# Simulation of a Soar-Based Autonomous Mission Management System for Unmanned Aircraft

Paolo Gunetti,* Haydn Thompson,† and Tony Dodd‡
*University of Sheffield, Sheffield, England S1 3JD, United Kingdom*

DOI: 10.2514/1.53282

**State-of-the-art unmanned aerial vehicles are typically able to autonomously execute a preplanned mission. However, unmanned aerial vehicles usually fly in a very dynamic environment that requires dynamic changes to the flight plan; this mission management activity is usually tasked to human supervision. Through the use of a set of theoretical concepts that allow the description of a flight plan, a software system that autonomously accomplishes the mission management task for an unmanned aerial vehicle was developed. The system was implemented using a combination of Soar intelligent agents and traditional control techniques, and it is thoroughly described in the first part of the paper. An extensive testing campaign, based on the use of a realistic simulation environment, was performed on the system; the second part of this paper presents results obtained during this campaign. The system was demonstrated to be capable of automatically generating and executing an entire flight plan after being assigned a set of objectives. In conclusion, possible future developments are discussed, focusing in particular on prospective hardware implementation for the system.**

## I. Introduction

Q1 FOR certain application fields, the many advantages in using unmanned aerial vehicles (UAVs) over conventional aircraft are clear and have been thoroughly discussed [1]. These advantages include reduced operating costs, better flight performance, and expendability. The latter advantage in particular resulted in a strong interest in UAVs for military applications, especially during the last two decades; civilian applications have instead been developing with a much slower pace, in part due to a persisting regulatory gap. As UAV technology grows more mature, civilian applications are expected to become more common [2].

The most basic form of UAV is essentially a remotely piloted airplane, but this control scheme is not very suitable to applications that demand out-of-sight operations. For this reason, some level of autonomy has always been preferable, and in fact there has been a constant trend toward increased autonomy. The current generation of UAVs is generally capable of carrying out a preplanned mission on its own [3] but relies on human supervision in order to address unforeseen events that require changes to the flight plan. This level of autonomy is generally not sufficient to ensure safety and is one of the main reasons for the relatively low number of civilian applications: until it is possible to ensure safety during incidents, such as loss of communication, loss of control, or loss of situational awareness, civilian applications will always be constrained to particular conditions (flight within restricted airspace or with the need for special permits). Eventually, it is expected that UAVs will be able to seamlessly merge with piloted aircraft within civilian airspace. This, however, at the moment, is only a distant vision, and a large amount of research work on UAV autonomy is still needed in order to achieve the required capabilities.

To evaluate the autonomy level of a UAV, Clough introduced a classification of UAV autonomy levels in a 2002 paper [4]; this has become the de facto standard for evaluation of UAV autonomy. The Clough classification includes 11 levels, ranging from level 0 (radio-controlled drone) to level 10 (fully autonomous); each level adds a set of functionality over the other levels. For example, level 1 adds the capability to autonomously execute a preplanned mission, while level 2 adds the capability to autonomously switch between a set of pregenerated flight plans according to current situational awareness. Levels 3 to 5 add increasingly complex capabilities that do not involve multiple UAVs, such as fault mitigation and contingency management (sense-and-avoid); levels 6 to 10 focus on coordination and cooperation between multiple UAVs.

The current generation of UAVs is rated around level 2 of the Clough classification [4], and significant efforts are being spent in order to research the technology needed to achieve higher levels. However, it can be noted that most research work is actually focused on the capabilities required by the higher autonomy levels and related to multi-UAV coordination (for example, see [5,6]). A comprehensive review of autonomy-related research efforts can be found in [7]; the authors note that, at present, there is no unified attempt to develop standards in this field, thus resulting in noncoordinated research efforts and in difficulties regarding certification of such technology. In the paper, it is pointed out that the possibility of certification is critical for the development of safety-critical systems such as UAVs, and intelligent agent (IA) technology is proposed as a possible approach that could allow the definition of a clearer path toward certification.

IAs based on the Soar architecture were used to control simulated aircraft within the TacAir-Soar project [8]; TacAir-Soar is built to simulate human behavior, modeling the tactical decision-making of a fighter pilot within a battlefield scenario. TacAir-Soar demonstrated the feasibility of real-time high-level control of aircraft using Soar agents; however, this is a system that is intended to simulate human behavior rather than to achieve actual control of a vehicle. Another attempt to use cognitive software to achieve UAV autonomy is presented by Karim et al. [9], Lucas et al. [10], and Karim and Heinze in several papers [11]: in this study, a cognitive system is used to provide the reasoning capability needed by a mission management system. This system is implemented using the JACK© IA language and has demonstrated a limited set of autonomy-related capabilities during actual flight tests.

In this paper, a similar system will be proposed, in which a cognitive architecture is used to provide the reasoning capability needed for mission management. The Soar architecture was chosen for the development of this system, for the reasons explained in Sec. II; Soar is the computational

Received 29 November 2010; revision received 4 January 2012; accepted for publication 29 February 2012. 

*Ph.D. Student, Automatic Control and Systems Engineering Department; paolo.gunetti@polito.it. Member AIAA.
†Professor, Automatic Control and Systems Engineering Department; h.thompson@shef.ac.uk.
‡Senior Lecturer, Automatic Control and Systems Engineering Department; t.j.dodd@shef.ac.uk.

implementation of a unified theory of cognition originally developed at Carnegie Mellon University. The system presented in this paper is designed to achieve an autonomy level residing between levels 4 and 5 of the Clough classification [4] (higher than the Karim and Heinze study [11]). It is therefore desired that the system should require minimal supervision (limited to the assignment of mission objectives) and the ability to generate "intelligent" flight plans that take account of situational awareness and can be updated during flight. Also, a key feature of the system is that it should be able to satisfy real-time requirements when executed on the typical hardware that can be expected on a small low-cost UAV; while this aspect has not been implemented yet, the goal is to have a system that can run on a single common-off-the-shelf (COTS) PC/104 board.

The system was named SAMMS, or Soar-based autonomous mission management system. Soar agents constitute the reasoning core of the system, but execution of flight plans generated by the mission management algorithms requires integration with traditional control techniques (e.g., autopilots). SAMMS is designed as a low-cost architecture for the development of the control software of a highly autonomous UAV, which should be able to entirely perform a mission once given a set of objectives by a user. SAMMS generates a flight plan that takes into account UAV status and environment information, and then it executes the generated flight plan and dynamically updates it as new information is available. SAMMS is at an early development stage, and certain aspects of UAV design are not fully addressed; this is true in particular for sensors and payload management. Rather than focusing on the full specification of such aspects, the architecture presents a methodology to take this information into account, considering it as provided by an external system. On an actual system, an intermediate layer would be needed between external systems and SAMMS; an example of this is already implemented, since the execution agent is basically acting as an intermediate layer between the high-level planning agents and the autopilot functions (as will be seen within the paper). Q2

Through simulation tests, this paper aims at demonstrating that the SAMMS architecture is viable and possesses highly desirable characteristics: autonomy, replanning ability, real-time operation, and potential for certification. The theoretical background for SAMMS is also presented, outlining a set of constructs that allows the computational description of a UAV mission. The paper is divided into four sections. In Sec. II, the theoretical base for SAMMS will be presented; the main abstractions used during development will be outlined and the system architecture described, particularly focusing on the three main software components (Soar agents). In Sec. III, the architecture will be tested using a realistic simulation environment; the environment is described, and results are presented. In Sec. IV, possible real-world implementation of the system is discussed. Finally, Sec. V will draw conclusions regarding the work and propose possible future work.

## II. Overview of Soar and Soar-Based Autonomous Mission Management System

In the system presented in this paper (SAMMS), reasoning and planning capabilities are provided by Soar IAs. IAs represent a new paradigm for software engineering and were first introduced in the early 1990s to improve the flexibility of software systems [12,13]. IA-based systems are used in very diverse applications, ranging from internet search engines to air traffic control and cognitive studies. Several architectures for IA-based systems have been proposed, including ACT-R, Soar, and JACK. In [14], several architectures are discussed. The Soar architecture was chosen for the development of SAMMS for several reasons:

1) It is a cognitive modeling tool that provides high potential in developing intelligent capabilities through the use of symbolic IA techniques.

2) It is proven to be capable to deal with very complex problem spaces while maintaining real-time operation.

3) It provides a good I/O interface, both between separate agents and with external components.

4) Its core is written in the C++ language (on the contrary, the aforementioned JACK package is based on Java), thus providing an easier path toward certification.

5) It is a fully open-source project.

Soar is the computational implementation of a cognitive architecture that was originally developed at Carnegie Mellon University since the late 1980s and is now maintained by the University of Michigan [15,16]. It provides a robust architecture for building complex human behavior models and intelligent systems that use large amounts of knowledge. A detailed explanation of the Soar architecture is outside the scope of this paper, but a short introduction to it is in order. As already stated, Soar can be used to model human behavior and to build intelligent systems; the SAMMS project is focused on the latter option. At the core of a Soar agent lies a perceive–decide–act cycle during which the agent samples the current state of the world, makes knowledge-rich decisions in the service of explicit goals, and performs goal-directed actions to change the world in intelligent ways. Long-term memory of an agent is represented by production rules, which come 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. Operators are constructs formed by one proposal rule and one application rule, and they possess the ability to modify short-term memory. Short-term knowledge is instead constituted by external input, and appropriate functions must be developed to interface the Soar agent with its environment. Soar also provides a learning mechanism; however, this was not used within this project due to the perceived impact on system determinism.

In practice, the Soar architecture serves as an inference engine, for which the job is to apply knowledge to the current situation and decide on internal and external actions [16]. The agent's current situation is represented by data structures representing the states of sensors (from the agent's I/O interface) and contextual information (stored in Soar's internal memory). Soar allows easy decomposition of the agent's actions through a hierarchy of operators; operators at the higher levels of the hierarchy explicitly represent the agent's goals, while the lower-level operators represent substeps and atomic actions used to achieve these goals. Soar selects and executes the operators relevant to the current situation that specify external actions, which are applied to the environment through the I/O interface, and internal actions, such as changes to the agent's internal goals.

A practical example can better clarify the nature of a Soar agent; suppose a Soar agent is being used as a thermostat, having two inputs (actual temperature sensor and desired temperature value) and one output (binary heating system activation command). The simplest form of such a Soar agent would have the following operators:

1) If the sensor temperature is less than (desired temperature $-$ 2), then set heating on.

2) If the sensor temperature is more than (desired temperature $+$ 2), then set heating off.

More rules can then be added to this; for example, a fuel sensor might be used to save fuel when the fuel level is low. A third operator would be the following: if the sensor temperature is more than (desired temperature $+$ 1), then set heating off. A preference rule could be used to choose between the second and third operators: if the fuel level is lower than the threshold, prefer the third operator, else prefer the second operator. Soar allows for the combination and organization in hierarchies of large numbers of such rules, leading to very complex agent behavior.

In practical terms, a Soar agent is a dynamic-link library, and C++ classes are provided to control its execution and develop appropriate I/O functions that 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 provides a simulation environment that is indispensible in testing the system.

This approach (based on the integration of Soar agents within Simulink) was initially applied in the development of a health-management system for gas-turbine engines [17–19]. This study proved the feasibility of the Soar/Simulink approach, involving multiple Soar agents

**Table 1 Action types**

| Number | Name | Description |
|---|---|---|
| 1 | Park | Wait until mission start time |
| 2 | Taxi | Move to runway position |
| 3 | Takeoff | Perform takeoff maneuver |
| 4 | Climb | Climb to specified altitude |
| 5 | MMS | Main-mission start |
| 6 | Travel | Travel to position |
| 7 | Reconnaissance | 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 maneuver |

concurrently running within a Simulink environment. However, the application field of gas-turbine engine health management is characterized by a limited problem space and severe constraints; consequently, the capabilities of Soar agents could not be fully exploited and the technology did not bring significant advantages over "conventional" technology with similar functionality. It was then concluded that Soar agents are not technologically practical for this application field; a more suitable application field was identified in autonomous UAV mission management, which presents a larger problem space and less severe safety constraints. This work has been presented in two earlier papers [20,21] focusing on the description of the system itself and on early results; major new results are included in this paper.

### A. Theoretical Background

Before outlining our autonomous mission management system, it is important to define how a flight plan can be computationally expressed. A set of four abstract constructs was defined within this study [20,21]: the objective, the action, the flight plan, and the entity.

The objective is the abstract construct through which a user interacts with SAMMS: a mission is assigned to a UAV by giving it a list of objectives. An objective represents a very high-level task for the UAV, defining a significant part of a mission. The types of objectives can vary greatly depending on the specific type of UAV, but at present, five generic types have been defined: 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 11 properties that can fully define any type of objective previously described; properties include the definition of the objective type and of its unique identifier, the position of the objective, the time constraints it should respect (called time priorities), and the details regarding specific objective types.

The action is an abstract construct that still represents a high-level task but is the finest subdivision that 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 fully describe a flight plan that accomplishes objectives of the types earlier described, as in Table 1. The action I/O object consists of 11 properties that together fully describe it; these properties include the action type; sequence number and parent objective tags; the coordinates of relevant positions; specific values related to flight heading, speed, and altitude; plus additional variables related to specific action types. Using these definitions, a flight plan is a numbered sequence of actions that fully describes a mission. The UAV will then be able to accomplish a mission by executing actions in the expected order.

The fourth and final abstract construct is the entity. The entity represents any external factor that may influence the generation of 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). In short, the entity is a format for all types of external information provided to the UAV. The entity I/O object consists of eight variables that describe its nature, its position and expected movement, its behavior, and how it can affect the UAV mission. While objectives are a user input, entities are expected to be received automatically from an information-gathering system (in military terms, the battlefield network).

Having defined these abstractions, a mission management system is a system that converts a list of objectives into a flight plan (formed by actions), taking into account all entities that are known. Figure 1 shows the flow diagram for such a system and includes indications regarding the various parts (agents) in which it will be divided in the SAMMS architecture.

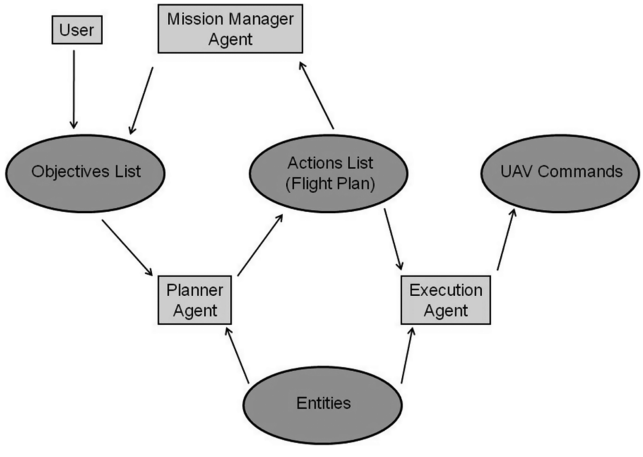



**Fig. 1 SAMMS flow diagram.**

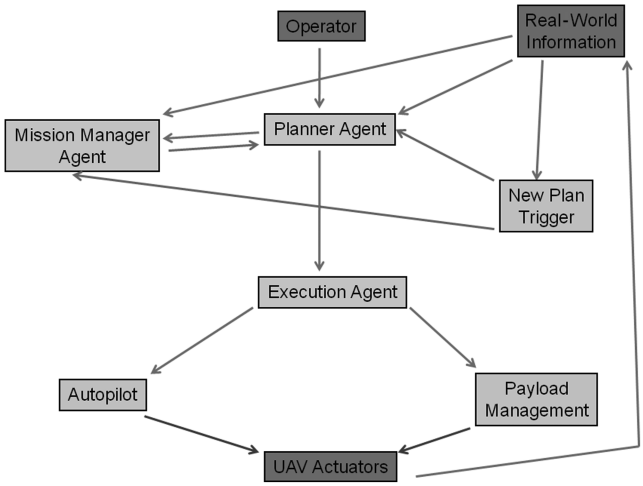



**Fig. 2 Architecture overview.**

### B. Architecture Description

SAMMS is designed to allow a user to instruct a UAV through a set of clearly defined objectives and then leave all details of mission execution to the autonomous system. No further supervision is expected, so the system should be able to develop a flight plan that is both viable and intelligent (ideally, the plan would be optimal, but guaranteeing such optimization is expected to be an issue in terms of computational requirements). The flight plan must be updated during flight according to situational awareness; the system must also be able to directly interface with low-level control algorithms (e.g., an autopilot).

To achieve this functionality, SAMMS was designed as a multiagent system. Three interacting Soar agents (planner agent, execution agent, and mission manager agent) are implemented in a Simulink environment, which provides supporting functionality such as a user interface, real-world sensory input (including onboard sensors and external data), and algorithms for low-level tasks (autopilot and payload management). Figure 2 schematically describes the architecture; the agents will be described separately in the following subsections.

The agents are complemented by a set of functions that use more traditional control techniques. The new plan trigger function 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 planner agent and the mission manager agent (MMA) are triggered separately when the relative conditions are met. The autopilot function consists of a set of standard autopilots that allow control of the flight path of the UAV during the various mission phases; it has two main operating modes of direct and automatic. In direct mode, the execution agent provides commands in the form of desired pitch, yaw, and speed, thus only four control loops (roll-hold, yaw-hold, pitch-hold, and speed-hold) are used. In automatic mode, the execution agent provides commands as desired destination, altitude, and speed, thus six loops are used, with an altitude-hold loop functioning as the outer loop for the pitch-hold loop and a bearing calculation algorithm functioning as the outer loop for the yaw-hold loop. The payload management function translates generic payload commands from the execution agent to actual controls for the payload actuators; it is not currently implemented, as it is largely platform-dependant. Completing the loop is the simulation environment, which is modeled in Simulink and receives input from the low-level functions and provides feedback as real-world sensor information.

### C. Planner Agent

The planner agent is tasked with receiving objectives as input from a user and then fusing them with real-world information (available as entity constructs) in order to obtain a full flight plan (as previously defined, a sequence of actions). The planner agent decides the order in which objectives are sequenced and includes several algorithms that allow improving the flight plan in several ways, such as avoiding dangerous areas, increasing flight speed in order to reach a target before a specified time, or decreasing flight speed in order to save fuel so that all parts of the mission can be accomplished.

Input for the planner includes base airport information (parking position, runway position and heading, altitude at ground level, and for the landing airport if different from the starting airport), a list of objectives from the user, a list of entities that 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 agent cycle, a valid flight plan is not available, so the planner decides to generate the initial one; this is then sent forward (to the execution agent) and stored internally for reference. At each following cycle, the planner will check the validity of the current flight plan and eventually generate a new one if the new plan trigger external function signals that sufficient changes have occurred to situational awareness. In this case, the planner cancels the current flight plan (keeping an internal record of it) and generates an entirely new one, taking into account parts of the old flight plan that have already been executed. Just as with the first one, the new plan is then sent forward and stored internally. The cycle is repeated until the mission is finished.

From a Soar implementation point of view, the planner agent includes 12 states and substates. From the main state, two substates 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.

Q3 Generate-plan is then split into five substates: 1) "old-plan", which copies parts of the old flight plan that have already been executed into the new one being created; 2) takeoff, which adds to the flight plan all actions related to takeoff operations (park, taxi, "take-off", and climb); 3) "main-mission", which develops the main part of the flight plan (during which all objectives are accomplished); 4) "approach", which adds all actions related to landing operations (descent, landing, taxi, and park); and 5) "plan-optimization", which changes the plan in intelligent manners in response to situational awareness.

The main-mission state has two further substates: "plan-sequencing" and actions definition.

During plan-sequencing, objectives are ordered in a sequence using a modified version of the nearest-neighbor (NN) algorithm to solve what is basically a classical traveling salesman problem (TSP) [22]. The NN algorithm is a well-known heuristic to solve TSPs, and although it presents issues (giving bad results under certain circumstances), it is computationally very fast and the small scale of the TSP considered here (counting

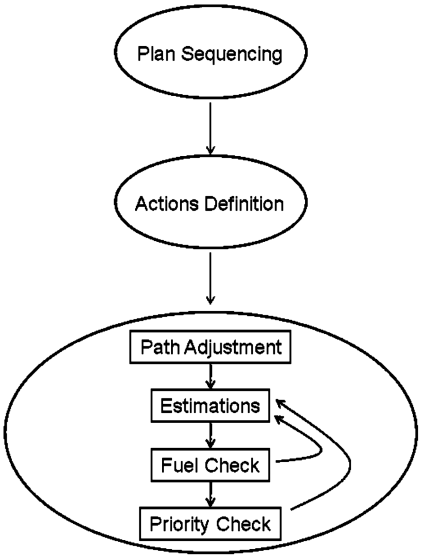



**Fig. 3 Planner agent flow diagram.**

visited points in the tens at most, rather than in thousands) allows us to expect sufficiently good results. Furthermore, the possibility of adding time constraints for objectives means that both the problem formulation and the algorithms needed to solve it have to be modified. The algorithm selects a starting point (base airport), and then it calculates the distance toward every objective and chooses the closest objective as the first visited point; the process is then repeated until all objectives have been visited. An estimate of the time required to reach each objective is calculated while the algorithm is running; if an objective's time priority is not respected, the objective will be moved up the list until the priority is found to be respected. The NN algorithm is then reapplied for all following objectives; for example, if with four objectives the NN algorithm returns the sequence 1-2-3-4 but objective 4 has a time priority that is not respected, objective 4 is moved up the list. If the sequence 1-4 allows to respect the time priority, then the NN algorithm will be reapplied from this; depending on relative positions, it may return the sequences 1-4-2-3 (2 closer to 4) or 1-4-3-2 (3 closer to 4).

During actions definition, every objective is converted into the corresponding list of actions, which are ordered reflecting the sequence of objectives defined during plan-sequencing. This operation varies greatly depending on the type of objective. Analyze-target, attack-target, and orbit-position objectives can be accomplished with a travel action and another action (respectively, a reconnaissance action, an attack action, and a circle action). Search-area objectives are converted into a list of travel actions that implements the chosen search pattern for the specified area. Transit objectives are accomplished by a single travel action but have to be placed at the end of the mission.

The plan-optimization part of the planner is separated from the rest so that it can be as simple or as complex as desired. This part of the planner agent includes algorithms that are designed to improve the flight plan in intelligent manners. It is to be noted that the word "optimization" is not used in a mathematical sense. At this stage, a viable flight plan is already available; however, it does not take into account many types of information that may be available to the UAV. It could be said that, while the rest of the planner implements autonomy (generation of a flight plan), it is the plan-optimization part that implements intelligence (improvement of the flight plan according to several criteria). In Fig. 3, the planner agent flow diagram is shown, highlighting the four plan-optimization algorithms: path adjustment, estimations, fuel check, and priority check. The algorithms are described in Table 2. It is important to note that, to avoid conflicts between the fuel-check and priority-check algorithms, priority check is inhibited if fuel check fires and changes the flight plan. A possible future improvement over this solution would be to allow priority check to override fuel check for top-priority objectives (e.g., objective to be accomplished regardless of risk to UAV survival).

### D. Execution Agent

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

**Table 2 Algorithms in the plan-optimization state**

| Number | Name | Description |
|---|---|---|
| 1 | Path adjustment | Checks whether the current flight path intersects any known entity that represents a threat (hostile presences but also bad weather areas or no-fly zones) and eventually changes the flight plan in order to take a detour around it. The algorithm works by calculating the shortest distance between the danger area center and the flight path, and then comparing it to the danger area radius to determine whether they intersect or not. If an intersection is detected, a new waypoint is added to the flight plan; the waypoint is placed along the perpendicular to the original flight path passing through the danger area center, at a distance sufficient to avoid intersection. |
| 2 | Estimations | Calculates an estimate of the distance covered and the time and fuel needed for each action; this is needed by the other algorithms. The algorithm works by calculating the distance covered for each action (the Haversine formula is used to calculate distance between waypoints), and then using the distance values to calculate time and fuel values. Time is obtained simply by multiplying distance by the expected flight speed in the flight plan. Fuel consumption is calculated by multiplying distance by the amount of fuel used per unit distance; this value is obtained from flight speed using a simple linear model. |
| 3 | Fuel check | Checks whether current onboard fuel is sufficient to accomplish the entire mission; in case fuel is deemed insufficient, it tries to reduce fuel consumption by reducing flight speed by fixed amounts. In case this reduction is still insufficient, the problem is left to another algorithm in the MMA. |
| 4 | Priority check | Checks whether objectives with a time priority are expected to be reached within the time limit; in case this is not true, the algorithm tries to solve the problem by increasing flight speed by fixed amounts for all the actions before the objective. |

**Table 3 Algorithms for each action type**

| Number | Name | Description |
|---|---|---|
| 1 | Park | Very simple action, 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. On a real system, preflight tests would probably also have to be performed during this action. |
| 2 | Taxi | Complex action, as it involves ground navigation (with all its constraints). At present, it is executed by directly steering the UAV toward the planned takeoff position (commands: yaw and low speed), and 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 air traffic control. The algorithm takes account of the maneuvering space needed to steer the UAV. |
| 3 | Takeoff | Once the expected takeoff position is reached, the UAV is steered in the runway direction (commands: yaw and low speed), and then full throttle is set (keep yaw, maximum speed) until the takeoff speed is reached. At this point, a pitch command is given, and then takeoff is considered finished when the UAV has cleared the 15 m level from the ground (obstacle clearance height). Decision speed is not considered at the moment. |
| 4 | Climb | Immediately after takeoff, the climb action keeps the UAV in the takeoff direction (and then in the direction of the first objective) 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. |
| 5/10 | MMS/MME | The main-mission start and main-mission end actions do not correspond to actual flight commands but are needed to correctly define plans and to deal correctly with replanning events. |
| 6 | Travel | This is the most important type of action, and the first type to make use of the automatic mode of the autopilot. It basically sets a great-circle route (shortest distance between two points on a sphere) between the current position and the intended destination, at the specified speed and altitude. The distance to the destination position is continuously verified in order to make decisions regarding the commitment to the objective. |
| 7 | Target reconnaissance | This action involves a pass over a target in order to allow a sensor payload to gather data. After the target position is reached, a turnaround approach waypoint is set, and the UAV then travels toward it before steering back toward the target for the actual data gathering pass. Usually, during a pass, the desired altitude is different from cruise altitude, so this is also changed |
| 8 | Target attack | This action is very similar to target reconnaissance but can use different parameters in determining the type of approach and uses a different type of payload |
| 9 | Circle | In this action, four waypoints forming a diamond are calculated around the central position. The UAV then cycles through those in a clockwise (or anticlockwise) direction until the specified time limit is reached |
| 11 | Descent | In this action, after the expected landing position has been reached in flight, two waypoints are calculated and reached using the automatic mode of the autopilot. These waypoints basically draw an ideal descent path that is in line with the runway; two waypoints are needed in order to account for the maneuvering space required |
| 12 | Landing | This action makes use of the direct mode of the autopilot and has the UAV descend at a specific angle, then perform a flare maneuver when close to the ground, and finally stop when ground contact has been ensured. |

action with real-time sensor data (Global Positioning System, attitude, airspeed, etc.), sends commands to the lower-level control systems, namely, the autopilot and the payload management system. Conceptually, the exag is very simple: it starts from action 1, passes the related commands to low-level controls, verifies the execution of the action, and then goes on to action 2 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; each of the action types outlined in Table 1 has a dedicated execution algorithm.

The exag outputs two very important sets of data: planner feedback, with information such as the current action being performed and the commitment to an objective; and the commands, which represent direct input to the UAV low-level controls and include data for the two autopilot modes: direct mode (speed, pitch, roll, yaw, brakes) and automatic mode (speed, altitude, and initial and final positions). The exag selects the current action to be performed, and then it calculates what commands need to be given in the light of action details and real-time sensor information. The action algorithms are described in Table 3.

### E. Mission Manager Agent

The MMA 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 authority to change the objectives (that are otherwise exclusively defined by the user) and to add new ones. For example, it can cancel a secondary objective that is close to a newly detected threat or change the parameters of a search mission if a minor fault places stricter endurance limits on the UAV or, finally, take advantage of targets of opportunity. The type of autonomy brought forward by this agent allows for intelligent decisions to be made; it is to be noted however that, in certain cases, this level of autonomy might be excessive. While the agent will react predictably to the external situation, it is in fact autonomously modifying the objective list; this might not be desirable for certain applications, thus this functionality was embedded in a separate agent from the planner and, in fact, the MMA can be excluded from the SAMMS architecture without consequence (apart from losing the capabilities it brings).

The MMA needs a large amount of information to work: the list of current user objectives, the entity information, airframe information, and the current flight plan generated by the planner (together with the estimates obtained by the estimations algorithm). The agent works similarly to the planner agent, as it is triggered by an external function and is waiting for changes during most of the time. When the trigger function detects that a new check of the current situation is needed, the MMA goes through a list of algorithms that determine and apply necessary changes to the objectives. There are basically three types of operations that the MMA can do: modify an objective, remove an objective, or add a new objective.

There are five types of inconsistency that can determine a change to an objective or its removal; these are described in Table 4, together with an explanation of how the MMA acts to solve them.

The other type of action that the MMA can perform is the addition of a new objective. This is related to search objectives specified by the user. Search objectives can be specified as pure searches, search-and-analyze missions, or search-and-attack missions; the latter two imply that if a new entity is detected within the search area, the related action should be performed on it (either analyze or attack). The MMA looks at the current entity status and, if a new entity is detected within the search area, it adds a new objective of the analyze or attack type to the list given by the user.

It is important to understand that giving the authority to autonomously change, add, or remove objectives to the system might not be desirable in certain situations. Thus, this functionality is implemented in an agent that is separate from the planner, allowing to completely disregard it without losing the functionality provided by the planner. The ability to respond to a dynamic environment is one of the driving ideas behind the entire system; however, this inevitably comes at the cost of losing control over the UAV's operation. Separate implementation of the algorithms with the authority to change mission objectives allows SAMMS to be more flexibly configured for the actual UAV on which it is used, depending on the specific needs (for example, a transport mission in a nonhostile environment would likely not require the MMA to be active).

**Table 4 Flight plan inconsistency types and solutions**

| Number | Name | Description |
|---|---|---|
| 1 | Priority | Occurs when an objective has a time priority for which the planner estimates imply that the time priority will not be respected (the objective cannot be executed before the specified time). At this stage, both the plan-sequencing and the plan-optimization algorithms have failed in generating a plan that respects the priority, thus the MMA actually decides to ignore the priority and advises the user of this decision (this type of problem can be caused by an incorrectly set or unrealistic time priority). |
| 2 | Target position | Occurs when the target for an analyze or attack objective is moving. In such cases, position information entered by the user needs to be updated using the corresponding entity information. |
| 3 | Fuel | Occurs when current onboard fuel is deemed insufficient to complete the mission, even after the planner has tried to reduce fuel consumption by decreasing flight speed. In this case, two possible courses of actions can be taken: if a search objective is present, the resolution of the search can be increased, so that the distance to be covered is reduced; otherwise, the algorithm chooses an objective to be canceled (based on two factors: execution priority of objectives, and relative distance between the mission starting point and the objectives). |
| 4 | Threat | Occurs when an objective is placed within the danger area of an entity. In this case, the algorithm will decide to remove the objective if its execution priority is lower than the threat level of the entity. |
| 5 | Payload | Occurs when airframe status data indicate that the payload related to a certain type of mission has failed. In this case, objectives of that type are removed, since they cannot be accomplished. |

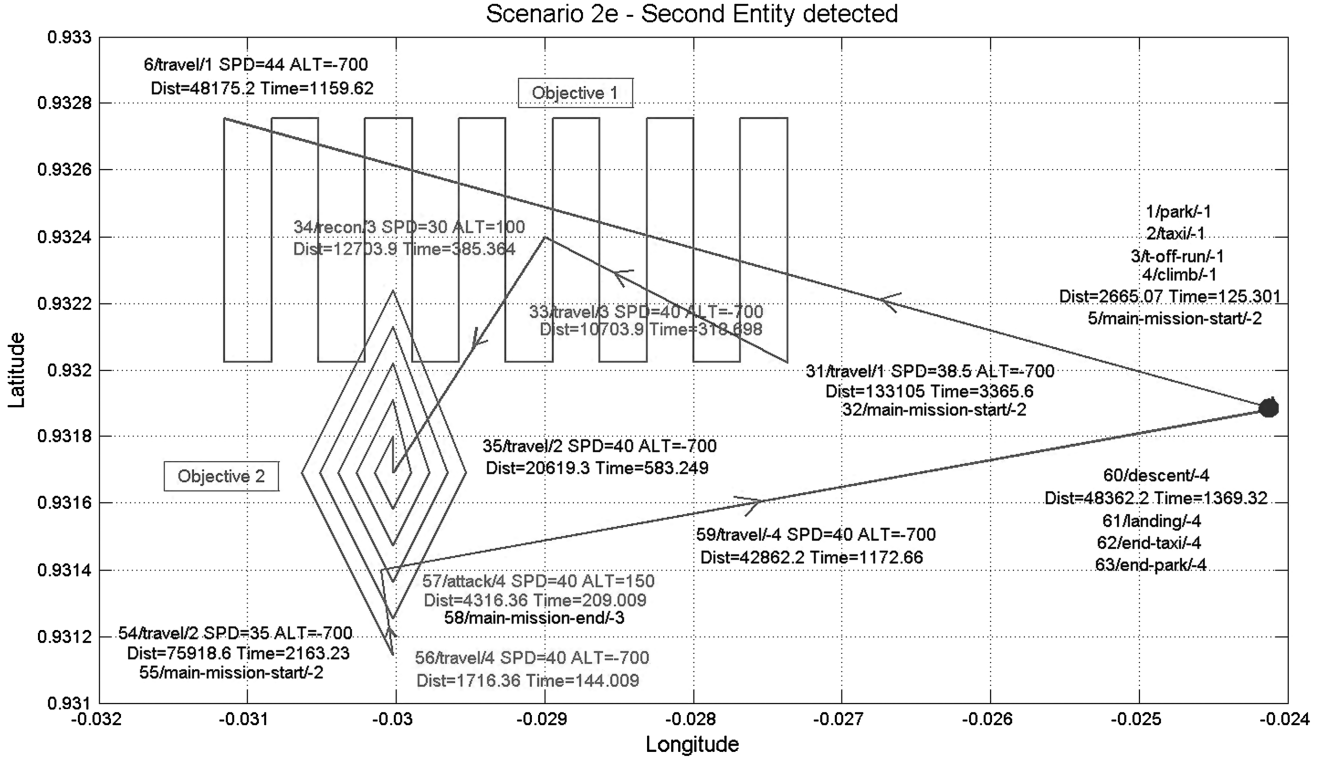
## III. Simulation and Testing

To verify the functionality of SAMMS, a dedicated simulation environment has been set up. During these simulations, realistic input is fed to SAMMS and the output is fed to a UAV dynamic model controlled through a standard autopilot, aiming to prove SAMMS's capability to guide the UAV through an entire mission. The simulation environment is based on a Simulink model of the Pioneer UAV. This was chosen as it is representative of the type of UAV toward which SAMMS is tailored (small, slow-flying, and low cost).

The model is based on the generic aircraft model that was released by Campa in 2004 [23], although several modifications were implemented to simulate ground operations (taxi, takeoff, and landing). This model uses nonlinear equations of motion; however, aerodynamic forces are calculated using a linear model and the calculation of thrust is simplified. Thus, the model does not provide high fidelity, but this has been deemed acceptable, since the purpose of the simulation is not to validate the low-level control algorithms but to test the mission management algorithms that operate on a different timescale and do not require tailoring for the specific aircraft being used.

The purpose of the simulations is twofold: primarily, to prove that the flight plans generated by SAMMS in various situations are correct; and secondarily, to verify that the system respects real-time requirements and is capable to control the aircraft during all flight phases. Two sets of data are logged: the flight plan that is the main output of the planner agent, and the flight data generated by the model. From these, a set of graphs is derived, displaying the flight plan and the flight trajectory resulting from its execution.

Since the amount of input variables for the system is very large, verification of all possible input configurations is practically impossible. For this reason, a set of seven test scenarios has been prepared, with each scenario including base airport information, a list of objectives, and a list of entities. The scenarios are representative of the situations that SAMMS might encounter and provide scenario variants that allow testing of particular algorithms. Four of these scenarios will now be analyzed by providing a description of the scenario, a plot of the resulting flight plan, and a plot of the simulated trajectory of the UAV. It is to be noted that scenario results are entirely repeatable.




**Q4 Fig. 4 Scenario 2e flight plan (SPD: speed, m/s; ALT: altitute, m).**

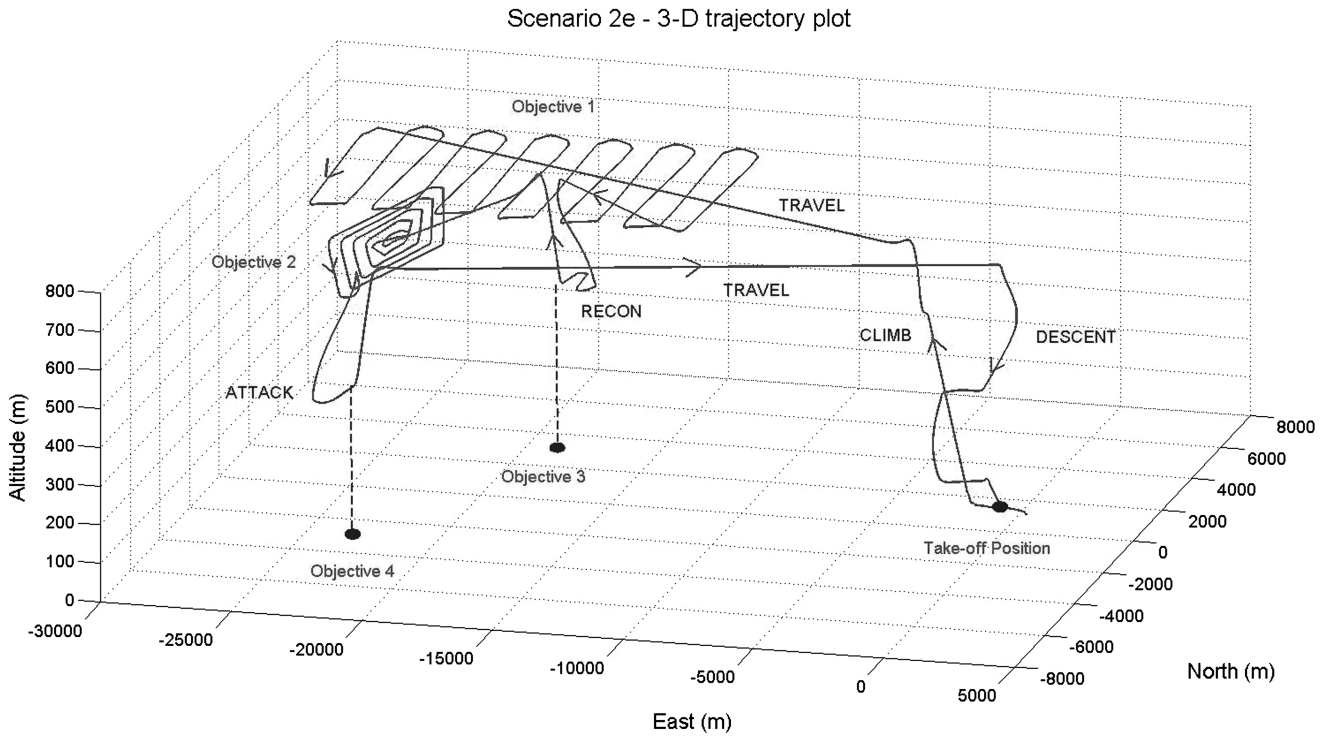



**Fig. 5 Scenario 2e trajectory plot.**

### A. Scenario 2

Scenario 2 is a relatively simple scenario for which the main purpose is to test the different search patterns that are implemented within SAMMS. Two types of search area can be defined: a rectangle, which will be covered using a standard parallel track search pattern; or a circle, which will be covered using an expanding diamond spiral pattern. The 2e variant of the scenario introduces a time priority for the rectangle search objective, so that the priority-check algorithm (see Sec. II) intervenes to ensure completion of the objective within the time limit. Also, new detected entities are added during the mission to simulate the detection of targets during searches. As these entities are detected, new objectives are added by the MMA, and the UAV proceeds to perform the intended action on the new targets.

Figure 4 is a plot of the final flight plan for scenario 2e, therefore including the two objectives that are added when the new entities are detected. The first search is a search-and-analyze objective (thus, a new analyze objective is added), while the second search is a search-and-attack objective (thus, a new attack objective is added). In this scenario, the UAV takes off and lands at the same airport.

Figure 5 is the three-dimensional (3-D) plot of the UAV flight trajectory for this scenario. It can be noted that searches are conducted at the normal cruise altitude, while analyze and attack objectives involve low-altitude flight (flight altitudes are defined as parameters depending on the UAV configuration).

In Fig. 6, it is possible to see a plot of flight speed versus time regarding the scenario, which demonstrates functionality of the priority-check algorithm. This algorithm checks whether objectives that have to be completed within a predefined time limit are expected to be accomplished in time using the time estimates. In case a priority is not respected, the algorithm tries to solve the issue by increasing flight speed for the corresponding part of the mission (from the beginning until the objective has been accomplished). This is necessary since the plan-sequencing algorithm orders the objectives on the basis of inaccurate estimates of the time needed (the flight plan is not known at that stage). In the figure, it is possible to see that flight speed is increased for the first part of the mission (first search) in order to ensure that the time limit is respected (flight speed increases by 20% from the standard 40 m/s cruise speed and 35 m/s search speed). Thus, the UAV covers the first part of the mission (which is time-limited) at a higher speed to ensure it is covered in time, and then it accomplishes the remaining part of the mission at the normal cruise speed (search speed is set to a lower value to simulate the fact that the sensors might need a lower speed to work properly). It is possible to notice that, when descending (during the reconnaissance and attack actions and during final descent), the UAV reaches high speeds, which can be

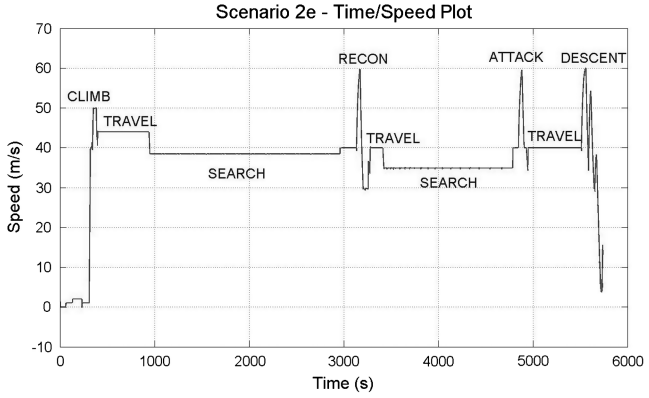



**Fig. 6 Speed plot highlighting priority-check algorithm.**

**Table 5 Scenario 3 objectives and time priorities**

| Identifier Tag | Type | Priority (initial) | Priority (change) |
|---|---|---|---|
| 1 | Circle | Immediate | Accomplished |
| 2 | Reconnaissance | N/A | Immediate |
| 3 | Transit | N/A | N/A |
| 4 | Reconnaissance | 2500 | N/A |
| 5 | Reconnaissance | 1500 | 1500 |
| 6 | Attack | N/A | 2500 |

Q5 [a]N/A, no value available.

noticed as spikes in the plot. In fact, the autopilot has to include a speed limiter function, which is activated when the UAV reaches a speed of 60 m/s/.

At the current stage of implementation, the search patterns present limited functionality; they cannot be orientated (especially limiting for the parallel track pattern) and cannot be interrupted (either they are completed or they will be begun from scratch). However, these two features are planned improvements that are feasible within the SAMMS architecture. For example, this would allow for the rectangle search objective to be carried out starting from the opposite corner (and yielding a reduction of used fuel) or for the new target-reconnaissance objective to be executed when the UAV passes close to it during the search (interrupting the search and reprising it later).

### B. Scenario 3

Scenario 3 is a medium-complexity scenario for which several variants were developed in order to test a wide range of SAMMS capabilities. The scenario revolves around a set of six objectives: three of type target reconnaissance and one each for the target-attack, circle, and transit types. Several time priorities are assigned, as in Table 5, and change during flight in some of the scenario variations.

Figure 7 shows the initial flight plan obtained for the scenario with the original set of time priorities; it is to be noted that a transit objective is present, so the UAV is expected to takeoff and land at different airports. In the plan-sequencing state, the time priorities override the NN algorithm (the objective sequence would be: 1-2-5-6-4-3, but it becomes 1-5-4-6-2-3), so that the flight path is not the shortest possible.

A replanning event is scheduled to occur after objective 1 is completed: the time priorities are changed, and thus the objective sequence is reordered. Figure 8 shows the updated flight plan; with objective 1 already accomplished, the new objective sequence is 1-2-5-6-4-3. The UAV is already flying toward objective 5 but switches course in order to execute its new instructions (e.g., immediate time priority for objective 2). It is also to be noted that, in order to achieve the 1500 s time priority for objective 5, a speed increase is needed; the new flight plan expects flight speeds 20% higher than normal until objective 5 is accomplished. This type of capability is particularly important for SAMMS: not only the flight plan is updated with the new objective sequence, but it is adapted to ensure that all mission requirements are still met.

Figure 9 instead shows the 3-D trajectory for the entire scenario, including the replanning event; it is possible to notice the takeoff and climb trajectories, as well as the diversion occurring because of the replanning event. Also, the spiraling trajectories adopted during the reconnaissance, attack, and descent actions can be noted, as well as the loitering pattern adopted during the circle action.

As previously stated, several variations of this scenario were developed in order to test specific algorithms. Some of the resulting flight plans will now be shown, although without the corresponding 3-D trajectory plot. In all these scenario variations, no replanning event is scheduled, in order to highlight the operation of a particular algorithm; however, all algorithms are in fact meant to operate concurrently.

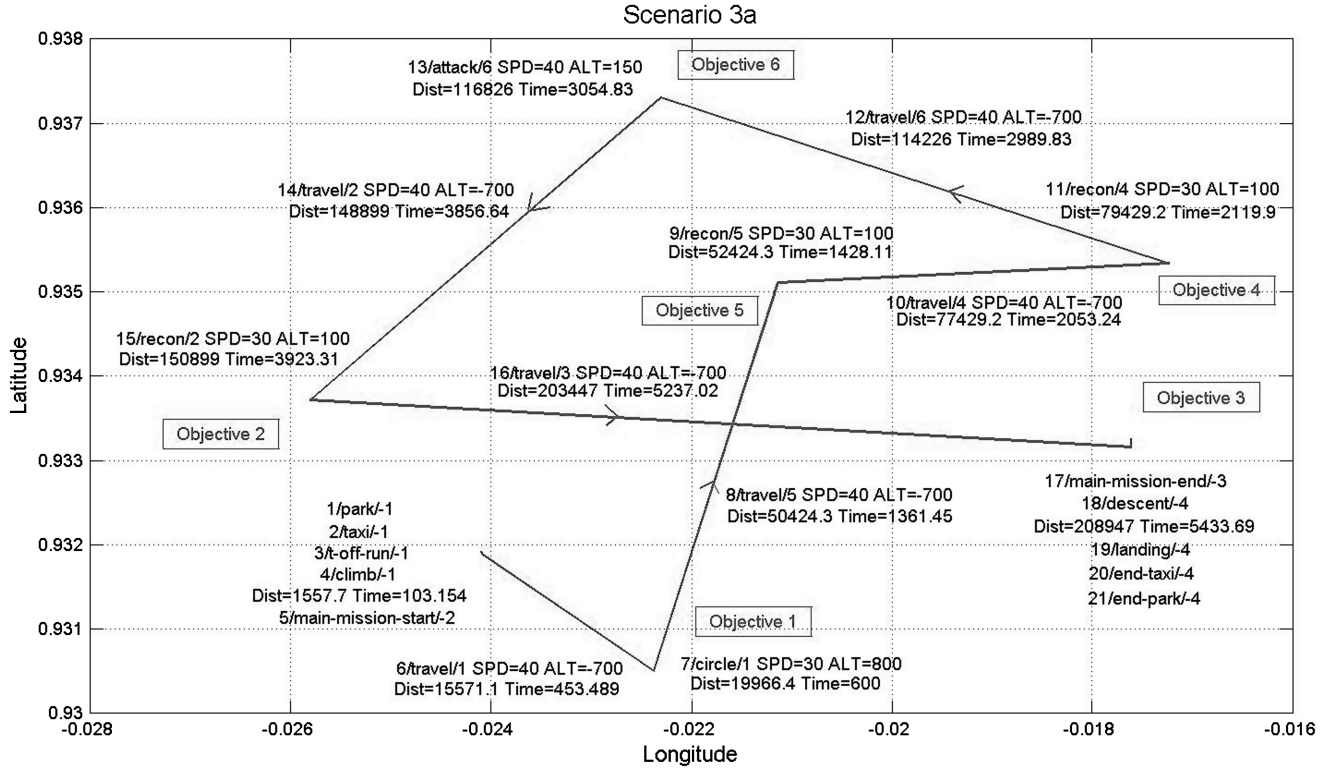



**Fig. 7 Plot of initial scenario 3 flight plan (SPD: speed, m/s; ALT: altitute, m).**

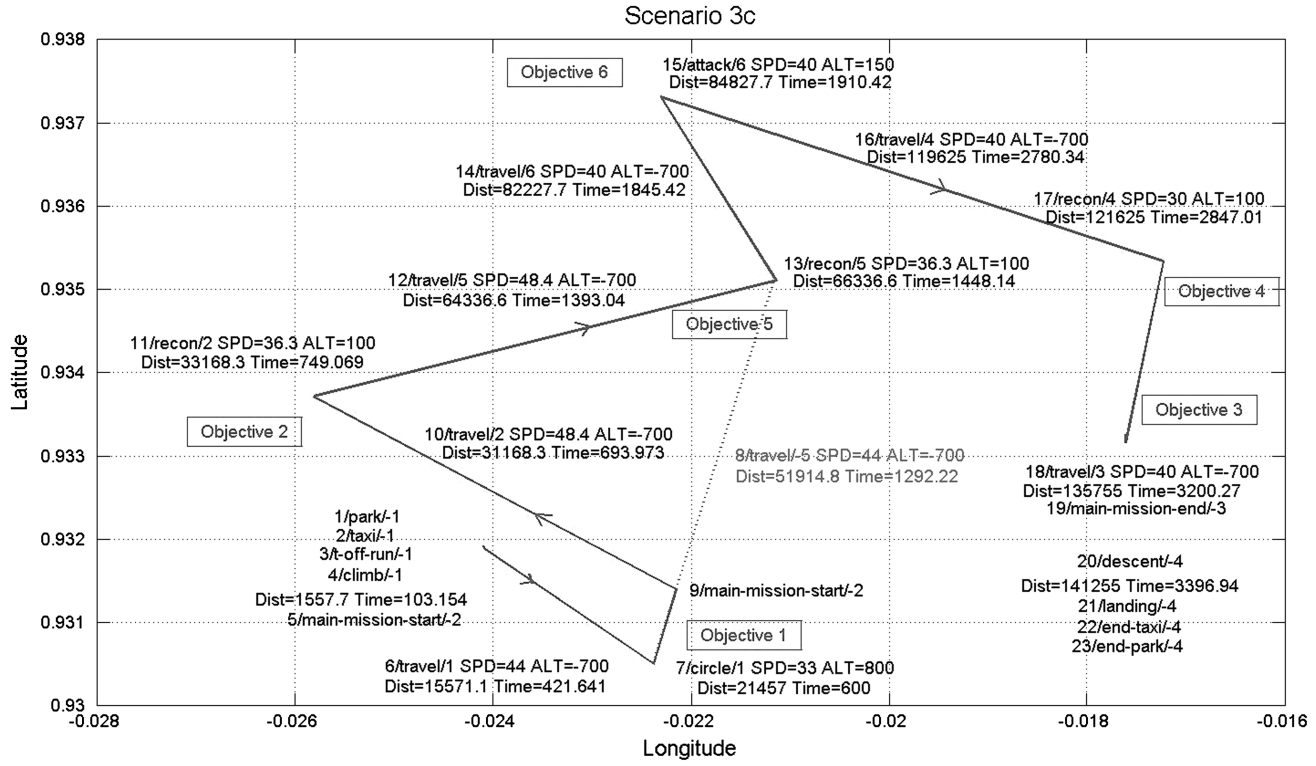



**Fig. 8 Plot of initial scenario 3 flight plan (SPD: speed, m/s; ALT: altitute, m).**

Figure 10 shows the flight plan for scenario variation 3d; this variation has the same objectives but introduces three dangerous entities that trigger the mission-path-adjust algorithm. The entities are represented as circles in the figure; it is possible to see that two travel actions intersect them. The travel action between objective 5 and objective 4 (shown as a dashed line in the figure) intersects with the threat range of entity 7; thus, a new waypoint is added, and the flight plan modified to include a new travel action for objective 4. The travel action between objective 6 and objective 2 intersects with the threat range of entity 6; again, a new waypoint is added to avoid the area. Entity 5 does not interfere with the flight plan and causes no modifications. It can be noted that the new waypoints cause an increase in distance and time estimates (compare with Fig. 7); the time priority for objective 4 set at 2500 s is still respected, so no speed increase is needed.

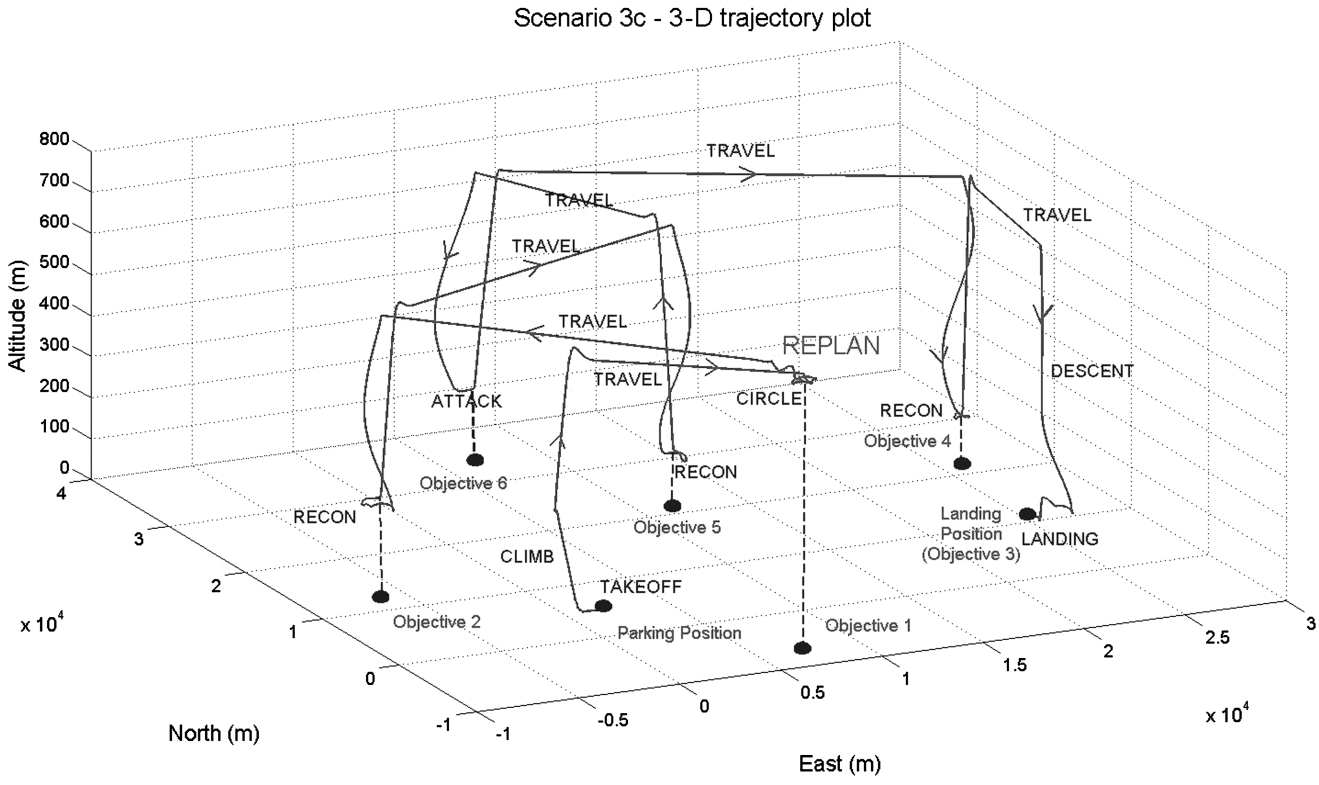



**Fig. 9 3-D trajectory plot for scenario 3.**

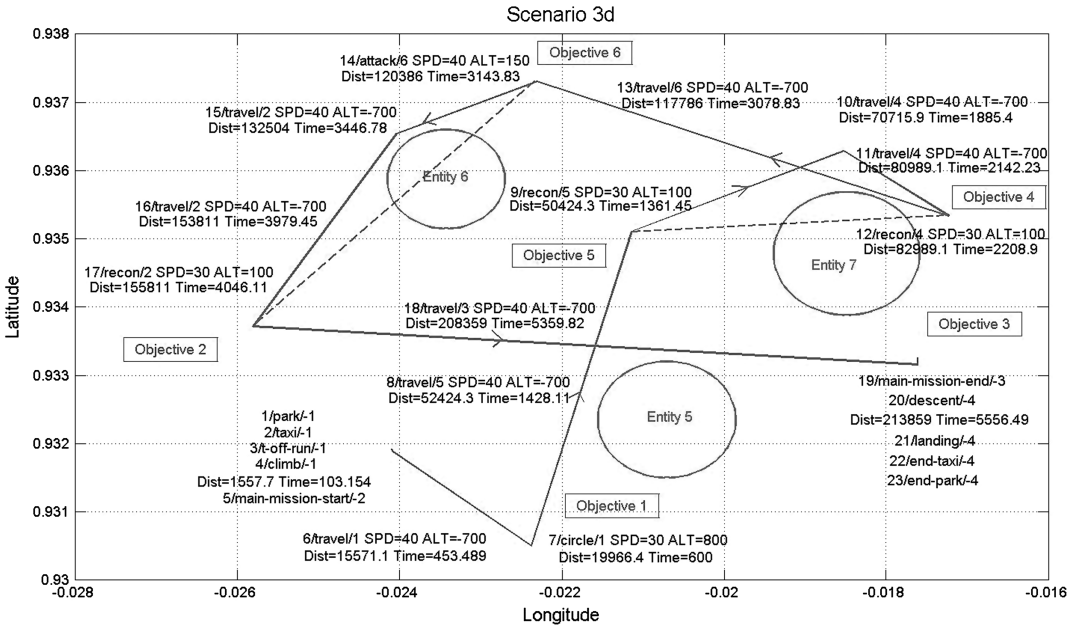



**Fig. 10 Plot of scenario variation 3d flight plan (SPD: speed, m/s; ALT: altitute, m).**

Figure 11 shows the flight plan for scenario variation 3f; the usual six objectives are defined but the amount of fuel available to the UAV is set at 24 kg (while the fuel consumption estimate for the original plan, as in Fig. 7, is 28.35 kg). This causes the reduce-speed algorithm in the planner to reduce flight speed by 19%, in order to save enough fuel to complete the mission. It is possible to see that the time estimate for the completion of objective 5 in the updated plan is 1622.36 s; objective 5 has a time priority value of 1500 s, which with the reduced flight speeds is not respected. Since the lack of fuel problem is obviously considered more important, the plan is accepted and eventually a priority problem is detected by the MMA. In the scenario, objective 5 has its time priority modified from the value of 1500 s to the value of 1682.36 s (obtained as its corresponding time estimate 1622.36 s, increased by a fixed amount of 60 s). The user interface should be designed so that this change will be communicated to the UAV operator.

If the available amount of fuel is further reduced, flight speed reduction will not be sufficient to ensure that the flight plan is completed and the MMA must decide to abort one of the objectives. In scenario 3g, the UAV has 18 kg of fuel at its disposal, and a fuel problem is detected by the MMA; since no search objectives are present, the decision to cancel an objective is made. All objectives have the same execution priority; thus, the canceled objective is objective 4, which is the farthest from the starting airport. To complete the mission, the planner still needs to reduce speed by 19%; the total estimated fuel consumption is now 17.86 kg. Note that the time priority for objective 5 is again not respected. In scenario 3h, the

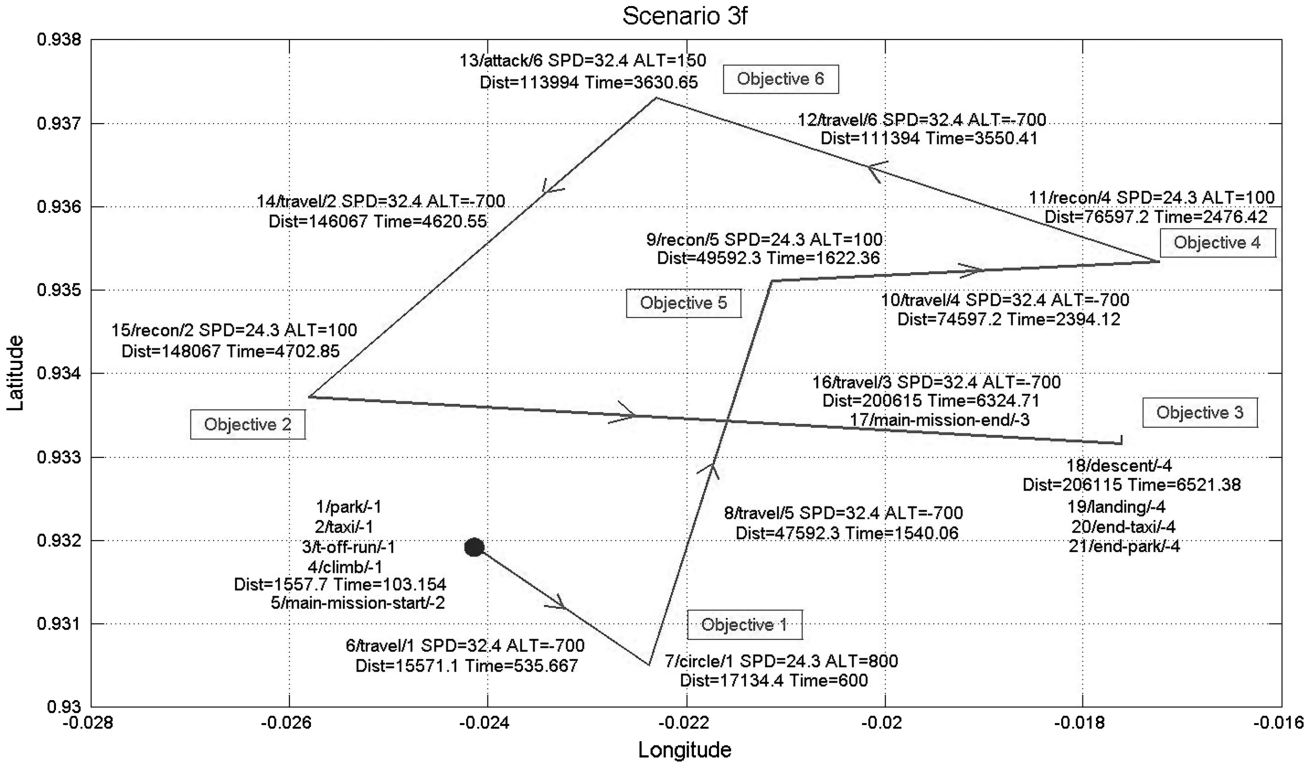



**Fig. 11 Plot of scenario variation 3f flight plan (SPD: speed, m/s; ALT: altitute, m).**

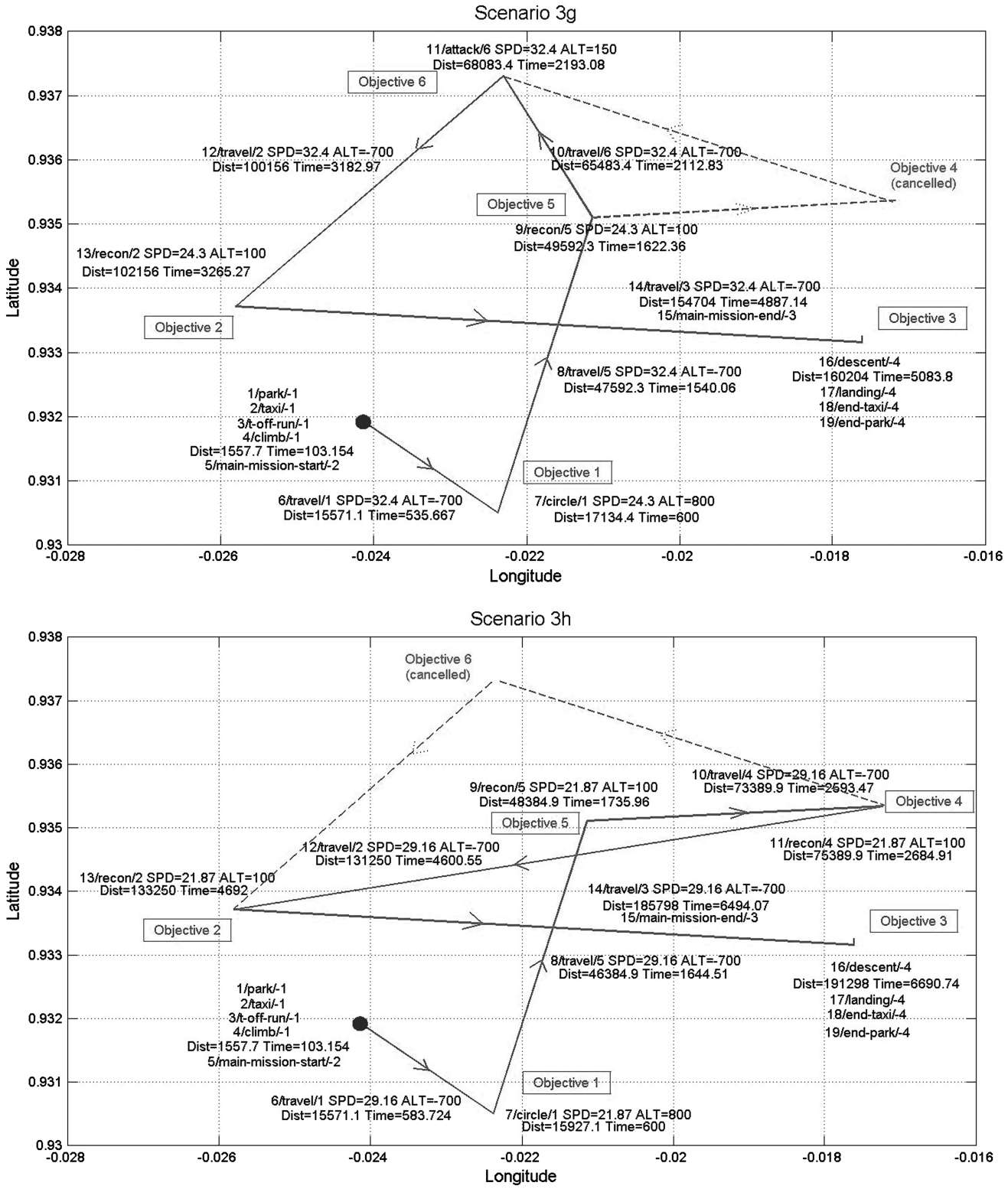



**Fig. 12 Plot of scenario variations 3g and 3h flight plan (SPD: speed, m/s; ALT: altitute, m).**

UAV has 19.2 kg of fuel and, in this case, objective 6 is chosen to be canceled since it has a lower execution priority. A reduction of flight speed by 27% (three iterations of the reduce-speed algorithm) is needed to ensure that the plan will be completed with the current amount of fuel; with these speed values, the fuel consumption estimate is 18.65 kg. In this case, the time priorities for objectives 4 and 5 are not respected. The flight plans for both scenario variations 3g and 3h are plotted in Fig. 12.

### C. Scenario 4

Scenario 4 is very complex and designed to test the ability of SAMMS to deal with many objectives. It involves eight different objectives of different types (analyze target, attack target, orbit position, search area). The 4a variant does not include a replanning event; however, it is designed to test the threat avoidance algorithm.

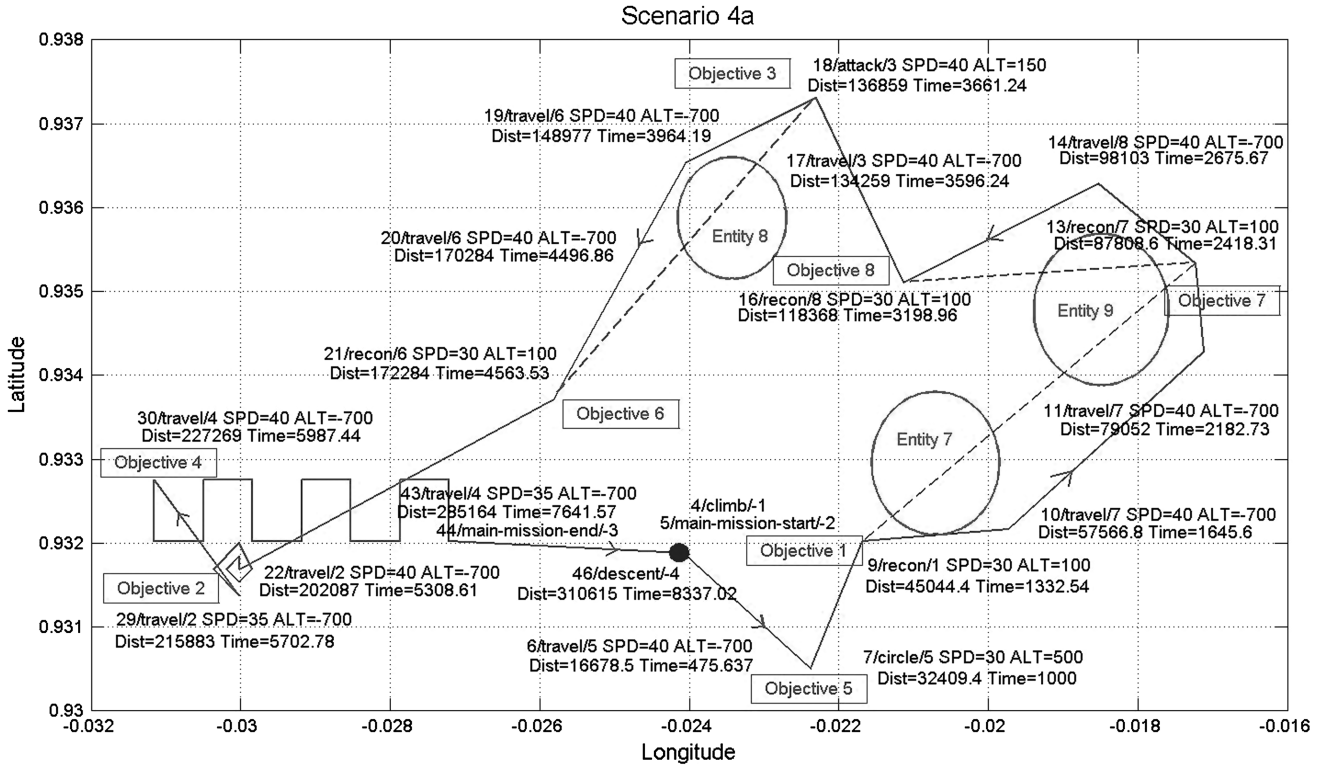



**Fig. 13 Threat avoidance algorithm demonstrated in scenario 4a (SPD: speed, m/s; ALT: altitute, m).**

In Fig. 13, the two-dimensional plot of the flight plan is shown, and operation of the threat avoidance algorithm is demonstrated. The dashed line represents the flight plan without threat avoidance. The algorithm works by introducing new waypoints, as can be seen on the actual flight plan that is represented by the solid line. Circles in the plot represent the danger areas that are introduced. The figure also shows distance and time estimates for all of the main waypoints of the flight plan.

The corresponding UAV trajectory is plotted in Fig. 14. For visual clarity reasons, the 3-D plot is represented with inverted axes compared to the flight plan plot. It is possible to note the circling trajectory used during orbit objectives (this is the first objective that is executed) and the order in which objectives are executed. Within the scenario, objective 5 (orbit objective) has an immediate priority, which means it is supposed to be prioritized over any other objective (only one objective can have an immediate priority at any time). The other objectives have no time priorities,

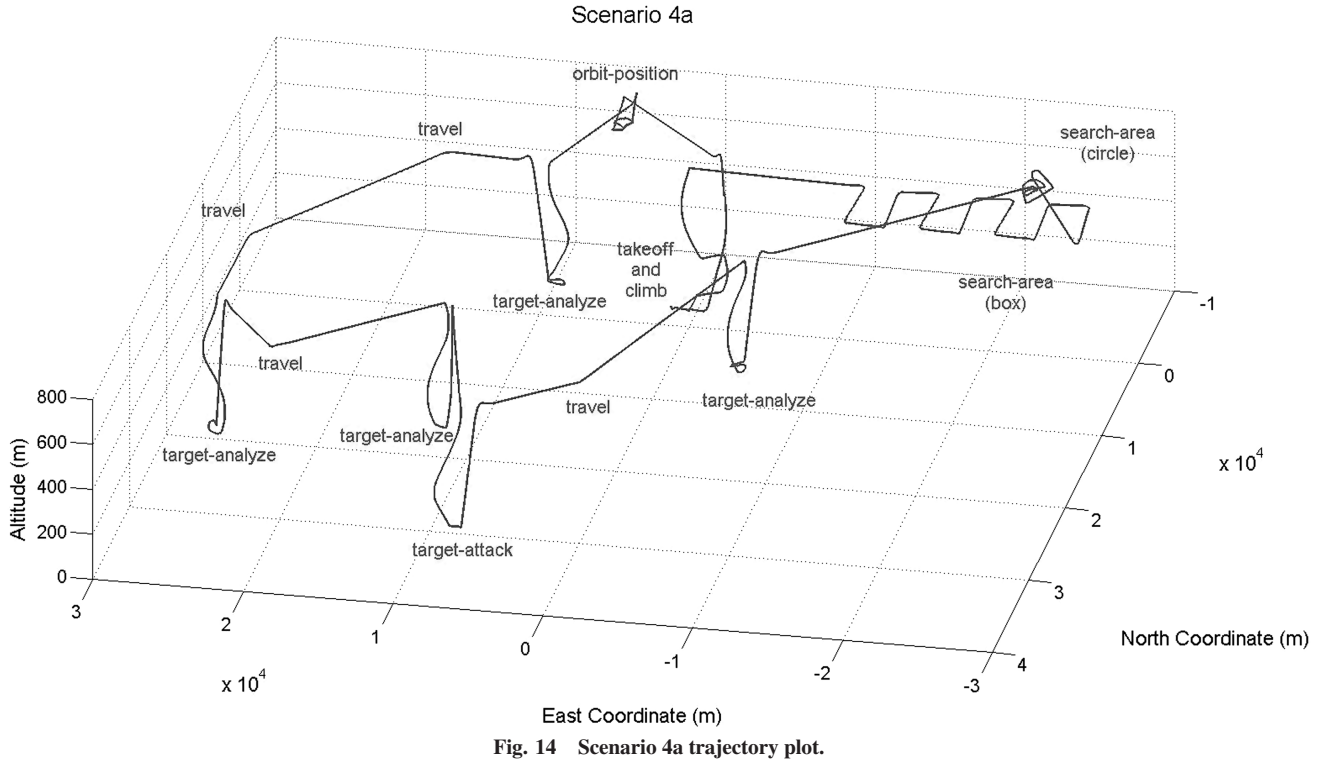



**Fig. 14 Scenario 4a trajectory plot.**

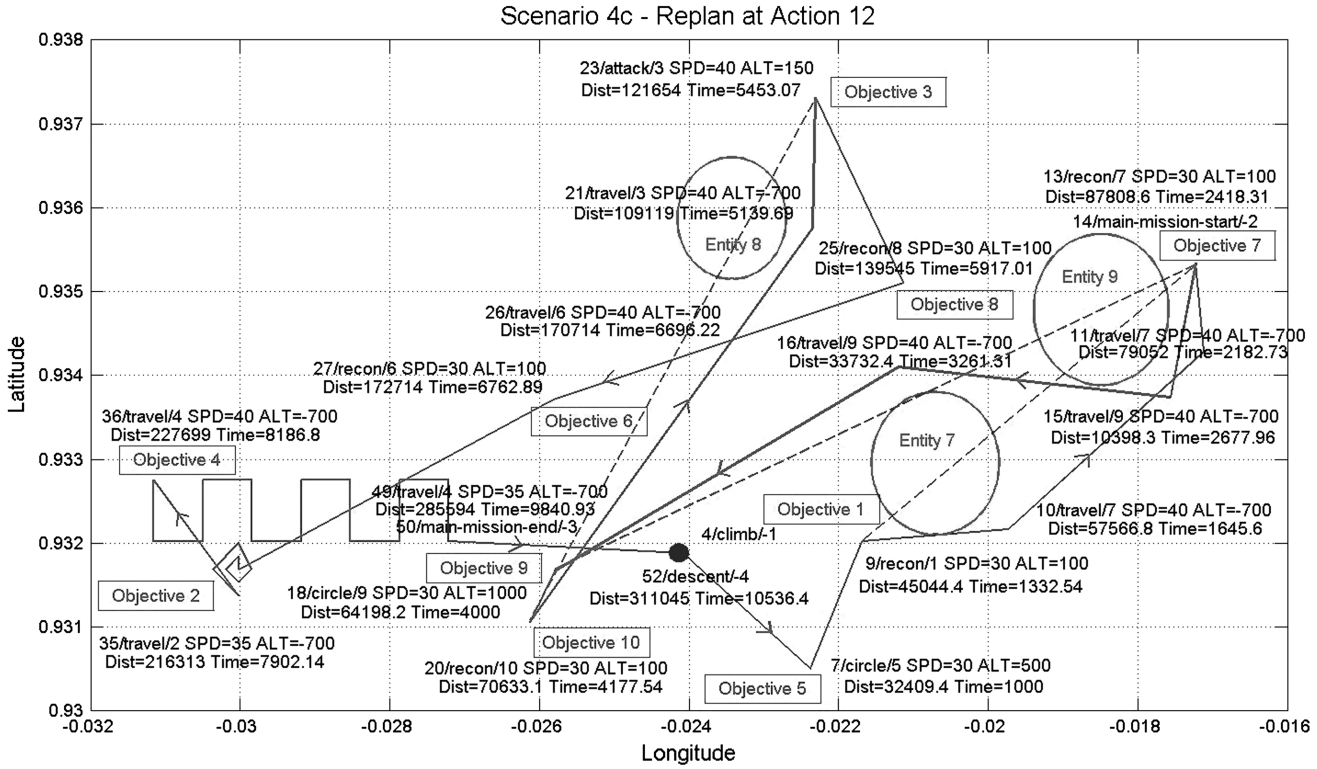



**Fig. 15 Plot of scenario 4c flight plan, replanning event at action 12 (SPD: speed, m/s; ALT: altitute, m).**

thus SAMMS chooses the order by trying to minimize the distance to be covered. The scenario does not involve a transit objective, thus the UAV takes off and lands at the same airport. Finally, danger areas are avoided by flying through the waypoints that are added by the threat avoidance algorithm.

To further test SAMMS when dealing with many objectives, a scenario variation including a replanning event was prepared. In short, scenario variation 4c introduces two new objectives, one of which (objective 9) has an immediate priority; however, the time at which the replanning event is scheduled can be changed, and the updated flight plan varies accordingly. Figure 15 shows the flight plan obtained when the replanning event occurs after action 12 (last travel leg to reach objective 7). Since objective 7 has almost been reached, it is completed before heading toward objective 9; the travel action toward objective 9 intersects two danger areas that are properly avoided and, after objectives 9 and 10 have been completed, objective 3 is scheduled before others because of its time priority. In the plan shown in Fig. 16, the replanning event is scheduled while performing action 15 of the original plan (while traveling toward objective 8). Objective 8 is completed before heading toward objective 9; no time priority issues arise with objective 3, and so objective 6 can be completed before yielding a reduction in traveled distance. Finally, Fig. 17 shows the flight plan obtained when replanning occurs during action 18 (while objective 3 is being completed). In this case, objectives 9 and 10 are simply added after objective 3 and before objective 6, and the rest of the flight plan is executed as originally planned. The corresponding trajectory plots for these flight plans are not shown, as they would not bring significant insight.

### D. Scenario 5

Scenario 5 is a scenario of medium complexity, designed to test SAMMS's ability to deal with multiple replanning events. In the 5e variant, three replanning events occur: every time the flight plan is updated.

In Fig. 18, the evolution of the flight plan for scenario 5e is shown. The scenario begins with four objectives; no transit objective is assigned, so the UAV is expected to takeoff and land at the same airport. Shortly after takeoff, a new objective with an immediate time priority is added; the UAV updates the flight plan and diverts toward it. A second replanning event is triggered later while accomplishing objective 2; objective 3 is removed and the flight plan updated accordingly. The third replanning event occurs when a transit objective is added; this means that the UAV should land at a different airport from the starting one, and a final update to the flight plan is done.

In Fig. 19, the 3-D trajectory plot for scenario 5e is shown; after the first replanning event, a diversion in the UAV flight can be noted. The diversion happens because the execution agent is not yet committed to complete the current objective. During other replanning events, the exag is instead committed to finish the current objective, so the UAV does not divert immediately. The rules used by the execution agent to decide commitment to finish the current objective depend on several factors; generally, the agent will be committed if a significant part of the objective has already been accomplished.

It is to be noted that the SAMMS architecture is designed to be able to deal with any number of replanning events. The new-plan trigger function is present to avoid excessive replanning: only significant events should trigger the generation of an updated flight plan. The three events that can be seen in scenario 5e are all changes in the user-provided mission objectives, which are obviously significant and consequently trigger replanning.

## IV. Hardware Implementation

In the previous sections, SAMMS has been thoroughly described and tested with simulations. In this section, details regarding its planned implementation will be discussed.

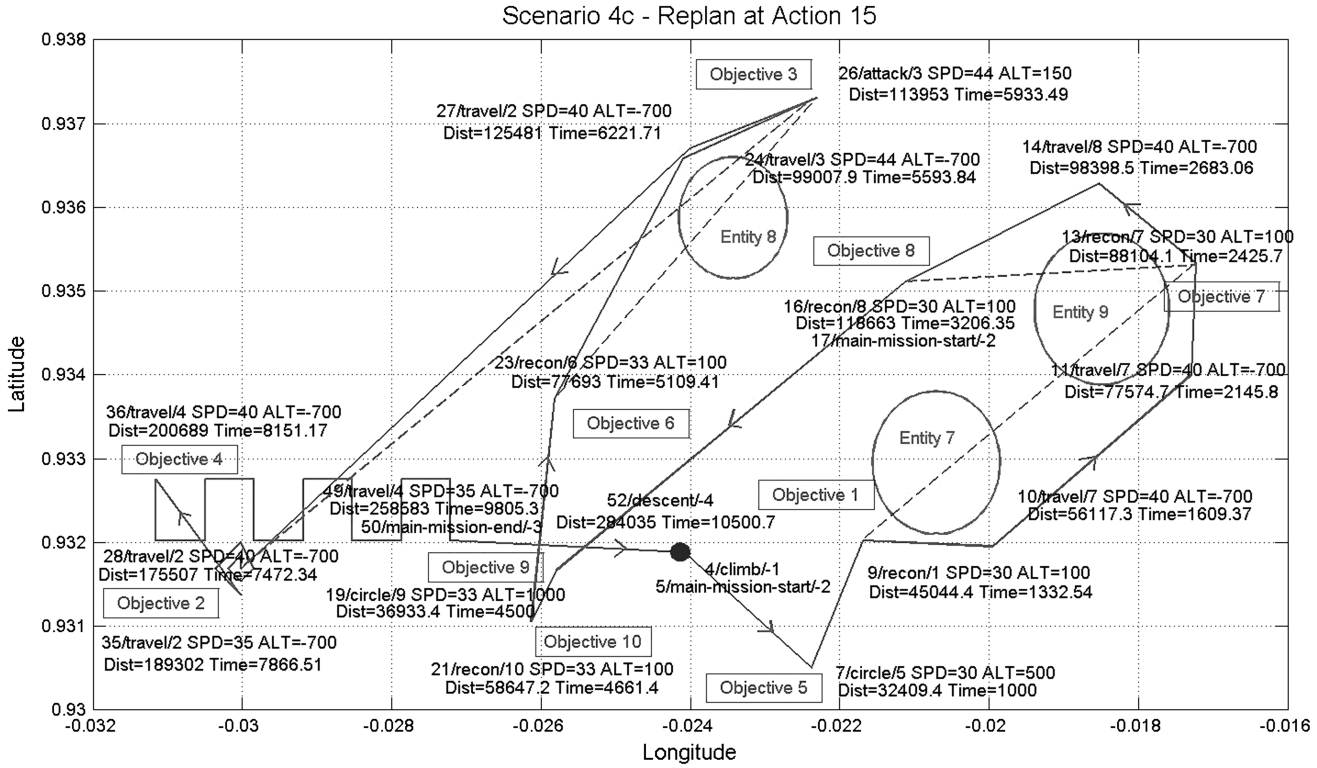



**Fig. 16 Plot of scenario 4c flight plan, replanning event at action 15 (SPD: speed, m/s; ALT: altitute, m).**

While the SAMMS architecture is not constrained to a specific UAV class (UAV performance is provided to SAMMS by a set of easily changeable parameters), it has been designed with application on small low-cost UAVs in mind. A key goal of the project has always been ensuring that the system could run on a PC/104 board running a real-time operating system such as QNX or VxWorks Tornado.

This phase of development has not been reached, so the following observations cannot currently be proved and are purely based on the observation of the simulations and from experience derived from other projects. The SAMMS architecture is currently executed within a

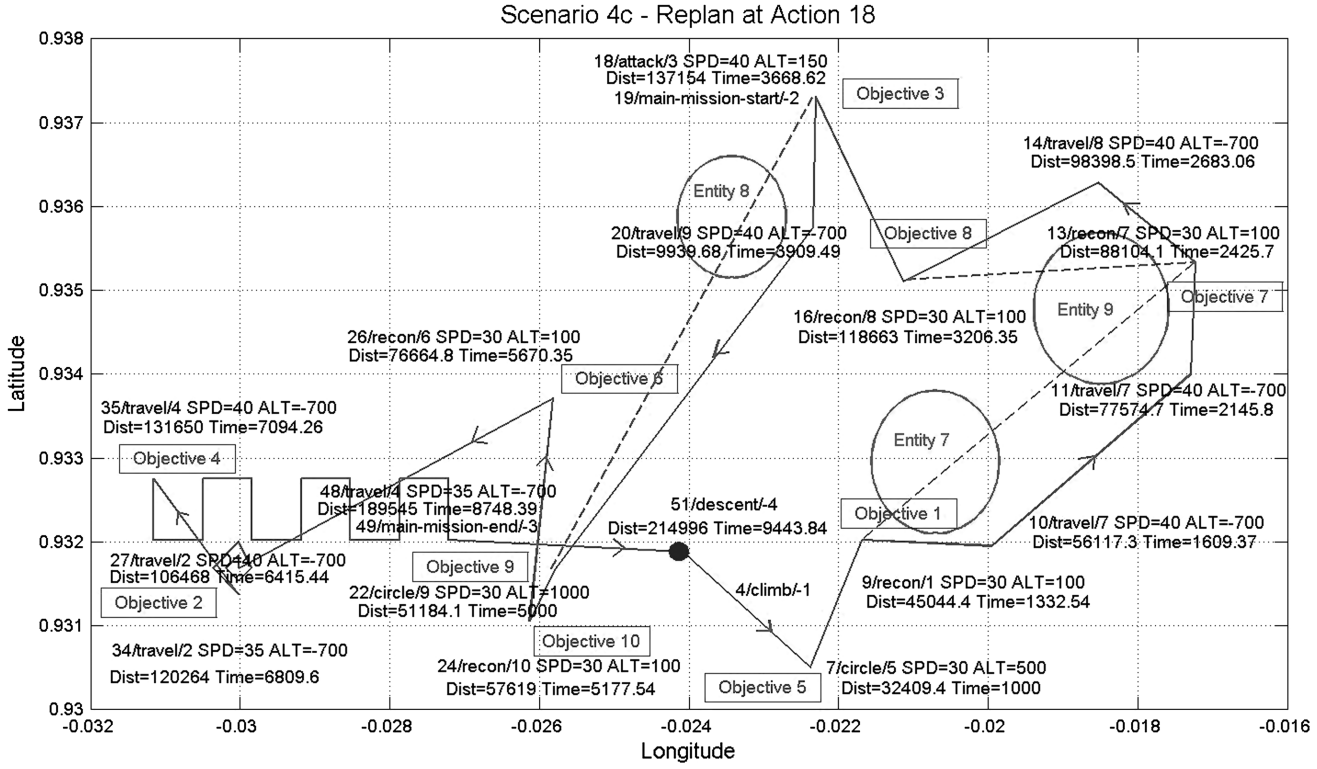



**Fig. 17 Plot of scenario 4c flight plan, replanning event at action 18 (SPD: speed, m/s; ALT: altitute, m).**

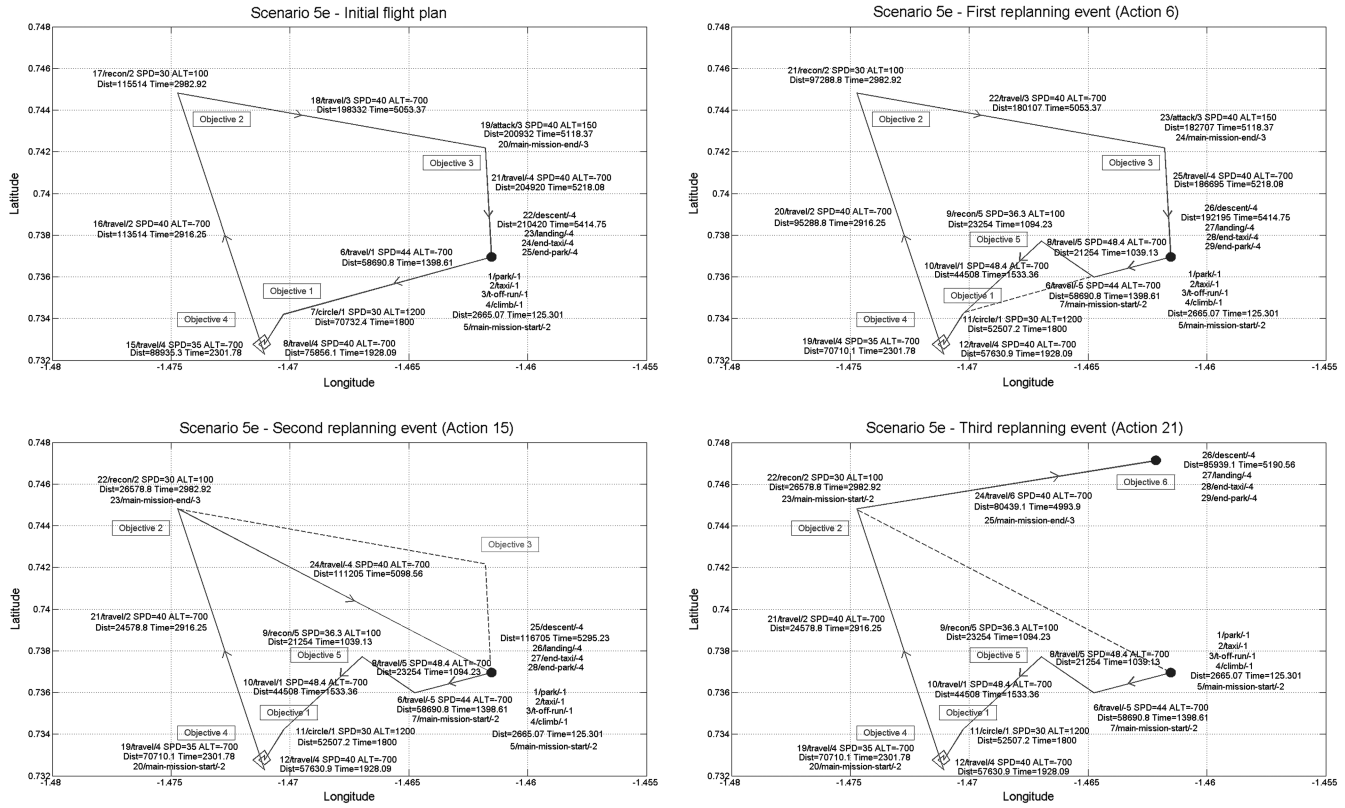



**Fig. 18 Evolution of the flight plan within scenario 5e (SPD: speed, m/s; ALT: altitute, m).**

MATLAB R2006b environment with a 100 ms sample time; a dual-core laptop computer can run an entire simulation (including the computationally heavy UAV model and a visualization feature based on Microsoft Flight Simulator) faster than real time.

The target platform for SAMMS is a single PC/104 board running either QNX or Tornado; using the Real-Time Workshop feature of MATLAB, the intention is to automatically generate executable code from the Simulink model. This code would include the three Soar agents and the autopilot function, so as to avoid the need for an external autopilot. Depending on the avionics suite, an additional PC/104 I/O board could be

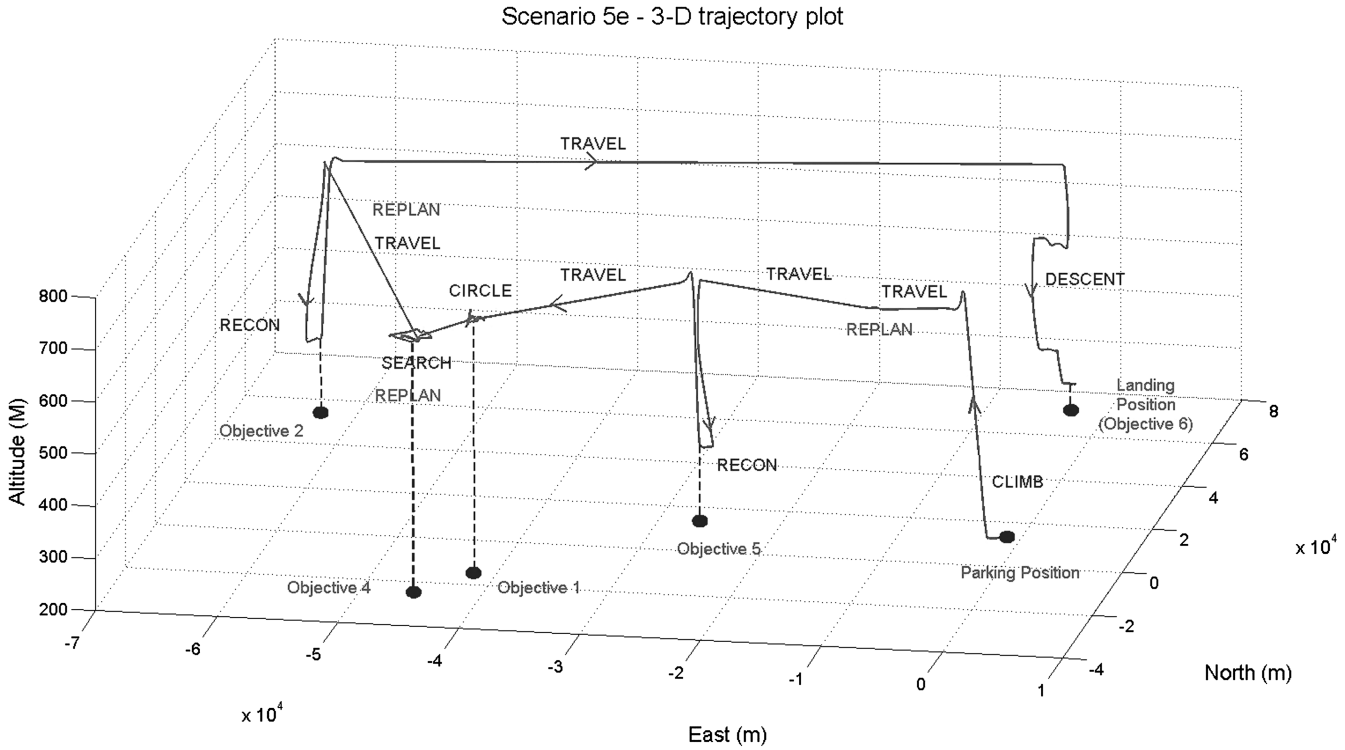



**Fig. 19 Scenario 5e trajectory plot.**

needed, but no additional components should be required. Using COTS hardware, such a configuration would weigh in the 200–500 g range and cost about $500–1000. This qualifies it for use within small UAVs with a maximum weight of 5–10 kg and costs in the $10,000 range.

At present, the fact that such a system possesses sufficient computational resources to execute the SAMMS software cannot be guaranteed. However, past experiences such as in [24], suggest that compilation of Simulink models allows for a dramatic increase of execution speed. The addition of Soar agents brings uncertainty; however, these are in fact executed as Simulink S functions (which do not pose an issue), and related literature [6,8,14] does not evidence problems regarding the computational resources required by the Soar architecture. While not proven, the authors are confident that a single PC/104 board will be sufficient to execute the SAMMS software respecting real-time requirements.

## V. Conclusions

In this paper, a novel software system for autonomous mission management and execution was presented. The system is based on a combination of three Soar-based intelligent agents, supported by additional software components implementing traditional control algorithms (autopilots and others). The whole system was integrated using the MATLAB/Simulink package, which also provided the simulation environment used to test it.

Simulation results proved the feasibility of the approach; the system demonstrated its capability to control a simulated model of the Pioneer unmanned aerial vehicle (UAV), deriving appropriate flight plans in all the test scenarios and then executing them smoothly.

A possible implementation strategy was also discussed, highlighting the fact that, once the software compiled in executable code using the Real-Time Workshop package, the Soar-Based Autonomous Mission Management System (SAMMS) computational requirements are estimated not to exceed the computational resources available on a PC/104 board running a real-time operating system such as QNX or VxWorks Tornado. This qualifies the architecture for use within small low-cost UAVs; it is to be noted that, in this form, the architecture will not be applicable to micro-UAVs (less than 2 kg total weight). Real-world application of SAMMS will require recalculation of the low-level control algorithm parameters, but otherwise, the high-level agents are platform independent and only need a small set of performance parameters in order to work. A thorough study regarding the certification possibilities for SAMMS will also be needed, especially since its capabilities make its suitable for civilian applications such as environmental monitoring. Within such applications, the SAMMS architecture could provide the basis for a small low-cost UAV to be used by untrained personnel, thus opening up new market sectors.

Apart from actual implementation, several improvements more specifically related to the Soar agents can be planned. First is the issue of improving search objectives; new search patterns can be implemented, as well as search-area orientation and the possibility to interrupt and reprise searches. A second issue is represented by the nearest-neighbor algorithm used during the plan-sequencing phase; literature on the subject is abundant, and a better heuristic could be used. Finally, other functionality could be added in order to further improve the flight plans being generated; for example, the interaction between the fuel-check and priority-check algorithms could be improved, and the execution agent might be modified so as to execute smoother flight trajectories, especially when altitude changes are involved (a slow descent can lead to significant fuel savings).

## References


[1] Schaefer, P.,Colgren, R. D., Abbott, R. J., Park, H., Fijany, A., Fisher, F., James, M. L., Chien, S., Mackey, R., Zak, M., Johnson, T. L., and Bush, S. F., "Reliable Autonomous Control Technologies (ReACT) for Uninhabited Aerial Vehicles," *2001 IEEE Aerospace Conference*, Vol. 2, Big Sky, MT, 2001, pp. 677–684.

[2] DeGarmo, M., and Nelson, G., "Prospective Unmanned Aerial Vehicle Operations in the Future National Airspace System," *4th Aviation Technology, Integration and Operations Conference*, ATIO, Chicago, IL, 2004.

[3] Miller, J. A, Minear, P. D., Niessner, A. F., Jr., DeLullo, A. M., Geiger, B. R., Long, L. N., and Horn, J. F., "Intelligent Unmanned Air Vehicle Flight Systems," InfoTech@Aerospace Conference 2005, AIAA Paper 2005-7081, 2005.

[4] Clough, B., "Metrics, Schmetrics! How the Heck Do You Determine A UAVs Autonomy Anyway?," *Performance Metrics for Intelligent Systems Workshop*, PerMIS, Gaithersburg, MD, 2002.

[5] Cummings, M., Bruni, S., Mercier, S., and Mitchell, P., "Automation Architecture for Single-Operator Multi-UAV Command and Control," *International Command and Control (C2) Journal*, Vol. 1, No. 2, 2007, pp. 1–24.

[6] Li, S.-M., Boskovic, J. D., Seereeram, S., Prasanth, R., Amin, J., Mehra, R. K., Beard, R. W., and McLain, T. W., "Autonomous Hierarchical Control of Multiple Unmanned Combat Air Vehicles," *Proceedings of the American Control Conference*, Vol. 1, Anchorage, AK, May 2002, pp. 274–279.

[7] Veres, S. M., Molnar, L., Lincoln, N. K., and Morice, C. P., "Autonomous Vehicle Control Systems: A Review of Decision Making," *Proceedings of the Institution of Mechanical Engineers. Part I, Journal of Systems and Control Engineering*, Vol. 225, 2011, pp. 155–195.

[8] Jones, R., Laird, J., Nielsen, R., Coulter, K., Kenny, R., and Koss, F., "Automated Intelligent Pilots for Combat Flight Simulation," *AI Magazine*, Vol. 20, No. 1, 1999, pp. 27–41.

[9] Karim, S., Heinze, C., and Dunn, S., "Agent-Based Mission Management for a UAV," *International Conference on Intelligent Sensors, Sensor Networks and Information Processing*, ISSNIP 2004, pp. 481–486, Melbourne, Australia, IEEE 0-7803-8894.

[10] Lucas, A., Heinze, C., Karim, S., et al., "Development and Flight Testing of an Intelligent, Autonomous UAV Capability," AIAA Unmanned Unlimited 2004, AIAA Paper 2004-6574, Sept. 2004.

[11] Karim, S., and Heinze, C., "Experiences with the Design and Implementation of an Agent-Based Autonomous UAV Controller," *4th International Conference on Autonomous Agents and Multi-Agent Systems*, AAMAS'05, Utrecht, The Netherlands, July 2005, pp. 19–26.

[12] Wooldridge, M., "Intelligent Agents," *Multi-Agent Systems: A Modern Approach to Distributed Artificial Intelligence*, edited by G. Weiss, MIT Press, Cambridge, MA, April 1999, pp. 3–44.

[13] Jennings, N., and Wooldridge, M., "Applications of Intelligent Agents," *Agent Technology: Foundation, Applications and Markets*, edited by N. Jennings, and M. Wooldridge, Springer-Verlag, New York, March 1998, pp. 3–27.

[14] Long, L., Hanford, S., Janrathitikarn, O., Sinsley, G., and Miller, J., "A Review of Intelligent Systems Software for Autonomous Vehicles," *Proceedings of the 2007 IEEE Symposium on Computational Intelligence in Security and Defense Applications*, IEEE, 2007, pp. 69–76.

[15] Laird, J., Newell, A., and Rosenbloom, P., "Soar: An Architecture for General Intelligence," *Artificial Intelligence*, Vol. 33, No. 1, 1987, pp. 1–64. doi:10.1016/0004-3702(87)90050-6

[16] Wray, R. E., and Jones, R. M., "An Introduction to Soar as an Agent Architecture," *Cognition and Multi-Agent Interaction: From Cognitive Modeling to Social Simulation*, edited by R. Sun, Cambridge Univ. Press, Cambridge, England, U.K., 2005, pp. 53–78.

[17] Gunetti, P., Mills, A., and Thompson, H., "A Distributed Intelligent Agent Architecture for Gas-Turbine Engine Health Management," *46th AIAA Aerospace Sciences Meeting and Exhibit*, AIAA Paper 2008-0883, Jan. 2008.

[18] Gunetti, P., and Thompson, H., "A Soar-Based Planning Agent for Gas-Turbine Engine Control and Health Management," *17th IFAC World Congress*, Seoul, Korea, July 2008, pp. 2200–2205.

[19] Gunetti, P., and Thompson, H., "Development and Evaluation of a Multi-Agent System for Gas-Turbine Engine Health Management," *Automatic Control in Aerospace* [online journal], Vol. 3, No. 1, May 2010.

[20] Gunetti, P., Dodd, T., and Thompson, H., "A Software Architecture for Autonomous Mission Management and Control," AIAA InfoTech@Aerospace Conference, AIAA, Paper 2010-3305, 2010.

[21] Gunetti, P., Thompson, H., and Dodd, T., "Autonomous Mission Management for UAVs Using Soar Intelligent Agents," *International Journal of Systems Science*.
doi:10.1080/00207721.2011.626902
[22] Rosenkrantz, D., Stearns, R., and Lewis, P., "An Analysis of Several Heuristics for the Travelling Salesman Problem," *Fundamental Problems in Computing*, Springer, New York, 2009, pp. 45–71.
[23] Campa, G., Airlib, Software Package, MathWorks, Natick, MA, March 2004, http://www.mathworks.com/matlabcentral/fileexchange/3019-airlib [retrieved 2012].
[24] Battipede, M., Gili, P., Lando, M., and Gunetti, P., "Flight Control System Rapid Prototyping for the Remotely-Controlled Elettra-Twin-Flyer Airship," IAA Modelling Simulation and Technologies Conference, AIAA Paper 2006-6624, 2006.

E. Atkins
*Associate Editor*

## Queries

**IMPORTANT: PLEASE READ CAREFULLY.**

When production of AIAA journal papers begins, the official approved PDF is considered the authoritative manuscript. Authors are asked to submit source files that match the PDF exactly, to ensure that the final published article is the version that was reviewed and accepted by the associate editor. Once a paper has been accepted, any substantial corrections or changes must be approved by the associate editor before they can be incorporated.

If you and the EIC settled on some final changes to your manuscript after it was accepted, it is possible that your page proofs do not reflect these final changes. If that is the case, please submit these changes as itemized corrections to the proofs.

If final changes were made to the figures, please check the figures appearing in the proofs carefully. While it is usual procedure to use the figures that exist in the source file, if discrepancies are found between figures (manuscript source file vs the approved PDF), the figures from the PDF are inserted in the page proofs, again deferring to the PDF as the authoritative manuscript. If you find that agreed-upon final changes to your figures are not appearing in your page proofs, please let us know immediately.

**Q1.** AU: Please review the revised proof carefully to ensure your corrections have been inserted properly and to your satisfaction.
**Q2.** AU: As per style, only proper names in the expressions, hence the term “Soar-based autonomous mission management system” and also the agent names have been retained as in the proof.
**Q3.** AU: As per style, italics are not allowed in the text hence first instance of state names have been surrounded with quotations to highlight them.
**Q4.** AU: Please check and confirm the edit made in the captions of Figs. 4, 7, 8, 10-13, and 15-18.
**Q5.** AU: Please check and confirm the introduction of table footnote in Table 5.