Simulation-Based Fault Analysis Methodology for Aerospace Vehicles

Andrés Marcos[*], Gabriele De Zaiacomo[†] and Luis F. Peñín[‡]
Deimos Space S.L., Madrid, 28760, Spain

This article presents a fault analysis methodology and toolbox developed to address the two main questions for any fault analysis methodology: i) what is the impact on the closed loop of a specific fault; ii) which fault is more critical. The methodology is based on the idea of computation-based data acquisition and quantitative-based data analysis. The software fault analysis toolbox developed in support of the process uses XML (Extensible Markup Language) and Matlab/Simulink files to provide a structured functionality that facilitates its use and adaptation to different types of missions, vehicles and simulation tools. Its two main functionalities are the generation of fault-data analysis sets (in a form reminiscent of Monte Carlo campaigns) and the automated evaluation of the results (determination of the fault criticality classification). Special emphasis is given to the manner the results are presented so as to facilitate their interpretation by fault analysts due to the explosion of data arising from the many possible faults, at many possible instances and at many possible components. The validity of the process and toolbox are exemplified using the results from a fault analysis study of an atmospheric reusable launch vehicle during the ascent and re-entry phases.

I. Introduction

Due to the increased levels of autonomy and safety being demanded for current and future aerospace systems, techniques and software tools capable of performing hazard/fault analysis as part of the control verification & validation (V&V) process are receiving considerably attention nowadays[14-15].

From the reliability and safety field there are various methods, each providing different fault/system coverage, that are traditionally used[12-13] for example: checklists, fault/event tree analysis (FTA/ETA), Failure Mode and Effect Analysis (FMEA) and Failure Mode Effect and Critically Analysis (FMECA) among others. The last two are especially important in critical engineering applications, an indeed for space systems they are listed in the European Cooperation For Space Standardization (ECSS) space control engineering requirements[16]. FMES is a reliability approach that uses an inductive (bottom-up / root-to-effect) analysis technique to predict equipment reliability assessment. FMECA approaches are indeed a more detailed FMEA that prioritizes faults based on assignation of failure critical levels. The main objectives[13] of both, and for that matter most other hazard analysis methods, are: to systematically examine a system for potential failures, to analyze the effects of these failures on system operation and to identify corrective actions or design modifications. On the other hand, possible problems include[13]:

1. Insufficient allocated project time for proper analysis
2. Poor understanding and/or inadequate training on the analysis methods
3. Need for computer-aid tools to perform and interpret the analysis
4. Perception of process as difficult, laborious, and mundane

Additionally, most of these methods are not validated nor are capable of providing information readily interpretable and usable by fault tolerant control (FTC) and/or fault detection & isolation (FDI) engineers. Thus, the control community has recently started to develop methods[1-2-3], rooted mostly in modern control analysis, which can be referred in general as fault analysis (FA) and precisely attempt to address this issue.

Any FA methodology must try to answer two questions mainly, and do so in a quantifiable manner in order to be useful for FDI/FTC engineers:

[*] Senior Control Engineer. Simulation & Control section. Email: andres.marcos@deimos-space.com.
[†] Project Engineer. Simulation & Control section.
[‡] Head of Simulation & Control section. Senior AIAA member.

- What is the impact on the closed loop of a specific fault?
- Which fault is more critical?

Furthermore, the FA methodology should be formulated within a framework easily automated (i.e. reliably and efficiently implemented in a software package) to reduce the data-mining[§] aspects of the analysis. Moreover, sometimes there is also the question of whether to perform the FA in open or closed-loop (or at least –and especially for FDI/FTC studies— there is the temptation of performing only an initial open-loop analysis due to programmatic constraints and extrapolate its findings to the closed-loop). Assessing the fault effects in the open loop could lead to wrong answers on fault criticality once the system is in closed loop since the controller could perfectly absorb or accommodate some of the faults considered critical for the open loop system. This masking of open loop fault effects by the controller was shown in Ref[6] from the perspective of the degradation of the open-loop FDI filter performance due to controller robustness.

In order to provide an answer and associate a quantitative measure for each of the above two questions, there are two main distinguishable FA fields in the literature from a control field perspective: analytical (mostly linear) [3-11] and computational[1-2].

The *analytical-based fault analyses* rely mostly on linear approaches and allow direct quantification of the fault effects on the system's performance/stability while providing an elegant mathematical framework. Linearization is fundamental (and non-trivial for most real systems) to apply the standard linear stability, robustness and worst-case methods[5]. Recently, Refs[3-11] presented closed-loop fault analysis approaches using the modelling paradigm of modern control, the linear fractional transformation (LFT). Indeed, the use of LFTs is also fundamental (and non-trivial) to apply linear robust and worst-case methods. Despite the general advantages of analytical approaches, most of those currently in use by industry (i.e. those based on linear techniques as opposed to more sophisticated analytical theories such as Lyapunov frameworks) suffer from the fact that the results stem from linear, time-invariant (LTI) systems. This becomes a especially critical issue for fault analysis as the faults that are likely to be the more safety compromising are those that will behave or initiate nonlinear behaviour of the system (for example, when a fault pushes an actuator to saturate resulting in control input constraint or windup phenomena). Furthermore, faults are typically a time-varying process and the key point in its identification/compensation is precisely in acting before they reach a state where the fault effects become catastrophic. This time-varying, build-up behaviour cannot be studied using LTI techniques. Nevertheless, linear-based FAs are very amenable for preliminary fault analysis and provide an excellent supporting fault assessment tool.

Current acceptable industrial-level FA approaches rely in Monte Carlo techniques (due to certification and safety requirements) and thus fall under the realm of *simulation-based fault analyses*. These FA approaches are computationally very expensive due to the too many possible faults, at too many different times, affecting too many components and subsystems –although these issues affect equally to analytical approaches they are more "expensive" in a nonlinear simulation setting. A further drawback of computational FAs is that quantification of fault effects is not an easy task despite its apparent simplicity, i.e. "just compare to the non-faulty case". Indeed, two immediate questions that arise from the last statement are: i) how to compare, and ii) with respect to which 'baseline' situation to compare. For example, if a decision is made to compare temporal characteristics of a fault, then a further question is what time metric to use: shall it be fastest divergence or the longest time violating a constraint? Similarly, for comparing magnitude metrics, do we prefer farthest departure or accumulative violation of constraints? Indeed, there is not much on the available literature on metrics or quantification of fault effects[1-2-3]. On the upside, computational approaches allow assessing the nonlinear and time-varying effects of faults and can quantify –after a careful selection of evaluation criteria and metrics— the fault effects on the "true" closed loop.

Based on the above issues, an FA approach is proposed in this article that uses high-fidelity nonlinear simulations to get the required data (i.e. computation-based measurement) and quantitative-based metrics and criteria to assess the fault impact on the closed loop system. In developing such an approach, it is highlighted again that two key issues are to adequately formulate the fault analysis framework and to automate it. The second is highly non-trivial since the explosion of data resulting from a comprehensive FA will excessively burden the analysis of the results otherwise. A five-step process is followed to satisfy these two issues (and define the layout of this article):

- *Fault model development*, Section II: from the perspective of fault analysis, this implies to develop a mathematical framework for fault modelling capable of considering a wide class of fault families. Additionally, in order to satisfy the automation issue the mathematical fault model should be easily implementable in Matlab/Simulink.

[§] By "data-mining" it is meant the analysis of the (exponentially) accumulated data arising from the FA of a system with multiple inputs and outputs subject each to a multitude of different faults occurring at many different times.

- Develop *fault quantification metrics and index*, Section III: these must be well reasoned and physically motivated metrics, i.e. should consider temporal and magnitude criteria. The metrics shall allow numerically quantification of the effects from the different faults. Based on these metrics, a global index normalizing the contribution from each metric shall be derived to allow for numerical comparison and classification of the faults.
- Develop *fault classification taxonomy*, Section IV: in order to automate the analysis process it is required to develop a fault classification taxonomy that categorizes the fault by criticality and provide different thresholds for analysis of the fault effects on the system. These thresholds and critical level definitions should be used to tune the previous fault index so that each fault can be immediately assigned to the corresponding level to allow for a quantified assessment of its importance.
- Implement and *perform intensive nonlinear closed-loop simulation campaign and assessment of the results*, Section V: all the above elements must be integrated into a high-fidelity functional engineering simulator (FES) in order to perform reliable nonlinear analyses. The resulting fault analysis FES shall then be used in intensive simulation campaigns to gather the required data for analysis. Subsequently, using the fault classification taxonomy and index (developed based on the fault quantification metrics) the results obtained from the intensive simulation fault campaigns can be readily classified and presented to the user. A key issue in this element is to present the analysis data in such a form that the user can extract the main features and conclusions in a straightforward manner.

The proposed fault analysis approach is implemented in a software package named fault analysis toolbox (FAT). The underlying FAT concepts and metrics are very general and applicable to a wide range of systems but at present, it has been tailored and applied to the fault analysis of the Hopper reusable launch vehicle[7].

II. Fault models

The first step to establish a FA methodology and toolbox is to describe, formulate and develop fault models. Thus, it is necessary to begin by describing the fault scenarios and their characterization, subsection A, which will lead to the mathematical formulation of the adequate fault models, subsection B, and subsequently to the development of an implementable fault model, subsection C.

A. Fault scenario and characterization

In characterizing the fault scenarios the following taxonomy is used:

- *Fault classes*: makes reference to the component that fails (i.e. sensor, actuator or system).
- *Fault profile*: represents basic fault evolution in the time domain (e.g. abrupt, intermittent and incipient) and shape of this evolution (e.g. step, ramp, pulse).
- *Fault type*: makes reference to the physical meaning of the fault within the faulty component (e.g. lock-in-place, drift...)

The set of *faults classes* considered in the present fault analysis methodology are circumscribed to the following two families: sensors and actuators. Regarding *fault profiles*, it was noted before that a pitfall of many safety-related projects is that the allocated time for fault analysis is very conservative. Thus, prioritization of the efforts must be performed, which is related to fault classification in term of its complexity for FDI/FTC design (e.g. trying to identify incipient faults first will likely consume most efforts since it is the most difficult type of faults to be identified). In order to provide an answer to this problem, a systematic FDI development method must be used where the fault analysis hierarchy is determined by the fault complexity (measured in terms of difficulty of estimation), that is, the process should be *from detection of abrupt faults to identification of incipient faults.* A basic classification of faults in terms of FDI complexity is given next, see also Fig. 1:

- <u>Abrupt</u>: abrupt faults occur instantaneously often as a result of hardware damage. Usually, they are easier (depending on their severity) to estimate than other fault profiles. They might strongly affect the performance and/or the stability of the controlled system.
- <u>Intermittent</u>: intermittent faults (including oscillatory) are those that appear and disappear repeatedly, for instance due to partially damaged wiring. It is assumed that an intermittent fault is an abrupt fault active only during a short time –thus, its FDI is more difficult than that for permanent abrupt faults.
- <u>Incipient</u>: incipient faults represent slow (in time) changes, often as a result of aging. They are more difficult to detect (specially on-board) due to their slow time characteristics. They are also initially less severe but could develop in time into severe faults if left uncorrected.

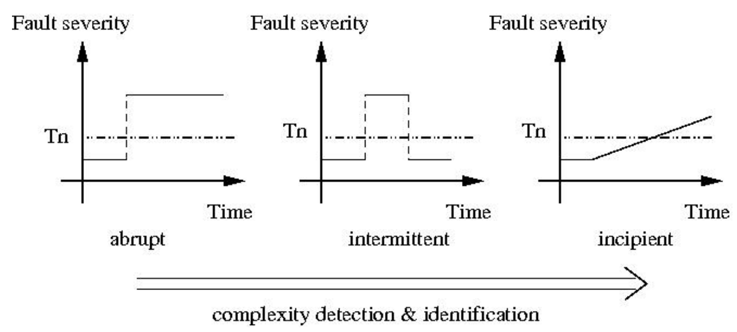


Fig. 1 Classification of faults based on time characteristics[**]

Finally, *Fault types* refer to the effect of the fault in the component. The fault effect could have different root cause of failure (RCOF), but in the development of the fault analysis method and toolbox our interest is only in detecting and identifying their effects on a GNC system *–without going into details on the physical origin of the fault, which would be part of a FMECA analysis.* A summary of the fault scenarios considered for fault analysis is shown in Table 1.

Table 1 Characterization of faults to be considered in the fault analysis toolbox

Fault characterization	**Options**	
Fault classes	· Sensor · Actuator	
Fault profiles	· Abrupt · Intermittent · Incipient	
Fault types	**Actuator**	· Blocked output (dead actuator) · Hard-over (saturation) · Loss-effectiveness · Random drift · Constant offset
	Sensor	· Blocked output (dead sensor) · Arbitrary saturation limits (hard-over saturation) · Constant offset (bias) · Random drift · Loss of effectiveness

B. Mathematical fault modeling

After characterizing the fault scenarios of interest, and prior to developing an implementable fault model, it is necessary to develop a general fault mathematical representation.

Two clear formulations can be chosen to represent a general fault model within the realm of modern control: additive and multiplicative. Both are acceptable, and equivalent to a certain level. An additive fault model is used since it provides complete flexibility to capture all representative faults/failures[††] cases in the context of state-space representations.

Consider the following mathematically formulation of a generalized unfaulty system based on state-space theory:

$$\dot{x}(t) = Ax(t) + Bu(t) = Ax(t) + \sum_{i=1}^{m} b_i u_i(t) \qquad given\ x(0)$$
$$y(t) = Cx(t) \qquad \text{Eq. 1}$$

Where $x(t) \in \Re^n$ is the state vector, $u(t) \in \Re^m$ the control vector, and $y(t) \in \Re^l$ the output vector. Therefore, $A \in \Re^{n \times n}$ represents the state matrix, $B \in \Re^{n \times m}$ the input injection map (thus, b_i constitutes the *i-th* control column), and $C \in \Re^{l \times n}$ is the output map. It is noted that the state matrix A can be considered to include as well the dynamics of the actuator and sensor sub-systems.

[**] A fault detection threshold *Tn* has been included in Fig. 1 to highlight that the standard FDI approach is to identify a minimal fault magnitude over which the faults need to be estimated.

[††] Note that in the paper, the terms fault and failure (i.e. a catastrophic fault) are loosely used.

The *general additive fault model* proposed is then given by:

$$\dot{x}(t) = Ax(t) + Bu(t) + L(t)\nu(t) \qquad given\ x(0)$$
$$y(t) = Cx(t) \qquad \text{Eq. 2}$$

Where *L(t)* is the failure signature and *ν(t)* the failure signal and both can be used to model a wide number of actuator faults and sensor faults –the latter requiring re-formulation as pseudo-actuator faults by means of state augmentation [8]. Table 2 shows the appropriate selection for actuator faults using the *L(t)ν(t)* fault term in Eq. 2, which for this fault class assumes the following structure[4]:

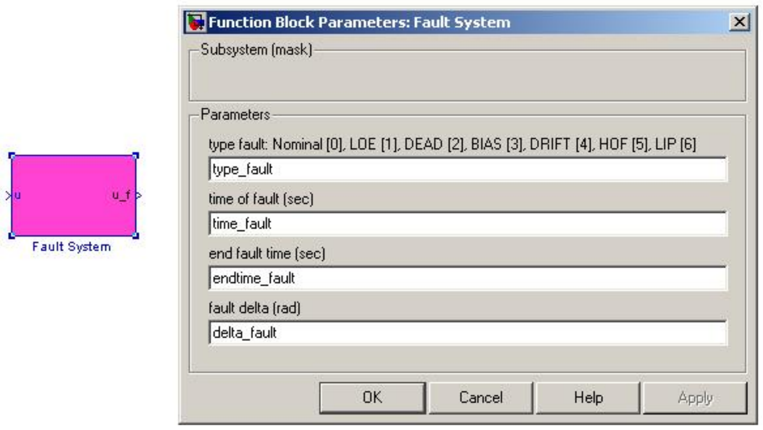
$$L(t)\nu(t) = \sum_{i=1}^{p} \left[\Delta b_i u_i(t) + b_i \Delta u_i(t)\right] \qquad \text{Eq. 3}$$

Table 2 Representation of common actuator faults[‡‡]

	Type	Element fault, $\Delta b_i u_i(t)$		Input fault, $b_i \Delta u_i(t)$	
Actuator	Hard-over (saturation)	Δb_i=0	$u_i(t)$= *arb*	b_i=b_k	$\Delta u_i(t)$=-($u_{kmax/min}$+$u_k(t)$)
	Loss-effectiveness	Δb_i=0	$u_i(t)$= *arb*	b_i=b_k	$\Delta u_i(t)$=-α $u_k(t)$
	Floating actuator	Δb_i=-b_k	$u_i(t)$=$u_k(t)$	b_i= *arb*	$\Delta u_i(t)$=0
	Bias (stuck actuator)	Δb_i=b_k	$u_i(t)$=α	b_i= *arb*	$\Delta u_i(t)$=0
	Noise	Δb_i=b_k	$u_i(t)$=η(t)	b_i= *arb*	$\Delta u_i(t)$=0

C. Fault model implementation

The next step is to port the above general additive fault model to a general Simulink model in order to be used within one of the most commonly available simulation environments. In order to facilitate usage, the fault model implementation should include the main fault types defined above and also allow the user to specify initial/end time of the fault activation and its magnitude. A specific Simulink masked block Fig. 2 has been developed which can be simply added to any actuator/sensor channel within a Simulink environment. The faulted output u_f from the Simulink fault model is as described by the aforementioned fault additive model representation: u_f=*u*+*Δ* where *Δ* is used to properly model the fault type effects and *u* is the nominal input signal.



Function Block Parameters: Fault System

Subsystem (mask)

Parameters

type fault: Nominal [0], LOE [1], DEAD [2], BIAS [3], DRIFT [4], HOF [5], LIP [6]

type_fault

time of fault (sec)

time_fault

end fault time (sec)

endtime_fault

fault delta (rad)

delta_fault

Fig. 2 Fault Simulink configuration mask

The fault types considered, as indicated in the top parameters of Fig. 2, are: loss-of-effectiveness (LOE), dead (DEAD), bias (BIAS), drift (DRIFT), hard-over (HO) and lock-in-position (LIP). These represent some of the most common fault types (but note that some of them make sense only for actuators or for sensors, see Table 1). The parameter *fault_delta* allows the user to specify the magnitude component of the fault, which depending on the *type fault* parameter will have a different meaning (e.g. for drift faults, *fault_delta* determines the slope of the drift ramp while for bias faults, it specifies the constant magnitude being added to the nominal signal). Note that by using cascade and series implementations of the fault model most of the relevant fault profiles can be obtained.

The detailed implementation of the fault models is shown in Fig. 3 and Fig. 4. The figure on the left shows the top-layer of the implemented masked Simulink fault model while the right figure the bottom-layer implementation

[‡‡] In the table, α represents a constant value, '*arb*' an arbitrary value (e.g. constant, time-varying...) and for ease of notation the sub-index *k* indicates *k*th channel.

for LOE, DEAD, BIAS and HO faults. Note that in Fig. 3, the LIP fault is considered independently to the other modeled fault types due to its special modelling complexity, which requires 'freezing' the value of the signal at the time of the fault occurrence. By appropriately selecting the gains *Delta_F1* and *Delta_F2* in Fig. 4 the specified fault type can be simulated –note that depending on the selected 'type fault' parameter, these *Delta_F*'s are fixed values (as those from Table 2) or correspond to the value given by the parameter 'fault_delta'. The Drift fault type has been implemented slightly different to the Fig. 4 fault types, i.e. the (square) constant term in Fig. 4 is substituted by a ramp Simulink block with tunable slope.

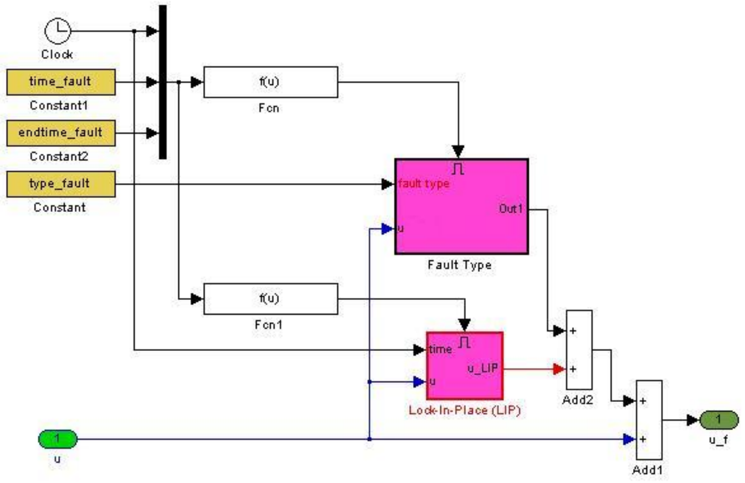


Fig. 3 Fault System model: top layer

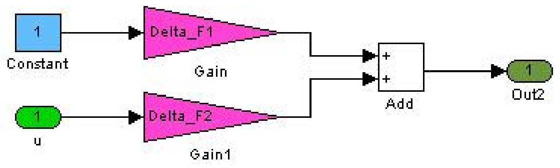


Fig. 4 Model template for LOE, Dead, Bias and HO

III. Fault metrics & Comparison Index

Once the fault mathematical model is developed and implemented, the next step is to develop appropriate metrics to quantify the fault effects and also selecting a general fault index that considers all these effects.

D. Fault metrics

Due to the FA connection to simulation-based analysis, the metrics are developed from a time-domain perspective and thus are divided into time and magnitude indexes (which includes both distances and errors). They consider general signal characteristics but are numerically defined based on system-specific bounds, constraints and thresholds. For example, they can be related to the performance specifications of the control design objectives (e.g. tracking errors) or to physical limits beyond which the system might be lost (e.g. heat/load constraints). Table 3 shows the selected quantification metrics.

Table 3 Temporal & spatial fault metrics

ACRONYM	NAME	DESCRIPTION
T_PB	Time to performance bound violation	Time interval necessary for the observed signal to violate a performance bound (e.g. a control objective limit).
T_MX	Time to maximum violation	Time interval necessary for the observed signal to reach its maximum value.
T_BU	Break-up time	Time interval necessary for the observed signal to violate a critical bound (i.e. that will result on loss of vehicle or mission).
Hdist	Distance from the constraint	Magnitude between observed signal value and a physical constraint (e.g. heat flux or gravity limits)
ε_∞	Maximum error	Maximum difference, in an L_∞ sense, between observed signal and a reference nominal profile of the signal
ε_2	Maximum violation area error	Maximum area (L_2 sense) above a performance or critical bound.

1. Time-related indexes

The necessity to monitor the time evolution of the fault effects is obvious, as is the need for selecting metrics that facilitate understanding of these effects. For example, certain faults result in fast violation while others result in longer violation (of limits). Additionally, these metrics should not be restricted to a specific signal type, i.e. to a measurement or to its error, although the selection of the signal to be observed has wider implications.

Three indexes are proposed capturing the most relevant temporal characteristics of a fault profile. Fig. 1 shows an example of two temporal fault metrics for the case of an angle of attack (AoA) error profile during the ascent phase of the Hopper RLV[7]. Notice that a performance bound (PB) is used to numerically determine one of the time indexes. It is important to highlight that these indexes facilitate the estimation of the temporal horizon for FDI, i.e. the temporal fault metrics help establish minimum and maximum times for fault detection and isolation.

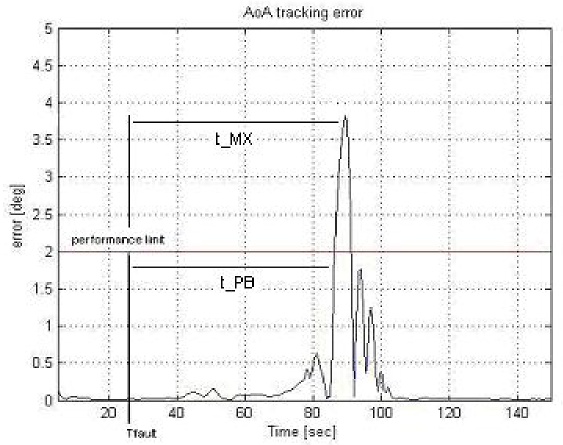


Fig. 1 Time-related performance indicators

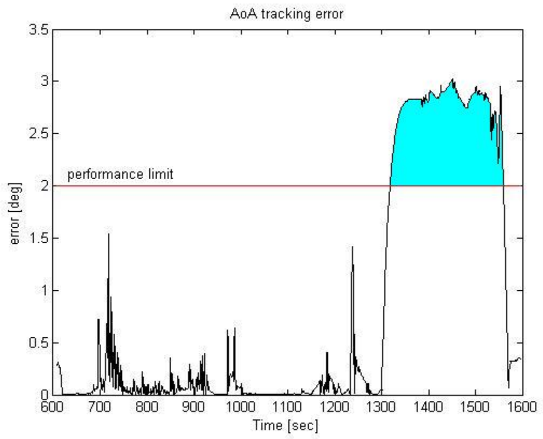


Fig. 2 Maximum violation area error

- *t_PB*: time to performance boundary violation

The performance boundary violation time *t_PB* is defined as the time interval necessary for the observed signal to violate a specified performance bound *t(PB)* starting from the time of the fault occurrence t_{fault}:

$$t_{PB} = t(PB) - t_{fault} \quad \text{Eq. 4}$$

Different *t_PB*s, possibly one for each performance variable and error signal, can be defined. If a variable does not violate the performance boundary, then the corresponding *t_PB* is set to 0.

- *t_MX*: time to maximum violation

The time to maximum violation *t_MX* is defined as the time interval necessary for the observed signal to reach its maximum value, starting from the time of the fault occurrence. The *t_MX*s always exists (even if it does not surpass the performance boundary its value is still registered).

- *t_BU*: break-up time

The break-up time *t_BU* is defined as the time interval necessary, starting from the time of the fault occurrence, for the observed signal to violate a critical threshold[§§]. Its mathematical definition is exactly as that for *t_PB*, but the violated limit is set to a different (upper) value. Again, this general definition allows the existence of different break-up times depending on the performance variable studied. The computation of each t_BU_i (with *i* indicating the different observed signals) is independent for each observed variable but the final value considered for a specific fault is the minimum of the resulting *t_BU* set. When a critical threshold is reached, then *t_BU* and *t_MX* coincide. In case the threshold limit is not reached, the corresponding *t_BU* is set to zero.

2. Magnitude-based metrics

Two general classes are considered: distance from constraint and error-based metrics.

- Hdist: distance from the constraint

The magnitude between the observed signal value and its (physical) constraint limit can be seen as an indicator of the danger related to that fault. For example, the occurrence of a fault could raise the peak heat flux value of a re-entry vehicle close to its maximum allowable magnitude indicating that this fault is critical.

[§§] A critical threshold defines the maximum value a signal can reach before it results in a catastrophic event (either mission loss or system/sub-system loss) that de-stabilizes the system.

The metric is computed as follows (*h* represents the monitored variable and h_{const} is the constraint limit –there is one *Hdist* for each constrained variable *h*):

$$Hdist = \frac{\max(h)}{h_{const}} \qquad \text{Eq. 5}$$

- Maximum error, ε_∞:

Error profiles of selected signals (e.g. control and guidance tracking) can be normalized with respect to reference values resulting in a measure of deviation from nominal behaviour. Since a weighted error profile provides a more explicit quantification on the degree of violation of a threshold, the maximum error metric is given by:

$$\varepsilon_{\infty i} = W_i \cdot \varepsilon_i \qquad \text{Eq. 6}$$

The weights are taken to be equal to $1/u_{L_i}$, where u_{L_i} corresponds to a performance boundary (not a critical threshold).

- Maximum violation area error, ε_2:

An indicator of the criticality of a threshold or bound violation is the violation area ε_2. The violation area is easily computed with the rule of trapeziums for each error profile. An example of ε_2 is illustrated in Fig. 2, where again an AoA error profile is used but this time during the re-entry phase.

E. Global classification index

All the described metrics are then combined in a weighted cost function that allows to automatically classify the faults into critical categories (following the fault categorization given in the next section). The designed cost function is called Global Criticality Index (GCI) – the sub-index *i* indicates signal under examination, e.g. $i = \alpha, \beta$:

$$GCI_i = GCI_{temporal\_i} + GCI_{spatial\_i} \qquad \text{Eq. 7}$$

$$GCI_{temporal\_i} = W_{BU} \cdot GCI_{BU\_i} + W_{PB} \cdot GCI_{PB\_i} + W_{MX} \cdot GCI_{MX\_i} \qquad \text{Eq. 8}$$

$$GCI_{spatial\_i} = W_{Hdist} \cdot GCI_{Hdist\_i} + W_\infty \cdot GCI_{e_\infty\_i} + W_2 \cdot GCI_{e_2\_i} \qquad \text{Eq. 9}$$

All the weights W_* are tuned based on an initial "learning" simulation data-set (using selected faults and nominal references) and serve to highlight the desired importance of the faulted profiles with respect to the analyses performed. The sets $CGI_{BU\,/\,PB\,/\ldots}$ are defined by normalizing the effects of a specific fault in the corresponding metric set. Typically, the normalization uses the maximum, minimum or total sum for the observed signals' metrics.

IV. Fault criticality categorization

Once appropriate metrics and a normalizing index have been defined, in order to automate the analysis process it is required to develop a fault classification taxonomy that categorizes the fault by criticality based on specially defined thresholds. These thresholds and critical level definitions are used to tune the previous fault index so that each fault can be immediately assigned to the corresponding level and thus allow for a quantified assessment of its importance.

F. Fault thresholding categorization approach

The proposed fault analysis process is based on a fault thresholding classification scheme Fig. 3 that helps identify the different FDI stages (prognosis, detection and identification) based on numerically defined thresholds – which are system-dependent.

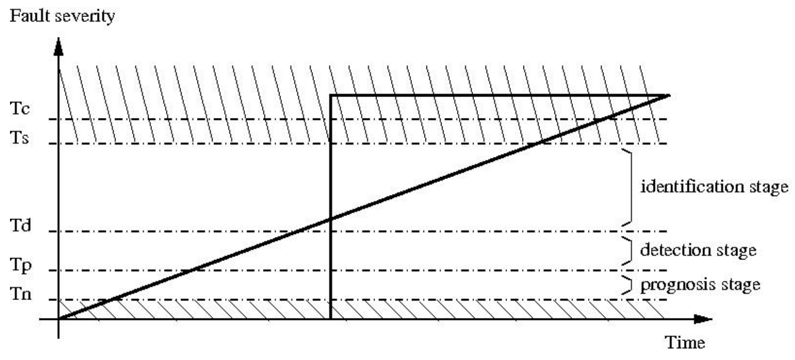


Fig. 3 Fault thresholding classification

The proposed thresholds are defined as follows:

- Noise threshold, *Tn*: represents the minimal detection threshold below which it is not possible to differentiate between noise and fault effects. It can be chosen by means of classical statistical residual and Root-Mean-Square (RMS) analyses for the un-faulty closed loop.
- Prognosis threshold, *Tp*: it is the maximum fault level the controller can tolerate (before requiring fault accommodation/tolerance properties) without degradation of closed-loop stability or performance. This value is obtained through robustness/fault analyses of the closed loop and using as reference the design specifications. This threshold was referred to previously as performance boundary.
- Detection threshold, *Td*: obtained through fault tolerance analysis of the closed loop in combination with failure analyses. It defines the level above which a fault needs to be identified, and below which its detection is sufficient.
- Safety margin threshold, *Ts*: it is the selected level at which drastic controller and/or pilot action is required for the safety of the mission or system (e.g. move to a fail-safe state, closed loop reconfiguration, or warning to abandon vehicle). It can be chosen using empirical methods (e.g. experimental, industrial experience, and/or manufacturer's technical data).
- Critical threshold, *Tc*: it defines the threshold at which a fault results in system instability, typically when it results in a failure (a catastrophic event at component and/or system level, e.g. complete malfunction of an actuator or of a component fault which results in loss of the whole system).

The above threshold classification allows differentiating three clear FDI stages:

- Prognosis stage: located between the noise and the prognosis thresholds. It is used for trending and prognosis of incipient faults: a very difficult task if tackled on-line. The controller is naturally robust (i.e. no reconfiguration or external robustification is required) to faults whose magnitude level is below the prognosis threshold, but their detection, identification and monitoring is of great interest for maintenance and safety purposes.
- Detection stage: during this stage it is considered that detection of an abnormal situation below the *Td* severity level (with no identification of the faulty component) is sufficient to perform an adequate fault-robustification of the controller. This can be related to the unstructured residual concept and the lumping of failure modes and effects.
- Identification stage: the stage at which the severity of the abnormal situation requires identification of the faulty component to avoid a too conservative robustification of the controller (i.e. a 'blind' overtly conservative robustification of the controller will result in minimal achievable performance, which might not be acceptable). Structural reconfiguration of the controller, i.e. control allocation or alternative sensor measurements' activation, might be required.

G. Categories of fault criticality

The proposed fault analysis approach is based on the concept of fault criticality which we develop based on the categorization given in Fig. 3 (where the y-axis presents a measure of the fault criticality based on fault intensity or fault effects' magnitude). For each defined channel and fault type, the specified thresholds can be considered as fault or fault effects magnitudes that delimit the level of criticality. The following fault critical categories are proposed reflecting these concepts, Table 4 summarizes them:

- Category 1 (level > Tc): faults that lead to an anticipated stop of the simulation (e.g. the guidance or the control error reaches an extremely large value, indicating system instability and vehicle/mission loss) belong to this category. To avoid system's rupture and/or mission failure these faults must be detected and control reconfiguration might be necessary.
- Category 2 (Ts < level < Tc): these faults won't lead to a simulation break up but the tracking error reaches unphysical levels (thus, mission/vehicle failure is also possible). These faults require identification, as controller reconfiguration is necessary. The main difference with C1 faults is that these faults do not result in system instability.
- Category 3 (Td < level < Ts): these faults can be controlled by the closed loop system, but reconfiguration might be necessary to avoid excessive deviations from nominal values. These faults indicate a possible failure of the mission but not a system loss.
- Category 4 (Tp < level < Td): these faults can be controlled by the system without any reconfiguration, but the control design performance levels are violated during flight. They result in degradation of the performance / robustness characteristics of the closed loop but pose no threat to mission success.
- Category 5 (Tn < level < Tp): these faults are easily controlled without violating any of the performance bounds.
- Category 6 (level < Tn): these faults can be assimilated to a disturbance or a system perturbation, because they do not introduce a relevant error in the control profiles.

Table 4 Fault classification taxonomy

Category	THRESHOLD	DESCRIPTION
C1	Fault level > critical threshold Tc	Faults for which the system becomes unstable leading to loss of the mission or vehicle
C2	Ts safety threshold < level < critical threshold Tc	Faults that will not lead to break up but where tracking errors reach levels above safety margin.
C3	Td detection threshold < level < safety threshold Ts	Faults that do not violate safety margins but result in excessive deviations from performance bounds
C4	Tp prognosis threshold < level < detection threshold Td	Faults resulting in degraded performance but present no risk to mission success.
C5	Tn noise threshold < level < prognosis threshold Tp	Faults that do not violate any performance bounds although are noticeable within the closed loop.
C6	Fault level < noise threshold Tn	Faults with similar or lower effect than noise disturbances and system perturbation levels.

While the separation between C1, C2, C5, and C6 categories is clear, it is more difficult to exactly establish a distinction between C3 and C4, because their associated thresholds are more difficult to obtain (specifically, the detection threshold *Td*). Therefore, in the following fault analysis, C3 and C4 categories will be considered as a single category, named C34.

V. Fault Analysis Toolbox (FAT)

In order for the previous fault analysis process to be used it has to be implemented in structured software that allows for automatic generation and evaluation of fault data sets. The automatic generation component must be such that it can be adequately interfaced with the most widespread simulation tools and standards (e.g. Simulink/Matlab, XML...). Similarly, the automated evaluation component must include functionalities for efficiently presenting the results to the analyst if they are to be understood and used.

Deimos FAT represents such software. It is developed to be easily adaptable to functional engineering simulators (FES), which are high-fidelity assessment environments used within industrial consortiums for design and validation[7], and automatically provides graphical and tabular analysis results in a format readily interpretable. In the following subsections, these two main functionalities are described.

H. XML Fault Data File: data generation

Extensible Markup Language (XML) is a set of syntax rules, guidelines and mechanism providing a standard language for the creation of structured documents[9]. XML specification is being introduced in industrial simulators to standardize and automate their development and distribution. Indeed, NASA has strongly pursue the use of this language within their projects (including Orion, X48A, X37…) and estimates that savings of up to $6 Million in productivity gains and reduced simulator down-time for a single aerospace system could be obtained in one year by adopting a distribution standard based on XML[10].

An XML-based FES is composed of a Simulink/Matlab simulation environment and XML data files, whose structure is formally defined by a specific XML schema based on a set of tables. All the XML data files can be opened and edited by any text editor, although, specialized XML editors facilitate their handling. Each table is a set of records and each record may contain a number of fields of two types: parameters and links. The links can access other XML data files, table or records and together the parameter field, both serve to specify unequivocally the FES (e.g. sets the simulation parameters to specified, and traceable, values).

The fault analysis toolbox uses an XML data file named *faultsDF*. This fault data file is dedicated to the definition of *failure scenarios*, *fault modes*, *fault sets* and *faults* following the definitions presented in the previous sections. *Fault modes* are regarded here as sets of individual faults injected into a subsystem. A *failure scenario* consists of all the fault modes and the related fault data that must be considered for the simulation of a mission scenario. In general, the fault modes of a failure scenario can occur at different times. Table 5 shows the XML structure of the fault data file:

Table 5 Fault Data File Parameters

Tables	Record field	Type	Description
scenario	(*)	link	Links to fault modes
mode	name	parameter	Name of fault mode block in Simulink model
	(*)	link	Link to faultset
faultset	(*)	link	Links to individual faults
fault	name	parameter	Name of fault in model
	type	parameter	Type of fault
	t_from	parameter	Time of occurrence (s)
	t_hor	parameter	Horizon time (s)
	fs	parameter	Fault strength

Note that Table 5 follows the XML scheme described above (tables $\rightarrow$ records $\rightarrow$ parameters /links). Each of the tables are described next in a reverse order:

- The ***fault*** table describes the individual faults for each component. The model of a generic fault occurrence is characterized by two time (*t_from* and *t_hor*) and a fault strength (*fs*) parameters, following the fault model definition given in subsection C, see also Fig. 5.

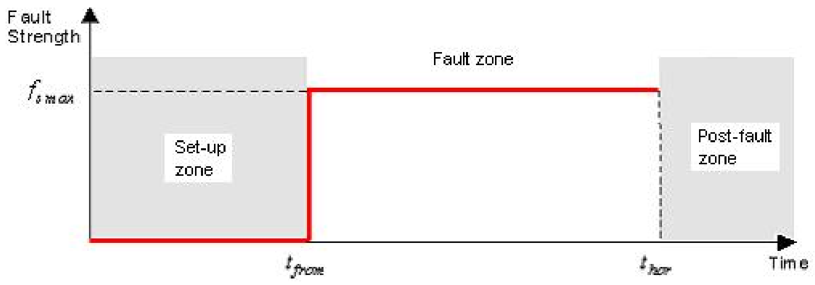


Fig. 5 Fault characterization

There is also a *name* parameter that indicates the name of the fault block in the Simulink model and a *type* parameter that selects the type of fault (following the fault types described in Section A). The fault strength parameter value can be tunable, depending on the fault type considered, see an example of the typical descriptions in Table 6.

Table 6 Fault strength variability

Type	Description	Variability	values	Description
1	LOE	(0, 1]	[0.25 0.5 0.75]	0 = nominal case, 1 = dead; Gain applied to the nominal value of the related channel (e.g. to simulate a reduced capability of action for actuators)
2	DEAD	no	0	To simulate dead component, the value must be fixed to 0
3	BIAS	∞	0.1/0.01*bias_act 0.1/0.01*bias_sen	The bias value is completely tunable. Example considers bias of 1 and 10 % of upper saturation values (bias_act) for actuators, and of nominal range values for sensors (bias_sen)
4	DRIFT	∞	[0.001 0.1]	The drift value is completely tunable. Example considers an increment of 0.1 % and 1 % of the signal value
5	HO	max, min	ho_act	The hard over value is fixed to the upper saturation (max) or the lower saturation (min) values of actuators. Hard over for sensors is meaningless
6	LIP	n/a	n/a	The output of the considered channel is fixed to the constant value

- The ***faultset*** table unifies links from different faults in a singular set that depends on the subsystem a single fault is affecting (actuators or sensors).
- The ***mode*** table is made of fault mode definitions consisting of sets of links to the *faultset* table. They include an additional parameter, the *name* parameter, which indicates the name of the fault mode block in the Simulink model.
- The ***scenario*** table consists of collections of links to fault modes. By calling the assigned name from a simulation XML data file, the FES is prepared to run under the specified fault setting.

Table 7 exemplifies the contents and settings for the Hopper RLV tailored fault data file[7].

Table 7 Fault Data File Default Contents

Tables name	Record name	Description
scenario	FAULT_FREE	Links to fault modes for a fault-free scenario
mode	ACT_FAULTS_FRE	Link to an actuator fault set
	SEN_FAULTS_FRE	Link to a sensor fault set
faultset	faultset_ACT_FREE	Links to actuator faults for a fault-free scenario
	faultset_SEN_FREE	Links to sensor faults for a fault-free scenario
fault	ERO_fault	Fault parameters for right outbound elevon: fault-free case
	ELO_fault	Fault parameters for left outbound elevon: fault-free case
actuator faults	ENG1dq_fault	Fault parameters for Engine 1 gimbal pitch: fault-free case
	ENG1dr_fault	Fault parameters for Engine 1 gimbal yaw: fault-free case
	Psen_fault	Fault parameters for rotational rate p sensor: fault-free case
	Qsen_fault	Fault parameters for rotational rate q sensor: fault-free case
sensors faults	Rsen_fault	Fault parameters for rotational rate s sensor: fault-free case
	BETsen_fault	Fault parameters for SSA sensor: fault-free case
	QDYNsen_fault	Fault parameters for dynamic pressure sensor: fault-free case
	Msen_fault	Fault parameters for Mach number sensor: fault-free case

By appropriately setting up fault scenarios within the XML *faultsDF* and incorporating this file into an XML-based functional engineering simulator (FES) a complete fault simulation environment is obtained to generate the required fault data sets. To set up a fault scenario, it is possible to proceed in two ways:

1. Modify the values in the fault table records defining fault type, occurrence time, horizon time, and fault strength (the default, 'fault-free', scenario sets all to 0).
2. Keep the 'fault-free' scenario and add similar tables to build new fault scenarios (appropriately changing the values). This second approach is better for traceability.

Note that in both cases, the name of the fault blocks must be the same as in the default scenario for each component and mode. This is because the correspondence between XML file and Simulink model must be guaranteed always. For the link parameter in the scenario and mode tables, the linked records can be changed but the id of the links must be also maintained. In the second case it is also important to maintain the fault class and to properly link the fault sets to the fault modes (e.g. do not put a dynamic pressure sensor fault in a fault set that is linked to an actuator fault mode).

I. Analysis functional structure: data analysis

The main tasks performed by the FAT analysis functionality are presented in Fig. 6 and listed below:

- The first step involves two dedicated functions that load the data-analysis set from the data-results files: one of the functions loads the nominal results (to be used for comparison purposes) while the other loads the fault-simulation results.
- The second step classifies the faults into the six proposed critical categories.
- The third step ranks faults within each critical category set based on their respective importance (given by the value of the global criticality index).
- Finally, analysis results are stored.

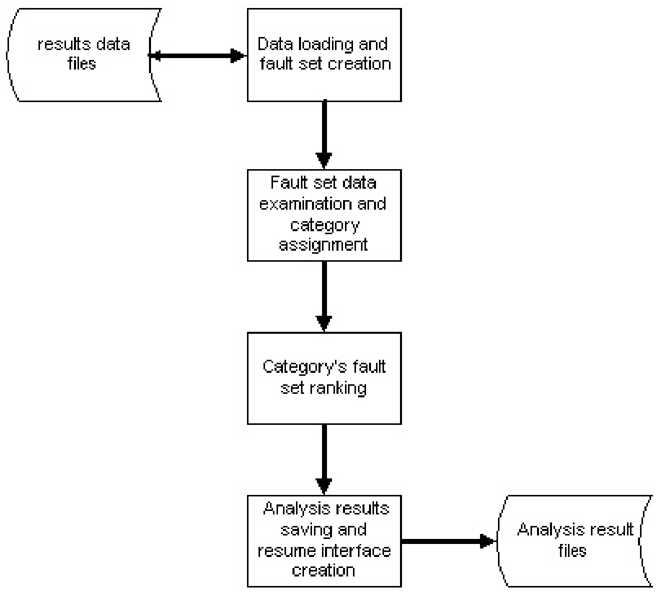

results data files
Data loading and fault set creation
Fault set data examination and category assignment
Category's fault set ranking
Analysis results saving and resume interface creation
Analysis result files

Fig. 6 FAT analysis main functionalities

A key component in the analysis capabilities of FAT is the manner the results are presented to the user. This is very critical and has already been tackled initially when proposing the fault metrics, a global classification index and a fault criticality hierarchy, all of which allow automating the analysis and presenting a ranking of faults. Nevertheless, due to the potentially large number of fault cases, the ranking and associated information must be presented in a manner that a human analyst can readily interpret. This is accomplished in a two-fold manner:

1. First, by providing an automated color-coded graphic summary showing which fault set has been simulated and at which critical category it belongs.
   For example, Table 8 shows the results from Ref7 on the fault analysis of the Hopper RLV during ascent. The columns show the fault types tested (e.g. F1= 25%-LOE, F4=DEAD...) and the rows the fault elements (e.g. E1= Right Outboard Elevon, E10= First Engine Gimbal Pitch). The table cells are numbered in terms of critical level and color-coded accordingly from highest criticality 1=red (darkest) to lowest 5=green (mid-dark). Notice the immediate identification of the more critical fault types and elements for the presented case.

Table 8 Fault summary for fault occurrence at time 25 s during Hopper RLV ascent[7]

	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10	F11
E1	5	3	1	1	5	3	1	1	1	1	1
E2	5	3	1	1	5	3	1	1	1	1	1
E3	3	3	1	1	5	3	1	1	1	1	1
E4	3	3	1	1	5	3	1	1	1	1	1
E5	5	5	5	3	5	5	3	1	1	1	3
E6	5	5	5	5	5	5	5	1	1	1	5
E7	5	5	5	5	5	5	5	5	1	1	5
E8	5	5	5	5	5	5	5	5	5	5	5
E9	5	5	5	5	5	5	5	5	1	1	5
E10	1	1	1	1	5	1	1	1	1	1	1

2. Second, the user has the choice to obtain a wide number of additional graphical and numerical results through a simple Matlab graphical-user-interface (GUI) or by line commands.
   An example of a tabular representation is given in Table 9 for the results related to three C1 (critical level 1) fault cases of the Hopper RLV[7]. For each fault, the table lists the element, fault type, time of fault occurrence, total GCI and all the GCI contributions (recall Eq. 8 and Eq. 9). In all cases, the faulty component, type of fault and fault time occurrence are given in an alphanumeric format pre-defined in the XML files (in this case E22= angle-of-attack sensor, F8=Drift, T3=250 seconds after ascent initialization; E16=roll sensor, F4=Dead, T2=125 seconds after ascent initialization; and E23=sideslip sensor). Notice that the GCI is very close for the three fault cases shown but that the contributions from each weighting sub-index are quite different.

Table 9 Example of three fault sets identified as C1 with characteristic GCIs

Element	Fault Type	Fault Time	GCI	W_tBU	W_tPB	W_tMX	W_Hdist	W_EinfG	W_EinfC	W_E2
E22	F8	T3	1,1690	0,6831	0,0009	0,0003	0,3652	0,0521	0,0637	0,0037
E16	F4	T2	1,1511	0,7808	0,0028	0,0003	0,2997	0,0335	0,0278	0,0063
E23	F8	T2	1,1498	0,7505	0,0015	0,0003	0,3014	0,0539	0,0343	0,0080

An example of the graphical results provided is given by Fig. 7, which shows the control and guidance errors profiles for the second of the above critical faults (a dead sensor fault at time T2=125). The bottom plot shows the controller output under the no fault and fault cases. Note that the control errors, but not the guidance, violate their design objectives (very strongly, indicating C1 as automatically recognized by FAT).

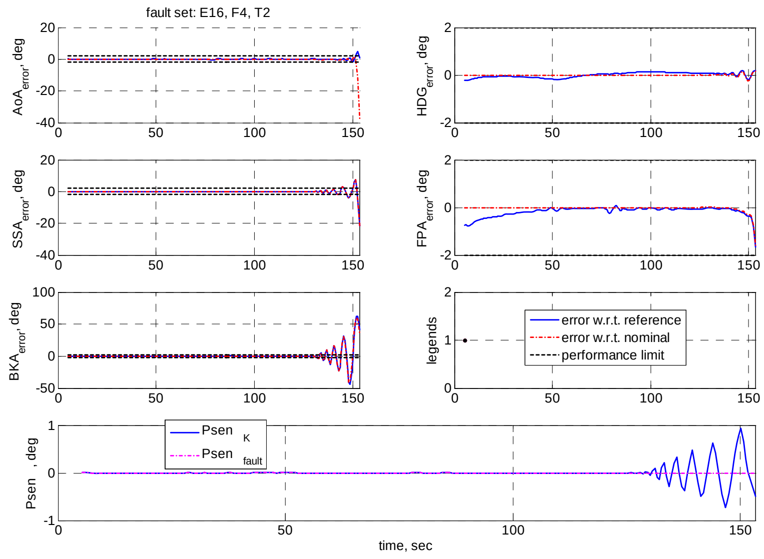


Fig. 7 Tracking error profiles for E16-F4-T2.

VI. Conclusion

In this article, a fault analysis methodology and associated software toolbox have been presented. The FA methodology developed is based on a mixed approach formed by computation-based data acquisition and quantitative-based data analysis. As opposed to current (FMECA-type) hazard analysis, the proposed methodology is oriented towards fault detection & isolation and fault tolerant control engineers. The FA methodology is reflected in a five-step process that considers: fault modelling, fault quantification metrics, fault classification taxonomy, simulation (data generation) and assessment (data analysis). A software fault analysis toolbox (FAT) has been developed in support of the methodology based on Matlab/Simulink and XML files and its application exemplified using results from a reusable launch vehicle fault analysis study. Special attention is given in developing the toolbox for its use within simulation tools and standards widely used nowadays in industry, as well as in the automated generation of results in an readily interpretable format to facilitate the analyst task.

Acknowledgments

We gladly thank and acknowledge the opportunity to apply the presented fault analysis methodology to the analysis of the Hopper Reusable launch vehicle within ESA's Health Management System (HMS) project, lead by Josef Sommer & Wolfgang Belau from EADS-Astrium Space Transportation (Germany) under supervision of Eric Bornschlegl ESA ESTEC technical officer.

References

[1] Boskovic J.D. and Mehra R.K., "Computer Simulations Analysis for Reconfigurable Flight Control Design," AIAA Guidance, Navigation and Control Conference, Paper AIAA-4787. August 2002.

[2] Boskovic J.D., Prasanth R.K. and Mehra R.K., "Reconfigurable Fault-Tolerant Flight Control: Algorithms, Implementation and Metrics," AIAA Guidance, Navigation and Control Conference, Paper AIAA-6549. August 2006.

[3] Shin J.Y., Belcastro C. and Khong T., "Closed-Loop Evaluation of An Integrated Failure Identification And Fault Tolerant Control System for a Transport Aircraft," AIAA Guidance, Navigation and Control Conference, Paper AIAA-6310. August 2006.

[4] Marcos, A. *Aircraft Application of Fault Detection and Isolation Techniques*. PhD. Thesis, Department of Aerospace Engineering and Mechanics, University of Minnesota, USA, February 2004.

[5] Balas, G.J., Doyle, J.C., Glover, K., Packard, A., Smith, R., "Mu-analysis and synthesis toolbox," June 1998. Musyn Inc. & The Mathworks.

[6] Niemann, H., Stoustrup, J., "Robust fault detection in Open Loop vs. Closed Loop", IEEE Conference on Decision and Control 1997.

[7] Marcos, A., De Zaiacomo, G., Peñín, L.F., Bornschlegl, E., "Fault Analysis for Robust FDI Design during RLV Ascent and Re-entry Phases," 7th International ESA Conference on Guidance, Navigation and Control Systems. Ireland, June 2008.

[8] Massoumnia, M.A. *A Geometric Approach to Failure Detection and Identification in Linear Systems*. PhD. Thesis, Department of Aeronautics and Astronautics, MIT. USA, February 1986.

[9] Sun Developer Network web-page: http://java.sun.com/developer/technicalArticles/xml/

[10] Jackson, B.E., Hildreth, B.L., "Flight Dynamic Model Exchange using XML," AIAA Guidance, Navigation and Control Conference, Paper AIAA-2002-4482. August 2002.

[11] Marcos, A., Peñín, L.F., Caramagno, A., Sommer, J., Belau, W., "Atmospheric Re-entry NDI Control Design for the Hopper RLV Concept," 17th IFAC Symposium on Automatic Control in Aerospace. Toulouse, France, June 2007.

[12] Leveson, N. G. *Safeware - System safety and computers*. Addison-Wesley. 1995.

[13] Izadi-Zamanabadi, R. *Lecture Notes – Practical Approach to Reliability, Safety and Active Fault-tolerance*. Aalborg University. Denmark, December 2000

[14] Belcastro, C.M., Belcastro, C.M., "On the validation of safety critical aircraft systems Part I: An overview of analytical & simulation methods," AIAA Guidance, Navigation, and Control Conference, Paper AIAA-5559. Austin, August 2003

[15] Buffington J.M. et al, "Validation and Verification of Intelligent and Adaptive Control Systems", 2nd AIAA Unmanned Unlimited Systems, Technologies and Operations. September 2003.

[16] ECSS-E-60A, *European Cooperation for Space Standardization: Space Engineering – Control Engineering,* September 2004.