Development and Evaluation of a Multi-Agent System for Gas-Turbine Engine Health Management

Paolo Gunetti, Haydn Thompson

Department of Automatic Control and Systems Engineering
The University of Sheffield
Mappin Street, Sheffield, UK, S1 3JD
Email: p.gunetti@shef.ac.uk, h.thompson@shef.ac.uk

Abstract

The development of Unmanned Aerial Vehicles (UAVs) is continuously pushing the boundaries for more autonomous systems. In order to make realistic the goal of autonomous civil UAVs flying within civil airspace, technology must be developed so that UAVs and all their critical subsystems can be certified for autonomous flight. In this paper, we address specifically the Propulsion subsystem and the problems that arise when engines are affected by failures. In a Pilot-Out-Of-the-Loop situation, the engine controller must be able to autonomously detect, isolate and eventually mitigate failures, which involves making decisions that are delegated to the pilot in current systems. We propose a Propulsion Health Management System for a general twin-engine UAV, based on Intelligent Agent technology. The system fuses different software techniques to achieve the Fault Evaluation and Fault Mitigation functions, which would normally be performed by the pilot after Fault Detection and Fault Isolation. The proposed system is described in detail and then the results of simulation tests are presented.

Keywords
intelligent agents, multi-agent systems, autonomous vehicles, UAVs, gas-turbine engine, GTE health management

1. Introduction: Autonomous UAVs and subsystems

Unmanned Aerial Vehicles (UAVs) started being developed even before World War II, although their main function at the time was as target drones (Goebel, 2008). 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 civil use. The problems included but were not limited to the following list:

- ensuring a radio link with sufficient bandwidth 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:

- operating costs can be greatly reduced, especially for aircraft with small payload
- UAVs are generally more expendable, which is an asset for military application
- flight performance can be greatly increased, for example achieving longer endurance or making possible high-G manoeuvres that would render a human pilot unconscious.

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 in the last decade, there has been a great increase of interest in the development and use of UAVs. They are now very common in military operations, especially reconnaissance and intelligence, but also for attack missions. UAVs have also found use in civil applications, mainly related to surveillance and environmental protection.

Military applications usually do not involve flight within civil airspace, which allowed for earlier application, since in such cases UAVs are subject to military flight rules, which can be generally seen as more permissive than civil flight rules. For instance, this had led to greater levels of autonomy for military

Figure 1. Predator UAV

UAVs, while civil regulations are far from allowing UAV flight without strict supervision. Current military UAVs (like the Predator in Figure 1) are capable of carrying out an entire pre-planned mission, and generally only need supervision in order to address situational changes that ask for a modification to the flight plan. Civil applications are instead generally restrained by the need to avoid commercial airspace for safety reasons (UAV Task Force, 2004).

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 full pilot training and control of multiple UAVs by a single user. Also, civil applications would certainly receive great help by the opening of civil airspace to UAV traffic. The challenges presented by such objectives are great, since 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 follow flight rules.

It is important to understand that 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 delegated to the pilot (for example, the 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.

In the UK, the ASTRAEA (Autonomous Systems Technology Related Airborne Evaluation and Assessment) programme was launched with the intention to develop the technology that would make autonomous UAV flight within civil airspace possible. ASTRAEA is a £32 million civil programme led by an industrial consortium incorporating Agent Oriented Software, BAE Systems, EADS, Flight Refuelling, QinetiQ, Rolls-Royce and Thales UK, working with leading academics and supported by investment from the DTI, Welsh Assembly Government, Scottish Enterprise and regional development agencies covering the North West, South East and South West of England.

Within the ASTRAEA programme, our group has focused on the development of technologies for the Propulsion subsystem. We particularly considered the case of Gas-turbine Engines, due to familiarity with related problems and the complexity of the problem space. When introducing autonomy, the main challenge related to this subsystem is Health Management. While automatic control systems have been developed for decades and are fully capable of ensuring smooth engine operation in normal situations, long-term mitigation actions to be taken after the occurrence of faults are decided by the pilot in current systems. Using Failure Modes Effect and Criticality Analysis (FMECA) data provided by Rolls-Royce, our industrial partner, we first developed a Prognosis framework that allows to quantitatively estimate the effect of a fault. Then, we used Intelligent Agent technology to build a Multi-Agent System capable of devising fault mitigation plans using the Prognosis information. The entire system (which will be referred to as the “Planner” from now on), comprising the Prognosis framework and the fault mitigation Planning Agents, was then tested in a simulated environment in order to demonstrate its ability to properly assess faults and propose effective mitigation plans.

At present, a consistent amount of research is dedicated to new Fault Detection and Isolation (FDI techniques for aircraft engine (Kobayashi and Simon, 2003, and Austin et al., 2004). This research focuses heavily on the Detection, Isolation and Prognosis of faults, and the outputs are generally represented by warnings to the pilot or to ground maintenance. This paper aims instead at introducing a system that can use the output of a FDI system in order to determine the course of action that an autonomous UAV should follow after a fault is detected. This task is usually left to the judgement of the pilot, but similar functionality is to be included in a truly autonomous UAV. The paper presents a Fault-Prognosis and Fault Mitigation system, based on the integration of conventional technologies (for the Fault Prognosis part) and unconventional ones (Cognitive Intelligent Agents are used for Fault Mitigation). The system is tested in a simulated environment and simulation results are included.

The paper is organized in three main sections; following this introduction (section 1), in section 2 the requirements for the Planner will be introduced; in section 3, the details of the system implementation will be covered; in section 4, the simulation environment will be presented together with simulation results; finally, a conclusions and future work section is placed in the end.

2. Propulsion subsystem: what is needed when the Pilot is out-of-the-loop

The Propulsion subsystem of an aircraft is the system which provides the thrust for forward movement and generates the electricity needed by other on-board subsystems and actuators. There are two main types of propulsion systems: alternative engines (usually comprising an Otto cycle engine and an alternator) and jet engines (with the typical layout of air inlet, compressor, combustion chamber, turbine and exhaust nozzle). For this study, we focused exclusively on jet engines, since these are the main product of our industrial partner and are more likely to be used within long endurance UAVs.

At the current state of technology, jet engines are equipped with Full Authority Digital Engine Controllers (FADECs), which are basically computer systems that control and monitor the state of the engine using various sensors and actuators and digital signal processing (DSP). FADECs are usually capable of detecting unusual or dangerous running conditions of an engine (such as a surge or an airflow stall) and perform immediate reversionary action to counter these conditions. These actions are fully automated, since they can happen in the timescale of tens of milliseconds, so that the pilot would not be able to react in time. However, when such an event occurs, the pilot gets a warning through his instrumentation.

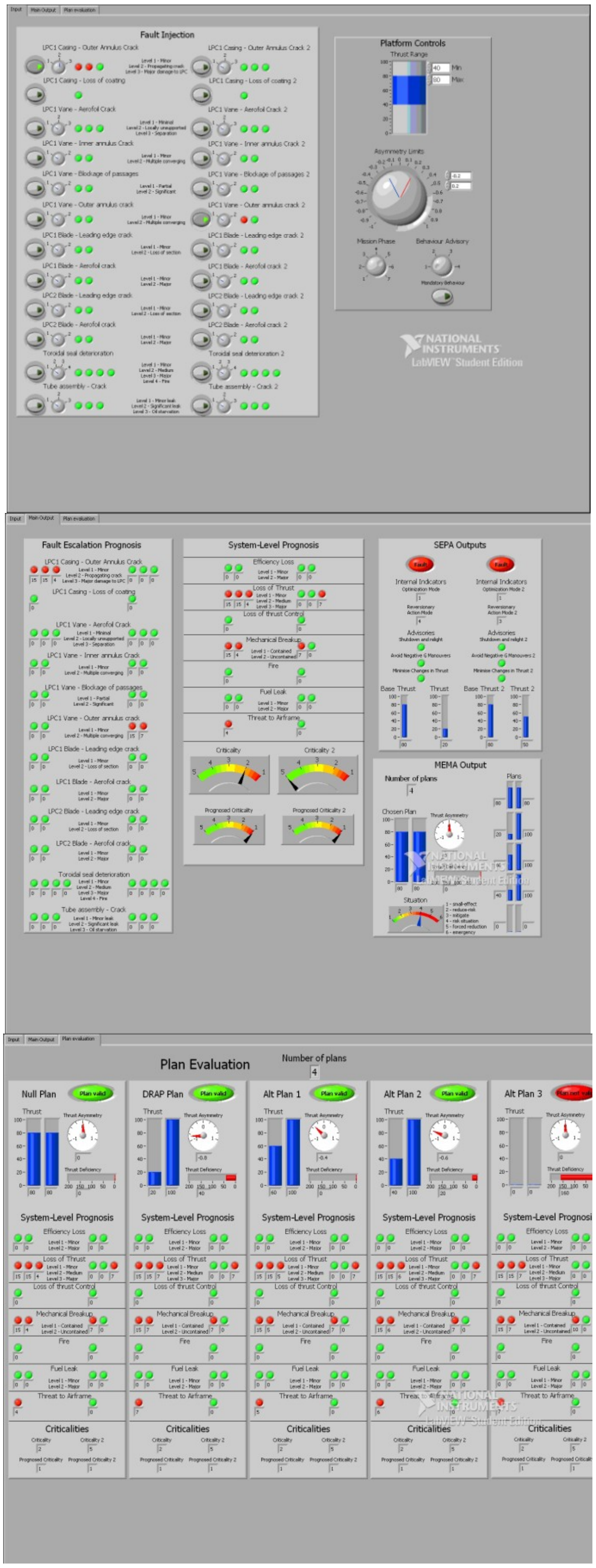

Figure 2. Screens of Visual Interface

Usually, a FADEC can detect several other types of anomalies that do not represent an immediate risk to the engine but are rather general hints of deteriorating engine performance. In such cases, the pilot would receive a warning through the instrumentation and the event would be recorded for ground inspection. A pilot could also detect unusual behaviour of the engine. In any case, the pilot will then use his experience and knowledge in operating the specific aircraft and engine configuration to determine the course of action.

When the pilot is taken out-of-the-loop, it is particularly this type of decision-making which is mostly affected. In fact, there is a great level of uncertainty related to such corrective action, and the judgement of a pilot is usually driven by a mixture of expert knowledge and experience that is very difficult to recreate in a computer system. While the mitigating actions for specific faults such as a surge are straight-forward decisions (fault is detected $\rightarrow$ mitigating action is taken), decisions about the course of action to take in the case of uncertain faults and in general non-severe propulsion subsystem anomalies are influenced by many factors and in general will fall under the realm of multi-objective optimization (since usually decisions involve a trade-off between various aspects of the mission). In fact, such decisions are influenced by not only the situation of the propulsion subsystem, but also the general condition of the aircraft and knowledge about the current state of the mission.

It is understood that a UAV which can fly in civil airspace must be able to make correct decisions on the occurrence of a fault, so as to ensure safety of other aircraft and on-ground structures at all times. Our aim is therefore to develop technology that will allow a fully autonomous UAV to make such correct decisions involving the propulsion subsystem. The resulting system has to be fully interfaced with the UAV supervisory authority and has four main functions: Fault Detection, which is the ability to detect anomalies in engine operation using sensor data; Fault Isolation, which is the ability to fuse information from Fault Detection to derive a Diagnosis of the current engine situation; Fault Evaluation, which is the ability to prognose the escalation of fault to higher levels of criticality and evaluate the effect of a fault in terms of airframe operations; Fault Mitigation, which is the ability to counteract a fault by performing various types of reversionary action, such as placing limitations on engine usage or demanding an engine shutdown and relight.

For the purpose of this project, we focus exclusively on Fault Evaluation and Fault Mitigation. We are going to consider Fault Detection and Fault Isolation to have already been achieved, and the output of the Fault Isolation function will basically constitute the main input for the functions which we will develop. Fault Detection and Fault Isolation are considered important for all types of engine-aircraft configuration, and great effort is already spent within industry in improving systems that perform those functions. The system we propose is based on a FMECA database provided by Rolls-Royce; we used a representative subset of the database, in order to focus on the development of the technology rather than the implementation within a real system (proof-of-concept). The Planner system is modelled

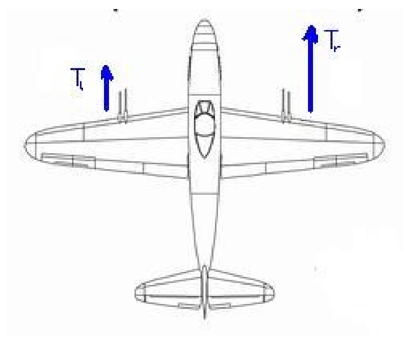

Figure 3. Thrust asymmetry scheme

using the Simulink software tool as a basis; other technologies (Soar Intelligent Agents) are integrated within Simulink, as is a visual interface that allows to fully control the execution of simulations.

Although it can be adapted to other cases, the system is configured to handle the case of a twin-engine UAV configuration. An input interface was developed, allowing injection of faults at different severity stages, along with thrust demands from the UAV supervisory authority. Inputs can come in two different formats: as manual input or as recorded input. Manual input is mainly used for demonstration purposes; a dedicated visual interface (Figure 2) was developed using NI LabView software, and is fully integrated with the Planner system model. The visual interface also shows the output in an easily understandable format, without the use of graphs. Recorded input takes the form of pre-prepared files that take the system through a series of different input conditions, and are mainly used during simulations. In the case of recorded input, data is also recorded in data files for further analysis. The FMECA database subset models a total of 12 realistic faults; many of these faults can escalate through different severity stages, for a total of 28 possible fault input conditions for each engine. It is assumed that a single engine will only ever be in one of these states – in case of multiple faults, it is assumed that only the most critical will be addressed by the system. However, it is possible to inject separate faults into the two engines, leading to a total number of fault input combinations of 841 (including no-fault states).

Within this project, we always assume the presence of a UAV supervisory authority, which is supposed to provide additional input for the Planner, represented by the total engine thrust demand and thrust asymmetry limits. Thrust asymmetry is calculated as $(T_l-T_r)/(T_l+T_r)$, where T_l and T_r are the thrust demands in the left and right engines respectively (see Figure 3); the supervisory authority inputs an allowed range for asymmetry, for example -0.5/0.5.

We make the assumption that the supervisory authority will possess the situational awareness needed to make the final decision regarding the course of action to take. Based on this assumption, the Planner system in practice generates a list of different reversionary action plans, ranging from the "optimal" plan (the best plan from the point of view of the engine subsystem) to the "do-nothing" plan (which basically ignores the fault). The number of generated plans is dependent on fault criticality and additional plans between the two extremes present "middle" options that are a trade-off. The Fault Evaluation algorithms are used to give an estimate of how effective a plan will be in mitigating the fault. The plans are then sent to the supervisory authority, together with the prognosis results from the Fault Evaluation algorithms. The authority can then decide which plan to apply, combining the data sent by the Planner system with its situational awareness.

Overall, the Planner system we propose (which will be thoroughly described in Section 3) takes Fault Isolation data as input and then develops the Fault Evaluation and Fault Mitigation functions, which are in current Propulsion systems completely delegated to the pilot (whereas Fault Detection and Isolation are already automated, at least partially, although they are extremely complex tasks in their own). The

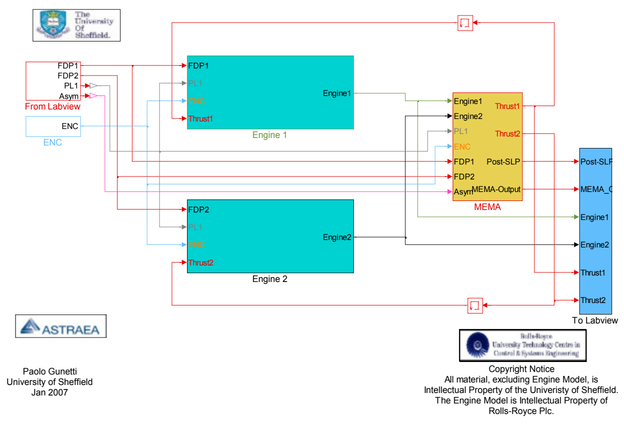

Figure 4. Top-level Planner architecture

next section will describe the algorithms and software technology that we employed to achieve this functionality in the Planner system.

3. Propulsion Health Management System: Prognosis Framework, Agents

"Propulsion Health Management System" is a more complete definition of the Planner system that has been introduced in the first two sections. In this section, the system will be described in detail.

The system was designed using Simulink as the main development tool. The main architecture is modelled in Simulink, and non-Simulink modelling tools are implemented within the architecture as S-Functions. The architecture relative to a single-engine configuration is described in (Gunetti et al, 2008), although it has been complemented by new functions since then.

Within the Planner system, three major subsystems can be identified: the Prognosis Framework, the Single-Engine Planner Agent (SEPA) and the Multi-Engine Manager Agent (MEMA). Figure 4 represents the top-level architecture of the Planner; the two cyan blocks represent the two engines, while the yellow block represents the MEMA. As can be seen in Figure 5, each of the engine blocks contains the Prognosis Framework (yellow and orange blocks) and SEPA (blue block) relative to that engine.

While the Prognosis Framework is based on standard control systems modelling techniques, the SEPA and MEMA components are based on Intelligent Agents. Intelligent agents are a new paradigm in the development of software applications (Jennings and Wooldridge, 1998) and are designed to address the need for flexible and autonomous computer systems. This technology is still at an early stage; it has been exploited thoroughly in certain areas of application

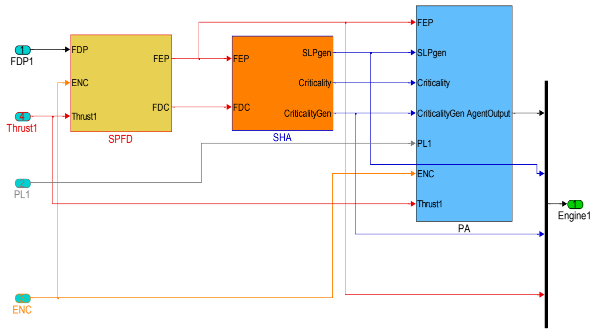

Figure 5. Engine subsystem

(like internet search engines), but its use in other areas of software engineering is restricted at best. In fact, even agreement on the definition of IA is not universally accepted among computer scientists. A popular definition, which we take as our own point of view, is that "an Agent is a computer system situated in some environment, and that is capable of *autonomous* action in this environment in order to meet its design objectives" (Wooldridge, 1999). Furthermore, we can say that an *Intelligent* Agent is one that is capable of *flexible* autonomous action, where flexible implies *reactivity* (ability to understand the environment and react to its changes), *proactiveness* (goal-oriented behaviour) and *social ability* (ability to interact with other agents).

The use of Intelligent Agents was one of the key elements to this project, because of a growing interest in the commercial application of this technology. In particular, results form the project will help in determining whether Intelligent Agents are ready for commercial application in the aerospace market, which is conservative in nature but also naturally open to new technologies (Dorfman, 1999).

We now proceed to describe each of the subsystems in detail.

3.1 Prognosis Framework

The Prognosis Framework performs the Fault Evaluation function described in section 2, which means that it must be able to complement Fault Diagnosis with two types of data: a Prognosis of how the fault can be expected to escalate based on engine usage and FMECA data (Fault Escalation Prognosis), and a Prognosis of how the airframe is affected by the fault (System-Level Prognosis).

Fault Escalation Prognosis is basically a direct implementation of the data extrapolated from the FMECA database. It uses a three-dimensional look-up table to prognose how a fault is expected to escalate in time. Each modelled fault is classified in a varying number of severity stages and the Fault Escalation Prognosis block estimates the

Table 1 - List of faults and severity stages

No	Fault type	Severity Stages	Code
1	LPC1 Casing - Outer annulus crack	Minor	1
		Medium	2
		Major	3
2	LPC1 Casing - Loss of coating		4
3	LPC1 Vane - Aerofoil crack	Minimal	5
		Locally unsopported	6
		Separation	7
4	LPC1 Vane - Inner annulus crack	Minor	8
		Multiple converging	9
5	LPC1 Vane - Blockage of passages	Partial	10
		Significant	11
6	LPC1 Vane - Outer annulus crack	Minor	12
		Multiple converging	13
7	LPC1 Blade - Leading edge crack	Minor	14
		Loss of section	15
8	LPC1 Blade - Aerofoil crack	Minor	16
		Major	17
9	LPC2 Blade - Leading edge crack	Minor	18
		Loss of section	19
10	LPC2 Blade - Aerofoil crack	Minor	20
		Major	21
11	Toroidal seal deterioration	Minor	22
		Medium	23
		Major	24
		Fire	25
12	Tube Assembly - Crack	Minor leak	26
		Significant leak	27
		Oil starvation	28

timescale after which the fault can be expected to escalate to a higher severity stage.

The key point is that the Escalation time is heavily influenced by engine usage, so the time to escalation generally increases as thrust demand reduces. This is the reason for using a 3D look-up table, as it necessary to provide escalation estimates for different running conditions. The output of the look-up table is the "time to escalation" estimate for the current stage of the fault towards more critical stages.

Both the fault severity stages and the timescale values are discretized. Table 1 lists the currently modelled faults and their stages, while Table 2 indicates the timescale discretization we used. Obviously, Fault Escalation Prognosis is useless for faults that have reached their final stage, but the real value of the entire Planner system is evident before a fault reaches the final stage. In fact, in this case the Planner will generate reversionary action plans that will maximise the time to escalation for a fault, while when the final stage is reached the plan is usually a very straight-forward action.

Table 2 - Timescale codes

Code	Time to failure
1	30 sec
2	1 min
3	5 min
4	10 min
5	30 min
6	1 h
7	2 h
8	5 h
9	10 h
10	20 h
11	50 h
12	100 h
13	150 h
15	Fault has happened
0	Fault not detected

System-Level Prognosis is instead aimed at extracting useful information from the Fault Diagnosis input and the Fault Escalation Prognosis. From a theoretical point of view, it provides an answer to the question: "How will this engine fault affect the operation of the entire UAV?". The concept behind this is the fact that a UAV supervisory authority is not concerned about the actual nature of an engine fault, but only about its effects on UAV capabilities. As an example, the supervisory authority is not interested in knowing that the outer annulus in the low-pressure compressor casing is cracked, but it needs to know that this will cause a reduction in actual thrust that is dependant on how severe the crack is. This is the rationale behind the System-Level Prognosis function.

Table 3 lists the 7 different types of System-Level Effects (SLEs) that have been identified for the purpose of this project. Some of these can evolve through subsequent stages, which are discretized as is the case with fault severity stages, leading to a total of 12 SLEs. The SLEs are directly derived from the FMECA database. The system uses a simple two-dimensional look-up table to calculate the SLEs related to a fault, since the relationship between fault and SLE is straight-forward.

Table 3 - System Level Effects (SLEs)

SLE	SLE description	Stages
1	Efficiency loss	Minor
		Major
2	Loss of thrust	Minor
		Medium
		Major
3	Loss of thrust control	
4	Mechanical break-up	Contained
		Uncontained
5	Engine fire	
6	Fuel leak	Minor
		Major
7	Threat to airframe	

The System-Level Prognosis function also performs another calculation: it assigns a Criticality level to the current detected fault and also to the relative escalation stages that are prognosed. These Criticality levels provide an immediate way of classifying the severity of a fault, and are discretized as per common practice within FMECA databases. Table 4 lists Criticality levels and their definitions.

Overall, the output of the Prognosis Framework consists of three vectors, reporting respectively Fault Escalation Prognosis, System-Level Effects and Criticality levels. The Framework operates on single engines, so that three of these vectors will be generated for each engine, just as Fault Diagnosis input is separate for each engine.

3.2 Single-Engine Planner Agent (SEPA)

The Single-Engine Agent Planner or SEPA is the agent entity which performs the Fault Mitigation function related to a single engine. Its task is to develop reversionary action plans that address a fault from the point of view of a single engine. Once a fault is detected and evaluated through the Prognosis Framework, the SEPA proposes two courses of action: a "do-nothing" plan and an "optimal" plan, which is the best action course for the engine regardless of what is the situation in the rest of the Propulsion system or the entire UAV. It is then the task of MEMA (subsequently described) to contextualize the plans and derive alternative ones.

Table 4 - Criticality Levels and definitions

Level	Definition	Description
5	No fault	No fault is detected
4	Negligible	Fault only has negligible effects
3	Minor	Fault affects performance but does not compromise mission
2	Severe	Fault can compromise the completion of the mission
1	Catastrophic	Fault can lead to loss of engine and eventually loss of aircraft

The SEPA has been thoroughly described in a previous paper. Therefore, in this section we will only introduce its main characteristics; for more details, please refer to (Gunetti and Thompson, 2008).

The SEPA is modelled using the Soar Intelligent Agent tool. Soar is the computational implementation of a cognitive architecture which has been developed at the University of Michigan since the late 1980s (Soar Technology Inc., 2002). It provides a robust architecture for building complex human behaviour models and intelligent systems that use large amounts of knowledge. At a high level of abstraction, it uses a standard information processing model including a processor, memory store, and peripheral components for interaction with the outside world. At a low level of abstraction, Soar uses a Perceive-Decide-Act cycle to sample the current state of the world, make knowledge-rich decisions in the service of explicit goals, and perform goal-directed actions to change the world in intelligent ways. The distinguishing features of Soar are: parallel and associative memory, belief maintenance, preference-based deliberation, automatic sub-goaling, goal decomposition and adaptation via generalization of 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.

Table 5 - SEPA states

No	State	Description
0	no-fault	No fault is detected
1	no-action	Fault is negligible and does not require reversionary action
2	increase-power	Fault causes a minor reduction in effective thrust that can be compensated
3	reduce-prognosis	Fault is minor but can escalate to critical, so usage limitation is requested
4	reduce-power	Fault is severe and usage limitation is requested
5	shutdown-engine	Fault should be addressed with an engine shutdown

A dedicated interface was developed in order to implement Soar agents within the Simulink model of the Planner system. This is achieved by executing the Soar kernel as an S-Function within the model. Multiple agents can run at the same time, and in fact two instantiations of the SEPA are executed, each connected to a single engine.

The SEPA core rules implement a decision making scheme that uses data generated by the Prognosis Framework to propose reversionary action plans. The SEPA enters different states depending on the current input and goes through a stepped decisional tree in order to derive a full action plan. The plan consists of a proposed thrust value for

Table 6 - MEMA decisional matrix

SEPA	0/1	2	3	4	5
0/1	1	1	2	3	3
2	1	1	2	4	4
3	2	2	5	4	4
4	3	4	4	5	6
5	3	4	4	6	6

No	State	Description
1	small-effect	No reversionary action is taken, apart from compensation of missing thrust due to minor faults
2	reduce-risk	Thrust is decreased on faulty engine in order to postpone further escalation of fault, increased on other engine
3	mitigate	Serious fault on one engine is addressed by placing usage limitation and compensating with other engine
4	risk-situation	Both engines are faulty, thrust is decreased where fault is more serious and increased on other engine, shortening the escalation time
5	forced-reduction	Both engines are faulty with similar criticality, a reduction of total thrust demand is requested
6	emergency	Both engines are in critical condition and need a shutdown (very unlikely case)

the engine (usually limited to some degree if a fault is detected), a “do-nothing” thrust value and three binary indicators that complement the plan by indicating other possible reversionary action.

Table 5 lists the possible states that the SEPA can enter when generating the most important part of the plan, which is the proposed engine thrust value.

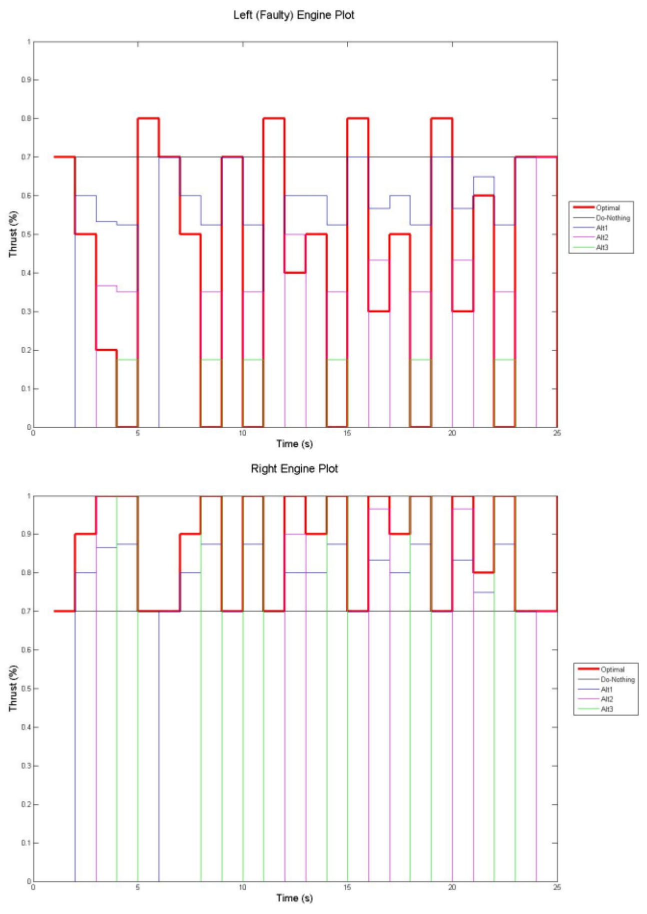

Figure 6. Full fault input combination tests

3.3 Multi-Engine Manager Agent (MEMA)

Once a reversionary action plan is generated for a single engine by the SEPA, this plan needs to be put in the context of the entire propulsion system. This involves considering the situation of the other engine at first, and then the demands that are coming from the UAV supervisory authority.

The MEMA generates at first a “do-nothing” plan and an “optimal” plan, which are basically a symmetric thrust distribution and an asymmetrical one respectively. The asymmetric thrust distribution usually found in the optimal plan is a consequence of the usage limitations that are placed on a faulty engine. In practice, the plan will usually involve following the advice from SEPA regarding the faulty engines, and then compensating a decrease in thrust on that engine by increasing the thrust on the other engine. The “optimal” plan will always keep the usage limits requested by SEPA, whereas the “do-nothing” plan disregards these and just provides a symmetrical thrust distribution that matches the total engine thrust demand by the supervisory authority. It is important to understand that the optimal plan does not guarantee that the total thrust provided will be meeting the total demand.

The MEMA then eventually generates alternative plans that are a trade-off between the optimal and “do-nothing” plans. The number of alternative plans is dependant on the Criticality of the faults being addressed, and the total number of plans ranges between two (Criticality Level 4, no alternative plans) and five (Criticality level 1, three alternative plans).

Since the system is meant to address the situation where both engines present a fault, a series of decisional behaviours have been outlined. Depending on the state of SEPA on each engine (see Table 5), the MEMA enters a combined state, which is an abstraction of the type of action that must be taken, basically indicating whether one engine should get priority in Fault Mitigation or if it is instead advisable to simply decrease the total thrust demand. Table 6 represents the decisional matrix for the MEMA state, depending on SEPA state, and explains the meaning of MEMA states.

In the end, the MEMA outputs a list of plans; each plan is represented by two thrust values, one for each engine. The plans are then sent to the supervisory authority, which has the final word on choosing the plan to be actuated. In order to make this decision, the supervisory authority needs an evaluation of the expected outcome of the plans. To obtain this, the proposed thrust levels are fed into the Fault Escalation Prognosis algorithms used in the Prognosis Framework; in this way, each plan can be presented to the supervisory authority together with an estimate of how the plan will affect the escalation timescale. The authority is then able to make a decision based on this data and other data situational awareness data that is not concerning the propulsion subsystem.

As an example, let us consider the following case: while the total engine thrust demand is 70%, a fault at the first severity stage is detected on the right engine. At 70% thrust, the fault would escalate to the next severity stage in one hour time (timecode 6). The SEPA proposes a plan that reduces thrust on the faulty engine to 40%, thus the MEMA proposes an optimal plan with thrust distribution 40%-100%. This plan is evaluated to extend the escalation of the fault to 5 hours time (timecode 8). However, this means that the asymmetrical thrust coefficient is -0.43. Due to the low fault severity, only one alternative plan is generated, corresponding to a 55%-85% thrust distribution (asymmetry coefficient -0.21) and an escalation timescale of 2 hours (timecode 7). These options are presented to the supervisory authority; we can assume that the supervisory authority has knowledge of a minor rudder fault that does not allow for an asymmetry coefficient greater than ±0.3, therefore excluding the optimal plan; however, it knows that the remaining mission time is between one and two hours, so the middle option is chosen since it represents a good trade-off, as the fault does not escalate before the end of the mission and the thrust is provided with an acceptable asymmetry.

This is just an example of how the decision process might work for our proposed Planner system; the range of possible situations is much wider and largely different strategies will be adopted under different conditions, however the governing philosophy remains the same: mitigating faults by reducing engine usage, while at the same time considering situational awareness that is not related directly to the Propulsion system.

4. Simulation tests: Architecture and Results

The Propulsion Health Management System was tested using a simulation environment, also modelled in Simulink. Within this environment, the system receives input including Isolated Faults and UAV supervisory authority commands, processes it and then outputs the plans together with their respective evaluation data. While most of the simulations stop here, where the output of the system is in a format that is meaningful only within this research, we have also performed simulations that included a real engine model. In this case, the plans are fed into a very simple decision-making algorithm that simulates the presence of the supervisory authority by choosing a plan among the proposed ones. The plan is then executed by feeding it into a twin-engine model, so that actual engine running parameters can be monitored. However, these engine models do not take account of the existence of a fault, so they are useful only in determining that the proposed plan can be realistically actuated by an engine.

The results presented in this section are obtained from simulations that do not include an engine model. Simulation runs use files for input and output, so that pre-recorded input can be used and output can be recorded for later analysis.

One type of simulation run is made assigning fixed demands from the supervisory authority and then inputting all of the 841 possible fault input combinations; this type of simulation is useful in proving the determinism of the system and verifying that it always responds within certain constraints. Other simulations involve verifying the behaviour of the system when demands from the supervisory authority are changing.

Finally, since evaluating the correctness and appropriateness of generated plans can be a very complex task, manual simulations were also performed. In these cases, a set of input configurations was determined randomly and then the behaviour of the system verified directly by a human user, which could evaluate the performance of the various components of the system. Such tests made large use of the LabView visual interface, which allows for a clear and immediate understanding of the results.

4.1 Full fault input combination tests

This is the type of test where the demands from the supervisory authority are set and the fault input spans all the 841 possible combinations. The simulation involves changing the fault input at every second of simulation. Figure 6 shows part of the results for such a test; in this case, the total thrust demand is set to 70% (which would normally require 70% on each engine). It is possible to note that the "do-nothing" plan is stable at 70%, while the "optimal" plan varies depending on the type of fault. The number of generated plans also changes, ranging from cases where only two plans are present to cases with five generated plans. It is also possible to note how the non-faulty engine is used to compensate for proposed limitations to the faulty engine, but in many cases cannot guarantee the same amount of thrust since it cannot go over 100%.

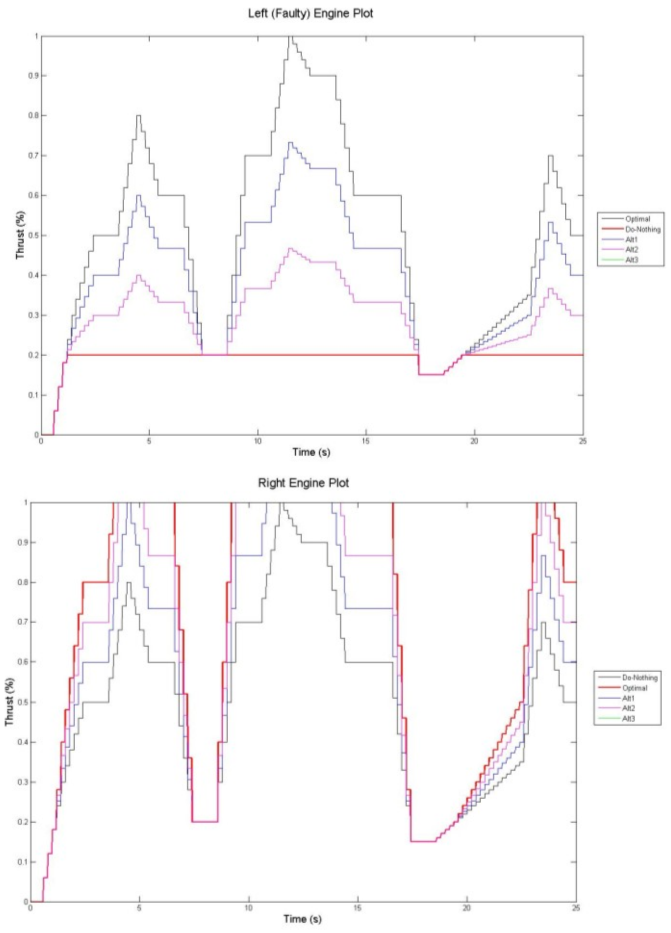

Figure 7. Platform input variation tests

4.2 Platform input variation tests

This is a type of test in which a specific fault condition is fixed and the behaviour of the system on the occurrence of changes in the demands from the supervisory authority (or platform) is monitored. Figure 7 shows the results for such a test; in this case, an outer annulus crack fault at the second severity stage is injected in the left engine and total thrust demand is varied in time. The number of generated plans in this case is four, and the fault is addressed using a limitation at 20% thrust. Note that for total thrust demand equal or less than 20%, four plans are still generated but they converge.

4.3 Manual tests

This type of tests is very simple in concept, just involving manual input through the interface and visual verification of the output. However, due to the complexity of the output data of the Planner system, these manual tests cannot easily be replaced with automated tests, since the analysis of the results involves evaluating a huge amount of data and the relationships between certain values.

For this reason, a representative set of input configurations was chosen and visually verified by a human user in order to verify the correctness and appropriateness of generated plans. Going back to Figure 2 (section 2), we can see capture screens for the visual interface (input and output) for one of these cases. In particular, a case with 80% total thrust demand and different faults on both engines is presented. Looking at the first output screen, it is possible to notice that the fault on the left engine is expected to escalate in timecode 4,while the fault on the right engine is expected to escalate in timecode 7. The faults present different SLEs that are also expected to escalate when the fault reaches the highest severity stage. The fault on the left engine is classified at Criticality Level 2 (Severe) while the fault on the right engine is classified as Negligible; however, both faults are prognosed to escalate to Criticality Level 1 (Catastrophic), although they will do so at different times. The SEPAs generate that place a 20% limit on the left engine and a 50% limit on the right engine. The MEMA decides to follow the "risk-situation" protocol, due to current low level of Criticality on the right engine. This involves actually increasing usage on this engine, since priority is given to addressing the other fault. The four generated plans are 80%-80% (do-nothing), 20%-100% (optimal), 40%-100% (alternative 1) and 60%-100% (Alternative 2). Note that only the Alternative plan 2 does not require a reduction of total thrust provided.

The second output screen analyzes plans in detail, providing the thrust asymmetry and thrust deficiency for each plans and showing the estimated effects on fault escalation. It is possible to note that the limitations on the left engine provide a substantial benefit (from timecode 4 to timecode 7 for the optimal plan), while the added usage on the right engine does not involve a reduced time to escalation (although in reality there will be a reduction, this is not big enough to be captured using the discretized timescale values).

5. Conclusions

This paper presents a system that is designed to maximize the advantages provided by advanced techniques for Gas-Turbine Engine Fault Detection and Fault Isolation. This is obtained by automating the Fault Evaluation and Fault Mitigation functions, which are in current systems delegated to the pilot. The system is designed to be useful in improving autonomous UAV safety by strengthening situational awareness through a detailed and meaningful analysis of the state of the engine and providing reversionary action plans from which the UAV supervisory authority can choose, considering general airframe situation in addition to engine situation. The system is described in detail and examples of simulation results are presented. The system is based on a novel Systems Engineering approach which integrates Soar Intelligent Agents and conventional control systems techniques. This approach was proven to be feasible, but unable to bring significant advantages within this application field. In fact, a comparison with a similar "conventional" system was performed, and the only clearly perceived advantage was a faster development time. This is offset by the many issues that arise when trying to apply unconventional software like Soar within a safety-critical application field such as Gas-Turbine Engines. It is now planned to use the same Systems Engineering approach within the larger problem of UAV Mission Management, which offers significantly more degrees of freedom. It is hoped that the higher complexity of the problem will allow to bring forward more significant advantages in the use of the integrated Soar approach, since Soar agents should scale better in terms of software complexity and hardware requirements when problem complexity arises.

Acknowledgments

The authors would like to acknowledge the support of the DTI, the ASTRAEA Consortium and Rolls-Royce Plc in this work.

References

[1] Goebel G., "History of Unmanned Aerial Vehicles", http://www.vectorsite.net/twuav.html, 2008

[2] UAV Task Force, "The Joint JAA/EUROCONTROL Initiative on UAVs", UAV Task Force Final Report, 2004.

[3] 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*, 7 – 10 January 2008, Reno, NV

[4] Gunetti, Thompson, "A Soar-based Planning Agent for Gas-Turbine Engine Control and Health Management", 17th IFAC World Congress, Seoul, Korea, July 2008

[5] Soar Technology Inc, "Soar – An overview", © 2002

[6] Jennings N., Wooldridge M., "Applications of Intelligent Agents", in *"Agent Technology: Foundation, Applications and Markets"*, Springer, 1998

[7] Wooldridge W., "Intelligent Agents", in *"Multi-Agent Systems: a modern approach to distributed artificial intelligence",* the MIT Press, 1999

[8] Dorfman M., "Commercial vs. Aerospace Worlds: Comparing Software Engineering Culture", in *IEEE Software Journal*, November/December 1999

[9] Kobayashi T., Simon D., "Application of a bank of Kalman filters for Aircraft Engine Fault Diagnostics", NASA/TM-2003-212526

[10] Austin J., Jackson T., Fletcher M., Jessop M., Cowley C. and Lobner P., "Predictive Maintenance: Distributed Aircraft Engine Diagnostics", from *"The Grid 2: blueprint for a new computing infrastructure"*, edited by Ian Foster and Karl Morgan, Kaufmann Publishers, 2004.