An Integrated Methodology for the Performance and Reliability Evaluation of Fault-Tolerant Systems

by

Alejandro D. Domínguez-García

Ingeniero Industrial
Universidad de Oviedo, Spain (2001)

Submitted to the Department of Electrical Engineering and Computer Science in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 2007 [June 2007]

Author [illegible]
Department of Electrical Engineering and Computer Science
May 2007

Certified by [illegible]
John G. Kassakian
Professor of Electrical Engineering and Computer Science
Thesis Supervisor

Certified by [illegible]
Joel E. Schindall
Bernard M. Gordon Professor of Electrical Engineering and Computer Science
Thesis Supervisor

Accepted by [illegible]
Arthur C. Smith
Chairman, Department Committee on Graduate Students

An Integrated Methodology for the Performance and Reliability Evaluation of Fault-Tolerant Systems

by
Alejandro D. Domínguez-García

Submitted to the Department of Electrical Engineering and Computer Science
on May 2007, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

Abstract

This thesis proposes a new methodology for the integrated performance and reliability evaluation of embedded fault-tolerant systems used in aircraft, space, tactical, and automotive applications. This methodology uses a behavioral model of the system dynamics, similar to the ones used by control engineers when designing the control system, but incorporates additional artifacts to model the failure behavior of the system components. These artifacts include component failure modes (and associated failure rates) and how those failure modes affect the dynamic behavior of the component. The methodology bases the system evaluation on the analysis of the dynamics of the different configurations the system can reach after component failures occur. For each of the possible system configurations, a performance evaluation of its dynamic behavior is carried out to check whether its properties, e.g., accuracy, overshoot, or settling time, which are called performance metrics, meet system requirements. Markov chains are used to model the stochastic process associated with the different configurations that a system can adopt when failures occur. Reliability and unreliability measures can be quantified, as well as probabilistic measures of performance, by merging the values of the performance metrics for each configuration and the system configuration probabilities yielded by the corresponding Markov model. This methodology is not only used for system evaluation, but also for guiding the design process, and further optimization. Thus, within the context of the new methodology, we define new importance measures to rank the contributions of model parameters to system reliability and performance.

In order to support this methodology, we developed a MATLAB/SIMULINK® tool, which also provides a common environment with a common language for control engineers and reliability engineers to develop fault-tolerant systems. We illustrate the use of the methodology and the capabilities of the tool with two case-studies. The first one corresponds to the lateral-directional control system of an advanced fighter aircraft. This case-study shows how the methodology can identify weak points in the system design; and point out possible solutions to eliminate them; compare different architecture alternatives from different perspectives; and test different failure detection, isolation, and reconfiguration (FDIR) techniques. This case-study also shows the effectiveness of the MATLAB/SIMULINK® tool to analyze large and complex systems. The second case-study compares two very different solutions to achieve fault-tolerance in a steer-by-wire (SbW) system. The first solution is based on the replication of components; and the introduction of failure detection, isolation, and reconfiguration mechanisms. In the second solution, a dissimilar backup mechanism called brake-actuated steering (BAS), is used to achieve fault-tolerance rather than replicating each component within the system. This case-study complements the flight control system one by showing how the performance and MATLAB/SIMULINK® tool can be used to compare very different architectural approaches to achieve fault-tolerance; and therefore, how the methodology can be used to choose the best design in terms of performance and reliability.

Thesis Supervisor: John G. Kassakian
Title: Professor of Electrical Engineering and Computer Science

Thesis Supervisor: Joel E. Schindall
Title: Bernard M. Gordon Professor of the Practice

To Cristina and my parents, Vicenta and Ángel

Acknowledgements

This acknowledgments not only include the people that have been directly involved in my work for the last four years and a half, but also people who were instrumental in shaping me as an individual and indirectly make this moment happen.

First, I want to thank both my advisors John Kassakian and Joel Schindall for guiding me through my Ph.D years. One of the things that I most appreciate is the freedom they both gave to do my work and find my own way. I am grateful to John for believing in me by offering my the opportunity to work with him when I arrived to MIT. I am really grateful to Joel for being a real mentor; for teaching me all those skills, not written in any book, any researcher needs to manage his/her own research; for listening to me and give me excellent advise in all matters.

I want also to thank my thesis committee: Phil Babcock, George Verghese and George Apostolakis. Phil is the person that most supported me when I started working on fault tolerance and I did not know anything about it. He managed to find the time, and had the patience to teach me even before he was involved in my thesis. He saw a lot of potential in the work I was doing and managed to make Draper fund it for three consecutive years. Even if George Verghese was not directly advising me, I felt he was doing so several times. He found the time to sit week after week with me and discuss my research. He even allowed me to attend and present my work at his networks group meetings, which was very important at the time. Finally George Apostalakis, even if he is in a different department, did not hesitate on being in my committee and helping me out, not only in my research, but also when I was exploring my post-graduation options.

I am also in debt with Jeff Zinchuk at the Charles Stark Draper Laboratory. Although he is not in the thesis committee, he was the one that met with me week after week and discussed my work, and gave me feedback. He is being extremely involved with my Ph.D., not only technically but listening to me when I was going through hard times. There are other people at Draper that I also want to thank for supporting me in different matters. These are Rob Hammett, Nick Borer, Jean Avery and Piero Miotto.

Special thanks go to my great friend and colleague Ivan Celanovic. He was the "unofficial" third advisor. He was the one that listened to me in the toughest moments. He was the one to share with me the joyful moments and the crazy nights at Middlesex. In summary, he was a true mentor to me. I also want to thank my friends and colleagues at LEES, specially Laura Zager and Jeff Lang.

Thanks also to my friends for the great moments we shared. They have names: Nenad, Mara, Velimir, Ivana, Nebojsa, Mauro, Fred, Joe, Neal, Bonna, Eric, Ashley, Shawdee, Laura Humpreys. It has been the most exciting of my life and you contributed big to make them a lot of fun.

I would also like to acknowledge the sponsors that funded this research; the Charles Stark Draper Laboratory and the MIT/Industry Consortium on Advanced Electrical/Electronic Components and Systems. I also want to thank the Society of Presidential Fellows at MIT, who provided me with a fellowship for my first year in the Ph.D. program.

I also want to thank several people I worked with during my summer internships in Germany and England. First, I want to thank Marcus Abele, Wolfgang Mueller, Norbert Hoffmann, and the rest of the people at Bosch; it was a real pleasure to work with you. Second, I want to thank Peter Miller and Ben Sewell at Ricardo. Some of the ideas of Chapter 4 were developed in collaboration with them. Finally, I want to thank Doug Milliken at Milliken Research Associates for being so kind and helpful when I was learning about vehicle dynamics. He provided us with a free licence of their vehicle dynamics MATLAB® simulator. We did not end up going in that direction, but it was really fun to learn all those things about vehicle dynamics.

I am also in debt with two people that were instrumental in making this moment happen. The first one is Javier Gómez-Aleixandre Fernández. He was the one that made me feel passionate about electric engineering, a great professor, a great educator, a great friend and a great mentor. The second person is Miguel Ángel Sanz Martínez. He was the one that really encouraged me to apply to MIT in the first place. He was the one that made me see it was possible and helped me to make it happen.

My family deserve more than a simple "thank you". Specially my parents, Vicenta and Ángel, who provided me with a great education, who supported me in all my endeavors and adventures: from buying me my first guitar to support me in the decision of emigrating to the USA for continuing my education and for pursuing my career. My brothers Juan, Ernesto, Vicente, and Victor; and sister Elena were fundamental in my education. Being the youngest I always looked at them as models to follow.

The final mention if for my fiancée Cristina, who has being a continuous support for me during the last four years. Thanks for all the things we shared. Thanks for all the things that we will we sharing very soon. All my love to her.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In this introductory chapter, we state the necessity of developing a new methodology for analyzing the reliability and performance of fault-tolerant systems used in aircraft, space, tactical, and automotive applications. In order to put this research in the appropriate context, a review of classical reliability evaluation methodologies and tools is provided, as well as some work relevant to this research. To facilitate the reading process, and highlight the main contributions of this thesis, the chapter ends with a summary of the research, in which the structure of the document and the main contributions of each chapter are provided.

1.1 Problem Statement

The safety-critical/mission-critical nature of embedded systems used in aircraft, space, tactical, and automotive applications, mandates that the systems' functionality for which they are designed be performed even in the presence of component failures [1]. Thus, safety-critical/mission-critical systems must have the capability to adapt and compensate for component failures in a planned, systematic way [3]. These types of adaptable systems are known as fault-tolerant systems[2].

Designing an effective fault-tolerant system requires a through and comprehensive analysis to fully understand and quantify potential failures and assess the effectiveness of failure

[1]According to Laprie [4], a failure occurs when the delivered service no longer complies with the agreed description of the component's expected function and/or service.

[2]The concept of fault-tolerance was originally formulated by Avižignis in the field of computers [5]. However, the concept of fault-tolerant systems is much broader than the computer field. There are several examples of fault-tolerant systems, e.g., aircraft, aerospace, defense, in which computers are just part of the system, but there are other equally important components or subsystems, e.g., actuators, sensors, valves, or generators. Therefore, we prefer the definition given in [3], which states fault-tolerance as the ability of a system to adapt and compensate in a planned, systematic way to random failures of their components that can cause a system failure.

detection, isolation, and reconfiguration (FDIR) mechanisms. There are well-developed techniques to support the reliability [3] evaluation of conventional systems (see section 1.2). However, all these techniques can yield ambiguous and/or incomplete results as they base the analysis on a qualitative description of the system's functionality. This functional description describes how components and subsystems are interconnected to fulfill the functions they are designed for, and may include how component failures can propagate to other components through their interconnections and affect the system functionality [6, 7, 8]. For conventional systems, this approach is valid, as it is possible to evaluate how a system can fail to perform the function for which it was designed by using this high-level qualitative system's functionality description. However, this is not the case for *large-complex* systems or embedded *software-intensive* systems.

The introduction of software to control engineering systems results in increased flexibility in the design of fault-tolerant systems. With this flexibility it is possible to perform a complete system reconfiguration to accommodate failures, i.e., not only compensate for hardware failures by substituting for the failed elements, but also by reconfiguring the control software [4]. However, as pointed out by Leveson in [9], the introduction of software in engineering systems, and the large numbers of components and subsystems (and the interconnections among these) has greatly increased the complexity of engineering systems. Thus, for *large-complex* systems or embedded *software-intensive* systems, typical of fault-tolerant systems, the additional complexity introduced by:

- the software controlling (and reconfiguring) the system; and
- the large number of interconnected components

makes it nearly impossible to fully understand the system performance –and thus determine its reliability, in the presence of hardware failures or software malfunctions, by only using a qualitative description of the system's functionality and components' failure behavior.

[3]There are different ways to define reliability. We will use the one given in [10], which states reliability as the ability of an item to perform a required function under stated conditions for a stated period of time.

[4]To illustrate this, on its return to Earth on May 4th, 2002 from the International Space Station (ISS), a Soyuz TMA-1 spacecraft recovered from a module failure by reconfiguring itself to perform a backup re-entry maneuver known as ballistic mode [11]. The problem, which caused the Soyuz TMA-1 to perform the re-entry in ballistic mode was a failure in the BUSP-M guidance system, which is necessary in order to carry out a controlled re-entry. The system responded to this failure by engaging higher control functions to take the BUSP-M system out of the control loop and convert to an entirely different (ballistic) re-entry mode.

Therefore, the main goal of this thesis is to develop a new methodology to analyze the performance and reliability of fault-tolerant systems; minimizing the subjectivity introduced in the system analysis due to incompleteness of current modeling techniques. Thus, rather than using a qualitative description of the system's functionality, this methodology uses a model of the system dynamics plus additional features to model component failure behavior. These features include component failure modes (and associated failure rates) and how these failure modes affect the behavior of the component. All reliability-related evaluation activities will be based on this quantitative system behavioral model, thus reducing the possible ambiguity that always arises when using qualitative models to analyze system reliability.

1.2 Background and Related Work

The *IEEE Standard Dictionary of Electrical and Electronics Terms* defines reliability as the ability of an item to perform a required function under stated conditions for a stated period of time [10]. For a conventional system, with no redundancy or backup mechanism to account for component failures, it is natural to think that the system will deliver its function only if all the components are operational. This is not the case in a fault-tolerant system, which implements additional components and subsystems to account for component failures, thus delivering its function despite the presence of certain failures. Therefore, in a fault-tolerant system, reliability will be dictated by the combinations of components at any time. Furthermore, the ordering in which these combinations of failed components is achieved may be important in some systems. Thus, the sequences of component failures leading to system failure give a qualitative measure of system reliability. However, if a quantitative measure of system reliability is required, knowing only the sequences of component failures leading to system failure is not enough. Since the failure behavior of single components is often modeled as a random process, i.e., the time to failure of the component is regarded as random [6], it is natural to quantify system reliability as the probability of the system delivering the function for which it was designed, for a period of time, e.g., system lifetime.

1.2.1 State-of-the Art in Probabilistic System Reliability Evaluation

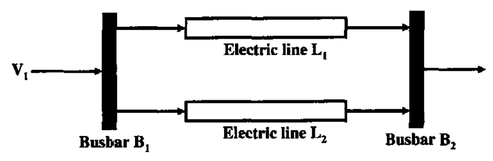
In this section, we present a thorough literature review of probabilistic system reliability evaluation techniques; highlighting the attractive features of each technique to model fault-tolerant systems reliability; but also pointing out its shortcomings. The use of each technique is illustrated with an example. Additionally, the use of Markov chains for reliability modeling is further illustrated with a case-study in Chapter 2. Discussions of advanced reliability evaluation techniques are also included, some of which are very relevant to this research. This is the case for a technique called Dynamic Probabilistic Risk Assessment (DPRA), as we will further detail in Chapter 3, when we introduce the mathematical framework of the methodology proposed in this thesis for the integrated performance and reliability evaluation of fault-tolerant systems.

1.2.1.1 Reliability Block Diagrams (RBD)

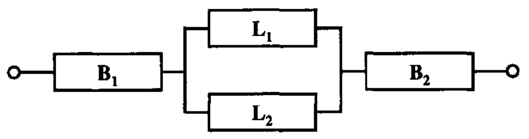
System Model. This is a graphical representation of the system's structure function[5] in the form of a success-oriented network [6]. Thus, each component within the system is represented by a two-terminal box. The components are connected through their terminals to form a two-terminal network. Each component can be failed or non-failed. If the component is non-failed then there is path between its two terminals, otherwise there is no path. The system will fulfill its function (in the presence of component failures) whenever there is a path between the two terminals of the network. If the system has more than one function, and the necessary components to fulfill each function are different, then separate RBDs will be developed for each function.

To illustrate how an RBD is developed, consider the parallel circuit for energy transmission displayed in Fig. 1.1(a). Let's assume that the system function is to deliver a constant output voltage V_2 for an input voltage V_1. Let's assume that the components can only fail open circuit. Then for the system to deliver its function, the existence of a path between the circuit ends is necessary. Thus, the two electric lines L_1 and L_2 will be represented in parallel in the RBD, while the two busbars B_1 and B_2 will be represented as series components. The resulting RBD is displayed in Fig. 1.1(b). Note that in this case, the

[5]The system structure function $\phi(x_1, x_2, \dots, x_N)$ is a binary function, where each x_i is an indicator variable that takes value 0 when the component failure event associated with x_i occurred and 1 otherwise (the associated component did not fail). The system structure function $\phi(x_1, x_2, \dots, x_N)$ will take value 1 whenever the system is functional for a given $(x_1, x_2, \dots, x_N)$ and zero whenever the system failed.



(a) Architecture functional description.



(b) RBD for voltage V_2 delivery.

Figure 1.1: Energy transmission circuit system functional description and reliability block diagram model.

resulting RBD has a structure similar to the system functional description. This is usually the case when modeling reliability of static networks, e.g., electric circuits with a fixed structure.

Model Evaluation. System reliability is obtained by quantifying the probability of existence of a path between the RBD ends. As mentioned before, an RBD is a graphical representation of the system's structure function. Thus, in order to build the structure function associated with the system RBD, it is necessary to first compute the minimal cut-sets[6]. For the parallel circuit RBD example, Fig. 1.1(b), the minimal cut-sets are: $\{B_1$ fails open circuit$\}$, $\{B_2$ fails open circuit$\}$, and $\{L_1$ fails open circuit, L_2 fails open circuit$\}$. The resulting structure function is given by:

$$\phi(x_1, x_2, x_3, x_4) = x_1 x_2 (x_3 + x_4 - x_3 x_4) \tag{1.1}$$

where x_1 is the indicator variable associated to the $\{B_1$ fails open circuit$\}$ event, x_2 is the indicator variable associated to the $\{B_2$ fails open circuit$\}$ event, x_3 is the indicator variable associated to the $\{L_1$ fails open circuit$\}$ event, and x_4 is the indicator variable associated to the $\{L_2$ fails open circuit$\}$ event. Then, the system reliability R can be computed as the expectation of the structure function [6]. By assuming independence of the simple events:

$$\begin{gathered} R = \mathbb{E}[\phi(x_1, x_2, x_3, x_4)] = \\ P(x_1 = 1)P(x_2 = 1)P(x_3 = 1) + P(x_1 = 1)P(x_2 = 1)P(x_4 = 1) - \\ P(x_1 = 1)P(x_2 = 1)P(x_3 = 1)P(x_4 = 1). \end{gathered} \tag{1.2}$$

If the time to occurrence of each simple event is considered to be exponentially distributed, then the system reliability can be computed as a function of time t. Let λ_{x_i} be the rate of

[6]A minimal cut-set is defined as a minimal-size set of simple events that cuts any path between the RBD ends [6]. Minimal-size implies that any set resulting from eliminating any simple event of the original cut-set is no longer a cut-set.

occurrence for event $i = 1, 2, \ldots 4$, then

$$P(x_i = 1) = 1 - e^{\lambda_{x_i} t}. \tag{1.3}$$

By substituting the expression for each component yielded by (1.3) into (1.2), an expression for the system reliability R as a function of time is obtained.

Shortcomings. RBDs are static structures, i.e., there are no time-dependencies between events within the RBD. Thus, an RBD is a snapshot of the components that must be operational at any time for the system to deliver its function. Therefore, this means that RBDs are not suitable to model systems where the order of component failure is important, i.e., at a given time, two combinations of failed components which are reached in different order may result in two different outcomes.

To illustrate this, consider the parallel parallel circuit for transmitting energy displayed in Fig. 1.2. In the system failure-free operational conditions, Electric line L_1 is engaged; and Electric line L_2 is disengaged. When the voltage sensor VS detects a voltage drop, then it sends a command to the Switch SW, and then Electric line L_1 is disengaged and Electric line L_2 engaged. Now, let's consider two different sequences of component failures involving the same events. Let's consider that the system is working in its nominal operation conditions and the Electric line L_1 fails open circuit first. Then, the voltage sensor will detect a voltage drop, and send a signal to the switch SW which will disengage Electric line L_1 and engage Electric line L_2. After Electric line L_2 is in operation, let the voltage sensor VS fail to detect voltage drops. In this case, since the system is already operating with Electric line L_2, the sensor failure will not affect the energy transmission. Now consider the inverse order in the failure events: the system is operating with Electric line L_1 the voltage sensor VS fails first to detect voltage drops. In this scenario, the system will keep its operation with Electric line L_1, but no voltage drop detection is possible. Then let the Electric line L_1 fail open circuit. In this conditions, the switch does not receive any voltage drop warning from the sensor VS since this one is already failed. Thus the Electric line L_2 will not be engaged and thus the system will fail to transmit energy. It is clear that in this case, the different events-ordering result in very different outcomes.

RBDs also can not capture other important features that may be important for modeling fault-tolerant systems. It is not possible to model repair processes that may be triggered to replace some components that may have failed without causing the system to fail. RBDs are also not suitable to model failure coverage. Failure coverage refers to the ability of a fault-tolerant system to recover from a component failure and stay operational. There are

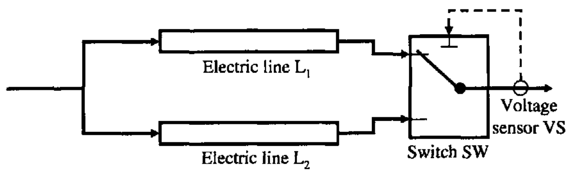


Figure 1.2: Electric energy transmission system fault-tolerant architecture in its failure-free operational mode.

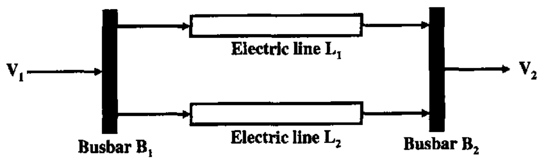
cases in which there is no uncertainty on the system recovery after a failure occurred, i.e., the system will always recover (perfect-coverage) or it never recovers (no-coverage). However, there might be cases in which depending on the system operational conditions at the time of failure, the system may recover or may not, i.e., there is uncertainty (imperfect-coverage) with respect to whether the system will survive the failure. RBDs can handle perfect-coverage or no-coverage situations. However, they are not suited to modeling systems with imperfect-coverage situations. Finally, state-dependent failures are another important feature of fault-tolerant conditions, i.e., the likelihood of certain component failures depends on the the components that are already failed. RBDs can not handle this important feature either.

Despite these limitations, RBD modeling is an appropriate technique to assess the reliability of networks with static structures[7], in which each component can only have two states: operational or failed. In this sense, RBDs are appealing because, unlike fault-trees (as we show next), the RBD structure resembles the network physical structure. Thus, it is easy to understand the system failure behavior when components have failed, and furthermore, it is possible to automatically build the RBD from the functional block diagram description.

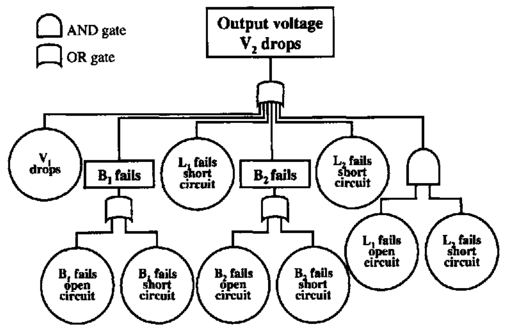
1.2.1.2 Fault-Trees

System Model. This is a graphical representation of the system's structure function in the form of a fault/success-oriented logical diagram [6]. Thus, the top event of the tree represents the conditions that must be met in order for the system to fail/not-fail to deliver the function for which it was designed. The events that lead to the top event occurrence are obtained through combinations of simple events (component failures), and are graphically represented in the tree using logical AND and OR logical gates.

[7]Pioneering work on developing reliable electrical networks was published in 1956 by Moore and Shannon [12].



(a) Architecture functional description.


AND gate
OR gate
Output voltage V_2 drops
V_1 drops
B_1 fails
L_1 fails short circuit
B_2 fails
L_2 fails short circuit
B_1 fails open circuit
B_1 fails short circuit
B_2 fails open circuit
B_2 fails short circuit
L_1 fails open circuit
L_2 fails short circuit

(b) Fault-tree for voltage V_2 drop top-event.

Figure 1.3: Fault-tolerant electric energy transmission system functional description and fault-tree.

To illustrate how a fault-tree is built, consider the same parallel circuit used to illustrate the construction of an RBD, Fig. 1.3(a). Let's assume that the system function is to deliver a constant output voltage V_2 for an input voltage V_1. Now, we assume that each element within the circuit can fail either short or open circuit. Under these assumptions, the resulting fault-tree is displayed in Fig. 1.3(b).

Model Evaluation. System reliability is obtained by quantifying the probability of occurrence of the the fault-tree's top-event. The simple events (component failure events) are assumed to be independent events. Thus, the calculation of the top-event's probability boils down to computing the probability of unions and/or intersections of simple events. However, for large fault-trees, this can be a daunting and error-prone task. Thus, there are techniques that simplify the calculation of the top-event probability. One of these techniques consist on computing the minimal cut-sets[8]. To illustrate this, let's compute the minimal cut-sets for the parallel circuit example, Fig. 1.3(b). In this example, the cut-sets are: $\{V_1$ drops$\}$, $\{B_1$ fails open circuit$\}$, $\{B_1$ fails short circuit$\}$, $\{B_2$ fails open circuit$\}$, $\{B_2$ fails short circuit$\}$,$\{L_1$ fails short circuit$\}$, $\{L_2$ fails short circuit$\}$, $\{L_1$ fails open circuit, L_2 fails open circuit$\}$.

[8]In Fault-Trees, a minimal cut-set is defined as a minimal-size set of simple events that leads to the occurrence of the top-event [6].

The resulting structure function is given by:

$$\phi(x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9) = x_1x_2x_3x_4x_5x_6x_7(x_8 + x_9 - x_8x_9), \tag{1.4}$$

where x_1 is the indicator variable associated with the $\{V_1$ drops$\}$ event, x_2 is the indicator variable associated with the $\{B_1$ fails open circuit$\}$ event, x_3 is the indicator variable associated with the $\{B_1$ fails short circuit$\}$ event, x_4 is the indicator variable associated with the $\{B_2$ fails open circuit$\}$ event, x_5 is the indicator variable associated with the $\{B_2$ fails short circuit$\}$ event, x_6 is the indicator variable associated with the $\{L_1$ fails open circuit$\}$ event, x_7 is the indicator variable associated with the $\{L_1$ fails short circuit$\}$ event, x_8 is the indicator variable associated with the $\{L_2$ fails open circuit$\}$ event event, and x_9 is the indicator variable associated with the $\{L_2$ fails short circuit$\}$ event. As mentioned before, each of these indicator variables will take the value 0 whenever the event occurs, and 1 otherwise. Then, the system reliability R can be computed as the expectation of the structure function. As mentioned before, by assuming independence of the simple events:

$$\begin{gathered}
R = \mathbb{E}[\phi(x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9)] = \\
P(x_1 = 1)P(x_2 = 1)P(x_3 = 1)P(x_4 = 1)P(x_5 = 1)P(x_6 = 1)P(x_7 = 1)P(x_8 = 1) + \\
P(x_1 = 1)P(x_2 = 1)P(x_3 = 1)P(x_4 = 1)P(x_5 = 1)P(x_6 = 1)P(x_7 = 1)P(x_9 = 1) - \\
P(x_1 = 1)P(x_2 = 1)P(x_3 = 1)P(x_4 = 1)P(x_5 = 1)P(x_6 = 1)P(x_7 = 1)P(x_8 = 1)P(x_9 = 1).
\end{gathered} \tag{1.5}$$

As for RBDs, if the time to occurrence of each simple event is considered exponentially distributed, then it is possible to compute the system reliability as a function of time t.

Shortcomings. One advantage of fault-trees is that, unlike in RBDs, different failure modes can be considered for the same component. However, fault-trees have shortcomings similar to those exposed for RBDs:

- Fault-trees are static structures. Therefore, fault-trees are not suitable for modeling systems where the order of component failure is important.
- Fault-trees are not suited to modeling repair processes.
- Fault-trees can not handle imperfect failure coverage.

Another shortcoming of fault-trees (and in general of any classical reliability evaluation technique) is the difficulty of generating the fault-tree in a systematic and objective way from the functional description of the system and the component failure description. Quoting from [13] "Fault-tree has at present, though, a number of drawbacks. It is an art, rather than a science. If two people analyze a system, the results are never the same". Furthermore, as pointed out in the introduction, for very large complex systems, it is impossible to understand how a system behaves in the presence of component failures by using only a description of the system functionality. This makes the development of the fault-tree even more difficult.

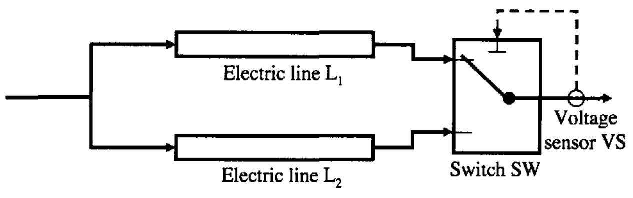
Advanced Fault-Tree Techniques

In the remainder of this section, several advancements in fault-tree analysis techniques will be presented. These techniques solve some of the shortcomings mentioned above. However, there are still several important features of fault-tolerant systems that these techniques cannot handle.

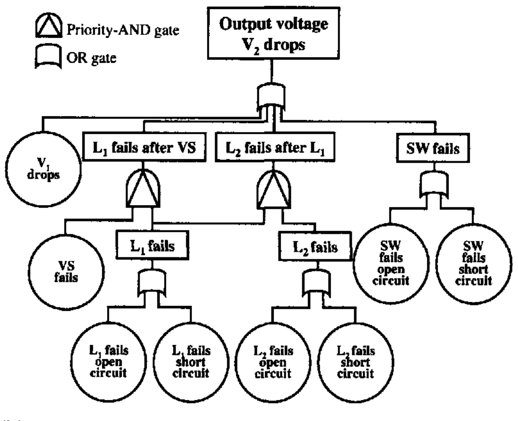
Dynamic Fault-Trees. Priority logical gates were introduced in order to solve the event time-dependency problem [14] , which conventional logical gates can not handle. Later on, logical gates were incorporated into fault-trees for the analysis of fault-tolerant computer system [15], and thus the concept of a dynamic fault-tree was introduced [16]. The dynamic fault-tree solved the problem of reliability modeling of systems where time-dependencies of simple events are relevant.

To illustrate the use of dynamic fault-trees for reliability modeling, let's consider the same parallel circuit example presented to show the inability of conventional fault-trees to handle sequence-dependencies. The system functional description is displayed in Fig. 1.4(a). As explained before, the system is operating with electric line L_1 in nominal conditions, when L_1 fails (open or short circuit), the voltage sensor VS will detect a voltage drop in V_2 and will switch over the operation to electric line L_2. The switch SW can fail open or short circuit, causing the system to fail. The voltage sensor can fail to detect voltage drops in V_2. As mentioned before, if the voltage sensor VS fails when electric line L_1 is in operation, a subsequent failure of L_1 will not be detected, causing the system to fail. On the contrary, if the voltage sensor VS fails after the line L_1 failed, and operation was already switched to line L_2, the system will remain operational. The resulting dynamic fault-tree to describe the system behavior in the presence of failures is displayed in Fig. 1.4(b).

There is a new element in this model that enables sequence-dependence modeling —the priority-AND gate [14]. This logical gate will yield a true value if all the inputs occurred in the order they go into the gate, starting from left to right. Several other priority logical



(a) Architecture functional description.



(b) Dynamic fault-tree for voltage V_2 drop top-event.

Figure 1.4: Fault-tolerant electric energy transmission system functional description and dynamic fault-tree.

gates (functional-dependency gate, cold-spare gate, sequence-enforcing gate) were introduced in [16], allowing other more complex sequence-dependencies to be modeled.

Although dynamic fault-trees solve the sequences-dependency problem, there are still several features of fault-tolerant systems that dynamic fault-trees can not properly handle, e.g., repair processes, failure coverage, and state-dependent failures. Furthermore, unlike fault-trees, which have algorithms to directly quantify the top-event probability, there are no direct evaluation techniques for dynamic fault-trees. It is necessary to first convert the dynamic fault-tree into an equivalent Markov model, and then solve this associated Markov model to compute the top-event probability [17].

Hierarchically Performed Hazard Origin and Propagation Studies (HiP-HOPS). HiP-HOPS is a methodology that allows the development of fault-trees from architecture functional block diagrams in which local failure models for each component within the system are defined [8]. It also supports system Failure Modes and Effects Analysis (FMEA) automatic construction [18]. The component failure models are defined as collections of logical expressions that relate deviations of the component output (called output deviations) to deviations of the component input nominal behavior (called input deviations) and component internal failures (called basic events). The architectural connections between components (data connections, energy connections) allow the interconnection of the logical expressions of each local failure model, and thus, the development of a collection of system fault-trees (or a system FMEA). The top-event of each fault-trees that HiP-HOPS generates corresponds to a component output deviation.

The HiP-HOPS methodology is conceptually very similar to previous work developed by Taylor in the context of electronic systems and software [13, 19]. Taylor also attempted to automatically build fault-trees from a set of "transfer statements"; each of them describing how a component output event can result from a component internal change (a failure), or from an input event passed by another component output event [19]. This clearly indicates the similarity between HiP-HOPS and the work by Taylor.

Although HiP-HOPS (and the seminal work of Taylor) attempts to fill the gap between the system functional description (augmented with component failure descriptions) and the construction of the fault-tree, it still has several shortcomings. In these methodologies, the output deviations due to input deviations or internal failures are based on the logical expressions defined in the component local failure model. This failure model is based on the analyst's skills in understanding the component failure behavior, and therefore it is subjective. One peculiarity of the HiP-HOPS (and Taylor's) approach is that when two components are connected, the output deviations of the first component must match the input deviations of the second component [19, 20], which requires caution when defining the local failure models and limits the input deviations of a component to the set of output deviations of the previous component for cascading connections.

HiP-HOPS (and Taylor's approach) has the same problem as any other methodology based on static failure models, it can not capture failure sequences. This is easy to understand since the resulting failure model it is either a fault-tree (or an FMEA built from a fault-tree), and fault-trees can only capture combinations of component failures as mentioned before. Finally, both methodologies allow the component output deviations to be related to simple event occurrences. However, it is still the task of the analyst to determine if those deviations can cause the system to totally fail to perform its function, or to just cause a degraded system performance which is acceptable in some situations. Thus, these methodologies do not allow us to understand the overall system performance in the presence of component failures.

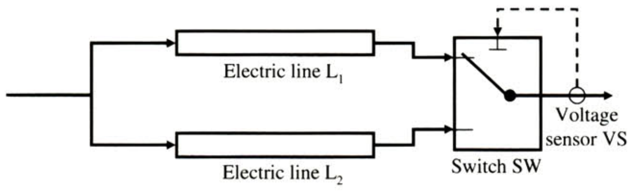
1.2.1.3 Markov Models

Among all the mathematical formalisms to model random process, continuous time Markov chains has proven to be a powerful approach to quantify the reliability of fault tolerant systems. This is due to the fact that many important features of fault-tolerant systems are naturally captured by a continuous-time Markov chain [21, 22]. Examples are: sequences of failures in which the order of component failures matters, different repair strategies, failure coverage, common mode failures, and state-dependent failure rates. In Chapter 2 the Markov reliability modeling principles presented in this section are illustrated with a case-study. A brief introduction to continuous-time Markov chains (CTMC) is provided in Appendix A. The reader is referred to [23, 24] for a more rigorous treatment of Markov processes.

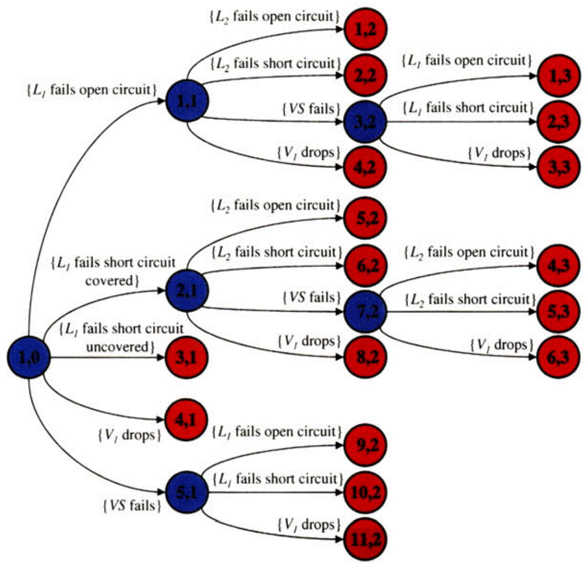
System Model. This is a graphical representation, in the form of a state-transition diagram, of the system status (failed or operational) for each system configuration reached after a unique sequence of component failures. The system is said to be in its failure-free configuration when all the components within the system are non-failed. The system can evolve from the failure-free configuration to other configurations depending on the status (failed or operational) of the components within the system.

The nodes of the state-transition diagram represent the status (operational or failed) of each system configuration, and the edges represent transitions between configurations triggered by component failures (or repair processes). There are two types of nodes: absorbing nodes and non-absorbing nodes. The system will fulfill its function whenever it is in a non-absorbing node. The system will fail to deliver the functions for which it was design whenever it transitions to an absorbing node. Even if the system must deliver more than one function, only one model is necessary. In this case, the non-absorbing states are associated with system configurations in which maybe one of the functions is not available, but the system is still operational. If the system is to operate in different mission phases, then several several state-transition diagrams are to be developed; one per mission phase.

We will use the same parallel circuit used before, Fig. 1.5(a), to illustrate two of the most important capabilities of Markov reliability models for modeling fault-tolerant systems: sequence-dependencies and failure coverage. The state-transition diagram corresponding to the Markov reliability model is displayed in Fig. 1.5(b). The absorbing states (displayed in red) correspond to system configurations declared as failed, while the states displayed in blue (transient states) correspond to the system configurations declared as non-failed.



(a) Architecture functional description.


{L_1 fails open circuit}
{L_2 fails open circuit}
{L_2 fails short circuit}
{VS fails}
{V_1 drops}
{L_1 fails open circuit}
{L_1 fails short circuit}
{V_1 drops}
{L_1 fails short circuit covered}
{L_2 fails open circuit}
{L_2 fails short circuit}
{VS fails}
{V_1 drops}
{L_2 fails open circuit}
{L_2 fails short circuit}
{V_1 drops}
{L_1 fails short circuit uncovered}
{V_1 drops}
{VS fails}
{L_1 fails open circuit}
{L_1 fails short circuit}
{V_1 drops}
1,0
1,1
2,1
3,1
4,1
5,1
1,2
2,2
3,2
4,2
5,2
6,2
7,2
8,2
9,2
10,2
11,2
1,3
2,3
3,3
4,3
5,3
6,3

(b) State-transition diagram for the function of delivering a constant V_2.

Figure 1.5: Fault-tolerant electric energy transmission system functional description and state-transition diagram.

Note that sequence-dependencies are naturally included: states $\{3,2\}$[9] and $\{9,2\}$ are two different sequences of the same failed components failed, with different results; the same applies to states $\{7,2\}$ and $\{10,2\}$.

As mentioned before, failure coverage refers to the ability of a fault-tolerant system to recover from a component failure and stay operational. There might be cases in which, depending on the system operational conditions, the system may recover or not, i.e., there is uncertainty (imperfect-coverage) with respect to whether the system will recover or not after a failure has occurred. In this case, for the $\{L_1$ short circuit$\}$ event, it may be the

[9]Each state of the Markov model is labeled with a double index $\{i,k\}$, where $k = 0, 1, \ldots$ represents the number of component failures, and $i = 1, 2, \ldots$ is a counter for the Markov states reached after sequences of failures of size k.

case that the short circuit path is not a direct contact to ground (there is some resistor in the path). It may happen that the voltage sensor VS is not able to detect this failure, and depending on the system operational conditions, this may cause the system to fail to deliver its functionality. This has an impact on the system reliability model; when the line L_1 fails short circuited, it will generate two states in the state-transition diagram instead of one. The first of these states $\{2,1\}$ will correspond to the case where L_1 fails short circuited and the system recovers from this failure. The second one $\{3,1\}$ corresponds to the uncovered short circuit failure of L_1, which results in a system failure.

Model Evaluation. System reliability R is quantified by calculating the probability that the system will be in any of the non-absorbing states at a given time given that the system was in the failure-free state $\{1,0\}$ at time $t = 0$ with probability 1. The transitions between configurations occur stochastically and are triggered by random failures within the system components. Each of these failures have an associated coverage probability. We assume the system evolves between configurations in a Markovian fashion, i.e., the system configurations (dictated by component status) at future times will only depend on the configuration at the present time. Therefore the Chapman-Kolmogorov equations (see Appendix A) can be used to compute the configuration status probabilities.

To illustrate the evaluation process, let's build the Chapman-Kolmogorov equations corresponding to two of the states of the state-transition diagram of Fig. 1.5(b). Let $\lambda_{L_{1(2)}}^{OC(SC)}$ be the failure rate for open circuit (short circuit) failure mode of component $L_1(L_2)$. Let λ_{VS} be the failure rate for the component VS, and λ_{V_1} the rate associated with voltage drops in V_1. Let $c_{L_1^{SC}}$ be the coverage probability of L_1 short circuit failure. Then, the Chapman Kolmogorov equations for states $\{1,0\}$ and $\{2,1\}$ are given by

$$\frac{dp_{1,0}(t)}{dt} = -(\lambda_{L_1}^{OC} + \lambda_{L_1}^{SC} + \lambda_{V_1} + \lambda_{VS})p_{1,0}(t), \qquad (1.6)$$

$$\frac{dp_{2,1}(t)}{dt} = c_{L_1^{SC}}\lambda_{L_1}^{SC}p_{1,0}(t) - (\lambda_{L_2}^{OC} + \lambda_{L_2}^{SC} + \lambda_{V_1} + \lambda_{VS})p_{2,1}(t), \qquad (1.7)$$

where $p_{1,0}(t)$ is the probability that no failure has occurred at time t, given that $p_{1,0}(0) = 1$, and $p_{2,1}(t)$ is the probability that the system is operating with electric line L_2 at time t (due to the fact that L_1 failed covered), given that $p_{1,0}(0) = 1$. Similar differential equations can be obtained for the remaining states of the Markov model. Then, after solving the resulting set of differential equations, the system reliability R at time t can be computed as the probability of being in a non-absorbing state of the Markov model:

$$R(t) = p_{1,0}(t) + p_{1,1}(t) + p_{2,1}(t) + p_{5,1}(t) + p_{3,2}(t) + p_{7,2}(t). \qquad (1.8)$$

Shortcomings. Markov reliability modeling allows us to include, in a natural and intuitive way, other important features of fault-tolerant systems, such as different repair strategies, failure coverage, common mode failures, and state-dependent failure rates when these are constant. However, there are some limitations in the time-varying case. For example, let us consider a component with different failure rates when in standby and when operational. We will assume that the failure rate when the component is operational increases with time; and that its initial value depends on the time the components is brought on-line. Thus, the configuration status probabilities depend not only on the time the system is operational, but also on the time components remain operational, i.e., two different time scales. The effect of this additional time scale violates the Markov property, i.e., visits to future states do not depend on visits to past states, only on the state visited at the present time. Therefore, Markov models cannot be used to model component state-dependent time-varying failure rates [25].

Another important drawback to the use of Markov reliability modeling is that the state space grows exponentially with the number of components. Therefore, for large fault-tolerant systems, it is necessary to apply model truncation techniques to control the state space explosion [21, 26]. However, when truncating the analysis, it is necessary to assess the impact of the truncation on the system reliability and unreliability estimates.

A technique to truncate the analysis consists of assuming that all system configurations with k_{max} or more failed components are declared as failed. Upper and lower bounds in the system reliability R, and unreliability Q, can be obtained by using [27]:

$$\hat{Q}_{k_{max}} < Q \leq \hat{Q}_{k_{max}} + p_{a,k_{max}}, \tag{1.9}$$

$$\hat{R}_{k_{max}} \leq R < \hat{R}_{k_{max}} + p_{a,k_{max}}, \tag{1.10}$$

where $\hat{Q}_{k_{max}}$ is the probability of being in an absorbing state associated with a configuration with less than k_{max} components failed; $\hat{R}_{k_{max}}$ is the probability of being in a non-absorbing state associated with a configuration with less than k_{max} components failed; and $p_{a,k_{max}}$ is the probability of the system having exactly k_{max} components failed, which corresponds to the probability of being in an absorbing state reached from all the non-absorbing states associated with configurations with $k_{max} - 1$ components failed. The lower bound in (1.9) is never achieved; the system will fail eventually for a sequence of component failures of size bigger than k_{max}. A similar conclusion can be reached for the upper bound in (1.10), which can never be achieved.

Finally, the main shortcoming of Markov reliability modeling (true for most reliability evaluation techniques, as mentioned before) is the difficulty of generating the Markov model from the functional description of the system and the component failure descriptions. For large complex systems, this can be a daunting and error-prone task. Chapter 2 introduces a technique to overcome this problem by building several system level FMEAs, which collect the different failure sequences and their effects on the system functionality. This gives a systematic solution for exploring all possible sequences of failures and their effects on the system operation. However, it is still a manual task.

Advanced Markov Modeling Techniques: Computer Aided Markov Evaluator (CAME)

In the remainder of this section, we present a technique that addresses the main problem of Markov reliability modeling techniques — the gap between the system functional model and the reliability model. It was developed in the form of a computer tool, named the Computer Aided Markov Evaluator (CAME), at the Charles Stark Draper Laboratory in the mid '80s [27, 28]. The tool was developed to analyze the reliability of fault-tolerant systems encountered in avionics and space applications, but it can be used to analyze the reliability of any engineering system.

Although the name might suggest that CAME is merely a computer tool to evaluate Markov models, CAME is the computer implementation of an advance methodology to evaluate the reliability of complex fault tolerant systems. This methodology automatically generates a Markov reliability model from the following:

- A description of the components within the system. For each component, it includes: its failure rate (the component is only regarded as failed or non-failed), the component failure coverage probability, and the component operational dependencies, i.e., which other components or subsystems within the system must be operational for the component to be operational.
- A system operational description, which captures the components that must be functional at any time for the system to be operational, and how the system reconfigures itself to account for component failures, thus modifying the system operational description.

The methodology implemented in CAME tried to fill the gap between the system functional description and the system reliability model —the same way as the advanced fault-tree techniques previously discussed. One of the advanced features introduced with CAME is the automatic modeling of system reconfigurations, which are extremely important in fault-tolerant systems. However, there are still several problems that the methodology implemented in CAME cannot handle. CAME cannot model different component failure modes or time-dependent failure rates. The failure rate associated with each component is assumed constant. The system operational description still relies on the subjective judgment of the analyst to determine the necessary components for the system to deliver its functionality.

1.2.1.4 Other System Reliability Evaluation Techniques

Although the techniques discussed previously are the most common ones to quantify the reliability of fault-tolerant systems, there are other techniques, some of which we briefly discuss in this section. Although these techniques might be appropriate to model and quantify the reliability of specific types of systems, they have the same common shortcoming as those already presented: the difficulty of generating the reliability model in a systematic and objective way from the functional descriptions of the system and the component failures (apart from the advancements in this matter introduced by HiP-HOPS and CAME).

Reliability Graphs. They are extensively used to model network reliability [29]. Reliability graphs are a graphical representation, in the form of directed graphs, of the system's structure function. The edges of the graph represent component failure events. There are two special nodes called the source node and the sink node. The source node has no incoming edges, and the sink node has no outgoing edges. The system will be functional whenever there is a connection between the source and the sink nodes. Reliability graphs are equivalent to reliability block diagrams [30]. Therefore, they have the same shortcomings, as RBDs for modeling fault-tolerant systems.

Event-Trees. They were developed during the $WASH - 1400$ study lead by Norman Rasmussen to assess the accident risk in nuclear power plants in the US [31]. Event-trees are a graphical representation, in the form of decision tree, of the outcomes that can result from an initiating event.

Nuclear power plants are designed with a philosophy called "defense-in-depth" which refers to (among other things) a set of safety systems that provide multiple barriers of defense between a hazard and the public [32]. Event-trees are a powerful tool to quantify the effectiveness of these different safety system swhen an undesirable event occurs. That is, to estimate the likelihood that, given an initiating event that perturbs the nominal operation of the plant, the multiple safety systems will bring the plant back to normal operation or safely shut it down.

Although, unlike fault-trees and RBDs, they cannot include repair processes event-trees allow modeling sequence-dependencies and imperfect failure coverage. Despite this, event-trees are not really suited to model the reliability of the class of fault-tolerant systems treated in this thesis, i.e., aircraft, space, and automotive systems. Event trees are suited to analyzing the consequences of a single initiating event, whereas the fault-tolerant systems used in the aforementioned applications are usually designed to withstand sequences of more than one initiating event. Furthermore, due to constraints on weight, cost, and volume, the "defense-in-depth" methodology is not employed in the design of this type of fault-tolerant systems. Usually only one, or at most two additional systems are put in place to compensate for component failures.

1.2.2 Dynamic Probabilistic Risk Assessment (DPRA): Removing the Subjectivity of Qualitative Functional Description Based Methodologies

One of the most important ideas introduced so far, the shortcoming of most current reliability evaluation techniques, is the difficulty of generating an objective reliability model from the functional description of the system and the components failure description. That is, to understand how the system behaves in the presence of failures. A methodology named Dynamic Probabilistic Risk Assessment (DPRA), developed in the nuclear engineering field, attempts to solve such a problem. It was developed to assess the likelihood of different accident sequences in a nuclear reactor [33, 34, 35].

The main difference (and most appealing feature) of DPRA with respect to the other reliability evaluation techniques described previously resides in the system description used to generate the reliability model. Rather than using a system qualitative functional model, DPRA bases the system reliability evaluation on quantifying the behavior of the system dynamic variables $x_s(t)$, for each possible configuration $\{i, k\}$ adopted by the system after the occurrence of a given sequences of k component failures. Thus, the ultimate objective of DPRA is to find the probability $p_{i,k}(x_s(t), t)$.

System Model. The system model is a collection of dynamic system models, each of which represents the system dynamic behavior for each possible configuration attained by the system [35]. Let $\{i, k\}$ be a configuration adopted by the system after a unique sequence of k component failures. The dynamics of this configuration is defined by

$$\begin{aligned} \dot{x}_s(t) &= f_s^{i,k}(x_s(t), w(t)), \\ y_s(t) &= g_s^{i,k}(x_s(t), w(t)), \end{aligned} \tag{1.11}$$

where k is the number of component operational mode transitions leading to system configuration $\{i, k\}$ from any of the initial system configurations $\{i, 0\}$, $x_s(t)$ is the vector of system state variables, $w(t)$ is the vector of system input variables, $y_s(t)$ is the vector of system instantaneous outputs, $f_s^{i,k}(\cdot)$ is the system state evolution function (for the $\{i, k\}$ configuration), and $g_s^{i,k}(\cdot)$ is the system instantaneous output function (for the $\{i, k\}$ configuration).

Model Evaluation. The system reliability evaluation results from quantifying the system dynamic behavior to check whether or not some sequences of events cause the system dynamic variables $x_s(t)$ to exceed some predetermined values. Assuming that the system state dynamic variables behave in a Markovian fashion, i.e., $x_s(t + dt)$ only depends on $x_s(t)$ and not on previous values of x_s [35], the probability distribution function $p_{i,k}(x_s(t), t)$ can be computed by solving the set of partial differential equations given by

$$\begin{aligned} &\tfrac{\partial p_{i,k}(x_s(t),t)}{\partial t} + div(f_s^{i,k}(x_s(t), w_s(t)) p_{i,k}(x_s(t), t)) = \\ &-\lambda_{i,k}(x_s(t), t) p_{i,k}(x_s(t), t) + \sum_{i,k-1} \lambda_{i,k-1}^{i,k,}(x_s(t), t) p_{i,k-1}(x_s(t), t) \end{aligned} \tag{1.12}$$

where $\lambda_{i,k}(x(t), t)$ results from adding all the individual transition rates that can trigger a system transition out of configuration $\{i, k\}$ and it depends on the system state variables $x_s(t)$. $\lambda_{i,k-1}^{i,k}(x_s(t), t)$ is the transition rate associated with the operational mode transition that causes the system to go from configuration $\{i, k - 1\}$ to system configuration $\{i, k\}$ and it depends on the system state variables $x_s(t)$ as well.

It might be possible to find analytical solutions to (1.12) for very simple systems [35]. However, the large number of components involved in the type of systems addressed by DPRA (nuclear power plants) makes an analytical solution untractable for (1.12). Thus, numerical methods (Monte Carlo simulation is one of them) are usually used to solve the problem. A comprehensive review of these methods can be found in [36].

Shortcomings. In nuclear power plants, the stochastic transitions between configurations and system state dynamic variables cannot be decoupled [37]. Thus, as discussed before, an analytical solution is untractable for (1.12) for very large systems. Numerical solutions are computationally very expensive. To illustrate this, let's take Monte Carlo simulation, which is one of the numerical methods proposed to solve the problem [38]. With Monte Carlo, random sequences of component failures are generated for the system operational time, and the system dynamic evolution is simulated (for each sequence of events) until meaningful reliability measures are obtained. For very reliable systems, only a few of these simulations will lead to system failure; e.g., in a system designed to have a reliability of 10^{-6}, there will be only one simulation out of one-million simulations resulting in system failure[21]. Therefore it is necessary to carry out a larger number of simulations to obtain a meaningful reliability result. Since component failures are regarded as rare events, achieving completeness in the possible sequences of failures is difficult [36]. However, there has been some work done to try to overcome the computational problems associated with DPRA. A good review of the main developments in this arena, as well as new work, can be found in [39]. Finally, unless there is only one operational point in nominal conditions (no failures), it is necessary to carry out the analysis for each possible plant initial operating point, which increases the computational burden even further.

1.3 Thesis Summary and Organization

The main shortcoming of current reliability evaluation techniques has been clearly identified —the incompleteness of the system models used to analyze the system behavior in the presence of failures. Thus, the main goal of this thesis is to introduce a new modeling and evaluation methodology for fault-tolerant systems addressing this problem. This methodology ought to minimize the subjectivity introduced in the system analysis due to incompleteness of current system modeling techniques.

Rather than using a qualitative description of the system's functionality (as in current techniques), the methodology proposed in this thesis will use a quantitative model of the system dynamic behavior (with no failures) plus additional features to model component failure behavior as well. These features include component failure modes (and associated failure rates) and how these failure modes affect the dynamic behavior of the component, thus reducing the ambiguity that might arise from the use of qualitative models to analyze system reliability.

The new methodology is based on the following principles:

1. Starting from the failure-free configuration, the system can evolve to different configurations, depending on which components failed, how these components failed, and the order in which the components failed (when more than one failed).
2. For each of these possible configurations, a system model that quantifies the system dynamic behavior is developed. This model will be used to determine whether or not the system delivers the performance for which it was designed. This is where the main difference with current techniques lies: the effect of component failures on the system dynamic behavior is modeled and quantified.
3. After the system transitions into a new configuration due to a component failure, two outcomes are possible. Either the system state variables remain within some predefined region dictated by the system performance requirements at all times, then the configuration is declared as *non-failed* (the failure is covered); or there are transient or permanent excursion outside this region, then the configuration is declared as *failed* (the failure is not covered).
4. The coverage probability (the probability of declaring a configuration as *non-failed*) will be computed as a function of the system state variables at the time of failure and the predefined region dictated by the system performance requirements.
5. The transitions between configurations occur stochastically, and the system is assumed to evolve between configurations in a Markovian fashion.

The other key aspect of the new methodology is that not only quantitative measures of reliability will result from the analysis, but also quantitative measures of dynamic performance. This is possible because the system behavior is quantified for each system configuration, and thus certain performance metrics of interest can be measured, e.g., the power consumption, the latency of certain signals, or the overshoot of certain system dynamic variables response. This allows the full integration of system reliability with dynamic performance evaluation, enabling a unified probabilistic-informed design framework, that will help to:

- identify weak points in a design so they can be improved in subsequent iterations of the design, and

- Compare different system architecture alternatives to help identifying the optimal one in terms of performance and reliability.

This will guide the system design towards optimal system dynamic performance, robustness, reliability, and fault tolerance. In the remainder of this section, the structure of the subsequent chapters of the thesis is detailed, explaining the contents of each chapter and main contributions

Chapter 2. Markov reliability models are used in the new methodology to model the system behavior due to component failures. Thus, the main purpose of this chapter is to make the reader familiar with the principles of Markov reliability modeling, and to introduce some of the terminology that will be used in the remainder of the thesis. To accomplish this, a case-study of a power net architecture for automotive safety-critical applications is presented.

The main contribution of this chapter is the development of a technique to systematically build system-level Failure Modes and Effects Analysis (FMEA) for each level of failure. This allows a direct mapping to the Markov reliability model, thus alleviating its construction. Most of the work presented in this chapter was published in [40].

Chapter 3. The mathematical foundations of the new integrated methodology for evaluating the performance and reliability of fault-tolerant systems is presented in this chapter. As mentioned before, the main advantage of the methodology is that, rather than using a qualitative description of the system's functionality, it uses a quantitative model of the system dynamic behavior (with no failures) plus additional features to model component failure behavior as well.

The main contributions of this chapter are:

- The rigorous definition of a generalized component behavioral model, which is a key part of the methodology.
- A rigorous approach for computing failure coverage probabilities, which is key to developing the system stochastic-behavior model due to component failures. In this regard, analytical solutions for the coverage probability in LTI systems are provided, as well as a methodology, based on Monte Carlo simulations, to compute these coverage probabilities in non-linear systems.

Chapter 4. One of the key features of fault-tolerant systems is complexity. The methodology presented in Chapter 3 attempts to solve some of the problems associated with complexity when analyzing the reliability of fault-tolerant systems. However, this methodology is useless if the analysis is not automated with the help of a computer. This chapter presents a MATLAB/SIMULINK® tool to support the methodology introduced in Chapter 3 —InPRESTo, an acronym for Integrated Performance and Reliability Evaluation SIMULINK® Toolbox.

The basic functionality is presented in this chapter. The automated analysis provided by InPRESto is not the only important contribution of the tool. InPRESTo also provides a common environment with a common languages for control engineers and reliability engineers to develop fault-tolerant systems. The InPRESTo developed as part of this thesis is being also used by the Systems Engineering and Evaluation Division at the Charles Stark Draper Laboratory for the evaluation of space and tactical systems.

Chapter 5. The purpose of this chapter is to show how the new methodology (and its supporting tool InPRESTo) can be used to: identify weak points in the system design; guide the design, pointing out to possible solutions to eliminate the uncovered weak points; compare different architecture alternatives from different perspectives; and test different failure detection, isolation, and reconfiguration (FDIR) techniques. To accomplish this, we present a case-study of a fault-tolerant architecture for a fighter aircraft lateral-directional flight control system [41].

This case-study proves the scalability of the tool to analyze large systems. Part of the this work was published in [42].

Chapter 6. This chapter illustrates how the new methodology (and its supporting tool) can be used to compare conceptually very different architectural approaches to achieve fault-tolerance. For this purpose, two different solutions to achieve fault-tolerance in a steer-by-wire (SbW) system are presented. The first solution is based on component redundancy and the introduction of failure detection, isolation, and reconfiguration mechanisms. In the second solution, a dissimilar backup mechanism called brake-actuated steering (BAS), is used to achieve fault-tolerance rather than replicating each component within the system.

This chapter complements Chapter 5 by showing how the performance and reliability evaluation SIMULINK ® toolbox -InPRESTo- can be used in a different way from that shown in Chapter 5. BAS is part of our earlier research in steer-by-wire and it was published in [43].

Chapter 7. The purpose of this chapter is to discuss the new methodology as the enabler of a unified system probabilistic-informed design framework. Existing probabilistic-informed decision making importance measures for system design will be reviewed. Definitions of new importance measures will be proposed within the framework of the new methodology. This chapter also discusses open questions and future research directions in the context of system probabilistic-informed design.

Chapter 8. The final chapter of the thesis collects the main conclusions extracted from this research. It summarizes the advancements of the new methodology with respect to existing ones, and highlights the fact that this research enables a new approach to probabilistic-informed design for fault-tolerant systems.

Chapter 2

On Markov Reliability Modeling: An Automotive Power Net Case-Study

The purpose of this chapter is to illustrate the principles of Markov reliability modeling when a block diagram of the system functionality (with no quantitative information of the component dynamic behavior) is used to understand the system behavior in the presence of component failures, and thus develop the Markov reliability model. In this context, a case study of an automotive power net architecture for automotive safety-critical applications is presented. Several system level Failure Modes and Effects Analysis (FMEA) will be developed from this block diagram, identifying the different sequences of component failures, helping the construction of the Markov reliability model. Sensitivity analysis will be used to understand the influence of perturbations in the model parameters. This will help to determine the robustness of the reliability estimate with respect to parameter uncertainty, and it will also help to improve the design. Most of the work presented in this chapter appears in [40].

2.1 Introduction

In automotive power nets, power for traditional loads is provided by a battery, an alternator, various switches, fuses or circuit breakers, and wiring. If any of these fails, there is a chance that the power net voltage will collapse and no actuation of any electrical system will be possible. Although this is a problem from the driver comfort point of view, the safety-critical systems of the car, such as conventional steering and braking systems are not electrical, and still function. However, with the introduction of steer-by-wire (SbW) and brake-by-wire (BbW), a loss of the power supply is no longer acceptable. Loss of electric power would mean loss of control of the vehicle, resulting in a dangerous situation for the driver. Considerable attention has been focused on the development of highly reliable SbW and BbW systems [1, 44, 45, 46], but only [1] and [46] talk about the fact that the power supply also has to be highly reliable and fault-tolerant, and even here, the authors' work is not focused on this issue. Current fault-tolerant power net designs are not reliable enough for use in safety-critical applications [47]. Therefore, it is important to develop new power net architectures and carry out a reliability analysis of these architectures to validate them for use in safety-critical applications. Some work has been done in this regard. In 1994, and anticipating the needs for future electrical loads in vehicles, [47] proposed alternative electrical distribution system architectures, already addressing the reliability issue of these new architectures. In [48], the requirements of vehicle power supply architectures were identified and, although some solutions were proposed, no further reliability analysis was done to validate them for their use in safety-critical applications.

In this chapter, a power net architecture based on one of the solutions given in [48] is proposed. It is not necessarily the optimal solution for the power net in terms of reliability, but it is complex enough to illustrate the use of Markov models for the reliability evaluation of Fault-Tolerant systems. To carry out the reliability analysis of the power supply for the proposed power net, several system level FMEAs are developed to identify the different sequences of component failures that can be reached from the failure-free configuration. A Markov model to quantify system reliability is constructed based on the FMEAs previously developed. The parameters of this model include time-dependent failure rates, and failure-coverage probabilities.

The power net architecture used in the study is defined in Section 2.2. Section 2.3 presents the system reliability model, including the different system level FMEAs, and the subsequently developed Markov reliability model. Section 2.4 presents the analysis results, including a sensitivity analysis of the reliability estimate with respect to some model parameters. Concluding remarks are presented in Section 2.5.

2.2 Dual Battery Power Net Architecture

The proposed power net architecture, shown in Fig. 2.1, consists of the following elements:

- Alternator G, which generates energy for the electric loads and for charging the battery.
- Main battery B_1, which provides energy for the electric loads.
- Backup battery B_2, which is in cold standby and only switched on in case of a failure of

the alternator G, or the main battery B_1. The backup battery B_2 will be chosen to have the same capacity as the main battery B_1.

- Voltage and current sensors S_V and S_A, which measure the voltage of the power supply and the current flowing through the main battery B_1 and alternator G.
- Switches SW_1, SW_2 and SW_3.
- Electronic control unit ECU, which receives signals from the voltage and current sensors S_V and S_A, and sends signals to the switches SW_1, SW_2 and SW_3 in case a failure occurs.
- Main wire harness MWH, which links the power supply with the fuse box.
- Fuses F, for short circuit protection.
- Wire harness H.
- Steer-by-Wire channels SbW_1 and SbW_2.
- Conventional electric loads L.

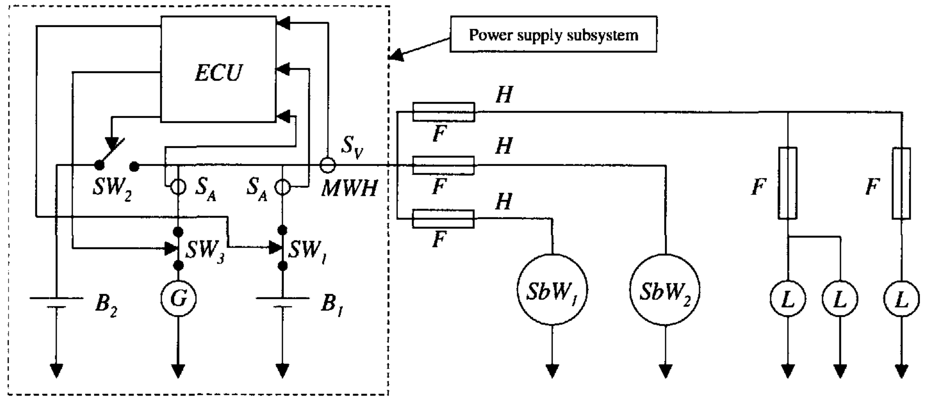


Figure 2.1: Dual battery power net architecture.

The primary difference between the proposed and conventional power nets is the backup battery B_2, and the detection and isolation system (composed of the electronic control unit ECU, the voltage and current sensors S_V and S_A, and the switches SW_1, SW_2 and SW_3). If a fault is detected in the alternator G or the main battery B_1, the detection and isolation system switches off the faulty element and switches on the backup battery B_2. Additionally, the backup battery is also switched on if there is a voltage drop in the power supply, even when no fault has been detected in the alternator G or the main battery B_1. No failure annunciation system is considered for non-failed configurations with one failed component, which means that the system must work for the stated period of time without maintenance.

2.3 System Reliability Model

The reliability analysis will be focused on how failures in the power supply subsystem components affect the power delivery to the electrical loads. Thus, the analysis is restricted to the alternator G, main battery B_1, backup battery B_2, voltage and current sensors S_V and S_A, switches SW_1, SW_2 and SW_3, electronic control unit ECU, and main wire harness MWH.

The process of building the system reliability model starts with a qualitative description of the system's functionality, Fig. 2.1. It describes how components and subsystems are interconnected to fulfill the functions the system was designed for. Additionally, component failure modes (and associated failure rates), and how these component failures can affect the system functionality are required. Then, based on this information, several system level FMEAs are developed, which collect the system configurations reached after one or more components failed; and if these configurations result in system failure, or the system is still operational. Finally, a Markov model is built from the information collected in the FMEAs, and system reliability measures can be obtained.

2.3.1 System Level Failure Modes and Effects Analysis (FMEA)

In order to quantify the reliability of a fault-tolerant system, it is necessary to build a Markov model. However this can be tedious and error-prone process for large systems. Thus, in order to alleviate this task, several system level FMEAs are usually first developed, which help in the subsequent construction of the Markov model. The first FMEA will help to identify first component failures in the system, which yield both failed and non-failed system configurations. From the non-failed system configurations with one component failed, a second FMEA will be developed. This second FMEA will identify failed and non-failed system configurations after two components have failed. The process will end when all the system configurations are declared as failed. To construct the system level FMEA, several simplifying assumptions are introduced:

- The backup battery B_2 has zero failure rate while it is in the standby condition.
- The main battery B_1 and the backup battery B_2 have equal failure rates when they are working.
- Since no annunciation system is considered in this architecture, repair processes for component failures that do not result in a failed configuration are not considered.
- The failure detection and isolation system is comprised of the ECU; the switches SW_1, SW_2, SW_3; and the sensors S_1, S_2, S_3.
- If a failure occurs in any of the elements of the failure detection and isolation system, this element will be disabled, and additional failures of any of its components will not affect the rest of the system.
- The failure detection and isolation system is disabled once a fault has been successfully detected and isolated.

Table 2.1: System level FMEA for single component failures.

Failure-free configuration	*State*	*Failure mode*	*Transition rate*	*System configurations after one failure*	*System status*	*State*
B_1 Delivering energy, and G generating energy, and DS monitoring the system, and MWH transporting energy	$\{1,0\}$	B_1 fails covered	$c\lambda_{B_1}$	B_2 delivering energy, and G generating energy, and MWH transporting energy	Non-failed	$\{1,1\}$
		B_1 fails uncovered	$(1-c)\lambda_{B_1}$	-	Failed	$\{2,1\}$
		G fails covered	$c\lambda_G$	B_1 delivering energy, and B_2 delivering energy, and MWH transporting energy	Non-failed	$\{3,1\}$
		G fails uncovered	$(1-c)\lambda_G$	-	Failed	$\{4,1\}$
		DS fails	$\lambda_{ECU} + 3\lambda_{SW} + 3\lambda_S$	B_1 delivering energy, and G generating energy, and MWH transporting energy	Non-failed	$\{5,1\}$
		MWH fails	λ_{MWH}	-	Failed	$\{6,1\}$

Table 2.1 corresponds to the FMEA for single component failures in the system. The first column of this table lists the system failure-free configuration (no component failures). The second column associates the failure-free configuration with state $\{1,0\}$ of the Markov reliability model developed in the next section. The third column describes all possible component failure modes. The fourth column lists the transition rates from the failure-free configuration $\{1,0\}$ to the new configurations. The fifth column describes the new system configurations resulting after the component failures described in the third column occurred. Column six describes the resulting configuration as failed or non-failed. Finally, column seven associates each new resulting configuration with a state in the Markov

model —absorbing states for the failed configurations, and transient states for the non-failed configurations. Table 2.2 corresponds to the FMEA for sequences of two component failures. The first column of this table corresponds to the non-failed configurations reported in column five of Table 2.1. The remaining columns of Table 2.2 are obtained in the same way as for Table 2.1.

Table 2.2: System level FMEA for sequences of two component failures.

Non-failed system configurations after one component failed	*State*	*Failure mode*	*Transition rate*	*System configurations after two failures*	*System status*	*State*
B_2 delivering energy, and	$\{1, 1\}$	B_2 fails.	λ_{B_2}	-	Failed	$\{1, 2\}$
G generating energy, and		G fails.	λ_G	-	Failed	$\{2, 2\}$
MWH transporting energy		MWH fails.	λ_{MWH}	-	Failed	$\{3, 2\}$
B_1 delivering energy, and	$\{3, 1\}$	B_1 fails.	λ_{B_1}	-	Failed	$\{4, 2\}$
B_2 delivering energy, and		B_2 fails.	λ_{B_2}	-	Failed	$\{5, 2\}$
MWH transporting energy		MWH fails.	λ_{MWH}	-	Failed	$\{6, 2\}$
B_1 delivering energy, and	$\{5, 1\}$	B_1 fails.	λ_{B_1}	-	Failed	$\{7, 2\}$
G generating energy, and		G fails.	λ_G	-	Failed	$\{8, 2\}$
MWH transporting energy		MWH fails.	λ_{MWH}	-	Failed	$\{9, 2\}$

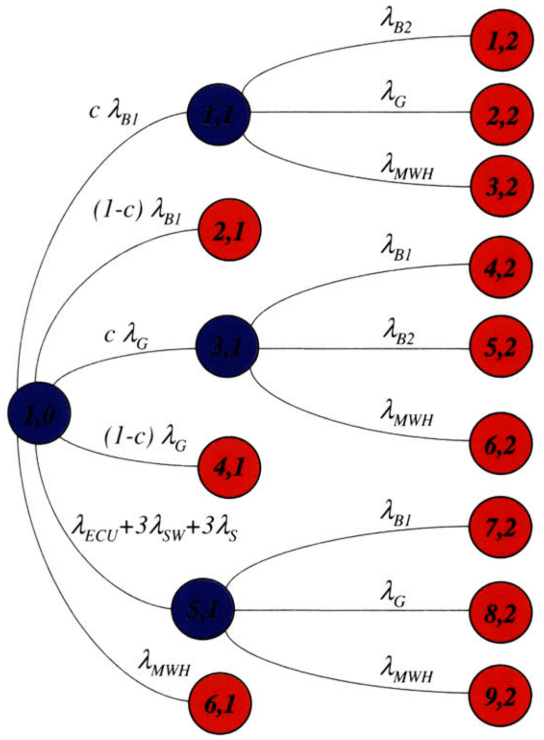


Figure 2.2: Markov model for the power supply of the dual battery power net architecture of Figure 2.1.

2.3.2 Markov Model

Based on the system level FMEA developed in the previous section, it is possible to construct a Markov model that represents the stochastic behavior of the system due to component failures. Figure 2.2 displays the state-transition diagram associated with the Markov model. State $\{1,0\}$ represents a system configuration with no failures. States $\{1,1\}$-$\{6,1\}$ represent the status of system configurations reached after one failure. Finally, states $\{1,2\}$-$\{9,2\}$ represent the status of system configurations after two failures.

Each coefficient $a_{i,j}$ of the state-transition matrix A is obtained by combining the component failure rates (which trigger the transitions between system configurations) and the coverage probabilities (column 4 in Tables 2.1 and 2.2), resulting in

$$A = \begin{bmatrix}
a_{1,1} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
a_{2,1} & a_{2,2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
a_{3,1} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
a_{4,1} & 0 & 0 & a_{4,4} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
a_{5,1} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
a_{6,1} & 0 & 0 & 0 & 0 & a_{6,6} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
a_{7,1} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & a_{8,2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & a_{9,2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & a_{10,2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & a_{11,4} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & a_{12,4} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & a_{13,4} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & a_{14,6} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & a_{15,6} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & a_{16,6} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix}. \tag{2.1}$$

The coefficients of the state-transition matrix A are given by:

- $a_{1,1} = -\lambda_{B_1} - \lambda_G - \lambda_{ECU} - 3\lambda_{SW} - 3\lambda_S - \lambda_{MWH}$,
- $a_{2,1} = c\lambda_{B_1}$, $a_{3,1} = (1-c)\lambda_{B_1}$,
- $a_{4,1} = c\lambda_G$, $a_{5,1} = (1-c)\lambda_G$,
- $a_{6,1} = \lambda_{ECU} + 3\lambda_{SW} + 3\lambda_S$,
- $a_{7,1} = \lambda_{MWH}$,
- $a_{2,2} = -\lambda_{B_2} - \lambda_G - \lambda_{MWH}$,
- $a_{8,2} = \lambda_{B_2}$, $a_{9,2} = \lambda_G$,

- $a_{10,2} = \lambda_{MWH}$, $a_{11,4} = \lambda_{B_1}$,
- $a_{4,4} = -\lambda_{B_1} - \lambda_{B_2} - \lambda_{MWG}$,
- $a_{12,4} = \lambda_{B_2}$, $a_{13,4} = \lambda_{MWH}$,
- $a_{6,6} = -\lambda_{B_1} - \lambda_G - \lambda_{MWG}$,
- $a_{14,6} = \lambda_{B_1}$ $a_{15,6} = \lambda_G$,
- $a_{16,6} = \lambda_{MWH}$,

with λ_{B_1} the battery B_1 failure rate, λ_{B_2} the battery B_2 failure rate, λ_G the alternator G failure rate, λ_{MWH} the main wire harness MWH failure rate, λ_{ECU} the electronic control unit ECU failure rate, λ_{SW} the switch SW failure rate, λ_S the sensor S failure rate, and c the failure coverage probability.

Component Failure Rates

The failure rates for the main battery λ_{B_1}, the backup battery λ_{B_2}, the alternator λ_G, and the main wire harness λ_{MWH} are considered to be time-dependent and Weibull distributed:

$$\lambda(t) = \alpha\lambda_0(\lambda_0 t)^{\alpha-1}, \tag{2.2}$$

where α is called the shape parameter, λ_0 is the scale parameter and t is the time [6]. The values of α and λ_0 for each component (alternator, main and backup batteries, and main wire harness) were obtained from field data provided by the Allgemeiner Deutscher Automobil Club [2], and are displayed in Table 2.3.

The failure rates for the rest of the components, i.e., ECU, sensors S_V and S_A, and switches SW, are assumed to be constant and were chosen based on typical data for automotive components [1]. Since the values of the failure rates for the ECU, the sensors S_V and S_A, and the switches SW are assumed, a sensitivity analysis will be carried out for each of these components to see how a change in its failure rate affects the reliability of the system.

Failure Coverage Probability

As stated in Appendix A, the failure coverage probability depends on the ability of the system to detect and isolate a failure, and reconfigure itself in order to keep delivering its functionality. Detection depends on the detection algorithm successfully detecting a

Table 2.3: Component failure rates and detection probabilities used in the development of the simplified Markov model for the power net of Figure 2.1 (from [1] and [2]).

Component	*Description*	α	λ_0 (/h)	λ (/h)	D
Alternator	G	2.68	$0.32 \cdot 10^{-5}$	$5.14 \cdot 10^{-15} \cdot t^{1.68}$	0.99
Main battery/backup battery	B_1/B_2	3.56	$0.21 \cdot 10^{-4}$	$7.69 \cdot 10^{-17} \cdot t^{2.56}$	0.99
Main wire Harness	MWH	1.95	$0.32 \cdot 10^{-6}$	$4.28 \cdot 10^{-13} \cdot t^{0.95}$	-
Electronic control unit	ECU	1	$5 \cdot 10^{-7}$	$5 \cdot 10^{-7}$	-
Voltage and current sensors	S_V/S_A	1	10^{-7}	10^{-7}	-
Switches	$SW_1/SW_2/SW_3$	1	10^{-6}	10^{-6}	-

failure, and the voltage and current sensors S_V and S_A, and the electronic control unit ECU working on demand. Thus the probability of detecting a failure when it occurs is given by

$$P(D/F) = DP(X_t^{S_V} = 0)P(X_t^{S_A} = 0)P(X_t^{S_A} = 0)P(X_t^{ECU} = 0), \tag{2.3}$$

where D is the detection probability given in Table 2.3. A successful failure isolation and reconfiguration occurs when there is no failure in the components involved in the isolation and reconfiguration mechanism, which are the switches SW_1, SW_2, and SW_3. Thus the probability of failure isolation and reconfiguration, given a failure occurred and was detected, is given by

$$P(I \cap R/F \cap D) = P(X_t^{SW_1} = 0)P(X_t^{SW_2} = 0)P(X_t^{SW_3} = 0). \tag{2.4}$$

Therefore, considering that all the switches have the same failure rate λ_{SW}, and the sensors having, as well, the same failure rate λ_S, the failure coverage probability can be computed using

$$c = De^{-(\lambda_{ECU} + 3\lambda_{SW} + 3\lambda_S)t}. \tag{2.5}$$

2.4 Results Analysis

A vehicle lifetime of 15 years and an average of 400 working hours per year was considered for the simulations, which gives an evaluation time T of 6000 h. The Dependability rate $\bar{\lambda}(T)$ (defined in A.5 in Appendix A as the ratio of the system unreliability $Q(T)$ to time T) will be used as the system reliability measure. Using the parameters of Table 2.3, which correspond to the assumed nominal failure rate values, and the failure coverage probabil-

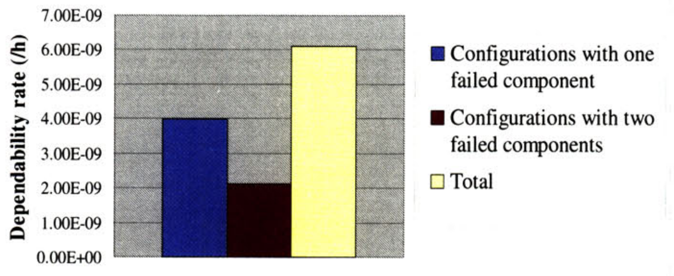


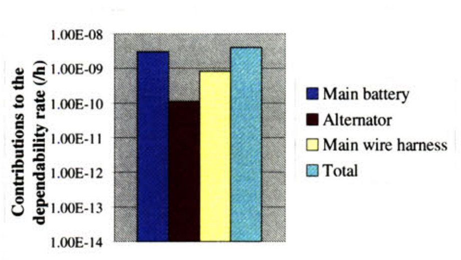
Figure 2.3: System configurations contributions to the total system dependability rate $\bar{\lambda}(T)$ sorted by number of components failed.

ities given by substituting the corresponding values in (2.5), the dependability rate $\bar{\lambda}(T)$ yielded by the Markov model is $6.1 \cdot 10^{-9}$ failures/hour. In the remainder of this section, further analysis, displayed in Figures 2.3 - 2.5, is carried out to gain more insight to:

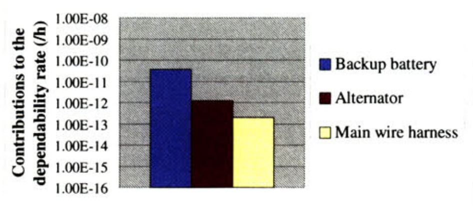
- contributions of system failed configurations (sorted by number of components failed) to the system dependability rate $\bar{\lambda}(T)$;
- contributions to the system dependability rate $\bar{\lambda}(T)$ of the last failed component within failed configurations (sorted by number of components failed); and
- the influence on the estimate of the dependability rate $\bar{\lambda}(T)$ of perturbations in some of the model parameters.

Dependability Rate Breakdown by Number of Failed Components

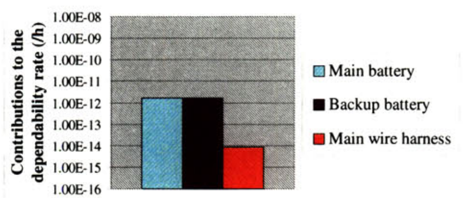
Figure 2.3 shows the distribution of system configurations declared as failed after one and two components failed. It is important to note that the most important contribution, amounting to 66% of the total system dependability rate $\bar{\lambda}(T)$ estimate, is given by failed system configurations reached after one component failed, which corresponds to uncovered failures of the main battery B_1 and alternator G, and the failure of the main wire harness MWH.



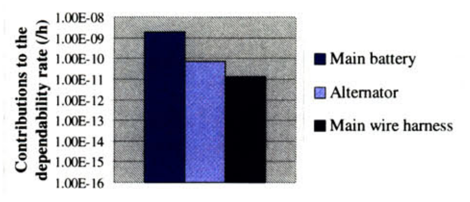
(a) Component contributions to the dependability rate of system configurations with one component failed.



(b) Component contributions to the dependability rate of system configurations with a second component failed, after the main battery B_1 failed first.



(c) Component contributions to the dependability rate of system configurations with a second component failed, after the alternator G failed first.



(d) Component contributions to the dependability rate of system configurations with a second component failed, after the DS failed first.

Figure 2.4: Contributions to the dependability rate $\bar{\lambda}(T)$ for system configurations with one and two components failed.

Impact of Individual Component Failures on the Dependability Rate

For a better understanding of individual impact, Figure 2.4 displays the contributions to the dependability rate $\bar{\lambda}(T)$ of the last failed component for failed system configurations with one and two failed components. Thus, Fig. 2.4(a) displays the single contributions of uncovered single failures in the main battery B_1, alternator G, and main wire harness MWH. In this case, the main contribution to system failure, being $3 \cdot 10^{-9}$ failures/hour, comes from the main battery B_1.

Figures 2.4(b) - 2.4(d) display the individual contributions of the last failed component within system configurations declared as failed after two component failures. It is interesting to note the influence of the main wire harness MWH, depending on the number of failed components. For system configurations with one component failed, Fig. 2.4(a), the contribution of the MWH failure is of the same order of magnitude as the other contributors. However, after each first failure, the contribution of the main wire harness is always the least important, at one or two orders of magnitude less than the other contributors. For example, after a covered failure in the main battery, the contribution of the backup battery is $3.7 \cdot 10^{-12}$ failures/hour and the alternator contribution is $1.2 \cdot 10^{-12}$ failures/hour, while the main wire harness contribution is $2 \cdot 10^{-13}$ failures/hour. The failure coverage probability is responsible for this behavior. For single failures, the transition rate out of the nominal configurations for uncovered failures of B_1 and G is obtained by multiplying the corresponding component failure rate by $1 - c$, where c is the coverage probability. It is expected that c will be close to one. Thus, even for failure rates in $B1$ and G one or two orders of magnitude bigger that the MWH failure rate, the uncovered failures of B_1 and G contribute in a similar amount to that corresponding to the MWH. For system configurations with two failures, the failure coverage probability is not present, thus the difference between the MWH failure rate, and the B_1 and G failures rates is no longer attenuated by the factor $1 - c$, resulting in the one or two order of magnitude difference observed in Figures 2.4(b)- 2.4(d).

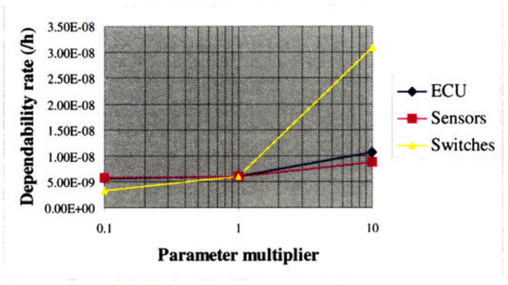
Sensitivity Analysis

The purpose of running a sensitivity analysis with respect to the the model parameters is twofold:

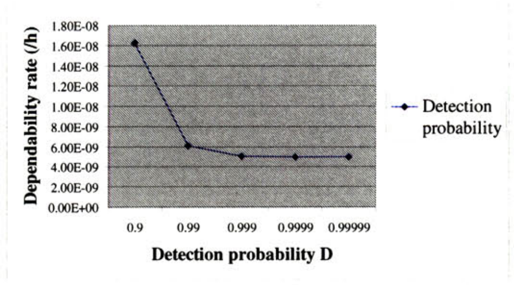
- Highlight the influence of the assumed failure rates of ECU, voltage and current sensors, S_V and S_A, and switches SW on the dependability rate $\bar{\lambda}(T)$.
- Determine how sensitive the dependability rate $\bar{\lambda}(T)$ is with respect to the detection probability D. This will help to establish how good the detection algorithm must be.

The procedure followed to estimate the influence of the failure rate parameters was to perturb the value of each component failure rate, one at a time; then recompute the dependability rate for each perturbed parameter to determine how this perturbations affected the nominal solution [49]. The parameter multipliers used were 0.1 and 10 for all components.

Figure 2.5(a) displays the sensitivity analysis results. The influence of changes in ECU and voltage and current sensor, S_V and S_A, failures rates, although difficult to see in the figure, is almost the same, and it is small in comparison with the effect of changes in the switch



(a) Sensitivity analysis to changes in the ECU, the switches SW and the sensors S failure rates.



(b) Sensitivity analysis to changes in the detection probability D.

Figure 2.5: Dependability rate $\bar{\lambda}(T)$ sensitivity to the failure rates of the electronic control unit λ_{ECU}, the sensors λ_S, and the switches λ_{SW}; and to the detection probability D.

SW failure rates. The switches SWs are the components of the detection and isolation system that most influence the dependability rate, i.e., an increase in the failure rate of the switches translates to a one order of magnitude increase of the dependability rate. The rest of the failure rate changes keep the dependability rate within the same order of magnitude as that obtained using the nominal failure rates.

A similar procedure to the one explained above was used for determining the influence of the detection probability D. The results are displayed in Fig. 2.5(b). This analysis shows that the influence of the detection and isolation system in the overall dependability rate is very important. As seen, the dependability rate strongly depends on the detection probability D when it is less than 0.99. Above $D = 0.99$, the dependability rate estimate is insensitive to the detection probability.

The results reported in this section suggest that one way to improve the dependability of this architecture, would be to improve the detection and isolation system by improving the detection algorithm. Thus, achieving a detection probability D of 0.99 or greater, which would make the dependability rate insensitive to this parameter. Another way to improve the dependability would be by redesigning the link between the power supply and the main fuse box, i.e., the main wire harness in the previous design. This would prevent single failures in the main wire harness from making the system fail despite the redundancy introduced by the second battery.

2.5 Conclusions

The case-study presented in this chapter illustrates the steps commonly applied to evaluate the reliability of a system. The process requires a qualitative evaluation of a functional description of the system in the presence of failures. For conventional systems, this approach is valid, as it is possible to infer the system behavior (under component failure conditions) without using a quantitative model of the system. However, for more complex systems, it is very difficult to understand the system behavior in the presence of component failures. For example, in this case-study, components were only regarded as failed or operational, without specifying in which sense they have failed. we could have considered, for example, the battery failing open or short circuit, or the alternator excitation system partially failing. In these cases, it is much more difficult to understand the system behavior; thus, it is very difficult to assess if these failures cause the system to stop delivering power to the loads. Furthermore, with a qualitative system description it is not possible to quantify system degraded operational modes, i.e., the system might be delivering enough power for some loads, but not for all of them. In the next chapter, we introduce a new methodology for evaluating system reliability that bases all the analyses on a quantitative model of the system to be evaluated.

Notation Used in this Chapter

A :	Markov reliability model state transition matrix
BbW:	Brake-by-wire
$B_{1(2)}$:	Main (Backup) battery
c :	Failure coverage probability
DS :	Detection and isolation system
ECU :	Electronic control unit
F :	Fuse
FMEA:	Failure Modes and Effects Analysis
G :	Alternator
H :	Wire harness
L :	Conventional electrical loads
MWH :	Main wire harness linking the power supply with the fuse box
$P(D/F), D$:	Probability of failure detection given a failure has occurred
$P(I \cap R/F \cap D)$:	Probability of failure isolation and reconfiguration given a failure has occurred, and it has been detected
S_A :	Current sensor
S_V :	Voltage sensor
SbW:	Steer-by-wire
SbW_1 :	Steer-by-Wire channel 1

SbW_2 :	Steer-by-Wire channel 2
SW_1 :	Main battery switch
SW_2 :	Alternator switch
SW_3 :	Backup battery switch
t :	Time
T :	System global evaluation time
X_t^{ECU} :	Number of failures in ECU for an operating time of t hours
$X_t^{S_A}$:	Number of failures in S_A for an operating time of t hours
$X_t^{S_V}$:	Number of failures in S_V for an operating time of t hours
$X_t^{SW_1}$:	Number of failures in SW_1 for an operating time of t hours
$X_t^{SW_2}$:	Number of failures in SW_2 for an operating time of t hours
$X_t^{SW_3}$:	Number of failures in SW_3 for an operating time of t hours
α :	Shape parameter of the Weibull distribution
λ :	Failure rate
$\lambda_{B_1}/\lambda_{B_2}$:	Battery failure rate
λ_G :	Alternator failure rate
λ_{ECU} :	Electronic control unit failure rate
λ_S :	Sensor failure rate
λ_{SW} :	Switch failure rate
λ_0 :	Scale parameter of the Weibull distribution
$\bar{\lambda}(T)$:	Dependability rate

Chapter 3

On the Integration of System Performance and Reliability Evaluation

In this chapter, we propose a methodology for integrating the evaluation of system performance and reliability. The methodology uses a behavioral model of the system dynamics, similar to the ones used by control engineers when designing the control system, but with additional features to model different failure behaviors of the component. The performance evaluation is based on the system dynamic behavior when component failures occur within the system. The proposed methodology allows one to assess not only system reliability, but other important dynamic performance metrics that might be relevant in the design of a fault-tolerant system. The system stochastic behavior due to component failures is modeled by using Markov models. In this context, a rigorous approach for computing failure coverage probabilities is presented, providing analytical solutions for LTI systems, and a Monte-Carlo based methodology for non-linear systems. Several examples are developed to illustrate the concepts introduced in the chapter.

3.1 Introduction

The analysis of system behavior in the presence of component or subsystem failures and therefore system reliability, availability, and safety, is usually carried out using a qualitative description of the system's functionality. It describes how components and subsystems are interconnected to fulfill the functions for which they are designed, and how component failures can propagate to other components through their interconnections and affect the system functionality [6, 7, 8]. For conventional systems this approach is valid, as it is possible to evaluate how a system can fail to perform its function without paying attention to its dynamics. However, this is not the case for large complex systems or embedded software-intensive systems, which are characteristic of fault-tolerant systems [9]. For these kinds of systems, it is very difficult, if not impossible, to understand how the system behaves in the presence of hardware failures or software malfunctions without using a model of the system dynamics to quantify its performance under failure conditions.

In this chapter, we propose a methodology that integrates performance and reliability evaluations of fault-tolerant systems. Rather than using a qualitative description of the system's functionality, this methodology uses a model of the system dynamics which is augmented with additional features to model component failure behavior. These features include component failure modes (and associated failure rates) and how these failure modes affect the dynamic behavior of the component.

Starting from the *failure-free* configuration, the system will evolve to different *system configurations*, depending on which components fail, how these components fail, and the order in which the components fail (when more than one fail). For each of the possible system configurations, an evaluation of the system dynamic behavior performance is carried out to check whether some of its properties, called *performance metrics*, meet some predefined operational *requirements*. After all system configurations have been evaluated, the values of the performance metrics for each configuration and the probabilities of going from the nominal configuration to any other configuration are merged into a set of *probabilistic measures of performance.*

As discussed in Chapter 1 (see Section 1.2.2), the idea of using a dynamic model of the system under control was first proposed in the nuclear engineering field and the resulting methodology is commonly known as Dynamic Probabilistic Risk Assessment (DPRA). However, as will be discussed in Section 3.7, there are certain aspects of the problems that DPRA addresses that make DPRA differ substantially from the methodology proposed in this chapter.

Section 3.2 introduces the mathematical framework for modeling the dynamic behavior of the system to be evaluated. Section 3.3 explains how to define the performance metrics (and associated requirements) that will be used to evaluate the dynamic performance of the system. In Section 3.4, the mathematical model associated with the system stochastic behavior due to component failures is presented. Section 3.5 provides an example of how the present methodology can be used to analyze a series RL circuit. Section 3.6 presents different probabilistic measures of performance, which are used to evaluate the overall system performance. Section 3.7 compares the new methodology with DPRA. Concluding remarks are presented in Section 3.8.

3.2 System Dynamics Behavioral Model

The system dynamics behavioral model will emerge from the interaction of the behavioral models of each individual component within the system. We understand the term behavioral model as understood in circuit simulation — a set of mathematical expressions that models the component external behavior (manifested through its connections to other components), without necessarily modeling the real physical processes involved in such behavior. A component behavioral model will define not only the component behavior under failure-free conditions, but also under different failure conditions (failure modes).

3.2.1 Components

In classical reliability analysis, the concept of *failure mode* is used to define a component operational behavior under specific internal failure conditions. Similarly, the concept of the *failure-free mode* of operation is used to define the component behavior under failure-free conditions. We will use the expression *operational mode* to refer to both failure and failure-free modes. Thus, we define a component behavioral model by:

- A list of the component variables of interest relevant to each component operational mode definition and a set of mathematical expressions, termed behavioral equations, that constrain those variables.
- A stochastic model that describes the transitions between different operational modes triggered by component-internal failure conditions, or by component-external events.

The mathematical expressions constraining the component variables can be of very different natures, depending on the component to be modeled, e.g., algebraic equations, differential equations, difference equations, logical expressions, and/or combinations of those.

In the remainder of this section, we will focus on defining component models for a class of systems in which the variables of interest evolve as a function of time —dynamic systems. Thus, differential (or difference) equations can be used to describe the constraints among these variables. In particular, the state-space description form of a dynamic system will be used to define each mode of operation (both failure and failure-free).

Behavioral Equations

Let a component c_i have k operational modes. The j operational mode behavioral equations for $j = \{0, 1, \dots k\}$ are defined by

$$\begin{aligned}\dot{x}_{c_i}(t) &= f^j_{c_i}(x_{c_i}(t), u_{c_i}(t)) \\ y_{c_i}(t) &= g^j_{c_i}(x_{c_i}(t), u_{c_i}(t)),\end{aligned} \tag{3.1}$$

where $x_{c_i}(t)$ is the vector of component state variables, $u_{c_i}(t)$ is the vector of component input variables, $y_{c_i}(t)$ is the vector of component instantaneous outputs, $f^j_{c_i}(\cdot)$ is the component state evolution function (for the j operational mode), and $g^j_{c_i}(\cdot)$ is the component instantaneous output function (for the j operational mode).

The transitions between different operational modes occur stochastically and can be triggered by internal failure conditions or by external events. Let $U_{c_i}(t)$ be a random variable that can take values in the set $B_{c_i} = \{0, 1, \dots k\}$, representing the component c_i instantaneous component operational modes. Assuming the transitions between different operational failure modes occur in a Markovian fashion, we can define the instantaneous transition rate $\lambda_{lm}(t)$ between any two component operational modes l and m by

$$\lambda_{lm}(t) = \frac{P(U_{c_i}(t+dt)=m|U_{c_i}(t)=l)}{dt}, \tag{3.2}$$

where $l = 0, 1, \dots k$ and $m = 0, 1, \dots k$.

Sensor Behavioral Model Example

To illustrate the process introduced above, the behavioral model for a sensor c_i will be defined. The sensor has four operational modes:

- **Failure-free mode (N):** bandwidth τ_s, resolution R_s, latency of τ_l, gain G_s.
- **Output-omission failure mode (O):** regardless of the sensor input reading, its output is set to zero, i.e, gain 0 and other properties the same as in the failure-free mode.
- **Gain-change failure mode (G):** bandwidth τ_s, resolution R_s, latency of τ_l, gain $\hat{G}_s$.
- **Bias failure mode (B):** bandwidth τ_s, resolution R_s, latency of τ_l, gain G_s, and output biased by a factor B_s.

Under the above conditions, the behavioral equations for each operational mode of the sensor are defined by

$$U_{c_1}(t) = 0\,(failure - free),$$

$$\dot{x}_{c_1}(t) = -\tfrac{1}{\tau_s}x_{c_1}(t) + \tfrac{1}{\tau_s}u_{c_1}(t); y_{c_1}(t) = G_s\frac{\lceil \sigma_{\tau_l} x_{c_1}(t)\frac{1}{R_s}\rceil}{\frac{1}{R_s}}; \tag{3.3}$$

$$U_{c_1}(t) = 1\,(output\ omission),$$

$$\dot{x}_{c_1}(t) = -\tfrac{1}{\tau_s}x_{c_1}(t) + \tfrac{1}{\tau_s}u_{c_1}(t); y_{c_1}(t) = 0; \tag{3.4}$$

$$U_{c_1}(t) = 2\,(gain\ change),$$

$$\dot{x}_{c_1}(t) = -\tfrac{1}{\tau_s}x_{c_1}(t) + \tfrac{1}{\tau_s}u_{c_1}(t); y_{c_1}(t) = \hat{G}_s\frac{\lceil \sigma_{\tau_l} x_{c_1}(t)\frac{1}{R_s}\rceil}{\frac{1}{R_s}}; \tag{3.5}$$

$$U_{c_1}(t) = 3\,(bias),$$

$$\dot{x}_{c_1}(t) = -\tfrac{1}{\tau_s}x_{c_1}(t) + \tfrac{1}{\tau_s}u_{c_1}(t); y_{c_1}(t) = G_s\frac{\lceil \sigma_{\tau_l} x_{c_1}(t)\frac{1}{R_s}\rceil}{\frac{1}{R_s}} + B_s. \tag{3.6}$$

Transitions from the failure-free mode to each failure mode can occur, as well as transitions from the gain-change and the bias failure modes to the output-omission failure mode (operational modes do not coexist). Thus, the instantaneous transition rates are defined by

$$\begin{aligned}
\lambda_{NO}(t) &= \tfrac{P(U_{c_1}(t+dt)=1|U_{c_1}(t)=0)}{dt}\\
\lambda_{NG}(t) &= \tfrac{P(U_{c_1}(t+dt)=2|U_{c_1}(t)=0)}{dt}\\
\lambda_{NB}(t) &= \tfrac{P(U_{c_1}(t+dt)=3|U_{c_1}(t)=0)}{dt}\\
\lambda_{GO}(t) &= \tfrac{P(U_{c_1}(t+dt)=1|U_{c_1}(t)=2)}{dt}\\
\lambda_{BO}(t) &= \tfrac{P(U_{c_1}(t+dt)=1|U_{c_1}(t)=3)}{dt}.
\end{aligned} \tag{3.7}$$

3.2.2 Configurations

In a component behavioral model, a single component can exhibit different operational modes depending on its internal failure conditions and on external events. Similarly, a system will adopt different *configurations* depending on the the status of its components [39], i.e., which component operational modes are active. Thus, the system is said to be in its failure-free configuration if all its components are in their failure-free modes. The system may evolve from its failure-free configuration to another dynamic configuration if a component transitions from its failure-free mode to one of its possible failure modes. Under these conditions, a system model is defined by:

- A set of component models and how they interact with each other. Depending on the components' operational modes, the system will adopt different configurations.
- A stochastic model that describes the transitions between different system configurations triggered by component operational mode transitions.

In the previous section, we focused on defining component behavioral models for dynamic systems, adopting the state-space description for their definition. In this section, we will use the same formalism; we will assume that the systems of interest are composed of interconnected dynamic systems, and therefore a state-space description will be used to define each system configuration.

Configuration Dynamics Equations

Let k be the number of component operational mode transitions leading to the system configuration $\{i, k\}$ from the initial system configuration $\{1, 0\}$, and let $i \geq 1$ index the set of system configurations which have $k > 0$ components failed. The dynamics of the $\{i, k\}$ system configuration reached after a unique sequence of k component operational mode transitions be represented by

$$\begin{aligned} \frac{dx_s(t^{i,k})}{dt^{i,k}} &= f_s^{i,k}(x_s(t^{i,k}), w(t^{i,k})) \\ y_s(t^{i,k}) &= g_s^{i,k}(x_s(t^{i,k}), w(t^{i,k})) \\ x_s(0^{i,k}) &\in \Omega_x^{i,k}(0^{i,k}) \\ w(t^{i,k}) &\in \Omega_w(t^{i,k}), \end{aligned} \tag{3.8}$$

where $x_s(t^{i,k})$ is the vector of system state variables, $w(t^{i,k})$ is the vector of system input variables, $y_s(t^{i,k})$ is the vector of system instantaneous outputs, $f_s^{i,k}(\cdot)$ is the system state evolution function for the $\{i, k\}$ configuration, and $g_s^{i,k}(\cdot)$ is the system instantaneous output function for the $\{i, k\}$ configuration. The time variable $t^{i,k}$ is also indexed in order to highlight the fact that two different system configurations cannot coexist, therefore their time-axis must be different.

The set $\Omega_x^{i,k}(0^{i,k})$ represents the possible value of the system state variables at the time the system transitions into configuration $\{i, k\}$. Similarly, $\Omega_{w_s}^{i,k}(t^{i,k})$ represents the set of possible values for the system control inputs at time $t^{i,k}$ from the transition into the $\{i, k\}$ configuration. Thus, even if each system configuration $\{i, k\}$ is defined by a unique sequence of component operational mode transitions, there is uncertainty regarding the system states and the system inputs at the time of failure. This uncertainty will play a very important role in defining the failure coverage probability (see Section 3.4 for details) and thus the system stochastic behavior due to component failures, as will be explained in Section 3.4.

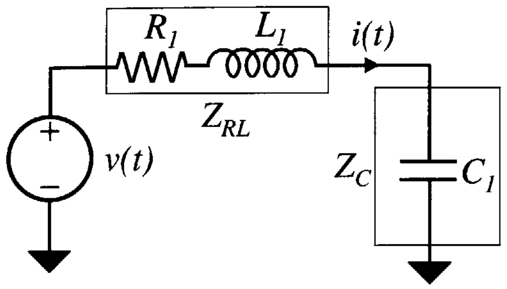


Figure 3.1: Series RLC Circuit Example.

Series RLC Circuit Example

To illustrate the concepts introduced in this section, let's consider the circuit displayed in Fig. 3.1, which results from interconnecting two impedances: Z_{RL} (corresponding to a resistor of value R_1 in series with an inductor of value L_1) and Z_C (corresponding to a capacitor of value C_1). Impedance Z_{RL} has an associated failure mode that results in a short-circuit in the resistor's terminals. Impedance Z_C can fail by modifying the value of the capacitor from C_1 to C_2. Thus, the dynamics of the failure-free configuration $\{1,0\}$ can be described by

$$\begin{aligned} \frac{d}{dt^{1,0}}\begin{bmatrix} i(t^{1,0}) \\ v_c(t^{1,0}) \end{bmatrix} &= \begin{bmatrix} -\frac{R_1}{L_1} & -\frac{1}{L_1} \\ \frac{1}{C_1} & 0 \end{bmatrix}\begin{bmatrix} i(t^{1,0}) \\ v_c(t^{1,0}) \end{bmatrix} + \begin{bmatrix} \frac{1}{L_1} \\ 0 \end{bmatrix} v(t^{1,0}) \\ x(0^{1,0}) &= [i(0^{1,0}), v_c(t^{1,0})]' \in \Omega_x^{1,0}(0^{1,0}) \\ v(t^{1,0}) &\in \Omega_v(t^{1,0}). \end{aligned} \tag{3.9}$$

The system configuration $\{1,1\}$ after the impedance Z_{RL} fails (the resistor's terminals are short-circuited) can be described by

$$\begin{aligned} \frac{d}{dt^{1,1}}\begin{bmatrix} i(t^{1,1}) \\ v_c(t^{1,1}) \end{bmatrix} &= \begin{bmatrix} 0 & -\frac{1}{L_1} \\ \frac{1}{C_1} & 0 \end{bmatrix}\begin{bmatrix} i(t^{1,1}) \\ v_c(t^{1,1}) \end{bmatrix} + \begin{bmatrix} \frac{1}{L_1} \\ 0 \end{bmatrix} v(t^{1,1}) \\ x(0^{1,1}) &= [i(0^{1,1}), v_c(t^{1,1})]' \in \Omega_x^{1,1}(0^{1,1}) \\ v(t^{1,1}) &\in \Omega_v(t^{1,1}). \end{aligned} \tag{3.10}$$

The system configuration $\{2,1\}$ after the impedance Z_C (the capacitance goes from C_1 to C_2) fails can be described by

$$\frac{d}{dt^{2,1}}\begin{bmatrix} i(t^{2,1}) \\ v_c(t^{2,1}) \end{bmatrix} = \begin{bmatrix} -\frac{R}{L} & -\frac{1}{L} \\ \frac{1}{C_2} & 0 \end{bmatrix}\begin{bmatrix} i(t^{2,1}) \\ v_c(t^{2,1}) \end{bmatrix} + \begin{bmatrix} \frac{1}{L} \\ 0 \end{bmatrix} v(t^{2,1})$$
$$x(0^{2,1}) = [i(0^{2,1}), v_c(t^{2,1})]' \in \Omega_x^{2,1}(0^{2,1})$$
$$v(t^{2,1}) \in \Omega_v(t^{2,1}). \tag{3.11}$$

For sequences of failures of size $k = 2$, there are two options: Z_C fails after Z_{RL} (corresponding to configuration $\{1,2\}$); or Z_{RL} fails after Z_C corresponding to configuration $\{2,2\}$). In both cases, the dynamics is described by the same differential equation, however the set of initial state-variables $\Omega_x^{1,2}(0^{1,2})$ (for configuration $\{1,2\}$); and $\Omega_x^{2,2}(0^{2,2})$ (for configuration $\{2,2\}$) may be different; thus

$$\frac{d}{dt^{1(2),2}}\begin{bmatrix} i(t^{1(2),2}) \\ v_c(t^{1(2),2}) \end{bmatrix} = \begin{bmatrix} -\frac{R_1}{L_1} & -\frac{1}{L_1} \\ \frac{1}{C_2} & 0 \end{bmatrix}\begin{bmatrix} i(t^{1(2),2}) \\ v_c(t^{1(2),2}) \end{bmatrix} + \begin{bmatrix} \frac{1}{L_1} \\ 0 \end{bmatrix} v(t^{1(2),2})$$
$$x(0^{1(2),2}) = [i(0^{1(2),2}), v_c(t^{1(2),2})]' \in \Omega_x^{1(2),2}(0^{1(2),2})$$
$$v(t^{1(2),2}) \in \Omega_v(t^{1(2),2}). \tag{3.12}$$

3.3 Performance Metrics, Requirements Definition, and System Evaluation

The system evaluation process is a forward search in the sense that the evaluation starts with the system in its failure-free configuration. Then single failures are introduced in the components and the resulting system configurations are evaluated to check if certain system dynamic properties, termed *Performance Metrics*, meet some predefined criteria, termed *Performance Requirements*. The configurations reached after a sequence of failures that do not meet the performance requirements are declared as *failed*, and no other system configurations can be reached from them. The configurations declared as *non-failed* meet the performance requirements, meaning that the system is still operational and can still perform the function for which it was designed. Other system configurations (after subsequent component failures) can be reached from them.

Performance metrics

Performance metrics are a set of m system-related properties, denoted by $Z_1, Z_2, \dots, Z_m$, that, for each system configuration $\{i, k\}$, can be computed by using the system dynamic equations (3.8). Performance metrics will quantify how well a system performs the function for which it was designed.

The performance metrics chosen to evaluate a system depend on the nature of the system. For example, in a tracking system, dynamic-related properties may be important, e.g., the tracking error, the overshoot, the settling time, or the pole and zero locations if the system is linear. If the system to be evaluated is a power system, performance metrics of interest may be the instantaneous (or the average) power consumption within the system, the maximum power delivered by a single power element, or the maximum current flowing through some components.

Performance Requirements

Performance requirements are defined through sets $\Phi^{i,k}_{Z_1}, \Phi^{i,k}_{Z_2}, \dots, \Phi^{i,k}_{Z_m}$, and represent the values that the performance metrics are allowed to take on each *non-failed* system configuration. For each system configuration $\{i, k\}$ reached after a sequence of failures of size k, the values of the performance metrics $Z_1, Z_2, \dots, Z_m$ will be checked to see if they are within the predefined performance requirements $\Phi^{i,k}_{Z_1}, \Phi^{i,k}_{Z_2}, \dots, \Phi^{i,k}_{Z_m}$.

Taking the circuit of Fig. 3.1 as an example, two performance metrics that may be relevant for assessing the performance of the system are the instantaneous power in resistor R_1 and the voltage across capacitor C_1, denoted by $v_c(t^*)$, where we are using t^* as a generic time-axis notation for any configuration. Let P_{max} be the maximum instantaneous power allowed in resistor R_1, and V_{max} the maximum allowed voltage across capacitor C. For any *non-failed* system configuration

$$\begin{aligned} p(t^*) &\leq P_{max} \\ v_c(t^*) &\leq V_{max}. \end{aligned} \tag{3.13}$$

Dynamic Performance Evaluation

To evaluate the system performance, it is necessary first to map the sets $\Phi_{Z_1}^{i,k}, \Phi_{Z_2}^{i,k}, \ldots, \Phi_{Z_m}^{i,k}$ defining the performance requirements onto the regions of the state space in which the system states must remain at any time in order to meet these requirements. It is important to note that sometimes is not straightforward to map sets of requirements onto the state space, particularly when the requirements depend on the system input, e.g., settling time.

Let $\Phi_{x(Z_1)}^{i,k}, \Phi_{x(Z_2)}^{i,k}, \ldots, \Phi_{x(Z_m)}^{i,k}$ be the regions of the state-space when the system is in configuration $\{i,k\}$, in which the requirements for the performance metrics $Z_1, Z_2, \ldots, Z_m$ (defined by the sets $\Phi_{Z_1}^{i,k}, \Phi_{Z_2}^{i,k}, \ldots, \Phi_{Z_m}^{i,k}$) are satisfied. Let $\Phi_x^{i,k} = \Phi_{x(Z_1)}^{i,k} \cap \Phi_{x(Z_2)}^{i,k} \cap \ldots \cap \Phi_{x(Z_m)}^{i,k}$. Let $s_{i,k}(t^{i,k})$ be an indicator variable that takes value 1 when the configuration $\{i,k\}$ is declared as *non-failed*, and 0 otherwise. Then

$$s_{i,k}(t^{i,k}) = \begin{cases} 1, & \text{if } x_s(t^{i,k}) \in \Phi_x^{i,k} \\ 0, & \textit{otherwise}, \end{cases} \tag{3.14}$$

where $x_s(t^{i,k})$ is the vector of system state variables. It is important to clarify the difference between t, which denotes the *global evaluation time*, i.e, the time the system may remain operational (system lifetime, preventive maintenance period or mission time), and the individual configuration time-axis $t^{i,k}$, which is a subset of t ($t^{i,k} \subseteq t$). The shape of $\Phi_x^{i,k}$ depends on the sets $\Phi_{Z_1}^{i,k}, \Phi_{Z_2}^{i,k}, \ldots, \Phi_{Z_m}^{i,k}$ that define the performance requirements. An example of this for a two-dimensional case is shown in Fig. 3.2, where $\Phi_x^{i,k}$ is defined as a rectangular box.

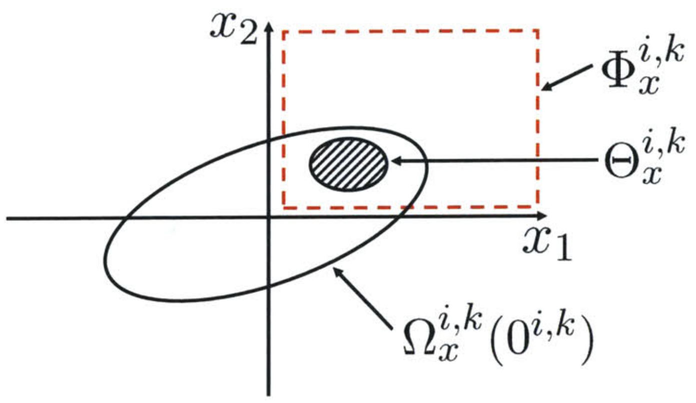


Figure 3.2: Two-dimensional example for $\Phi_x^{i,k}$, $\Theta_x^{i,k}$, and $\Omega_x^{i,k}(0^{i,k})$.

3.4 System Stochastic-Behavior Model

The transitions between configurations occur stochastically and are triggered by random operational mode transitions within the system components. After the system randomly transitions into configuration $\{i,k\}$ (from configuration $\{j,k-1\}$ due to a component failure) two outcomes are possible. As stated before, if the system state variables remain within the allowed region $\Phi_x^{i,k}$ dictated by the performance requirements at all times, then the configuration is declared as *non-failed;* if there are transient or permanent excursions outside these regions, then the configuration is declared as *failed.* The probability of declaring configuration $\{i,k\}$ *non-failed* or *failed* is a function of the failure coverage probability, i.e., the probability that the system will be able to detect and isolate the failure, and reconfigure itself so the system state variables remain within the region $\Phi_x^{i,k}$.

Failure Coverage Probability Definition

The coverage probability is defined as the ratio of the volume of the subset of possible initial conditions that results in trajectories which are contained in the set of "acceptable" states to the volume of the set of possible initial conditions (i.e., the conditions at the time of transition). Let $\Omega_x^{i,k}(0^{i,k})$ define the set of possible state variable values at the time of failure when the system transitions from configuration $\{j,k-1\}$ into configuration $\{i,k\}$. Let $\Theta_x^{i,k} \subseteq \Omega_x^{i,k}(0^{i,k})$ such that if $x_s(0^{i,k}) \in \Theta_x^{i,k}$, then $x_s(t^{i,k}) \in \Phi_x^{i,k}$ for all $t^{i,k} \geq 0$ (an example of $\Theta_x^{i,k} and\ \Omega_x^{i,k}(0^{i,k})$, for a two-dimensional case is shown in Fig. 3.2). Then, the coverage probability $c_{j,k-1}^{i,k}$ when the system is in configuration $\{j,k-1\}$ and transitions to configuration $\{i,k\}$ is defined by

$$c_{j,k-1}^{i,k} = \frac{\text{vol}(\Theta_x^{i,k})}{\text{vol}(\Omega_x^{i,k}(0^{i,k}))}. \tag{3.15}$$

Stochastic Behavior of Configuration $\{i,k\}$

It is important to note that each system configuration $\{i,k\}$ generates two states in the Markov model. One of these states is transient ({2i-1,k}), and corresponds to the configuration $\{i,k\}$ being declared as *non-failed,* where the other one ({2i,k}), which is absorbing, corresponds to the same configuration $\{i,k\}$ being declared as *failed.*

Let $X(t)$ denote the system configuration at time t. Let $p_{2i-1,k}(t)$ be the probability that, at any time $t \geq 0$, the configuration $\{i,k\}$ is declared as *non-failed*, conditioned on the fact that the system was in the failure-free configuration $\{1,0\}$ at $t = 0$ (with probability 1). Then

$$p_{2i-1,k}(t) = P(X(t) = \{i,k\}, s_{i,k}(t^{i,k}) = 1 | X(0) = \{1,0\}). \tag{3.16}$$

Similarly, let $p_{2i,k}(t)$ be the probability that, at a time $t > 0$, the configuration $\{i,k\}$ is declared as *failed*, conditioned on the fact that the system was in the failure-free configuration $\{1,0\}$ at $t = 0$. Then

$$p_{2i,k}(t) = P(X(t) = \{i,k\}, s_{i,k}(t^{i,k}) = 0 | X(0) = \{1,0\}). \tag{3.17}$$

From (3.16) and (3.17), it follows that:

$$P(X(t) = \{i,k\} | s_{1,0}(0) = 1) = p_{2i-1,k}(t) + p_{2i,k}(t). \tag{3.18}$$

Let $c^{i,k}_{j,k-1}$ be the failure coverage probability when the system is in configuration $\{j, k-1\}$ and transitions to configuration $\{i,k\}$. Let $\lambda^{i,k}_{j,k-1}$ be the transition rate (associated with the transition rate between two component operational modes) that causes the system to go from from configuration $\{j, k-1\}$ to configuration $\{i,k\}$.

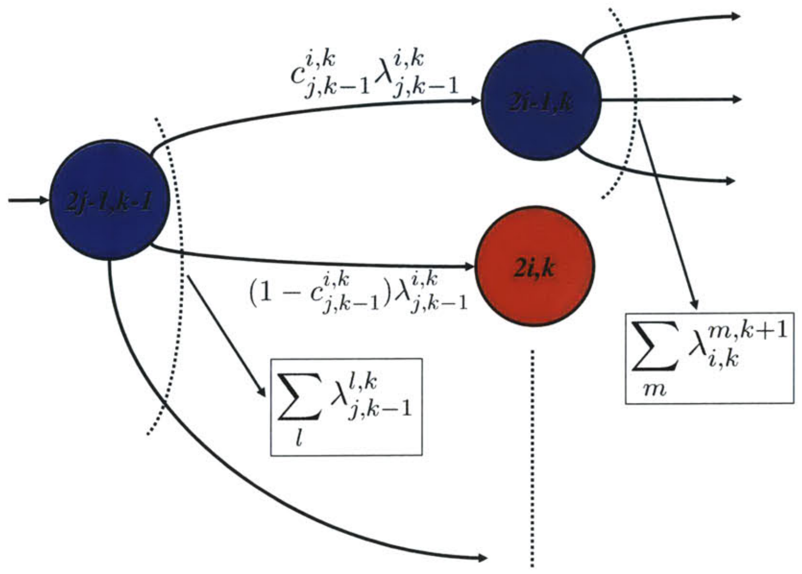

$c^{i,k}_{j,k-1}\lambda^{i,k}_{j,k-1}$
2i-1,k
2j-1,k-1
2i,k
$(1-c^{i,k}_{j,k-1})\lambda^{i,k}_{j,k-1}$
$\sum_l \lambda^{l,k}_{j,k-1}$
$\sum_m \lambda^{m,k+1}_{i,k}$

Figure 3.3: Stochastic behavior of the non-failed system configuration (i,k) when a failure occurs in component m.

We assume that the system evolves between configurations in a Markovian fashion, i.e., the system configurations at future times will only depend on the configuration at the present time. Therefore the Chapman-Kolmogorov equations can be used to compute the probabilities of each system configuration status (failed or non-failed). Then the stochastic behavior of system configuration $\{i, k\}$, displayed in Fig. 3.3, can be represented by

$$\frac{d}{dt}\begin{bmatrix} P_{2j-1,k-1} \\ P_{2i-1,k} \\ P_{2i,k} \end{bmatrix} = \begin{bmatrix} -\sum_{l} \lambda_{j,k-1}^{l,k} & 0 & 0 \\ c_{j,k-1}^{i,k}\lambda_{j,k-1}^{i,k} & -\sum_{m} \lambda_{i,k}^{m,k+1} & 0 \\ (1-c_{j,k-1}^{i,k})\lambda_{j,k-1}^{i,k} & 0 & 0 \end{bmatrix} \begin{bmatrix} P_{2j-1,k-1} \\ P_{2i-1,k} \\ P_{2i,k} \end{bmatrix}. \tag{3.19}$$

The state-transition matrix associated with the Markov model that governs the transitions between all system configurations is obtained by assembling the blocks (3.19) corresponding to each configuration $\{i, k\}$. Let A be the state-transition matrix associated with the transitions between all system configurations. Then the system configurations' probability vector $P(t)$, can be computed by solving

$$\begin{aligned} \tfrac{dP(t)}{dt} &= AP(t) \\ P(0) &= [1\,0\,0\,...\,0]'. \end{aligned} \tag{3.20}$$

Each component $p_{2i-1,k}(t)$ of $P(t)$ corresponds to the probability that the system is in configuration $\{i, k\}$ and it has been declared *non-failed*, given that the system was in configuration $\{1, 0\}$ at time $t = 0$. Similarly, $p_{2i,k}(t)$ corresponds to the probability that the same configuration $\{i, k\}$ is declared *failed*, given that the system was in configurations $\{1, 0\}$ at time $t = 0$ with probability $p_{1,0}(0) = 1$.

The state-transition matrix A is a random matrix with the following properties:

1. It is a lower triangular matrix.
2. The columns add up to zero.
3. The diagonal elements are negative or zero.
4. The off-diagonal elements are zero or positive.
5. It is a diagonally-dominant matrix.

6. The columns and rows of A can be rearranged to obtain an equivalent matrix A^* [24]:

$$A^* = \begin{bmatrix} M_R & 0 \\ M_Q & 0 \end{bmatrix}, \tag{3.21}$$

where matrix M_R is associated with the transient states and M_Q is associated with the absorbing states.

3.4.1 On the Computation of Failure Coverage Probabilities

As mentioned before, the probability of declaring configuration $\{i,k\}$ *non-failed* or *failed* depends on the failure coverage probability. In this section, methods for computing the failure coverage probability are introduced. First, its computation through Monte-Carlo simulation is discussed. This technique can be applied to any system. Then analytical techniques for computing the failure coverage probability in LTI systems are introduced.

This approach is based on the definition of *non-failed* configuration, which states that a configuration $\{i,k\}$ is declared as *non-failed* when the system state variables remain at all time within the "allowed" region $\Phi_x^{i,k}$ of the state-space dictated by the performance requirements. Thus, the coverage probability will be defined by the set of reachable state variables at the time of transition $\Omega_x^{i,k}(0^{i,k})$, and by a subset of that set of initial conditions $\Theta_x^{i,k}$. This subset $\Theta_x^{i,k}$ is such that, if the system state variables are contained inside, then the trajectories followed by the system are to remain, at all times, within the "allowed" region $\Phi_x^{i,k}$ dictated by the performance requirements.

Failure Coverage Probability Computation Through Simulation

To compute the coverage probability $c_{j,k-1}^{i,k}$ when the system transitions from configuration $\{j,k-1\}$ to configuration $\{i,k\}$, it is necessary to obtain the set of possible state variable values $\Omega_x^{i,k}(0^{i,k})$ at the time of transition, and also the "acceptable" set of initial conditions $\Theta_x^{i,k}$.

Set of possible state variable values $\Omega_x^{i,k}(0^{i,k})$ at the time of transition. Let $\Omega_x^{j,k-1}(\infty)$ define the steady-state set of reachable states when the system is in the configuration $\{j,k-1\}$ and this configuration was declared as *non-failed*. This set defines the set of possible initial conditions $\Omega_x^{j,k}(0^{i,k})$ for system configuration $\{j,k\}$:

$$\Omega_x^{i,k}(0^{i,k}) \equiv \Omega_x^{j,k-1}(\infty). \tag{3.22}$$

"acceptable" set of initial conditions $\Theta_x^{i,k}$. This set can be mapped through Monte-Carlo simulation by randomly taking initial conditions $x_s(0^{i,k})$ from the the set of possible state variables at the time of transition $\Omega_x^{i,k}(0^{i,k})$. This technique has been used before to map domains of attraction in non-linear systems [21]. The principles are the same, except that in this case the attractor is replaced by the "acceptable" region of the state-space $\Phi_x^{i,k}$. Thus, for every $x_s(0^{i,k}) \in \Theta_x^{i,k}$

$$x_s(t^{i,k}) \in \Phi_x^{i,k} \, \forall \, t^{i,k} \geq 0. \tag{3.23}$$

It is important to note that for computing $\Omega_x^{i,k}(0^{i,k})$, it is necessary to know $\Omega_x^{j,k-1}(\infty)$ (which is computed from $\Omega_x^{j,k-1}(0^{j,k-1})$) and $\Theta_x^{j,k-1}$. Therefore $\Omega_x^{i,k}(0^{i,k})$ and $\Theta_x^{i,k}$ can be obtained by recurrence given that $\Omega_x^{1,0}(0^{1,0})$ is known.

Analytical Calculation of Failure Coverage Probability in LTI Systems

For LTI systems, it is possible to analytically obtain bounds on the coverage probability. This is possible due to the fact that for each system configuration $\{i,k\}$, it is possible to analytically obtain bounding ellipsoids for $\Omega_x^{i,k}(0^{i,k})$ and $\Theta_x^{i,k}$. In the remainder of this section, we will illustrate the process of obtaining the failure coverage probability for a configuration reached after one component failure, i.e., a configuration with index $\{1,1\}$. Let the dynamics of a system in the failure-free configuration $\{1,0\}$ be represented by

$$\begin{aligned} \tfrac{dx(t^{1,0})}{dt} &= F^{1,0}x(t^{1,0}) + G^{1,0}w(t^{1,0}) \\ x(0^{1,0}) &\in \Omega_x(0^{1,0}), \; with \; \Omega_x(0^{1,0}) = \{x : x'(\Psi^{1,0})^{-1}x \leq 1\} \\ w(t^{1,0}) &\in \Omega_w(t^{1,0}), \; with \; \Omega_w(t^{1,0}) = \{w : w'Q^{-1}w \leq 1\}, \end{aligned} \tag{3.24}$$

where $F^{1,0}$ is the system state-transition matrix, and $G^{1,0}$ is the system input matrix, both for configuration $\{1,0\}$. $\Psi^{1,0}$ and Q are positive definite matrices and the matrix inequalities in (3.24) define ellipsoids.

Let $\Phi_x^{1,0}$ be the region of the state-space that contains the states that meet the functional performance requirements of the system. Let $\Omega_x^{1,0}(t^{1,0})$ denote the set of reachable states, such that $\Omega_x^{1,0}(t^{1,0}) \subseteq \Phi x^{1,0}$. A bounding ellipsoid $\Omega_{x,b}^{1,0}(t^{1,0}) = \{x : x'(\Gamma^{1,0}(t^{1,0}))^{-1}x \leq 1\}$, such that $\Omega_x^{1,0}(t^{1,0}) \subseteq \Omega_{x,b}^{1,0}(t^{1,0}) \subseteq \Omega_r^{1,0}$, can be computed by solving [50]

$$\begin{aligned} \tfrac{d\Gamma^{1,0}(t^{1,0})}{dt} &= F^{1,0}\Gamma(t^{1,0}) + \Gamma^{1,0}(t^{1,0})(F^{1,0})' + \beta^{1,0}\Gamma^{1,0}(t^{1,0}) + \tfrac{1}{\beta^{1,0}}G^{1,0}Q(G^{1,0})' \\ \Gamma(0^{1,0}) &= \Psi \end{aligned} \tag{3.25}$$

with $0 \leq \beta^{1,0} \leq 1$. Let $\Gamma_{ss}^{1,0}$ denote the steady-state value of $\Gamma^{1,0}(t^{1,0})$, thus $\Omega_{x,b}^{1,0}(\infty) = \{x : x(\Gamma_{ss}^{1,0})^{-1}x \leq 1\}$, where $\Gamma_{ss}^{1,0}$ can be computed by solving

$$0 = F^{1,0}\Gamma_{ss}^{1,0} + \Gamma_{ss}^{1,0}(F^{1,0})' + \beta^{1,0}\Gamma_{ss}^{1,0} + \tfrac{1}{\beta^{1,0}}G^{1,0}Q(G^{1,0})'. \tag{3.26}$$

$\Omega_{x,b}^{1,0}(\infty)$ contains the set of reachable states after the "initial transient" settles. This "initial transient" has nothing to do with possible transients of x within the system dynamics due to changes in the input $w(t^{1,0})$, i.e., $\Omega_{x,b}^{1,0}(\infty)$ takes into account these transients of x.

Decoupling Assumption. It is assumed that the system dynamics time constants are much smaller that the time constants dictated by the failure rates. In this scenario, the likelihood of two components failing within a time on the order of magnitude of the system dynamic time constants is negligible relative to the likelihood of just one component failing. Therefore, the steady-state set of reachable states $\Omega_x^{1,0}(\infty) \subseteq \Omega_{x,b}^{1,0}(\infty)$ is reached before another failure occurs.

Under the above assumption, another failure occurs within the system, and thus the system transitions from configuration $\{1,0\}$ to configuration $\{1,1\}$. The dynamics of the new configuration is defined by

$$\begin{aligned} \tfrac{dx(t^{1,1})}{dt} &= F^{1,1}x(t^{1,1}) + G^{1,1}w(t^{1,1}) \\ x(0^{1,1}) &\in \Omega_x^{1,1}(0^{1,1}),\ \textit{with}\ \Omega_x^{1,1}(0^{1,1}) \equiv \Omega_{x,b}^{1,0}(\infty) = \{x : x'(\Gamma_{ss}^{1,0})^{-1}x \leq 1\} \\ w(t^{1,1}) &\in \Omega_w(t^{1,1}),\ \textit{with}\ \Omega_w(t^{1,1}) = \{w : w'Q^{-1}w \leq 1\}, \end{aligned} \tag{3.27}$$

where $F^{1,1}$ is the system state-transition matrix, and $G^{1,1}$ is the system input matrix, both for configuration $\{1,1\}$. As before, $\Gamma_{ss}^{1,0}$ and Q are positive definite matrices. It is extremely important to note that the new set of initial conditions $\Omega_x^{1,1}(0^{1,1})$ is defined by the bounding ellipsoid $\Omega_{x,b}^{1,0}$ for the steady-state of reachable states before the transition.

Let $\Omega_x^{1,1}(t^{1,1})$ denote the new set of reachable states at time $t^{1,1}$. A new bounding ellipsoid $\Omega_{x,b}^{1,1}(t) = \{x : x'\Gamma^{1,1}(t^{1,1})^{-1}x \leq 1\}$, such that $\Omega_x^{1,1}(t^{1,1}) \subseteq \Omega_{x,b}^{1,1}(t^{1,1})$, can be computed by solving

$$\frac{d\Gamma^{1,1}(t^{1,1})}{dt} = F^{1,1}\Gamma^{1,1}(t^{1,1}) + \Gamma(t^{1,1})^{1,1}(F^{1,1})' + \beta^{1,1}\Gamma^{1,1}(t^{1,1}) + \frac{1}{\beta^{1,1}}G^{1,1}Q(G^{1,1})'$$
$$\Gamma^{1,1}(0^{1,1}) = \Gamma^{1,0}_{ss} \tag{3.28}$$

with $0 \leq \beta^{1,1} \leq 1$. Let $\Phi_x^{1,1}$ be the region of the state-space that contains the states that meet the functional performance requirements of the system in this new configuration after the failure has occurred. Then, depending on $\Omega_{x,b}^{1,1}(t^{1,1})$ and $\Phi_x^{1,1}$, we have the following cases:

1. If $\Omega_{x,b}^{1,1}(\infty) \subseteq \Phi_x^{1,1}$, the coverage probability is:

$$c_{1,0}^{1,1} = \frac{\text{vol}(\Omega_{x,b}^{1,1}(t^{1,1}) \cap \Omega_{x,b}^{1,1}(0^{1,1}))}{\text{vol}(\Omega_{x,b}^{1,1}(0^{1,1}))}. \tag{3.29}$$

where $t_1^{1,1} > 0$ is such that $\Omega_{x,b}^{1,1}(t^{1,1}) \subseteq \Phi_x^{1,1}\ \forall t^{1,1} \geq t_1^{1,1}$. The following extremes cases are possible:

(i) if $t_1^{1,1} = 0$, then the coverage probability is:

$$c_{1,0}^{1,1} = 1. \tag{3.30}$$

In this case, the set of reachable states at any time is fully contained within the state-space region defined by the performance requirements. Therefore, the system always survives this failure. This situation can be visualized in Fig 3.4 for a two-dimensional case and two different situations. The ruled area represents the region of the set of all possible initial conditions $\Omega_{x,b}^{1,1}(0^{1,1})$ that will result in trajectories fully contained in the set $\Phi_x^{1,1}$ defined by the performance requirements.

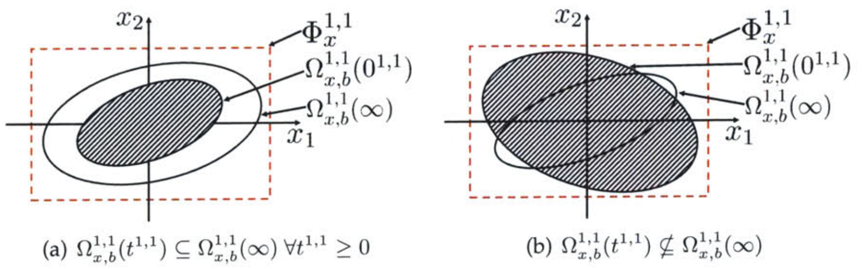

x_2
$\Phi_x^{1,1}$
$\Omega_{x,b}^{1,1}(0^{1,1})$
$\Omega_{x,b}^{1,1}(\infty)$
x_1
(a) $\Omega_{x,b}^{1,1}(t^{1,1}) \subseteq \Omega_{x,b}^{1,1}(\infty)\ \forall t^{1,1} \geq 0$
(b) $\Omega_{x,b}^{1,1}(t^{1,1}) \not\subseteq \Omega_{x,b}^{1,1}(\infty)$

Figure 3.4: Bounding ellipsoid $\Omega_{x,b}^{1,1}(t^{1,1}$ and region $\Phi_x^{1,1}$ of the state-space that contains the states that meet the functional performance requirements of the system for a two-dimensional for the case when $\Omega_{x,b}^{1,1}(\infty) \subseteq \Phi_x^{1,1}$, and $\Omega_{x,b}^{1,1}(t^{1,1}$ is such that $c_{1,0}^{1,1} = 1$.

(ii) if $t_1^{1,1}$ is very large, a lower bound of the coverage probability is:

$$c_{1,0}^{1,1} = \frac{\text{vol}(\Omega_{x,b}^{1,1}(\infty))}{\text{vol}(\Omega_{x,b}^{1,1}(0^{1,1}))}. \tag{3.31}$$

In this case, although the set of reachable states may not be fully contained within the state-space region defined by the performance requirements at all times, the steady-state set of reachable states is. The steady-state set of reachable states is an invariant set with respect to the system dynamics [51]. Therefore, the trajectories with initial conditions within the steady-state set of reachable states will remain within this set. This situation is depicted for a two-dimensional case in Fig. 3.5, where the ruled area (the region of the set of all possible initial conditions $\Omega_{x,b}^{1,1}(0^{1,1})$ that will result in trajectories fully contained in the set $\Phi_x^{1,1}$ defined by the performance requirements) is now $\Omega_{x,b}^{1,1}(\infty)$.

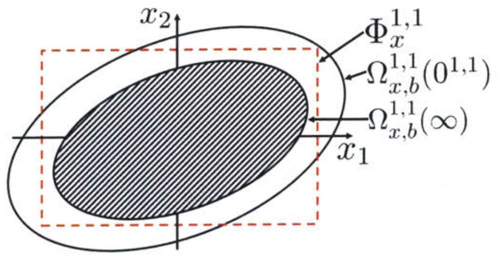


Figure 3.5: Bounding ellipsoid $\Omega_{x,b}^{1,1}(t^{1,1}$ and region $\Phi_x^{1,1}$ of the state-space that contains the states that meet the functional performance requirements of the system for a two-dimensional for the case when $\Omega_{x,b}^{1,1}(\infty) \subseteq \Phi_x^{1,1}$ and the value of $c_{1,0}^{1,1}$ given by 3.31.

2. If $\Omega_{x,b}^{1,1}(\infty) \nsubseteq \Phi_x^{1,1}$, then:

(i) if $\Omega_{x,b}^{1,1}(\infty) \cap \Phi_x^{1,1} = \emptyset$, the coverage probability is

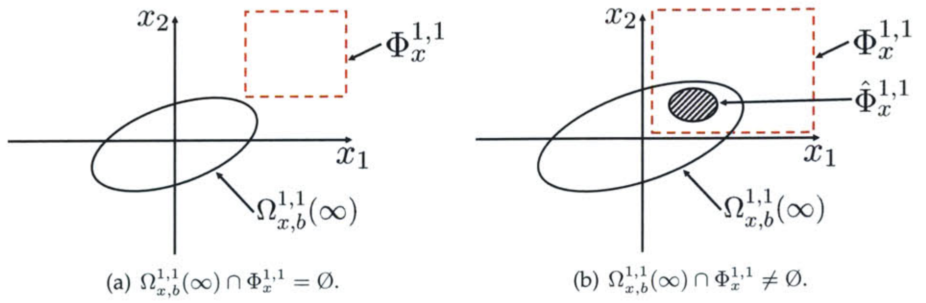
$$c_{1,0}^{1,1} = 0. \tag{3.32}$$

In this case, the steady-state set of reachable states is outside the state-space region defined by the performance requirements. Therefore, the system never survives this failure. This situation is depicted for a two-dimensional case in Fig. 3.6(a), where it can be seen that there is no overlap between $\Omega_{x,b}^{1,1}(\infty)$ and $\Phi_x^{1,1}$.

(ii) if $\Omega_{x,b}^{1,1}(\infty) \cap \Phi_x^{1,1} \neq \emptyset$, a lower bound for the coverage probability is

$$c_{1,0}^{1,1} = \frac{\mathrm{vol}(\hat{\Phi}_x^{1,1} \cap \Omega_{x,b}^{1,1}(0^{1,1}))}{\mathrm{vol}(\Omega_{x,b}^{1,1}(0^{1,1}))}. \tag{3.33}$$

where $\hat{\Phi}_x^{1,1}$ is the largest invariant set with respect to the dynamics of configuration $\{1,1\}$, such that $\hat{\Phi}_x^{1,1} \subseteq \Omega_{x,b}^{1,1}(\infty) \cap \Phi_x^{1,1}$. This is the most difficult case, as it is difficult to find $\hat{\Phi}_x^{1,1}$. Analytical techniques for computing invariant sets with respect to LTI systems are discussed in [51]. This situation is depicted in Fig. 3.6(b).



(a) $\Omega_{x,b}^{1,1}(\infty) \cap \Phi_x^{1,1} = \emptyset$. (b) $\Omega_{x,b}^{1,1}(\infty) \cap \Phi_x^{1,1} \neq \emptyset$.

Figure 3.6: Bounding ellipsoid $\Omega_{x,b}^{1,1}(t^{1,1}$ and region $\Phi_x^{1,1}$ of the state-space that contains the states that meet the functional performance requirements of the system for a two-dimensional for the case when $\Omega_{x,b}^{1,1}(\infty) \nsubseteq \Phi_x^{1,1}$.

3.5 A Series RL Circuit Example

The purpose of this section is to illustrate the concepts introduced in previous sections of this chapter. Let's consider the series RL circuit displayed in Fig. 3.7, and let's assume that:

1. The analysis is constrained to the effect of resistor failures on the current flowing through the inductor.
2. The resistors can only fail open circuit with a failure rate of λ_{R_1} and λ_{R_2}, respectively.
3. The initial current $i(0)$ flowing through the circuit is unknown, but it is such that $|i(0)| \leq \Psi^{1/2}$.
4. The voltage source $v(t)$ is unknown, but it is such that $|v(t)| \leq Q^{1/2}$.

5. The maximum current that resistors R_1 and R_2 can process is $i_{max}^{R_1}$ and $i_{max}^{R_2}$, respectively, and once this current is reached the resistors fail open.

6. Q, R_1, R_2, $i_{max}^{R_1}$, and $i_{max}^{R_2}$ are such that $\sqrt{\frac{Q}{R_2^2}} < i_{max}^{R_2} < \sqrt{\frac{Q}{R_{eq}^2}}$, and $i_{max}^{R_1} > \sqrt{\frac{Q}{R_{eq}^2}}$, where $\sqrt{\frac{Q}{R_{eq}^2}}$ is the maximum DC current that can flow through the circuit when both resistors are operational. Similarly, $\sqrt{\frac{Q}{R_2^2}}$ is the maximum DC current that can flow through the circuit when resistor R_1 is failed.

7. The system fails with probability 1 if both resistors fail open.

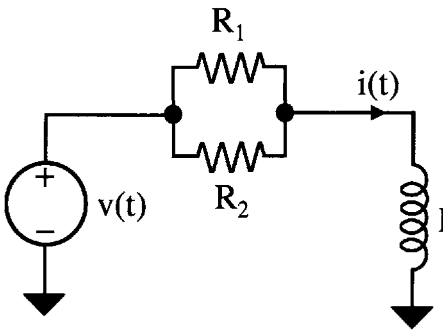


Figure 3.7: Series RL Circuit.

Failure-Free Dynamic Behavior

Before any failure, the current $i(t^{1,0})$ can be computed by solving

$$\begin{aligned} \frac{di(t^{1,0})}{dt^{1,0}} &= -\frac{R_{eq}}{L} i(t^{1,0}) + \frac{1}{L} v(t^{1,0}) \\ i(0^{1,0}) &\in \Omega_i(0^{1,0}), \ with\ \Omega_i^{1,0}(0^{1,0}) = \{i(0^{1,0}) : |i(0^{1,0})| \leq \Psi^{1/2}\} \\ v(t^{1,0}) &\in \Omega_v(t^{1,0}), \ with\ \Omega_v(t^{1,0}) = \{v(t^{1,0}) : |v(t^{1,0})| \leq Q^{1/2}\} \end{aligned} \tag{3.34}$$

where $R_{eq} = \frac{R_1 R_2}{R_1+R_2}$. Let $\Phi_i^{1,0} = \{i(t^{1,0}) : |i(t^{1,0})| \leq i_{max}^{1,0}\}$ be the set of acceptable current flowing through the circuit, where $i_{max}^{1,0}$ is the maximum current flowing through the circuit such that the current flowing through each resistor is less that its maximum allowed current. Let $\Omega_i^{1,0}(t^{1,0})$ denote the set of all possible current at time $t^{1,0}$, such that $\Omega_i^{1,0}(t^{1,0}) \subseteq \Phi_i^{1,0}$. A bounding ellipsoid $\Omega_{i,b}^{1,0}(t^{1,0}) = \{i : i'\Gamma^{1,0}(t^{1,0})^{-1} i \leq 1\}$, such that

$\Omega_i^{1,0}(t^{1,0}) \subseteq \Omega_{i,b}^{1,0}(t^{1,0}) \subseteq \Phi_i^{1,0}$, can be computed by solving

$$\frac{d\Gamma^{1,0}(t^{1,0})}{dt^{1,0}} = -2\frac{R_{eq}}{L}\Gamma^{1,0}(t^{1,0}) + \beta\Gamma^{1,0}(t^{1,0}) + \frac{R_{eq}}{\beta^{1,0}L^2}Q$$
$$\Gamma^{1,0}(0^{1,0}) = \Psi. \tag{3.35}$$

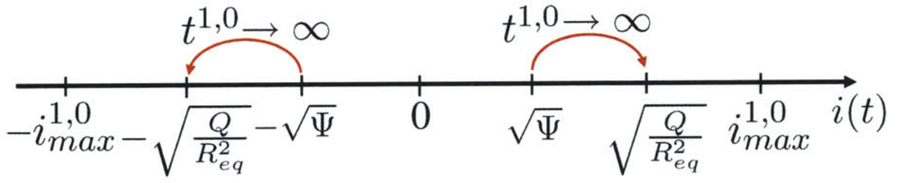
where $\Gamma^{1,0}(t^{1,0})$ is a positive number, and $0 \leq \beta^{1,1} \leq 1$. By taking $\beta^{1,0} = R_{eq}/L$, which is the value that minimizes $\Gamma_{ss}^{1,0}$ [50], the solution to (3.35) is given by

$$\Gamma^{1,0}(t^{1,0}) = \frac{Q}{R_{eq}^2} + (\Psi - \frac{Q}{R_{eq}^2})e^{-\frac{R_{eq}}{L}t^{1,0}}, \tag{3.36}$$

and the steady-state $\Gamma_{ss}^{1,0}$ value is given by

$$\Gamma_{ss}^{1,0} = \Gamma^{1,0}(\infty) = \frac{Q}{R_{eq}^2}. \tag{3.37}$$

Figure 3.8 represents the evolution of the interval in which lies the current flowing through the circuit. It is important to note that $\sqrt{\Gamma_{ss}} = \sqrt{\frac{Q}{R_{eq}^2}} < i_{max}^{1,0}$. Thus, even if there is uncertainty in the value of $i(t)$, it will always be within the maximum limits requirement.



$t^{1,0} \longrightarrow \infty$
$t^{1,0} \longrightarrow \infty$
$-i_{max}^{1,0}$
$-\sqrt{\frac{Q}{R_{eq}^2}}$
$-\sqrt{\Psi}$
0
$\sqrt{\Psi}$
$\sqrt{\frac{Q}{R_{eq}^2}}$
$i_{max}^{1,0}$
$i(t)$

Figure 3.8: Series RL Circuit current evolution for $\Psi < \frac{Q}{R_{eq}^2}$.

Failure Coverage Probability Computation

Now, let's assume $\Omega_{i,b}^{1,0}(\infty)$ has been reached and resistor R_1 fails open circuit. The current flowing through the circuit is now governed by

$$\frac{di(t^{1,1})}{dt^{1,1}} = -\frac{R_2}{L}i(t^{1,1}) + \frac{1}{L}v(t^{1,1})$$
$$i(0^{1,1}) \in \Omega_i(0^{1,1}),\ with\ \Omega_i^{1,1}(0^{1,1}) = \{i(0^{1,1}) : |i(0^{1,1})| \leq (\frac{Q}{R_{eq}^2})^{1/2}\}$$
$$v(t^{1,1}) \in \Omega_v(t^{1,1}),\ with\ \Omega_v(t^{1,1}) = \{v(t^{1,1}) : |v(t^{1,1})| \leq Q^{1/2}\}. \tag{3.38}$$

It is important to note that the new initial condition for $\Omega_i^{1,1}(0^{1,1})$ is the steady-state value $\Omega_i^{1,0}(\infty)$ before the failure occur.

Let $\Phi_i^{1,1} = \{i(t^{1,1}) : |i(t^{1,1})| \leq i_{max}^{1,1}\}$, with $i_{max}^{1,1} = i_{max}^{R_1} < i_{max}^{1,0}$, be the new set that contains the acceptable current flowing through the circuit. It is easy to understand why the maximum current flowing through the circuit is smaller now: there is only one resistor in series with the inductor. Let $\Omega_i^{1,1}(t^{1,1})$ denote the new set of reachable states.

A new bounding ellipsoid $\Omega_{i,b}^{1,1}(t^{1,1}) = \{i : i'\Gamma^{1,1}(t^{1,1})^{-1}i \leq 1\}$, such that $\Omega_i^{1,1}(t^{1,1}) \subseteq \Omega_{i,b}^{1,1}(t^{1,1})$, is given by

$$\Gamma^{1,1}(t^{1,1}) = \frac{Q}{R_2^2} + (\frac{Q}{R_{eq}^2} - \frac{Q}{R_2^2})e^{-\frac{R_2}{L}t^{1,1}}. \tag{3.39}$$

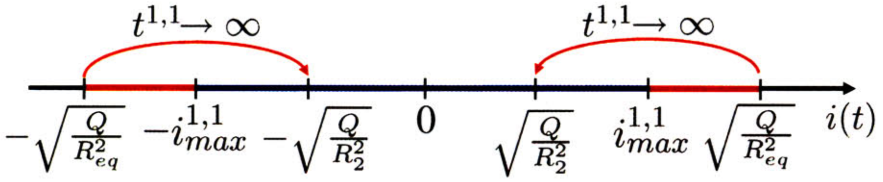

$t^{1,1} \rightarrow \infty$
$t^{1,1} \rightarrow \infty$
$-\sqrt{\frac{Q}{R_{eq}^2}}$
$-i_{max}^{1,1}$
$-\sqrt{\frac{Q}{R_2^2}}$
0
$\sqrt{\frac{Q}{R_2^2}}$
$i_{max}^{1,1}$
$\sqrt{\frac{Q}{R_{eq}^2}}$
$i(t)$

Figure 3.9: Series RL Circuit current evolution after R_1 fails open circuit for Q, R_1, R_2, and $i_{max}^{1,1}$ such that $\sqrt{\frac{Q}{R_2^2}} < i_{max}^{1,1} < \sqrt{\frac{Q}{R_{eq}^2}}$.

By assumption (6), we have that $\sqrt{\frac{Q}{R_2^2}} < i_{max}^{1,1} < \sqrt{\frac{Q}{R_{eq}^2}}$. Figure 3.9 represents this situation. If $i(0^{1,1})$ takes values within the intervals in red, then the system will fail; on the contrary, if $i(0^{1,1})$ takes values within the interval in blue, the circuit will survive the failure of resistor R_1. In this case, this is always true because of the dissipative nature of this system. One of the underlying assumptions is that the reachable currents before and after the failure are uniform distributed. Thus, the computation of the coverage probability is as follows:

$$c_{1,0}^{1,1} = \frac{i_{max}^{1,1} - (-i_{max}^{1,1})}{\sqrt{\frac{Q}{R_{eq}^2}} - (-\sqrt{\frac{Q}{R_{eq}^2}})} = i_{max}^{1,1}\sqrt{\frac{R_{eq}^2}{Q}}. \tag{3.40}$$

Similarly, the following coverage probabilities are obtained:

$$c_{1,0}^{2,1} = 1, \tag{3.41}$$
$$c_{1,1}^{1,2} = 0, \tag{3.42}$$
$$c_{2,1}^{2,2} = 0. \tag{3.43}$$

The state transition diagram associated with the Markov reliability model is displayed in Fig. 3.10, and can be described by

$$\frac{d}{dt}\begin{bmatrix} p_{1,0} \\ p_{1,1} \\ p_{2,1} \\ p_{3,1} \\ p_{2,2} \\ p_{4,2} \end{bmatrix} = \begin{bmatrix} -\lambda_{R_1}-\lambda_{R_2} & 0 & 0 & 0 & 0 & 0 \\ i_{max}^{1,1}\sqrt{\frac{R_{eq}^2}{Q}}\lambda_{R_1} & -\lambda_{R_2} & 0 & 0 & 0 & 0 \\ (1-i_{max}^{1,1}\sqrt{\frac{R_{eq}^2}{Q}})\lambda_{R_1} & 0 & 0 & 0 & 0 & 0 \\ \lambda_{R_2} & 0 & 0 & -\lambda_{R_1} & 0 & 0 \\ 0 & \lambda_{R_2} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \lambda_{R_1} & 0 & 0 \end{bmatrix} \begin{bmatrix} p_{1,0} \\ p_{1,1} \\ p_{2,1} \\ p_{3,1} \\ p_{2,2} \\ p_{4,2} \end{bmatrix}. \tag{3.44}$$

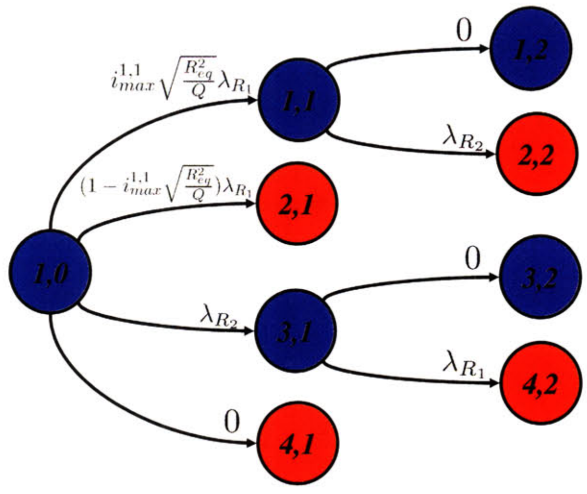

$i_{max}^{1,1}\sqrt{\frac{R_{eq}^2}{Q}}\lambda_{R_1}$
$(1-i_{max}^{1,1}\sqrt{\frac{R_{eq}^2}{Q}})\lambda_{R_1}$
λ_{R_2}
0
λ_{R_2}
0
λ_{R_1}
0
1,0
1,1
2,1
3,1
4,1
1,2
2,2
3,2
4,2

Figure 3.10: State transition diagram associated with the Markov reliability model for the series RL circuit case-study.

3.6 Probabilistic Measures of Performance and Reliability

The performance metrics $Z_1, Z_2, \ldots Z_m$ are useful in determining whether each individual system configuration is declared as *failed* or *non-failed*, and the Markov model given by (3.20) allows the probabilities of declaring each configuration as *non-failed* or *failed* to be computed. However, it is necessary to define other sets of measures to quantify the system as a whole, i.e., aggregated measures for all the possible system configurations. System Reliability R and unreliability Q are examples of these aggregated measures. The definition of these aggregated measures is very important since they will be used to compare different system architectures, and to find weak points in a design.

In order to define these aggregated measures, we will make use of Markov reliability models (MRM) [52, 53], which have been used extensively to quantify the performability of computer systems [54] . Every aggregated measure, including System Reliability and Unreliability, will be derived from the general MRM formulation [30], which is defined by:

1. a Markov chain $X = \{X(t) : t \geq 0\}$, indexed in time by $[0, \infty[$, and taking values in some countable set $C = \{1, 2, \ldots N\}$; and
2. a reward function $r : C \to \mathbb{R}$, where the reward associated with each $i \in C$ is denoted by $r(i)$.

System reliability R can be obtained by using the indicator function defined in (3.14) (which takes value 1 when the configuration $\{i, k\}$ is declared as *non-failed*, and 0 otherwise) to define the reward model, thus $r_S(2i-1,k) = 1$ and $r_S(2i,k) = 0$. Then, system reliability is computed as

$$R = \mathbb{E}(r_S) = \sum_{i,k} r_S(i,k) p_{i,k}(T). \tag{3.45}$$

Similarly, system unreliability Q can be computed by defining the reward function $r_{\overline{S}}(2i - 1, k) = 0$ and $r_S(2i,k) = 1$, thus

$$Q = \mathbb{E}(r_{\overline{S}}) = \sum_{i,k} r_{\overline{S}}(i,k) p_{i,k}(T). \tag{3.46}$$

Aggregated measures of performance can also be computed for each performance metric Z_j, with $j = 1, 2, \ldots m$, by defining a reward function as

$$r_{Z_j}(i,k) = h_j(Z_j) \; \forall \, j = 1, 2, \ldots m \tag{3.47}$$

where $h_j(\cdot)$ is a real function. For example, if the Z_j performance metric considered is the electrical power consumption, it is possible to obtain the system average power consumption among all possible *non-failed* operational conditions by defining $r_{\overline{P}}(2i-1,k) = z_j(i,k)$ and $r_{\overline{P}}(2i,k) = 0$, thus the average power consumption $\overline{P}$, when the system is in a *non-failed* configuration, can be computed as

$$\overline{P} = \mathbb{E}(r_{\overline{P}}) = \sum_{i,k} r_{\overline{P}}(i,k) p_{i,k}(t). \tag{3.48}$$

3.7 Differences Between DPRA and the New Methodology

The methodology proposed in this chapter is similar to DPRA in the sense that it makes use of a dynamic (behavioral) system model, with additional information to model component behavior. It also makes use of a Markov chain to model the stochastic transitions between system configurations that take place when components fail. However, there are certain aspects to the kind of problems that DPRA tries to solve that make the mathematical formulation of DPRA much more complex than the formulation of our methodology.

In aerospace and automotive problems the decoupling assumption stated in Section 3.4.1 holds. The system dynamic time constants are on the order of seconds; therefore, when a component failure occurs, the system either recovers within a short transient period (on the order of seconds), reaching a new state within the "acceptable" region of the state-space dictated by the performance requirements, or it quickly diverts towards a state outside the "acceptable region". In this scenario, the likelihood of two components failing within a time on the order of magnitude of the system dynamic time constants is negligible relative to the likelihood of just one component failing. Therefore, the system stochastic behavior due to component failures can be be modeled as a process independent of the system dynamics. Thus, the resulting model is formulated in terms of just the probability of declaring a system configuration *non-failed* or *failed* after a sequence of component failures has occurred.

This is not the case in nuclear power plants, where it is not possible to decouple the stochastic transitions between configurations and system state dynamic variables [37]. In nuclear power plants, the time constants of the transitions from one configuration to another are large enough that the likelihood of another failure happening before the system reaches a new steady-state is relevant. Thus, in DPRA the stochastic model is formulated in terms of the probability of the system dynamic variables in a given configuration at a given time.

Component Model Definition in DPRA

In the context of DPRA, Amendola proposed a methodology called Logical Analytical Methodology (LAM) to quantitatively model a component with the following steps [33]:

1. An input-output scheme of the involved physical variables.

2. A set of equations describing the component nominal behavior.
3. A component failure modes and effect analysis (FMEA).
4. Parametric operators for FMEA synthesis
5. A component logical-analytical model, which analytically describes the component behavior in all its possible states.

There are similarities between this modeling method and the operational modes modeling technique introduced in this chapter. However, there are several aspects that makes the new modeling approach more general. First, the logical-analytical model described in step 5 is just a parametric representation of the nominal model described in step 2. The parameters of this logical-analytical model are changed according to the FMEA. In contrast, the new approach is more general because the non-nominal operational modes models are not necessarily related to the component failure-free behavior. Second, in the LAM approach, only transitions from the nominal behavior to the different failure modes are allowed (according to the linear structure of the FMEA). As mentioned before, the new approach allows us to define transitions between any behavioral modes. Finally, and most importantly, unlike the new modeling formalism, the LAM modeling formalism does not include the stochastic model that governs the transitions among behavioral modes.

System Stochastic Behavior Formulation in DPRA

The stochastic model associated with DPRA is formulated in terms of the probability of the system dynamic variables in a given configuration at a given time $p_{i,k}(x_s(t), t)$. Therefore the formulation of the Chapman-Kolmogorov equations becomes much more complicated [35]:

$$\begin{array}{c}\frac{\partial p_{i,k}(x_s(t),t)}{\partial t} + div(f_s^{i,k}(x_s(t), w(t))p_{i,k}(x_s(t), t)) = \\ -\lambda_{i,k}(x_s(t), t)p_{i,k}(x_s(t), t) + \sum\limits_{i,k-1} \lambda_{i,k-1}^{i,k}(x_s(t), t)p_{i,k-1}(x_s(t), t)\end{array} \tag{3.49}$$

where $\lambda_{i,k}(x(t), t)$ results from adding all the individual transition rates that can trigger a system transition out of configuration $\{i, k\}$ and it depends on the system state variables $x_s(t)$. $\lambda_{i,k-1}^{i,k}(x_s(t), t)$ is the transition rate associated with the behavioral mode transition that causes the system to go from configuration $\{i, k-1\}$ to system configuration $\{i, k\}$ and it depends on the system state variables $x_s(t)$ as well. The added complexity in (3.49) with respect to (3.19) resides in the additional divergency *div* term on the left side of (3.49) and the fact that the transition rates depend on the system state variables $x_s(t)$. Thus, unlike the proposed methodology, to obtain an analytical solution in DPRA becomes untractable for even simple problems. Thus several techniques based on discretization of time and state variables, and Monte Carlo simulation have been proposed to obtain a solution [36].

3.8 Conclusions

The methodology presented in this chapter uses a quantitative mathematical model of the behavior of the system to be analyzed. The model includes not only the system nominal behavior (no component failures), but also degraded behaviors due to failed components. This is an important feature of the methodology, since the system evaluation in the presence of failures no longer relies on the judgment of the analyst to assess whether or not a sequence of component failures will cause the system to fail or not. Furthermore, by using a quantitative system behavioral model, it is possible to evaluate degraded system operational modes, i.e., the analysis is not "system fails or not", the analysis now allows one to assess the "degree of failure" of a system.

Notation Used in this Chapter

A :	Markov model state-transition matrix
B_s :	Bias factor
$c_{j,k-1}^{i,k}$:	Failure coverage probability for a transition from configuration $\{j, k-1\}$ to $\{i, k\}$
c_i :	Component i
c, r :	Markov reward model parameters
C_1, C_2 :	Capacitor values in RLC circuit example
DPRA:	Dynamic Probabilistic Risk Assessment
ESCS:	Event Sequences and Consequences Spectrum
FMEA:	Failure Modes and Effects Analysis
$f_{c_i}^{j}(\cdot,\cdot)$:	State evolution function for the j behavioral mode of component c_i
$f_s^{i,k}(\cdot,\cdot)$:	State evolution function for system configuration $\{i, k\}$
$F^{i,k}(\cdot,\cdot)$:	State-transition matrix for system configuration $\{i, k\}$
$g_{c,i}^{j}(\cdot,\cdot)$:	Output function for the j behavioral mode of component c_i
$g_s^{i,k}(\cdot,\cdot)$:	Output function for system configuration $\{i, k\}$
$F^{i,k}(\cdot,\cdot)$:	Control matrix for system configuration $\{i, k\}$
$G_s, \hat{G}_s$:	Sensor behavioral model example gain change factor
$h_j(\cdot)$:	Reward model function for performance metric j
$i(t)$:	Current through RLC circuit example

LAM:	Logical Analytical Methodology
L_1 :	Inductance value in RLC circuit example
$p_{i,k}(t)$:	Probability that the system configuration is in configuration $\{i, k\}$ at time t given that at $t = 0$ is in $\{1, 0\}$
$P(t)$:	System configurations probability vector
Q :	System unreliability, Positive semi definite matrix associated to the system input uncertainty
r_{Z_j} :	Reward model associated with performance metric Z_j
r_S :	Reward function associated to the reliability measure computation
$r_{\overline{S}}$:	Reward function associated to the unreliability measure computation
$r_{\overline{P}}$:	Reward function associated to the power aggregated performance measure example
R_1 :	Resistor value in RLC circuit example
R_s :	Sensor behavioral model example resolution
$s_{i,k}(t^{i,k}$:	Indicator function for the status of configuration $\{i, k\}$
t :	System global evaluation time
$t^{i,k}$:	Configuration $\{i, k\}$ time-axis
$w(t)$:	System input
$u_{c_1}(t)$:	Sensor behavioral model example input
$u_{c_i}(t)$:	Component c_i input
$U_{c_i}(t)$:	Component c_i operational modes random variable
$v(t)$:	Voltage source in RLC circuit example
$x_{c_1}(t)$:	Sensor behavioral model example state variables
$x_{c_i}(t)$:	Component c_i state variables
$x_s(t)$:	System state variables
$X(t)$:	Random variable associated to the status of a Markov chain
$y_s(t)$:	System output
$y_{c_1}(t)$:	Sensor behavioral model example output
$y_{c_i}(t)$:	Component c_i output
Z_j :	Performance metric j
λ_{mn} :	m-to-n component behavioral mode transition rate
λ_{NO} :	Nominal-to-omission behavioral mode transition rate for sensor behavioral model example
λ_{NG} :	Nominal-to-gain-change behavioral mode transition rate for sensor behavioral model example
λ_{NB} :	Nominal-to-bias behavioral mode transition rate for sensor behavioral model example
λ_{GO} :	Gain-change-to-omission behavioral mode transition rate for sensor behavioral model example
λ_{BO} :	Bias-to-omission behavioral mode transition rate for sensor behavioral model example
λ_{lm} :	Transition from operational mode l to operational mode m
$\lambda^{i,k}_{j,k-1}$:	Transition rate from system configuration $\{j, k-1\}$ to configuration $\{i, k\}$
$\sigma_{\tau_l}(\cdot)$:	τ_l-shift operator
τ_l :	Sensor behavioral model example latency
τ_s :	Inverse of sensor behavioral model example bandwidth
$\Omega^{i,k}_x(t^{i,k})$:	Set of reachable states for configuration $\{i, k\}$
$\Omega_w(t^{i,k})$:	Set of possible system inputs
$\Phi^{i,k}_{Z_j}$:	Set that defines the performance requirements of performance metric Z_j for configuration $\{i, k\}$
$\Phi^{i,k}_{x(Z_j)}$:	State-space region dictated by the performance requirements set of performance metric Z_j for configuration $\{i, k\}$
$\Phi^{i,k}_x$:	State-space region dictated by the intersection of the performance requirements sets for configuration $\{i, k\}$
$\Theta^{i,k}_x$:	A subset of the set $\Omega^{i,k}_x(0^{i,k})$ such that, if the system state variables are contained inside, then the trajectories followed by the system are to remain, at all times, within $\Phi^{i,k}_x$
$\Psi^{i,k}$:	Positive semi definite matrix associated with the uncertainty of the initial conditions when the system transitions to configuration $\{i, k\}$
$\Gamma^{i,k}(t^{i,k})$:	Positive semi definite matrix associated with the system states evolution uncertainty when the system is in configuration $\{i, k\}$
$\Gamma^{i,k}_{ss}$:	Steady-state value of $\Gamma^{i,k}(t^{i,k})$

Chapter 4

InPRESTo -SIMULINK® Toolbox for Integrating Performance and Reliability

This chapter presents a MATLAB/SIMULINK® tool that supports the methodology introduced in Chapter 3 —InPRESTo, an acronym for Integrated Performance and Reliability Evaluation SIMULINK® Toolbox. An overview of the tool structure and capabilities will be presented. The definition of system behavioral models in the SIMULINK® environment will be explained in detail, as well as the system performance metrics definition and their associated requirements. The basic functionality of the tool will also be presented, as well as an explanation of the tool structure. Appendix B contains the tool subroutines flow diagram and the MATLAB® source code

4.1 Introduction

The methodology presented in Chapter 3 can be used to evaluate the reliability and performance of a system architecture and help identify weak points in the system. This allows for improvements in subsequent iterations of the system design. It can also be used in a slightly different way to compare different architecture alternatives to help identify the optimal design in terms of reliability and performance. However, these tasks can be daunting if performed manually, due to the large number of components that a fairly complex system may have. Therefore, in order to make the application of the methodology feasible, a MATLAB/SIMULINK® based tool —InPRESTo— was developed. This tool automates the evaluation process of a system defined in the SIMULINK® environment.

The ability of InPRESTo to analyze different complex systems has been shown in several case-studies. In Chapter 5, a lateral-directional flight control system case-study will be presented to illustrate how the toolbox can be used to identify weak points in a system design and how they can be improved thorough different design iterations. Chapter 6 presents a steer-by-wire case-study that shows how to use InPRESTo for comparing two conceptually very different architectural approaches for achieving fault tolerance. InPRESTo is now also being used by the Systems Engineering and Evaluation Division at the Charles Stark Draper Laboratory for the evaluation of space and tactical systems.

The purpose of the chapter is to explain the toolbox structure and main features, and also point to ways in which the tool could be improved. Section 4.2 presents the functionality of InPRESTo. Section 4.3 explains how to define the necessary inputs to perform a system analysis, i.e., the system dynamics behavioral model, the system performance metrics and their associated requirements, and the evaluation parameters. Section 4.4 explains how to run an analysis. Section 4.5 explains how to visualize the analysis results. Section 4.6 presents an overview of the tool structure. Finally, Section 4.7 highlights further development that could be done to improve InPRESTo's functionality.

4.2 Functionality

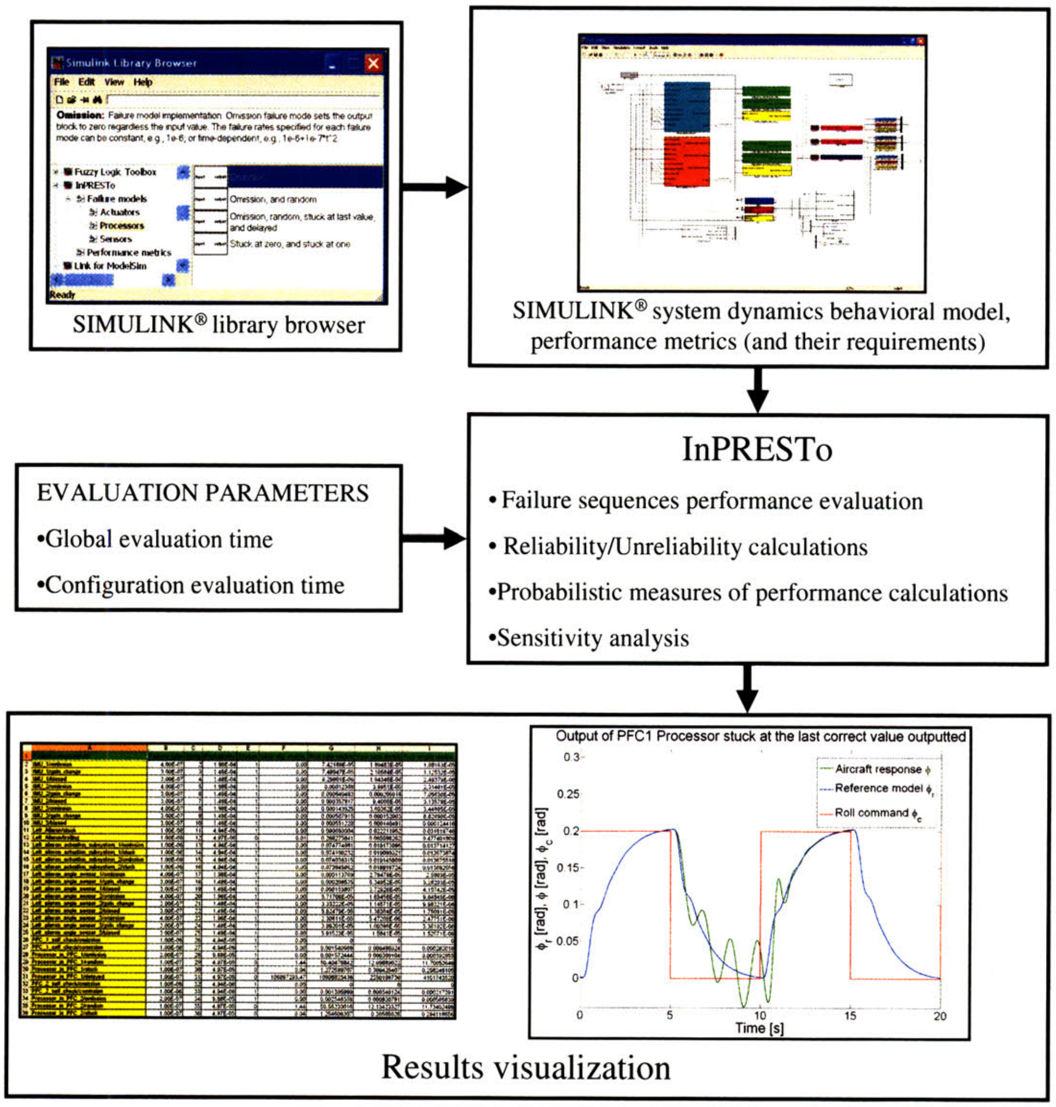
InPRESTo provides an environment for integrating system performance and reliability evaluation. The toolbox helps the analyst to:

- thoroughly evaluate system behavior in the presence of component failures;
- evaluate the effectiveness of failure detection, isolation, and reconfiguration mechanisms (FDIR);
- uncover weak design points, i.e., single points of failure and common modes of failure;
- quantify the main contributors to system unreliability;
- quantify the advantages of using a specific architecture among different alternatives.

The basic functionality of InPRESTo is displayed in Fig. 4.1. The inputs to InPRESTo are:

- The system dynamics behavioral model defined in the SIMULINK® environment. The component failure behavior can be built into each component model by *"drag and drop"* from a SIMULINK® library called *Failure Models*. This library can be accessed from the SIMULINK® GUI, and contains several failure models.

- System performance metrics and their requirements, which are defined within the SIMULINK® system behavioral model. There is a library called *Performance Metrics* with predefined performance metrics models from which the analyst can *"drag and drop"* as well.
- Evaluation parameters, which are additional parameters needed to run the analysis.


Simulink Library Browser
File Edit View Help
Fuzzy Logic Toolbox
InPRESTo
Failure models
Actuators
Processors
Sensors
Performance metrics
Link for ModelSim
Omission, and random
Omission, random, stuck at last value, and delayed
Stuck at zero, and stuck at one
Ready
SIMULINK® library browser
SIMULINK® system dynamics behavioral model, performance metrics (and their requirements)
EVALUATION PARAMETERS
•Global evaluation time
•Configuration evaluation time
InPRESTo
• Failure sequences performance evaluation
• Reliability/Unreliability calculations
•Probabilistic measures of performance calculations
•Sensitivity analysis
Output of PFC1 Processor stuck at the last correct value outputted
Aircraft response ϕ
Reference model ϕ_r
Roll command ϕ_c
ϕ_r [rad], ϕ [rad], ϕ_c [rad]
Time [s]
0
0.05
0.1
0.15
0.2
0.25
0.3
5
10
15
20
Results visualization

Omission: Failure model implementation. Omission failure mode sets the output block to zero regardless the input value. The failure rates specified for each failure mode can be constant, e.g., 1e-6, or time-dependent, e.g., 1e-6+1e-7*t^2

Figure 4.1: Basic functionality of InPRESTo.

InPRESTo will use the evaluation engine described in 4.6, which was coded as a collection of MATLAB® functions. The tool can perform exhaustive analyses of all possible sequences of component failures that can yield a system failure, or the analysis can be truncated when sequences of component failures of a certain size are reached. In the latter case, bounds on the reliability and unreliability are calculated to estimate the error introduced by truncating the analysis. InPRESTo can also calculate probabilistic measures of performance.

All the analysis results are automatically collected in several excel spreadsheets similar to the system level FMEAs shown in Section 2.3.1. The tool also allows the user to analyze and visualize the system behavior for chosen sequences of component failures. All the data analysis is also collected in several MATLAB® data files, and the information they contained can be retrieved any time after the evaluation is finished.

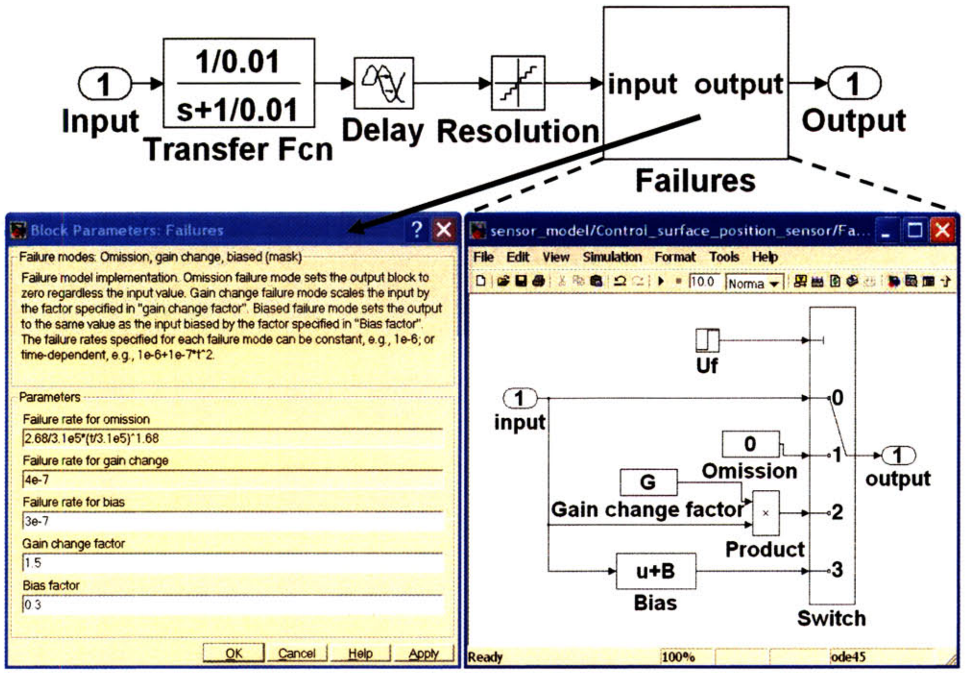
4.3 Defining the Inputs to InPRESTo

This section explains in detail how to define the three inputs necessary to carry out a system performance and reliability evaluation.

1. **System dynamics behavioral model**, which is constructed in the SIMULINK® environment;
2. **Performance metrics and their associated requirements** which are defined, as well, within the SIMULINK® system behavioral model; and
3. **Evaluation parameters**, which are defined through the MATLAB® Command Window.

4.3.1 System Dynamics Behavioral Model Definition

The system dynamics behavioral model is defined in SIMULINK®. As mentioned in Chapter 3, the system configurations are one of the main elements of the methodology. Theoretically it is necessary to have all the system configurations beforehand in order to carry out the analysis. In practice, the system failure-free configuration ($\{1, 0\}$ in equation (3.8)) is modeled in SIMULINK® by interconnecting the nominal models of each component,



Failure modes: Omission, gain change, biased (mask)

Failure model implementation. Omission failure mode sets the output block to zero regardless the input value. Gain change failure mode scales the input by the factor specified in "gain change factor". Biased failure mode sets the output to the same value as the input biased by the factor specified in "Bias factor". The failure rates specified for each failure mode can be constant, e.g., 1e-6; or time-dependent, e.g., 1e-6+1e-7*t^2.

Parameters

Failure rate for omission: 2.68/3.1e5*(t/3.1e5)^1.68

Failure rate for gain change: 4e-7

Failure rate for bias: 3e-7

Gain change factor: 1.5

Bias factor: 0.3

Figure 4.2: SIMULINK® Control Surface Position Sensor Model.

and afterwards augmenting each component nominal model with a corresponding failure behavior model. The realization of the remaining system configurations is carried out through simulation by injecting different sequences of component failure modes.

To illustrate how a system model is defined, we will show how to model one of the components within a model. The complete system model will emerge after connecting all the component models. Fig. 4.2 shows a SIMULINK® model for one of the control surface position sensors of the case-study presented in Chapter 5.

The mathematical model from which this example was developed is also shown in Section 3.2.1. On the upper part of Fig. 4.2, the elements labeled *Transfer Fcn*, *Delay*, and *Resolution* are common SIMULINK® blocks that allow the sensor functional properties to be modeled.

The block on the upper-right of Fig. 4.2, named *Failures*, is the component failure behavior model. The content of the *Failures* block is displayed on the right bottom of Fig. 4.2. The failure behavior of the sensor depends on the *Switch* control input U_f, which is controlled automatically by the MATLAB® evaluation engine (Section 4.6). For $U_f = 0$, the *Switch*

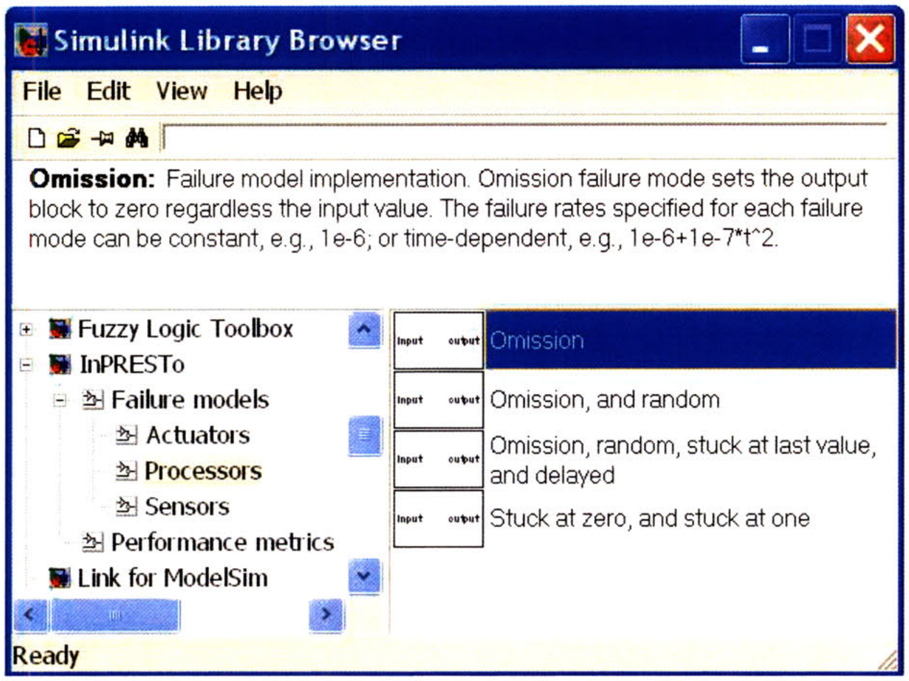
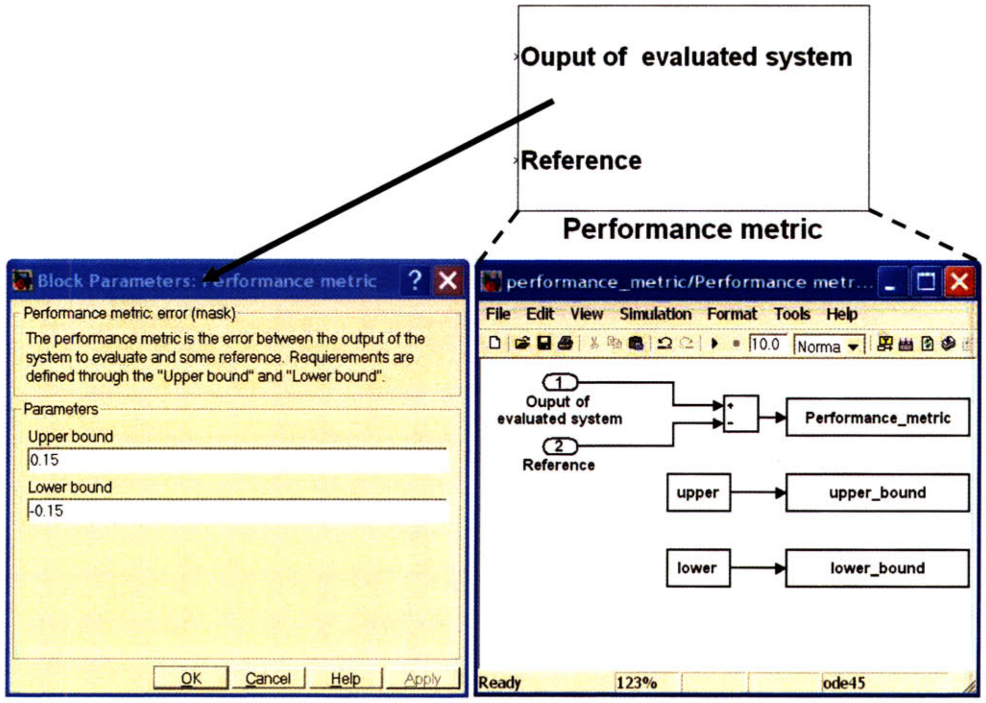

Omission: Failure model implementation. Omission failure mode sets the output block to zero regardless the input value. The failure rates specified for each failure mode can be constant, e.g., 1e-6; or time-dependent, e.g., 1e-6+1e-7*t^2.

Figure 4.3: SIMULINK® library browser displaying a library called InPRESTo that includes predefined failure models and performance metrics models.

block input 0 is passed to the output, meaning the sensor is failure-free. When the *Switch* control input U_f is set to $1, 2$, or 3, the inputs $1, 2, 3$ of the *Switch* block are passed to the output, modeling, respectively, Omission, Gain-Change and Bias failure modes. Several failure models have been predefined and can be inserted in the model by *"drag and drop"* from a SIMULINK® library called *Failure models*, displayed in Fig. 4.3. This library can be accessed from the SIMULINK® GUI. Failure models are available for actuators, processors, and sensors, and new categories could be created if considered necessary by the user. The failure models within the library have been created using SIMULINK® masked subsystems and can be modified by the user.

The bottom-left of Fig. 4.2 shows the GUI through which the user interfaces with the *Failures* block, allowing the definition of failure rates for each failure mode. In this case, we have used Omission, Gain-change, and Bias. It is possible to define constant failure rates and time-dependent failure rates if component wear-out mechanisms are to be modeled. The *Failures* block GUI allows other parameters of the component failure model to be defined. In this case, the Gain-change factor and the Bias factor. The component can only go from the non-failed status to any of the failed statuses defined by the failure modes, but


Ouput of evaluated system
Reference
Performance metric
Block Parameters: Performance metric
performance_metric/Performance metr...
File Edit View Simulation Format Tools Help
10.0
Norma
1
Ouput of evaluated system
2
Reference
+
-
Performance_metric
upper
upper_bound
lower
lower_bound
OK
Cancel
Help
Apply
Ready
123%
ode45

Performance metric: error (mask)

The performance metric is the error between the output of the system to evaluate and some reference. Requierements are defined through the "Upper bound" and "Lower bound".

Parameters

Upper bound
0.15

Lower bound
-0.15

Figure 4.4: SIMULINK® performance metric model definition.

once it is in a failed status, it remains there, it cannot fail in a different mode.

4.3.2 Performance Metrics Definition and Requirements Specification

The performance metrics, as well as their requirements, are defined within the SIMULINK® system behavioral model. Figure 4.4 illustrates how to define a performance metric. The content of the *Performance metric* block is displayed on the right bottom of Fig. 4.4, which defines the performance metric as the difference between the output of the system under evaluation and a reference, e.g., the output of the system under evaluation with no failures. The bottom-left of the figure shows the GUI through which the user interfaces with the *Performance metric* block, allowing the definition of requirements (in the form of upper and lower bounds).

There are several performance metrics models predefined in the SIMULINK® InPRESTo library, Fig. 4.3, that can be inserted in the model by *"drag and drop"*. The user can add new performance metrics models to this library, and modify the existing ones.

4.3.3 Evaluation Parameters

There are several parameters that need to be defined before a system evaluation can be carried out. These are:

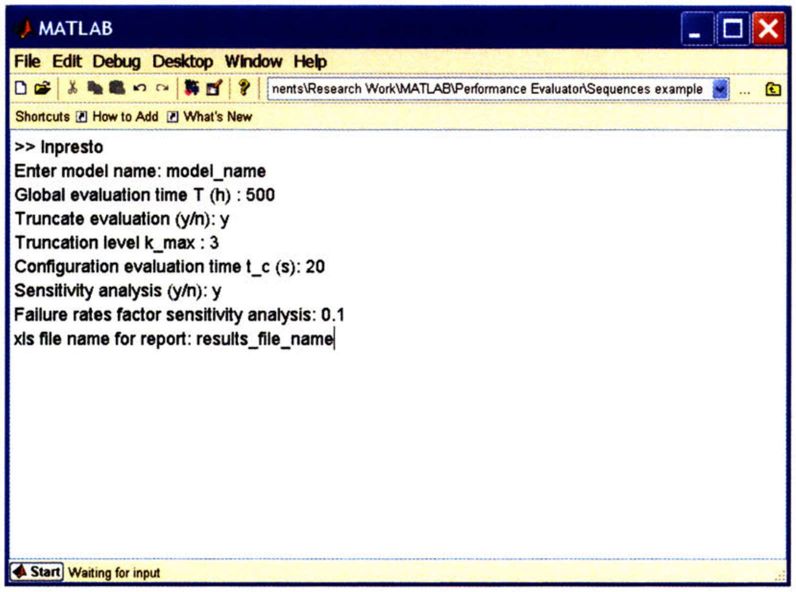
- **Global evaluation time** T(h). This is the time that will be used for evaluating the Markov reliability model. Depending on the system, it can be the lifetime, or the mission time, or the time between scheduled maintenance actions.
- **Truncation level** k_{max}. If system evaluation truncation is enabled, the truncation level k_{max} is the maximum size of failure sequences that will evaluated, i.e., the maximum number of failed elements within a sequence of failures.
- **Configuration evaluation time** t_c(s). The total time for simulating a system configuration reached after a sequence of component failures occurred. It is assumed that the analyst will pick t_c several times larger than the system's largest time constant.
- **Failure rates factor for sensitivity analysis.** If sensitivity analysis is required, this is the factor by which each component failure rate value will be multiplied and the result added to the nominal failure rate value.

4.4 Invoking InPRESTo

InPRESTo is invoked from the MATLAB® command window. Figure 4.5 shows a snapshot of the this window when invoking InPRESTo. As can be seen, the parameters mentioned in Section 4.3.3 are entered through the MATLAB® command window.

Before invoking InPRESTo, make sure that the SIMULINK® model of the system to be evaluated is opened; and that the MATLAB® path is set to the system SIMULINK® model directory. Then follow these steps:

1. Type `Inpresto` in the MATLAB® command window.


MATLAB
File Edit Debug Desktop Window Help
nents\Research Work\MATLAB\Performance Evaluator\Sequences example
Shortcuts How to Add What's New
Start Waiting for input

```
>> Inpresto
Enter model name: model_name
Global evaluation time T (h) : 500
Truncate evaluation (y/n): y
Truncation level k_max : 3
Configuration evaluation time t_c (s): 20
Sensitivity analysis (y/n): y
Failure rates factor sensitivity analysis: 0.1
xls file name for report: results_file_name
```

Figure 4.5: Snapshot of the MATLAB® command window when invoking InPRESTo.

2. Enter the name of the SIMULINK® model without including the `mdl` extension, e.g., if the model file name is `model_name.mdl`, then just type `model_name`.

3. Enter the global evaluation time T in hours.

4. If truncation of system evaluation is to performed, then type the command `y`, otherwise type the command `n`.

5. If truncation was chosen, then enter the truncation level k_{max}.

6. Enter the configuration evaluation time t_c in seconds.

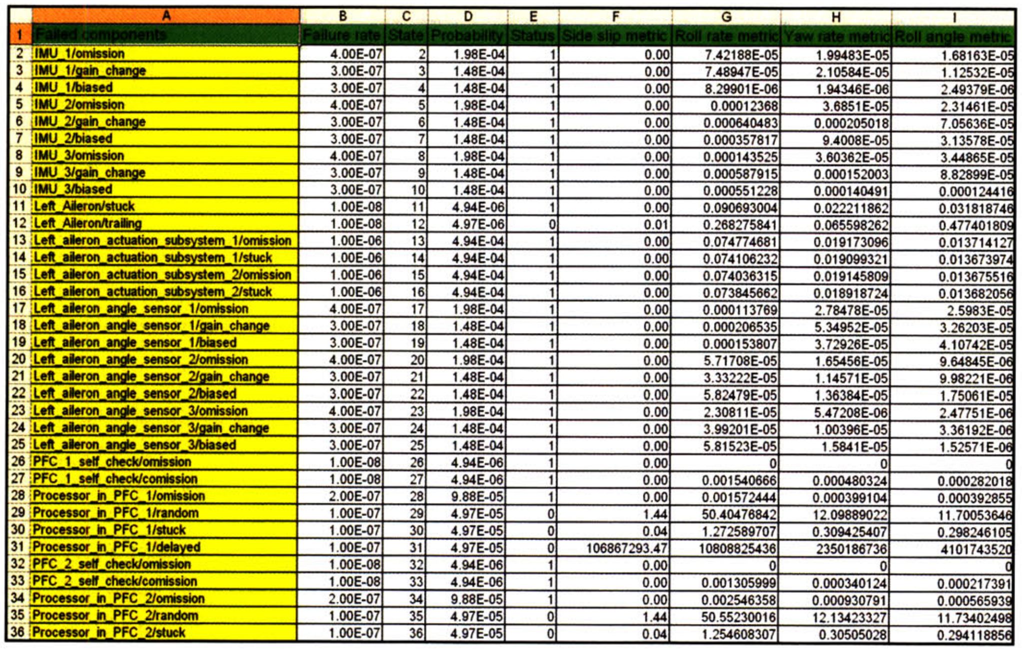
7. If sensitivity analysis is to be performed, then enter the command `y`, otherwise enter `n`.

8. If sensitivity analysis was required in step 7, then enter the failure rates factor for sensitivity analysis.

9. Enter the name of the EXCEL® `results_file_name.xls`, in which the evaluation results are to be collected.

4.5 Analysis Results Visualization

The analysis results are collected in an EXCEL® spreadsheet and all the relevant variables created during the evaluation are stored in several MATLAB® data files: `matrix_model.mat`, which stores the Markov reliability model state-transition matrix, and `model_results.mat`, which stores the rest of the variables created during the analysis. It is possible to visualize the dynamic behavior of the performance metrics for individual system configurations after all the analyses are completed.



	A	B	C	D	E	F	G	H	I
1	Failed components	Failure rate	State	Probability	Status	Side slip metric	Roll rate metric	Yaw rate metric	Roll angle metric
2	IMU_1/omission	4.00E-07	2	1.98E-04	1	0.00	7.42188E-05	1.99483E-05	1.68163E-05
3	IMU_1/gain_change	3.00E-07	3	1.48E-04	1	0.00	7.48947E-05	2.10584E-05	1.12532E-05
4	IMU_1/biased	3.00E-07	4	1.48E-04	1	0.00	8.29901E-06	1.94346E-06	2.49379E-06
5	IMU_2/omission	4.00E-07	5	1.98E-04	1	0.00	0.00012368	3.6851E-05	2.31461E-05
6	IMU_2/gain_change	3.00E-07	6	1.48E-04	1	0.00	0.000640483	0.000205018	7.05636E-05
7	IMU_2/biased	3.00E-07	7	1.48E-04	1	0.00	0.000357817	9.4008E-05	3.13578E-05
8	IMU_3/omission	4.00E-07	8	1.98E-04	1	0.00	0.000143525	3.60362E-05	3.44865E-05
9	IMU_3/gain_change	3.00E-07	9	1.48E-04	1	0.00	0.000587915	0.000152003	8.82899E-05
10	IMU_3/biased	3.00E-07	10	1.48E-04	1	0.00	0.000551228	0.000140491	0.000124416
11	Left_Aileron/stuck	1.00E-08	11	4.94E-06	1	0.00	0.090693004	0.022211862	0.031818746
12	Left_Aileron/trailing	1.00E-08	12	4.97E-06	0	0.01	0.268275841	0.065598262	0.477401809
13	Left_aileron_actuation_subsystem_1/omission	1.00E-06	13	4.94E-04	1	0.00	0.074774681	0.019173096	0.013714127
14	Left_aileron_actuation_subsystem_1/stuck	1.00E-06	14	4.94E-04	1	0.00	0.074106232	0.019099321	0.013673974
15	Left_aileron_actuation_subsystem_2/omission	1.00E-06	15	4.94E-04	1	0.00	0.074036315	0.019145809	0.013675516
16	Left_aileron_actuation_subsystem_2/stuck	1.00E-06	16	4.94E-04	1	0.00	0.073845662	0.018918724	0.013682056
17	Left_aileron_angle_sensor_1/omission	4.00E-07	17	1.98E-04	1	0.00	0.000113769	2.78478E-05	2.5983E-05
18	Left_aileron_angle_sensor_1/gain_change	3.00E-07	18	1.48E-04	1	0.00	0.000206535	5.34952E-05	3.26203E-05
19	Left_aileron_angle_sensor_1/biased	3.00E-07	19	1.48E-04	1	0.00	0.000153807	3.72926E-05	4.10742E-05
20	Left_aileron_angle_sensor_2/omission	4.00E-07	20	1.98E-04	1	0.00	5.71708E-05	1.65456E-05	9.64845E-06
21	Left_aileron_angle_sensor_2/gain_change	3.00E-07	21	1.48E-04	1	0.00	3.33222E-05	1.14571E-05	9.98221E-06
22	Left_aileron_angle_sensor_2/biased	3.00E-07	22	1.48E-04	1	0.00	5.82479E-05	1.36384E-05	1.75061E-05
23	Left_aileron_angle_sensor_3/omission	4.00E-07	23	1.98E-04	1	0.00	2.30811E-05	5.47208E-06	2.47751E-06
24	Left_aileron_angle_sensor_3/gain_change	3.00E-07	24	1.48E-04	1	0.00	3.99201E-05	1.00396E-05	3.36192E-06
25	Left_aileron_angle_sensor_3/biased	3.00E-07	25	1.48E-04	1	0.00	5.81523E-05	1.5841E-05	1.52571E-06
26	PFC_1_self_check/omission	1.00E-08	26	4.94E-06	1	0.00	0	0	0
27	PFC_1_self_check/comission	1.00E-08	27	4.94E-06	1	0.00	0.001540666	0.000480324	0.000282018
28	Processor_in_PFC_1/omission	2.00E-07	28	9.88E-05	1	0.00	0.001572444	0.000399104	0.000392855
29	Processor_in_PFC_1/random	1.00E-07	29	4.97E-05	0	1.44	50.40476842	12.09889022	11.70053646
30	Processor_in_PFC_1/stuck	1.00E-07	30	4.97E-05	0	0.04	1.272589707	0.309425407	0.298246105
31	Processor_in_PFC_1/delayed	1.00E-07	31	4.97E-05	0	106867293.47	10808825436	2350186736	4101743520
32	PFC_2_self_check/omission	1.00E-08	32	4.94E-06	1	0.00	0	0	0
33	PFC_2_self_check/comission	1.00E-08	33	4.94E-06	1	0.00	0.001305999	0.000340124	0.000217391
34	Processor_in_PFC_2/omission	2.00E-07	34	9.88E-05	1	0.00	0.002546358	0.000930791	0.000565939
35	Processor_in_PFC_2/random	1.00E-07	35	4.97E-05	0	1.44	50.55230016	12.13423327	11.73402498
36	Processor_in_PFC_2/stuck	1.00E-07	36	4.97E-05	0	0.04	1.254608307	0.30505028	0.294118856

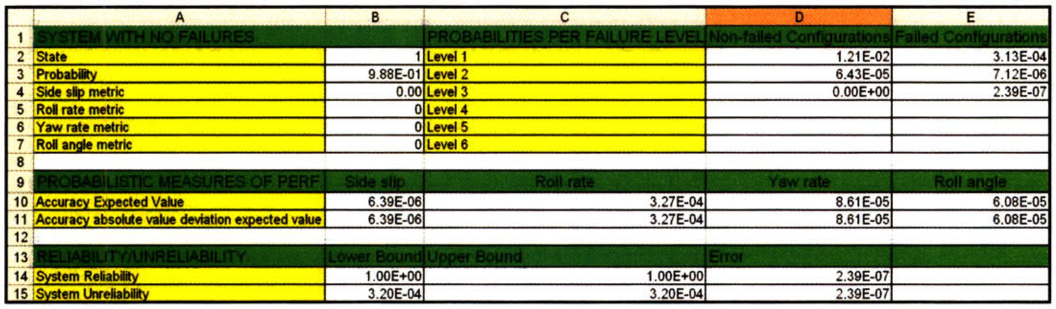
Figure 4.6: A snapshot of an EXCEL® spreadsheet results file showing the results for one of the sets of sequences of failures of the same size.

4.5.1 Reliability and Probabilistic Measures of Performance

The performance and reliability analysis results are collected in several sheets of the same EXCEL® spreadsheet. Each row of the EXCEL® spreadsheet shown in Fig. 4.6 is associated with a system configuration, and it includes the following information:

- **Column A: Failed components.** The sequence of component failures leading to a unique system configuration.

- **Column B: Failure rate.** The failure rate value associated with the last component to fail in the sequence displayed in Column A.
- **Column C: State.** An identifier that maps each system configuration into a state of the Markov reliability model.
- **Column D: Probability.** The probability of being in that system configuration at the global evaluation time T.
- **Column E: Status.** A binary variable that will take the value 1 if the configuration reached after the sequence of component failures displayed in Column A is declared non-failed, and 0 otherwise.
- **Columns F, G, H, I,...: Performance metrics.** The performance metrics' values in that system configuration.



	A	B	C	D	E
1	SYSTEM WITH NO FAILURES		PROBABILITIES PER FAILURE LEVEL	Non-failed Configurations	Failed Configurations
2	State	1	Level 1	1.21E-02	3.13E-04
3	Probability	9.88E-01	Level 2	6.43E-05	7.12E-06
4	Side slip metric	0.00	Level 3	0.00E+00	2.39E-07
5	Roll rate metric	0	Level 4		
6	Yaw rate metric	0	Level 5		
7	Roll angle metric	0	Level 6		
8					
9	PROBABILISTIC MEASURES OF PERF	Side slip	Roll rate	Yaw rate	Roll angle
10	Accuracy Expected Value	6.39E-06	3.27E-04	8.61E-05	6.08E-05
11	Accuracy absolute value deviation expected value	6.39E-06	3.27E-04	8.61E-05	6.08E-05
12					
13	RELIABILITY/UNRELIABILITY	Lower Bound	Upper Bound	Error	
14	System Reliability	1.00E+00	1.00E+00	2.39E-07	
15	System Unreliability	3.20E-04	3.20E-04	2.39E-07	

Figure 4.7: Snapshot of an EXCEL® spreadsheet results file showing system reliability and unreliability values, the failure-free configuration probability, and the probabilistic measures of performance associated with each performance metric.

Additionally, system reliability and unreliability values, the failure-free configuration probability; and the probabilistic measures of performance associated with each performance metric are collected in a separate sheet of the EXCEL® spreadsheet results file. Figure 4.7 shows a snapshot of this sheet, which includes the following information:

- **Column B; rows 2-7: System with no failures.** The Probability of being in the failure-free configuration at the global evaluation time T; and the performance metrics' values for the nominal configuration.
- **Column D, and E; rows 2-7: Probabilities per failure level.** For each failure level (system configurations with the same number of failed components), the probability of having declared a configuration as non-failed, or as failed.

- **Column B, C, D, E, ...; rows 10-11: Probabilistic measures of performance.** For each performance metric, different probabilistic measures of performance can be computed, i.e., expected value, or expected value of the absolute value deviation with respect to the nominal behavior.
- **Column B, C, D ...; rows 14-15: Reliability and unreliability.** System reliability and unreliability. If the evaluation is truncated, lower and upper bounds for those are displayed, as well as the error in their estimation.

4.5.2 Performance of Individual System Configurations

When the system evaluation is completed, it is possible to visualize the dynamic behavior of one or more system variables. This option allows the analyst to inspect the system transient behavior for the last component failure for any given sequence of component failures.

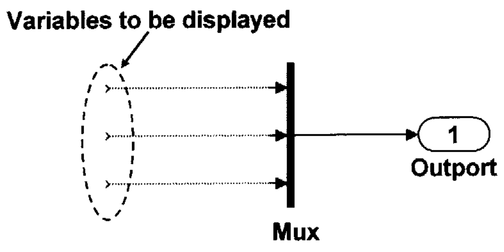


Figure 4.8: Definition of the system variables whose dynamic behavior is to be displayed.

Before invoking the plotting subroutine, it is necessary to define, in SIMULINK®, which system variables are to be plotted. The steps to define the variables to be displayed, Fig. 4.8, can be summarized as:

- From the SIMULINK® library browser, drag an `Outport` block into the top level of the SIMULINK® system model.
- From the SIMULINK® library browser, drag a `Mux` block into the top level of the SIMULINK® system model and connect it to the `Outport`.
- Modify the number of inputs that go into the `Mux` block to accommodate the number of system variables to be displayed.

- Connect each system variable to be displayed to an input of the `Mux` block.

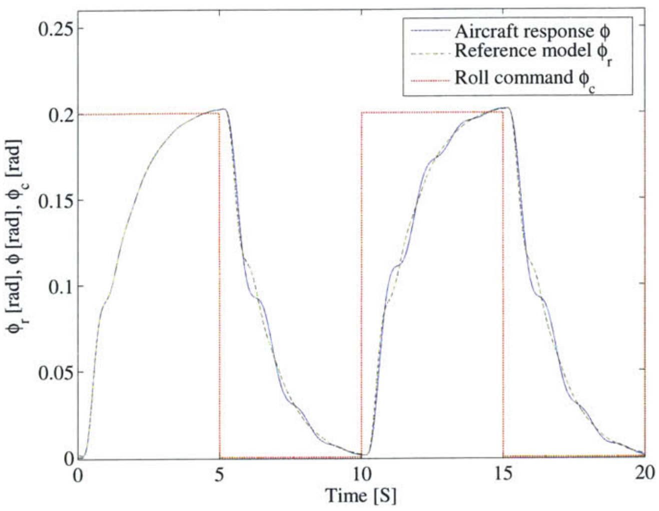


Figure 4.9: Visualization of the behavior corresponding to the lateral-directional flight control system case-study presented in Chapter 5.

After the variables to be displayed are defined as explained above, InPRESTo's plotting subroutine is invoked by typing the command `Inpresto_plot` in the MATLAB® command window. Then, the MATLAB® prompt will request the failure injection time t_f, which is the time when a failure is initiated after the simulation of each configuration is running, with $t_f < t_c$. The MATLAB® prompt will then request which system configuration dynamic behavior is to be displayed. By typing the identifier of each configuration as it appears in column C of the EXCEL® spreadsheet, Fig. 4.6, the required system configuration dynamic behavior will be plotted. Figure 4.9 shows a dynamic behavior plot corresponding to the lateral-directional flight control system for the case-study presented in Chapter 5.

4.6 Program Flow

InPRESTo will simulate each possible system configuration to check whether or not the performance requirements are met, until all possible sequences are exhausted or until the analysis is truncated. As the performance of the system is evaluated for each sequence of component failures, the state-transition matrix associated with the Markov reliability model is built. The functional flow diagram of InPRESTO is displayed in Fig. 4.10, and the functions performed in each box of the flow diagram are explained in the remainder of this section. Appendix B contains a more detailed flow diagram, as well as the subroutines' MATLAB® source codes.

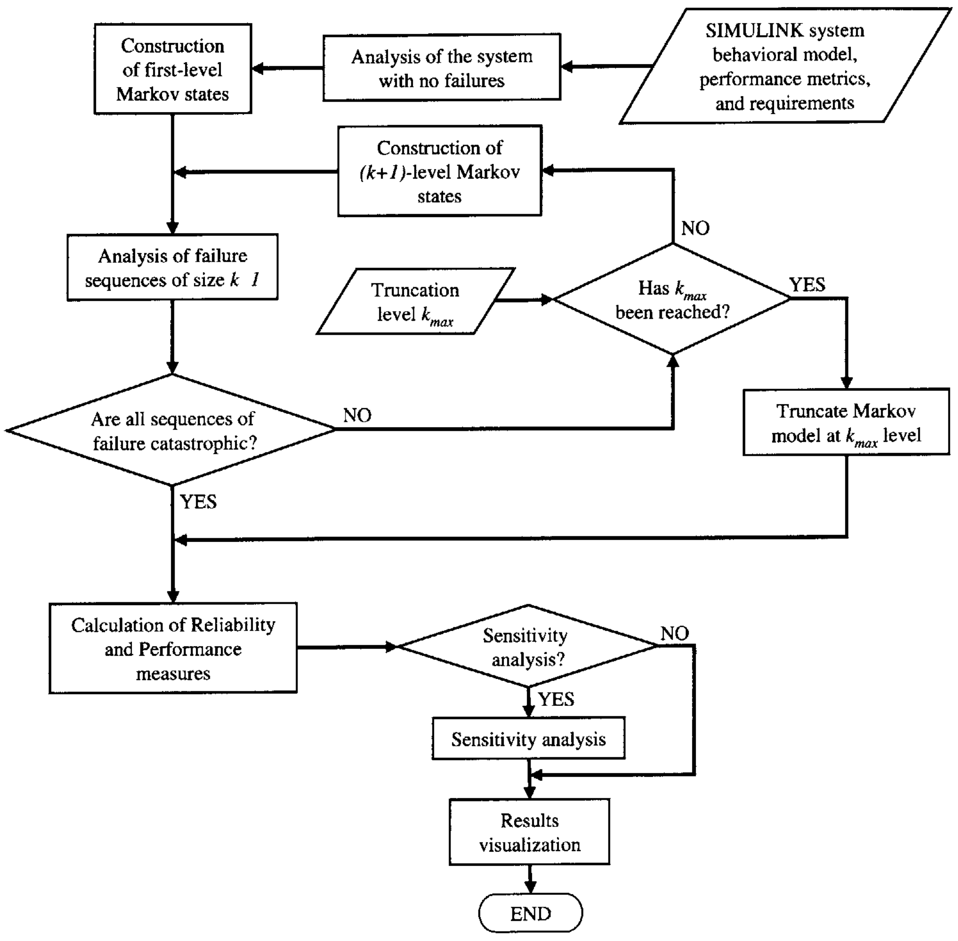

Construction of first-level Markov states
Analysis of the system with no failures
SIMULINK system behavioral model, performance metrics, and requirements
Construction of (k+1)-level Markov states
NO
Analysis of failure sequences of size k 1
Truncation level k_{max}
Has k_{max} been reached?
YES
Are all sequences of failure catastrophic?
NO
Truncate Markov model at k_{max} level
YES
Calculation of Reliability and Performance measures
Sensitivity analysis?
NO
YES
Sensitivity analysis
Results visualization
END

Figure 4.10: Functional flow diagram.

Analysis of the system with no failures

The SIMULINK® system model is simulated to obtain the system's nominal behavior, i.e., its behavior with no failures.

Construction of first-level Markov states

1. States for all possible system single failures N_1 are constructed, where N_1 is the sum of

failure modes of all the system components.

2. For each system single-failure constructed in step 1 a new entry is added to the Markov model state-transition matrix.

Analysis of failure sequences of size 1

1. Each possible system single-failure is injected in the SIMULINK® model, resulting in the realization of N_1 new system configurations.
2. The behavior of the N_1 single-failure configurations is simulated for a period of time t_c (*configuration evaluation time*) and for a given system control input $w(t)$. At the beginning of the simulation period, the system state variables $x_s(t)$ are set to their values at the end of the simulation period for the system with no failures.
3. The simulation of each configuration is carried out for 9 different equidistant *failure injection times* t_f^i, i.e., $t_f^i = \frac{t_c}{10}i$ for $i = 1, \dots, 9$. For each failure injection time t_f^i, the performance metrics are checked. If they do not meet requirements, the configuration is declared as failed (for the particular injection time). If the performance requirements are met, then the configuration is declared as non-failed (for the particular injection time).
4. The failure coverage probability is computed as the ratio of the number of runs that meet performance requirements in step (3) and the total number of runs 9.

After the analysis of single-failure configurations is carried out, the operations listed below take place until all the sequences of component failures result in failed system configurations, or the analysis is truncated.

Construction of $(k+1)$-level Markov states

1. The sequences of component failures of size $k \geq 1$ resulting in configurations with non-zero failure coverage probability are used to construct the remaining sequences with $(k+1)$ failed elements.
2. For each $(k+1)$-failure sequence a new entry is added to the Markov model state-transition matrix.

Analysis of failure sequences of size $k + 1$

1. Each possible $(k + 1)$ failure sequence is injected in the SIMULINK® model, resulting in the realization of N_{k+1} new system configurations.

2. The behavior of the N_{k+1} configurations (with $(k + 1)$ components failed) is simulated for a period of time t_c (*configuration evaluation time*) and for a given system control input $w(t)$. At the the beginning of the simulation period, the system state variables $x_s(t)$ are set to their values at the end of the simulation period for the configuration with the sequence of k failures that generated the new $k + 1$ failures sequence.

3. The simulation of each configuration is carried out for 9 different equidistant (*failure injection times*) t_f^i, i.e., $t_f^i = \frac{t_c}{10} i \; for \; i = 1, \ldots, 9$. For each failure injection time, the performance metrics are checked. If they do not meet requirements, the configuration is declared as failed (for the particular injection time). If the performance requirements are met, then the configuration is declared as non-failed (for the particular injection time).

4. The failure coverage probability is computed as the ratio of the number of runs of step (3) that meet the performance requirements in step (3) and the total number of runs 9.

Once all possible sequences of component failures are analyzed and the state-transition matrix is built, the following steps take place in order to compute reliability and performance measures.

Calculation of reliability and performance measures. The set of differential equations associated with the Markov reliability model is automatically solved, and reliability, unreliability and other probabilistic measures of performance are automatically obtained.

Sensitivity Analysis. In order to understand the influence of component failure rates, it is possible to carry out a sensitivity analysis on the system reliability and performance measures solution.

4.7 Further Development

Although InPRESTo has proved to be a very powerful tool for the performance and reliability evaluation of fault-tolerant systems, it needs further development. In order to make it more user-friendly, development of a *GUI* would facilitate the data input, post-evaluation results handling and visualization.

As shown in Section 4.3.1, the component behavioral models are built in such a way that the failure model only modifies the nominal output of the component. This is a powerful approach since the system nominal model (no component failure behavior models) can be developed first. It is possible then to add the failure models to each component tp obtain the complete system behavioral model. This approach to modeling component failure behavior is sufficient in most cases. However, there might be cases where this approach is not sufficient. Therefore, an improvement to the tool would be to implement the more general component modeling approach presented in Section 3.2.1. In this approach, the component behavior is modeled as a collection of dynamic models; each of them representing a different component failure behavior.

In terms of the stochastic model that governs the failure behavior of the components, the current implementation only allows the component to fail from its failure-free operational mode to each of the predefined failure modes. Once these are reached, no other transitions are allowed. To make the stochastic model more general, it would be necessary to modify the tool so transitions between different operational modes and even back to the failure-free operational mode would be possible, to allow modeling transient component failures. Regarding the component failure rates, time-dependent failure rates can be defined, which is a big advantage to modeling wear-out mechanisms. However, to improve further the component failure models, it would necessary to be able to define state-dependent failure rates.

The current implementation of InPRESTo only considers one predefined system input signal to carry out the analysis. For time-varying inputs, the uncertainty at which time the failure occurs is taken into account in the computation of the failure coverage probability, as explained in Section 4.6. However, considering only one predefined input signal is not enough to cover all the possible values that the control input can take. Therefore, it is necessary to implement a simulation strategy to choose random values from the set of possible system control inputs. As a result of this, it will be necessary to implement the Monte-Carlo approach to compute failure coverage probabilities as discussed in Section 3.4.1.

Notation Used in this Chapter

T :	Global evaluation time
k_{max} :	Truncation level
t_c :	Configuration evaluation time
t_f :	Failure injection time
$w(t)$:	System input
U_f :	Component failure model switch control input
$x(t)$:	System state variables

Chapter 5

Lateral-Directional Flight Control System Case-Study

In this chapter, we present a case-study of a fault-tolerant architecture for a fighter aircraft lateral-directional flight control system. The purpose of this case-study is to show how the performance and reliability evaluation SIMULINK® toolbox —InPRESTo— presented in Chapter 4 can be used to identify weak points in the system design, guide the design pointing out to possible solutions to eliminate the uncovered weak points, compare different architecture alternatives from different perspective, and test different failure detection, isolation, and reconfiguration (FDIR) techniques. Part of the the work presented in this chapter appears in [42].

5.1 Introduction

Most modern fighter aircrafts make use of fly-by-wire technology, i.e., there is no mechanical linkage between the stick or the pedals, and the control surfaces. Therefore, there is a need for a control system, including sensors, actuators and a computer, to transform the pilot inputs into commands for the control surfaces. There are two main reasons for the use of fly-by-wire technology: the lack of mechanical linkages reduces the weight of the aircraft; and the use of a computer to control the aircraft allows an aerodynamically unstable design that results in increased maneuverability. Thus, in order to fly and maintain the aircraft controllability, the control system must work at any time. Therefore, it is clear that a fault-tolerant control system is needed in order to prevent losing the aircraft, aborting a mission, or endangering the pilot's life if a failure occurs in the control system.

In this chapter, a case-study for the lateral-directional flight control system of a fighter aircraft is presented. This case-study shows how to use InPRESTo to guide the design process of aircraft avionics systems. In Section 5.2, the design goals that must be achieved are set forth. Section 5.3 defines the performance metrics and their associated requirements that will be used to conduct the system performance evaluation. Section 5.4 presents a first architectural alternative, called the dual channel architecture (DCa), in which only pure redundancy is used as a vehicle to achieve fault-tolerance. The analysis results of this first alternative will show that redundancy is not enough to achieve the design goals. This analysis also uncovers the weak design points, and gives guidance to improving the architecture. In Section 5.5 an improved design of the dual channel architecture, called enhanced dual channel architecture (EDCa), is presented and thoroughly analyzed. The analysis will show that this architecture does not meet all the design goals either, and will also show how this design can be further improved. Section 5.6 presents a further improved architecture, called dual-dual channel architecture (DDCa). The three architectural alternatives are compared in Section 5.7. Concluding remarks are presented in Section 5.8.

5.2 Design Goals

For the system under consideration, the design goals are threefold:

1. The system must tolerate any single component failure, i.e, it must be single fault-tolerant.
2. The system must be able to operate without any maintenance for 500 h.
3. The system dependability rate $\bar{\lambda}(T)$ (as defined in (A.5)) must not exceed 10^{-6} failures/hour.

From (2) and (3), a system unreliability Q at the end of the maintenance period of $5 \cdot 10^{-4}$ results.

5.3 System Performance Metrics Definition and Associated Requirements

The first step in carrying out the analysis of any system is to establish its performance metrics and associated requirements. In this case, the performance metrics chosen are the aircraft state variables: the sideslip $\beta(t)$, the body axis roll rate $p_b(t)$, the body axis yaw rate $r_b(t)$, and the body axis roll angle $\phi(t)$. Thus:

$$Z_1 = \beta(t), \tag{5.1}$$

$$Z_2 = p_b(t), \tag{5.2}$$

$$Z_3 = r_b(t), \tag{5.3}$$

$$Z_4 = \phi(t). \tag{5.4}$$

The dynamic behavior of a reference aircraft [41] (see Appendix C for the model details) will be used to define the performance metrics requirements. This reference aircraft state variables are denoted by the sub index $_r$, i.e., the sideslip is denoted by $\beta_r(t)$, the body axis roll rate by $p_{b_r}(t)$, the body axis yaw rate by $r_{b_r}(t)$, and the body axis roll angle by $\phi_r(t)$. Thus, the performance metrics requirements are defined as

$$\Omega_{Z_1} = \{\beta(t) \in \mathbb{R} \;/\; \| \beta(t) - \beta_r(t)\|_\infty \leq r_\beta\}, \tag{5.5}$$

$$\Omega_{Z_2} = \{p_b(t) \in \mathbb{R} \;/\; \| p_b(t) - p_{b_r}(t)\|_\infty \leq r_{p_b}\}, \tag{5.6}$$

$$\Omega_{Z_3} = \{r_b(t) \in \mathbb{R} \;/\; \| r_b(t) - r_{b_r}(t)\|_\infty \leq r_{r_b}\}, \tag{5.7}$$

$$\Omega_{Z_4} = \{\phi(t) \in \mathbb{R} \;/\; \| \phi(t) - \phi_r(t)\|_\infty \leq r_\phi\}, \tag{5.8}$$

where $r_\beta = 0.15$rad, $r_{p_b} = 0.45$rad/s, $r_{r_b} = 0.45$rad/s, and $r_\phi = 0.15$rad.

As mentioned in Chapter 4, when using InPRESTo for the analysis of any system, it is necessary to define the system control inputs that will be used to evaluate each configuration. In this case, a 0.2rad, 0.1Hz square wave in the roll command ϕ_c was chosen, which is displayed in Fig. 5.1 together with the reference aircraft dynamic response.

5.4 Dual Channel Architecture: Pure Redundancy

The proposed dual channel architecture (DCa), Fig. 5.2, is based on the use of pure redundancy. No FDIR mechanisms are implemented, except for voting algorithms for the triple redundant measurements. The architecture is composed of two redundant primary flight computers (PFC_1 and PFC_2) that receive information about aircraft attitude from three redundant inertial measurement units (IMU_1, IMU_2 and IMU_3) cross strapped to both computers; and also information about the control surface position from triple redundant position sensors for the rudder (RPS_1, RPS_2 and RPS_3), left aileron (LAPS_1, LAPS_2 and LAPS_3),

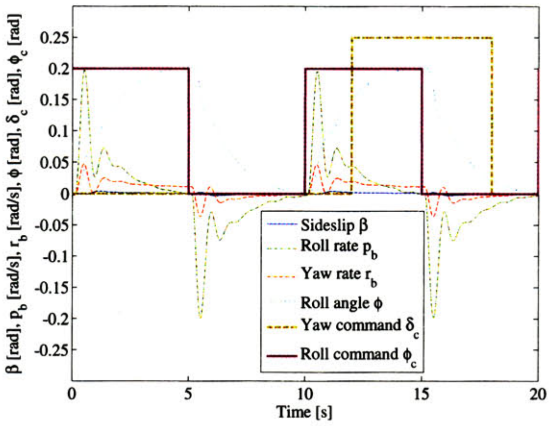


Figure 5.1: Reference aircraft response to a 0.2rad, 0.1Hz square wave in the roll command ϕ_c. Sideslip response β, roll rate response p_b, yaw rate response r_b, and roll angle response, for the system in its nominal configuration.

and right aileron (RAPS$_1$, RAPS$_2$ and RAPS$_3$) also cross strapped to both computers. Both PFCs have a voting algorithm implemented to compute the actual aircraft attitude from the triple redundant IMUs measurements, and voting algorithms for each set of triple LAPSs, RAPSs, and RPSs measurements. Both PFCs have also implemented control laws for computing the appropriate commands for the control surface actuation subsystems based on the IMUs measurements and the pilot inputs through the stick and the pedals. Each control surface is actuated by two redundant actuation subsystems, LAAS$_1$, and LAAS$_2$ for the left aileron; RAAS$_1$, and RAAS$_2$ for the right aileron; and RAS$_1$, and RAS$_2$ for the rudder. The outputs of each pair of actuation subsystems are mechanically combined to produce the appropriate command for the corresponding control surface. Each element of the control-surface-actuation-subsystem pair is commanded independently from each PFC. Therefore, there are three actuation subsystems per PFC, commanding both left and right ailerons, and the rudder. Each actuation subsystem is composed of a current-controlled electric motor. When any of the above mentioned hardware components fails, it remains in the control loop, and the additional redundant units are supposed to compensate for the failure.

To complete the system behavioral model, a linear lateral-directional aircraft dynamics model [55], [56] interacting with the avionics architecture model described above is included. The state variables of this model are the sideslip β, the body axis roll rate p_b, the body axis yaw rate r_b, and the body axis roll angle ϕ. The control surface commands are both left and right aileron angles δ_a^l and δ_a^r, and the rudder angle δ_r. The complete state-space representation of the aircraft lateral-directional dynamics is shown in Appendix C.

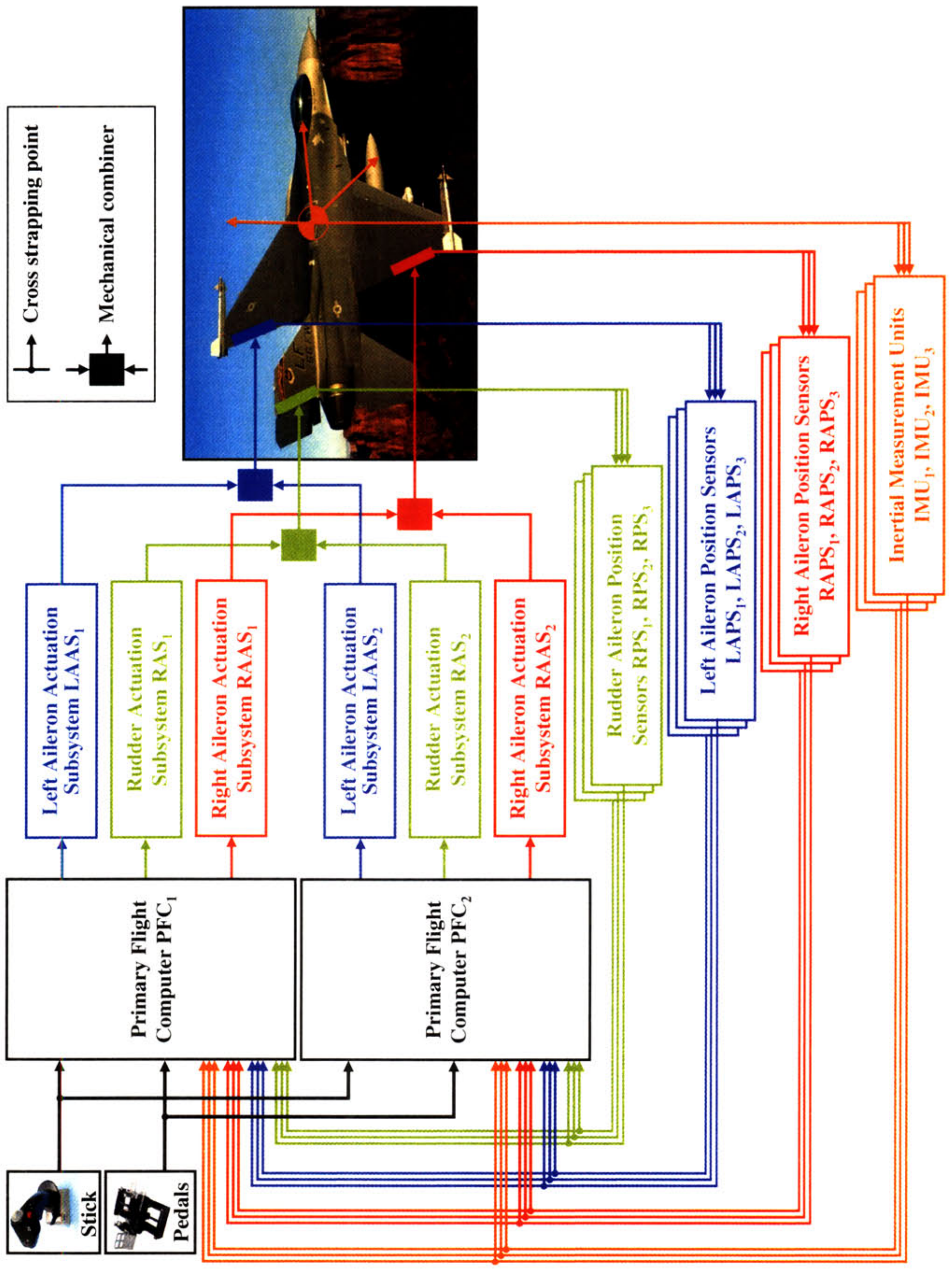

Cross strapping point
Mechanical combiner
Stick
Pedals
Primary Flight Computer PFC$_1$
Primary Flight Computer PFC$_2$
Left Aileron Actuation Subsystem LAAS$_1$
Rudder Actuation Subsystem RAS$_1$
Right Aileron Actuation Subsystem RAAS$_1$
Left Aileron Actuation Subsystem LAAS$_2$
Rudder Actuation Subsystem RAS$_2$
Right Aileron Actuation Subsystem RAAS$_2$
Rudder Aileron Position Sensors RPS$_1$, RPS$_2$, RPS$_3$
Left Aileron Position Sensors LAPS$_1$, LAPS$_2$, LAPS$_3$
Right Aileron Position Sensors RAPS$_1$, RAPS$_2$, RAPS$_3$
Inertial Measurement Units IMU$_1$, IMU$_2$, IMU$_3$

Figure 5.2: Lateral-directional flight control system fault-tolerant architecture.

The component behavioral models for each hardware and software component, i.e., primary flight computers (PFC), voting algorithms, control laws, inertial measurements units (IMU), rudder position sensors (RPS), left and right aileron position sensors (LAPS and RAPS), rudder actuation subsystems (RAS), left and right aileron actuation subsystems (LAAS and RAAS) rudder (R), left aileron (LA), and right aileron (RA) are described in Appendix C.

Table 5.1: Component failure model parameters.

Component	*Failure modes*	*Description*	U_f	$\lambda(/h)$
PFC_1, PFC_2	Omission	Output set to zero	1	$2 \cdot 10^{-7}$
	Random	Random output between -5 and 5	2	10^{-7}
	Stuck	Output stuck at last correct value	3	10^{-7}
	Delayed	Output delayed 0.2 s	4	10^{-7}
$LAAS_1$, $LAAS_2$, $RAAS_1$	Omission	Output set to zero	1	10^{-6}
$RAAS_2$, RAS_1, RAS_2	Stuck	Output stuck at last correct value	2	10^{-6}
R, LA, RA,	Omission	Output set to zero	1	10^{-8}
	Trailing	Output commanded by the aircraft dynamics	2	10^{-8}
IMU_1, IMU_2, IMU_3	Omission	Output set to zero	1	$4 \cdot 10^{-7}$
	Gain change	Output scaled by a factor of 1.5	2	$3 \cdot 10^{-7}$
	Biased	Output biased by a factor of 0.3	3	$3 \cdot 10^{-7}$
$LAPS_1$, $LAPS_2$, $LAPS_3$,	Omission	Output set to zero	1	$4 \cdot 10^{-7}$
$RAPS_1$, $RAPS_2$, $RAPS_3$,	Gain change	Output scaled by a factor of 1.5	2	$3 \cdot 10^{-7}$
RPS_1, RPS_2, RPS_3	Biased	Output biased by a factor of 0.3	3	$3 \cdot 10^{-7}$

Table 5.1 collects information corresponding to the failure models of the different hardware components. The possible failure modes of each component are listed in column 2, while column 3 is an explanation of the effect of each failure mode on the component behavior. U_f in column 4 is the variable that assigns the corresponding failure mode to the component behavioral model equations (see Appendix C). The last column of the table collects the failure rates λ associated with each failure mode, which are necessary to build the state-transition matrix associated with the Markov reliability model.

5.4.1 Performance and Reliability Evaluation

The system evaluation was carried under specific conditions. The aircraft is considered to be in a cruising phase with forward velocity $V = 178$ m/s, pitch angle $\alpha_0 = 0.216$ rad and a cruising altitude of 10,668 m. Under these conditions, the aircraft time constants dictate a configuration evaluation time $t_c = 20$ s.

Table 5.2: Dual channel architecture: single points of failure and unreliability for different levels of truncation and an evaluation time of 500 h.

Truncation level	*Unreliability lower bound*	*Unreliability upper bound*	Single points of failure	*# of system configurations*
2	$5.12 \cdot 10^{-4}$	$5.82 \cdot 10^{-4}$	11	64
3	$5.20 \cdot 10^{-4}$	$5.20 \cdot 10^{-4}$	11	3088

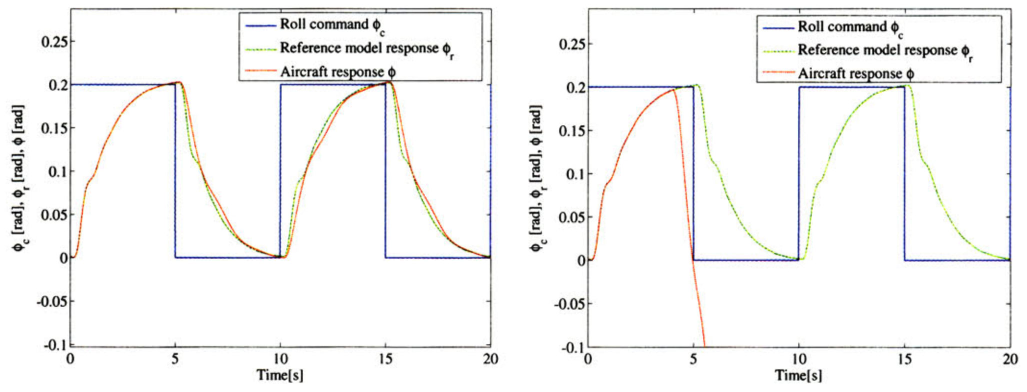
Table 5.2 shows he number of single points of failure and the probability of system failure (unreliability) at the end of the maintenance period for different levels of truncation. This design does not meet the single fault-tolerance requirement as there are 11 single points of failure. Truncating after three component failure events yields a system unreliability of $5.20 \cdot 10^{-4}$, which is slightly larger than the design requirement ($Q < 5 \cdot 10^{-4}$).

There is a trade-off between achieving a higher accuracy in estimating reliability and computational time for performing the evaluation. By truncating the evaluation at the second level, only 64 system configurations are evaluated, and the evaluation takes less than 4 minutes. If the truncation is done at the third level, 3088 possible configurations are analyzed and the evaluation takes 2 hours and 46 minutes. The computation was carried out on a machine with a 2.1 GHz Pentium® M processor, and 1.5Gb of RAM.

For clarity, in the remaining results analysis, only the response of one performance metric — the aircraft roll angle ϕ — will be analyzed. For several single failures, the aircraft roll angle response ϕ will be plotted together with the reference aircraft model response ϕ_r, for a 0.2rad, 0.1Hz square wave roll command ϕ_c.

Single aileron failures

Figure 5.3(a) shows the roll angle aircraft response ϕ, and reference model response ϕ_r, for a single failure in the left aileron, in which it fails by getting stuck at the position it was in when the failure occurred. Although the system performance is degraded, the performance metrics (i.e., the aircraft state variables) remain within their requirements. Fig. 5.3(b) shows the aircraft response for another failure of the left aileron. The failure mode is such that the aileron is now commanded by the aircraft dynamics, i.e., the aileron trails. In this case, the failure is catastrophic. It can be seen that 4s after the failure occurs, ϕ rapidly increases. This means that the aircraft is rolling without any control.



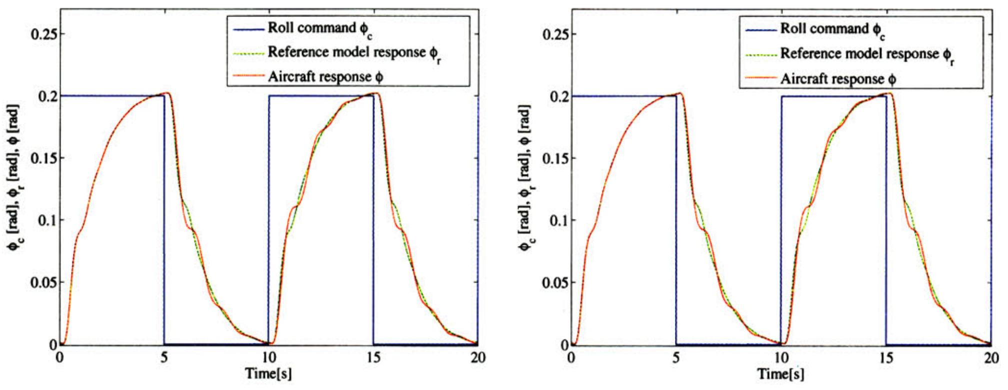
(a) Stuck failure mode. (b) Trailing failure mode.

Figure 5.3: Dual channel architecture performance in response to a 0.2 rad, 0.1 Hz square wave in roll command ϕ_c. Aircraft roll angle response ϕ compared to reference aircraft model response ϕ_r, for different single failure modes in the left (or right) aileron, and a failure injection time $t_f = 4$ s.

In these cases, there are not many things that can be done to improve the system performance (in the stuck-failure-mode), or to keep the aircraft stable (in the trailing-failure-mode), since the aileron is non-redundant, and when it fails, it affects the aircraft aerodynamics. However, as reported in [57], NASA developed a system, called propulsion controlled aircraft (PCA), to compensate for failures in the control surface by reconfiguring the engines thrust control system, enabling the use of differential thrust to maneuver the aircraft. This could be an example of how to achieve fault-tolerance in a system in which is not possible to use redundancy.

Single aileron actuation subsystem failures

For any of the left aileron actuation subsystems, Fig. 5.4 displays the aircraft behavior for a failure-by-omission, Fig. 5.4(b), and a failure-by-stuck, Fig. 5.4(a), i.e, the actuation subsystem acts as a load of constant torque for the remaining healthy actuation subsystem. In this condition, the roll angle response ϕ does not perfectly match the reference model response ϕ_r, thus a degraded performance behavior results. Nevertheless, the system is stable and the performance metrics lie within the requirements. Therefore this configuration is declared as non-failed. In this case, having pure redundancy is enough to compensate for any failure in the left (or right) aileron actuation subsystems.



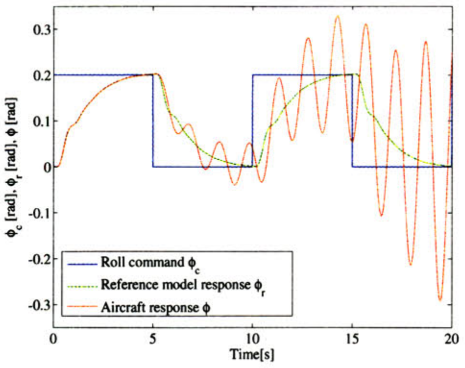
(a) Output omission failure mode. (b) Stuck failure mode.

Figure 5.4: Dual channel architecture performance in response to a 0.2 rad, 0.1 Hz square wave in roll command ϕ_c. Aircraft roll angle response ϕ compared to reference aircraft model response ϕ_r, for different single failure modes in one of the left (or right) aileron actuation subsystems, and a failure injection time $t_f = 4$ s.

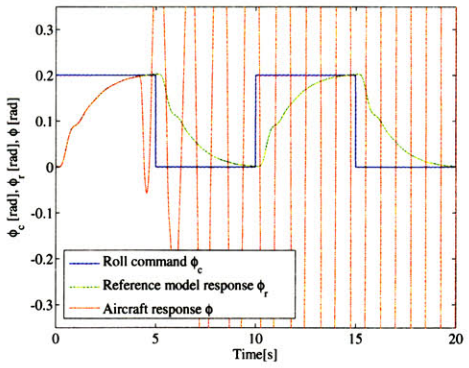
However, there are ways to improve the performance of the aircraft when failures in the left (or right) aileron actuation subsystems occur. An example is to introduce some sort of FDIR mechanism to detect and isolate (some, or all) failures within the actuation subsystem, and if necessary, reconfigure the control laws in the remaining actuation subsystem to account for those failures.

Single primary flight computer failures

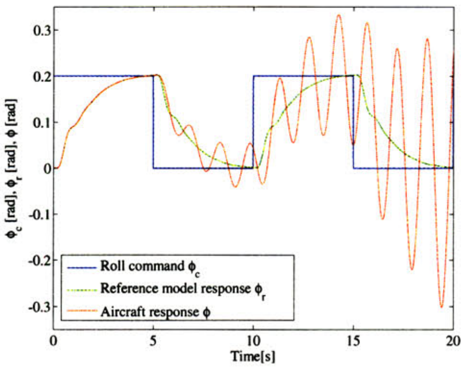
Figure 5.5 shows the aircraft behavior for different failure modes in any of the primary flight computers. It can be seen that despite the presence of another primary flight computer in command of half of the system, any failure will cause the aircraft to become unstable. For a failure-by-output-omission, Fig. 5.5(a), one of the channels in the forward loop (Primary flight computer, left and right aileron actuation subsystems, and rudder actuation subsystems) is effectively removed, i.e., the computer stops sending any command to the actuation subsystems, therefore these stop commanding the control surfaces. This result in an alteration of the closed-loop dynamics that makes the system become unstable. In the case of a linear system, the effect of removing one channel of the control-loop would result in the relocation of the system poles, some of them moving to the right-half-plane, which would make the system unstable. A similar explanation can be given for the failure-by-stuck-output, 5.5(b). In this case, the computer output is set to a constant. Therefore the



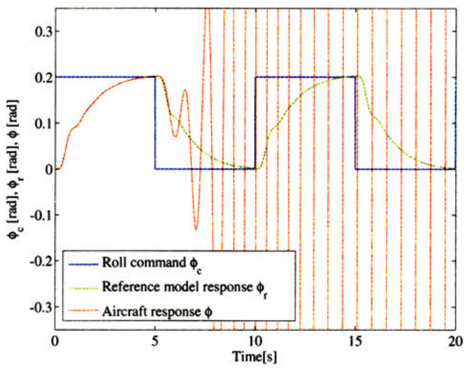
(a) Output omission failure mode.



(b) Random output between -5 and +5 failure mode



(c) Output stuck failure mode.



(d) Output delayed failure mode.

Figure 5.5: Dual channel architecture performance in response to a 0.2 rad, 0.1 Hz square wave in roll command ϕ_c. Aircraft roll angle response ϕ compared to reference aircraft model response ϕ_r, for different single failure modes in one of the primary flight computers, and a failure injection time $t_f = 4$ s.

control surface actuation subsystems commanded by the faulty computer set their outputs to a constant value, acting as a load for the actuation subsystems in the other channel. This causes the same effect as before: an alteration of the closed-loop dynamics that makes the system become unstable. The effect of the other two failure modes is even more dramatic as can be seen in Fig. 5.5(c) for the failure-by-delayed-output, and in Fig. 5.5(d) for the failure-by-random-output.

The most important conclusion extracted from the effect of primary flight computer failures on the aircraft response is that using redundancy alone is not sufficient to achieve fault-tolerance; unlike the case of failures in the control surface actuation subsystems, where redundancy alone was sufficient. The results analyses also point to possible solutions to overcome the problems shown. First, failure detection and isolation is not enough; reconfiguration of the control laws in the second computer is also necessary. The reason for this can be understood if the failure-by-output-omission is analyzed. The effect of this failure on the system behavior is equivalent to the effect of detecting any failure in the primary flight computer and isolating the failure by shutting the computer down. Looking at Fig. 5.5(a) is enough to understand that this strategy will not work. Thus, it is necessary to do something else: to reconfigure the control laws in the remaining computer after the failed computer is shut down.

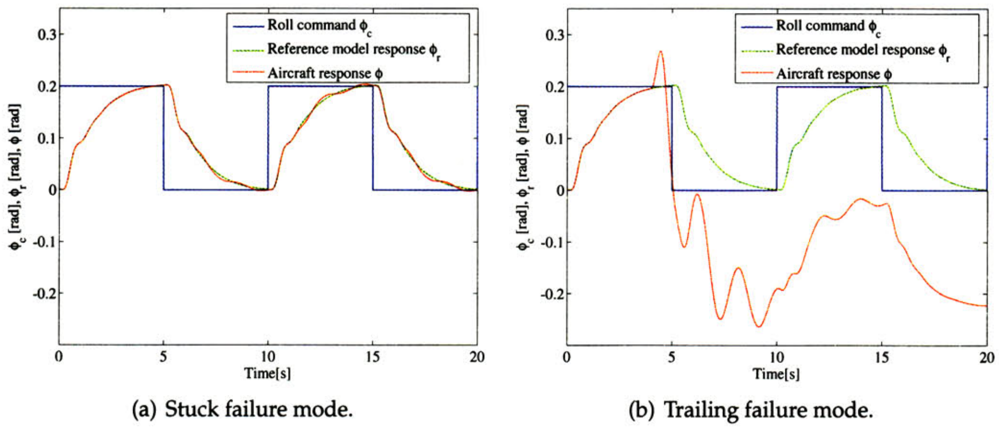
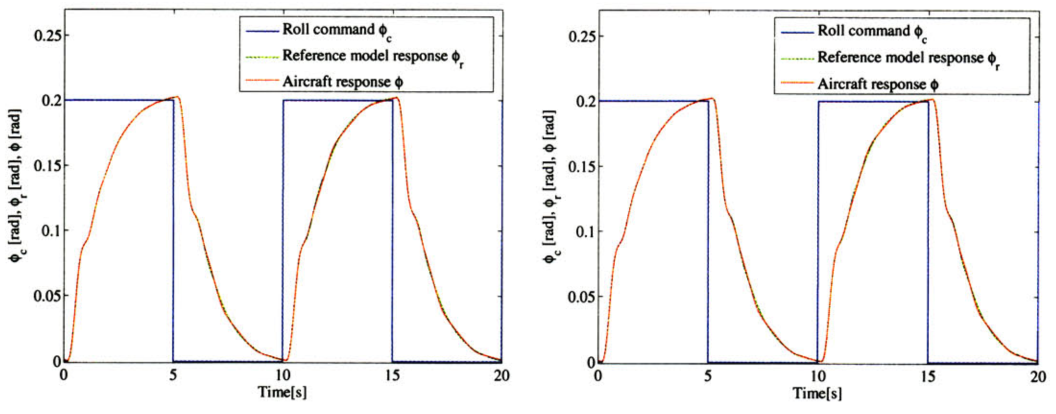


Figure 5.6: Dual channel architecture performance in response to a 0.2 rad, 0.1 Hz square wave in roll command ϕ_c. Aircraft roll angle response ϕ compared to reference aircraft model response ϕ_r, for different single failure modes in the rudder, and a failure injection time $t_f = 4$ s.

Single rudder failures

Figure 5.6 shows the aircraft response for failures in the rudder. Similar to the aircraft behavior displayed for failures in the left (or right) ailerons, Fig. 5.6(a) corresponds to a failure-by-stuck of the rudder, which degrades the aircraft performance, but the performance metrics are still within requirements. Figure 5.6(b) corresponds to a trailing failure of the rudder, which results in a system failure.

As mentioned for failures in left and right ailerons, this is not a problem that can be solved using redundancy, since there is only one rudder in the aircraft. Perhaps this can be solved using the propulsion controlled aircraft approach already mentioned, and reported in [57].



(a) Output omission failure mode. (b) Stuck failure mode.

Figure 5.7: Dual channel architecture performance in response to a 0.2 rad, 0.1 Hz square wave in roll command ϕ_c. Aircraft roll angle response ϕ compared to reference aircraft model response ϕ_r, for different single failure modes in one of the rudder actuation subsystems, and a failure injection time $t_f = 4$ s.

Single rudder actuation subsystem failures

Figure 5.7 shows the effect of failures in any of the rudder actuation subsystems. As shown in Fig. 5.4, any failures in the actuation subsystems for left and right ailerons affected the system performance, but did not cause the aircraft to become unstable. This is the same effect that failures in one of the rudder actuation subsystems cause in the system: degraded performance, but not instability.

As mentioned for the case of failures in the left or right ailerons, the aircraft performance could be improved by introducing failure detection, isolation, and reconfiguration (FDIR) mechanisms to — partially or completely — detect and isolate failures within the rudder actuation subsystem; and if necessary, reconfigure the control laws in the remaining rudder actuation subsystem to account for those failures.

5.5 Enhanced Dual Channel Architecture: Introducing Failure Self-Detection in the PFCs

In the dual channel architecture presented in Section 5.4, every component, except the control surfaces, is duplicated or triplicated in order to achieve fault-tolerance. The detailed analysis of this architecture shows that redundancy was not enough to achieve fault-tolerance in some cases. In this section, and based on the results analysis of the dual channel architecture, an enhanced dual channel architecture is proposed.

The main conclusions extracted from the analysis of single failures in the dual channel architecture were:

1. Some failures in the control surface cause the system to become unstable which cannot be overcome using conventional fault-tolerant techniques.
2. Failures in the control surface actuation subsystem cause degraded performance but do not cause the aircraft to become unstable.
3. Despite the presence of two primary flight computers, any single failure in a computer will cause the aircraft to become unstable.

As mentioned before, it may be possible to solve each of the problems listed above. However, it is not the purpose of this chapter to design sophisticated FDIR mechanisms, but only to illustrate how the methodology can be used to uncover weak design points, how these can be improved, and how the effectiveness of different FDIR strategies can be tested. Therefore, only the issues related to the third item listed above, i.e., how to compensate failures in any of the PFCs, will be treated.

Focusing on the PFC failures, from the results analysis of the dual channel architecture it was concluded that failure detection and isolation is not sufficient, but reconfiguration of the control laws in the remaining computer is also necessary. As explained before, the effect of a failure-by-output-omission in the computer is equivalent to the effect of detecting any failure in the primary flight computer and isolating the failure by shutting the computer down. Figure 5.5(a) shows that this strategy would not work. It is also necessary to reconfigure the control laws in the remaining computer after the failed computer is shut down.

The proposed enhanced dual channel architecture (EDCa) is very similar to the dual channel architecture presented in Fig. 5.2 in the sense that it has the same components connected in a very similar way. The three main differences are: within each PFC, there is a failure self-detection circuit (PFC_1-SD and PFC_2-SD) that will be the core of the FDIR mechanism described later. Each PFC exchanges information of its status (failed or operational) with the other PFC. The control laws within the processors of each PFC are reconfigurable depending on the status of the other PFC. These additional components and features allow the implementation of an FDIR mechanism with the following features:

- **Detection.** The self-detection circuit PFC-SD implemented in each PFC checks the range of the output signals of the PFC processor, and if they are within a certain range, then they are considered valid, otherwise the self-detection circuit reports a failure. Additionally, the rate of change of the outputted signals is checked, and if the self-detection circuit detects no rate of change, then a failure is reported.
- **Isolation.** Once the self-detection circuit detects a failure, the main PFC processor is shut down.
- **Reconfiguration.** Once the self-detection circuit detects a failure, a reconfiguration signal is sent to the remaining PFC to double the gain of the control surface actuation subsystems controllers. This reconfiguration strategy should compensate for the fact that only the control surface actuation subsystems commanded by the remaining computer are operational.

The proposed FDIR candidate should handle failure-by-output-omission, failure-by-stuck-output, and failure-by-random-output. It is not clear whether failure-by-delayed-output will be handled effectively by this FDIR. This will be explored further in the next section.

Table 5.3 collects the information corresponding to the failure models of the PFC-SDs introduced in the enhanced dual channel architecture — the self-detection circuits for each PFC. The failure models parameters for the rest of the components are the same as for the pure redundancy architecture, Table 5.1.

5.5.1 Performance and Reliability Evaluation

The conditions under which the enhanced dual channel architecture was evaluated are the same as the ones used to evaluate the pure redundancy architecture, i.e., the aircraft

Table 5.3: Enhanced dual channel architecture: primary flight computers' self-detection circuits failure model parameters.

Component	*Failure modes*	*Description*	U_f	$\lambda(/h)$
PFC_1-SD, PFC_2-SD	Omission	Output set to zero regardless of input	1	10^{-8}
	Commission	Output set to one regardless of input	2	10^{-8}

cruising at an altitude of 10, 668 m with forward velocity $V = 178$ m/s, pitch angle $\alpha_0 = 0.216$ rad, same control input, and the same configuration evaluation time ($t_c = 20$ s) as for the evaluation of the dual channel architecture.

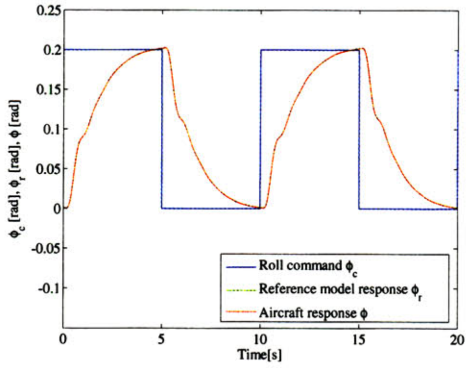
Table 5.4 shows the number of single points of failure and the probability of system failure (unreliability) at the end of the maintenance period for different levels of truncation. Truncating at the third level of failure yields a system unreliability of $1.17 \cdot 10^{-4}$, which meets the unreliability design requirement ($Q < 5 \cdot 10^{-4}$). This design still does not meet the single fault-tolerance requirement as there are still 5 single points of failure.

Table 5.4: Enhanced dual channel architecture: single points of failure and unreliability for different levels of truncation and an evaluation time of $500h$.

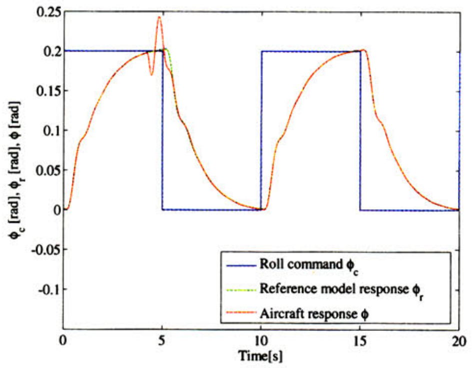
Truncation level	*Unreliability lower bound*	*Unreliability upper bound*	Single points of failure	*# of system configurations*
2	$1.14 \cdot 10^{-4}$	$1.87 \cdot 10^{-4}$	5	68
3	$1.17 \cdot 10^{-4}$	$1.17 \cdot 10^{-4}$	5	3924

Single primary flight computer failures

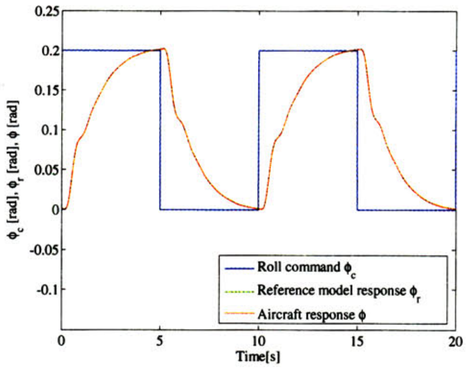
With the FDIR mechanism, it can be seen that except for the failure-by-delayed-output, Fig. 5.8(d), which causes the system to fail; the other PFCs failures are detected and isolated, and the aircraft stays stable after the control laws in the remaining PFC are reconfigured, Fig. 5.8(a) — Fig. 5.8(c). Additionally, the aircraft behavior is almost unaffected for a failure-by-output-omission, Fig. 5.8(a); and for a failure-by-stuck-output, Fig. 5.8(c). For a failure-by-random-output, there is a small transient after the failure occurs, as can be seen in Fig. 5.8(b), but the aircraft recovers in less than 2s.



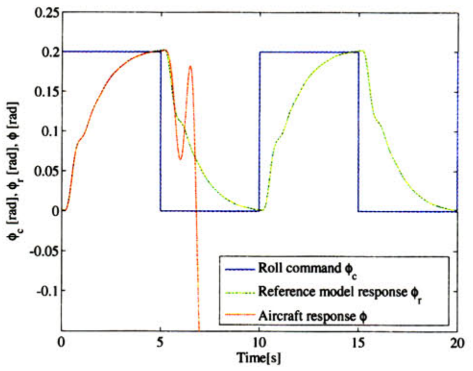
(a) Output omission failure mode.



(b) Random output between -5 and +5 failure mode



(c) Output stuck failure mode.

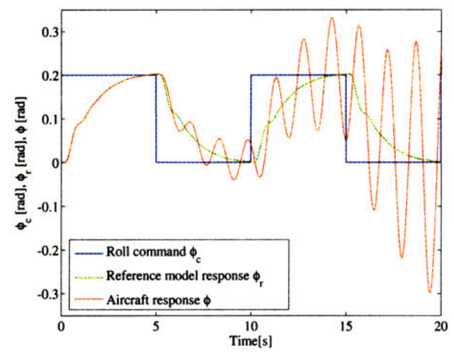


(d) Output delayed failure mode.

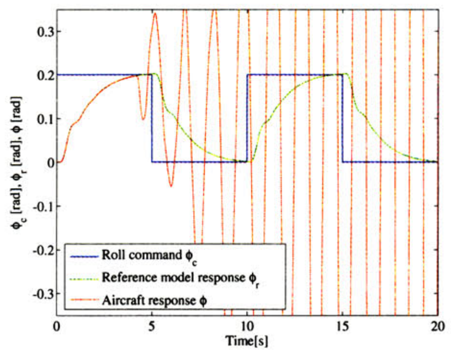
Figure 5.8: Enhanced dual channel architecture performance in response to a 0.2 rad, 0.1 Hz square wave in roll command ϕ_c. Aircraft roll angle response ϕ compared to reference aircraft model response ϕ_r, for different single failure modes in one of the primary flight computers, and a failure injection time $t_f = 4$ s.

Primary flight computer failures after failures in the PFC-SD

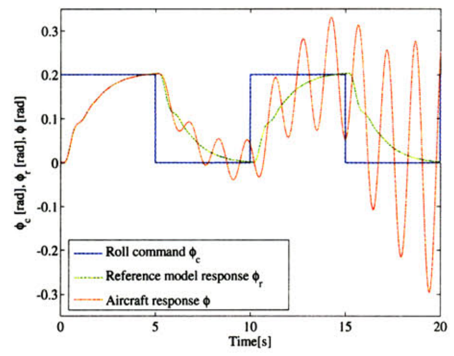
Although not shown here, single failures in a PFC-SD do not affect the aircraft performance at all. In the case of a single failure-by commission of a PFC-SD, the computer which has the PFC-SD will be shut down despite the fact that the computer did not fail itself, and the control laws in the remaining computer will be reconfigured. Although these failures will not cause the system to fail, if not announced, they can create a dangerous situation in which the system is believed to have both computers operational. This last problem becomes obvious when analyzing the results shown in Fig. 5.9. A first failure-by-omission in a PFC-SD circuit followed by a failure in its own PFC will cause the system to fail.



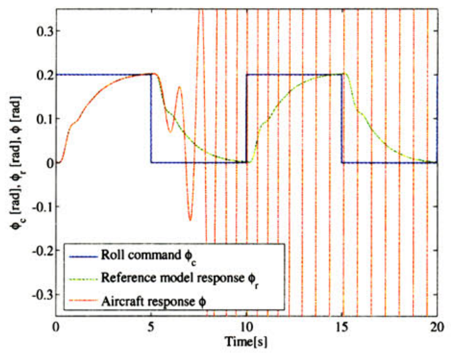
(a) PFC-self check circuit omission failure mode followed by an output omission failure mode.



(b) PFC-self check circuit omission failure mode followed by a random output between -5 and +5 failure mode



(c) PFC-self check circuit omission failure mode followed by an output stuck failure mode.



(d) PFC-self check circuit omission failure mode followed by an output delayed failure mode.

Figure 5.9: Enhanced dual channel architecture performance. Aircraft roll angle response ϕ compared to reference aircraft model response ϕ_r, for different failure sequences initiated by an omission failure mode in the PFC-SD circuit, and followed by different failure modes in one of the PFCs.

5.6 Dual-Dual Channel Architecture: Introducing Lock-Step Processors

The FDIR mechanism introduced in the previous section improves the system overall performance in the sense that unreliability is smaller and several single failures have been removed. However, it is not perfect since it is not capable of detecting failure-by-delayed output in the PFCs. In this section, an improved FDIR mechanism will be introduced, which will handle all the failures within the PFCs.

The proposed further improved architecture, called dual-dual channel architecture (DDCa) is very similar to the EDCa architecture. The main difference is that now each PFC will contain a pair of lock-step processors receiving the same inputs and doing exactly the same computations. Additionally, a dual-comparator circuit PFC-DC will compare their outputs. As in the EDCa design, each PFC exchanges status information (failed or operational) with the other PFC. The control laws within the processors of each PFC are reconfigurable depending on the status of the other PFC. With these additional elements the FDIR has the following features:

- **Detection.** The dual-comparator circuit within each PFC will check whether the outputs of the processor pair agree or not. If the outputs disagree, the dual-comparator reports a failure.
- **Isolation.** Once the dual-comparator circuit detects a failure, both lock-step processors within the failed computer are shut down.
- **Reconfiguration.** Once the dual-comparator circuit detects a failure, a reconfiguration signal is sent to the remaining PFC to double the gain of the control surface controllers on each lock-step processor.

The use of lock-step processors requires the introduction of an additional processor in each PFC, thus increasing its complexity and cost. However, the use of lock-step processors should result in a perfect detection, isolation, and reconfiguration of any failure within a PFC, thus removing all the PFC-related single points of system failure.

Table 5.5: Dual-dual channel architecture: primary flight computers' dual-comparator circuits failure model parameters.

Component	*Failure modes*	*Description*	U_f	$\lambda(/h)$
PFC_1-DC, PFC_2-DC	Omission	Output set to zero regardless the input	1	10^{-8}
	Commission	Output set to one regardless the input	2	10^{-8}

Table 5.5 displays the failure model information for the dual-comparator circuits within each PFC. The failure model parameters for the rest of the components is the same as for the dual channel architecture, Table 5.1.

5.6.1 Performance and Reliability Evaluation

The dual-dual channel architecture was evaluated under the same conditions as the dual channel architecture and the enhanced dual channel architecture, see Section 5.4.1 for the details. Table 5.6 shows shows the number of single points of failure and the system unreliability estimates. Truncating at the third level of failure yields a system unreliability of $1.51 \cdot 10^{-5}$ and a truncation error of $2.76 \cdot 10^{-7}$. This is an improvement of one order of magnitude with respect to both the dual architecture and the enhanced dual architecture.

Table 5.6: Dual-dual channel architecture: single points of failure and unreliability for different levels of truncation and an evaluation time of $500h$.

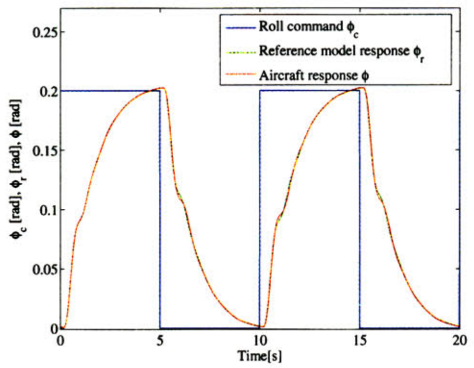
Truncation level	*Unreliability lower bound*	*Unreliability upper bound*	Single points of failure	*# of system configurations*
2	$1.49 \cdot 10^{-5}$	$9.47 \cdot 10^{-5}$	3	76
3	$1.51 \cdot 10^{-5}$	$1.51 \cdot 10^{-5}$	3	4640

This design still does not meet the single fault tolerance requirement. However, as will be shown, the single failures are exclusively control surface related. Therefore, this design is a potential candidate to fulfill the single fault tolerance requirement if appropriate control strategies are implemented without making any other architectural modifications; e.g., adding more redundancy.

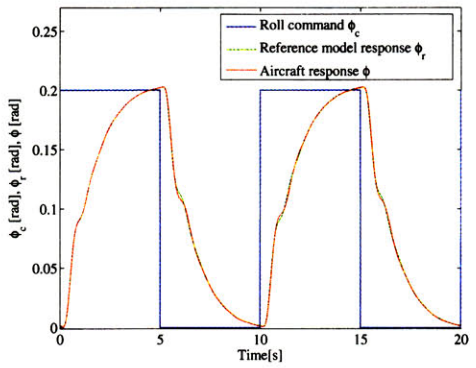
Figure 5.10 completes the analysis of the dual-dual architecture. It shows the aircraft behavior for all possible single failures of one of the PFC processors. As can be seen, the aircraft performance is not altered by any of these failures, thus the FDIR provided by the lock-step processors scheme is satisfactory. Although not shown here, single failures in the PFC-DC cause problems similar to those mentioned in the analysis of the enhanced dual channel architecture for single failures in any PFC-SD. Thus, for reasons similar to those given in section 5.5.1, a failure-by-omission or by-commission in a PFC-DC will not affect the system performance.

Table 5.7: Results comparison.

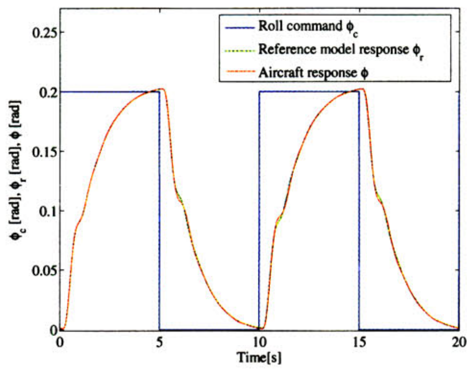
Architecture	*Unreliability*	*PFC-originated single points of failure*	*Control-surface-originated single points of failure*
DCa	$5.20 \cdot 10^{-4}$	8	3
EDCa	$1.17 \cdot 10^{-4}$	2	3
DDCa	$1.51 \cdot 10^{-5}$	0	3



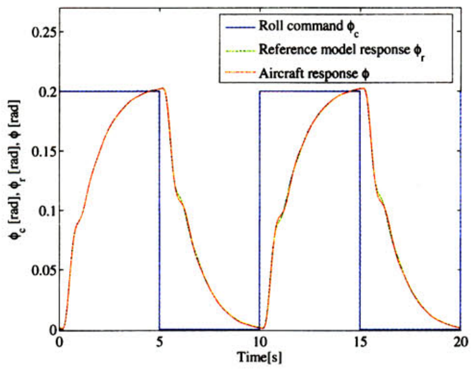
(a) Output omission failure mode.



(b) Random output between -5 and +5 failure mode



(c) Output stuck failure mode.



(d) Output delayed failure mode.

Figure 5.10: Dual-dual channel architecture performance in response to a 0.2 rad, 0.1 Hz square wave in roll command ϕ_c. Aircraft roll angle response ϕ compared to reference aircraft model response ϕ_r, for different single failure modes in one of the processors of a primary flight computer, and a failure injection time $t_f = 4$ s.

5.7 Architecture Comparisons

Table 5.7 summarizes the main analysis results for the three architectural solutions. The unreliability of both the DCa and the EDCa designs is very similar; however, the EDCa design removed 75% of the single points of system failure due to PFC failures with respect to the DCa design. The introduction of the lock-step processors in the DDCa design made a huge impact on the results. Its unreliability is one order of magnitude smaller than in the other two designs, and all the single points of system failure due to PFC failures were effectively removed.

However, it is important to note that the DCa design is the less complex one in terms of operation, i.e., there is no need for detecting, isolating or reconfiguring the system, and in the number of components. These two issues make this architecture less expensive to produce and to maintain. On the other hand, the DDCa design is much more complex than the other two: more components, and more complex operation, which makes this design more expensive to produce and to maintain. However, the impact on reliability and fault-tolerance is very important. Therefore, it is the work of the designer to make trade-offs between complexity and cost, and among performance, reliability, and fault-tolerance.

5.8 Conclusions

This chapter illustrated the power of the methodology presented in Chapter 3 (and its supporting tool InPRESTo) for guiding the design process of a fault-tolerant system by uncovering weak design points and pointing out possible ways to improve the system design. It also showed how important FDIR is to achieve fault-tolerance, and how InPRESTo can be used to test the effectiveness of different FDIR strategies. Finally, it also showed how to compare different architectural solutions from different points of view: reliability, performance, and fault-tolerance.

The analysis shown in this chapter opens up further research questions. The design improvement shown here was done manually, analyzing the results for each iteration and using expert judgment to guide the changes in the design. Guiding the design in a structured and automatic way is a challenging problem worthy of exploring, i.e., how to extend the methodology to automatically determine where the problems that most impact a design are, and how they could be discussed. These issues are further addressed in Chapter 7.

Notation Used in this Chapter

FDIR:	Failure detection, isolation, and recovery
IMU_1, IMU_2, IMU_3:	Inertial measurement units
LA:	Left aileron
LAAS_1, LAAS_2:	Left aileron actuation subsystems
LAPS_1, LAPS_2, LAPS_3:	Left aileron position sensors
PCA:	Propulsion controllred aircraft
PFC_1, PFC_2:	Primary flight computers
PFC_1-SD, PFC_2-SD:	Primary flight computers failure self-detection circuits
PFC_1-DC, PFC_2-DC:	Primary flight computers dual-comparator circuits
p_b :	Roll rate
p_{b_r} :	Reference model roll rate
R:	Rudder
RAS_1, RAS_2:	Rudder actuation subsystems
RPS_1, RPS_2, RAPS_3:	Rudder position sensors
RA:	Right aileron
RAAS_1, RAAS_2:	Right aileron actuation subsystems
RAPS_1, RAPS_2, RAPS_3:	Right aileron position sensors
r_b :	Yaw rate
r_{b_r} :	Reference model yaw rate
t :	Time
t_c :	Configuration evaluation time
t_f :	Failure injection time
T :	System global evaluation time
U_f :	Behavioral modes random variable
V :	Forward velocity
Z_1 :	Sideslip angle performance metric
Z_2 :	Roll rate performance metric
Z_3 :	Yaw rate performance metric
Z_4 :	Roll angle performance metric
α_0 :	Pitch angle
β :	Sideslip angle
β_r :	Reference model sideslip angle
δ_a^l, δ_a^r :	Left and right aileron angles
δ_c :	Yaw command
δ_r :	Rudder angle
λ :	Failure rate
Ω_{Z_1} :	Set of requirements for performance metric Z_1
Ω_{Z_2} :	Set of requirements for performance metric Z_1
Ω_{Z_3} :	Set of requirements for performance metric Z_1
Ω_{Z_4} :	Set of requirements for performance metric Z_1
ϕ :	Roll angle
ϕ_c :	Roll command
ϕ_r :	Reference model roll angle

Chapter 6

Steer-by-Wire/Brake-Actuated-Steering System Case-Study

In this chapter, two conceptually very different designs to achieve fault-tolerance in a steer-by-wire (SbW) system are presented. The first one —referred as SbW design, is based on the replication of components and the introduction of failure detection, isolation, and reconfiguration mechanisms. In the second design –referred as SbW/BAS design, a dissimilar backup mechanism, called brake-actuated steering (BAS), is used to achieve fault-tolerance rather than replicating each component within the system. This chapter complements Chapter 5 by showing how the performance and reliability evaluation SIMULINK ® toolbox –InPRESTo– can be used to compare very different architectural approaches to achieve fault-tolerance.

6.1 Introduction

Safety-critical systems in a car, for example, hydraulic brakes and power steering, require a secondary or backup activation mechanism to prevent catastrophic failure. In a conventional power assisted steering system, the mechanical connection between the driver and the steering rack serves as the secondary steering mechanism [44]. In a SbW system the secondary steering mechanism cannot rely on a mechanical link (e.g., the steering column connection to the pinion) between the steering wheel and the steering rack since the goal of SbW is to eliminate such a mechanical connection. Lacking this connection, SbW design efforts have focused on developing fault-tolerant systems based on redundancy, i.e., the duplication of components and modules at all levels [1], [44], [45], [46]. This solution is widely applied in aircraft, but it adds a significant amount of complexity and cost. Therefore, it is important to explore alternatives to classical redundancy for achieving system integrity.

In this chapter, a case-study of a steer-by-wire system is presented. Two different approaches to achieve fault tolerance are presented, analyzed, and compared. The first approach is based on the use of redundancy and failure detection, isolation, and reconfiguration (FDIR) mechanisms, as in the flight control system presented in Chapter 5. The second approach is basically a single-string system for the computing and actuation elements dedicated to traditional steering. The fault-tolerance is achieved by using brake-actuated-steering [43] as a backup mechanism to overcome component failures in the SbW that otherwise would have catastrophic consequences. Brake-actuated-steering utilizes the already-existing selective wheel braking capability provided by the antilock-braking-system (ABS), and electronic stability programs (ESP) to actuate the steering mechanism. As mentioned in Chapter 5, similar approaches have been proposed by NASA, in order to develop the technology for future aircraft designs for emergency flight control, using engine thrust to augment or replace the flight control system [57]. Both approaches will be analyzed, and the advantages and disadvantages of using each one will be highlighted. Section 6.2 presents a detailed analysis of the first solution in which classical redundancy and FDIR are used to achieve fault tolerance. The steer-by-wire system with the dissimilar backup mechanism, SbW/BAS, is introduced and analyze in Section 6.3. The comparison of both solutions is presented in Section 6.4. Concluding remarks are presented in 6.5.

6.2 Fault-Tolerant Steer-by-Wire System

The proposed SbW architecture, Fig. 6.1, is based on the fault-tolerant architecture introduced in Section 5.6 for the lateral-directional flight control system, which proved to be an appropriate solution to achieve a high level of fault-tolerance. The architecture is composed of two redundant steer-by-wire computers ($SbWC_1$ and $SbWC_2$) that receive information about the vehicle road wheel angle δ from triple redundant road wheel angle sensors ($RWAS_1$, $RWAS_2$ and $RWAS_3$) cross strapped to both computers; and also receive information about the driver steering wheel command δ_w from triple redundant steering wheel angle sensors ($SWAS_1$, $SWAS_2$ and $SWAS_3$), also cross strapped to both computers. Each SbWC has a voting algorithm to compute the vehicle road wheel angle from the redundant RWASs measurements and the driver command from the redundant SWASs measurements. Both SbWCs have control laws that, based on the RWASs measurements and the driver command measured by the SWASs, compute the appropriate commands for two redundant rack actuation subsystems ($RaAS_1$ and $RaAS_2$), which are responsible for positioning the steering rack (SRa). Each rack actuation subsystem is commanded in-

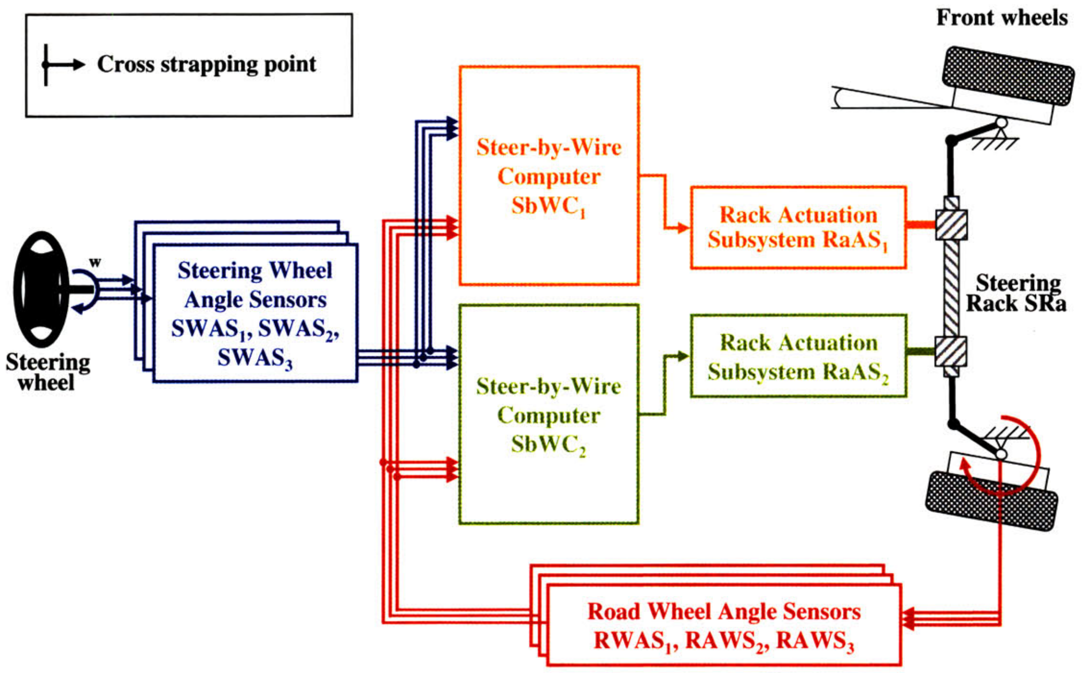

Cross strapping point
Front wheels
Steer-by-Wire Computer SbWC$_1$
Rack Actuation Subsystem RaAS$_1$
Steering Wheel Angle Sensors SWAS$_1$, SWAS$_2$, SWAS$_3$
w
Steering wheel
Steering Rack SRa
Rack Actuation Subsystem RaAS$_2$
Steer-by-Wire Computer SbWC$_2$
Road Wheel Angle Sensors RWAS$_1$, RAWS$_2$, RAWS$_3$

Figure 6.1: SbW design: steer-by-wire system with replicated components and failure detection, isolation, and reconfiguration mechanisms to achieve fault-tolerance.

dependently by each SbWC (not cross strapped). Each actuation subsystem is composed of a current-controlled electric motor. When one of the rack actuation subsystems fails, it remains within the control loop (connected to the SRa), and the other actuation subsystem must accommodate any impacts the failed one may have on moving the rack.

To have perfect computer failure detection of random failures, each SbWC is internally redundant. Each SbWC is made up of a pair of lock-step processors receiving the same inputs and doing exactly the same computations. A dual-comparator circuit within each SbWC (SbWC$_1$-DC and SbWC$_2$-DC) compares the outputs from both lock-step processors in its SbWC. If the outputs disagree, a failure has been detected, and an isolation and reconfiguration process starts. The isolation is accomplished by the dual comparator circuit sending a signal to the power management system to shut down the processors in the faulty SbWC. The reconfiguration starts when the dual-comparator within the faulty computer sends a signal to the remaining SbWC to reconfigure the control laws that are implemented in both of its lock-step processors to accommodate the extra loads of the (now uncommanded) RaAS.

The component behavioral models for each component are described in Appendix D. The component failure model parameters are collected in Table 6.1. Column 2 lists the possible failure modes for each component, column 3 is an explanation of each behavioral model mode. U_f in column 4 is the variable that assigns the corresponding failure mode to the component behavioral model equations (see Appendix D). The failure rate λ associated with each failure mode is collected in the last column of the table.

Table 6.1: Component failure model parameters.

Component	*Failure modes*	*Description*	U_f	$\lambda(/h)$
$SbWC_1$, $SbWC_2$	Omission	Output set to zero	1	$2 \cdot 10^{-7}$
	Random	Random output between -5 and 5	2	10^{-7}
	Stuck	Output stuck at last correct value	3	10^{-7}
	Delayed	Output delayed 0.2 s	4	10^{-7}
$SbWC_1$-DC, $SbWC_2$-DC	Omission	Output set to zero regardless the input	1	10^{-8}
	Commission	Output set to one regardless the input	2	10^{-8}
$RaAS_1$, $RaAS_2$	Omission	Output set to zero	1	10^{-6}
	Stuck	Output stuck at last correct value	2	10^{-6}
SRa	Stuck	Road wheel angle stuck at last correct position	2	10^{-7}
$RWAS_1$, $RWAS_2$, $RWAS_3$, $SWAS_1$, $SWAS_2$, $SWAS_3$	Omission	Output set to zero	1	$4 \cdot 10^{-7}$
	Gain change	Output scaled by a factor of 1.5	2	$3 \cdot 10^{-7}$
	Biased	Output biased by a factor of 0.3	3	$3 \cdot 10^{-7}$

A linear single-track vehicle dynamics model [58], interacting with the architecture descried above, completes the system behavioral model. The models inputs are the road wheel angle δ, and the state variables are the sideslip β, and the yaw rate r_b. See Appendix D for a detailed description of the linear single-track vehicle dynamics model.

6.2.1 Performance Metrics Definition and Associated Requirements

The performance metrics chosen are the vehicle state variables: the sideslip $\beta(t)$, the yaw rate r_b, and the heading angle Ψ. Thus

$$Z_1 = \beta(t), \tag{6.1}$$

$$Z_2 = r_b(t), \tag{6.2}$$

$$Z_3 = \Psi(t). \tag{6.3}$$

The dynamic behavior of a reference vehicle will be used to define the performance metrics requirements. This reference vehicle state variables are denoted by the sub index $_r$, i.e., the sideslip is denoted by $\beta_r(t)$, the yaw rate by $r_{b_r}(t)$, and the heading angle by $\Psi_r(t)$. Thus, the performance metrics requirements are defined as

$$\Omega_{Z_1} = \{\beta(t) \in \mathbb{R} \,/\, \| \beta(t) - \beta_r(t)\|_\infty \leq r_\beta\}, \tag{6.4}$$

$$\Omega_{Z_2} = \{r_b(t) \in \mathbb{R} \,/\, \| r_b(t) - r_{b_r}(t)\|_\infty \leq r_{r_b}\}, \tag{6.5}$$

$$\Omega_{Z_3} = \{\Psi(t) \in \mathbb{R} \,/\, \| \Psi(t) - \Psi_r(t)\|_\infty \leq r_\Psi\}, \tag{6.6}$$

where $r_\beta = 0.25$ deg, $r_{r_b} = 0.6$ deg/s, and $r_\Psi = 0.5$ deg.

A 10 deg, 0.5 Hz sinusoidal steering wheel angle input δ_w will be used as system control input to evaluate each configuration.

6.2.2 Results Analysis

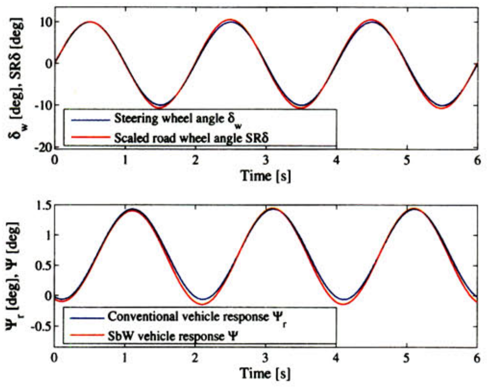
The system was evaluated for a vehicle speed (of center of gravity) $V = 70$ km/h. The vehicle time constants dictate a configuration evaluation time $t_c = 6$ s. A vehicle lifetime of 15 years and an average of 400 working hours per year was considered for the reliability evaluation, which gives an evaluation time $T = 6000$ h.

Table 6.2: SbW design: unreliability for an evaluation time of $6000h$.

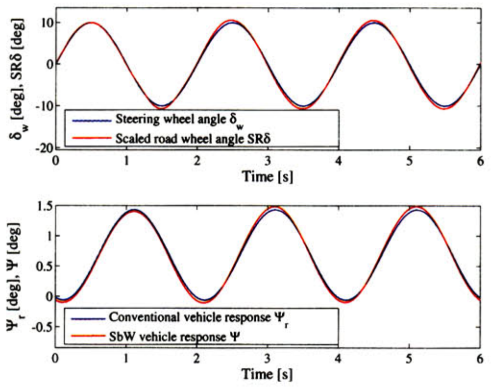
Truncation level	*Unreliability lower bound*	*Unreliability upper bound*	*# of system configurations*
3	$1.10 \cdot 10^{-3}$	$1.13 \cdot 10^{-3}$	1716

Table 6.2 shows the probability of system failure (unreliability) at the end of the vehicle lifetime. Truncating after three component failure events yields a system unreliability upper bound of $1.13 \cdot 10^{-3}$ and a lower bound of $1.10 \cdot 10^{-3}$.

The remaining analysis will display the vehicle behavior for different component component failure modes, i.e, the road wheel angle δ scaled by the steering ratio SR, the vehicle heading angle response Ψ (and the vehicle heading angle response Ψ_r of a vehicle with a conventional mechanical steering system as a reference), for a 10 deg, 0.5 Hz sinusoidal steering wheel angle input δ_w. Figure 6.2 shows the vehicle behavior for the two possible failure modes of either rack actuation subsystem. In both cases, failure-by-output-



(a) Rack actuation subsystem output failing to provide commanding torque for the rack.



(b) Rack actuation subsystem failing stuck, and thus providing a constant torque for the rack.

Figure 6.2: SbW design performance in response to a 10 deg, 0.5 Hz sinusoidal steering wheel input at 70 km/h vehicle speed. Steering wheel angle δ_w compared to road wheel angle (scaled to steering wheel angle $SR\delta$) under failure conditions, and vehicle heading angle Ψ_r with a mechanical steering system compared to the vehicle (with the SbW design) heading angle response Ψ, under failure conditions. Both performance comparisons shown for two failure modes of the rack actuation subsystem, and for a failure injection time $t_f = 1$ s.

omission, Fig. 6.2(a), and failure-by-stuck-output, Fig. 6.2(b), the vehicle performance is almost unaffected. The difference between the heading angle response Ψ and the heading angle reference response Ψ_r is less than 0.1deg in both cases. Although there is no FDIR mechanism to account for single failures in one of the actuation subsystems, the fault-tolerance is provided by having a redundant rack actuation subsystem.

The component failure of interest in the analysis, for the reasons explained in the Section 6.3, corresponds to a steering rack failure where it is stuck, Fig. 6.3. In this case, the steering rack gets stuck at a fixed position and the steering rack actuation subsystem is not able to position it according to the driver's command. Thus, the vehicle heading angle does not track the model reference heading angle. This is considered a catastrophic failure because the driver is not able to control the vehicle direction. However, the same would occur with a conventional mechanical steering system

Although not displayed here, any failure in any of the lock-step processors within each computer does not affect the vehicle performance at all. The dual-comparator circuit will detect every disagreement between processors within the pair, shutting down the pair, and reconfiguring the control law within the remaining computer to account for this failure. The same occurs when any of the steering wheel angle sensors, or the road wheel

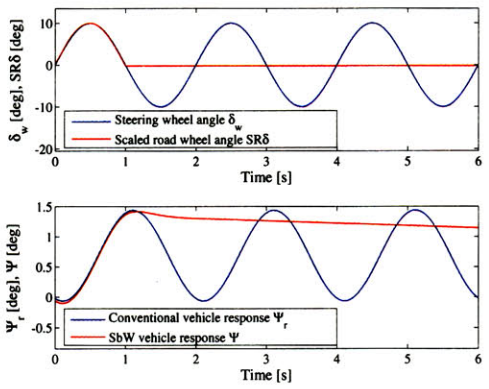


Figure 6.3: SbW design performance in response to a 10 deg, 0.5 Hz sinusoidal steering wheel input at 70 km/h vehicle speed. Steering wheel angle δ_w compared to road wheel angle (scaled to steering wheel angle $SR\delta$) under failure conditions, and vehicle heading angle Ψ_r with a mechanical steering system compared to the vehicle (with the SbW design) heading angle response Ψ, under failure conditions. Both performance comparisons shown for the steering rack failing by getting stuck at a certain position, and for a failure injection time $t_f = 1$ s.

angle sensors fail: the voting algorithms implemented in the computers will disregard the measurement yielded by the faulty sensor. Failures by omission, or by commission in the dual-comparator circuit do not affect the vehicle performance. In the case of a failure-by-commission (a false alarm that shuts down the computer) of the self-detection circuit, the computer with the faulty dual-comparator circuit will be shut down despite the fact that it did not fail, and the control laws in the remaining computer will be reconfigured. In the case of the failure-by-omission, although the system will not fail after this failure occurs, if not announced, it can create a dangerous situation. This is due to the fact that the system is believed to have both computers operational, when they are not. Therefore, a first failure-by-omission in the dual-comparator circuit of an SbWC, followed by a failure in the same SbWC, will go undetected causing the system to fail.

6.3 Steer-by-Wire/Brake-Actuated-Steering System

In this section, a substantially different architectural solution from the one in Section 6.2 is presented. This approach implements brake-actuated-steering (BAS) as a means to achieve fault tolerance, thus reducing the amount of component redundancy. This will result in a less complex steer-by-wire system, which may result, as well, in a less expensive system. Furthermore, the additional components that are necessary for implementing BAS are already present in the vehicle as part of other safety systems, such as anti-lock braking system (ABS) or electronic stability programs (ESP). Additionally, in the steer-by-wire architecture in Section 6.2, despite the presence of component redundancy and FDIR to achieve fault-tolerance; it was shown that a stuck failure of the steering rack would result in a single point of failure. To overcome this, a SbW/BAS architecture will be presented that implements a skid steering algorithm.

The proposed SbW/BAS architecture is showed in Fig. 6.4. The only elements common to both SbW and BAS are the road wheel angle sensors RWASs and the steering wheel angle sensors SWASs. Therefore these elements are triple redundant (with their corresponding voting algorithms within the steer-by-wire computer SbWC and the brake-actuated-steering computer BASC) to avoid common modes of failure in both SbW and BAS. Unlike the previously discussed fault-tolerant SbW architecture, there is only one SbWC with a single processor inside; and there is only one rack actuation subsystem (RaAS). Therefore, fault-tolerance cannot be achieved with these elements alone. The way this system achieves fault tolerance is by using selective wheel braking to steer the vehicle when the main SbW functionality is lost. When a failure results in disagreement between the steering wheel angle δ_w and the road wheel angle δ, the failure detection circuit (SbWC-FD) implemented in the SbWC shuts down the processor inside the SbWC, and thus, disables the main SbW functionality. Then, the SbWC-FD sends a reconfiguration signal to the brake-actuated-steering computer (BASC), which is in hot-standby, and the BAS computer starts sending commands.

The BAS computer has two different control laws implemented: one for brake-actuated steering, and one for skid steering. The BAS controller kicks in after a failure in the main SbW has been detected, and first attempts to steer with brake-actuated steering commanding both front left and right caliper actuation subsystems (FLCAS and FRCAS). This will produce the appropriate longitudinal braking forces in the front left and right tires, enabling the vehicle to be steered according to the steering wheel angle command and the rack position. Thus the BAS overcomes the dangerous situation produced by the failure of the main SbW functionality.

Additionally, if the steering rack gets stuck, the BAS computer will detect a disagreement between the steering wheel angle δ_w and the road wheel angle δ; and the BAS computer can no longer position the rack with the BAS control law. Therefore, it will switch to the skid-steering control law. In the skid-steering mode, the BAS computer commands all four wheel caliper actuation subsystems (FLCAS, FRCAS, RLCAS, and RRCAS) to produce ap-

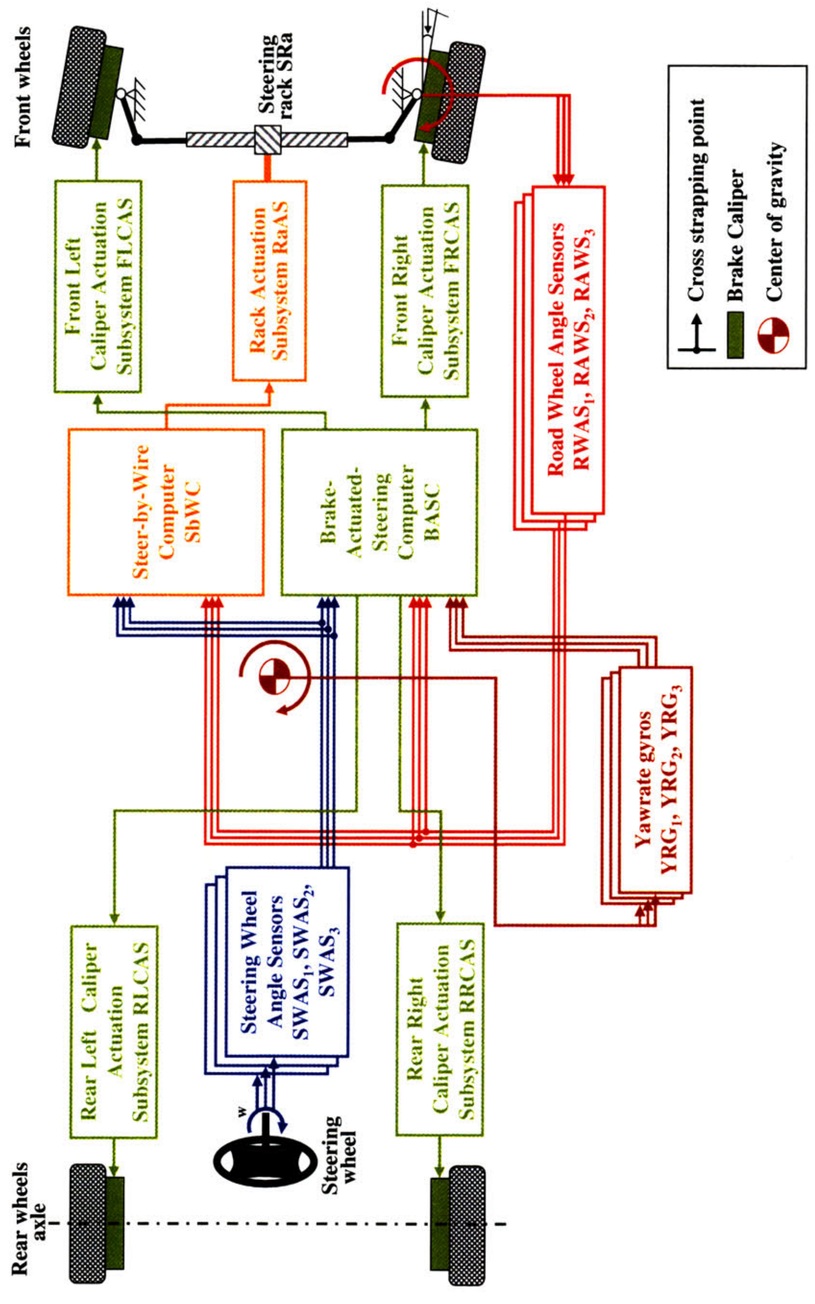

Front wheels
Front Left Caliper Actuation Subsystem FLCAS
Rack Actuation Subsystem RaAS
Steering rack SRa
Front Right Caliper Actuation Subsystem FRCAS
Steer-by-Wire Computer SbWC
Brake-Actuated-Steering Computer BASC
Road Wheel Angle Sensors RWAS$_1$, RAWS$_2$, RAWS$_3$
Yawrate gyros YRG$_1$, YRG$_2$, YRG$_3$
Steering Wheel Angle Sensors SWAS$_1$, SWAS$_2$, SWAS$_3$
w
Steering wheel
Rear Left Caliper Actuation Subsystem RLCAS
Rear Right Caliper Actuation Subsystem RRCAS
Rear wheels axle
Cross strapping point
Brake Caliper
Center of gravity

Figure 6.4: SbW/BAS design: steer-by-wire system with a dissimilar backup mechanism (brake-actuated steering) to achieve fault tolerance.

propriate front and rear differential longitudinal braking forces to steer the vehicle. Since the steering rack position is no longer a valid measurement to determine the vehicle heading, three redundant yaw rate gyros (YRG_1, YRG_2, YRG_3) will measure the vehicle yaw rate.

Table 6.3: Component failure model parameters for the additional components of the SbW/BAS architecture.

Component	*Failure modes*	*Description*	U_f	$\lambda(/h)$
BASC	Omission	Output set to zero	1	$2 \cdot 10^{-7}$
	Random	Random output between −5 and 5	2	10^{-7}
	Stuck	Output stuck at last correct value	3	10^{-7}
	Delayed	Output delayed 0.2 s	4	10^{-7}
SbWC-FD	Omission	Output set to zero regardless the input	1	10^{-8}
	Commission	Output set to one regardless the input	2	10^{-8}
FLCAS, FRCAS, RLCAS, RRCAS	Omission	Output set to zero	1	10^{-8}
	Stuck	Output stuck at last correct value	2	10^{-8}
YRG_1, YRG_2, YRG_3	Omission	Output set to zero	1	$4 \cdot 10^{-7}$
	Gain change	Output scaled by a factor of 1.5	2	$3 \cdot 10^{-7}$
	Biased	Output biased by a factor of 0.3	3	$3 \cdot 10^{-7}$

The component behavioral models for the additional components introduced in this architecture, i.e., the brake-actuated-steering computer (BASC), the front left (and right) caliper actuation subsystems (FLCAS and FRCAS), and the rear left (and right) caliper actuation subsystems (RLCAS and RRCAS) are described in Appendix D. The component failure model parameters of these additional components, as well as the failure detection circuit implemented in the SbWC, are collected in Table 6.3. A linear two-track vehicle dynamics model [43], interacting with the architecture descried above, completes the system behavioral model. The models inputs are the road wheel angle δ and the the differential longitudinal forces at the front and rear tires. The state variables are the sideslip β, and the yaw rate r_b. See Appendix D for the description of the two-track vehicle dynamics model.

6.3.1 Results Analysis

The SbW/BAS system was evaluated for the same vehicle speed as before $V = 70$ Km/h, the same configuration evaluation time $t_c = 6$s, and the same global evaluation time T of 6000 h. The performance metrics definition (and associated requirements), as well as the control system input chosen to evaluate the SbW/BAS are the same as the ones used to evaluate the SbW design.

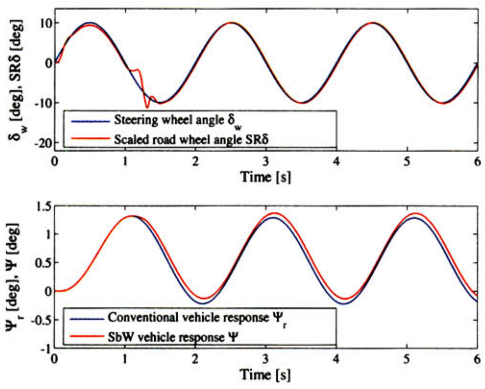
Table 6.4 displays the system unreliability results at the end of the vehicle lifetime for a truncation level of three. A system unreliability upper bound of $8.01 \cdot 10^{-4}$ and a lower bound of $7.02 \cdot 10^{-4}$ result. Although a comparison of the SbW and the SbW/BAS architectures will be carried in the next section, it is very interesting to note that the unreliability upper bound for the SbW/BAS architecture is smaller than the unreliability lower bound of the SbW architecture. This means that without taking the analysis to a further level of truncation, it can be concluded that the SbW/BAS is predicted to have better reliability than the SbW architecture.

Table 6.4: SbW/BAS design: unreliability for an evaluation time of 6000h.

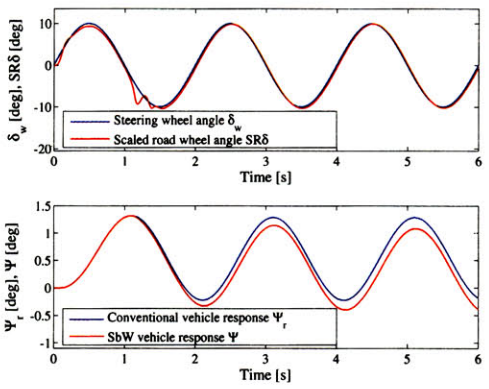
Truncation level	*Unreliability lower bound*	*Unreliability upper bound*	# of system configurations
3	$7.02 \cdot 10^{-4}$	$8.01 \cdot 10^{-4}$	2215

For the remainder of this section, the SbW/BAS system performance will be shown for different component failure modes. In each case, the vehicle road wheel angle δ scaled by the steering ratio SR will be compared to the steering wheel angle δ_w, and the vehicle heading angle response Ψ will be compared to the heading angle response Ψ_r of the reference vehicle model (a vehicle with a conventional mechanical steering system). These output responses will be displayed when a 10deg, 0.5Hz sinusoidal input is applied to the steering wheel. In order to show small errors between Ψ and Ψ_r, different scales has been used to display the comparison between δ_w and $SR\delta$; and between Ψ_r and Ψ.

As shown in Fig. 6.5, the two possible steering rack actuation subsystem failure modes cause a similar effect in the vehicle response. When the failure occurs at $t = 1$ s, the road wheel angle starts diverting from its reference value. When the absolute value of the difference between the steering wheel angle δ_w and the road wheel angle scaled by the steering ratio $SR\delta_w$ hits 2 deg, the failure detection circuit with the SbW computer sends a reconfiguration signal to the BASC computer. At this instant the BAS controller kicks in, taking over the control of the vehicle after the SbWC is shut down. The most important consequence after this failure is that the vehicle heading angle Ψ starts diverting from the vehicle reference response by a value of 0.23 deg for the failure-by-omission, and 0.11 deg for the failure-by-stuck, both considered acceptable to keep the vehicle on its heading path. A model of the driver has not been included in the system model, but in reality, the driver would remove this heading error.



(a) Rack actuation subsystem output failing to provide commanding torque for the rack.

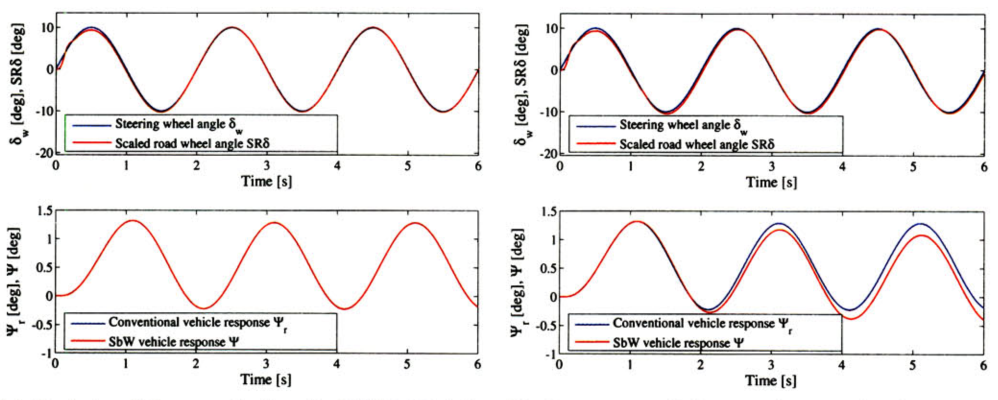


(b) Rack actuation subsystem failing stuck, and thus providing a constant torque for the rack.

Figure 6.5: SbW/BAS design performance in response to a 10 deg, 0.5 Hz sinusoidal steering wheel input at 70 km/h vehicle speed. Steering wheel angle δ_w compared to road wheel angle (scaled to steering wheel angle $SR\delta$) under failure conditions, and vehicle heading angle Ψ_r with a mechanical steering system compared to the vehicle (with the SbW/BAS design) heading angle response Ψ, under failure conditions. Both performance comparisons shown for different single failure modes in the rack actuation subsystem, and for a failure injection time $t_f = 1$ s.

Figure 6.6 displays the vehicle performance when failures occur in the failure detection circuit within the SbW computer. In this case, for a failure-by-omission (i.e., the detection circuit does not report a failure), Fig. 6.6(a), the vehicle response is completely unaffected. This is expected —a failure-by-omission will disable the detection of second failures, but it will not cause a degraded system behavior. In contrast, a failure-by-commission will cause a degraded system performance. In this case, the failure detection circuit sends a false alarm to the BASC computer, thus, the brake-actuated-steering controller takes over the control of the vehicle, resulting in the vehicle response displayed in Fig. 6.6(b).

The vehicle behaviors corresponding to different failure modes in the steer-by-wire computer are displayed in Fig. 6.7. A failure-by-omission, as displayed in Fig. 6.7(a), will cause a transient in the road wheel angle δ, but the vehicle heading angle Ψ is almost unaffected. In this case, when the computer stops sending commands out, the steering rack actuation subsystem stops commanding the steering rack for a few instants until the SbWC-FD detects the anomaly and switches control to the the BAS controller. In this scenario, the steering rack time constants are much smaller than the vehicle time constants, therefore the vehicle heading is almost unaffected. A similar behavior takes place when a random output failure occurs in the SbW computer, Fig. 6.7(b), There is a transient in the road wheel angle δ, but the heading angle Ψ is barely affected. The explanation in this



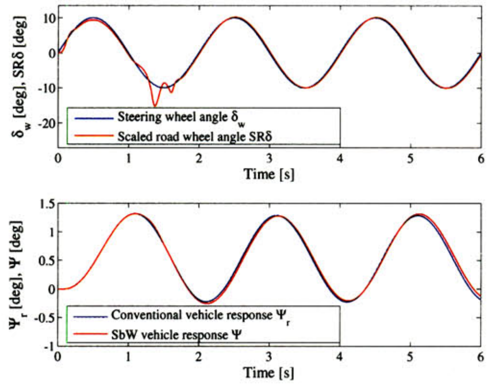
(a) Omission failure mode, i.e., the SbWC-FD fails to report a failure in the SbWC.

(b) Commission failure mode, i.e. the SbWC-FD reports a false alarm in the SbWC.

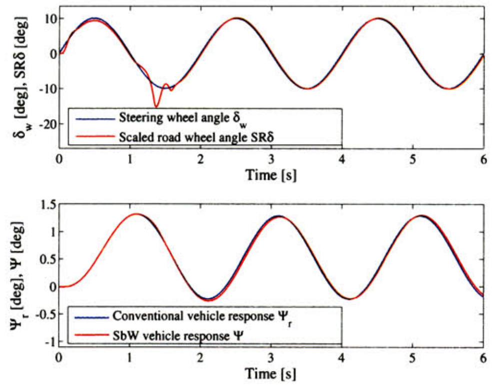
Figure 6.6: SbW/BAS design performance in response to a 10 deg, 0.5 Hz sinusoidal steering wheel input at 70 km/h vehicle speed. Steering wheel angle δ_w compared to road wheel angle (scaled to steering wheel angle $SR\delta$) under failure conditions, and vehicle heading angle Ψ_r with a mechanical steering system compared to the vehicle (with the SbW/BAS design) heading angle response Ψ, under failure conditions. Both performance comparisons shown for different single failure modes in the steer-by-wire computer failure detection circuit SbWC-FD, and for a failure injection time $t_f = 1$ s.

case is slightly different: the computer starts sending random commands to the steering rack actuation subsystem, which will try to position the rack accordingly. The actuation subsystem time constants are much smaller than the rack time constants; therefore, the steering rack only perceives the command as a random noise, which has an effect similar to not having commanded at all (as in the previous case). The essentially uncommanded steering rack is centered by the self-centering forces of the front wheels. The SbW-FD will detect the anomaly, and switch control to the BAS controller. Again, the vehicle dynamics are much slower than the rack dynamics, therefore the vehicle heading is almost unaffected.

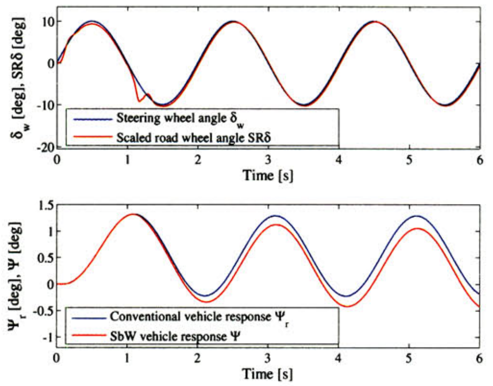
When the SbWC fails stuck, the vehicle behavior diverts from its nominal behavior as displayed in Fig. 6.7(c). In this case, the computer is sending a constant command to the rack actuation subsystem, which will try to position the rack accordingly. The SbWC-FD will detect the anomaly after a short transient and the BAS mechanism will take over; however the actuation subsystem acted long enough on the rack, causing the vehicle heading angle Ψ to divert from its nominal behavior response by 0.25 deg. A similar effect takes place when the output of the SbW computer gets delayed, Fig. 6.7(d). In this case, the steering rack actuation subsystem is receiving its command delayed, thus delaying the correct positioning of the rack. By the time the SbWC-FD detects the failure and the control is handed to the BAS controller, the vehicle heading has been modified from its nominal path by 0.40 deg. Both diversions are consider small enough to keep the vehicle on an acceptable heading path.



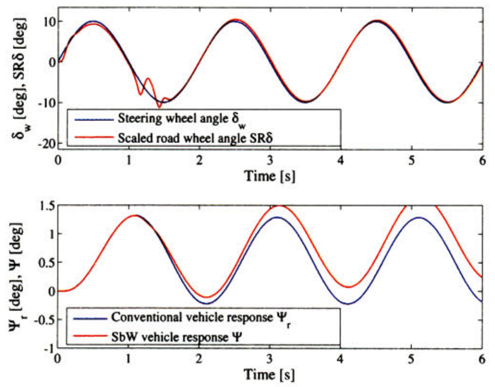
(a) SbWC output omission failure mode.



(b) SbWC random output between -5 deg and $+5$ deg failure mode



(c) SbWC output stuck failure mode.



(d) SbWC output delayed failure mode.

Figure 6.7: SbW/BAS design performance in response to a 10 deg, 0.5 Hz sinusoidal steering wheel input at 70 km/h vehicle speed. Steering wheel angle δ_w compared to road wheel angle (scaled to steering wheel angle $SR\delta$) under failure conditions, and vehicle heading angle Ψ_r with a mechanical steering system compared to the vehicle (with the SbW/BAS design) heading angle response Ψ, under failure conditions. Both performance comparisons shown for different single failure modes in the steer-by-wire computer, and for a failure injection time $t_f = 1$s.

Up to this point in the analysis of the SbW/BAS, there is nothing that has been achieved using BAS that could not be achieved with a fault-tolerant SbW architecture as presented in Section 6.2. Furthermore, for the failure modes analyzed in this section so far, better performance was achieved with the SbW architecture.

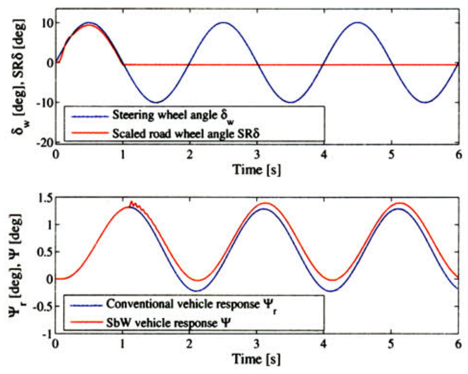


Figure 6.8: SbW/BAS design performance in response to a 10 deg, 0.5 Hz sinusoidal steering wheel input at 70 km/h vehicle speed. Steering wheel angle δ_w compared to road wheel angle (scaled to steering wheel angle $SR\delta$) under failure conditions, and vehicle heading angle Ψ_r with a mechanical steering system compared to the vehicle (with the SbW/BAS design) heading angle response Ψ, under failure conditions. Both performance comparisons shown for the steering rack failing by getting stuck at a certain position, and for a failure injection time $t_f = 1$ s.

The main advantage of the SbW/BAS architecture is displayed in Fig. 6.8, which shows the vehicle dynamic behavior when the steering rack gets stuck. In this case, the SbWC-FD will detect the anomaly, switching the control to the BAS computer. The brake-actuated-steering control law kicks in first, trying to control the vehicle, without success. The skid-steering controller will take control after it is detected that steering rack is not moving, despite the fact that the steering wheel angle is not constant. It can be seen that the heading angle Ψ error is small enough, 0.22 deg, for the driver to keep control of the vehicle and bring it to a safe stop.

Table 6.5: Reliability estimates for the SbW and the SbW/BAS designs at the end of vehicle lifetime (6000 h).

	Unreliability lower bound	*Unreliability upper bound*	*Single points of failure*
SbW	$1.10 \cdot 10^{-3}$	$1.13 \cdot 10^{-3}$	1
SbW/BAS	$7.02 \cdot 10^{-4}$	$8.01 \cdot 10^{-4}$	0

6.4 SbW-SbW/BAS Comparison

The system unreliability estimates for both the SbW and the SbW/BAS systems are displayed in Table 6.5. These results were obtained by truncating the analysis at the third level of failure. As can be noted from the results, the estimated unreliability intervals do not overlap. This means that without the need of taking the analysis to a further level of failure it can be concluded that the SbW/BAS has a better reliability estimate than the SbW architecture. Furthermore, with the SbW architecture it is not possible to achieve complete fault-tolerance at the first level of failure since a failure-by-stuck of the steering rack will cause the system to fail. In contrast, the SbW/BAS has a skid controller implemented to overcome this problem, and when the steering rack gets stuck the vehicle can still be steered and brought to a safe stop. Thus the SbW/BAS is single fault tolerant.

To achieve fault tolerance, the SbW architecture implements redundancy in each component within the system: the computers are duplicated, and furthermore, each computer has two internal processors in lock-step; the rack actuation subsystems are duplicated as well; and the road wheel and steering wheel angle sensors are triply redundant. The SbW/BAS uses only redundancy for the road wheel and steering wheel sensors, which are the only elements common to SbW and BAS. Instead of having two computers with lock-step processor pairs, the SbW/BAS uses two single-processor computers, one for the SbW functionality, and the other for the BAS functionality (which is expected to already be present in the vehicle for ABS or ESP). The steering rack actuation subsystem is not duplicated either. As explained before, the fault-tolerance is achieved by using the functionality provided by the ABS/ESP system elements. The SbW/BAS implements an additional triple redundant set of gyro units, which are necessary for the skid steering control. Although ESP systems usually include a gyro unit, thus providing the necessary vehicle yaw rate measurement for the skid controller, it is not usually redundant.

So far, it seems that it is more advantageous to use the SbW/BAS: a better reliability estimate; no single points of failure; less redundancy, and therefore less complexity and cost; and use of already-implemented elements to achieve fault-tolerance. However, there are other important aspects, such as the vehicle performance, that must be analyzed to obtain a fair comparison between SbW and SbW/BAS. In the first place, it must be noted that BAS must only be be used as an ultimate safety mechanism to bring the vehicle to a safe stop when a failure disables the main SbW functionality. This means that although the SbW/BAS architecture introduced here is single fault-tolerant in the sense that any single failure will not cause a catastrophic loss-of-control system failure, once a failure in the SbW causes the BAS to kick in, the vehicle must be brought to a safe stop. This is not the case in the SbW fault-tolerant architecture —any first failure, except for a rack failure, does not degrade the system performance substantially, and therefore, the vehicle can still be driven for a period until repaired. Table 6.6 displays the number of possible system configurations after a single failure, together with the number of those that are required to prevent a loss of vehicle control. As can be seen, not a single failure at the first level for the SbW needs immediate action (the car is brought to a safe-stop, it cannot even be driven for a warning period). On the contrary, 8 out of 48 single failures in the SbW/BAS need immediate action, which represents 17% of the total number of first level of failures.

Table 6.6: Degraded performance comparison between the SbW and the SbW/BAS.

	# of operational first failure configurations	*# of configurations resulting in a degraded mode requiring the vehicle to stop*
SbW	42 out of 43	0
SbW/BAS	48 out of 48	8

6.5 Conclusions

This chapter has illustrated how the methodology developed in Chapter 3, and its supporting tool — InPRESTo (presented in Chapter 4) —, can be used to evaluate the pros and cons of two very different architectural approaches to achieving fault tolerance. Furthermore, from the case-study presented in this chapter, it is clear that the methodology presented in this thesis does not only address the integration of system performance and reliability, it also provides new insights (the designs comparison discussed in Section 6.4) that were not possible to obtain before.

Focusing on the case study, it has been shown how a dissimilar backup mechanism — brake-actuated steering — can be used successfully to reduce the amount of redundant elements in a fault-tolerant system and to remove single points of failure that cannot be remove by just adding redundancy (such as a stuck steering rack failure). Although the BAS technology looks promising to achieve fault tolerance in SbW, BAS enables bringing the vehicle to a safe stop after a failure has occurred, as opposed to allowing the vehicle to be driven for a long period of time after the failure. The BAS technology could be thought of as the ultimate safety net in a classical-redundancy-based SbW design (such as the one discussed in Section 6.2) to accommodate very unlikely events. For example, an additional failure between the first failure and the time the vehicle is brought into the repair shop, or the stuck steering rack failure — a failure that even in a conventional mechanical steering system will cause the system to fail catastrophically.

Notation Used in this Chapter

ABS:	Antilock braking system
BAS:	Brake-actuated-steering
BASC:	Brake-actuated-steering computer
ESP:	Electonic stability program
FDIR:	Failure detection, isolation, and recovery
FLCAS, FRCAS:	Front left and right caliper actuation subsystems
r_b :	Yaw rate
r_{b_r} :	Reference model yaw rate
RaAS:	SbW/BAS design rack actuation subsystem
RaAS_1, RaAS_2:	SbW design rack actuation subsystem
RLCAS, RRCAS:	Rear left and right caliper actuation subsystems
RWAS_1, RWAS_2, RWAS_3:	Road wheel angle sensors
SRa:	Steering rack
SbW:	Steer-by-wire
SbWC:	SbW/BAS design computers
SbWC_1, SbWC_2:	SbW design computers
SbWC_1-DC, SbWC_2-DC:	SbW design computers dual-comparator circuits
SbWC-FD:	SbW/BAS design computers failure detection circuits
SWAS_1, SWAS_2, SWAS_3:	Steering wheel angle sensors
t :	Time
t_c :	Configuration evaluation time
t_f :	Failure injection time
T :	System global evaluation time
U_f :	Behavioral modes random variable
V :	Forward velocity
YRG_1, YRG_2, YRG_3:	SbW/BAS yaw rate gyros
Z_1 :	Sideslip angle performance metric
Z_2 :	Yaw rate performance metric
Z_3 :	Heading angle performance metric
β :	Sideslip angle
β_r :	Reference model sideslip angle
δ :	Road wheel angle
δ_w :	Steering wheel angle
λ :	Failure rate
Ω_{Z_1} :	Set of requirements for performance metric Z_1
Ω_{Z_2} :	Set of requirements for performance metric Z_1
Ω_{Z_3} :	Set of requirements for performance metric Z_1
Ψ :	Heading angle
Ψ_r :	Reference model heading angle

Chapter 7

Towards Probabilistic-Informed Design

The purpose of this chapter is to discuss the use of the modeling and evaluation methodology introduced in this thesis as the enabler of a probabilistic-informed design framework for fault-tolerant systems. The discussion begins with a review of existing probabilistic-informed decision-making importance measures for system design. Sensitivity analysis techniques in the context of probabilistic-informed design will be also discussed. However, these techniques only scratch the surface of the problem. To understand the extent of the problem, the remainder of the chapter focuses on the issue of investigating how reliability is affected by the system dynamic behavior. In this context, we propose new importance measures that have the potential to be used as part of a probabilistic-informed design framework within the context of the new methodology. The Chapter concludes with a discussion of the monotonic behavior and the functional dependencies of the system reliability function, and the main open questions on this matter. Its purpose is to fully enable probabilistic-informed design within the framework of the modeling and evaluation methodology introduced in this thesis.

7.1 Introduction

In Chapter 5, we showed how the methodology presented in this thesis (and its supporting tool InPRESTo) can be used to guide the design of a fault-tolerant system to meet certain goals. Thus, if a given design does not meet these goals, it is necessary to extract certain information from the system evaluation analysis in order to improve its design. This information includes identifying the weakest design points (for example single points of failure), and the components and subsystems that are driving the system performance and reliability.

The design goals for the case-study presented in Chapter 5 were to achieve single fault-tolerance and a minimum reliability estimate. Thus, improving the design was a matter of identifying and eliminating any single point of failure and meeting the reliability goal. The first proposed design was already very close to meeting the reliability goal. Therefore we focused on eliminating the single points of failure. It turned out that once all the single points of failure were eliminated, the reliability also improved an order of magnitude with respect to the original goal. Thus, in this case, extracting the relevant information from the analysis to improve the system design was not a difficult task.

However, extracting this relevant information might not be an easy task in other situations. Taking the same flight control system, let's assume that even after removing all single points of failure, the resulting design does not meet the reliability goal. In this case, the analyst must find those parts of the system that are driving the reliability estimate and modify them with the hope of meeting the reliability goal as well.

The problem becomes even more acute when the design goals also include probabilistic performance goals, e.g., the average power consumption among all possible *non-failed* operational conditions. In this case, perhaps the architectural modification that could improve reliability might also harm the performance goals. Therefore, it is necessary to investigate appropriate techniques to extract the information relevant to improving a design to optimize performance and reliability.

The purpose of this chapter is to discuss some existing techniques for this purpose, and also discuss open questions and future research directions. Section 7.2 discusses several existing techniques, such as importance measures and sensitivity analysis, used to rank the impact of system parameters on reliability. Section 7.3 discusses the work we have done in probabilistic-informed design, defining a new importance measure in the context of the methodology proposed in this thesis. This section also discusses several open questions and future research directions within the context of probabilistic-informed design. Concluding remarks are presented in Section 7.4.

7.2 Existing Techniques for Probabilistic Informed Design

In the framework of the nuclear industry, without focusing on a particular reliability/risk modeling methodology, several importance measures have been proposed to quantify which plant normal operation disturbing events, and which component failure modes have the greatest impact on a particular risk metric in case of an accident. Focusing on

Markov reliability models, sensitivity analysis has also been proposed to rank the components that most influence the reliability estimate yielded by the Markov model. In this section, all these techniques will be discussed.

7.2.1 Existing Importance Measures

The definition of the common importance measures used in the nuclear industry has been adapted to the framework of this research. The reader is referred to [59] and [60] for their definition in the context of the nuclear industry. Thus, the most commonly used importance measures for ranking the importance of the m failure mode of component l are

Risk Achievement Worth:

$$RAW_l(m) = \frac{Q_l(m)^+}{Q}; \tag{7.1}$$

Risk Reduction Worth:

$$RRW_l(m) = \frac{Q}{Q_l(m)^-}; \tag{7.2}$$

Fussell-Vesely:

$$FV_l(m) = \frac{Q - Q_l(m)^-}{Q}; \tag{7.3}$$

where Q is the system unreliability (as defined in Appendix A), $Q_l(m)^+$ is the system unreliability when the transition to the m failure mode of component l occurs with probability one, and $Q_l(m)^-$ is the system unreliability when the transition to the m failure mode of component l occurs with probability zero.

These measures can only be used to compute the importance of single events; and except for Fusell-Vesely, they cannot be applied to parameters, e.g., failure rates, or coverage probabilities [61]. Furthermore, to compute any of these importance measures, it is necessary to compute $Q_l(m)^+$ and $Q_l(m)^-$ separately; i.e., it is not possible to obtain them directly from Q.

7.2.2 Sensitivity Analysis

Focusing on Markov reliability modeling, sensitivity analysis has been proposed to rank the influence of changes in the model parameters on the system reliability (or unreliability) [62], [63], [64]. For each parameter a_j, with $j = 1, 2, \ldots, K$, of the state-transition matrix A, it is necessary to solve

$$\frac{d}{dt}\begin{bmatrix} P(t) \\ \frac{\partial P(t)}{\partial a_1} \\ \frac{\partial P(t)}{\partial a_2} \\ \vdots \\ \frac{\partial P(t)}{\partial a_K} \end{bmatrix} = \begin{bmatrix} A & 0 & \ldots & 0 \\ \frac{\partial A}{\partial a_1} & A & \ldots & 0 \\ \frac{\partial A}{\partial a_2} & 0 & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial A}{\partial a_K} & 0 & \ldots & A \end{bmatrix}\begin{bmatrix} P(t) \\ \frac{\partial P(t)}{\partial a_1} \\ \frac{\partial P(t)}{\partial a_2} \\ \vdots \\ \frac{\partial P(t)}{\partial a_K} \end{bmatrix}$$
$$[P(0)' \; \frac{\partial P(t)'}{\partial a_1} \; \frac{\partial P(t)'}{\partial a_2} \; \ldots \; \frac{\partial P(t)'}{\partial a_K}] = [P_0' \; \bar{0}' \; \bar{0}' \ldots \bar{0}']$$
$$\frac{\partial Q(t)}{\partial a_j} = \sum_{i,k} \frac{\partial p_{2i+1,k}(t)}{\partial a_j}; \; \forall j = 1, 2, \ldots, K \tag{7.4}$$

where $P_0 = [1 \; 0 \; 0 \ldots 0]'$, $\bar{0} = [0 \; 0 \; 0 \ldots 0]'$, and $p_{2i,k}(t)$ are the components of vector $P(t)$ corresponding to the absorbing states.

The main advantage of using sensitivity analysis is that the ranking of the most influencing effects on the system unreliability is done at the parameter level. However, the computational cost of using sensitivity analysis is expensive, since the set of differential equations (7.4) is of size $N \times K$, where N is the number of states of the Markov model, and K is the number of parameters of the state-transition matrix.

It is possible to break the system of differential equations into K systems of size N and obtain an exact solution for the sensitivity. In this case, although the size of the problems to solve is of size N rather than of size $N \times K$, there is still need to solve K of them. To alleviate the computational burden, approximate solutions to (7.4) have been proposed [64].

Another approach to solving the sensitivity problem is to perturb each parameter a_i of the state-transition matrix A, solve the K resulting systems, and then rank the parameters according to the unreliability yielded by each a_i-perturbed system. With this approach it is again necessary to solve K sets of differential equations to obtain the sensitivity solution [49]. This technique is illustrated in the analysis of the case-study presented in Chapter 2.

The Differential Importance Measure

In the context of sensitivity analysis as a technique to rank the influence of each system parameter a_i, an importance- measure, called the Differential Importance Measure (DIM), is proposed in [60]. DIM is defined as:

$$DIM(a_i) = \frac{\frac{\partial Q}{\partial a_i} da_i}{\sum_j \frac{\partial Q}{\partial a_j} da_j}. \tag{7.5}$$

The computation of DIM exhibits the same shortcomings as those discussed for sensitivity analysis techniques. In fact, the solution proposed in [60] to compute DIM relies on the same techniques used in [49] to solve the sensitivity problem; i.e., to perturb each parameter a_i, and use the perturbed results together with the unperturbed solution to compute an approximation of (7.5).

7.3 Further Discussion and Open Research Questions

So far, we presented some of the existing techniques, such as importance measures and sensitivity analysis, to extract the information relevant to improving the design of a system for optimal performance and reliability. However, these techniques only scratch the surface of the problem. To understand the extent of the problem, we will focus on one aspect of integrating performance and reliability, which is how reliability is affected by system performance, and, therefore, how reliability is affected by the system dynamics of each configuration.

In the remainder of this section, we will discuss the work we have done in this area, proposing a new importance measure that has the potential to be used as part of a probabilistic-informed design framework within the context of the new methodology. Its main advantage is that it can be computed using only the nominal solution of the Markov model. The purpose of this measure is not to rank only the importance of component failures, but also the importance of the different system performance metrics. Several related issues which are still unsolved will be also discussed.

7.3.1 On the Functional Dependence of the Reliability Function

The system reliability is a function of the component failure rates and the coverage probabilities:

$$R = F(\bar{\lambda}, \bar{c}, t), \tag{7.6}$$

where $\bar{\lambda}$ is a vector containing the individual component failure rates, $\bar{c}$ is the vector of failure coverage probabilities, and t represents time.

On the Monotonicity of the Reliability Function with respect to time

The first question that must be answered is how the system reliability function behaves with respect to time. Usually reliability requirements must be met for a certain system evaluation time, e.g., the system life-time or the maintenance period. This makes sense as the reliability function is monotonically decreasing with time, and, therefore, by fixing a reliability requirement for some time T, we ensure that this requirement is met at any time $t < T$.

Lemma. Let $R = F(\bar{\lambda}, \bar{c}, t)$ be the reliability function of a system, then R is monotonically decreasing with t.

Proof. It is sufficient to prove that $\frac{dR}{dt}$ is non-positive for any value of t. The result follows from (3.21) and the following properties of the state-transition matrix A associated with the Markov reliability model (3.20): the diagonal elements are negative or zero, the off-diagonal elements are zero or positive, and A is a diagonally-dominant matrix.

Importance of the Component Failure Modes

The natural issue that arises now is to find the component failures that are driving the system reliability. As stated in Chapter 3, each system configuration $\{i, k\}$ is determined by a unique sequence of component failure mode transitions. Thus, some transitions between certain component failure modes will be driving the system reliability/unreliability. Therefore, identifying those transitions will allow us to modify the components with the largest possible impact on the system reliability.

Let $s_{l,m}^{i,k}$ be an indicator function that takes value 1 when the sequence of component failures that yields system configuration $\{i,k\}$ includes a transition to failure mode m of component l, and 0 otherwise. Then the importance measure for failure mode m of component l is defined by

$$I_{l,m}(t) = 1 - Q + \sum_{i,k} s_{l,m}^{i,k} p_{2i,k}(t), \tag{7.7}$$

where $p_{2i,k}(t)$ is the probability that, at a time $t > 0$, the system is in configuration $\{i,k\}$, and it is being declared as as *failed*, conditioned on $p_{1,0}(0) = 1$.

This importance measure can be interpreted as the contribution of the failure mode m of component j to the total system unreliability Q. The following cases are possible:

- $I_{l,m}(t) = 1$ means that the m failure mode of component l is present in each sequence of component failures that contributes to the system unreliability.
- If $I_{l,m}(t) < 1$ means that the m failure mode of component l is not present in at least one of the sequences of component failures that contributes to the system unreliability.
- If $I_{l,m}(t) = 1 - Q$ means that the m failure mode of component l is not present in any of the sequences of component failures that contribute to the system unreliability.

Therefore the largest value of the importance measure $I_{l,m}$ indicates the component l (and its associated failure mode m) that most influences the system reliability.

On the Monotonicity of the Reliability Function with respect to failure rates and coverage probabilities

Once the component failures that most affect the reliability are identified, the natural question that arises is how the system reliability function behaves with respect to changes in the failure rates of those components.

We could think that a way to improve the system reliability is by improving the component failure rates. However, the reliability function $R = F(\bar{\lambda}, \bar{c}, t)$ is not, in general, monotonic with respect to each entry λ_i of the vector of component failure rates $\bar{\lambda}$. Therefore, it might be the case that by making a component more reliable, the system reliability is being harmed.

As an example of this kind of behavior, let a system with three components C_1, C_2 and C_3 be arranged as follows:

- The system will be functional if C_1 or C_3 are operational, operating with component C_1 before any failures has occurred.
- When C_1 is in operation C_3 can not fail.
- C_1 needs component C_2 to work, if C_2 fails while C_1 is in operation, the system fails.
- If C_1 fails, C_3 is brought on-line and can perform its function without C_2, i.e, C_2 does not affect the function of the system when C_3 is operational.

Under the above assumptions, the state-transition matrix A of the resulting Markov reliability model is given by:

$$A = \begin{bmatrix} -\lambda_{C_1} - \lambda_{C_2} & 0 & 0 & 0 \\ \lambda_{C_1} & -\lambda_{C_3} & 0 & 0 \\ \lambda_{C_2} & 0 & 0 & 0 \\ 0 & \lambda_{C_3} & 0 & 0 \end{bmatrix}. \tag{7.8}$$

The Reliability function corresponding to this system is given by:

$$R(t) = e^{-(\lambda_{C_1}+\lambda_{C_2})t} + \frac{\lambda_{C_1}}{\lambda_{C_3} - (\lambda_{C_1} + \lambda_{C_2})}(e^{-(\lambda_{C_1}+\lambda_{C_2})t} - e^{-\lambda_{C_3}t}) \tag{7.9}$$

For $\lambda_{C_1} = 2 \cdot 10^{-6}$ failures/h, $\lambda_{C_2} = 3 \cdot 10^{-6}$ failures/h, $\lambda_{C_3} = 10^{-6}$ failures/h, and $t = 1000$h, $R_1(1000) = 0.99700648951303$. Now making $\lambda_{C_1} = 2.5 \cdot 10^{-6}$, $R_2(1000) = 0.99700698760028$. It is clear that $R_1(1000) < R_2(1000)$. Thus, by making C_1 less reliable, i.e., increasing λ_{C_1}, the system Reliability at $t = 1000$ increases.

Therefore, this is one of the research problems that must be further explored, i.e., under what conditions the system reliability function is monotonic with respect to component failure rates. In this context, we will pose a hypothesis that if proven right, could be a powerful technique for identifying weak points in the design. Although not discussed here, similar ideas apply to the coverage probabilities. Thus, we state the following hypothesis, which is something to be proven by further work.

Hypothesis. If a system reliability function $R = F(\bar{\lambda}, \bar{c}, t)$ is not monotonic decreasing with respect to each entry λ_i of the vector of component failure rates $\bar{\lambda}$, there is a weak point in the design that can be identified by finding the λ_i's for which $R = F(\bar{\lambda}, \bar{c}, t)$ does not exhibit monotonic behavior.

7.3.2 On the Functional Dependence of the Coverage Probability

The coverage probabilities are a function of the performance metrics and their requirements, and also of the system dynamics, therefore

$$\bar{c} = G(\bar{Z}, \Phi_{\bar{Z}}, \bar{f}(\cdot), \bar{g}(\cdot)), \tag{7.10}$$

where $\bar{Z}$ represents the system performance metrics, and $\Phi_{\bar{Z}}$ represent the associated requirements to those performance metrics. The functions $\bar{f}(\cdot)$ and $\bar{g}(\cdot)$ represents the dynamics of the different system configurations.

Performance Metrics Dependence

As stated before, the performance metrics $\bar{Z}$ are system-related properties that will quantify how well a system performs the function for which it was designed. Therefore, even if the coverage probability depends on them, there is not much we can do to improve the coverage probability, as they cannot be changed. However, by identifying the performance metric that most often violates its requirements, we are identifying the system properties that are driving the system reliability; and, therefore, the system design could be modified accordingly.

Let $s_{Z_j}^{i,k}$ be an indicator function that takes value 1 whenever the system configuration $\{i, k\}$ is declared as *failed* due to performance metric Z_j failing to meet its requirements. Then the importance measure for performance metric Z_j is defined by

$$I_{Z_j}(t) = 1 - Q + \sum_{i,k} s_{Z_j}^{i,k} p_{2i,k}(t), \tag{7.11}$$

where $p_{2i,k}(t)$ is the probability that, at a time $t > 0$, the system is in configuration $\{i, k\}$, and it is being declared as *failed*, conditioned on $p_{1,0}(0) = 1$.

The following cases are possible:

- $I_{Z_j}(t) = 1$ means that the Z_j performance metric is always violated whenever a system configuration is declared as *failed*.
- If $I_{Z_j}(t) < 1$ means that the Z_j performance metric is not violated in at least one of the system configuration declared as *failed*.
- If $I_{Z_j}(t) = 1 - Q$ means that the Z_j performance metric is never violated whenever a system configuration is declared as *failed*.

Therefore, the largest value of the importance measure I_{Z_j} indicates the performance metric that most influences the system reliability.

Requirements Dependence.

The requirements $\Omega_{\bar{Z}}$ associated with the performance metrics constrain the values these can take; therefore there is not much we can do about them either. However, it might be the case that the reliability is very sensitive to certain performance requirements. Thus, as with the performance metrics, if we could identify the requirements to which reliability is more sensitive, we would be able to identify the system properties that are driving the system reliability.

System Dynamics Dependence.

The most challenging issue related to the functional dependence of the coverage probabilities is related to the system dynamics $\bar{f}(\cdot), \bar{g}(\cdot)$ adopted by the system for every sequence of failures. Thus it is necessary to investigate the behavior of the coverage probability with respect to changes in $\bar{f}(\cdot), \bar{g}(\cdot)$.

7.4 Conclusions

In this chapter, we discussed the use of the methodology introduced in this thesis as the enabler of a probabilistic-informed design framework for fault-tolerant systems. Existing probabilistic-informed decision making importance-measures and sensitivity analysis techniques were reviewed. We also discussed the work we have done in the area of probabilistic-informed design, proposing a new importance measure that has the potential to be used within the framework of the new methodology. The discussion was completed with open questions and suggestions for future research directions in probabilistic-informed design. Although this chapter poses more questions than answers, the challenging issues brought up here are crucial for enabling probabilistic-informed design within the context of the methodology developed in this thesis.

Notation Used in this Chapter

a_j :	j parameter of the Markov model state-transition matrix
A :	Markov model state-transition matrix
$DIM(a_j)$:	Differential Importance Measure for parameter a_j
$\bar{f}(\cdot)$:	Vector of system configurations state evolution functions
$FV_l(m)$:	Fussell-Vesely importance measure for the m failure mode of component l
$\bar{g}(\cdot)$:	Vector of system configurations output functions
$I_{l,m}$:	Importance measure for the m failure mode of component l
I_{Z_j} :	Importance measure for performance metric Z_j
$p_{2i,k}(t)$:	Probability that the system configuration i, k is declared as failed at time t given that at $t = 0$ is in $\{1, 0\}$
$P(t)$:	System configurations probability vector
Q :	System unreliability
$Q_l^+(m)$:	System unreliability when the transition to the m failure mode of component l occurs with probability one
$Q_l^-(m)$:	System unreliability when the transition to the m failure mode of component l occurs with probability zero
R :	System reliability
$RAW_l(m)$:	Risk achievement worth importance measure for the m failure mode of component l
$RRW_l(m)$:	Risk reduction worth importance measure for the m failure mode of component l
$s_{l,m}^{i,k}$:	Indicator function for the operational status of component l when the system is in configuration $\{i, k\}$
$s_{Z_j}^{i,k}$:	Indicator function for the Z_j performance metric when the system failed after reaching configuration $\{i, k\}$
t :	Time
$\bar{Z}$:	Vector of system performance metrics
$\Phi_{\bar{Z}}$:	Set of sets of system performance requirements
$\bar{\lambda}$:	Vector of component failure rates
λ :	Failure rate
$\bar{c}$:	Vector of system failure coverage probabilities

Chapter 8

Concluding Remarks

This last chapter presents a summary of the thesis, highlighting its main contributions to the disciplines of system reliability analysis and system design. It concludes by discussing the observations the author has made over four years of research in the fields of reliability theory, dynamic system theory, and system design theory.

8.1 Thesis Summary and Highlights of Major Contributions

Chapter 1 stated the necessity of developing a new methodology for analyzing the reliability and performance of fault-tolerant systems for aircraft, space, tactical, and automotive applications. The main shortcoming of current reliability evaluation techniques was clearly identified in this chapter as: *the incompleteness of the system models used to analyze the system behavior in the presence of component failures.* Thus, the main goal of this thesis was set forth: *the development of a new methodology for evaluating fault-tolerant systems.* The main concern this methodology would address would is the gap between the system model behavior and the reliability model. That is, to minimize the subjectivity introduced in the analysis due to incompleteness of current methodologies. In order to put this thesis in the appropriate context, we reviewed classical reliability evaluation methodologies and tools, as well as related work in the field of Dynamic Probabilistic Risk Assessment (DPRA).

Chapter 2 illustrated the application of Markov models for system reliability evaluation, as Markov modeling is used to model the system stochastic behavior in the framework of the new methodology presented in Chapter 3. A case-study of an automotive power net architecture for automotive safety-critical applications was presented. The main contribution of this chapter was the development of a technique to systematically build a system-level Failure Modes and Effects Analysis (FMEA) that allows direct mapping to the Markov reliability model, thus making its construction easier. Sensitivity analysis (discussed in details in Chapter 7) was used to understand the influence of perturbations in the Markov model parameters. The results of the sensitivity analysis helped to determine the robustness of the reliability estimate with respect to parameter uncertainty, and also helped to improve the design in terms of reliability. Most of the work presented in this chapter was published in [40].

Chapter 3 introduced the mathematical foundations of the new modeling and evaluation methodology proposed in the thesis. This methodology introduced the use of behavioral models of the system dynamics, similar to the ones used by control engineers when designing the control system, but with additional features to model the dynamic behavior of the component in the presence of different failures. The performance evaluation is based on the system dynamic behavior when component failures within the system occur. The proposed methodology allows one to assess not only system reliability, but other important dynamic performance metrics that might be relevant in the design of a fault-tolerant system. The system stochastic behavior due to component failures is modeled by using Markov models. In this framework, a rigorous approach for computing failure coverage probabilities was presented; providing analytical solutions for LTI systems, and a Monte-Carlo based methodology for non-linear systems. Several examples were developed to illustrate the concepts introduced in this chapter.

Chapter 4 presented a MATLAB/SIMULINK® based tool that was developed in order to make the application of the new methodology feasible. This tool automates the evaluation process of a system defined by the analyst using the SIMULINK® environment. The tool is called InPRESTo, an acronym for Integrated Performance and Reliability Evaluation SIMULINK® Toolbox. The functionality, structure, and main futures of InPRESTo were explained in detail in this chapter. This chapter is also a succinct user manual, which includes explanations on how to define the necessary inputs to perform a system analysis, and how to visualize the analysis results. A testament to InPRESTo's importance is the fact that it is being used by the Systems Engineering and Evaluation Division at the Charles Stark Draper Laboratory for the evaluation of space and tactical systems.

Chapter 5 explained how the new performance and reliability modeling and evaluation methodology introduced in Chapter 3 and its supporting SIMULINK® toolbox —InPRESTo— can be used to: identify weak points in the system design; guide the design by pointing to possible solutions to eliminate the uncovered weak points; compare different architecture alternatives from different perspectives; and test different failure detection, isolation, and reconfiguration (FDIR) techniques. To demonstrate this, a case-study of a fault-tolerant architecture for a fighter aircraft lateral-directional flight control system was presented and analyzed in detail. This case-study proved the scalability of the tool to analyze large systems.

Chapter 6 illustrated how the new methodology (and its supporting tool) can be used to compare conceptually very different architectural approaches to achieve fault-tolerance. For this purpose, two different designs to achieve fault-tolerance in a steer-by-wire (SbW) system were presented, analyzed, and compared in terms of performance, reliability, and fault-tolerance. The first design was based on redundancy of components, and the introduction of failure detection, isolation, and reconfiguration mechanisms. In the second design, a dissimilar backup mechanism called brake-actuated steering (BAS), was used to achieve fault-tolerance. This chapter complements Chapter 5 by showing how the new modeling and evaluation methodology (and its supporting SIMULINK® toolbox) can be used to compare very different architectural approaches to fault-tolerance design.

Chapter 7 discussed the use of the modeling and evaluation methodology introduced in this thesis as the enabler of a probabilistic-informed design framework for fault-tolerant systems. This chapter posed more questions than gave answers. However, the challenging issues brought up here are crucial for developing a probabilistic-informed design framework based on the methodology presented in this thesis. To understand the extent of the problem, we focused the discussion on the issue of investigating how reliability is affected by system performance, and therefore, how reliability is affected by system dynamics. In this context, we proposed new importance measures that have the potential to be used as part of a probabilistic-informed design framework within the context of the new methodology. The chapter concluded with a discussion of functional dependencies of the system reliability function and the main open questions in this matter.

8.2 Conclusions

The design of an effective fault-tolerant system requires a thorough and comprehensive analysis to fully understand and quantify potential failures and assess the effectiveness of failure detection, isolation, and reconfiguration (FDIR) mechanisms. The literature abounds with well-established techniques that support the system reliability evaluation during its design phase.

Unfortunately, all these techniques have a common shortcoming: the difficulty of generating the reliability model in a systematic and objective way from the functional descriptions of the system and the component failures, i.e, there is a gap between the system functional model description and the system reliability model. For some conventional systems, this shortcoming can be overcome, as it might not be very difficult to generate an objective and complete reliability model from the system functional description, and from expert judgment to assess the impact of component failures on the system functionality. However, this is not the case for *large complex* systems or embedded *software-intensive* systems, typical of fault-tolerant systems. For the analysis of these systems, all these techniques can yield ambiguous and/or incomplete results, as it is virtually impossible to fully understand the system behavior in the presence of failures —and thus, to determine its reliability— by only using a qualitative description of the system's functionality and expert judgment.

We postulated that an excellent way to fill the gap between the system functional and reliability models would be to use a behavioral model of the system dynamics, similar to the ones used by control engineers when designing the control system, but with additional features to model different failure behaviors of the components. The reliability model is thus based on the system dynamic behavior when component failures within the system occur. In this context, Markov models are the perfect formalism to model the stochastic behavior due to component failures of fault-tolerant systems.

One of the most important challenges when formulating the mathematical foundations of the new methodology was to link the system dynamic behavioral model and the system reliability model. The way we approached and solved this problem was by introducing failure coverage probabilities in the stochastic model. We defined these probabilities through the set of possible initial conditions at the time of failure, and a subset of possible initial conditions that result in trajectories which are contained in some predefined set of "acceptable" states. This resulted in the integration of system dynamic performance and reliability, which enabled a completely new way of analyzing fault-tolerant systems, and provided new insights from the analysis that were not possible to obtain before. Examples of these are the way the testing of FDIR effectiveness can now be incorporated into the system performance and reliability analysis, and the multiple ways conceptually very different architectural approaches to the same application can now be compared.

This thesis is a culmination of four years of research and development in the area of fault-tolerant systems evaluation and design. The main contribution of this thesis is bringing control theory, reliability theory, and system design a step closer to a unified systems science discipline. In addition, this thesis bridges the gap between the system model behavior and the reliability model, thus integrating dynamic performance and reliability evaluation into one framework. The thesis covers many aspects of fault-tolerant systems design, yet one of its major contributions is to provide a rigorous mathematical framework that will enable future researchers to develop a fully integrated probabilistic-informed design framework.

Appendix A

Basic Concepts and Definitions

A.1 Basic Definitions

The purpose of this appendix section is to define some basic concepts that appear in this thesis. This section is by no means an extensive collection of reliability concepts and definitions. The reader is referred to [6, 65] for other reliability-related definitions.

Failure Rate. Let $T_f^{i,j}$ be a random variable that represents the time-to-failure-mode-j for component i. The failure rate for failure mode j of component i is defined as

$$\lambda_i^j(t) = \lim_{\Delta t \to 0} \frac{P(t < T_f^{i,j} \leq t + \Delta t \mid T_f^{i,j} > t)}{\Delta t}. \tag{A.1}$$

It is frequently assumed that the failure rate follows the general shape of a bathtub curve [6]. Most reliability analyses consider only the flat part of the curve, therefore assuming a constant failure rate. This is not very realistic in applications where the system lifetime is large enough to make component wear-out effects important. It is possible to model wear-out effects using time-dependent failure rates.

Repair Rate. Let $T_r^{i,j}$ be a random variable that represents the time-to-repair for failure mode j of component i. The repair rate for failure mode j of component i is defined as

$$\mu_i^j(t) = \lim_{\Delta t \to 0} \frac{P(t < T_r^{i,j} \leq t + \Delta t \mid T_r^{i,j} > t)}{\Delta t}. \tag{A.2}$$

Failure Coverage Probability. A component is said to *failed covered* when a failure event that occurs within the component, affecting its performance, can be detected, isolated, and the system reconfigured to compensate for that failure [21]. Thus, the failure coverage probability c is defined as the probability that given a failure event has occurred, it can be detected, isolated, and the system reconfigured to compensate for the failure (in order to keep delivering its function) before an unrecoverable transient occurs.

Point-Wise Reliability $R(t)$. The point-wise system reliability $R(t)$ is defined as the probability that the system is in a non-failed configuration at time t and can be computed as

$$R(t)dt = S'P^{\overline{FC}}(t), \tag{A.3}$$

where $S = [1\ 1\ 1\ \ldots\ 1]'$ is a vector of ones and size m (the number of transient states), and $P^{\overline{FC}}$ corresponds to the transient states probabilities.

Point-Wise Unreliability $Q(t)$. The point-Wise Unreliability $Q(t)$ is defined as the probability that the system is in a failed configuration at time t and can be computed by

$$Q(t) = 1 - R(t). \tag{A.4}$$

Dependability Rate $\bar{\lambda}(T)$. The dependability rate $\bar{\lambda}(T)$ is defined as the ratio of the point-wise unreliability $Q(T)$ to the global system evaluation time T:

$$\bar{\lambda} = \frac{Q(T)}{T}, \tag{A.5}$$

where T is the system evaluation time, e.g., system lifetime, or mission time. It has been adopted from the Federal Aviation Administration (FAA) regulations [66]. It can be thought of as an average failure rate for the system at time T.

A.2 Continuous-Time Markov Chains

Let the random process $X = \{X(t) : t \geq 0\}$ be a family of random variables which take values in some countable set $C = \{1, 2, \ldots N\}$, called the state space and indexed by $[0, \infty[$. The process X is called a continuous time Markov chain if it satisfies the Markov condition, stated by

$$P(X(t_n) = j \mid X(t_1) = i_1, ..., X(t_{n-1}) = i_{n-1}) = P(X(t_n) = j \mid X(t_{n-1}) = i_{n-1})$$
$$for\ all\ n \geq 1\ and\ all\ j, i_1, ..., i_{n-1} \in S\ and\ any\ sequence\ t_1 < \ldots < t_n. \tag{A.6}$$

Denoting t_{n-1} by t and t_n by $t + \Delta t$, with $\Delta t \to 0$, it is possible to write

$$P(X(t + \Delta t) = j \mid X(t) = i) = \lambda_{ij}(t)\Delta t, \tag{A.7}$$

where $\lambda_{ij}(t)$ is known as the transition rate from state i to state j.

By using the Chapman-Kolmogorov equations, we obtain the set of differential equations

$$\begin{aligned} \tfrac{dp_k(t)}{dt} &= -\lambda_k(t)p_k(t) + \Sigma_{j\neq k}\lambda_{jk}(t)p_j(t) \; with \; j,k = \{1,2,\dots N\} \\ &[p_1(0)\; p_2(0) \dots p_N(0)]' = p_0, \end{aligned} \tag{A.8}$$

where $p_k(t)$ is the probability of being in the state $k \in C$ at time t, given the states initial probability density function p_0, $\lambda_{jk}(t)$ is defined by

$$\lambda_{jk}(t) = \lim_{\Delta t \to 0} \frac{P(X(t+\Delta t) = k \mid X(t) = j)}{\Delta t}, \tag{A.9}$$

and $\lambda_k(t)$ is the transition rate out of state k, and it is defined as

$$\lim_{\Delta t \to 0} P(X(t+\Delta t) = k \mid X(t) = k) = \lim_{\Delta t \to 0}(1 - \lambda_k(t)\Delta t). \tag{A.10}$$

Equation (A.8) can be written in matrix form as

$$\begin{aligned} \tfrac{dP(t)}{dt} &= AP(t) \\ P(0) &= p_0, \end{aligned} \tag{A.11}$$

where $P(t) = [p_1(t)\; p_2(t) \dots p_N(t)]'$ and A is called the generator of the chain (or the state-transition matrix), and it is built using the transition rates $\lambda_k(t)$ and $\lambda_{jk}(t)$.

Notation Used in this Chapter

A :	Markov chain state-transition matrix
$p_k(t)$:	Probability of being in state k at time t, given the states initial probability density function
$Q(t)$:	Point-wise system unreliability
$R(t)$:	Point-wise system reliability
t :	Time
T :	System global evaluation time
$T_f^{i,j}$:	Time-to-failure-mode-j for component i
$T_r^{i,j}$:	Time-to-repair for failure mode j for component i
λ_k :	Transition rate out of state k
λ_{jk} :	Transition rate from state j to state k
λ_i^j :	Failure rate for failure mode j of component i
$\bar{\lambda}(T)$:	Dependability rate
μ_i^j :	Repair rate for failure mode j of component i
$P^{FC}(t)$:	System non-failed configurations probability vector

Appendix B

InPRESTo Subroutines

Subroutines Description

Inpresto InPRESTo main script. All the subroutines, except *Inpresto_plot* are called from this script.

failure_information searches in the Simulink model for the failure information of each block.

init_evaluation will evaluate the initial behavior of the system when no failures has occurred.

metrics_assessment will check whether the requirement specifications are met.

sequences_first_level will create the failure sequences to inject at the first level of failure.

evaluation This function will evaluate each sequence of failures at the k level.

sequences_separation This function will return the failed and non-failed sequences at each level of failure

sequences_k_level This function will create the failure sequences to inject at the k level of failure.

matrix helps establishing the state-transition matrix.

truncate_matrix truncates the state transition matrix at truncation level k-level after the system was evaluated at k-level-1

markov_model_solver Script to solve the system of differential equations associated to the Markov model.

metrics_models_evaluation This function will analyze the results of the evaluation.

sensitivity_analysis Similar to the previous one, when sensitivity is carried out.

results_analysis This function will collect the results in an excel spreadsheet.

Inpresto_plot This function will plot some of the the results of the evaluation.

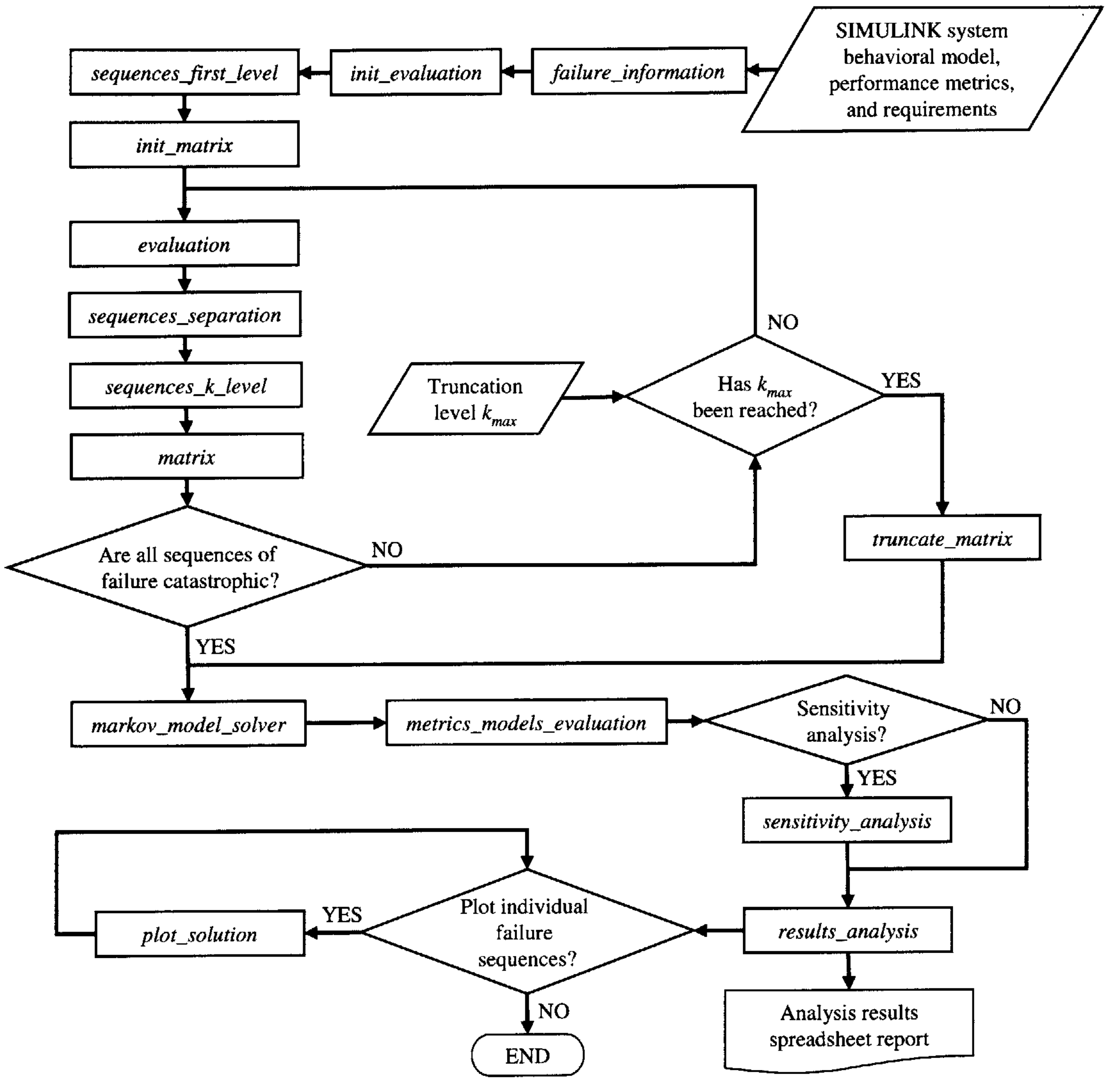


Figure B.1: Subroutines flow diagram.

InPRESTo's Main Script

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                       Alejandro D. Dominguez-Garcia
%                              aledan@MIT.EDU
%           Laboratory for Electromagnetic and Electronic Systems
%                  Massachusetts Institute of Technology
%
%                        Created:  October  20, 2005
%                        Modified: April 19, 2006
%
%
% Main script of the Performance Evaluator. Type Evaluator to invoke it.
% Make sure the Simulink model to be analyzed is opened.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function Inpresto
clear all
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% INIT VARIABLES %%%%%%%%%%%%%%%%%%%%%%%%%
% Init variables and global variables declaration
states_per_failure_level=1;             % Only one possible state when
                                        % no components have failed
system_states.metrics=[];               % Metrics to assess the system
                                        % performance
system_states.state_of_the_system=[];   % Vector of zeros and ones
                                        % representing if the system
                                        % exhibitis some performance
system_states.previous_state=[];        % Previous system state
system_states.xfinal=[];                % Final states that will be used
                                        % as initial conditions for the
                                        % next sequence
system_states.current_failure_mode=[];  % Current component failure mode
global stop_time;
global injection;
global time;                            % Row vector with the
                                        % information regarding the time
                                        % at which a failure occurs
global failure_modes;                   % Row vector with the
                                        % information regarding each
                                        % component failure mode
global sym_matrix;                      % It will be used by several
                                        % subroutines
```

```
global P;                               % Markov model state-transition
                                        % matrix
global sensitivity                      % Structure to store the
                                        % sensitivity matrices
global time_arr;
global level_number;                    % Number of failure levels
global sens_analysis;
global sensitivity_factor;
global truncation;
time_arr=[];

%%%%%%%%%%%%%%%%%%%%%%%%%% ANALYSIS PARAMETERS INPUT %%%%%%%%%%%%%%%%%
model_name=input('Enter model name: ' , 's');
evaluation_time=input('Global evaluation time T (h) : ');
truncation=input('Truncate evaluation (y/n): ', 's');
if truncation=='y'
    t_level=input('Truncation level k_max : ');
else
    t_level=1e6;
end
stop_time=input('Configuration evaluation time t_c (s): ');
injection=input('Failure injection time (between 0 and 1) : ');
sens_analysis=input('Sensitivity analysis (y/n): ', 's');
if sens_analysis=='y'
   sensitivity_factor=...
       input('Failure rates factor sensitivity analysis: ');
else
   sensitivity_factor=0;
end
filename=input('xls file name for report: ' , 's');
tic
%%%%%%%%%%%%%%%%%%%%%%%%%%%% MODEL PREPARATION %%%%%%%%%%%%%%%%%%%%%%%%%
%tic
% Obtain the failure information included in the model.
[components_information,failure_info,num_of_components,...
    failure_modes_vector,metrics_information,lambda,lambda_sym]=...
    failure_information(model_name);
%disp(['Time taken to gather data and prepare model: ' num2str(toc)]);
%%%%%%%%%%%%%%%%SYSTEM EVALUATION AND MARKOV MODEL CONSTRUCTION %%%%%%%
%tic
% Initial performance of the system
signals=[0 zeros(1,num_of_components) 1 0 0];
```

```
evaluated_states_counter=0;
[system_states,evaluated_states_counter,state_of_the_system]=...
    init_evaluation(model_name,signals,evaluated_states_counter,...
    num_of_components,system_states);
% First set of signals for failure modes injection
[signals,state_counter,failure_level_counter]=...
    sequences_first_level(failure_modes_vector);
% First column of the state-transition matrix from the original set of
% signals.
[P sensitivity]=init_matrix(signals,failure_info,lambda,lambda_sym);
while ((sum(state_of_the_system)~=0) & (failure_level_counter+1)<=(t_level))
     states_per_failure_level=[states_per_failure_level ; size(signals,1)];
     [system_states,evaluated_states_counter,state_of_the_system]=...
         evaluation(model_name,signals,evaluated_states_counter,...
         num_of_components,system_states);
     [non_failure_sequences,failure_sequences]=...
         sequences_separation(state_of_the_system,signals);
     if  (failure_level_counter+1)<=(t_level)
         [signals,state_counter,failure_level_counter]=...
         sequences_k_level(non_failure_sequences,...
         failure_modes_vector,state_counter,failure_level_counter);
         [P sensitivity]=matrix(signals,failure_info,lambda,lambda_sym);
     end
     evaluated_states_counter
end
if  ((failure_level_counter)==(t_level) & (sum(state_of_the_system)~=0))
     [P,sensitivity]=truncate_matrix(evaluated_states_counter);
     system_states(evaluated_states_counter+1).metrics=0;
     system_states(evaluated_states_counter+1).state_of_the_system=0;
     system_states(evaluated_states_counter+1).previous_state=[];
     system_states(evaluated_states_counter+1).xfinal=[];
     system_states(evaluated_states_counter+1).current_failure_mode=[];
     states_per_failure_level=[states_per_failure_level ; 1];
end
disp(['Time taken to evaluate system and build Markov model: ' num2str(toc)]);
disp(['Size of the Markov model transition matrix: ' num2str(size(P,1))]);
%tic
save matrix_model P
save lambda_matrix lambda
disp(['Time taken to save the Markov model: ' num2str(toc)]);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% MARKOV MODEL SOLVER %%%%%%%%%%%%%%%%%%%%
tic
```

```
[t,p,p_states] = markov_model_solver(evaluation_time,lambda);
disp(['Time taken to solve the Markov model: ' num2str(toc)]);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%% METRICS EVALUATION %%%%%%%%%%%%%%%%%%%%%%%
tic
[non_functional_metrics_measures,...
    non_functional_metrics_measures_levels,...
    probabilistic_performance_measures]=...
    metrics_models_evaluation(system_states,states_per_failure_level,t,p);
disp(['Time taken to evaluate metrics: ' num2str(toc)]);
%%%%%%%%%%%%%%%%%%%%%%%%%%% SENSITIVITY ANALYSIS %%%%%%%%%%%%%%%%%%%%%%%
if sens_analysis=='y'
    [sensitivity_measures sensitivity_driver driver]=...
        sensitivity_analysis(system_states,t,p_states,...
        components_information);
else
    sensitivity_measures=[];
    sensitivity_driver=[];
    driver=[];
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%% ANALYSIS RESULTS %%%%%%%%%%%%%%%%%%%%%%%%%
tic
results_analysis(components_information,system_states,...
    states_per_failure_level,t,p,non_functional_metrics_measures,...
    non_functional_metrics_measures_levels,...
    probabilistic_performance_measures,sensitivity_measures,...
    driver,filename)
disp(['Time taken to analyze results: ' num2str(toc)]);
disp(['Time taken to evaluate the system: ' num2str(toc)]);
%%%%%%%%%%%%%%%%%%%%%%%%%% SAVE IMPORTANT RESULTS %%%%%%%%%%%%%%%%%%%%%%%
save model_results model_name evaluation_time stop_time ...
    injection filename lambda system_states t p num_of_components
```

failure_information

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                      Alejandro D. Dominguez-Garcia
%                             aledan@MIT.EDU
%           Laboratory for Electromagnetic and Electronic Systems
%                  Massachusetts Institute of Technology
%
%                       Created:  October 21, 2005
%                       Modified: February 08, 2006
% This script searches in the Simulink model for the failure information
% of each block.
% INPUTS:
% -model_name: name of the Matlab Simulink model we want to search for
%  failure
%  information.
% OUTPUTS:
% -components_information: an array variable with the following fields:
%   -name: name of the component with associated failure information
%   -failure_info: an array variable with the the following fields:
%      -failure_modes: component failure modes
%      -failure_rates: failure rates associated to each failure mode
%       listed in the failure_modes field.
% -failure_info: components_information.failure_info. This is necessary
%  to pass to the matrix subroutine.
% -num_of_components: variable that returns the number of components in
%  the system
% -failure_modes_vector: a vector of dimension the number of components
%  with associated failure information. Each elements has an integer
%  thatrepresents the number of failure modes associated to the
%  component.Each element has an integer that represent the number of
%  failure modes considered for each component.
% -lambda: a matrix with as many rows as components and columns equal to
% the number of failure modes of each component.
% COMMENTS:
% This function is able to reado both symbolic and numerical data from
% the simulink failure blocks information. It is possible to introduce
% time-dependent failure rates or constants to carry out sensitivity
% analysis later on.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
function [components_information,failure_info,num_of_components,...
    failure_modes_vector,metrics_information,lambda,lambda_sym]=...
failure_information(model_name)
%Init
components_information.name=[];
components_information.failure_info=[];
failure_info.failure_modes=[];
failure_info.failure_rates=[];
failure_info.failure_rates_sym=[];
failure_info.lambda=[];
failure_modes_vector=[];
metrics_information.name=[];
syms lambda_sym;
syms t;
%%%%%%%%%%%%%%%%%%%%%% COMPONENTS INFORMATION %%%%%%%%%%%%%%%%%%%%%%%%%%%
% It breaks the links with all the libraries so it is possible to modify
% the library blocks
save_system(model_name,model_name, 'BreakLinks');
% Look for the components with associated failure information.
failure_blocks=...
find_system...
(model_name,'LookUnderMasks','all','Regexp', 'on','Name','Failures');
failure_modes_tag=...
    find_system...
    (model_name,...
    'LookUnderMasks','all','Regexp', 'on','Name','Failure_mode');
for i=1:size(failure_blocks,1)
    % We label each block with the right index
    set_param...
        (char(failure_modes_tag(i)), 'Time', ['time(' num2str(i) ')'],...
        'After', ['failure_modes(' num2str(i) ')']);
    % look for the last two field of the string
    failures_tag=regexp(char(failure_blocks(i)),'(\w*)/Failures','match');
    % look for the name of the component with failure information
    component_name=regexp(char(failures_tag),'(\w*)/','match');
    components_information.name=...
        [components_information.name ; component_name];
    % Look for all the variables withing the mask
    failure_variables=get_param(char(char(failure_blocks(i))),'MaskNames');
    % look for the different values of the mask
    failure_rates=get_param(char(failure_blocks(i)),'MaskValues');
    for j=1:size(failure_variables,1)
```

```
        % it only takes the information regarding failure modes
        if size(char(failure_variables(j)),2)>1
            failure_info.failure_modes=...
                [failure_info.failure_modes ; failure_variables(j)];
            failure_info.failure_rates=...
                [failure_info.failure_rates; sym(char(failure_rates(j)))];
            lambda(i,j)=eval(sym(char(failure_rates(j))));
            lambda_sym(i,j)=['lambda(' num2str(i) ',' num2str(j) ')'];
            failure_rates_sym=['lambda(' num2str(i) ',' num2str(j) ')'];
            failure_info.failure_rates_sym=...
                [failure_info.failure_rates_sym ; sym(failure_rates_sym)];
        end
    end
    components_information.failure_info=...
        [components_information.failure_info ; failure_info];
    % Empty the auxiliary variables for the next iteration
    failure_info.failure_modes=[];
    % Empty the auxiliary variables for the next iteration
    failure_info.failure_rates=[];
    % Empty the auxiliary variables for the next iteration
    failure_info.failure_rates_sym=[];
end
% We will be passing only the failure_infor portion to the matrix
% subroutine
failure_info =components_information.failure_info;
num_of_components=size(failure_info,1);
% Failure modes vector. It is a row vector with as many elements as
% system components. Each element has an integer that represent the number of
% failure modes considered for each component.
for i=1:size(failure_info,1)
    failure_modes_i=size(failure_info(i).failure_modes,1);
    failure_modes_vector=[failure_modes_vector failure_modes_i];
end
%%%%%%%%%%%%%%%%%%%% METRICS INFORMATION %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
metrics_blocks=...
    find_system(model_name,'LookUnderMasks','all','Regexp', 'on',...
    'Name','Metrics');
for i=1:size(metrics_blocks,1)
    % look for the name of the metrics
    metrics_information(i).name=get_param(metrics_blocks(i),'VariableName');
end
```

init_evaluation

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                      Alejandro D. Dominguez-Garcia
%                             aledan@MIT.EDU
%           Laboratory for Electromagnetic and Electronic Systems
%                  Massachusetts Institute of Technology
%                        Created:  January 24, 2005
%                        Modified: April 19, 2006
% This function will evaluate the initial behavior of the system when no
% failures has occured.
% INPUTS:
% -model_name: name of the Matlab Simulink model we want to analyze.
% -signals: matrix with failure sequences information. Each row
% represents a sequence of failures. The first column is an index
% associated with the
%  element that has failed. Columns 2 to num_of_components+1 represents
%  the state of each component. If the element signals(2,3) has a value
%  of k=4, means that element 1 in the third possible failure sequence
%  failed in failure mode with index 4. If k=0, it means normal
%  operation of the component.
% -evaluated_states_counter: index representing the highest state
%  already evaluated by the algorithm.
% -num_of_components: variable that returns the number of components in
%  the system.
% -system_states: an array with the following fields:
%   -metrics: metrics to assess the system performance.
%   -state_of_the_system: vector of zeros and ones representing if the
%    system exhibitis some performance.
%   -previous_state: previous system state.
%   -current_failure_mode: current component failure mode. It has two
%    sub-fields:
%     -component: index to represent which component has failed
%     -component_failure_mode: index to represent the corresponding
%      failure mode.
% OUTPUTS:
% -system_states: see above.
% -evaluated_states_counter: see above.
% -state_of_the_system: column vector with zeros and ones. A one
%  represent the system exhibiting some performance.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
function [system_states,evaluated_states_counter,state_of_the_system]=...
    init_evaluation(model_name,signals,evaluated_states_counter,...
    num_of_components,system_states)
global stop_time;
global time;
global failure_modes;
global level_number;
level_number=0;
state_of_the_system=[];
y=[];
time=zeros(1,num_of_components);
failure_modes=signals(2:num_of_components+1);
options=[];
% Final conditions for this sequence of failures that will be passed to
% the next sequence of failures
[time_arr,x,yout]=sim(model_name,[0 stop_time],options);
xfinal=x(size(x,1),:);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% This is just to get an steady-state picture of the system response
xinitial=xfinal;
[time_arr,x,yout]=sim(model_name,[0 stop_time],options);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
[state_of_the_system_i,metrics]=...
    metrics_assessment(Performance_metrics,upper_bound,lower_bound);
state_of_the_system= [state_of_the_system ; state_of_the_system_i];
system_states(evaluated_states_counter + 1).metrics=metrics';
system_states(evaluated_states_counter + 1).state_of_the_system=...
    state_of_the_system_i;
system_states(evaluated_states_counter + 1).previous_state=...
    signals(size(signals,2) - 1);
system_states(evaluated_states_counter + 1).xfinal=xfinal;
current_failure_mode.component=signals(1,1);
current_failure_mode.component_failure_mode=signals(signals(1)+1);
system_states(evaluated_states_counter + 1).current_failure_mode=...
    current_failure_mode;
evaluated_states_counter=evaluated_states_counter+size(signals,1);
%%%%%%%%%%%%%%%%%%%%%%%% AUXILIARY FUNCTION %%%%%%%%%%%%%%%%%%%%%%%%
function [state_of_the_system_i,metrics]=...
    metrics_assessment(Performance_metrics,upper_bound,lower_bound)
metrics=[];
% Metrics definition for each of the performance metrics.
for i=1:size(Performance_metrics,2)
```

```
    metrics(i)=norm(Performance_metrics(:,i),inf);
end
%%%%%%%%%%%%%%%%%%%%%% AUXILIARY FUNCTIONS %%%%%%%%%%%%%%%%%%%%%%%%%%%%
if max((metrics(:)>upper_bound(:) | metrics(:)<lower_bound(:)))==1;
   state_of_the_system_i=0;
else
   state_of_the_system_i=1;
end
end
```

sequences_first_level

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                       Alejandro D. Dominguez-Garcia
%                              aledan@MIT.EDU
%            Laboratory for Electromagnetic and Electronic Systems
%                   Massachusetts Institute of Technology
%
%                        Created:  October 20, 2005
%                        Modified: October 20, 2006
%
%
% This function will create the failure sequences to inject at the first
% level of failure.
%
% INPUTS:
% -failure_modes_vector: a vector of dimension the number of components
%  with associated failure information. Each elements has an integer
%  that represents the number of failure modes associated to the
%  component.
% OUTPUTS:
% -signals: a matrix with as many rows as the sum of failure modes
%  of all the components and many columns as components plus 4. The
%  first column is an index to represent the failed component. The last
%  column represents the component that failed just before. The one
%  before the  last and the one before the one before the last is the
%  state number.
% -state_counter: it counts the number of states.
% -failure_level: it counts the levels of failure.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


function [signals,state_counter,failure_level_counter]=...
    sequences_first_level(failure_modes_vector)

% Init
state_counter=1;
signals=sparse([]);
failure_level_counter=1;
```

```
% Create signals
for i=1:size(failure_modes_vector,2)
    failure_modes_i=failure_modes_vector(i);
    signals_aux=...
        [i*ones(failure_modes_i,1) ...
        zeros(failure_modes_i,size(failure_modes_vector,2))...
        state_counter*ones(failure_modes_i,1)+[1:1:failure_modes_i]' ...
        ones(failure_modes_i,1) zeros(failure_modes_i,1)];
    signals_aux(:,i+1)=[1:1:failure_modes_i]';
    signals=[signals; signals_aux];
    state_counter=state_counter+failure_modes_i;
end
```

init_matrix

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                       Alejandro D. Dominguez-Garcia
%                              aledan@MIT.EDU
%           Laboratory for Electromagnetic and Electronic Systems
%                  Massachusetts Institute of Technology
%
%                        Created:  October 24, 2005
%                        Modified: April 19, 2006
%
%
% This script helps establishing the state-transition matrix.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [P sensitivity]=...
init_matrix(signals,failure_info,lambda,lambda_sym)

global sym_matrix;
global sensitivity;
for i=1:size(signals,1)
    if sym_matrix=='n'
        lambda_i=lambda(signals(i,1),signals(i,signals(i,1)+1));
    elseif sym_matrix=='y'
        lambda_i=lambda_sym(signals(i,1),signals(i,signals(i,1)+1));
    end
    % State transition matrix
    P(signals(i,size(signals,2)-2),signals(i,size(signals,2)-1))=...
        lambda_i;
    aux=-lambda_i;
    P(signals(i,size(signals,2)-1),signals(i,size(signals,2)-1))=...
        P(signals(i,size(signals,2)-1),signals(i,size(signals,2)-1))+aux;
    % Sensitivity matrices
    k=signals(i,1);
    l=signals(i,signals(i,1)+1);
    m=signals(i,size(signals,2)-2);
    n=signals(i,size(signals,2)-1);
    sensitivity(k,l).matrix(m,n)=sparse(1);
    sensitivity(k,l).matrix(n,n)=sparse(-1);
end
P=sparse(P);
```

evaluation

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                       Alejandro D. Dominguez-Garcia
%                              aledan@MIT.EDU
%           Laboratory for Electromagnetic and Electronic Systems
%                  Massachusetts Institute of Technology
%
%                        Created:  October 24, 2005
%                        Modified: February 08, 2006
%% This function will evaluate each sequence of failures at the k level
%% INPUTS:
% -model_name: name of the Matlab Simulink model we want to analyze.
% -signals: matrix with failure sequences information. Each row
%  represents a sequence of failures. The first column is an index
%  associated with the element that has failed. Columns 2 to
%  num_of_components+1 represents the state of each component.
%  If the element signals(2,3) has a value of k=4, means that element 1
%  in the third possible failure sequence failedin failure mode with
%  index 4. If k=0, it means normal operation of the component.
% -evaluated_states_counter: index representing the highest state
%  already evaluated by
%  the algorithm.
% -num_of_components: variable that returns the number of components in
%  the system.
% -system_states: an array with the following fields:
%   -metrics: metrics to assess the system performance.
%   -state_of_the_system: vector of zeros and ones representing if the
%    system exhibitis some performance.
%   -previous_state: previous system state.
%   -current_failure_mode: current component failure mode. It has two
%    sub-fields:
%     -component: index to represent which component has failed
%     -component_failure_mode: index to represent the corresponding
%      failure mode.
% OUTPUTS:
% -system_states: see above.
% -evaluated_states_counter: see above.
% -state_of_the_system: column vector with zeros and ones. A one
%  represent the system exhibiting some performance.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
function [system_states,evaluated_states_counter,state_of_the_system]=...
    evaluation(model_name,signals,evaluated_states_counter,...
    num_of_components,system_states)
global stop_time;
global injection
global time;
global failure_modes;
state_of_the_system=[];
global level_number
options=[];

level_number=level_number+1;

for i=1:size(signals,1)
    time=zeros(1,num_of_components);
    time(signals(i,1))=injection*stop_time;
    failure_modes=signals(i,2:num_of_components+1);
    xinitial=system_states(signals(i, size(signals,2) - 1)).xfinal;
    %options=simset('InitialState',xinitial);
    [time_arr,x,yout]=sim(model_name,[0 stop_time],options);
    % Final conditions for this sequence of failures that will be passed
    %  to the next sequence of failures
    xfinal=x(size(x,1),:);
    [state_of_the_system_i,metrics]=...
        metrics_assessment(Performance_metrics,upper_bound,lower_bound);
    state_of_the_system= [state_of_the_system ; state_of_the_system_i];
    system_states(evaluated_states_counter + i).metrics=metrics';
    system_states(evaluated_states_counter + i).state_of_the_system=...
        state_of_the_system_i;
    system_states(evaluated_states_counter + i).previous_state=...
        signals(i, size(signals,2) - 1);
    system_states(evaluated_states_counter + i).xfinal=xfinal;
    current_failure_mode.component=signals(i,1);
    current_failure_mode.component_failure_mode=signals(i,signals(i)+1);

end
evaluated_states_counter=evaluated_states_counter+size(signals,1);
```

```
%%%%%%%%%%%%%%%%%%%%%%% AUXILIARY FUNCTION %%%%%%%%%%%%%%%%%%%%%%%%
function [state_of_the_system_i,metrics]=...
    metrics_assessment(Performance_metrics,upper_bound,lower_bound)

metrics=[];
% Metrics definition for each of the performance metrics.
for i=1:size(Performance_metrics,2)
    metrics(i)=norm(Performance_metrics(:,i),inf);
end

%%%%%%%%%%%%%%%%%%%%%% AUXILIARY FUNCTIONS %%%%%%%%%%%%%%%%%%%%%%%%%%%%%


if max((metrics(:)>upper_bound(:) | metrics(:)<lower_bound(:)))==1;
   state_of_the_system_i=0;
else
   state_of_the_system_i=1;
end

end
```

sequences_separation

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                       Alejandro D. Dominguez-Garcia
%                              aledan@MIT.EDU
%           Laboratory for Electromagnetic and Electronic Systems
%                  Massachusetts Institute of Technology
%
%                        Created:  July 5, 2005
%                        Modified: July 5, 2005
%
%
% This function will return the failed and non-failed sequences at each
% level of failure
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [non_failure_sequences,failure_sequences]=...
    sequences_separation(state_of_the_system,signals)
% Init of auxiliary variables
aux2=[];
aux4=[];
% Augmented matrix with the last row including the state of the system
signals_aug=[state_of_the_system signals];
% The number of columns and rows is needed for following calculations
[m,n]=size(signals_aug);
for i=1:m
     % we will compute the sequences of failures that are not
     % catastrophic
    if signals_aug(i,1)==1
        aux1=signals_aug(i,2:n);
        aux2=[aux2; aux1];
    else
        aux3=signals_aug(i,2:n);
        aux4=[aux4; aux3];
    end
end

non_failure_sequences=aux2;
failure_sequences=aux4;
```

sequences_k_level

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                       Alejandro D. Dominguez-Garcia
%                              aledan@MIT.EDU
%           Laboratory for Electromagnetic and Electronic Systems
%                  Massachusetts Institute of Technology
%
%                        Created:  October 20, 2005
%                        Modified: October 20, 2005
%
%
% This function will create the failure sequences to inject at the k
% level of failure.
%
% INPUTS:
% -non_failure_sequences: the sequences of failures corresponding to the
%  states at level k-1 that did not produce a system failure.
% -failure_modes_vector: vector of dimension the number of components
%  with associated failure information. Each elements has an integer
%  that represents the number of failure modes associated to the
%  component.
% -state_counter: it counts the number of states.
% -failure_level: it counts the levels of failure
% OUTPUTS:
% -signals: a matrix with as many rows as the sum of failure modes
%  of all the components and many columns as components plus 4. The
%  first column is an index to represent the failed component. the last
%  column represents the component that failed just before. The one
%  before the last and the one before the one before the last is the
%  state number.
% -state_counter: see above.
% -failure_level: see above.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [signals,state_counter,...
    failure_level_counter]=sequences_k_level(non_failure_sequences,...
    failure_modes_vector,state_counter,failure_level_counter)

% Init signals
```

```
signals=sparse([]);


% Increases the failure level counter
failure_level_counter=failure_level_counter+1;

for j=1:size(non_failure_sequences,1)
    single_sequence=non_failure_sequences(j,:);
    for i=2:size(failure_modes_vector,2)+1
        if single_sequence(i)==0;
            failure_modes_i=failure_modes_vector(i-1);
            new_sequences_aux_j=...
            [ones(failure_modes_i,1)*single_sequence];
            new_sequences_aux_j(:,i)=[1:1:failure_modes_i]';
            new_sequences_aux_j(:,1)=i-1;
            new_sequences_aux_j(:,size(failure_modes_vector,2)+3)=...
                single_sequence(size(failure_modes_vector,2)+2);
            new_sequences_aux_j(:,size(failure_modes_vector,2)+2)=...
            state_counter*ones(failure_modes_i,1)+[1:1:failure_modes_i]';
            new_sequences_aux_j(:,size(failure_modes_vector,2)+4)=...
                single_sequence(1);
            state_counter=state_counter+failure_modes_i;
            signals=[signals ; new_sequences_aux_j];
        end
    end
end
```

matrix

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                      Alejandro D. Dominguez-Garcia
%                             aledan@MIT.EDU
%           Laboratory for Electromagnetic and Electronic Systems
%                  Massachusetts Institute of Technology
%
%                         Created:  October 24, 2005
%                         Modified: April 19, 2005
%
%
% This script helps establishing the state-transition matrix.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


function [P sensitivity]=matrix(signals,failure_info,lambda,lambda_sym)

global sym_matrix;
global P;
global sensitivity;

for i=1:size(signals,1)
    if sym_matrix=='n'
        lambda_i=lambda(signals(i,1),signals(i,signals(i,1)+1));
    elseif sym_matrix=='y'
        lambda_i=lambda_sym(signals(i,1),signals(i,signals(i,1)+1));
    end
    P(signals(i,size(signals,2)-2),signals(i,size(signals,2)-1))=...
        lambda_i;
    aux=-lambda_i;
    P(signals(i,size(signals,2)-1),signals(i,size(signals,2)-1))=...
      P(signals(i,size(signals,2)-1),signals(i,size(signals,2)-1))+aux;
    % Sensitivity matrices
    k=signals(i,1);
    l=signals(i,signals(i,1)+1);
    m=signals(i,size(signals,2)-2);
    n=signals(i,size(signals,2)-1);
    sensitivity(k,l).matrix(m,n)=sparse(1);
    sensitivity(k,l).matrix(n,n)=sparse(-1);
end
```

truncate_matrix

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                       Alejandro D. Dominguez-Garcia
%                              aledan@MIT.EDU
%           Laboratory for Electromagnetic and Electronic Systems
%                  Massachusetts Institute of Technology
%
%                        Created:  February 10, 2006
%                        Modified: April 19, 2006
%
%
% This script truncates the state transition matrix at truncation level
% t_level after the system was evaluated at t_level-1.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [P,sensitivity]=truncate_matrix(evaluated_states_counter);

global P;
global sensitivity;

% Truncate transition matrix
P = [P(1:evaluated_states_counter, :) ; ...
    sum(P((evaluated_states_counter+1):size(P,1),:),1) ];

% Truncate sensitivity matrices
for i=1:size(sensitivity,1)
    for j=1:size(sensitivity,2)
    if size(sensitivity(i,j).matrix,1)>evaluated_states_counter
    sensitivity(i,j).matrix=...
    [sensitivity(i,j).matrix(1:evaluated_states_counter, :) ; ...
    sum(sensitivity(i,j).matrix(...
    evaluated_states_counter+1):size(sensitivity(i,j).matrix,1),:),1) ];
    end
    end
end
```

markov_model_solver

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                       Alejandro D. Dominguez-Garcia
%                              aledan@MIT.EDU
%           Laboratory for Electromagnetic and Electronic Systems
%                  Massachusetts Institute of Technology
%
%                        Created:  August 23, 2005
%                        Modified: April 19, 2006
%
% Script to solve the system of differential equations associated to the
% Markov model.
%
% INPUTS:
% -evaluation_time: time for which we want to analyze the system under
%  study information.
% OUTPUTS:
% -t: column vector with the times corresponding to each row of the
%  matrix p
% -p: matrix with as many columns as system states. The number of
%  columns is equal to the number of time steps taken by the solver to
%  solve the system.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%% NEW IMPLEMENTATION %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% This is a very fast algorithm that only works for Sparse numerical
% matrices. It has implemented the inline evaluation of matrices for
% time-dependent coefficients, but this is not compatible with sparse
% matrices, therefore, only sparse connstant matrices can be solved
function [t,p,p_states] = markov_model_solver(evaluation_time,lambda)
global P;
global A;
global sensitivity
global sym_matrix;
global Lambda_ij;
global sensitivity_factor;
global sens_analysis;
% Number of equations
[N,M]=size(P);
P=[P sparse(N,N-M)];
```

```
% Time span for simulation
tspan=[0,evaluation_time];
% Initial conditions
p_0=zeros(N,1);
p_0(1)=1;
% To check if the matrix is purely numerical. In this case, a faster
% evaluator will be used
syms t
if sym_matrix=='y'
    A=eval(P);
else
    A=P;
end
options=odeset('RelTol',1.e-4);
whos_lambda=whos('lambda');
if length(whos_lambda.class)==3
    A=inline(P);
    [t,p]=ode23(@jacobian_time,tspan,p_0,options);
else
    [t,p]=ode23(@jacobian_constant,tspan,p_0,options);
end
%%%%%%%%%% SENSITIVITY ANALYSIS WITH RESPECT TO EACH FAILURE RATE %%%%%%
if sens_analysis=='y';
    for i=1:size(sensitivity,1)
        for j=1:size(sensitivity,2)
            [a,b]=size(sensitivity(i,j).matrix);
            aux=sparse([sensitivity(i,j).matrix ; zeros(N-a,b)]);
            Delta_Lambda_ij=[aux zeros(N,N-b)];
            Lambda_ij=P+sensitivity_factor*lambda(i,j)*Delta_Lambda_ij;
            [t,p_s]=ode23(@jacobian_sensitivity,tspan,p_0,options);
            p_states(i,j).p=p_s;
        end
    end
else
     p_states=[];
end
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%% AUXILIARY FUNCTIONS %%%%%%%%%%%%%%%%%%%%%%%%%%
function dpdt=jacobian_time(t,p)
global A
dpdt=A(t)*p;
end
```

```
function dpdt=jacobian_constant(t,p)
global A
dpdt=A*p;
end
function dpdt=jacobian_sensitivity(t,p)
global Lambda_ij
dpdt=Lambda_ij*p;
end
```

metrics_models_evaluation

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                        Alejandro D. Dominguez-Garcia
%                               aledan@MIT.EDU
%           Laboratory for Electromagnetic and Electronic Systems
%                  Massachusetts Institute of Technology
%
%                         Created:  November 2, 2005
%                         Modified: March 21, 2006
%
%
% This function will analyze the results of the evaluation.
%
% INPUTS:
% -system_states: an array with the following fields:
%   -metrics: metrics to assess the system performance.
%   -state_of_the_system: vector of zeros and ones representing if the
%    system exhibitis some performance.
%   -previous_state: previous system state.
%   -current_failure_mode: current component failure mode. It has two
%    sub-fields:
%     -component: index to represent which component has failed.
%     -component_failure_mode: index to represent the corresponding
%      failure mode.
% -states_per_failure_level: row vector with as many elements as levels
%  per failure. Each element is an integer representing the number of
%  each states at that level.
% -t: column vector with the times corresponding to each row of the
%  matrix p
% -p: matrix with as many columns as system states. The number of
%  columns is equal to the number of time steps taken by the solver to
%  solve the system.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [non_functional_metrics_measures,...
    non_functional_metrics_measures_levels,...
    probabilistic_performance_measures]=...
    metrics_models_evaluation(system_states,states_per_failure_level,t,p)
level=[];
global truncation;
```

```
if isempty(system_states(size(system_states,2)).current_failure_mode)==0
    %%%%%%%%%%%%%%%%%%%%%%%% NON-FUNCTIONAL METRICS MEASURES %%%%%%%%%%%%%
    %%% NOMINAL VALUES (UNALTERED FAILURE RATES) %%%%%%%%
    % Reliability
    system_state=[system_states.state_of_the_system];
    state_probability=p(size(t,1),:);
    reliability_lower_bound=system_state*state_probability';
    reliability_upper_bound=reliability_lower_bound;
    non_functional_metrics_measures(1).name='System Reliability';
    non_functional_metrics_measures(1).value=...
        [reliability_lower_bound reliability_upper_bound];
    non_functional_metrics_measures(1).error=0;
    % Unreliability
    unreliability_lower_bound=...
        (ones(1,size(system_state,2))-system_state)*state_probability';
    unreliability_upper_bound=unreliability_lower_bound;
    non_functional_metrics_measures(2).name='System Unreliability';
    non_functional_metrics_measures(2).value=...
        [unreliability_lower_bound unreliability_upper_bound];
    non_functional_metrics_measures(2).error=0;
%%%%%%%%%%%%%%%%%%%% PROBABILISTIC PERFORMANCE METRICS %%%%%%%%%%%%%
    % Expected value model
    metric_1=[system_states.metrics];
    ev_acc=metric_1*(state_probability'*size(metric_1,1));
    probabilistic_performance_measures(1).name='Accuracy Expected Value';
    probabilistic_performance_measures(1).value=ev_acc;
    % Expected value of the deviation absolute value model
    dev_metric_1=...
        metric_1-[system_states(1).metrics  ...
        zeros(size(metric_1,1),size(metric_1,2)-1)];
    ev_acc_abs_val_dev=...
        abs(dev_metric_1)*(state_probability'*size(metric_1,1));
    probabilistic_performance_measures(2).name=...
        'Accuracy absolute value deviation expected value';
    probabilistic_performance_measures(2).value=ev_acc_abs_val_dev;
else
%%%%%%%%%%%%%%%%%%%%%%%% NON-FUNCTIONAL METRICS MEASURES %%%%%%%%%%%%%
    % Reliability
    system_state=[system_states.state_of_the_system];
    state_probability=p(size(t,1),:);
    reliability_lower_bound=system_state*state_probability';
    reliability_upper_bound=...
```

```
        system_state*state_probability'+...
        state_probability(size(state_probability,2));
    non_functional_metrics_measures(1).name='System Reliability';
    non_functional_metrics_measures(1).value=[reliability_lower_bound ...
        reliability_upper_bound];
    non_functional_metrics_measures(1).error=[state_probability(size...
        state_probability,2))];
    % Unreliability
    unreliability_lower_bound=...
        (ones(1,size(system_state,2)-1)-system_state(1,1:(...
        size(system_state,2)-1)))*state_probability(1,1:(...
        size(system_state,2)-1))';
    unreliability_upper_bound=(ones(...
        1,size(system_state,2))-system_state)*state_probability';
    non_functional_metrics_measures(2).name='System Unreliability';
    non_functional_metrics_measures(2).value=...
        [unreliability_lower_bound unreliability_upper_bound];
    non_functional_metrics_measures(2).error=...
        [state_probability(size(state_probability,2))];
%%%%%%%%%%%%%%%%%%%%% PROBABILISTIC PERFORMANCE METRICS %%%%%%%%%%%%
    % This is the new probabilistic performance measure in which only the
    % non-failed states are considered.
    % Expected value model
    metric_1=...
        [system_states(1,1:size(system_states,2)-1).metrics zeros(...
        size(system_states(1).metrics,1),1)];
    ev_acc=...
    metric_1*((1/reliability_lower_bound)*system_state.*state_probability)';
    probabilistic_performance_measures(1).name='Accuracy Expected Value';
    probabilistic_performance_measures(1).value=ev_acc;
    % Expected value of the deviation absolute value model
    dev_metric_1=metric_1-[system_states(1).metrics ...
        zeros(size(metric_1,1),size(metric_1,2)-1)];
    ev_acc_abs_val_dev=...
        abs(dev_metric_1)*((1/reliability_lower_bound)*...
        system_state.*state_probability)';
    probabilistic_performance_measures(2).name=...
        'Accuracy absolute value deviation expected value';
    probabilistic_performance_measures(2).value=ev_acc_abs_val_dev;
end
for i=2:size(states_per_failure_level,1)
    initial_state_level_i=sum(states_per_failure_level(1:i-1)) + 1;
    final_state_level_i=sum(states_per_failure_level(1:i));
    reliability_i=...
        system_state(initial_state_level_i:final_state_level_i)*...
        state_probability(initial_state_level_i:final_state_level_i)';
    unreliability_i...
        =(ones(1,final_state_level_i-initial_state_level_i+1)-...
        system_state(initial_state_level_i:final_state_level_i))*...
        state_probability(initial_state_level_i:final_state_level_i)';
    level_i=[reliability_i unreliability_i];
    level=[level ; level_i];
end
non_functional_metrics_measures_levels=level;
```

sensitivity_analysis

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                       Alejandro D. Dominguez-Garcia
%                              aledan@MIT.EDU
%           Laboratory for Electromagnetic and Electronic Systems
%                  Massachusetts Institute of Technology
%
%                        Created:  April 19, 2006
%                        Modified: April 19, 2006
%
%
% This function will carry out the sensitivy analysis
%  system.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [sensitivity_measures sensitivity_driver driver]=...
    sensitivity_analysis(system_states,t,p_states,components_information)
sensitivity_measures.driver=[];
sensitivity_measures.value_lower_bound=[];
sensitivity_measures.value_upper_bound=[];
sensitivity_driver=[];
if isempty(system_states(size(system_states,2)).current_failure_mode)==0
%%%%%%%%%%%%%%%%%%%%%% NON-FUNCTIONAL METRICS MEASURES %%%%%%%%%%%%%
    system_state=[system_states.state_of_the_system];
    for i=1:size(p_states,1)
        for j=1:size(p_states(i),2)
            state_probability=p_states(i,j).p(size(t,1),:);
                        unreliability_ij=...
           (ones(1,size(system_state,2))-system_state)*state_probability';
            sensitivity_measures(i,j).driver=...
                {[cell2mat(components_information.name(i)) ...
                cell2mat(...
                components_information.failure_info(i).failure_modes(j))]};
            sensitivity_measures(i,j).value_lower_bound=unreliability_ij;
            sensitivity_measures(i,j).value_upper_bound=unreliability_ij;
            sensitivity_driver(i,j)=[unreliability_ij];
       end
    end
    [driver_value_i driver_index_i]=max(sensitivity_driver');
    [driver_value_j driver_index_j]=max(driver_value_i);
    driver={[cell2mat(components_information.name(driver_index_i)) ...
cell2mat(components_information.failure_info(...
driver_index_i).failure_modes(driver_index_j))]};
else
%%%%%%%%%%%%%%%%%%%%%% NON-FUNCTIONAL METRICS MEASURES %%%%%%%%%%%%%
    % Reliability
    system_state=[system_states.state_of_the_system];
    for i=1:size(p_states,1)
        for j=1:size(p_states(i),2)
            state_probability=p_states(i,j).p(size(t,1),:);
            unreliability_ij_lower_bound=...
(ones(1,size(system_state,2)-1)-...
system_state(1,1:(size(system_state,2)-1)))*state_probability(1,1:(...
size(system_state,2)-1))';
            unreliability_ij_upper_bound=...
(ones(1,size(system_state,2))-system_state)*state_probability';
sensitivity_measures(i,j).driver={[cell2mat(...
components_information.name(i)) ...
cell2mat(components_information.failure_info(i).failure_modes(j))]};
sensitivity_measures(i,j).value_lower_bound=unreliability_ij_lower_bound;
sensitivity_measures(i,j).value_upper_bound=unreliability_ij_upper_bound;
sensitivity_driver(i,j)=[unreliability_ij_upper_bound;];
        end
    end
[driver_value_i driver_index_i]=max(sensitivity_driver');
[driver_value_j driver_index_j]=max(driver_value_i);
driver={[cell2mat(components_information.name(driver_index_i)) ...
cell2mat(...
components_information.failure_info(...
driver_index_i).failure_modes(driver_index_j))]};
end
```

results_analysis

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                     Alejandro D. Dominguez-Garcia
%                            aledan@MIT.EDU
%           Laboratory for Electromagnetic and Electronic Systems
%                  Massachusetts Institute of Technology
%
%                      Created:  October 26, 2005
%                      Modified: February 08, 2006
%
% This function will analyze the results of the evaluation.
%
% INPUTS:
% -components_information: an array variable with the following fields:
%   -name: name of the component with associated failure information
%   -failure_info: an array variable with the the following fields:
%      -failure_modes: component failure modes
%      -failure_rates: failure rates associated to each failure mode
%       listed in the failure_modes field.
% -system_states: an array with the following fields:
%   -metrics: metrics to assess the system performance.
%   -state_of_the_system: vector of zeros and ones representing if the
%    system exhibitis some performance.
%   -previous_state: previous system state.
%   -current_failure_mode: current component failure mode. It has two
%    sub-fields:
%     -component: index to represent which component has failed.
%     -component_failure_mode: index to represent the corresponding
%      failure mode.
% -states_per_failure_level: row vector with as many elements as levels
%  per
%  failure. Each element is an integer representing the number of each
%  states at that level.
% -t: column vector with the times corresponding to each row of the
%  matrix p
% -p: matrix with as many columns as system states. The number of columns
%  is equal to the number of time steps taken by the solver to solve the
%  system.
% -filename: name of the *.xls file where the analysis will be stored.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
function results_analysis(components_information,system_states,...
    states_per_failure_level,t,p,non_functional_metrics_measures,...
    non_functional_metrics_measures_levels,...
    probabilistic_performance_measures,...
    sensitivity_measures,driver,filename)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% MAIN ANALYSIS %%%%%%%%%%%%%%%%%%%%%%%%%%%
% Init variables
sequence=[];
failure_level_info=[];
metrics=[];
non_functional_metrics_info=[];
probabilistic_performance_measures_info=[];
probabilistic_performance_measures_value=[];
drivers=[];
value_lower_bound=[];
value_upper_bound=[];
global sens_analysis;
% Copy the template analysis file into the file entered for outputing the
% results
copyfile('results_analysis.xls', filename,'f');
% Initial state, system with no failures
state_probability_0=p(size(t,1),1);
metrics_1=system_states(1).metrics;
failure_level_info = [state_probability_0 ; metrics_1];
xlswrite(filename, failure_level_info, ...
['Failure Level 0 and Metrics'], 'B3');
failure_level_info=[];
% Subsequent states, system with one, two, ..., n failures
for i=2:size(states_per_failure_level,1)-1
    % Initial state label at level i
    initial_state_level_i=sum(states_per_failure_level(1:i-1)) + 1;
     % Final state label at level i
    final_state_level_i=sum(states_per_failure_level(1:i));
    for j=initial_state_level_i:final_state_level_i
    current_failure_mode=system_states(j).current_failure_mode;
    % Sequence of failures at the first level
    if i==2
    sequence(j).aux={[cell2mat(components_information.name(...
    current_failure_mode.component)) cell2mat(...
    components_information.failure_info(...
    current_failure_mode.component).failure_modes(...
    current_failure_mode.component_failure_mode))]};
    % Sequence of failures after the first level
    else
    sequence(j).aux=...
    {[cell2mat(sequence(system_states(j).previous_state).aux) ...
    '->' cell2mat(components_information.name(...
    current_failure_mode.component)) ...
    cell2mat(components_information.failure_info(...
    current_failure_mode.component).failure_modes(...
    current_failure_mode.component_failure_mode))]};
    end
    % Sequence of failures yielding to the current state,
    % including the current failure mode
    sequence_j=cell2mat(sequence(j).aux);
    % Failure rate for the last failure mode
    failure_rate_i=char((components_information.failure_info(...
    current_failure_mode.component).failure_rates(...
    current_failure_mode.component_failure_mode)));
    % State number
    state_index=j;
    % State probability at the end of the evaluation_time
    state_probability_j=p(size(t,1),j);
    % System metrics for each state
    metrics_j=system_states(j).metrics';
    % State of the system (failed or operational)
    state_of_the_system=system_states(j).state_of_the_system;
    failure_level_info = ...
     [failure_level_info ; {sequence_j, failure_rate_i, ...
      state_index, state_probability_j, ...
      state_of_the_system}];
      metrics=[metrics; metrics_j];
    end
    % Failure level information
    xlswrite(filename, failure_level_info, ...
    ['Failure Level ' num2str(i-1)], 'A2');
    % Performance metrics
    xlswrite(filename, metrics, ['Failure Level ' num2str(i-1)], 'F2');
    failure_level_info=[];
    metrics=[];
end
%This is to write the last level of failure%%%
    i=size(states_per_failure_level,1);
    % Initial state label at level i
```

```
initial_state_level_i=sum(states_per_failure_level(1:i-1)) + 1;
% Final state label at level i
final_state_level_i=sum(states_per_failure_level(1:i));
% The system has more than one component
if  initial_state_level_i<=final_state_level_i
if isempty(system_states(...
initial_state_level_i).current_failure_mode)==0
for j=initial_state_level_i:final_state_level_i
current_failure_mode=system_states(j).current_failure_mode;
% The system is not single string
if initial_state_level_i>2
sequence(j).aux=...
{[cell2mat(sequence(system_states(j).previous_state).aux) '->' ...
cell2mat(components_information.name(...
current_failure_mode.component)) ...
cell2mat(components_information.failure_info(...
current_failure_mode.component).failure_modes(...
current_failure_mode.component_failure_mode))]};
else
% All the failure modes are at the first level
sequence(j).aux=...
{[cell2mat(components_information.name(...
current_failure_mode.component)) ...
cell2mat(components_information.failure_info(...
current_failure_mode.component).failure_modes(...
current_failure_mode.component_failure_mode))]};
end
% Sequence of failures yielding to the current state,
% including the current failure mode
sequence_j=cell2mat(sequence(j).aux);
% Failure rate for the last failure mode
failure_rate_i=char((components_information.failure_info(...
current_failure_mode.component).failure_rates(...
current_failure_mode.component_failure_mode)));
% State number
state_index=j;
% State probability at the end of the evaluation_time
state_probability_j=p(size(t,1),j);
% System metrics for each state
metrics_j=system_states(j).metrics';
% State of the system (failed or operational)
state_of_the_system=system_states(j).state_of_the_system;
```

```
    failure_level_info = [failure_level_info ; ...
    {sequence_j, failure_rate_i, ...
    state_index, state_probability_j, state_of_the_system}];
    metrics=[metrics; metrics_j];
    end
    % Failure level information
    xlswrite(filename, failure_level_info,....
     ['Failure Level ' num2str(i-1)], 'A2');
    % Performance metrics
    xlswrite(filename, metrics, ['Failure Level ' num2str(i-1)], 'F2');
    failure_level_info=[];
    else
    state_index=initial_state_level_i;
    state_probability_j=p(size(t,1),initial_state_level_i);
    failure_level_info = [failure_level_info ; {'ABSORBING STATE', 'N/A', ...
    state_index, state_probability_j, 'N/A', 'N/A'}];
    % Failure level information
    xlswrite(...
    filename, failure_level_info, ['Failure Level ' num2str(i-1)], 'A2');
    failure_level_info=[];
    metrics=[];
    end
    end
state_probability_0=p(size(t,1),1);
metrics=system_states(1).metrics;
failure_level_info = [state_probability_0 ; metrics];
xlswrite(filename, failure_level_info, ['Failure Level 0 and Metrics'], 'B3');
failure_level_info=[];
%%%%%% NON-FUNCTIONAL METRICS RESULTS %%%%%%%%%%%%%%%%%%%
for j=1:size(non_functional_metrics_measures,2)
    non_functional_metrics_info=[non_functional_metrics_info ; ...
[non_functional_metrics_measures(j).value non_functional_metrics_measures(j...
).error]];
end
    xlswrite(filename,...
        non_functional_metrics_info, 'Failure Level 0 and Metrics','E7');
    xlswrite(filename, ...
        non_functional_metrics_measures_levels, ...
        'Failure Level 0 and Metrics','E12');
for j=1:size(probabilistic_performance_measures,2)
    probabilistic_performance_measures_info=...
[probabilistic_performance_measures_info ; ...
{probabilistic_performance_measures(j).name}];
    probabilistic_performance_measures_value=...
[probabilistic_performance_measures_value ; ...
probabilistic_performance_measures(j).value'];
end
    xlswrite(filename, probabilistic_performance_measures_info, ...
        'Failure Level 0 and Metrics','D2');
    xlswrite(filename, probabilistic_performance_measures_value, ...
        'Failure Level 0 and Metrics','E2');
%%%%%%%%%%%%%%%%% SENSITIVITY ANALYSIS RESULTS %%%%%%%%%%%
if sens_analysis=='y'
    for j=1:size(sensitivity_measures,1)
    drivers=[drivers ; {cell2mat(sensitivity_measures(j).driver)}];
    value_lower_bound=...
    [value_lower_bound ; sensitivity_measures(j).value_lower_bound];
    value_upper_bound=...
    [value_upper_bound ; sensitivity_measures(j).value_upper_bound];
    end
    xlswrite(filename, drivers, 'Failure Level 0 and Metrics','A25');
    xlswrite(filename, value_lower_bound, ...
        'Failure Level 0 and Metrics','B25');
    xlswrite(filename, value_upper_bound, ...
        'Failure Level 0 and Metrics','C25');
    xlswrite(filename, driver, 'Failure Level 0 and Metrics','E25');
end
% Open xls file with report
winopen(filename);
```

Inpresto_plot

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                        Alejandro D. Dominguez-Garcia
%                               aledan@MIT.EDU
%            Laboratory for Electromagnetic and Electronic Systems
%                   Massachusetts Institute of Technology
%
%                         Created:  February 09, 2005
%                         Modified: May 23, 2006
%
%
% This function will plot some of the the results of the evaluation.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function Inpresto_plot

% We need to load the results in the workspace in order to be able to
% plot any dynamic behavior
load model_results

global stop_time;
global time;
global injection
global failure_modes;

plotting='y';

while plotting=='y'
    state=input('Introduce the state number (spreadsheet column C): ');
    failure_modes=zeros(1,num_of_components);
    time=zeros(1,num_of_components);
    current_failure_mode=system_states(state).current_failure_mode;
    [time_arr,x,yout]=sim(model_name,[0 stop_time]);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % This is just to get an steady-state picture of the system response
    xinitial=x(size(x,1),:);
    options=simset('InitialState',xinitial);
    [time_arr,x,yout]=sim(model_name,[0 stop_time],options);
    y_1=[time_arr yout];
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    if system_states(state).previous_state ~=0
        time(current_failure_mode.component)=injection*stop_time;
        previous=system_states(state).previous_state;
        previous_failure_mode=system_states(state).current_failure_mode;
        while previous~=0
            failure_modes(previous_failure_mode.component)=...
                previous_failure_mode.component_failure_mode;
            previous_failure_mode=...
                system_states(previous).current_failure_mode;
            previous=system_states(previous).previous_state;
        end
        xinitial=...
            system_states(system_states(state).previous_state).xfinal;
        options=simset('InitialState',xinitial);
        [time_arr,x,yout]=sim(model_name,[0 stop_time],options);
        y_k=[time_arr,yout];
    else
        [time_arr,x,yout]=sim(model_name,[0 stop_time],options);
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        y_k=[time_arr,yout];
    end
%%%%%%%%%%%%%%%%%%%%%% FcS case study Chapter 5 %%%%%%%%%%%%%%%%%%%%%%%
figure(state);
plot(y_k(:,1),y_k(:,2),'-','LineWidth',1); hold
plot(y_k(:,1),y_k(:,3),'g--','LineWidth',1)
plot(y_k(:,1),y_k(:,4),'r:','LineWidth',1)
set(gca,'fontsize',16,'fontname','times new roman');
xlabel('Time[s]','fontsize',16,'fontname','times new roman');
ylabel('\phi_c [rad], \phi_r [rad], \phi [rad]','fontsize',16,...
    'fontname','times new roman');
legend('Roll command \phi_c','Reference model response \phi_r',...
    'Aircraft response \phi')
axis([0 stop_time -0.25 0.25]);
hold off

%%%%%%%%%%%%%%%%%%%%%% SbW case study Chapter 6 %%%%%%%%%%%%%%%%%%%%%%
figure(state);
subplot(2,1,1)
plot(y_k(:,1),y_k(:,2),'b-','LineWidth',1); hold
plot(y_k(:,1),y_k(:,3),'r-','LineWidth',1)
```

```
set(gca,'fontsize',14,'fontname','times new roman');
xlabel('Time [s]','fontsize',16,'fontname','times new roman');
ylabel('\delta_w [deg], SR\delta [deg]','fontsize',16,...
    'fontname','times new roman');
legend('Steering wheel angle \delta_w',...
    'Scaled road wheel angle SR\delta')
axis([0 stop_time -20.5 13.5]);
subplot (2,1,2)
plot(y_k(:,1),y_k(:,4),'b-','LineWidth',1); hold
plot(y_k(:,1),y_k(:,5),'r-','LineWidth',1);
set(gca,'fontsize',14,'fontname','times new roman');
xlabel('Time [s]','fontsize',16,'fontname','times new roman');
ylabel('\Psi_r [deg], \Psi [deg]','fontsize',16,...
    'fontname','times new roman');
legend('Conventional vehicle response \Psi_r', 'SbW vehicle response \Psi')
axis([0 stop_time -0.85 1.5]);
hold off
plotting=...
input('Do you want to plot some of the performance solutions? (y/n): ','s');
end
```

Appendix C

Component Behavioral Models for the Flight Control System Case-Study

C.1 Primary Flight Computer

C.1.1 Voter

$$\begin{aligned} \epsilon_1 &= |u_1 - u_2| \\ \epsilon_2 &= |u_1 - u_3| \\ \epsilon_3 &= |u_2 - u_3| \end{aligned}$$

$$\tilde{u} = \begin{cases} \sum_{i=1}^{3} u_i - \max_i\{u_i\} - \min_i\{u_i\}, & \text{if } \exists\, i,j = 1,2,3 \;/\; \epsilon_i + \epsilon_j < 2\epsilon \\ \dfrac{u_i + u_j}{2}, & \text{if } \exists\, i = 1,2,3 \;/\; \epsilon_i < \epsilon \\ NIL, & otherwise \end{cases} \tag{C.1}$$

where $\epsilon = 0.1$rad.

C.1.2 Roll Control Law

$$\begin{aligned} R_r(s) &= K_{r_1}\phi_c(s) + K_{r_2}R_b(s) + K_{r_3}\tfrac{s+z_r}{s+p_r}P_b(s) \\ \delta^{l(r)}_{a^*_r}(s) &= (P_r + \tfrac{I_r}{s} + D_r s)(R_r(s) \pm K_r\delta^{l(r)}_a(s)) \end{aligned}$$

$$\delta_{a_r}^{l(r)}(t) = \begin{cases} \delta_{a_r^*}^{l(r)}(t), & \text{for } r = 0 \\ 2\delta_{a_r^*}^{l(r)}(t), & \text{for } r = 1 \end{cases}$$

$$\hat{\delta}_{a_r}^{l(r)}(t) = \begin{cases} \delta_{a_r}^{l(r)}(t), & \text{for } U_f = 0 \\ 0, & \text{for } U_f = 1 \\ rand(l, u), & \text{for } U_f = 2 \\ \delta_{a_r}^{l(r)}(\tau), & \text{for } U_f = 3 \\ \sigma_{\tau_l}(\delta_{a_r}^{l(r)}(t)), & \text{for } U_f = 4 \end{cases} \qquad \text{(C.2)}$$

where $K_{r_1} = 0.66$, $K_{r_2} = -0.145\text{s}$, $K_{r_3} = 2.16\text{s}$, $z_r = 11.1\text{s}^{-1}$, $p_r = 25\text{s}^{-1}$, $P_r = 0.45\text{A}$, $I_r = 6\text{A/s}$, $D_r = 0.01\text{As}$, and $K_r = -1.33$ were taken from [41]; and $l = -5\text{A}$, $u = 5\text{A}$ $\tau_l = 0.2\text{s}$, and τ is the time at which the computer gets stuck.

C.1.3 Yaw Control Law

$$R_y(s) = K_{y_1}\delta_c(s) + \tfrac{s^2+z_{y_1}s+z_{y_2}}{s(s+p_y)}(K_{y_2}R_b(s) + K_{y_3}P_b(s))$$
$$\delta_{r_r^*}(s) = (P_y + \tfrac{I_y}{s} + D_y s)(R_y(s) + K_y\delta_r(s))$$

$$\delta_{r_r}(t) = \begin{cases} \delta_{r_r^*}(t), & \text{for } r = 0 \\ 2\delta_{r_r^*}(t), & \text{for } r = 1 \end{cases}$$

$$\hat{\delta}_{r_r}(t) = \begin{cases} \delta_{r_r}(t), & \text{for } U_f = 0 \\ 0, & \text{for } U_f = 1 \\ rand(l, u), & \text{for } U_f = 2 \\ \delta_{r_r}(\tau), & \text{for } U_f = 3 \\ \sigma_{\tau_l}(\delta_{r_r}(t)), & \text{for } U_f = 4 \end{cases} \qquad \text{(C.3)}$$

where $K_{y_1} = 0.001$, $K_{y_2} = -0.7816\text{s}$, $K_{y_3} = 0.172\text{s}$, $z_{y_1} = -0.00125\text{s}^{-2}$, $z_{y_2} = -0.001875\text{s}^{-1}$, $p_y = 1.5\text{s}^{-1}$, $P_y = 0.45\text{A}$, $I_y = 6\text{A/s}$, $D_y = 0.01\text{As}$, $K_y = -1.33$, were taken from [41]; and $l = -5\text{A}$, $u = 5\text{A}$ $\tau_l = 0.2\text{s}$, and τ is the time at which the computer gets stuck.

C.2 Actuation Subsystem

C.2.1 Ailerons

$$G(s) = k_1 \frac{(s+\frac{1}{\tau_c})(s+\frac{1}{\tau_m})}{(s+\frac{1}{\tau_1})(s+\frac{1}{\tau_2})(s+\frac{1}{\tau_3})}$$
$$T^{l(r)}_{a_{1(2)}}(s) = k_2 \frac{G(s)}{1+G(s)} \hat{\delta}^{l(r)}_{a_r}(s)$$

$$\hat{T}^{l(r)}_{a_{1(2)}}(t) = \begin{cases} T^{l(r)}_{a_{1(2)}}(t), & \text{for } U_f = 0 \\ 0, & \text{for } U_f = 1 \\ T^{l(r)}_{a_{1(2)}}(\tau), & \text{for } U_f = 2 \end{cases} \tag{C.4}$$

where $k_1 = 5000\text{s}^2$, $k_2 = 50$, $\tau_c = 10\text{s}$, $\tau_m = 10\text{s}$, $\tau_1 = 4\text{s}$, $\tau_2 = 10\text{s}$, $\tau_3 = 100\text{s}$, and τ is the time at which the actuation subsystem gets stuck.

C.2.2 Rudder

$$G(s) = k_1 \frac{(s+\frac{1}{\tau_c})(s+\frac{1}{\tau_m})}{(s+\frac{1}{\tau_1})(s+\frac{1}{\tau_2})(s+\frac{1}{\tau_3})}$$
$$T_{r_{1(2)}}(s) = k_2 \frac{G(s)}{1+G(s)} \hat{\delta}_r(s)$$

$$\hat{T}_{r_{1(2)}}(t) = \begin{cases} T_{r_{1(2)}}, & \text{for } U_f = 0 \\ 0, & \text{for } U_f = 1 \\ T_{r_{1(2)}}(\tau), & \text{for } U_f = 2 \end{cases} \tag{C.5}$$

where $k_1 = 5000\text{s}^2$, $k_2 = 50$, $\tau_c = 10\text{s}$, $\tau_m = 10\text{s}$, $\tau_1 = 4\text{s}$, $\tau_2 = 10\text{s}$, $\tau_3 = 100\text{s}$, and τ is the time at which the actuation subsystem gets stuck.

C.3 Mechanical Combiner

C.3.1 Ailerons

$$T_a^{l,r} = G_a(\hat{T}_{a_1}^{l(r)} + \hat{T}_{a_2}^{l(r)}) \tag{C.6}$$

where $G_a = 0.5$.

C.3.2 Rudder

$$T_r = G_r(\hat{T}_{r_1} + \hat{T}_{r_2}) \tag{C.7}$$

where $G_r = 0.5$.

C.4 Control Surfaces

C.4.1 Ailerons

$$\begin{aligned} \dot{x}_a^{l,r} &= -\tfrac{1}{\tau_a}x_a^{l,r} + \tfrac{1}{\tau_a}T_a^{l,r} \\ \delta_a^{l(r)} &= \sigma_{\tau_l}(x_a) \end{aligned}$$

$$\hat{\delta}_a^{l(r)} = \begin{cases} \delta_a^{l(r)}, & \text{for } U_f = 0 \\ \delta_a(\tau), & \text{for } U_f = 1 \\ \alpha_0, & \text{for } U_f = 2 \end{cases} \tag{C.8}$$

where $\tau_a = 0.04$s and $\tau_l = 0.11$s, $\alpha_0 = 0.216$rad, were taken from [41]. τ is the time at which the aileron gets stuck.

C.4.2 Rudder

$$\begin{aligned} \dot{x}_r &= -\tfrac{1}{\tau_r}x_r + \tfrac{1}{\tau_r}T_r \\ \delta_r &= \sigma_{\tau_l}(x_r) \end{aligned}$$

$$\hat{\delta}_r = \begin{cases} \delta_r, & \text{for } U_f = 0 \\ \delta_r(\tau), & \text{for } U_f = 1 \\ \Psi, & \text{for } U_f = 2 \end{cases} \tag{C.9}$$

where $\tau_r = 0.05$s and $\tau_l = 0.1$s were taken from [41]. τ is the time at which the rudder gets stuck.

C.5 Inertial Measurement Unit

$$\begin{aligned} x_{imu}(t + \Delta t) &= x_{imu}(t) + randn(0, \sigma_{ss}\sqrt{1 - e^{-2\Delta t/\tau}}) \\ x_{imu}(0) &= \sigma_{ss} \end{aligned}$$

$$\begin{aligned} y_{imu}(t + \Delta t) &= x_{imu}(t + \Delta t) + \\ [M_1 + M_2 + I] & \begin{bmatrix} p_b \\ q_b \\ r_b \end{bmatrix} + \begin{bmatrix} randn(\mu_b, \sigma_b) \\ randn(\mu_b, \sigma_b) \\ randn(\mu_b, \sigma_b) \end{bmatrix} \\ M_1 = K_1 & \begin{bmatrix} randn(0,1) & 0 & 0 \\ 0 & randn(0,1) & 0 \\ 0 & 0 & randn(0,1) \end{bmatrix} \\ M_2 = K_2 & \begin{bmatrix} 0 & 0 & 0 \\ -randn(0,1) & 0 & 0 \\ randn(0,1) & 0 & -randn(0,1) \end{bmatrix} \end{aligned}$$

$$\hat{y}_{imu} = \begin{cases} y_{imu}, & \text{for } U_f = 0 \\ 0, & \text{for } U_f = 1 \\ G y_{imu}, & \text{for } U_f = 2 \\ y_{imu} + B, & \text{for } U_f = 3 \end{cases} \tag{C.10}$$

where $\sigma_{ss} = 3.15 \cdot 10^{-6}$, $\Delta t = 0.01$s, $\tau = 100$s, $\mu_b = 4.85 \cdot 10^{-6} rand(0,1)$, $\sigma_b = 2.09 \cdot 10^{-5}$, $K_1 = 10^{-4}$, $K_2 = 9.7 \cdot 10^{-5}$, $G = 1.5$, and $B = 0.3$deg.

C.6 Control Surfaces Position Sensor

C.6.1 Ailerons

$$\begin{aligned} \dot{x}_{s_a}^{l(r)} &= -\frac{1}{\tau_s} x_{s_a}^{l(r)} + \frac{1}{\tau_s} \delta_a^{l(r)} \\ y_{s_a}^{l(r)} &= \frac{\lceil \sigma_{\tau_l}(x_{s_a}^{l(r)}) \frac{1}{R} \rceil}{\frac{1}{R}} \end{aligned}$$

$$\hat{y}_{s_a}^{l(r)} = \begin{cases} y_{s_a}^{l(r)}, & \text{for } U_f = 0 \\ 0, & \text{for } U_f = 1 \\ G y_{s_a}^{l(r)}, & \text{for } U_f = 2 \\ y_{s_a}^{l(r)} + B, & \text{for } U_f = 3 \end{cases} \tag{C.11}$$

where $\tau_s = 0.01$s, $\tau_l = 0.001$s, $R = 0.001$, $G = 1.5$, and $B = 0.3$deg.

C.6.2 Rudder

$$\begin{aligned} \dot{x}_{s_r} &= -\frac{1}{\tau_s} x_{s_r} + \frac{1}{\tau_s} \delta_r \\ y_{s_r} &= \frac{\lceil \sigma_{\tau_l}(x_{s_r}) \frac{1}{R} \rceil}{\frac{1}{R}} \end{aligned}$$

$$\hat{y}_{s_r} = \begin{cases} y_{s_r}, & \text{for } U_f = 0 \\ 0, & \text{for } U_f = 1 \\ Gy_{s_r}, & \text{for } U_f = 2 \\ y_{s_r} + B, & \text{for } U_f = 3 \end{cases} \tag{C.12}$$

where $\tau_s = 0.01\text{s}$, $\tau_l = 0.001\text{s}$, $R = 0.001$, $G = 1.5$, and $B = 0.3\text{deg}$.

C.7 Linear Lateral-Directional Aircraft Dynamics

$$\begin{bmatrix} \dot{\beta} \\ \dot{p}_b \\ \dot{r}_b \\ \dot{\phi} \end{bmatrix} = \begin{bmatrix} \frac{Y_\beta}{V} & \sin\alpha_0 & -\cos\alpha_0 & \frac{g\cos\alpha_0}{V} \\ L_\beta & L_p & L_r & 0 \\ N_\beta & N_p & N_r & 0 \\ 0 & 1 & \tan\alpha_0 & 0 \end{bmatrix} \begin{bmatrix} \beta \\ p_b \\ r_b \\ \phi \end{bmatrix} + \begin{bmatrix} 0 & 0 & Y_{\delta_r} \\ N_{\delta_a^l} & N_{\delta_a^r} & N_{\delta_r} \\ L_{\delta_a^l} & L_{\delta_a^r} & L_{\delta_r} \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \delta_a^l \\ \delta_a^r \\ \delta_r \end{bmatrix} \tag{C.13}$$

where $Y_\beta = -50.69\text{m/s}^2$, $V = 178\text{m/s}$, $\alpha_0 = 0.216\text{rad}$, $g = 9.81\text{m/s}^2$, $L_\beta = -32.3\text{s}^{-2}$, $L_p = -0.374\text{s}^{-1}$ $L_r = 2.40\text{s}^{-1}$, $N_\beta = 1.06\text{s}^{-2}$, $N_p = -0.0406\text{s}^{-1}$, $N_r = -0.0809\text{s}^{-1}$, $Y_{\delta_r} = 0.0179\text{s}^{-1}$, $N_{\delta_a^l} = -3.175\text{s}^{-2}$, $N_{\delta_a^r} = 3.175\text{s}^{-2}$, $N_{\delta_r} = 6.66\text{s}^{-2}$, $L_{\delta_a^l} = -0.855\text{s}^{-2}$, $L_{\delta_a^r} = 0.855\text{s}^{-2}$, and $L_{\delta_r} = -1.18\text{s}^{-2}$, were taken from [67].

Notation Used in this Appendix

B :	Bias factor for IMUs and control surface position sensors
g :	Acceleration of gravity
G :	Gain change factor for IMUs and control surface position sensors
G_a :	Left and right mechanical combiner gain
G_r :	Rudder mechanical combiner gain
L_β :	Dimensional variation of rolling moment about X_s with β
L_p :	Dimensional variation of rolling moment about X_s with p
L_r :	Dimensional variation of rolling moment about X_s with r
$L_{\delta_a^l}$:	Dimensional variation of rolling moment about X_s with δ_a^l

$L_{\delta_a^r}$:	Dimensional variation of rolling moment about X_s with δ_a^r
L_{δ_r} :	Dimensional variation of rolling moment about X_s with δ_r
N_β :	Dimensional variation of yawing moment about Z_s with β
N_p :	Dimensional variation of yawing moment about Z_s with p
N_r :	Dimensional variation of yawing moment about Z_s with r
$N_{\delta_a^l}$:	Dimensional variation of yawing moment about Z_s with δ_a^l
$N_{\delta_a^r}$:	Dimensional variation of yawing moment about Z_s with δ_a^r
N_{δ_r} :	Dimensional variation of yawing moment about Z_s with δ_r
$p_b, P_b(s)$:	Roll rate
r :	Primary Flight Computers reconfiguration signals
R :	Control surface position sensors resolution
$r_b, R_b(s)$:	Yaw rate
s :	Laplace transform variable
t :	Time
$T_a^{l(r)}$:	Left (right) aileron torque command
$T_{a_{1(2)}}^{l(r)}, \hat{T}_{a_{1(2)}}^{l(r)}$:	Left (right) aileron actuation subsystem torque output
T_r :	Rudder torque command
$T_{r_{1(2)}}, \hat{T}_{r_{1(2)}}$:	Rudder actuation subsystem torque output
u_1, u_2, u_3 :	Voter inputs
$\tilde{u}$:	Voter output
U_b :	Component failure model switch control input
V :	Forward velocity
$x_a^{l,r}$:	Left (right) aileron state variable
x_{imu} :	Inertial measurement unit state variables
x_r :	Rudder state variable
$x_{s_a}^{l(r)}$:	Left (right) aileron position sensor state variable
x_{s_r} :	Rudder position sensor state variable
$y_{imu}, \hat{y}_{imu}$:	Inertial measurement output
$y_{s_a}^{l(r)}, \hat{y}_{s_a}^{l(r)}$:	Left (right) aileron position sensor output
$y_{s_r}, \hat{y}_{s_r}$:	Rudder position sensor output
Y_β :	Dimensional variation of Y_s-force with β
Y_{δ_r} :	Dimensional variation of Y_s-force with δ_r
α_0 :	Pitch angle
β :	Sideslip angle
δ_c :	Yaw command
$\delta_a^{l(r)}, \hat{\delta}_a^{l(r)}$:	Left (right) aileron angle
$\delta_r, \hat{\delta}_r$:	Rudder angle
$\delta_{a_r^*}^{l(r)}, \delta_{a_r}^{l(r)}, \hat{\delta}_{a_r}^{l(r)}$:	Roll control output
$\delta_{r_r^*}, \delta_{r_r}, \hat{\delta}_{r_r}$:	Yaw control output
ϵ :	Voter tolerance error
ϕ :	Roll angle
ϕ_c :	Roll command
$\sigma_{\tau_l}(\cdot)$:	τ_l-shift operator
Ψ :	Heading angle
τ_a:	Inverse of aileron bandwidth
τ_r:	Inverse of rudder bandwidth
τ_s :	Inverse of left, right, and rudder position sensors bandwidth
$\lceil \cdot \rceil$:	Ceiling function
$rand(l, u)$:	Uniformly distributed random generator between l and u
$randn(\mu, \sigma)$:	Gaussian distributed random generator with mean μ and standard deviation σ
$k_1, k_2, \tau_1, \tau_2, \tau_3,$ τ_c, τ_m :	Left, right, and rudder actuation subsystem parameters

$K_1, K_2, \Delta t, \tau, \mu_b,$	
σ_{ss}, σ_b :	IMU parameters
$K_{r_1}, K_{r_2}, K_{r_3},$	
$z_r, p_r, P_r, I_r,$	
D_r, K_r :	Roll control law parameters
$K_{y_1}, K_{y_2}, K_{y_3},$	
$z_{y_1}, p_{y_2}, p_y, P_y,$	
I_y, D_y, K_y :	Yaw control law parameters

Appendix D

Component Behavioral Models for the SbW/BAS Case-Study

D.1 Steer-by-wire Computer

D.1.1 Voter

$$\begin{aligned}\epsilon_1 &= |u_1 - u_2|\\ \epsilon_2 &= |u_1 - u_3|\\ \epsilon_3 &= |u_2 - u_3|\end{aligned}$$

$$\tilde{u} = \begin{cases} \sum_{i=1}^{3} u_i - \max_i\{u_i\} - \min_i\{u_i\}, & \text{if } \exists\, i,j = 1,2,3 \;/\; \epsilon_i + \epsilon_j < 2\epsilon \\ \dfrac{u_i + u_j}{2}, & \text{if } \exists\, i = 1,2,3 \;/\; \epsilon_i < \epsilon \\ NIL, & otherwise \end{cases} \tag{D.1}$$

where $\epsilon = 0.1$rad.

D.1.2 Steering Rack Position Controller for the lock-step processors architecture

$$T_{r_r^*}(s) = (P_1 + \tfrac{I_1}{s} + D_1 s)(y_{\delta_w} - y_\delta)$$

$$T_{r_r}(t) = \begin{cases} T_{r_r^*}(t), & \text{for } r = 0 \\ 2T_{r_r^*}(t), & \text{for } r = 1 \end{cases}$$

$$\hat{T}_{r_r}(t) = \begin{cases} T_{r_r}(t), & \text{for } U_f = 0 \\ 0, & \text{for } U_f = 1 \\ rand(l,u), & \text{for } U_f = 2 \\ T_{r_r}(\tau), & \text{for } U_f = 3 \\ \sigma_{\mathcal{T}_l}(T_{r_r}(t)), & \text{for } U_f = 4 \end{cases} \tag{D.2}$$

where $P_1 = 60\text{A}$, $I_1 = 7\text{A/s}$, $D_1 = 15\text{As}$, $l = -5\text{A}$, $u = 5\text{A}$ $\mathcal{T}_l = 0.2\text{s}$, and τ is the time at which the computer gets stuck.

D.1.3 Steering Rack Position Controller for the SbW/BAS architecture

$$T_{r_r^*}(s) = 2(P_2 + \tfrac{I_2}{s} + D_2 s)(y_{\delta_w} - k_1 y_\delta)$$

$$T_{r_r}(t) = \begin{cases} T_{r_r^*}(t), & \text{for } r = 0 \\ 2T_{r_r^*}(t), & \text{for } r = 1 \end{cases}$$

$$\hat{T}_{r_r}(t) = \begin{cases} T_{r_r}(t), & \text{for } U_f = 0 \\ 0, & \text{for } U_f = 1 \\ rand(l,u), & \text{for } U_f = 2 \\ T_{r_r}(\tau), & \text{for } U_f = 3 \\ \sigma_{\mathcal{T}_l}(T_{r_r}(t)), & \text{for } U_f = 4 \end{cases} \tag{D.3}$$

where $P_2 = 15\text{A}$, $I_2 = 15\text{A/s}$, $D_2 = 3\text{As}$, $k_1 = 1.23$, $l = -5\text{A}$, $u = 5\text{A}$ $\mathcal{T}_l = 0.2\text{s}$, and τ is the time at which the computer gets stuck.

D.2 Brake-Actuated-Steering Computer

D.2.1 Voter

$$\epsilon_1 = |u_1 - u_2|$$
$$\epsilon_2 = |u_1 - u_3|$$
$$\epsilon_3 = |u_2 - u_3|$$

$$\tilde{u} = \begin{cases} \sum_{i=1}^{3} u_i - \max_i\{u_i\} - \min_i\{u_i\}, & \text{if } \exists\, i, j = 1,2,3 \,/\, \epsilon_i + \epsilon_j < 2\epsilon \\ \dfrac{u_i + u_j}{2}, & \text{if } \exists\, i = 1,2,3 \,/\, \epsilon_i < \epsilon \\ NIL, & otherwise \end{cases} \tag{D.4}$$

where $\epsilon = 0.1$rad.

D.2.2 Longitudinal Force Tires Controller

$$F_{lfl_r^*}(s) = -F_{lfr_r^*}(s) = \tfrac{1}{S}(P_2 + \tfrac{I_2}{s} + D_2 s)(y_{\delta_w} - k_\delta y_2)$$

$$F_{lfl(r)_r}(t) = \begin{cases} 0, & \text{for } r = 0 \\ F_{lfl(r)_r^*}(t), & \text{for } r = 1 \end{cases}$$

$$\hat{F}_{lfl(r)_r}(t) = \begin{cases} F_{lfl(r)_r}(t), & \text{for } U_f = 0 \\ 0, & \text{for } U_f = 1 \\ rand(l, u), & \text{for } U_f = 2 \\ F_{lfl(r)_r}(\tau), & \text{for } U_f = 3 \\ \sigma_{T_l}(F_{lfl(r)_r}(t)), & \text{for } U_f = 4 \end{cases} \tag{D.5}$$

where $P_3 = 60$A, $I_3 = 10$A/s, $D_3 = 7$As, $k_2 = 1.07$, $l = -5$A, $u = 5$A $T_l = 0.2$s, and τ is the time at which the computer gets stuck.

D.3 Steering Rack Actuation Subsystem

$$G(s) = k_1^r \frac{(s+\frac{1}{\tau_c^r})(s+\frac{1}{\tau_m^r})}{(s+\frac{1}{\tau_1^r})(s+\frac{1}{\tau_2^r})(s+\frac{1}{\tau_3^r})}$$

$$T_{r_{1(2)}}(s) = k_2^r \frac{G(s)}{1+G(s)} \hat{\delta}_r(s)$$

$$\hat{T}_{r_{1(2)}}(t) = \begin{cases} T_{r_{1(2)}}, & \text{for } U_f = 0 \\ 0, & \text{for } U_f = 1 \\ T_{r_{1(2)}}(\tau), & \text{for } U_f = 2 \end{cases} \tag{D.6}$$

where $k_1^r = 5000\text{s}^2$, $k_2^r = 1$ for the lock-step processor architecture, and $k_2^r = 11.3$ for the SbW/BAS architecure, $\tau_c^r = 10\text{s}$, $\tau_m^r = 10\text{s}$, $\tau_1^r = 4\text{s}$, $\tau_2^r = 10\text{s}$, $\tau_3^r = 100\text{s}$, and τ is the time at which the actuation subsystem gets stuck.

D.4 Mechanical Combiner

$$T_r = \frac{SR}{10}(\hat{T}_{r_1} + \hat{T}_{r_2}) \tag{D.7}$$

where $SR = 17$.

D.5 Caliper Actuation Subsystem

$$G(s) = k_1^c \frac{(s+\frac{1}{\tau_c^c})(s+\frac{1}{\tau_m^c})}{(s+\frac{1}{\tau_1^c})(s+\frac{1}{\tau_2^c})}$$

$$F_{lfl(r)}(s) = k_2^c \frac{G(s)}{1+G(s)} \hat{F}_{lfl(r)_r}(s)$$

$$\hat{F}_{lfl(r)}(t) = \begin{cases} F_{lfl(r)}, & \text{for } U_f = 0 \\ 0, & \text{for } U_f = 1 \\ F_{lfl(r)}(\tau), & \text{for } U_f = 2 \end{cases} \tag{D.8}$$

where $k_1^c = 0.5\text{s}^2$, $k_2^c = 0.1$, $\tau_c^c = 10\text{s}$, $\tau_m^c = 10\text{s}$, $\tau_1^c = 4\text{s}$, $\tau_2^c = 0.1\text{s}$, and τ is the time at which the actuation subsystem gets stuck.

D.6 Road Wheel Angle Sensor

$$\begin{aligned} \dot{x}_\delta &= -\tfrac{1}{\tau_s} x_\delta + \tfrac{1}{\tau_s}\delta \\ y_\delta &= G_s \frac{\lceil \sigma_{\tau_l}(x_\delta)\frac{1}{R} \rceil}{\frac{1}{R}} \end{aligned}$$

$$\hat{y}_\delta = \begin{cases} y_\delta, & \text{for } U_f = 0 \\ 0, & \text{for } U_f = 1 \\ G y_\delta, & \text{for } U_f = 2 \\ y_\delta + B, & \text{for } U_f = 3 \end{cases} \tag{D.9}$$

where $\tau_s = 0.01\text{s}$, $\tau_l = 0.001\text{s}$, $R = 0.001$, $G_s = 17.85$, $G = 1.5$, and $B = 0.3\text{deg}$.

D.7 Steering Wheel Angle Sensor

$$\begin{aligned} \dot{x}_{\delta_w} &= -\tfrac{1}{\tau_s} x_{\delta_w} + \tfrac{1}{\tau_s}\delta_w \\ y_{\delta_w} &= \frac{\lceil \sigma_{\tau_l}(x_{\delta_w})\frac{1}{R} \rceil}{\frac{1}{R}} \end{aligned}$$

$$\hat{y}_{\delta_w} = \begin{cases} y_{\delta_w}, & \text{for } U_f = 0 \\ 0, & \text{for } U_f = 1 \\ G y_{\delta_w}, & \text{for } U_f = 2 \\ y_{\delta_w} + B, & \text{for } U_f = 3 \end{cases} \tag{D.10}$$

where $\tau_s = 0.01\text{s}$, $\tau_l = 0.001\text{s}$, $R = 0.001$, $G = 1.5$, and $B = 0.3\text{deg}$.

D.8 Linear Single-Track Vehicle Dynamics

$$\begin{bmatrix} \dot{\beta} \\ \dot{r}_b \\ \dot{\Psi} \end{bmatrix} = \begin{bmatrix} \frac{C_f+C_r}{mV} & \frac{aC_f-bC_r}{mV^2}-1 & 0 \\ \frac{aC_f-bC_r}{J} & \frac{a^2C_f+b^2C_r}{VJ} & 0 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} \beta \\ r_b \\ \Psi \end{bmatrix} + \begin{bmatrix} \frac{-C_f}{mV} \\ \frac{-aC_f}{J} \\ 0 \end{bmatrix} \delta$$

$$F_{sfl} = F_{sfr} = \tfrac{C_f}{2}(\beta + \tfrac{ar}{V} - \delta) \qquad \text{(D.11)}$$

where $a = 1.046\text{m}$, $b = 1.712\text{m}$, $C_f = -1090\text{N/deg}$, $m = 1741.6$ Kg, $J = 3007$ Kgm^2, and $V = 70$ km/h, correspond to a medium-size Ford® vehicle.

D.9 Linear Two-Track Vehicle Dynamics

$$\begin{bmatrix} \dot{\beta} \\ \dot{r}_b \\ \dot{\Psi} \end{bmatrix} = \begin{bmatrix} \frac{C_f+C_r}{mV} & \frac{aC_f-bC_r}{mV^2}-1 & 0 \\ \frac{aC_f-bC_r}{J} & \frac{a^2C_f+b^2C_r}{VJ} & 0 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} \beta \\ r_b \\ \Psi \end{bmatrix} + \begin{bmatrix} \frac{-C_f}{mV} \\ \frac{-aC_f}{J} \\ 0 \end{bmatrix} \delta$$

$$+ \begin{bmatrix} 0 & 0 \\ \frac{c}{2J} & -\frac{c}{2J} \\ 0 & 0 \end{bmatrix} \begin{bmatrix} \Delta F_{lf} \\ \Delta F_{lr} \end{bmatrix}$$

$$F_{lfl} + F_{lfr} + F_{lrl} + F_{lrr} = 0$$

$$F_{sfl} = F_{sfr} = \tfrac{C_f}{2}(\beta + \tfrac{ar}{V} - \delta)$$

$$F_{srl} = F_{srr} = \tfrac{C_r}{2}(\beta - \tfrac{br}{V}) \qquad \text{(D.12)}$$

where $a = 1.046\text{m}$, $b = 1.712\text{m}$, $C_f = -1090\text{N/deg}$, $C_r = -1090\text{N/deg}$, $m = 1741.6$ Kg, $J = 3007$ Kgm^2, and $V = 70$ km/h, correspond to a medium-size Ford® vehicle.

D.10 Rack

$$
\begin{aligned}
J_c\ddot{\delta} + B_c\dot{\delta} = S(\hat{F}_{lfl} - \hat{F}_{lfr}) + T(F_{sfl} + F_{sfr}) + T_r,\ for\ -4\pi < \delta < 4\pi \\
\dot{\delta} = 0,\ for\ \delta = 4\pi \\
\dot{\delta} = 0,\ for\ \delta = -4\pi
\end{aligned}
\tag{D.13}
$$

where $J_c = 10\text{Kgm}^2$, $B_c = 0.1\text{Nms}$, $S = -0.02\text{m}$, and $T = 0.025\text{m}$.

Notation Used in this Appendix

a :	Distance from center of gravity to front axle
b :	Distance from center of gravity to rear axle
B :	Bias factor for road wheel angle and steering wheel angle sensors
B_c :	Steering rack and actuation motors combined viscous coefficient
C_f :	Front axle equivalent cornering stiffness
C_r :	Rear axle equivalent cornering stiffness
G :	Gain change factor for road wheel angle and steering wheel angle sensors
$F_{lfl}, F_{lfr}, \hat{F}_{lfl}, \hat{F}_{lfr}$:	Longitudinal forces acting on the front left and right tire
F_{sfl}, F_{sfr} :	Side forces acting on the front left and right tire
$F_{lfl^*_r}, F_{lfl_r}, \hat{F}_{lfl_r}, F_{lfr^*_r}, F_{lfr_r}, \hat{F}_{lfr_r}$:	Longitudinal force tires controller output
G_s :	Road wheel angle sensors gain
J :	Yaw inertia
J_c :	Steering rack and actuation motors combined inertia
$k_1^r, k_2^r, \tau_1^r, \tau_2^r, \tau_3^r, \tau_c^r, \tau_m^r$:	Steering rack actuation subsystem parameters
$k_1^c, k_2^c, \tau_1^c, \tau_2^c, \tau_3^c, \tau_c^c, \tau_m^c$:	Caliper actuation subsystem parameters
m :	Mass
P_1, I_1, D_1 :	Steering rack position controller parameters for the lock-step processors architecture
P_2, I_2, D_2, k_1 :	Steering rack position controller parameters for the SbW/BAS architecture
P_3, I_3, D_3, k_3 :	Longitudinal force tires controller parameters for the SbW/BAS architecture
r :	Primary Flight Computers reconfiguration signals
R :	Control surface position sensors resolution
r_b :	Yaw rate
s :	Laplace transform variable
S :	Scrub radius
SR :	Steering ratio
t :	Time
R :	Mechanical trail
T_r :	Steering rack torque command
$T_{r_{1(2)}}, \hat{T}_{r_{1(2)}}$:	Steering rack actuation subsystem torque output
$T_{r^*_r}, T_{r_r}, \hat{T}_{r_r}$:	Steering rack position controller output
u_1, u_2, u_3 :	Voter inputs

$\tilde{u}$:	Voter output
V :	Center of gravity speed
x_δ :	Road wheel angle sensor sensor state variable
x_{δ_w} :	Steering wheel angle sensor sensor state variable
$y_\delta, \hat{y}_\delta$:	Road wheel angle sensor sensor output
$y_{\delta_w}, \hat{y}_\delta$:	Steering wheel angle sensor sensor output
β :	Sideslip angle
δ :	Road wheel angle
δ_w :	Steering wheel angle
ϵ :	Voter tolerance error
Ψ :	Heading angle
$\sigma_{\tau_l}(\cdot)$:	τ_l-shift operator
τ_s :	Inverse of road wheel angle and steering wheel angle sensors bandwidth
$\lceil \cdot \rceil$:	Ceiling function
$rand(l, u)$:	Uniformly distributed random generator between l and u

Bibliography

[1] R. Hammett and P. Babcock, "Achieving 10^{-9} dependability with drive-by-wire systems," *SAE Technical Paper Series*, no. 2003-01-1290, 2003.

[2] M. Abele, "Modellierung und bewertung von fehlertoleranzmassnahmen in kfz-energiebordnetzen fr sicherheitsrelevante verbraucher," Master's thesis, Unikassel Versitat, Kassel, Germany, 2004.

[3] E. Gai and M. Adams, "Measures of merit for fault-tolerant systems," The Charles Stark Draper Laboratory, Cambridge, MA, Tech. Rep. CSDL-P-1752, 1983.

[4] J. Laprie, Ed., *Dependability: Basic Concepts and Terminology.* New York, NY: Springer-Verlag, 1991.

[5] A. Avižienis, "Design of fault-tolerant computers," in *Proceedings of the Fall Joint Computer Conference, AFIPS Conference,* Washington, DC, 1967, pp. 733–743.

[6] A. Høyland and M. Rausand, *System Reliability Theory.* New York, NY: John Wiley and Sons, 1994.

[7] Y. Papadopoulos, J. McDermid, R. Sasse, and G. Heiner, "Analysis and synthesis of the behavior of complex programmable electronic systems in conditions of failure," *Journal of Reliability Engineering and System Safety*, vol. 71, no. 3, pp. 229–247, Mar. 2001.

[8] Y. Papadopoulos, D. Parker, and C. Grante, "Model-based automated synthesis of fault trees from matlab-simulink models," in *Proceedings of the International Conference on Dependable Systems and Networks*, Gothenburg, Sweden, 2001, pp. 77–82.

[9] N. Leveson, *Safeware: System Safety and Computers.* Boston, MA: Addison-Wesley Publishing Company, 1995.

[10] *IEEE Standard Dictionary of Electrical and Electronics Terms*, IEEE Std. 100-1996, 1996.

[11] E. S. Agency. (2002) New Soyuz TMA spacecraft cleared for next mission with ESA astronaut. [Online]. Available: http://www.esa.int/esaCP/Pr_13_2003_i_EN.html

[12] E. Moore and C. Shannon, "Reliable circuits using less reliable relays, parts I & II," *Journal of the Franklin Institute*, vol. 262, pp. 191–208 and 281–297, Sept./Oct. 1956.

[13] J. Taylor, "An algorithm for fault-tree construction," *IEEE Transactions on Reliability*, vol. 31, no. 2, pp. 219–254, June 1982.

[14] J. Fussel, E. Aber, and R. Rahl, "On the quantitative analysis of priorty-and failure logic," *IEEE Transactions on Reliability*, vol. 31, no. 2, pp. 219–254, June 1982.

[15] J. Dugan, S. Bavuso, and M. Boyd, "Fault trees and sequence dependencies," in *Proceedings of the Annual Reliability and Maintainability symposium*, Los Angeles, CA, 1990.

[16] ——, "Dynamic fault fault-tree models for fault-tolerant computer systems," *IEEE Transactions on Reliability*, vol. 41, no. 3, pp. 363–377, 1992.

[17] R. Manian, J. Dugan, D. Coppit, and K. Sullivan, "Combining various solution techniques for dynamic fault tree analysis of computer systems," in *Proceedings of the Third IEEE International High-Assurance Systems Engineering Symposium*, Washington, DC, 1998.

[18] Y. Papadopoulos, D. Parker, and C. Grante, "A method and tool support for model-based semi-automated failure modes and effect analysis of engineering designs," in *Proceedings of the 9th Australian Workshop on Safety Critical Systems*, Brisbane, Australia, 2004.

[19] J. Taylor, "An integrated approach to the treatment of design and specification errors in eletronic systems and software," in *Proceedings of the Fifth European Conference on Electrotechnics*, Copenhagen, Denmark, 1982.

[20] Y. Papadopoulos, D. Parker, M. Walker, U. Pertersen, R. Hamann, and Q. Wu, "Automated failure modes and effects analysis of systems on-board ship," in *Proceedings of the International Conference on Marine Research and Transportation*, Ischia, Italy, 2005.

[21] P. Babcock, "An introduction to reliability modeling of fault-tolerant systems," The Charles Stark Draper Laboratory, Cambridge, MA, Tech. Rep. CSDL-R-1899, 1987.

[22] A. Reibman and M. Veeraraghavan, "Reliability modeling: An overview for system designers," *IEEE Computer*, vol. 24, no. 4, pp. 49–57, Apr. 1991.

[23] G. Grimmett and D. Stirzaker, *Probability and Random Processes*, 3rd ed. Oxford, UK: Oxford University Press, 2001.

[24] J. Kemeny and J. Snell, *Finite Markov Chains*. New York, NY: Springer-Verlag, 1983.

[25] M. Shooman, *Probabilisitic Reliability: An Engineering Approach*. Malabar, FL: Robert E. Krieger Publishing Company, 1990.

[26] J. Dugan, M. Veeraraghavan, M. Boyd, and N. Mittal, "Proceedings of the eighth symposium on reliable distributed systems," in *Proceedings of the Third IEEE International High-Assurance Systems Engineering Symposium*, Seattle, WA, 1989.

[27] P. Babcock, G. Rosch, and J. Zinchuk, "An automated environment for optimizing fault-tolerant systems design," in *Proceedings of the Reliability and Maintainability Symposium*, Orlando, FL, 1991, pp. 360–367.

[28] M. Hutchins, P. Babcock, and G. Rosch, "An introduction to the came program via example," The Charles Stark Draper Laboratory, Cambridge, MA, Tech. Rep. CSDL-R-2082, 1987.

[29] M. Malhotra and K. Trivedi, "Power-hierarchy of dependability model types," *IEEE Transactions on Reliability*, vol. 43, no. 3, pp. 493–502, 1994.

[30] R. Sahner, K. Trivedi, and A. Puliafito, *Performance and Reliability Analysis of Computer Systems*. Boston, MA: Kluwer Academic Publishers, 1996.

[31] "Reactor safety study," U.S. Nuclear Regulatory Commission, Washington, DC, Tech. Rep. NUREG 75/014 (WASH-1400), 1975.

[32] K. Fleming and F. Silady, "A risk informed defense-in-depth framework for existing and advanced reactors," *Journal of Reliability Engineering and System Safety*, vol. 78, no. 3, pp. 205–225, December 2002.

[33] A. Amendola, "Event sequences and consequence spectrum: A methodology for probabilistic transient analysis," *Nuclear Science An Engineering*, vol. 77, no. 3, pp. 297–315, 1981.

[34] T. Aldemir, "Computer-assisted markov failure modeling of process control systems," *IEEE Transactions on Reliability*, vol. R-36, no. 1, pp. 133–144, Apr. 1987.

[35] J. Devooght and C. Smidts, "Probabilistic reactor dynamics-I: The theory of continuos event trees," *Nuclear Science and Engineering*, vol. 111.

[36] P. Labeau, C. Smidts, and S. Swaminathan, "Dynamic reliability: Towards an integrated platform for probabilistic risk assessment," *Journal of Reliability Engineering and System Safety*, vol. 68, no. 3, pp. 219–254, June 2000.

[37] J. Devooght and C. Smidts, "Probabilistic dynamics as a tool for dynamic psa," *Reliability Engineering and System Safety*, vol. 52, no. 3, pp. 185–196, June 1996.

[38] C. Smidts and J. Devooght, "Probabilistic reactor dynamics-II: A monte carlo study of a fast reactor trasient," *Nuclear Science and Engineering*, vol. 111.

[39] Y. Hu, "A guided simluation methodology for dynamic probabilistic risk assessment of complex systems," Ph.D. dissertation, University of Maryland, College Park, MD, 2005.

[40] A. Dominguez-Garcia, J. Kassakian, and J. Schindall, "Reliability evaluation of the power supply of an electrical power net for safety-relevant applications," *Journal of Reliability Engineering and System Safety*, vol. 91, no. 5, pp. 505–514, May 2006.

[41] D. McRuer, T. Myers, and P. Thompson, "Literal singular-value-based flight control system design techniques," *AIAA Journal of Guidance*, vol. 12, no. 6, pp. 913–919, 1989.

[42] A. Dominguez-Garcia, J. Kassakian, J. Schindall, and J. Zinchuk, "On the use of behavioral models for the integrated performance and reliability evaluation of fault-tolerant avionics systems," in *Proceedings of DASC 2006*, Portland, OR, 2005.

[43] ——, "A backup system for automotive steer-by-wire, actuated by selective braking," in *Proceedings of PESC 2004*, Aachen, Germany, 2004.

[44] W. Harter, W. Pfeiffer, P. Dominke, G. Ruck, and P. Blessing, "Future electrical steering systems: Realizations with safety requirements," *SAE Technical Paper Series*, no. 2000-01-0822, 2000.

[45] R. Isermann, R. Schwarz, and S. Stolzl, "Fault-tolerant drive-by-wire systems," *IEEE Control Systems Magazine*, vol. 22, no. 5, pp. 64–81, Oct. 2002.

[46] P. Dominke and G. Ruck, "Electric power steering,the first step on the way to steer-by-wire," *SAE Technical Paper Series*, no. 1999-01-0401, 1999.

[47] K. Afridi, R. Tabors, and J. Kassakian, "Alternative electrical distribution system architectures for automobiles," in *Proceedings of Power Electronics in Transportation*, Dearborn, MI, 1994, pp. 33–38.

[48] H. Brinkmeyer, "Architecture of vehicle power supply in the throes of change," in *Proceedings of Automobile Electronics Congres*, Stuttgart, Germany, 2002.

[49] P. Babcock and J. Zinchuk, "Fault-tolerant design optimization: Application to an autonomous underwater vehicle navigation system," in *Proceedings of the Symposium on Autonomous Underwater Vehicle Technology*, Washington, DC, 1990, pp. 34–43.

[50] F. Schweppe, *Uncertain Dynamic Systems.* Englewood Cliffs, NJ: Prentice-Hall Inc., 1973.

[51] F. Blanchini, "Set invariance in control," *Automatica*, vol. 35, pp. 1747–1767, 1999.

[52] K. Trivedi, M. Malhotra, and R. Fricks, "Markov reward approach to performability and reliability analysis," in *Proceedings of the Second International Workshop on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems*, Durham, NC, 1994, pp. 7–11.

[53] A. Reibman, "Modeling the effect of reliability on performance," *IEEE Transactions on Reliability*, vol. 39, no. 3, pp. 314–320, Aug. 1990.

[54] J. Meyer, "On evaluating the performability of degradable computing system," *IEEE Transactions on Computers*, vol. C-29, no. 8, pp. 720–731, Aug. 1980.

[55] D. McRuer, I. Ashekenas, and D. Graham, *Aircraft Dynamics and Automatic Control.* Princeton, NJ: Princeton University Press, 1973.

[56] J. Roskan, *Flight Dynamics of Rigid and Elastic Airplanes.* Lawrence, KS: The University of Kansas, 1972.

[57] T. Tucker, "Touchdown: the development of propulsion controlled aircrafts at nasa dreyden," NASA, Monograph in aerospace history no. 16, 1999.

[58] W. Milliken and D. Milliken, *Race car vehicle dynamics.* Warrendale, PA: Society of Automotive Engineers, 1995.

[59] M. Cheok, G. Parry, and R. Sherry, "Use of importance measures in risk-informed regulatory applications," *Journal of Reliability Engineering and System Safety*, vol. 60, no. 3, pp. 213–226, June 1997.

[60] E. Borgonovo and G. Apostolakis, "A new importance measure for risk-informed decision making," *Journal of Reliability Engineering and System Safety*, vol. 72, no. 2, pp. 193–212, May 2001.

[61] E. Borgonovo, G. Apostolakis, S. Tarantola, and A. Saltelli, "Comparison of global sensitivity analysis techniques and importance measures in psa," *Journal of Reliability Engineering and System Safety*, vol. 79, no. 2, pp. 175–185, February 2003.

[62] J. Blake, A. Reibman, and K. Trivedi, "Sensitivity analysis of reliability and performability measures for multiprocessor systems," in *Proceedings of ACM Sigmetrics*, Santa Fe, NM, 1988.

[63] A. Gandini, "Importance and sensitivity analysis in assessing system reliability," *IEEE Transactions on Reliability*, vol. 39, no. 1, pp. 61–70, April 1990.

[64] E. Borgonovo, G. Apostolakis, S. Tarantola, and A. Saltelli, "Approximate sensitivity analysis for acyclic markov reliability models," *IEEE Transactions on Reliability*, vol. 52, no. 2, pp. 175–185, June 2003.

[65] R. Barlow and F. Proschan, *Mathematical Theory of Reliability.* New York, NY: John Wiley and Sons, 1965.

[66] *U.S. Federal Air Regulations 25.1309 and the supporting advisory circular* AC-25. *1309.*

[67] D. McRuer and T. Myers, "Advanced piloted aircraft flight control system design methodology volume I: Knowledge base," NASA, Langley, VA, Contractor Report 181726, 1988.