

A software architecture for Autonomous UAV Mission Management and Control

Paolo Gunetti¹, Tony Dodd² and Haydn Thompson³
University of Sheffield, Sheffield, UK, S1 3JD

During the last decade, Uninhabited Aerial Vehicles (UAVs) have become increasingly popular in a large variety of application fields, both in the military and civil sectors. As a consequence, significant efforts have been spent on UAV research. The “race” towards increased UAV autonomy captured a considerable part of these efforts. At present, UAVs are capable of carrying out on their own a pre-planned mission. However, supervision is still needed, especially in highly dynamic environments where the flight plan might need to be changed. In this paper, a software architecture for Autonomous UAV Mission Management and Control is introduced. This is implemented using a combination of Soar intelligent agents and traditional control techniques, and is designed to be able to work in real-time. The system can generate a Flight Plan, execute it and update it during execution in order to deal with changes in the perceived environment. The architecture is outlined in detail, including the definition of necessary abstractions, prior to a thorough description of the system components. The simulation environment is described and simulation results are presented for both the Planner agent and the Execution agent. Conclusions focus on the suitability of this approach to autonomous UAV flight and on future developments planned.

I. Introduction

UNINHABITED (or Unmanned) Aerial Vehicles (UAVs) were first introduced even before World War II, although their main function at the time was as target drones [1]. These were simple Radio-Controlled (RC) aircraft that focused on low cost and expendability, and were produced in large scale during the war. After WW2, UAVs gathered an increasing amount of interest and started being used for other functions. A whole new market was opened with the introduction of missiles, and in the 1960s the first reconnaissance UAVs were deployed. However, these were plagued by many issues and were not a credible solution for missions where expendability was not an asset, including of course all types of civilian usage. The problems included but were not limited to the following [2]:

- ensuring a radio link with sufficient bandwidth (both uplink for control and downlink for sensor information) as the range increases

- having on-board instrumentation allowing the remote pilot to get complete situational awareness
- designing complex control systems that could reduce the workload on the remote pilot
- implementing the control systems using the limited capabilities of available computer hardware
- dealing with safety issues which are worsened by the lack of situational awareness.

However, UAVs present a series of potential advantages over piloted aircraft [3]:

- operating costs can be greatly reduced, especially for aircrafts with a small payload
- UAVs are generally more expendable, which is an asset for military applications
- flight performance can be greatly increased, since there is no human pilot to be carried (allowing for longer endurance, tighter manoeuvres and higher payload).

Such advantages kept high the interest in developing military and commercial use of UAVs and, thanks to the huge advances in avionics and related technologies, UAVs rapidly became popular in the early 1990s. Especially the last decade has seen an enormous increase of interest in the development and application of UAVs. They are now very common in military operations, especially for reconnaissance and intelligence, but also for attack missions. It is

¹ PhD Student, ACSE Department, Mappin Street, S1 3JD, Sheffield, p.gunetti@shef.ac.uk, AIAA Member.

² Lecturer, ACSE Department, Mappin Street, S1 3JD, Sheffield, t.j.dodd@shef.ac.uk.

³ Professor, ACSE Department, Mappin Street, S1 3JD, Sheffield, h.thompson@shef.ac.uk.

clear that UAVs also have a great potential for civilian applications, such as surveillance and environmental protection [4]. However, these applications are being held back by safety concerns and a regulatory gap [5].

It is foreseeable that UAVs will present increasing levels of autonomy in the future, as many research studies focus on two trends: control of UAVs by personnel without extensive pilot training, and control of multiple UAVs by a single user. Safety concerns can only be worsened by an increasing level of autonomy, however since military applications usually allow for earlier implementation of new technologies, the biggest advances in UAV autonomy are being made in this area. From the autonomy point of view, current military UAVs are capable of carrying out autonomously an entire pre-planned mission [6], and generally only need supervision in order to address situational changes that ask for a modification to the flight plan.

Civilian applications instead usually involve low levels of autonomy, focusing most of the UAV functionality on the pilot [7]. However, it is clear that they would benefit even more from increased levels of autonomy, since it would allow decreased operating costs and would result in UAVs being routinely used in applications where they are not currently considered.

Safety issues have to be addressed before opening civil airspace to UAV traffic. The challenges presented by such an objective are great, especially in the case of a fully autonomous UAV [8]. In fact, an autonomous UAV must be able not only to fly a pre-planned mission, but also to actively adapt to situational changes (such as the detection of new obstacles or the occurrence of a fault), communicate with other entities and generally follow flight rules while interacting with a dynamic environment.

In general, UAV autonomy involves not only the ability of the UAV to control its path and perform its planned mission, but also the need to achieve sufficient external and internal situational awareness so that it can react properly to changes: on one hand, the UAV must be aware of what is happening around itself, on the other hand it must have knowledge about the operation of all of its subsystems. While there are many specific tasks that are usually handled by automatic control systems on a piloted aircraft (for example, an autopilot usually keeps the aircraft on its intended route during normal cruise), several other decisions are left to the pilot (for example, the course of action to take after the occurrence of a fault). A completely autonomous UAV must be able to make all types of decisions, including those that are normally delegated to the pilot, since minimal supervision is to be assumed. Furthermore, an autonomous UAV should also show a high degree of intelligence, in order to maximise its capabilities and effectiveness.

Several approaches have been tried to achieve autonomy and intelligence [9, 10]. Within the present study, the focus is placed on one of these approaches, the use of cognitive architectures in control applications. Cognitive architectures are basically models of the functionality of the human brain, and are pre-eminently used in psychological and neurological studies. However, they also possess the potential to be very useful in the control engineering and artificial intelligence fields, especially when complemented by other AI approaches (fuzzy logic, neural networks) and traditional control techniques. The use of a cognitive architecture can bring substantial improvement in terms of overall “system intelligence”, but is problematic due to the necessity of real-time operation and the possibility of non-deterministic behaviour (which is always considered negatively in aerospace applications).

The main purpose of this paper is to present a novel approach to Automatic Flight Plan Generation, based on the fusion of a cognitive architecture with traditional control techniques. Within the paper, the Soar architecture and Soar Intelligent Agents are introduced in section II. The proposed system architecture is outlined in section III, along with the abstractions needed to define it. The Planner agent is then described in section IV and then tested, with simulation results presented in section V. Results are preliminary, especially due to the lack of several flight plan optimization features that are expected to be added at a later stage, after the feasibility of this approach is demonstrated. The Execution agent is described and tested in section VI. Finally, the Conclusions section analyzes the lessons learnt during the study and sets the way for future research work.

II. Soar Intelligent Agents

The research work that is described in this paper began as a general idea of integrating Intelligent Agent (IA) technology with traditional control techniques in order to increase the intelligence and autonomy of software systems. Intelligent Agents were introduced in the early 1990s [11, 12] as a new software engineering paradigm. They represent a substantial innovation and have found many uses, especially in web-related applications. However, this trend did not extend yet to control applications, especially in the aviation market where safety is a major issue.

The first objective was then to verify the suitability of Intelligent Agents for aerospace control applications [13, 14], judging whether such a system could not only be feasible but also bring improved functionality.

The choice of available software packages is quite limited, as most IA applications are still custom developed. The main ones are JACK, which is basically a set of Java classes that implement IA theoretical concepts (the Belief-Desire-Intention paradigm, see [10]), and Soar and ACT-R, which are instead cognitive architectures. JACK was excluded due to its non-cognitive nature, and Soar was preferred over ACT-R due to a better I/O interface and C++ coding (instead of Lisp).

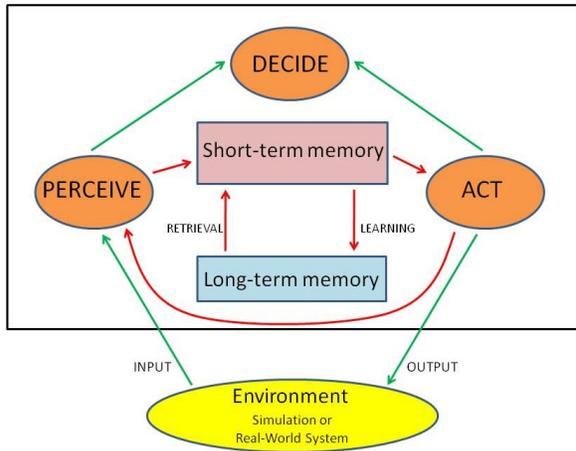


Figure 1. Perceive-Decide-Act cycle

experience. A Soar agent is based on its production rules; these represent long-term knowledge and are practically the program code for the agent. Production rules are in the form of if-then statements, where an action is performed only if the conditions are met. When the conditions of a production are met, the production is said to fire; as Soar treats all productions as being tested in parallel, several productions can fire at once, and this can happen at different levels of abstraction, giving the Soar agent natural pro-active behaviour (the agent is inherently aware whether the conditions to apply certain production rules are still valid). Short-term knowledge is instead constituted by external input, and appropriate functions must be developed to interface the Soar agent with its environment. Figure 2 presents a graphical description of the Soar architecture.

In practical terms, a Soar agent is a C++ class, and appropriate I/O functions have to be developed in order to interface it with its environment. Since the objective is to combine Soar agents with other control techniques, we chose to integrate them with Matlab/Simulink, which is the most commonly used software package in Control Systems Design. This allows seamless integration of the control algorithms, once the Soar\Simulink interface is set up, and also provides a simulation environment which is indispensable in testing the system.

The Soar+Simulink approach was initially applied in the development of a Health Management system for Gas-Turbine Engines [13, 14]. The work presented in this paper is partially based on the findings described in those early studies. In particular, the idea of implementing Soar agents as Simulink S-Functions derives from that experience, as well as the idea of interfacing multiple Soar agents in a Simulink environment. While the mentioned studies proved the feasibility of such a system, they did not bring improved functionality due to a limited problem space and severe constraints. It is expected that Autonomous UAV Mission Management, especially when applied to a dynamic environment, will instead bring out the advantages of integrating a cognitive architecture with traditional control techniques in order to achieve improved autonomy and intelligence in a control system.

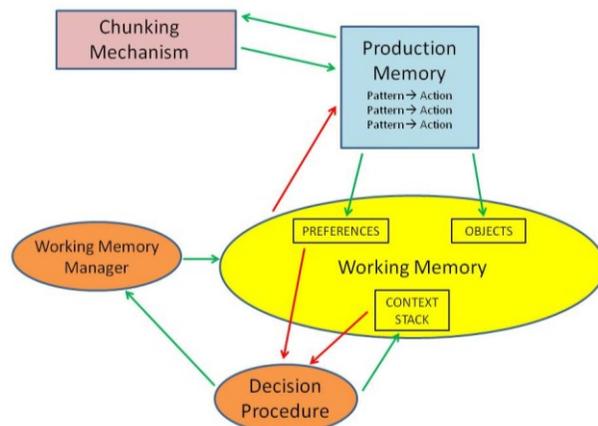


Figure 2. Soar architecture concepts

III. Autonomous UAV architecture

Modularization is one of the key concepts of Intelligent Agent theory: rather than having a single all-encompassing agent, it is usually preferred to have multiple ones that are dedicated to specific functions. A modular approach is also generally well suited to control applications, as many of the most common techniques benefit from the subdivision of complex problems into several simpler ones.

A sound way to achieve modularity involves beginning development with a systems engineering study of the problem. Especially in a case such as this, where the intention is to fuse very different technologies (a cognitive architecture and traditional control techniques implemented in Simulink), it is necessary to think a priori about how functionality can be modularized and what functions will be performed by what components. It is also very important to define the interfaces between the various components, so that their integration can be achieved seamlessly.

The first step in defining the Autonomous Mission Management architecture is outlining the general capabilities that it should possess:

- very simple user interface – an untrained end-user is expected, that should only need to define the mission as very high level “Objectives”; the UAV then can carry it out without supervision; the User monitors mission execution only if so desired (to access real-time sensor data or input a new Objective, for example)
- automatic mission plan optimization – the UAV should be able to perceive and evaluate the factors that can influence the mission (weather, threats, fuel consumption, damage to the airframe, etc.) and plan accordingly (for example, by detouring around a known bad weather area or hostile site)
- in-flight replanning – it is crucial that the UAV is able to automatically change its flight plan in order to react to new situational awareness during the mission; however, this should only happen when the situation really demands it (i.e. the trade-off between pro-active and reactive behaviours must be considered)
- real-time oriented – the system must be able to work under real-time constraints, which are especially high for a UAV
- automatic take-off and landing – the UAV should be able to perform these operations without supervision (apart from Air Traffic Control in the case of civilian airspace)

Some limitations must also be defined:

- the focus for this project is a single UAV, rather than multiple coordinated UAVs
- instead of looking for optimal solutions, we aim at extracting “good” ones that are derived from a series of abstractions (this is because algorithms for optimal solutions usually do not meet real-time requirements)
- while the Soar architecture has a dedicated learning mechanism, this will not be exploited, at least in this stage

Before being able to picture a system-level view of the proposed architecture, it is necessary to introduce some definitions regarding concepts and abstractions that are needed in defining the high-level tasks of the UAV and its control agents:

- Objective: a very high level task for the UAV, that defines part of a Mission, and usually consists of several Actions
- Action: a high level task for the UAV, representing the finest subdivision which is relevant to Mission Management
- Flight Plan: a numbered sequence of Actions, that completely describes a Mission
- Entity: a known presence that can potentially influence the generation of the Flight Plan (targets, bad weather areas, threats, ATC zones, geographical features, etc.)

In particular, the concepts of Objective and Action are central to the definition of the system. The Objective is the main User input abstraction. It represents a very high-level task for the UAV, defining a part of a mission. Five

Table 1. Objective properties

1	Objective Type	Analyze, attack, orbit, search or transit
2	ID Tag	A code that identifies an Objective
3	Objective Position	Coordinates for the specific Objective
4	Priority	Time and execution priority
5	Duty	Task being accomplished during Orbit
6	Area Type	Defines the type of Search pattern
7	Search Accuracy	Accuracy for a Search mission
8	Box Corner	Defines a Box-type Search area
9	Radius	Defines a Circle-type Search area
10	Target Tag	Identifies a specific target for analyze\attack
11	Orbit Time	Defines the time for an Orbit mission

types of Objectives have been defined at present: analyze target (go to a position to gather data on a specific target using payload sensors), attack target (deliver a weapon payload on a specific target), orbit position (circle about a position for a specified time, for example to act as communications relay), search area (patrol an area using standard patterns in order to identify targets) and transit (travel to a destination airport and land there). The Objective I/O object has a total of thirteen variables that can fully define any type of

Objective previously described. Table 1 defines all of these variables (note that some properties are described by multiple variables). In particular, it is important to define Priorities, which are very important during Flight Plan generation. Two types of priority are defined: time priority which is basically a time limit for the Objective to be accomplished, and execution priority, which indicates how important the Objective is in terms of mission success (critical, important, minor).

Table 2. Action types

1	Park	Wait until Mission Start time
2	Taxi	Move to runway position
3	Take-off	Perform take-off manoeuvre
4	Climb	Climb to specified altitude
5	MMS	Main Mission Start
6	Travel	Travel to position
7	Recon	Perform Reconnaissance on target
8	Attack	Perform Attack on target
9	Circle	Circle about specified position
10	MME	Main Mission End
11	Descent	Enter descent path
12	Landing	Perform landing manoeuvre

The Action is the defining element of a Flight Plan. It is still a high-level task, but is the finest subdivision which is relevant from a Mission Management point of view. In general, an Objective will always correspond to two or more Actions. Twelve types of Action have been identified as necessary to describe a full Flight Plan,

Table 2. The Action I/O object consists of thirteen variables that together fully describe it (Table 3). Each Action is assigned a sequence number that orders it within the Flight Plan.

An accessory abstraction is the concept of Entity. This represents any external factor that may influence the Flight Plan. Entities include targets of various types (buildings, vehicles), but also known threats (hostile presences, bad weather areas, etc.) and constraints (geography, Air Traffic Control zones).

Table 3. Action properties

1	Action Type	One from table 2
2	Sequence	Sequence number for the Action
3	Start Position	Initial position for certain Action Types
4	Position	Coordinates relevant to Action
5	Time	Time properties of Action
6	Heading	Bearing to be kept for certain Action Types
7	Altitude	UAV Altitude specified for Action
8	Duty	Duty type for Circle Actions
9	Speed	UAV Speed for Action
10	Target	Defines a specific target for Recon and Attack
11	Objective	Parent Objective ID tag and type

The Entity I/O object consists of eleven variables that describe its nature, its position and its behaviour among other things. While Objectives are a User Input, Entities are expected to be received automatically from an information gathering system (in military terms, the "Battlefield Network").

Figure 3 schematically describes the software architecture for Autonomous UAV Mission Management that we propose. Its main components are three Soar Intelligent Agents that perform very different functions: the Planner Agent, the Execution Agent and the Mission Manager Agent. The Soar agents are complemented by a User interface, real world sensory input (including on-board sensors and external data), and a set of Simulink functions that perform lower level tasks, such as Autopilot and Payload Management algorithms.

The Planner agent is tasked with receiving Objectives as input from a User and then fusing them with real world information (mainly target and threat positions) in order to obtain a full Flight Plan (as previously defined, a sequence of Actions). Generation of a new plan is triggered by an external component, the New Plan Trigger, to better balance between pro-active and reactive behaviour.

The Execution agent takes as input the Flight Plan and then executes it Action by Action. It basically acts as a transition layer between the Planner and low-level controls. As the mission is executed, it chooses what Action is to be performed and then, fusing the information contained within the Action with real-time sensor data (GPS, attitude, airspeed, etc.) sends commands to the lower level control systems, namely the Autopilot, the Direct Controls and the Payload Management System.

The Mission Manager agent is tasked with dealing with contingencies in the Flight Plan. It is very important for the intelligence of the overall system, since it has the power to change the Objectives inputted by the user (for example, by cancelling a secondary Objective that is close to a newly detected threat, or changing the parameters of a Search mission if a minor fault places stricter endurance limits on the UAV) or to add new ones (taking advantage of targets

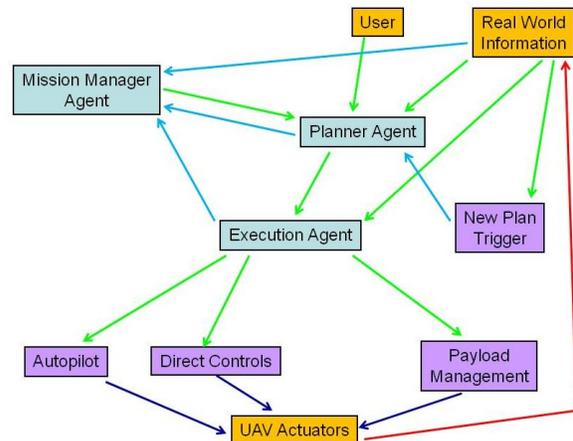


Figure 3. Architecture overview

of opportunity, for example). This kind of autonomy is needed in order to really make intelligent decisions, however finding the balance between excessive autonomy and truly intelligent behaviour is a difficult task.

The agents are complemented by a set of functions that use more traditional control techniques:

- the New Plan Trigger function, which monitors all inputs to the system and compares them with the situation recorded when the last Flight Plan was generated, in order to trigger the generation of a new Flight Plan only when truly needed
- the Autopilot function, which consists of a standard autopilot that controls the flight path of the UAV during the various mission phases, receiving its input (origin and destination coordinates, desired speed and altitude) from the Execution agent
- the Direct Controls function, which takes direct attitude and speed input from the Execution agent (at present, these are used only for the Taxi, Take-Off and Landing parts of the mission)
- the Payload Management function, which translates generic payload commands from the Execution agent to actual controls for the payload actuators

Completing the loop is the simulation environment, which is modelled in Simulink and receives input from the low level functions and provides feedback as real world sensor information.

IV. The Planner Agent

At the current stage of the project, while the overall architecture has been outlined, only some of the components have been implemented. The following sections (section IV, V and VI) will describe those components (the Planner agent and the Execution agent) and present testing results.

Figure 4 shows the current Simulink implementation of the system. The configuration includes a set of user input scenarios (white top left block), an instantiation of the Planner agent (coloured top centre block), the Execution agent itself (coloured bottom right block), a UAV model (coloured bottom centre block) and visualization routines (other white blocks).

Using the abstractions defined in section III, the task of an automatic Flight Plan generator is to convert a list of Objectives into a numbered sequence of Actions, which constitutes a full description of the Flight Plan, while taking account of all known Entities. This functionality is fully implemented by the Planner Agent.

The Perceive-Decide-Act cycle for the Planner will now be described. Input to the Planner agent includes base

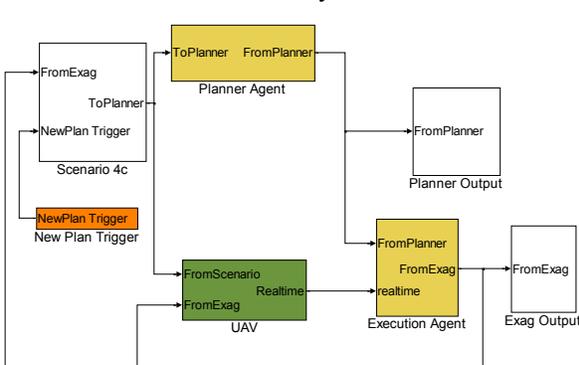


Figure 4. Simulation configuration

airport information (runway position and heading, altitude at ground level, and in future cognitive representation of taxiing paths), a list of Objectives from the User, a list of Entities which should be automatically updated by a dedicated data-link, and feedback from the Execution agent (basically indicating what stage of the Flight Plan has been reached). On the first iteration after the list of Objectives is available, the Planner generates a first Flight-Plan, which is then sent forward (to the Execution agent) and stored internally for reference. When sufficient changes have occurred to situational awareness, an external function signals the Planner that a new plan is needed. The Planner then cancels the current Flight Plan (but keeps an internal record of it) and

completely generates a new one, taking into account what parts of the old Flight Plan have already been executed. Just as with the first one, the new plan is then sent forward and stored internally. The same cycle is repeated until the mission is over.

From a Soar implementation point of view, the Planner agent can be described by ten states and sub-states, as can be seen in figure 5. From the main state, two sub-states can be reached: generate-plan, which is valid only when no Flight Plan is currently selected and causes the generation of the entire Flight Plan during a single iteration, and modify-plan, which stores old plans and watches for input from the New Plan Trigger function. Generate-plan is then split into four sub-states:

- old-plan, which copies parts of the old Flight Plan that have already been executed into the new one being generated
- take-off, which adds to the Flight Plan all Actions related to take-off operations (Park, Taxi, Take-off and Climb)
- main-mission, which develops the main part of the Flight Plan (when all Objectives are accomplished)

- approach, which adds all Actions related to landing operations (Descent, Landing, Taxi, Park)
- Finally, the main-mission state has three sub-states:
- plan-sequencing; during this phase, Objectives are ordered in a sequence, considering Priority levels and minimising the distance to be covered (a modified nearest-neighbour algorithm is used)
 - actions-definition; during this phase, every Objective is converted into the corresponding list of Actions, which are ordered reflecting the sequence of Objectives defined during plan-sequencing
 - plan-optimization; during this phase, the Flight Plan is optimized with respect to influences by the various Entities currently detected (for example, if a Travel Action intersects with a perceived threat, a detour around the threat area is added to the plan)

The plan-sequencing phase uses a modified version of the Nearest-Neighbour algorithm to solve what is basically a classical Travelling Salesman Problem. However, modifications to the algorithm are required in order to take account of the added constraints that are added by the different time priorities that can be assigned to Objectives. Regarding the plan-optimization part, examples of how a Flight Plan can be optimized include: adding a detour around a threat area if a Travel action intersects it, optimization of speed and altitude in light of situational awareness, rearranging the Plan in order to deal with airframe faults that arose during flight

The Planner agent generates the Flight Plan in an iterative manner; as it goes from sub-state to sub-state, Actions are added to the plan, until the final Park Action has been placed. When that is detected, the plan is sent to the Execution agent (and other listening agents) and the modify-plan state is entered.

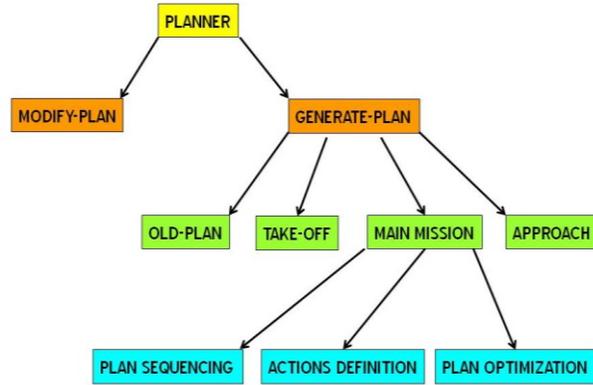


Figure 5. Planner agent states

It is essential to understand the importance of the “plan-optimization” part of the Planner. Although it is not strictly necessary, it is in this phase that true intelligence can be demonstrated. While the rest of the plan generation process is quite straight-forward, during this phase a complex decision-making algorithm can be implemented. The plan-optimization part of the Planner is separated from the rest so that it can be as simple or as complex as desired. Obviously, the more complex it is, the higher will be the level of intelligence of the system. In other words, while the rest of the Planner implements autonomy, it is the plan-optimization part that implements intelligence.

At the current stage of the project, plan-optimization has not yet been implemented. In fact, the results presented in section V are preliminary because of the absence of advanced intelligent behaviour. While autonomous Flight Plan generation is demonstrated, during testing it was clear that in certain situations the Planner opted for a Flight Plan which was not optimal. Expanding the optimization part of the Planner will represent a considerable part of the future work for this project.

V. Planner Simulation Results

The Planner agent is built to be able to cope with any type of mission which can be described using the

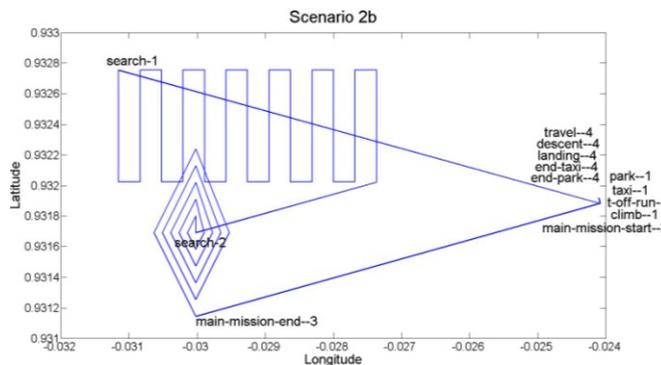


Figure 6. Flight Plan, Scenario 2b

abstractions and concepts that have been introduced in section III. It is obvious that, due to the high number of input variables connected to the generation of a Flight Plan and considering that replanning can occur at any moment, testing every conceivable situation is practically impossible. Consequentially, a set of realistic and challenging test scenarios was needed in order to verify the functionality of the Planner.

At present, a total of six test scenarios have been developed. All of these are built so as to exploit the replanning capabilities of the Planner. Their design presents high variety so that the Planner can be faced with as many different

situations as possible. Input scenarios are constituted by base airport information, the list of current Objectives and the list of current Entities.

Results of the simulations are presented as longitude/latitude plots, with the values in radians. Altitude plot is excluded at present, since it not really relevant at this stage (it becomes relevant when the Execution agent is involved).

In figure 8 (see section VI), it is possible to see the plot of the Flight Plan for scenario 3c. This scenario involves a complex mission including five Objectives with different priorities. A replanning event occurs when priorities are changed. A further Transit Objective is also present, causing the UAV to land at a different airport than the starting one. The plot evidences how the Planner puts Objectives in a sequence that is highly dependent on time priorities, but also tries to minimize the distance to be covered.

In figure 6, the plot for scenario 2b is showed. This scenario is aimed at demonstrating the different Search patterns that have been implemented in the Planner. Two types of search areas have been defined: a rectangle, to be searched using the classical parallel track search pattern, or a circular area, to be searched using an expanding diamond spiral pattern. It is to be noted that the plot is not to scale, for this reason the circle search pattern appears deformed. In both cases, the accuracy of the search is inputted by the User as part of the Search Objective definition. In this scenario, the UAV takes-off and lands at the same airport.

Figure 7 shows all plans generated within scenario 5e. This is a highly dynamic scenario where new flight plan generation is needed three times. The mission starts with four Objectives (1-orbit, 2-recon, 3-attack, 4-search); only Objective 1 has a higher priority. After the take-off sequence, a new Objective with “immediate” priority is entered while the UAV is flying towards Objective 1. Although it is not clear from the graph, the Planner then aborts the Travel Action to Objective 1 and immediately goes to the new Objective, because of the higher priority. The mission then goes on as previously planned. While performing Objective 4 (the search), a new change occurs: Objective 3 is cancelled, so a new plan is generated which excludes it. Finally, while reaching Objective 2, a new Transit Objective is added, telling the UAV to land at a different airport than the starting one. This scenario demonstrates the ability of the Planner to cope with a dynamic environment, since it is clear that a new plan can simply be generated when the need arises (which might not be just in the case of a change in the Objectives, but also when new threats are perceived or in general when the situational awareness changes so that the New Plan Trigger function decides that a new plan is needed).

It is clear from these results that the basic Planner functionality has been achieved. The agent is capable of converting high level Objectives into a list of detailed Actions, that can then be executed by another agent (the Execution agent). However, results from these simulations also evidenced a lack of intelligence, especially in certain situations, where the lack of plan optimization was clear.

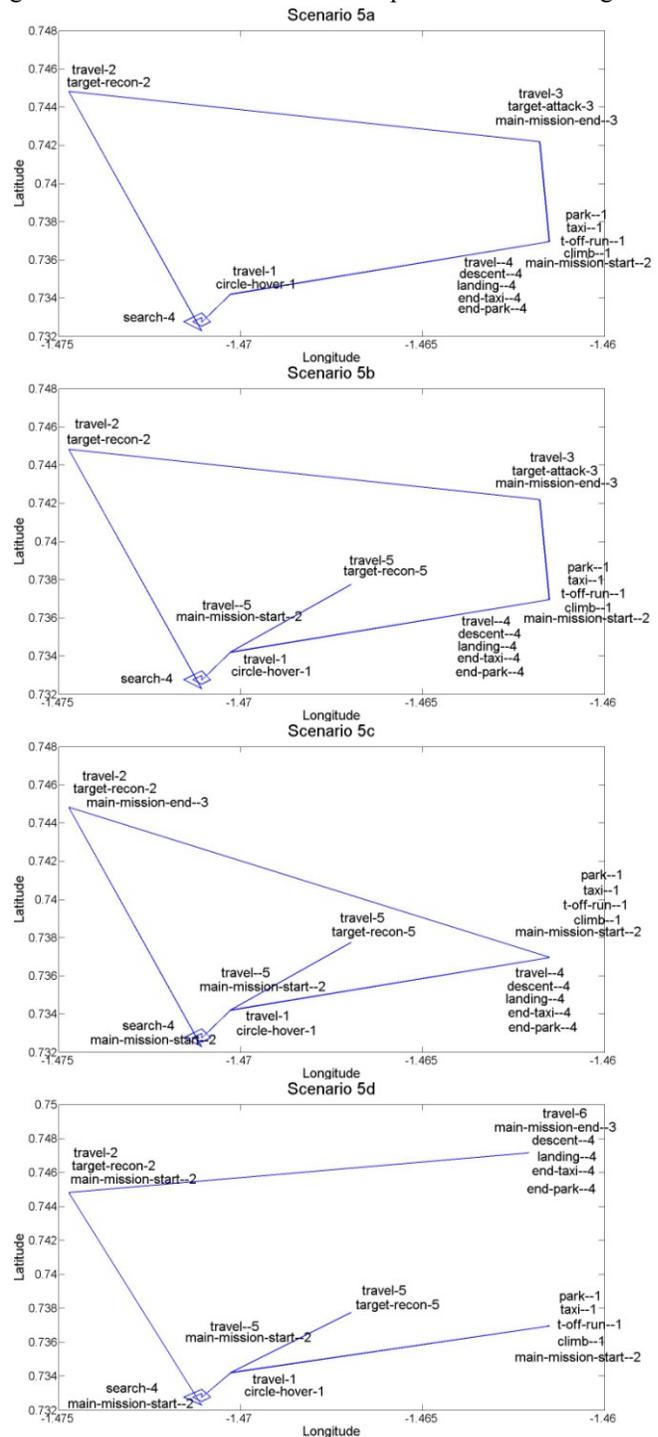


Figure 7. Flight Plan, Evolution of Scenario 5

In general, it is then possible to say that while simulation results have confirmed the validity of this approach, consistent additional work should be done in order to really demonstrate its value.

VI. Execution Agent Description and Testing

In the remaining part of this paper, we will focus on the Execution agent. It is however important to remind that testing the Execution agent involves having real-time input from the Planner, which is therefore always executed together with it.

The Execution agent has basically to act as a transition layer between the abstract concept of Action and the actual controls of a UAV. The main input for the Execution Agent (or Exag) is the Flight Plan that is generated by the Planner agent. This is basically a numbered sequence of Actions, and the Exag is tasked with executing all of these in the determined sequence.

Conceptually, the Exag is very simple: from a Soar point of view, all it does is starting from Action number One, executing related commands, verifying the execution of the Action and then going to Action number Two, to repeat the cycle again. However, this is made more complex by the fact that every type of Action needs to be dealt with in a different way. Basically, each of the Action types outlined in Table 2 has a dedicated Execution algorithm.

The output of the Exag can be divided in three main sections: Planner Feedback, Current Action Details and Commands. Planner Feedback includes data such as the number of the current Action being performed and the Commitment to an Objective, which is needed by the Planner in the case of replanning. Current Action Details basically extracts from the Flight Plan all details about the specific Action being executed. Finally, Commands represent direct input to the UAV low-level controls, and include Direct controls (speed, pitch, roll, yaw, brakes), Autopilot controls (speed, altitude, initial and final position) and Payload commands.

The Exag selects the current Action to be performed, then calculates what commands need to be given in the light of Action details and real-time sensor information. Every type of Action is translated into a different list of commands. We will now describe how each type of Action is accomplished:

- *parking*; this action is very simple, only requiring to keep the UAV still on ground until the Mission Start Time is reached. Most commands are set to zero value, apart from the brakes
- *taxi*; this action is one of the most complex, as it involves ground navigation (with all its constraints). At present, it is executed by directly steering the UAV towards the planned take-off position (commands: yaw and low speed), then moving at a higher speed and finally stopping the UAV when the position is reached. It is planned to improve the taxi algorithm with navigation within runways and communication with ATC control
- *take-off*; once the expected take-off position is

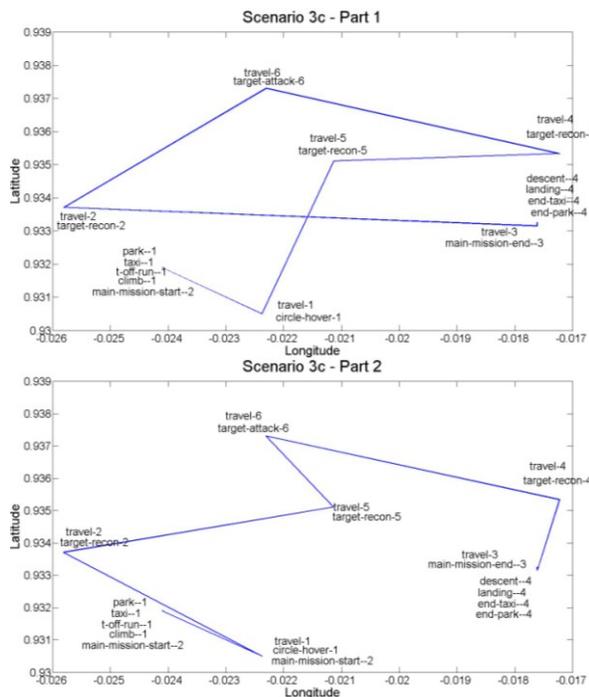


Figure 8. Flight Plans, Scenario 3c

reached, the UAV is steered in the runway direction (commands: yaw and low speed), then full throttle is set (keep yaw, maximum speed) until the take-off speed is reached. At this point a pitch command is given, and then Take-off is considered finished when the UAV has cleared the 15m level from the ground

- *climb*; immediately after take-off, the Climb action keeps the UAV in a specific direction and sets a fixed climb rate that allows it to reach a desired altitude. When this altitude is reached, a level flight condition is entered and then the main mission begins to be executed
- *main-mission-start*; this is not a proper Action, but still needs to be dealt with by the Exag, since it is part of a Flight Plan
- *travel*; this is the most important type of Action, and the first type to make full use of the Autopilot functions. It basically sets a great-circle route (shortest distance on a sphere) between the current position and the intended destination, at a specified speed and altitude. The distance to the destination position is continuously verified in

order to make decisions regarding the Commitment to the Objective

- *target-recon*; this Action involves a pass over a target in order to allow a sensory payload to gather data. After the target position is reached, a turn-around approach waypoint is set, the UAV then travels towards it before steering back towards the target for the actual data gathering pass
- *target-attack*; this Action is very similar to target-recon, but can use different parameters in determining the type of approach and of course uses a different type of payload
- *circle-hover*; in this Action, four waypoints forming a diamond are calculated around the central position. The UAV then cycles through those in clock-wise (or anti-clockwise) direction, until the specified time limit
- *main-mission-end*; as with main-mission-start
- *descent*; in this Action, after the expected landing position has been reached in flight, two waypoints are calculated and reached using the autopilot. These waypoints basically draw an ideal descent path that is in line with the runway
- *landing*; this Action makes use of the Direct controls rather than the Autopilot, and has the UAV descend at a specific angle, then perform a flare manoeuvre when close to the ground and finally stop when ground contact has been ensured

The Execution Agent has been tested in a simulation environment. The simulation configuration is described in *Figure 4*. The scenarios used are the same described in *section V*.

The inputs to the Execution agent are the Flight Plan from the Planner and a set of real-time world information, provided by the UAV model. It is important to note that while the Planner agent effectively operates in a static manner, without the need for synchronization with the real system, the Exag has to dynamically follow the evolution of the mission, operating under strict real-time constraints. The Exag outputs the commands that are then fed into the UAV model (which includes an Autopilot), which calculates the response of the physical system.

Figure 8 is the graphical description of the Flight Plans generated during Scenario 3c. This scenario involves a mission with six Objectives with different Time Priorities During the execution of the mission (after the execution of Objective 1), some of the priorities are changed, causing a replanning event. The UAV then diverts from its current Objective and executes the Objectives in a different order, so that the distance covered is optimized while respecting the new priority settings.

Figure 9 plots the actual position of the UAV during the evolution of the mission. The trajectory is captured at different time points in order to show the dynamic nature of the Execution agent (the Flight Plans from *figure 8* are generated instantaneously). It is possible to note the diamond trajectory that the UAV keeps during an Orbit Objective, the turn-around waypoint navigation during Recon and Attack Objectives and the waypoints used during Descent.

The first “shot” is taken just after the UAV has begun circling for Objective 1, the second is taken just prior to Objective 5, the third shows the entire mission. It is possible to note the point when replanning occurs, with the UAV that is going from Objective 1 to Objective 5 but then diverts toward Objective 2.

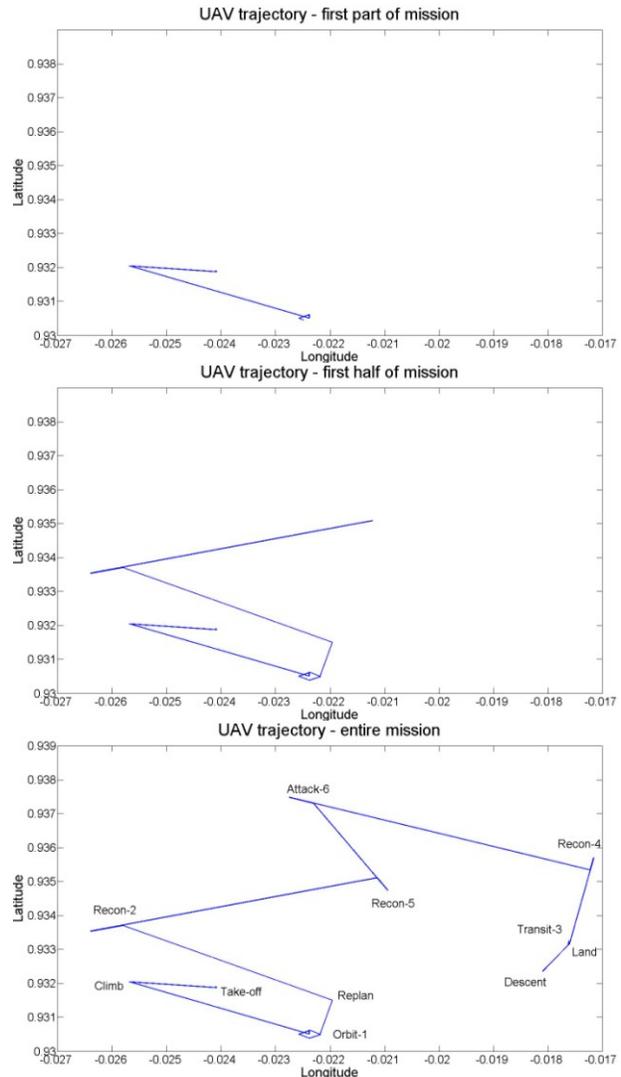


Figure 9. UAV trajectory, evolution of Scenario 3c

VII. Conclusion

This paper presented a novel approach to Autonomous UAV Mission Management and Control. A software architecture, based on the integration of Soar Intelligent Agents and traditional control techniques, was introduced, together with the concepts and abstractions which are necessary to its definition. A thorough description of implemented components at the current stage of the study was presented, together with simulation results that demonstrated the feasibility of this approach.

Future work will involve the implementation of the third agent (the Mission Manager agent), together with support functions and algorithms allowing for direct control of a UAV model (e.g. an Autopilot translating agent output to a properly formatted command to the UAV). Furthermore, the Planner agent will be improved by adding Optimization functions, such as the ability to avoid dangerous areas and the ability to manage fuel consumption. This should constitute an important step in showing the validity of this approach and the advantages that it can bring.

References

- [1] G. Goebel, "History of Unmanned Aerial Vehicles", <http://www.vectorsite.net/twuav.html>, 2008
- [2] Y. Tan, X. Zhu, R. Zhao, B. Zhang, "The Design and Implementation of Autonomous Mission Manager for Small UAVs", International Conference on Control and Automation, 30 May – 1 Jun 2007, Guangzhou, China
- [3] P. Schaefer et al., "Reliable Autonomous Control Technologies (ReACT) for Uninhabited Aerial Vehicles", 2001 IEEE Aerospace Conference, Big Sky, MT
- [4] V. Ambrosia, S. Wegener, J. Brass, S. Schoenung, "The UAV Western States Fire Mission: Concepts, Plans and Developmental Advancements", AIAA 3rd "Unmanned Unlimited" Technical Conference, Workshop and Exhibit, 20 - 23 September 2004, Chicago, Illinois, AIAA 2004-6415
- [5] M. DeGarmo and G. Nelson, "Prospective Unmanned Aerial Vehicle Operations in the Future National Airspace System", 4th Aviation Technology, Integration and Operations Conference, ATIO 2004
- [6] J. Miller et al., "Intelligent Unmanned Air Vehicle Flight Systems", American Institute of Aeronautics and Astronautics, InfoTech@Aerospace Conference 2005, Paper No. 2005-7081
- [7] UAV Task Force, "The Joint JAA/EUROCONTROL Initiative on UAVs", UAV Task Force Final Report, 2004.
- [8] V. Crum, D. Homan, R. Bortner, "Certification Challenges for Autonomous Flight Control Systems", AIAA Guidance, Navigation, and Control Conference and Exhibit, 16 - 19 August 2004, Providence, Rhode Island, AIAA 2004-5257
- [9] L. Long, S. Hanford, O. Janrathitikarn, G. Sinsley, J. Miller, "A Review of Intelligent Systems Software for Autonomous Vehicles", Proceedings of the 2007 IEEE Symposium on Computational Intelligence in Security and Defense Applications (CISDA 2007)
- [10] A. Lucas, C. Heinze, S. Karim et al., "Development and Flight Testing of an Intelligent, Autonomous UAV Capability", AIAA Unmanned Unlimited 2004, 20-23 September 2004, Chicago, IL, AIAA2004-6574
- [11] N. Jennings, M. Wooldridge, "Applications of Intelligent Agents", in "Agent Technology: Foundation, Applications and Markets", Springer, 1998
- [12] M. Wooldridge, "Intelligent Agents", in "Multi-Agent Systems: a modern approach to distributed artificial intelligence", the MIT Press, 1999
- [13] P. Gunetti, A. Mills, H. Thompson, "A distributed Intelligent Agent architecture for Gas-Turbine Engine Health Management", 46th AIAA Aerospace Sciences Meeting and Exhibit, 7 – 10 January 2008, Reno, NV
- [14] P. Gunetti, H. Thompson, "A Soar-based Planning Agent for Gas-Turbine Engine Control and Health Management", 17th IFAC World Congress, Seoul, Korea, July 2008
- [15] Soar Technology Inc, "Soar – An overview", © 2002
- [16] J. Laird, A. Newell, and P. Rosenbloom, "Soar: An Architecture for General Intelligence", Artificial Intelligence, 1987, 33(3), pp. 1-64