VisualCam>
    <FalseAlarmDetector>false</FalseAlarmDetector>
    <Overlapping>60</Overlapping>
    <NextStat>detailed</NextStat>
  </State>
  - <State Name="Detailed">
    - <HotSpotList Value="True">
      <HotSpot latitude="41.360000610351562" longitude="2.0799999237060547" />
    </HotSpotList>
    <ScanPattern>hold</ScanPattern>
    <Fusion>true</Fusion>
    <FalseAlarmDetector>true</FalseAlarmDetector>
    <RejectDistance>3</RejectDistance>
    <Overlapping>60</Overlapping>
    <NextStat>enrouteland</NextStat>
  </State>
```

```
  <State Name="EnRouteLand" />
  - <State Name="Landing">
    <NextStat>Save</NextStat>
  </State>
  <State Name="Save" />
  <State Name="End" />
</Mission>
```

ANNEX D: MISSION INTERFACE DESCRIPTION

Objects	Variables	Description
Area	Four positions	• Instead of sending 4 points individually, it is created an object Area to send the AoI to the MMa
Hot-spot	One position and one number	• The object Hot-spot is used for adding, modifying and removing a hot-spot • The position is used to indicate the position of the hot-spot • The number indicates the position of the hot-spot inside the list of hot-spots
State Objects (Detector, NextState, Overlapping, Pattern, Reject Distance and Visual Camera)	There are three variables. In all of them appear a string and a number. The third depends on the object.	• The State Objects are used to reconfigure state parameters (e.g. the next state) • The common string is used to indicate the state in which we want to apply the change • The common number, for its part, is used to indicate the state number, in case that we have chosen a state that is repeated (e.g. 2 Reconnaissance states) • The variable that depends on the object is the new value that we want to apply the reconfiguration. • In the case of the Detector there is a Boolean to indicate if we want the false alarm detector enabled • In the case of the NextState, a string to indicate the name of the next state • In the Overlapping there is a number to indicate the new value of the overlapping • In the Reject Distance there is also a number to indicate the reconfiguration on the reject distance • The Visual Camera contains a Boolean to state that we want the cameras on/off

Table Annex D.20 Objects and variables of the Mission Interface

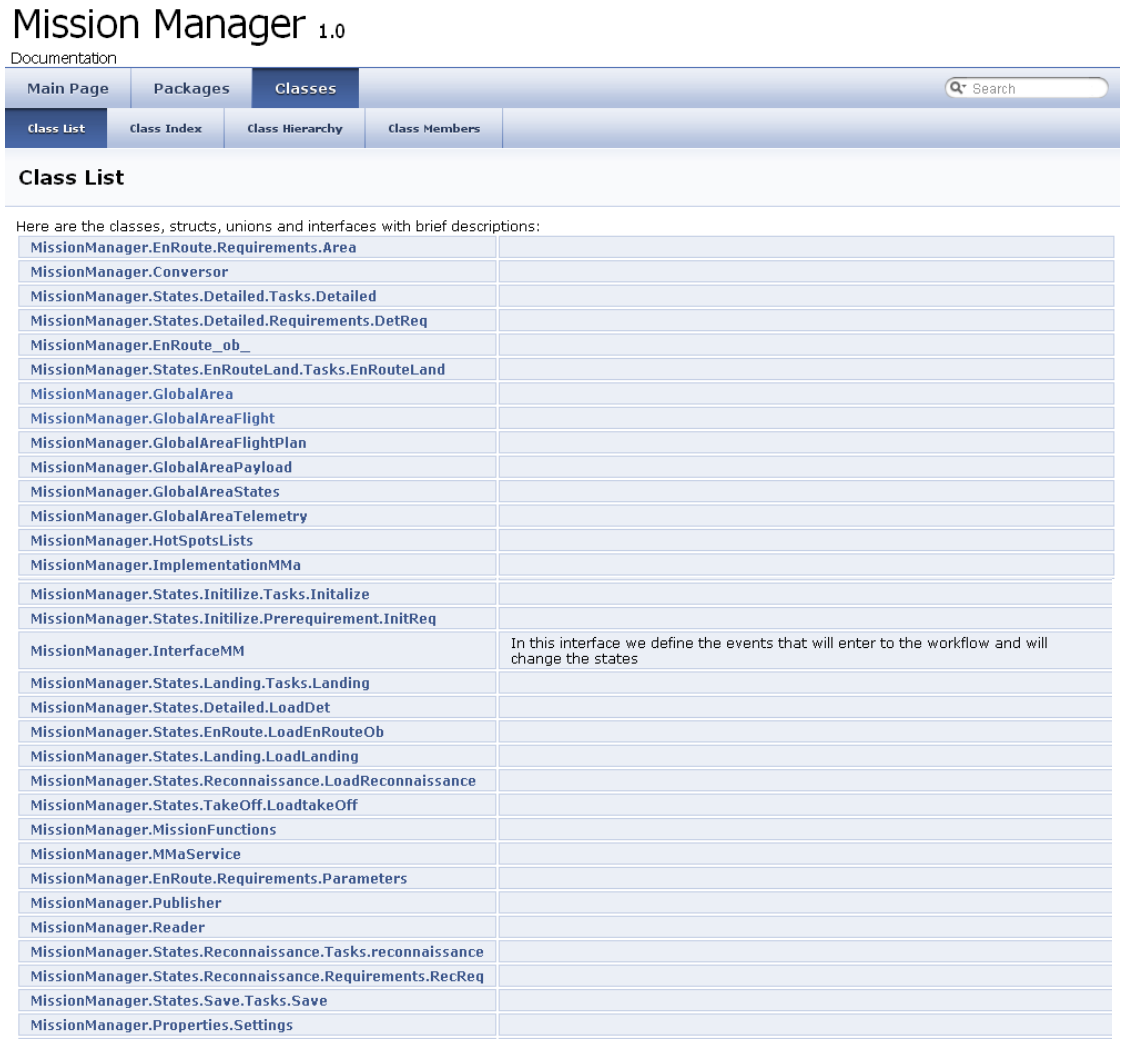
ANNEX E: MISSION CATEGORY DOCUMENTATION

The code of the Mission Category has been documented in XML and Doxygen [10]. Doxygen is a free program that converts XML comments to an accurate, informative documentation with a common look and feel.

The complete documentation of the Mission Category is in a HTML format for its simplicity and visual representation. Printing the whole documentation in this project would have not been practical and therefore it has been included in the CD attached instead. Nevertheless, the next two subsections shows abstracts of the documentation of the MMa and the MMo. It has been included the list of the classes, the functions, the variables, the properties, the events and one representative class of each service that will let making a good idea of the whole code documentation.

E.1. Mission Manager Documentation

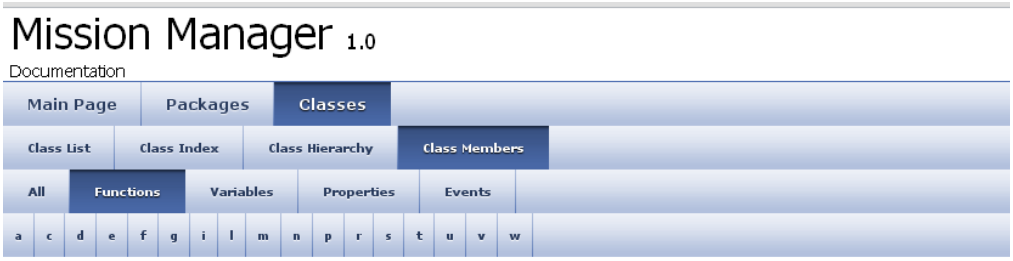
The following figure shows the list of classes of the MMa:



MissionManager.Takeoff	
MissionManager.TakeOff.Requirements.TakeOffReq	
MissionManager.Workflow1	This class executes the actions that the WF has implemented

Figure Annex E.1.40 MMa classes documentation

The following figure shows the functions used in the MMa:



- **a** -

- Add_container() : MissionManager.Publisher
- AddCameras() : MissionManager.Publisher
- AnalizeHotSpot() : MissionManager.MissionFunctions
- Area() : MissionManager.EnRoute.Requirements.Area

- **c** -

- ChangeDim1() : MissionManager.MissionFunctions
- ChargeParameters() : MissionManager.Reader
- CloseService() : MissionManager.Publisher
- ConDisService() : MissionManager.Publisher
- ConvertToUTM() : MissionManager.Conversor
- CreateArea() : MissionManager.EnRoute.Requirements.Area

- **d** -

- DetReqMet() : MissionManager.States.Detailed.Requirements.DetReq

- **e** -

- EventFired() : MissionManager.MMaService

- **f** -

- FunctionCall() : MissionManager.MMaService

- **g** -

- getActualPositioninUTM() : MissionManager.GlobalAreaTelemetry
- GetDescription() : MissionManager.MMaService
- gethightakeoffaltitude() : MissionManager.TakeOff.Requirements.TakeOffReq
- getInstance() : MissionManager.GlobalArea , MissionManager.States.EnRoute.LoadEnRouteOb , MissionManager.HotSpotsLists , MissionManager.Publisher , MissionManager.States.Initilize.Prerequirement.InitReq , MissionManager.States.Landing.LoadLanding , MissionManager.Reader , MissionManager.States.Reconnaissance.LoadReconnaissance , MissionManager.States.TakeOff.LoadtakeOff , MissionManager.States.Detailed.LoadDet
- getlowtakeoffaltitude() : MissionManager.TakeOff.Requirements.TakeOffReq
- GlobalAreaTelemetry() : MissionManager.GlobalAreaTelemetry
- Gotodetailed() : MissionManager.ImplementationMMa
- Gotoend() : MissionManager.ImplementationMMa
- Gotoenrouteland() : MissionManager.ImplementationMMa
- Gotolanding() : MissionManager.ImplementationMMa
- GoToLegID() : MissionManager.GlobalAreaFlightPlan
- Gotoreconnaissance() : MissionManager.ImplementationMMa
- Gotosave() : MissionManager.ImplementationMMa

- **i** -

- InitilizeRun() : MissionManager.Workflow1
- Is_inside() : MissionManager.EnRoute.Requirements.Area
- Is_insideBig() : MissionManager.EnRoute.Requirements.Area
- IsInside() : MissionManager.MissionFunctions

- **l** -

- Leavinginit() : MissionManager.ImplementationMMa
- Leavingtakeoff() : MissionManager.ImplementationMMa

- **m** -

- MetAreareq() : MissionManager.EnRoute.Requirements.Area
- MetEnRouteState() : MissionManager.EnRoute.Requirements.Parameters
- MissionFunctions() : MissionManager.MissionFunctions
- modify_hot_spot() : MissionManager.HotSpotsLists

- n -

- NumberOfHotSpots() : MissionManager.Publisher

- p -

- Parameters() : MissionManager.EnRoute.Requirements.Parameters
- PhotoTime() : MissionManager.MissionFunctions
- PublishDetector() : MissionManager.Publisher
- PublishScan() : MissionManager.Publisher
- PublishState() : MissionManager.Publisher

- r -

- RecReq() : MissionManager.States.Reconnaissance.Requirements.RecReq
- Remove_hot_spot() : MissionManager.HotSpotsLists
- ReturnEight() : MissionManager.GlobalAreaFlightPlan
- ReturnHold() : MissionManager.GlobalAreaFlightPlan
- ReturnScan() : MissionManager.GlobalAreaFlightPlan
- ReturnScannDEight() : MissionManager.GlobalAreaFlightPlan
- Run() : MissionManager.States.Landing.Tasks.Landing , MissionManager.States.Landing.LoadLanding , MissionManager.States.Initilize.Tasks.Initalize , MissionManager.EnRoute_ob_ , MissionManager.States.EnRoute.LoadEnRouteOb , MissionManager.States.EnRouteLand.Tasks.EnRouteLand , MissionManager.States.Detailed.Tasks.Detailed , MissionManager.States.Detailed.LoadDet , MissionManager.Reader , MissionManager.States.TakeOff.LoadtakeOff , MissionManager.States.Save.Tasks.Save , MissionManager.Takeoff , MissionManager.States.Reconnaissance.LoadReconnaissance , MissionManager.States.Reconnaissance.Tasks.reconnaissance

- s -

- SelectFP() : MissionManager.Publisher
- Separation_Distance() : MissionManager.MissionFunctions
- SkipFP() : MissionManager.Publisher
- Start() : MissionManager.MMaService
- StartService() : MissionManager.Publisher , MissionManager.EnRoute_ob_
- Stop() : MissionManager.MMaService
- StopService() : MissionManager.Publisher

- t -

- TakeOffReq() : MissionManager.TakeOff.Requirements.TakeOffReq
- TakeOffreqMet() : MissionManager.TakeOff.Requirements.TakeOffReq
- TakePhotos() : MissionManager.Publisher
- ThermalReqMet() : MissionManager.States.Reconnaissance.Requirements.RecReq
- TODetailed() : MissionManager.Workflow1
- TOEnRouteLand() : MissionManager.Workflow1
- TOEnRouteOb() : MissionManager.Workflow1
- TOLanding() : MissionManager.Workflow1
- TORun() : MissionManager.Workflow1
- TOSave() : MissionManager.Workflow1
- TOThermal() : MissionManager.Workflow1

- u -

- UpdateFP() : MissionManager.Publisher

- v -

- VariableChanged() : MissionManager.MMaService

- w -

- Workflow1() : MissionManager.Workflow1

Figure Annex E.1.41 MMa functions documentation

The following figure shows the variables of the MMa:

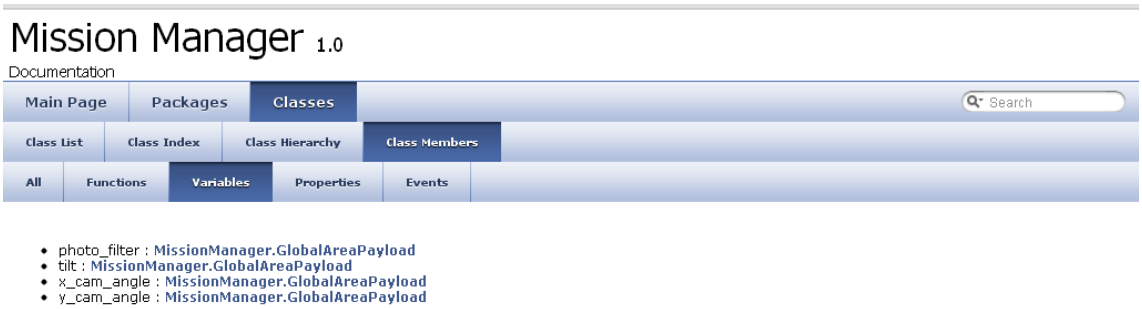


Figure Annex E.1.42 MMa variables documentation

The following figure shows the properties of the MMa:

Mission Manager 1.0

Documentation

Main Page | Packages | Classes | Search

Class List | Class Index | Class Hierarchy | Class Members

All | Functions | Variables | Properties | Events

a c d e f g h i n o p r s t v x y

- **a** -
  - Ack : MissionManager.GlobalAreaFlight
  - ActualPosition : MissionManager.GlobalAreaTelemetry
  - ActualSpeed : MissionManager.GlobalAreaTelemetry
  - ActualState : MissionManager.GlobalAreaStates
  - Angle_Scan : MissionManager.GlobalAreaFlight
- **c** -
  - Cont_det : MissionManager.GlobalAreaStates
  - Cont_rec : MissionManager.GlobalAreaStates
  - Coordx : MissionManager.GlobalAreaFlight
  - Coordy : MissionManager.GlobalAreaFlight
- **d** -
  - Detector : MissionManager.GlobalAreaPayload
  - Dim1 : MissionManager.GlobalAreaFlight
  - Dim2 : MissionManager.GlobalAreaFlight
- **e** -
  - EndScan : MissionManager.GlobalAreaFlight
- **f** -
  - Fusion : MissionManager.GlobalAreaPayload
- **g** -
  - guidWF : MissionManager.GlobalArea
- **h** -
  - HotSpotAlarm : MissionManager.GlobalAreaPayload
- **i** -
  - Imp_State : MissionManager.GlobalAreaStates
- **n** -
  - NextStat : MissionManager.GlobalAreaStates
  - Number_States_Detailed : MissionManager.GlobalAreaStates
  - Number_States_Reconnaissance : MissionManager.GlobalAreaStates
- **o** -
  - Overlapping : MissionManager.GlobalAreaPayload
- **p** -
  - Pattern : MissionManager.GlobalAreaPayload
  - PositionTO : MissionManager.GlobalAreaFlight
- **r** -
  - Reject_distance : MissionManager.GlobalAreaPayload
- **s** -
  - Separation : MissionManager.GlobalAreaFlight
  - State_number : MissionManager.GlobalAreaStates
- **t** -
  - Time_Services : MissionManager.GlobalAreaPayload
  - Transalt : MissionManager.GlobalAreaStates
- **v** -
  - VarDim1 : MissionManager.GlobalAreaFlight
  - Vis_Cam : MissionManager.GlobalAreaPayload
- **x** -
  - x1Pos : MissionManager.GlobalAreaStates
  - x2Pos : MissionManager.GlobalAreaStates
- **y** -
  - y1Pos : MissionManager.GlobalAreaStates
  - y2Pos : MissionManager.GlobalAreaStates

Figure Annex E.1.43 MMa properties documentation

The next figure shows the MMa events:

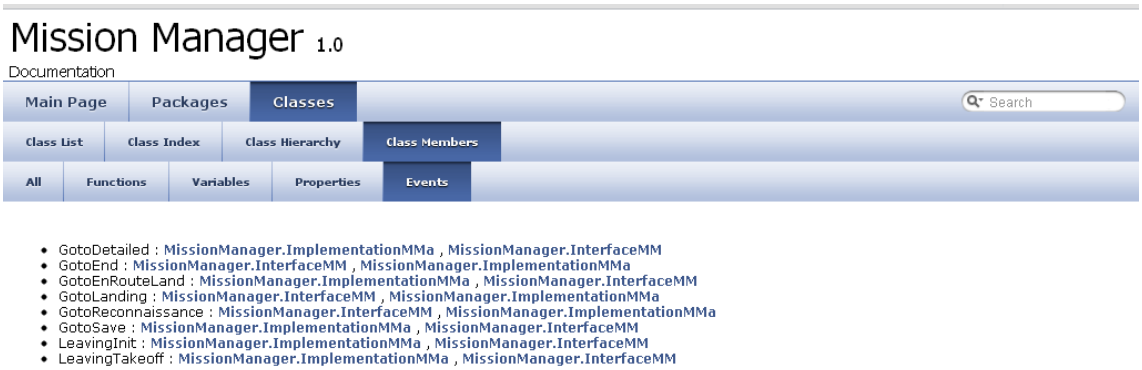
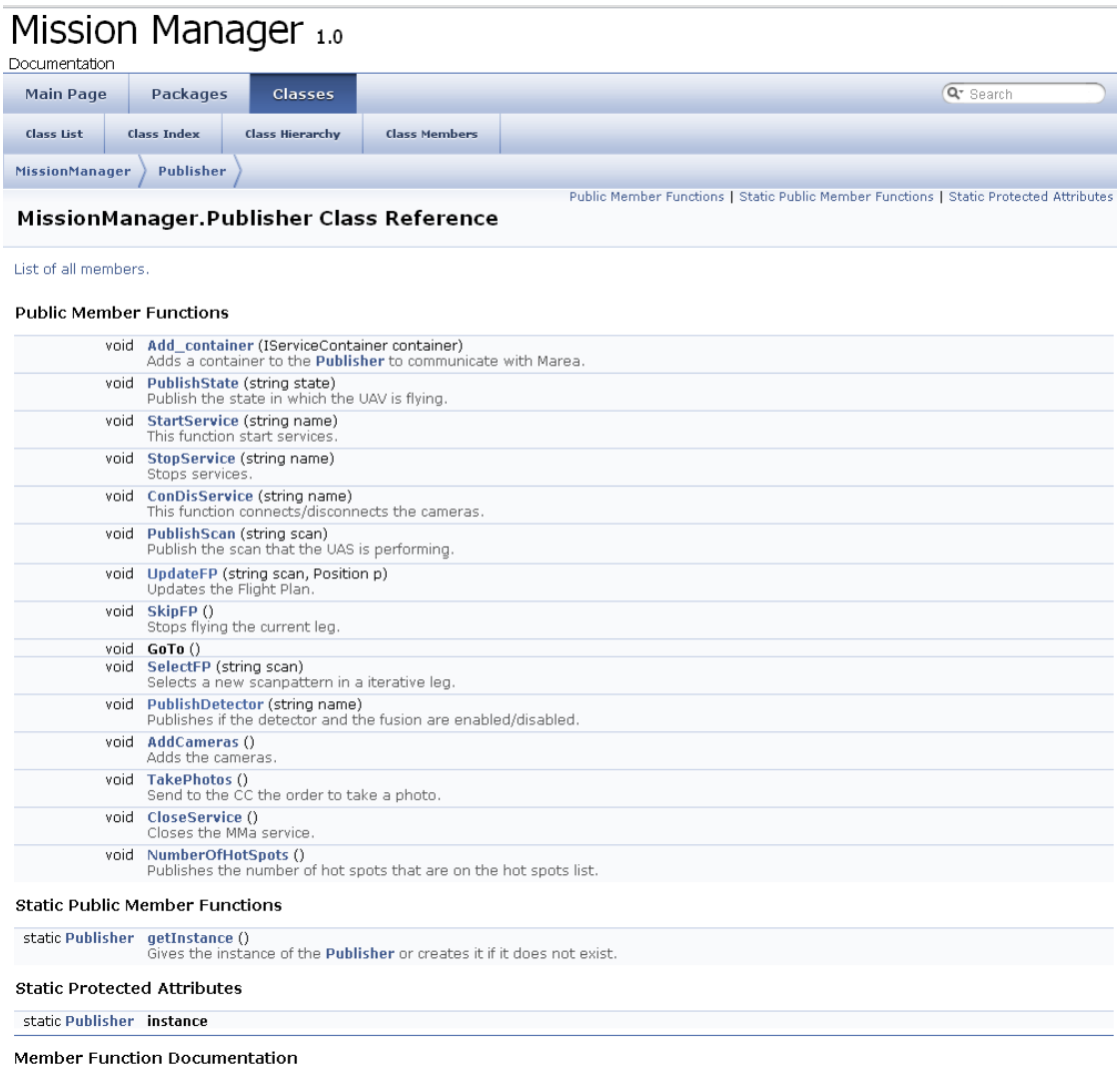


Figure Annex E.1.44 MMa events documentation

Next, it will be showed one class that will exemplify the documentation produced for the classes of the MMa. The class elected is the Publisher. This class sends most of the events that the MMa produces:



void MissionManager.Publisher.Add_container (IServiceContainer container)

Adds a container to the Publisher to communicate with Marea.

Parameters:
container the container

void MissionManager.Publisher.AddCameras ( )

Adds the cameras.

void MissionManager.Publisher.CloseService ( )

Closes the MMa service.

void MissionManager.Publisher.ConDisService (string name)

This function connects/disconnects the cameras.

Parameters:
name connect|disconnect

static Publisher MissionManager.Publisher.getInstance ( ) [static]

Gives the instance of the Publisher or creates it if it does not exist.

Returns:
the publisher instance

void MissionManager.Publisher.NumberOfHotSpots ( )

Publishes the number of hot spots that are on the hot spots list.

void MissionManager.Publisher.PublishDetector (string name)

Publishes if the detector and the fusion are enabled/disabled.

Parameters:
name detector_on detector_off fusion_on fusion_off

void MissionManager.Publisher.PublishScan (string scan)

Publish the scan that the UAS is performing.

Parameters:
scan the name of the scan

void MissionManager.Publisher.PublishState (string state)

Publish the state in which the UAV is flying.

Parameters:
state the name of the state

void MissionManager.Publisher.SelectFP (string scan)

Selects a new scanpattern in a iterative leg.

/

Parameters:
scan the name of the scan that is is wanted to select

void MissionManager.Publisher.SkipFP ( )

Stops flying the current leg.

void MissionManager.Publisher.StartService (string name)

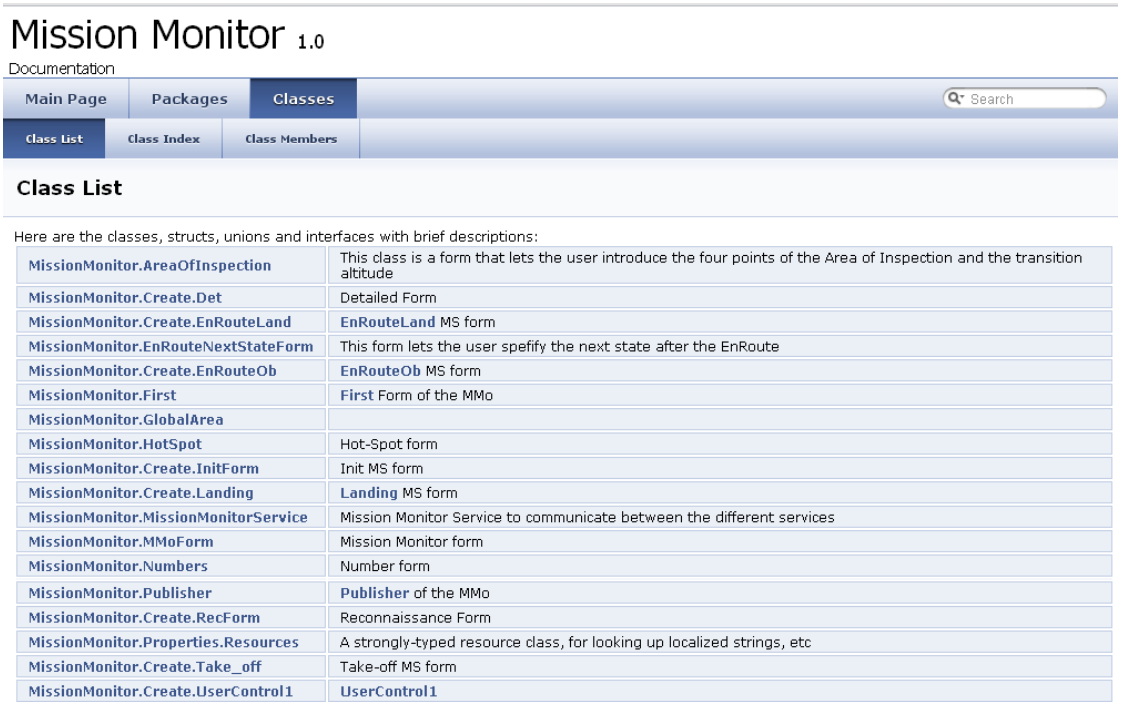
This function start services.

Parameters:
name the service is wanted to turn on

Figure Annex E.1.45 Publisher class documentation

E.2. Mission Monitor Documentation

The following figure shows the list of classes of the MMo:



Mission Monitor 1.0

Documentation

Main Page | Packages | Classes | Search

Class List | Class Index | Class Members

Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

MissionMonitor.AreaOfInspection	This class is a form that lets the user introduce the four points of the Area of Inspection and the transition altitude
MissionMonitor.Create.Det	Detailed Form
MissionMonitor.Create.EnRouteLand	EnRouteLand MS form
MissionMonitor.EnRouteNextStateForm	This form lets the user spefify the next state after the EnRoute
MissionMonitor.Create.EnRouteOb	EnRouteOb MS form
MissionMonitor.First	First Form of the MMo
MissionMonitor.GlobalArea	
MissionMonitor.HotSpot	Hot-Spot form
MissionMonitor.Create.InitForm	Init MS form
MissionMonitor.Create.Landing	Landing MS form
MissionMonitor.MissionMonitorService	Mission Monitor Service to communicate between the different services
MissionMonitor.MMoForm	Mission Monitor form
MissionMonitor.Numbers	Number form
MissionMonitor.Publisher	Publisher of the MMo
MissionMonitor.Create.RecForm	Reconnaissance Form
MissionMonitor.Properties.Resources	A strongly-typed resource class, for looking up localized strings, etc
MissionMonitor.Create.Take_off	Take-off MS form
MissionMonitor.Create.UserControl1	UserControl1

Figure Annex E.2.46 MMo classes documentation

The following figure shows the functions used in the MMo:

Mission Monitor 1.0

Documentation

Main Page | Packages | Classes | Search

Class List | Class Index | Class Members

All | Functions | Variables | Properties

_ | a | c | d | e | f | g | h | i | l | n | r | s | t | v

- _ -

- _closeForm() : MissionMonitor.MMoForm
- _Detector() : MissionMonitor.MMoForm
- _Fusion() : MissionMonitor.MMoForm
- _NumberHotspots() : MissionMonitor.MMoForm
- _Patterns() : MissionMonitor.MMoForm
- _Ther() : MissionMonitor.MMoForm
- _Vis() : MissionMonitor.MMoForm

- a -

- Add_container() : MissionMonitor.First , MissionMonitor.MMoForm , MissionMonitor.Publisher
- AreaOfInspection() : MissionMonitor.AreaOfInspection

- **c** -

  - changeDetector() : **MissionMonitor.MMoForm**
  - changeFusion() : **MissionMonitor.MMoForm**
  - changeNumberHot() : **MissionMonitor.MMoForm**
  - changePattern() : **MissionMonitor.MMoForm**
  - changeStatus() : **MissionMonitor.MMoForm**
  - changeTher() : **MissionMonitor.MMoForm**
  - changeVis() : **MissionMonitor.MMoForm**
  - CloseForm() : **MissionMonitor.MMoForm**
  - CloseService() : **MissionMonitor.Publisher**

- **d** -

  - Det() : **MissionMonitor.Create.Det** , **MissionMonitor.Create.EnRouteOb** , **MissionMonitor.Create.RecForm**
  - Dispose() : **MissionMonitor.HotSpot** , **MissionMonitor.AreaOfInspection** , **MissionMonitor.First** , **MissionMonitor.EnRouteNextStateForm** , **MissionMonitor.Create.Take_off** , **MissionMonitor.Create.RecForm** , **MissionMonitor.Create.Landing** , **MissionMonitor.Create.InitForm** , **MissionMonitor.Create.EnRouteOb** , **MissionMonitor.Create.EnRouteLand** , **MissionMonitor.Create.Det** , **MissionMonitor.Create.Landing** , **MissionMonitor.MMoForm** , **MissionMonitor.Numbers**

- **e** -

  - EnRoute() : **MissionMonitor.Create.EnRouteOb**
  - EnRouteLand() : **MissionMonitor.Create.Det** , **MissionMonitor.Create.RecForm** , **MissionMonitor.Create.EnRouteLand**
  - EnRouteNextStateForm() : **MissionMonitor.EnRouteNextStateForm**
  - EnRouteOb() : **MissionMonitor.Create.Take_off** , **MissionMonitor.Create.EnRouteOb**
  - EventFired() : **MissionMonitor.MissionMonitorService**

- **f** -

  - First() : **MissionMonitor.First**
  - FunctionCall() : **MissionMonitor.MissionMonitorService**

- **g** -

  - GetDescription() : **MissionMonitor.MissionMonitorService**
  - getInstance() : **MissionMonitor.Publisher** , **MissionMonitor.GlobalArea** , **MissionMonitor.MMoForm**

- **h** -

  - Hot() : **MissionMonitor.Create.Det**
  - HotSpot() : **MissionMonitor.HotSpot**

- **i** -

  - InitForm() : **MissionMonitor.Create.InitForm**
  - InitializeComponent() : **MissionMonitor.Create.UserControl1**

- **l** -

  - Land() : **MissionMonitor.Create.EnRouteLand**
  - Landing() : **MissionMonitor.Create.Landing**

- **n** -

  - Numbers() : **MissionMonitor.Numbers**

- **r** -

  - Rec() : **MissionMonitor.Create.Det** , **MissionMonitor.Create.EnRouteOb**
  - RecForm() : **MissionMonitor.Create.RecForm**
  - Run() : **MissionMonitor.MissionMonitorService**
  - RunCreate() : **MissionMonitor.First**
  - RunFollow() : **MissionMonitor.First**
  - RunHotSpot() : **MissionMonitor.MMoForm**
  - RunLittle() : **MissionMonitor.MMoForm**
  - RunNumbers() : **MissionMonitor.MMoForm**

- **s** -

  - Start() : **MissionMonitor.MissionMonitorService**
  - StartWriter() : **MissionMonitor.GlobalArea**
  - Stop() : **MissionMonitor.MissionMonitorService**

- **t** -

  - Take_off() : **MissionMonitor.Create.Take_off**
  - TO() : **MissionMonitor.Create.InitForm**

- **v** -

  - VariableChanged() : **MissionMonitor.MissionMonitorService**

Figure Annex E.2.47 MMo functions documentation

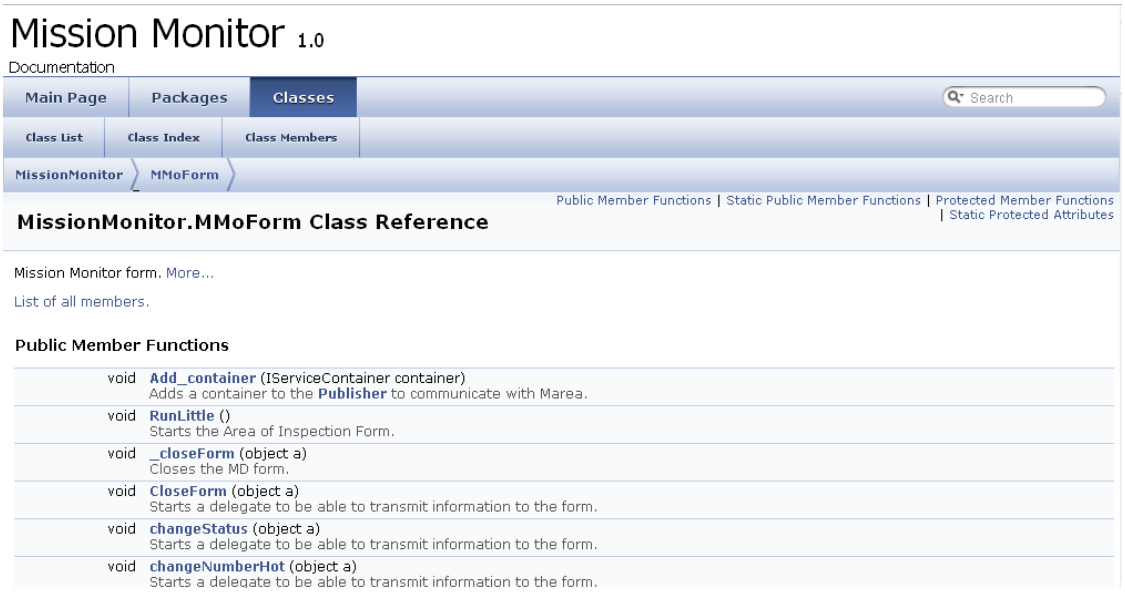
The following figure shows the variables of the MMo:

Figure Annex E.2.48 MMo variables documentation

The following figure shows the properties of the MMo:

Figure Annex E.2.49 MMo properties documentation

This time as a class documentation representative for the classes of the MMo, it has been chosen the main form of the MD. Next figure shows its documentation:



void	**_NumberHotspots** (object a) Informs of the number of hot spots.
void	**_Vis** (object a) This function informs if the visual cameras are connected/disconnected.
void	**changeVis** (object a) Starts a delegate to be able to transmit information to the form.
void	**_Ther** (object a) This function informs if the themal cameras are connected/disconnected.
void	**changeTher** (object a) Starts a delegate to be able to transmit information to the form.
void	**_Fusion** (object a) Function to show the Fusion.
void	**changeFusion** (object a) Starts a delegate to be able to transmit information to the form.
void	**changePattern** (object a) Starts a delegate to be able to transmit information to the form.
void	**_Patterns** (object a) Function to show in the MMo the pattern that is following the UAV.
void	**_Detector** (object a) Function to show in the Detector.
void	**changeDetector** (object a) Starts a delegate to be able to transmit information to the form.
void	**RunNumbers** () Starts a "comodin" form.
void	**RunHotSpot** () Starts the **HotSpot** form.

Static Public Member Functions

static **MMoForm**	**getInstance** () Returns the instance of **GlobalArea** or creates it if it does not exist.

Protected Member Functions

override void	**Dispose** (bool disposing) Clean up any resources being used.

Static Protected Attributes

static **MMoForm**	**instance** Instance of the MMo.

Detailed Description

Mission Monitor form.

Member Function Documentation

void MissionMonitor.MMoForm._closeForm (object a)

Closes the MD form.

Parameters:
- a

void MissionMonitor.MMoForm._Detector (object a)

Function to show in the Detector.

Parameters:
- a the value of the detector

void MissionMonitor.MMoForm._Fusion (object a)

Function to show the Fusion.

Parameters:
- a the value of the fusion

void MissionMonitor.MMoForm._NumberHotspots (object a)

Informs of the number of hot spots.

Parameters:
- a the value of hot spots

void MissionMonitor.MMoForm._Patterns (object a)

Function to show in the MMo the pattern that is following the UAV.

Parameters:
- a the name of the pattern

void MissionMonitor.MMoForm._Ther (object a)

This function informs if the themal cameras are connected/disconnected.

Parameters:
- a the value of the fusion

void MissionMonitor.MMoForm._Vis (object a)

This function informs if the visual cameras are connected/disconnected.

Parameters:
a the value of the fusion

void MissionMonitor.MMoForm.Add_container (IServiceContainer container)

Adds a container to the Publisher to communicate with Marea.

Parameters:
container the container

void MissionMonitor.MMoForm.changeDetector (object a)

Starts a delegate to be able to transmit information to the form.

Parameters:
a

void MissionMonitor.MMoForm.changeFusion (object a)

Starts a delegate to be able to transmit information to the form.

Parameters:
a

void MissionMonitor.MMoForm.changeNumberHot (object a)

Starts a delegate to be able to transmit information to the form.

Parameters:
a

void MissionMonitor.MMoForm.changePattern (object a)

Starts a delegate to be able to transmit information to the form.

Parameters:
a

void MissionMonitor.MMoForm.changeStatus (object a)

Starts a delegate to be able to transmit information to the form.

Parameters:
a

void MissionMonitor.MMoForm.changeTher (object a)

Starts a delegate to be able to transmit information to the form.

Parameters:
a

void MissionMonitor.MMoForm.changeVis (object a)

Starts a delegate to be able to transmit information to the form.

Parameters:
a

void MissionMonitor.MMoForm.CloseForm (object a)

Starts a delegate to be able to transmit information to the form.

Parameters:
a

override void MissionMonitor.MMoForm.Dispose (bool disposing) [protected]

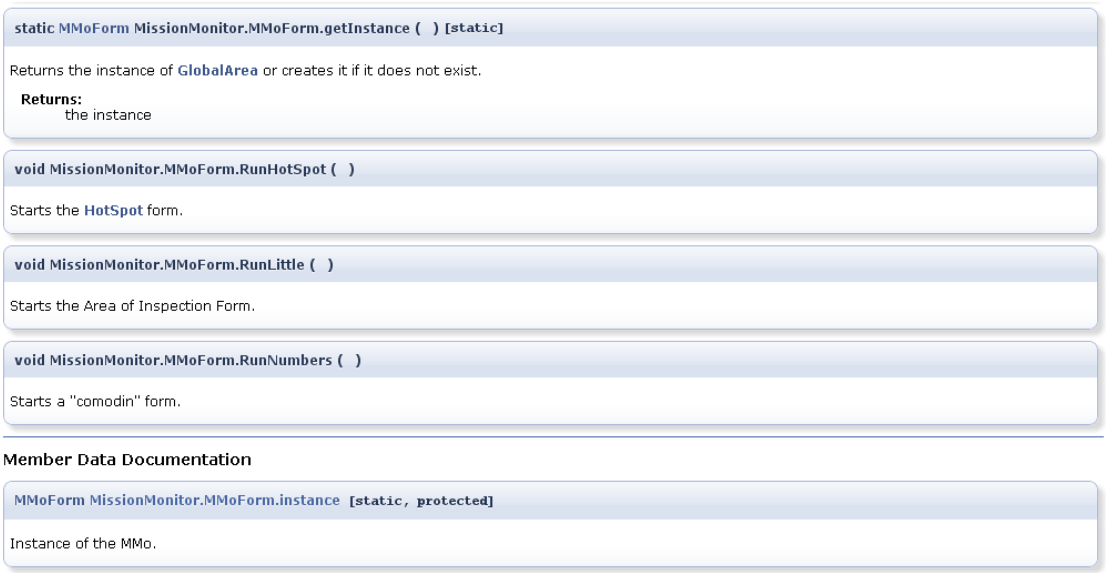
Clean up any resources being used.

Parameters:
disposing true if managed resources should be disposed; otherwise, false.

static MMoForm MissionMonitor.MMoForm.getInstance ( ) [static]

Returns the instance of GlobalArea or creates it if it does not exist.

Returns:
the instance



static MMoForm MissionMonitor.MMoForm.getInstance ( ) [static]

Returns the instance of GlobalArea or creates it if it does not exist.

Returns:
the instance

void MissionMonitor.MMoForm.RunHotSpot ( )

Starts the HotSpot form.

void MissionMonitor.MMoForm.RunLittle ( )

Starts the Area of Inspection Form.

void MissionMonitor.MMoForm.RunNumbers ( )

Starts a "comodin" form.

Member Data Documentation

MMoForm MissionMonitor.MMoForm.instance [static, protected]

Instance of the MMo.

Figure Annex E.2.50 MD class documentation